Relatório do Trabalho Prático de Processamento e Análise de Imagens

Davi Lorenzo Bambirra Braga[1]

[1]Instituto de Ciências Exatas e Informática – PUC MG

1. Introdução e problema

Desde o final dos anos 90, com o desenvolvimento vertiginoso das placas gráficas juntamente com a melhora na qualidade das imagens geradas por máquinas fotográficas e o aumento da capacidade computacional, a área de reconhecimento de imagens tem recebido cada vez mais importância em diversas áreas além da computação. Uma delas é na área médica, na qual o reconhecimento de problemas de saúde o mais cedo possível no paciente podem causar uma probabilidade de recuperação muito maior tanto em qualidade quanto em tempo. O problema apresentado neste trabalho consiste no treinamento de um algoritmo para reconhecimento de imagens de células cancerígenas de acordo com as classificações do Teste de Bethesda. O trabalho também apresenta outras funcionalidades para avaliação das imagens, como por exemplo análise do histograma na tonalidade HSV (Hue, Saturação e Valor), bem como o uso de descritores de Haralick para analisar a imagem de acordo com sua entropia, contraste e homogeneidade.

2. Técnicas implementadas

O código do projeto foi realizado na linguagem Python e dividido em três partes de forma a facilitar na compreensão e também facilitar na depuração e realização de testes. Cada tópico a seguir contém as funções presentes em cada um dos arquivos:

2.1. parte1.py

Este arquivo apresenta todas as funções solicitadas na primeira parte do trabalho, além de funções de abertura e verificação de imagens e arquivos .csv.

2.1.1. Conversão para Preto e Branco

A conversão de uma imagem para preto e branco é realizada pela função `EscalaDeCinza`. Esta função utiliza o método `convert("L")` para transformar a imagem no modo "L" (luminância), resultando em uma imagem em escala de cinza. Após a conversão, a profundidade de cor da imagem é reduzida para 16 tons distintos de cinza usando a função `point(lambda p: p // 16 * 16)`. Essa redução simplifica a análise subsequente ao limitar o número de tons que a imagem pode ter, o que pode ser útil em diversas aplicações de processamento de imagem.

2.1.2. Histogramas

Dois tipos de histogramas são calculados: em escala de cinza e HSV. Para o histograma em escala de cinza, a função `HistogramaCinza` primeiro converte a imagem para

escala de cinza usando a função `EscalaDeCinza`. Em seguida, calcula a distribuição dos tons de cinza em 16 bins usando `np.histogram` e plota essa distribuição em um gráfico de barras com `plt.bar`. Já a função `HistogramaHSV` converte a imagem para o espaço de cores HSV e separa a imagem em três canais: Hue (H), Saturação (S) e Valor (V). Para cada canal, um histograma é calculado com 256 bins utilizando `np.histogram`, e cada histograma é plotado em subgráficos distintos para visualizar a distribuição dos valores de cada canal.

2.1.3. Descritores de Haralick

Os descritores de Haralick são extraídos a partir das matrizes de co-ocorrência, que são calculadas pela função `CoOcorrencia` para diferentes distâncias especificadas. A função `Haralick` utiliza essas matrizes para calcular descritores texturais importantes, como entropia, homogeneidade e contraste. A entropia mede a aleatoriedade na textura, a homogeneidade avalia a uniformidade dos valores na matriz de co-ocorrência e o contraste mede a variação de intensidade. Cada descritor é calculado para todas as distâncias fornecidas e armazenado em uma lista de dicionários, cada um correspondente a uma distância específica.

2.1.4. Momentos de Hu

Os momentos invariantes de Hu são calculados para fornecer uma descrição da forma da imagem que é invariável a transformações geométricas como escala, rotação e translação. A função `Hu` converte a imagem para um array numpy e calcula os momentos até a terceira ordem. Calcula-se o centroide da imagem e, a partir disso, os momentos centrais são derivados. Esses momentos são então normalizados para formar os momentos de Hu. Os sete momentos de Hu resultantes capturam diferentes aspectos da forma da imagem, sendo invariáveis às transformações mencionadas, o que os torna úteis para reconhecimento de padrões e classificação de imagens.

2.2. parte2.py

No segundo arquivo, trata-se da implementação dos classificadores rasos e profundos, conforme solicitado no enunciado do trabalho.

2.2.1. Classificadores Rasos

Os classificadores rasos utilizados no código são baseados em descritores de Haralick e no algoritmo XGBoost. A função `ExtraiFeatures` é responsável por extrair as características das imagens usando descritores de Haralick. Para cada imagem, a função calcula as matrizes de co-ocorrência para diferentes distâncias e extrai três características principais: entropia, homogeneidade e contraste. Essas características são concatenadas em um vetor de características plano. Posteriormente, a função `ClassificadoresRasos` usa esses vetores de características para treinar dois modelos de XGBoost: um para classificação binária e outro para classificação multiclasse. O modelo de classificação binária é treinado para distinguir entre imagens negativas para lesão intraepitelial e outras

classes, enquanto o modelo multiclasse é treinado para classificar as imagens em todas as classes disponíveis. Os resultados de ambos os modelos são avaliados usando a acurácia e a matriz de confusão.

2.2.2. Classificadores Profundos

Os classificadores profundos são baseados na rede neural EfficientNet. A função `GerarDatasets` prepara os conjuntos de dados de treinamento e teste, padronizando as imagens para um tamanho uniforme e aplicando técnicas de aumento de dados. As imagens são redimensionadas para 224x224 pixels e normalizadas para o intervalo [0, 1]. A função `ClassificadorProfundo` treina dois modelos de EfficientNetB0: um para classificação binária e outro para classificação multiclasse. A função `GeraClassificador` cria o modelo de EfficientNetB0 com uma camada densa final adequada para o tipo de classificação (sigmoide para binária e softmax para multiclasse). Durante o treinamento, as camadas da EfficientNetB0 pré-treinada são congeladas, e apenas as camadas adicionadas são treinadas. A função `TreinaClassificador` realiza o treinamento do modelo, enquanto a função `AvaliaClassificador` avalia a performance do modelo no conjunto de dados de validação, retornando a acurácia e a matriz de confusão. Esses classificadores profundos são capazes de capturar características complexas das imagens, o que pode melhorar a precisão da classificação em comparação com os classificadores rasos.

2.3. main.py

Conforme seu nome, o arquivo main trata-se da implementação e chamada das funções dos dois tópicos anteriores dentro de uma interface gráfica, além da plotagem dos e histogramas resultantes das devidas funções.

3. Bibliotecas

O motivo principal da escolha da linguagem Python para a implementação desse projeto se deve às suas bibliotecas com funções bastante completas ao realizar análise e processamento de dados e imagens, além do uso para treinamento e teste de modelos de reconhecimento de imagem. Nos tópicos a seguir, apresentará quais bibliotecas foram usadas no código e qual sua determinada função:

3.1. Pillow (Python Imaging Library)

O PIL foi a biblioteca escolhida para a abertura, análise e aplicação de filtros de acordo com a tonalidade desejada (cinza e HSV) nas imagens escolhidas.

3.2. NumPY

Juntamente com o PIL, o NumPY foi a biblioteca de maior importância para o projeto, uma vez que é através deste que realiza-se a conversão das imagens para as matrizes e vetores de tonalidades necessárias para o processamento e análise dos métodos requeridos durante a primeira parte do trabalho, além da geração dos histogramas e de outros cálculos, como o somatório da matriz, calculo da normal e retorno do tamanho das matrizes.

3.3. Matplotlib

O Matplotlib tem como principal objetivo gerar os gráficos e histogramas dos resultados alcançados pelos métodos do projeto, possibilitando a adição de legendas, sobreposições de gráficos e ajustes no tamanho das fontes.

3.4. Pandas

No projeto, o Pandas serviu para realizar a etapa de leitura, recorte e separação das imagens através da leitura do arquivo .csv e obtenção dos valores importantes como a classe, posições x e y e nome do arquivo recortado.

3.5. TKinter

A interface padrão do Python foi utilizada para gerar o menu com as opções de cada um dos requisitos solicitados do trabalho, além de auxiliar na parte gráfica do projeto (com exceção na geração de gráficos e histogramas).

3.6. SciKit Learn

O calculo das matrizes de confusão e acurácia dos classificadores rasos e profundos foi realizado através das funções presentes no SciKit Learn.A biblioteca também teve importante função na geração dos classificadores binários (através da função LabelEncoder).

3.7. XGBoost

Conforme solicitado nos requerimentos do projeto, o XGBoost serviu na geração,treinamento e retorno dos resultados dos classificadores rasos.

3.8. Tensorflow

Toda a parte dos classificadores profundos foram realizados com essa biblioteca, desde a geração do modelo utilizando a EfficientNet (conforme requisitado no enunciado), até o armazenamento do histórico do treinamento por épocas e realização das predições.

3.9. OpenCV

Embora de grande uso no reconhecimento de imagens, o OpenCV no projeto foi utilizado apenas na padronização do tamanho das imagens utilizadas para treinamento e teste, uma vez que para o funcionamento do Tensorflow, era necessário que todas as imagens possuíssem a mesma dimensão.

3.10. Random

A biblioteca random teve papel crucial no embaralhamento e separação das imagens para teste e treino de acordo com o peso oferecido.

3.11. OS

A os teve seu papel na busca e criação de diretórios para a distribuição dos recortes das imagens de acordo com a sua classe.

4. Análise dos Resultados

4.1. Parte Um

Para a análise da primeira parte do trabalho, utilizará como parâmetro para análise a célula de número 4901, presente nas Figuras 1 e 2. Este recorte de imagem possui tamanho 100x100 e se trata de uma célula classificada como *"Negative for intraepithelial lesion"*.

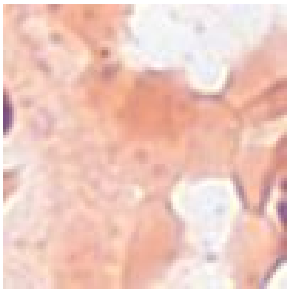

Figure 1. Original

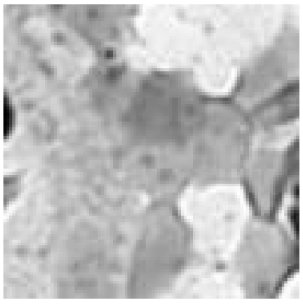

Figure 2. Tons de Cinza

As imagens 3 e 4 se tratarão dos histogramas para a imagem selecionada na tonalidade de cinza e ao ser aplicado dentro da tonalidade HSV. É possível observar pelo histograma que a imagem com a tonalidade cinza possui maior proximidade para o branco, evidenciando a necessidade de ajustes manuais para aumentar o contraste e melhorar a observação de características presentes.

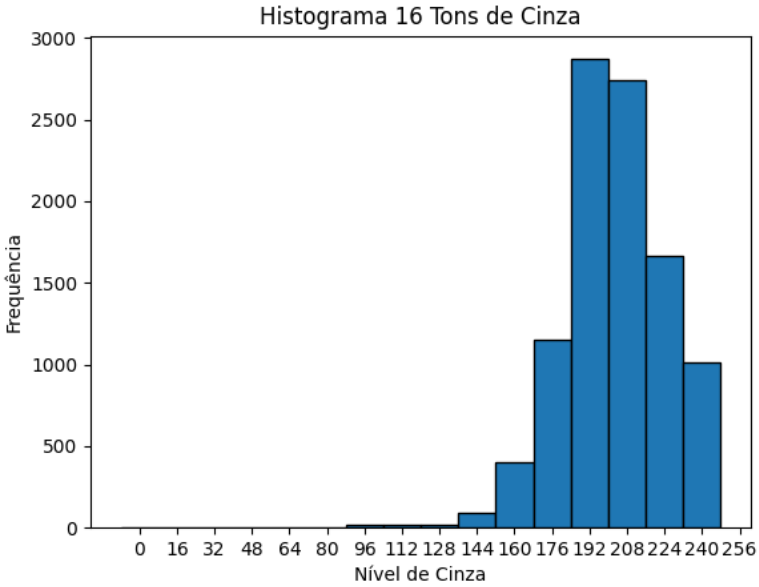

Figure 3. Histograma na Tonalidade de Cinza

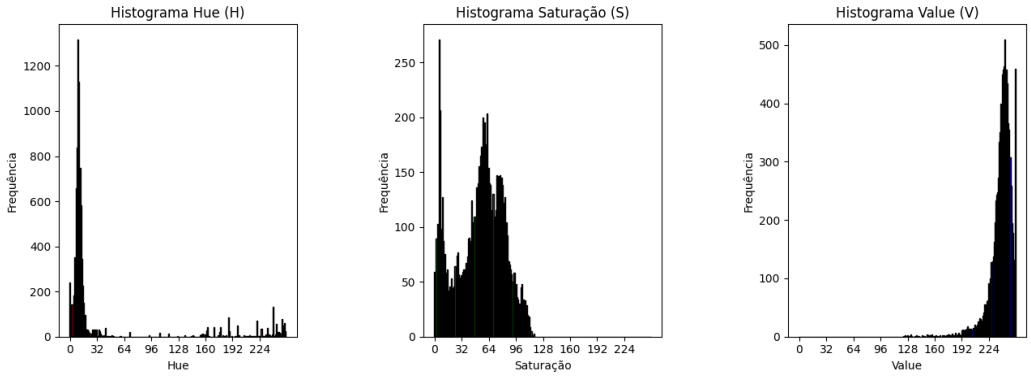

Figure 4. Histograma na tonalidade HSV

O histograma presente na figura 4 confirma uma necessidade de maiores ajustes na imagem. Neste caso, uma equalização nos três valores para diminuir a diferença entre as duas extremidades dos histogramas poderia ajudar na melhora da qualidade da imagem para análise, embora é necessário que haja uma certa parcimônia nessas alterações de forma a não causar perda de características na imagem.

A figura 5 representa o histograma 2D da imagem com base na tonalidade 2D. É possível observar toda a frequência da imagem presente em uma única região do gráfico, corroborando a análise com base nos histogramas apresentados anteriormente. Para a imagem na Escala de cinza, é possível observar essa predominância de uma tonalidade através das matrizes de Co-Ocorrência,presentes na figura 6, nas quais avaliam a frequência de tons de cinza de acordo com a distancia definida. Quanto maior a distancia entre os dois pixels, mais aglomerada a diferença entre tons de cinza.

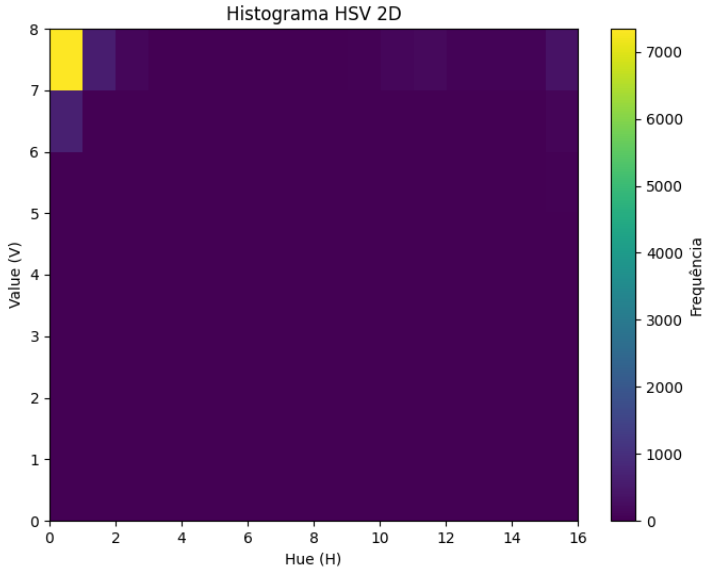

Figure 5. Histograma 2D

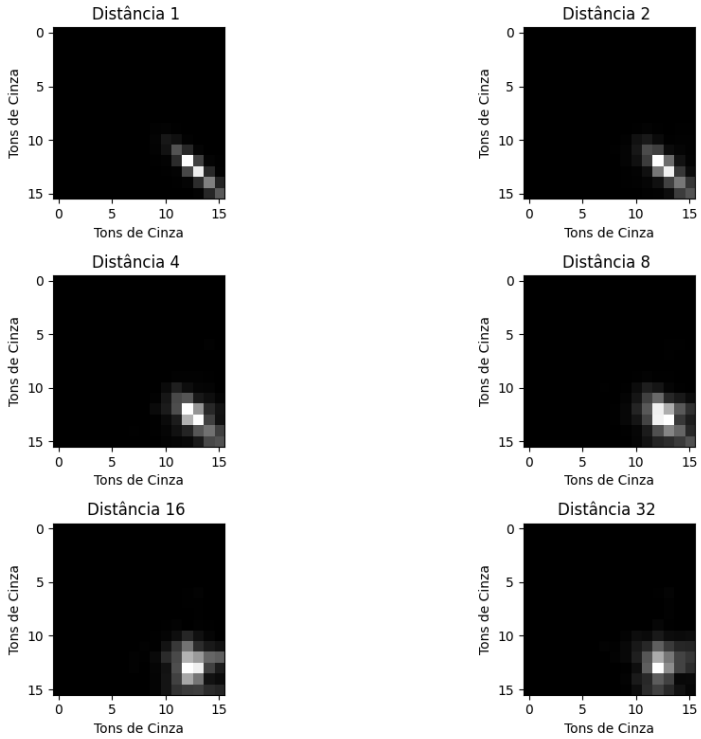

Figure 6. Matrizes de Co-Ocorrencia

A seguir, se apresentará os resultados obtidos pelos Descritores de Haralick, na figura 7. O cálculo deste foi feito com base nos resultados obtidos pelas matrizes de Co-Ocorrência,presentes na figura 6. Enquanto os valores de Entropia permanecem os mesmos ao aumentar a distancia, a homogeneidade diminui enquanto o contraste aumenta.

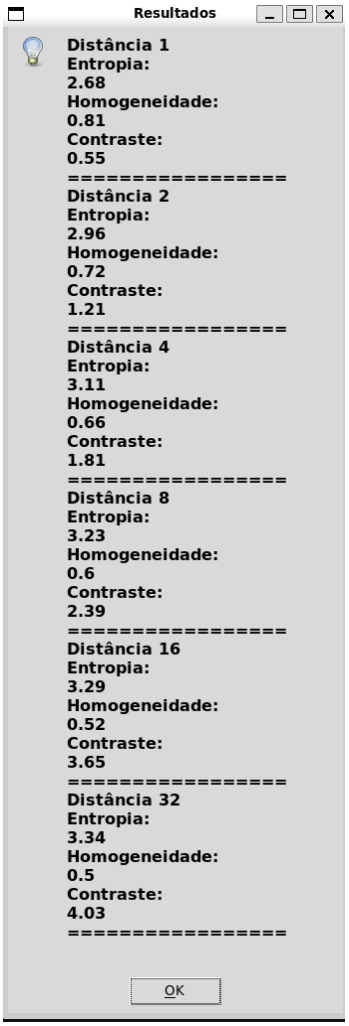

Figure 7. Descritores de Haralick com base na distancia

Por último nesse tópico, os momentos invariantes de Hu. Assim como nos histogramas, foi realizado o calculo para tanto na escala de cinza quanto na tonalidade HSV. Não há muita variação entre as duas, indicando nenhuma alteração em relação ao tamanho ou a rotação da célula.

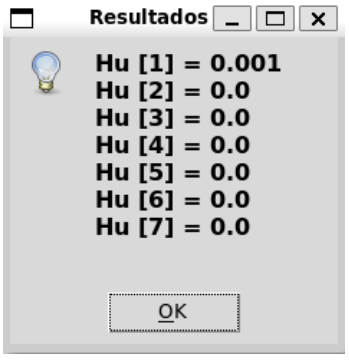

Figure 8. Momento de Hu na escala de Cinza

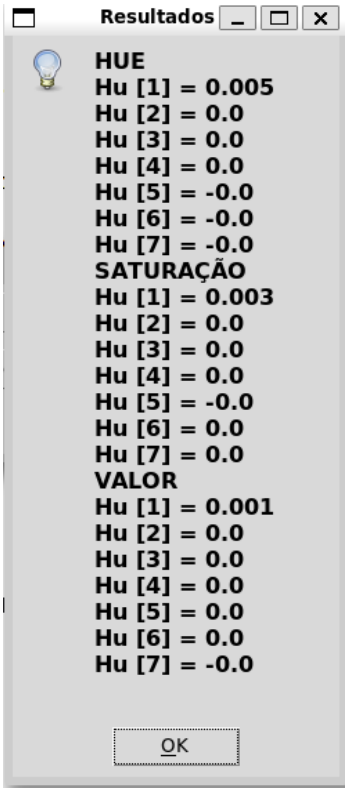

Figure 9. Momento de Hu na tonalidade HSV

4.2. Parte 2

Para a parte 2 do projeto consiste na criação de classificadores rasos e profundos,ambos com classificações únicas (ou binária) e multi classe. Para diminuir o tempo de execução do algoritmo e facilitar a depuração, utilizou-se um arquivo .csv com menos valores. Para o classificador raso, com o resultado na Figura 9 apresentou um tempo para treino e predição mais rápido, porém, mesmo com a acurácia entre as duas classificações bem próximas entre si, a matriz de confusão apresenta um valor considerável de falsos negativos, levando a crer que o classificador precisa de mais treinamento afim de diminuir a taxa de falsos negativos.

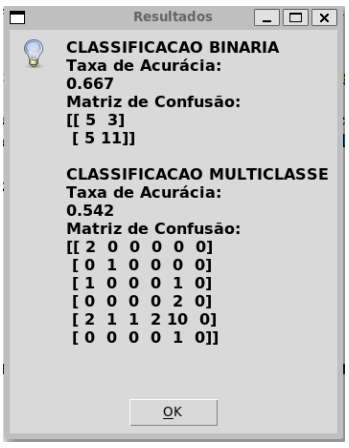

Figure 10. Resultado do Classificador Raso

Já para os classificadores profundos, o resultado,presente na figura 10, apresentou uma enorme disparidade entre os dois classificadores. De acordo com a matriz de confusão, o classificador raso chegou a ter uma taxa altíssima de falsos negativos, levando a entender que este necessita de um treinamento muito maior em comparação a todos os outros algoritmos presentes nessa parte. Porém, o classificador multi-classe apresentou um resultado próximo aos obtidos pelos classificadores rasos, com uma leve taxa de erro de falsos positivos, mas com uma matriz de confusão muito mais próxima do aceitável. A figura 11 apresenta um gráfico com a acurácia do classificador de acordo com a época (no caso do código foi definido como igual a 10).

Figure 11. Resultado do Classificador Profundo

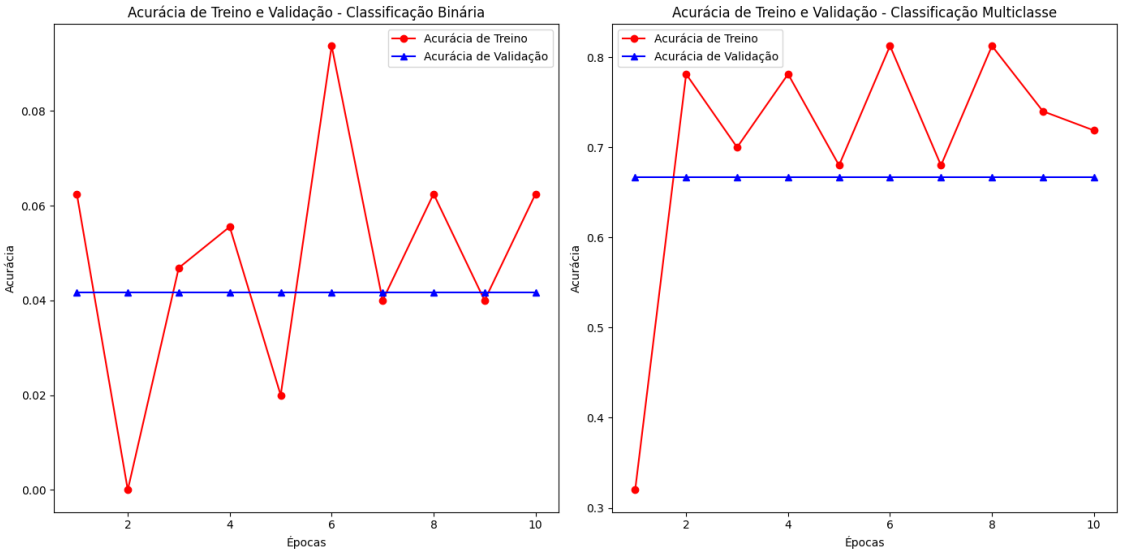

Figure 12. Comparação de Acurácia entre Épocas

Embora o algoritmo criado para esse projeto não apresente resultados ótimos e tenha-se usado uma entrada bem menor em comparação ao arquivo original, é possível

concluir que os classificadores rasos podem conseguir melhores resultados com a entrada de classificadores maiores, enquanto o classificador profundo, mesmo com acurácia próxima a dos rasos, obteve uma matriz de confusão muito mais aceitável, levando a concluir que este consegue bons resultados de predição com entradas menores, o que faz sentido ao levar em consideração que os classificadores profundos utilizam o conceito de épocas para poder reprocessar os dados da entrada repetidadmente antes de gerar o output ou predição.

5. Bibliografias Utilizadas

- https://pandas.pydata.org/docs/reference/api/pandas.read _csv.html
- https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/
- https://gogulilango.com/software/texture-recognition
- https://scikit-image.org/skimage-tutorials/lectures/00_images_are_arrays.html
- https://stackoverflow.com/questions/22236956/rgb-to-hsv-via-pil-and-colorsys
- https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_glcm.html
- https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/
- https://stackoverflow.com/questions/33203645/how-to-plot-a-histogram-using-matplotlib-in-python-with-a-list-of-data
- https://python-graph-gallery.com/83-basic-2d-histograms-with-matplotlib/
- https://learnopencv.com/shape-matching-using-hu-moments-c-python/
- Löfstedt T, Brynolfsson P, Asklund T, Nyholm T, Garpebring A. Gray-level invariant Haralick texture features. PLoS One. 2019 Feb 22;14(2):e0212110. doi: 10.1371/journal.pone.0212110. PMID: 30794577; PMCID: PMC6386443.