UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

MARCELA WANDERLEY DE SIQUEIRA GALINDO

AUTOMAÇÃO DE CONTROLE DE ACESSO POR RECONHECIMENTO FACIAL DESENVOLVIDO EM LINGUAGEM PYTHON

Recife
2019

MARCELA WANDERLEY DE SIQUEIRA GALINDO

AUTOMAÇÃO DE CONTROLE DE ACESSO POR RECONHECIMENTO FACIAL DESENVOLVIDO EM LINGUAGEM PYTHON

Trabalho de Conclusão do Curso de Engenharia de Controle e Automação da Universidade Federal de Pernambuco, como requisito da disciplina de Trabalho de Conclusão de Curso (EL403)

Orientador: Profº. Dr. Hermano Andrade Cabral

Recife

2019

Catalogação na fonte

Bibliotecário Josias Machado , CRB-4 / 1690

G158a Galindo, Marcela Wanderley de Siqueira

Automação de controle de acesso por reconhecimento facial desenvolvido em linguagem Python / Marcela Wanderley de Siqueira Galindo. – Recife, 2019.

59 folhas, il., figs., gráfs.

Orientador: Hermano Andrade Cabral.

TCC (Graduação) – Universidade Federal de Pernambuco. CTG. Departamento de Graduação em Engenharia Elétrica, 2019.

Inclui Referências.

1. Engenharia elétrica. 2. Reconhecimento facial. 3. Machine learning. 4. Raspberry. 5. Controle de acesso. 6. Automação. I. Cabral, Hermano Andrade (orientador). II. Título.

UFPE

621.3 CDD (22. ed.) BCTG/2019-296

MARCELA WANDERLEY DE SIQUEIRA GALINDO

AUTOMAÇÃO DE CONTROLE DE ACESSO POR RECONHECIMENTO FACIAL DESENVOLVIDO EM LINGUAGEM PYTHON

Trabalho de Conclusão de Curso do Curso de Engenharia de Controle e Automação da Universidade Federal de Pernambuco, como requisito da disciplina de Trabalho de Conclusão de Curso (EL403).

Aprovada em: 04/07/2019.

BANCA EXAMINADORA

_____

Profº. Me. Geraldo Leite Maia Junior
Universidade Federal de Pernambuco

_____

Profº. Dr. Márcio Evaristo da Cruz Brito
Universidade Federal de Pernambuco

AGRADECIMENTOS

Agradeço, primeiramente, a Deus e a todos aqueles que fizeram parte da minha caminhada na universidade, tornando-a menos penosa.

Aos meus amigos, por todas as madrugadas de estudo que nos levaram a aprovação.

Aos meus pais e familiares, pelo apoio em todos os momentos e pela compreensão nos piores deles.

A todos que acreditaram em mim, que me ouviram quando precisei ser escutada e aos conselhos que me ofereceram.

À Universidade que me preparou, não apenas como profissional, mas como ser humano que sou hoje.

RESUMO

Esta pesquisa tem como objetivo propor um método de automação de controle de acesso de forma fácil, simples, segura e de baixo custo, utilizando-se de tecnologias modernas de *hardware* e *software*. No método proposto foram usados algoritmos de *machine learning*, através de redes neurais convolucionais, em linguagem Python. Primeiramente, foi feita uma programação executada num computador com sistema operacional Linux. Em seguida, com a validação do *software,* a implementação foi feita numa raspberry pi 3 modelo B, acoplada a uma câmera, para captura de imagens e realização do reconhecimento facial; e a um pequeno circuito eletrônico que simulou uma fechadura eletrônica. Esta, ao reconhecer uma face, permitiu o acesso. Como o reconhecimento é feito através da escolha de um limiar, o modelo mostrou-se flexível, tornando-se cada vez mais restritivo à medida que este limiar foi reduzido. Concluiu-se, através de testes com esse limiar, um aumento na confiabilidade ao reduzi-lo, pois diminuiu a probabilidade de se reconhecer um falso positivo.

Palavras-chave: Reconhecimento facial. *Machine learning*. Raspberry. Controle de acesso. Automação.

ABSTRACT

This research aims to propose a method of access control automation in an easy, simple, safe and low cost way using modern hardware and software technologies. In the proposed method, machine learning algorithms were used, through convolutional neural networks, in Python language. First, a program was run on a computer with a Linux operating system. Then, with the validation of the software, the implementation was done on a raspberry pi model B, coupled to a camera, for image capture and facial recognition; and a small electronic circuit that simulated an electronic lock. This, when recognizing a face, allowed access. As recognition is made by choosing a threshold, the model has shown to be flexible, becoming increasingly restrictive as this threshold is reduced. It was concluded, through tests with this threshold, an increase in reliability by reducing it, because it reduces the probability of recognizing a false positive.

Keywords: Facial recognition. Machine learning. Raspberry. Access control. Automation.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
IA	Inteligência Artificial
HOG	Histogram of Oriented Gradient
SVM	Support Vector Machine
IEEE	Institute of Electrical and Electronics Engineers
LFW	Labeled Faces in the Wild

SUMÁRIO

1 INTRODUÇÃO

A ineficiência dos métodos tradicionais de controle de acesso tem causado sérios problemas àqueles que os utilizam. Os exemplos mais comuns são: as chaves e cartões, cuja perda ou roubo podem representar um empecilho para o acesso dos seus usuários; e os métodos com senha, que exigem boa memória ou comprometimento de segurança (para aqueles que costumam anotá-la), causando dificuldades de acesso, aborrecimentos e perda de tempo. Entretanto, características biológicas do indivíduo não podem ser perdidas, esquecidas ou roubadas. E isso reforça a ideia de que, automação de controle de acesso utilizando características biológicas possui estas e outras vantagens, em comparação com os acessos tradicionais (BAKSHI, SINGHAL, 2014).

Nesse contexto, pergunta-se: “Como elaborar, de forma automatizada e com baixo custo, um método para o controle de acesso que utilize como chave as características biológicas?”

Primeiramente, é necessário admitir que máquinas, devidamente programadas, são capazes de enxergar, processar e interpretar a realidade a sua volta. Assim sendo, sistemas simuladores de cognição humana associados a tecnologias de captura de imagens podem identificar e reconhecer traços e feições humanas de forma precisa e aguçada em qualquer computador (ANDRADE, 2019).

Admitida tal possibilidade, para que se possa fazer com que máquinas pensem e se comportem de maneira inteligente, é indispensável a utilização de sistemas visuais cujos hardwares e softwares permitam capturar, armazenar e manipular imagens e que, ao mesmo tempo, tenham suporte para tomada de decisão baseados em algoritmos de análise ou de inteligência artificial (ALVES et al., 2018).

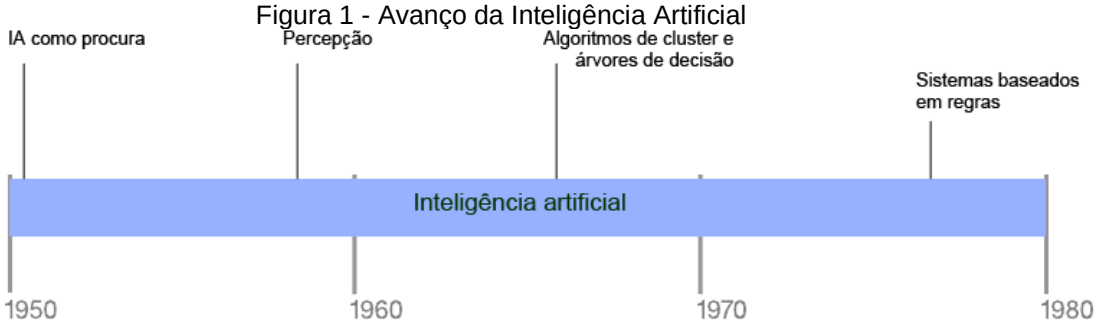
1.1 A evolução da Inteligência Artificial

O termo “inteligência artificial” se refere à inteligência presente nas máquinas, inteligência essa que vem sendo tema de pesquisa de diversas áreas como: computação, linguística, filosofia, matemática, neurociência, dentre outras, desde 1950. Muitas das grandes empresas de tecnologia, como a *Google* e o *Facebook*, têm investido nesta área, devido à corrida tecnológica atual (NEXO JORNAL, 2019).

Os primeiros projetos sobre esse tema utilizaram redes neurais simples e as máquinas tinham sua inteligência testada ao desafiar a inteligência dos seres humanos em atividades que apenas estes conseguiriam executar, como em jogos, por exemplo (NEXO JORNAL, 2019).

Até os anos de 1980, as inteligências artificiais eram relativamente simples e seu avanço era lento (Figura 1) (JONES, 2017).

Figura 1 - Avanço da Inteligência Artificial

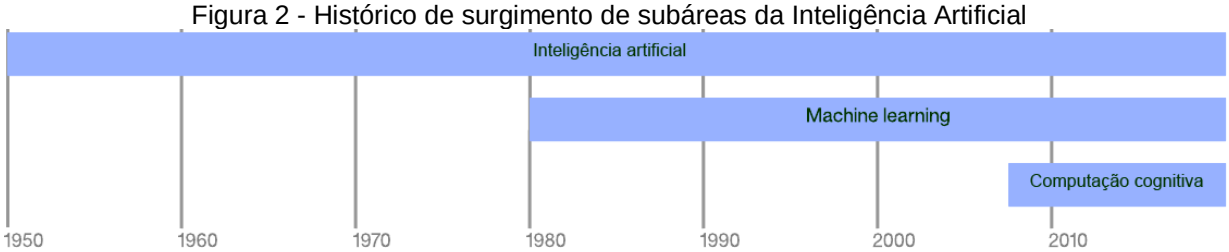


Fonte: JONES (2017).

Paralelamente, os avanços computacionais permitiram que inteligências cada vez mais complexas pudessem ser desenvolvidas, possibilitando o reconhecimento de letra recursiva, voz e imagem. (NEXO JORNAL, 2019).

A evolução da inteligência artificial também deu margem a outros âmbitos de pesquisa, aparecendo novas subáreas, como: *machine learning* (ou aprendizado de máquina), e computação cognitiva. A primeira surgindo em meados dos anos 1980 e a segunda, mais recente, aparecendo na década passada (Figura 2) (JONES, 2017).

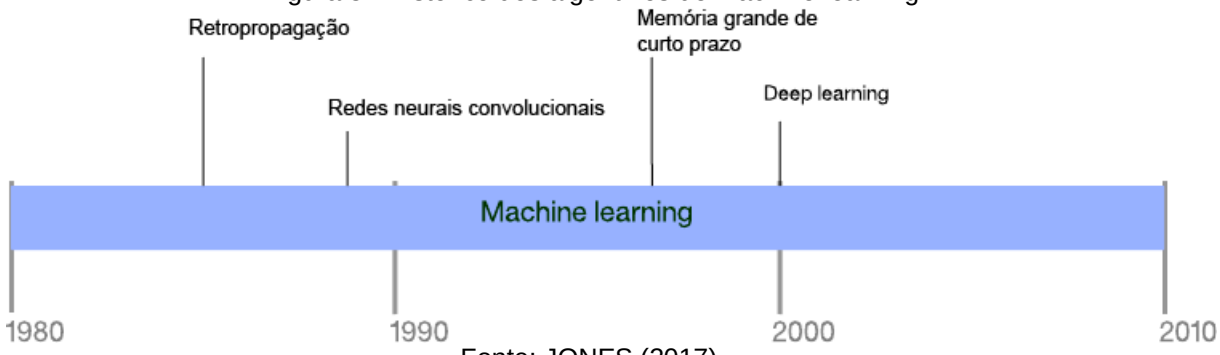
Figura 2 - Histórico de surgimento de subáreas da Inteligência Artificial



Fonte: JONES (2017).

O aprendizado de máquina surge com o “propósito de oferecer aos computadores a capacidade de aprender e desenvolver modelos para realizar atividades como previsão dentro de domínios específicos” (JONES, 2017). Na Figura 3, os algoritmos de *machine learning* são citados em ordem cronológica, sendo o das redes neurais convolucionais de interesse para esta dissertação.

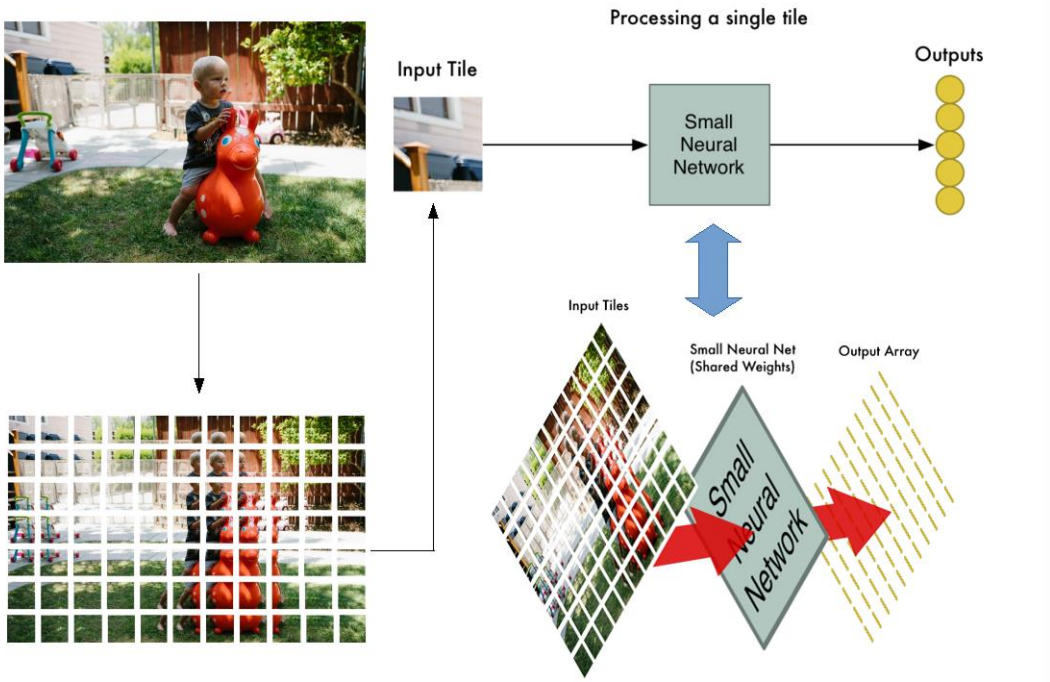
Figura 3 - Histórico dos algoritmos de Machine learning



Fonte: JONES (2017).

O algoritmo de redes neurais convolucionais é um dos algoritmos que pode ser utilizado para reconhecer padrões. Esse algoritmo, que foi estabelecido por um artigo de 1998, “Gradient-based learning applied to document recognition“ (LeCun et. al, 1998), de Yann LeCun, Léon Bottou, Yoshua Bengio e Patrick Haffneré, é baseado em redes neurais multicamadas que se inspiram no córtex humano, por isso é muito utilizado para o processamento de imagens. “A imagem é dividida em campos receptivos e são refletidos em uma camada convolucional que extrai recursos da imagem de entrada” (Figura 4) (JONES, 2017). Numa rede treinada para o padrão de faces, o uso de aprendizado de máquina pode ser uma boa alternativa para o reconhecimento facial.

Figura 4 - Funcionamento das redes convolucionais



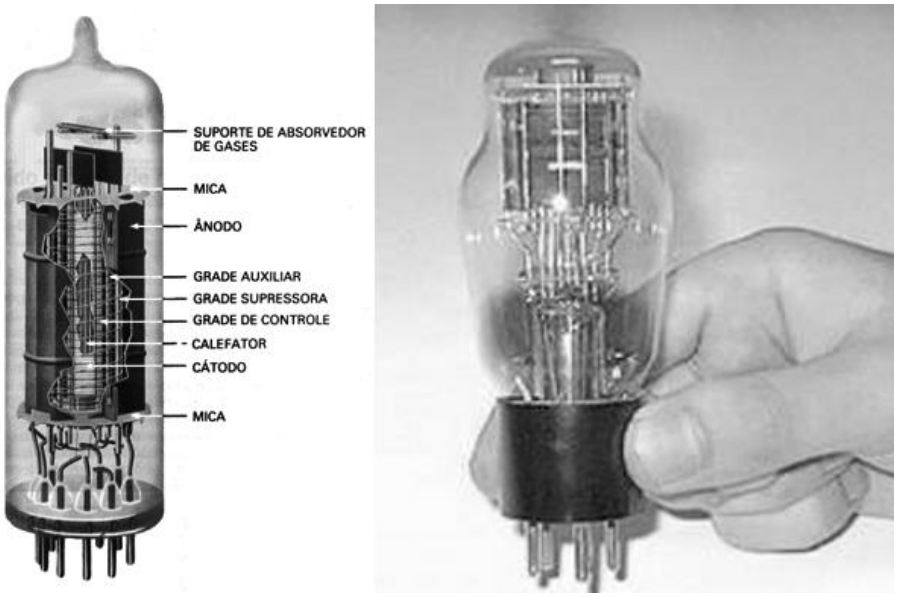
Fonte: GEITGEY (2016).

1.2 A evolução dos hardwares

O primeiro computador eletrônico foi datado de meados dos anos de 1940 e, além de ter uma velocidade de processamento milhares de vezes menor do que a dos computadores mais modernos, o seu tamanho é uma característica que vem sendo reduzida com os novos eletrônicos, assim como o custo para a sua aquisição (MOHAPATRA, 2015).

A primeira geração desses eletrônicos funcionava com tubos à vácuo (Figura 5) e, além de serem extensivos ao tamanho de uma sala e terem preços inacessíveis à maioria, eles consumiam uma grande quantidade de energia e produziam excessiva quantidade de calor. Ademais, possuíam uma tecnologia bastante simples e ineficiente, utilizando a “linguagem de máquina”, linguagem de mais baixo nível compreendida por computadores eletrônicos (MOHAPATRA, 2015).

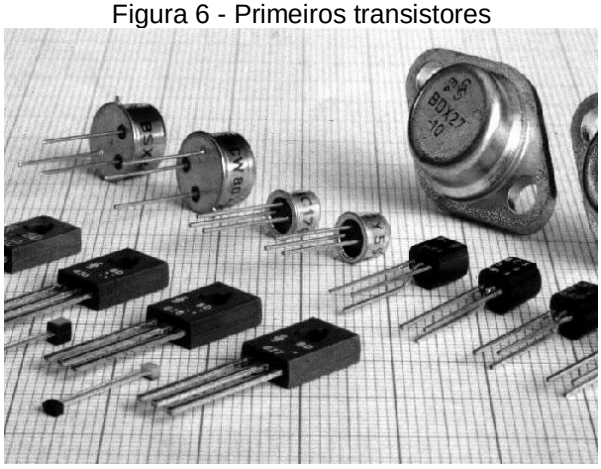
Figura 5 - Tubos à vácuo



Fonte: MARTINS (2015).

A partir de 1956, a segunda geração surgiu, trazendo a tecnologia dos transistores (Figura 6) em substituição aos tubos à vácuo. Isso permitiu a produção de computadores menores, mais rápidos, baratos e com menor consumo de energia. Essa geração trazia, também, compreensão a novas linguagens, as linguagens simbólicas, que utilizam palavras como instrução (MOHAPATRA, 2015).

Figura 6 - Primeiros transistores



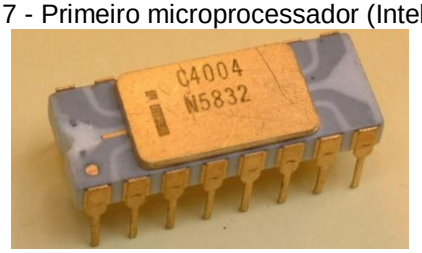
Fonte: PIROPO (2012).

Em 1960, na "*International Solid-State Circuits Conference"*, realizada no campus da Universidade da Pensilvânia, na Filadélfia, o engenheiro da computação, Douglas Engelbart, introduziu a teoria de que, quanto menores os circuitos eletrônicos fossem feitos, mais rápidos seriam, demandariam menos potência e a

produção seria mais barata. Sua teoria também dizia que “o número de transistores que poderiam ser gravados em um chip dobraria anualmente por pelo menos uma década, levando a aumentos astronômicos no poder do computador”. Essa teoria foi comprovada, poucos anos depois, por Gordon Moore, fundador da Intel, e ficou conhecida como a Lei de Moore (*Moore’s Law*) (MARKOFF, 2015).

Esse acontecimento permitiu que as próximas gerações de computadores tivessem seus circuitos eletrônicos cada vez mais miniaturizados, proporcionando uma maior velocidade e eficácia. Isso contribuiu, também, para o surgimento de microprocessadores (Figura 7), com todos os componentes de um computador em um único chip (MOHAPATRA, 2015).

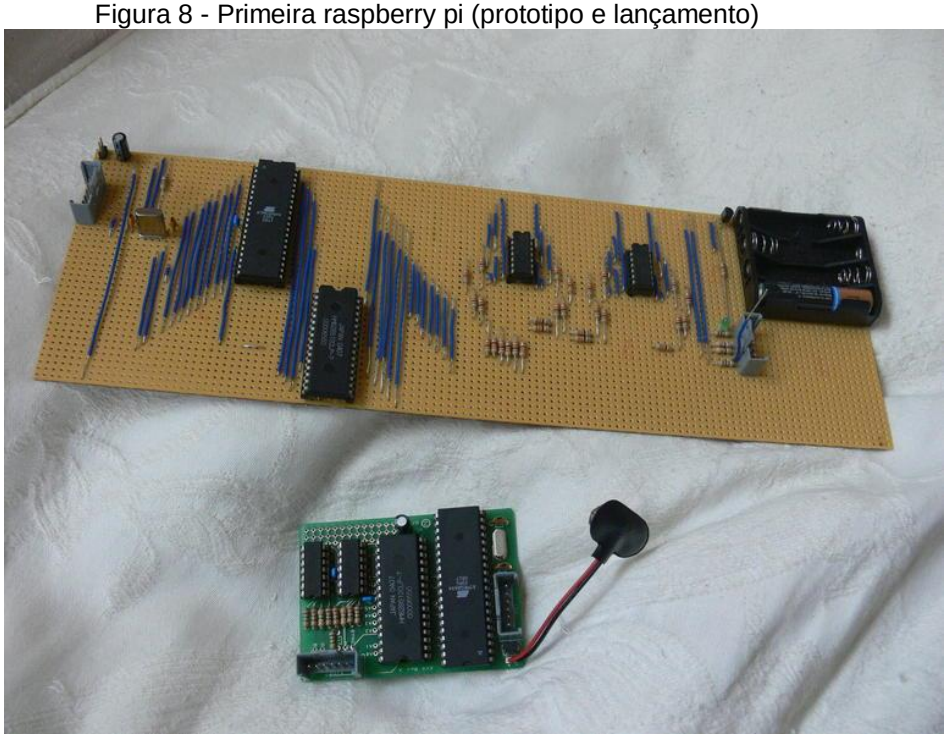
Figura 7 - Primeiro microprocessador (Intel 4004)



Fonte: MOREIRA (2011).

Anos depois, em 2011, surge a primeira raspberry Pi (Figura 8) como uma proposta de um computador pequeno, portátil, facilmente programável e de baixíssimo custo ($35). O “Pi”, em seu nome, refere-se à linguagem de programação Python, trazendo o conceito de facilmente programável, como mencionado em sua proposta (HEATH, 2018). Este microcomputador, que possui um processador com desempenho inversamente proporcional ao seu dimensionamento, foi utilizado para a proposta desta monografia.

Figura 8 - Primeira raspberry pi (prototipo e lançamento)



Fonte: HEATH (2018).

1.3 Justificativa

A proposta sugerida nesta pesquisa encontra respaldo no fato de que o reconhecimento facial, em comparação com outros métodos biométricos, possui inúmeras vantagens, entre elas: não ter a necessidade do uso de equipamentos caros e sensíveis, como é o caso para reconhecimento da íris; possuir uma aquisição de dados mais fácil por não ser necessário um contato direto, como no caso dos métodos que utilizam mão e dedos; e ser livre dos ruídos mesmo em ambientes muito populosos, em oposição ao reconhecimento de voz (BAKSHI and SINGHAL, 2019).

Como explicado nos tópicos anteriores, existem algoritmos e hardware que permitem esse tipo de aplicação de reconhecimento de faces por um preço acessível e baixo consumo.

1.4 Objetivos

1.4.1 Objetivos Gerais

O presente trabalho tem como objetivo geral propor uma alternativa à automação de controle de acesso, desenvolvida em linguagem Python, que permita facilidade de acesso e, ao mesmo tempo, confiabilidade e segurança, utilizando tecnologia de baixo custo e consumo.

1.4.2 Objetivos Específicos

- Apresentar uma forma de controle de acesso por reconhecimento facial, desenvolvida em linguagem Python.
- Selecionar possíveis algoritmos que satisfaçam a proposta de objetivo geral.
- Realizar simulação e testes do modelo escolhido que garantam sua segurança e confiabilidade.
- Implementar o modelo escolhido em hardware compatível, de baixo custo e eficiente.
- Apresentar o modelo em funcionamento sendo utilizado para a abertura de uma porta.

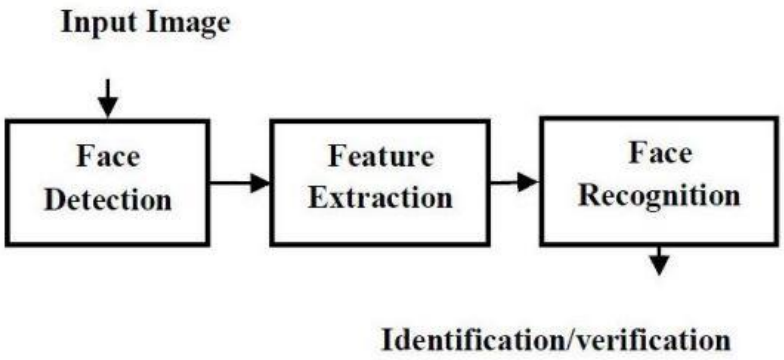
2 REFERENCIAL TEÓRICO

Para um melhor entendimento sobre a metodologia seguida na composição desta monografia, nas próximas sessões serão detalhados os principais conteúdos para a compreensão da proposta sugerida. Tais conteúdos permitem o aprendizado sobre: o passo a passo necessário para a realização do reconhecimento facial, assim como a apresentação dos algoritmos mais populares; o conhecimento e a expressividade da linguagem Python, para justificar a sua escolha como linguagem de programação utilizada na proposta; e, por fim, um maior aprofundamento a respeito da biblioteca escolhida para sua implementação: a “face_recognition”.

2.1 O Reconhecimento Facial

Existem inúmeros algoritmos, hoje em dia, que permitem o reconhecimento facial. Para todos eles é preciso seguir, pelo menos, três passos: a detecção facial, a extração das características (*features*) e, então, a identificação dos rostos, ou seja, o reconhecimento facial propriamente dito (Figura 9).

Figura 9 - Passo a passo para o reconhecimento facial

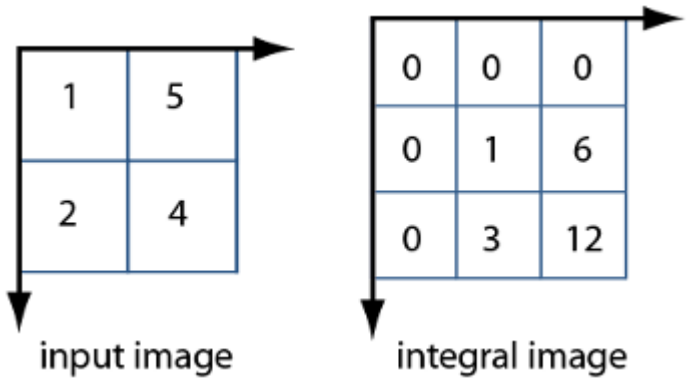


Fonte: BAKSHI and SINGHAL (2019).

Para cada um desses passos, várias são as técnicas disponíveis; o grande desafio está em escolher a mais adequada a cada aplicação. Os algoritmos de detecção mais populares nos dias atuais são o *Haar feature-based cascade* e o HOG (*Histogram of Oriented Gradient*). O primeiro foi apresentado por Paul Viola e Michael Jones no artigo “*Rapid Object Detection using a Boosted Cascade of Simple Features*” (em 2001) e o segundo proposto foi por Navneet Dalal e Bill Triggs no artigo *“Histograms of Oriented Gradients for Human Detection”* (em 2005).

O algoritmo proposto por Viola e Jones utiliza imagem em escala cinza para análise e apresenta um conceito de *Integral image* que consiste numa imagem que contém a soma dos pixels da esquerda a *x* e de cima a *y* (Figura 10). Essa imagem, similar a tabela de áreas somadas, é utilizada para operações com retângulos de pixels (somas e subtrações) necessárias para a determinação das *features*. E essa determinação é feita treinando uma variante do algoritmo de aprendizado de máquina AdaBoost (VIOLA; JONES, 2001).

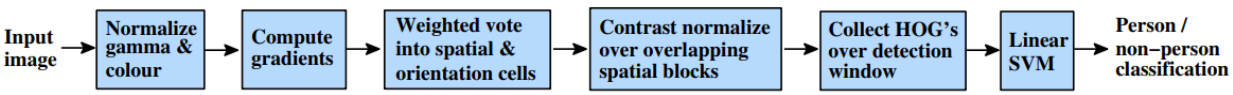
Figura 10 - *Integral Image*



Fonte: *MATHWORKS* (2019)

O HOG é um algoritmo baseado em histogramas e é implementado dividindo-se a janela de imagem em pequenas regiões (“células”) e para cada célula acumula-se um histograma 1-D local de direções de gradiente ou orientações de pixels da célula. Pode-se acumular uma medida da “energia” do histograma local em regiões espaciais maiores para normalizar todas as células do bloco e, assim, melhorar a invariância à iluminação do método. Esses blocos são chamados Histograma de Gradiente Orientado (HOG). A janela de detecção é formada por uma densa grade de descritores HOG e é utilizado um SVM (*Support Vector Machine* – método de *machine learning,* utilizado para reconhecer padrões e para classificação e análise de regressão) treinado para reconhecer histogramas de faces e, assim, realizar a detecção (N., B., 2005). O funcionamento do método pode ser visto através de um fluxograma, conforme Figura 11.

Figura 11 - Passo a passo para HOG



Fonte: N., B. (2005).

Para a extração das características (*features*) de cada face, existem 3 principais abordagens: *holistic approach*, *feature-based approach*, e *hybrid approach*. A primeira utiliza a face como um todo para retirar as características faciais individuais, enquanto a segunda utiliza as partes específicas da face e a relação geométrica entre elas. Já a terceira é uma mesclagem entre a primeira e a segunda abordagens (BAKSHI and SINGHAL, 2019).

Os métodos de extração mais conhecidos são: *eigenfaces* e *fisherfaces* (ambos classificados como *Holistic approach*) e alguns métodos que utilizam redes neurais convolucionais para classificação das faces.

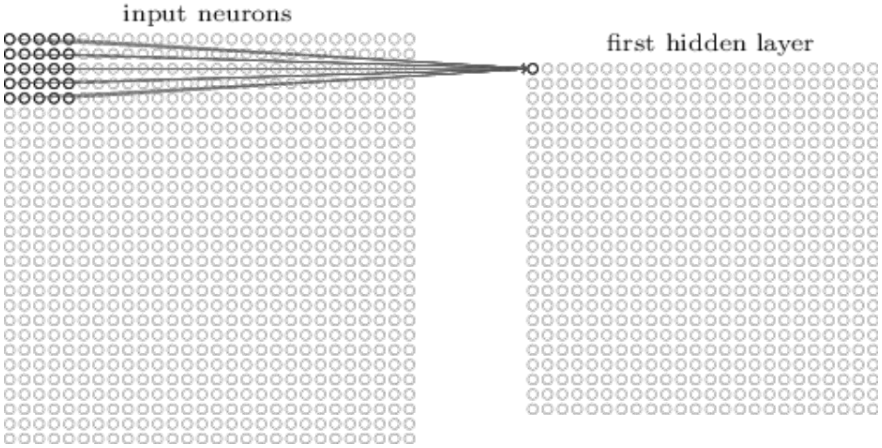
O método e*igenfaces* usa o conceito de *principal componentes*, que são partes que são consideradas mais relevantes dentre cada imagem treinada, considerando estes componentes como importantes e descartando o resto. Nesse algoritmo, a iluminação é considerada um *principal component*, tornando luz e sombra fatores importantes no reconhecimento (OPTICAL SOCIETY OF AMERICA A., 1987).

O método *fisherfaces* é similiar ao anterior, diferenciando-se no fato de que os *principal components* são extraídos de todas as imagens treinadas, como um todo. Assim, todos os componentes importantes para diferenciar o grupo de imagens são considerados *principal components* (WILEY ONLINE LIBRARY, 1936).

Um último método de extração popular, hoje em dia, é o das redes neurais convolucionais. Esse método, que foi citado na introdução desta monografia, tem sua popularidade não apenas para o reconhecimento de faces, como também para o processamento de imagens, no geral. Isso ocorre devido à sua semelhança com o funcionamento do *Visual Cortex* cerebral dos humanos, que é composto por milhões de grupos complexos de células sensíveis apenas a pequenas regiões do campo visual, chamadas de “campos receptivos”. Tais campos receptivos fazem parte da primeira entre três ideias básicas deste método, são estas: campos receptivos locais, pesos compartilhados e *pooling*. O primeiro conceito, de campos receptivos locais, trata-se de uma estratégia utilizando filtros para conectar a imagem de entrada, particionada em pixels, com uma camada escondida de neurônios (camada convolucional) (Figura 12). O segundo conceito fala a respeito dos pesos entre *inputs* e camada convolucional, estes são compartilhados, pois o método permite identificar um mesmo recurso da imagem da mesma forma para todas as partes desta, tornando-o invariante à rotação e translação. O terceiro conceito, de *pooling*,

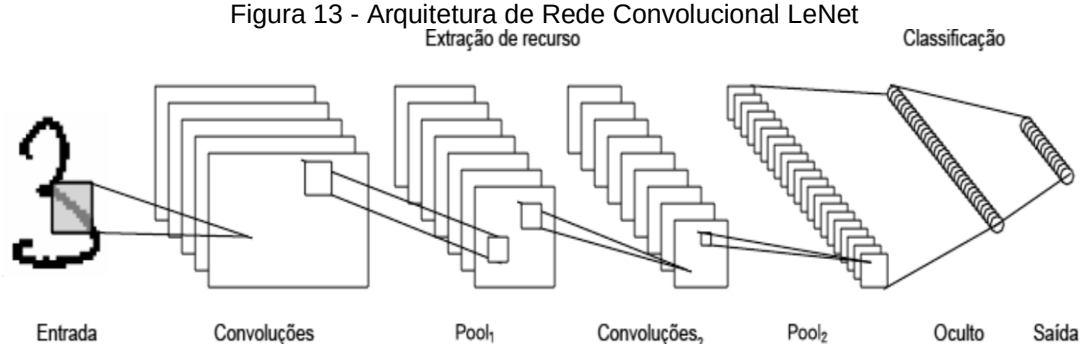
permite que a imagem seja reduzida, a partir do momento em que a importância de cada uma das partes da imagem é determinada, mantendo para a próxima camada apenas as partes importantes desta. Um exemplo de arquitetura de rede neural convolucional, arquitetura LeNet, apresentada por LeCun et. al (1998) pode ser vista na Figura 13, com camadas convolucionais e de *pooling.*

Figura 12 - Exemplo de funcionamento de Campos receptivos



Fonte: Data Science Academy (2019).

Figura 13 - Arquitetura de Rede Convolucional LeNet



Fonte: JONES (2017).

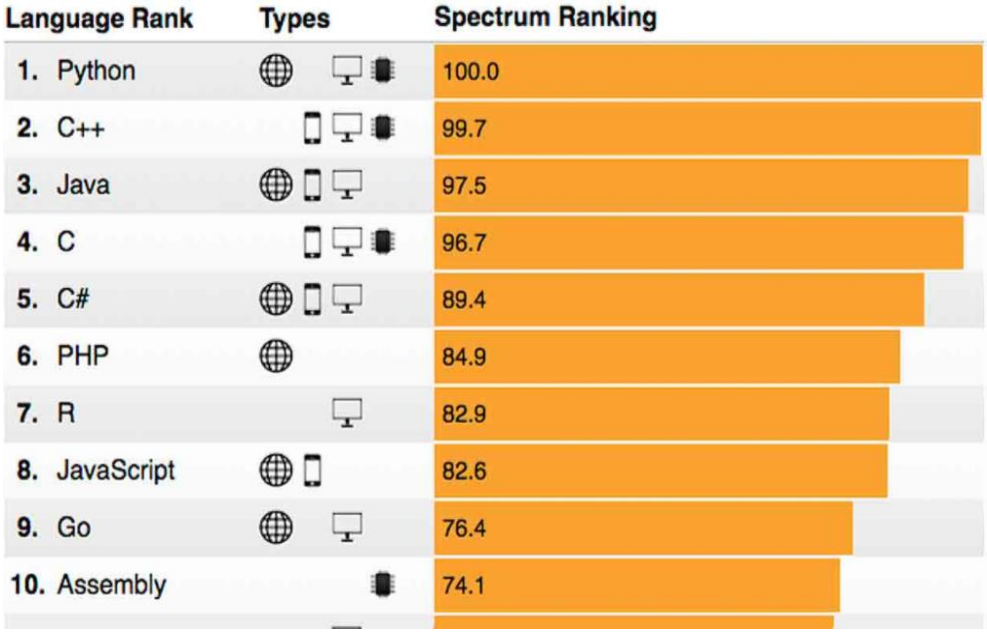
2.2 Python: O que é uma biblioteca?

> Python é uma linguagem de programação poderosa e fácil de aprender. Ela possui estruturas de dados de alto nível eficientes e uma abordagem simples mas eficaz para programação orientada a objetos. A sintaxe elegante e a tipagem dinâmica do Python, junto com sua natureza interpretada, fazem dele uma linguagem ideal para scripts e desenvolvimento rápido de aplicativos em muitas áreas na maioria das plataformas (PYTHON SOFTWARE FOUNDATION, 2019).

No último ano, a linguagem Python ficou em primeiro lugar no ranking de melhores linguagens de programação, de acordo com o *IEEE Spectrum's fifth annual interactive ranking of the top programming languages* (Figura 14). Esta expressividade pode ser explicada devido à sua flexibilidade, pois esta linguagem

pode ser utilizada tanto em *softwares* embarcados (já que, hoje em dia, existem muitos hardwares que suportam um interpretador Python) quanto para a manipulação de estatística e *big data* (devido às suas bibliotecas de alta qualidade) (CASS, 2018).

Figura 14 - IEEE Spectrum's fifth annual interactive ranking of the top programming languages



Fonte: CASS (2018).

A popularidade dessa linguagem é tão expressiva que grandes empresas de tecnologia, como *Google, Facebook, Instagram, Spotify, Netflix, Dropbox,* entre outras, a utilizam (REYNOLDS, 2019). Como exemplos dessa utilização temos, dentre outros: o *YouTube*, na *Google*, que utiliza essa linguagem em todo o site para ver e administrar vídeos, acessar dados canônicos, controlar modelos de sites, além de outros propósitos (QUINTAGROUP, 2016); a utilização na distribuição de binários, imagens de *hardware*, automação operacional e até em gerenciamento de infraestrutura, no *Facebook*, que tem o Python como sua terceira linguagem mais popular (KOMORN, 2016); o *Instagram* que também a utiliza, executando a maior implementação do mundo do *framework web* Django, que é escrito inteiramente em Python, devido à reputação de simplicidade e praticidade dessa linguagem.

Acrescente-se, também que o Python tem sido bastante utilizado, nos dias atuais, em aplicações que abordam Inteligência Artificial (IA), como em reconhecimento de letra recursiva, imagens em geral e faces, sendo este último o objeto da presente dissertação.

Algumas vantagens desta linguagem caracterizam-se pela: legibilidade, facilitando a programação e a tornando mais rápida e dinâmica; o extensivo suporte da biblioteca padrão, que conta com inúmeras ferramentas muito usadas que abrangem áreas como operações de *string*, *Internet*, ferramentas de serviços da *Web*, interfaces de sistema operacional e protocolos; os *designs* limpos das bibliotecas e a linguagem orientada a objeto, que aumentam a produtividade do programador; dentre outras vantagens.

Em contrapartida, essa linguagem possui algumas limitações também, como a velocidade de execução, por exemplo, visto que é uma linguagem que necessita de um interpretador ao invés de um compilador, tornando-a um pouco mais lenta que outras (MINDFIRE SOLUTIONS, 2017).

Python é um projeto *open source* que conta com a contribuição de uma comunidade de usuários ativos. Esses usuários, ao contribuírem, deixam disponíveis suas soluções para todos os outros usuários sob os termos de licença *open source* (PYTHON SOFTWARE FOUNDATION, 2019). Desta maneira, estas soluções (esses *softwares*) são disponibilizadas (os) através de módulos, pacotes e bibliotecas e podem ser encontrados (as) no repositório público *Python Packaging Index* (PYTHON SOFTWARE FOUNDATION, 2019). Esses pacotes podem ser instalados com o auxílio do *pip*, um instalador de preferência para o Python.

Um módulo (Figura 15) pode ser qualquer arquivo Python com a extensão "py". Enquanto, um pacote Python (Figura 16) é um conjunto de módulos contidos em um diretório, o qual possui um arquivo "__init__.py" que permite que o Python trate os diretórios deste diretório principal como pacotes (subpacotes) e evita diretórios com mesmo nome (PYTHON SOFTWARE FOUNDATION, 2019). Já uma biblioteca é um conjunto de módulos e pacotes.

Figura 15 - Exemplo de Módulo Python

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

Fonte: PYTHON SOFTWARE FOUNDATION (2019).

Figura 16 - Exemplo de pacote Python

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Fonte: PYTHON SOFTWARE FOUNDATION (2019).

2.3 Biblioteca “Face_recognition”

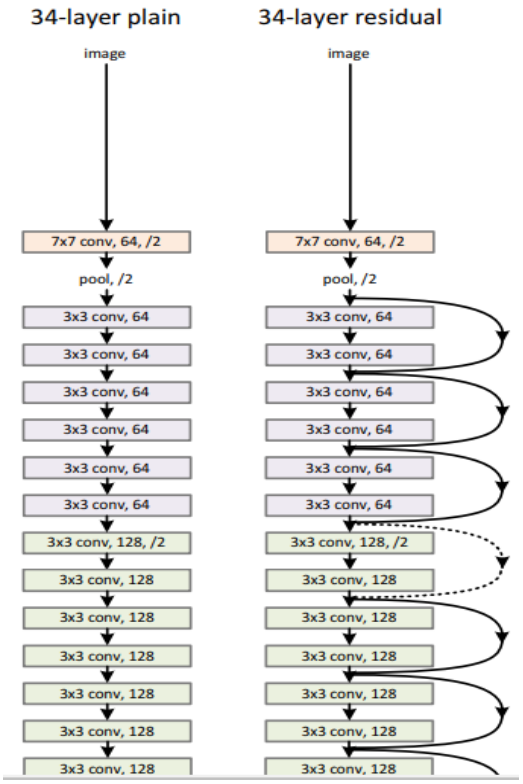
“Face_recognition” (PYTHON SOFTWARE FOUNDATION, 2018) é uma biblioteca *Python* de domínio público que oferece reconhecimento facial. Essa biblioteca necessita de outras bibliotecas e pacotes para que se possa usufruir do seu pleno funcionamento, onde os mais importantes são a “Dlib” (DLIB DEVELOPERS, 2019) e o “numpy” (NUMPY DEVELOPERS, 2019). A “Dlib” é uma biblioteca utilizada para algoritmos de *machine learning*, que é a base para a “face_recognition”. “Numpy” é um pacote para computação científica, com ele é possível utilizar *arrays* (vetores) de N dimensões.

A biblioteca “face_recognition” utiliza um modelo da biblioteca “Dlib” pré-treinado em uma rede ResNet (Rede residual) com 29 redes convolucionais. Esta

rede baseia-se essencialmente na ResNet-34 (3x3 filtros por camada e 34 redes convolucionais) proposta pelo artigo "Deep Residual Learning for Image Recognition by He, Zhang, Ren and Sun" (2016), diferenciando-se pela quantidade de camadas (algumas foram removidas) e pelo número de filtros por camada (reduzindo-se à metade).

A ResNet é assim chamada porque introduz atalhos a cada par de filtros 3x3 pertencentes à rede (Figura 17). Esses atalhos determinam uma ligação entre a rede e sua versão residual e, dessa forma, a convergência deste tipo de rede é mais rápida que sua versão não-residual. E quanto mais profunda (*deep system*), maior a acurácia, porém maior a complexidade da rede (HE et al., 2016).

Figura 17 - Parte da ResNet-34



Fonte: HE et al. (2016).

O princípio de funcionamento da biblioteca inicia-se com o uso do algoritmo HOG para a detecção das faces. Sequencialmente, as faces detectadas são alinhadas ou até mesmo deformadas, adequando-se a um outro modelo da biblioteca “Dlib” que utiliza os cantos do olhos e a ponta do nariz para o alinhamento do rosto. Por fim, o modelo de reconhecimento fornece os pesos à ResNet e esta recebe como entrada a face alinhada. Então, é gerado um numpy-array com 128 dimensões. Este numpy-array é utlizado para fazer o reconhecimento de faces e,

para isso, é escolhido um limiar para o qual a distância entre dois destes numpy-arrays pode ser considerada pertencente a uma mesma identidade.

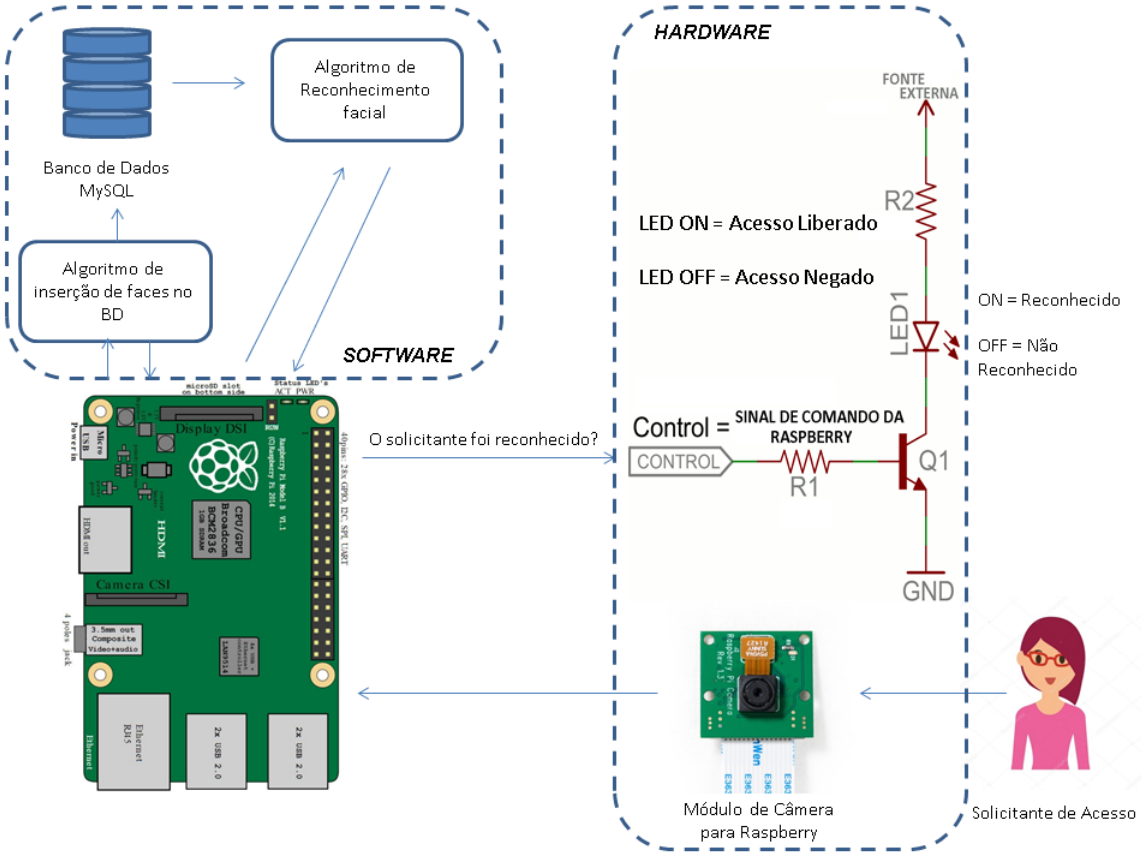
Além disso, este procedimento prevê uma acurácia de 99,38% no banco de dados padrão *LFW (Labeled Faces in the Wild)*.

3 MÉTODO

Dentro da metodologia utilizada podem ser destacados os seguintes pontos: inicialmente, foi realizado um estudo de literatura sobre o reconhecimento facial para o levantamento dos algoritmos relevantes para aplicação e implementação em linguagem Python; em seguida, após um estudo a respeito das bibliotecas disponíveis e capazes de realiza-lo, foi escolhida aquela que seria ideal para o desenvolvimento do software; posteriormente este foi implementado, primeiramente num computador e, sequencialmente, depois de alguns testes, no *hardware* escolhido (Raspberry); por fim, utilizou-se um circuito eletrônico para a simulação do acesso controlado e se realizou testes com o sistema completo, para a sua validação.

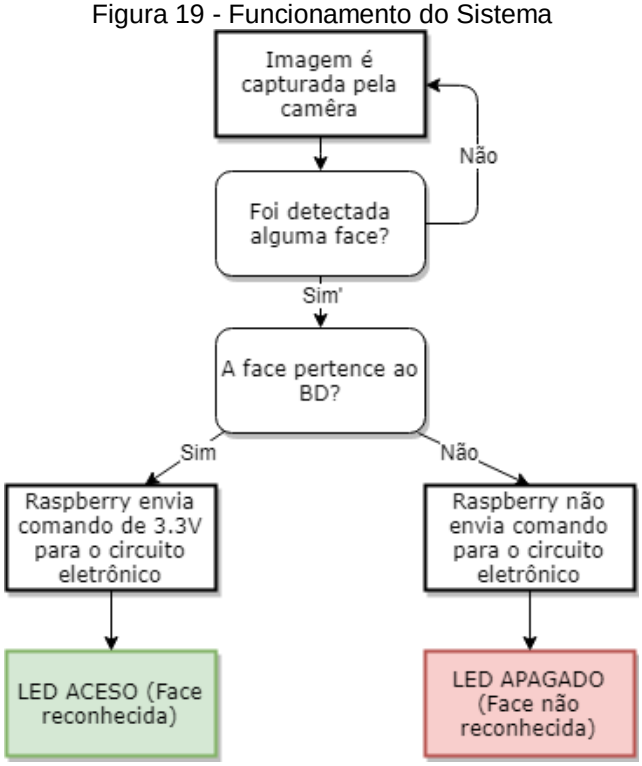
A arquitetura proposta encontra-se na Figura 18, e na Figura 19 encontra-se um fluxograma para sua melhor compreensão. Nos próximos capítulos será esclarecida toda a metodologia utilizada para a geração da proposta.

Figura 18 - Arquitetura da Proposta



Fonte: Elaboração do autor

Figura 19 - Funcionamento do Sistema



Fonte: Elaboração do autor

3.1 Estudo de Literatura sobre o Reconhecimento Facial

Inicialmente foi feito um estudo a respeito de literaturas sobre reconhecimento facial para o levantamento de algoritmos relevantes e, posteriormente, selecionar o algoritmo que seria implementado. Como o intuito era implementar esses algoritmos na linguagem Python, o estudo foi feito a partir das bibliotecas disponíveis para reconhecimento facial nesta linguagem.

Uma das bibliotecas Python mais populares para o processamento de imagens é a biblioteca “OpenCV”. Essa biblioteca possui inúmeros algoritmos para a detecção e o reconhecimento facial, incluindo: o algoritmo de Viola e Jones (para a detecção), um dos mais populares utilizados nos dias atuais; e para o reconhecimento, *eigenfaces*, *fisherfaces* e *local binary patterns histograms* (OPENCV, 2019). Por essa razão, a princípio, foram os primeiros algoritmos estudados.

Mesmo sendo muito popular por ter um desempenho bastante rápido, o algoritmo de Viola e Jones não é tão eficaz. Ele pode ocasionar muitos falsos positivos, ou seja, ele pode detectar que partes da imagem são rostos quando não

são realmente, e para determinadas posições de face ele não chega a reconhecer (falsos negativos).

Os algoritmos de reconhecimento desta biblioteca (o *eigenfaces* e o *fisherfaces*) consideram iluminação (luz e sombra) e mudanças de iluminação como importantes informações sobre a face, tornando-os não muito eficazes, pois é preciso estar em condições favoráveis de iluminação para que o reconhecimento seja feito. Já o algoritmo *local binary patterns histograms*, por não levar em conta a iluminação como essencial, poderia ser considerado.

Em resumo, essa biblioteca tem sua enorme popularidade devido ao tempo em que ela existe e, também, pela diversidade de ferramentas que ela possui para o processamento de imagens, inclusive *real-time camera*, ou seja, processamento de imagem de câmera e vídeo em tempo real, que são muito eficazes também, porém não aplicáveis diretamente para o caso estudado. Essa biblioteca ainda foi utilizada para a simulação no computador, para a visualização da imagem da câmera em tempo real.

Outras bibliotecas encontradas para o reconhecimento em Python foram a “Openface” e a “face_recognition”, ambas similares entre si quanto à metodologia, utilizando as mesmas bibliotecas necessárias para o funcionamento e redes neurais convolucionais para o reconhecimento.

O algoritmo utilizado pela biblioteca “Openface” (B. Amos, B. Ludwiczuk, M. Satyanarayanan, 2016) foi apresentado pela *Google* no artigo FaceNet (SCHROFF; KALENICHENKO; PHILBIN, 2015) e possui uma acurácia 99.63%±0.09 no banco de dados de faces *LFW* (Huang et al., 2007).

Já o algoritmo utilizado pela biblioteca “face_recognition” foi apresentado por Adam Geitgey. Essa biblioteca teve seu método de reconhecimento baseado na citada anteriormente (GEITGEY, 2016), por essa razão, ambas são similares. Porém, essa segunda possui uma implementação mais simples e acurácia de 99.38% no mesmo banco de dados utilizado pela outra.

No quadro 1 há uma análise dos algoritmos levantados nesta pesquisa. Nessa análise, as informações foram classificadas por cores: em verde, estão as informações que tornaram o algoritmo atrativo; em vermelho, estão as informações que tornaram o algoritmo não atraente; em amarelo, estão as informações pouco atraentes e, em branco, estão apenas informações adicionais para a análise.

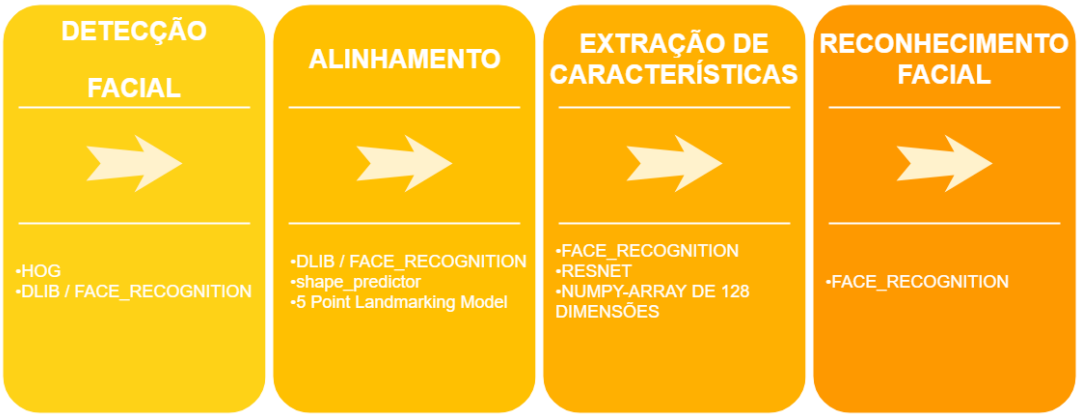
Quadro 1 - Análise dos Algoritmos estudados

Algoritmos	Viola e Jones	HOG	Eigenfaces	Fisherfaces	Redes Neurais convolucionais	Redes Neurais convolucionais
Biblioteca	OpenCV	Dlib	OpenCV	OpenCV	Openface	face_recognition
Tipo de algoritmo	Detecção Facial	Detecção Facial	Reconhecimento Facial	Reconhecimento Facial	Reconhecimento Facial	Reconhecimento Facial
Acurácia	Aproximadamente 31% (no BD *WIDER FACE*)	Aproximadamente 37% (no BD *WIDER FACE*)	47,7% de erro por extrapolação e alta luz direta	4,6% de erro por extrapolação e alta luz direta	99.63%±0.09 (no BD *LFW*)	99.38% (no BD *LFW*)
Facilidade de implementação	Fácil	Fácil	Fácil	Fácil	Médio	Fácil

Fonte(s): BELHUMEUR; HESPANHA; KRIEGMAN, 1996; DWIYANTORO, 2018; GEITGEY, 2016; B. Amos, B. Ludwiczuk, M. Satyanarayanan, 2016.

Após a análise, por possuírem desempenhos semelhantes, a biblioteca selecionada foi a “face_recognition” devido à sua fácil implementação. Assim sendo, na Figura 20 será apresentado o processo de reconhecimento, com os algoritmos selecionados.

Figura 20 - Processo de Reconhecimento Facial



Fonte: Elaboração da autora

3.2 Implementação do face_recognition

A princípio, a implementação do algoritmo foi feita num computador com sistema operacional Linux, sistema semelhante ao utilizado na raspberry pi,

hardware no qual seria feita a implementação, posteriormente. É importante ressaltar que é informado que o módulo é compatível com o hardware escolhido.

A versão do Python utilizada foi a mais recente, Python 3.3+, como recomendado, mas também poderia ser utilizada a versão Python 2.7, igualmente compatível de acordo com os requisitos do módulo. Seguiu-se o passo a passo para a instalação de acordo com as instruções e requisitos da biblioteca (GEITGEY, 2018), instalando-se todos os módulos e bibliotecas necessários como a “Dlib”, o “numpy”, e a “OpenCV” (essa última, como citado no tópico anterior, foi utilizada para a visualização das imagens da câmera e verificação da detecção e reconhecimento facial na simulação em computador).

A sequência de comandos utilizados para a instalação dos módulos no sistema operacional Linux, foram:

1. Clonar o código do github (repositório de códigos):

   ```
   git clone https://github.com/davisking/dlib.git
   ```

2. Criar dentro do diretório dlib, clonado, a biblioteca principal dlib:

   ```
   sudo apt-get install cmake
   cd dlib
   mkdir build; cd build; cmake ..; cmake --build .
   ```

3. Instalar as extensões Python para a versão utilizada (Python 3.3+):

   ```
   cd ..
   sudo python3 setup.py install
   sudo python2 setup.py install (caso utilize a versão 2.7 do Python)
   ```

4. Instalar a biblioteca face_recogntion:

   ```
   pip3 install face_recognition
   pip2 install face_recognition (caso utilize a versão 2.7 do Python)
   ```

5. Instalar a bibioteca OpenCV:

   ```
   sudo apt-get install python3-opencv
   sudo apt-get install python-opencv (caso utilize a versão 2.7 do Python)
   ```

6. Para a criação do banco de dados (LINUXCONFIG, 2018):
   6.1. Instalar mysql-server:

   ```
   sudo apt install mysql-server
   ```

   6.2. Fazer instalação do MySQL:

   ```
   sudo mysql_secure_installation
   ```

   6.3. Editar mysqld.cnf e mude bind-address para 0.0.0.0 para acessar o MySQL-server usando MySQL-client:

   ```
   sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
   ```

   6.4. Reinicializar o serviço:

   ```
   sudo service mysql restart
   ```

   6.5. Criar senha para o usuário root:

   ```
   sudo mysql -u root ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'SuaSenha';
   ```

7. Para conectar o MySQL-server com o Python:

   ```
   sudo apt-get install libmysqlclient-dev python-dev
   export PATH=$PATH:/usr/local/mysql/bin/
   pip3 install mysqlclient --user
   ```

Após instalação dos módulos necessários, a implementação foi feita levando-se em consideração o arquivo de exemplo disponibilizado (Anexo B) e a documentação da biblioteca (GEITGEY, 2017). Com o algoritmo funcionando, introduziu-se também, um banco de dados para o armazenamento das informações das faces, as quais seriam reconhecidas e foram utilizadas nos testes sequencialmente.

O banco de dados criado foi do tipo MySQL (Figura 21) e para essa implementação, no computador, foi utilizada a ferramenta *Workbench* (Figura 22). Essa ferramenta permite a criação e manipulação de bancos de dados do tipo MySQL de forma mais simplificada e ágil. Através dela, criou-se um banco de dados no qual foi inserida uma tabela (Quadro 2) e nessa tabela foram definidos três tipos de dados: id (“inteiro”), nome (“varchar (40)”) e o vetor *encoding* (“BLOB”). O dado “id” foi configurado como chave primária e não nulo (*not null*), o “nome” e “encoding” foram configurados, ambos, como não nulos também. Após isso, pôde-se inserir dados no banco de dados para os testes de software.

Figura 21 - Processo de Criação do Banco de Dados MySQL

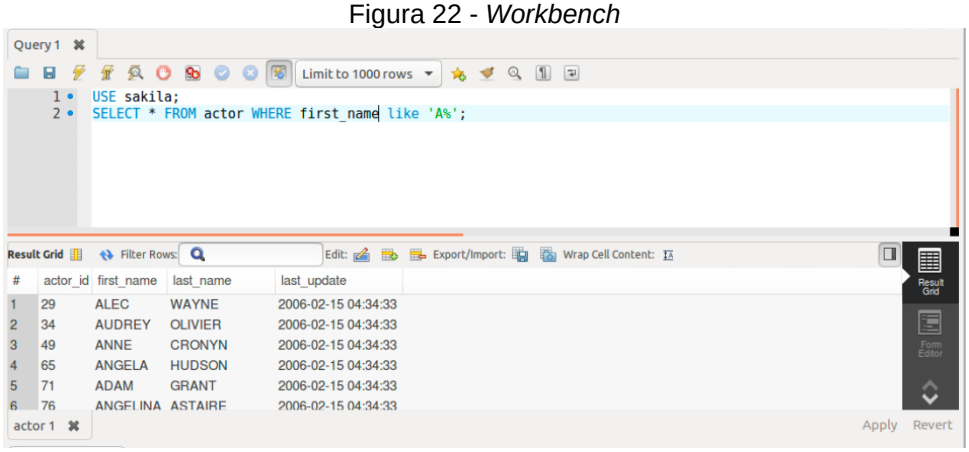
Fonte: Criação da autora

Quadro 2 - Tabela criada no Banco de Dados

	id	nome	encoding
Tipo	INT	VARCHAR(40)	BLOB
Nulo	NÃO NULO	NÃO NULO	NÃO NULO
Chave Primária	SIM	NÃO	NÃO
Auto Incremento	SIM	NÃO	NÃO

Fonte: Elaboração da autora no Software Microsoft Excel

Figura 22 - *Workbench*



Fonte: LINODE (2018).

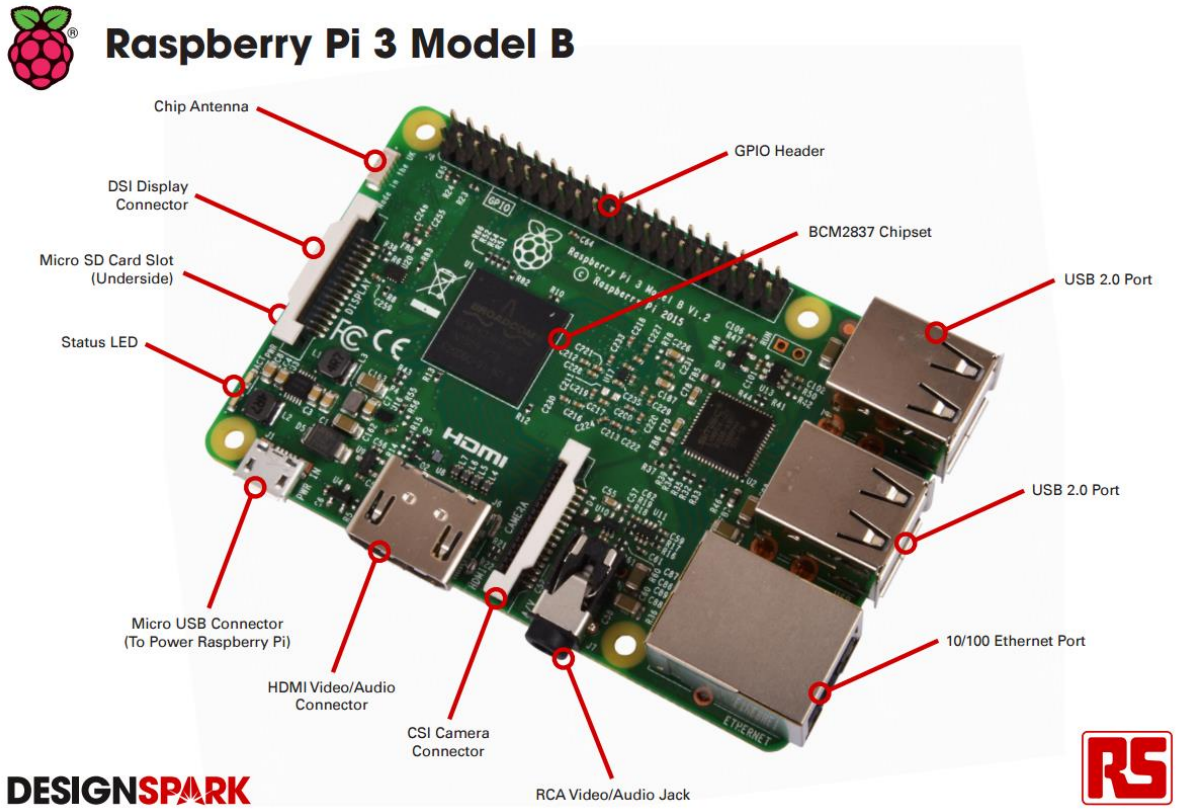
3.3 Implementação do Hardware

3.3.1 Raspberry Pi 3 Modelo B

Após os testes realizados através do computador, implementou-se o código numa raspberry pi 3 modelo B (Figura 23), fazendo-se os ajustes necessários. Seguiu-se o mesmo processo para a instalação dos módulos necessários, porém respeitando as particularidades do hardware. Para isso, foi necessário seguir as instruções de instalação presentes nos requisitos da biblioteca, as quais são fornecidas, de forma clara e explicada, para a raspberry pi, especificamente. Foi necessária também, a instalação de um módulo de câmera para raspberry (Figura

24), o qual se conectou a esta através de um cabo *flat*. Nessa implementação, não foi necessária a instalação da biblioteca OpenCV.

Figura 23 - Raspberry Pi 3 Modelo B



Fonte: SMITH (2016).

Figura 24 - Pi Câmera

Fonte: AMAZON (2019?).

Os comandos necessários para a instalação dos módulos na raspberry pi foram (GEITGEY, 2017):

1. Instalar bibliotecas auxiliaries:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install build-essential \
    cmake \
    gfortran \
    git \
    wget \
    curl \
    graphicsmagick \
    libgraphicsmagick1-dev \
    libatlas-dev \
    libavcodec-dev \
    libavformat-dev \
    libboost-all-dev \
    libgtk2.0-dev \
    libjpeg-dev \
    liblapack-dev \
    libswscale-dev \
    pkg-config \
    python3-dev \
    python3-numpy \
    python3-pip \
    zip
sudo apt-get clean
```

2. Instalar a biblioteca para o módulo de câmera com suporte para *arrays* (vetores):

```
sudo apt-get install python3-picamera

sudo pip3 install --upgrade picamera[array]
```

3. Baixar a versão v19.6 da biblioteca Dlib:

```
mkdir -p dlib
```

```
git clone -b 'v19.6' --single-branch https://github.com/davisking/dlib.git dlib/
cd ./dlib
sudo python3 setup.py install --compiler-flags "-mfpu=neon"
```

4. Instalar a biblioteca face_recognition:
   ```
   sudo pip3 install face_recognition
   ```

5. E, para baixar e testar o código de exemplo (Anexo A), pode-se utilizar, também, os comandos:
   ```
   git clone --single-branch https://github.com/ageitgey/face_recognition.git
   cd ./face_recognition/examples
   python3 facerec_on_raspberry_pi.py
   ```

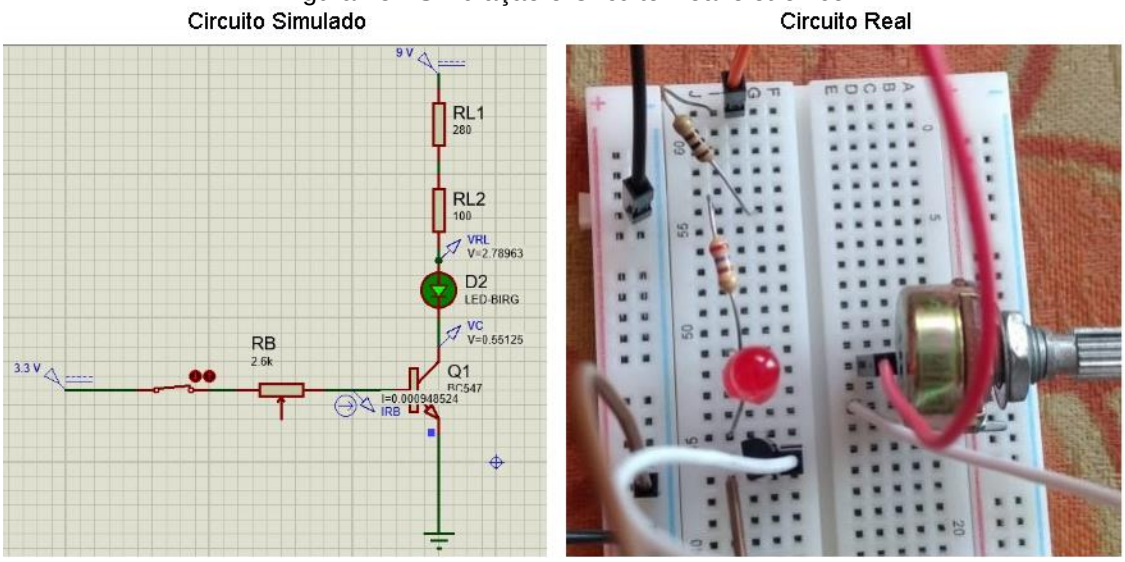
As instruções utilizadas para a instalação do banco de dados foram as mesmas que foram citadas anteriormente para o computador. Entretanto, a criação do banco não foi possível através da ferramenta *Workbench* e, por essa razão, o banco de dados foi elaborado através de comandos no terminal, utilizando os mesmos comandos do *Workbench*. E, neste caso, criou-se um usuário para a raspberry pi, não se utilizando o usuário root como na aplicação no computador, seguindo como base um projeto exemplo de criação de banco de dados MySQL, existente no site da comunidade raspberry (IBEX, 2013?).

3.3.2 Simulação da Porta de Acesso

Em seguida, foram feitos testes através de um circuito eletrônico, implementado numa protoboard, utilizando-se um led, dois resistores (280Ω e 100Ω, ambos em série, o equivalente a um único resistor de 380Ω), um potenciômetro (5kΩ, porém calibrado para 2,6kΩ), um transistor npn (BC547) e uma fonte de 9V; para simular a fechadura de uma porta. Este circuito foi, inicialmente, projetado e simulado no *software* Proteus 8 (Figura 25).

Sabendo-se que a saída dos pinos de comando da raspberry tem uma tensão de 3,3V, na simulação utilizou-se uma fonte de 3,3V e uma chave para o comando emitido quando fosse reconhecida uma face. Para a verificação do reconhecimento, o led do circuito simulou a fechadura de uma porta.

Figura 25 - Simulação e Circuito Real eletrônico



Fonte: *Software* Proteus 8; fotografia realizada pela autora.

4 RESULTADOS

Como explicado nos tópicos anteriores desta monografia, para o reconhecimento facial feito através de redes neurais, como é o caso da biblioteca “face_recognition”, é necessário definir qual o limiar ideal para que o reconhecimento seja feito, eliminando falsos positivos e descartando o mínimo de falsos negativos. Desta forma, quatro testes foram feitos para a escolha deste limiar. Foi considerado que é citado na documentação que o melhor limiar é em torno de 0,6.

4.1 Teste 1

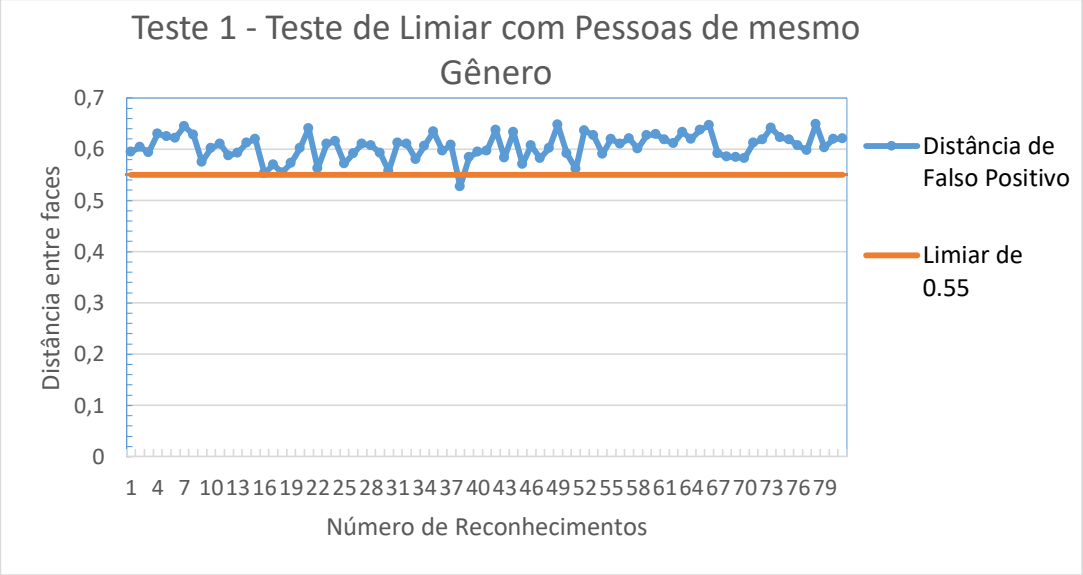
Neste teste, duas pessoas semelhantes foram utilizadas para verificar o limiar que mais se adequava ao desejado. Utilizaram-se duas pessoas com o mesmo gênero e com laços familiares (Figura 26).

Figura 26 - Pessoas utilizadas nos Testes 1 e 2

Fonte: Compilação da autora

Foram feitos 101 reconhecimentos, com o algoritmo implementado, pela captura da face, em tempo real, de uma das pessoas-teste, através da câmera. A partir de então, obteve-se todas as distâncias de falsos positivos (caso em que a pessoa é reconhecida como outra) dentro de um limiar de 0,65 (Gráfico 1 e Tabela 1).

Gráfico 1 - Teste de Limiar com Pessoas de mesmo Gênero



Fonte: Gráfico gerado no software Microsof Excel.

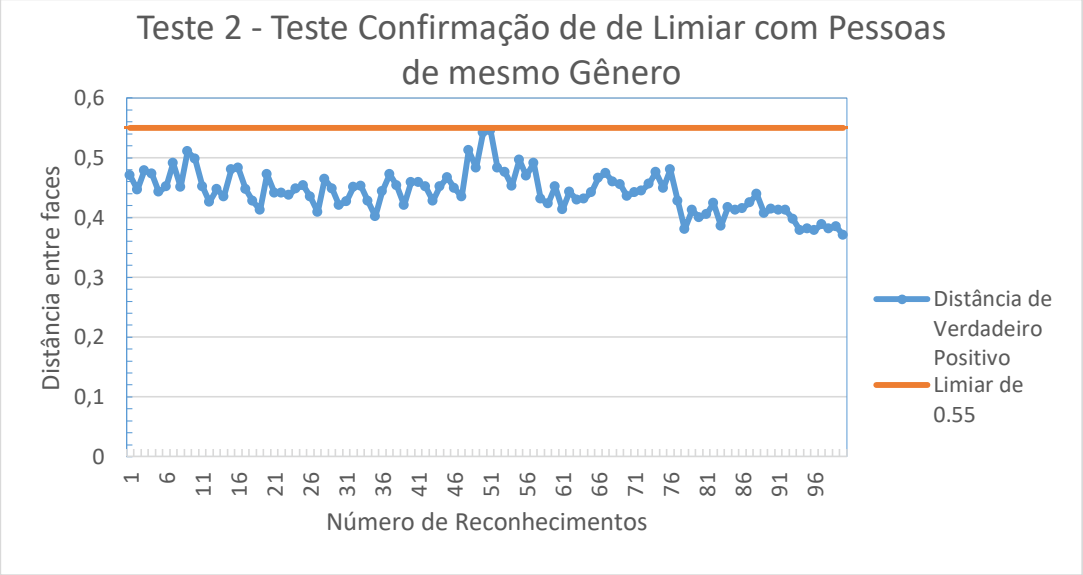
Através do Gráfico 1 é possível perceber que 81 dos reconhecimentos feitos foram falsos positivos. Com o intuito de manter uma maior velocidade de reconhecimento e anular os falsos positivos, obteve-se um limiar de 0,55, eliminando 98,76% dos falsos positivos.

Desta maneira, o novo limiar obtido (de 0,55) foi utilizado nos testes 2 e 3.

4.2 Teste 2

Utilizando-se as mesmas pessoas do teste anterior (Figura 26), porém modificando-se o limiar para o obtido no teste 1, repetiu-se o teste com a mesma pessoa-teste sendo capturada pela câmera. Porém, desta vez, todos os falsos positivos foram eliminados e obteve-se as distâncias reais.

Gráfico 2 - Teste Confirmação de de Limiar com Pessoas de mesmo Gênero



Fonte: Gráfico gerado no software Microsof Excel.

Através do Gráfico 2, é possível visualizar as distâncias reais as quais a pessoa-teste foi reconhecida corretamente para 100 dos 101 reconhecimentos. Numa única vez, a pessoa-teste não pôde ser reconhecida, considerando-se um percentual de 0,99% de falso negativo.

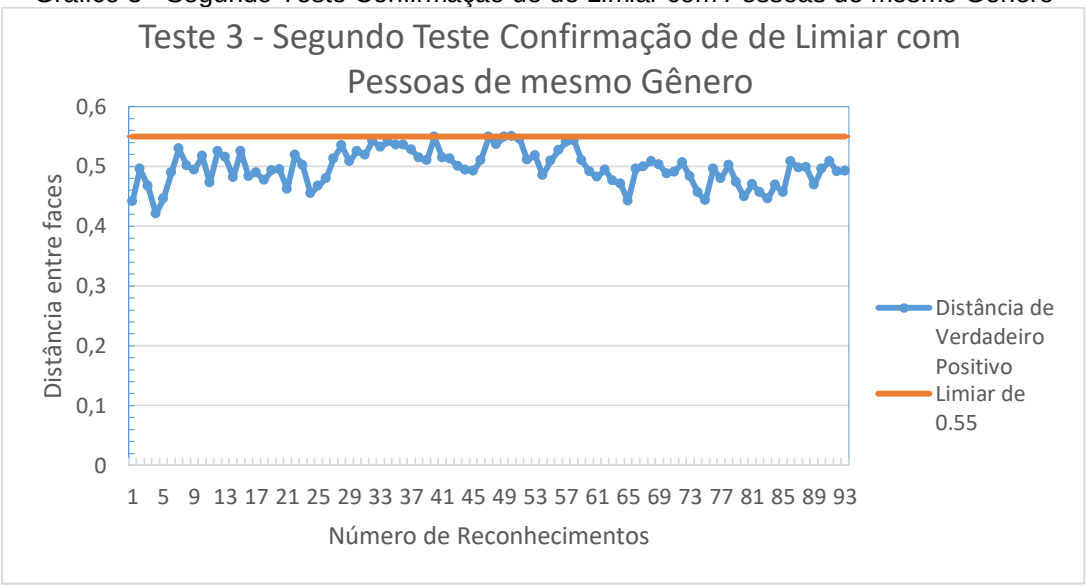
4.3 Teste 3

Um teste semelhante ao teste 2 foi realizado, diferenciando-se que, neste caso, trocou-se a pessoa-teste a ser capturada pela câmera (Figura 27) e obteve-se as distâncias reais para as quais esta foi reconhecida corretamente. Os falsos positivos também foram eliminados, porém, o número de falsos negativos aumentou para 7,92%.

Figura 27 - Pessoas utilizadas no Teste 3

Fonte: Compilação da autora.

Gráfico 3 - Segundo Teste Confirmação de de Limiar com Pessoas de mesmo Gênero



Fonte: Gráfico gerado no software Microsof Excel.

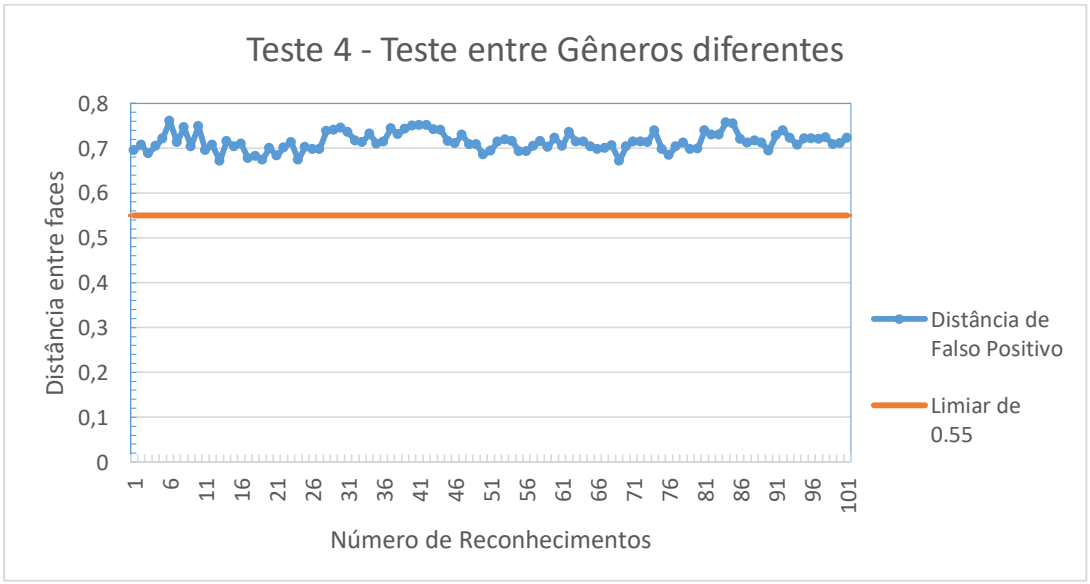
4.4 Teste 4

Neste teste foram utilizadas duas pessoas de gêneros diferentes (Figura 28) para verificar a possibilidade de serem reconhecidas como a mesma pessoa. Essa verificação foi feita obtendo-se todas as distâncias entre a imagem da pessoa capturada pela câmera (pessoa-teste do sexo feminino) e a outra pessoa (pessoa-teste do sexo masculino), esta teve os dados inseridos no banco de dados.

Figura 28 - Pessoas utilizadas no Teste 4

Fonte: Compilação da autora.

Gráfico 4 - Teste entre Gêneros diferentes



Fonte: Gráfico gerado no software Microsof Excel.

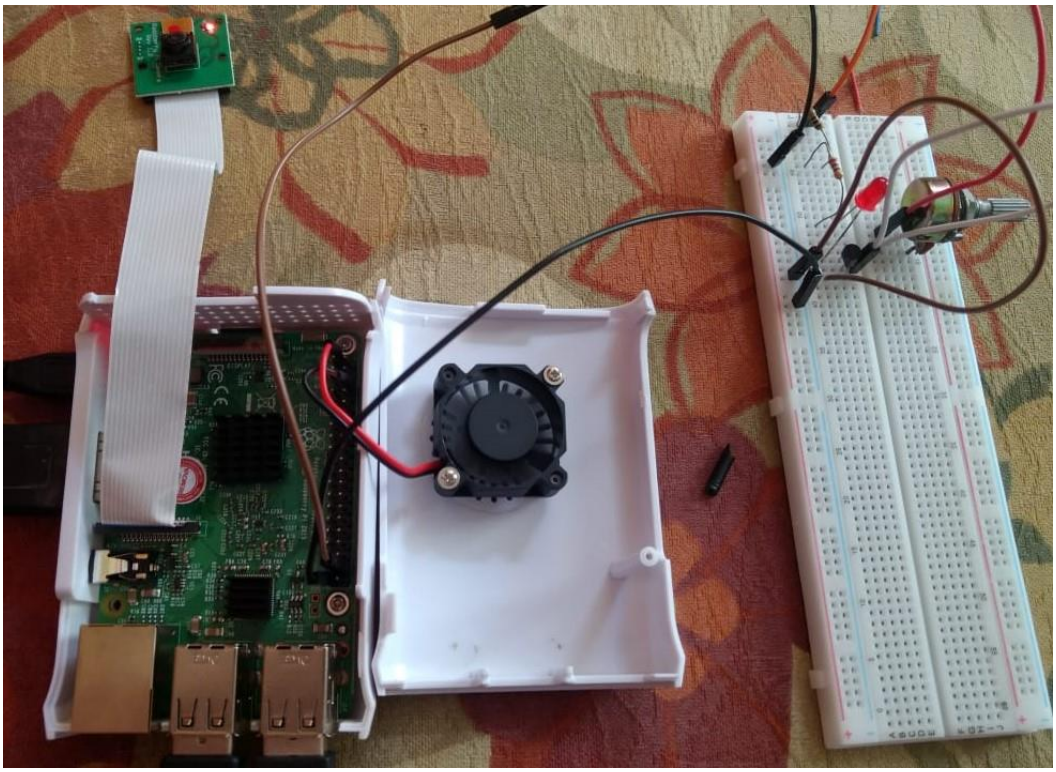
Calculando-se a média entre as menores distâncias obtidas na tabela 4 obtêm-se uma distância de 0,678, que ultrapassa o limiar escolhido (0,55) em 23,3%. Esta percentagem torna o algoritmo mais confiável ao diferenciar gêneros sexuais. Porém, vale ressaltar que esse valor não é absoluto para todos os reconhecimentos entre gêneros diferentes. Esta análise serve apenas para testar o desempenho do algoritmo numa amplitude maior de cenários.

Alguns outros testes, incluindo uso de acessórios, como óculos, foram realizados, mas não foram registrados. Nesses casos, o algoritmo conseguiu identificar com sucesso as faces.

4.5 Testes de Hardware

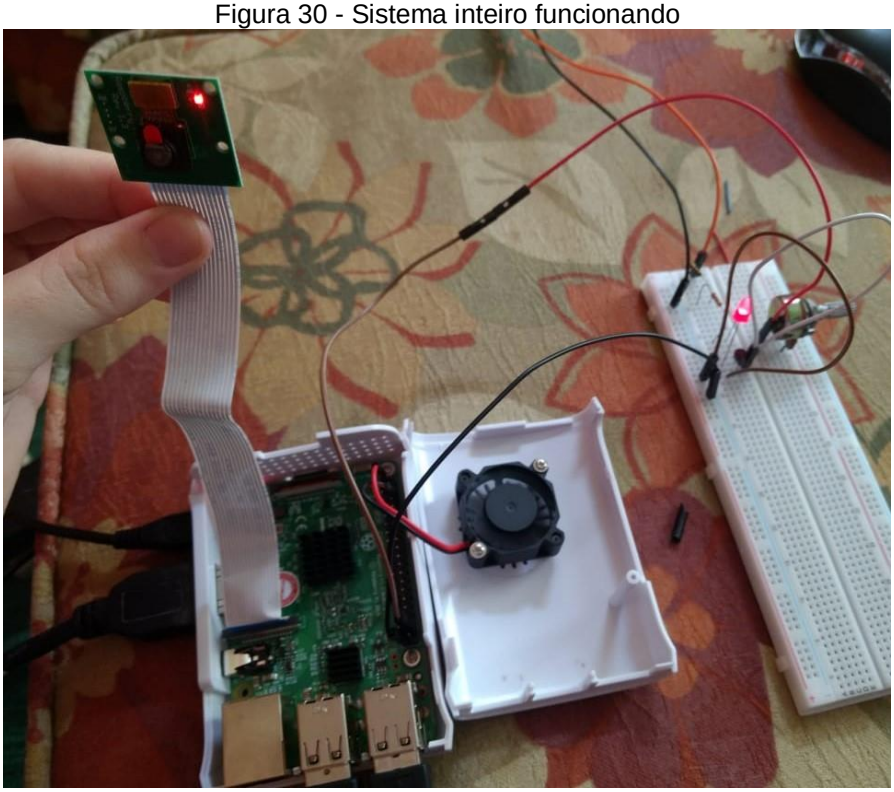
Como citado no tópico sobre a implementação do *hardware*, foram feitos testes com o circuito eletrônico simulado. Os testes foram realizados com o sistema inteiro em funcionamento (Figura 29).

Figura 29 - Sistema para teste



Fonte: Foto realizada pela autora.

Figura 30 - Sistema inteiro funcionando



Fonte: Foto realizada pela autora.

Na Figura 30 tem-se o sistema inteiro funcionando em conjunto, hardware e *software*, após o reconhecimento da face conhecida. É possível verificar-se o led do circuito eletrônico brilhando aceso, expressando o reconhecimento e validando o funcionamento do sistema.

5 CONCLUSÃO

Diante de tudo que aqui foi exposto, conclui-se que é perfeitamente possível a elaboração, a baixo custo, de um algoritmo implementado numa raspberry pi 3 Modelo B, desenvolvido na linguagem Python e com o auxílio da biblioteca “face_recognition”, que seja capaz de controlar o acesso de pessoas utilizando como chave as suas características biométricas.

Conclui-se também que esse método de reconhecimento facial além de representar um avanço tecnológico de suma importância para a sociedade (devido à facilidade de captura de características faciais e à praticidade por ele proposta), possui uma ampla variedade de aplicações nos mais diversos setores, posto que representa segurança e economia para aqueles que o utilizam, sendo certa e indiscutível sua futura disseminação.

Como sugestão, para garantir maior precisão dos resultados, outros testes poderão ser feitos utilizando uma diversidade maior de dados. Da mesma forma, para facilitar o uso do administrador, poderá ser elaborada uma interface amigável para a inserção de dados no Banco de Dados, via *web*, por exemplo.

Seria interessante, também, uma comparação entre a metodologia ora utilizada e outros algoritmos como, por exemplo, a biblioteca “Openface”, já citada anteriormente.

REFERÊNCIAS

ALVES, Bruno Alexandre Fernandes *et al*. Inteligência Artificial: Conceitos e Aplicações. **Revista Conexão Eletrônica**, Três Lagoas, MT, p. 907-918, 2018. Disponível em: file:///C:/Users/GRA%C3%87AWS/Downloads/93-INTELIGENCIA-ARTIFICIAL-Conceitos-e-Aplica%C3%A7%C3%B5es.pdf. Acesso em: 20 jun. 2019.

AMAZON. **Raspberry Pi Camera Board**. [*S. l.*], 2019?. Disponível em: https://www.amazon.in/Raspberry-Pi-Camera-Board/dp/B00L1FOIIS#customerReviews. Acesso em: 21 jun. 2019.

ANDRADE, Rodrigo de Oliveira. Visão Computacional. AS Máquinas que tudo veem. **Revista: Pesquisa FAPESP**, São Paulo, 24 jun. 2019. Disponível em: https://revistapesquisa.fapesp.br/2019/03/14/as-maquinas-que-tudo-veem/. Acesso em: 20 jun. 2019.

B. Amos, B. Ludwiczuk, M. Satyanarayanan. **"Openface: A general-purpose face recognition library with mobile applications".** CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

BAKSHI, Urvashi; SINGHAL, Mr. Rohit. A SURVEY ON FACE DETECTION METHODS AND FEATURE EXTRACTION TECHNIQUES OF FACE RECOGNITION. **International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)**, [*S. l.*], p. 233-237, maio-junho 2014. Disponível em: https://pdfs.semanticscholar.org/4f9a/dc4d6aa78b1a882a517a927bb45e5f38e407.pdf. Acesso em: 19 mar. 2019.

BELHUMEUR, Peter N.; HESPANHA, João P.; KRIEGMAN, David J. Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. **IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE**, [*S. l.*], p. 711-720, 15 fev. 1996. Disponível em: https://ieeexplore.ieee.org/document/598228. Acesso em: 8 jul. 2019.

CASS, Stephen. The 2018 Top Programming Languages. **I.E.E.E. Spectrum**, [*S. l.*], 31 jul. 2018. Disponível em: https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages). Acesso em: 10 jun. 2019.

Data Science Academy. **Deep Learning Book**, 2019. Disponível em: http://www.deeplearningbook.com.br/. Acesso em: 02 Julho. 2019.

DLIB DEVELOPERS. **Dlib C++ Library**. [*S. l.*], 2019. Disponível em: http://dlib.net/. Acesso em: 8 jun. 2019.

DWIYANTORO, Alvin Prayuda Juniarta. Performance Showdown of Publicly Available Face Detection Model. **Medium Corporation**, [*S. l.*], p. 1, 30 abr. 2018. Disponível em: https://medium.com/nodeflux/performance-showdown-of-publicly-available-face-detection-model-7c725747094a. Acesso em: 8 jul. 2019.

Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. **University of Massachusetts**, Amherst, Technical Report 07-49, October, 2007

GEITGEY, Adam. Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks. **Medium Corporation**, [*S. l.*], 13 jun. 2016. Disponível em: https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721. Acesso em: 20 jun. 2019.

GEITGEY, Adam. Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning. **Medium Corporation**, [S. l.], p. 1, 24 jul. 2016. Disponível em: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78. Acesso em: 4 maio 2019.

GEITGEY, Adam. **Face Recognition**. [*S. l.*], 2017. Disponível em: https://face-recognition.readthedocs.io/en/latest/readme.html. Acesso em: 8 jun. 2019.

GEITGEY, Adam. **Face_recognition 1.2.3**. [*S. l.*], 2018. Disponível em: https://pypi.org/project/face_recognition/. Acesso em: 18 maio 2019.

GEITGEY, Adam. **Install dlib and face_recognition on a Raspberry Pi**. [*S. l.*], 2017. Disponível em: https://gist.github.com/ageitgey/1ac8dbe8572f3f533df6269dab35df65. Acesso em: 20 jun. 2019.

HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing, SUN, Jian. Deep Residual Learning for Image Recognition. **IEEE Xplore**, Las Vegas, NV, USA, 12 dez. 2016. Disponível em: https://ieeexplore.ieee.org/document/7780459. Acesso em: 1 jun. 2019.

HEATH, Nick. Inside the Raspberry Pi: The story of the $35 computer that changed the world. **Tech Republic**, [S. l.], 19 dez. 2018. Disponível em: https://www.techrepublic.com/article/inside-the-raspberry-pi-the-story-of-the-35-computer-that-changed-the-world/. Acesso em: 18 jun. 2019.

IBEX. **MySQL**. [*S. l.*], [2013?]. Disponível em: https://raspberry-projects.com/pi/software_utilities/web-servers/mysql. Acesso em: 21 jun. 2019.

JONES, M Tim. Um guia para iniciantes sobre inteligência artificial, aprendizado de máquina e computação cognitiva. **IBM Developer**, [*S. l.*], p. 1-10, 1 jun. 2017. Disponível em: https://www.ibm.com/developerworks/br/library/guia-iniciantes-ia-maquina-computacao-cognitiva/index.html. Acesso em: 20 jun. 2019.

KOMORN, Romain. Python in production engineering. **Production Engineering**, [*S. l.*], 27 maio 2016. Disponível em: https://code.fb.com/production-engineering/python-in-production-engineering/. Acesso em: 10 jun. 2019.

LECUN, Yann *et al*. Gradient-based learning applied to document recognition. **I.E.E.E.**, [*S. l.*], p. 1-46, 1998. Disponível em: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf. Acesso em: 18 jun. 2019.

LINODE. **Install and Configure MySQL Workbench on Ubuntu 16.04**. [*S. l.*], 1 jun. 2018. Disponível em: https://www.linode.com/docs/databases/mysql/install-and-configure-mysql-workbench-on-ubuntu/. Acesso em: 20 jun. 2019.

LINUXCONFIG. **Install MySQL on Ubuntu 18.04 Bionic Beaver Linux**. [*S. l.*], 2018. Disponível em: https://linuxconfig.org/install-mysql-on-ubuntu-18-04-bionic-beaver-linux. Acesso em: 8 jun. 2019.

MARKOFF, John. Smaller, Faster, Cheaper, Over: The Future of Computer Chips. **The New York Times**, [*S. l.*], 26 set. 2015. Disponível em: https://www.nytimes.com/2015/09/27/technology/smaller-faster-cheaper-over-the-future-of-computer-chips.html. Acesso em: 18 jun. 2019.

MARTINS, Isabela. Válvulas e a Primeira Geração de Computadores. **De Boa na Rede**, [*S. l.*], [2015?]. Disponível em: http://dboanarede.blogspot.com/2014/08/valvulas_26.html. Acesso em: 20 jun. 2019.

MATHWORKS. Image Processing Toolbox. **MathWorks**, [*S. l.*], [2019?] Disponível em: https://fr.mathworks.com/help/images/integral-image.htm. Acesso em: 12 jun. 2019.

MINDFIRE SOLUTIONS. Advantages and Disadvantages of Python Programming Language. **Medium Corporation**, [*S. l.*], 24 abr. 2017. Disponível em: https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121. Acesso em: 12 jun 2019.

MOHAPATRA, Neha. Evolution of Computers. **Nettantra Crafting Excellence**, [*S. l.*], 31 jan. 2015. Disponível em: https://www.nettantra.com/2015/01/evolution-computers/. Acesso em: 20 jun. 2019.

MOREIRA, Eduardo. Intel 4004, o primeiro processador da história, comemora 40 anos de idade. **Tech Tudo**, [*S. l.*], p. 1-10, 16 nov. 2011. Disponível em: https://www.techtudo.com.br/artigos/noticia/2011/11/intel-4004-o-primeiro-processador-da-historia-comemora-40-anos-de-idade.html. Acesso em: 18 jun. 2019.

N., Dalal; B., Triggs. Histograms of Oriented Gradients for Human Detection. In: COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2., 2005, San Diego. **Proceedings... .** San Diego: Ieee, 2005. v. 1, p. 886 - 893. Disponível em: https://ieeexplore.ieee.org/document/1467360. Acesso em: 18 jun. 2019.

NEXO JORNAL. Desenvolver máquinas que pensam por conta própria é um dos desafios da nossa época, e a questão já é vista com temor por pesquisadores e políticos. **Nexo Jornal**, [*S. l.*], 2019. Disponível em: https://www.nexojornal.com.br/explicado/2017/02/07/Intelig%C3%AAncia-artificial-entre-a-pr%C3%B3xima-revolu%C3%A7%C3%A3o-tecnol%C3%B3gica-e-o-fim-da-humanidade. Acesso em: 20 jun. 2019.

NUMPY DEVELOPERS. **NumPy**. [*S. l.*], 2019. Disponível em: https://www.numpy.org/index.html. Acesso em: 15 jun. 2019.

OPENCV. **OpenCV**. [*S. l.*], 2019. Disponível em: https://opencv.org/. Acesso em: 4 maio 2019.

OPTICAL SOCIETY OF AMERICA A. Low-dimensional procedure for the characterization of human faces. **Optical Society of America**, Rhode Island, mar. 1987. Disponível em: http://www.face-rec.org/interesting-papers/General/ld.pdf. Acesso em: 10 jun. 2019.

PIROPO, B. Os primeiros transistores. **Tech Tudo**, [*S. l.*], 11 out. 2012. Disponível em: https://www.techtudo.com.br/artigos/noticia/2012/10/os-primeiros-transistores.html. Acesso em: 20 jun. 2019.

PYTHON SOFTWARE FOUNDATION. Modules. **Python Software Foundation**, [*S. l.*], 1 jan. 2019. Disponível em: https://docs.python.org/3/tutorial/modules.html#packages. Acesso em: 10 jun. 2019.

PYTHON SOFTWARE FOUNDATION. Face_recognition 1.2.3. **Python Software Foundation**, [*S. l.*], 21 ago. 2018. Disponível em: https://pypi.org/project/face_recognition/. Acesso em: 10 jun. 2019.

PYTHON SOFTWARE FOUNDATION. Find, install and publish Python packages with the Python Package Index. **Python Software Foundation**, [*S. l.*], 1 jan. 2019. Disponível em: https://pypi.org/. Acesso em: 10 jun. 2019.

PYTHON SOFTWARE FOUNDATION. Installing Python Modules. **Python Software Foundation**, [*S. l.*], [2019?]. Disponível em: https://docs.python.org/3/installing/index.html#installing-index. Acesso em: 10 jun. 2019.

PYTHON SOFTWARE FOUNDATION. Python 3.7.4rc1 documentation. **Python Software Foundation**, [*S. l.*], 1 jan. 2019. Disponível em: https://docs.python.org/3/. Acesso em: 10 jun. 2019.

PYTHON SOFTWARE FOUNDATION. The Python Tutorial. **Python Software Foundation**, [*S. l.*], [2019?]. Disponível em: https://docs.python.org/3/tutorial/index.htm. Acesso em: 10 jun. 2019.

QUINTAGROUP. PYTHON AT GOOGLE. **Quintagroup**, [*S. l.*], [2016?]. Disponível em: https://quintagroup.com/cms/python/google. Acesso em: 10 jun. 2019.

REYNOLDS, Jason. 8 World-Class Software Companies That Use Python. **Real Python**, [*S. l.*], 2019. Disponível em: https://realpython.com/world-class-companies-using-python/. Acesso em: 10 jun. 2019.

SCHROFF, Florian; KALENICHENKO, Dmitry; PHILBIN, James. FaceNet: A Unified Embedding for Face Recognition and Clustering. **Cornell University**, [*S. l.*], p. 1-10,

12 mar. 2015. Disponível em: https://arxiv.org/pdf/1503.03832.pdf. Acesso em: 17 maio 2019.

SMITH, Brendan. **Setting up Raspberry Pi**. [*S. l.*], 9 set. 2016. Disponível em: https://github.com/bwasmith/Rhew-R-Pi/wiki/Setting-up-Raspberry-Pi. Acesso em: 20 jun. 2019.

VIOLA, Paul; JONES, Michael. Rapid Object Detection using a Boosted Cascade of Simple Features. **Conference on Computer Vision and Pattern Recognition**, [*S. l.*], 2001. Disponível em: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf. Acesso em: 18 jun. 2019.

WILEY ONLINE LIBRARY. THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS. **Wiley Online Library**, [*S. l.*], set. 1936. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x. Acesso em: 10 jun. 2019.

ANEXO A – CÓDIGO DE EXEMPLO DA BIBLIOTECA FACE_RECOGNITION PARA A RASPBERRY PI

```
# This is a demo of running face recognition on a Raspberry Pi.
# This program will print out the names of anyone it recognizes to the console.

# To run this, you need a Raspberry Pi 2 (or greater) with face_recognition and
# the picamera[array] module installed.
# You can follow this installation instructions to get your RPi set up:
# https://gist.github.com/ageitgey/1ac8dbe8572f3f533df6269dab35df65

import face_recognition
import picamera
import numpy as np

# Get a reference to the Raspberry Pi camera.
# If this fails, make sure you have a camera connected to the RPi and that you
# enabled your camera in raspi-config and rebooted first.
camera = picamera.PiCamera()
camera.resolution = (320, 240)
output = np.empty((240, 320, 3), dtype=np.uint8)

# Load a sample picture and learn how to recognize it.
print("Loading known face image(s)")
obama_image = face_recognition.load_image_file("obama_small.jpg")
obama_face_encoding = face_recognition.face_encodings(obama_image)[0]

# Initialize some variables
face_locations = []
face_encodings = []

while True:
    print("Capturing image.")
    # Grab a single frame of video from the RPi camera as a numpy array
```

```
camera.capture(output, format="rgb")

# Find all the faces and face encodings in the current frame of video
face_locations = face_recognition.face_locations(output)
print("Found {} faces in image.".format(len(face_locations)))
face_encodings = face_recognition.face_encodings(output, face_locations)

# Loop over each face found in the frame to see if it's someone we know.
for face_encoding in face_encodings:
    # See if the face is a match for the known face(s)
    match = face_recognition.compare_faces([obama_face_encoding], face_encoding)
    name = "<Unknown Person>"

    if match[0]:
        name = "Barack Obama"

    print("I see someone named {}!".format(name))
```

ANEXO B – CÓDIGO DE EXEMPLO DA BIBLIOTECA FACE_RECOGNITION PARA COMPUTADOR

```
import face_recognition

import cv2

import numpy as np


# This is a demo of running face recognition on live video from your webcam. It's a little more complicated than the

# other example, but it includes some basic performance tweaks to make things run a lot faster:

#   1. Process each video frame at 1/4 resolution (though still display it at full resolution)

#   2. Only detect faces in every other frame of video.


# PLEASE NOTE: This example requires OpenCV (the `cv2` library) to be installed only to read from your webcam.

# OpenCV is *not* required to use the face_recognition library. It's only required if you want to run this

# specific demo. If you have trouble installing it, try any of the other demos that don't require it instead.


# Get a reference to webcam #0 (the default one)

video_capture = cv2.VideoCapture(0)


# Load a sample picture and learn how to recognize it.

obama_image = face_recognition.load_image_file("obama.jpg")

obama_face_encoding = face_recognition.face_encodings(obama_image)[0]


# Load a second sample picture and learn how to recognize it.

biden_image = face_recognition.load_image_file("biden.jpg")

biden_face_encoding = face_recognition.face_encodings(biden_image)[0]


# Create arrays of known face encodings and their names

known_face_encodings = [

    obama_face_encoding,

    biden_face_encoding

]

known_face_names = [

    "Barack Obama",
```

```
    "Joe Biden"
]

# Initialize some variables
face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()

    # Resize frame of video to 1/4 size for faster face recognition processing
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # Convert the image from BGR color (which OpenCV uses) to RGB color (which face_recognition uses)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Only process every other frame of video to save time
    if process_this_frame:
        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            name = "Unknown"

            # # If a match was found in known_face_encodings, just use the first one.
            # if True in matches:
            #     first_match_index = matches.index(True)
```

```
        #     name = known_face_names[first_match_index]

        # Or instead, use the known face with the smallest distance to the new face
        face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = known_face_names[best_match_index]

        face_names.append(name)

process_this_frame = not process_this_frame



# Display the results
for (top, right, bottom, left), name in zip(face_locations, face_names):
    # Scale back up face locations since the frame we detected in was scaled to 1/4 size
    top *= 4
    right *= 4
    bottom *= 4
    left *= 4

    # Draw a box around the face
    cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

    # Draw a label with a name below the face
    cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
    font = cv2.FONT_HERSHEY_DUPLEX
    cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

# Display the resulting image
cv2.imshow('Video', frame)

# Hit 'q' on the keyboard to quit!
```

```
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()
```