UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE LORENA

MATHEUS DE MENDONÇA CHITAN

DESENVOLVIMENTO DE UM APLICATIVO DE DETECÇÃO E RECONHECIMENTO FACIAL

Lorena
2021

MATHEUS DE MENDONÇA CHITAN

DESENVOLVIMENTO DE UM APLICATIVO DE DETECÇÃO E RECONHECIMENTO FACIAL

Trabalho de Conclusão de Curso apresentado ao Bacharelado em Engenharia Física da Universidade de São Paulo, como requisito parcial para a obtenção do título de Bacharel em Engenharia Física.

Orientador: Prof. Dr. Carlos Antônio Reis Pereira Baptista

Lorena

2021

Ficha catalográfica elaborada pelo Sistema Automatizado
da Escola de Engenharia de Lorena,
com os dados fornecidos pelo(a) autor(a)

Chitan, Matheus de Mendonça
Desenvolvimento de um aplicativo de detecção e reconhecimento facial / Matheus de Mendonça Chitan; orientador Carlos Antônio Reis Pereira Baptista. – Lorena, 2021.
94 p.

Monografia apresentada como requisito parcial para a conclusão de Graduação do Curso de Engenharia Física – Escola de Engenharia de Lorena da Universidade de São Paulo. 2021

1. Reconhecimento facial. 2. Detecção facial. 3. Local binary patterns histogram. 4. Aplicativo. 5. Python. I. Título. II. Baptista, Carlos Antônio Reis Pereira, orient.

MATHEUS DE MENDONÇA CHITAN

DESENVOLVIMENTO DE UM APLICATIVO DE DETECÇÃO E RECONHECIMENTO FACIAL

Trabalho de Conclusão de Curso apresentado ao Bacharelado em Engenharia Física da Universidade de São Paulo, como requisito parcial para a obtenção do título de Bacharel em Engenharia Física.

Aprovado em: 10 de dezembro de 2021

Banca Examinadora

_____

Dr. Carlos Antônio Reis Pereira Baptista, Escola de Engenharia de Lorena

_____

Dr. Luiz Tadeu Fernandes Eleno, Escola de Engenharia de Lorena

_____

Dr. Durval Rodrigues Junior, Escola de Engenharia de Lorena

Acima de tudo, dedico a Deus. Pois sem Ele nada disso seria possível.

Dedico também à minha mãe e ao meu pai, por todo o suporte e amor dado a mim.

AGRADECIMENTO

Acima de tudo, sou grato a Deus. Sem Ele nada seria possível.

Sou eternamente grato aos meus pais, João Carlos Chitan e Marcia Aparecida de Mendonça Chitan, pelo amor incondicional e apoio durante os meus anos de graduação. Sem eles eu não teria chego até aqui e me tornado quem sou hoje. Amo Muito Vocês.

Sou grato ao meu orientador, Carlos Antônio Reis Pereira Baptista, por aceitar a minha ideia de projeto, me auxiliar, indicando a direção correta que o projeto deveria seguir e pela paciência de me corrigir quando necessário.

Agradeço aos meus amigos, Christopher Skibbe, Pedro Gabriel Rocha Gomes, Maria Eduarda de Oliveira D'Avila Gardingo e Vitor Yang Chiba, que ao longo da graduação foram comigo, me apoiando, incentivando e ajudando a superar as dificuldades da graduação e deste projeto.

Agradeço ao corpo docente da Escola de Engenharia de Lorena – Universidade de São Paulo, por tornar esse momento realidade e pelos ensinamentos passados a mim durante os anos de graduação.

“Sirvam uns aos outros, cada um conforme o dom que recebeu, como encarregados de administrar bem a multiforme graça de Deus.” – 1 Pedro 4:10

“E tudo quanto fizerdes, fazei-o de todo coração, como para o Senhor e não para homens, conscientes de que recebereis do Senhor a recompensa da herança. É Cristo, o Senhor, que estais servindo! ” – Colossenses 3:23-24

RESUMO

Dentre as inúmeras demandas decorrentes das transformações econômico-tecnológicas do mundo atual, o desenvolvimento de novos meios segurança, que garantam maior assertividade na identificação e verificação de pessoas, tem se mostrado cada vez mais necessário para assegurar o bem-estar da sociedade. Este trabalho tem como objetivo, demonstrar, através da utilização de códigos open-source e da linguagem de programação Python, o desenvolvimento de um aplicativo de detecção e reconhecimento facial que possa ser implementado em estabelecimentos e instituições como um meio de auxílio para a segurança. São apresentados conceitos relacionados a natureza das imagens, visão computacional, como algoritmo *Haar Cascade* e *Local Binary Patterns Histogram* e o processamento de imagens. Para o desenvolvimento do aplicativo e implementação dos algoritmos, foram utilizados o framework Bootstrap e microframework Flask e as bibliotecas OpenCV e Numpy. Em condições semelhantes de luminosidade, para captura de imagens (para se tornarem imagens de referência) e o reconhecimento, foi verificado que o algoritmo pode alcançar cerca de 75% de correspondência, além de possibilitar que mais de uma pessoa possa ser verificada simultaneamente.

Palavras-chave: Reconhecimento Facial. Detecção Facial. *Local Binary Patterns Histogram*. Aplicativo. Python.

ABSTRACT

Among the several needs arising from the economic and technological changes currently experienced by the world, the development of new and more assertive security means has proven to be increasingly necessary to ensure the well-being of society. This work aims to demonstrate, through the use of open-source codes and the Python programming language, the development of a detection and facial recognition application that can be implemented in establishments and institutions as a means of aid for security. Concepts related to the nature of images, computer vision, such as Haar Cascade algorithm and Local Binary Patterns Histogram and image processing, are presented. For the development of the application and implementation of the algorithms, the Bootstrap framework and the Flask microframework and the OpenCV and Numpy libraries were used. Under similar lighting conditions, for image capture (to become reference images) and for recognition, it was found that the algorithm can achieve about 75% of correspondence, in addition to allowing more than one person to be verified simultaneously.

Keywords: Facial recognition. Face Detection. Local Binary Patterns Histogram. Application. Python.

LISTA DE FIGURAS

Figura 1 – Linha do tempo das tecnologias de detecção e reconhecimento facial. . 19
Figura 2 – Breve resumo das ferramentas, softwares, módulos Python e frameworks utilizados 29
Figura 3 – Imagem com 400 pixels 30
Figura 4 – Imagem policromática (esquerda) seguida de suas componentes monocromáticas nos canais de cor Verde, Azul e Vermelho, respectivamente. 31
Figura 5 – Imagem policromática (esquerda superior) seguida de suas componentes monocromáticas nos canais Amarelo, Magenta, Ciano (inferior esquerda) e Preto (inferior direto) 32
Figura 6 – Imagem em seu formato policromático (esquerda) e imagem em seu formato monocromático cinza (direita) 33
Figura 7 – Recursos (features) utilizados para a detecção de linhas, cantos, traços e variações de intensidade de cor. 34
Figura 8 – Representação de uma parte do método para a detecção de traços, linhas de variações de intensidade de pixel 35
Figura 9 – Recursos (*features)* cruzando a imagem 35
Figura 10 – Algoritmo da imagem Integral para a determinação linhas, cantos, traços e variações de intensidade de cor. 36
Figura 11 – Representação do algoritmo Attentional Cascade 38
Figura 12 – Representação esquemática da atribuição de valores aos pixels vizinhos 39
Figura 13 – Transformação dos valores dos pixels vizinhos em binário e transformação do binário para decimal 39
Figura 14 – Representação dos pixels finais após as transformações e a aplicação de todo o algoritmo em uma imagem real 40
Figura 15 – Representação esquemática da formação dos histogramas 41
Figura 16 – Representação do histograma final da Figura 15. 41
Figura 17 – Organização e estrutura do aplicativo 43
Figura 18 – Código do arquivo __init__.py na íntegra 44
Figura 19 – Bibliotecas utilizadas no código forms.py 45
Figura 20 – Cadastramento do perfil de administrador 46

Figura 21 – Formulário para o *login* do administrador 46
Figura 22 – Formulário para o cadastramento de pessoas no banco de dados . . . 47
Figura 23 – Formulários para o encaminhamento de e-mail e recadastramento de senhas 47
Figura 24 – Bibliotecas utilizadas no código models.py 48
Figura 25 – Determinação de qual administrador esta *logado* no aplicativo 48
Figura 26 – Tabela de perfil de administrador 49
Figura 27 – Tabelas de cadastro de pessoas e de registro de reconhecimento . . . 50
Figura 28 – Bibliotecas utilizadas no código routes.py 51
Figura 29 – Constantes para a detecção facial e reconhecimento facial. 51
Figura 30 – Código de detecção facial 52
Figura 31 – Captura de imagens para o banco de dados do usuário 52
Figura 32 – Código de reconhecimento facial 53
Figura 33 – Definição do caminho para as páginas “home.html” (Início) e “register.html” (registro) e definição da função “deletar” 54
Figura 34 – Sistema de login para os administradores 55
Figura 35 – Sistema de Logout, Reconhecimento facial, Detecção Facial e Captura de imagens para o banco de dados de usuários 55
Figura 36 – Treinamento do algoritmo LBPH 56
Figura 37 – Página dos administradores e página de registro de reconhecimento. . 56
Figura 38 – Sistema de consulta de pessoas 57
Figura 39 – Definição do número de administradores e função de envio de e-mails para o recadastramento de senhas 57
Figura 40 – Página de envio de e-mail e página de recadastramento de senha. . . 58
Figura 41 – Código do arquivo treinamento.py. 59
Figura 42 – Código base do arquivo layout.html I 60
Figura 43 – Código base do arquivo layout.html II 61
Figura 44 – Exemplificação do menu responsivo (*responsive navbar)*. 62
Figura 45 – Código base do arquivo layout.html III 63
Figura 46 – Código account.html I 64
Figura 47 – Código account.html II 65
Figura 48 – Código account.html III 65
Figura 49 – Código account.html IV 66

Figura 50 – Código account.html V 66
Figura 51 – Código tabelas.html I 67
Figura 52 – Código tabelas.html II. 67
Figura 53 – Código tabelas.html III 68
Figura 54 – Código filtro.html I 68
Figura 55 – Código filtro.html II. 69
Figura 56 – Código home.html 70
Figura 57 – Código register.html I 71
Figura 58 – Código register.html II. 71
Figura 59 – Código register.html III 72
Figura 60 – Código register.html IV 72
Figura 61 – Código login.html I 73
Figura 62 – Código login.html II. 73
Figura 63 – Código login.html III 74
Figura 64 – Código reset_request.html 75
Figura 65 – Código reset_token.html I 76
Figura 66 – Código reset_token.html II 76
Figura 67 – Atributos do código main.css I. 77
Figura 68 – Atributos do código main.css II 78
Figura 69 – Atributos do código tabelas.css I 78
Figura 70 – Atributos do código tabelas.css II. 79
Figura 71 – Atributos do código tabelas.css III 80
Figura 72 – Código run.py 81
Figura 73 – Página “home.html” sem administradores cadastrados. 82
Figura 74 – Página “home.html” com administradores cadastrados 82
Figura 75 – Página “register.html” para o cadastramento de administradores. . . . 83
Figura 76 – Página de login de administradores 83
Figura 77 – Página “account.html” que estará disponível a todos os administradores. 84
Figura 78 – Página “tabelas.html” 85
Figura 79 – Página “filtro.html” com apenas o nome “Matheus” selecionado . . . 85
Figura 80 – Página “reset_request.html” 86
Figura 81 – Página “reset_token.html” 86

Figura 82 – A esquerda é apresentado o detector facial e a direita o reconhecimento facial 87
Figura 83 – Reconhecimento facial com duas pessoas distintas 87
Figura 84 – Influência da inclinação na detecção facial com a utilização do algoritmo Haar Cascade A 88
Figura 85 – Influência da inclinação na detecção facial com a utilização do algoritmo Haar Cascade B 89
Figura 86 – Porcentagem da CPU que está sendo utilizada ao executar a função de detecção facial do aplicativo 91

LISTA DE TABELAS

Tabela 1 – Exemplo da utilização do mapeamento objeto-relacional. 26

Sumário

1. INTRODUÇÃO. 18

1.1. Objetivo do Trabalho 23

2. FUNDAMENTAÇÃO TEÓRICA 24

2.1. Ferramentas e *Softwares Open Source* 24

2.1.1. Python 24

2.1.2. Anaconda 25

2.1.3. PyCharm 25

2.1.4. Máquina Utilizada. 25

2.2. Módulos Python e Framework 26

2.2.1. Flask Microframework 26

2.2.2. Flask SQLAlchemy Framework 26

2.2.3. SQLite 27

2.2.4. Bootstrap 27

2.2.5. Numpy 28

2.2.6. OpenCV 28

2.2.7. Resumo das tecnologias utilizadas 28

2.3. Imagens 29

2.3.1. O que são imagens 29

2.3.2. O que é uma imagem digital. 29

2.3.3. Imagens digitais policromáticas. 30

2.3.3.1. Canal de Cor – RGB (*Red, Green, Blue*) . . 31

2.3.3.2. Canal de Cor – CMYK (*Cyan, Magenta, Yellow, Black*). 31

2.3.3.3. Conversão dos canais RGB para escala cinza 32

2.4. Algoritmo. 33

2.4.1. Classificador Haar Cascade 34

2.4.2. Adaboost. 37

2.4.3. *Attentional Cascade* 37

2.4.4. Reconhecimento facial LBPH (*Local Binary Patterns Histogram)*. 38

2.4.4.1. Pixels Vizinhos. 38

2.4.4.2. Pixel Central. 39

2.4.4.3. Histograma 40

2.4.4.4. Comparação de Imagens. 42

3. DESENVOLVIMENTO DO TRABALHO 43

3.1. Configurações 43

3.1.1. __init__.py 43

3.1.2. forms.py 45

3.1.3. models.py 48

3.1.4. routes.py 50

3.1.5. treinamento.py 59

3.1.6. Templates 60

3.1.6.1. layout.html 60

3.1.6.2. account.html 63

3.1.6.3. tabelas.html. 66

3.1.6.4. filtro.html. 68

3.1.6.5. home.html 69

3.1.6.6. register.html. 70

3.1.6.7. login.html 73

3.1.6.8. reset_request.html 74
3.1.6.9. reset_token.html. 75
3.1.7. Static 76
3.1.7.1. main.css 77
3.1.7.2. tabelas.css 78
3.1.8. Images 80
3.2. Dataset. 80
3.3. Trainer 80
3.4. run.py 81
3.5. Aplicativo 81
3.5.1. Páginas 81
3.5.1.1. home.html 81
3.5.1.2. register.html 82
3.5.1.3. login.html 83
3.5.1.4. account.html 84
3.5.1.5. tabelas.html e filtro.html 84
3.5.1.6. reset_request.html e reset_token.html. . . . 85
3.5.2. Detecção e Reconhecimento facial 86
4. DISCUSSÃO DE RESULTADOS. 88
4.1. Detecção Facial. 88
4.2. Reconhecimento Facial 89
4.3. Processamento 90
5. CONCLUSÃO. 92
REFERÊNCIAS 93

1. INTRODUÇÃO

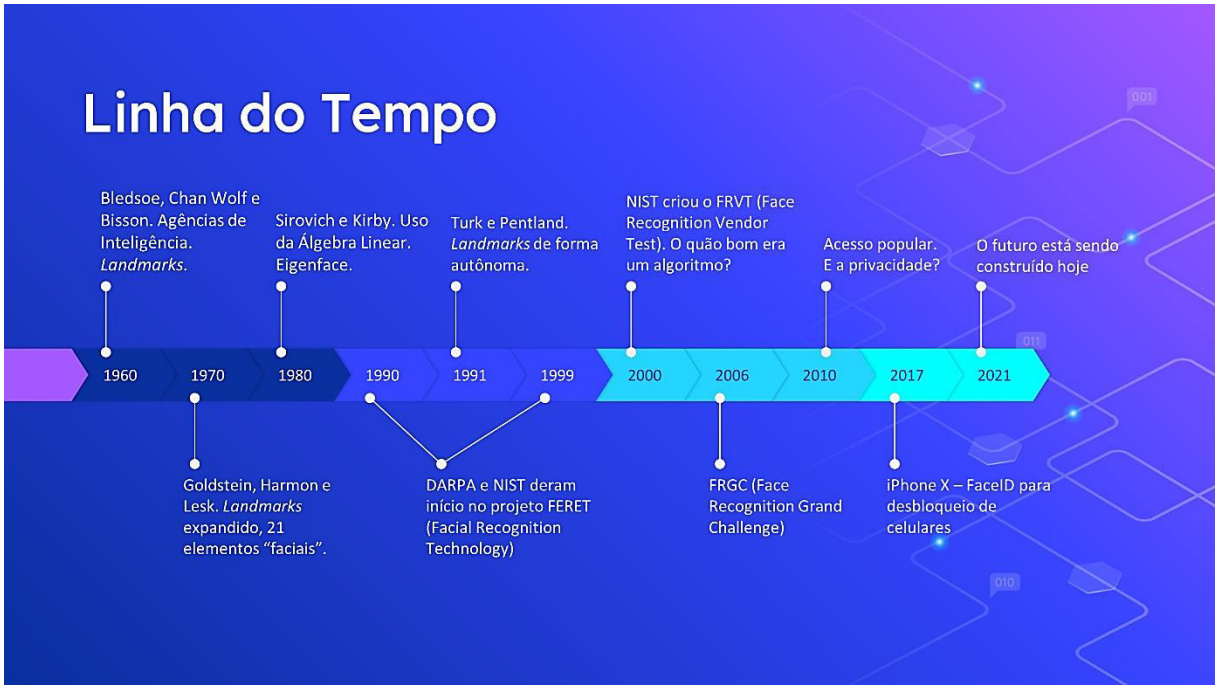
Com o crescente avanço da tecnologia e da dependência que o homem apresenta dessa, novos meios segurança que garantam maiores assertividades na identificação e verificação de pessoas, quando comparados com antigos métodos, como simples senhas, têm se mostrados cada vez mais necessários para o assegurar o bem-estar da sociedade como um todo.

Ao longo dos anos diversos métodos foram propostos, apesar de apenas alguns terem se mostrado efetivos e viáveis, como os "Sistemas Biométricos". Tais sistemas fazem uso de características específicas de cada indivíduo, como digitais, íris e palma da mão (conhecidas como fisiologia estática), e "características comportamentais", como padrão de pegada, reconhecimento de voz e reconhecimento facial, para verificar a real identidade do indivíduo em questão. Claramente, assim como qualquer outro, tais sistemas não são 100% seguros, mas são muitos mais confiáveis do que aqueles utilizados anos atrás, que já se tornaram obsoletos com o avanço da tecnologia. (TASKIRAN; KAHRAMAN; ERDEM, 2020)

Entretanto, dentre todos estes mencionados acima, o que se torna de maior valia é o reconhecimento facial, uma vez que a face humana carrega diversas informações sobre o indivíduo, como idade, gênero, etnia, emoções e estado metal, e não necessita da cooperação da pessoa para que seja realizado, em contraste dos demais métodos, tornando este método excelente para aplicações de segurança. (TASKIRAN; KAHRAMAN; ERDEM, 2020)

O desenvolvimento do primeiro sistema de reconhecimento facial, como verificado na Figura 1, que apresenta uma linha do tempo das tecnologias de detecção e reconhecimento facial, pode ser atribuído a três protagonistas, em meados dos anos 60: Woody Bledsoe, Helen Chan Wolf e Charles Bisson. A princípio, pelo fato do projeto ser financiado por agências de inteligência não divulgadas, grande parte do projeto só veio à tona em 2011, quando ficou sabido que o sistema se baseava no cálculo das distâncias entre "*landmarks*", as quais eram pontos atribuídos a determinados elementos do rosto, como olhos, nariz, sobrancelhas e muitos outros. Tal sistema, como se pode imaginar, foi muito prejudicado pela tecnologia da época, apesar de ainda ter sido um grande passo para provar a viabilidade do projeto. (NEC, 2020)

Figura 1: Linha do tempo das tecnologias de detecção e reconhecimento facial



Fonte: Autoria Própria

Já na década de 70, o projeto teve continuidade nas mãos de outro trio: Goldstein, Harmon e Lesk. Em seu projeto, o grupo focou em estender o conceito das "*landmarks*" para 21 elementos faciais, os quais, dessa vez, incluíam desde os olhos, nariz e sobrancelhas até cor de cabelo e grossura dos lábios. (SINFIC, 2008)

Entretanto, foi só na década de 80 que, realmente, houve um grande desenvolvimento no campo do reconhecimento facial, quando Sirovich e Kirby implementaram o uso da álgebra linear em seus algoritmos. Mais especificamente, a álgebra linear permitiu que dentro um conjunto de imagens, fossem escolhidos quais os principais traços que definiriam melhor uma face humana. A esse algoritmo foi dado o nome de Eigenface e foi um dos primeiros algoritmos de detecção facial desenvolvido. (NEC, 2020)

Em 1991, uma segunda dupla de cientistas, Turk e Pentland, descobriram como implementar um algoritmo de reconhecimento facial capaz, de forma autônoma, atribuir as "*landamrks*" aos seus respectivos elementos da face, uma vez que até então era necessário marcar manualmente as "*landmarks*" em cada uma das faces que se desejava reconhecer. Tal algoritmo possibilitou uma revolução dentro do ramo, permitindo pela primeira vez que todo o processo fosse realizado sem o auxílio humano. (SINFIC, 2008)

Entre a década de 90 e o ano 2000, a DARPA (The Defense Advanced Research Projecto Agency) em conjunto com o NIST (National Institute of Standards and Technology) dos

Estados Unidos, deram início ao projeto FERET (Face Recognition Technology), como forma de incentivar o uso do reconhecimento facial para segurança, inteligência e para aplicação da lei. O projeto contava que a criação de um banco de dados com cerca de 2500 imagens de faces humanas de mais de 800 pessoas. (NIST, 2020)

Já nos anos 2000, através do NIST foi criado o FRVT (Face Recognition Vendor Test), o qual consistia em um teste que avaliava o quão bom era um algoritmo de reconhecimento facial para aplicações civis, legais e para segurança. Os testes tinham como critérios a velocidade, precisão, armazenamento e memória dos algoritmos e são referência dentro do ramo até os dias de hoje. (IDEMIA, 2021)

Em 2006, mais uma vez, com o objetivo de promover um avanço na tecnologia do reconhecimento facial, principalmente para aplicações relacionadas ao governo americano, foi criado o FRGC (Face Recognition Grand Challenge). De forma similar ao FRVT, o FRGC avaliava os algoritmos de reconhecimento facial mais recentes disponíveis na época. (NIST, 2020)

Já em 2010, com o rápido avanço da tecnologia e o aumento dos processamentos dos computadores, o reconhecimento facial passou a ser utilizado de maneiras menos extraordinárias, estando ao alcance das pessoas. Um exemplo disso foi a implementação do reconhecimento facial na rede social Facebook, com a finalidade de identificar pessoas nas fotos que os usuários postavam. A princípio foi algo que trouxe à tona uma grande discussão sobre privacidade, entretanto, não causou um grande impacto na rede social e ela continuou crescendo e sendo utilizada mundialmente, como foi visto ao longo dos anos. (BBC, 2020)

E a partir de 2010, o reconhecimento facial apresentou um rápido avanço e se desenvolveu de forma muito eficiente. Porém, só em 2017 que apresentou mais um marco em sua história, quando os celulares iPhone X, da empresa Apple, com a funcionalidade “Face ID”, implementaram a possibilidade de desbloqueio através do reconhecimento facial. Desde então, todos os celulares da empresa, que sucederam o iPhone X, possuem tal funcionalidade. (SYMANOVICH, 2021)

Disso, é possível verificar que o reconhecimento facial não é algo novo. Desde da década de 60 vem provando a sua viabilidade através de aplicações nos mais variados locais, como: desbloqueio de celulares, campeonatos de futebol, como lembrado por Chokshi (2019), quando em 2001 em Tampa, na Flórida, 19 pessoas vítimas de mandados pendentes foram

identificadas durante a realização do 35° Super Bowl e na identificação de terrorista, como no caso do Osama Bin Laden pelo governo americano em sua captura.

Entretanto, muitas preocupações ainda são levantadas com relação ao tema e vêm dificultando a implementação em larga escala. Segundo Harding (2019), da revista Security, algumas das preocupações com relação a implementação do reconhecimento facial são:

- A criação de um *Big Brother State* – a criação de um estado controlador que está em constante vigilância da sociedade (falta de privacidade)
- Algoritmos que apresentem vieses sociais, principalmente com mulheres e grupos minoritários dentro da sociedade
- Desconfiança com relação a segurança do armazenamento dos dados
- E falta de explicações adequadas e honestas sobre onde e como a tecnologia será empregada

Tais pontos, embora sejam válidos e de real importância, não devem ser unicamente utilizados para validar a implementação ou não dos sistemas de reconhecimento. O reconhecimento facial, como mostrado anteriormente, também traz consigo diversas vantagens, como as apresentadas por Marr (2019) para a Forbes:

- Utilização para encontrar pessoas desaparecidas, como crianças e idosos, ou para encontrar pessoas suspeitas como no caso do campeonato de futebol americano em 2001.
- Implementados em aeroportos a fim de verificar a identidade dos passageiros, automatizando processos (check in e entradas nas aeronaves), aumentando a segurança da entrada de pessoas no país e diminuindo o contato pessoa-pessoa, como desejado por muitos em uma situação pós pandemia, como constatado por Roll (2019)
- Implementação em lojas, a fim de substituir cartões ou dinheiro
- Utilizado para a prevenção de fraudes

Com isso, é possível verificar que o reconhecimento facial, como qualquer outro sistema, apresenta seus prós e contras, ficando à sociedade determinar se as vantagens e facilidades que o reconhecimento facial traz consigo superam, ou não, as suas desvantagens.

Cabe apenas ressaltar que este é um ramo extremamente versátil e que está em constante desenvolvimento e que nos próximos anos, com o rápido avanço da tecnologia, aumento das

velocidades de processamento e grande tendência em sistema baseados em IA (Inteligência Artificial), algoritmos mais refinados e bem treinados, tendem a surgir.

1.1. Objetivo do Trabalho

O seguinte projeto tem como objetivo o desenvolvimento de um aplicativo de detecção e reconhecimento facial baseado na linguagem de programação Python, mediante a implementação da biblioteca OpenCV e dos algoritmos de detecção Haar Cascade e de reconhecimento LBPH (Local Binary Patterns Histogram).

2. FUNDAMENTAÇÃO TEÓRICA

Nesta seção do projeto, serão apresentadas as ferramentas, softwares, frameworks e microframework, ambientes de desenvolvimento, bibliotecas/módulos Python e os algoritmos utilizados para o desenvolvimento do aplicativo de reconhecimento facial em si.

2.1. Ferramentas e *Softwares Open Source*

Todas as ferramentas e softwares utilizados neste projeto possuem licença de código aberto - open source, do inglês. Segundo a Open Source Initiative, ferramentas e softwares de código aberto podem ser acessados, usados, modificados (ou não, se desejado) e distribuídos por pessoa, sem nenhuma restrição, apenas tendo como objetivos a colaboração e desenvolvimento da ferramenta ou software. (Open Source Initiative, [entre 1995 e 2005])

2.1.1. Python

Python, da definição, é uma linguagem de programação de alto nível, interpretada e orientada a objeto, desenvolvida por Guido van Rossum no ano de 1991 (Python Software Foundation, [2001]). Segundo Guido, a linguagem Python tem como objetivos:

- Ser fácil e apresentar uma linguagem intuitiva de programação;
- Ser tão eficiente quanto seus maiores competidores, como C, C++, Java, entre outros;
- Ser de código aberto, possibilitando que qualquer um possa contribuir para o desenvolvimento da linguagem;
- Ser de fácil entendimento, assim como uma linguagem falada (no caso em específico o inglês);
- Possibilitar um rápido desenvolvimento, sendo capaz de realizar tarefas cotidianas (COMPUTER HISTORY MUSEUM, 2018).

Segundo o índice PYPL (2021) (PopularitY of Programming Language), a qual é obtido através da análise de frequência com que tutorias da linguagem são procurados no Google, a linguagem de programação mais popular, atualmente, é o Python, com um total de 29,93% das pesquisas. Tal valor não é surpreendente, uma vez que a linguagem Python, por apresentar uma elevada produtividade, tem chamado a atenção de desenvolvedores de longa data e de amadores que desejam se inserir no mundo da programação.

Dentre as principais características que tornam a linguagem de programação Python tão popular, estão elencadas:

- Multiplataforma - A linguagem Python é executável em qualquer sistema operacional, seja Windows, Linux / UNIX e Macintosh, sendo necessário apenas instalar o seu interpretador;
- Open Source - Seu código é aberto e editável, fazendo com que os próprios usuários possam disponibilizar atualizações e ajudar o desenvolvimento da linguagem;
- Grande quantidade de bibliotecas e Frameworks - Os usuários podem utilizar ambientes de desenvolvimento e funções pré-definidas dentro de seus códigos, a fim de agilizar o processo de escrita e dar uma maior atenção ao problema a ser resolvido;
- Linguagem Dinâmica e Tipada - Não há necessidade em declarar qual o tipo das variáveis que estão sendo utilizadas

Um fato curioso é que o nome Python, apesar de remeter a espécie de cobra Píton (Python, no inglês), teve sua origem a partir do título do seriado de TV "Monty Python's Flying Circus".

2.1.2. Anaconda

Anaconda é uma distribuição aberta da linguagem de programação Python, que concentra uma grande variedade de módulos e pacotes, organizados de uma maneira fácil de instalar e compatível com as diferentes plataformas, Windows, Linux / UNIX e Macintosh. A distribuição conta com a presença de diferentes ambientes e editores, como PyCharm, Jupyter e Spyder, e também conta com a presença de diferentes bibliotecas relacionadas ao Aprendizado de Máquina (Machine Learining do Inglês), como TensorFlow, Scikit-Learn e PyTorch. (Anaconda, 2021)

2.1.3. Pycharm

O ADI - Ambiente de Desenvolvimento Integrado - PyCharm (IDE - Integrated Development Environment, do inglês) é um dos editores de texto fornecidos pela distribuição Anaconda. O PyCharm conta com diferentes ferramentas internas, que possibilitam que o desenvolvedor/programador mantenha o foco nos maiores problemas, enquanto o ADI monitora o código, a fim de auxiliar, através de preenchimentos automáticos e verificação dinâmica de erros, possíveis inconsistências no código. (JET BRAINS, [201-])

2.1.4. Máquina Utilizada

A máquina utilizada para hospedar os bancos de dado, executar os algoritmos de detecção e reconhecimento e possibilitar a execução do aplicativo é um Lenovo ideadpad 710S

Plus-13IKB Signature Edition. A máquina apresenta um processador Intel(R) Core™ i7-7500U CPU @2.70GHZ 2.90 GHz com 16,0 GB de RAM.

2.2. Módulos Python e Frameworks

2.2.1. Flask Microframework

O Flask microframework, assim com o próprio nome define, é um microframework, o qual pode ser caracterizado como um conjunto predefinido de soluções que buscam facilitar a resolução de problemas recorrentes no desenvolvimento de aplicações. Especificamente, o Flask é utilizado para o desenvolvimento de aplicações web, devido a sua simplicidade, quando comparado a Frameworks como Django e Web2py.

Pelo fato de ser um microframework, o Flask não possui, por padrão, conexões com banco de dados, sistemas de validação ou qualquer outro tipo de funcionalidade mais robusta. Dessa forma, a fim de utilizar o Flask de maneira completa, com sistemas de login, cadastramento e histórico de dados, é necessário fazer uso de módulos externos ao programa base. (Pallets, 2010)

2.2.2. Flask-SQLAlchemy Framework

Structured Query Language - SQL - é a linguagem padrão para acesso e manipulação de banco de dados. Através desta linguagem é possível inserir, deletar, alterar, procurar e filtrar dados e tabelas dentro de um determinado banco de dados. Entretanto, tal linguagem apresenta sua própria sintaxe, o que dificulta a implementação desta dentro de uma aplicação na linguagem Python. (W3SCHOOL, [200-]) (FLASK-SQLAlchemy, 2010)

Assim, é necessário a utilização do Flask-SQLAlchemy, o qual é um framework de mapeamento objeto-relacional SQL, de código aberto sob a licença do MIT - Massachusetts Institute of Technology. Tal framework, através do mapeamento objeto-relacional, é responsável pela conversão de códigos da linguagem Python em linguagem SQL, a fim de que não seja necessário a utilização de códigos SQL (SQLALchemy, [20--]). Um exemplo pode ser verificado na tabela 1.

Tabela 1 – Exemplo da utilização do mapeamento objeto-relacional

Ação	Código
Pesquisa sem a utilização do mapeamento objetor-relacional	SELECT * FROM users WHERE EMAIL = 'test@test.com'

Pesquisa com a utilização do mapeamento objeto-relacional	users.query.filter_by(email='test@test.com').first()

Fonte: Modificado de (Flask-SQLAchemy, 2010)

2.2.3. SQLite

SQLite é um módulo Python em desenvolvimento, implementado na linguagem de programação C, que apresenta um mecanismo de banco de dados SQL transacional, independente, sem servidor e com zero configurações. (SQLite Consortium, [entre 1995 e 2005])

Um banco de dados transacional, é composto por quatro propriedades: Atomicidade, Consistência, Isolamento e Durabilidade. Tais propriedades garantem a validação correta dos dados que entram e que saem do banco de dados, independente de quaisquer tipos de erros que possam ocorrer durante a inserção ou remoção. Em palavras mais simples, todas as ações relacionadas ao banco de dados são consideradas um sucesso total, sendo posteriormente efetuadas, ou um fracasso total, sendo posteriormente descartadas, sem haver transações intermediárias. (SQLite Consortium, [entre 1995 e 2005].)

O SQLite é considerado um banco de dados independente, devido as suas baixíssimas dependências externas. Além disso, devido ao fato de não apresentar servidores, o banco é gerado dentro do próprio sistema operacional da máquina, não tendo conexões com a internet ou qualquer meio de comunicação.

Por fim, por apresentar zero configurações, o banco de dados SQLite não precisa ser "instalado" antes de ser usado. Não há processos que necessitem ser configurados, iniciados ou parados. Além de não haver a necessidade de administradores que permitam o acesso de usuários.

2.2.4. Bootstrap

Bootstrap é um framework aberto, que faz uso de JavaScript, HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), Less (Leaner Style Sheets) e Sass (Syntactically Awesome Style Sheets) para a elaboração de interfaces e front-ends para sites e aplicações web. Dentre as 4 linguagens apresentadas, o JavaScript é a única considerada linguagem de programação, sendo as demais consideradas linguagens de marcação, não interferindo no funcionamento back-end das aplicações. Cabe ressaltar, que as linguagens Less e Sass são utilizadas para acrescer a linguagem CSS de funcionalidades e estilos. (Bootstrap, [201-])

2.2.5. Numpy

Numpy é um módulo Python desenvolvido para a realização de operações matemáticas baseadas em arranjos, vetores e matrizes, dos mais variados tipos e tamanhos. A grande vantagem da utilização do módulo é a sua velocidade de processamento, quando comparada com as funções nativas do Python. Tal velocidade se deve a sua implementação na linguagem de programação C (Numpy, 2021). Dois exemplos da utilização do módulo Numpy são:

- Através da utilização dos módulos SciPy, Matplotlib e Numpy, foi possível que o Event Horizon Telescope produzisse a primeira imagem de um buraco negro;
- O módulo Numpy, juntamente com os módulos Pandas, Scikit-learn e Scipy, foram utilizados para a análise e confirmação da existência de ondas gravitacionais, captadas pelo LIGO (Laser Interferometer Gravitational-Wave Observatory).

2.2.6. OpenCV

O OpenCV, traduzido para o português como Biblioteca de Código Aberto para Visão Computacional, é um módulo Python multiplataforma, inicialmente desenvolvido pela Intel no ano 2000, para proporcionar uma infraestrutura comum para aplicações de Visão Computacional, acadêmicas ou comerciais. O módulo conta com mais de 2500 algoritmos otimizados que podem ser utilizados para as mais diversas aplicações, como: tratamento de imagens, detecção facial, reconhecimento facial, aprendizado de máquina e muitos outros. (About OpenCV, [200-])

2.2.7. Resumo das tecnologias utilizadas

Afim de resumir e facilitar o entendimento do que foi apresentado nas seções 2.1. e 2.2., a Figura 2 traz uma breve contextualização de cada uma das ferramentas, softwares, módulos Python e frameworks utilizados no desenvolvimento do aplicativo.

Figura 2: Breve resumo das ferramentas, softwares, módulos Python e frameworks utilizados

Fonte: Autoria Própria

2.3. Imagens

Nesta seção serão discutidos conceitos relacionados a formação de imagens monocromáticas e policromáticas, canais de cores e o seu papel dentro da formação destas e a conversão da escala RGB para a escala monocromática cinza.

2.3.1. O que são Imagens

Uma imagem pancromática é uma função de duas variáveis baseada na intensidade de luz. Normalmente, a função é denotada por $f(M,N)$, onde M e N são as coordenadas espaciais e f, na coordenada (M,N), é proporcional ao brilho do objeto que se deseja capturar naquele ponto. Caso a imagem seja multiespectral, $f(M,N)$ será denotado por um vetor, no qual cada componente indicará a intensidade luminosa do objeto, no ponto (M,N), para o espectro correspondente. (PETROU; PETROU, 2010)

2.3.2. O que é uma imagem digital

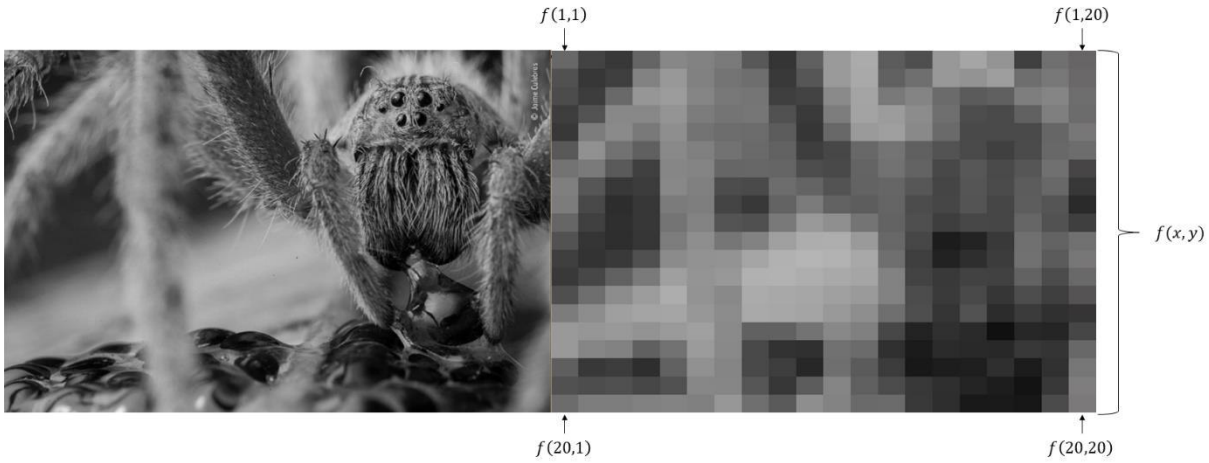
Uma imagem digital é uma imagem, $f(M,N)$, que foi discretizada tanto nas coordenadas espaciais quanto na intensidade luminosa. As coordenadas espaciais passam a ser representados apenas por números inteiros, como 1, 2, 3, e a intensidade luminosa entre limites de mínimo e máximo, como de 0 a 255 ou de 0 a 100. (PETROU; PETROU, 2010)

Uma imagem digital monocromática se assemelha ao que é mostrada na equação (1), ou seja, uma matriz de dimensões $M \times N$. Em termos gerais, a imagem digital é formada por $M \times N$ pixels (*picture element*), onde cada pixel corresponde a um elemento da matriz.

$$f(M,N) = \begin{pmatrix} f(1,1) & \cdots & f(1,N) \\ \vdots & \ddots & \vdots \\ f(M,1) & \cdots & f(M,N) \end{pmatrix} \quad (1)$$

Na Figura 3, à esquerda, pode ser observada uma imagem policromática, utilizada apenas como base de referência para comparação e entendimento e, à direita, uma imagem policromática, com 400 pixels (20×20) e com os elementos demonstrados da equação (1).

Figura 3: Imagem com 400 pixels



Fonte: Modificado de REVISTA GALILEU (2020)

2.3.3. Imagens digitais policromáticas

Da mesma forma que para a imagem monocromática, uma imagem digital policromática será representada por uma imagem $f(M,N)$, onde as suas variáveis espaciais e de intensidade luminosa serão discretizadas.

A grande diferença entre as imagens digitais monocromáticas e a policromáticas, como o próprio nome já sugere, reside na quantidade de espectros luminosos, também conhecidos como canais de cores, que formarão a imagem. Mais especificamente, para cada espectro de luz que se deseja trabalhar, haverá uma matriz $M \times N$, com as coordenadas espaciais e intensidades luminosas do respectivo espectro (PETROU; PETROU, 2010). Dessa forma podemos escrever, matematicamente, que a imagem final será dada de forma similar a equação (2), onde os elementos "A" e "Z" correspondem a diferentes canais de cor.

$$f(M,N) = \begin{pmatrix} (A,Z)_{1,1} & \cdots & (A,Z)_{1,N} \\ \vdots & \ddots & \vdots \\ (A,Z)_{M,1} & \cdots & (A,Z)_{M,N} \end{pmatrix} \quad (2)$$

2.3.3.1. Canal de Cor – RGB (*Red*, *Green*, *Blue*)

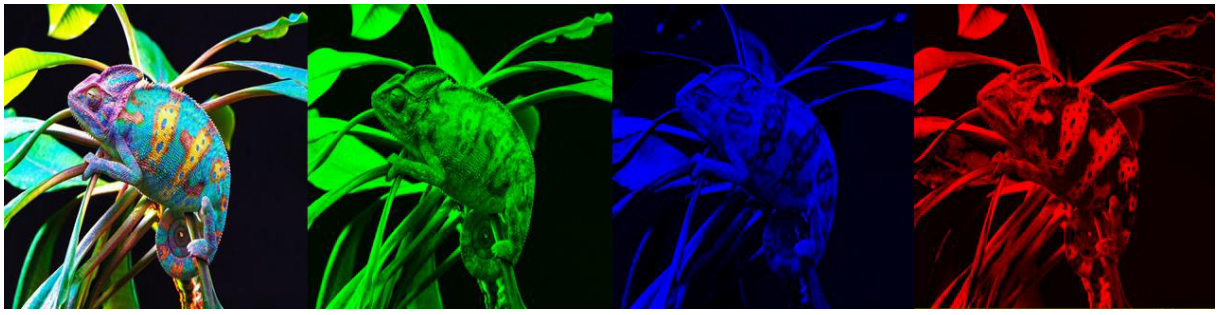
O sistema RGB de cores, o qual será utilizado no projeto em questão, faz uso de 3 canais como espectros primários para a formação das imagens. Neste sistema, os pixels que compõe as imagens apresentam componentes nas cores Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*), as quais possuem 256 níveis de intensidade, cada. Dessa forma, ao combinar diferentes intensidades em diferentes pontos da tela ou monitor, é possível que uma imagem seja formada. (ACDSystem, [2003?])

Caso fosse desejado, a imagem, $f(M,N)$ poderia também ser representada da forma matricial. Cada canal de cor apresentaria uma matriz de dimensões $M \times N$ que quando sobrepostas resultariam na matriz da equação (3).

$$f(M,N) = \begin{pmatrix} (R,G,B)_{1,1} & \cdots & (R,G,B)_{1,N} \\ \vdots & \ddots & \vdots \\ (R,G,B)_{M,1} & \cdots & (R,G,B)_{M,N} \end{pmatrix} \quad (3)$$

Na Figura 4 é possível verificar uma imagem na sua forma policromática, apenas com o canal de cor verde, apenas com canal de cor azul e apenas com canal de cor vermelho.

Figura 4: Imagem policromática (esquerda) seguida de suas componentes monocromáticas nos canais de cor Verde, Azul e Vermelho, respectivamente



Fonte: Modificado de GAY (2020)

2.3.3.2. Canal de Cor – CMYK (*Cyan, Magenta, Yellow and Black)*

Diferentemente do sistema de cores RGB, o sistema CMYK trabalha com 4 canais como espectros primários para a formação da imagem, sendo esses as cores Ciano, Magenta, Amarelo e Preto, as quais possuem apenas 100 níveis de intensidade, cada. (ACDSystem, [2003?])

Entretanto, a forma matricial, dada pela equação (4) para imagem $f(M,N)$, no sistema CMYK é semelhante à forma matricial da imagem no sistema RBG.

$$f(M,N) = \begin{pmatrix} (C,M,Y,K)_{1,1} & \cdots & (C,M,Y,K)_{1,N} \\ \vdots & \ddots & \vdots \\ (C,M,Y,K)_{M,1} & \cdots & (C,M,Y,K)_{M,N} \end{pmatrix} \quad (4)$$

Na Figura 5 é possível verificar uma figura em sua forma policromática e respectivamente com os canais de cores Amarelo, Magenta, Ciano e Preto.

Figura 5: Imagem policromática (esquerda superior) seguida de suas componentes monocromáticas nos canais Amarelo, Magenta, Ciano (inferior esquerda) e Preto (inferior direta)

Fonte: Modificado de GAY (2020)

2.3.3.3. Conversão dos canais RGB para escala cinza

Presentes nas mais diversas áreas do conhecimento, as cores têm sido utilizadas há anos para nos expressar e entender o mundo em que vivemos. Entretanto, quando algoritmos computacionais são requeridos, é desejável que os múltiplos canais de cores sejam substituídos por um único cinza.

As duas principais razões pela escolha de um único canal cinza, para este projeto, foram: velocidade de processamento e algoritmo utilizado.

- Velocidade de processamento - Se supormos que seja necessário um tempo t para o processamento de qualquer tipo de canal, será necessário um tempo $n \times t$ para uma imagem de n canais. Porém, se estivermos trabalhando com um banco de dados com

milhões de imagens, o tempo necessário será $m \times n \times t$, onde m é o número de imagens. Assim, fica claro, que o tempo de processamento será n vezes mais rápido para um único canal cinza, quando comparado com imagens que apresentem diferentes canais de cores.

- Algoritmo utilizado - O algoritmo utilizado para o treinamento das imagens, necessita que estas apresentem um único canal (cinza), a fim de facilitar as operações matemáticas que serão detalhadas nas seções seguintes.

O algoritmo matemático utilizado para a transformação das imagens policromáticas, do sistema RGB para a escala cinza pode ser verificado na equação (5), onde Y representa a cor do pixel na escala cinza e R, G e B as componentes Vermelho, Verde e Azul, respectivamente, do pixel antes da transformação. (OpenCV, [201-])

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (5)$$

Na Figura 6, à esquerda, é possível verificar uma imagem policromática no sistema RGB e, à direita, uma imagem na escala cinza, após passar pelo algoritmo demonstrado pela equação 5.

Figura 6: Imagem em seu formato policromático (esquerda) e imagem em seu formato monocromático cinza (direita)

Fonte: Modificado de GAY (2020)

2.4. Algoritmos

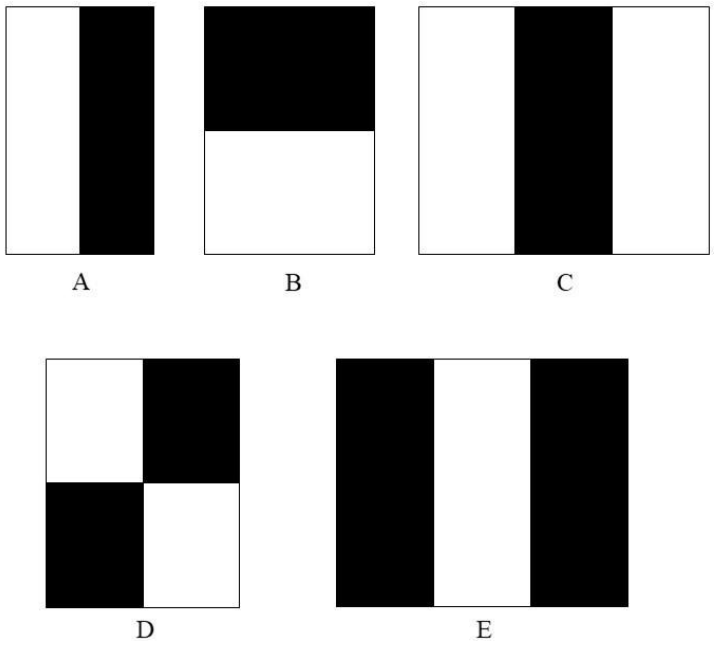
Nesta seção serão discutidos alguns dos algoritmos utilizados para a detecção e reconhecimento facial que serão, posteriormente, implementados na forma de códigos na linguagem Python, para o desenvolvimento de um aplicativo.

2.4.1. Classificador Haar Cascade

Um classificador Haar Cascade é um algoritmo de detecção de objetos, comumente utilizado para a detecção de traços faciais, em imagens ou vídeos em tempo real. O algoritmo foi proposto por Viola e Jones em seu trabalho *"Rapid Object Detection using a Boosted Cascade of Simple Features"*, publicado na Conferência de Visão Computacional e Reconhecimento de Padrões em 2001.

O método consiste na utilização de recursos, do inglês *features*, os quais são mostrados na Figura 7 para a detecção de linhas, cantos, traços e variações da intensidade de cor, que possam representar o objeto em questão. (VIOLA; JONES, 2001)

Figura 7: Recursos (features) utilizados para a detecção de linhas, cantos, traços e variações de intensidade de cor.



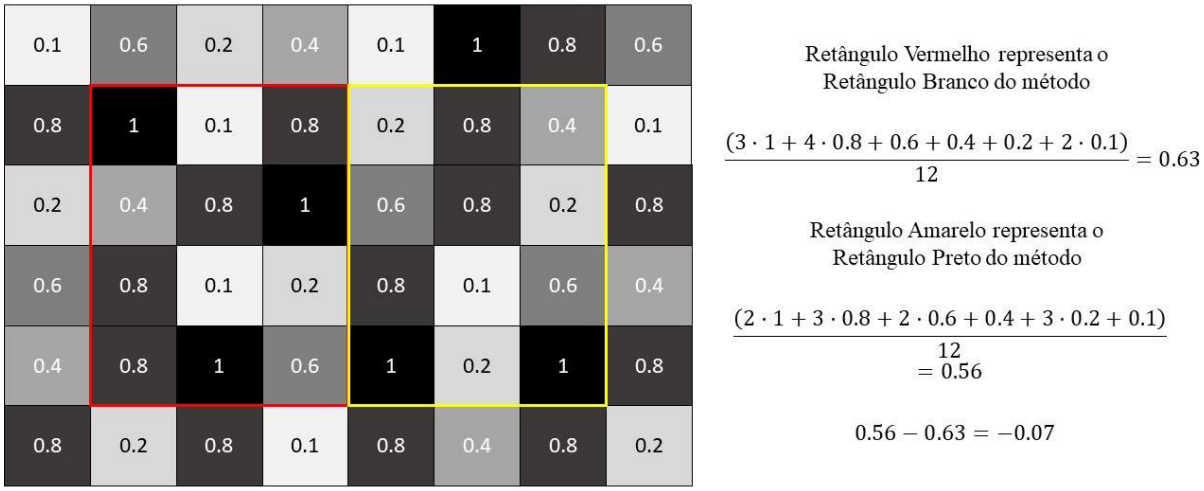
Fonte: Autoria Própria

Cada tipo de recurso é responsável por verificar a existência de uma característica na imagem. Da Figura 7, o recurso "A" é responsável por verificar a presença de linhas e traços na vertical, o recurso "B" por verificar na horizontal, o "C" e o "E" por regiões que estejam separados por regiões mais escuros ou claras, respectivamente, e o "D" por traços na diagonal.

Tais recursos, ao passarem pela imagem, realizam as somas das intensidades dos pixels contidos em cada um dos retângulos. Posteriormente, a soma dos brancos é subtraída da soma dos pretos e, só então, é verificado se o valor resultante é próximo ou não de 1 (valor arbitrário

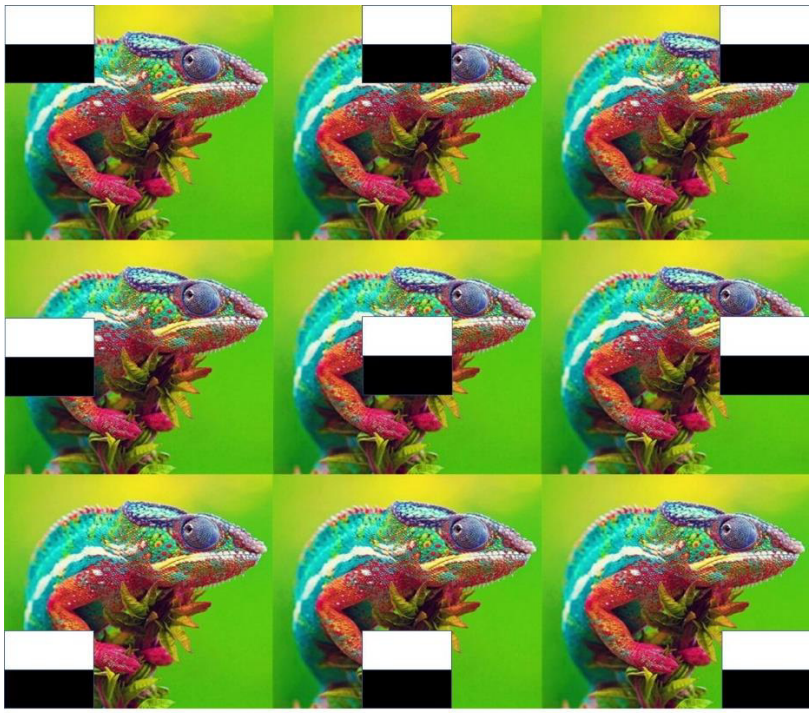
utilizado para o entendimento do método). Caso o valor seja próximo de 1, há um indicativo que há uma grande diferença entre os pixels dos dois retângulos, o que significa que há um traço separando os dois. Abaixo, na Figura 8 há uma representação esquemática desta parte do método e na Figura 9 uma representação de como os recursos cruzam a imagem. (VIOLA; JONES, 2001)

Figura 8: Representação de uma parte do método para a detecção de traços, linhas de variações de intensidade de pixel.



Fonte: Autoria Própria

Figura 9: Recursos (*features)* cruzando a imagem



Fonte: Modificado de POPUP PAINTING (2018)

Entretanto, cruzar uma imagem com a utilização dos recursos é computacionalmente custoso, uma vez que é necessário iterar por cada pixel mais de uma vez. Dessa forma, foi implementado um algoritmo chamado de imagem integral, onde o valor de cada pixel da imagem integral será a soma dos valores dos pixels posicionados acima e à esquerda da imagem original (VIOLA; JONES, 2001). Abaixo, na equação (6), é possível verificar o conceito da imagem integral de forma matemática.

$$f'(i',j') = \begin{cases} f(1,1), & i',j' = 1,1 \\ \sum_{i=1}^{i'} \sum_{j=1}^{j'} f(i,j), & i',j' \neq 1,1 \end{cases} \quad (6)$$

Onde $f(i,j)$ representa o pixel da imagem original, $f'(i',j')$ representa o pixel da imagem integral, i e j a posição do pixel na imagem original e i' e j' a posição do pixel na imagem integral.

Tal algoritmo possibilita que a quantidade de operações matemáticas seja reduzida, uma vez que, ao formar a imagem integral, os recursos que cruzaram a imagem não necessitarão realizar a soma de n valores de intensidade de pixels, como mostrados na Figura 8, mas apenas a soma de 4 valores, os quais estão posicionados nas extremidades dos recursos. Na Figura 10 é possível verificar a soma sendo realizada na imagem integral. (VIOLA; JONES, 2001)

Figura 10: Algoritmo da imagem Integral para a determinação linhas, cantos, traços e variações de intensidade de cor.

0.1	0.7	0.9	1.3	1.4	2.4	3.2	3.8
0.9	2.5	2.8	4	4.3	6.1	7.3	8
1.1	3.1	4.2	6.4	7.3	9.9	11.3	12.8
1.7	4.5	5.7	8.1	9.8	12.5	14.5	16.4
2.1	5.7	7.9	10.9	13.6	16.5	19.5	22.2
2.9	6.7	9.7	12.8	16.3	19.6	23.4	26.3

Imagem Integral

Retângulo Vermelho representando o Retângulo Branco do método

$$\frac{(10.9 - 1.3 - 2.1 + 0.1)}{12} = 0.63$$

Retângulo Amarelo representando o Retângulo Preto do método

$$\frac{(19.5 - 3.2 - 10.9 + 1.3)}{12} = 0.56$$

Subtração:

$$0.56 - 0.63 = -0.07$$

Fonte: Autoria Própria

2.4.2. Adaboost

É de grande importância ressaltar que tal método, para um funcionamento adequado, necessita de imagens positivas, que apresentem o que se deseja encontrar, e imagens negativas, que não apresentem o que se deseja encontrar. Um exemplo prático seria uma face humana como positiva e um pássaro como negativa, no contexto de detecção facial.

Dessa forma, ao utilizar o método descrito na seção 2.4.1, é possível imaginar que muitos dos recursos (*features*) utilizados não detectem traços relevantes nas imagens positivas, ou que detectem traços errôneos nas imagens negativas, ou que até mesmo não sejam úteis para a detecção de objetos.

Assim, a fim de solucionar tal problema, é utilizado um algoritmo de aprendizado/técnica de aprimoramento (do inglês, *Learning Algorithm/Boosting Technique*) conhecido como AdaBoost. Tal algoritmo/técnica é responsável por, através da criação de *Weak Learners*, filtros fracos no português, determinar os recursos que melhor representam o objeto de se deseja detectar. Cabe ressaltar que no algoritmo/técnica muitos dos recursos utilizados previamente são excluídos, assim mantendo apenas aqueles que trazem resultados relevantes para a detecção. (VIOLA; JONES, 2001)

2.4.3. *Attentional Cascade*

A partir dos recursos que melhor representam os objetos que se deseja detectar, advindos do algoritmo de aprendizado/técnica de aprimoramento AdaBoost, foi proposto um terceiro algoritmo, a fim de diminuir os esforços computacionais, conhecido como *Attentional Cascade*.

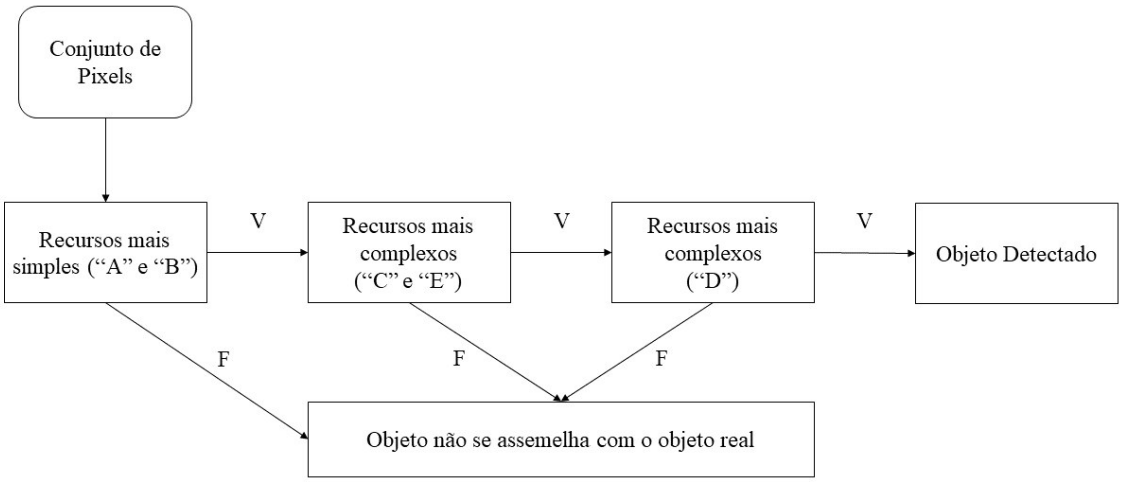
Tal algoritmo, como o próprio nome sugere, funciona em forma de cascata, onde é necessário que todos os procedimentos intermediários apresentem respostas verdadeiras, para que a resposta final seja verdadeira. Caso alguma das respostas intermediárias seja falsa, o algoritmo para de ser executado e passa para o próximo conjunto de pixels na imagem. A Figura 11 exemplifica a ideia principal do algoritmo.

Em mais detalhes, o método se baseia na utilização de diferentes recursos, em uma ordem determinada, a fim de verificar se em um conjunto de pixels há ou não traços que se assemelhem ao objeto que se deseja detectar. Inicialmente são utilizados recursos mais simples, como os exemplos "A" e "B" da Figura 7. Caso não sejam detectados traços relevantes, o algoritmo para de ser executado e passa para o próximo conjunto de pixels. Agora, caso sejam detectados traços relevantes, recursos mais complexos, como os exemplos "C", "D" e "E", são

utilizados para verificar mais a fundo a semelhança com o objeto que deseja ser encontrado. (VIOLA; JONES, 2001)

Caso todos os recursos retornem respostas verdadeiras para a semelhança com o objeto que se deseja detectar, o algoritmo fornece uma resposta final verdadeira para a detecção do objeto.

Figura 11: Representação do algoritmo Attentional Cascade



Fonte: Autoria Própria

2.4.4. Reconhecimento facial LBPH – Local Binary Patterns Histogram

O algoritmo *Local Binary Patterns Histogram* foi introduzido no contexto de reconhecimento facial no ano de 2004 por Ahonen, Hadid e Pietikäinen em seu trabalho "*Face Recognition with Local Binary Patterns*" (ROSEBROCK, 2021). Em termos gerais, o algoritmo consiste na utilização de histogramas para o armazenamento do número de vezes que uma determinada intensidade está presente na imagem. Abaixo será explicado o método em mais detalhes.

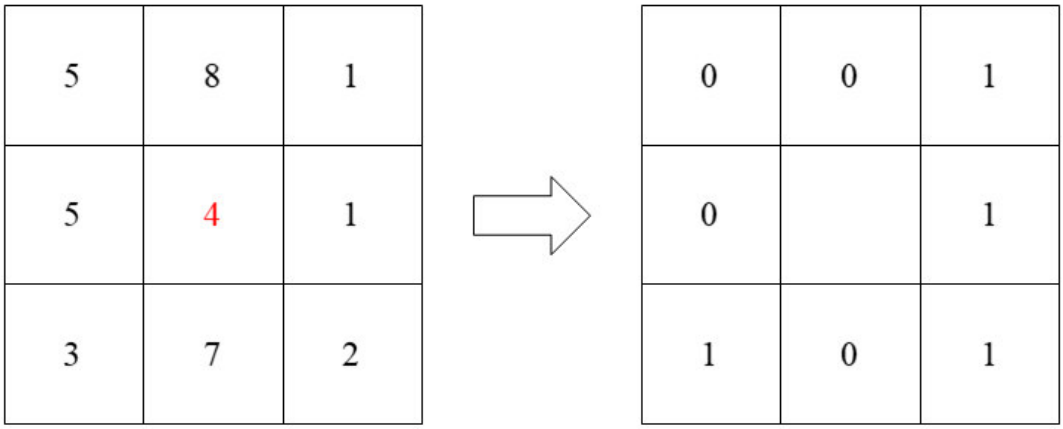
2.4.4.1. Pixels Vizinhos

Primeiramente, a fim de utilizar o algoritmo é necessário que a imagem seja convertida em escalas de cinza através da utilização da equação (5), apresentada na seção 2.3.3.3.

Em seguida, para cada pixel, já na escala cinza, são selecionados 8 pixels vizinhos, pelos quais se deseja que o algoritmo itere. Tal iteração consiste na comparação do valor do pixel central com os pixels vizinhos. Caso o valor do pixel central seja maior ou igual ao do pixel vizinho em análise, é atribuído o número 1 a esse vizinho. Caso o valor do pixel central seja

menor, é atribuído o número 0 ao vizinho. Tais valores (0 e 1) serão utilizados posteriormente para a formação de um número binário. Abaixo, na Figura 12, pode ser verificado um exemplo esquemático desse processo. (ROSEBROCK, 2015)

Figura 12: Representação esquemática da atribuição de valores aos pixels vizinhos.

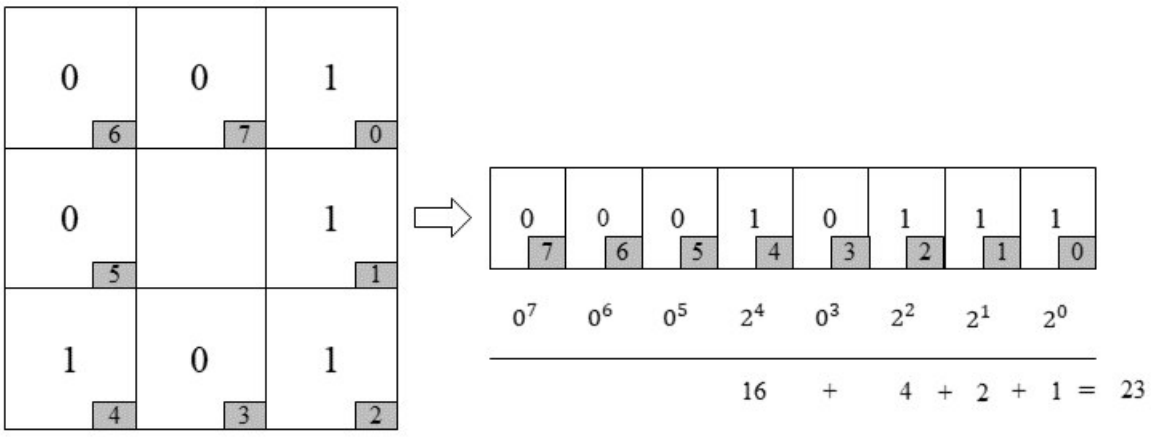


Fonte: Autoria Própria

2.4.4.2. Pixel Central

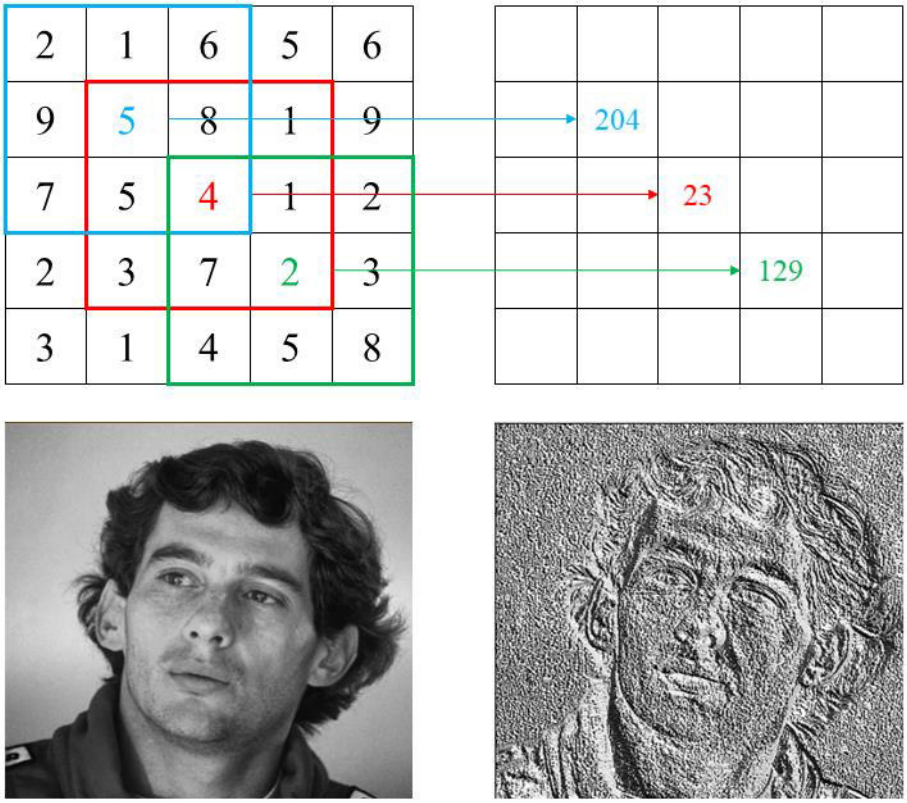
Para a determinação do valor associado ao pixel central, o qual é o pixel que desejamos determinar desde o início do algoritmo, os pixels vizinhos serão enumerados de 0 a 7 e percorridos no sentido horário ou anti-horário. O importante é manter o mesmo padrão para todos os pixels da imagem. Com isso, os valores de 0 e 1 serão concatenados formando um número binário de 8-bits que será transformado em decimal, retornando o valor do pixel central. Abaixo, na Figura 13, é possível verificar o processo descrito acima e na Figura 14 a imagem final após as transformações. (ROSEBROCK, 2015)

Figura 13: Transformação dos valores dos pixels vizinhos em binário e transformação do binário para decimal



Fonte: Autoria Própria

Figura 14: Representação dos pixels finais após as transformações e a aplicação de todo o algoritmo em uma imagem real

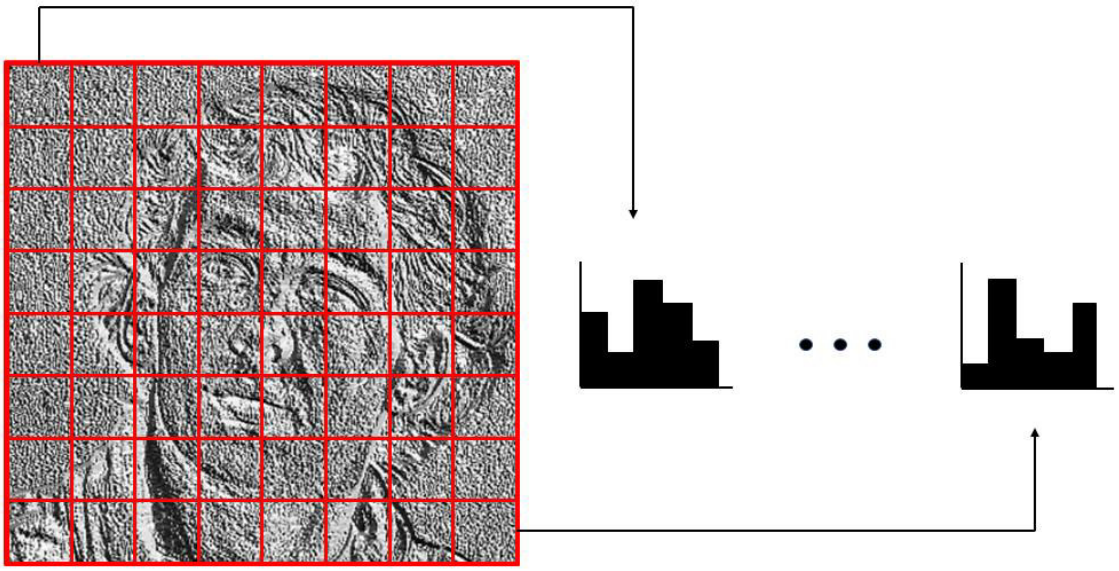


Fonte: Modificado de SOUZA (2019)

2.4.4.3. Histograma

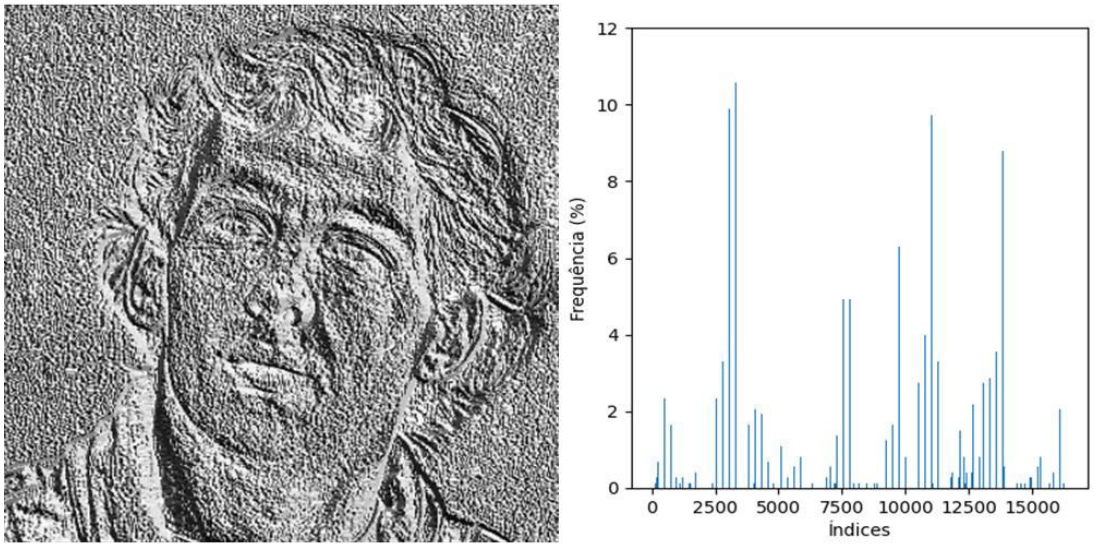
Com a imagem devidamente transformada, como verificado na Figura 14, a mesma será dividida em 64 sub-regiões. Para cada sub-região será gerado um histograma contendo as intensidades dos pixels presentes naquela porção da imagem e a frequência com que cada intensidade se apresenta. E ao final, todos os 64 histogramas serão concatenados em um único histograma, o qual conterá a frequência com que cada intensidade de pixel se apresenta. Nas Figuras 15 e 16 é possível verificar um modelo esquemático do processo e o histograma final da Figura 14. (SERENGIL, 2020)

Figura 15: Representação esquemática da formação dos histogramas.



Fonte: Modificado de SOUZA (2019)

Figura 16: Representação do histograma final da Figura 15.



Fonte: Modificado de SOUZA (2019)

Como foi dito anteriormente, o histograma final é uma concatenação dos 64 histogramas individuais. Dessa forma, o histograma final não combinará os resultados individuais, mas irá ordená-los um ao lado do outro. Tal ordenamento explica, com clareza, o motivo do eixo x, do histograma da Figura 16, possuir 16.384 (64 retângulos $\times$ 256 intensidade) valores. (OpenCV, [201-])

2.4.4.4. Comparação de Imagens

Uma vez realizado o processo descrito na seção 2.4.4.3, os histogramas finais serão armazenados em um arquivo do tipo YML, que funcionará como um banco de dados para futuras comparações, que terão como finalidade determinar a semelhança entre uma nova imagem e as utilizadas para treinar o algoritmo.

Tais comparações se darão mediante a entrada de uma nova imagem, a qual passará por todos os processos e métodos descritos acima, a fim de também gerar um novo histograma que será comparado com os presentes no banco de dados, através da equação (7).

$$D = \sqrt{(p_0 - q_0)^2 + (p_1 - q_1)^2 + \cdots + (p_i - q_i)^2} = \sqrt{\sum_{i=0}^{n} (p_i - q_i)^2} \qquad (7)$$

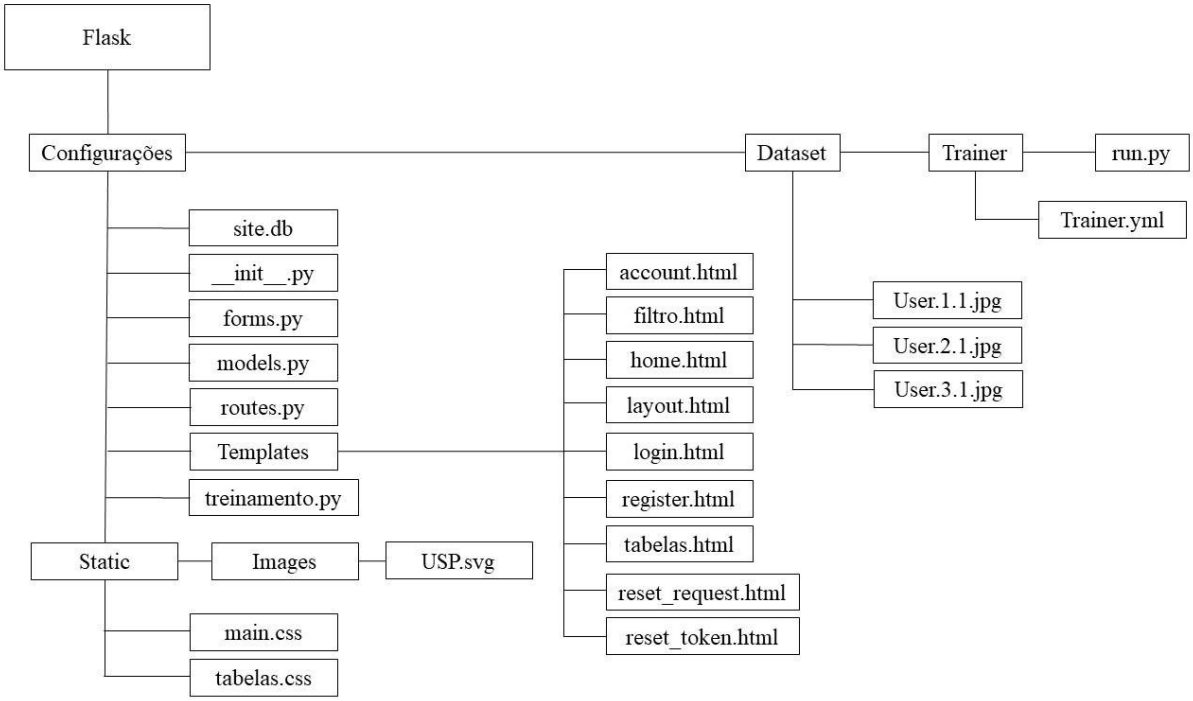
Onde p e q representam, respectivamente, as frequências de cada um dos índices i dos histogramas 1 (nova imagem) e 2 (banco de dados) e D a distância euclidiana entre os dois histogramas. Quanto maior o valor de D, menor será a correspondência entre os histogramas, indicando que a nova imagem não corresponde a determinada imagem armazenada no banco de dados.

3. DESENVOLVIMENTO DO TRABALHO

Esta seção tem como objetivo apresentar e explicar os resultados do desenvolvimento do aplicativo para a detecção e reconhecimento facial e criação de um banco de dados para futuras consultas.

Afim de facilitar o entendimento e compreensão dos resultados, a organização/estrutura do aplicativo de reconhecimento facial desenvolvido é mostrado na Figura 17. A partir dessa figura, será explicado cada um dos diretórios e suas funções dentro do aplicativo.

Figura 17: Organização e estrutura do aplicativo



Fonte: Autoria Própria

3.1. Configurações

O diretório de Configurações é responsável por armazenar os códigos e funcionalidades que possibilitaram a detecção e reconhecimento facial, a criação das páginas HTML (criação das páginas web), a formatação das páginas web (CSS) e a utilização do banco de dados (site.db).

3.1.1. __init__.py

O código presente no arquivo __init__.py, apesar de ser pequeno, é responsável por configurar a aplicação do microframework Flask, configurar a aplicação do banco de dados

SQLite e do framework SQLAlchemy e possibilitar a encriptação da senha cadastrada pelo administrador. Na Figura 18 é possível verificar o código do arquivo __init__.py na íntegra.

Figura 18: Código do arquivo __init__.py na íntegra

```
1   from flask import Flask
2   from flask_sqlalchemy import SQLAlchemy
3   from flask_bcrypt import Bcrypt
4   from flask_login import LoginManager
5   from flask_mail import Mail
6
7   #Aplicação Flask
8   app = Flask(__name__)
9   app.config['SECRET_KEY'] = '5791628bb0b13ce0c676dfde280ba245'
10
11  #Banco de dados SQLite e SQLAlchemy
12  app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
13  db = SQLAlchemy(app)
14
15  #Encriptar de Senhas
16  bcrypt = Bcrypt(app)
17
18  #Sistema de Login
19  login_manager = LoginManager(app)
20  login_manager.login_view = 'login'
21  login_manager.login_message_category = 'info'
22
23  #Configurações de Email
24  app.config['MAIL_SERVER'] = 'smtp.googlemail.com'
25  app.config['MAIL_PORT'] = 587
26  app.config['MAIL_USE_TLS'] = True
27  app.config['MAIL_USERNAME'] = '[redacted]'
28  app.config['MAIL_PASSWORD'] = '[redacted]'
29  mail = Mail(app)
```

Fonte: modificado de SCHAFER, [201-]b; [201-]c; [201-]d; [201-]e

Linhas 1 e 2: foram importados, respectivamente, o microframework Flask e SQLAlchemy, que serão responsáveis por criar a aplicação web e por gerenciar o banco de dados, como previamente explicados nas seções 2.2.1 e 2.2.2.

Linhas 3, 4 e 5: são importados, respectivamente, as bibliotecas/módulo **flask_bcrypt**, **flask_login e flask_mail**. A biblioteca **flask_bcrypt** é responsável por encriptar e descriptar a senha utilizada pelo administrador, a fim de que a senha não possa ser verificada visualmente no banco de dados por qualquer um. A biblioteca **flask_login** é responsável por implementar à

aplicação um sistema de *login* e *logout* fácil e rápido, além de permitir que verificações sejam utilizadas para o acesso a determinadas pastas no aplicativo. E a biblioteca **flask_mail** é responsável por mediar os processos de envio de e-mail no caso do esquecimento da senha utilizada.

Linhas 8 e 9: são criados, respectivamente, a aplicação Flask e a chave secreta (Secret Key), que não permite que os dados utilizados na sessão sejam alterados por malwares ou sistemas externos.

Linhas 12, 13 e 16: são criados, respectivamente, o banco de dados SQLite, os métodos de interação com o banco e os métodos de encriptação para as senhas.

Linhas 19 a 21 e 24 a 29: são incorporados ao código, respectivamente, o sistema de login, com suas validações e o sistema de encaminhamento de e-mail. No lugar dos traços vermelhos, o administrador deve inserir o seu e-mail (Gmail) e senha.

3.1.2. forms.py

No arquivo forms.py é possível verificar os formulários, com suas respectivas validações, utilizados para o cadastramento do administrador, *login* do administrador, cadastramento de pessoas no banco de dados, encaminhamento de e-mail e recadastramento de senhas. Na Figura 19 é possível verificar as bibliotecas utilizadas para a criação dos formulários.

Figura 19: Bibliotecas utilizadas no código forms.py

```
1  from flask_wtf import FlaskForm
2  from wtforms import StringField, PasswordField, SubmitField, BooleanField, IntegerField
3  from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
4  from flaskblog.models import User, Number
```

Fonte: modificado de SCHAFER, [201-]b

Para a construção dos formulários foi utilizada a biblioteca **flask_wtf**, a qual possibilita que diferentes validações sejam executadas ao submeter um novo formulário. Um exemplo prático dessas validações se dá no cadastramento de um nome que já é existente no banco de dados. O formulário exibe uma mensagem de erro, indicando que o nome já foi utilizado e até que a pessoa não altere, o formulário não será enviado para o cadastramento. Cabe ressaltar que o nome é apenas um dos parâmetros de verificação, sendo assim, e-mail, número, senhas ou qualquer tipo de parâmetro pode ser utilizado como parâmetro de validação.

Na Figura 20 é possível verificar os formulários escritos para o cadastramento dos administradores e suas respectivas validações

Figura 20: Cadastramento do perfil de administrador

```
6   #Forms para Cadastramento do Administrador
7   class RegistrationForm(FlaskForm):
8       nome_administrador = StringField('Nome do Usuário',
9                                        validators=[DataRequired(), Length(min=2, max=20)])
10      email_administrador = StringField('Email',
11                                        validators=[DataRequired(), Email()])
12      senha_administrador = PasswordField('Senha', validators=[DataRequired()])
13      confirmar_senha = PasswordField('Confirmar Senha',
14                                      validators=[DataRequired(), EqualTo('senha_administrador')])
15      submit = SubmitField('Registrar')
16
17  #Validação do nome de usuário e email
18      def validate_nome_administrador(self, nome_administrador):
19          user = User.query.filter_by(nome_administrador=nome_administrador.data).first()
20          if user:
21              raise ValidationError('Nome já utilizado. Escolha Outro.')
22
23      def validate_email_administrador(self, email_administrador):
24          user = User.query.filter_by(email_administrador=email_administrador.data).first()
25          if user:
26              raise ValidationError('Email já utilizado. Escolha Outro.')
```

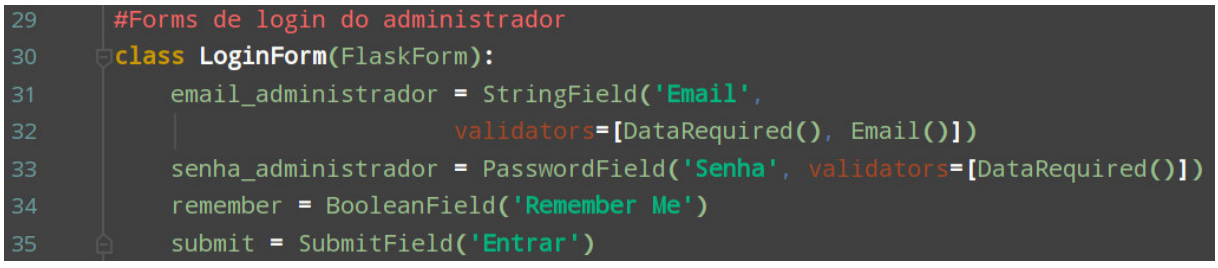
Fonte: modificado de SCHAFER, [201-]b

Linhas 7 a 15: É possível verificar a construção do formulário para o cadastramento do administrador. Nele é requerido que seja fornecido um nome de usuário (nome_administrador), um e-mail para *login* (email_administrador), uma senha (senha_administrador) e uma confirmação da senha (confirmar_senha).

Linhas 18 a 26: É verificado se o nome e e-mail do administrador é existente no banco de dados. Caso afirmativo para qualquer um dos casos, é pedido que seja trocado o parâmetro em questão, até que o escolhido não esteja presente no banco. Outro ponto que, apesar de não aparecer junto as validações, funciona como uma é a confirmação de senha (linha 13), a qual impossibilita que o usuário seja cadastrado caso as senhas sejam diferentes.

Nas Figuras 21, 22 e 23, é possível encontrar os formulários de *login* do administrador, cadastramento de pessoas no banco de dados e encaminhamento de e-mail para o recadastro da senha do administrador.

Figura 21: Formulário para o *login* do administrador



```
29  #Forms de login do administrador
30  class LoginForm(FlaskForm):
31      email_administrador = StringField('Email',
32                                        validators=[DataRequired(), Email()])
33      senha_administrador = PasswordField('Senha', validators=[DataRequired()])
34      remember = BooleanField('Remember Me')
35      submit = SubmitField('Entrar')
```

Fonte: modificado de SCHAFER, [201-]b

Linhas 30 a 35: Após o cadastramento do número de administradores desejados, a opção de cadastramento será retirada e no lugar será disponibilizado a opção de *login*, na qual será requerido o e-mail (email_administrador) do usuário e senha (senha_administrador). Caso um dos parâmetros esteja incorreto, não será possível efetuar o *login* e a pessoa será redirecionado novamente para página de *login*.

Figura 22: Formulário para o cadastramento de pessoas no banco de dados

```
#Forms para o cadastramento de pessoas no banco de dados
class CadastroForm(FlaskForm):
    nome_usuario = StringField("Nome", validators=[DataRequired()])
    numero_usuario = IntegerField("Número", validators=[DataRequired()])
    submit = SubmitField('Submit')

#Validação do nome da pessoa que se deseja cadastrar e do número associado a essa pessoa
    def validate_nome_usuario(self, nome_usuario):
        user = Cadastro.query.filter_by(nome_usuario=nome_usuario.data).first()
        if user:
            raise ValidationError('Nome já existente no banco de dados. Escolha Outro.')

    def validate_numero_usuario(self, numero_usuario):
        user = Cadastro.query.filter_by(numero_usuario=numero_usuario.data).first()
        if user:
            raise ValidationError('Número já existente no banco de dados. Escolha Outro.')
```

Fonte: Autoria Própria

Linhas 38 a 52: Com o login efetuado pelo administrador, o mesmo será redirecionado para uma página de cadastramento de pessoas e treinamento do algoritmo de reconhecimento. No formulário de cadastramento de pessoas é requerido que o administrador atribua um nome (nome_usuario) e número (numero_usuario) a essa pessoa. Cada nome e número deve ser único, de forma que o sistema possa reconhecer diferentes pessoas. O nome é utilizado para ser disposto na foto final da pessoa e o número para o treinamento do algoritmo.

Figura 23: Formulários para o encaminhamento de e-mail e recadastramento de senhas

```
#Forms para o encaminhamento do email para o recadastramento da senha
class RequestResetForm(FlaskForm):
    email_reset = StringField('Email',
                        validators=[DataRequired(), Email()])
    submit = SubmitField('Enviar e-mail')

#Validação do email utilizado pelo administrador
    def validate_email(self, email_administrador):
        user = User.query.filter_by(email_administrador=email_administrador.data).first()
        if user is None:
            raise ValidationError('Email não é utilizado em nenhuma conta! Verifique o email utilizado')

#Forms para o cadastramento da nova senha
class ResetPasswordForm(FlaskForm):
    nova_senha = PasswordField('Senha', validators=[DataRequired()])
    confirmar_nova_senha = PasswordField('Confirmar Senha',
                                         validators=[DataRequired(), EqualTo('nova_senha')])
    submit = SubmitField('Resetar Senha')
```

Fonte: modificado de SCHAFER, [201-]b

Linha 55 a 64: Caso o administrador esqueça a senha do *login*, foi criado um formulário que possibilita o envio de um *link* para o recadastramento. Neste formulário é necessário que o administrador informe apenas o e-mail utilizado no registro do perfil. Caso o e-mail informado não esteja presente no banco de dados, o formulário exibirá um erro e o e-mail não será enviado.

Linha 67 a 71: No formulário de recadastramento é necessário que o administrador informe a sua nova senha e a confirmação da nova senha. Feito isso, o administrador será redirecionado para a página de login, onde poderá efetuar o mesmo.

3.1.3. models.py

No arquivo models.py é possível encontrar todos os códigos relacionados a criação de tabelas dentro do banco de dados SQLite. Essas tabelas são diferentes instâncias dentro do banco de dados, responsáveis pelo armazenamento de um conjunto de dados específicos.

A primeira tabela é relacionada ao armazenamento dos perfis dos administradores. A segunda tabela é relacionada ao cadastro das pessoas no banco de dados, a fim de serem reconhecidas posteriormente pelo sistema. E a terceira é relacionada ao armazenamento dos nomes das pessoas que foram reconhecidas, a data e hora em que foram reconhecidas, como um registro de entrada e saída. A seguir é possível verificar os códigos escritos, Figuras 24, 25, 26, 27 e 28, e suas respectivas explicações.

Figura 24: Bibliotecas utilizadas no código models.py

```
1  from datetime import datetime
2  from flaskblog import db, login_manager, app
3  from flask_login import UserMixin
4  from itsdangerous import TimedJSONWebSignatureSerializer as Serializer
```

Fonte: modificado de SCHAFER, [201-]c

Para a criação das tabelas foi utilizado o banco de dados SQLite, localizado na própria máquina, no arquivo **site.db**, assim como o SQLAlchemy, que é responsável por mediar a entra, saída e alteração do banco.

Para a determinação da data e hora e criação de um *token*/*link* aleatório, foram utilizadas as bibliotecas **datetime** e **itsdangerous**, respectivamente. E a fim de determinar quando e qual administrador esta *logado* ao aplicativo, foi utilizado a biblioteca **flask_login**, novamente.

Figura 25: Determinação de qual administrador esta *logado* no aplicativo

```
6  #Verificação do usuário
7  @login_manager.user_loader
8  def load_user(user_id):
9      return User.query.get(int(user_id))
```

Fonte: modificado de SCHAFER, [201-]d

A fim de o sistema de login funcionar corretamente, é necessário que a aplicação entenda quando e qual administrador está *logado* ao sistema. Para isso é utilizado a função **load_user**, a qual, com base no **id** do usuário, determina qual dos usuários está *logado*, e consequentemente, quando está *logado*.

Figura 26: Tabela de perfil de administrador

```
11  class User(db.Model, UserMixin):
12      id = db.Column(db.Integer, primary_key=True)
13      nome_administrador = db.Column(db.String(20), unique=True, nullable=False)
14      email_administrador = db.Column(db.String(120), unique=True, nullable=False)
15      image_file = db.Column(db.String(20), nullable=False, default='default.jpg')
16      senha_administrador = db.Column(db.String(60), nullable=False)
17      posts = db.relationship('Post', backref='author', lazy=True)
18
19      def get_reset_token(self, expires_sec=1800):
20          s = Serializer(app.config['SECRET_KEY'], expires_sec)
21          return s.dumps({'user_id': self.id}).decode('utf-8')
22
23      @staticmethod
24      def verify_reset_token(token):
25          s = Serializer(app.config['SECRET_KEY'])
26          try:
27              user_id = s.loads(token)['user_id']
28          except:
29              return None
30          return User.query.get(user_id)
31
32      def __repr__(self):
33          return f"User('{self.nome_administrador}', '{self.email_administrador}', '{self.image_file}')"
```

Fonte: modificado de SCHAFER, [201-]c; [201-]e

Linhas 11 a 17: foi criado a tabela **User** com os seguintes parâmetros: **id**, **nome_administrador**, **email_administrador** e **senha_administrador**. O **id**, do inglês, **identification**, é um valor inteiro e único, atribuído automaticamente a cada um dos administradores conforme se cadastram no sistema. O **nome_administrador**, **email_administrador** e **senha_administrador** vem diretamente do formulário **RegistrationForm** apresentado na Figura 20. Assim, os valores fornecidos nos formulários serão salvos dentro da tabela **User**.

Linhas 19 a 21: foi criado um *token/link* aleatório, com base na chave secreta definida no arquivo **__ini__.py**, que será responsável pelo redirecionamento até a página de recadastramento de senha, caso ela seja esquecida. O *token/link* terá uma duração de 1800 segundos, ou 30 minutos, que será verificada pelo código das linhas 24 a 30. Caso o tempo seja excedido, o *token/link* se tornará inválido, sendo necessário gerar um novo.

Linhas 36 a 39: foi criada a tabela **Cadastro**, responsável por armazenar os parâmetros **id**, **nome_usuario** e **numero_usuario**. Da mesma forma que para a tabela **User**, os parâmetros

da tabela **Cadastro**, são advindos dos formulários apresentado na seção 3.1.2, mais especificamente do formulário **CadastroForm** apresentado na Figura 22.

Figura 27: Tabelas de cadastro de pessoas e de registro de reconhecimento

```
36  class Cadastro(db.Model):
37      id = db.Column(db.Integer, primary_key=True)
38      nome_usuario = db.Column(db.String(50), nullable=False)
39      numero_usuario = db.Column(db.Integer, unique=True, nullable=False)
40
41      def __repr__(self):
42          return f"Number('{self.nome_usuario}', '{self.numero_usuario}')"
43
44
45  class Registro(db.Model):
46      id = db.Column(db.Integer, primary_key=True)
47      nome_registro = db.Column(db.String(50), nullable=False)
48      data_registro_dia = db.Column(db.String(50), nullable=False)
49      data_registro_hora = db.Column(db.String(50), nullable=False)
50
51      def __repr__(self):
52          return f"Registro('{self.nome_registro}'," \
53                 f" '{self.data_registro_dia}', '{self.data_registro_hora}')"
```

Fonte: modificado de SCHAFER, [201-]c

E por último, foi criado uma tabela chamada **Registro**, onde são inseridos os nomes de todas as pessoas que são reconhecidas pelo algoritmo, juntamente com a hora e dia em que foram reconhecidas.

3.1.4. routes.py

No arquivo routes.py é possível encontrar todos os códigos responsáveis pelas funcionalidades da aplicação, como: o efetivo cadastramento das pessoas no banco de dados, o cadastramento dos administradores, treinamento do algoritmo de reconhecimento, acesso ao registro de reconhecimento, funcionamento do sistema de login e recadastramento de senhas. Todos esses códigos são mostrados das Figuras 28 a 40 e são seguidos de suas respectivas explicações.

Linhas 1 a 12: foram importadas as bibliotecas OpenCV, Numpy, os, datetime e Pillow (PIL), as variáveis do código __init__.py (seção 3.1.1), os formulários de cadastro e recadastro (seção 3.1.2), as tabelas do banco de dados (seção 3.1.3), sistema de login e sistema de envio de e-mail.

Figura 28: Bibliotecas utilizadas no código routes.py

```
1  from flask import render_template, url_for, flash, redirect, request, Response
2  from flaskblog import app, db, bcrypt, mail
3  from flaskblog.forms import (RegistrationForm, LoginForm, CadastroForm,
4                               RequestResetForm, ResetPasswordForm)
5  from flaskblog.models import User, Cadastro, Registro
6  from flask_login import login_user, current_user, logout_user, login_required
7  from flask_mail import Message
8  import os
9  import cv2
10 import datetime
11 from PIL import Image
12 import numpy as np
```

Fonte: modificado de SCHAFER, [201-]b; [201-]c; [201-]d; [201-]e

Figura 29: Constantes para a detecção facial e reconhecimento facial

```
15 reconhecimento = cv2.face.LBPHFaceRecognizer_create()
16 reconhecimento.read('trainer/trainer.yml')
17 caminho_face_cascade = os.path.dirname(cv2.__file__) + \
18                         "/data/haarcascade_frontalface_alt2.xml"
19 faceCascade = cv2.CascadeClassifier(caminho_face_cascade)
20 font = cv2.FONT_HERSHEY_SIMPLEX
```

Fonte: Autoria própria

Ao contrário do que se parece ao explicar, o algoritmo de detecção facial, apresentado na seção 2.4, é computacionalmente custoso, necessitando de um grande tempo e uma grande quantidade de imagens para o seu treinamento adequado.

Dessa forma, a fim de não serem necessárias múltiplas execuções do algoritmo, uma para cada imagem, os desenvolvedores da biblioteca OpenCV disponibilizaram um arquivo XML, com o algoritmo pré-treinado, o qual pode ser implementado no código como visto nas linhas 17, 18 e 19.

Entretanto, diferentemente do algoritmo de detecção facial, o algoritmo de reconhecimento deve ser executado múltiplas vezes, a fim de gerar um histograma para cada imagem fornecida. Assim, com o intuito de executá-lo eficientemente, os desenvolvedores da biblioteca OpenCV, disponibilizaram códigos simples, como os verificados nas linhas 15 e 16, que podem ser implementados facilmente dentro do código.

Figura 30: Código de detecção facial

```
23  def deteccao():
24      cap = cv2.VideoCapture(0)
25      while True:
26          ret, img = cap.read()
27          gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
28          faces = faceCascade.detectMultiScale(gray, scaleFactor=1.2,
29                                               minNeighbors=5, minSize=(20, 20))
30          for (x,y,w,h) in faces:
31              cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
32              roi_gray = gray[y:y+h, x:x+w]
33              roi_color = img[y:y+h, x:x+w]
34          ret, buffer = cv2.imencode('.jpg', img)
35          frame = buffer.tobytes()
36          yield (b'--frame\r\n'
37                 b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Fonte: Autoria própria

Linhas 23 a 37: foi definida a função **"deteccao"**, a qual é responsável por detectar as faces presentes em um vídeo ou imagem. Esta função foi idealizada com o objetivo de verificar a posição da câmera antes da função de captura, pois imagens que estejam com uma qualidade ruim ou com objetos cobrindo parcialmente o rosto, podem atrapalhar o reconhecimento.

Figura 31: Captura de imagens para o banco de dados do usuário

```
40  def captura_banco_dados():
41      contador = 0
42      cam = cv2.VideoCapture(0)
43      face_id = Cadastro.query.order_by(Cadastro.id.desc()).first()
44      index = face_id.numero_usuario
45      while (True):
46          ret, img = cam.read()
47          gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
48          faces = faceCascade.detectMultiScale(gray, 1.3, 5)
49          for (x, y, w, h) in faces:
50              cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
51              contador += 1
52              cv2.imwrite("dataset/User." + str(index) + '.' +
53                          str(contador) + ".jpg", gray[y:y + h, x:x + w])
54          ret, buffer = cv2.imencode('.jpg', img)
55          frame = buffer.tobytes()
56          yield (b'--frame\r\n'
57                 b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Fonte: Autoria própria

Linhas 40 a 57: foi escrita a função "captura_banco_dados", a qual é responsável por capturar diversas imagens da face, a fim de gerar uma quantidade significativa de histogramas

para comparações. Estas imagens são armazenadas dentro da pasta **Dataset**, apresentada na Figura 17, da seguinte forma: **User.número.contador.jpg**. O nome **User** apenas indica que a foto é relacionada ao banco de dados de usuários, enquanto o **número** é relacionado ao número fornecido pelo administrador no formulário **CadastroForm** (seção 3.1.2) e o **contador** registra o número da imagem que foi tirado.

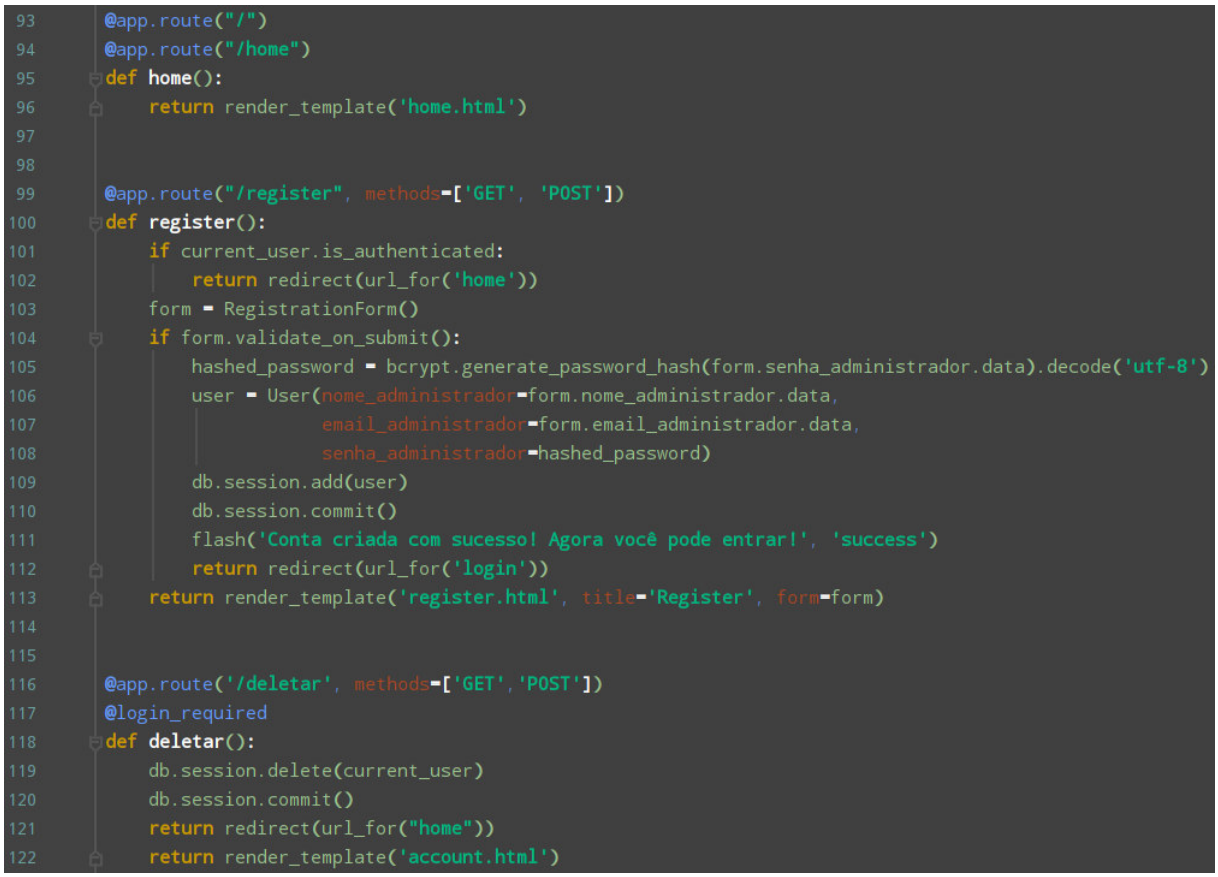
Figura 32: Código de reconhecimento facial

```
60  def gen_frames():
61      cam = cv2.VideoCapture(0)
62      while True:
63          ret, img = cam.read()
64          gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
65          faces = faceCascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5)
66          for (x, y, w, h) in faces:
67              cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
68              id, confidence = reconhecimento.predict(gray[y:y + h, x:x + w])
69              user = Cadastro.query.filter_by(numero_usuario=id).first()
70              id = user.nome_usuario
71              if (confidence < 100):
72                  id = id
73                  confidence = "  {0}%".format(round(100 - confidence))
74              else:
75                  id = "unknown"
76                  confidence = "  {0}%".format(round(100 - confidence))
77              cv2.putText(img, str(id), (x + 5, y - 5), font, 1, (255, 255, 255), 2)
78              cv2.putText(img, str(confidence), (x + 5, y + h - 5), font, 1, (255, 255, 0), 1)
79              now = datetime.datetime.now()
80              dia = str(now.strftime("%d/%m/%Y"))
81              hora = str(now.strftime("%H:%M:%S"))
82              registro = Registro(nome_registro=id, data_registro_dia=dia,
83                                  data_registro_hora=hora)
84              db.session.add(registro)
85              db.session.commit()
86          ret, buffer = cv2.imencode('.jpg', img)
87          frame = buffer.tobytes()
88          yield (b'--frame\r\n'
89                 b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Fonte: Autoria própria

Linhas 60 a 89: foi definida a função **"gen_frames"**, a qual é responsável por: gerar um novo histograma para cada face detectada na imagem, comparar os novos histogramas com os histogramas armazenados no arquivo **Trainer.yml**, armazenar na tabela **Registro** (seção 3.1.3) o nome da pessoa, a data e a hora em que foi reconhecida (caso ela seja conhecida) e escrever na imagem o nome e a percentagem de semelhança com a pessoa do banco de dados. Em outras palavras, a função "gen_frames" é responsável por uma parcela relativamente grande e importante da aplicação no geral.

Figura 33: Definição do caminho para as páginas “home.html” (Início) e “register.html” (registro) e definição da função “deletar”



```
93  @app.route("/")
94  @app.route("/home")
95  def home():
96      return render_template('home.html')
97
98
99  @app.route("/register", methods=['GET', 'POST'])
100 def register():
101     if current_user.is_authenticated:
102         return redirect(url_for('home'))
103     form = RegistrationForm()
104     if form.validate_on_submit():
105         hashed_password = bcrypt.generate_password_hash(form.senha_administrador.data).decode('utf-8')
106         user = User(nome_administrador=form.nome_administrador.data,
107                     email_administrador=form.email_administrador.data,
108                     senha_administrador=hashed_password)
109         db.session.add(user)
110         db.session.commit()
111         flash('Conta criada com sucesso! Agora você pode entrar!', 'success')
112         return redirect(url_for('login'))
113     return render_template('register.html', title='Register', form=form)
114
115
116 @app.route('/deletar', methods=['GET','POST'])
117 @login_required
118 def deletar():
119     db.session.delete(current_user)
120     db.session.commit()
121     return redirect(url_for("home"))
122     return render_template('account.html')
```

Fonte: modificado de SCHAFER, [201-]b; [201-]c

Linhas 93 a 96: foram definidos dois caminhos para o acesso a página inicial (Home). Um deles é com a utilização da URL http://127.0.0.1:5000/ e o outro pela URL http://127.0.0.1:5000/home. Os números 127.0.0.1:5000 são referentes ao localhost, ou servidor local em português, criado pelo microframework Flask.

Linhas 99 a 113: o usuário pode ter acesso a página de cadastro de administrador (URL http://127.0.0.1:5000/register), onde, com o formulário **RegistrationForm** (seção 3.1.2), poderá se registrar como um administrador, tendo os dados salvos na tabela **User** (seção 3.1.3). Cabe ressaltar que o número de administradores pode ser limitado.

Linhas 116 a 122: o administrador, caso desejado, tem a possibilidade de deletar o seu próprio registro do banco de dados **User**, apresentado na Figura 26.

Figura 34: Sistema de login para os administradores

```
125  @app.route("/login", methods=['GET', 'POST'])
126  def login():
127      if current_user.is_authenticated:
128          return redirect(url_for('home'))
129      form = LoginForm()
130      if form.validate_on_submit():
131          user = User.query.filter_by(email_administrador=form.email_administrador.data).first()
132          if user and bcrypt.check_password_hash(user.senha_administrador,
133                                                 form.senha_administrador.data):
134              login_user(user, remember=form.remember.data)
135              next_page = request.args.get('next')
136              return redirect(next_page) if next_page else redirect(url_for('account'))
137          else:
138              flash('Login sem sucesso. Por favor, verifique email e senha.', 'danger')
139      return render_template('login.html', title='Login', form=form)
```

Fonte: modificado de SCHAFER, [201-]d

Linhas 125 a 139: foi implementado o sistema de login para os administradores. Este sistema faz uso da tabela **User** a fim de verificar a existência do e-mail e senha que foram inseridos no formulário **LoginForm**, na página de *login*. Caso o e-mail e senha estejam corretos, o administrador é direcionado a página **account**, onde poderá cadastrar novas usuários no banco de dados.

Figura 35: Sistema de Logout, Reconhecimento facial, Detecção Facial e Captura de imagens para o banco de dados de usuários

```
141  @app.route("/logout")
142  def logout():
143      logout_user()
144      return redirect(url_for('home'))
145
146
147  @app.route('/video_feed')
148  def video_feed():
149      return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
150
151
152  @app.route('/detecção_facial')
153  def deteccao_facial():
154      return Response(deteccao(), mimetype='multipart/x-mixed-replace; boundary=frame')
155
156
157  @app.route('/data_set')
158  def banco_dados():
159      return Response(captura_banco_dados(), mimetype='multipart/x-mixed-replace; boundary=frame')
```

Fonte: Autoria Própria

Linhas 141 a 144: é realizado o sistema de *logout* do administrador. O sistema é relativamente simples, uma vez que é utilizada a biblioteca **flask_login** apresentada na Figura 16.

Linhas 147 a 149: é definida a resposta mais importante do aplicativo, o reconhecimento facial. Tal resposta, pelo fato de ser em formato de vídeo, assim como as demais que serão

definidas logo abaixo, necessita da utilização do **Response**, o qual permitirá que as imagens que formam o vídeo sejam atualizadas constantemente. E por último, dentro do **Response** é colocada a função de vídeo que se deseja utilizar, no caso a função "**gen_frames**".

Linhas 152 a 154 e 157 a 159: são definidas, respectivamente, as aplicações das funções **"deteccao"** e **"captura_banco_dados"**, definidas nas Figuras 30 e 31.

Figura 36: Treinamento do algoritmo LBPH

```
162  @app.route('/data_file')
163  def treinamento_imagens():
164      file = open(r'face_training.py', 'r').read()
165      exec(file)
166      return Response()
```

Fonte: Autoria própria

Linhas 162 a 166: foi definida a função que possibilita o treinamento o algoritmo LBPH, explicado na seção 2.4.4. Entretanto, diferentemente das funções de vídeos mencionadas acima, as quais possuem suas respostas apresentadas na própria aplicação, o código do treinamento é executado diretamente no console Python, não sendo mediado pelo microframework Flask. O principal problema que isso gera, é a necessidade de reiniciar a aplicação após a execução do código, uma vez que o arquivo Trainer.yml não será atualizado na mesma sessão.

Figura 37: Página dos administradores e página de registro de reconhecimento

```
169  @app.route("/account", methods=['GET', 'POST'])
170  @login_required
171  def account():
172      form = CadastroForm()
173      if form.validate_on_submit():
174          dados = Cadastro(nome_usuario=form.nome_usuario.data, numero_usuario=form.numero_usuario.data)
175          db.session.add(dados)
176          db.session.commit()
177          flash('Agora, clique em Iniciar Captura', 'success')
178          return redirect(url_for('account'))
179      return render_template('account.html', title='Account', form=form)
180
181
182  @app.route('/tabelas', methods=['GET', 'POST'])
183  @login_required
184  def show_all():
185      return render_template('tabelas.html', registro = Registro.query.all(),
186                             numero_usuario = Cadastro.query.all(),
187                             data = Registro.query.with_entities(Registro.data_registro_dia).distinct())
```

Fonte: modificado de SCHAFER, [201-]c

Linhas 169 a 179 e 182 a 187: são definidas, respectivamente, as páginas dos administradores, onde está exibido o formulário **CadastroForm**, mostrado na Figura 22 e utilizado a tabela **Cadastro** do banco de dados SQLite e a página de registro, onde somente os administradores podem acessar e verificar os registros de quem foi reconhecido, a data e a hora. Cabe ressaltar que, por se tratarem de páginas onde somente administradores podem ter acesso,

foi definida na linha 170 e 183 o comando **@login_required**, que, como o próprio nome sugere, faz com que qualquer um que deseje entrar, necessite fazer o login.

Entretanto, a fim de diminuir o tempo e esforços de uma procura, nas linhas 190 a 204 (Figura 38) através da utilização do SQLAlchemy, foi criado um sistema de consulta, onde os administradores podem procurar alguém pelo nome, data ou pelos dois ao mesmo tempo.

Figura 38: Sistema de consulta de pessoas

```
190 @app.route('/select', methods=['GET', 'POST'])
191 @login_required
192 def filtrar():
193     nome = request.args.get('nome_usuario')
194     data = request.args.get('data_usuario')
195     if nome == 'Nulo':
196         registro = Registro.query.filter_by(data_registro_dia=data)
197     elif data == 'Nulo':
198         registro = Registro.query.filter_by(nome_registro=nome)
199     elif nome == 'Nulo' and data == 'Nulo':
200         registro = Registro.query.all()
201     else:
202         registro = Registro.query.filter(Registro.data_registro_dia == data).\
203             filter(Registro.nome_registro == nome)
204     return render_template('filtro.html', registro=registro)
```

Fonte: Autoria Própria

Figura 39: Definição do número de administradores e função de envio de e-mails para o recadastramento de senhas

```
206 def numero_admin():
207     if len(User.query.all()) <= 3:
208         return 0
209     else:
210         return 1
211
212
213 app.jinja_env.globals.update(clever_function=numero_admin)
214
215
216 def send_reset_email(user):
217     token = user.get_reset_token()
218     msg = Message('Recadastramento de senha',
219                   sender='noreply@demo.com',
220                   recipients=[user.email_administrador])
221     msg.body = f'''Para recadastrar a senha, clique no link abaixo:
222 {url_for('reset_token', token=token, _external=True)}
223 Caso este e-mail seja um engano, por favor desconsiderar.
224 '''
225     mail.send(msg)
```

Fonte: modificado de SCHAFER, [201-]e

Linhas 206 a 210: foi implementado um sistema de contagem de administradores, no qual a opção de se registrar como administrador estará disponível até um certo número de pessoas se cadastrarem. Após esse número ser atingido, a opção será retirada da página inicial.

Linhas 216 a 225: foram definidos os seguintes parâmetros do e-mail, que será enviado ao administrador caso este esqueça a senha: e-mail do administrador (para qual e-mail que a mensagem será enviada), título, corpo do e-mail e link para o recadastramento da senha.

Figura 40: Página de envio de e-mail e página de recadastramento de senha

```
228  @app.route("/reset_password", methods=['GET', 'POST'])
229  def reset_request():
230      if current_user.is_authenticated:
231          return redirect(url_for('home'))
232      form = RequestResetForm()
233      if form.validate_on_submit():
234          user = User.query.filter_by(email_administrador=form.email_reset.data).first()
235          send_reset_email(user)
236          flash('Um e-mail foi enviado com instruções.', 'info')
237          return redirect(url_for('login'))
238      return render_template('reset_request.html', title='Reset Password', form=form)
239
240
241  @app.route("/reset_password/<token>", methods=['GET', 'POST'])
242  def reset_token(token):
243      if current_user.is_authenticated:
244          return redirect(url_for('home'))
245      user = User.verify_reset_token(token)
246      if user is None:
247          flash('Este é um link não válido.', 'warning')
248          return redirect(url_for('reset_request'))
249      form = ResetPasswordForm()
250      if form.validate_on_submit():
251          hashed_password = bcrypt.generate_password_hash(form.nova_senha.data).decode('utf-8')
252          user.senha_administrador = hashed_password
253          db.session.commit()
254          flash('Sua senha foi atualizada. Agora é possível fazer login', 'success')
255          return redirect(url_for('login'))
256      return render_template('reset_token.html', title='Reset Password', form=form)
```

Fonte: modificado de SCHAFER, [201-]e

Por último, porém não menos importante, temos a configuração das páginas responsáveis por enviar o e-mail de recadastramento da senha e pelo recadastramento em si.

Para a primeira (Linhas 228 a 238), a fim de garantir que o link de recadastramento seja enviado apenas para o e-mail do administrador em questão, é verificado se o e-mail está armazenado na tabela **User** do banco de dados. Em caso afirmativo, o e-mail é enviado e é possível haver o recadastramento. Em caso negativo, é acusado um erro e o e-mail não é enviado.

E com relação a segunda (Linhas 241 a 256), pelo fato de ser apenas de recadastramento, apresenta apenas os campos para a inserção da nova senha e a confirmação da mesma. Com

isso, a senha antiga é substituída pela nova senha na tabela **User**, e o administrador pode efetuar o login normalmente.

3.1.5 treinamento.py

Neste arquivo, treinamento.py, é possível encontrar os códigos responsáveis pelo treinamento do algoritmo apresentado na seção 2.4. Na figura 41 é possível verificar o código na íntegra.

Figura 41: Código do arquivo treinamento.py

```
1   import cv2
2   import numpy as np
3   from PIL import Image
4   import os
5
6   caminho = 'dataset'
7   reconhecimento = cv2.face.LBPHFaceRecognizer_create()
8
9   CascadeCaminho = os.path.dirname(cv2.__file__) + "/data/haarcascade_frontalface_alt2.xml"
10  faceCascade = cv2.CascadeClassifier(CascadeCaminho)
11
12  def ImagemEIndice(caminho):
13      caminhoImagens = [os.path.join(caminho,f) for f in os.listdir(caminho)]
14      facesLocais = []
15      ids = []
16      for imagePath in caminhoImagens:
17          PIL_img = Image.open(imagePath).convert('L')
18          img_numpy = np.array(PIL_img,'uint8')
19          id = int(os.path.split(imagePath)[-1].split(".")[1])
20          faces = faceCascade.detectMultiScale(img_numpy)
21          for (x,y,w,h) in faces:
22              facesLocais.append(img_numpy[y:y+h,x:x+w])
23              ids.append(id)
24      return facesLocais,ids
25
26  facesLocais,ids = ImagemEIndice(caminho)
27  reconhecimento.train(facesLocais, np.array(ids))
28
29  reconhecimento.write('trainer/trainer.yml')
```

Fonte: Autoria Própria

Linhas 1 a 4: Foram importadas as bibliotecas OpenCV (cv2), responsável pela criação dos algoritmos mencionados na seção 2.4., Numpy e Pillow (PIL), responsáveis pelas manipulações das imagens e os, responsável por acessar os diretórios e nomes dos arquivos.

Linhas 6: Foi definido o caminho até o diretório responsável por armazenar as fotos dos usuários.

Linha 7: Criação do algoritmo de reconhecimento facial LBPH – Local Binary Patterns Histogram.

Linha 9 e 10: Criação do algoritmo de detecção facial HaarCascade.

Linhas 12 a 24: Criação de uma função responsável por acessar os nomes das imagens no diretório “dataset”, extrair os valores do número fornecido pelo administrador ao usuário (lembrando que o número extraído é o mesmo número presente no nome das imagens - **User.número.contador.jpg**), transformar a imagem para a escala cinza e salvar a localização das faces detectadas.

Linhas 26 e 27: Treinamento do algoritmo LBPH.

Linha 28: Os resultados advindos do treinamento do algoritmo são salvos em um arquivo denominado “trainer.yml” dentro do diretório “trainer”.

3.1.6. Templates

Na seção Templates serão apresentados todos os códigos referentes a estrutura das páginas web/HTML e como as funcionalidades definidas na seção 3.1.4 são implementadas dentro dessas.

3.1.6.1. layout.html

O código presente no arquivo **layout.html** é a base de todos os demais códigos HTML, uma vez que nele é importado o framework Bootstrap, são estruturados os cabeçalhos do aplicativo, com as opções disponíveis e uma breve explicação do projeto em questão. O objetivo da escrita de um código que sirva de base é o reaproveitamento, assim, ao invés de escrever sempre as mesmas linhas para todas as páginas que se deseja utilizar, é possível apenas importar o layout/código base para a página e alterar o que é desejado. Abaixo, nas Figuras 42 a 45, é possível verificar o código na íntegra e exemplificações do código.

Figura 42: Código base do arquivo layout.html I

```
<!DOCTYPE html>
<html>
<head>

    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
          integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
          crossorigin="anonymous">

    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='main.css') }}">

    {% if title %}
        <title>CTOS - {{ title }}</title>
    {% else %}
        <title>CTOS</title>
    {% endif %}
</head>
```

Fonte: modificado de SCHAFER, [201-]a

Linhas 1 a 6 e 8 a 10: Códigos de inicialização de um arquivo HTML e utilização do framework Bootstrap, respectivamente.

Linhas 12 a 19: Importação do arquivo CSS e verificação do nome atribuído a página web. Caso nenhum nome tenha sido atribuído a página, o nome padrão, CTOS, foi definido.

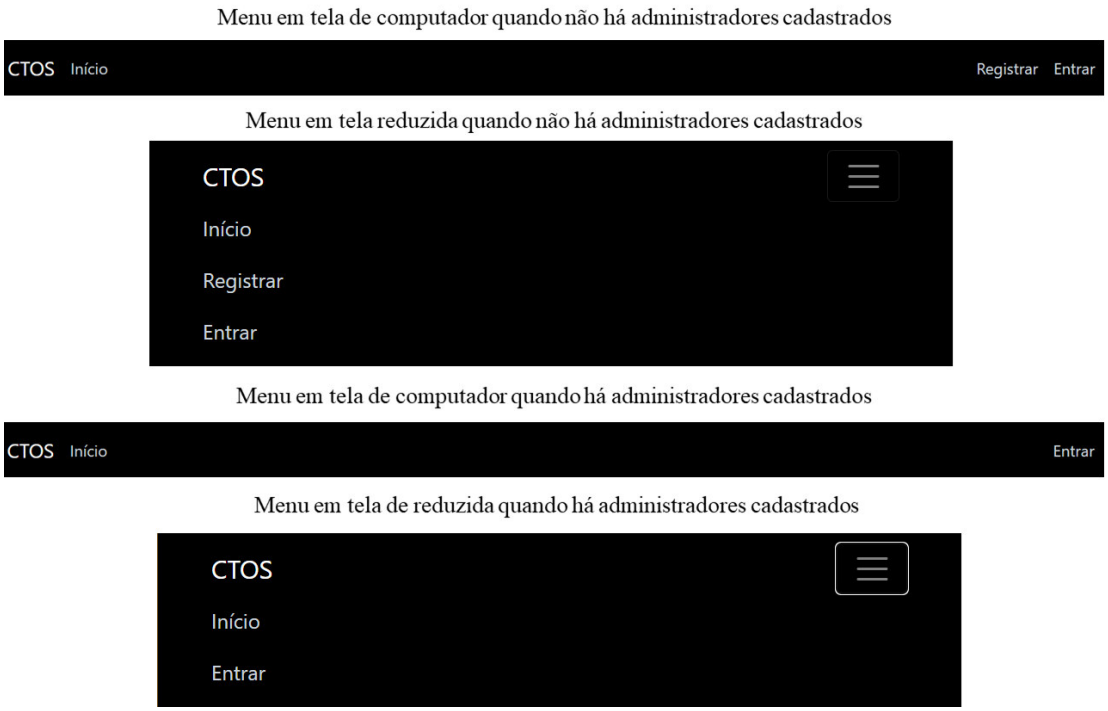
Linhas 22 a 28: Foi definida a cor do menu do aplicativo (preto) a forma com que o menu responderá no caso do aplicativo ser utilizado em dispositivo diferente de um computador (tal resposta do menu é mais conhecida como responsive navbar, do inglês) e o ícone do menu no caso do aplicativo ser utilizado em dispositivos que não sejam um computado.

Figura 43: Código base do arquivo layout.html II

```
20  <body>
21      <header class="site-header">
22        <nav class="navbar navbar-expand-md navbar-dark bg-steel fixed-top">
23          <div class="container">
24            <a class="navbar-brand mr-2" href="/">CTOS</a>
25            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarToggle"
26                    aria-controls="navbarToggle" aria-expanded="false" aria-label="Toggle navigation">
27              <span class="navbar-toggler-icon"></span>
28            </button>
29            <div class="collapse navbar-collapse" id="navbarToggle">
30              <div class="navbar-nav mr-auto">
31                <a class="nav-item nav-link" href="{{ url_for('home') }}">Início</a>
32              </div>
33              <div class="navbar-nav">
34                  {% if current_user.is_authenticated %}
35                      <a class="nav-item nav-link" href="{{ url_for('deletar') }}">Deletar</a>
36                      <a class="nav-item nav-link" href="{{ url_for('show_all') }}">Registro</a>
37                      <a class="nav-item nav-link" href="{{ url_for('account') }}">Usuário</a>
38                      <a class="nav-item nav-link" href="{{ url_for('logout') }}">Sair</a>
39                  {% else %}
40                      {% if clever_function() == 0 %}
41                          <a class="nav-item nav-link" href="{{ url_for('register') }}">Registrar</a>
42                          <a class="nav-item nav-link" href="{{ url_for('login') }}">Entrar</a>
43                      {% else %}
44                          <a class="nav-item nav-link" href="{{ url_for('login') }}">Entrar</a>
45                      {% endif %}
46                  {% endif %}
47              </div>
48            </div>
49          </div>
50        </nav>
51      </header>
```

Fonte: modificado de SCHAFER, [201-]a

Figura 44: Exemplificação do menu responsivo (*responsive navbar)*



Fonte: Autoria Própria

Linhas 29 a 48: São definidas quais as funções estarão presentes no menu. Como visto na Figura 44, caso não haja administradores cadastrados, aparecerá a opção de se registrar como administrador. Agora, caso haja o número total de administradores registrados, haverá somente a opção de fazer *login*/entrar.

Também é possível verificar nas linhas 34 a 38, que caso o administrador esteja *logado*, no lugar de "Entrar" ou "Registrar" haverá as opções "Deletar", "Registro", "Usuário" e "Sair". A opção "Deletar" irá deletar o registro do administrador do banco de dados User, "Registro" direcionará o administrador ao banco de registro de pessoas que foram reconhecidas, a opção "Usuário" direcionará o administrador a página de cadastro de usuários e "Sair" efetuará o *logout* do administrador.

Figura 45: Código base do arquivo layout.html III

```
52      <main role="main" class="container">
53        <div class="row">
54          <div class="col-md-8">
55            {% with messages = get_flashed_messages(with_categories=true) %}
56              {% if messages %}
57                {% for category, message in messages %}
58                  <div class="alert alert-{{ category }}">
59                    {{ message }}
60                  </div>
61                {% endfor %}
62              {% endif %}
63            {% endwith %}
64            {% block content %} {% endblock %}
65          </div>
66          <div class="col-md-4">
67            <div class="content-section">
68            <img src="/static/images/USP.svg" style=" width: 100% \9;" class="img-fluid" alt=" ">
69                  <h4 class='text-center'><b>Universidade de São Paulo</b></h4>
70                  <br>
71                  <p class='text-justify'><b>
72                  Aplicativo apresentado ao Curso de Engenharia Física como parte
73                  do Trabalho de Conclusão para a obtenção do Título de Bacharel em Engenharia Física
74                  </b></p>
75                  <p><b>Aluno: Matheus de Mendonça Chitan</b></p>
76                  <p><b>Orientador: Dr. Carlos Antônio Reis Pereira Baptista</b></p>
77                  <p><b>Escola de Engenharia de Lorena</b></p>
78            </div>
79          </div>
80        </div>
81      </main>
82    </body>
83    </html>
```

Fonte: modificado de SCHAFER, [201-]a

Linhas 52 a 63: código responsável por mostrar mensagens do tipo “Login Efetuado com Sucesso” ou “Login fracassado. Por favor tente outro e-mail”.

Linha 64: responsável por exibir o conteúdo das outras páginas, que serão apresentadas posteriormente.

Linhas 66 a 79: responsável por exibir uma coluna onde é contado sobre o objetivo do projeto.

3.1.6.2. account.html

O código presente no arquivo account.html é responsável por estruturar em uma página web, os formulários e funcionalidades que se deseja utilizar na página do administrador, demonstrados nos códigos acima. Um fato interessante é que todos os códigos presentes nessa e nas próximas seções, serão inseridos dentro da linha 64 do código layout.html, a fim de, mesmo em diferentes páginas, ser possível acessar o menu e ver o objetivo do projeto. Das

Figuras 46 a 50, apresentadas abaixo, é possível verificar os códigos e as suas respectivas explicações.

Figura 46: Código account.html I

```
1   {% extends "layout.html" %}
2   {% block content %}
3   <div class="content-section">
4       <form method="POST" action="">
5           {{ form.hidden_tag() }}
6           <div class="media-body">
7               <div class="article-metadata">
8                   <a class="mr-2" href="#">Detecção Facial</a>
9                   <small class="text-muted">Verificação da captura de
10                      imagens e posicionamento da câmera</small>
11              </div>
12              <button class="btn btn-outline-dark mt-3 mb-3" type="button"
13                      onclick="window.location='{{ url_for( 'deteccao_facial') }}';">
14                      Iniciar Captura
15              </button>
16          </div>
17      </form>
18  </div>
```

Fonte: Autoria Própria

Linhas 1 e 2: Fazendo uso do código presente no arquivo layout.html e inserindo dentro deste código (layout.html) o código do arquivo account.html, respectivamente.

Linhas 3 a 18: Definição de uma seção reservada para a execução do código “deteccao facial”, demonstrado na Figura 35.

Linhas 23 a 39: É definido a estrutura/estética do formulário utilizado para o cadastro do nome de usuários por parte do administrador.

Linhas 40 a 52: É definido a estrutura/estética do formulário utilizado para o cadastro do número atribuído ao usuário por parte do administrador.

Linhas 54 a 56: É estruturado um botão que realizado o cadastro do usuário no banco de dados “Cadastro”, mostrado na Figura 27.

Figura 47: Código account.html II

```
<div class="content-section">
    <form method="POST" action="">
        {{ form.hidden_tag() }}
        <fieldset class="media-body">
            <div class="article-metadata">
                <a class="mr-2" href="#">Cadastro</a>
                <small class="text-muted">Cadastro de pessoas</small>
            </div>
            <div class="form-group">
                {{ form.nome_usuario.label(class="form-control-label") }}
                {% if form.nome_usuario.errors %}
                    {{ form.nome_usuario(class="form-control is-invalid") }}
                    <div class="invalid-feedback">
                        {% for error in form.nome_usuario.errors %}
                            <span>{{ error }}</span>
                        {% endfor %}
                    </div>
                {% else %}
                    {{ form.nome_usuario(class="form-control") }}
                {% endif %}
            </div>
```

Fonte: Autoria Própria

Figura 48: Código account.html III

```
            <div class="form-group">
                {{ form.numero_usuario.label(class="form-control-label") }}
                {% if form.numero_usuario.errors %}
                    {{ form.numero_usuario(class="form-control is-invalid") }}
                    <div class="invalid-feedback">
                        {% for error in form.numero_usuario.errors %}
                            <span>{{ error }}</span>
                        {% endfor %}
                    </div>
                {% else %}
                    {{ form.numero_usuario(class="form-control") }}
                {% endif %}
            </div>
        </fieldset>
        <div class="form-group">
            {{ form.submit(class="btn btn-outline-dark") }}
        </div>
    </form>
</div>
```

Fonte: Autoria Própria

Figura 49: Código account.html IV

```
59  <div class="content-section">
60      <form method="POST" action="">
61          {{ form.hidden_tag() }}
62          <div class="media-body">
63              <div class="article-metadata">
64                  <a class="mr-2" href="#">Banco de Dados</a>
65                  <small class="text-muted">Captura de fotos para o banco de dados</small>
66              </div>
67              <button class="btn btn-outline-dark mt-3 mb-3" type="button"
68                      onclick="window.location='{{ url_for( 'banco_dados') }}';">
69                      Iniciar Captura
70              </button>
71          </div>
72      </form>
73  </div>
```

Fonte: Autoria Própria

Linhas 59 a 73: Estruturação de um botão responsável por executar a função "banco_dados", mostrada na Figura 35.

Figura 50: Código account.html V

```
74  <div class="content-section">
75      <form method="POST" action="">
76          {{ form.hidden_tag() }}
77          <div class="media-body">
78              <div class="article-metadata">
79                  <a class="mr-2" href="#">Treinamento</a>
80                  <small class="text-muted">Treinamento do Algoritmo de Reconhecimento Facial</small>
81              </div>
82              <button class="btn btn-outline-dark mt-3 mb-3" type="button"
83                      onclick="window.location.href='{{ url_for( 'treinamento_imagens') }}';">
84                      Iniciar Treinamento
85              </button>
86          </div>
87      </form>
88  </div>
89  {% endblock content %}
```

Fonte: Autoria Própria

Linhas 74 a 89: Estruturação de um botão responsável por executar o treinamento do algoritmo de reconhecimento facial, mostrado na Figura 36.

3.1.6.3. tabelas.html

No arquivo tabelas.html será encontrado apenas uma parte dos códigos referentes ao registro de pessoas que foram reconhecidas pelo aplicativo. Nesta parte do código serão exibidos as tabelas gerais e os botões para a escolha do filtro, que será tratado mais à frente. Os códigos e suas respectivas explicações são exibidos da Figuras 51 a 53.

Figura 51: Código tabelas.html I

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Registro</title>
6      <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='tabelas.css') }}">
7  </head>
```

Fonte: Autoria Própria

Linhas 1 a 7: Inicialização do arquivo HTML, definição do título da página (Registro, no caso) e importação do arquivo CSS, responsável pelo estilo/design da tabela.

Figura 52: Código tabelas.html II

```
9   <body>
10      <form action="/select" method="GET">
11          <div class="container">
12              <div class="row">
13                  <div class="col-sm"></div>
14                  <div class="col-sm">
15                      <select name="nome_usuario" placeholder="Selecionar Nome">
16                          <option value="Nulo">Selecione um nome...</option>
17                          {% for usuario in numero_usuario %}
18                              <option value="{{usuario.nome_usuario}}">{{ usuario.nome_usuario }}</option>
19                          {% endfor %}
20                      </select>
21                      <select name="data_usuario">
22                          <option value="Nulo">Selecione uma data...</option>
23                          {% for data in data %}
24                              <option value="{{data.data_registro_dia}}">{{ data.data_registro_dia }}</option>
25                          {% endfor %}
26                      </select>
27                      <input type="submit" value="Filtrar">
28                  </div>
29                  <div class="col-sm"></div>
30              </div>
31          </div>
32      </form>
```

Fonte: Autoria Própria

Linhas 10 a 32: Estruturação dos botões responsáveis por permitir que o administrador selecione, dentre todas as pessoas que foram reconhecidas e as respectivas datas em que foram reconhecidas, a pessoa e data desejada.

Linhas 34 a 62: Estruturação da tabela que exibirá os nomes, datas e horas das pessoas que foram reconhecidas pelo aplicativo.

Figura 53: Código tabelas.html III

```
    <section>
      <h1 style="color:white;">Registro de Dados</h1>
      <div class="tbl-header">
          <table cellpadding="0" cellspacing="0" border="0">
              <thead>
              <tr>
                  <th>Nome</th>
                  <th>Data</th>
                  <th>Hora</th>
              </tr>
              </thead>
          </table>
      </div>
      <div class="tbl-content">
          <table cellpadding="0" cellspacing="0" border="0">
              <tbody>
              {% for registro in registro %}
              <tr class="active-row">
                  <td>{{ registro.nome_registro }}</td>
                  <td>{{ registro.data_registro_dia }}</td>
                  <td>{{ registro.data_registro_hora }}</td>
              </tr>
              {% endfor %}
              </tbody>
        </table>
      </div>
    </section>
</body>
</html>
```

Fonte: Autoria Própria

3.1.6.4. filtro.html

Neste arquivo serão encontrados os demais códigos referentes ao registro das pessoas que foram reconhecidas. Mais especificamente, no arquivo filtro.html será discutido a página de filtro, ou seja, página posterior à escolha de determinada pessoa ou data ou, até mesmo, ambas na página tabelas.html. Nas Figuras 54 e 55, mostradas abaixo, é possível verificar o código na íntegra.

Figura 54: Código filtro.html I

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Filtro</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='tabelas.css') }}">
</head>
```

Fonte: Autoria própria

Linhas 1 a 7: Da mesma forma que para os demais códigos HTML, o código é inicializado, é definido um título para a página (no caso, Filtro) e é importado o arquivo CSS (tabelas.css), responsável pelo design e estilo da tabela.

Figura 55: Código filtro.html II

```
8   <body>
9       <section>
10          <h1>Registro de Dados</h1>
11          <div class="tbl-header">
12              <table cellpadding="0" cellspacing="0" border="0">
13                  <thead>
14                  <tr>
15                      <th>Nome</th>
16                      <th>Data</th>
17                      <th>Hora</th>
18                  </tr>
19                  </thead>
20              </table>
21          </div>
22          <div class="tbl-content">
23              <table cellpadding="0" cellspacing="0" border="0">
24                  <tbody>
25                  {% for registro in registro %}
26                  <tr class="active-row">
27                      <td>{{ registro.nome_registro }}</td>
28                      <td>{{ registro.data_registro_dia }}</td>
29                      <td>{{ registro.data_registro_hora }}</td>
30                  </tr>
31                  {% endfor %}
32                  </tbody>
33            </table>
34          </div>
35      </section>
36  </body>
37  </html>
```

Fonte: Autoria Própria

Linhas 8 a 37: Dependendo do que foi selecionado nos botões na página tabelas.html, serão exibidos diferentes resultado na tabela da página filtro.html. Para o caso de o administrador selecionar apenas o nome de uma pessoa, na coluna “Nome” será exibido apenas o nome da pessoa e nas colunas “Data” e “Hora” serão exibidas as diferentes datas e horas na qual a pessoa escolhida foi reconhecida. No caso de ser selecionado apenas uma data, serão exibidos os diferentes nomes e horas de todas as pessoas que foram reconhecidas no dia especificado. E por último, no caso de ser especificado o nome e data de uma pessoa, serão exibidos os diferentes horários em que a pessoas selecionada foi reconhecida no dia selecionado.

3.1.6.5. home.html

No arquivo home.html é possível encontrar os códigos relacionados a página inicial do aplicativo. Tanto os administradores quanto os usuários têm acesso a está página e nela está contido um texto explicativo do autor do projeto, o objetivo do mesmo e o algoritmo de reconhecimento facial.

Tanto aqueles que forem cadastrados quanto os que não forem, poderão utilizar o reconhecimento facial. Entretanto, aqueles que não estiverem cadastrados na tabela “Cadastro”

do banco de dados serão identificados como “unknown”, ou “desconhecido” em português. Na Figura 56, abaixo, é possível verificar o código do arquivo home.html na íntegra:

Figura 56: Código home.html

```
1   {% extends "layout.html" %}
2   {% block content %}
3           <article class="media content-section">
4             <div class="media-body">
5               <div class="article-metadata">
6                 <a class="mr-2" href="#">Bem-Vindo ao CTOS</a>
7                 <small class="text-muted">aplicativo de reconhecimento facial</small>
8               </div>
9               <p class="article-content" align="justify">Meu nome é Matheus de Mendonça Chitan,
10                 graduando de Engenharia Física da Universidade de São Paulo. Este projeto tem
11                 como objetivo, demonstrar na prática o funcionamento do reconhecimento facial
12                 como um sistema de segurança para a Escola de Engenharia de Lorena.</p>
13            </div>
14          </article>
15          <article class="media content-section">
16            <div class="media-body">
17              <div class="article-metadata">
18                <a class="mr-2" href="#">Reconhecimento Facial</a>
19                <small class="text-muted">Iniar reconhecimento?</small>
20              </div>
21                 <button class="btn btn-outline-dark mt-3 mb-3" type="button"
22                         onclick="window.location='{{ url_for( 'video_feed') }}';">Iniciar Captura
23                 </button>
24              </div>
25          </article>
26  {% endblock content %}
```

Fonte: Autoria Própria

Linha 1: Utilização do arquivo layout.html, a fim de gerar o mesmo layout das demais página do aplicativo (utilização de um padrão para todas as páginas)

Linhas 3 a 14: Definição de um bloco de texto explicando quem é o criador do aplicativo e o objetivo do mesmo.

Linhas 15 a 25: Estruturação de um botão responsável por executar a função “*vídeo_feed*”, mostrada na Figura 35.

3.1.6.6. register.html

No arquivo register.html é possível encontrar todos os códigos referentes a página de cadastro dos administradores. Tal página, como foi dito anteriormente, não ficará disponível para todos os usuários do aplicativo, mas sim, apenas para os *n* primeiros usuários, sendo *n* o número de administradores desejados. Nas figuras 57 a 59, mostradas abaixo, é possível verificar o código do arquivo na íntegra, seguido por suas devidas explicações.

Figura 57: Código register.html I

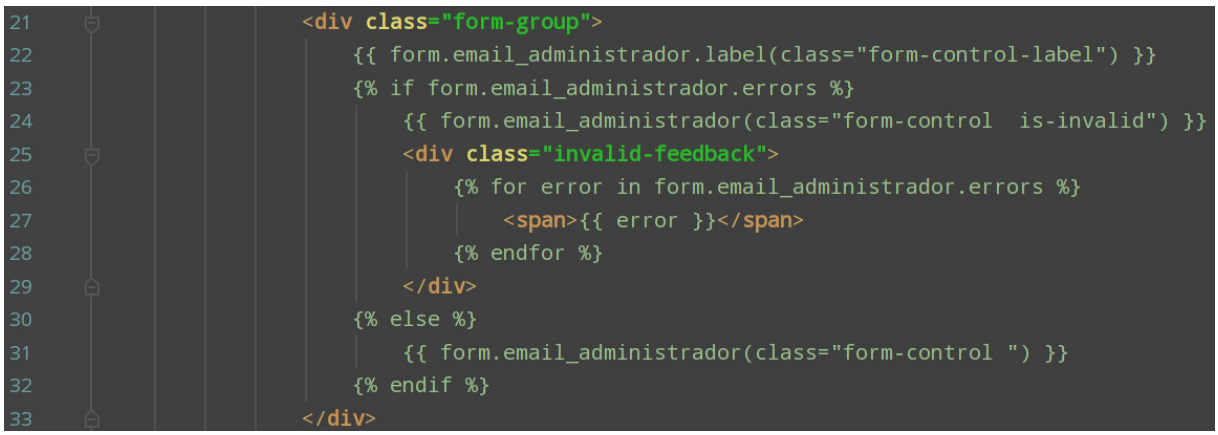
```
1  {% extends "layout.html" %}
2  {% block content %}
3      <div class="content-section">
4          <form method="POST" action="">
5              {{ form.hidden_tag() }}
6              <fieldset class="form-group">
7                  <legend class="border-bottom mb-4">Registrar</legend>
8                  <div class="form-group">
9                      {{ form.nome_administrador.label(class="form-control-label") }}
10                     {% if form.nome_administrador.errors %}
11                         {{ form.nome_administrador(class="form-control  is-invalid") }}
12                         <div class="invalid-feedback">
13                             {% for error in form.nome_administrador.errors %}
14                                 <span>{{ error }}</span>
15                             {% endfor %}
16                         </div>
17                     {% else %}
18                         {{ form.nome_administrador(class="form-control ") }}
19                     {% endif %}
20                 </div>
```

Fonte: Autoria Própria

Linha 1: Da mesma forma que foi explicado anteriormente, é implementado o código layout.html a fim das diferentes páginas do aplicativo possuírem o mesmo layout/estilo.

Linha 8 a 20: Estruturação do campo “nome_administrador” do formulário “RegistrationForm” na página web. Neste Campo o administrador pode definir um nome pelo qual será reconhecido.

Figura 58: Código register.html II



```
21                 <div class="form-group">
22                     {{ form.email_administrador.label(class="form-control-label") }}
23                     {% if form.email_administrador.errors %}
24                         {{ form.email_administrador(class="form-control  is-invalid") }}
25                         <div class="invalid-feedback">
26                             {% for error in form.email_administrador.errors %}
27                                 <span>{{ error }}</span>
28                             {% endfor %}
29                         </div>
30                     {% else %}
31                         {{ form.email_administrador(class="form-control ") }}
32                     {% endif %}
33                 </div>
```

Fonte: autoria Própria

Linhas 21 a 33: Estruturação do campo “email_administrador” do formulário “RegistrationForm” na página web. Neste campo o administrador irá inserir um e-mail pelo qual também será reconhecido.

Figura 59: Código register.html III

```
34                  <div class="form-group">
35                      {{ form.senha_administrador.label(class="form-control-label") }}
36                      {% if form.senha_administrador.errors %}
37                          {{ form.senha_administrador(class="form-control  is-invalid") }}
38                          <div class="invalid-feedback">
39                              {% for error in form.senha_administrador.errors %}
40                                  <span>{{ error }}</span>
41                              {% endfor %}
42                          </div>
43                      {% else %}
44                          {{ form.senha_administrador(class="form-control ") }}
45                      {% endif %}
46                  </div>
```

Fonte: Autoria Própria

Linhas 34 a 46: Estruturação do campo “senha_administrador” do formulário “RegistrationForm” na página web. Neste Campo o administrador pode definir uma senha com a qual efetuará o login posteriormente.

Figura 60: Código register.html IV

```
47                  <div class="form-group">
48                      {{ form.confirmar_senha.label(class="form-control-label") }}
49                      {% if form.confirmar_senha.errors %}
50                          {{ form.confirmar_senha(class="form-control  is-invalid") }}
51                          <div class="invalid-feedback">
52                              {% for error in form.confirmar_senha.errors %}
53                                  <span>{{ error }}</span>
54                              {% endfor %}
55                          </div>
56                      {% else %}
57                          {{ form.confirmar_senha(class="form-control ") }}
58                      {% endif %}
59                  </div>
60              </fieldset>
61              <div class="form-group">
62                  {{ form.submit(class="btn btn-outline-dark") }}
63              </div>
64          </form>
65      </div>
66  {% endblock content %}
```

Fonte: Autoria Própria

Linhas 47 a 59: Estruturação do campo “confirmar_senha” do formulário “RegistrationForm” na página web. Neste campo o administrador deve confirmar a senha fornecida no campo “senha_administrador” (caso as senhas sejam diferentes, o sistema irá acusar um erro).

Linhas 61 a 63: Definição de um botão responsável por enviar os campos mostrados nas Figuras 57 a 60 para o banco de dados User.

3.1.6.7. login.html

Neste arquivo é possível encontrar todos os códigos referentes a estruturação da página web de login, a qual estará apenas disponível após o cadastramento de todos os administrados. Das Figuras 61 a 63 é possível verificar o código na íntegra e suas respectivas explicações.

Figura 61: Código login.html I

```
1   {% extends "layout.html" %}
2   {% block content %}
3       <div class="content-section">
4           <form method="POST" action="">
5               {{ form.hidden_tag() }}
6               <fieldset class="form-group">
7                   <legend class="border-bottom mb-4">Entrar</legend>
8                   <div class="form-group">
9                       {{ form.email_administrador.label(class="form-control-label") }}
10                      {% if form.email_administrador.errors %}
11                          {{ form.email_administrador(class="form-control is-invalid") }}
12                          <div class="invalid-feedback">
13                              {% for error in form.email_administrador.errors %}
14                                  <span>{{ error }}</span>
15                              {% endfor %}
16                          </div>
17                      {% else %}
18                          {{ form.email_administrador(class="form-control") }}
19                      {% endif %}
20                  </div>
```

Fonte: Autoria Própria

Linha 1: Implementação do código layout.html a fim das diferentes páginas do aplicativo possuírem o mesmo layout/estilo.

Linhas 8 a 20: Estruturação do campo que receberá o e-mail do administrador.

Figura 62: Código login.html II

```
21                  <div class="form-group">
22                      {{ form.senha_administrador.label(class="form-control-label") }}
23                      {% if form.senha_administrador.errors %}
24                          {{ form.senha_administrador(class="form-control is-invalid") }}
25                          <div class="invalid-feedback">
26                              {% for error in form.senha_administrador.errors %}
27                                  <span>{{ error }}</span>
28                              {% endfor %}
29                          </div>
30                      {% else %}
31                          {{ form.senha_administrador(class="form-control") }}
32                      {% endif %}
33                  </div>
```

Fonte: Autoria Própria

Linhas 21 a 33: Estruturação do campo que receberá a senha cadastrada pelo administrador.

Figura 63: Código login.html III

```
34              </fieldset>
35              <div class="form-group">
36                  {{ form.submit(class="btn btn-outline-dark") }}
37              </div>
38              <small class="text-muted ml-2">
39                  <a href="{{url_for('reset_request')}}">Esqueceu a senha?</a>
40              </small>
41          </form>
42      </div>
43  {% endblock content %}
```

Fonte: Autoria Própria

Linhas 35 a 37: Estruturação de um botão responsável por verificar se o e-mail e senha informados estão realmente presentes na tabela “User” do banco de dados. Caso positivo, o login é efetuado. Caso negativo, é acusado um erro.

Linhas 38 a 40: Disponibilização de um link para o recadastramento da senha, caso necessário.

3.1.6.8. reset_request.html

No arquivo reset_request.html é possível encontrar os códigos referentes a criação da página web responsável por enviar o e-mail de recadastro de senha do administrador. Na Figura 62 é possível verificar o código na íntegra.

Linha 1: Utilização do arquivo layout.html para a criação de design/estilos iguais para as páginas do aplicativo.

Linha 6 a 21: Estruturação de um campo de preenchimento, no qual o administrador que esqueceu a senha deve inserir o seu e-mail de cadastro. Caso o e-mail informado pelo administrador esteja contido na tabela “User”, o e-mail poderá ser enviado. Caso o e-mail informado no campo não esteja contido na tabela, será acusado um erro e o e-mail não poderá ser enviado.

Linhas 22 a 24: Estruturação do botão responsável por enviar o e-mail para o recadastramento da senha.

Figura 64: Código reset_request.html

```
{% extends "layout.html" %}
{% block content %}
    <div class="content-section">
        <form method="POST" action="">
            {{ form.hidden_tag() }}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Reset Password</legend>
                <div class="form-group">
                    {{ form.email_reset.label(class="form-control-label") }}
                    {% if form.email_reset.errors %}
                        {{ form.email_reset(class="form-control is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.email_reset.errors %}
                                <span>{{ error }}</span>
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.email_reset(class="form-control") }}
                    {% endif %}
                </div>
            </fieldset>
            <div class="form-group">
                {{ form.submit(class="btn btn-outline-dark") }}
            </div>
        </form>
    </div>
{% endblock content %}
```

Fonte: Autoria Própria

3.1.6.9. reset_token.html

No arquivo reset_token.html é possível encontrar os códigos referentes a estruturação da página web responsável pelo recadastramento da nova senha do administrador. Nas Figuras 65 e 66, é possível verificar o código na íntegra.

Linhas 7 a 20: Estruturação de um campo no qual os administradores podem inserir a nova senha desejada.

Linhas 21 a 33: Estruturação do campo de confirmação da nova senha, no qual os administradores devem inserir a mesma senha que foi escrita/especificada no campo da nova senha. Caso a senha e a confirmação da senha sejam diferentes, será acusado um erro.

Linhas 35 a 37: Estruturação do botão responsável pela troca da senha antiga pela nova senha na tabela “User”.

Figura 65: Código reset-token.html I

```
{% extends "layout.html" %}
{% block content %}
    <div class="content-section">
        <form method="POST" action="">
            {{ form.hidden_tag() }}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Reset Password</legend>
                <div class="form-group">
                    {{ form.nova_senha.label(class="form-control-label") }}
                    {% if form.nova_senha.errors %}
                        {{ form.nova_senha(class="form-control is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.nova_senha.errors %}
                                <span>{{ error }}</span>
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.nova_senha(class="form-control") }}
                    {% endif %}
                </div>
```

Fonte: Autoria Própria

Figura 66: Código reset_token.html II

```
                <div class="form-group">
                    {{ form.confirmar_nova_senha.label(class="form-control-label") }}
                    {% if form.confirmar_nova_senha.errors %}
                        {{ form.confirmar_nova_senha(class="form-control is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.confirmar_nova_senha.errors %}
                                <span>{{ error }}</span>
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.confirmar_nova_senha(class="form-control") }}
                    {% endif %}
                </div>
            </fieldset>
            <div class="form-group">
                {{ form.submit(class="btn btn-outline-dark") }}
            </div>
        </form>
    </div>
{% endblock content %}
```

Fonte: Autoria Própria

3.1.7. Static

Nesta seção serão tratados os códigos relacionados ao design da página. Enquanto os códigos HTML são responsáveis pela estruturação da página em si, é possível fazer uso dos códigos CSS para a criação de melhores design e de uma maneira mais fácil.

3.1.7.1. main.css

Neste arquivo e possível encontrar os atributos utilizados para a criação dos estilos das páginas web, que não são derivados do framework Bootstrap. Nas Figuras 67 e 68 é possível verificar os atributos na íntegra.

Figura 67: Atributos do código main.css I

```
1   body {
2     background: #fafafa;
3     margin-top: 5rem;
4   }
5
6   .bg-steel {
7     background-color: #000000;
8   }
9
10  .site-header .navbar-nav .nav-link {
11    color: #cbd5db;
12    font-size: 100%;
13  }
14
15  .site-header .navbar-nav .nav-link:hover {
16    color: #ffffff;
17  }
```

Fonte: Autoria Própria

Linhas 1 a 4: Definição da cor da página web (código hexadecimal #fafafa – espécie de branco) e da margem superior que há entre o início da página e o início dos conteúdos.

Linhas 6 a 8: Definição da cor #000000 (preto) para o menu responsivo, apresentado na Figura 43.

Linhas 10 a 12: Definição da cor #cbd5db (espécie de branco com cinza) para os itens apresentado no menu responsivo, apresentado na Figura 44.

Linhas 15 a 17: Estruturação da mudança da cor #cbd5db para #fffff (branco) quando o mouse passar por cima de um dos itens do menu responsivo.

Figura 68: Atributos do código main.css II

```
19  .content-section {
20    background: #fffff;
21    padding: 10px 20px;
22    border: 1px solid #dddddd;
23    border-radius: 3px;
24    margin-bottom: 20px;
25  }
26
27  .article-metadata {
28    padding-bottom: 1px;
29    margin-bottom: 4px;
30    border-bottom: 1px solid #e3e3e3
31  }
32
33  .article-metadata a:hover {
34    color: #000000;
35    text-decoration: none;
36  }
```

Fonte: Autoria própria

Linhas 19 a 31: Definição dos atributos dos blocos de textos e formulário, como: cor de fundo (*background* – #fffff – branco), distanciamento do texto/botões até as bordas (*padding*), cor e raio das bordas (*border* e *border-radius*) e distanciamento para a bloco/formulário abaixo (*margin-bottom*).

Linhass 33 a 36: Estruturação da mudança de cor, azul para preto, do título do bloco de texto ou formulário quando o mouse passar por cima.

3.1.7.2. tabelas.css

Neste arquivo é possível encontrar os atributos utilizados para a estruturação das tabelas de registro de reconhecimento, apresentadas na seção 3.1.6.3. Nas Figuras 69, 70 e 71 é possível verificar os atributos na íntegra.

Figura 69: Atributos do código tabelas.css I

```
1  h1{
2    font-size: 35px;
3    color: #FFFFFF;
4    text-transform: uppercase;
5    text-align: center;
6    margin-bottom: 15px;
7    font-family: "Times New Roman", Times, serif;
8  }
```

Fonte: Autoria própria

Linhas 1 a 8: Definição do tamanho (35px), cor (#FFFFFF – branco), formato (maiúsculo), posição (centralizado), margem para objeto abaixo (15px) e fonte (Times New Roman) do título que aparecerá quando o administrador abrir a página de registro.

Figura 70: Atributos do código tabelas.css II

```
10  table{
11    width:100%;
12    table-layout: fixed;
13    font-family: "Times New Roman", Times, serif;
14  }
15
16  .tbl-header{
17    background-color: rgba(255,255,255,0.3);
18    font-family: "Times New Roman", Times, serif;
19   }
20
21  .tbl-content{
22    height:300px;
23    overflow-x:auto;
24    margin-top: 0px;
25    border: 1px solid rgba(255,255,255,0.3);
26    font-family: "Times New Roman", Times, serif;
27  }
```

Fonte: Autoria Própria

Linhas 10 a 14: Definição da largura da tabela, no caso 100% da largura da página, definição da divisão entre as colunas (divisão igualitária – *fixed*) e fonte que será utilizada, Times New Roman.

Linhas 16 a 19: Definição da cor do cabeçalho de registro (cinza claro) e fonte utilizada, Times New Roman.

Linhas 21 a 27: Definição da altura (height), preenchimento (overflow-x), distância até o cabeçalho (margin-top), borda (border) e fonte (font-family) da tabela que irá exibir os dados. Cabe ressaltar que o atributo preenchimento diz respeito a quantidade de linhas que é possível exibir dentro da altura estabelecida. Assim, ao fazer uso do **auto** no atributo preenchimento, o número de linhas exibidas, nunca ultrapassará os limites de altura da tabela.

Linhas 29 a 36: Definição das margens, posição (centralizado), tamanho (14px), cor (#FFFFFF – branco), formato (maiúsculo) e fonte (Times New Roman) das palavras que estarão no cabeçalho da tabela, no caso “Nome”, “Data” e “Hora”.

Linhas 38 a 45: Definição da margem, posição (centralizado), tamanho (14px), cor (#FFFFFF – branco), borda inferior (rgba(255, 255, 255, 0.1 - cinza escuro)) e fonte (Times New Roman).

Figura 71: Atributos do código tabelas.css III

```
29    th{
30      padding: 20px 15px;
31      text-align: center;
32      font-size: 14px;
33      color: #FFFFFF;
34      text-transform: uppercase;
35      font-family: "Times New Roman", Times, serif;
36    }
37
38    td{
39      padding: 15px;
40      text-align: center;
41      font-size: 14px;
42      color: #FFFFFF;
43      border-bottom: solid 1px rgba(255,255,255,0.1);
44      font-family: "Times New Roman", Times, serif;
45    }
```

Fonte: Autoria Própria

3.1.8. Images

Na pasta “Images” localizada dentro do diretório “Static” é possível encontrar todas as imagens utilizadas na criação do aplicativo. No caso em específico, a única imagem utilizada foi a do logo da Universidade de São Paulo, “USP.svg”, o qual é presente na grande maioria das páginas do aplicativo.

3.2. Dataset

No diretório “Dataset” é possível encontrar todas as imagens dos usuários que serão utilizadas para o treinamento do algoritmo de reconhecimento. As imagens, como mencionado anteriormente na seção 3.1.4., são armazenadas da seguinte forma: **User.número.contador**, O nome “User” é utilizado apenas para identificar que as imagens são referentes aos usuários do aplicativo, enquanto o “número” é referente ao número atribuído a pessoa pelo administrador no formulário “CadastroForm” e o “contador” é referente a número da imagem dentre as diversas imagens que foram tiradas, como 3ª imagem de 40 que foram tiradas.

3.3. Trainer

No diretório “Trainer” é possível encontrar o arquivo “trainer.yml”, no qual são armazenados os histogramas gerados pelo algoritmo descrito na seção 2.4.4.3. Juntamente com os histogramas é gerado um índice ao fim do arquivo identificando qual histograma é de qual

pessoa. Nesse momento que o “número” estipulado pelo administrador é importante, pois cada histograma será identificado pelo número atribuído.

3.4. run,py

Neste arquivo é possível encontrar o código responsável por executar o aplicativo e todas as demais funcionalidades. O código pode ser verificado na Figura 72.

Figura 72: Código run.py

```
1   from flaskblog import app
2
3   if __name__ == '__main__':
4       app.run(debug=True)
5
```

Fonte: Autoria Própria

3.5. Aplicativo

Desta subseção em diante, serão exibidas todas as páginas que compõe o aplicativo, como também, o sistema de detecção facial e o sistema de reconhecimento facial.

3.5.1. Páginas

3.5.1.1. home.html

Ao inicializar o aplicativo, a primeira página exibida será a página web “home.html”. Todos os usuários têm acesso a ela e, como dito anteriormente, até que todos os administradores não efetuem o devido cadastro, a opção de se cadastrar como um administrador será exibida a qualquer um que utilizar o aplicativo. Entretanto, após o cadastro dos administradores, a opção de cadastro será substituída pela opção de login apenas.

Nela também é possível encontrar um breve resumo do aplicativo, sua finalidade, motivo do desenvolvimento e a função de reconhecimento fácil, a qual será mostrada mais à frente. Nas Figuras 73 e 74, é possível verificar as duas variações da página “home.html”.

Figura 73: Página “home.html” sem administradores cadastrados

Fonte: Autoria Própria

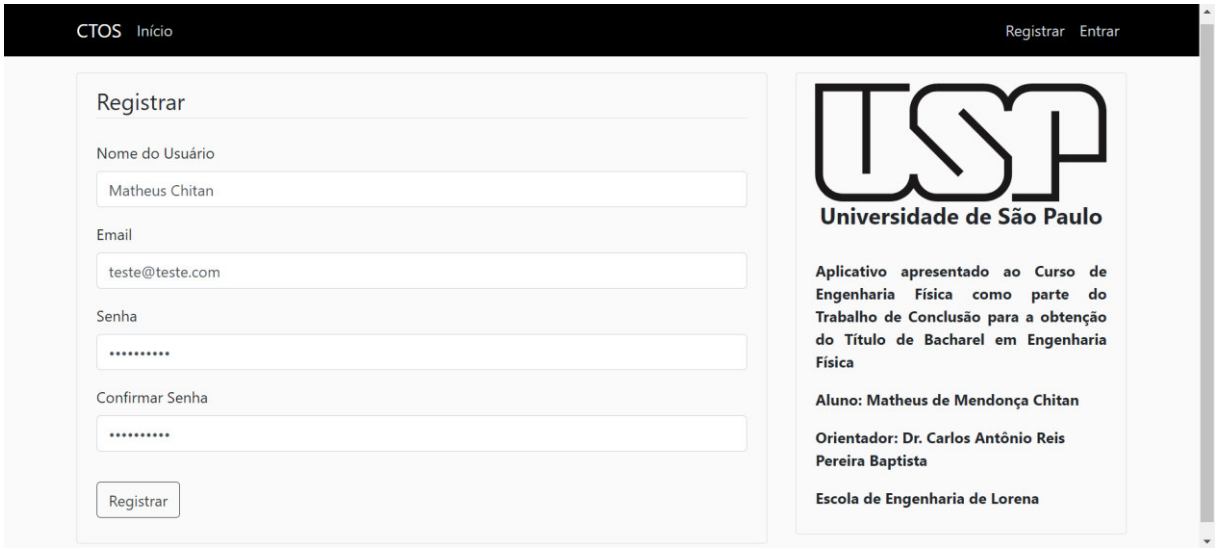
Figura 74: Página “home.html” com administradores cadastrados

Fonte: Autoria Própria

3.5.1.2. register.html

Na página register.html é possível encontrar o formulário de cadastro do administrador. Nela o administrador deve inserir um nome, e-mail, senha e confirmação de senha. Na Figura 75 é possível encontrar um exemplo da página de cadastramento de administradores.

Figura 75: Página “register.html” para o cadastramento de administradores

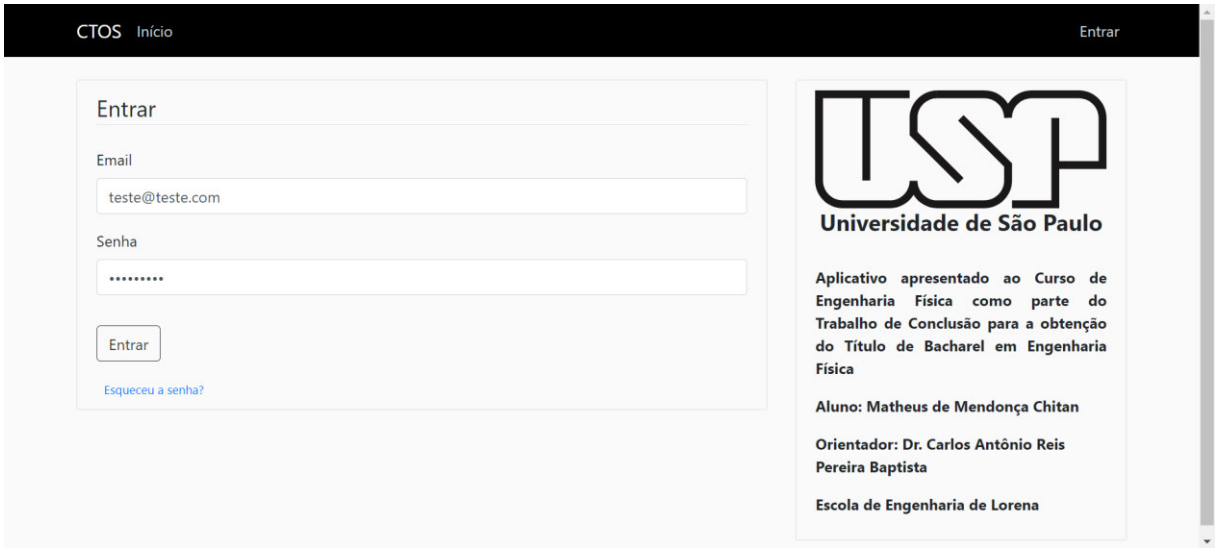


Fonte: Autoria Própria

3.5.1.3. login.html

Na página login.html será encontrado o formulário para o administrador efetuar login no aplicativo. Com o login efetuado, o administrador terá acesso as demais funcionalidades do aplicativo, como cadastramento de pessoa no banco de dados, treinamento do algoritmo de reconhecimento e acesso ao registro de pessoas que foram detectadas. Na figura 76 é possível verificar a página de login.

Figura 76: Página de login de administradores

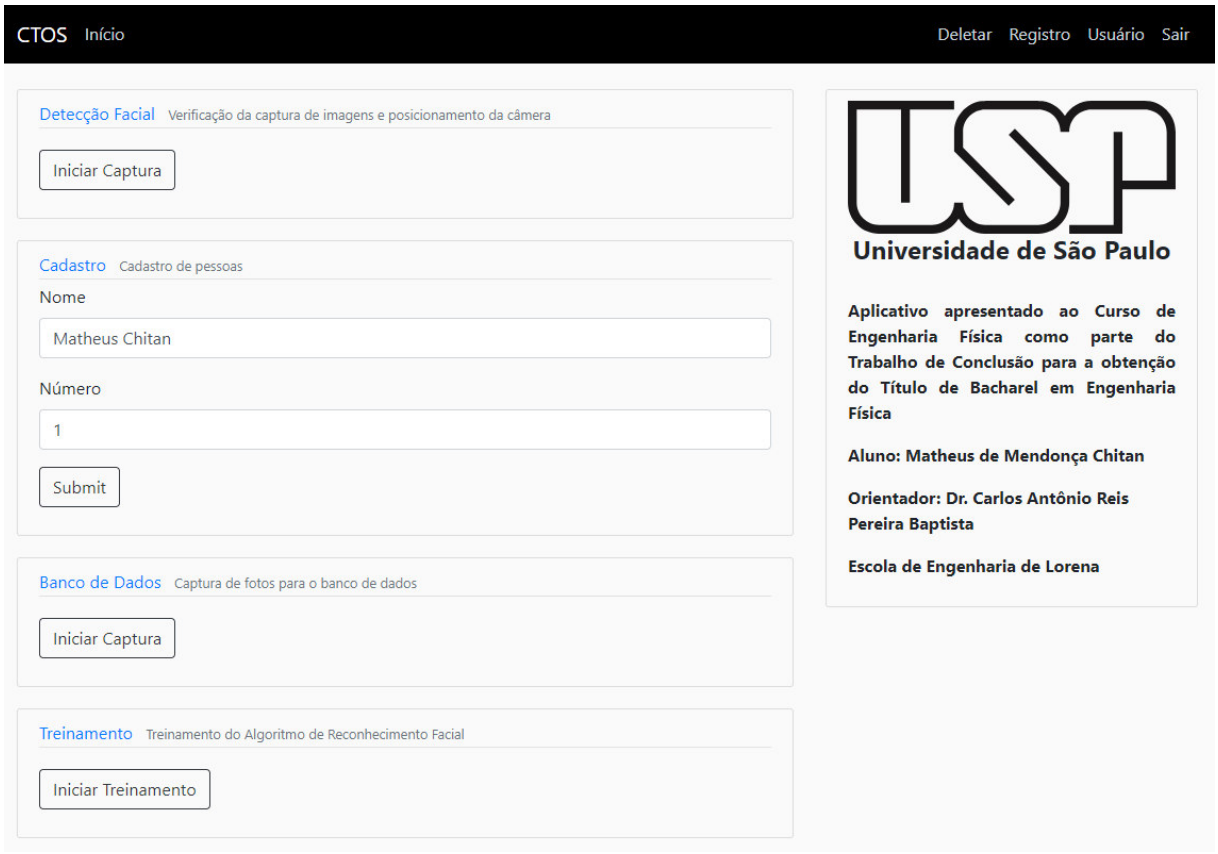


Fonte: Autoria Própria

3.5.1.4. account.html

Na página “account.html” é possível encontrar a funcionalidade de detecção facial (para ajuste da câmera), a do cadastro de pessoas no banco de dados (o administrador deve fornecer um nome e número para cada pessoa), a captura de imagens para o treinamento do algoritmo de reconhecimento e o próprio treinamento do reconhecimento. Na Figura 77, é possível verificar a página “account.html”.

Figura 77: Página “account.html” que estará disponível a todos os administradores



Fonte: Autoria Própria

3.5.1.5. tabelas.html e filtro.html

Na página “tabelas.html” é possível encontrar os registros de pessoas que foram reconhecidas. Neste registro consta o nome, dia e hora em que a pessoa foi reconhecida. Também é possível encontrar os dois filtros que serão utilizados na página “filtro.html”, para selecionar, dentre o banco de dados, uma pessoa ou dia específico. Nas Figuras 78 e 79, mostradas abaixo, é possível verificar o funcionamento da página “tabelas.html”, com dois nomes “Matheus” e “Christopher”, e da página “filtro.html” com apenas o nome “Matheus”.

Figura 78: Página “tabelas.html”

Fonte: Autoria Própria

Figura 79: Página “filtro.html” com apenas o nome “Matheus” selecionado

Fonte: Autoria Própria

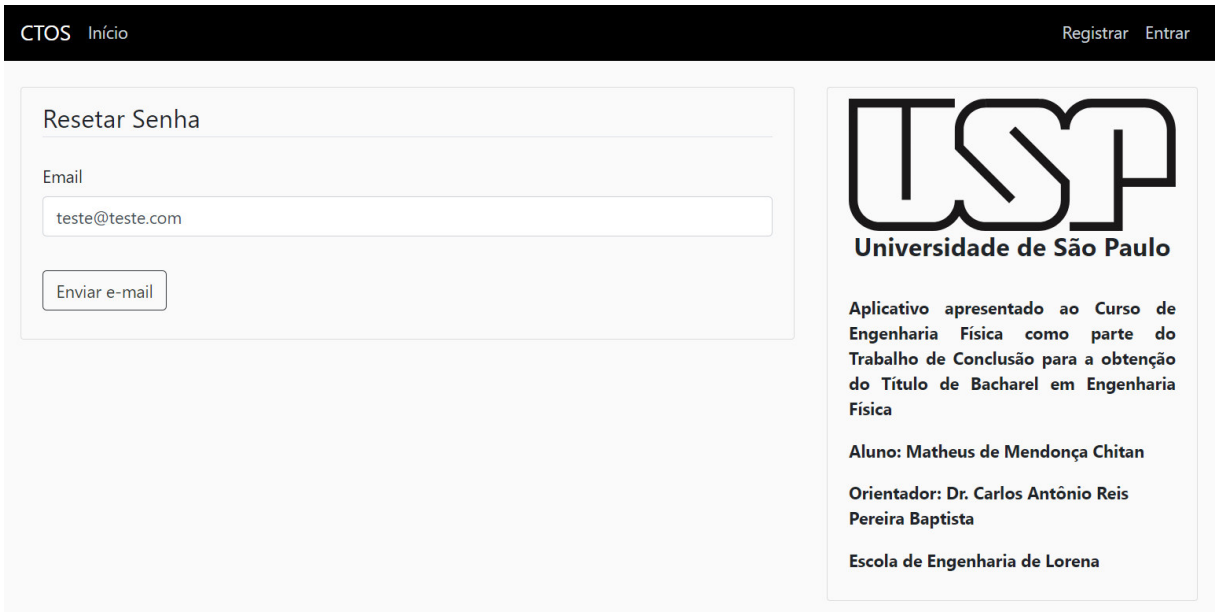
3.5.1.6. reset_request.html e reset_token.html

A fim de evitar que os administradores fiquem permanentemente bloqueados ao esquecer suas senhas, a página “reset_request.html” e “reset_token.html” foram criadas. Como explicado nas seções anteriores, a página “reset_request.html” é responsável por enviar um e-mail ao administrador, com instruções um link, que redirecionará o administrador para a página “reset_token.html”.

Na página “reset_token.html” o administrador pode escolher uma nova senha e necessita apenas confirma-la digitando novamente. Após isso, estará apto a fazer o login novamente e

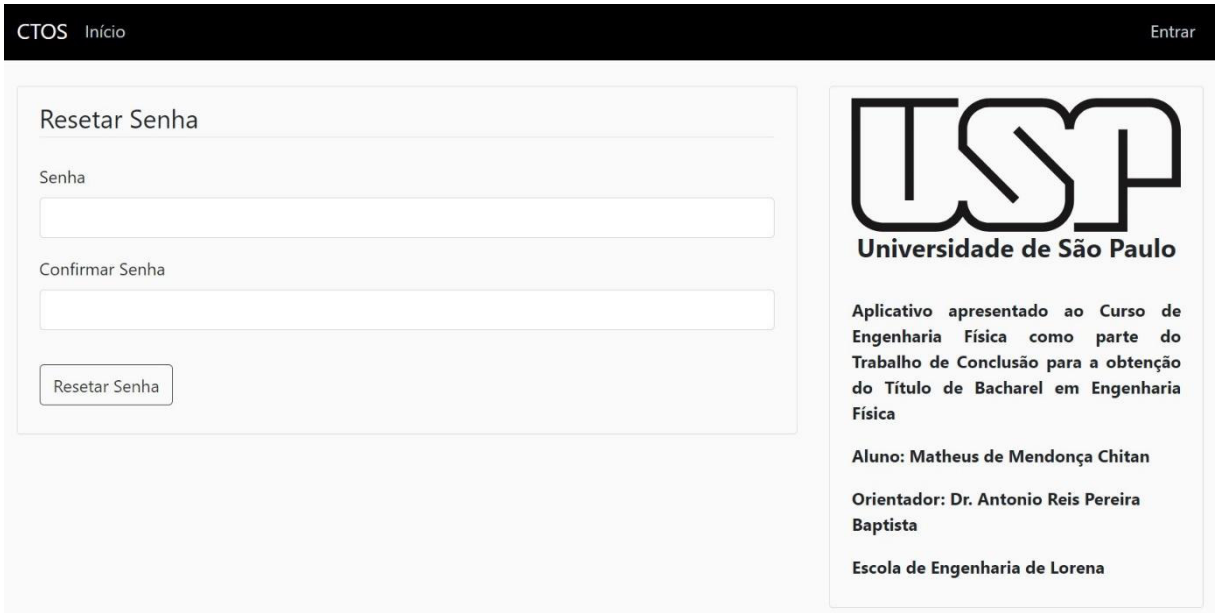
utilizar as funcionalidades do aplicativo. Nas Figuras 80 e 81, é possível verificar as páginas “reset_request.html” e “reset_token.html”, respectivamente.

Figura 80: Página “reset_request.html”



Fonte: Autoria Própria

Figura 81: Página “reset_token.html”



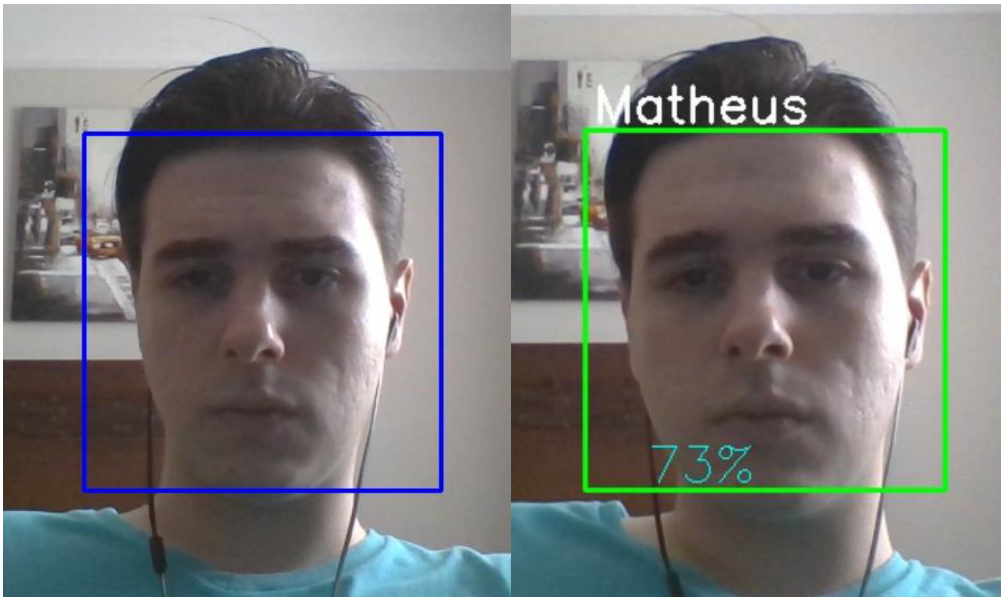
Fonte: Autoria Própria

3.5.2 Detecção e Reconhecimento facial

Nesta subseção serão exibidos, na prática, a atuação do algoritmo de detecção e reconhecimento facial. Cabe ressaltar que a parte teórica, que explica o funcionamento dos algoritmos de detecção e reconhecimento, foram descritos na seção 2.4.

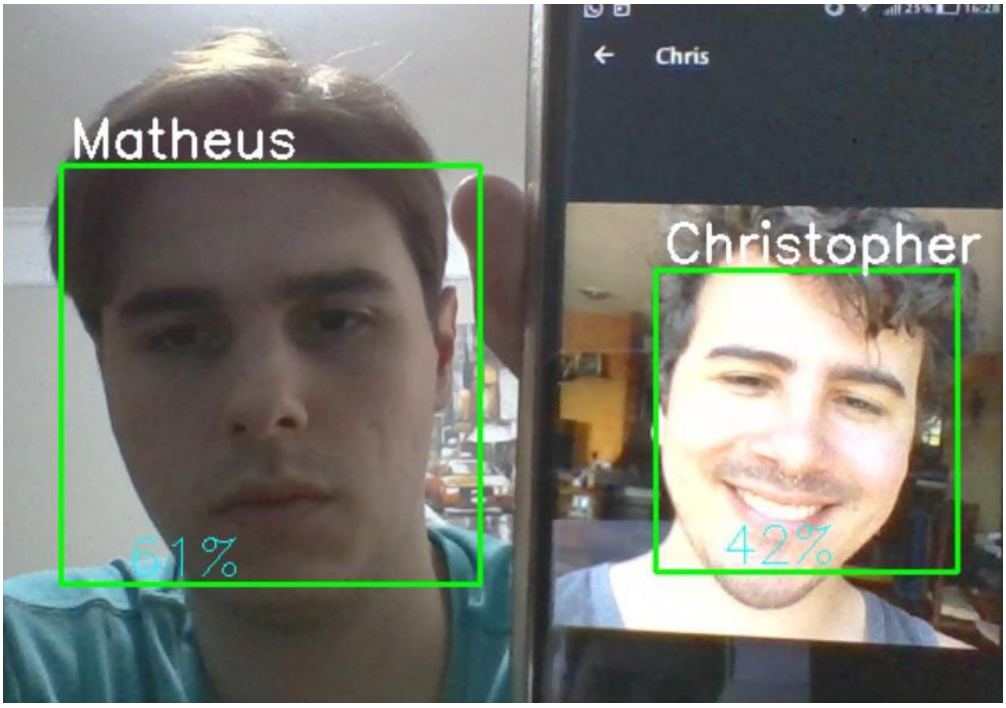
Na Figuras 82 e 83 é possível verificar, na prática, as atuações do detector facial e do reconhecimento facial.

Figura 82: A esquerda é apresentado o detector facial e a direita o reconhecimento facial



Fonte: Autoria Própria

Figura 83: Reconhecimento facial com duas pessoas distintas



Fonte: Autoria Própria

Para verificar a atuação na prática da detecção e reconhecimento facial, assim como o funcionamento integral do aplicativo, é possível acessar o Github a seguir: MChitan77/TCC-Videos: Vídeos apresentados na defesa de Trabalho de Conclusão de Curso de Matheus de Mendonça Chitan (github.com), onde estarão disponíveis 2 vídeos.

4. DISCUSSÃO DE RESULTADOS

É importante ressaltar que o aplicativo desenvolvido, juntamente com a detecção e reconhecimento facial, está sendo executado em um notebook com processador Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz e 16GB de RAM instalada. Com isso, é possível discutir alguns dos resultados obtidos, principalmente relacionados a detecção facial e reconhecimento facial.

4.1. Detecção facial

O algoritmo de detecção utilizado, Haar Cascade, pelo fato de ter sido um dos primeiros algoritmos desenvolvidos para a detecção de objetos, apresenta suas limitações. Mais especificamente, com relação a detecção facial, um dos problemas que deve ser levado em consideração, quando utilizado o algoritmo, é a inclinação das faces na imagem. Em outras palavras, faces que não estejam na posição vertical ou muito próxima dessa, não serão detectadas ou haverá uma detecção muito precária. Nas Figuras 84 e 85, é possível verificar a influência da inclinação da face na detecção.

Figura 84: Influência da inclinação na detecção facial com a utilização do algoritmo Haar Cascade A

Fonte: Autoria Própria

Figura 85: Influência da inclinação na detecção facial com a utilização do algoritmo Haar Cascade B

Fonte: Autoria Própria

Da Figura acima é possível verificar que inclinações na face impossibilitam que o algoritmo seja executado corretamente. Entretanto, apesar de apresentar tal problema, a utilização do algoritmo Haar Cascade ainda é grande valia, uma vez que em comparação com algoritmos mais modernos, que garantem uma maior taxa de detecção, este apresenta uma maior velocidade de processamento. Assim, em aplicações em que haja a possibilidade das pessoas/usuários pararem na frente da câmera de captura, como é o caso, a utilização do algoritmo Haar Cascade é grande utilidade.

E por último, a escolha do algoritmo Haar Cascade foi mediante a velocidade de processamento que este possui, pela não necessidade de algoritmos mais robusto para a aplicação em questão e pela facilidade de implementação dentro do código Python.

4.2. Reconhecimento facial

Com relação ao reconhecimento facial é possível verificar pela Figura 82, que foi apresentado um valor de 73% para a semelhança entre a imagem captura naquele instante e as imagens armazenadas no diretório “Dataset”. Entretanto, há uma grande dificuldade em estabelecer um número médio para a porcentagem geral de reconhecimento, uma vez que diferentes fatores podem interferir no cálculo.

A fim de explicar melhor, é importante enfatizar que o algoritmo de reconhecimento facial utilizado, LBPH – Local Binary Patterns Histogram, diferentemente do que muitos entendem ou pensando quando o termo “Reconhecimento Facial” é utilizado, não é baseado em uma inteligência artificial ou apresenta um aprendizado de máquina.

O Algoritmo utilizado se baseia em Visão Computacional, trabalhando diretamente com a manipulação das imagens fornecidas. Assim, o algoritmo é fortemente influenciado pelas condições das imagens fornecidas, como: brilho, sombra, contraste, cor, saturação, entre outras. Isso faz com que as condições ao redor da pessoa, tanto na captura da imagem quanto na hora do reconhecimento, influenciem na porcentagem de semelhança.

Exemplo: Supondo que no momento da captura para o banco de dados, esteja ensolarado e quando a pessoa foi utilizar o reconhecimento facial esteja chovendo. É muito provável que a porcentagem de semelhança seja em torno de 45-50% (tendo em mente que 75% de semelhança é extremamente alto para o algoritmo). Agora, supondo que esteja ensolarado no momento do reconhecimento, é bem provável que a porcentagem chegue a 65-75%. Também é importante ressaltar que o reconhecimento facial é executado mais de uma vez por segundo e que está constantemente comparando a foto captura pela câmera do computador (por exemplo) com todas as fotos armazenadas no diretório “Dataset”. Assim, a porcentagem está constantemente se alterando.

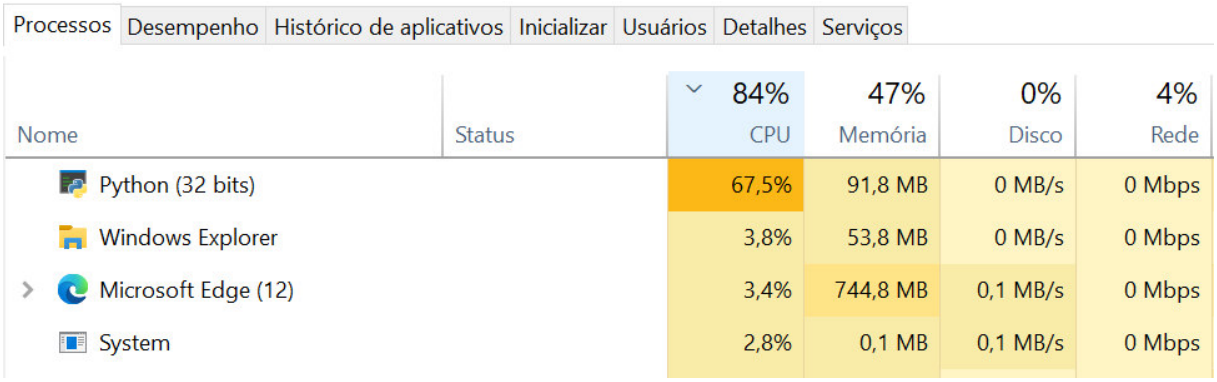
Outro fator que é consequência direta do algoritmo utilizado ser baseado em imagens e manipulação das mesmas, é a questão de fraudes. Caso alguém utilize uma máscara que se assemelhe a alguma das faces que esteja presente no banco de imagens, no diretório “dataset”, é muito provável que o algoritmo irá assumir que a máscara seja realmente a pessoa em questão.

Dessa forma, pelo que foi discutido acima, é uma difícil tarefa estimar um número médio para a porcentagem de semelhança entre a foto capturada pela câmera e as fotos armazenadas no diretório “Dataset” e evitar fraudes. Ainda mais quando aplicativo estiver sendo utilizado para o reconhecimento de mais de uma pessoa/usuário.

4.3. Processamento

Como dito anteriormente, o tratamento e processamento de imagens é custoso computacionalmente, uma vez que diversas funções estão sendo executadas simultaneamente. Um exemplo que pode ser dado, é da porcentagem da CPU que está sendo utilizada ao executar a função de detecção facial do aplicativo, mostrada na figura 86.

Figura 86: Porcentagem da CPU que está sendo utilizada ao executar a função de detecção facial do aplicativo



Fonte: Autoria Própria

Da Figura é possível verificar que 67,5% da CPU está sendo utilizada para a execução da função de detecção facial. Tal valor é relativamente alto, ainda mais para um computador Intel Core i7 de 7ª geração com 16 GB de RAM.

Assim, a fim de reduzir o custo computacional/porcentagem da CPU utilizada para a execução do aplicativo, seria de grande interesse a utilização de uma unidade de processamento gráfico, ou GPU, como também são conhecidas, para o processamento das imagens e atividades relacionadas a elas. Tais unidades são especialmente desenvolvidas para a administração e controle de funções de exibição de vídeo.

5. CONCLUSÃO

Considerando o rápido avanço da tecnologia e a grande necessidade de propor soluções inovadoras para as atividades de trabalho, a automação de tarefas repetitivas e cansativas, de modo que não necessitem da supervisão humana, vem ganhando espaço e sendo utilizada de forma crescente em nível mundial. Desde simples processos de automações para a obtenção de dados e informações até complexos desenvolvimentos de redes neurais para sistemas de orientação e controle de atuadores do mundo real.

Tais automações, além de proporcionarem ao ser humano conforto, uma vez que tais tarefas podem ser cansativas e repetitivas, possibilitam uma melhor otimização de tempo e recursos, fazendo com que diferentes aspectos possam ser tratados e trabalhos dentro de uma instituição.

O presente trabalho teve como objetivo propor e desenvolver um desses sistemas de automação, mais especificamente, um aplicativo de detecção e reconhecimento facial que possa ser implementado em estabelecimentos e instituições como um meio de auxílio à segurança, com potencial para ser adotado pela Escola de Engenharia de Lorena, auxiliando no registro e controle de entrada e saída de pessoas no *campus* universitário.

Com base nos resultados e discussões apresentadas, foi verificado que o aplicativo desenvolvido cumpriu com o esperado, podendo ser utilizado na prática, caso fosse desejado e se houvesse pequenas mudanças, como sugeridas na seção 4.

REFEREÊNCIAS

15 Most Colourful Animals. **PopUp Painting**. [S.I.]: mar, 2018. Disponível em: https://popuppainting.com/2018/03/15-colourful-animals/. Aceso em: 03 set. 2021

ACDSystem. **Canvas Tips and Techniques: Understanding color channesl**. [S.I.]: ACDSystem, [2003?]. 14 p. Disponível em: http://files.acdsystems.com/english/support/canvas/canvas-software-downloads/pdf-tutorials/color_channels.pdf. Acesso em: 03 set. 2021.

Anaconda. **Anaconda.Documentation**. [S.I.]: Anaconda, 2021. Disponível em: https://docs.anaconda.com/anaconda/index.html. Acesso em 31 ago. 2021.

BBC. **Facebook settles facial recognition dispute**. [S.I.]: BBC, 2020. Disponível em: https://www.bbc.com/news/technology-51309186. Acesso em: 22 out. 2021.

Bootstrap. **The most popular HTML, CSS and JSlibrary in the world**. [S.I.]: Bootstrap, [201-]. Disponível em: https://getbootstrap.com/. Acesso em: 31 ago. 2021

Businesswire: A Berkshire Hathaway Company. **IDEMIA's Facial Recognition Ranked #1 in NIST's Latest FRVT Test**. [S.I.]: Businesswire, 2021. Disponível em: https://www.businesswire.com/news/home/20210406005123/en/IDEMIA%E2%80%99s-Facial-Recognition-Ranked-1-in-NIST%E2%80%99s-Latest-FRVT-Test. Acesso em: 23 out. 2021.

CARBONNELLE, Pierre. **PYPL PopularitY of Programming Language**. [S.I.]: CARBONNELLE, 2021. Disponível em: https://pypl.github.io/PYPL.html. Acesso em: 31 ago. 2021.

CHOKSHI, Niraj. **Facial Recognition's Many Controversies, From Stadium Surveillance to Racist Software**. Nova York, NY: CHOKSHI, 2019. Disponível em: https://www-nytimes-com.translate.goog/2019/05/15/business/facial-recognition-software-controversy.html?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt-BR&_x_tr_pto=nui. Acesso em: 22 out. 2021.

CHM – Computer History Museum. **Guido Van Rossum: For the creation and evolution of the Python programming language and for leadership of its community**. Mountain View, CA: CHM, 2018. Disponível em: https://computerhistory.org/profile/guido-van-rossum/. Acesso em: 31 ago. 2021.

GALILEU. **Concurso divulga 10 das melhores fotos da natureza selvagem de 2020**. [S.I.]: GALILEU, 2020. Disponível em: https://revistagalileu.globo.com/Ciencia/Meio-Ambiente/noticia/2020/09/concurso-divulga-10-das-melhores-fotos-da-natureza-selvagem-de-2020.html. Acesso em: 01 set. 2021.

Flask-SQLAlchemy. **FLask-SQLAlchemy Documentation (2.x): Select, Inser, Delete**. [S.I.]: Flask-SQLAlchemy, 2010 Disponível em: https://flask-sqlalchemy.palletsprojects.com/en/2.x/queries/#querying-records. Acesso em: 31 ago. 2021.

GAY, Jeremy. **How to look after a Yemen chameleon**. Manchester, UK: SwellReptiles, 2020 Disponível em: https://www.reptiles.swelluk.com/blog/how-to-look-after-a-yemen-chameleon/. Acesso em: 02 set. 2021.

HARDING, David. **Facial Recognition: When Convenience and Privacy Collide**. [S.I.]: HARDING, 2019. Disponível em: https://www.securitymagazine.com/articles/90533-facial-recognition-when-convenience-and-privacy-collide. Acesso em: 23 out. 2021.
JET BRAINS. **PyCharm, O IDE Python para desenvolvedores profissionais**. [S.I.]: JET BRAINS, [201-]. Disponível em: https://www.jetbrains.com/pt-br/pycharm/. Acesso em: 31 ago. 2021.

MARR, Bernard. **Facial Recognition Technology: Here Are The Important Pros And Cons**. [S.I.]: MARR, 2019. Disponível em: https://www.forbes.com/sites/bernardmarr/2019/08/19/facial-recognition-technology-here-are-the-important-pros-and-cons/?sh=c8de57714d16. Acesso em: 22 out. 2021.

NEC. **A brief history of Facial Recognition**. Nova Zelândia: NEC, 2020 Disponível em: https://www.nec.co.nz/market-leadership/publications-media/a-brief-history-of-facial-recognition/. Acesso em: 25 out. 2021.

NIST. **Face Recognition Grand Challenge (FRGC)**. Gaithersburg, MD: NIST, 2010. Disponívem em: https://www.nist.gov/programs-projects/face-recognition-grand-challenge-frgc. Acesso em: 22 out. 2021.

NIST. **Face Recognition Vendor Test (FRVT): Prior Tests and Activities**. Gaithersburg, MD: NIST, 2010. Disponível em: https://www.nist.gov/programs-projects/face-recognition-vendor-test-frvt. Acesso em: 23 out. 2021.

Numpy. **What is NumPy?**.[S.I.]: Numpy, 2021. What is NumPy?. Disponível em: https://numpy.org/doc/stable/user/whatisnumpy.html#who-else-uses-numpy. Acesso em: 31 ago. 2021.

OpenCV - Open Source Computer Vision. **About OpenCV**. [S.I.]: OpenCV, [200-]. Disponível em: https://opencv.org/about/. Acesso em: 01 set. 2021.

OpenCV - Open Source Computer Vision. **Color Conversion**. [S.I.]: OpenCV, [201-] . Disponível em: https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html. Acesso em: 03 set. 2021.

OpenCV - Open Source Computer Vision. **Face Recognition with OpenCV.** [S.I.]: OpenCV [201-]. Disponível em: https://docs.opencv.org/3.4/da/d60/tutorial_face_main.html. Acesso em: 08 set. 2021.

Open Source Initiative. **Frequently Answered Questions: What is "Open Source" software?.** [S.I.]: Open Source Initiative, [entre 1995 e 2005]. Disponível em: <https://opensource.org/faq#osd>. Acesso em: 31 ago. 2021.

Pallets. **Flask Documentation (2.0.x)**. [S.I.]: Pallets. 2010. Disponível em: https://flask.palletsprojects.com/en/2.0.x/. Acesso em: 31 ago. 2021.

PETROU, Maria; PETROU, Costa. **Image Processing: The Fundamentls**. 2ª edição. Reino Unido: John Wiley and Sons Ltd, 2010. 794 p.

Python Software Foundation. **What is Python? Executive Summary**. [S.I.]: Python Software Foundation, [2001]. Disponível em: https://www.python.org/doc/essays/blurb/. Acesso em: 31 ago. 2021

ROLL, Lara C. **Worktech Academy. How the pandemic has given a boost to workplace automation: The 'next normal' in the workplace is set to accelerate company moves towards greater automation. Lara Roll gathers the evidence**. Londres, UK: ROLL, 2019. Disponível em: https://www.worktechacademy.com/how-the-pandemic-has-given-a-boost-to-workplace-automation/. Acesso em: 22 out. 2021.

ROSEBROCK, Adrian. **Face Recognition with Local Binary Patterns (LBPs) and OpenCV**. [S.I.]: Pyimagesearch, 2021. Disponível em: https://www.pyimagesearch.com/2021/05/03/face-recognition-with-local-binary-patterns-lbps-and-opencv/. Acesso em: 20 nov. 2021

ROSEBROCK, Adrian. **Local Binary Patterns with Python & OpenCV**. [S.I.]: Pyimagesearch, 2015. Disponível em: https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/. Acesso em: 08 set. 2021.

SCHAFER, Corey. Python Flask Tutorial: Full-Featured Web App Part 2 – Templates. [S.I.]: SCHAFER, [201-]a. 1 vídeo (31 min.). Disponível em: https://www.youtube.com/watch?v=QnDWIZuWYW0&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH&index=2. Acesso em 31 jul; 2021.

SCHAFER, Corey. Python Flask Tutorial: Full-Featured Web App Part 3 - Forms and User Input. [S.I.]: SCHAFER, [201-]b. 1 vídeo (48 min). Disponível em: https://www.youtube.com/watch?v=UIJKdCIEXUQ&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH&index=3. Acesso em: 1 ago. 2021.

SCHAFER, Corey. Python Flask Tutorial: Full-Featured Web App Part 4 - Database with Flask-SQLAlchemy. [S.I.]: SCHAFER, [201-]c. 1 vídeo (29 min). Disponível em: https://www.youtube.com/watch?v=cYWiDiIUxQc&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH&index=4. Acesso em: 3 ago. 2021

SCHAFER, Corey. Python Flask Tutorial: Full-Featured Web App Part 6 - User Authentication. [S.I.]: SCHAFER, [201-]d. 1 vídeo (47 min). Disponível em: https://www.youtube.com/watch?v=CSHx6eCkmv0&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH&index=6. Acesso em: 4 ago. 2021.

SCHAFER, Corey. Python Flask Tutorial: Full-Featured Web App Part 10 - Email and Password Reset. [S.I.]: SCHAFER, [201-]e. 1 vídeo (47 min). Disponível em: https://www.youtube.com/watch?v=vutyTx7IaAI&list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH&index=10. Acesso em: 5 ago. 2021.

SERENGIL, Sefik Ilkin. **A Beginner's Guide to Face Recognition with OpenCV in Python**. [S.I.]: Sefik Ilkin Serengil, 2020. Disponível em: https://sefiks.com/2020/07/14/a-beginners-guide-to-face-recognition-with-opencv-in-python/. Acesso em: 10 set. 2021.

SINFIC. **Reconhecimento Facial: um Pouco de História e Principais Abordagens**. Portugal: SINFIC, 2008. Disponível em: http://www.sinfic.pt/SinficWeb/displayconteudo.do2?numero=24923. Acesso em: 22 out. 2021.

SOUZA, Leonardo. **Veja homenagens para Ayrton Senna, maior ídolo na Formula 1.** São Paulo, SP: R7, 2019. Disponível em: https://esportes.r7.com/automobilismo/fotos/veja-homenagens-para-ayrton-senna-maior-idolo-na-formula-1-03052019#/foto/1. Acesso em 03 set. 2021.

SQLALchemy. **The Python SQL Toolkit and Object Relational Mapper**. [S.I.]: SQLALchemy, [20--]. Disponível em: https://www.sqlalchemy.org/. Acesso em: 31 ago. 2021.

SQLite Consortium. **What Is SQLite?**. [S.I.]: SQLite, [entre 1995 e 2005]. Disponível em: https://www.sqlite.org/index.html. Acesso em: 01 set. 2021.

SYMANOVICH, Steve. **What is facial recognition? How facial recognition works**. [S.I.]: SYMANOVICH, 2021. Disponível em: https://us.norton.com/internetsecurity-iot-how-facial-recognition-software-works.html. Acesso em: 21 out. 2021.

TASKIRAN Murat, KAHRAMAN Nihan, ERDEM Cigdem Eroglu. **Face recognition: Past, present and future (a review),** Digital Signal Processing, Volume 106, 2020, 102809, ISSN 1051-2004. Disponível em: https://www.sciencedirect.com/science/article/pii/S1051200420301548. Acesso em: 18 nov. 2021.

VIOLA, Paul; JONES, Michael. **Rapid Object Detection using a Boosted Cascade of Simple Features**. In: IEEE computer society conference on computer vision and pattern recognition, 2001, Kauai, HI. Proceedings... Kauai: 2001. 1.v. p. 511-518.

W3Schools. **Introduction to SQL**. [S.I.]: W3School, [200-] Disponível em: https://www.w3schools.com/sql/sql_intro.asp. Acesso em: 31 ago. 2021.