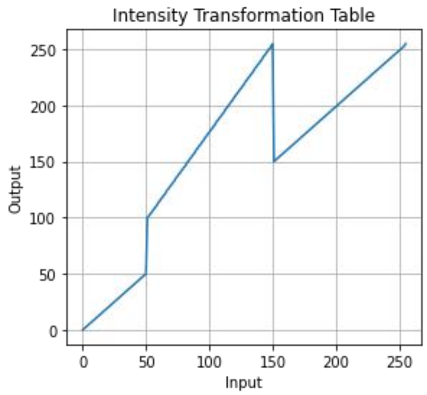
EN2550 - Fundamentals of Image Processing and Machine Vision

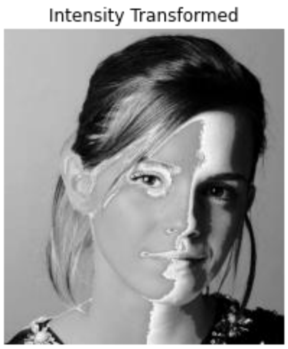
Assignment – 1

M. D. A. J. Abeyratne – 190009U

Question 1

In this question, an intensity transformation is done. The intensities from 0 - 50 and 150 - 255 have not changes, and are remained the same, while the intensity values between 50 and 150 are mapped to a wider, and brighter range of intensities, as shown in the intensity transformation table. The intensities of the left side of the face have not changes as they are more than 150, the beginning of the right side has a higher intensity as the intensities of that region are situated in the slightly less than 150 region.



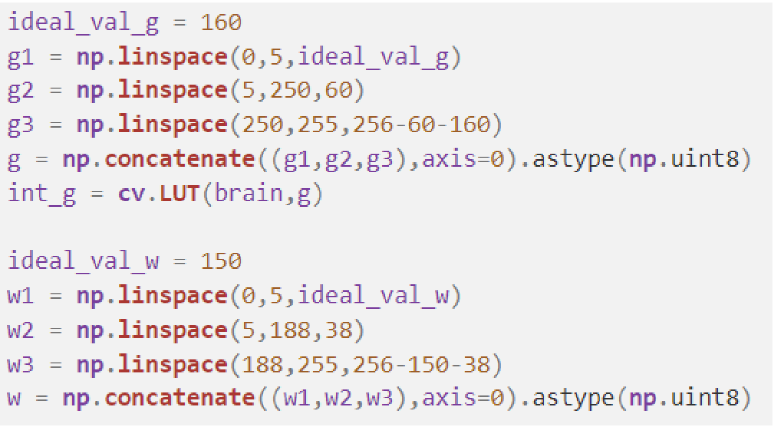


Question 2

The gray matter of the brain consists of neurons, while the white matter consists of glial cells and myelinated axons. Here in the given image, the darker part is the white matter, while the lighter part is the gray matter. First the histogram of the image was observed. The larger number of dark pixels were present due to the background of the image. The intensity levels from 150 - 180 area represents the white matter, and the intensity levels from 180 - 220.

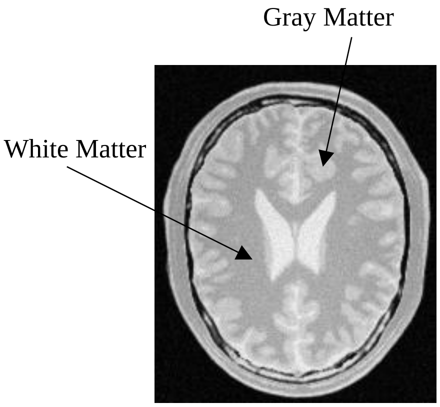
To accentuate the white matter, the range 150 - 188 is mapped to a wider range of values from 5 - 188. These values were taken by a trial-and-error method, by using several values, and choosing the more appropriate one. The other brighter area was kept the same, without doing any changes to the transform. To emphasize the white matter more, the image was inverted using the `cv.bitwise_not()` function.

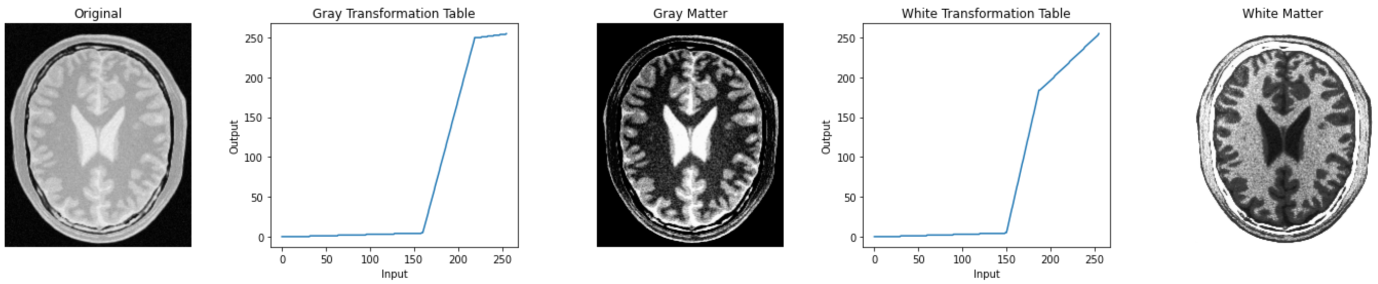
To accentuate the gray matter, the range from 160 - 220 is mapped to a wider range from 2 - 255. While reducing the range of the white matter, the gray matter range is increased.



```
ideal_val_g = 160
g1 = np.linspace(0,5,ideal_val_g)
g2 = np.linspace(5,250,60)
g3 = np.linspace(250,255,256-60-160)
g = np.concatenate((g1,g2,g3),axis=0).astype(np.uint8)
int_g = cv.LUT(brain,g)

ideal_val_w = 150
w1 = np.linspace(0,5,ideal_val_w)
w2 = np.linspace(5,188,38)
w3 = np.linspace(188,255,256-150-38)
w = np.concatenate((w1,w2,w3),axis=0).astype(np.uint8)
```





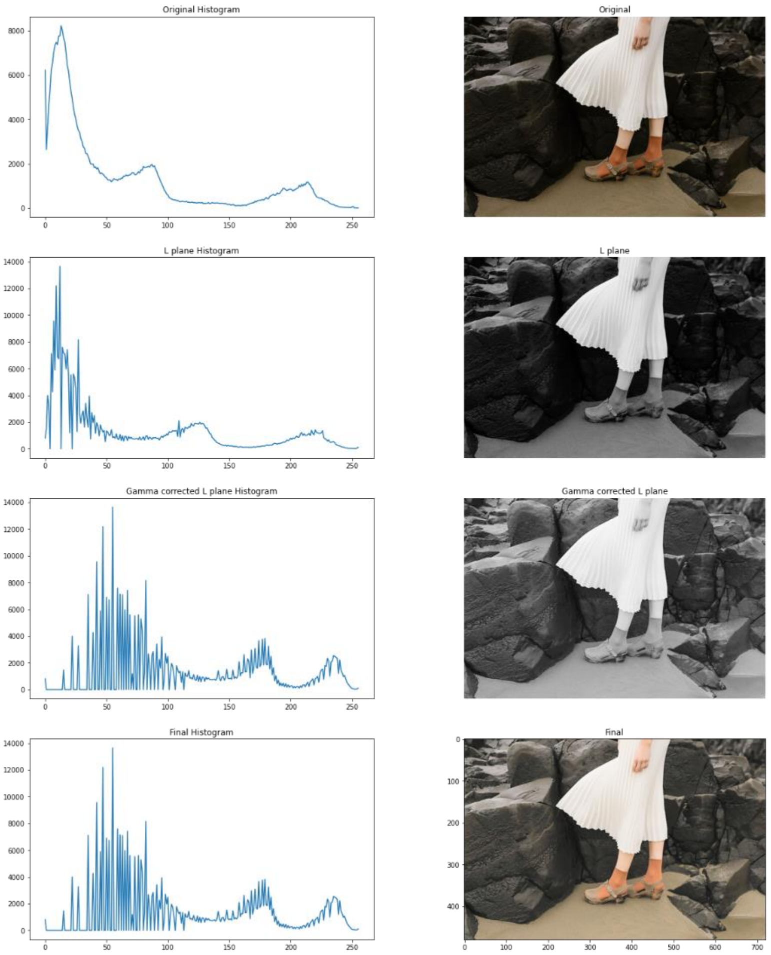
Question 3

The LAB color space of an image represents the following data.

L - Lightness (Intensity)

A - Color component ranging from Green to Magenta.

B - Color component ranging from Blue to Yellow.

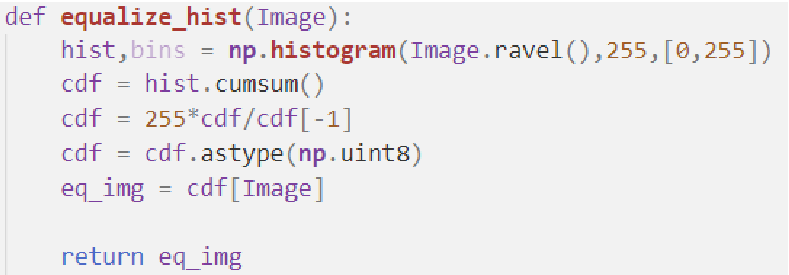


Here, the gamma correction is to be made to the L plane. So, the image is first converted to the LAB color space from the BGR color plane using the function `cv.cvtColor(img, cv.COLOR_BGR2LAB)`. The output is split to the L, A, B, color spaces using `(L,A,B)=cv.split(img_con)`. By choosing the gamma value as 0.5, the gamma correction is done to the L plane. The corrected values are assigned by lookup table function `cv.LUT()` and merged to together with the unchanged A and B planes and make the 3D array again using `cv.merge([gam,A,B])` where `gam` is the gamma corrected L plane. When the gamma correction is 0.5 or less than 1, what it does is, it maps the darker region of an image to a wider region, so it will show more details in the darker region and wise versa.

Applying the gamma correction directly to the RGB image will give a different result as it will do the gamma correction to all the R, G, B planes.

Question 4

Histogram equalization is a method used to have more vibrant images. When the histogram values are equal, the details for all the color intensities will be prominently visible in the image and will increase the contrast. The code for histogram equalization is as follows



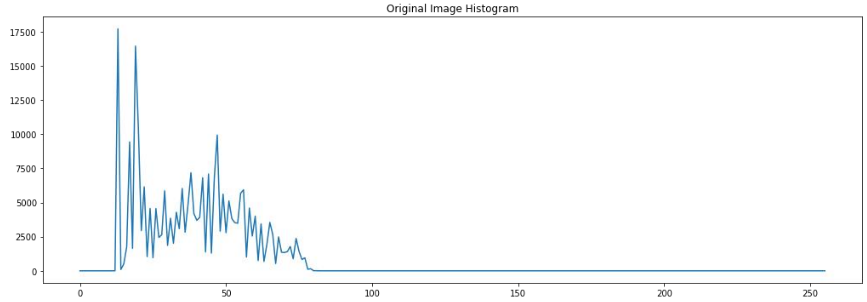
```
def equalize_hist(Image):
    hist,bins = np.histogram(Image.ravel(),255,[0,255])
    cdf = hist.cumsum()
    cdf = 255*cdf/cdf[-1]
    cdf = cdf.astype(np.uint8)
    eq_img = cdf[Image]

    return eq_img
```

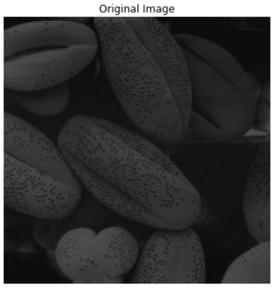
In this histogram equalization function, the histogram of the image is taken as a cumulative sum, and normalized it by dividing from the highest value, and multiplying by 255

The following diagram represents the histogram and the image of the original, histogram and the image after manually equalizing histogram, and histogram and the image after equalizing the histogram using the in-built `cv.equalizeHist()` function.

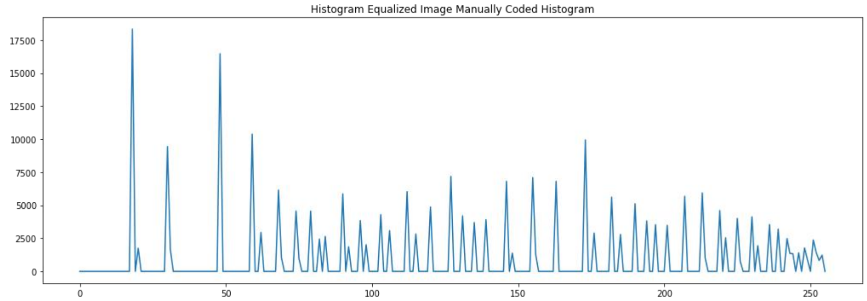
Original Image Histogram



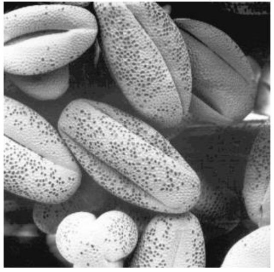
Original Image



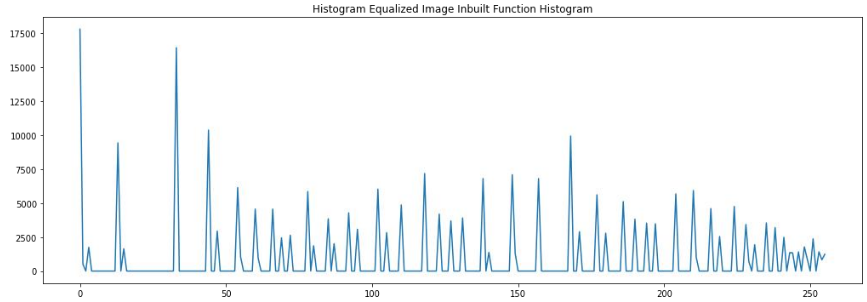
Histogram Equalized Image Manually Coded Histogram



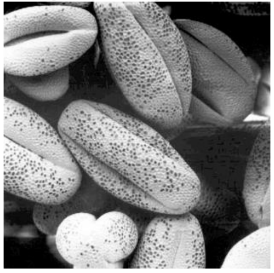
Histogram Equalized Image Manually Coded



Histogram Equalized Image Inbuilt Function Histogram



Histogram Equalized Image Inbuilt Function

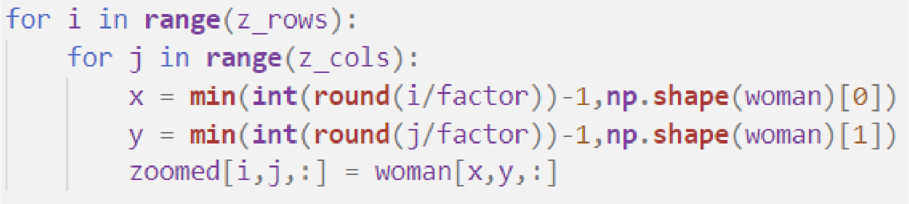


Question 5

Zooming of an image is done in two different methods, such as nearest neighbor, and bilinear interpolation.

1. Nearest Neighbor Method

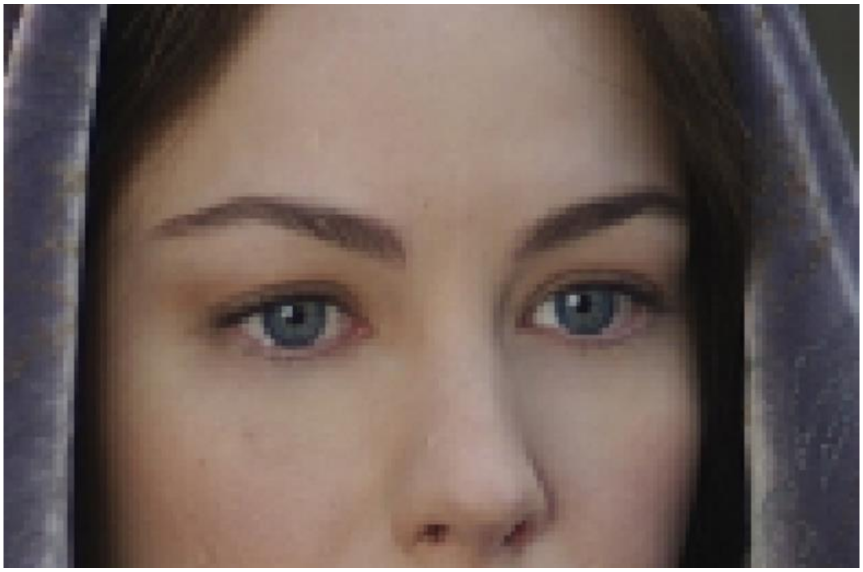
In the nearest neighbor method, the pixel value of the zoomed image will be equal to the value of the pixel, which is most close to it. The code for this calculation is as follows



```
for i in range(z_rows):
    for j in range(z_cols):
        x = min(int(round(i/factor))-1,np.shape(woman)[0])
        y = min(int(round(j/factor))-1,np.shape(woman)[1])
        zoomed[i,j,:] = woman[x,y,:]
```

Using two nested for loops, the values the nearest value in the “woman” image is mapped to the pixels of the “zoomed” image. Round function is used to make it to the nearest value. Minimum value is taken, as the rounded value can be higher than the image size, in the bottom and right edges of the image.

A cropped part of the output is given in below to get an idea on how the image is visible, for a factor of 4. It is quite pixelated and not that smooth.

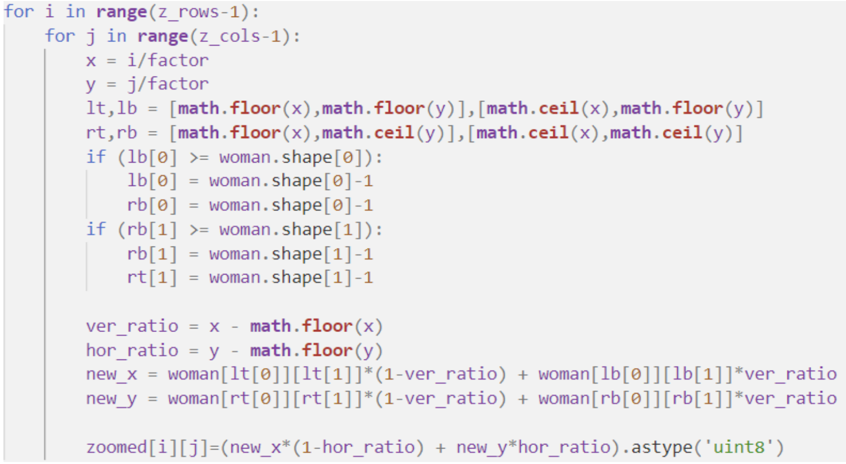


SSD value for im01 - 21.622020816188034

SSD value for im02 - 11.322993528053551

2. Bilinear Interpolation Method

In this method, a linear interpolation will be done to get the values of the pixels. So, a pixel in the zoomed image will have a value, which will be a value in the original image corresponding pixel values around it, something between the minimum and the maximum values. The code for this is rather complex than the nearest neighbor method.



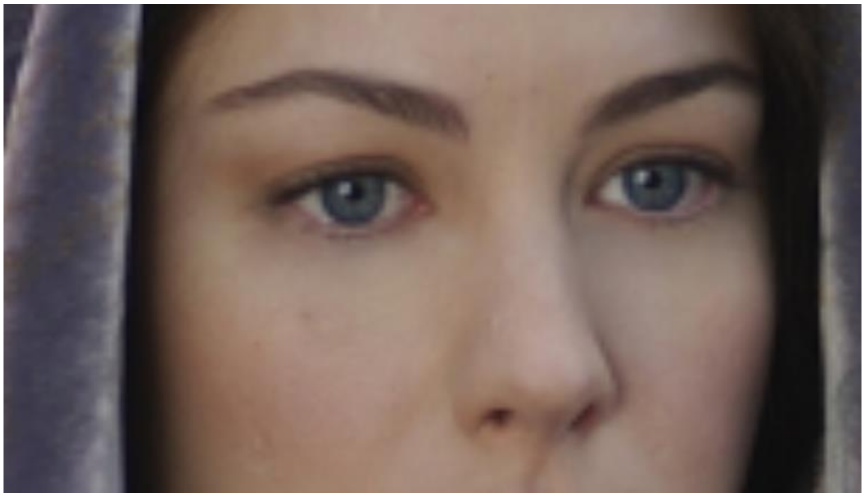
```
for i in range(z_rows-1):
    for j in range(z_cols-1):
        x = i/factor
        y = j/factor
        lt,lb = [math.floor(x),math.floor(y)],[math.ceil(x),math.floor(y)]
        rt,rb = [math.floor(x),math.ceil(y)],[math.ceil(x),math.ceil(y)]
        if (lb[0] >= woman.shape[0]):
            lb[0] = woman.shape[0]-1
            rb[0] = woman.shape[0]-1
        if (rb[1] >= woman.shape[1]):
            rb[1] = woman.shape[1]-1
            rt[1] = woman.shape[1]-1

        ver_ratio = x - math.floor(x)
        hor_ratio = y - math.floor(y)
        new_x = woman[lt[0]][lt[1]]*(1-ver_ratio) + woman[lb[0]][lb[1]]*ver_ratio
        new_y = woman[rt[0]][rt[1]]*(1-ver_ratio) + woman[rb[0]][rb[1]]*ver_ratio

        zoomed[i][j]=(new_x*(1-hor_ratio) + new_y*hor_ratio).astype('uint8')
```

The surrounding pixels of the original image is taken by `math.floor(x)` and `math.ceil(x)` functions. The two if conditions are used to stop the overflowing of the values in the bottom and right corners of the image when taking the ceil. The corresponding intensity value for the pixel is taken using the vertical and horizontal ratios as shown in the above code.

A cropped part of the output is given in below to get an idea on how the image is visible, for a factor of 4. It is less pixelated looking and the transition between the pixels are smooth than the nearest neighbor method and looks clearer.



SSD value for im01 - 6.271345506851507

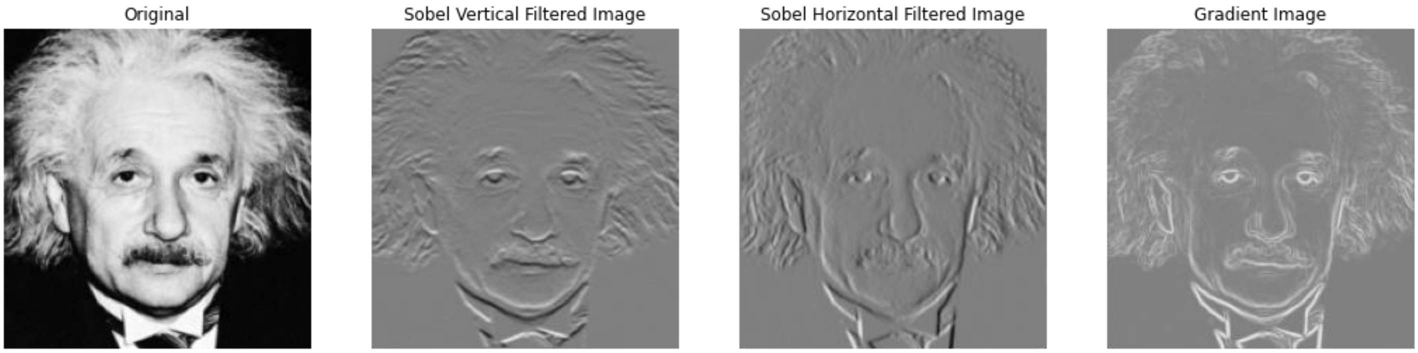
SSD value for im02 - 4.042958359599928

By comparing the two SSD values in the two methods, it can be seen that the bilinear interpolation method gives rather lower error than the nearest neighbor method. It can be observed visually as well. The error calculation was not possible to be done to im03, as the given zoomed image had a different shape (1459, 2400, 3) than the manually zoomed image (1460, 2400, 3)

Question 6

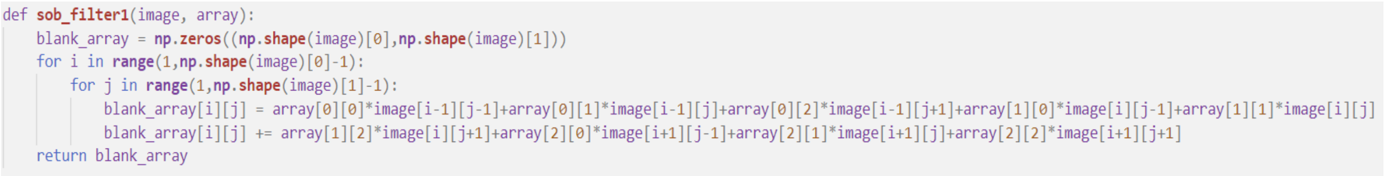
The Sobel operator is used in image processing for edge detection. In a mathematical point of view, what happens is, the image is convolved with a 3x3 matrix. There are two such matrices in Sobel operation, and they are used for vertical and horizontal edge detection. In the Sobel filter, the main difference from the Prewitt filter is that the middle pixel has been prioritized. The Sobel vertical filter is used to detect horizontal edges, while the Sobel horizontal filter is used to detect vertical edges. By taking the magnitude matrix of the outputs of these two filters, we can get the gradient of the image.

In the first part of the question, the OpenCV inbuilt function `cv.filter2D()` was used for getting the outputs of the filters. Sobel vertical and horizontal matrices were defined appropriately. The outputs were as follows


Original
Sobel Vertical Filtered Image
Sobel Horizontal Filtered Image
Gradient Image

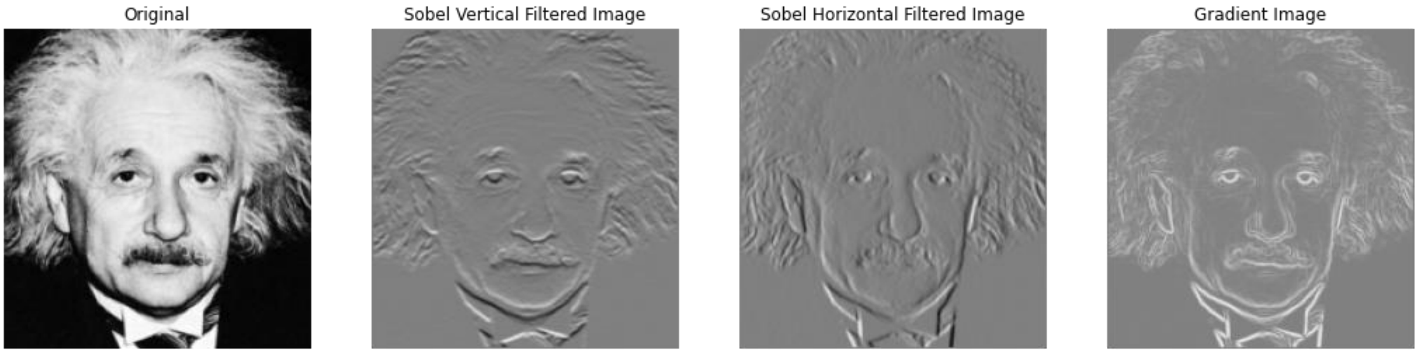
It can be clearly seen that the Sobel vertical and the Sobel horizontal filters have shown the horizontal and vertical edges respectively. The gradient of the image has given a clear outline of the image.

In the second part, the inbuilt function was not allowed to use, so a function was defined for it.

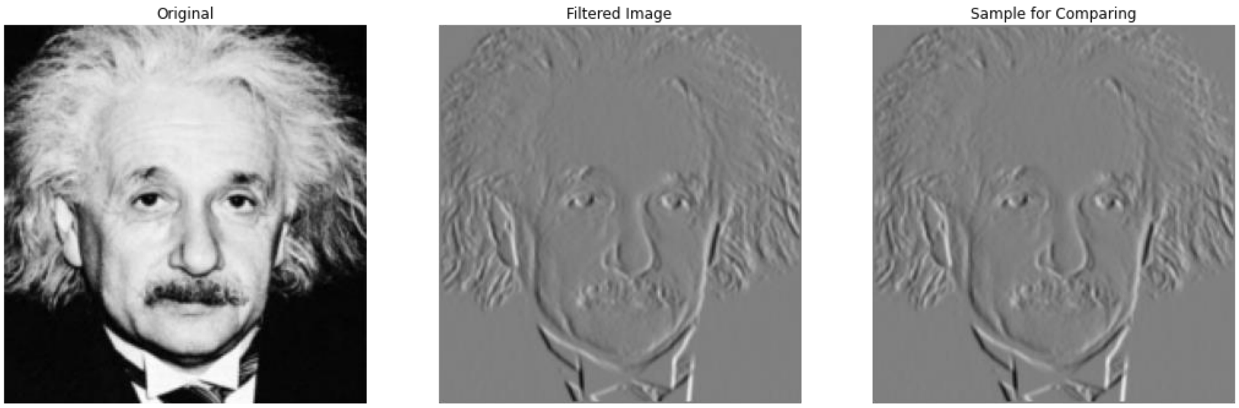


```
def sob_filter1(image, array):
    blank_array = np.zeros((np.shape(image)[0],np.shape(image)[1]))
    for i in range(1,np.shape(image)[0]-1):
        for j in range(1,np.shape(image)[1]-1):
            blank_array[i][j] = array[0][0]*image[i-1][j-1]+array[0][1]*image[i-1][j]+array[0][2]*image[i-1][j+1]+array[1][0]*image[i][j-1]+array[1][1]*image[i][j]
            blank_array[i][j] += array[1][2]*image[i][j+1]+array[2][0]*image[i+1][j-1]+array[2][1]*image[i+1][j]+array[2][2]*image[i+1][j+1]
    return blank_array
```

The inputs to this function will be the image, and the 3x3 kernel. Inside the two nested for loops, the convolution is carried out manually. As the array inputs, the previously defined Sobel vertical and Sobel horizontal filters will be given. The outputs of this manually coded Sobel filters and the result we get from the inbuilt function are both pretty much equal.

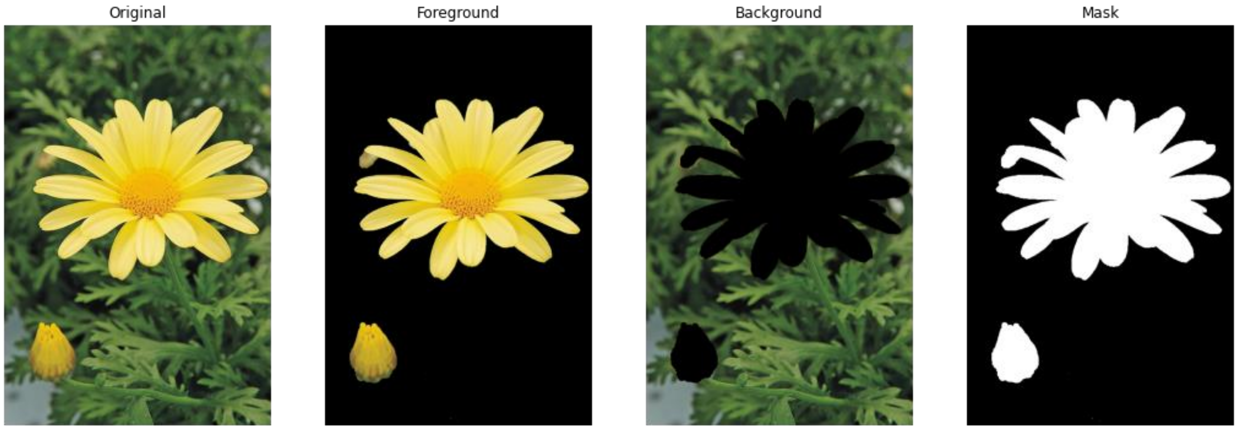

Original
Sobel Vertical Filtered Image
Sobel Horizontal Filtered Image
Gradient Image

In the third part of the question, a filter, which is not one of the defined Sobel vertical or Sobel horizontal filters, but a filter which is closely related to Sobel vertical filter. Although on the previous part, the image was convolved by a 3x3 matrix, here it has been convolved with a 1x3 and a 3x1 matrix separately. If we compare the result, we get by convolving it from the 3x3 matrix, and the two matrices separately, both the outputs are identical. By that, we can confirm that the convolution is associative.

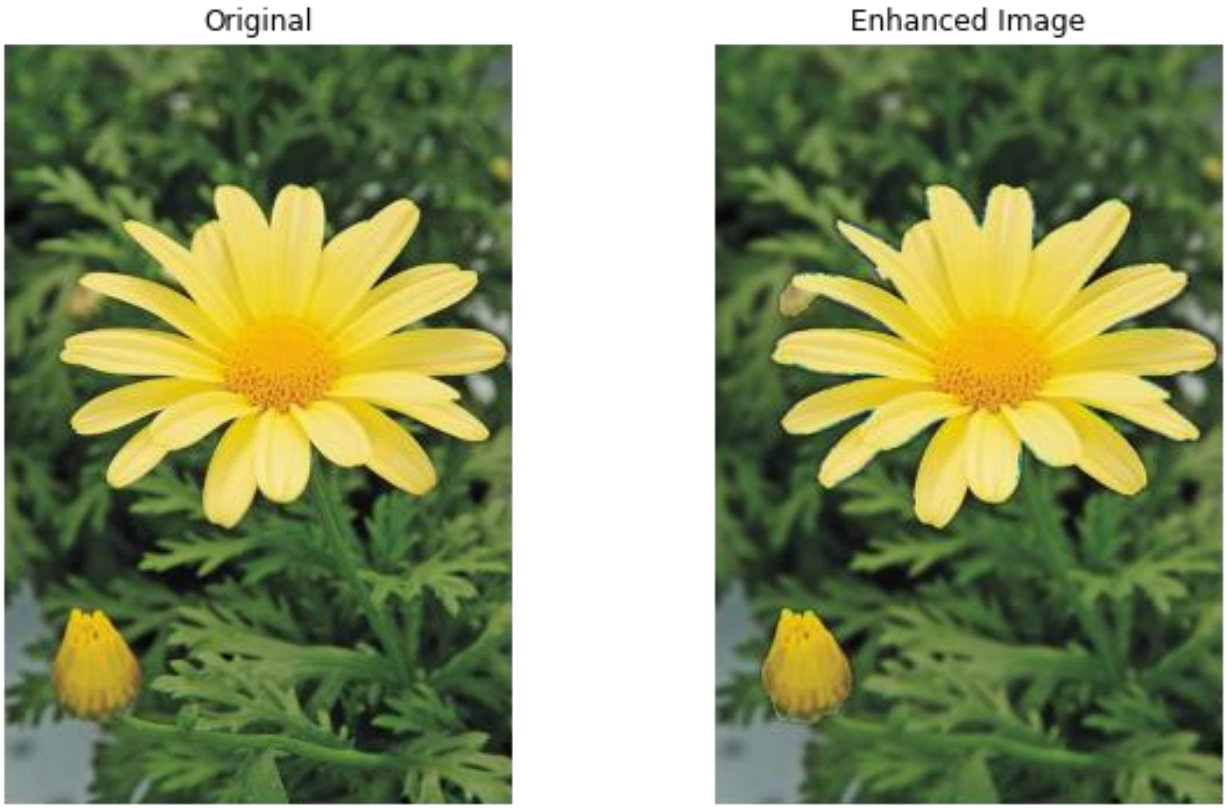


Question 7

Grabcut is an algorithm which was developed for foreground extraction by judging the color distribution, with the use of a Gaussian Mixture Model (GMM). The required area for consideration is given as a rectangle coordinate to the function as an input. The background model and the foreground model are given as arrays of zeros. These are arrays used by the algorithm internally. A mask is also given, to indicate which part of the image should be the foreground, and which part should be the background. The number of time the loop should run is also given as an input to the `cv.grabCut()` function.



After getting the mask, foreground, and the background, the background image is blurred using the `cv.GaussianBlur()` function. To get the enhanced image, the blurred background image is added with the foreground image. Then we get an output where the background is blurred, and the flower is in focus.



If we observe the enhanced image well, it can be seen that the borders of the flower have been distorted and become dark than the original. In the background image, the edges of the flower which is cut out is sharp. But, when you do the gaussian blurring, the blurring will happen in the black area as well, and as a result of that, the greener area will sort of combine with the black area at the edge of the flower. So, once you add the blurred background image to the foreground image of the flower, the distorted edge will add with the yellow edge of the flower, resulting a darker edge.

Github Link

https://github.com/akilaabeyratne