DIP HW2 P10942A05 鄭閎軒

1、

set P as total pixel number

and set P_{rj} is its pixel number multiply its intensity

$$f_k = T(r_k) = \sum_{j=0}^{k} \frac{P_{rj}}{P} = \frac{1}{P}\sum_{j=0}^{k} P_{rj} \quad \text{--} \quad \text{①}$$

pixel with r_k intensity will be mapped to f_k

follow that $P_{fk} = P_{rk}$

In second pass of histogram equalization we get S_k through

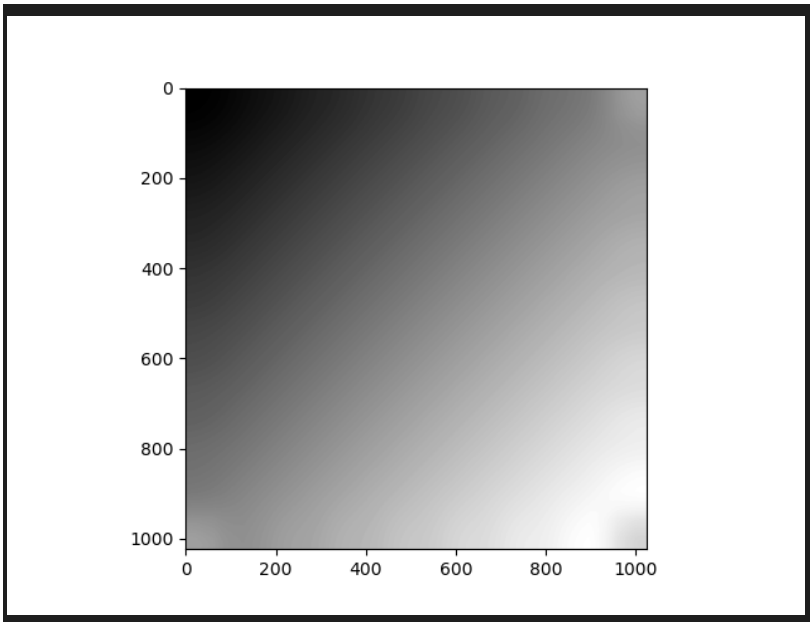
$$S_k = T(f_k) = \frac{1}{P}\sum_{j=0}^{k} P_{fj}$$

$P_{fj} = P_{rj}$ and follow ① $S_k = T(f_k) = \frac{1}{P}\sum_{j=0}^{k} P_{rj} = f_k$

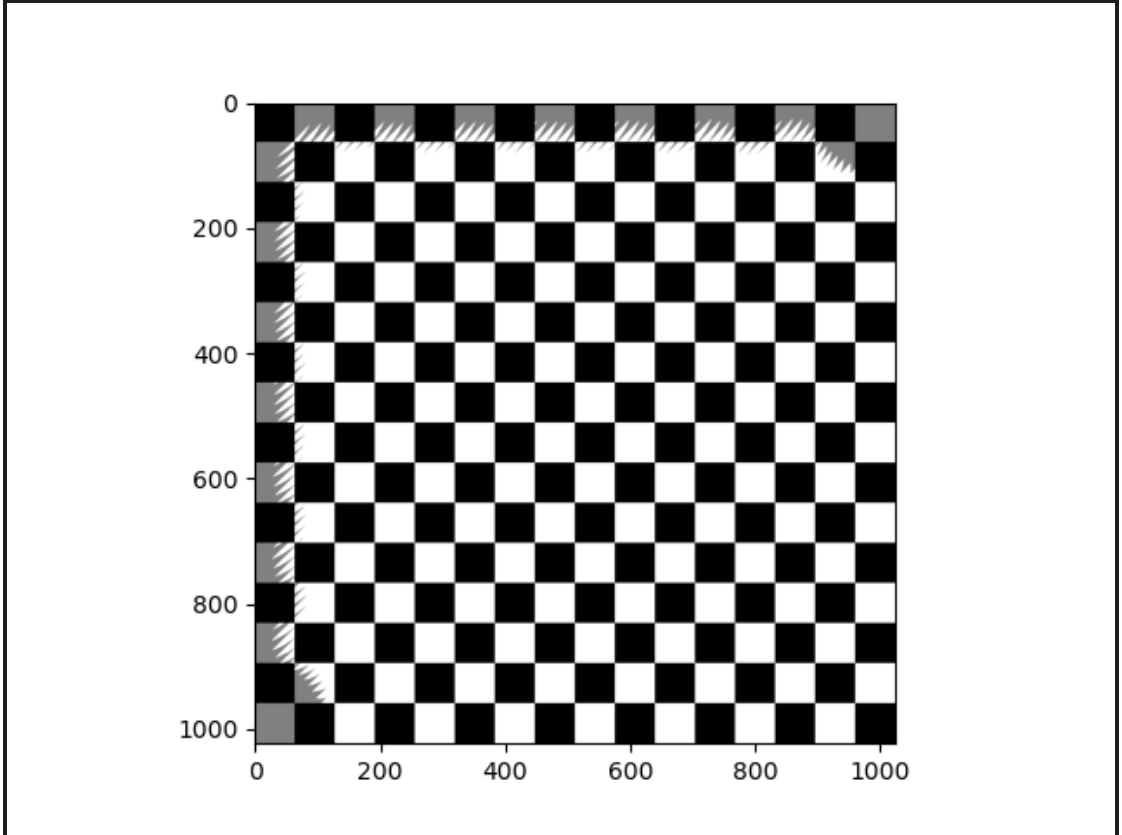
So we can get first and second pass of histogram equalization will get same result when negligible round-off error.

2、

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def imgfilter2D(img, filter, ratio):
    img_x, img_y = img.shape
    filter_x, filter_y = filter.shape
    result_mtx = np.zeros(((img_x - filter_x + 1), (img_y - filter_y +
1)))
    for i in range(result_mtx.shape[0]):
        for j in range(result_mtx.shape[1]):
            result_mtx[i][j] = np.abs(np.sum(img[i:i + filter_x, j:j +
filter_y] * filter) / ratio)
    return result_mtx.astype('uint8').reshape(result_mtx.shape[0],
result_mtx.shape[1], 1)
def gaussian_filter(filter_size, sigma):
    filter = np.zeros((filter_size, filter_size))
    img_x, img_y = filter.shape
    center = (img_x//2, img_y//2)
    for x_loc in range(img_x):
        for y_loc in range(img_y):
            dist_x_center = x_loc-center[0]
            dist_y_center = y_loc-center[1]
            r_sqr = ((dist_x_center)**2 + (dist_y_center)**2)
            filter[x_loc][y_loc] = np.exp(-r_sqr/(2*(sigma**2)))
    return filter
img=cv2.imread("479632_checkerboard1024-shaded.tif",
cv2.IMREAD_GRAYSCALE)
padding_img = np.pad(img, (128, 128), 'reflect')
g_filter = gaussian_filter(257, 64)
gaussian_blur_img = imgfilter2D(padding_img, g_filter,
np.sum(g_filter))
plt.imshow(gaussian_blur_img, "gray")
plt.show()
```

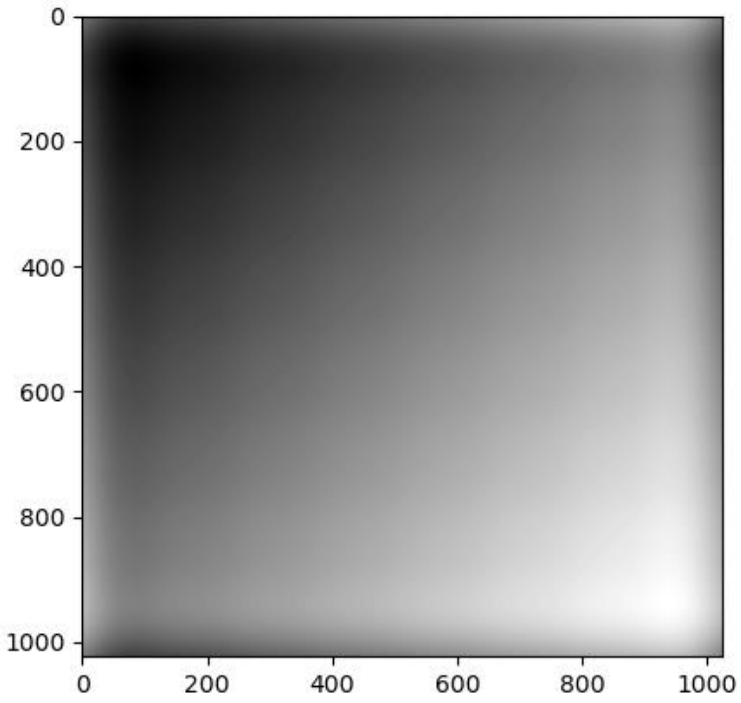



```
process_img = (img/gaussian_blur_img)
plt.imshow(process_img, "gray")
plt.show()
```

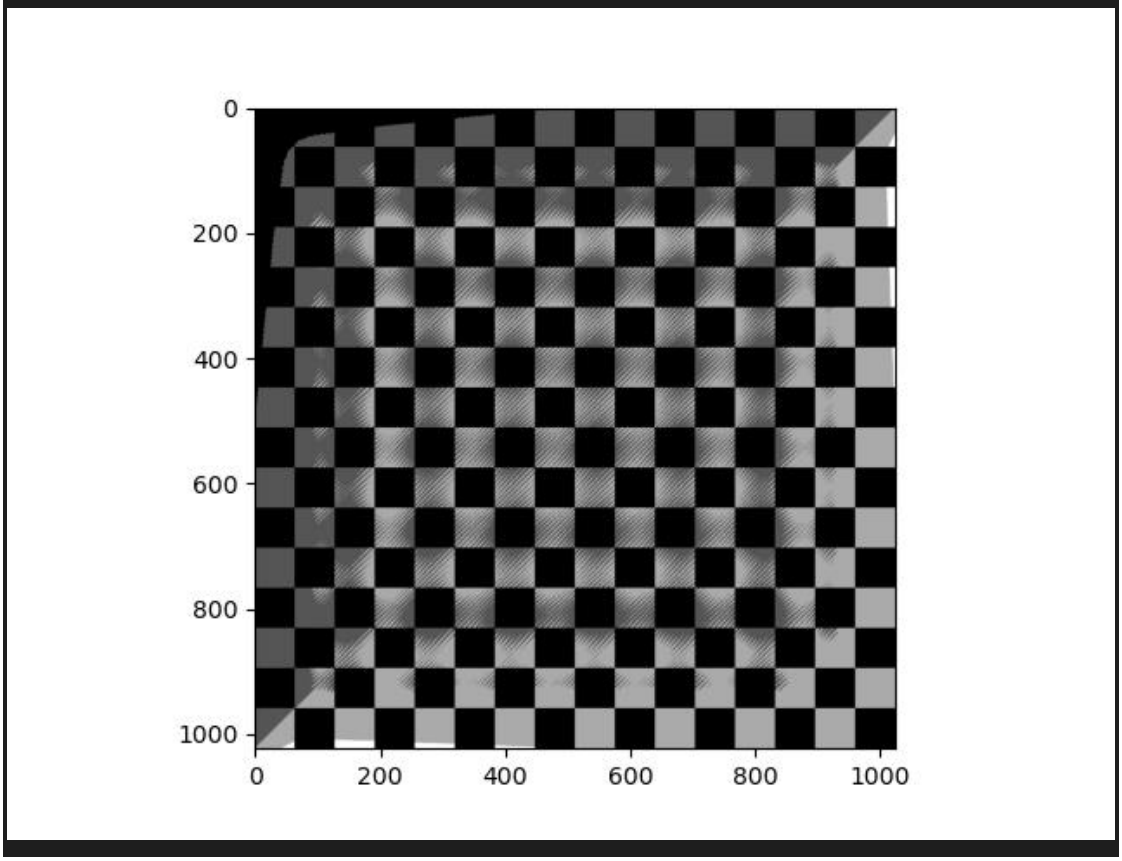



```
In another way we use
FFT base low pass filter will rect kernel radius 8
def low_pass_filter(dia, img_shape):
    filter = np.zeros(img_shape[:2])
```

```
    img_x, img_y = filter.shape
    center = (img_x//2, img_y//2)
    for x_loc in range(img_x):
        for y_loc in range(img_y):
            if dia > ((x_loc-center[0])**2 + (y_loc-center[1])**2)**0.5:
                filter[x_loc][y_loc] = 1
    return filter
def rever_fft_with_filter(fftshift_img, filter_size):
        passfilter_fftshift_img = fftshift_img *
low_pass_gaussian_filter(filter_size, sigma, (fftshift_img.shape))
    passfilter_ifftshift_img = np.fft.ifftshift(passfilter_fftshift_img)
    passfilter_img = np.fft.ifft2(passfilter_ifftshift_img)
    return passfilter_img
fft_img = np.fft.fft2(img)
fftshift_img = np.fft.fftshift(fft_img)
low_pass_8 = np.abs(rever_fft_with_filter(fftshift_img, 8))
low_pass_img = np.clip(np.abs(rever_fft_with_filter(fftshift_img, 8,
4)), 0, 255)
process_img = (img-low_pass_img).astype("uint8")
plt.imshow(low_pass_img, "gray")
plt.show()
```



```
plt.imshow(process_img, "gray")
plt.show()
```



2-2、

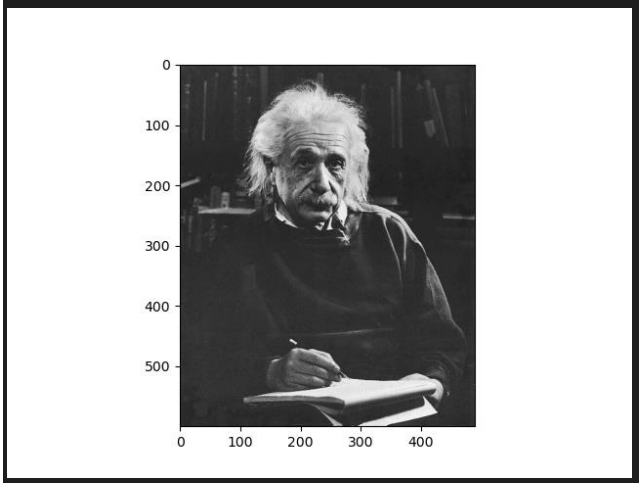
Due to our use reflecting method padding so the right bottom block padding will add another 3 black block around his neighbor.

3、

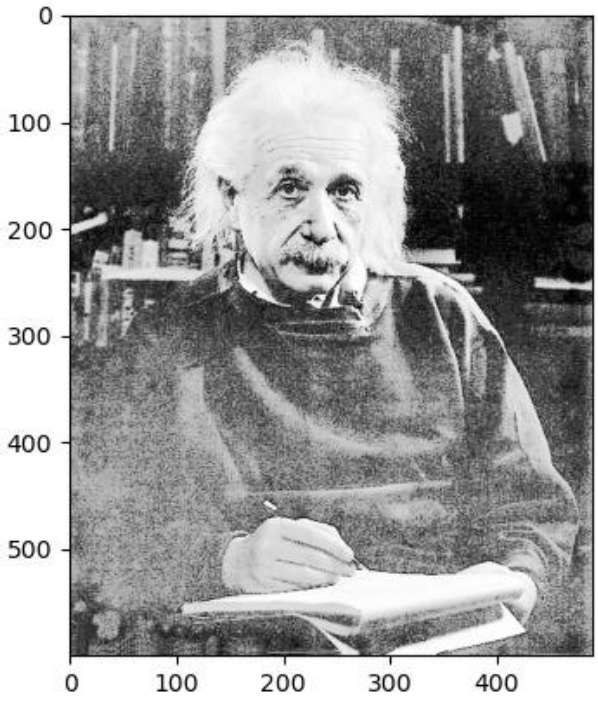
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def bgr2gray(img):
    return ( img[:, :, 0]*0.114 + img[:, :, 1]*0.587 + img[:, :,
2]*0.299).astype('uint8').reshape(img.shape[0],img.shape[1],1)
def hist_equal(img_array):
```

```
    flat = img_array.flatten()
    n = len(flat)
    img_bincount = np.bincount(flat)
    T = 190 * np.cumsum(img_bincount)/n
    T = T.astype('uint8')
    return T[img_array.astype('uint8')]
def custom_equal(img_array):
    flat = img_array.flatten()
    n = len(flat)
    img_bincount = np.bincount(flat)
    new_mapping_table = np.arange(len(img_bincount))
    T = ((new_mapping_table-73) * 0.5) + np.clip(((new_mapping_table-85)
* 6.5), 0, 2222) - np.clip(((new_mapping_table-103) * 5.5), 0, 2222)
    print(T)
    T = T.astype('uint8')
    return T[img_array.astype('uint8')]
img=cv2.imread("479632_einstein-low-contrast.tif",
cv2.IMREAD_GRAYSCALE)
after_hist = hist_equal(img)
# gray_img = bgr2gray(img)
array_img = np.asarray(img)
result = custom_equal(array_img)
flat = after_hist.flatten()
plt.hist(flat, bins=50)
plt.show()
plt.imshow(after_hist, cmap = 'gray')
plt.show()
```

we read as gray image will show a not bad result

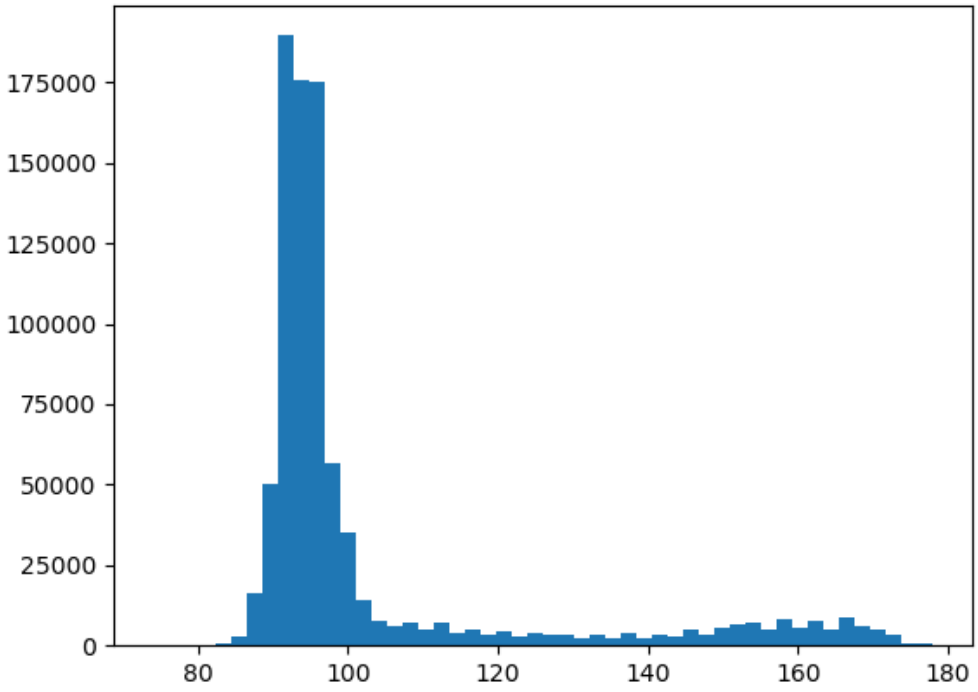


use normal histogram equalizer we will get

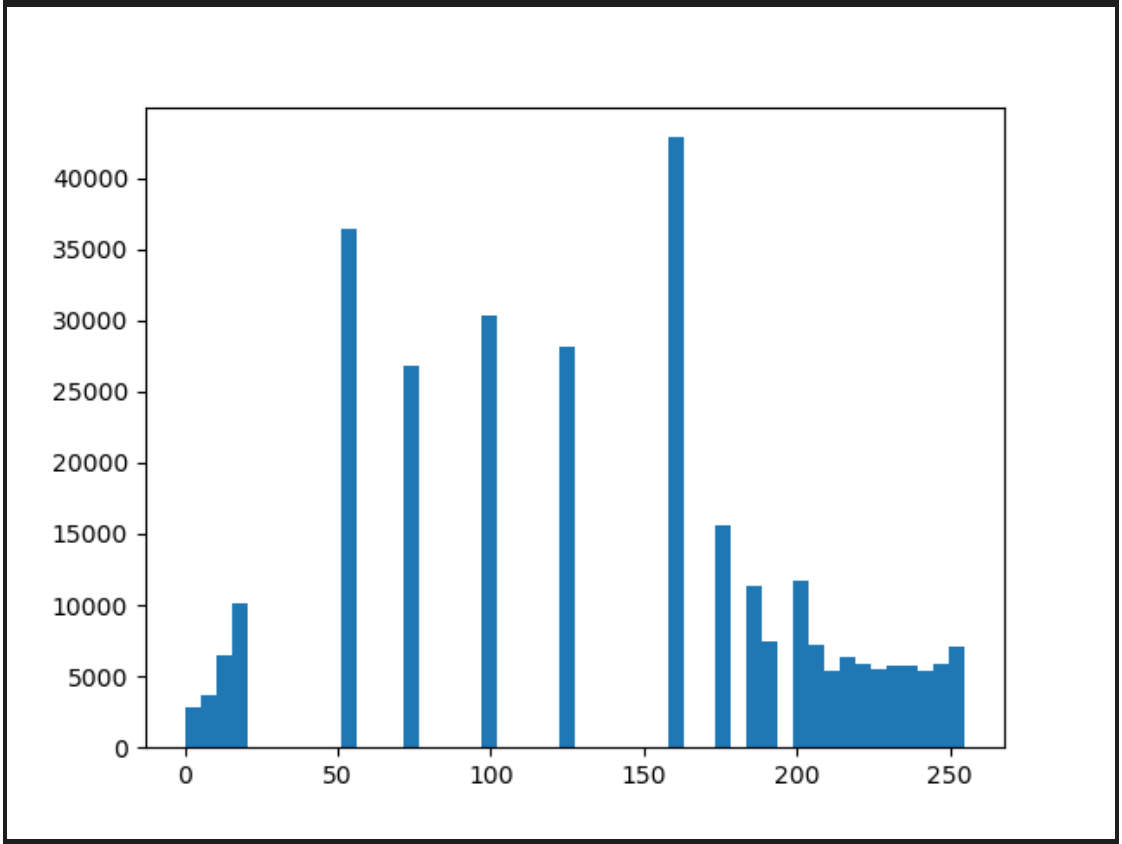


And we use our customization way to enhance the data distribution

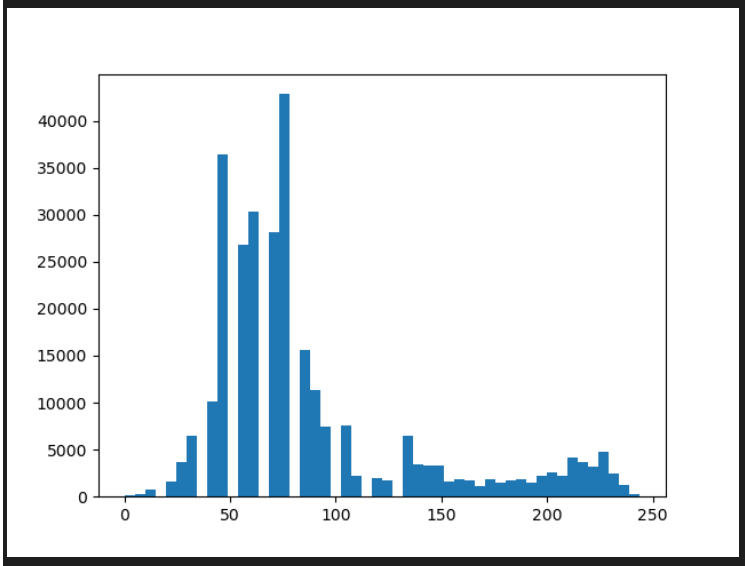
Raw data



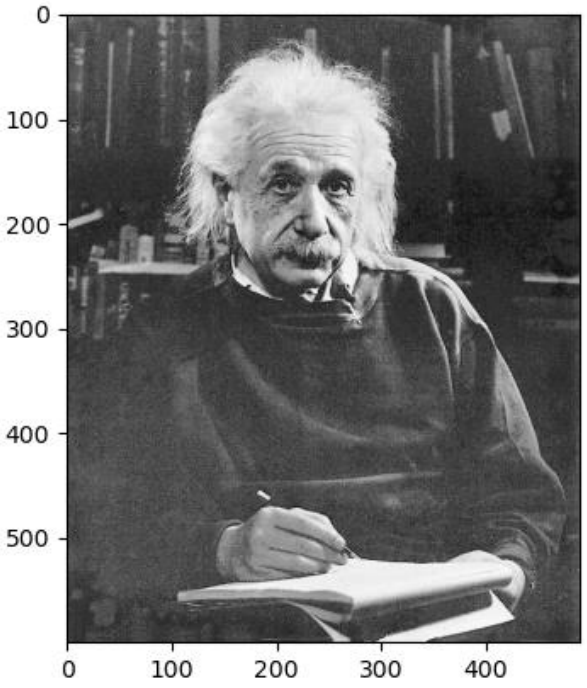
After histogram equalization



In above result we have to enhance 80-110 intensity resolution in raw picture we use our customization way



Get better result in distribution



Also in our processed image