University of Rajshahi
Department of Computer Science & Engineering
CSE4182 - Digital Image Processing Lab

Final Assignment

Prepared by: MD.Ashifujjman Rafi
student id: 1810376106
Instructor: Professor Dr. Md. Khademul Islam Molla
Professor Dr. Md. Rokanujjaman
Associate Professor Dr. Sangeeta Biswas

1 Some Basics Works Before Processing the Image

Every time before process the image we need to read, convert and display an image. This job is repetitive. So, at the beginning I will try to explain those script. Later I am only writing the core part of every problems.

1.1 Import Library

To read and process image we will use Python well known library like, matplotlib, opencv and numpy.

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

1.2 Read Image

To read image we are using imread function. This function takes the path of the image and return the image.

```
path = 'mountain.jpeg'
img = plt.imread(path)
print(img.shape, img.max(), img.min())
```

1.3 Convert Image in Different Formats

To convert RGB to gray we are using cv2.CvtColor() method.

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

To convert grayscale to binary we are using cv2.threshold() method.

```
_, binary = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
```

1.4 Plot Images

Here is an example of plotting images.

```
def im_show(img_show,img_title):
    plt.figure(figsize  = (30, 30))
    for i in range(len(img_show)):
        plt.subplot(3,2,i+1)
        plt.title(img_title[i])
        plt.imshow(img_show[i],cmap='gray')
    plt.tight_layout()
    plt.show()
```

2 Histogram

2.1 Introduction

Histogram is a graphical representation of an image.The histogram measure the color distribution of an image. X axis contains the gray level intensity and Y axis contains frequency of these intensity.

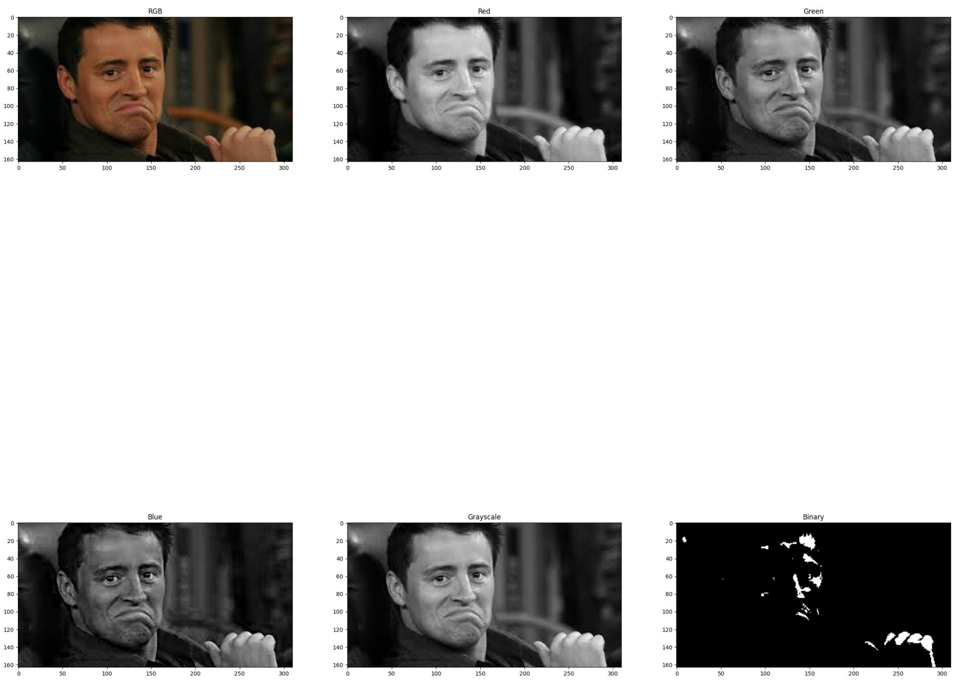


Figure 2.1: RGB,Red,Green,Blue,Binary,Gray Images

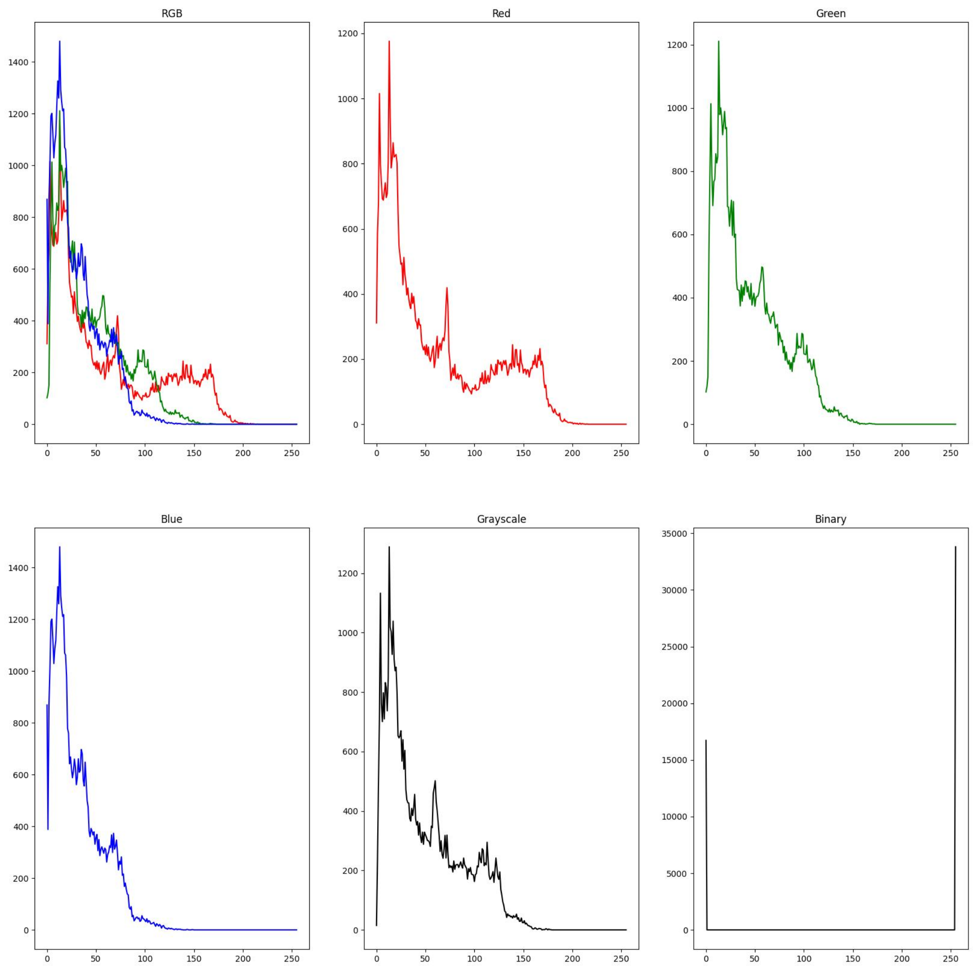


Figure 2.2: Histogram of RGB,Red,Green,Blue,Gray,binary

2.2 Using Builtin Method

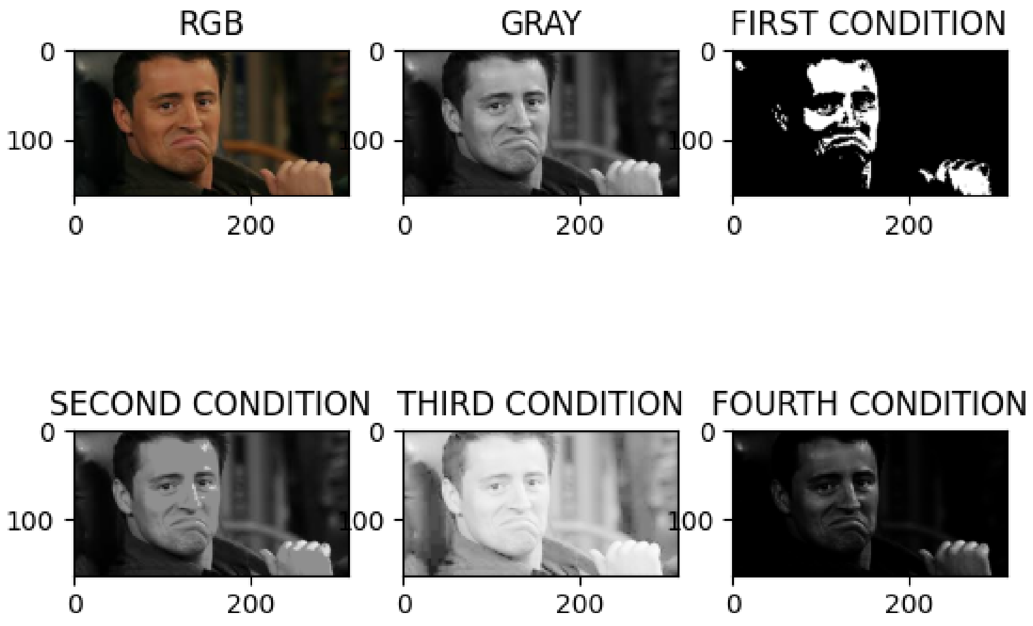
For this we are using cv2.calcHist() method.

```
green = cv2.calcHist([img], [1], None, [256], [0, 256])
blue = cv2.calcHist([img], [2], None, [256], [0,256])
```

3 Point Processing

3.1 Introduction

Point processing is used to transform an image by operating on individual pixels. If array A represents an input image then an output array B is produced by a transformation B[x, y] = T(A[x, y]), T is necessery operation



Condition for Processed_First condition

s = 100, if $r >= T1$ and $r <= T2$
otherwise s = 10.

Condition for Processed_Second condition

s = 100, if $r >= T1$ and $r <= T2$
otherwise s = r

Condition for Processed_Third condition

$$s = clog(1 + r)$$

Condition for Processed_Fourth condition

$$s = c(r + epsilon)^p$$

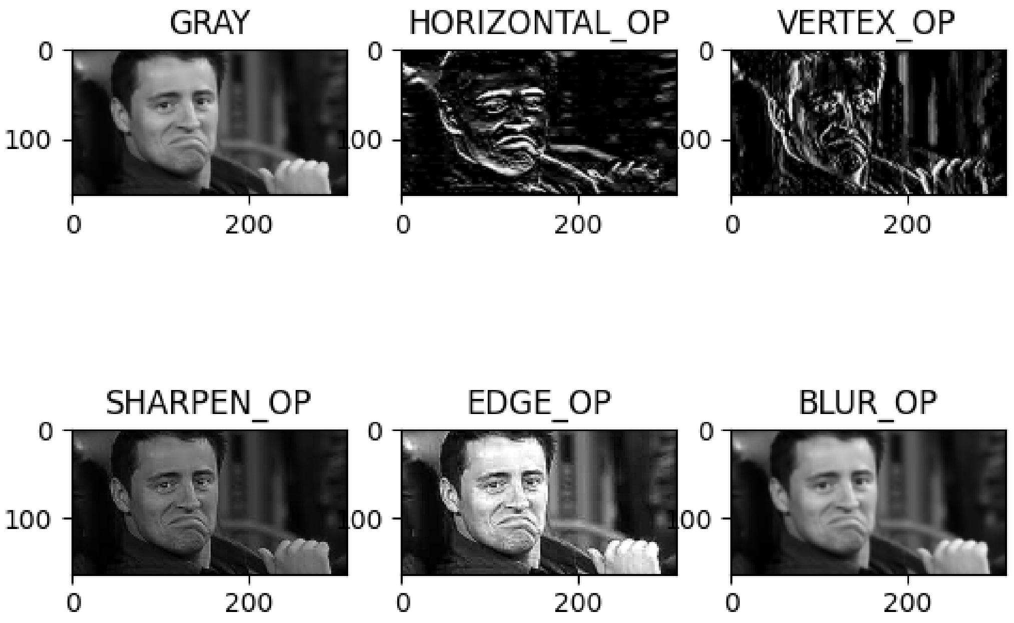
Here 'r' is the old intensity of a pixel and 's' is the new intensity.'c' and 'p' are two positive constants with any value.For 'epsilon' is very small value, like as 0.0000001.'T1' and 'T2' are two thresholds.

```
#first condition:s = 100, if r >= T1 and r <= T2; otherwise s = 10.
    condition1 = np.zeros((row,col),dtype=np.uint8)
    for x in range(row):
        for y in range(col):
            if(gray[x][y]>=t1 and gray[x][y]<=t2):
                condition1[x][y] = 100
            else:
                condition1[x][y]=10

    #second condition:s = 100, if r >= T1 and r <= T2; otherwise s = r.
    condition2 = np.zeros((row,col),dtype = np.uint8)
    for x in range(row):
        for y in range(col):
            if(gray[x][y] >= t1 and gray[x][y] <= t2):
                condition2[x][y] = 100
            else:
                condition2[x][y] = gray[x][y]

    #third condition:s = c log(1 + r) .
    condition3 = np.zeros((row,col),dtype=np.uint8)
    condition3 = c * np.log(gray + 1)

    #fourth condition : s = c ( r + epsilon ) ^ p
    condition4 = np.zeros((row,col),dtype = np.uint8)
    condition4 = c * (epsilon + gray) ** p
```

4 The effect of 5 different kernels on a image

4.1 Convolution and Kernel

A convolution is done by multiplying a pixel's and its neighboring pixels color value by a matrix.And a kernel is a small matrix. It is also known as mask. The convolution of kernel and image produces different types of effects such as blurring, sharpening, embossing and so on. That is why, kernel is also known as convolution matrix.



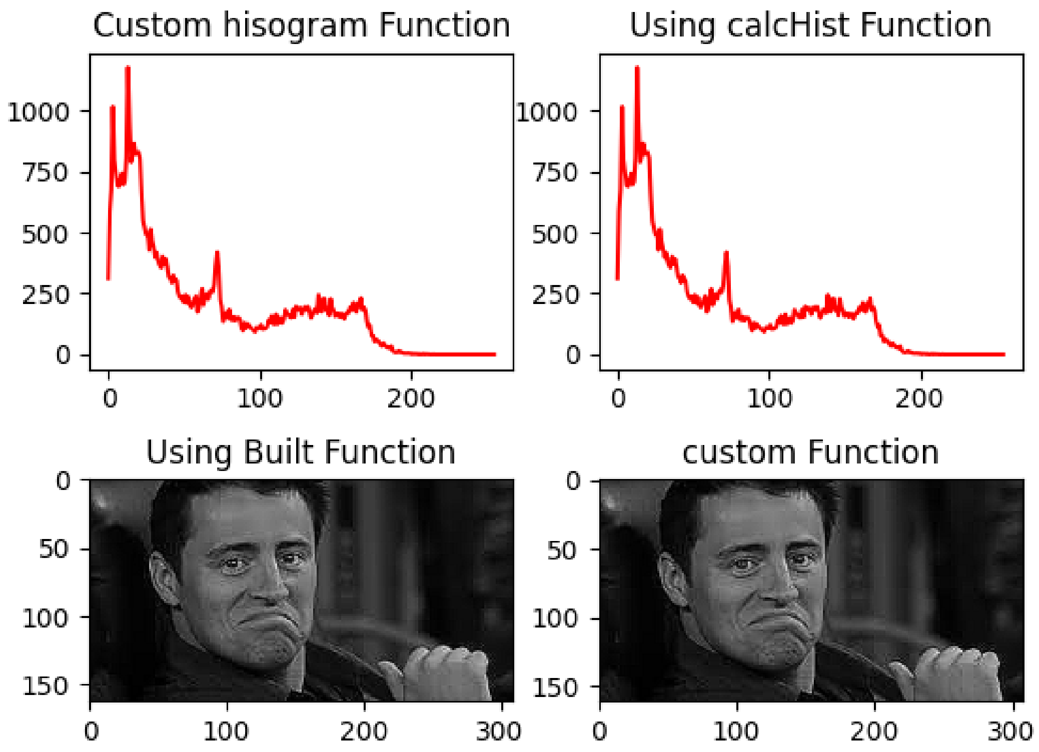
For this we are using cv2.filter2D() method.

```
    Horizontal_OP = cv2.filter2D(grayscale_imag, -1,hor_kenal)
    VERTEX_OP = cv2.filter2D(grayscale_img, -1, vertex_kernel)
    SHARPEN_OP = cv2.filter2D(grayscale_img, -1, sharpen_kernel)
    EDGE_OP = cv2.filter2D(grayscale_img, -1, edge_kernel)
    BLUR_OP = cv2.filter2D(grayscale_img, -1, box_kernel)
```

5 Comparison between build-in and Custom function(Histogram and Convolution)

5.1 Builtin vs Custom Convolution and Histogram

A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel, which is called the center pixel.



5.2 Builtin vs Custom Convolution

Builtin function:
histogram = cv2.calcHist([img], [0], None, [256], [0, 256])
custom function:

```
def custom_hist(img):
    d = dict()
    for i in range(256):
        d[i]=0

    for i in img:
        for j in i:
            d[j] += 1
    return d
```

5.3 Builtin vs Custom Histogram

Builtin function:
Convolution = cv2.filter2D(img, -1,kenal)
custom function:

```
def convulation(img,kernal):

    row , col = img.shape

    r , c = kernal.shape[0]//2 , kernal.shape[1]//2

    r , c = r*2 , c*2

    new_image = np.zeros((row-r,col-c),dtype=np.uint8)
    print(new_image.shape)

    for i in range(row-r):
        for j in range(col-c):
            x = np.sum(np.multiply(img[i:3+i,j:3+j],kernal))
            if(x < 0):
                new_image[i][j] = 0
            elif(x > 255):
                new_image[i][j] = 255
            else:
                new_image[i][j] = x
    return new_image
```

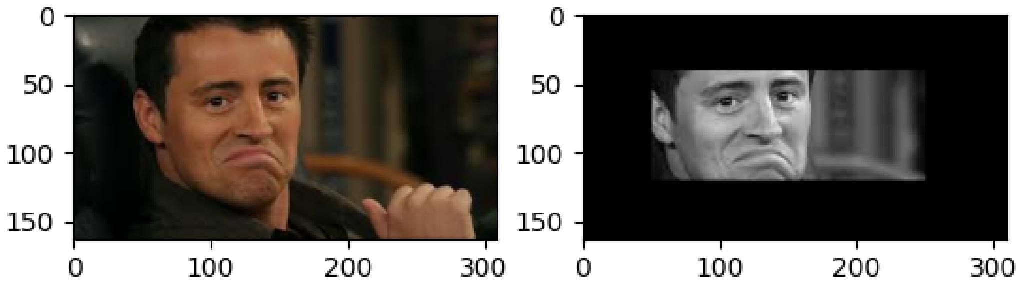
5.4 Output Analysis

The build-in function and custom function has produced same answer.

6 Binary masking, Bit slicing and Convolution with a Laplacian and Sobel filters.

6.1 Binary masking

Binary mask contains only two types of pixel values[0,1] that mean only two colors black and white. When we want to see only specific portion of image then we used it.



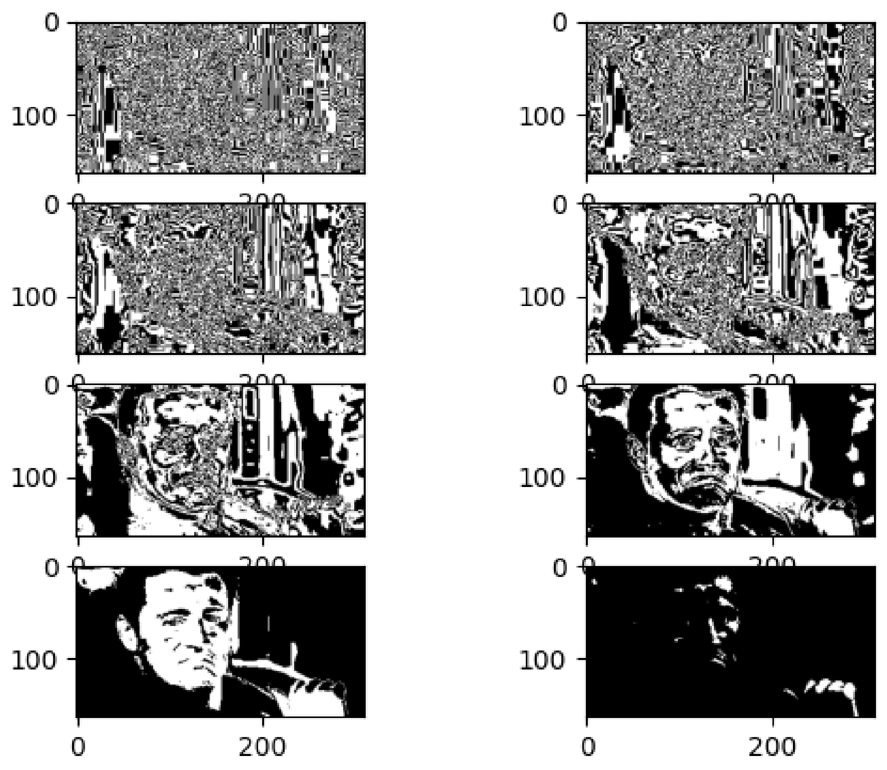
```
mask = np.zeros((r,c),dtype=np.uint8)
mask[40:120,50:250] = 255
mask = mask & grayscale
```

6.2 Output Analysis of Binary Masking

Output image contained gray image pixel values where binary masked value was 1 and others portions of image became 0.

6.3 Bit Slicing

Bit plane slicing is a method of representing an image with one or more bits of the byte used for each pixel. Often by isolating particular bits of the pixel values in an image and that's highlight interesting aspects of that image.



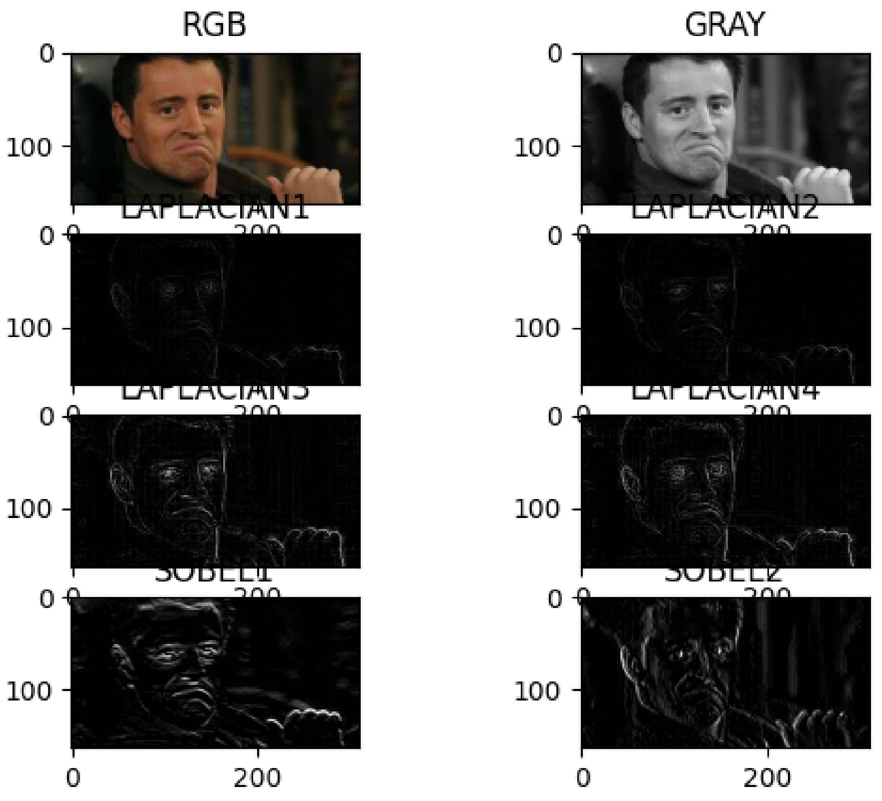
```
bit1 = grayscale & 1
bit2 = grayscale & 2
bit3 = grayscale & 4
bit4 = grayscale & 8
bit5 = grayscale & 16
bit6 = grayscale & 32
bit7 = grayscale & 64
bit8 = grayscale & 128
```

6.4 Output Analysis of Bit Slicing

According to output an images, higher order bits usually contain most of the significant visual information. Eight bit image has contained most visual information than others images and it is nearly to gray image. Then seven bit image has contained less information than eight bit image. Similarly to next bit images.Lower bit contain noise.

6.5 Convolution

In image processing, a kernel, convolution matrix, or mask is a small matrix used for blurring, sharpening, embossing, edge detection, and more. This is accomplished by doing a convolution between the kernel/filter and an image.



```
laplacian1 = np.array([[0,-1,0],[-1,4,-1],[0,-1,0]])
lap1 = cv2.filter2D(grayscale,-1,laplacian1)

laplacian2 = np.array([[1,1,1],[1,-8,1],[1,1,1]])
lap2 = cv2.filter2D(grayscale,-1,laplacian2)

laplacian3 = np.array([[0,1,0],[1,-4,1],[0,1,0]])
lap3 = cv2.filter2D(grayscale,-1,laplacian3)

laplacian4 = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])
lap4 = cv2.filter2D(grayscale,-1,laplacian4)

shovel1 = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
sov1 = cv2.filter2D(grayscale,-1,shovel1)
shovel2 = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
sov2 = cv2.filter2D(grayscale,-1,shovel2)
```

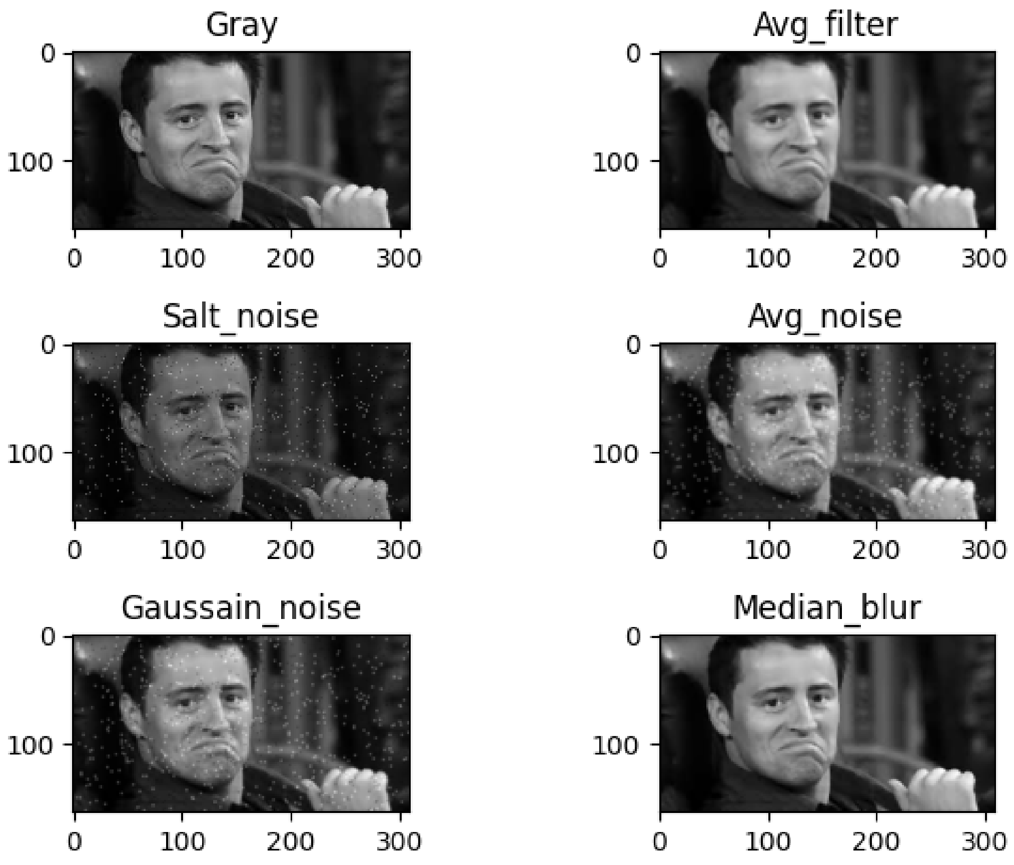
6.6 Output Analysis of convolution

According to output an images, we used builtin cv2.filter2D() function.

7 Salt and Pepper Noise

7.1 Introduction

Salt-and-pepper: An image having salt-and-pepper noise will have a few dark pixels in bright regions and a few bright pixels in dark regions. Salt-and-pepper noise is also called impulse noise.



```
#making noise image
    salt = gray.copy()
    r,c = gray.shape
    t = (r*c) // 50
    for i in range(t):
        x = np.random.randint(0,r)
        y = np.random.randint(0,c)
        if(i % 2==0):
            salt[x][y] = 255
        else:
            salt[x][y] = 0
#filters
    avarage_kernal = np.ones((3,3))/9
    gaussain_kernal = np.array([[1,2,1],[2,4,2],[1,2,1]])/16
#filtered image :
```

```
avg_gray = cv2.filter2D(gray,-1,avarage_kernal)
avarage_salt = cv2.filter2D(salt,-1,avarage_kernal)
gaussain_salt = cv2.filter2D(salt,-1,gaussain_kernal)
median_salt = cv2.medianBlur(salt,3)
```

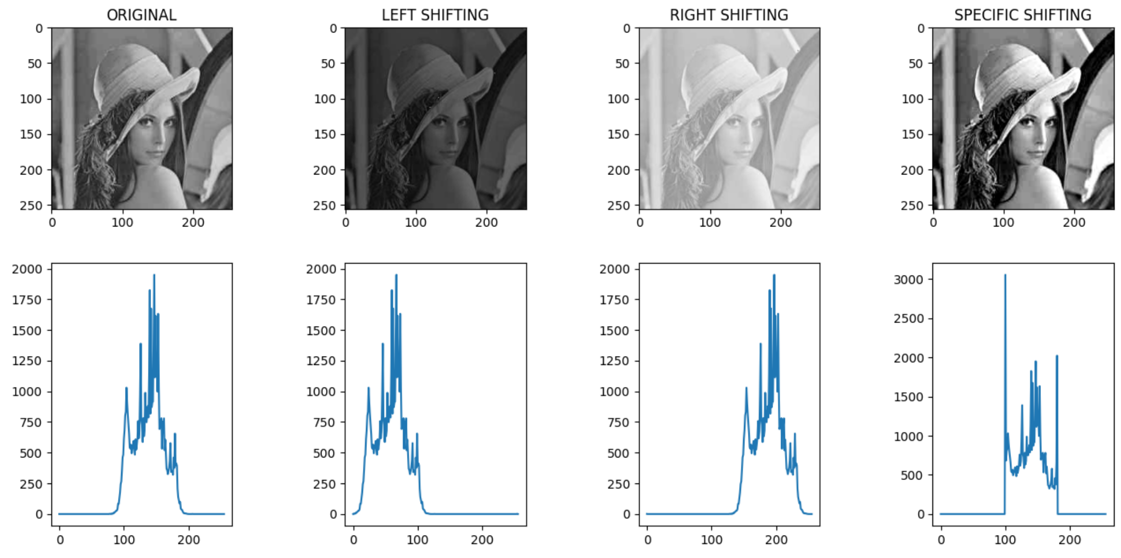
7.2 Output Analysis

We have seen that when we applied averaging filter on salt and pepper noise image it is removed some noise but remain some. On other hand there have more noise in image when we applied Gaussian kernel. But we got more effective result when we applied Median filter that is removed completely salt and pepper noise of an image.

8 Move image left, right and into a specific rang

8.1 Introduction

An image histogram is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image.



```
left,right,narrow = gray.copy(),gray.copy(),gray.copy()
        #narrow band shifting
        for i in range(r):
                for j in range(c):
                        if(narrow[i][j]<=100):
                                narrow[i][j]=100
                        elif(narrow[i][j]>=180):
                                narrow[i][j]=180


        #print(right.shape)
```

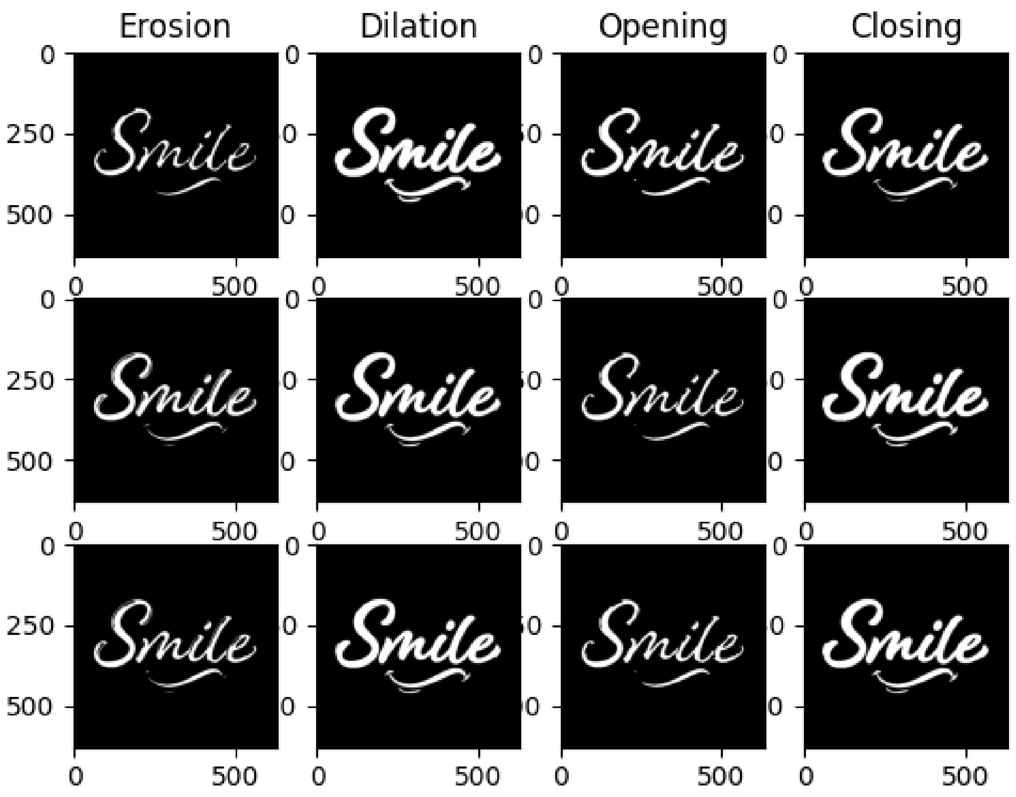
```
#left shifting
left = left -80
#right shifting
right = right+50
```

8.2 Output Analysis

An image became more white when histogram is moved right side. An image became more dark when histogram is moved right side and image contained dark and white equally when it's within narrow band.

9 Morphological Operations

Morphological image processing describe a range of image processing techniques that deal with the shape of features in an image.



9.1 Dilation

Morphological dilation makes objects more visible and fills in small holes in objects.Dilation, image (f) and structuring element (s) is derived by fs. Hit:Dilation works based on hit so any pixel is 1 and similar position structure element has 1 then output become 1 so dilation increase white pixel value in edge position that mean expanding the shapes.

```
kernal1 = np.ones((5,5),dtype=np.uint8)
    kernal2 = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]],dtype=np.uint8)
        img_dilation2 = cv2.dilate(binary_img,kernal3)
```

9.2 Erosion

Erosion works based on fit so it is increased number of back pixel in edge. We saw that an output image back portion increased in edge position. Even some parts of white within back is completely disappeared.

```
    kernal1 = np.ones((5,5),dtype=np.uint8)
        img_erosion =  cv2.erode(binary_img,kernal1)
```

9.3 Opening

Opening is a process in which first erosion operation is performed and then dilation operation is performed.

```
    opening2 = cv2.dilate(img_erosion,kernal3)
```

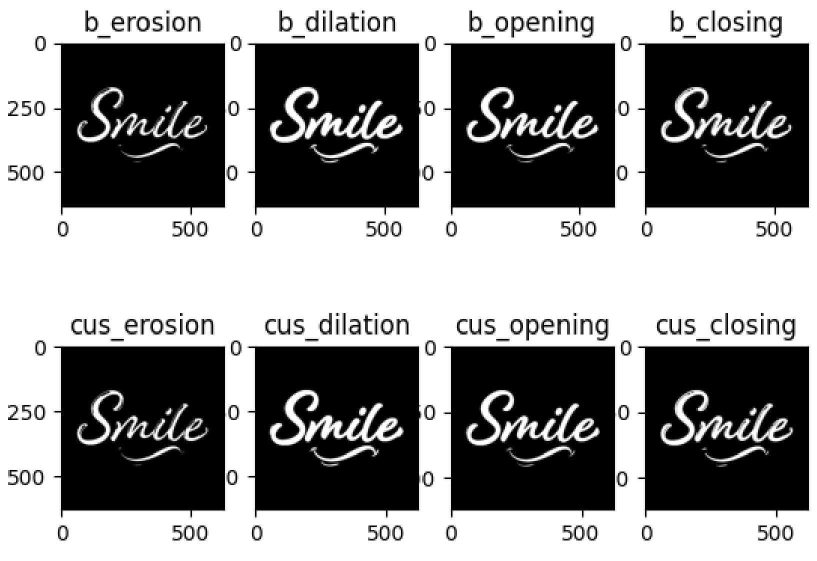
9.4 Closing

Closing is a process in which first dilation operation is performed and then erosion operation is performed.

```
    closing2 = cv2.erode(img_dilation,kernal3)
```

10 Customized Morphological Operation

First row show the result using builtin function,second row show the result using custom function



10.1 Customized Dilation

```
def dilation(img, kernal):
    r, c = img.shape
    x, y = kernal.shape
    out = np.zeros((r-x-1, c-y-1))
    for i in range(r-x-1):
        for j in range(c-y-1):

                sum = np.sum(np.multiply(img[i:i+x, j:j+y], kernal))
            if(sum >= 255):
                out[i][j] = 255
    return out
```

10.2 Customized Erosion

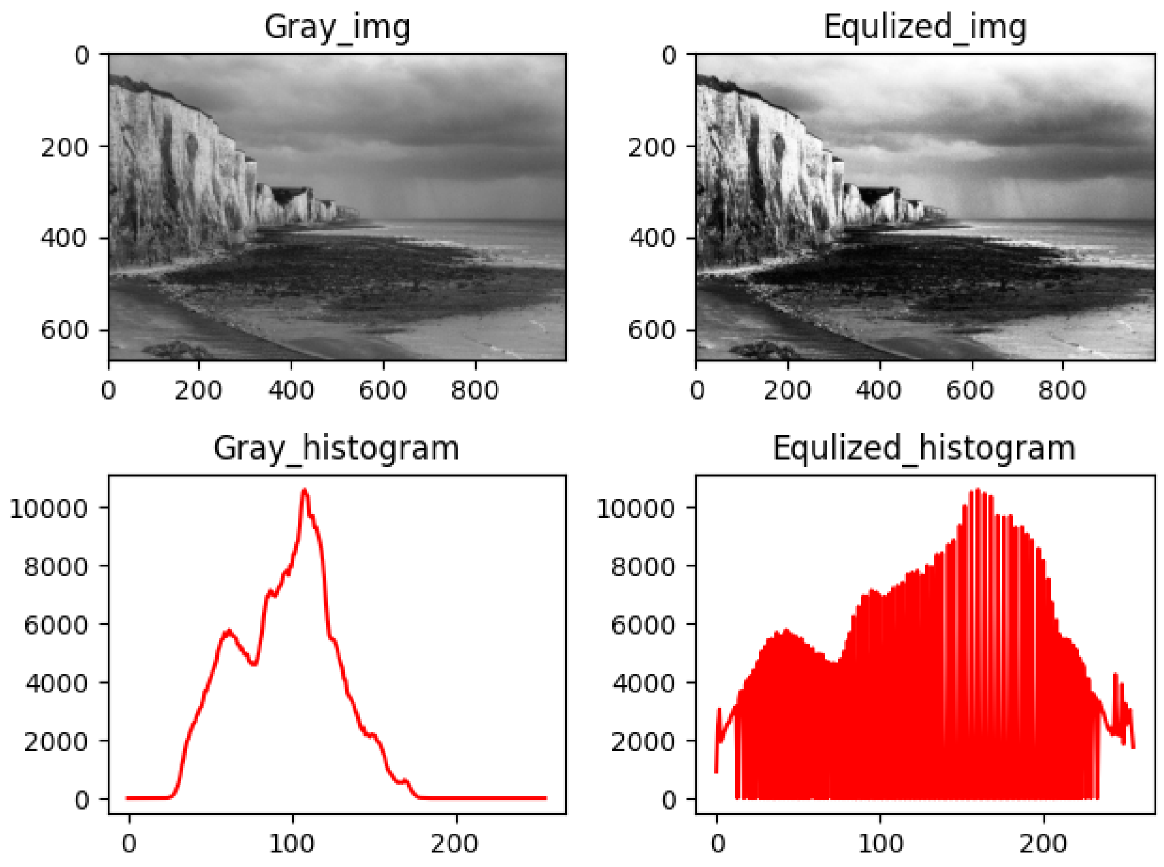
```
r, c = img.shape
    x, y = kernal.shape
    out = np.zeros((r-x-1, c-y-1))
    for i in range(r-x-1):
        for j in range(c-y-1):
            sum = np.sum(np.multiply(img[i:i+x, j:j+y], kernal))
            if(sum == 2295):
                out[i][j] = 255
    return out
```

10.3 Output Analysis

I used identical kernel in morphological operation and Performed builtin and custom erosion,dilation,opening and closing functions. For custom erosion function all the values should be 255 in 3*3 matrix ,sum will be 2295. If any section meets the condition that's origin will be 255 or 1. For custom dilation function any of pixel of 3*3 is 255 the origin will be 255 or 1.

11 Histogram Equalization

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram, more like the histogram will be evenly distributed. I used a builtin histogram equalizer function that takes one parameter , the image.

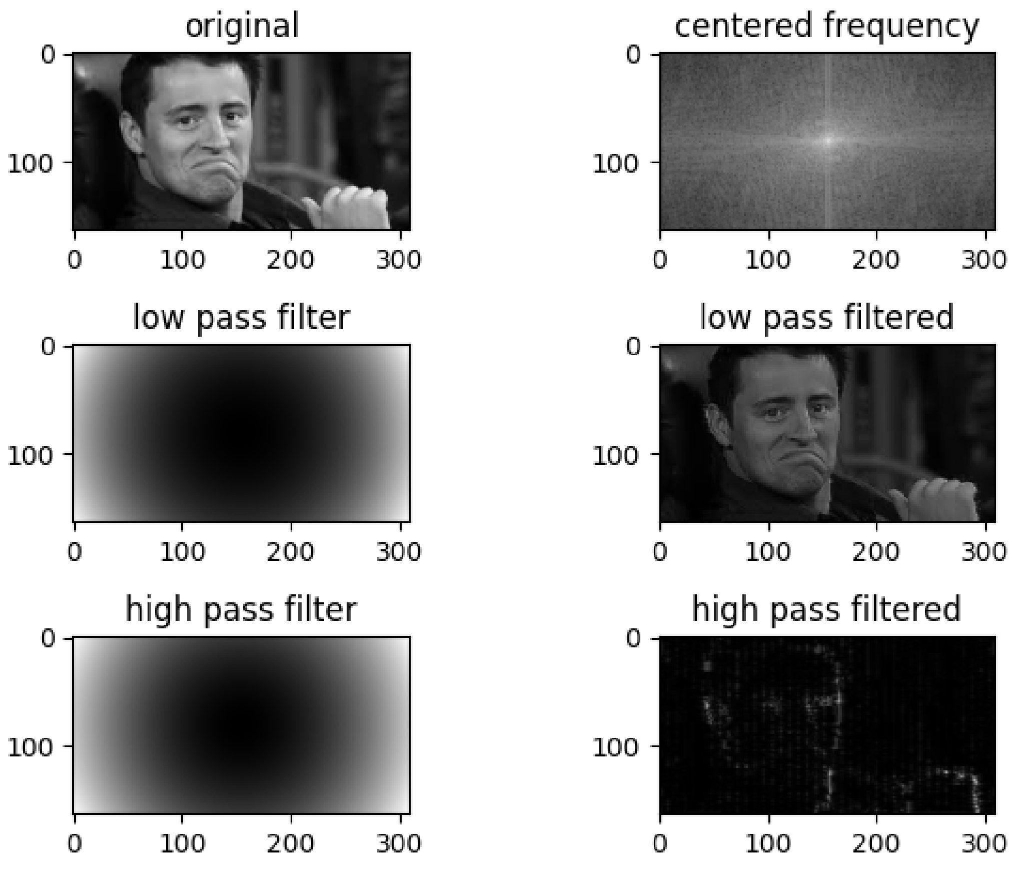


```
Equalized_image = cv2.equalizeHist(img)
```

12 Apply Filtering in Frequency Domain on Gray Image

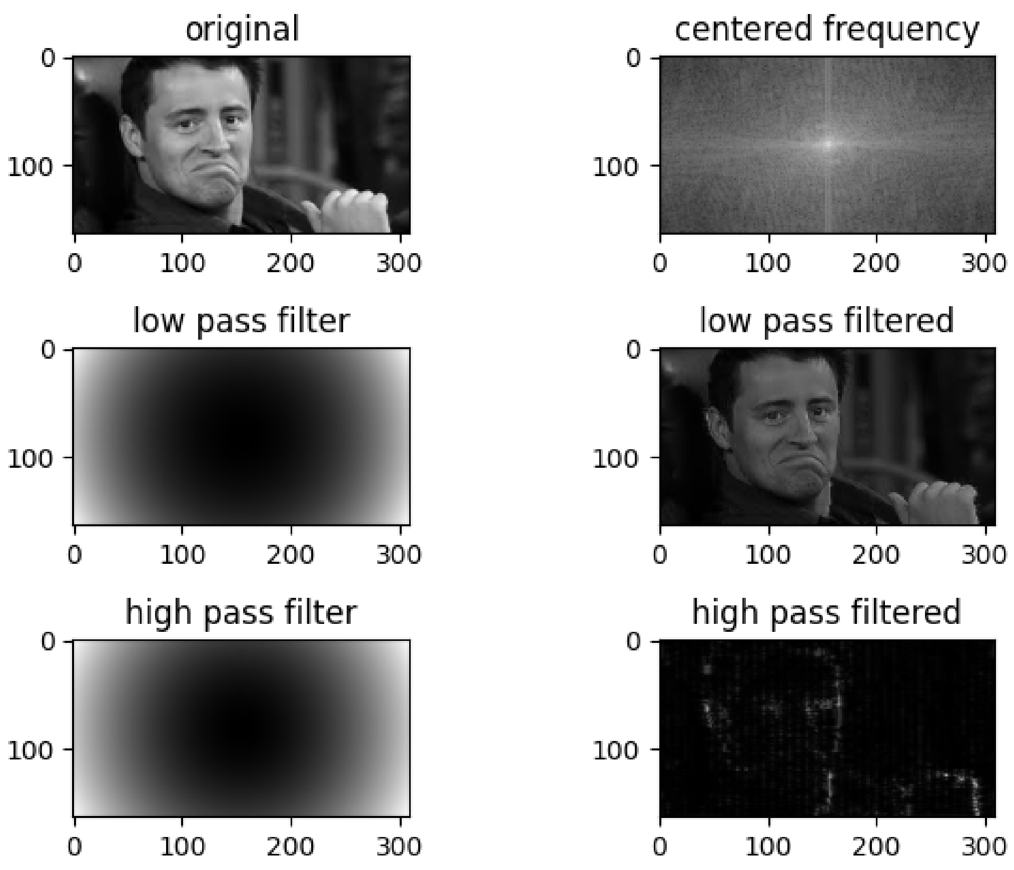
Our main aim is to use Fourier transform to reduce the computational complexity for convolution. The Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.The Fast Fourier Transform (FFT) is commonly used to transform an image between the spatial and frequency domain.
We will be following these steps :
1)Transform an image to frequency domain.(fd)
2) Moving the origin to the center for better visualization and understanding.(fds)
3) Apply filters to filter out frequencies.(fds*h)
4) Moving the origin back to its place.(fd*h)
5) Inverse fourier transform to get image back from the frequency domain.(sd*h)
I use two different filters
1.Butterworth lowpass and high pass filter

12.1 Gausssain filter



```
def gaussaian(img):
    m,n = img.shape
    #fd = frequency domain
    #fds = centered frequency
    #h = filter
    #fdsh = filtered in frequency domain
    #fdh = invert shifted
    #sdh = invert in spatial domain

    fd = np.fft.fft2(img)
    fds = np.fft.fftshift(fd)
    #for printing centered frequency abs will need
    fds_abs = np.log1p(np.abs(fds))

    #now making the low pass filter
    h = np.zeros((m,n),dtype=np.float32)
    d0 = 70
    for u in range(m):
        for v in range(n):
            d = np.sqrt((u-m/2)**2+(v-n/2)**2)
            h[u,v] = np.exp((-d)**2/(2*d0**2))
    h_abs = np.log1p(np.abs(h))
    fdsh= fds*h
    #filtered in frequency domain
    fdsh_abs = np.log1p(np.abs(fdsh))

    #converted to sptial domain
    fdh = np.fft.ifftshift(fdsh)
    sdh = np.abs(np.fft.ifft2(fdh))

    #high pass filter
    hp = 1-h
    hp_abs = np.log1p(np.abs(hp))
    #filtered in frequency domain
    fdshp = fds*hp
    #filtered in spatial domain
    fdhp = np.fft.ifftshift(fdshp)
    sdhp = np.abs(np.fft.ifft2(fdhp))
```

12.2 Butterworth filter

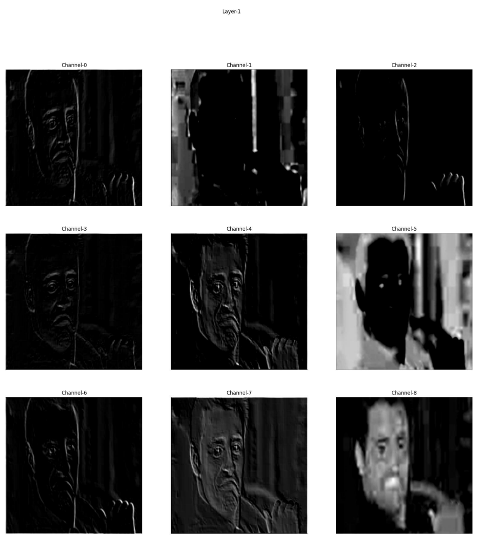


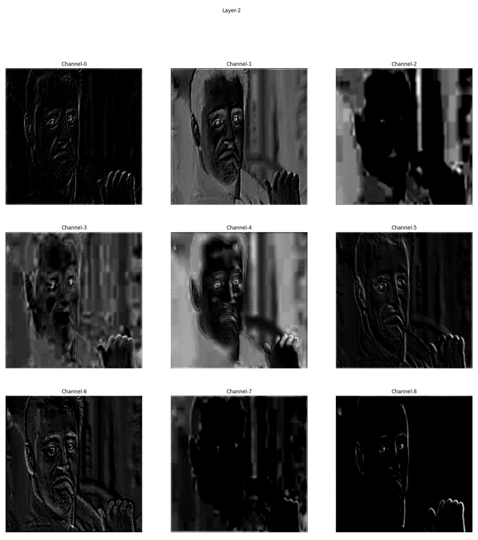
```
def butterworth(img):
    m,n = img.shape
    h = np.zeros((m,n),dtype=np.float32)
    fd = np.fft.fft2(img)
    fds = np.fft.fftshift(fd)
    fds_abs = np.log1p(np.abs(fds))
    d0 = 30
    for u in range(m):
        for v in range(n):
            d = np.sqrt((u-m/2)**2+(v-n/2)**2)
            h[u,v] = 1/(1+(d/d0)**2)
    h_abs = np.log1p(np.abs(h))
    fdsh = fds*h
    fdh = np.fft.ifftshift(fdsh)
    sdh = np.abs(np.fft.ifft2(fdh))

    #highpass butterworth
    hp = 1-h
    hp_abs = np.log1p(np.abs(hp))
    fdshp = fds*hp
    fdhp = np.fft.ifftshift(fdshp)
    sdhp = np.abs(np.fft.ifft2(fdhp))
```

13 Effect of different convolution layers of a CNN based image classifier

A convulational neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data.We need a pre-trained model for processing image that produces output of giving image. We used VGG16() pre-trained model. It has 16 layers that are numbered from 0 to 15.

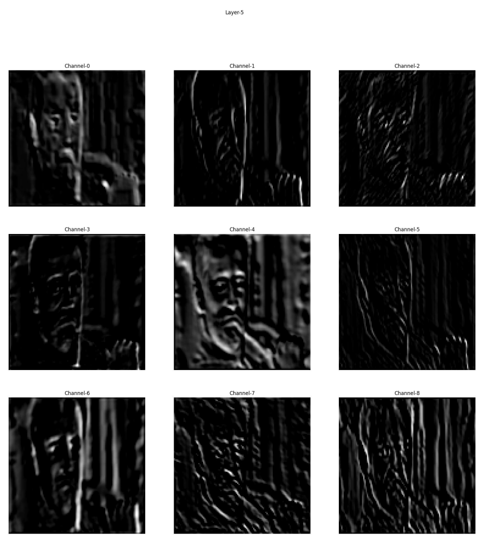
13.1 Effect of different layers



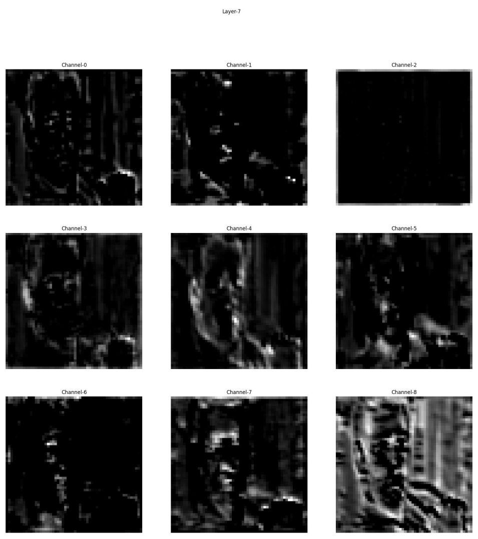


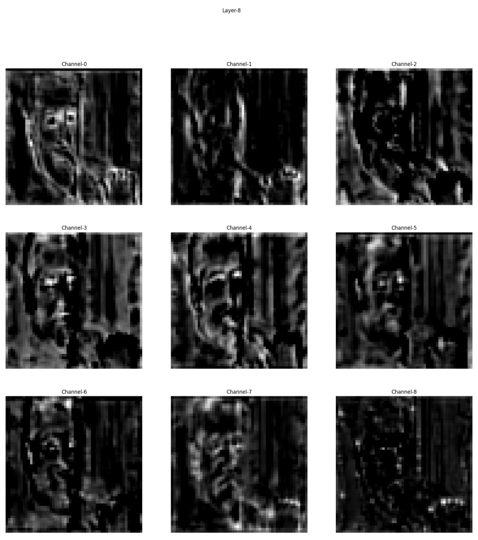
Layer-3
Channel-0
Channel-1
Channel-2
Channel-3
Channel-4
Channel-5
Channel-6
Channel-7
Channel-8

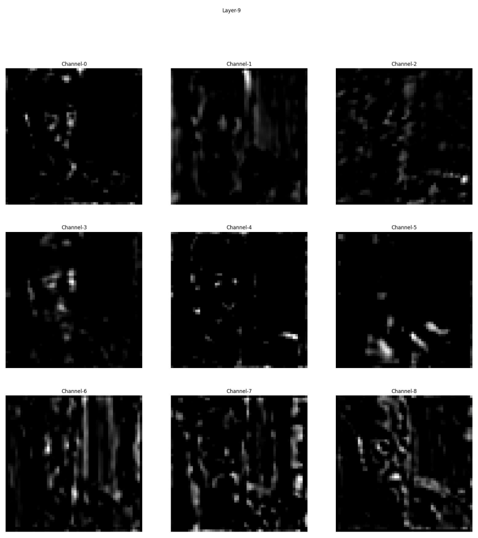
Layer-4
Channel-0
Channel-1
Channel-2
Channel-3
Channel-4
Channel-5
Channel-6
Channel-7
Channel-8

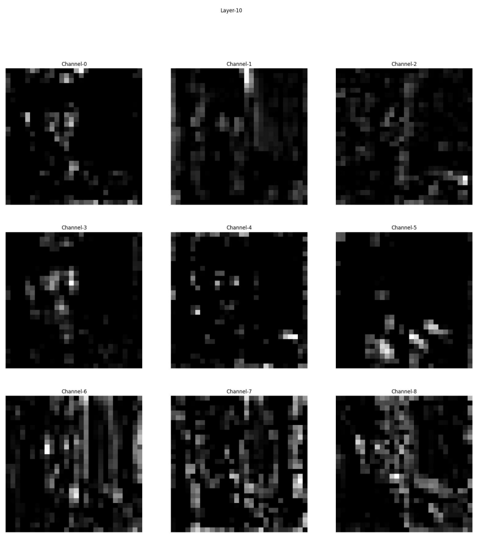

Layer-5
Channel-0
Channel-1
Channel-2
Channel-3
Channel-4
Channel-5
Channel-6
Channel-7
Channel-8

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


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


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





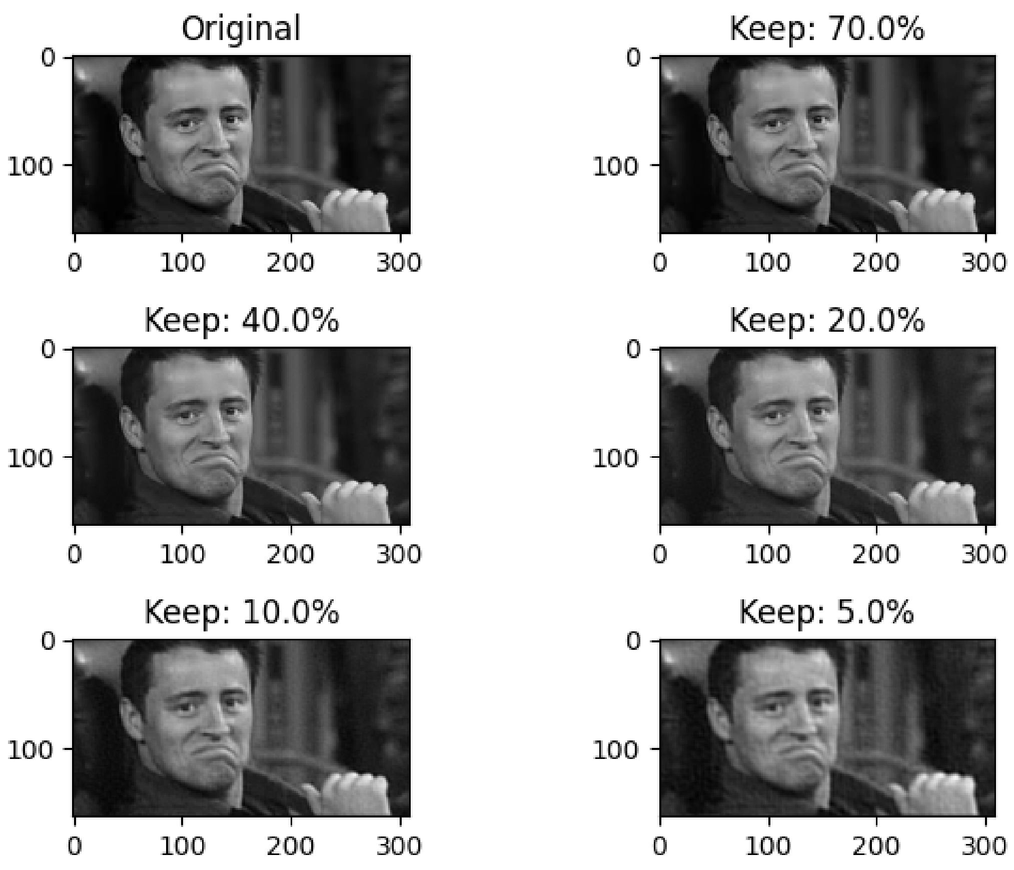
14 JPEG image into a PNG image and vice-versa

I used builtin function to convert image to png,

```python
def jpg2png():
    img_path = "chan.jpg"

    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)
```

15 Image Compression

I got similar of gray image When i compressed 300 percent. When we compressed 40 percent of gray image, then output contains less data of original image and when we compressed 80 ,it also removed fine details of image, similar to 90 percent compression it blury.



```
def imgcompress(img):
    img_set = [img]
    img_title = ["Original"]

    fd = np.fft.fft2(img)
    fd_sort = np.sort(np.abs(fd.flatten()))
    size = len(fd_sort)

    for keep in (.7,.4,.2,.1,.05):

        thresh = fd_sort[int(np.floor(size*(1-keep)))]
        index = np.abs(fd) > thresh
        compressed = fd*index
        after_com = np.fft.ifft2(compressed).real
        print(after_com.size)

        img_set.append(after_com)
```

```
img_title.append("Keep: "+str(keep*100)+"%")
```