UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

CAIO HENRIQUE FREITAS

Detecção e determinação da porcentagem de pessoas usando máscaras faciais em imagens capturadas por câmeras fixas ou drones para auxílio na prevenção do COVID19

Guaratinguetá
2021

Caio Henrique Freitas

Detecção e determinação da porcentagem de pessoas usando máscaras faciais em imagens capturadas por câmeras fixas ou drones para auxílio na prevenção do COVID19

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estatual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica .

Orientador: Profº Dr. Daniel Julien Barros da Silva Sampaio

Guaratinguetá
2021

Freitas, Caio Henrique
F866d Detecção e determinação da porcentagem de pessoas usando máscaras faciais em imagens capturadas por câmaras fixas ou drones para auxílio na prevenção do COVID19 / Caio Henrique Freitas – Guaratinguetá, 2021.
50 f : il.
Bibliografia: f. 38-40

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2021.
Orientador: Prof. Dr. Daniel Julien Barros da Silva Sampaio

1. Redes neurais (Computação). 2. Processamento de imagem assistida por computador. 3. Transmissão de imagem.
I. Título.

CDU 681.3

Luciana Máximo
Bibliotecária CRB-8-3595

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

CAIO HENRIQUE FREITAS

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUANDO EM ENGENHARIA ELÉTRICA "**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Profº Dr. DANIEL JULIEN BARROS DA SILVA SAMPAIO
Coordenador

BANCA EXAMINADORA:

Profº Dr. Daniel Julien Barros da Silva Sampaio
Orientador/UNESP-FEG

Profº MSc. Thiago José Michelin
UNESP-FEG

MSc. Luís Fernando de Souza Cardoso
Membro Externo

Abril , 2021

Dedico este trabalho de graduação, em especial, a minha família.

AGRADECIMENTOS

Agradeço a Deus, pelo grande privilégio de cursar esta universidade e realizar este trabalho de graduação.

Agradeço a minha família, que teve papel fundamental de apoio para realização deste trabalho, bem como para conclusão do meu curso.

Agradeço aos meus amigos, que estiveram ao meu lado nos bons e difíceis momentos ao longo da graduação.

Agradeço a minha namorada e companheira Ana Laura, por ter me apoiado e me incentivado no desenvolvimento deste trabalho.

Agradeço imensamente a universidade e ao corpo docente, por todo conhecimento adquirido ao longo deste curso.

RESUMO

A pandemia do Corona vírus foi responsável pela perda de milhares de vidas de brasileiros e de milhões de pessoas ao redor do mundo. Uma das formas mais eficazes de prevenção e proteção é a utilização de máscaras e distanciamento social, como afirma a Organização Mundial de Saúde. De forma a auxiliar na detecção do uso de máscaras pelos indivíduos em um determinado local, o presente trabalho visa o estudo das técnicas presentes hoje na literatura para detecção de imagens. Fazendo uso de rede convolucional VGG-16, em conjunto com a rede YOLO e ISR de super resolução, foi desenvolvido um sistema capaz de calcular a porcentagem de pessoas utilizando máscaras através de imagens capturadas por câmeras fixas e Drones. A ferramenta desenvolvida foi capaz de alcançar uma precisão de aproximadamente 87% para o conjunto de imagens utilizadas em sua validação.

PALAVRAS-CHAVE: Aprendizagem de máquina. Aprendizagem profunda de máquina. Redes Neurais Convolucionais. Reconhecimento de imagem.

ABSTRACT

The spread of Corona Virus around the world, took away millions of lives. The use of facial mask is one of the best ways to stop the spreading process of the virus, as long as social distancing, and they are widely disclosed by World Health Organization. This research work has the objective of developing a software capable of detecting the use of facial masks, by analysing images from surveillance cameras and drones. It uses VGG-16, YOLO and ISR algorithms, and it is capable of defining a percentage of people using mask. The architecture used has reached about 87% accuracy for the used validation dataset.

KEYWORDS: Machine learning. Deep learning. Convolutional Neural Networks. Image recognition.

LISTA DE ILUSTRAÇÕES

Figura 1 Exemplo de aplicação de visão computacional e aprendizado de máquina em veículos autônomos. 14
Figura 2 Unidade Linear com *Threshold*. 16
Figura 3 Multicamada *Perceptron*. 17
Figura 4 Exemplo de rede neural convolucional completa para reconhecimento de algarismos de zero a nove. 18
Figura 5 Função de ativação ReLU. 19
Figura 6 Função de agrupamento máximo aplicado a uma camada 4x4. 19
Figura 7 Esquemático da SRCNet proposta por QIN at all 20
Figura 8 Esquemático da metodologia utilizada 21
Figura 9 Drone utilizado na pesquisa 22
Figura 10 Website City Cameras do governo do estado de SP 22
Figura 11 Sistema de detecção do Yolo. O sistema (1) redimensiona a imagem para a resolução 448x448, (2) roda apenas uma rede convolucional na imagem, e (3) mostra os resultados obtidos através do modelo, após utilizar um sistema de supressão das caixas de detecção sobrepostas. 23
Figura 12 Versão simplificada do módulo *Inception* 24
Figura 13 Exemplo de gradiente descendente para encontro de ponto mínimo da função. . 24
Figura 14 Arquitetura *Darknet-19* com 19 camadas de convolução e 5 camadas de *maxpooling* 25
Figura 15 Arquitetura *Residual Dense Network* 26
Figura 16 IBM Watson, ferramenta de inteligência artificial da IBM 27
Figura 17 *Teachable Machine*, ferramenta de aprendizado de máquina da Google disponível no navegador. 28
Figura 18 Arquitetura do classificador proposto 29
Figura 19 Exemplo de imagens contidas no conjunto de dados utilizado 31
Figura 20 Resultados do treinamento, na esquerda a curva de precisão por época e na direita a curva de perda por época. 32
Figura 21 Interface de usuário do *software*. 32
Figura 22 Exemplo de recorte feito pelo software em imagem utilizada. 33
Figura 23 Exemplo de processamento do módulo de super resolução. 34
Figura 24 Exemplos de imagens processadas pelo *software* 35
Figura 25 Problemas com pessoas de costas nas imagens. 36

LISTA DE TABELAS

Tabela 1 – Matriz de Confusão do sistema completo 35

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CNN	*Convolutional Neural Network*
COCO	*Common Object in Context*
COVID	*Corona Virus Disease*
ISR	*Image Super Resolution*
LTU	*Linear Threshold Unit*
OMS	Organização Mundial da Saúde
RAM	*Random Access Memory*
RDN	*Residual Dense Network*
RNA	Rede Neural Artificial
TM	*Teachable Machine*
UNESP	Universidade Estadual Paulista
VGG	*Visual Geometry Group*
YOLO	*You Only Look Once*

SUMÁRIO

1 INTRODUÇÃO 12
1.1 MOTIVAÇÃO 12
1.2 OBJETIVOS 13
1.3 ESTRUTURA DO TRABALHO 13

2 ASPECTOS TEÓRICOS 14
2.1 VISÃO COMPUTACIONAL E APRENDIZADO DE MÁQUINA 14
2.2 REDES NEURAIS ARTIFICIAIS (RNA) 15
2.3 REDES NEURAIS CONVOLUCIONAIS (CNN) 17
2.4 ESTADO DA ARTE 20

3 MATERIAIS E MÉTODOS 21
3.1 IMAGEM DE DRONES E CÂMERAS DE SEGURANÇA 21
3.2 MÓDULO DE DETECÇÃO DE OBJETOS 23
3.3 MÓDULO ISR 26
3.4 MÓDULO DE CLASSIFICAÇÃO DE IMAGENS 27
3.4.1 Definição da ferramenta para classificação 27
3.4.2 Modelo e conjunto de dados utilizado 28
3.5 LINGUAGEM DE PROGRAMAÇÃO UTILIZADA 29

4 RESULTADOS E DISCUSSÕES 31
4.1 TREINAMENTO DO CLASSIFICADOR 31
4.2 FUNCIONAMENTO DO PROGRAMA 32
4.3 RESULTADOS 35

5 CONCLUSÃO 37

REFERÊNCIAS 38

APÊNDICE A – TRECHO DO CÓDIGO DE YOLO PARA DETECÇÃO DE PESSOAS 41

APÊNDICE B – MÓDULO DE SUPER RESOLUÇÃO 42

APÊNDICE C – MÓDULO DE CLASSIFICAÇÃO DE IMAGEM 43

APÊNDICE D – MÓDULO PRINCIPAL 44

APÊNDICE E – CÓDIGO DE TREINAMENTO UTILIZANDO TÉCNICA DE *TRANSFER LEARNING* NAS 4 ÚLTIMAS CAMADAS DA REDE. **46**

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

O ano de 2020 foi atípico para toda a população mundial. A doença COVID-19 causada pelo novo Corona vírus, até o dia 29 de março de 2021 infectou 127 milhões de pessoas, causando 2,79 milhões de mortes. A informação é mostrada pelo monitor do Covid da Google, que reúne informações de diferentes fontes, incluindo departamentos de saúde ao redor do mundo. De acordo com este mesmo monitor, os Estados Unidos e o Brasil estão respectivamente em primeiro e segundo lugar no ranking de países com mais mortes pelo Covid-19.

Dentre várias das recomendações da OMS, a utilização de máscaras em ambientes públicos e o distanciamento social estão entre as medidas mais eficazes contra a doença. Porém, a fiscalização de tais medidas em ambientes de aglomeração se torna um desafio, uma vez que é difícil a presença de fiscais em todos estes pontos. Outro desafio é também estimar a quantidade de pessoas utilizando ou não máscaras nesses ambientes.

A doença causada pelo Corona vírus pode ser endêmica, tornando-se parte do cotidiano e fazendo com que a utilização de máscaras faciais possa vir a se tornar permanente (OMS, 2020). Dessa forma, é necessário que a população se conscientize em relação a utilização das máscaras em locais públicos para proteção individual e comum. Porém, segundo pesquisa da universidade de medicina de Cubatão, cidade localizada no litoral do estado de São Paulo, certa de 55% das pessoas não utilizam máscara ou não a utilizam de forma correta (G1, 2020). Tal informação corresponde a um alto risco de infecção da doença e pode ser relacionado a desinformação compartilhada através das redes sociais (ELPAIS, 2020).

De forma a ajudar na desmistificação algumas dessas notícias falsas, o site do Sistema Único de Saúde disponibilizou uma matéria, que cita uma calculadora desenvolvida por dois doutorandos poloneses (NISHIOKA, 2020). Nela, é possível preencher informações como porcentagem de pessoas utilizando máscaras, R0 (taxa de transmissão do vírus, que varia entre 2 e 4), bem como o material da máscara a ser utilizada que pode ser de algodão, cirúrgica, entre outras. Dados obtidos através da calculadora mostram que, mesmo com a utilização de máscaras, para uma porcentagem de 45% das pessoas utilizando corretamente a proteção facial feita de algodão, a uma taxa de transmissão de 2.5 indicada como padrão pela calculadora, tem-se que nenhuma vida é salva. Tal dado demonstra que, a utilização de máscara é uma adoção comunitária, que deve contar com o comprometimento de todos, e não uma atitude individual (MICHAŁOWSKA; CZERNIA, 2020).

Atualmente, temos a disposição uma enorme gama de algoritmos capazes de realizar o reconhecimento de objetos e pessoas, que são utilizados não só em ambiente acadêmico como também em empresas. Nossos celulares hoje, são capazes de realizar a detecção dos nossos rostos e aumentar a segurança dos aplicativos através dessa verificação, e as tecnologias não param de avançar, tornando inclusive carros autônimos – e até aeronaves autônomas – uma realidade (ÉPOCA, 2019).

Este trabalho foi realizado, então, com o intuito de ajudar na fiscalização do uso de máscaras, fazendo através de técnicas de processamento de imagens, a determinação da porcentagem de pessoas

as utilizando. O *software* proposto realiza o processamento de imagens que podem ser capturadas utilizando câmeras de Drones ou de segurança, e combina diferentes tipos de algoritmos de detecção para definição do valor de porcentagem e identifica através de caixas de detecção, quais pessoas estão utilizando máscara e quais não estão.

Para a realização do processamento de imagens, é necessário um computador que seja capaz de executar a aplicação e pode variar em desempenho a depender das configurações da máquina utilizada e do número de pessoas na imagem. O emprego real do *software* pode ser feito de diversas formas, uma vez que o programa é capaz de detectar uma lista de imagens presentes em qualquer pasta e processá-las em sequência. Dessa forma, torna possível integrar o programa a um outro sistema, onde o fiscal com a utilização de um Drone, pode tirar uma foto, enviar esta imagem para uma pasta localizada na nuvem, e ter esta imagem processada por um computador central responsável por devolver a resposta alguns minutos depois.

1.2 OBJETIVOS

Objetivo geral deste trabalho é apresentar um estudo a respeito dos conceitos de aprendizagem de máquina bem como aprendizado profundo de máquina para reconhecimento de imagens. O objetivo específico é a implementação de uma aplicação simples e prática para a detecção do uso de máscaras em indivíduos presentes em ambientes de concentração de pessoas, utilizando modelos já presentes na literatura.

1.3 ESTRUTURA DO TRABALHO

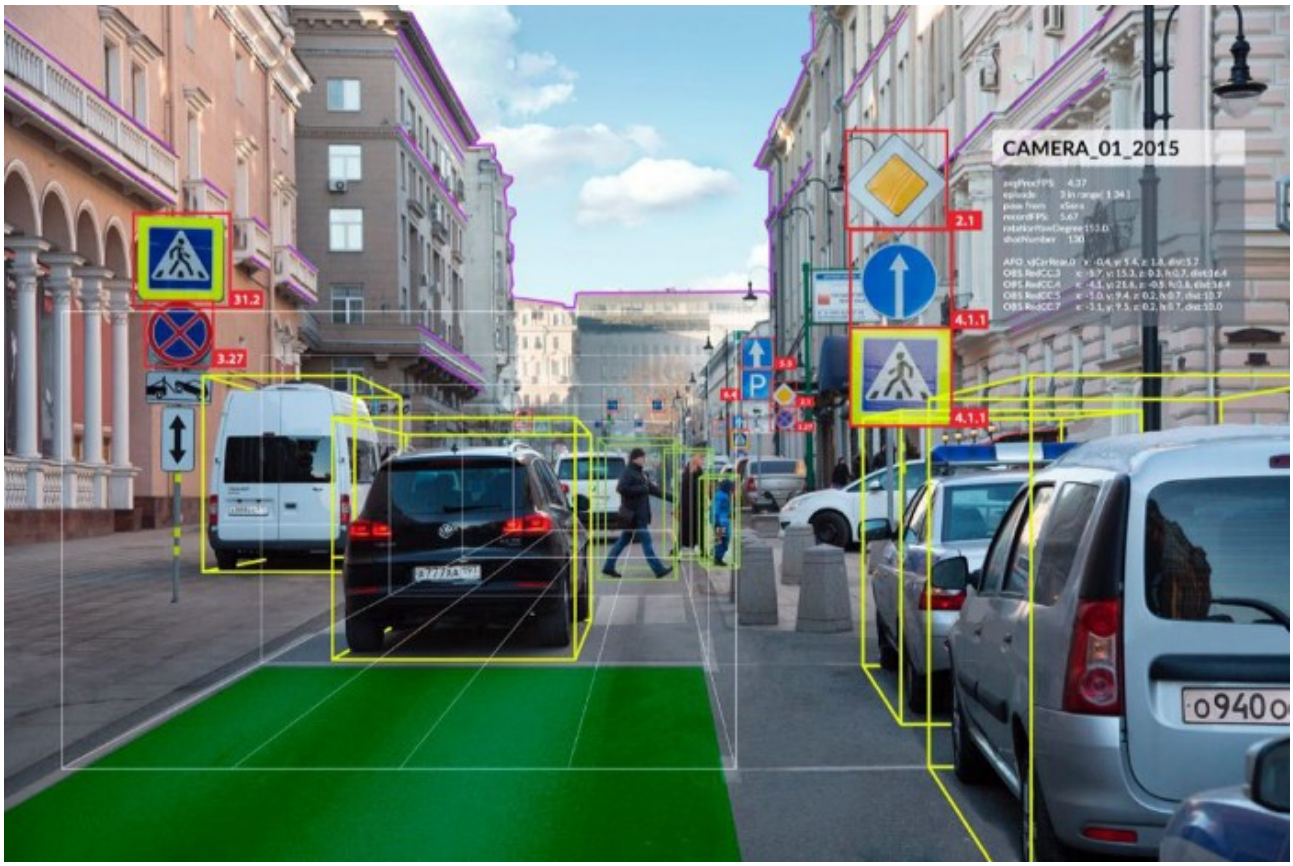
No capítulo 1 deste trabalho são apresentados a motivação e objetivos; no capítulo 2, mostra-se alguns dos trabalhos sendo desenvolvidos hoje com diferentes abordagens de aprendizado de máquina e visão computacional para detecção de máscaras, bem como alguns termos e teorias que presentes nos algoritmos estudados; o capítulo 3 discute-se a aplicação realizada exemplificando as escolhas da arquitetura adotada e suas aplicações; o capítulo 4, tem por função apontar as principais dificuldades no reconhecimento de imagens, e o desempenho da arquitetura escolhida. Por fim, no capítulo 5 é realizado a conclusão do trabalho com possíveis sugestões para desenvolvimentos de trabalhos futuros.

2 ASPECTOS TEÓRICOS

2.1 VISÃO COMPUTACIONAL E APRENDIZADO DE MÁQUINA

A chamada visão computacional é uma prática presente em muitas aplicações do mundo atual como na detecção de obstáculos em carros autônomos como na Figura 1, por exemplo. Visão computacional é definida como uma aplicação que utiliza métodos estatísticos para simplificar dados utilizando-se de modelos físicos, geométricos e teoria de aprendizado para geração de modelos (FORSYTH; PONCE, 2012).

Figura 1 – Exemplo de aplicação de visão computacional e aprendizado de máquina em veículos autônomos.



Fonte: Cohen (2020).

Ao olharmos para um objeto, a imagem capturada pelos olhos é transmitida para o nosso cérebro que, através do uso de diversos neurônios, pode classificar aquela imagem e reconhecer um objeto com base em suas características. Quando olhamos para uma bicicleta, sabemos que se trata de uma bicicleta, pois nosso cérebro consegue reconhecer suas características (duas rodas, pedais, quadro, correia, coroa, etc.) diferenciado-a de outros objetos. Mas para reconhecer algo, é necessário primeiro conhecê-lo e com computadores não é diferente.

Em um *software* comum as regras são estabelecidas pelo programador e tem por objetivo gerar respostas que utilizem os valores das entradas, além de, por exemplo, classificá-las de alguma forma.

Voltando ao exemplo da bicicleta, um programa estruturado poderia dizer se a imagem processada é uma bicicleta, comparando um vetor de informações sobre o objeto e verificando se o mesmo contém as características armazenadas em um vetor de características que a representam.

Em um programa que utiliza aprendizagem de máquina, o programador é responsável por oferecer as entradas, oferecer as respostas a essa entrada, e o programa é responsável por definir as regras, que se adaptam e se tornam melhores a medida que são treinadas. Enquanto no primeiro caso, o programa teria que ser desenvolvido do zero para reconhecer uma bicicleta de aro diferente, por exemplo, o *software* que utiliza aprendizagem de máquina, pode dizer se aquele objeto é ou não uma bicicleta dentro de um conjunto de outras imagens com diversos objetos, sendo capaz de generalizar seus resultados.

O aprendizado de máquina se relaciona com a visão computacional em muitas aplicações, e até alguns anos atrás, a barreira que não permitia a utilização em massa dessas técnicas era o poder computacional que pode ser muito custoso para o treinamento de modelos de aprendizado de máquina profunda. A utilização de GPU's (*Graphic Processing Unit*) de última geração, impulsionado pelas diversas melhorias gráficas presentes nos games ano após ano, fez com que o trabalho repetitivo que envolve o processo de treinamento de redes neurais, fosse passível de ser realizado, criando modelos ainda melhores e mais profundos.

2.2 REDES NEURAIS ARTIFICIAIS (RNA)

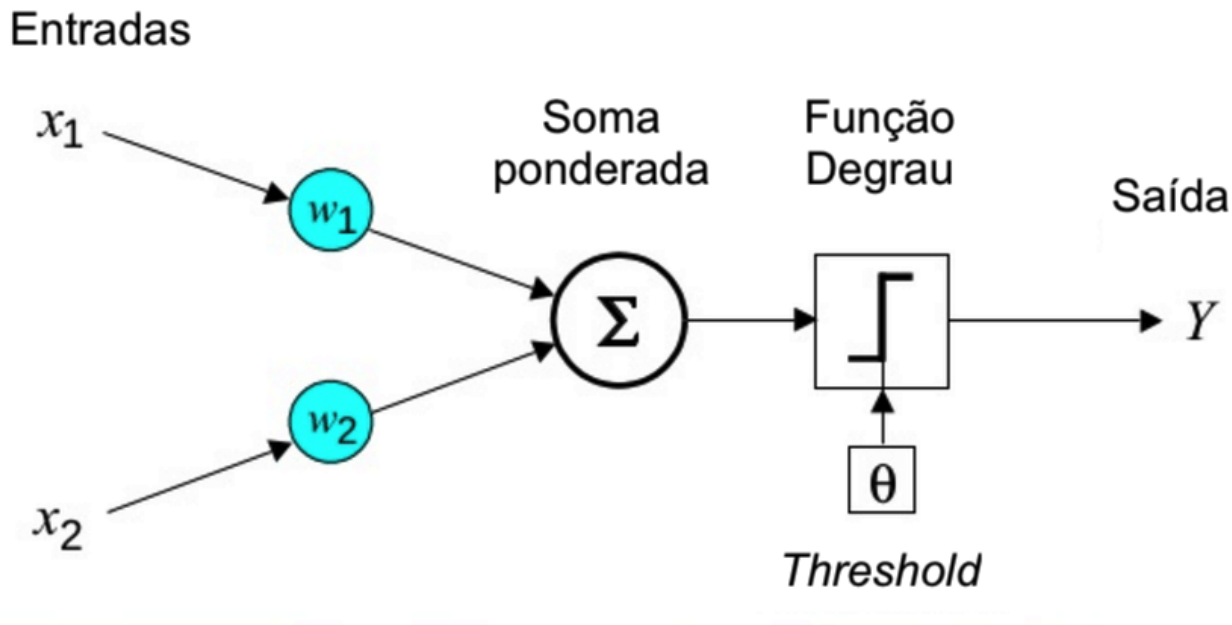
Estudos de redes neurais tiveram início com o objetivo de entender como o cérebro humano funcionava, de forma a construir uma máquina capaz de imitá-lo. Porém, esta imitação pode não ser bem literal. Assim como um avião é inspirado em pássaros, mas não tem a necessidade de bater asas para voar, redes neurais não funcionam exatamente como os neurônios humanos, e não devem ser limitadas a essa definição segundo Géron (2019). Os RNA são essenciais para compreender o aprendizado profundo de máquinas, como os modelos que utilizam redes neurais convolucionais.

Ao contrário do que se pode pensar, o estudo com redes neurais não é recente. Estudos iniciados em 1957, já indicavam uso de conceitos de redes neurais para resolução de problemas complexos (ROSENBLATT, 1957), e acreditava-se naquela época que em pouco tempo seria possível conversarmos com robôs. Ao longo dos anos os estudos minguaram, tendo uma volta na década de 90 com estudos relacionados a uma nova arquitetura, as das chamadas Máquinas de Vetores de Suporte. Atualmente vê-se uma nova onda da utilização de redes neurais, tornando-se comum, por exemplo, a comunicação com robôs no dia-a-dia (NGUYEN, 2020; MARR, 2018). O mundo tornou-se também, um lugar muito mais favorável há utilização dessas ferramentas, haja visto a quantidade de dados disponíveis hoje para utilização em treinamentos (CETAX, 2020) e o interesse por grandes corporações em pesquisas relacionadas ao assunto.

De forma a entender como redes neurais funcionam, pode-se considerar o estudo realizado por Rosenblatt em 1957 que apresenta o modelo do *Perceptron*, conhecida por ser um dos exemplos mais simples de uma RNA. Na Figura 2, pode-se observar que a rede tem como entradas os valores x_1 e x_2, multiplicados pelos respectivos pesos w_1 e w_2. Os dados são então somados uns aos outros e sujeitos a função de ativação Degrau que pode ser uma função *heaviside* ou uma função sinal a

depender da classificação a ser feita. O *threshold* define o limite inferior pelos quais os resultados serão considerados, que para o uso de uma função degrau, pode variar entre 0 e 1 (por exemplo, considerando valores de saída válidos, apenas como acima de um *threshold* de 0,2).

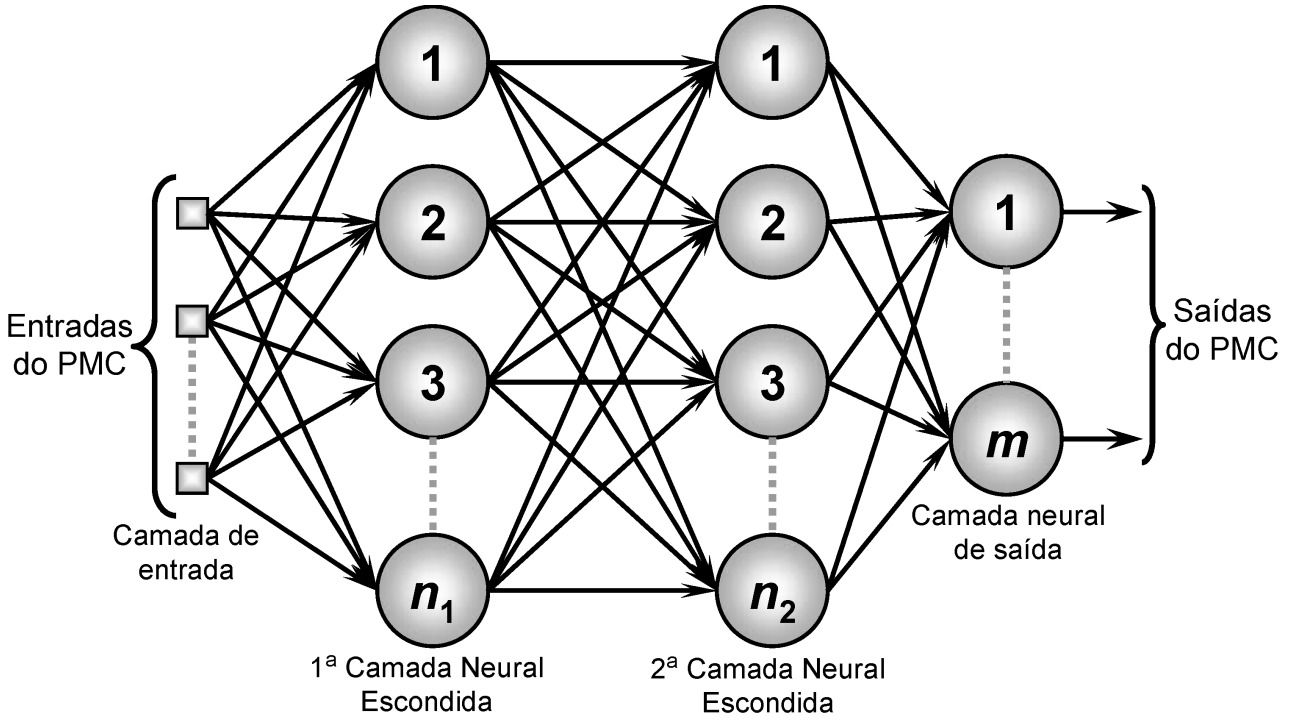
Figura 2 – Unidade Linear com *Threshold*.



Fonte: Adaptado de Rustan (1999).

De forma a exemplificar, pode-se com um único LTU definir uma aplicação que classifica categorias de flores, por exemplo, de acordo com os dados de altura e largura de cada pétala. Treinar uma rede neural desse tipo, significa encontrar os valores dos pesos w_1 e w_2, de modo a detectar determinado objeto, ou seja, os pesos vão refletir características presentes em cada classe, tornando possível determinar qual característica é mais relevante. Mas isso é apenas um neurônio, a combinação de diversos neurônios em paralelo, formando várias camadas, pode ser capaz de gerar ainda mais classificações com bases em mais características. A Figura 3 mostra um exemplo de uma rede neural profunda utilizando Unidades Lineares de *Threshold* como cada neurônio. As camadas localizadas entre a entrada e a saída são chamadas de camadas ocultas, pois não é possível determinar o seu valor de saída. Neste exemplo tem-se também uma rede totalmente conectada, ou seja, cada neurônio da camada anterior se conecta a cada neurônio da camada posterior. Chama-se rede neural profunda, qualquer rede que contenha duas ou mais camadas ocultas.

Figura 3 – Multicamada *Perceptron*.



Fonte: adaptado de Moreira (2018).

Porém, mesmo após a definição dos modelos em 1957, treiná-los de forma eficiente sempre foi um desafio, até que em 1985 um trabalho publicado revela um algoritmo de treinamento de retro-propagação (RUMELHART; HINTON; WILLIAMS, 1985). O algoritmo proposto por Rumelhart et al. trata-se de um Gradiente Descendente, que pode encontrar pontos de mínimo de uma função de custo através de um ponto inicial aleatório. Dessa forma, torna-se possível determinar os pesos para cada camada da rede neural.

2.3 REDES NEURAIS CONVOLUCIONAIS (CNN)

Levando-se em conta o modelo multicamada *Perceptron* visto na seção anterior, caso fosse necessária a análise de uma imagem e a determinação do que nela está contido, seria necessário ter uma entrada para cada píxel, o que se tornaria inviável do ponto de vista de processamento (GÉRON, 2019). De forma a ilustrar o problema, uma imagem de 128x128 pixels, por exemplo, necessitaria 16.384 neurônios apenas na camada de entrada. Além disso, existiriam ainda milhões de conexões para cada uma das camadas ocultas inteiramente conectadas. Torna-se então muito custoso a abordagem de uma RNA simples como a apresentada para a determinação de objetos em uma imagem, e foi necessário o desenvolvimento de outra abordagem.

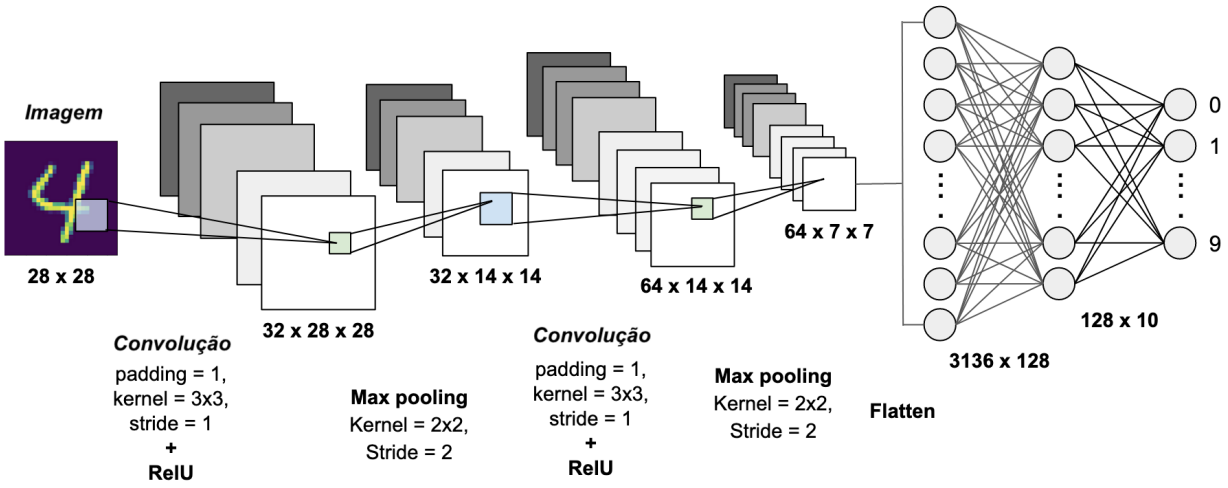
Tem-se então, as chamadas Redes Neurais Convolucionais (*Convolutional Neural Networks* ou CNN em inglês), que derivam de uma operação matemática chamada de convolução. A convolução, nada mais é do que a integral de uma função sobre outra no espaço onde ambas estão sobrepostas. A equação 1 mostra a operação de convolução de forma matemática, onde f e g são as funções envolvidas no processo de convolução. A equação mostra a notação discreta utilizada em processamento de

imagens em duas dimensões que representam a largura e o comprimento das camadas de imagem.

$$f * g(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i, j).g(x - i, y - j) \tag{1}$$

Dessa forma, supondo que a função f representa uma imagem, e a função g um *Kernel* (responsável por aplicar alguma categoria de filtro a imagem e extrair informação), teríamos como resultado uma nova camada, que possui uma dimensão menor a depender do passo - *Stride* em inglês - a ser utilizado. O passo, é o espaço entre as áreas onde o *Kernel* é aplicado. A Figura 4 mostra um exemplo de uma rede convolucional completa e através dela, explicam-se alguns conceitos importantes sobre componentes presentes nessa categoria de rede.

Figura 4 – Exemplo de rede neural convolucional completa para reconhecimento de algarismos de zero a nove.

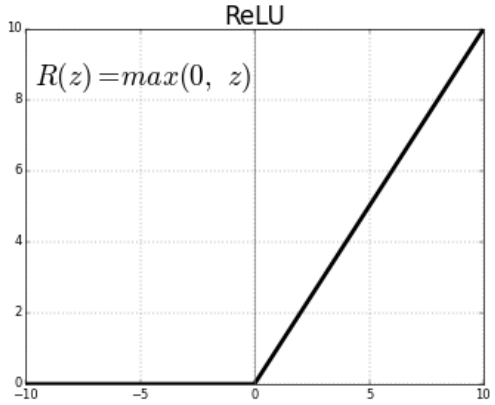


Fonte: adaptado de Shreyak (2020).

Padding: O primeiro parâmetro a ser mencionado, significa é o *padding* (preenchimento em português). Ele é necessário para que o *kernel* seja capaz de capturar os detalhes da borda da imagem, bem como adaptar a imagem para as dimensões de entrada da próxima camada. É adicionado uma faixa de zeros nas bordas de cada camada da imagem a ser lida, de forma a não alterar o resultado da camada onde será aplicada a convolução.

Função de ativação: na seção anterior, viu-se em um RNA a necessidade de uma função de ativação, a função degrau. Para a CNN não difere, sendo também necessária tal função para cada camada. Em grande parte das arquiteturas propostas, utiliza-se a função chamada ReLU, ilustrada na Figura 5, que é responsável por eliminar a linearidade e evitar o *overfitting* dos dados (quando a rede neural se pode detectar apenas os dados do treino, não sendo capaz de generalizar para o mundo real).

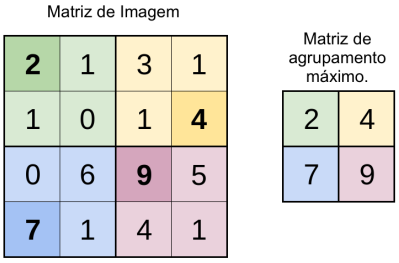
Figura 5 – Função de ativação ReLU.



Fonte: adaptado de Silva e Siebert (2020).

Camada de *pooling*: também conhecida como camada de agrupamento, é utilizada para redução de dimensionalidade. Normalmente utiliza-se a chamada *Maxpooling* que pode ser vista no exemplo da Figura 4, onde é extraído do *kernel* apenas o seu valor máximo, como mostra o exemplo da Figura 6.

Figura 6 – Função de agrupamento máximo aplicado a uma camada 4x4.



Fonte: adaptado de Walters (2019).

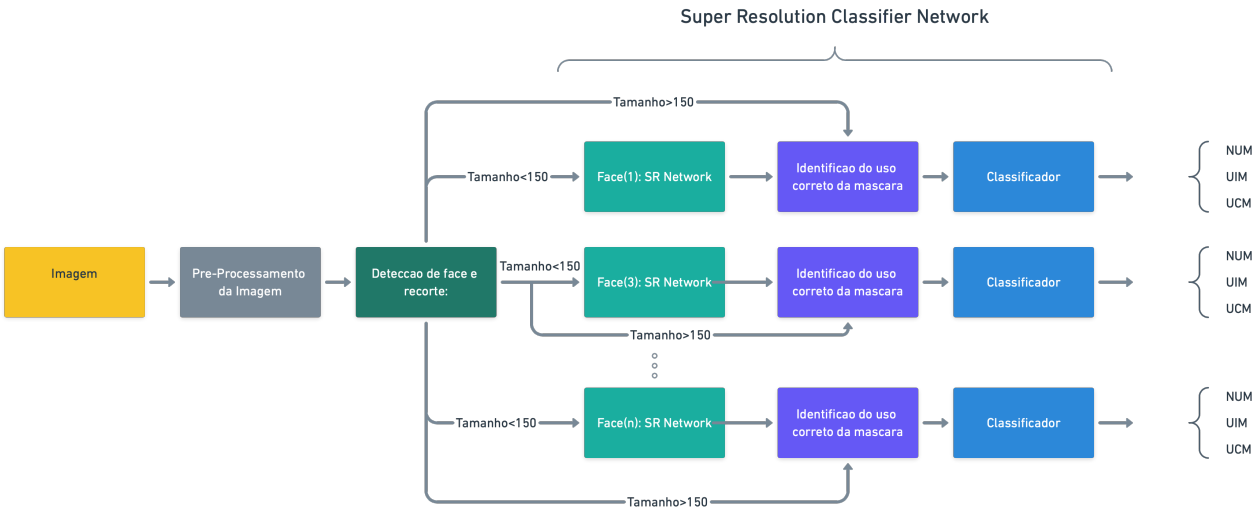
Camada de *Flatten*: ao final das redes convolucionais, é necessária uma camada resultante composta das características extraídas durante as camadas anteriores da rede neural. Essa camada, por ser a última e ter passado por uma série de camadas de agrupamentos, costuma ter dimensões reduzidas, onde é então aplicada a camada de *Flatten* que atua transformando esse tensor em um vetor linear. Tal vetor passa então pela aplicação de uma camada totalmente conectada, responsável por aplicar pesos e classificar a imagem da entrada. Normalmente, tem-se duas camadas densas, ou totalmente conectadas: a primeira normalmente contém a função de ativação ReLU, e a segunda, contém a função de ativação *Softmax* utilizada para quando se tem múltiplas classes. No exemplo da Figura 4, temos por tanto uma rede capaz de detectar diferentes dígitos, de 0 a 9, contendo 10 classes de saídas possíveis. A CNN irá calcular as probabilidades referentes a cada uma das saídas através da função *softmax*, e a de valor maior será a classe a qual a imagem pertence. Durante o processo de treinamento, muita da memória RAM do computador é utilizada, pois, é necessário armazenar todos os parâmetros de cada conexão das redes neurais, de forma a realizar a retro propagação responsável por determinar os pesos de cada conexão das camadas densas.

2.4 ESTADO DA ARTE

A detecção de faces utilizando máscaras segundo Loey (LOEY et al., 2021), por muito tempo esteve relacionada ao desafio do reconhecimento facial, muito utilizado em aplicações de segurança. A abordagem por ele realizada usa dois módulos principais, o primeiro consiste na aplicação do método chamado *Residual Neural Network ResNet50*, utilizada em aprendizagem profunda de máquina onde 49 camadas da rede já foram previamente treinadas, e apenas a última camada é alterada para a classificação de imagem desejada. O segundo método, remove a última camada da rede residual, e a substitui por um grupo de três classificadores clássicos de aprendizado de máquina, os chamados Maquina de Vetores de Suporte, árvore de decisão e método dos agrupamentos. São utilizados três conjuntos de dados diferentes, de modo a classificar a utilização correta da máscara, a utilização incorreta da máscara e a ausência da máscara

Em (QIN; LI, 2020) uma solução similar é proposta, também baseada no aprendizado profundo de máquina. O autor visa identificar a utilização de máscaras em ambiente público e utiliza de 3 funções distintas para este fim. Três redes neurais são implementadas, a primeira chamada de *multitask cascated convolutional neural network* busca identificar os rostos em uma imagem, tais rostos são então recortados da imagem e dependendo da sua resolução, quando esta se encontra abaixo de 150 px, aplica-se uma segunda rede neural, a chamada *image super resolution network*. Tal rede é responsável por restaurar detalhes de imagens, tornando possível a aplicação da terceira rede neural, que utilizará do conjunto de dados utilizado pelo autor, chamado de *Medical Masks Dataset* presente no site Kaggle (https://www.kaggle.com/vtech6/medical-masks-dataset), para a classificação proposta. O autor chama essa estrutura de SRCNET, e sua arquitetura pode ser vista no diagrama da Figura 7.

Figura 7 – Esquemático da SRCNet proposta por QIN at all



Fonte: adaptado de Qin e Li (2020).

3 MATERIAIS E MÉTODOS

De forma a viabilizar a classificação de imagens, foi necessário pré-processar as imagens das câmeras. A Figura 8 demonstra a abordagem utilizada, onde a imagem é capturada por um Drone ou câmera de segurança, armazenada em uma pasta a qual pode ser local ou hospedada na nuvem. Depois, é necessário reconhecer as pessoas na imagem (utilizando rede neural que será explicada mais a fundo adiante) de forma a recortar os torsos das pessoas detectadas, e assim facilitar na aplicação do módulo de classificação de imagens. Tal função e desempenhada pelo módulo *yoloModule()*, que tem trecho do código apresentado no Apêndice A. Para análise de imagens provenientes de câmeras de segurança, é necessário assegurar que tenhamos uma boa resolução de imagem, caso contrário, tem-se resolução insuficiente para processamento do módulo de classificação. Para tal tarefa, foi aplicado o módulo de ISR (CARDINALE, 2018) presente no Apêndice B, onde se tem o aumento da resolução dos recortes dos torsos identificados pelo *yoloModule()*. Por fim, com as imagens já recortadas e pré-processadas é feita a classificação das imagens, implementado pelo último módulo chamado de *maskDetection()* presente no Apêndice C. Todas as funções são chamadas pela função principal *Main()* presente no Apêndice D, responsável também por retornar para a interface de usuário as informações de quais pessoas estão utilizando máscaras e quais não estão.

Figura 8 – Esquemático da metodologia utilizada

Fonte: Produção do Próprio Autor.

3.1 IMAGEM DE DRONES E CÂMERAS DE SEGURANÇA

Como entrada para o sistema, utilizaram-se imagens de Drones e câmeras de segurança. O drone é ilustrado na Figura 9 é chamado de Tello, produzido pela empresa chinesa Ryze em parceria com a empresa também chinesa DJI. Tello tem como proposta ser acessível, apresentando uma câmera HD de 720p que pode capturar vídeo e fotos. O controle do dispositivo é realizado através de qualquer celular,

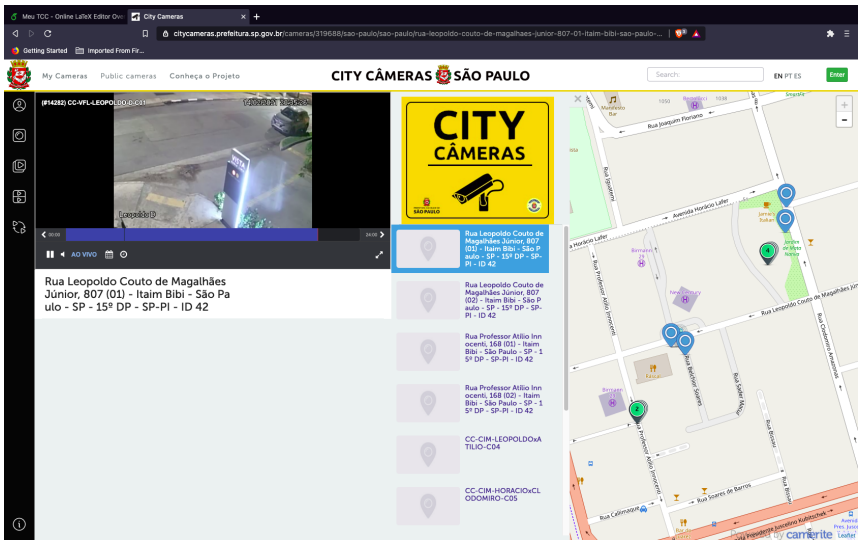
com aplicativo disponibilizado pela própria empresa, e a comunicação para o controle do Drone é feita a partir do sinal de wifi de 2.4 GHz. A autonomia do Tello é de cerca de 12 minutos, que se mostrou suficiente para as aplicações propostas pelo trabalho. As capturas de imagem são salvas diretamente no dispositivo utilizado para controlá-lo e podem ser facilmente compartilhadas.

Figura 9 – Drone utilizado na pesquisa

fonte: Produção do Próprio Autor.

Para a captura de câmeras, utilizou-se o site *city cameras* da governo da cidade de São Paulo. Através dele, é possível ter acesso às transmissões ao vivo as câmeras presentes na cidade, que possuam link com o website ilustrado na Figura 10. É possível inclusive cadastrar novos equipamentos de segurança privados utilizadas por moradores, viabilizando dessa forma, o aumento da rede de monitoramento da cidade, facilitando na detecção de crimes e apoiando na resolução dos mesmos.

Figura 10 – Website City Cameras do governo do estado de SP



Fonte: CITYCAMERAS (2021).

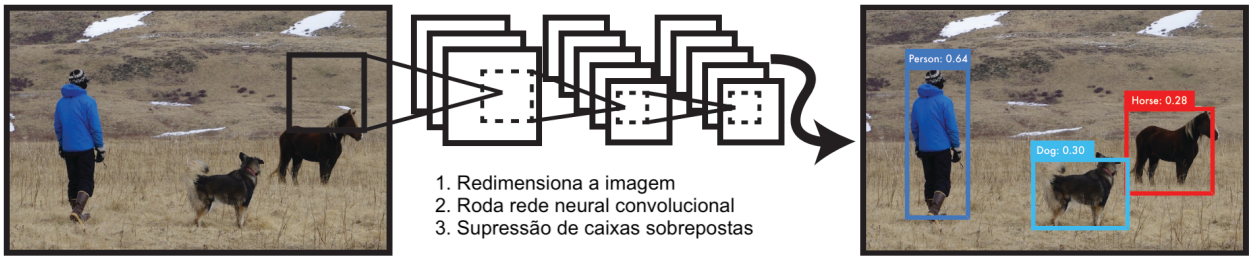
A imagem é extraída de uma dessas duas fontes (mas pode ser qualquer imagem do formato ".jpg") seja de câmeras particulares ou de drones particulares. É importante atentar-se para o ângulo de visão o qual o algoritmo foi adaptado. A utilização do módulo de detecção de objetos foi implementado para que se, através dele, fosse possível detectar as pessoas com uma distância maior, o que poderia

não ser viável caso fosse utilizado algum método de reconhecimento de faces, como o classificador *haarcascade frontal faces* (VIOLA; JONES, 2001) amplamente utilizado para este fim.

3.2 MÓDULO DE DETECÇÃO DE OBJETOS

Depois da coleta da imagem, a primeira etapa do pré-processamento é implementada. YOLO (*You Only Look Once*) da Figura 11 é um algoritmo desenvolvido em 2016 como parte das pesquisas de inteligência artificial do *Facebook* (REDMON et al., 2016). Ela se destaca pela sua capacidade de detecção de objetos de forma rápida. Diferente dos outros algoritmos de detecção de imagem que utilizam adaptações de classificadores de imagem, o YOLO detecta objetos através de regressão, recorrendo a um sistema de fluxo de dados simplificado. YOLO pode ser diretamente otimizado de ponta a ponta, pois usa apenas uma rede neural central (ou seja, diferente de outros sistemas que utilizam combinações de diferentes abordagens de rede neural, ele utiliza apenas uma rede, facilitando na hora de realizar um ajuste fino ou treinamento completo da ferramenta).

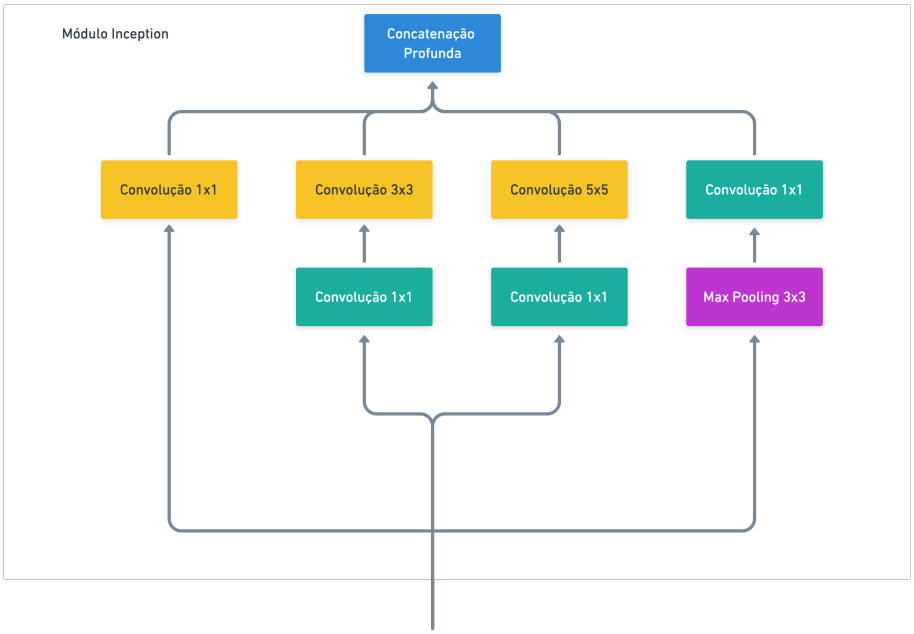
Figura 11 – Sistema de detecção do Yolo. O sistema (1) redimensiona a imagem para a resolução 448x448, (2) roda apenas uma rede convolucional na imagem, e (3) mostra os resultados obtidos através do modelo, após utilizar um sistema de supressão das caixas de detecção sobrepostas.



Fonte: adaptado de Redmon et al. (2016).

O algoritmo é baseado em uma rede neural desenvolvido pela Google, chamada de GoogLeNet (ZENG et al., 2016) que tem por característica a implementação das chamadas módulos *inception* apresentados na Figura 12 que torna possível a configuração de CNNs mais profundos, produzindo sistemas de aprendizado de máquinas muito mais precisos.

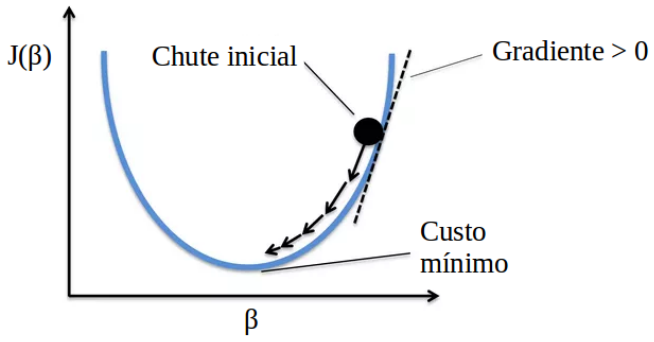
Figura 12 – Versão simplificada do módulo *Inception*



Fonte: adaptado de Géron (2019).

O chamado módulo *"inception"*, visa resolver um problema recorrente na detecção de imagens: o chamado *vanishing gradient* (desaparecimento de gradiente em português). O treinamento de redes profundas utiliza funções de otimização baseadas em uma operação chamada de gradiente descendente. Como dito nas seções anteriores, um modelo de aprendizado de máquina consiste na determinação de pesos de uma equação capaz de prever uma saída de acordo com os valores de entrada. Mas é necessário determinar o melhor conjunto de pesos para esta função, e para isso é necessário de uma operação chamada de função de custo: a cada estimativa realizada para determinação dos pesos da mesma, um custo é calculado, sendo importante que tal valor esteja em seu mínimo global para que se tenha um ponto de ótimo. Normalmente, segundo (GÉRON, 2019), a função de custo mais utilizada é a MSE (*Mean Squared Error*) que aparece em dezenas de aplicações diferentes. Para a diminuição deste custo, é utilizada a função de gradiente, que através de derivadas parciais, pode definir se o código se encontra em uma tendência de decida ou de subida, como mostra a Figura 13.

Figura 13 – Exemplo de gradiente descendente para encontro de ponto mínimo da função.



Fonte: adaptado de Vaz (2020).

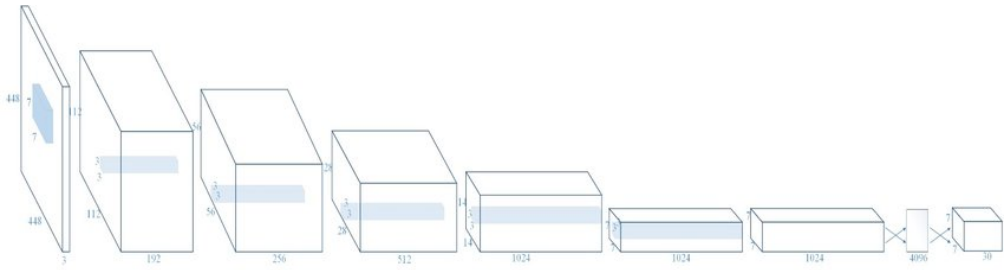
Porém, para redes muito profundas, tem-se uma dificuldade em se ter controle deste valor de gradiente, que tende a desaparecer a medida que se aprofundam devido aos milhares de variáveis que necessitam ser determinadas em cada camada.

De forma a resolver esse problema, o módulo *Inceptionv1*, apresentado pela primeira vez no *paper* de Zeng et al. (2016), tem por objetivo a otimização desse processo, ao colocar *kernels* diferentes em cada camada da rede neural, seguidas de uma camada de redução, de dimensão 1x1 de forma não perder o gradiente do início da mesma. Isso faz com que a rede preserve as suas características antes de passar para a próxima camada da arquitetura. Além disso, cada convolução possui um *padding* que normaliza as saídas dos diferentes *kernels*, que permite que os resultados de cada um deles sejam concatenados, gerando apenas um mapa das características.

O GoogLeNet possuí uma rede neural profunda, e YOLO se aproveita desta mesma arquitetura, utilizando 24 camadas de rede convolucional seguidas de duas redes totalmente conectadas, gerando um tensor de predição com dimensões 30 x 7 x 7. O algoritmo, porém, apresenta como limitação a incapacidade de detectar grupos de objetos pequenos e isso se dá devido às caixas de que servem de entrada para a rede, que faz com que cada área seja capaz de determinar apenas 2 caixas de detecção e apenas uma classe. Isso faz com que bandos de pássaros, por exemplo, que possuem grande volume de componentes com espaço pequeno entre eles, não sejam facilmente detectados pela ferramenta.

De forma a corrigir este problema, YOLO9000 foi desenvolvido com o objetivo de trazer a detecção de objetos para um outro nível ao se propor a realizar a detecção de mais de 9000 objetos diferentes (REDMON; FARHADI, 2017). Nesta versão, Redmon se predispõem a suprir a falta de dados entre classificação de imagens e detecção de objetos, ao combinar o treinamento de dois diferentes conjuntos de dados, o conjunto de detecção de objetos COCO - Common Objets in Context (LIN et al., 2014) e o conjunto para classificação de imagens ImageNet (RUSSAKOVSKY et al., 2015). O Yolov2, como também pode ser chamado, apresenta também melhorias significativas em suas métricas, ao apresentar um sistema rápido na detecção, capaz de ser executado em um grande número de objetos com tamanhos diferentes, principal melhoria em relação ao seu antecessor. A primeira versão tinha muita dificuldade na detecção de objetos que eram menores que os seus conjuntos de receptores, de forma que o YoloV2 separa em mais áreas de detecção e aplica um novo modelo de classificação, chamado de Darknet-19 Figura 14, que ocasiona em uma menor precisão em relação aos sistemas baseados em VGG-16 (SIMONYAN; ZISSERMAN, 2015), mas possui uma velocidade muito mais rápida.

Figura 14 – Arquitetura *Darknet-19* com 19 camadas de convolução e 5 camadas de *maxpooling*



Fonte: Ghenescu et al. (2018).

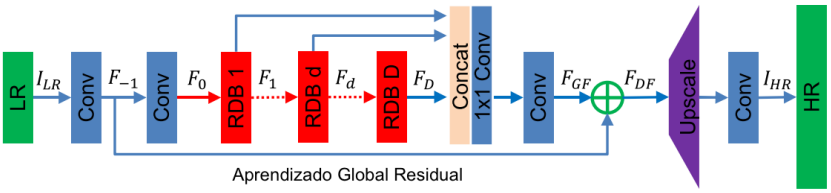
E por fim, YOLOv3 foi desenvolvido (REDMON; FARHADI; AP, 2017) com o objetivo de detectar os objetos de maneira mais rápida e precisa, através de uma pequena mudança na forma como as suas caixas de detecção são determinadas. Trata-se de uma versão com melhorias incrementais, o que não afetaram muito o seu funcionamento em relação a sua versão anterior. Agora, YOLO pode detectar mais de uma classe em simultâneo, em uma mesma caixa de detecção e utiliza também uma arquitetura mais profunda que o antecessor, capaz de determinar com mais precisão os objetos detectados devido à habilidade de sobrepor classes de detecção.

3.3 MÓDULO ISR

Por se tratar de câmeras de segurança e drones, é comum observar a baixa resolução das imagens a serem utilizadas. De forma a tornar possível a análise correta dessas imagens, o pré-processamento é essencial, pois evita que o módulo de classificação de imagens sofra com a presença de falsos positivos e falsos negativos. O módulo ISR (CARDINALE, 2018) foi então implementado em uma tentativa de mitigar esses efeitos. O projeto ISR usa 4 diferentes redes, a *super-scaling Residual Dense Network* (XU; STENGER, 2018), a *super-scaling Residual in Residual Dense Network* (WANG et al., 2018), VGG19 e a *General Adversial Network* (LEDIG et al., 2017). O projeto proposto por Cardinale fornece todos os modelos já encapsulados prontos para serem utilizados sendo necessário apenas selecionar o que melhor se encaixa para cada aplicação. Para o presente trabalho, utilizou-se o modelo RDN (*super-scaling Residual Dense Network*), e por conta disso, apenas este modelo será mais detalhado para melhor entendimento.

De acordo com a Figura 15, o modelo de Rede Densa Residual é composto de quatro principais componentes: a chamada característica de extração rasa líquida (SFENet), blocos residuais densos (RDB), a fusão de característica densa (DFF) e a amostragem líquida para cima (UPNet). De forma simplificada as primeiras camadas de convolução são utilizadas para fazer a extração rasa das características da imagem de baixa resolução (LR), ou seja, são duas camadas de convoluções simples aliadas a um *kernel* que ficará responsável pela primeira análise da imagem. Dessa forma, tem-se uma nova maneira de se lidar com o desaparecimento de gradiente, discutido na seção anterior.

Figura 15 – Arquitetura *Residual Dense Network*



Fonte: adaptado de Xu e Stenger (2018).

O módulo ISR recorre a uma interconexão entre os blocos anteriores e futuros, de forma que todas essas características são guardadas, concatenadas e são expostas com a função final *Upscale* responsável por prever e preencher os pixeis que faltam na imagem de alta resolução Tal imagem é então alvo de um novo processo de convolução e a entrega em uma resolução mais alta. Observou-se

que para esta aplicação, a RDN teve desempenho melhor com a função de "psnr-small"por isso tais configurações foram escolhidas.

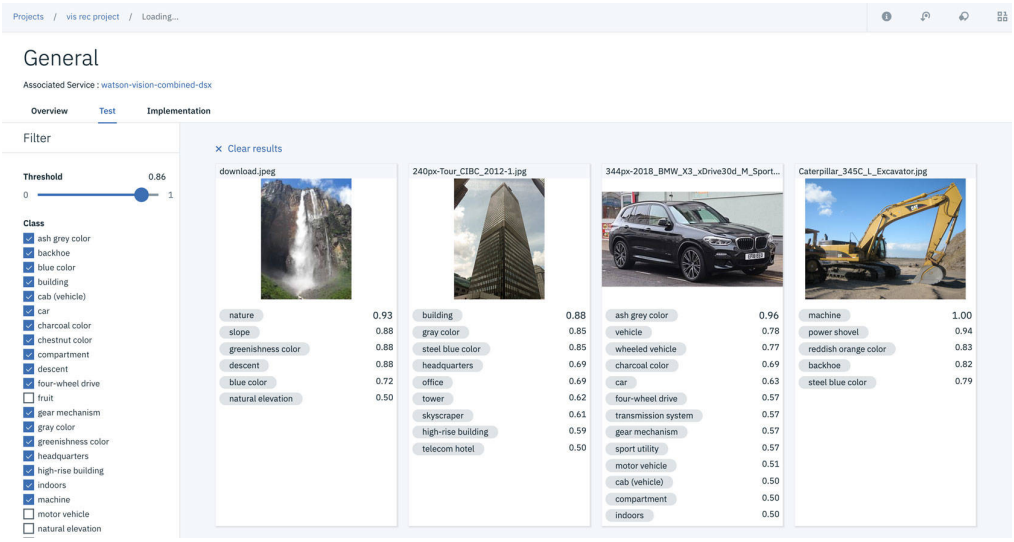
3.4 MÓDULO DE CLASSIFICAÇÃO DE IMAGENS

Após detectar a imagem das pessoas na foto e aumentar a sua resolução através do módulo de super resolução, tem-se uma imagem filtrada e devidamente pronta para ser classificada. Este módulo, diferente dos outros, foi concebido desde a definição de sua arquitetura até o seu treinamento e utilização. Explica-se a seguir, as dificuldades encontradas e pesquisas realizadas para definição da abordagem bem como a arquitetura geral do módulo.

3.4.1 Definição da ferramenta para classificação

Primeiramente, foi concebida a ideia de trabalhar com ferramentas já prontas para classificação da utilização de imagens. A proposta era realizar a construção de um sistema simples e explorar o poder das ferramentas já disponíveis no mercado, tornando a pesquisa mais prática e habilitada para servir de base para aplicações, inclusive, distintas das presentes no trabalho. A ferramenta de Inteligência Artificial da IBM chamada de Watson, foi a primeira alternativa (Figura 16). Por ser bastante flexível como API, encaixar a ferramenta de inteligência artificial em nuvem da IBM seria bastante útil também por não consumir poder de processamento do computador para o treinamento. Porém, assim como serviços hospedados na nuvem de outros provedores, o Watson é um serviço pago onde cada requisições tem um custo. Dessa forma, os testes com o classificador acabaram consumindo a cota destinada a contas universitárias (utilizada pelo autor para se eximir do pagamento do serviço) e esta abordagem acabou por ser descartada.

Figura 16 – IBM Watson, ferramenta de inteligência artificial da IBM



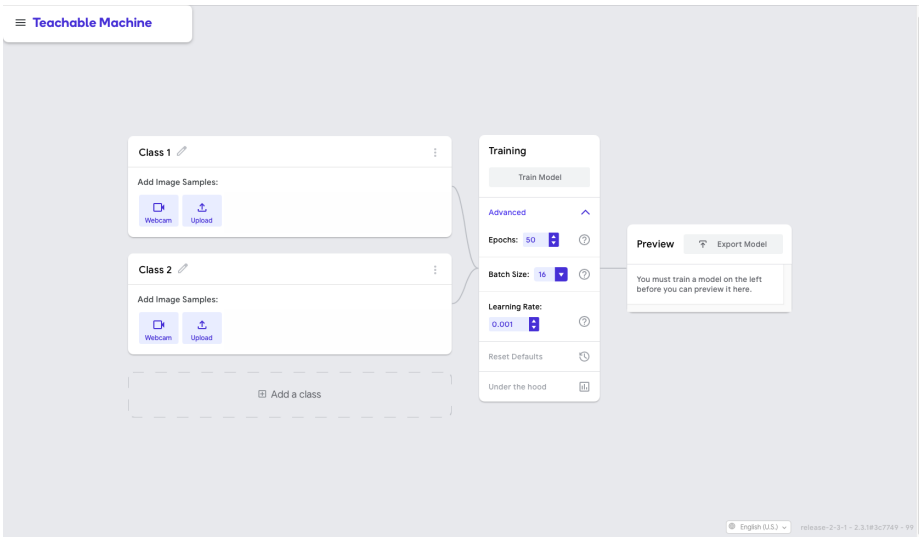
Fonte: IBM (2021).

A segunda abordagem idealizada, foi a utilização da ferramenta criada pela Google chamada de *Teachable Machine* (TM) apresentada na Figura 17. A TM tem muitas vantagens, pois é totalmente gratuita, possui interface com a ferramenta de Drive da Google (facilitando a manipulação dos conjuntos

de dados) e permite a alteração de alguns parâmetros a serem utilizados para o processo de treinamento, como os de Épocas (quantidade de imagens que fazem parte de uma mesmo conjunto de treinamento) e de Lotes. A ferramenta realiza também automaticamente a definição do conjunto de treino, testes e validação.

Porém, *Teachable Machine* não se saiu bem nas imagens em que foi testado, apresentando resultado muito bom para utilização da *webcam*, em ambiente de iluminação ideal e com a posição do rosto voltada para frente, mas muito ruim com imagens fornecidas na saída do módulo ISR, com a presença de muitos falsos negativos. A necessidade de internet nos dois casos também limita as aplicações reais da ferramenta, sendo necessária uma boa conexão para o uso do *software*.

Figura 17 – *Teachable Machine*, ferramenta de aprendizado de máquina da Google disponível no navegador.



Fonte: captura de tela do website da Google (2021).

Porém, por mais que não tenha sido implementada na solução final, a ferramenta TM possuí um conceito que foi utilizado, chamado de *Transfer Learning*, onde o treinamento não é feito do zero e apenas um ajuste fino no treinamento é realizado. Tal método extingue a exigência de um alto poder de processamento para o treinamento do modelo, e se aproveita dos pesos do modelo principal para predição de resultados. Sendo assim, a alternativa encontrada foi a geração de uma rede neural, utilizando o modelo VGG-16, construída através da biblioteca *Tensorflow* em Python.

3.4.2 Modelo e conjunto de dados utilizado

Por se tratar de um modelo de classificação simples, onde tem-se apenas duas classes (com máscara e sem máscara), utilizou-se o modelo chamado VGG-16 (PARKHI; VEDALDI; ZISSERMAN, 2015) que contém 16 camadas (considerado uma rede muito profunda de aprendizado de máquina). No presente trabalho, a reutilização deste modelo foi feita utilizando a técnica do *Transfer Learning* que tornou possível realizar um ajuste fino do modelo para o caso de uso a ser utilizado, onde apenas as 4 últimas camadas foram re-treinadas. A arquitetura utilizada pode ser vista na Figura 18, e as camadas fc6, dropout, fc7 e fc8 foram as que utilizaram treinamento demonstrado no código do Apêndice E.

Figura 18 – Arquitetura do classificador proposto

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1_1 (Conv2D)             (None, 224, 224, 64)      1792
_________________________________________________________________
conv1_2 (Conv2D)             (None, 224, 224, 64)      36928
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 112, 112, 64)      0
_________________________________________________________________
conv2_1 (Conv2D)             (None, 112, 112, 128)     73856
_________________________________________________________________
conv2_2 (Conv2D)             (None, 112, 112, 128)     147584
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 56, 56, 128)       0
_________________________________________________________________
conv3_1 (Conv2D)             (None, 56, 56, 256)       295168
_________________________________________________________________
conv3_2 (Conv2D)             (None, 56, 56, 256)       590080
_________________________________________________________________
conv3_3 (Conv2D)             (None, 56, 56, 256)       590080
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 28, 28, 256)       0
_________________________________________________________________
conv4_1 (Conv2D)             (None, 28, 28, 512)       1180160
_________________________________________________________________
conv4_2 (Conv2D)             (None, 28, 28, 512)       2359808
_________________________________________________________________
conv4_3 (Conv2D)             (None, 28, 28, 512)       2359808
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 14, 14, 512)       0
_________________________________________________________________
conv5_1 (Conv2D)             (None, 14, 14, 512)       2359808
_________________________________________________________________
conv5_2 (Conv2D)             (None, 14, 14, 512)       2359808
_________________________________________________________________
conv5_3 (Conv2D)             (None, 14, 14, 512)       2359808
_________________________________________________________________
pool5 (MaxPooling2D)         (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc6 (Dense)                  (None, 256)               6422784
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
fc7 (Dense)                  (None, 128)               32896
_________________________________________________________________
fc8 (Dense)                  (None, 2)                 258
=================================================================
```

Fonte: Produção do Próprio Autor.

A rede possui um total de 21.170.626 de parâmetros, que, se o modelo estivesse sendo treinado do zero, precisariam ser determinados utilizando as técnicas de treinamento como a retro propagação, o que custaria muito do processamento da máquina a ser utilizada. Porém, como estamos realizando apenas um ajuste fino, cerca de 13.535.362 parâmetros referentes as camadas mais externas necessitaram de treino, o que demorou em torno de 1 hora para as configurações presentes na lista a seguir utilizando 9 conjuntos de imagem por época, até que a época de número 30 fosse atingida.

- Processador Dual-Core Intel Core i5 de 2,7GHz;
- Memória Ram de 8 GB 1867 MHz DDR3;
- Unidade gráfica integrada Intel Iris Graphics 6100 1536 MB

Como *dataset*, utilizamos o chamado *Face Mask Dataset* disponibilizado no website Kaggle. O *dataset* possui cerca de 12 mil imagens, separadas em duas classes (com máscara e sem máscara), em três diferentes pastas destinadas ao treino, teste e validação. Cerca de 10.000 imagens são utilizadas para o treino sendo o conjunto considerado equilibrado por separar igualmente as imagens de ambas as classes.

3.5 LINGUAGEM DE PROGRAMAÇÃO UTILIZADA

Após a definição de todos os componentes da solução proposta, foi necessário a escolha de uma linguagem de programação que fosse capaz de manipular bem as imagens de treinamento e

utilizadas pela ferramenta, possuir suporte a alguma biblioteca de visão computacional para a leitura e processamento dos dados, bem como possuir suporte aos *frameworks* utilizados para o treinamento dos modelos. Primeiro, a implementação utilizando o Node-RED foi idealizada, uma linguagem de programação em blocos baseados em *JavaScript* para aplicações de IoT. O Node-RED possui uma vasta biblioteca de blocos desenvolvidos pela comunidade, suportando diversas aplicações diferentes.

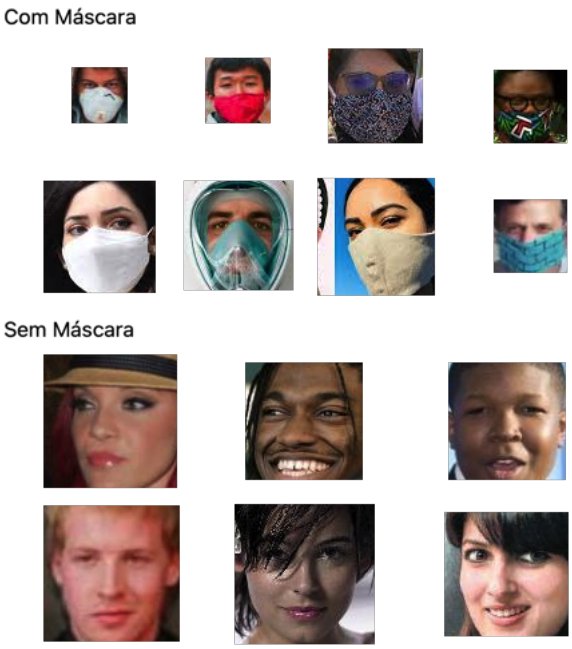
Porém, por ser necessário a integração de vários blocos distintos, que deveriam suportar interfaces de dados para comunicação, e devido a não familiaridade do autor com a linguagem *JavaScript*, foi escolhida a linguagem Python para o desenvolvimento da aplicação. Com o auxílio das bibliotecas PySimpleGUI, Keras, Tensorflow e OpenCV, foi possível construir um *software* que possuí interface de usuário e que pode manipular as imagens oferecendo a porcentagem de pessoas utilizando máscaras nas imagens mais facilmente, facilitando integração com outros sistemas, devido ao desenvolvimento modular adotado.

4 RESULTADOS E DISCUSSÕES

4.1 TREINAMENTO DO CLASSIFICADOR

Para a execução do programa, foi necessário treinar a rede neural responsável pela classificação e chegar a um modelo que fosse capaz de fazer a detecção do uso de máscaras. A Figura 19 apresenta algumas figuras contidas no conjunto de dados de treinamento utilizado, pode-se observar uma variedade de resoluções, etnias, expressões, posições e até mesmo tipos diferentes de máscara. É importante termos um conjunto de imagens variado para que seja possível tornar o *software* menos enviesado, sem que favoreça um tipo apenas de raça ou até mesmo uma mesma posição para a detecção das máscaras.

Figura 19 – Exemplo de imagens contidas no conjunto de dados utilizado

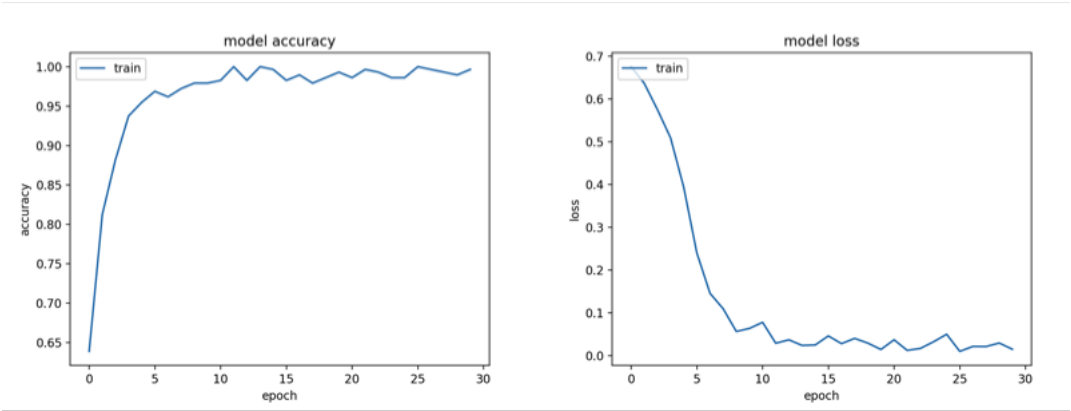


Fonte: Produção do Próprio Autor.

Os resultados do treinamento podem ser observados na Figura 20. O parâmetro utilizado de 30 épocas e lotes de 9 iterações ajustou-se bem ao modelo, haja visto que a sua medida de precisão se encontra próximo a 1 e a medida de erro se próxima de zero. Para o treinamento, o modelo de otimização utilizado foi o Adam, com passo de treinamento de 1.10^{-5}.

A definição do passo de treinamento é muito importante, pois um passo de menor magnitude é importante para o encontro do mínimo, mas pode prolongar o tempo de treinamento, ou ser levado a interpretar um ponto local como mínimo global. Para o conjunto de dados de validação, o desempenho da ferramenta também atingiu boas marcas se aproximando de 1, demonstrando que o modelo não sofreu sobre ajuste.

Figura 20 – Resultados do treinamento, na esquerda a curva de precisão por época e na direita a curva de perda por época.

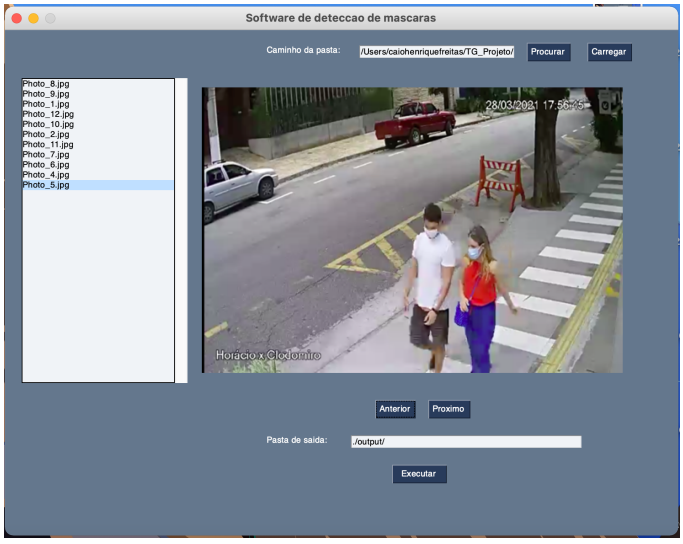


Fonte: Produção do Próprio Autor.

4.2 FUNCIONAMENTO DO PROGRAMA

A Figura 21 mostra a interface de usuário criada pela ferramenta utilizando a biblioteca PySimpleGUI que permite a criação de áreas de integração simples em Python. Na parte superior do programa temos o caminho para a pasta que contém as imagens, quando o caminho é especificado e o botão "Carregar"é pressionado inicia-se o carregamento dos arquivos de imagem. Através dos dois botões localizados abaixo da imagem, é possível navegar pelas imagens da pasta e no campo inferior, é possível especificar o caminho o qual os arquivos processados serão salvos. O botão "Executar"é responsável por iniciar o processamento das imagens.

Figura 21 – Interface de usuário do *software*.



Fonte: Produção do Próprio Autor.

Após a finalização da execução do programa, a coluna à esquerda é atualizada com as porcentagens de pessoas utilizando máscaras para cada imagem, onde o cálculo da porcentagem é feito como mostra

a 1. Na equação, tem-se que $N_{\text{com Máscara}}$ é o número de pessoas detectadas utilizando máscara e $N_{\text{sem Máscara}}$ é o número de pessoas detectadas que não estão utilizando máscara.

$$P_{\%} = (\frac{N_{comMascara}}{N_{comMascara} + N_{SemMascara}}) * 100 \quad (1)$$

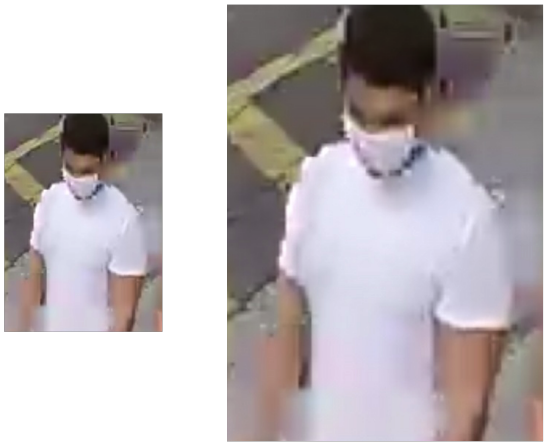
Para os testes com o *software* 14 imagens foram utilizadas de diferentes fontes, entre câmeras de segurança, imagens da internet e capturadas pelo Drone. Durante o processamento dessas imagens, em cada etapa, um arquivo é salvo com o resultado obtido evidenciando a detecção de máscaras através de caixas com cores diferentes para cada caso. De forma a fornecer uma captura que facilite a classificação das imagens, as caixas de detecção do YOLO tem sua altura divididas ao meio quando sua altura é maior que a sua largura, fazendo com que apenas o torso de cada pessoa seja enviado para a próxima etapa, para que o módulo de super resolução seja responsável pelo processamento da imagem. Para exemplificar o funcionamento do código, a Figura 22 mostra o recorte feito pelo código antes de aplicar o módulo de super resolução.

Figura 22 – Exemplo de recorte feito pelo software em imagem utilizada.

Fonte: Produção do Próprio Autor.

O módulo de Super Resolução é responsável por aumentar em 2x a resolução das imagens, no exemplo da Figura 23 as dimensões das imagens originais são de 215x297 e após o *software* de super resolução, foi expandida para 430x594. Pode ser observado que nenhuma informação é perdida entre às duas imagens, sendo a imagem da direita uma cópia fiel da primeira possuindo uma resolução maior, apenas com o intuito de melhorar a classificação feita pelo módulo seguinte. Antes de realizar a classificação, devido à arquitetura proposta, as imagens necessitam chegar ao próximo módulo com dimensões 224x224. O módulo de super resolução nos casos em que as imagens resultantes são superiores a este valor, apresentam densidade de pixeis maior facilitando a posterior classificação das imagens.

Figura 23 – Exemplo de processamento do módulo de super resolução.



Fonte: Produção do Próprio Autor.

Por fim, a imagem é então processada pelo módulo classificador e dependendo da probabilidade associada a classe, terá a porcentagem de pessoas utilizando máscara definida. O *software* posicionará na imagem uma caixa verde caso a pessoa esteja utilizando máscara ou vermelha caso não esteja utilizando. O módulo de classificação, porém, apresentou um alto índice de falsos positivos, principalmente para pessoas posicionadas de costas para a câmera, o que necessitou a adaptação dos códigos com valores de *threshold* personalizados, para melhorar a generalização do modelo proposto. Para a detecção de pessoas a imagem só será considerada válida e enviada para o próximo módulo caso o valor de confiança seja superior a 80% e para a classificação de máscaras, será apenas considerada a utilização de máscara caso o grau de confiança da predição seja superior a 90%. Valores abaixo disso são considerados sem máscara. Haja visto a limitação da ferramenta para o reconhecimento de pessoas utilizando máscara de forma incorreta, tais casos foram considerados reais Positivos, já que o conjunto de dados utilizado não apresenta treinamento para esse caso.

Figura 24 – Exemplos de imagens processadas pelo *software*

Fonte: Produção do Próprio Autor.

4.3 RESULTADOS

Para que a análise de resultados fosse possível, utilizou-se uma amostra de 14 imagens as quais serão alvo de estudo para a definição da precisão geral da ferramenta através do uso da Matriz de Confusão gerada. Ao todo 31 pessoas foram detectadas nas 14 imagens estudadas, cada uma delas sofreu aumento de resolução e foi avaliada separadamente pelo algoritmo no módulo de classificação de imagens. Na tabela 1 são indicados os valores de Positivos (P), Negativos (N), Falsos Positivos (FP) e Falsos Negativos (FN).

Tabela 1 – Matriz de Confusão do sistema completo

	Classe Esperada	
Classe Prevista	Com Máscara	Sem Máscara
Com Máscara	22 (P)	4 (FP)
Sem Máscara	0 (FN)	5 (N)

fonte: Produção do Próprio Autor.

O cálculo da precisão global da ferramenta é indicado na 2. No modelo proposto com base nas 14 imagens utilizadas para teste, obteve-se uma taxa de acerto de 87%.

$$Preciso_{\%} = (\frac{P+N}{P+N+FP+FN}) * 100 = 87,09\% \quad (2)$$

Durante o desenvolvimento do *software*, observaram-se algumas dificuldades para a determinação de alguns parâmetros e principalmente, na definição dos *thresholds* utilizados. O primeiro problema diz respeito a necessidade do módulo de alta resolução de imagem. O tempo de execução do programa sem o módulo de aumento de resolução é de apenas alguns segundos já que o YOLOv3, algoritmo utilizado para detecção de pessoas, é utilizado para aplicações em tempo real (a depender das configurações de computador em que se está o executando). Observou-se que, para a execução do programa sem o módulo de alta resolução, tem-se a duração do processamento de cada imagem fica em torno de 2 segundos. Com a adição do módulo de resolução de imagens esse tempo aumenta para cerca de 20 segundos, chegando a marca de até 100 segundos a depender da imagem e da memória RAM disponível no computador, inviabilizando a execução do programa para processamento de vídeos, por exemplo. Porém, a depender da resolução da câmera do dispositivo utilizado, deve-se considerar a relação tempo-qualidade e adaptar a arquitetura abordada para gravações e transmissões de vídeos ao considerar cada quadro como uma imagem a ser analisada.

Outro problema encontrado foi o alto índice de Falsos Positivos. Por utilizar-se de um conjunto de treinamento que considera apenas as imagens dos rostos das pessoas e não o corpo inteiro ou torso, a análise das imagens tornou-se mais desafiadora, e por vezes figuras de pessoas que se encontravam de costas para câmera como as vistas na Figura 25, deixaram o algoritmo confuso o levando a crer que as mesmas estariam utilizando máscara.

Figura 25 – Problemas com pessoas de costas nas imagens.

Fonte: Produção do Próprio Autor.

De forma a melhorar o desempenho para estes casos, a caixa de detecção foram adaptadas para a captura do torso das pessoas detectadas. Porém, apenas a diminuição das áreas não foi suficiente necessitando do ajuste dos *thresholds* do módulo de detecção de pessoas e do módulo de classificação de imagens, como mostrado anteriormente. Uma abordagem interessante, seria a captura de múltiplos quadros na mesma imagem, de forma a realizar uma votação entre as predições alcançadas e tentando melhorar na determinação da certeza em relação ao uso de máscaras. Esta abordagem, aliada a captura dos rostos das pessoas por meio da adição de mais um módulo a arquitetura proposta, pode levar a uma taxa ainda maior de acerto e em um ajuste menos agressivos nos parâmetros de detecção.

5 CONCLUSÃO

A pandemia causada pelo Corona vírus certamente não será esquecida nas gerações futuras dada os milhares de vidas perdidas devido à doença COVID - 19. A importância do distanciamento social e utilização de máscaras é inquestionável por autoridades sérias de saúde, mas a dificuldade no monitoramento e na fiscalização do uso das mesmas, tanto para medidas corretivas quanto para instrução da população, acaba acarretando um número de infecções que não para de crescer no Brasil. Com a utilização do *software* proposto, pode ajudar na instrução e detecção de locais de maior aglomeração e menor taxa de utilização de máscaras. Tal informação pode ajudar, por exemplo, na definição de zonas que necessitam de *lockdown* mais severo e zonas que podem ter medidas restritivas mais brandas, permitindo a abertura parcial de comércios e circulação, ainda que restrita, de pessoas em nesses locais onde a porcentagem de uso de máscara é alta. A arquitetura utilizada teve desempenho esperado para aplicação, com os devidos ajustes realizados, porém, espaços para melhorias futuras estão presentes. Viu-se que existem inúmeras abordagens diferentes para os algoritmos de classificação de imagens, sendo o escolhido um dos muito mencionados na literatura. O módulo de detecção de pessoas pode ser substituído por um capaz de detectar faces, que pode até mesmo utilizar o algoritmo YOLO, porém com uma forma diferente não focado na geração de caixas de detecção, mas visando detectar a cabeça das pessoas na imagem. Tal possibilidade pode permitir a captura das imagens da face das mesmas e melhorar o desempenho do algoritmo. Assim sendo, utilizando o algoritmo de classificação proposto e o módulo de aumento de resolução, pode-se esperar resultados que não dependam de ajustes externos nos *thresholds* de detecção, permitindo classificar as imagens de acordo com às duas classes já disponíveis. Como mencionado, vivemos um período muito próspero para o desenvolvimento de aplicações que utilizam Inteligência Artificial, haja visto o enorme volume de dados disponíveis hoje em diferentes conjuntos ao redor da internet o que pode fomentar ainda mais pesquisas relacionadas a está e encorajar soluções ainda melhores.

REFERÊNCIAS

CARDINALE, F. **Isr - image super resolution**. 2018. Disponível em: <https://github.com/idealo/image-super-resolution>. Acesso em: 10 fev. 2021.

CETAX. **Big data**: o que é, conceito e definição. 2020. Disponível em: <https://www.cetax.com.br/blog/big-data/>. Acesso em: 18 mar. 2021.

CITYCAMERAS. **Cameras da cidade de São Paulo.** 2021. Disponível em: <https://www.citycameras.prefeitura.sp.gov.br/>. Acesso em: 15 fev. 2021.

COHEN, J. **Computer vision applications in self-driving cars**. Becoming Human: Artificial Intelligence Magazine, 2020. Disponível em: <https://becominghuman.ai/computer-vision-applications-in-self-driving-cars-610561e14118>. Acesso em: 10 jan. 2021.

ELPAIS. **Verdades e mentiras sobre o uso de máscaras contra o coronavírus**. 2020. Disponível em: <https://brasil.elpais.com/internacional/2020-07-27/verdades-e-mentiras-sobre-o-uso-de-mascaras-contra-o-coronavirus.html>. Acesso em: 16 jan. 2021.

FORSYTH, D. A.; PONCE, J. **Computer Vision A Modern Approach**. New Jersey, Nova York, USA: Prentice Hall, 2012.

G1. **Pesquisa diz que 55% das pessoas usam máscara de forma errada no litoral de SP**. 2020. Disponível em: <https://g1.globo.com/sp/santos-regiao/noticia/2020/06/23/pesquisa-diz-que-55percent-das-pessoas-usam-mascara-de-forma-errada-no-litoral-de-sp.ghtml>. Acesso em: 16 jan. 2021.

GÉRON, A. **Hands-on machine learning with scikit-learn, keras, and fensorflow**: concepts, tools, and techniques to build intelligent systems. 1. ed. Beijing, China: O'Reilly, 2019.

GHENESCU, V. et al. Face detection and recognition based on general purpose dnn object detector. In: INTERNATIONAL SYMPOSIUM ON ELECTRONICS AND TELECOMMUNICATIONS, 2018, Timisoara, Romania. **Proceedings[...]**. Timisoara, Romania: IEEE, 2018.

GOOGLE. **Teachable machine website**. Google, 2021. Disponível em: <https://teachablemachine.withgoogle.com/>. Acesso em: 15 fev. 2021.

IBM. **Artificial intelligence**. 2021. Disponível em: <https://developer.ibm.com/technologies/artificial-intelligence/>. Acesso em: 15 fev. 2021.

LEDIG, C. et al. Photo-realistic single image super-resolution using a generative adversarial network. In: COMPUTER VISION AND PATTERN RECOGNITION, 2017, Puerto Rico. **Proceedings[...]**. Puerto Rico: IEEE, 2017.

LIN, T. et al. Microsoft coco: Common objects in context. In: EUROPEAN CONFERENCE ON COMPUTER VISION, 13th., 2014, Zurich, Switzerland. **Anais[...]**. Zurich, Switzerland: Springer International Publishing, 2014.

LOEY, M. et al. A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the covid-19 pandemic. **Measurement : journal of the International Measurement Confederation**, Elsevier Ltd, Budapest, Hungria, v. 167, 2021.

MARR, B. **Machine learning in practice**: how does amazon's alexa really work? Forbes Magazine, 2018. Disponível em: <https://www.forbes.com/sites/bernardmarr/2018/10/05/how-does-amazons-alexa-really-work/?sh=5a68145f1937>. Acesso em: 23 fev. 2021.

MICHAŁOWSKA, J.; CZERNIA, D. **Mask vs. no mask calculator**. 2020. Disponível em: <https://www.omnicalculator.com/health/mask-vs-no-mask>. Acesso em: 16 jan. 2021.

MOREIRA, S. **Rede neural perceptron multicamadas**. Ensina.AI, 2018. Disponível em: <https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9>. Acesso em: 18 mar. 2021.

NGUYEN, M. **How artificial intelligence and machine learning produced robots we can talk to**. Business Insider, 2020. Disponível em: <https://www.businessinsider.com/chatbots-talking-ai-robot-chat-machine>. Acesso em: 23 fev. 2021.

NISHIOKA, S. **Usar uma máscara salva quantas vidas? É fácil calcular**. 2020. Disponível em: <https://www.unasus.gov.br/especial/covid19/markdown/279>. Acesso em: 05 jan. 2021.

OMS. **Coronavirus may 'never go away,' says WHO official**. 2020. Disponível em: <https://edition.cnn.com/2020/05/14/health/coronavirus-endemic-who-mike-ryan-intl/index.html>. Acesso em: 16 jan. 2021.

PARKHI, O. M.; VEDALDI, A.; ZISSERMAN, A. Deep face recognition. In: PROCEEDINGS OF THE BRITISH MACHINE VISION CONFERENCE, 2015, Swancea, United Kingdom. **Resumos[...]**. Swancea, United Kingdom: BMVA Press, 2015.

QIN, B.; LI, D. Identifying facemask-wearing condition using image super-resolution with classification network to prevent covid-19. **Sensors**, Basel, Switzerland, v. 20, n. 18, 2020.

REDMON, J. et al. You only look once: unified, real-time object detection. In: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2016, Las Vegas, Nevada, USA. **Anais[...]**. Las Vegas, Nevada, USA: IEEE, 2016.

REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2017, Honolulu, HI, USA. **Anais[...]**. Honolulu, HI, USA: IEEE, 2017.

REDMON, J.; FARHADI, A.; AP, C. Yolov3: an incremental improvement. **arXiv preprint**, 2017.

The Perceptron - A Perceiving and Recognizing Automaton. Cheektowaga, New York, USA: Cornell Aeronautical Laboratory, 1957.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation. In: ____. **Parallel Distributed Processing: Explorations in the Microstructure of Cognition**. Cambridge: MIT Press, 1985. v. 1, cap. 8.

RUSSAKOVSKY, O. et al. Imagenet large scale visual recognition challenge. **International Journal of Computer Vision**, Springer, New York, v. 115, n. 3, 2015.

RUSTAN, A. Using artificial neural networks to improve hardware branch predictors. In: INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 1999, Washington, DC, USA. **Proceedings[...]**. Washington, DC, USA: IEEE, 1999.

SHREYAK. **Building a convolutional neural network (CNN) model for image classification.** Becoming Human: Artificial Intelligence Magazine, 2020. Disponível em: <https://becominghuman.ai/building-a-convolutional-neural-network-cnn-model-for-image-classification-116f77a7a236>. Acesso em: 18 mar. 2021.

SILVA, A. D.; SIEBERT, C. Densenet-dc: optimizing densenet parameters through feature map generation control. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v. 27, n. 3, 2020.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, 2015, San Diego, CA, USA. **Anais[...]**. San Diego, CA, USA: the dblp computer science bibliography, 2015.

VAZ, A. **Gradientes descendentes na prática - melhor jeito de entender**. Data Hackers, 2020. Disponível em: <https://medium.com/data-hackers/gradientes-descendentes-na-prática-melhor-jeito-de-entender-740ef4ff6c43>. Acesso em: 20 mar. 2021.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: PROCEEDINGS OF THE 2001 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION., 26th., 2001, Kauai, HI, USA. **Anais [...]**. Washington: IEEE, 2001.

WALTERS, A. **Convolutional neural networks (CNN) to classify sentences**. 2019. Disponível em: <https://austingwalters.com/>. Acesso em: 20 mar. 2021.

WANG, K. X. et al. Esrgan: Enhanced super-resolution. In: THE EUROPEAN CONFERENCE ON COMPUTER VISION WORKSHOPS, 2018, Munich, Germany. **Proceedings[...]**. Munich: Springer, 2018.

XU, Y. J.; STENGER, A. B. Dense bynet: Residual dense network for image super resolution. In: IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING (ICIP), 25th., 2018, Athens, Greece. **Proceedings[...]**. Athens, Greece: IEEE, 2018.

ZENG, G. et al. Preparation of novel high copper ions removal membranes by embedding organosilane-functionalized multi-walled carbon nanotube. **Journal of Chemical Technology and Biotechnology**, Oxford, United Kingdom, n. 8, 2016.

ÉPOCA. **Embraer faz o primeiro teste de aeronave autônoma no Brasil**. 2019. Disponível em: <https://epocanegocios.globo.com/Empresa/noticia/2019/10/embraer-faz-o-primeiro-teste-de-aeronave-autonoma-no-brasil.html>. Acesso em: 10 jan. 2021.

APÊNDICE A – TRECHO DO CÓDIGO DE YOLO PARA DETECÇÃO DE PESSOAS

```
#Função responsável por receber as imagens presentes na pasta,reconhecer as
#pessoas presentes
#nas imagens, extrair um recorte de seus torsos para que possam ser
#posteriormente processadas
def yoloModule(image):
    #carrega o modelo que será utilizado para detecção
    yolo= "./yolo_object_detection/yolo-coco"
    #carrega os argumentos que serão utilizados pela função
    args = getYoloArguments(image, yolo)
    #carrega as configurações que serão utilizadas no processo de detecção
    #de pessoas na imagem
    LABELS, original_image, image, COLORS, net, H, W = setConfigs(args)
    #determina as saídas que serão consideradas na hora de definir a detecção
    #das pessoas na imagem
    idxs, boxes, classIDs, confidences = layersOutput(net, image, args, LABELS,
    H, W)
    #dicionário contendo os dados processados pelo YOLO
    detection_dict = {"idxs": idxs, "boxes": boxes, "classIDs": classIDs,
     "confidences": confidences, "LABELS": LABELS, "original_image":
     original_image, "image": image, "COLORS": COLORS,}
    #recorta as imagens reconhecidas da imagem e as salva no vetor que será
    #utilizado para o reconhecimento de imagens
    cropped_images, crop_dict_boxes = detectionFunction(detection_dict)

return cropped_images, crop_dict_boxes

```

APÊNDICE B – MÓDULO DE SUPER RESOLUÇÃO

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import cv2
from ISR.models import RDN
import uuid
#função utilizada para o aumento da resolução das imagens, utilizando
#rede neural residual densa
def superResolutionConvertion(cropped_images):
    SR_cropped_images=[]
    #escolha do tipo de rede e definição do modelo a ser utilizado
    #PSNR - Peak Signal to Noise Ratio, e uma forma de calcular a qualidade
    #de um sinal
    #o modelo foi escolhido por oferecer melhor desempenho no software
    rdn = RDN(weights='psnr-small')
    #contador para salvar as imagens em ordem
    img_num = 1
    for img in cropped_images:

        lr_img = np.array(img)
        sr_img = rdn.predict(lr_img)
        new_img = Image.fromarray(sr_img)

        new_img = cv2.cvtColor(np.array(new_img), cv2.COLOR_RGB2BGR)
        #lista contendo todas as imagens em alta resolução
        SR_cropped_images.append(new_img)
        new_img = cv2.cvtColor(np.array(new_img), cv2.COLOR_BGR2RGB)
        #salva as imagens para utilização dos resultados no trabalho
        result = cv2.imwrite("/Users/caiohenriquefreitas/TG_Projeto/ \
        Mask_Detection/output/SR_Crops/"+str(uuid.uuid1())\
        +"_"+str(img_num)+".jpg", new_img)
        img_num+=1

    return SR_cropped_images

```

APÊNDICE C – MÓDULO DE CLASSIFICAÇÃO DE IMAGEM

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import cv2
from ISR.models import RDN
import uuid
#função utilizada para o aumento da resolução das imagens, utilizando
#rede neural residual densa
def superResolutionConvertion(cropped_images):
    SR_cropped_images=[]
    #escolha do tipo de rede e definição do modelo a ser utilizado
    #PSNR - Peak Signal to Noise Ratio, e uma forma de calcular a qualidade
    #de um sinal
    #o modelo foi escolhido por oferecer melhor desempenho no software
    rdn = RDN(weights='psnr-small')
    #contador para salvar as imagens em ordem
    img_num = 1
    for img in cropped_images:

        lr_img = np.array(img)
        sr_img = rdn.predict(lr_img)
        new_img = Image.fromarray(sr_img)

        new_img = cv2.cvtColor(np.array(new_img), cv2.COLOR_RGB2BGR)
        #lista contendo todas as imagens em alta resolução
        SR_cropped_images.append(new_img)
        new_img = cv2.cvtColor(np.array(new_img), cv2.COLOR_BGR2RGB)
        #salva as imagens para utilização dos resultados no trabalho
        result = cv2.imwrite("/Users/caiohenriquefreitas/TG_Projeto/ \
        Mask_Detection/output/SR_Crops/"+str(uuid.uuid1())\
        +"_"+str(img_num)+".jpg", new_img)
        img_num+=1

    return SR_cropped_images

```

APÊNDICE D – MÓDULO PRINCIPAL

```
#Função principal responsável pela chamada das funções auxiliares
#que compõem o sistema completo de execução
import os, sys
import cv2
import matplotlib.pyplot as plt
sys.path.append(os.path.dirname(os.path.dirname(os.path.realpath(__file__))))
import VGG16_project.Mask_detection_module as mask
import SR_module.ISR_module as SR
import yolo_object_detection.yolo_module as yolo
import time
#Declaração das variáveis globais
global output_files, results
output_files = []
results = []
#Função principal
def main(image_path="/Users/caiohenriquefreitas/TG_Projeto/Mask_Detection/ \
input/Photo_3.jpg", output_file="/Users/caiohenriquefreitas/TG_Projeto/ \
Mask_Detection/output/output_image.jpg"):
  print(output_file)
  original_image = cv2.imread(image_path)
  original_image = cv2.cvtColor(original_image, cv2.COLOR_RGB2BGR)
  color = {"MASCARA": (0,255,0), "SEM MASCARA": (255,0,0)}
  start = time.time()
  #extracao dos torsos das pessoas reconhecidas
  cropped_images, crop_dict_boxes = yolo.yoloModule(image_path)
  SR_cropped_images = SR.superResolutionConvertion(cropped_images)
  #caso a imagem contenha pessoas e o software de super resolução funcione
  if SR_cropped_images!=[] and SR_cropped_images!=None:
    #realiza software de detecção
    total_mask_probabilities = mask.maskDetection(SR_cropped_images)
    end = time.time()
    # mostra tempo de reconhecimento dos objetos na imagem
    print("Software demorou {:.6f} segundos para processar as imagens.". \
    format( end - start))
    ppl_with_mask = 0
    ppl_without_mask = 0
    #função que desenha e contabiliza as caixas de detecção para cada opção
    for i in range(len(SR_cropped_images)):
      x,y = crop_dict_boxes["crop_"+str(i+1)][0],crop_dict_boxes["crop_"+ \
      str(i+1)][1]
      w,h = crop_dict_boxes["crop_"+str(i+1)][2],crop_dict_boxes["crop_"+ \
      str(i+1)][3]
      if total_mask_probabilities["crop_"+str(i+1)][0] != '':
        cv2.rectangle(original_image, (x, y), (x + w, y + h),\
        color[total_mask_probabilities["crop_"+str(i+1)][0]], 2)
```

```
        text = "{:.4f},{:.4f} ".format(total_mask_probabilities["crop_"+\
        str(i+1)][1], total_mask_probabilities["crop_"+str(i+1)][2])
        cv2.putText(original_image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
          0.5, color[total_mask_probabilities["crop_"+str(i+1)][0]], 2)
        if (total_mask_probabilities["crop_"+str(i+1)][0] == "MASCARA" :
          ppl_with_mask += 1
        if (total_mask_probabilities["crop_"+str(i+1)][0] == "SEM MASCARA" ):
          ppl_without_mask += 1
    # calculo da porcentagem de pessoas utilizando máscara
    ppl_total = ppl_with_mask+ppl_without_mask
    if ppl_total!=0:
      image_result = (ppl_with_mask/ppl_total)*100.0
    else:
      image_result = 0
    print ("A porcentagem de pessoas usando mascara nesta imagem é de: {:.2f}" \
    .format(image_result))
    #salva imagem processada
    processed_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
    result= cv2.imwrite(output_file,processed_image)
    if result==True:
      print(output_file, "File saved successfully")
      output_files.append(output_file)
      results.append(image_result)
    else:
      print("Error in saving file")


def getOutputFiles():
  return output_files, results

if __name__ == "__main__":
  main()

```

APÊNDICE E – CÓDIGO DE TREINAMENTO UTILIZANDO TÉCNICA DE *TRANSFER LEARNING* NAS 4 ÚLTIMAS CAMADAS DA REDE.

```
from PySimpleGUI.PySimpleGUI import Drop
from tensorflow import keras
from keras.applications.vgg19 import VGG19
from keras import Sequential
from keras.engine import Model
from keras.layers import Flatten, Dense, Input, Dropout
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np
import cv2
import sys, os
from keras.callbacks import CSVLogger
sys.path.append(os.path.dirname(os.path.dirname(os.path.realpath(__file__))))
from keras_vggface import VGGFace

#Caminhos das pastas de treinamento, teste e validacao
train_dir = '/Users/caiohenriquefreitas/TG_Projeto/Mask_Detection/input/\
Face Mask Dataset/Train'test_dir = '/Users/caiohenriquefreitas/TG_Projeto/
    Mask_Detection/input/\
Face Mask Dataset/Test'
val_dir = '/Users/caiohenriquefreitas/TG_Projeto/Mask_Detection/input/\
Face Mask Dataset/Validation'

#Responsável por gerar figura contendo erro e precisão por época.
def generate_graph(history):
    plt.plot(history.history['Precisao'])
    plt.title('Precisao do Modelo')
    plt.ylabel('Precisao')
    plt.xlabel('Epoca')
    plt.legend(['train'], loc='upper left')
    plt.show()

    plt.plot(history.history['Erro'])
    plt.title('Erro do modelo')
    plt.ylabel('Erro')
    plt.xlabel('Epoca')
    plt.legend(['train'], loc='upper left')
    plt.show()


# Geração de imagens otimizadas para completar conjunto de dados

train_datagen = ImageDataGenerator(rescale=1.0/255, horizontal_flip=True,\
zoom_range=0.2,shear_range=0.2)
```

```
44
45 train_generator = train_datagen.flow_from_directory(directory=train_dir,\
46 target_size=(224,224),class_mode='categorical',batch_size=32)
47
48 val_datagen = ImageDataGenerator(rescale=1.0/255)
49 val_generator = train_datagen.flow_from_directory(directory=val_dir,\
50 target_size=(224,224),class_mode='categorical',batch_size=32)
51
52 test_datagen = ImageDataGenerator(rescale=1.0/255)
53 test_generator = train_datagen.flow_from_directory(directory=val_dir,\\
54 target_size=(224,224),class_mode='categorical',batch_size=32)
55
56 #Arquivo de Log para cada modelo
57 csv_logger = CSVLogger('VGG16_Ultima_4layers.log', separator=',', append=False)
58
59 #N mero de classes do modelo
60 nb_class = 2
61
62 #Modelo pré carregado utilizando pesos do VGGFaces
63 vgg_model = VGGFace(model="vgg16", weights = 'vggface', include_top=False,\
64      input_shape=(224, 224, 3))
65
66 #Congelamento do modelo, extraindo as  ltimas  4 classes para treinamento
67 for layer in vgg_model.layers[:-4]:
68     layer.trainable = False
69
70 #Extração da  ltima  camada a ser utilizada para entrada das camadas treináveis
71 last_layer = vgg_model.get_layer('pool5').output
72
73 #Camadas treinaveis
74 x = Flatten(name='flatten')(last_layer)
75 x = Dense(256, activation='relu', name='fc6')(x)
76 x= Dropout(0.5)(x)
77 x = Dense(128, activation='relu', name='fc7')(x)
78 out = Dense(nb_class, activation='sigmoid', name='fc8')(x)
79
80 #Geração do modelo
81 model = Model(vgg_model.input, out)
82
83 #Configuração do Otimizador
84 opt = keras.optimizers.Adam(learning_rate=1e-5)
85
86 #Compilação do Modelo e resumo da arquitetura que
87 # utilizado no trabalho de graduação
88 model.compile\
89 (loss='categorical_crossentropy', optimizer=opt, metrics ="accuracy")
90 model.summary()
91
92 #Treinamento do modelo com Batch size de 32 e epochs de 30
```

```
history = model.fit(train_generator,
                    steps_per_epoch=len(train_generator)//32,
                    epochs=30, validation_data=val_generator,
                    validation_steps=len(val_generator)//32,
                    callbacks=[csv_logger],
                    )

#Avaliação do modelo para os casos de teste
model.evaluate(test_generator)

#Armazenamento do modelo a ser utilizado para predição do sistema
model.save('/Users/caiohenriquefreitas/TG_Projeto/ \
    Mask_Detection/VGG16_project/masknet_VGG16.h5')
#Geração do c digo
generate_graph(history)

#Teste com imagem do autor
sample_mask_img = cv2.imread('/Users/caiohenriquefreitas/TG_Projeto/\
    Mask_Detection/input/exemplos/Photo.jpg')
sample_mask_img = cv2.resize(sample_mask_img,(224,224))
plt.imshow(sample_mask_img)
plt.show()
sample_mask_img = np.reshape(sample_mask_img,[1,224,224,3])
sample_mask_img = sample_mask_img/255.0
print(model.predict(sample_mask_img))
            \begin{lstlisting}[language=Python]


```