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.

TODO: Conferir numeração abaixo

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 5 mostra o relatório do Github e, por fim, a Seção 6 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 neural '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.

TODO: Conferir se foi utilizada mais alguma biblioteca

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.

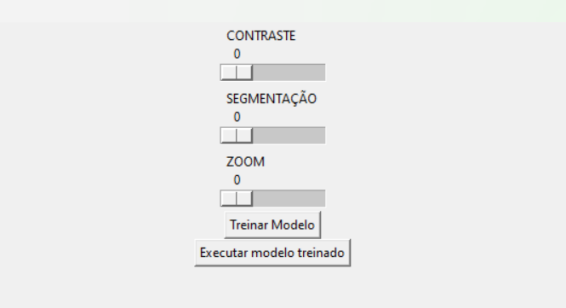


Figura 2. Janela de visualização ao inciar o software

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.

```
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 possuía muito ruído, impossibilitando dessa forma identificar com clareza as bordas. 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 precisão maior durante os testes.

TODO: Descrever aqui a dificuldade de montar a rede neural ConvNext, as funções selecionadas na imagem 4

EXEMPLO: Na tentativa de implementar a rede neural ConvNext nos deparamos com algumas dificuldades e problemas de compatibilidade com nossa rede.

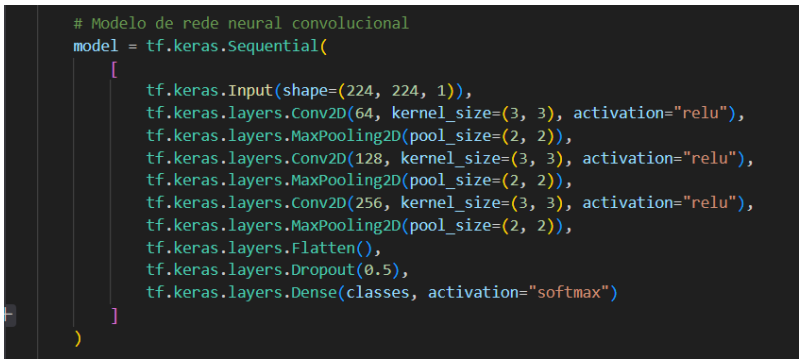
TODO: Conferir texto abaixo.

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.

Inicialmente, implementamos um modelo com duas camadas contendo 32 e 64 neurônios, respectivamente. Em seguida, realizamos uma ampliação para 64 e 128

```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



```python
# Modelo de rede neural convolucional
model = tf.keras.Sequential(
    [
        tf.keras.Input(shape=(224, 224, 1)),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(256, 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

neurônios, mantendo as duas camadas. Posteriormente, aumentamos para 128 e 256 neurônios em cada camada, entretanto, notamos que não houve um avanço significativo na acurácia da classificação das quatro classes. 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 adicionar mais uma camada ao modelo, com neurônios distribuídos nas quantidades de 64, 128 e 256. Essa modificação nos trouxe o melhor resultado nos testes realizados, permitindo alcançar um desempenho mais eficiente na classificação das imagens. 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

TODO: Conferir texto abaixo.

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

TODO: verificar texto

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.

TODO: adicionar print da matriz de confusão

TODO: adicionei essa parte abaixo mas como não vi o que a matriz de confusão mostra, de repente seja melhor tirar caso não tenha esses valores descritos

Com base nos valores presentes na matriz, podemos calcular métricas como a precisão, o recall e a taxa de erro do modelo para cada categoria. Essas métricas fornecem uma compreensão mais abrangente do desempenho do modelo.

3.3. Hiperparâmetros do classificador

TODO: Verificar texto abaixo.

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 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 três camadas. A primeira camada possui 64 neurônios, a segunda camada contém 128 neurônios e a terceira camada é composta por 256 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.

TODO: EXEMPLO DE TEXTO - RETIRAR Para efeitos comparativos, também realizamos medições de tempo em minutos. Essas medidas nos auxiliaram na análise comparativa do desempenho dos modelos em relação ao tempo de processamento. Por fim, dividimos a base de dados em quatro classes, correspondendo aos quatro tipos de BIRADS. Essa divisão nos permitiu uma análise mais precisa e específica durante o treinamento e avaliação do modelo, considerando as diferentes características e padrões presentes em cada classe de BIRADS.

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 TODO: Adicionar tempo.

TODO: Conferir o texto abaixo. Durante o treinamento, realizamos diversos testes (medidas de cada epoch?)

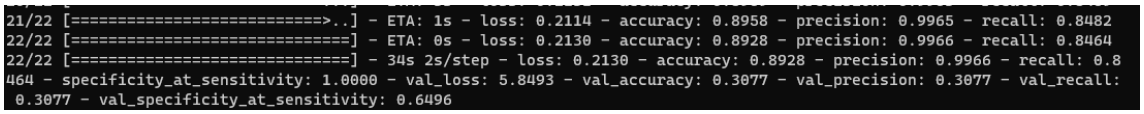


Figura 6. Visualização dos valores durante a execução de cada epoch

TODO: EXEMPLO: Geramos também uma matriz de confusão da base de dados (imagem a seguir) e como visto, os BIRADS 1 e 2 estão sendo mais reconhecidos do que os BIRADS 3 e 4.

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 (I+II x III+IV) e da classificação de 4 classes (IxIIxIIIxIV) como mostrado na figura abaixo:

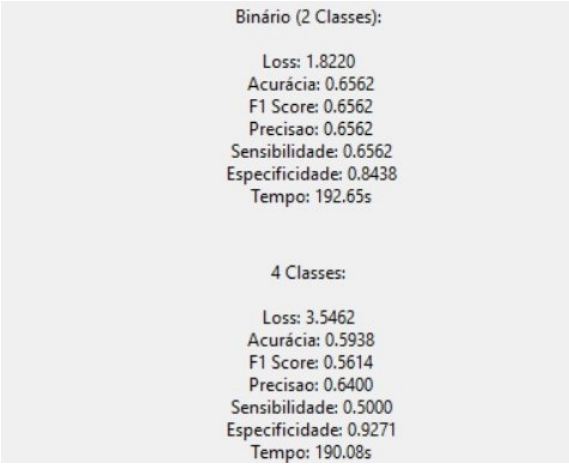


Figura 7. Visualização dos valores do processo de treinamento e classificação

TODO: Descrever as métricas.

5. Relatório

6. Referências Bibliográficas

Referências