LEANDRO DANIEL

DETECÇÃO DE ROSTOS COM USO DE MÁSCARA DE PROTEÇÃO PARA COVID-19

Mini-paper apresentado ao MBA Executivo de Business Analytics e Big Data, FGV, como requisito parcial para a disciplina Análise Econômica e Geração de Valor.

Professor: Bernardo Aflalo

SÃO PAULO

2020

SUMÁRIO

LISTA DE ILUSTRAÇÕES

RESUMO

Detecção de rostos com uso de máscara de proteção para COVID-19.

Objetivo: O presente trabalho tem como objetivo principal, desenvolver um modelo de detecção de indivíduos utilizando máscaras faciais, através de técnicas de Deep Learning, como ferramenta possível de ser implementada em sistemas de monitoramento preventivo ao alastramento da COVID-19 em locais e áreas de convívio público ou privado. **Material e Método:** A base de dados utilizada para este projeto é uma composição de bancos de imagens classificadas de forma binária entre faces com máscara e faces sem máscara. Ambas as fontes estão publicamente disponíveis. O framework utilizado será o train_test_split do scikitlearn. Foram utilizadas duas das arquiteturas mais usadas no campo de inteligência artificial ANNs (Artificial Neural Networks) e CNNs (Convolutional Artificial Neural Networks). **Resultados:** No processo de seleção das análises finalistas, foram construídas diferentes profundidades de redes até a obtenção de arquiteturas satisfatórias. Este foi um processo não exaustivo. Ambas as redes tiveram ativação final sigmoide, com função de perda Binary Cross-entropy e otimizador SGD. Para a ANN final foram usadas camadas Flatten, Dense (ReLU) e Dropout, obtendo-se um total de 34.080.769 parâmetros treináveis. **Conclusão**: A proposta do trabalho foi atendida com o desenvolvimento de um modelo, com técnicas de deep learning, capaz de realizar a detecção de pessoas utilizando máscaras faciais. O modelo também se mostrou viável para posterior implementação em sistemas de controle de entrada em espaços públicos.

Descritores: Deep Learning, TensorFlow, Keras, OpenCV, Scikit-learn

1 INTRODUÇÃO

1.1 CONTEXTO DO TRABALHO

Vivemos tempos singulares, onde nossa sociedade enfrenta o desafio de combate e prevenção contra o crescente alastramento da COVID-19 em uma escala global. Deste desafiador contexto, emergem efeitos colaterais inéditos na era moderna, como o distanciamento social, dentre diversas mudanças de hábitos. Com base nas evidências atuais, o vírus COVID-19 é transmitido entre pessoas através de contato próximo e gotículas. Calcula-se que uma pessoa com infecção o transmita para de duas a quatro pessoas.

A Organização Mundial de Saúde (OMS) recomenda o uso de máscaras combinadas com a correta higiene frequente das mãos como parte da chamada EPI (Equipamento de Proteção Individual, do inglês PPE, ou Personal Protective Equipment), dentre outras orientações e recomendações. A Organização Mundial de Saúde disponibiliza uma série de orientações e guias para o uso correto das máscaras, bem com seu descarte e combinação com procedimentos de lavagem das mãos, como uma das medidas possíveis de contenção do alastramento da COVID-19.

Recentemente o estado de São Paulo decretou o uso de máscaras de proteção facial como obrigatório, por tempo indeterminado. A medida, que entrou em vigor em 07/05/2020, estabelece o uso de máscaras em: espaços públicos, estabelecimentos que executem atividades essenciais, repartições públicas estaduais e transporte por aplicativo.

1.2 OBJETIVO

O presente trabalho tem como objetivo principal, desenvolver um modelo de detecção de indivíduos utilizando máscaras faciais, através de técnicas de Deep Learning, como ferramenta possível de ser implementada em sistemas de monitoramento preventivo ao alastramento da COVID-19 em locais e áreas de convívio público ou privado.

É possível desenvolver, com técnicas de deep learning, um modelo de detecção de pessoas utilizando máscaras faciais para posterior implementação em

sistemas de controle de entrada em espaços públicos, estabelecimentos que executem atividades essenciais, repartições públicas estaduais, transporte por aplicativo para um público alvo de consumidores, fornecedores, clientes, empregados, colaboradores, agentes públicos e prestadores de serviço?

2 NOTEBOOK COM O DESENVOLVIMENTO DO TRABALHO

Este trabalho está disponível em um *notebook Jupyter*, escrito em *Pyhton 3*, publicado no *GitHub* no endereço a seguir: https://github.com/ldaniel/Artificial-Intelligence-Applications/blob/master/notebooks/fgv_group/fgv-mba-analise-economica-e-geracao-de-valor.ipynb.

Como alternativa, este notebook também pode ser executado a partir da plataforma *Binder*, disponibilizado em: https://mybinder.org/v2/gh/ldaniel/Artificial-Intelligence-Applications/master.

3 MATERIAL E MÉTODO

3.1 BASE DE DADOS

Para este trabalho, foi utilizada uma base de dados de imagem pública e pseudo artificial, produzida por Prajna Bhandary, com 1.376 imagens das quais 690 imagens estão classificadas como faces com máscara e 686 para a classe complementar. O conceito de pseudo-artificialidade mencionado é uma liberdade poética dos autores para expressar que, apesar de serem imagens reais de pessoas, as máscaras foram artificialmente criadas e posicionadas, via algoritmo, para o específico propósito de treinamento de modelos como o modelo que se pretende criar.

FIGURA 1 - EXEMPLO DE CRIAÇÃO DA IMAGEM PSEUDO-ARTIFICIAL

3.2 ANÁLISE ESTATÍSTICA

As arquiteturas de modelos criadas para este trabalho serão treinadas com o mesmo conjunto de treino e teste, deste modo será garantida uma equivalência para comparação. A proporção definida foi de 75% para o treino e 25% para o teste.

Para garantirmos que as classes estejam equilibradas em ambas as amostras, estratificaremos pela coluna de classes. O framework utilizado será o ***train_test_split do scikitlearn***.

Dado o desbalanceamento entre as classes também será necessária a técnica do undersampling (de forma aleatória) da classe majoritária.

3.3 PROCESSO DE PREPARAÇÃO DAS IMAGENS

Em trabalhos de ciências de dados é muito importante utilizar um banco de dados robusto e grande o suficiente para habilitar a generalização dos modelos criados. Por essa razão, uma prática comum aplicada às imagens é o processo de data augmentation, e este será o processo usado para fazer o enriquecimento do banco de imagens.

Este processo, basicamente, toma as imagens originais a aplica pequenas modificações, tais como: rotação, translação, cisalhamento, ampliação e espelhamento a fim de gerar novas imagens para o treinamento dos modelos.

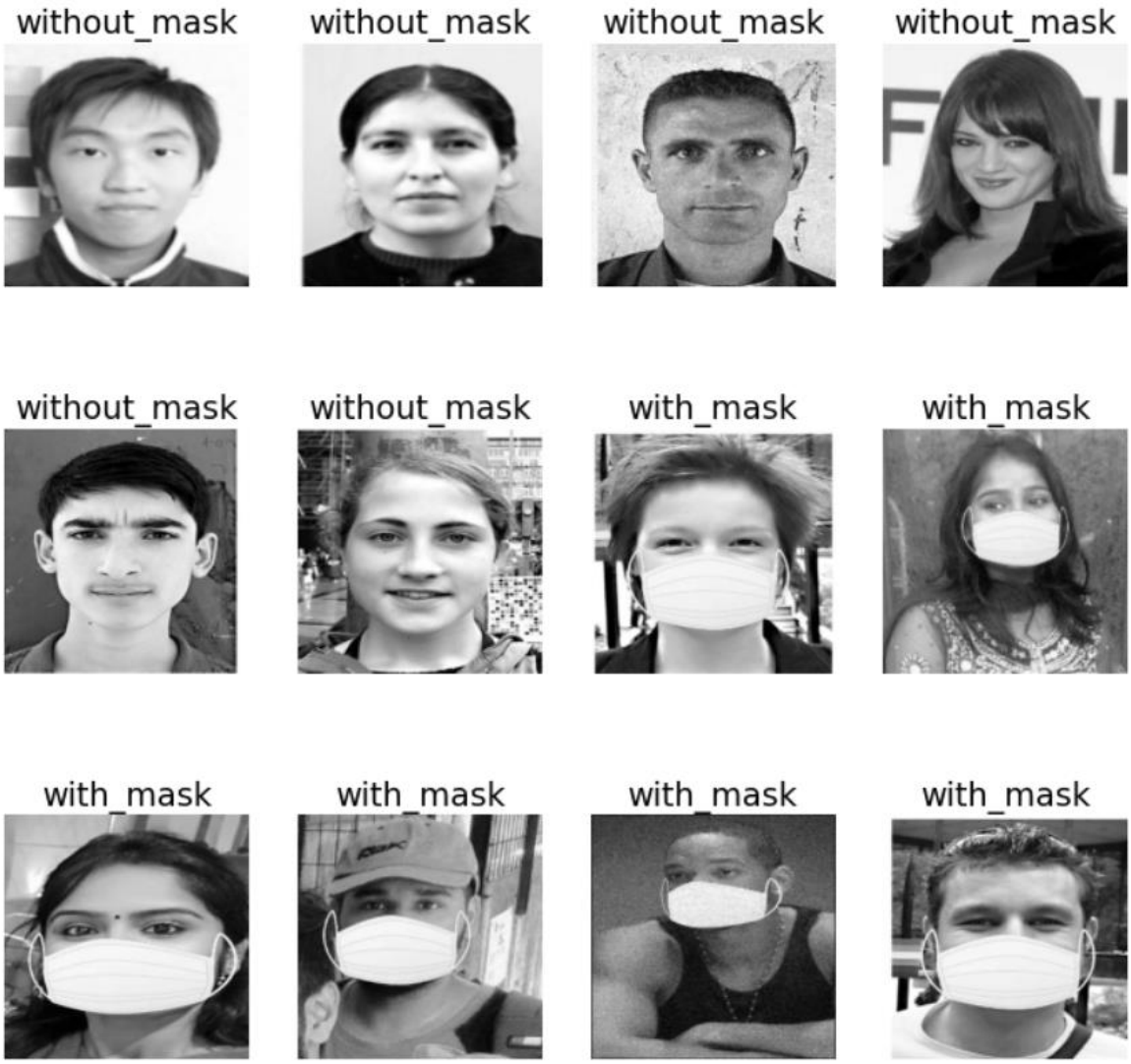
O conceito por trás desse processo é assumirmos que uma imagem continua representando a mesma classe mesmo com estas pequenas modificações, permitindo com que tenhamos outras ópticas sobre a mesma imagem. Esse contexto pode ser considerado como o equivalente à feature engineering em bases de dados tradicionais (não-imagens).

Para executar o data augmentation, a metodologia usada baseia-se no framework do Keras de ImageDataGenerator associado ao flow_from_dataframe, que cria um fluxo direto do diretório de imagens enquanto a definição de classes fica à cargo de um dataframe com o nome do arquivo (imagem) e sua respectiva classe. Desta forma, é possível trabalhar com um volume de dados maior sem onerar a capacidade computacional disponível.

Vale ressaltar que, como parte do tratamento das imagens também é necessário fazer o *reshaping* das imagens para formatos quadrados (por exemplo, 256x256 pixels), que são mais bem aceitos em redes neurais, especialmente as convolucionais. Adicionalmente, é realizada a normalização dos valores dos pixels para que eles assumam valores apenas entre 0 e 1, que é uma prática padrão para o aumento da performance de modelos estatísticos de aprendizado de máquina.

Por fim, para redução do consumo computacional e, entendendo que não existe perdas significativas, realiza-se a conversão das imagens de input para escala de cinza.

FIGURA 2 - IMAGENS COM O PROCESSO INICIAL APLICADO



3.4 TÉCNICAS DE MACHINE LEARNING UTILIZADAS

Para o desenvolvimento deste trabalho, foram utilizados dois tipos de redes neurais: ANN e CNN. As redes criadas em cada caso estão demonstradas a seguir.

Uma Rede Neural Artificial (do inglês, Artificial Neural Network ou ANN) são sistemas de computação inspirados nas redes neurais biológicas que constituem cérebros de animais. Esses sistemas são uma coleção de unidades ou nós conectados chamados de neurônios artificiais. Cada conexão, como as sinapses no cérebro biológico, pode transmitir um sinal para outros neurônios. Nas ANNs pode-se ter várias camadas de neurônios, aumentando a complexidade da rede.

TABELA 1 - SUMÁRIO DA ARQUITETURA ANN CRIADA

```
Model: "ANN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
entrada (InputLayer)         [(None, 256, 256, 1)]     0
_________________________________________________________________
achatamento (Flatten)        (None, 65536)             0
_________________________________________________________________
densa1 (Dense)               (None, 512)               33554944
_________________________________________________________________
reducao1 (Dropout)           (None, 512)               0
_________________________________________________________________
densa2 (Dense)               (None, 512)               262656
_________________________________________________________________
reducao2 (Dropout)           (None, 512)               0
_________________________________________________________________
densa3 (Dense)               (None, 512)               262656
_________________________________________________________________
reducao3 (Dropout)           (None, 512)               0
_________________________________________________________________
previsao (Dense)             (None, 1)                 513
=================================================================
Total params: 34,080,769
Trainable params: 34,080,769
Non-trainable params: 0
_________________________________________________________________
```

Uma Rede Neural Convolucional (do inglês, Convolutional Neural Network ou CNN) é um algoritmo de Deep Learning que pode captar uma imagem de entrada, atribuir importância (pesos e vieses que podem ser aprendidos) a vários aspectos / objetos da imagem e ser capaz de diferenciar um do outro. O pré-processamento exigido em uma CNN é muito menor em comparação com outros algoritmos de classificação.

Enquanto nos métodos primitivos os filtros são feitos à mão, com treinamento suficiente, as CNNs têm a capacidade de aprender esses filtros / características.

A arquitetura de uma CNN é análoga àquela do padrão de conectividade de neurônios no cérebro humano e foi inspirada na organização do Visual Cortex. Os neurônios individuais respondem a estímulos apenas em uma região restrita do campo visual conhecida como Campo Receptivo. Uma coleção desses campos se sobrepõe para cobrir toda a área visual.

TABELA 2 - SUMÁRIO DA ARQUITETURA CNN CRIADA

Model: "CNN"

Layer (type)	Output Shape	Param #
entrada (InputLayer)	[(None, 256, 256, 1)]	0
convolucao2d1 (Conv2D)	(None, 256, 256, 32)	320
normalizacao1 (BatchNormaliz	(None, 256, 256, 32)	128
convolucao2d2 (Conv2D)	(None, 256, 256, 32)	9248
normalizacao2 (BatchNormaliz	(None, 256, 256, 32)	128
acumulacao1 (MaxPooling2D)	(None, 128, 128, 32)	0
convolucao2d3 (Conv2D)	(None, 128, 128, 64)	18496
normalizacao3 (BatchNormaliz	(None, 128, 128, 64)	256
convolucao2d4 (Conv2D)	(None, 128, 128, 64)	36928
normalizacao4 (BatchNormaliz	(None, 128, 128, 64)	256
acumulacao2 (MaxPooling2D)	(None, 64, 64, 64)	0
convolucao2d5 (Conv2D)	(None, 64, 64, 128)	73856
normalizacao5 (BatchNormaliz	(None, 64, 64, 128)	512
convolucao2d6 (Conv2D)	(None, 64, 64, 128)	147584
normalizacao6 (BatchNormaliz	(None, 64, 64, 128)	512
acumulacao3 (MaxPooling2D)	(None, 32, 32, 128)	0
convolucao2d7 (Conv2D)	(None, 32, 32, 256)	295168
normalizacao7 (BatchNormaliz	(None, 32, 32, 256)	1024
convolucao2d8 (Conv2D)	(None, 32, 32, 256)	590080
normalizacao8 (BatchNormaliz	(None, 32, 32, 256)	1024
acumulacao4 (MaxPooling2D)	(None, 16, 16, 256)	0
achatamento (Flatten)	(None, 65536)	0
densa1 (Dense)	(None, 512)	33554944
reducao1 (Dropout)	(None, 512)	0
densa2 (Dense)	(None, 512)	262656
reducao2 (Dropout)	(None, 512)	0
densa3 (Dense)	(None, 512)	262656

```
_________________________________________________________________
reducao3 (Dropout)           (None, 512)               0
_________________________________________________________________
previsao (Dense)             (None, 1)                 513
=================================================================
Total params: 35,256,289
Trainable params: 35,254,369
Non-trainable params: 1,920
_________________________________________________________________
```

3.5 ESTRUTURAÇÃO DAS CAMADAS, TÉCNICAS E FUNÇÕES UTILIZADAS

Para a comparação de diferentes arquiteturas, foram construídas diferentes estruturas baseadas nas técnicas: *Flatten*, *Dense*, *Dropout*, *Conv2D*, *BatchNormalization* e *MaxPooling2D*. Cada técnica é aplicada a uma camada diferente das redes neurais.

A técnica de achatamento (no inglês, flatten) converte os dados de uma matriz multidimensional em uma matriz unidimensional (vetor) para os dados serem usados como input em uma próxima camada.

Uma camada densa (do inglês, *dense layer*) é apenas uma camada regular de neurônios em uma rede neural. Cada neurônio recebe a entrada de todos os neurônios da camada anterior estando, portanto, densamente conectados. Eles também, possuem uma função de ativação que é responsável por normalizar a saída dos neurônios (por exemplo, *ReLU*).

A função de ativação linear retificada (ou, abreviando do inglês *ReLU*) é uma função linear que produzirá a entrada 1 se for positiva; caso contrário, produzirá 0. Tornou-se a função de ativação padrão para muitos tipos de redes neurais, porque um modelo que a utiliza é mais fácil de treinar por ser de menor consumo computacional, geralmente obtendo melhor desempenho.

O processo de *BatchNormalization*consiste da normalização da camada ajustando os valores e dimensionando as ativações. Por exemplo, quando temos recursos de 0 a 1 e alguns de 1 a 1000, devemos normalizá-los para uma mesma escala.

A técnica de *Dropout* é usada para combater o *overfitting* (o estado de overfit de um modelo faz com que ele não seja generalizável para fora do universo de

possibilidades onde foi treinado). O método consiste no descarte aleatório de neurônios de uma camada densa.

A técnica de *MaxPooling2D* consiste em um processo de discretização baseado em amostra. O objetivo é fazer uma amostragem reduzida de uma representação de entrada (imagem, matriz de saída de uma camada oculta etc.), reduzindo sua dimensionalidade e permitindo suposições sobre os recursos contidos nas sub-regiões classificadas.

Para compilação dos modelos, algumas funções foram utilizadas: *BinaryCrossEntropy*, *Accuracy* e *SGD*, conforme explicado a seguir.

Para execução do trabalho, foi utilizada a função de perda *BinaryCrossEntropy* para medir os resultados de um modelo binário (no caso, com ou sem máscara). É também chamada de *Sigmoid Cross-Entropy loss* (função de perda de entropia cruzada sigmoide) ou *Binary Cross-Entropy* loss (função de perda de entropia binária cruzada).

A acurácia (do inglês, *Accuracy*) é uma métrica para avaliar modelos de classificação. Informalmente, precisão é a fração de previsões que nosso modelo acertou. Formalmente, a acurácia tem a seguinte definição:

$$Acurácia = \frac{Número\ de\ predições\ corretas}{Número\ total\ de\ predições}$$

A função de ativação sigmoide, também chamada de função logística, é tradicionalmente uma função de ativação muito popular para redes neurais. O resultado dessa ativação é a transformação de uma entrada em um valor entre 0 e 1. Esta função é especialmente usada em casos de classificação binária e é, atualmente, a única função de ativação compatível com a função de perda de entropia cruzada binária.

Por fim, para criação das redes foi utilizada a função de otimização *SGD* (do inglês, *Stochastic Gradient Descent* ou gradiente descendente estocástico), metodologia usada para a otimização dos parâmetros do modelo. No gradiente descendente estocástico, algumas amostras são selecionadas aleatoriamente em vez de todo o conjunto de dados para cada iteração.

4 RESULTADOS

Usando o referencial teórico mencionado anteriormente, treinou-se dois tipos de redes, uma rede neural artificial densa tradicional (ANN) e uma rede neural convolucional (CNN). No processo de seleção dos modelos finalistas, construiu-se diferentes profundidades de redes até que arquiteturas satisfatórias fossem encontradas. Este processo foi não-exaustivo. Ambas tiveram ativação final sigmoide, com função de perda *Binary Cross-entropy* e otimizador SGD.

Para a ANN final utilizou-se de camadas *Flatten*, *Dense* (*ReLU*) e *Dropout* e obteve-se um total de 34.080.769 parâmetros treináveis.

Já para a CNN final, aplicou-se camadas *Conv2D*, *Dropout*, *BatchNormalization*, *MaxPooling2D*, *Flatten* e *Dense* (*ReLU*) e obteve-se um total de 35.254.369 parâmetros treináveis.

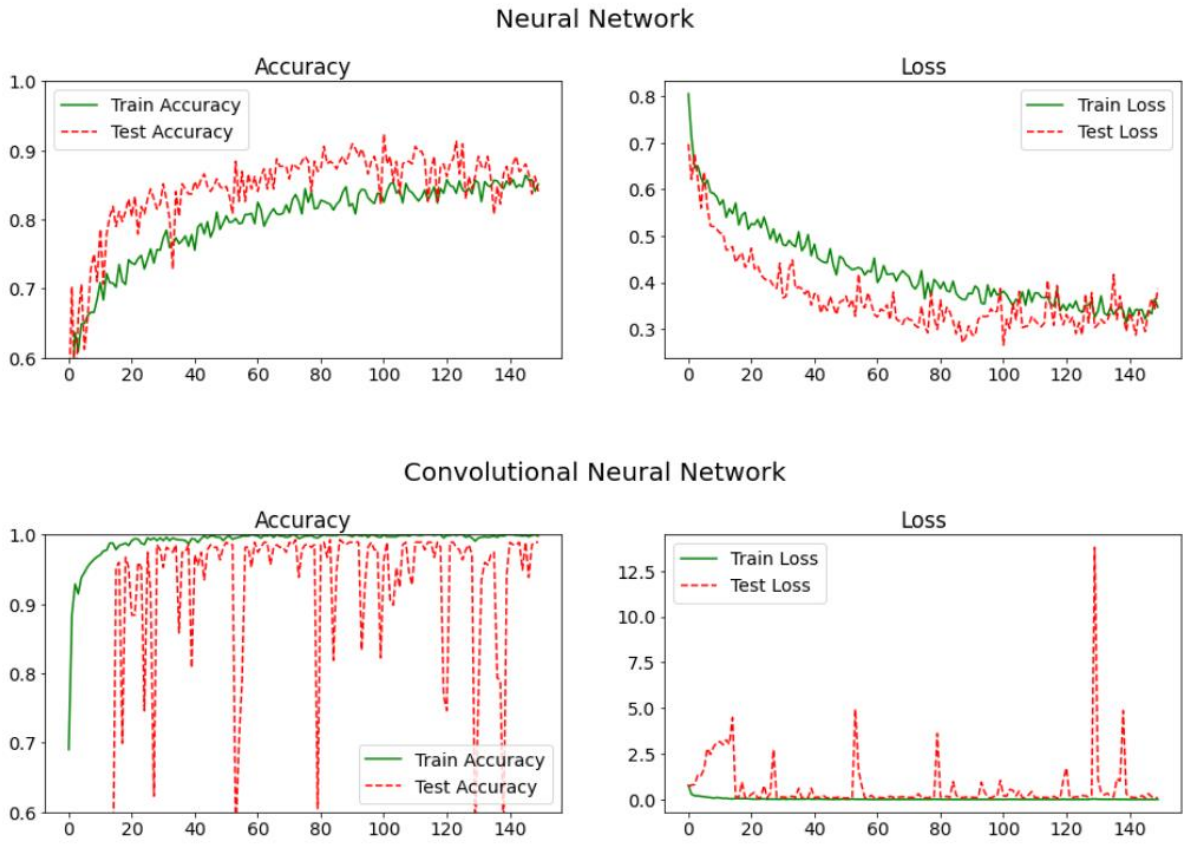
A métrica de performance usada foi acurácia. Ela foi usada tanto para a seleção da arquitetura desejada quanto para a escolha do modelo final. Todos os treinamentos foram feitos com um total de 150 épocas.

Após a execução do treino e teste, verificou-se as frequências de classificação para cada classe do modelo, utilizando Matrix de Confusão (do inglês, *Confusion Matrix*). As seguintes frequências foram mostradas:

- Verdadeiro positivo (*true positive*, TP), ocorre quando no conjunto real (*true label*), a classe que estamos buscando foi prevista corretamente.
- Falso positivo (*false positive*, FP), ocorre quando no conjunto real (*true label*), a classe que estamos buscando prever foi prevista incorretamente.
- Falso verdadeiro (*true negative*, TN): ocorre quando no conjunto real (*true label*), a classe que não estamos buscando prever foi prevista corretamente.
- Falso negativo (*false negative*, FN): ocorre quando no conjunto real (*true label*), a classe que não estamos buscando prever foi prevista incorretamente.

Para as arquiteturas finais, observamos que a CNN converge mais rápido que a ANN a resultados ótimos. Entretanto, observamos que a ANN converge com menor variância da função perda.

GRÁFICO 1 - VERIFICANDO A ACURÁCIA DOS MODELOS



Os resultados pós-treino nos mostram que a CNN performou muito melhor que a ANN (avaliado pela curva ROC). Apesar de os resultados da ANN terem sido bastante satisfatórios a rede convolucional, como descrito em literatura (RANJAN, R. et al. e LAWRENCE, S. et al.) costumam ter resultados extraordinários com processamento de imagem.

Uma observação importante quanto aos testes de predição finais mostrados no *notebook Jupyter* deste trabalho, é quanto a possibilidade de realização de um pré-processamento da face a ser analisada antes que seja passada para o modelo. Este ponto será explorado no próximo capítulo, onde serão abordadas sugestões de *deployment* do modelo.

GRÁFICO 2 - MATRIX DE CONFUSÃO DAS REDES ANN E CNN

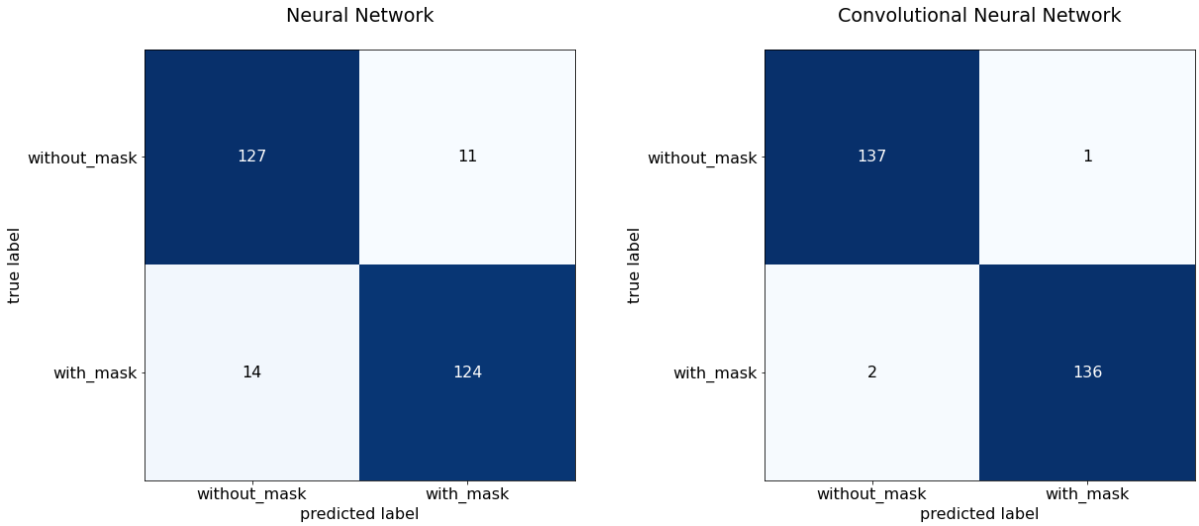
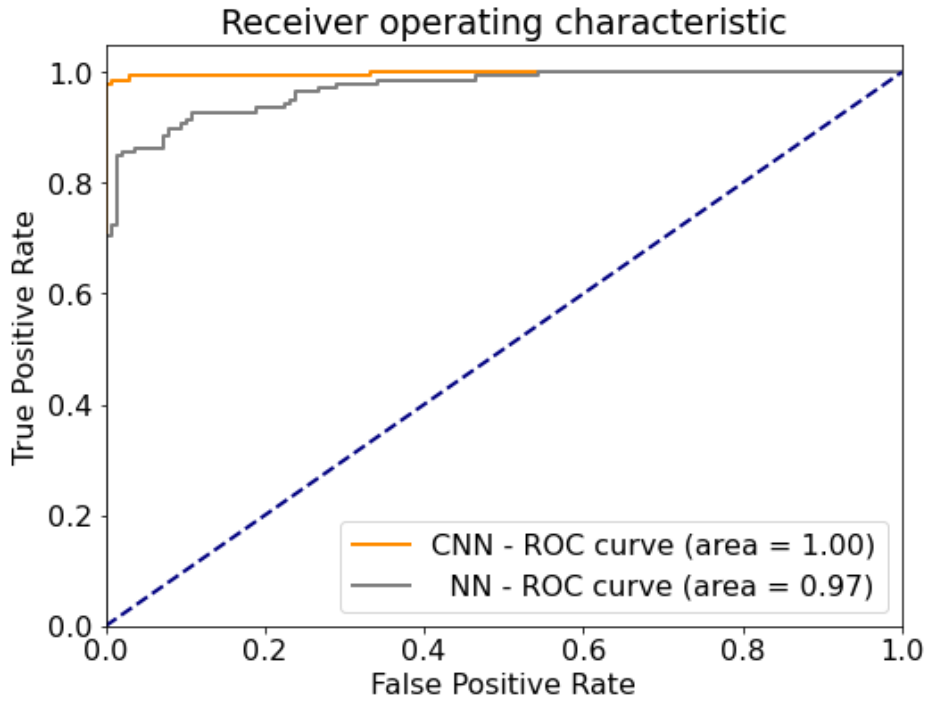


GRÁFICO 3 - AVALIAÇÃO DA PERFORMANCE PELA CURVA ROC



5 DEPLOYMENT DOS MODELOS EM UMA APLICAÇÃO REAL

O modelo desenvolvido não está acoplado a um framework de *computer vision*, portanto, seu foco de predição é na identificação do uso da máscara facial. O deployment é um fator importante e que pode trazer seus próprios desafios.

Como desdobramento do trabalho proposto nesta pesquisa, objetiva-se aplicar o modelo construído em um ambiente de produção para detecção do uso de máscara facial. Uma das possibilidades de entrega (*deployment*) do modelo em uma aplicação real poderia ser através do monitoramento em tempo real de pessoas circulando em um ambiente público, por exemplo.

Para este fim, a utilização de bibliotecas existentes de detecção de faces traria um ponto de partida ideal para o modelo de detecção de máscaras. Na Listagem 2, vemos um exemplo de declaração importando as bibliotecas do *OpenCV* e do *Keras* (*TensorFlow*).

LISTAGEM 1 - CARREGANDO MODELO KERAS DO TENSORFLOW

```
1  import cv2
2  import numpy as np
3  from tensorflow.keras.models import load_model
4
5  model = load_model("../models/model_CNN.h5")
```

5.1 OPENCV COMO FRAMEWORK DE SUPORTE PARA O MODELO

Existem diversos frameworks disponíveis no mercado para realizar tasks de computer vision, como o projeto open source OpenCV (disponível para Python), que possui a capacidade de identificação e recorte de face (crop), realizando um pré-processamento para o modelo executar a predição.

Em uma imagem típica, capturada por uma câmera em um ambente público, a maior parte da região da mesma não possui um rosto (face). Portanto, é melhor ter um método simples para verificar se uma janela não é uma região de face. Caso contrário, deve-se descartá-la.

Um dos algorítmos mais conhecidos para este fim, faz uso conceito de cascata de classificadores (do inglês, *Cascade Classifier*). O OpenCV vem com um

treinador e um detector, e já contém muitos classificadores pré-treinados para rosto, olhos, sorriso etc. Os classificadores estão disponíveis em arquivos XML, e podem ser utilizados como demostrado nas linhas 26 e 31 da Listagem 2.

LISTAGEM 2 - CLASSIFICADOR HAAR CASCADE DO OPENCV

```
26  classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
27
28  cap = cv2.VideoCapture(0)
29  while True:
30      ret, img = cap.read()
31      faces = classifier.detectMultiScale(cv2.cvtColor(img,
32                                          cv2.COLOR_BGR2GRAY), 1.1, 2)
33
34      for face in faces:
35
36          # Código para executar a predição
37          # com cada face detectada pelo OpenCV
38
39      cv2.imshow('FaceMask Detection', img)
40      if cv2.waitKey(1) & 0xff == ord('q'):
41          break
42
43  cap.release()
44  cv2.destroyAllWindows()
```

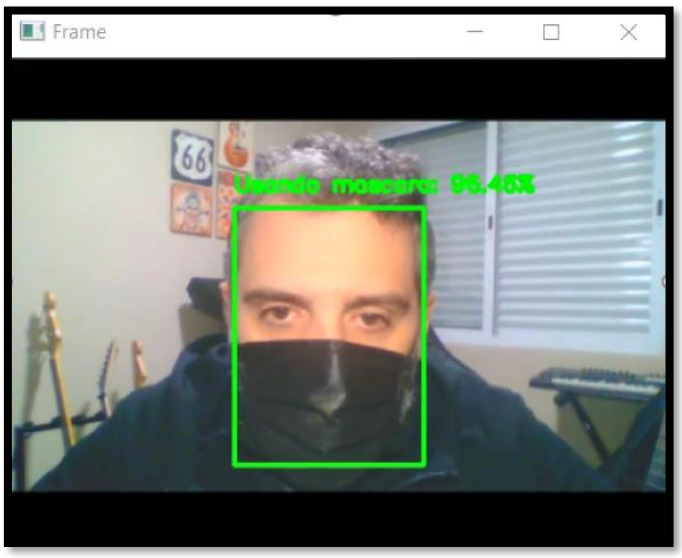
Uma vez que o framework *OpenCV* fez a detecção da face, utilizando o classificador *Haar Cascade*, basta recortar a imagem da face detectada e passá-la para que o modelo *Keras* faça a sua predição, conforme ilustrado na Listagem 3. O OpenCV também permite adicionar um destaque no rosto detectado e combiná-lo com a predição do modelo, conforme mostrado nas linhas 39 a 42.

LISTAGEM 3 - CARREGANDO MODELO KERAS DO TENSORFLOW

```
34  for face in faces:
35      sliced_img = img[face[1]:face[1] + face[3], face[0]:face[0] + face[2]]
36      prediction = model.predict(prepare_image(sliced_img))
37      prediction = np.argmax(prediction)
38
39      cv2.rectangle(img, (face[0], face[1]), (face[0] + face[2], face[1] +
40                    face[3]), pred_color[prediction], 2)
41      cv2.putText(img, pred_label[prediction], (face[0], face[1] - 10),
42                  cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
```

Um exemplo de uso real do modelo e do OpenCV é ilustrado na Figura 3, onde o rosto é primeiramente detectado pelo OpenCV e na sequência passado para detecção do uso de máscara pelo modelo *Keras*, criando anteriormente.

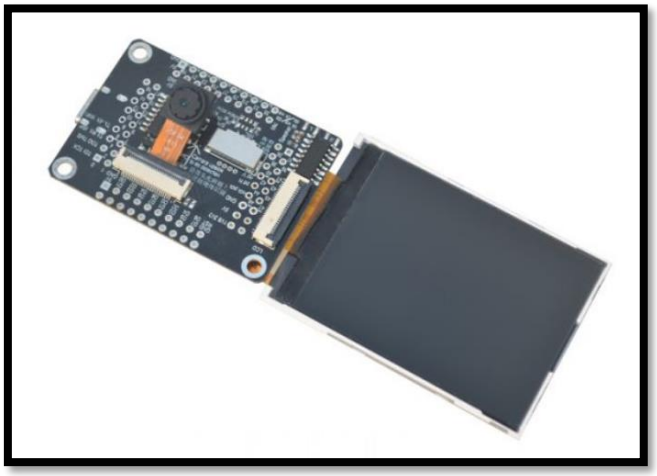
FIGURA 3 - OPENCV UTILIZANDO O MODELO KERAS



5.2 HARDWARES PROJETADOS PARA MACHINE LEARNING

Outra opção de implementação é utilizando hardwares projetados especialmente para rodar modelos de machine learning baseados em CNN, como os dispositivos de AIoT (um acrônimo que junta *Artificial Intelligence*, ou inteligência artificial, com *Internet of Things*) da *Seeed*, como o *Sipeed M1*.

FIGURA 4 - SIPEED M1 DOCK SUIT



Este hardware roda utilizando *MicroPython*, uma implementação enxuta e eficiente da linguagem Python 3 e é otimizado para rodar em microcontroladores e em ambientes restritos.

6 CONCLUSÕES

A proposta do trabalho foi plenamente atendida com o desenvolvimento de um modelo, com técnicas de deep learning, capaz de realizar a detecção de pessoas utilizando máscaras faciais. O modelo também se mostrou viável para posterior implementação em sistemas de controle de entrada em espaços públicos, estabelecimentos que executem atividades essenciais, repartições públicas estaduais, transporte por aplicativo para um público alvo de consumidores, fornecedores, clientes, empregados, colaboradores, agentes públicos e prestadores de serviço.

O presente trabalho também suporta e apoia futuros trabalhos acadêmicos ou voltados para aplicações no mercado, oferecendo pontes de corroboração com a aplicação de técnicas de machine learning para predições baseadas em análise de imagens.

REFERÊNCIAS

LI, H.; LIN, Z.; SHEN, X.; BRANDT, J.; HUA, G. **A Convolutional Neural Network Cascade for Face Detection**. 28th IEEE Conference on Computer Vision and Pattern Recognition. Boston, MA. 2015. DOI 10.1109/CVPR.2015.7299170.

RANJAN, R.; SANKARANARAYANAN, S.; CASTILLO, C. D.; CHELLAPPA, R. **An All-In-One Convolutional Neural Network for Face Analysis**. IEEE 12th International Conference on Automatic Face & Gesture Recognition. Washington, USA. 2017. DOI 10.1109/FG.2017.137.

NAUDÉ, W. **Artificial Intelligence against COVID-19: An Early Review**. IZA – Institute of Labor Economics. Bonn, Alemanha. 2020. ISSN 2365-9793.

GUO, J.; LIN, C.; WU, M.; CHANG, C.; LEE, H. **Complexity Reduced Face Detection Using Probability-Based Face Mask Prefiltering and Pixel-Based Hierarchical-Feature Adaboosting**. IEEE SIGNAL PROCESSING LETTERS. VOL. 18, NO. 8, AUGUST 2011. DOI 10.1109/LSP.2011.2146772.

LAWRENCE, S.; GILES, C. L.; TSOI, A. C.; BACK, A. D. **Face Recognition: A Convolutional Neural-Network Approach**. IEEE Transactions on Neural Networks, Vol. 8, No. 1. 1997. DOI 10.1109/72.554195.

QIN, B.; LI, D. **Identifying Facemask-wearing Condition Using Image SuperResolution with Classification Network to Prevent COVID-19.** 2020. DOI 10.21203/rs.3.rs-28668/v1.

WANG, Z.; et al. **Masked Face Recognition Dataset and Application**. 2020. arXiv 2003.09093v2.

YIN, X.; LIU, X. **Multi-Task Convolutional Neural Network for Pose-Invariant Face Recognition**. IEEE Transactions on Image Processing. 2018. DOI 10.1109/TIP.2017.2765830.

JIANG, M.; FAN, X. **RetinaMask: A Face Mask Detector**. 2020. arXiv 2005.03950.

MATSUGU, M.; MORI, K.; MITARI, Y.; KANEDA, Y. **Subject independent facial expression recognition with robust face detection using a convolutional neural network**. Neural Networks, Volume 16, Issues 5-6, Pages 555-559. 2003. DOI 10.1016/S0893-6080(03)00115-1.

CHOLLET, F. **Deep Learning with Python**. Manning. ISBN 9781617294433. NY, USA. 2018.

GÉRON, A. **Hands-On Machine Learning with Scikit-Learn & TensorFlow: Concepts, Tools, And Techniques To Build Intelligent Systems**. O'Reilly. ISBN 978-1-491-96229-9. CA, USA. 2017.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R**. An Introduction to Statistical Learning with Applications in R**. Springer. ISBN 978-1-4614-7138-7. DOI 10.1007/978-1-4614-7138-7. NY, USA. 2015.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. Springer. ISBN 978-0-387-84858-7. DOI: 10.1007/b94608. NY, USA. 2009.

KUHN, M.; JOHNSON, K. **Applied Predictive Modeling**. Springer. ISBN 978-1-4614-6849-3. DOI 10.1007/978-1-4614-6849-3. NY, USA. 2013.

Keras API Reference. Disponível em <https://keras.io/api/>. Acesso em: 30 de Maio de 2020.

RUSSEL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach.** 2010. DOI 10.1016/j.artint.2011.01.005.

LUNDBERG, S.; LEE, S. **A Unified Approach to Interpreting Model Predictions**. 2017. arXiv 1705.07874.

World Health Organization. **Rational Use of Personal Protective Equipment for Coronavirus Disease (COVID-19) and Considerations During Severe Shortages.** 2020. WHO/2019-nCov/IPC_PPE_use/2020.3.

SELVARAJU, R.; et al. **Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization**. 2019. DOI 10.1007/s11263-019-01228-7.