Segmentação e classificação de imagens mamográficas

Lucas Saliba[1], **Ygor Matheus Lacerda de Melo**[2]

[1]Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Av. Dom José Gaspar, 500 – 30.535-610 – Belo Horizonte – MG – Brazil

1. Introdução

O câncer de mama é uma das doenças mais prevalentes entre mulheres em todo o mundo, superando o câncer de pulmão em número de casos, de acordo com a Organização Mundial da Saúde (OMS) em 2020. A densidade mamária pode ser um indicador de risco para o desenvolvimento do câncer de mama, pois mulheres com maior densidade podem ter lesões ocultas que podem ser indicativas da doença. Infelizmente, essas lesões geralmente são descobertas tardiamente, resultando em um agravamento do problema. Portanto, é crucial detectar o câncer o mais cedo possível, pois isso aumenta significativamente as chances de cura.

A densidade da mama está diretamente relacionada ao risco de desenvolvimento do câncer. Mulheres com maior densidade mamária podem ter lesões que passam despercebidas, levando a um diagnóstico tardio da doença. O American College of Radiology desenvolveu uma escala de densidade chamada BIRADS, que fornece informações aos radiologistas sobre a diminuição da sensibilidade dos exames à medida que a densidade da mama aumenta.

A motivação deste trabalho é realizar uma análise visual das imagens disponíveis resultantes de exames de mamografia, que é a principal ferramente de rastreamento do câncer de mama. É essencial que a densidade mamária seja levada em consideração durante a avaliação, a fim de garantir uma detecção precoce e um tratamento adequado para o câncer de mama.

O objetivo deste trabalho é o desenvolvimento de um software que seja capaz de treinar uma rede neural através de imagens mamográficas a fim de ser capaz de realizar a classificação da imagem por tipo de BIRADS. Para obtermos um resultado satisfatório, é necessário a alta precisão de detecção da rede neural, diminuindo as taxas de erro e aumentando as taxas de acerto.

O restante deste artigo está organizado da seguinte forma: a Seção 2 trata da descrição de implementação do software e informações de bibliotecas utilizadas; a Seção 3 lista as técnicas implementadas para realização da classificação e método de segmentação das imagens; a Seção 4 discute os resultados obtidos; a Seção 6 mostra o relatório do Github e, por fim, a Seção 5 as referências bibliográficas.

2. Descrição do Projeto

O projeto foi desenvolvido utilizando a linguagem Python (versão 3.10.2 disponível para download no site oficial) e o Visual Studio Code como IDE. A seguir, forneceremos uma breve descrição sobre as bibliotecas aplicadas em nosso projeto.

2.1. Bibliotecas utilizadas

A biblioteca 'Tkinter' é uma biblioteca que já vem com a instalação do Python, utilizamos na criação da interface e importamos a função "filedialog". Para auxiliar no processamento das imagens e execução de tarefas de visão computacional, utilizamos a biblioteca 'CV2' do OpenCv, além da função "ImageTk"da biblioteca 'Pillow' que realiza a junção das bibliotecas 'Pillow' e 'Tkinter'. A biblioteca 'Numpy' também foi importada para auxiliar na manipulação das imagens. Já a biblioteca 'OS', é um módulo que auxilia na criação de scripts e a biblioteca 'TQDM' utilizamos para adicionar barras de progresso.

```
import cv2
import PIL
import PIL.ImageTk
import tkinter
import tkinter.filedialog
import os
import time
import tqdm
import numpy as np
import tensorflow as tf

from sklearn.metrics import confusion_matrix
from seaborn import heatmap

import matplotlib
```

Figura 1. Bibliotecas utilizadas no desenvolvimento da aplicação

A biblioteca 'Time' tem o objetivo de medir o tempo de execução do treinamento e da classificação da rede neural. Se tratando da classificação, esta medição é utilizada tanto para medir o tempo de execução da classificação da imagem completa quanto da classificação de uma região específica selecionada pelo usuário. Utilizamos a função Keras da biblioteca de rede 'TensorFlow', voltada para o aprendizado de máquina, com o objetivo de criar e treinar a rede neural para detectar os padrões e classificar as imagens no tipo de BIRADS pertencente. Por fim, a biblioteca 'Mathplolib' utilizamos para a matriz de confusão.

3. Técnicas Implementadas

Ao iniciar o software, deve-se selecionar a imagem para ser carregada no projeto, é permitido apenas a leitura de imagens no formato PNG e TIFF. Após realizar a abertura da imagem, é permitido ao usuário que aplique zoom, altere o contraste e realize a segmentação através de uma barra deslizante selecionando o valor de tons de cinza correspondente.

No diretório 'mamografia' estão presentes as imagens utilizadas para realizar o treinamento dos classificadores de BIRADS da nossa rede neural. Realizamos a leitura da base através da função 'read_data', separando as imagens de treino e de teste. Ao realizamos a leitura de cada imagem, a função 'flip_data' realiza o espelhamento e a função 'equalize_data' a equalização do seu histograma, logo, para cada imagem, é gerado 4 variações: original e espelhada X original e equalizada.

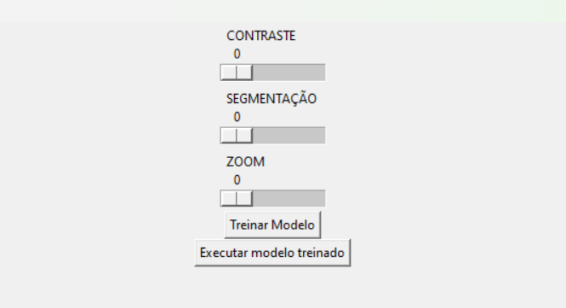


Figura 2. Janela de visualização ao inciar o software

Ao realizar a leitura das imagens, a função 'apply_threshold' aplica a segmentação binária de forma automática nas mesmas, transformando os elementos de fundo e anotações com valor preto e a imagem da mama de branco.

```
167  # Segmentacao automatica de imagens do diretorio
168  def apply_threshold(X_array: list) -> list:
169      X_transformed = []
170      for X in X_array:
171          # Aplicacao de segmentacao binaria em cada item recebido
172          X_transformed.append(cv2.threshold(X, 7, 255, cv2.THRESH_BINARY)[1])
173
174      return X_transformed
```

Figura 3. Função apply_threshold que realiza a segmentação

Na função da segmentação binária acima, inicialmente testamos o valor do kernel igual a 15, porém a região de dentro da mama ainda possuía algumas variações de tons de cinza, impossibilitando uma identificação com clareza. Ao realizar testes, definidos que o melhor valor para o kernel é defini-lo igual a 7, gerando a segmentação da imagem com menos ruido e melhor contorno das mamas.

Realizamos o aumento de dados das imagens de treino via espelhamento e equalização de histogramas salvando no mesmo array (x_train) a fim de ter uma melhor praticidade e facilidade nos testes.

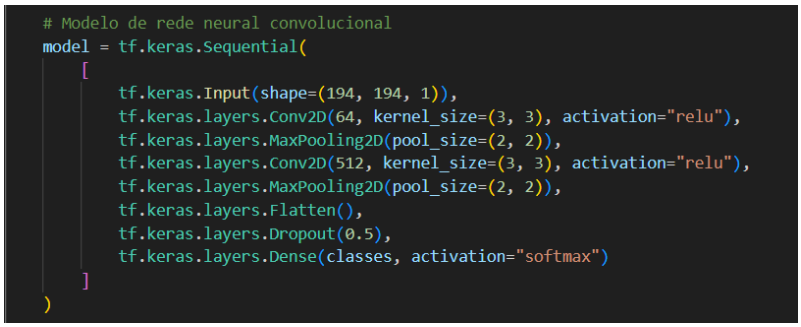
Na tentativa de implementar a rede neural ConvNext nos deparamos com algumas dificuldades e problemas de compatibilidade com nosso dataset. A princípio, tentamos usar ela com o Hugging Face porém não conseguimos pois ele já trabalha com um tipo de dataset deles mesmo, seguindo um diferente padrão.

A ConvNext foi pré-treinada com imagens padrão de 224x224 pixels e por conta disso a sua arquitetura já estava projetada para isso. Mesmo tentando fazer um rezise de nossas imagens, estávamos tendo um problema com a classificação. Ao tentar usar a ConvNext do Keras, se limitarmos o número de classes para 4, diferente da original da ConvNext que é 1000, não podíamos alterar a primeira camada da rede e pelo visto isso estava dando algum problema no shape dos dados, a forma em que eles estavam entrando. Ou seja, se alterasse a primeira camada não poderíamos mexer no número de classes, estando isso escrito na documentação do Keras, assim não sendo possível treinar com nosso dataset com a quantidade de classes que gostaríamos.

```python
# Aplicacao da rede neural ConvNext
def conv_next(X_train: np.array, X_test: np.array, y_train: np.array, y_test: np.array) -> None:
    model = tf.keras.applications.ConvNeXtBase(
        model_name="convnext_base",
        include_top=False,
        include_preprocessing=False,
        weights=None,
        input_tensor=tf.keras.layers.Input(shape=(224, 224, 1)),
        input_shape=None,
        pooling=None,
        classes=4,
        classifier_activation="softmax"
    )

    print('convnext')

    model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

    print('compile')

    model.fit(X_train, y_train, verbose=1)

    print('fit')
```

Figura 4. Implementação da rede neural ConvNext

Após as tentativas da implementação da rede ConvNext, optamos pela implementação da rede neural convolucional, em que conseguimos realizar a classificação binária (I+II x III+IV) e a classificação de 4 classes (IxIIxIIIxIV). A imagem 5 mostra os diferentes hiperparâmetros utilizados.



```python
# Modelo de rede neural convolucional
model = tf.keras.Sequential(
    [
        tf.keras.Input(shape=(194, 194, 1)),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(512, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(classes, activation="softmax")
    ]
)
```

Figura 5. Implementação da rede neural convolucional

Inicialmente, implementamos um modelo com três camadas contendo 32, 64 e 128 neurônios, respectivamente. Em seguida, realizamos uma ampliação para 64, 128 e 256 neurônios, mantendo as três camadas. Posteriormente, aumentamos para 128, 256 e 512 neurônios em cada camada, entretanto, notamos que não houve um avanço significativo na acurácia da classificação das quatro classes.

Percebemos que utilizando 3 camadas e aumentando a quantidade de neurônios, tínhamos um grande ganho na acurácia da classificação binaria, porém a classificação de 4 classes estava sendo muito prejudicada. Além disso, começamos a enfrentar o risco de overfitting, onde o modelo estava se ajustando excessivamente aos dados de treinamento, prejudicando sua capacidade de generalização.

Como estratégia para superar essa limitação, decidimos reduzir uma camada ao modelo, com neurônios distribuídos nas quantidades de 128 e 256. Essa modificação nos trouxe o melhor resultado nos testes realizados, permitindo alcançar um desempenho mais eficiente na classificação das imagens. Percebemos que mesmo utilizando apenas 2 camadas, quanto mais aumentávamos os valores de seus neurônios melhor era a acurácia da

classificação binaria e pior a de 4 classes. Ao rodar com 128 e 512 neurônios em cada camada respectivamente, obtivemos bons resultados, com ambos acima de uma acurácia de 55%. Entretanto, ao realizar os treinos vimos que os valores estavam muito instáveis, eles poderia ter uma grande variação ao executar o mesmo treino com os mesmos parâmetros.

Então realizamos a seguinte estrategia: removemos 15px de laterais da imagem, removendo bordas e ruídos indesejáveis e executamos a rede neural com 2 camadas com 64 e 512 neurônios cada. Assim, obtivemos bons resultados, ambos acima de 50% de acurácia e estáveis, passando mais segurança ao executar o dataset por completo. Com essa nova configuração de camadas e neurônios, conseguimos melhorar a capacidade de representação do modelo, resultando em uma maior precisão na classificação das quatro categorias do BIRADS.

3.1. Medidas

Para de iniciarmos a medição do tempo de execução do algoritmo, realizamos uma análise do custo na resolução do nosso problema, que consiste no treinamento e classificação de imagens. Com base nessa análise, teremos o número de vezes que cada etapa do algoritmo foi executada.

3.2. Descritores

No nosso projeto, utilizamos a matriz de confusão, na classificação das categorias de imagens de acordo com o BIRADS, ela irá nos ajudar a entender como o modelo está realizando a classificação para cada categoria. A matriz de confusão nos permite visualizar não apenas os acertos do modelo em cada categoria, mas também os erros de classificação.

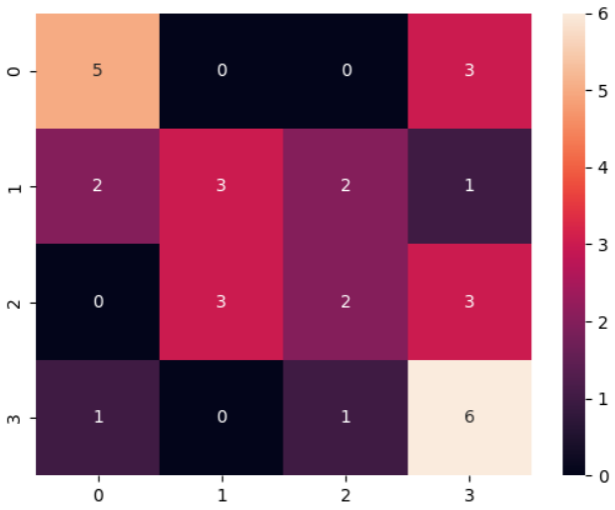


Figura 6. Matriz de Confusão

3.3. Hiperparâmetros do classificador

O resultado obtido foi baseado em uma métrica que avalia a precisão do modelo, utilizando a acurácia e a perda (loss). Durante o processo de treinamento, foram utilizados

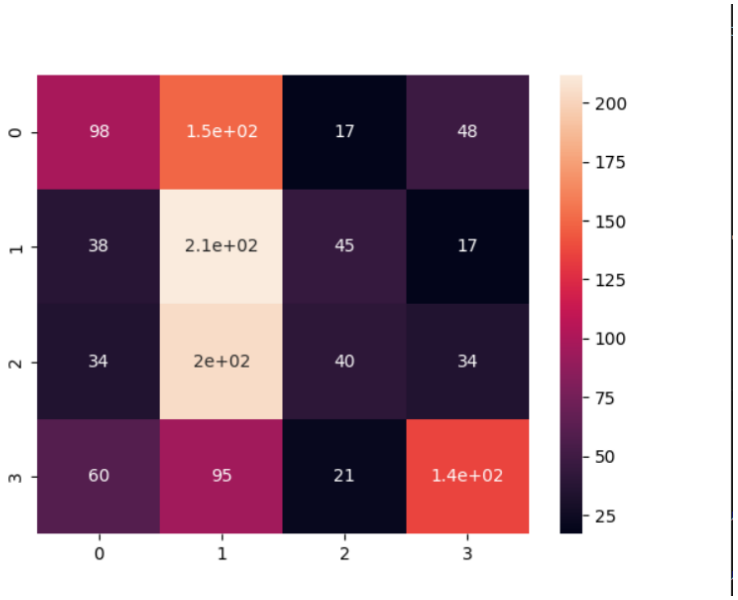


Figura 7. Matriz de Confusão

múltiplos ”epochs”, que representam os ciclos completos em que a rede neural percorre todos os dados de treinamento. A fim de otimizar o treinamento, definimos o tamanho do lote (batch size) da rede neural como 16, permitindo o processamento eficiente dos dados.

Nosso modelo foi projetado com duas camadas. A primeira camada possui 64 neurônios, a segunda camada contém 512 neurônios. Essa configuração de camadas e neurônios permite uma representação mais complexa e expressiva dos dados, melhorando o desempenho e a capacidade de aprendizado do modelo.

Além disso, utilizamos 1024 nós ou neurônios para testar as bases de dados. Essa escolha foi feita com o objetivo de garantir uma capacidade de processamento adequada para a análise dos dados e a obtenção dos melhores resultados. Ao combinar essa configuração com os hiperparâmetros ajustados durante o treinamento, conseguimos alcançar um desempenho otimizado e um modelo bem ajustado às características das bases de dados.

4. Resultados obtidos

Ao realizarmos o treinamento e classificação da base de dados, levamos em consideração os critérios de precisão e eficiência de processamento. O tempo total para rodar a base de dados foi de 7horas e 30minutos.

Durante a execução do treinamento, é mostrado o tempo após cada medida do epoch

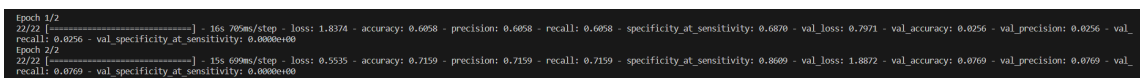


Figura 8. Visualização dos valores durante a execução de cada epoch

Após o processo de treinamento, o algoritmo armazena o modelo encontrado em

um arquivo chamado ”conv”. É exibido na janela de execução, os valores referentes a classificação binária e da classificação de 4 classes como mostrado na figura abaixo:

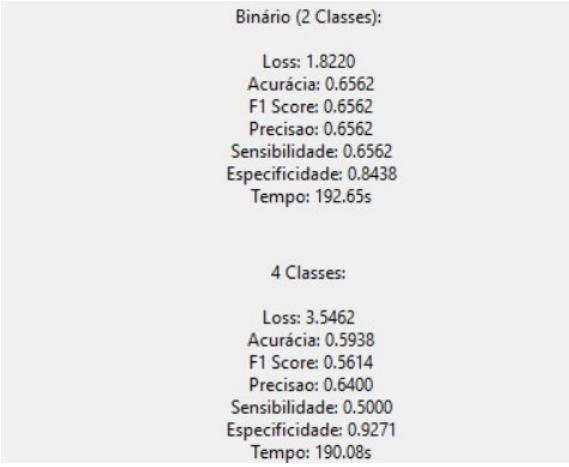


Figura 9. Visualização dos valores do processo de treinamento e classificação

5. Referências Bibliográficas

A seguir, as referências que ajudaram o desenvolvimento do trabalho.

https://www.tensorflow.org/api_docs

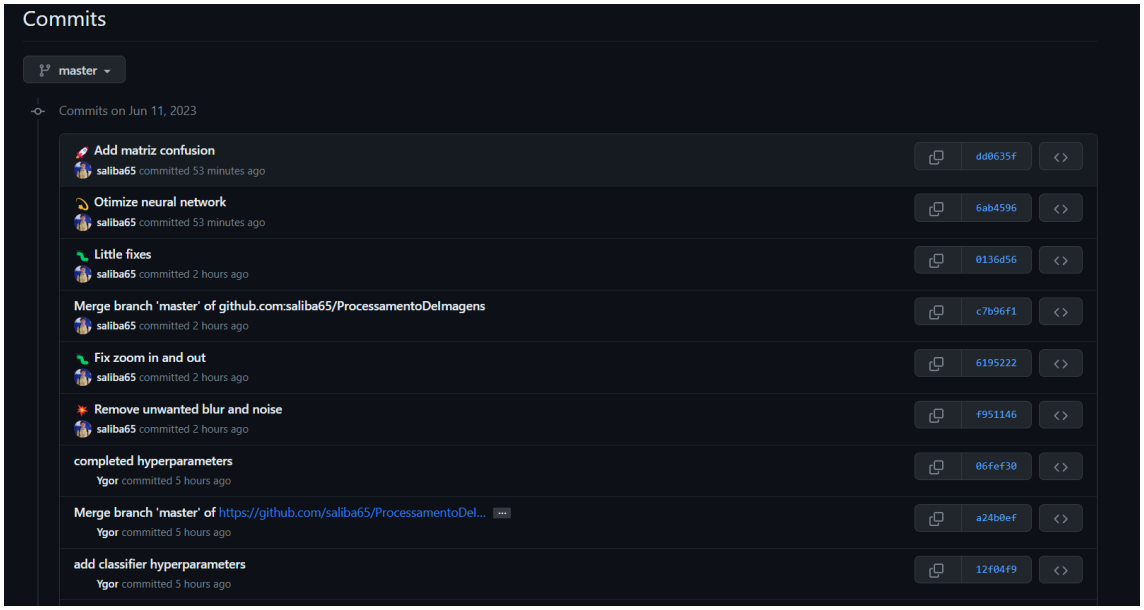
https://docs.python.org/3/

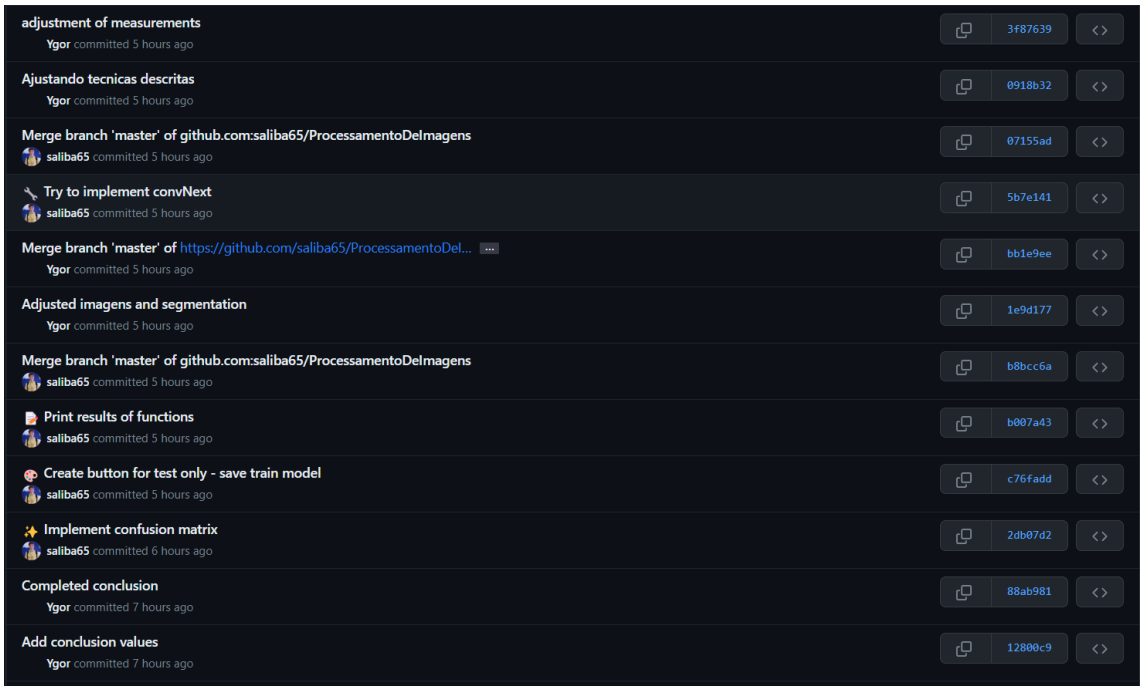
https://docs.opencv.org/4.x/

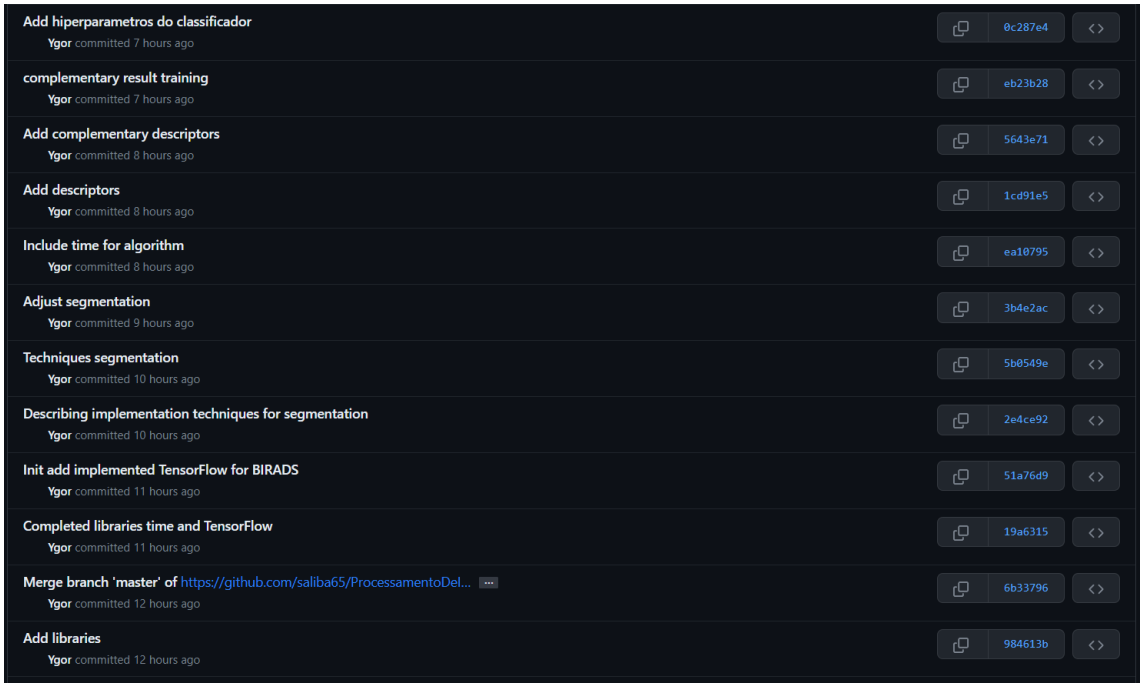
https://huggingface.co/docs/transformers/model_doc/convnext

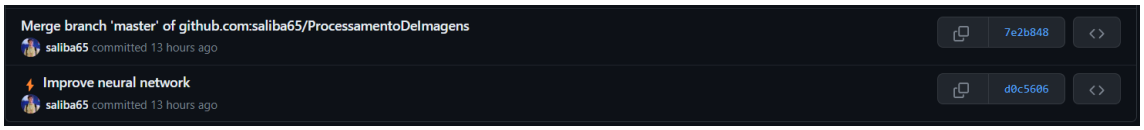
https://keras.io

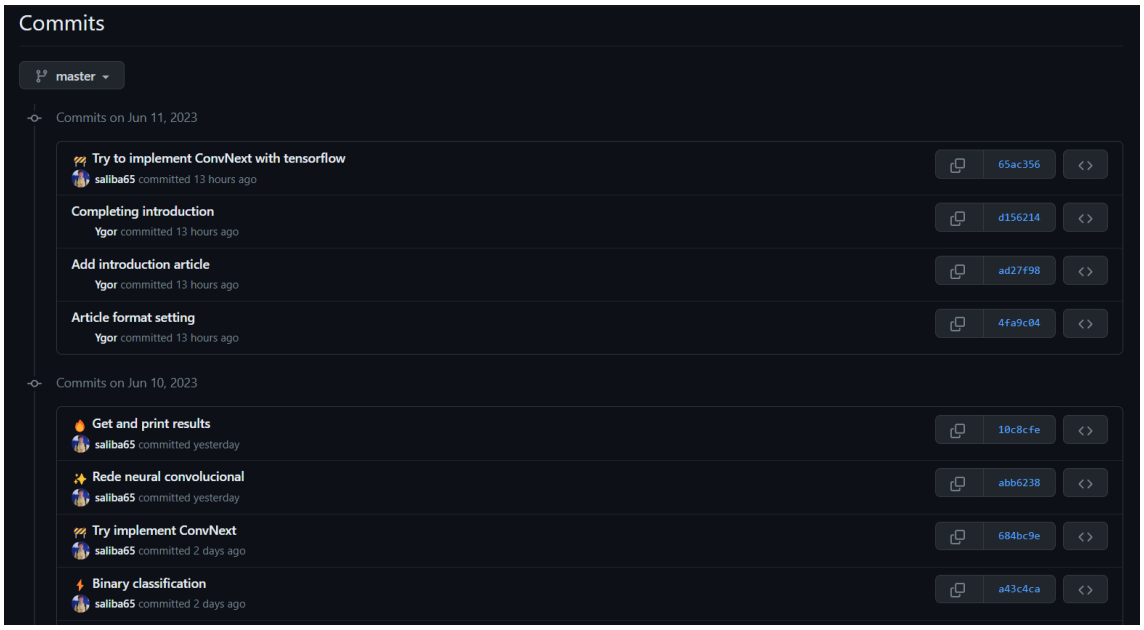
6. Relatório de commits

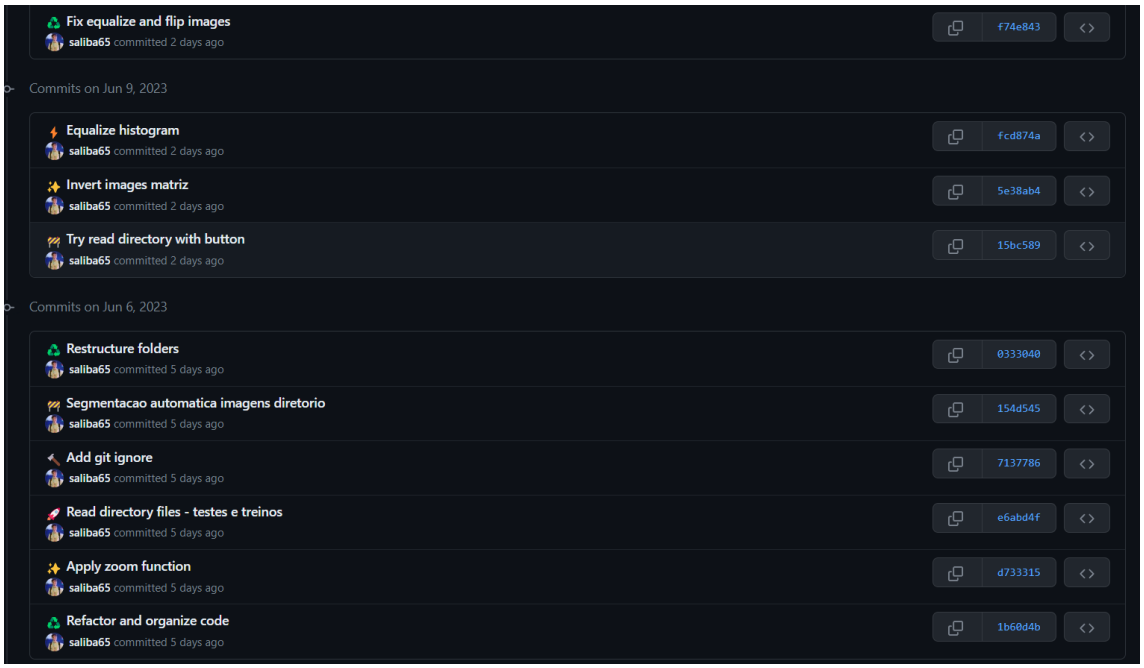


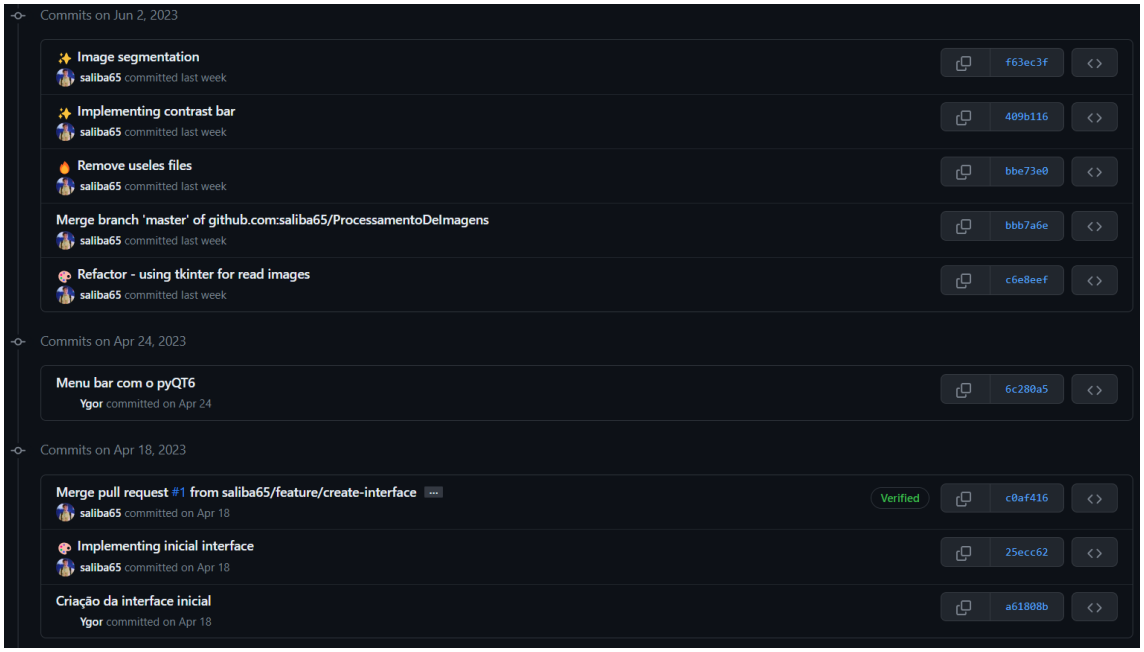

adjustment of measurements
Ygor committed 5 hours ago
3f87639
Ajustando tecnicas descritas
Ygor committed 5 hours ago
0918b32
Merge branch 'master' of github.com:saliba65/ProcessamentoDeImagens
saliba65 committed 5 hours ago
07155ad
Try to implement convNext
saliba65 committed 5 hours ago
5b7e141
Merge branch 'master' of https://github.com/saliba65/ProcessamentoDel...
Ygor committed 5 hours ago
bb1e9ee
Adjusted imagens and segmentation
Ygor committed 5 hours ago
1e9d177
Merge branch 'master' of github.com:saliba65/ProcessamentoDeImagens
saliba65 committed 5 hours ago
b8bcc6a
Print results of functions
saliba65 committed 5 hours ago
b007a43
Create button for test only - save train model
saliba65 committed 5 hours ago
c76fadd
Implement confusion matrix
saliba65 committed 6 hours ago
2db07d2
Completed conclusion
Ygor committed 7 hours ago
88ab981
Add conclusion values
Ygor committed 7 hours ago
12800c9

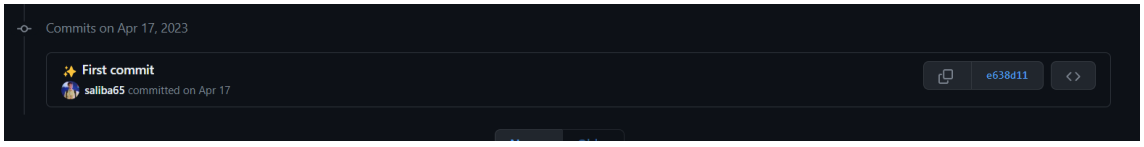

Add hiperparametros do classificador
Ygor committed 7 hours ago
0c287e4
complementary result training
Ygor committed 7 hours ago
eb23b28
Add complementary descriptors
Ygor committed 8 hours ago
5643e71
Add descriptors
Ygor committed 8 hours ago
1cd91e5
Include time for algorithm
Ygor committed 8 hours ago
ea10795
Adjust segmentation
Ygor committed 9 hours ago
3b4e2ac
Techniques segmentation
Ygor committed 10 hours ago
5b0549e
Describing implementation techniques for segmentation
Ygor committed 10 hours ago
2e4ce92
Init add implemented TensorFlow for BIRADS
Ygor committed 11 hours ago
51a76d9
Completed libraries time and TensorFlow
Ygor committed 11 hours ago
19a6315
Merge branch 'master' of https://github.com/saliba65/ProcessamentoDel...
Ygor committed 12 hours ago
6b33796
Add libraries
Ygor committed 12 hours ago
984613b


Merge branch 'master' of github.com:saliba65/ProcessamentoDeImagens
saliba65 committed 13 hours ago
7e2b848
Improve neural network
saliba65 committed 13 hours ago
d0c5606


Commits
master
Commits on Jun 11, 2023
Try to implement ConvNext with tensorflow
saliba65 committed 13 hours ago
65ac356
Completing introduction
Ygor committed 13 hours ago
d156214
Add introduction article
Ygor committed 13 hours ago
ad27f98
Article format setting
Ygor committed 13 hours ago
4fa9c04
Commits on Jun 10, 2023
Get and print results
saliba65 committed yesterday
10c8cfe
Rede neural convolucional
saliba65 committed yesterday
abb6238
Try implement ConvNext
saliba65 committed 2 days ago
684bc9e
Binary classification
saliba65 committed 2 days ago
a43c4ca


Fix equalize and flip images
saliba65 committed 2 days ago
f74e843
Commits on Jun 9, 2023
Equalize histogram
saliba65 committed 2 days ago
fcd874a
Invert images matriz
saliba65 committed 2 days ago
5e38ab4
Try read directory with button
saliba65 committed 2 days ago
15bc589
Commits on Jun 6, 2023
Restructure folders
saliba65 committed 5 days ago
0333040
Segmentacao automatica imagens diretorio
saliba65 committed 5 days ago
154d545
Add git ignore
saliba65 committed 5 days ago
7137786
Read directory files - testes e treinos
saliba65 committed 5 days ago
e6abd4f
Apply zoom function
saliba65 committed 5 days ago
d733315
Refactor and organize code
saliba65 committed 5 days ago
1b60d4b


Commits on Jun 2, 2023
Image segmentation
saliba65 committed last week
f63ec3f
Implementing contrast bar
saliba65 committed last week
409b116
Remove useles files
saliba65 committed last week
bbe73e0
Merge branch 'master' of github.com:saliba65/ProcessamentoDeImagens
saliba65 committed last week
bbb7a6e
Refactor - using tkinter for read images
saliba65 committed last week
c6e8eef
Commits on Apr 24, 2023
Menu bar com o pyQT6
Ygor committed on Apr 24
6c280a5
Commits on Apr 18, 2023
Merge pull request #1 from saliba65/feature/create-interface
saliba65 committed on Apr 18
Verified
c0af416
Implementing inicial interface
saliba65 committed on Apr 18
25ecc62
Criação da interface inicial
Ygor committed on Apr 18
a61808b


Commits on Apr 17, 2023
First commit
saliba65 committed on Apr 17
e638d11