HACETTEPE UNIVERSITY

ELECTRICAL&ELECTRONICS ENGINEERING

ELE 492: Image Processing

2021-2022 Spring Term

Homework 2

Name: Murat Karakuş
Number: 21989138

Date: 13/05/2022

I have not received or given any aid in this homework. All the work presented below is my own work.

Murat Karakuş

1-) Device a method to enhance the image by taking away the existing pattern in the background.

Given image

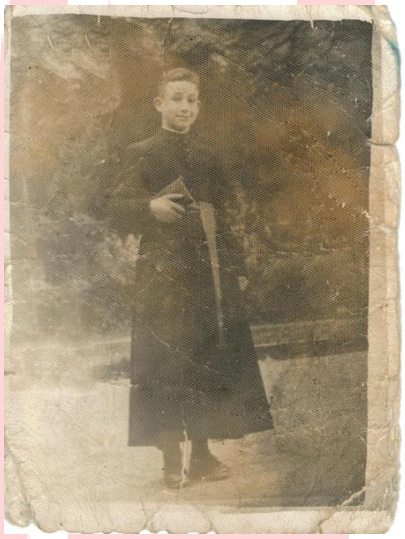

Figure 0

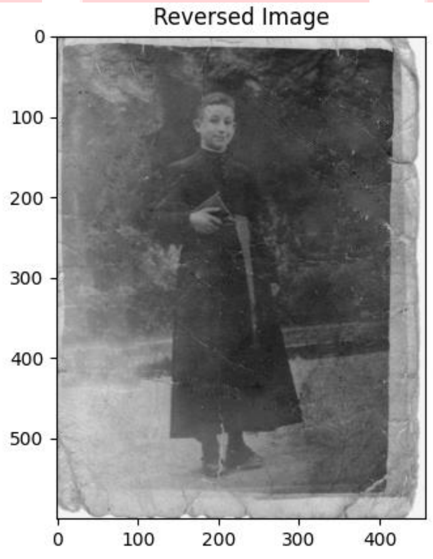

Figure 1

The image on the left is the given image that needed to be enhanced. First of all, I wanted to see it in the frequency domain. So used some special function as showed in my codes.(np.fft.fftshift(np.fft.fft2(img)))

Warning: To use this function image should be grayscaled!

The frequency domain result is shown in Figure 3. It is clearly seen that there are some periodic noises. I designed a specific filter to get rid of these noises. My filter is given in Figure 4. When Shifted FFT image is multiplied with the filter the result in Figure 5 is reached.

From my point of view, this is what sould be done, but there are some other ways to go through. Basically, applying some different types of low pass filters would work fine because the DC component of the signal is just in the middle and a low pass filter covers it. So it eliminates noises shown in Figure 3.

After elimination of noises in frequency domain, I tried to get back in the spatial domain. I used the following function:

```
img_ifft = np.fft.ifft2(np.fft.ifftshift(img_fftshift))
```

When I plot this img_ifft somehow I get the result on the left side(Figure 1). I played a lot with the pixels to make my enhanced image colored. The way is expressed below.

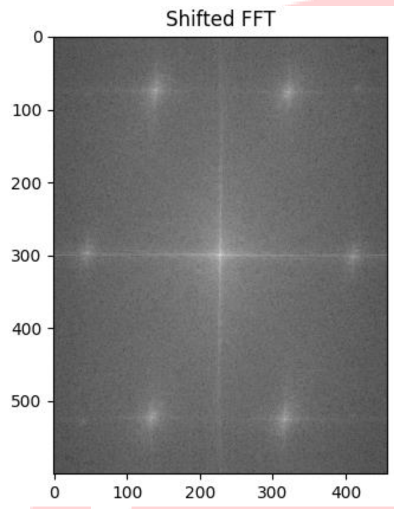

Figure 3

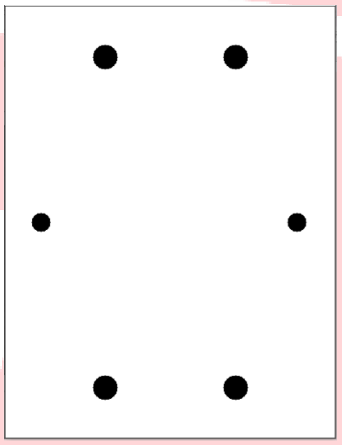
Figure 4

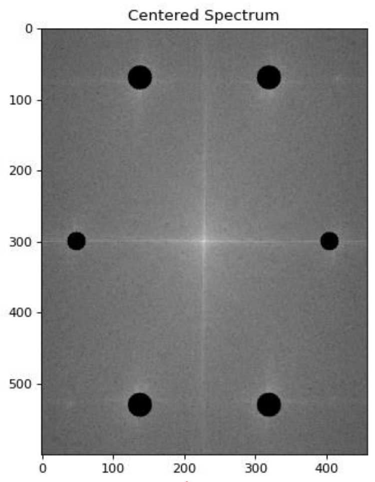

Figure 5

The Enhanced Image

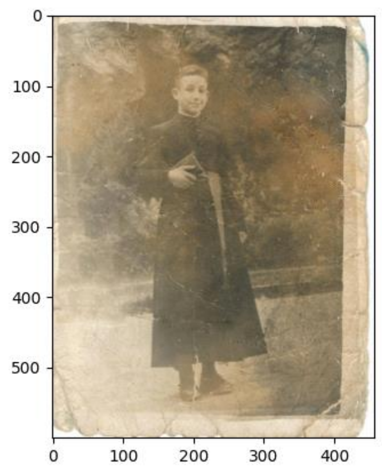
Figure 6

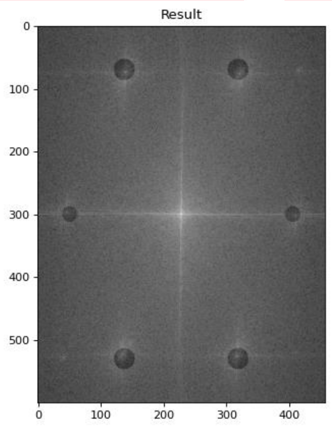

Figure 7

Coloring the image as it was, was the hardest part for me while doing this homework. This is mostly because I was not able to understand the logic behind of it. Simple cv.cvtColor(img, cv.COLOR_GRAY2RGB) doesn't work on this example. So I figured out the weightinesses of colors in RGB, then by multiplying by three and these weights, I get the result given in Figure 6. To see if my work is correct, I showed the fft version of this last image in Figure 7. As it is seen, it perfectly matches what I wanted.

In the end, a good-looking image without periodic noises is obtained. This was hard but not fun work. But, I am grateful that I worked on it and I think it is very instructional.

2-) Device a method to restore the image.

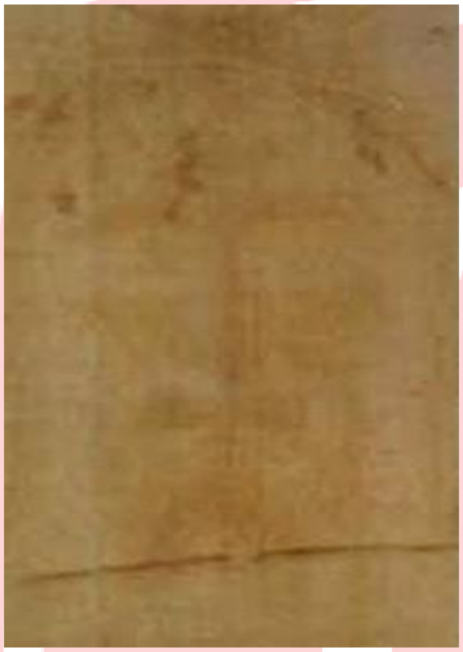

Figure 8

Figure 8 is the given image. This time there is no periodic noise or noise I can say. I personally wanted to change its color and brightness.

In my trying, I couldn't reach a good solution but I'll share what I did.

The simple histogram of the image is given in Figure 9. By using cv.equalizeHist() function, I equalized the histogram of the given image. The result of this equalization is given in Figure 10. Of course, it didn't make me reach what I wanted.
So, I applied simple thresholding. In figure 11, the result of this thresholding is given on the right half part and the image which shall be restored is on the left part. The difference is clearly seen. However, this is not the perfect solution I wanted to have. After that, I continued to tryings.

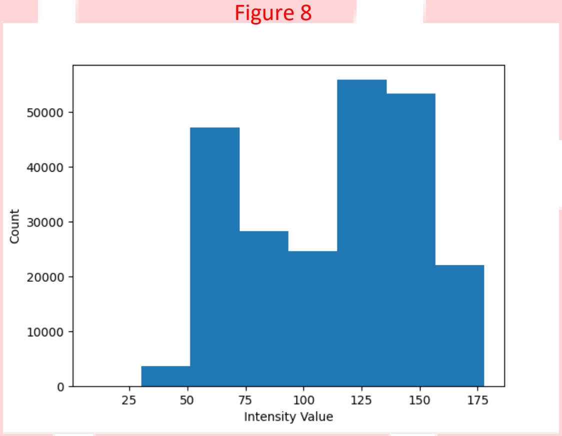

Figure 9

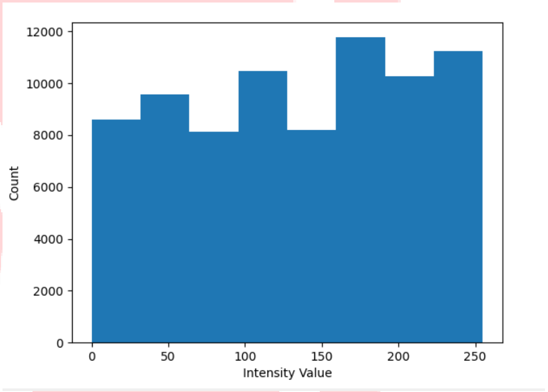

Figure 10

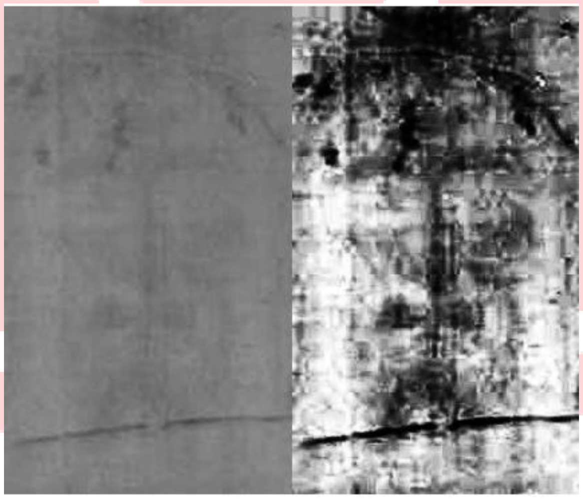

Figure 11

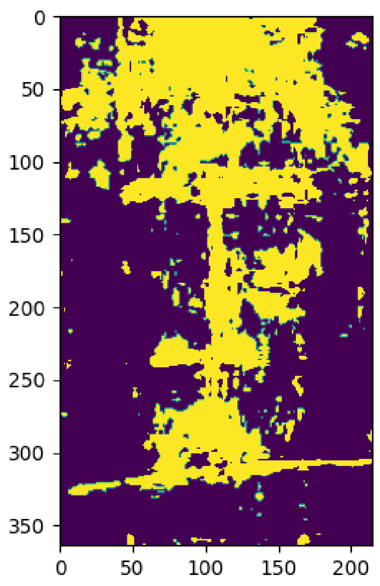

Figure 12

By checking its pixels values, I figured out that it goes like Blue, Green, and Red color. Then, I realized that this is related to the way of opening the image actually. After grayscaling the image by caring this information and using the function cv.cvtColor(img, cv.COLOR_BGR2GRAY), I made some thresholding to this grayscaled image. Simple Threshold and Adaptive Threshold are two of them. With so many plays, I reached the result given in Figure 12. There is a guy in the image and it is unambiguously seen. I wish I could do better but that is all I can do now.

The codes used for Q1 are given below:

```
import cv2 as cv
import numpy as np
from skimage import io, color
import matplotlib.pyplot as plt
from pylab import *
img = io.imread('oldPhoto.jpg')
print(img.shape)

b, g, r = cv.split(img)

total = np.zeros((600, 457))
for i in range(600):
    for j in range(457):
        total[i][j] = int(b[i][j]) + int(g[i][j]) + int(r[i][j])
print(total.shape)

figure(0)
plt.imshow((np.abs(img).astype(int)))
print(img.shape)

img = cv.cvtColor(img, cv.COLOR_RGB2GRAY)

img_fftshift = np.fft.fftshift(np.fft.fft2(img))
```

```
blank = np.ones(img.shape[:2], dtype='uint8')*255
cv.circle(blank, (138, 70), 17, 0, -1)
cv.circle(blank, (319, 70), 17, 0, -1)
cv.circle(blank, (138, 530), 17, 0, -1)
cv.circle(blank, (319, 530), 17, 0, -1)
cv.circle(blank, (49, 300), 13, 0, -1)
cv.circle(blank, (404, 300), 13, 0, -1)
cv.imshow('Blank Image', blank)

plt.figure(num=None, figsize=(8, 6), dpi=80)
figure(1)
plt.imshow(np.log(1+np.abs(img_fftshift)), "gray"), plt.title("Shifted FFT")

img_fftshift = img_fftshift * blank
img_fftshift = img_fftshift/255

plt.figure(num=None, figsize=(8, 6), dpi=80)
figure(2)
plt.imshow(np.log(1+np.abs(img_fftshift)), "gray"), plt.title("Centered
Spectrum")
img_ifft = np.fft.ifft2(np.fft.ifftshift(img_fftshift))
figure(3)
plt.imshow(np.abs(img_ifft), "gray"), plt.title("Reversed Image")

avgNew = (np.abs(img_ifft).astype(int))
avgNew = avgNew * 3
bNew = np.zeros((600, 457))
gNew = np.zeros((600, 457))
rNew = np.zeros((600, 457))

for i in range(600):
    for j in range(457):
        bNew[i][j] = ((int(b[i][j]) / total[i][j]) * avgNew[i][j]).astype(int)
        gNew[i][j] = ((int(g[i][j]) / total[i][j]) * avgNew[i][j]).astype(int)
        rNew[i][j] = ((int(r[i][j]) / total[i][j]) * avgNew[i][j]).astype(int)

merged = cv.merge([bNew, gNew, rNew])
figure(4)
plt.imshow((np.abs(merged).astype(int)))

img_float32 = np.float32(merged)
merged = cv.cvtColor(img_float32, cv.COLOR_RGB2GRAY)

merged_fftshift = np.fft.fftshift(np.fft.fft2(merged))
plt.figure(num=None, figsize=(8, 6), dpi=80)
figure(5)
plt.imshow(np.log(1+np.abs(merged_fftshift)), "gray"), plt.title("Result")
```

The codes used for Q2 are given below:

```
from skimage import io
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np
from pylab import *
from skimage import io, color
image = io.imread('findIt.jpg')
"""
_ = plt.hist(image.ravel(), bins = 8 )
_ = plt.xlabel('Intensity Value')
_ = plt.ylabel('Count')
"""

img = cv.imread('findIt.jpg', 0)
equ = cv.equalizeHist(img)
figure(0)
_ = plt.hist(equ.ravel(), bins = 8 )
_ = plt.xlabel('Intensity Value')
_ = plt.ylabel('Count')
threshold, thresh = cv.threshold(equ, 100, 150, cv.THRESH_BINARY)
cv.imshow('Simple Thresholded', thresh)
figure(1)
io.imshow(equ)
res = np.hstack((img, equ)) #stacking images side-by-side
cv.imwrite('res.png', res)
plt.show()
io.show()
```

Other trying :

```
from skimage import io, color, data, exposure, transform
from pylab import *
import numpy as np
import cv2 as cv
img = io.imread('findIt.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
threshold, thresh = cv.threshold(gray, 150, 255, cv.THRESH_BINARY)
cv.imshow('Simple Thresholded', thresh)
figure(3)
plt.imshow((np.abs(thresh).astype(int)))

adaptive_thresh = cv.adaptiveThreshold(gray, 255, cv.ADAPTIVE_THRESH_MEAN_C,
cv.THRESH_BINARY, 11, 3)
cv.imshow('Adaptive Thresholded', adaptive_thresh)
cv.waitKey(0)
plt.show()
```

I would like to share a Google Colab version of my work:

https://colab.research.google.com/drive/1pDkFalgNzhpNpubIIEdpqwfSc-AUyS0Y

Reference List:

https://pythonwife.com/color-spaces-and-channels-in-opencv/

https://web.stanford.edu/class/cs101/image-6-grayscale-adva.html

https://programming.vip/docs/opencv-python-image-high-pass-filter-and-low-pass-filter.html

http://www.guillermoluijk.com/tutorial/fft/index.htm

https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html

https://towardsdatascience.com/histograms-in-image-processing-with-skimage-python-be5938962935