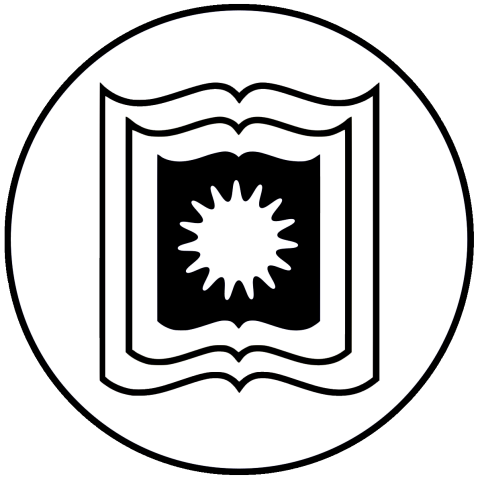


University of Rajshahi

Department of Computer Science & Engineering

CSE4182 - Digital Image Processing Lab

Final Assignment

Submission Date: November 11, 2022

Prepared by:

Monirul Islam

ID: 1810476144

Part 4, Odd Semester

Session: 2017-18

Submitted To:

Dr. Md. Khademul Islam Molla

Professor, University of Rajshahi

Dr. Md. Rokanujjaman

Professor, University of Rajshahi

Dr. Sangeeta Biswas

Associate Professor,

University of Rajshahi

Contents

1 Assignment 1

1.1 Problem statement

Plot histogram of the grayscale, red channel, green channel, blue channel and binary image of an RGB image.

1.2 Input Image

1.3 Method

The histogram is the graphical representation of the intensity distribution of an image. We used opencv built-in function calcHist().

1.4 Code

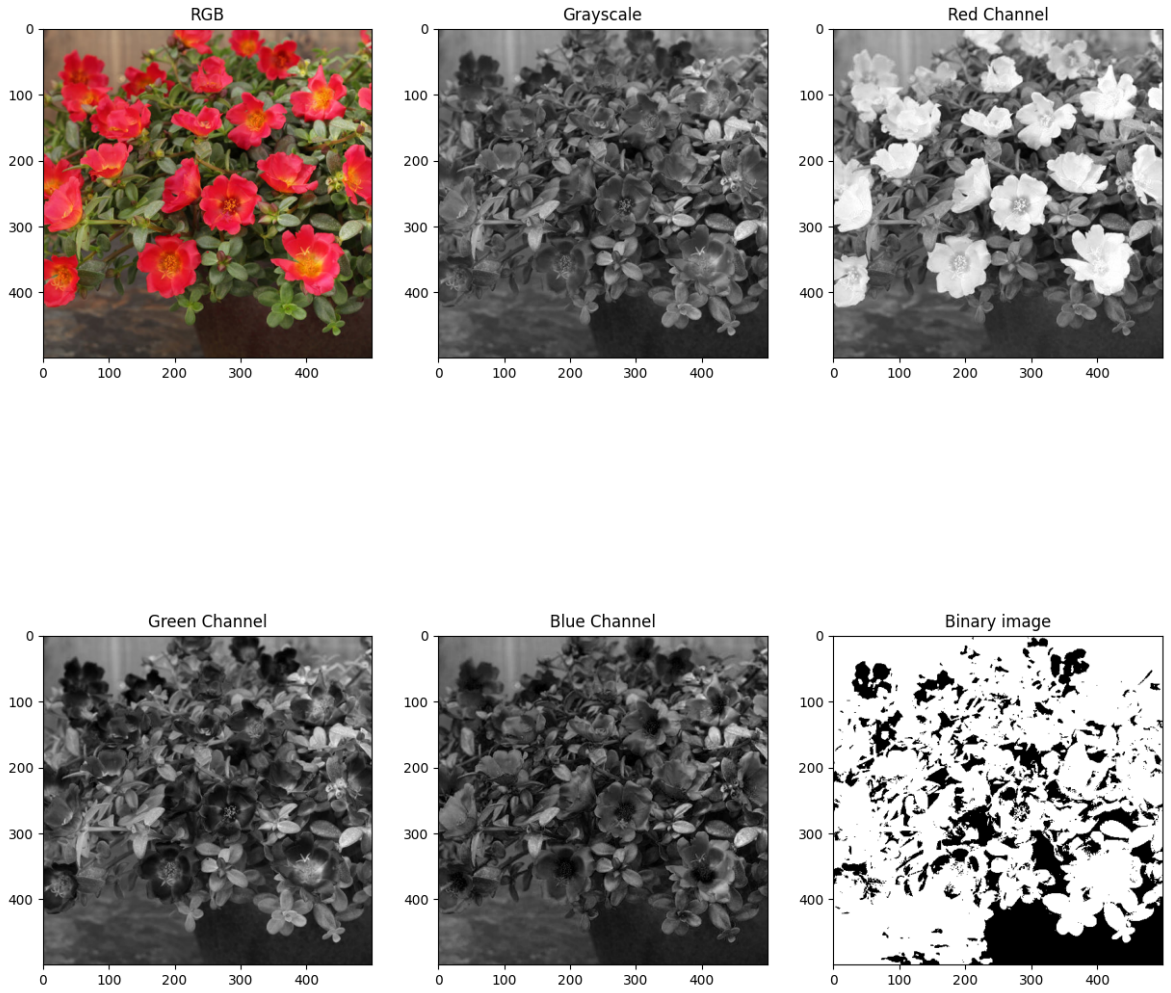
```
import matplotlib.pyplot as plt
import cv2

def main():
  img_path = "./portulica.jpg"
  img = plt.imread(img_path)

  # Grayscale image
  gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

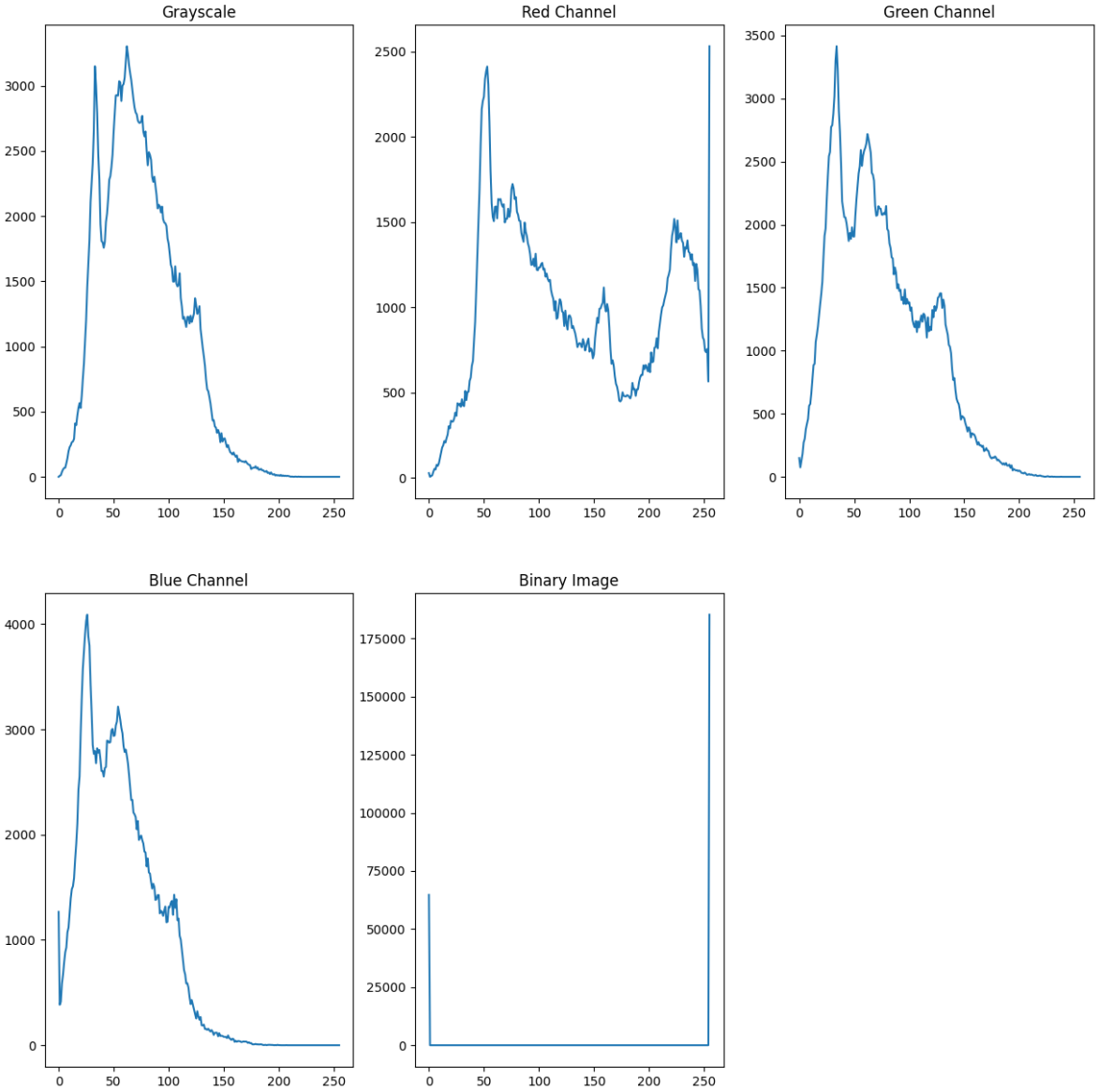
```
# Creating Histogram
gray_hist = cv2.calcHist([gray_img], [0], None, [256], [0, 256])


plt.figure(figsize=(15,15))

# Plotting Histogram
plt.subplot(2, 3, 1)
plt.title("Grayscale")
plt.plot(gray_hist)


#extract red channel
red_channel = img[:,:,0]

# Histogram
red_hist = cv2.calcHist([red_channel], [0], None, [256], [0, 256])

# Plotting Histogram
plt.subplot(2, 3, 2)
plt.title("Red Channel")
plt.plot(red_hist)


#extract green channel
green_channel = img[:,:,1]

# Creating Histogram
green_hist = cv2.calcHist([green_channel], [0], None, [256], [0, 256])

# Plotting Histogram
plt.subplot(2, 3, 3)
plt.title("Green Channel")
plt.plot(green_hist)


#extract blue channel
blue_channel = img[:,:,2]

# Creating Histogram
blue_hist = cv2.calcHist([blue_channel], [0], None, [256], [0, 256])

# Plotting Histogram
plt.subplot(2, 3, 4)
plt.title("Blue Channel")
plt.plot(blue_hist)

# Binary Image
```

```
  b_tuple = cv2.threshold(gray_img, 50, 255, cv2.THRESH_BINARY)
  binary = b_tuple[1]

  # Creating Histogram
  binary_hist = cv2.calcHist([binary], [0], None, [256], [0, 256])

  # Plotting Histogram
  plt.subplot(2, 3, 5)
  plt.title("Binary Image")
  plt.plot(binary_hist)

  # set the spacing between subplots
  #plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.5, hspace=

  plt.savefig("All Histograms")

  plt.show()
 # plt.clf()

  plt.figure(figsize=(15,15))


# section for images


  plt.subplot(2, 3, 1)
  plt.title("RGB")
  plt.imshow(img)

  plt.subplot(2, 3, 2)
  plt.title("Grayscale")
  plt.imshow(gray_img, cmap = 'gray')

  plt.subplot(2, 3, 3)
  plt.title("Red Channel")
  plt.imshow(red_channel, cmap = 'gray')

  plt.subplot(2, 3, 4)
  plt.title("Green Channel")
  plt.imshow(green_channel, cmap = 'gray')

  plt.subplot(2, 3, 5)
  plt.title("Blue Channel")
  plt.imshow(blue_channel, cmap = 'gray')

  plt.subplot(2, 3, 6)
  plt.title("Binary image")
  plt.imshow(binary, cmap = 'gray')
```

```
  plt.savefig("All Images")
  plt.show()

if __name__ == '__main__':
        main()
```

1.5 Observation





Using Pythons OpenCV module the RGB image is converted to grayscale image. Looking at the image and the histogram we can easily see most of the pixel intensity are on left side.As the source image has less amount of whitish color.
In Red channel, the red areas are brightened, so there are more high intensity value of red in the histogram. The other channel's histogram are also showing their corresponding gray levels.
Binary images are images which have only two values(which is clear from the histogram).We used thresh value 50.

2 Assignment 2

2.1 Problem statement

Perform 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 a 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:

1. $s = 100, if r >= T1 and r <= T2; otherwise s = 10$
2. $s = 100, if r >= T1 and r <= T2; otherwise s = r$
3. $s = clog(1 + r)$
4. $s = c(r + epsilon)\hat{p}$

2.2 Input Image

2.3 Method

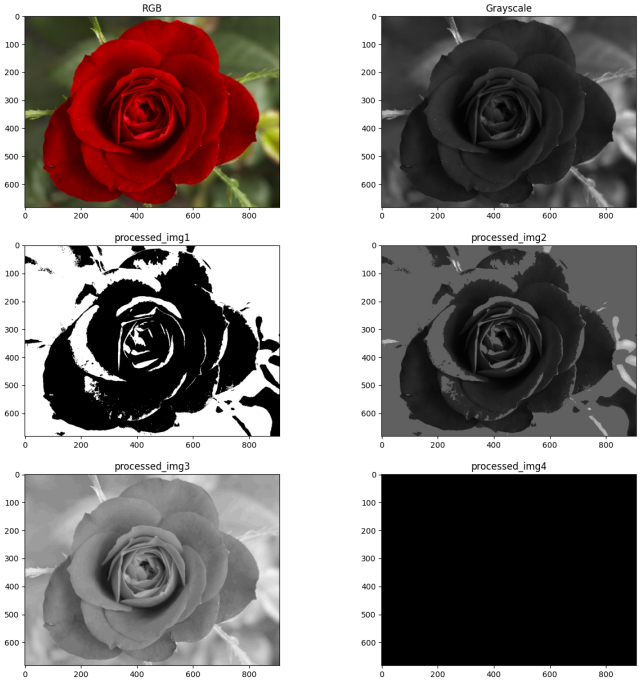
Point processing is defined as an operation which calculates the new value of a pixel in g(x, y) based on the value of the pixel in the same position in f(x, y) and some operation. That is, the values of a pixel's neighbors in f(x, y) have no effect whatsoever, hence the name point processing.
So firstly, we transformed the image into a grayscale image. Then we applied point processing by iterating all pixels.

2.4 Code

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


def plot_img(img_set, title_set):
    n = len(img_set)
    plt.figure(figsize = (15, 15))
    for i in range(n):
        img = img_set[i]
        ch = len(img.shape)

        plt.subplot( 3, 2, i + 1)
        if (ch == 3):
            plt.imshow(img_set[i])
        else:
            plt.imshow(img_set[i], cmap = 'gray')
        plt.title(title_set[i])
    plt.savefig('Output')
    plt.show()

def main():
    img_path = './a.jpg'
    rgb = plt.imread(img_path)
    print(rgb.shape)

    grayscale = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
    print(grayscale.shape)

    processed_img1 = np.copy(grayscale)
    processed_img2 = np.copy(grayscale)
    processed_img3 = np.copy(grayscale)
    processed_img4 = np.copy(grayscale)

    t1 = 50
    t2 = 160

    c = 10
    p = 40

    epsilon = 0.000001

    #    1st process
    row, column = grayscale.shape
    for x in range(row):
        for y in range(column):
```

```
            r = processed_img1[x, y]

            if r >= t1 and r <= t2:
                processed_img1[x, y] = 100
            else:
                processed_img1[x, y] = 10

    #    2nd process
    row, column = grayscale.shape
    for x in range(row):
        for y in range(column):

            r = processed_img2[x, y]

            if r >= t1 and r <= t2:
                processed_img2[x, y] = 100

    #    3rd process
    row, column = grayscale.shape
    for x in range(row):
        for y in range(column):

            r = processed_img3[x, y]
            s = c * np.log(1 + r)

            processed_img3[x, y] = s

    #    4th process
    row, column = grayscale.shape
    for x in range(row):
        for y in range(column):

            r = processed_img4[x, y]
            s = c * pow(( r + epsilon ), p)

            processed_img4[x, y] = s

    img_set = [rgb, grayscale, processed_img1, processed_img2, processed_img3, proces
    title_set = ['RGB', 'Grayscale', 'processed_img1', 'processed_img2', 'processed_i
    plot_img(img_set, title_set)

if __name__ == '__main__':
        main()
```

2.5 Observation



We used T1 = 50 and T2 = 160.

Equation 1: The image becomes more blackish. Here s = 10.

Equation 2: The image looks like a low contrast image.

Equation 3: The image is more whitish as logarithmic equation expand the pixel intensity range is wider.

Equation 4: The image is completely black. As, we assign p = 50.

3 Assignment 3

3.1 Problem statement

Show the effect of 6 different kinds of kernels on a 2D image.

3.2 Input Image

3.3 Method

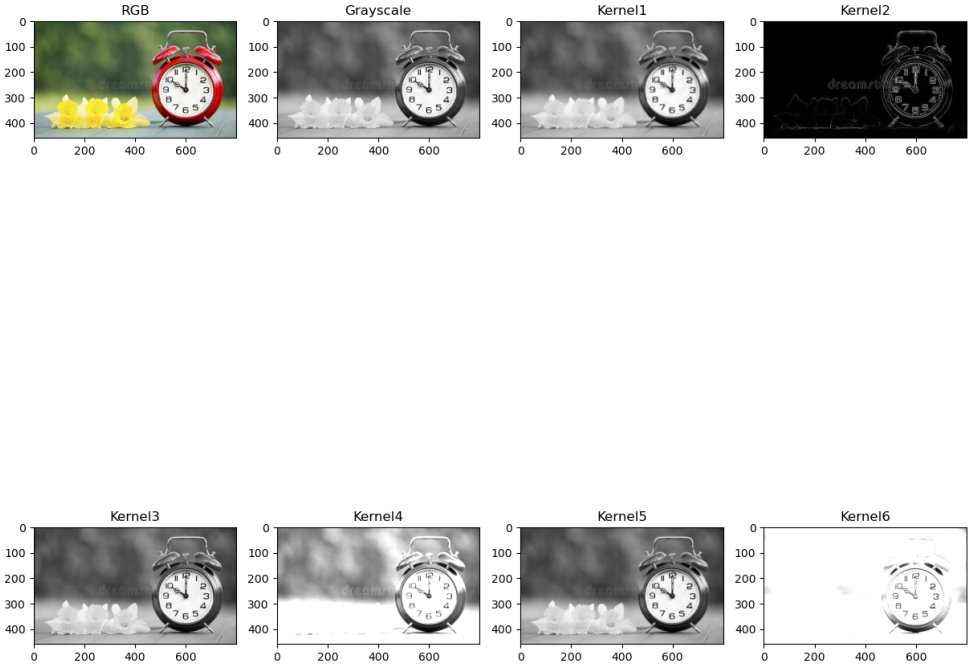
After reading the image and converting it to grayscale, we defined the kernels. Then we used the built-in cv2.filter2D function of OpenCV Module to perform the filtering AKA correlation.

3.4 Code

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

def main():
        path = './clock.jpg'

        rgb = plt.imread(path)
        print(rgb.shape)

        gray = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
        print(gray.shape)

        kernel_1 = np.ones((5, 5), np.float32)/30
        kernel_2 = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
        kernel_3 = np.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]])
        kernel_4 = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])/5
```

```
        kernel_5 = np.ones((3, 3), dtype = np.float32) / 9
        kernel_6 = np.array([[0, 5, 0], [1, -2, 1], [-1, 1, -1]])

        print('Kernel 1: ' + format(kernel_1))
        print('Kernel 2: ' + format(kernel_2))
        print('Kernel 3: ' + format(kernel_3))
        print('Kernel 4: ' + format(kernel_4))
        print('Kernel 5: ' + format(kernel_5))
        print('Kernel 6: ' + format(kernel_6))

        processed_img1 = cv2.filter2D(gray, -1, kernel_1)
        processed_img2 = cv2.filter2D(gray, -1, kernel_2)
        processed_img3 = cv2.filter2D(gray, -1, kernel_3)
        processed_img4 = cv2.filter2D(gray, -1, kernel_4)
        processed_img5 = cv2.filter2D(gray, -1, kernel_5)
        processed_img6 = cv2.filter2D(gray, -1, kernel_6)

    #plot

        img_set = [rgb, gray, processed_img1, processed_img2, processed_img3, process
        title_set = ['RGB', 'Grayscale', 'Kernel1', 'Kernel2', 'Kernel3', 'Kernel4',
        PlotImg(img_set, title_set)


def PlotImg(img_set, title_set):
    n = len(img_set)
    plt.figure(figsize=(15, 15))
    for i in range(n):
        img = img_set[i]
        channel = len(img.shape)

        plt.subplot(2, 4, i+1)

        if(channel == 3):
            plt.imshow(img)
        else:
            plt.imshow(img, cmap = 'gray')

        plt.title(title_set[i])
    #plt.show()
    plt.savefig("outputs")

if __name__ == '__main__':
    main()
```

3.5 Observation



Our first kernel is 5/30. almost average filter. which smoothened the image. The 2nd filter was edge detector. The 3rd filter is nothing but the image preserving filter. 4th, one highlighted the image a bit. And the last filter is avg filter which also smoothened the image.

4 Assignment 4

4.1 Problem statement

1. Implement histogram and compare the result with built-in histogram function.

2. Implement neighborhood processing and compare the result with built-in cv2.filter2D

4.2 Input Image

4.3 Method

Image histogram is nothing but a graphical representation of the gray level distribution in a digital image
Here, we stored frequencies of gray levels in a list, which indicated our histogram. And also used built in method.

And for neighborhood processing, we padded our image with zero. Then we took a slice from the image about kernel size and performed a correlation operation.

4.4 Code

4.4.1 Histogram Compare

```
from typing import Counter
import matplotlib.pyplot as plt
import numpy as np
import cv2

a = plt.imread('./clk.jpg')
```

```
g = cv2.cvtColor(a, cv2.COLOR_RGB2GRAY)

''' Built in function '''

hist_b, bins = np.histogram(g, 256)

''' User defined function '''

hist_u = []
for i in range(256):
    hist_u += [np.count_nonzero(g == i)]

# printing histogram
print('Built in result: ')
#print (hist_b)
print()

print('Built in result: ')
#print (hist_u)
print()

''' plot '''
plt.figure(figsize=(15,15))

plt.subplot(1, 2, 1)
plt.title('Built in function')
plt.plot(hist_b)

plt.subplot(1, 2, 2)
plt.title('User defined function')
plt.plot(hist_u)

#plt.show()
plt.savefig("hist out")
```

4.4.2 Neighborhood processing Compare

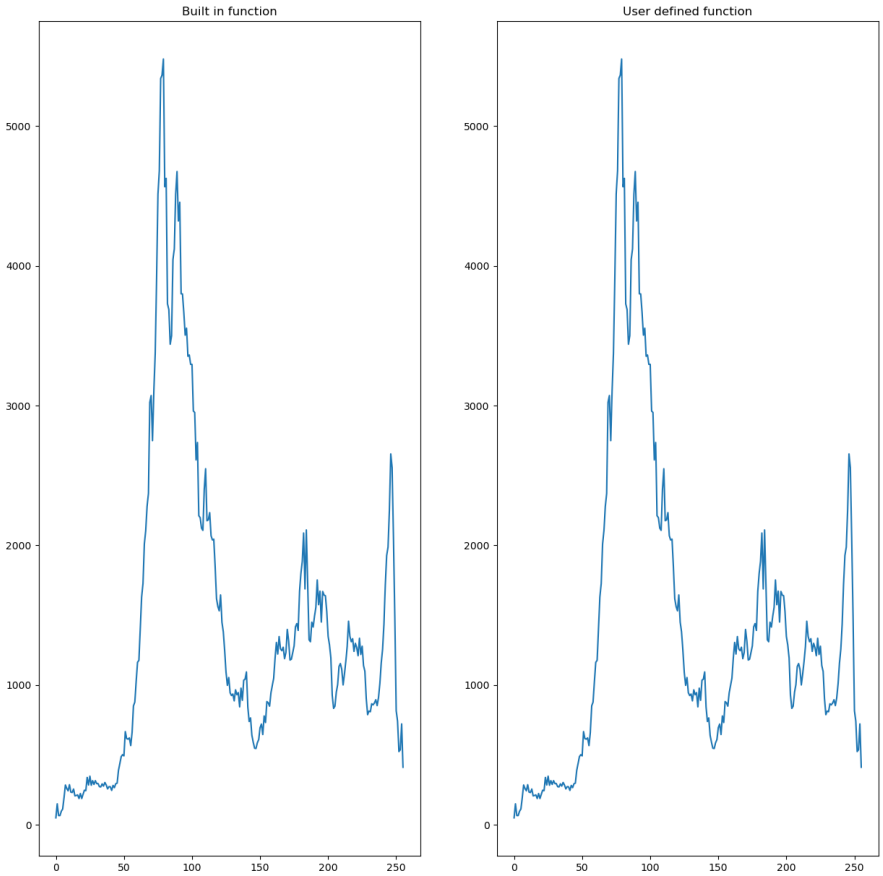
```
import numpy as np
#import sys
#np.set_printoptions(threshold=sys.maxsize)
import matplotlib.pyplot as plt
import cv2

a = plt.imread('./clk.jpg')
g = cv2.cvtColor(a, cv2.COLOR_RGB2GRAY)

g_width = g.shape[0]
```

```
g_height = g.shape[1]

''' set kernel and padding image'''

#kernel1 = np.ones((5, 5), dtype = np.float32)/20
kernel1 = np.array([[-1, -1, -1],[-1, 8, -1],[-1, -1, -1]])
#kernel1 = np.array([[-1, -1, -1, -1, -1],[-1, -1, 8, -1, -1],[-1, -1, 8, -1, -1],[

pad_quantity = int((len(kernel1) - 1) / 2)

padded_g = pad_arr = np.pad(g, pad_quantity)

#print(padded_g)
#padded_g = cv2.copyMakeBorder(src=g, top=1, bottom=1, left=1, right=1, borderType=

''' Built in func '''

processed_img1 = cv2.filter2D(padded_g, -1, kernel1, cv2.BORDER_CONSTANT)
processed_img1 = processed_img1[pad_quantity:g_width+pad_quantity, pad_quantity:g_hei

print(padded_g[0:5,padded_g.shape[1]-5:]) #cropped padding
print(processed_img1) #cropped padding

''' User defined func '''

m, n = kernel1.shape

y, x = padded_g.shape

processed_img2 = np.zeros((g_width, g_height))

for i in range(g_width):
    for j in range(g_height):
        processed_img2[i][j] = int(np.sum(padded_g[i:i+m, j:j+n]*kernel1))

        if(processed_img2[i][j] > 255):
            processed_img2[i][j] = 255
        if(processed_img2[i][j] < 0):
            processed_img2[i][j] = 0

print(processed_img2)

''' plot '''
plt.figure(figsize=(15,15))

plt.subplot(1, 2, 1)
plt.title('Built in function')
plt.imshow(processed_img1, cmap = 'gray')
```

```
plt.subplot(1, 2, 2)
plt.title('User defined function')
plt.imshow(processed_img2, cmap = 'gray')

#plt.show()
plt.savefig("conv out")
```

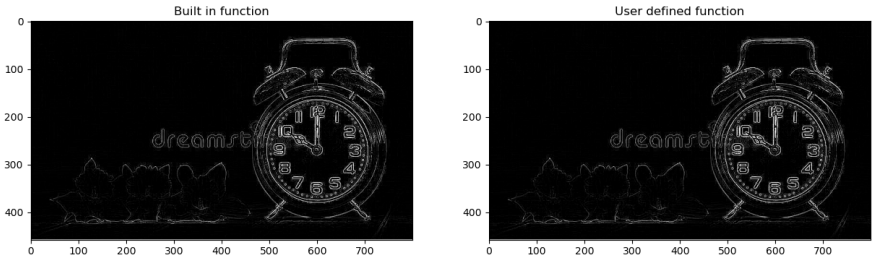
4.5 Observation

4.5.1 Histogram



Here we can see both shows the same output.

4.5.2 Filter



Here we can see both shows the same output.

5 Assignment 5

5.1 Problem statement

1. Show what happens when we apply a binary mask on a grayscale image.
2. Slice an 8-bit grayscale image into 8 planes.
3. Show the effect of convolution of a grayscale image with a Laplacian filters and Sobel filters.

5.2 Input Image

5.3 Method

Bit masking: A binary mask defines a region of interest (ROI) of an image. Mask pixel values of 1 indicate image pixels that belong to the ROI. Mask pixel values of 0 indicate image pixels that are part of the background. Here we took the mask and performed bitwise operation with the image

Filtering: We used built in Laplacian filter and Sobel filter using cv2.filter() function. Laplacian and Sobel filter both are edge detectors

Bit slicing: Bit plane slicing is a method of representing an image with one or more bits of the byte used for each pixel.
Here we converted each pixels into binary then look 1 bit from each pixel and made one bit plane. Thus we traversed from LSB to MSB. Which left us with 8 bit planes.

5.4 Code

5.4.1 Binary Mask

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

gray = cv2.imread('s.jpg', 0)

mask = np.zeros(gray.shape, dtype=np.uint8)

mask = cv2.circle(mask, (260, 300), 255, (255,255,255), -1)

result = cv2.bitwise_and(gray, mask)

plt.figure(figsize = (10, 10))

plt.subplot(1,3,1)
plt.title("Grayscale")
plt.imshow(gray, cmap = 'gray')

plt.subplot(1,3,2)
plt.title("Mask")
plt.imshow(mask, cmap = 'gray')

plt.subplot(1,3,3)
plt.title("Masked result")
plt.imshow(result, cmap = 'gray')

plt.savefig('Masking')
#plt.show()
```

5.4.2 Filters

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

gray = cv2.imread('s.jpg', 0)

laplacian_kernel = np.array([[0,-1,0],[-1,4,-1],[0,-1,0]])
sobel_kernel = np.array([[-1,0, 1],[-2,0,2],[-1,0, 1]])

processed_img1 = cv2.filter2D(gray, -1, laplacian_kernel)
processed_img2 = cv2.filter2D(gray, -1, sobel_kernel)

plt.figure(figsize = (10, 10))
```

```
plt.subplot(1,3,1)
plt.title("Grayscale")
plt.imshow(gray, cmap = 'gray')

plt.subplot(1,3,2)
plt.title("laplacian filter")
plt.imshow(processed_img1, cmap = 'gray')

plt.subplot(1,3,3)
plt.title("Sobel filter")
plt.imshow(processed_img2, cmap = 'gray')

plt.savefig('Filter')
#plt.show()
```

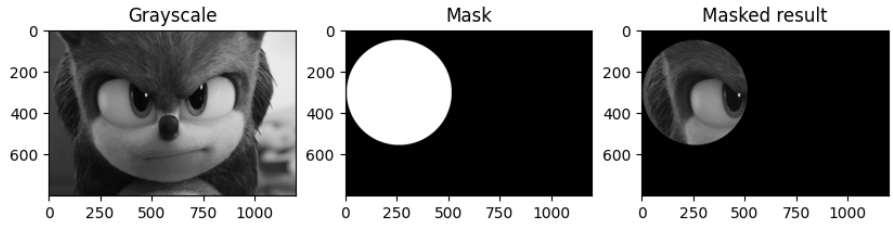
5.4.3 Bit Slicing

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

# Reading the image in grayscale
img_path = "./s.jpg"
img = cv2.imread(img_path, 0)

plt.figure(figsize = (10, 10))

plt.subplot(3, 3, 1)
plt.imshow(img, cmap = 'gray')
plt.title("Grayscale")

# Change each pixel to it's corresponding binary value
sliced_image = np.zeros([img.shape[0], img.shape[1]], dtype = np.uint8)

for k in range(8):
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            bin = np.binary_repr(img[i][j], width = 8)
            sliced_image[i][j] = int(bin[7-k])

    # Plotting Images
    plt.subplot(3, 3, k+2)
    plt.imshow(sliced_image, cmap = 'gray')
    plt.title("Bit {} image".format(k))

plt.savefig('Bit Slicing')
```

```
plt.show()
```

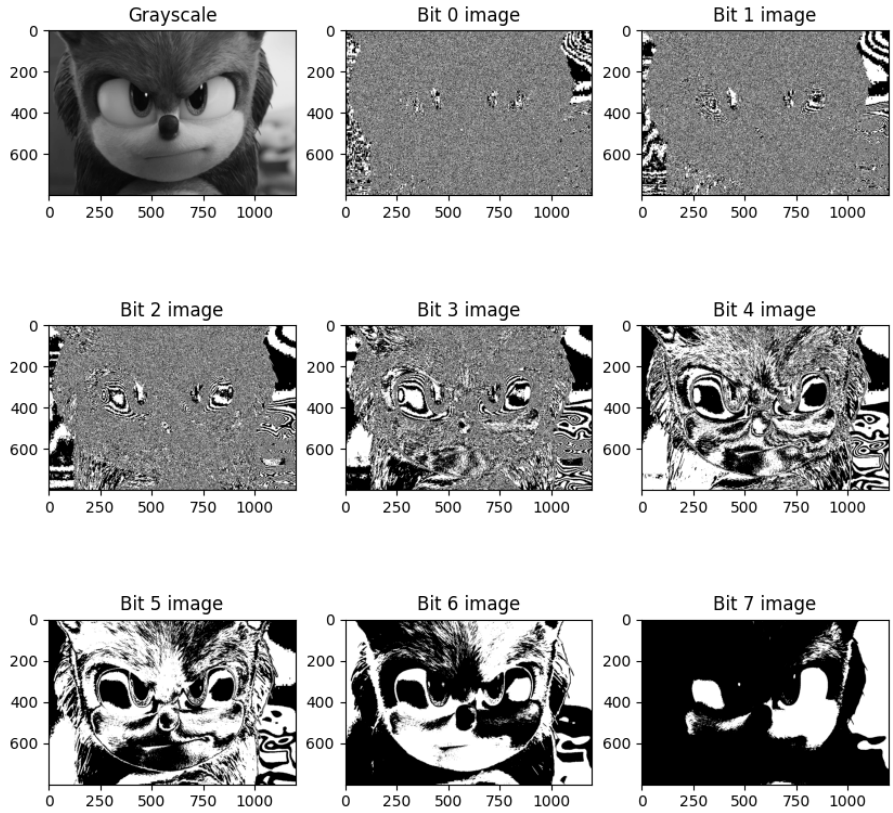
5.5 Observation

5.5.1 Binary Mask



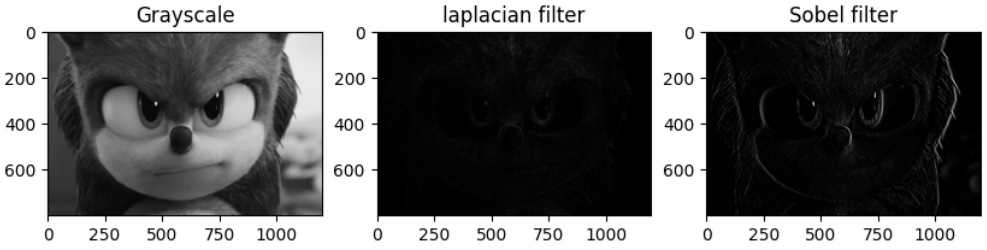
The output shows only the image inside ROI(Region of interest).

5.5.2 Bit Slicing



We can see for the most of the information of the image is on bit-7 plane.That's why bit-7 plane is more detailed than others. This is the MSB plane.
Similarly, We can also see that LSB plane has very less information.

5.5.3 Filters



Laplacian filter uses for edge detection which generally compute the second derivative of the images.

Sobel filters are also uses for edge detection.But Sobel uses horizontal and vertical kernels which is 1st derivative based.

6 Assignment 6

6.1 Problem statement

Add salt and pepper noise with an image and see the effect of average kernel, Gaussian kernel and median filter on the noisy image.

6.2 Input Image

6.3 Method

We iterated the image several times, took random pixels from the image and added 1 for salt and 0 for paper noise.

We applied average filter, Gaussian filter and median filter using cv2.filter() built is method.

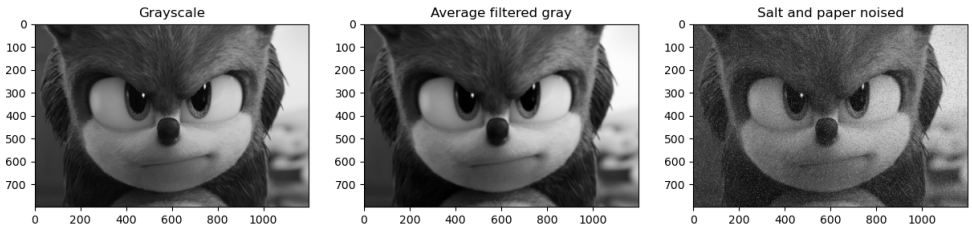
6.4 Code

```
from turtle import color
import matplotlib.pyplot as plt
import cv2
import numpy as np
import random

def add_noise(img, min_pixel_range, max_pixel_range, noise_color):
    row , col = img.shape
```

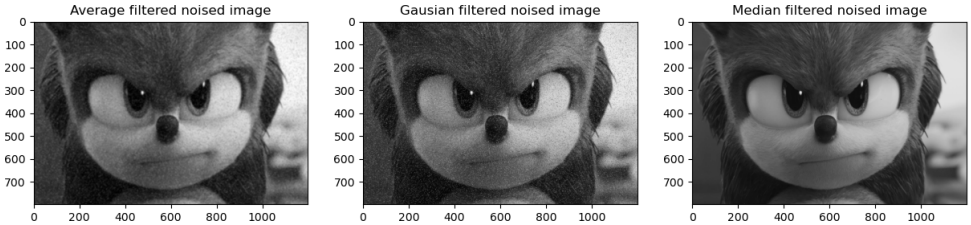
```
    number_of_pixels = random.randint(min_pixel_range, max_pixel_range)

    for i in range(number_of_pixels):
        y=random.randint(0, col - 1)

        x=random.randint(0, row - 1)

        img[x][y] = noise_color

    return img

gray = cv2.imread("./s.jpg", 0)

tmp = np.copy(gray)

white_noised_image = add_noise(tmp, 5, 100000, 255)

both_noised_image = add_noise(white_noised_image, 5, 100000, 0)

average_kernel = np.ones((7,7))/49
gaussian_kernel = np.array(([1,2,1], [2,4,2], [1,2,1]))/16

average_kernel_image_gray = cv2.filter2D(gray, -1, average_kernel)
average_kernel_imag_noised = cv2.filter2D(both_noised_image, -1, average_kernel)
gaussian_kernel_image = cv2.filter2D(both_noised_image, -1, gaussian_kernel)
median_kernel_image = median = cv2.medianBlur(both_noised_image,3)

plt.figure(figsize = (10, 10))

plt.subplot(1,3,1)
plt.title("Grayscale")
plt.imshow(gray, cmap = "gray")

plt.subplot(1,3,2)
plt.title("Average filtered gray")
plt.imshow(average_kernel_image_gray, cmap = "gray")

plt.subplot(1,3,3)
plt.title("Salt and paper noised")
plt.imshow(both_noised_image, cmap = "gray")
plt.savefig("sp1")
plt.subplot(1,3,1)
plt.title("Average filtered noised image")
plt.imshow(average_kernel_imag_noised, cmap = "gray")

plt.subplot(1,3,2)
plt.title("Gausian filtered noised image")
```

```
plt.imshow(gaussian_kernel_image, cmap = "gray")

plt.subplot(1,3,3)
plt.title("Median filtered noised image")
plt.imshow(median_kernel_image, cmap = "gray")

plt.savefig("Salt and paper")
#plt.show()
plt.savefig("sp2")
```

6.5 Observation



Here we can see grayscal image, averege filterd image and noised image.



Here we can see images after applying filters on the noised image.
The averaging filter and Gaussian filter is not optimum in the case of salt and pepper noise removing. Because salt and pepper noises are impulse noises. Median filter is the best solution in the case of salt and pepper noise.

7 Assignment 7

7.1 Problem statement

Move intensity of a grayscale image left, right and into a specific range and check its effect on histogram. An example output is attached.

7.2 Input Image

7.3 Method

Intensity transformations are applied on images for contrast manipulation or image thresholding. Here we took a threshold value and shifted the pixel values based on the threshold.

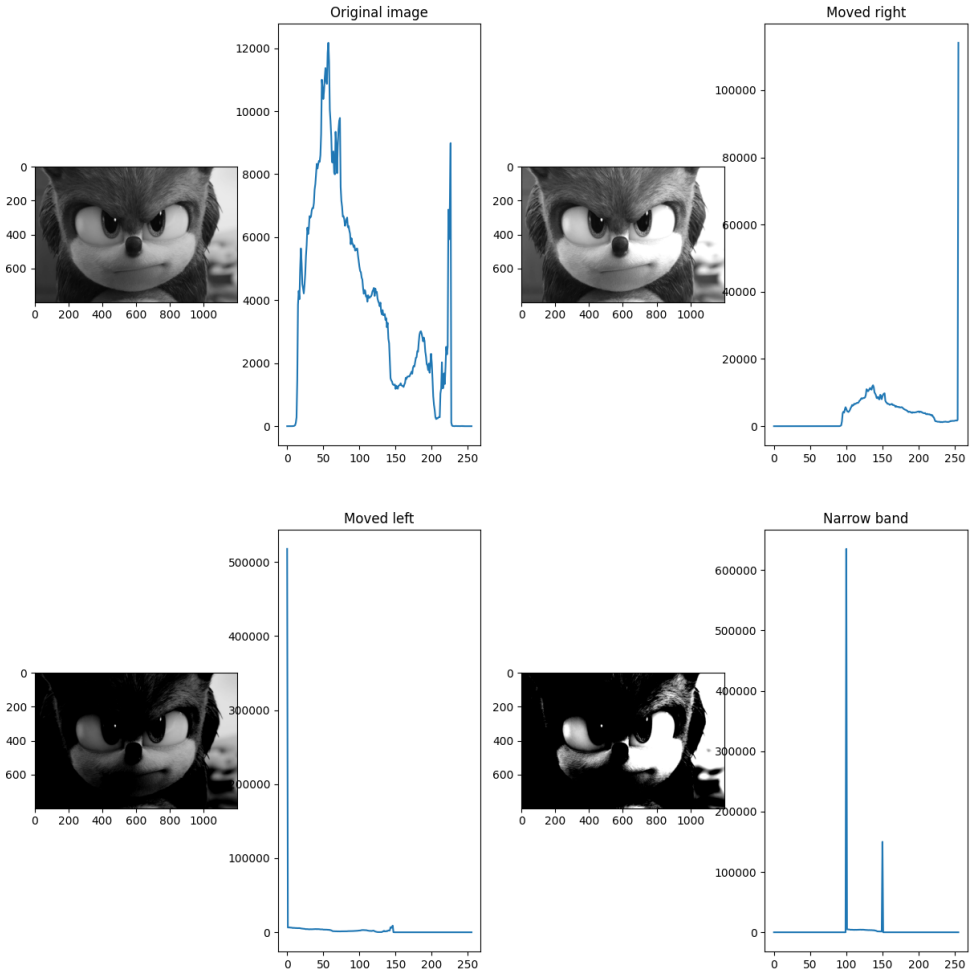
7.4 Code

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

def plot_image_and_histogram(img_set):
    subplot_row = 2
    subplot_column = 4
    j = 1

    plt.figure(figsize = (10, 10))
```

```
    for i in img_set:
        plt.subplot(subplot_row, subplot_column, j)
        plt.imshow(img_set[i][0], cmap = 'gray')

        j += 1

        plt.subplot(subplot_row, subplot_column, j)
        plt.title(i)
        plt.plot(img_set[i][1])

        j += 1

    plt.savefig("Contrast stretching")
    plt.show()

gray = cv2.imread("./s.jpg", 0)

left_intensity = np.copy(gray)
right_intensity = np.copy(gray)
narrow_band_intensity = np.copy(gray)

for i in range(gray.shape[0]):
    for j in range(gray.shape[1]):
        #move left
        if(left_intensity[i][j]  - 80 < 0):
            left_intensity[i][j] = 0
        else:
            left_intensity[i][j] = left_intensity[i][j] - 80

        #move right
        if(right_intensity[i][j]  + 80 > 255):
            right_intensity[i][j] = 255
        else:
            right_intensity[i][j] = right_intensity[i][j] + 80

        #narrow band
        if(narrow_band_intensity[i][j] < 100):
            narrow_band_intensity[i][j] = 100
        elif(narrow_band_intensity[i][j] > 150):
            narrow_band_intensity[i][j] = 150

hist1 = cv2.calcHist([gray], [0], None, [256], [0, 256])
hist2 = cv2.calcHist([right_intensity], [0], None, [256], [0, 256])
hist3 = cv2.calcHist([left_intensity], [0], None, [256], [0, 256])
hist4 = cv2.calcHist([narrow_band_intensity], [0], None, [256], [0, 256])

img_set =   {   "Original image":[gray, hist1],
```

```
            "Moved right":[right_intensity, hist2],
            "Moved left":[left_intensity, hist3],
            "Narrow band":[narrow_band_intensity, hist4]
        }

plot_image_and_histogram(img_set)
```

7.5 Observation



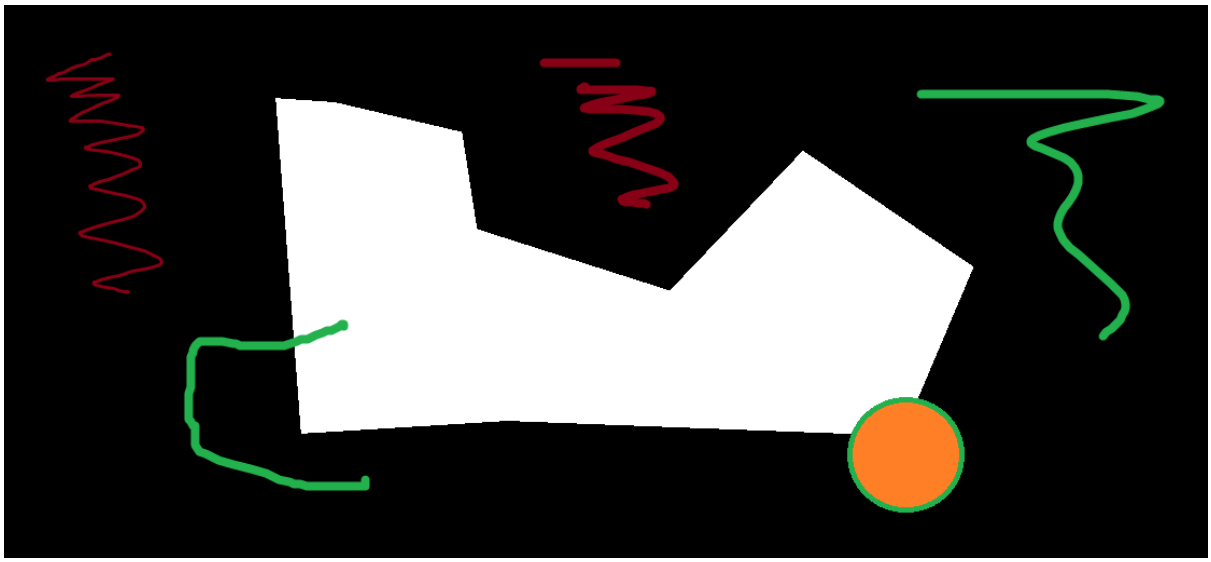
We can see after shifting from left by 80 the image became more brighten and after shifting from right by 80 the image became darker.
And when we shifted in range of 100 to 150 the image became a low-contrast image as all intensity is shifted in the middle.

8 Assignment 8

8.1 Problem statement

Explain what happens when you apply morphological dilation, erosion, opening, and closing operations using at least three kinds of structuring elements on your favorite binary image.

8.2 Input Image



8.3 Method

Morphological operations apply a structuring element to an input image, creating an output image of the same size. There are several operations. They are:

1. Dilation: The basic idea of dilation is it dilates the boundaries of the foreground object. The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if at least one of the pixels under the kernel is 1.

   So the thickness or size of the foreground object increases, or it increases, the white region in the image or size of the foreground object increases.

2. Erosion: The basic idea of erosion is it erodes away the boundaries of the foreground object. The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

   So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises.

3. Opening: Opening is just another name for erosion followed by dilation. It is useful in removing noise.

4. Closing: Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object.

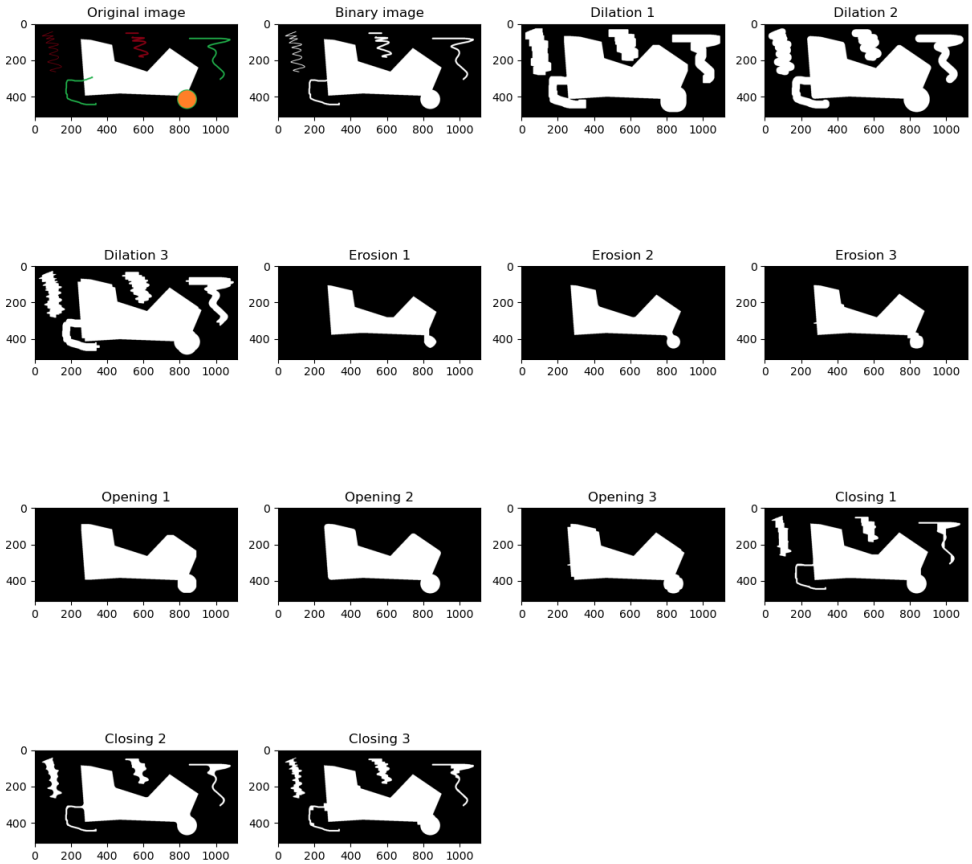
We used built-in function of OpenCV, getStructuringElement() and performed the necessary morphological operations using the necessary functions, dilate() for Dilation, erode() for Erosion, and morphologyEx() for Opening and Closing.

8.4 Code

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

def plot_img(img_set, title_set):
    n = len(img_set)
    plt.figure(figsize = (10, 10))
    for i in range(n):
        img = img_set[i]
        ch = len(img.shape)

        plt.subplot(3, 5, i + 1)
        plt.imshow(img_set[i], cmap = 'gray')
        plt.title(title_set[i])

    plt.savefig("morphological transformation")

img_path = 'image.jpg'
img = cv2.imread(img_path, 0)

_,binary_img = cv2.threshold(img, 127, 1, cv2.THRESH_BINARY)

kernel1 = cv2.getStructuringElement(cv2.MORPH_RECT,(9,9))
kernel2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(9,9))
kernel3 = cv2.getStructuringElement(cv2.MORPH_CROSS,(9,9))

dilation1 = cv2.dilate(binary_img,kernel1)
dilation2 = cv2.dilate(binary_img,kernel2)
dilation3 = cv2.dilate(binary_img,kernel3)

erosion1 = cv2.erode(binary_img,kernel1)
erosion2 = cv2.erode(binary_img,kernel2)
erosion3 = cv2.erode(binary_img,kernel3)

opening1 = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel1)
opening2 = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel2)
opening3 = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel3)

closing1 = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, kernel1)
```

```
closing2 = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, kernel2)
closing3 = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, kernel3)

'''        Plot images. '''
img_set = [binary_img, dilation1, dilation2, dilation3,
erosion1, erosion2, erosion3, opening1, opening2, opening3, closing1, closing2, closi

title_set = ['Binary', 'dilation1', 'dilation2', 'dilation3',
'erosion1', 'erosion2', 'erosion3', 'opening1', 'opening2', 'opening3', 'closing1','c

plot_img(img_set, title_set)
```

8.5 Observation



when DIlation is applied, the boundary became thicker than the original image and in erosion it becomes thinner. In opening which is erosion followed by dilation, the image

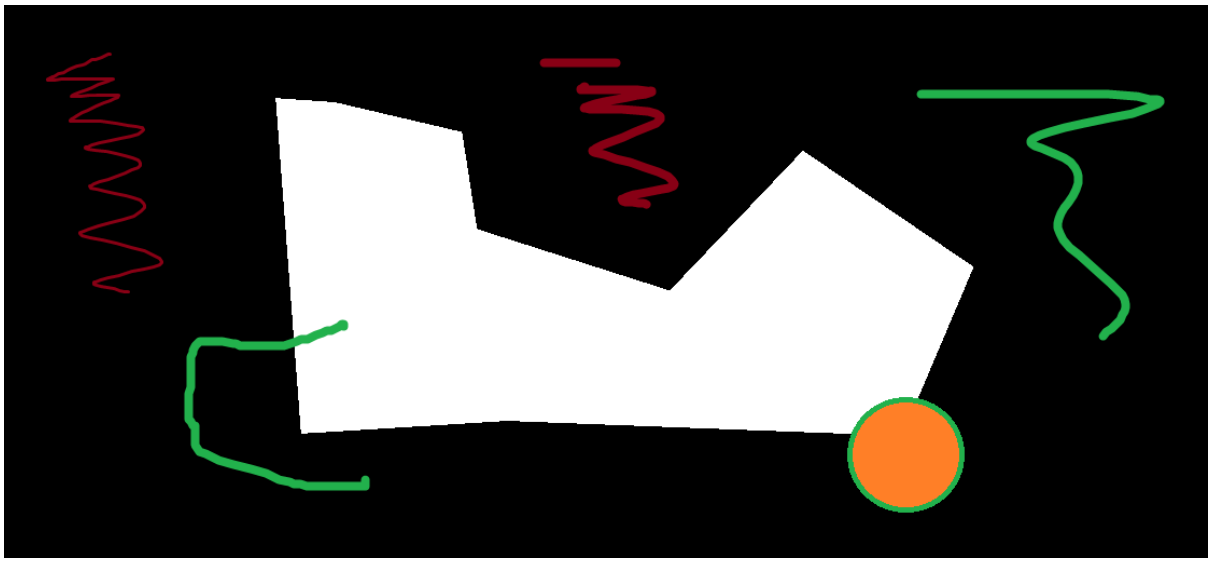
became thinner in foreground.Some portion of the image is erased due to erosion process. In closing the image expanded at first then became thin.

9 Assignment 9

9.1 Problem statement

Implement functions for erosion, dilation, opening and closing operations by Python. Then compare the output of your implemented functions with the output of the OpenCV's built-in functions.

9.2 Input Image



9.3 Method

9.4 Code

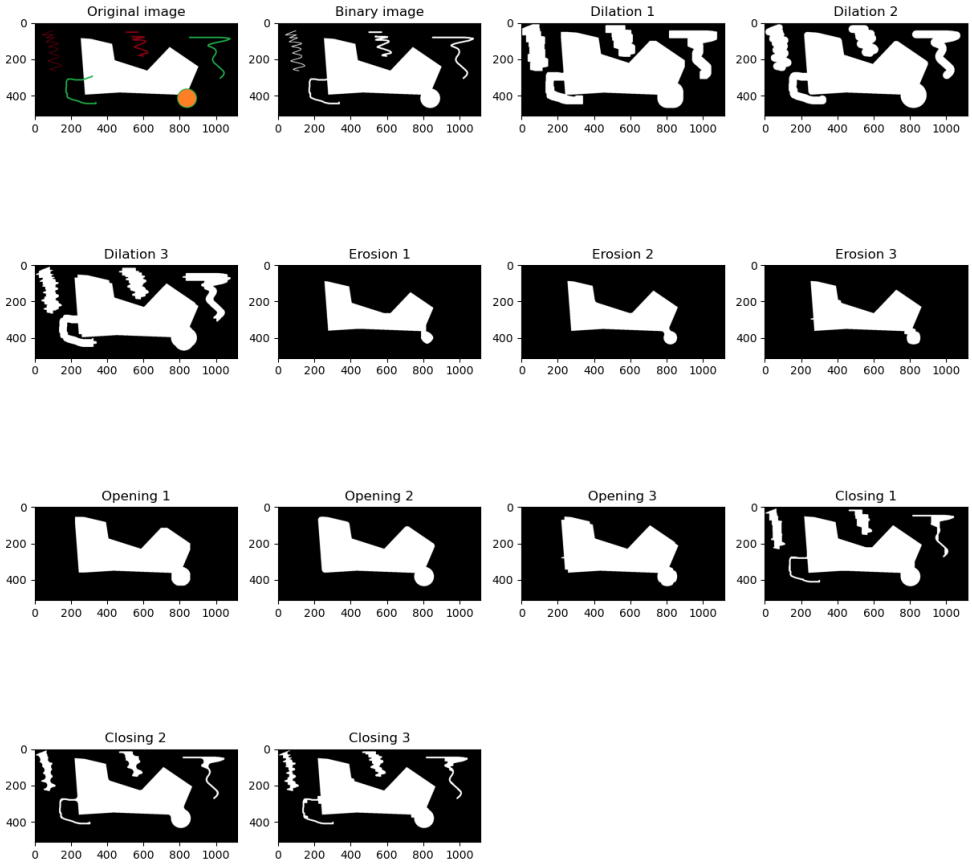
```
import matplotlib.pyplot as plt
import cv2
from numpy import dtype, uint, uint8
import numpy as np

def plot_image(img_set):
    subplot_row = 4
    subplot_column = 4
    j = 1

    plt.figure(figsize = (15, 15))

    for i in img_set:
        plt.subplot(subplot_row, subplot_column, j)

        if(img_set[i][1] != ""):
            plt.imshow(img_set[i][0], cmap = img_set[i][1])
        else:
            plt.imshow(img_set[i][0])
```

```
        plt.title(i)
        j += 1

    plt.savefig("Morphological analysis")
    plt.show()

def my_erosion(img, kernel):
    m, n = kernel.shape
    row, col = img.shape
    res = np.zeros((row, col))

    for i in range(row-m):
        for j in range(col-n):
            tmp = img[i:i+m, j:j+n] * kernel

            if(tmp == kernel).all():
                res[i, j] = 1
            else:
                res[i, j] = 0

    return res

def my_dilation(img, kernel):
    m, n = kernel.shape
    row, col = img.shape
    tmp = np.zeros((row, col))

    for i in range(row-m):
        for j in range(col-n):
            tmp[i, j] = np.max(img[i:i+m, j:j+n] * kernel)
    return tmp

def my_opening(img, kernel):
    erode = my_erosion(img, kernel)
    res = my_dilation(erode, kernel)
    return res

def my_closing(img, kernel):
    dilate = my_dilation(img, kernel)
    res = my_erosion(dilate, kernel)
    return res

def main():
    img = plt.imread('./s.png')

    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    _, bin_img = cv2.threshold(gray_img, 0, 1, cv2.THRESH_BINARY)

    kernel_1 = cv2.getStructuringElement(cv2.MORPH_RECT,(35,35))
    kernel_2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(35,35))
    kernel_3 = cv2.getStructuringElement(cv2.MORPH_CROSS,(35,35))

    dilation_1 = my_dilation(bin_img,kernel_1)
    dilation_2 = my_dilation(bin_img,kernel_2)
    dilation_3 = my_dilation(bin_img,kernel_3)

    erosion_1 = my_erosion(bin_img, kernel_1)
    erosion_2 = my_erosion(bin_img, kernel_2)
    erosion_3 = my_erosion(bin_img, kernel_3)

    opening_1 = my_opening(bin_img, kernel_1)
    opening_2 = my_opening(bin_img, kernel_2)
    opening_3 = my_opening(bin_img, kernel_3)

    closing_1 = my_closing(bin_img, kernel_1)
    closing_2 = my_closing(bin_img, kernel_2)
    closing_3 = my_closing(bin_img, kernel_3)


    img_set =   {
                "Original image":[img, ""],

                "Binary image":[bin_img, "gray"],

                "Dilation 1":[dilation_1, "gray"],
                "Dilation 2":[dilation_2, "gray"],
                "Dilation 3":[dilation_3, "gray"],

                "Erosion 1":[erosion_1, "gray"],
                "Erosion 2":[erosion_2, "gray"],
                "Erosion 3":[erosion_3, "gray"],

                "Opening 1":[opening_1, "gray"],
                "Opening 2":[opening_2, "gray"],
                "Opening 3":[opening_3, "gray"],

                "Closing 1":[closing_1, "gray"],
                "Closing 2":[closing_2, "gray"],
                "Closing 3":[closing_3, "gray"],
                }

    plot_image(img_set)

if __name__ == '__main__':
```

```
main()
```

9.5 Observation



Here we can see the output of built in function and user defined function produced same result

10 Assignment 10

10.1 Problem statement

Check what happens when you apply histogram equalization on your favorite grayscale image.

10.2 Input Image

10.3 Method

Histogram equalization is used to enhance contrast. This method usually increases the global contrast of many images, especially when the image is represented by a narrow range of intensity values. Through this adjustment, the intensities can be better distributed on the histogram, utilizing the full range of intensities evenly. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the highly populated intensity values, which are used to degrade image contrast.

OpenCV has built-in function "equalizeHist" for histogram equalization and "calcHist" for generating histogram of the image.

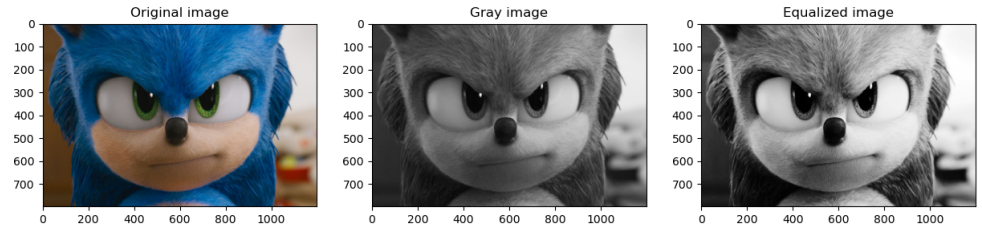
```
import matplotlib.pyplot as plt
import cv2
from numpy import dtype, uint, uint8
import numpy as np

def plot_image(img_set):
```

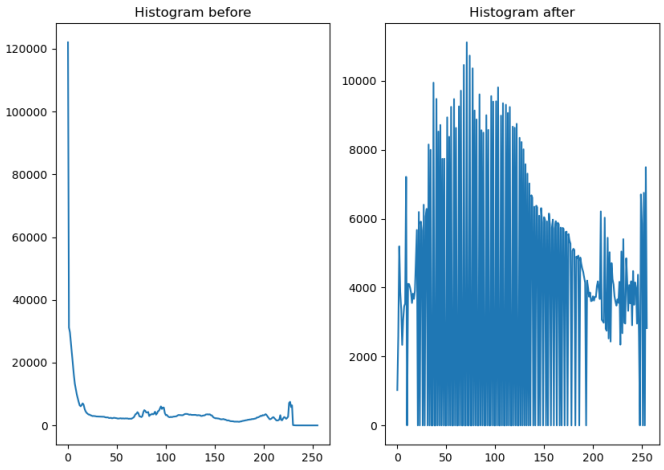
```
    subplot_row = 2
    subplot_column = 3
    j = 1

    plt.figure(figsize = (15, 15))

    for i in img_set:
        plt.subplot(subplot_row, subplot_column, j)

        if(img_set[i][1] == "hist"):
            plt.plot(img_set[i][0])
        elif(img_set[i][1] == "gray"):
            plt.imshow(img_set[i][0], cmap = img_set[i][1])
        else:
            plt.imshow(img_set[i][0])

        plt.title(i)
        j += 1

    plt.savefig("Histogram equalization")
    plt.show()



def main():
    img = plt.imread('./s.JPG')

    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    img1=np.uint8(cv2.normalize(gray_img, None, 0, 255, cv2.NORM_MINMAX))

    equ = cv2.equalizeHist(img1)

    hist_before = cv2.calcHist([img],[0],None,[256],[0,256])
    hist_after = cv2.calcHist([equ],[0],None,[256],[0,256])

    img_set =   {
                "Original image":[img, ""],

                "Gray image":[gray_img, "gray"],

                "Equalized image":[equ, "gray"],

                "Histogram before":[hist_before, "hist"],

                "Histogram after":[hist_after, "hist"],
                }
```

```
    plot_image(img_set)

if __name__ == '__main__':
    main()
```

10.4 Observation

Below we can see the difference between non equalized histogram vs equalized histogram





11 Assignment 11

11.1 Problem statement

Explain what happens when you apply filtering in frequency domain. Try at least 4 different filters.

11.2 Input Image

11.3 Method

The Fast Fourier Transform (FFT) is commonly used to transform an image between the spatial and frequency domain. Unlike other domains such as Hough and Radon, the FFT method preserves all original data. Plus, FFT fully transforms images into the frequency domain, unlike time-frequency or wavelet transforms.
Here we used numpy's fft built in methods.

11.4 Code

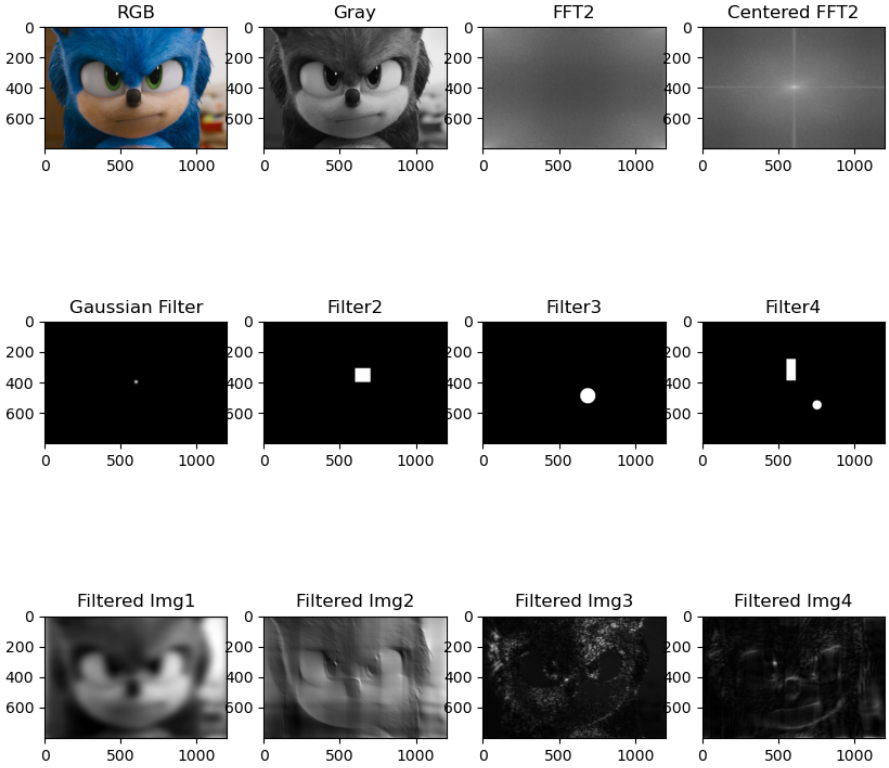
```
import matplotlib.pyplot as plt
import cv2
import numpy as np

def main():
    img_path = 's.jpg'
    rgb = plt.imread(img_path)

    gray = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
```

```
        # Perform Fast Fourier Transformation for 2D signal, i.e., image
    ftimg = np.fft.fft2(gray)
    centered_ftimg = np.fft.fftshift(ftimg)
    magnitude_spectrum = 100 * np.log(np.abs(ftimg))
    centered_magnitude_spectrum = 100 * np.log(np.abs(centered_ftimg))

        # Build filters.
    ncols, nrows = gray.shape

    global cx, cy
    cx, cy = (int)(nrows/2),(int) (ncols/2)

    filter1 = build_gaussian_filter(ncols, nrows)

    prefilter = np.ones((ncols, nrows), np.uint8)
    filter2 = cv2.rectangle(prefilter,(cx, cy),(cx + 100, cy - 90),(255,255,255),-1)

    prefilter = np.ones((ncols, nrows), np.uint8)
    filter3 = cv2.circle(prefilter,(cx + 90,cy + 90),50,(255,255,255),-1)

    prefilter = np.ones((ncols, nrows), np.uint8)
    filter4 = cv2.rectangle(prefilter,(cx - 50, cy - 150),(cx + 10, cy - 10),(255,255
    filter4 = cv2.circle(filter4,(cx + 150,cy + 150),30,(255,255,255),-1)

        # Apply filters
    ftimg_gf = centered_ftimg * filter1
    filtered_img = np.abs(np.fft.ifft2(ftimg_gf))

    ftimg_gf = centered_ftimg * filter2
    filtered_img2 = np.abs(np.fft.ifft2(ftimg_gf))

    ftimg_gf = centered_ftimg * filter3
    filtered_img3 = np.abs(np.fft.ifft2(ftimg_gf))

    ftimg_gf = centered_ftimg * filter4
    filtered_img4 = np.abs(np.fft.ifft2(ftimg_gf))

        # Save images all together by matplotlib.
    img_set = [rgb, gray, magnitude_spectrum, centered_magnitude_spectrum, filter1, f

    title_set = ['RGB', 'Gray', 'FFT2', 'Centered FFT2', 'Gaussian Filter', 'Filter2'

    matplotlib_plot_img(img_set, title_set)

def build_gaussian_filter(ncols, nrows):
    sigmax, sigmay = 10, 10
    x = np.linspace(0, nrows, nrows)
    y = np.linspace(0, ncols, ncols)
```

```
    X, Y = np.meshgrid(x, y)
    gaussian_filter = np.exp(-(((X-cx)/sigmax)**2 + ((Y-cy)/sigmay)**2))

    return gaussian_filter

def matplotlib_plot_img(img_set, title_set):
        plt.figure(figsize = (10, 10))
        n = len(img_set)
        for i in range(n):
                plt.subplot(3, 4, i + 1)
                plt.title(title_set[i])
                img = img_set[i]
                ch = len(img.shape)
                if (ch == 2):
                        plt.imshow(img, cmap = 'gray')
                else:
                        plt.imshow(img)

        plt.savefig("Fourier transform")
        plt.show()

if __name__ == '__main__':
        main()
```

11.5 Observation

After using 4 different filters on our image using Fourier transform, we got this.



Filter1:

Here we can see the in the fft output, all the low frequencies are in center. So when we applied a low pass blurry centered mask/filter to this, we got a Gaussian blur image.

Filter2:

When we applied a crisp almost centered mask/filter to this, we got a blur with boxy filtered image.

Filter3:

As we have high frequency components outside the middle. So when we applied a high pass circular mask/filter to this, we got a near to edge detection image.

Filter4:

Here we used a band pass filter and passing for both low and high frequency. Hence, we are getting both blurry and edgy image.

12 Assignment 12

12.1 Problem statement

Write a report on what different convolution layers of a CNN based image classifier sees. You may use the attached code.

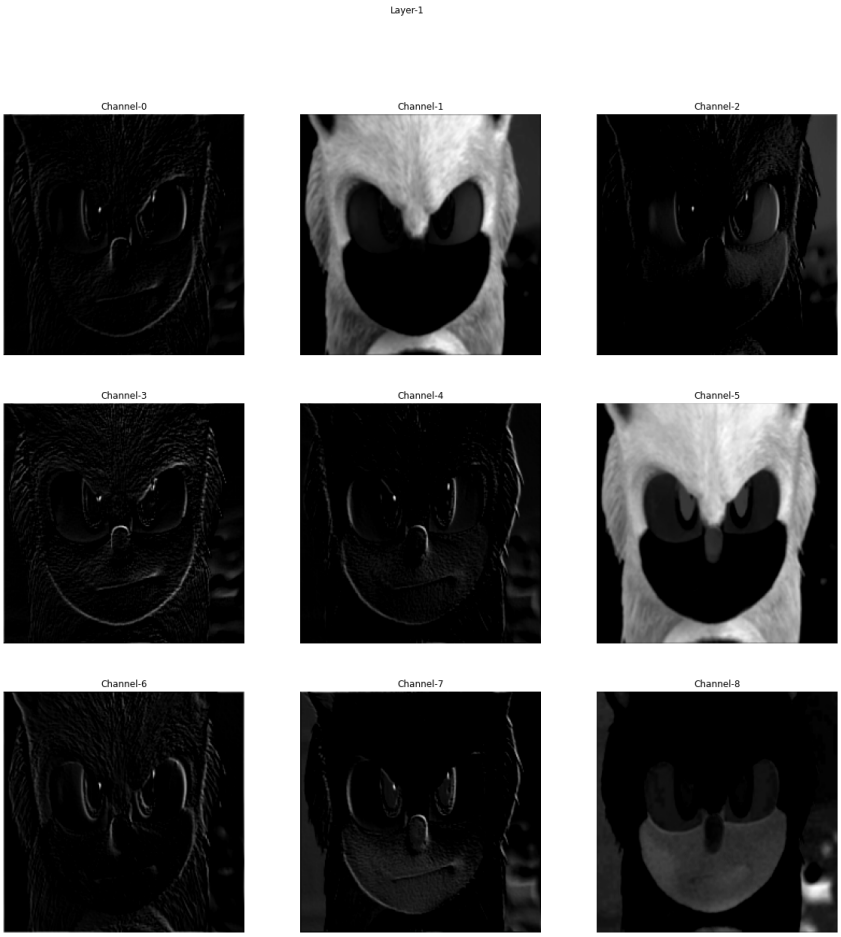
12.2 Input Image

12.3 Code

12.4 Observation

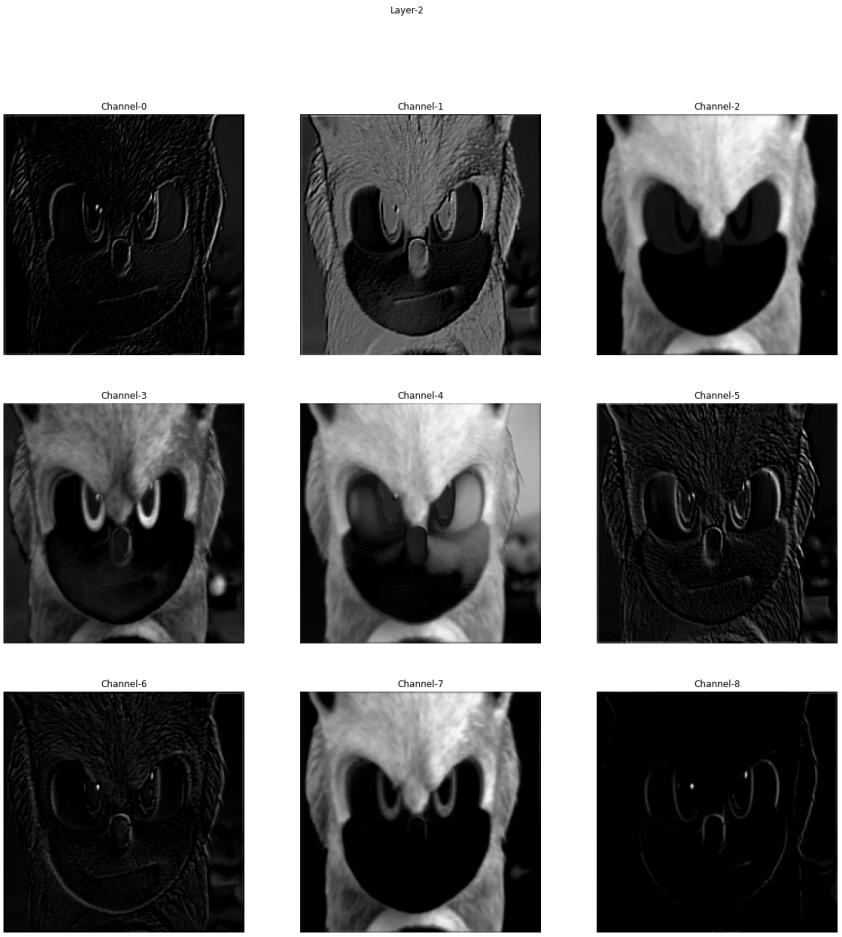
12.4.1 Layer 1

In the 1st layer channels, the edges' and the object are detected.



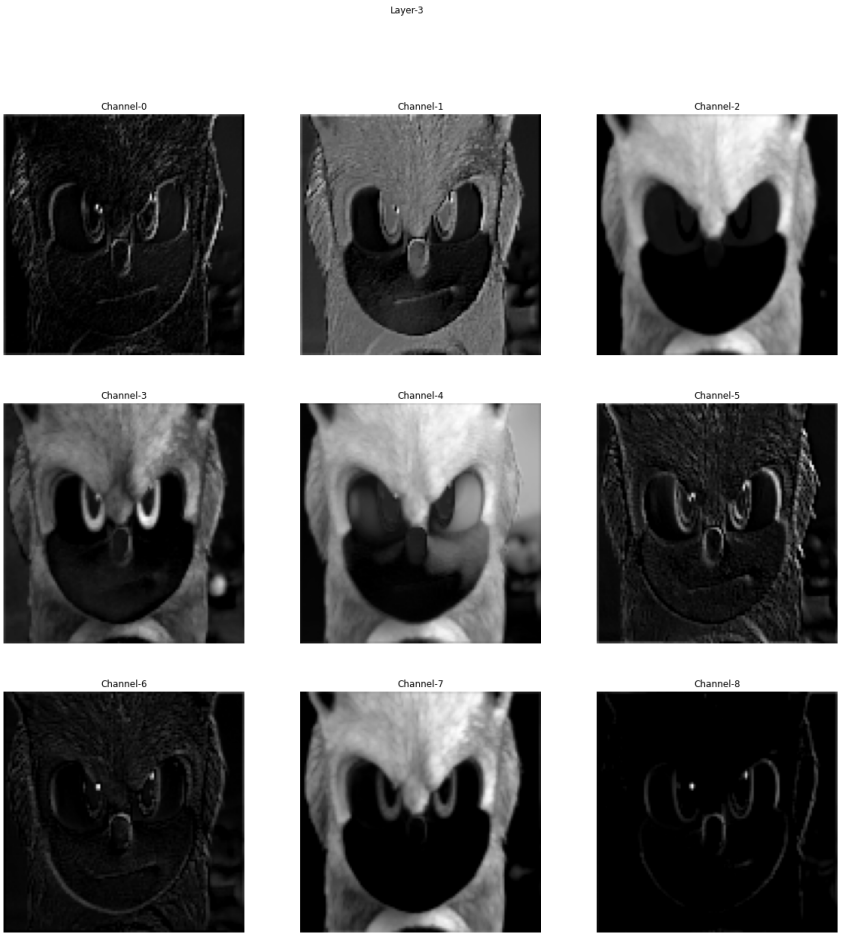
12.5 Layer 2

In 2nd layer channels again the edges are detected. But in this layer the channels are sharpened.



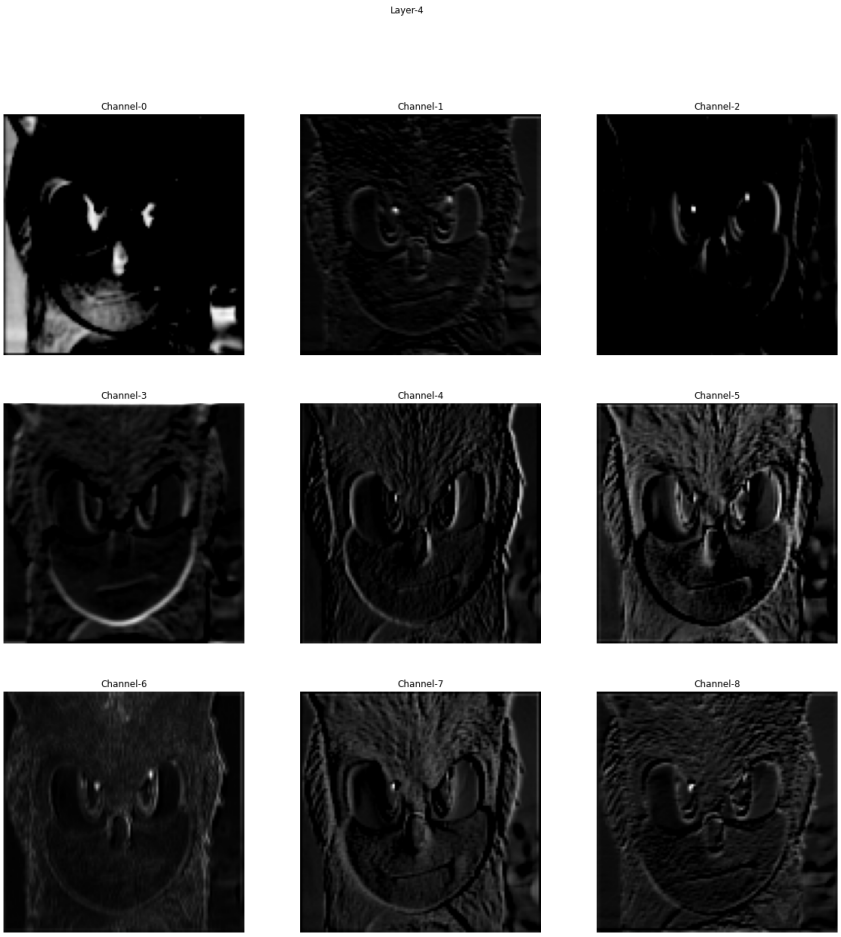
12.6 Layer 3

3rd layer is almost like 2nd layer. But they are less sharp than 2nd layer.



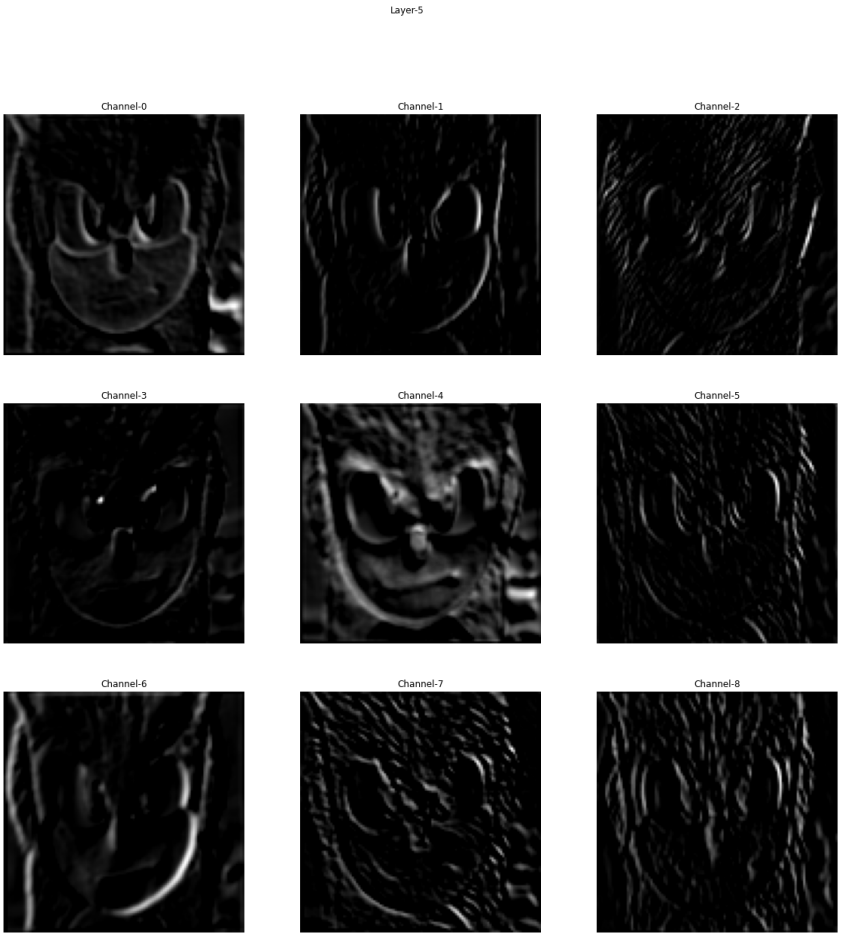
12.7 Layer 4

4th layer channels have more details than previous channels.



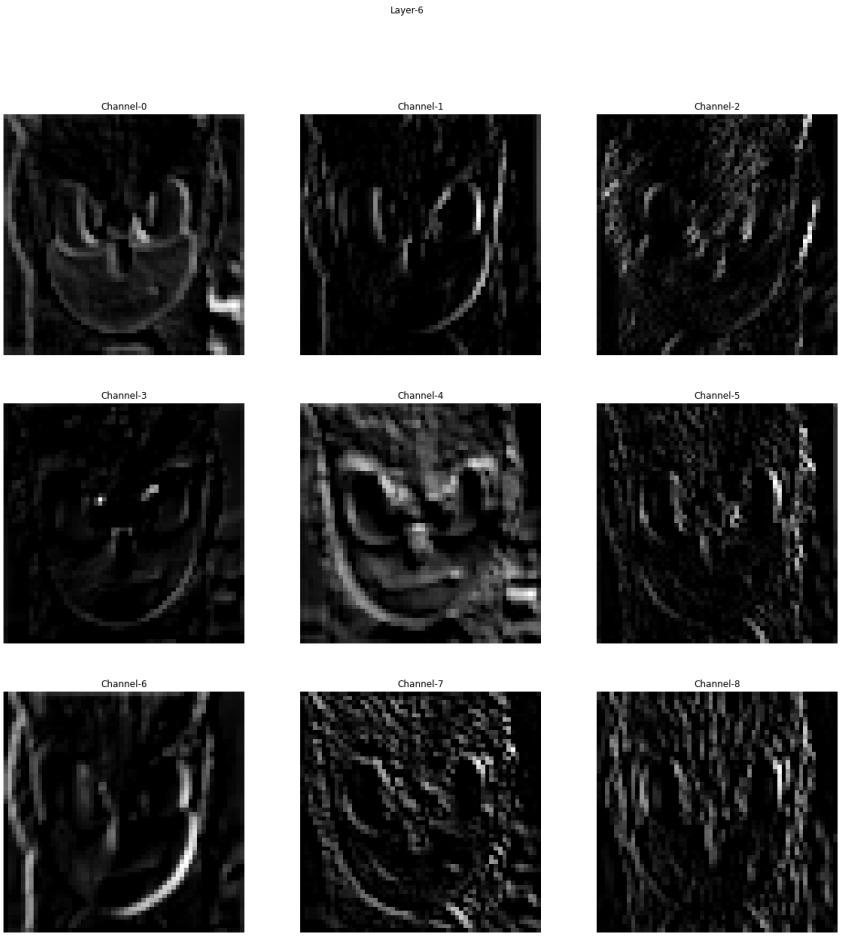
12.8 Layer 5

5th layer channels detected textures and mostly blurry.



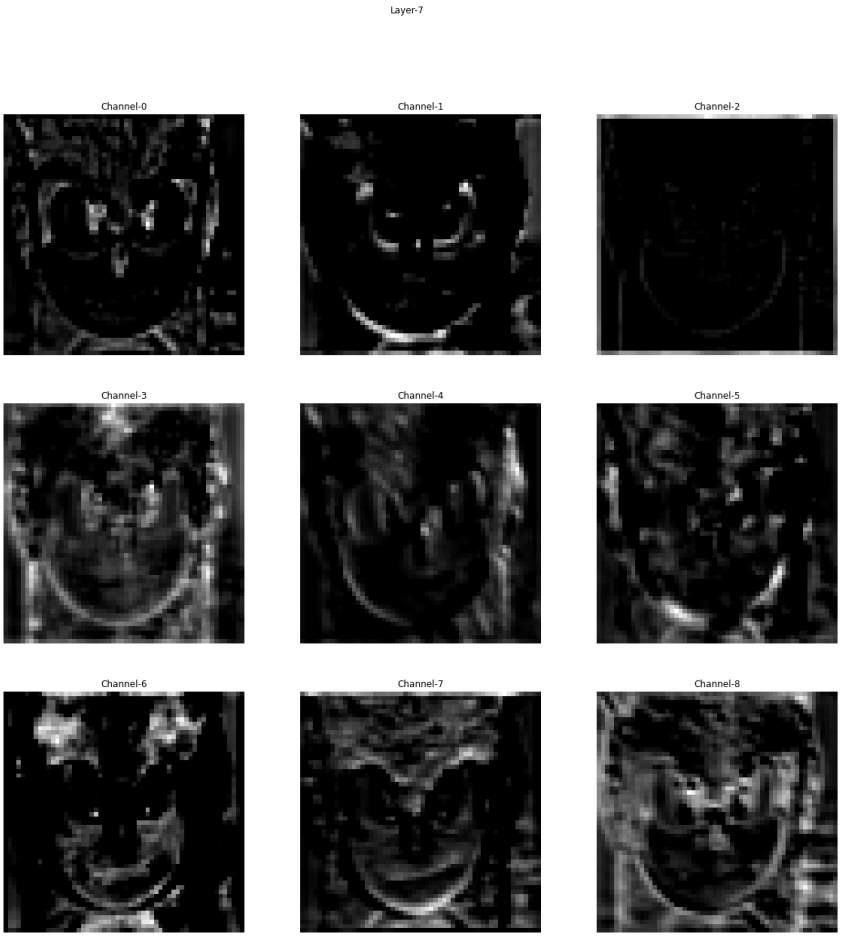
12.9 Layer 6

In the 6th layer the sampling rate is very low and detecting object became harder.



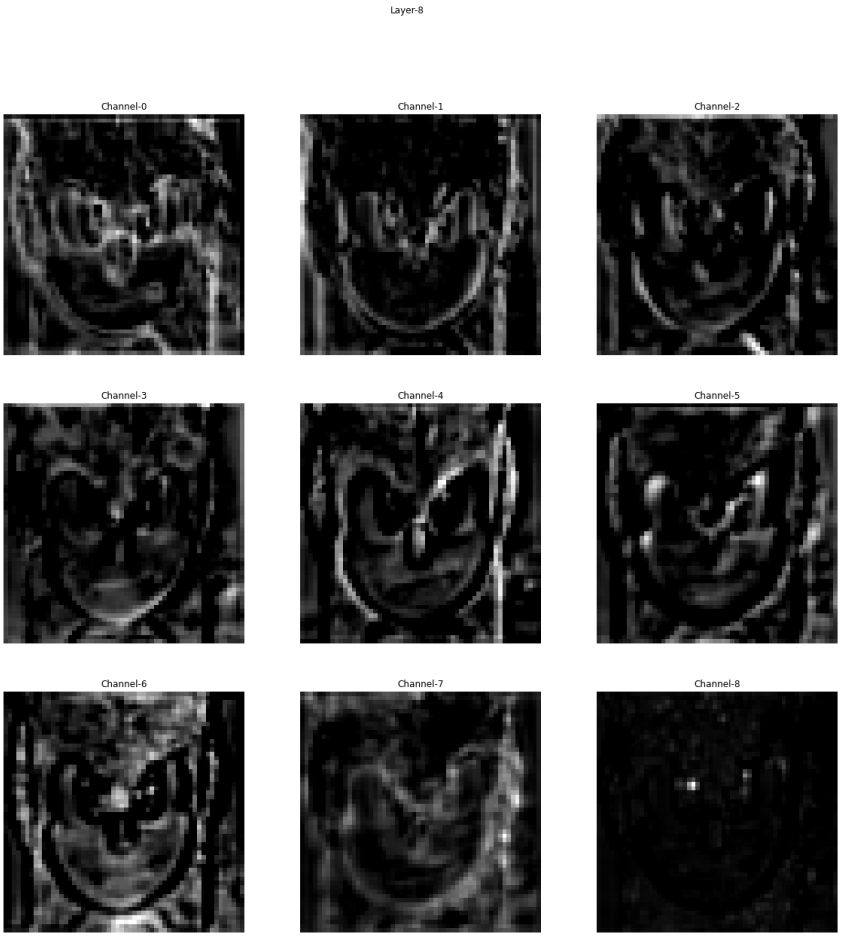
12.10 Layer 7

In this layer, only brighter intensities are detected.



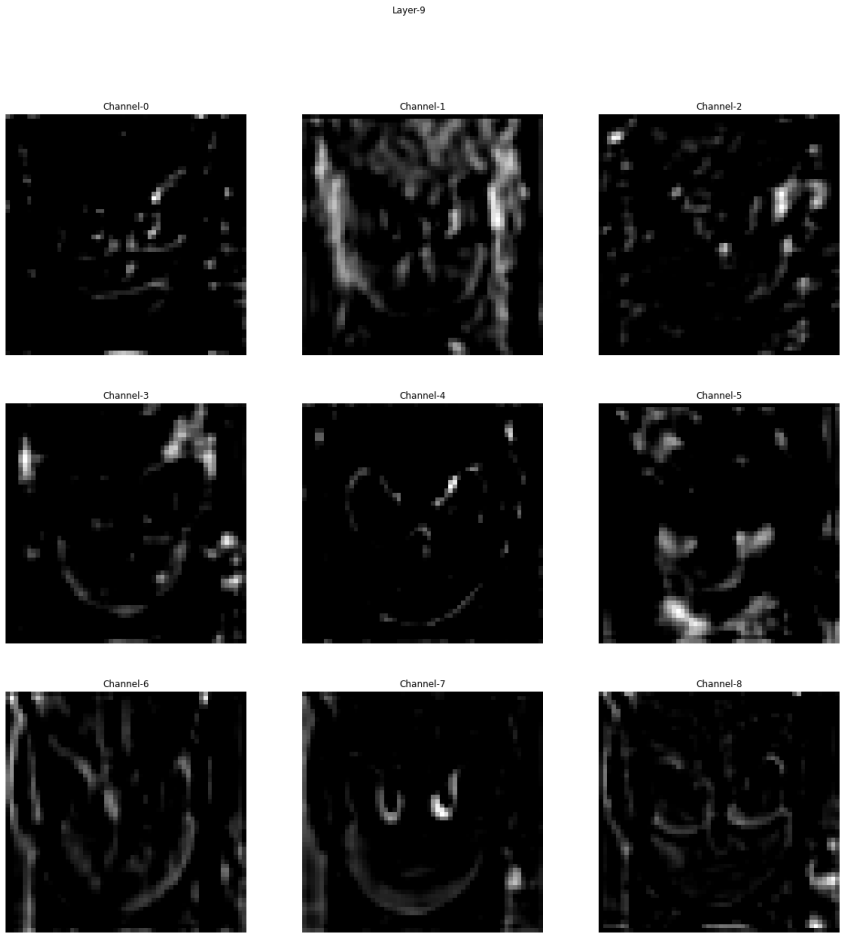
12.11 Layer 8

In this layer, the object is hardly recognized.



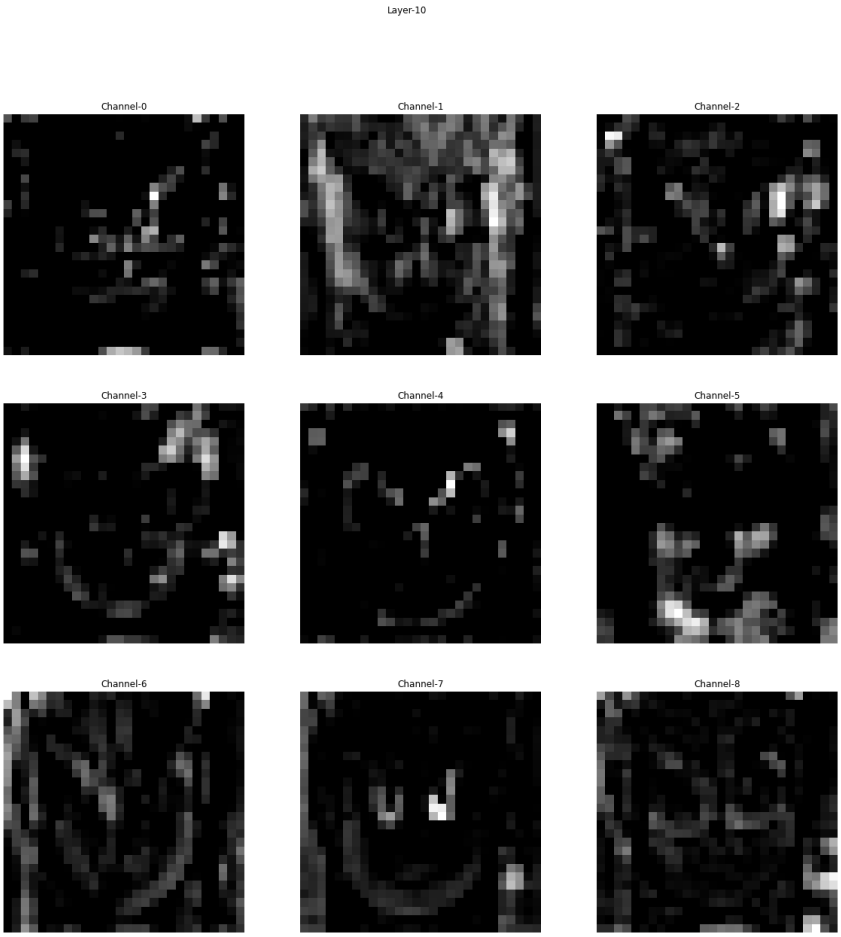
12.12 Layer 9

Change of intensities are detected in this layer.



12.13 Layer 10

Change of intensities are also detected in this layer. But in a lower sampling rate.


Layer-10
Channel-0
Channel-1
Channel-2
Channel-3
Channel-4
Channel-5
Channel-6
Channel-7
Channel-8

13 Assignment 13

13.1 Problem statement

Write a program to turn a JPEG image into a PNG image and vice-versa.

13.2 Input Image

13.3 Method

We will be using pillow python library to do that.

13.4 Code

```
from PIL import Image
# To convert the image From JPG to PNG
img = Image.open("s.jpg")
img.save("s_png.png")

# To convert the Image From PNG to JPG
img = Image.open("s_png.png")
img.save("s_jpg.jpg")
```

13.5 Observation

The Image class represents the image object. This class provides the open() method that is used to open the image.

Using the save() method, the opened image was saved as the desired format.

This is the png version of the jpg image

This is the png version of the png image

14 Assignment 14

14.1 Problem statement

Write a program to perform JPEG compression on frequency domain.

14.2 Input Image

14.3 Method

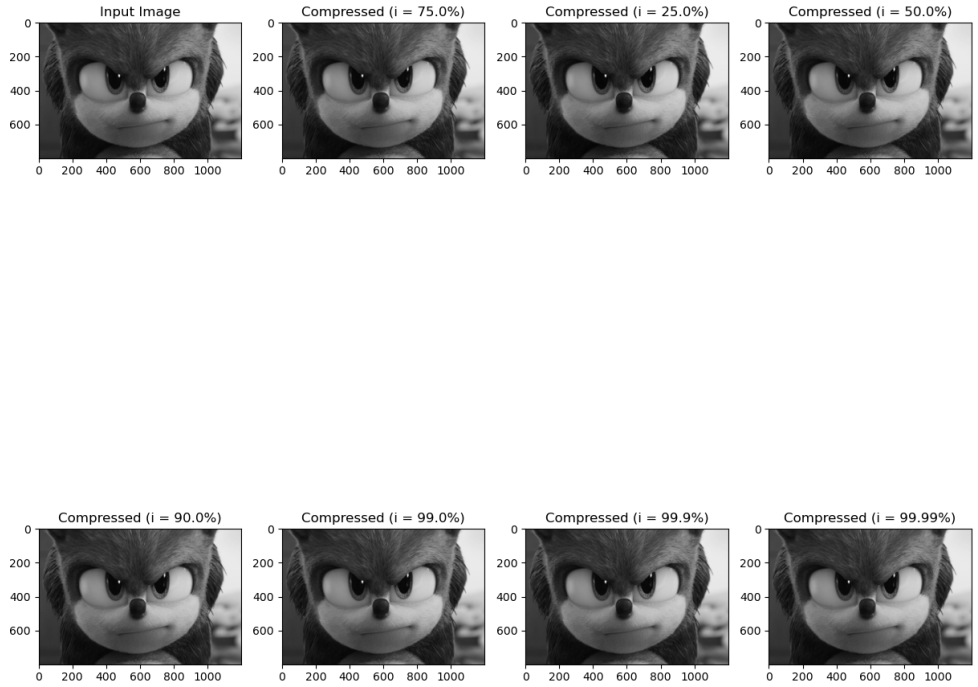
First, we did Fourier transform (FFT) on the input image. Then we compressed in frequency domain. Then we performed inverse Fourier transform to get the output in spatial domain.

14.4 Code

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from PIL import Image

src_img = cv2.imread('s.jpg',0)

fft_img = np.fft.fft2(src_img)
fft_img_sort = np.sort(np.abs(fft_img.ravel()))
n = len(fft_img_sort)
```

```
#print(src_img.shape,fft_img.shape , fft_img_sort.shape)

img_set = [src_img]
title_set = ['Input Image']

compress = [0.75, 0.25, 0.5, 0.9, 0.99, 0.999, 0.9999]
for i in compress:

    thresh = fft_img_sort[int(np.floor(n * (1 - i)))]
    ind = np.abs(fft_img) > thresh
    allow_pass = fft_img * ind
    ifft_img = np.fft.ifft2(allow_pass).real

    img_set.append(ifft_img)
    title_set.append("Compressed (i = {}%)".format(i*100))

def plot_img(img_set , title_set):
    n = len(img_set)
    r , c = 2,4
    plt.figure(figsize=(15,15))
    for i in range(n):
        plt.subplot(r,c,i+1)
        plt.imshow(img_set[i],cmap='gray')
        plt.title(title_set[i])

    plt.savefig('output.png')
    #plt.show()

plot_img(img_set,title_set)
```

14.5 Observation



After taking source image in frequency domain we convert 2D matrix in 1D and sort it as we will remove the smallest coefficient of the fourier transformation real part. In fig-13.2.1 we can see compressing more will lose the information of the picture.How much we will compress depends on our region of interest in that specific image

15 Assignment 15

15.1 Problem statement

15.2 Input Image

15.3 Method

15.4 Code

15.5 Observation