PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e Informática

Darlan Francisco Gandos da Cunha
Kaiser Gabriel Silvério Batista dos Santos
Larissa Leite Matias

PROCESSAMENTO DE IMAGEM:

Trabalho Prático - Reconhecimento de padrões por textura em imagens mamográficas

Belo Horizonte
2021

Larissa Leite Matias
Darlan Francisco Gandos da Cunha
Kaiser Gabriel Silvério Batista dos Santos

PROCESSAMENTO DE IMAGEM:

Trabalho Prático - Reconhecimento de padrões por textura em imagens mamográficas

Trabalho prático com o intuito de praticar os conhecimentos apresentados e adquiridos durante as aulas da disciplina de Processamento de Imagens, do curso de Ciência de Computação da Pontifícia Universidade Católica de Minas Gerais.

Professor: Alexei Manso Correa Machado

Belo Horizonte
2021

Sumário

5. **CONCLUSÃO**

6. **BIBLIOGRAFIA**

1. INTRODUÇÃO

O objetivo era construir um software para reconhecimento de padrões por textura em imagens mamográficas com interface gráfica, com os conceitos obtidos na disciplina Processamento de Imagens.

Foi construído um software na linguagem de programação Python 3 e é capaz de abrir imagens nos formatos Portable Network Graphics (.PNG) e Tagged Image File Format (.TIFF). Ao abrir a imagem, permite aplicar e remover o zoom para selecionar uma área de interesse pelo usuário, recortar regiões, reduzir a resolução da área recortada e aplicar escalas de tons de cinza. Com a área de interesse selecionada o programa apresenta opções para alteração de padrão como quantização e equalização.

Foi desenvolvido algumas features do Haralick para ser aplicado como descritor de textura, e após treinarmos uma rede neural é exibido os resultados da acurácia e especificidade. Assim dando a opção para o usuário analisar a área selecionada, exibindo a entropia, homogeneidade e contraste.

2. IMPLEMENTAÇÃO

2.1. Ambiente de desenvolvimento

O software foi desenvolvido utilizando os sistemas operacionais Linux Ubuntu (versão 20.04), Microsoft Windows 10 e Mac OS (Big Sur), linguagem de programação Python 3 (versão 3.8), gerenciador de pacotes Pip (versão 21.1.1) para instalação das bibliotecas utilizadas.

Para conseguir executar o projeto não é necessário instalar o Python 3, pois ele vem junto com o Ubuntu 20.04 e outras distros Debian. Caso esteja executando o projeto em um sistema operacional diferente, acesse o site Python.org e siga a documentação para instalá-lo. Para instalar o Pip, execute o comando *sudo apt install -y python3-pip* no seu terminal. Também é necessário utilizar um editor de códigos, nós utilizamos e recomendamos o VS Code, abra o site e instale-o.

2.2. Bibliotecas utilizadas

As bibliotecas utilizadas para o desenvolvimento deste software foram:

- Tkinter (versão 3.9.5)
- Matplotlib (versão 3.4.2)
- NumPy (versão 1.20.0)
- Opencv-python (versão 4.5.2.52)
- Mahotas (versão 1.4.3)

- Scikit-learn (versão 0.24.2)
- argparse (versão 3.2)
- PIL (versão 8.2.0)

Tkinter é uma biblioteca da linguagem Python que acompanha a instalação padrão e permite desenvolver interfaces gráficas. Isso significa que qualquer computador que tenha o interpretador Python instalado é capaz de criar interfaces gráficas usando o Tkinter, com exceção de algumas distribuições Linux que exigem o download do módulo separadamente.

O Tkinter já vem por padrão na maioria das instalações Python, então tudo que temos que fazer é importar a biblioteca.

```
''' IMPORT '''

import tkinter as tk

''' FROM '''

from tkinter import *

from tkinter import ttk
```

Matplotlib é uma biblioteca para a visualização de dados em Python. Ele apresenta uma API orientada a objetos que permite a criação de gráficos em 2D de uma forma simples e com poucos comandos. A ferramenta disponibiliza diversos tipos de gráficos, como em barra, em linha, em pizza, histogramas entre outras opções. A API foi projetada para ser compatível com o MATLAB , apesar de ser referência na área de processamento numérico.

O PyPlot é um módulo do matplotlib para criação de gráficos. Para utilizá-lo é necessário fazer a importação, após importar o módulo, já é possível criar gráficos de uma forma simples e com poucos comandos.

```
''' FROM '''

from matplotlib import pyplot as plt
```

NumPy é uma poderosa biblioteca Python que é usada principalmente para realizar cálculos em Arrays Multidimensionais. O NumPy fornece um grande conjunto de funções e operações de biblioteca que ajudam os programadores a executar facilmente cálculos numéricos. Esses tipos de cálculos numéricos são amplamente utilizados em tarefas. Imagens no computador são representadas como Arrays Multidimensionais de números. NumPy torna-se a escolha mais natural para o mesmo. O NumPy, na verdade, fornece algumas excelentes funções de biblioteca para rápida manipulação de imagens. Alguns exemplos são o espelhamento de uma imagem, a rotação de uma imagem por um determinado ângulo etc.

```
''' IMPORT '''

import numpy

import numpy as np
```

OpenCV (Open Source Computer Vision) é uma biblioteca de programação, de código aberto e inicialmente desenvolvida pela Intel com o objetivo de tornar a visão computacional mais acessível a desenvolvedores e hobistas. Atualmente possui mais de 500 funções, pode ser utilizada em diversas linguagens de programação (C++, Python, Ruby, Java…) e é usada para diversos tipos de análise em imagens e vídeos, como detecção, tracking e reconhecimento facial, edição de fotos e vídeos, detecção e análise de textos.

```
''' IMPORT '''

import cv2
```

Mahotas é uma biblioteca de visão computacional e processamento de imagens para Python. Inclui muitos algoritmos implementados em C ++ para velocidade durante a operação em matrizes numpy e com uma interface Python muito limpa. Mahotas atualmente tem mais de 100 funções para processamento de imagens e visão computacional e continua crescendo.

```
''' IMPORT '''

import mahotas as mt
```

Scikit-learn é uma biblioteca da linguagem Python desenvolvida especificamente para aplicação prática de machine learning. Esta biblioteca dispõe de ferramentas simples e eficientes para análise preditiva de dados, é reutilizável em diferentes situações, possui código aberto, sendo acessível a todos e foi construída sobre os pacotes NumPy, SciPy e Matplotlib. Uma das melhores opções para aplicação prática de machine learning é através da linguagem Python. Um dos fatores que trás destaque a linguagem são justamente suas bibliotecas e pacotes, que proporcionam muita simplicidade as aplicações, além de garantir scripts descomplicados e eficientes. Dentre estes pacotes, temos o NumPy e o Pandas como os principais para a preparação dos dados, e o scikit-learn, ou apenas sklearn, sendo o mais utilizado para efetiva criação de modelos de machine learning.

```
''' IMPORT '''

from skimage import io, color, img_as_ubyte

from skimage.feature import greycomatrix, greycoprops
```

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix

from sklearn.neural_network import MLPClassifier
```

PIL ou Biblioteca de Imagens do Python é um pacote que expõe muitas funções para manipular imagens.

```
''' IMPORT '''

import PIL

''' FROM '''

from PIL import Image,ImageTk
```

3. DESCRIÇÃO SOBRE IMPLEMENTAÇÃO

3.1. Ler e visualizar imagens

O método para abrir imagens consiste em abrir uma caixa de diálogos do sistema para escolher uma imagem com extensão Portable Network Graphics (.PNG) ou Tagged Image File Format (.TIFF). Ao selecionar a imagem e confirmar, a imagem é carregada no canvas do software.

```
      #INÍCIO DO MÉTODO PARA LER A IMAGEM

      def openImagen():

         ''' Variavel global utilizada em outro metodo '''

         global image

         global aux

         global val1

         global val2

         global name

         ''' ---------------------------------------- '''

      #ABRI UMA NOVA TELA PARA SELECIONAR A IMAGEM,SÃO ACEITOS ARQUIVOS
.PNG E .TIFF
```

```
        name  = filedialog.askopenfilename(initialdir='',
title="Imagens",filetypes=(("png files",".png"),("tiff files",".tiff*")))
        nameAux = name
        image = Image.open(name)
        aux = ImageTk.PhotoImage(image)
    #ABRE O CANVAS COM O TAMANHO PRÉ DEFINIDO
        canvas.config(width=aux.width(), height=aux.height())
        canvas.pack(expand = True)
        canvas.image = aux
    #CARREGA A IMAGEM NO CANVAS
        canvas.create_image(0,0, image = aux, anchor = tk.NW)
        canvas.place(x=110,y=10)
        val1 = aux.width()
        val2 = aux.height()
```

3.2. Zoom

O zoom é composto de dois métodos, o Zoom In e o Zoom Out. O Zoom in consiste em expandir a imagem para visualização melhor dos detalhes, já o Zoom Out diminui o tamanho da imagem. Ambos os métodos aplicam um coeficiente de Zoom de 1,25x.

```
#MÉTODO PARA REALIZAR O ZOOM IN
def zoomIN():
   ''' Variável global utilizada em outro método '''
   global image
   global aux
   global val1
   global val2
   ''' ---------------------------------------- '''
  #CARREGA A IMAGEM
```

```
    aux = ImageTk.PhotoImage(image)
    #DEFINIR O ZOOM IN ATRAVÉS DO CÁLCULO
    width = aux.width()*1.25
    height = aux.height()*1.25
    #REALIZA O RESIZE PARA APLICAR O ZOOM
    image = image.resize((int(width),int (height)), Image.ANTIALIAS)
    aux = ImageTk.PhotoImage(image)
    #CONFIGURA O CANVAS
    canvas.config(width=aux.width(), height=aux.height())
    canvas.pack(expand=True)
    canvas.image = aux
    #CARREGA A IMAGEM COM ZOOM NO CANVAS
    canvas.create_image(0,0,image=aux, anchor=tk.NW)
    canvas.place(x=110,y=10)
    val1 = aux.width()
    val2 = aux.height()
```

```
#MÉTODO PARA REALIZAR O ZOOM IN
def zoomOUT():
    ''' Variável global utilizada em outro método '''
    global image
    global aux
    global val1
    global val2
    ''' ---------------------------------------- '''
    #CARREGA A IMAGEM
    aux = ImageTk.PhotoImage(image)
    #DEFINIR O ZOOM OUT ATRAVÉS DO CÁLCULO
    width = aux.width()*0.75
```

```
    height = aux.height()*0.75

    #REALIZA O RESIZE PARA APLICAR O ZOOM

    image = image.resize((int(width),int (height)), Image.ANTIALIAS)

    aux = ImageTk.PhotoImage(image)

     #CONFIGURA O CANVAS

    canvas.config(width=aux.width(), height=aux.height())

    canvas.pack(expand=True)

    canvas.image = aux

    #CARREGA A IMAGEM COM ZOOM NO CANVAS

    canvas.create_image(0,0,image=aux, anchor=tk.NW)

    canvas.place(x=110,y=10)

    val1 = aux.width()

    val2 = aux.height()
```

3.3. Região de interesse

A região de interesse é um retângulo com bordas azuis de tamanho 128px por 128px que aparece na posição do mouse que o usuário clicou. Ela serve de referência para o usuário escolher uma área para analisar. O usuário pode clicar duas vezes ou clicar em “Recortar Área” para confirmar o corte da imagem. Ao confirmar o corte da região de interesse, o fragmento é salvo em disco com o nome “.corte.png”.

```
#MÉTODO PARA SELECIONAR DETERMINADA REGIÃO DA IMAGEM

def selecao():

    ''' Variável global utilizada em outro método '''

    global image

    global aux

    global val1

    global val2

    global name

    ''' ---------------------------------------- '''
```

```
#IDENTIFICA OS EVENTOS DO MOUSE
   def get_mouse_pos(event):
    nonlocal topy, topx, botx, boty
    nonlocal aux_tela
    topx, topy = event.x, event.y
    botx = topx + 64
    boty = topy + 64
    topx = topx - 64
    topy = topy - 64
    #DETERMINA AS POSIÇÕES NO CANVAS
    canvas.coords(aux_tela, topx, topy, botx, boty)
    return
#CONFIRMA O CORTE DA ÁREA SELECIONADA
def confirm_cut(event):
    nonlocal topy, topx, botx, boty
    border = (topx, topy, botx, boty)
    #SELECIONA A PARTE DA IMAGEM
    aux_img = Image.open(name)
    #REALIZA O RESIZE DA PARTE SELECIONADA
    aux_img = aux_img.resize((val1,val2))
    #REALIZA O CORTE PARA APRESENTAR APENAS A PARTE CORTADA
    img_crop = aux_img.crop(border)
    #SALVAR IMAGEM NO DISCO
    img_crop.save(".corte.png")
    #CARREGA A IMAGEM NO DISCO
    im = Image.open(".corte.png")
    aux = ImageTk.PhotoImage(im)
    #CONFIGURA O CANVAS
```

```
        canvas.config(width=aux.width(), height=aux.height())
        canvas.pack(expand=True)
        canvas.image = aux
        #CARREGA A IMAGEM CORTADA NO CANVAS
        canvas.create_image(0,0,image=aux, anchor=tk.NW)
        canvas.place(x=110,y=10)
        return
    topx, topy, botx, boty = 0, 0, 0, 0
    aux_tela = None
    #CRIA O RETÂNGULO DE COR AZUL COM TAMANHO 128x128
    aux_tela = canvas.create_rectangle(topx, topy, botx, boty, fill='',
outline='Blue', width=2)
    #EVENTO DO MOUSE
    canvas.bind('<Button-1>', get_mouse_pos)
    #IDENTIFICA DOIS CLIQUES PARA CONFIRMAR O CORTE DA IMAGEM
    canvas.bind('<Double-Button-1>', confirm_cut)
```

3.4. Resolução da região de interesse

Há uma pop-up no software que permite ao usuário escolher com qual resolução da imagem quer trabalhar, elas podem ser respectivamente 32x32, 64x64 e 128x128. O software reduz a qualidade da imagem para uma das resoluções selecionadas pelo usuário e sobrescreve o arquivo “.corte.png”.

```
#REDUZ A RESOLUÇÃO PARA 64x64
def redux64():
        #ABRE A IMAGEM CORTADA ANTERIORMENTE
        img = cv2.imread(".corte.png")
        #SELECIONA A DIMENSÃO ORIGINAL
        dimensao_original = img.shape
        #ALTERA A RESOLUÇÃO DIMINUINDO A ESCALA ATRAVÉS DE UMA
```

```
PORCENTAGEM

scale_percent = 50

width = int(img.shape[1] * scale_percent/100)

height = int(img.shape[0] * scale_percent/100)

#DEFINE O TAMANHO

dim = (width, height)

#REDUZ A IMAGEM

resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

#SELECIONA UMA EM CADA 2 COLUNAS, UMA EM CADA DUAS

LINHAS

n = 2

m = 2

img_red = img[::n, ::m]

#AUMENTA A IMAGEM

#OS PIXELS DA IMAGEM ATUAL SERÃO DUPLICADOS NO EIXO X

E Y

# np.repeat(matriz, vezes, eixo). O EIXO 0 É A ALTURA

E 1 A LARGURA

m = 2

img_aum = np.repeat(img_red, m, axis = 0)

img_aum = np.repeat(img_aum, m, axis = 1)

cv2.imwrite(".corte.png", img_aum)

#CARREGA A IMAGEM NO CANVAS

im = Image.open(".corte.png")

aux = ImageTk.PhotoImage(im)

#CONFIGURA O CANVAS

canvas.config(width=aux.width(), height=aux.height())

canvas.pack(expand=True)
```

```
        canvas.image = aux
        #CARREGA A IMAGEM CORTADA NO CANVAS
        canvas.create_image(0,0,image=aux, anchor=tk.NW)
        canvas.place(x=110,y=10)
```

```
#REDUZ A RESOLUÇÃO PARA 32x32
def redux32():
        #ABRE A IMAGEM CORTADA ANTERIORMENTE
        img = cv2.imread(".corte.png")
        #SELECIONA A DIMENSÃO ORIGINAL
        dimensao_original = img.shape
        #ALTERA A RESOLUÇÃO DIMINUINDO A ESCALA ATRAVÉS DE UMA
        PORCENTAGEM
        scale_percent = 75
        width = int(img.shape[1] * scale_percent/100)
        height = int(img.shape[0] * scale_percent/100)
        #DEFINE O TAMANHO
        dim = (width, height)
        #REDUZ A IMAGEM
        resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
        #SELECIONA UMA EM CADA 2 COLUNAS , UMA EM CADA 2 LINHAS
        n = 2
        m = 2
        img_red = img[::n, ::m]
        #AUMENTANDO A IMAGEM
        #OS PIXELS DA IMAGEM ATUAL SERÃO DUPLICADOS NO EIXO
        # np.repeat(matriz, vezes, eixo). O EIXO 0 É A ALTURA
```

```
        E 1 PARA LARGURA

        m = 2

        img_aum = np.repeat(img_red, m, axis = 0)

        img_aum = np.repeat(img_aum, m, axis = 1)

        cv2.imwrite(".corte.png", img_aum)

        #CARREGA A IMAGEM NO CANVAS

        im = Image.open(".corte.png")

        aux = ImageTk.PhotoImage(im)

        #CONFIGURA O CANVAS

        canvas.config(width=aux.width(), height=aux.height())

        canvas.pack(expand=True)

        canvas.image = aux

        #CARREGA A IMAGEM CORTADA NO CANVAS

        canvas.create_image(0,0,image=aux, anchor=tk.NW)

        canvas.place(x=110,y=10)
```

3.5. Quantização

É possível selecionar em uma outra pop-up a escala de tons de cinza que será aplicada na imagem, as escalas possíveis são 16, 32 ou 256 tons. Ao selecionar, o software aplica os tons de cinza na imagem do canvas e sobrescreve a imagem “.corte.png” no disco.

```
#APLICA 16 TONS DE CINZA

def tons16():

        global tom

        #LE A IMAGEM CORTADA

        img = cv2.imread(".corte.png",0)

        #CALCULA A QUANTIZAÇÃO

        r = 15

        imgQuant = np.uint8 (img/r) * r
```

```
        tom = imgQuant
        #APLICA A QUANTIZAÇÃO NO CORTE
        cv2.imwrite(".corte.png", imgQuant)
        #CARREAGA A IMAGEM NO CANVAS
        im = Image.open(".corte.png")
        aux = ImageTk.PhotoImage(im)
        # CONFIGURA O CANVAS
        canvas.config(width=aux.width(), height=aux.height())
        canvas.pack(expand=True)
        canvas.image = aux
        # CARREGA A IMAGEM COM TOM APLICADO NO CANVAS
        canvas.create_image(0,0,image=aux, anchor=tk.NW)
        canvas.place(x=110,y=10)
```

```
# APLICA 32 TONS DE CINZA
def tons32():
        global tom
        #LE A IMAGEM CORTADA
        img = cv2.imread(".corte.png",0)
        # CALCULA A QUANTIZAÇÃO
        r = 8
        imgQuant = np.uint8 (img/r) * r
        tom = imgQuant
        #APLICA A QUANTIZAÇÃO NO CORTE
        cv2.imwrite(".corte.png", imgQuant)
        #CARREAGA A IMAGEM NO CANVAS
        im = Image.open(".corte.png")
        aux = ImageTk.PhotoImage(im)
```

```
        # CONFIGURA O CANVAS
        canvas.config(width=aux.width(), height=aux.height())
        canvas.pack(expand=True)
        canvas.image = aux
        # CARREGA A IMAGEM COM TOM APLICADO NO CANVAS
        canvas.create_image(0,0,image=aux, anchor=tk.NW)
        canvas.place(x=110,y=10)
```

3.6. Equalização

Ao clicar no botão “Equalização”, o software exibe a imagem equalizada numa pop-up, sobrescreve a imagem “.corte.png” no disco e exibe uma outra pop-up com o gráfico do histograma da imagem.

```
#EQUALIZAÇÃO E HISTOGRAMA
def equalizacao():
        #LE A IMAGEM CORTADA, COM QUANTIZAÇÃO E TONS APLICADO
        img = cv2.imread(".corte.png",0)
        #FUNÇÃO PARA EQUALIZAR
        equaliza = cv2.equalizeHist(img)
        #APLICA A QUANTIZAÇÃO
        cv2.imwrite(".corte.png",equaliza)
        #FUNÇÃO PARA CONFIGURAR HISTOGRAMA
        plt.hist(equaliza.ravel(), 256, [0, 256])
        #MOSTRA EQUALIZAÇ˜ÃO
        cv2.imshow("Imagem com equalizacao", equaliza)
        #MOSTRA HISTOGRAMA
        plt.show()
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        #CARREGA A IMAGEM NO CANVAS
```

```
        im = Image.open(".corte.png")
        aux = ImageTk.PhotoImage(im)
        #CONFIGURA O CANVAS
        canvas.config(width=aux.width(), height=aux.height())
        canvas.pack(expand=True)
        canvas.image = aux
        # CARREGA A IMAGEM EQUALIZADA
        canvas.create_image(0,0,image=aux, anchor=tk.NW)
        canvas.place(x=110,y=10)
```

3.7. Haralick

O método do Haralick consiste em uma pop-up para configurar as características da imagem, como a homogeneidade, a entropia e o contraste da imagem. Ao confirmar, a pop-up fecha, aplica as características na imagem presente no canvas e retorna a matriz de co-ocorrências.

```
#ABRE POP UP PARA SELEÇÃO DO HARALICK
def telaHaralick(selectTam_opened):
   global aux_cls
   if not selectTam_opened:
       selectTam_opened = True
       entropy = False
       homogeneity = False
       contrast = False
      #DEFINE TAMANHO POP UP
       width = 400
       height = 350
       # obtém metade da largura / altura da tela e largura / altura da
janela
       # criação de interface
       top = Toplevel()
```

```
        top.title("Selecionar")
        top.lift()
        top.resizable(False, False)
        # posiciona a janela no centro da página
        top.geometry("%dx%d+%d+%d" % (width, height, 400, 350))
        l = Label(top, text = '\n\nSelecionar características:\n')
        l.pack()
        check_entropy = IntVar()
        check_homogeneity = IntVar()
        check_contrast = IntVar()
        #OPÇÃO PARA USUÁRIO SELECIONAR TEXTURA
        C1 = Checkbutton(top, text="Entropia", variable=check_entropy,
                         onvalue=1, offvalue=0, height=2, width=20)
        C2 = Checkbutton(top, text="Homogeneidade",
variable=check_homogeneity,
                         height=2, width=20)
        C3 = Checkbutton(top, text="Contraste", variable=check_contrast,
                          height=2, width=20)
        #IRÁ SELECIONAR DE ACORDO COM O QUE O USUÁRIO
      ESCOLHER
      if entropy:
            C1.select()
       if homogeneity:
            C2.select()
       if contrast:
            C3.select()
       C1.pack()
       C2.pack()
       C3.pack()
```

```
        def on_closing():
            top.quit()
            top.destroy()
        top.protocol("WM_DELETE_WINDOW", on_closing)
        top.mainloop()
        #QUANDO FECHAR IRÁ DEFINIR A TEXTURA SELECIONADA  PELO
        USUÁRIO
        if check_entropy.get() == 1:
            entropy = True
        else:
            entropy = False
        if check_homogeneity.get() == 1:
            homogeneity = True
        else:
            homogeneity = False
        if check_contrast.get() == 1:
            contrast = True
        else:
            contrast = False
        selectTam_opened = False
        aux_cls = [entropy, homogeneity, contrast]
        #print(aux_cls)
        msgbx.showinfo(title="Selecionar características",
                       message="As características marcadas foram
selecionadas.")
```

```
#Features do Haralick
def features(rgbImg, properties):
   #Converte uma imagem em formato de byte, com valores em [0, 255]
   grayImg = img_as_ubyte(color.rgb2gray(rgbImg))
```

```
    distances = [1,2,4,8,16]
    #ângulos 0, 45, 90, 135
    #0, pi/4, pi/2, 3pi/4
    #função np para pegar o valor de pi
    angles = [0, np.pi/4, np.pi/2, 3*np.pi/4]
    # o greycomatrix irá utilizar a imagem, a distância e o angulo
    matrix_coocurrence =
greycomatrix(grayImg,distances=distances,angles=angles,symmetric=True,normed
=True)
    return(matrix_feature(matrix_coocurrence,properties))


def matrix_feature(matrix_coocurrence, properties):
  feature = np.hstack([greycoprops(matrix_coocurrence, prop).ravel() for
prop in properties])
    return feature
```

```
def haralick(grayImg):
    global aux_cls
    # Todas as opçoes
    if aux_cls == [True,True, True]:
        properties = ['contrast', 'energy', 'homogeneity']
        return features(grayImg, properties)
    # Homogeneidade e Entropia
    elif aux_cls == [True,True,False]:
        properties = ['energy', 'homogeneity']
        return features(grayImg, properties)
    # Entropia e Contraste
    elif aux_cls == [True,False,True]:
        properties = ['energy', 'homogeneity', 'contrast']
        return features(grayImg, properties)
```

```
    # Homogeneidade e Contraste
    elif aux_cls == [False,True,True]:
        properties = ['contrast', 'homogeneity']
        return features(grayImg, properties)
    # Entropia
    elif aux_cls == [True,False,False]:
        properties = ['energy', 'homogeneity']
        return features(grayImg, properties)
    # Homogeneidade
    elif aux_cls == [False,True,False]:
        properties = ['homogeneity']
        return features(grayImg, properties)
    # Contrate
    elif aux_cls == [False,False,True]:
        properties = ['contrast']
        return features(grayImg, properties)
```

3.8. Rede neural

A rede neural foi construída utilizando a classe **MLP Classifier** da biblioteca Sklearn, automatizando a construção de uma rede neural multicamada. Após a classificação da rede neural, mostra à qual classe de predição pertence a imagem que o usuário selecionou.

Optamos em usar o selecionador “ **solver= ‘adam’** ”, pois é o padrão da rede neural e a quantidade de neurônios de duzentos, duzentos e duzentos“ **hidden_layer_sizes=(200,200,200)** ”. Com isso o classificador foi preparado com as quatrocentas imagens recebidas e com a matriz da imagem gerada pelo Haralick. O teste foi realizado utilizando o **predict** que retorna os valores da classificação das imagens de teste.

Depois disso a rede neural está pronta para uso, com os dados recebidos foi possível gerar a matriz de confusão utilizando o **confusion_matrix** .

3.8.1. Treinar os classificadores

Ao clicar em “Treinar Rede” o software abre uma caixa de diálogo do

sistema para selecionar o diretório das imagens de treino. Ao confirmar a seleção de pastas, o software carrega as imagens modelo, presente em 4 subdiretórios numerados de 1 a 4.

Após carregar as imagens, é chamado o método para treinar a rede neural, que utiliza as imagens carregadas e testa a matriz de concorrência retornada pelo método Haralick.

3.8.2. Matriz de confusão

Após o treino da rede neural, o método gera a matriz de confusão. Com ela, é possível calcular a acurácia e especificidade do classificador.

```
      # Carrega as 400 imagens que o professor disponibilizou para o
treino da rede neural
      def directorio():
          '''
          Ao clicar em "Treinar Rede" o software abre uma caixa de
diálogo do sistema para selecionar o diretório das imagens de treino.
          Ao confirmar a seleção de pastas, o software carrega as
imagens modelo, presente em 4 subdiretórios numerados de 1 a 4.
          '''
          imagens = []
          try:
              folder = filedialog.askdirectory()
              for i in range(1,5):
                  subFolder = folder + '/' + str(i)
                  files = os.listdir(subFolder)
                  for arquivo in files:
                      img = cv2.imread(subFolder + '/' + arquivo)
                      imagens.append(img)
              msgbx.showinfo(title="ATENÇÃO", message=str(len(imagens))
+ " imagens foram carregadas com sucesso!")
              return imagens
          except:
```

```
            msgbx.showinfo(title="ATENÇÃO", message="Erro ao carregar
imagens. Verifique a pasta!")
      '''---------------------------------------- FIM CARREGAR
DIRETORIO ------------------------------ '''


      # Treina a rede neural com as 400 imagens disponibilizadas pelo
professor, alem de testar a matriz retornada do Haralick
      def treinarRedeNeural():
         global aux_mlp
         img_matriz = directorio()
         Img = io.imread(".corte.png")
         if len(img_matriz) == 400:
             width = 450
             height = 50
             posX = int(1100 / 2 - width / 2)
             posY = int(500/ 2 - height / 2)
             tela = Toplevel()
             tela.title("Treinar Rede")
             tela.resizable(False, False)
             tela.lift()
             tela.geometry("%dx%d+%d+%d" % (width, height, posX,
posY))
             l = Label(
                 tela, text='\n\nTreinando, aguarde.\n\n').pack()
             tela.after(1000, tela.quit)
             tela.mainloop()
             tela.destroy()
             Ftreino = []
             Ltreino = []
             '''
             Ao clicar em "Treinar os classificadores" o software abre
```

```
uma caixa de diálogo do sistema
            para selecionar o diretório das imagens de treino. Ao
confirmar a seleção de pastas,
            o software carrega as imagens modelo, presente em 4
subdiretórios numerados de 1 a 4.
            Após carregar as imagens, é chamado o método para treinar
a rede neural,
            que utiliza as imagens carregadas e testa a matriz de
concorrência retornada pelo método Haralick.
            '''
            # Definições dos rotulos
            for i in range(0, 400):
                Ltreino.append(int(i / 100) + 1)
            for img in img_matriz:
                # Aplica o retorno do haralick no caso a matriz
                val = haralick(Img)
                Ftreino.append(val)
            # Para 4 classes com 25 imagens
            classificador1, classificador2, classificador3,
classificador4 = np.array_split(
                Ftreino, 4)
            classifica1, classifica2, classifica3, classifica4 =
np.array_split(
                Ltreino, 4)
            # 75% das imagens escolhidas de forma aleatória, mas
balanceadas entre as classes
            # Classificar os 25% das imagens restantes
            treinador1, Ctreino1, treino1, teste1 = train_test_split(
                classificador1, classifica1, test_size=0.25,
random_state=1)
            treinador2, Ctreino2, treino2, teste2 = train_test_split(
                classificador2, classifica2, test_size=0.25,
```

```
random_state=1)
            treinador3, Ctreino3, treino3, teste3 = train_test_split(
                classificador3, classifica3, test_size=0.25,
random_state=1)
            treinador4, Ctreino4, treino4, teste4 = train_test_split(
                classificador4, classifica4, test_size=0.25,
random_state=1)
            # Recebendo os dados gerados
            treinador = np.concatenate(
                (treinador1, treinador2, treinador3, treinador4))
            Ctreino = np.concatenate(
                (Ctreino1, Ctreino2, Ctreino3, Ctreino4))
            treino = np.concatenate(
                (treino1, treino2, treino3, treino4))
            teste = np.concatenate(
                (teste1, teste2, teste3, teste4))
            # Rede neural sendo criada
            '''
            A rede neural foi construída utilizando a classe MLP
Classifier da biblioteca Sklearn, automatizando a construção de uma
rede neural multicamada.
            Optamos em usar o selecionador " solver= 'adam' " pois é
o padrao da rede neural. aplicamos a quantidade de neurônios de
duzentos e trezentos " hidden_layer_sizes=(200,200,200) ".
            Com isso o classificador foi preparado com as
quatrocentas imagens recebidas e com a matriz da imagem gerada pelo
Haralick.
            O teste foi realizado utilizando o predict que retorna os
valores da classificação das imagens de teste.
            Depois disso a rede neural está pronta para uso, com os
dados recebidos foi possível gerar a matriz de confusão utilizando o
confusion_matrix
            '''
```

```
            mlp = MLPClassifier(
                solver='adam',hidden_layer_sizes=(200,200,200))
            mlp.fit(treinador, treino)
            aux_mlp = mlp
            aux_pred = mlp.predict(Ctreino)
            '''
            Após o treino da rede neural, o método gera a matriz de
confusão.
            Com ela, é possível calcular a acurácia e especificidade
do classificador.
            '''
            # Gera a matriz de confusão com os dados recebido da
rede neural
            matrix_confusion = confusion_matrix(teste, aux_pred)
            acu = acuracia(matrix_confusion)
            especife = especificidade(matrix_confusion)
            print_valRede(matrix_confusion,especife,acu)
        else:
            msgbx.showinfo(
                title="Atenção", message="Leia o diretório com as
imagens de teste para poder treinar o classificador.")
```

3.9. Classificar região

Depois que a rede neural realiza o treinamento, a imagem de corte **“.corte.png”**, primeiro verifica se o classificador existe, após isso realiza a predição da imagem recortada e passa para o método de impressão sua classe de predição.

```
# Analisar área do corte
def analisarArea():
   global aux_mlp
   if aux_mlp != None:
       Img = io.imread(".corte.png")
```

```
            analisar = haralick(Img)

            analisar = np.array(analisar)

            predição = aux_mlp.predict(analisar.reshape(1, -1))[0]

            print_valCorte(predição)

        else:

            msgbx.showinfo(title="Atenção",

                           message="A não foi treinado.")

    ''' ---------------------------------------- FIM ANALISAR AREA
---------------------------------------- '''
```

3.10. Especificidade e acurácia

Com a matriz de confusão gerada, conseguimos calcular a acurácia e a especificidade.

Na função criada, a **acurácia** recebe a matriz de confusão, realiza o cálculo da diagonal principal da matriz e divide para quantidade de imagens de teste no caso cem.

Na função criada, a **especificidade** utiliza os valores restantes e divide pela quantidade de imagens do treinamento da rede neural no caso trezentos.

```
    # A sensibilidade média = acurácia = Σi=1..4 Mi,i /100

    def acuracia(matriz):

        result = 0

        for i in range(0, 4):

            result += matriz[i][i]

        return result / 100

    ''' ---------------------------------------- FIM PRECISÃO
---------------------------------------- '''


    # A especificidade = 1- Σi=1..4 Σj≠i Mj,i / 300

    def especificidade(matriz):

        result = 0
```

```
        for i in range(0, 4):
            for j in range(0, 4):
                if i != j:
                    result += matriz[i][j]
        result = 1 - result / 300
        return result
```

4. TESTES

4.1. O primeiro teste foi verificar se os métodos do Haralick estavam gerando a matriz de co-ocorrências.

4.1.1. Entropia

```
ENTROPIA:  [0.42124047 0.37205033 0.40468852 0.37385267 0.36519914 0.37205033
 0.33859363 0.37385267 0.31615627 0.29904519 0.2995786  0.3080594
 0.27774613 0.27001681 0.27434998 0.28045169 0.25300374 0.25528089
 0.25898644 0.26555851 0.85741969 0.73462954 0.81947969 0.73940142
 0.71257619 0.73462954 0.63609398 0.73940142 0.56391363 0.50044094
 0.50081639 0.53222632 0.42448698 0.37769032 0.40580609 0.43247864
 0.30055947 0.29236184 0.31414403 0.34435713]
```

4.1.2. Homogeneidade

```
HOMOGENEIDADE:  [0.85741969 0.73462954 0.81947969 0.73940142 0.71257619 0.73462954
 0.63609398 0.73940142 0.56391363 0.50044094 0.50081639 0.53222632
 0.42448698 0.37769032 0.40580609 0.43247864 0.30055947 0.29236184
 0.31414403 0.34435713]
```

4.1.3. Contraste

```
CONTRASTE:  [ 548.75406004 1086.07440015  729.02620571 1050.52669105 1106.21850198
 1086.07440015 1469.62425595 1050.52669105 1943.18548387 2492.975808
 2457.51512097 2217.507136   3302.8375     4017.96425692 3917.33567708
 3478.93093255 5046.77399554 6205.52355906 6098.24804688 5063.82591862]
```

4.2. Após as matrizes de co-ocorrências serem geradas com êxito, passamos ela para o classificador que começou a treinar a rede junto com as 400 imagens do banco de dados, podendo nos passar as informações da acurácia, especificidade e a matriz de confusão.

4.2.1. Acurácia

```
ACURACIA:  0.25
```

4.2.2. Especificidade

```
ESPECIFICIDADE:  0.75
```

4.2.3. Matriz de confusão

```
MATRIZ DE CONFUSAO:
 [[ 0  0 25  0]
 [ 0  0 25  0]
 [ 0  0 25  0]
 [ 0  0 25  0]]
```

4.3. Com os dados de Especificidade, Acurácia e a Matriz de confusão foi possível fazer a análise da área recortada, informando em qual classe de predição ela se encontra.

4.3.1. Classe de Predição

```
CLASSE DE PREDICAO:  3
```

4.4. Tempo de execução

Para saber o tempo de execução do programa, usamos o comando **time** na linha de comando para execução.

```
python3 main.py  11,55s user 2,74s system 12% cpu 1:57,06 total
```

4.5. Teste da Rede Neural

Teste 1

Números de Neuronio = [100, 100, 100]

Acurácia encontrada = 25%

Teste 2

Números de Neuronio = [100, 200, 100]

Acurácia encontrada = 25%

Teste 3

Números de Neuronio = [200, 200, 200]

Acurácia encontrada = 25%

Teste 4

Números de Neuronio = [200, 300, 100]

Acurácia encontrada = 25%

Teste 5

Números de Neuronio = [300, 300, 200]

Acurácia encontrada = 25%

Teste 6

Números de Neuronio = [300, 300, 300]

Acurácia encontrada = 25%

Observe que ambos testes feitos com a rede neural obtêm o mesmo valor de acurácia em todos, no qual é 25%.

5. Conclusão

O tema processamento de imagem está sendo comentado principalmente quando se trata de análise de exames, em específico o exame da mama e a rede neural tem causado um grande impacto nos diagnósticos trazendo uma eficiência maior.

O desenvolvimento do presente estudo possibilitou uma análise de como um software de análise de imagens é feito e pode aprimorar a apuração de resultado de exames de mamografia. Além disso, permitiu um estudo mais aprofundado do conteúdo passado em aula. E nos ajudou a obter um conhecimento mais consistente sobre desenvolvimento de interface gráfica.

Com o software funcional utilizando a quantização e equalização junto com os descritores de textura do Haralick foi possível realizar o treinamento da rede neural e realizar os testes na região selecionada, obtendo 75% de especificidade e 25% de acurácia.

6. Bibliografia

Sites utilizados como base para a construção do software

https://www.devmedia.com.br/tkinter-interfaces-graficas-em-python/33956

https://king.host/blog/2018/03/visualizacao-de-dados-matplotlib/

https://medium.com/ensina-ai/entendendo-a-biblioteca-numpy-4858fde63355

https://blog.cedrotech.com/opencv-uma-breve-introducao-visao-computacional-com-python/

https://mahotas.readthedocs.io/en/latest/

https://didatica.tech/a-biblioteca-scikit-learn-pyhton-para-machine-learning/

https://pythonhelp.wordpress.com/2011/11/20/tratando-argumentos-com-argparse/

https://python-guide-pt-br.readthedocs.io/pt_BR/latest/scenarios/imaging.html

https://imasters.com.br/back-end/primeiros-passos-com-pil-a-biblioteca-de-imagens-do-python

https://www.pucsp.br/~jarakaki/pai/Roteiro4.pdf

https://www.monolitonimbus.com.br/histograma-em-python/

https://www.youtube.com/watch?v=83RZSK1j8Ak

https://code.tutsplus.com/pt/tutorials/histogram-equalization-in-python--cms-30202

https://medium.com/data-hackers/equalização-de-histograma-em-python-378830368d60

https://www.geeksforgeeks.org/python-pil-image-quantize-method/

https://www.pyimagesearch.com/2014/07/07/color-quantization-opencv-using-k-means-clustering/

https://www.pyimagesearch.com/2015/03/09/capturing-mouse-click-events-with-python-and-opencv/

https://github.com/lmoesch/py-glcm

https://www.ti-enxame.com/pt/python/calculo-da-entropia-do-glcm-de-uma-imagem/829236908/

https://github.com/LendisFabri/Gray-Level-Cooccurrence-Matrix/blob/master/glcm_fix.py

https://www.geeksforgeeks.org/mahotas-haralick-features/

https://www.letscode.com.br/blog/visao-computacional-como-o-computador-ve-uma-imagem

https://cadernodelaboratorio.com.br/en/converting-image-color-spaces/

https://numpy.org/doc/stable/reference/generated/numpy.hstack.html

https://stackoverflow.com/questions/54725203/mlpclassifier-model-accuracy-fine-tuning-with-20000-samples

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

https://stackoverflow.com/questions/51337067/extracting-haralick-features-from-glcm-why-do-i-get-multiple-values-for-each-fe

https://living-sun.com/pt/python/712567-calculating-entropy-from-glcm-of-an-image-python-numpy-entropy-scikit-image-glcm.html

http://www.ic.uff.br/~aconci/co-ocorrenciaMathLab.pdf