Intensity Transformations and Neighborhood Filtering

Assignment 1

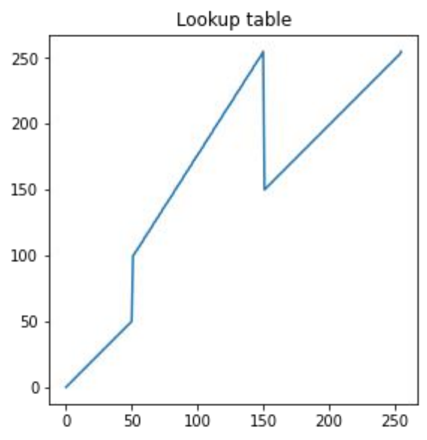
Biyon Fernando - 190178J

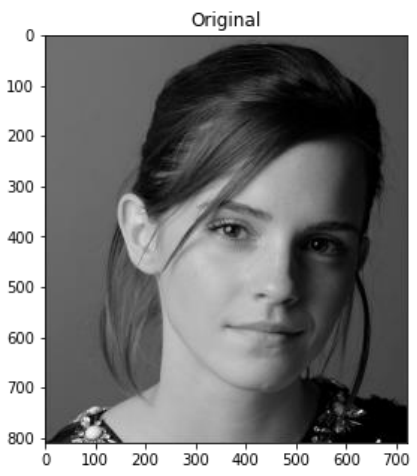
March 1, 2022

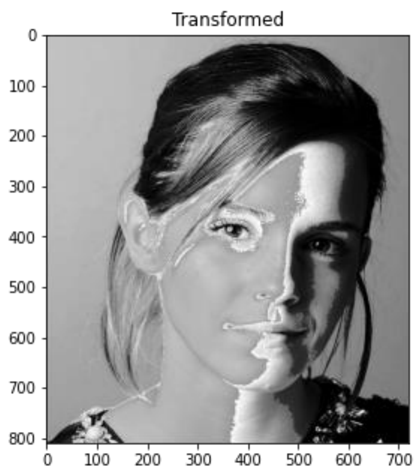
GitHub Link

Question 1

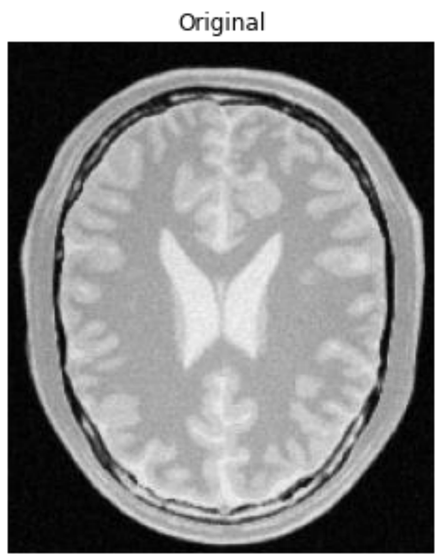
To achieve intensity, transform for the image given a lookup table method is used. As the pixel values are in uint8 type those values can be directly used as indexes to access mapped values in the lookup table. Using a NumPy array of 256 elements functionality of the lookup table can be achieved..





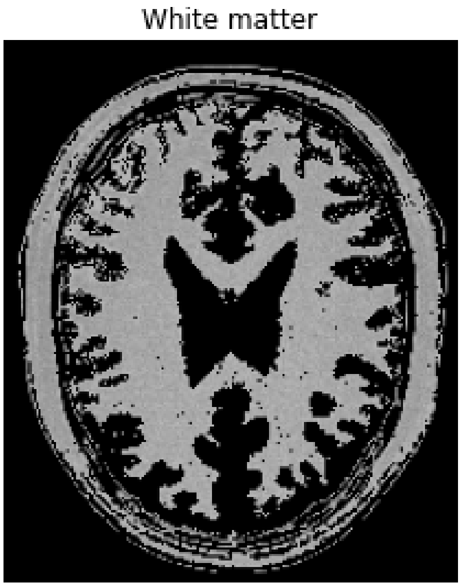


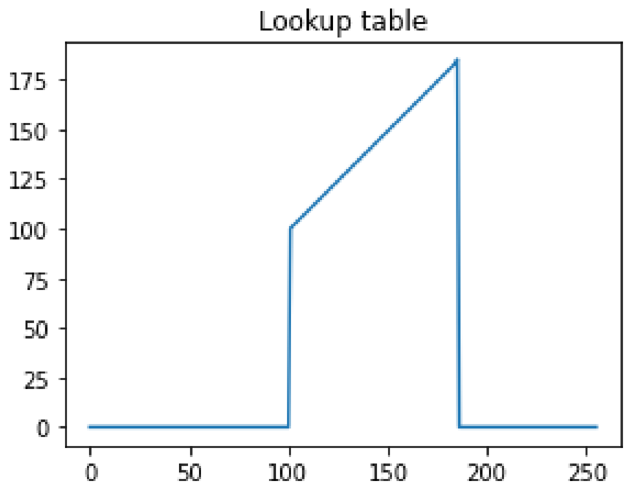
Question 2

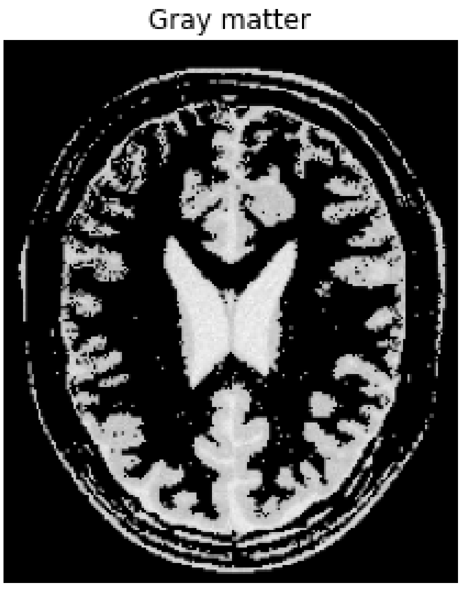


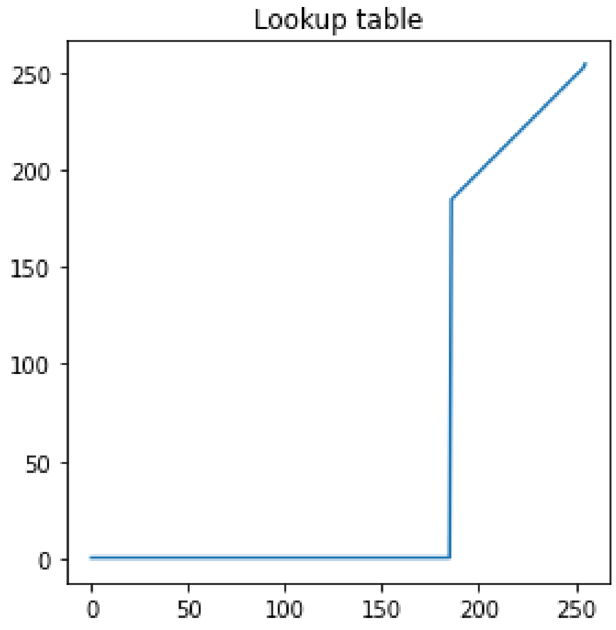
In the question, we were given a proton density image of the brain. These images show gray color for the white matter and white color for the gray matter. Therefore, we can attenuate white color and dark colors using point function to accentuate the gray color and attenuate gray colors and dark colors to accentuate white color as below.

Here I have preserved the original variations of the white and gray matter areas.







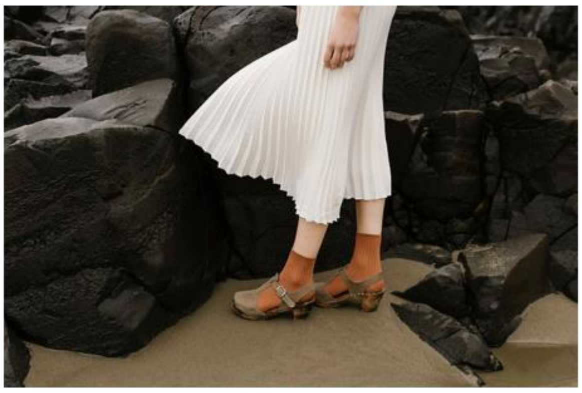


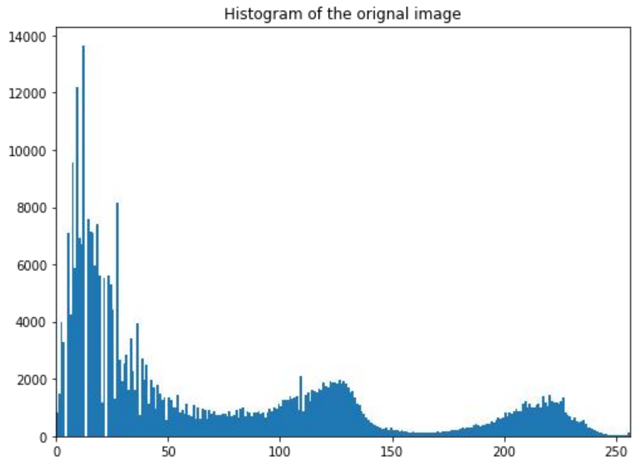
Question 3

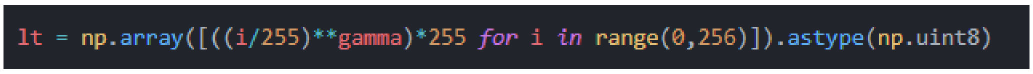
In L*a*b color space, we can manipulate perceptual lightness and red, green, blue, and yellow four unique colors of human vision. L axis contains the lightness factor. Applying gamma correction changes the lightness of the image. For $\gamma < 1$ higher lightness can observe while $\gamma > 1$ shows lower lightness.

The original image and its histogram are shown below.

original

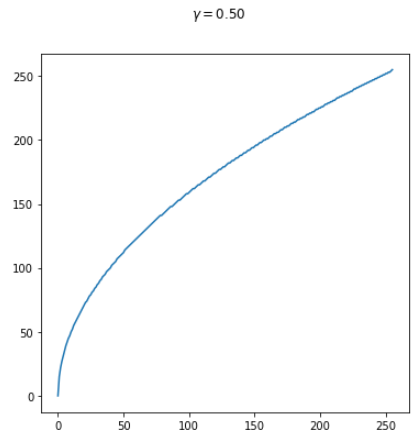






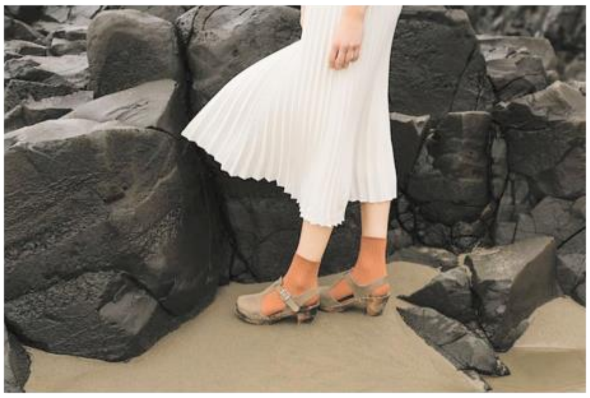
```
lt = np.array([((i/255)**gamma)*255 for i in range(0,256)]).astype(np.uint8)
```

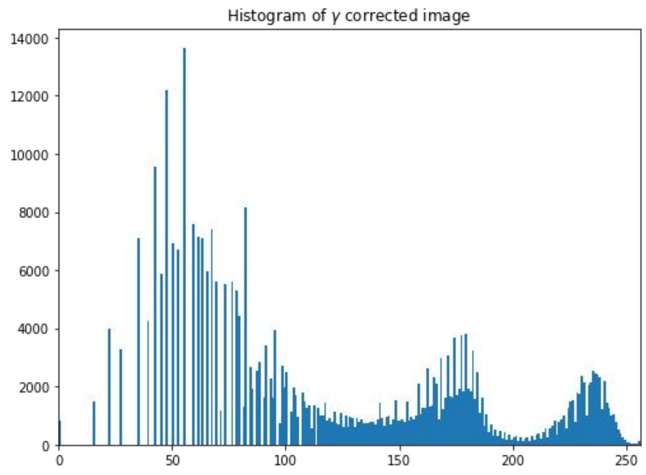
Gamma correction is done using a lookup table as above. This lookup table can visualize as below.



After Applying this correction, we can observe the output image and its histogram as below.

$\gamma = 0.50$





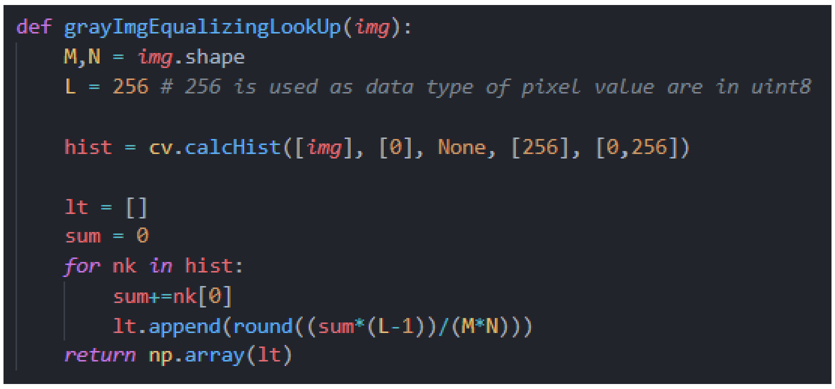
Question 4

We can derive pointwise intensity transform for the equalized image using a basic probability thermos. The equation of the transform is given below

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j),$$
$$= \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j \quad k = 0, 1, \ldots, L-1.$$

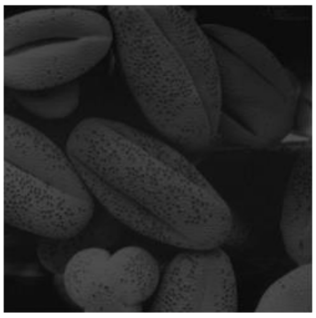
(Reference – lecture slides)

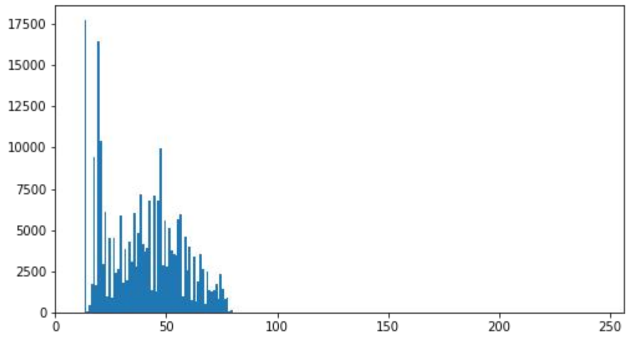
A python function can write according to this equation as below to return a lookup table for transformation.

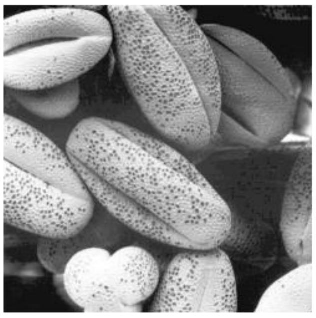


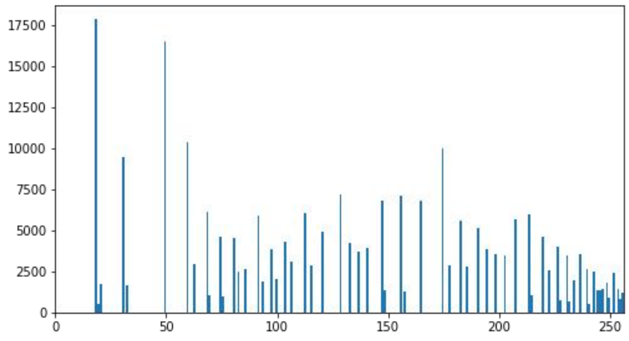
```python
def grayImgEqualizingLookUp(img):
    M,N = img.shape
    L = 256 # 256 is used as data type of pixel value are in uint8

    hist = cv.calcHist([img], [0], None, [256], [0,256])

    lt = []
    sum = 0
    for nk in hist:
        sum+=nk[0]
        lt.append(round((sum*(L-1))/(M*N)))
    return np.array(lt)
```

After applying the equalizing transform to a gray scale image, we can observe enhancement of colors as bellow. And the equalized property of colors can significantly observe by comparing original and equalized images histograms.



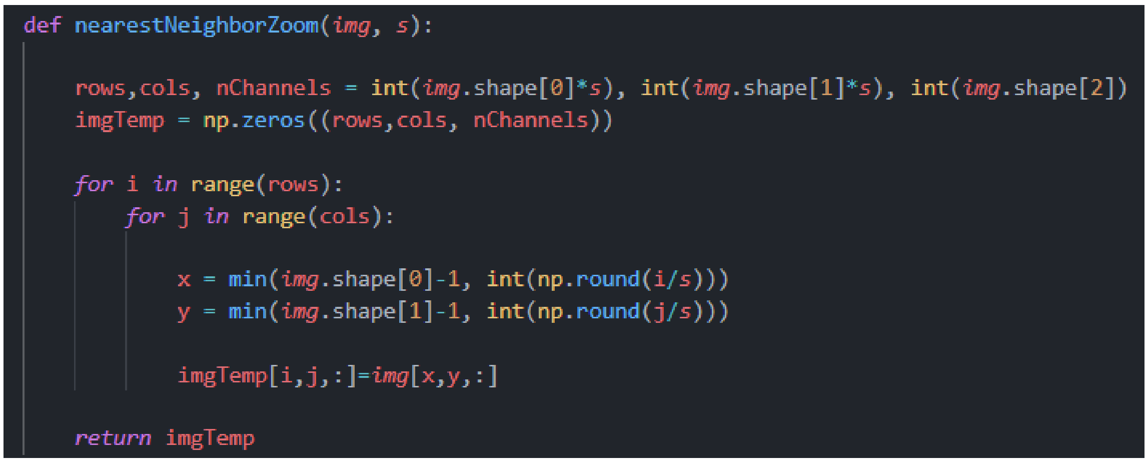






Question 5

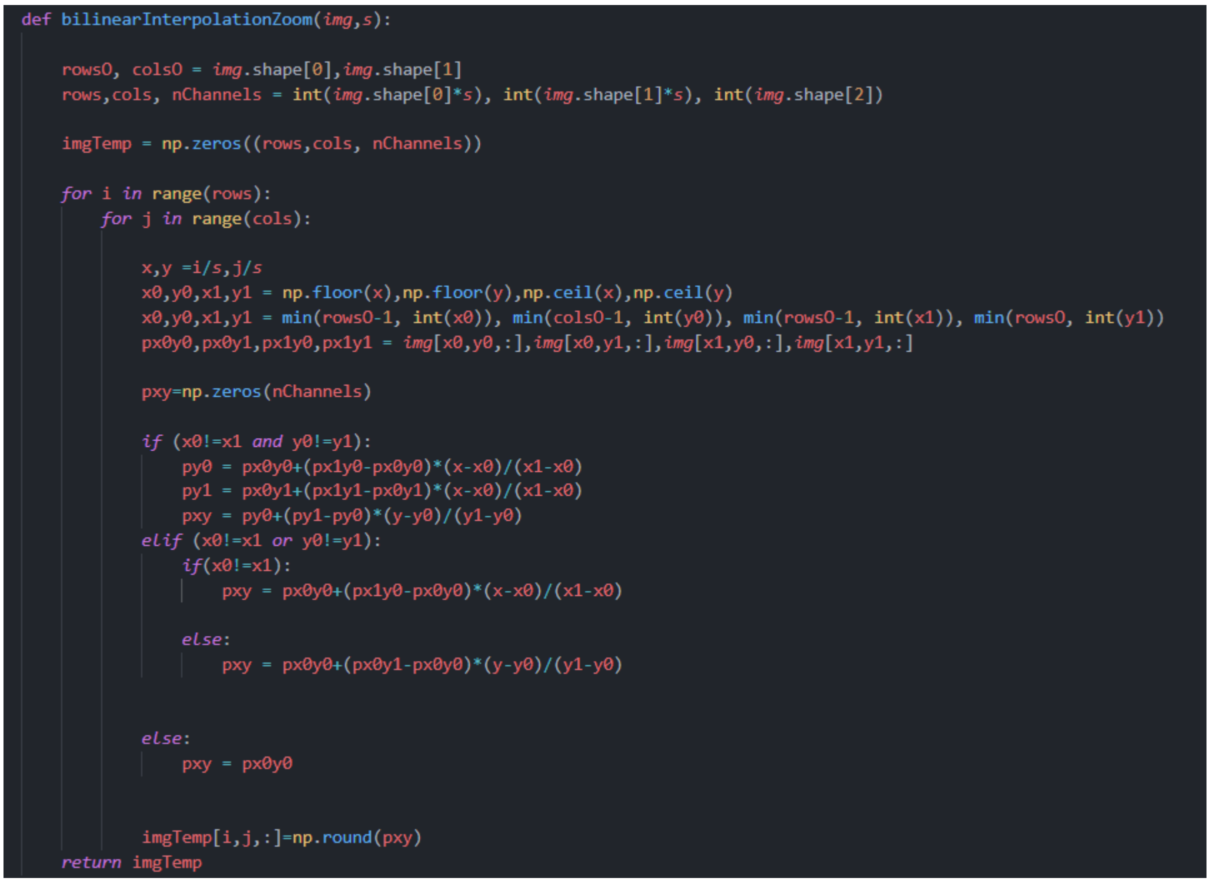
In nearest-neighbor zoom method uses pixel values of the nearest position to the corresponding coordinates in the original image to the zoomed image.



```python
def nearestNeighborZoom(img, s):

    rows,cols, nChannels = int(img.shape[0]*s), int(img.shape[1]*s), int(img.shape[2])
    imgTemp = np.zeros((rows,cols, nChannels))

    for i in range(rows):
        for j in range(cols):

            x = min(img.shape[0]-1, int(np.round(i/s)))
            y = min(img.shape[1]-1, int(np.round(j/s)))

            imgTemp[i,j,:]=img[x,y,:]

    return imgTemp
```

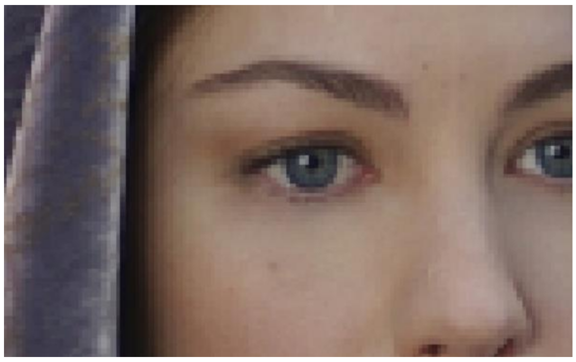
To remove the edge problems original image shape is used as a threshold.

In the bilinear interpolation method, we interpolate pixel values for the exact coordinate using values in the neighbor pixel values.

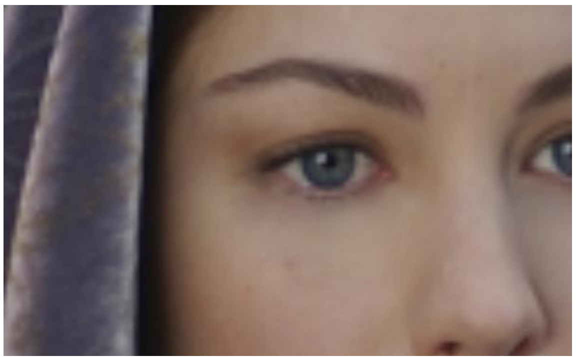


```python
def bilinearInterpolationZoom(img,s):

    rows0, cols0 = img.shape[0],img.shape[1]
    rows,cols, nChannels = int(img.shape[0]*s), int(img.shape[1]*s), int(img.shape[2])

    imgTemp = np.zeros((rows,cols, nChannels))

    for i in range(rows):
        for j in range(cols):

            x,y =i/s,j/s
            x0,y0,x1,y1 = np.floor(x),np.floor(y),np.ceil(x),np.ceil(y)
            x0,y0,x1,y1 = min(rows0-1, int(x0)), min(cols0-1, int(y0)), min(rows0-1, int(x1)), min(rows0, int(y1))
            px0y0,px0y1,px1y0,px1y1 = img[x0,y0,:],img[x0,y1,:],img[x1,y0,:],img[x1,y1,:]

            pxy=np.zeros(nChannels)

            if (x0!=x1 and y0!=y1):
                py0 = px0y0+(px1y0-px0y0)*(x-x0)/(x1-x0)
                py1 = px0y1+(px1y1-px0y1)*(x-x0)/(x1-x0)
                pxy = py0+(py1-py0)*(y-y0)/(y1-y0)
            elif (x0!=x1 or y0!=y1):
                if(x0!=x1):
                    pxy = px0y0+(px1y0-px0y0)*(x-x0)/(x1-x0)

                else:
                    pxy = px0y0+(px0y1-px0y0)*(y-y0)/(y1-y0)


            else:
                pxy = px0y0


            imgTemp[i,j,:]=np.round(pxy)
    return imgTemp
```

After zooming the image shown in right 4 times, using nearest-neighbor and bilinear interpolation algorithms bellow results were able to achieve. Among the two of these algorithms, the bilinear interpolation method gives a more smooth intensity distribution in neighbor pixels.



A Cropped part of the zoom image from nearest-neighbor method



A Cropped part of the zoom image from bilinear interpolation method

To calculate the normalized sum of squared difference (SSD) following equation is used.

```
error = np.sum((img1-img2)**2)/(img2.shape[0]*img2.shape[1]*3)
```

Normalized SSD values for each algorithm

Image number	nearestNeighborZoom	bilinearInterpolationZoom
01	40.1117	37.7171
02	16.7930	16.1221

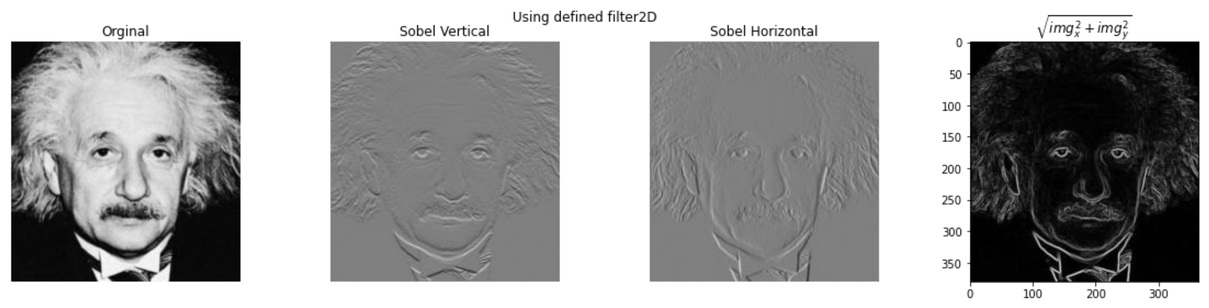
To interpret the above value, we can use the square root values of these SSD values. For the above-applied algorithms square root value was around 4-6. As we can observe bilinear interpolation method give less error than the nearest neighbor method.

Question 6

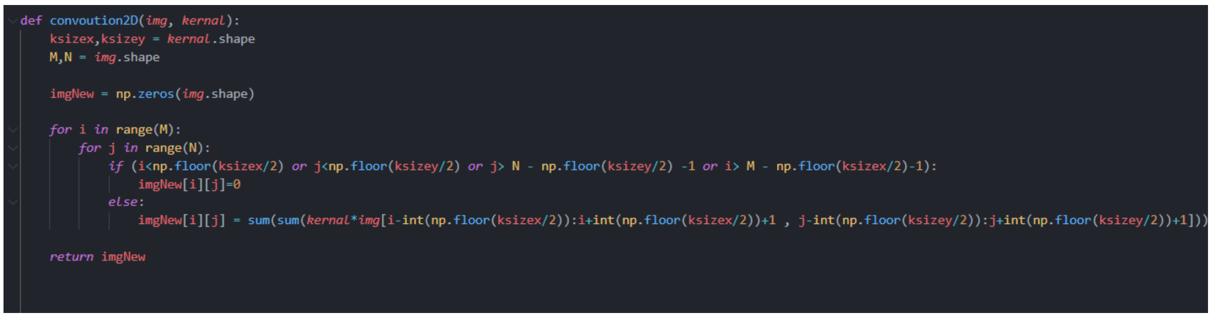
a)

```
xDir =  np.array([ ( -1 , -2 , -1 ) , ( 0 , 0 , 0 ) , ( 1 , 2 , 1 ) ] , dtype = np.float32 )
yDir =  np.array([ ( -1 , 0 , 1 ) , ( -2 , 0 , 2 ) , ( -1 , 0 , 1 ) ] , dtype = np.float32 )
```

Sobel kernels shown above are used.

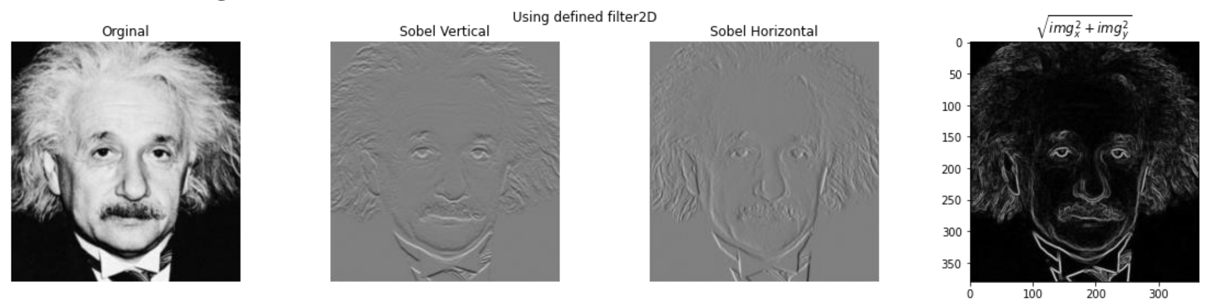


b)

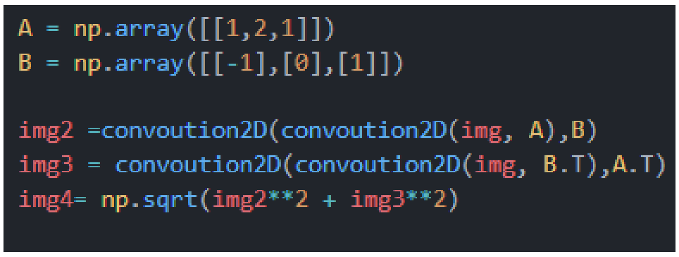


```
def convoution2D(img, kernal):
    ksizex,ksizey = kernal.shape
    M,N = img.shape

    imgNew = np.zeros(img.shape)

    for i in range(M):
        for j in range(N):
            if (i<np.floor(ksizex/2) or j<np.floor(ksizey/2) or j> N - np.floor(ksizey/2) -1 or i> M - np.floor(ksizex/2)-1):
                imgNew[i][j]=0
            else:
                imgNew[i][j] = sum(sum(kernal*img[i-int(np.floor(ksizex/2)):i+int(np.floor(ksizex/2))+1 , j-int(np.floor(ksizey/2)):j+int(np.floor(ksizey/2))+1]))

    return imgNew
```

The above code gives the same results as filter2D.

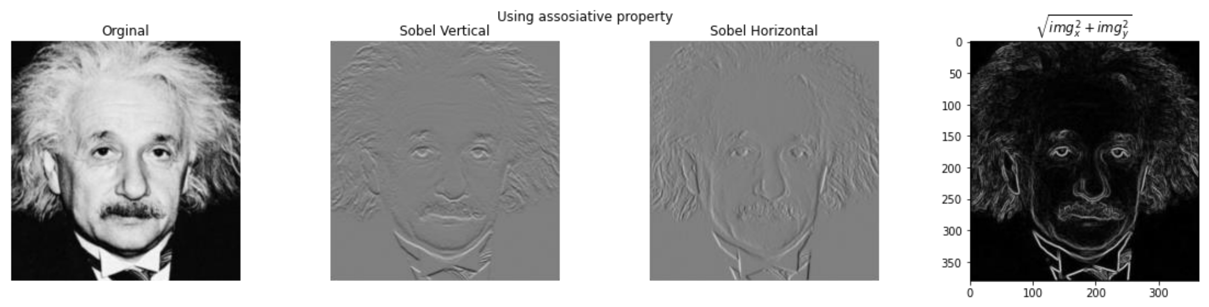


c)



```
A = np.array([[1,2,1]])
B = np.array([[-1],[0],[1]])

img2 =convoution2D(convoution2D(img, A),B)
img3 = convoution2D(convoution2D(img, B.T),A.T)
img4= np.sqrt(img2**2 + img3**2)
```

To apply the same effect as above Sobel filters these A and B 1D matrixes were convolved with the image. This is possible because of the associativity property of convolution. The results obtained are given below.

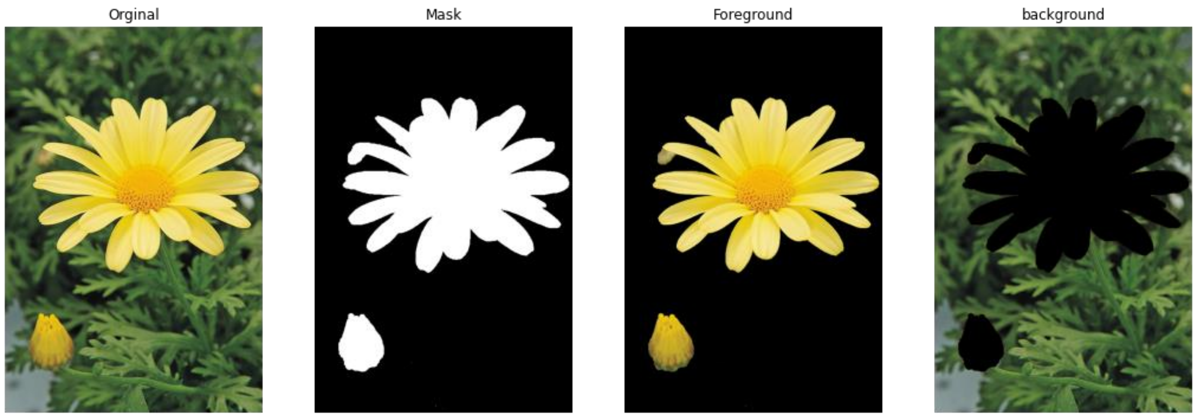


Question 7

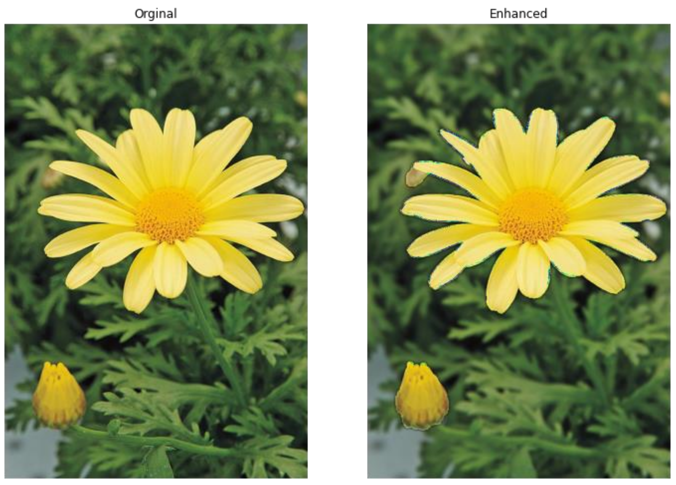
GrabCut algorithm is capable of extracting the foreground and background of the image by labeling image pixels with its color statistics, edge detection, and data provided by the user as rectangle or mask.

a)

segmentation mask, foreground image, and background image generated by the algorithm given below.



b)



By adding a gaussian blur to the background image and combining it with the foreground image we can enhance the image.

c)

As the background image has a black area for the foreground parts, convolution results give lower values when we do the gaussian blurring around those edge areas therefore we can observe quite dark in the enhanced image just beyond the edge of the flower.