histogram_opencv

September 22, 2022

1 Introdução

Grupo: Germano Barcelos (3873), Guilherme Melos (3882), Jhonata Miranda (3859)

Neste relatório, temos como objetivo aplicar alguns conceitos vistos na disciplina CCF 394, Processamento Digital de Imagens. Podemos dividir o conteúdo em criação e visualização de histogramas, Equalização, brilho e contraste e thresholding.

O histograma é uma ferramenta da qualidade muito importante para análises estatísticas. É um gráfico que mostra a distribuição de acontecimentos registrados em todo o espectro. Na equalização do histograma, pretende-se que o histograma final da imagem seja o mais constante possível, distribuindo concentrações de brilhos como acontece nos gráficos. Deve-se notar que a operação de equalização de histograma só conduziria a histogramas efectivamente constantes se a gama de brilhos fosse contínua e fosse infinito o número de pontos na imagem. No caso das imagens digitais, a discretização do espaço e dos brilhos não permite que esse resultado seja conseguido, como podemos ver pelos dois histogramas, o inicial e o obtido após uma operação de equalização.

Já a alteração do brilho e do contraste de uma imagem, consiste na aplicação de uma fórmula linear, cujo objetivo é alterar o valor de todos os pixels de uma imagem e também a forma que sua distribuição se apresenta em um histograma. Já os métodos de thresholding, ou limiarização, têm como objetivo segmentar uma imagem em conjuntos de pixels ou objetos com objetivo de mudar a representação de uma imagem. Os metodos de thresholding que serão apresentados aqui são: o thresholding com limiar calculado à partir do método de Otsu e o thresholding adaptativo.

2 Desenvolvimento

Iremos, neste relatório, utilizar o processamento digital de imagens para: gerar histogramas da distribuição de pixels em uma imagem, aplicar a equalização no histograma e na imagem, modificar o contraste, gerar trackbars para alterar o contraste e brilho da imagem e aplicar métodos de Thresholding como o Otsu Thresholding e o Adaptative Thresholding.

2.1 Imagem e Histograma

Para utilizar os recursos para modificar e extrair dados das imagens, é necessário importar as bibliotecas OpenCV, Numpy e MatPlotLib. Do módulo `__future__`, também importamos `print_function` e `division`.

[12]:
```
from __future__ import print_function
from __future__ import division
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
```

Já para mostrar as imagens, plotando-as no notebook, precisamos definir a função `show_img` que plota as imagens normalmente e em escala de cinza:

A função abaixo nomeada de `show_img` tem como objetivo a generalização de mostrar imagens de acordo com sua assinatura `show_img(titulo, img, cmap=None)`. O primeiro parâmetro tem relação ao título da figura, ou seja, quando a imagem for mostrada, utilizando a biblioteca `matplotlib`, o título será igual a `titulo` (é esperado que seja uma cadeia de caracteres). O segundo parâmetro dessa função recebe um array de n dimensões, as quais representam os canais das imagens. O `cmap` é um parâmetro que define qual mapeamento de cores vai ser utilizado.

[ ]:
```
def show_img(titulo, img, cmap=None):
    if cmap is not None:
        plt.imshow(img, cmap=cmap)
    else:
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        plt.imshow(img)
    plt.title(titulo)
    plt.axis("off")
    plt.show()
```

Abaixo, salvamos em `src` a imagem da lena.

[ ]:
```
src = cv.imread("lena.jpg")
```

A partir da imagem `lena.jgp`, dividimos a mesma no modelo BGR.

[ ]:
```
bgr_planes = cv.split(src)
```

Já para criar o histograma, temos que definir suas dimensões, salvando-as em `histSize`, a quantidade de linhas do histograma, e `histRange`, e definir o parâmetro `accumulate`.

[ ]:
```
# [Establish the number of bins]
histSize = 256

## [Set the ranges ( for B,G,R) )]
histRange = (0, 256) # the upper boundary is exclusive

## [Set histogram param]
accumulate = False
```

Agora, utilizando `calcHist` da OpenCV, criamos um histograma para cada uma das bandas BGR. Como parâmetros para cada um dos histogramas, passamos a imagem dividida, a posição do vetor

onde está a faixa desejada, `None` para não aplicar nenhuma máscara, as dimensões que salvamos anteriormente e o parâmetro `accumulate` que foi definido como `False`.

[ ]:
```
b_hist = cv.calcHist(bgr_planes, [0], None, [histSize], histRange,␣
 ↪accumulate=accumulate)
g_hist = cv.calcHist(bgr_planes, [1], None, [histSize], histRange,␣
 ↪accumulate=accumulate)
r_hist = cv.calcHist(bgr_planes, [2], None, [histSize], histRange,␣
 ↪accumulate=accumulate)
```

Para os histogramas de cada banda foram geradas 3 matrizes correspondentes ao BGR, ou seja, `b_hist` corresponde ao histograma do pixel azul na imagem analisada e assim por diante. No entanto, cada pixel, varia de 0 a 255, como queremos mostrar uma imagem com maiores dimensões é necessário normalizar/padronizar para uma diferente proporção. Logo a operação de normalizar os histogramas é feita utilizando a função `normalize`, com o parâmetro de `NORM_MINMAX` que significa que a padronização será feita utilizando valores de min e max na proporção.

$$Normalization(x) = \frac{x - min}{max - min}$$

[ ]:
```
## [Draw the histograms for B, G and R]
hist_w = 512
hist_h = 400
bin_w = int(round( hist_w/histSize ))

histImage = np.zeros((hist_h, hist_w, 3), dtype=np.uint8)

## [Normalize the result to ( 0, histImage.rows )]
cv.normalize(b_hist, b_hist, alpha=0, beta=hist_h, norm_type=cv.NORM_MINMAX)
cv.normalize(g_hist, g_hist, alpha=0, beta=hist_h, norm_type=cv.NORM_MINMAX)
cv.normalize(r_hist, r_hist, alpha=0, beta=hist_h, norm_type=cv.NORM_MINMAX)

for i in range(1, histSize):
    cv.line(histImage, ( bin_w*(i-1), hist_h - int((b_hist[i-1])) ),
            ( bin_w*(i), hist_h - int((b_hist[i])) ),
            ( 255, 0, 0), thickness=2)
    cv.line(histImage, ( bin_w*(i-1), hist_h - int((g_hist[i-1])) ),
            ( bin_w*(i), hist_h - int((g_hist[i])) ),
            ( 0, 255, 0), thickness=2)
    cv.line(histImage, ( bin_w*(i-1), hist_h - int((r_hist[i-1])) ),
            ( bin_w*(i), hist_h - int((r_hist[i])) ),
            ( 0, 0, 255), thickness=2)

show_img('Source image', src)
show_img('calcHist Demo', histImage)
```

2.2 Equalizador

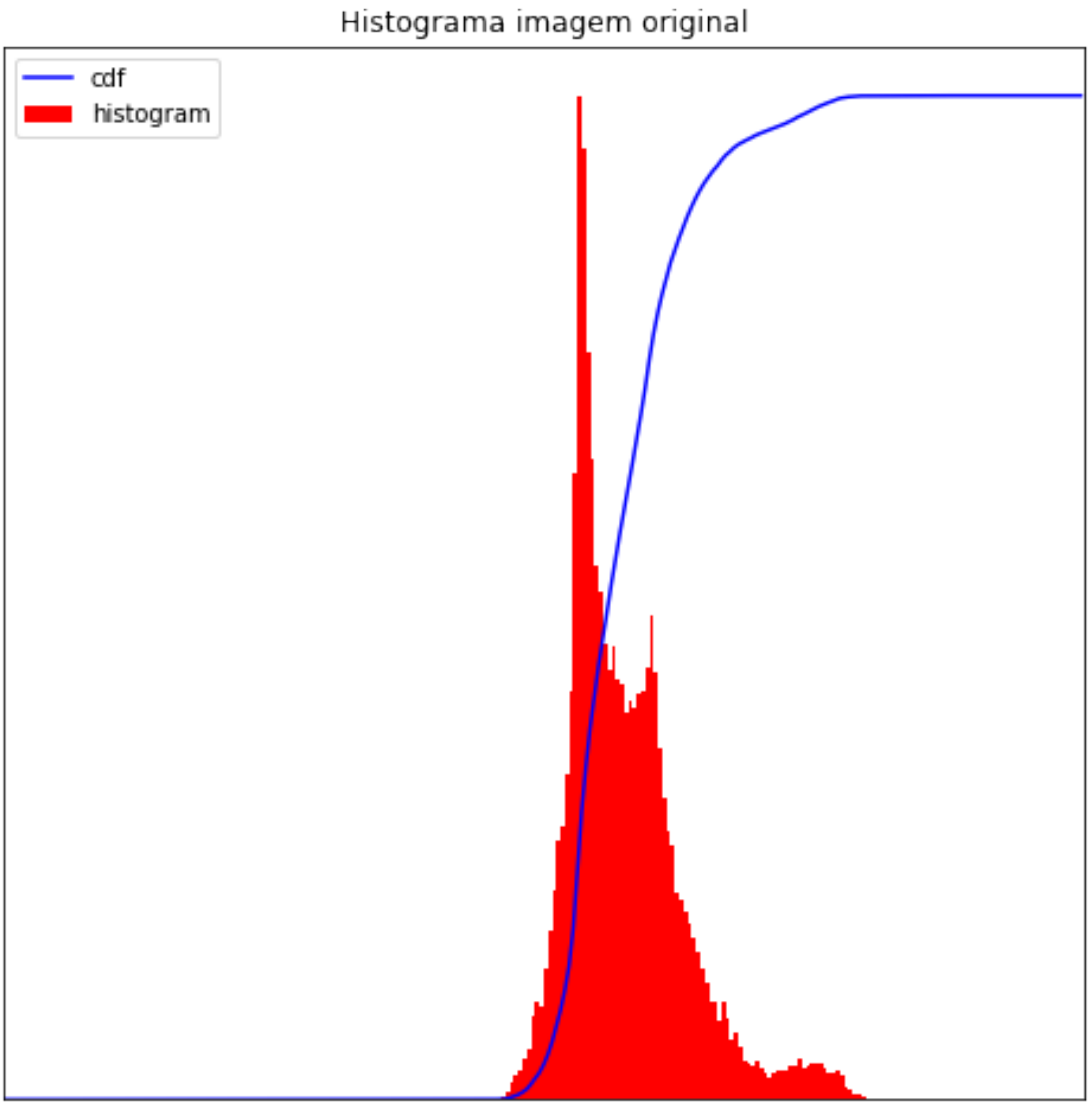
O Equalizador recebe um histograma e muda a distribuição de valores para reduzir diferenças acentuadas. Utilizamos a imagem `wiki.png` que está em escala de cinza para aplicar a equalização. Salvamos esta imagem na variável `img` e e a imagem com o histograma equalizado em `equ`. Para aplicar esta modificação, utilizamos o método `equalizeHist` que: calcula o histograma da imagem, normaliza o histograma de acordo com uma Look-Up Table. Com a equalização, ocorre, geralmente a normalização no brilho e aumenta o contraste da imagem, conforme a tabela de mapeamento.

[11]:
```
img = cv.imread('wiki.png',0)
equ = cv.equalizeHist(img)
```

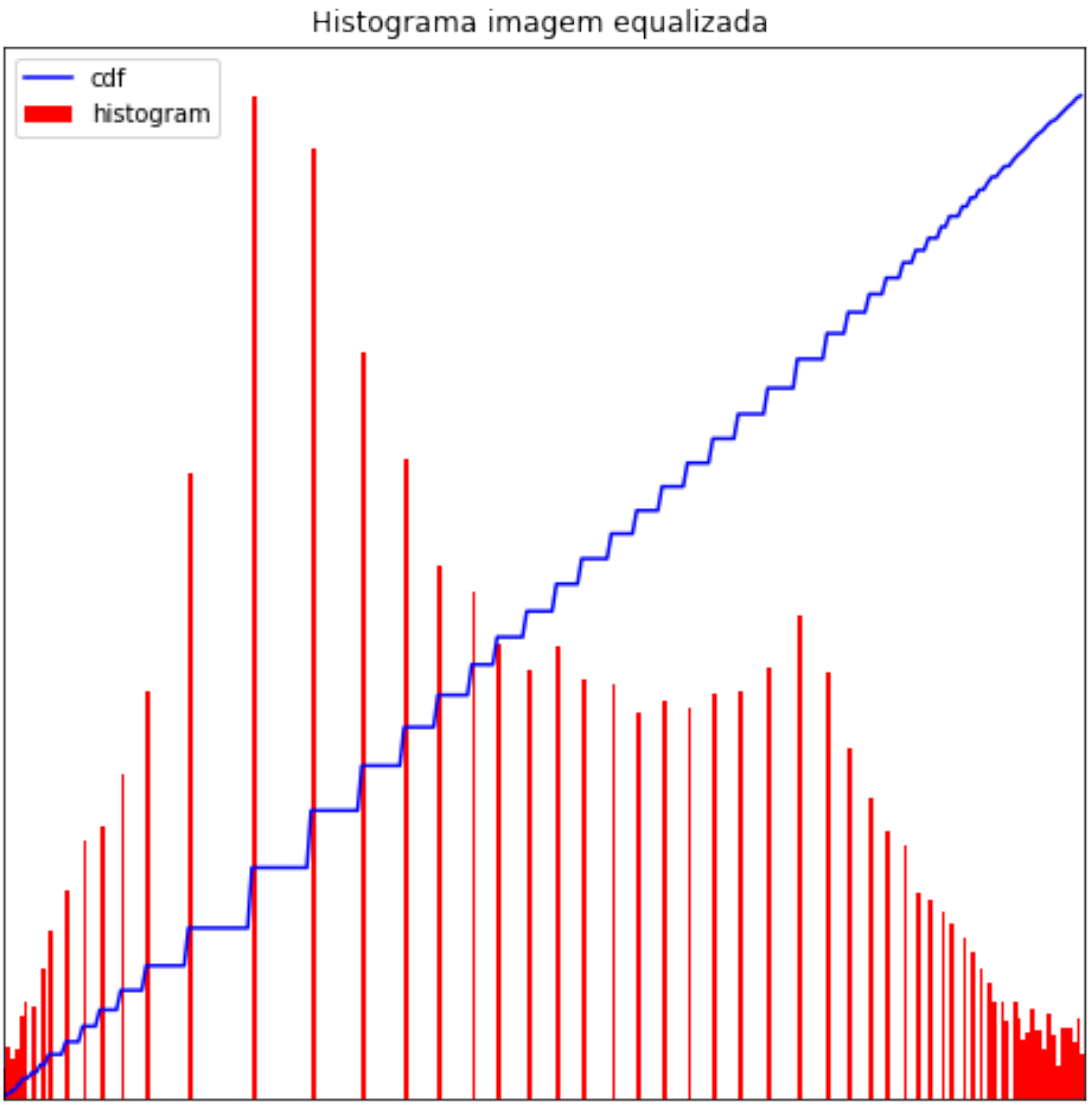
Nas células abaixo, foram plotadas o histograma da imagem original e da imagem equalizada. Note que na imagem equalizada, a função de distribuição cumulativa é formada quase por uma step function. Isso se deve ao fato de que alguns pixels serem transformados em outros de acordo com determinado intervalo, e por isso, a função de distribuição não sofre alterações nesses intervalos. A imagem do histograma equalizado, mostra também que comparado ao histograma original, temos uma distribuição mais uniforme dos pixels.

[13]:
```
#histograma e cdf  da imagem original
hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()
cdf_normalized = cdf * hist.max()/ cdf.max()

fig = plt.figure(figsize=(18, 18))
plt.subplot(221)
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.title('Histograma imagem original'), plt.xticks([]), plt.yticks([])
```

[13]:
```
(Text(0.5, 1.0, 'Histograma imagem original'), ([], []), ([], []))
```



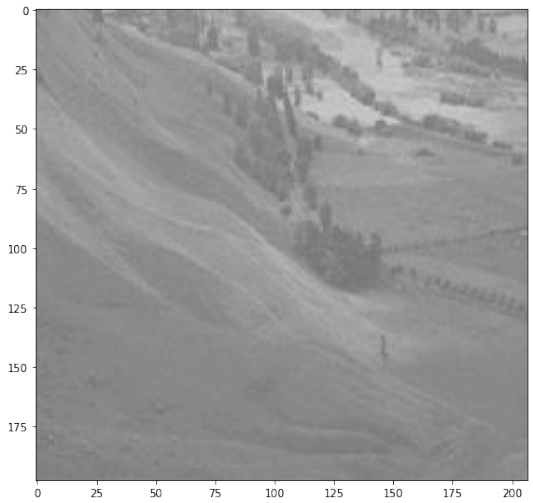
[9]:
```
#histograma e cdf da imagem equalizada
hist,bins = np.histogram(equ.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * hist.max()/ cdf.max()

fig = plt.figure(figsize=(18, 18))
plt.plot(cdf_normalized, color = 'b')
plt.hist(equ.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.title('Histograma imagem equalizada '), plt.xticks([]), plt.yticks([])
plt.show()
```

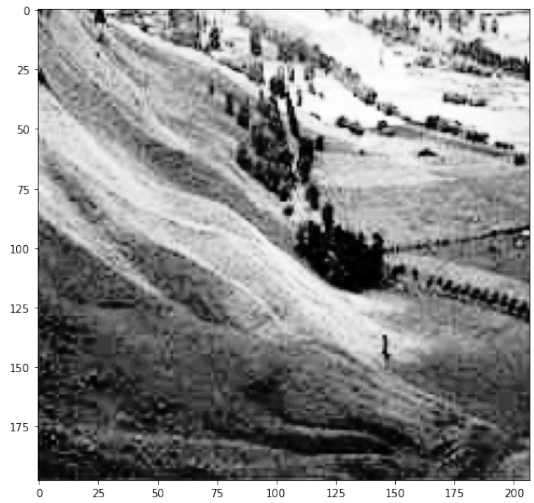


A célula abaixo apenas mostra as imagens original e a equalizada.

[22]:
```
# o formato opencv é BGR, e o plot usa RGB, entao usa COLOR_BGR2RGB
f, axarr = plt.subplots(1,2, figsize=(18,18))
axarr[0].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
axarr[1].imshow(cv.cvtColor(equ, cv.COLOR_BGR2RGB))
```

[22]:
```
<matplotlib.image.AxesImage at 0x7f521df4aca0>

<Figure size 1296x1296 with 0 Axes>
```





2.3 Transformação Linear

No caso da transformação linear, temos a mudança dos pixels de acordo com uma função.

Na célula abaixo, a imagem é lida utilizando a função `cv.imread("lena.jpg", cv.IMREAD_GRAYSCALE)`

[14]:
```
image = cv.imread("lena.jpg",cv.IMREAD_GRAYSCALE)
print(image.shape)
try:
    x,y,z=image.shape
except ValueError:
    print("imagem 2d")
    img2d=True

min=np.min(image)
max=np.max(image)
print(max,min)
```

```
(512, 512)
imagem 2d
245 21
```

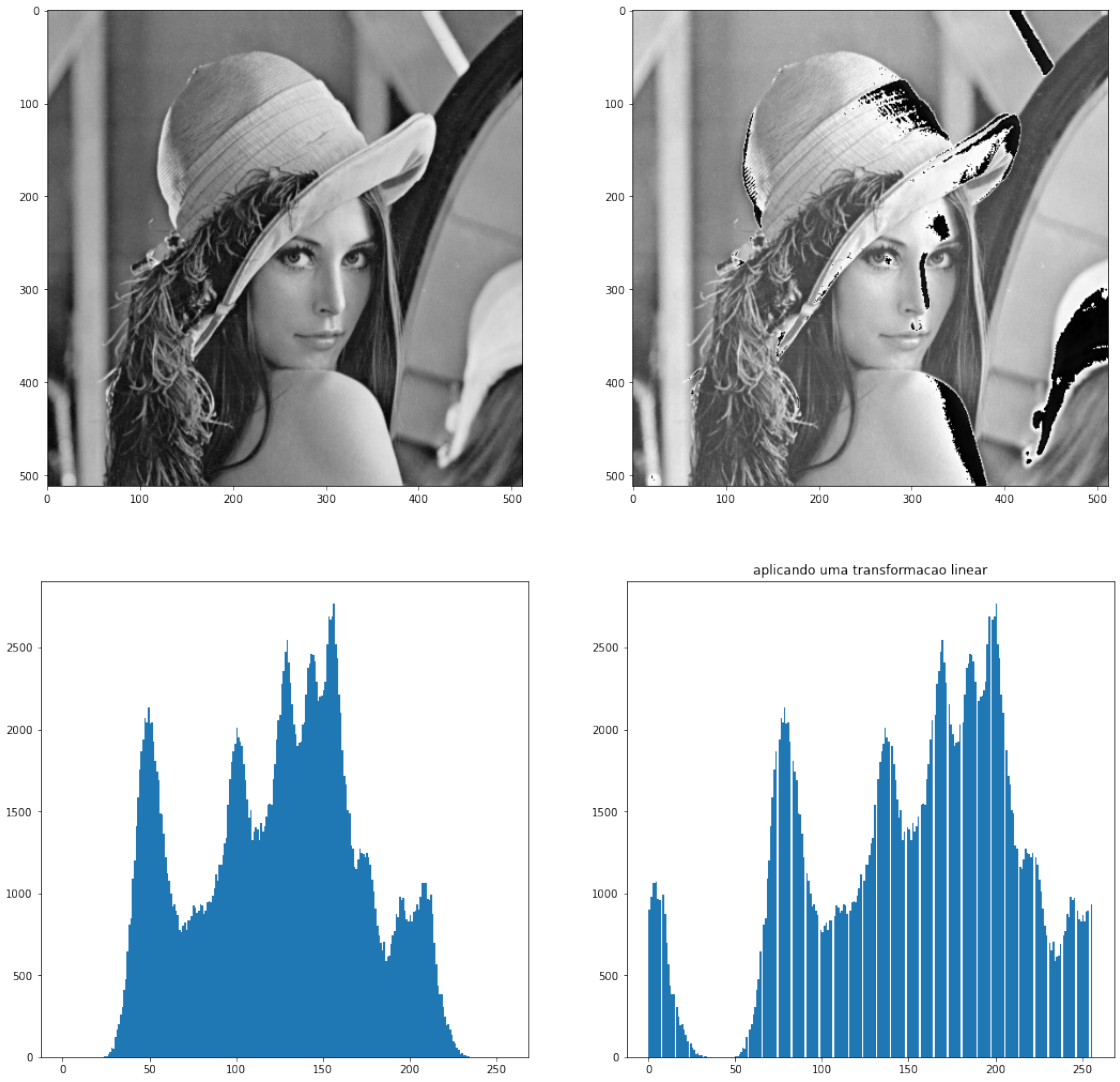
Na célula abaixo, um deslocamento de 20 posições, ou seja, se o pixel $p_i = 20$, valerá $p_i = 40$. Caso, o pixel p_i ultrapasse o valor máximo $v_{max} = 255$, a informação daquele pixel é perdida, resultando em falhas de preto na imagem, como visto abaixo. Para fazer o deslocamento, basta somar 20, nesse caso, para cada posição da matriz que representa a imagem; como estamos utilizando o *numpy*, há um broadcasting da operação simples de soma para todas as posições.

[26]:
```
# um deslocamento de 20 na imagem
nova=(image+20)*(255/(max-min))
nova=np.uint8(nova)
print(np.max(nova),np.min(nova))

f, axarr = plt.subplots(2,2, figsize=(18,18))
axarr[0,0].imshow(image,cmap='gray')
axarr[0,1].imshow(nova,cmap='gray')

axarr[1,0].hist(image.ravel(),256,[0,256]),plt.title('Histograma para uma␣
 ↪imagem em tons de cinza')
axarr[1,1].hist(nova.ravel(),256,[0,256]),plt.title('aplicando uma␣
 ↪transformacao linear')
```

```
255 0
```

[26]:
```
((array([9.020e+02, 9.760e+02, 1.064e+03, 1.066e+03, 1.067e+03, 9.640e+02,
         9.590e+02, 0.000e+00, 9.910e+02, 8.720e+02, 7.000e+02, 5.700e+02,
         4.360e+02, 3.850e+02, 3.810e+02, 0.000e+00, 3.050e+02, 2.470e+02,
         1.960e+02, 2.040e+02, 1.690e+02, 1.330e+02, 9.500e+01, 0.000e+00,
         8.100e+01, 6.000e+01, 4.600e+01, 2.100e+01, 2.300e+01, 1.200e+01,
         1.100e+01, 0.000e+00, 7.000e+00, 7.000e+00, 1.000e+00, 0.000e+00,
         0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
         0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 1.000e+00, 0.000e+00,
         0.000e+00, 0.000e+00, 2.000e+00, 3.000e+00, 1.700e+01, 3.100e+01,
         5.600e+01, 5.300e+01, 1.200e+02, 0.000e+00, 1.670e+02, 2.030e+02,
         2.630e+02, 3.070e+02, 4.090e+02, 4.740e+02, 6.470e+02, 0.000e+00,
         8.100e+02, 8.490e+02, 1.087e+03, 1.199e+03, 1.410e+03, 1.588e+03,
         1.758e+03, 1.864e+03, 0.000e+00, 1.937e+03, 2.069e+03, 2.041e+03,
         2.132e+03, 2.038e+03, 2.042e+03, 1.925e+03, 0.000e+00, 1.807e+03,
         1.745e+03, 1.688e+03, 1.489e+03, 1.481e+03, 1.366e+03, 1.221e+03,
         0.000e+00, 1.125e+03, 1.077e+03, 9.960e+02, 9.220e+02, 9.340e+02,
         8.940e+02, 8.660e+02, 0.000e+00, 7.790e+02, 7.610e+02, 8.020e+02,
         8.240e+02, 7.750e+02, 8.350e+02, 8.370e+02, 0.000e+00, 8.600e+02,
         9.230e+02, 9.160e+02, 8.830e+02, 8.960e+02, 9.310e+02, 9.240e+02,
         8.720e+02, 0.000e+00, 8.920e+02, 9.440e+02, 9.530e+02, 9.480e+02,
         9.830e+02, 1.031e+03, 1.113e+03, 0.000e+00, 1.077e+03, 1.172e+03,
         1.172e+03, 1.235e+03, 1.304e+03, 1.339e+03, 1.543e+03, 0.000e+00,
         1.694e+03, 1.803e+03, 1.867e+03, 1.912e+03, 2.013e+03, 1.948e+03,
         1.927e+03, 0.000e+00, 1.899e+03, 1.786e+03, 1.687e+03, 1.572e+03,
         1.460e+03, 1.507e+03, 1.322e+03, 1.378e+03, 0.000e+00, 1.403e+03,
         1.388e+03, 1.325e+03, 1.427e+03, 1.374e+03, 1.409e+03, 1.469e+03,
         0.000e+00, 1.539e+03, 1.549e+03, 1.540e+03, 1.696e+03, 1.789e+03,
         1.936e+03, 2.056e+03, 0.000e+00, 2.088e+03, 2.275e+03, 2.355e+03,
         2.476e+03, 2.545e+03, 2.409e+03, 2.283e+03, 0.000e+00, 2.151e+03,
         2.032e+03, 1.971e+03, 1.901e+03, 1.920e+03, 1.922e+03, 2.031e+03,
         0.000e+00, 2.040e+03, 2.214e+03, 2.373e+03, 2.402e+03, 2.460e+03,
```

```
       2.456e+03, 2.412e+03, 2.288e+03, 0.000e+00, 2.175e+03, 2.199e+03,
       2.209e+03, 2.239e+03, 2.291e+03, 2.519e+03, 2.687e+03, 0.000e+00,
       2.671e+03, 2.689e+03, 2.766e+03, 2.522e+03, 2.437e+03, 2.214e+03,
       2.099e+03, 0.000e+00, 1.874e+03, 1.714e+03, 1.662e+03, 1.506e+03,
       1.490e+03, 1.294e+03, 1.275e+03, 0.000e+00, 1.163e+03, 1.150e+03,
       1.205e+03, 1.271e+03, 1.243e+03, 1.237e+03, 1.219e+03, 1.244e+03,
       0.000e+00, 1.219e+03, 1.172e+03, 1.084e+03, 1.009e+03, 9.040e+02,
       8.000e+02, 7.440e+02, 0.000e+00, 6.980e+02, 6.520e+02, 7.010e+02,
       5.860e+02, 6.100e+02, 6.180e+02, 6.940e+02, 0.000e+00, 7.410e+02,
       7.700e+02, 8.750e+02, 8.520e+02, 9.760e+02, 9.590e+02, 9.730e+02,
       0.000e+00, 8.960e+02, 8.430e+02, 8.290e+02, 8.690e+02, 8.280e+02,
       8.850e+02, 8.910e+02, 0.000e+00, 9.350e+02]),
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
        99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
       110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
       121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
       132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
       143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
       154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
       165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
       176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
       187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
       198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
       209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
       220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
       231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
       242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
       253., 254., 255., 256.]),
 <BarContainer object of 256 artists>),
Text(0.5, 1.0, 'aplicando uma transformacao linear'))
```



Abaixo, a mesma técnica de transformação linear é feita, porém utilizando a função $g(x) = \alpha * f(x) + \beta$, sendo α o controle do contraste e o β o controle do brilho. As entradas são pedidas ao usuário por meio de *input*.

A função `np.clip(array, min, max)` faz a restrição da transformação. Será aplicado às todas posições o novo `array` se os pixels tiverem o valor entre 0 e 255.

[29]:
```
# agora, lendo os valores de contraste e brilho do teclado g(x) = alfa*f(x) +␣
 ↪beta
new_image = np.zeros(image.shape, image.dtype)
alpha = 1.0 # Simple contrast control
beta = 0    # Simple brightness control

# Initialize values
print(' Basic Linear Transforms ')
```

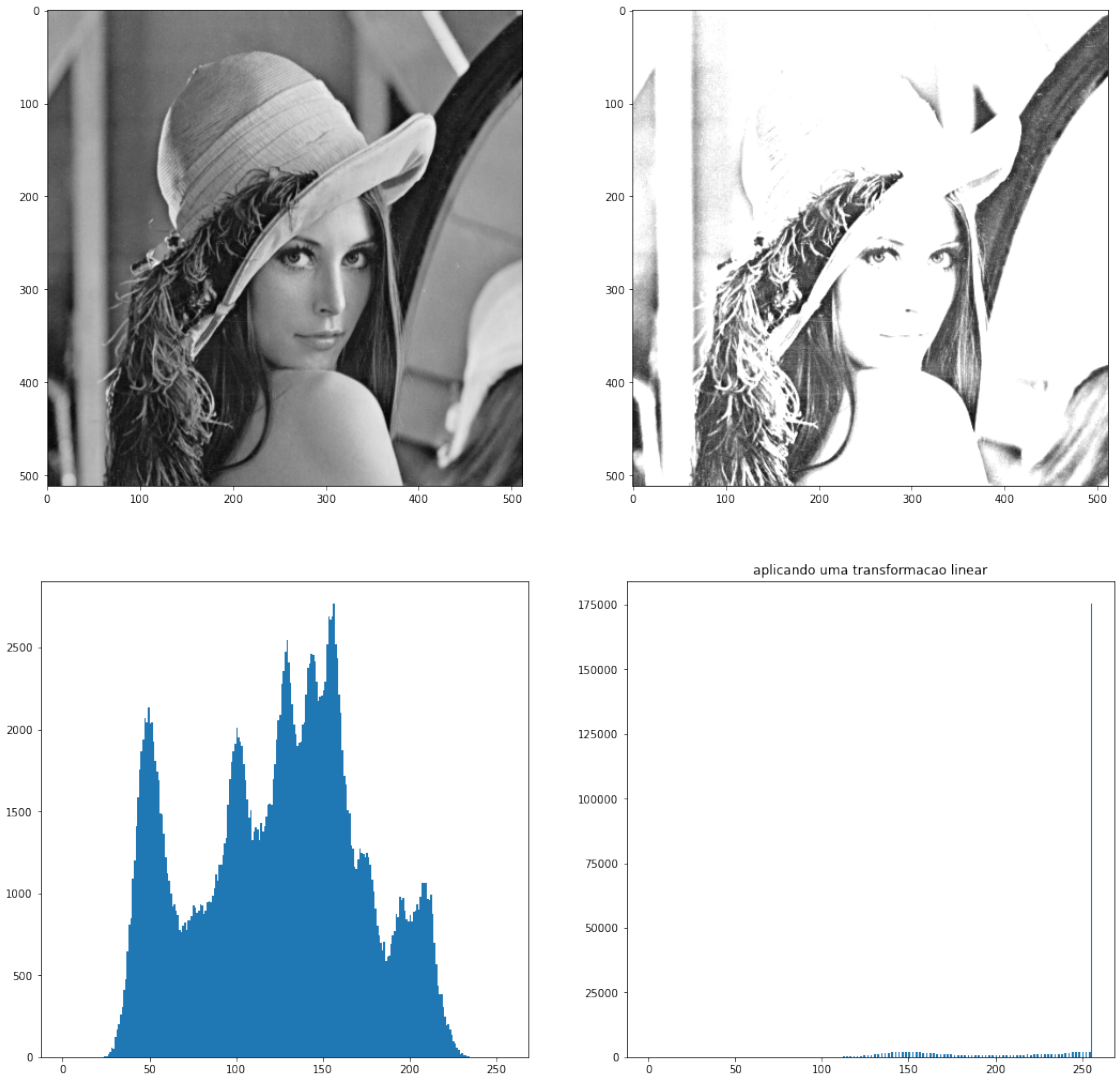
```
print('-------------------------')

try:
    alpha = float(input('* Enter the alpha value [1.0-3.0]: '))
    beta = int(input('* Enter the beta value [0-100]: '))
except ValueError:
    print('Error, not a number')

# Do the operation new_image(i,j) = alpha*image(i,j) + beta
# Instead of these 'for' loops we could have used simply:
# new_image = cv.convertScaleAbs(image, alpha=alpha, beta=beta)
# but we wanted to show you how to access the pixels :)
for y in range(image.shape[0]):
    for x in range(image.shape[1]):
        if (img2d):
            new_image[y,x] = np.clip(alpha*image[y,x] + beta, 0, 255)
        else:

            for c in range(image.shape[2]):
                new_image[y,x,c] = np.clip(alpha*image[y,x,c] + beta, 0, 255)

f, axarr = plt.subplots(2,2, figsize=(18,18))
axarr[0,0].imshow(image,cmap='gray')
axarr[0,1].imshow(new_image,cmap='gray')

axarr[1,0].hist(image.ravel(),256,[0,256]),plt.title('Histograma para uma␣
 ↪imagem em tons de cinza')
axarr[1,1].hist(new_image.ravel(),256,[0,256]),plt.title('aplicando uma␣
 ↪transformacao linear')
```

```
 Basic Linear Transforms
-------------------------
```

[29]:
```
((array([0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
         0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
```

```
       0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
       0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
       0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
       0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
       0.00000e+00, 0.00000e+00, 1.00000e+00, 0.00000e+00, 0.00000e+00,
       0.00000e+00, 0.00000e+00, 0.00000e+00, 2.00000e+00, 0.00000e+00,
       3.00000e+00, 0.00000e+00, 1.70000e+01, 0.00000e+00, 3.10000e+01,
       0.00000e+00, 5.60000e+01, 0.00000e+00, 5.30000e+01, 0.00000e+00,
       1.20000e+02, 0.00000e+00, 1.67000e+02, 0.00000e+00, 2.03000e+02,
       0.00000e+00, 2.63000e+02, 0.00000e+00, 3.07000e+02, 0.00000e+00,
       4.09000e+02, 0.00000e+00, 4.74000e+02, 0.00000e+00, 6.47000e+02,
       0.00000e+00, 8.10000e+02, 0.00000e+00, 8.49000e+02, 0.00000e+00,
       1.08700e+03, 0.00000e+00, 1.19900e+03, 0.00000e+00, 1.41000e+03,
       0.00000e+00, 1.58800e+03, 0.00000e+00, 1.75800e+03, 0.00000e+00,
       1.86400e+03, 0.00000e+00, 1.93700e+03, 0.00000e+00, 2.06900e+03,
       0.00000e+00, 2.04100e+03, 0.00000e+00, 2.13200e+03, 0.00000e+00,
       2.03800e+03, 0.00000e+00, 2.04200e+03, 0.00000e+00, 1.92500e+03,
       0.00000e+00, 1.80700e+03, 0.00000e+00, 1.74500e+03, 0.00000e+00,
       1.68800e+03, 0.00000e+00, 1.48900e+03, 0.00000e+00, 1.48100e+03,
       0.00000e+00, 1.36600e+03, 0.00000e+00, 1.22100e+03, 0.00000e+00,
       1.12500e+03, 0.00000e+00, 1.07700e+03, 0.00000e+00, 9.96000e+02,
       0.00000e+00, 9.22000e+02, 0.00000e+00, 9.34000e+02, 0.00000e+00,
       8.94000e+02, 0.00000e+00, 8.66000e+02, 0.00000e+00, 7.79000e+02,
       0.00000e+00, 7.61000e+02, 0.00000e+00, 8.02000e+02, 0.00000e+00,
       8.24000e+02, 0.00000e+00, 7.75000e+02, 0.00000e+00, 8.35000e+02,
       0.00000e+00, 8.37000e+02, 0.00000e+00, 8.60000e+02, 0.00000e+00,
       9.23000e+02, 0.00000e+00, 9.16000e+02, 0.00000e+00, 8.83000e+02,
       0.00000e+00, 8.96000e+02, 0.00000e+00, 9.31000e+02, 0.00000e+00,
       9.24000e+02, 0.00000e+00, 8.72000e+02, 0.00000e+00, 8.92000e+02,
       0.00000e+00, 9.44000e+02, 0.00000e+00, 9.53000e+02, 0.00000e+00,
       9.48000e+02, 0.00000e+00, 9.83000e+02, 0.00000e+00, 1.03100e+03,
       0.00000e+00, 1.11300e+03, 0.00000e+00, 1.07700e+03, 0.00000e+00,
       1.17200e+03, 0.00000e+00, 1.17200e+03, 0.00000e+00, 1.23500e+03,
       0.00000e+00, 1.30400e+03, 0.00000e+00, 1.33900e+03, 0.00000e+00,
       1.54300e+03, 0.00000e+00, 1.69400e+03, 0.00000e+00, 1.80300e+03,
       0.00000e+00, 1.86700e+03, 0.00000e+00, 1.91200e+03, 0.00000e+00,
       2.01300e+03, 0.00000e+00, 1.94800e+03, 0.00000e+00, 1.92700e+03,
       1.75343e+05]),
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
        11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
        22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
        33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
        44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
        55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
        66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
        77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
        88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
```

```
         99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
        110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
        121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
        132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
        143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
        154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
        165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
        176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
        187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
        198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
        209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,
        220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
        231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
        242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
        253., 254., 255., 256.]),
 <BarContainer object of 256 artists>),
Text(0.5, 1.0, 'aplicando uma transformacao linear'))
```



2.4 Trackbar Contraste e Brilho

Abaixo mostraremos como criar trackbars para ajustar o brilho e o contraste. Reforçando o que foi dito na seção anterior, utilizando a função $g(x) = \alpha * f(x) + \beta$, o coeficiente α tem o objetivo de ajustar o contraste da imagem e o coeficiente β ajustar o brilho. Com isso, para obter o valor de cada coeficiente da função, utilizamos uma trackbar. A célula abaixo define uma função `histograma` para calcular o histograma da imagem, como explicado na Seção Imagem e Histograma. Além disso define uma função `BrilhoContraste` que, dada a posição das duas trackbars obtidas pelo método `getTrackbarPos`, salva na variável `effect` a imagem com os efeitos aplicados pelo `controller`. Esse método faz o cálculo dos valores α e β.

[1]:
```
import cv2
import numpy as np
```

```
def histograma(src):

  ## [Separate the image in 3 places ( B, G and R )]
  bgr_planes = cv2.split(src)
  ## [Separate the image in 3 places ( B, G and R )]

  ## [Establish the number of bins]
  histSize = 256
  ## [Establish the number of bins]

  ## [Set the ranges ( for B,G,R) )]
  histRange = (0, 256) # the upper boundary is exclusive
  ## [Set the ranges ( for B,G,R) )]

  ## [Set histogram param]
  accumulate = False
  ## [Set histogram param]

  ## [Compute the histograms]
  b_hist = cv2.calcHist(bgr_planes, [0], None, [histSize], histRange,␣
↪accumulate=accumulate)
  g_hist = cv2.calcHist(bgr_planes, [1], None, [histSize], histRange,␣
↪accumulate=accumulate)
  r_hist = cv2.calcHist(bgr_planes, [2], None, [histSize], histRange,␣
↪accumulate=accumulate)
  ## [Compute the histograms]

  ## [Draw the histograms for B, G and R]
  hist_w = 512
  hist_h = 400
  bin_w = int(round( hist_w/histSize ))

  histImage = np.zeros((hist_h, hist_w, 3), dtype=np.uint8)
  ## [Draw the histograms for B, G and R]

  ## [Normalize the result to ( 0, histImage.rows )]
  cv2.normalize(b_hist, b_hist, alpha=0, beta=hist_h, norm_type=cv2.NORM_MINMAX)
  cv2.normalize(g_hist, g_hist, alpha=0, beta=hist_h, norm_type=cv2.NORM_MINMAX)
  cv2.normalize(r_hist, r_hist, alpha=0, beta=hist_h, norm_type=cv2.NORM_MINMAX)
  ## [Normalize the result to ( 0, histImage.rows )]
  ## [Draw for each channel]
  for i in range(1, histSize):
      cv2.line(histImage, ( bin_w*(i-1), hist_h - int((b_hist[i-1])) ),
               ( bin_w*(i), hist_h - int((b_hist[i])) ),
               ( 255, 0, 0), thickness=2)
      cv2.line(histImage, ( bin_w*(i-1), hist_h - int((g_hist[i-1])) ),
```

```
              ( bin_w*(i), hist_h - int((g_hist[i])) ),
              ( 0, 255, 0), thickness=2)
      cv2.line(histImage, ( bin_w*(i-1), hist_h - int((r_hist[i-1])) ),
              ( bin_w*(i), hist_h - int((r_hist[i])) ),
              ( 0, 0, 255), thickness=2)
  ## [Draw for each channel]
  ## [Display]
  plt.imshow('calcHist Demo', histImage)

  ## [Display]

def BrilhoContraste(Brilho=0):

    # getTrackbarPos returns the
    # current position of the specified trackbar.
    Brilho = cv2.getTrackbarPos('Brilho','CCF394')

    Contraste = cv2.getTrackbarPos('Contraste','CCF394')

    effect = controller(img,Brilho, Contraste)

    plt.imshow('Effect', effect)
    histograma(effect)

def controller(img, Brilho=255, Contraste=127):
    Brilho = int((Brilho - 0) * (255 - (-255)) / (510 - 0) + (-255))
    Contraste = int((Contraste - 0) * (127 - (-127)) / (254 - 0) + (-127))

    if Brilho != 0:
         if Brilho > 0:
             shadow = Brilho
             max = 255
         else:
            shadow = 0
            max = 255 + Brilho
         al_pha = (max - shadow) / 255
         ga_mma = shadow

        # The function addWeighted calculates the weighted sum of two arrays
         cal = cv2.addWeighted(img, al_pha, img, 0, ga_mma)

    else:
        cal = img

    if Contraste != 0:
        Alpha = float(131 * (Contraste + 127)) / (127 * (131 - Contraste))
        Gamma = 127 * (1 - Alpha)
```

```
        cal = cv2.addWeighted(cal, Alpha, cal, 0, Gamma)
```

Aqui, aplicamos o que foi feito acima. Primeiramente, estaremos utilizando a imagem `lena.jpg`, salvamos a imagem original em `original` e uma cópia dela em `img`. Criamos uma janela, chamada `CCF394`, para exibir as imagens. Mostramos a imagem original e em seguida, criamos uma trackbar para obter o brilho e outra para obter o contraste. O brilho pode variar entre -255 e 255, já o contraste, pode variar entre -127 e 127. Por fim, utilzamos a função `BrilhoContraste` para aplicar os efeitos.

[ ]:
```
original = cv2.imread("lena.jpg")

# Making another copy of an image.
img = original.copy()

# The function namedWindow creates
# a window that can be used as
# a placeholder for images.
cv2.namedWindow('CCF394')


plt.imshow('CCF394', original)

# createTrackbar(trackbarName, windowName, value, count, onChange)
# Brilho range -255 to 255
cv2.createTrackbar('Brilho', 'CCF394',255, 2 * 255, BrilhoContraste)

# Contraste range -127 to 127
cv2.createTrackbar('Contraste', 'CCF394',127, 2 * 127, BrilhoContraste)

BrilhoContraste(0)
```

```
Canceled future for execute_request message before replies were done
```

```
The Kernel crashed while executing code in the the current cell or a␣
↪previous cell. Please review the code in the cell(s) to identify a possible␣
↪cause of the failure. Click <a href='https://aka.ms/
↪vscodeJupyterKernelCrash'>here</a> for more info. View Jupyter <a␣
↪href='command:jupyter.viewOutput'>log</a> for further details.
```

2.5 Thresholding

A técnica de plotar histograma é interessante para ver diferenças na imagem e dividir em duas classes. Se o histograma tiver uma divisão clara entre duas classes é possível definir um limit (threshold) T para: se um pixel p_{ij} é menor que T, então p_{ij} será classificado como classe 0, senão será classificado como classe 1.

2.5.1 Otsu Thresholding

O método de Otsu tem como objetivo a divisão das classes de tal forma que a variância intraclasses seja a menor possível. Formalmente temos: Dado que a variância é: $\sigma_w^2(T) = w_0(T)\sigma_0^2(T) + w_1(T)\sigma_1^2(t)$, queremos o T que mininmize o σ_w^2.

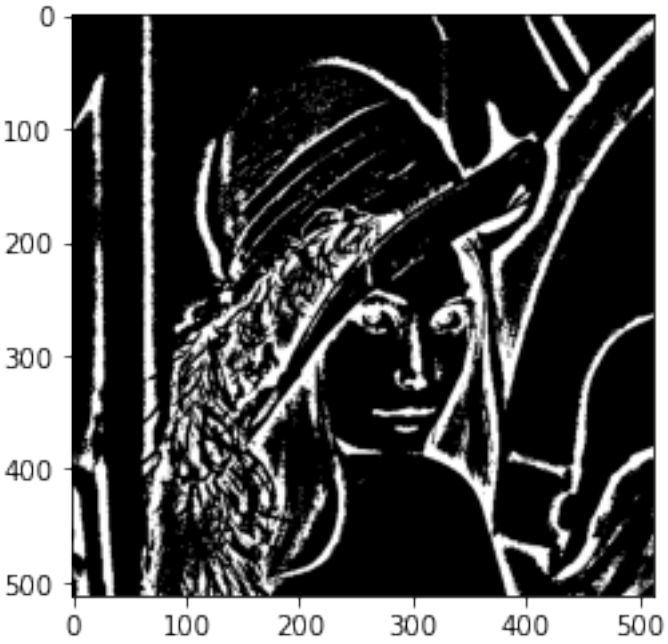
[3]:
```
import cv2
import matplotlib.pyplot as plt

# implementando 5 metodos de binarização do opencv
threshold = 0
max_value = 255

image = cv2.imread("lena.jpg", 0)

# when applying OTSU threshold, set threshold to 0.

_, output1 = cv2.threshold(image, threshold, max_value, cv2.THRESH_BINARY + cv2.
 ↪THRESH_OTSU)
_, output2 = cv2.threshold(image, threshold, max_value, cv2.THRESH_BINARY_INV +␣
 ↪cv2.THRESH_OTSU)
_, output3 = cv2.threshold(image, threshold, max_value, cv2.THRESH_TOZERO + cv2.
 ↪THRESH_OTSU)
_, output4 = cv2.threshold(image, threshold, max_value, cv2.THRESH_TOZERO_INV +␣
 ↪cv2.THRESH_OTSU)
_, output5 = cv2.threshold(image, threshold, max_value, cv2.THRESH_TRUNC + cv2.
 ↪THRESH_OTSU)

images = [image, output1, output2, output3, output4, output5]
titles = ["Orignals", "Binary", "Binary Inverse", "TOZERO", "TOZERO INV",␣
 ↪"TRUNC"]

f, axarr = plt.subplots(2,3, figsize=(18,18))
for i in range(2):
    for j in range(3):
        axarr[i, j].imshow(images[(i*3)+j], cmap='gray')
        axarr[i, j].set_title(titles[i*3+j])

plt.show()
```

2.5.2 Adaptative Thresholding

O método Adaptive Thresholding calcula o threshold para pequenas regiões da imagem. Abaixo mostramos como aplicar o Adaptive Thresholding a uma imagem em escala de cinza e o Threshold Local usando o filtro da biblioteca Scikit-image. Primeiramente, usando a imagem `lena.jgp`, salva em img, vamos converte-la para a escala de cinza usando `cvtColor`. Após isso, temos que definir qual será o tamanho da vizinhança do pixel ao qual calcularemos o threshold e salvamos o valor 25 em `neighbourhood_size`. Também temos que definir uma constante que será subtraida da média ou soma ponderada calculada pelo Adaptive Thresholding, então escolhemos o valr 15 e salvamos em `constant_c`. Por fim, utilizamos o método `adaptativeThreshold` da OpenCV, com os parâmetros: escala de cores = gray, valor máximo do pixel = 255, modo de calcular o valor de threshold = média da matriz formada pelo pixel e sua vizinhança - constante, então passamos `cv2.ADAPTIVE_THRESH_MEAN_C`, o tipo de thresholding a ser aplicado = binário inverso, o tamanho da vizinhança e a constante definidos anteriomente. Após salvar a imagem resultante, a convertemos e exibimos.

[11]:
```
from skimage.filters import threshold_local
import cv2
import argparse
import numpy as np

# load the image and convert to grayscale and blur it slightly
img = cv2.imread("lena.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


# apply adaptive thresholding with OpenCV
neighbourhood_size = 25
constant_c = 15
thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.
 ↪THRESH_BINARY_INV,
                                neighbourhood_size, constant_c)
plt.imshow(cv2.cvtColor(thresh, cv2.COLOR_BGR2RGB), cmap='gray')
```

[11]:
```
<matplotlib.image.AxesImage at 0x7f9f1f969490>
```

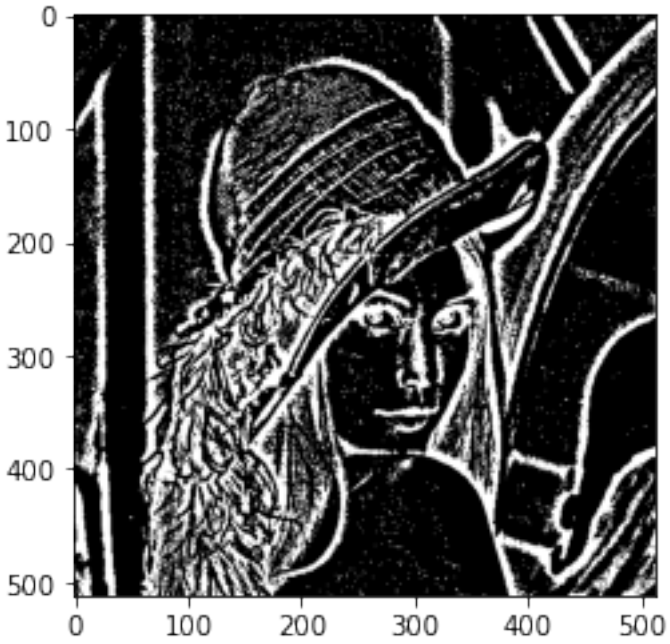


Aqui aplicamos o filtro treshold_local, para uma vizinhança de tamanho 29 e uma constante de valor 5. Com esse filtro, a imagem resultante mostrou ter mais detalhes que a anterior.

[12]:
```
# apply adaptive thresholding with scikit-image
neighbourhood_size = 29
constant_c = 5
```

```
threshold_value = threshold_local(gray, neighbourhood_size, offset=constant_c)
# np.uint8 devolve a matriz para a faixa de 8 bits
thresh = (gray < threshold_value).astype(np.uint8) * 255
plt.imshow(cv2.cvtColor(thresh, cv2.COLOR_BGR2RGB), cmap='gray')
```

[12]: <matplotlib.image.AxesImage at 0x7f9f1f8c9430>



3 Conclusão

Levando em conta o que foi observado, temos que os métodos estudados viabilizam a análise e obtenção de dados de imagens aplicando a criação do histograma da imagem, a equalização do histograma, as modificações no brilho e contraste e os métodos de thresholding. Com isso, o grupo aprendeu a utilizar técnicas disponibilizadas por bibliotecas Python, que auxiliam no processamento das imagens e, desde que combinadas, podem ser úteis em áreas como detecção de objetos e melhoria da qualidade de uma imagem.