D.M.F.: uma solução para detectar o uso da máscara facial

Marcos Miguel Ferreira Gomes, Carlos Roberto Santos Junior

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Campus Hortolândia CEP 13183-250 - Hortolândia-SP – Brasil

marcos.fgomes@outlook.com, carlos.rsantos@ifsp.edu.br

Abstract. *COVID19 is a disease linked to a virus. The coronavirus can be through the mouth or nose of an infected person, for this reason the use of the face mask is mandatory in a pandemic period. Therefore, this article aims to present a methodology capable of verifying whether or not an individual is wearing a face protection mask. The developed model uses computer vision techniques through the OpenCV library together with a Deep Learning model. The tests carried out explore different approaches and ways in the use of the mask and some results obtained detection rates of up to 100%. In times of all care is necessary and all technology that helps prevent society from disease is welcome.*

Resumo. *O COVID-19 é uma doença infecciosa causada por um vírus. O coronavírus pode ser transmitido pela boca ou nariz de uma pessoa infectada, por este motivo o uso da máscara facial é obrigatório em período pandêmico. Diante disso, este artigo tem como finalidade apresenta uma metodologia capaz de verificar se um indivíduo está ou não utilizando uma máscara de proteção facial. O modelo desenvolvido utiliza técnicas de visão' computacional implementadas através da biblioteca OpenCV em conjunto com um modelo de Deep Learning. Os testes realizados exploraram ângulos e formas diferentes no uso da máscara e em alguns casos obtiveram taxas de detecção de até 100%. Em tempos de pandemia, todo o cuidado é necessário e toda a tecnologia que ajuda a prevenir a propagação de doenças é bem-vinda na sociedade.*

1. Introdução

Desde 2019, com a chegada da doença COVID-19 ao mundo, todas as pessoas foram obrigadas a utilizar máscaras de proteção facial para sua devida segurança visando diminuir a disseminação do vírus que é facilmente transmitido pela boca ou nariz, e se espalha mais facilmente em lugares fechados ou em multidões, de acordo com a BVSMS (Biblioteca Virtual em Saúde Ministério Saúde). Apesar de anos vivendo em pandemia, poucos estabelecimentos no Brasil, principalmente os pequenos comércios como: padarias, mini mercados, farmácias etc., não possuem uma verificação se o indivíduo está ou não utilizando máscara ao adentrar no local e quando possuem, geralmente são funcionários que precisam estar de prontidão nas entradas.

Atualmente existem tecnologias relacionadas a visão computacional que podem ajudar na solução desse problema. Um exemplo é a empresa Tryolabs que desenvolveram alguns *softwares* com essa funcionalidade, segundo Wudan Yan (2021). O grande gargalo se dá por essas aplicações serem disponibilizados de maneira comercial, ou seja, o estabelecimento que desejar adquirir tais soluções teriam que desembolsar uma quantia elevada de dinheiro, muitas vezes incompatíveis com os recursos disponíveis, para utilizar o produto. Dito isso, com o obtivo de contribuir com pequenos estabelecimentos, este trabalho apresenta uma metodologia para detecção do uso de máscara facial por câmeras de monitoramento, podendo servir como base para a implementação de produtos de softwares como solução de baixo custo e acessível à pequenos estabelecimentos.

Para o desenvolvimento do modelo de detecção foi necessário o estudo de subáreas da Inteligência Artificial, como visão computacional, definida por Szeliski (2010) como técnicas matemáticas para recuperar a forma tridimensional da imagem e a aparência de objetos e apresentá-las em formato digital. Aplicações utilizando visão computacional são encontradas em diversas áreas como robótica, medicina, física, biologia, indústria entre outras. A robótica é um bom exemplo de aplicação da visão computacional, muitos robôs utilizam reconhecimento de objetos através de câmera para executar diversas tarefas ou até mesmo o reconhecimento de pessoas através do reconhecimento facial. Segundo Prasanna e Reddy (2017) o reconhecimento facial é utilizado para reconhecer uma pessoa usando algumas características do rosto e se for utilizada uma base de dados bem treinada, uma aplicação é capaz de reconhecer todas as partes individuais (nariz, boca, olhos etc.).

Diversas tarefas compõem a metodologia proposta neste trabalho, juntas elas definem o fluxo das informações processadas desde da aquisição das imagens até a identificação do uso da máscara facial. Na fase de identificação duas etapas são propostas afim de atingir um melhor desempenho da metodologia, a primeira verifica se existe um rosto presente na imagem, caso exista, a segunda etapa é habilitada para identificar o uso da máscara facial. Após realizar essa verificação sinalizações de alerta (visual ou sonora) serão emitidas caso não seja detectado o uso da mesma.

Na fase de identificação, definido como identificação em duas etapas, a biblioteca OpenCV é responsável não só pela primeira etapa, mas também por todo processo de aquisição e processamento das imagens. Já na segunda etapa técnicas de *deep learning* (aprendizado profundo) foram utilizadas, mais especificamente uma arquitetura de rede neural convolucional chamada MobileNetV2 disponibilizada pela biblioteca de código aberto TensorFlow.

Para avaliação da metodologia, um formulário contendo instruções de uso, perguntas sobre o desempenho e ambiente no qual foi testado foi disponibilizado para usuários. Os resultados apresentados variaram as taxas de detecção entre 0% à 100%, de acordo com a dificuldade do teste.

A classificação da pesquisa quanto aos seus objetivos, se divide em três grandes grupos: exploratórias, descritivas e explicativas (KIPNIS, 2005). Nesse estudo foi utilizado a pesquisa exploratória, pois a pesquisa teve início a partir de um fenômeno que nesse caso foi a necessidade de maior segurança em tempos de pandemia e a partir de pesquisas sobre tecnologias de visão computacional foi desenvolvido uma solução.

Neste artigo, a seção 2 apresenta os principais conceitos e tecnologias relacionadas e abordadas neste trabalho. Na seção 3 são explorados trabalhos que compartilham das mesmas tecnologias empregadas nesse artigo, assim como um trabalho desenvolvido de maneira diferente, mas que possui a mesma finalidade. Na seção 4 é apresentada a arquitetura proposta desenvolvida para a solução, bem como o processo de desenvolvimento de cada módulo e detalhes das tecnologias utilizadas. Na seção 5 é apresentado os métodos para a medir a eficácia do trabalho e quais foram os resultados obtidos. Por fim, as seções 6 e 7, são responsáveis por abordar a conclusão desse trabalho e as possíveis melhorias futuras.

2. Referencial Teórico

A seguir será apresentado os principais conceitos que foram utilizados para o desenvolvimento desse trabalho.

2.1. Visão Computacional

Segundo Szeliski (2010) uma aplicação utilizando visão computacional pode ser definido como a máquina tentando criar uma retratação do mundo real, algo que é bem difícil, pois diferente dos nossos olhos, um computador opera via sistema numéricos, na maioria das vezes sendo eles binários. Para que o software consiga reconhecer uma imagem, precisa ocorrer um processamento na mesma e depois apresentar padrões que podem ser compreendidos.

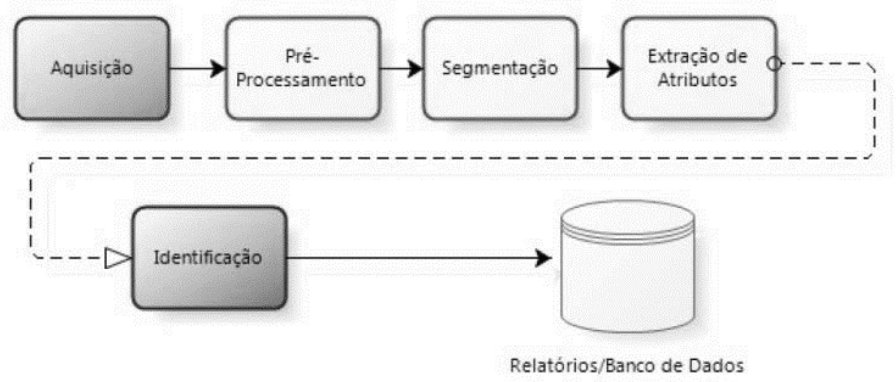


Figura 1. Etapas de um sistema de visão computacional genérico.

Na Figura 1 é possível visualizar as etapas que uma imagem passa até chegar ao nível de entendimento da máquina.

- **Aquisição:** etapa que tem como objetivo captar as imagens, podendo ser realizada por câmeras fotográficas, câmeras de celular, webcam, etc.
- **Pré-Processamento:** agora que a imagem já foi adquirida ela vai passar por uma séria de processamentos com a finalidade de otimizar a mesma. Esses processamentos geralmente são: redução de ruído, melhoria de contraste, exposição de cores, etc.
- **Segmentação:** isola regiões de interesse, no caso desse trabalho o objetivo é isolar o rosto do indivíduo do resto da imagem.
- **Extração de Atributos:** essa etapa tem como principal objetivo criar padrões que serão facilmente acessados para se referir as regiões de interesse particionadas na etapa anterior.
- **Identificação:** após ter os padrões de identificação definidos nas etapas acima, está última vai relacionar com outras imagens, ou seja, um banco de dados. Por exemplo: o objetivo dessa aplicação é separar um rosto de toda a imagem e depois validar se nesse rosto aparece ou não uma máscara de proteção facial.

2.2. Redes Neurais

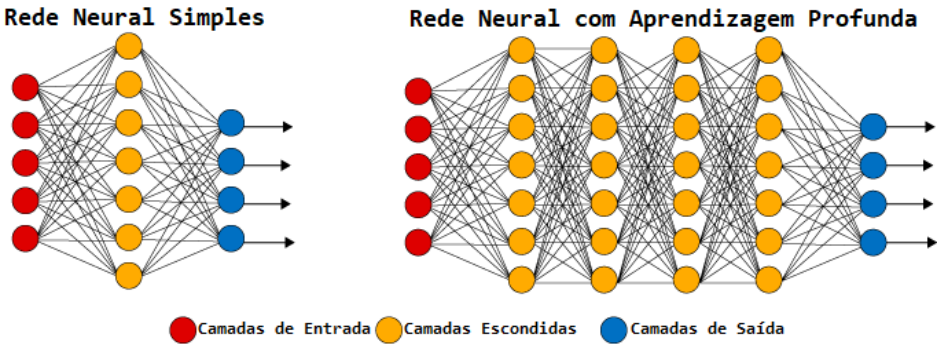
Segundo (RIBEIRO e GUIMARÃES, 2018) a criação de modelos aprendizado de máquina (*machine learning*) tem como principal objetivo desenvolver padrões de dados reais em grande escala que podem servir para predição ou classificação. Esses padrões são definidos por meio de um treinamento, onde geralmente partirá de uma base de dados, seja ela composta por imagens, caracteres (referentes a algum tema) ou qualquer tipo de informação que pode ser interpretada pela máquina. Após o processamento será gerado padrões numéricos, criando então uma rede neural afim de solucionar problemas reais.

Uma rede neural tem seu funcionamento inspirado nos neurônios do ser humano, que são capazes de adquirir conhecimentos através de experiências. Partindo para o conceito da rede neural artificial, seu funcionamento é explicado através de camadas responsáveis por trabalhos

específicos, sendo elas: camada de entrada, lugar onde os dados são apresentados a rede; camada escondida ou intermediária, é responsável por realizar o processamento, processo onde os dados recebidos receberão pesos referentes ao objetivo final; e por fim, a camada de saída, onde o resultado final é concluído e apresentado (*Deep Learning Book*, 2021).

Alguns anos após a criação de redes neurais simples, foi desenvolvida uma técnica nomeada de *deep learning* (aprendizado profundo), que utiliza camadas de neurônios matemáticos para processamento de dados, compreender a fala humana e reconhecer objetos visualmente. O diferencial da aprendizagem profunda é que ela utiliza diversas camadas ocultas contendo um tipo de função de ativação diferente (*Deep Learning Book*, 2021). Na figura 2 pode-se observar a diferença entre uma rede neural simples e uma rede neural que utiliza a técnica de aprendizagem profunda.

Figura 2. Rede neural simples e rede neural com aprendizagem profunda.



Fonte: *Deep Learning Book*, 2021.

Atualmente existem diferentes arquiteturas para treinamento de uma rede neural. O que diferencia uma arquitetura de outra é a quantidade de neurônios e como será feita a distribuição dos pesos na hora do treinamento. Para a finalidade desse projeto, no treinamento será utilizada uma arquitetura de rede neural convolucional (CNN), estruturada para criação de redes neurais que utilizam imagens como entrada. Segundo (SANDLER et al, 2018) a CNN torna o processamento de imagens computacionalmente gerenciável através da filtragem das conexões por proximidade. Em uma determinada camada, em vez de ligar cada entrada a cada neurônio, as CNNs restringem as conexões intencionalmente para que qualquer neurônio aceite as entradas apenas a partir de uma pequena subseção da camada anterior (como por exemplo 5x5 px ou 3x3 px). Portanto, cada neurônio é responsável por processar apenas uma certa porção da imagem.

Visando a problemática a ser desenvolvida nesse trabalho, será utilizado o MobileNetV2 como arquitetura de treinamento da rede neural. Os dados de entrada serão compostos por imagens de indivíduos utilizando máscaras de proteção facial e indivíduos que não estão utilizando máscara, gerando duas diferentes entradas de dados e sequentemente disponibilizando duas saídas, capazes de distinguir se existe, ou não, uma máscara na face da pessoa.

2.2.1. MobileNetV2

MobileNetV2 é uma arquitetura de *deep learning* convolucional que busca um bom desempenho principalmente em dispositivos móveis. Essa arquitetura foi desenvolvida pelos criadores da biblioteca TensorFlow, um pacote de código aberto utilizado para aprendizado de máquina, sendo aplicável a uma ampla variedade de tarefas, sejam elas predições, qualificações ou detecções de padrões (SANDLER et al, 2018).

Essa arquitetura possuí uma camada de expansão intermediária que usa convoluções profundas leves para filtrar recursos como uma fonte de não linearidade. Como um todo, a arquitetura do MobileNetV2 contém a camada inicial totalmente de convolução com 32 filtros (SANDLER et al, 2018). Esses filtros tem como finalidade extrair os traços mais marcantes da imagem.

A partir dessa arquitetura será realizado o treinamento da rede neural, com uma base de dados de duas entradas, sendo elas: imagens de rostos com máscara e imagens de rostos sem máscara. A partir do treinamento será gerada uma rede neural com duas saídas, responsável por resolver o problema de predição, ou seja, depois do rosto detectado, será realizada a predição, verificando a existência da máscara.

2.3. Reconhecimento Facial

De um modo geral, os métodos de reconhecimento facial funcionam comparando as características faciais selecionadas de uma determinada imagem com os rostos existentes num banco de dados (Orvalho, V., 2019). O sistema de reconhecimento facial tem menor exatidão se comparado com a tecnologia biométrica, mas ele ainda é muito utilizado por ser um sistema fácil de implementar.

2.3.1. OpenCV

Na elaboração desse projeto será utilizada a biblioteca OpenCV, originalmente desenvolvida pela Intel, em 2003, é uma biblioteca totalmente livre ao uso acadêmico e comercial, comumente utilizada para soluções de problemas na área de visão computacional.

Partindo para a problemática do trabalho, o OpenCV será utilizado com a finalidade de realizar o reconhecimento da face humana e para isso ele utiliza o método *Eigenfaces* de reconhecimento facial 2D para fazer a detecção da face. O reconhecimento facial 2D é baseado em técnicas onde a rede neural capaz de realizar o reconhecimento do rosto é treinada com imagens de faces frontais como pode ser observado na figura 3.

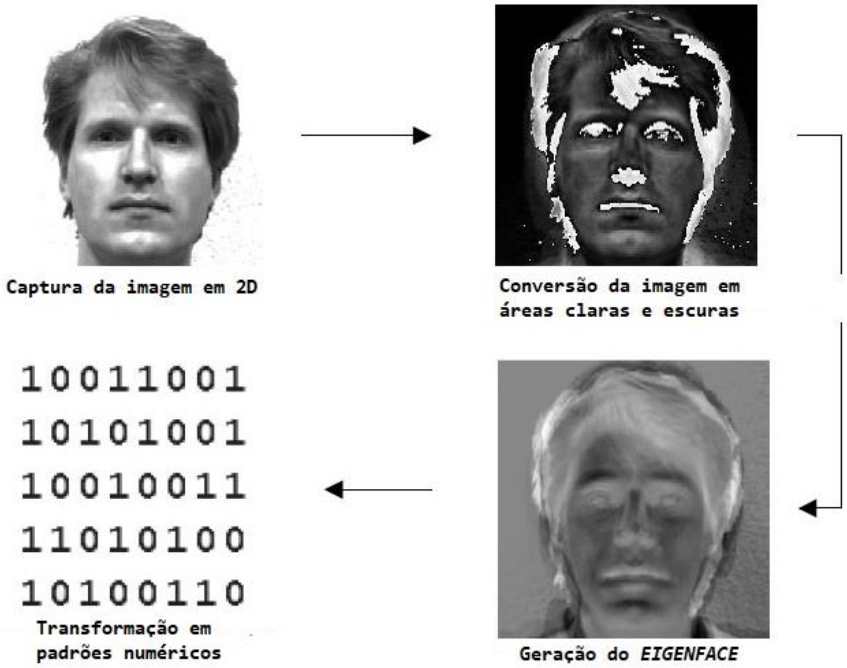
Figura 3. Imagem de face 2D.

Fonte: imagem retirada da internet, 2022.

Retornando ao conceito de *Eigenfaces*, ele é um conjunto de autovetores de uma matriz de covariância formada por imagens de rostos frontais em tons de cinza (SIROVISH e KIRBY, 1987). Em 1991, esse método foi implementado na biblioteca OpenCV para a classificação de

faces, os autores dessa implementação foram Matthew Turk e Alex Pentland, pesquisadores da IEEE. Para compreender o funcionamento dessa técnica pode-se separar o processo em 4 etapas: no primeiro momento a imagem 2D é capturada e pré-processada com o intuito de ter apenas dois canais de cores: o preto e o branco; na sequência essa imagem é convertida para uma saída onde serão destacadas as áreas claras e escuras da imagem; o terceiro passo é responsável pela geração do *eigenface*, por outras palavras, é realizar o destaque das áreas mais marcantes do rosto (boca, olhos, nariz e sobrancelha) retirando toda informação desnecessária da imagem; por fim essa imagem é transformada em padrões numéricos, gerando no final uma matriz de covariância. Pode-se observar o funcionamento desse processo através da Figura 4.

Figura 4. Processo de criação do *Eigenface*.



Fonte: *Face Recognition White Paper Technology* (FingerTec), 2000-2014.

3. Trabalhos Correlatos

Após uma revisão bibliográfica com os seguintes termos pesquisados: “reconhecimento facial”, “reconhecimento de faces através de *deep learning*” e “reconhecimento facial via OpenCV” de 2017 até os dias atuais foram encontrados alguns trabalhos que abordam esses temas e contribuíram de forma efetiva a está pesquisa.

O trabalho desenvolvido por Jean Carlos Corrêa Saffi (2018), apresenta uma aplicação utilizando visão computacional, que tem a finalidade de validar rostos para ambientes de acesso restrito. Durante o desenvolvimento desse projeto o autor apresenta técnicas de reconhecimento facial utilizando o OpenCV. Com a leitura desse artigo foram adquiridos muitos conceitos técnicos e teóricos da implementação do OpenCV e como funciona uma aplicação de reconhecimento facial.

Ribeiro e Guimarães (2018), desenvolveram um trabalho referente a redes neurais, utilizando o TensorFlow como base do trabalho. Muitos conhecimentos sobre redes neurais e técnicas de abordagem desse conceito foram extraídos com a leitura desse trabalho.

Já Yu e Wei (2021), apresentaram um trabalho sobre detecção de máscara utilizando o algoritmo de detecção de objetos YOLOv4. Apesar de serem técnicas diferentes, pois o treinamento da rede neural desse trabalho foi realizado com o MobileNetV2, esse artigo

agregou conhecimentos de forma efetiva. Além da arquitetura, outro diferencial é que os autores realizaram o reconhecimento da máscara em um processo único, ou seja, a mesma rede capaz de validar a existência da máscara realizava a detecção da face. Por fim, eles optaram por não apresentar nenhuma interface de usuário, ao contrário desse projeto que será apresentado uma demonstração de interface de usuário para detecção da máscara em tempo real, através de uma webcam.

4. Arquitetura Proposta

A arquitetura proposta diz respeito à disposição das etapas que compõem a metodologia apresentada nesse trabalho de forma que demonstre o fluxo das informações processadas desde a aquisição dos dados até a identificação do uso da máscara facial. De acordo com a problemática desse trabalho, a solução desenvolvida foi separada em três etapas em que cada uma delas possuem tecnologias específicas que ao final, funcionando em conjunto, apresentam uma solução funcional. A Figura 5 apresenta todo o escopo de desenvolvimento.

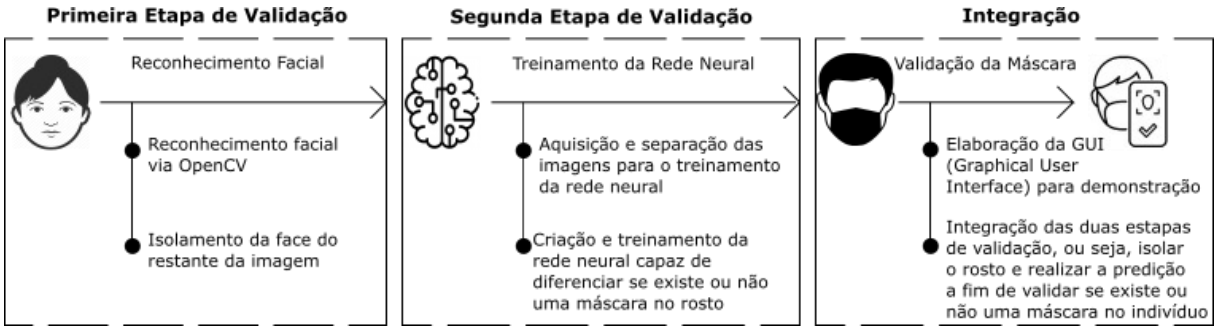


Figura 5. Fluxo de funcionamento e tecnologias empregadas no trabalho.

O desenvolvimento da primeira etapa consiste em realizar o reconhecimento da face humana utilizando o OpenCV. Através dessa biblioteca é possível que um rosto seja detectado a partir de uma imagem de entrada e o mesmo seja isolado do restante da imagem, diminuindo a área para a detecção da máscara facial no futuro.

Na sequência será feito a segunda etapa de validação, ou seja, a rede neural prevê se existe uma máscara na face ou não. A primeira parte desse processo é ter um banco de imagens separadas em: pessoas com e sem máscaras. As imagens para o treinamento da rede neural foram disponibilizadas por uma empresa da região de Campinas, que possui um departamento de câmera, no qual realizam a capturas de muitas fotos em seu dia-a-dia. Apesar da disponibilização das imagens a empresa preferiu manter-se anônima e permitiu o uso das imagens apenas para o treinamento da rede, não podendo ser disponibilizadas para terceiros. Após a separação do *dataset*, esses dados são ampliados é realizado o treinamento da RNA.

Na última etapa, é realizado a integração dos dois processos anteriores, ou seja, a rede neural já treinada detecta a máscara no rosto isolado a partir da primeira etapa de desenvolvimento. Essa validação da máscara só começa a ser feita quando um rosto é detectado na imagem, caso um rosto seja encontrado, é atribuída uma porcentagem de predição negativa quando nenhuma máscara for encontrada ou positiva se existir a presença da máscara facial, vale ressaltar que se nenhum rosto for detectado, essa predição não ocorrerá.

4.1. Reconhecimento Facial

Para a primeira etapa de verificação, será abordada a técnica de reconhecimento facial, utilizando o OpenCV. Em conjunto com essa biblioteca, foi utilizado o arquivo *open source* para aplicações sem fins lucrativos: “haarcascade_frontalface_alt2.xml” desenvolvido e

treinado pela Intel, essa rede neural é utilizada em conjunto a biblioteca OpenCV para realizar o reconhecimento de faces humanas.

O processo de reconhecimento é dividido em algumas etapas, sendo elas:

- Ler a imagem utilizando o método "*detectMultiScale*", no caso desse projeto a imagem é coletada em tempo real, através de uma webcam. Esse processo também pode ser identificado como: etapa de aquisição;
- Na sequência é realizado o processo de pré-processamento. A imagem é transformada em tons de cinza, isso é feito pois as imagens se tornam mais fáceis de serem processadas, pois contém apenas um canal de preto e branco;
- Após a imagem ser segmentada e processada, o método retorna quatro valores, sendo eles: coordenada (x), coordenada (y), largura (w) e altura (h). Esses valores são referentes a posição do rosto na imagem. Essa penúltima etapa é conhecida como extração dos atributos, ou seja, transforma a imagem em dados numéricos;
- Na última etapa ocorre o processo de identificação com base nos dados numéricos. Caso um rosto seja detectado na imagem adquirida esse método retorna a posição do rosto encontrado, o mesmo é enquadrado, redimensionado para 224x224 e isolado do restante do frame.

4.2. Treinamento da Rede Neural

Nessa etapa do desenvolvimento é treinada uma rede neural capaz de diferenciar um rosto com máscara de um rosto sem máscara. Para o treinamento é utilizada a técnica de aprendizado supervisionado. Essa técnica consiste em aprender uma função que mapeia uma entrada para uma saída a partir de dados de treinamento rotulados, ou seja, dados separados e especificados antes de iniciar o treinamento (*Deep Learning Book*, 2021).

Para que essa rede neural seja desenvolvida, é necessário realizar o treinamento utilizando um *dataset* com imagens de rostos com máscara e rostos sem máscara. Nessa etapa também é utilizado a arquitetura de aprendizagem profunda MobileNetV2. Essa arquitetura é adaptada ao propósito desse trabalho, servindo como base para a criação do modelo a fim de fazer o reconhecimento do uso de máscara.

Após a aquisição das imagens, elas são separadas em duas pastas "com_mascara" e "sem_mascara". Com as imagens já separadas, é iniciado o treinamento da rede, onde no primeiro momento, é importado todas as bibliotecas que serão utilizadas ao longo do processo (Figura 6).

```
#Bibliotecas#
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
```

Figura 6. Bibliotecas para treinamento da rede neural.

O próximo passo é ler todas as imagens que foram coletadas e atribuí-las a alguma lista (Figura 7). Aqui, é obtido todos os caminhos associados a essas imagens e os rótulos de acordo (faces com e sem máscara). Além disso, as imagens são pré-processadas e redimensionadas para as dimensões de 224x224. Esse redimensionamento acontece, pois, a saída do frame do OpenCV é realizada nessas dimensões.

```
# Diretório das imagens
imagePaths = list(paths.list_images(r'C:\Users\Pichau\Desktop\TCC\TCC_Dev\imagensTreinarNN'))
data = []
labels = []
# Percorrer todas as imagens
for imagePath in imagePaths:
    # rotular as imagens
    label = imagePath.split(os.path.sep)[-2]
    # Carregar as imagens no formato 224x224 e pré-processar
    image = load_img(imagePath, target_size=(224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)
    # Atualizar os dados pré-processados e as labels
    data.append(image)
    labels.append(label)
# Converter os dados e as labels em array
data = np.array(data, dtype="float32")
labels = np.array(labels)
```

Figura 7. Leitura das imagens e pré-processamento.

O próximo passo é carregar o modelo MobileNetV2 e customizá-lo de acordo com a problemática do trabalho (Figura 8). Portanto, serão removidas algumas camadas superiores

deste modelo pré-treinado e será adicionado algumas camadas próprias, as camadas customizadas serão armazenadas no *headModel*. Após essa etapa, será criado o modelo final utilizando o MobileNetV2 base como entrada e o *headModel* como saída. Por fim, todas as camadas do modelo base serão bloqueadas para que não ocorra alterações durante o processo de treinamento.

```
# Carregar o modelo pré treinado (MobileNetV2) e customiza-lo
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_shape=(224, 224, 3))
# Construir a saída do modelo, ou seja, onde ficará as customizações da rede
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
# Criar o modelo real, utilizando a base do MobileNetV2 como entrada e a saída onde foi implementada as alterações
model = Model(inputs=baseModel.input, outputs=headModel)
# Percorrer as camadas no modelo base e congelá-las para que não sejam atualizadas durante o processo de treinamento
for layer in baseModel.layers:
    layer.trainable = False
```

Figura 8. Customização do modelo MobileNetV2.

Próximo passo é converter as *labels* (os rótulos criados anteriormente no pré-processamento das imagens) em *One-Hot-Encoding*, ou seja, gerar vetores binários para todos os valores extraídos no pré-processamento (Figura 9). Depois, os dados são divididos em conjuntos de treinamento e teste. Para divisão dos dados será utilizado o princípio de Pareto, uma regra que afirma que 80% dos efeitos vêm de 20% das causas, ou seja, 80% dos dados serão destinadas ao treinamento da rede neural e os 20% restantes serão usados para testes.

Além disso, a próxima etapa ocorre o processo de aumento de dados, que aumenta significativamente a diversidade de dados disponíveis para modelos de treinamento, sem realmente coletar novos dados. Essa técnica de aumento de dados é usada para treinar grandes redes neurais, ela aplica algumas particularidades a cada imagem como: recorte, rotação, zoom, inversão horizontal e inversão vertical.

```
# Converter os labels em One-Hot-Encoding (gerar vetores binários para cada valor)
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
# Particionar os dados em divisões de treinamento e teste usando 80% dos dados para treinamento e o restante para testes
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)
# Construir o gerador de imagem, para ampliação dos dados
aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")
```

Figura 9. Gerar vetores binários (*One-Hot-Enconding*), particionar dados e ampliá-los.

Após todas as etapas anteriores concluídas o modelo é compilado e treinado utilizando os dados aumentados da etapa anterior (Figura 10). Para realizar o treinamento é utilizado uma taxa de aprendizado de 1e-4 (0,0001), recomendada por Tanksale (2018). Na etapa de treinamento são utilizadas vinte épocas como parâmetro, ou seja, o algoritmo realizará a leitura

do conjunto de dados por vinte vezes. Apesar do treinamento começar a estabilizar a partir de dezesseis épocas, é utilizado vinte para ter uma maior exatidão.

Após o treinamento, o modelo será salvo em um arquivo ".h5", que pode ser interpretado pelo TensorFlow e juntamente ao Keras (interface de redes neurais da linguagem de programação Python, se destaca por trabalhar em cima do TensorFlow e por permitir uma experimentação rápida na utilização de redes neurais profundas) realiza a detecção se existe uma máscara de proteção facial no rosto ou não.

```
# Compilar o modelo e treinar nos dados aumentados
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
print("Compilando...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
# Treinar a rede
print("Treinando...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)

#Salvar o modelo
model.save('detectarMascara.h5')
```

Figura 10. Compilar, treinar e salvar a rede pronta para ser utilizada.

O treinamento foi realizado em um notebook com a seguinte configuração: processador Intel i5; vídeo integrado; 16 *gigabyte* de memória RAM e 1 *terabyte* de disco rígido. E o tempo de treinamento ficou na casa dos trinta minutos.

4.3. Validação da Máscara

Essa etapa do desenvolvimento é a junção das tecnologias apresentadas até o presente momento, formando então a identificação em duas etapas. Após o modelo de reconhecimento de máscara pronto, o processo de identificação da máscara é executado. O modelo já treinado é carregado a partir dos pacotes mencionados, se tornando apto a realizar predições. É utilizada uma estrutura de decisão na qual a predição da rede neural só acontece caso um rosto seja detectado na webcam. Após o rosto ser detectado e os valores de posição serem retornados pela função "*detectMultiFaces*" o modelo faz a predição nas coordenadas informadas pelo método do OpenCV e retorna a porcentagem se existe ou não uma máscara na face.

Caso a predição aponte que existe uma máscara no rosto, o mesmo é enquadrado com bordas verdes e uma mensagem de "MÁSCARA" é apresentada ao usuário. Caso essa predição seja o inverso, as bordas ficarão vermelhas e a mensagem de "SEM MÁSCARA" é apresentada.

4.4. Interface Gráfica de Usuário para Demonstração

A interface gráfica para validação da solução foi desenvolvida a partir da biblioteca PySimpleGUI. Esse pacote Python utiliza a estrutura de GUI (*Graphical User Interface*) do Tkinter, porém o desenvolvimento é mais simples e apresenta resultados tão bons quanto.

Na Figura 11 pode-se observar a aplicação em execução, nessa demonstração o software realiza a detecção da máscara no usuário e retorna as informações de máscara encontrada (mensagem "Mascara" na cor verde) e a porcentagem de reconhecimento da máscara, nesse caso em específico um acerto de 99,94%.

Figura 11. Demonstração da aplicação em funcionamento.

O layout foi desenvolvido de maneira que o usuário possa selecionar o modo de operação da aplicação, podendo escolher entre:

- *None*: será apresentado somente a imagem da webcam.
- Detecção de Face: a aplicação detecta a face do usuário e a enquadra.
- Detecção de Máscara: a aplicação detecta a face do usuário e aplica a verificação se existe uma máscara ou não. Caso uma máscara seja detectada, o rosto é enquadrado com bordas verdes, apresenta a mensagem "Máscara" e uma porcentagem de reconhecimento. Caso a máscara não seja detectada, o rosto é enquadrado com bordas vermelhas, apresentar a mensagem "Sem Máscara" e a porcentagem de não reconhecimento.
- Detecção de Máscara c/ Mecanismo de Trava: realiza todos os passos da condição de Detecção de Máscara, incluindo uma mensagem de "Bloqueado" com bordas vermelhas quando a máscara não for detectada e uma mensagem de "Desbloqueado" com bordas verdes caso a máscara seja encontrada.

5. Resultados

Com o objetivo de avaliar o correto funcionamento da solução, foi desenvolvido um formulário contendo perguntas técnicas do funcionamento e do ambiente no qual foi testado. Esse formulário foi disponibilizado para alguns usuários, capazes de executar uma aplicação em

Python, no total nove pessoas participaram respondendo o formulário e realizando os testes propostos.

Foram utilizadas diversas métricas de avaliação, sendo elas: resolução da webcam, quais diferentes tons de cor de máscara foram utilizados, diferentes ângulos para captar o rosto, testes com máscara e sem máscara e por qual plataforma o usuário executou a aplicação (VS Code, terminal windows/linux, PyCharm etc.). A figura 12 ilustra todos os ângulos requisitados para os usuários testarem. Com base nesse gabarito, as pessoas responderam perguntas sobre o quão bem a solução funcionou nesses ângulos específicos.

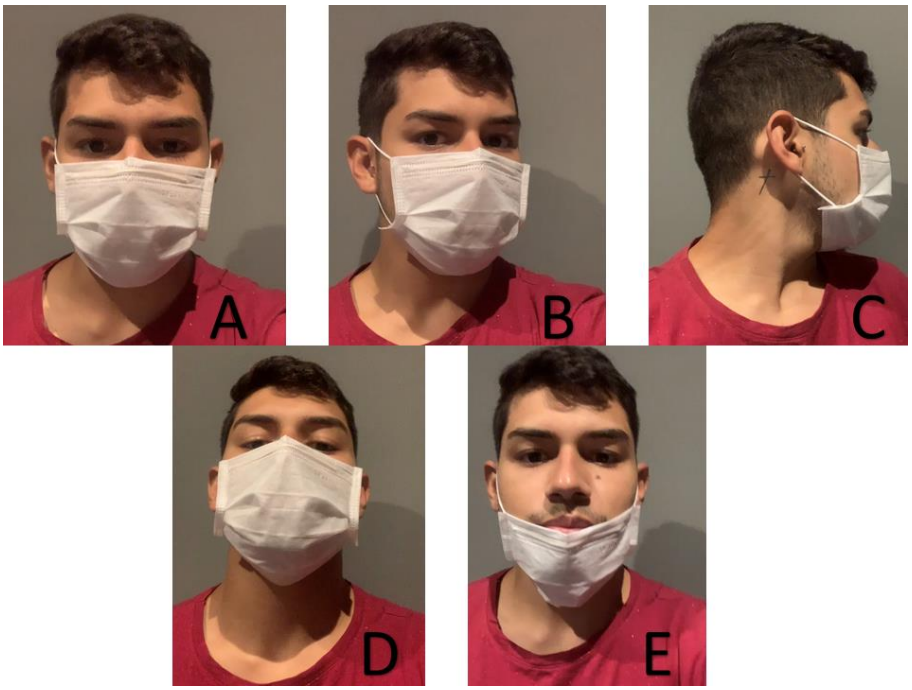


Figura 12. Gabarito para testes em diferentes ângulos de funcionamento.

No primeiro teste o usuário precisava estar utilizando máscara e olhando fixamente para a câmera, como mostra a Figura 12-A. Com o resultado do gráfico da Figura 13 pode-se chegar à conclusão que a aplicação é cem por cento funcional para validações quando o usuário está olhando fixamente e de forma frontal para a câmera.

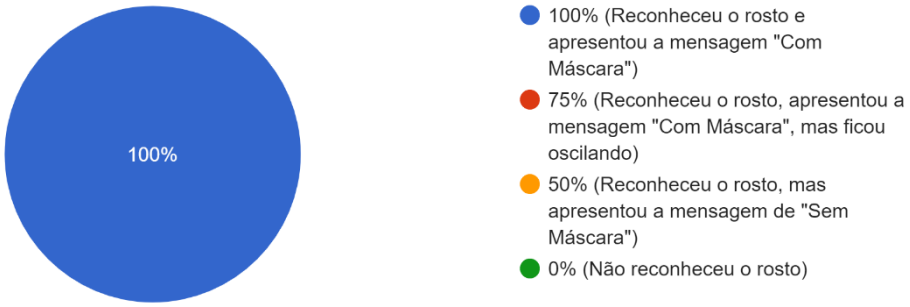


Figura 13. Funcionamento usando máscara e olhando fixamente para a câmera.

Na segunda validação o indivíduo necessitava usar a máscara e inclinar levemente a cabeça para a direita ou esquerda (Figura 12-B). Apesar de acontecer alguma oscilação do software, ele se comportou bem para validações quando o usuário está com a cabeça levemente inclinada horizontalmente (Figura 14).

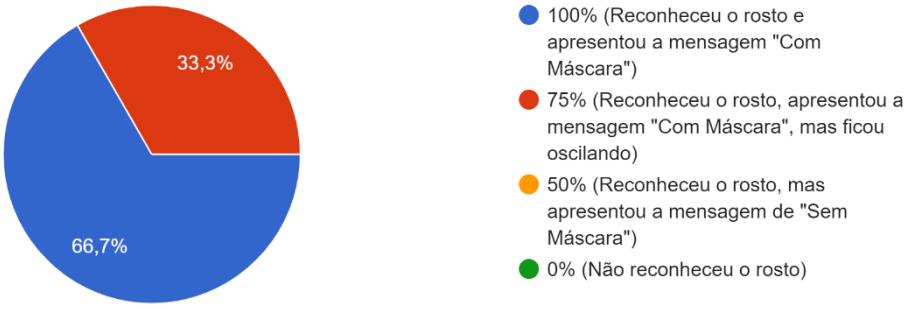


Figura 14. Funcionamento usando máscara e cabeça inclinada horizontalmente a 45º.

Para a terceira etapa de validação era necessário usar a máscara e virar totalmente a cabeça para esquerda ou direita, em um ângulo de 90% graus (Figura 12-C). Quando o usuário mantém a cabeça totalmente inclinada horizontalmente o software não consegue reconhecer o rosto do usuário e por isso não ocorre a detecção da máscara (Figura 15).

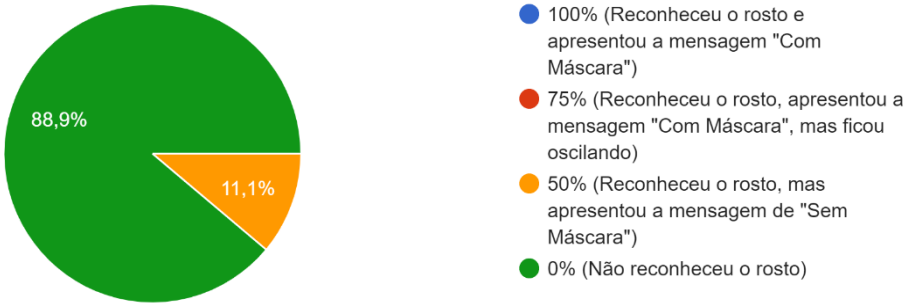


Figura 15. Funcionamento usando máscara e cabeça inclinada horizontalmente a 90º.

O penúltimo teste consistia em inclinar a cabeça verticalmente usando a máscara no rosto (Figura 12-D). Nessa validação ocorre o mesmo problema do teste citado anteriormente, dependendo da inclinação do rosto o software não o reconhece e então acontece o reconhecimento se existe uma máscara na imagem (Figura 16).

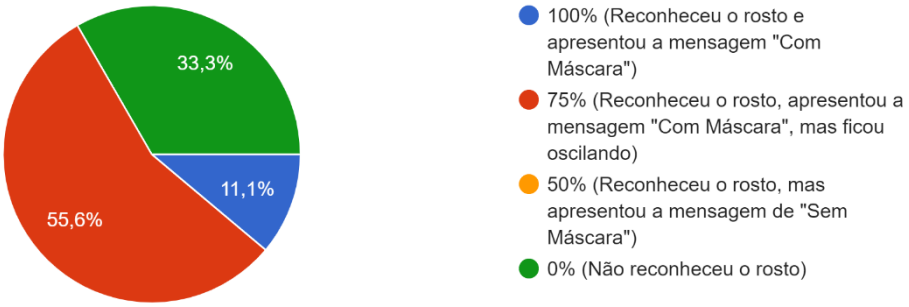


Figura 16. Funcionamento usando máscara e cabeça levemente inclinada verticalmente.

No último teste era necessário que o usuário colocasse a máscara a baixo da boca e olhasse fixamente para a webcam (Figura 12-D). A intenção nesse caso era que a aplicação apresentasse a mensagem “Sem Máscara” ao usuário, pois a mesma está sendo utilizada de forma errada. Como pode ser observado na Figura 17, a grande maioria relatou que a solução apresentou a mensagem “Sem Máscara”, o que era esperado nesse caso, já que a solução não possui uma verificação se a máscara está sendo utilizada de maneira correta.

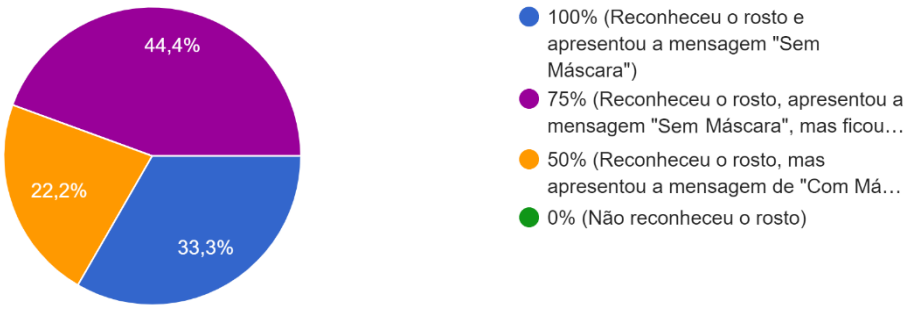


Figura 17. Funcionamento usando máscara a baixo da boca.

A aplicação apresentou ótimo desempenho para validações quando o usuário está olhando de forma frontal ou com uma leve inclinação em relação a webcam. Apesar de apresentar algumas instabilidades nas outras formas de validação como um rosto inclinado, isso é algo que pode ser aprimorado em trabalhos futuros utilizando um sistema de reconhecimento facial 3D, ou seja, uma detecção com maior exatidão em diferentes ângulos.

6. Conclusão

O objetivo de desenvolver uma solução que seja capaz de validar se existe uma máscara de proteção facial em um rosto humano foi cumprido. O modelo de rede neural em conjunto a interface gráfica para demonstração, obtiveram êxito em testes realizados com usuários, tanto em validações realizadas de forma frontal quanto em validações com a cabeça inclinada horizontalmente a um ângulo de 45º.

Para um projeto de cunho experimental e que pode servir de base para futuros projetos na área, esse trabalho está apto a abrir portas para futuros aprimoramentos e ser colocado em situações reais do cotidiano.

Como Trabalho de Conclusão de Curso do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, foram utilizados conhecimentos adquiridos em muitas disciplinas, vale destaque para as seguintes: Linguagem de Programação, apesar de não utilizar nenhuma linguagem estudada durante o curso a disciplina serviu de apoio principalmente com o conhecimento lógico; Engenharia de Software e Arquitetura de Software, fundamentais para toda a base do projeto, desde a escolha do modelo de desenvolvimento até a estipulação de prazos; Interface Humano Máquina, servindo como alicerce para todo o desenvolvimento das interfaces gráficas; entre outras disciplinas que contribuíram tanto para o desenvolvimento desse projeto como para o meu desenvolvimento pessoal e profissional.

7. Trabalhos Futuros

Para trabalhos a serem desenvolvidos futuramente sugere-se a ampliação do *dataset* e o treinamento da rede utilizando esses novos conjuntos de dados. Essa ampliação dos dados visa deixar a solução capaz de reconhecer se uma máscara está sendo utilizada de maneira correta no rosto, além de verificar se a mesma está presente, isso tornará a aplicação mais robusta e estará propícia a ter um desempenho ainda melhor.

A segunda etapa de melhoria seria implementar um método de reconhecimento facial 3D, para um desempenho ainda melhor em variáveis ângulos de detecção do rosto.

O terceiro ponto que pode ser desenvolvido no futuro é transformar a solução em um aplicativo, capaz de ser implementado em ambientes reais, com um grande número de movimentação de pessoas. Sejam eles: comércios, escolas e/ou estabelecimentos governamentais.

Agradecimentos

Primeiramente agradeço a Deus por me proporcionar saúde para realização desse projeto, agradeço a minha família pelo apoio incondicional. Agradeço aos professores do Instituto Federal de São Paulo unidade Hortolândia que contribuíram para a minha formação, com ênfase a professora Daniela Marques que lecionou a matéria "Projetos de Sistemas" e contribuiu de forma efetiva e memorável para o desenvolvimento desse trabalho. Por fim, um agradecimento especial ao docente Carlos R. Santos Jr, professor orientador desse artigo, contribuindo de forma direta em todas etapas.

Referências

Corrêa Saffi, J. C. (2018) "Aprendizagem e reconhecimento de faces através de computer vision e machine learning, aplicado a ambientes de acesso restrito", https://www.imed.edu.br/Uploads/JEAN%20CARLOS%20MAFFI.pdf, julho 2021.

Data Science Academy (2021) "Deep Learning Book", https://www.deeplearningbook.com.br, dezembro 2021.

Orvalho, V. (2019) "Reconhecimento Facial", Rev. Ciência Elem., V7(4):073

Pisarevsky, V. (2013) "haarcascade_frontalface_alt2", https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt2.xml, setembro 2021.

Prassana, D. M. e Reddy, C. G. (2017) "Development of Real Time Face Recognition System Using OpenCV", https://www.irjet.net/archives/V4/i12/IRJET-V4I12151.pdf, julho 2021.

Resende, C. A. de P. e Pereira, M. R. (2015) "Visão computacional aplicada em reconhecimento facial na busca por pessoas desaparecidas", https://revistas.unibh.br/dcet/article/view/1661/932, julho 2021.

Ribeiro, M. e Guimarães, S. (2018) "Redes Neurais Utilizando TensorFlow e Keras", https://revistas.unifenas.br/index.php/RE3C/article/view/231, agosto 2021.

Sandler, Mark et al. (2018) "MobileNetV2: Inverted Residuals and Linear Bottlenecks", https://openaccess.thecvf.com/content_cvpr_2018/papers/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.pdf, setembro 2021.

Szeliski, R. (2010) Computer Vision: Algorithms and Applications, páginas 1-100

Tanksale, N. (2018) "Finding Good Learning Rate and The One Cycle Policy", https://towardsdatascience.com/finding-good-learning-rate-and-the-one-cycle-policy-7159fe1db5d6, setembro 2021.

Vargas, Ana et al. (2018) "Um estudo sobre Redes Neurais Convolucionais e sua aplicação em detecção de pedestres", http://gibis.unifesp.br/sibgrapi16/eproceedings/wuw/7.pdf, setembro 2021.

Yan, W. (2021) "Reconhecimento facial com máscara já é uma realidade, gostemos ou não", https://www.nationalgeographicbrasil.com/ciencia, setembro 2021.

Yu, J. e Wei, Z. (2021) "Face Mask Wearing Detection Algorithm Based on Improved YOLO-v4", https://doi.org/10.3390/s21093263, dezembro 2021.

Documento Digitalizado Público

Anexo I (Artigo) - Marcos Miguel Ferreira Gomes - HT3002039

Assunto: Anexo I (Artigo) - Marcos Miguel Ferreira Gomes - HT3002039
Assinado por: Carlos Junior
Tipo do Documento: Outro
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Carlos Roberto dos Santos Junior**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 01/02/2022 15:22:56.

Este documento foi armazenado no SUAP em 01/02/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 877185
Código de Autenticação: 18cbc1914a