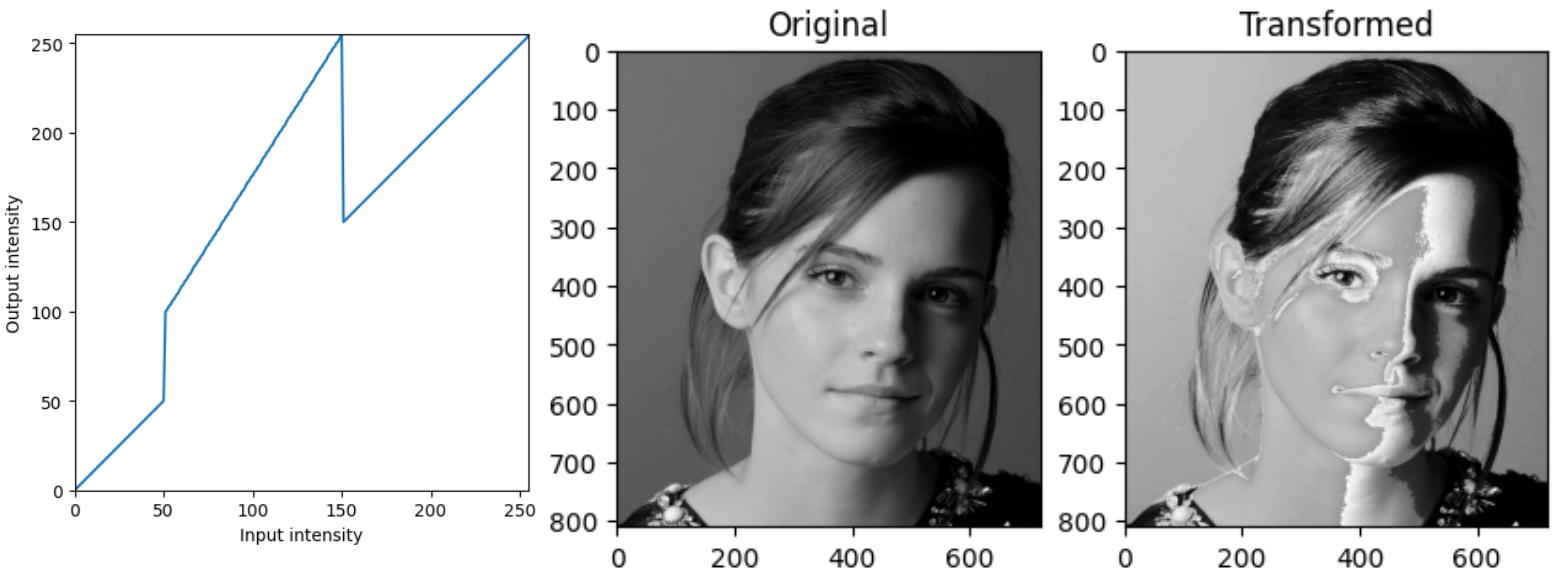
A.D.U.Dilhara
200128D

[github link]

Q1). Transformation function is as follow

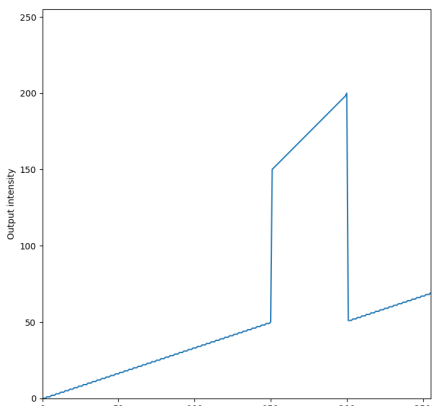
```
points = np.array([(50,50),(50,100),(150,255),(150,150)])
t1 = np.linspace(0,points[0,1],points[0,0]+1).astype('uint8')
print(len(t1))
t2 = np.linspace(points[1,1],points[2,1],points[2,0]-points[1,0]).astype('uint8')
print(len(t2))
t3 = np.linspace(points[3,1],255,255-points[3,0]).astype('uint8')
print(len(t3))
transform = np.concatenate((t1,t2),axis=0)
transform = np.concatenate((transform,t3),axis =0)
```



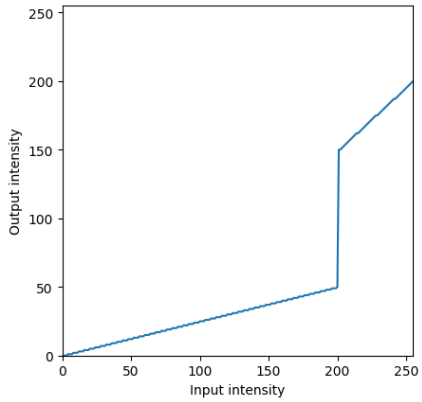
Q2).
To enhance the visibility of white matter, the transformation process has been tailored to ensure that the resulting image contains pure white in regions that were originally very close to white. This approach is intended to improve visualization, and the specific grayscale range chosen for this enhancement is 200-255.

In order to emphasize gray matter, the transformation has been devised to render regions originally characterized as gray with an absolute white appearance in the output image. This adjustment aims to enhance visualization, and the chosen grayscale range for this purpose is 50-200.

Transformation for the gray matter



Transformation for the white matter



Q3)

The gamma correction only needs to be applied to the L plane in the L*a*b* color space. This was done using the following segment of code.

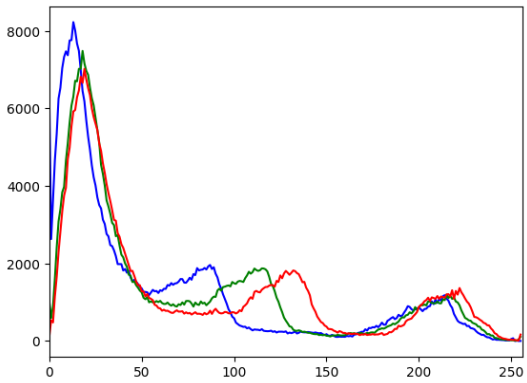
The gamma value 0.45 is selected to obtain a spread-out histogram for the transformed image.

```
image = cv.imread("highlights_and_shadows.jpg",1)
gamma =0.45
f = lambda x: min(((x/255)**gamma*255),255)
#table = np.array([(i/255)**gamma*255 for i in np.arange(0,256)]).astype('uint8')

image_Lab = cv.cvtColor(image, cv.COLOR_BGR2Lab)

imshow("Original image",image)
cv.imwrite("Q3_Original image.jpg",image)


# Split the HSV image into its components
L, a, b = cv.split(image_Lab)

L_trans= np.round(np.vectorize(f)(L)).astype('uint8')
```

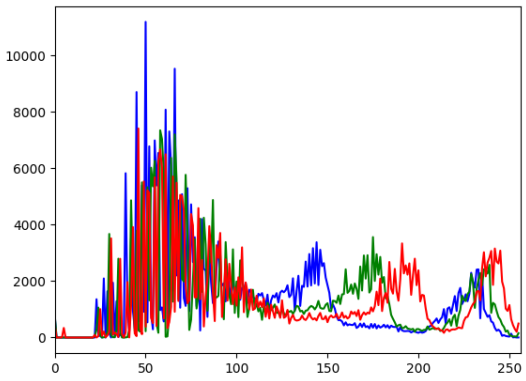
Code for Lab color space

Before transformation

After transformation



Before transformation

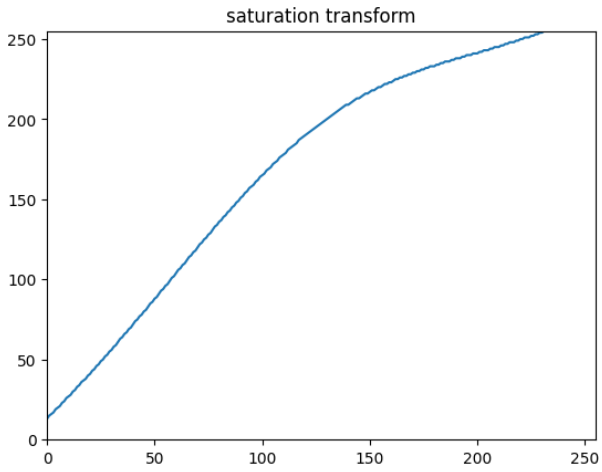


After transformation

Q4)

The intensity transformation only needs to be applied to the S plane in the HSV color space.This was done using the following segment of code.

```
alpha=0.55

f=lambda x: min(x + alpha*128*math.exp(-(x-128)**2/(2*70**2)),255)

def imshow(name, img1):
    cv.imshow(name,img1)
    cv.waitKey(0)
    cv.destroyAllWindows()

image = cv.imread("spider.png",1)

image_hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)

imshow("Original image",image)
cv.imwrite("Q4_Original image.jpg",image)
#imshow("HSV image",image_hsv)

# Split the HSV image into its components
hue, saturation, value = cv.split(image_hsv)
#imshow("Hue plane",hue)
#imshow("Saturation palne",saturation)
#imshow("Value plane", value)

saturation_trans= np.round(np.vectorize(f)(saturation)).astype('uint8')
#print(saturation)
#print(saturation_trans)
```



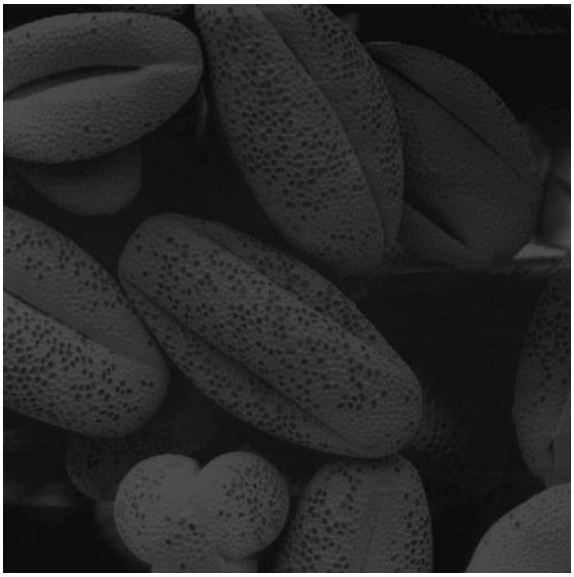
Before Saturation transformed

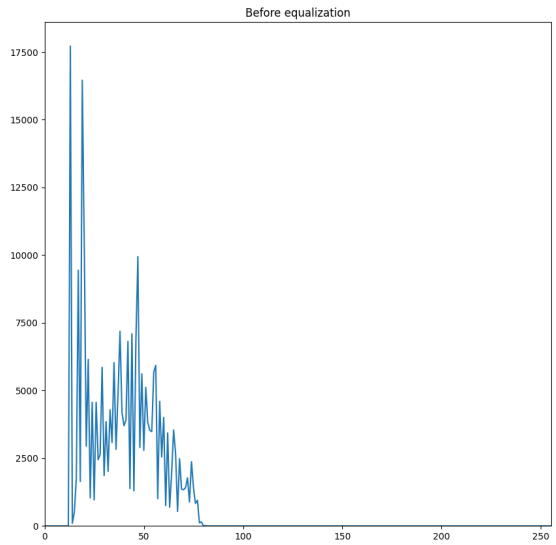
After Saturation transformed

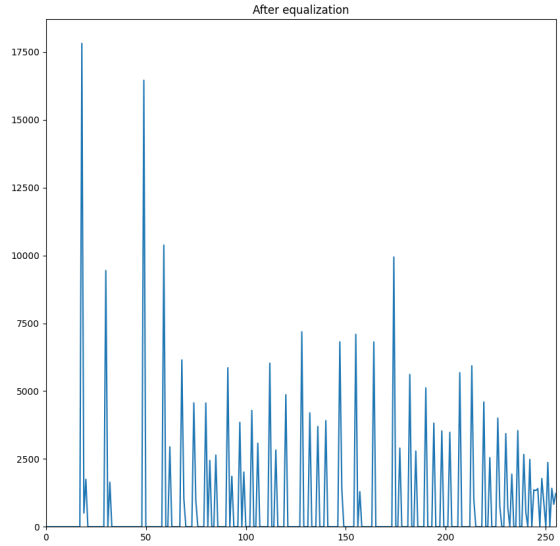
Q5)

```python
def graylevel_eq(hist):
    f1 = lambda x: x/sum(hist)
    pdf= np.vectorize(f1)(hist)
    cdf = np.zeros(len(hist))

    for i in range(len(hist)):
        cdf[i] = sum(pdf[0:i+1])

    f2 = lambda x: x*(len(hist)-1)
    graylevel_new = np.vectorize(f2)(cdf)
    graylevel_new = np.round(graylevel_new).astype('uint8')
    return graylevel_new

image_ori = cv.imread("shells.tif", 0)
histogram = cv.calcHist([image_ori],[0],None,[256],[0,256])

equalized_graylevel = graylevel_eq(histogram)
image_eq = cv.LUT(image_ori,equalized_graylevel)
```

Function for histogram equalization







Q6)

Hue Plane

Saturation Plane

Value Plane

The mask

Original Image

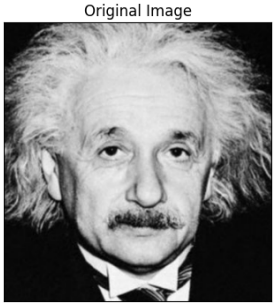
Possessed image

Q7)

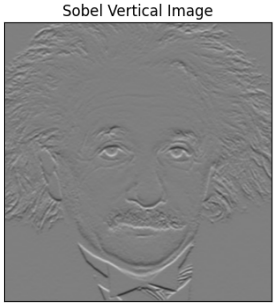
Using the filter2D function

```
sobel_ver = np.array([[-1,-2,-1],[0,0,0],[1,2,1]], dtype=np.float32)
img_x = cv.filter2D(img6, -1, sobel_ver)

sobel_hor = np.array([[-1,0,1],[-2,0,2],[-1,0,1]], dtype=np.float32)
img_y = cv.filter2D(img6, -1, sobel_hor)

img_grad = np.sqrt(img_x**2 + img_y**2)
```

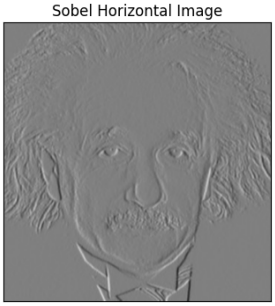
Original Image



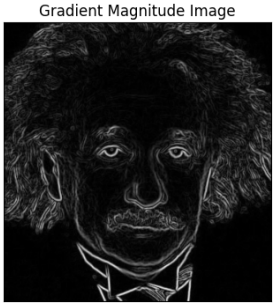
Sobel Vertical Image



Sobel Horizontal Image



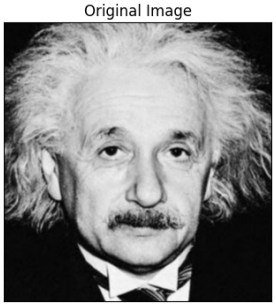
Gradient Magnitude Image



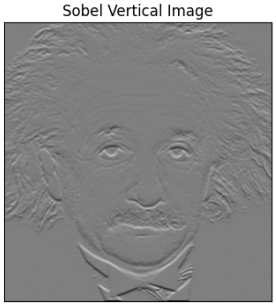
Using own function

```
padding = 0
padded = np.full((rows + 2, columns + 2), padding, dtype=np.uint8)

padded[1:rows + 1, 1:columns + 1] = img6_m

for i in range(rows):
    for j in range(columns):
        img_ym[i,j] = np.sum(np.multiply(sobel_hor_m, padded[i:i + 3, j:j + 3]))

for i in range(rows):
    for j in range(columns):
        img_xm[i,j] = np.sum(np.multiply(sobel_ver_m, padded[i:i + 3, j:j + 3]))
```

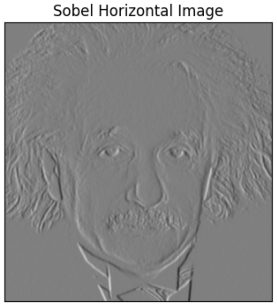
Original Image



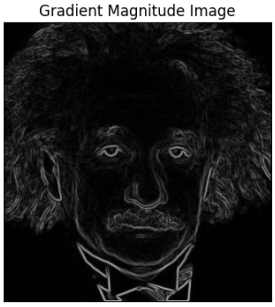
Sobel Vertical Image



Sobel Horizontal Image



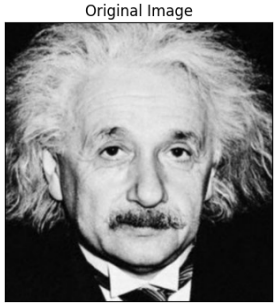
Gradient Magnitude Image



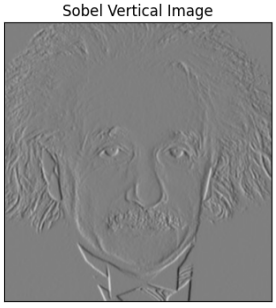
Using Associative Property

```
sobel_ver_array1 = np.array([[1],[2],[1]])
sobel_ver_array2 = np.array([[1,0,-1]])

sobel_hor_array1 = np.array([[1],[0],[-1]])
sobel_hor_array2 = np.array([[1,2,1]])

img_xp_1 = sig.convolve2d(img6_p, sobel_ver_array1, mode="same")
img_xp = sig.convolve2d(img_xp_1, sobel_ver_array2, mode="same")
img_yp_1 = sig.convolve2d(img6_p, sobel_hor_array1, mode="same")
img_yp = sig.convolve2d(img_yp_1, sobel_hor_array2, mode="same")

img_grad_p = np.sqrt(img_xp**2 + img_yp**2)
```

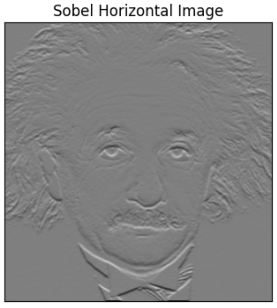
Original Image



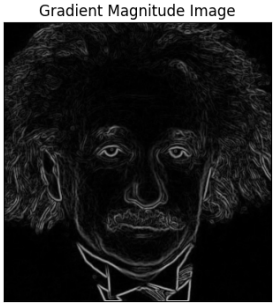
Sobel Vertical Image



Sobel Horizontal Image



Gradient Magnitude Image



Q8)

The assignment images folder contained small images that were utilized for the purpose of zooming. These images were then compared with the zoomed versions provided. Following the given instructions, the zooming operation was performed on the images using the OpenCV library's resize function, which employed two distinct algorithms for the zooming process. Zooming factor is 4.

1. Nearest Neighbor
2. Bilinear Interpolation

```python
def nearest_neighbor(img, zoom):
    rows = img.shape[0]*zoom
    colums = img.shape[1]*zoom
    planes = img.shape[2]
    zoomed = np.zeros((rows,colums,planes), dtype =np.uint8)
    for i in range(0,rows):
        for j in range(0,colums):
                zoomed[i,j] =  img[int(i/zoom),int(j/zoom)]
    return zoomed
```

Function for Nearest neighbor

```python
def zoom_bilinear_interpolation( image,scale=4):
    rows = int(scale*image.shape[0])
    columns = int(scale*image.shape[1])
    zoomed = cv.resize(image,(columns,rows),interpolation = cv.INTER_LINEAR)
    return zoomed
```

Function for Bilinear Interpolation

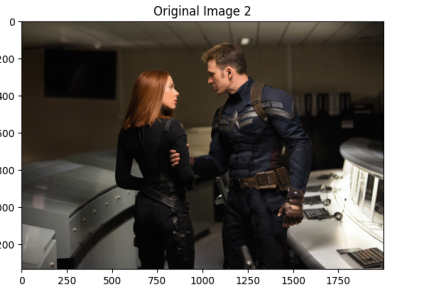
```python
ssd_1 = np.sum(((image[:,:]-nearest_zoomed[:,:])**2)/(3*255**2))/(image.shape[0]*image.shape[1])
ssd_2 = np.sum(((image[:,:]-bilinear_zoomed[:,:])**2)/(3*255**2))/(image.shape[0]*image.shape[1])

print(ssd_1)
print(ssd_2)
```

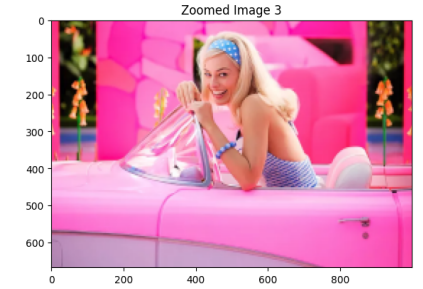
✓ 7.4s

```
0.0004811121335890121
0.0004775562417232688
```

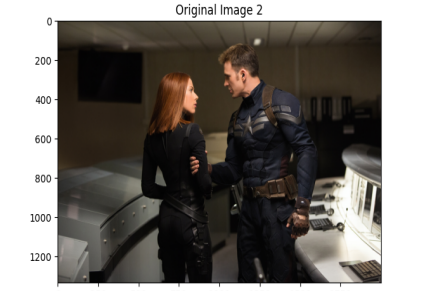
Compute SSD

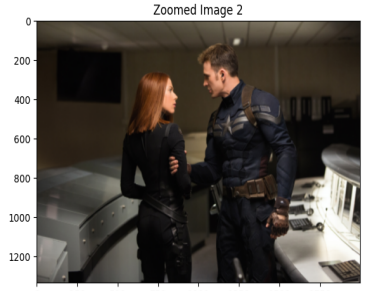
Using Nearest Neighbor method

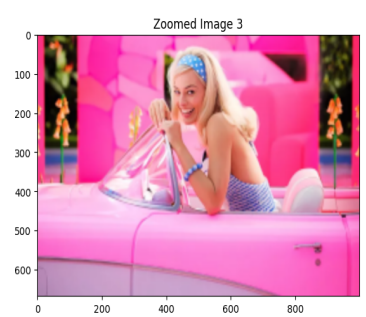




Using Bilinear Interpolation method







Q9)To isolate the foreground and background elements within the images, a process was employed where a bitwise AND operation was applied between the masks and the original images. This operation resulted in the separation of the desired foreground and background components from the overall images.

```python
mask = np.zeros(img7.shape[:2], dtype="uint8") # Mask - an empty image to store the mask
rect = (40, 150, 560, 500) # Defining the rectangle that contains the object

fgModel = np.zeros((1, 65), dtype="float")
bgModel = np.zeros((1, 65), dtype="float")

# Apply GrabCut using the the bounding box segmentation method
(mask, bgModel, fgModel) = cv.grabCut(img7, mask, rect, bgModel, fgModel, 5, mode=cv.GC_INIT_WITH_RECT)

# Mask for the foreground
fore_mask = (mask == cv.GC_PR_FGD).astype("uint8") * 255
outputMask_fore = np.where((mask == cv.GC_BGD) | (mask == cv.GC_PR_BGD), 0, 1)
outputMask_fore = (outputMask_fore * 255).astype("uint8")
foreground = cv.bitwise_and(img7, img7, mask=outputMask_fore) # Foreground Image

# Mask for the background
back_mask = (mask == cv.GC_PR_BGD).astype("uint8") * 255
outputMask_back = np.where((mask == cv.GC_FGD) | (mask == cv.GC_PR_FGD), 0, 1)
outputMask_back = (outputMask_back * 255).astype("uint8")
background = cv.bitwise_and(img7, img7, mask=outputMask_back) # Background Image
```

Original Image

Mask

Foreground

Background

Original Image

Enhanced Image

During the execution of the grabCut process, certain pixels near the image edges might exhibit characteristics of both the foreground and background. Consequently, when these two images are combined, these edge pixels are summed together, potentially exceeding the maximum value of 255. As a result, they tend to appear darker in the final result due to the clipping effect that limits values to the maximum brightness level.