IFCE - Visão Computacional - Processamento Digital de Imagens

TÉCNICAS DE VISÃO COMPUTACIONAL BÁSICAS

Leonardo Costa de Sousa

Trabalho 1 - Operações básicas

Filtros passa-baixa:

Os Filtros passa-baixa minimizam os efeitos de ruído na imagem, suavizando (smoothing) as transições abruptas (altas freqüências), são atenuadas.

A filtragem passa-baixas tem o efeito indesejado de diminuir a resolução da imagem, provocando assim, um leve borramento. Ou seja, diminui a nitidez e a definição da imagem.

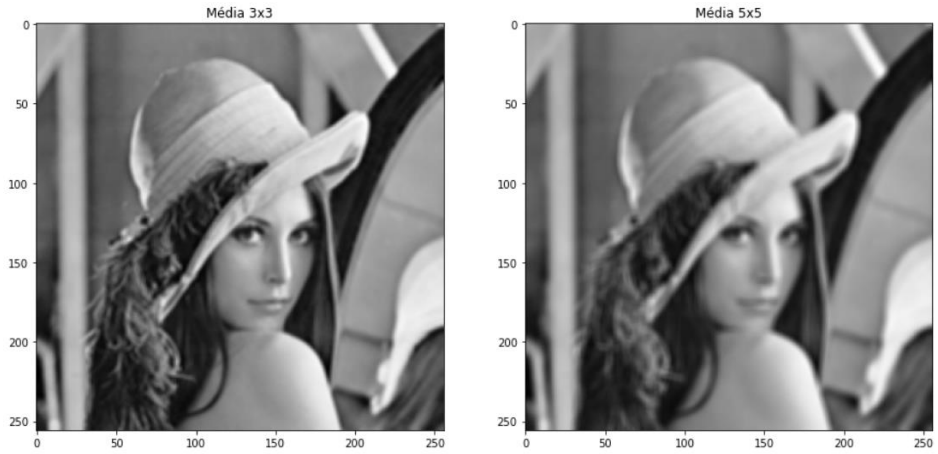
1. Média

O filtro de média é um filtro passa-baixa que trabalha diretamente nos valores de média dos pixels substituindo pela média aritmética do pixel dos seus vizinhos.

Logo abaixo é mostrado duas matrizes usados como máscara de média, uma de ordem 3 e outra de ordem 5 e para fazer a convolução das matrizes com a imagem é utilizado a função convolve2d do módulo scipy.signal.

```
m3x3 = np.array([[1, 1, 1],
                 [1, 1, 1],
                 [1, 1, 1]])/9

m5x5 = np.array([[1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1]])/25

img_media3x3 = convolve2d(img, m3x3, "same", "symm", fillvalue=0)
img_media5x5 = convolve2d(img, m5x5, "same", "symm", fillvalue=0)
```

O uso da máscara de média é utilizado para reduzir ruídos da imagem apos à aplicação do filtro, mas como pode ser visto nos resultados após a convolução, quanto maior a máscara, maior o efeito de borramento.



2. Mediana

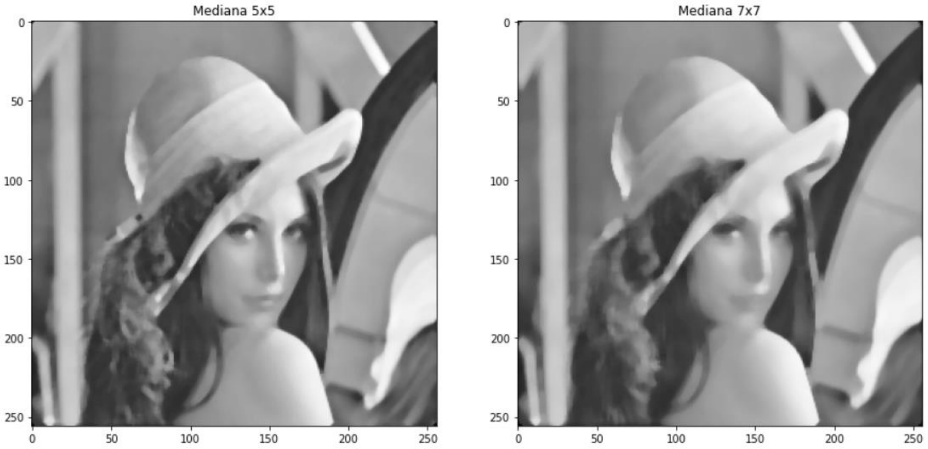
Os filtros de mediana é um filtro não linear e geralmente é usada para suaviza a imagem sem diminuir sua resolução ou borrar.

Nos filtros de mediana os pontos da vizinhança de (x,y), dentro de uma janela da imagem, são ordenados e tomado como novo valor para (x,y) o valor mediano desta ordenação.

No exercício foi usada uma função, a função mediana, para aplica esse conceito de mediana na imagem, passada como parâmetro junto com o a dimensão da janela da imagem.

```
def mediana(img, tam=3):
    import numpy as np
    import math as m

    lin, col = img.shape

    kernel = np.ones((tam, tam))

    central = m.floor((tam/2))

    C = np.zeros((lin + central * 2, col + central * 2))
    C[(0 + central):(lin + central), (0 + central):(col + central)] = img

    buffer = np.zeros((tam**2))
    D = np.zeros(img.shape)

    for j in range(0, lin):
        for k in range(0, col):

            for kl in range(0, tam):
                ## for each collumn:
                for kk in range(0, tam):

                    buffer[(tam * kl + kk)] = (C[j + kl, k + kk])

            buffer = np.sort(buffer)
            value = buffer[int(np.floor((tam**2)/2))]

            D[j, k] = value

    D = np.uint8(D)

    return D
```

Podemos ver nas imagens geradas que elas são suavizadas e o efeito de borramento e menos visível em relação ao filtro de média como pode ser observado comparando as imagens de Média 5x5 e Mediana 5x5, ou seja, esta última e bem mais nítida. Conclui-se então que o filtro da mediana procura suavizar a imagem e mantendo a nitidez.



3. Gaussiano

O filtro Gaussiano utiliza a função gaussiana para o cálculo dos coeficientes da máscara apresentando uma simetria rotacional, tratando todas as direções igualmente, é um filtro passa-baixa utilizado para suavização de ruídos.

$$h(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

A função get_gauss abaixo gera essa máscara com as dimensões da variável size e o desvio padrão definida pela variável sigma (σ, na fórmula), aplicando sua fórmula característica. Quanto maior a máscara e o valor do desvio padrão, maior será o nível de borramento.

```
def get_gauss(size, sigma=1.0):
    import numpy

    x, y = numpy.mgrid[-size//2 + 1:size//2 + 1, -size//2 + 1:size//2 + 1]
    g = numpy.exp(-((x**2 + y**2)/(2.0*sigma**2)))
    return g/g.sum()
```

Logo abaixo é mostrada uma máscara 3x3 com desvio 1.5 gerado por essa função:

```
máscara 3x3, desvio de 1.5:
[[0.09474166 0.11831801 0.09474166]
 [0.11831801 0.14776132 0.11831801]
 [0.09474166 0.11831801 0.09474166]]
```

Então são geradas duas máscaras, de 5x5, uma com desvio de 1.5 e outra com desvio de 3.5, aplicados a imagem.

```
img_gauss1 = convolve2d(img, g5x5_1, "same", "symm", fillvalue=0)
img_gauss2 = convolve2d(img, g5x5_2, "same", "symm", fillvalue=0)
```

A convolução é realizada através da função convolve2d do módulo scipy.signal. E como pode ser observado nas imagens quanto maior o desvio maior influencia em um maior nível de borramento.

Filtros passa-alta:

Os filtros passa-alta são normalmente usados para realçar os detalhes na imagem tendo como efeito tornar mais nítidas as transições entre regiões diferentes realçando o contraste, mas como efeito indesejado enfatizar o ruído presente na imagem.

Para esse item foi utilizado a função convolution2d, algumas vezes, para realizar a convolução, em vez da função convolve2d do módulo scipy.signal utilizada no item anterior.

```
def convolution2d(image, kernel, bias):
    m, n = kernel.shape
    if (m == n):
        y, x = image.shape
        y = y - m + 1
        x = x - m + 1
        new_image = np.zeros((y,x))
        for i in range(y):
            for j in range(x):
                new_image[i][j] = np.sum(image[i:i+m, j:j+m]*kernel) + bias
    return new_image
```

1. Laplaciano

O filtro Laplaciano é um filtro isotrópico, ou seja, não diferencia direções, bastando uma máscara, sendo mais fácil de implementar, não dá informações sobre a direção da borda mais é sensível a ruído. Sua mascara pode ter o centro negativo para remover bordas exteriores ou centro positivo para remover bordas interiores.

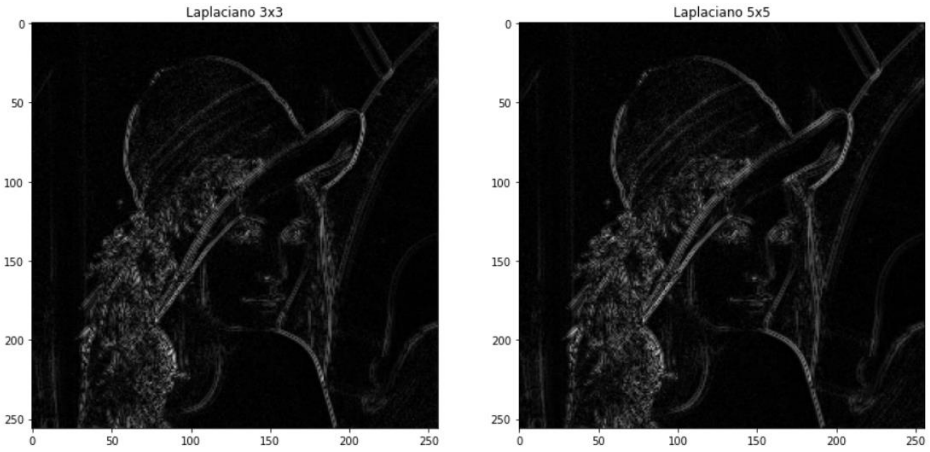
Foram aplicados duas matrizes, uma 3x3 com centro negativo e outra 5x5 com centro positivo.

```
l3x3 = np.asarray([[1,  1, 1],
                   [1, -8, 1],
                   [1,  1, 1]])

l5x5 = np.asarray([[ 0, 0,-1, 0, 0],
                   [ 0,-1,-2,-1, 0],
                   [-1,-2,16,-2,-1],
                   [ 0,-1,-2,-1, 0],
                   [ 0, 0,-1, 0, 0]])

img_laplaciano3x3 = convolve2d(img, l3x3, "same", "symm", fillvalue=0)
img_laplaciano5x5 = convolve2d(img, l5x5, "same", "symm", fillvalue=0)
```

$$\nabla^2 f(x,y) = \frac{\partial^2}{\partial x^2} f(x,y) + \frac{\partial^2}{\partial y^2} f(x,y)$$

A gradiente para a construção da imagem e mostrada e aplicada através da formula abaixo:

```
img_laplaciano3x3 = np.sqrt(2*img_laplaciano3x3**2)
img_laplaciano5x5 = np.sqrt(2*img_laplaciano5x5**2)
```



Como pode ser observado acima nos resultado da aplicação do filtro Laplaciano, ocorre uma detecção de bordas, mas de forma mais suave com menos ruído.

2. Prewit

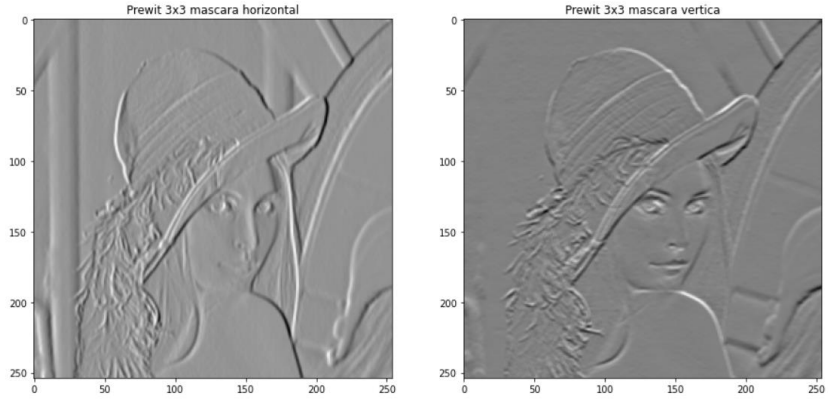
O filtro Prewitt utiliza duas máscaras que são convoluídas com a imagem original para calcular as derivadas nas direções verticais e horizontais. Esse operador é mais simples de ser implementado que o Sobel, mas apresenta mais ruídos.

No exemplo foi utilizado duas mascaras, uma 3x3 e outra 5x5.

```
p3x3 = np.asarray([[-1, 0, 1],
                   [-1, 0, 1],
                   [-1, 0, 1]])


p5x5 = np.asarray([[ 9, 9, 9, 9, 9],
                   [ 9, 5, 5, 5, 9],
                   [-7,-3, 0,-3,-7],
                   [-7,-3,-3,-3,-7],
                   [-7,-7,-7,-7,-7]])
```

No caso da mascara 3x3 foi aplicado individualmente sua versão em horizontal (que é mostrada abaixo) e sua versão vertical (sua transposta).

```
img_prewit_x = convolution2d(img, p3x3, 1) #máscara 3x3 na horizontal
img_prewit_y = convolution2d(img, p3x3.T, 1) #máscara 3x3 na vertical
```

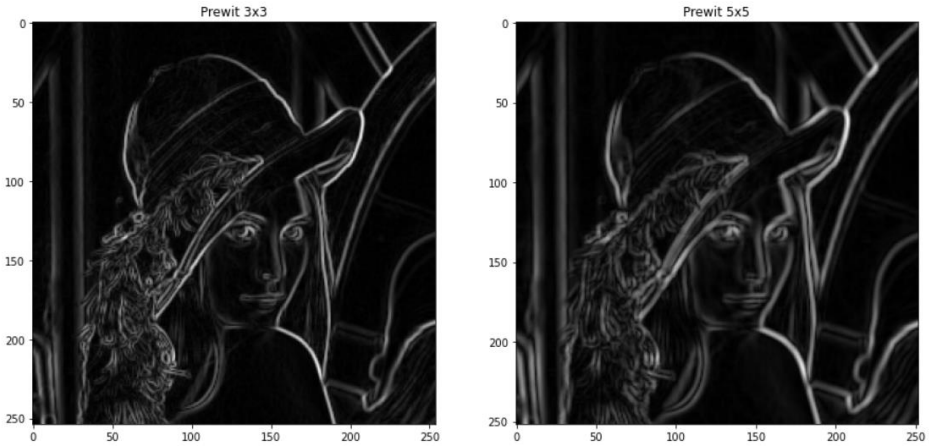


Para calcular as derivadas das duas mascaras horizontal e vertical e utilizada à função operadorDirecional, utilizada também no operador Sobel.

$$\sqrt{(Gx)^2 + (Gy)^2} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

```python
def operadorDirecional(img, gx):
    from scipy.signal import convolve2d

    gy = gx.T
    #Gx = convolve2d(img, gx, "same", "symm", fillvalue=0)
    #Gy = convolve2d(img, gy, "same", "symm", fillvalue=0)

    Gx = convolucao2d(img, gx, 1)
    Gy = convolucao2d(img, gy, 1)

    return np.sqrt(Gx**2 + Gy**2)
```

Abaixo é mostrado o resultado do operador Prewitt para mascaras de 3x3 e 5x5, com pode ser observado ele atua destacando as bordas da imagem.



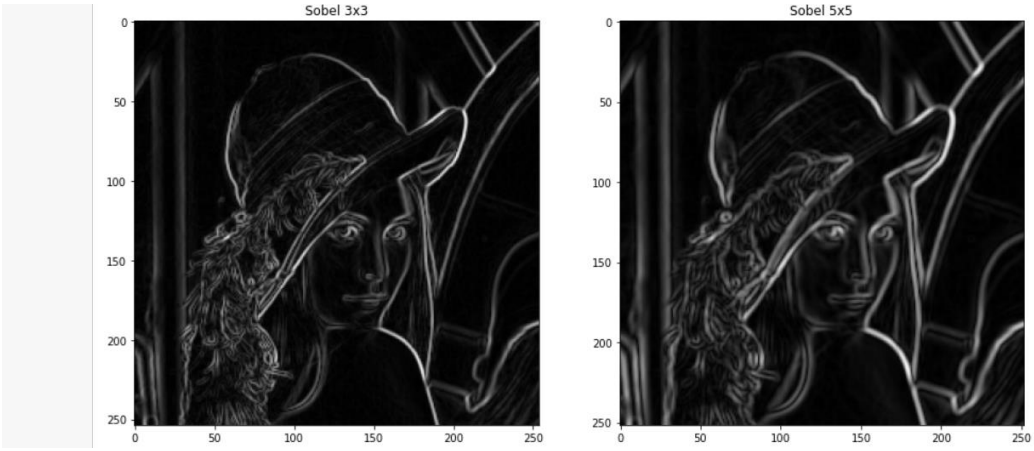
3. Sobel

O filtro Sobel utiliza duas máscaras deslocadas em 90° para encontrar os gradientes verticais e horizontais das bordas semelhante ao operador de Prewitt, porém com mais peso nos pontos próximos ao pixel central. Por esse motivo, a máscara de Sobel obtém as bordas mais destacadas em relação ao operador de Prewitt sendo muito menos sensível ao ruído.

Foi utilizado dois filtros sobel, um 3x3 e outro 5x5, e para o calculo do gradiente foi usado a função operadorDirecional.

```
s3x3 = np.asarray([[ 1,  0, -1],
                   [ 2,  0, -2],
                   [ 1,  0, -1]])


s5x5 = np.asarray([[ 2, 2, 4, 2, 2],
                   [ 1, 1, 2, 1, 1],
                   [ 0, 0, 0, 0, 0],
                   [-1,-1,-2,-1,-1],
                   [-2,-2,-4,-2,-2]])

sobel3x3 = operadorDirecional(img, s3x3)
sobel5x5 = operadorDirecional(img, s5x5)
```

Como pode ser observado nas imagens abaixo, o operador Sobel, e mais eficiente na detecção de bordas que o Prewitt. Podemos observar isso, por exemplo, nos detalhes no chapel e no ombro da moça na imagem.

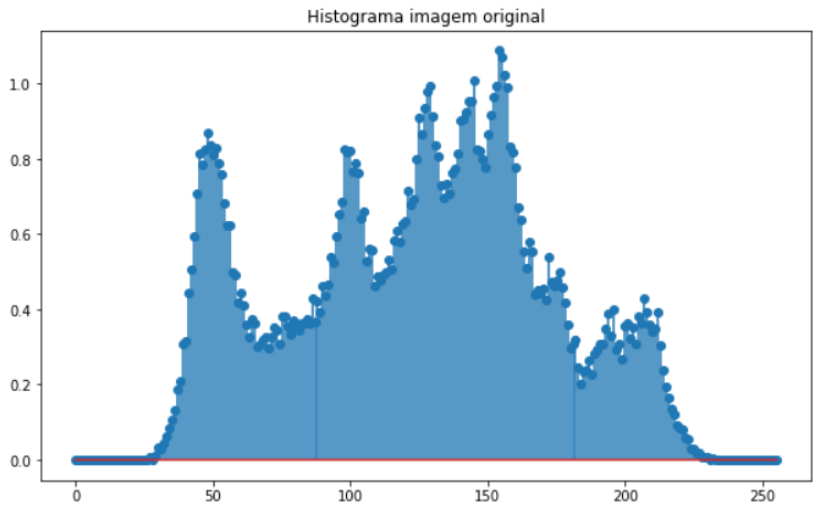


Outras operações:

1. Cálculo e apresentação do histograma

```
def calcHist(img):
    import numpy as np
    import matplotlib.pyplot as plt
    import math as m

    altura, largura = img.shape

    if img.dtype == 'uint8':
        buffer = np.zeros((256))

    for y in range(0, altura):
        for x in range(0, largura):

            buffer[(img[y, x])] += 1

    return 100*buffer/(altura*largura)
```

A função calcHist recebe uma imagem como argumento e calcula a freqüência de níveis de cinza dos pixels distribuídas da imagem e retorna esses dados para ser plotado em um gráfico.



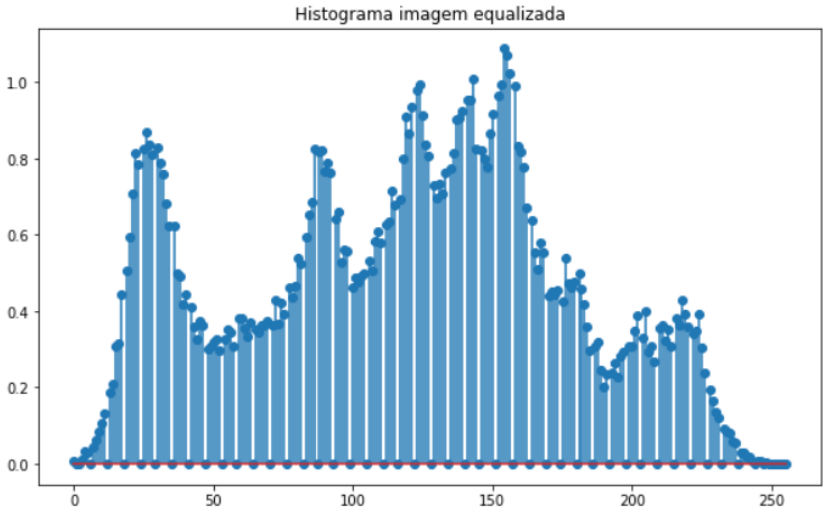
O gráfico acima mostra o resultado do histograma do arquivo de imagem lena_color_256.tif em tons de cinza.

2. Equalização do histograma:

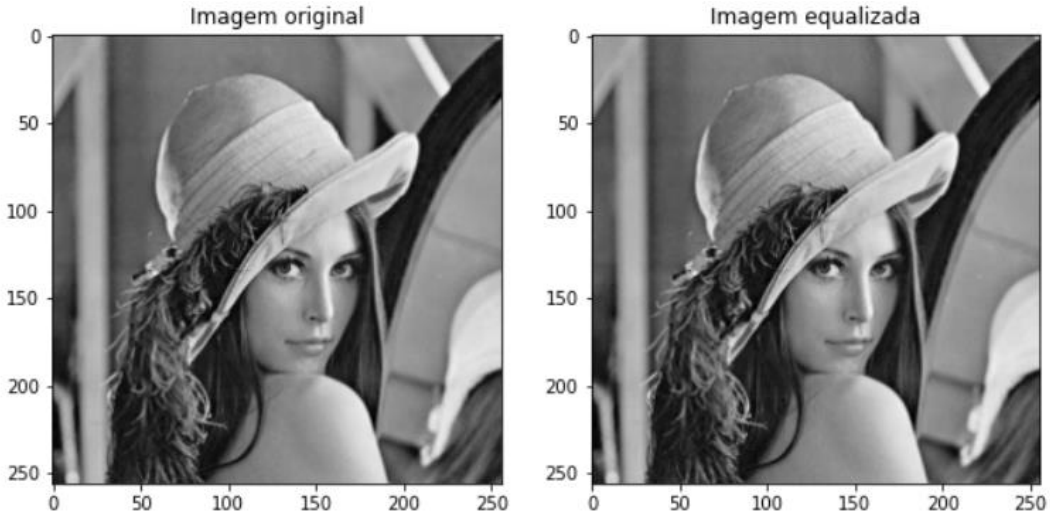
A função eqHistograma realiza a equalização de histograma, permitindo balancear os níveis de cinza em uma imagem resultando em um melhor contraste e vizualização dos detalhes da imagem.

```
def eqHistograma(img):
    import numpy as np
    import matplotlib.pyplot as plt
    import math as m

    lin, col = img.shape

    nova_img = np.zeros((lin, col))

    Tmin = np.min(img)
    Tmax = np.max(img)

    for j in range(0, lin):
        for k in range(0, col):
            nova_img[j, k] = np.ceil(255*((img[j, k] - Tmin)/(Tmax - Tmin)))

    nova_img = np.uint8(nova_img)

    return nova_img
```

Abaixo temos o resultado da equalização da imagem através de seu histograma onde pode ser percebida uma maior distribuição dos valores através do eixo horizontal do gráfico que antes estavam mais concentrados no centro.



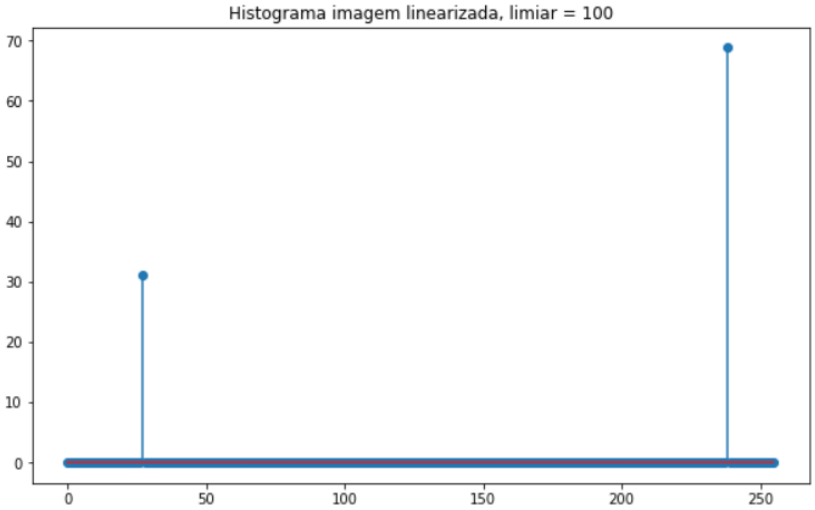
E abaixo compara a imagem original e a imagem equalizada, onde pode ser percebido um leve aumento de definição da imagem.

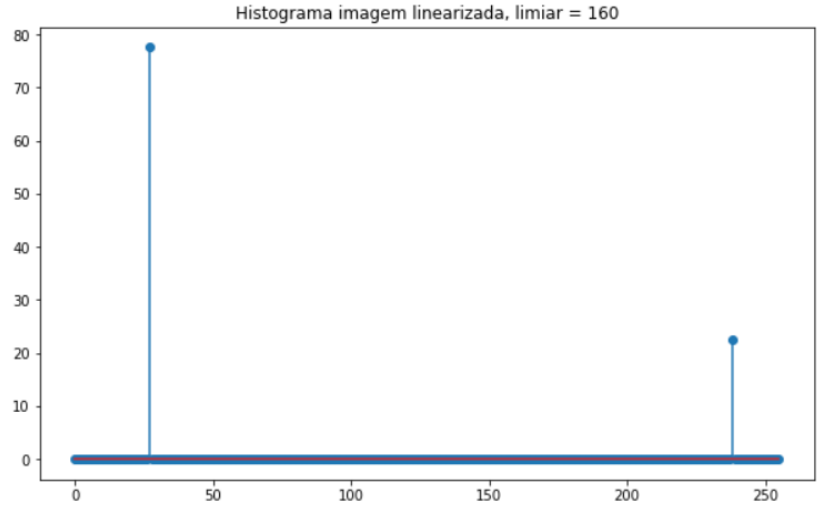


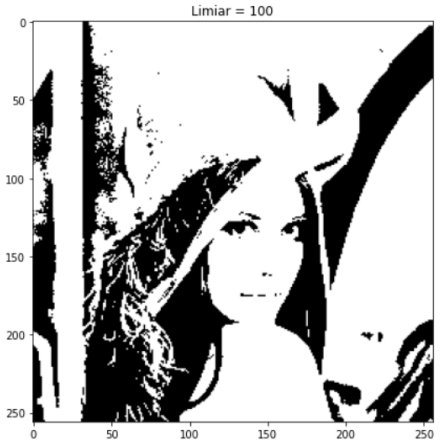
3. Limiarizacao:

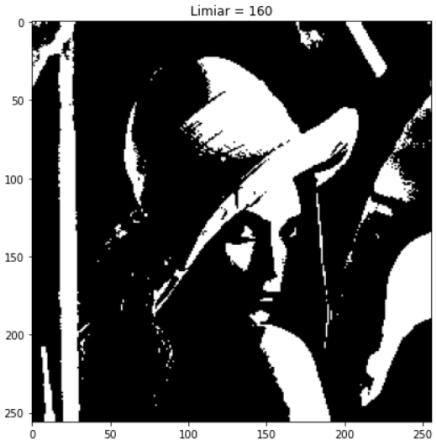
```python
def limiarizacao(img, limiar):

    img_saida = np.zeros(img.shape)

    Tmin = np.min(img)
    Tmax = np.max(img)

    altura, largura = img_saida.shape

    for y in range(0, altura):
        for x in range(0, largura):
            if img[y][x] < limiar:
                img_saida[y, x] = Tmin
            else:
                img_saida[y, x] = Tmax

    img_saida = np.uint8(img_saida)

    return img_saida
```

A função linearização recebe uma imagem e um valor limiar como parâmetro, tira da imagem os seus valores maximo e mínimo de tom de cinza e utilizando o valor limiar como parâmetro para que então todos os níveis acima sejam substituídos pelo valor máximo, e abaixo deste nível pelo valor de mínimo. Abaixo são mostrados histograma e imagem limiazidadas.









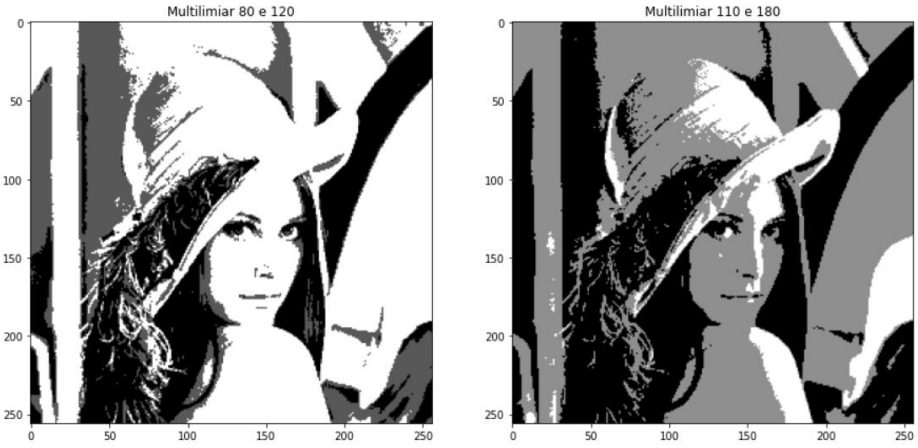
Podemos observar nos resultados que variar o limiar vai resultar numa maior concentração de tons de cinza no valor maximo ou no valor mínimo.

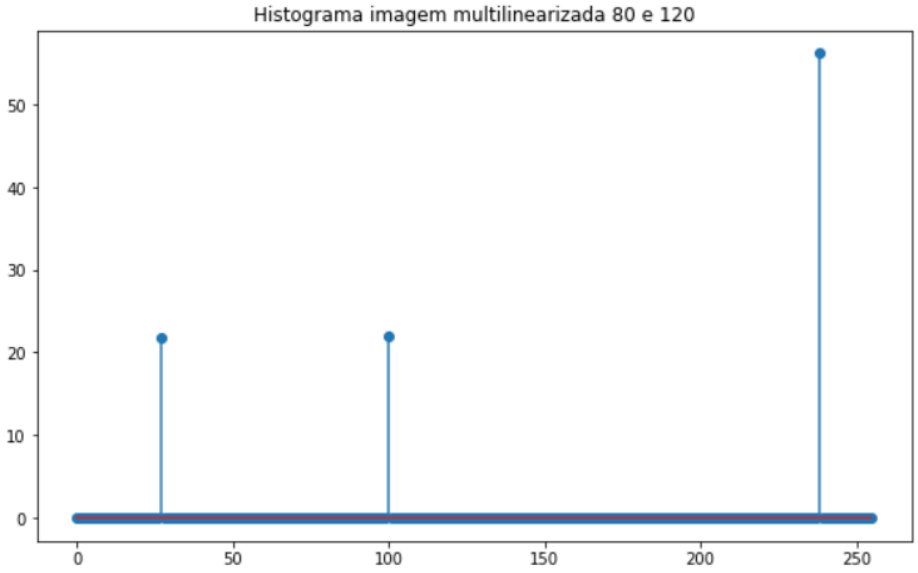
4. Multilimearizações:

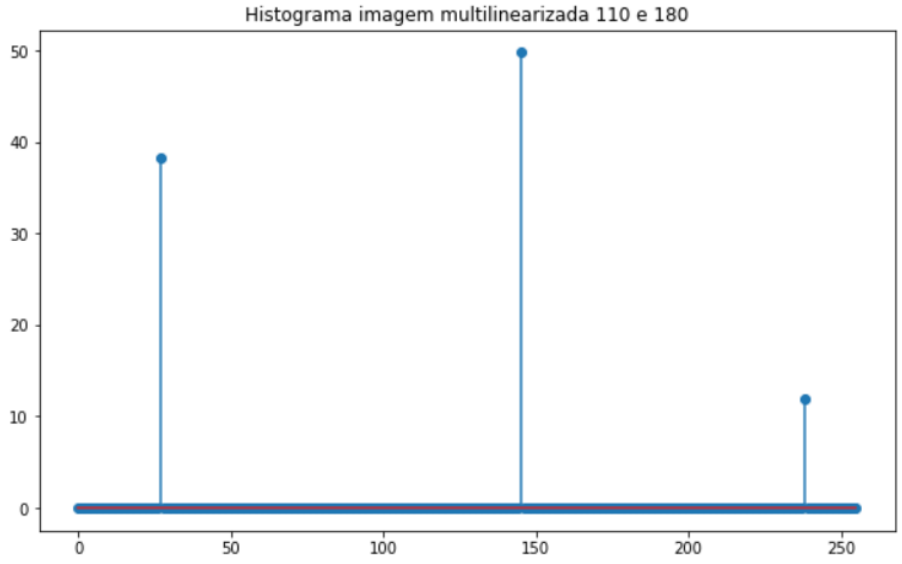
A função multilimearizacao, aplica multilimearização na imagem, que semelhante a linearização contudo com mais de um limiar, recebe como argumento a imagem, um limiar inferior, e um limiar superior, onde todo pixel com valor menor que o limiar inferior ele aplica o valor mínimo, todo valor maior que o limiar superior e aplicado o valor maximo e valor entre o limiar mínimo e maximo ele aplica um valor entre o maximo e mínimo da imagem.

```
def multiLimiarizacao(img, Linf, Lsup):

    img_saida = np.zeros(img.shape)

    Tmin = np.min(img)
    Tmax = np.max(img)

    altura, largura = img_saida.shape

    for y in range(0, altura):
        for x in range(0, largura):

            if img[y][x] > Lsup:
                img_saida[y, x] = Tmax
            elif img[y][x] <= Lsup and img[y][x] > Linf:
                img_saida[y, x] = Linf + int((Lsup - Linf)/2)
            elif img[y][x] <= Linf:
                img_saida[y, x] = Tmin

    img_saida = np.uint8(img_saida)

    return img_saida
```

Abaixo podemos ver o histograma da imagem que passou pelo processo de multilimearização onde pode ser observado que todos os tons de cinza se concentram em três regiões distintas, um tom mais escuro, um mais claro e um intermediário, onde essa concentração e variada pelos valores dos limiares como pode ser observado abaixo nos gráficos de histograma.




Histograma imagem multilinearizada 80 e 120
50
40
30
20
10
0
0
50
100
150
200
250


Histograma imagem multilinearizada 110 e 180
50
40
30
20
10
0
0
50
100
150
200
250