EEL 5820: Digital Image Processing
Fall 2022

Homework # 8

Laplacian of a Gaussian

Submitted by:

Your name: Maryna Veksler

PID#: 5848258

Department of Electrical and Computer Engineering

Date: _____12/12/2022_____

Objective

To Implement the Laplacian of Gaussian filter and determine the different results for the different values of σ.

Method

To implement the Laplacian of Gaussian the next steps can be taken:

1. Determine the kernel dimensions based on the sigma as
$$\boxed{2 \times int(4 \times \sigma + 0.5) + 1}$$
2. Creating a gaussian kernel
3. Applying Laplacian to the kernel using following formula:
$$-\left(-\frac{1}{\pi\sigma^4}\right)\left(1-\frac{u^2+v^2}{2\sigma^2}\right) \times e^{-\frac{u^2+v^2}{2\sigma^2}}$$
4. Convolve original image with the Laplacian of Gaussian kernel

Results

Example 1:

Figure 3.1 Original Image

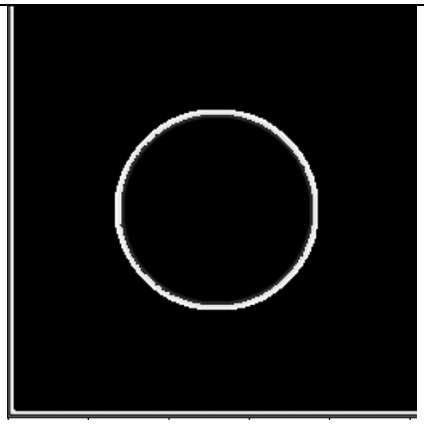


Figure 3.2 Laplacian of Gaussian function filtered Image for $\sigma = 1$

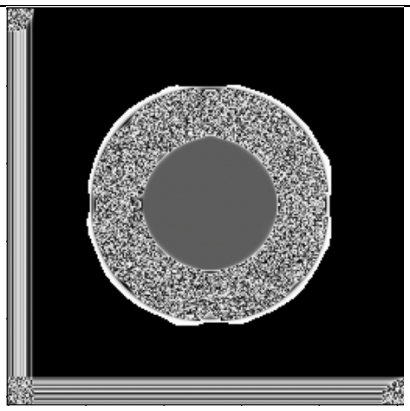


Figure 3.3 Laplacian of Gaussian function filtered Image for $\sigma = 4$

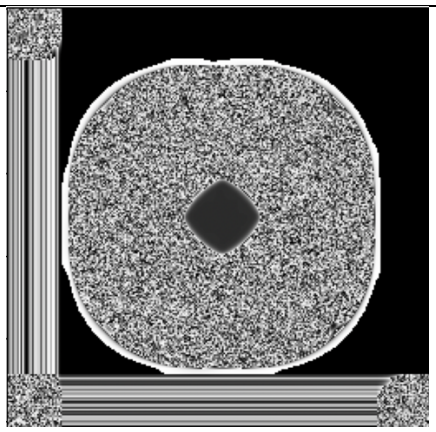


Figure 3.4 Laplacian of Gaussian function filtered Image for $\sigma = 8$

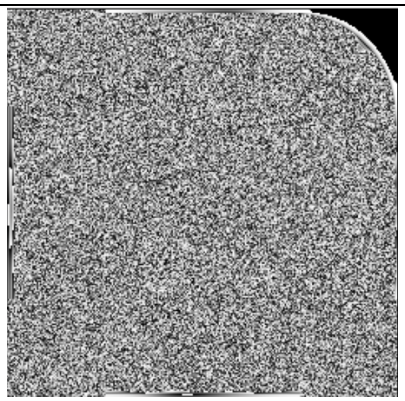


Figure 3.5 Laplacian of Gaussian function filtered Image for $\sigma = 16$

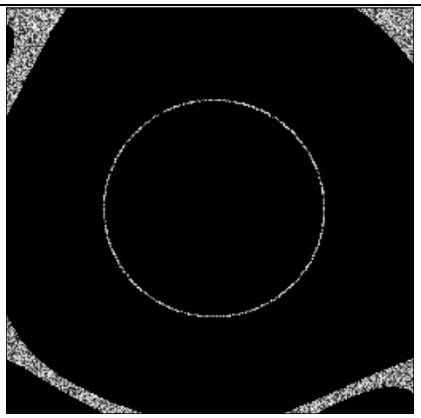


Figure 3.6 Laplacian of Gaussian function filtered Image for $\sigma = 32$

Example 2:

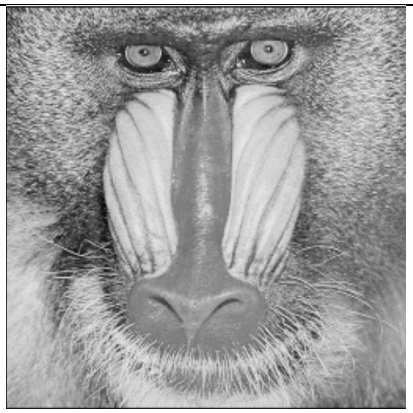


Figure 3.1 Original Image

Figure 3.2 Laplacian of Gaussian function filtered Image for $\sigma = 1$

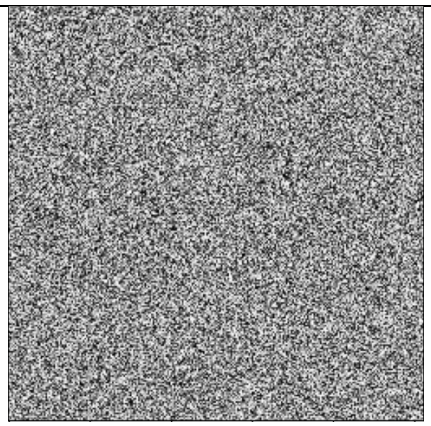


Figure 3.3 Laplacian of Gaussian function filtered Image for $\sigma = 4$

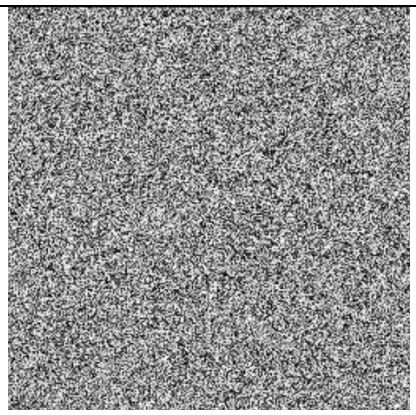


Figure 3.4 Laplacian of Gaussian function filtered Image for $\sigma = 8$

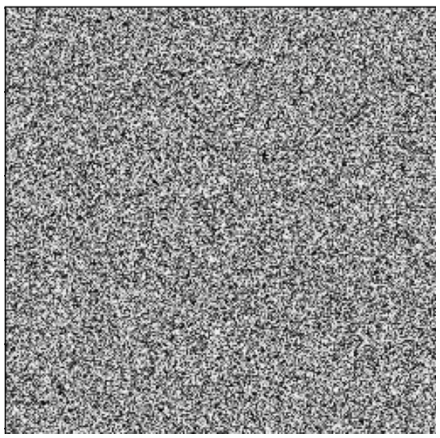


Figure 3.5 Laplacian of Gaussian function filtered Image for $\sigma = 16$

Figure 3.6 Laplacian of Gaussian function filtered Image for $\sigma = 32$

Example 3:

Figure 3.1 Original Image

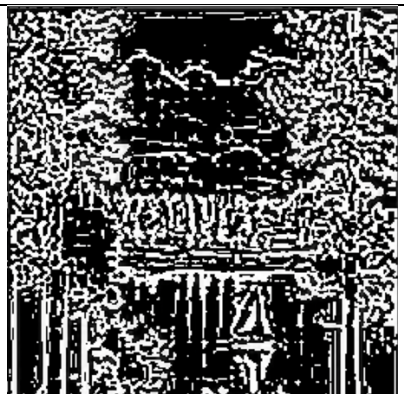


Figure 3.2 Laplacian of Gaussian function filtered Image for $\sigma = 1$

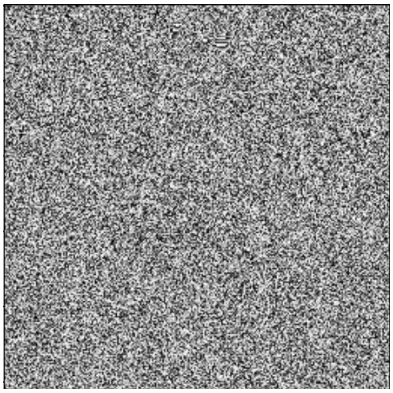


Figure 3.3 Laplacian of Gaussian function filtered Image for $\sigma = 4$

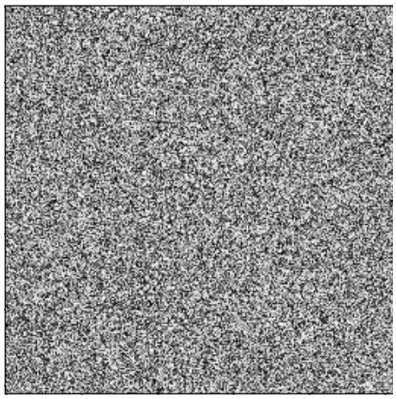


Figure 3.4 Laplacian of Gaussian function filtered Image for $\sigma = 8$

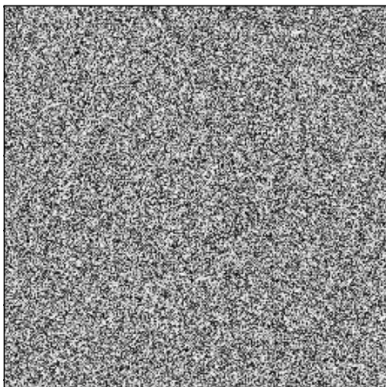


Figure 3.5 Laplacian of Gaussian function filtered Image for $\sigma = 16$

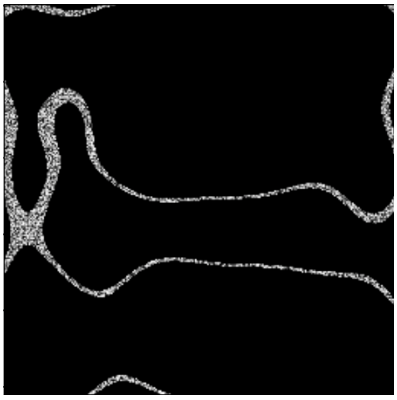


Figure 3.6 Laplacian of Gaussian function filtered Image for $\sigma = 32$

Discussion

The Laplacian of Gaussian function filter is implemented for the different sigma values, σ. Increasing σ results in the sharper edges, however, the edges start appearing as noisy representations for the complex images (natural scene). Interestingly, after we keep increasing σ value, the results are unexpected, and it is hard to recognize the image at all.

Program

```
import numpy as np
from matplotlib import pyplot as plt
import cv2, math

def get_lap_of_gaus(u,v, sigma):
    s2 = sigma**2
    sub = -((u**2) + (v**2))/(2*s2)

    return -(-1 / math.pi * (s2**2)) * (1 +sub) * np.exp(sub)

def build_kernel(dimension, sigma, kernel):
    large = dimension // 2

    linSp = np.linspace(-large, large + 1, dimension)

    X, Y = np.meshgrid(linSp, linSp)

    return kernel(u=X, v=Y, sigma=sigma)

def convolution(image, kernel):
    dimKer = kernel.shape[0]
    large = dimKer // 2

    image = np.pad(image, pad_width=((large, large), (large, large)),
mode='constant', constant_values=0) \
            .astype(np.float32)

    result = np.zeros(image.shape)

    for h in range(large, image.shape[0] - large):
        for w in range(large, image.shape[1] - large):
            square = image[h - large:h - large + dimKer, w - large:w - large +
dimKer]
            result[h, w] = np.sum(np.multiply(square, kernel))

    # Return the convolved image
    return result[large:-large, large:-large]

def transform(image, sigma):
    kernel_dimensions = 2 * int(4 * sigma + 0.5) + 1
    kernel = build_kernel(dimension=kernel_dimensions, sigma=sigma,
kernel=get_lap_of_gaus)

    result = convolution(image, kernel)

    return result.astype(np.uint8)

if __name__ == "__main__":
```

```
image_path = "/Users/marynavek/Projects/ImageProcessing/natural_scene.png"

image = cv2.imread(image_path, 0)
image = cv2.resize(image, (256,256))
M, N = image.shape
plt.imshow(image, cmap='gray')
plt.show()
log = transform(image, 1)

plt.imshow(log, cmap='gray')
plt.show()

log = transform(image, 4)

plt.imshow(log, cmap='gray')
plt.show()

log = transform(image, 8)

plt.imshow(log, cmap='gray')
plt.show()

log = transform(image, 16)

plt.imshow(log, cmap='gray')
plt.show()

log = transform(image, 32)

plt.imshow(log, cmap='gray')
plt.show()
```

EEL 5820: Digital Image Processing
Fall 2022

Homework # 9

Histogram Equalization

Submitted by:

Your name: Maryna Veksler

PID#: 5848285

Department of Electrical and Computer Engineering

Date: _____12/12/2022_____

Objective

To Implement Histogram equalization for the image stabilization.

Method

We implement the Histogram equalization in two ways. First, Histogram stretching is implemented to stretch the intensity values in order to cover the desired range. This is beneficial in cases when image gray levels are narrowed down to a specific histogram range. The Histogram stretching can be implemented as

$$s = (r - c)\left(\frac{b - a}{d - c}\right) + a$$

where ***r*** is the actual pixel, ***s*** is processed pixel, ***a and b*** are the lowest and highest intensity values, and ***c and d*** can be selected as desired.

On the other hand, histogram equalization is more complex and aims to uniformly distribute the intensity values to obtain flat-like image. The equalization is implemented using following steps.

1. Calculate probability density function

$$p(x_k) = \frac{n_k}{MXN}$$

where ***M*x*N*** is a total number of pixels, with ***M and N*** standing for number of rows and columns. n_k corresponds to the number of pixels belong the intensity level ***"k"***.

2. Calculate Cumulative Distribution function

$$c(x_i) = \sum_{i=0}^{k} p(x_i)$$

3. Redistribution of the intensities.

$$f(x) = X_{L-1} \times c(x_i)$$

where ***X(L-1)*** corresponds to maximum intensity values

Results

Example 1: Histogram (Contrast) Stretching

Figure 1.1 Original Image

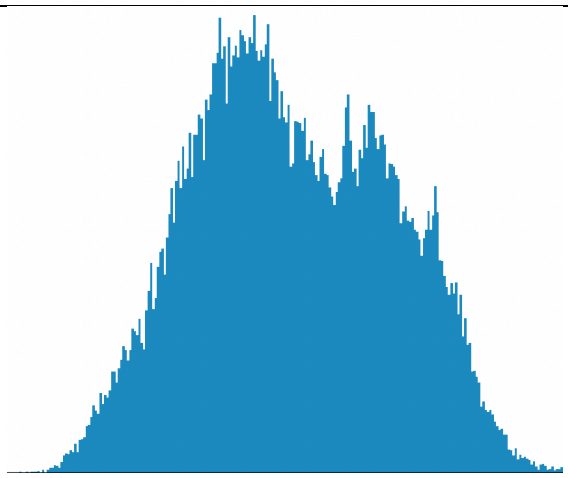


Figure 1.2 Original Histogram

Figure 1.3 Histogram Image (500)

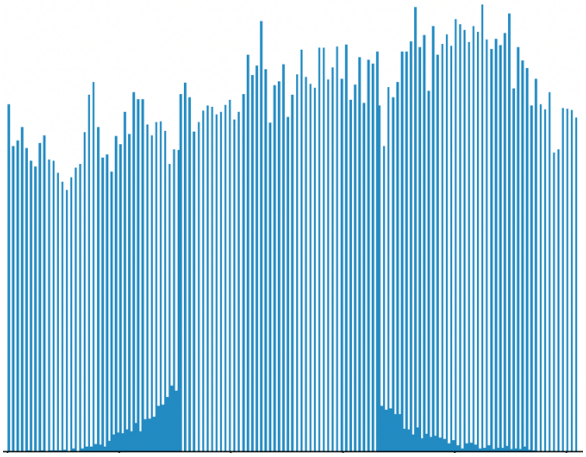


Figure 1.4 Normalized Histogram (500)

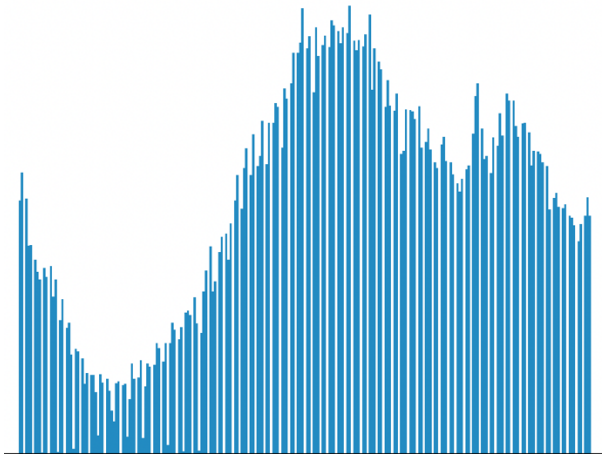


Figure 1.6 Normalized Histogram (350)

Figure 1.5 Histogram Image (350)

Figure 1.7 Original Histogram (150)

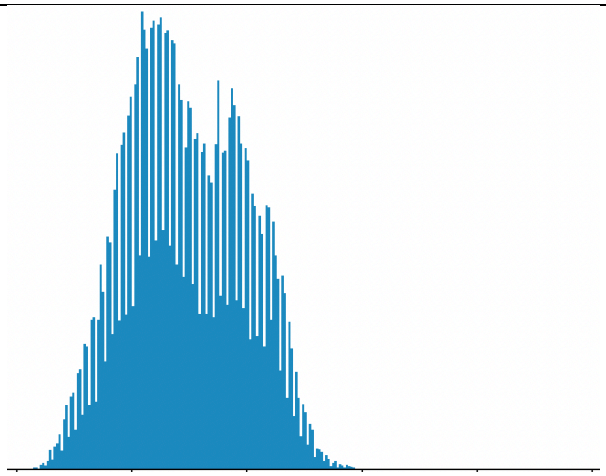


Figure 1.8 Normalized Histogram (150)

Figure 1.9 Original Image

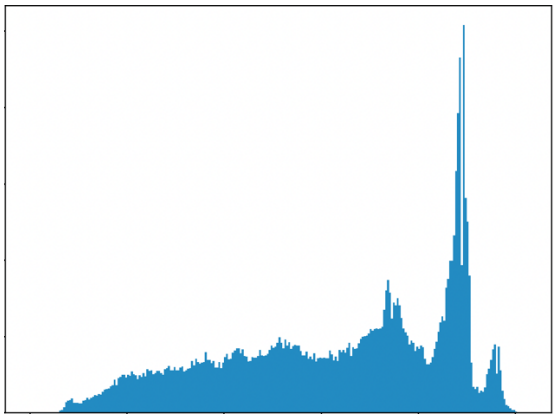


Figure 1.10 Original Histogram

Figure 1.11 Histogram Image (500)

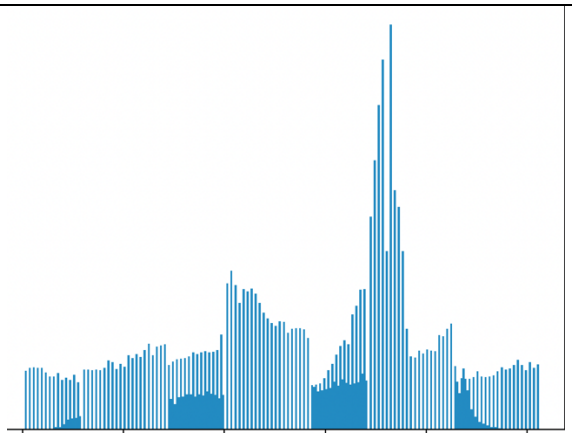
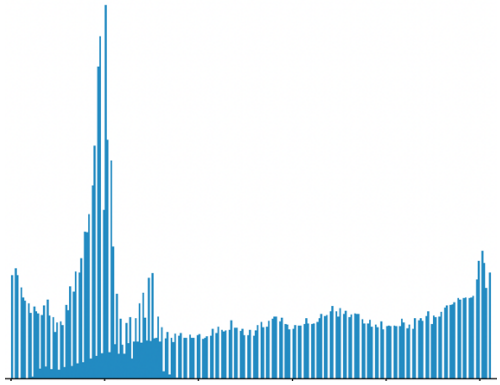
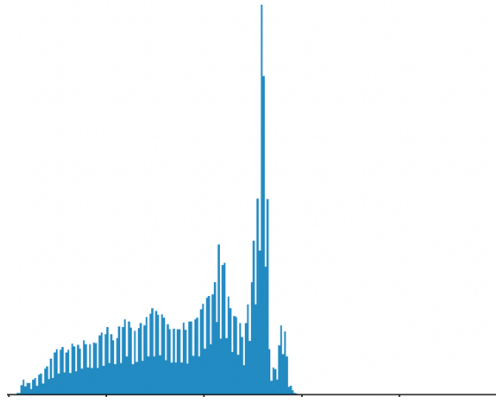


Figure 1.12 Normalized Histogram

	(500)
 **Figure 1.13 Histogram Image (3500)**	 **Figure 1.14 Normalized Histogram (350)**
 **Figure 1.15 Histogram Image (150)**	 **Figure 1.16 Normalized Histogram (150)**

Example 2: Histogram Equalization

 **Figure 1.1 Original Image**	 **Figure 1.2 Histogram Equalization**

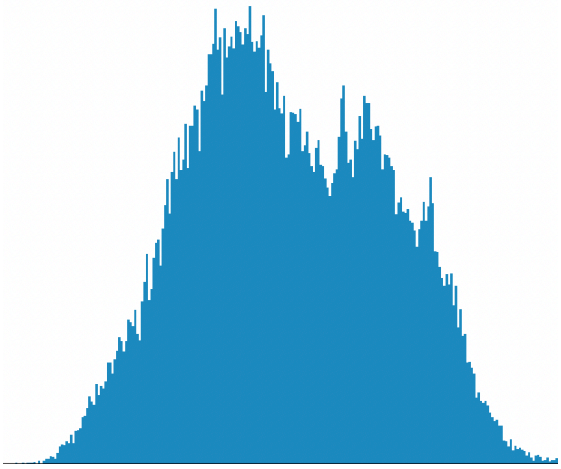


Figure 1.3 Original Histogram

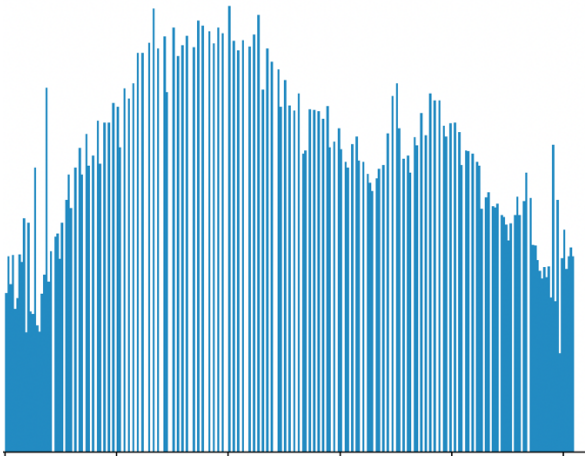


Figure 1.4 Normalized Histogram

Figure 1.5 Original Image

Figure 1.6 Histogram Equalization

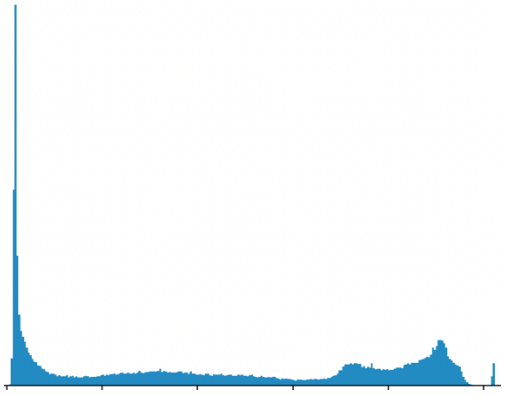


Figure 1.7 Original Histogram

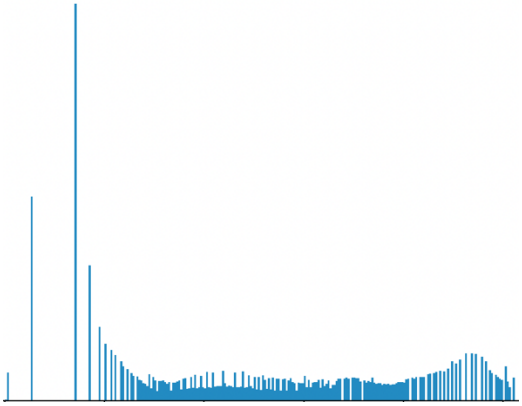


Figure 1.8 Normalized Histogram

Figure 1.9 Original Image

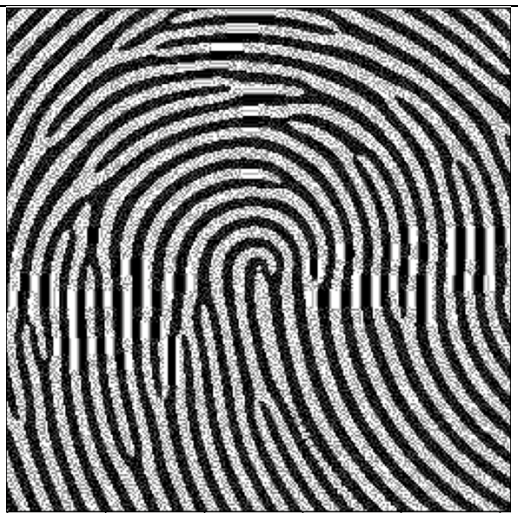


Figure 1.10 Histogram Equalization

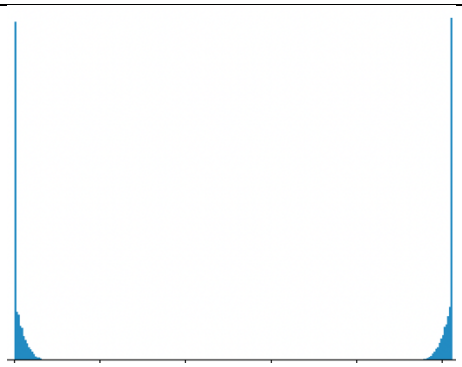


Figure 1.11 Original Histogram

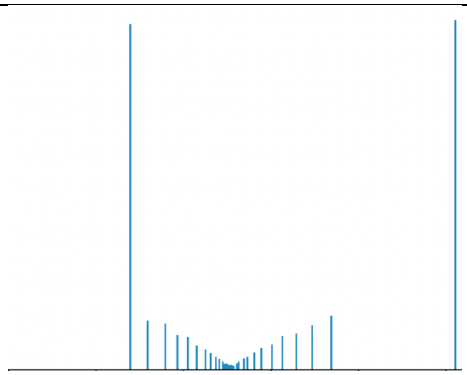


Figure 1.12 Normalized Histogram

Figure 1.13 Original Image

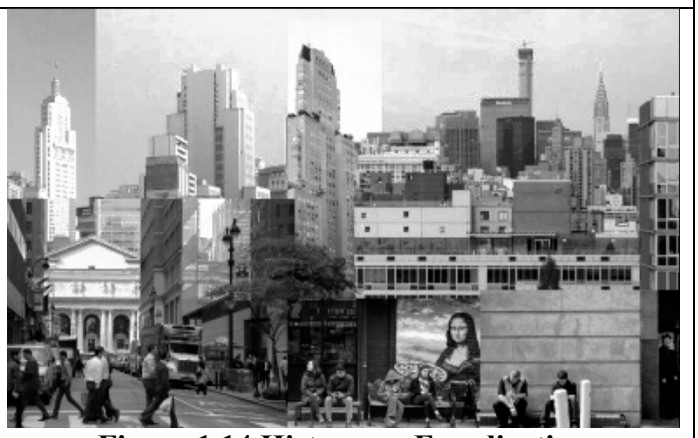


Figure 1.14 Histogram Equalization

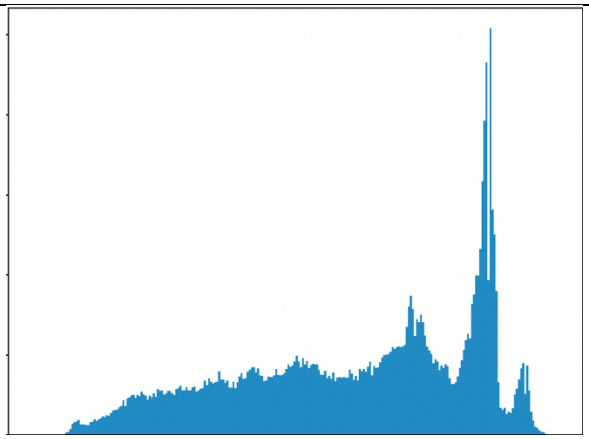


Figure 1.15 Original Histogram

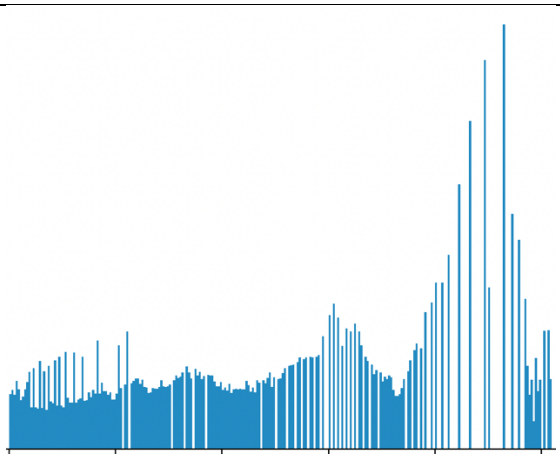


Figure 1.16 Normalized Histogram

Discussion

In this work, we implemented two types of image enhancement using histogram equalization. Specifically, histogram equalization and dynamic range stretching techniques were compared.

From the results, we observe that stretching is able to distribute the histogram values more evenly. However, since the expansion range is a constant that needs to be set manually, it is hard to determine the proper value to ensure actual image improvement, rather than degradation. Thus, for the values 500 and 350 as the range maximum, we can observe the image transformation such that all of the details are being significantly emphasized, but not necessarily to improve the image quality. At the same time, reducing value of stretching to below 256, as an example 150, leads to the image histogram being skewed to the left

We determine that a standard histogram equalization is more beneficial, as the original values of gray levels directly affect the transformation, while the improvement depends on the requirement. Therefore, histogram equalization is more adaptive than histogram expansion/skewing and produces good quality images consistently.

Program

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

```
def histogram_equalization(image):
    histogram_array = np.bincount(image.flatten(), minlength=256)

    num_pixels = np.sum(histogram_array)
    histogram_array = histogram_array/num_pixels

    christogram_array = np.cumsum(histogram_array)

    transform_map = np.floor(255 * christogram_array).astype(np.uint8)

    img_list = list(image.flatten())

    eq_img_list = [transform_map[i] for i in img_list]

    eq_img_array = np.reshape(np.asarray(eq_img_list), image.shape)

    return eq_img_array


def cont_stretch(im, levels):
    im_out = np.zeros((im.shape[0],im.shape[1]), dtype=np.uint8)
    a, b = 0, levels-1
    c, d = im.min(), im.max()

    h, w = im.shape
    im_out[0:h, 0:w] = (im[0:h, 0:w] - c)*((b - a)/(d-c)) + a
    return im_out

if __name__ == "__main__":

    image_path =
"/Users/marynavek/Projects/ImageProcessing/reduced_version.jpg"

    image = cv2.imread(image_path, 0)

    plt.imshow(image, cmap='gray')
    plt.show()

    plt.hist(image.ravel(),256,[0,256])
    plt.show()

    transform = histogram_equalization(image)
    plt.imshow(transform, cmap='gray')
    plt.show()

    plt.hist(transform.ravel(),256,[0,256])
    plt.show()
```

```
    transform = cont_stretch(image, 500)
    plt.imshow(transform, cmap='gray')
    plt.show()

    plt.hist(transform.ravel(),256,[0,256])
    plt.show()

    transform = cont_stretch(image, 350)
    plt.imshow(transform, cmap='gray')
    plt.show()

    plt.hist(transform.ravel(),256,[0,256])
    plt.show()

    transform = cont_stretch(image, 150)
    plt.imshow(transform, cmap='gray')
    plt.show()

    plt.hist(transform.ravel(),256,[0,256])
    plt.show()
```

EEL 5820: Digital Image Processing
Fall 2022

Homework # 10

Edge Operators

Submitted by:

Your name: Maryna Veksler

PID#: 5848285

Department of Electrical and Computer Engineering

Date: _____12/12/2022_____

Objective

To Implement the Laplacian of Gaussian filter and determine the different results for the different standard deviations.

Implement two of the edge detection operators (Kirsch and Sobel) and compare their respective results as a function of their edge data extraction for a same gray scale threshold. Again use different images and even to those where you applied the Laplacian of a Gaussian from Homework 7

Method

The Laplacian of Gaussian function operation on the image in frequency domain is implemented as

$$(u^2+v^2)\cdot e^{-\frac{u^2+v^2}{2\sigma^2}}\cdot F(u,v)$$

where σ is the standard deviation of Gaussian function in frequency domain, $F(u,v)$ is the Fourier transform of the image.

Implement the FFT of the image, multiply the Laplacian of Gaussian function with the Fourier transform of the image and do the inverse FFT to reconstruct the Laplacian filtered image.

Results

Example 1: Sobel Filter

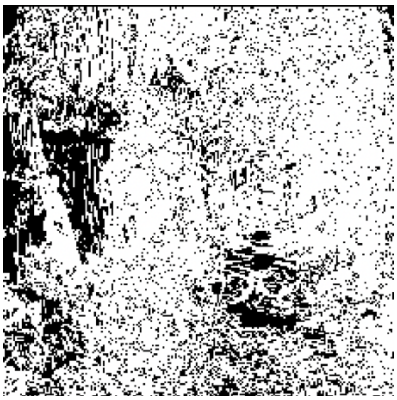
<table>
<tr>
<td> 
Figure 1.1 Original Image</td>
<td> 
Figure 1.2 Alpha = 0.2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</table>

Figure 1.3 Alpha = 0.5

Figure 1.4 Alpha = 0.8

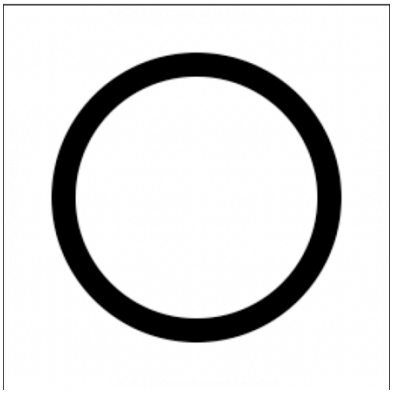


Figure 1.5 Original Image

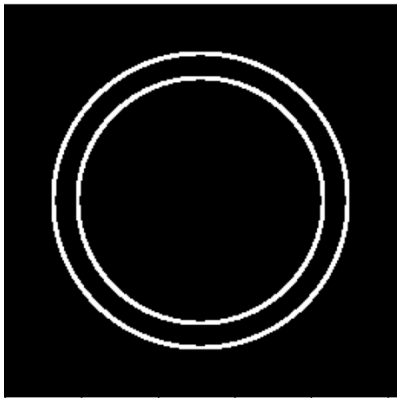


Figure 1.6 Alpha = 0.2

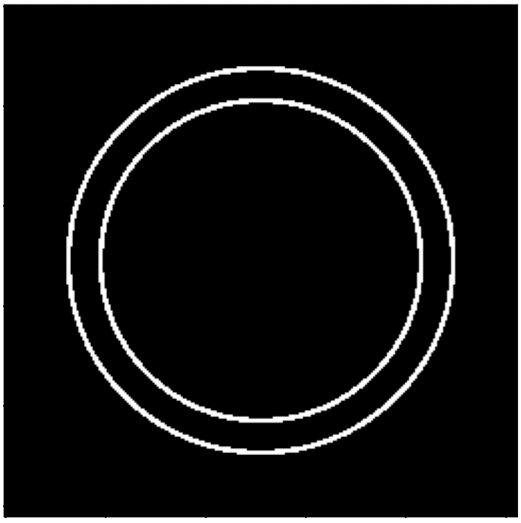


Figure 1.7 Alpha = 0.5

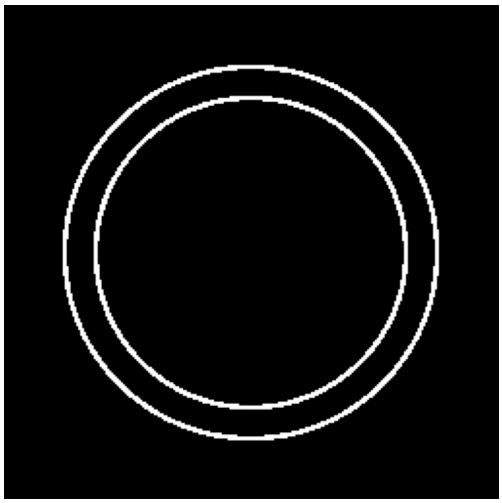


Figure 1.8 Alpha = 0.8

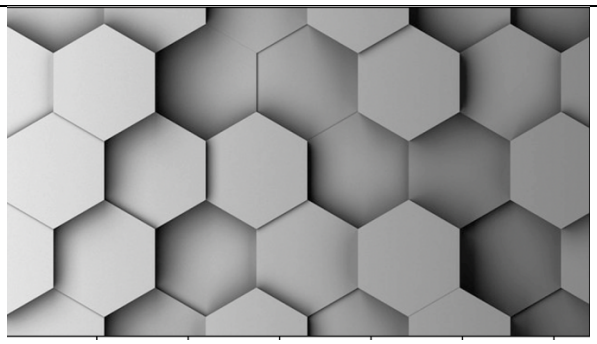


Figure 1.5 Original Image

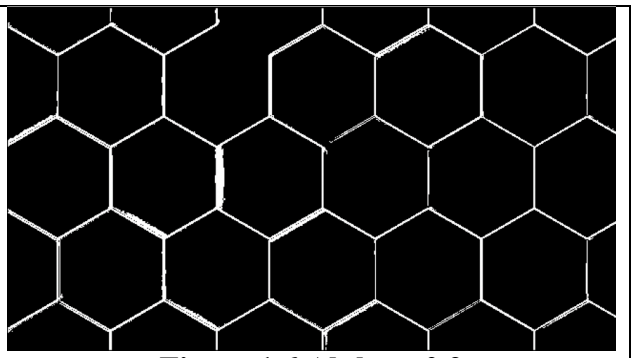


Figure 1.6 Alpha = 0.2

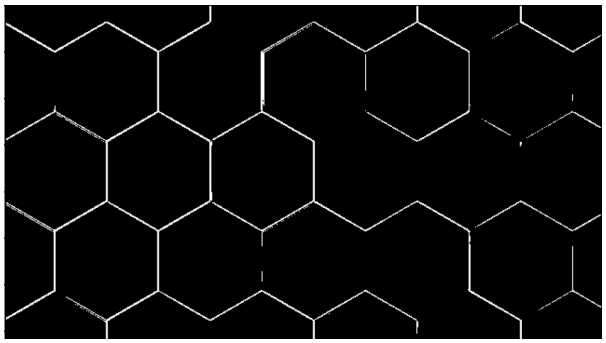


Figure 1.7 Alpha = 0.5

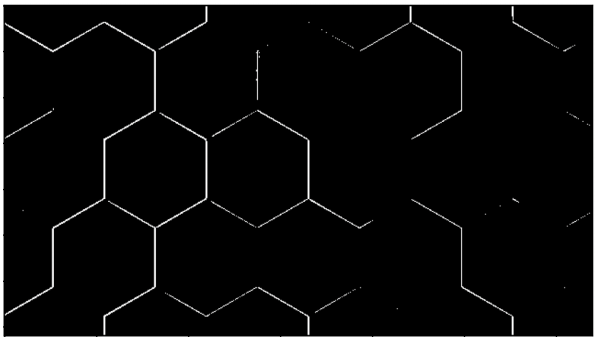


Figure 1.8 Alpha = 0.8

Example 2: Kirsh Filter

Figure 1.1 Original Image

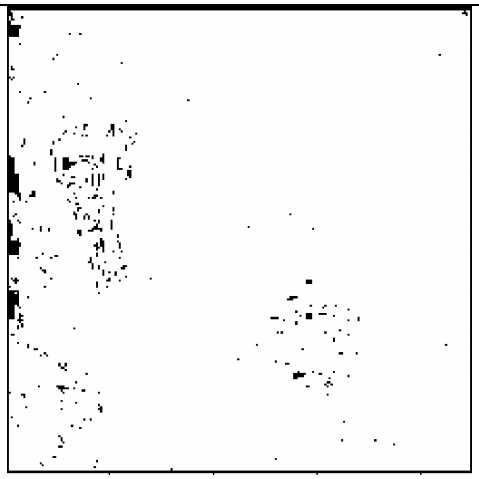


Figure 1.2 Alpha = 0.2

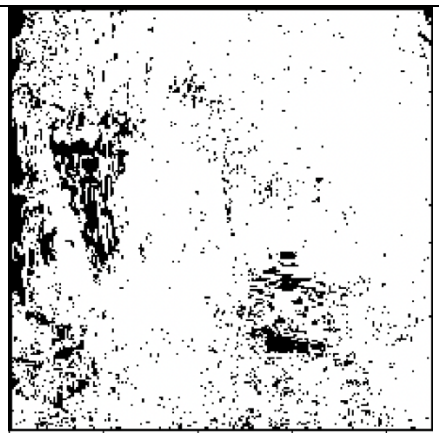


Figure 1.3 Alpha = 0.5

Figure 1.4 Alpha = 0.8

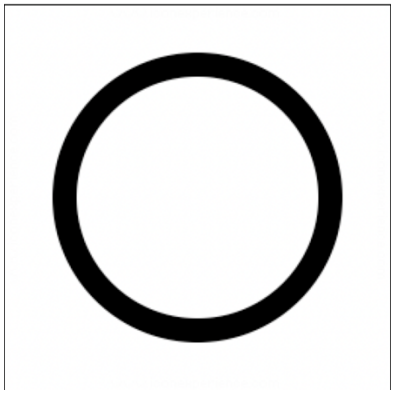


Figure 1.5 Original Image

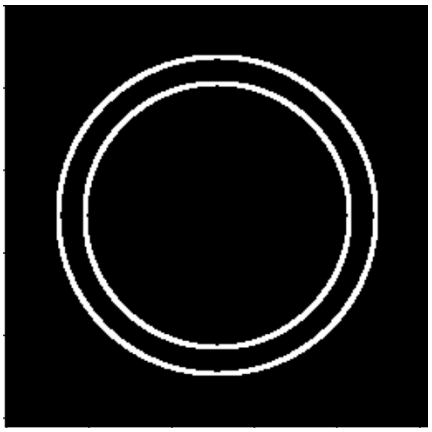


Figure 1.6 Alpha = 0.2

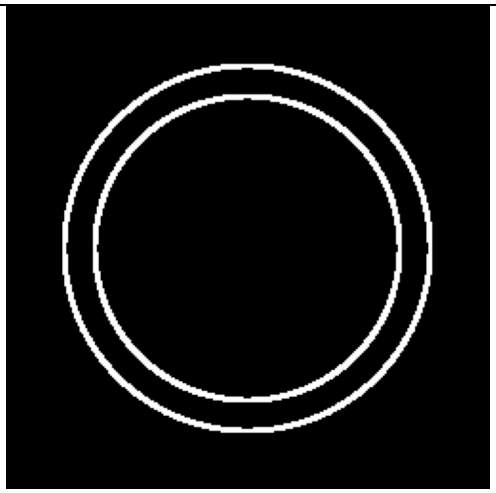


Figure 1.7 Alpha = 0.5

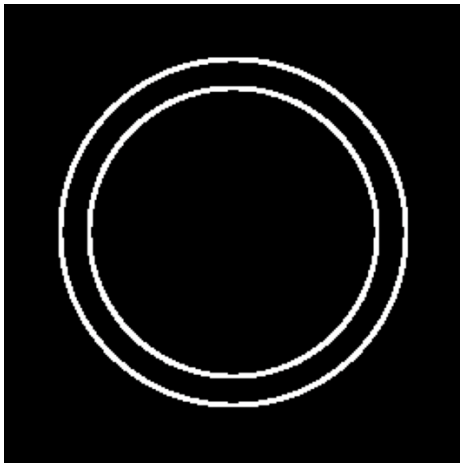


Figure 1.8 Alpha = 0.8

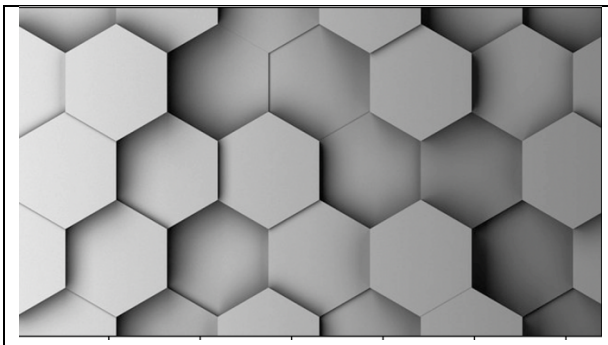


Figure 1.5 Original Image

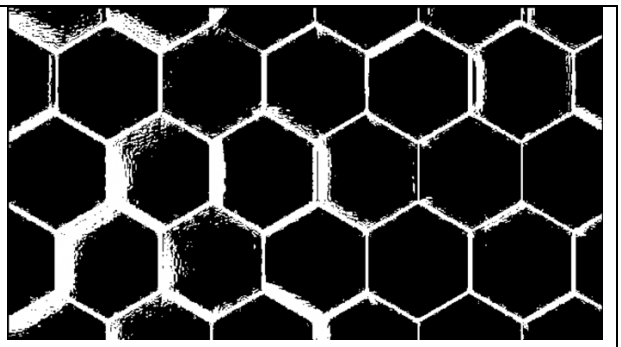


Figure 1.6 Alpha = 0.2

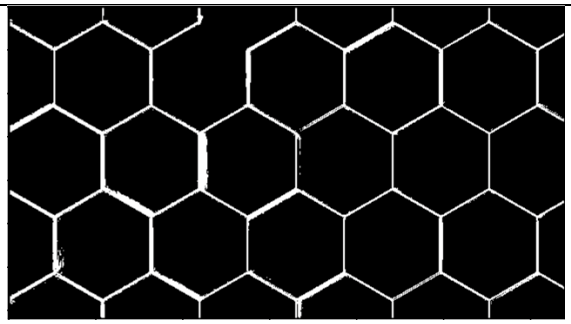


Figure 1.7 Alpha = 0.5

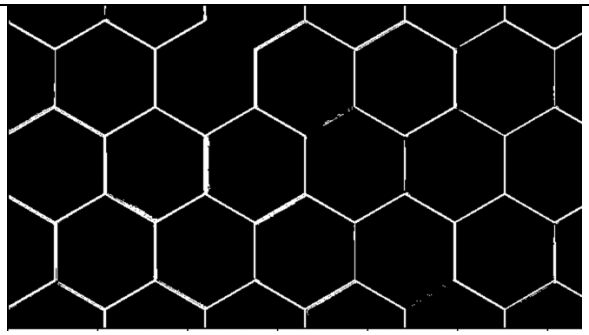


Figure 1.8 Alpha = 0.8

Example 3

Figure 1.1
Original Image

Figure 1.2
Sobel Alpha = 0.8

Figure 1.3 Kirsh Alpha = 0.8

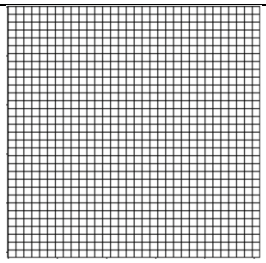


Figure 1.4
Original Image

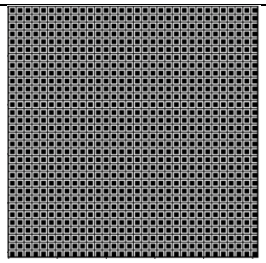


Figure 1.5
Sobel Alpha = 0.8

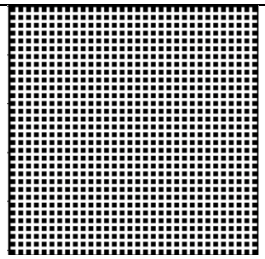


Figure 1.6 Kirsh Alpha = 0.8

Figure 1.7
Original Image

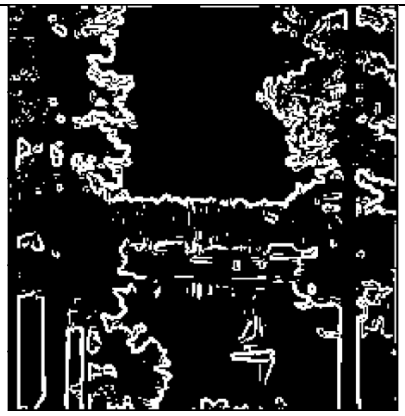


Figure 1.8
Sobel Alpha = 0.8

Figure 1.9 Kirsh Alpha = 0.8

Discussion

In this homework, we compared two edge detectors – Sobel and Kirsh. From the experiments, we determined that the behavior of the detector is affected by the image content. For example, in case of a circle shape, there were no significant changes as we increase or decrease the detector alpha.

When analyzing the Sobel edge detector, we observed that as alpha increase, the number of detected edges decreases significantly. Similar conclusion can be made about the Kirsh detector.

When compared against each other, Kirsh is able to detect edges with higher accuracy. Specifically, the edges appear sharper

Program

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

def kirsch_filter(image, threshold):
    x,y = image.shape
    list=[]
    kirsch = np.zeros((x,y))
    for i in range(2,x-1):
        for j in range(2,y-1):
            d1 = np.square((-3) * image[i - 1, j - 1] - 3 * image[i - 1, j] + 5
* image[i - 1, j + 1] -
```

```
                  3 * image[i, j - 1] + 5 * image[i, j + 1] - 3 * image[i + 1,
j - 1] -
                  3 * image[i + 1, j] + 5 * image[i + 1, j + 1])
            d2 = np.square((-3) * image[i - 1, j - 1] + 5 * image[i - 1, j] + 5
* image[i - 1, j + 1] -
                  3 * image[i, j - 1] + 5 * image[i, j + 1] - 3 * image[i + 1,
j - 1] -
                  3 * image[i + 1, j] - 3 * image[i + 1, j + 1])
            d3 = np.square(5 * image[i - 1, j - 1] + 5 * image[i - 1, j] + 5 *
image[i - 1, j + 1] -
                  3 * image[i, j - 1] - 3 * image[i, j + 1] - 3 * image[i + 1,
j - 1] -
                  3 * image[i + 1, j] - 3 * image[i + 1, j + 1])
            d4 = np.square(5 * image[i - 1, j - 1] + 5 * image[i - 1, j] - 3 *
image[i - 1, j + 1] +
                  5 * image[i, j - 1] - 3 * image[i, j + 1] - 3 * image[i + 1,
j - 1] -
                  3 * image[i + 1, j] - 3 * image[i + 1, j + 1])
            d5 = np.square(5 * image[i - 1, j - 1] - 3 * image[i - 1, j] - 3 *
image[i - 1, j + 1] +
                  5 * image[i, j - 1] - 3 * image[i, j + 1] + 5 * image[i + 1,
j - 1] -
                  3 * image[i + 1, j] - 3 * image[i + 1, j + 1])
            d6 = np.square((-3) * image[i - 1, j - 1] - 3 * image[i - 1, j] - 3
* image[i - 1, j + 1] +
                  5 * image[i, j - 1] - 3 * image[i, j + 1] + 5 * image[i + 1,
j - 1] +
                  5 * image[i + 1, j] - 3 * image[i + 1, j + 1])
            d7 = np.square((-3) * image[i - 1, j - 1] - 3 * image[i - 1, j] - 3
* image[i - 1, j + 1] -
                  3 * image[i, j - 1] - 3 * image[i, j + 1] + 5 * image[i + 1,
j - 1] +
                  5 * image[i + 1, j] + 5 * image[i + 1, j + 1])
            d8 = np.square((-3) * image[i - 1, j - 1] - 3 * image[i - 1, j] - 3
* image[i - 1, j + 1] -
                  3 * image[i, j - 1] + 5 * image[i, j + 1] - 3 * image[i + 1,
j - 1] +
                  5 * image[i + 1, j] + 5 * image[i + 1, j + 1])

            list=[d1, d2, d3, d4, d5, d6, d7, d8]
            kirsch[i,j]= int(np.sqrt(max(list)))

    for i in range(x):
        for j in range(y):
            if kirsch[i,j]>255*threshold:
                kirsch[i,j]=255
            else:
                kirsch[i,j]=0
```

```
    return kirsch


def sobel_filter(image, threshold):
    #define horizontal and Vertical sobel kernels
    Gx = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
    Gy = np.array([[-1, -2, -1],[0, 0, 0],[1, 2, 1]])

    x,y  = image.shape
    sobel = np.zeros(shape=(x,y))

    for i in range(x - 2):
        for j in range(y - 2):
            gx = np.sum(np.multiply(Gx, image[i:i + 3, j:j + 3]))
            gy = np.sum(np.multiply(Gy, image[i:i + 3, j:j + 3]))
            sobel[i + 1, j + 1] = np.sqrt(gx ** 2 + gy ** 2)

    for i in range(x):
        for j in range(y):
            if sobel[i,j]>255*threshold:
                sobel[i,j]=255
            else:
                sobel[i,j]=0

    return sobel

if __name__ == "__main__":

    image_path = "/Users/marynavek/Projects/ImageProcessing/natural_scene.png"

    image = cv2.imread(image_path, 0)
    plt.imshow(image, cmap='gray')
    plt.show()

    transform = kirsch_filter(image, 0.2)
    plt.imshow(transform, cmap='gray')
    plt.show()

    transform = sobel_filter(image, 0.4)
    plt.imshow(transform, cmap='gray')
    plt.show()

    transform = kirsch_filter(image, 0.5)
    plt.imshow(transform, cmap='gray')
    plt.show()

    transform = sobel_filter(image, 0.8)
    plt.imshow(transform, cmap='gray')
```

```
    plt.show()

    transform = kirsch_filter(image, 0.8)
    plt.imshow(transform, cmap='gray')
    plt.show()
```

EEL 5820: Digital Image Processing
Fall 2022

Homework # 11

JPEG Compression

Submitted by:

Your name: Maryna Veksler

PID#: 5848285

Department of Electrical and Computer Engineering

Date: _____12/12/2022_____

Objective

In this work we implement JPEG compression using quantization matrix and DCT compression.

Method

The compression requires following steps:

1. Identify quantization matrix

```
Q = np.array([[16, 11, 10, 16, 24, 40, 52, 61],
              [12, 12, 14, 19, 26, 58, 60, 55],
              [14, 13, 16, 24, 40, 57, 69, 56],
              [14, 17, 22, 29, 51, 87, 80, 62],
              [18, 22, 37, 56, 68, 109, 103, 77],
              [24, 35, 55, 64, 81, 104, 113, 92],
              [49, 64, 78, 87, 103, 121, 120, 101],
              [72, 92, 95, 98, 112, 100, 103, 99]])
```

2. Split image into 8x8 blocks
3. For each block: (1) apply DCT transform and (2) multiply with quantization matrix.

NOTE: Omitting the encoding part for simplicity

Results

Example 1: Compressing with level = 1
Image 1, SNR: 0.9615
Image 2, SNR: 0.8032
Image 3, SNR: 0.0597

Figure 1.1 Original Image

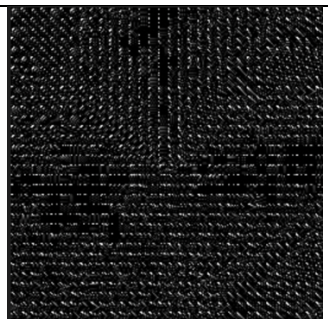


Figure 1.2 DCT

Figure 1.3 Decompressed

Figure 1.4 Original Image

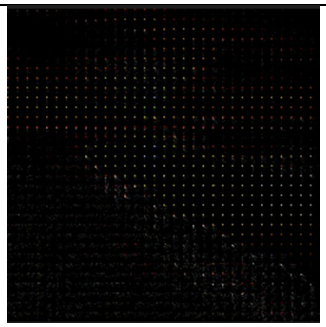


Figure 1.5 DCT

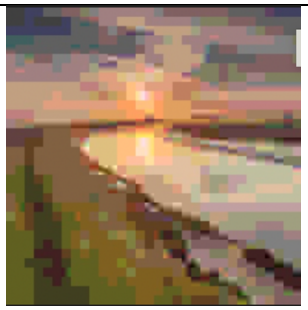


Figure 1.6 Decompressed

Figure 1.7 Original Image

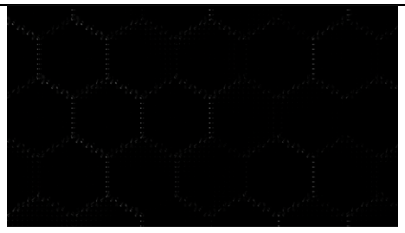


Figure 1.8 DCT

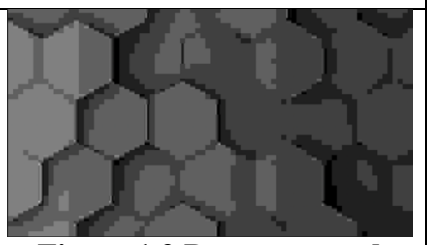


Figure 1.9 Decompressed

Example 2: Compressing with level = 3
Image 1, SNR: 0. 9643
Image 2, SNR: 0. 8291
Image 3, SNR: 0. 0742

Figure 2.1 Original Image

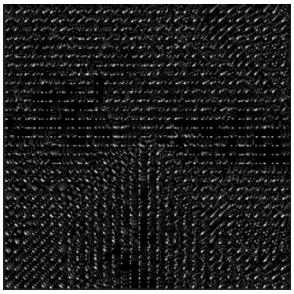


Figure 2.2 DCT

Figure 2.3 Decompressed

Figure 2.4 Original Image

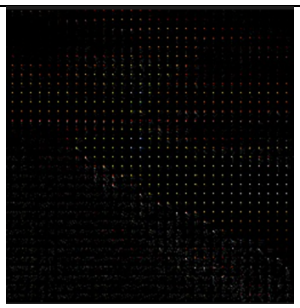


Figure 2.5 DCT

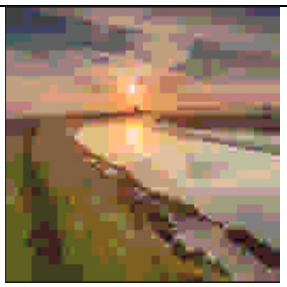


Figure 2.6 Decompressed

Figure 2.7 Original Image

Figure 2.8 DCT

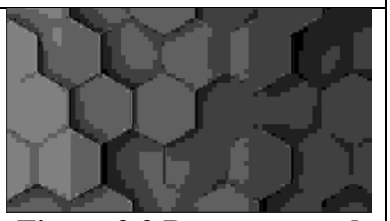


Figure 2.9 Decompressed

Example 3: Compressing with level = 5
Image 1, SNR: 0. 9723
Image 2, SNR: 1.5308
Image 3, SNR: 1.2007

Figure 3.1 Original Image

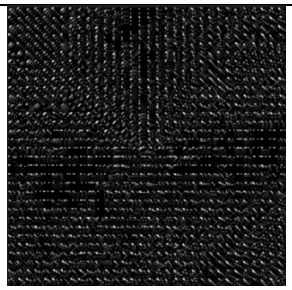


Figure 3.2 DCT

Figure 3.3 Decompressed

Figure 3.4 Original Image

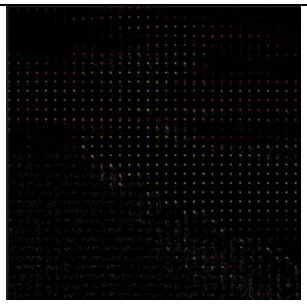


Figure 3.5 DCT

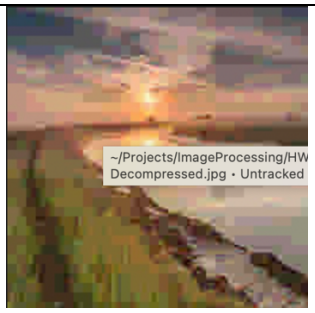


Figure 3.6 Decompressed

Figure 3.7 Original Image

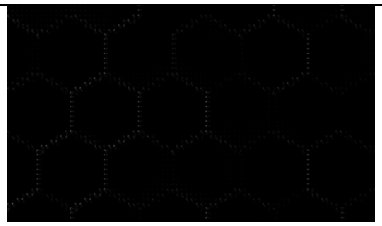


Figure 3.8 DCT

Figure 3.9 Decompressed

Discussion

In this homework, we implemented and applied JPEG compression using DCT. From the results, we can observe that as the level of compression increases the signal to noise ratio between original and compressed image increases proportionally.

Throughout the experiments, we demonstrated that regardless of the compression level, the JPEG compression indeed is valuable as we were able to decrease the total size of each image by 2/3 on average, and the final result was close to 7KB.

Program

```
import numpy as np
```

```
import cv2
from matplotlib import pyplot as plt
from numpy.linalg import inv
import time
import sys

def dct_coeff():
    T = np.zeros([8, 8])
    for i in range(8):
        for j in range(8):
            if i == 0:
                T[i, j] = 1/np.sqrt(8)
            elif i > 0:
                T[i, j] = np.sqrt(2/8)*np.cos((2*j+1)*i*np.pi/16)
    return T

def quantization_level(n):
    Q50 = np.zeros([8, 8])

    Q50 = np.array([[16, 11, 10, 16, 24, 40, 52, 61],
                    [12, 12, 14, 19, 26, 58, 60, 55],
                    [14, 13, 16, 24, 40, 57, 69, 56],
                    [14, 17, 22, 29, 51, 87, 80, 62],
                    [18, 22, 37, 56, 68, 109, 103, 77],
                    [24, 35, 55, 64, 81, 104, 113, 92],
                    [49, 64, 78, 87, 103, 121, 120, 101],
                    [72, 92, 95, 98, 112, 100, 103, 99]])

    Q = np.zeros([8, 8])
    for i in range(8):
        for j in range(8):
            if n > 50:
                Q[i, j] = min(np.round((100-n)/50*Q50[i, j]), 255)
            else:
                Q[i, j] = min(np.round(50/n * Q50[i, j]), 255)
    return Q

def quantiz_div(a, b):
    tmp = np.zeros(a.shape)
    for i in range(8):
        for j in range(8):
            tmp[i, j] = np.round(a[i, j]/b[i, j])
    return tmp

def quantiz(D, Q):
    tmp = np.zeros(D.shape)
    mask = np.zeros([8, 8])
    for i in range(D.shape[0]//8):
```

```
        for j in range(D.shape[1]//8):
            mask = quantiz_div(D[8*i:8*i+8, 8*j:8*j+8], Q)
            tmp[8*i:8*i+8, 8*j:8*j+8] = mask
    return (tmp)

def decompress_mul(a, b):
    tmp = np.zeros(a.shape)
    for i in range(8):
        for j in range(8):
            tmp[i, j] = a[i, j]*b[i, j]
    return tmp

def decompress(C, Q, T, T_prime):
    R = np.zeros(C.shape)
    mask = np.zeros([8, 8])
    for i in range(C.shape[0]//8):
        for j in range(C.shape[1]//8):
            mask = decompress_mul(C[8*i:8*i+8, 8*j:8*j+8], Q)
            R[8*i:8*i+8, 8*j:8*j+8] = mask

    N = np.zeros(C.shape)

    for i in range(R.shape[0]//8):
        for j in range(R.shape[1]//8):
            mask = T_prime @ R[8*i:8*i+8, 8*j:8*j+8] @ T
            N[8*i:8*i+8, 8*j:8*j+8] = np.round(mask) + 128*np.ones([8, 8])

    return N

def Compress_img(file, level):

    I = cv2.imread(file)

    B, G, R = cv2.split(I)

    H = I.shape[0]
    W = I.shape[1]

    print("Image size: ", I.shape)

    B = B - 128*np.ones([H, W])
    G = G - 128*np.ones([H, W])
    R = R - 128*np.ones([H, W])

    T = dct_coeff()
    T_prime = inv(T)
    Q = quantization_level(level)
```

```
    D_R = dct(R, T, T_prime)
    D_G = dct(G, T, T_prime)
    D_B = dct(B, T, T_prime)

    tmp = cv2.merge((D_B, D_G, D_R))

    cv2.imwrite('DCT.jpg', tmp)

    C_R = quantiz(D_R, Q)
    C_R[C_R == 0] = 0
    C_G = quantiz(D_G, Q)
    C_G[C_G == 0] = 0
    C_B = quantiz(D_B, Q)
    C_B[C_B == 0] = 0

    tmp = cv2.merge((C_B, C_G, C_R))

    cv2.imwrite('After_Quantiz.jpg', tmp)
    return C_B, C_G, C_R, Q, T, T_prime

def Decompress_img(C_B, C_G, C_R, Q, T, T_prime):
    N_R = decompress(C_R, Q, T, T_prime)
    N_G = decompress(C_G, Q, T, T_prime)
    N_B = decompress(C_B, Q, T, T_prime)

    N_I = cv2.merge((N_B, N_G, N_R))
    cv2.imwrite('Decompressed.jpg', N_I)

def Evaluate(file):

    I = cv2.imread(file)

    I1 = cv2.imread("Decompressed.jpg")

    m, n, k = I1.shape

    rms = np.sqrt(np.sum(np.square(I1-I)))/(m*n)

    snr = np.sum(np.square(I1))/np.sum(np.square(I1-I))

    return rms, snr

def dct(M,T,T_prime):
    dct_res = np.zeros(M.shape)
    mask = np.zeros([8,8])
    for i in range(M.shape[0]//8):
        for j in range(M.shape[1]//8):
            mask = M[8*i:8*i+8,8*j:8*j+8]
```

```
            dct_res[8*i:8*i+8,8*j:8*j+8] = T @ mask @ T_prime

    return (dct_res)

if __name__ == "__main__":
    file = "/Users/marynavek/Projects/ImageProcessing/synthetic_im_3.jpeg"
    level = 5
    print("Filename: ", file)
    print("Level of compression: ", level)

    print("Compressing....")
    start = time.time()
    C_B, C_G, C_R, Q, T, T_prime = Compress_img(file, level)
    time_comp = time.time()
    print("Compression Time: ", np.round(time_comp - start, 1), " sec")

    print("Decompressing...")
    Decompress_img(C_B, C_G, C_R, Q, T, T_prime)
    time_decomp = time.time()

    print("Decompression Time: ", np.round(time_decomp - time_comp, 1), " sec")

    end = time.time()
    print("Total: ", np.round(end - start, 1), " sec")
    rms, snr = Evaluate(file)
    print("RMS: ", np.round(rms, 4))
    print("SNR: ", np.round(snr, 4))
```