Relatório - Atividade Prática 01

UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
DC/CCN037 - Processamento Digital de Imagens
Professor: Kelson Romulo Teixeira
Aluno: Amarildo Junior

Sumário

Introdução

Este trabalho foi realizado com a intenção de implementar e praticar determinados conceitos da disciplina de Processamento Digital de Imagens. Foram disponibilizadas algumas questões e arquivos para que o trabalho possa ser feito. A linguagem de programação escolhida foi Python, pela popularidade e facilidade de implementação, e a biblioteca Pillow para abrir e manpular as imagens.

Resolução das Questões

Questão 1

Nessa questão foi pedido para criar uma imagem contendo apenas os pontos da fronteira do objeto da imagem *folha.png* utilizando adjacência-4 e adjacência-8.

Para solucionar essa questão, foi desenvolvida uma função `imagem_fronteira()` recebendo o caminho da imagem e o tipo de adjacência como parâmetro, como é possível visualizar no anexo 1. A princípio, foram definidas as cores de cada vizinhança em um dicionário. Logo após, a imagem é aberta utilizando o método Image.open() da biblioteca Pillow e guardada na variável `im`. Como a questão se refere a criar uma imagem, a linha correspondente a `Image.new(mode = "RGB", size = (im.width, im.height), color = (0, 0, 0))` irá criar uma imagem no formato RGB, com as mesmas dimensões da imagem de entrada e com todos os pixels pretos. Após isso, é definido o nome do

arquivo da imagem de saída. Ainda na definição de variáveis que são importantes para o código, é necessário algo que possibilite a visualização e manipulação de cada pixel. Para isso, o método `im.load()` é chamado e o retorno é guardado na variável `px`. Assim, será possível manipular qualquer pixel passando apenas os valores das coordenadas: `px[i, j]`, como em uma matriz.

Depois de definir as variáveis, foram utilizados dois *for* para percorrer a matriz que representa a imagem de entrada, um que irá de `i` até a altura da imagem de entrada e outro que irá de `j` até a largura da imagem. Como a imagem tem em sua composição apenas um fundo com todos os pixels pretos, que representa (0, 0, 0, 255) no sistema *rgba*, e um objeto com os pixels da cor branca, representada por (255, 255, 255, 255), a ideia foi percorrer a matriz analisando se cada pixel faz parte do objeto, tendo o valor correspondente à cor branca, ou do fundo da imagem, com o valor correspondente ao preto. Considerando que a fronteira estará entre o objeto e o fundo, então é verificado se o pixel é da cor branca, caso seja, são verificados também os valores dos possíveis pixels adjacentes 4 que fazem parte do fundo da imagem, em outras palavras, verifica-se se o valor do pixel acima, abaixo ou dos lados corresponde ao valor (0, 0, 0, 255), se sim, px[i, j] é um pixel que faz parte da fronteira correspondete à adjacência-4 e, consequentemente, à adjacência-8. Considerando que a imagem de saída foi definida com o mesmo tamanho da imagem de entrada, então se o pixel for de fronteira, o pixel com as mesmas coordenadas na imagem de saída tem seu valor alterado com o método `im_saida.putpixel((i, j), cor)`. Entretanto, caso nenhuma das condições foram satisfeitas, é analisado se a adjacência utilizada é 8, caso seja, os valores dos pixels de cada diagonal de `px[i, j]` são verficados para o valor da cor preta, ou seja, se compõem o fundo da imagem e assim, se `px[i, j]` é um pixel que faz parte da fronteira, o pixel correspondente às coordenadas i e j na im_saida tem seus valores alterados. Ao fim disso, a imagem é salva utilizando o método `im_saida.save(nome_do_arquivo)`.

Questão 2

A questão 2 solicita os histogramas normal, normalizado (f.d.p.) e acumulado(f.d.a.) da imagem “lena_gray.bmp”.

Para calcular o histograma, foi criada uma função `histograma()` que recebe como parâmetro o caminho da imagem - anexo 2.1. Considerando que a imagem tem 8 bits pra representar os níveis de intensidade, ou seja, 256 níveis diferentes, é criado uma lista `histograma` com 256 elementos, todos iguais a 0 e cada posição representando um nível de intensidade. A imagem de entrada é carregada com o método já mostrado `im.load()` e assim como antes o valor é guardado numa variável `px`, que reprensentará a matriz da imagem. A imagem será percorrida pixel a pixel, onde cada valor de intensidade é contabilizado na lista `histograma[px[i, j]] += 1`. Ao terminar, a função retorna a lista `histograma`. O gráfico é formado com a função `grafico()` que utiliza a biblioteca matplotlib e recebe como parâmetro os valores dos eixos x e y, o titulo do gráfico e o nome do arquivo que será gerado - anexo 2.4. Para gerar o gráfico, foi passado a lista dos valores de intensidade (0 a 255) para o eixo x e a quantidade de pixels com determinada intensidade para o eixo y.

A função `histograma_normalizado()`, que recebe o caminho da imagem como parâmetro - anexo 2.2 - foi criada com o objetivo de gerar o histograma normalizado de uma

imagem. A princípio, o histograma da imagem é calculado utilizando a função anterior e o retorno é armazenado na variável `h_normalizado`. Para ter acesso aos atributos da imagem, ela é aberta com o método `Image.open()`. Então, é utilizado um laço de repetição que vai de 0 ao maior nível de intensidade e a cada iteração o valor da quantidade de pixels com aquela intensidade é dividido pelo tamanho da imagem, obtido multiplicando a largura pela altura. O resultado da divisão é colocado na mesma posição da lista `h_normalizado` e ao fim do laço essa lista é retornada. O gráfico será gerado com a mesma função `grafico()` tendo a lista dos valores de intensidade (0 a 255) correspondente ao eixo x e a probablidade de ocorrer cada intensidade na imagem para o eixo y.

Por fim, para obter o histograma acumulado, foi implementada a função `histograma_acumulado()` que recebe o caminho da imagem como parâmetro - anexo 2.3. Bem como o histograma normalizado, primeiramente é calculado o histograma normal da imagem com a função `histograma()` e o retorno é guardado na variável `h_acumulado`. A partir disso, essa lista é percorrida no intevalo de 1 até 255, onde a cada iteração o elemento atual será a soma entre ele e seu anterior. Assim, o retorno da função será uma lista com os valores acumulados de cada intensidade dentro do intervalo (0 - 255). Após o retorno da função, bastar gerar o gráfico utilizando `grafico()` e passando como parâmetro a lista dos valores de intensidade (0 a 255) correspondente ao eixo x e a lista dos valores acumulados para o eixo y.

Questão 3

Esta questão pede a implementação de uma função que permita equalizar uma imagem. Assim, a função deve ser aplicada às imagens “lena_gray.bmp” e “image1.png” e ainda equalizar a imagem equalizada.

Para a solução dessa questão, foram necessárias duas funções, uma para equalizar o histograma e outra para mostrar a imagem a partir do histograma. A função `equalizar()` - anexo 3.1 -que recebe o caminho de uma imagem como parâmetro foi criada para equalizar um histograma de uma imagem. Primeiramente, a imagem é aberta e o histograma acumulado é calculado e guardado numa variável h_equalizado, utilizando a função citada na questão 2 e no anexo 2.3. Além disso, também é calculada a quantidade de pixels, obtida através da multiplicação da altura pela largura da imagem. Começando o processo de equalização, é feito um laço de repetição que irá de 0 ao valor máximo de intensidade, e a posição da lista correspondente à determinada intensidade `i` será igual ao valor arredondado obtido de `h_equalizado[i] / pixels * 255`, ou seja, o somatório das probabilidades até o nível de intensidade de `i` multiplicado pelo valor máximo de intensidade. Assim, a função retorna `h_equalizado`. Para visualizar a imagem equalizada, basta chamar a função `mostrar_imagem()` mostrada no anexo 3.2, que mostra uma imagem a partir de um histograma dado, passando como parâmetros o caminho da imagem, o histograma equalizado e um título para o nome do arquivo.

Questão 4

Foi solicitado que sejam aplicadas as seguintes transformações lineares em uma imagem:
a. g = c * f + b

b. g = c * log2(f + 1)
c. g = c * (f + 1) ** b

A solução desta questão consiste em uma função principal `transformacao()`, mostrada no anexo 4.1, que recebe como parâmetro o caminho de uma imagem e a operação a ser feita, onde esta operação corresponde aos índices mostrados acima (a, b, c). Nesta função será analisada qual operação será feita, tendo cada operação sua respectiva função auxiliar para aplicar a transformação linear. Para todas as funções que aplicam diretamente a transormação linear, o método foi praticamente o mesmo, modificando apenas a fórmula para calcular o resultado do pixel na imagem de saída. Assim, a imagem é aberta com `Image.open()` e uma imagem de saída é criada com `Image.new()`, tendo as mesmas dimensões da imagem de entrada. E então a imagem é percorrida pixel a pixel por dois *for* aninhados, onde o primeiro vai de 0 até o valor da altura da imagem e o segundo de 0 até a o valor da largura, tendo como variáveis de iteração `i` e `j` respectivamente. Assim, o valor do pixel na imagem de saída com as coordenadas i e j referentes às transformações será:

a. `im_saida.putpixel((i, j), (c * im.getpixel((i, j)) + b))` , da função auxiliar denominada de `transformacao_linear()`, mostrada no anexo 4.2 e recebe como parâmetros o caminho da imagem de entrada e os valores das constantes b e c
b. `im_saida.putpixel((i, j), round(c * log2(im.getpixel((i, j)) + 1)))`, da função auxiliar `transformacao_logaritmica()`, que pode ser visualizada no anexo 4.3 e recebe como parâmetros o valor da constante c e o caminho da imagem de entrada
c. `im_saida.putpixel((i, j), round (c * (im.getpixel((i, j)) + 1) ** b))`, calculado na função auxiliar `transformacao_potencia()`, no anexo 4.4 e que recebe como parâmetros o caminho da imagem de entrada e os valores das constantes b e c.

Questão 5

A última questão se refere à criação de uma função que permita especificar o histograma de uma imagem e aplicá-la na imagem “image1.png” utilizando o histogram de “lena_gray.bmp”.

Para a realização desta questão, foi criada apenas uma função `especifcar_histograma()` que recebe uma imagem de entrada que será manipulada e uma imagem que terá seu histograma especificado - anexo 5.1. Primeiramente, todas as variáveis que serão utilizadas na função são definidas: a imagem é aberta e armazenada em im, uma imagem de saída é criada como im_saida, o histograma especificado e o histograma normal são equalizados para que possam ser verificadas as correspondências de valores de intensidad, além de um array ser criado com o auxílio da biblioteca numpy. Primeiramente é feito um *for* que percorre cada valor do histograma equalizado do histograma especificada para calcular o valor mais próximo no histograma equalizado da imagem de entrada. Após isso, o valor é indexado no histograma final na posição em que o valor do histograma especificado teve o valor mais próximo no histograma equalizado da imagem de entrada. Após isso, basta percorrer a imagem de saída distribuindo o valor de cada pixel na imagem de saída de acordo com a correspondência do valor do pixel na imagem de entrada com o histograma final. Assim, a função salva a imagem de saída e retorna o histograma final.

Análise de resultados

- Questão 1

  Abaixo segue a imagem original “folha.png” e os resultados obtidos destacando apenas a fronteira da imagem utilizando os diferentes tipos de adjacência. Ao aproximar as imagens, é possível perceber que a fronteira utilizando adjecência-8 está bem mais grossa que a fronteira quando foi utilizada adjacência-4. O que era esperado, uma vez que a adjecência-8 abrange mais pixels.

Imagem original “folha.png”

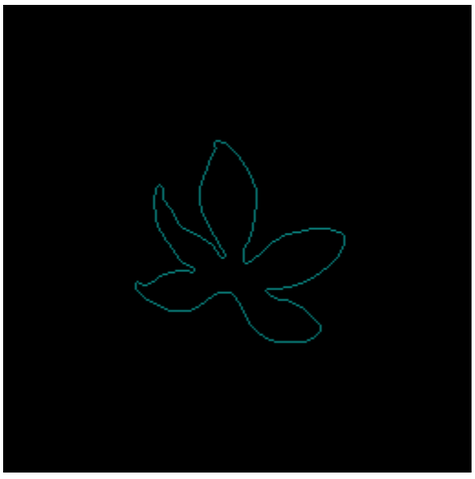


Imagem da fronteira do objeto de “folha.png” utilizando adjacência-4

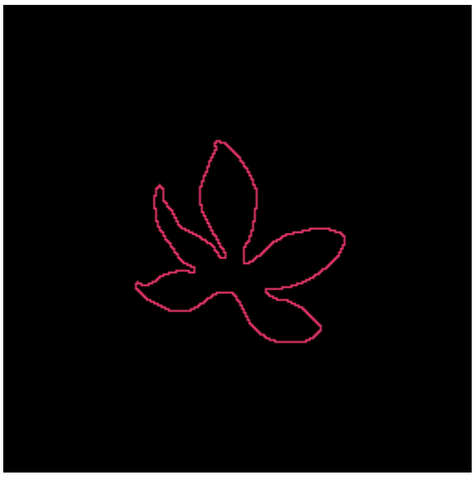
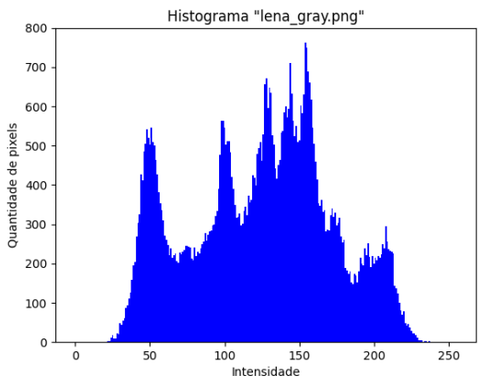
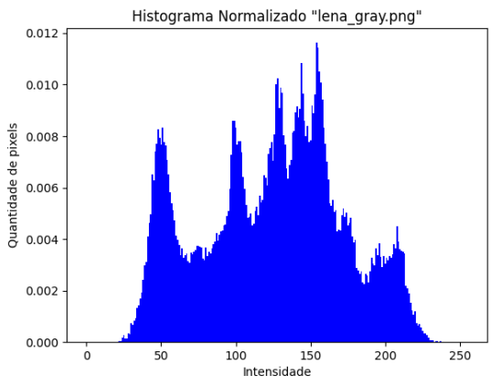


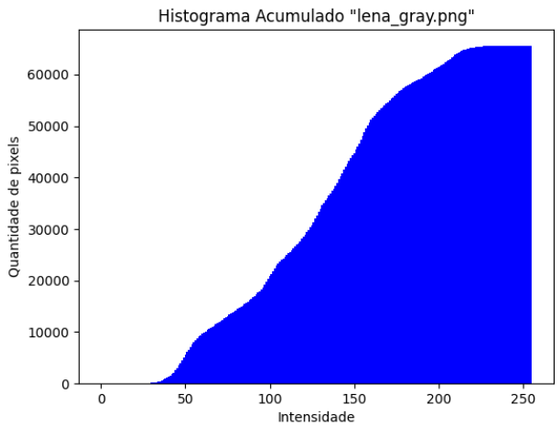
Imagem da fronteira do objeto de “folha.png” utilizando adjacência-8

- Questão 2

  Após calcular os histogramas pedidos na questão da imagem “lena_gray.bmp”, foram obtidos os resultados abaixo:







- Questão 3

  Segue abaixo todos os resultados das equalizações solicitadas na questão 3. Na “image1.png” é possível ver que após a equalização imagem obtida possui um melhor nível de detalhes, porém com algumas manchas. Já em “lena_gray.bmp” o resultado foi bem melhor, uma vez que é bem notório um realce com relação à imagem original.

  Ao se tratar de uma segunda equalização em nas imagens já equalizadas, não foi possível obter alguma diferença perceptível. Por isso, a diferença entre as imagens da segunda equalização e a primeira foi calculada e das duas imagens resultaram em uma imagem com todos os pixels pretos, conforme o histograma.

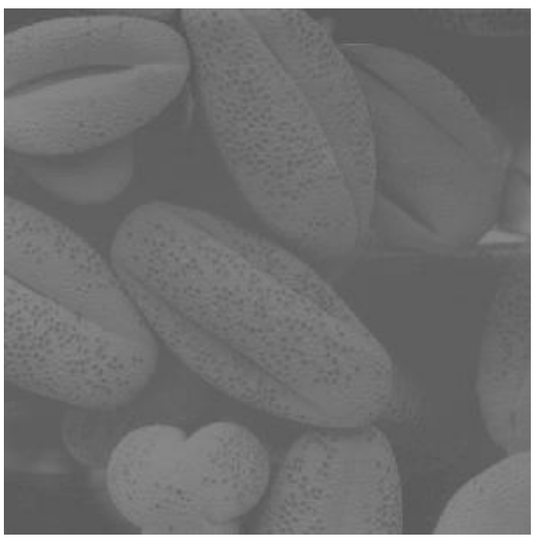
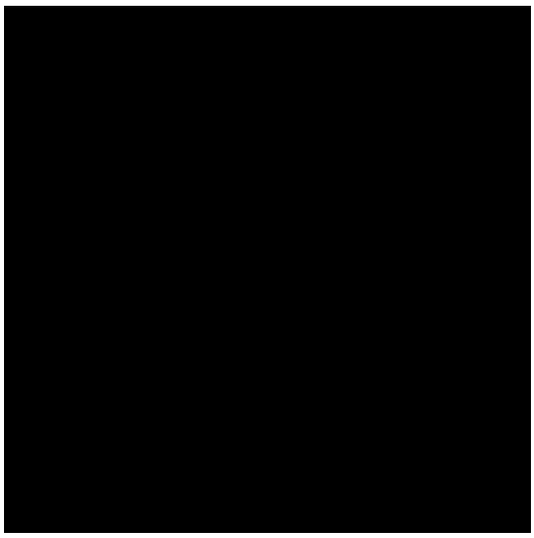


Imagem original

Imagem após equalização

Imagem após uma segunda equalização



Diferença entre as imagens da segunda e primeira equalização

Imagem original

Imagem após equalização

Imagem após uma segunda equalização

Diferença entre as imagens da segunda e primeira equalização

- Questão 4

  Para a aplicação de todas as transformações foi utilizada a imagem “lena_gray.png”. Ademais, foram aplicados alguns valores nas transformações lineares e os resultados obtidos analisados com o que se esperava.

Transformação a: c = 1, b = 1. A imagem não sofreu alterações.

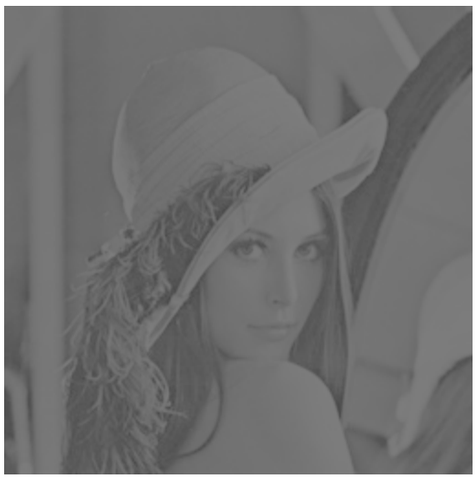
Transformação a: c = 1, b = 50. A imagem se tornou mais clara que a original.

Transformação a: c = 1, b = -50. A imagem se tornou mais escura que a original.

Transformação a: c = -1, b = 100. A imagem se tornou bem mais escura que a original.

Transformação b: c = 4. A imagem se tornou mais escura que a original.

Transformação b: c = 8. A imagem se torna mais clara conforme c aumenta



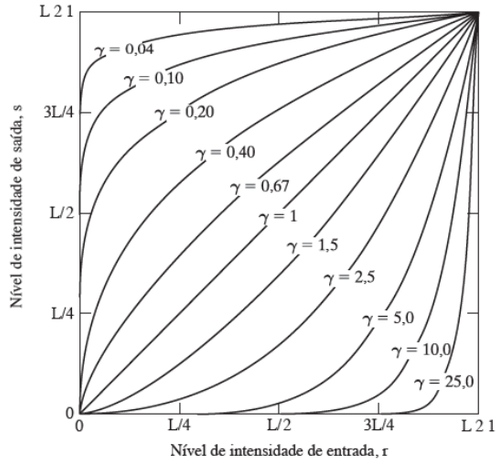
Transformação b: c = 16. A imagem ainda mais clara em relação ao valor de c anterior

Transformação b: c = 32. Imagem com pixels concentrados nas maiores intensidades

Transformação c: c = 1, b = 0.67. A imagem mais escura, pois b < 1

Transformação c: c = 1, b = 1. A imagem sem alterações, pois b = 1

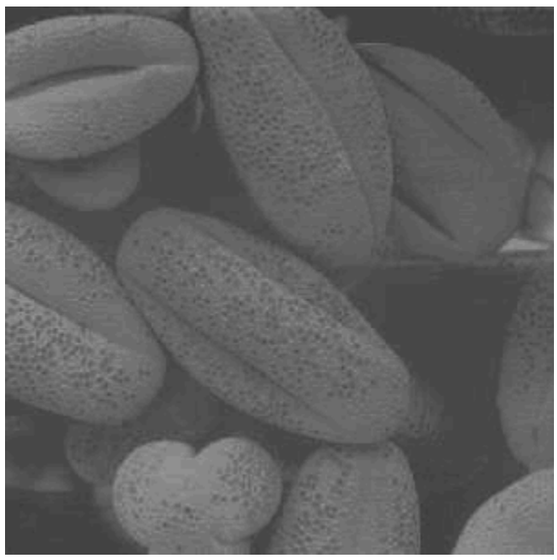
Transformação c: c = 1, b = 1.5. A imagem saturada, pois b > 1.
O que também caso b = 1 e o valor de c aumentar.



É possível perceber que as imagens da transformação c seguem esses valores, onde b = gama.

- Questão 5

O resultado obtido da especificação do histograma de “lena_gray.bmp” em “image1.png” pode ser visto abaixo. A imagem parece ter aumentado o nível de detalhes com relação à original, mas ainda sim não é uma diferença tão grande.



Resultado da especificação do histograma de “lena_gray.bmp” em “image1.png”

Anexos

Anexo 1.1 - Código referente à questão 1 sobre pixels de fronteira.

```
def imagem_fronteira(imagem, adjacencia):
    cores = {4:(10, 131, 127, 255), 8:(241, 57, 109, 255)}
    im = Image.open(imagem)
    im_saida = Image.new(mode = "RGB", size = (im.width, im.height), color = (0, 0, 0))

    nome_arquivo = f"folha-fronteira-adj{adjacencia}.png"

    px = im.load()
    for i in range(0, im.height):
        for j in range(0, im.width):
            if px[i, j] == (255, 255, 255, 255):
                if px[i, j + 1] == (0, 0, 0, 255) or px[i, j - 1] == (0, 0, 0, 255) or px
[i + 1, j] == (0, 0, 0, 255) or px[i - 1, j] == (0, 0, 0, 255):
                    im_saida.putpixel((i, j), cores[adjacencia])
                elif adjacencia == 8:
                        if px[i + 1, j + 1] == (0, 0, 0, 255) or px[i - 1, j - 1] == (0,
 0, 0, 255) or px[i + 1, j - 1] == (0, 0, 0, 255) or px[i - 1, j + 1] == (0, 0, 0, 255):
                            im_saida.putpixel((i, j), cores[adjacencia])
    im_saida.save(f"questao1/{nome_arquivo}")
```

Anexo 2.1 - Código referente à questão 2 sobre cálculo de histograma.

```
def histograma(imagem):
    histograma = [0] * 256
    im = Image.open(imagem)
    px = im.load()
    for i in range(0, im.height):
        for j in range(0, im.width):
            histograma[px[i, j]] += 1
    return histograma
```

Anexo 2.2 - Código referente à questão 2 sobre cálculo de histograma normalizado.

```
def histograma_normalizado(imagem):
    h_normalizado = histograma(imagem)
    im = Image.open(imagem)
    for i in range(0, len(h_normalizado)):
        h_normalizado[i] /= im.width*im.height
    return h_normalizado
```

Anexo 2.3 - Código referente à questão 2 sobre cálculo de histograma acumulado.

```
def histograma_acumulado(imagem):
    h_acumulado = histograma(imagem)
    for i in range(1, len(h_acumulado)):
        h_acumulado[i] += h_acumulado[i-1]
    return h_acumulado
```

Anexo 2.4 - Código referente à criação de gráficos na questão 2 sobre cálculo de histogramas.

```
def grafico(eixo_x, eixo_y, titulo, nome_arquivo):
    plt.bar(eixo_x, eixo_y, color="blue", width = 1)
    plt.xticks([0, 51, 102, 153, 204, 255])
    plt.ylabel("Quantidade de pixels")
    plt.xlabel("Intensidade")
    plt.title(titulo)
    plt.savefig('questao2/' + nome_arquivo, format='png')
```

Anexo 3.1 - Código referente à questão 3 sobre equalização de imagens.

```
def equalizar(imagem):
    im = Image.open(imagem)
    h_equalizado = histograma_acumulado(imagem)
    pixels = im.width * im.height
    for i in range(0, len(h_equalizado)):
        h_equalizado[i] = round(h_equalizado[i] / pixels * 255)
    return h_equalizado
```

Anexo 3.2 - Código referente à questão 3 sobre equalização de imagens.

```
def mostrar_imagem(imagem, histograma, titulo):
    im = Image.open(imagem)
    im_saida = Image.new(mode = "L", size = im.size)
    for i in range (im.size[0]):
        for j in range (im.size[1]):
            im_saida.putpixel((i, j), histograma[im.getpixel((i, j))])
    im_saida.save(f"questao3/equalizacao-{titulo}")
```

Anexo 4.1 - Código referente à função principal da questão 4 sobre aplicação de transformações lineares.

```
def transformacao(imagem, operacao, c, b):
    if operacao == 'a':
        transformacao_linear(imagem, c, b)
    elif operacao == 'b':
        transformacao_logaritmica(imagem, c)
    elif operacao == 'c':
        transformacao_potencia(imagem, c, b)
```

Anexo 4.2 - Código referente à função auxiliar da questão 4 sobre aplicação da transformação linear a.

```
def transformacao_linear(imagem, c , b):
    im = Image.open(imagem)
    im_saida = Image.new(mode = "L", size = (im.width, im.height))
    for i in range(0, im.height):
        for j in range(0, im.width):
            im_saida.putpixel((i, j), (c * im.getpixel((i, j)) + b))
    im_saida.save("questao4/letra-a.png")
```

Anexo 4.3 - Código referente à função auxiliar da questão 4 sobre aplicação da transformação linear b.

```
def transformacao_logaritmica(imagem, c):
    im = Image.open(imagem)
    im_saida = Image.new(mode = "L", size = (im.width, im.height))
    for i in range(0, im.height):
        for j in range(0, im.width):
            im_saida.putpixel((i, j), round(c * log2(im.getpixel((i, j)) + 1)))
    im_saida.save("questao4/letra-b.png")
```

Anexo 4.4 - Código referente à função auxiliar da questão 4 sobre aplicação da transformação linear c.

```
def transformacao_potencia(imagem, c, b):
    im = Image.open(imagem)
    im_saida = Image.new(mode = "L", size = (im.width, im.height))
    for i in range(0, im.height):
        for j in range(0, im.width):
            im_saida.putpixel((i, j), round (c * (im.getpixel((i, j)) + 1) ** b))
    im_saida.save("questao4/letra-c.png")
```

Anexo 5.1 - Código referente à questão 5 sobre especificação de histogramas.

```
def especificar_histograma(imagem_entrada, imagem_especificada):
    im = Image.open(imagem_entrada)
    im_saida = Image.new(mode = "L", size = im.size)
    hist = equalizar((imagem_entrada))
    hist_esp = equalizar((imagem_especificada))
    histograma_final = []

    arr = np.array(hist)
    for i in hist_esp:
        difference_array = np.absolute(arr-i)
        index = difference_array.argmin()
        histograma_final.append(arr[index])

    for i in range (im.size[0]):
        for j in range (im.size[1]):
            im_saida.putpixel((i, j), int (histograma_final[im.getpixel((i, j))]))
    im_saida.save(f"questao5/especificacao-{imagem_entrada}")
    return histograma_final
```