

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VICTOR HUGO BRAGUIM CANTO

USO DE REDES NEURAIS CONVOLUCIONAIS PARA
IDENTIFICAÇÃO BIOMÉTRICA DE INDIVÍDUOS EM TEMPO REAL
UTILIZANDO INTERNET DAS COISAS

Bauru/SP

2019

VICTOR HUGO BRAGUIM CANTO

**USO DE REDES NEURAS CONVOLUCIONAIS PARA
IDENTIFICAÇÃO BIOMÉTRICA DE INDIVÍDUOS EM TEMPO REAL
UTILIZANDO INTERNET DAS COISAS**

Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação apresentado ao Departamento de Computação da Faculdade de Ciências da Universidade Estadual Paulista “Júlio de Mesquita Filho” – UNESP, Campus de Bauru.

Orientação: Prof. Dr. Aparecido Nilceu Marana

Bauru/SP

2019

CANTO, Victor Hugo Braguim.

Uso de redes neurais convolucionais para a identificação biométrica de indivíduos em tempo real utilizando Internet das Coisas/ Victor Hugo Braguim Canto, 2019 57 p. : il.

Orientador: Prof. Dr. Aparecido Nilceu Marana
Monografia (Graduação)–Universidade Estadual Paulista. Faculdade de Ciências, Bauru, 2019

1. Redes Neurais Neurais Convolucionais. Multi-task Cascaded Convolutional Networks. VGG-Face. Internet das Coisas. Universidade Estadual Paulista. Faculdade de Ciências. II.
Título.

VICTOR HUGO BRAGUIM CANTO

**USO DE REDES NEURAS CONVOLUCIONAIS PARA
IDENTIFICAÇÃO BIOMÉTRICA DE INDIVÍDUOS EM TEMPO REAL
UTILIZANDO INTERNET DAS COISAS**

BANCA EXAMINADORA

Prof. Dr. Aparecido Nilceu Marana

Orientador

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Departamento de computação
Faculdade de Ciências

Profa. Dra. Simone das Graças Domingues Prado

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Departamento de computação
Faculdade de Ciências

Prof. Dr. Kelton Augusto Pontara da Costa

FATEC – Faculdade de Tecnologia
Bauru

Bauru, 12 de Junho de 2019.

Dedico este trabalho à minha família e, em especial, ao meu avô (*in memoriam*) Paulo Aparecido Braguim

AGRADECIMENTOS

Agradeço primeiramente a Deus, por todo apoio e sustentação para eu chegar até aqui.

Aos meus pais Amarildo Aparecido Canto, Cléia Regina Braguim Canto e a minha irmã Bruna Mayara Braguim Canto, por todo o amor, companheirismo e motivação, pois sem vocês eu não teria chegado até aqui.

À minha namorada, Jaqueline Alves da Silva, por todo amor, companheirismo, apoio, paciência e motivações nas horas mais difíceis.

Ao meu orientador, Prof. Nilceu, por todo o aprendizado, incentivo e ajuda para o desenvolvimento do presente projeto.

Aos meus professores, por todas as lições e ensinamentos para toda a vida.

Aos meus colegas, amigos e familiares não citados diretamente, mas que contribuíram de alguma forma e me ajudaram.

*"Que os vossos esforços desafiem as impossibilidades,
lembrai-vos de que as grandes coisas do homem foram
conquistadas do que parecia impossível."*

Charles Chaplin

RESUMO

A área de Visão Computacional se tornou bastante popular nos últimos anos. Para realizar o processo de visão computacional é crucial o desenvolvimento de técnicas de reconhecimento de padrões, em imagens digitais. Dentre as inúmeras aplicações recentes de Visão Computacional destaca-se a identificação biométrica de pessoas, realizada por meio de suas características físicas ou comportamentais, como a face, a impressão digital, a íris e o modo de caminhar. Tendo isto em vista, o presente trabalho apresenta um sistema implementado utilizando Redes Neurais Convolucionais, mais especificamente as MTCNNs (*Multi-task Cascaded Convolutional Networks*) e a VGG-Face, que tem como objetivo a identificação biométrica de indivíduos por meio do reconhecimento facial no ambiente da Internet das Coisas, especialmente com o uso do *Raspberry Pi* e uma câmera de vídeo acoplada.

Palavras-chave: Redes Neurais Convolucionais. Multi-task Cascaded Convolutional Networks. VGG-Face. Internet das Coisas.

ABSTRACT

The area of Computational Vision has become very popular in recent years. In order to perform the Computational Vision process, it is fundamental to develop patterns recognition techniques in digital images. Among the latest applications of Computational Vision is the biometric identification of people, performed through their physical or behavioral characteristics, such as a face, fingerprint, iris and the walking mode. In view of this, this paper presents a system implemented using Convolutional Neural Networks, in particular the MTCNNs (Multitasking Cascade Networks) and the VGG-Face, which aims biometric identification of people through facial recognition in the environment of the Internet of Things, especially with the use of Raspberry Pi and a coupled camcorder.

Keywords: Convolutional Neural Networks. Multi-task Cascaded Convolutional Networks. VGG-Face. Internet of Things.

LISTA DE FIGURAS

Figura 1 – Perceptron.	19
Figura 2 - Perceptron de Múltiplas Camadas.	20
Figura 3 – Arquitetura de uma rede neural convolucional.	21
Figura 4 – Aplicação de CNNs para reconhecimento de pessoas e animais.	21
Figura 5 - Representação da extração de uma camada convolucional.	22
Figura 6 – Representação da regra.	23
Figura 7 – Exemplo de aplicação.	23
Figura 8 – Camada totalmente conectada.	24
Figura 9 – P-NET.	25
Figura 10 - MTCNN - Primeira Etapa.	25
Figura 11- R- NET.	26
Figura 12 - MTCNN - Segunda Etapa.	26
Figura 13 – O-NET.	26
Figura 14 - MTCNN - Terceira Etapa.	27
Figura 15 - Aplicação da MTCNN na torcida do Corinthians.	27
Figura 16 – Aplicação da MTCNN em amostras da base de faces LFW.	28
Figura 17 – Arquitetura VGG-Face.	28
Figura 18 - Raspberry Pi.	30
Figura 19 - Estrutura do Virtualenv.	32
Figura 20 – Criação de um ambiente virtual com virtualenv.	33
Figura 21 – Ativação do virtualenv.	33
Figura 22 - Exemplo de instalação de pacotes do Python utilizando PIP.	34
Figura 23 - Git clone do projeto da API.	35
Figura 24 - Fluxo da API de reconhecimento facial.	36
Figura 25 - Criação e ativação do virtualenv do projeto.	37
Figura 26 - Instalação dos pacotes base do projeto.	38
Figura 27 - Raspi-config.	38
Figura 28 - Interfacing Options.	39
Figura 29 - Opção SSH <i>do raspi-config</i> .	39
Figura 30 - Ativando a versão do SSH Server no Raspberry Pi.	40

Figura 31 – Conexão SSH com Raspberry Pi.	41
Figura 32 - Conector de Câmera no <i>Raspberry Pi</i> .	42
Figura 33: Câmera conectada no <i>Raspberry Pi</i> .	42
Figura 34: <i>Interfacing Options</i> para câmeras.	43
Figura 35 - Opção “Camera” do <i>Interfacing Options</i> .	43
Figura 36 - Opção “Camera”.	44
Figura 37 - Arquitetura MVC.	44
Figura 38 – Estrutura de Camadas do Flask.	45
Figura 39 - Imagem enviada a API.	46
Figura 40 - JSON de retorno da API - Método de Detecção Facial.	47
Figura 41 - JSON de retorno da API - Método de Reconhecimento Facial.	47
Figura 42 – Plataforma do <i>Google Colab</i> .	48
Figura 43 - Imagens após a execução do algoritmo para a execução da MTCN	49
Figura 44 - Detecção de pontos sem face da MTCNN.	49
Figura 45 - Filtro por <i>accuracy</i> da base LFW.	50
Figura 46 – Algoritmo de separação de bases da LFW.	51
Figura 47 – Comparação de faces com VGG-Face.	51
Figura 48 – Reconhecimento facial no <i>Raspberry Pi</i> em vídeo.	52
Figura 49 – Aplicação da MTCNN em vídeo.	53
Figura 50 – Aplicação da VGG-Face em uma imagem de vídeo.	53

LISTA DE ABREVIATURAS E SIGLAS

CSS	<i>Cascading Style Sheets</i>
CNN	Rede Neural Convolucional (<i>Convolutional Neural Network</i>)
CNTK	<i>Cognitive Toolkit</i>
GPU	<i>Graphics Processing Unit</i>
IA	Inteligência Artificial
IoT	<i>Internet of Things</i>
JS	<i>JavaScript</i>
JSON	<i>JavaScript Object Notation</i>
KNN	<i>K Nearest Neighbor</i>
LFW	<i>Labeled Faces in the Wild</i>
MLP	Perceptron de Múltiplas Camadas
MTCNN	Redes Neurais Convolucionais em Cascata Multitarefas
MVC	<i>Model - View - Controller</i>
ReLU	<i>Rectified Linear Unit</i>
RNA	Redes Neurais Artificiais
SSH	<i>Secure Shell</i>
SVM	<i>Support Vector Machine</i>
VGG	<i>Visual Geometry Group</i>

SUMÁRIO

1. INTRODUÇÃO	15
1.1 Objetivos Gerais	16
1.2 Objetivos Específicos	16
1.3 Organização da Monografia	16
2. FUNDAMENTAÇÃO TEÓRICA	18
2.1 Aprendizagem de Máquina	18
2.1.1 Aprendizagem supervisionada	18
2.1.2 Aprendizagem não supervisionada	18
2.2 Redes Neurais Artificiais	19
2.3 Redes Neurais Convolucionais	20
2.3.1 Camada de convolução	22
2.3.2 Função de ativação ReLu	22
2.3.3 Pooling	23
2.3.4 Classificação	24
2.4 MTCNN – Mult-Task Cascaded Convolutional Neural Network	24
2.4.1 MTCNN - Primeira Etapa	25
2.4.2 MTCNN - Segunda Etapa	25
2.4.3 MTCNN - Terceira Etapa	26
2.5 VGG – Face	28
2.6 Internet das Coisas	29
2.6.1 Raspberry Pi	29
3. MATERIAL UTILIZADO	31
3.1 A Linguagem Python	31
3.1.2 Ambientes Virtuais (Virtualenv)	32
3.1.3 Gerenciador de Pacotes (PIP)	34
3.2 Git e Github	34
4. PROJETO PROPOSTO	36
4.1 Configuração do Ambiente Virtual	37
4.2 Habilitando o protocolo SSH no <i>Raspberry Pi</i>	38
4.3 Habilitando a câmera no <i>Raspberry Pi</i>	41
4.4 Estrutura do Projeto em Flask	44
4.5 Comunicação Cliente-Servidor	45
4.6 Exemplo de retorno da aplicação	45

5. RESULTADOS	48
5.1 MTCNN	48
5.2 VGG-Face.....	50
6.CONCLUSÃO.....	54
REFERÊNCIAS.....	55

1. INTRODUÇÃO

Atualmente, um dos maiores problemas enfrentados pelas instituições, em termos de segurança, é o monitoramento de acesso a lugares restritos. Mesmo com inúmeros funcionários capacitados, uma falha nesse aspecto, poderá afetar o nível de segurança esperado.

Para isto, podem ser utilizadas técnicas de Visão Computacional com ênfase no reconhecimento facial biométrico de indivíduos. Com isso, o reconhecimento facial biométrico facial permite identificar padrões, como por exemplo, de rosto, boca e distância entre olhos (SILVA & CINTRA, 2015).

Além disso, com a evolução de computadores, em especial com o uso de GPU (*Graphics Processing Unit*) e a disponibilização de bibliotecas, como *TensorFlow* e *Keras*, possibilitou-se o desenvolvimento e pesquisas em Redes Neurais Artificiais com a evolução da arquitetura e complexidade dos algoritmos (HAYKIN, 2001).

Ademais, por meio de dispositivos de IoT (Internet das Coisas, do inglês *Internet of Things*), torna-se o processo de implementação de baixo custo, utilizando, em especial, o *Raspberry Pi* com uma câmera de vídeo acoplada.

O objetivo do presente trabalho é estudar e avaliar algoritmos de detecção e reconhecimento facial bem conhecidos e difundidos na literatura, a saber a *Multi-task Cascaded Convolutional Networks* (MTCNN)¹ e a VGG-Face², ambos com o foco na identificação e no reconhecimento facial, e implementar um sistema utilizando tais Redes Neurais Convolucionais, como o objetivo de realizar a identificação biométrica de indivíduos por meio do reconhecimento facial no ambiente da Internet das Coisas, especialmente com o uso do *Raspberry Pi*³ e uma câmera de vídeo acoplada.

Para o treinamento das redes neurais utilizadas neste trabalho e a avaliação do sistema implementado, foram utilizadas imagens de faces do banco de dados LFW (*Labeled Faces in the Wild*)⁴.

¹ Disponível em: < https://kpzhang93.github.io/MTCNN_face_detection_alignment/paper/spl.pdf Acesso em 30 de maio de 2019

² Disponível em: <<https://www.robots.ox.ac.uk/~vgg/>> Acesso em 30 de maio de 2019

³ Disponível em: <<https://www.raspberrypi.org/about/>> Acesso em 30 de maio de 2019

⁴ Disponível em: <http://vis-www.cs.umass.edu/lfw/> Acesso em 02 de abril de 2019

1.1 Objetivo Geral

Desenvolver um sistema utilizando redes neurais convolucionais para identificar indivíduos em tempo real utilizando técnicas e dispositivos de IoT (*Internet of Things*).

1.2 Objetivos Específicos

- A. Estudar e aplicar conceitos básicos de Aprendizado de Máquina Profundo e Redes Neurais Convolucionais;
- B. Estudar e aplicar conceitos de Visão Computacional, Processamento de imagens e Reconhecimento de Padrões;
- C. Desenvolver e implementar o sistema para identificação biométrica de indivíduos em tempo real, com ênfase no reconhecimento facial;
- D. Alocação do sistema para funcionamento em uma arquitetura IoT (*Internet of Things*) utilizando *Raspberry Pi*;
- E. Obter um conjunto de dados adequado para treinamento das redes neurais que comporão o sistema;
- F. Treinar as redes neurais do sistema desenvolvido;
- G. Testar a acurácia do sistema.

1.3 Organização da Monografia

O presente trabalho divide-se em seções, sendo esta a primeira (Introdução). As demais seções estão dispostas na seguinte ordem:

- **Seção 2** - Fundamentação Teórica: apresentação dos conceitos teóricos envolvidos no trabalho;
- **Seção 3** - Metodologia: descrição do método de pesquisa e desenvolvimento, bem como as ferramentas utilizadas;
- **Seção 4** - Arquitetura do Projeto: descrição da arquitetura do projeto e da abordagem utilizada na implementação e testes;
- **Seção 5** - Resultados: apresentação dos resultados obtidos no trabalho;

- **Seção 6** - Conclusão: considerações finais sobre o trabalho e possíveis trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados os principais fundamentos, conceitos e teorias para o presente projeto. Dentre eles, Aprendizagem de Máquina, Redes Neurais Artificiais, Redes Neurais Convolucionais, Classificação, Internet das Coisas, Processamento de Imagens e Visão Computacional.

2.1 Aprendizagem de Máquina

A Aprendizagem de Máquina se diferencia da Estatística tradicional, conhecida como Estatística de Inferência, por ter como foco a predição e a busca por modelos práticos que podem ser utilizados para a tomada de decisões (KLEINBERG et al., 2015).

Além disso, a aprendizagem de máquina é dividida em aprendizagem supervisionada e não supervisionada.

2.1.1 Aprendizagem supervisionada

Na aprendizagem supervisionada os conjuntos de dados já possuem os seus rótulos de saída. Em outras palavras, o seu objetivo é prever uma variável dependente por meio de variáveis independentes. Problemas de regressão e classificação são resolvidos por meio de aprendizagem supervisionada. As definições de classificação e regressão são as seguintes:

- **Classificação:** As variáveis dependentes do problema são discretas. Alguns exemplos são classificar se uma pessoa é propensa ou não a ter diabetes, classificar tweets como negativos ou positivos etc.
- **Regressão:** As variáveis dependentes do problema são contínuas. Alguns exemplos são estimar a média de valores de aluguéis por região, estimar o salário, etc.

2.1.2 Aprendizagem não supervisionada

Na aprendizagem não supervisionada os conjuntos de dados não possuem os seus rótulos de saída. Dessa maneira, os dados são agrupados em *clusters* por meio

de padrões e relações. Alguns exemplos são o agrupamento de mensagens de e-mail com as mesmas características, perfis de clientes em um *e-commerce*, etc.

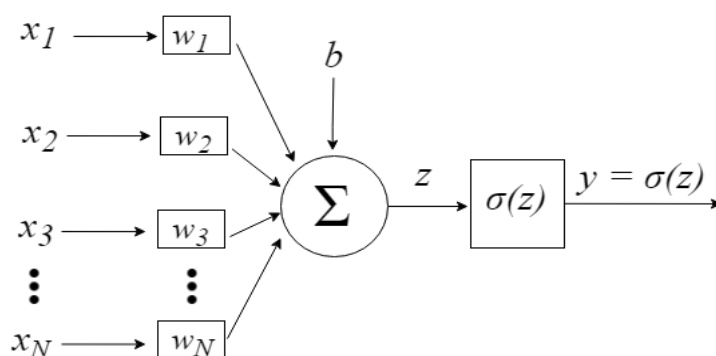
2.2 Redes Neurais Artificiais

No cenário atual, a maioria das organizações teve um aumento significativo no uso e no desenvolvimento de tecnologias que utilizam Inteligência Artificial (IA), Redes Neurais Artificiais e Aprendizado de Máquina. Dentre os seus principais objetivos, está o desenvolvimento de sistemas capazes de trabalhar analogamente a capacidade humana para tomada de decisões.

Outrossim, sistemas que realizam tomadas de decisões, possuem também a propriedade de aprender e, tudo isso é possível, por meio das Redes Neurais Artificiais (RNA). As RNAs, iniciaram por meio da pesquisa realizada por McCulloch e Pitts (1943) e por Rosenblatt (1958), que deu início a arquitetura mais simples de uma rede neural, denominada Perceptron.

O perceptron é a representação de um neurônio biológico em um modelo matemático que permite várias entradas e produz uma saída binária. Na Figura 1, há como entrada x_1, x_2 e x_N , produzindo uma saída binária (DEEP LEARNING BOOK, 2019). Além disso, Rosenblatt também propôs um algoritmo para ajuste dos pesos, permitindo que a sua convergência ocorra em dados linearmente separáveis (SILVA, 1998).

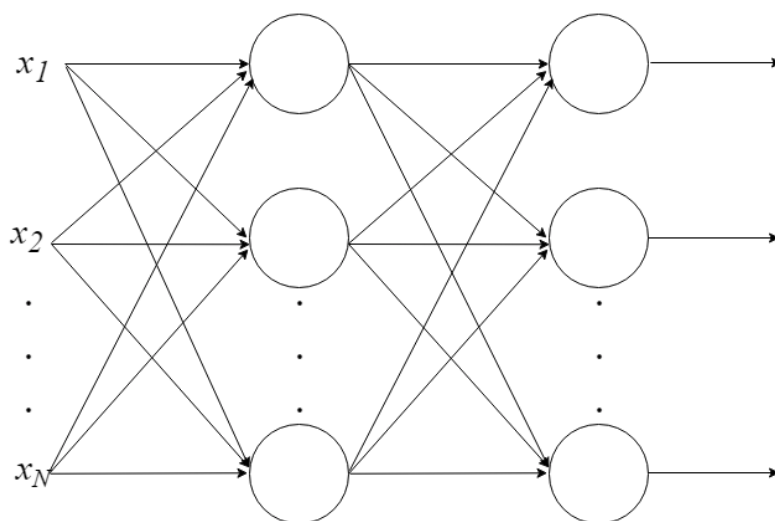
Figura 1 – Perceptron.



Fonte – Ensina (2018).

Além disso, a combinação de diversos perceptrons, alinhando por meio de camadas de entrada, camadas intermediárias (também chamadas de camadas ocultas) e uma camada de saída definem uma arquitetura denotada por Perceptron de Múltiplas Camadas (MLP). Outrossim, o aprendizado da rede utiliza o algoritmo denominado *backpropagation* que realiza a correção dos pesos e do erro durante o processo de treinamento. Na Figura 2, há uma representação de uma MLP.

Figura 2 - Perceptron de Múltiplas Camadas.



Fonte – Ensina (2018).

2.3 Redes Neurais Convolucionais

Uma Rede Neural Convolucional (CNN, do inglês *Convolutional Neural Network*) é uma variação do Perceptron Múltiplas camadas (MLP) (VARGAS, PAES, VASCOSCELOS, s.d). Ademais, as redes neurais convolucionais foram inspiradas no processamento de imagens realizada pelo córtex visual humano para a resolução de problemas de visão computacional (DESHPANDE, 2018).

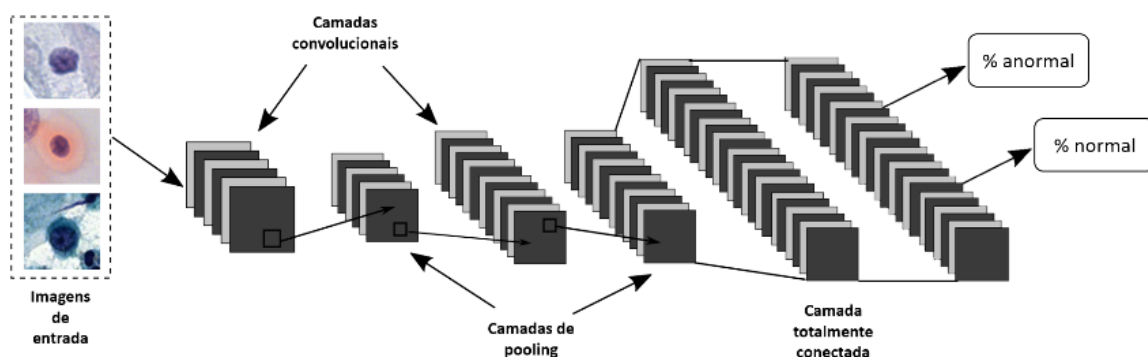
Como consequência da alta disponibilidade de dados, as técnicas de aprendizado de máquina profundo têm revolucionado de maneira significativa diversas áreas e, um exemplo disso na Computação, são as aplicações de Visão Computacional e Reconhecimento de Padrões em imagens (PONTI & COSTA, 2018).

O objetivo da Visão Computacional e do Reconhecimento de Padrões em imagens é simular por meio de softwares e hardware o comportamento da visão humana para identificação e classificação de, por exemplo, pessoas, objetos, etc.

Outrossim, a união de técnicas de Visão Computacional e Aprendizagem de Máquina Profunda, em especial as redes neurais convolucionais, proporcionam maior eficácia em suas tarefas, como em problemas de identificação e autenticação biométrica de indivíduos.

Na Figura 3 há um exemplo de uma rede neural convolucional com arquitetura LeNet, proposta por LeCun (2015) para classificação de imagens de células em normal ou anormal.

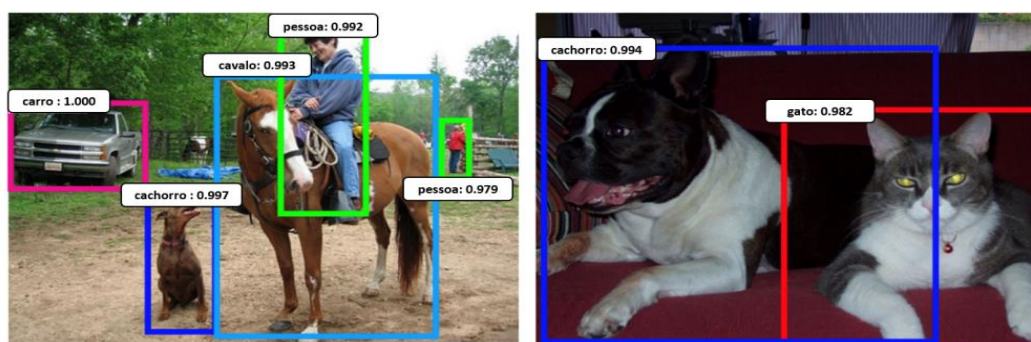
Figura 3 – Arquitetura de uma rede neural convolucional.



Fonte: Araújo, Carneiro e Silva (2017) apud Miyazaki (2017).

Na Figura 4, há exemplos de aplicações que utilizam as CNNs para reconhecimento de objetos, pessoas e animais.

Figura 4 – Aplicação de CNNs para reconhecimento de pessoas e animais.



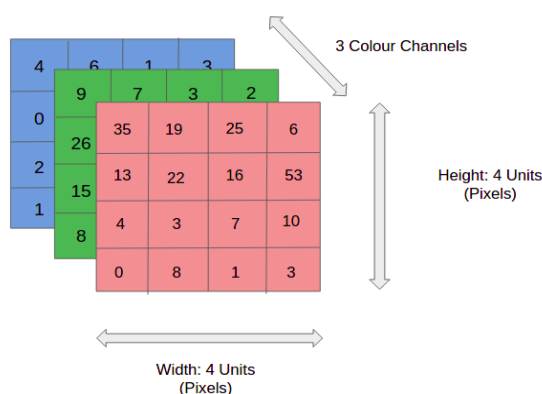
Fonte: Modificado de Ren et al. (2015).

2.3.1 Camada de convolução

A camada convolucional possui como entrada um conjunto de filtros com agrupamentos tridimensionais aplicados às imagens, denominado volume. Em outras palavras, por meio das camadas convolucionais são extraídas todas as características de uma imagem.

Na Figura 5, há uma representação da extração de uma camada convolucional, utilizando matrizes tridimensionais, sendo que, existe a largura, o comprimento e a profundidade.

Figura 5 - Representação da extração de uma camada convolucional.



Fonte: Neuronio.ai (2018).

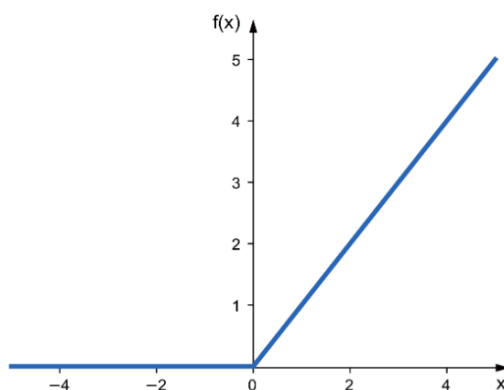
2.3.2 Função de ativação ReLU

A função de ativação ReLU (*Rectified Linear Unit*) possui a função de criar um limite para cada entrada após a camada de convolução. Dessa forma, a função segue a seguinte regra:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Em outras palavras, a função segue a seguinte regra $\max\{0, x\}$ e, graficamente, pode ser representada pela Figura 6.

Figura 6 – Representação da regra.



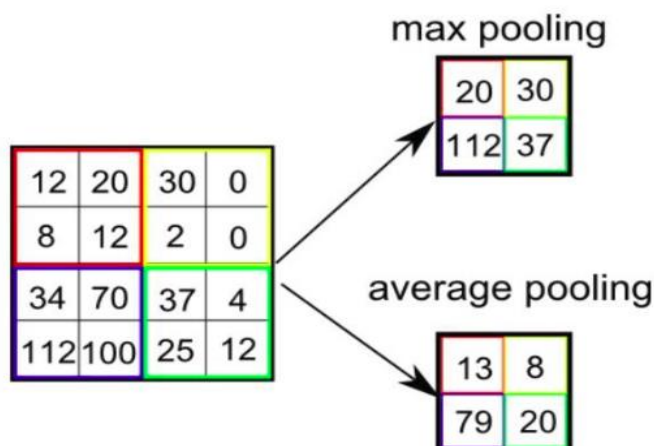
Fonte: Sebastian Raschka (2015).

2.3.3 Pooling

A camada de *pooling* possui o objetivo de reduzir a dimensionalidade da entrada. Como consequência da redução, há uma minimização do tempo de treinamento da rede. Há dois tipos de funções que podem ser realizadas na camada de *pooling*, a função *max pooling* (máximo) e *average pooling*. (média) (NETO et al., 2017).

A função *max pooling* utiliza o valor máximo do kernel de $h \times l$ dimensões e isso proporciona uma minimização dos ruídos contidos na imagem. Dessa forma, o valor máximo torna-se entrada da próxima camada. Já a função *average pooling* utiliza o valor médio para atribuir como entrada à próxima camada. Na Figura 7, há um exemplo da aplicação das duas funções acima.

Figura 7 – Exemplo de aplicação.

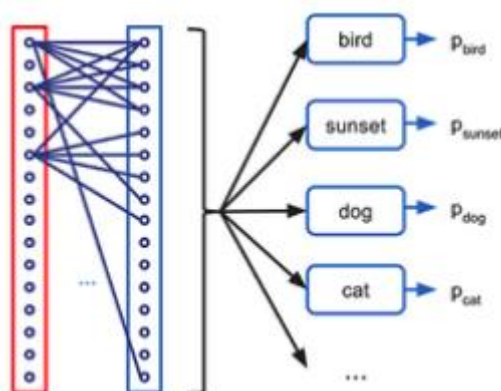


Fonte: Towards Data Science (2018).

2.3.4 Classificação

A camada de classificação ou também chamada de camada totalmente conectada, possui o objetivo de realizar a classificação da imagem com base nos processamentos realizados nas camadas anteriores. Na Figura 8, há um exemplo da camada totalmente conectada com o objetivo de classificar animais.

Figura 8 – Camada totalmente conectada.



Fonte: Eletrical (2018).

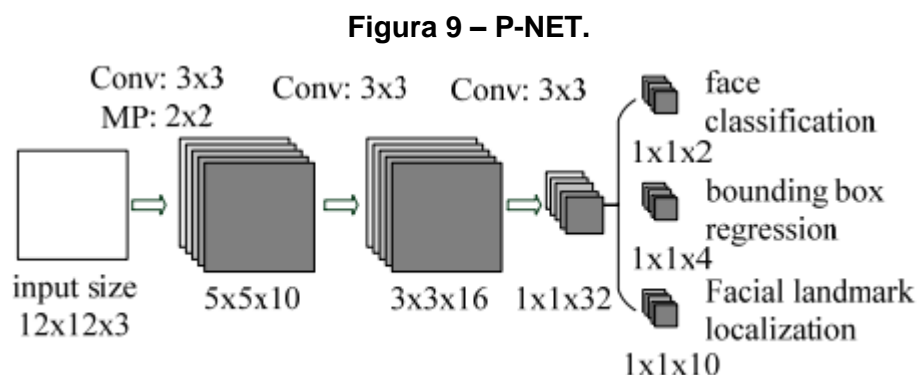
2.4 MTCNN – Multi-Task Cascaded Convolutional Neural Network

Um dos grandes desafios da identificação biométrica de indivíduos com uso de CNNs é a detecção de faces, a extração de características apropriadas e o reconhecimento da face, de forma robusta e invariante às alterações de pontos de vista, iluminação expressões faciais e obstruções (ZHANG et al., 2016).

Contudo, por meio das Redes Neurais Convolucionais em Cascata Multitarefa (MTCNN, do inglês *Multi-task Cascaded Convolutional Neural Network*) é possível encontrar uma solução para a identificação biométrica facial de indivíduos de maneira eficaz. Dessa forma, as MTCNNs são organizadas em três etapas descritas nas próximas três subseções.

2.4.1 MTCNN - Primeira Etapa

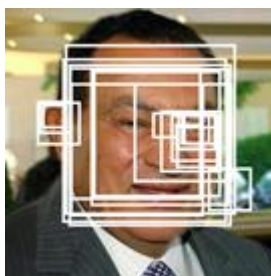
A primeira etapa das MTCNNs utiliza como arquitetura a P-NET (ZHANG et al., 2016) (Figura 9) e possui o objetivo de prever posições da face e suas delimitações.



Fonte: ZHANG et al. (2016).

Um exemplo da P-NET aplicada a uma face, está apresentado na Figura 10.

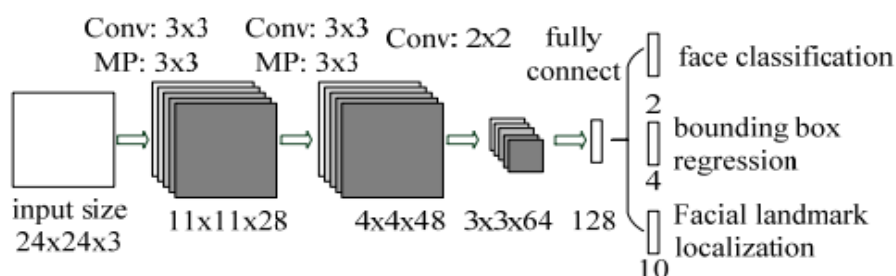
Figura 10 - MTCNN - Primeira Etapa.



Fonte: ZHANG et al. (2016).

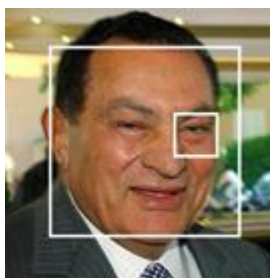
2.4.2 MTCNN - Segunda Etapa

Na segunda etapa da MTCNN, são utilizadas imagens para realizar as primeiras classificações, com isso há um refinamento dos resultados. Dessa forma, a arquitetura utilizada é da R-NET (Figura 11), com objetivo de remover os chamados não candidatos de faces.

Figura 11- R- NET.

Fonte: ZHANG et al. (2016).

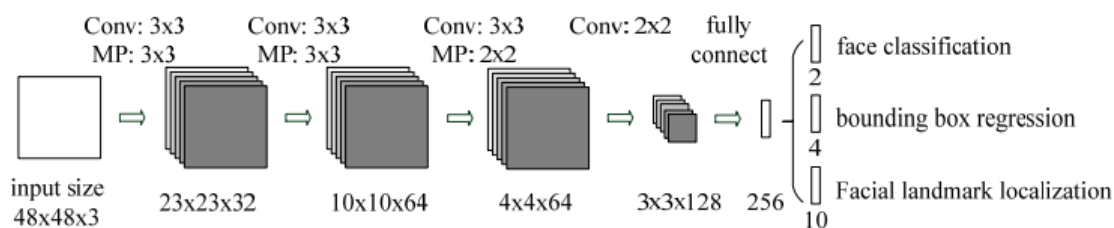
Um exemplo da R-NET aplicada a uma face, está apresentado na Figura 12.

Figura 12 - MTCNN - Segunda Etapa.

Fonte: ZHANG et al. (2016).

2.4.3 MTCNN - Terceira Etapa

Na terceira e última etapa da MTCNN, há um maior refinamento dos resultados e as marcações faciais. Além disso, é utilizada a arquitetura O-NET (Figura 13), com o objetivo de marcar cinco posições faciais: os dois olhos, o nariz e as extremidades esquerda e direita da boca.

Figura 13 – O-NET.

Fonte: ZHANG et al. (2016).

Um exemplo da O-NET aplicada a uma face está apresentado na Figura 14:

Figura 14 - MTCNN - Terceira Etapa.



Fonte: ZHANG et al. (2016).

Outro exemplo do resultado da aplicação das MTCNNs é mostrado na Figura 15.

Figura 15 - Aplicação da MTCNN na torcida do Corinthians.

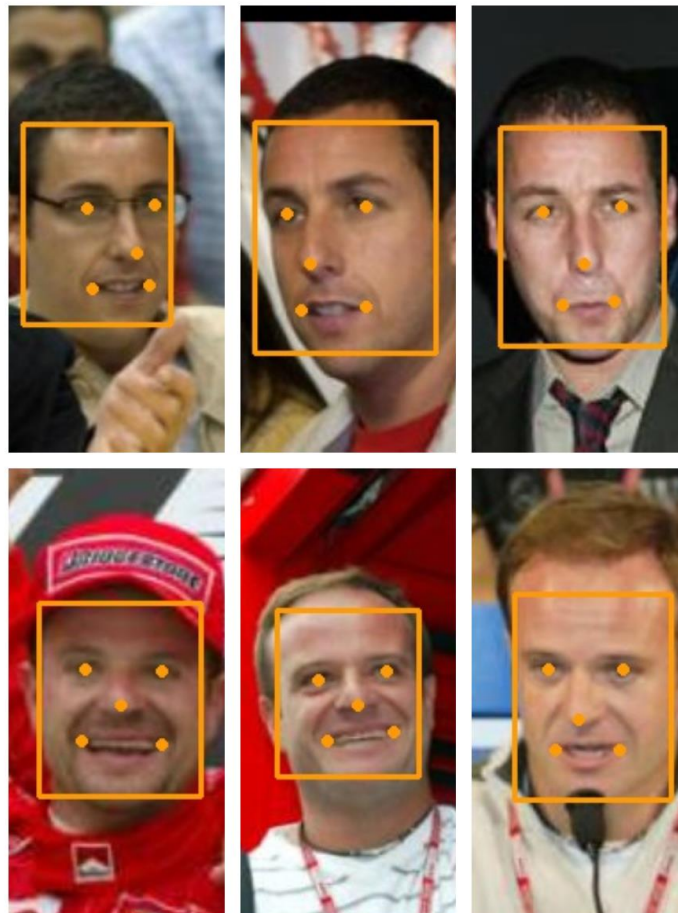


Fonte: Elaborado pelo autor.

A Figura 16 mostra outro exemplo do uso da MTCNN aplicada a uma amostra dos dados da LFW (*Labeled Faces in the Wild*)⁵ base de faces disponibilizada pela Universidade de Massachusetts.

⁵ Disponível em: <http://vis-www.cs.umass.edu/lfw/> Acesso em 02 de abril de 2019

Figura 16 – Aplicação da MTCNN em amostras da base de faces LFW.



Fonte: Elaborado pelo autor.

2.5 VGG – Face

A VGG-Face é uma arquitetura *deep* de uma rede neural convolucional que permite realizar o reconhecimento facial de indivíduos. Na Figura 17, há uma representação das camadas da mesma.

Figura 17 – Arquitetura VGG-Face.

layer type name	0 input	1 conv	2 relu	3 conv	4 relu	5 mpool	6 conv	7 relu	8 conv	9 relu	10 mpool	11 conv	12 relu	13 conv	14 relu	15 conv	16 relu	17 mpool	18 conv
support	–	3	1	3	1	2	3	1	3	1	2	3	1	3	1	3	1	2	3
filt dim	–	3	–	64	–	–	64	–	128	–	–	128	–	256	–	256	–	–	256
num flts	–	64	–	64	–	–	128	–	128	–	–	256	–	256	–	256	–	–	512
stride	–	1	1	1	1	2	1	1	1	1	2	1	1	1	1	1	1	2	1
pad	–	1	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	1
layer type name	19 relu	20 conv	21 relu	22 conv	23 relu	24 mpool	25 conv	26 relu	27 conv	28 relu	29 conv	30 relu	31 mpool	32 conv	33 relu	34 conv	35 relu	36 conv	37 softmax
support	1	3	1	3	1	2	3	1	3	1	3	1	2	7	1	1	1	1	1
filt dim	–	512	–	512	–	–	512	–	512	–	512	–	–	512	–	4096	–	4096	–
num flts	–	512	–	512	–	–	512	–	512	–	512	–	–	4096	–	4096	–	2622	–
stride	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1
pad	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0

Fonte: PARKHI et al. (2015).

Segundo Parkhi et al. (2015), para coletar os dados necessários para treinar a base foram utilizadas as características dos cinco passos a seguir:

- Primeira etapa: Realizar uma busca de nomes de celebridades na internet. Foram coletados 2.622 nomes de celebridades.
- Segunda etapa: Após isso, os nomes das celebridades são pesquisados no Google e na busca de imagens do Bing. Além disso, após realizar as buscas normais, são realizadas as mesmas buscas, mas adicionando a palavra “ator” no início.
- Terceira etapa: Nesta etapa é necessário remover todas as faces encontradas que não fazem parte da pesquisa. Para a classificação, foi utilizado o algoritmo SVM (*Support Vector Machine*) Linear.
- Quarta etapa: Agrupamento das imagens por clusterização e remoção de duplicidade.
- Quinta etapa: Nesta etapa, é realizada a classificação manual das imagens encontradas, refinando assim, os resultados das buscas para a realização do treino da rede.

2.6 Internet das Coisas

Nesta seção, são detalhados o hardware e as tecnologias envolvendo IoT no presente trabalho.

2.6.1 Raspberry Pi

O *Raspberry Pi* foi criado pela *Raspberry Pi Foundation*⁶ no Reino Unido e é um microcomputador do tamanho de um cartão de crédito que possui processador ARM, *slot* para cartões de memória, Ethernet, HDMI e USB, ou seja, possui todo o hardware essencial para um computador em uma única placa.

Além disso, por meio de um cartão de memória, é possível alocar os seguintes sistemas operacionais:

A. **Raspbian**: Distribuição Linux baseada no Debian para arquitetura ARM;

⁶ Disponível em: <<https://www.raspberrypi.org/about/>> Acesso em 30 de maio de 2019

- B. **Ubuntu Mate:** Distribuição Linux baseada no Ubuntu para arquitetura ARM com interface Mate;
- C. **Ubuntu Core:** Distribuição Linux baseada no Ubuntu para arquitetura ARM com foco em Cloud;
- D. **Ubuntu Server:** Distribuição Linux baseada no Ubuntu para arquitetura ARM com foco em servidores;
- E. **Windows IoT Core:** Windows 10 desenvolvido para arquitetura ARM;
- F. **RISC OS:** É um sistema operacional *non-Linux* desenvolvido para arquitetura ARM pela Acorn.

O *Raspberry* possui as seguintes versões: *Raspberry Pi A+*, *Raspberry Pi B*, *Raspberry Pi 2*, *Raspberry Pi Zero*, *Raspberry Pi Zero W*, *Raspberry Pi 3* e *Raspberry Pi 3 B+*.

Na Figura 18, há um exemplo do *Raspberry Pi 3 B+*, o modelo utilizado no presente projeto.

Figura 18 - Raspberry Pi.



Fonte: Elaborado pelo autor.

3. MATERIAL UTILIZADO

Nesta seção, são apresentadas a linguagem de programação e o sistema de controle de versão de projetos utilizados neste trabalho.

3.1 A Linguagem Python

A linguagem de programação Python teve seu início em 1991, sendo criada por Guido van Rossum. É uma linguagem de programação interpretada, orientada a objetos e portátil, ou seja, não é necessário realizar alterações na mudança de plataformas. Os algoritmos desenvolvidos em Python têm sido cada vez mais utilizados para finalidades comerciais e científicas (RASCHKA, 2015).

No presente projeto, são utilizadas as principais bibliotecas da Python, a saber:

- **Pandas:** é uma biblioteca *open source* mantida por uma comunidade ativa de programadores, o que contribui para melhorias contínuas e com alto desempenho, fornecendo ferramentas práticas para estruturas e análises de dados (ANTHONY, 2015). A biblioteca Pandas foi desenvolvida com o objetivo de realizar análises de dados de maneira análoga à linguagem de programação R;
- **Numpy:** é uma biblioteca desenvolvida para a linguagem Python por meio da linguagem C e possui o objetivo de trabalhar com vetores, matrizes e realizar as principais operações utilizadas em Álgebra Linear de maneira simplificada;
- **Scikit-learn:** É uma biblioteca com o objetivo de aplicar algoritmos de *Machine Learning*, como por exemplo, Regressão Linear, *Support Vector Machines*, kNN, Regressão Logística, *Random Forests*, *Naive Bayes* e entre outros;
- **Flask:** Flask é um micro framework desenvolvido em Python, baseado nas bibliotecas Werkzeug e Jinja 2 que possui foco no desenvolvimento de aplicações web;
- **TensorFlow:** É uma biblioteca *open-source* desenvolvida pelo Google que possui o objetivo de realizar a criação e o treinamento de Redes Neurais;
- **Keras:** Keras é um *framework* em Python desenvolvida para ter como opções de backend, o TensorFlow, CNTK ou Theano. Possui também, o objetivo de realizar

o treinamento de Redes Neurais. Entretanto, por abstrair conceitos das bibliotecas de *backend*, torna o desenvolvimento de novos modelos extremamente ágil;

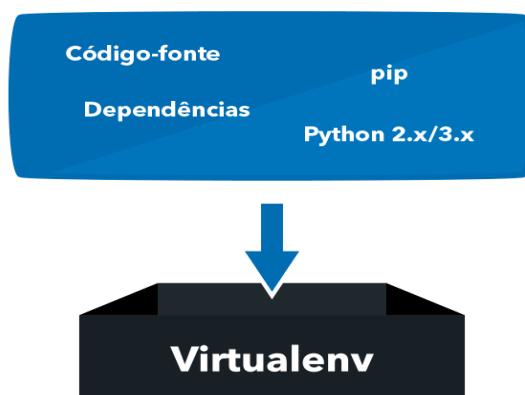
- **OpenCV:** É uma biblioteca *open-source* desenvolvida para realizar o processamento de imagens e aplicar algoritmos de visão computacional.

3.1.2 Ambientes Virtuais (Virtualenv)

Ambientes Virtuais são ferramentas que permitem separar um projeto e suas dependências, como por exemplo, bibliotecas, gerenciadores de pacotes e versões dos interpretadores Python em apenas um único lugar (CODE TUTSPLUS, 2017).

Um ambiente virtual, funciona conforme ilustrado na Figura 19.

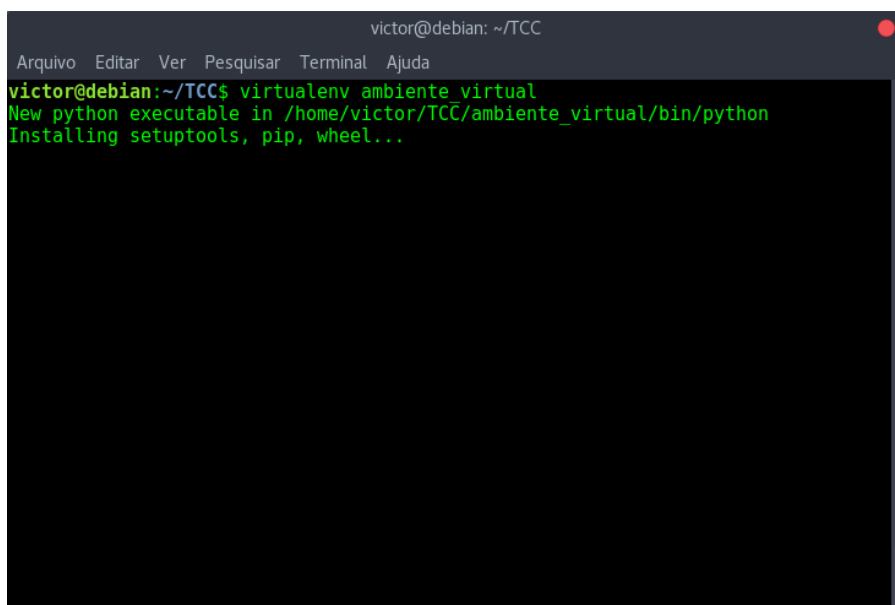
Figura 19 - Estrutura do Virtualenv.



Fonte –Treinaweb (2018).

Para a criação de um ambiente virtual, é necessário seguir os passos da mostrados na Figura 20.

Figura 20 – Criação de um ambiente virtual com virtualenv.

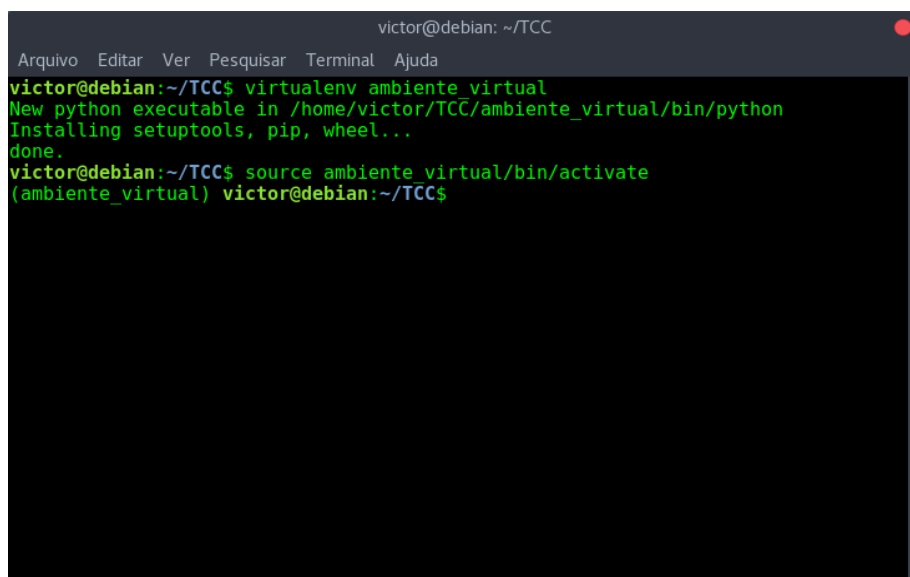
A terminal window titled 'victor@debian: ~/TCC' with a menu bar (Arquivo, Editar, Ver, Pesquisar, Terminal, Ajuda). The command 'virtualenv ambiente_virtual' has been executed, resulting in the output: 'New python executable in /home/victor/TCC/ambiente_virtual/bin/python' and 'Installing setuptools, pip, wheel...'.

```
victor@debian: ~/TCC
Arquivo Editar Ver Pesquisar Terminal Ajuda
victor@debian:~/TCC$ virtualenv ambiente_virtual
New python executable in /home/victor/TCC/ambiente_virtual/bin/python
Installing setuptools, pip, wheel...
```

Fonte: Elaborado pelo autor.

Além disso, para ativar um ambiente virtual em ambiente Linux, é necessário seguir passos mostrados na Figura 21.

Figura 21 – Ativação do virtualenv.

A terminal window titled 'victor@debian: ~/TCC' with a menu bar (Arquivo, Editar, Ver, Pesquisar, Terminal, Ajuda). The command 'source ambiente_virtual/bin/activate' has been executed, resulting in the output: '(ambiente_virtual) victor@debian:~/TCC\$'.

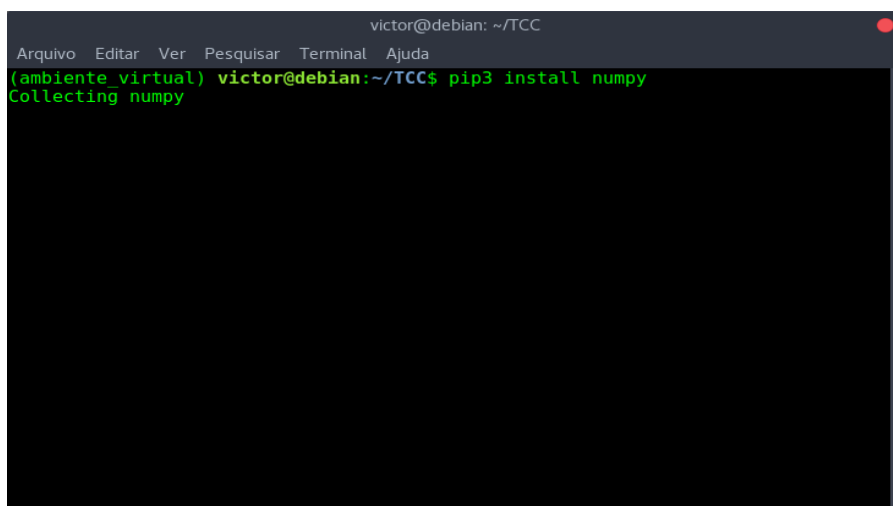
```
victor@debian: ~/TCC
Arquivo Editar Ver Pesquisar Terminal Ajuda
victor@debian:~/TCC$ virtualenv ambiente_virtual
New python executable in /home/victor/TCC/ambiente_virtual/bin/python
Installing setuptools, pip, wheel...
done.
victor@debian:~/TCC$ source ambiente_virtual/bin/activate
(ambiente_virtual) victor@debian:~/TCC$
```

Fonte – Elaborado pelo autor.

3.1.3 Gerenciador de Pacotes (PIP)

O PIP é um gerenciador de pacotes para projetos desenvolvidos em Python, permitindo instalar, remover e atualizar pacotes. Para realizar a instalação de pacotes, como no exemplo para o pacote Numpy, deve-se seguir os passos ilustrados na Figura 22.

Figura 22 - Exemplo de instalação de pacotes do Python utilizando PIP.

A screenshot of a terminal window with a dark background. The title bar at the top reads 'victor@debian: ~/TCC'. Below the title bar is a menu bar with the options 'Arquivo', 'Editar', 'Ver', 'Pesquisar', 'Terminal', and 'Ajuda'. The terminal content shows a green prompt '(ambiente virtual) victor@debian:~/TCC\$' followed by the command 'pip3 install numpy'. The next line shows the output 'Collecting numpy' in green text. The rest of the terminal area is black and empty.

```
victor@debian: ~/TCC
Arquivo  Editar  Ver   Pesquisar  Terminal  Ajuda
(ambiente virtual) victor@debian:~/TCC$ pip3 install numpy
Collecting numpy
```

Fonte – Elaborado pelo autor.

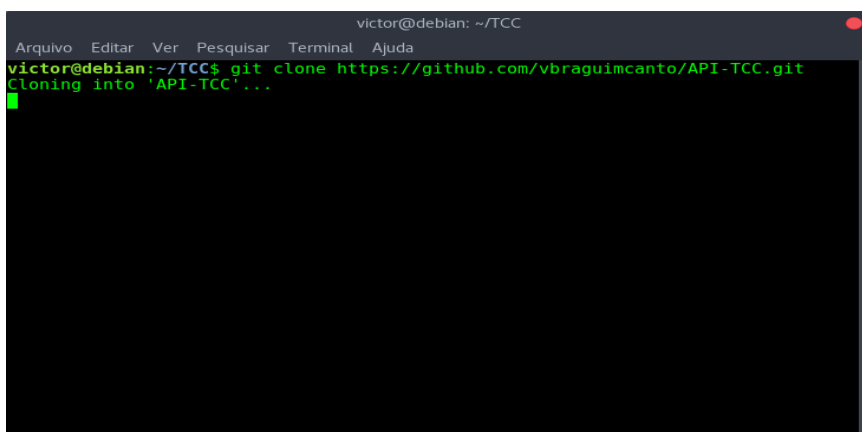
3.2 Git e Github

O Git é um sistema de controle de versão de projetos. Através do Git é possível realizar o versionamento de arquivos no decorrer do projeto, registrando assim o seu período de alteração e, proporcionando também, a recuperação de versões registradas no passado.

Além disso, para o presente projeto, foi utilizado o Github como repositórios dos códigos fonte e, através dele, é possível realizar ações como, por exemplo, *push* (enviar mudanças), *pull* (buscar mudanças) e controles de *branches* (ramificações).

Para realizar o download do repositório do projeto atual disponibilizado no Github, deve-se seguir os passos ilustrados na Figura 23.

Figura 23 - Git clone do projeto da API.

A terminal window with a dark background and light green text. The title bar at the top reads 'victor@debian: ~/TCC'. Below the title bar is a menu bar with the options 'Arquivo', 'Editar', 'Ver', 'Pesquisar', 'Terminal', and 'Ajuda'. The main area of the terminal shows the command 'victor@debian:~/TCC\$ git clone https://github.com/vbraguimcanto/API-TCC.git' and its output 'Cloning into 'API-TCC'...' followed by a green cursor.

```
victor@debian: ~/TCC
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
victor@debian:~/TCC$ git clone https://github.com/vbraguimcanto/API-TCC.git
Cloning into 'API-TCC'...
```

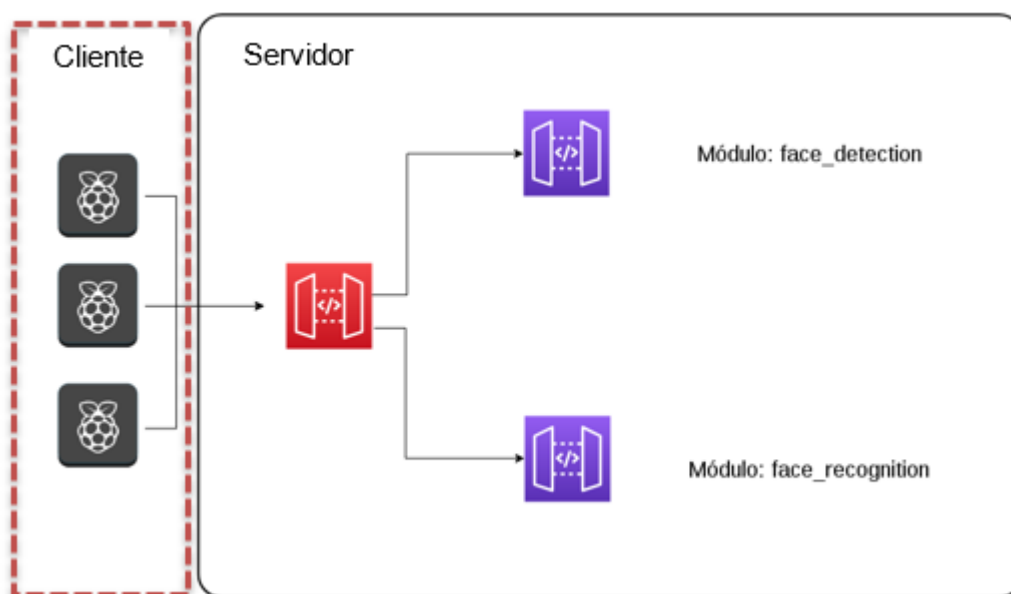
Fonte: Elaborado pelo autor.

4. PROJETO PROPOSTO

Neste trabalho, foi desenvolvido uma API para o reconhecimento facial. A arquitetura utilizada para a API de reconhecimento facial, segue a proposta apresentada na Figura 24, onde há um módulo cliente em funcionamento no *Raspberry Pi* com o objetivo de realizar a captura de uma imagem e o módulo servidor da API alocado em outro computador, devido a limitação no dispositivo *IoT*. O computador, utilizado como servidor, possui o sistema operacional Linux com a distribuição Debian, com 8 GB de memória RAM e processador Core i5 8ª geração e placa de vídeo NVIDIA GeForce MX110. Além disso, o desenvolvimento do projeto foi dividido nas seguintes etapas:

- Configuração de um ambiente virtual utilizando Virtualenv;
- Desenvolvimento da estrutura da API utilizando Python com o Framework Flask;
- Desenvolvimento do módulo de detecção facial utilizando a arquitetura da rede neural MTCNN;
- Desenvolvimento do módulo de reconhecimento facial utilizando VGG-Face, configuração do *Raspbian* no *Raspberry Pi* e alocação do módulo cliente no *Raspbian*.

Figura 24 - Fluxo da API de reconhecimento facial.

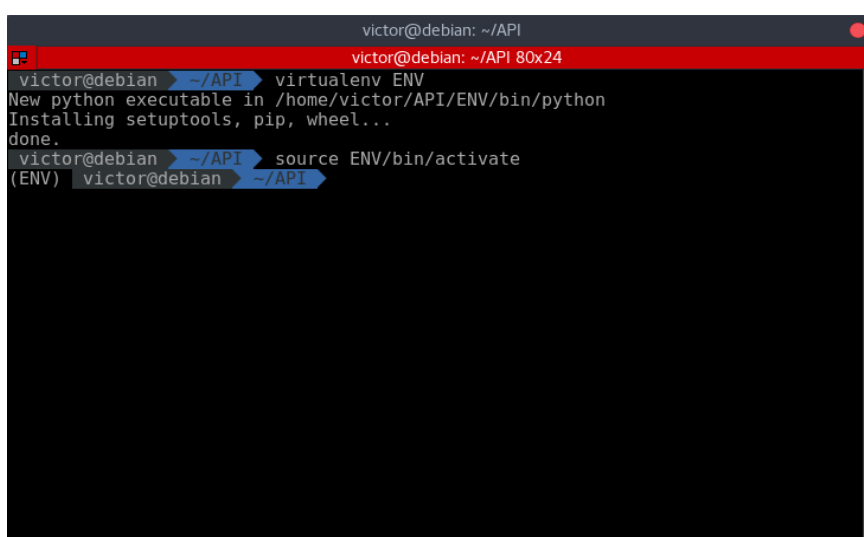


Fonte: Elaborado pelo autor.

4.1 Configuração do Ambiente Virtual

Na seção 3, foi mencionado o conceito e a utilização de ambientes virtuais em projetos. Dessa forma, antes do início do projeto, foi necessário realizar a configuração do ambiente virtual por meio do `virtualenv` e a instalação dos pacotes necessários, conforme mostrado na Figura 25. Inicialmente, foi criado o ambiente virtual, denotado por ENV e, logo após, o ambiente virtual foi ativado por meio do `source`.

Figura 25 - Criação e ativação do virtualenv do projeto.

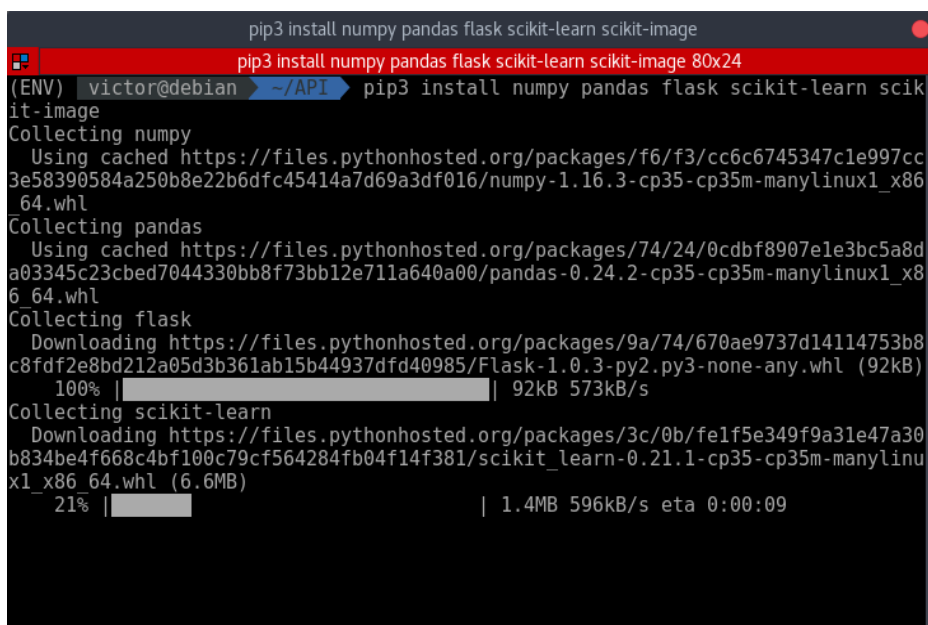


```
victor@debian: ~/API
victor@debian: ~/API 80x24
victor@debian > ~/API > virtualenv ENV
New python executable in /home/victor/API/ENV/bin/python
Installing setuptools, pip, wheel...
done.
victor@debian > ~/API > source ENV/bin/activate
(ENV) victor@debian > ~/API
```

Fonte: Elaborado pelo autor.

Após ativar o ambiente, foi necessário instalar as bibliotecas que farão parte do projeto, conforme a Figura 26 e, com isso, o ambiente virtual ficou preparado para o desenvolvimento.

Figura 26 - Instalação dos pacotes base do projeto.



```

pip3 install numpy pandas flask scikit-learn scikit-image
pip3 install numpy pandas flask scikit-learn scikit-image 80x24
(ENV) victor@debian ~$ pip3 install numpy pandas flask scikit-learn scikit-image
Collecting numpy
  Using cached https://files.pythonhosted.org/packages/f6/f3/cc6c6745347c1e997cc3e58390584a250b8e22b6dfc45414a7d69a3df016/numpy-1.16.3-cp35-cp35m-manylinux1_x86_64.whl
Collecting pandas
  Using cached https://files.pythonhosted.org/packages/74/24/0cdbf8907e1e3bc5a8da03345c23cbcd7044330bb8f73bb12e711a640a00/pandas-0.24.2-cp35-cp35m-manylinux1_x86_64.whl
Collecting flask
  Downloading https://files.pythonhosted.org/packages/9a/74/670ae9737d14114753b8c8fdf2e8bd212a05d3b361ab15b44937dfd40985/Flask-1.0.3-py2.py3-none-any.whl (92kB)
    100% |#####| 92kB 573kB/s
Collecting scikit-learn
  Downloading https://files.pythonhosted.org/packages/3c/0b/felf5e349f9a31e47a30b834be4f668c4bf100c79cf564284fb04f14f381/scikit_learn-0.21.1-cp35-cp35m-manylinux1_x86_64.whl (6.6MB)
    21% |#####| 1.4MB 596kB/s eta 0:00:09
  
```

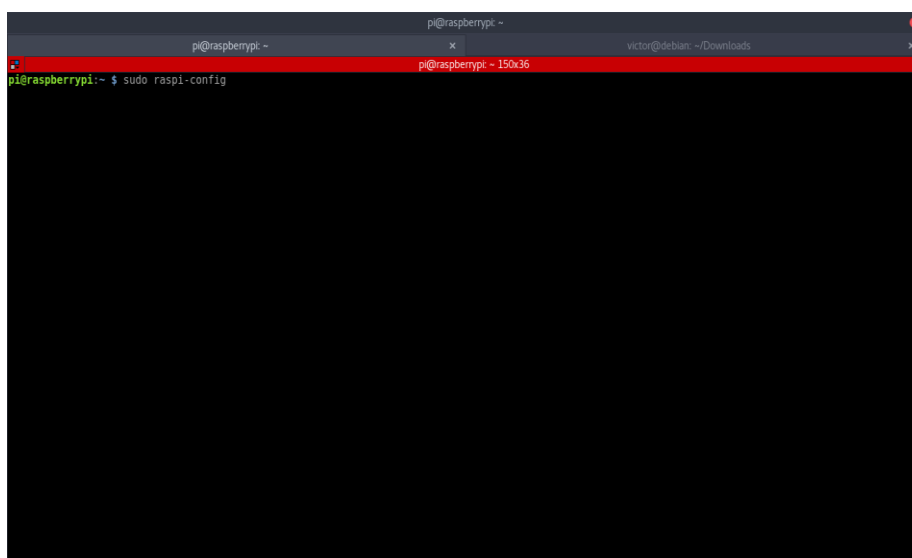
Fonte: Elaborado pelo autor.

4.2 Habilitando o protocolo SSH no *Raspberry Pi*

A conexão com o *Raspberry Pi* foi realizada utilizando o protocolo SSH (*Secure Shell*). O sistema operacional utilizado para acessar o *Raspberry Pi* foi o Debian 9 (versão *Stretch*) na arquitetura x64.

Para habilitar o SSH no *Raspberry Pi* foi necessário utilizar o comando “*sudo raspi-config*”, conforme mostrado na Figura 27.

Figura 27 - Raspi-config.



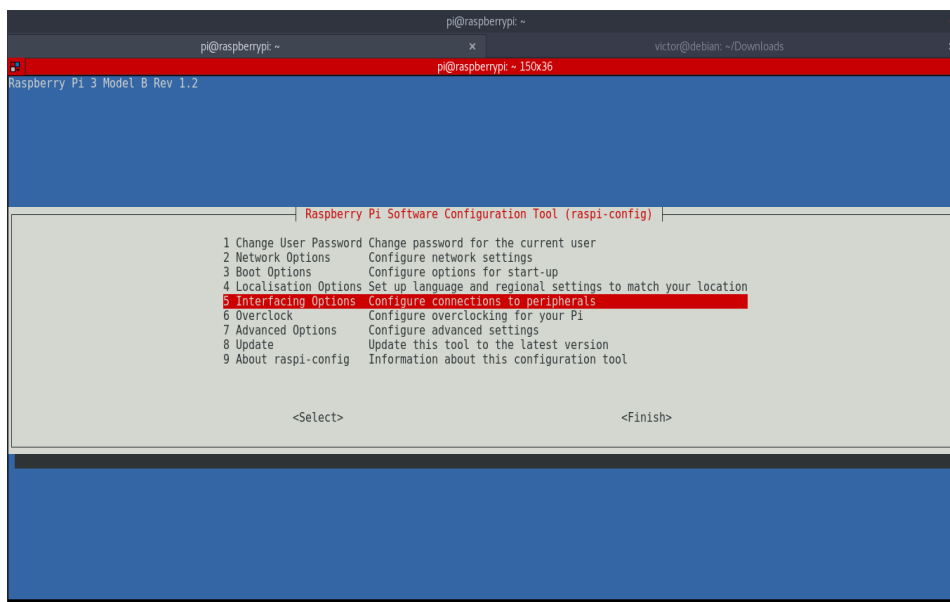
```

pi@raspberrypi ~$ sudo raspi-config
pi@raspberrypi ~$
  
```

Fonte: Elaborado pelo autor.

Após isso, uma configuração é aberta, conforme ilustrado na Figura 28 e, para habilitar o SSH, é necessário selecionar a opção “*Interfacing Options*”.

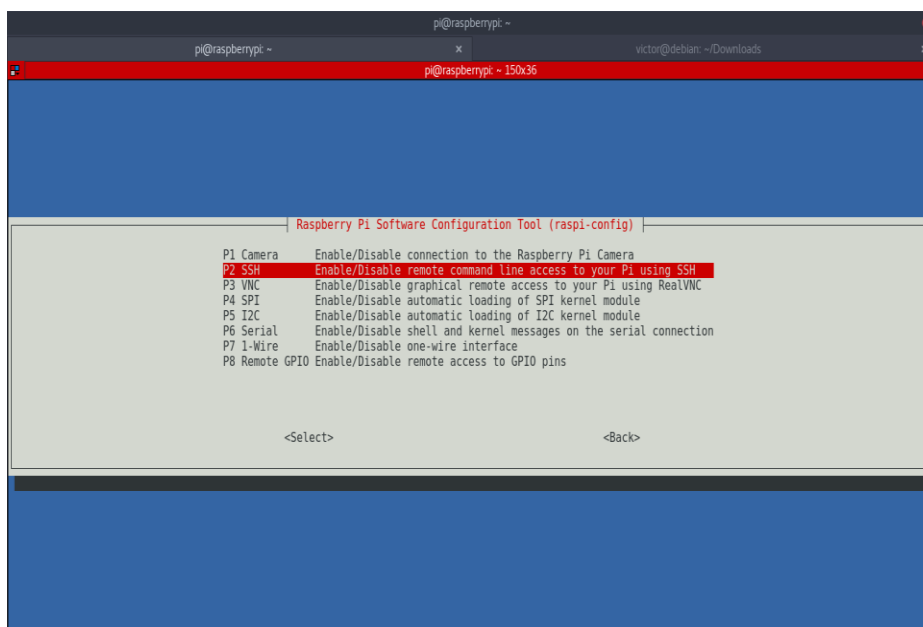
Figura 28 - Interfacing Options.



Fonte: Elaborado pelo autor.

Logo após, aparecerá a opção SSH, conforme ilustrado na Figura 29.

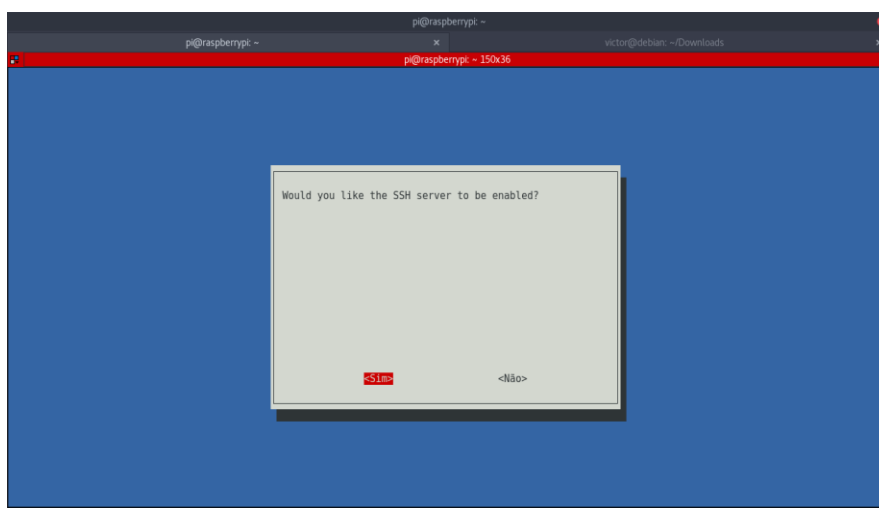
Figura 29 - Opção SSH do raspi-config.



Fonte: Elaborado pelo autor.

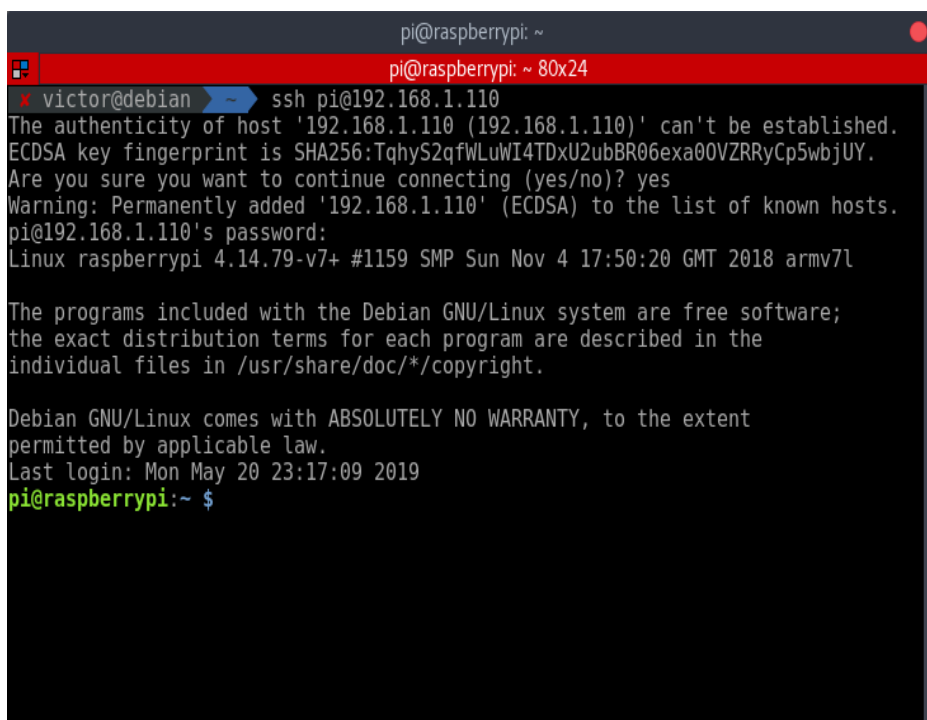
Após selecionar a opção SSH, aparecerá a opção para ativar. Dessa forma, a próxima opção para selecionar deverá ser “Sim”, conforme ilustra a Figura 30.

Figura 30 - Ativando a versão do SSH Server no Raspberry Pi.



Fonte: Elaborado pelo autor.

Após ativado, para acessar, basta utilizar a sintaxe “*ssh nome_usuario@ip_rasbperrypi*”, conforme ilustra a Figura 31.

Figura 31 – Conexão SSH com Raspberry Pi.

```
pi@raspberrypi: ~
victor@debian ~$ ssh pi@192.168.1.110
The authenticity of host '192.168.1.110 (192.168.1.110)' can't be established.
ECDSA key fingerprint is SHA256:TqhyS2qfwLuwI4TDxU2ubBR06exa00VZRRyCp5wbjUY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.110' (ECDSA) to the list of known hosts.
pi@192.168.1.110's password:
Linux raspberrypi 4.14.79-v7+ #1159 SMP Sun Nov 4 17:50:20 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon May 20 23:17:09 2019
pi@raspberrypi:~ $
```

Fonte: Elaborado pelo autor.

4.3 Habilitando a câmera no *Raspberry Pi*

A câmera do *Raspberry Pi* utilizada no projeto é a versão 2, com 8 MP e sensor Sony IMX219. Esta câmera consegue captar vídeos em 1080p30 e 720p60, dentre outras resoluções

Para conectar a câmera no *Raspberry Pi*, há um conector específico, conforme mostra a Figura 32.

Figura 32 - Conector de Câmera no *Raspberry Pi*.



Fonte: Elaborado pelo autor.

É necessário colocar o conector da câmera de tal maneira que o lado azul fique voltado para o lado da conexão com o dispositivo de ethernet, conforme ilustra a Figura 33.

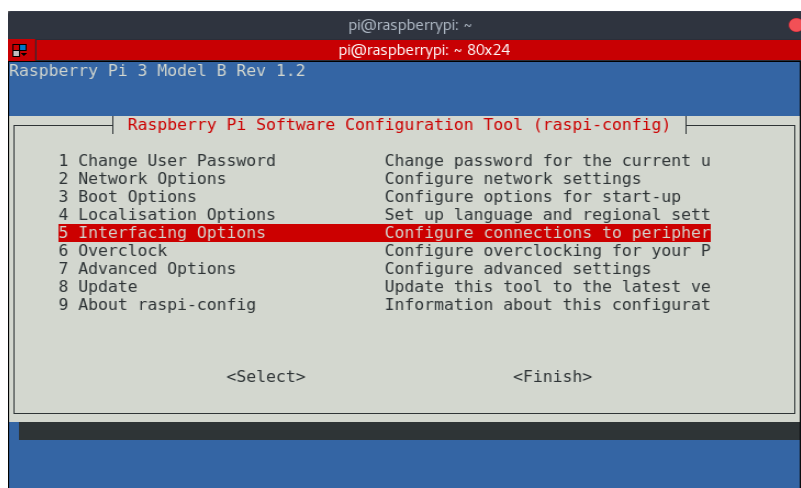
Figura 33: Câmera conectada no *Raspberry Pi*.



Fonte: Elaborado pelo autor.

Após a conexão da câmera, é necessário realizar um processo análogo ao tópico anterior utilizado para habilitar o protocolo SSH. Para isto, é necessário entrar novamente no “*raspi-config*” e ir na opção “*Interfacing Options*”, conforme mostra a Figura 34.

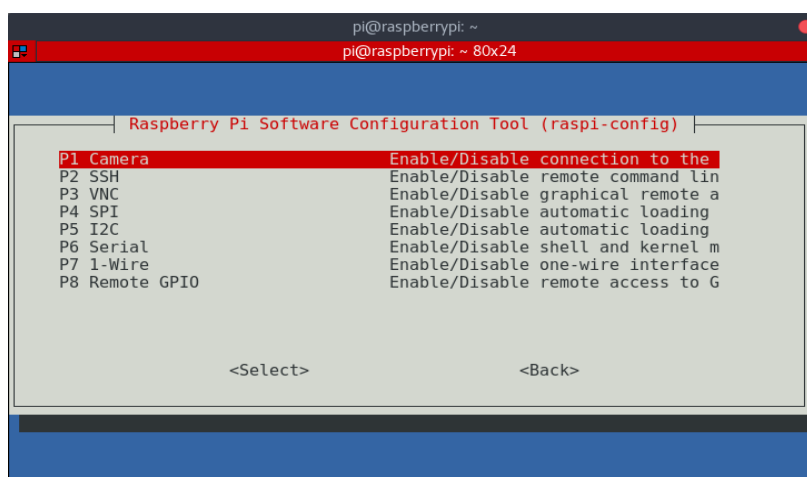
Figura 34: Interfacing Options para câmeras.



Fonte: Elaborado pelo autor.

Logo após, é necessário selecionar a opção “*Camera*”, conforme mostra a Figura 35.

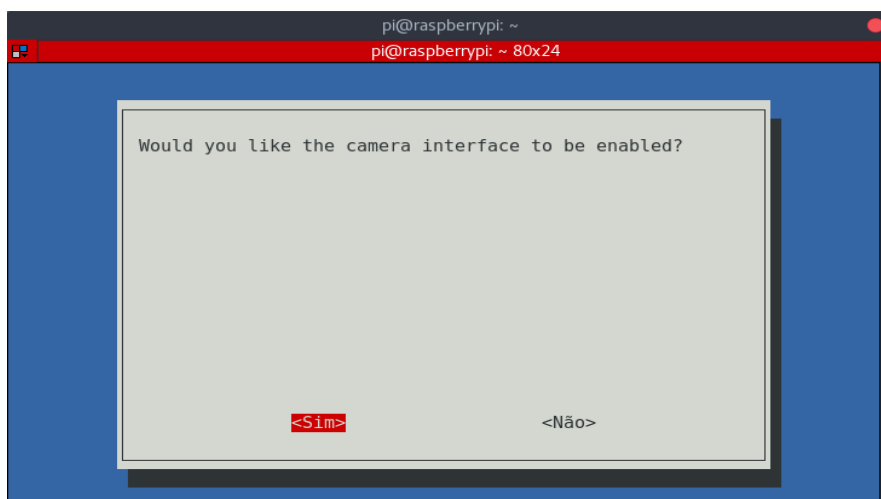
Figura 35 - Opção “Camera” do Interfacing Options.



Fonte: Elaborado pelo autor.

Após selecionar a opção “*Camera*”, aparecerá um menu para confirmação e, a próxima opção deverá ser “*Sim*”, conforme ilustra a Figura 36.

Figura 36 - Opção “Camera”.



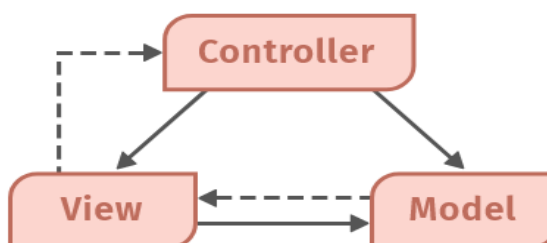
Fonte: Elaborado pelo autor.

Para realizar um teste com a câmera, basta utilizar o comando via terminal “*raspistill -o nome_foto.png*” para fotos e o comando “*raspivid -o nome_video.h264*” para vídeos.

4.4 Estrutura do Projeto em Flask

No presente projeto, desenvolvido em Python, utilizando o microframework Flask, a arquitetura de desenvolvimento do sistema utilizada foi a MVC (*Model - View - Controller*), conforme mostra a Figura 37.

Figura 37 - Arquitetura MVC.



Fonte: (Taller, 2018).

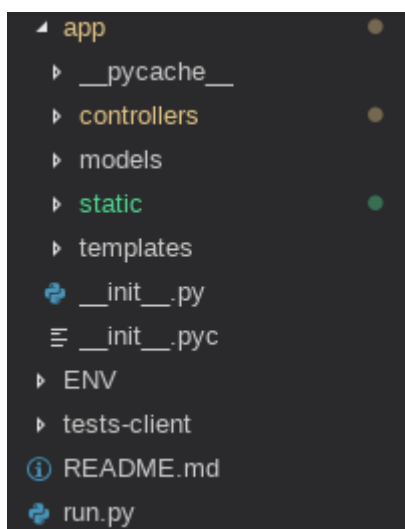
Na camada denominada Model, também chamada de camada de persistência, são alocadas todas as estruturas que realizam qualquer operação com o banco de dados.

Na camada denominada View, são alocadas todas as regras que fazem interações com o usuário de maneira gráfica, como por exemplo, via HTML ou XML.

Na camada denominada Controller, são alocadas todas as regras de negócio para que, ao receber uma requisição de um usuário, a mesma direcione, analogamente a um hub, para qual Model ou View será utilizada. Alguns exemplos de rotas no sistema são: `/api/face_detection` e `/api/face_recognition`.

Além disso, o Flask, por padrão, utiliza os diretórios conforme mostra a Figura 38.

Figura 38 – Estrutura de Camadas do Flask.



Fonte: Elaborado pelo autor.

Nos diretórios *static* e *templates*, ficam alocados arquivos CSS (*Cascading Style Sheets*), HTML e JS (*JavaScript*).

4.5 Comunicação Cliente-Servidor

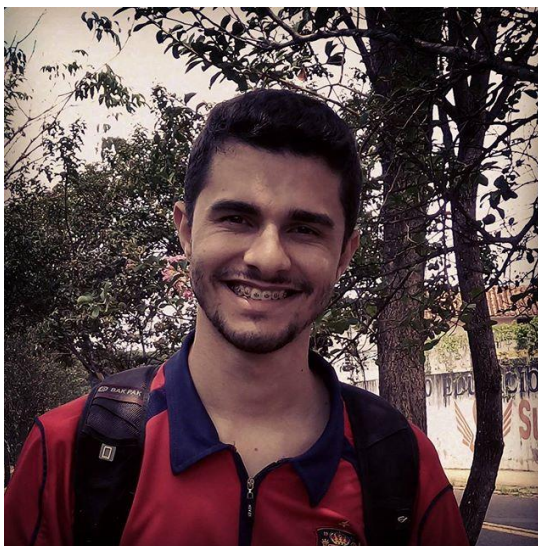
Para a comunicação entre o script local com a API desenvolvida em Flask, utiliza-se por padrão o envio de imagem na forma de *string* utilizando o método “*imencode*” do *OpenCV* e, ao receber no servidor, a imagem é transformada em um *array numpy* via método “*fromstring*” e, logo após, é utilizado o método “*imdecode*”.

4.6 Exemplo de retorno da aplicação

Conforme mencionado na Seção 4.5, a comunicação toda é realizada por meio de *strings*. No servidor, após realizar um *request* na rota “http://localhost:5000/api/face_detection” e a decodificação da imagem ter sido realizada com sucesso, o servidor retornará um arquivo JSON (*JavaScript Object Notation*) contendo informações das posições faciais, como por exemplo, nariz, olhos, boca e uma área retangular (*bounding box*) criada na face.

Por exemplo, para a imagem apresentada na Figura 39, que possui o tamanho original de 640x640 pixels, é gerado o arquivo JSON mostrado na Figura 40.

Figura 39 - Imagem enviada a API.



Fonte: Elaborado pelo autor.

Figura 40 - JSON de retorno da API - Método de Detecção Facial.

```

1 {
2   "face1": [
3     {
4       "left_eye": {
5         "x": 275,
6         "y": 251
7       }
8     },
9     {
10      "right_eye": {
11        "x": 356,
12        "y": 263
13      }
14    },
15    {
16      "nose": {
17        "x": 310,
18        "y": 299
19      }
20    },
21    {
22      "left_mouth": {
23        "x": 262,
24        "y": 325
25      }
26    },
27    {
28      "right_mouth": {
29        "x": 346,
30        "y": 335
31      }
32    },
33    {
34      "bounding_box": {
35        "h": 386,
36        "w": 396,
37        "x": 220,
38        "y": 172
39      }
40    },
41    {
42      "accuracy": {
43        "value": 0.9999593496322632
44      }
45    }
46  ]
47 }

```

Fonte: Elaborado pelo autor.

Para a rota de reconhecimento facial “http://localhost:5000/api/face_recognition”, devolverá um arquivo JSON análogo ao apresentado na Figura 41, com a identificação da pessoa.

Figura 41 - JSON de retorno da API - Método de Reconhecimento Facial.

```

1 {
2   "face": "Victor Hugo Braguim Canto"
3 }

```

Fonte: Elaborado pelo autor.

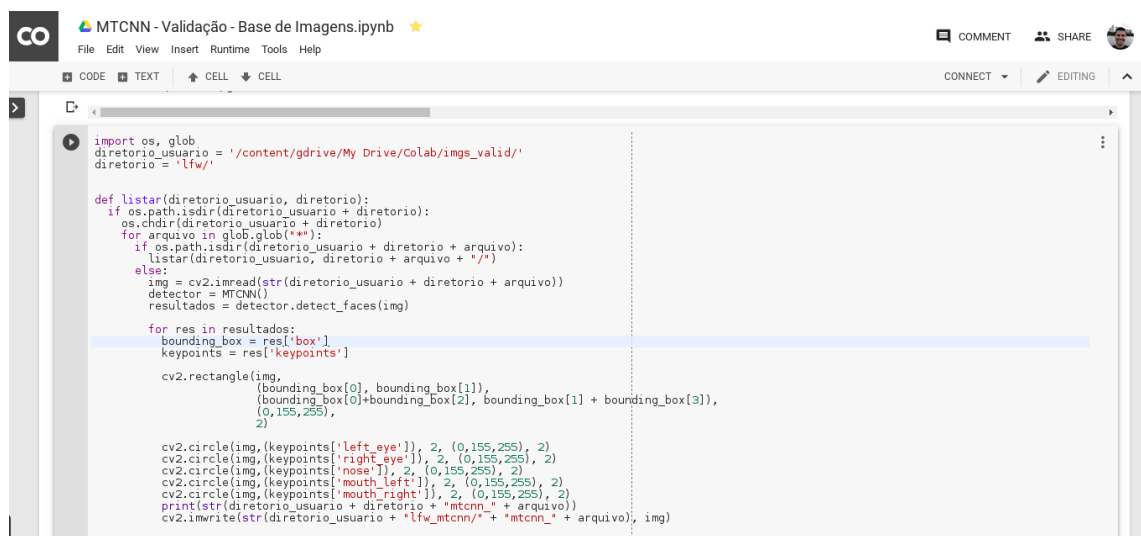
5. RESULTADOS

Nesta seção são detalhados os resultados obtidos por meio do uso de Redes Neurais Convolucionais, mais especificamente MTCNN (ZHANG et al., 2016) para a detecção facial e o uso do VGG-Face (PARKHI et al., 2015) para o reconhecimento facial.

5.1 MTCNN

Para analisar a performance da MTCNN na base da LFW, inicialmente, foi criado um algoritmo em Python para percorrer todos os diretórios e subdiretórios para avaliar as faces nas imagens. Além disso, o algoritmo foi executado utilizando o *Google Colab*⁷, que é uma plataforma *Cloud* gratuita da Google e que permite executar códigos em Python (versões 2.7 e 3.5) no formato do *Jupyter Notebook* e, também, permite o uso de GPU, que no caso da plataforma, é a GPU Tesla K80. Dessa forma, na Figura 42, há um trecho do código no *Colab* para execução.

Figura 42 – Plataforma do Google Colab.



```
import os, glob
diretorio_usuario = '/content/gdrive/My Drive/Colab/imgs_valid/'
diretorio = 'lfw/'

def listar(diretorio_usuario, diretorio):
    if os.path.isdir(diretorio_usuario + diretorio):
        os.chdir(diretorio_usuario + diretorio)
        for arquivo in glob.glob('*'):
            if os.path.isdir(diretorio_usuario + diretorio + arquivo):
                listar(diretorio_usuario, diretorio + arquivo + '/')
            else:
                img = cv2.imread(str(diretorio_usuario + diretorio + arquivo))
                detector = MTCNN()
                resultados = detector.detect_faces(img)

                for res in resultados:
                    bounding_box = res['box']
                    keypoints = res['keypoints']

                    cv2.rectangle(img,
                                (bounding_box[0], bounding_box[1]),
                                (bounding_box[0]+bounding_box[2], bounding_box[1] + bounding_box[3]),
                                (0,155,255),
                                2)

                    cv2.circle(img, (keypoints['left_eye']), 2, (0,155,255), 2)
                    cv2.circle(img, (keypoints['right_eye']), 2, (0,155,255), 2)
                    cv2.circle(img, (keypoints['nose']), 2, (0,155,255), 2)
                    cv2.circle(img, (keypoints['mouth_left']), 2, (0,155,255), 2)
                    cv2.circle(img, (keypoints['mouth_right']), 2, (0,155,255), 2)
                    print(str(diretorio_usuario + diretorio + "mtcnn_" + arquivo))
                    cv2.imwrite(str(diretorio_usuario + "lfw_mtcnn/" + "mtcnn_" + arquivo), img)
```

Fonte: Elaborado pelo autor.

Ao executar o algoritmo, ele aloca todas as imagens em um só diretório, análogo ao apresentado na Figura 43. Seguindo a estrutura de nome do arquivo *mtcnn_nomepessoa_numeroface.jpg*.

⁷ Disponível em: <https://colab.research.google.com/> Acesso em 30 de maio de 2019

Figura 43 - Imagens após a execução do algoritmo para a execução da MTCNN.



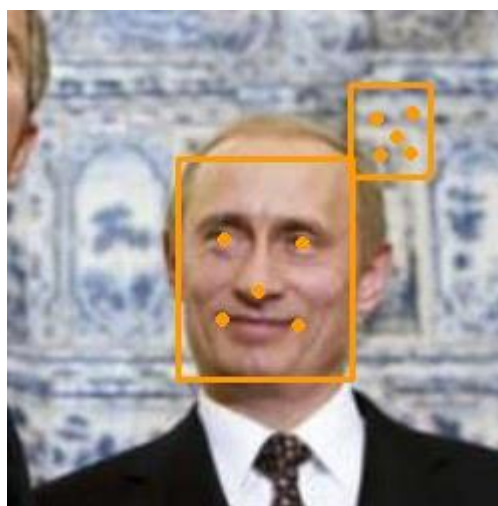
Fonte: Elaborado pelo autor.

A base de imagens LFW possui 13.233 imagens, sendo 5.784 pessoas e 1.680 imagens com mais de uma pessoa. Após aplicar o algoritmo com a MTCNN foi realizada uma validação de forma manual de todas as imagens da base e a acurácia do modelo foi de 98,54%.

Em relação aos 1,46% dos dados com algum tipo de falha, foram constatados alguns detalhes, como por exemplo, identificação de pontos onde não havia faces, faces com inclinações acentuadas sem identificação entre outros.

Na Figura 44, há um exemplo de um caso de identificação de pontos sem face.

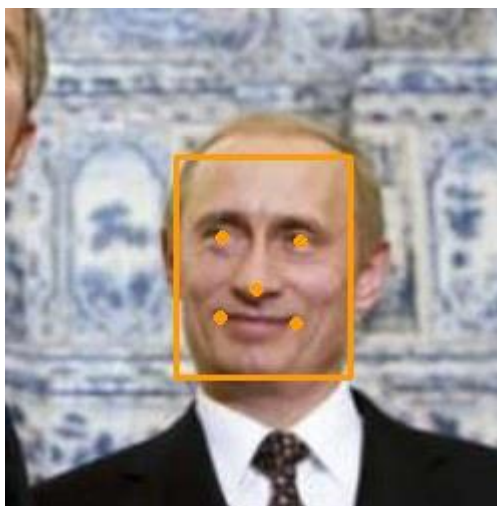
Figura 44 - Detecção de pontos sem face da MTCNN.



Fonte: Elaborado pelo autor.

Entretanto, como mencionado na seção 4.5, através do arquivo de JSON do sistema há um campo chamado *accuracy* e, por meio dele, é possível realizar um filtro do valor base aceitável para a aplicação. Na Figura 45, foi realizado um filtro onde $accuracy > 0.85$ e, com isso, a identificação na imagem foi correta.

Figura 45 - Filtro por *accuracy* da base LFW.



Fonte: Elaborado pelo autor.

Para o sistema atual, por padrão, não são realizados filtros de precisão. Sendo assim, caso for necessário, o filtro pode ser feito pelo arquivo JSON de retorno com os pontos faciais.

5.2 VGG-Face

Para a realização do reconhecimento facial, foi utilizada a rede neural VGG-Face e, para a avaliação da sua performance preditiva, foi utilizada a base LFW. Para isto, inicialmente, foi aplicado a MTCNN para todas as faces agrupando os arquivos com a seguinte nomenclatura:

- Padrão dos nomes dos arquivos: Segue o seguinte padrão, `mtcnn_X_nome_Y`.
- O elemento X representa o número da face encontrada na imagem.
- O elemento Y é um número de quatro dígitos que representa o número da imagem que contém a face da mesma pessoa. Por exemplo: 0015.

Além disso, após aplicar a MTCNN, os arquivos foram divididos em duas bases. Na primeira base, todas as faces cadastradas sem repetição, ou seja, todos os arquivos com o elemento Y igual a 0001. Na segunda base, todas as faces com

elemento Y igual a 0002. No total, temos uma base A com aproximadamente 70% das faces e uma base B de validação com 30% das faces.

Na Figura 46, há o algoritmo em Python para separar as bases:

Figura 46 – Algoritmo de separação de bases da LFW.

```

1 import os.path
2 import shutil
3 import os
4
5 path = '/home/victor/LFW/bases'
6
7 files = []
8 for r, d, f in os.walk(path):
9     for file in f:
10         if '.jpg' in file:
11             strNomeArquivo = str(file)
12             strNomeArquivoQuebrado = strNomeArquivo.split(".")
13             codigo = strNomeArquivoQuebrado[0][-4:]
14
15             if int(codigo) > 1:
16                 shutil.copy2(os.path.join(r, file), '/home/victor/LFW/Test/' + str(file))
17             else:
18                 shutil.copy2(os.path.join(r, file), '/home/victor/LFW/DB/' + str(file))
19 
```

Fonte: Elaborado pelo autor.

Para a verificação da similaridade, foi utilizado o retorno do *predict* do modelo da rede neural e comparado por meio da distância de Cosseno com limitante de 0.4 para considerar como mesma pessoa.

Após aplicar o algoritmo com a MTCNN e a VGG-Face foi realizada uma validação utilizando como base os nomes encontrados nos arquivos e a acurácia do modelo foi de 96,13%.

Na Figura 47, há um exemplo da comparação entre duas imagens, como o valor de retorno da distância de cosseno é de 0.2539, as imagens são consideradas como sendo da mesma pessoa.

Figura 47 – Comparação de faces com VGG-Face.



Fonte: Elaborado pelo autor.

Ademais, um exemplo de teste em vídeo utilizando o sistema no *Raspberry Pi* e na câmera acoplada, está demonstrado na Figura 48, onde há a detecção e o reconhecimento facial e a marcação na imagem via OpenCV.

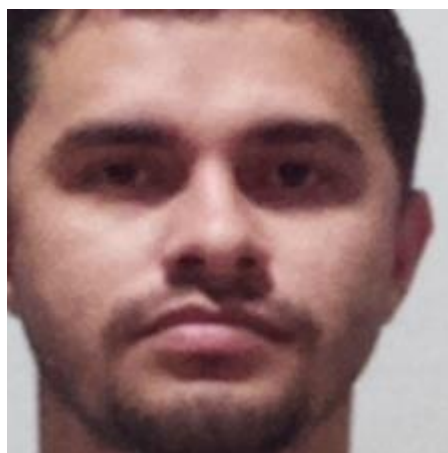
Figura 48 – Reconhecimento facial no *Raspberry Pi* em vídeo.



Fonte: Elaborado pelo autor.

Para obter o resultado acima, inicialmente, foi aplicada a MTCNN. Na Figura 49 há um exemplo da face obtida.

Figura 49 – Aplicação da MTCNN em vídeo.



Fonte: Elaborado pelo autor.

Logo após, as faces, da base e a coletada acima, são comparadas por meio da distância de cosseno, obtendo o valor de 0.2494.

Figura 50 – Aplicação da VGG-Face em uma imagem de vídeo.



Fonte: Elaborado pelo autor.

6.CONCLUSÃO

O presente trabalho apresenta um sistema utilizando Redes Neurais Convolucionais, mais especificamente, MTCNNs e a VGG-Face com o objetivo de realizar a identificação e o reconhecimento facial de indivíduos.

Os testes realizados demonstraram uma boa performance de identificação biométrica combinando o uso de MTCNNs para identificação facial e o uso da VGG-Face para reconhecimento facial, com as acurácias de, respectivamente, 98,54% e 96,13%.

Entretanto, o desenvolvimento do projeto, utilizou a arquitetura cliente-servidor, devido a limitação do hardware presente no *Raspberry Pi*.

Sendo assim, o desenvolvimento em uma arquitetura *IoT* para captura de imagens de vídeo, em especial o *Raspberry Pi* e a câmera acoplada, permite um baixo custo e uma solução eficaz para problemas envolvendo reconhecimento facial.

Dessa forma, esta implementação, considerando aspectos de reconhecimento facial, pode apresentar melhor desempenho em imagens que possuem as faces frontais e sem muita alteração na posição facial.

Como sugestão para trabalhos futuros pode-se considerar a aplicação de mais um módulo para detectar ataques de apresentação ao sensor (também conhecidos como *spoofing*) com o objetivo de realizar a autenticação biométrica facial de forma mais segura.

REFERÊNCIAS

ADIT DESHPANDE. **A Beginner's Guide to Understanding Convolutional Neural Networks**. 2018. Disponível em: <<https://goo.gl/3gswpe>>. (Acesso em: 27 de Agosto de 2018).

AHONEN T, HADID A, PIETIKAINEN M. **Face description with local binary patterns: Application to face recognition**. *IEEE Transactions on Pattern Analysis & Machine Intelligence*. 2006 Dec 1(12):2037-41.)

ANTHONY, Femi. **Mastering pandas**. Packt Publishing Ltd, 2015.

CODE TUTSPLUS. Disponível em< <https://code.tutsplus.com/> Acesso em 30 de maio de 2019

DEEP LEARNING BOOOK. Disponível em: <http://deeplearningbook.com.br/> Acesso em 30 de maio de 2019

ELETRICAL. **O que são redes neurais convolucionais?**. (2018). Disponível em: <http://www.electricalibrary.com/2018/11/20/o-que-sao-redes-neurais-convolucionais/> Acesso em 22 de maio de 2019

ENSINA. LEITE, T, M. **Redes Neurais, Perceptron Multicamadas e o Algoritmo Backpropagation**. 2018. Disponível em <<https://medium.com/ensina-ai/redes-neurais-perceptron-multicamadas-e-o-algoritmo-backpropagation-eaf89778f5b8>> Acesso em 23 de maio de 2019

HAYKIN, SIMON S. **Redes Neurais** - 2ed. Bookman, 2001.

IEEEEXPLORE. **A real-time face recognition system based on the improved LBPH algorithm**. (2017). Disponível em:< <https://ieeexplore.ieee.org/document/8124508> Acesso em 26 de maio de 2019

KLEINBERG J, LUDWING J, MULLAINATHAN S, OBERMEYER Z. **Prediction policy problems**. *American Economic Review*. 2015;105(5):491-95. Disponível em <<https://www.cs.cornell.edu/home/kleinber/aer15-prediction.pdf>> Acesso em 30 de maio de 2019

LABELLED Faces in the Wild. Labeled Faces in the Wild Home. Disponível em: <http://vis-www.cs.umass.edu/lfw/> Acesso em 28 de maio de 2019

LECUN, Lenet. **LeNet-5, convolutional neuralnetworks**. (2015) Disponível em: <http://yann.lecun.com/exdb/lenet> Acesso em 30 de maio de 2019

NEURONIO.AI. **Entendendo Redes Convolucionais (CNNs)**. 2018. Disponível em: <https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184> Acesso em 23 de maio de 2019

NETO, F. G. M.; JÚNIOR, R. F. P.; ROCHA, M. G. O.; JUNIOR, J. J. de M. S.; JÚNIOR, I. C. de P. **Aprendizado profundo: conceitos, técnicas e estudo de caso de**

análise de imagens com java. III Escola Regional de Informática do Piauí. Livro Anais - Artigos e Minicursos 2017)

MCCULLOCH, Warren. PITTS, Walter H. **A logical calculus of the ideas immanent in nervous activity.** In **Bulletin of mathematical biophysics**, Vol 5, 1943, p. 115-133. Disponível em: < <http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf>> Acesso em 30 de maio de 2019

MIYAZAKI, Caio Kioshi. **Redes neurais convolucionais para aprendizagem e reconhecimento de objetos 3D.** 2017.

PARKHI, Omkar M; VEVALDI, Andrea; ZISSERMAN, Andrew. **Deep Face Recognition.** 2015. Disponível em: < <http://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/parkhi15.pdf> Acesso em 02 de junho de 2019

PRADO, Kevin Salton do. **Face Recognition: Understanding LBPH Algorithm.** (2017). Disponível em: < <https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b> Acesso em 27 de maio de 2019

SHAOGING Ren, KAIMINHG He, ROSS Girshick, and JIAN Sun. **Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.** 2016. Disponível em <https://arxiv.org/pdf/1506.01497.pdf> Acesso em 30 de maio de 2019

SILVA, A. L.; CINTRA, M. E. **Reconhecimento de padrões faciais: Um estudo.** In: **Encontro Nacional de Inteligência Artificial e Computacional.** 2015. Proceedings ENIAC.[S.l.: s.n.], 2015. p. 224-231

SILVA, L.N.C. **Análise e síntese de estratégias de aprendizado par redes neurais artificiais.** Capítulo 2. 1998. Disponível em: ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/theses/lnunes_mest/cap2.pdf Acesso em 20 de maio de 2019

RASCHKA, Sebastian. **Machine Learning FAQ Why is the ReLU function not differentiable at $x=0$?** Disponível em < <https://sebastianraschka.com/faq/docs/relu-derivative.html> Acesso em 23 de maio de 2019

RASCHKA, Sebastian. **Python machine learning.** Birmingham; Packt Publishing, 2015.

RASPBERRY PI .**Raspberry Pi Blog.** Disponível em:< <https://www.raspberrypi.org/about/>> Acesso em 30 de maio de 2019

ROSENBLATT, F. **THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN.** 1958. Disponível em< <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf>> Acesso em 30 de maio de 2019

TALLER. **MVC vs. PAC: qual a diferença entre as arquiteturas?**. (2018). Disponível em: <https://blog.taller.net.br/mvc-vs-pac-diferenca-entre-arquiteturas/> Acesso em 27 de maio de 2019

TREINAWEB. PINHEIRO, F. **Criando ambientes virtuais para projetos Python com o Virtualenv**. 2018. Disponível em: <<https://www.treinaweb.com.br/blog/criando-ambientes-virtuais-para-projetos-Python-com-o-virtualenv/>> Acesso em 20 de maio de 2019

TOWARDS, Data Science. **A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way**. (2018). Disponível em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> Acesso em 22 de maio de 2019

TOWARDS, Data Science. Disponível em< <https://towardsdatascience.com/>> Acesso em 30 de maio de 2019

VARGAS, Ana Caroline Gome; PAES Aline; VASCONCESLOS Cristina Nader. **Um Estudo sobre Redes Neurais Convolucionais e sua Aplicação em Detecção de Pedestres**. s.d.

VISUAL GEOMETRY GROUP. Disponível em:< <https://www.robots.ox.ac.uk/~vgg/>> Acesso em 30 de maio de 2019

VIOLA, Paul, and M. Jones. **Robust object detection using a boosted cascade of simple features**. Proc Cvpr 1(2001):l-511-l-518 vol.1.

ZHANG, K.; ZHANG, Z.; ZHIFENG, L.; QIAO, Y. **Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks**. 2016. Disponível em <<https://arxiv.org/abs/1604.02878>> (Acesso em 27 de agosto de 2018).