Digital Image Processing Lab
Study Notes

Farhin Mashiat Mayabee

Update on October 12, 2022

Assignment-1

1 Histogram Generation

We have made histogram for the grayscale, red channel, green channel, blue channel and binary image of an RGB image. We used matplotlibs hist() function to calculate the histogram.
We generated binary using open-cv.
Used $, binary = cv2.threshold(grayscale, 50, 255, cv2.THRESH-BINARY) for binary.$
$Also used cv2 built in cvtColor() function to generate grayscale image.$

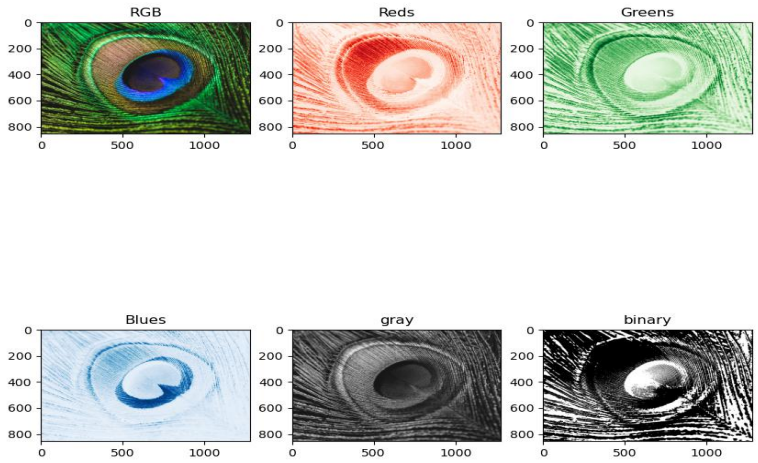

Figure 1: grayscale, red channel, green channel, blue channel and binary image of an RGB image

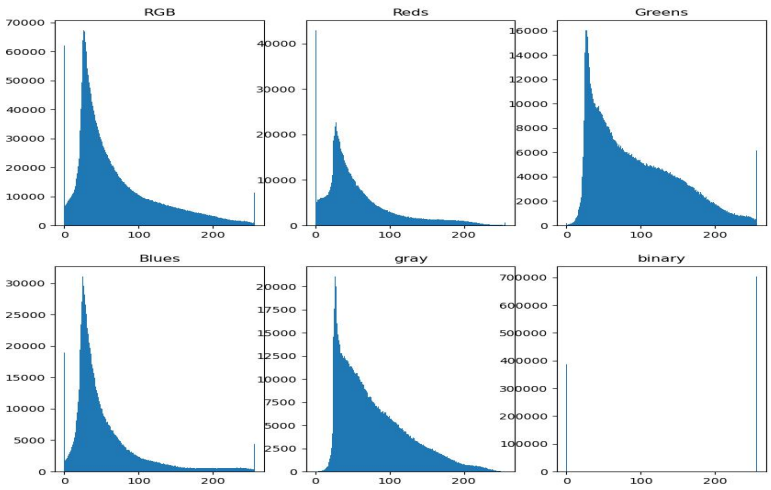

Figure 2: Histogram of grayscale, red channel, green channel, blue channel and binary image of an RGB image

Assignment 2

2 Point processing on a 2D image

Let 'r' is the old intensity of a pixel and 's' is the new intensity. Let 'c' and 'p' are two positive constants. You can assume any value for 'c and 'p'. For 'epsilon' choose very small value, such as 0.0000001. 'T1' and 'T2' are two thresholds. You can assume any value in the range 0 - 255.

Check what happens for the following transformations:
s = 100, if r ¿= T1 and r ¡= T2; otherwise
s = 10.
s = 100, if r ¿= T1 and r ¡= T2; otherwise
s = r.
s = c log(1 + r) .
s = c (r + epsilon) ^{p}
Check what happens for the following transformations:
s = 100, if r ¿= T1 and r ¡= T2; otherwise
s = 10.
s = 100, if r ¿= T1 and r ¡= T2; otherwise
s = r.
s = c log(1 + r) .
s = c (r + epsilon) ^{p}**After applying the above operation to my image we get the following results.**

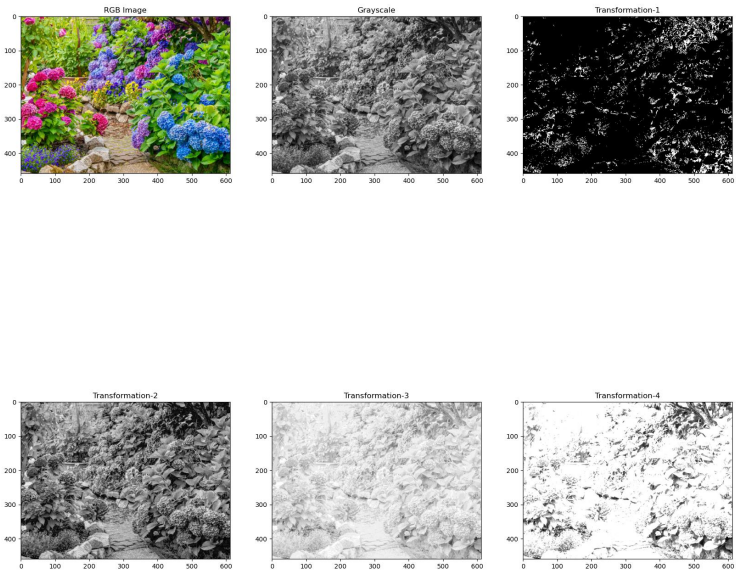

Figure 3: Point Processing

Assignment 3

3 The effect of 6 different kinds of kernels on a 2D image.

```
kernel1 = np.array([[-1,-1,-1], [-1,8,-1], [-1,-1,-1]])
  print('kernel1: {}'.format(kernel1))
  processed_img1 = cv2.filter2D(grayscale,-1,kernel1)
  kernel2 = np.array([[-1,0,-1], [-1,8,-1], [-1,0,-1]])
  print('kernel2: {}'.format(kernel2))
  processed_img2 = cv2.filter2D(grayscale,-1,kernel2)

  kernel3 = np.array([[0,0,0], [0,1,0], [0,0,0]])
  print('kernel3: {}'.format(kernel3))
  processed_img3 = cv2.filter2D(grayscale,-1,kernel3)

  kernel4 = np.array([[0,-1,0], [-1,5,-1], [0,-1,0]])
  print('kernel4: {}'.format(kernel4))
  processed_img4 = cv2.filter2D(grayscale,-1,kernel4)

  kernel5 = np.array([[1,1,1], [1,1,1], [1,1,1]])*1/9
  print('kernel5: {}'.format(kernel5))
  processed_img5 = cv2.filter2D(grayscale,-1,kernel5)

  kernel6 = np.array([[1,1,1], [1,1,1], [1,1,1]])
  print('kernel6: {}'.format(kernel6))
  processed_img6 = cv2.filter2D(grayscale,-1,kernel6)
```

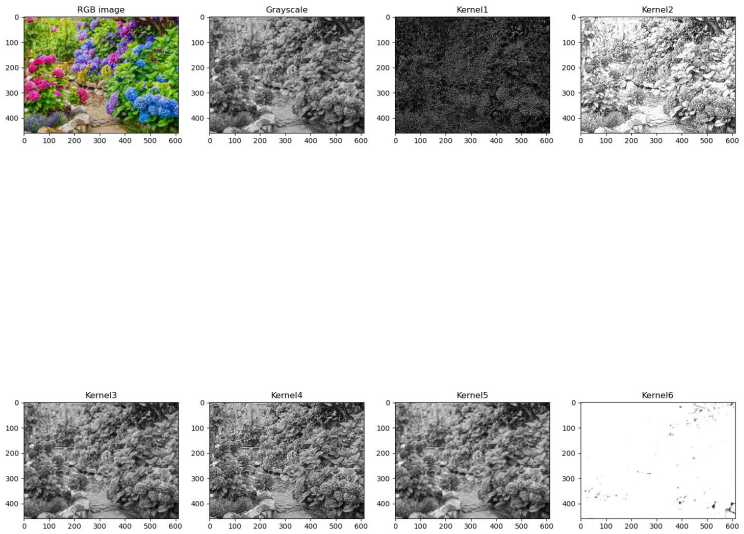

Figure 4: Point Processing

Assignment-4

4 Implement Histogram and Built-in histogram function comparison

We have made histogram of a grayscale image. We used matplotlib.hist() function to calculate the histogram.
The implementation code of histogram is given below.

```
h = np.zeros(256)
n = range(0, 256)
for i in range(r):
```

```
        for j in range(c):
            h[gray[i,j]] = h[gray[i,j]]+1
    plt.subplot(2,1,1)
    plt.stem(n,h)h = np.zeros(256)
    n = range(0, 256)
    for i in range(r):
        for j in range(c):
            h[gray[i,j]] = h[gray[i,j]]+1
    plt.subplot(2,1,1)
    plt.stem(n,h)
```

The output both functions provide for the same image is shown below.

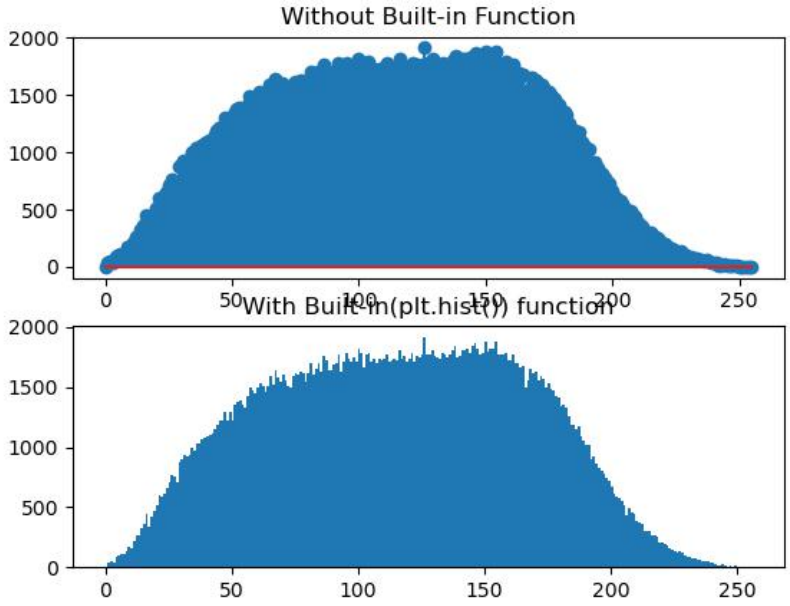

Figure 5: Builtin histogram VS implemented Histogram

Implemented function is quite similar to the built-in function of matplotlib.hist() function.

5 Neighborhood processing using built-in and implemented function

The neighborhood processing using cv2.filter2D() and implemented function showed very different results with the same kernel.
At first, we added padding to the image and then done the convolution operation on the image.

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

kernel shows different outputs for the same image.

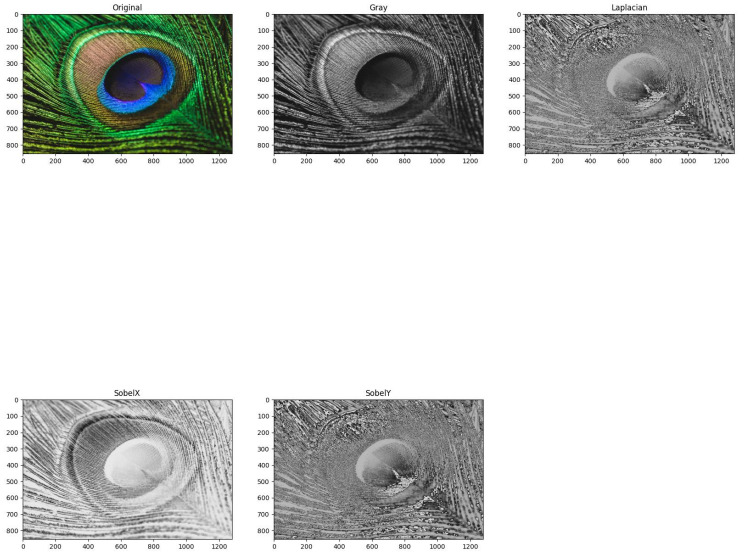

Figure 6: Builtin histogram VS implemented Histogram

Assignment 5

6 Binary Masking

We took two similar sized photos and converted one of them into binary image and the other into grayscale image. Then used and operation to mask the second grayscale image with the binary of the first image.

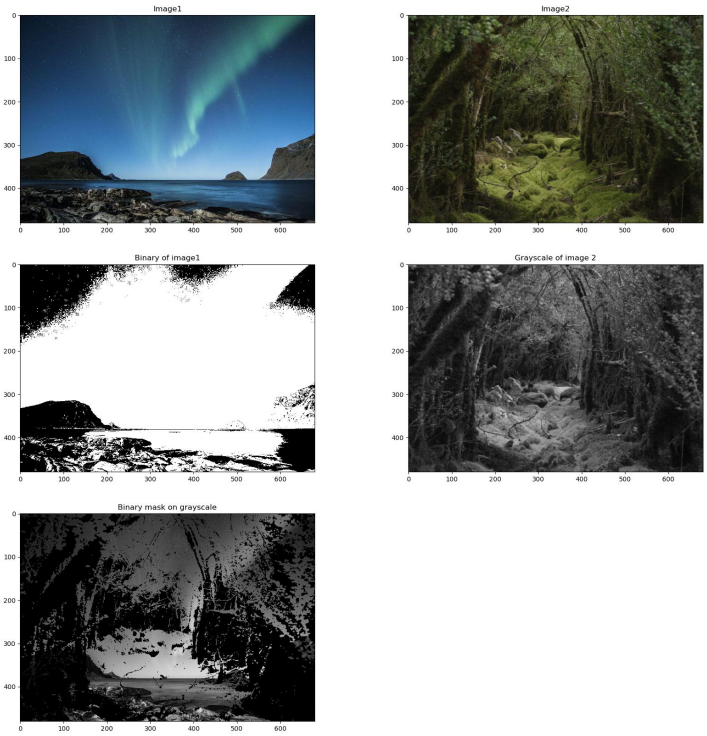

Figure 7: Binary Masking

7 Bit Slicing

We sliced an 8-bit grayscale image into 8 planes. Bit plane slicing is a method of representing an image with one or more bits of the byte used for each pixel. One can use only MSB to represent the pixel, which reduces the original gray level to a binary image. The three main goals of bit plane slicing is: Converting a gray level image to a binary image.
The gray level of each pixel in a digital image is stored as one or more bytes in a computer. For an 8-bit image, 0 is encoded as 00000000 and 255 is encoded as 11111111. Any number between 0 t0 255 is encoded as one byte. The bit in the far left side is referred to as the most significant bit (MSB) because a change in that bit would significantly change the value encoded by the byte.
The bit in the far right is referred to as the least significant bit (LSB), because a change in this bit does not change the encoded gray value much. The bit plane representation of an eight-bit digital image is given by:

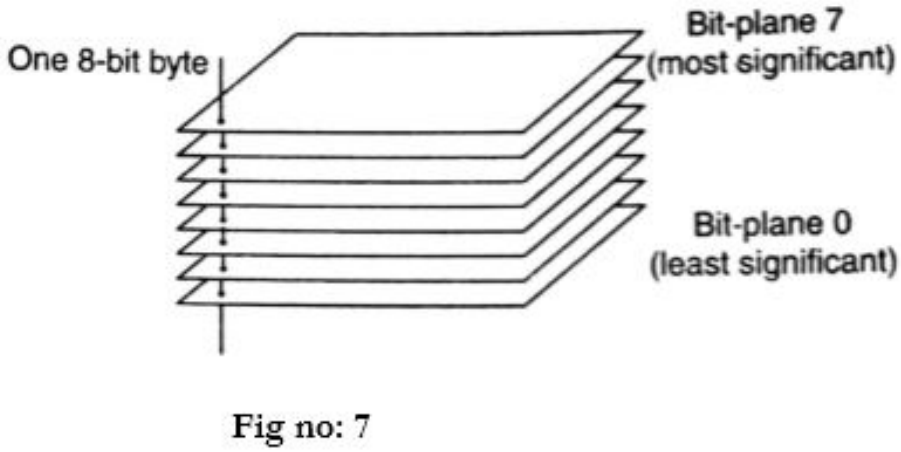

Fig no: 7

Figure 8: Visual representation of bit slicing

Here is the bit sliced of an image.
code for the bit slicing

```
for k in range(8):
        p = n<<k
        print(p)
        processed_img= cv2.bitwise_and(gray, p, mask=None)
```

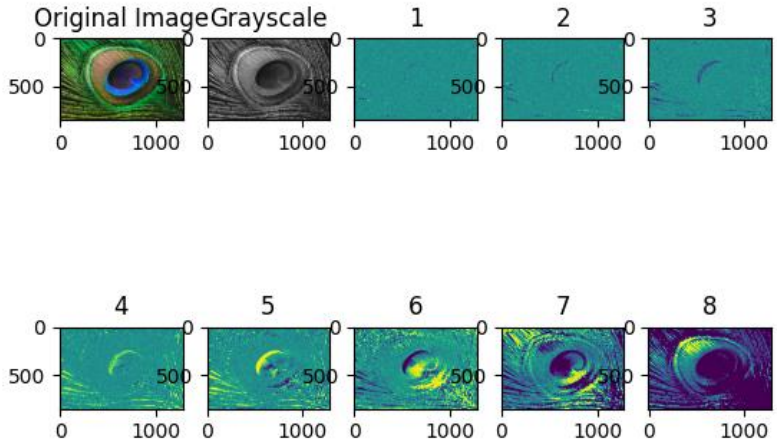

Figure 9: Bit Slicing

8 Convolution of a grayscale image with a Laplacian filters and Sobel filters.

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection (see zero crossing edge detectors). The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single graylevel image as input and produces

another graylevel image as output.
The Sobel method, or Sobel filter, is a gradient-based method that looks for strong changes in the first derivative of an image. The Sobel edge detector uses a pair of 3 $\times$ 3 convolution masks, one estimating the gradient in the x-direction and the other in the y-direction.
This filter is used for edge ditection. Vertical and Horizontal.

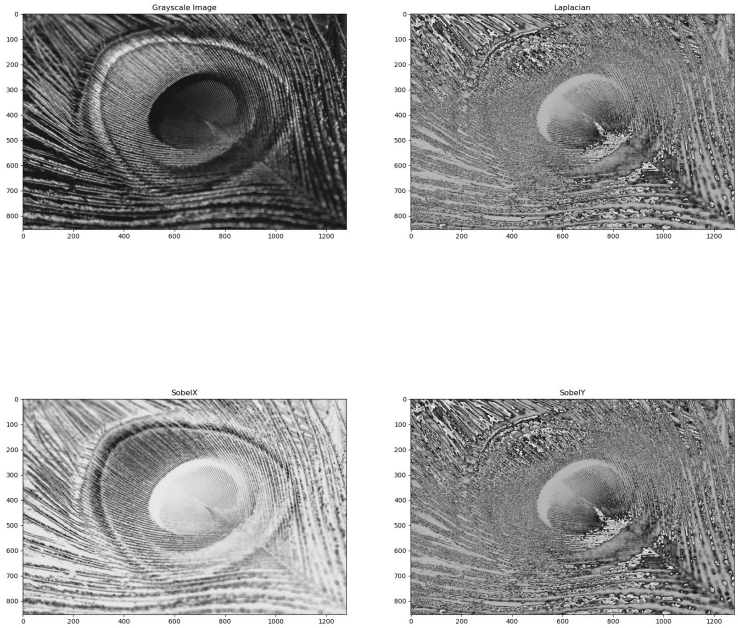

Figure 10: Bit Slicing

Assignment 6

9 Salt Pepper Noise

Salt-and-pepper noise, also known as impulse noise, is a form of noise sometimes seen on digital images. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels.
this noise is totally removed when median filter is used.

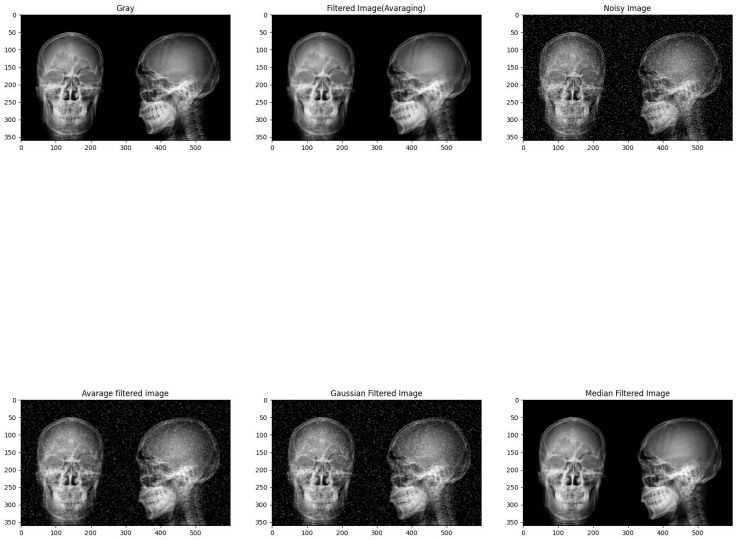

Figure 11: Salt Pepper Noise

Assignment 7

10 Histogram Shifting

We shifted the histogram left, right and middle. We moved intensity of a grayscale image left, right and into a specific range and check its effect on histogram.
Code for doing that is bellow.

```
right = np.zeros((r,c), dtype=np.uint8)
    left = np.zeros((r,c), dtype=np.uint8)
    mid = np.zeros((r,c), dtype=np.uint8)

    for i in range(r):
        for j in range(c):
            temp = gray[i,j]
            temp = temp + 120
            if temp > 255:
                temp = 255
            right[i,j] = temp
    for i in range(r):
        for j in range(c):
            temp = gray[i,j]
            temp = temp - 120
            if temp < 0:
                temp = 0
            left[i,j] = temp
    for i in range(r):
        for j in range(c):
            temp = gray[i,j]
            if temp < 50:
                temp = 50
            elif temp >200:
                temp = 200
            mid[i,j] = temp
```

As the histogram is moved to the right the picture becomes brighter. and the pictur becomes darker for left shift.

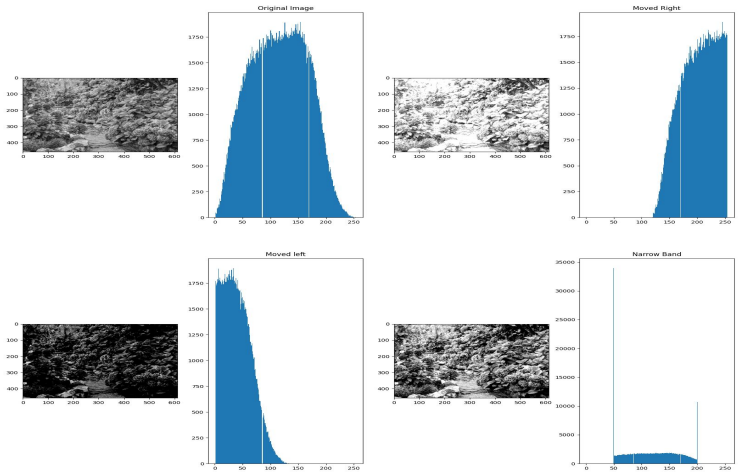

Figure 12: Histogram Intensity Shift

Assignment 8

11 Morphological Operations

In erosion, I have found that the picture gets darker, and the black part expands.The more I increased the dimension of the kernel, the image gets more blurry and with fewer details. The image loses many details.
code for erosion

```
image_eroded = cv2.erode(binary, kernel)
```

Dilation:

In dilation, the white part expands and the black part shrinks. opposite of the erosion. And also the more the size of the kernel, the more white is on the output image.
code for dilation

```
image_dilated = cv2.dilate(binary, kernel)
```

Opening:

In opening, the output was clear. less white noises. it can be used for internal noise cancelation.

code for opening

```
opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN,kernel, iterations=1)
```

Closing:

In closing, the output was with more white pixels. It can be used as outer damage fixing.
code for closing

```
closing = cv2.morphologyEx(binary, cv2.MORPH_Close,kernel, iterations=1)
```

Figure 13: Morphological Operations

Assignment 9

12 Morphological Operations

Erosion:

In erosion, I have found that the picture gets darker, and the black part expands.The more I increased the dimension of the kernel, the image gets more blurry and with fewer details. The image loses many details.
code for erosion

```
image_eroded = cv2.erode(binary, kernel)
```

Dilation:

In dilation, the white part expands and the black part shrinks. opposite of the erosion. And also the more the size of the kernel, the more white is on the output image.
code for dilation

```
image_dilated = cv2.dilate(binary, kernel)
```

Opening:

In opening, the output was clear. less white noises. it can be used for internal noise cancelation.
code for opening

```
opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN,kernel, iterations=1)
```

Closing:

In closing, the output was with more white pixels. It can be used as outer damage fixing.
code for closing

```
closing = cv2.morphologyEx(binary, cv2.MORPH_Close,kernel, iterations=1)
```

Here is my implemented functions

```
def opening(img, element):
    r,c = img.shape
    img1 = mor_erode(img, element)
    img2 = mor_dilate(img1, element)
    return img2
def closing(img, element):
    r,c = img.shape
    img1 = mor_dilate(img, element)
```

```
    img2 = mor_erode(img1, element)
    return img2

def mor_dilate(img, element):
    r,c = img.shape
    m,n = element.shape
    p = (m//2)
    proc = np.zeros((r,c), dtype = np.int8)
    img = np.pad(img, p, constant_values = 0)

    for i in range(r):
        for j in range(c):
            res = np.sum(img[i:i+m, j:j+n] * element)
            if res > 0:
                proc[i,j] = 255
    return proc

def mor_erode(img, element):
    r,c = img.shape
    m,n = element.shape
    p = (m//2)
    proc = np.zeros((r,c), dtype = np.int8)
    img = np.pad(img, p, constant_values = 0)

    for i in range(r):
        for j in range(c):
            res = np.sum(img[i:i+m, j:j+n] * element)
            if res  == 255*m*n:
                proc[i,j] = 255
    return proc
```

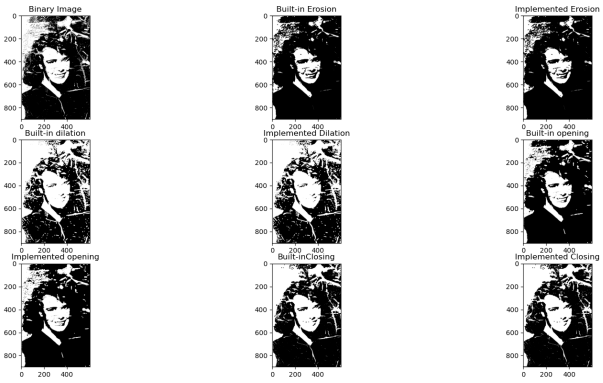

Figure 14: Morphological Operations

Assignment 10

13 Histogram Equalization

Applied histogram equalization on an image. it balanced the image. We can observe that if the histogram is equalized the image becomes better. It is used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image.

```
equalizedImg = cv2.equalizeHist(grayscale)
```

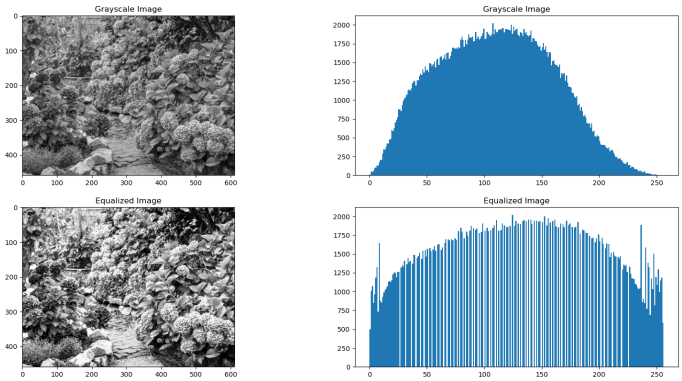

Figure 15: histogram Equalization

Assignment 11

14 FFT

Frequency domain filters are different from spatial domain filters as it mainly focuses on the frequency of the images. It is done for two basic operations i.e., Smoothing and Sharpening.

Let us perform some Domain Filter using numpy.fft.fft2() method.

In this, we see what happens when we apply filtering in frequency domain.

Frequency Domain Filters are used for smoothing and sharpening of images by removal of high or low-frequency components.

Frequency domain filters are different from spatial domain filters as it mainly focuses on the frequency of the images. It is done for two basic operations i.e., Smoothing and Sharpening.

Let us perform some Domain Filter using numpy.fft.fft2() method.

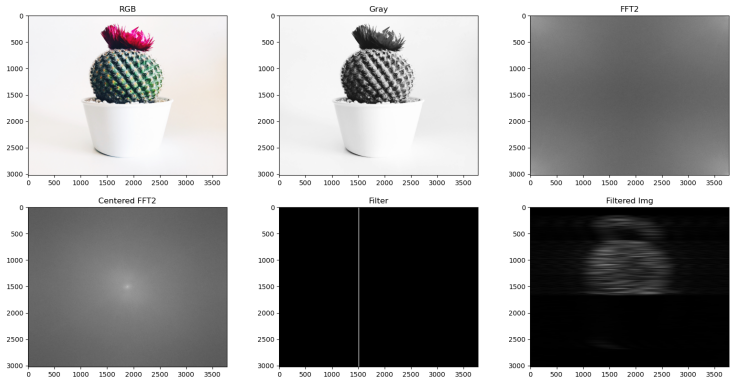

Figure 16: FFT

Assignment 13

15 Turn a JPEG image into a PNG image and vice-versa.

Here we used open-cv for converting image format.

```
import cv2
def main():
    img_path = "img.jpg"
    #img_path = "bird.png"
    name,format = img_path.split(".")
    if format == 'jpeg' or format == 'jpg':
        img = cv2.imread(img_path)
        cv2.imwrite(name+".png", img)
    else:
        img = cv2.imread(img_path)
        cv2.imwrite(name+".jpeg", img)
if __name__ == "__main__":
    main()
```