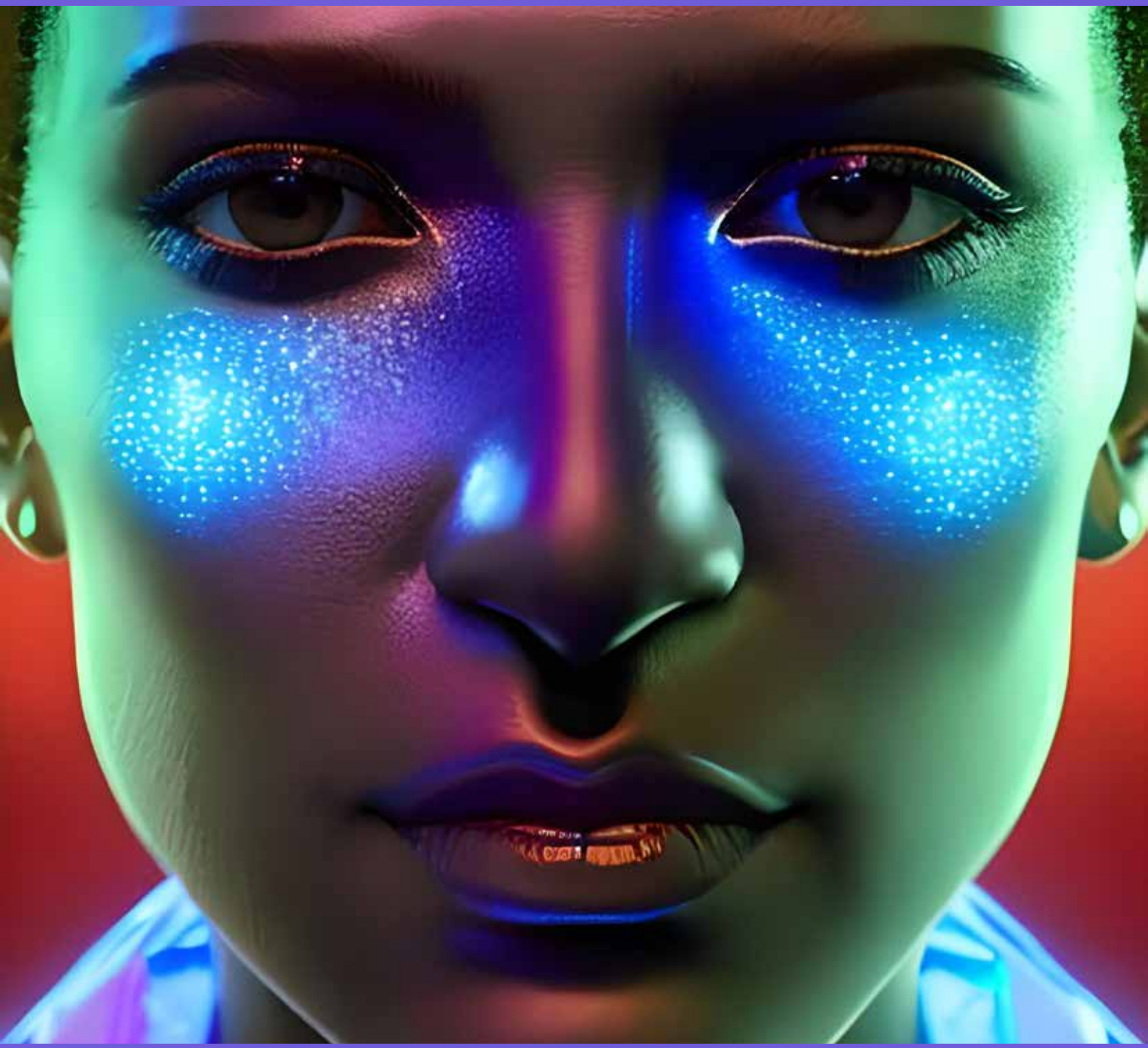
Detecção e reconhecimento facial: uma abordagem prática

Dilermando Piva Junior • Maria Rafaela Junqueira Bruno Rodrigues
André Santos Alckmin Carvalho • Ed Carlos da Silva Pereira
Erick José da Silva • Francisco de Assis de Freitas



FoxTablet

Detecção e reconhecimento facial: uma abordagem prática

Dilermando Piva Junior
Maria Rafaela Junqueira Bruno Rodrigues
André Santos Alckmin Carvalho
Ed Carlos da Silva Pereira
Erick José da Silva
Francisco de Assis de Freitas

Diagramação e publicação
Jean Pluvinage / Editora FoxTablet

ISBN: 978-65-89010-78-4

Ficha catalográfica

Dados Internacionais de Catalogação na Publicação (CIP)
(eDOC BRASIL, Belo Horizonte/MG)

D479 Detecção e reconhecimento facial [livro eletrônico] : uma abordagem prática / Dilermando Piva Junior... [et al.]. – Salto, SP: FoxTablet, 2023.
321 p. : il.

Formato: PDF
Requisitos de sistema: Adobe Acrobat Reader
Modo de acesso: World Wide Web
Inclui bibliografia
ISBN 978-65-89010-78-4

1. Reconhecimento facial (Computação). 2. Sistemas de segurança. 3. Identificação biométrica. I. Piva Junior, Dilermando. II. Rodrigues, Maria Rafaela Junqueira Bruno. III. Carvalho, André Santos Alckmin. IV. Pereira, Ed Carlos da Silva. V. Silva, Erick José da. VI. Freitas, Francisco de Assis de.

CDD 005.8

Elaborado por Maurício Amormino Júnior – CRB6/2422

FOXTABLET: A EDITORA COMPLETA!
Livros, revistas, jornais, ebooks e emagazines
www.foxtablet.com.br
contato@foxtablet.com.br
(11) 98689-1789

Agradecemos as nossas famílias, que sempre nos apoiaram, incondicionalmente.

Esta obra é dedicada aos nossos mestres, que semearam em nossas mentes o desejo incansável pela busca contínua do conhecimento.

Sobre os autores

Dilermando Piva Junior

Doutor em Engenharia de Computação pela Universidade Estadual de Campinas (Unicamp-SP) na área de Automação (Inteligência Artificial e Ensino a Distância). Atualmente é professor e Coordenador do Curso de Análise e Desenvolvimento de Sistemas da Fatec Itu, unidade do Centro Paula Souza. Idealizador do projeto pyPRO (*pypro.com.br* e *youtube.com.br/@pypro_br*). Foi coordenador de Educação a Distância do Centro Paula Souza para o Ensino Superior (Fatecs) e membro do conselho curador da Univesp (Universidade Virtual do Estado de São Paulo). Tem várias publicações nacionais e internacionais na área, com destaque para os livros "Algoritmos e Linguagem de Programação", "Estrutura de Dados e Técnicas de Programação", "Organização Básica de Computadores e Linguagem de Montagem", "Trabalhando de Casa: o home office passado a limpo", "Metodologias Ativas e Personalizadas de Aprendizagem", "Sala de Aula Digital" e "EaD na prática". É avaliador do Ministério da Educação e do Conselho Estadual de Educação de São Paulo e membro do Comitê Científico da Associação Brasileira de Educação a Distância (ABED). ***Link para o Lattes:***

http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4731701P6

Maria Rafaela Junqueira Bruno Rodrigues

Graduação em Direito pela Faculdade de Direito de Franca(1991), Especialização em Metodologia do Ensino Superior(2001), Mestrado em Direito pela Universidade de Franca(2000), Especialização em Psicanálise Contemporânea(2006), Doutorado em Direito pela Universidade do Vale do Rio dos Sinos(2006), Pós Doutorado em Direito e Saúde na Università Degli Studi Di Messina - Itália(2014/2015), Aperfeiçoamento em Bioética Aplicada às Pesquisas em Seres Humanos(2013), Aperfeiçoamento em Educação para Jovens e Adultos pelo CEETEPS(2014) e Especialização em Gestão da Organização da Saúde Pública(UNIRIO/2014).Professora Universitária do Ensino Superior, da Faculdade de Tecnologia Dr. Thomaz Novelino em Franca - FATEC FRANCA(2010). Profissional liberal - Ordem dos Advogados do Brasil(1992), Professora Coordenadora Autora na Área de Direito Empresarial do Programa UNIVESP/Centro Paula Souza - Universidade Virtual do Estado de São Paulo(2010); Aperfeiçoamento em Direito à Saúde Baseada em Evidências(2015); Hospital Sirio Libanês/SP. Aperfeiçoamento em Educação, Pobreza e Desigualdade Social - UFSCar(2019); Especialização em Informática na Educação no Instituto Federal de Educação, Ciência e Tecnologia de São Paulo Câmpus São João da Boa Vista(2021); Especialização em Direitos Humanos pela Universidade Federal do ABC - UFABC(2022). Pesquisadora - Cátedra Oscar Sala - Instituto de Estudos Avançados da Universidade de São Paulo - IEA/USP(2022-2023).Professora Titular na Faculdade de Direito de Franca(2020).Professora de Ensino Superior na Faculdade de Tecnologia de Ribeirão Preto/SP(2018).Experiência nas áreas Direito Constitucional, Direito Civil, Direito Comercial/Empresarial, Direito do Trabalho, Direito Tributário, Propriedade Industrial, Inovação Tecnológica e Direito Autoral, Direito Digital, Direito Educacional e Direito à Saúde, Bioética e Políticas Pública. ***Link para o Lattes:***

http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K4774414J1

André S. A. Carvalho

Estudante do curso de Análise e Desenvolvimento de Sistemas na Faculdade de Tecnologia de Itu - Fatec Itu. Autor de capítulos de livro e artigos referentes à área de Inteligência Artificial com enfoque no tema de Detecção e Reconhecimento Facial. Atualmente, atua como Estagiário de TI na empresa Auto Geral Autopeças LTDA e utiliza tecnologias Backend e Frontend para desenvolver soluções que suprem as necessidades estabelecidas pela organização. Possui diversas certificações, possuindo maior conhecimento na área de Desenvolvimento Móvel. ***Link para o Lattes:***

http://lattes.cnpq.br/7307398822553705

Link para o Linkedin:

https://www.linkedin.com/in/andre-alckmin/

Link para o GitHub:

https://github.com/andre-alck

Ed Carlos S. Pereira

Analista de TI pleno na Guarnieri Sistemas de Segurança e estudante do 5º semestre do curso de Análise e Desenvolvimento de Sistemas da Fatec Itu. Tem 18 anos de experiência na área de tecnologia, tendo começado como técnico em informática fazendo manutenção e montagem de computadores. Especializou-se em instalação e administração de servidores Linux e obteve certificações em cabeamento estruturado de redes das marcas Amp Tyco e LCS2 Legrand. Atualmente, está se concentrando em certificações em linguagens de programação, como Python, e em análise de dados com Python. ***Link para o Lattes:***

http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K1152044Y6

Link para o Linkedin:

https://www.linkedin.com/in/dev-python-junior/

Link para o GitHub:

https://github.com/Goghed

Erick José da Silva

Tecnólogo em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia Dom Amaury Castanho – FATEC Itu. É autor de um capítulo de livro e um artigo na área de detecção e reconhecimento facial. Ambos os trabalhos buscam mostrar e avaliar algoritmos utilizados nesta área da Inteligência Artificial. Atualmente é Analista de Dados, e utiliza o Power BI na criação de relatórios e dashboards. Também possui duas certificações Microsoft: PL-900 (Power Platform Fundamentals) e DP-900 (Azure Data Fundamentals). ***Link para o Lattes:***

http://lattes.cnpq.br/7004174461105224

Link para o LinkedIn:

https://www.linkedin.com/in/erick-jsilva/

Francisco de Assis de Freitas

Graduado em Análise e Desenvolvimento de sistemas pela FATEC Itu e Pós-graduado em Desenvolvimento em tecnologias digitais pela Estácio. Atua como Analista de Sistemas na B3 (Brasil Bolsa Balcão), um dos idealizadores do projeto SAA (Sistema de Avaliação da Aprendizagem) com publicações focadas em Sala de Aula Invertida (Flipped Classroom). ***Link para o Lattes:***

http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K8059101Y0

Link para o LinkedIn:

https://www.linkedin.com/in/francisco-freitas

Link para o GitHub:

https://github.com/assis-freitas

Índice

Apresentação

Introdução

Parte 1 – O Universo do Reconhecimento Facial: Histórico, teoria e legislação

Capítulo 1: Os principais métodos e algoritmos de detecção e reconhecimento facial Parte 1: de 1973 a 2008.
Capítulo 2: Os principais métodos e algoritmos de detecção e reconhecimento facial Parte 2: de 2009 aos dias atuais.
Capítulo 3: Classificação de Métodos e Algoritmos de Detecção e Reconhecimento Facial.
Capítulo 4: Ferramentas e Serviços para Detecção e Reconhecimento Facial.
Capítulo 5: O reconhecimento facial à luz da Legislação Brasileira

Parte 2 – A prática: Detalhamento dos Principais Algoritmos.

Capítulo 6: Eigenfaces.
Capitulo 7: Haar Cascade.
Capítulo 8: HOG – Histogram Orientaded Gradient.
Capítulo 9: CNN - Redes Neurais Convolucionais.
Capítulo 10: DeepFace.
Capítulo 11: YOLO – You Only Look Once.
Capítulo 12: SisCoP – Sistema de Controle de Presença: um exemplo de aplicação utilizando reconhecimento facial.

Referências Bibliográficas

Anexo A - Criação de um detector Haar Cascade: Manual e Automaticamente.
Anexo B - Preparação do ambiente para treinamento e detecção utilizando CNN.

Apresentação

Pouco tempo atrás, precisávamos da chamada “caderneta” para identificarmos a presença de nossos alunos em sala de aula. Hoje em dia, há pouca coisa imaginada que não possa ser desenvolvida, e, sem dúvida, essa é uma obra que demonstra os avanços das tecnologias da informação e suas aplicações em diferentes espaços.

A iniciativa de publicar o livro intitulado Detecção e Reconhecimento Facial: uma abordagem prática é resultado da dedicação e esforço dos autores com o objetivo de expandir o diálogo e as análises sobre a temática em pauta, realizando um breve histórico em relação aos seus principais métodos e algoritmos e demonstrando a aplicação destes métodos e algoritmos de detecção e reconhecimento facial criados ao longo do tempo.

Destaca-se a atualidade do tema e a preocupação em envolver profissionais que desenvolveram pesquisas recentemente, sobretudo, representa a concretização do desejo dos autores em produzir textos consistentes e modernos, voltados para a aplicação dos conceitos e integrando outras tecnologias, evidenciando o reconhecimento facial como um dos métodos biométricos de fácil acesso, já que a sua coleta é facilitada em qualquer ambiente, bastando que pessoa esteja presente na hora e lugar da autenticação e o seu funcionamento depende da utilização de câmeras simples.

Coube a mim a prazerosa atividade de realizar essa apresentação e dizer que, todos nós, que desenvolvemos pesquisas que aplicam as diferentes tecnologias, sentimo-nos honrados com o privilégio de trazer a público este livro.

É um livro a ser adotado, principalmente, em nossos cursos de graduação tecnológica, por parte dos alunos, professores e profissionais das áreas de tecnologia da informação e áreas correlatas.

Profa Dra Juliana Augusta Verona
Diretora da Faculdade de Tecnologia de Itu

Introdução

O reconhecimento facial consiste em identificar padrões em características faciais como o formato da boca e do rosto, distância entre os olhos, contornos, entre outros.

O ser humano é capaz de identificar as pessoas por meio de suas faces e faz isso muito bem, mesmo em ambientes não controlados, com variações na iluminação, na pose, nas expressões faciais, no uso de maquiagens e acessórios etc.

Essas funcionalidades despertaram o interesse de um grande número de cientistas e pesquisadores, que passaram então a estudar o sistema cognitivo e perceptual dos indivíduos a fim de compreendê-los e então reproduzi-los artificialmente.

O reconhecimento facial automático se mostra muito importante em nossa sociedade, possibilitando o desenvolvimento de diversas aplicações, tais como: controle de acesso, segurança pública e privada, autenticação de documentos, validações de transações financeiras entre muitas outras.

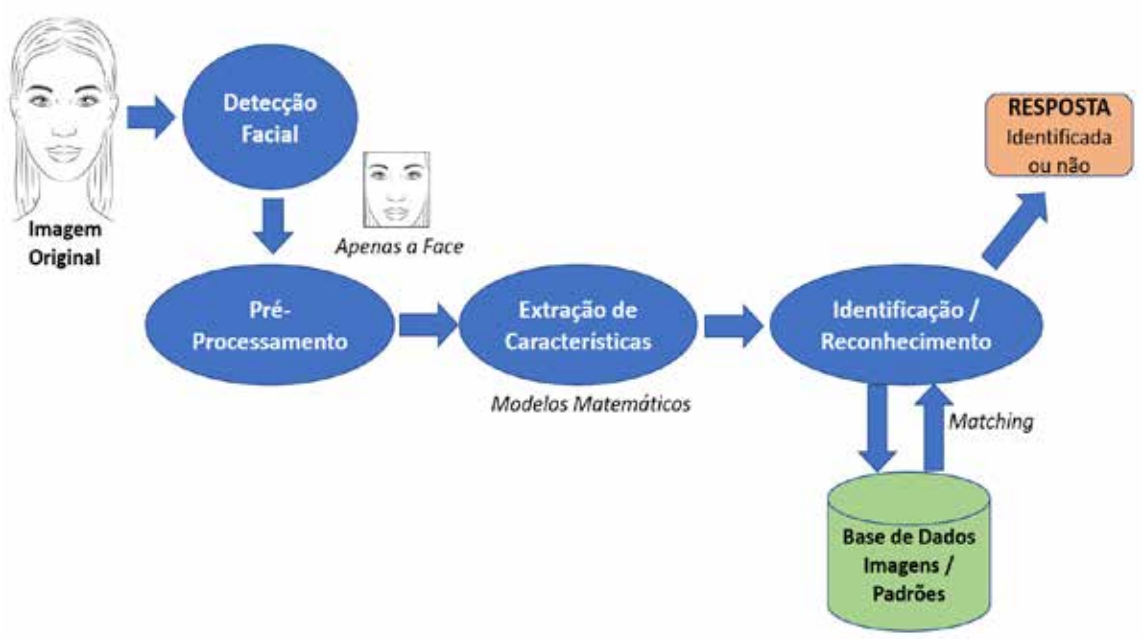
Considerado como um dos métodos biométricos de reconhecimento de padrão, o reconhecimento facial, utiliza características particulares da face. Biometria é a ciência que estabelece a identidade de uma pessoa baseada em seus atributos físicos, químicos ou comportamentais.

A utilização do reconhecimento facial em detrimento de outras técnicas biométricas proporciona algumas vantagens, as quais podemos enumerar: os traços biométricos da face não podem ser perdidos e nem esquecidos, são difíceis de serem copiados, compartilhados ou distribuídos. Além disso, a sua coleta requer que a pessoa esteja presente na hora e lugar da autenticação. O reconhecimento facial dispensa a utilização de equipamentos especializados, bastando para tanto a utilização de câmeras simples. (STAN e ANIL, 2011).

Este tema vem despertando, há décadas, o interesse de muitos pesquisadores ao redor do mundo todo. Um dos primeiros trabalhos de grande relevância na área foi apresentado por Kanade (1973), onde as faces são descritas pelas suas características geométricas (distâncias e ângulos de seus pontos fiduciais, como cantos dos olhos e extremos da boca).

Depois disso, ao longo dos anos, as pesquisas nesta área específica de Ciência da Computação foram se desenvolvendo. Notamos que as pesquisas tiveram grande avanço na década de 1990 e depois, com a utilização das redes neurais e principalmente as redes neurais profundas, após 2009, atingiu patamares espetaculares, onde a acurácia chega a ultrapassar os próprios seres humanos.

O processo atualmente utilizado pelos principais algoritmos de reconhecimento facial, que pode ser resumido na figura a seguir:



Como pode ser observado, uma vez feita a captura de uma imagem, os algoritmos de reconhecimento facial verificam inicialmente se existe uma face na imagem. A essa tarefa, dá-se o nome de **Detecção Facial**. Normalmente quando utilizamos uma máquina fotográfica moderna, seria aquele quadrado colocado ao redor de uma face. Como ocorre na figura ao lado.

Detecção Facial

Reconhecimento Facial

Em seguida, uma vez identificada a face, é feito o pré-processamento (normalização) da face, e então extraídas característica / recursos por meio de diversos métodos. Esses recursos são então comparados a um banco de dados de recursos anteriormente armazenado (pré-existente). Caso exista uma compatibilidade de recursos (coincidência ou *matching*), a face identificada é então associada a essa pessoa inicialmente armazenada. A essa etapa dá-se o nome de **Reconhecimento Facial**. A figura ao lado apresenta essa última etapa.

Trazemos neste livro uma visão geral da retrospectiva dos principais métodos e algoritmos que tratam dos temas de Detecção e Reconhecimento Facial ao longo do tempo nos capítulos 1 e 2. O primeiro trata dos métodos antes das redes neurais e o segundo capítulo, foca os métodos pós-revolução das redes neurais e redes neurais profundas.

No terceiro capítulo trazemos uma visão geral da classificação dos métodos e algoritmos segundo várias perspectivas e no capítulo 4 apresentamos as principais ferramentas atualmente disponibilizadas, gratuitamente ou não, que servem para agilizar processos de desenvolvimento de ferramentas voltadas para a área de detecção e reconhecimento facial.

O quinto capítulo apresenta o que há de mais atual em nossa legislação (legislação brasileira) no que tange a utilização de técnicas, métodos e algoritmos de reconhecimento facial, em projetos de todos os níveis. Este capítulo é fundamental para os profissionais da área de desenvolvimento, principalmente a parte gerencial, para saberem os limites e as precauções necessárias que devem ser tomadas em projetos envolvendo tais tecnologias.

A partir do capítulo 6, chegamos a parte prática deste livro. É onde aprofundamos e detalhamos os principais métodos e algoritmos de detecção e reconhecimento facial criados ao longo da história. Serão mostrados 6 métodos/algoritmos, sendo 3 deles pré-redes neurais e outros 3 envolvendo a utilização intensiva das redes neurais profundas.

O capítulo 6 apresenta o método/algoritmo Eigenfaces. O próximo capítulo, 7, apresenta o método/algoritmo Haar Cascade e no capítulo 8, apresentamos o HOG, acrônico de *Histogram Oriented Gradient* (ou Histograma de Gradiente Orientado).

Já os métodos/algoritmos que utilizam redes neurais e redes neurais profundas iniciam no capítulo 9 com as Redes Neurais Convolucionais. No próximo capítulo, 10, é apresentado o método/algoritmo DeepFace e por fim, no capítulo 11, o método/algoritmo YOLO, acrônico de *You Only Look Once* (ou Você Olha ou Observa apenas uma Vez) é descrito.

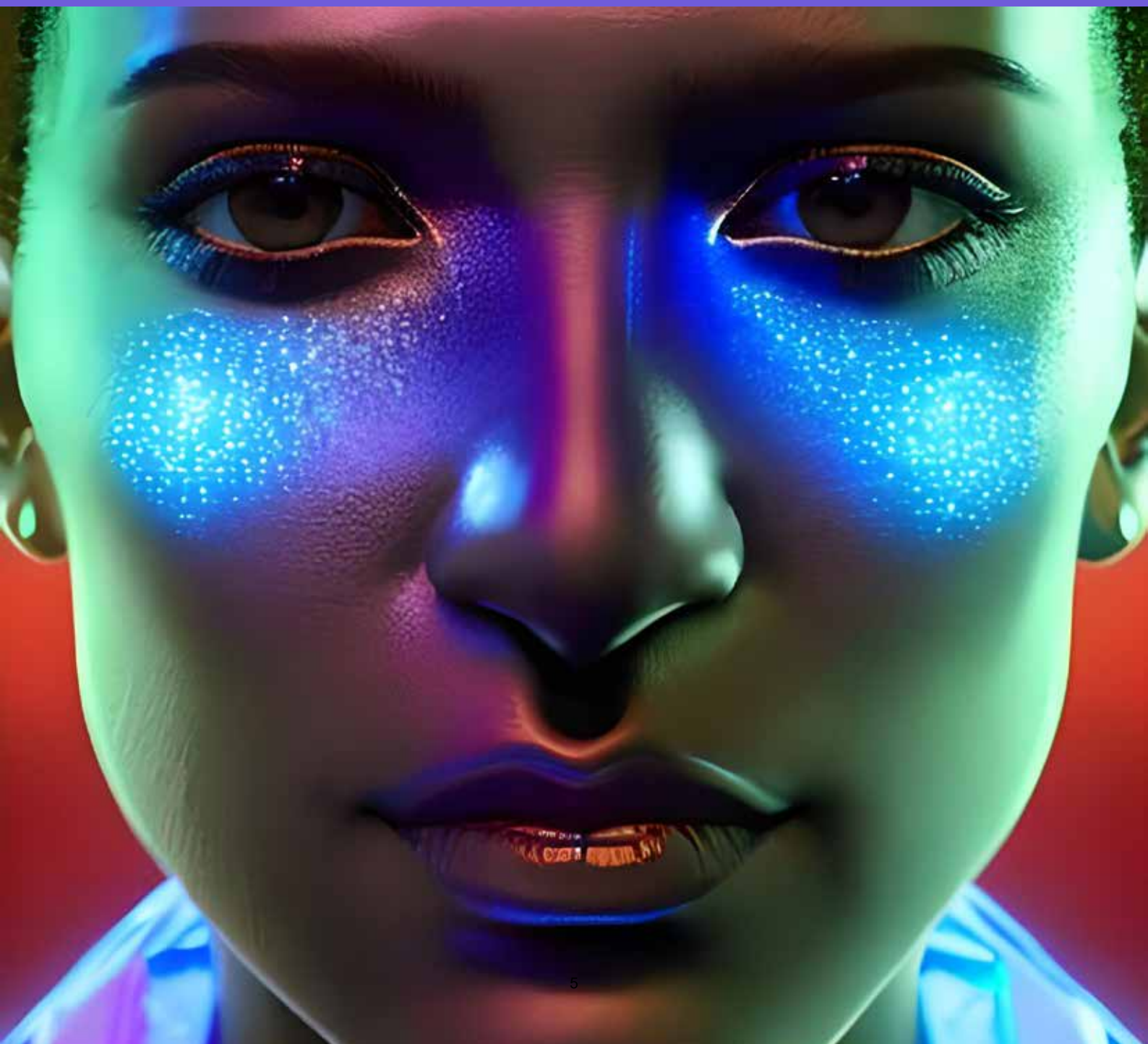
Para fechar a parte prática, apresentamos um sistema desenvolvido por dois dos autores, que utiliza a tecnologia de detecção e reconhecimento facial para validar a presença de estudantes em um local específico, como a sala de aula (utilizando também a tecnologia de geoprocessamento). O código fonte completo é também disponibilizado no final do capítulo.

Esperamos que você aprecie esta obra, sabendo que ela representa um retrato em um determinado momento do desenvolvimento tecnológico, que além de ser dinâmico é muito veloz.

Mesmo diante disso, acreditamos que esta obra pode ser um passo inicial para as pessoas que se interessam em conhecer e, posteriormente, se aprofundar nesta fantástica área da tecnologia da informação. Boa leitura.

Parte 1

O Universo do Reconhecimento Facial: Histórico, teoria e legislação



1

Os Principais Métodos e Algoritmos de Detecção e Reconhecimento Facial
Parte 1: de 1973 a 2012

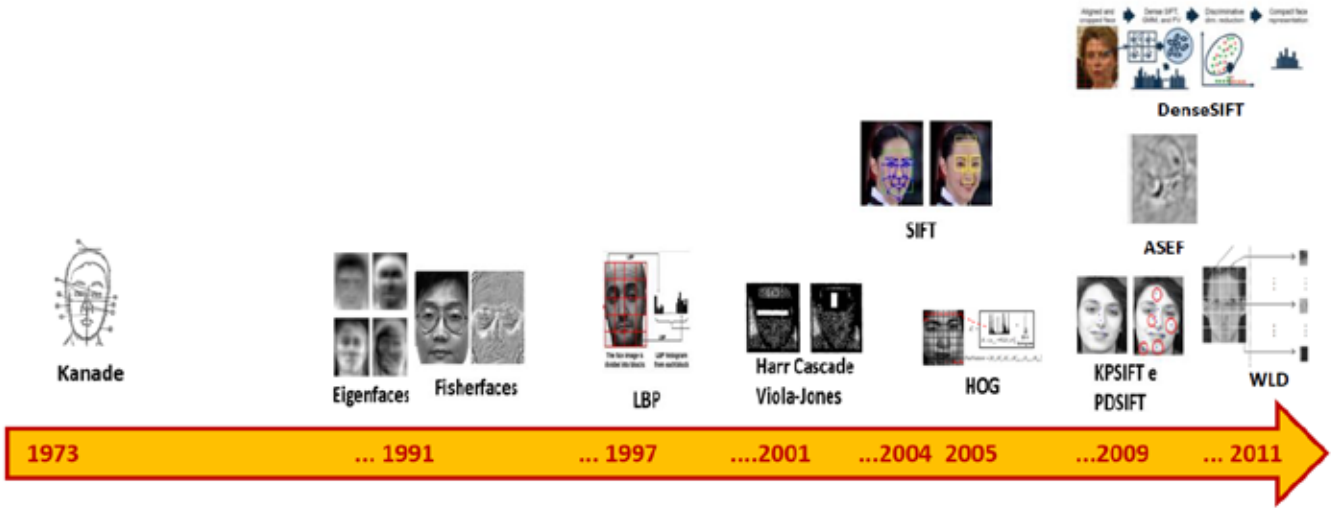
Os métodos e algoritmos de detecção e reconhecimento facial, até o início deste milênio, se restrin-giam a algoritmos que tinham como base modelos matemáticos e possuíam uma performance e acurácia ainda não tão boas.

Com o advento das redes neurais, isso tudo mudou. Os métodos e algoritmos de detecção e reconhecimento facial ganharam robustez, velocidade e acurácia.

Nesta primeira parte, vamos fazer uma recapitulação dos principais métodos e algoritmos até 2012, que são, essencialmente, implementações sem a utilização de redes neurais.

A Figura 1.1 a seguir ilustra a linha do tempo e os principais métodos desta primeira parte de trabalhos sobre reconhecimento facial.

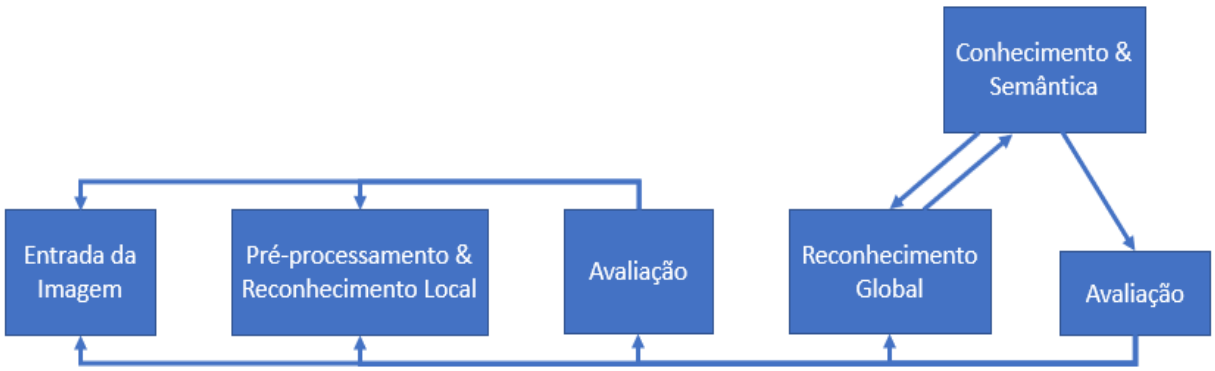
Figura 1.1: Principais métodos de reconhecimento facial da Primeira Parte da linha do tempo (de 1973 a 2012)



Um dos primeiros trabalhos de grande relevância na área foi apresentado por Kanade(1973), onde as faces são descritas pelas suas características geométricas (distâncias e ângulos de seus pontos fiduciais, como cantos dos olhos e extremos da boca).

É neste trabalho que encontramos a ideia inicial de um fluxo de processamento da imagem capturada, que depois vai ser utilizado, ampliado e aprimorado por outros estudos. A Figura 1.2 apresenta esta ideia.

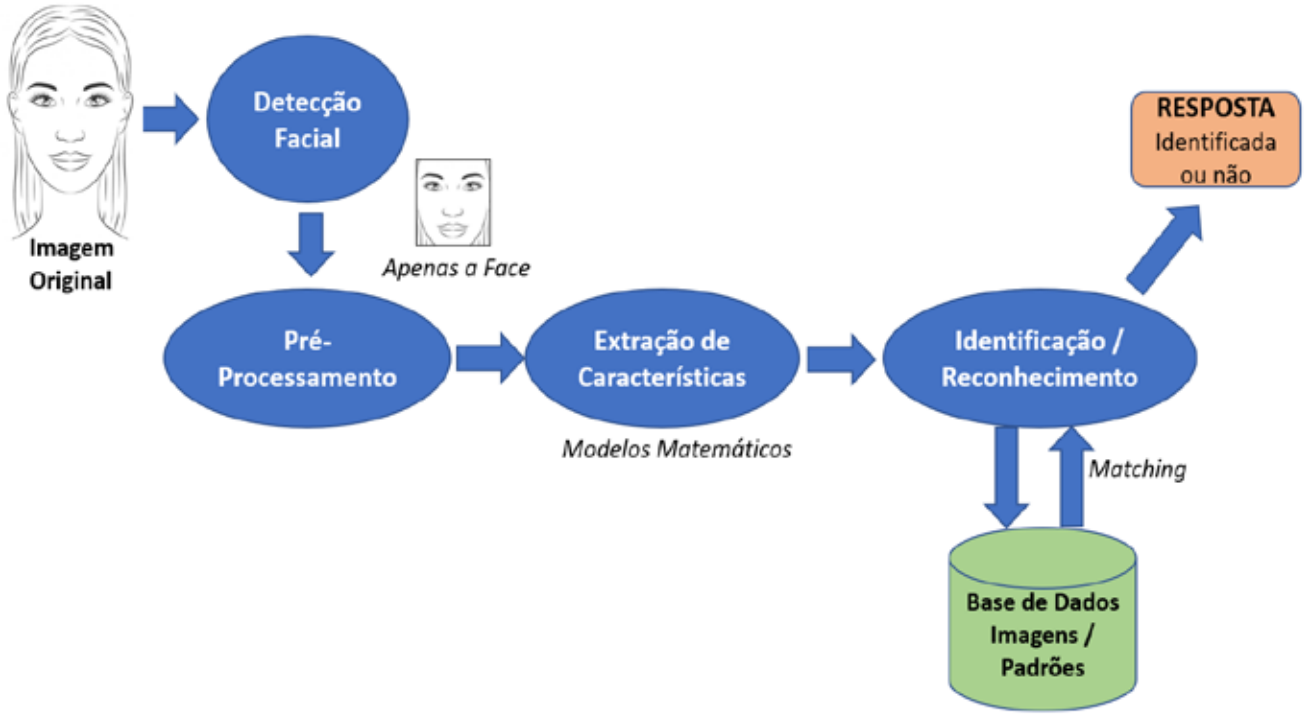
Figura 1.2: Etapas do fluxo de processamento e reconhecimento da imagem proposto por Kanade (1973).



Fonte: Adaptado de (KANADE, 1973 p. 13)

Como pode ser visto, esse fluxo proposto por Kanade (1973) é muito semelhante ao processo atualmente utilizado pelos principais algoritmos de reconhecimento facial, que pode ser resumido na Figura 1.3.

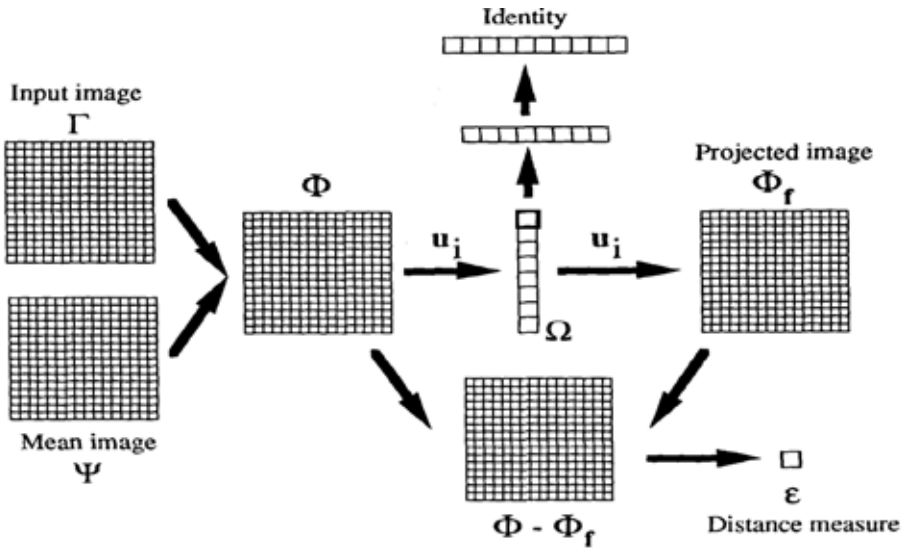
Figura 1.3: Etapas do processo de reconhecimento facial (métodos/abordagens recentes)



Pode-se perceber, seguindo a linha do tempo na Figura 1.1, que o primeiro trabalho relevante após Kanade(1973) foi o proposto por Turk e Pentland (1991): as autofaces ou, do inglês, as *Eigenfaces*.

Neste método, os autores propuseram a aplicação da técnica PCA (do inglês, *Principal Component Analysis*) para a redução da dimensionalidade dos dados no reconhecimento facial, onde uma face passa a ser representada por um vetor contendo seus componentes principais. A Figura 1.4, ilustra essa redução vetorial pela técnica PCA.

Figura 1.4: Processo de redução vetorial baseado em componentes principais proposta por Turk e Pentland (1991).



Fonte: TURK e PENTLAND, 1991, p. 85.

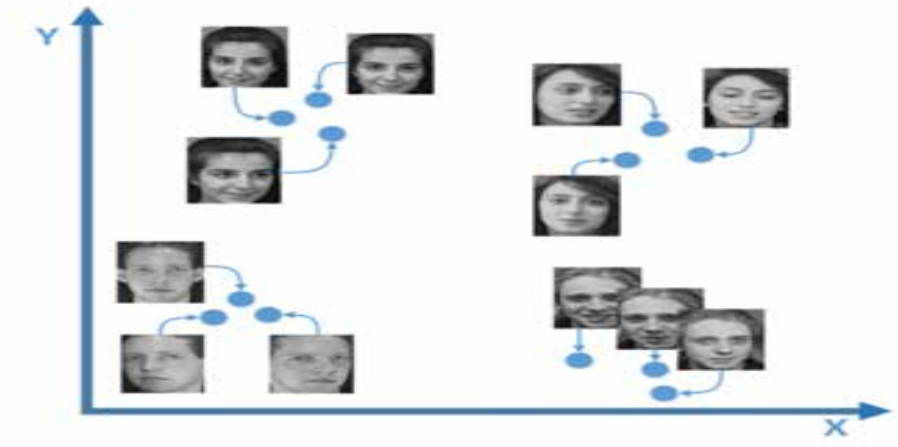
O método/algoritmo Eigenfaces será tratado com mais detalhes, inclusive todo o processo de implementação no capítulo 6.

Ainda com base na linha do tempo (Figura 1.1), a próxima técnica proposta, e ainda muito conhecida, foram as *Fisherfaces* (BELHUMEUR; HESPANHA; KRIEGMAN, 1997), onde é empregada a técnica de redução de dimensão dos dados conhecida como LDA (do inglês, *Linear Discriminant Analysis*). Inclusive no trabalho, a técnica *Fisherfaces* é comparada com a *Engenfaces*, com uma redução de taxa de erros em dez vezes (de 52,6% - PCA para 5,3% *Fisherface* – LDA).

Essa redução foi possível em virtude de a técnica LDA, mais do que reduzir a dimensão dos dados, maximiza o raio de variância entre as classes ao mesmo tempo em que minimiza a variância dentro das classes.

No método Fisherface, a LDA é trabalhada com o uso de “rótulos” (cada face é identificada como pertencente a uma pessoa específica). Em seguida, as faces são agrupadas por pessoa, e cada agrupamento desses é conhecido como uma classe. Assim, esse método busca modelar a dispersão dos pontos visando maior confiabilidade para a classificação. O LDA busca encontrar e otimizar a melhor linha de separação das classes em uma superfície. A Figura 1.5 ilustra esses espaços/classes. Nesta figura, é possível notar a separação das classes com base nas semelhanças. Quanto mais distinta uma imagem da outra, maior a distância, e vice-versa.

Figura 1.5: Espaço de faces da LDA



Fonte: SILVA, 2016, p.33

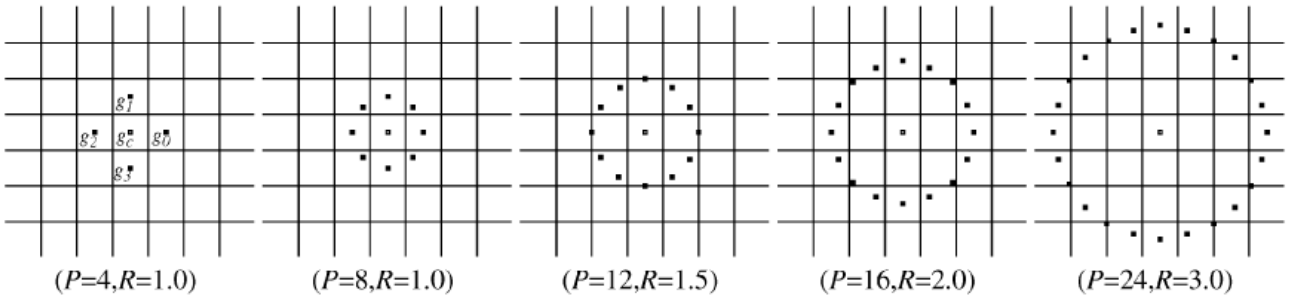
O algoritmo *Fisherfaces* como detalha Bissi (2018), é composto por 8 etapas, sendo elas:

1. Cálculo da face média por classe.
2. Cálculo de face média geral.
3. Transformação das imagens em vetores (pixels).
4. Construção da matriz de dispersão intra-classe.
5. Construção da matriz de dispersão inter-classe.
6. Cálculo das fisherfaces (os autovetores e os autovalores).
7. Cálculo dos vetores de características.
8. Cálculo da Similaridade (ex. distância euclidiana). (BISSI, 2018, p. 25-26)

Ainda no estudo comparativo proposto por Belhumeur, Hespanha e Kriegman (1997), a técnica PCA reduz a dimensão dos dados pela representação dos mesmos pelas seus componentes principais sem reduzir a variância dos pontos de dentro de uma determinada classe. Enquanto que, a técnica LDA utiliza as informações de agrupamento, incrementando assim a separação entre as classes. Por essa razão, os vários experimentos apresentados pelos autores, mostram que LDA tem melhores resultados que PCA.

O próximo método, na linha do tempo apresentada na Figura 1.1, é o LBP (do inglês, *Local Binary Pattern ou Padrão Local Binário*) proposto em Ojala; Pietikainen; e Harwood (1996) e que foi largamente explorado por diversos autores. Trata-se de um método que utiliza outros métodos empregando descritores locais para o reconhecimento facial. A Figura 1.6, ilustra um dos princípios trabalhados pelos autores em sua publicação: Rotação LBP invariante.

Figura 1.6: Rotação LBP invariante, apresentando o conjunto de vizinhos circularmente simétricos para diferentes (P.,R.)



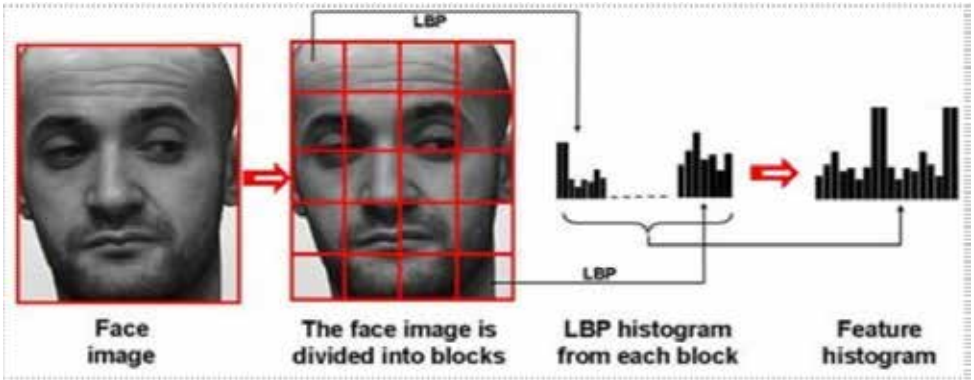
Fonte: (OJALA; PIETIKAINEN; HARWOOD, 1996, p. 973).

A ideia por trás da criação do operador LBP era que as texturas de superfície bidimensionais podem ser descritas por duas medidas complementares: padrões espaciais locais e contraste na escala de cinza. (PIETIKÄINEN, 2010)

O LBP original considerava uma vizinhança 3x3 em torno de cada pixel e, para cada pixel, comparava-o com o pixel central e atribuí 1 para o pixel se ele é maior ou igual ao pixel central e 0 caso contrário. O código LBP de cada pixel é calculado como um número de 8 bits em que o limiar de cada pixel recebe os pesos abaixo. Por fim, é feito um histograma dos códigos LBP calculados. Este histograma representa a textura presente na imagem.

A Figura 1.7, resume esses passos do algoritmo LBP.

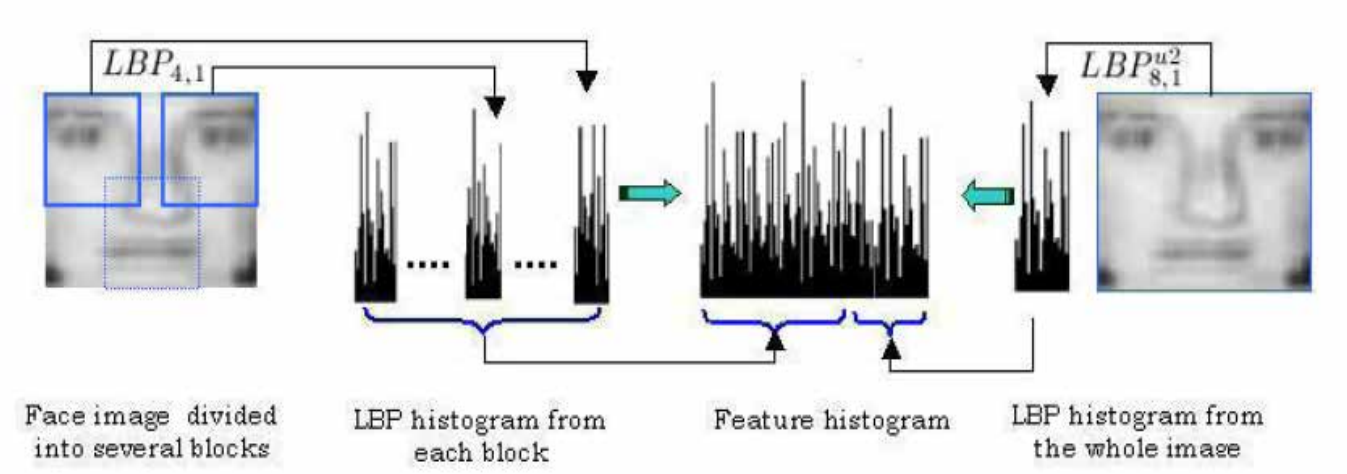
Figura 1.7: Etapas do algoritmo LBP.



Fonte: PIETIKÄINEN, 2010

Ainda dentro dessa mesma técnica LBP, os autores Hadid, Pietikainen e Ahonen (2004) em seu trabalho mostram que uma face é representada por suas características LBP, obtidas local e globalmente, e codificadas em um histograma. A Figura 1.8 apresenta a representação facial proposta pelos autores, onde cada imagem da face é representada pela concatenação de histogramas globais e locais dos LBP.

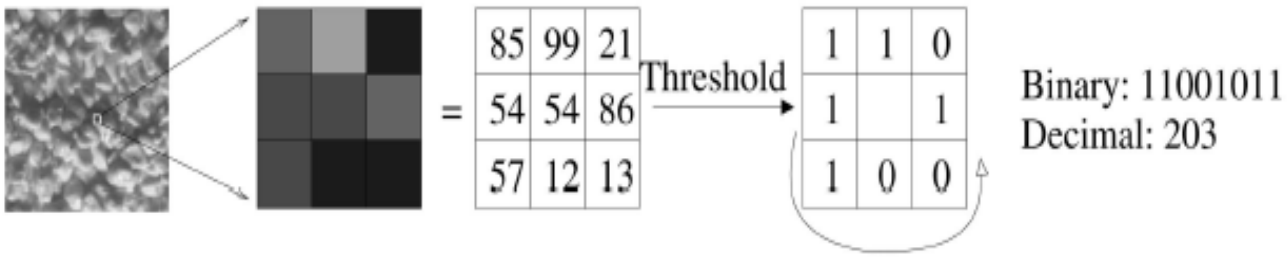
Figura 1.8: Representação facial – uma imagem de uma face é representada pela concatenação de histogramas locais e globais dos LBP da imagem.



Fonte: HADID; PIETIKAINEN; AHONEN, 2004 p. 1065.

Os mesmos autores, em um trabalho mais recente (AHONEN; HADID; PIETIKAINEN, 2006) e ainda utilizando a técnica LBP, dividiram a imagem da face em várias regiões e, para cada região, foi extraído um vetor de características baseado no histograma LBP da mesma. Desta forma, cada região foi descrita localmente e a combinação de todas as regiões levou a uma descrição global da face. A Figura 1.9, apresenta essa ideia.

Figura 1.9: Uma região da face com seu vetor de características baseado no histograma LBP



Fonte: (AHONEN; HADID; PIETIKAINEN, 2006, p. 2038).

Regressando à Figura 1.1, na linha do tempo, em 2001 um marco na história do reconhecimento facial aconteceu com o trabalho de Viola e Jones (2001). Neste trabalho os autores propuseram uma abordagem de um detector de base holística, treinando vários classificadores usando uma cascata impulsionada de características simples, para reconhecimento de objetos, que após encontrar a imagem integral, cada classificador reconhecia um padrão sobre a imagem, de modo que combinados em modo cascata, entravam em acordo sobre o objeto reconhecido, classificando-o com base sobre o voto majoritário.

O algoritmo proposto pelos autores é dividido em três etapas:

Etapa 1) criação da imagem integral e a representação da imagem em um espaço de características baseados nos filtros de *Haar*. Essa etapa também é conhecida como tabela de soma de áreas, que é um algoritmo proposto por Frank Crow em 1984 (CROW, 1984).

Etapa 2) consiste, segundo os autores, na montagem de um classificador de aprendizado *Boosting* chamado *Adaptative Boosting* (AdaBoost) para selecionar as características relevantes. Conforme Viola e Jones (2001), "*[...] é necessário treinar o sistema com imagens das faces (positivas) e imagens de não face (negativas) e para isso, deve-se utilizar algum algoritmo de aprendizagem que use o método AdaBoost*".

Etapa 3) a última etapa proposta pelos autores é a criação de uma estrutura em árvore chamada de cascata de classificadores (*HaarCascade*). Segundo Viola e Jones (2001):

> "[...] cada um dos estágios da cascata aplica um classificador mais específico e mais complexo que o aplicado na cascata anterior. Isso é feito de tal forma, para que o algoritmo rejeite rapidamente regiões que sejam muito distintas da característica procurada e, assim, termine o processo de procura, evitando que os estágios posteriores sejam executados desnecessariamente.". (VIOLA; JONES, 2001, p.514).

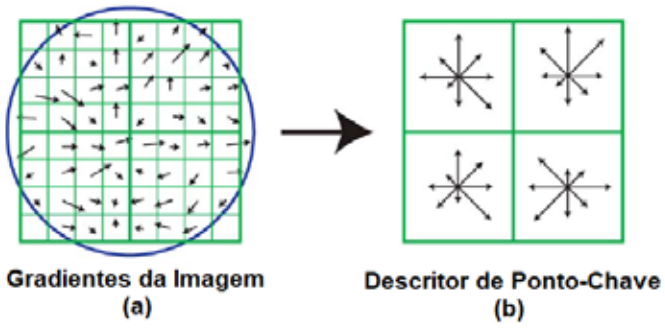
O algoritmo proposto por Viola e Jones (2001) quando bem treinado, tem uma boa precisão, e consegue detectar a maioria das faces que são captadas frontalmente, além de apresentar um baixo número de falsos positivos.

No capítulo 7 este algoritmo e todo o processo de implementação serão melhor detalhados.

Continuando em nossa linha do tempo apresentada na Figura 1.1, quase que simultaneamente com os descritores LBP, é proposto um dos descritores mais clássicos encontrados na literatura: o SIFT (do inglês, *Scale Invariant Feature Transform*). Inicialmente proposto por Lowe (2004). A Figura 1.10 ilustra um descritor de ponto-chave que foi criado calculando primeiro a magnitude e a orientação do gradiente em cada ponto de amostra de imagem em uma região ao redor do local do ponto-chave, conforme mostrado em (a).

São ponderados por uma janela gaussiana, indicada pelo círculo sobreposto. Essas amostras são então acumuladas em histogramas de orientação, resumindo o conteúdo em sub-regiões 4x4, como mostrado em (b), com o comprimento de cada seta correspondente à soma das magnitudes do gradiente próximas àquela direção na região.

Figura 1.10: Descritor de ponto-chave baseado no gradiente da imagem.



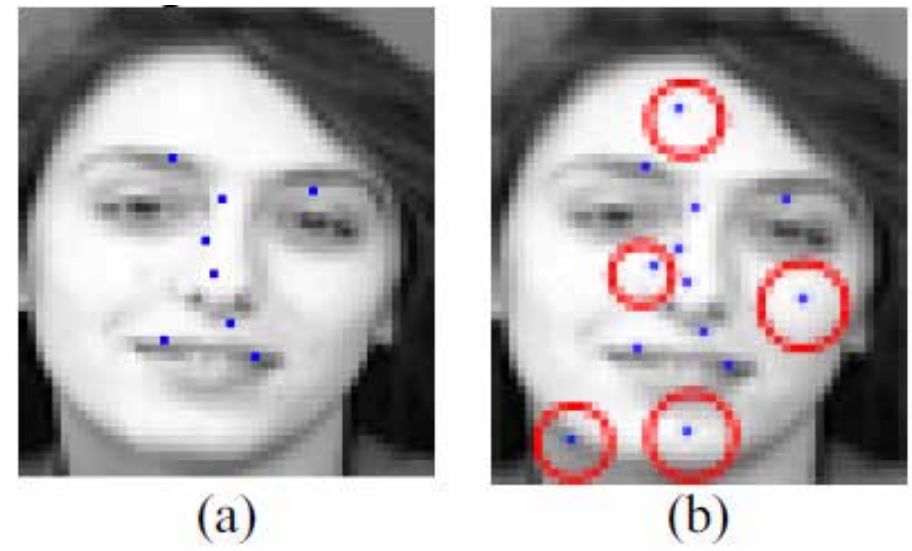
Fonte: Adaptado de LOWE, 2004, p. 101.

Essa proposta de SIFT feita por Lowe (2004) remove os pontos chave com baixo contraste ou que são localizados ao longo de bordas dos objetos.

Outros trabalhos, posteriormente, utilizaram SIFT em processos de reconhecimento facial: Bicego e colaboradores (2006), Luo et al. (2007), Zhang et al. (2008), Majumdar e Ward (2009) e Geng e Jiang (2009),

Especificamente no trabalho proposto por Geng e Jiang (2009), onde a partir da versão original do SIFT propuseram duas variações. A primeira é o KPSIFT (do inglês *Keypoints-Preserving SIFT*) que preservam todos os pontos-chave detectados pelo método SIFT. A segunda é o PDSIFT (do inglês *Partial-Descriptor-SIFT*) que descreve parcialmente os descritores cujos pontos-chave são localizados em escalas grandes ou próximos aos limites da face. A Figura 1.11 ilustra os dois novos métodos, com as marcações correspondentes.

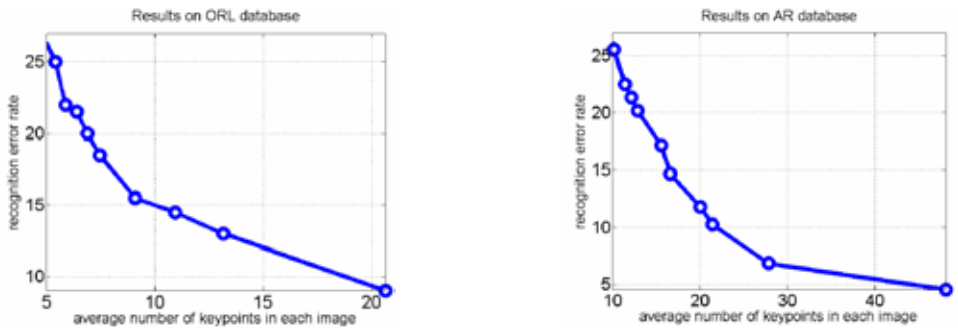
Figura 1.11: Pontos-chave detectados por (a) KPSIFT e (b) PDSIFT



Fonte: GENG; JIANG, 2009, p. 601

Os autores comparam os dois novos métodos com o método original SIFT, utilizando para isso duas bases de dados: ORL e AR. Os dois métodos apresentam, em ambas as bases de dados, melhores resultados que o método tradicional SIFT. A Figura 1.12 ilustra a redução na taxa de erros, ao se utilizar os novos métodos em ambas as bases de dados (a) ORL e (b) AR.

Figura 1.12: Taxa de erro no reconhecimento utilizando SIFT em comparação com outros métodos KPSIFT e PDSIFT, utilizando as bases: (a) ORL e (b) AR.



(a) (b)

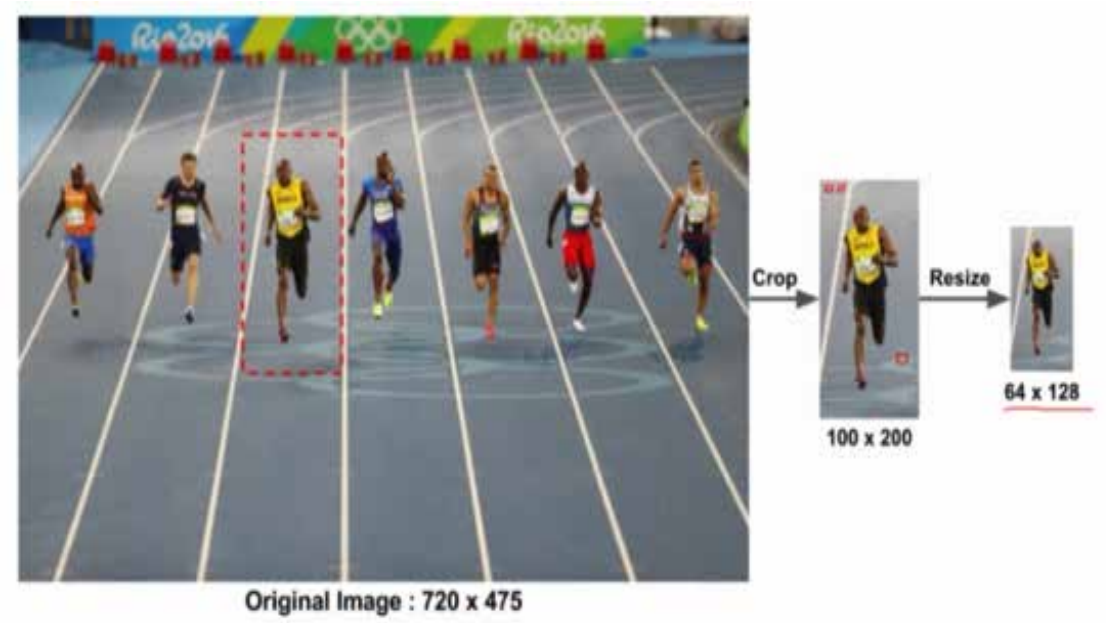
Fonte: GENG; JIANG, 2009, p. 601 e 602

Retornando à linha do tempo esquematizada na Figura 1.1, um outro método proposto por Dalal e Triggs (2005) é conhecido como HOG (do inglês, *Histograms of Oriented Gradients ou Histograma Orientado*

a Gradientes). Este método utiliza um descritor de características que identifica as informações principais de parte da imagem por meio de formas e textura. Ao longo do desenvolvimento de algoritmos de reconhecimento facial, ele é visto como um dos melhores descritores para identificar pessoas, pois se pensarmos pelo olho humano, nós conseguimos distinguir que o veículo não é uma pessoa só pela forma dos dois.

O funcionamento básico do algoritmo, descrito pelos autores, se dá da seguinte forma: o algoritmo captura uma parte da imagem, redimensiona essa parte, geralmente 64x128 pixels (mas dependendo da situação esse número pode mudar). A Figura 1.13, ilustra esse redimensionamento.

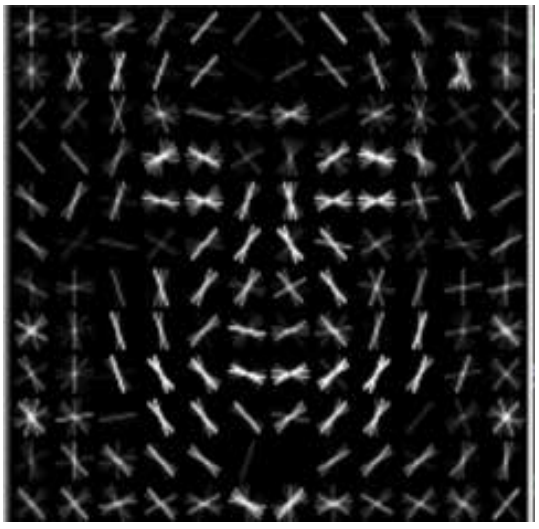
Figura 1.13: Aplicação do recorte e redimensionamento da imagem



Fonte: MALLICK, 2019

Em seguida, o algoritmo aplica derivada para saber a taxa de variação de cada pixel, e o gradiente para saber o nível de intensidade de cada pixel vizinho. Geralmente existem 3 situações diferentes para o resultado da derivada: zero (sem variação), baixo (pouca variação) e alto (grande variação). Em seguida, calcula as matrizes de gradiente de magnitude e de direção. Com base nestas duas matrizes, o algoritmo calcula então o histograma, por classe de pixels (direção do gradiente e intensidade). O resultado disso, além do histograma, é a imagem texturizada (que também poderá ser utilizada para o processo de *matching* futuro). A Figura 1.14 ilustra a imagem texturizada da face.

Figura 1.14: Texturização da Face



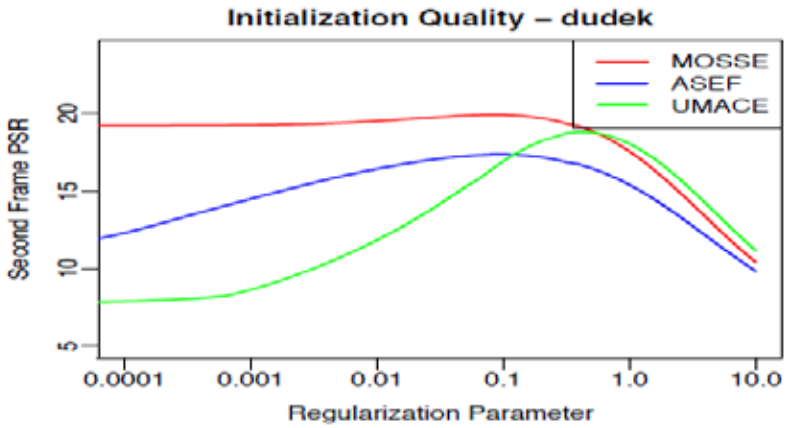
Fonte: MALLICK, 2019

No capítulo 8, traremos mais detalhes e o processo de implementação deste método/algoritmo.

Depois de Dalal e Triggs (2005), outros trabalhos apresentaram a utilização de HOG, principalmente combinado com outros métodos/técnicas. Em todos eles, o desempenho dos algoritmos de reconhecimento facial foi sempre muito bom, mostrando sua versatilidade e a baixa utilização de imagens na base de dados. Alguns deles são: (QIANG et. Al, 2006), (NING HE; LIN SONG, 2008), (DÉNIZ et al, 2011), (DO; KIJAK, 2012) e (XIE; JIANG; ZHANG, 2017).

Observando agora a Figura 1.1, o próximo método que se apresenta é conhecido como ASEF (do inglês, *Average of Synthetic Exact Filters*) foi proposto por Bolme, Draper e Beveridge (2009) e utiliza filtros de correlação no domínio da frequência para localizar pontos de interesse em uma face como, por exemplo, os centros dos olhos. Os autores introduzem também outro filtro, semelhante ao ASEF, o qual eles chamam de MOSSE (do inglês, *Minimum Output Sum of Squared Error*). A vantagem desse último filtro é que necessita de menos imagens de treinamento para atingir um bom desempenho. A Figura 1.15 ilustra os filtros utilizados para ajustes nos parâmetros de regularização de oito imagens de teste.

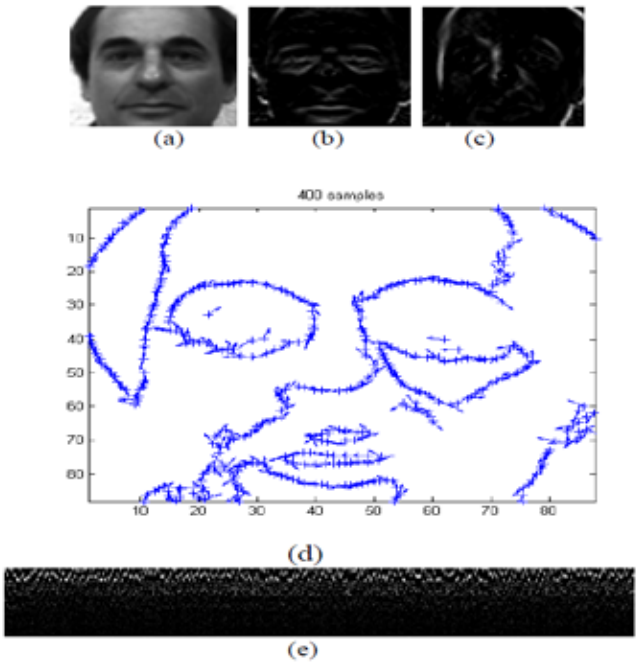
Figura 1.15: Nesta figura, todos os três filtros foram inicializados usando as mesmas oito imagens ao ajustar o parâmetro de regularização. Em 0: 1, todos os três filtros têm um PSR alto.



Fonte: BOLME; DRAPER; BEVERIDGE, 2009, p. 2547

Voltando a Figura 1.1, observamos que em 2010, existe uma continuação da utilização de métodos baseados em SIFT. São agora potencializados um conjunto mais denso de descritores e recebem o nome de Dense SIFT. O primeiro foi proposto por Wang et al. (2010) apresenta um método para identificação do sexo a partir de imagens faciais. Para tanto, combina descritores SIFT com o contorno de imagens faciais. No método proposto D-SIFT + Shape contexts (do inglês, *Dense SIFT + Shape Hape contexts*) os descritores são obtidos em uma densa grade regular (não só ao redor dos pontos-chave). Os autores perceberam que o contorno das faces possui informações discriminantes para o reconhecimento do sexo de uma pessoa. Dessa forma os descritores SIFT são concatenados em um vetor e combinados com os descritores de contorno. Para finalizar, utilizam o algoritmo AdaBoost para selecionar as características mais importantes (WANG et al., 2010). A Figura 1.16 ilustra esse processo.

Figura 1.16: Contextos de forma de uma imagem de face (a) imagem de face original (b) arestas horizontais (c) arestas verticais (d) 400 pontos de amostra desenhados a partir dos contornos (e) contextos de forma (60 × 400) da imagem de face, n- a coluna corresponde aos histogramas do n-ésimo ponto do total de 400 pontos

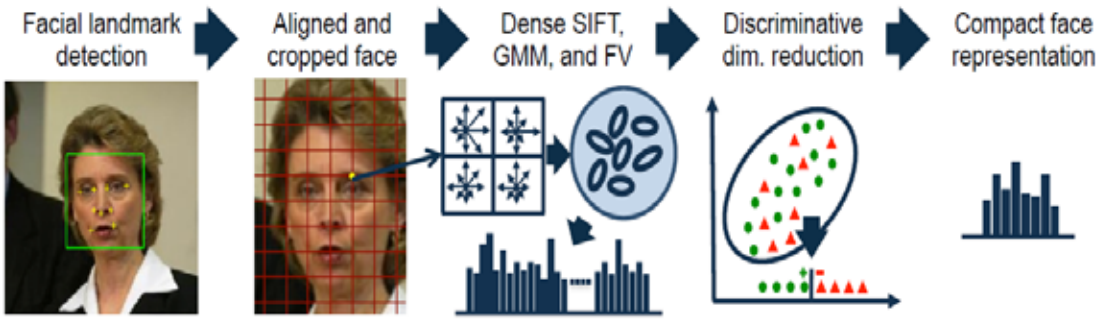


Fonte: WANG et al., 2010, p. 99

Um outro trabalho apresentado pelos autores Li, Park e Jain (2011) propõe um modelo discriminativo para o reconhecimento facial com certa tolerância a variação de envelhecimento, utilizando para isso o método SIFT combinado com o MLBP (do inglês, Multi-Scale Local Binary Pattern). Os descritores SIFT são calculados de maneira densa e uma extensão do LDA é usada para reduzir a dimensão dos vetores de características.

Por fim, um outro trabalho relevante que utilizou a ideia de Dense-SIFT foi proposto pelos autores Simonyan et al. (2013). Eles propõem um método de reconhecimento facial baseado em SIFT e FV (do inglês, *Fisher Vector*), conforme visão geral apresentada na Figura 1.17.

Figura 1.17: Visão geral do método de reconhecimento facial SIFT+FV



Fonte: SIMONYAN et al, 2013, p. 3

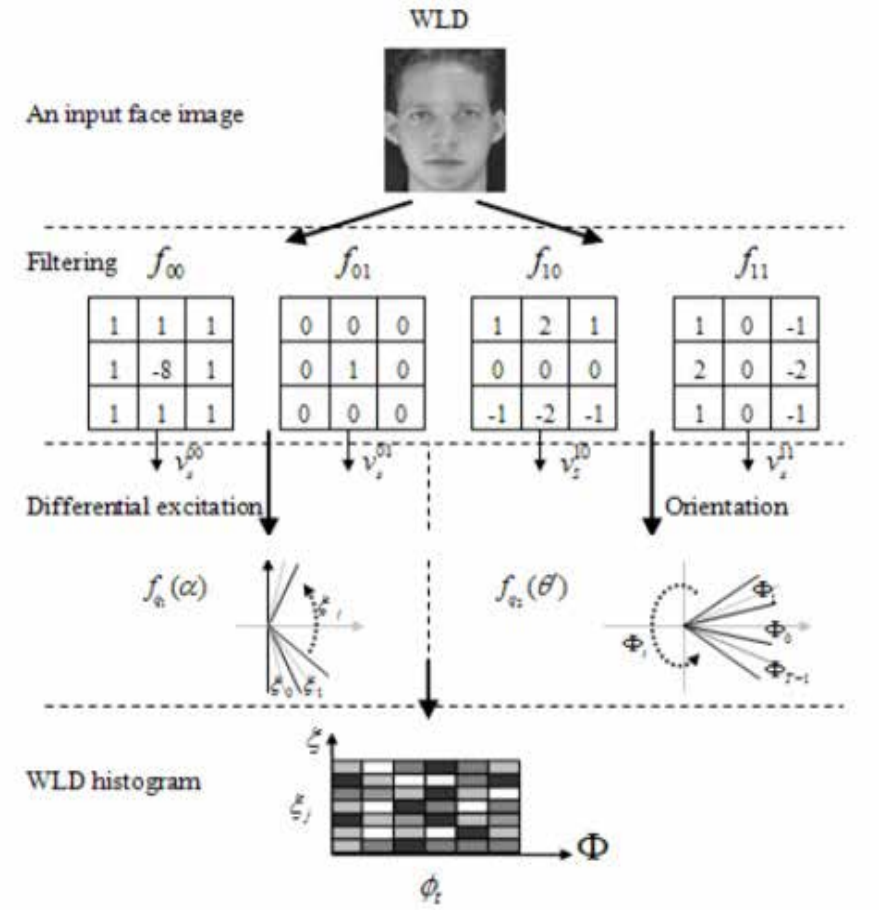
O método é baseado na descrição dos pixels em grades regulares por meio da aplicação de SIFT denso, na escala e no espaço, utilizam-se apenas os descritores. Em seguida, as características são codificadas em vetor FV. Finalmente, um algoritmo reduz a discriminante da dimensão dos FVs, o que melhora o desempenho do processo de reconhecimento, baseado na projeção dos dados em um subespaço. O método foi testado utilizando a base de imagens LFW e obteve melhores resultados quando comparado com outros métodos.

Retornando a Figura 1.1, dando prosseguimento a linha do tempo, um conjunto de trabalhos utilizam um outro método de reconhecimento facial conhecido como WLD (do inglês, *Weber Local Descriptor*).

Um dos primeiros trabalhos que utilizaram esse método foi o proposto pelos autores Gong, Li e Xiang (2011). No artigo os autores apresentam esse método (WLD), onde uma imagem de uma face, já detectada, é dividida em um conjunto de regiões e as características de cada região são extraídas pelo WLD.

A Figura 1.18 apresenta o cálculo do WLD a partir de uma face.

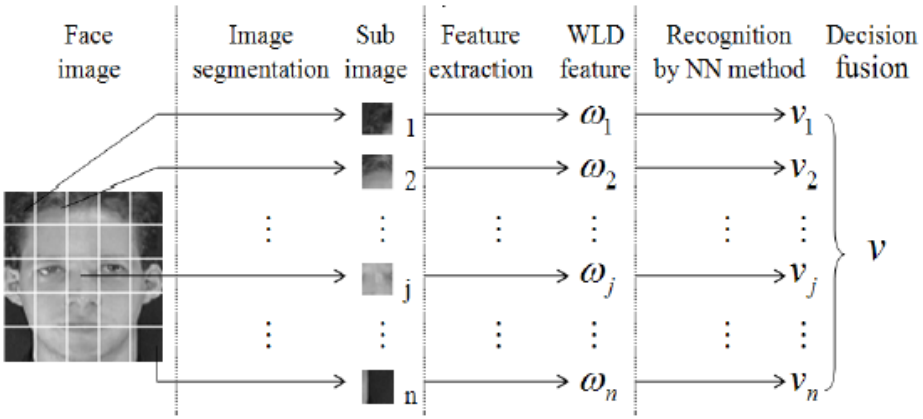
Figura 1.18: Cálculo do WLD a partir da entrada de uma imagem



Fonte: Adaptado de GONG; LI; XIANG, 2011, p. 590.

Os autores indicam que no método proposto, a orientação do gradiente é calculada utilizando-se o operador de Sobel, tornando assim o método mais imune ao ruído da imagem de entrada. Como pode ser observado na Figura 1.19, o reconhecimento da imagem é feito calculando-se a distância Euclidiana das características WLD de todas as regiões da imagem de teste com as imagens do banco de dados e aplica-se o método do vizinho mais próximo.

Figura 1.19: Método de reconhecimento utilizando WLD



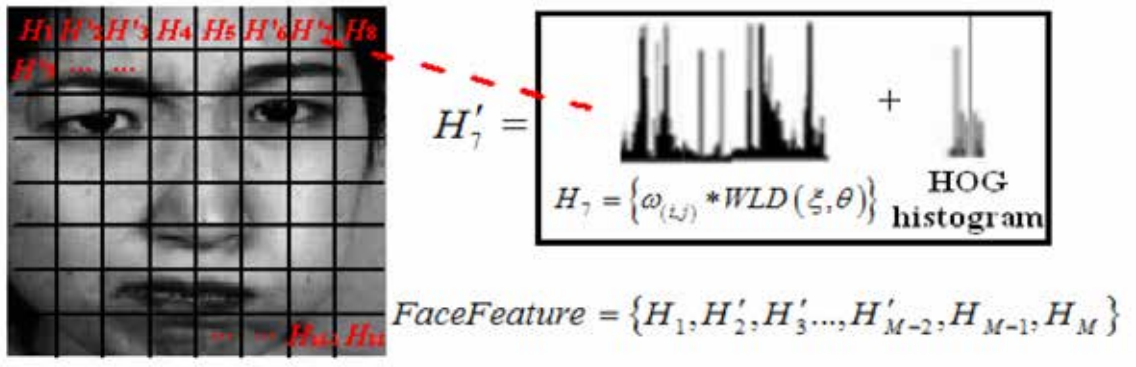
Fonte: GONG; LI; XIANG, 2011, p. 591.

Com o resultado obtido em cada região e utilizando-se uma lógica de decisão baseada na quantidade de regiões identificadas corretamente, a imagem é então classificada como um todo. Segundo os autores, o método WLD foi testado utilizando-se as bases de dados ORL e YALE e os resultados indicam uma acurácia melhor que o método LBP, com maior robustez para variações de iluminação, poses e expressões faciais.

O método WLD também foi utilizado nos trabalhos de Ullah et al. (2012), Wang et al.(2013) e de Suaishi; Yaun e Keping(2014) para o reconhecimento de expressões faciais.

Especificamente em Wang et al.(2013) as características obtidas com o descritor são combinadas com aquelas obtidas pelo HOG (do inglês, *Histograms of Oriented Gradients*). A Figura 1.20 ilustra essa fusão dos dois métodos: WLD e HOG.

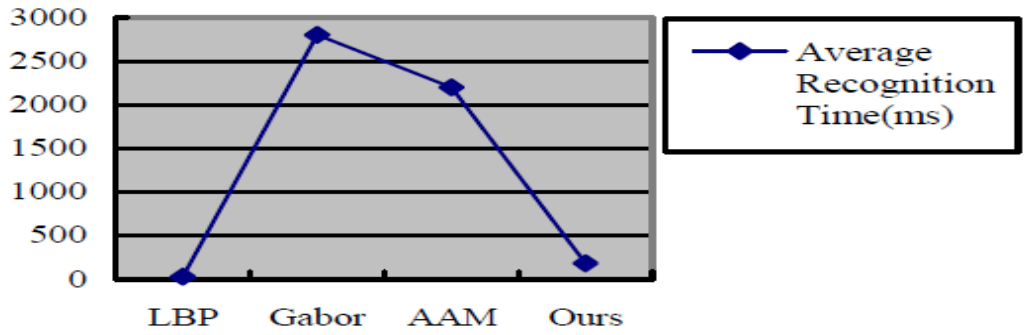
Figura 1.20: O processo de extração dos recursos WLD e HOG. Toda a face é representada por esses histogramas em cascata



Fonte: WANG et al., 2013, p. 230

Um dos principais ganhos obtidos por essa combinação entre WLD e HOG foi a redução do tempo médio de reconhecimento, sem perder a qualidade do processo (compatível com outros métodos analisados). A Figura 1.21, ilustra a comparação do tempo de reconhecimento entre os métodos analisados. O método classificado como *Ours* (nosso) representa o método resultante da combinação WLD e HOG.

Figura 1.21: A comparação do tempo de reconhecimento entre LBP, Gabor, AAM e o método proposto (WLD+HOG).



Fonte: WANG et al., 2013, p. 231

Já o trabalho dos autores Suaishi, Yaun e Keping (2014) utilizam apenas o WLD aplicado em sub-regiões da imagem, com a utilização do classificador SVM.

Assim, terminamos essa primeira fase dos principais métodos e algoritmos que foram criados e utilizados no período de 1973 a 2012. Detacam-se, nesta primeira etapa, os seguintes métodos/algoritmos: EigenFaces e Fisherfaces, Haar-Cascade, SIFT e HOG. Três desses principais: EigenFaces, Haar-Cascade e HOG, serão objeto de aprofundamento nos capítulos 6, 7 e 8, respectivamente.

Vamos agora nos deter aos métodos e algoritmos da segunda parte da história do reconhecimento facial: de 2012 até os nossos dias!

2

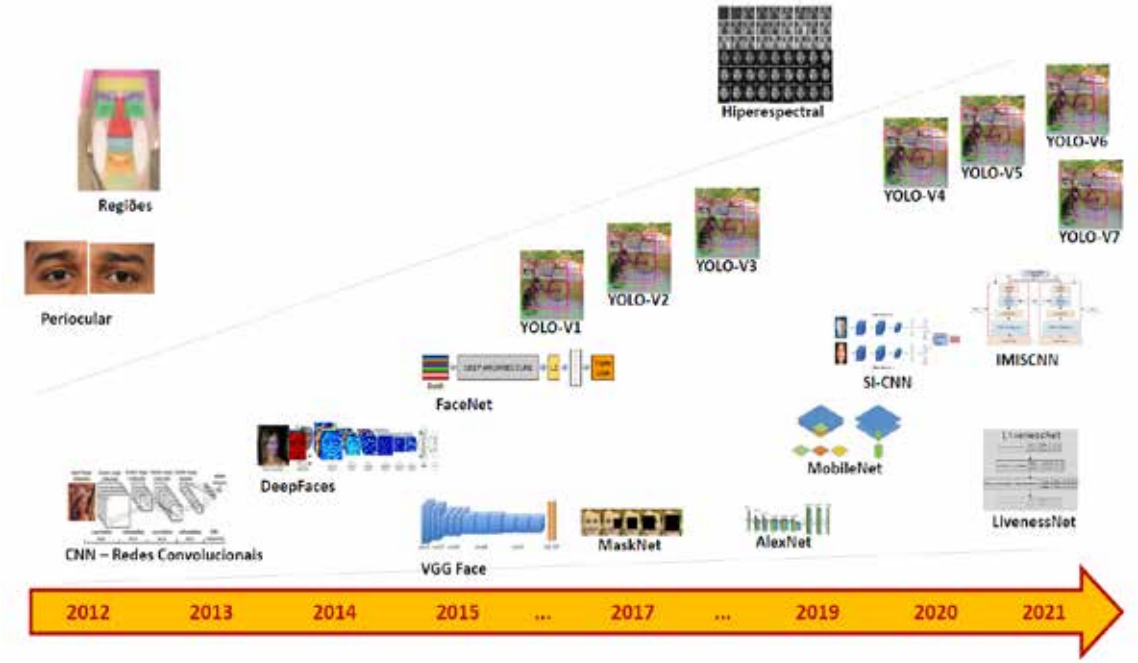
Os Principais Métodos e Algoritmos de Detecção e Reconhecimento Facial
Parte 2: de 2012 aos dias atuais

Nesta segunda parte da história dos principais métodos e algoritmos de detecção e reconhecimento facial o foco se concentra na utilização massiva das redes neurais e suas variações.

Além de aumentar a complexidade, a acurácia dos métodos e algoritmos cresceram enormemente, chegando em alguns casos a ultrapassar a acurácia humana!

A Figura 2.1 apresenta um resumo dos métodos e algoritmos que apresentaremos neste capítulo.

Figura 2.1: Principais métodos de reconhecimento facial da Segunda Parte da linha do tempo (2012 aos dias atuais)



Ao observarmos a Figura 2.1, notamos o ano inicial é 2012 e, nesta segunda parte, passamos às abordagens mais recentes que tratam da utilização do reconhecimento facial com o emprego, essencialmente, de redes neurais (*neural networks*) e redes neurais profundas (*deep neural networks*).

Um dos conceitos centrais, além de redes neurais artificiais (RNA), é o conceito de uma rede neural específica, conhecida como Rede Neural Convolucional (do inglês, *Convolutional Neural Network* - CNN).

As bases conceituais das CNN foram inicialmente cunhadas em 1989 por Yann le Cun (LE CUN, 1989), que trabalhou em um algoritmo para reconhecimento da escrita a mão. De lá para cá, com o aumento do poder computacional das duas últimas décadas, o método proposto por Le Cun (1989) pode então ser utilizado em outras aplicações e contextos.

As redes convolucionais são um tipo especializado de rede neural para o processamento de dados que possui uma topologia conhecida como grade. Os exemplos incluem dados de séries temporais, que podem ser pensados como uma grade 1-D, colhendo amostras em intervalos regulares, e dados de imagem, que podem ser considerados como uma grade 2-D de pixels (GOODFELLOW et al, 2016).

As redes convolucionais têm sido tremendamente bem-sucedidas em aplicações práticas. Atualmente, algumas das suas principais aplicações são: carros autônomos, equipamentos de reconhecimento de faces, e reconhecimento de escrita à mão, dentre muitas outras voltadas para o reconhecimento de padrões.

O nome “rede neural convolucional” indica que a rede emprega uma operação matemática chamada convolução. *Convolução é um tipo especializado de operação linear. As redes convolucionais são simplesmente redes neurais que usam a convolução no lugar da multiplicação geral da matriz em pelo menos uma de suas camadas*.

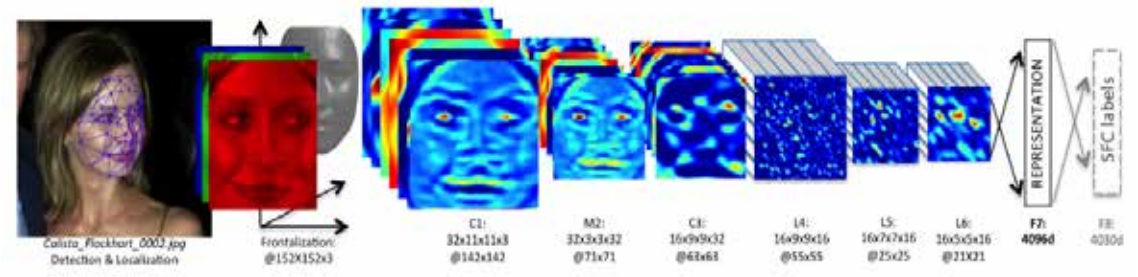
O processo de convolução e as próprias redes neurais convolucionais (CNN) serão detalhadas no capítulo 9.

Retornando a Figura 2.1, logo em seguida ao método CNN temos os métodos conhecidos como DeepFace, que também vão utilizar a abordagem de redes neurais profundas com CNN. Um dos primeiros trabalhos com essa abordagem foi dos autores Taigman et al. (2014) onde apresentaram um método de reconhecimento facial que emprega uma rede neural profunda de 9 camadas, com mais de 120 milhões de parâmetros, para a extração de características e classificação das imagens faciais.

A Figura 2.2 mostra o fluxo de convoluções, a partir da imagem de entrada até a sua representação. Na imagem é possível verificar que os autores utilizam no método um *front-end* de uma única filtragem de convolução-associação-convolução na entrada retificada, seguido por três camadas localmente conectadas e duas camadas totalmente conectadas.

As cores ilustram os mapas de recursos produzidos em cada camada. Como já dito, a rede inclui mais de 120 milhões de parâmetros, dos quais mais de 95% são provenientes das camadas local e totalmente conectada.

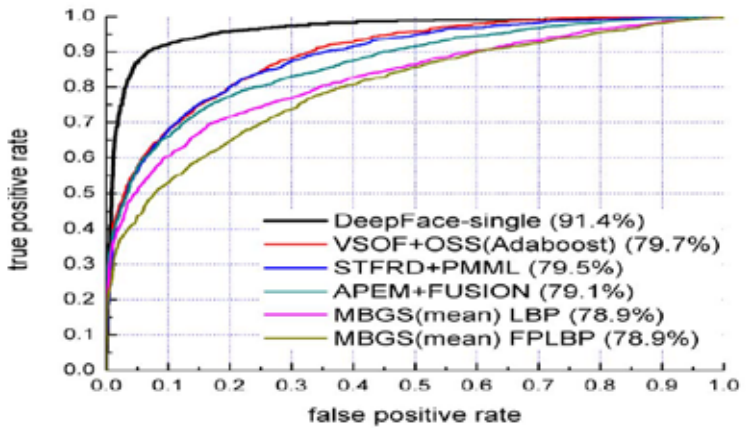
Figura 2.2: Esboço da arquitetura do método DeepFace



Fonte: TAIGMAN et al., 2014, p. 1704.

Este método foi testado nas bases LFW (*Labeled Faces in the Wild*) e YTF (*YouTube Faces*) e obteve uma acurácia de 97,35% no modo irrestrito. A Figura 2.3 apresenta a curva ROC (do inglês *Receiver Operating Characteristic*) ilustrando o desempenho do método DeepFace utilizando a base YTF, comparativamente com outros métodos tradicionais.

Figura 2.3: Curva ROC do desempenho do método DeepFace na base YTF

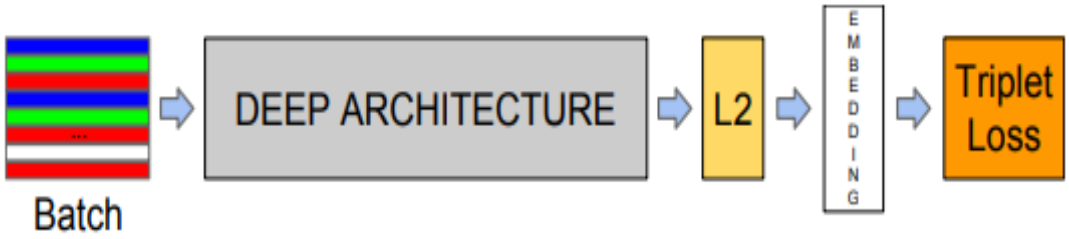


Fonte: TAIGMAN et al., 2014, p. 1708.

O método/algoritmo DeepFaces será detalhado no capítulo 10.

Em 2015, os autores Schroff, Kalenichenko e Philbin (todos funcionários do Google) publicaram um artigo apresentando uma rede neural convolucional profunda que chamaram de FaceNet. A Figura 2.4 apresenta a estrutura do modelo do FaceNet proposta pelos autores.

Figura 2.4: Estrutura do Modelo FaceNet: A rede consiste de uma camada de entrada em lote e uma CNN profunda, seguida de normalização L2, que resulta na incorporação de faces. Isto é seguido pela perda tripla durante o treinamento



Segundo os autores, além de um modelo mais otimizado, o processo gerava uma maior eficiência representacional, com apenas 128 bytes por face, o modelo chegou a uma eficiencia superior a 99% (LFW – 99,63% e YouTube Faces – 95,12%), reduzindo em até 30% a taxa de erro dos melhores resultados até esse momento publicados.

Em 2016, os autores Redmon et al (2016) lançaram um algoritmo/método detector de objetos ao qual nomearam de YOLO (*You Only Look Once – Você só observa uma vez*). Além de reconhecer qual objeto está presente em uma determinada imagem, também identifica em qual posição está localizado. Este método é composto de dois processos: classificação e localização. A Figura 2.5 apresenta a junção desses dois processos na função de detecção.

Figura 2.5: Função de Detecção do YOLO, formado pelos processos de Classificação e Localização

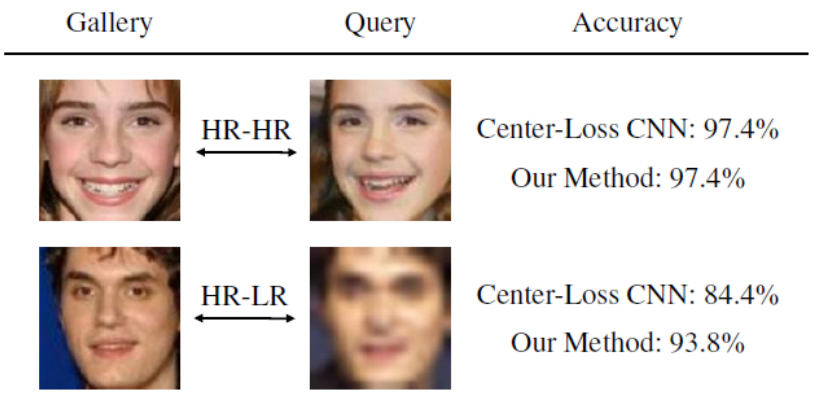
A versão 1 (YOLO V1) foi lançada em 2016 como dito anteriormente. Posteriormente, os autores e demais contribuidores, foram atualizando e aperfeiçoando o método, lançando as versões YOLO V2 (2017), YOLO V3(2018), YOLO V4(2020), YOLO V5 (2021), YOLO V6 (2022) e YOLO V7 (2022).

No capítulo 11, este método/algoritmo será detalhado com mais profundidade.

Ainda nessa vertente de utilização de Redes Neurais Profundas (Deep Neural Networks), temos uma infinidade de trabalhos propondo abordagens com algumas variações na organização da rede neural profunda ou então a combinação de redes neurais com outros métodos de aprendizagem de máquina (*machine learning*).

Os autores Fu et al. (2017) sugeriram a CNN guiada para resolver problemas com reconhecimento facial de resolução cruzada. Seu projeto sugerido aprende o modelo paralelo nas BR (Baixa Resolução) com funções especiais de perda, avançando o modelo CNN pré-treinado em imagens de Alta Resolução (AR) como referência. A Figura 2.6 Ilustra essa relação entre imagens de Alta e Baixa resolução e a relação da precisão de outros métodos com o da CNN com perda-central.

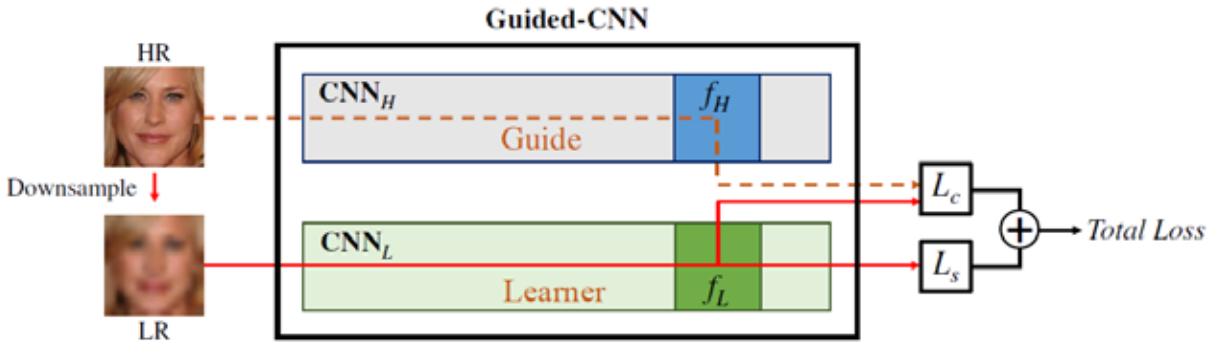
Figura 2.6: Desafio de reconhecimento facial de resolução cruzada.



Embora o desempenho promissor seja relatado para CNNs recentes no conjunto de dados LFW (por exemplo, Center-Loss CNN), ele não generaliza bem se a imagem da consulta estiver com resolução insuficiente. Observe que HR e LR denotam resoluções altas e baixas, respectivamente. Fonte: (Fu et al., 2017, p.1)

Dentro e entre as resoluções de imagem, as funções de perda implementadas permitem maximizar conjuntamente a semelhança das fotos. A Figura 2.7 ilustra a arquitetura proposta de CNN-Guiada para reconhecimento de face com resolução cruzada.

Figura 2.7: A arquitetura proposta de CNN guiada para reconhecimento facial de resolução cruzada.

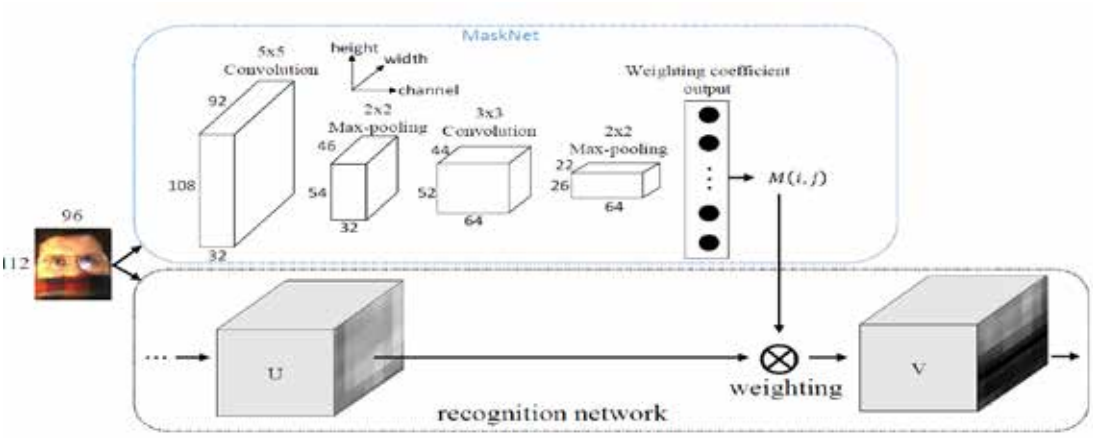


Durante o treinamento, foi reduzida a resolução das imagens HR para serem as entradas LR do CNNL. A função de perda Softmax Ls é aplicada para aprender as informações de identificação de imagens LR e a perda única de domínio cruzado Lc associa as representações de recursos de imagens LR às correspondentes HR. Na fase de teste, simplesmente inserimos HR / LR no CNNH / CNNL respectivamente e calculamos as semelhanças usando a distância do cosseno dos recursos associados fH / fL. Fonte: (Fu et al., 2017, p.2)

Os autores confirmaram a partir de seus testes que seu sistema supera vários métodos básicos e recentes de reconhecimento de faces com resolução cruzada, e uma extensão robusta de reconhecimento de rosto também foi testada com sucesso, com resultados acima de 97,1% para imagens com alta resolução, 93,8 com imagens de baixa resolução e 93% com imagens com oclusão de até 50%.

Os autores Wan & Chen (2017) propuseram um módulo MaskNet que pode ser construído com as arquiteturas atuais da CNN. MaskNet descobre uma maneira adequada de produzir de forma adaptativa várias máscaras de mapa de recursos para várias imagens de rosto ocluído com instruções supervisionadas de ponta a ponta apenas pelas marcas de identidade pessoal. A Figura 2.8 apresenta a arquitetura da MaskNet.

Figura 2.8: A arquitetura do MaskNet incluída na rede de reconhecimento.



Todas as camadas de ativação são omitidas para simplificar. A saída da camada totalmente conectada do MaskNet é remodelada para o tamanho espacial de U. Então, a máscara M é calculada com uma função de ativação. A unidade de ponderação multiplica cada peso na máscara M com as características de U na mesma localização espacial. V é o mapa de recursos após a ponderação. Toda a rede é otimizada com treinamento ponta a ponta usando apenas o rótulo de identidades pessoais. Fonte: (Wan & Chen, 2017, p. 3796)

Intuitivamente, o MaskNet sinaliza dinamicamente pesos maiores para as unidades secretas acionadas pelas partes faciais que não estão ocluídas e pesos menores para aquelas ativadas pelas partes faciais que estão ocluídas. A Figura 2.9 apresenta exemplos de imagem com 40, 50, 60 e 70% de oclusão.

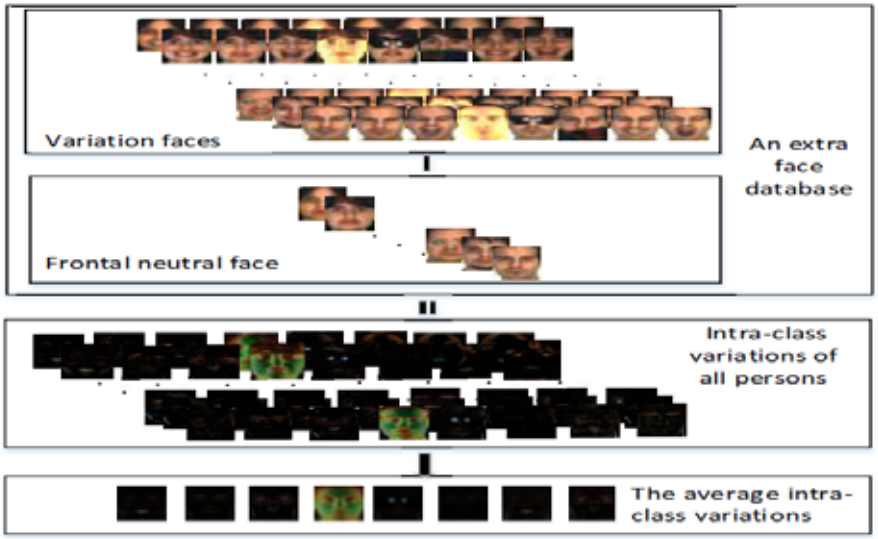
Figura 2.9: Exemplo de imagens de teste LFW com oclusão sintética.

O mais à esquerda é a imagem original seguida por imagens com tamanhos de blocos aleatórios n = 40, 50, 60, 70 respectivamente. (Wan & Chen, 2017, p. 3798)

Testes de conjunto de dados consistindo em faces oclusas simuladas e reais para revelar que o MaskNet pode efetivamente aumentar a robustez dos modelos CNN no reconhecimento facial contra oclusões.

Os autores Zeng et al. (2017) sugeriu uma nova abordagem para resolver o problema de reconhecimento facial (RF) com uma amostra única por pessoa (Single Sample Per Person - SSPP). Em primeiro lugar, para SSPP-RF, uma nova abordagem de expansão de amostra é proposta. A Figura 2.10 apresenta o framework de construção da variação do conjunto intraclasse.

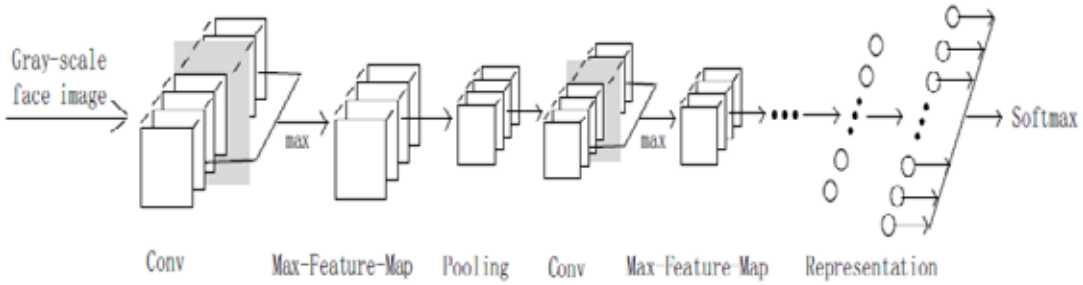
Figura 2.10: Framework de construção da variação do conjunto intraclasse.



Fonte: (Zeng et al., 2017, p. 1648)

Em segundo lugar, ele implementa um modelo bem treinado (DCNN) e, em seguida, as amostras de expansão são usadas para ajustar o modelo (DCNN). A Figura 2.11 apresenta a arquitetura da rede CNN mais leve (lightened CNN)

Figura 2.11: arquitetura da rede CNN mais leve (*lightened CNN*).



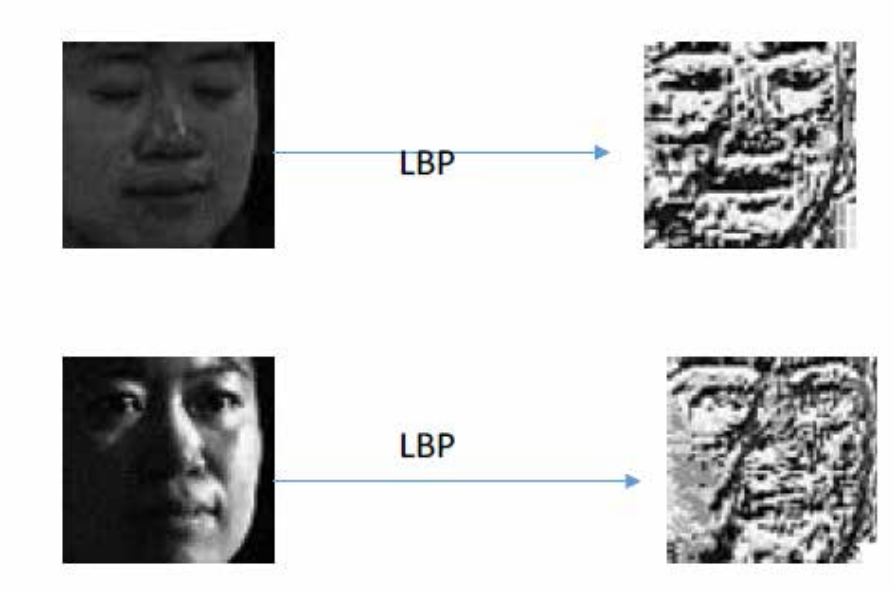
Fonte: (Zeng et al., 2017, p. 1648)

Ainda neste trabalho, os autores utilizaram o banco de dados AR face[1], usado para testar a precisão do SSPP-RF no modelo DCNN ajustado. Os resultados experimentais mostram que o método proposto (SSPP-RF) atinge algumas taxas de reconhecimento de iluminação e expressão mais altas do que o segundo (DCNN) e também atinge a maior precisão em todas as imagens do banco de dados AR face. Os resultados demonstram que o método proposto apresenta um melhor desempenho no SSPP em comparação ao que utiliza amostra única para o ajuste fino do modelo DCNN.

No ano seguinte, os autores Ke et al. (2018) investigam um algoritmo de reconhecimento facial baseado na combinação de LBP e CNN. O operador LBP foi usado para obter uma imagem codificada de padrão binário local. A Figura 2.12 ilustra essa codificação.

1 AR Face database: Banco de Dados de Face AR. Uma base de dados com 126 pessoas com mais de 4.000 imagens coloridas, criadas por Aleix martinez e Robert Benavente (AR) na Universidade Estadual de Ohio (Ohio State University). O endereço da base de dados é: https://www2.ece.ohio-state.edu/~aleix/AR-database.html

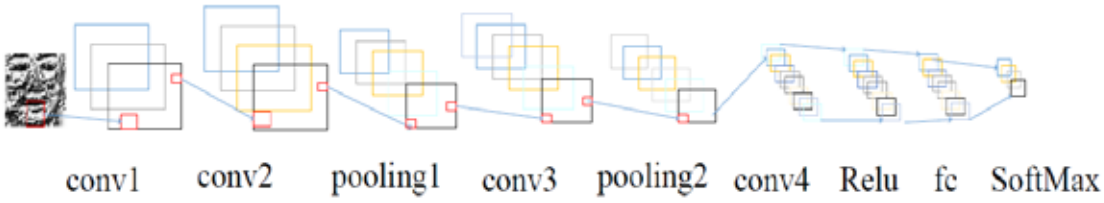
Figura 2.12: LBP codificando a imagem.



Fonte: (Ke et al., 2018, p. 540)

Em segundo lugar, as fotos foram usadas para praticar a CNN (Figura 2.13).

Figura 2.13: Estrutura da Rede Neural CNN.

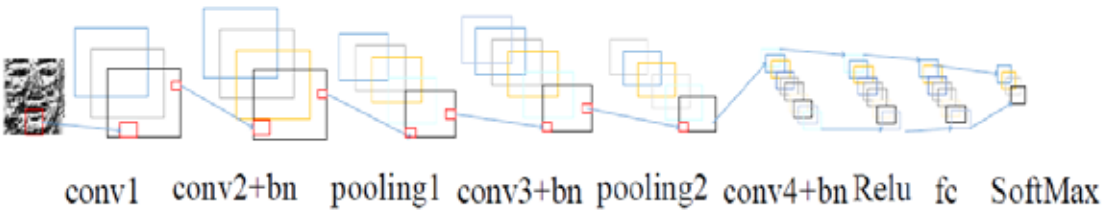


Fonte: (Ke et al., 2018, p. 541)

O banco de dados de faces CMU-PIE foi usado como referência. Experimentos mostraram que a seleção do tamanho do lote pode ter um efeito na taxa de identificação e que o tamanho do lote ideal é 100 para esta rede. A rede original pode aumentar substancialmente a taxa de identificação do rosto. A maior precisão é 97,65 por cento.

Para melhorar a taxa de identificação, a rede foi configurada com a introdução de três camadas de padronização de lote (Figura 2.14)

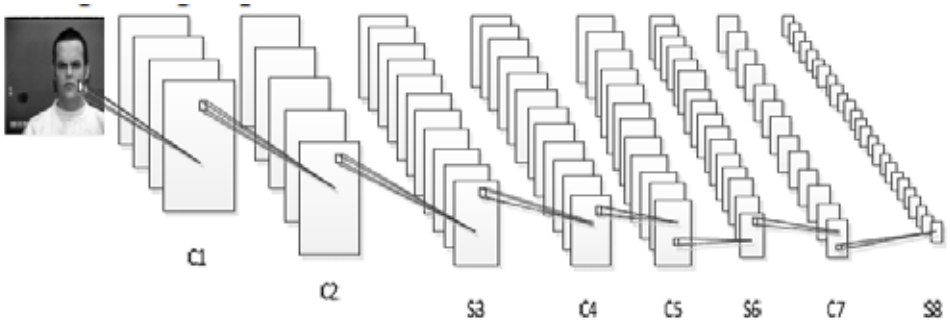
Figura 2.14: Estrutura da Rede Neural CNN otimizada.



Fonte: (Ke et al., 2018, p. 542)

No mesmo ano, os autores Wang et al. (2018) propuseram um sistema baseado em redes convolucionais profundas (DCNN) para detecção de expressão facial. o artigo usa um modelo de rede neural convolucional profunda para derivar características faciais e usa o classificador Softmax para identificar expressões faciais. A Figura 2.15 apresenta a estrutura da rede DCNN.

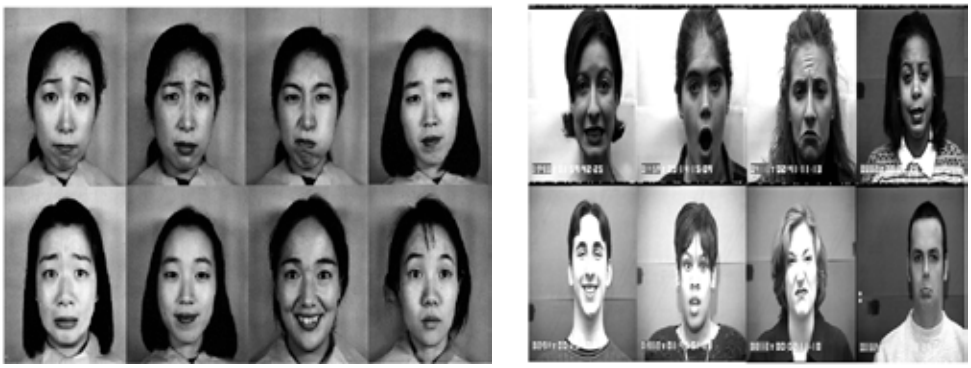
Figura 2.15: Diagrama da Estrutura da DCNN.



Fonte: (Wang et al., 2018, p. 1)

O algoritmo utilizado não necessita de intervenção humana na aprendizagem supervisionada e oferece um método automático de extração de características para que o efeito encontrado seja maior. Eles conduziram e compararam testes nos conjuntos de dados JAFFE e CK + com outras abordagens. A Figura 2.16 apresenta exemplos de imagens das duas bases de dados (JAFFE e CK+).

Figura 2.16: Base de dados de Imagens utilizadas pelos autores.



(a) (b)

(a) JAFFE com 213 imagens com 10 mulheres e 7 expressões faciais e (b) CK+ com 593 imagens com 123 sujeitos, com as expressões etiquetadas. Fonte: (Wang et al., 2018, p. 3)

Os resultados experimentais indicam que esta abordagem é muito mais bem-sucedida do que outros métodos de extração manual características faciais de reconhecimento facial. Redes neurais convolucionais profundas podem conduzir o reconhecimento de expressões faciais de maneira eficaz.

No ano seguinte, os autores Wu et al. (2019) sugeriram uma nova estratégia focada em Aprendizagem Profunda (DL) para classificar a face ocluída. O algoritmo de reconhecimento facial é treinado e aprende com base na rede neural de convolução DL, que possui alta robustez à variação de luz, mudança na expressão facial e oclusão facial. A Figura 2.17 apresenta a estrutura do sistema proposto pelos autores (semelhante a FaceNet – 2016).

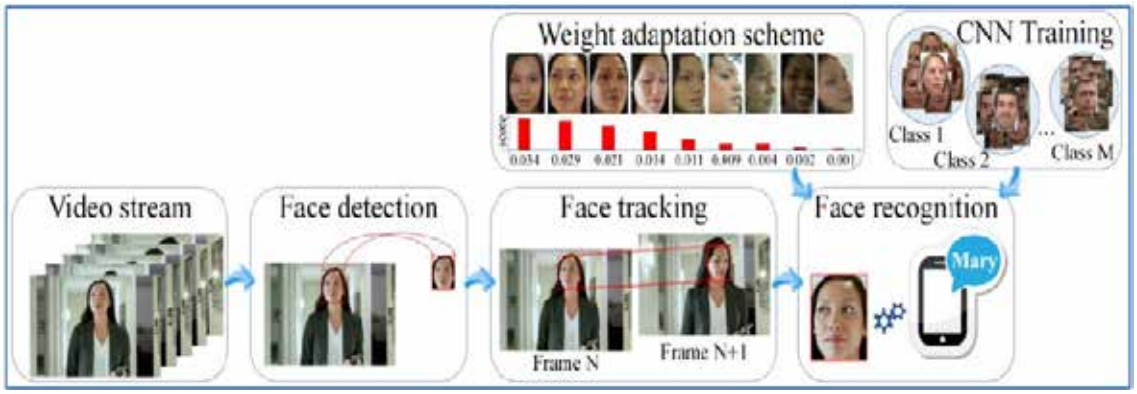
Figura 2.17: Estrutura do sistema

Fonte: (Wu et al., 2019, p. 795)

A pontuação de reconhecimento facial ocluída chegou a 98,6% por meio de uma vasta gama de experimentos de laboratório e análises de resultados. O trabalho de Wu et al. considera a detecção facial de oclusão em diversos ambientes e satisfaz os critérios de implementações funcionais.

O trabalho dos autores Mocanu et al. (2019) introduziu uma tecnologia de reconhecimento facial baseada em redes neurais convolucionais (CNN) para auxiliar pessoas cegas. O sistema tem como objetivo potencializar o engajamento e contato de pessoas cegas em encontros sociais. A Figura 2.18 apresenta a arquitetura proposta pelos autores.

Figura 2.18: Arquitetura proposta para o sistema



Fonte: (Monacu et al., 2019, p. 1)

O modelo VGG16 e o conjunto de dados ImageNet são usados. A precisão de seu sistema é de 90% quando verificado em 30 fontes de vídeo.

Já os autores Perdana & Prahara (2019) sugeriram uma Rede Neural Convolucional Leve (Light-CNN) para identificar faces com um pequeno conjunto de dados baseado em um modelo VGG16 modificado. A Figura 2.19 apresenta a arquitetura VGG16 original.

Figura 2.19: Arquitetura da VGG-16 original.

Fonte: (Perdana e Prahara, 2019, p.1)

O VGG16 tem camadas muito profundas com muitas camadas estreitas de convolução com números de kernels separados, seguidos por camada de agrupamento máximo (max-pooling) otimizado para classificação em grande escala.

A arquitetura planejada usa como imagens de entrada com tamanho de (120 X 120) pixels e tem dois tipos de camadas convolucionais, seguidas pela camada de agrupamento máximo (max-pooling). Cada camada convolutiva é precedida pelo recurso da função de ativação linear retificada (ReLU). A Figura 2.20 apresenta o modelo de Rede Neural Convolucional Leve (Light-CNN) proposta pelos autores.

Figura 2.20: Arquitetura Light-CNN proposta

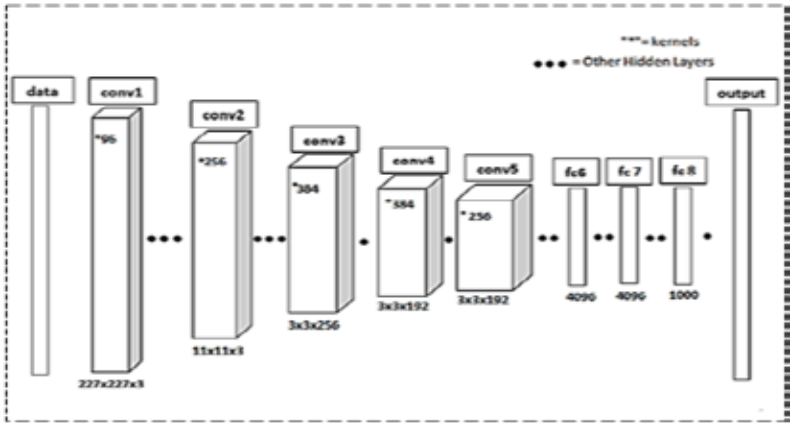
Layer Type	Kernel	Size	Stride
Conv1	64	3 x 3	1 x 1
Max Pooling	-	2 x 2	1 x 1
Conv2-1	128	3 x 3	1 x 1
Conv2-2	128	3 x 3	1 x 1
Conv2-3	128	3 x 3	1 x 1
Max Pooling	-	2 x 2	1 x 1
FC	512	-	-
FC	30	-	-
Soft-max	1	-	-

Fonte: (Perdana e Prahara, 2019, p.2)

O modelo light-CNN proposto é pequeno, mas oferece boa eficiência com 94,4% de precisão, segundo seus autores.

O trabalho dos autores Khan et al. (2019) propôs um sistema de reconhecimento de face usando uma Rede Neural Convolucional (CNN) que é a AlexNet, que utiliza um paradigma de aprendizado profundo (DL) com muitas camadas. A Figura 2.21 apresenta as camadas da rede AlexNet.

Figura 2.21: Camadas da AlexNet

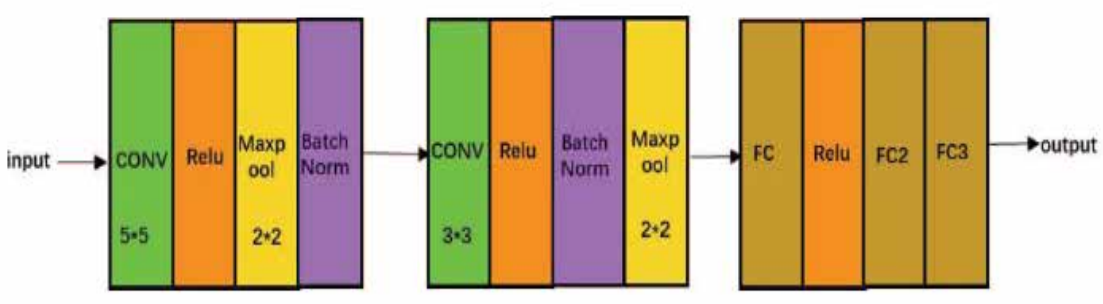


Fonte: (Kan et al., 2019, p.3)

Eles também conduziram o aprendizado de transição desta rede. A rede foi então treinada em seus servidores. Essa rede acabou sendo usada para reconhecimento facial. Essa rede precisa de um amplo banco de dados para treinamento, mas a precisão é boa. Foram utilizadas 4 mil imagens para o treinamento, divididas em 4 classes. Foram utilizados como solução para treinamento, o número de épocas igual a 20 e o tamanho do lote igual a 10. Com esses parâmetros, a precisão alcançada pela rede treinada foi de 97,95 por cento. Para comportar essa estrutura, os autores utilizaram o MATLAB.

Liu (2019), em seu trabalho, sugeriu o sistema de reconhecimento de expressões faciais (REF) usando o conjunto de dados FER2013 e uma rede neural convolucional profunda, eficaz para treinar um modelo eficiente. A Figura 2.22 apresenta a arquitetura da rede neural convolucional profunda utilizada pelo autor.

Figura 2.22: Arquitetura da CNN Profunda utilizada.



Fonte: (Liu, 2019, p.222)

Em seguida, o autor usou a ferramenta Tkinter para desenvolver a interface gráfica do usuário (GUI) do sistema, utilizada para avaliar a imagem de expressão e alcançar um desempenho realista. Além disso, AlexNet, VGG16, VGG19 e ResNet152 foram utilizadas para realizar os testes. A precisão final da pesquisa dos testes foi de 15,2%, 37,4%, 39,68% e 48,67%, respectivamente, nessas redes, conforme ilustra a Tabela 2.1.

Tabela 2.1: Comparação de resultados

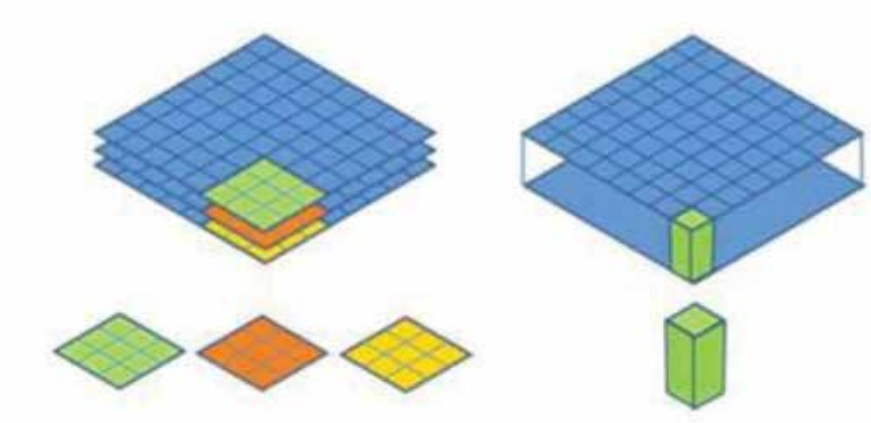
method	Test accuracy
AlexNet	15.2%
VGG16	37.4%
VGG19	39.68%
Resnet152	48.67%
Proposed network	49.8%

Fonte: (Liu, 2019, p.223)

Os autores Zhou et al. (2019) sugeriram um modelo de CNN baseado na fusão de recursos locais e globais da MobileNet para permitir um bom uso das informações de recursos de cada camada de imagem, uma vez que a CNN padrão não fundiria as informações de camadas convolucionais altas e baixas na fase de treinamento de fotos de rosto.

A Figura 2.23 apresenta a diferença entre os filtros de convolução em profundidade e os filtros de convolução pontuais.

Figura 2.23: Filtros de convolução em profundidade e Filtros de convolução pontuais

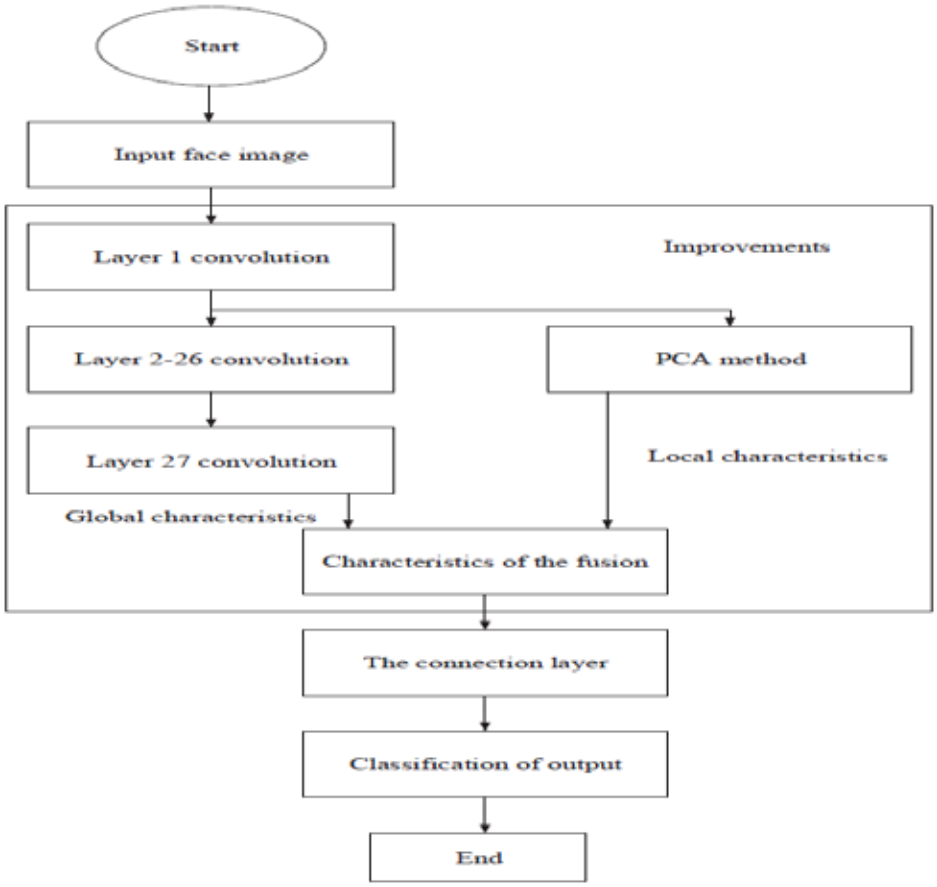


(a) Filtros em Profundidade (b) Filtros Pontuais

Fonte: (Zhou et al., 2019, p. 2772)

Após a análise do componente principal (PCA), o algoritmo funde os recursos locais da primeira camada da rede com os recursos globais. A Figura 2.24 apresenta a arquitetura melhorada pela presente proposta.

Figura 2.24: Arquitetura MobileNet aprimorada



Fonte: (Zhou et al., 2019, p. 2773)

Portanto, a expressão de recursos superficiais é aprimorada, o efeito de extração de recursos profundos é aprimorado e os dados recuperados pela MobileNet aprimorada são mais precisos. A Tabela 2.2, apresenta os resultados dos testes realizados pelos autores com quatro diferentes modelos.

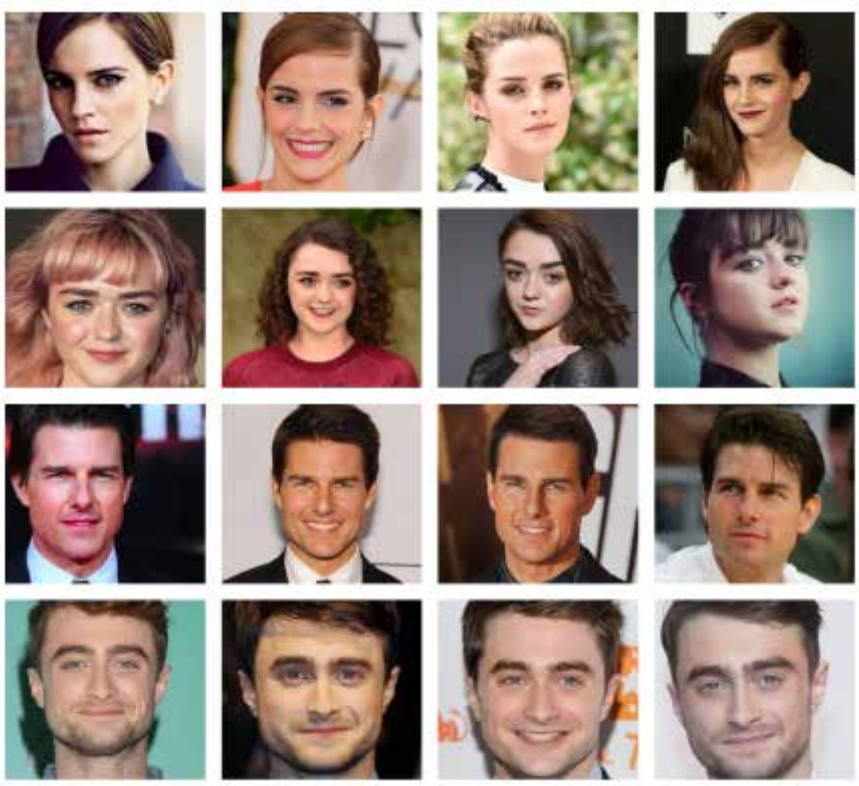
Tabela 2.2: Comparação da precisão entre os modelos

Model	Model parameters/M	Accuracy of CASIA-Web Face data set/%	Accuracy of LFW data set /%
SqueezeNet	4.8	93.12	93.57
ShuffleNet	4.0	94.28	95.13
MobileNet	4.2	95.12	95.78
Improved MobileNet	4.2	95.87%	96.80

Fonte: (Zhou et al., 2019, p. 2774)

O trabalho dos autores Ahmed et al. (2020) propôs uma análise comparativa usando modelos baseados em CNN, como VGG16, VGG19, MobileNet e AlexNet para capturar as faces de um conjunto de dados personalizado de 10 identidades de celebridades. A Figura 2.25 apresenta um exemplo dessas imagens.

Figura 2.25: Exemplo de imagens de celebridades utilizada no estudo



Fonte: (Ahmed et al., 2020, p. 2)

Com a implementação da Transferência de Aprendizagem e do Ajuste Fino, foram então utilizados esses modelos pré-treinados no conjunto de dados ImageNet. Os autores utilizaram o TensorFlow com backend da API Keras escrito em Python em seu experimento. A revisão de desempenho envolve preparação, avaliação e verificação em várias imagens iniciais do conjunto de dados. A precisão da validação do modelo VGG19 é considerada maior do que os outros três, mas a melhor precisão do teste foi mostrada pelo modelo MobileNet. A Tabela 2.3 apresenta esses resultados.

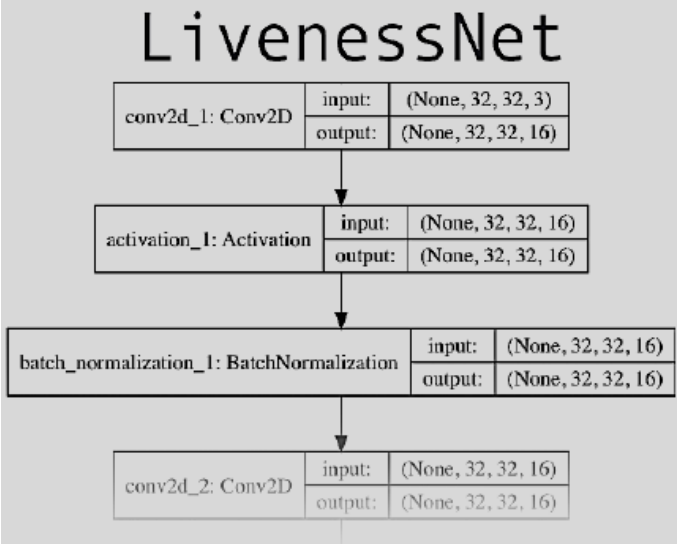
Tabela 2.3: Comparação entre os modelos

Model Name	Maximum Training Accuracy	Maximum Validation Accuracy	Test Accuracy
AlexNet	99.8%	64.5%	57%
VGG16	100%	86%	71%
VGG19	99.8%	87%	56%
MobileNet	100%	85%	84%

Fonte: (Ahmed et al., 2020, p. 3)

Os autores Jafri et al. (2020) identificaraam uma abordagem de reconhecimento facial baseada em redes neurais profundas para testar a detecção de rosto humano com livenessNet. A Figura 2.26 apresenta a arquitetura LivenessNet.

Figura 2.26: Arquitetura LivenessNet para detecção de faces



Fonte: (Jafri et al., 2020, p. 147)

O modelo proposto, procura a posição e as dimensões de todos os recursos pertencentes a uma classe chamada face. O maior valor dessa técnica é que ela é extremamente precisa e pode ser realizada em tempo real. A biblioteca OpenCV é usada para a aplicação desta tecnologia de reconhecimento facial usando redes neurais profundas. O princípio primário do paradigma de detecção é a detecção de rosto frontal. O reconhecimento facial como FPAF (Foco Automático com Prioridade ao Rosto) também é representado. O maior valor dessa técnica é que ela é extremamente precisa e pode ser realizada em tempo real. A Figura 2.27 apresenta um exemplo da aplicação identificando a face real e a falsa.

Figura 2.27: Resultado de imagem processado, mostrando rosto real e falso

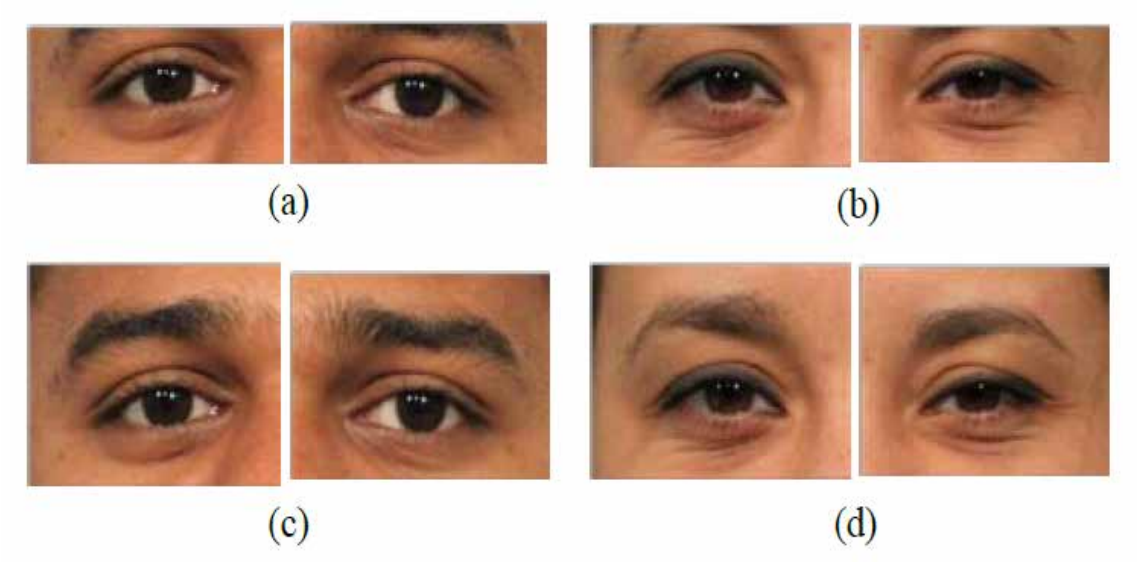
Fonte: (Jafri et al., 2020, p. 148)

E muitos outros trabalhos foram propostos nesta mesma linha. Alguns deles para finalizar: Liu et al. (2020) propõe uma arquitetura de rede neural FER (Face Expression Recognition ou Reconhecimeto de Expressões Faciais) baseada em uma operadora CNN aprimorada por Sobel e uma rede L2-SVM combinada; Ruan et al. (2020) sugeriram um método de escalar os valores de ativação de seus processos de treinamento e teste, que é propício para determinar o modelo de uma vez, e então, ajustando sua probabilidade de retenção / abandono, o processo de treinamento e teste será feito no mesmo modelo; Xu et al. (2020) projetaram um sistema de rede neural convolucional profunda generalizada que tem alta robustez para reconhecimento de face sob condições ilimitadas (IMISCNN); Yao (2020) sugeriu uma rede neural convolucional compacta para reconhecimento facial.

A maioria dos métodos analisados até aqui utilizam a abordagem holística, ou seja, consideram as características da face como um todo. Existem também métodos que se baseiam em características extraídas localmente como, por exemplo, os métodos baseados em SIFT. Um outro grupo de métodos, combina diversas regiões e características, ou mesmo extrai as características locais de uma maneira densa, ou seja, ao longo de toda a face, como ocorre em Simonyan et al.(2013).

Uma região da face que apresenta algumas pesquisas em aplicações de reconhecimento facial é a região periocular, pois é uma das regiões mais discriminantes de uma face (MILLER et al., 2010; JUEFEI-XU; LUU; SAVVIDES, 2015). A região periocular é aquela localizada na proximidade, na periferia, dos olhos e é relativamente fácil de ser capturada (PARK; ROSS; JAIN, 2009). No entanto, segundo esses autores "[...] não há uma definição formal para a região periocular, de modo que a mesma pode incluir ou excluir as sobrancelhas ou outros componentes.". A Figura 2.28 ilustra a região periocular de duas pessoas, sem e com sobrancelhas.

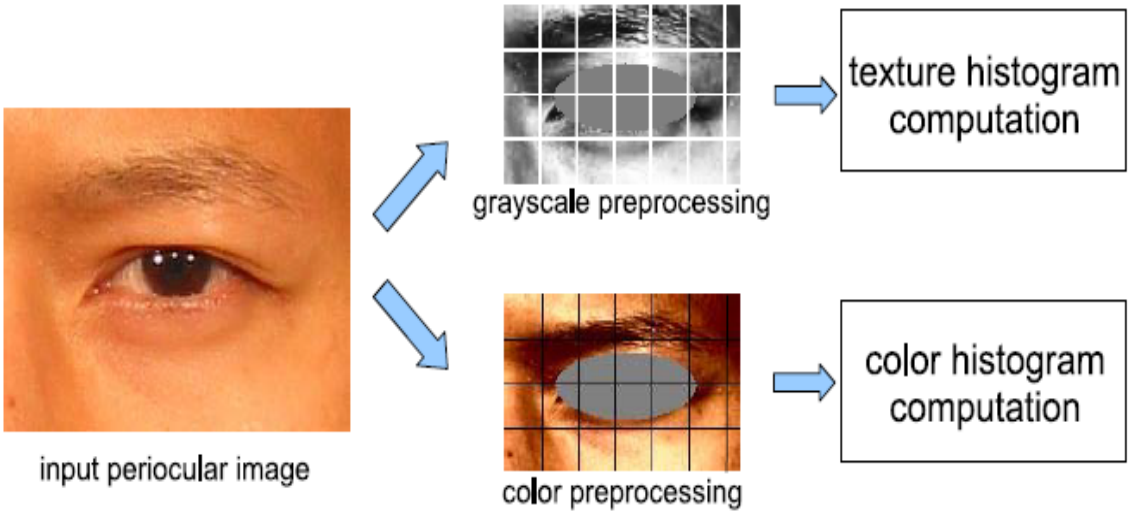
Figura 2.28 – Exemplos de região periocular: (a) e (b) sem sobrancelhas; (c) e (d) com sobrancelhas.



Fonte: PARK; ROSS; JAIN, 2009, p. 1

A taxa de reconhecimento obtida pela utilização da região periocular é comparável com a taxa obtida pela utilização da região facial como um todo, usando-se características locais similares em ambos os casos (WOODARD et al.,2010). Nesse trabalho os autores utilizam a análise da textura e coloração para montagem do histograma que servirá de comparação entre as imagens (no caso da região periocular). A Figura 2.29 ilustra essas etapas.

Figura 2.29: Etapas da montagem dos histogramas de textura e coloração da região periocular.

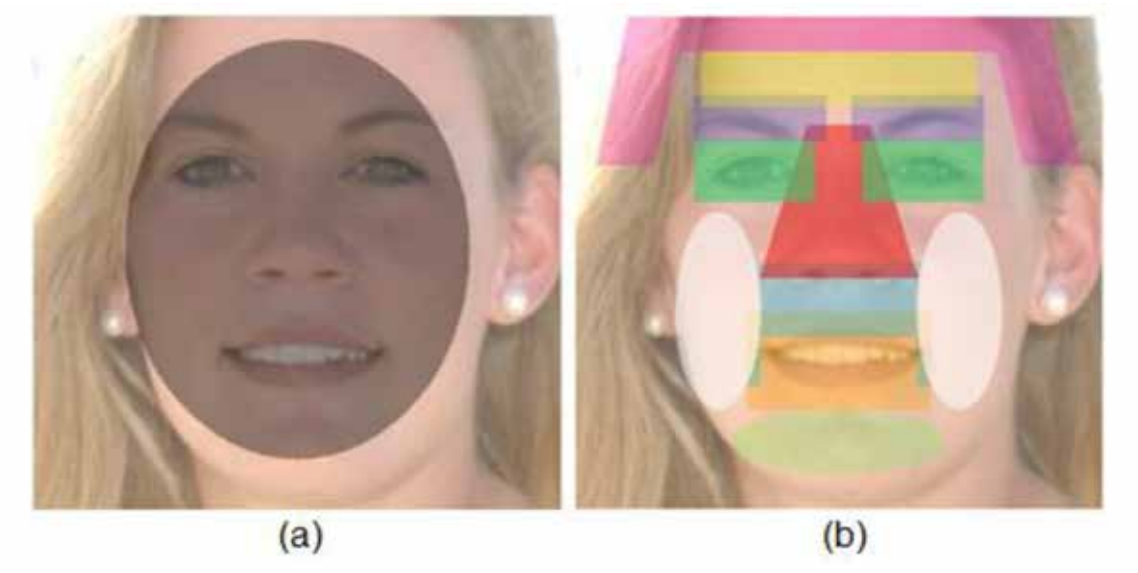


Fonte: WOODARD et al, 2010, p. 164

Existem casos em que a região periocular pode trazer melhores resultados que a abordagem holística.

Existem outras áreas utilizadas na face. Alguns trabalhos utilizam várias regiões, extraindo assim as características dessas várias regiões. Por exemplo, Kumar et al. (2011), dividem a face em nove regiões. A Figura 2.30 ilustra essas regiões.

Figura 2.30: Regiões utilizadas para reconhecimento facial: (a) face toda; (b) nove regiões locais

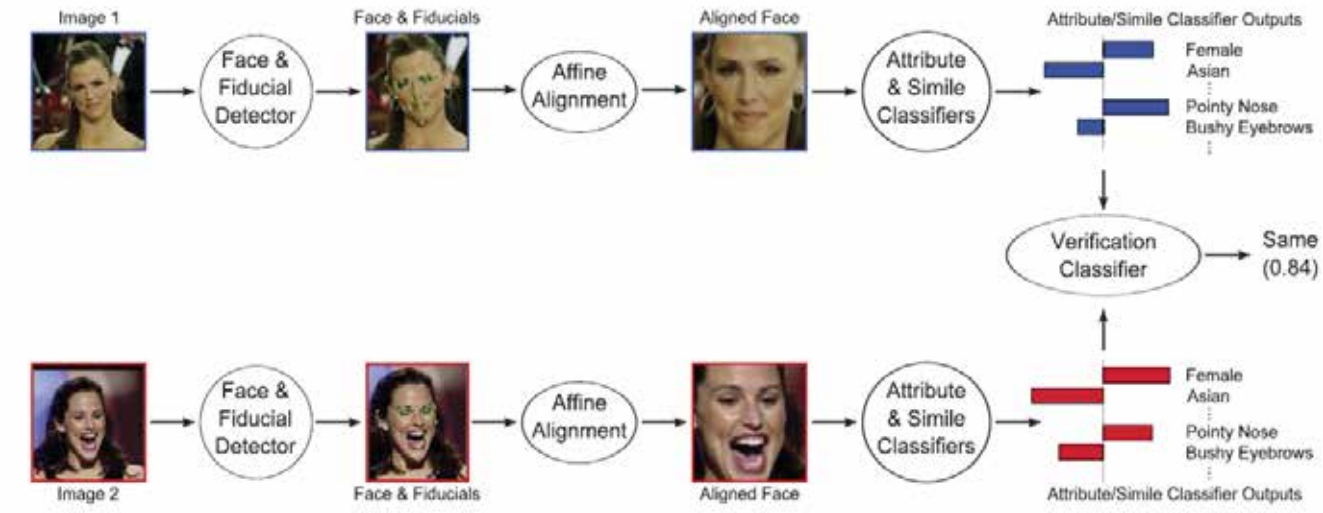


Fonte: KUMAR et al, 2011, p. 1969.

Na Figura 2.30 (a) mostra a abordagem holística, observando a face como um todo. Já na Figura 2.30 (b) são destacadas as nove regiões locais, as quais são utilizadas para extração de características locais. O que os autores observam em suas análises é que as regiões identificadas são relativamente grandes, o que contribui para minimizar os erros de alinhamento, diferenças entre indivíduos e variações nas poses.

A Figura 2.31 ilustra a abordagem das 9 regiões, mostrando, mesmo com grande variação nas duas imagens de uma mesma pessoa, o índice de similitude foi superior a 80%.

Figura 2.31: O processo de identificação de face em nove regiões.



Fonte: KUMAR et al, 2011, p. 1969.

Na Figura 2.31, as imagens são capturadas e alinhadas. As imagens de face alinhadas são alimentadas para cada um dos nossos atributos e classificadores de símile individualmente para obter um conjunto de valores de atributos. Por fim, esses valores são comparados usando um classificador de verificação para fazer a determinação da saída, que é retornada juntamente com a distância até o limite da decisão. Todo o processo é totalmente automático.

O trabalho dos autores Xie et al. (2019) propôs uma ideia para reconhecimento de face hiperespectral de banda bloqueada para maximizar a eficiência da preferência de banda. Neste trabalho, uma pequena rede CNN é treinada para capturar informações visuais discriminativas para diferentes blocos nas imagens das faces. em conhecimento espacial para obter características de discriminação. A Tabela 2.4 apresenta a estrutura da rede SI-CNN utilizada pelos autores.

Tabela 2.4: Estrutura da rede SI-CNN

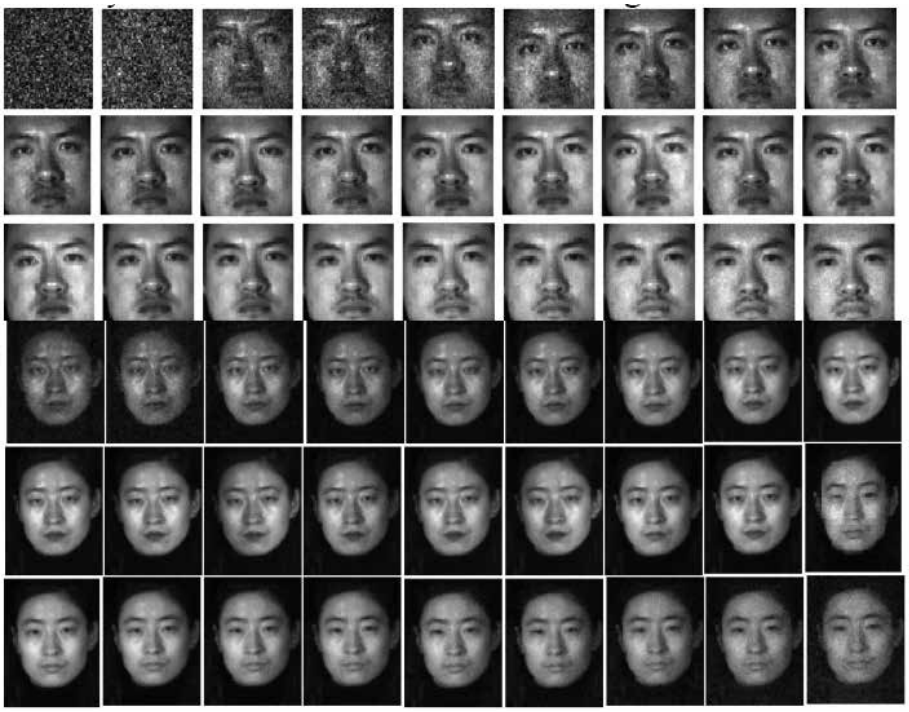
SI-CNN (5 weight layers)
Input(264x264 images)
Conv1(6x6x96 Stride=1)
Bnorm(96)
Maxpool(2x2)
Conv2(3x3x256 Stride=1)
Bnorm(256)
Maxpool(2x2)
Conv3(3x3x512 Stride=1)
Bnorm(512)
Maxpool(2x2)
Fc1(1024)
Dropdout（P =0.5）
Fc(25)
Softmax

Fonte: (Xie et al., 2019, p. 1271)

Em seguida, um algoritmo otimizador AdaBoost é introduzido para escolher diferentes as bandas mais otimizadas para diferentes blocos. Daí, cada bloco etiquetado pode ser classificado por aprendizagem ensemble.

Em todos os experimentos, uma base de dados hiperespectral da Universidade Politécnica de Hong Kong (PolyU-HSFD) foi utilizada. A Figura 2.32 apresenta uma amostra das imagens hiperspectrais dessa base de dados.

Figura 2.32: Faces hiperspectrais parciais da base de dados PolyU-HSFD



Fonte: (Xie et al., 2019, p. 1272)

Finalmente, os métodos de reconhecimento de face hiperespectral são comparados. A Tabela 2.5 apresenta a comparação dos resultados experimentais de diferentes métodos testados pelos autores.

Tabela 2.5: Comparação dos resultados experimentais de diferentes métodos

Methods	Accuracy
3D-DCT[9]	83%
Band fusion + PLS[10]	76%
Image-level LBP + SVM[11]	75%
Hyperspectral-SCNN +AdaBoostSVM[2]	54%
Blocking image SI-CNN+AdaBoost.M1	88%

Fonte: (Xie et al., 2019, p. 1273)

Os resultados do experimento indicam que o reconhecimento facial hiperespectral sugerido é preferível a outras abordagens, as bandas ótimas de cada bloco diferem e podem melhorar a identificação.

E com isso, fechamos essa segunda etapa, na linha do tempo dos métodos e algoritmos de Reconhecimento Facial, onde a utilização de Redes Neurais Profundas domina quase todo o período, sendo utilizada apenas a Rede Neural, com ou sem especialidades, e em alguns casos, combinadas com outros métodos de aprendizado de máquina para sua potencialização.

Muitos outros trabalhos não foram aqui citados, não pela importância, mas sim, pela necessidade de otimização e objetividade na elaboração deste capítulo.

Os destaques aqui são as CNN (redes convolucionais – 2012, que melhor discutiremos no capítulo 9), DCNN (redes convolucionais profundas), as DeepFaces 2014 (que ampliaremos a discução no capítulo 10), com destaque para os modelos ou arquiteturas: VGG-Face (2015), FaceNet (2015), SqueezeNet (2016), YOLO 2016 (que abordaremos com mais detalhes no capítulo 11), AlexNet (2019), MobileNet (2019), SI-CNN (2019), IMISCNN (2020) e LivenessNet (2020).

No capítulo seguinte abordaremos uma forma de classificação dos métodos de detecção e reconhecimento facial.

3

Classificação de Métodos e Algoritmos de Detecção e Reconhecimento Facial

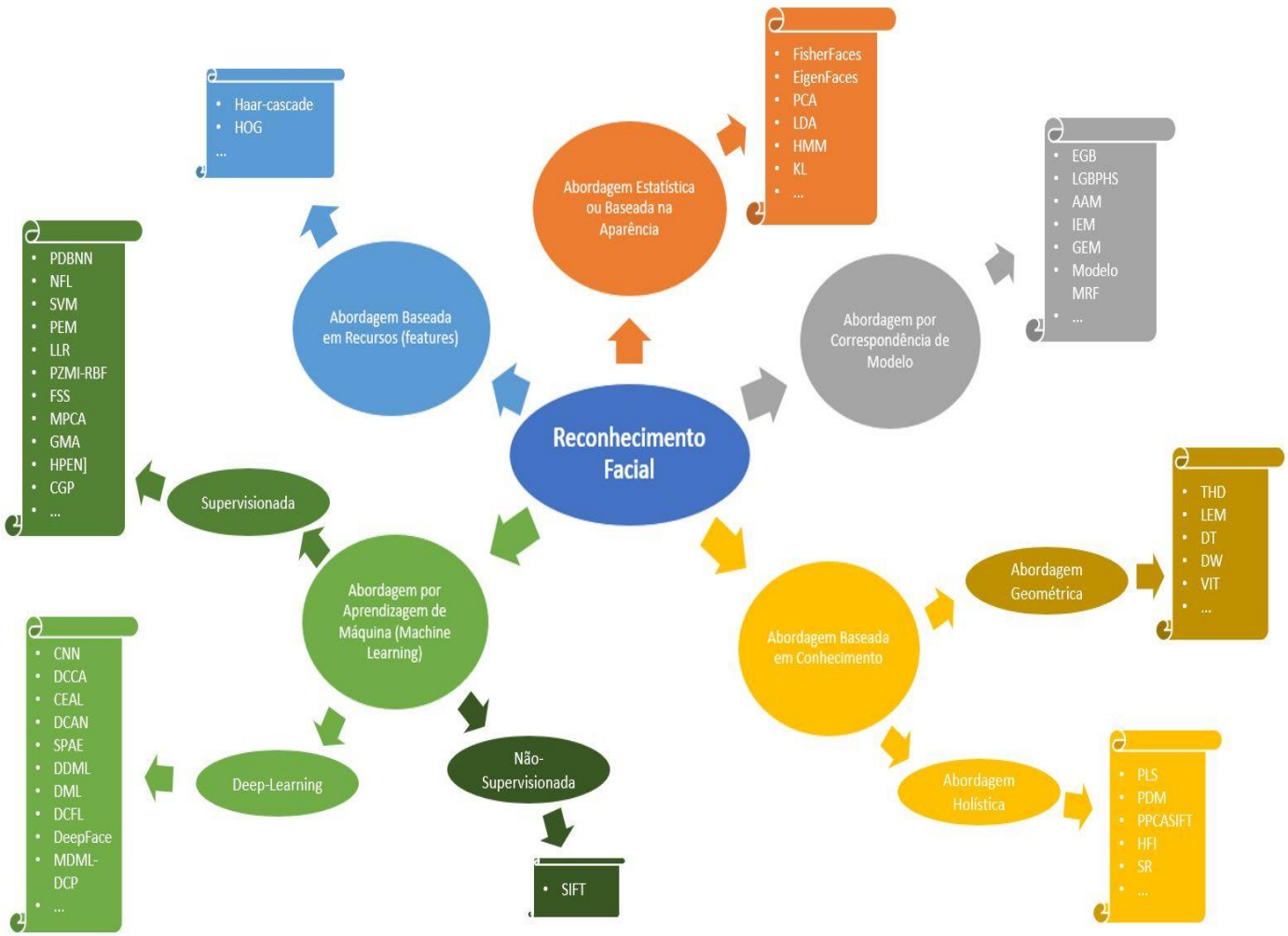
Como já vimos, existe um processo que a grande maioria dos algoritmos e métodos de detecção e reconhecimento facial segue: Primeiro, é realizado uma análise da foto ou uma imagem de vídeo que tenta distinguir rostos (faces) de quaisquer outros objetos no fundo (Detecção Facial). Logo em seguida, uma vez identificado as faces, existe a consulta a uma base de dados de faces registradas para verificar se existe alguma cujas características (features) sejam semelhantes (Reconhecimento Facial).

Existe uma infinidade de algoritmos e métodos que um computador (desktop, notebook, celular, tablet etc.) pode usar para fazer isso, compensando a iluminação, orientação ou distância da câmera.

Esses algoritmos ou métodos são geralmente agrupados em categorias ou classes. Existem muitas classificações para esses algoritmos e métodos de detecção/reconhecimento de faces.

A Figura 3.1 apresenta uma compilação de diversas classificações encontradas na literatura (YANG; KRIEGMAN; AHUJA, 2002; AHMED et al, 2019; ANWARUL; DAHIYA, 2019; ADJABI et al, 2020; DEBEY; JAIN, 2019; FIROZE; DEB, 2018 apenas para citar algumas).

Figura 3.1: Classificação dos métodos de detecção/reconhecimento facial



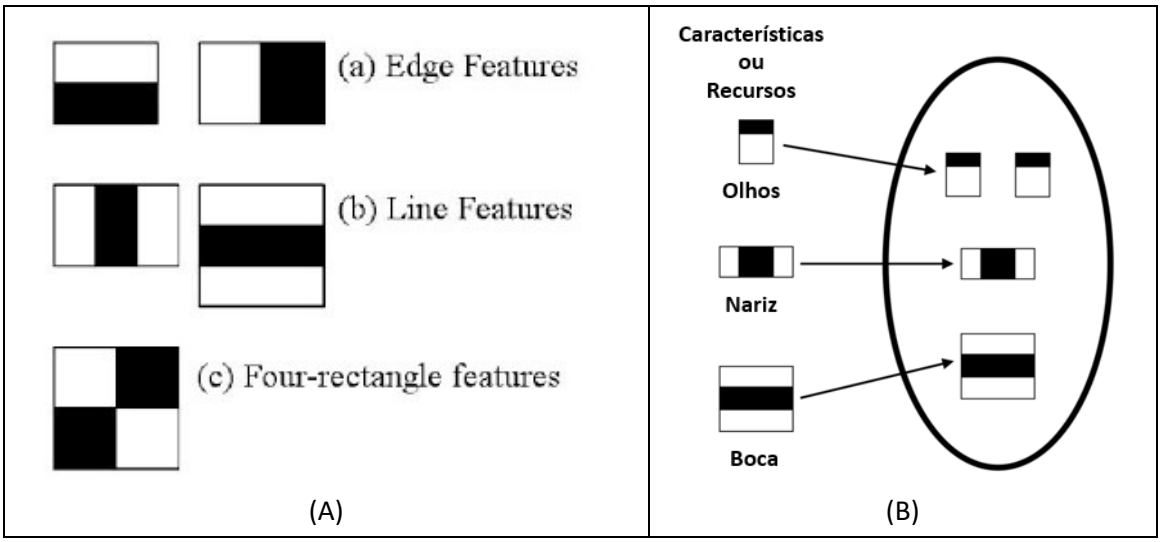
Vamos ver, com um pouco mais de detalhe, cada uma dessas cinco principais classes de algoritmos / métodos de detecção e reconhecimento facial.

3.1. Detecção de face Baseada em Características ou Recursos (*features*)

A classe de métodos baseados em recursos, extrai recursos estruturais da face. É treinado como um classificador e depois usado para diferenciar as regiões faciais das não faciais. Um exemplo desse método é a detecção de rosto com base em cores, que verifica imagens ou vídeos coloridos em busca de áreas com a cor de pele típica e, em seguida, procura segmentos de rosto.

A seleção de recursos Haar se baseia em propriedades semelhantes de rostos humanos para formar correspondências a partir de características faciais: localização e tamanho do olho, boca, ponte do nariz e gradientes orientados de intensidades de pixel. Existem 38 camadas de classificadores em cascata para obter o número total de 6061 recursos de cada face frontal. Existem diversos classificadores *Haar-Cascade* treinados. A Figura 3.2A apresenta alguns dos filtros Haar utilizados para se chegar aos classificadores. A Figura 3.2B apresenta um modelo teórico buscado pelos classificadores para identificação de uma face frontal.

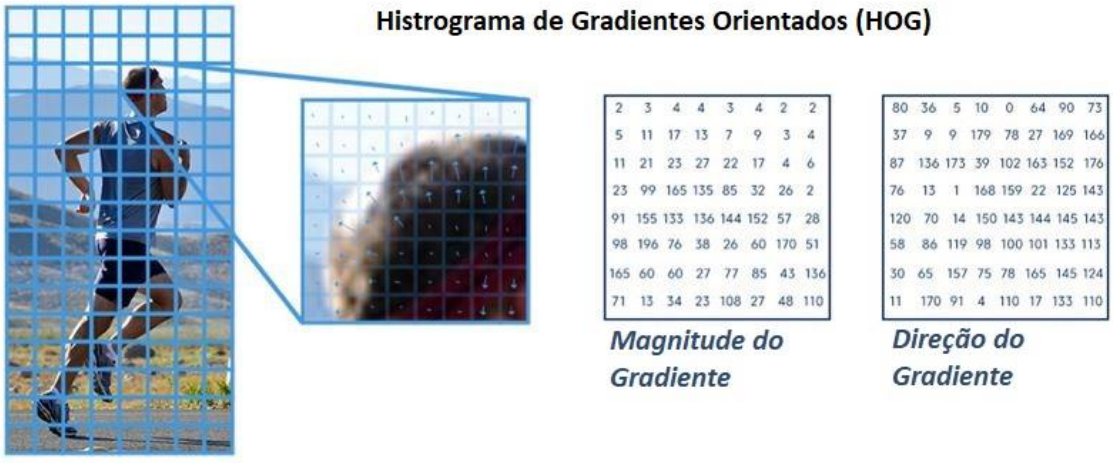
Figura 3.2: (A) Filtros Haar utilizados para classificação. (B) Modelo teórico buscado pelos classificadores p/ identificação de uma face frontal.



Fonte: Adaptado de Viola e Jones, 2001.

Já o Histograma de Gradientes Orientados (*Histogram Oriented Gradients* - HOG) é um extrator de recursos para detecção de objetos. Os recursos extraídos são a distribuição (histogramas) das direções dos gradientes (gradientes orientados) da imagem. A Figura 3.3 apresenta esse processo de extração de recursos utilizando HOG.

Figura 3.3: Extração de recursos utilizando HOG

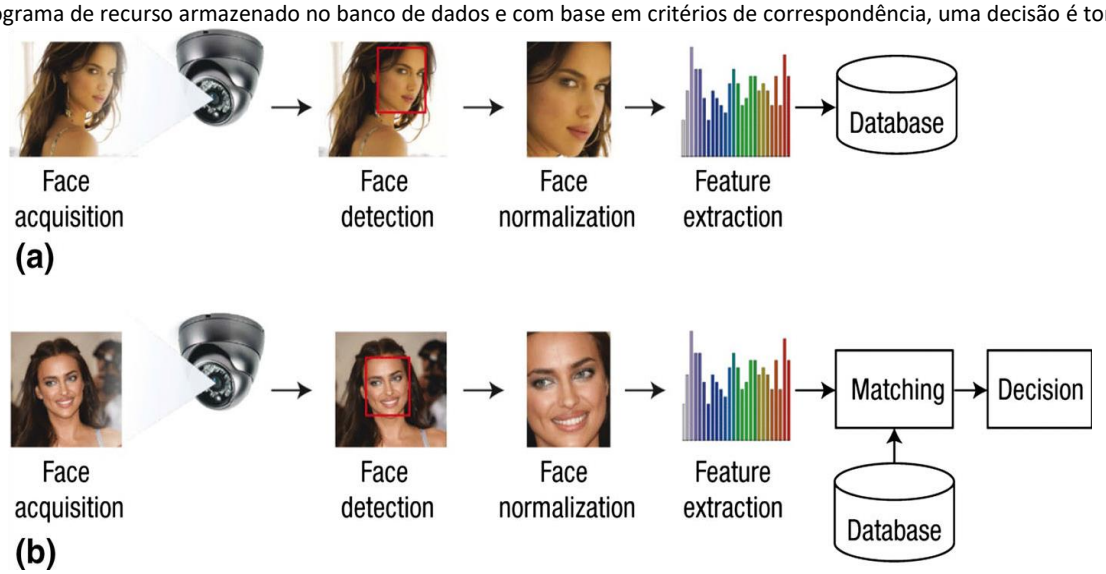


Os gradientes são normalmente grandes bordas e cantos arredondados e nos permitem detectar essas regiões. Em vez de considerar as intensidades de pixel (como é o caso do *Haar-Cascade*), eles contam as ocorrências de vetores de gradiente para representar a direção da luz para localizar segmentos de imagem. O método usa normalização de contraste local sobreposta para melhorar a precisão.

3.2. Detecção de face Baseada na Aparência ou Abordagem Estatística

Abordagens baseadas em aparência ou estatísticas são aquelas abordagens que lidam com técnicas estatísticas (histograma, gráfico de barras, modelos etc.) para reconhecimento de face, como mostrado na Figura 3.4.

Figura 3.4: Abordagem baseada em aparência ou estatísticas: Em (a), o procedimento básico de extração de recursos é feito e armazenado em banco de dados em forma de histograma. Em (b), procedimento semelhante é seguido, mas um novo histograma de recurso é comparado com o histograma de recurso armazenado no banco de dados e com base em critérios de correspondência, uma decisão é tomada.



Fonte: (Akhtar; Rattani 2017, p.81)

O método mais avançado baseado em aparência ou estatística depende de um conjunto de imagens de face de treinamento para descobrir modelos de face. Ele se baseia em aprendizado de máquina e análise estatística para encontrar as características relevantes das imagens de face (e de não face) e extrair recursos delas.

As características aprendidas estão, como já mencionado na forma de modelos de distribuição (histogramas ou gráficos) ou funções discriminantes que são consequentemente usadas para detecção de faces. A redução da dimensionalidade é geralmente realizada por uma questão de eficiência de computação e eficácia de detecção.

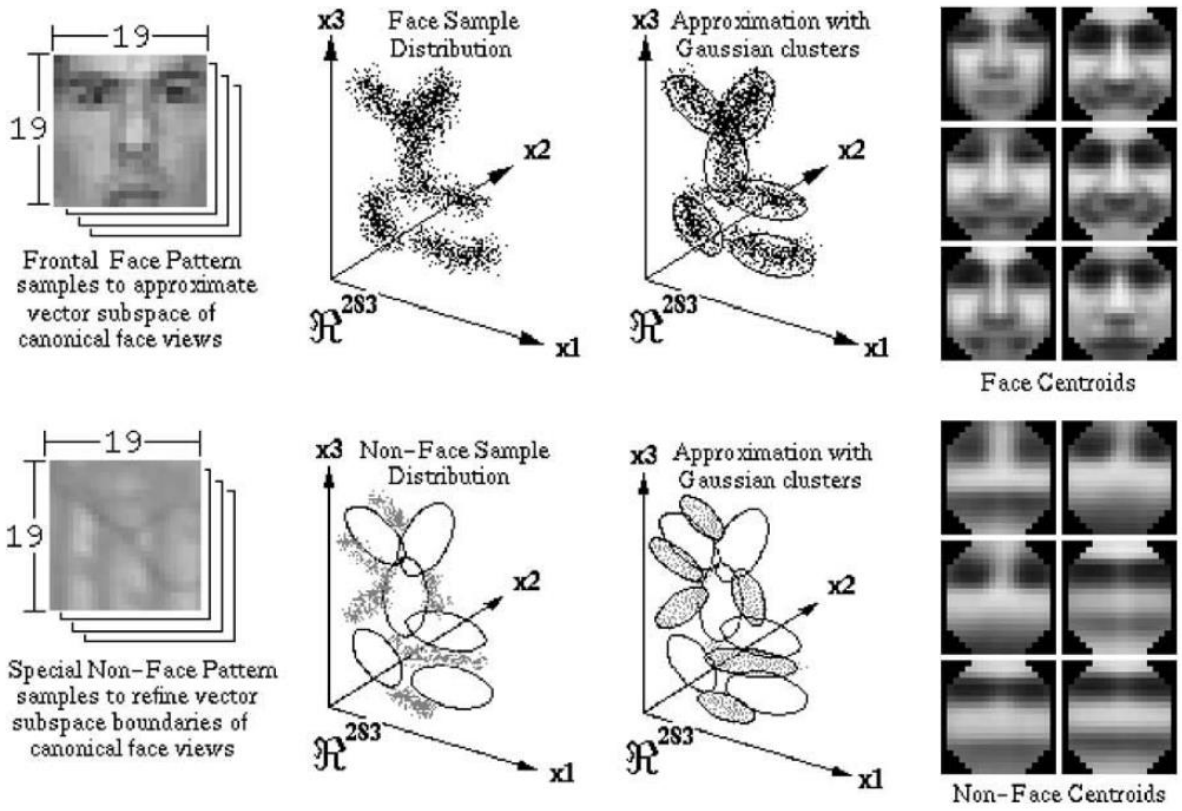
Muitos métodos baseados em aparência ou estatística podem ser entendidos em uma estrutura probabilística. Uma imagem ou vetor de característica derivado de uma imagem é visto como uma variável aleatória x, e essa variável aleatória é caracterizada para faces e não faces pelas funções de densidade condicional de classe p(x|face) e p(x|nonface). A classificação bayesiana ou probabilidade máxima pode ser usada para classificar a localização de uma imagem candidata como face ou não face. Infelizmente, uma implementação direta da classificação bayesiana é inviável devido à alta dimensionalidade de x, porque p(x|face) e p(x|nonface) são multimodais e porque ainda não é compreendido se existem formas parametrizadas naturais para p(x|face) e p(x|nonface). Conseqüentemente, muito do trabalho em um método baseado em aparência diz respeito a aproximações paramétricas e não paramétricas empiricamente validadas para p(x|face) e p(x|nonface).

Outra abordagem em métodos baseados em aparência ou estatística é encontrar uma função discriminante (ou seja, superfície de decisão, separação de hiperplano ou função de limite) entre as classes de face e não-face. Convencionalmente, os padrões de imagem são projetados para um espaço dimensional inferior e, em seguida, uma função discriminante é formada (geralmente com base em métricas de distância) para classificação, ou uma superfície de decisão não linear pode ser formada usando redes neurais multicamadas. Recentemente, máquinas de vetores de suporte (SVM – *Support Vector Machine*s) e outros métodos de kernel foram propostos.

Esses métodos projetam padrões implicitamente para um espaço dimensional superior e, em seguida, formam uma superfície de decisão entre a face projetada e os padrões não faciais. (Yang,Kriegman,Ahuja 2002, p. 43).

A Figura 3.5 apresenta um processo de segmentação (clustering) entre faces e não-faces utilizando funções de densidade, com distribuição Gaussiana.

Figura 3.5: Segmentação (clustering) de imagens de face e não-face utilizando funções de densidade



Fonte: Yang,Kriegman,Ahuja 2002, p. 43

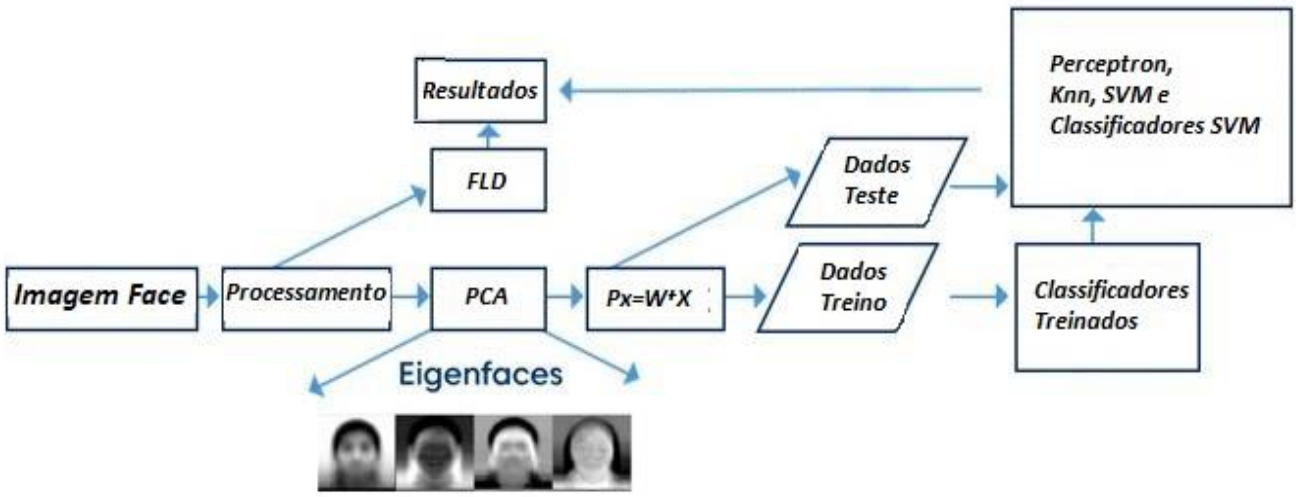
Portanto, como pode ser observado, esse método de detecção de face baseada em aparência reúne vários algoritmos/abordagens, onde as principais são Eigenface, FisherFace, PCA, HMM, Naive Bayes, LDA, KL etc.

Eigenface: O algoritmo baseado em Eigenface representa com eficiência faces usando Análise de Componentes Principais (PCA – *Principal Component Analysis*). O PCA é aplicado a um conjunto de imagens para diminuir a dimensão do conjunto de dados, descrevendo melhor a variação dos dados. Neste método, uma face pode ser modelada como uma combinação linear de autofaces (conjunto de autovetores). O reconhecimento facial, neste caso, é baseado na comparação de coeficientes de representação linear.

PCA e Discriminante de Fisher (FLD – Fisher´s Linear Discriminant): Algoritmos baseados em distribuição, como PCA e FLD, definem o subespaço que representa os padrões faciais. Eles geralmente têm um classificador treinado que identifica instâncias da classe de padrão de destino a partir dos padrões de imagem de fundo.

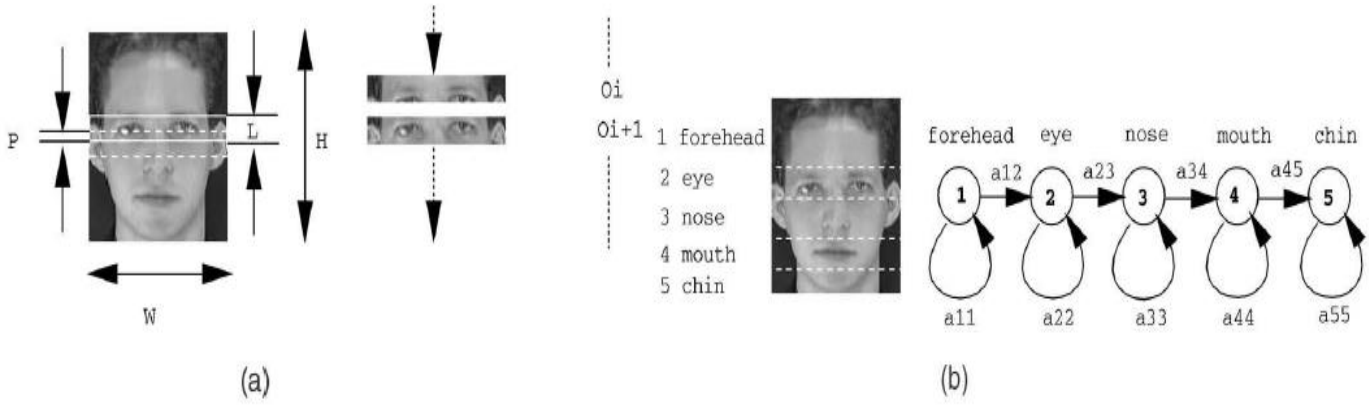
A Figura 3.6 apresenta um exemplo de processo de utilização de várias abordagens/algoritmos para detecção de face utilizando o método baseado na aparência.

Figura 3.6: Exemplo de processo de utilização de várias abordagens/algoritmos para detecção de face utilizando o método baseado na aparência



Modelo Oculto de Markov: O modelo oculto de Markov (HMM – Hidden Markov Model) é um método padrão para tarefas de detecção. Seus estados seriam os traços faciais, geralmente descritos como faixas de pixels. A Figura 3.7 ilustra a utilização do HMM para localização da face. Inicialmente (a) é gerado um vetor de observação, que serve como entrada de treinamento do HMM. Em seguida, (b) é gerado os estados ocultos, quando um HMM com cinco estados é treinado para uma sequência de observação.

Figura 3.7: Exemplo de utilização do Modelo Oculto de Markov para localização de uma face



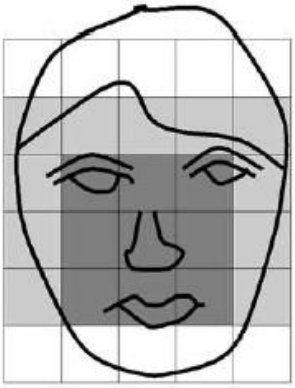
Fonte: Yang,Kriegman,Ahuja 2002, p. 47

Naive Bayes: Os classificadores Naive Bayes calculam a probabilidade de um rosto aparecer na imagem com base na frequência de ocorrência de uma série do padrão (geralmente padrões locais, como por exemplo o contorno ao redor dos olhos) nas imagens de treinamento.

3.3. Detecção de face baseada em conhecimento

Esta classe de métodos se baseia no conjunto de regras desenvolvidas pelos humanos de acordo com nosso conhecimento. Sabemos que uma face deve ter nariz, olhos e boca dentro de certas distâncias e posições entre si. A Figura 3.8 apresenta uma típica face usada por métodos deste tipo.

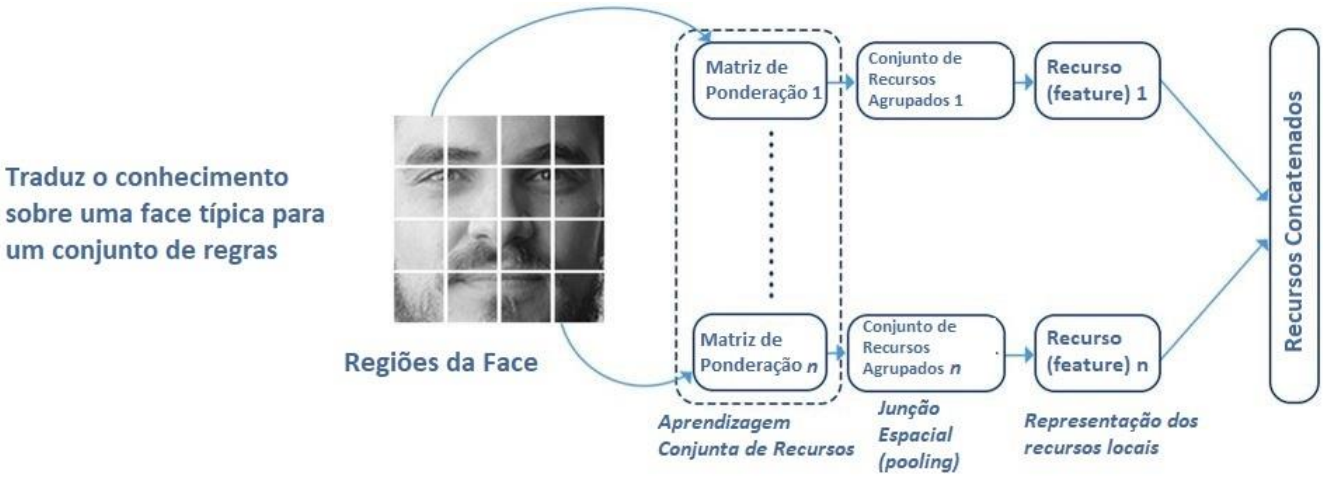
Figura 3.8: Uma típica face utilizada pelos métodos de detecção de face baseados em conhecimento



Fonte: Yang,Kriegman,Ahuja 2002, p. 37

O problema com este método é construir um conjunto apropriado de regras. Se as regras forem muito gerais ou muito detalhadas, o sistema acabará com muitos falsos positivos. No entanto, isso não funciona para todas as cores de pele e depende das condições de iluminação que podem alterar a tonalidade exata da pele de uma pessoa na foto. A Figura 3.9 apresenta o processo utilizado por essa classe de métodos.

Figura 3.9: Processo utilizado por métodos de detecção facial baseados em conhecimento



Na literatura encontramos uma subdivisão para este tipo de abordagem. A abordagem baseada em conhecimento pode ser dividida em Abordagem Geométrica e Abordagem Holística.

3.3.1 Abordagem Geométrica

As abordagens de base geométrica são aquelas que consideram as principais características do rosto humano, como olhos, boca, nariz, curvatura do rosto etc. para o reconhecimento da face.

Um exemplo da abordagem é mostrado na Figura 3.10. Com cada rosto humano, a informação das características faciais torna-se diferente e distinta. São elas: recursos visuais, recursos estatísticos, recursos de coeficiente de transformação e recursos algébricos (Hong 1991).

Figura 3.10: Abordagem baseada em características da face: baseado características chave geométricas e de forma incluindo os componentes da face como olhos, boca, nariz, queixo e curvaturas.

Fonte: Ahmed et al., 2019, p. 5

Nesta abordagem, os principais métodos/algoritmos encontrados na literatura são: THD, LEM, DT, DW, VIT entre outros.

3.3.2 Abordagem Holística

Abordagens com base holística são aqueles métodos que extraem detalhes vitais da imagem capturada e modelam a estrutura facial a partir dela. Posteriormente, é comparado com modelos faciais sintéticos existentes em banco de dados ou em forma de imagens de galeria.

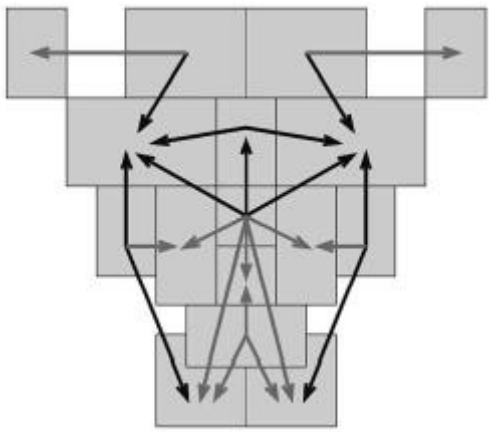
Nesta abordagem, os principais métodos/algoritmos encontrados na literatura são: PLS, PDM, PPCASIFT, HFI, SR entre outros.

3.4. Detecção de face por Correspondência de modelo

A classe de métodos de correspondência de modelo usa modelos de faces predefinidos ou parametrizados, como fragmentos de uma imagem de entrada bruta, para localizar ou detectar as faces pela correlação entre os modelos predefinidos ou deformáveis e as imagens de entrada.

A Figura 3.11 apresenta um modelo para localização da face com base em um modelo composto por 16 regiões (as caixas cinza) e 23 relações (mostradas por setas). Trata-se aqui de um modelo pré-definido.

Figura 3.11: Exemplo de um modelo (*template*) composto por 16 regiões e 23 relações

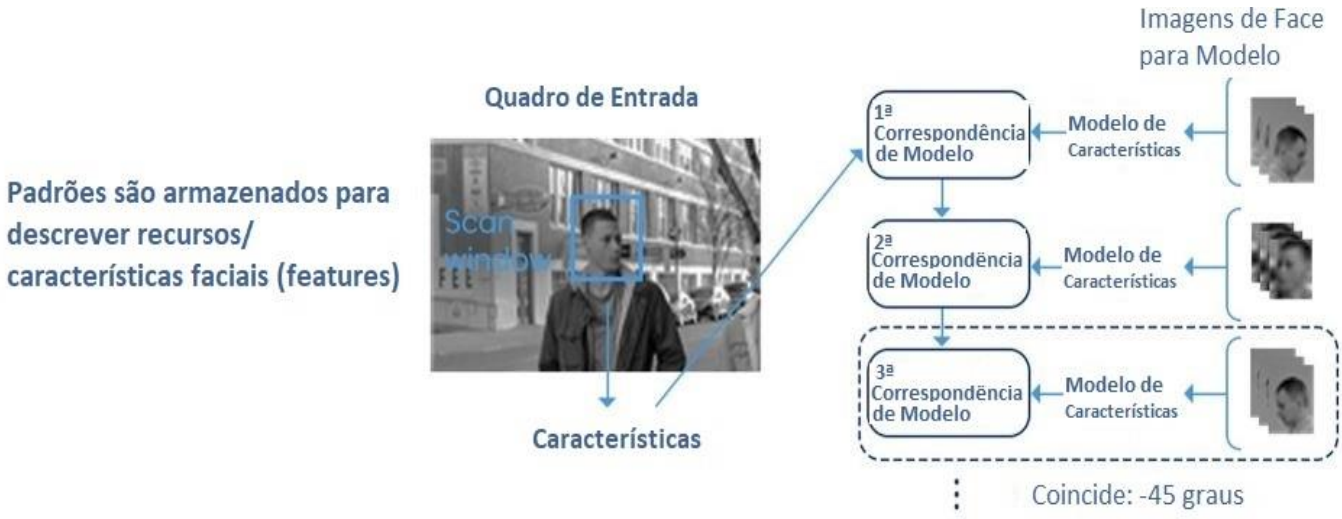


Fonte: Yang,Kriegman,Ahuja 2002, p. 42]

Dada uma imagem de entrada, os valores de correlação com os padrões pré-definidos são calculados para o contorno do rosto, olhos, nariz e boca de forma independente. A existência de uma face é determinada com base nos valores de correlação. Essa abordagem tem a vantagem de ser simples de implementar.

No entanto, este modelo provou ser inadequado para detecção de rosto, uma vez que não pode lidar com a variação de escala, pose e forma de maneira eficaz. Para contornar essa deficiência, são utilizadas multirresolução, multiescala, submodelos e modelos deformáveis para atingir invariância de escala e forma. A Figura 3.12 apresenta um processo padrão de utilização de templates (modelos pré-definidos) para detecção de uma face.

Figura 3.12: Processo utilizado por métodos de detecção facial baseados *templates* (modelos pré-definidos)



Os principais métodos/algoritmos dessa classe são: EGB, LGBPHS, AAM, IEM, GEM, Modelo MRF entre outros.

3.5 Detecção de face pela Abordagem por Aprendizagem de Máquina

Aprendizado de Máquina (ML) é a divisão de algoritmos que auxilia no fornecimento de softwares ou aplicativos para a previsão de resultados precisos sem a necessidade de programação extra. Esse tipo de algoritmo fornece a capacidade de aprendizado para o aplicativo, pois ele atualiza e melhora automaticamente seu desempenho para fornecer os melhores resultados possíveis.

Para tanto, existem, dentre muitas de suas possíveis divisões, a aprendizagem de máquina clássica, que divide os algoritmos em dois grupos principais: algoritmos de aprendizagem supervisionada e algoritmos de aprendizagem não-supervisionada. Existe uma terceira divisão, fora da aprendizagem de máquina clássica, que trata do aprendizado de máquina utilizando redes neurais, e atualmente, com uma profundidade (quantidade de nós e camadas) extremamente grande. A esse tipo, chamamos de Aprendizagem Profunda ou *Deep Learning*.

3.5.1 Aprendizagem Supervisionada

Nesta abordagem, existe a figura de um conjunto de dados (que chamamos de treinamento) que possuem as respostas, já inseridas por um especialista humano. No caso de imagens de face, existem rótulos identificando se a imagem é ou não positiva (possui ou não uma face) e, as vezes, em que local da imagem (coordenadas) a face está localizada.

Assim os algoritmos neste tipo de abordagem tentarão extrair um conjunto de pesos, correspondentes às características da imagem e assim, aplicar esses pesos para outras imagens futuras.

Nesta abordagem, os principais métodos/algoritmos encontrados na literatura são: PDBNN, NFL, SVM, PEM, LLR, PZMI-RBF, FSS, MPCA, GMA, HPEN, CGP entre outras.

3.5.2 Aprendizagem Não-Supervisionada

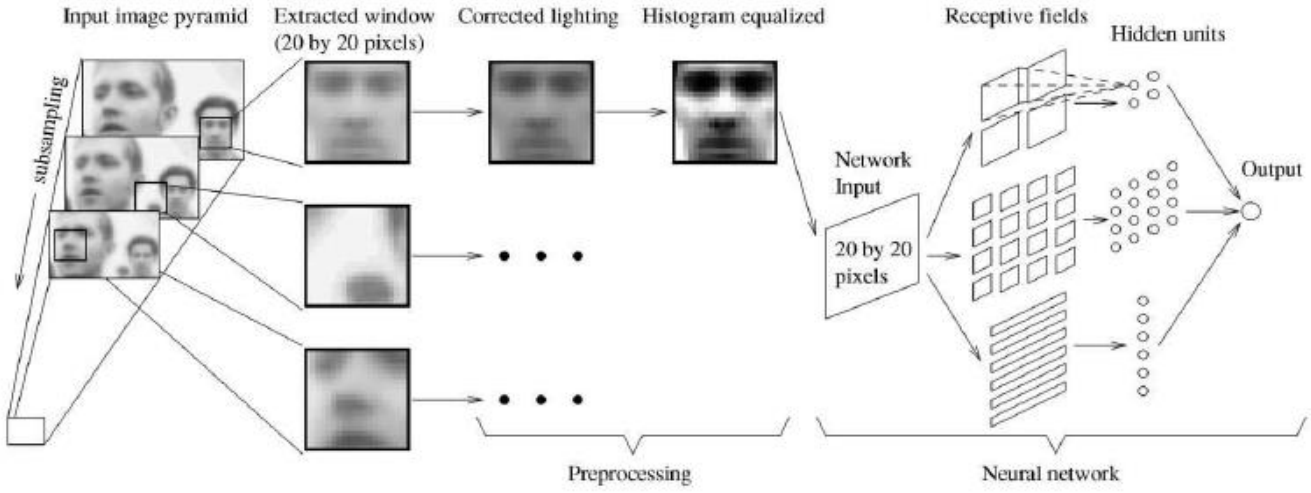
Nesta abordagem não existe um conjunto de dados de treinamento, com rótulos pré-definidos. Os algoritmos e a rede neural, terão que "aprender" por conta própria as características relevantes das imagens. No caso de reconhecimento facial, o algoritmo extrairá as informações relevantes e por tentativa e erro, irá treinando a rede neural e seus pesos.

Nesta abordagem, o principal método/algoritmo encontrado para o caso de reconhecimento facial na literatura é o SIFT.

3.5.3 Aprendizagem Profunda (Deep Learning)

As redes neurais, como CNN (Convolucional Neural Network ou Redes Neurais Convolucionais) e GANs (*Generative Adversarial Networks* ou Redes Adversariais Generativas), estão entre os métodos mais recentes e poderosos para detecção de problemas, incluindo detecção de face, detecção de emoção e reconhecimento de face. A Figura 16 apresenta um diagrama de pré-processamento da imagem de entrada e a análise (detecção da face) realizada por meio de uma rede neural.

Figura 16: O pré-processamento e a análise (detecção da face) por meio de uma Rede Neural



Fonte: Yang,Kriegman,Ahuja 2002, p. 46]

Nesta abordagem, os principais métodos/algoritmos encontrados na literatura são: CNN, DCCA, CEAL, DCAN, GAN, SPAE, DDML, DML, DCFL, DeepFace, MDML-DCP entre outras.

Essa é uma das muitas classificações de algoritmos e métodos de detecção e reconhecimento facial. No próximo capítulo vamos abordar ferramentas e serviços de detecção e reconhecimento facial disponíveis de forma paga ou gratuitamente para utilização em projetos.

4

Principais Ferramentas e Serviços para Detecção e Reconhecimento Facial

A implementação do processo de detecção e reconhecimento facial pode ser realizada de diversas formas. Podemos implementar, utilizando algoritmos e bibliotecas disponíveis em diversas linguagens de programação, ou então utilizarmos ferramentas e serviços disponíveis na rede (de forma paga ou gratuitamente).

Nos capítulos anteriores vimos uma série de algoritmos e métodos que permitem, com certa complexidade envolvida, a implementação completa desses processos em projetos. Entretanto, para quem precisa de uma implementação rápida, com alto índice de efetividade e baixa complexidade de implementação, pode optar pela segunda estratégia, ou seja, consumir ferramentas e serviços existentes.

Existem dezenas de soluções de detecção e reconhecimento de face, proprietárias e de código aberto, que oferecem vários recursos, desde a simples detecção da face até detecção de emoção e reconhecimento da face.

Vamos conhecer algumas dessas soluções disponíveis. Elas estão divididas em proprietárias (e que dependendo do nível de utilização exigirão o pagamento para sua utilização) e de código aberto.

4.1. Ferramentas e Serviços proprietários de detecção de face

Amazon Rekognition: O Amazon Rekognition é baseado no aprendizado profundo e está totalmente integrado ao ecossistema Amazon Web Service. É uma solução robusta para detecção e reconhecimento de rosto e é aplicável para detectar oito emoções básicas como "feliz", "triste", "zangado", etc.

Além disso, a ferramenta da Amazon permite determinar até 100 rostos em uma única imagem. Existe uma opção para vídeo e o preço é diferente para diferentes tipos de uso.

A API detecta rótulos - objetos (ou seja, pessoas, carros, móveis, roupas, animais de estimação), cenas (ou seja, bosque, praia, uma rua da cidade) ou conceitos (ao ar livre), atividades (ou seja, jogar futebol, patinar). Você pode detectar uma pessoa em uma foto ou vídeo, detectar pontos de referência faciais, expressar emoções e salvar metadados faciais. Além disso, você também pode comparar um rosto em uma imagem com os rostos detectados em outra imagem.

Portanto, trata-se de uma ferramenta multifuncional que pode ser usada não apenas para detectar rostos, mas também reconhecer entidades, objetos e até atividades. Além disso, com ReKognition você pode misturar e combinar fontes de fotos com IDs de usuário, o que pode permitir que você, digamos, reconheça objetos ou faces em fotos específicas.

Para iniciar, o link da documentação da API é o seguinte:

https://docs.aws.amazon.com/rekognition/latest/dg/API_Reference.html

O link para um curso específico de como utilizar a ferramenta ReKognition é o seguinte:

https://cloudacademy.com/course/amazon-rekognition/image-processing-api/

Face ++: Face ++ é um serviço em nuvem de análise facial que também possui um SDK offline para iOS e Android. Você pode realizar uma quantidade ilimitada de solicitações, mas apenas três por segundo. Também oferece suporte a Python, PHP, Java, Javascript, C ++, Ruby, iOS, Matlab, fornecendo serviços como reconhecimento de gênero e emoção, estimativa de idade e detecção de pontos de referência.

Eles operam principalmente na China, são excepcionalmente bem financiados e são conhecidos por sua inclusão nos produtos Lenovo. No entanto, lembre-se de que sua controladora, a Megvii, foi sancionada pelo governo dos EUA no final de 2019.

O funcionamento ocorre da seguinte forma: as faces detectadas são armazenadas no FaceSet. Um Face_token é utilizado como um Identificador exclusivo para detecção da face no sistema. A partir daí existem dois "endpoints":

- *Detecção da face*: detecta informações da imagem passada para a API (como por exemplo localização do rosto, idade, raça, sexo etc.).
- *Pontos de referência da Face*: possibilita a obtenção de mais de 1000 pontos-chave da imagem passada para a API, o que permite a localização com precisão dos elementos, contornos faciais e suas características.

Em termos gerais, possui uma excelente precisão, técnicas robustas de anti-falsificação, atualização frequente de modelos e leve (em termos computacionais).

Para iniciar sua utilização, você pode acessar o seguinte link:

https://console.faceplusplus.com/documents/6329584

O link de referência para a API do Face++ é o seguinte:

https://console.faceplusplus.com/documents/5678948

O link do Face++ no GitHub é o seguinte:

https://github.com/FacePlusPlus

Lambda Labs: A API de reconhecimento facial e detecção de rosto (Lambda Labs) fornece reconhecimento facial, detecção facial, posição dos olhos, posição do nariz, posição da boca e classificação de gênero. Ele oferece até 1000 solicitações gratuitas por mês.

O site oficial da empresa está no link: https://lambdalabs.com/index.html

A API de reconhecimento facial da Lambda Labs pode ser acessada pelo link: https://lambdalabs.com/face-recognition-api . Além da API web padrão, existe também disponível as APIs para iOS e Glass.

Existe uma boa documentação disponível no link:

https://lambdalabs.com/api-documentation

Kairos: A Kairos oferece uma variedade de soluções de reconhecimento de imagem. Seus endpoints de API incluem a identificação de gênero, idade, reconhecimento facial e profundidade emocional em fotos e vídeos. Eles oferecem uma avaliação gratuita de 14 dias com um limite máximo de 10.000 solicitações, fornecendo SDKs para PHP, JS, .Net e Python.

Para reconhecer faces, ele utiliza o poder da visão computacional e do aprendizado profundo. É muito simples de usar. Tudo que você precisa é apenas enviar imagens e / ou vídeos para a API, os algoritmos irão analisar os rostos encontrados, então a API retorna um monte de dados úteis sobre os rostos que encontra.

Mas, há algo para o qual você precisa estar preparado. Esta ferramenta aceita apenas alguns parâmetros: “image”, “gallery_name”, “subject_id”, “selector”. Isso significa que podemos construir rapidamente solicitações HTTP descomplicadas. Além disso, o parâmetro “imagem” aceita uma URL acessível publicamente, upload de arquivo ou foto codificada em *base64*. A ferramenta pode ser testada online pelo seguinte endereço: https://www.kairos.com/demos

Ao utilizar esta ferramenta você não precisa construir seu próprio banco de faces ou entender sobre algoritmos estatísticos. Como a maioria das ferramentas proprietárias, é muito simples e fácil de utilizar.

A documentação para utilizar a API pode ser acessada pelo seguinte link:

https://www.kairos.com/docs/api/

O tutorial para realizar o reconhecimento facial por meio da API da Kairos pode ser acessado pelo seguinte link:

https://www.kairos.com/docs/getting-started-with-kairos-face-recognition

Para usar a API da Kairos com JavaScript, utilize o seguinte link:

https://rapidapi.com/blog/face-recognition-api-javascript/

A documentação da API da Kairos é bem completa. Inclusive, apresenta um roteiro de melhores práticas para o processo de reconhecimento facial. Disponível no seguinte link:

https://www.kairos.com/docs/api/best-practices

Microsoft Face: A API Face dos Serviços Cognitivos do Microsoft Azure permite que você faça 30.000 solicitações por mês, 20 solicitações por minuto gratuitamente. Para solicitações pagas, o preço depende do número de reconhecimentos por mês, a partir de US$ 1,00 por 1000 reconhecimentos (numa taxa de 10 transações por segundo). Os recursos incluem estimativa de idade, reconhecimento de gênero e emoção, detecção de pontos de referência.

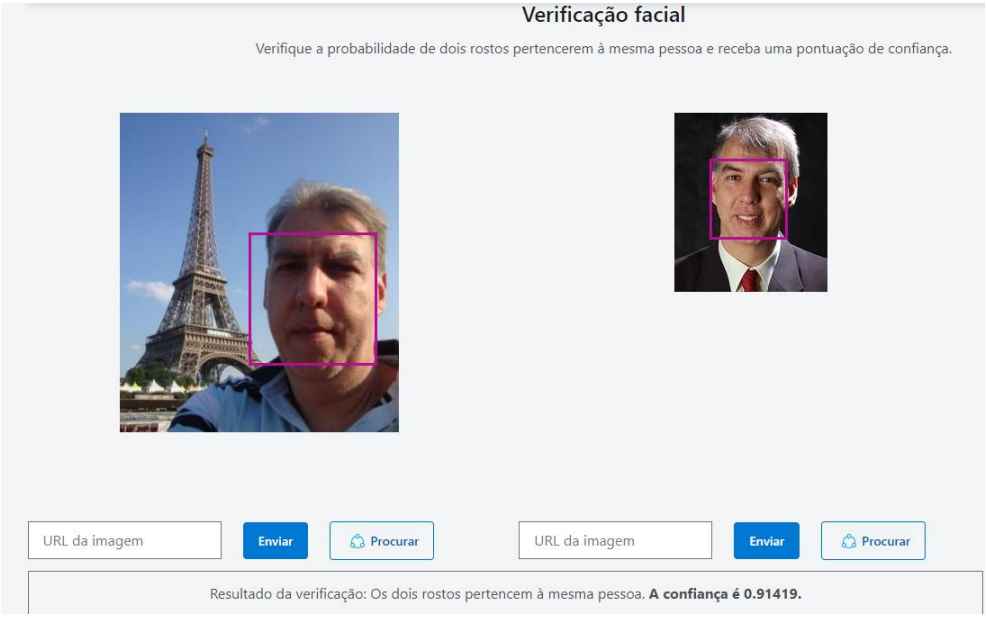
Além desses recursos, segundo as informações oficiais do produto, a API do Microsoft Face percebe também as seguintes características e atributos faciais: face com máscara, com óculos e a localização do rosto na imagem). Os SDKs são compatíveis com Go, Python, Java, .Net e Node.js.

A página inicial do produto, com informações gerais, funcionalidades, demonstração, preços, primeiros passos e documentação é a seguinte:

https://azure.microsoft.com/pt-br/services/cognitive-services/face/

Nesta mesma página, pode-se testar a API, com o envio de imagens. A API compara, verifica se encontra uma face nas imagens e as compara, indicando um grau de confiança de semelhança entre as imagens. A Figura 4.1 apresenta um teste desta API.

Figura 4.1: Teste da API do serviço Microsof Face



Fonte: Página principal do serviço Microsoft Face

Existe também um curso específico disponibilizado pela própria Microsoft para que você possa aprender com maior profundidade a utilizar a API e os demais serviços cognitivos da Microsoft. O link é o seguinte:

https://docs.microsoft.com/pt-br/learn/modules/detect-analyze-faces/

Além desse curso, a documentação disponível é bem ampla e abrangente. O link é o seguinte:

https://docs.microsoft.com/pt-br/azure/cognitive-services/face/

Paravision: A Paravision é uma empresa de reconhecimento facial para empresas que fornecem soluções auto hospedadas. Entre os seus serviços estão as soluções de detecção, verificação e identificação facial, vídeo streaming em tempo real, SDKs e contêineres Docker, soluções anti-falsificação e pós-COVID-19 (reconhecimento facial com máscaras, integração com detecção térmica etc.). A empresa possui SDKs para C ++ e Python.

O link de acesso ao produto de reconhecimento facial da empresa Paravision é o seguinte:

https://www.paravision.ai/product/face-recognition/

Existe uma gama de documentos e recursos disponíveis em:

https://www.paravision.ai/resources/

4.2. Ferramentas e Serviços de detecção de face de código aberto

OpenBR: Trata-se de um framework de código aberto (open source) de reconhecimento biométrico. Conforme ilustra em sua página principal (http://openbiometrics.org/), possibilita a detecção, normalização, representação, extração de recursos e a correspondência (matching) de uma imagem (de face).

O framework oferece suporte ao desenvolvimento de algoritmos abertos e avaliações reproduzíveis. Ele opera em sistemas operacionais baseados em Windows, Linux, OS X e Raspbian. A propósito, o projeto está licenciado de acordo com o Apache 2.0. Além disso, implementa o algoritmo 4SF2 para realizar o reconhecimento facial. Os algoritmos de software também funcionam para estimativa de idade e estimativa de gênero.

É um software/framework completo em conformidade com o NIST que avalia o reconhecimento facial, detecção e marcação de terreno. OpenBR expõe uma API C++ que pode ser embutida em seus próprios aplicativos.

Para maiores informações a página principal do projeto pode ser acessada. Além dessa página, existe o repositório no GitHub: https://github.com/biometrics .

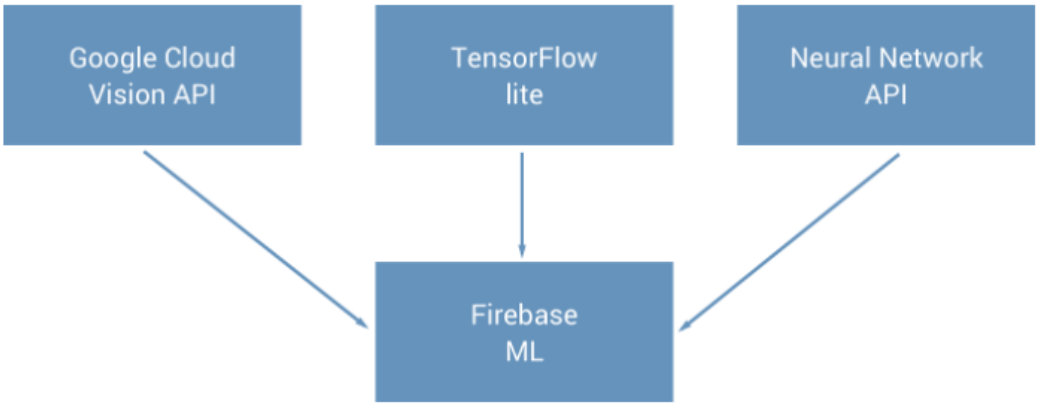
Existe também uma boa documentação sobre a API do OpenBR, que pode ser acessada em:

http://openbiometrics.org/docs/

Firebase ML Kit

O Firebase ML Kit, que é uma parte do pacote Firebase, oferece suporte a recursos inteligentes com menos complexidade de implementação, tudo "empacotado" dentro de um único SDK. A Figura 4.2 ilustra esse conjunto.

Figura 4.2: Estrutura do SDK Firebase ML Kit



Fonte: https://developers.google.com/ml-kit

Conforme indica a página oficial de documentação do ML-Kit do Firebase, atualmente o pacote oferece suporte a duas grandes APIs: Visão Computacional e Linguagem Natural. A API de Visão Computacional, oferece suporte para os seguintes serviços/processos: Escaneamento de código de barras, Detecção facial, Rotularização de imagens, Detecção e rastreamento de objetos, Reconhecimento de textos, Reconhecimento de escrita a mão, Detecção de pose e segmentação de objetos em imagens. Já a API de Linguagem Natural, oferece suporte para os seguintes serviços/processos: Identificação de Linguagem, Tradução no dispositivo, Resposta inteligente e Extração de entidades.

Para poder utilizar esses recursos, tudo o que se precisa fazer é passar os dados desejados para o SDK e, em troca, receberemos os dados de volta, dependendo da parte do ML Kit que se estiver utilizando. Os dados retornados dependerão do recurso de aprendizado de máquina que está sendo usado.

Um dos atrativos para se utilizar o Firebase ML é que ele oferece seus recursos de aprendizado de máquina no **dispositivo** e **na nuvem**, o que permite que você seja criativo e atento a como e quando usar o aprendizado de máquina. Para utilizá-lo no dispositivo, o próprio SDK permite baixar uma versão "lite" do

TensorFlow. Por exemplo, algumas operações podem ser intensas, assim, pode-se focar o processamento no dispositivo ou na nuvem. Tudo depende do balanceamento entre a latência de utilização de recursos em nuvem e o baixo poder de processamento dos dispositivos móveis. As APIs no dispositivo no Firebase MLKit são projetadas para funcionar rapidamente e podem fornecer resultados mesmo quando uma conexão de rede não está presente. Por outro lado, as APIs baseadas na nuvem utilizam a tecnologia de aprendizado de máquina das plataformas do Google Cloud para fornecer um maior nível de precisão. A Tabela 4.1 apresenta alguns desses recursos e se os mesmos estão disponíveis na nuvem, no dispositivo ou em ambos.

Tabela 4.1: Exemplo de disponibilidade de alguns recursos do Firebase ML Kit

Recurso	No Dispositivo	Na Nuvem
Reconhecimento de Texto	☑	☑
Detecção de Face	☑	☒
Escaneamento de Código de Barras	☑	☒
Rotularização de Imagem	☑	☑
...		

Fonte: https://developers.google.com/ml-kit

ageitgey/
face_recognition

Ageitgey / face recognition: é um repositório GitHub com 42k estrelas, uma das bibliotecas de reconhecimento facial mais extensas. Os colaboradores também afirmam que é a "API de reconhecimento facial mais simples para Python e linha de comando".

No entanto, suas desvantagens são o lançamento mais recente foi em 2018 e a precisão de reconhecimento de modelo de 99,38% (esse valor se baseia nos testes realizados com a Base de Dados Labeled Faces in the Wild (LFW) disponível em http://vis-www.cs.umass.edu/lfw/), o que poderia ser muito melhor em 2021. Ele também não tem uma API REST.

O algoritmo utiliza a biblioteca Dlib com aprendizagem profunda (deep learning).

O repositório no GitHub pode ser acessado pelo seguinte link:

https://github.com/ageitgey/face_recognition.

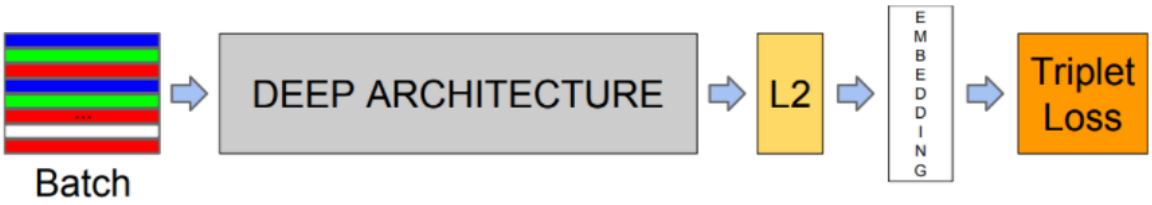
A forma de utilização, os principais comandos e a forma de instalação e utilização são bem informados no próprio arquivo de descrição no GitHub.

Google FaceNet

FaceNet: desenvolvido por pesquisadores do Google e é descrito no artigo apresentado no CVPR 2015 (SCHROFF et al, 2015). Foi utilizada as arquiteturas ZF-Net e Inception para o desenvolvimento do modelo FaceNet . O repositório possui mais de 12,2K estrelas. Enquanto isso, as últimas atualizações significativas foram em 2018. A precisão de reconhecimento é de 99,65%, e não possui API REST.

O modelo foi testado (conforme descrito no artigo) em duas bases de dados LFW (com 99,63% de acurácia) e Youtube Face database (com 95,12% de acurácia). A Figura 4.3 apresenta a arquitetura do modelo FaceNet.

Figura 4.3: Arquitetura do modelo FaceNet



Fonte: Schroff et al, 2015

O repositório oficial do modelo no GitHub pode ser acessado pelo seguinte link:

https://github.com/davidsandberg/facenet .

Existem vários tutoriais e documentações disponível sobre este modelo.

Um tutorial, mostrando o funcionamento do modelo, pode ser acessado pelo seguinte link:

https://www.pluralsight.com/guides/face-recognition-walkthrough-facenet

Uma boa explicação do funcionamento do modelo, inclusive da função de perda tripla, pode ser encontrada no artigo https://medium.com/analytics-vidhya/introduction-to-facenet-a-unified-embedding-for-face-recognition-and-clustering-dbdac8e6f02

Um outro exemplo, utilizando o modelo FaceNet em dispositivos móveis pode ser acessado em:

https://towardsdatascience.com/using-facenet-for-on-device-face-recognition-with-android-f84e36e19761

Outro exemplo pode ser o projeto de Reconhecimento Facial com FaceNet no site do Kaggle. O link é o seguinte:

https://www.kaggle.com/yhuan95/face-recognition-with-facenet

Por fim, um dos mais interessantes e completos tutoriais está no link:

https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier/

(entretanto, ele utiliza uma variação, das muitas que existem, do modelo FaceNet: o OpenFace (que descreveremos a seguir).

OpenFace

OpenFace: é uma implementação Python e Torch de reconhecimento facial com redes neurais profundas. Ele se baseia no Modelo FaceNet e no artigo (SCHROFF et al, 2015), descritos anteriormente. Além de ser de código aberto, possui um dos melhores desempenho. Tem uma licença gratuita e permite fins comerciais.

O algoritmo por de traz do OpenFace tem o seu funcionamento da seguinte forma: começa detectando o rosto usando dlib - uma biblioteca de aprendizado de máquina. Em seguida, a face detectada e devidamente recortada é passada para um extrator de recursos CNN que gera incorporações faciais com 128 dimensões. Uma vez que a face está devidamente identificada por recursos, ela pode ser então agrupada, classificada e comparada com outras faces (similaridade).

Um dos pontos positivos é a possibilidade de utilização em dispositivos móveis (em virtude da redução de dados necessários para treinar o modelo) e também uma excelente documentação.

Para Configuração do OpenFace: https://cmusatyalab.github.io/openface/setup/

Documentação da API: https://openface-api.readthedocs.io/en/latest/index.html

Link do Projeto no GitHub: https://cmusatyalab.github.io/openface/

Tutorial para Reconhecimento Facial utilizando OpenFace e OpenCV:

https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

Repositório da Biblioteca Dlib: http://dlib.net/

Repositório do Projeto PyTorch: http://torch.ch/

Deepface: é um framework completo para Python com 2,9K estrelas no GitHub, fornecendo análise de atributos faciais como idade, sexo, raça e emoção. Ele também fornece uma API REST.

Trata-se de um framework completo para reconhecimento facial (leve) e análise de atributos faciais (como mencionado acima: idade, sexo, emoção e raça).

É um framework hibrido incluindo o estado da arte de vários modelos, tais como:

VGG-Face (https://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/parkhi15.pdf) ,

Google FaceNet (https://arxiv.org/abs/1503.03832) ,

OpenFace (https://cmusatyalab.github.io/openface/) ,

Facebook DeepFace (https://research.fb.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/) ,

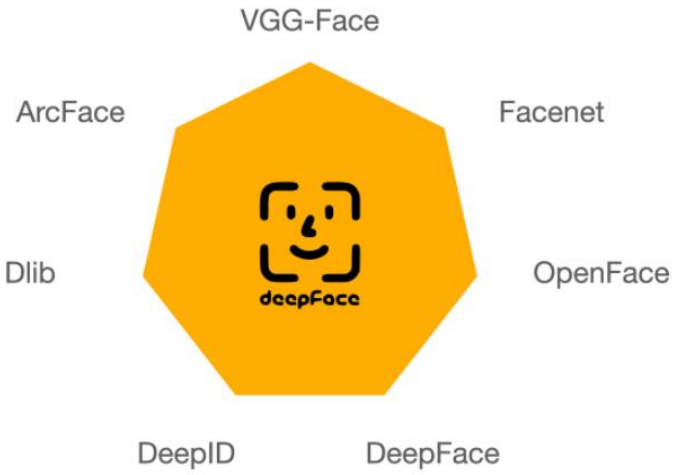
DeepID (https://arxiv.org/pdf/1406.4773.pdf) ,

ArcFace (https://arxiv.org/pdf/1801.07698.pdf) e

Dlib (https://github.com/davisking/dlib-models).

A Figura 4.4 ilustra graficamente isso.

Figura 4.4: O Framework DeepFace e sua base de funcionamento



Fonte: GitHub https://github.com/serengil/deepface

O repositório no GitHub pode ser acessado pelo seguinte link: https://github.com/serengil/deepface

A Documentação, forma de utilização e demais descrições são bem completas e de fácil entendimento. Estão, essencialmente, na descrição da página inicial do repositório GitHub do framework.

Segundo os autores, o framework tem um índice de acurácia superior aos seres humanos, ou seja, maior que 97,53%.

Os autores detalham e comparam o framework a outros modelos. Essas informações podem ser acessadas a partir do link: https://sefiks.com/2019/02/13/apparent-age-and-gender-prediction-in-keras/

InsightFace: É um projeto desenvolvido em Python, baseado em PyTorch e MXNet. Seu repositório no GitHub possui mais de 10,8k estrelas, e está sendo atualizado ativamente. A precisão do reconhecimento é de 99,86%. Os autores afirmam fornecer uma variedade de algoritmos para detecção, reconhecimento e alinhamento de rosto.

Na página oficial do projeto (link: https://insightface.ai/), podem ser encontrados todos os links e informações necessárias para utilizar o modelo (que é de código aberto sob licença MIT). O repositório do projeto no GitHub pode ser acessado em: https://github.com/deepinsight/insightface

O projeto ganhou vários prêmios entre os anos de 2019 e 2021, entre eles o primeiro lugar no NIST-FRVT1:1 VISA 2021 e primeiro lugar no iQIVI VID Challenge 2019.

A documentação é bem completa e pode ser acessada em: https://github.com/deepinsight/insightface/tree/master/python-package

O projeto também possui uma API (API-REST) que, segundo as palavras de seus autores, "[...] visa fornecer API REST conveniente, facilmente implementável e escalonável para pipeline de detecção e reconhecimento de face InsightFace usando FastAPI para veiculação e NVIDIA TensorRT para inferência otimizada". O link no GitHub da API é: https://github.com/SthPhoenix/InsightFace-REST

OpenCV: não é uma API, mas é uma ferramenta valiosa com mais de 3.000 algoritmos de visão computacional otimizados. Ele oferece muitas opções para desenvolvedores, incluindo módulos de reconhecimento facial Eigenfacerecognizer, LBPHFacerecognizer ou lpbhfacerecognition.

A própria API / Biblioteca, assim como toda a documentação, podem ser acessadas pelo link:

https://opencv.org/

Macgyver API: Esta API fornece um conjunto de ferramentas para detecção e reconhecimento facial em imagens. Os recursos fornecidos incluem: comparar dois rostos (reconhecimento de rosto), detectar a presença de rostos em uma imagem e retornar as coordenadas X, Y dos rostos detectados nas imagens.

Essa API aproveita o aprendizado de máquina e, especificamente, as redes neurais convolucionais profundas criadas no TensorFlow. Existe uma grande variedade de algoritmos para escolher. Com essa API e seus recursos avançados de reconhecimento facial e de imagens, podem potencializar a construção de aplicativos.

O link do projeto no GitHub: https://github.com/MacgyverCode

Além dos algoritmos, métodos, ferramentas e serviços de detecção e reconhecimento facial, os profissionais que trabalharão com a implementação desses recursos devem estar cientes de que a matéria-prima de seus projetos são as faces (rostos) das pessoas. Segundo a Lei Geral de Proteção de Dados (LGPD) de xx , as imagens que podem identificar as pessoas são consideradas "dados sensíveis", e como tal, devem ser tratadas de acordo com o que propõe a lei.

No próximo capítulo isso e muitos outros aspectos legais envolvidos na captura, guarda e utilização das imagens das faces das pessoas será tratado em detalhes.

5

O Reconhecimento Facial à Luz da Legislação Brasileira

A utilização do reconhecimento facial e de diversos sistemas inteligentes em várias aplicações, tanto por pessoas jurídicas de Direito Privado como de Direito Público e por pessoas físicas, levando a tomada de decisões e ainda, a ter os responsáveis que assumirem as responsabilidades e consequências de sua adoção desperta a necessidade de um olhar jurídico atencioso às lesões provocadas por esse tipo de relação entre humanos e a inteligência artificial e demais recursos tecnológicos.

O propósito precípuo de tais recursos tecnológicos é apresentar soluções mais ágeis e que tornem a vida dos indivíduos em sociedade mais facilitada, resolvendo problemas cotidianos, sem o dispêndio de tempo humano, por ser este cada vez mais precioso para todos, sendo a utilização destes aparatos inovativos um viés de possibilidade para a realização de outras tarefas e atividades eleitas no dia a dia como prioritárias.

Talvez, o problema surja com a consciência humana de que esses recursos tecnológicos precisam ser adotados em larga escala, trazendo para as diversas relações sociais e econômicas, a maioria delas protegidas ou tuteladas pelo Direito, uma fragilidade quanto ao sistema de proteção jurídica por acreditar-se não se achar suficientemente regulamentado para que possa proteger os cidadãos de lesões a direitos advindas dessas relações, ainda mais quando se trata da utilização do reconhecimento facial.

Ledo engano, pois, como restará demonstrado, a falha legislativa ou a ausência de legislações específicas quer para a utilização do reconhecimento facial, quer para a adoções de inteligências artificiais nas mais diversas áreas de atuação humana e para o uso de diversos outros recursos tecnológicos existentes e para os que ainda irão ser criados ou desenvolvidos, não afeta ou deixa desprovido os cidadãos de proteção jurídica necessária a limitar os abusos decorrentes da adoção desses recursos.

As legislações existentes, embora apresentem muitas vezes lacunas e falhas, deixando brechas para a aplicação não eficaz ou que satisfaça os cidadãos não são insuficientes para garantir-lhes a devida tutela como restará demonstrado.

5.1. A Constituição Federal de 1988 e o desenvolvimento tecnológico

A legislação brasileira tem como fundamento para a tutela dos direitos a Constituição Federal Brasileira, promulgada em 05 de outubro de 1988, composta de 250 (duzentos e cinquenta) artigos (BRASIL, 2023), que a estrutura e consequentemente a União dos Estados Federados do Brasil, como se pode ver a Constituição, constitui as ações, tanto do Estado, representado pela Administração Pública Direta e Indireta, como pelos cidadãos que a legitimam obedecendo seus ditames.

Importante é reconhecer-se essa função atribuída à Constituição Federal de um documento que traz os princípios ideológicos a serem seguidos, pois, nada que se apresente nas relações humanas, quer seja entre cidadãos, quer entre o Estado e estes e o Estado e outras nações, podem se furtar ao que neste documento encontra-se parametrizado, não sendo diferente com relação a utilização de recursos científicos

e tecnológicos, mesmo que de forma indireta, através do respeito ao Princípio da Dignidade da Pessoa Humana.

A Constituição constitui ação (STRECK, 2004, p. 51-93), se podendo entender que:

> ... a Constituição do Brasil não é um mero "instrumento de governo", enunciador de competências e regulador de processos, mas, além disso, enuncia diretrizes, fins e programas a serem realizados pelo Estado e pela sociedade. Não compreende tão somente um "estatuto-jurídico-político", mas, um "plano global normativo" da sociedade e, por isso mesmo, do Estado brasileiro. Daí ser ela a Constituição do Brasil, e não apenas a Constituição da República Federativa do Brasil. Os fundamentos e os fins definidos em seus artigos 1 e 3 são os fundamentos e os fins da sociedade brasileira. (GRAU, 2005, 144p.)

Temos então determinado no art. 3º da Constituição, os objetivos fundamentais da República Federativa do Brasil, consistem em:

I – Construir uma sociedade livre, justa e solidária;

II – Garantir o desenvolvimento nacional;

III – Erradicar a pobreza e a marginalização e reduzir as desigualdades sociais e regionais.

Para que o Estado Brasileiro consiga atingir tais objetivos considerados fundamentais da República são necessárias políticas públicas que efetivamente incentivem e levem ao desenvolvimento, promovendo condições ao mercado financeiro, econômico, de consumo e às várias instâncias sociais, de forma rápida e com a devida segurança jurídica, através de instrumentos e instituto que permitam ultrapassar as estruturas do subdesenvolvimento, dentre outros, por exemplo, a criação de empresas e *startups* (MATIAS, 2021, 203p.).

Inclusive no Capítulo IV, o texto constitucional trouxe a tratativa relativa no que diz respeito a ciência, a tecnologia e a inovação, estando no artigo 218 determinado que o Estado promoverá e incentivará o desenvolvimento científico, a pesquisa, a capacitação científica e tecnológica e a inovação. Para tanto, determinou ainda que o mercado interno integra o patrimônio nacional e será incentivado de modo a viabilizar o desenvolvimento cultural e socioeconômico, o bem-estar da população e a autonomia tecnológica do País, nos termos de lei federal, assim dispondo no seu art. 219.

Restou definido constitucionalmente que cabe ao Estado, promover e incentivar o desenvolvimento científico, a pesquisa, e a capacitação tecnológica, sendo, inclusive, facultado aos Estados Membros e ao Distrito Federal vincular parcela de sua receita orçamentária a entidades públicas de fomento ao ensino e à pesquisa científica e tecnológica, parametrizando a forma como os recursos materiais deverão ser arrecadados para que possa haver o cumprimento do que determinou a Constituição Federal.

Não se pode deixar de enfatizar, que em dias atuais, desde o início da Quarta Revolução Industrial (SCHWAB, 2016, 159p.), que tem estabelecido uma nova ordem mundial, onde a tecnologia tem intermediado de forma quase que absoluta as relações humanas, essa realidade preconizada na Constituição do Brasil, passou a ser premente no que se refere à promoção e incentivo da criação de tecnologias inovadoras e disruptivas (ITO e HOWE, 2018, 305p.).

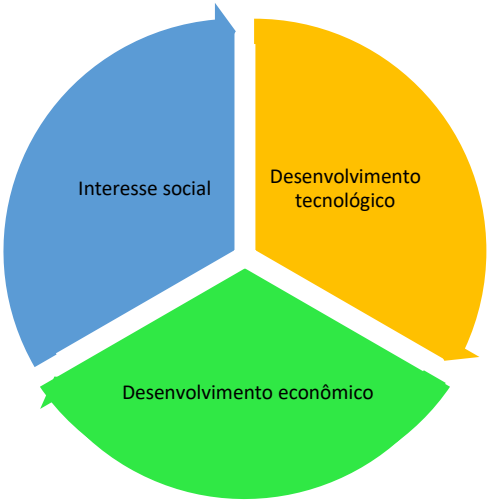
Inclusive, SCHWAB adverte:

> As mudanças são tão profundas que, na perspectiva da história da humanidade, nunca houve um momento tão potencialmente promissor ou perigoso. A minha preocupação, no entanto, é que os tomadores de decisão costumam ser levados pelo pensamento tradicional linear (e

> sem ruptura) ou costumam estar muito absorvidos por preocupações imediatas; e, portanto, não conseguem pensar de forma estratégica sobre as forças de ruptura e inovação que moldam nosso futuro. (SCHWAB, 2016, p.12)

Acredita-se que o artigo 5º da Constituição, nas garantias e deveres fundamentais dos cidadãos, assegurou aos autores de inventos industriais o privilégio temporário para sua utilização, bem como proteção às criações industriais, à propriedade das marcas, aos nomes de empresas e a outros signos distintivos, tendo em vista o interesse social e o desenvolvimento tecnológico e econômico do País, no inciso XXIX, reconhecendo a necessidade de se promover a inovação de forma constante.

Portanto, para que haja o desenvolvimento do Estado Brasileiro como se preconiza, deverá haver uma confluência entre o desenvolvimento tecnológico, o desenvolvimento econômico e, ainda, interesse social; em todos esses aspectos deverão ser propostas políticas públicas e fomento suficiente para que possam ser implementados a contento.



O parágrafo único do artigo 219 da Constituição é claro ao afirmar que o Estado estimulará a formação e o fortalecimento da inovação nas empresas, bem como nos demais entes, públicos ou privados, a constituição e a manutenção de parques e polos tecnológicos e de demais ambientes promotores da inovação, a atuação dos inventores independentes e a criação, absorção, difusão e transferência de tecnologia.

Sob a égide de tais ditames a Constituição autorizou que os cidadãos fossem além das estruturas empresariais e gestão usualmente existentes e praticadas, o modelo empresarial clássico restou superado para dar espaço a inovações e formas de gestão e empreendedorismo cada vez mais sofisticadas, inclusive, pela utilização das mais variadas espécies de recursos tecnológicos, através de programas, plataformas e aplicativos, aptos a desenvolverem a prática comercial de uma maneira diferente.

A tutela relativa às novas tecnologias da rede mundial de computadores, tem sua base na Constituição Federal e, se acha protegida através de normas infraconstitucionais. Por determinação da Lei nº 12.965/2014 (BRASIL, 2023), que estabeleceu o Marco Civil da *Internet*, trouxe em seu artigo 4º, III, a exigência por parte do Estado em sua atividade disciplinadora, o fomento da inovação e a difusão de novas tecnologias nos seguintes termos:

> Art. 4º – A disciplina do uso da internet no Brasil tem por objetivo a promoção:
> (...)
> III – da inovação e do fomento à ampla difusão de novas tecnologias e modelos de uso e acesso.

No caminho do desenvolvimento tecnológico, legalmente estabelecido como objetivo estatal, as empresas têm procurado nortear seus investimentos, de forma que as atividades por elas realizadas possam gerar, circular e distribuir riquezas, contribuindo para que haja a continuidade social e uma economia que garanta possibilidades de subsistência, apresentando soluções inovadoras às questões socialmente prementes como restou demonstrado pela utilização de programas, aplicativos e plataformas tendo em vista a superação necessária trazida pela pandemia Covid-19 (AGÊNCIA BRASIL, 2021).

Nesse aspecto, o Supremo Tribunal Federal enalteceu a importância da ciência e tecnologia, tendo a Ministra Carmem Lúcia assim se manifestado quando da realização de um dos julgamentos na Corte:

> O termo 'ciência', enquanto atividade individual, faz parte do catálogo dos direitos fundamentais da pessoa humana (inciso IX do art. 5º da CF). Liberdade de expressão que se afigura como clássico direito constitucional-civil ou genuíno direito de personalidade. Por isso que exigente do máximo de proteção jurídica, até como signo de vida coletiva civilizada. Tão qualificadora do indivíduo e da sociedade é essa vocação para os misteres da Ciência que o Magno Texto Federal abre todo um autonomizado capítulo para prestigiá-la por modo superlativo (capítulo de nº IV do título VIII). A regra de que 'O Estado promoverá e incentivará o desenvolvimento científico, a pesquisa e a capacitação tecnológicas' (art. 218, caput) é de logo complementada com o preceito (§ 1º do mesmo art. 218) que autoriza a edição de normas como a constante do art. 5º da Lei de Biossegurança. A compatibilização da liberdade de expressão científica com os deveres estatais de propulsão das ciências que sirvam à melhoria das condições de vida para todos os indivíduos. Assegurada, sempre, a dignidade da pessoa humana, a CF dota o bloco normativo posto no art. 5º da Lei 11.105/2005 do necessário fundamento para dele afastar qualquer invalidade jurídica (Ministra Cármen Lúcia)." (ADI 3.510, rel. min. Ayres Britto, julgamento em 29-5-2008, Plenário, DJE de 28-5-2010.)

Também não se pode deixar de reconhecer que as instituições que representam os três Poderes sempre devem apoiar, garantir e incentivar o desenvolvimento científico e tecnológico, pois, as empresas de tecnologia, hoje representadas pela *Amazon, Microsoft*, Meta (*Facebook-Instagran-WhatsApp*), *Apple, Google*, Franquias de tecnologia, dentre outras, movimentam o mercado financeiro, geram tecnologias inovadoras, se apresentam como plataforma de negócios, superam barreiras físicas e ainda, promovem representatividade internacional, devendo assim se manter (VALVERDE, 2023), no entanto, sempre respeitando os Direitos Humanos e Constitucionais envolvidos.

Portanto, quanto aos aspectos de proteção legal devem acompanhar a realidade, de forma a conscientizar-se de que a dimensão da tutela estatal mudou, devendo estar sempre preparada a desempenhar a tutela necessária às inovações tecnológicas, para tanto, devendo buscar contribuições dos ramos e diversas áreas do Direito, para sempre encontrar caminhos que garantam a segurança jurídica necessária aos cidadãos envolvidos nesse tipo de relação 'digital', norteando as condutas praticadas.

Tais caminhos existem e a quebra da observância legal não se justifica em hipótese alguma, como exemplo, se pode citar vivências de desrespeito quanto ao Direito à privacidade ocorrido na época da Pandemia Covid 19, relativas à colheita de dados, tendo sido necessária a intervenção do Supremo Tribunal Federal, como se pode ver:

> Houve a propositura de cinco ações diretas de inconstitucionalidade (ADIs 6.387, 6.388, 6.389, 6.390 e 6.393) contra a MP n. 954 e a Procuradoria-Geral da República manifestou-se a favor da medida provisória. Contudo, o STF considerou que o panorama fático não ficou nítido, não tendo o Poder Público trazido, de forma clara, a necessidade, adequação e proporcionalidade da medida (compartilhamento de dados) com as finalidades pretendidas, violando o princípio da proporcionalidade (ou o devido processo legal em sua dimensão substantiva) que deve

> imperar na compreensão do direito à privacidade. Para a Ministra Rosa Weber, a medida provisória apenas se referiu genericamente à pandemia e não definiu 'como' e 'para que' seriam utilizados os dados, impedindo uma análise mais detida sobre sua proporcionalidade. Além disso, a medida provisória não mencionou os mecanismos de segurança da informação que seriam implementados pelo IBGE e nem estava em vigor a Lei Geral de Proteção de Dados (LGPD), que, ao menos, poderia gerar a responsabilização dos agentes pelo uso ilegítimo desses dados (STF, ADI 6.387, rel. Min. Rosa Weber, medida cautelar de 24-4-2020, referendada pelo Plenário em 7-5-2020).

Mencionar-se o posicionamento do Supremo Tribunal Federal – STF é de suma importância, haja vista que abriu um importante precedente que poderá vir no futuro a ser utilizado numa circunstância análoga, pois, embora a referida Medida Provisória tenha perdido sua vigência no mês de agosto de 2020, o fato que mais importa é reconhecer que não importa o tipo de situação nefasta ou de crise, esta não servirá para justificar toda e qualquer ação que fira Direitos Humanos.

No caso em específico, a situação da Pandemia Covid-19, não serviu a justificar houvesse abuso no compartilhamento e ausência de segurança quanto aos dados utilizados, devendo o Estado esclarecer com transparência e racionalidade suas motivações quanto às medidas que pretende verem impostas aos cidadãos, sob pena de estar sendo ferido Direito Humano decorrente de sua dignidade enquanto pessoa sujeita de direitos (RAMOS, 2021, p. 1.128).

O Estado deve garantir de forma prioritária o progresso, como se encontra preconizado na Constituição Federal nos parágrafos 1º a 4º do artigo 218 da Constituição Federal, *in verbis*:

> Art. 218 (…)
> § 1º A pesquisa científica básica e tecnológica receberá tratamento prioritário do Estado, tendo em vista o bem público e o progresso da ciência, tecnologia e inovação.
> § 2º A pesquisa tecnológica voltar-se-á preponderantemente para a solução dos problemas brasileiros e para o desenvolvimento do sistema produtivo nacional e regional.
> § 3º O Estado apoiará a formação de recursos humanos nas áreas de ciência, pesquisa, tecnologia e inovação, inclusive por meio do apoio às atividades de extensão tecnológica, e concederá aos que delas se ocupem meios e condições especiais de trabalho.
> § 4º A lei apoiará e estimulará as empresas que invistam em pesquisa, criação de tecnologia adequada ao País, formação e aperfeiçoamento de seus recursos humanos e que pratiquem sistemas de remuneração que assegurem ao empregado, desvinculada do salário, participação nos ganhos econômicos resultantes da produtividade de seu trabalho.

Através do cumprimento de todos os aspectos legais e em específico de tais ditames constitucionais o Estado garantirá o apoio e recursos necessários para que o desenvolvimento científico e tecnológico contínuos ao processo de industrialização, para que haja de forma ininterrupta a oferta de produtos e/ou serviços ao mercado com preços competitivos que garantam sempre a superação de crises, próprias de países em desenvolvimento, como é o caso do Brasil, viabilizando negócios internacionais e nacionais, contribuindo para o crescimento do país.

Portanto, o conhecimento dos principais ditames constitucionais se apresenta de suma importância e premente, como a base estrutural legal para que se contextualize a situação que se apresenta relativa ao reconhecimento facial, sua utilização nos mais variados setores da vida humana e sua regulamentação.

5.2. O Reconhecimento Facial e a tutela de Direitos quanto ao risco de sua utilização

Inegável é o fato de que a realidade atual normalmente impõe por não haver outro tipo de opção ou recurso o uso contínuo de tecnologias e algumas delas com a utilização do reconhecimento facial, que ganha por parte de várias empresas que utilizam tecnologias cada vez mais adeptos do seu uso. A expansão do uso da tecnologia do reconhecimento facial, cresceu no país principalmente em 2019 (IDEC, 2020), sendo muito utilizada nas áreas de segurança pública, educação e no comércio, muitas vezes gerando violação à privacidade dos cidadãos e discriminação.

A prática do reconhecimento facial pode ser obtida através de uma técnica biométrica em que os *softwares* codificam o rosto humano. Para realizar o mapeamento, os sistemas utilizam as características do rosto de uma pessoa, como o tamanho do queixo e a distância entre os olhos, que são denominadas de pontos nodais, atualmente, a face humana possui cerca de 80 (oitenta) pontos nodais e a extração de cada ponto vai formando a assinatura facial e é armazenada em um banco de dados (CHARÃO, 2018).

Diante dessa realidade inconteste, a possibilidade de lesões a direitos é inevitável (OLIVEIRA, 2021, 272p.) e ao mesmo tempo inaceitável, principalmente por ferirem Direitos Humanos e Direitos da Personalidade, gerando prejuízos morais e pecuniários, que acabaram por levar o legislador a tratar do tema.

Dentre outros, atualmente se pode apontar como riscos trazidos pela utilização do reconhecimento facial o abuso dos mais variados direitos e controle de dados, vez que a imagem ou dados da face são pessoais e biométricos, sendo reconhecidos pela legislação como dados sensíveis; tais dados se compartilhados com autoridades policiais e governamentais, coloca os cidadãos diante de um sistema imposto de vigilância e sua utilização sem controle e finalidades definidas a lesar Direitos Humanos, pois, poderá gerar discriminação por raça, gênero e etnia, invasão de privacidade, reconhecimento falho de emoções e possibilidade de provocar incidentes de segurança, podendo levar ao caos social, ao invés de auxiliar, como é o propósito das mais variadas tecnologias inovadoras.

Tais riscos, não se apresentam de forma diferente no que diz respeito à iniciativa privada, pelo contrário, a utilização da tecnologia do reconhecimento facial por pessoas jurídicas de direito privado gera discriminação, é utilizada para gerar tanto direta quanto indiretamente, negação de bens ou serviços, assim como afeta a variação de preços quanto a estes por condições desvantajosas ofertadas.

Em todos esses casos, pela afetação a direitos essenciais à vida dos cidadãos com a devida qualidade, fato que leva a segurança e paz no uso de tecnologias inovadoras ou não, é necessário, no caso da utilização do reconhecimento facial que haja um mecanismo que permita a conscientização humana quanto ao recurso utilizado e por sua vez, que esta leve a obtenção livre e consciente do consentimento prévio para que possa ser utilizada.

Compreender quais direitos dos cidadãos poderão ser afetados ou lesados, leva a demanda por análise concreta ou prática de fatos já ocorridos e que causaram lesões a direitos e torna possível reconhecer a necessidade de limites legais ao uso do reconhecimento facial, fazendo com que haja a melhoria das normas legais já existentes e a elaboração de novas legislações que permitam a devida tutela a tais direitos.

A LGPD - Lei Geral de Proteção de Dados Pessoais, Lei nº 13.709, de 14 de agosto de 2018 (BRASIL, 2023) tratou de regulamentar, de forma geral a temática que ainda, no Brasil não possui legislação específica, tratando de exigir uma maior transparência.

Além da legislação mencionada, o IDEC - Instituto de Defesa do Consumidor e o InternetLab lançaram um guia de boas práticas a empresas que adotam a tecnologia de reconhecimento facial, cada vez

mais disseminada nos setores público e privado, que pode ser acessada gratuitamente através do link https://idec.org.br/sites/default/files/reconhecimento_facial_diagramacao_digital_2.pdf

Segundo os institutos envolvidos, o documento se originou de um relatório, produzido de forma colaborativa pelo InternetLab e pelo Idec, para buscar introduzir boas práticas que possam nortear o setor privado no desenvolvimento de suas atividades, no que diz respeito ao oferecimento de produtos e serviços com base em tecnologias de reconhecimento facial.

O relatório oferece um panorama das principais questões ligadas à utilização de tecnologias de reconhecimento facial por pessoas jurídicas de direito privado no Brasil, apresentando as características básicas de funcionamento dessas ferramentas e alguns conceitos necessários para a compreensão da discussão, pois, a adoção de boas práticas na utilização de tecnologias de reconhecimento facial é uma necessidade ética e legal para as empresas que pretendem promover a inovação de forma responsável.

No entanto, o guia elaborado se direcionou ao setor privado, restando o setor público ausente de diretrizes ou mesmo legislação específica que regulamente de forma efetiva sua utilização e as consequências caso haja lesão a direitos dos cidadãos e mesmo das empresas.

No referido guia, restou evidenciada oito práticas que se considerou serem recomendadas ao setor privado quando utilizar o reconhecimento facial, que são:

I. Análise de proporcionalidade e respeito a princípios

Antes do uso de qualquer sistema de reconhecimento facial, é indicado que a empresa avalie se essa é a única forma de atingir seu objetivo e se as finalidades da coleta estão de acordo com a legislação vigorante no país.

II. Transparência aos titulares

Tal transparência consistente em prestar informações completas sobre a utilização de dispositivos para coleta, quais dados serão coletados, qual a forma de tratamento e as finalidades para qual está sendo realizado; informações sobre prazo, condições de armazenamento, medidas de segurança e hipótese de compartilhamento com terceiros e para qual finalidade se dará.

III. Transparência pública

As práticas adotadas na implementação e execução da tecnologia devem ser documentadas em relatórios de impacto à proteção de dados pessoais e disponíveis a serem apresentados quando necessário e legalmente previsto.

IV. Consentimento

Consentimento ou anuência quanto à coleta, utilização e tratamento dos dados, obtido de forma livre, esclarecido e portanto, consciente, a ser fornecido pelo titular, de forma expressa, antes do início da captura dos dados faciais.

V. Local de uso das câmeras

É indicado que sejam instaladas em locais acessível a obtenção do consentimento prévio.

VI. Medidas antidiscriminatórias

A tecnologia não deve ser usada para negação de bens e serviços, variação de preços ou oferecimento de condições desvantajosas, visto que o reconhecimento facial tem falhas em detectar diferentes perfis de pessoas. Essas e outras situações devem ser monitoradas e acompanhadas por quem faz uso da tecnologia do reconhecimento facial, pois, a utilização contínua de tecnologia tem levado ao

aparecimento de situações constrangedoras aos seres humanos, provocando os mais variados tipos de lesões a direitos.

VII. Exclusão, anonimização e proteção de dados biométricos

Imagens devem ser excluídas depois de um período e dados devem ser anonimizados, para que não sejam facilmente identificados a uma pessoa, como determina a Lei Geral de Proteção de Dados Pessoais.

VIII. Crianças e adolescentes

Por se tratar de vulneráveis por sua condição quanto à idade, o reconhecimento facial não pode ocorrer, exceto se consentido pelos pais ou responsável legal.

Como se pode constatar as recomendações feitas à iniciativa privada, não se apresentam com impossibilidades de adoção por parte da iniciativa pública, vez que procura de modo geral balizar as condições de utilização da tecnologia do reconhecimento facial nos termos do que a própria legislação infraconstitucional já determinou.

Mesmo havendo tais cuidados quanto à condição do uso da tecnologia do reconhecimento facial lesões e danos provocados aos cidadãos não estão de todo afastados, podendo ocorrer incidentes de segurança; por esta razão, não se deve desprezar critérios de gestão que levem a decisões melhores, por serem seguras, quanto à sua utilização (ARMSTRONG, 2019, 301p.).

Caso venham a ocorrer, ressalta-se que todo e qualquer incidente deve ser comunicado aos titulares de dados e às autoridades públicas para que providências sejam tomadas no sentido de se coibir e evitar consequências danosas ou ainda maiores do que as que no momento se admite ter havido.

Atualmente o órgão governamental responsável por responsável por zelar, implementar e fiscalizar o cumprimento da LGPD; alterar procedimentos; criar e gerenciar canais de atendimento permitindo que o público registre reclamações sobre empresas que estão atuando em desconformidade com a lei; aplicar sanções previstas em lei quando há a ocorrência de lesões a direitos pelo mau uso de dados pessoais e manter contato com órgãos internacionais da mesma natureza, para que possa o país estar alinhado com regramentos estrangeiros e utilizar de experiências obtidas através do ocorrido com estrangeiros é a ANPD – Autoridade Nacional de Proteção de Dados, cujos serviços podem ser obtidos através do site https://www.gov.br/anpd/pt-br que possui acesso livre e gratuito aos cidadãos.

Caso haja a extrapolação pelo mau uso da tecnologia do reconhecimento facial, sob a égide dos ditames legais, tanto constitucionais, como infraconstitucionais e não se consiga uma solução à controvérsia pelos órgãos responsáveis e de forma amigável, haverá que se judicializar a questão, devendo ser levada ao Poder Judiciário para que decida a favor de serem restabelecidos os direitos dos cidadãos lesados, com as devidas e proporcionais sansões e reparação por parte dos responsáveis.

5.3. O reconhecimento facial e a legislação brasileira

Além das normas constitucionais já mencionadas, que trataram de dar a base, o fundamento para que as condutas envolvendo tecnologia fossem consagradas em normas infraconstitucionais, temos, neste aspecto a LGPD – Lei Geral de Proteção de Dados Pessoais, Lei nº 13.709, de 14 de agosto de 2018, que trata, de modo geral as questões envolvendo possibilidades de violação quanto ao uso de dados, sendo sua tutela a existente a ser aplicada, no Brasil, caso possa haver danos pela utilização do reconhecimento facial.

Logo no seu artigo primeiro se propõe a determinar seus objetivos, assim dispondo:

> Art. 1º Esta Lei dispõe sobre o tratamento de dados pessoais, inclusive nos meios digitais, por pessoa natural ou por pessoa jurídica de direito público ou privado, com o objetivo de proteger os direitos fundamentais de liberdade e de privacidade e o livre desenvolvimento da personalidade da pessoa natural.

Restando claro os princípios que devem servir de fundamento para qualquer assunto ou tema envolvendo a proteção de dados pessoais, sendo eles o princípio da liberdade (art. 5º e inciso I, ambos da CF/88), da privacidade (art. 5º, inciso XII, da CF/88) e do livre desenvolvimento da pessoa natural, que está diretamente vinculado ao princípio fundamental da dignidade da pessoa humana (art. 1º, inciso III, CF/88).

Referente à proteção de dados pessoais na Constituição, embora não esteja presente de forma específica, esta ausência não afeta a possibilidade de tutela, pois, implicitamente a Constituição Federal possibilita que direitos relativos à utilização de dados pessoais sejam protegidos através dos princípios nela consagrados, principalmente o Princípio da Dignidade da Pessoa Humana, princípio este fundante, como se explica:

> ...por um lado, a privacidade é encarada como um direito fundamental, as informações pessoais em si parecem, a uma parte da doutrina, serem protegidas somente em relação à sua 'comunicação', conforme art. 5º, XII, que trata da inviolabilidade da comunicação de dados. Tal interpretação traz consigo o risco de sugerir uma grande permissividade em relação à utilização de informações pessoais. Nesse sentido, uma decisão do STF, relatada pelo Ministro Sepúlveda Pertence, reconheceu expressamente a inexistência de uma garantia de inviolabilidade sobre dados armazenados em computador com fulcro em garantias constitucionais... O sigilo, no inciso XII, art. 5º, está referido à comunicação, no interesse da defesa da privacidade... Obviamente o que se regula é comunicação por correspondência e telegrafia, comunicação de dados e telefônica... A distinção é decisiva: o objeto protegido no direito à inviolabilidade do sigila não são os dados em si, mas a sua comunicação restringida (liberdade de negação). A troca de informações (comunicação) privativa é que não pode ser violada por sujeito estranho... A decisão tem sido, desde então, constantemente mencionada como precedente em julgados nos quais o STF identifica que a natureza fundamental da proteção de dados está restrita ao momento de sua comunicação. (DONEDA, 2006, p. 262)

Além dos princípios constitucionais norteadores, os princípios e demais regramentos estabelecidos pela LGPD devem ser seguidos não somente pelos cidadãos e em seu favor, pelas pessoas jurídicas de Direito Privado, mas inclusive, pelos entes da Federação, assim, estando União, Distrito Federal, Estados Membros e Municípios, sendo de seus respectivos interesses, conforme reforça o parágrafo único do artigo primeiro.

Sob a égide de tais princípios se vislumbra nortear condutas daqueles responsáveis por lidar com dados pessoais, tanto aqueles que são públicos, como aqueles que a própria legislação os considera como sendo sensíveis, o reconhecimento facial está dentre os dados tidos como sensíveis.

Dentro do Direito Constitucional, o constitucionalismo digital é uma realidade por cuidar de estudar de que maneira as tecnologias da informação afetam a sociedade e, portanto, o Estado Democrático de Direito, estando relacionado a possibilidade de lesões provocadas a direitos, tais como: a liberdade de expressão; privacidade na *internet;* proteção de dados pessoais; neutralidade da rede; atuação Estatal nos vários tipos de proteção legal necessários; atribuição de responsabilidades às empresas desenvolvedoras de tecnologia; fiscalização na liberação de uso de tecnologias emergentes e acompanhamento das consequências de seu uso (O'Neil, 2020, 339p.); combate a desinformação e *Fake News* (PRADO, 2022, 424p.)*;* dentre outros.

Para tanto, estabeleceu em seu art. 2º, os fundamentos, os quais, as condutas e decisões devem ser pautados, sendo eles:

I - o respeito à privacidade;

II - a autodeterminação informativa;

III - a liberdade de expressão, de informação, de comunicação e de opinião;

IV - a inviolabilidade da intimidade, da honra e da imagem;

V - o desenvolvimento econômico e tecnológico e a inovação;

VI - a livre iniciativa, a livre concorrência e a defesa do consumidor; e

VII - os direitos humanos, o livre desenvolvimento da personalidade, a dignidade e o exercício da cidadania pelas pessoas naturais.

Ao estabelecer tais fundamentos, procurou limitar condutas abusivas que poderiam vir a ferir à dignidade humana, através da invasão à privacidade de dados pessoais, utilizando de interesses particulares, quer advenham do Estado ou de pessoas jurídicas de Direito Privado ou de cidadãos mal-intencionados, seguindo o rol de Liberdades Públicas previstas no art. 5º, *caput,* da Constituição Federal.

A delimitação quanto a maneira de se fundamentar e lidar com tais dados não se restringe ao rol mencionado, mas cuidou o legislador da questão espacial, ou seja, quanto ao espaço ou limites territoriais abrangidos pela atual legislação, dispondo quanto a este aspecto no seu art. 3º.

Restou quanto a delimitação espacial ou territorial que a LGPD se aplica a qualquer operação de tratamento realizada por pessoa natural ou por pessoa jurídica de direito público ou privado, independentemente do meio, do país de sua sede ou do país onde estejam localizados os dados, desde que a operação de tratamento seja realizada no território nacional; a atividade de tratamento tenha por objetivo a oferta ou o fornecimento de bens ou serviços ou o tratamento de dados de indivíduos localizados no território nacional; ou os dados pessoais objeto do tratamento tenham sido coletados no território nacional.

Para tanto, considerou que são coletados no território nacional os dados pessoais cujo titular nele se encontre no momento da coleta (§1º, do art. 3º, da LGPD), apontando como exceção a operação de tratamento ser realizada no território nacional (§2º, do art. 3º, da LGPD), os dados provenientes de fora do território nacional e que não sejam objeto de comunicação, uso compartilhado de dados com agentes de tratamento brasileiros ou objeto de transferência internacional de dados com outro país que não o de proveniência, desde que o país de proveniência proporcione grau de proteção de dados pessoais adequado ao previsto nesta Lei (inciso IV, do art. 4º, da LGPD).

Como já mencionado anteriormente, a utilização do reconhecimento facial, por ser um dado biométrico, deve receber o tratamento quanto aos dados coletados nos termos predeterminados na LGPD, por se tratarem e enquadrarem como dados sensíveis, pois, no art. 5º, inciso II, a definição é clara, tendo sido estabelecido que é considerado um dado pessoal sensível, o dado pessoal sobre origem racial ou étnica, convicção religiosa, opinião política, filiação a sindicato ou a organização de caráter religioso, filosófico ou político, dado referente à saúde ou à vida sexual, dado genético ou biométrico, quando vinculado a uma pessoa natural.

Além da LGPD, o Código Civil (BRASIL, 2023), instituído a partir da Lei nº 10.406, de 10 de janeiro de 2002, estabeleceu na Parte Geral, no Livro I, intitulado Das Pessoas, no Título I, denominado Das Pessoas Naturais, no Capítulo II nomeado Direitos da Personalidade, nos seus artigos 11 a 21, tais direitos, como sendo de ordem física, psíquica e moral, estabelecendo condutas de respeito ao corpo e a partes dele, à imagem, a voz, dentre outros, devendo ser observados para que não haja lesão a tais direitos, sendo eles ão

intransmissíveis e irrenunciáveis, não podendo o seu exercício sofrer limitação voluntária (art. 11, do Código Civil).

No art. 20, do Código Civil, há mais precisão com relação à tutela legal estabelecida, quando se dispôs que com exceção de autorizadas ou se necessárias à administração da justiça ou à manutenção da ordem pública, a divulgação de escritos, a transmissão da palavra, ou a publicação, a exposição ou a utilização da imagem de uma pessoa poderão ser proibidas, a seu requerimento e sem prejuízo da indenização que couber, se lhe atingirem a honra, a boa fama ou a respeitabilidade, ou se se destinarem a fins comerciais.

A legislação estabelece as exceções e possibilidades a previsões abstratas de lesões a Direitos da Personalidade, no entanto, parametriza sua utilização, vedando qualquer uma que se furte a administração da justiça ou à manutenção da ordem pública, aliado a tais imposições, no caso do reconhecimento facial se deve ponderar que sua utilização mesmo observando tais finalidades poderá vir a causar lesões que atingem a dignidade da pessoa humana, principalmente quanto ao uso deturpado desses dados, seu vazamento ou apropriação para fins outros, através do seu vazamento, mesmo que praticadas de forma involuntária.

Há que se atentar ainda que mesmo tendo sido o direito à proteção de dados pessoais especificado quanto ao seu titular ou detentor desses direitos nos artigos 17 e 18, da LGPD, não se pode desprezar o fato de que as diferenças sociais, culturais, econômicas, dentre outras, enfrentadas pela população brasileira leva a crer que muitos desconhecem seus direitos, também se mantem alienados quanto às tecnologias, seu uso e consequências individuais e gerais que poderão ser provocadas em termos de lesões, o que torna a lei desprovida de um dos critérios de validade, que é o fundamento social, importante para que seja legitimada.

O rol estabelecido em tais dispositivos legais não se apresenta como taxativo, podendo ser contemplados outros direitos, em outras normas, assim, no artigo 17, se assegura que toda pessoa natural tem assegurada a titularidade de seus dados pessoais e garantidos os direitos fundamentais de liberdade, de intimidade e de privacidade, nos termos da LGPD.

De maneira que, segundo o artigo 18, do mesmo estatuto legal mencionado, tenha direito a obter do controlador, em relação aos seus dados por ele tratados, a qualquer momento e mediante requisição:

I - Confirmação da existência de tratamento;

II - Acesso aos dados;

III - Correção de dados incompletos, inexatos ou desatualizados;

IV - Anonimização, bloqueio ou eliminação de dados desnecessários, excessivos ou tratados em desconformidade com o disposto nesta Lei; não incluindo, neste caso, dados que já tenham sido anonimizados pelo controlador (§7º, do art. 18, da LGPD).

V - Portabilidade dos dados a outro fornecedor de serviço ou produto, mediante requisição expressa, de acordo com a regulamentação da autoridade nacional, observados os segredos comercial e industrial;

VI - Eliminação dos dados pessoais tratados com o consentimento do titular, com exceção das hipóteses previstas no art. 16, ou seja, quando os dados pessoais ao serem após o término de seu tratamento, no âmbito e nos limites técnicos das atividades, terem autorizada a sua conservação para a finalidade de cumprimento de obrigação legal ou regulatória pelo controlador; para a realização de estudo por órgão de pesquisa, garantida, sempre que possível, a anonimização dos dados pessoais; quando houver

a transferência a terceiro, desde que respeitados os requisitos de tratamento de dados dispostos nesta Lei; ou por uso exclusivo do controlador, vedado seu acesso por terceiro, e desde que anonimizados os dados.

VII - Informação das entidades públicas e privadas com as quais o controlador realizou uso compartilhado de dados;

VIII - Informação sobre a possibilidade de não fornecer consentimento e sobre as consequências da negativa.

Para tanto, deve ser disponibilizado o Termo de Consentimento para uso de Imagem (adendo II e III), no caso de utilização de reconhecimento facial, onde haverá de ser discriminada a forma de tratamento desses dados, considerados sensíveis, se observando as exigências da LGPD.

IX - Revogação do consentimento, que pode se dar a qualquer momento mediante manifestação expressa do titular, por procedimento gratuito e facilitado, ratificados os tratamentos realizados sob amparo do consentimento anteriormente manifestado enquanto não houver requerimento de eliminação, como disposto no § 5º do art. 8º, da LGPD.

Com relação a tais direitos do titular, nos termos do § 1º, do art. 18, tem o direito de peticionar em relação aos seus dados contra o controlador perante a autoridade nacional e organismos de defesa do consumidor (§8º, do art. 18), o que significa poder judicializar as controvérsias existentes, reafirmando o direito do acesso à Justiça, previsto na Constituição Federal, no art. 5º, inciso XXXV, que determina que a lei não excluirá da apreciação do Poder Judiciário lesão ou ameaça a direito.

Nos termos do § 2º, do art. 18, se estabeleceu ainda que o titular dos dados pode opor-se a tratamento realizado com fundamento em uma das hipóteses de dispensa de consentimento, em caso de descumprimento das previsões legais estabelecidas na Lei Geral de Proteção de Dados Pessoais.

Estabelece ainda, que os direitos previstos no art. 18, serão exercidos mediante requerimento expresso do titular ou de representante legalmente constituído, sem custos, a agente de tratamento e que, caso haja a impossibilidade de tal providência ser tomada que o controlador envie em resposta, comunicação de que não é agente de tratamento dos dados e indicar, sempre que possível, o agente; ou indicar as razões de fato ou de direito que impedem a adoção imediata da providência, conforme estabelecido nos parágrafos 3º, 4º e 5º do art.18, da LGPD.

O responsável deverá informar, de maneira imediata, aos agentes de tratamento com os quais tenha realizado uso compartilhado de dados a correção, a eliminação, a anonimização ou o bloqueio dos dados, para que repitam idêntico procedimento, exceto nos casos em que esta comunicação seja comprovadamente impossível ou implique esforço desproporcional (§6ª, do art. 18, da LGPD).

Com relação ao exercício de tais direitos elencados nos arts. 17 e 18 da LGPD, o art. 20 estabelece que o titular dos dados tem direito a solicitar a revisão de decisões tomadas unicamente com base em tratamento automatizado de dados pessoais que afetem seus interesses, incluídas as decisões destinadas a definir o seu perfil pessoal, profissional, de consumo e de crédito ou os aspectos de sua personalidade.

Devendo o controlador fornecer, sempre que solicitadas, informações claras e adequadas a respeito dos critérios e dos procedimentos utilizados para a decisão automatizada, observados os segredos comercial e industrial. Em caso de não oferecimento de tais informações, baseado na observância de segredo comercial e industrial, a autoridade nacional poderá realizar auditoria para verificação de aspectos discriminatórios em tratamento automatizado de dados pessoais, conforme disposto nos parágrafos 1º e 2º, do art. 20, da LGPD.

Sendo ainda que no seu art. 21 e 22 há determinado que os dados pessoais referentes ao exercício regular de direitos pelo titular não podem ser utilizados em seu prejuízo e a defesa dos interesses e dos direitos dos titulares de dados poderá ser exercida em juízo, individual ou coletivamente, na forma do disposto na legislação pertinente, acerca dos instrumentos de tutela individual e coletiva.

Como se pode constatar a proteção a dados pessoais no Brasil, se dá através da LGPD e o reconhecimento facial, por se tratar de um dado pessoal sensível se acha englobado e regulamentado, mesmo que de maneira superficial, não sendo diferente com a utilização de outras tecnologias.

Tal realidade se dá ante ao fato de que tais tecnologias inovam de forma constante, tendo seus programas atualizados continuamente e, obviamente trazendo outras maneiras de serem utilizados, concebidos e com possibilidades de controle humano cada vez maior, tanto no que diz respeito ao controle de uso da tecnologia, como de controle das ações humanas, haja vista, estas interessarem a governos, ao mercado econômico, financeiro e de consumo, o que leva a abusos e lesões a direitos.

6. EXEMPLOS QUE LEVAM A PREOCUPAÇÃO DO USO DO RECONHECIMENTO FACIAL

A utilização do reconhecimento facial é um fato que não pode ser desprezado, pois, se amplia cada vez mais, trazendo muitas vezes preocupação pela extrapolação, levando a lesões de direitos quanto aos dados pessoais que pode provocar, já tenha acontecido ou está sendo vivenciado por alguns cidadãos em tempo real; assim, manifestando poder ocorrer e se apresentando como possível nas dimensões temporais existentes.

6.1. A utilização da tecnologia do reconhecimento facial e as empresas privadas

Como sistema de vigilância, o reconhecimento facial foi utilizado recentemente na Copa do Mundo, em Qatar, tendo se dado dentro e fora de campo de futebol, sendo que no seu interior os jogos contaram com um reforço a mais para aumentar a agilidade e a eficácia nesse torneio, o chamado “impedimento semiautomático”, que utiliza a tecnologia de inteligência artificial com a função de diminuir o tempo médio em que será decidido um lance de impedimento (CHAVES, 2022).

Fora do campo, a tecnologia do reconhecimento facial foi realizada durante os jogos, focada nas pessoas que assistiam aos jogos da copa, enquanto expectadores dos jogos; por outro lado, foram assistidas e vigiadas, levando os interessados pela questão jurídica a refletirem sobre a falta de privacidade dessas pessoas, que por medida de proteção dos responsáveis pela organização, como precaução tiveram que se submeter à vigilância.

O fato controverso não fica adstrito às 22.000 (vinte e duas mil) câmeras equipadas com tecnologia de reconhecimento facial espalhadas pelas oito arenas em Qatar, monitoradas diuturnamente pelos mais de 100 (cem) técnicos que trabalhavam no Centro de Comando e Controle Aspire, invadindo à privacidade das pessoas ali presentes.

Criar-se o sistema de vigilância por reconhecimento facial, em Qatar teve por escopo, além de vigiar os torcedores via geolocalização, já que para entrar no país era necessário possuir o aplicativo *Hayya Card,* cujo acesso se dava através do link https://www.mofa.gov.qa/en/qatar/2022-fifa-world-cup/hayya-card , que tinha a mesma função de um visto, só que no celular para que pudesse ser realizado o

compartilhamento de localização; também, aumentar a segurança e prevenir contra a ação de terroristas e *hooligans* (RODRIGUES, 2023).

Na análise jurídica não cabe tecer elogios ao reconhecimento facial, pois, suas qualidades e benesses falam por si próprias, tanto que seus adeptos são inúmeros e a tecnologia tem despontado no dia a dia fazendo com que outros adiram a ela; porém, o que incomoda aos profissionais da área jurídica é a necessidade de tutela legal a direitos que não podem e nem devem ser lesionados pela utilização de qualquer tipo de tecnologia inovadora e principalmente aquelas que para servirem ao propósito para o qual foram idealizadas necessitam fazer uso de dados pessoais.

Por sua vez, o legislador pátrio não consegue em tempo real, regulamentar as questões oriundas da utilização massiva da tecnologia; diante de tal realidade os critérios dependem de uma interação entre a utilização de Inteligência Artificial e tudo que há em seu entorno e o Direito (PEIXOTO e DA SILVA, 2019, 35-45p.), a partir da ética, pois, somente pela observância de critérios éticos será possível nortear condutas que estejam deficitárias ou com ausência de regulamentação legal.

Notório é o histórico de casos envolvendo controvérsias e questionamentos relacionados a utilização do reconhecimento facial, por exemplo, em 2018, a concessionária da Linha 4 Amarela do metrô de São Paulo, a ViaQuatro, instalou câmeras para identificar emoções no metrô da capital.

Tais câmeras foram dispostas acopladas aos painéis de publicidade, com o objetivo de efetuar o reconhecimento das expressões das pessoas diante de determinado anúncio publicitário; o que levou o IDEC Instituto Brasileiro de Defesa do Consumidor, promover uma ação civil pública, por ser a prática pouco transparente e operacionalizada sem a opção de consentimento do cidadão, estando em desconformidade com regras então vigentes (IDEC, 2021). A ação teve como resultado uma decisão liminar que determinou o desligamento das câmeras.

A defesa da ViaQuatro, se ateve a mencionar que o sistema instalado não possuía recurso ou dispositivo que permita o reconhecimento facial e não dispunha de qualquer tecnologia que viabilizasse a identificação ou armazenamento de dados pessoais ou imagens, afirmando que há uma diferença entre sistema de detecção e de reconhecimento facial.

Em 10 de maio de 2021, na 37ª Vara Cível de São Paulo, houve o desfecho da ação, com a condenação da ViaQuatro ao pagamento de uma multa no valor de R$ 100 mil a título de indenização pela coleta irregular de dados de passageiros por meio de reconhecimento facial (VICENTIN, 2001).

A questão suscitada a partir do caso envolvendo a ViaQuatro, abriu precedente para que houvesse outras investigações e processos, gerando para as empresas notificações por parte das autoridades responsáveis por zelar do Direito do Consumidor.

A loja *Hering* localizada no *Shopping* Morumbi, na Capital São Paulo, adotou um sistema de câmeras que permitia a coleta de dados sobre o gênero dos seus clientes ou pretensos/interessados clientes sem a devida transparência e informações detalhas aos clientes, a coleta de dados também era composta de faixa etária e humor daquelas pessoas. Após tomar conhecimento da prática contrária ao direito consumerista, através de uma notificação do IDEC, a Secretaria Nacional do Consumidor - SENACON, órgão vinculado ao Ministério da Justiça, aplicou-lhe uma multa no valor de R$ 58.767,00 (cinquenta e oito mil setecentos e sessenta e sete reais) por conduta abusiva na aplicação da tecnologia (IDEC, 2020). O valor da multa, foi destinado ao Fundo de Defesa de Direitos Difusos - FDDD.

Outras instituições privadas, como por exemplo a Taxi 99; o Banco Itaú S/A.; o Banco do Brasil; Quod, Zaitt, etc, como se vê os riscos e problemas são muitos, pela falta de consentimento e ausência de transparência que acabam por provocar lesão a direitos, no entanto, o uso do reconhecimento facial parece ser um caminho sem volta e, resta diante dessa realidade buscar-se um equilíbrio e integração, que acredita-se seja

possível, não somente pela edição de leis, mas pelo estabelecimento de critérios éticos a partir dos ditames legais.

Órgãos como o IDEC e outro organismos nacionais e internacionais também podem auxiliar, para tanto, o que se deve é encontrar um caminho de respeito às leis vigentes, estabelecer critérios éticos e criar a partir de então uma cultura para que a utilização do reconhecimento facial se dê de maneira a não provocar lesões.

Com relação a tal proposta, o pacto ou contrato social (ROUSSEAU, 2017, 128p.), que há séculos estabeleceu princípios do Direito Político, serve de norteador, assim como as lições herdadas de Jean-Jacques Rousseau, filósofo e escritor franco-suíço, poderá lembrar às sociedades, que há como parametrizar condutas a partir de situações semelhantes, outrora vivenciadas, que por analogia poderá direcionar pessoas físicas e jurídicas, visando o bem dos cidadãos, principalmente daqueles tipos por vulneráveis, seja pela razão que for.

Anteriormente a Rosseau, já havia por parte do pensamento de São Tomás de Aquino (2020, p.173), a menção de que "O direito de modo simples é o direito político. O direito político é aquele que existe em uma comunidade que se ordena a que haja suficiência das coisas que pertencem à vida humana. Tal comunidade é a cidade, na qual devem encontrar todos o que é suficiente à vida humana."

O que não se pode perder de vista jamais é o fato de que o desenvolvimento científico e tecnológico é pujante, desde o surgimento da humanidade e, as Revoluções Industriais vividas são prova dessa realidade, no entanto, são aparatos inovadores que surgem para trazer soluções, resolver questões e facilitar a vida dos seres humanos e não para causar-lhes malefícios.

No entanto, os resultados, consequências, efeitos nefastos ocorrem quando há extrapolação de limites e deturpação das finalidades para as quais foram idealizados; diante dessa conscientização cabe aos seres humanos realizar escolhas a partir dos mecanismos legais e posturas que levem os critérios éticos em consideração, para que não haja danos, lesões a direito e quem sabe à destruição da humanidade.

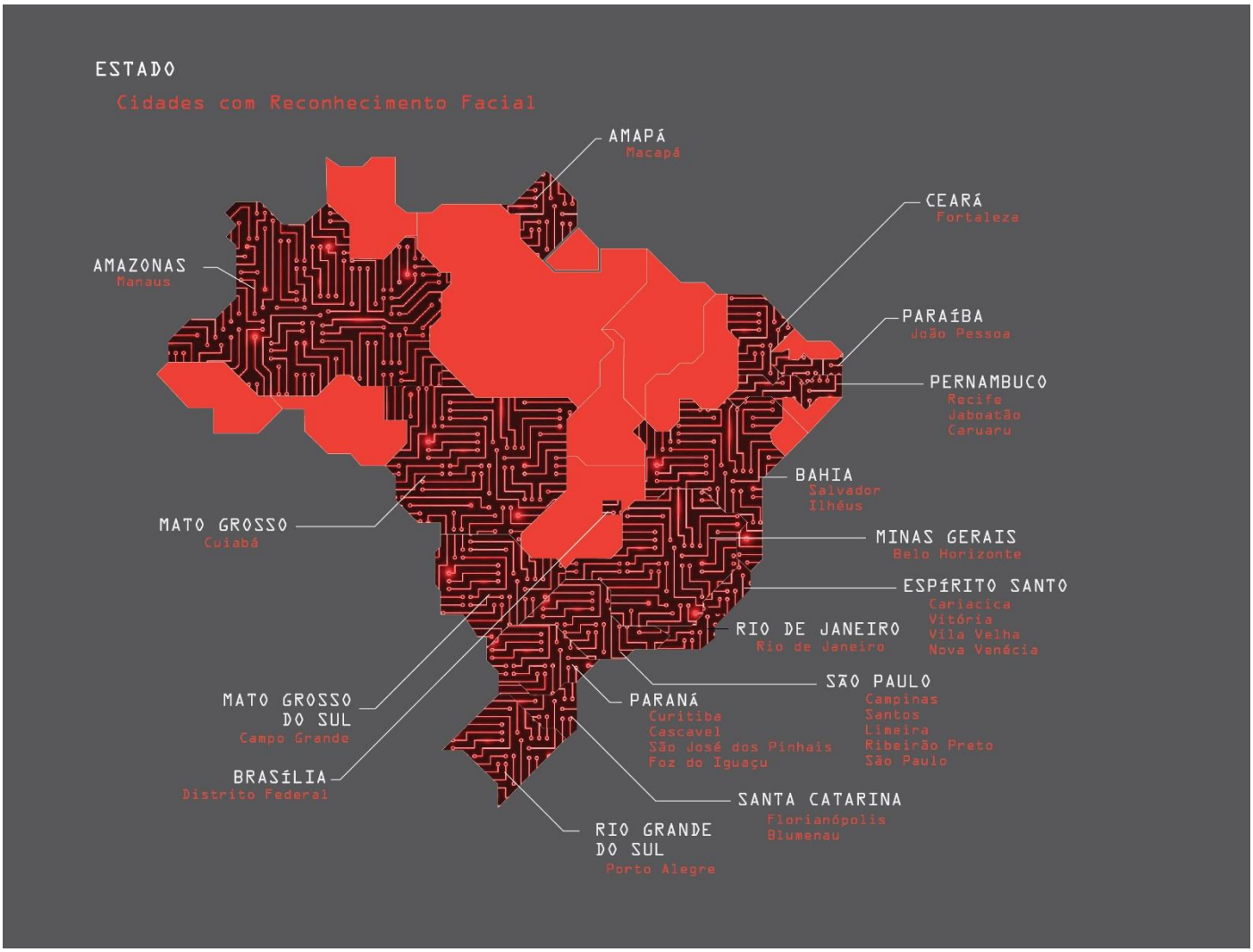
6.2. O reconhecimento facial e o setor público

A utilização da tecnologia do reconhecimento facial no setor público tem trazido desconforto aos cidadãos tanto quanto ocorre nas empresas da iniciativa privada. Tal fato se dá justamente por ser mais ampla, pois, o setor público possui mais condições, tanto com relação a existência de recursos materiais, mesmo que precariamente, quanto de recursos humanos, em específico por ter à disposição agentes de segurança, fatos que levam a ausência de transparência de seus atos para com os cidadãos, na maioria das vezes.

No setor público a utilização desse tipo de sistema permeia a área da educação, transporte coletivo e urbano, a segurança pública municipal, estadual e federal e no controle de fronteiras, fazendo parte do cotidiano do país, desde 2011, utilizando-se de parceiros do setor privado (INSTITUTO IGARAPÉ, 2019).

Na pesquisa realizada pelo Instituto Igarapé (2019), restou evidenciado a utilização do reconhecimento facial em vários Estados, mas evidenciado em 2019, em virtude de uma comitiva do PSL que foi à China, com a finalidade de adquirir a tecnologia do reconhecimento facial (FORUM, 2019); o Carnaval, do Rio de Janeiro quanto o da Bahia foram o centro de implementação de reconhecimento facial.

No Quadro 1 abaixo, é possível vislumbrar as cidades dos Estados que utilizam o reconhecimento facial no país, demonstrando que a tecnologia do reconhecimento facial está presente na vida dos cidadãos.



Fonte: Instituto Igarape, 2019.

Mesmo que a tecnologia do reconhecimento facial venha a ser utilizada na totalidade do país, tal realidade não significa ter que se admitir o desrespeito aos direitos dos cidadãos, pelo contrário, sob a observância dos princípios da transparência com relação ao tratamento dos dados pessoais utilizados; a proteção intensificada a crianças e adolescentes, consentimento livre e esclarecido das pessoas com relação a captura de imagens do rosto por câmeras, mapeamento e indicação dos locais onde câmeras estão instaladas.

Na proposta de regulação do reconhecimento facial no setor público, elaborado após uma avaliação de experiências internacionais, o Instituto Igarapé e o *Data Privacy* Brasil *Research* (2019), ressalta o que vem a ser os princípios que devem nortear a utilização do reconhecimento facial no país (FRANCISCO et. al., 2019, p.4 e 5), que se acredita devam ser os mesmos a serem utilizado pelo setor privado.

Explicam os autores Francisco et.al. (2019, p. 13-14) sobre os propósitos da Lei Geral de Proteção de Dados Pessoais, mencionando que:

> A LGPD, antes de tudo, é uma regulação de riscos. Isso porque, além de garantir direitos e prescrever princípios, ela traz dispositivos voltados aos controladores de dados pessoais para

> que documentem suas atividades, prestem contas e também calibra sanções de acordo com o nível das medidas de prevenção ao risco adotadas previamente. Por fim, trabalha com a ideia de Relatórios de Impacto à Proteção de Dados, que, embora não sejam obrigatórios, são altamente recomendáveis e podem servir de inspiração para a prática de documentação e análise de risco pelos agentes envolvidos no ecossistema do reconhecimento facial. (FRANCISCO et. al., 2019, p.13-14)

O que se reconhece é que não somente a LGPD, mas qualquer outro regulamento, talvez não atenda às necessidades dos cidadãos com relação à proteção de seus dados pessoais e possíveis lesões oriundas do uso da tecnologia, até porque incessantemente inovadora e disruptiva, no entanto, haverá, a bem do futuro dos cidadãos (WEBB, 2020, p. 241-267), que se integrar ética e direito, em ambos possibilitando resgate de valores, realizados em épocas passadas, para que o ser humano se coloque a serviço dele mesmo e as tecnologias sejam utilizadas a seu favor, sem que estejam na condição de escravos dela ou apáticos ou indiferentes sobre as mudanças.

Enfim...

Na prática cotidiana que já se reconhece, utilizará cada vez mais o reconhecimento facial é que se poderá lidar com os problemas oriundos tanto do setor privado como do setor público, se podendo estabelecer, inclusive, um diálogo entre as instituições e à sociedade, no que diz respeito aos riscos e benefícios oriundos de seu uso, não desprezando os erros, riscos e prejuízos que poderão ser judicializados em busca da reparação dos danos que venham a causar.

Não devendo os seres humanos relegar sua condição ao ponto de se tornar escravo das tecnologias, mas eticamente reconhecendo que esta é quem deve estar ao seu serviço, lhe trazendo benesses e resolvendo problemas, com a finalidade de melhorar a qualidade de vida dos cidadãos, no caso em questão, principalmente vinculado a possibilitar-lhe mais segurança.

Com relação a este aspecto, sempre se deve fomentar o debate público, aberto e transparente, para que as decisões sobre o uso do reconhecimento facial estejam eivadas de legitimidade, pois, o fundamento ético deverá validá-lo.

Essa realidade nos leva a pensar sempre em buscar a integração entre o uso do reconhecimento facial, a ética nas relações que envolvam o uso da tecnologia e o direito, buscando regulamentar através das previsões abstratas da lei sobre possibilidades de desvios de condutas e ainda, a reparar danos já causados, para que outros sejam no futuro evitados; assim se esperando que a sociedade e instituições públicas e privadas possam conduzir as decisões que afetarão não somente o que há no presente, mas o que haverá de ser encontrado e vivenciado pelas gerações futuras.

ADENDO I

TERMO DE CONSENTIMENTO PARA USO DE IMAGEM (LGPD)

O **TERMO DE CONSENTIMENTO PARA USO DE IMAGEM** é um documento ou instrumento particular no qual a pessoa física, denominada **TITULAR,** concede através de seu consentimento ao **CONTROLADOR,** o direito de utilizar a sua imagem.

O **CONTROLADOR** é a pessoa natural ou jurídica, de direito público ou privado, a quem competem as decisões referentes ao tratamento de dados pessoais (art. 5º, inciso VI, da LGPD).

O controlador que necessita utilizar dados pessoais de algum cidadão precisará possuir em mãos o fundamento jurídico que o autorize, o mais conhecido é o consentimento, que deve ser livre e esclarecido, fornecido de forma expressa, ou seja, por escrito.

Através do uso da tecnologia do reconhecimento facial há a captação da imagem do cidadão, que é um dado pessoal, considerado pela LGPD, como sendo um dado sensível devendo-se observar as disposições da Lei Federal nº 13.709/2018 - Lei Geral de Proteção de Dados - LGPD, exceto nos casos em que seu uso seja realizado por pessoa natural para fins exclusivamente particulares e não econômicos.

Também, nos casos em que seja realizado reconhecimento facial para fins exclusivamente jornalísticos, artísticos ou acadêmicos**,** sendo que, neste último caso, um outro termo deverá ser utilizado, para fins de Direitos Autorais ou cessão de uso de imagem – Lei de Direitos Autorais - Lei nº 6.910/98.

VALIDADE DO CONSENTIMENTO

O consentimento deve ser concedido pelo titular de forma livre, informada e inequívoca, concordando com o tratamento de dados pessoais que deve ser feito nos termos da LGPD, assim, deve receber informações claras e seguras a respeito do que será feito com seus dados, não devendo restar dúvidas sobre o fato de que efetivamente consentiu, ou seja, somente deve assinar o documento se compreender integralmente seu conteúdo.

Segundo a LGPD, ainda, o consentimento deve referir-se a finalidades específicas**,** não genéricas; especificando ao titular, de forma precisa, as finalidades para as quais seus dados serão tratados, como por exemplo, a utilização da imagem do titular para compor materiais publicitários de uma campanha para difusão de uma marca específica da qual o titular é representante ou ainda, no caso de campanhas internas que valorizem o trabalhador.

No caso do titular dos dados ser criança ou adolescente (art. 2º, do Estatuto da Criança e do Adolescente), os pais, o seu representante legal ou quem detém sua guarda devem, no caso de criança, por ser considerado menor impúbere, deverá ser representado, devendo estes assinarem o termo de consentimento do uso da imagem (conforme adendo III). Caso seja relativamente incapaz, deverá ser assistido por seus pais ou por seu representante legal.

Após estar integralmente preenchido, o termo de consentimento do uso de imagem, deve ser impresso e disponibilizado ao titular para assinatura, que pode ser feita pessoalmente, de próprio punho; eletrônica ou digital, com ou sem certificado digital, desde que de acordo com a lei (conforme adendo III).

O documento deve ser arquivado pelo controlador após ser assinado e uma cópia deve ser entregue ao titular que poderá ter acesso ao termo de consentimento de uso de imagem sempre que quiser ou necessitar, bastando, para tanto solicitá-lo ao controlador.

O RECONHECIMENTO FACIAL E A POLÍTICA DE PRIVACIDADE

A política de privacidade é um documento que se propõe a fornecer informações ao titular dos dados, sobre a maneira com que seus dados pessoais estão sendo utilizados, de forma generalizada faz menção às atividades de tratamento de dados pessoais, portanto, meramente informativo.

TITULARES ESTRANGEIROS E O TERMO DE CONSENTIMENTO DE USO DE IMAGEM

A utilização de dados pessoais de titulares estrangeiros pelo controlador, pode estar sujeita a regramentos internacionais, em sendo assim, o termo não deve ser utilizado.

ADENDO II

TERMO DE CONSENTIMENTO PARA USO DE IMAGEM

TITULAR DOS DADOS PESSOAIS

Nome completo, nacionalidade, estado civil, profissão, portador do RG nº _____ e do CPF nº _____, residente e domiciliado na Rua/Avenida/Praça (endereço completo), endereço eletrônico (e-mail).

CONTROLADOR

Nome completo, nacionalidade, estado civil, profissão, portador do RG nº _____ e do CPF nº _____, residente e domiciliado na Rua/Avenida/Praça (endereço completo), endereço eletrônico (e-mail).

Por meio deste Termo de Consentimento para o uso de Imagem, o TITULAR concorda de maneira livre e esclarecida, de forma inequívoca, com o tratamento de seus dados pessoais para as finalidades de _____, nos termos da Lei nº 13.709/2018 – Lei Geral de Proteção de Dados Pessoais – LGPD.

CLÁUSULA I – DOS DADOS PESSOAIS

O TITULAR, já devidamente qualificado, concorda com o tratamento de sua imagem, a ser captada pelo CONTROLADOR por meio de fotos/vídeos/outro tipo de tecnologia/reconhecimento facial.

PARÁGRAFO ÚNICO. O presente termo de consentimento para o uso de imagem abrange tão somente os dados pessoais explicitamente mencionados nesta cláusula.

CLÁUSULA II – DAS FINALIDADES PARA O USO DA IMAGEM

O TITULAR concorda com o tratamento de sua imagem para a(s) seguinte(s) finalidade(s):

(descrever a/as finalidade(s))

PARÁGRAFO 1º. O CONTROLADOR não poderá com base neste termo, utilizar dados pessoais descritos na Cláusula I para finalidades diversas das previstas nesta cláusula, sob pena de o tratamento ser considerado contrário aos ditames da LGPD.

PARÁGRAFO 2º. Sempre que pretender, com base no consentimento do TITULAR, realizar atividades de tratamento de dados pessoais não abrangidas por esta cláusula, o CONTROLADOR deverá obter novo consentimento.

CLÁUSULA III – DO NÃO CONSENTIMENTO E DAS SUAS CONSEQUÊNCIAS

O TITULAR declara para os devidos fins e direito, a quem possa interessar que, foi informado pelo CONTROLADOR de que poderia deixar de consentir com o tratamento de seus dados pessoais nos termos deste instrumento particular, também declara que recebeu tal informação antes que o presente termo fosse assinado, ficando ciente de que a negativa quanto ao seu consentimento ao tratamento dos dados lhe traria as seguintes consequências:

(descrever a/as consequência(s), como por exemplo ausência de responsabilização do CONTROLADOR em caso de vazamento de dados)

CLÁUSULA IV – DO COMPARTILHAMENTO NACIONAL

Os dados pessoais objeto do presente termo de consentimento do uso de imagem serão compartilhados com as pessoas abaixo elencadas:

(descrever e qualificar de forma completa as pessoas físicas ou jurídicas com as quais os dados serão compartilhados)

PARÁGRAFO ÚNICO. O eventual compartilhamento de dados pessoais com pessoas não mencionadas nesta cláusula somente será possível, desde que o TITULAR seja previamente informado e conceda seu consentimento de forma expressa.

CLÁUSULA V – DA SEGURANÇA DOS DADOS PESSOAIS

O CONTROLADOR adotará medidas de segurança, técnicas e administrativas aptas a proteger os dados pessoais do TITULAR de acessos não autorizados e de situações acidentais ou ilícitas de destruição, alteração, desvio, perda, comunicação ou qualquer outra forma de tratamento inadequado ou ilícito.

PARÁGRAFO 1º. O CONTROLADOR somente poderá compartilhar os dados pessoais do TITULAR, nos termos deste instrumento particular, se aquele para quem transferir assumir expressamente a obrigação de garantir a segurança dos referidos dados pessoais.

PARÁGRAFO 2º. O CONTROLADOR comunicará ao TITULAR, no prazo definido pela ANPD – Autoridade Nacional de Dados Pessoais – link https://www.gov.br/anpd/pt-br/canais_atendimento/agente-de-tratamento/comunicado-de-incidente-de-seguranca-cis , a eventual ocorrência de incidente de segurança que possa lhe acarretar risco ou dano relevante, devendo dar ao TITULAR, todo apoio e informação que necessitar.

CLÁUSULA VI – DA RESPONSABILIDADE DO CONTROLADOR E DEMAIS AGENTES DE TRATAMENTO DE DADOS PESSOAIS

O CONTROLADOR, de *per si* ou solidariamente com outros agentes de tratamento de dados pessoais, responderá(ão) perante o TITULAR, por qualquer/quaisquer dano(s) ou lesão/lesões a direitos que eventualmente possa sofrer ou vir a sofrer em razão de violação à Lei nº 13.709/2018 – Lei Geral de Proteção de Dados Pessoais – LGPD.

CLÁUSULA VII – DOS DIREITOS DO TITULAR

O TITULAR terá todos os direitos previstos e decorrentes deste termo de consentimento do uso de imagem, demais direitos previstos na Lei nº 13.709/2018 – Lei Geral de Proteção de Dados Pessoais – LGPD e ainda, os seguintes direitos:

I - Confirmação da existência de tratamento;
II - Acesso aos seus dados pessoais;
III - Correção de dados incompletos, inconsistentes, inexatos ou desatualizados;
IV - Anonimização, bloqueio ou eliminação de dados desnecessários, excessivos ou tratados em desconformidade com a LGPD ou outros documentos legais;
V - Eliminação dos dados pessoais tratados com o seu consentimento, exceto nos casos previstos em lei;
VI - Portabilidade dos dados a outros fornecedores de serviços ou produto(s), mediante requisição expressa, de acordo com a regulamentação dada pela ANPD – Autoridade Nacional de Dados, observados os segredos industriais e comercial;
VII – Eliminação dos dados pessoais tratados mencionados neste termo de consentimento do uso de imagem, com exceção dos casos previstos em lei;
VIII – Revogação do consentimento dado, a qualquer momento;
IX – Informação das entidades públicas e privadas com as quais o controlador realizou uso compartilhado de dados; e
X – Informação sobre a possibilidade de não fornecer consentimento e sobre as consequências da negativa.

PARÁGRAFO 1º. O exercício, pelo TITULAR, de seus direitos relacionados à proteção de dados pessoais será realizado mediante solicitação expressa por ele dirigida ao encarregado da proteção dos dados pessoais mencionado neste termo de consentimento de uso de imagem.

PARÁGRAFO 2º. O disposto nesta Cláusula não exclui a fruição, pelo TITULAR, em relação aos seus dados pessoais, de outros direitos que venham a ser conferidos por lei.

CLÁUSULA VIII – DO TÉRMINO DO TRATAMENTO DOS DADOS PESSOAIS

O tratamento dos dados pessoais objeto deste termo de consentimento de uso de imagem terá seu término/finalização/encerramento, nos seguintes casos:

I – Pelo exaurimento de suas finalidades;

II – No caso que haja revogação por parte do TITULAR de seu consentimento, nos termos estabelecidos neste instrumento particular, sem prejuízo da indenização ou ressarcimento de eventuais danos ou prejuízos que a revogação causar ao CONTROLADOR.

CLAÚSULA IX – DO ENCARREGADO DA PROTEÇÃO DOS DADOS PESSOAIS DO TITULAR

O ENCARREGADO da proteção dos dados pessoais do TITULAR nomeado pelo CONTROLADOR é:

(inserir a qualificação completa do ENCARREGADO)

PARÁGRAFO ÚNICO. As eventuais comunicações e solicitações dirigidas pelo TITULAR ao ENCARREGADO deverão ser encaminhadas presencialmente ou por *e-mail*, ficando a escolha a critério do TITULAR.

O consentimento ora manifestado pelo titular é conferido nos estritos termos mencionados neste instrumento particular e não poderá ser utilizado para quaisquer outras finalidades.

_____

(Local e data)

_____

Assinatura do TITULAR

ADENDO III

TERMO DE CONSENTIMENTO PARA USO DE IMAGEM

TRATAMENTO DE DADOS DE CRIANÇAS E ADOLESCENTES

O presente instrumento particular trata de um termo de consentimento para uso de imagem e tratamento de dados obtido através de reconhecimento facial, de menor impúbere ou menor relativamente incapaz devidamente representado por seus pais ou seu representante legal, que supervisionarão as condições para que _____ possa se submeter a todas as regras aplicáveis aos usuários e visitantes da(o) página/app … (informe os o nome da página ou do aplicativo/aplicáveis em caso de reconhecimento facial) podendo dispor sobre os dados pessoais e sensíveis conforme dispõe a Lei Geral de Proteção de Dados Pessoais – LGPD - Lei nº 13.709/2018 e as regras previstas no presente termo.

Para fins de esclarecimento o presente termo de consentimento de uso de imagem terá os seguintes envolvidos:

a) Controlador: É a pessoa natural ou jurídica que toma as decisões relacionadas ao tratamento dos dados pessoais coletados.
b) Titular dos dados: É aquele que fornece os seus dados para serem coletados.
c) Usuário: é a pessoa que acessa o(a) aplicativo/ página na 'web'/tem coletado sua imagem através de reconhecimento facial.
d) Responsável: Os pais ou responsável legal da criança e do adolescente, portanto, figurando como titular dos dados.

TITULAR DOS DADOS PESSOAIS

Nome completo da criança/adolescente, nacionalidade, estado civil, profissão, portador do RG nº _____ e do CPF nº _____, residente e domiciliado na Rua/Avenida/Praça (endereço completo), menor impúbere (criança – menor de 12 anos)/ assistido (adolescente – menor que 18 anos, salvo se tiver sido emancipado aos 16 anos) neste ato representado por seus pais/aquele que possui sua guarda/representante legal, nome completo _____, nacionalidade, estado civil, profissão, portador do RG nº _____ e do CPF nº _____, residente e domiciliado na Rua/Avenida/Praça (endereço completo), endereço eletrônico (e-mail).

CONTROLADOR

Nome completo, nacionalidade, estado civil, profissão, portador do RG nº _____ e do CPF nº _____, residente e domiciliado na Rua/Avenida/Praça (endereço completo), endereço eletrônico (e-mail).

CLÁUSULA I – DOS DADOS PESSOAIS

Os pais/o genitor que possui a guarda/o responsável legal do TITULAR, já devidamente qualificado, concorda com o tratamento de sua imagem, a ser captada pelo CONTROLADOR por meio de fotos/vídeos/outro tipo de tecnologia/reconhecimento facial.

Parágrafo único. O presente termo de consentimento para o uso de imagem abrange tão somente os dados pessoais explicitamente mencionados nesta cláusula.

CLÁUSULA II – DAS FINALIDADES PARA O USO DA IMAGEM

Os pais/o genitor que possui a guarda/o responsável legal do TITULAR concorda(m) com o tratamento de sua imagem para a(s) seguinte(s) finalidade(s):

(descrever a/as finalidade(s))

PARÁGRAFO 1º. O CONTROLADOR não poderá com base neste termo, utilizar dados pessoais descritos na Cláusula I para finalidades diversas das previstas nesta cláusula, sob pena de o tratamento ser considerado contrário aos ditames da LGPD.

PARÁGRAFO 2º. Os princípios a serem observados pelo CONTROLADOR no que diz respeito ao tratamento dos dados são os da boa-fé, finalidade, não discriminação, adequação, necessidade, livre acesso, qualidade dos dados, transparência, segurança, prevenção, e responsabilização, e prestação de contas.

PARÁGRAFO 3º. Ao concordar com o presente termo os pais/o genitor que possui a guarda/o responsável legal do TITULAR concorda(m) totalmente com o tratamento dos dados a seguir listados (o rol abaixo é exemplificativo, podendo ser acrescentados outros dados necessários à utilização do aplicativo, assim se houver necessidade de uso de outros dados, estes deverão ser especificados/incluídos no rol abaixo):

I – Nome completo do titular/pais/representantes legais.

II – Número do RG e do CPF.

III - Data de nascimento.

IV - Cidade/Estado.

V - Número de telefone.

VI - Idade.

VII – Estado civil.

VIII - Usuário e senha de acesso.

IX – Imagem (fotos-vídeo) – expressões.

X - Comunicação (verbal e escrita) relacionada aos contatos realizados entre o Titular e o controlador dos dados.

XI - Horário de acesso ao aplicativo/página na 'web'.

XII - Endereço eletrônico.

XIII - Sexo.

XIV - Coletamos ainda endereço de IP, 'id' do 'cookie' para identificar cada um dos dispositivos acessados.

XV - Todas as informações coletadas são aquelas que consideramos estritamente necessárias, não sendo coletada ou requerida qualquer informação que não seja relacionada ao interesse de uma participação protegida, segura e eficaz no e-game.

XVI - Os dados coletados são do usuário que tem o seu reconhecimento facial realizado e se houver necessidade os de seu responsável.

XVII – Os pais/o genitor que possui a guarda/responsável legal do TITULAR dão o seu consentimento por meio desse documento para que os dados pessoais e sensíveis coletados possam ser utilizados para a finalidade descrita, conforme a previsão do art. 11 da Lei Geral de Proteção de Dados.

PARÁGRAFO 4º. Sempre que pretender, com base no consentimento dos pais/genitor que possui a guarda/responsável legal do TITULAR, realizar atividades de tratamento de dados pessoais não abrangidas por esta cláusula, o CONTROLADOR deverá obter novo consentimento.

CLÁUSULA III – DO NÃO CONSENTIMENTO E DAS SUAS CONSEQUÊNCIAS

Os pais/o genitor que possui a guarda/representante legal do TITULAR declara para os devidos fins e direito, a quem possa interessar que, foi informado pelo CONTROLADOR de que poderia deixar de consentir com o tratamento de seus dados pessoais nos termos deste instrumento particular, também declara que recebeu tal informação antes que o presente termo fosse assinado, ficando ciente de que a negativa quanto ao seu consentimento ao tratamento dos dados lhe traria as seguintes consequências:

(descrever a/as consequência(s) – por se tratar de criança ou adolescente, não poderá haver o uso do reconhecimento facial).

CLÁUSULA IV – DO COMPARTILHAMENTO NACIONAL

Os dados pessoais objeto do presente termo de consentimento do uso de imagem serão compartilhados com as pessoas abaixo elencadas:

(descrever e qualificar de forma completa as pessoas físicas ou jurídicas com as quais os dados serão compartilhados)

PARÁGRAFO ÚNICO. O eventual compartilhamento de dados pessoais com pessoas não mencionadas nesta cláusula somente será possível, desde que os pais/genitor que tiver a guarda/responsável legal do TITULAR seja previamente informado e conceda seu consentimento de forma expressa.

CLÁUSULA V – DA SEGURANÇA DOS DADOS PESSOAIS

O CONTROLADOR adotará medidas de segurança, técnicas e administrativas aptas a proteger os dados pessoais dos pais/do genitor que possui a guarda/do responsável legal e do TITULAR de acessos não autorizados e de situações acidentais ou ilícitas de destruição, alteração, desvio, perda, comunicação ou qualquer outra forma de tratamento inadequado ou ilícito.

PARÁGRAFO 1º. O CONTROLADOR somente poderá compartilhar os dados pessoais dos pais/do genitor que possui a guarda/do responsável legal e do TITULAR, nos termos deste instrumento particular, se aquele para quem transferir assumir expressamente a obrigação de garantir a segurança dos referidos dados pessoais.

PARÁGRAFO 2º. O CONTROLADOR comunicará ao pais/genitor que tiver a guarda/responsável legal do TITULAR, no prazo definido pela ANPD – Autoridade Nacional de Dados Pessoais – link https://www.gov.br/anpd/pt-br/canais_atendimento/agente-de-tratamento/comunicado-de-incidente-de-seguranca-cis , a eventual ocorrência de incidente de segurança que possa lhe acarretar risco ou dano relevante, devendo dar aos pais/o genitor que possui a guarda/o responsável legal do TITULAR, todo apoio e informação que necessitar.

CLÁUSULA VI – DA RESPONSABILIDADE DO CONTROLADOR E DEMAIS AGENTES DE TRATAMENTO DE DADOS PESSOAIS

O CONTROLADOR, de *per si* ou solidariamente com outros agentes de tratamento de dados pessoais, responderá(ão) perante o TITULAR, por qualquer/quaisquer dano(s) ou lesão/lesões a direitos que eventualmente possa sofrer ou vir a sofrer em razão de violação à Lei nº 13.709/2018 – Lei Geral de Proteção de Dados Pessoais – LGPD.

CLÁUSULA VII – DOS DIREITOS DO TITULAR

Os pais/o genitor que possui a guarda/o responsável legal do TITULAR e O TITULAR terão todos os direitos previstos e decorrentes deste termo de consentimento do uso de imagem, demais direitos previstos na Lei nº 13.709/2018 – Lei Geral de Proteção de Dados Pessoais – LGPD e ainda, os seguintes direitos:

I - Confirmação da existência de tratamento;
II - Acesso aos seus dados pessoais;
III - Correção de dados incompletos, inconsistentes, inexatos ou desatualizados;
IV - Anonimização, bloqueio ou eliminação de dados desnecessários, excessivos ou tratados em desconformidade com a LGPD ou outros documentos legais;
V - Eliminação dos dados pessoais tratados com o seu consentimento, exceto nos casos previstos em lei;
VI - Portabilidade dos dados a outros fornecedores de serviços ou produto(s), mediante requisição expressa, de acordo com a regulamentação dada pela ANPD – Autoridade Nacional de Dados, observados os segredos industriais e comercial;
VII – Eliminação dos dados pessoais tratados mencionados neste termo de consentimento do uso de imagem, com exceção dos casos previstos em lei;
VIII – Revogação do consentimento dado, a qualquer momento;
Conforme a previsão do art.7 º, §5º da Lei Geral de Proteção de Dados a qualquer momento o titular e/ou responsável do titular, poderá revogar o consentimento para o tratamento dos dados coletados por meio do formulário preenchido.
A revogação do consentimento poderá ser solicitada pelos pais/genitor que possui a guarda/responsável legal do TITULAR, via e-mail a qualquer momento, para tanto endereço eletrônico para que possa ser realizada a solicitação da revogação do consentimento é _____ (preencha informando o endereço eletrônico para solicitação da revogação do consentimento).
IX – Informação das entidades públicas e privadas com as quais o controlador realizou uso compartilhado de dados; e
X – Informação sobre a possibilidade de não fornecer consentimento e sobre as consequências da negativa.

PARÁGRAFO 1º. O exercício, pelos pais/pelo genitor que possui a guarda/pelo responsável legal do TITULAR, de seus direitos relacionados à proteção de dados pessoais será realizado mediante solicitação expressa dos seus pais/do genitor que possui a guarda/do responsável legal do TITULAR dirigida ao encarregado da proteção dos dados pessoais mencionado neste termo de consentimento de uso de imagem.

PARÁGRAFO 2º. O disposto nesta Cláusula não exclui a fruição, pelos pais/o genitor que possui a guarda/o responsável legal do TITULAR e do TITULAR, em relação aos seus respectivos dados pessoais e, de outros direitos que venham a ser conferidos por lei.

CLÁUSULA VIII – DO TÉRMINO DO TRATAMENTO DOS DADOS PESSOAIS

O tratamento dos dados pessoais objeto deste termo de consentimento de uso de imagem terá seu término/finalização/encerramento, nos seguintes casos:

I – Pelo exaurimento de suas finalidades;

II – No caso que haja revogação por parte dos pais/genitor que tiver a guarda/responsável legal do TITULAR de seu consentimento, nos termos estabelecidos neste instrumento particular, sem prejuízo da indenização ou ressarcimento de eventuais danos ou prejuízos que a revogação causar ao CONTROLADOR.

PARÁGRAFO ÚNICO. Finalizando o período de tratamento dos dados dos pais/o genitor que possui a guarda/o responsável legal do TITULAR e os do TITULAR, ainda poderão estes ser utilizados para situações previstas na Lei n.º 13.709/2018, em seu art. 16 e incisos.

CLAÚSULA IX – DO ENCARREGADO DA PROTEÇÃO DOS DADOS PESSOAIS DO TITULAR

O ENCARREGADO da proteção dos dados pessoais dos pais/o genitor que possui a guarda/o responsável legal do TITULAR e do TITULAR nomeado pelo CONTROLADOR é:

(inserir a qualificação completa do ENCARREGADO)

PARÁGRAFO ÚNICO. As eventuais comunicações e solicitações dirigidas pelos pais/genitor que tiver a guarda/responsável legal do TITULAR ao ENCARREGADO deverão ser encaminhadas presencialmente ou por *e-mail*, ficando a escolha a critério dos pais/genitor que tiver a guarda/responsável legal do TITULAR.

O consentimento ora manifestado pelos pais/genitor que tiver a guarda/responsável legal do TITULAR é conferido nos estritos termos mencionados neste instrumento particular e não poderá ser utilizado para quaisquer outras finalidades.

_____

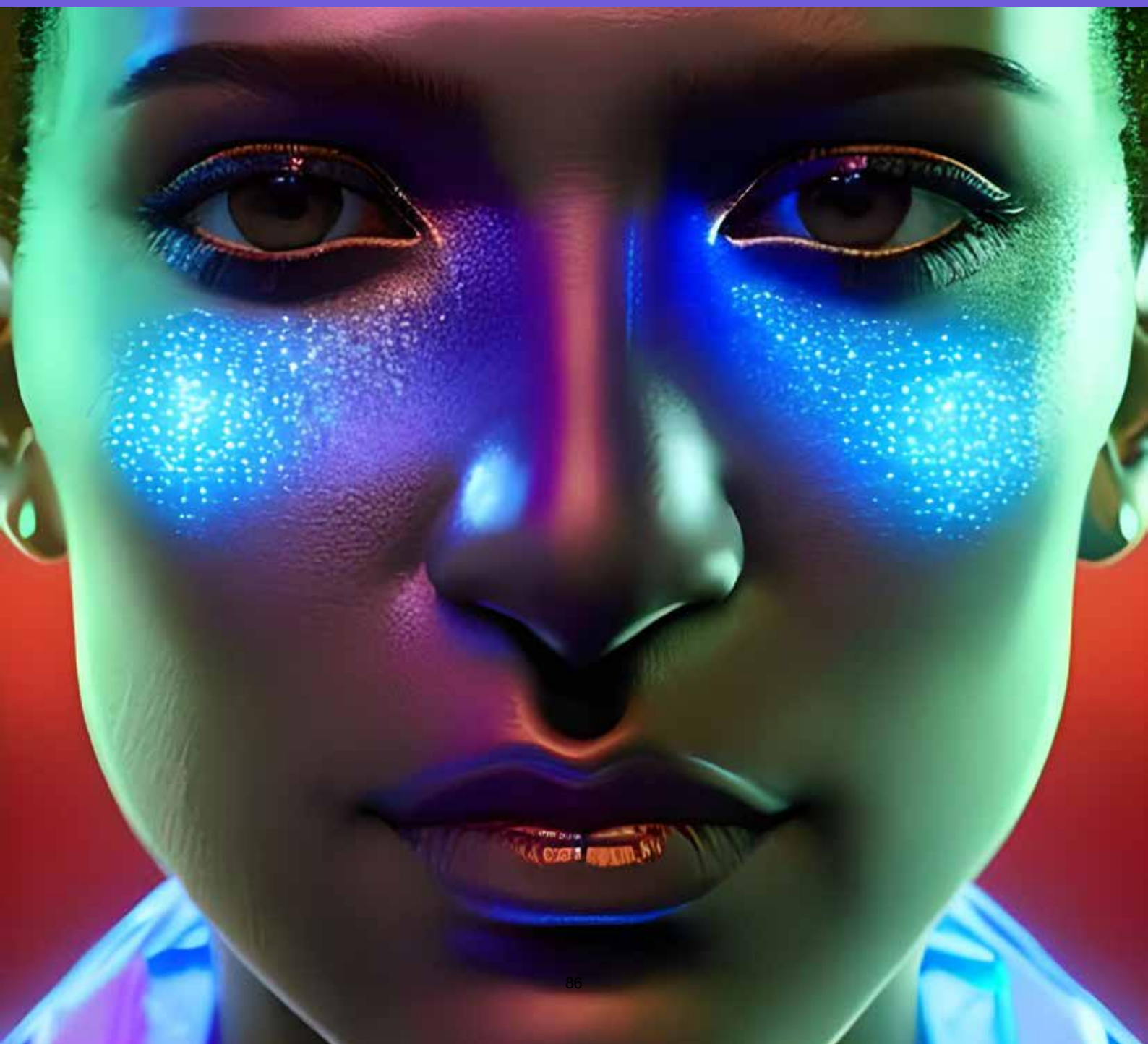
(Local e data)

_____

Assinatura dos pais/genitor que possui a guarda/responsável legal

Parte 2

A prática: Detalhamento dos Principais Algoritmos



6

EIGENFACES

Iniciamos esta parte prática de detalhamento dos principais algoritmos com o Eigenfaces. Trata-se de um método utilizado para fazer o reconhecimento facial de pessoas utilizando como base o método de Análise de Componentes Principais (PCA), que se utiliza da decomposição de autovalores (em inglês Eigenvalues, daí o nome do método Eigenfaces) de uma matriz de covariâncias de atributos da face.

Este método teve seu inicio com Sirovich e Kirby em 1987 e foi aperfeiçoado por Turk e Pentland em 1991 e ainda pode ser encontrada sua utilização em textos e projetos em combinação com outros métodos mais atuais, principalmente em se tratando de aplicações que envolvam o reconhecimento facial.

6.1 O método Eigenfaces

É um método utilizado para fazer o reconhecimento facial de pessoas. Sua abordagem começou em 1987 por Sirovich, L e Kirby, M que estavam na busca de encontrar uma representatividade de baixa dimensão para imagens faciais. Abaixo, temos alguns fatos históricos que aconteceram em relação aos métodos de detecção e reconhecimento facial, afim de obter um melhor entendimento em quais pontos os autores do método de eigenfaces procuraram evoluir e fazer diferente (TURK; PENTLAND, 1991).

Na década de 1960, Woodrow W Bledsoe desenvolveu um sistema que podia classificar fotos de rostos criadas a partir de um dispositivo chamado tablet HAND. O tablet HAND era um dispositivo de entrada de informações nos computadores da época, que permitia a inserção de coordenadas do eixo x e do eixo y em uma grade, conforme ilustrado na Figura 6.1.

Figura 6.1: Homem inserindo informações no sistema de reconhecimento facial com a utilização de um dispositivo tablet HAND.

Fonte: WIKIPEDIA, 2022

O sistema foi usado para registrar manualmente as localizações coordenadas das características faciais de um indivíduo, incluindo olhos, nariz, cabelo e boca. Por questões de poder de processamento na época, o reconhecimento facial era muito limitado, mas provou ser um método de reconhecimento biométrico bastante viável.

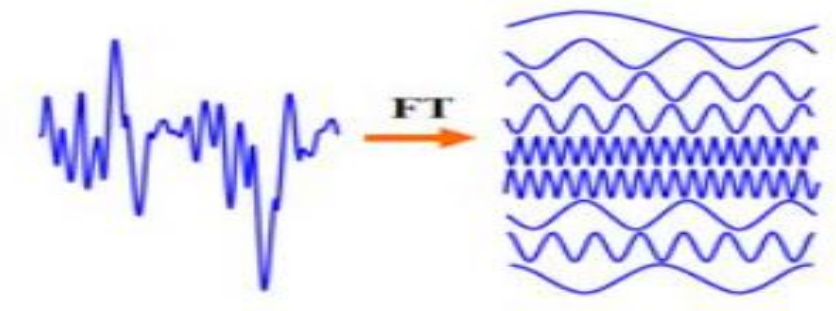
Na década de 1970, Goldstein, Harmon e Lesk conseguiram adicionar maior precisão a um sistema de reconhecimento facial manual. Eles usaram a técnica de adicionar 21 marcadores subjetivos específicos. Assim como no sistema de Bledsoe, porém, esses métodos ainda pesavam muito no poder de processamento da época.

Chegando assim na década de 1980, onde Sirovich, L e Kirby, M começaram a aplicar álgebra linear ao problema de reconhecimento facial. O que ficou conhecido como o método Eigenfaces. Começou como uma busca por uma representação de imagens faciais em baixa dimensão. Sirovich, L e Kirby, M conseguiram mostrar que, a análise de características feita em uma coleção de imagens faciais pode formar um conjunto de características básicas e um padrão. Eles também foram capazes de mostrar que menos de cem valores foram necessários para codificar com precisão a imagem normalizada de um rosto (TURK; PENTLAND, 1991).

O método Eigenface, consiste em fazer a extração de toda informação que seja relevante da imagem facial a ser analisada. A codificação dessa informação, o mais eficiente possível, e depois fazer a comparação da face codificada com toda uma base de dados composta por faces codificadas de forma semelhante. Em palavras mais simples é pegar uma imagem, fazer toda a leitura dela e comparar com uma outra imagem armazenada no banco de dados para fazer a comparação dos padrões de uma imagem a outra, fazendo assim a confirmação de igualdade. Na realidade, é uma das formas mais intuitivas de classificar uma face (ALMEIDA; BENTO, 2014).

Contrariamente às técnicas mais antigas, que se baseavam nas características particulares das faces, este método utiliza uma maior quantidade de informação, devido a classificação das faces com base em padrões faciais gerais. Comparada a transformada de Fourier (FT), utilizada muito frequentemente em aplicações relacionadas com processamento de sinais, esta operação matemática consiste na decomposição de uma função (sinal adquirido) em várias funções oscilatórias com parâmetros bem conhecidos (por exemplo, amplitude e frequência) (ALMEIDA; BENTO, 2014). A Figura 6.2 ilustra esse processo.

Figura 6.2: Comparação da representação de um sinal elétrico com a de um sinal na transformada de Fourier (FT).



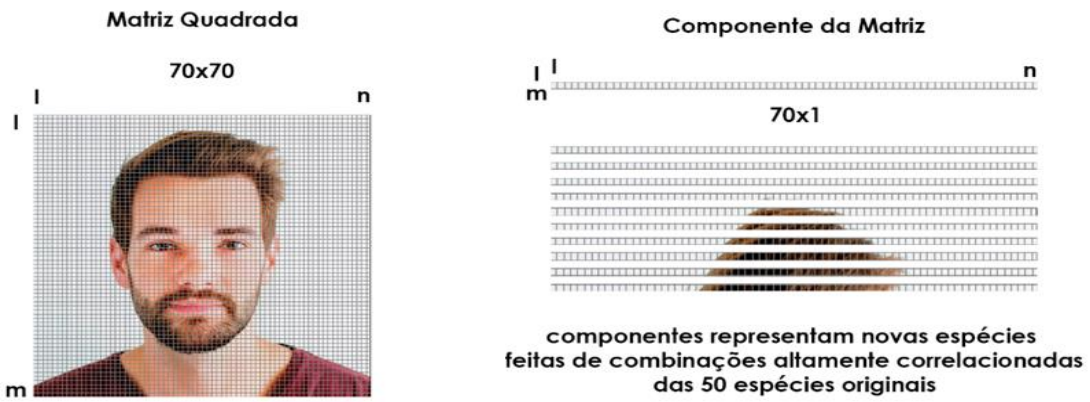
Fonte: ALMEIDA; BENTO, 2014

Este método consiste em efetuar uma operação muito parecida com à FT. Cada face recebia é decomposta em vários tipos de componentes importantes e transformada em vetores próprios da matriz principal, definindo por um conjunto de faces de referência. Tem o objetivo de representar de forma eficiente as imagens da face através de uma análise de componentes principais (PCA – *Principal Component Analysis*).

6.2 Método de Análise de Componentes Principais (PCA)

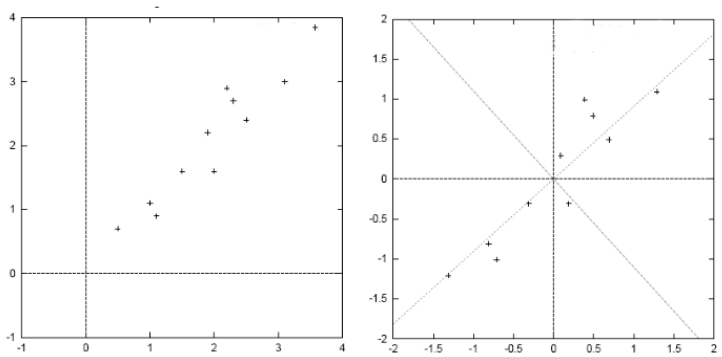
A Análise de Componentes Principais (Principal Component Analysis - PCA), é um procedimento matemático que utiliza uma transformação ortogonal (ortogonalização de vetores), ou seja, é o método que utilizamos quando pegamos uma matriz quadrada e verificamos se sua matriz inversa é parecida com a sua matriz transposta para converter um conjunto de observações de variáveis possivelmente correlacionadas num conjunto de valores de variáveis linearmente não correlacionadas chamadas de componentes principais (PESSOA, 2019).

Figura 6.3: Exemplo de decomposição de uma matriz quadrada em vários autovetores. Onde I seria o pixel inicial, m o pixel final de altura e o n pixel final de comprimento.



O PCA pode ser feito decompondo em autovalores (Eigenvalues) de uma matriz de covariâncias, geralmente após centralizar a matriz de dados para cada atributo. Os resultados do PCA são frequentemente discutidos em termos de pontuações de componentes, também chamadas de pontuações de fatores (os valores das variáveis transformadas correspondem a um determinado ponto de dados)

Figura 6.4: Comparação de um conjunto de dados com dimensão alta(esquerda) e um conjunto de dados com dimensão baixa(direita), verificando a similaridade entre ambos.



Fonte: BERNADINA, 2007

Resumidamente, o método PCA consiste em pegar um dado ou uma matriz de grande dimensão e reduzir em pequenas partes, procurando preservar o máximo possível seus valores para fazer comparação entre de similaridade entre elas.

O método Eigenface foi criado pensando nas características que a face de uma pessoa possa ter. Pois, fazer o reconhecimento de um rosto é semelhante a qualquer outro sistema biométrico onde cada indivíduo tem suas características próprias, como por exemplo a impressão digital que também tem suas estruturas e características únicas onde determina as características de um único indivíduo (PESSOA, 2019).

Esses tipos de problemas são muito desafiadores para os sistemas que fazem a detecção e o reconhecimento facial, pois sistemas assim precisam ser confiáveis e funcionar com precisão onde ao receberem uma nova imagem, fazem acesso a um banco de dados que armazena a imagem de várias pessoas e no meio de tantas imagens, precisa identificar se o rosto recebido já existe no banco de dados ou não.

Entretanto, no banco de dados também temos fatores que prejudicam um pouco para o lado do sistema, pois nem toda foto é igual a outra; como por exemplo no aspecto de iluminação, envelhecimento de uma pessoa para outra, fazendo com que o sistema tenha mais alguns problemas a resolver além de apenas fazer a detecção ou identificação.

Na Tabela 6.1, pode-se observar informações de como se dispõem a precisão de alguns algoritmos de detecção e reconhecimento facial criados no período de 1980 a 2000:

Tabela 6.1: Identifica o grau de precisão dos tipos de método de reconhecimento facial existente entre 1980 a 2000.

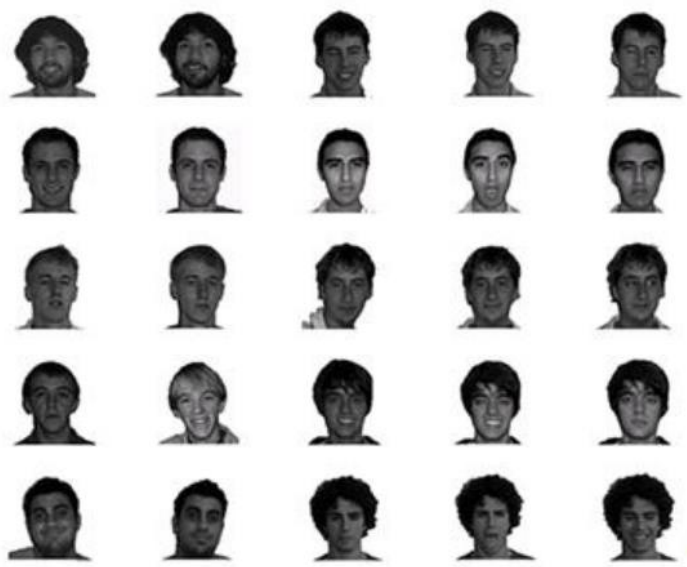
Method	Number of images in the training set	Success rate	Reference
Principal Component Analysis	400	79.65%	[1]
Principal Component Analysis + Relevant Component Analysis	400	92.34%	[1]
Independent Component Analysis	170	tanh function 69.40%	[2]
	40	Gauss function 81.35%	[2]
Hidden Markov Model	200	84%	[3]
Active Shape Model	100	78.12-92.05%	[4], [5]
Wavelet Transform	100	80-91%	[6]
Support Vector Machines	-	85-92.1%	[7], [8]
Neural Networks	-	93.7%	[9]
Eigenfaces Method	70	92-100%	[10]

Fonte: ÇARIKÇI; ÖZEN, 2012

Como pode ser observado na Tabela 6.1, o método de Eigenfaces fica entre 92% e 100% de precisão. A estratégia do método Eigenfaces consiste em extrair os traços característicos da face e representar a face em questão com uma combinação linear das chamadas 'eigenfaces' obtidas a partir de características retiradas no processo de extração. Abaixo, veremos uma sequência de como é feito esse processo.

A primeira coisa a ser feita, é obter a leitura do conjunto de treinamento de imagens NxN.

Figura 6.5: Banco de imagens para Treino armazenadas em um banco de dados.



Fonte: ALMEIDA; BENTO, 2014

Em seguida, é feito o cálculo da matriz de covariância para a obtenção da face média ou também conhecida como face de baixa dimensão que é extraída a partir do modelo de treinamento armazenado no banco de dados.

A matriz de covariância é uma matriz simétrica NxN (matriz quadrada de tamanho N, onde N é o número de variáveis dentro da matriz) que tem suas entradas de covariações associadas com todos os pares de valores iniciais e.g. [Pessoa 2019].

Por exemplo, um conjunto de dados tridimensional (com três variáveis, por exemplo: x, y e z) possui a matriz de covariância no formato indicado na Figura 6.6.

Figura 6.6: Covariância de uma matriz tridimensional.

$$\begin{bmatrix} Cov(x,x) & Cov(y,x) & Cov(z,x) \\ Cov(x,y) & Cov(y,y) & Cov(z,y) \\ Cov(x,z) & Cov(y,z) & Cov(z,z) \end{bmatrix}$$

Fonte: PESSOA, 2019

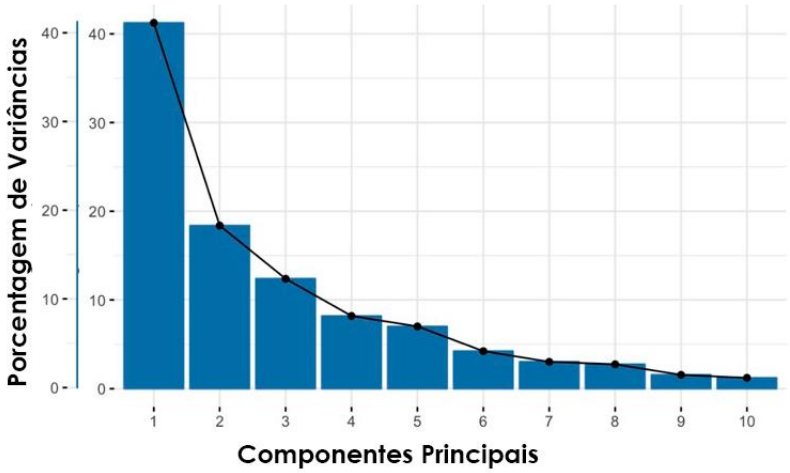
A diagonal principal são as variâncias de todas as variáveis do sistema. Se positivo, ambas as variáveis aumentam ou diminuem juntas (correlacionadas). Se negativa, uma das variáveis aumenta enquanto a outra diminui (inversamente correlacionada). O próximo passo é realizar o cálculo para a identificação de autovalores e autovetores.

Principais componentes são as novas variáveis que serão construídas através das combinações lineares das variáveis iniciais. Estas combinações são feitas de forma que as novas variáveis (principais componentes) não estejam correlacionadas entre si e a

maior parte da informação inicial (variáveis iniciais) foram compactadas nos primeiros componentes (PESSOA, 2019).

Por exemplo, se tivermos um conjunto de dados de dimensão 10, os mesmos fornecem 10 componentes principais. No entanto, o PCA tenta mover o máximo de informações para o primeiro componente, depois tenta mover o máximo das informações restantes para o segundo componente e assim por diante. No final, o dado terá uma aparência como o gráfico apresentado na Figura 6.7.

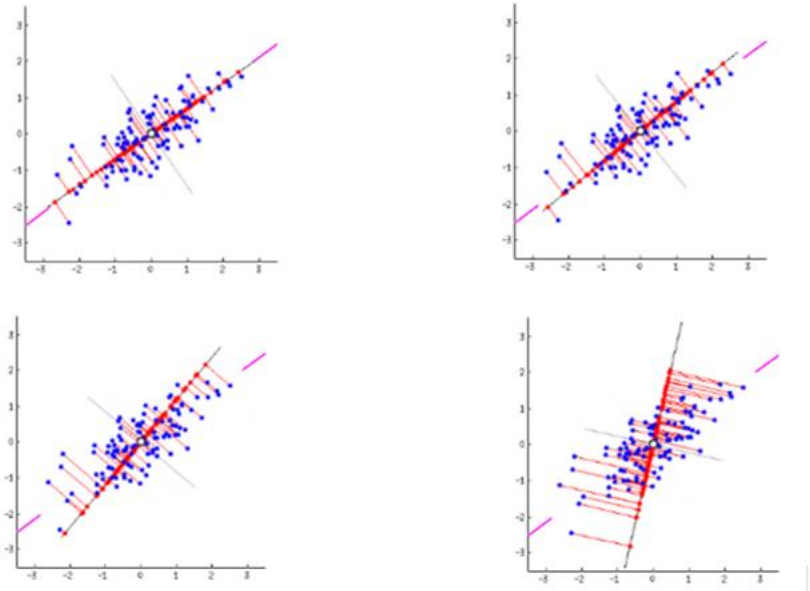
Figura 6.7: Ilustrando os componentes principais da imagem e a porcentagem de variações.



Fonte: PESSOA, 2019

Os componentes principais são menos interpretáveis do que as informações originais e não têm significado real. Os componentes principais tratarão os dados de forma generalizada, retornando à "direção dos dados". Essa direção explica a quantidade máxima de variância, ou seja, as linhas (vetores) que capturam mais informações dos dados.

Figura 6.8: O momento em que os pontos vermelhos estão mais espaçados é na linha onde o principal componente reside.



Fonte: PESSOA, 2019

Ao analisarmos a Figura 6.8, percebemos que o primeiro componente principal (localizado nas retas rosas) não perdeu tanta informação (variância dos pontos em sua reta). Os autovalores da matriz de covariância são as direções dos eixos onde possuem a maior variância (maior quantidade de informações) e que chamamos de componentes principais.

Assim, ao classificarmos os autovetores e autovalores da matriz de covariância, do maior para o menor autovetor, obtemos os componentes principais em ordem de importância.

Nota: quando dizemos "maior autovetor" e "menor autovetor" estamos nos referindo, respectivamente, ao "autovetor com maior autovalor" e "autovetor com menor autovalor". Suponhamos que nosso conjunto de dados são bidimensionais, variáveis *x* e *y*, e que seus autovetores e autovalores da matriz de covariância são os seguintes:

Autovetor 1 = v1;

Autovetor 2 = v2;

$$v1 = \begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix} \qquad \lambda_1 = 1.284028$$

$$v2 = \begin{bmatrix} -0.7351785 \\ 0.6778736 \end{bmatrix} \qquad \lambda_2 = 0.04908323$$

Classificado os autovetores do maior para o menor, temos que λ1>λ2. O que significa que o autovetor *v1* é o nosso componente principal (CP1) e que o vetor *v2* é o segundo componente principal (CP2).

Para determinarmos o quanto a variância (informação) que cada componente principal possui, calculamos a porcentagem dividindo o autovalor relacionado a esse componente principal (autovetor) pela soma de todos os autovalores.

CP1 -> 96% *

CP2 -> 4% *

* (CP = Componente Principal).

Podemos reduzir uma dimensão do nosso dado descartando o componente principal de menor importância (autovetor com menor autovalor da matriz de covariância).

Para o exemplo anterior, nossa matriz bidimensional:

$$\begin{bmatrix} 0.6778736 & -0.7351785 \\ 0.7351785 & 0.6778736 \end{bmatrix}$$

Teria o segundo (e menos importante) componente descartado:

$$\begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix}$$ Reduzindo, assim, em uma dimensão.

Quanto de informação nós perdemos? 4%.

Quanto de informação foi mantida? 96%.

Conseguimos reduzir a grande quantidade de informação em uma dimensão menor mantendo 96% de informação. O que precisa fazer agora é transformar estes 96% para o universo dos dados originais, com seus respectivos eixos, os quais poderiam ser interpretados.

Por fim, precisa multiplicar a transposta da matriz com os autovetores da matriz de covariância pelo conjunto de dados obteve a padronização.

Figura 6.10: Representação do cálculo final de padronização para obtenção da dimensão reduzida da imagem original.

$$dataset_{final} = \left(autovetores_{\text{cov}}\right)^{T} \cdot dataset_{padronizado}$$

Fonte: PESSOA, 2019

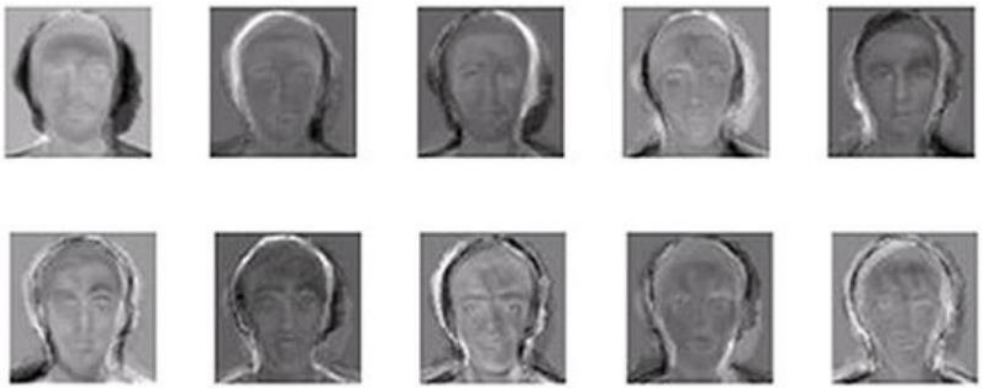
Após efetuar o cálculo de padronização dos dados, obtemos um conjunto de dados que podemos interpretar e assim fazer a comparação com sua versão original. Na Figura 6.11, temos o resultado final de uma das faces do modelo de treinamento após o processo de extração e padronização dos principais componentes, chamamos de face média ou face de baixa dimensão.

Figura 6.11: Face média ou de baixa dimensão obtida do modelo de treinamento após o processamento.

Fonte: ALMEIDA; BENTO, 2014

Após obter a face média, a imagem é transformada em vários autovetores, onde cada imagem pode ser representada exatamente como uma combinação linear ou aproximada utilizando o melhor conjunto de Eigenfaces, que apresentam os maiores autovalores e uma maior variância no conjunto de faces. A Figura 6.12 apresenta um exemplo de Eigenfaces, onde melhor conjunto de Eigenfaces representa um subespaço dimensional de tamanho M, chamado de "espaço de faces", de todas as imagens possíveis.

Figura 6.12. Conjunto de Eigenfaces com maior variância no espaço das faces definido pelo conjunto de treino.

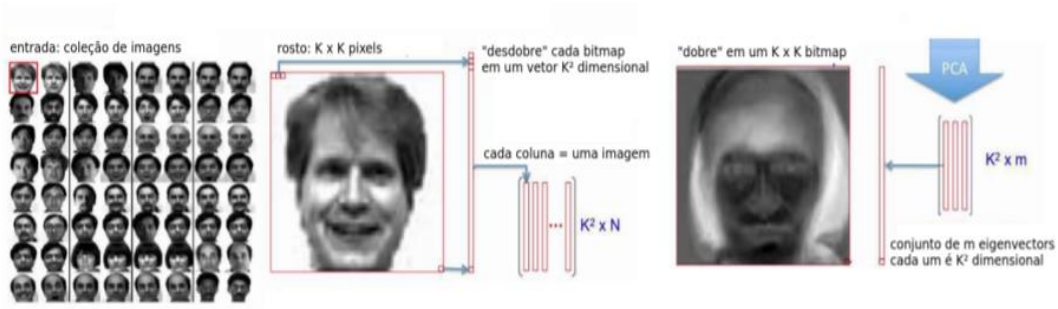


Fonte: ALMEIDA; BENTO, 2014

Cada Eigenface é simplesmente um vetor plano K^2 (k elevado à segunda potência) que pode ser transformado em uma nova imagem K x K.

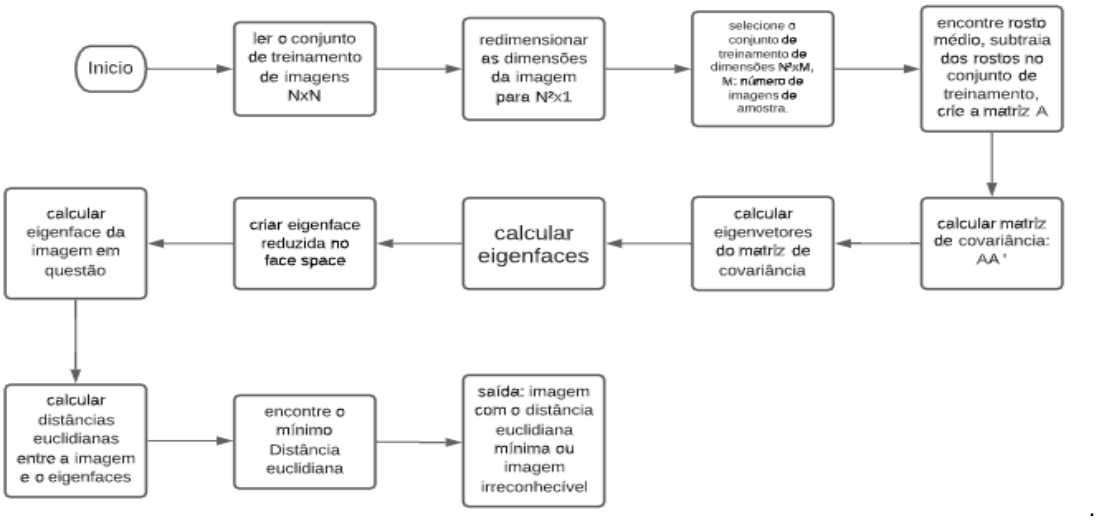
Para identificar a face, é necessário projetar a imagem no subespaço Eigenfaces. Usando um classificador como kNN (*k-Nearest Neighbors* ou os K vizinhos mais próximos) com a distância euclidiana para encontrar o(s) *k* vizinho(s) mais próximo(s).

Figura 6.13: Demonstração resumida do processo de análise de imagem do método de Eigenface.



Fonte: FUJIKAWA, 2016

Figura 6.14: Processo feito por um sistema de detecção e reconhecimento facial utilizando o método de eigenfaces.



Fonte: ÇARIKÇI; ÖZEN, 2012

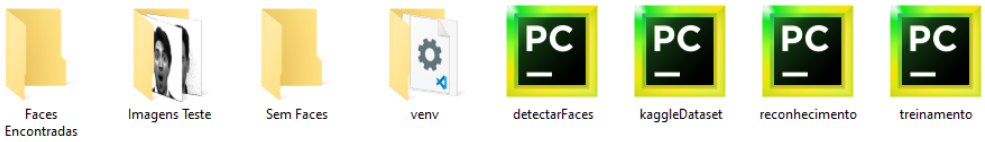
Temos na Figura 6.14, um fluxograma explicando de forma detalhada o processo feito por um sistema de detecção e reconhecimento facial utilizando o método de Eigenface.

O Eigenfaces considera o fato de que nem todas as partes de um rosto são igualmente importantes ou úteis para o reconhecimento facial. Na verdade, quando você olha para alguém, você reconhece essa pessoa por suas características distintas, como os olhos, nariz, bochechas ou testa; e como eles variam em relação um ao outro. Nesse sentido, você está se concentrando nas áreas de mudança máxima. Por exemplo, dos olhos para o nariz há uma mudança significativa, e o mesmo se aplica do nariz à boca (PESSOA, 2019).

6.3 Implementação do método de Eigenfaces com PCA no Python

Para fazer a implementação do método de Eigenfaces, antes de tudo, é preciso criar uma estrutura onde fique visível todo o processo a ser feito pelo algoritmo.

Figura 6.15: Estrutura de pastas e arquivos para o algoritmo Eigenfaces.



Foi criado o diretório *Faces Encontrada*, que receberá as imagens detectadas pelo arquivo *detectarFaces.py*, o diretório *Imagens Teste* receberá imagens aleatórias escolhidas a partir do dataset escolhido para fazer o reconhecimento facial. O diretório *Sem Faces* receberá as imagens que não foram detectada face humana. O diretório *venv* é o ambiente virtual do Python onde faremos toda as configurações de módulos para o funcionamento do algoritmo.

O script *detectarFaces.py* fica responsável por fazer a detecção da face nas imagens e fazer a separação enviando cada imagem para seu destino, se tiver face vai para *Faces Encontradas* e se não tiver vai para *Sem Faces*.

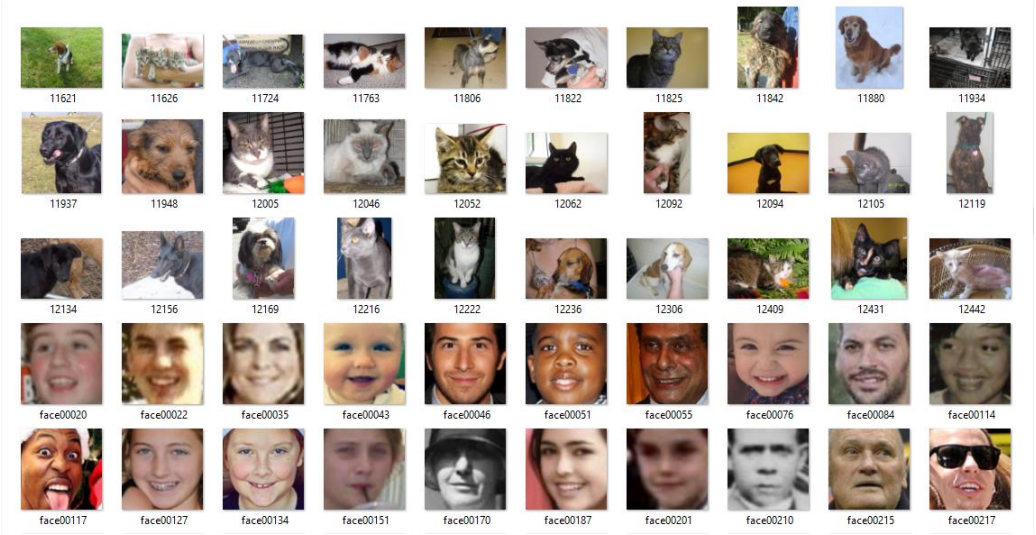
O script *kaggleDataset.py* é responsável por acessar a plataforma Kaggle.com e fazer o download do dataset que será utilizado para fazer o teste do algoritmo.

O script *reconhecimento.py* ficará responsável por fazer o reconhecimento das imagens no diretório *Imagens Teste* comparando com os resultados entregue pelo modelo de treinamento.

E por fim, mas não menos importante, o script *treinamento.py* que ficará a cargo de analisar todas as imagens do diretório *Faces Encontradas* e detectar os principais componentes da face salvando em um modelo de treinamento fara efetuar o reconhecimento facial.

Para fazer o teste do algoritmo, foi utilizado um dataset personalizado com 600 imagens positivas e 400 negativas, dentre elas, um misto feito entre três dataset mais famosos para detecção e reconhecimento facial, o Labeled Faces in the Wild (LFW), o Dogs vs Cats e o Yalefaces.

Figura 6.16: Base de dados para treinamento do algoritmo.

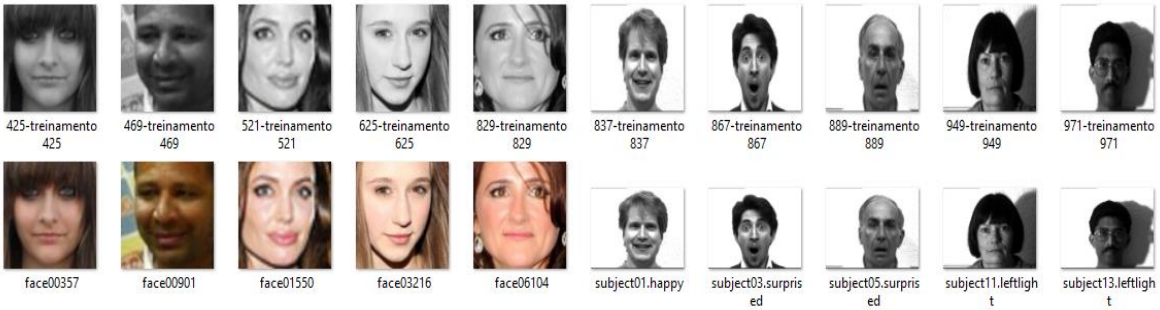


De forma simplificada, foram selecionadas de forma aleatória algumas dessas imagens para obter o reconhecimento facial conforme ilustra a Figura 6.17.

Figura 6.17: Imagens escolhidas antes do processamento.

A imagens apresentadas anteriormente, foram escolhidas antes de fazer o preparo. Para deixa-las ideal para o teste do algoritmo, após a seleção, foi feito o redimensionamento, a conversão para escala de cinza e renomeados os arquivos para o algoritmo receber todas as imagens no mesmo padrão.

Figura 6.18: Imagens escolhidas após processamento.



Após o processamento, a imagem recebe um nome para facilitar o treinamento e o reconhecimento, onde recebe o id que será seu registro.

6.4 Explicando o código

Primeiramente, para o teste do algoritmo, foi utilizado a IDE de desenvolvimento Visual Studio Code e a versão 3.10 do Python, devidamente instalada. Quanto a IDE, você poderá utilizar a que mais lhe convier.

Ao abrir a pasta do projeto dentro da IDE, é preciso acessar o terminal que fica no menu superior da IDE e digitar o `python -m venv venv` para criar o ambiente virtual do python no projeto. Será então criada uma pasta chamada *venv* dentro do diretório do projeto, essa pasta fica responsável por controlar todas as instalações de módulos que serão utilizados no projeto sem instalar fisicamente no seu computador, com isso consegue-se obter um ambiente mais seguro para realizar os teste sem danificar ou alterar algum arquivo importante dentro do seu computador.

6.4.1 kaggledataset.py

Código 6.1: Importação das bibliotecas.

```
import kaggle
from kaggle.api.kaggle_api_extended import KaggleApi
import os
import cv2
from PIL import Image
```

Para esse primeiro script, está sendo utilizado a API fornecida pelo kaggle para fazer o download do dataset escolhido para teste. Foi feito a importação da biblioteca kaggle que tem todos as configurações de comandos a serem utilizados na plataforma kaggle, kaggle.api.kaggle_api_extend para utilizar todos os comandos fornecidos pela api, OS para utilizar todos os comandos do sistema operacional como por exemplo acessar os diretórios, CV2 para fazer o processamento nas imagens e a PIL importando o módulo Image responsável por fazer manipulações em imagem.

Código 6.2: Configuração para autenticação na API.

```
api = KaggleApi()
api.authenticate()
```

Depois disso, foi criado uma variável chamada api que receberá o método kaggleApi para iniciar o processo de autenticação na plataforma kaggle, assim coloca-se o nome da variável seguido de ponto e chama a função authenticate.

Código 6.3: Criação do diretório Banco Imagens.

```
if not os.path.exists('Banco Imagens'):
    os.makedirs('Banco Imagens')
```

Em seguida, cria-se uma condição para que se o diretório chamado Banco Imagens não existir dentro do diretório do projeto, seja criado.

Código 6.3: Download do dataset.

```
kaggle.api.dataset_download_files(
      'asacxyz/ic-fatecitu',
      path='Banco Imagens',
      unzip=True
)
```

Agora, precisa chamar a função da api para fazer o download do dataset, passando como parâmetro o nome do proprietário seguido do dataset e informar em qual local você quer salvar pedindo para descompactar o arquivo .zip que irá receber da plataforma, nesse caso está pegando o dataset chamado ic-fatecitu do usuário asacxyz, pedindo para salvar no diretório Banco Imagens e logo após o download pedindo para descompactar o arquivo nesse mesmo diretório.

Código 6.4: Variáveis de controle.

```
cont = 1
naoTemFace = 0
```

Também é criado algumas variáveis de controle para verificar se o algoritmo está identificando arquivo por arquivo dentro do diretório e adicionar o id nas imagens.

Código 6.5: Acessando o diretório.

```
caminhos = [os.path.join('Banco Imagens', f) for f in os.listdir('Banco Imagens')]
```

Neste trecho de código, é criado uma variável que receberá o caminho de todas as imagens obtida pela função os.path.join e em seguida é criado um laço de repetição for para listar todos os arquivos dentro desse diretório utilizando a função os.listdir.

Código 6.6: Processando as imagens.

```
for caminhoImagem in caminhos:

    if ".jpeg" in caminhoImagem:

        nomeArquivo = os.path.splitext(caminhoImagem)[0]
        imagem = Image.open(caminhoImagem).convert('RGB')
```

```
                                    .save(nomeArquivo+'.'+'jpg')
        os.remove(caminhoImagem)

    if ".jpg" in caminhoImagem:

        imagem = cv2.imread(caminhoImagem)
        if imagem is None:
            os.remove(caminhoImagem)
            naoTemFace += 1

        img = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (600, 600))
        cv2.imwrite(caminhoImagem, img)
        os.rename(caminhoImagem, os.path.join('Banco Imagens', str(cont)
                                    +'-treinamento'+str(cont)+'.jpg'))
        cont += 1
```

Logo após, foi criado um laço de repetição for para percorre cada arquivo de imagem dentro do diretório *Banco Imagens*, também foi criado duas condições, a primeira se o arquivo que for feito download estiver com a extensão no final como .jpeg, pega-se o nome do arquivo e separa ele da extensão pela função os.path.splitext e armazenado na variável chamada *nomeArquivo*, após isso ser feito, cria-se uma nova variável chamada imagem que receberá a imagem com o primeiro nome e adicionará a extensão .jpg no arquivo e salvando ele dentro do diretório atual, após isso ser feito é preciso apagar a imagem anterior para não ficar com duplicidade dentro do diretório utilizando a função os.remove.

A segunda se o arquivo já estiver com a extensão .jpg no final, é criado uma nova condição para que se o arquivo apresentar problemas na leitura após ter sido feito a troca da extensão anteriormente que seja descartado do Banco de Imagens, se o arquivo estiver normal é feito a conversão dele para escala de cinza com o método cv2.cvtColor da biblioteca cv2, é feito o redimensionamento dele para 600x600 com o método cv2.resize é salvo dentro do diretório com o método cv2.imwrite e renomeado com a função os.rename onde estará sendo colocado o id no arquivo seguido de treinamento e a cópia do id para facilitar o reconhecimento futuro, e para finalizar o código é colocado uma variável de controle para verificar se está passando de arquivo em arquivo.

6.4.2 detectarFaces.py

Código 6.7: Import das bibliotecas.

```
import cv2
import os
import time
```

Para esse arquivo, será utilizado as bibliotecas cv2, os e time que será utilizado para obter o tempo de desempenho da análise do algoritmo.

Código 6.8: Variáveis de controle.

```
cont = 0
naoTemFace = 0
tempos = []
```

É criado as variáveis de controle para se caso achar uma face na imagem ou se não achar e criado um array que receberá os tempos de processamento de cada imagem.

Código 6.9: Pegando o tempo inicial.

```
tempInicial = time.time()
```

É criada uma variável chamada tempInicial, que receberá o valor inicial do tempo de execução do algoritmo.

Código 6.10: Acessando o diretório.

```
caminhos = [os.path.join('Banco Imagens', f) for f in os.listdir('Banco
Imagens')]
```

Depois é preciso acessar o diretório onde está contida as imagens para detecção criando uma variável chamada caminhos que irá receber o caminho da cada imagem obtido pela função os.path.join indicando qual o nome do diretório esta as imagens e sendo listada pela função os.listdir.

Código 6.11: Detectando as faces.

```
for caminhoImagem in caminhos:

    tempInicialImagem = time.time()

    imagemFace = cv2.cvtColor(cv2.imread(caminhoImagem),
                                                cv2.COLOR_BGR2GRAY)
    imagemFace = cv2.resize(imagemFace, (400, 400))
    detectorFace = cv2.CascadeClassifier('C:\\Users\\gog_e\\OneDrive
\\Ambiente de Trabalho\\Eigenfaces v2\\venv\\Lib\\site-packages\\cv2\\
data\\haarcascade_frontalface_alt.xml')
    detectorEyes = cv2.CascadeClassifier('C:\\Users\\gog_e\\OneDrive
\\Ambiente de Trabalho\\Eigenfaces v2\\venv\\Lib\\site-packages\\cv2\\
data\\haarcascade_eye.xml')
    detectorGlass = cv2.CascadeClassifier('C:\\Users\\gog_e\\OneDrive
\\Ambiente de Trabalho\\Eigenfaces v2\\venv\\Lib\\site-packages\\cv2\\
data\\haarcascade_eye_tree_eyeglasses.xml')
           facesDetectadas    =    detectorFace.detectMultiScale(imagemFace,
scaleFactor=1.1, minNeighbors=4)
```

Assim é criado um laço de repetição for, que percorrerá arquivo por arquivo dentro do diretório, é criado uma variável chamada *tempInicialImagem* que pegará o tempo de início da execução da detecção da imagem, após é criado uma variável chamada imagemFace que receberá a leitura da imagem pelo método imread e se caso acontecer de a imagem ainda estar colorida ser convertida para escala de cinza através do método cvtColor.

Após fazer a leitura e conversão se houver a necessidade é feito o redimensionamento da imagem para um grau um pouco menor do atual de 600x600 para facilitar ainda mais a detecção facial utilizando o método cv2.resize e em seguida criado três variáveis que funcionarão como filtro para diminuir ainda mais o percentual de possíveis erros, uma chamada detectorFace, outra chamada detectorEyes e outra chamada detectorGlass, como o Eigenfaces é um algoritmo focado em reconhecimento facial, nesse script estaremos utilizando o algoritmo Haar-Cascade (que será explicado em detalhes no próximo capítulo) para auxiliar na filtragem das imagens separando as imagens que tem face das que não tem face e para isso após a criação das três variáveis que será utilizado como filtro, é preciso chamar os modelos pré-processados que já tem junto da biblioteca cv2.

Para a variável detectorFace é configurado o arquivo haarcascade_frontalface_alt.xml que ficará responsável por passar as coordenadas se for detectado uma face no arquivo, para a variável detectorEyes será configurado o arquivo haarcascade_eye.xml para identificação dos olhos de uma pessoa e a variável detectorGlass será configurado para identificar possíveis faces que contenha óculos na face.

Após essa configuração, é criado uma variável chamada faces detectadas que receberá o valor entregue pelo método detectMultiScale que recebe a imagem como parâmetro, o scaleFactor que indica até qual tamanho a imagem pode ser redimensionada em escala, o minNeighbors que identifica quantos vizinhos cada retângulo da face deve ter.

Código 6.12: Criando a bouding Box na região dos olhos.

```
for (x, y, l, a) in facesDetectadas:
  cv2.rectangle(imagemFace, (x, y), (x + l, y + a), (0, 0, 255), 2)
  olhosDetectados = detectorEyes.detectMultiScale(imagemFace, scaleFactor=1.3, minNeighbors=4)
```

É criado um novo laço for dentro do anterior para pegar as coordenadas obtidas pelo reconhecedor de olhos e criar uma bouding box ao redor das coordenadas passadas para o eixo x, y, l, onde x é o inicial do x, o y o inicial do y e l tamanho da largura e a tamanho da altura da imagem.

Código 6.13: Condição para se caso não achar face na imagem.

```
if len(facesDetectadas) == 0:
        naoTemFace += 1
        os.rename(caminhoImagem, 'Sem Faces\\' + str(naoTemFace) +'-semFace' + '.jpg')
```

Após é criado uma condição para se caso o valor recebido pela variável facesDetectadas seja igual a zero adiciona mais um na variável de controle naoTemFace e transfere a imagem para o diretório Sem Faces utilizando a função os.rename para mudar o caminho atual para o caminho que direciona para o diretório *Sem Faces*.

Código 6.14: Condição para se caso achar uma face na imagem.

```
else:
        cont += 1
          os.rename(caminhoImagem, 'Faces Encontradas\\' + str(cont) + '-treinamento' + str(cont) + '.jpg')
```

Se caso achar a face, adiciona mais um na variável de controle, e renomeia o caminho da imagem para o diretório Faces Encontradas.

Código 6.15: Condição para se caso achar uma face na imagem.

```
tempFinalImagem = time.time()
tempos.append(tempFinalImagem - tempInicialImagem)
```

Após a condição, é criado a variável tempFinalImagem que receberá o valor final do tempo e adicionamos cada tempo ao array tempos para analisar depois qual foi o maior, qual foi o menor e a média que obtivemos de todos os tempos.

Código 6.16: Remover o diretório vazio.

```
os.rmdir('Banco Imagens')
tempFinal = time.time()
```

É preciso remover o diretório vazio do diretório de teste do algoritmo, e pegar o tempo final de todo o processo de reconhecimento facial passando para a variável tempFinal.

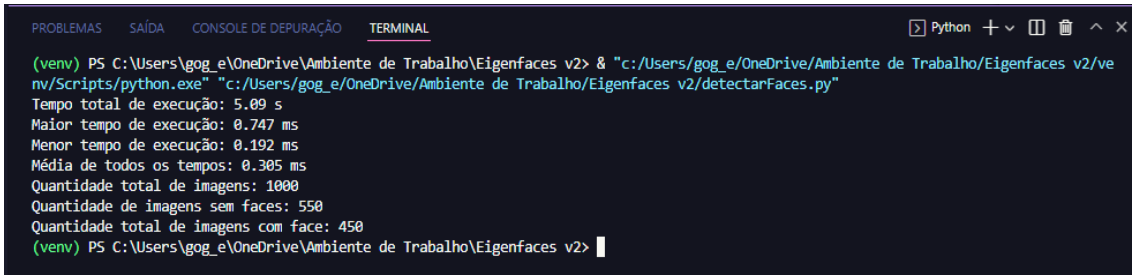
Código 6.17: Amostragem dos resultados final de detecção facial.

```
print('Tempo total de execução: {:.2f} s '.format((tempFinal - tempInicial) / 60))
print('Maior tempo de execução: {:.3f} ms '.format(max(tempos)))
print('Menor tempo de execução: {:.3f} ms'.format(min(tempos)))
print('Média de todos os tempos: {:.3f} ms'.format(sum(tempos) / len(tempos)))
print('Quantidade total de imagens: {}'.format(len(caminhos)))
print('Quantidade de imagens sem faces: {}'.format(naoTemFace))
print('Quantidade total de imagens com face: {}'.format(cont))
```

Por último, é feita a amostragem dos resultados de todos os tempos feito pelo desempenho do algoritmo, mostrando o tempo total de execução, o maior tempo, o menor tempo, a média de todos os tempos, a quantidade total de imagens analisadas, a quantidade total de imagens com faces e a quantidade total de imagens sem face.

Ao analisar as imagens contidas nos diretórios *Faces Detectadas* e no diretório *Sem Faces*, pode-se ver que algumas imagens não foram analisadas corretamente, no diretório *Faces Detectadas* houveram 16 imagens de cachorros e gatos que passaram como se tivessem face e no diretório *Sem Faces* passaram 166 imagens de face humana como não tendo faces, para fazer o cálculo de precisão, foi somado o total de imagens certas identificadas que ficou no valor de 818 imagens analisadas corretamente menos o total de imagens analisadas incorretamente que deu 182, divido pelo total de imagens entregue e multiplicado por 100 para achar a porcentagem, com isso chegamos no valor de 63,3% de precisão do algoritmo Haar-Cascade para detecção facial.

Figura 6.19. Amostragem dos resultados final de detecção facial.



6.4.3 treinamento.py

Código 6.18: Importação dos módulos.

```
import cv2
import os
import numpy as np
import time
```

Para esse arquivo, foi feito o import da biblioteca cv2, OS, numpy para utilizarmos cálculos com matrizes e time para verificação dos tempos.

Código 6.19: Criação do Classificador.

```
eigenface = cv2.face.EigenFaceRecognizer_create()
```

Depois é preciso criar uma variável chamada eigenface que irá receber os dados necessários para criar o classificador que irá treinar o modelo de treinamento a partir do método EigenFaceRecognizer_create.

Código 6.20: Função para pegar os Ids e Faces.

```
def getImagemComId():
    caminhos = [os.path.join('Faces Encontradas', f) for f in
os.listdir('Faces Encontradas')]
    faces = []
    ids = []
    tempos = []
    for caminhoImagem in caminhos:
        tempoInicial = time.time()
        imagemFace = cv2.cvtColor(cv2.imread(caminhoImagem),
cv2.COLOR_BGR2GRAY)
        imagemFace = cv2.resize(imagemFace, (400, 400))
        id = int(os.path.split(caminhoImagem)[-1].split('-')[0])
        ids.append(id)
        faces.append(imagemFace)
        tempoFinal = time.time()
        tempos.append(tempoFinal - tempoInicial)

    return np.array(ids), faces, tempos
```

Então, é preciso criar um método que ficará responsável por acessar cada imagem no diretório de Faces Encontradas e pegar cada id da imagem e adicionar na lista id, pegando também cada imagem e adicionando elas na lista faces, a variavel caminhos receberá os caminhos de cada imagem, foi criado três arrays para adicionar os ids, as faces e os tempos, foi criado também um laço de repetição for para percorrer

cada imagem dentro do diretório, criado a variavel de tempo inicial, convertido a imagem para escala de cinza e redimensionado de 600x600 para 400x400. Foi pego também o id que está no início do nome do arquivo e adicionado ele para a lista de ids e depois foi pego a imagem e feito a adição dela para a lista de faces, foi criado a variavel de tempo final e adicionado na lista de tempos. E por último pedimos para o método retornar os ids, faces e tempos em forma de matriz com a função np.array.

Código 6.21: Chamando a função.

```
ids, faces, tempos = getImagemComId()
```

Depois é feita a chamada do método passando as variáveis que irão armazenar os dados obtidos.

Código 6.22: Passando as listas para treinamento do modelo.

```
print('Treinando...')

tempTrainInicial = time.time()

eigenface.train(faces, ids)
eigenface.write('classificadorEigen.yml')
tempTrainFinal = time.time()
```

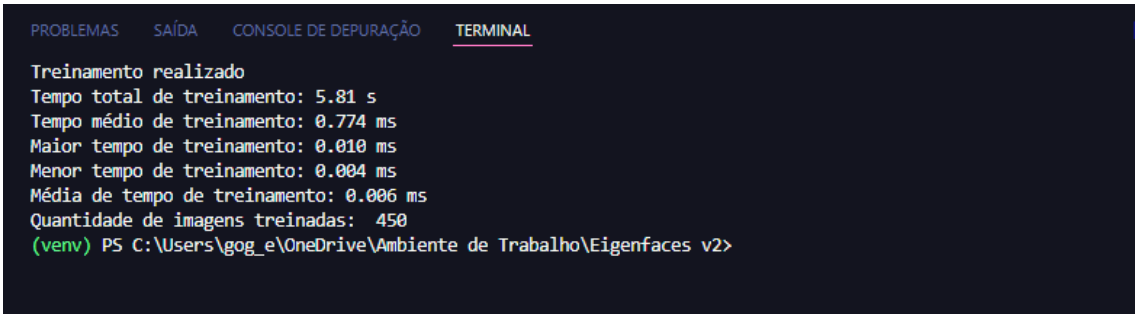
É feita a impressão na tela, informando que o treinamento foi iniciado, cria-se uma nova variavel de tempo inicial para pegar o tempo inicial do treinamento, é chamado o método .train e passado como parâmetros a lista de ids e a lista de faces, e depois utiliza-se o método .write para salvar esse arquivo na raiz do projeto do algoritmo chamado de classificadorEigen.yml. E por último cria-se a variável final do tempo de execução do treinamento.

Código 6.23: Mostrando os Resultados do treinamento.

```
print('Treinamento realizado')

print('Tempo total de treinamento: {:.2f} s'.format((tempTrainFinal -
tempTrainInicial)/60))
print('Tempo médio de treinamento: {:.3f} ms'.format((tempTrainFinal -
tempTrainInicial)/len(ids)))
print('Maior tempo de treinamento: {:.3f} ms'.format(max(tempos)))
print('Menor tempo de treinamento: {:.3f} ms'.format(min(tempos)))
print('Média de tempo de treinamento: {:.3f} ms'.format(np.mean(tempos)))
print('Quantidade de imagens treinadas: ', len(ids))
```

Por fim, é preciso pedir para exibir os resultados finais dos tempos e a quantidade de imagens treinada pelo modelo.

Figura 6.20: Mostrando os Resultados do treinamento.



Resultado final do treinamento das imagens e tempo de desempenho.

6.3.4 reconhecimento.py

Código 6.24: Import das bibliotecas.

```
import cv2
import os
import numpy as np
import time
```

Foi feito o import das bibliotecas, cv2, os, numpy e time para a verificação dos tempos.

Código 6.25: Variáveis de controle.

```
cont = 0
idReconhecido = 0
tempos = []
```

Criado duas variáveis de controle e um array para colocar todos os tempos dentro dele.

Código 6.26: Criação do reconhecedor de faces.

```
reconhecedor = cv2.face.EigenFaceRecognizer_create()
reconhecedor.read('classificadorEigen.yml')
```

Foi criado uma variável para receber o valor do reconhecedor EigenfaceRecognizer_create, pedindo para ler o arquivo gerado pelo arquivo treinamento.py.

Código 6.27: Acessando o diretório.

```
caminhos = [os.path.join('Imagens Teste', f) for f in os.listdir('Imagens Teste')]
```

Criado a variável para acessar o diretório onde contém as imagens para realização dos testes.

Código 6.28: Variáveis de controle de tempo.

```
tempInicial = time.time()
```

Criado a variável de tempo inicial para receber o tempo de execução total do reconhecimento facial.

Código 6.29: Criando a bouding box na face.

```
for caminhoImagem in caminhos:
    tempInicialImagem = time.time()
    imagemFace = cv2.cvtColor(cv2.imread(caminhoImagem), cv2.COLOR_BGR2GRAY)
    imagemFace = cv2.resize(imagemFace, (400, 400))
    faceDetect = cv2.CascadeClassifier('C:\\Users\\gog_e\\OneDrive
\\Ambiente de Trabalho\\Eigenfaces v2\\venv\\Lib\\site-packages\\cv2
\\data\\haarcascade_frontalface_default.xml')
    faces = faceDetect.detectMultiScale(imagemFace,
scaleFactor=1.1,  minNeighbors=4, minSize=(200, 200))
```

É criado um laço de repetição for para percorrer arquivo por arquivo dentro do diretório Imagens Teste, convertendo cada imagem para escala de cinza e redimensionando para 400x400, e também foi utilizado o método de Haar-Cascade para identificar se há uma face na imagem.

Código 6.30: Criando a predição das imagens a partir das características principais.

```
for (x, y, l, a) in faces:
        imagemFace = cv2.rectangle(imagemFace, (x, y), (x + l, y + a), (0, 0,
255), 2)
        id, confianca = reconhecedor.predict(imagemFace)
        print('ID da imagem que contém as mesmas caracteristicas no diretorio
de faces reconhecidas: {}'.format(id))
        print('Confiança: {}'.format(confianca))
        idReconhecido += 1
        cv2.imshow('Face de Teste', imagemFace)
        cv2.waitKey(0)
```

Também foi criado um outro laço de repetição dentro do anterior, para fazer a criação da bouding box ao redor das coordenadas da face e chamar o método predict para mostrar a predição da imagem recebida e mostrando ela logo em seguida com a função imshow do cv2.

Código 6.31: Variáveis final de tempo.

```
tempFinalImagem = time.time()
tempos.append(tempFinalImagem - tempInicialImagem)
```

Foi criado a variável final de tempo para pegar o tempo final da execução da predição e adicionada na lista de tempos.

Código 6.32: Mostrando a imagem que for semelhante ao id entregue na predição.

```
caminhos = [os.path.join('Faces Encontradas', f) for f in
os.listdir('Faces Encontradas')]

    for caminhoImagem in caminhos:
        if int(os.path.split(caminhoImagem)[-1].split('-')[0]) == id:
            imagemFace = cv2.cvtColor(cv2.imread(caminhoImagem),
cv2.COLOR_BGR2GRAY)
            imagemFace = cv2.resize(imagemFace, (400, 400))
            cont += 1
            cv2.imshow('Face do Banco Imagens', imagemFace)
            cv2.waitKey(0)
```

Após esse processo, é criado uma variavel para acessar o diretório de Faces Encontradas, criado um laço de repetição for para percorrer todos os arquivos dentro do diretório e criado uma condição para que todo arquivo que ele que tiver o mesmo Id entregue pelo predict mostrar ela na tela pegando direto do diretório de Faces encontradas.

Código 6.33: Finalizando os tempos e acessando o diretório de teste.

```
tempFinal = time.time()
caminhos = [os.path.join('Imagens Teste', f) for f in os.listdir('Imagens
Teste')]
```

Cria a variável final de tempo e cria uma variavel para acessar novamente o diretório de Imagens teste.

Código 6.34: Impressão dos resultados na tela.

```
print('Quantidade de imagens no diretório de teste:{}'.format(len(caminhos)))
print('Quantidade de imagens que foram reconhecidas:
{}'.format(idReconhecido))
print('Quantidade de imagens que não foram reconhecidas: {}'.format(cont -
idReconhecido))
print('Taxa de acerto: {:.2f}%'.format((idReconhecido / len(caminhos))*100))
print('Acuracia da predict: {}'.format(reconhecedor.getThreshold()))
print('Tempo total de reconhecimento: {:.2f} segundos'.format(tempFinal -
tempInicial))
print('Tempo medio de reconhecimento de cada imagem: {:.2f}
segundos'.format(sum(tempos) / len(tempos)))
print('Maior tempo de reconhecimento: {:.2f} segundos'.format(max(tempos)))
print('Menor tempo de reconhecimento: {:.2f} segundos'.format(min(tempos)))
```

E depois, é mostrado na tela todos os resultados dos tempos, e resultados de porcentagem de precisão.

Nas figuras abaixo, pode-se analisar melhor o resultado final de todos os ids passados pelo reconhecedor.

Figura 6.21: Resultado da primeira imagem analisada.

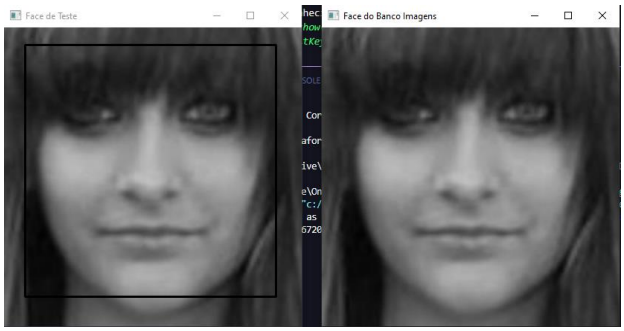


Figura 6.22: Resultado da segunda imagem analisada.

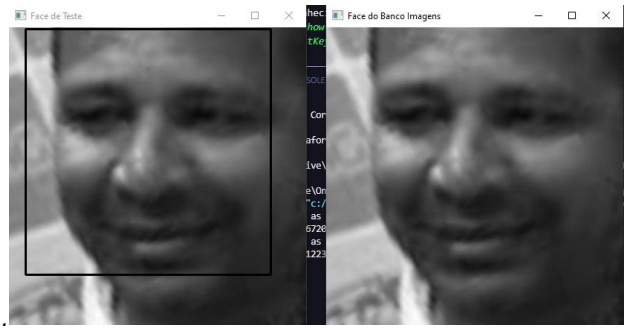


Figura 6.23: Resultado da terceira imagem analisada.

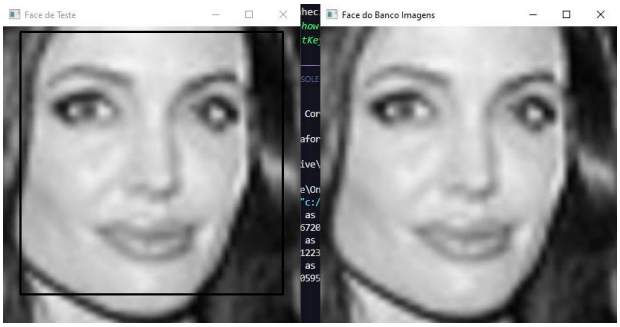


Figura 6.24: Resultado da quarta imagem analisada.

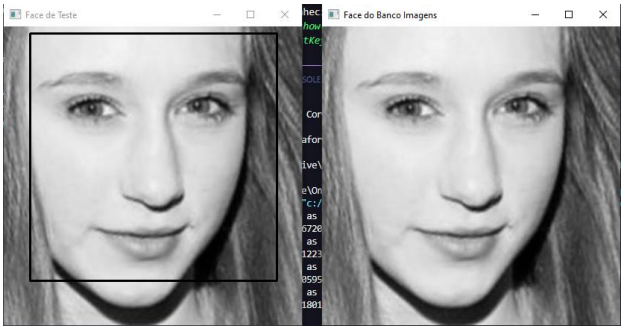


Figura 6.25: Resultado da quinta imagem analisada.

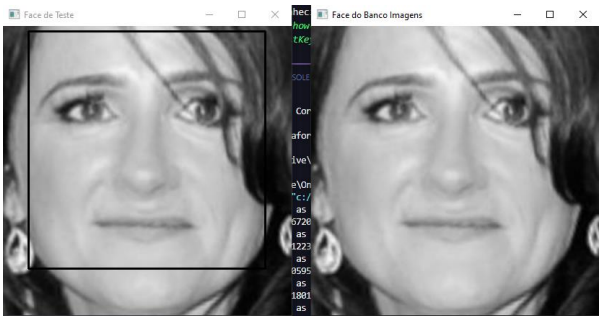


Figura 6.26: Resultado da sexta imagem analisada.

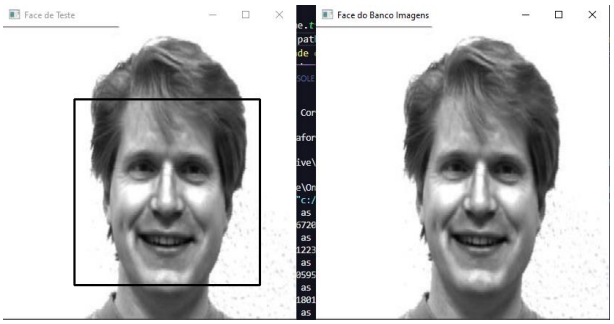


Figura 6.27: Resultado da sétima imagem analisada.

Figura 6.28: Resultado da oitava imagem analisada.

Figura 6.29: Resultado da nona imagem analisada.

Figura 6.30: Resultado da décima imagem analisada.

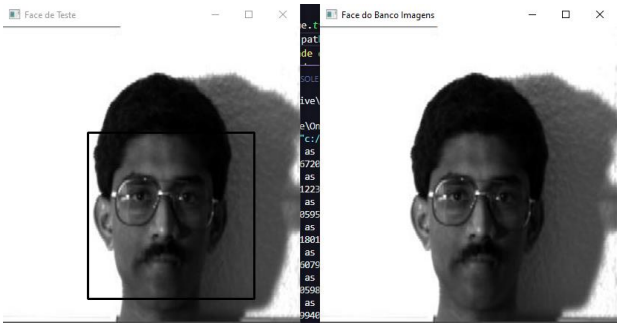


Figura 6.31: Resultado dos tempos e resultado da precisão do algoritmo

```
Quantidade de imagens no diretório de teste: 10
Quantidade de imagens que foram reconhecidas: 10
Quantidade de imagens que não foram reconhecidas: 0
Taxa de acerto: 100.00%
Acuracia da predict: 1.7976931348623157e+308
Tempo total de reconhecimento: 25.07 segundos
Tempo medio de reconhecimento de cada imagem: 1.97 segundos
Maior tempo de reconhecimento: 15.21 segundos
Menor tempo de reconhecimento: 0.39 segundos
```

Figura 6.32: Resultado das predições.

```
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 34
Confiança: 1524.0024551672063
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 56
Confiança: 3214.929627112234
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 81
Confiança: 3440.5331322059506
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 144
Confiança: 4417.920422618019
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 281
Confiança: 3979.7398493607925
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 287
Confiança: 4941.138729205983
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 317
Confiança: 4887.935511399408
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 339
Confiança: 5306.926344941472
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 400
Confiança: 3274.668727971563
ID da imagem que contém as mesmas caracteristicas no diretorio de faces reconhecidas: 422
Confiança: 3444.971452172442
```

Figura 6.33: Resultado das predições.

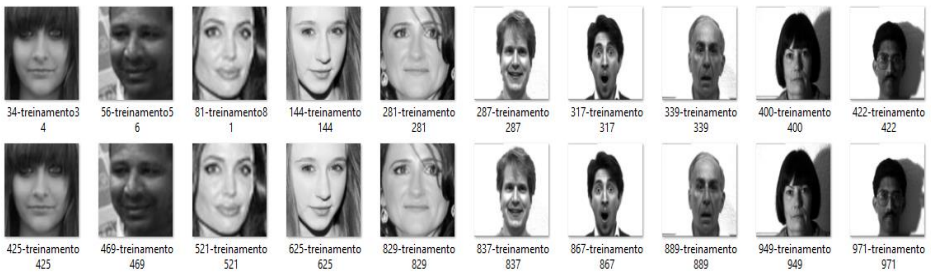
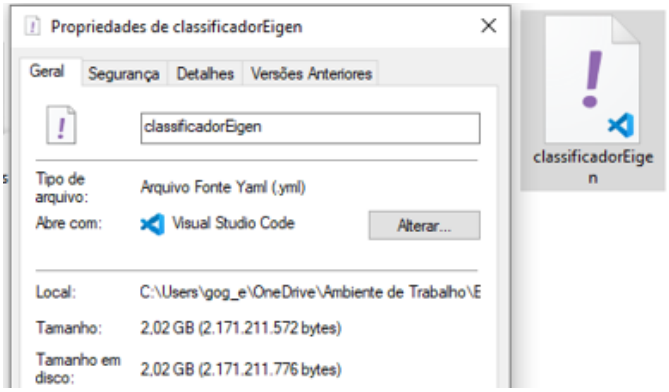


Figura 6.33: Tamanho do arquivo de modelo de treinamento.



Como se constata pelos resultados apresentados, o algoritmo de Eigenfaces é um excelente reconhecedor de faces, sua precisão alta, fazendo com que ainda seja utilizado nos sistemas de reconhecimento facial atuais em situações onde algoritmos de redes neurais não são ou não possam ser utilizados.

Com isso finalizamos a explicação e implementação do método/algoritmo EigenFaces e no próximo capítulo, vamos ver o método/algoritmo implementado por Viola e Jones, conhecido como Haar-Cascade.

7

HAAR-CASCADE

O *framework* proposto por Paul Viola e Michael Jones, Detecção Rápida de Objetos utilizando uma Cascata Impulsionada de Recursos Simples, é uma abordagem de aprendizado de máquina capaz de realizar o processamento de imagens rapidamente obtendo alta acurácia na tarefa de detecção de objetos.

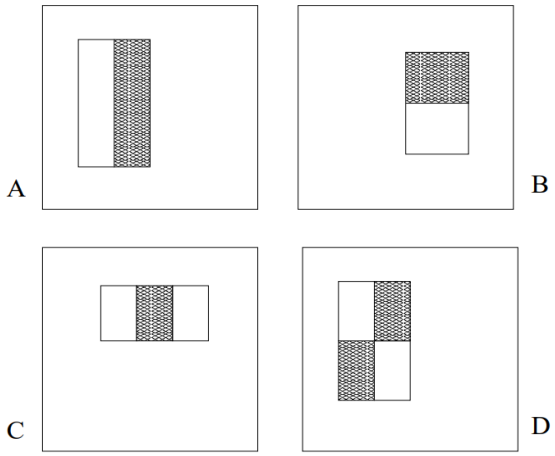
Este framework tem como base três contribuições-chave introduzidas pelos autores. A primeira é a Imagem Integral, uma representação de imagem criada para realizar cálculos eficientemente. A segunda é um algoritmo de aprendizado baseado em AdaBoost, que seleciona um pequeno grupo de recursos críticos de um grupo maior. A terceira é um método de combinar classificadores cada vez mais complexos numa estrutura de cascata, aumentando a velocidade do detector focando somente nas regiões promissoras da imagem. A cascata pode ser vista como um mecanismo de foco de atenção específico do objeto que, ao contrário de abordagens de detecção de objetos anteriores, é capaz de fornecer garantias estatísticas de que regiões descartadas não possuem, de fato, o objeto de interesse. Comparável aos melhores sistemas de detecção da época, este sistema pode ser utilizado em aplicações em tempo-real e, em 2001, alcançava 15 quadros por segundo num Intel Pentium III 700 MHz.

7.1 Recursos Semelhantes à Haar (Haar-Like Features)

O procedimento de análise de objetos é baseado em recursos ou características (do inglês, *features*). O primeiro motivo desta escolha é que os recursos podem codificar o que seria considerado conhecimento de domínio, ou seja, dados muito específicos, que dificultariam o processo de aprendizagem em uma base de dados limitada. O segundo motivo é que um sistema baseado em recursos opera mais rapidamente do que um sistema baseado diretamente em pixels.

Utilizando a Imagem Integral, tem-se que o valor de um recurso de dois retângulos é a soma da área direita subtraída da soma da área esquerda. O valor de um recurso de três retângulos é a soma das áreas dos retângulos externos subtraída da área da região central. O valor de um recurso de quatro retângulos é a diferença entre a soma dos pares diagonais das áreas. Cada recurso pode ser utilizado vertical e horizontalmente.

Figura 7.1 - Representação de quatro recursos.



Fonte: adaptado de Paul Viola e Michael Jones (2001, p. 2).

A: recurso vertical de dois retângulos. B: recurso vertical de dois retângulos. C: recurso horizontal de três retângulos. D: recurso vertical de quatro retângulos vertical.

Os autores utilizaram como resolução base do detector 24x24 pixels. Isso acarreta em mais de 180.000 possibilidades de cálculos de características Haar (recursos).

Embora existam alternativas mais sofisticadas do que recursos semelhantes à Haar, o agrupamento destas em conjunto com a Imagem Integral é capaz de fornecer uma representação rica da imagem, que suporta um aprendizado efetivo.

7.2 Imagem Integral

A Imagem Integral é um tipo de representação de imagem introduzida por Frank Painter em 1984. Os autores deste framework, Paul Viola e Michael Jones, em compensação, popularizaram o uso desta técnica no campo da visão computacional. Esta é uma das ferramentas mais importantes quando se trata de computação de recursos, sendo utilizada em muitas aplicações de detecção de objetos.

O valor de qualquer ponto (x,y) numa área quadrada é a soma de todos os pixels acima e à esquerda de (x,y), inclusive.

Fórmula 7.1 - Cálculo da posição (x,y) na Imagem Integral.

$$I(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y')$$

Fonte: adaptado de Paul Viola e Michael Jones (2001, p. 2).

$I(x,y)$ é o valor do pixel em (x,y).

A tabela de áreas somadas pode ser computada necessitando de apenas uma única passagem sobre a imagem, já que o valor do pixel em (x, y) é:

Fórmula 7.2 - Valor do pixel em (x, y).

$$I(x, y) = i(x, y) + I(x, y - 1) + I(x - 1, y) - I(x - 1, y - 1)$$

Fonte: disponível em https://en.wikipedia.org/wiki/Summed-area_table.

Uma vez que a tabela de somas foi computada, realizar a avaliação da soma de intensidade em qualquer área retangular exige exatamente quatro adições, independentemente do tamanho da área. Ou seja, tendo $A = (x0, y0)$, $B = (x1, y0)$, $C = (x0, y1)$, $D = (x1, y1)$, a soma da área de $i(x, y)$ é:

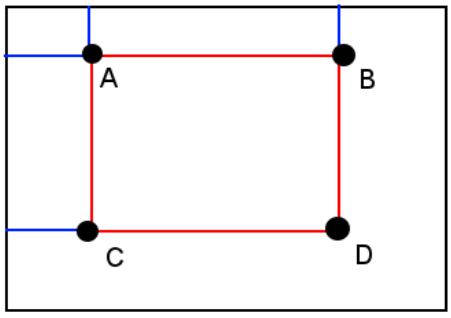
Fórmula 7.3 - Valor do pixel em (x, y).

$$\sum_{x0 \le x \le x1, y0 < y \le y1} = I(D) + I(A) - I(B) - I(C)$$

Fonte: disponível em https://en.wikipedia.org/wiki/Summed-area_table.

Os valores de A, B, C e D em uma imagem devem ser graficamente interpretados conforme a Figura 7.2.

Figura 7.2 - Quatro pontos utilizados para calcular determinada área em uma Imagem Integral.



Fonte: disponível em https://en.wikipedia.org/wiki/Summed-area_table.

Neste ponto, já é possível realizar a verificação das características Haar utilizando os valores da Imagem Integral. Observe o exemplo a seguir. É apresentada uma representação de imagem em tons de cinza, onde cada pixel possui um determinado valor numérico.

Figura 7.3 - Representação de imagem em tons de cinza.

5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

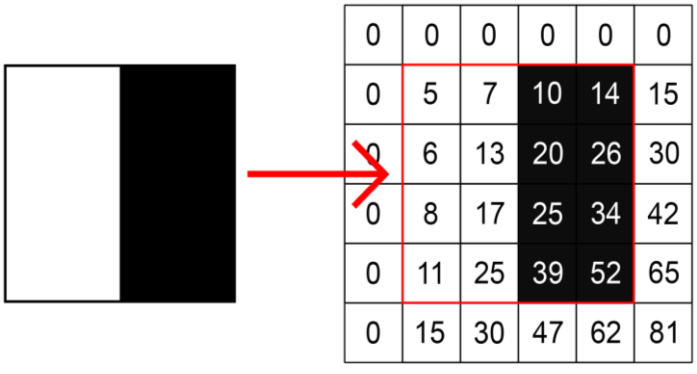
É necessário calcular a Imagem Integral da imagem representada na Figura 7.3 e, para isso, é utilizada a Fórmula 7.1. Numa aplicação prática, todos os valores seriam computados e adicionados em suas respectivas posições numa nova matriz bi-dimensional, como demonstrado na Figura 7.4.

Figura 7.4 - Representação da Imagem Integral.

0	0	0	0	0	0
0	5	7	10	14	15
0	6	13	20	26	30
0	8	17	25	34	42
0	11	25	39	52	65
0	15	30	47	62	81

Todos os valores da Imagem Integral já foram devidamente computados e uma nova representação da imagem foi criada e, portanto, as características Haar já podem ser calculadas. Neste caso, um recurso vertical de dois retângulos será aplicado na Imagem Integral, conforme a Figura 7.5.

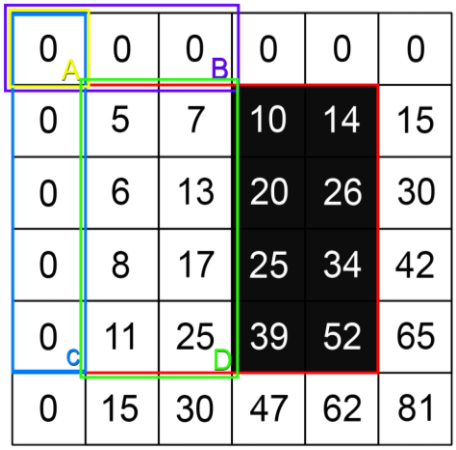
Figura 7.5 - Aplicação de uma característica Haar vertical de dois retângulos na Imagem Integral.



Para calcular o valor da aplicação do recurso utilizado, é utilizada a Fórmula 7.3. O resultado desta aplicação informará a diferença de luminosidade entre dois pontos desta determinada localização da imagem.

Observa-se que a caraterística que será aplicada é formada essencialmente por dois retângulos de cores diferentes. Para calcular o valor desta, deve-se, inicialmente, calcular o valor da área do retângulo esquerdo.

Figura 7.6 - Representação dos pontos para cálculo do retângulo esquerdo.



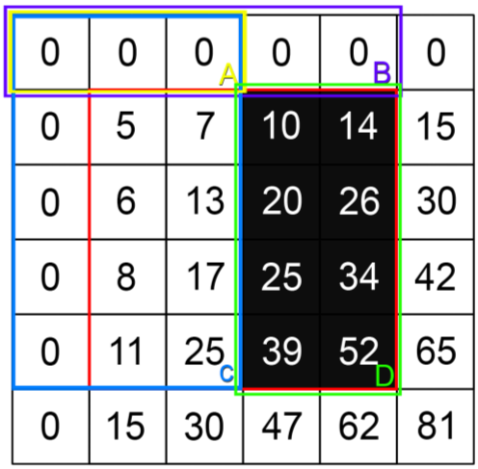
0	0	0	0	0	0
0	5	7	10	14	15
0	6	13	20	26	30
0	8	17	25	34	42
0	11	25	39	52	65
0	15	30	47	62	81

Utilizando a Fórmula 7.3, calcula-se:

$$\Sigma ie(x,y) = I(D) + I(A) - I(B) - I(C)$$
$$\Sigma ie(x,y) = 25 + 0 - 0 - 0$$
$$\Sigma ie(x,y) = 25$$

Em seguida, calcula-se o valor da área do retângulo direito.

Figura 7.7 - representação dos pontos para cálculo do retângulo direito.



0	0	0	0	0	0
0	5	7	10	14	15
0	6	13	20	26	30
0	8	17	25	34	42
0	11	25	39	52	65
0	15	30	47	62	81

Utilizando a Fórmula 7.3, calcula-se:

$$\Sigma id(x,y) = I(D) + I(A) - I(B) - I(C)$$
$$\Sigma id(x,y) = 52 + 0 - 25 - 0$$
$$\Sigma id(x,y) = 27$$

Conforme informado anteriormente, tem-se que o valor de um recurso de dois retângulos é a soma da área direita subtraída da soma da área esquerda. Portanto, calcula-se:

$$Recurso = \Sigma id - \Sigma ie$$
$$Recurso = 27 - 25$$
$$Recurso = 2$$

Neste caso, pelo fato das intensidades de brilho da imagem original terem valores próximos, o valor da aplicação do recurso não foi alto. Ou seja, há alta probabilidade da localização da imagem em questão não possuir nenhuma borda ou alguma característica que auxiliará efetivamente na detecção do objeto.

7.3 Funções de Classificação de Aprendizagem

Neste framework foi utilizada uma variação do AdaBoost que seleciona um grupo pequeno de recursos, além de treinar o classificador. No entanto, em sua forma original, o algoritmo de aprendizado AdaBoost é utilizado para acelerar a performance de classificação de um algoritmo de aprendizado fraco. Ao se utilizar o classificador forte (formado pela combinação linear de classificadores fracos), a taxa de erro se aproxima a zero exponencialmente, de acordo com o número de iterações (Schapire et al., 1998).

Os autores deste framework, após observarem o imenso número de possíveis recursos que poderiam ser calculados em cada subjanela da imagem, criaram uma hipótese de que existe um número específico e limitado de características que melhor separam uma imagem positiva (onde há o objeto de detecção presente) de uma negativa (onde não há o objeto de detecção presente). Por exemplo, uma característica Haar pode ser classificada como relevante ou irrelevante, dependendo do local onde foi aplicada. Ao aplicar um recurso vertical de três retângulos na ponte do nariz, este deve mostrar a diferença de brilho entre a ponte do nariz (região entre os olhos) e a região dos olhos, se tornando um recurso relevante. Em compensação, ao aplicar o mesmo recurso nos lábios de uma face, este não irá apontar quase ou nenhuma diferença de luminosidade, se tornando um recurso irrelevante. É necessário, portanto, identificar quais características Haar são relevantes e suas respectivas localizações na imagem.

Figura 7.8 - Exemplo de relevância de recursos.

Fonte: (adaptado) disponível em https://www.embecosm.com/2020/03/04/face-detection-with-the-edgetpu-using-haar-cascades/.

7.3.1 O Algoritmo Adaboost

Observa-se que, na prática, não existe um único recurso que separa com alta precisão uma imagem positiva de uma negativa. Na prática, imagens positivas e

negativas em tons de cinza (imagens em tons de cinza possuem somente um canal, aumentando a eficiência dos cálculos) serão fornecidas como exemplo. Todas as imagens terão o mesmo peso inicialmente. Para cada característica Haar disponível, um classificador será treinado a partir desta. No final da iteração, será selecionado o que apresentou menor taxa de erro. Este processo será repetido até a taxa de erro alcançar o número desejado ou até que o número de recursos seja alcançado.

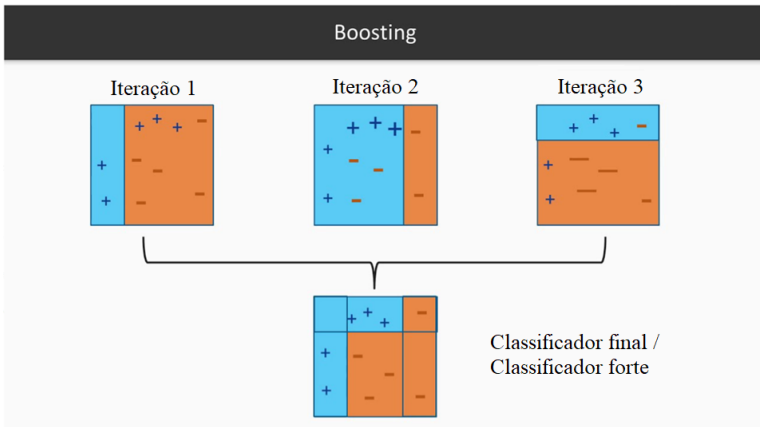
Desta forma, a principal ideia do AdaBoost é definir pesos para determinados classificadores para que estes sejam forçados a dar maior foco para observações que são difíceis de se classificar corretamente. Este processo é feito sequencialmente, ou seja, pesos que antes teriam o mesmo valor, serão ajustados de acordo com a performance da iteração passada. Em seguida, após o AdaBoost identificar quais características são relevantes e suas principais localizações, será criado um classificador forte, composto pela soma linear de classificadores fracos. Para finalizar, o resultado deste treinamento deve ser exportado como um arquivo XML (extensão ".xml") que irá conter as principais características Haar e suas respectivas localizações nas imagens.

O Treinamento do Algoritmo Adaboost

Essencialmente, o algoritmo utiliza uma combinação entre classificadores fracos para criar um classificador forte que será utilizado para detectar objetos.

Os classificadores fracos são criados ao se deslizar a subjanela sobre determinada imagem e calculando as características Haar. O resultado desse cálculo é comparado a um limite aprendido que separa imagens positivas de negativas.

Figura 7.9 - Processo de criação de um classificador forte.



Fonte: (adaptado) disponível em https://www.youtube.com/watch?v=BoGNyWW9-mE.

Cada classificador fraco possui um respectivo classificador forte.

Após todos os classificadores fracos e seus respectivos classificadores fortes serem encontrados, deve-se combiná-los a fim de formar um classificador em cascata. Este será composto por diversos estágios, onde cada um dos estágios é composto por um grupo de classificadores fracos.

Figura 7.10 - Formação de um classificador em cascata.

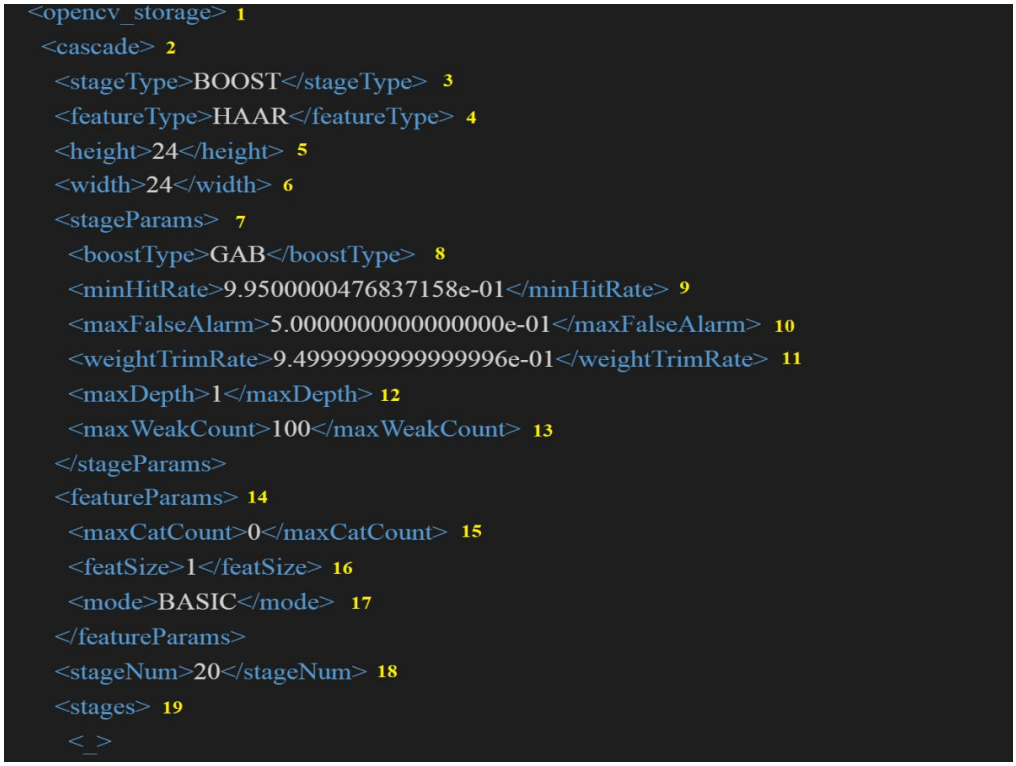
Fonte: (adaptado) disponível em https://technopediabphc.wordpress.com/2016/06/12/car-detector/.

Inicialmente, imagens (rotuladas devidamente) positivas e negativas serão passadas como parâmetro para uma função de treino de detecção de objeto em cascata (trainCascadeObjectDetector). Em sequência, após realizar o treino das características mais relevantes, estas serão divididas em estágios em ordem de complexidade, formando o classificador em cascata (estágio um, estágio dois, [...]). Os estágios, por sua vez, serão armazenados no arquivo XML. Para finalizar, linguagens de programação e frameworks podem utilizar este arquivo para construir uma biblioteca ou função, por exemplo.

7.3.2 Analisando O Arquivo Xml

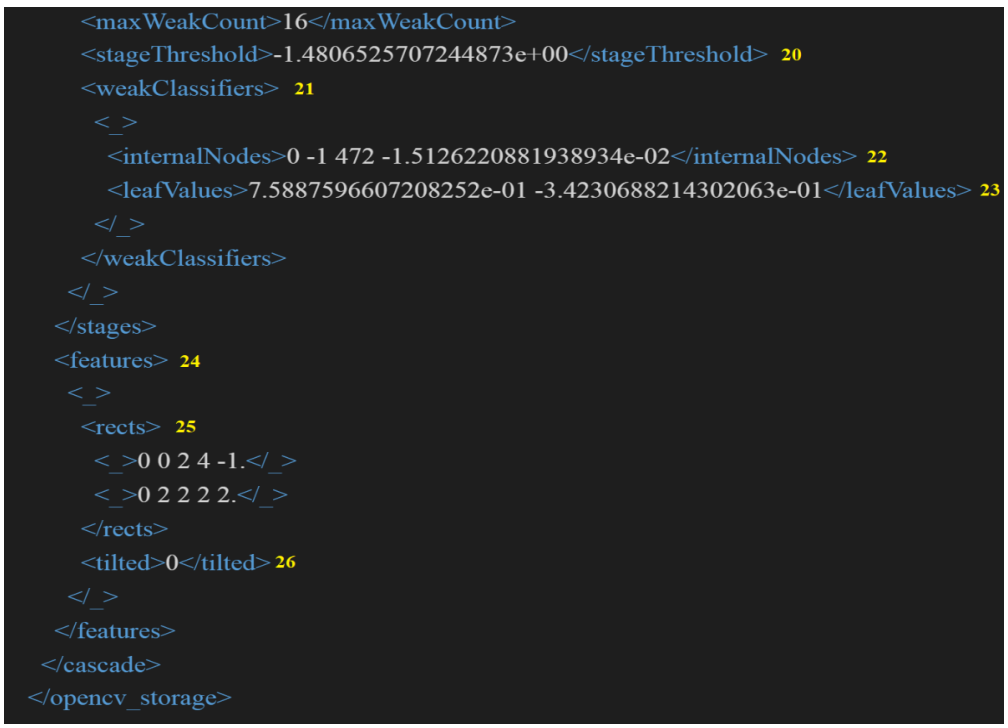
São necessários diferentes arquivos XML para diferentes processos de detecção. Logo, para se identificar uma caneta, por exemplo, é necessário um arquivo com os dados respectivos deste objeto. Para identificar um rosto, por sua vez, é necessário outro arquivo com seus respectivos dados.

O arquivo XML com os principais dados utilizados no processo de detecção facial frontal pode ser encontrado através do seguinte link: https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml



Observação: estão presentes somente algumas partes específicas do documento XML citado; os números presentes ao lado de determinadas tags não estão presentes no arquivo original e só estão presentes nesta figura para efeito de ilustração.

Figura 7.11 (B) - Arquivo XML.



Observação: estão presentes somente algumas partes específicas do documento XML citado; os números presentes ao lado de determinadas tags não estão presentes no arquivo original e só estão presentes nesta figura para efeito de ilustração.

A explicação das marcações (*tags*) presentes na Figura 7.11 (partes A e B) são explicadas conforme numeração nas figuras:

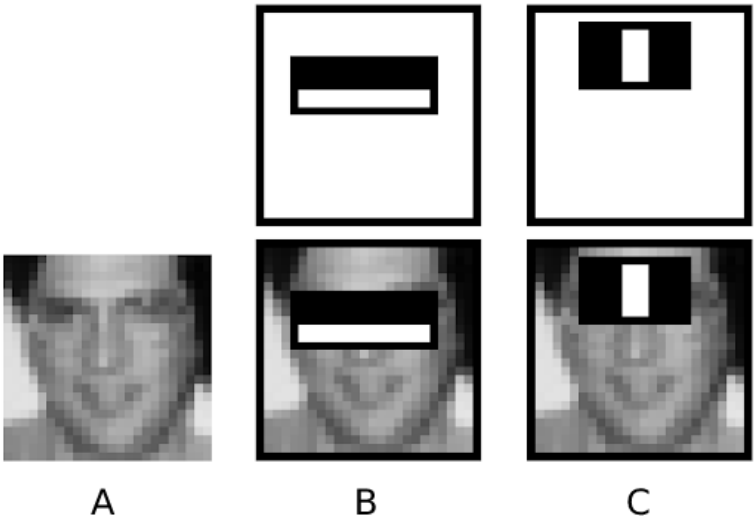
1. opencv_storage: marcação necessária para o XML ser válido.
2. cascade: marcação que contém o descritor de recursos.
3. stageType: tipo de estágio. Neste caso, informam que as cascatas estão aumentando em BOOST.
4. featureType: tipo de recursos. Neste caso, HAAR.
5. height: altura dos filtros utilizados pelos classificadores.
6. width: largura dos filtros utilizados pelos classificadores.
7. stageParams: define os parâmetros gerais dos estágios.
8. boostType: define o tipo de impulsionador.
9. minHitRate: taxa de acerto mínima desejada para cada estágio do classificador.
10. maxFalseAlarm: taxa máxima de falsos alarmes desejada para cada estágio do classificador.
11. weightTrimRate: limite entre 0 e 1 usado para economizar tempo computacional.
12. maxDepth**:** valor de profundidade.
13. maxWeakCount: o número máximo de classificadores fracos em cada estágio.
14. featureParams**:** região onde serão definidos os parâmetros dos recursos.
15. maxCatCount: dimensiona as respostas dos recursos para o intervalo correto e cria classificadores fracos que diferem regiões suficientemente.
16. featSize: método de varredura da imagem.
17. mode: modo de operação do algoritmo (BASIC).
18. stageNum: o número de estágios. Neste caso, 25.
19. stages: o número de classificadores ou o número de estágios. Cada estágio analisa a ativação de seus classificadores para decidir se existe um rosto na janela ou não.
20. stageThreshold: o limite que os classificadores precisam superar para avançar para o próximo estágio.
21. weakClassifiers: conjunto de classificadores fracos com base no qual uma decisão é tomada, se o objeto está na imagem ou não.
22. internalNodes: contém informações sobre os nós da árvore.
    - 1° valor: índice do nó atual.
    - 2° valor: índice do nó que você deseja ir. A transição da folha acaba quando o index se torna menor que 0.
    - 3° valor: número de filtros retangulares (localizado adiante no arquivo XML, abaixo da tag "features").
    - 4° valor: o valor limite do classificador INIweak.
23. leafValues: o valor do atributo Haar. O primeiro valor é retornado caso o resultado obtido seja menor do que o limite da árvore, se não, o segundo é retornado.
24. features: armazenam os retângulos de convolução.
25. rect: cria o retângulo da característica Haar.
    - 1° valor: valor da coordenada xl do retângulo.
    - 2° valor: valor da coordenada yl do retângulo.
    - 3° valor: valor da coordenada x2 do retângulo.
    - 4° valor: valor da coordenada y2 do retângulo.
    - 5° valor: se o quinto número for negativo (-1), então os pixels deste retângulo serão subtraídos. Se o número for positivo (3), são adicionados.
26. tilted: define a inclinação (0).

7.3.3 Resultado da Aprendizagem

Experimentos realizados pelos autores demonstraram que cerca de 200 recursos (selecionados a partir da variação do AdaBoost) alcançaram cerca de 95% de precisão na tarefa de detecção facial, com um falso positivo de 1 em 14084. Para aumentar a taxa de precisão, também é necessário aumentar a quantidade de recursos, consequentemente aumentando a necessidade de poder computacional e tempo.

A característica Haar selecionada que melhor separa imagens positivas de negativas é um recurso horizontal de dois retângulos, que foca na propriedade de que a região dos olhos é mais escura do que a região das bochechas e nariz. A segunda característica selecionada é um recurso vertical de três retângulos, que foca na propriedade de que a região dos olhos também é mais escura que a ponte do nariz.

Figura 7.12 - Demonstração da aplicação das duas principais características Haar em um rosto.



Fonte: adaptado de Paul Viola e Michael Jones (2001, p. 4).

A: Imagem positiva, onde há uma face presente. B: Aplicação de um recurso horizontal de dois retângulos na imagem positiva. Este recurso aponta a diferença de brilho entre a região dos olhos e a região das bochechas e nariz. C: Aplicação de um recurso horizontal de três retângulos na imagem positiva. Este recurso aponta a diferença de luminosidade entre a região da ponte do nariz e a região dos olhos.

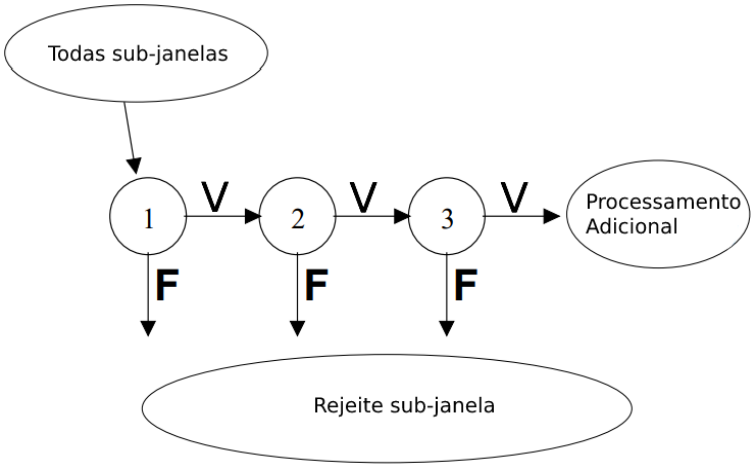
7.4 Cascata Atencional

Para reduzir a necessidade de poder e tempo de computação e aumentar a eficiência do framework, foi criada uma maneira inteligente de se rejeitar janelas negativas, simultaneamente detectando quase todas as janelas positivas. O método utilizado consiste em realizar avaliações em ordem de complexidade, dispensando a necessidade de se aplicar todas as características Haar numa determinada janela.

Se, ao se aplicar todas as características de um estágio numa determinada subjanela, estas possuam os valores (descritor de recursos) esperados, um segundo estágio será desencadeado. Se as características calculadas no segundo estágio também possuam os valores esperados, um terceiro estágio será desencadeado e assim sucessivamente. Caso seja obtido um resultado negativo (não esperado) no cálculo de

qualquer uma das características em qualquer estágio, esta subjanela será registrada como negativa, sendo imediatamente descartada. Observe a Figura 7.13, representando o processo da cascata atencional.

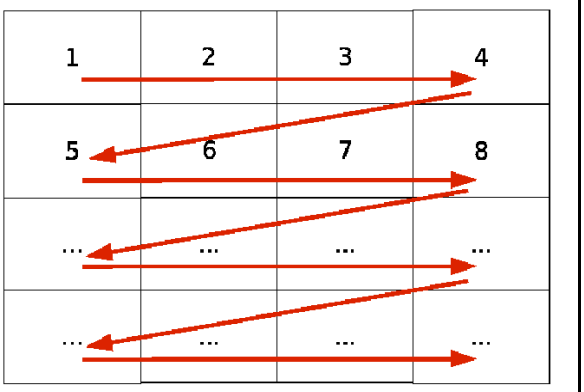
Figura 7.13 - Representação da estrutura em cascata.



Fonte: adaptado de Michael Jones e Paul Viola (2001, p. 5).

Cada um dos círculos representa um estágio do processo de classificação na cascata atencional. Dessa forma, quando se aplica uma determinada característica numa subjanela e o valor extraído não é o esperado no descritor de recursos, esta subjanela será rejeitada e continuará deslizando em forma de raster scanning, ou seja, da esquerda para a direita e de cima para baixo.

Figura 7.14 - Representação da movimentação Raster Scanning.



(A)

(B)

Fonte (A): disponível em https://www.researchgate.net/figure/Raster-Scan-Organization_fig1_228663375.

Fonte (B): (adaptado) disponível em https://unsplash.com/photos/rDEOVtE7vOs.

7.5 Resultados

Os autores treinaram um classificador em cascata com 38 estágios (*layers*) para treinar rostos retos e frontais. Para treinar o detector, foi utilizado um grupo de 4916 imagens positivas (onde há a presença de um rosto) com resolução de 24 x 24 pixels e um grupo de 9544 imagens negativas (onde não há a presença de um rosto).

O número de recursos nos primeiro 5 estágios (layers) é 1, 10, 25, 25 e 50, respectivamente. O restante de layers possuem cada vez mais características, alcançando um total de 6061.

7.6 Implementação do Algoritmo Haar-Cascade

Para implementação do sistema discutido, será utilizada uma biblioteca multiplataforma focada em desenvolvimento de aplicações de visão computacional: OpenCV. Utilizando esta biblioteca, é possível trabalhar com os algoritmos e conceitos analisados neste capítulo, além de permitir o controle de imagem e vídeo, com recursos como: módulos de Processamento de Imagens e Vídeo, Interface Gráfica de Usuário, calibração de câmera etc.

A OpenCV fornece tanto métodos de treinamento quanto módulos já treinados para detecção facial. Dessa forma, é possível realizar a validação de uma imagem sem a necessidade de treinar um classificador novo.

7.6.1 Código e Explicação

O Código 7.1 analisa uma base de imagens já transferida anteriormente e realiza a detecção facial em todas as imagens presentes no diretório em questão. É possível encontrar mais detalhes na obtenção desta base de dados e os procedimentos realizados nela no relatório final (compilação do autor).

Código 7.1 - Detecção de faces em massa.

```
import cv2
import urllib
import time

y_train_predict = [], timing = []


def haar_cascade(x_train):

urllib.urlretrieve("https://raw.githubusercontent.com/opencv/opencv/master/da
ta/haarcascades/haarcascade_frontalface_default.xml",
"haarcascade_frontalface_alt.xml")
    face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")

    for image in x_train:
        start_point = time()
```

```
        face = face_cascade.detectMultiScale(
            image, scaleFactor=1.1, minNeighbors=3, minSize=(30, 30))
        end_point = time()

        if len(face) != 0:
            y_train_predict.append(1)
        else:
            y_train_predict.append(0)

        timing.append(end_point - start_point)

    return y_train_predict, timing

# path = "caminho_das_imagens"
# haar_cascade(path)
```

Explicação do Código:

```
import cv2
import urllib
import time
```

São importadas as bibliotecas “cv2”, “urllib” e “time”, que serão utilizadas para o processamento de imagens, carregamento de arquivos necessários para a execução da aplicação e registro de tempo, respectivamente.

```
y_train_predict = [], timing = []
```

São criadas as variáveis “y_train_predict” e “timing”. Que retornarão os resultados da aplicação e o tempo de processamento dos mesmos, respectivamente.

```
def haar_cascade(x_train):
```

É definida a função “haar_cascade”. Esta será responsável por retornar os resultados obtidos nas análises das imagens e de seus respectivos tempos. A função recebe como parâmetro o caminho do diretório que contém as imagens, "x_train".

```
urllib.urlretrieve("https://raw.githubusercontent.com/opencv/opencv/maste
r/data/haarcascades/haarcascade_frontalface_default.xml",
"haarcascade_frontalface_alt.xml")
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")
```

A variável "face_cascade" carregará o modelo de detecção. É importante notar que o modelo irá buscar características baseadas no arquivo de detecção. Em outras palavras, para detectar faces frontais, deve-se fornecer um arquivo XML com tais características. Neste caso, tal arquivo foi nomeado como "haarcascade_frontalface_alt.xml".

Em sequência, para efetivamente criar o modelo, é possível utilizar a função CascadeClassifier (disponível na biblioteca OpenCV), onde seu único argumento é o próprio arquivo de detecção.

```
for image in x_train:
```

Para cada imagem presente no diretório, deve-se, em ordem:

```
start_point = time()
```

Registrar o momento antes da detecção.

```
image, scaleFactor=1.1, minNeighbors=3, minSize=(30, 30))
```

Verificar a presença de faces. O resultado será armazenado em uma variável "face". Para verificação, é possível utilizar a função detectMultiScale. Esta possui como argumentos a imagem que será analisada, o número real descrevendo o fator de escala, um inteiro descrevendo a quantidade de vizinhos mínimos e uma tupla descrevendo o tamanho mínimo da imagem, respectivamente.

```
end_point = time()
```

Registrar o momento após a detecção.

```
if len(face) != 0:
    y_train_predict.append(1)
else:
    y_train_predict.append(0)
```

Neste ponto, o resultado da detecção já está armazenado no objeto "face". Verifica-se se a quantidade de itens presentes no objeto é diferente de 0. Caso verdadeiro, adiciona-se o valor 1 à variável "y_train_predict". Caso contrário, 0.

```
timing.append(end_point - start_point)
```

Finalizando o laço, subtrai-se o tempo final do tempo inicial, obtendo o tempo de execução. Este valor é adicionado na lista "timing".

```
return y_train_predict, timing
```

Finalmente, devem ser retornadas as listas contendo os resultados individuais das imagens, bem como seus respectivos tempos de execução. Ou seja, retornam-se as listas "y_train_predict" e "timing".

```
# path = "caminho_das_imagens"
# haar_cascade(path)
```

É importante observar que a aplicação desenvolvida, caso executada sem alterações, não retornaria resultados. Isso ocorre pois a função principal (“haar_cascade”) não foi chamada. Para isso, deve-se remover os comentários das 2 linhas selecionadas acima e alterar o valor da variável “path”. Esta, por sua vez, receberá o caminho das imagens desejadas. Finalizando, deve-se adicionar a variável como argumento na função “haar_cascade”.

Para efeitos didáticos, foi desenvolvido outra aplicação onde seu resultado pode ser analisado mais facilmente. Neste caso, o Código 7.2 pode ser utilizado na tarefa de detecção facial frontal, estando escrito na linguagem de programação Python versão 3.10.

Código 7.2 - Detecção de face frontal.

```
import cv2

image = cv2.imread("img.jpg")
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
faces = detector.detectMultiScale(image_gray, scaleFactor = 1.1,
minNeighbors = 10, minSize = (30, 30))
```

```
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.imshow("Faces", image)
cv2.waitKey(0)
```

Explicação do Código:

```
import cv2
```

É importada a biblioteca “cv2” (OpenCV).

```
image = cv2.imread("img.jpg")
```

É criada uma variável “image”, que terá como valor a leitura da imagem desejada utilizando a função “imread”, da OpenCV.

Figura 7.15 - Imagem com face.

Fonte: disponível em https://unsplash.com/photos/rDEOVtE7vOs.

Esta imagem JPG deve estar contida no mesmo diretório que contém o arquivo Python. Além disso, deve ser nomeada como “img.jpg”.

```
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

É criada uma variável “image_gray”, que armazenará a imagem lida convertida para tons de cinza, utilizando a função “cvtColor”, da OpenCV. Os argumentos desta função são a própria imagem e a tonalidade desejada, respectivamente.

Figura 7.16 - Imagem com face em tons de cinza.

Fonte: (adaptado) disponível em: https://unsplash.com/photos/rDEOVtE7vOs.

```
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
```

É criada uma variável “detector” que será utilizada para calcular as características Haar utilizando a função “CascadeClassifier”, da OpenCV. O argumento desta função é o arquivo XML previamente apresentado: “haarcascade_frontalface_default.xml”. Este arquivo contém o descritor de recursos respectivo a detecção facial frontal. Observação: o detector sempre será respectivo ao arquivo XML, ou seja, caso seja passado como argumento um arquivo XML respectivo à faces, o detector será utilizado para detectar faces. Por outro lado, caso seja passado como argumento um arquivo XML respectivo à olhos, o detector será utilizado para detectar olhos.

```
faces = detector.detectMultiScale(image_gray, scaleFactor = 1.1, minNeighbors = 10,
minSize = (30, 30))
```

É criada uma variável “faces” que será utilizada para realizar a detecção de objetos na imagem de entrada utilizando a função “detectMultiScale”. Os argumentos desta função são descritos abaixo:

- image_gray: é a imagem de entrada. Imagem onde será realizada a detecção de objetos.
- scaleFactor: fator de escala. O modelo treinado pelo OpenCV possui um tamanho fixo definido durante o treinamento (estes dados podem ser verificados analisando o documento XML). Quando uma nova imagem é apresentada na aplicação, este determinado tamanho é detectado na imagem (se houver). Em compensação, se as dimensões do rosto apresentado não forem as mesmas utilizadas no descritor de recursos presente no XML, é possível redimensionar uma face para maior ou menor (dependendo da situação) e, para isso, é alterado o valor deste parâmetro. Logo, caso necessário, pode-se utilizar o valor 1.05, que reduzirá o tamanho da imagem em 5%, deixando o rosto detectável para o algoritmo, por exemplo.
- minNeighbors: “vizinhos mínimos”. Para evitar ocorrências falso-positivas, são adicionados “vizinhos” na imagem. Assim, é adicionado um mínimo de ocorrências, para que a chance de realmente haver um rosto naquela janela seja aumentada. Se o valor 10 for atribuído, então será necessário que dez vizinhos também tenham detectado um rosto nas regiões próximas.
- minSize: determina o quão pequeno é o tamanho do objeto que deseja detectar. De início, (30,30) é um tamanho ideal.

```
for (x, y, w, h) in faces:
```

Utilizando uma estrutura de repetição “for”, ocorre uma iteração a partir dos valores de “x”, “y”, “w” e “h” (eixo X, eixo Y, largura e altura, respectivamente) da variável “faces”..

```
cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

A função “rectangle”, da OpenCV, recebe:

- 1° argumento: a variável “image”, com os rostos já detectados.
- 2° argumento: uma tupla contendo os pontos iniciais (x,y) do retângulo.
- 3° argumento: uma tupla contendo os pontos finais (x,y) do retângulo.
- 4° argumento: uma tupla contendo os valores RGB do retângulo.

- 5° argumento: a espessura do traço.

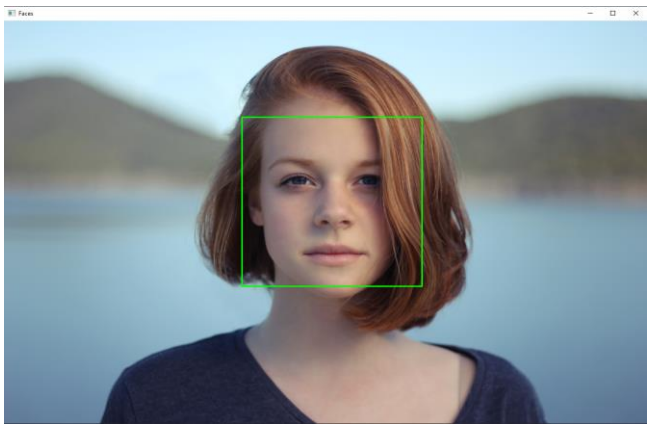
```
cv2.imshow("Faces", image)
```

A função “imshow”, da OpenCV, irá apresentar a imagem final, recebendo como primeiro parâmetro o nome da janela e como segundo a própria imagem “image”.

```
cv2.waitKey(0)
```

A função “waitKey”, da OpenCV, irá finalizar o programa quando for recebida determinada entrada do teclado. Neste caso, qualquer entrada.

Figura 7.17 - Imagem com face.



Fonte: (adaptado) disponível em: https://unsplash.com/photos/rDEOVtE7vOs.

O resultado final, ou seja, a imagem com a(s) face(s) destacada(s) é apresentada. O programa se encerra quando recebe alguma entrada do teclado.

7.6.2 Execução do Algoritmo

Utilizando uma base de imagens desenvolvida pelos autores do livro (disponível em https://www.kaggle.com/datasets/asacxyz/ic-fatecitu/download) totalizando 1000 imagens, onde 600 são positivas e 400 são negativas, será analisado o resultado do algoritmo através da Matriz de Confusão, curva ROC e tempo de detecção.

Figura 7.18

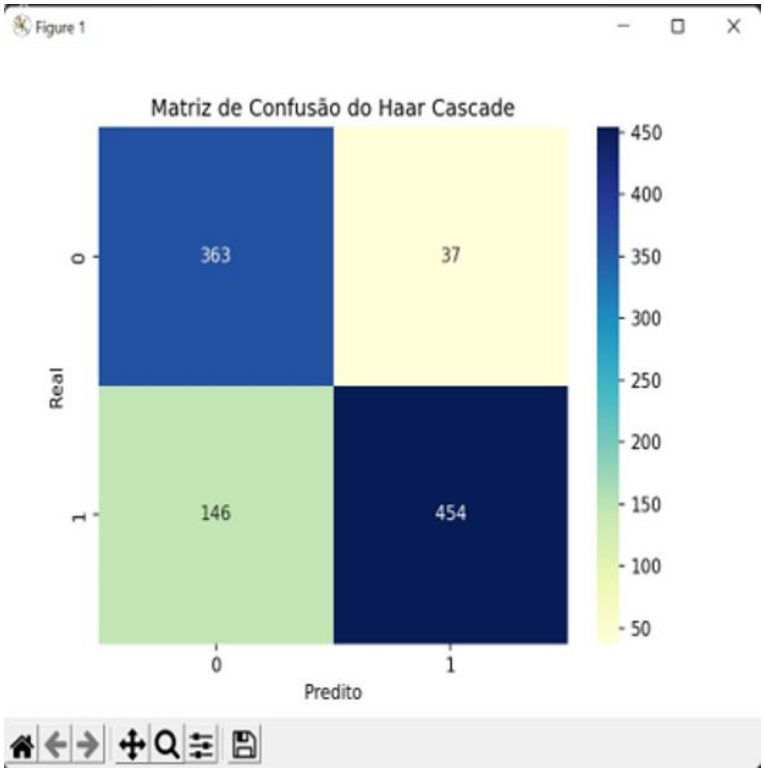


Figure 1
Matriz de Confusão do Haar Cascade
363
37
146
454
Real
0
1
Predito
0
1
450
400
350
300
250
200
150
100
50

Figura 7.19

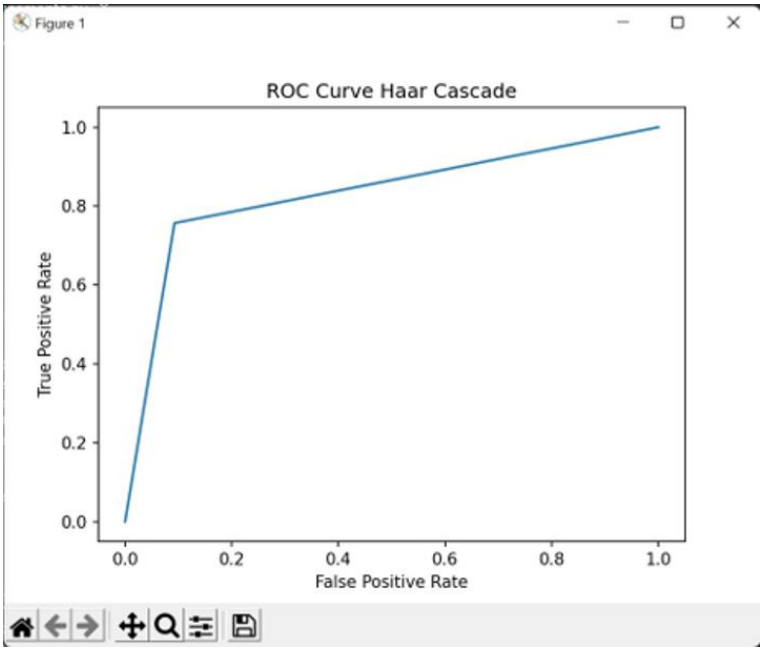
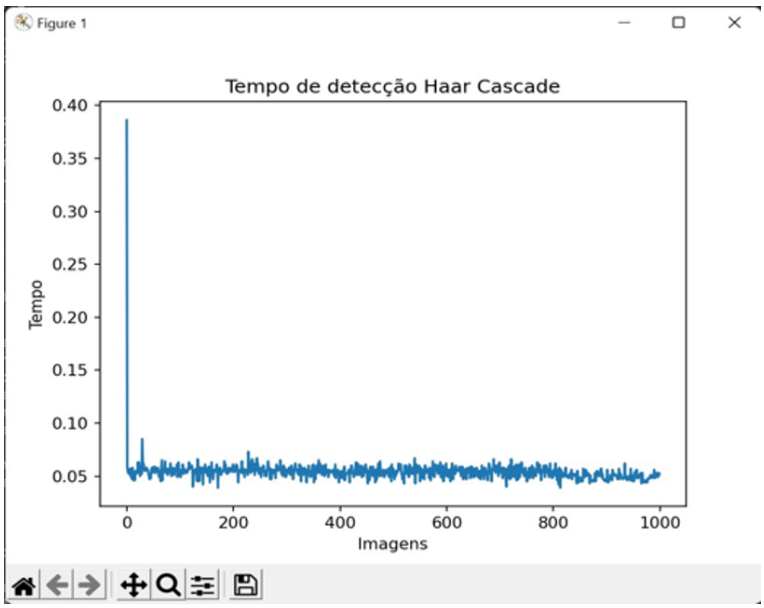


Figura 7.20



A matriz de confusão pode ser utilizada para avaliação de modelos de classificação em Aprendizado de Máquina. É uma tabela que apresenta a taxa de verdadeiro positivos (onde os dados obtidos são verdadeiros em ambas ocasiões), falso positivos (dado real é falso, mas o obtido é verdadeiro), falso verdadeiro (ambos os dados são falsos) e falso negativo (dado real é verdadeiro, mas o obtido é falso). Os resultados obtidos ao executar o algoritmo Haar Cascade foram 454 imagens verdadeiro

positivas, 146 imagens falso positivas, 363 imagens falso verdadeiras e 37 imagens falso negativas.

A curva ROC tipicamente representa a taxa de positivos-verdadeiros no eixo das ordenadas e a taxa de falso-positivos no eixo das abcissas. Ou seja, o canto superior esquerdo do gráfico é o ponto ideal, onde a taxa de falso-positivos é 0. Neste caso, nota-se que o ponto se encontra na taxa de aproximadamente 0.8 (indicando, ainda, que existe um número significativo de falso-positivos).

O gráfico de detecção apresenta o desempenho da aplicação, onde a quantidade de imagens percorridas está em função do tempo. Nota-se que o tempo médio para análise de uma imagem foi de aproximadamente 0.05 segundos.

Em suma, foram apresentados aqui o algoritmo Haar Cascade e a forma de funcionamento e implementação e duas diferentes maneiras de criação do arquivo xml com a cascata de recursos utilizada pelo algoritmo: a primeira utiliza uma máquina virtual e a compilação do detector na mesma utilizando linguagem Python; a segunda automatiza este processo utilizando o software Cascade Trainer GUI.

No próximo capítulo, vamos ver um outro método/algoritmo amplamente utilizado e com excelentes resultados: o HOG (do inglês Histogram of Oriented Gradients ou Histograma de Gradientes Orientados).

8

HOG (Histogram Oriented Gradients)

O presente capítulo abordará o descritor de recursos HOG (*Histogram of Oriented Gradients*) e a sua implementação para reconhecer faces em imagens e vídeos. Serão explicadas as diferenças e similaridades desse algoritmo com outros descritores, bem como o conceito de gradientes. Como forma de demonstração de seu funcionamento, é apresentado ao final a acurácia e o tempo de detecção que o HOG apresentou ao processar 1000 imagens, sendo 60% dessas imagens de pessoas. Através dessas informações, é possível ter uma ideia de sua perfomance.

8.1 HOG e outros descritores

Em 2005, os pesquisadores Dalal e Triggs apresentaram um descritor de recursos que extraía as bordas de um objeto para utilizá-las em sua detecção. O artigo utilizou imagens de pedestres com o intuito de testar o algoritmo na identificação de pessoas. Chamado de Histogram Oriented Gradients (HOG), o método proposto por Dalal e Triggs possui uma abordagem diferente dos descritores de características anteriores a ele. Todavia, HOG apresenta alguns processos semelhantes com esses mesmos trabalhos. Assim, um dos principais objetivos dos autores foi comparar esse algoritmo com ferramentas similares, levantando as diferenças encontradas no HOG e os resultados obtidos através dele.

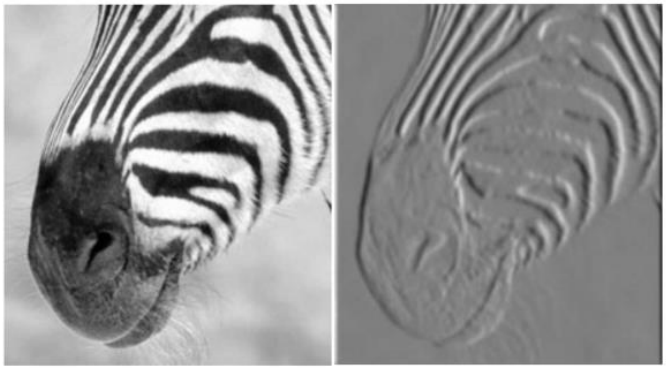
Histogram Oriented Gradients trabalha com a distribuição (histograma) de gradientes orientados de uma imagem para detectar um objeto ou uma pessoa. De maneira resumida, o gradiente representa como a intensidade da cor varia no decorrer da imagem. Essa mudança de tonalidade possui uma orientação, conforme mostrado na Figura 8.1. Além disso, quando os pixels de uma região apresentam uma alteração abrupta de cores, conseguimos identificar a borda de um corpo (vide Figura 8.2). Explicaremos com mais detalhes sobre gradientes no decorrer deste capítulo.

Figura 8.1: Representação simplificada de gradientes. As cores presentes em cada quadrado aumentam sua intensidade seguindo uma orientação: horizontal (imagem da esquerda), vertical (centro) e diagonal (direita).

Conforme Dalal e Triggs comentam no artigo, o uso de gradientes orientados na detecção de objetos não foi algo novo que apareceu em HOG. Muitos trabalhos anteriores já utilizavam essas características das imagens para detectar objetos e humanos. McConnell (1985) patenteou um descritor baseado em gradientes. Freeman e Roth (1995) desenvolveram um algoritmo que captava movimentos das mãos através desses atributos. Um ano depois, Freeman et al (1996) aplicaram esse trabalho em jogos digitais. O objetivo desses pesquisadores era permitir a interação do usuário com os jogos através de movimentos das mãos e do corpo, sem precisar de joysticks. Lowe (2004) criou um dos mais famosos descritores de recursos: o SIFT (Scale Invariant Feature Descriptor). Esse algoritmo utiliza gradientes orientados para extrair os pontos de um objeto que não são alterados quando a imagem é rotacionada ou distorcida. Esses pontos são chamados de pontos chave (keypoints).

Outro aspecto que HOG compartilha com outros descritores, em particular com o proposto por Belongie et al (2001), é o contexto de forma (Shape Context). O algoritmo de Belongie divide a imagem em blocos para conseguir extrair as bordas de um objeto. Shape Context não adota gradientes orientados para realizar esta tarefa, como ocorre em SIFT. Dalal e Triggs se basearam no trabalho de Belongie em dividir os pixels de uma imagem em quadros e aplicaram um histograma de gradientes orientados em cada célula, distinguindo HOG dos demais descritores.

Figura 8.2: Uso de gradientes orientados na detecção de bordas. A cabeça da zebra (esquerda) é formada através de mudanças significativas das cores na imagem. Note que a cor do fundo se altera bruscamente para tons de preto e branco, gerando o focinho e as listras do animal. A segunda imagem (direita) é um filtro que utiliza os gradientes para destacar os contornos do formato, ignorando os pixels que tiveram variações mínimas de cor.



Fonte: Forsyth e Ponce (2012), p. 145

Ao comparar o desempenho de HOG com Shape Context e os demais algoritmos de gradientes orientados, Dalal e Triggs afirmam que o seu descritor retorna menos falsos positivos na identificação de humanos. Isso porque os detectores de ponto chave anteriores ao HOG não apresentam uma estrutura voltada a identificação de pessoas. Além disso, HOG apresenta uma arquitetura mais simples e com excelente eficácia se confrontado com descritores *wavelet*[1].

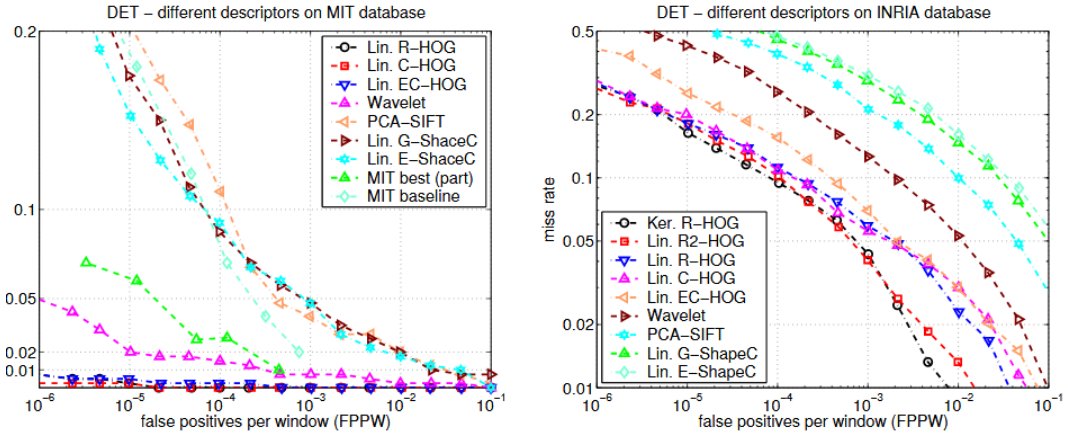
Figura 8.3: Exemplo de falso positivo, mostrando que descritores de recursos podem erroneamente identificar objetos e pessoas, dependendo das imagens analisadas. Ao invés de destacar a cabeça da pessoa, o descritor da foto realçou a sombra do indivíduo. HOG obteve resultados satisfatórios na redução dessa ocorrência.

Fonte: Mohan *et al* (2001), p. 353

Abaixo é mostrado o desempenho de HOG e de outros descritores na geração de falsos positivos (Figura 8.4). Note que HOG apresenta variações, dependendo da maneira como foi estruturada a lógica (por ex.: R-HOG e C-HOG). Veremos com mais detalhes sobre essas classes mais a frente. Dalal e Triggs utilizaram os bancos de imagens MIT e INRIA para testar os algoritmos.

[1] As funções *wavelet* (ou transformada de *wavelet*) são aplicadas para reduzir a complexidade dos dados, descartando os irrelevantes. São recursos do Cálculo Diferencial que utilizam integração para comprimir ou decompor um conjunto de dados em amostras mais fáceis de trabalhar. Nas áreas de Visão Computacional e Processamento de Imagens, essas funções são aplicadas em descritores de recursos para diminuir o número de características em uma imagem, também chamada de dimensionalidade. O mais famoso exemplo de descritor *wavelet* é o Haar-like, desenvolvido por Viola e Jones (2001). Mohan *et al* (2001) e Guo e Li (2013) apresentam uma breve explicação sobre *wavelet*s, enquanto o criador dessa função, Mallat (1989), explica o conceito sob uma visão matemática.

Figura 8.4: Gráficos mostrando o número de falsos positivos gerados pelos descritores de recursos. Percebe-se que HOG é mais eficaz nos dois bancos de imagens usados nos testes. Todos os demais descritores apresentam uma elevada taxa de perda, isto é, retornaram uma grande quantidade de falsos positivos. As variações de HOG foram mais assertivas, com um pequeno número de falsos positivos.



Fonte: Dalal e Triggs (2005), p. 4

8.2 Gradiente

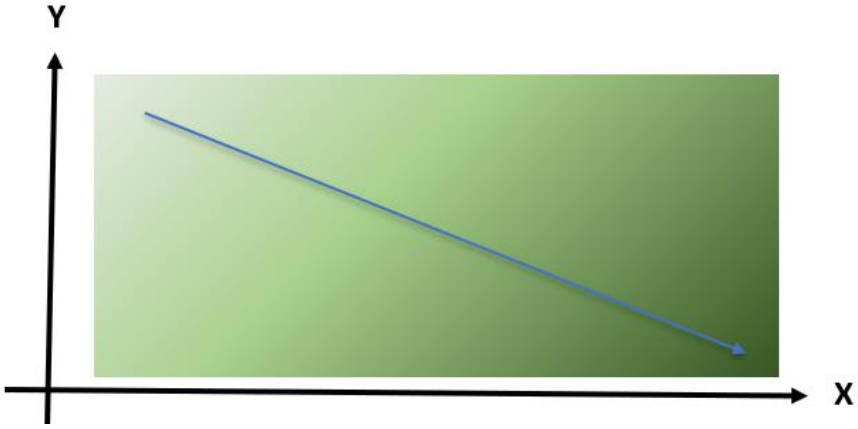
Como visto anteriormente, HOG é um descritor de recursos que detecta corpos através de gradientes orientados. Essas características da imagem mostram como a tonalidade de cor muda no objeto, com base em uma orientação (ver Figura 8.1). Ou seja, a intensidade da cor pode ser representada em um plano cartesiano, por meio dos eixos X e Y (Figura 8.5). Em uma abordagem matemática, o gradiente é obtido através da derivada de uma função multivariável (uma função f com n variáveis). Por exemplo, dada uma função com quatro variáveis independentes - $f(w, x, y, z)$ - o gradiente pode ser representado por

$$\nabla f = \frac{\partial f}{\partial w} i + \frac{\partial f}{\partial x} j + \frac{\partial f}{\partial y} k + \frac{\partial f}{\partial z} l$$

Onde:

- ∇f é o gradiente da função f;
- $\frac{\partial f}{\partial v}$ é a derivada de f em função da variável independente v;
- i, j, k e f são as coordenadas da função f em um plano cartesiano.

Figura 8.5: Representação simplificada de um gradiente. Perceba que a imagem está inserida em um plano cartesiano para melhor identificar a orientação do gradiente. Grande parte dos *pixels* mais claros tem números altos em Y e pequenos no eixo X. A tonalidade começa a crescer quando o valor de X aumenta gradativamente, enquanto em Y ocorre o inverso. De acordo com essa observação, conclui-se que o gradiente tem uma orientação diagonal.



Dada uma imagem m, sabemos que ela possui uma dimensionalidade 2-d (duas variáveis independentes). Assim, a imagem será $m(x, y)$. O gradiente terá a estrutura:

$$\nabla m = \frac{\partial m}{\partial x} i + \frac{\partial m}{\partial y} j.$$

Os conceitos que englobam a obtenção do gradiente pertencem ao Cálculo Diferencial. A derivada é o principal deles. De forma resumida, a derivada é a taxa de variação de uma função. Por isso que o gradiente é representado por esse recurso matemático, uma vez que a sua principal característica é a variação de tonalidade da cor. Sua definição é dada por:

$$\frac{\partial f}{\partial x} = \lim_{\partial x \to 0} \frac{f(x+\partial x) - f(x)}{\partial x}.$$

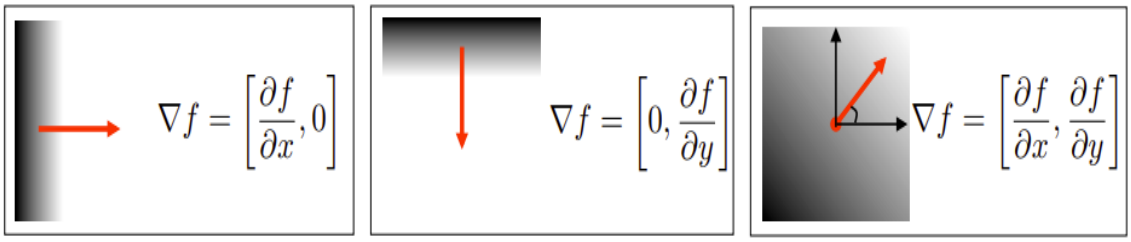
A função da imagem $m(x, y)$, por ser composta em duas variáveis independentes, terá duas derivadas:

$$\frac{\partial m}{\partial x} = \lim_{\partial x \to 0} \frac{m(x+\partial x, y) - m(x, y)}{\partial x}$$

$$\frac{\partial m}{\partial y} = \lim_{\partial y \to 0} \frac{m(x, y+\partial y) - m(x, y)}{\partial y}$$

Essas derivadas formarão os gradientes dos eixos X e Y, conforme vemos na Figura 8.6.

Figura 8.6: Representação gráfica das derivadas. O gradiente com orientação horizontal terá uma derivada em seu eixo X (imagem da esquerda), enquanto Y será igual a zero. Para uma orientação vertical, é em Y que haverá uma taxa de variação, e X permanecerá zerado (centro). Quando há um gradiente diagonal, os dois eixos não se tornam nulos (direita).



Fonte: Kitani (2017), p. 22

A orientação do gradiente também é obtida através de um cálculo, conforme mostrado abaixo.

$$\Theta = tan^{-1}\left(\frac{\partial f}{\partial x} \ / \ \frac{\partial f}{\partial y}\right)$$

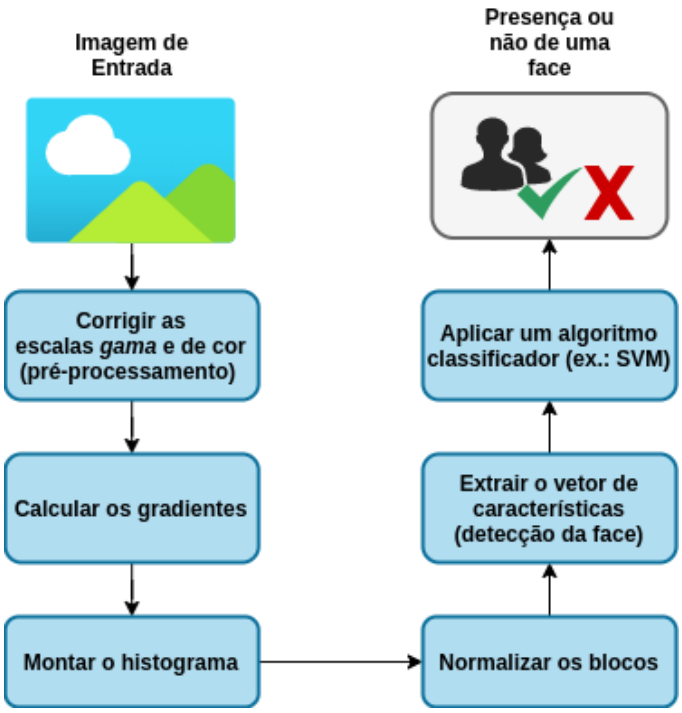
Além da orientação, o gradiente apresenta um outro atributo importante: a magnitude. Essa característica mostra o quão rápida a cor varia na imagem, sendo importante na detecção de mudanças bruscas na cor. É através da magnitude que é possível obter as regiões onde há a presença de bordas. Seu cálculo é dado por:

$$\mu = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

8.3 As etapas de HOG

Dalal e Triggs dividiram o seu descritor de recursos em 6 etapas, como mostrado na Figura 8.7:

Figura 8.7: Etapas da identificação facial com HOG.



Fonte: Adaptado de Dalal e Triggs (2005), p. 2.

8.3.1 Corrigir as escalas gama e de cor (pré-processamento)

Esta primeira etapa consiste em reduzir a dimensionalidade da imagem. Geralmente, as imagens de entrada são coloridas, ou seja, possuem valores nas três camadas da escala RGB (Red, Green, Blue). Dessa forma, o conjunto de características será tridimensional, gerando uma função multivariável de 3 parâmetros. Como vimos anteriormente, o cálculo do gradiente consiste em atribuir para cada eixo uma derivada, o que torna muito complexo o processamento da imagem. Isso porque a detecção das bordas precisa considerar três camadas de informação. Ademais, o contraste e brilho presentes na imagem podem afetar a detecção e o reconhecimento facial.

A etapa de pré-processamento tem como objetivo reduzir a dimensionalidade das características, com o foco de melhorar o processamento e desconsiderar o brilho e contraste presentes na imagem. Aplicar escalas de cinza ou fazer correções gama[2] são algumas técnicas de pré-processamento. Grande parte dos descritores de recursos utilizam alguma técnica de normalização de cor nas imagens, melhorando o seu desempe-

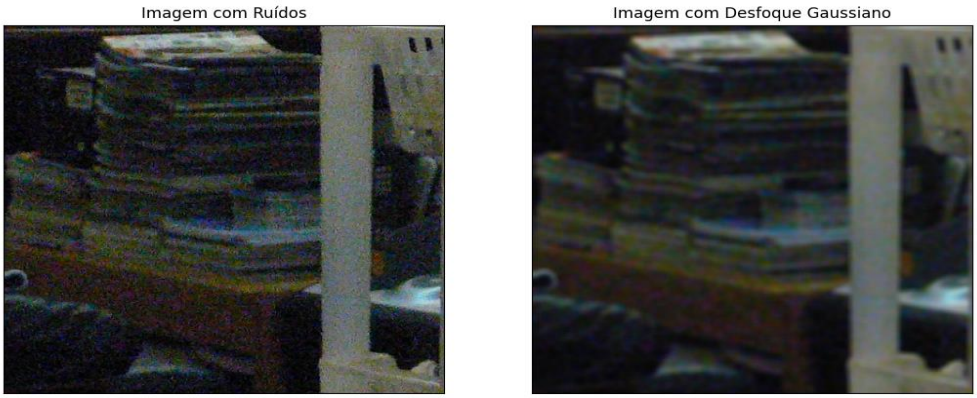
[2] A escala gama (representada pela letra grega γ) é uma função formada pela relação entre luminância e amplitude de sinal (POYNTON, 2003). Stone (2003) explica que a luminância é a intensidade de luz emitida que é perceptível ao olho humano. Para Máximo e Alvarenga (1997), a amplitude é um aspecto físico ondulatório, que é definido pela "distância entre a posição de equilíbrio e a posição extrema ocupada por um corpo que oscila". Trazendo esse conceito a geração de imagens, o corpo oscilante seria a luz formada em um monitor ou TV.

nho. Contudo, HOG se destaca por tornar o pré-processamento não obrigatório. Conforme Dalal e Triggs mencionam em seu artigo, foram utilizadas várias técnicas de normalização de cor: escalas de cinza, correções gama, uso de RGB e LAB3. No entanto, não houve melhorias significativas na implantação do algoritmo.

8.3.2 Calcular os gradientes

Para obter os gradientes, é necessário aplicar um filtro sobre a imagem. Dalal e Triggs utilizam o desfoque gaussiano com o objetivo de diminuir os ruídos, facilitando o cálculo dos gradientes. A Figura 8.8 é um exemplo de ruídos encontrados em uma foto, e ocorrem quando os pixels não se parecem com os seus vizinhos (Forsyth e Ponce, p. 142). Desconsiderar esse problema pode tornar a identificação dos gradientes menos eficaz, reduzindo as chances de extrair as bordas de um objeto.

Figura 8.8: A figura da esquerda possui ruídos que afetam a nitidez das cores. Ao aplicar o desfoque gaussiano (direita), a imagem recebe uma diminuição dos *pixels* com ruídos, tornando as cores mais uniformes.



Fonte: Wikipedia

Portanto, desfocar a imagem reduz essas discrepâncias encontradas nos pixels. O desfoque consiste em aplicar pesos em cada pixel, dentro de uma amostra. Para os pixels centrais são atribuídos pesos maiores que os localizados em bordas. Em seguida, é calculada a média ponderada dos pontos da imagem. Por fim, é construída uma matriz com os pesos, que chamamos de kernel ou máscara.

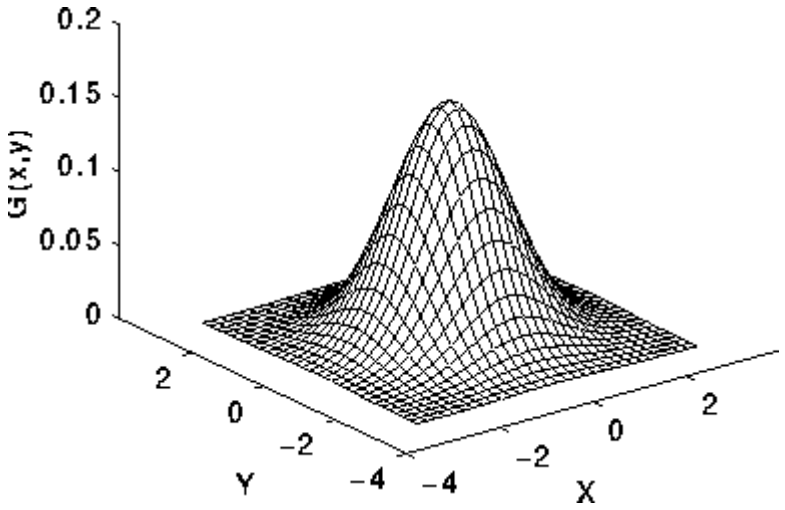
Essa matriz precisa ser preenchida com uma função gaussiana, que é dada por:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2a^2}}$$

onde σ é o desvio padrão da distribuição (Dalal e Trigs observaram que HOG tem melhor desempenho com $\sigma = 0$). Cada peso do kernel será convertido através desta função. A distribuição dos valores é representada pelo gráfico mostrado na Figura 8.9.

[3] Também chamada de CIELAB ou L*a*b*, é um espaço de cores formado por três elementos: o brilho (L) e os pare de cores vermelho-verde (A) e azul-amarelo (B) (JAIN, 1989).

Figura 8.9: Representação gráfica dos valores obtidos pela função gaussiana. Por ser uma função com duas variáveis independentes, o gráfico apresenta três dimensões.



Fonte: Fisher *et al*

Em HOG, o desfoque é aplicado nos sentidos horizontal e vertical da imagem, através dos kernels $[-1 \quad 0 \quad 1]$ para o eixo X e $[-1 \quad 0 \quad 1]^T$ para Y. Foram testadas outras máscaras, como as matrizes 3 X 3, mas o desempenho não foi muito satisfatório. Após aplicar o filtro gaussiano, é possível obter a magnitude e a orientação do gradiente, conforme visto na seção anterior.

8.3.3 Montar o histograma

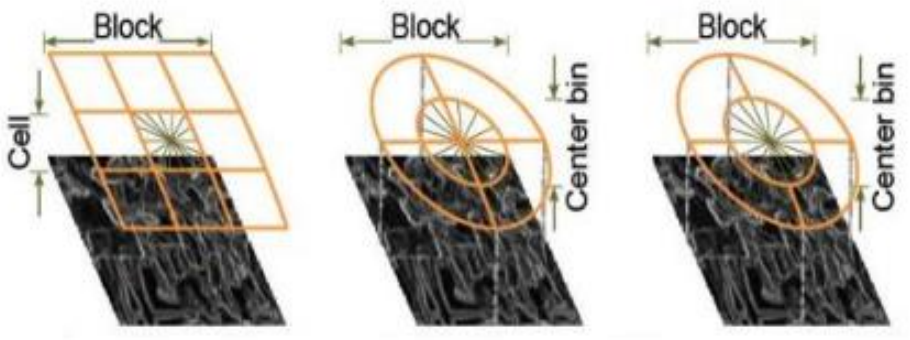
Com os gradientes calculados, o próximo passo é dividir a imagem em células de 192 pixels, ou seja, 8 pixels de largura e 8 pixels de altura. Sabendo que uma imagem colorida possui matriz tridimensional, então o total de pontos que teremos será 8 x 8 x 3 = 192. Em cada divisão é calculada a distribuição dos gradientes, também chamada de histograma. Este gráfico irá exibir a orientação predominante nos gradientes obtidos na célula, através de 9 caixas (ou binários - bins) que representam o ângulo de orientação. Como Dalal e Triggs mencionam, o descritor de recursos teve um melhor desempenho considerando orientações que estão numa faixa de 0º a 180º. Os valores que estão na escala 0º - 180º são chamados de ângulos não assinados, pois o gradiente e o seu oposto irão possuir o mesmo valor, ignorando o sinal. Dessa forma, um gradiente com 180º de orientação será igual ao seu oposto, se assemelhando com valores modulares. Quanto a faixa de ângulos assinados, os valores acima de 180º são levados em consideração, permitindo haver números negativos.

8.4.4 Normalizar os blocos

Após a construção dos histogramas, será necessário aplicar alguma técnica que diminua a interferência da iluminação ao obter o vetor de recursos. De fato, os gradientes são bastante afetados pela iluminação. Para resolver isso, os autores de HOG elaboraram um método de dividir a imagem em blocos e normalizá-los separadamente. Um bloco é um conjunto de células, que podem ser de diferentes formas geométricas. Os autores de HOG classificam esses agrupamentos em duas classes:

- R-HOG (Rectangular-HOG) - são blocos formados por três parâmetros: células por cada bloco; pixels por cada célula; e canais por cada histograma. Esses blocos apresentam formato retangular ou quadrático.
- C-HOG (Circular-HOG) - apresentam 4 parâmetros: o número de binários angulares e radiais; o raio do binário central em pixels; e o fator de expansão para os raios subsequentes.

Figura 8.10: Os tipos de blocos apresentados por Dalal e Triggs. O da esquerda é R-HOG, dividido em células 3 x 3. As duas últimas figuras são blocos circulares (C-HOG) e apresentam características que diferem entre si. A imagem central mostra um bloco circular cujas células estão divididas em setores. O bloco da direita não possui segmentação no centro da figura.



Fonte: Chandel e Vatta (2015), p. 8

Para normalizar os blocos, foram testados quatro métodos. Considere v o array de recursos desnormalizado, $\|v\|_k$é o k índice do módulo do vetor, para $k = 1,2$. Por fim, ϵ uma pequena constante. As regras aplicadas são:

- **L1-norm -** $\frac{v}{\|v\|_1+\epsilon^2}$
- **L2-norm -** $\frac{v}{\sqrt{\|v\|_2^2+\epsilon^2}}$
- **L2-*hys*** - L2-norm seguido por *clipping* (limitar os valores de v para 0.2). Em seguida, renormalize.
- **L1-sqrt -** L1-norm seguido pela raiz quadrada $\sqrt{\frac{v}{\|v\|_1+\epsilon^2}}$

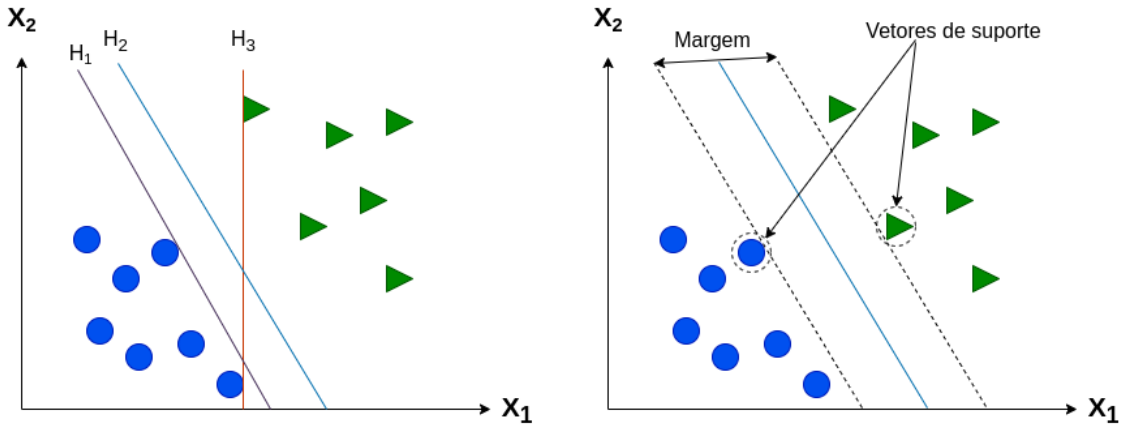
Dalal e Triggs testaram o desempenho dessas quatro técnicas, e concluíram que L2-norm obteve uma melhor perfomance.

8.3.5 Extrair o vetor de recursos e classificar a imagem

Esta etapa é a saída do algoritmo. Após normalizar os blocos, os dados obtidos em cada histograma são combinados em um vetor, cujo tamanho irá variar de acordo com o tamanho da imagem. Por fim, o vetor de características alimentará um algoritmo classificador, que será responsável por verificar se em uma imagem há a presença de uma face.

Dalal e Triggs utilizaram o SVM como algoritmo classificador. O *support vector machines* é um algoritmo de *Machine Learning*, cujo objetivo é encontrar, dentro de um conjunto de dados, a maior margem de classificação (RASCHKA, 2016). Define-se como margem a distância entre os agrupamentos de dados, que são construídos com base em padrões obtidos nos dados (RASCHKA, 2016). Ao encontrar essa região, o SVM é capaz de classificar se um valor pertence a um grupo X ou a um Y. A Figura 8.11 demonstra o funcionamento do algoritmo.

Figura 8.11: Demonstração do SVM. Dado dois grupos de dados, o algoritmo inicialmente aplica todos os possíveis hiperplanos (linhas de separação) que os separam. Em seguida, é escolhido o hiperplano que possui a maior distância entre as coleções. Por fim, os pontos mais próximos do hiperplano são chamados de vetores de suporte, ou support vectors.



Fonte: Adaptado de Raschka (2016), p. 69

8.4 Implementando o algoritmo HOG

Através das bibliotecas Python de processamento de imagens e Machine Learning, implementar o algoritmo HOG é relativamente simples. Abaixo são listados os recursos para desenvolver o código:

- OpenCV - trata-se de uma biblioteca utilizada em processamento de imagens. Com ela, é possível realizar atividades simples de edição de imagens, como converter uma escala de cor, e até mesmo operações mais complexas de identificação de bordas e cálculo de gradientes.
- Dlib - utilizada para desenvolver algoritmos de alta complexidade, incluindo modelos de Machine Learning. Através dessa biblioteca, será possível implementar o descritor de recursos HOG.
- Imutils - biblioteca utilizada em processamento de imagens, apresentando recursos parecidos com o Open CV.
- Matplotlib - módulo Python para visualização de dados. Através dele, é possível criar diversos gráficos, como: histogramas, mapas de calor, gráficos de pizza etc.
- Numpy - uma biblioteca de computação matemática, que oferece suporte a cálculos com matrizes multidimensionais. Também permite realizar operações de Álgebra Linear.

Como será mostrado, o código para desenvolver o algoritmo HOG será de baixa complexidade, uma vez que todas as etapas do descritor mostradas anteriormente são processadas de forma implícita, através de um único método (inclusive o algoritmo classificador). O primeiro passo é importar as bibliotecas necessárias, conforme a imagem abaixo.

Código 8.1: Bibliotecas utilizadas no algoritmo.

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
import dlib
from imutils import face_utils
```

Na primeira linha, é importado o OpenCV, enquanto nas demais são importadas as bibliotecas Matplotlib, Numpy e Dlib. Na última linha, foi chamada a função “face_utils” da biblioteca Imutils. Através dessas bibliotecas, será possível detectar faces através de HOG tanto em imagens quanto em vídeos. As próximas seções irão mostrar a implementação do descritor de recursos nesses dois casos

8.4.1 Implementando HOG em imagens

Código 8.2: Inserindo e tratando a imagem.

```
imagem = cv.imread('pessoas.jpg')
imagem_normalizada = np.float32(imagem) / 255.0
```

Com as bibliotecas prontas para serem utilizadas, o próximo passo é definir o endereço da imagem que será utilizada no algoritmo. Para isso, foi atribuída a variável “imagem” o endereço do arquivo, através do método “imread()” que pertence ao OpenCV. Posteriormente, a imagem que está nessa variável receberá edições geradas pelo descritor de recursos, sendo preciso adicionar outra variável que guardará o arquivo original. A “imagem_normalizada” apresenta essa função, recebendo os pixels da foto convertidos em números decimais. Cada valor é dividido por 255 (a cor mais “alta” do padrão RGB, o branco), mantendo as cores originais da imagem.

Figura 8.12: Imagem de entrada do algoritmo. Perceba que não foi necessário converter o espaço de cores para preto e branco, uma vez que a etapa de pré-processamento é opcional em HOG.

Fonte: vishalbpatil1 (2021)

Ao definir qual imagem será utilizada, é necessário calcular o gradiente da imagem, conforme o Código 8.3.

Código 8.3: Cálculo do gradiente.

```
# Calculando o gradiente
gx = cv.Sobel(imagem_normalizada, cv.CV_32F, 1, 0, ksize=1)
gy = cv.Sobel(imagem_normalizada, cv.CV_32F, 0, 1, ksize=1)
magnitude, angulo = cv.cartToPolar(gx, gy)

# Exibindo os gradientes da imagem
fig, gradiente = plt.subplots()
gradiente.axis('off')
gradiente.imshow(cv.cvtColor(magnitude, cv.COLOR_BGR2RGB))
gradiente.set_title('Extração dos gradientes')
plt.show()
```

As variáveis “gx” e “gy” recebem os gradientes horizontal e vertical, respectivamente. O método “Sobel()” do OpenCV recebe os pixels de “imagem_normalizada”. Em seguida, o parâmetro “cv.CV_32F” declara que os pixels são números decimais. Os próximos dois argumentos numéricos correspondem as derivadas dos eixos X e Y. Perceba que “gx” receberá a derivada horizontal (gradiente X), enquanto “gy” apresenta a derivada vertical (gradiente Y). A palavra-chave “ksize” define o tamanho do kernel. No exemplo acima, o valor informado é 1, correspondendo as matrizes para o eixo X e $[-1| \quad |0| \quad |1]^T$ para o eixo Y. Por fim, as variáveis “magnitude” e “angulo” recebem a magnitude e o ângulo do gradiente, por meio do método “cartToPolar()”. Os seus argumentos são os vetores dos eixos X e Y, representados pelas variáveis “gx” e “gy”, respectivamente.

É possível visualizar os gradientes extraídos. A variável “gradiente” irá receber a imagem com os gradientes. O método “axis()” permite configurar a visibilidade dos eixos. Já “imshow()” exibe a figura, sendo que “cvtColor()” habilita o espaço de cor para RGB. Por fim, o método “set_title()” altera o título da imagem. Conforme a Figura 8.13, uma janela se abre através do comando “plt.show()”.

Figura 8.13: Imagem com os gradientes extraídos.

A etapa que sucede o cálculo dos gradientes é a aplicação do descritor de recursos, junto com um algoritmo classificador. É criada a variável “detectar_face”, que receberá o método “get_frontal_face_detector()”, responsável por aplicar o descritor HOG. Em seguida, essa variável passará a ser uma função, que receberá uma imagem com o objetivo de identificar a existência de faces. O segundo argumento dessa função indica o número de vezes que a amostra será analisada. Quanto maior o número, maior será a acurácia dos resultados. Contudo, o tempo de processamento também será maior, afetando o desempenho do algoritmo. Logo, o valor 1 para a função é a mais recomendada.

Código 8.4: Aplicação do descritor de recursos HOG.

```
# Extraindo o descritor de recursos com HOG
detectar_face = dlib.get_frontal_face_detector()
retangulos = detectar_face(imagem, 1)
for (i, retangulo) in enumerate(retangulos):
    (x, y, w, h) = face_utils.rect_to_bb(retangulo)

    # Identificando a face na imagem
    cv.rectangle(imagem, (x, y), (x + w, y + h), (0, 255, 0), 3)

    # Imprimindo os vértices de cada retângulo
    print(retangulo)
```

Com as faces encontradas, será necessário desenhar os retângulos sobre elas. Para isso, foi preciso criar uma estrutura iterativa que percorre cada índice da lista "retangulos". A tupla "(x, y, w, h)" irá receber, em cada elemento, os pontos que formam os vértices do retãngulo. Por fim, é desenhada a forma, através do método "rectangle" do OpenCV.

A saída da função "get_frontal_face_detector()" é uma lista contendo todas as faces encontradas na imagem. Em outras palavras, o resultado obtido possui todos os vértices que formam retângulos onde estão localizados os rostos da imagem (ver Figura 8.14). Esses dados serão armazenados na variável "retangulos".

Figura 8.14: Listas dos pontos que formam os vértices do retângulo onde está inserido os rostos.

```
[(304, 22) (347, 65)]
[(391, 26) (434, 70)]
[(567, 78) (619, 130)]
[(303, 271) (378, 345)]
[(356, 122) (418, 184)]
[(545, 283) (653, 390)]
[(220, 105) (295, 179)]
[(436, 246) (511, 320)]
[(458, 61) (510, 113)]
[(467, 135) (529, 198)]
[(-22, 150) (150, 305)]
[(193, 49) (245, 101)]
```

A última parte do algoritmo é a exibição da imagem com as faces identificadas. Como mostrado no Código 8.5, os comandos utilizados são os mesmos adotados na exibição dos gradientes.

Código 8.5: Código que exibe as imagens de entrada, os gradientes e a saída do descritor.

```
# Exibindo a imagem com as faces detectadas
fig, resultado = plt.subplots()
resultado.axis('off')
resultado.imshow(cv.cvtColor(imagem, cv.COLOR_BGR2RGB))
resultado.set_title('Face detectada, através de HOG')
plt.show()
```

A saída da imagem é apresentada na Figura 8.15.

Figura 8.15: Imagem com as faces detectadas, por meio das caixas delimitadoras.

8.4.2 Implementando HOG em vídeos

A lógica para identificar faces com HOG em vídeos é parecida com a utilizada em imagens. Isso porque um vídeo é um conjunto de fotos em sequência, sendo, portanto, necessário que o descritor seja aplicado em cada imagem para detectar a presença das faces.

Código 8.6: Aplicação do descritor de recursos HOG em um vídeo.

```
video = cv.VideoCapture("pessoas.mp4")

while video.isOpened():
    ret, quadro = video.read()
    imagem = cv.cvtColor(quadro, cv.COLOR_BGR2RGB)
    detectar_face = dlib.get_frontal_face_detector()
```

```
    retangulos = detectar_face(imagem, 1)

    for (i, retangulo) in enumerate(retangulos):
        (x, y, w, h) = face_utils.rect_to_bb(retangulo)
        cv.rectangle(quadro, (x, y),
                     (x + w, y + h),
                     (0, 255, 0), 2)

        cv.imshow('Video', quadro)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break
```

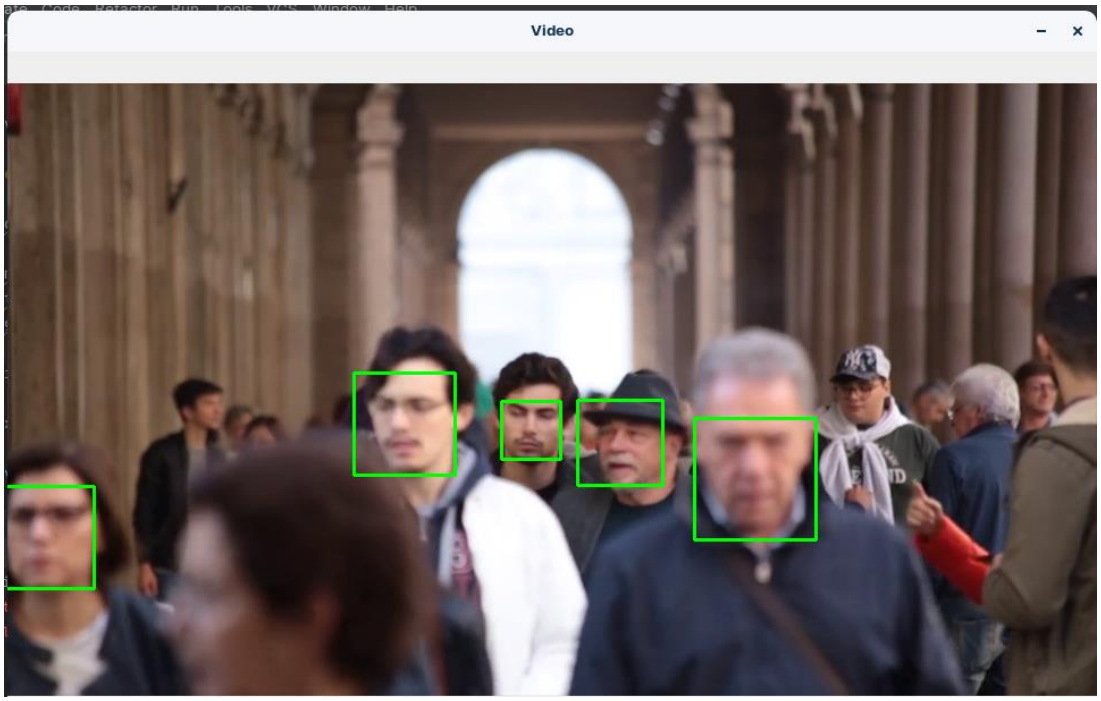
Com as bibliotecas importadas, o primeiro passo é indicar o diretório onde o vídeo está salvo. Isso é realizado através do método “videoCapture()” do OpenCV. No código acima, é criada uma variável que recebe a saída deste método. Em seguida, é inserida uma estrutura de repetição que será utilizada para executar o vídeo. Enquanto a gravação estiver aberta (“video.isOpened()”), ocorrerá a manipulação do vídeo para a detecção das faces. Cada imagem do vídeo será atribuída às variáveis “ret” e “quadro”. Em “imagem”, o quadro obtido será recebido com escala de cor RGB.

Assim como foi aplicado nas imagens, o método “get_frontal_face_detector()” será utilizado para trabalhar com HOG nos vídeos. A variável “detectar_faces” recebe este método do Dlib, sendo usada para identificar as faces. Em “retangulos”, ela passa a ser uma função que recebe dois argumentos: o primeiro é a imagem de entrada e o segundo o número de vezes que a amostra será analisada. A saída será a lista de pontos que formam os vértices dos retângulos onde estão as faces encontradas.

Com as faces detectadas, cria-se uma estrutura de repetição que percorre cada índice da lista salva em “retangulos” para exibir as formas no vídeo. A tupla “(x, y, w, h)” recebe os vértices do retângulo, que será desenhado através do método “rectangle”. Por fim, o quadro (também chamado de frame) do vídeo será exibido. Dessa forma, todas as imagens que compõem o vídeo serão editadas para que se mostre as faces presentes em cada instante da gravação.

Para que o algoritmo finalize o seu processamento, o usuário deverá pressionar a tecla “q”, conforme a estrutura condicional mostrada no código acima. Por fim, a saída gerada pelo algoritmo é mostrada abaixo.

Figura 8.16: Trecho de um vídeo com a detecção facial através de HOG.



8.4.3 Avaliando o desempenho de HOG

Após mostrar a implementação de HOG em imagens e vídeos, agora será explicado como avaliar o seu desempenho, através da acurácia e do tempo de processamento. Para isso, serão utilizadas três bases de imagens: Yalefaces, Labeled Faces in the Wild (LFW) e Dogs VS Cats. As duas primeiras bases possuem imagens de pessoas, enquanto a última contém fotos de cães e gatos. Desses três bancos, retiramos uma amostra de 1000 fotos, que foram colocadas em um único diretório. Dessas 1000 imagens, 600 são de pessoas. Nesta seção, serão abordados a função HOG e os resultados obtidos.

Conforme exibido no Código 8.7, foi criada uma função que recebe como parâmetro o endereço da imagem. Esse endereço é atribuído a variável “imagem”, por meio do método “cv.imread()” da biblioteca OpenCV. Em seguida, o método “get_frontal_face_detector()” da biblioteca Dlib é utilizado na variável “detectar_face” para aplicar o descritor HOG na imagem. Em seguida, essa variável passará a ser uma função em “retangulos”, que irá receber uma lista de coordenadas para a construção das caixas delimitadoras.

Por fim, é verificado se existe uma face na imagem. Se o tamanho da lista “retangulos” for 0, então a saída da função é “False” (não existe faces na foto). Caso contrário, retornará “True”.

Código 8.7: Função para reconhecer uma face com HOG.

```
def hog(endereco_imagem):
    imagem = cv.imread(endereco_imagem)

    # Extraindo o descritor de recursos com HOG
    detectar_face = dlib.get_frontal_face_detector()
    retangulos = detectar_face(imagem, 1)

    if len(retangulos) == 0:
        return False
    else:
        return True
```

Essa função irá percorrer cada imagem, retornando uma saída booleana (True para face encontrada e False para não encontrada). Os resultados obtidos são mostrados abaixo. A Figura 8.17 mostra a matriz de confusão, cujo objetivo é comparar o número de resultados preditos pelo algoritmo com a quantidade real de dados por categoria. A matriz $M_{linha,coluna}$ divide os dados em quatro categorias: falsos positivos ($M_{0,1}$), verdadeiros positivos ($M_{1,1}$), falsos negativos ($M_{1,0}$) e verdadeiros negativos ($M_{0,0}$) (SCIKITLEARN, 2022).

Além disso, com a matriz de confusão, é possível calcular a acurácia do algoritmo. Mishra (2018) explica que a acurácia de um modelo é dada pela razão entre a quantidade de acertos do algoritmo e o total de predições realizadas. Ou seja:

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN}$$

Onde:

- **VP –** verdeiros positivos;
- **VN –** verdadeiros negativos;
- **FP –** falsos positivos;
- **FN –** falsos negativos

Com base na matriz de confusão, HOG terá uma acurácia de 95%, aproximadamente, pois:

$$Acurácia = \frac{559 + 390}{559 + 390 + 41 + 10} = 0{,}949$$

A Figura 8.18 mostra a curva Característica de Operação do Receptor (ou ROC – Receiver Operating Characteristic curve). Segundo Avelar (2019): “ROC é uma curva de probabilidade. Ela é criada traçando a taxa verdadeiro positivo contra a taxa de falsos-

positivos. Ou seja, número de vezes que o classificador acertou a predição contra o número de vezes que o classificador errou a predição". Geralmente, o eixo X da curva é a taxa de falsos positivos, que é dada por falsos positivos/(falsos positivos + verdadeiros negativos), enquanto o eixo Y é gerado pela taxa de verdadeiros positivos, cujo cálculo é verdadeiros positivos/(verdadeiros positivos + falsos negativos) (AVELAR, 2019).

Por fim, a Figura 8.19 mostra o tempo de detecção do algoritmo. O gráfico mostra o tempo em função da quantidade de imagens. É possível concluir que em boa parte das fotos, o algoritmo ficou no intervalo de 0.4 a 0.7 segundos para processá-las.

Figura 8.17: Matriz de confusão de HOG. Este gráfico mostra o número de acertos e erros do algoritmo. Note que HOG teve uma maior quantidade de acertos em identificar as imagens com e sem faces.

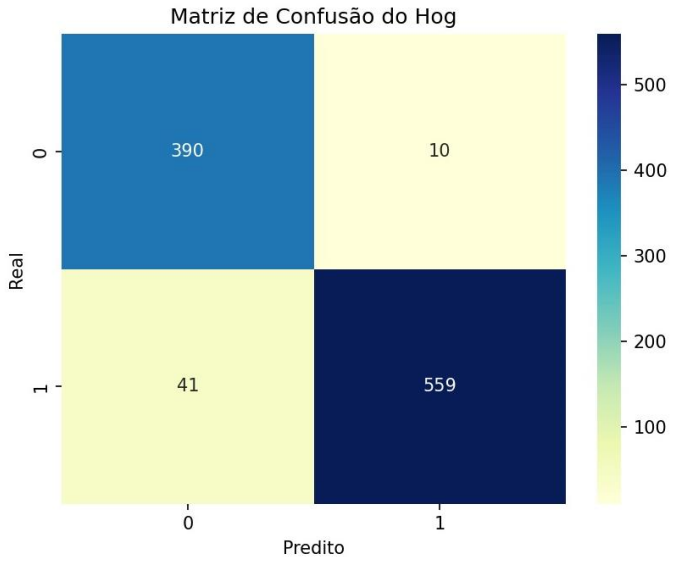


Figura 8.18: Curva ROC mostrando a taxa de verdadeiros positivos em relação a taxa de falsos positivos.

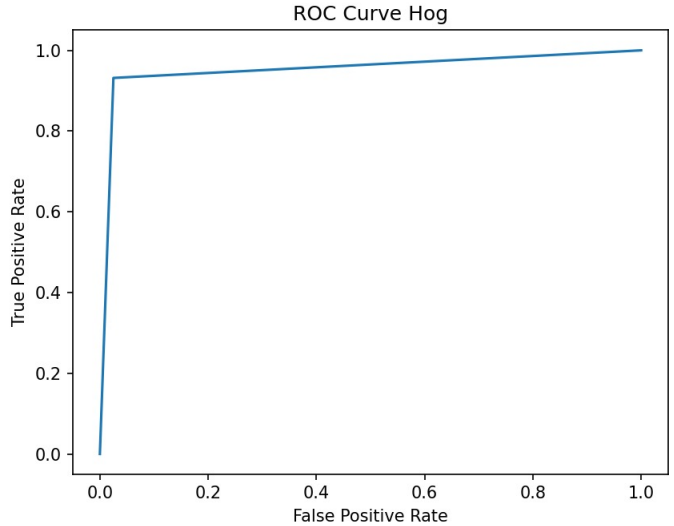
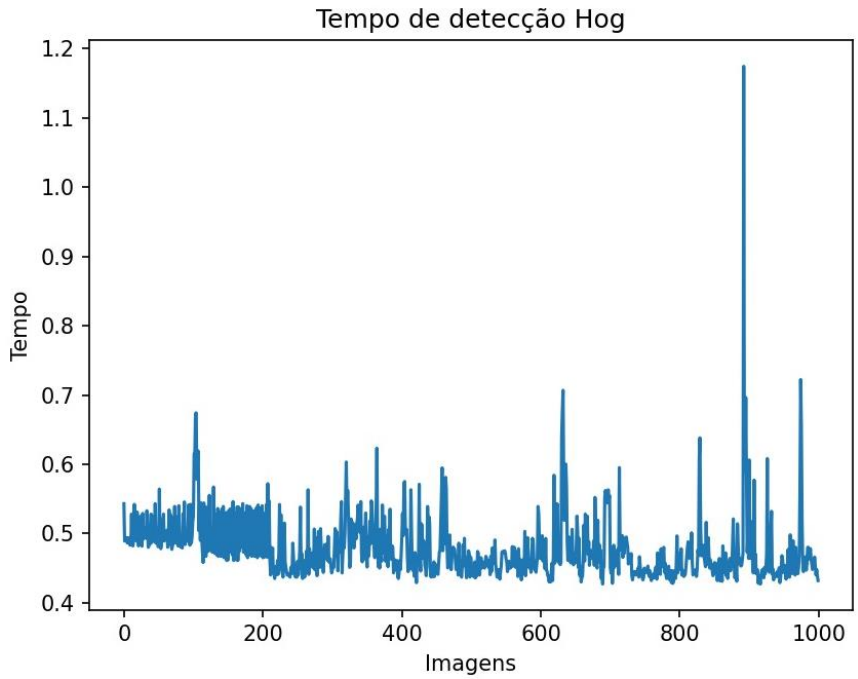


Figura 8.19: Tempo de detecção do HOG. O algoritmo levou de 0,4 a 0,7 segundos para processar a maioria das imagens.



Com base na acurácia e no tempo de processamento obtidos, percebe-se que HOG é um descritor de recursos eficiente ao detectar faces, com baixos números de falsos positivos e negativos. Ao mesmo tempo que consegue atingir esse objetivo, o algoritmo é rápido ao processar as imagens, tornando viável o seu uso em aplicações onde a baixa latência é um requisito importante.

A partir do próximo capítulo, serão abordados algoritmos de rede neural utilizados na detecção de faces, começando com o CNN.

9

CNN – Redes Neurais Convolucionais

A Visão Computacional vem ganhando destaque pela sua premissa de capacitar máquinas a verem o mundo como humanos fazem, sendo capazes de entender de maneira similar e utilizar o conhecimento adquirido para diversas tarefas, como Reconhecimento de Imagem e Vídeo, Sistemas de Recomendação, Processamento de Linguagem Natural etc. Um dos algoritmos que contribui de maneira significativa para o avanço desta área é a Rede Neural Convolucional, introduzida em 1998 por Yann Lecun, pesquisador com pós-doutorado em Ciência da Computação e trabalhos nas áreas de aprendizado de máquina, visão computacional, robótica móvel e neurociência computacional.

9.1 Neurônio Biológico

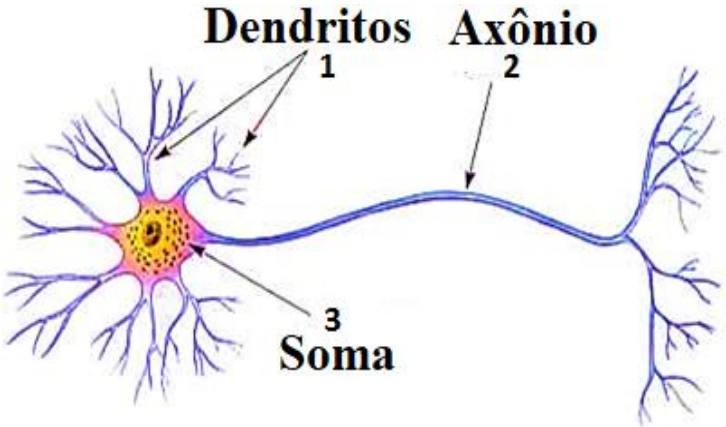
O neurônio é a célula do sistema nervoso responsável pela condução do impulso nervoso. É composto por 3 partes: os dendritos, a soma e o axônio. Há cerca de 86 bilhões de neurônios formando a rede neural biológica no sistema nervoso humano.

Os dendritos têm como função receber estímulos do ambiente ou de outros neurônios e transmiti-los para a soma.

A soma processa os impulsos nervosos recebidos pelos dendritos. Dependendo da relevância do impulso, a soma pode ficar excitada. Se o nível de excitação exceder certo limite, a excitação será transferida adiante. Cada neurônio possui sua própria configuração, portanto, o grau de excitação é variável.

Os axônios têm como função transmitir os novos impulsos nervosos aos dendritos de outros neurônios.

Figura 9.1 - Neurônio biológico



Fonte: (adaptado) disponível em https://1.bp.blogspot.com/--f9Cpwd3hDU/VEgUpcQySVI/AAAAAAAABSU/VKP4ZmvtMqw/s1600/Sem%2Bt%C3%ADtulo.pngaaartu.png.

9.2 Neurônio Artificial

Embora existam diversos modelos de neurônios artificiais, a alternativa mais aceita foi proposta por Warren McCulloch e Walter Pitts em 1943. Os autores propuseram um neurônio matemático onde este é um componente de uma rede neural artificial que calcula a soma ponderada de várias entradas, aplica uma função e propaga os novos dados adiante.

O neurônio matemático, diferentemente do neurônio biológico, possui um caráter binário. Logo, recebe um ou mais sinais de entrada e retorna apenas um sinal de saída que, por sua vez, pode também ser utilizado por outros neurônios (formando uma rede neural).

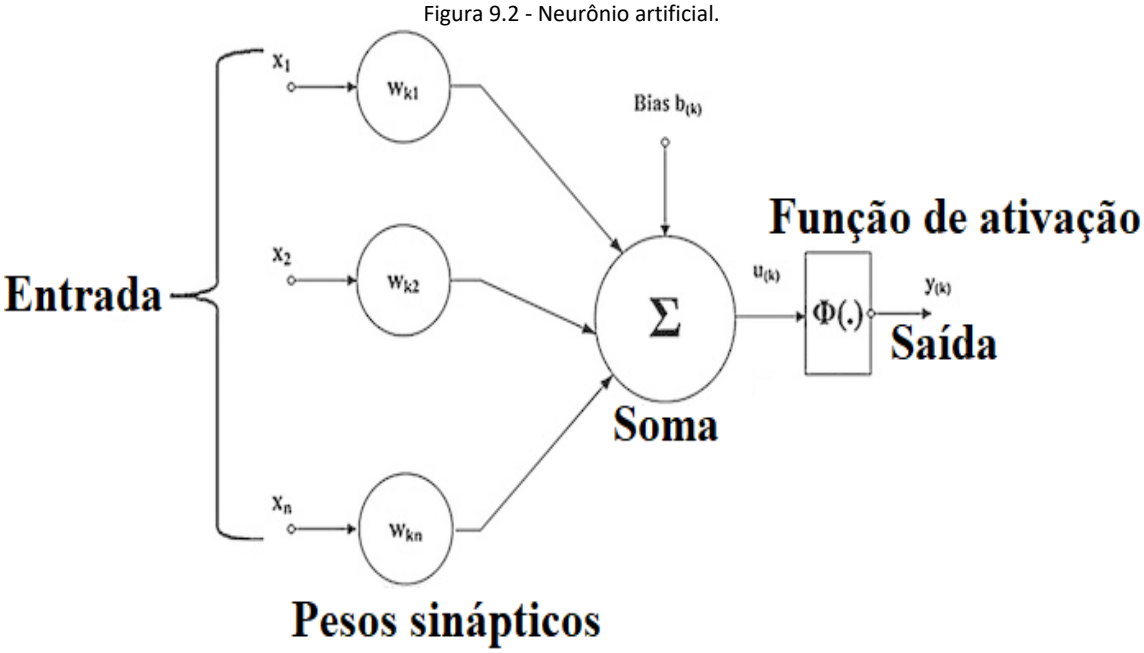
Neste modelo de neurônio matemático, os impulsos elétricos presentes em uma rede neural biológica são representados por sinais de entrada — dados que alimentam o modelo da rede neural artificial — e correspondem às letras "x" na Figura 9.2.

Naturalmente, o neurônio receptor terá seu nível de excitação alterado conforme o estímulo recebido. Este nível é representado através de pesos sinápticos, onde quanto maior seu valor, mais excitatório é o estímulo. Na Figura 9.2, os pesos sinápticos são representados por "w_{kn}", onde "w" representa o próprio peso; "k" representa o índice do neurônio e "n" representa a entrada a qual o peso sináptico se refere.

A soma é representada pela composição de dois módulos: junção aditiva e função de ativação. Onde o primeiro é responsável pela soma dos sinais de entrada multiplicados pelos pesos sinápticos e o segundo é responsável por definir a saída do neurônio.

O axônio é representado por "$y_{(k)}$" e possui o valor calculado pela função de ativação, podendo ser excitatório ou inibitório.

Figura 9.2 - Neurônio artificial.



Fonte: (adaptado) disponível em https://www.gsigma.ufsc.br/~popov/aulas/rna/neuronio_artificial/index.html.

9.3 Rede Neural Biológica

A troca de informação entre neurônios ocorre através de terminais axônicos conectados via sinapses a dendritos de outros neurônios. Se um neurônio possuir uma determinada informação e a somatória dos sinais de entrada ultrapassar certo nível, o neurônio original gera um potencial de ação e propaga a informação para outros neurônios via neurotransmissores químicos.

9.4 Rede Neural Artificial

Uma rede neural artificial (RNA) é um modelo computacional inspirado no funcionamento do cérebro humano capaz de realizar o aprendizado de máquina e o reconhecimento de padrões.

É possível alimentar a rede com diversos tipos de dados, variando de imagens a áudios, por exemplo. Na prática, as informações de entrada serão fragmentadas em diferentes neurônios. Portanto, caso a rede seja alimentada com uma imagem de dimensões 10 pixels de largura por 10 pixels de altura, estarão presentes na primeira camada 100 neurônios, cada um correspondendo a informação contida num único pixel.

Os neurônios irão verificar se determinadas características estão presentes na imagem em ordem de complexidade utilizando um mapa de recursos, que possui dados contendo características presentes em uma determinada categoria de imagens (animais, faces humanas, veículos etc.). Inicialmente, a rede pode verificar se a imagem possui bordas ou traços retos, por exemplo. Em sequência, pode testar padrões mais avançados e abstratos, como laços, quadrados, formas etc. Cada resultado da verificação é armazenado e utilizado no resultado final para decidir se a característica em questão existe na imagem ou não.

9.4.1 Deep Learning

O Aprendizado Profundo (*Deep Learning*, em inglês) é um ramo do aprendizado de máquina que, através de algoritmos e técnicas, cria modelos matemáticos que serão aplicados em camadas de processamento com a finalidade de prever e classificar informações. Para seu desenvolvimento, foram necessários, essencialmente, avanços em 4 áreas distintas: aprendizado de máquina, Big Data, unidades de processamento gráfico e redes neurais artificiais.

O aprendizado de máquina é uma área que permite computadores analisarem dados brutos e, através de algoritmos de extração e modelos matemáticos, organizá-los de modo que produzam uma saída correta e esperada. Este modelo ainda deve ser capaz de produzir resultados esperados em situações onde os dados informados são desconhecidos.

O Big Data, por sua vez, é a área que estuda novas maneiras de processamento de dados para resolução de problemas, sendo proveniente do grande aumento em volume, crescimento e variedade de dados.

Finalizando, a criação e desenvolvimento de GPUs (abreviação de *Graphical Processing Unit* — Unidades de Processamento Gráfico, em português), permitiram a realização de operações matemáticas paralelamente, especialmente operações envolvendo matriz e vetores.

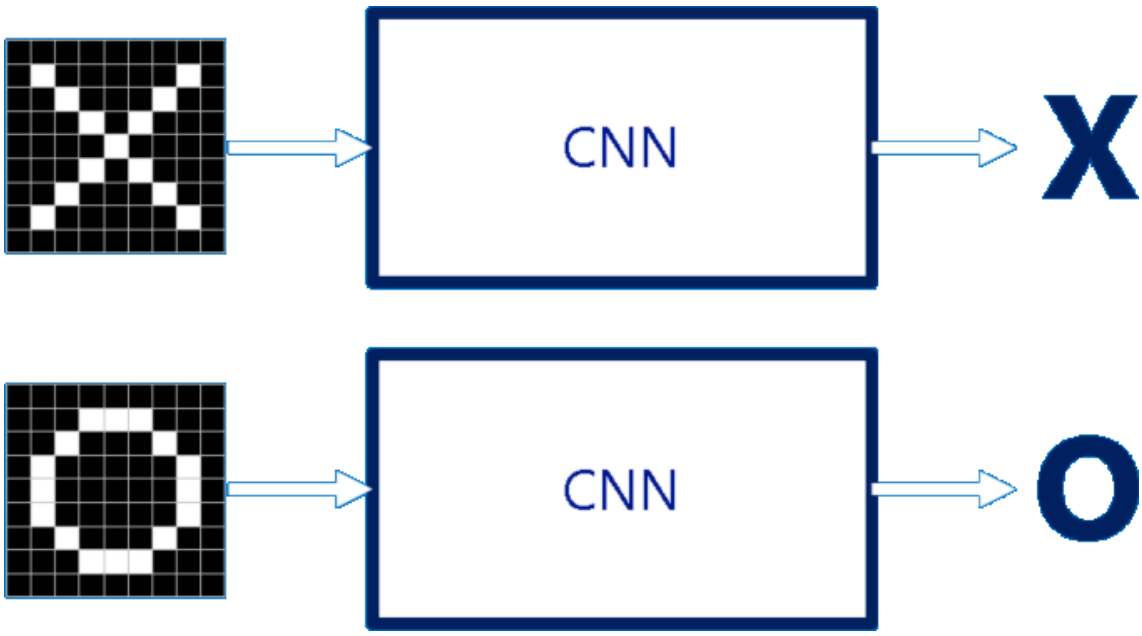
9.5 A Rede Neural Convolucional

Uma CNN ou ConvNet (abreviação de *Convolutional Neural Network* — Rede Neural Convolucional, em português) é um algoritmo de Aprendizado Profundo capaz de captar uma determinada entrada, atribuir importância através de filtros a diferentes aspectos e diferenciar entre seus principais elementos. O pré-processamento necessário em uma ConvNet é menor quando comparado a outros algoritmos de classificação. Em métodos primitivos, filtros de classificação são projetados manualmente, enquanto ConvNets são capazes de criar os próprios filtros.

Embora uma Rede Neural Convolucional possua estrutura adequada para lidar com diversos tipos de aplicações, é geralmente utilizada para tarefas envolvendo o reconhecimento de elementos em imagens e vídeos. Isso ocorre pois a principal vantagem que uma Rede Neural Convolucional oferece quando comparada com Redes Neurais Feedforward (onde a saída de uma camada é usada como entrada para a próxima camada) é sua capacidade de capturar com alta taxa de precisão dependências espaciais e temporais em uma imagem utilizando a aplicação de filtros. Em outras palavras, a arquitetura é capaz de "entender" a sofisticação da imagem.

Para exemplificar, será projetada durante o capítulo uma Rede Neural Convolucional especializada em reconhecer as letras "X" e "O" em imagens, como consta a Figura 9.3.

Figura 9.3 — Aplicação abstrata de uma Rede Neural Convolucional



Fonte: (adaptado) disponível em https://e2eml.school/how_convolutional_neural_networks_work.html.

9.6 Camada Convolucional

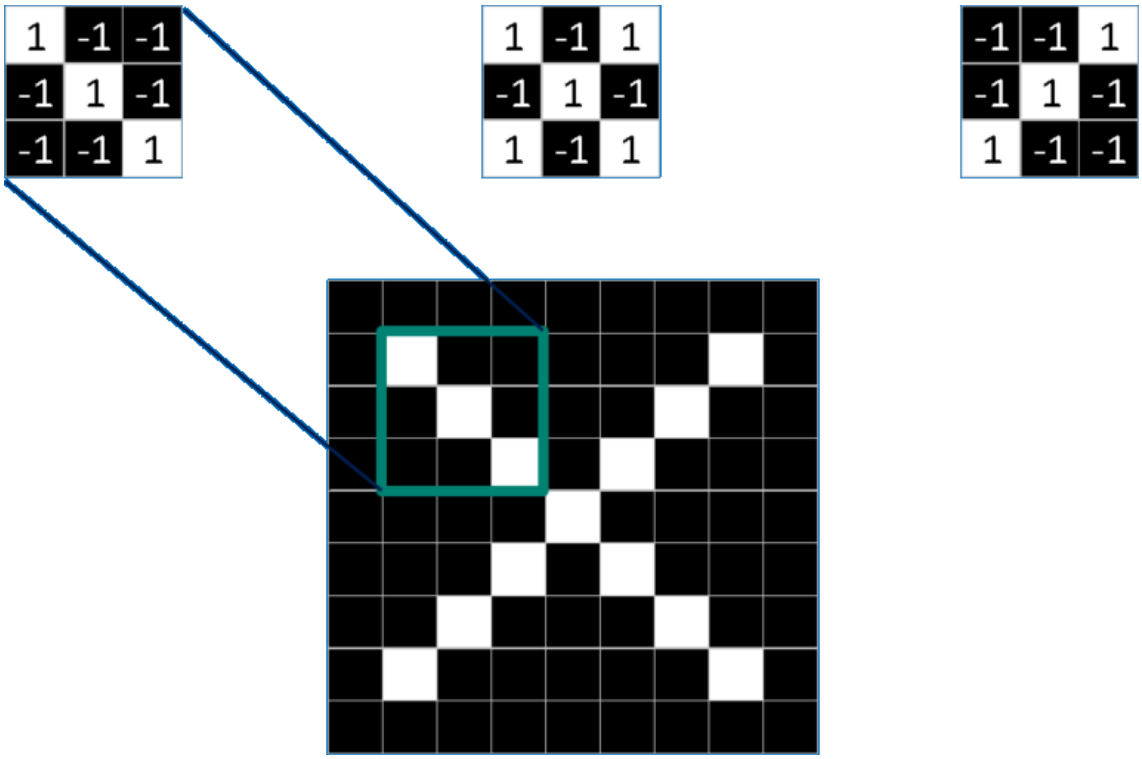
A ConvNet, neste caso, é responsável por diferenciar duas letras independentemente de suas condições iniciais de entrada. Ou seja, deve reconhecer as letras em outras imagens independentemente de posição, cor, tamanho, espessura etc. Nota-se que certos métodos de detecção antigos não seriam adequados para resolver

este problema, já que mínimos detalhes poderiam resultar em uma saída imprecisa e falha.

Uma Rede Neural Convolucional resolve esta situação utilizando recursos (*features*, em inglês). Um recurso é, em sua forma mais básica, um fragmento da imagem. Ao encontrar o mesmo recurso em regiões próximas da imagem, a Rede é capaz de entender a similaridade entre as duas imagens mais facilmente do que comparando imagens inteiras.

Os recursos críticos para reconhecimento da letra “X” em uma imagem, por exemplo, são: a interseção entre as linhas diagonais, a linha diagonal esquerda e a linha diagonal direita, como consta a Figura 9.4.

Figura 9.4 - Recursos críticos



Fonte: (adaptado) disponível em https://e2eml.school/how_convolutional_neural_networks_work.html.

Na imagem, cada pixel claro possui valor igual a 1, enquanto pixels escuros possuem valor igual a -1.

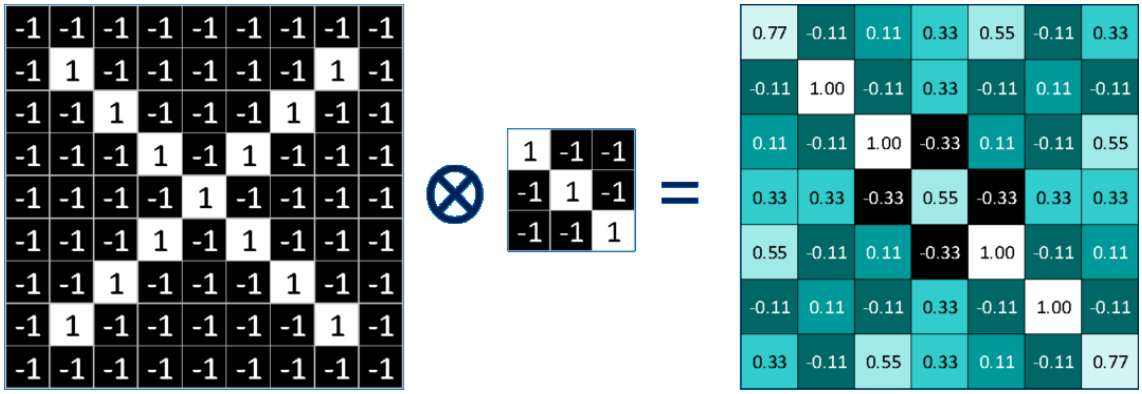
9.6.1 Convolução

A Rede, inicialmente, não sabe onde os recursos devem estar presentes na imagem, portanto são aplicados em todas as posições possíveis. Calculando todas as combinações possíveis do recurso através da imagem, é formado um filtro. A matemática utilizada neste processo é chamada de convolução.

Quando uma imagem de entrada é apresentada, os recursos críticos são devidamente alinhados acima da imagem. Cada valor de pixel da imagem de entrada é multiplicado pelo valor de pixel do recurso, criando uma nova matriz bidimensional. Logo, contanto que os valores dos pixels sejam iguais, o resultado será 1, do contrário, 0. Os valores da matriz são somados entre si. Para finalizar, o valor da soma é dividido

pela quantidade de pixels presentes no recurso utilizado. Se todos os pixels forem iguais entre si, o resultado final será 1. Da mesma forma, se todos os pixels forem diferentes entre si, o resultado final será -1. Este processo pode ser observado na Figura 9.5.

Figura 9.5 - Processo de convolução

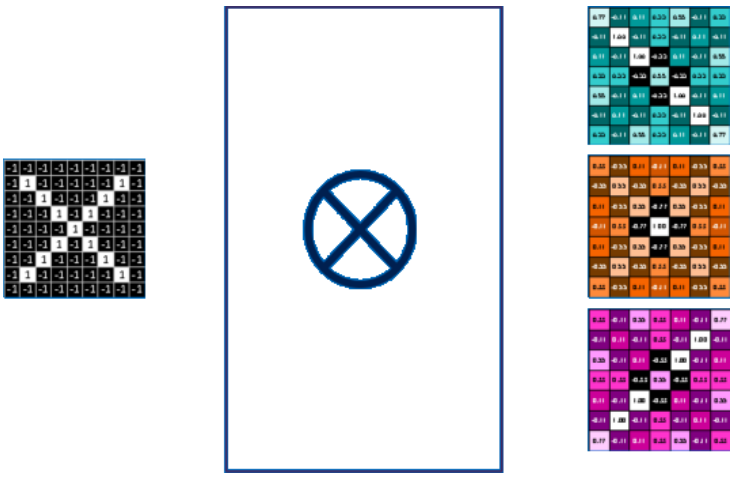


Fonte: (adaptado) disponível em https://e2eml.school/how_convolutional_neural_networks_work.html.

A convolução termina após a aplicação de todos os recursos na imagem de entrada. Neste caso, a Rede possui 3 recursos, como demonstrado na Figura 9.4. A região do filtro que ocupa as dimensões da imagem original é denominada Campo Receptivo Local.

Os valores obtidos após a convolução devem ser armazenados e uma nova matriz bidimensional é formada a partir deles. Na matriz, valores próximos a 1 representam maior semelhança entre a imagem original e o recurso utilizado. Valores próximos a 0 demonstram que, nesta região, há pouca semelhança entre a imagem de entrada e o recurso.

Figura 9.6 - Convolução finalizada

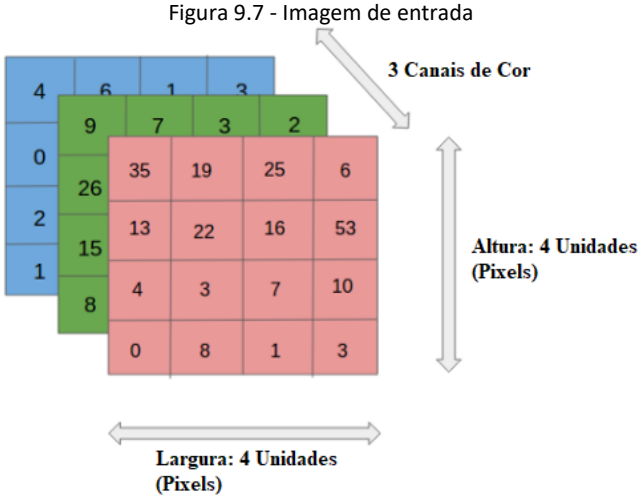


Fonte: (adaptado) disponível em https://e2eml.school/how_convolutional_neural_networks_work.html.

Na primeira imagem após a convolução, os valores mais próximos a 1 correspondem a linha diagonal sentido inferior direito. Na segunda, os valores mais próximos a 1 correspondem a intersecção entre as 2 linhas diagonais. Na terceira, os valores mais próximos a 1 correspondem a linha diagonal sentido superior direito.

Neste caso, a imagem apresentada está em escalas de cinza e, portanto, as cores não são separadas por camadas. Do contrário, ao apresentar uma imagem em RGB (abreviação de *Red, Green and Blue* — Vermelho, Verde e Azul, em português), a mesma será separada a partir de seus planos de cor, como consta a Figura 9.7.

Figura 9.7 - Imagem de entrada

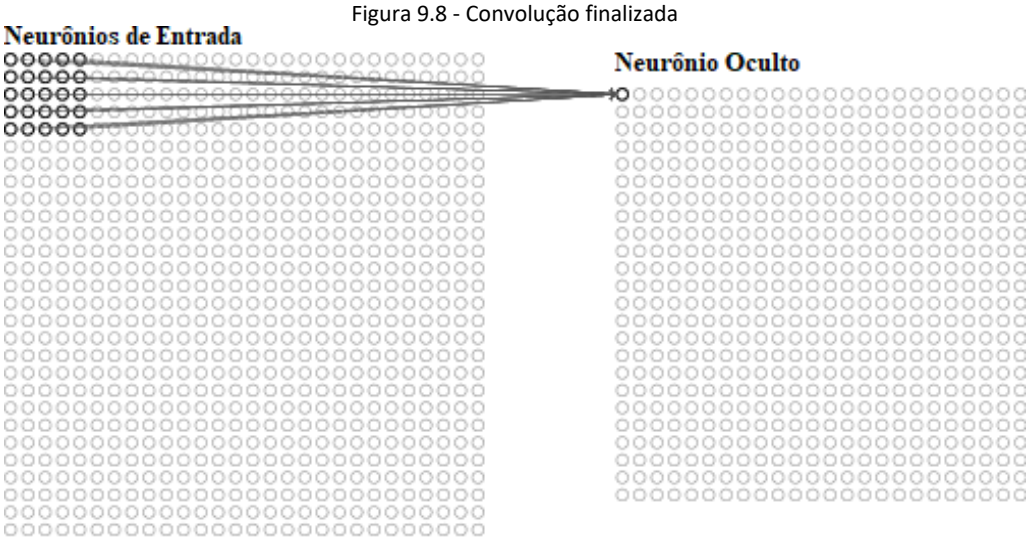


Fonte: (adaptado) disponível em https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

As características presentes na Figura 9.7, como as dimensões da imagem e os valores de pixel são meramente ilustrativas, servindo somente ao propósito de representar a divisão de camadas em uma imagem RGB.

A Camada Convolucional é composta, portanto, de uma série de convoluções obtidas através de comparações entre recursos e imagens de entrada, conforme a Figura 9.8.

Figura 9.8 - Convolução finalizada



Fonte: (adaptado) disponível em https://e2eml.school/how_convolutional_neural_networks_work.html.

A quantidade de neurônios utilizada nesta Figura é meramente ilustrativa, servindo somente ao propósito de representar a formação da Camada Convolucional. São realizadas convoluções nos neurônios de entrada (pixels da imagem de entrada) a partir dos recursos e uma nova matriz bidimensional é criada.

No exemplo utilizado na Figura 9.4, foi utilizada uma imagem com dimensões iguais a 9 pixels de largura e 9 pixels de altura. A quantidade de cálculos necessários aumenta conforme a quantidade de pixels na imagem e a quantidade de pixels em cada recurso. Consequentemente, uma imagem com dimensões maiores necessita de maior tempo e poder computacional. A estrutura utilizada em uma rede neural convolucional é capaz de reduzir a dimensão de uma imagem mantendo seus principais recursos críticos para garantir uma previsão com alta acurácia. Nota-se, portanto, que a arquitetura é escalável, podendo ser utilizada para pequenas aplicações ou trabalhar com conjuntos de dados massivos.

9.7 Recursos

Cada recurso utilizado na Camada Convolucional será aplicado em todas as localizações da imagem. Cada neurônio, portanto, é capaz de reconhecer uma mesma característica da imagem. Ou seja, cada neurônio realiza análises em seu próprio campo receptivo local, englobando, consequentemente, todas as regiões da imagem. Esta operação é muito vantajosa, pois permite uma análise detalhada e espacial em todas as regiões da imagem. Se um recurso, por exemplo, possui seus pesos configurados para reconhecer linhas horizontais, ao ser aplicado em qualquer imagem, resultará numa matriz onde todas as linhas horizontais estão ressaltadas.

Conforme a complexidade da ConvNet aumenta, faz-se necessário o uso de novas terminologias. O mapa que possui as características que serão analisadas (recursos) é denominado mapa de recursos. Os pesos que definem o mapa de recurso (valores dos pixels) são chamados de pesos compartilhados. O viés utilizado no mapa de recurso é denominado viés compartilhado. A junção de pesos e vieses compartilhados configura um núcleo (*kernel*, em inglês) ou filtro.

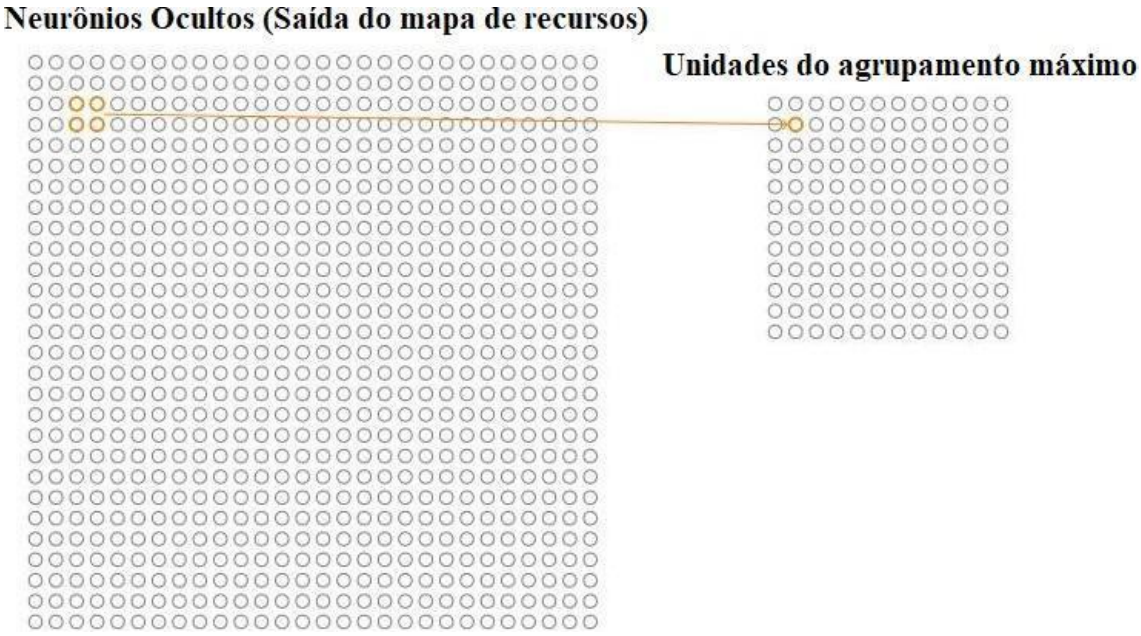
9.8 Camada de Agrupamento

Uma Camada de Agrupamento (*Pooling*, em inglês) é responsável por reduzir as dimensões de imagens enquanto armazena suas principais informações. A camada recebe a saída da camada anterior e, através de cálculos, comprime as informações de modo que os principais recursos e possíveis objetos de reconhecimento sejam armazenados em uma nova matriz bidimensional.

Vamos ao seu funcionamento... Existem 3 principais técnicas de agrupamento: agrupamento mínimo; agrupamento médio e agrupamento máximo. Neste caso, será utilizada a última técnica.

Uma região da imagem (geralmente 2 pixels de largura e 2 pixels de altura) é selecionada e o pixel com maior valor (neste caso, o pixel mais claro) é armazenado. Aplicando o Max-Pooling, portanto, 1 pixel será selecionado em meio a 4. Em outras palavras, a nova matriz terá, geralmente, somente ¼ da quantidade de pixels presentes na matriz bidimensional da camada anterior, como consta a Figura 9.9.

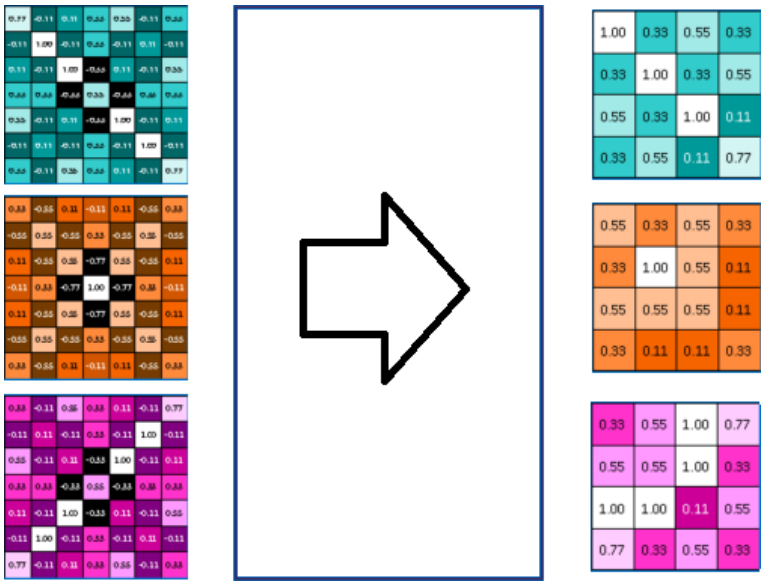
Figura 9.9 - Agrupamento



Fonte: (adaptado) disponível em https://programmerclick.com/article/88061367322/.

Após a finalização da tarefa, os valores obtidos serão organizados em uma nova matriz bidimensional e, considerando que a quantidade de números de parâmetros necessários para processamentos futuros foi diminuída e a porcentagem de recursos críticos aumentou, a Rede consegue realizar uma convolução eficiente, ainda que a localização entre os recursos e a imagem em questão esteja ligeiramente diferente.
Novamente, o agrupamento será realizado em cada um dos mapas gerados pela camada anterior, como consta a Figura 9.10.

Figura 9.10 - Agrupamento de todas matrizes bidimensionais



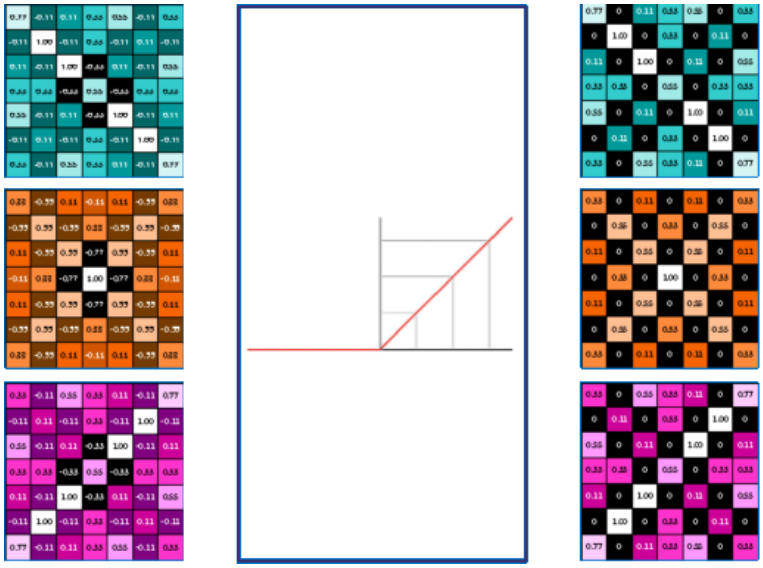
Fonte: (adaptado) disponível em https://e2eml.school/how_convolutional_neural_networks_work.html.

9.9 Unidades Lineares Retificadas

A Unidade Linear Retificada (*Rectified Linear Unit*, em inglês) ou ReLU converte todos os números negativos obtidos na camada anterior para 0. Assim, a Rede garante

que os números aprendidos não ficarão presos próximos a 0 ou próximos ao infinito. A ReLU irá converter os números negativos presentes em todas as matrizes bidimensionais, como consta a Figura 9.11.

Figura 9.11 - Aplicação da ReLU nas matrizes bidimensionais



Fonte: (adaptado) disponível em https://e2eml.school/how_convolutional_neural_networks_work.html.

9.10 Organização De Camadas

Até este ponto, todas as camadas geram uma matriz bidimensional como saída. Assim, podem ser facilmente utilizadas como dados de entrada em camada seguintes, como a Camada de Agrupamento pode receber como entrada a saída da Camada Convolucional, por exemplo.

Empilhando as camadas, por sua vez, as possíveis operações presentes numa ConvNet são construídas baseando-se umas nas outras. Além disso, nota-se que não existe uma ordem única e correta de camadas, devendo ser organizadas conforme experimentação no desenvolvimento da aplicação.

9.11 Camada Totalmente Conectada

Em uma Camada Totalmente Conectada (*Fully Connected Layer*, em inglês), todas as matrizes bidimensionais serão convertidas em apenas 1 matriz unidimensional. Em outras palavras, todos os neurônios serão agrupados numa única lista. Em sequência, cada um dos neurônios se conectará com cada uma das entradas da camada seguinte.

A Camada Totalmente Conectada não se restringe à ser conectada na Camada de Saída, podendo ser utilizada, também, para futuro processamento na Rede.

9.12 Camada De Saída

A Camada de Saída (*Output Layer*, em inglês) irá conter todos os objetos de detecção possíveis. No exemplo construído ao longo do capítulo, por exemplo, a saída é formada por 2 neurônios, representando a probabilidade de cada uma das letras.

Caso a aplicação receba uma imagem contendo a letra “X”, como consta o primeiro exemplo presente na Figura 9.3, um determinado grupo de neurônios será mais estimulado que outros. Em compensação, caso a aplicação receba uma imagem contendo a letra “O”, um grupo de neurônios diferente será estimulado.

Supondo que uma imagem contendo a letra “X” tenha sido utilizada como entrada na aplicação construída durante o capítulo. Em termos práticos, no final de sua execução, todos os neurônios que apontaram para o resultado “X” terão seus valores somados entre si. Em sequência, será calculada a média deste valor (divide-se o resultado da soma pela quantidade de neurônios). Este cálculo também é realizado utilizando os valores dos neurônios que apontaram para o resultado “O”.

Os valores obtidos mostram o resultado da aplicação. Se a saída resultou em 0,91 de probabilidade da imagem conter a letra “X”, e 0,1, por exemplo, de conter a letra “O”, pode-se dizer que a Rede classificou corretamente, embora ainda possua espaço para melhorias e calibração.

9.13 Retropropagação (Backpropagation)

A saída obtida na Camada de Saída pode apresentar erros, principalmente quando a Rede não foi treinada o suficiente e não está calibrada adequadamente. Os detalhes de cada erro obtido são cruciais para aumentar a acurácia da aplicação, já que são utilizados para determinar o quanto a Rede deve ser ajustada.

Cada erro é calculado subtraindo a resposta obtida na Camada de Aplicação da resposta correta e ideal. Desta forma, o valor que será ajustado dependerá do quão grave foi o erro. Se a resposta obtida na Camada de Saída foi ligeiramente fora do esperado, os valores que serão alterados na Rede serão baixos. Do contrário, se o valor obtido estiver muito longe do ideal, os valores que serão alterados também serão altos.

Em compensação, a ConvNet também se baseará em parâmetros selecionados pelo designer da aplicação. É responsabilidade do criador da aplicação definir, por exemplo: o tamanho da janela de agrupamento, a quantidade de passos de pixels (*stride*, em inglês), o número de neurônios na Camada Totalmente Conectada, a quantidade de cada tipo de camada, a ordem das camadas etc.

9.14 Aplicação de uma Convnet na Tarefa de Detecção Facial

Estão presentes no Anexo B instruções detalhadas para instalação de software para virtualização de máquinas virtuais; criação de uma máquina virtual; instalação de um sistema operacional para realizar a aplicação de uma ConvNet e a instalação e atualização de dependências no sistema operacional (neste caso, Linux Mint).

9.14.1 Implementação Do Código

Para compilar o código corretamente, deve-se primeiramente baixar tanto o arquivo de detecção quanto a imagem em que se deseja realizar os testes, conforme Script 9.1 e Script 9.2, respectivamente.

Script 9.1 - Baixando arquivo de detecção

```
cd ~/Documentos/CNN && wget https://github.com/justadudewhohacks/face-recognition.js-models/raw/master/models/mmod_human_face_detector.dat
```

Script 9.2 - Baixando imagem para testes

```
cd ~/Documentos/CNN && wget https://images.generated.photos/wNBCjycfAo-aK-71TBH0CeMxAo28FiwQ31du6kJ2xK8/rs:fit:512:512/wm:0.95:sowe:18:18:0.33/czM6Ly9pY29uczgu/Z3Bob3Rvcy1wcm9k/LnBob3Rvcy92M18w/NDY0NzA5LmpwZw.jpg && mv NDY0NzA5LmpwZw.jpg img.jpg
```

O comando “wget”, transfere os arquivos de um website para a máquina em uso.

O comando “mv”, neste caso, substitui o primeiro argumento com o segundo. Neste caso, é responsável por alterar o nome original da imagem para “img,jpg”.

Neste ponto, todos os arquivos necessários para a compilação do programa estão no local correto. Em sequência, deve-se adicionar o Código 9.1 no arquivo IPYNB criado previamente.

Código 9.1 - Detecção facial

```
import cv2
import dlib

from matplotlib import pyplot as plt

def imprima_imagem(imagem, titulo):
    imagem_RGB = imagem[:, :, ::-1]

    plt.imshow(imagem_RGB)
    plt.title(titulo)
    plt.axis('off')

def imprima_deteccao(imagem, rostos):
    for rosto in rostos:
        cv2.rectangle(imagem,      (rosto.rect.left(),      rosto.rect.top()), (rosto.rect.right(), rosto.rect.bottom()), (255, 0, 0),10)

    return imagem

detector_facial                                                                    = dlib.cnn_face_detection_model_v1("mmod_human_face_detector.dat")

imagem = cv2.imread("img.jpg")

retangulos = detector_facial(imagem, 0)
imagem_rostos = imprima_deteccao(imagem.copy(), retangulos)

plt.suptitle("Detecção facial utilizando Redes Neurais Convolucionais através da biblioteca Dlib", fontsize=14, fontweight='bold')
imprima_imagem(imagem_rostos,   "Quantidade   de   faces   presentes:   "   + str(len(retangulos)))
plt.show()
```

Fonte: (adaptado) disponível em https://gist.github.com/edward1986/648a8fb5a51088c5bd368d95648625ac.

Explicação do código:

```
import cv2
```

```
import dlib
```

São importadas 2 bibliotecas. "cv2" (OpenCV) é responsável pela leitura de imagens e destaque das faces. "dlib" é responsável por detectar as faces presentes na imagem.

```
from matplotlib import pyplot as plt
```

É importada como "plt" a coleção de funções "pyplot" contidas na biblioteca Matplotlib. É responsável por criar visualizações estáticas de dados.

```
def imprima_imagem(imagem, titulo):
```

É criada a função "imprima_imagem" que configura e ajusta os parâmetros para o plot final (tela de impressão) da aplicação.

```
imagem_RGB = imagem[:, :, ::-1]
```

Inverte a ordem de cores original “Blue, Green e Red” para “Red, Green e Blue”.

```
plt.imshow(imagem_RGB)
```

Imprime a imagem convertida em RGB na figura (que, por sua vez, serve como canvas para a plot final).

```
plt.title(titulo)
```

Define o título que aparecerá na plot.

```
plt.axis('off')
```

Remove os eixos x e y da plot final.

```
def imprima_deteccao(imagem, rostos):
```

É criada a função "imprima_deteccao" possuindo como parâmetros as variáveis "imagem" e "rostos". É responsável por destacar cada rosto presente na imagem através de um retângulo azul.

```
for rosto in rostos:
```

É utilizada uma estrutura de repetição "for", onde cada valor presente na variável "rostos" será indexado. Ou seja, a repetição irá iterar por cada rosto presente na imagem.

```
cv2.rectangle(imagem, (rosto.rect.left(), rosto.rect.top()),
(rosto.rect.right(), rosto.rect.bottom()), (255, 0, 0),10)
```

É utilizada a função “rectangle”, presente na biblioteca “cv2”, para desenhar um retângulo ao redor de cada rosto encontrado.

return imagem

A função retorna a imagem passada como argumento com os rostos destacados.

```
detector_facial = dlib.cnn_face_detection_model_v1("mmod_human_face_detector.dat")
```

É criada a variável de detecção através da função "cnn_face_detection_model_v1", disponível na biblioteca Dlib. A função recebe como argumento o modelo treinado disponível em http://dlib.net/files/mmod_human_face_detector.dat.bz2.

```
imagem = cv2.imread("img.jpg")
```

É criada uma variável "imagem" que recebe como valor a leitura da imagem através da função "imread", disponível na biblioteca “cv2”.

```
retangulos = detector_facial(imagem, 0)
```

É criada a variável "retangulos" que recebe o detector criado previamente. O detector, por sua vez, recebe a imagem como primeiro argumento.

```
imagem_rostos = imprima_deteccao(imagem.copy(), retangulos)
```

É criada a variável "imagem_rostos", que receberá o valor retornado na função "imprima_deteccao" (a imagem com os rostos presentes destacados). A função, por sua vez, recebe como primeiro argumento a cópia da imagem original e, para finalizar, a variável que representa a localização individual de cada rosto, "retangulos".

```
plt.suptitle("Detecção facial utilizando Redes Neurais Convolucionais através da biblioteca Dlib", fontsize=14, fontweight='bold')
```

Define o título da janela que apresentará o resultado da aplicação. É utilizada a função "suptitle", disponível na biblioteca Matplotlib, que recebe como primeiro argumento o tamanho da fonte e, para finalizar, o estilo da fonte.

```
imprima_imagem(imagem_rostos, "Quantidade de faces presentes: " + str(len(retangulos)))
```

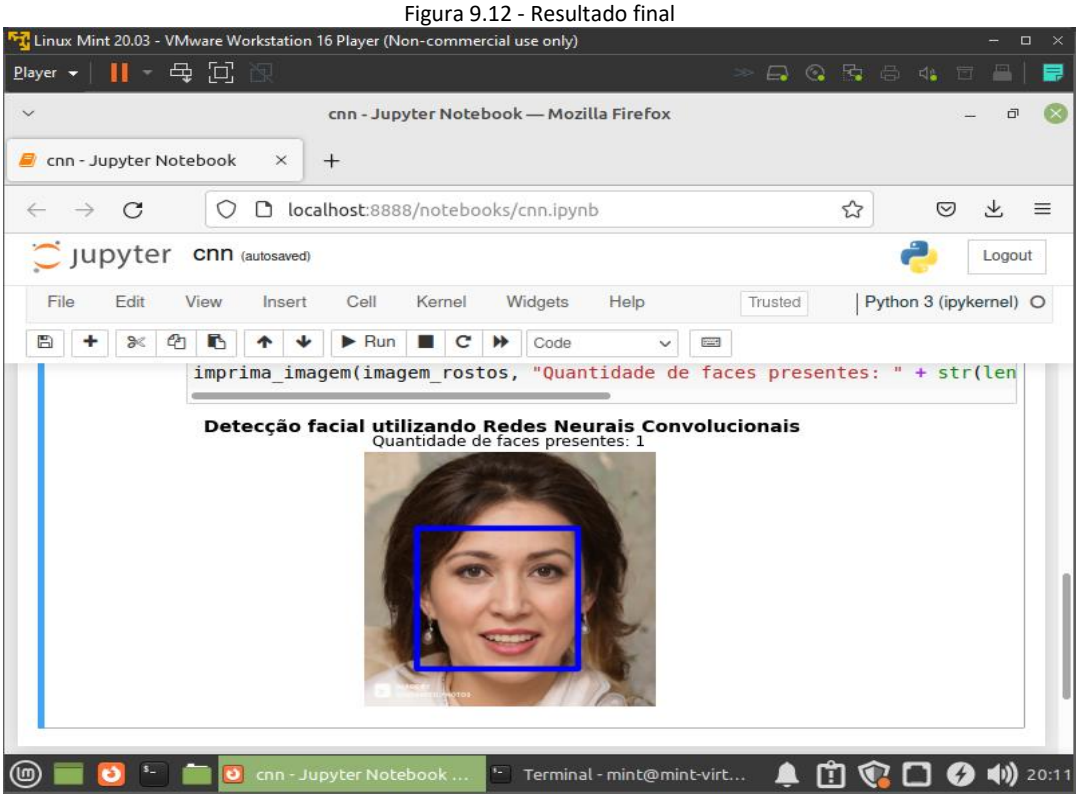
Através da função "imprima_imagem" é apresentado o resultado final da aplicação (a imagem destacando as faces encontradas). O primeiro argumento é a própria imagem com os rostos já destacados, "imagem_rostos" e, finalizando, o título.

```
plt.show()
```

Imprime a plot.

Deve-se, em sequência, pressionar as teclas SHIFT + ENTER, a fim de compilar o bloco de código. Finalizando, será apresentada a imagem com as faces destacadas como resultado final da aplicação, como consta a Figura 9.12. Neste caso, a face reconhecida foi gerada através de inteligência artificial, podendo ser obtido através do seguinte link: https://generated.photos/face/joyful-white-young-adult-female-with-medium-brown-hair-and-brown-eyes--5e680c0d6d3b380006d6243d.

Figura 9.12 - Resultado final



9.14.2 Utilização de Servidores

Existem outras alternativas disponíveis quando se trata de acessar uma distribuição Linux além da instalação de máquinas virtuais, como a utilização de servidores. Os serviços escolhidos, neste caso, são da empresa Linode, uma empresa americana de hospedagem em nuvem que fornece servidores virtuais privados. Além de preços competitivos, fornece serviços de acordo com as necessidades de cada usuário.

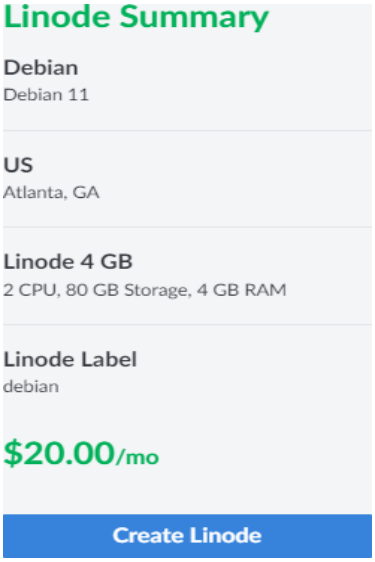
A página de configuração de servidores está disponível em https://cloud.linode.com/linodes/create (nota-se que é necessário informar os dados pessoais e bancários). Considerando que existe um número limitado de sistemas operacionais disponíveis para contratação e que o Linux Mint 20.03 não consta como um deles, é necessário selecionar a distribuição mais próxima, neste caso, Debian 11.

A região selecionada deve ser a que possui menor latência, a fim de aumentar a velocidade do servidor. A Linode disponibiliza um teste de velocidade disponível em https://www.linode.com/pt/speed-test/. A opção escolhida foi Atlanta–Estados Unidos, já que possui a menor latência entre as alternativas.

Em relação ao desempenho do servidor, a CPU (abreviação de Central Processing Unit — unidade central de processamento, em português) compartilhada escolhida é a “Linode 4 GB”, possuindo 4 gigas de memória ram, 2 CPU’s e 80 gigas de armazenamento.

O nome escolhido, embora não afete prejudicialmente em nenhuma etapa, é “debian”. A Figura 9.13 representa as configurações finais escolhidas para o servidor.

Figura 9.13 - Configuração do servidor



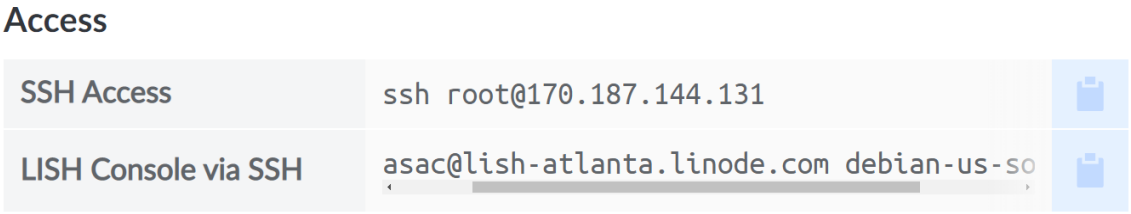
Ao selecionar a opção “Create Linode”, será criado o servidor.
Nota-se que o valor a ser pago é proporcional a quantidade de horas que o servidor foi utilizado.

Após a criação do servidor, serão apresentadas informações como credenciais, status geral, configurações etc.

Para acessar o servidor, deve-se abrir a CLI do sistema operacional hospedeiro (“Prompt de comando” para versões do Windows 10 e “Terminal” para sistemas Unix) e, em sequência, realizar o login utilizando o protocolo de comunicação SSH (Secure Shell) para realizar a comunicação entre 2 computadores.

Cada servidor criado terá um endereço de IP diferente. Portanto, é necessário consultar o website para verificação, conforme Figura 9.14.

Figura 9.14 - Acesso ao servidor



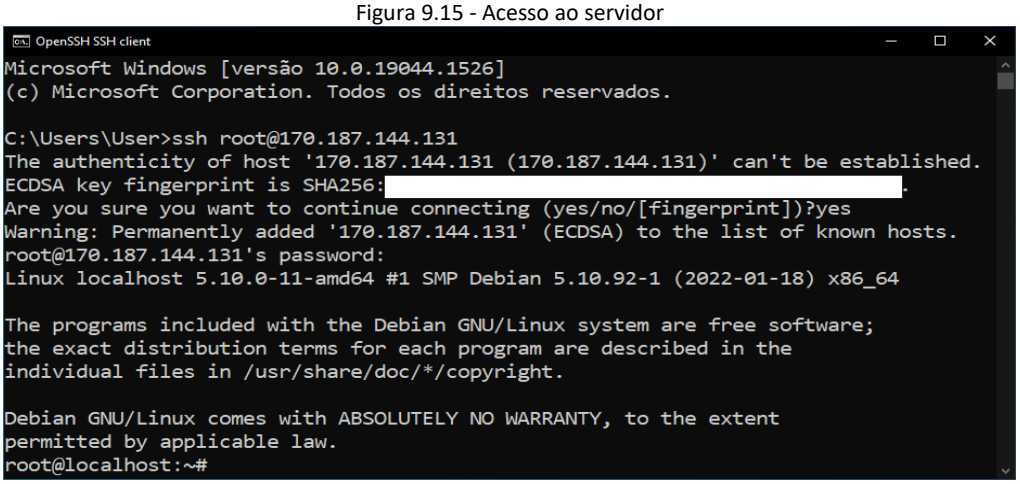
Na seção “Access” é informado o comando completo que deve ser executado para realizar o login no servidor, conforme Script 9.3.

Script 9.3 - Utilização do servidor

```
ssh root@170.187.144.131
```

Fonte: disponível na página de configuração do servidor criado (link variável).

Figura 9.15 - Acesso ao servidor



Após inserir as credenciais criadas previamente em qualquer terminal, o servidor estará pronto para ser utilizado.

Embora o servidor não possua uma *Graphical User Interface* (Interface Gráfica de Usuário, em português), o processo de instalação e atualização de pacotes é similar ao realizado em uma máquina virtual. Para atualizar o sistema, deve-se executar os comandos presentes no Script 9.4.

Script 9.4 - Atualização da máquina

```
apt update && apt upgrade
```

Em sequência, deve-se instalar todas as dependências necessárias para a instalação do Anaconda, como consta o Script 9.5.

Script 9.5 - Instalação de pré-requisitos

```
apt install git libgl1-mesa-glx libegl1-mesa libxrandr2 libxss1 libxcursor1
libxcomposite1 libasound2 libxi6 libxtst6
```

Fonte: (adaptado) disponível em https://docs.anaconda.com/anaconda/install/linux/#.

Utilizando o comando “wget”, será baixado o instalador do Anaconda para Linux, como consta o Script 9.6.

Script 9.6 - Instalação de pré-requisitos

```
wget https://repo.anaconda.com/archive/Anaconda3-2021.11-Linux-x86_64.sh
```

O Script 9.7 verifica a integridade do arquivo baixado utilizando o comando “sha256sum”.

Script 9.7 - Verificação de integridade

```
sha256sum Anaconda3-2021.11-Linux-x86_64.sh
```

Fonte: (adaptado) disponível em https://docs.anaconda.com/anaconda/install/linux/#.

Novamente, é necessário certificar de que a saída produzida pelo comando é a mesma presente no website oficial. Se tratando do mesmo arquivo, a saída produzida foi igual a “fedf9e340039557f7b5e8a8a86affa9d299f5e9820144bd7b92ae9f7ee08ac60”, que condiz exatamente com a disponível no website oficial: https://docs.anaconda.com/anaconda/install/hashes/Anaconda3-2021.11-Linux-

x86_64.sh-hash/#:~:text=fedf9e340039557f7b5e8a8a86affa9d299f5e9820144bd7b92ae9f7ee08ac60.

Finalmente, deve-se executar os comandos presentes no Script 9.8, a fim de começar o processo de instalação do Anaconda.

Script 9.8 - Atualização da máquina

```
bash Anaconda3-2021.11-Linux-x86_64.sh
```

Fonte: (adaptado) disponível em https://docs.anaconda.com/anaconda/install/linux/#.
Nota-se que o processo de instalação pode ser acompanhado na Figura 9.15 (Parte A - E).

Para efetivar as novas alterações, deve-se reiniciar o sistema.

Diferentemente da máquina virtual, será criado um ambiente virtual isolado do restante do sistema. Neste caso, isso é vantajoso pois torna possível a utilização de versões específicas de pacotes e dispensa a necessidade de instalar pacotes que não serão utilizados. A criação do ambiente virtual pode ser feita através do Script 9.9.

Script 9.9 - Criação do ambiente virtual

```
conda create -n cnn_env
```

Fonte: (adaptado) disponível em https://www.geeksforgeeks.org/how-to-setup-conda-environment-with-jupyter-notebook/.
Para prosseguir, o programa necessita que o usuário digite “y”, seguido da tecla ENTER.

Para ativar o ambiente, deve-se executar o Script 9.10.

Script 9.10 - Ativação do ambiente virtual

```
conda activate cnn_env
```

Fonte: (adaptado) disponível em https://www.geeksforgeeks.org/how-to-setup-conda-environment-with-jupyter-notebook/.

Neste ponto, onde o ambiente virtual já está sendo utilizado, deve-se instalar os pacotes que serão necessários para executar a aplicação. Assim como na máquina virtual, o código será compilado e executado através da plataforma Jupyter Notebook e, para instalar a mesma, deve-se executar o Script 9.11.

Script 9.11 - Instalação do Jupyter

```
conda install jupyter notebook
```

Fonte: (adaptado) disponível em https://www.geeksforgeeks.org/how-to-setup-conda-environment-with-jupyter-notebook/.
Para prosseguir, o programa necessita que o usuário digite “y”, seguido da tecla ENTER.

Novamente, serão instaladas as bibliotecas “OpenCv”, “Dlib” e “Matplotlib”, conforme Script 9.12 e Script 9.13, respectivamente.

Script 9.12 - Instalação da OpenCV

```
pip install opencv-python matplotlib
```

Fonte: disponível em https://pypi.org/project/opencv-python/.

Script 9.13 - Instalação da Dlib

```
conda install -c conda-forge dlib
```

Fonte: disponível em https://anaconda.org/conda-forge/dlib.
Para prosseguir, o programa necessita que o usuário digite “y”, seguido da tecla ENTER.

Para criar uma pasta reservada para o projeto, é executado o comando presente no Script 9.14.

Script 9.14 - Criação do ambiente de trabalho

```
mkdir ~/cnn
```

Assim como anteriormente, deve-se baixar tanto o arquivo de detecção quanto a imagem em que se deseja realizar os testes, conforme Script 9.15 e Script 9.16, respectivamente.

Script 9.15 - Baixando arquivo de detecção

```
cd ~/cnn && wget https://github.com/justadudewhohacks/face-recognition.js-models/raw/master/models/mmod_human_face_detector.dat
```

Script 9.16 - Baixando imagem para testes

```
cd ~/cnn && wget https://images.generated.photos/wNBCjycfAo-aK-71TBH0CeMxAo28FiwQ31du6kJ2xK8/rs:fit:512:512/wm:0.95:sowe:18:18:0.33/czM6Ly9pY29uczgu/Z3Bob3Rvcy1wcm9k/LnBob3Rvcy92M18w/NDY0NzA5LmpwZw.jpg && mv NDY0NzA5LmpwZw.jpg img.jpg
```

Em sequência, o código Python será adicionado em um arquivo Jupyter (extensão IPYNB) conforme Script 9.17.

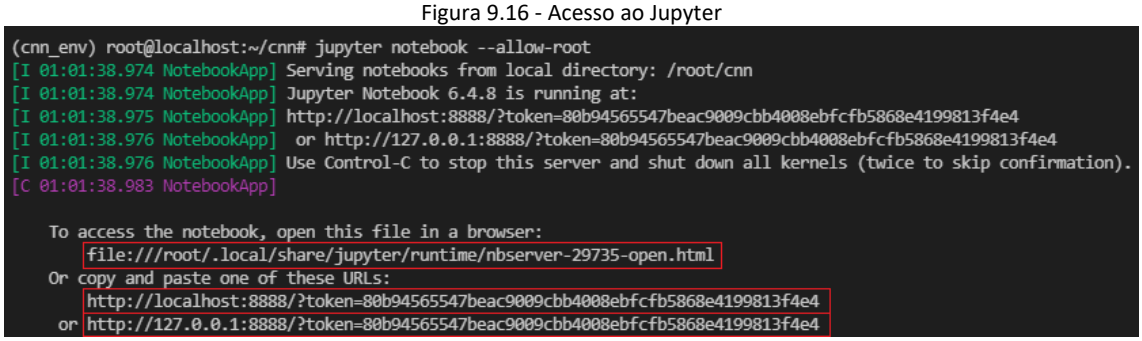
Script 9.17 - Utilização de um Jupyter Notebook

```
cd ~/cnn && jupyter notebook --allow-root
```

A *flag* (bandeira, em português) “allow-root” é utilizada pois o servidor está sendo executado em modo de administrador.

A plataforma Jupyter é baseada em uma Interface Gráfica de Usuário, porém, dependendo de como o servidor está sendo acessado, a mesma não estará disponível. Em compensação, nota-se que ao executar o Script 9.17, a saída do comando informa um endereço que pode ser acessado diretamente no computador host, conforme Figura 9.16.

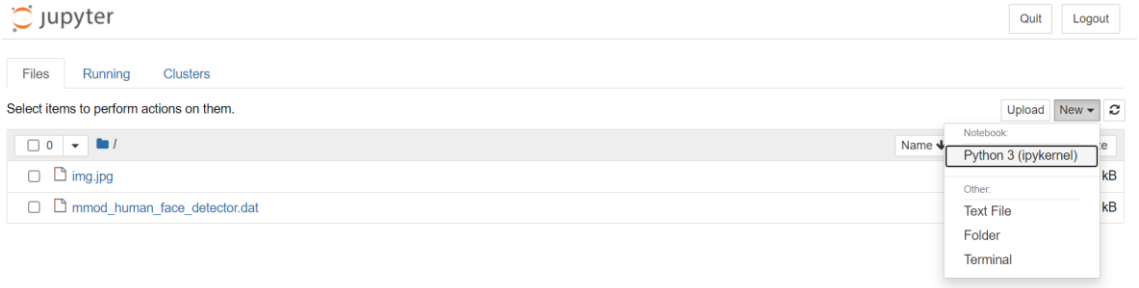
Figura 9.16 - Acesso ao Jupyter



Qualquer um dos 3 endereços destacados em vermelho pode ser acessado a fim de acessar uma Interface Gráfica de Usuário para a plataforma Jupyter.

Ao abrir a plataforma Jupyter, deve-se criar um arquivo conforme Figura 9.17.

Figura 9.17 - Criação do pykernel



Finalmente, deve-se adicionar o Código 9.1 e, em sequência, pressionar as teclas SHIFT + ENTER, para compilar e executar o bloco de código. Após a finalização da compilação, a aplicação deve mostrar a imagem com as faces presentes destacadas através de retângulos azuis, bem como a quantidade de faces presentes (neste caso, 1), conforme Figura 9.18.

Figura 9.18 - Resultado da aplicação

9.14.3 Implementação do Código 2

O Código 9.2 analisa uma base de imagens já transferida anteriormente e realiza a detecção facial em todas as imagens presentes no diretório em questão. É possível encontrar mais detalhes na obtenção desta base de dados e os procedimentos realizados nela no relatório final (compilação do autor).

Código 9.2 - Detecção de faces em massa

```
import dlib
import urllib
import time

def cnn(x_train):
    urlretrieve("https://github.com/justadudewhohacks/face-recognition.js-
models/raw/master/models/mmod_human_face_detector.dat",
"mmod_human_face_detector.dat")
    facial_detector                                                    =
cnn_face_detection_model_v1("mmod_human_face_detector.dat")
```

```
    for image in x_train:
        start_point = time()
        face = facial_detector(image, 0)
        end_point = time()

        if len(face) > 0:
            y_train_predict.append(1)
        else:
            y_train_predict.append(0)
        timing.append(end_point - start_point)

    return y_train_predict, timing

# path = "caminho_das_imagens"
# cnn(path)
```

Explicação do código:

```
import dlib
import urllib
import time
```

São importadas as bibliotecas “cv2”, “urllib” e “time”, que serão utilizadas para a detecção de imagens, carregamento de arquivos necessários para a execução da aplicação e registro de tempo, respectivamente.

```
y_train_predict = [], timing = []
```

São criadas as variáveis “y_train_predict” e “timing”. Que retornarão os resultados da aplicação e o tempo de processamento dos mesmos, respectivamente.

```
def cnn(x_train):
```

É definida a função “cnn”. Esta será responsável por retornar os resultados obtidos nas análises das imagens e de seus respectivos tempos. A função recebe como parâmetro o caminho do diretório que contém as imagens, "x_train".

```
urlretrieve("https://github.com/justadudewhohacks/face-recognition.js-
models/raw/master/models/mmod_human_face_detector.dat",
"mmod_human_face_detector.dat")
```

A variável “facial_detector” carregará o modelo de detecção a partir do arquivo descarregado “mmod_human_face_detector.dat”, sendo este o arquivo que possui as características de uma face frontal. Para efetivamente criar a função, deve-se utilizar a função “cnn_face_detection_model_v1”, presente na biblioteca Dlib. O resultado da mesma deve ser armazenado na variável “facial_detector”, onde seu único argumento é a o próprio arquivo de detecção.

```
for image in x_train:
```

Para cada imagem presente no diretório, deve-se, em ordem:

```
start_point = time()
```

Registrar o momento antes da detecção.

```
face = facial_detector(image, 0)
```

Verificar a presença de faces. A variável “face” receberá o valor retornado pelo modelo de detecção criado anteriormente. Como é possível observar, o modelo recebe como argumento a imagem que o laço “for” está indexando no momento.

```
end_point = time()
```

Registrar o momento após a detecção.

```
if len(face) > 0:
            y_train_predict.append(1)
        else:
            y_train_predict.append(0)
```

Caso a variável “face” (que já possui os valores respectivos à análise realizada na imagem) possua tamanho maior do que 0, será adicionado o valor 1 à variável “y_train_predict”. Caso contrário, 0. 1, neste caso, indica que a imagem analisada é positiva e 0, neste caso, indica uma imagem negativa.

```
timing.append(end_point - start_point)
```

Finalizando o laço, subtrai-se o tempo final do tempo inicial, obtendo o tempo de execução. Este valor é adicionado na lista "timing".

```
return y_train_predict, timing
```

Finalmente, devem ser retornadas as listas contendo os resultados individuais das imagens, bem como seus respectivos tempos de execução. Ou seja, retornam-se as listas "y_train_predict" e "timing".

```
# path = "caminho_das_imagens"
# cnn(path)
```

Nota-se que caso a aplicação desenvolvida seja executada sem alterações, a mesma não retornaria resultados, considerando que a função principal da aplicação (“cnn”) não foi executada. Para isso, deve-se remover os comentários das 2 linhas selecionadas acima e alterar o valor da variável “path”. Esta, por sua vez, receberá o caminho das imagens desejadas. Finalizando, deve-se adicionar a variável como argumento na função “cnn”.

9.15 Criação de uma Rede Neural Convolucional

Para criar uma Rede Neural Convolucional, existem alguns elementos chave que devem ser explicados anteriormente. O primeiro é um dataset contendo imagens positivas e negativas. Neste caso, o dataset pode ser obtido através do seguinte link: https://github.com/andre-alck/cnn-material.git. A estrutura do dataset é organizada da seguinte forma, onde os nomes das imagens, neste caso, são apenas representativos::

1. faces_dataset (pasta contendo todo o material);
    a. test (pasta que contém pastas das imagens de teste após o treinamento);
        i. faces (pasta que contém as imagens positivas para teste);
            1. face.jpg
        ii. nao_faces (pasta que contém as imagens negativas para teste);

            1. nao_face.jpg
    b. train (pasta que contém as pastas das imagens de treinamento);
        i. faces (pasta que contém as imagens positivas para o treinamento);
            1. face.jpg
        ii. faces (pasta que contém as imagens negativas para o treinamento).
            1. nao_face.jpg
    c. val (pasta que contém as pastas das imagens de avaliação durante o treinamento).
        i. faces (pasta que contém as imagens positivas para avaliação do treinamento).
            1. face.jpg
        ii. faces (pasta que contém as imagens negativas para avaliação do treinamento).
            1. nao_face.jpg

O dataset, compilação do autor, é uma modificação de 2 datasets já existentes: o “Faces in the Wild” (disponível em https://www.kaggle.com/c/recognizing-faces-in-the-wild/data) e o “Dogs vs. Cats” (disponível em https://www.kaggle.com/c/dogs-vs-cats/data), onde foram selecionadas 6000 imagens de cada um e organizadas no novo dataset adequadamente, conforme estrutura listada acima. Em relação a quantidade total de imagens, foi reservado 20% para a pasta “test”, 70% para a pasta “train” e 10% para a pasta “val”.

O treinamento, por ser uma tarefa que exige um grande poder computacional, será executado somente no servidor previamente criado, garantindo que a máquina host não seja sobrecarregada. Primeiramente, no servidor, deve-se atualizar o sistema, como consta o Script 9.18.

Script 9.18 - Atualização do sistema

```
apt update && apt upgrade
```

Em sequência, será criada uma pasta reservada para o treinamento da aplicação, conforme Script 9.19.

Script 9.19 - Criação de uma pasta para o treinamento

```
mkdir ~/cnn_treinamento && cd ~/cnn_treinamento
```

Para transferir o dataset que será utilizado durante o treinamento, deve-se clonar o repositório armazenado no GitHub, como consta o Script 9.20.

Script 9.20 - Criação de uma pasta para o treinamento

```
git clone https://github.com/andre-alck/cnn-material.git
```

Considerando que a aplicação precisará de todos os pacotes baixados previamente no servidor, não há necessidade de se criar um novo ambiente virtual. Portanto, deve-se utilizar o ambiente “cnn_env”, como consta o Script 9.21.

Script 9.21 - Ativação do ambiente “cnn_env”

```
conda activate cnn_env
```

O único novo pacote que deve ser instalado é o “tensorflow”, uma biblioteca de código aberto para aprendizado de máquina especializada em criação e treinamento de redes neurais que pode ser obtida através do comando “pip”. Este comando, por sua vez, deve ser atualizado conforme Script 9.22.

Script 9.22 - Atualização do “pip”

```
pip install --upgrade pip
```

Fonte: disponível em https://www.tensorflow.org/install/pip?hl=pt-br.

Finalizando, deve-se executar os comandos presentes no Script 9.23 para instalar o TensorFlow.

Script 9.23 - Instalação do TensorfFlow

```
pip install --upgrade tensorflow
```

Fonte: disponível em https://www.tensorflow.org/install/pip?hl=pt-br.

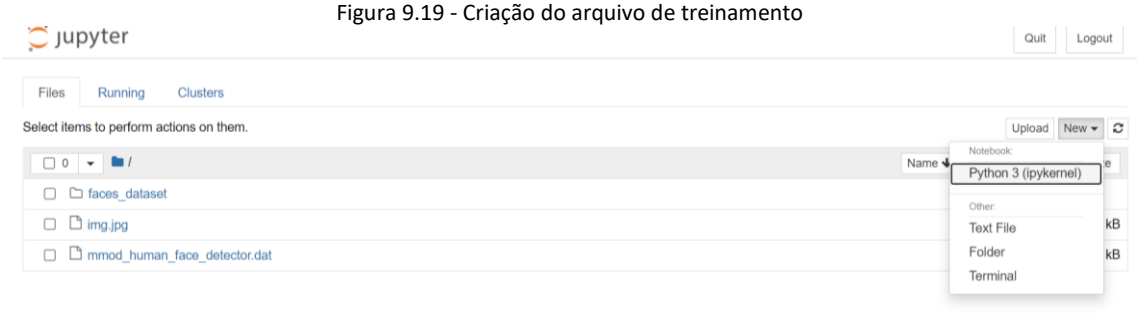
Assim como anteriormente, o código com o código será adicionado em um arquivo IPYNB na plataforma Jupyter. Para abrir a plataforma Jupyter no diretório correto e adicionar o código de treinamento, deve-se executar os comandos presentes no Script 9.24.

Script 9.24 - Atualização do “pip”

```
cd ~/cnn_treinamento/cnn-material && jupyter notebook --allow-root
```

Ao abrir a plataforma Jupyter, deve-se criar um arquivo conforme Figura 9.19.

Figura 9.19 - Criação do arquivo de treinamento



Finalmente, deve-se adicionar o Código 9.3 e, em sequência, pressionar as teclas SHIFT + ENTER, para compilar e executar o bloco de código. Após a finalização da compilação, a aplicação deve mostrar a imagem o resultado da avaliação do modelo, conforme Figura 9.20.

Figura 9.20 - Resultado da avaliação

```
75/75 [==============================] - 94s 1s/step - loss: 0.0611 - accuracy: 0.9811
Test accuracy: 0.981
```

9.15.1 Implementação do Código do Treinamento

Código 9.3 - Treinamento do modelo

```
import os, numpy as np, tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.layers.experimental.preprocessing import Rescaling
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import image_dataset_from_directory

current_dir = os.getcwd()
folder = "/faces_dataset"
train_folder = current_dir + folder + "/train"
val_folder = current_dir + folder + "/val"
test_folder = current_dir + folder + "/test"

train_dataset          =          image_dataset_from_directory(train_folder,
image_size=(180,180), batch_size=32)
val_dataset  =  image_dataset_from_directory(val_folder,  image_size=(180,180),
batch_size=32)
test_dataset = image_dataset_from_directory(test_folder, image_size=(180,180),
batch_size=32)

model = keras.Sequential()
model.add(Rescaling(scale=1.0/255))
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
callbacks = [
    ModelCheckpoint(
        filepath="model.keras",
        save_best_only=True,
        monitor="val_loss"
    )
]
history = model.fit(
    train_dataset,
    epochs=10,
    validation_data=val_dataset,
    callbacks=callbacks)

model = keras.models.load_model("model.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

Fonte: (adaptado) disponível em https://github.com/lucaslattari/neural-network-series/blob/main/09/dogs_and_cats_3.ipynb.

Explicação do código:

```
import os, numpy as np, tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.layers.experimental.preprocessing import Rescaling
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import image_dataset_from_directory
```

Neste bloco de código, são importadas diferentes bibliotecas, sendo elas:
"tensorflow": em conjunto com módulos e funções, é responsável pelo aprendizado de máquina e treinamento da rede.

Em sequência, são extraídas módulos e funções específicas de bibliotecas para melhor entendimento e eficiência do código.
"tensorflow.keras.layers": a partir deste módulo, são importadas as funções "Conv2D", "MaxPooling2D", "Flatten" e "Dense", que formarão a estrutura da Rede Neural Convolucional.

"tensorflow.keras.layers.experimental.preprocessing": a partir deste módulo, é importada a função "Rescaling" que, posteriormente, servirá ao propósito de redimensionar a entrada de dados da Rede.

"tensorflow.keras.callbacks": a partir deste módulo, é importada a função ModelCheckpoint, que analisa os resultados durante o treinamento e salva o modelo enquanto estiver no seu ápice.

"tensorflow.keras.utils": a partir deste módulo, é importada a função "image_dataset_from_directory", que gera os datasets necessários para o treinamento.

```
current_dir = os.getcwd()
folder = "/faces_dataset"
train_folder = current_dir + folder + "/train"
val_folder = current_dir + folder + "/val"
test_folder = current_dir + folder + "/test"
```

Através da variável "folder", que representa o dataset original "faces_dataset", são criadas variáveis que corresponderão as pastas contidas no dataset. Neste caso, "train_folder", "val_folder" e "test_folder".

A organização de pastas corresponde a estrutura necessária para utilização do Keras e Tensorflow.

```
train_dataset = image_dataset_from_directory(train_folder,
image_size=(180,180), batch_size=32)
val_dataset = image_dataset_from_directory(val_folder, image_size=(180,180),
batch_size=32)
test_dataset = image_dataset_from_directory(test_folder, image_size=(180,180),
batch_size=32)
```

São gerados 3 datasets (train_dataset, validation_dataset e test_dataset) a partir das pastas “train_folder”, “validation_folder” e “test_folder” previamente criadas.
A função "image_dataset_from_directory" recebe como primeiro argumento a pasta contendo as imagens desejadas divididas em 2 categorias. Neste caso, “faces” e “nao_faces”. Como segundo argumento, recebe uma tupla contendo as dimensões de largura e altura que a imagem deve ser redimensionada. Finalizando, recebe como terceiro argumento o “batch_size”, que consiste em fragmentar a imagem em porções para não sobrecarregar a memória do sistema.

```
model = keras.Sequential()
```

É criado o modelo onde as camadas serão implementadas.

```
model.add(Rescaling(scale=1.0/255))
```

Adiciona-se uma camada de Rescaling que, através do parâmetro "scale", redimensiona a entrada.

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
```

Adiciona-se uma camada Conv2D (camada de convolução). Explicação de seus parâmetros:

Como primeiro argumento, recebe a quantidade de filtros: 32.

"kernel_size": recebe uma tupla de 2 inteiros que especifica a largura e altura da janela de convolução. Neste caso, as dimensões possuem o mesmo valor, 3.

"activation": especifica qual função de ativação será utilizada. Neste caso, "relu".

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Adiciona-se uma camada MaxPooling2D (camada de agrupamento). Seu único parâmetro é o "pool_size", que recebe uma tupla de 2 inteiros que especifica a largura e altura da janela de agrupamento.

Esta sequência (onde é adicionada uma camada de convolução e uma camada de agrupamento) se repetirá mais 3 vezes, onde a única alteração será na quantidade de filtros, presente na função Conv2D, que terá seu valor dobrado progressivamente.

```
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
```

É adicionada uma última camada de convolução, onde a quantidade de filtros será igual a 256.

```
model.add(Flatten())
```

Conforme diferentes camadas são adicionadas na Rede, o número de dimensões existentes também aumenta. No entanto, a função de ativação deve receber as informações de maneira não multidimensional. A camada “Flatten”, portanto, achata a camada, linearizando a matriz.

```
model.add(Dense(1, activation="sigmoid"))
```

Adiciona-se uma camada Dense (camada totalmente conectada). Explicação de seus parâmetros:

“units”: embora oculto, o parâmetro define a dimensionalidade do espaço de saída.

"activation": especifica qual função de ativação será utilizada. Neste caso, "sigmoid".

```
model.compile(loss="binary_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

A função "compile" configura o modelo para treinamento. Explicação de seus parâmetros:

"loss": define a função de perda. Neste caso, "binary_crossentropy", uma função que isola observações em 2 rótulos dados como base.

"optimizer": define o algoritmo de otimização que será utilizado. Neste caso, o algoritmo é um método estocástico de gradiente descendente que se baseia na estimativa adaptativa de momentos de primeira e segunda ordem.

"metrics": lista de métricas que serão avaliadas pelo modelo durante os períodos de treino e teste. Neste caso, a acurácia do modelo.

```
callbacks = [
    ModelCheckpoint(
        filepath="model.keras",
        save_best_only=True,
        monitor="val_loss"
    )
]
```

É criada uma lista denominada "callbacks", que executa a função ModelCheckpoint. Esta, por sua vez, é responsável por gravar o modelo (extensão KERAS) enquanto estiver no seu ápice. Explicação de seus parâmetros:

"filepath": o caminho onde o modelo deverá ser salvo e seu nome.

"save_best_only": se o parâmetro for positivo, define que o modelo será salvo quando for considerado sua melhor versão.

"monitor": define o nome da métrica que será monitorada. Neste caso, "val_loss", para monitorar a perda total.

```
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=val_dataset,
    callbacks=callbacks)
```

É criada uma variável “history”, que recebe como resultado o método model.fit. Explicação de seus parâmetros:

Como primeiro argumento, recebe o dataset de treino criado previamente.

“epochs”: quantidade de épocas que deverão ser executadas no treinamento. Neste caso, 30.

“validation_data”: recebe como primeiro argumento o dataset contendo as imagens de validação e, como segundo argumento, "callbacks", a lista de callbacks que deverão ser aplicadas durante o treinamento.

9.16 Execução do Algoritmo

A matriz de confusão, que pode ser observada na Figura 9.21, demonstra que foram obtidas 589 imagens verdadeiro positivas, 11 imagens falso positivas, 400 imagens falso verdadeiras e 0 imagens falso negativas.

Podemos observar na Figura 9.22 que o ponto da curva ROC se encontra no canto superior esquerdo, onde a taxa de falso-positivos é 0, evidenciando a eficiência do algoritmo.

Figura 9.21 - Matriz de Confusão do CNN

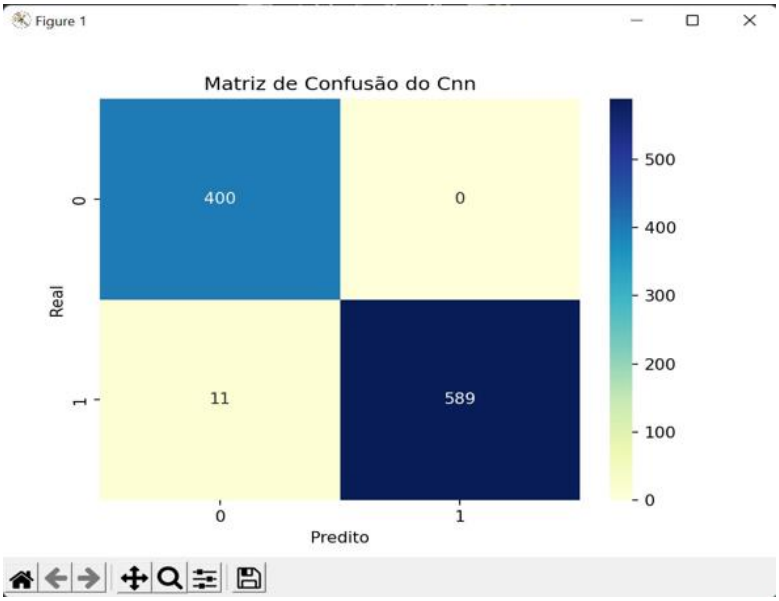
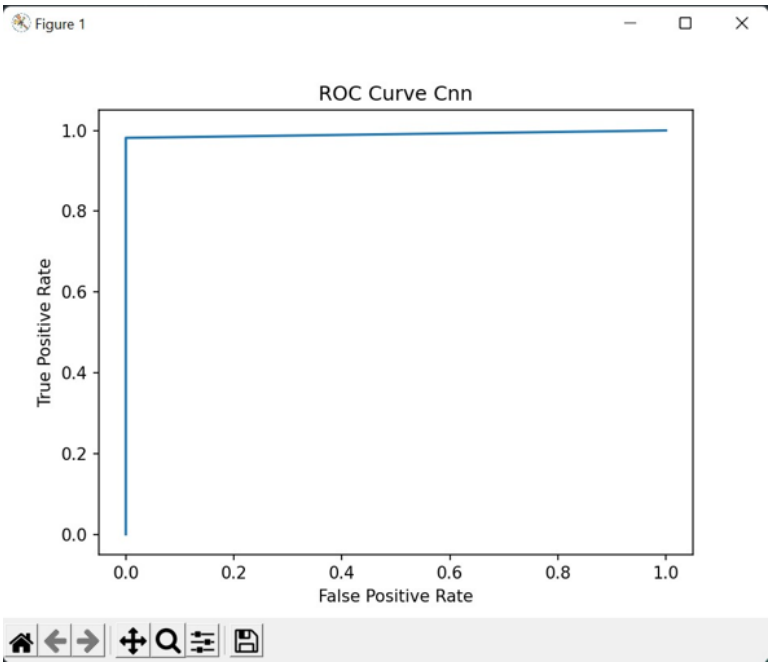
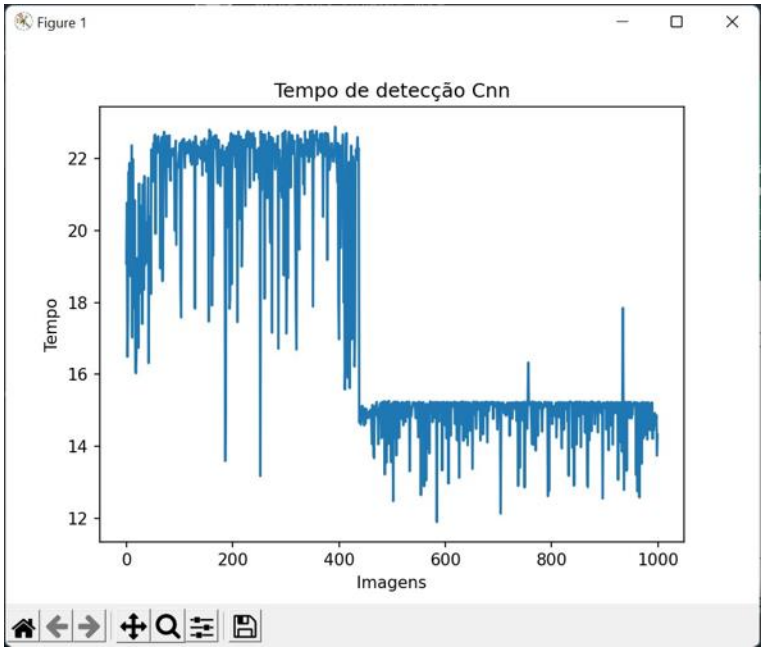


Figura 9.22 - Curva ROC do CNN



Ao analisar a Figura 9.23, nota-se uma diminuição no tempo de detecção após o marco de 400 imagens. Isso ocorre pois imagens positivas necessitam de menos processamento computacional, já que possuem características claras de que a imagem se trata de uma face.

Figura 9.23 - Tempo de detecção do CNN



Fonte: compilação dos autores.

Com isso encerramos mais este capítulo onde explicamos o funcionamento das Redes Neurais Convolucionais e mostramos a sua implementação. No próximo capítulo vamos abordar mais um método/algoritmo de redes neurais, baseado também em redes neurais convolucionais com a adição de mais alguns ingredientes: o DeepFace.

10

DEEPFACE

Nesse capítulo, continuando com o foco em redes neurais, falaremos sobre o DeepFaces e sua tecnologia promissora chamada de Redes Neurais Profundas ou mais conhecida como *Deep Learning*. Um algoritmo focado em reconhecimento facial que foi criado em 2014, sua tecnologia imita a forma do cérebro humano de pensar utilizando neurônios digitais para realizar suas análises.

10.1 Deepfaces

É um método de detecção e reconhecimento facial, muito utilizado nos dias atuais, esse método utiliza de uma tecnologia totalmente inovadora, para um melhor entendimento, é preciso conhecer um pouco sobre a sua história dentro dos métodos de reconhecimento faciais criados nos últimos anos (TAIGMAN, 2014).

No início da década de 1990, o reconhecimento facial ganhou popularidade graças à introdução histórica dos métodos Eigenfaces e Fisherfaces. Abordagens holísticas dominaram a comunidade de reconhecimento facial nas décadas de 1990 e 2000. Sua abordagem fornece representações de baixa dimensão por meio de certas suposições de difusão, como subespaços lineares, polimorfismos e representações de dispersão. O problema com os métodos holísticos é sua incapacidade de lidar com mudanças faciais descontroladas que se desviam de suas suposições anteriores. Isso levou ao desenvolvimento da tecnologia de reconhecimento facial de origem local no início dos anos 2000.

No início dos anos 2000 e 2010, o reconhecimento facial baseado em recursos locais e as descrições locais baseadas em aprendizado foram introduzidos. O reconhecimento facial com filtros Gabor e o modelo binário local conhecido como LBP e suas extensões multinível e de alta dimensão alcançar desempenho robusto por meio de algumas propriedades invariantes do filtro local. Infelizmente, essas características do ofício sofrer com a falta de distinção e concisão. No início de 2010, foi introduzido um descritor local focado no aprendizado para reconhecimento facial. que aprende filtros locais para melhor discriminação e aprenda o guia de código para mais concisão.

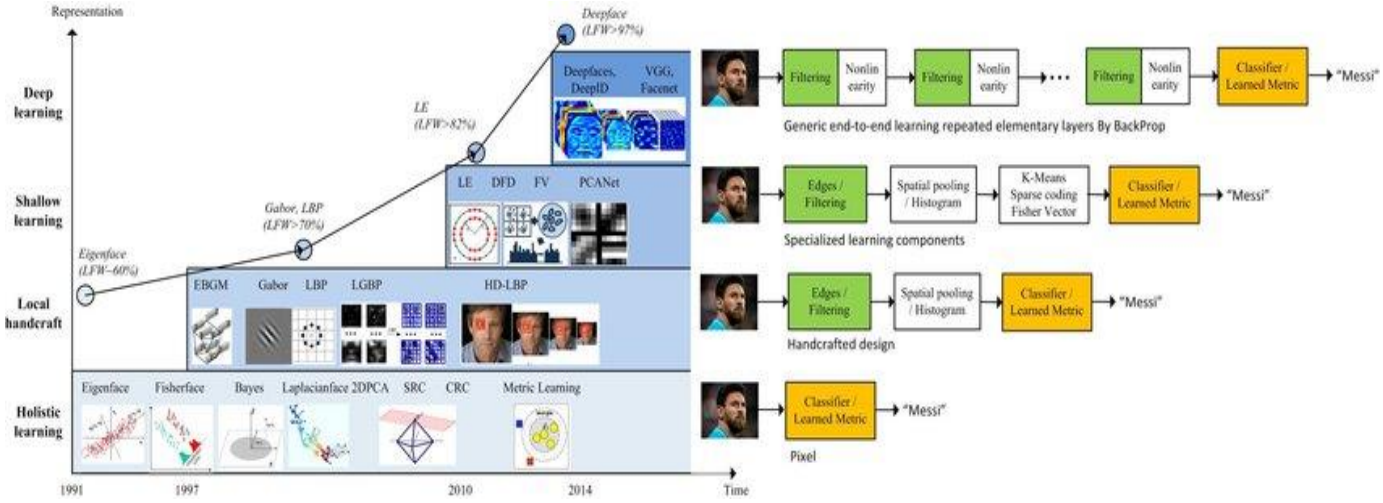
Em 2014, Deepfaces e DeepID criados pela empresa Facebook, alcançaram precisão de última geração quando usados em um famoso teste de desempenho realizado por pesquisadores da universidade de Massachusetts usando um banco de dados de fotografias de rostos desenvolvido para estudar o problema de reconhecimento, conhecido como Labeled Faces in the Wild (LFW), superando o reconhecimento humano pela primeira vez. Desde então, o foco da enquete mudou para abordagens baseadas em aprendizado profundo (TAIGMAN, 2014).

Os métodos de aprendizado profundo usam uma cascata de várias camadas de unidades de processamento para extração e transformação de recursos. Assim, bancos

de dados de rostos em maior escala e técnicas avançadas de processamento de rostos foram desenvolvidas para facilitar o reconhecimento facial profundo. Como resultado, com os pipelines de representação se tornando mais profundos, o desempenho do LFW (Labeled Face in-the-Wild) melhorou constantemente de cerca de 60% para mais de 97%.

Na Figura 10.1, é explicado de forma mais objetiva essa linha do tempo dos métodos de reconhecimento faciais mais recentes, segundo os autores do método DeepFaces.

Figura 10.1: Representação dos marcos de reconhecimento facial.



Fonte: WANG; DENG, 2020

O aprendizado profundo, em particular as redes neurais profundas (DNN) que será um tema abordado logo a seguir, tem recebido crescente interesse para ser utilizado em reconhecimento facial, e vários métodos de aprendizado profundo têm sido propostos.

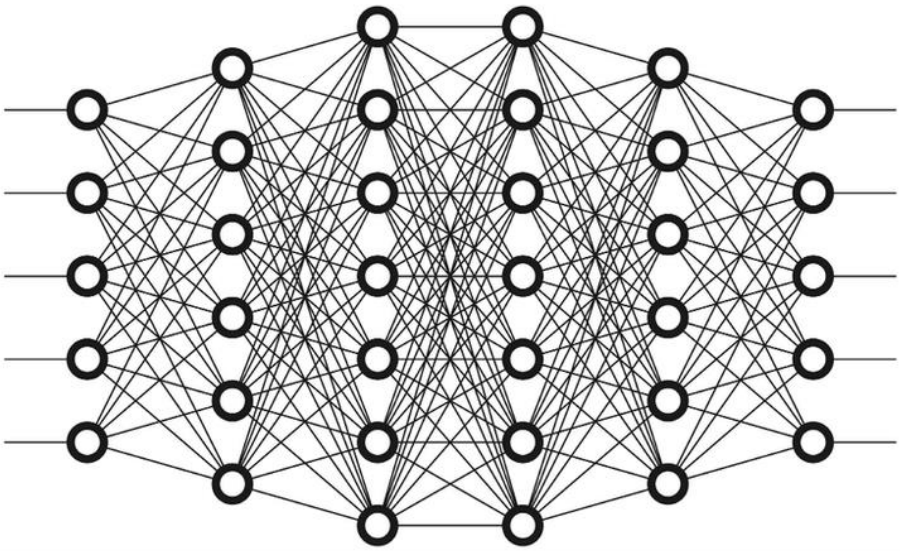
A tecnologia de aprendizado profundo reformulou o cenário de pesquisa de reconhecimento facial desde 2014, lançado pelos avanços dos métodos Deepfaces e DeepID. Desde então, técnicas de reconhecimento facial profundo, que aproveitam a arquitetura hierárquica para aprender a representação discriminativa de rostos, melhoraram drasticamente o desempenho de última geração e promoveram vários aplicativos bem-sucedidos no mundo real. O aprendizado profundo aplica várias camadas de processamento para aprender representações de dados com vários níveis de extração de recursos.

10.2 Deep Neural Network (DNN)

Deep Neural Network ou DNN (redes neurais profundas), é um conjunto de algoritmos modelados com inspiração no funcionamento do cérebro humano. Cada rede neural foi projetada para atuar como se fosse um neurônio, de forma que consiga reconhecer padrões, agrupando e analisando dados coletados no mundo real como por exemplo imagens, sons, textos ou séries temporais, sempre traduzindo-os em alguma coisa,

abaixo na Figura 10.2 podemos verificar a representação de como é a configuração de uma arquitetura de rede neural profunda.

Figura 10.2: Estrutura de uma rede neural profunda.



Fonte: *GRIGSBY, 2018*.

Dentro da DNN, as redes neurais são as partes centrais dos processamentos e compreensões dos dados analisados. Diversas arquiteturas moldadas com formas e objetivos diferentes se articulam para chegar em alguma conclusão definitiva com base nos dados apresentados a elas.

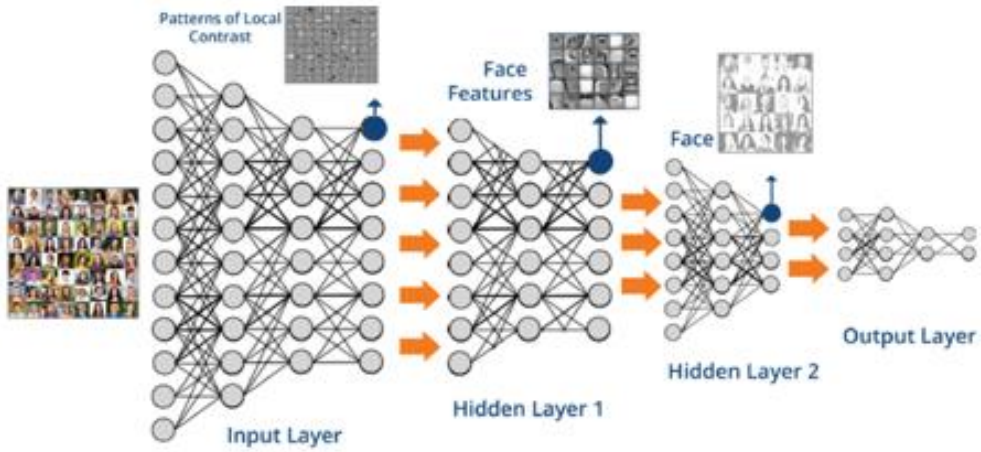
O conceito de redes neurais pode parecer coisa de um futuro distante, mas tudo isso já está presente no mundo atual. Empresas e organizações já empregam a dnn em seu cotidiano, automatizando tarefas até então manuais, como o reconhecimento de voz e a leitura e compreensão de imagens e vídeos, exemplos da aplicação de uma deep neural network compreendem diversas áreas do conhecimento e organizações em segmentos variados.

Um dos usos mais comuns para as redes neurais é o de reconhecimento e classificação de imagens. Esse método já é utilizado por organizações como o Facebook, que consegue reconhecer padrões faciais, sugerindo aos usuários quem está presente nas fotos postadas.

A compreensão de imagens pelos algoritmos também pode ser responsável por catalogar obras de arte. Dessa forma, é possível identificar o período de uma determinada pintura ou então capturar o estilo de uma obra, organizando um grande acervo ou aplicando-a em uma fotografia ou vídeo.

Na Figura 10.3, pode-se observar como é organizada a estrutura de uma DNN configurada para receber as imagens a serem analisadas e quais serão os processos a serem utilizados para assim conseguir chegar em um padrão de classificação bem definido e depois fazer a certificação da detecção facial quando receber novas imagens.

Figura 10.3: Configuração de uma rede neural profunda.



Fonte: *GRIGSBY, 2018*.

Ao observar a Figura 10.3, pode-se concluir que uma DNN, dependendo da quantidade de camadas feitas para verificar padrões das imagens, requer muito esforço computacional para fazer o treinamento, pois é um modelo matemático que consiste em várias camadas de elementos que executam cálculos paralelos. Inicialmente, essa arquitetura foi criada por analogia com os menores elementos computacionais do cérebro humano — neurônios. Os elementos computacionais mínimos de uma rede neural artificial também são chamados de neurônios.

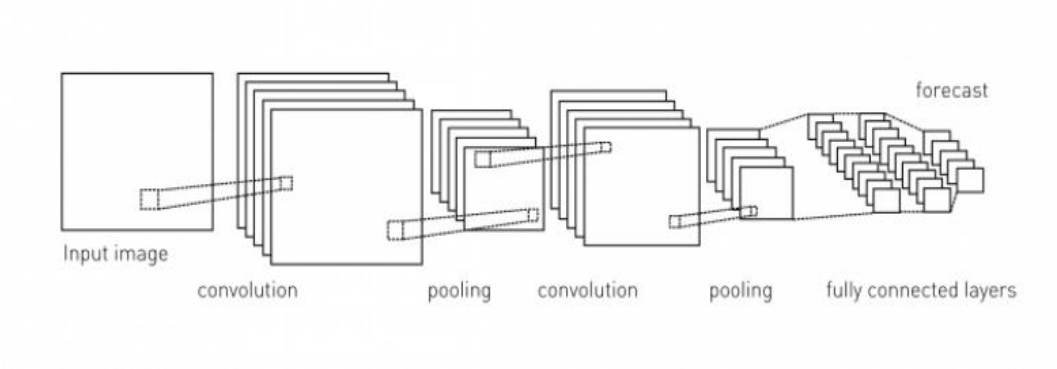
As redes neurais geralmente consistem de três ou mais camadas: a camada de entrada, a camada oculta (ou camadas) e a camada de saída conforme ilustrado na Figura 25; em alguns casos, as camadas de entrada e saída não são levadas em consideração e, em seguida, o número de camadas na rede é calculado pelo número de camadas ocultas.

Uma característica importante de uma rede neural é sua capacidade de aprender com os exemplos, isso é chamado de aprendizagem com um professor. A rede neural é treinada em um grande número de exemplos, consistindo em pares de entrada-saída (entrada e saída correspondentes). Nas tarefas de reconhecimento de objetos, esse par será a imagem de entrada e a etiqueta correspondente ao nome do objeto.

O treinamento da rede neural é um processo interativo que reduz o desvio da saída da rede de uma determinada “resposta do professor” a etiqueta correspondente a esta imagem. Esse processo é formado de etapas chamadas épocas de aprendizado (geralmente são de milhares), em cada uma das quais os pesos da rede neural são ajustados aos parâmetros das camadas ocultas da rede.

Após a conclusão do processo de treinamento, a qualidade da rede neural geralmente é boa o suficiente para concluir a tarefa para a qual foi treinada, embora muitas vezes seja impossível selecionar o conjunto ideal de parâmetros que reconheça idealmente todas as imagens. A seguir na Figura 10.4, a arquitetura geral da rede neural profunda para reconhecimento de imagens é mostrada, a imagem de entrada é representada como um conjunto de pixels ou pequenas seções da imagem (por exemplo, 5 por 5 pixels).

Figura 10.4: Diagrama de Rede Neural Profunda para detecção de imagens.



Fonte: *GRIGSBY, 2018*.

Normalmente, as redes neurais profundas são representadas de forma simplificada: como etapas de processamento, às vezes chamados de filtros, cada etapa é diferente do outro com uma série de características: o tamanho do campo receptivo, o tipo de características que a rede aprende a reconhecer nessa camada e o tipo de cálculo realizado em cada etapa.

As áreas de uso das redes neurais profundas, incluindo as redes convolucionais, não se limitam ao reconhecimento facial. Elas são amplamente utilizadas para reconhecimento de voz e sinais de áudio, para processar leituras de vários tipos de sensores ou para segmentar imagens complicadas de várias camadas (como mapas de satélite) ou imagens médicas (raios-x, imagens da Ressonância magnética funcional, veja aqui). Elas são aplicadas na área de segurança, no setor financeiro e muito mais.

A interação humana é predominantemente dependente do reconhecimento de uma grande variedade de padrões de vários tipos. Esses padrões consistem em facetas como a familiaridade de parentes, a voz de uma pessoa familiar, a aparência de objetos/indivíduos e assim por diante. A precisão do reconhecimento dos padrões acima mencionados dita diariamente a sobrevivência e tem sido um fator estimulante na evolução da complexidade social do ser humano e, assim, possibilitou nossa existência como uma espécie progressivamente avançada.

A capacidade inerente do cérebro humano de reconhecer padrões em uma maneira rápida e precisa tem motivado os cientistas a tentar emular esse comportamento usando vários técnicas e algoritmos. O Reconhecimento Facial é uma dessas habilidades cognitivas preeminentes que os pesquisadores têm se esforçado arduamente para emular usando uma infinidade de algoritmos intrincados e distintamente perceptivos metodologias.

Avançou em uma escala incrível nas últimas duas décadas e ofereceu um número de aplicações práticas que serviram imensamente em uma ampla gama de aplicações comerciais e de aplicação da lei definições. A operação fundamental dos sistemas depende da identificação efetiva de vários recursos-chave, como estrutura do nariz, posicionamento da cavidade ocular, proeminência da linha da mandíbula, testa e distância entre os olhos, forma de as maçãs do rosto e assim por diante. Produziu consistentemente um número atraente de inovações pioneiras, como o recente advento das Smart TVs Samsung, que incorporam a tecnologia utilizando a câmera embutida para que fornecem controle de acesso sem precedentes em termos de navegação de canal e navegação na web.

Deepfaces é o sistema de reconhecimento facial usado pelo Facebook para marcar imagens. Foi proposto por pesquisadores do Facebook AI Research (FAIR) no IEEE Computer Vision and Pattern Recognition Conference 2014 (CVPR). No reconhecimento facial moderno, existem 4 etapas: Detectar, Alinhar, Representar, Classificar e esta abordagem se concentra no alinhamento e representação de imagens faciais.

10.2.1 Alinhamento

Uma das etapas fundamentais para o bom funcionamento do DeepFace é o processo de alinhamento das imagens. Esse processo é responsável por ajustar a imagem de modo que o rosto esteja sempre na mesma posição e orientação, o que facilita a análise do algoritmo.

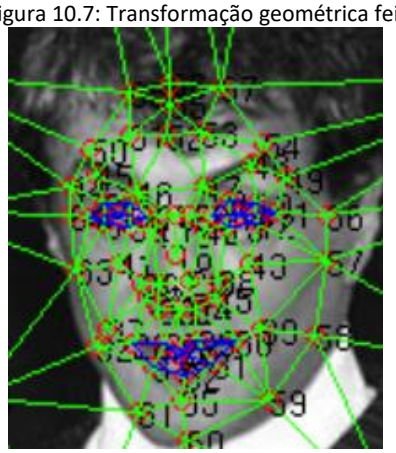
O processo de alinhamento do DeepFace é realizado em duas etapas: a detecção dos pontos faciais e o alinhamento propriamente dito. Na primeira etapa, o algoritmo utiliza uma rede neural para detectar os principais pontos faciais, como olhos, nariz, boca e queixo. Esses pontos são usados como referência para ajustar a imagem.

Figura 10.5: Detectando a face utilizando 6 pontos fiduciais.

Fonte: TAIGMAN; YANG; RANZATO; WOLF, 2014.

Na segunda etapa, o algoritmo realiza o alinhamento propriamente dito da imagem. Isso é feito por meio de uma transformação geométrica, que ajusta a imagem de modo que os pontos faciais detectados fiquem sempre na mesma posição e orientação. Essa transformação pode ser realizada por meio de rotação, escala e translação da imagem.

Figura 10.7: Transformação geométrica feita na face.

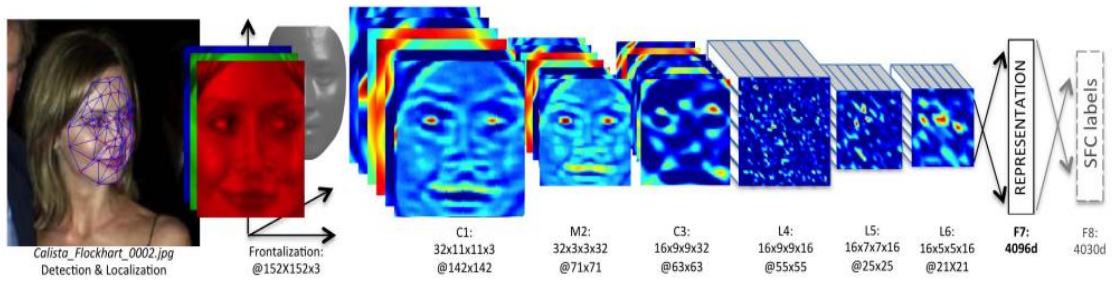


Fonte: TAIGMAN; YANG; RANZATO; WOLF, 2014.

Depois do processo de alinhamento, o DeepFace está pronto para realizar a análise da imagem e tentar identificar o rosto presente. O algoritmo utiliza uma rede neural para comparar a imagem com um conjunto de imagens de rostos já conhecidos e tentar encontrar uma correspondência. Se a correspondência for encontrada com uma imagem de referência, o algoritmo conclui que o rosto presente na imagem é o mesmo da imagem de referência. Caso contrário, o algoritmo conclui que o rosto é desconhecido.

Em resumo, o processo de alinhamento do DeepFace é fundamental para garantir a precisão do algoritmo. Ele consiste na detecção dos pontos faciais e na realização de uma transformação geométrica na imagem, de modo a ajustá-la de acordo com esses pontos. Depois do processo de alinhamento, o algoritmo está pronto para realizar a análise da imagem e tentar identificar o rosto presente na imagem.

Figura 10.9: Esboço da arquitetura DeepFace

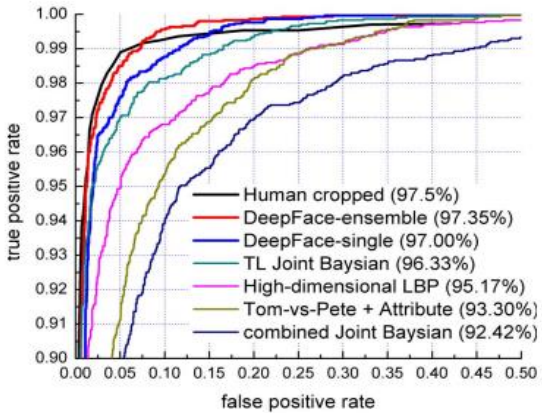


Fonte: TAIGMAN; YANG; RANZATO; WOLF, 2014.

O algoritmo DeepFace é um algoritmo de reconhecimento facial que utiliza uma rede neural profunda (DNN) para extrair características relevantes das imagens de rosto e realizar o reconhecimento. A DNN é composta por seis camadas, cada uma delas com uma função específica no processo de reconhecimento. As camadas L4, L5 e L6 são conectadas localmente, ou seja, cada unidade de saída está conectada a um patch específico da entrada. Isso é necessário porque as imagens de entrada estão alinhadas e, portanto, diferentes regiões do rosto têm estatísticas locais diferentes. As camadas F7 e F8 são camadas totalmente conectadas, ou seja, cada unidade de saída está conectada a todas as unidades de entrada. Elas são responsáveis por capturar correlações entre características de diferentes partes do rosto.

A saída da primeira camada totalmente conectada (F7) é usada como o vetor de representação de rosto, enquanto a saída da última camada totalmente conectada (F8) é alimentada a um K-way softmax, que produz uma distribuição sobre as classes de rótulos. O objetivo do treinamento é maximizar a probabilidade para a classe correta (identidade da face) minimizando a perda de entropia cruzada para cada amostra de treinamento. As características produzidas pela DNN do DeepFace são muito escassas e invariantes à escala e à rotação, o que é importante para garantir que o algoritmo funcione bem mesmo em condições adversas. O algoritmo foi treinado com um grande conjunto de dados e atingiu um desempenho excepcional em diversos conjuntos de dados de teste.

Figura 10.10: As curvas ROC no conjunto de dados LFW. Melhor visualizado em cores, onde mostra a taxa de assertividade dos algoritmos ao reconhecer as imagens.



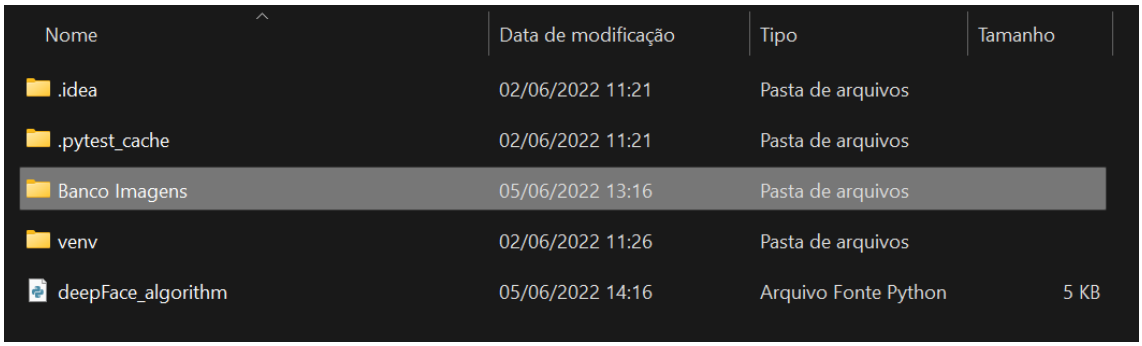
Fonte: TAIGMAN; YANG; RANZATO; WOLF, 2014.

Como pode-se observar, o algoritmo Deepfaces, tem uma arquitetura de rede neural profunda muito bem elaborada, onde há n quantidades de camadas responsáveis para executar processos de filtragem nas imagens recebidas onde as mesmas passam por processos de detecção, alinhamento, representação, classificação, e mesmo ao receber uma imagem em 2D faz a conversão para 3D e vice e versa. É realmente uma arquitetura sofisticada, onde análises foram feitas em cima das dificuldades dos algoritmos anteriores onde fez com que o DeepFace conseguisse melhorar ou até mesmo fazer a correção desses problemas para aumentar o desempenho ao fazer o reconhecimento em grande escala. Conforme a Figura 10.10 demonstra, a taxa de assertividade está muito próxima da capacidade humana de fazer reconhecimento.

10.3 Implementação do método de detecção facial do Deepfaces

Para fazer a implantação do método de Deepfaces, é preciso fazer a criação dos diretórios que serão utilizados para a verificação das imagens, porém, os scripts a seguir irá gerar algumas pastas automaticamente a única pasta que será preciso criar, será a venv onde ficará configurado todo o ambiente virtual para realização dos testes do algoritmo.

Figura 10.11. Diretórios para teste do método.



Para esse método, foi utilizado o seguinte ambiente e ferramentas, Visual Studio Code que é a IDE para a programação do código, o Python 3.10.4 que é a versão mais recente do python, biblioteca DeepFace 0.0.73 para fazer a verificação das imagens, entre outras.

Por padrão, a biblioteca DeepFace contém alguns modelos de treinamentos já prontos onde algumas imagens já foram processadas e feito o aprendizado da DNN que

está embutida dentro da biblioteca, que são esses: "opencv", "ssd", "mtcnn", "dlib", "retinaface". Na nossa abordagem, estará sendo utilizado o modelo retinaface que segundo a documentação é o modelo que obteve a maior precisão no assunto de detecção facial.

10.3.1 Explicando o código

Código 10.1: Importação das bibliotecas no python.

```
from genericpath import exists
from deepface import DeepFace
import os
import cv2
import numpy as np
from sklearn.metrics import confusion_matrix
from time import time
import kaggle
from kaggle.api.kaggle_api_extended import KaggleApi
from PIL import Image
from sklearn.metrics import accuracy_score
from sklearn import metrics
import seaborn as sns
from matplotlib import pyplot as plt
```

Ao iniciar a programação do código, é preciso fazer a importação da biblioteca deepface onde será utilizado o método especifico chamado DeepFace, que ficará responsável por fazer as análises das imagens fornecida, e também será preciso utilizar a cv2 do Opencv, que ficará responsável por fazer a leitura das imagens e adiciona-las nas variáveis para ser utilizada no método do DeepFaces, também foi feito a importação das bibliotecas os para aplicar comandos do sistema operacional ao algoritmo, conjunto de módulos da biblioteca sklearn para obter os resultados das análises, seaborn para colocar informações visual dentro dos gráficos gerados pela biblioteca matplotlib, a time para capturar os tempos de processamento e a kaggle para fazer a conversação com o site kaggle utilizando sua api para fazer o download do dataset utilizado para análise, também foi utilizado a Pil para fazer manipulações com a imagem e por último a numpy para fazer a transformação em matrizes das imagens.

Código 10.2: Função para fazer o download do dataset.

```
def downloadDataset():
    api = KaggleApi()
    api.authenticate()

    if not os.path.exists('Banco Imagens'):
        os.makedirs('Banco Imagens')
```

```
    kaggle.api.dataset_download_files('asacxyz/ic-fatecitu', path='Banco
Imagens', unzip=True)

    caminho = [os.path.join('Banco Imagens', f) for f in os.listdir('Banco
Imagens')]

    for caminhoImagem in caminho:
        if ".jpeg" in caminhoImagem:
            nomeArquivo = os.path.splitext(caminhoImagem)[0]
            Image.open(caminhoImagem).convert('RGB').save(nomeArquivo + '.' +
'jpg')
            os.remove(caminhoImagem)
```

Foi criado uma função com o nome downloadDataset, que será responsável por fazer o download do dataset escolhido na plataforma Kaggle por meio de sua api, foi criado uma variável chamada api, que ficará responsável por receber a conexão da api e depois essa variável irá chamar a função authenticate que fará a autenticação da api com a plataforma, para isso é preciso ter uma conta criada no kaggle para fazer o download.

Abaixo, foi criado uma condição para que se caso não existir o diretório chamado Banco Imagens que ele seja criado utilizando a função os.mkdir logo abaixo passando o nome do diretório a ser criado. Depois é utilizado o método kaggle.api.dataset_download_files para fazer o download do dataset passando como parâmetro o perfil do usuário da plataforma kaggle onde está localizado o dataset, seguido do local onde será armazenado e pedindo para que logo após o download o arquivo seja descompactado.

Depois foi criado uma variável chamada caminho onde receberá o caminho do diretório Banco Imagens pela função os.path.join em seguida listando todos os arquivos que estão armazenados dentro desse diretório usando um laço for em seguida utilizando a função os.listdir, com isso todas as imagens terão o caminho absoluto C:\Deepfaces v2\Banco Imagens\1-treinamento1.jpg.

Mais abaixo, foi criado um laço for para percorrer cada imagem dentro do diretório e criado uma nova condição para que se os arquivos terminarem com a extensão .jpeg, renomear eles para .jpg utilizando a função Image.open para abrir a imagem depois .convert para converter a imagem para RGB e salvar a imagem utilizando o .save passando o nome da imagem e modificando a extensão para .jpg, e logo após utilizando a função os.remove para remover a imagem anterior do Banco Imagens mantendo somente a imagem com extensão .jpg dentro do diretório.

Código 10.3: Função para preparar o dataset.

```
def prepararDataset(origem):

    diretorio = os.listdir(origem)

    cont = 1
```

```
    for caminhoImagem in diretorio:
        if ".jpg" in caminhoImagem:
            imagem = cv2.imread(caminhoImagem)
            img = cv2.resize(imagem, (600, 600))
            cv2.imwrite(caminhoImagem, img)
            os.rename(caminhoImagem, str(cont) + '-treinamento' + str(cont) +
'.jpg')
            cont += 1
```

Também foi criado uma função para fazer a preparação das imagens para análise de detecção, essa função recebe a origem como parâmetro para fazer a preparação. Foi criado uma variável diretório que receberá os valores da função os.listdir que por sua vez, receberá a origem para fazer a listagem dos arquivos que contem dentro do diretório Banco Imagens, e em seguida foi criado uma variável chamada cont para adicionar o id as imagens de acordo com a quantidade de imagens dentro do diretório, nesse caso cont iniciasse de 1.

Depois foi criado um laço for para percorrer cada imagem dentro do diretório e a condição se caso tiver .jpg como extensão do arquivo, o arquivo será lido pela função cv2.imread e armazenar essa imagem na variável imagem e depois foi criada a variável img que receberá a imagem redimensionada pela função cv2.resize que receberá a imagem como parâmetro e o redimensionamento será feito na escala de 600x600.

Depois é feita a gravação dessa imagem utilizando o método cv2.imwrite passando a imagem antiga e a nova imagem como parâmetro e em seguida é utilizado o método os.rename passando a imagem antiga como parâmetro e o nome da imagem nova, nesse caso a imagem ficará com o número cont no início e depois o nome treinamento seguindo novamente do número cont para ficar nesse formato, 1-treinamento1.jpg, e depois o cont recebe o valor mais 1 para fazer a contagem de todas as imagens dentro do diretório.

Código 10.4: Função para filtrar o dataset.

```
def filtrarDataset(origem):
    y_train_verdadeiro = []
    diretorio = os.listdir(origem)
    for imagem in diretorio:
        nome = imagem.split('-')[0]
        if int(nome) <= 400:
            y_train_verdadeiro.append(0)
        elif int(nome) > 400:
            y_train_verdadeiro.append(1)
    return y_train_verdadeiro
```

Essa Função serve para passar os valores verdadeiros para cada imagens e depois fazer a comparação com os valores preditos pelo algoritmo, no caso o diretório tem 400 imagens de animais e 600 imagens de face, e é preciso passar esses valores reais para fazer a análise com os valores que o algoritmo irá identificar, nesse caso foi criado um

array chamado y_train_verdadeiro que irá receber os valores 1 ou 0 se caso as imagens forem 1 positivo e 0 negativo, é feita a listagem do diretório para pegar todos os arquivos dentro dele e feito um laço for para acessar um a um e verificar seus ids, se ele tiver o id menor que o valor 400 será adicionado no array o valor 0 para essas imagens e se ele tiver o valor acima de 400 será adicionado ao array pela função .append o valor 1 declarando que essa imagem é positiva. E por fim retorna o array para ser utilizado por outra função mais tarde.

Código 10.5: Função para carregar as imagens em memória.

```
def carregarImagem(origem):
    diretorio = os.listdir(origem)
    x_train = []
    for imagem in diretorio:
        img = cv2.imread(imagem)
        x_train.append(img)
    return x_train
```

Essa função, serve para fazer o carregamento das imagens para a memória do computador, recebe como parâmetro a origem, onde tem uma variável que irá listar os arquivos dentro do diretório. Também foi criado um array chamado x_train que recebera as imagens lidas pela função cv2.imread e será adicionado no array pela função .append. E por fim, retornara o array para ser utilizados por outras funções dentro do algoritmo.

Código 10.6: Função para detectar as faces das imagens.

```
def detectarDeepFace(x_train):
    y_train_predict = []
    tempos = []
    diretorio = x_train
    for imagem in diretorio:
        inicio = time()
        detectors = ["opencv", "ssd", "mtcnn", "dlib", "retinaface"]
        face = DeepFace.detectFace(imagem, detector_backend = detectors[4],
enforce_detection = False)
        if face.all() != 0:
            y_train_predict.append(1)

        else:
            y_train_predict.append(0)
        fim = time()
        tempos.append(fim - inicio)
    return y_train_predict, tempos
```

Essa função foi criada para fazer a detecção das faces dentro das imagens, recebe a matriz contendo as imagens carregada em memória como parâmetro. Foi criado um

array onde será armazenado os dados preditos pelo algoritmo, também foi criado um novo array chamado tempos onde será adicionado os tempos de processamento de cada imagem.

A seguir, é criado uma variável que recebera o array x_train, e criado um laço for para percorrer cada imagem dentro desse array, após é criado uma variável que recebera do método time() o tempo inicial do processamento da imagem. Em Seguida é criado uma lista chamada detector que irá passar para o método detectFace qual será o modelo de treinamento de detecção facial que será utilizado para fazer as analises nas imagens, assim foi criado uma variável chamada face que receberá o resultado do método que recebe a imagem, o modelo de detecção e o parâmetro enforce_detection como false que é responsável por não deixar o código quebrar ao receber imagens de animais ou qualquer outra imagem que não tenha uma face humana, assim o método ao receber esses tipos de imagens não irá parar repentinamente e irá fazer a análise das imagens.

A seguir, é criado uma condição para que se todos os valores que a variável face receber forem maiores que 0, significa que encontrou uma face e assim irá adicionar o número 1 no array de predição e se caso ele receber com todos os elementos da matriz valor igual a 0, significa que não encontrou nenhuma face na imagem e ele colocará 0 no array.

Por fim, é criado uma variável chamada fim que receberá o tempo final do processamento e adicionar o tempo no array tempos. Depois é retornado os arrays y_train_predict e tempos para usar em outras funções.

Código 10.7: Função para mostrar os resultados do algoritmo.

```
def resultados(y_train_verdadeiro, y_train_predict, tempos, nome):
    matriz = confusion_matrix(y_train_verdadeiro, y_train_predict)
    falsoNegativo, falsoPositivo, verdadeiroNegativo, verdadeiroPositivo =
matriz.ravel()
    matrizConfusao = np.array([[falsoNegativo, falsoPositivo],
[verdadeiroNegativo, verdadeiroPositivo]])
    sns.heatmap(matrizConfusao, annot=True, cmap='YlGnBu', fmt='g')
    plt.title('Matriz de Confusão do ' + nome)
    plt.ylabel('Real')
    plt.xlabel('Predito')
    plt.show()

    fpr, tpr, thresholds = metrics.roc_curve(y_train_verdadeiro,
y_train_predict)
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve ' + nome)
    plt.show()

    maiorTempo = max(tempos)
    menorTempo = min(tempos)
```

```
    mediaTempo = sum(tempos) / len(tempos)
    tempoTotal = sum(tempos)
    plt.plot(tempos)
    plt.title('Tempo de detecção ' + nome)
    plt.ylabel('Tempo')
    plt.xlabel('Imagens')
    plt.show()

    plt.bar(['Maior Tempo', 'Menor Tempo', 'Media de Tempo', 'Tempo Total'],
[maiorTempo, menorTempo, mediaTempo, tempoTotal])
    plt.title('Tempo de detecção ' + nome)
    plt.ylabel('Tempo')
    plt.xlabel('Imagens')
    plt.show()

    accuracy = accuracy_score(y_train_verdadeiro, y_train_predict)
    print('Accuracy ' + nome + ' : %.2f' % accuracy)
```

Foi criado uma função chamada resultados, onde receberá os arrays y_train_verdadeiro, y_train_predict e tempos como parâmetro e também a variável nome onde será colocado no nome DeepFace para fazer a apresentação nos gráficos.

Após é criado uma variável matriz que irá receber a matriz de confusão feita entre os arrays verdadeiro e predict e depois através do método ravel será colocado cada valor da matriz de confusão dentro de uma variável, no caso foi criado 4 variáveis que receberão os valores de falso positivo, falso negativo e os valores de verdadeiro positivo e verdadeiro negativo. Assim depois de fazer essa seleção dos dados é feita a criação da matriz de confusão de forma organizada passando para a função np.array as variáveis com os dados obtidos.

Utilizando a função heatmap da biblioteca seaborn, cria-se a coloração para obter o melhor entendimento da análise gráfica e utilizando as funções do matplotlib é passado o nome que será utilizado no título do gráfico e as colunas de acordo com os valores que estão sendo recebidos e pedindo para mostrar na tela com a função .show.

Após esse processo é criado 3 variáveis para receber o valor do método metrics.roc_curve que irá receber como parâmetros os arrays verdadeiro e predict, fazendo assim a criação do gráfico da curva roc, e utilizando as funções do matplotlib para mostrar os respectivos valores na tela do gráfico

A seguir, é criado a variável maior tempo que receberá o maior tempo do array tempos, a variável menor tempo que receberá o menor tempo, a variável media tempo que fará o cálculo da soma de todos os tempos e dividir pela quantidade de tempos que tem dentro do array e a variável tempo total que recebera a soma de todos os tempos processados das imagens. Também é criado um gráfico que mostrará todas as imagens na tela.

Depois é criado uma variável chamada accuracy que receberá o valor da acurácia final do algoritmo comparado com os dois arrays recebidos e utilizando a função print para mostrar o resultado na tela.

Código 10.8: Função para mostrar os resultados do algoritmo.

```
print('Fazendo o download do dataset .......')
downloadDataset()
print('Download concluído')
origem = os.chdir('Banco Imagens')
print('Preparando o dataset .......')
prepararDataset(origem)
print('Dataset pronto')

nome = 'Detecção Facial do DeepFace'
y_train_verdadeiro = filtrarDataset(origem)
x_train = carregarImagem(origem)
print('DeepFace')
print('Iniciando detecção...')
y_train_predict, tempos = detectarDeepFace(x_train)
print('Detecção finalizada')
print('Resultados:')
resultados(y_train_verdadeiro, y_train_predict, tempos, nome)
```

Após a criação de todas as funções anteriores é preciso fazer a chamada delas para o funcionamento do algoritmo, assim é utilizado a função print para mostrar na tela que está sendo feito o download das imagens através da função downloadDataset e após o termino do download é impressa na tela a mensagem de que o download foi concluído.

É passado para a variável origem o local onde estão armazenadas as imagens, impresso na tela que o dataset está sendo preparado e chamando a função prepararDataset é passado a origem onde estão as imagens. Assim que terminar a preparação a mensagem Dataset pronto é impressa na tela.

Foi criado uma variável e passado uma string para ela escrito Detecção Facial do DeepFace para mostrar no título dos gráficos. Depois é chamado a função para filtrar o dataset e armazenar os dados na variável y_train_verdadeiro e criado a variável x_train para armazenar as imagens na memória do computador.

Após o processo, imprime Deepfaces na tela, imprime que a detecção está iniciando e cria as variáveis para receber os dados da função detectarDeepFace passando a variável x_train. Após esse processo é impresso na tela que a detecção foi finalizada e imprime os resultados chamando a função resultados passando como parâmetros os arrays que contém os dados analisados e o nome que recebe a string.

Com a finalização desse processo, é apresentado os resultados a seguir:

Figura 10.12: Matriz de Confusão do Deepfaces.

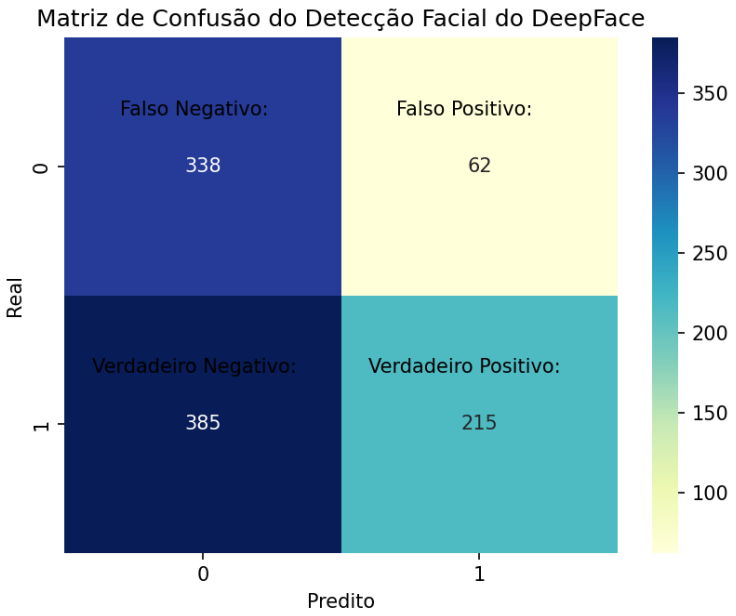


Figura 10.13: Curva roc do Deepfaces.

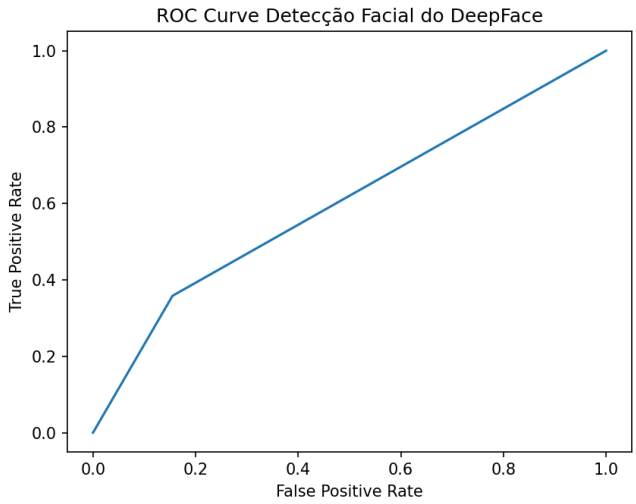
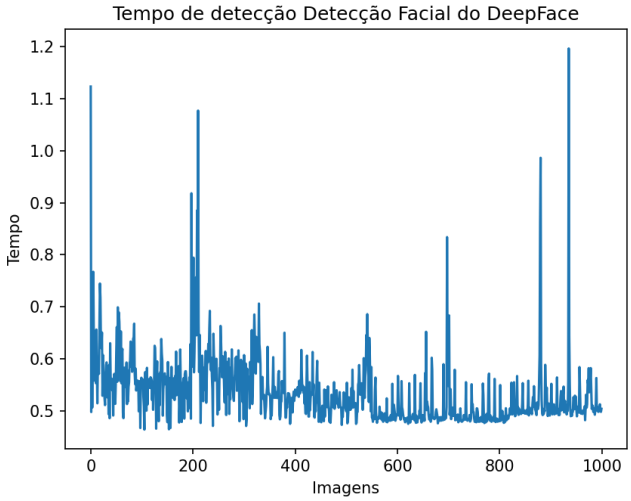


Figura 10.14: Análise gráfica do tempo médio de processamento de cada imagem.



Na imagem acima, pode-se ver qual foi o tempo processado por imagem que o método de detecção do deepface levou para fazer.

Figura 10.15: Tempo levado pelo processamento das imagens.

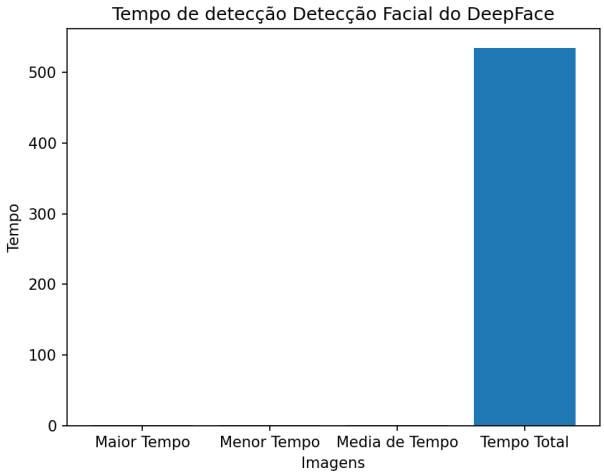


Figura 10.16: Accuracia de detecção Facial do DeepFace.

```
Resultados:
Accuracy Detecção Facial do DeepFace : 0.55
```

Como pode-ver, o DeepFace é um algoritmo focado em reconhecimento facial, porem por ter um método que faz detecção facial, ele tem vários modelos para ser utilizado, como Opencv que por traz roda o modelo do Haar-Cascade para fazer as analises, o Dlib que por traz roda o CNN e o que foi utilizado para esse teste é o Retinaface.

O Retinaface é um modelo que utiliza CNN para fazer as análises nas imagens tem uma característica específica que é determinante para obter alta acurácia: é preciso que as imagens entregue a ele tenha um posicionamento único da face, de preferência na posição frontal e sem a utilização de óculos que são fatores que pode atrapalhar o algoritmo na sua análise.

Outra característica importante é a iluminação, onde ela tem que realçar os contornos da face a ser analisada

Por fim, a definição da imagem. É necessário que as imagens enviadas estejam com uma boa qualidade de visualização para ajudar na precisão da detecção.

O Dataset utilizado continha muitas faces em vários tipos de posição, iluminação e nitidez, fazendo com que o algoritmo se perdesse nas suas referências e aumentando a quantidade de falso negativo.

Com isso, finalizamos a explicação do método/algoritmo DeepFaces e sua forma de implementação e aproveitamos para mostrar alguns resultados de sua aplicação sob uma base de imagens.

No próximo capítulo vamos ver o último método/algoritmo selecionado em virtude de sua importância, também baseado em redes neurais, conhecido como YOLO que é um acrônico de *You Only Look Once* (Você só precisa observar/ver uma vez).

11

YOLO

Neste capítulo, será abordado o algoritmo YOLO (*You Only Look Once*), um detector de objetos em tempo real baseado em redes neurais. Serão explicadas a definição e as etapas que formam o algoritmo, bem como as versões que ele foi recebendo desde o ano de seu lançamento. Além disso, haverá a implementação de YOLO para reconhecer faces em imagens e vídeos. Por fim, o desempenho do algoritmo é avaliado, testando sua acurácia e o tempo de processamento utilizando a mesma base de imagens utilizadas para testar os outros algoritmos até aqui apresentados.

11.1 Contextualização

Desde os primeiros trabalhos de reconhecimento facial por computadores, que ocorreram nos anos 1970[1], a tecnologia evoluiu a passos largos. Muitas técnicas de processamento de imagem foram criadas com o objetivo de melhorar a detecção de rostos e objetos, gerando avanços importantes na área de Inteligência Artificial. Na década de 2000, descritores de recursos foram criados com o objetivo de reduzir o tempo de processamento sem perder a eficácia de extrair as características da imagem. Além disso, com a evolução computacional, foi possível aplicar em dispositivos móveis algoritmos classificadores de objetos, tornando acessível ao público utilizar soluções baseadas em *Machine Learning*, como a possibilidade de câmeras identificarem rostos e até mesmo *smartphones* pesquisarem em mecanismos de buscas com base em fotos tiradas pelo usuário.

No entanto, um dos importantes marcos em reconhecimento facial foi a utilização de redes neurais para a identificação de faces. Antes desse paradigma, as etapas que envolviam a abordagem tradicional de detecção de objetos consistiam em extrair as características da imagem por meio de um descritor de recursos (por ex.: HOG, SIFT ou Haar-like). Geralmente, o espaço de cores da foto de entrada é reduzido para escalas de cinza antes do descritor de recursos ser aplicado. Após a obtenção do vetor de características, essa estrutura de dados

[1] Ver, por exemplo, Goldstein, Harmon e Lesk (1971) que propôs identificar rostos através de 22 recursos classificados como “subjetivos” pelos autores. No artigo, as características adotadas pelos pesquisadores eram as regiões dos olhos, orelhas, nariz, entre outros componentes faciais. Uma abordagem semelhante foi apresentada por Kanade (1973), que fez um sistema de reconhecimento facial que extraía características-chave dos rostos para identificá-los.

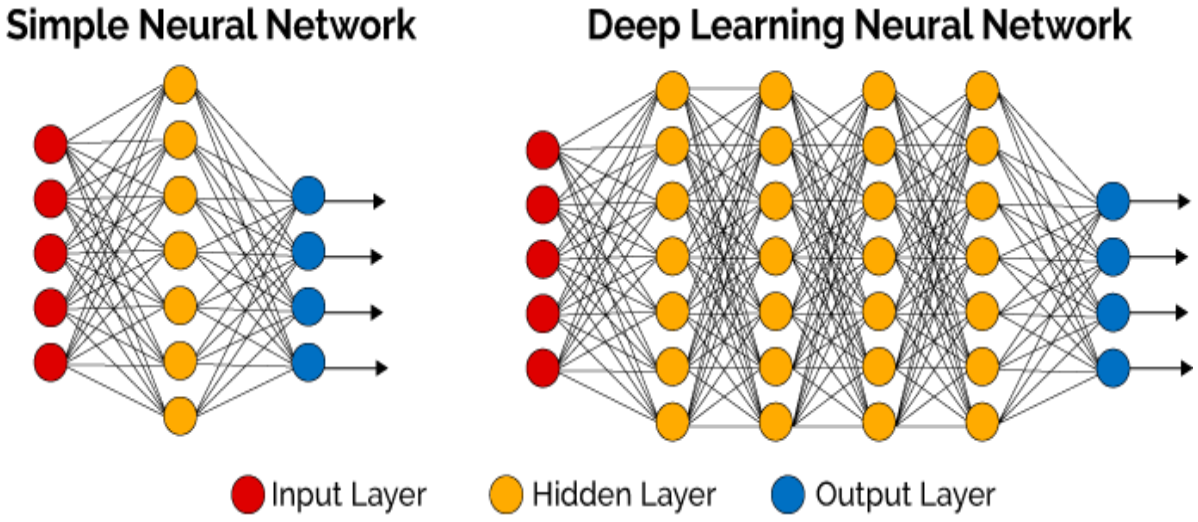
é utilizada em um algoritmo de *Machine Learning* para treiná-lo a identificar um objeto (GEORGIOU *et al*, 2019). Este modelo de aprendizado de máquina é comumente chamado de algoritmo classificador (por ex.: SVM), sendo responsável por reconhecer uma face ou objeto.

Com redes neurais, a técnica de utilizar um par de algoritmos (descritor de recursos e classificador) é descartada, focando somente no treinamento do modelo para identificar objetos. Antes de explicar essa importante característica, é necessário mostrar a definição de redes neurais artificiais:

> Uma rede neural é um processador maciçamente paralelamente distribuído constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso. Ela se assemelha ao cérebro em dois aspectos:
>
> 1. O conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem.
> 2. Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido. (HAYKIN, 2008)

Os modelos de redes neurais são algoritmos de *Machine Learning* que buscam simular o funcionamento dos neurônios biológicos para obter conhecimento. Quando há várias camadas intermediárias de neurônios, a rede é classificada como algoritmo de aprendizagem profunda (*Deep Learning*). A figura abaixo ilustra as diferenças entre uma rede neural simples e uma rede neural profunda.

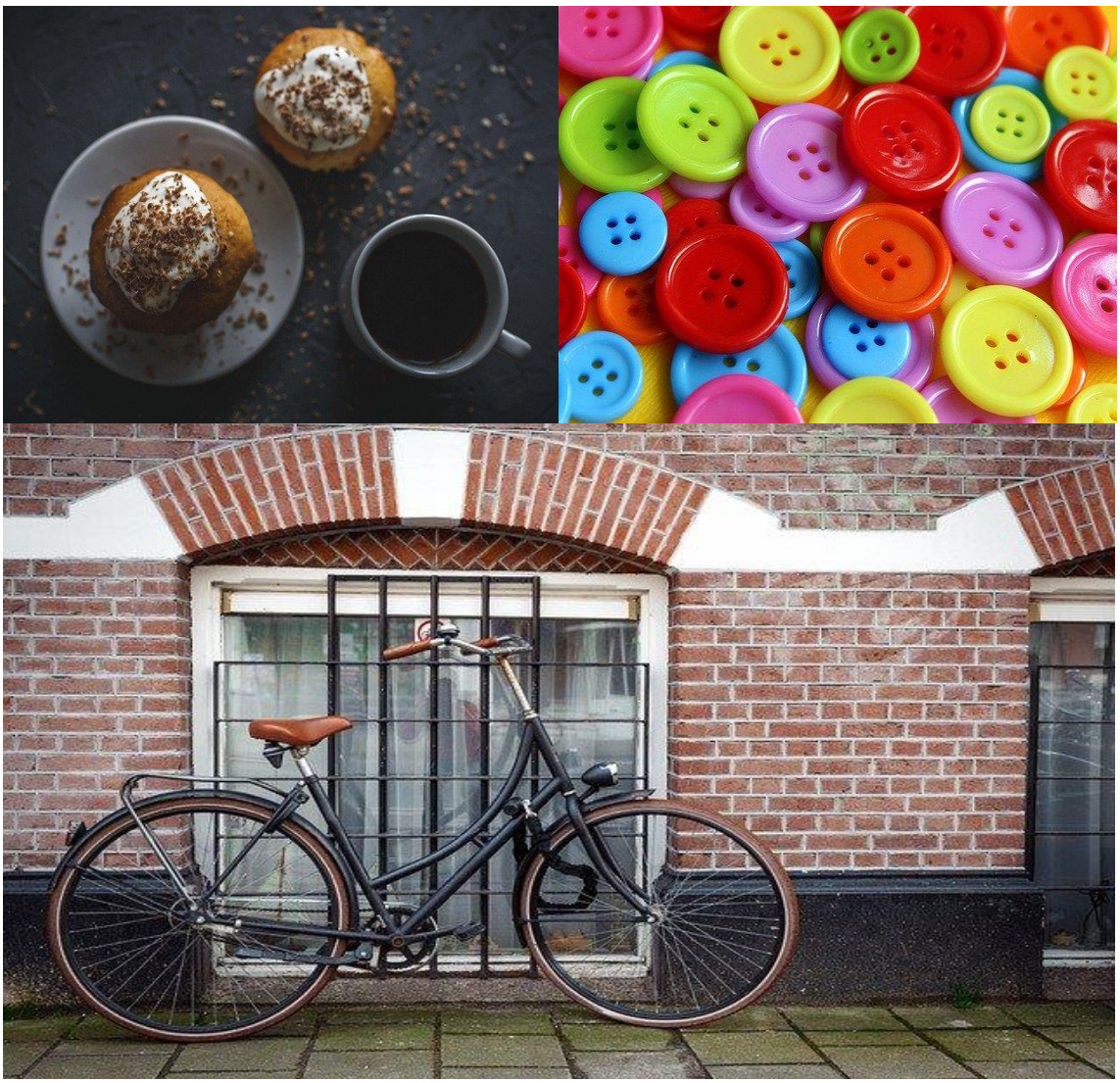
Figura 11.1: Comparação entre redes neurais simples e de aprendizagem profunda. Quanto maior o número de camadas intermediárias (ou ocultas) de neurônios, mais complexa se torna a rede neural.



Fonte: Deep Learning Book (2021)

Algoritmos classificadores baseados em *Deep Learning* conseguem superar a tarefa de extrair bordas em imagens. Eles são capazes de obter mais informações detalhadas. Por exemplo: em uma aplicação que busca detectar rodas de veículos, o modelo classificador baseado em *Deep Learning* não somente identifica o formato circular destes objetos, mas os distinguem de outros parecidos, como bolos, pratos e botões (Figura 11.2).

Figura 11.2: Algoritmos de Deep Learning conseguem não apenas detectar bordas, mas distinguir objetos semelhantes. Nas imagens abaixo, os objetos de destaque são circulares. No entanto, estes modelos de redes neurais extraem propriedades que os permitem diferenciar rodas de botões e pratos.



Fonte: Pixabay

Levando em consideração os atributos que definem redes neurais, será explicado o detector de objetos YOLO (*You Only Look Once*), um algoritmo de rápido processamento

lançado em 2016. Ademais, YOLO vem recebendo importantes atualizações, tanto dos seus criadores quanto de equipes de desenvolvimento.

11.2 YOLO (*You Only Look Once*)

Lançado em 2016 por Redmon *et al* (2016), YOLO (*You Only Look Once*) é um algoritmo detector de objetos. Sua principal função não é apenas reconhecer qual objeto está presente na imagem (como ocorre em algoritmos classificadores), mas também identificar em qual posição ele está localizado. Em outras palavras, YOLO é capaz de responder duas perguntas: quais objetos estão na imagem? Onde eles estão posicionados?

A detecção é formada por dois processos: a classificação e a localização. Algoritmos classificadores são capazes de atribuir um rótulo a um conjunto de recursos extraídos da imagem (FORSYTH; PONCE, 2012). Já os localizadores distinguem o objeto do pano de fundo, e identificam as relações espaciais entre os objetos na imagem (BLASCHKO; LAMPERT, 2018). YOLO é a combinação das características presentes nesses algoritmos, uma vez que utiliza redes neurais profundas para reconhecer os objetos. Além disso, esses dois processos juntos permitem identificar vários objetos diferentes em uma imagem, conforme mostrado na Figura 11.3.

Outra característica importante de YOLO é ser um detector de objetos em tempo real. Redmon *et al* (2016) fizeram testes que comprovaram um ótimo desempenho do algoritmo ao reconhecer objetos através de *webcams*. O tempo de processamento foi rápido, chegando a atingir um valor menor que 25 milissegundos de latência. Com base nos testes feitos com *webcams*, os autores perceberam que YOLO apresentou uma solução eficaz e de fácil usabilidade. A detecção ocorria independente da movimentação do objeto ou da forma como a sua aparência mudava no vídeo.

YOLO foi comparado com outras técnicas de identificação de objetos. Uma delas foi o Modelo Baseado em Peças Deformáveis (*Deformable parts models,* DPM), cujo funcionamento ocorre por meio da abordagem clássica de detecção de objetos. Ott e Everingham (2011) explicam que o reconhecimento de objetos em DPM ocorre através de caixas delimitadoras, após a imagem ser dividida em blocos para que um descritor de recursos extraia as características. Contudo, essas caixas molduram não apenas o objeto como um todo, mas também as suas partes.

Figura 11.3: Diferenças entre classificação, localização e detecção. Um algoritmo classificador apenas descreve quais objetos estão na imagem, enquanto os localizadores retornam onde eles estão localizados. A combinação dessas duas características resulta na detecção. Algoritmos detectores, como YOLO, localizam e mostram quais objetos estão na imagem.

Por ser uma rede neural profunda, YOLO apresenta uma maior acurácia que a DPM, uma vez que os membros de cada objeto são processados em uma única CNN (*Convolutional Neural Network*). Ou seja, todas as etapas feitas por uma DPM são realizadas de forma simultânea em YOLO, tornando a sua *perfomance* melhor e com menor tempo de processamento.

R-CNN e suas variantes também foram comparadas com YOLO. Criada por Girshick *et al* (2014), *Regions with* CNN (R-CNN) utiliza CNN e SVM para identificar objetos. A sua principal característica é encontrar regiões na imagem onde possam ter algum objeto, para depois extrair as suas características através de CNN. Dessa forma, R-CNN é uma técnica que

evita percorrer cada área da imagem para realizar tal procedimento, focando apenas em propostas de região onde devem ter objetos. Com a extração de recursos efetuada, o algoritmo SVM é utilizado para classificar os objetos.

YOLO, por sua vez, tem semelhanças com as etapas presentes em R-CNN. Inicialmente ocorre uma busca na imagem por possíveis regiões onde há objetos, Em seguida, as caixas delimitadoras geradas passam pela etapa de classificação, onde são analisados quais objetos estão na imagem. Quanto as diferenças, Redmon *et al* (2016) mostram que o seu algoritmo apresenta menor tempo de processamento do que R-CNN. Girshick (2015) explica que R-CNN requer um tempo elevado para detectar objetos: 47 segundos por imagem. Ademais, este modelo prevê 2000 regiões onde possa haver objetos, corroborando com este tempo. YOLO reduz o número de regiões para 98 por imagem.

Fast R-CNN e *Faster* R-CNN buscam diminuir a latência do modelo publicado por Girshick *et al* (2014), mas não chegam a ser algoritmos detectores em tempo real. Também houve pesquisas que buscaram acelerar a velocidade de processamento do DPM, como Yan *et al* (2014), Sadeghi e Forsyth (2014) e Felzenszwalb, Girshick e McAllester (2010). Foram desenvolvidas melhorias no desempenho de HOG, bem como a aplicação de *cascades* como descritor de recursos (YAN *et al*, 2014). O trabalho de Felzenszwalb, Girshick e McAllester (2010) é outra pesquisa que utiliza *cascades*. Contudo, apenas 30 Hz realmente executam em tempo real (SADEGHI; FORSYTH, 2014).

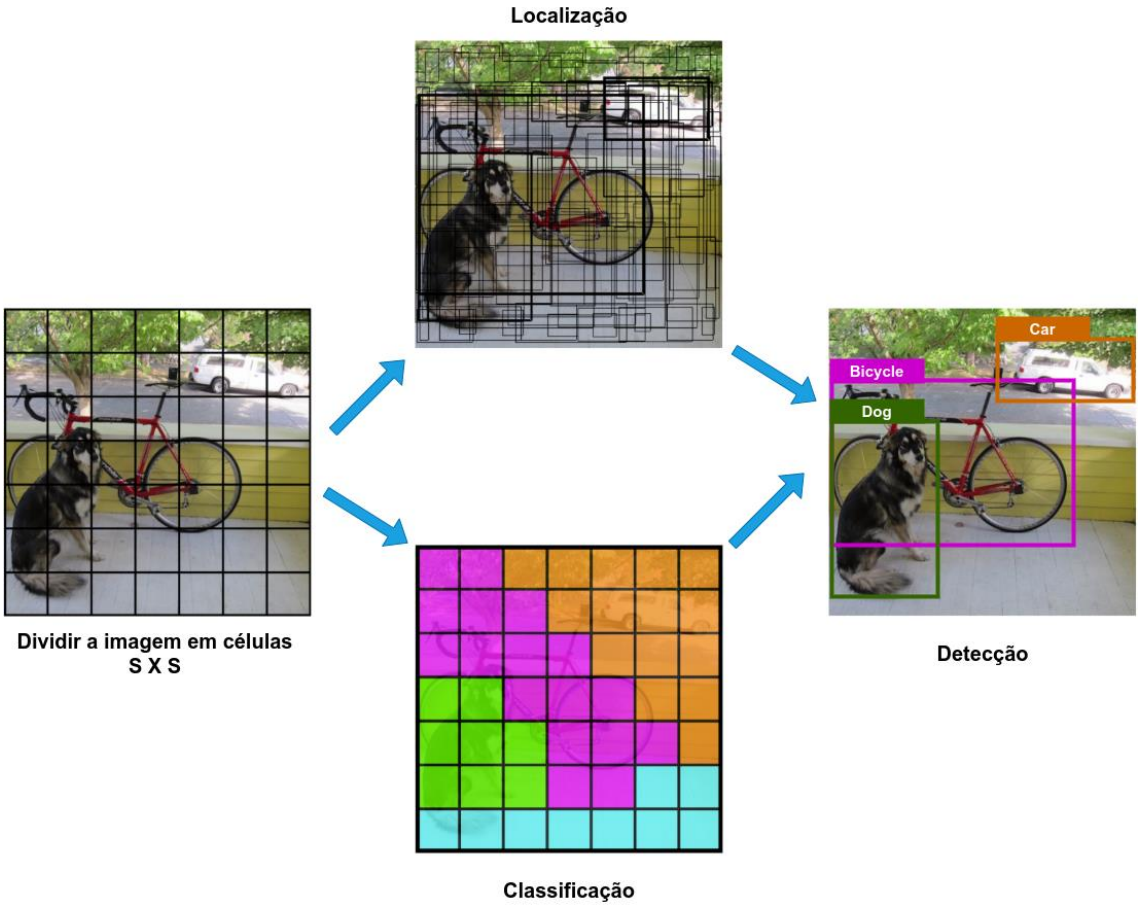
Diferente dos detectores faciais e *Deep MultiBox*, que lidam apenas com uma classe de objetos, YOLO é capaz de reconhecer várias simultaneamente, sendo, portanto, um detector geral de objetos. Por fim, o algoritmo desenvolvido por Redmon *et al* (2016) otimiza a detecção, ao contrário das abordagens realizadas por Sermanet *et al* (2013) Redmon e Angelova (2013), que melhoram a localização.

A maneira como ocorre a detecção de objetos em YOLO justifica o menor tempo de processamento. Através de CNN, a imagem é olhada apenas uma única vez (daí o seu nome *You Only Look Once*) para definir quais objetos estão presentes e onde eles estão localizados. A próxima seção irá mostrar o funcionamento do algoritmo.

11.3 Etapas do algoritmo

As etapas que formam YOLO são mostradas na Figura 11.4. Primeiramente, aplica-se uma grade de célula SxS sobre a imagem de entrada para depois ocorrer a classificação e a localização dos objetos. Conforme visto no diagrama abaixo, a definição e a posição dos objetos são obtidas simultaneamente, uma vez que a arquitetura por trás de YOLO consiste em utilizar uma CNN.

Figura 11.4: Etapas de YOLO. Inicialmente, aplica-se uma grade de células *SxS* sobre a imagem. Em seguida, a classificação e a localização dos objetos ocorrem de maneira simultânea. Por fim, a saída obtida é a detecção, com as caixas delimitadoras e os rótulos.



Fonte: Adaptado de Redmon *et al* (2016)

Cada célula gerada irá prever N caixas delimitadoras e os pontos de confiança para elas. Esses pontos serão utilizados para definir se as caixas molduram um objeto, Ou seja, quanto maior a probabilidade de uma caixa possuir um objeto, maior será a pontuação de confiança. Esse valor é representado pela espessura das caixas, conforme a Figura 11.5. Por fim, a confiança de uma caixa é dada por:

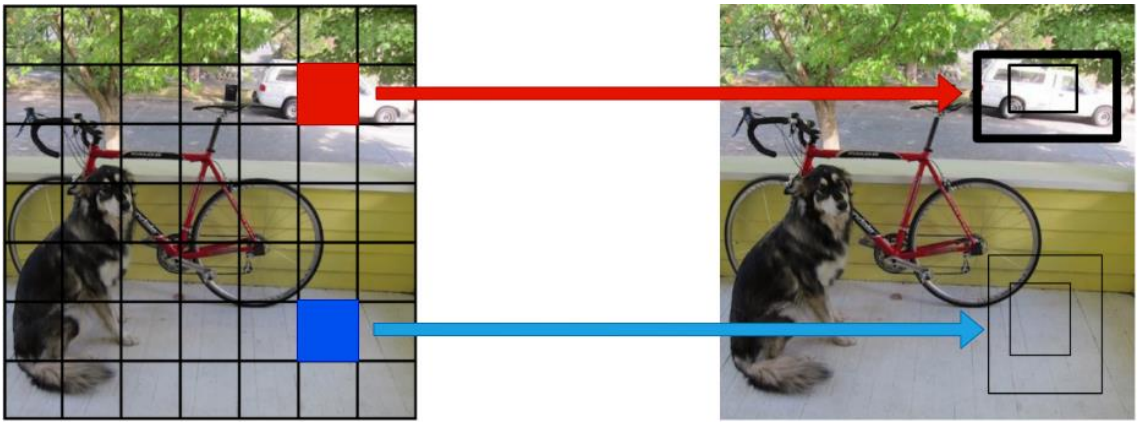
$$P(objeto) \times IOU$$

Onde:

- $P(objeto)$ é a probabilidade de haver um objeto na caixa;
- IOU (*Intersection Over Union*) é a divisão da área compartilhada entre dois formatos A e B pelo espaço total que eles ocupam (REZATOFIGHI *et al.*, 2019). O cálculo é dado por $\frac{A \cap B}{A \cup B}$, onde A é a caixa delimitadora gerada pelo algoritmo

detector e B a área real que ocupa o objeto. A Figura 11.6 mostra um exemplo da aplicação de IOU.

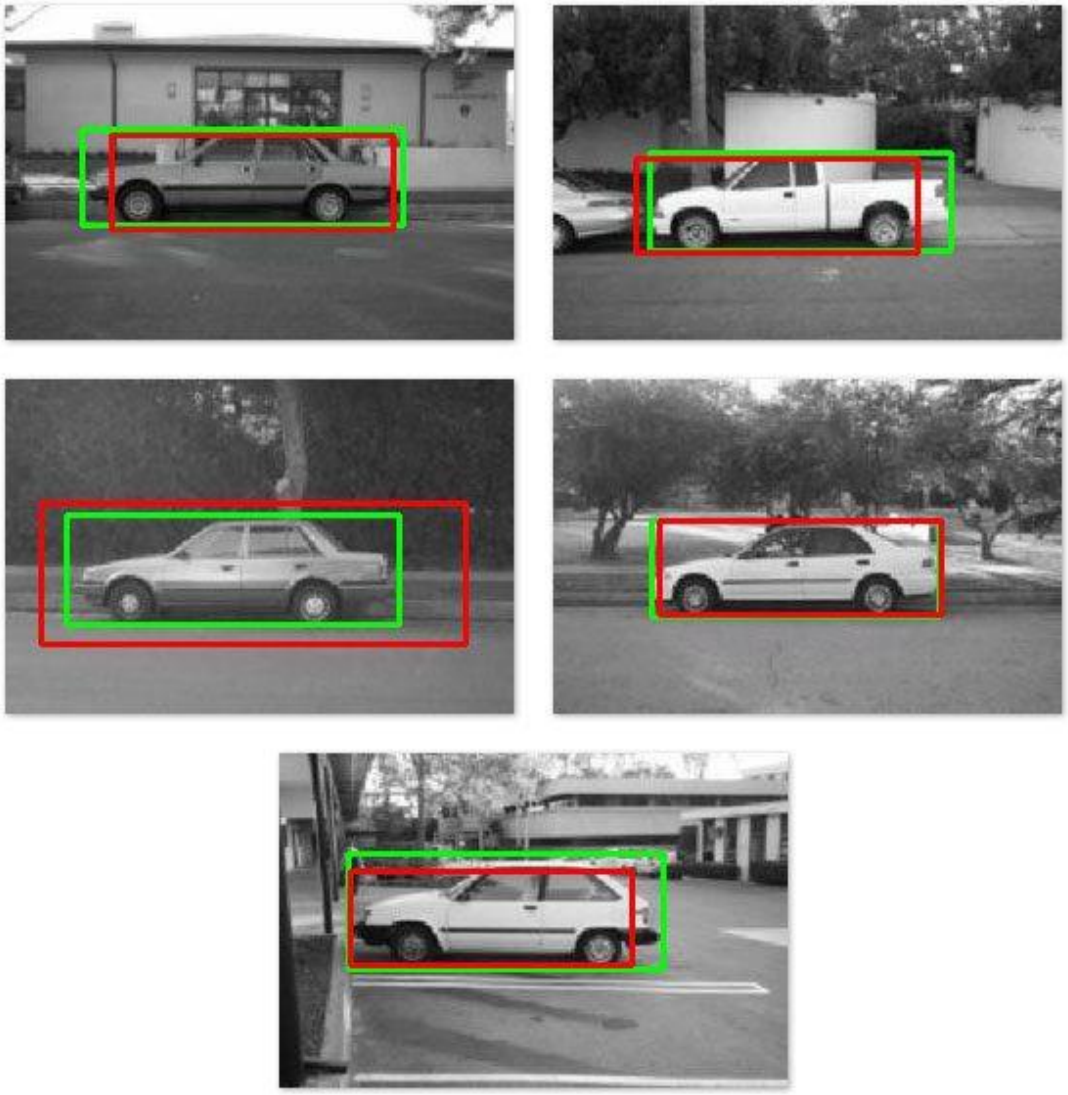
Figura 11.5: Previsão de caixas delimitadoras. Células que contenham partes de um objeto irão gerar caixas com maior ponto de confiança. Por causa disso, os formatos resultantes da célula vermelha apresentam maior espessura que as obtidas através da célula azul.



Fonte: Adaptado de Redmon *et al* (2016)

Com os cálculos obtidos, haverá um elevado grau de acurácia em relação as dimensões das caixas. Ou seja, os formatos obtidos estarão preenchendo as áreas reais que os objetos ocupam na imagem. Dessa forma, YOLO obtém a localização dos objetos ao mesmo tempo que prevê as suas classes. Além disso, é importante mencionar que as células preveem 5 elementos que formam as caixas: x, y, w, h e o ponto de confiança explicado anteriormente (REDMON *et al*, 2016). As coordenadas (x, y) são os pontos centrais de cada caixa, em função das bordas da célula. Já w e h são a largura (*width*) e a altura (*height*) da caixa em relação ao tamanho da imagem.

Figura 11.6: Aplicação de IOU para medir a acurácia da detecção de um objeto. Os retângulos vermelhos foram obtidos através de algoritmos detectores, enquanto os verdes correspondem ao espaço real que os automóveis ocupam. Quanto maior for a semelhança entre os dois retângulos, maior será a qualidade da detecção.



Fonte: Rosebrock (2016)

Para responder quais objetos estão na imagem, YOLO utiliza as células geradas na primeira etapa. Cada uma delas prevê C classes aos objetos contidos nos formatos delimitadores. Em seguida, são calculados os índices de confiança, cuja função é definir o grau de acerto de cada classe obtida. Finalmente, esses índices são combinados com os pontos obtidos na etapa de localização. O resultado final define quais objetos estão localizados nas caixas. O exemplo de classificação é mostrado na Figura 11.7.

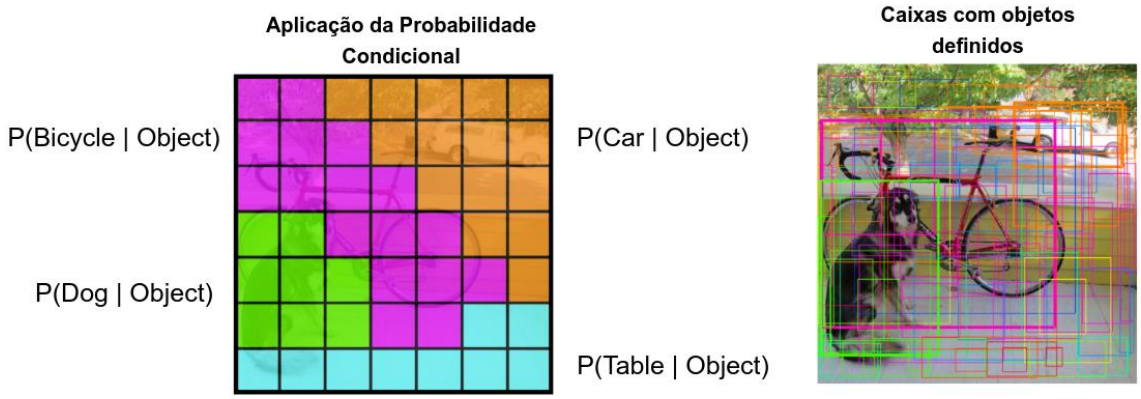
Em termos matemáticos, a classificação é dada por:

$$P(classe_i|objeto) \times P(objeto) \times IOU = P(classe_i) \times IOU$$

Onde:

- $P(classe_i|objeto)$ é a probabilidade condicional[2] da $classe_i$ aparecer sabendo que um $objeto$ foi localizado. $classe_i$ representa alguma classe que o algoritmo sugeriu como resposta (ex.: árvore, pessoa, cachorro, barco etc.);
- $P(classe_i)$ é a probabilidade de $classe_i$ ocorrer;
- $P(objeto)$ e IOU apresentam as mesmas definições mostradas anteriormente.

Figura 11.7: A etapa de classificação em YOLO. Cada célula prevê uma classe, atribuindo, em seguida, um índice de confiança (esquerda). O cálculo dos índices é realizado através de probabilidade condicional. No final, as caixas com maior espessura apresentam maior índice de confiança, tendo, portanto, as classes corretas.



Fonte: Adaptado de Redmon *et al* (2016)

Com a localização e classificação prontas, a saída do algoritmo será o conjunto de caixas com os objetos detectados. No entanto, é necessário aplicar um filtro de pós-processamento para exibir somente as caixas que delimitam corretamente os objetos, excluindo os retângulos duplicados. Redmon *et al* (2016) utilizam NMS (*Non-maximum Suppression*) para realizar essa filtragem.

Salscheider (2021) explica as etapas que formam NMS. Essa técnica consiste em utilizar duas listas: P, cujos elementos são as pontuações de confiança; e D, que guardará as detecções finais. D é vazia inicialmente. Então, é aplicada uma estrutura iterativa até que P fique vazia. O maior ponto p da lista P é removido e comparado com os elementos de D. Se a intersecção sobre união (IOU) entre p e todos os elementos de D for menor que o limite L, então p será adicionado em D. Caso contrário, p será descartado. Geralmente, L é igual a 0.5, mas o usuário pode atribuir um outro valor.

[2] Dado dois eventos A e B, $P(A|B)$ é a probabilidade condicional de A ocorrer, já sabendo que B aconteceu. Formalmente, $P(A|B) = \frac{P(A \cap B)}{P(B)}$, sendo $P(B) > 0$ (HAZZAN, 2013).

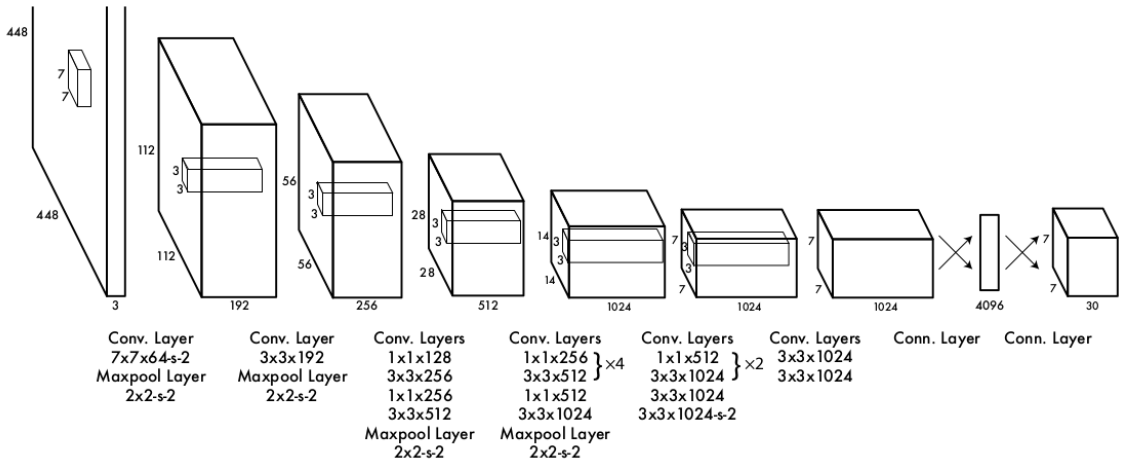
Ao juntar todas as previsões de YOLO (números de célula, caixas e classes), Redmon *et al* (2016) obteve $SxSx(Nx5 + C)$ tensores[3], que são as saídas do modelo. Quando o algoritmo foi treinado utilizando a base de imagens PASCAL VOC de 2007 (VOC2007), que possui 20 classes de objetos (EVERINGHAM *et al.*, 2009), foram utilizados 1470 tensores, o que resulta em $7x7x(2x5 + 20)$. É importante mencionar que além da base PASCAL VOC, YOLO foi previamente treinada com outro banco de imagens: a ImageNet 1000-class.

11.4 Arquitetura da CNN

O modelo desenvolvido por Redmon *et al* (2016) é uma rede neural convolucional (já apresentada no capítulo 9) (CNN) *open source*, chamada Darknet. Suas primeiras camadas são responsáveis por extrair os recursos da imagem, enquanto as camadas totalmente conectadas preveem a probabilidade das classes e as coordenadas de saída. A rede possui 24 camadas convolucionais e duas totalmente conectadas. Além disso, cada camada possui uma redução 1 X 1 de seu antecessor, seguidas por camadas convolucionais 3 X 3. A Figura 11.8 mostra a arquitetura desenvolvida.

Redmon *et al* (2016) também desenvolveu um modelo mais rápido do algoritmo, chamado de *Fast* YOLO. O CNN utilizado em *Fast* YOLO possui 9 camadas ao invés de 24. Contudo, os parâmetros de treinamento e teste utilizados em YOLO e *Fast* YOLO são os mesmos.

Figura 11.8: Representação da CNN utilizada. São utilizadas 24 camadas convolucionais seguidas por duas camadas totalmente conectadas.



Fonte: Redmon *et al* (2016)

[3] Um tensor é uma matriz multidimensional, muito utilizado em Inteligência Artificial (CAMMOUN *et al.*, 2009).

11.5 Versões de YOLO

11.5.1 YOLO V1

Embora YOLO apresente uma excelente eficácia na detecção de objetos e um tempo de processamento muito baixo, os seus autores identificaram limitações importantes. Após treinar o modelo com a base de imagens PASCAL VOC, notou-se que a acurácia da detecção era 69, um valor baixo se comparado com *Fast* R-CNN e *Faster* R-CNN (REDMON *et al*, 2016). Adarsh, Rathi e Kumar (2020) explicam que o modelo criado por Redmon *et al* (2016) possui erros de detecção quando há um agrupamento de objetos bastante próximos, como cardumes ou aves voando uma ao lado da outra. Se o conjunto possuir objetos de pequenas dimensões, haverá dificuldades de reconhecê-los. O problema de detecção também ocorre quando as imagens a serem processadas apresentam dimensões diferentes das fotos encontradas na base de treinamento (ADARSH; RATHI; KUMAR, 2020).

Por causa desses problemas, YOLO passou por várias atualizações. A sua primeira versão, portanto, foi publicada em 2016. Desde aquele ano, houve mais quatro versões, com objetivo de melhorar o seu desempenho.

Tabela 11.1: Comparação dos algoritmos detectores de objetos. Embora YOLO apresente a menor latência de processamento, a sua acurácia ainda é média.

Comparação dos algoritmos detectores de objetos (PASCAL VOC 2007)			
	Acurácia	Latência	
DPM V5	33,7	0,07 FPS	14 s/img
R-CNN	66	0,05 FPS	20 s/img
Fast R-CNN	70	0,5 FPS	2 s/img
Faster R-CNN	73,2	7 FPS	140 ms/img
YOLO	69	45 FPS	22 ms/img

Fonte: Redmon *et al* (2016)

11.5.2 YOLO V2

Redmon e Farhadi (2017) desenvolveram a segunda versão de YOLO, cuja principal característica foi um ótimo equilíbrio entre a acurácia e o tempo de processamento. Os autores introduziram novas funcionalidades no algoritmo, que geraram uma acurácia de 73,4 nos testes com a base PASCAL VOC 2012. Abaixo são mostradas algumas das novidades presentes em YOLO V2.

- **Normalização de lote –** Ioffe e Szegedy (2015) explicam que a normalização de lote é uma técnica para melhorar a média de aprendizado de uma rede neural. Conforme mostrado por eles, a distribuição das entradas de cada camada neural muda enquanto acontece o treinamento, uma vez que os parâmetros das camadas anteriores variam. A normalização de lote busca melhorar e normalizar este processo de aprendizado, reduzindo as altas variações dos dados. Redmon e Farhadi (2017) apontaram que houve mais de 2% de melhoria na normalização da acurácia.
- **Classificador de alta resolução –** Durante o pré-treinamento com ImageNet, YOLO V1 utilizou imagens de entrada com metade de sua resolução: 224 X 224 (REDMON *et al*, 2016). Em YOLO V2, a resolução adotada é 448 X 448 para o treinamento com ImageNet, melhorando a acurácia em quase 4% (REDMON; FARHADI, 2017).
- **Caixas de âncoras –** YOLO V2 substituiu as camadas totalmente conectadas por caixas de âncoras para gerar os retângulos delimitadores (REDMON; FARHADI, 2017). Segundo Zhong *et al* (2020):

  > Muitos detectores modernos baseados em aprendizado profundo usam as caixas de âncora (ou caixas padrão), que servem como as estimativas iniciais das caixas delimitadoras. Essas caixas de ancoragem são densamente distribuídas pela imagem, geralmente centralizadas em cada célula do mapa de recursos final. Em seguida, a rede neural é treinada para prever os deslocamentos de translação em relação ao centro da célula (às vezes normalizados pelo tamanho da âncora) e os deslocamentos de largura/altura em relação à forma da caixa de âncora, bem como a confiança de classificação. (ZHONG *et al*, 2020)

  Redmon e Farhadi (2017) explicam que YOLO V2 prevê mais de 1000 retângulos delimitadores por causa das caixas de âncoras. YOLO V1 previa 98.
- **Recursos com maior refinamento –** YOLO V2 divide as imagens de entrada em células 13 X 13 para detectar objetos pequenos. Isso resolve o problema encontrado em YOLO V1, que possuía dificuldades em reconhecer objetos de pequenas dimensões.
- **Treinamento multi-escala –** para resolver a limitação encontrada em YOLO V1 com relação a detecção de objetos de tamanhos variados, a nova versão do algoritmo é treinada com imagens de resoluções diferentes (320 X 320, 352 X 352, ..., 608 X 608). Por causa disso, YOLO V2 é capaz de detectar objetos em imagens com tamanhos distintos.
- **Darknet-19 –** YOLO V2 utiliza a rede neural Darknet-19, que apresenta 19 camadas convolucionais e 5 camadas *max pooling*. A estrutura da Darknet-19 é rápida e capaz de detectar objetos em tempo real (JONNALAGADDA, 2019).

11.5.3 YOLO V3

A terceira versão de YOLO, publicada por Redmon e Farhadi (2018), se caracterizou pelo uso de regressão logística[4] na geração de caixas delimitadoras e na classificação de objetos.Wasserman (2004) explica que a regressão logística é utilizada em situações onde o resultado esperado não depende apenas de uma única variável. Além disso, a saída é binária (ou $Y \in [0,1]$). Um exemplo de problema cujo resultado apresenta essa característica é a própria detecção facial. Numa foto, o algoritmo treinado para essa situação irá retornar se há ou não um rosto, bem como se pertence a uma determinada pessoa cujos atributos foram aprendidos por ele.

Assim como em YOLO V2, a terceira versão também utiliza caixas de ancoragem para gerar as formas delimitadoras. Cada caixa terá quatro coordenadas previstas pela rede: t_x, t_y, t_w e t_h. Se a célula representada pela coordenada (c_x, c_y) estiver no canto superior esquerdo da imagem e a caixa delimitadora anterior possuir largura p_w e altura p_h, então os cálculos das previsões serão:

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = e^{t_w} * p_w$$
$$b_h = e^{t_h} * p_h$$

Onde $\sigma(x)$ é uma função sigmoide[5]. Os pontos de confiança de cada formato delimitador são obtidos através de regressão logística. Se a caixa obtida estiver delimitando corretamente um objeto, o seu ponto será 1. Mas se o retângulo estiver sobreponto parte de um objeto, a forma será descartada. Também é aplicada a técnica de NMS para escolher as caixas que melhor delimitam os objetos, descartando aquelas com menor objetividade (ADARSH; RATHI; KUMAR, 2020).

A etapa de classificação apresenta um importante diferencial em relação as duas primeiras versões de YOLO. O algoritmo é capaz de atribuir mais de um rótulo a um determinado objeto. Por exemplo, ao identificar um ser humano, YOLO V3 pode retornar não somente o rótulo “pessoa”, mas também o sexo. Isso ocorre graças ao uso de classificadores logísticos, que retornam altas probabilidades para as classes que melhor se enquadram em um objeto. Logo, em YOLO V3, o problema de detecção é multi-rótulos ao invés de ser multi-classes (ADARSH; RATHI; KUMAR, 2020). Nas versões anteriores, o modelo

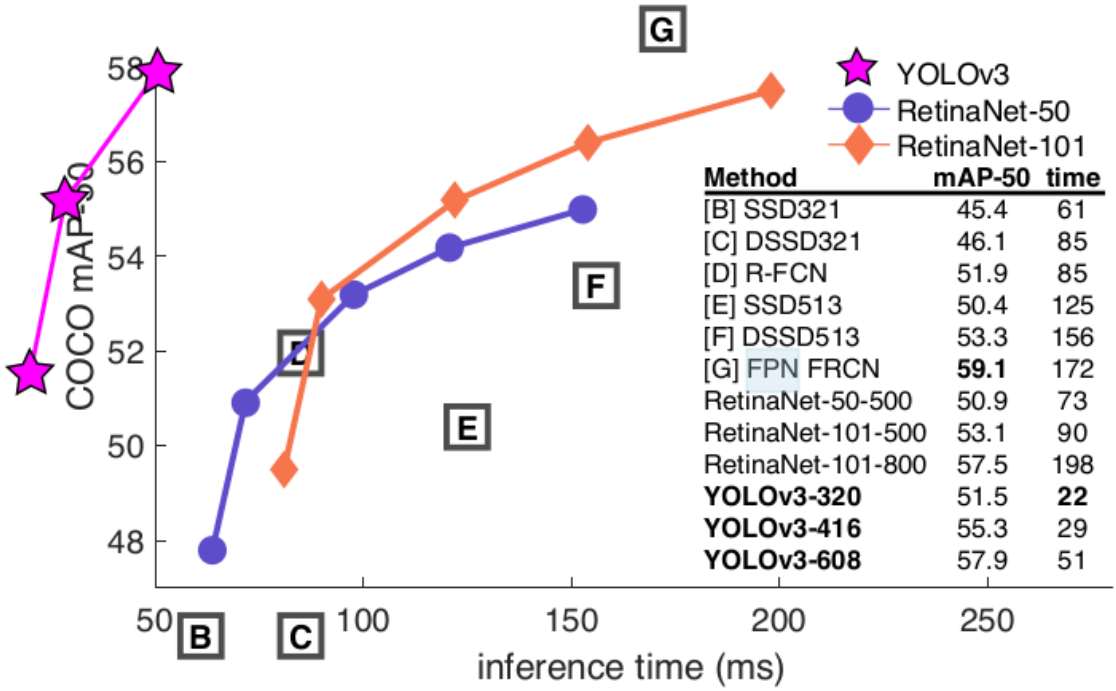
[4] Para uma abordagem matemática e com maior profundidade de regressão linear, ver Hastie, Tibshirani e Friedman (2017).

[5] É a função de uma regressão logística. O valor de y dessa função será maior ou igual a 0 e menor ou igual a 1 ($0 \leq y \leq 1$) (MITCHELL, 2018). Graficamente, a função sigmoide é curvilínea, parecendo um “S”. Mitchell (2018) define que esta função é dada por: $\sigma(x) = \frac{1}{1+e^{-x}}$. Geralmente, a letra grega sigma (σ) representa a função sigmoide.

não era capaz de atribuir mais de um rótulo a um objeto, retornando apenas a classe com maior probabilidade.

Quanto a arquitetura, YOLO V3 utiliza a Darknet-53, que possui 53 camadas convolucionais 3 X 3 e 1 X 1. Esta CNN também apresenta conexões de atalho. Por fim, YOLO V3 apresenta maior acurácia e menor latência, conforme mostrado na Figura 11.9. Por fim, essa versão possui avanços significativos em relação ao seu antecessor, como na detecção de objetos pequenos e no uso da classificação *multilabel* (ADARSH; RATHI; KUMAR, 2020).

Figura 11.9: Gráfico com o desempenho de YOLO V3 comparado com outros detectores. A base de imagens utilizada nos testes foi a Microsoft *Common Objects in Context* (MS COCO). Note que YOLO V3 apresenta a menor latência de processamento, chegando a ser 22 ms. Além disso, sua acurácia atinge 57.9.



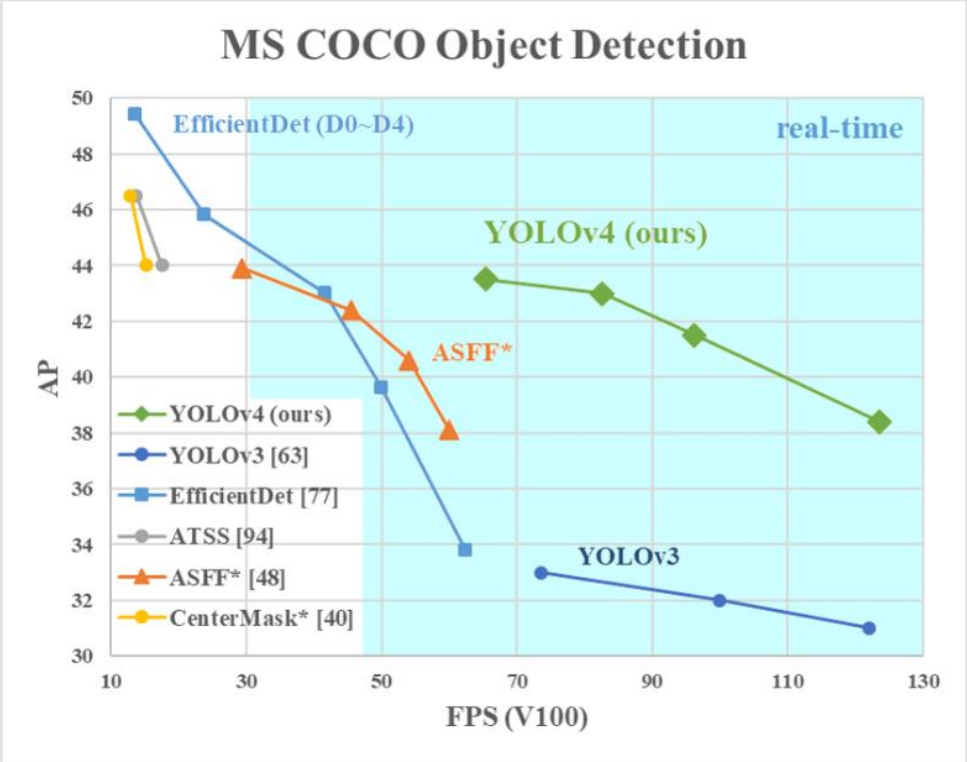
Fonte: Redmon e Farhadi (2018)

11.5.4 YOLO V4

Após publicar a terceira versão de YOLO, Joseph Redmon decidiu parar suas pesquisas voltadas a Visão Computacional. Em um *tweet* de 2020, ele explicou o motivo: “Parei de fazer pesquisas em Visão Computacional pois vi o impacto que meu trabalho estava tendo. Adorei o trabalho, mas as aplicações militares e as preocupações com a privacidade acabaram se tornando impossíveis de ignorar” (REDMON, 2020). Contudo, outros pesquisadores continuaram desenvolvendo melhorias no algoritmo lançado originalmente em 2016.

Em abril de 2020, um artigo apresentando YOLO V4 foi lançado por Alexey Bochkovskiy[6], Chien-Yao Wang e Hong-Yuan Mark Liao. A nova versão conseguiu ser mais acurada e com menor latência em relação ao seu antecessor (*vide* Figura 11.10). YOLO V4 melhora em 10% a acurácia obtida em YOLO V3, e 12% em relação ao tempo de processamento (*Frames per Second* - FPS) (BOCHKOVSKIY; WANG; LIAO, 2020).

Figura 11.10: Gráfico com o desempenho de YOLO V4 em relação a outros detectores de objetos. A base de imagens utilizada nos testes foi a MS COCO.

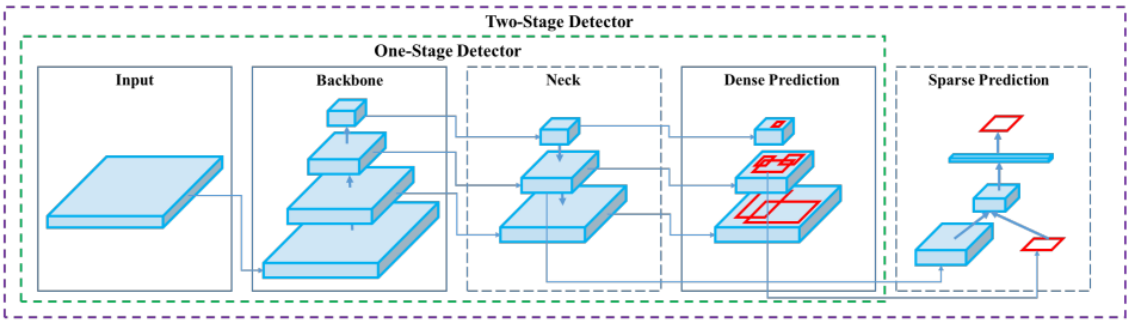


Fonte: Bochkovskiy, Wang e Liao (2020)

Os diferenciais presentes em YOLO V4 estão localizados principalmente na arquitetura da rede neural. Por isso, Bochkovskiy, Wang e Liao (2020) tiveram que explicar previamente como ocorre a detecção de objetos em uma rede neural. A Figura 11.11 ilustra as etapas desse processo.

[6] Alexey Bochkovskiy ficou conhecido por ter desenvolvido as versões YOLO V2 e V3 para Windows (COHEN, 2020).

Figura 11.11: Etapas de detecção de objetos em uma rede neural.



Fonte: Bochkovskiy, Wang e Liao (2020)

Dada uma imagem de entrada (*input*), camadas adicionais serão aplicadas para identificar objetos. Dependendo do algoritmo, isso é realizado em uma única etapa (ex.: YOLO) ou em duas (como ocorre em R-CNN e suas variações). Abaixo serão explicados os termos apresentados na Figura 11.11.

- ***Backbone* –** é a arquitetura da extração de recursos (COHEN, 2020). Darknet é um exemplo de *backbone*, muito utilizada desde a primeira versão de YOLO. Contudo, na quarta versão, Bochkovskiy, Wang e Liao (2020) decidiram utilizar a arquitetura CSPDarknet53 (*Cross-Stage-Partial connections* – CSP), caracterizada por dividir os recursos da imagem em duas partes: uma parte percorre um bloco denso de convoluções e uma camada de resolução; e outra é combinada com o mapa de características transmitido no próximo estágio. No final, os resultados obtidos nos dois segmentos juntam-se, agregando as saídas (WANG *et al.*, 2020). CSPDarknet53 é uma arquitetura que requer alto poder de processamento, sendo utilizada em GPUs. Os autores também fizeram estudos com um *backbone* menor: a arquitetura MobileNet.
- ***Neck* –** é o bloco de camadas intermediárias entre *backbone* e *head*. Sua função é extrair recursos da imagem em diferentes estágios da CNN. Essas camadas garantem uma maior acurácia na detecção de objetos. No YOLO V4, as técnicas utilizadas nessa etapa são SPP (*Spatial Pyramid Pooling*) e PAN (*Path Aggregation Network*). SPP busca adaptar imagens de vários tamanhos entre as camadas convolucionais. Dessa forma, não é preciso redimensionar uma imagem antes de alimentar uma rede neural, uma vez que somente as camadas totalmente conectadas precisam de tamanhos fixos, cujos valores são alterados pelo algoritmo (He *et al.*, 2018). Cohen (2020) salienta que PAN busca adicionar camadas para obter maiores informações de cada recurso. Essas informações são agregadas em bloco, gerando uma maior acurácia (LIU *et al.*, 2018).

- ***Head* –** é a camada de geração das caixas delimitadoras e da previsão das classes (BOCHKOVSKIY; WANG; LIAO, 2020). Nessa etapa, o processo adotado é o mesmo em YOLO V3. Ou seja, a detecção dos objetos ocorre em um único estágio (*Dense Prediction*) ao invés de dois (*Sparse Prediction*).

Tendo em vista as etapas de detecção de objetos em redes neurais, Bochkovskiy, Wang e Liao (2020) aplicaram duas técnicas que diferenciaram YOLO V4 de seus antecessores: *Bag-of-Freebies* (BoF) e *Bag-of-Specials* (BoS). Segundo Cohen (2020): "BoF é um conjunto de técnicas que auxiliam durante o treinamento sem adicionar muito tempo de inferência (COHEN, 2020)". Os autores de YOLO V4 aplicaram BoF tanto no *backbone* quanto no detector. Em relação as técnicas BoS, ocorre o aumento do custo de inferência. A causa disso é a alteração da arquitetura da rede (COHEN, 2020). Os algoritmos PAN e SPP são exemplos de técnicas BoS. Por fim, em YOLO V4 ocorre a aplicação de BoS tanto em *backbone* quanto no detector.

11.5.5 YOLO V5

A quinta versão de YOLO foi a primeira a não ser publicada originalmente em um artigo. Desenvolvido pelo CEO e pesquisador da Ultralystics LCC, Glenn Jocher, YOLO V5 foi publicado um mês após a versão criada por Alexey Bochkovskiy, Chien-Yao Wang e Hong-Yuan Mark Liao[7] (THUAN, 2021). YOLO V5 foi desenvolvido com linguagem Python ao invés de C, como aconteceu com as versões anteriores. Isso facilitou a aplicação desse algoritmo em dispositivos IoT (*Internet of Things*), conforme explicado por Thuan (2021).

Outra característica de YOLO V5 é a sua arquitetura. A rede neural Darknet era utilizada até a quarta versão do algoritmo, mas o criador de YOLO V5 optou por utilizar o PyTorch, *framework* de *Machine Learning* utilizado na construção de redes neurais através de tensores. Entretanto, YOLO V5 apresenta mais similaridades do que diferenças com a sua versão anterior (THUAN, 2021). Jiang *et al* (2022) salientam que YOLO V5 é menos inovador que YOLO V4, mas que possui vantagens quanto ao desempenho. A saber:

- PyTorch é *user-friendly*, isto é, de fácil uso e compreensão. Além disso, possui maior facilidade no treinamento dos dados. Por fim, as soluções desenvolvidas com PyTorch são menos difíceis de implantar no ambiente de produção do que aquelas feitas com Darknet;
- O código é de fácil leitura, e pode ser integrado com um largo número de tecnologias relacionadas a Visão Computacional;

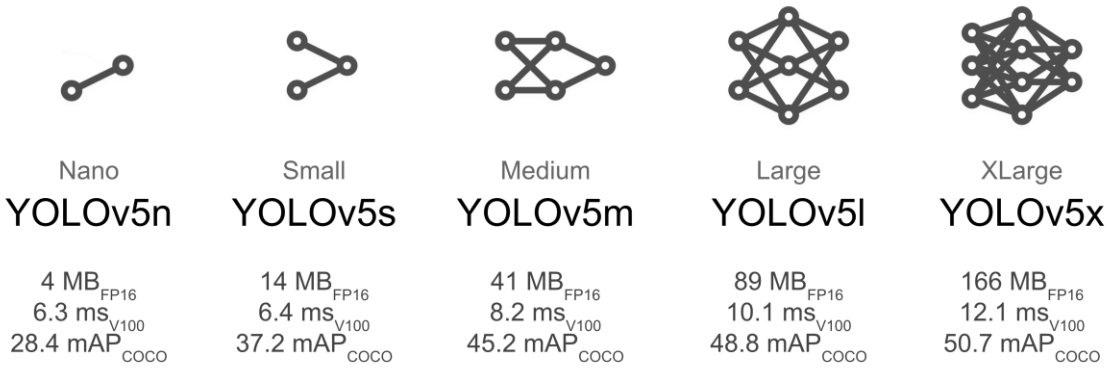
[7] Bochkovskiy, Wang e Liao (2020) lançaram YOLO V4 em abril de 2020, através de um artigo. Em maio de 2020, Jocher (2020) publicou um repositório no GitHub contendo o código de YOLO V5.

- O ambiente de desenvolvimento é de fácil configuração, e a etapa de treinamento do modelo se caracteriza por ser rápida. O uso de lotes trás resultados em tempo real.

Os lotes de treinamento em YOLO V5 são criados a partir dos dados carregados no modelo. De forma simultânea, também ocorre o tratamento desses dados, focando em três aspectos: dimensionamento das imagens, ajustes na escala de cor e aprimoramento de mosaico[8]. Em específico, esse último aspecto é eficiente no reconhecimento de objetos em escala menor (JIANG *et al*, 2022).

O algoritmo de Jocher apresenta cinco variações com acurácia e latência diferentes: YOLO V5 *nano* (n), YOLO V5 *small* (s), YOLO V5 *medium* (m), YOLO V5 *large* (l) e YOLO V5 *xlarge* (x) (JOCHER, 2020). Como mostrada na Figura 11.12, outra diferença dessas variações está no tamanho e no desenho da rede neural.

Figura 11.12: Diferenças entre as cinco variações de YOLO V5.

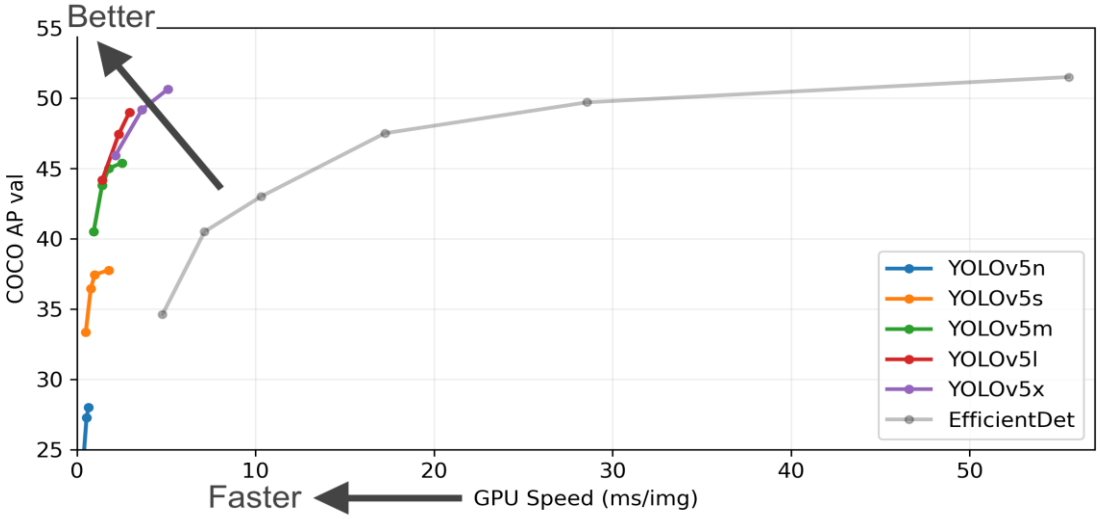


Fonte: YOLO V5 *Documentation* (2020)

Jocher (2020) realizou testes com as cinco variações de YOLO V5, utilizando a base de imagens MS COCO. Os resultados obtidos foram muito satisfatórios: o tempo de processamento de todas as variações é abaixo de 20 ms, e a acurácia chega em níveis acima de 50 (ver Figura 11.13). Embora YOLO V5 não apresente técnicas inovadoras em relação ao seu antecessor, é inegável afirmar que o seu desempenho é um dos melhores desde a primeira versão publicada em 2016. Como visto acima, o uso do PyTorch trouxe mais facilidades na hora de utilizar esse detector de objetos.

[8] Mosaico de imagens é uma técnica que consiste em unir vários segmentos de imagens de uma cena em uma única foto (VAGHELA; NAINA, 2014). Nas versões 4 e 5 de YOLO, esse processo ocorre com imagens de objetos em diferentes lugares, com o objetivo de melhorar a habilidade do modelo em reconhecer objetos em qualquer ambiente (BOCHKOVSKIY; WANG; LIAO, 2020).

Figura 11.13: Comparação do conjunto de algoritmos YOLO V5 com o detector de objetos EfficientDet.



Fonte: Jocher (2020)

11.5.6 YOLO V6

Inicialmente lançado em um repositório do GitHub pelo departamento de Visão Computacional da empresa Meituan, um *e-commerce* chinês, a sexta versão de YOLO foi desenvolvida com foco na área industrial. Em setembro de 2022, Li *et al* (2022) publicaram um artigo explicando os diferenciais presentes em YOLO V6, que se encontram sobretudo na arquitetura da rede neural. Contudo, este detector adota algumas técnicas da versão criada por Bochkovskiy, Wang e Liao (2020). Abaixo são mostradas as características da arquitetura de YOLO V6:

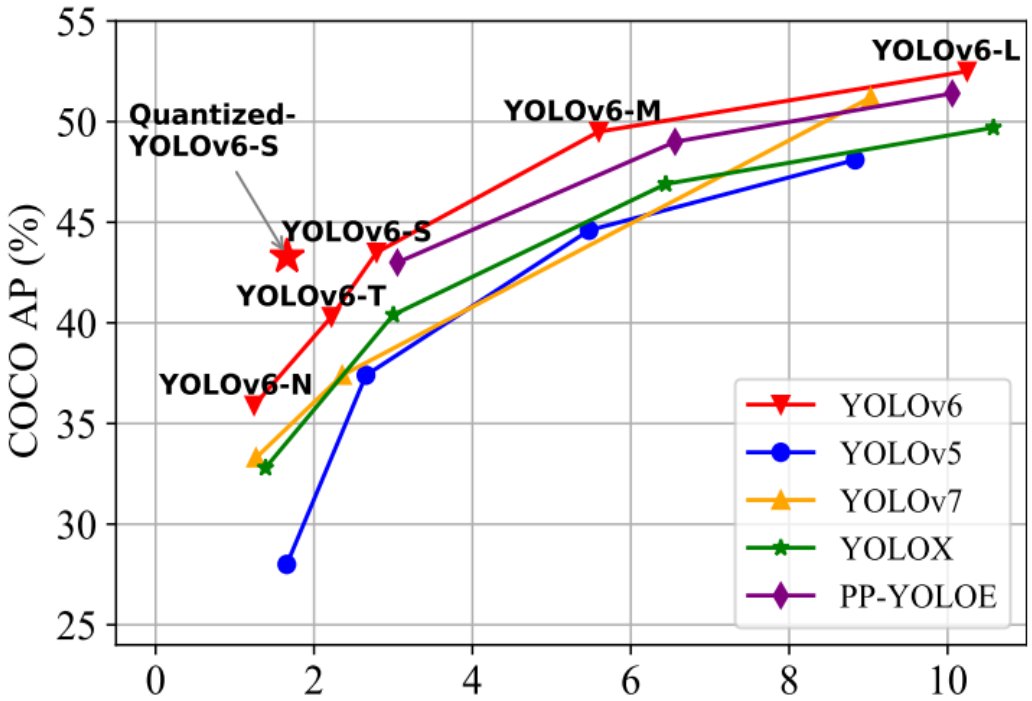
- ***Backbone* –** os autores utilizam reparametrização em *backbones* de redes com variadas dimensões. Dinh (2018) explica que a reparametrização busca mudar o comportamento de uma função de perda[9]. Wang *et al* (2022) caracteriza esta técnica como um procedimento que busca unir múltiplos núcleos computacionais em apenas um. Seria, portanto, uma forma de tornar o modelo de aprendizado mais assertivo e ao mesmo com uma estrutura mais simples. Para redes pequenas, YOLO V6 adota a RepVGG como *backbone*, uma reparametrização focada em simplificar a arquitetura de uma CNN, melhorando dessa forma o seu processamento (DING *et al*, 2021). Para redes maiores, foi utilizada a estrutura CSP com reparametrização, chamada de CSPStackRep.

[9] A função de perda é uma função que calcula a distância entre a saída de um modelo de *Machine Learning* com o resultado esperado (PERE, 2020).

- ***Neck*** **–** YOLO V6 usa PAN (*Path Aggregation Network*), o mesmo recurso encontrado em YOLO V4.
- ***Head*** **–** essa estrutura é formada por duas segmentações de camadas convolucionais, sendo uma focada na classificação e outra na localização dos objetos. Quanto a geração das caixas delimitadoras, os autores de YOLO V6 utilizam detectores sem âncoras. Esses detectores não geram estimativas iniciais acerca das possíveis áreas onde há objetos, evitando o processamento com a validação desses locais através de IOU (TIAN *et al*, 2019).

YOLO V6 possui 6 variações: YOLO V6-N, YOLO V6-T, YOLO V6-S, YOLO V6-M e YOLO V6-L. Ao compará-las com outros detectores, os autores obtiveram resultados satisfatórios em relação a acurácia e a latência. Conforme a Figura 11.14, YOLO V6-N atingiu 35,9 % de acurácia no banco de imagens MS COCO, enquanto o modelo YOLO V6-S obteve 43,5%. Os algoritmos foram testados em uma GPU NVIDIA Tesla T4.

Figura 11.14: Gráfico de acurácia das variações de YOLO V6.



Fonte: Li *et al* (2022)

11.5.7 YOLO V7

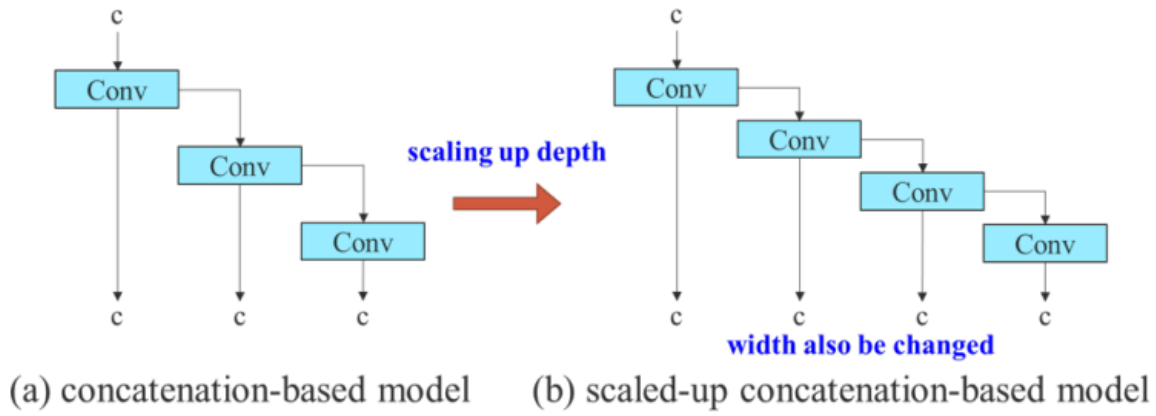
Em julho de 2022, um mês após o código-fonte de YOLO V6 ser lançado em um repositório do GitHub, o trio de pesquisadores responsáveis pela criação de YOLO V4 volta a contribuir com o detector originalmente feito por Joseph Redmon. Lançado em julho de 2022, o artigo de YOLO V7 traz a reparametrização como uma de suas principais

características, assim como o seu antecessor. Dessa forma, a rede neural de YOLO V7 possui uma estrutura que torna o processamento mais rápido, uma vez que a quantidade de parâmetros diminuiu 40% graças a essa técnica. Consequentemente, o modelo pode ser treinado com banco de imagens menores sem necessitar de pesos pré-treinados, resultando em menor custo computacional (VISIO.AI, 2022).

As novidades de YOLO V7 estão presentes em sua arquitetura e no aprimoramento da técnica de *Bag-of-Freebie*. Na arquitetura, há dois recursos importantes: *Extended Efficient Layer Aggregation Networks* (E-ELAN) e dimensionamento escalar de modelos baseados em concatenação (WANG *et al*, 2022).

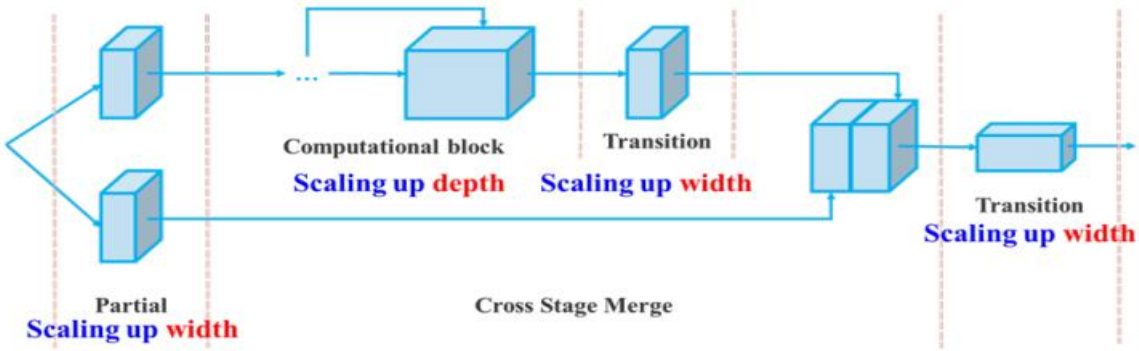
- **E-ELAN –** ELAN é um tipo de *design* de arquitetura eficiente, que foca na quantidade de parâmetros e na densidade computacional. Para melhorar o desempenho de uma rede neural, ELAN busca controlar os caminhos de gradiente mais curto e mais longo da estrutura. Os autores de YOLO V7 aprimoraram ELAN, buscando mesclar blocos de processamento da rede, bem como aumentar a cardinalidade das segmentações. A esta técnica, chamaram de E-ELAN.
- **Dimensionamento escalar de modelos baseados em concatenação –** o dimensionamento escalar consiste em ajustar os atributos de um modelo ou obter novos de variadas escalas com o objetivo de atender diferentes velocidades de processamento (WANG *et al*, 2022). Contudo, essa técnica apresenta limitações quando aplicadas em arquiteturas baseadas em concatenação de segmentos, que é a encontrada em YOLO V7. Conforme mostrado na Figura 11.15, quando se aumenta a escala, a largura do modelo também cresce. Para resolver isso, os autores propuseram adicionar um bloco de transição para cada escala inserida A Figura 11.16 ilustra essa característica do algoritmo.

Figura 11.15: Comportamento de uma rede neural baseada em concatenação. Quando se aumenta uma escala (item b), a largura do modelo também cresce.



Fonte: Wang *et al* (2022)

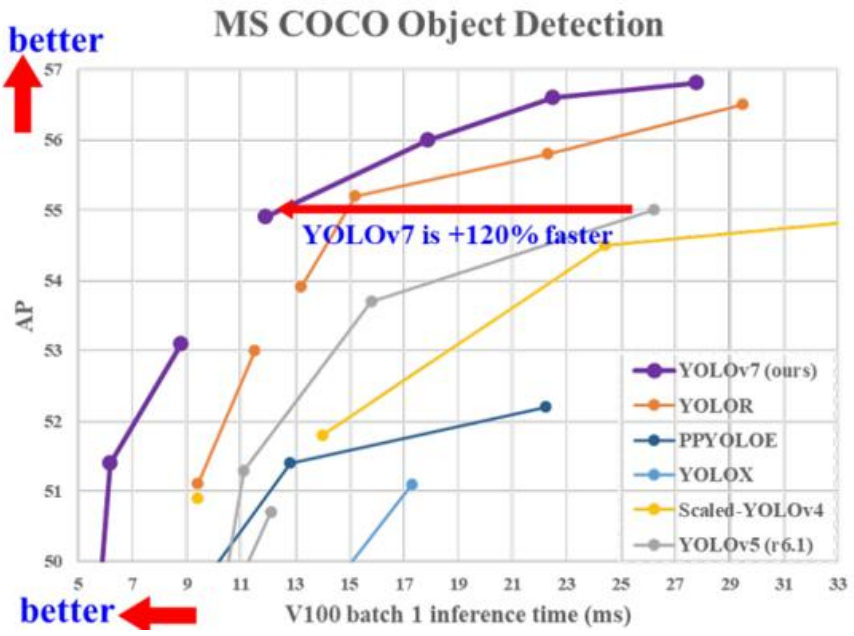
Figura 11.16: Dimensionamento escalar presente em YOLO V7. Ao adicionar uma escala, é inserido um bloco de transição que une as segmentações.



Fonte: Wang *et al* (2022)

Bag-of-Freebie (BoF) teve melhorias graças ao uso de reparametrização. Wang *et al* (2022) usaram a arquitetura RepVGG, a mesma adotada em YOLO V6. De forma resumida, RepVGG combina as convoluções 3 x 3, 1 x 1 e a conexão identidade em apenas uma única camada convolucional (WANG *et al*, 2022). Em YOLO V7, a conexão identidade não é unida a reparametrização, pois descobriu-se que o RepVGG prejudica as concatenações (WANG *et al*, 2022). YOLO V7 apresentou uma acurácia de 56,8% na base de imagens MS COCO, atingindo 11 milissegundos de latência (Figura 11.17).

Figura 11.17: Comparação de YOLO V7 com outros detectores.



Fonte: Wang *et al* (2022)

11.6 Implementando YOLO

Conforme visto anteriormente, YOLO não é um algoritmo dedicado ao reconhecimento facial. Ele é capaz de identificar vários objetos em milissegundos, incluindo pessoas, animais, veículos e outras classes. Entretanto, é possível treiná-lo para reconhecer objetos específicos, através de uma base de dados elaborada pelo próprio desenvolvedor. Logo, para realizar a implementação do algoritmo YOLO para identificação facial, será utilizada a biblioteca Yoloface, que traz o modelo treinado ao desenvolvedor. Através desse recurso, o usuário irá focar somente em chamar os métodos que detectem as faces de uma imagem, não se preocupando em treinar a rede neural ou definir as classes de objetos. Além disso, essa biblioteca utiliza a terceira versão de YOLO, e apresenta um excelente desempenho em computadores que utilizam CPU[10].

11.6.1 Implementando YOLO em imagens

Para fins de exemplo, a Figura 11.18 será utilizada na aplicação, cujo nome será "pessoas.jpg". A saída esperada será um conjunto de caixas delimitadoras sobre as faces encontradas.

Figura 11.18: Imagem de entrada do algoritmo.

Fonte: MS COCO (2022)

[10] Algoritmos de redes neurais, como YOLO, apresentam melhor desempenho quando treinados em dispositivos que utilizam computação paralela (GPU). Por ser um modelo pronto, Yoloface não precisa de treinamento e requer pouco processamento na detecção facial. Por causa disso, o uso de uma CPU já é suficiente para desenvolver as aplicações.

A primeira etapa é importar a biblioteca OpenCV e a classe face_analysis da biblioteca Yoloface, como mostrado abaixo. A terceira linha do Código 11.1 efetua uma instanciação da classe face_analysis, cujo objeto será “face”.

Código 11.1: Bibliotecas utilizadas no algoritmo YOLO.

```
from yoloface import face_analysis
import cv2 as cv
face=face_analysis()
```

Em seguida, são criadas três variáveis: “imagem”, “retangulos” e “grau_certeza”. Os dados atribuídos a elas são obtidos através dos pesos e configurações da rede neural que forma YOLO V3. Abaixo é descrita a função dessas variáveis:

- **“imagem” –** recebe o endereço da imagem de entrada;
- **“retangulos” –** guarda uma matriz formada por N listas. Cada lista é estruturada por quatro números, que formarão a caixa delimitadora sobre o rosto encontrado;
- **“grau_certeza” –** é a lista contendo as porcentagens de confiança das faces localizadas, podendo ser de 0 a 1. Por meio desses números, é possível avaliar o desempenho do algoritmo ao analisar uma determinada imagem.

O Código 11.2 mostra como os dados resultantes do modelo são atribuídos a essas variáveis.

Código 11.2: Chamada dos dados obtidos por YOLO V3, e que são armazenados nas variáveis “imagem”, “retangulos” e “grau_certeza”.

```
imagem, retangulos, grau_certeza =
  face.face_detection(image_path='pessoas.jpg',model='full')
```

O método “face_detection()” irá carregar os pesos e as configurações de YOLO V3. Note que ele possui dois parâmetros: “image_path”, que receberá o endereço da imagem de entrada; e “model”, cuja função é definir como a rede neural será estruturada. O parâmetro “model” pode ser “full” ou “tiny”. A primeira opção indica que YOLO V3 terá uma CNN mais complexa, enquanto “tiny” apresenta uma versão simplificada do modelo. Outra diferença entre essas duas opções é a saída do algoritmo. As caixas delimitadoras possuem coordenadas mais exatas em “full” YOLO V3 do que em “tiny”. Ou seja, as faces tendem a ser mais destacadas no modelo completo.

O próximo passo será imprimir as coordenadas e desenhar as caixas sobre a imagem. Também serão exibidos os graus de confiança de cada retângulo. Conforme mostrado em Código 11.3, será necessário utilizar uma estrutura iterativa para obter essas informações.

Código 11.3: Trecho do código que desenha os retângulos sobre a imagem, exibe as coordenadas e o grau de confiança.

```
indice = 0

while indice < len(retangulos):
    print("Face " + str(indice + 1) + ": " +
          str(retangulos[indice]) + "\tGrau de certeza: %.2f"
    % (grau_certeza[indice] * 100) + "%")

    x = retangulos[indice][0]
    y = retangulos[indice][1]
    l = retangulos[indice][3]
    a = retangulos[indice][2]

    rosto = cv.rectangle(imagem, (x,y), (x + l, y + a),
                         (0, 255, 0), 3)

    indice += 1
```

A variável “indice” inicializa em zero, e será incrementada de 1 em 1 enquanto for menor que o tamanho de “retangulos”. Como na imagem “pessoas.jpg” há quatro faces, então “retangulos” armazenará quatro listas. Logo no começo do laço de repetição “while”, irá ocorrer a impressão de cada uma dessas estruturas (“retangulo[indice]”), bem como a porcentagem de confiança, que é a multiplicação de “grau_certeza[indice]” por 100. A Figura 11.19 mostra a saída do comando “print”.

Figura 11.19: Saída com as coordenadas e grau de confiança de cada face.

```
Face 1: [497, 121, 67, 51]      Grau de certeza: 99.90%
Face 2: [352, 130, 53, 40]      Grau de certeza: 99.80%
Face 3: [223, 134, 67, 50]      Grau de certeza: 99.76%
Face 4: [88, 101, 110, 73]      Grau de certeza: 99.49%
```

As variáveis “x”, “y”, “l” e “a” são os vértices de cada retângulo. O método “rectangle” possui quatro parâmetros: a imagem a ser editada; a coordenada do vértice superior esquerdo; a coordenada do vértice inferior direito[11]; a cor da borda e a espessura. Os retângulos serão guardados na variável “rosto”. Por fim, será exibida a imagem com os retângulos, conforme o código abaixo.

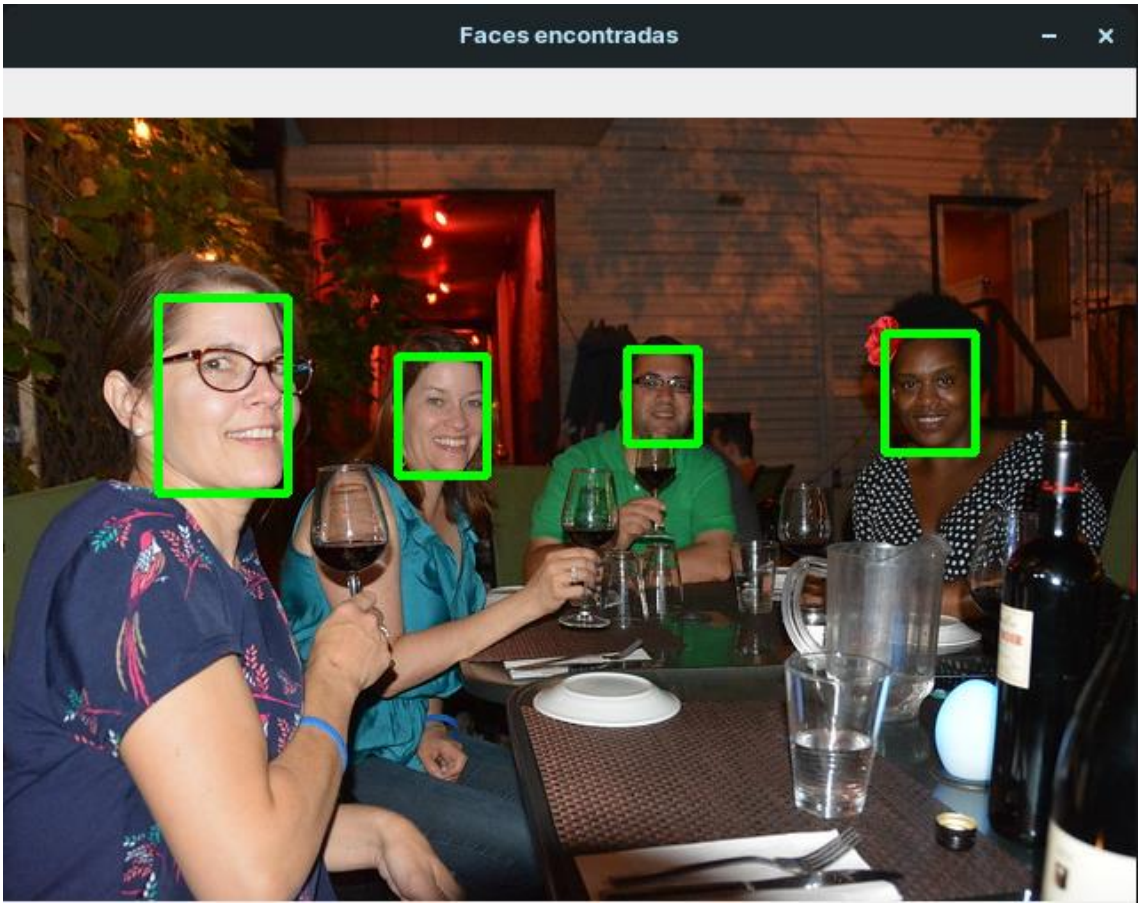
[11] No código, esse vértice é obtido da seguinte forma: o ponto horizontal é dado por “x” + “l” (largura), enquanto o vertical por “y” + “a” (altura).

Código 11.4: Código de saída do algoritmo.

```
cv.imshow("Faces encontradas", rosto)
cv.waitKey(0)
```

O resultado do algoritmo é mostrado na Figura 11.20. É importante salientar que ao executar pela primeira vez o código, serão carregados dois arquivos no diretório do projeto: uma lista de pesos e a configuração da rede neural.

Figura 11.20: Imagem com as caixas delimitadoras. Neste modelo customizado de YOLO, o algoritmo é capaz de localizar o objeto, mas não de aplicar um rótulo.



11.6.2 Implementando YOLO em vídeos

O código aplicado em vídeos apresenta semelhanças ao utilizado em imagens. Conforme visto no Código 11.5, é preciso importar o módulo “face_analysis” de Yoloface, bem como a biblioteca OpenCV. Em seguida, cria-se um objeto que será a instanciação da classe “face_analysis”, cujo nome será “face” no exemplo proposto. Na variável “video”, é indicado o diretório onde está salvo o vídeo de entrada, por meio do comando

“VideoCapture()”. Essa gravação será executada quadro a quadro, enquanto o arquivo estiver aberto (“while video.isOpened()”).

Ao abrir o vídeo, são criadas as variáveis “quadro”, “retangulos” e “grau_certeza”. “quadro” irá receber o *frame* do vídeo, enquanto “retangulos” e “grau_certeza” terão a mesma função vista na seção 11.6.1. Em Python, o uso de *underline* (_) indica que haverá valores que serão ignorados. Também o método “face_detection” é utilizado na detecção facial em vídeos, mas os parâmetros utilizados são diferentes daqueles aplicados em imagens. O argumento “frame_arr” recebe o conjunto de quadros do vídeo, sendo que “model” receberá o modelo simplificado de YOLO V3. Será utilizado *Tiny* YOLO neste exemplo, pois o tempo de processamento é menor do que o obtido com *Full* YOLO.

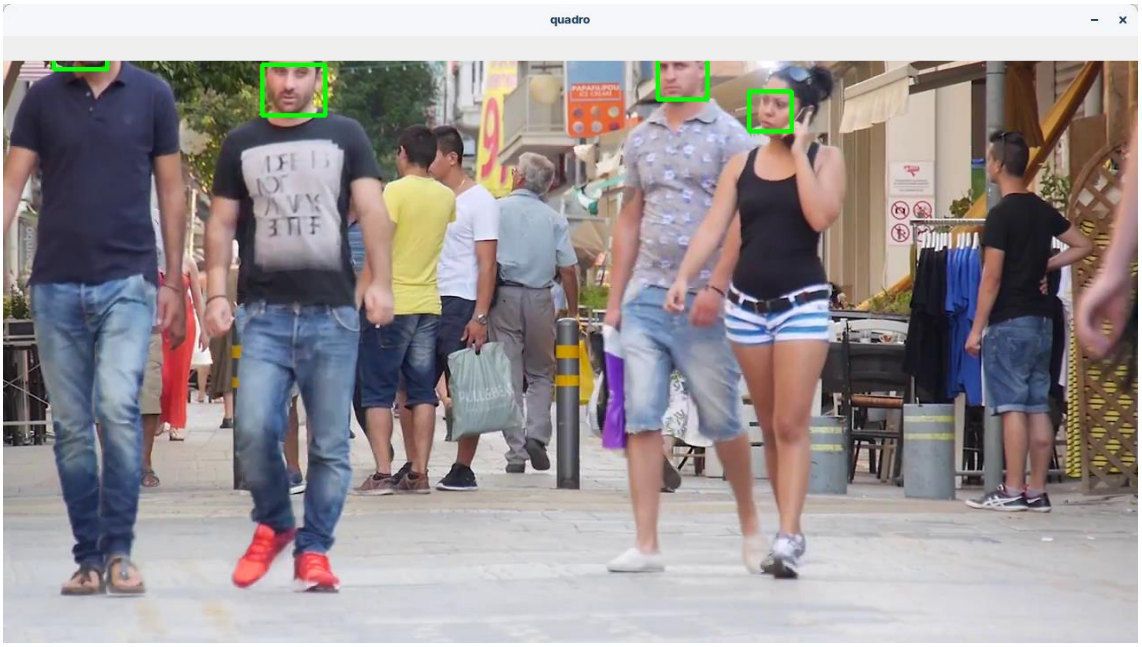
Após a declaração das variáveis, o código é idêntico ao que foi utilizado no reconhecimento facial em imagens. Será impressa a lista de coordenadas e o grau de confiança de cada retângulo que delimitará os rostos. Por fim, cada *frame* do vídeo será editado com as caixas delimitadoras. O processamento do algoritmo é finalizado se o usuário pressionar a tecla “q”.

Código 11.5: Implementação de YOLO em vídeos.

```
from yoloface import face_analysis
import cv2 as cv
face=face_analysis()
video = cv.VideoCapture(r'teste_pessoas.mp4')
while video.isOpened():
    _, quadro = video.read()
    _,retangulos,grau_certeza = face.face_detection(frame_arr=quadro,
                    model='tiny')
    indice = 0

    while indice < len(retangulos):
        print("Face     "     +     str(indice     +     1)     +     ":     "     +
              str(retangulos[indice]) +
        "\tGrau de certeza: %.2f"
        % (grau_certeza[indice] * 100) + "%")

        x = retangulos[indice][0]
        y = retangulos[indice][1]
        l = retangulos[indice][3]
        a = retangulos[indice][2]

        rosto = cv.rectangle(quadro,
                             (x,y), (x + l, y + a),
                             (0, 255, 0), 3)
        indice += 1

    cv.imshow('quadro',rosto)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break
```

A saída gerada pode ser vista na Figura 11.21.

Figura 11.21: Trecho do vídeo com a detecção facial através de YOLO.



11.6.3 Avaliando o desempenho de YOLO

Assim como foi mostrado nos demais algoritmos, agora será avaliado o desempenho de YOLO. Foram utilizadas as bases de imagens Yalefaces, *Labeled Faces in the Wild* (LFW) e *Dogs VS Cats* para analisar a *perfomance* de YOLO. Primeiro, será explicado o código presente no Código 11.6.

Código 11.6: Função para reconhecer uma face com YOLO.

```
def detectarYolo(x_train):
    y_train_predict = []
    tempos = []
    diretorio = x_train
    cont = 0
    print('Carregando os modelos de detecção .....')
    face=face_analysis()
    print('Fazendo detecção nas imagens ......')
    for imagem in diretorio:
        inicio = time()
 imagens,  retangulos,  grau_certeza  =  face.face_detection(image_path=imagem,
model='full')
        if len(retangulos) > 0:
            y_train_predict.append(1)
```

```
        else:
            y_train_predict.append(0)
        fim = time()
        tempos.append(fim - inicio)
    print('Fim da detecção.......')
    return y_train_predict, tempos
```

O código 11.6 é uma função que recebe como parâmetro o diretório de imagens que serão analisadas por YOLO (“x_train”). A saída será a lista “y_train_predict”, que armazenará 0 (caso a imagem não tenha faces) ou 1 (se possui rostos). Inicialmente, “y_train_predict” é vazia. A lista “tempos” irá receber o tempo de processamento de cada imagem. Por fim, a lista “diretorio” recebe as imagens de “x_train_predict”.

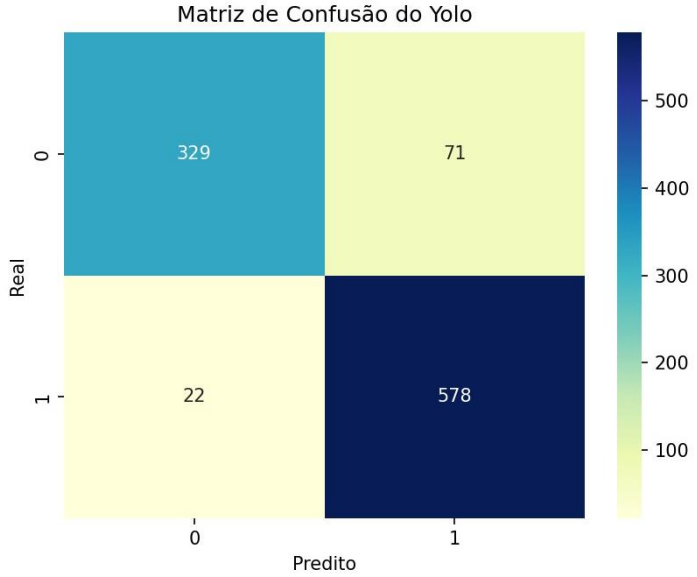
Como visto nas seções 11.6.1 e 11.6.2, é preciso instanciar um objeto da classe “face_analysis”. Na função, o objeto se chama “face”. Em seguida, é criada uma estrutura iterativa que percorre todas as imagens do diretório. Dentro desse comando, a variável “inicio” recebe o tempo inicial do processamento da imagem, através da função “time()” oriunda da biblioteca de mesmo nome.

As variáveis “imagens”, “retangulos” e “grau_certeza” recebem, respectivamente, as seguintes informações: o endereço da imagem; a lista de coordenadas que formam as caixas delimitadoras; e a lista contendo os graus de confiança de cada caixa. Essas informações são obtidas através do método “face_detection()”, cuja função é carregar os pesos e as configurações de YOLO V3. Após gerar as caixas delimitadoras, a lista “retangulos” terá o seu tamanho avaliado. Se o tamanho for maior que 0, então há pelo menos uma face na imagem. Ao ocorrer isso, a lista “y_train_predict” recebe o valor 1. Caso contrário, receberá 0, uma vez que não foi identificado um rosto.

A variável “fim” receberá o tempo final de processamento da imagem, sendo que a lista “tempos” receberá a diferença das variáveis “fim” e “inicio”. As listas dos resultados da detecção e do tempo de cada processamento são as saídas da função.

Os gráficos obtidos são mostrados abaixo. A Figura 11.22 é a matriz de confusão de YOLO.

Figura 11.22: Matriz de confusão de YOLO. O algoritmo possui uma elevada acurácia, uma vez que apresentou um grande número de verdadeiros positivos e falsos positivos.



Com base na matriz, a acurácia de YOLO é, aproximadamente, de 91%, conforme o cálculo abaixo:

$$Acurácia = \frac{578 + 329}{578 + 329 + 22 + 71} = 0{,}907$$

As figuras 5.23 e 5.24 mostram a curva ROC e o tempo de detecção de YOLO.

Figura 11.23: Curva ROC de YOLO.

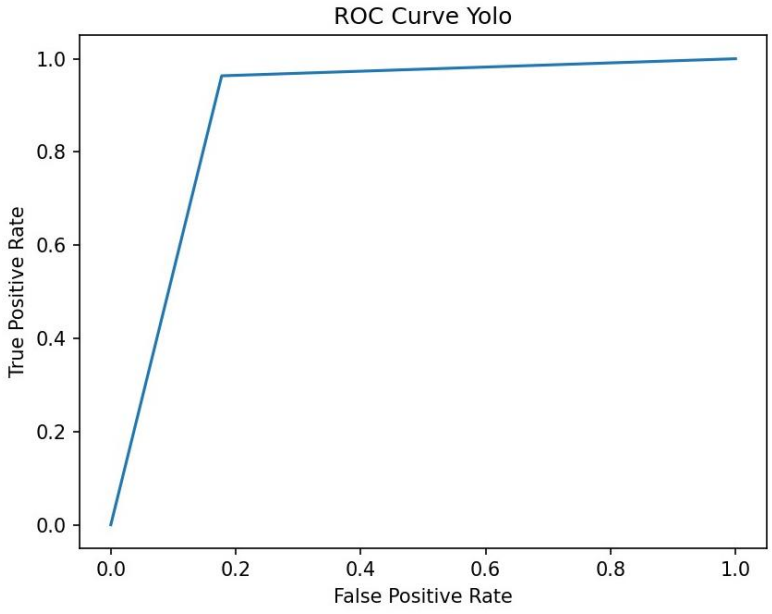
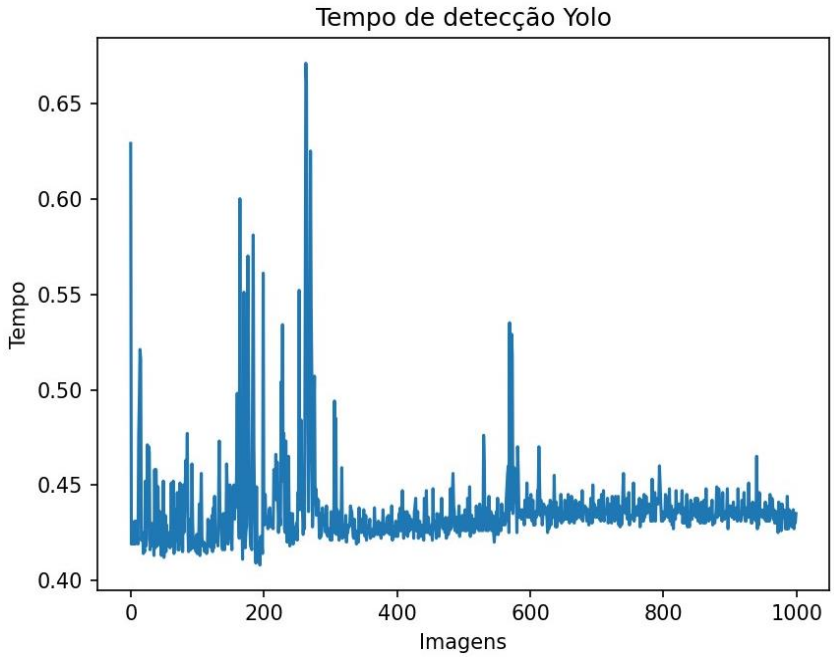


Figura 11.24: Tempo de processamento de YOLO. Na maioria das imagens, o algoritmo levou menos que 0.45 segundos para processar.



YOLO é um detector de objetos que está revolucionando a área de Visão Computacional. Com um processamento rápido e ao mesmo tempo atingindo altas porcentagens de acurácia, este algoritmo tem sido usado em ambientes onde a detecção em tempo real é um requisito fundamental.

Quanto ao seu uso na detecção facial, nota-se que YOLO apresenta um excelente desempenho e acurácia, ao mesmo tempo que é rápido ao processar as imagens. No próximo capítulo, será mostrado um caso real de aplicação utilizando tecnologias de reconhecimento facial, voltado para a área educacional: o SisCoP, Sistema de Controle de Presença que utiliza o reconhecimento facial para a marcação de presença dos estudantes em sala de aula.

12

SISCOP – SISTEMA DE CONTROLE DE PRESENÇA: UM EXEMPLO DE APLICAÇÃO DE RECONHECIMENTO FACIAL

O Reconhecimento facial, que é uma técnica biométrica, tem se tornado uma área de pesquisa muito ativa nos últimos anos. Biometria é a ciência que estabelece a identidade de uma pessoa baseada em seus atributos físicos, químicos ou comportamentais (STAN; ANIL, 2011).

A utilização do reconhecimento facial em detrimento de outras técnicas biométricas proporciona algumas vantagens, as quais podemos enumerar: os traços biométricos da face não podem ser perdidos e nem esquecidos, são difíceis de serem copiados, compartilhados ou distribuídos. Além disso, requer que a pessoa esteja presente na hora e lugar da autenticação. O reconhecimento facial dispensa a utilização de equipamentos especializados, bastando para tanto a utilização de câmeras simples (TOLBA et al 2006; PARKHI et al, 2015).

O reconhecimento facial utilizando computadores (notebooks e desktop) já está bem consolidado e apresenta uma infinidade de abordagens, algoritmos e técnicas bem sedimentadas. Entretanto, a utilização de computadores de baixo custo, como *smartphones*, ainda apresenta um campo fértil para pesquisas e desenvolvimento de novos produtos. É neste contexto que se encaixa o presente projeto, com o desenvolvimento de um aplicativo para celulares (plataformas Android e iOS) e toda infraestrutura de *backend* para suportar o registro de presença em um local específico, utilizando o reconhecimento facial e a geolocalização.

Para tanto, foram traçados os seguintes objetivos: 1) desenvolvimento do aplicativo para as plataformas iOS e Android; 2) desenvolver sistema Web para cadastro e gerenciamento de informações para o registro de presença; 3) pesquisar, analisar e propor termos de utilização do sistema web e do aplicativo SisCoP (política de privacidade) em coerência com a Lei Geral de Proteção de Dados; 4) Testar e validar o aplicativo; 5) Tornar público o código-fonte e toda a infraestrutura com a finalidade de fomentar novas pesquisas na área e contribuir com o desenvolvimento social.

Os resultados e os endereços onde os aplicativos, páginas e sistema podem ser acessados são disponibilizados ao longo do trabalho. A página oficial de detalhamento do projeto pode ser acessada pelo seguinte endereço: http://siscop.com.br.

12.1. Materiais e métodos

Em termos de materiais, ferramentas e linguagens utilizadas para o desenvolvimento do projeto, destacam-se:

- Utilização do framework Flutter e linguagem Dart para desenvolvimento do aplicativo e sua refatoração, para ambas as plataformas (Android e iOS).
- Utilização dos frameworks Laravel e GraphQL e linguagem PHP para desenvolvimento da API.
- Utilização dos frameworks Typescript, Sass, Angular, Webpack e Bootstrap e linguagens Html, CSS e JavaScript para o desenvolvimento da aplicação backend e página web.
- Utilização do banco de dados MySQL no servidor web para guarda e gestão das informações.

Para consecução dos objetivos inicialmente traçados, o presente projeto foi desenvolvido em doze etapas:

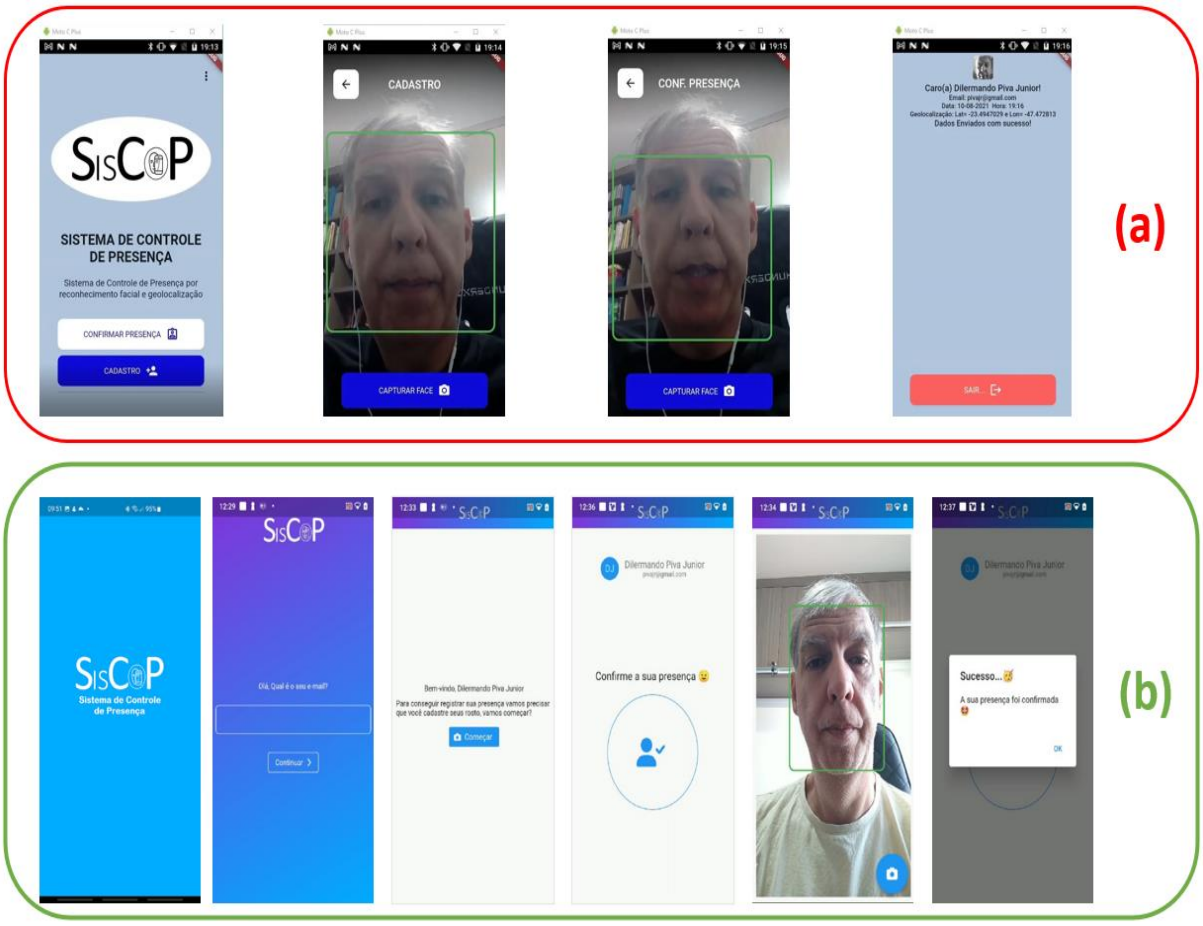
- Etapa 1. Revisão da Literatura: atualização e ampliação do referencial inicial utilizado para o presente projeto.
- Etapa 2. Pesquisa, análise e identificação de Algoritmo(s) de Reconhecimento Facial para Computadores com baixo poder de processamento.
- Etapa 3. Pesquisa, análise e identificação de Algoritmo(s) de Reconhecimento Facial para ambientes não propícios.
- Etapa 4. Elaboração de Arquitetura em Nuvem para suportar o envio, armazenamento e consumo das informações de reconhecimento facial.
- Etapa 5. Desenvolvimento de Sistema específico (app para ambiente Android e servidor) - parte 1.
- Etapa 6. Primeira rodada de Testes do sistema – ambiente educacional.
- Etapa 7. Análise dos dados de teste (parte 1)
- Etapa 8. Desenvolvimento / Ampliação / Melhoria do sistema (plataforma iOS) (parte 2)
- Etapa 9. Segunda rodada de Testes do sistema.
- Etapa 10. Análise dos dados de teste (parte 2) e melhoria no sistema/arquitetura.
- Etapa 11. Disponibilização de versão atualizada do sistema e da arquitetura para a comunidade acadêmica/empresarial.
- Etapa 12. Elaboração de Relatório Técnico Final.

12.2. Resultados e Discussão

Ao longo da pesquisa e do desenvolvimento realizados em 2021, foi proposta uma arquitetura em nuvem e implementado um aplicativo na plataforma Android (Google).

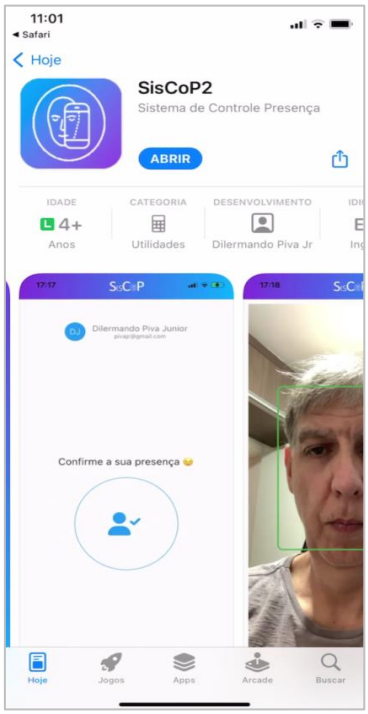
Após a realização dos testes no âmbito acadêmico, observou-se a necessidade de ajustes, principalmente da interface gráfica do aplicativo, para adequação também à plataforma iOS (Apple). O aplicativo foi totalmente refatorado assim como também a arquitetura, que abandonou a utilização do serviço de *backend* do Firebase, adotando uma arquitetura própria, desenvolvida utilizando o framework Laravel e dezenas de outras tecnologias. A Figura 12.1 apresenta a mudança da interface entre as duas versões, sendo (b) a versão final implementada nas duas plataformas (iOS e Android).

Figura 12.1. Refatoração da Interface do aplicativo SisCop. (a) Versão original e (b) Versão final, refatorada para as plataformas iOS e Android.

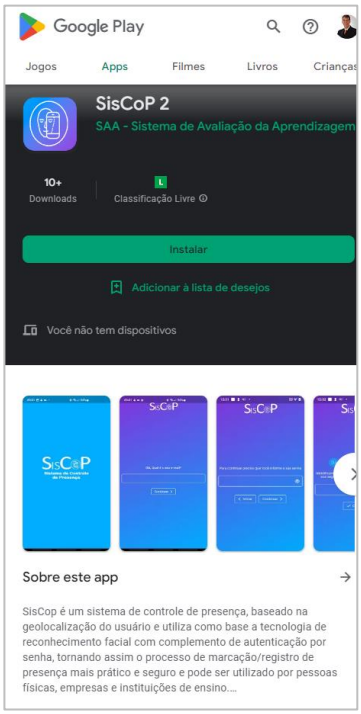


O acesso (*links*) à aplicação nas lojas de aplicativos das duas plataformas encontra-se disponível no endereço https://www.siscop.com.br/page2.html. Este endereço, quando acessado pelo aparelho celular, ao clicar no ícone correspondente a loja de sua preferência, direciona automaticamente ao aplicativo, como pode ser visto na Figura 12.2, (a) o aplicativo SisCoP na Apple Store e (b) o aplicativo na Play Store (Google).

Figura 12.2: O aplicativo SisCoP disponível nas lojas (a) Apple Store e (b) Play Store



(a)



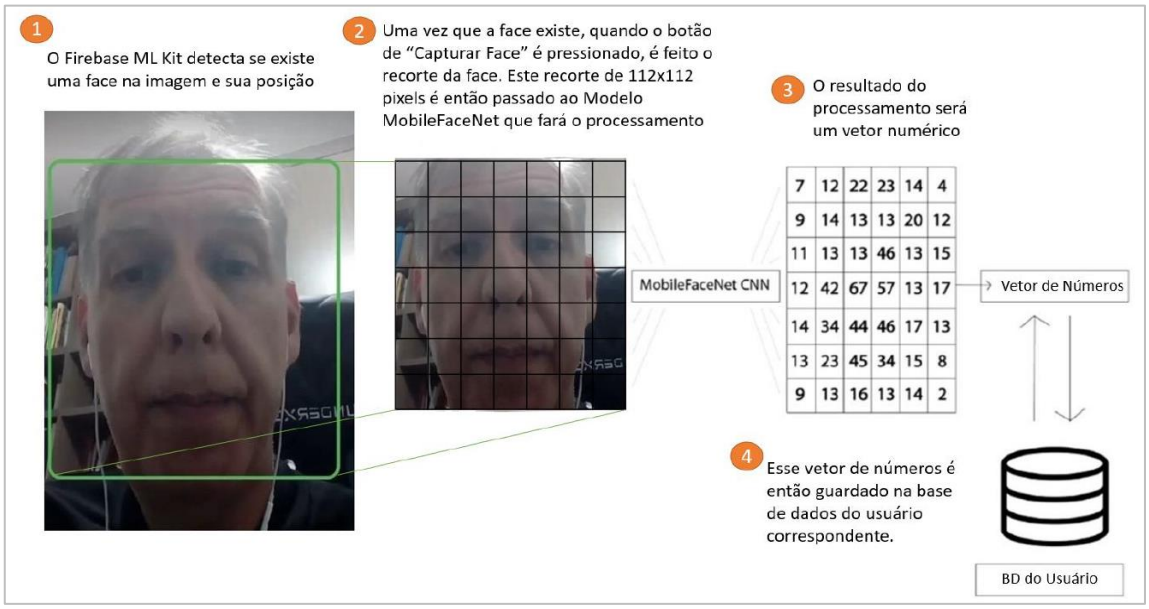
(b)

Em paralelo, ocorreu o estudo, análise e o desenvolvimento da política de privacidade do sistema SisCoP à luz da LGPD (disponível no endereço: https://www.siscop.com.br/page4.html). Esse desenvolvimento e, também, a consulta a advogados e empresas parceiras, levou-nos a descartar a utilização do atual sistema pelas empresas, em virtude da falta do amparo legal para a validação do registro de presença, por exemplo de um funcionário. Isso se deve, basicamente a dois motivos:

1) em virtude da LGPD (RAPÔSO et al, 2019) e para evitar a responsabilização por guarda de informações pessoais e sensíveis, ao invés do envio da imagem capturada da face da pessoa que está validando a presença, adotamos o envio de uma chave (um vetor numérico) gerada a partir do processamento de um recorte da face do usuário (112 x 112 pixels) pelo modelo de rede neural MobileFaceNet (CHEN et al, 2018), utilizando características (features[1]) do rosto da pessoa, o que permite sua distinção de outras faces, possibilitando probabilisticamente inferir que a face do presente usuário é semelhante a imagem (deste mesmo usuário) armazenada inicialmente. O E-mail é utilizado para fazer a busca da face do usuário e depois armazenar de maneira única, as informações de data, hora e geolocalização. A Figura 12.3 apresenta como o vetor numérico que representa a face é gerado a partir do modelo MobileFaceNet.

[1] Features – conjunto de números resultante da análise das características morfológicas da face de uma pessoa, o qual identifica de forma inequívoca aquela pessoa.

Figura 12.3: Processo de geração da matriz de feature a partir de uma face, utilizando o modelo MobileFaceNet.



Todavia, como a inferência ocorre de maneira probabilística, o que acarreta em não 100% de certeza, a justiça não aceita como prova de registro (pelo menos por enquanto). (TEFFÉ; VIOLA, 2020)

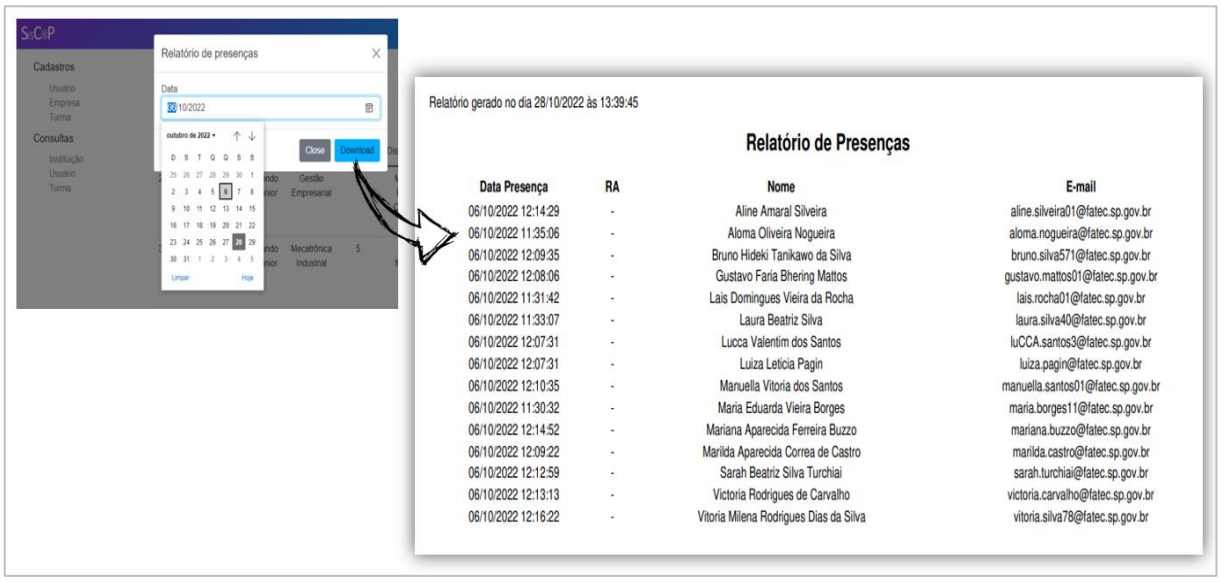
2) A informação de geolocalização dos celulares pode ser, propositadamente, alterada pelo usuário. Dessa forma, não há maneira, pelo menos com a tecnologia utilizada no projeto, de garantir a localização da pessoa que está validando a presença. Por este motivo, a utilização, testes e adequação do sistema para o ambiente empresarial e pessoal foram deixados para um possível desenvolvimento futuro.

Por fim, a API e o sistema Web foram desenvolvidos tomando como base a utilização apenas no ambiente educacional (já que os aspectos descritos no parágrafo acima inviabilizam sua utilização pelas empresas e pessoas físicas). A Figura 12.4 apresenta as telas inicial (login), a principal (menu) e a tela de consulta de turmas da aplicação web.

Figura 12.4: Telas do Sistema SisCoP Web

Uma vez logado no sistema Web, pode-se realizar a consulta da presença dos estudantes em uma determinada data. O sistema apresenta as informações dos estudantes presentes em um relatório PDF. Futuramente, essa consulta pode ser feita, automaticamente, por meio de uma API pelo próprio Sistema de Gestão Acadêmica (SIGA). A Figura 12.5 apresenta a tela de consulta (que aparece ao clicar em um dos botões vermelhos – Presença - da Figura 12.4) e o relatório em PDF dos estudantes presentes em uma determinada data escolhida.

Figura 12.5: Consulta do Relatório de Presença no Sistema SisCoP Web

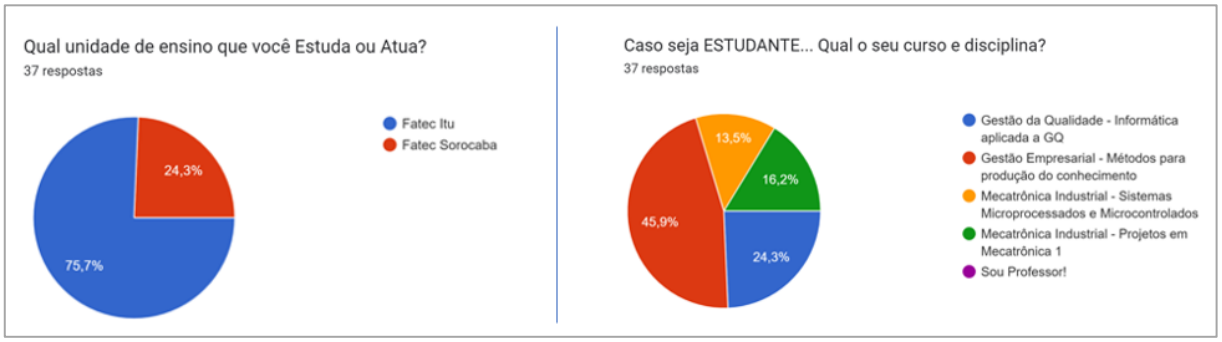


Conforme planejado, o sistema foi testado em quatro turmas em duas Faculdades (Fatec Itu e Fatec Sorocaba). Um total de 54 estudantes foram cadastrados. No período

de teste, o professor responsável realizou, concomitantemente, o registro manual de presença no SIGA. A análise dos dados revelou 100% de compatibilidade entre o registrado manualmente o registrado pelo sistema SisCoP.

Encerrado o período de testes, foi realizada uma pesquisa junto a esses estudantes que utilizaram o sistema com vistas a registrar suas experiências de utilização do aplicativo. Até a data de escrita deste relatório foram registradas 37 respostas (68,5% dos estudantes que foram cadastrados para utilização do aplicativo). A Figura 12.6 apresenta o panorama da origem das respostas, indicando a Faculdade (unidade) e a Turma desses estudantes que responderam a pesquisa.

Figura 12.6: Origem dos estudantes que utilizaram e responderam a pesquisa de avaliação do aplicativo



A Tabela 12.1 apresenta os indicadores avaliados ao longo da pesquisa, a quantidade de respostas separadas pelo grau (variação de 1 a 5), e o percentual de satisfação (bom – 4 e ótimo – 5) dos estudantes na utilização do aplicativo SisCoP.

Tabela 12.1: Resultado da Avaliação do Aplicativo SisCoP realizada pelos estudantes

Indicadores Avaliador	Qtd. De Estudantes por Conceito					% Satisfação
	1	2	3	4	5	Bom (4) e Excelente (5)
Como você avaliaria a instalação do aplicativo? Foi fácil encontrá-lo na Loja de Aplicativos, baixá-lo e instalá-lo?	2	1	0	4	30	92%
Como você avalia a Interface Visual, botões e funcionalidades do APP?	2	0	5	8	22	81%
Como você avalia o processo de Cadastro Inicial?	3	1	3	14	16	81%
Como você avalia o processo de Confirmação de Presença?	1	3	2	14	17	84%
No processo de Confirmação de Presença, você teve alguma dificuldade para efetuar o Reconhecimento Facial?	2	4	9	5	17	59%
De forma geral, como você avalia sua experiência de utilização do sistema SisCoP?	2	2	3	16	14	81%
Depois da utilização do APP SisCoP, qual o grau de recomendação do aplicativo a outros estudantes?	2	1	4	10	20	81%

Como pode ser observado, o processo de confirmação da presença pelo reconhecimento facial, utilizando o aplicativo, ainda tem que ser melhorado, principalmente a questão do impacto da iluminação do ambiente. Muitas vezes isso

dificultava o reconhecimento facial, já que o modelo utilizado cria a matriz de compatibilidade a partir de apenas uma foto/quadro (*one shot*).

Nos demais indicadores, acreditamos que o resultado foi adequado ao tempo e recursos disponíveis para o desenvolvimento do aplicativo.

O aplicativo, a API e o sistema Web foram devidamente registrados junto ao INPI (patente de software de computador) sob número BR 51 2022 002733-0. A Figura 12.7 apresenta o recorte da publicação desse processo de pedido de registro na revista RPI n. 2700 de 04 de outubro de 2022, p. 21.

Figura 12.7: Recorte da Publicação do pedido de Registro (patente de software de computador) junto ao INPI, publicado na Revista RPI n. 2700 de 04/10/2022, p. 21.

Processo: BR 51 2022 002733-0	Código 730 - Expedição do Certificado de Registro Título: Sistema de Controle de Presença (SisCoP) Titular: DILERMANDO PIVA JUNIOR Criador: DILERMANDO PIVA JUNIOR; FRANCISCO DE ASSIS DE FREITAS Linguagem: CSS; FRAMEWORK; HTML; JAVA; JAVA SCRIPT; PHP; PYTHON; SWIFT Campo de Aplicação: AD-01; AD-05; ED-03; IN-02 Tipo de Programa: AP-01; AV-02; IA-01 Data da Criação: 02/09/2022

Por fim, conforme planejado, ao término dos testes e validação do Sistema SisCoP, o código fonte de todo o sistema ficará disponível no repositório GitHub e que poderá ser acessado pelo seguinte endereço: https://github.com/pivajr/SisCoP.

12.3. Considerações Finais do Projeto SisCoP

Quando se observa o objetivo geral deste projeto que era “o desenvolvimento do sistema de controle de presença (SisCoP) com arquitetura em nuvem para gerenciamento das informações”, percebe-se que, diante dos resultados apresentados, o projeto obteve êxito.

Depois da reconstrução da aplicação móvel para adaptação às duas plataformas (Android e iOS) e dos resultados dos testes e avaliação por parte dos estudantes do aplicativo, percebe-se que a decisão em optar pelo framework Flutter e linguagem Dart foi acertada. Assim como também foi acertada, e reduziu drasticamente o tempo de desenvolvimento, a opção de comunicação entre aplicativo e Sistema Web por APIs.

Infelizmente, o sistema não poderá ser utilizado para fins de registro de presença de funcionários e outras circunstâncias em virtude de falta de amparo legal da forma e tecnologia de armazenamento das informações e registro de geolocalização atualmente

utilizados pelo aplicativo. Um possível trabalho futuro poderia ser a resolução deste problema e sua viabilização legal de utilização por empresas e pessoas físicas.

Por fim, mas não menos importante, com o desenvolvimento do presente projeto, além da capacitação e atualização do presente pesquisador, cinco projetos de iniciação científica na mesma área do projeto (inteligência artificial / reconhecimento facial) foram realizados neste período e, também, o desenvolvimento de um curso 100% gratuito de qualificação profissional em linguagem Python, de livre acesso e disponibilizado online, acessível pelo link: http://www.pypro.com.br.

Fica aqui o agradecimento a Fatec Itu e ao Centro Paula Souza pelo apoio e financiamento desta pesquisa.

Referências

ADARSH, Pranav; RATHI, Pratibha; KUMAR, Manoj. *YOLO v3-Tiny: object detection and recognition using one stage improved model*. **2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)**, [S.L.], p. 687-694, mar. 2020. IEEE. http://dx.doi.org/10.1109/icaccs48705.2020.9074315.

Adjabi, I.; Ouahabi, A.; Benzaoui, A.; Taleb-Ahmed, A. "Past, Present, and Future on Face Recognition: a review". Electronics 2020, 9, 1188; doi:10.3390/electronics9081188

AFFONSO, A. A., RODRIGUES, E. L. L., PAIVA, M. S. *High-Boost Weber Local Filter for Precise Eye Localization under Uncontrolled Scenarios*. **Pattern Recognition Letters**, v. 102 , p. 50-57, January 2018. DOI:https://doi.org/10.1016/j.patrec.2017.12.015.

AFFONSO, A.A. *Reconhecimento facial em ambientes não controlados por meio do "High-Boost Weber Descriptor" na região periocular*. **Tese de Doutorado**. Escola de Engenharia de São Carlos – USP, 2018.

AGULLA, et al. *Is My Student at the Other Side? Applying Biometric Web Authentication to E-Learning Environments*. **8th IEEE International Conference on Advanced Learning Technologies (ICALT)**, p. 551-553, 2008.

Ahmed, S. B., Ali, S. F., Ahmad, J., Adnan, M., & Fraz, M. M. (2019). *On the frontiers of pose invariant face recognition: a review. Artificial Intelligence Review.* doi:10.1007/s10462-019-09742-3

Ahmed, T., Das, P., Ali, Md. F., & Mahmud, M.-F. (2020). A Comparative Study on Convolutional Neural Network Based Face Recognition. *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1–5. https://doi.org/10.1109/ICCCNT49239.2020.9225688

AHONEN, T.; HADID, A.; PIETIKAINEN, M. *Face Recognition with Local Binary Patterns*. In: **Proceedings of 8th European Conference on Computer Vision (EECV)**, Prague, Czech Republic, p. 469-481, May 2004. DOI: 10.1007/978-3-540-24670-1_36.

AHONEN, T; HADID, A.; PIETIKAINEN, M. *Face Description with Local Binary Patterns: Application to Face Recognition*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 28, n. 12, p. 2037-2041, December 2006. DOI: 10.1109/TPAMI.2006.244

AKBAR, M. S.; SARKER, P.; MANSOOR, A. T.; Al ASHRAY, A. M.; UDDIN, J. *Face Recognition and RFID Verified Attendance System.* **2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)**. doi:10.1109/iccecome.2018.8658705

Akhtar, Z., & Rattani, A. (2017). *A Face in any Form: New Challenges and Opportunities for Face Recognition Technology. Computer, 50(4), 80–90.* doi:10.1109/mc.2017.119

Almeida, J. M. and Bento, H. (2014). Reconhecimento de padrões através de Eigenfaces disponível em http://www.sinfic.pt/SinficWeb/displayconteudo.do2?numero=44666, Sinfic SA, acessado em 14/10/2021.

AMINI, Alexander. YouTube, 2020. MIT 6.S191 (2020): Convolutional Neural Networks. Disponível em: <https://www.youtube.com/watch?v=iaSUYvmCekI>. Acesso em 22 mar. 2022.

AMORIM, T.T. *Comparação de técnicas de reconhecimento facial no controle de frequência acadêmica*. **Trabalho de Conclusão de Curso**. UniEvangélica, curso de Engenharia da Computação, 2018.

Anwarul, S.; Dahiya, S. "A Compreensive Review on Face Recognition Methods and Factors Affecting Facial Recognition Accuracy". In P. K. Singh et al. (eds.), Proceedings of ICRIC 2019, Lecture Notes in Electrical Engineering 597, p. 495-514. Doi: https://doi.org/10.1007/978-3-030-29407-6_36

AQUINO, São Tomás de. **Comentário à ética e à política de Aristóteles.** Vol.1. Condensado dos comentários de São Tomás de Aquino a Aristóteles. Tradução: Antonio Donato P. Rosa. Paraná: Associação Centro Cultural Hugo de São Vitor, 2020.

ARMSTRONG, Paul. **Dominando as tecnologias disruptivas: aprenda a compreender, avaliar e tomar melhores decisões sobre qualquer tecnologia que possa impactar seu negócio.** Tradução: Afonso Celso da Cunha Serra. São Paulo: Autêntica Business, 2019.

Arsenovic, M., Sladojevic, S., Anderla, A., & Stefanovic, D. (2017). FaceTime—Deep learning based face recognition attendance system. *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*, 000053–000058. https://doi.org/10.1109/SISY.2017.8080587

AVELAR, Adriano. **O que é AUC e ROC nos modelos de *Machine Learning***. 2019. Disponível em: https://medium.com/@eam.avelar/o-que-%C3%A9-auc-e-roc-nos-modelos-de-machine-learning-2e2c4112033d. Acesso em: 26 jun. 2022.

Bassi, P. and Rampazzo, W (2022). Redes Neurais Profundas Triplet Aplicadas a Classificação de Sinais em Interfaces Cérebro-Computador.

BELHUMEUR, P.; HESPANHA, J.; KRIEGMAN, D. *Eigenfaces vs. Fisherfaces: recognition using class specific linear projection*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 19, n. 7, p. 711-720, July 1997. DOI: 10.1109/34.598228.

BELONGIE, Serge; MALIK, Jitendra; PUZICHA, Jan. *Matching Shapes*. **8th IEEE International Conference on Computer Vision**, Vancouver, Canadá, p. 454-461, jul. 2001.

BICEGO, M. et al. *On the Use of SIFT Features for Face Authentication*. In: **Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)**, p. 35, 2006. DOI: 10.1109/CVPRW.2006.149

BISSI, T. D. *Reconhecimento Facial com os algoritmos Eigenfaces e Fisherfaces*. **UNIVERSIDADE FEDERAL DE UBERLÂNDIA**, 2018. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/22158/3/ReconhecimentoFacialAlgotimos.pdf>. Acesso em: 03 nov. 2019.

BLASCHKO, Matthew B.; LAMPERT, Christoph H.. *Learning to Localize Objects with Structured Output Regression*. **Lecture Notes in Computer Science**, [S.L.], p. 2-15, 2008. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-540-88682-2_2.

BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. *YOLOv4: optimal speed and accuracy of object detection*. **Arxiv**, [S.L.], p. 1-17, 2020. ArXiv. http://dx.doi.org/10.48550/ARXIV.2004.10934.

BOLME, D. S. et al. *Visual object tracking using adaptive correlation filters*. **IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, p. 2544-2550, 2010. DOI 10.1109/CVPR.2010.5539960.

BRADSKI, Gary. *The OpenCV Library*. **Dr. Dobb's Journal of Software Tools**, [S. L.], p. 122-125, 1 nov. 2009.

BRASIL, 2023. CÓDIGO CIVIL - LEI Nº 10.406, DE 10 DE JANEIRO DE 2002. Disponível em https://www.planalto.gov.br/ccivil_03/LEIS/2002/L10406compilada.htm Acesso em 02.01.2023.

BRASIL, 2023. CONSTITUIÇÃO DA REPÚBLICA FEDERATIVA DO BRASIL DE 1988. Disponível em https://www.planalto.gov.br/ccivil_03/Constituicao/Constituicao.htm Acesso em 02.01.2023.

BRASIL, 2023. LGPD – LEI GERAL DE PROTEÇÃO DE DADOS PESSOAIS - LEI Nº 13.709, DE 14 DE AGOSTO DE 2018. Disponível em https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm Acesso em 03.01.2023.

BRASIL, 2023. MARCO CIVIL DA INTERNET - LEI Nº 12.965, DE 23 DE ABRIL DE 2014. Disponível em https://www.planalto.gov.br/ccivil_03/_ato2011-2014/2014/lei/l12965.htm Acesso em 02.01.2023.

BUDDHARAJU, P.; PAVLIDIS, I. T.; TSIAMYRTZIS, P.; BAZAKOS, M., *Physiology-Based Face Recognition in the Thermal Infrared Spectrum*, **IEEE Trans.PAMI**, Vol. 29, No. 4, pp. 613-626, April 2007. DOI: 10.1109/tpami.2007.1007

CAMMOUN, L. *et al*. *A Review of Tensors and Tensor Signal Processing*. **Tensors in Image Processing and Computer Vision**, [S.L.], p. 1-32, 2009. Springer London. http://dx.doi.org/10.1007/978-1-84882-299-3_1.

CAO, F.; WANG, M.; WANG, K. *The Classroom Attendance Management System of Face Recognition Based on LBS*. **Proceedings of the 2018 5th International Conference on Education, Management, Arts, Economics and Social Science (ICEMAESS 2018)**. Doi: 10.2991/icemaess-18.2018.188.

Çarikçi, M. and Özen, F. (2012. Face Recognition System Based on Eigenfaces Method - Istambul, Turquia, SciVerse ScienceDirect.

CHANDEL, Himanshu; VATTA, Sonia. *A Vision based Vehicle Detection System*. **Communications on Applied Electronics (CAE)**, Nova York, EUA, v. 2, n. 6, p. 6-16, ago. 2015.

CHARÃO, Daniele. **O reconhecimento facial vai afetar a publicidade e a sua vida.** Escrito em 15.05.2018. Disponível em https://blog.runrun.it/reconhecimento-facial/ Acesso em 18.01.2023.

CHAURASIA, Ayush. YouTube, 2019. Haar Cascade, Adaboost and Integral Image - ML4Face-detection Part 2. Disponível em: <https://www.youtube.com/watch?v=ANeuqpI_VcY>. Acesso em: 18 nov. 2021.

CHAVES, Maria Cecília Sandoval. **O Reconhecimento Facial e a LGPD.** Data: 16/12/2022. Almeida Prado Hoffmann Advogados. Disponível em https://aphoffmann.com.br/o-reconhecimento-facial-e-a-lgpd/ Acesso em 22.01.2023.

CHEN, J. et al. *WLD: A Robust Local Image Descriptor*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 32, n. 9, p. 1705-1720, August 2009. DOI: 10.1109/TPAMI.2009.155.

Chen, J., Zhang, Z., Yao, L., Li, B., & Chen, T. (2019. Face Recognition Using Depth Images Base Convolutional Neural Network. *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, 1–4. https://doi.org/10.1109/CITS.2019.8862099

CHEN, S.; LIU, Y.; GAO, X.; HAN, Z. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. In: Zhou J. et al. (eds Biometric Recognition. **CCBR 2018. Lecture Notes in Computer Science**, vol 10996. Springer, Cham. https://doi.org/10.1007/978-3-319-97909-0_46

CHIACHIA, G.; PENTEADO, B.E.; MARANA, A.N. *Fusão de métodos de reconhecimento facial através da otimização por enxame de partículas*. **UNESP - Faculdade de Ciências**, Brasil: Bauru – São Paulo, 2003.

CODEBASICS. YouTube, 2020. Simple explanation of convolutional neural network | Deep Learning Tutorial 23 (Tensorflow & Python). Disponível em: <https://www.youtube.com/watch?v=zfiSAzpy9NM>. Acesso em 22 mar. 2022.

COLLINS, Bob *et al*. ***Image Gradients and Gradient Filtering***. [S. L.], 2017. 43 slides, color. Disponível em: https://www.cs.cmu.edu/~16385/s17/Slides/4.0_Image_Gradients_and_Gradient_Filtering.pdf. Acesso em: 14 nov. 2021.

COMPUTERPHILE. YouTube, 2016. CNN: Convolutional Neural Networks Explained - Computerphile. Disponível em: <https://www.youtube.com/watch?v=py5byOOHZM8>. Acesso em 22 mar. 2022.

CONTRIBUIDORES DA WIKIPEDIA. Wikipedia, The Free Encyclopedia, 2021. Haar wavelet. Disponível em: <https://en.wikipedia.org/w/index.php?title=Haar_wavelet&oldid=1054878822>. Acesso em: 18 nov. 2021.

CONTRIBUIDORES DA WIKIPEDIA. Wikipedia, The Free Encyclopedia, 2021. Haar-like feature. Disponível em: <https://en.wikipedia.org/wiki/Haar-like_feature>. Acesso em: 18 nov. 2021.

CONTRIBUIDORES DA WIKIPEDIA. Wikipedia, The Free Encyclopedia, 2021. OpenCV. Disponível em: <https://pt.wikipedia.org/wiki/OpenCV>. Acesso em: 18 nov. 2021.

CONTRIBUIDORES DA WIKIPEDIA. Wikipedia, The Free Encyclopedia, 2020. Rede neural convolucional. Disponível em: <https://pt.wikipedia.org/wiki/Rede_neural_convolucional>. Acesso em 22 mar. 2022.

CROW, F.C. *Summed-area tables for texture mapping*. **Computer Graphics**, v. 18, n. 3, July, 1983. Disponível em: <http://www.florian-oeser.de/wordpress/wp-content/2012/10/crow-1984.pdf>. Acessado em: 6/4/2020.

DALAL, N.; TRIGGS, B.. *Histograms of Oriented Gradients for Human Detection*. **2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)**, [S.L.], p. 886-893, 2005. IEEE. http://dx.doi.org/10.1109/cvpr.2005.177.

Dalla, B. P. (2007. PCA-tree: uma proposta para indexação multidimensional.

DATA SCIENCE ACADEMY. ***Deep Learning Book***, 2021. Capítulo 41 – Campos Receptivos Locais em Redes Neurais Convolucionais. Disponível em: <https://www.deeplearningbook.com.br/campos-receptivos-locais-em-redes-neurais-convolucionais/>. Acesso em 22 mar. 2022.

DATA SCIENCE ACADEMY. ***Deep Learning Book***, 2021. Capítulo 42 – Compartilhamento de Pesos em Redes Neurais Convolucionais. Disponível em: <https://www.deeplearningbook.com.br/compartilhamento-de-pesos-em-redes-neurais-convolucionais/>. Acesso em 22 mar. 2022.

DATA SCIENCE ACADEMY. ***Deep Learning Book***, 2021. Capítulo 43 – Camadas de Pooling em Redes Neurais Convolucionais. Disponível em: <https://www.deeplearningbook.com.br/camadas-de-pooling-em-redes-neurais-convolucionais/>. Acesso em 22 mar. 2022.

DATA SCIENCE ACADEMY. ***Deep Learning Book***, 2021. Capítulo 44 – Reconhecimento de Imagens com Redes Neurais Convolucionais em Python – Parte 1. Disponível em: <https://www.deeplearningbook.com.br/reconhecimento-de-imagens-com-redes-neurais-convolucionais-em-python-parte-1/>. Acesso em 22 mar. 2022.

DATA SCIENCE ACADEMY. ***Deep Learning Book***, 2021. Capítulo 45 – Reconhecimento de Imagens com Redes Neurais Convolucionais em Python – Parte 2. Disponível em: <https://www.deeplearningbook.com.br/reconhecimento-de-imagens-com-redes-neurais-convolucionais-em-python-parte-2/>. Acesso em 22 mar. 2022.

DATA SCIENCE ACADEMY. ***Deep Learning Book***, 2021. Capítulo 46 – Reconhecimento de Imagens com Redes Neurais Convolucionais em Python – Parte 3. Disponível em: <https://www.deeplearningbook.com.br/reconhecimento-de-imagens-com-redes-neurais-convolucionais-em-python-parte-3/>. Acesso em 22 mar. 2022.

DATA SCIENCE ACADEMY. ***Deep Learning Book***, 2021. Capítulo 47 – Reconhecimento de Imagens com Redes Neurais Convolucionais em Python – Parte 4. Disponível em: <https://www.deeplearningbook.com.br/reconhecimento-de-imagens-com-redes-neurais-convolucionais-em-python-parte-4/>. Acesso em 22 mar. 2022.

DATA SCIENCE ACADEMY. ***Deep Learning Book***. 2021. Disponível em: https://www.deeplearningbook.com.br/o-que-sao-redes-neurais-artificiais-profundas/. Acesso em: 01 fev. 2022.

Debey, A.K.; Jain, V. "A Review of Face Recognition Methods Using Deep Learning Networks", Journal of Information and Optimization Sciences, 40:2, 547-558, 2019. doi: 10.1080/02522667.2019.1582875

DEEPLIZARD. YouTube, 2017. Convolutional Neural Networks (CNNs) explained. Disponível em: <https://www.youtube.com/watch?v=YRhxdVk_sIs>. Acesso em 22 mar. 2022.

DÉNIZ, O.; BUENO, G.; SALIDO, J.; DE LA TORRE, F. *Face recognition using Histograms of Oriented Gradients.* **Pattern Recognition Letters***, 32(12), 1598–1603.* 2011. doi:10.1016/j.patrec.2011.01.004

DIKEVAR, Ankur. YouTube, 2014. Haar Cascade Visualization. Disponível em: <https://www.youtube.com/watch?v=hPCTwxF0qf4>. Acesso em: 18 nov. 2021.

DING, Xiaohan et al. *RepVGG: making vgg-style convnets great again*. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), [S.L.], p. 13733-13742, jun. 2021. IEEE. http://dx.doi.org/10.1109/cvpr46437.2021.01352.

DINH, Laurent. *Reparametrization in Deep Learning*. **Ph.D.'s Thesis**, Montreal University, 2018.

DO, T.-T.; KIJAK, E. *Face recognition using Co-occurrence Histograms of Oriented Gradients*. **2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).** 2012. doi:10.1109/icassp.2012.6288128

DONEDA, Danilo. **Da privacidade à proteção dos dados pessoais.** Rio de Janeiro: Renovar, 2006.

DUDA, R.O.; HART, P.E.; STORK, D.G. **Pattern classification**. John Wileyi & Sons, 2001.

EQUIPE ALIGER. Aliger – A Empresa de Inteligência das Coisas, 2019. As Redes Neurais Convolucionais no Deep Learning. Disponível em: <https://www.aliger.com.br/blog/as-redes-neuronais-convolutivas-no-deep-learning/>. Acesso em 22. mar 2022.

EVERINGHAM, Mark *et al*. *The Pascal Visual Object Classes (VOC) Challenge*. **International Journal of Computer Vision**, [S.L.], v. 88, n. 2, p. 303-338, 9 set. 2009. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s11263-009-0275-4.

FABIEN, Maël. ***A guide to Face Detection in Python (with code)***. 2019. Disponível em: https://towardsdatascience.com/a-guide-to-face-detection-in-python-3eab0f6b9fc1. Acesso em: 10 jan. 2022.

FELZENSZWALB, Pedro F.; GIRSHICK, Ross B.; MCALLESTER, David. *Cascade object detection with deformable part models*. **2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, [S.L.], p. 2241-2248, jun. 2010. IEEE. http://dx.doi.org/10.1109/cvpr.2010.5539906.

FERREIRA, A.S. *Redes Neurais Convolucionais Profundas na detecção de plantas daninhas em lavouras de soja*. **Dissertação de Mestrado**. Universidade Federal do Mato Grosso do Sul. Curso de Ciência da Computação, Campo Grande, 2017.

Firoze, A.; Deb, T. “Face Recognition Time Reduction Based on Partitioned Faces without Compromising Accuracy and a Review of state-of-art Face Recognition Approaches”. ICIGP 2018, ACM ISBN 978-1-4503-6367-9. Doi: https://doi.org/10.1145/3191442.3191467

FIRST PRINCIPLES OF COMPUTER VISION. YouTube 2021. **First Principles of Computer Vision**. Disponível em: <https://www.youtube.com/watch?v=1EJ84QqkxWc>. Acesso em: 18 nov. 2021.

FIRST PRINCIPLES OF COMPUTER VISION. YouTube 2021. **Haar Features for Face Detection | Face Detection**. Disponível em: <https://www.youtube.com/watch?v=ZSqg-fZJ9tQ>. Acesso em: 18 nov. 2021.

FISHER, Robert *et al*. ***Gaussian Smoothing***. 2000. Disponível em: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm. Acesso em: 13 nov. 2021.

FORSYTH, David A.; PONCE, Jean. ***Computer Vision:*** *a modern approach*. 2. ed. New Jersey: Pearson Education, 2012. 761 p.

FORUM, 2019. **Comitiva do PSL vai à China para conhecer tecnologia de reconhecimento facial do Partido Comunista.** POLÍTICA - 16/1/2019 – 14h07. Disponível em https://revistaforum.com.br/politica/2019/1/16/comitiva-do-psl-vai-china-para-conhecer-tecnologia-de-reconhecimento-facial-do-partido-comunista-37768.html Acesso em 25.01.2023.

FRANCISCO, Pedro Augusto P. HUREL, Louise Marie. RIELLI, Mariana Marques. **Regulação do reconhecimento facial no setor público: avaliação de experiências internacionais.** INSTITUTO IGARAPÉ e DATA PRIVACY BRASIL RESEARCH - JUNHO 2020. Disponível em https://igarape.org.br/wp-content/uploads/2020/06/2020-06-09-Regula%C3%A7%C3%A3o-do-reconhecimento-facial-no-setor-p%C3%BAblico.pdf Acesso em 20.01.2023.

FREEMAN, W.T. *et al*. *Computer vision for computer games*. **Proceedings of the Second International Conference on Automatic Face and Gesture Recognition**, [S.L.], p. 100-105, 1996. IEEE Comput. Soc. Press. http://dx.doi.org/10.1109/afgr.1996.557250.

FREEMAN, William T.; ROTH, Michal. *Orientation Histograms for Hand Gesture Recognition*. **IEEE Intl. Wkshp. on Automatic Face and Gesture Recognition**, Zurique, Suíça, p. 296-301, jun. 1995.

Fu, T.-C., Chiu, W.-C., & Wang, Y.-C. F. (2017). Learning guided convolutional neural networks for cross-resolution face recognition. *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, 1–5. https://doi.org/10.1109/MLSP.2017.8168180

Fujikawa, C. S. (2016). Reconhecimento Facial utilizando Descritores de Textura e Aprendizado Não Supervisionado - Unicamp Rio Claro.

FUTUROLOGY — AN OPTIMISTIC FUTURE. YouTube, 2020. Convolutional Neural Networks Explained (CNN Visualized). Disponível em: <https://www.youtube.com/watch?v=pj9-rr1wDhM>. Acesso em 22 mar. 2022.

Gao, S., Zeng, C., Bai, M., & Shu, K. (2020). Facial Ethnicity Recognition Based on Transfer Learning from Deep Convolutional Networks. *2020 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, 310–314. https://doi.org/10.1109/AEMCSE50948.2020.00073

GENG, C.; JIANG, X. *SIFT features for face recognition*. In: **IEEE International Conference on Computer Science and Information Technology (ICCSIT)**, 2nd, p. 598-602, 2009. DOI: 10.1109/ICCSIT.2009.5234877.

GEORGIOU, Theodoros *et al*. *A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision*. **International Journal of Multimedia Information Retrieval**, [S.L.],

v. 9, n. 3, p. 135-170, 22 nov. 2019. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s13735-019-00183-w.

GHOSAL, V.; TIKMANI, P.; GUPTA, P. *Face Classification Using Gabor Wavelets and Random Forest*. **In: Proceedings of the Canadian Conference on Computer and Robot Vision (CRV '09), IEEE Computer Society**. Washington, DC, USA, 68-73, 2009. DOI: 10.1109/crv.2009.10

GIRSHICK, Ross *et al*. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*. **2014 IEEE Conference on Computer Vision and Pattern Recognition**, [S.L.], p. 580-587, jun. 2014. IEEE. http://dx.doi.org/10.1109/cvpr.2014.81.

GIRSHICK, Ross. *Fast R-CNN*. **2015 IEEE International Conference on Computer Vision (ICCV)**, [S.L.], p. 1440-1448, dez. 2015. IEEE. http://dx.doi.org/10.1109/iccv.2015.169.

GLOBAL SOFTWARE SUPPORT. YouTube, 2018. Computer Vision - Haar-Features. Disponível em: <https://www.youtube.com/watch?v=F5rysk51txQ>. Acesso em: 18 nov. 2021.

GOLDSTEIN, A.J.; HARMON, L.D.; LESK, A.B.. *Identification of human faces*. **Proceedings of the IEEE**, [S.L.], v. 59, n. 5, p. 748-760, 1971. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/proc.1971.8254.

GONG, D.; LI, S.; XIANG, Y. *Face recognition using the Weber Local Descr*iptor. In: **Asian Conference on Pattern Recognition (ACPR)**, p. 589-592, 2011. DOI: 10.1109/ACPR.2011.6166675.

GOODFELLOW, I; BENGIO, Y; COURVILLE, A. **Deep Learning**. MIT Press, 2016. Disponível em: < http://www.deeplearningbook.org> Acessado em: 7/04/2020.

GRAU, Eros Roberto. **Canotilho e a Constituição Dirigente.** Jacinto N. M. Coutinho (org.). Rio de Janeiro: Renovar, 2005.

Grigsby, S.S. (2018). Artificial Intelligence for Advanced Human-Machine Symbiosis. In: Schmorrow, D., Fidopiastis, C. (eds) Augmented Cognition: Intelligent Technologies. AC 2018. Lecture Notes in Computer Science, vol 10915. Springer, Cham.

GUO, Chong Yun; LI, Jian Ping. *Development and future of wavelet analysis*. **2013 10th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)**, [S.L.], p. 335-338, dez. 2013. IEEE. http://dx.doi.org/10.1109/iccwamtip.2013.6716661.

HADID, A.; PIETIKAINEN, M.; AHONEN, T. *A discriminative feature space for detecting and recognizing faces*. In: **Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 797-804, 2004. DOI: 10.1109/CVPR.2004.1315246.

HAFEMANN, L.G. *An analysis of Deep Neural Networks for Texture Classification*. **Dissertação de Mestrado**. Universidade Federal do Paraná. Programa de Informática. Curitiba, 2014.

HANSEN, L.A.; VIDAL, F.B.; RALHA, C.G. *Desenvolvimento de módulo de reconhecimento facial para sistema de informação acadêmica*. **Brazilian Journals of Business**, Curitiba, v. 1, n. 3, p.1155-1165, jul/set, 2019.

HARRIS, Charles R. *et al*. *Array programming with NumPy*. **Nature**, [S.L.], v. 585, n. 7825, p. 357-362, 16 set. 2020. Springer Science and Business Media LLC. http://dx.doi.org/10.1038/s41586-020-2649-2.

HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. ***The Elements of Statistical Learning:*** *Data Mining, Inference, and Prediction*. 2. ed. [S.L.] Springer, 2017. p. 745

HAYKIN, Simon. **Redes Neurais:** princípios e práticas. 2. ed. São Paulo: Bookman, 2008. 899 p.

HAZZAN, Samuel. **Fundamentos da Matemática Elementar, 5**: combinatória, probabilidade. 8. ed. São Paulo: Atual, 2013. 204 p.

HE, Kaiming *et al*. *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, [S.L.], v. 37, n. 9, p. 1904-1916, 1 set. 2015. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/tpami.2015.2389824.

Hong, Z.-Q. (1991). *Algebraic feature extraction of image for recognition. Pattern Recognition, 24(3), 211–219.* doi:10.1016/0031-3203(91)90063-b

HUANG, D. et al. *Local Binary Patterns and Its Application to Facial Image Analysis: A Survey*. **IEEE Transactions on Systems, Man, and Cybernetics**, Part C: Applications and Reviews, v. 41, n. 6, p. 765-781, March 2011. DOI: 10.1109/TSMCC.2011.2118750.

HUNTER, John D.. *Matplotlib: a 2D graphics environment.* **Computing in Science & Engineering**, [S.L.], v. 9, n. 3, p. 90-95, 2007. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mcse.2007.55.

IBM TECHNOLOGY. YouTube, 2021. What are Convolutional Neural Networks (CNNs)?. Disponível em: <https://www.youtube.com/watch?v=QzY57FaENXg>. Acesso em 22 mar. 2022.

IDEC – Instituto Brasileiro de Defesa do Consumidor (2020). **Após denúncia do Idec, Hering é condenada por uso de reconhecimento facial: secretaria Nacional do Consumidor condenou a empresa ao pagamento de multa de R$ 58,7 mil por violações ao CDC.** Disponível em https://idec.org.br/noticia/apos-denuncia-do-idec-hering-e-condenada-por-uso-de-reconhecimento-facial Acesso em 24.01.2023.

IDEC – Instituto Brasileiro de Defesa do Consumidor (2020). **Idec obtém vitória contra reconhecimento de emoções no Metrô de SP: Decisão inédita no País proíbe a captura de dados realizada por câmeras da ViaQuatro e aplica multa de R$ 100 mil em concessionária.** INTERNET, TELEFONIA E TV. 10/05/2021. Atualizado: 22/07/2021. Disponível em https://idec.org.br/noticia/idec-obtem-vitoria-contra-reconhecimento-de-emocoes-no-metro-de-sp Acesso em 24.01.2023.

IDEC – Instituto Brasileiro de Defesa do Consumidor (2020). **Reconhecimento facial: a banalização de uma tecnologia controversa.** Disponível em https://idec.org.br/idec-na-imprensa/reconhecimento-facial-banalizacao-de-uma-tecnologia-controversa Acesso em 26.12.2022.

INSTITUTO IGARAPÉ (2019). **Infográfico: Desde 2011 vem sendo utilizado o Reconhecimento Facial no Brasil.** Disponível em https://igarape.org.br/infografico-reconhecimento-facial-no-brasil/ Acesso em 22.01.2023.

IOFFE, Sergey; SZEGEDY, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. **arXiv:1502.03167 [cs]**, [S.L.], 2 mar. 2015.

ITO, Joichi. HOWE, Jeff. **Disrupção e inovação: como sobreviver ao futuro incerto.** Tradução: Carlos Bacci. Rio de Janeiro: Alta Books, 2018.

Jafri, S., Chawan, S., & Khan, A. (2020). Face Recognition using Deep Neural Network with "LivenessNet." *2020 International Conference on Inventive Computation Technologies (ICICT)*, 145–148. https://doi.org/10.1109/ICICT48043.2020.9112543

JAIN, A.K.; FLYNN, P.;ROSS, A.A., **Handbook of Biometrics**. http://www.springer.com/us/book/9780387710402: Springer US, 2008.

JAIN, Anil K.. ***Fundamentals of Digital Image Processing***. Englewood Cliffs, EUA: Prentice Hall, 1989. 569 p. (Prentice Hall Information and System Sciences Series).

JIANG, Peiyuan *et al*. *A Review of Yolo Algorithm Developments*. **Procedia Computer Science**, [S.L.], v. 199, p. 1066-1073, 2022. Elsevier BV. http://dx.doi.org/10.1016/j.procs.2022.01.135.

JOCHER, Glenn. **YOLOv5**. 2020. Disponível em: https://github.com/ultralytics/yolov5. Acesso em: 18 mar. 2022.

JONNALAGADDA, Venkata Krishna. ***Object Detection YOLO v1 , v2, v3***. Disponível em: https://medium.com/@venkatakrishna.jonnalagadda/object-detection-yolo-v1-v2-v3-c3d5eca2312a. Acesso em: 11 mar. 2022.

JUEFEI-XU, F.; LUU, K.; SAVVIDES, M. *Spartans: Single-Sample Periocular-Based Alignment-Robust Recognition Technique Applied to Non-Frontal Scenarios*. **IEEE Transactions on Image Processing**, v. 24, Issue 12, August 2015. DOI: 10.1109/TIP.2015.2468173

JURASZEK, G.D. *Reconhecimento de produtos por imagem utilizando palavras visuais e redes neurais convolucionais*. **Dissertação de Mestrado**. UDESC – Universidade do Estado de Santa Catarina. Metrado em Computação Aplicada. Joinville, 2014.

Kadambari, S., Prabhu, G., Mistry, D., & Khanore, M. (2019). *Automation of Attendance System Using Facial Recognition***. 2019 International Conference on Advances in Computing***, Communication and Control (ICAC3).* doi:10.1109/icac347590.2019.9036819

KANADE, T. *Picture Processing by Computer Complex and Recognition of Human Faces*. **Ph.D. dissertation**, Kyoto University, 1973.

KANG, Pankaj; SINGH, Atul. ***Understanding Image Gradients***. 2019. Disponível em: https://theailearner.com/2019/05/11/understanding-image-gradients/. Acesso em: 10 nov. 2021.

Ke, P., Cai, M., Wang, H., & Chen, J. (2018). A novel face recognition algorithm based on the combination of LBP and CNN. *2018 14th IEEE International Conference on Signal Processing (ICSP)*, 539–543. https://doi.org/10.1109/ICSP.2018.8652477

Khan, S., Ahmed, E., Javed, M. H., A Shah, S. A., & Ali, S. U. (2019). Transfer Learning of a Neural Network Using Deep Learning to Perform Face Recognition. **2019 International Conference on Electrical, Communication, and Computer Engineering** *(ICECCE)*, 1–5. https://doi.org/10.1109/ICECCE47252.2019.8940754

KING, Davis E.. *Dlib-ml: a machine learning toolkit*. **Journal of Machine Learning Research 10**, Baltimore, EUA, p. 1755-1758, jul. 2009.

KINSLEY, Harrison. YouTube, 2016. **Creating your own Haar Cascade OpenCV Python Tutorial**. Disponível em: <https://www.youtube.com/watch?v=jG3bu0tjFbk>. Acesso em: 18 nov. 2021.

KINSLEY, Harrison. YouTube, 2016. **Haar Cascade Object Detection Face & Eye - OpenCV with Python for Image and Video Analysis 16**. Disponível em: <https://www.youtube.com/watch?v=88HdqNDQsEk>. Acesso em: 18 nov. 2021.

KINSLEY, Harrison. YouTube, 2016. **Making your own Haar Cascade Intro - OpenCV with Python for Image and Video Analysis 17**. Disponível em: <https://www.youtube.com/watch?v=jG3bu0tjFbk>. Acesso em: 18 nov. 2021.

KONG, S. G et. al, *Recent advances in visual and infrared face recognition—a review*, **Computer Vision and Image Understanding** *97*, p. 103–135, 2005. DOI: 10.1016/j.cviu.2004.04.001

KSHIRSAGAR, V. P.; BAVISKAR, M. R.; GAIKWAD, M. E. *Face recognition using Eigenfaces*. **Computer Research and Development (ICCRD), 3rd International Conference on**, vol. 2, no., pp. 302-306, 11-13, 2011.DOI: 10.1109/ICCRD.2011.5764137

KUMAR, N. et al. *Describable Visual Attributes for Face Verification and Image Search*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 33, n.10, p. 1962-1977, March 2011. DOI: 10.1109/TPAMI.2011.48.

LECUN, Y. *Generalization and network design strategies*. **Technical Report CRG-TR-89-4**. University of Toronto, 326-345, 1989.

LI, Chuyi et al. *YOLOv6: a single-stage object detection framework for industrial applications*. **a**rXiv, [S. L.], p. 1-17, 7 set. 2022. https://doi.org/10.48550/arXiv.2209.02976.

Li, Haifeng, & Zhu, X. (2016). Face recognition technology research and implementation based on mobile phone system. *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 972–976. https://doi.org/10.1109/FSKD.2016.7603310

Li, Hao, & Li, G. (2019). Research on Facial Expression Recognition Based on LBP and DeepLearning. *2019 International Conference on Robots & Intelligent System (ICRIS)*, 94–97. https://doi.org/10.1109/ICRIS.2019.00032

LI, Z.; PARK, U.; JAIN, A. *A Discriminative Model for Age Invariant Face Recognition*. **IEEE Transactions on Information Forensics and Security**, v. 6, n. 3, p. 1028-1037, May 2011. DOI: 10.1109/TIFS.2011.2156787.

Littlestone, N. "Learning Quickly when Irrelevant Attributes Abound: A New Linear-Threshold Algorithm," **Machine Learning**, vol. 2, pp. 285-318, 1988.

Liu, L. (2019). Human Face Expression Recognition Based on Deep Learning-Deep Convolutional Neural Network. *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 221–224. https://doi.org/10.1109/ICSGEA.2019.00058

Liu, S., Tang, X., & Wang, D. (2020). Facial Expression Recognition Based on Sobel Operator and Improved CNNSVM. *2020 IEEE 3rd International Conference on Information Communication and Signal Processing (ICICSP)*, 236–240. https://doi.org/10.1109/ICICSP50920.2020.9232063

LIU, Shu *et al*. *Path Aggregation Network for Instance Segmentation*. **arXiv**, [S.L.], p. 1-17, 2018. ArXiv. http://dx.doi.org/10.48550/ARXIV.1803.01534.

LOWE, David G.. *Distinctive Image Features from Scale-Invariant Keypoints*. I**nternational Journal of Computer Vision**, [S.L.], v. 60, n. 2, p. 91-110, nov. 2004. Springer Science and Business Media LLC. http://dx.doi.org/10.1023/b:visi.0000029664.99615.94.

LUO, J. et al. *Person-Specific SIFT Features for Face Recognition*. In: **IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)** vol. 2, p. 593-596, 2007. DOI: 10.1109/ICASSP.2007.366305.

MAJUMDAR, A.; WARD, R. *Discriminative SIFT features for face recognition*. In: **Canadian Conference on Electrical and Computer Engineering (CCECE)**, 9th, p. 27-30, 2009. DOI: 10.1109/CCECE.2009.5090085.

MALLAT, S.G.. *A theory for multiresolution signal decomposition: the wavelet representation*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, [S.L.], v. 11, n. 7, p. 674-693, jul. 1989. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/34.192463.

MALLICK, S. *Histogram of Oriented Gradients.* **Learn OpenCV**, 6 dez. 2016. Disponivel em: <https://www.learnopencv.com/histogram-of-oriented-gradients/>. Acesso em: 31 out. 2019.

MATIAS, Eduardo Felipe Pérez. (Coordenação) **Marco Legal das Startups: Lei Complementar 182/2021 e o fomento ao empreendedorismo inovador no Brasil.** São Paulo: Tompson Reuters Brasil, 2021.

MÁXIMO, Antônio; ALVARENGA, Beatriz. **Curso de Física**. 4. ed. São Paulo: Scipione, 1997. 906 p.

MCCONNELL, R. K.. ***Method of and apparatus for pattern recognition***. . US n. 4567610. Depósito: 22 jul. 1982. Concessão: 28 jan. 1986.

MILLER, P. et al. *Performance Evaluation of Local Appearance Based Periocular Recognition*. In: **IEEE International Conference on Biometrics: Theory Applications and Systems (BTAS)**, p. 1-6, 2010. DOI: 10.1109/BTAS.2010.5634536

MISHRA, Aditya. ***Metrics to Evaluate your Machine Learning Algorithm***. 2018. Disponível em: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234. Acesso em: 26 jun. 2022.

MITCHELL, Tom M.. ***Machine Learning***. [S.L.]: McGraw-Hill Education, 1997. 414 p.

MITTAL, Aditya. Medium – Where good ideas find you, 2021. Haar Cascades, Explained. Disponível em: <https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d>. Acesso em: 18 nov. 2021.

Mocanu, B., Tapu, R., & Zaharia, T. (2019). Design of a CNN Face Recognition System Dedicated to Blinds. **2019 IEEE International Conference on Consumer Electronics** *(ICCE)*, 1–2. https://doi.org/10.1109/ICCE.2019.8661933

MOHAN, A.; PAPAGEORGIOU, C.; POGGIO, T.. *Example-based object detection in images by components*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, [S.L.], v. 23, n. 4, p. 349-361, abr. 2001. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/34.917571.

MOTHWA, L.; TAPAMO, J.-R.; MAPATI, T. *Conceptual Model of the Smart Attendance Monitoring System Using Computer Vision.* **2018 14th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS),** 2018**.** doi:10.1109/sitis.2018.00042

NEGRÃO, F.S.; SANTOS, L.R.; SOARES, H.R. *Reconhecimento de Face Aplicada ao controle de chamada de classe*. **Revista Eletrônica E-RAC**, v. 8, n.1, 2018.

NING HE, Jiaheng Cao; LIN SONG. *Scale Space Histogram of Oriented Gradients for Human Detection.* **2008 International Symposium on Information Science and Engineering**. 2008. DOI:10.1109/isise.2008.308

NITAHARA, Akemi. **Estudo mostra que pandemia intensificou uso das tecnologias digitais: desigualdade de inclusão digital foram acentuadas.** Publicado em 25.11.2021. 15h26. Rio de Janeiro: Agência Brasil, 2021. Disponível em https://agenciabrasil.ebc.com.br/geral/noticia/2021-11/estudo-mostra-que-pandemia-intensificou-uso-das-tecnologias-digitais Acesso em 16.01.2023.

NOGUEIRA, G.R.G.; SANTOS, F.G. *Desenvolvimento de protótipo de fechadura eletrônica com reconhecimento facial*. **VII Escola Regional de Informática de Goiás**, Goiânia, 22 a 23 de novembro, 2019.

NUR AI. YouTube, 2020. **Haar Cascade Explained**. Disponível em: <https://www.youtube.com/watch?v=ozf1hM0W8lU>. Acesso em: 18 nov. 2021.

O'Neil, Cathy. **Algoritmos de destruição em massa: como o Big Data aumenta a desigualdade e ameaça a Democracia.** Tradução: Rafael Abraham. São Paulo: Rua do Sabão, 2020.

OJALA, T.; PIETIKAINEN, M.; HARWOOD, D. *A comparative study of texture measures with classification based on featured distributions*. **Pattern Recognition**, v.29, n.1, p. 51-59, January 1996. DOI: 10.1016/0031-3203(95)00067-4.

OJALA, T.; PIETIKAINEN, M.; MAENPAA, T. *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 24, n. 7, p. 971-987, July 2002. DOI: 10.1109/TPAMI.2002.1017623.

OLIVEIRA, Samuel R de. **Sorria, Você Está Sendo Filmado!: Repensando Direitos na Era do Reconhecimento Facial.** São Paulo: Revista dos Tribunais, 2021.

OPENCV. OpenCV, 2021. **Cascade Classifier Training**. Disponível em: <https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html>. Acesso em: 18 nov. 2021.

OPENCV. OpenCV, 2021. **Cascade Classifier**. Disponível em: <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>. Acesso em: 18 nov. 2021.

OTT, Patrick; EVERINGHAM, Mark. *Shared parts for deformable part-based models*. **CVPR 2011**, [S.L.], p. 1513-1520, jun. 2011. IEEE. http://dx.doi.org/10.1109/cvpr.2011.5995357.

PARK, U. et al. *Periocular Biometrics in the Visible Spectrum*. **IEEE Transactions on Information Forensics and Security**, v. 6, n. 1, p. 96-106, December 2010. DOI: 10.1109/TIFS.2010.2096810.

PARKHI, O.M.; VEDALDI, A.; ZISSERMAN, A. Deep Face Recognition. **Proceedings of the British Machine Vision. Conference 2015, BMVC 2015**, Swansea, UK, September 7-10, 2015. 41.1--41.12.

PEIXOTO, Fabiano Hartmann. DA SILVA, Roberta Zumblick Martins. **Inteligência Artificial e Direito.** Vol.1. Curitiba: Alteridade, 2019.

PERAMO, L.F.; MACEDO, P.C.; MORAES, M.R. *Desenvolvimento de um protótipo de software de reconhecimento facial: um estudo do sistema presente!!!*. **Revista Universitas**, ano 13, n. 24, jan/jun, 2019.

Perdana, A. B., & Prahara, A. (2019). Face Recognition Using Light-Convolutional Neural Networks Based On Modified Vgg16 Model. **2019 International Conference of Computer Science and Information Technology** *(ICoSNIKOM)*, 1–4. https://doi.org/10.1109/ICoSNIKOM48755.2019.9111481

PERE, Christophe. **What are Loss Functions?** 2020. Disponível em: https://towardsdatascience.com/what-is-loss-function-1e2605aeb904. Acesso em: 28 nov. 2022.

Pessoa, W. (2019). **Reconhecimento de padrões — Eigenfaces**. Brasil. Universidade Federal do Rio de Janeiro.

PHILLIPS, P.J.; MOON, H.; RIZVI, S.A.; RAUSS, P.J. *The FERET Evaluation Methodology for Face-Recognition Algorithms*. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 22:1090–1104, 2000. DOI: 10.1109/34.879790.

POYNTON, Charles. ***Digital Video and HDTV:*** *algorithms and interfaces*. San Francisco, EUA: Elsevier Science, 2003. 692 p.

PRADO, Magaly. **Fake News e inteligência artificial: o poder dos algoritmos na guerra da desinformação.** São Paulo: Edições 70, 2022.

QIANG Zhu; MEI-CHEN YEH, Kwang-Ting Cheng; AVIDAN, S. *Fast Human Detection Using a Cascade of Histograms of Oriented Gradients*. **2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition** - Volume 2 (CVPR'06). 2006. DOI:10.1109/cvpr.2006.119

RAMOS, André de Carvalho. **Curso de Direitos Humanos.** 8.Ed. São Paulo: Saraiva Educação, 2021.

RAPÔSO, C.F.L. et al. LGPD – Lei Geral de Proteção de Dados Pessoais em Tecnologia da Informação: Revisão Sistemática. **RACE Revista de Administração**. Issn 1806-0714, v. 4, Ano 2019. Disponível em: http://encurtador.com.br/pwLQ8 . Acessado em 28/10//2022.

RASCHKA, Sebastian. ***Python Machine Learning***. Birmingham, Inglaterra: Packt Publishing Ltd, 2016. 425 p.

REDMON, Joseph *et al*. *You Only Look Once: unified, real-time object detection*. **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, [S.L.], p. 779-788, jun. 2016. IEEE. http://dx.doi.org/10.1109/cvpr.2016.91.

REDMON, Joseph *et al*. **You Only Look Once**: real-time detection. [S. L.]: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. 48 slides, color.

REDMON, Joseph. ***I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore*.** [S. L.], 20 fev. 2020. Twitter: @pjreddie. Disponível em: https://robocademy.com/2020/05/01/a-gentle-introduction-to-yolo-v4-for-object-detection-in-ubuntu-20-04/#YOLO_v4. Acesso em: 15 mar. 2022.

REDMON, Joseph; ANGELOVA, Anelia. *Real-time grasp detection using convolutional neural networks*. **2015 IEEE International Conference on Robotics and Automation (ICRA)**, [S.L.], p. 1316-1322, maio 2015. IEEE. http://dx.doi.org/10.1109/icra.2015.7139361.

REDMON, Joseph; FARHADI, Ali. *YOLO9000: Better, Faster, Stronger*. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, [S.L.], p 6517-6525, jul. 2017. IEEE. http://dx.doi.org/10.1109/CVPR.2017.690.

REDMON, Joseph; FARHADI, Ali. *YOLOv3: An Incremental Improvement*. **arXiv**, [S.L.], p. 1-6, 8 abr. 2018.

REIS, João. YouTube, 2018. **OpenCV Python - Detecção de face e olhos usando Haar Cascade**. Disponível em: <https://www.youtube.com/watch?v=5n6MZNhzcII>. Acesso em: 18 nov. 2021.

REZATOFIGHI, Hamid *et al*. *Generalized Intersection Over Union: a metric and a loss for bounding box regression*. **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**, [S.L.], p. 658-666, jun. 2019. IEEE. http://dx.doi.org/10.1109/cvpr.2019.00075.

RODRIGUES, Lucas de Oliveira. **"Hooligans"**. Brasil Escola. Disponível em: https://brasilescola.uol.com.br/sociologia/hooligans.htm Acesso em 24 de janeiro de 2023.

ROHRER, Brandon. YouTube, 2016. **How Convolutional Neural Networks work**. Disponível em: <https://www.youtube.com/watch?v=FmpDIaiMIeA>. Acesso em 22 mar. 2022.

ROLIM, A. L.; BEZERRA, E. P. *Um Sistema De Identificação Automática De Faces Para Um Ambiente Virtual De Ensino E Aprendizagem*. In: **XIV Simpósio Brasileiro de Sistemas Multimídia e Web**, Vila Velha. 2008. DOI: 10.1145/1809980.1810015.

ROSEBROCK, Adrian. PyImageSearch, 2021. **OpenCV Haar Cascades**. Disponível em: <https://www.pyimagesearch.com/2021/04/12/opencv-haar-cascades/>. Acesso em: 18 nov. 2021.

ROSEBROCK, Adrian. ***Intersection over Union (IoU) for object detection***. 2016. Disponível em: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. Acesso em: 01 mar. 2022.

ROSSEAU, Jean-Jacques. **O contrato social: princípios do Direito Político.** Tradução: Edson Bini. São Paulo: Edipro, 2017.

Ruan, X., Tian, C., & Xiang, W. (2020). Research on Face Recognition Based on Improved Dropout Algorithm. **2020 IEEE 5th Information Technology and Mechatronics Engineering Conference** *(ITOEC)*, 700–703. https://doi.org/10.1109/ITOEC49072.2020.9141891

SADEGHI, Mohammad Amin; FORSYTH, David. *30Hz Object Detection with DPM V5*. **Computer Vision – ECCV 2014**, [S.L.], p. 65-79, 2014. Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-10590-1_5.

SALSCHEIDER, Niels Ole. *FeatureNMS: non-maximum suppression by learning feature embeddings*. **2020 25th International Conference on Pattern Recognition (ICPR)**, [S.L.], p. 7848-7854, 10 jan. 2021. IEEE. http://dx.doi.org/10.1109/icpr48806.2021.9412930.

SANLI, O.; ILGEN, B. *Face Detection and Recognition for Automatic Attendance System.* **Intelligent Systems and Applications***, 237–245.* 2018. doi:10.1007/978-3-030-01054-6_17

Schapire, R. E., Freund, Y., Bartlett, P., & Lee, W. S. (1998). Boosting the margin: a new explanation for the effectiveness of voting methods. **The Annals of Statistics**, 26(5), 1651–1686. doi:10.1214/aos/1024691352

SCHNEIDER, E.; HOPPE, A.F. *Yourface: um protótipo de reconhecimento facial em ambientes indoor sem obstáculo*. **Trabalho de Conclusão de Curso**. Universidade Regional de Blumenau, Curso de Ciência da Computação, 2018.

Schroff, F.; Kalenichenko, D.; Philbin, J. "*FaceNet: A Unified Embedding for Face Recognition and Clustering*". **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition** (CVPR), pp. 815-823, 2015. DOI: 10.1109/CVPR.2015.7298682

SCHWAB, Klaus. **A quarta revolução industrial.** Tradução: Daniel Moreira Miranda. São Paulo: Edipro, 2016.

SCHWARTZ, W.R.; GUO, H.M.; DAVIS, L.S. *A Robust and Scalable Approach to Face Identification*. In **European Conference on Computer Vision**, volume 5, pages 476–489, 2010. DOI: 10.1007/978-3-642-15567-3_35.

SCIKITLEARN. **sklearn.metrics.confusion_matrix**. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. Acesso em: 26 jun. 2022.

Serengil, S. I. (2020). LightFace: A Hybrid Deep Face Recognition Framework, Istanbul, Turkey.

SERMANET, Pierre *et al*. *OverFeat: integrated recognition, localization and detection using convolutional networks*. **International Conference on Learning Representations**, [S. L.], p. 1-16, 21 dec. 2013.

SHANMUGAMANI, R. Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras. Birmingham, 2018.

SHUAISHI, L.; YAUN, Z.; KEPING, L. *Facial expression recognition under partial occlusion based on Weber Local Descriptor histogram and decision fusion*. In: **Chinese Control Conference (CCC),** p. 4664-4668, 2014. DOI: 10.1109/ChiCC.2014.6895725.

SILVA, A. L. **Redução de características para classificação de imagens de faces**, 2016. Disponível em: <https://ppgcc.ufersa.edu.br/wp-content/uploads/sites/42/2014/09/REDU%C3%87%C3%83O-DE-CARACTER%C3%8DSTICAS-PARA-CLASSIFICA%C3%87%C3%83O-DE-IMAGENS-DE-FACES.pdf>. Acesso em: 06 nov. 2019.

SILVA, A. L.; CINTRA, M. E. *Reconhecimento de padrões faciais: Um estudo*. **In: Encontro Nacional de Inteligência Artificial e Computacional, 2015, Proceedings ENIAC**, 224-231, 2015.

SIMÃO, Bárbara; FRAGOSO, Nathalie; ROBERTO, Enrico. **Reconhecimento Facial e o Setor Privado: Guia para a adoção de boas práticas.** InternetLab/IDEC, São Paulo, 2020. Disponível em https://idec.org.br/sites/default/files/reconhecimento_facial_diagramacao_digital_2.pdf Acesso em 10.01.2023.

SIMONYAN, K. et al. *Fisher Vector Faces in the Wild*. In: **British Machine Vision Conference (BMVC)**, 2013. DOI: 10.5244/C.27.8

SOARES, A. D. S*. Algoritmo da retropropagação de erros (Backpropagation) para redes multi-layer perceptron*. **Videoaula - Deep Learning Brasil**, 22 jul. 2018. Disponivel em: <https://www.youtube.com/watch?v=DGNbd2FGw2s&t=4007s>. Acesso em: 29 set. 2019.

SOARES, A. D. S. *Deep Learning Brasil (Em português). Redes Neurais profundas Convolucionais - Parte I - Fundamentos***, Videoaula - Deep Learning Brasil**, 27 maio 2018. Disponivel em: <https://www.youtube.com/watch?v=n4rmrZg1_58&t=6246s>. Acesso em: 05 nov. 2019.

STAN, Z. Li; ANIL, K. J. **Handbook of Face Recognition**. 2nd Edition, Springer, 2011.

STATQUEST WITH JOSH STARMER. YouTube, 2021. Neural Networks Part 8: Image Classification with Convolutional Neural Networks. Disponível em: <https://www.youtube.com/watch?v=HGwBXDKFk9I>. Acesso em 22 mar. 2022.

STONE, Maureen C.. ***A Field Guide to Digital Color***. [S. L.]: A K Peters/Crc Press, 2003. 326 p.

STRECK, Lenio Luiz. **Jurisdição Constitucional e hermenêutica: uma nova crítica do Direito.** 2.Ed. Rio de Janeiro: Forense, 2004.

TAIGMAN, Y. et al. *DeepFace: Closing the Gap to Human-Level Performance in Face Verification*. In: **IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 1701-1708, 2014. DOI: 10.1109/CVPR.2014.220.

TAN, X.; TRIGGS, B. *Fusing Gabor and LBP Feature Sets for Kernel-Based Face Recognition*. In **IEEE International Conference on Automatic Face and Gesture Recognition**, pages 235–249, 2007.

TEFFÉ, C. S.; VIOLA, M. Tratamento de dados pessoais na LGPD: estudo sobre as bases legais. **Civilistica.com.** Rio de Janeiro, a. 9, n. 1, 2020. Disponível em: http://civilistica.com/tratamento-de-dados-pessoais-na-lgpd/ . Acessado em: 28/10/2022.

TENSORFLOW. YouTube, 2019. Introducing convolutional neural networks (ML Zero to Hero - Part 3). Disponível em: <https://www.youtube.com/watch?v=x_VrgWTKkiM>. Acesso em 22 mar. 2022.

THUAN, Do. *Evolution of YOLO Algorithm and YOLOv5: The State-of-the-art Object Detection Algorithm*. **Bachelor's Thesis**, Oulu University of Applied Sciences, 2021.

TIAN, Zhi et al. *FCOS: fully convolutional one-stage object detection*. **a**rXiv, [S.L.], p. 1-13, 2019. http://dx.doi.org/10.48550/ARXIV.1904.01355.

TOLBA, A.S.; EL-BAZ, A.H.; EL-HARBY, A.A. Face Recognition: A Literature Review. **International Journal of Signal Processing**, 2:88–103, 2006.

TURK, M. A.; PENTLAND, A P.. *Eigenfaces for Recognition*. **Journal of Cognitive Neuroscience**, v. 3, n. 1, p. 71-86, January 1991. DOI: 10.1162/jocn.1991.3.1.71.

ULLAH, I. et al. *Gender recognition from face images with local WLD descriptor*. In: **International Conference on Systems, Signals and Image Processing (IWSSIP)**, 2012. p. 417-420.

UNIVERSO DISCRETO. YouTube, 2021. **REDES NEURAIS CONVOLUCIONAIS: TUTORIAL COMPLETO | Redes Neurais e Deep Learning 09**. Disponível em: <https://www.youtube.com/watch?v=0lvHURoyhtc>. Acesso em 22 mar. 2022.

VAGHELA, Dushyant; NAINA, Prof. Kapildev. *A Review of Image Mosaicing Techniques*. **arXiv**, [S.L.], p. 1-6, 2014. ArXiv. http://dx.doi.org/10.48550/ARXIV.1405.2539.

VALVERDE, Eduardo. **O mercado da tecnologia em 2023.** Publicado às 06:00 do 16 de Janeiro de 2023. Fortaleza: Diário do Nordeste, 2023. Disponível em https://diariodonordeste.verdesmares.com.br/opiniao/colaboradores/o-mercado-de-tecnologia-em-2023-1.3323962 Acesso em 16.01.2023.

VEERAMACHANENI , K. et al., *Improving Classifier Fusion Using Particle Swarm Optimization,* **Proceedings of the IEEE Symposium on Computational Intelligence in Multicriteria Decision Making**, pp. 128-135, 2007. DOI: 10.1109/MCDM.2007.369427.

VICENTINI, Tissiane. **ViaQuatro é condenada por coleta irregular de dados; multa é de R$ 100 mil.** Olhar Digital. Publicação: 11/05/2021 12h24, atualizada em 03/06/2021 01h01. Disponível em https://olhardigital.com.br/2021/05/11/pro/viaquatro-e-condenada-por-coleta-irregular-de-dados-multa-e-de-r-100-mil/ Acesso em 24.01.2022.

VIOLA, P.; JONES, M.. *Rapid object detection using a boosted cascade of simple features*. **Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition**. CVPR 2001, [S.L.], p. 511-518, dez. 2001. IEEE Comput. Soc. http://dx.doi.org/10.1109/cvpr.2001.990517.

VISHALBPATIL1. **Yoloface**. 2021. Disponível em: https://github.com/vishalbpatil1/yoloface/blob/main/test/a56.jpg. Acesso em: 04 jul. 2022.

VISO.AI (org.). **YOLOv7: the most powerful object detection algorithm (2022 guide)**. 2022. Disponível em: https://viso.ai/deep-learning/yolov7-guide/. Acesso em: 29 nov. 2022.

WAGHELA, T.; DHANUKA, H.; BARNWAL, A.; LABDE, S. *Attendance Management system (using face recognition).* **International Research Journal of Engineering and Technology (IRJET)**. Vol. 5, issue 4, p. 3177-3180, Apr., 2018.

Wan, W., & Chen, J. (2017). Occlusion robust face recognition based on mask learning. **2017 IEEE International Conference on Image Processing** *(ICIP)*, 3795–3799. https://doi.org/10.1109/ICIP.2017.8296992

WANG, Chien-Yao *et al*. *CSPNet: a new backbone that can enhance learning capability of CNN*. **2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)**, [S.L.], p. 1571-1580, jun. 2020. IEEE. http://dx.doi.org/10.1109/cvprw50498.2020.00203.

WANG, Chien-Yao et al. *YOLOv7: trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. **ArXiv**, [S.L.], p. 1-15, jul. 2022. ArXiv. http://dx.doi.org/10.48550/ARXIV.2207.02696.

WANG, J. et al. *Boosting dense SIFT descriptors and shape contexts of face images for gender recognition*. In: **IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)**, p. 96-102, 2010. DOI: 10.1109/CVPRW.2010.5543238

Wang, M. and Deng, W. (2020). Reconhecimento facial profundo: uma pesquisa. **Neurocomputação**. doi:10.1016/j.neucom.2020.10.08

Wang, M., Wang, Z., Zhang, S., Luan, J., & Jiao, Z. (2018). Face Expression Recognition Based on Deep Convolution Network. **2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics** *(CISP-BMEI)*, 1–9. https://doi.org/10.1109/CISP-BMEI.2018.8633014

WANG, X. et al. *Feature fusion of HOG and WLD for facial expression recognition*. In: **IEEE/SICE International Symposium on System Integration (SII)**, p. 227-232, 2013. DOI: 10.1109/SII.2013.6776664.

WASSERMAN, Larry. ***All of statistics:*** *a concise course in statistical inference*. New York: Springer, 2004. 736 p.

WEBB, Amy. **Os nove titãs da IA: como as gigantes da tecnologia e suas máquinas pensantes podem subverter a humanidade.** Rio de Janeiro: Alta Books, 2020.

Wikipedia. (2016). **Eigenface**, disponível em https://en.wikipedia.org/wiki/Eigenface, acessado em 13/09/2021.

WOODARD, D. et al. *Periocular Region Appearance Cues for Biometric Identification*. In: **IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)**, p. 162-169, 2010. DOI: 10.1109/CVPRW.2010.5544621.

WU, B.; NEVATIA, R. *Optimizing Discrimination-Efficiency Tradeoff in Integrating Heterogeneous Local Features for Object Detection*. In **IEEE Conference on Computer Vision and Pattern Recognition**, 2008.

Wu, G., Tao, J., & Xu, X. (2019). *Occluded Face Recognition Based on the Deep Learning.* **2019 Chinese Control And Decision Conference** *(CCDC).* doi:10.1109/ccdc.2019.8832330

Xie, Z., Li, Y., Niu, J., Yu, X., & Shi, L. (2019). Hyperspectral Face Recognition Using Block based Convolution Neural Network and AdaBoost Band Selection. **2019 6th International Conference on Systems and Informatics** *(ICSAI)*, 1270–1274. https://doi.org/10.1109/ICSAI48974.2019.9010511

XIE, Z.; JIANG, P.; ZHANG, S. *Fusion of LBP and HOG using multiple kernel learning for infrared face recognition.* **2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS).** 2017. doi:10.1109/icis.2017.7959973

XU, J.; SAVVIDES, M. *Unconstrained periocular biometric acquisition and recognition using COTS PTZ camera for uncooperative and non-cooperative subjects*. In: **IEEE Workshop on Applications of Computer Vision (WACV)**, p. 201-208, 2012. DOI: 10.1109/WACV.2012.6163051.

Xu, X.-F., Zhang, L., Duan, C.-D., & Lu, Y. (2020). Research on Inception Module Incorporated Siamese Convolutional Neural Networks to Realize Face Recognition. ***IEEE Access***, *8*, 12168–12178. https://doi.org/10.1109/ACCESS.2019.2963211

YAN, Junjie *et al*. *The Fastest Deformable Part Model for Object Detection*. **2014 IEEE Conference On Computer Vision And Pattern Recognition**, [S.L.], p. 2497-2504, jun. 2014. IEEE. http://dx.doi.org/10.1109/cvpr.2014.320.

Yang, M.H; Kriegman, D.; Ahuja, N. (2002). "*Detecting Faces in Images: A Survey*". **IEEE Transactions on Pattern Analysis and Machine Intelligence** 24(1):34 – 58. February 2002. DOI:10.1109/34.982883

Yang, M.-H; Roth, D.; Ahuja, N. "*A SNoW-Based Face Detector*," **Advances in Neural Information Processing Systems 12**, S.A. Solla, T., T. K. Leen, and K.-R. Muller, eds., pp. 855-861, MIT Press, 2000.

Yao, L. (2020). A Compressed Deep Convolutional Neural Networks for Face Recognition. **2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics** *(ICCCBDA)*, 144–149. https://doi.org/10.1109/ICCCBDA49378.2020.9095672

YNOGUTI, P. D. C. A. *Processamento Digital de Sinais: Convolução*. **Notas de aula da Inatel**, 2019. Disponivel em: <https://www.inatel.br/docentes/ynoguti/downloads-arquivos/dsp-s886637-1/18-convolucao-s104246-1>. Acesso em: 04/12/2019.

Zadeh, T.; Imani, M.M.; Majidi, B. (2019). *Fast Facial emotion recognition Using Convolutional Neural Networks and Gabor Filters.* **2019 5th Conference on Knowledge Based Engineering and Innovation** *(KBEI)*. doi:10.1109/kbei.2019.8734943

Zeng, J., Zhao, X., Qin, C., & Lin, Z. (2017). Single sample per person face recognition based on deep convolutional neural network. **2017 3rd IEEE International Conference on Computer and Communications** *(ICCC)*, 1647–1651. https://doi.org/10.1109/CompComm.2017.8322819

ZHANG, B.; RUAN, Q. *Facial feature extraction using improved deformable templates*. **Signal Processing, 8th International Conference on**, vol.4, no., 16-20, 2006. DOI: 10.1109/ICOSP.2006.345929.

ZHANG, L. et al. *Face Recognition Using Scale Invariant Feature Transform and Support Vector Machine*. In: **International Conference for Young Computer Scientists (ICYCS),** 9th, p. 1766-1770, 2008. DOI: 10.1109/ICYCS.2008.481.

ZHANG, W.; SHAN, S.; GAO, W.; CHEN, X.; ZHANG, H. *Local Gabor Binary Pattern Histogram Sequence (LGBPHS): A Novel Non-Statistical Model for Face Representation and Recognition*. In **International Conference on Computer Vision**, pages 786–791, 2005. DOI: 10.1109/ICCV.2005.147.

ZHAO, W.; CHELLAPPA, R.; PHILLIPS, P.J.; ROSENFELD, A. *Face Recognition: A Literature Survey*. **ACM Computing Surveys**, 35(4):399–458, 2003. DOI: 10.1145/954339.954342.

ZHONG, Y. *et al*. *Anchor Box Optimization for Object Detection*. **2020 IEEE Winter Conference on Applications of Computer Vision (WACV)**, [S.L.], p. 1275-1283, mar. 2020. IEEE. http://dx.doi.org/10.1109/WACV45572.2020.9093498.

Zhou, Y., Liu, Y., Han, G., & Zhang, Z. (2019). Face Recognition Based on Global and Local Feature Fusion. *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2771–2775. https://doi.org/10.1109/SSCI44817.2019.9003045

Ziani, C., & Sadiq, A. (2019). *Shearlet Convolutional Neural Network Approach for Age and Gender recognition. 2019 Third International Conference on Intelligent Computing in Data Sciences (ICDS)*. doi:10.1109/icds47004.2019.8942359

ZOU, J.; JI, Q.; NAGY, G. *A Comparative Study of Local Matching Approach for Face Recognition*. **IEEE Transactions on Image Processing**, 16:2617–2628, 2007. DOI: 10.1109/TIP.2007.904421.

Anexo A

Criação de um detector Haar Cascade: Manual e Automaticamente

O Anexo A é referente ao Capítulo 07 - Haar Cascade e tem intuito de apresentar descrições detalhadas sobre certos processos presentes no mesmo, como a instalação de software para virtualização de máquinas, instalação de um Sistema Operacional, instalação de programas etc.

A.1 **Criação Manual de um Detector**

A.1.1 Instalação Do Virtualbox

O Sistema Operacional utilizado para a tarefa de treinamento do detector será o Linux Ubuntu, já que é um sistema open-source e gratuito. Sua instalação será realizada numa máquina virtual utilizando o software VirtualBox (versão 6.1.30). Em compensação, é possível realizar o treinamento utilizando outras alternativas, como servidores, por exemplo.

O VirtualBox é um software gratuito de virtualização que permite a utilização de diferentes sistemas operacionais numa mesma máquina. Nele é possível configurar a quantidade de armazenamento, memória RAM, memória de vídeo etc. que será dedicada a cada máquina individualmente.

Para a instalação do VirtualBox, deve-se baixar seu arquivo de instalação. Este pode ser obtido através do seguinte link: https://www.virtualbox.org/wiki/Downloads (nota-se que para a instalação a seguir foi utilizado o Sistema Operacional Windows 10). Quando o download estiver concluído, deve ser executado o arquivo e instalado conforme a Figura A.1 (Parte A - G).

Figura A.1 (A) - Instalação do VirtualBox.

Para instalar o software, deve-se selecionar as opções cercadas por um retângulo vermelho.

Figura A.1 (B)

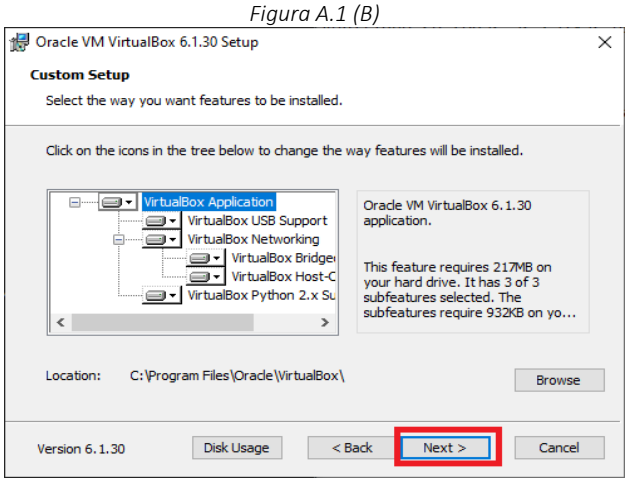
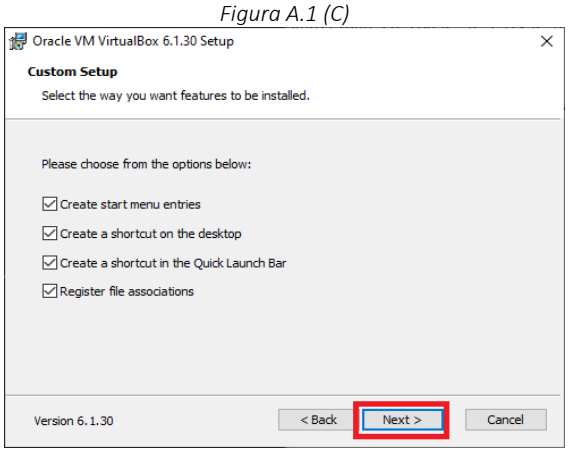


Figura A.1 (C)



É recomendável que todas as caixas de seleção estejam marcadas.

Figura A.1 (D)

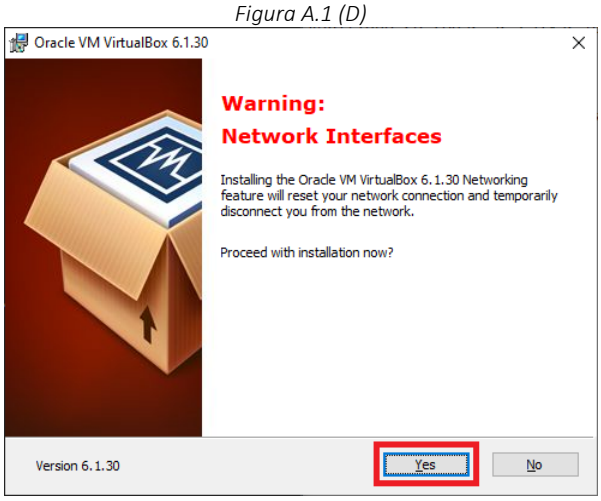


Figura A.1 (E)

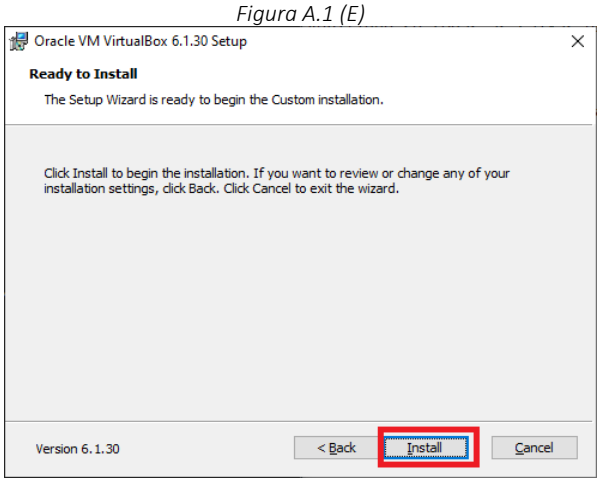


Figura A.1 (F)

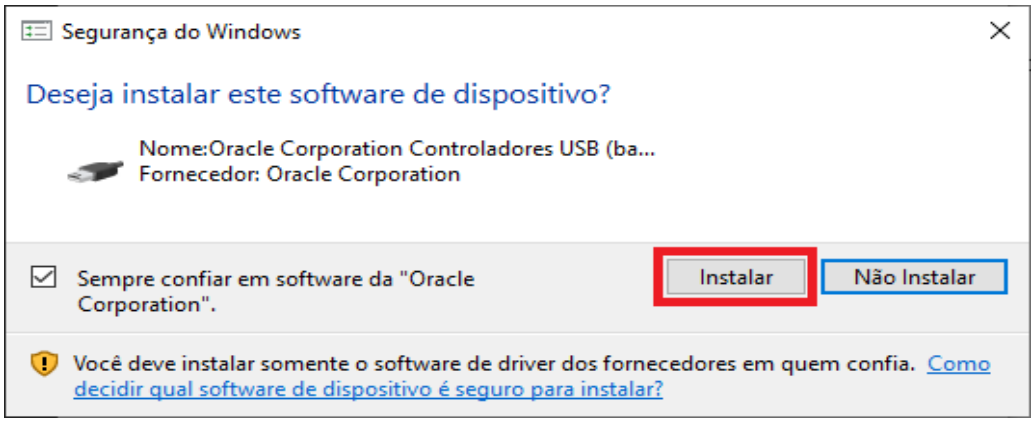
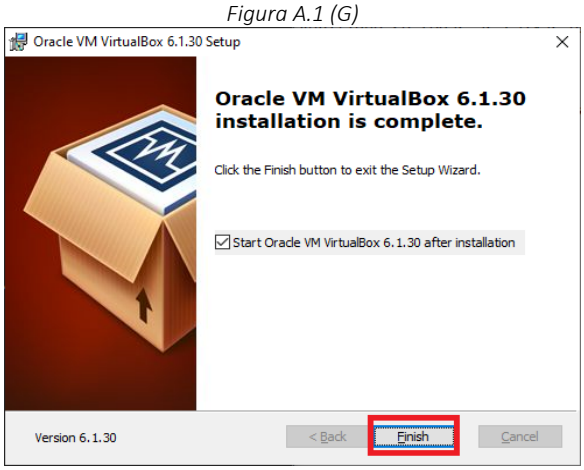


Figura A.1 (G)



Ao selecionar "Finish", será finalizada a instalação do VirtualBox.

A.1.2 Criação da Máquina Virtual

Com o VirtualBox instalado, já é possível instalar uma máquina virtual. No entanto, o VirtualBox consegue criar máquinas virtuais apenas a partir de uma mídia de instalação. Ou seja, é necessário baixar o arquivo ISO (por exemplo) do Linux Ubuntu e, em seguida, instalar a partir do VirtualBox. O arquivo pode ser baixado através do seguinte link: https://ubuntu.com/download/desktop/thank-you?version=20.04.3&architecture=amd64.

Quando o download estiver concluído, é necessário executar o VirtualBox e seguir o tutorial de instalação selecionando os respectivos itens marcados na Figura A.2 (Parte A - H).

Figura A.2 (A) - Instalação da máquina virtual.

Figura A.2 (B)

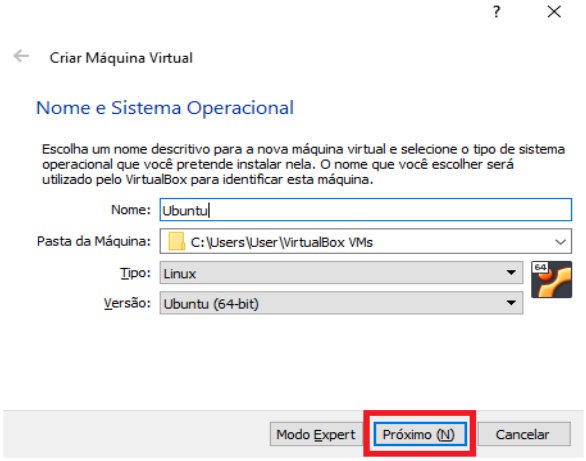


Figura A.2 (C)

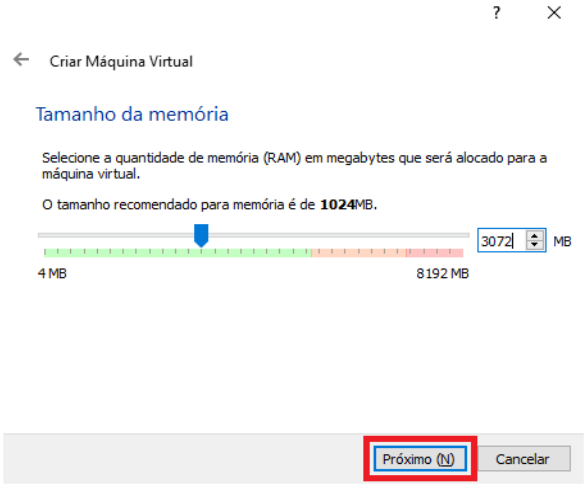


Figura A.2 (D)

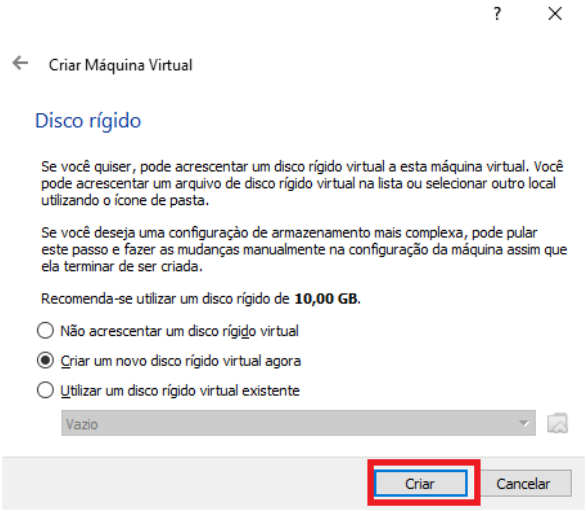


Figura A.2 (E)

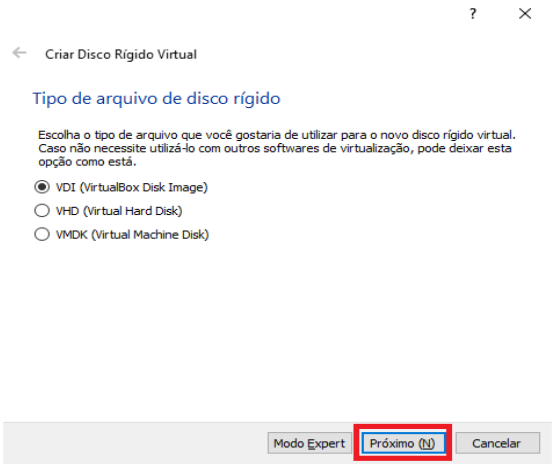


Figura A.2 (F)

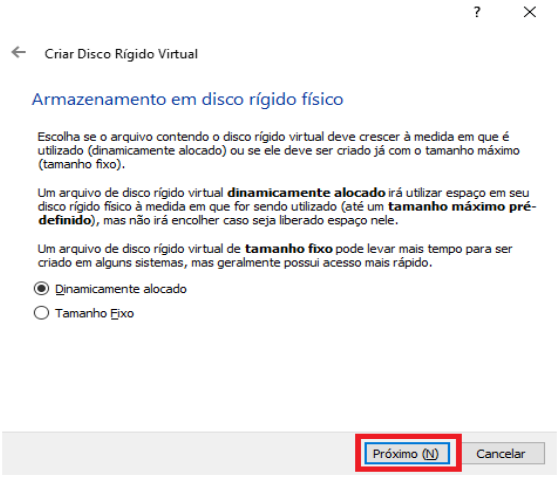
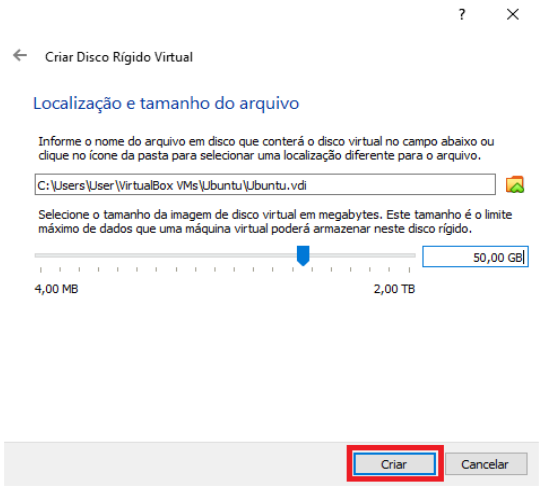
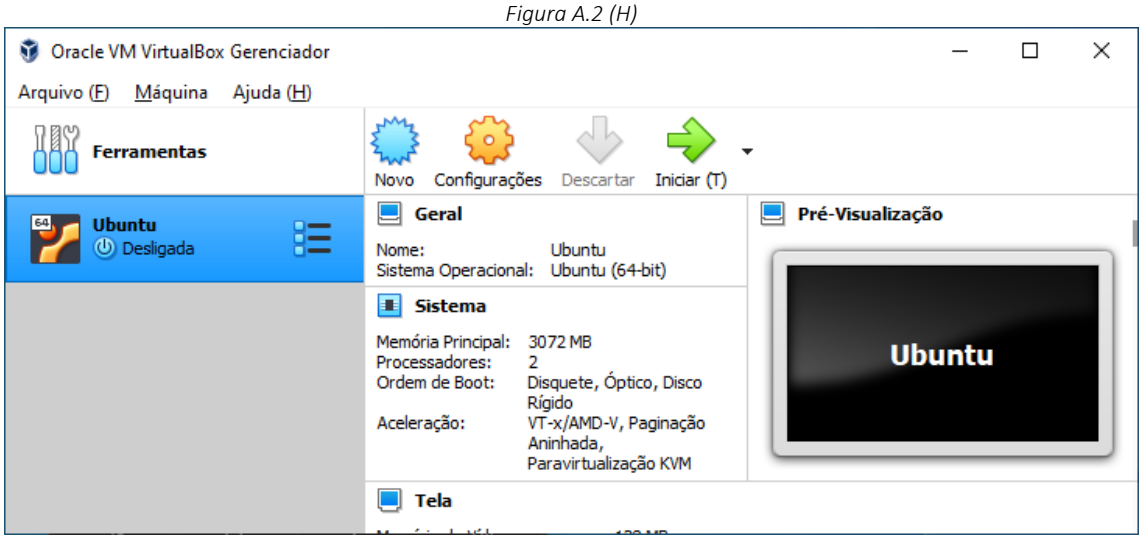


Figura A.2 (G)



Embora o sistema não ocupe 50 gigas de armazenamento, é recomendável dedicar certa quantidade de espaço adicional para garantir um bom funcionamento geral.

Figura A.2 (H)



É possível alterar as especificações dedicadas à máquina alterando as configurações em "Configurações" para necessidades de cada usuário. Neste caso, nada será alterado.

A partir do momento que a máquina estiver pronta para uso, é necessário instalar o sistema operacional.

A.1.3 Instalação do Sistema Operacional e configurações iniciais

Figura A.3 (A) - Instalação do sistema operacional.

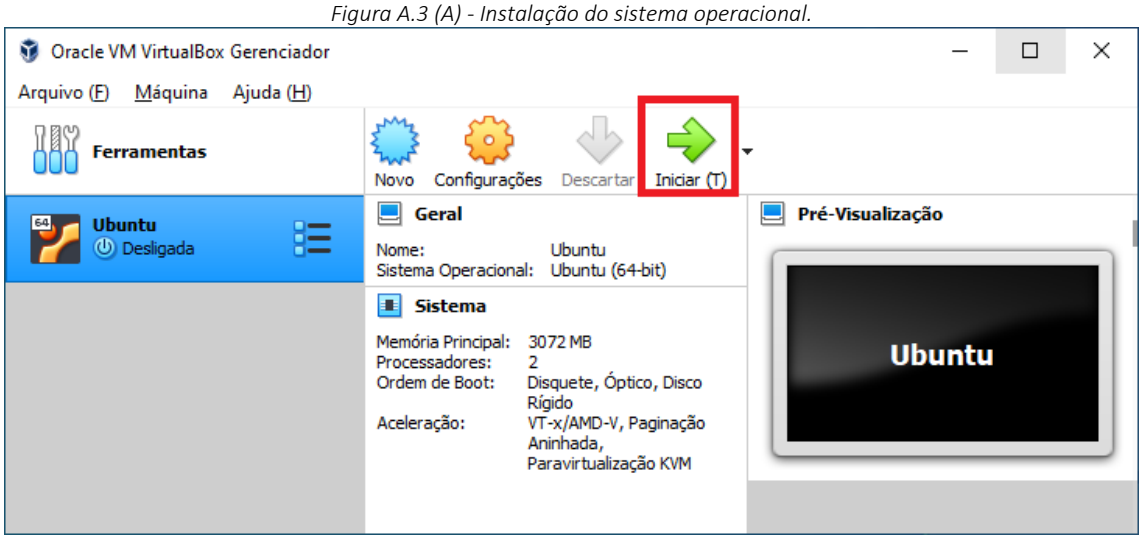
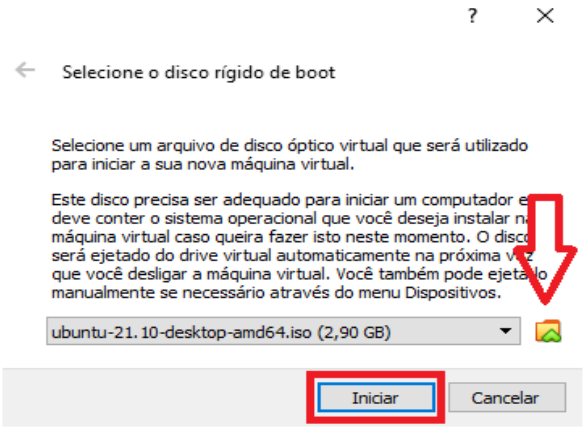
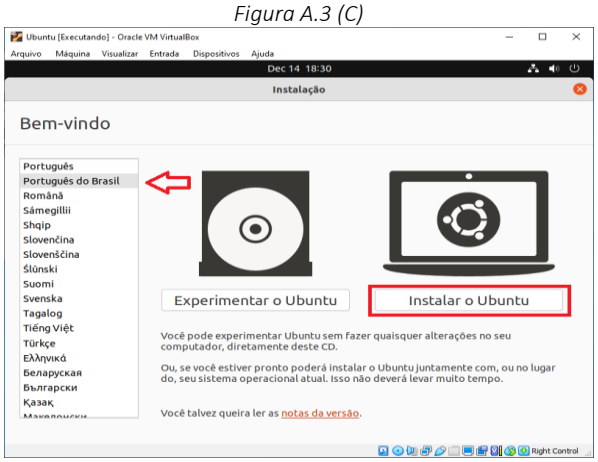


Figura A.3 (B)



Deve-se selecionar a imagem ISO baixada anteriormente.

Figura A.3 (C)



Deve-se selecionar a língua desejada.

Figura A.3 (D)

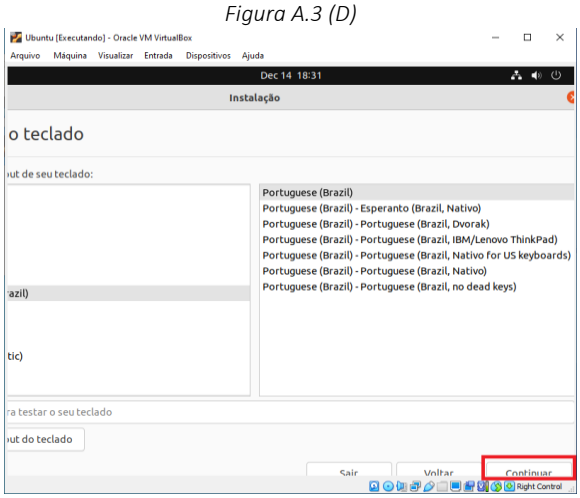
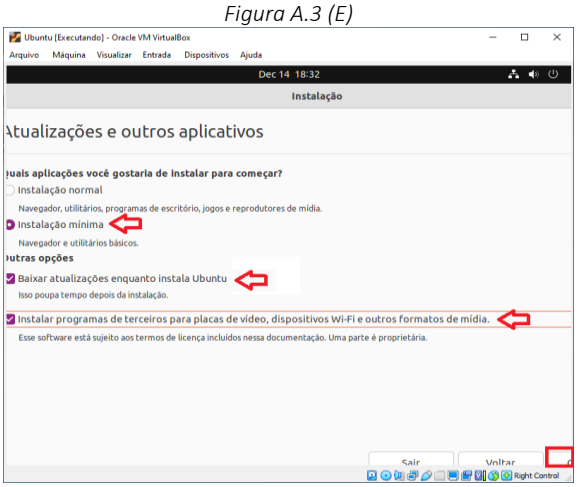


Figura A.3 (E)



Deve-se selecionar as opções marcadas na imagem. Nota-se que a resolução do sistema operacional ainda não está compatível com a original do monitor e, consequentemente, certas partes da máquina virtual não estão completamente visíveis. Este problema será resolvido posteriormente, mas uma solução temporária é "arrastar" a janela de "Instalação" para os lados, aumentando o campo de visão.

Figura A.3 (F)

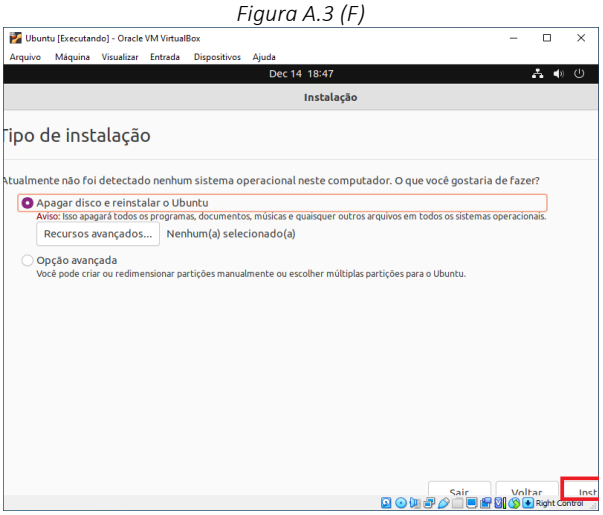


Figura A.3 (G)

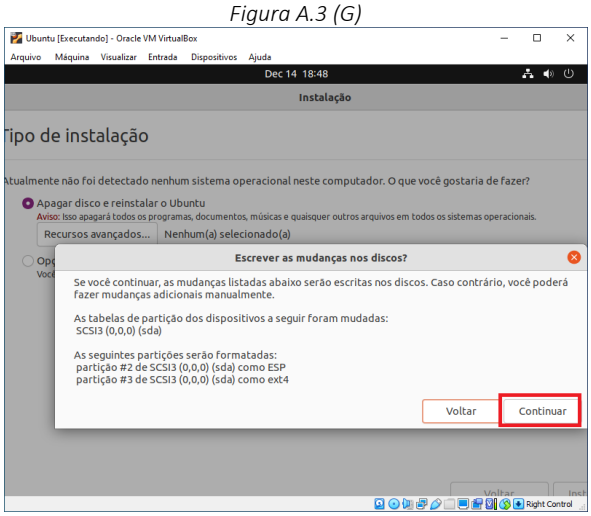


Figura A.3 (H)

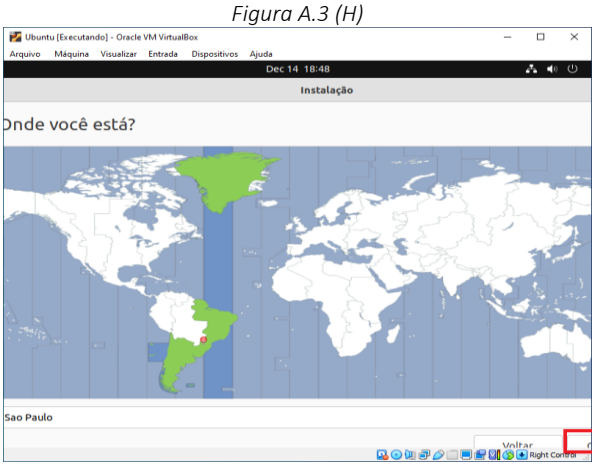
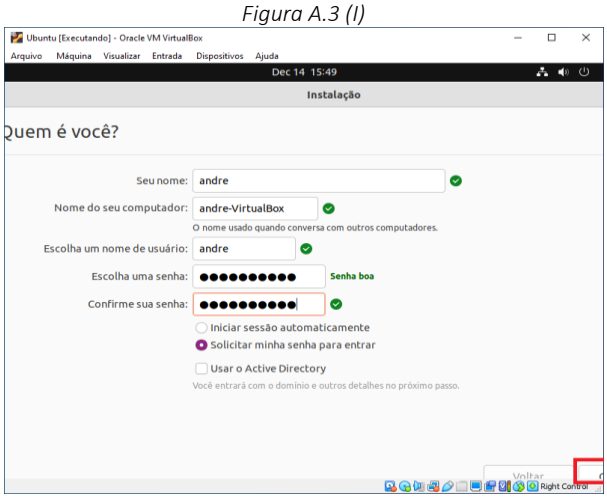
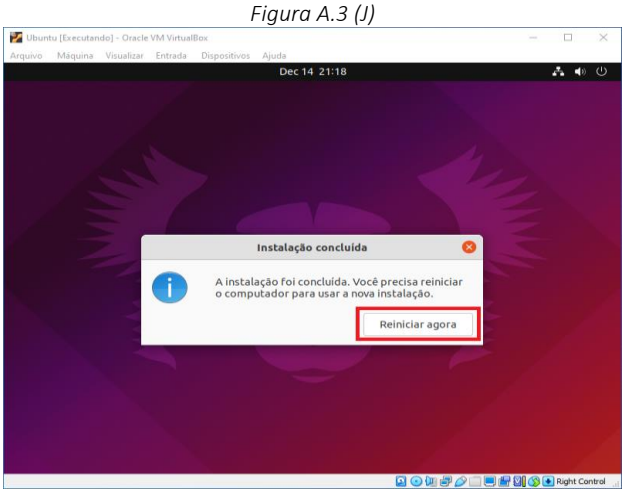


Figura A.3 (I)



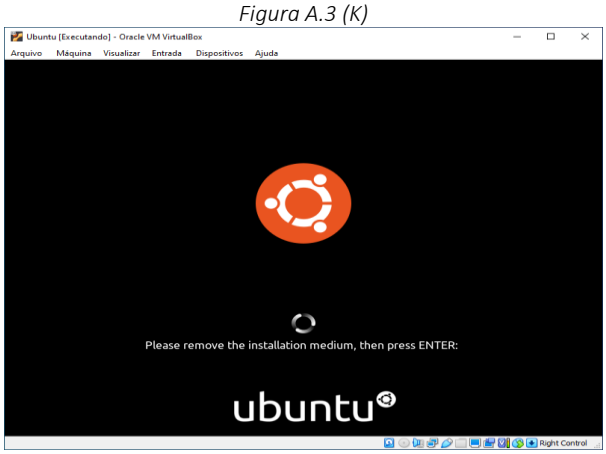
É necessário certificar-se de que: o nome informado não possui caracteres especiais ou espaços; a senha criada é forte; o sistema irá solicitar a senha ao ligar a máquina. Embora alguns destes passos não sejam obrigatórios, são recomendados para um uso mais seguro do sistema.

Figura A.3 (J)



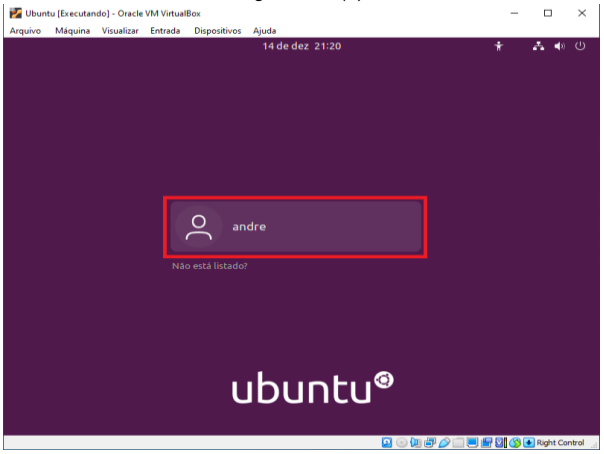
Deve-se reiniciar o computador.

Figura A.3 (K)



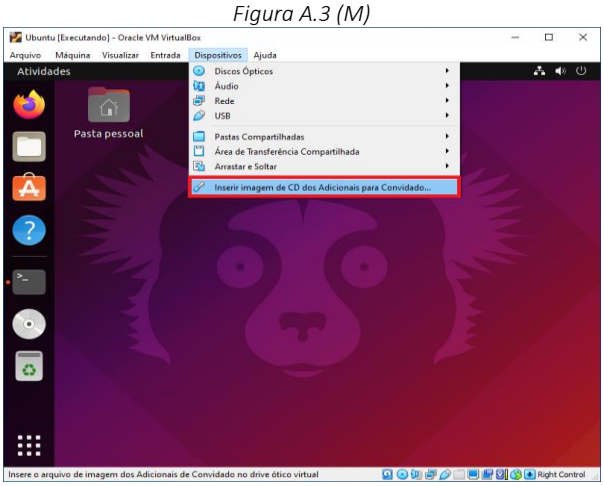
Deve-se pressionar a tecla ENTER para prosseguir com a inicialização.

Figura A.3 (L)



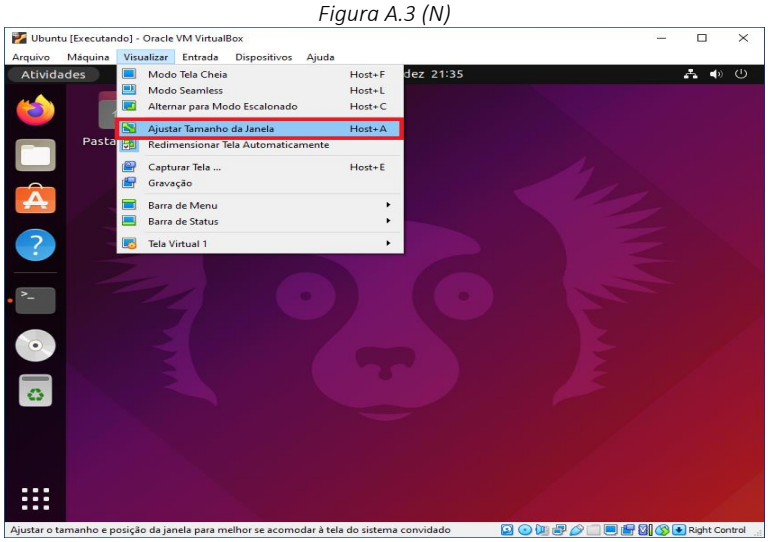
É necessário informar as credenciais de usuário.

Figura A.3 (M)



Na janela do VirtualBox, é selecionado "Dispositivos" e, em sequência, "Inserir imagem de CD dos Adicionais para Convidado...". Dependendo da versão do sistema instalado, pode ser necessário informar a senha do usuário. Quando concluído, é necessário reiniciar o computador.

Figura A.3 (N)



Na janela do VirtualBox, é selecionado "Visualizar" e, em sequência, "Ajustar Tamanho da Janela". Quando concluído, será possível redimensionar a janela para o tamanho desejado.

Para finalizar a configuração inicial, é necessário executar dois comandos no terminal (que pode ser aberto através do atalho CTRL + ALT + T) para garantir o melhor funcionamento possível. Para isso, deve ser executado o Script A.1.

Script A.1- Atualização da máquina.

```
sudo apt update

sudo apt upgrade
```

Explicação do Código:

```
sudo apt update
```

Ao se utilizar o comando "sudo", é permitida a utilização de comandos em modo de administrador do sistema. É necessário saber as credenciais de login do administrador para utilizar este comando.

O comando "apt" (Advanced Package Tool), por sua vez, é uma coleção de ferramentas que podem ser utilizadas para administrar pacotes de Softwares em distribuições baseadas em Linux Debian.

Para finalizar, o comando "update" verifica, informa e atualiza os pacotes de softwares que podem ser atualizados no sistema operacional.

É recomendado utilizar estes comandos logo após instalar um sistema operacional baseado em Debian, já que garantem que todos os pacotes utilizados serão atualizados para a última versão disponível. Além disso, é comum serem executados em sequência (como explicado), ou "concatenados" - com a utilização do operador "&&", que adiciona um segundo comando à fila (o segundo comando só será executado quando o primeiro for finalizado).

```
sudo apt upgrade
```

O comando “upgrade” atualiza os pacotes que possuem versões novas disponíveis.

Neste ponto, o sistema está devidamente atualizado e rodando em sua última versão. Para instalar o Python, será executado o Script A.2.

Script A.2 - Instalação do Python.

```
sudo apt install python3
```

Embora seja comum o Linux Ubuntu vir com o Python instalado, esta prática pode variar dependendo da versão instalada.

O comando “install” instala novos pacotes de software na máquina. “python3” é o pacote que possui a última versão do Python.

Para instalar a OpenCV, deve-se, primeiramente, instalar todas as dependências necessárias, como consta o Script A.3.

Script A.3 - Instalação das dependências da OpenCV.

```
sudo apt install build-essential cmake git pkg-config libgtk-3-dev
libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev
libx264-dev libjpeg-dev libpng-dev libtiff-dev gfortran openexr libatlas-
base-dev python3-dev python3-numpy libtbb2 libtbb-dev libdc1394-22-dev
```

Fonte: disponível em https://linuxize.com/post/how-to-install-opencv-on-ubuntu-18-04/.

A instalação da OpenCV será feita manualmente, ou seja, pelo código-fonte. Para prosseguir para a instalação, será criado um diretório específico para a OpenCV, como consta o Script A.4.

Script A.4 - Criação do diretório da OpenCV.

```
mkdir ~/opencv_build

cd ~/opencv_build
```

Dois repositórios armazenados no GitHub serão baixados quando executado o Script A.5. O primeiro se refere à versão padrão da OpenCV. O segundo, ao repositório que contém recursos adicionados por contribuidores da biblioteca. Ambas versões serão utilizadas neste processo.

Script A.5 - Download do código-fonte da OpenCV.

```
git clone https://github.com/opencv/opencv.git
```

```
git clone https://github.com/opencv/opencv_contrib.git
```

Explicação do Código:

```
git clone
```

O comando “git clone” baixará o arquivo GIT armazenado, neste caso, no GitHub.

A versão da OpenCV necessária para continuar o processo de treinamento é a 3.4.14. Para selecioná-la, é necessário executar o Script A.6 e o Script A.7.

Script A.6 - Selecionando a versão correta da OpenCV.

```
cd ~/opencv_build/opencv

git checkout 3.4.14
```

Script A.7 - Selecionando a versão correta da OpenCV Contrib.

```
cd ~/opencv_build/opencv_contrib

git checkout 3.4.14
```

Explicação do Código:

```
git checkout
```

Altera a versão do ramo (*branch*) da versão atual para, neste caso, a versão 3.4.14.

Em sequência, será criada a pasta onde a OpenCV será efetivamente instalada, conforme o Script A.8.

Script A.8 - Criação do diretório de build.

```
mkdir ~/opencv_build/opencv/build

cd ~/opencv_build/opencv/build
```

É necessário configurar os parâmetros da instalação do software. Para isso, será executado o Script A.9.

Script A.9 - Configurando a build da OpenCV com o CMake.

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_C_EXAMPLES=ON \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D OPENCV_GENERATE_PKGCONFIG=ON \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_build/opencv_contrib/modules \
    -D BUILD_EXAMPLES=ON ..
```

Fonte: disponível em https://linuxize.com/post/how-to-install-opencv-on-ubuntu-18-04/.

Explicação do Código:

```
cmake
```

Este comando é utilizado para controlar o processo de compilação de um software.

Script A.10 - Compilação da OpenCV.

```
make -j1
```

Modifique o valor "1" de acordo com o número de cores presentes no processador. Este valor pode ser encontrado ao executar o comando "nproc" no terminal, que retornará o número desejado.

Explicação do Código:

```
make
```

Este comando é utilizado para detectar automaticamente os arquivos de um programa que devem ser compilados novamente.

Script A.11 - Finalização da instalação da OpenCV.

```
sudo make install
```

Explicação do Código:

```
install
```

Este comando instala o OpenCV, finalizando o processo de instalação.

A.1.4 Preparação das Imagens

No ambiente Linux já configurado, crie um diretório “pasta_principal”. Este procedimento pode ser feito manualmente ou automaticamente conforme o Script A.12.

Script A.12 - Criação de um ambiente de trabalho.

```
mkdir ~/pasta_principal
```

O comando “mkdir” (*make directory*) é utilizado para criar diretórios. A utilização deste comando em combinação com o sinal “til” (~) cria um diretório na pasta pessoal do usuário. Em sequência, é informado o nome do diretório.

As imagens negativas utilizadas neste capítulo podem ser obtidas através do seguinte link: http://www.vision.caltech.edu/html-files/archive.html. Os arquivos TAR utilizados foram: Cars 2001 (Rear), Cars 1999 (Rear) 2, Motorcycles 2001 (Side) e Faces 1999 (Front).

Após a coleta de imagens negativas, estas devem ser armazenadas num diretório, “neg”, dentro do diretório “pasta_principal”, conforme o Script A.13.

Script A.13 - Criação do diretório “neg”.

```
mkdir ~/pasta_principal/neg
```

As imagens ainda não estão padronizadas, já que estão com nomes aleatórios, possuem extensões de arquivos, dimensões, espaços de cores diferentes etc. Para resolver este problema, deve-se abrir o terminal e criar um arquivo “padronizar_imagens.py” dentro do diretório “neg”, como consta o Script A.14.

Script A.14 - Criação do arquivo de padronização de imagens.

```
touch ~/pasta_principal/neg/padronizar_imagens.py
```

Em sequência, é adicionado o Código 7.3 no arquivo “padronizar_imagens.py”.

Código 7.3 - Padronização de imagens em Python.

```
import cv2, os


def padronize_imagens():
    numero_imagem = 1


    for arquivo in os.listdir('.'):
        if arquivo != 'padronizar_imagens.py':
```

```
            try:
                os.rename(arquivo, str(numero_imagem) + '.jpg')

                imagem = cv2.imread(str(numero_imagem) + '.jpg',
cv2.IMREAD_GRAYSCALE)
                imagem_redimensionada = cv2.resize(imagem, (100, 100))

                cv2.imwrite(str(numero_imagem) + '.jpg',
imagem_redimensionada)

                linha = 'neg/' + str(numero_imagem) + '.jpg\n'
                with open('../bg.txt', 'a') as f:
                    f.write(linha)

                numero_imagem += 1

            except Exception as e:
                print(str(e))

padronize_imagens()
```

Explicação do Código:

```
import cv2, os
```

São importadas as importadas as bibliotecas “cv2” (OpenCV) e “os” (Operating System).

```
def padronize_imagens():
```

Criação de uma função. O conteúdo dessa função é a própria padronização das imagens.

```
numero_imagem = 1
```

Esta variável terá seu valor incrementado toda vez que uma nova imagem for lida. Os novos nomes das imagens serão respectivos ao valor atual desta variável.

```
for arquivo in os.listdir('.'):
```

Este trecho de código irá fazer com que o programa percorra todas as imagens disponíveis no diretório atual. Pode ser lido da seguinte maneira: “Para cada arquivo presente neste diretório…”.

```
if arquivo != 'padronizar_imagens.py':
```

Esta verificação garante que o próprio código não terá seu nome alterado conforme o valor da variável “numero_imagem”.

```
try:
```

Não é garantido que todas as imagens serão padronizadas efetivamente. Esta estrutura garantirá um tratamento de exceções caso o código não produza o resultado esperado.

```
os.rename(arquivo, str(numero_imagem) + '.jpg')
```

É utilizada a função “rename”, da biblioteca Operating System. Esta função permite renomear arquivos presentes no sistema operacional. Os argumentos desta função são descritos abaixo:

- 1° argumento: o arquivo que será renomeado.
- 2° argumento: o novo nome do arquivo.

```
imagem = cv2.imread(str(numero_imagem) + '.jpg', cv2.IMREAD_GRAYSCALE)
```

É criada uma variável “imagem” que receberá a leitura da imagem em tons de cinza.

É utilizada a função “imread”, da OpenCV. Esta função permite ler uma determinada imagem. Os argumentos desta função são descritos abaixo:

- 1° argumento: o nome da imagem que se deseja ler. Neste caso, é o novo nome da imagem.
- 2° argumento: o tipo de leitura que será realizado. Neste caso, o modo de leitura é em tons de cinza.

```
imagem_redimensionada = cv2.resize(imagem, (100, 100))
```

É criada uma variável “imagem_redimensionada” que receberá a imagem lida e a redimensionará para as dimensões especificadas.

É utilizada a função “resize”, da OpenCV. Esta função permite redimensionar uma determinada imagem. Os argumentos desta função são descritos abaixo:

- 1° argumento: a imagem que se deseja redimensionar. Neste caso, é a variável criada anteriormente, “imagem”.

- 2° argumento: uma tupla contendo dois valores respectivos às dimensões do eixo X e Y.

```
cv2.imwrite(str(numero_imagem) + '.jpg', imagem_redimensionada)
```

É utilizada a função “imwrite”, da OpenCV. Esta função permite salvar uma determinada imagem. Os argumentos desta função são descritos abaixo:

- 1° argumento: o nome que a imagem receberá.
- 2° argumento: a imagem que se deseja salvar. Neste caso, é a imagem onde foram realizadas as alterações. Nota-se que ambos os nomes são os mesmos, já que se deseja substituir as imagens originais pelas padronizadas. É recomendado fazer uma cópia das imagens originais, caso encontrado algum erro inesperado durante a execução do programa.

```
linha = 'neg/' + str(numero_imagem) + '.jpg\n'
```

É criada uma variável "linha" que receberá o caminho e o nome da imagem e uma quebra de linha.

```
with open('../bg.txt', 'a') as f:
```

É criado e aberto um arquivo de texto "bg.txt" em modo de *append* (onde é possível adicionar dados). Este arquivo irá conter o caminho de todas as imagens negativas.

```
f.write(linha)
```

É escrito no documento "bg.txt" o conteúdo da variável linha.

```
numero_imagem += 1
```

É somado 1 à variável “numero_imagem”.

```
            except Exception as e:
```

Caso a padronização da imagem falhe, o programa irá entender como uma exceção. Ou seja, irá executar os comandos dentro deste bloco de código.

```
print(str(e))
```

O programa irá informar qual foi o erro encontrado durante a padronização da imagem.

```
padronize_imagens()
```

É chamada a função “padronize_imagens()”.

Para finalizar, será executado o Script A.15.

Script A.15 - Padronização de imagens em Python.

```
cd ~/pasta_principal/neg/

python3 padronizar_imagens.py
```

Este comando irá compilar o Código 7.3 e, consequentemente, padronizar as imagens negativas.

Neste ponto, o diretório “pasta_principal” deve conter, essencialmente:

- o diretório “neg”, que contém: as imagens negativas padronizadas e o arquivo “padronizar_imagens.py” e
- o documento “bg.txt”, que contém o caminho das imagens negativas.

A.1.5 Criação do Detector

Como discutido anteriormente, para se obter um detector de objetos com alta acurácia, é necessário realizar um treinamento com um grande número de imagens positivas e negativas.

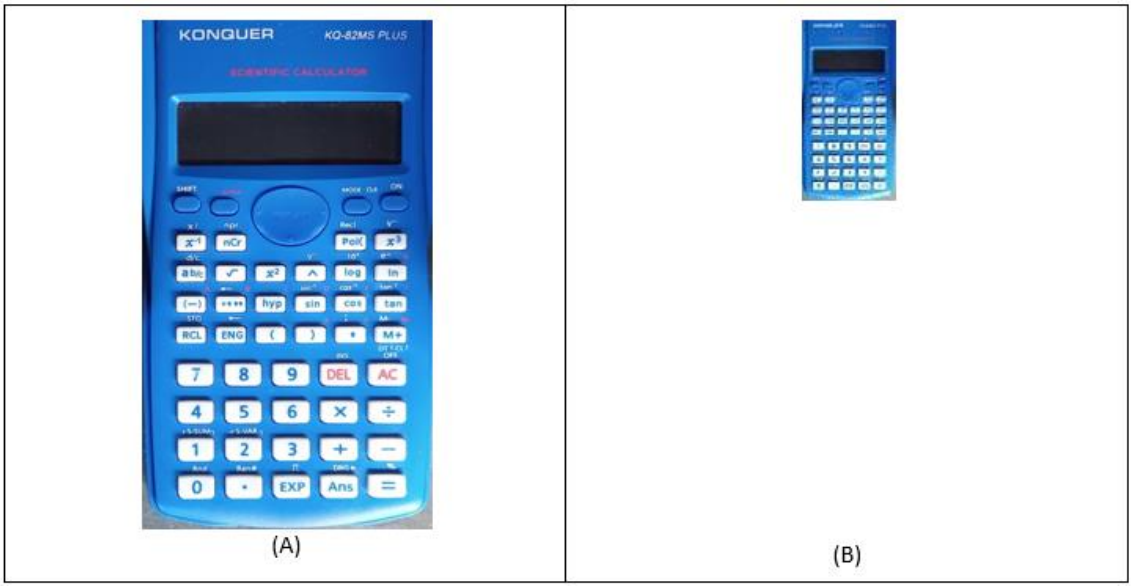
Para isso, é possível utilizar o programa "opencv_createsamples" para se obter imagens positivas, da OpenCV. Este programa irá receber dois tipos de grupos de imagens que serão denominadas (neste capítulo), como "A" e "B".

O primeiro grupo, "A", irá receber a imagem base que será aplicada em outras imagens. O segundo grupo, "B", irá receber as imagens onde as imagens do grupo "A" serão aplicadas.

Na prática, o grupo "A" receberá a imagem positiva, ou seja, a imagem do objeto que se deseja detectar (neste caso, a imagem de uma calculadora). Por sua vez, o grupo "B" receberá todas as imagens negativas.

A OpenCV não mantém o programa "opencv_createsamples" atualizado, mas em sua versão 3.4.14 é possível encontrar o programa disponível. As versões anteriores podem ser encontradas na página de lançamentos (*releases*) no GitHub através do seguinte link: https://github.com/opencv/opencv/releases.

Figura A.4 - Objeto de detecção.



A Figura A.4 (A) foi coletada pelo autor, sendo o objeto de detecção com dimensões originais de 1057 x 2049 pixels. A Figura A.4 (B) será utilizada no processo de criação de imagens positivas. Será aplicada em todas as imagens negativas alterando seu tamanho e ângulo. Sua dimensão é de 50 x 96 pixels.

A imagem deve ser adicionada no diretório “pasta_principal”. Neste caso, a imagem foi nomeada como “calc_50px.jpg”.

Em sequência, um diretório “info” será criado dentro do diretório “pasta_principal”. Este é o local de armazenamento de imagens positivas. Este procedimento pode ser feito manualmente ou automaticamente conforme o Script A.16.

Script A.16 - Criação do diretório “info”.

```
mkdir ~/pasta_principal/info
```

Todos os requisitos estão completos para gerar as imagens positivas através do “opencv_createsamples”. Para utilizar o comando, será executado o Script A.17.

Script A.17 - Geração de imagens positivas.

```
opencv_createsamples -img calc_50px.jpg -bg bg.txt -info info/info.lst -
pngoutput info -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5 -num 1920
```

Explicação do Código:

```
opencv_createsamples
```

Este comando irá, essencialmente, gerar as imagens positivas de acordo com os parâmetros informados. Foi adicionado ao caminho (*path*) da máquina ao instalar a OpenCV.

```
-img calc_50px.jpg
```

É informada a *flag* “img”, que receberá a imagem positiva que será utilizada.

```
-bg bg.txt
```

É informada a *flag* “bg”, que receberá o arquivo “bg.txt” onde os caminhos de todas imagens negativas podem ser encontrados.

```
Info/info.lst
```

É informada a *flag* “info”, que receberá o arquivo “info.lst” contendo, essencialmente, as informações das imagens positivas.

```
-pngoutput info
```

É informada a *flag* “pngoutput”, que receberá a localização onde as novas imagens devem ser armazenadas.

```
-maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5
```

São informadas as *flags* “maxxangle”, “maxyangle” e “maxzangle”, que definirão o máximo de variações possíveis em relação ao ângulo e posição do objeto de detecção. Neste caso, o ângulo pode variar 0.5 nos eixos X, Y e Z (maxxangle, maxyangle e maxzangle, respectivamente).

```
-num 1920
```

É informada a flag “num”, que receberá a quantidade de imagens positivas que deve ser gerada. Neste caso, 1920.

Em sequência, é necessário gerar o arquivo vetor respectivo às imagens positivas. Para isso, deve-se executar o Script A.18.

Script A.18 – Criação do arquivo vetor.

```
Opencv_createsamples -info info/info.lst -num 1920 -w 20 -h 20 -vec
positives.vec
```

Explicação do Código:

```
opencv_createsamples
```

Neste caso, o comando irá gerar o arquivo VEC contendo as especificações das imagens positivas.

```
-w 20 -h 20
```

São informadas as *flags* “w” e “h”, que receberão a altura e largura (respectivamente) do detector. Neste caso, 20 pixels para ambas.

```
-vec
```

É informada a *flag* “vec”, que receberá o nome do arquivo que, essencialmente, servirá de base para o treinamento.

Deve-se, também, criar o diretório que irá conter os arquivos XML respectivos aos estágios, conforme o Script A.19.

Script A.19 - Criação do diretório “data”.

```
mkdir ~/pasta_principal/data
```

Script A.20 - Treinamento do detector.

```
opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 1800
-numNeg 900 -numStages 10 -w 20 -h 20
```

Explicação do Código:

```
opencv_traincascade
```

Este comando receberá parâmetros onde serão especificados os detalhes finais do detector e, finalmente, gerar o arquivo XML de detecção.

```
-data data
```

É informada a *flag* “data”, que receberá o local onde os arquivos dos estágios devem ser armazenados.

```
-vec positives.vec
```

É informada a *flag* “vec”, que receberá o arquivo VEC, “positives.vec”, gerado no Script A.18.

```
-bg bg.txt
```

É informada a *flag* “bg”, que receberá o arquivo contendo o caminho de todas as imagens negativas, “bg.txt”.

```
-numPos 1800 -numNeg 900
```

São informadas as *flags* “numPos” e “numNegs”, que receberão o número de imagens positivas e o número de imagens negativas, respectivamente. Nota-se que este valor varia de acordo com o número de imagens coletadas e geradas previamente. O número de imagens positivas deve ser aproximadamente o dobro do número de imagens negativas.

```
-numStages 10
```

É informada a *flag* “numStages”, que receberá a quantidade de estágios que devem ser realizados. Neste caso, 10.

```
-w 20 -h 20
```

Novamente, são informadas as *flags* “w” e “h”, que receberão a altura e largura (respectivamente) do detector. Neste caso, 20 pixels para ambas.

Nota-se que o treinamento pode ser demorado, já que depende da configuração da máquina, da quantidade de imagens positivas e negativas, quantidade de estágios etc. Quando o treinamento for finalizado, será gerado um arquivo XML, “cascade”, dentro do diretório “data”. Este arquivo é o resultado do treinamento e deve ser utilizado juntamente com a OpenCV para a tarefa de detecção. O arquivo será copiado para o diretório “pasta_principal”. Este procedimento pode ser feito manualmente ou automaticamente conforme o Script A.21.

Script A.21 – Cópia do arquivo “cascade.xml”.

```
cp ~/pasta_principal/data/cascade.xml ~/pasta_principal/cascade.xml
```

Explicação do Código:

```
cp
```

O comando “cp” copia o conteúdo informado como primeiro argumento para a localização especificada no segundo.

O nome do arquivo pode ser modificado de acordo com o objeto de detecção para melhor compreensão. Neste caso, foi renomeado como “calccascade10stages.xml”.

Finalmente, será criado o arquivo para testar o resultado do treinamento. Este procedimento pode ser feito manualmente ou automaticamente conforme o Script A.22.

Script A.22 - Criação do diretório “info”.

```
touch ~/pasta_principal/deteccao.py
```

Em sequência, é adicionado o Código 7.4 no arquivo “deteccao.py”.

Código 7.4 - Detecção da calculadora em Python.

```
import cv2


imagem = cv2.imread("img.jpg")


porcentagem_escala = 0.6

largura, altura  = int(imagem.shape[1] * porcentagem_escala),
int(imagem.shape[0] * porcentagem_escala)

proporcao = (largura, altura)

imagem = cv2.resize(imagem, proporcao, interpolation = cv2.INTER_AREA)


imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)


detector_calculadora = cv2.CascadeClassifier("calccascade10stage.xml")

calculadoras = detector_calculadora.detectMultiScale(imagem_cinza,
scaleFactor = 1.1, minNeighbors = 8, minSize = (20, 20))


fonte = cv2.FONT_HERSHEY_SIMPLEX
```

```
for (x, y, w, h) in calculadoras:

    cv2.putText(imagem, 'Calculadora', (x-w, y-h), fonte, 0.5, (11, 255, 
255), 2, cv2.LINE_AA)



cv2.imshow("Calculadora", imagem)

cv2.waitKey(0)
```

É necessário adicionar a imagem JPG que se deseja realizar a detecção no mesmo diretório onde está armazenado o arquivo “deteccao.py”. A imagem deve ser nomeada como “img.jpg”.

Explicação do Código:

```
import cv2
```

É importada a biblioteca “cv2” (OpenCV).

```
imagem = cv2.imread("img.jpg")
```

É criada uma variável “imagem”, que terá como valor a leitura da imagem desejada utilizando a função “imread”, da OpenCV.

```
porcentagem_escala = 0.6
```

É criada uma variável “porcentagem_escala”, que receberá o valor 0.6. Esta variável será utilizada para diminuir o tamanho da imagem original.

```
largura, altura  = int(imagem.shape[1] * porcentagem_escala), 
int(imagem.shape[0] * porcentagem_escala)
```

São criadas duas variáveis, “largura” e “altura”. A primeira receberá a largura atual da imagem e multiplicará pela variável “porcentagem_escala” (ou seja, 0.6). A segunda receberá a altura atual da imagem e multiplicará variável “porcentagem_escala”. Como resultado, ambas as variáveis estarão proporcionalmente iguais às originais.

```
proporcao = (largura, altura)
```

É criada uma variável “proporcao” que receberá uma tupla contendo as variáveis “largura” e “altura”, respectivamente.

```
imagem = cv2.resize(imagem, proporcao, interpolation = cv2.INTER_AREA)
```

O valor da variável “imagem” é atualizado, recebendo a imagem lida e a redimensionando para as dimensões especificadas.

Novamente é utilizada a função “resize”, da OpenCV. Esta função permite redimensionar uma determinada imagem. Os argumentos desta função são descritos abaixo:

- 1° argumento: o nome da imagem que se deseja redimensionar. Neste caso, é a variável criada anteriormente, “imagem”.
- 2° argumento: uma tupla contendo dois valores respectivos às dimensões do eixo X e Y.
- 3° argumento: define-se o tipo de interpolação. Neste caso, é utilizado a “INTER_AREA”, que realiza uma reamostragem da imagem utilizando a relação entre pixels de uma área.

```
imagem_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
```

É criada uma variável “imagem_cinza”, que armazenará a imagem lida convertida para tons de cinza, utilizando a função “cvtColor”, da OpenCV.

```
detector_calculadora = cv2.CascadeClassifier("calccascade10stage.xml")
```

É criada uma variável “detector_calculadora” que será utilizada para calcular as características Haar utilizando a função “CascadeClassifier”, da OpenCV.

```
calculadoras = detector_calculadora.detectMultiScale(imagem_cinza,
scaleFactor = 1.1, minNeighbors = 8, minSize = (20, 20))
```

É criada uma variável “calculadoras” que será utilizada para realizar a detecção de objetos na imagem de entrada utilizando a função “detectMultiScale”, da OpenCV.

```
fonte = cv2.FONT_HERSHEY_SIMPLEX
```

É criada uma variável “fonte” que receberá a fonte Hershey Simplex da OpenCV.

```
for (x, y, w, h) in calculadoras:
```

Este trecho de código irá fazer com que o programa percorra pelos valores “x”, “y”, “w” e “h” da variável “calculadoras”. Pode ser lido da seguinte maneira: “Para cada valor presente na variável “calculadoras”…”.

```
cv2.putText(imagem, 'Calculadora', (x-w, y-h), fonte, 0.5, (11, 255,
255), 2, cv2.LINE_AA)
```

É utilizada a função “putText” da OpenCV, que recebe:

- 1° argumento: a variável “image”, com a(s) calculadora(s) já detectada(s).
- 2° argumento: o texto que deve ser adicionado.
- 3° argumento: uma tupla contendo os pontos (x,y) onde deve ser adicionado o texto.
- 4° argumento: a fonte que deve ser utilizada.
- 5° argumento: fator de escala da fonte.
- 6° argumento: uma tupla contendo os valores RGB da fonte.
- 7° argumento: espessura da fonte em pixels.
- 8° argumento: tipo de linha que deve ser utilizada.

```
cv2.imshow("Calculadora", imagem)
```

A função “imshow”, da OpenCV, irá apresentar a imagem final, recebendo como primeiro parâmetro o nome da janela e como segundo a própria imagem “imagem”.

```
cv2.waitKey(0)
```

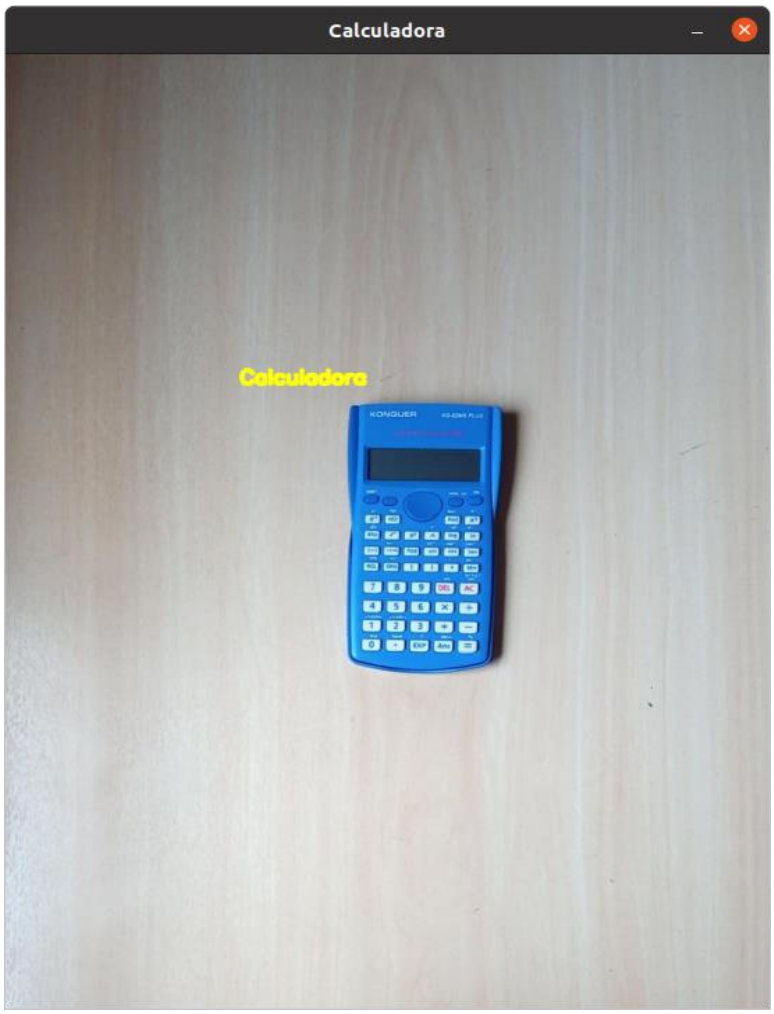
A função “waitKey”, da OpenCV, irá finalizar o programa quando for recebida determinada entrada do teclado. Neste caso, qualquer entrada.

Para compilar o programa deve ser executado o Script A.23.

Script A.23 - Criação do diretório “info”.

```
python3 pasta_principal/deteccao.py
```

Figura A.5 - Resultado da função “imshow”, da OpenCV.



O resultado final, ou seja, a imagem com os textos apontando a localização da(s) calculadora(s) é apresentada. O programa se encerra quando recebe alguma entrada do teclado.

O arquivo XML criado e utilizado para a detecção da calculadora pode ser encontrado através do seguinte link: https://github.com/andre-alck/IC-RF-HAAR/blob/main/haarcascade_calculator.xml.

A.2 **Criação de um Detector a partir do *Cascade Trainer Gui***

É possível criar o arquivo final de detecção XML a partir do Cascade Trainer GUI - um software especializado em criar e testar detectores de objetos utilizando o *framework* discutido ao longo do capítulo.

Embora este software não possua suporte para sistemas Linux, é possível instalá-lo no sistema operacional Windows 10. O arquivo de download pode ser obtido através do seguinte link:

https://amin-ahmadi.com/downloadfiles/cascadetrainergui/CascadeTrainerGUI_3.3.1_x64_Setup.exe
Quando o download estiver concluído, deve ser executado o arquivo e instalado conforme a Figura A.6 (Parte A - F).

Figura A.6 (A) - Instalação do Cascade Trainer GUI.

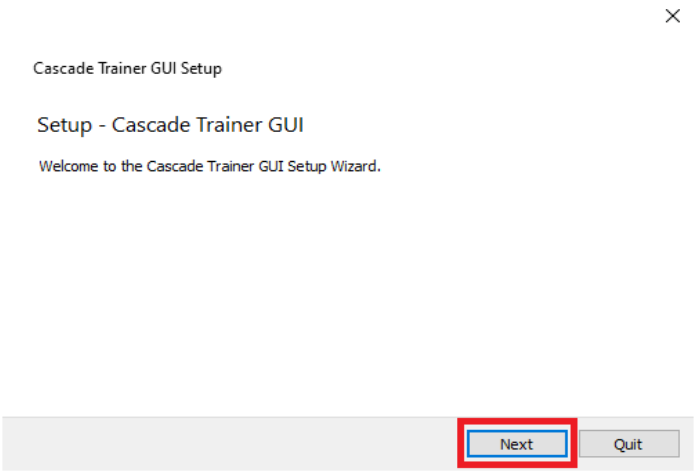


Figura A.6 (B)

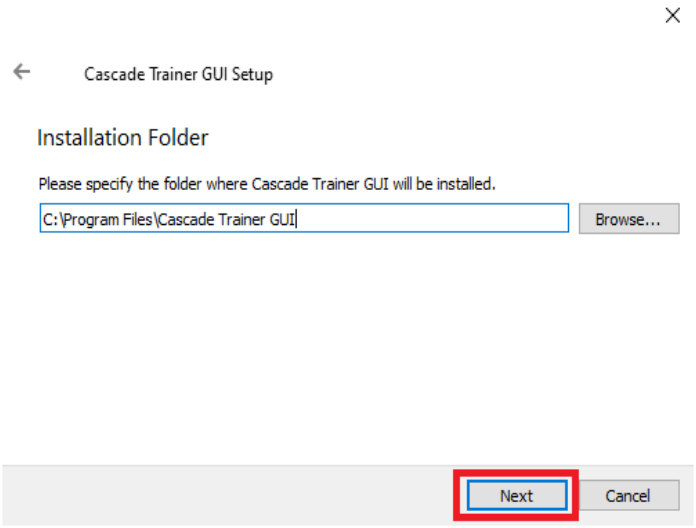


Figura A.6 (C)

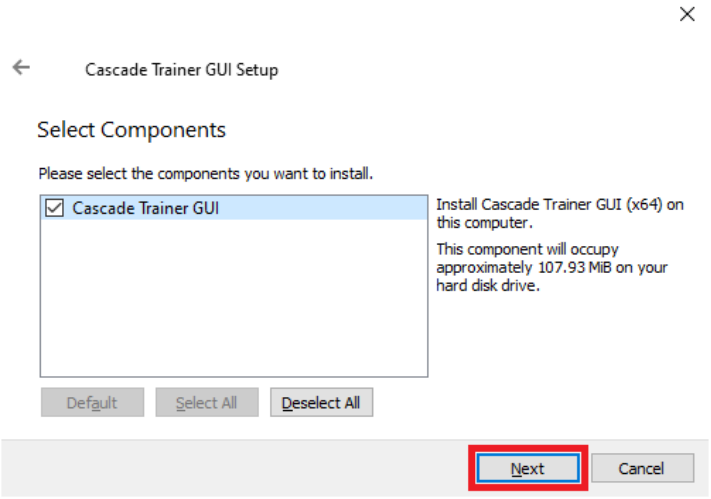


Figura A.6 (D)

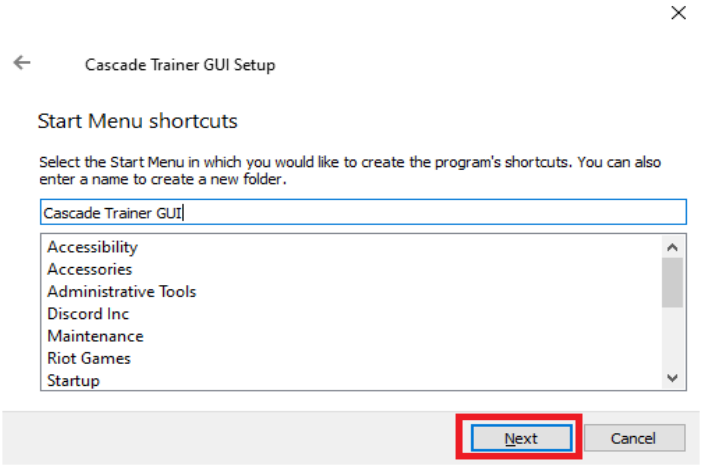


Figura A.6 (E)

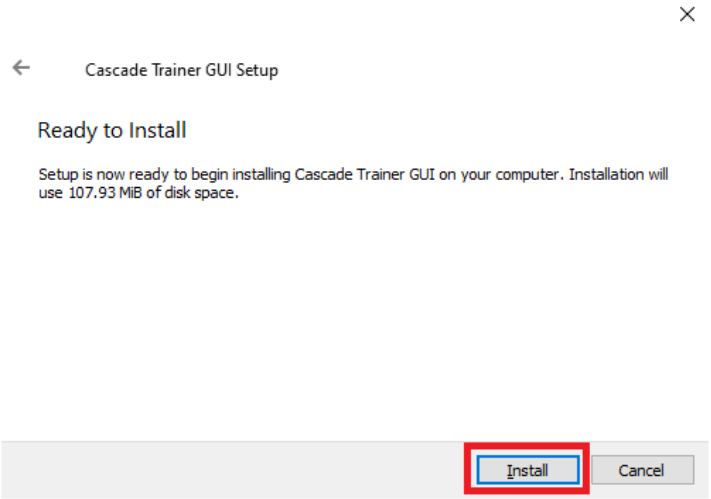
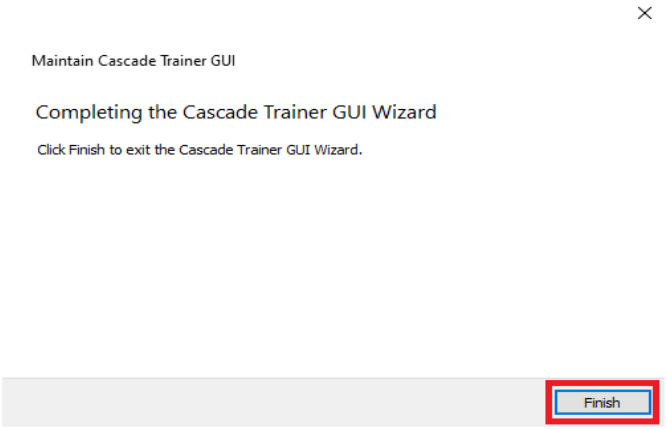


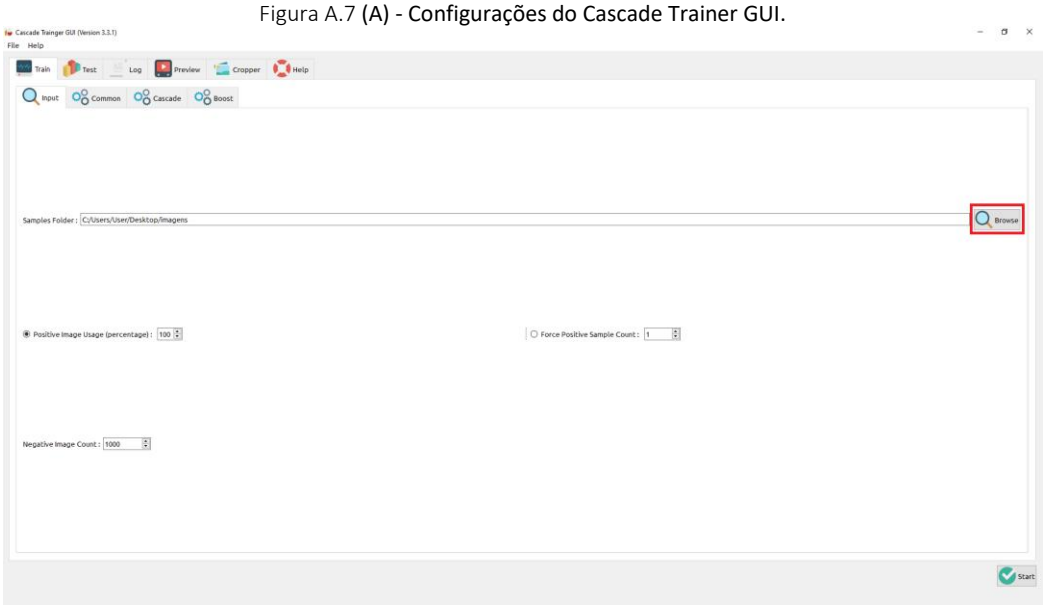
Figura A.6 (F)



É necessário criar um diretório “imagens” contendo dois diretórios “p” e “n”, contendo as imagens positivas e negativas, respectivamente.

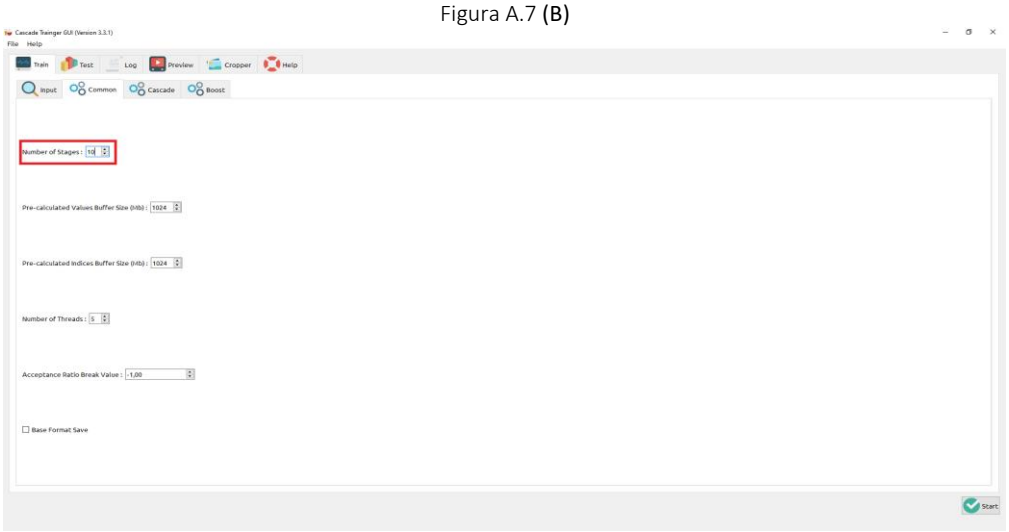
Para efetivamente criar o arquivo XML respectivo ao detector, deve-se seguir o processo como consta a Figura A.7 (A - D)

Figura A.7 (A) - Configurações do Cascade Trainer GUI.



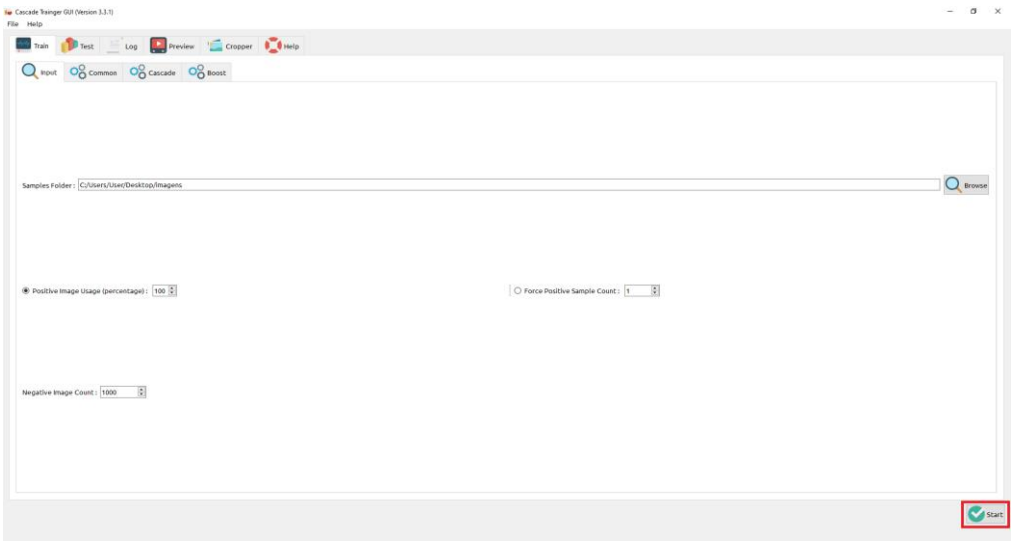
Na página inicial da aplicação, deve-se informar o caminho até o diretório “imagens” criado anteriormente.

Figura A.7 **(B)**



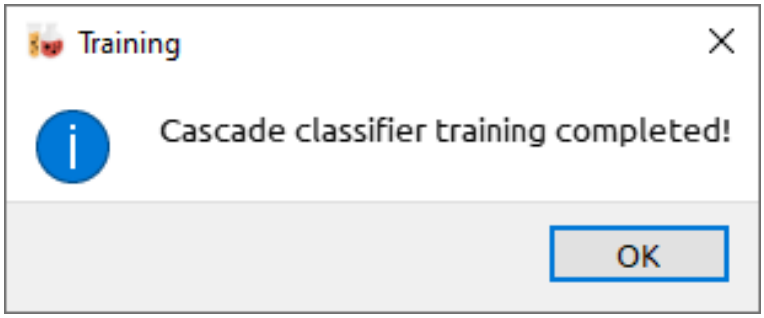
O número de estágios foi ajustado para 10. Embora este ajuste não seja necessário, quanto maior o número de estágios, mais lenta será realizada a detecção.

Figura A.7 (C)



Assim como anteriormente, o treinamento pode ser demorado, já que depende da configuração da máquina, da quantidade de imagens positivas e negativas, quantidade de estágios etc.

Figura A.7 (D)



Quando o treinamento for finalizado, um arquivo XML, “cascade”, terá sido criado no diretório “classifier” (este foi automaticamente criado dentro do diretório “imagens”). Este arquivo é o resultado do treinamento e já pode ser utilizado na tarefa de detecção da calculadora.

Anexo B

Preparação do ambiente para treinamento e detecção utilizando CNN

O Anexo B é referente ao Capítulo 09 - CNN – Redes Neurais Convolucionais e apresenta descrições aprofundadas sobre instalação de software para virtualização de máquinas, instalação de um Sistema Operacional para executar testes e Scripts, treinamento de uma rede neural artificial etc.

B.1 Instalação Do Vmware

O Sistema Operacional utilizado para a tarefa de detecção facial será o Linux Mint 20.03, uma distribuição que vem ganhando muita popularidade pela estabilidade, leveza e simplicidade de utilização. Além disso, pelo fato do sistema ser baseado em Debian, é similar às distribuições encontradas na maioria serviços de *cloud computing* (computação em nuvem, em português) disponíveis (como a Amazon Web Services, por exemplo). O sistema pode ser obtido através do seguinte link: http://mirror.ufscar.br/mint-cd/stable/20.3/linuxmint-20.3-cinnamon-64bit.iso.

A instalação do sistema será realizada em uma máquina virtual utilizando o software gratuito VMware Workstation 16 Player. O software permite a virtualização de diferentes sistemas operacionais, permitindo, também, a configuração de recursos físicos e virtuais presentes em cada computador.

Para instalação do programa, é pressuposto que o usuário esteja utilizando um Sistema Operacional Microsoft Windows 10. Neste caso, as especificações do sistema são: Microsoft Windows 10 Versão 21H2, Compilação do Sistema Operacional 19044.1526.

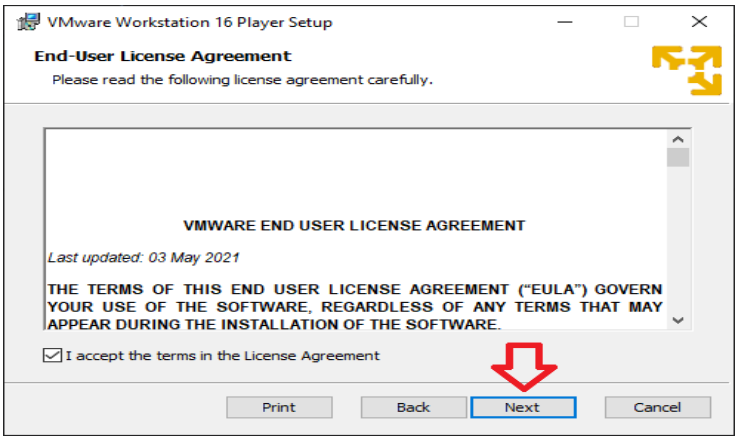
Caso o usuário esteja utilizando uma distribuição Linux baseada em Debian, não é necessário executar o passo de instalação da máquina virtual. Finalizando, caso o usuário esteja utilizando um Sistema Operacional OS X, o processo de instalação da máquina virtual pode ser realizado através do VirtualBox, cujo arquivo de instalação pode ser obtido através do seguinte link: https://download.virtualbox.org/virtualbox/6.1.32/VirtualBox-6.1.32-149290-OSX.dmg.

O arquivo de instalação para máquinas executando o Microsoft Windows 10 pode ser obtido através do seguinte link: https://www.vmware.com/go/getplayer-win. Após o arquivo ser baixado, deve ser executado e instalado conforme Figura B.1 (Parte A - G).

Figura B.1 (A) - Instalação do VMware Workstation 16 Player

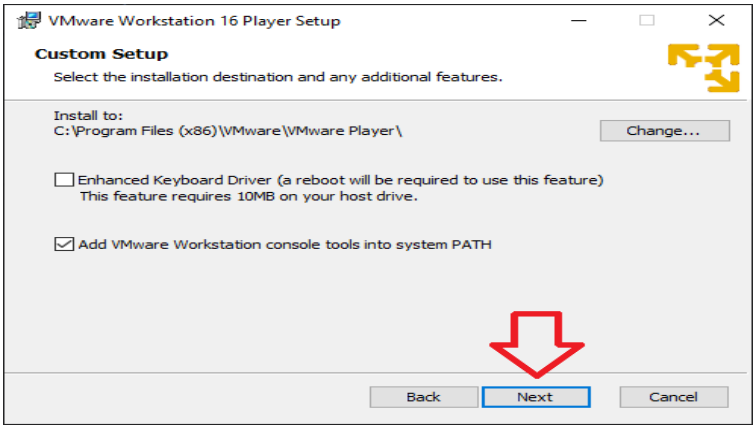
Para instalar o software, é necessário selecionar as opções destacadas pela flecha vermelha.

Figura B.1 (B)



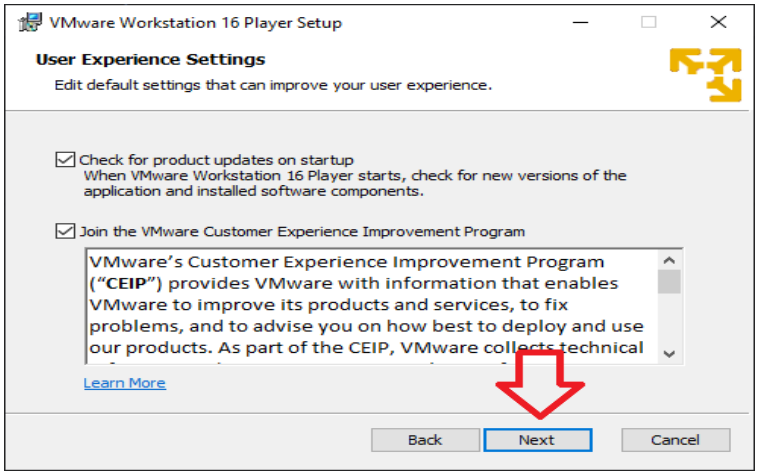
Antes de instalar o software, deve-se ler o acordo de licença do usuário final e, somente se estiver de acordo com todos os termos, marcar a opção "I accept the terms in the License Agreement".

Figura B.1 (C)



Dependendo do uso do software, as ferramentas do console podem ser úteis. Portanto, é recomendável marcar a opção “Add VMware Workstation console tools into system PATH”, para garantir que as ferramentas estarão disponíveis para uso independentemente do local em que o usuário se encontra.

Figura B.1 (D)

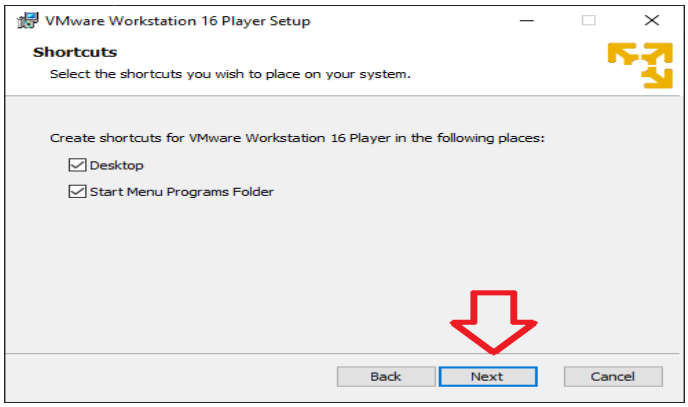


A fim de manter o programa atualizado, deve-se marcar a opção “Check for product updates on startup

When Vmware Workstation 16 Player starts, check for new versions of the application and installed software components.” para o sistema automaticamente se o mesmo está em sua última versão.

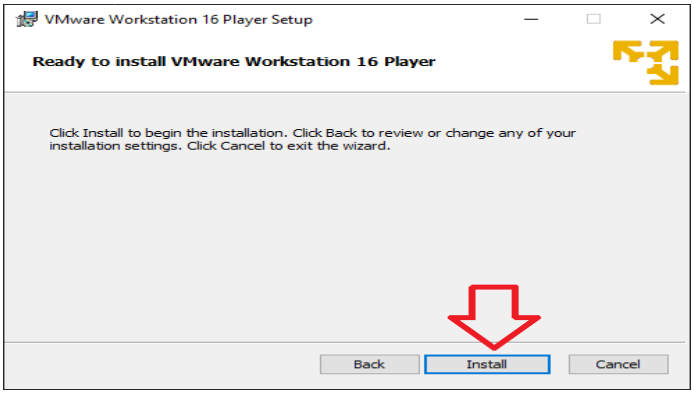
A opção “Join the VMware Customer Experience Improvement Program” é opcional. Neste caso, foi selecionada para informar à empresa sobre as experiências com o software.

Figura B.1 (E)



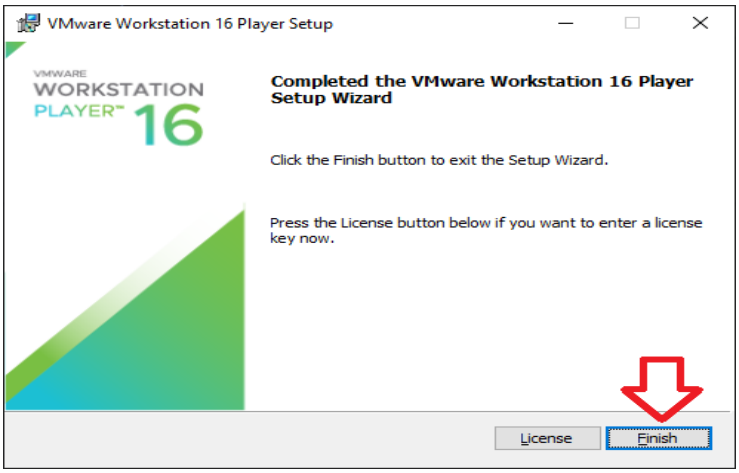
As opções “Desktop” e “Start Menu Programs Folder” são opcionais, mas foram selecionadas, neste caso, para facilitar a execução do programa, já que estarão na Área de Trabalho do Sistema Operacional.

Figura B.1 (F)



Para efetivamente instalar o programa, seleciona-se a opção “Install”.

Figura B.1 (G)



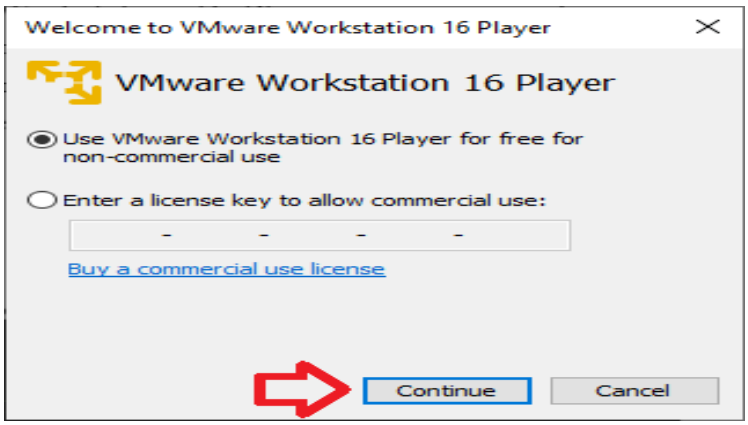
Finalmente, ao clicar na opção “Finish”, o instalador será encerrado.

B.2 Criação Da Máquina Virtual

Para instalar o Linux Mint 20.03 no VMware Workstation 16 Player, é necessário baixar a mídia de instalação do sistema, disponível em http://mirror.ufscar.br/mint-cd/stable/20.3/linuxmint-20.3-xfce-64bit.iso.

Quando o arquivo terminar de ser baixado, é necessário executar o VMware Workstation 16 Player e seguir o tutorial de instalação de acordo com a Figura B.2 (Parte A - I).

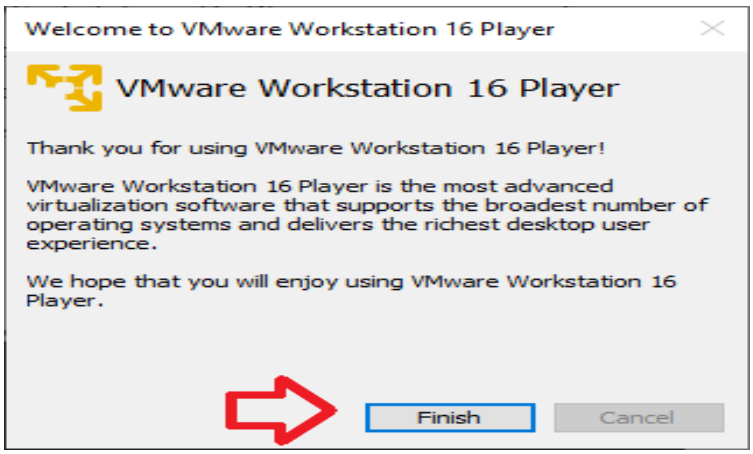
Figura B.2 (A) - Criação da máquina virtual



Para instalar o software, é necessário selecionar as opções destacadas pela flecha vermelha.

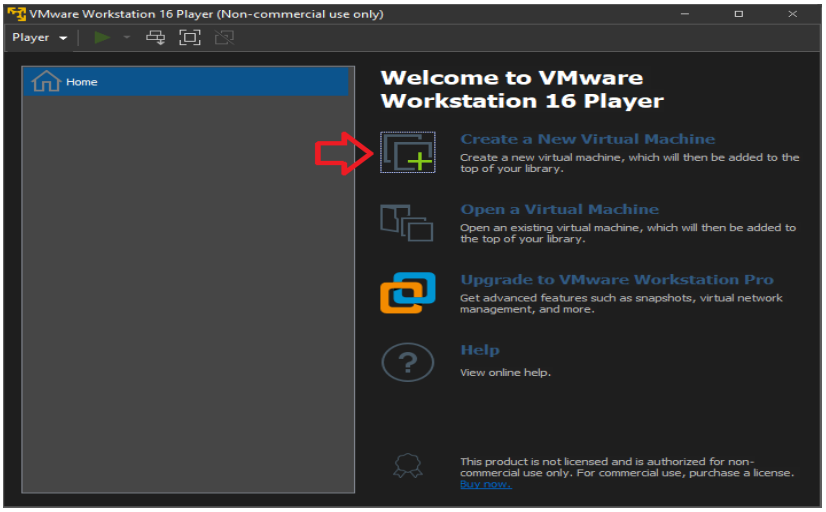
Neste caso, como o software não será utilizado para uso comercial, seleciona-se a primeira opção.

Figura B.2 (B)



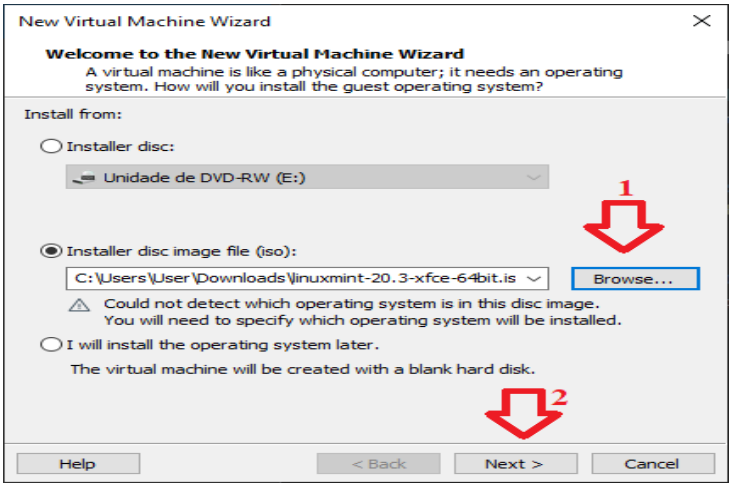
Ao clicar na opção “Finish”, é finalizada a configuração inicial.

Figura B.2 (C)



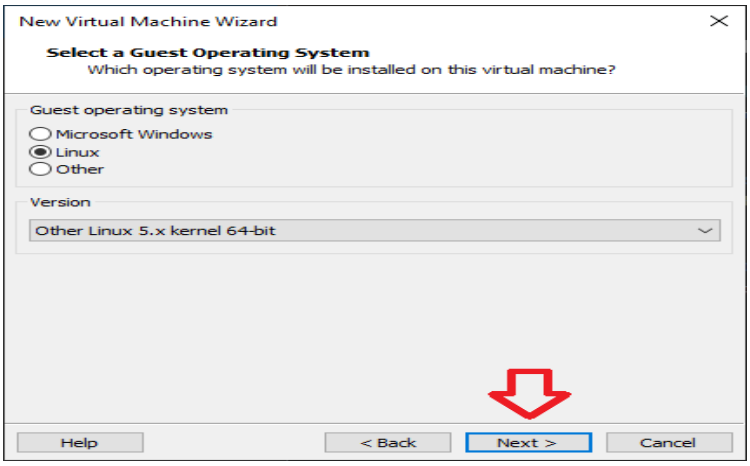
Para criar uma nova máquina virtual, deve-se selecionar a opção “Create a New Virtual Machine”.

Figura B.2 (D)



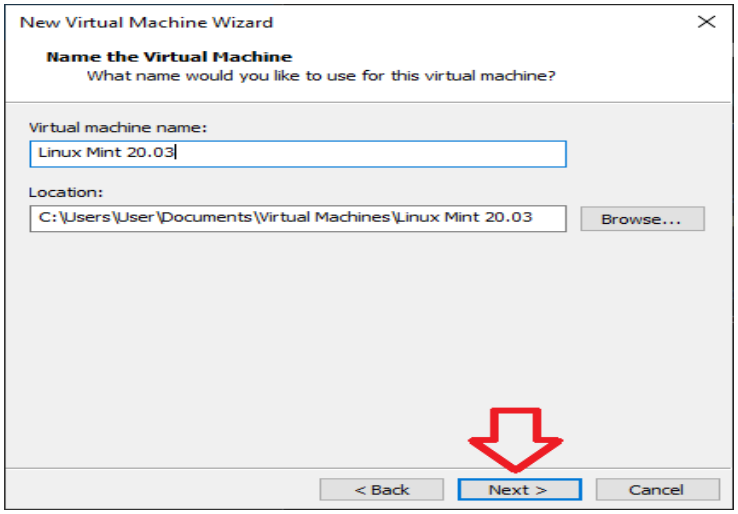
A opção “Browse”, destacada pela flecha 1, permite a procura da imagem ISO previamente baixada. Após informado o caminho em que a imagem se encontra, deve-se selecionar a opção “Next”, indicada pela flecha 2, para dar sequência à instalação.

Figura B.2 (E)



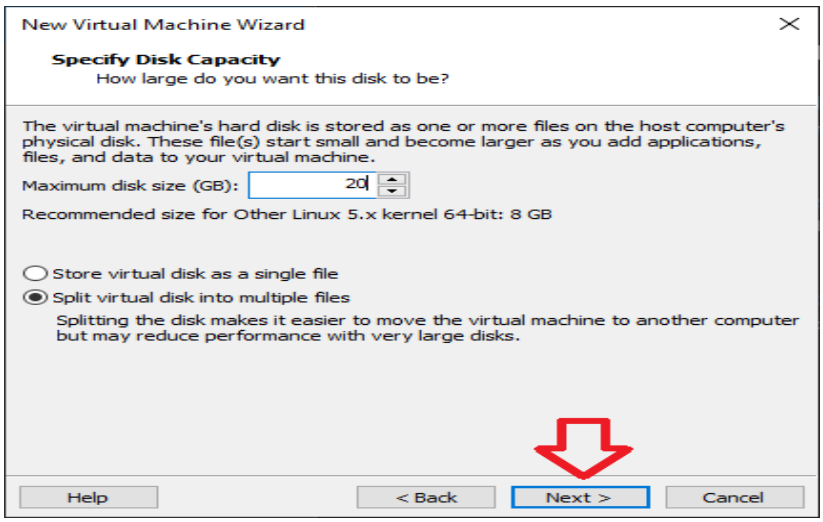
A versão especificada na opção “Version” pode ser alterada, dependendo da versão do sistema que será utilizado.

Figura B.2 (F)



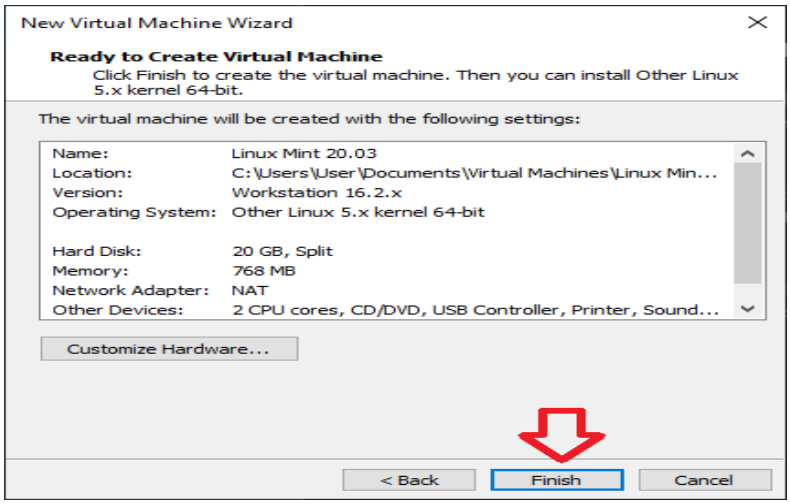
O nome da máquina também pode ser alterado. Neste caso, a máquina será nomeada como “Linux Mint 20.03”.

Figura B.2 (G)



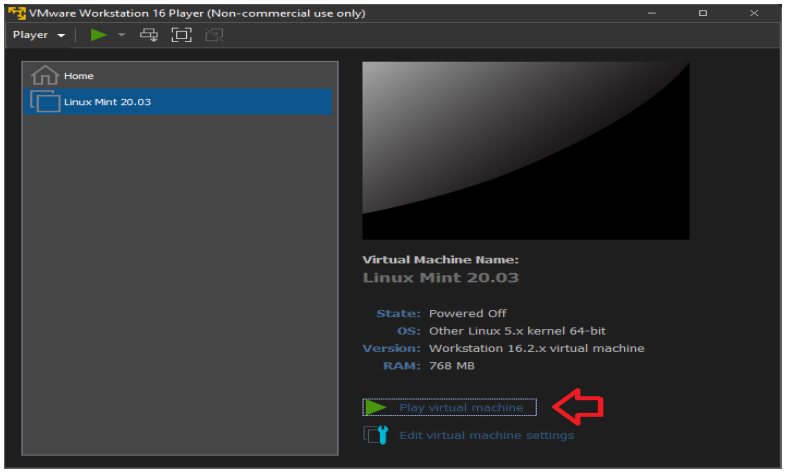
Neste caso, o sistema poderá utilizar 20 gigas de armazenamento. Nota-se que a distribuição não necessita dessa quantidade, mas é vantajoso não utilizar todo o armazenamento para um melhor funcionamento do sistema.

Figura B.2 (H)



Para finalizar a configuração da máquina, deve-se selecionar a opção “Finish”.

Figura B.2 (I)



Neste ponto, a imagem já está configurada e pronta para uso. Para executá-la, seleciona-se a opção “Play virtual machine”, destacada pela flecha vermelha.

B.3 Instalação do Sistema Operacional

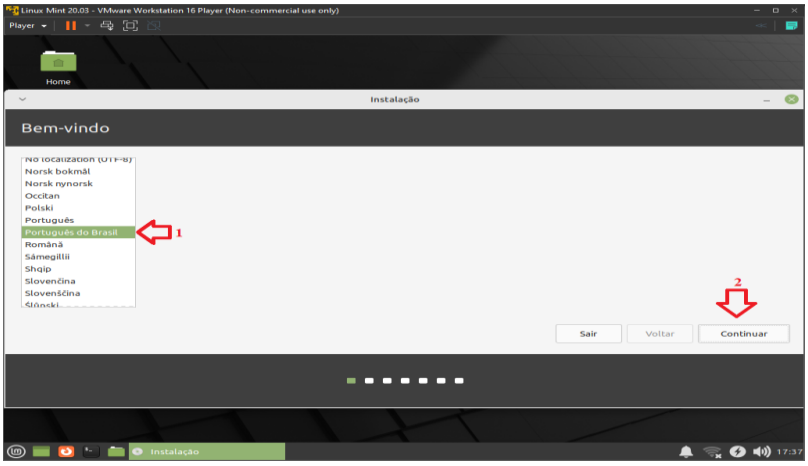
Ao selecionar a opção “Play virtual machine”, conforme Figura B.2 (Parte I), o sistema será executado no modo “Live”. Para efetivamente instalar o sistema, é necessário seguir o tutorial conforme Figura B.3 (A - L).

Figura B.3 (A)

Para instalar o software, é necessário selecionar as opções destacadas pela flecha vermelha.

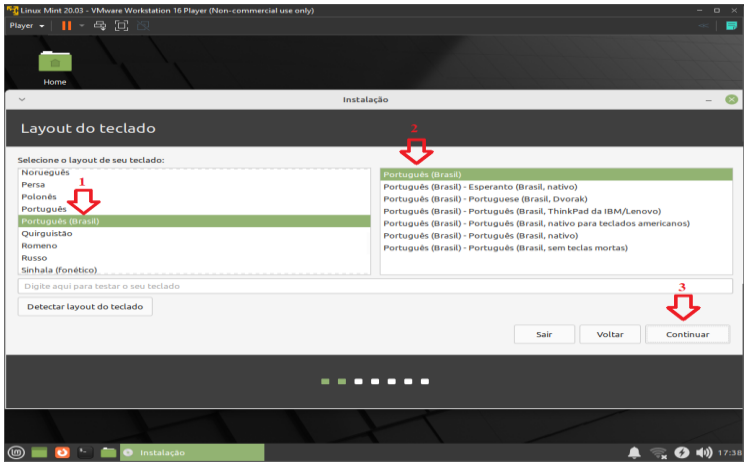
O sistema, até este ponto, está sendo executado em modo “Live”, onde não é necessário fazer alterações no disco para utilizá-lo. Em compensação, para ter acesso a todas as funcionalidades do sistema, deve-se o mesmo por completo.

Figura B.3 (B)



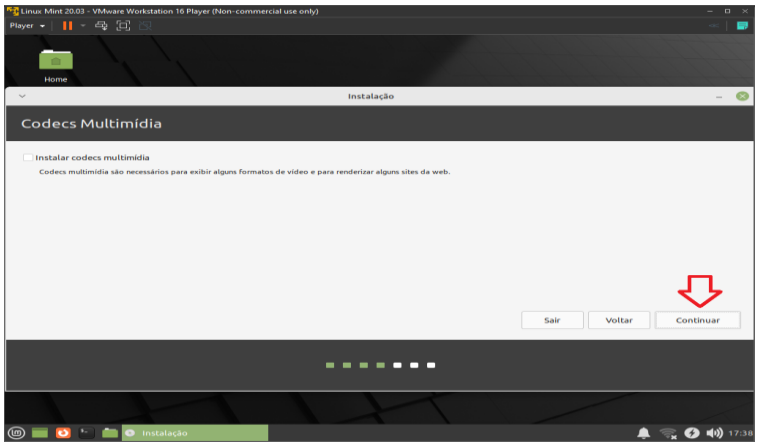
Neste caso, foi escolhida a opção “português do Brasil”. Nota-se, no entanto, que esta preferência pode ser alterada posteriormente e sua alteração não prejudica o processo de detecção.

Figura B.3 (C)



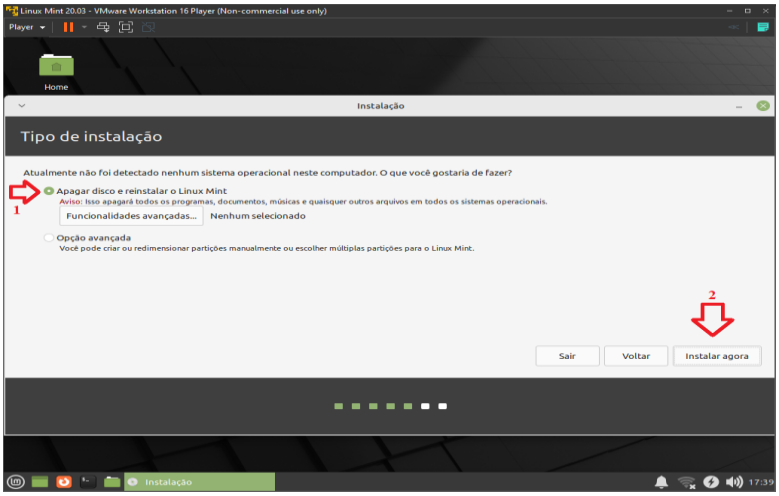
Através da opção destacada pela flecha denominada “2”, especifica-se a versão da correta língua escolhida.

Figura B.3 (D)



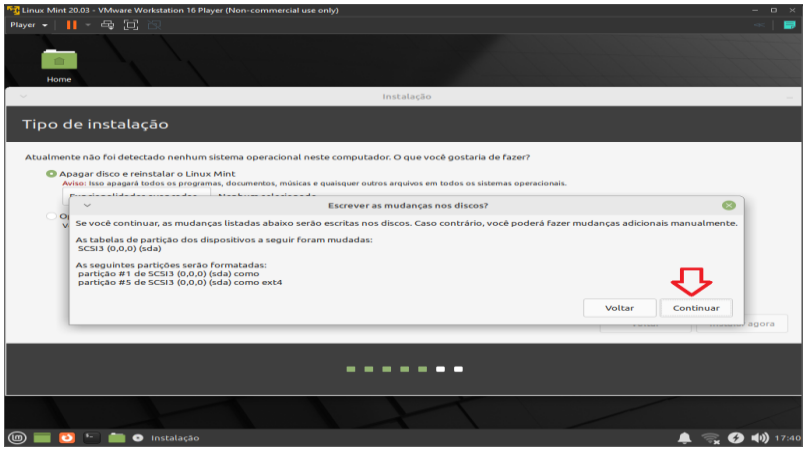
Não é necessário selecionar a opção “Instalar codecs multimídia” pois o uso da máquina virtual, neste caso, servirá somente ao propósito de detecção facial, dispensando a necessidade de recursos adicionais.

Figura B.3 (E)



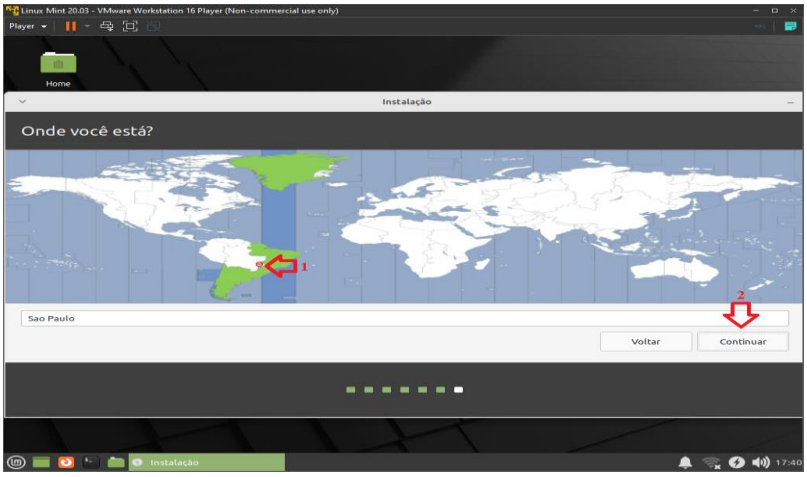
A flecha “1” indica que todos os dados contidos previamente neste disco serão deletados. Nota-se que o disco utilizado pela máquina virtual não é o mesmo utilizado pela máquina host.

Figura B.3 (F)



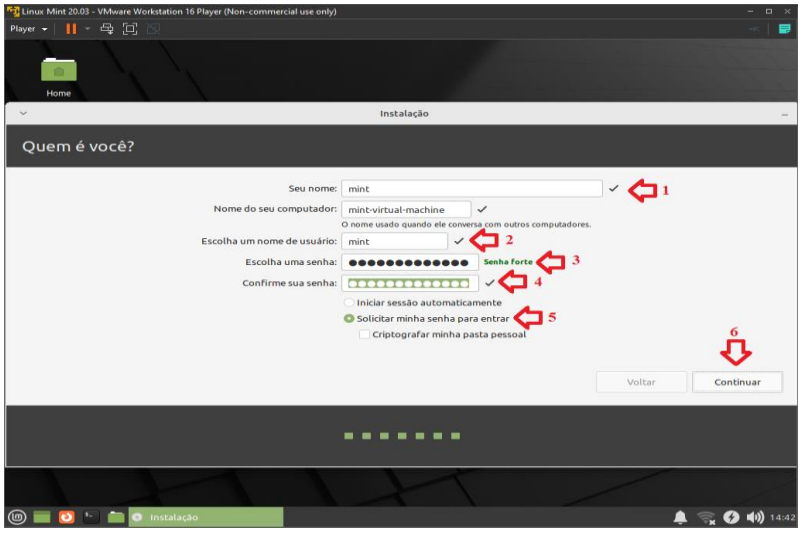
É apresentado um resumo contendo informações a respeito das mudanças que serão realizadas.

Figura B.3 (G)



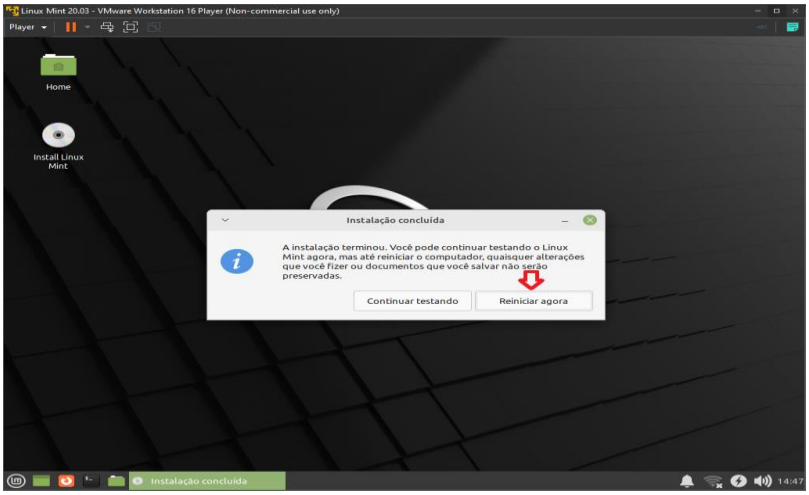
Neste caso, a região escolhida foi “São Paulo”. Escolhendo a região mais próxima de si, haverá maior velocidade de transferência de dados, considerando a proximidade do servidor com a máquina host.

Figura B.3 (H)



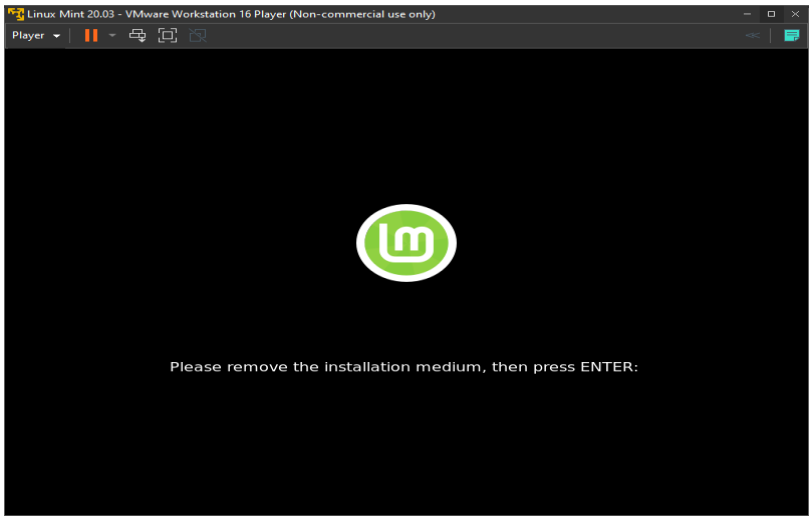
É recomendável que se utilize uma senha considerada forte por motivos de segurança.

Figura B.3 (I)



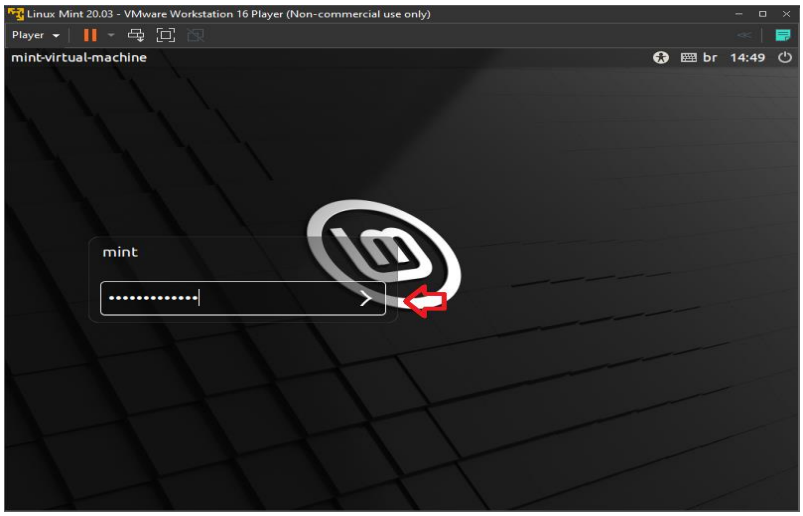
Fonte: após a reinicialização da máquina virtual, todas as alterações terão sido aplicadas.

Figura B.3 (J)



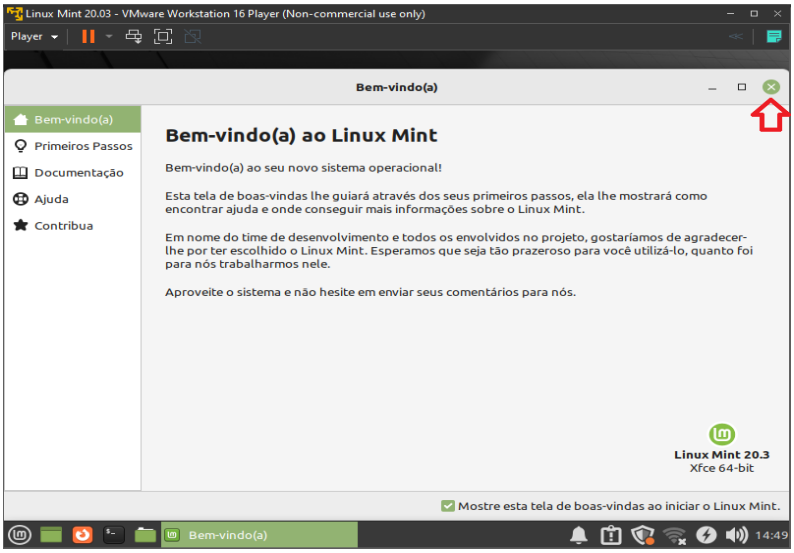
Deve-se pressionar a tecla ENTER para prosseguir com a inicialização.

Figura B.3 (K)



Para realizar o login, deve-se informar as credenciais de usuário criadas durante o processo de instalação.

Figura B.3 (L)



Para utilizar o sistema operacional, basta fechar a janela.

Para finalizar, é necessário executar 2 comandos no terminal, conforme Script B.1, para atualizar o sistema e garantir que todos seus pacotes estão em suas últimas versões. O terminal pode ser aberto através do atalho CTRL + ALT + T.

Script B.1 - Atualização da máquina

```
apt update && apt upgrade
```

O comando “apt” (Advanced Package Tool) administra pacotes de Softwares presentes em distribuições Linux baseadas em Debian.

O comando “update” procura atualizações disponíveis nos repositórios oficiais e compara as versões instaladas na máquina com as últimas aprovadas e disponíveis.

O operador “&&” concatena dois comandos, adicionando o segundo à fila.

O comando “upgrade” baixa as atualizações pendentes e, finalmente, atualiza todos os itens desatualizados.

B.4 Instalação de Dependências

Em vez de instalar a versão padrão do Python, será instalada a Anaconda, uma distribuição das linguagens de programação Python e R para computação científica. Para isso, deve-se instalar todos os requisitos necessários, conforme o Script B.2.

Script B.2 - Instalação de pré-requisitos

```
apt install libgl1-mesa-glx libegl1-mesa libxrandr2 libxss1 libxcursor1
libxcomposite1 libasound2 libxi6 libxtst6
```

Fonte: (adaptado) disponível em https://docs.anaconda.com/anaconda/install/linux/#.

Em sequência, deve-se baixar o instalador do Anaconda para Linux, disponível em https://repo.anaconda.com/archive/Anaconda3-2021.11-Linux-x86_64.sh. Para garantir que o arquivo baixado não está corrompido, é recomendado verificar seu hash, conforme Script B.3.

Script B.3 - Verificação de integridade

```
sha256sum ~/Downloads/Anaconda3-2021.11-Linux-x86_64.sh
```

Fonte: disponível em https://docs.anaconda.com/anaconda/install/linux/#.

É necessário verificar o diretório onde o arquivo foi baixado, bem como seu nome.

Neste caso, a saída produzida foi "fedf9e340039557f7b5e8a8a86affa9d299f5e9820144bd7b92ae9f7ee08ac60", que condiz exatamente com a disponível no website oficial: https://docs.anaconda.com/anaconda/install/hashes/Anaconda3-2021.11-Linux-x86_64.sh-hash/#:~:text=fedf9e340039557f7b5e8a8a86affa9d299f5e9820144bd7b92ae9f7ee08ac60. Ou seja, o arquivo baixado não está corrompido e condiz com o presente no website.

Para instalar o Anaconda, deve-se executar os comandos presentes no Script B.4.

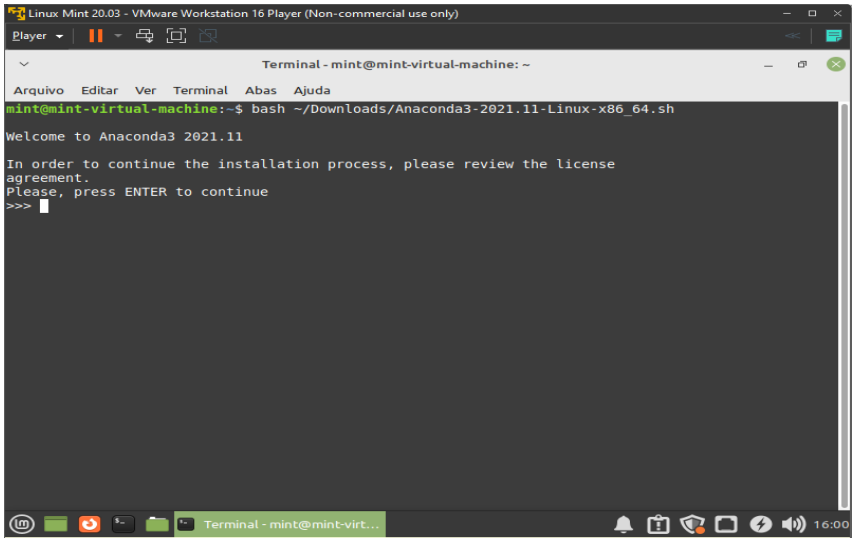
Script B.4 - Atualização da máquina

```
bash ~/Downloads/Anaconda3-2021.11-Linux-x86_64.sh
```

Fonte: disponível em https://docs.anaconda.com/anaconda/install/linux/#.

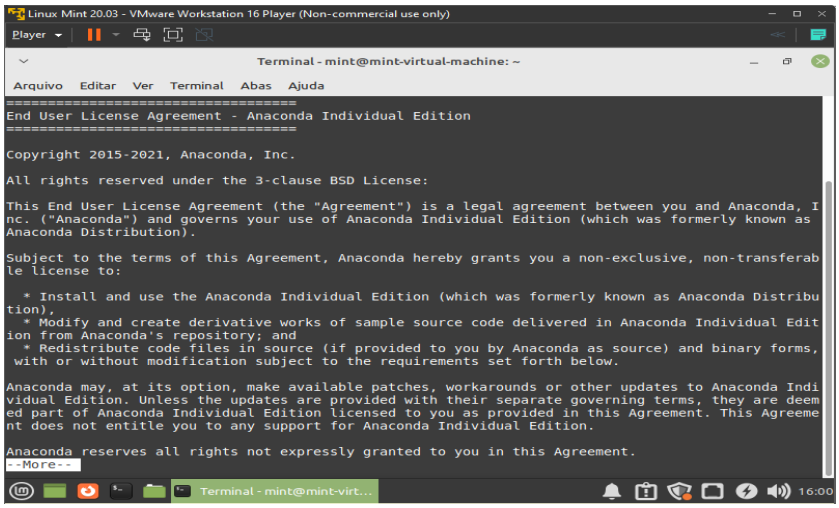
Dependendo do arquivo baixado, o instalador pode solicitar entradas do usuário, conforme Figura B.4 (A - E).

Figura B.4 (A) - Instalação do Anaconda



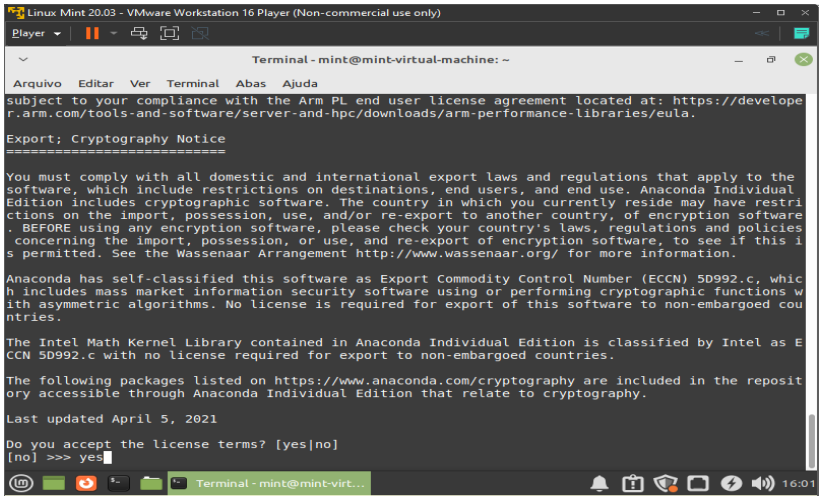
Ao pressionar a tecla ENTER, é informado ao instalador que o usuário está de acordo com o acordo de licença do usuário final.

Figura B.4 (B)



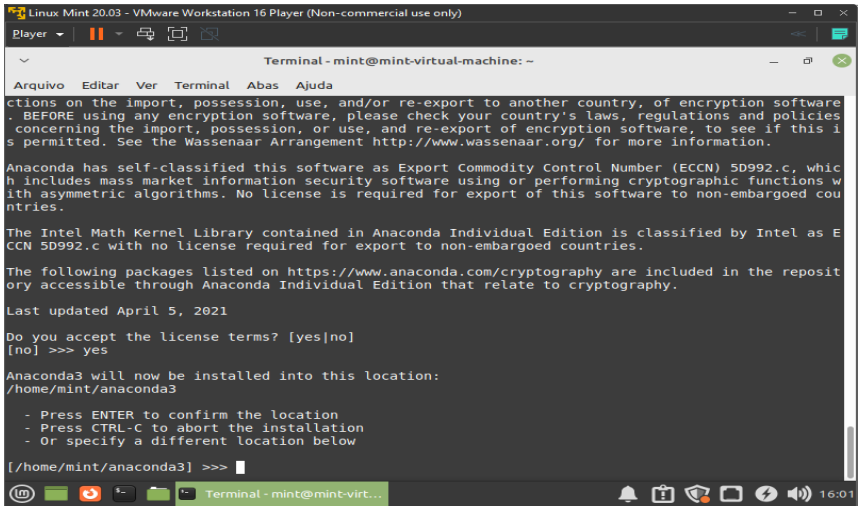
Para prosseguir, deve-se ler todos os termos de uso. A tecla ENTER pode ser utilizada para avançar a leitura.

Figura B.4 (C)



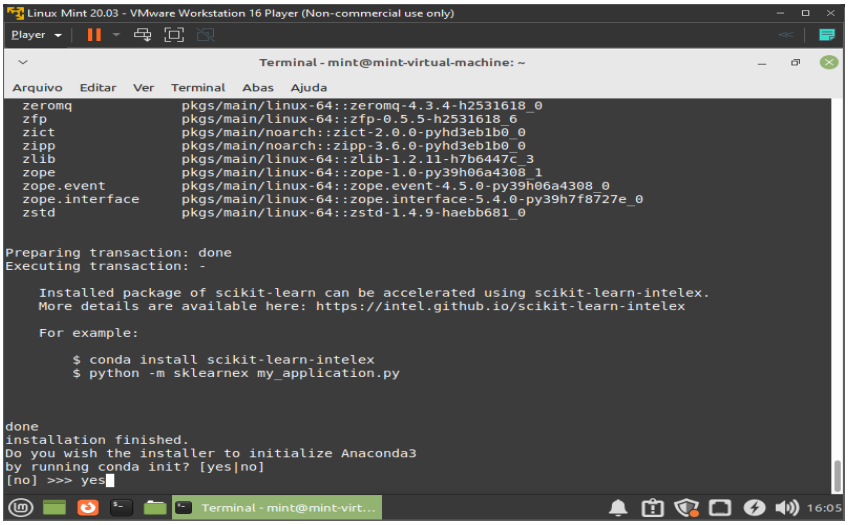
Ao digitar “yes” (seguido da tecla ENTER), o acordo de licença do usuário final é aceito.

Figura B.4 (D)



É possível alterar o caminho onde a distribuição Anaconda será instalada. Neste caso, será mantido a localização padrão e, portanto, deve-se pressionar a tecla ENTER para dar sequência.

Figura B.4 (E)



Para prosseguir, o programa necessita que o usuário digite “yes”, seguido da tecla ENTER, onde é permitido que o comando “conda init” inicialize o Anaconda 3.

Em sequência, serão instaladas as bibliotecas “OpenCV” e “Dlib”, conforme Script B.5 e Script B.6, respectivamente.

Script B.5 – Instalação da OpenCV

```
pip install opencv-python
```

Fonte: disponível em https://pypi.org/project/opencv-python/.

Script B.6 – Instalação da Dlib

```
conda install -c conda-forge dlib
```

Fonte: disponível em https://anaconda.org/conda-forge/dlib.

Para criar uma pasta reservada para o projeto, é executado o comando presente no Script B.7.

Script B.7 – Criação do ambiente de trabalho

```
mkdir ~/cnn
```

O código Python será adicionado em um arquivo Jupyter (extensão IPYNB) conforme Script B.8. O framework oferece interfaces interativas com diferentes recursos, sendo capaz de compilar diversos trechos de códigos de diferentes linguagens de programação.

Script B.8 – Utilização de um Jupyter Notebook.

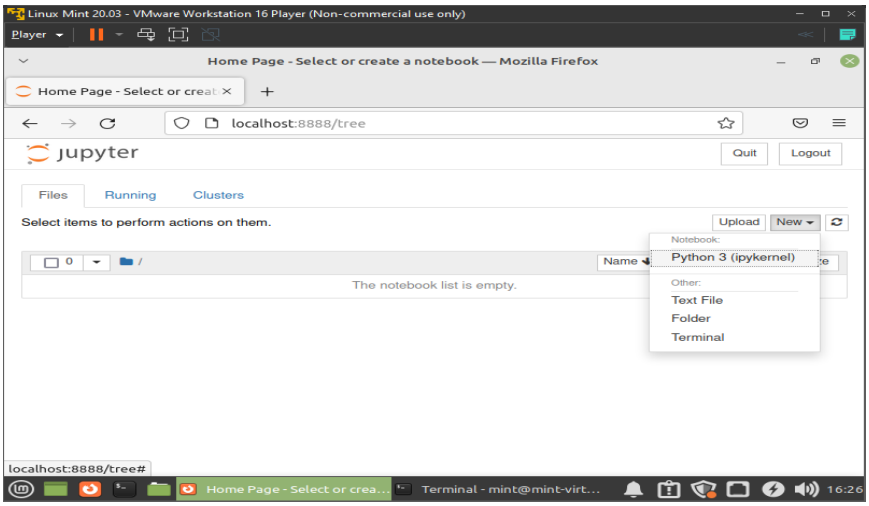
```
Cd ~/cnn && jupyter notebook
```

O comando “cd” navega até o diretório especificado.

O comando “jupyter notebook” cria um “caderno” no diretório em que o usuário se encontra.

É criado um arquivo IPYNB, conforme Figura B.5. O código Python será adicionado neste arquivo, que irá efetivamente detectar as faces nas imagens passadas como argumento.

Figura B.5 - Criação do pykernel



Para criar o arquivo, é necessário selecionar as opções destacadas na imagem.

FATEC ITU

https://fatecitu.edu.br

Aprenda a linguagem Python!

pyPRO

Acesse: