Uso de Machine Learning Para o Reconhecimento de Máscaras

Rebecca Pereira Martinho
Instituto Federal Catarinense - IFC
Videira, SC - Brasil
Email: rebeccamartinho1@gmail.com

***Resumo*—Com o uso de Machine Learning foi possível criar um script na linguagem Python para reconhecer se um rosto está ou não portando uma máscara. O editor utilizado foi o Google Colab, e as principais bibliotecas utilizadas foram a OpenCV para o processamento das imagens e scikit-learn para o treinamento do modelo de classificação. Além da possibilidade de utilizar imagens com um conjunto de pessoas, recortar as faces encontradas e criar uma arquivos com as mesmas, o código já utiliza uma base de imagens (1.276 no total) com e sem máscara para o treinamento, que resultou numa acurácia de 92%. Transferindo o código para o editor Visual Studio Code, foi possível utilizar a webcam e realizar a verificação em tempo real.**

***Index Terms*—Machine Learning, Inteligência Artificial, Máscara, Detecção.**

I. Introdução

O aumento na busca por tecnologias que visem facilitar as interações humanas é notório, assim como a necessidade de automação das mesmas. As técnicas de Inteligência buscam fazer com que as soluções tecnológicas realizarem atividades de um modo considerado inteligente. Uma dessas técnicas é a *Machine Learning*, muito popular atualmente e que consiste no aprendizado de acordo com as respostas esperadas por meio associações de diferentes dados [1], desse modo, pode ser usada para realizar atividades afim de proporcionar mais conforto e praticidade para as pessoas.

No contexto do estado de pandemia, decretado pela Organização Mundial da Saúde no ano de 2020, diversas medidas preventivas foram tomadas em todo país, dentre elas o isolamento social e o uso obrigatório da máscara facial [2]. Diversos estabelecimentos frisam a obrigatoriedade do uso da máscara, porém dispender um funcionário para ficar somente verificando essa normativa é custoso. Diante dessa realidade, o uso de um sistema detector de máscara utilizando a técnica de *Machine Learning* é de grande utilidade.

Nesse âmbito, este trabalho busca exemplificar um sistema detector de máscara facial com visão computacional e *Machine Learning* com aprendizado supervisionado (detectando padrões para estabelecer previsões), usando a linguagem Python, e algumas bibliotecas necessárias como OpenCV, Matplotlib, ScikitLearn Pandas, OS e Numpy.

II. Revisão literária

A. Machine Learning

Machine Learning é uma área da Inteligência Artificial que fornece ao computador a habilidade de aprender uma determinada tarefa sem ser explicitamente programada, ela consegue identificar padrões através de um conjunto de dados robusto [3]. Para [4], Machine Learning é um sistema que pode modificar seu comportamento autonomamente tendo como base a sua própria experiência, com pouca interferência humana.

B. Visão computacional

Nesse projeto, a visão computacional foi utilizado sem o Deep Learning. Nessa situação, o aprendizado de máquina, para cada imagem do Dataset usado, é realizado um processo chamado extração de características (*feature extraction*), em que a imagem de entrada é quantificada de acordo com um algoritmo específico (*feature extractor*), que retorna um vetor que tem o objetivo de quantificar o conteúdo da imagem, que é usado como entrada para o modelo de *Machine Learning* com a quantificação da imagem (o processo de extração de recursos), que, por sua vez, é usado como entrada para o modelo de *Machine Learning* [5].

C. K-nearest neighbors (KNN)

O KNN (K-nearest neighbors, ou “K-vizinhos mais próximos”, é um algoritmo onde k é um número determinado de vizinho. Ao supormos um conjunto dividido em duas classes (azul e vermelho), k=3, e um novo dado adicionado porém não classificado, é medida a distância do novo dado em relação a cada um dos outros já classificados, assim, ocorre uma espécie de "votação"onde a classificação da maioria desses k próximos vizinhos vence [6]. A classificação do novo dado é dada como pertencente à classe que mais apareceu, que no caso da figura 1, será a cor vermelha. Dependendo do valor de k, poderemos ter resultados diferentes para cada situação.

D. Cascade Classifier e o Algoritmo Viola-Jones

O Cascade Classifier é uma implementação do algoritmo Viola-Jones, uma abordagem de*Machine Learning* para detecção de objetos visuais que é capaz de processar imagens de modo extremamente rápido [7]. Esse algoritmo utiliza Features de Haar para detectar faces a partir de padrões de luminosidade utilizando as "máscaras"presente na figura 2. Cada máscara

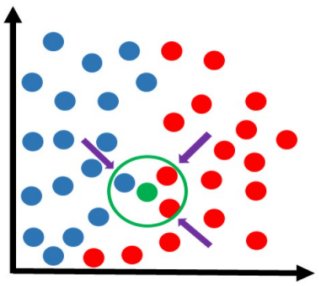

Figura 1. Algoritmo KNN Fonte: [6]

detecta as features de Haar em direções diferentes. O Haar produz um único valor pegando a soma das intensidades das regiões claras e subtraindo pela soma das intensidades das regiões resultando na detecção ou não de uma face [8]. Para melhorar o desempenho do algoritmo, o OpenCV utiliza o Adaboost, que é um algoritmo de aprendizado de máquina usado para aumentar a performance de outros algoritmos. O Adaboost (Adaptive Boosting) essencialmente escolhe os melhores recursos e treina os classificadores para usá-los, formando um "classificador forte" [9].

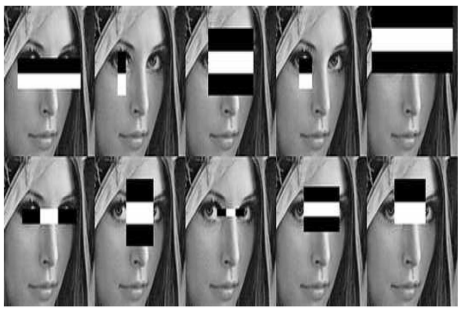

Figura 2. atributos Haar calculadas sobre uma imagem Fonte: [9]

III. Desenvolvimento

A. Linguagem e bibliotecas

A linguagem utilizada foi a Python, e o editor foi o Google Colab. As principais bibliotecas utilizadas foram a OpenCV para o processamento das imagens e scikit-learn para o treinamento do modelo de classificação, além damatplotLib, OS, Pandas e Numpy, assim, foi possível que importemos imagens direto pelo Google Colab com apenas um célula do notebook, com a leitura da imagem importada através do método imread do OpenCV. A visualização das imagens foi feita através do Matplotlib convertida do padrão do OpenCV (BGR) para o padrão RGB. Isso é útil para a visualização do resultado do processamento da imagem carregada.

B. Reconhecimento de faces

Antes de reconhecer se um rosto está ou não utilizando uma máscara facial, é necessário reconhecer uma face. Ao pré-processar a imagem a ser utilizada, a ocorrência de ruídos é diminuída. O único processamento necessário é a conversão da escala de cor BGR para tons de cinza RGB a partir do método cvtColor.

É utilizado do método Cascade Classifier(classificador de cascata) que é uma implementação do algoritmo Viola-Jones para a detecção de objetos em imagens e reconhecimento facial.

Ao importarmos o OpenCV, temos acesso a diversos arquivos .xml que podem ser utilizados no treinamento de modelos de reconhecimento, que são chamados de haar cascades [10]. Para a identificação de rostos foi utilizado o modelo haarcascade_frontalface_alt2.xml. Desse modo, esse arquivo irá sinalizar a localização das features de Haar que já vem com a biblioteca, assim é possível formatar uma string que irá nos mostrar com precisão o caminho do nosso arquivo. Após a instância e treinamento do modelo através das features de Haar, foi possível realizar as predições necessárias para detectar faces na imagem usando o método detectMultiscale, bastando passar a imagem em tons de cinza, pois o algoritmo de Viola-Jones não reconhece todos os canais de cores (RGB). O retorno desse método foi um array de coordenadas, e cada uma se refere a um posição a qual uma face foi detectada na imagem. Após a detecção das faces, foi possível mostrar a localização exata das mesmas, para isso, é utilizado o método rectangle do OpenCV que basicamente desenha um retângulo em volta dos rostos detectados de cada face presente no array (figura 3). Cada face retorna uma coordenada do ponto inicial da largura da face, o ponto inicial da altura da face e os pontos finais de largura e altura.

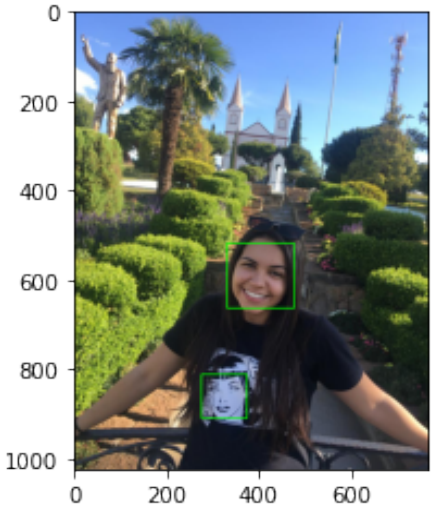

Figura 3. Reconhecimento de faces. Fonte: A autora

C. Padronização das imagens

É importante possuir as imagens padronizadas, por isso todas as faces identificadas foram recortadas, redimensionadas e salvas em uma lista. Para o recorte, é utilizado o slice e para o redimensionamento é utilizado o resize, que padroniza as imagens com o valor da altura e da largura desejados, nesse

caso, 160x160, como é exemplificado pela figura 4. Essas imagens podem ser salvas e utilizadas para treino e teste.

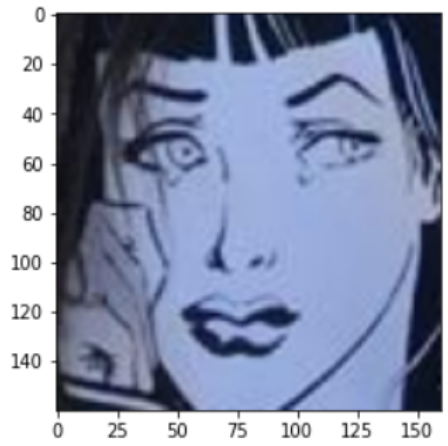

Figura 4. Face reconhecida, recortada e padronizada. Fonte: A autora

D. Organização das imagens em um diretório

Ao utilizarmos a biblioteca os é possível criar um caminho referente a uma pasta na qual serão salvas as imagens. No código em questão, o caminho foi uma pasta no drive que estava conectado ao Google Colab. O método cv.imwrite foi utilizado para escrever um arquivo com a imagem, bastando receber um caminho com o nome da imagem (que será o index) e a própria imagem. Desse modo é possível criar um conjunto de dados para o treinamento do modelo.

E. Treinamento do modelo

O algoritmo de *Machine Learning* escolhido para classificação foi o K-Nearest Neighbors (KNN), simples porém eficaz [11], que após treinado, classifica o rosto como com ou sem máscara. Com o drive montado no Google Colab, foi possível importar duas pastas já previamente preparadas. Esse conjunto de dados consiste em 1.276 imagens pertencentes a duas classes: Com máscara: 318 imagens e Sem máscara: 958 imagens. Após a importação dessas pastas, foi possível montar e carregar um DataFrame da biblioteca Pandas com as imagens. O DataFrame possui um dicionário Dados com três chaves de listas, a primeira receberá o nome do arquivo, a segunda um rótulo que informará se a face está com ou sem máscara e o último irá informar o estado da imagem (alvo), com 1 significando com a máscara e 0 sem. Para ambos os conjuntos, com máscara e sem máscara, será listado os arquivos contidos em cada diretório, que serão percorridos e salvos no dicionário, que após ser preenchido, é passado como um parâmetro na criação de um DataFrame como um arquivo CSV buscando agilizar a consulta.

A leitura do DataFrame foi feita a partir da chave arquivo, que foi percorrida, transformando a imagem em escalas de cinza e em um vetor utilizando o método flatten, e inserindo em uma lista de imagens, buscando simplificar o ato de passar a imagem como uma entrada. Também foi acrescida uma nova coluna chamada de imagem ao DataFrame, que irá conter o vetor de todas as imagens lidas. O conjunto das características (X) será a coluna imagem, enquanto a classe para identificação (y) será o alvo. A separação do conjunto de treino e de teste é de 99% pois o conjunto está desbalanceado (maioria das imagens sem máscara), e o *random_state* será 13.

F. Refinamento das características das imagens

Buscando refinar as principais características encontradas em uma face, é implementado a Análise de Componentes (PCA), que é um método estatístico que tem a finalidade de analisar um conjunto de dados e extrair as características mais notáveis para realizar a classificação. Nesse caso, 30 componentes serão extraídos da imagem. O modelo é treinado usando o conjunto de treino e todas as entradas de características (X treino e X teste) serão transformadas em vetores de tamanho 30 filtrados a partir do modelo PCA.

Após a extração das características das imagens, foi possível realizar o treinamento do algoritmo KNN. Esse algoritmo funciona a partir da inserção de um novo dado no plano. O KNN irá calcular a distância dele para K elemento já existentes dentro do conjunto, quando a maior parte dos K elementos mais próximos do nosso novo dado é igual a uma classe, classificamos esse dado como sendo da mesma classe.

A estratégia do Grid Search também foi usada, visando a escolha dos hiperparâmetros. Um dicionário recebeu uma lista de opções para os hiperparâmetros do algoritmo KNN, que foram: o número k (Quantidade de vizinhos), uma lista de opções de pesos e uma lista de opções de cálculos de métricas. Instanciando o GridSearchCV passando o algoritmo KNN vazio e os parâmetros do dicionário, o algoritmo realiza a escolha do melhores parâmetros para o treinamento do modelo, que após ser treinado foi possível verificar algumas métricas de desempenho como o score, falsos negativos, falsos positivos, verdadeiros negativos e verdadeiros positivos. O score (acurácia) foi de 92%, enquanto a matriz de confusão foi formada pela detecção de 10 pessoas sem mascara corretamente e apenas duas incorretamente, enquanto em relação ao uso de máscara foi verificado 2 reconhecimento certos e 1 errado (figura 5).

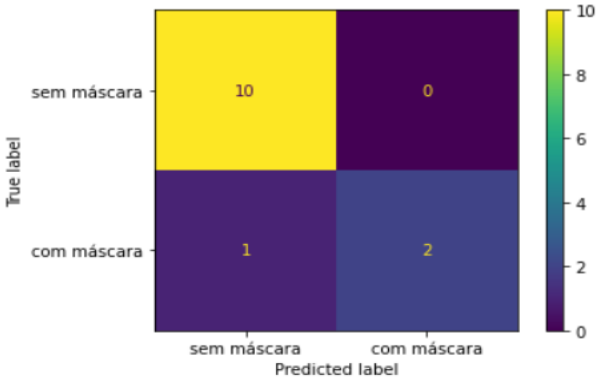

Figura 5. Métricas de desempenho. Fonte: A autora

G. Testagem

Para testar o modelo, foram processadas imagens aleatórias de pessoas na rua com base no modelo PCA e no classificador de faces. Foi atribuído o valor "com máscara"na classe de número 1 e o valor "sem máscara"nas classes de número 0. Ao realizar a predição para cada face encontrada foi possível verificar que na figura 6 três faces foram detectadas, sendo a primeira um ruído, porém as outras duas imagens foram acertadas, enquanto na figura 7, houve 100% de acerto.

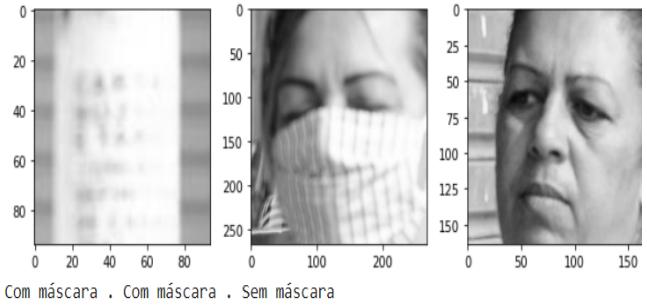

Figura 6. Testagem do treinamento. Fonte: A autora

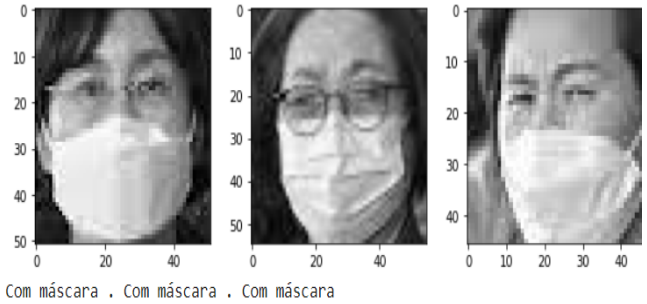

Figura 7. Testagem do treinamento. Fonte: A autora

H. Aplicação utilizando o modelo treinado

A partir do treinamento do modelo foi possível criar um script que realiza a verificação do uso ou não da máscara em tempo real, que irá por um quadrado verde em volta da face que estiver usando máscara, e um quadrado vermelho quando a face estiver sem a máscara, como é ilustrado pela figura 8. Na tela também é mostrado a quantidade de faces detectadas pelo algoritmo assim como o seu status (com máscara ou sem máscara).

IV. Conclusão

Nesse trabalho foi apresentado um estudo de caso de aplicação de *Machine Learning* para o reconhecimento de máscaras. O algoritmo apresentou um bom funcionamento, com uma acurácia de 92% no momento dos teste. É importante salientar algumas instabilidades no momento da detecção da máscara em tempo real, provavelmente por conta cor da máscara e/ou luminosidade do ambiente. Para futuros estudos é interessante aumentar o conjunto de dados, com máscaras de tipos e cores variadas além de faces utilizando acessórios. Também é

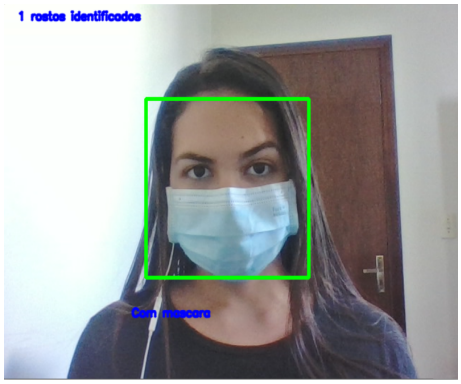

Figura 8. Reconhecimento de faces em tempo real. Fonte: A autora

interessante utilizar *Deep Learning* para melhorar o algoritmo, implementando o *image augmentation* e melhorando a base de dados.

Referências

[1] A. C. Müller and S. Guido, *Introduction to machine learning with Python: a guide for data scientists*. "O'Reilly Media, Inc.", 2016.

[2] A. T. Toledo, "Trabalho remoto no serviço público: O novo normal?," *Boletim Economia Empírica*, vol. 1, no. 3, 2020.

[3] W. L. HOMEM, "Apostila de machine learning," *UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO PROGRAMA DE EDUCAÇÃO TUTORIAL ENGENHARIA MECÂNICA*, 2020.

[4] E. ALECRIM, "Machine learning: o que é e por que é tão importante," 2018. Online; acesso em 17 Julho, 2021, https://tecnoblog.net/247820/machine-learning-ia-o-que-e/.

[5] M. TIBAU, "Visão computacional na era do deep learning," 2021. Online; acesso em 15 Julho, 2021, https://www.updateordie.com/2021/05/14/visao-computacional-na-era-do-deep-learning/.

[6] D. Tech, "O que é e como funciona o algoritmo knn?," 2020. Online; acesso em 18 Julho, 2021, https://didatica.tech/o-que-e-e-como-funciona-o-algoritmo-knn/.

[7] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001.

[8] S. Lee, "Understanding face detection with the viola-jones object detection framework," 2020. Online; acesso em 17 Julho, 2021, https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc2a9da14.

[9] A. MITTAL, "Haar cascades, explained," 2020. Online; acesso em 12 Julho, 2021, https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d.

[10] OpenCV, "Cascade classifier." Online; acesso em 17 Julho, 2021, https://docs.opencv.org/3.4/db/d28/tutorial$_c ascade_c lassifier.html$.

[11] B. POULSON, "Fundamentos da ciência de dados," 2019. Online; acesso em 17 Julho, 2021, https://www.linkedin.com/learning/fundamentos-da-ciencia-de-dados/k-vizinhos-mais-proximos-knn.