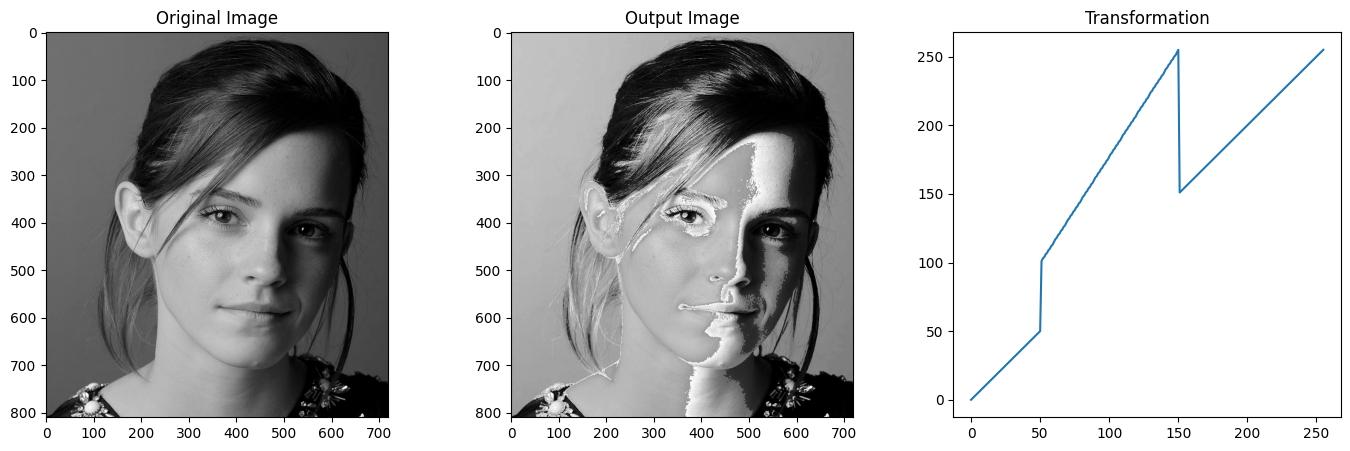
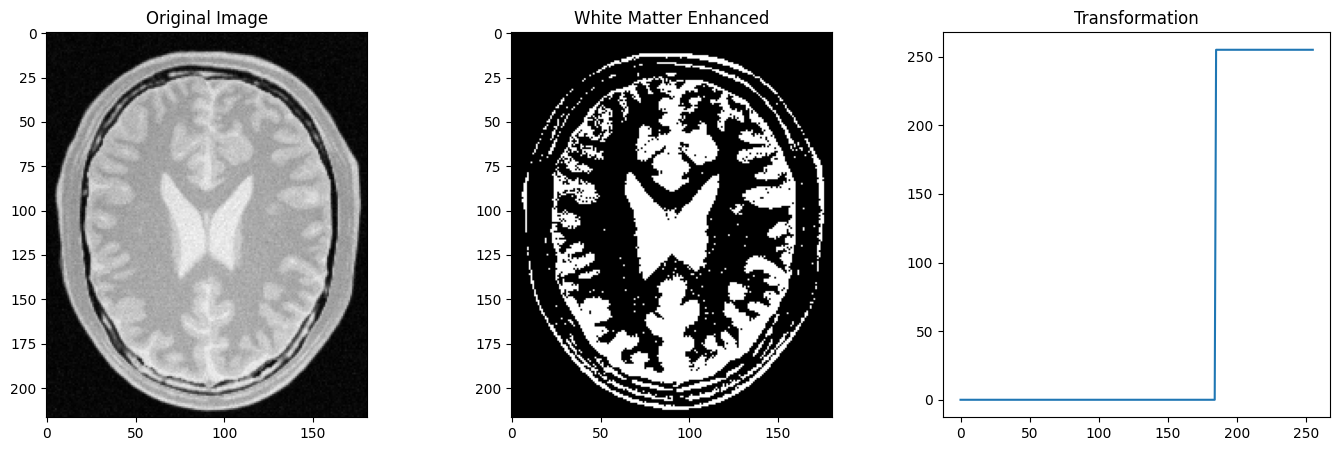
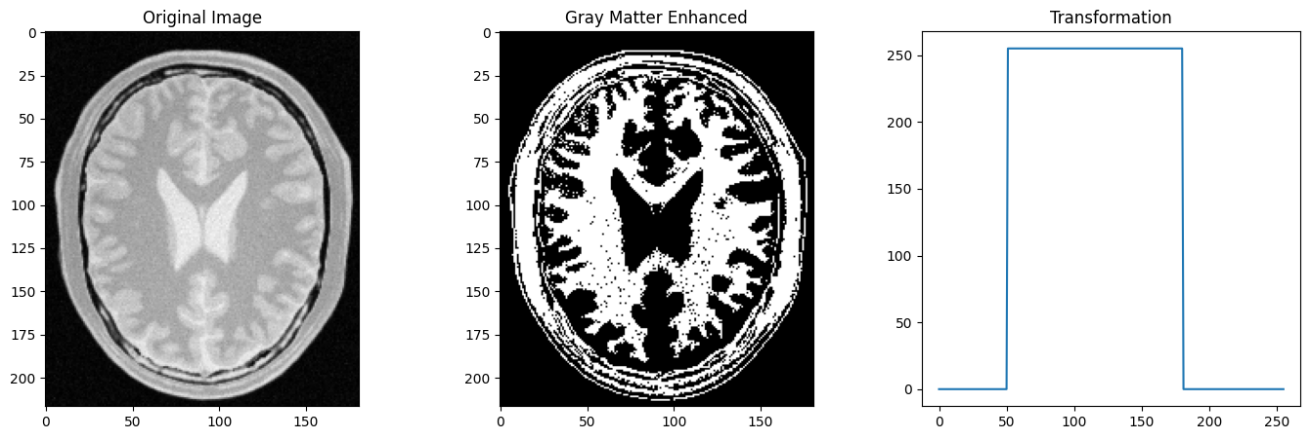
**Only the necessary parts of the codes are included. (No imread,imshow functions are showed)**

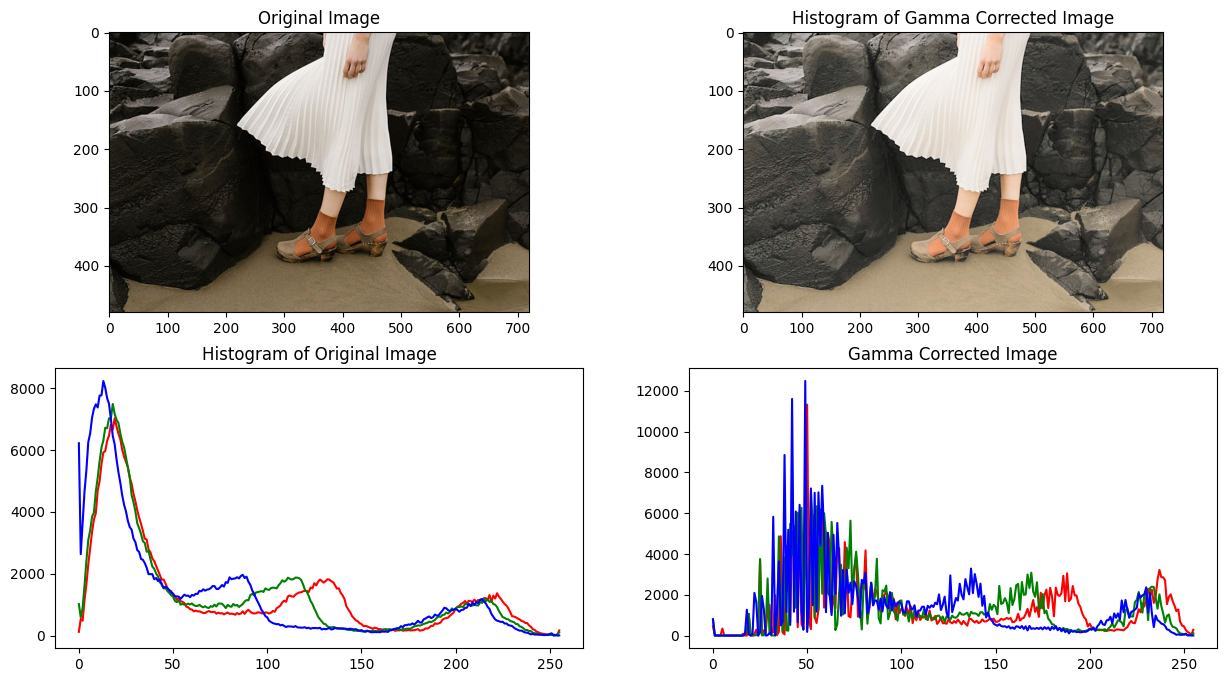
**Question 01**

import cv2 as cv  
import numpy as np  
import matplotlib.pyplot as plt  
  
coordinates = np.array([(0,0),(50,50),(50,100),(150,250),(150,150),(255,255)])  
trans = np.array([0])  
for i in range(int(len(coordinates)/2)):  
 line = np.linspace(coordinates[0+2\*i ,1], coordinates[1+2\*i ,1], coordinates[1+2\*i , 0] - coordinates[0+2\*i , 0] + 1)  
 trans = np.concatenate((trans , line)).astype(np.uint8)  
transformed = trans[image1]

***Question 02***

***a)White Matter Enhanced***  
x = np.arange(0,256).astype(np.uint8)  
divert\_point = 184  
coordinates = np.array([(1,0),(divert\_point,0),(divert\_point+1,255),(255,255)])  
trans = np.array([0])  
  
for i in range(int(len(coordinates)/2)):  
 line = np.linspace(coordinates[0+2\*i ,1], coordinates[1+2\*i ,1], coordinates[1+2\*i , 0] - coordinates[0+2\*i , 0] + 1)  
 trans = np.concatenate((trans , line)).astype(np.uint8)  
transformed = trans[image2]

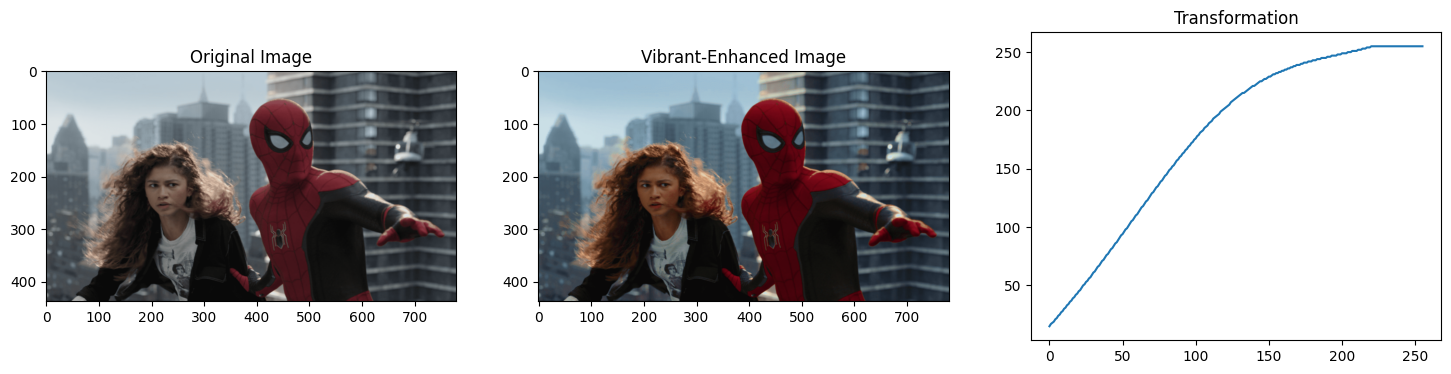
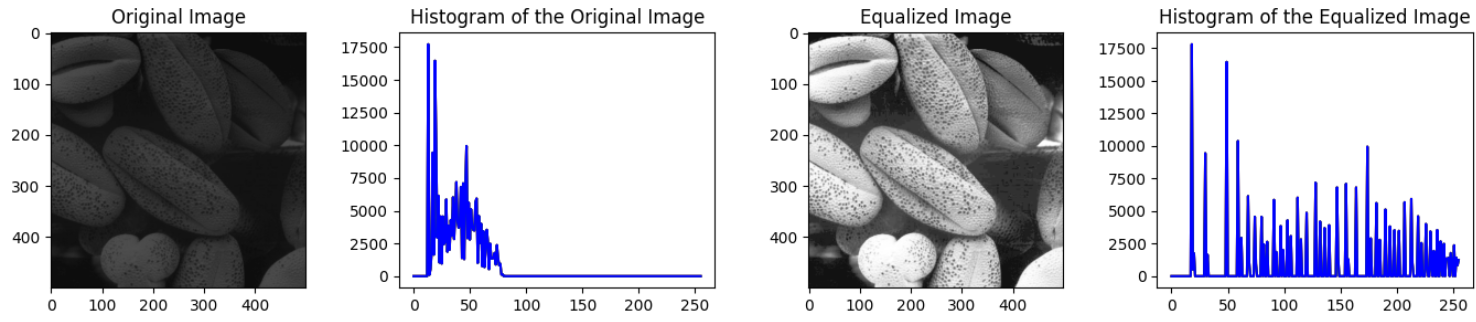
***b)*** ***Gray Matter Enhanced***  
coordinates = np.array([(1,0),( 50,0),( 50+1,255),( 180,255),(180+1,0),(255,0)])  
trans = np.array([0])  
for i in range(int(len(coordinates)/2)):  
 line = np.linspace(coordinates[0+2\*i ,1], coordinates[1+2\*i ,1], coordinates[1+2\*i ,0] - coordinates[0+2\*i , 0] + 1)  
 trans = np.concatenate((trans, line)).astype(np.uint8)  
transformed = trans[image2]

***Question 03***  
image3\_copy = image3.copy()  
image3\_copy = cv.cvtColor(image3\_copy,cv.COLOR\_BGR2RGB)  
image3 = cv.cvtColor(image3, cv.COLOR\_BGR2LAB)  
l\_img , a\_img , b\_img = cv.split(image3)  
  
gamma = 0.5  
  
table = np.array([(i/255.0)\*\*(gamma)\*255.0 for i in np.arange(0,256)]).astype(np.uint8)  
img\_gamma = cv.LUT(l\_img, table)  
final\_image = cv.merge((img\_gamma, a\_img, b\_img))  
final\_image = cv.cvtColor(final\_image, cv.COLOR\_LAB2RGB)

***Discussion***

* L channel : This plane represent the Lightness / Brightness of image.When adding gamma correction with gamma=0.5,it will increase the brightness of the L plane and the final output image.
* A plane : This plane represent the color from green to red. Negative value correspond to green while positive values correspond to red.
* B plane : This plane represent the color from blue to yellow. Negative value correspond to blue while positive values correspond to yellow.

***Question 04***  
hsv\_image = cv.cvtColor(image4, cv.COLOR\_BGR2HSV)  
hue, saturation, value = cv.split(hsv\_image)  
a , alpha = 0.65, 70  
x = np.arange(0,256,1).astype(np.uint8)  
table = np.array([min(255, (x + (a \* 128) \* np.exp(-((x - 128) \*\* 2) / (2 \* alpha \*\* 2)))) for x in np.arange(0, 256)]).astype('uint8')  
sat\_update = cv.LUT(saturation, table)  
merge\_img = cv.merge((hue, sat\_update , value))  
merge\_img = cv.cvtColor(merge\_img, cv.COLOR\_HSV2RGB)

**‘a’ value that output a visually pleasing image : 0.65**  
 ***Question 05***  
height, width = image5.shape[0], image5.shape[1]  
r\_plane, g\_plane, b\_plane = cv.split(image5)  
def intensity\_count(plane) :  
 count\_list = np.zeros(256).astype(np.uint16)  
 for i in range(height) :  
 for j in range(width) :  
 count\_list[plane[i][j]] += 1  
 return count\_list  
def histogram\_eq(plane, intensity\_count) :  
 equalized\_intensities = np.zeros(256).astype(np.uint16)  
 count = 0  
 for i in range(len(intensity\_count)) :  
 count += intensity\_count[i]   
 new\_val = (255/(height\*width)) \* count  
 new\_val = round(new\_val)  
 equalized\_intensities[i] = new\_val  
  
 for h in range(height) :  
 for w in range(width) :  
 plane[h][w] = equalized\_intensities[plane[h][w]]  
 return plane  
  
r\_plane\_count = intensity\_count(r\_plane)  
eq\_r\_plane = histogram\_eq(r\_plane, r\_plane\_count)  
g\_plane\_count = intensity\_count(g\_plane)  
eq\_g\_plane = histogram\_eq(g\_plane, g\_plane\_count)  
b\_plane\_count = intensity\_count(b\_plane)  
eq\_b\_plane = histogram\_eq(b\_plane, b\_plane\_count)  
new\_image = cv.merge((eq\_r\_plane,eq\_g\_plane,eq\_b\_plane))  
  
  
***Question 06  
Part a,b,c***  
h\_plane, s\_plane, v\_plane = cv.split(image6\_hsv)  
th\_val, binary\_image = cv.threshold(s\_plane, 11, 255, cv.THRESH\_BINARY)  
result = cv.bitwise\_and(image6, image6, mask = binary\_image)

***Part d***

hist, bins = np.histogram(result.ravel(),256,[0,256])  
cdf = hist.cumsum()  
cdf\_normalized = cdf \* hist.max() / cdf.max()

***Part e***

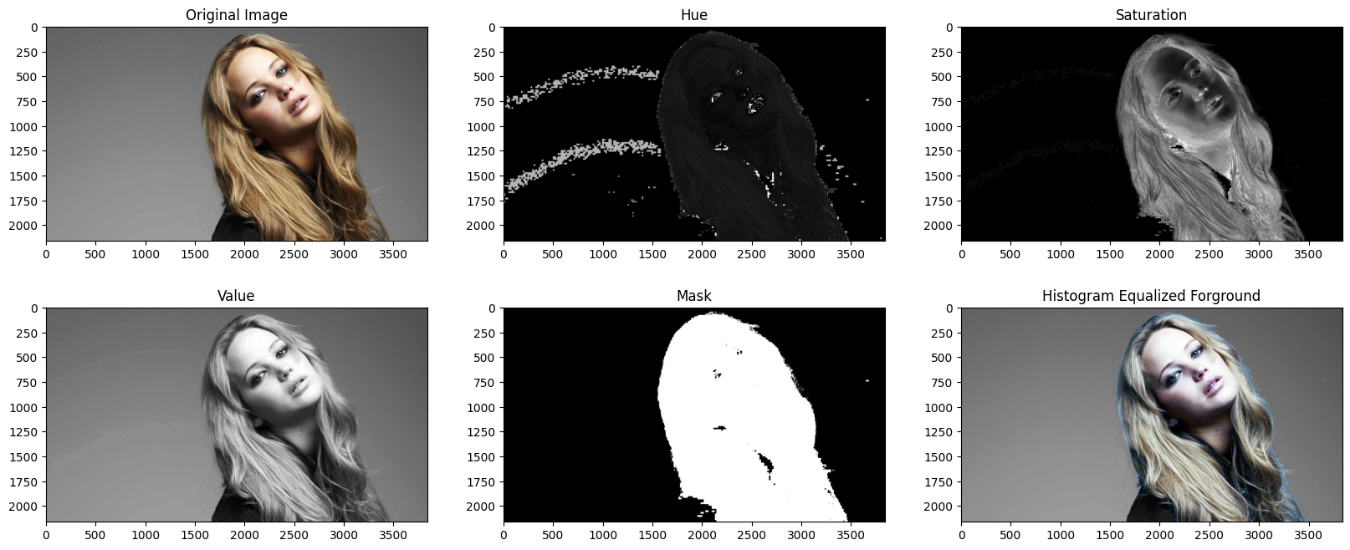
result\_r , result\_g, result\_b = cv.split(result)  
equ\_r, equ\_g, equ\_b = cv.equalizeHist(result\_r), cv.equalizeHist(result\_g), cv.equalizeHist(result\_b)

***Part f***

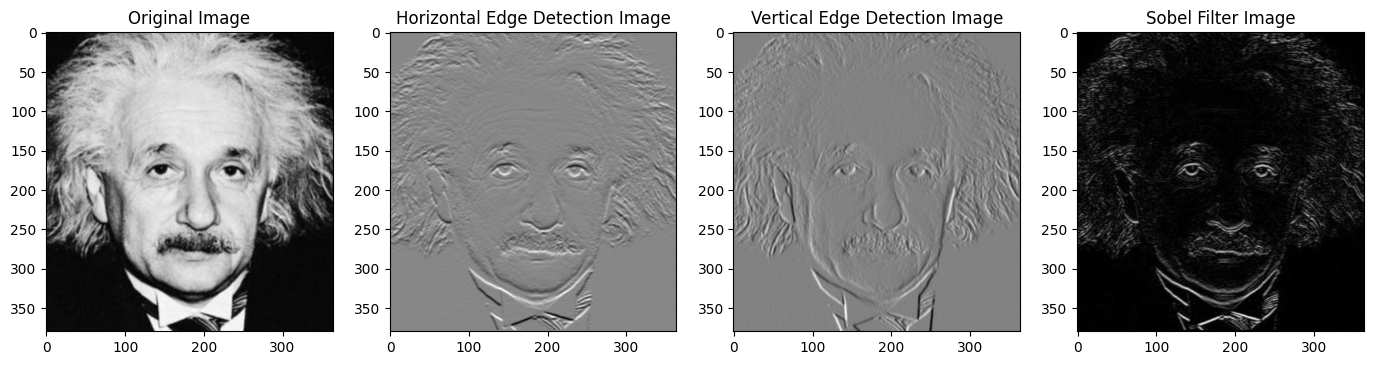
equalized\_result = cv.merge((equ\_r, equ\_g, equ\_b))

backgnd\_mask = cv.bitwise\_not(binary\_image)

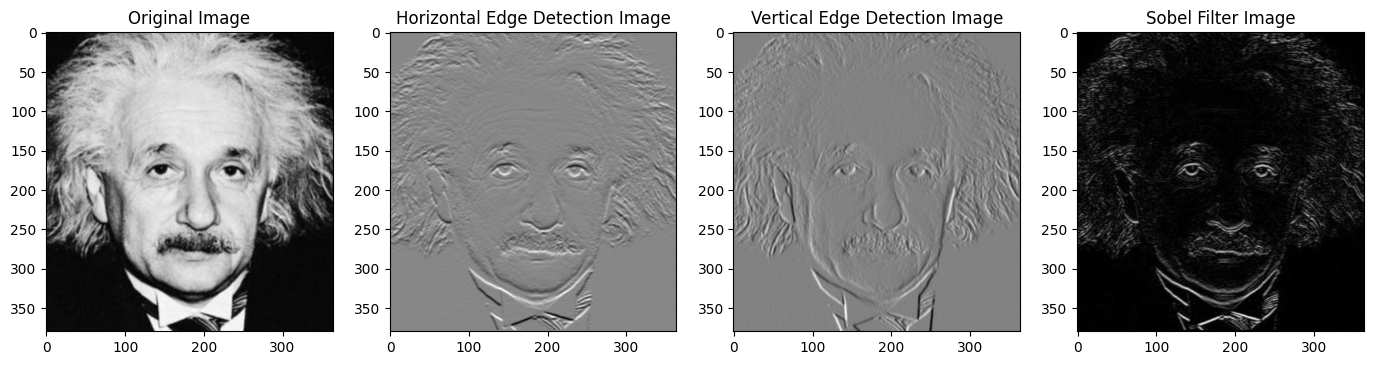
backgnd = cv.bitwise\_and(image6, image6, mask = backgnd\_mask)  
final = cv.add(equalized\_result, backgnd)

  
***Question 07***

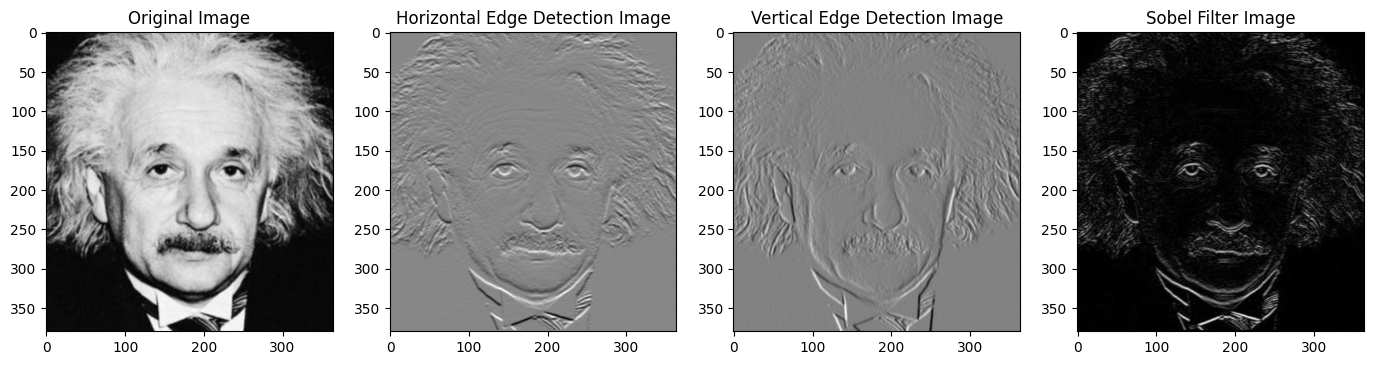
***Part a***

image7 = cv.imread('einstein.png', cv.IMREAD\_GRAYSCALE).astype(np.float32)  
vertical\_sobel = np.array([[-1,-2,-1],[0,0,0],[1,2,1]], dtype = np.float32)  
horizontel\_sobel = np.array([[-1,0,1],[-2,0,2],[-1,0,1]], dtype = np.float32)  
img\_horizontal = cv.filter2D(image7, -1, vertical\_sobel)  
img\_vertical = cv.filter2D(image7, -1, horizontel\_sobel)  
final\_image = np.sqrt(img\_horizontal\*\*2, img\_vertical\*\*2)

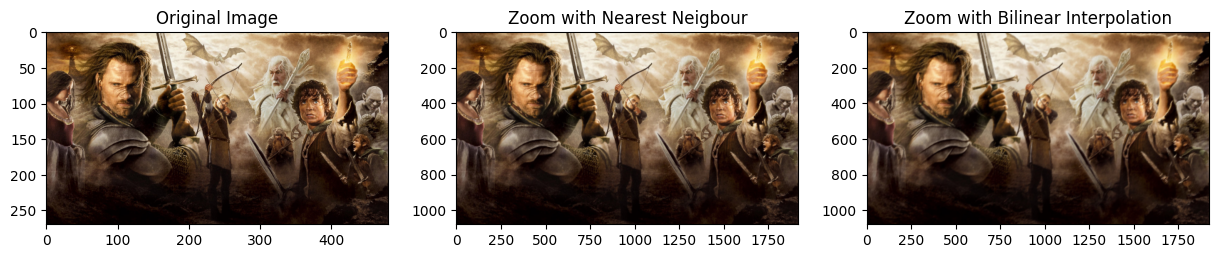
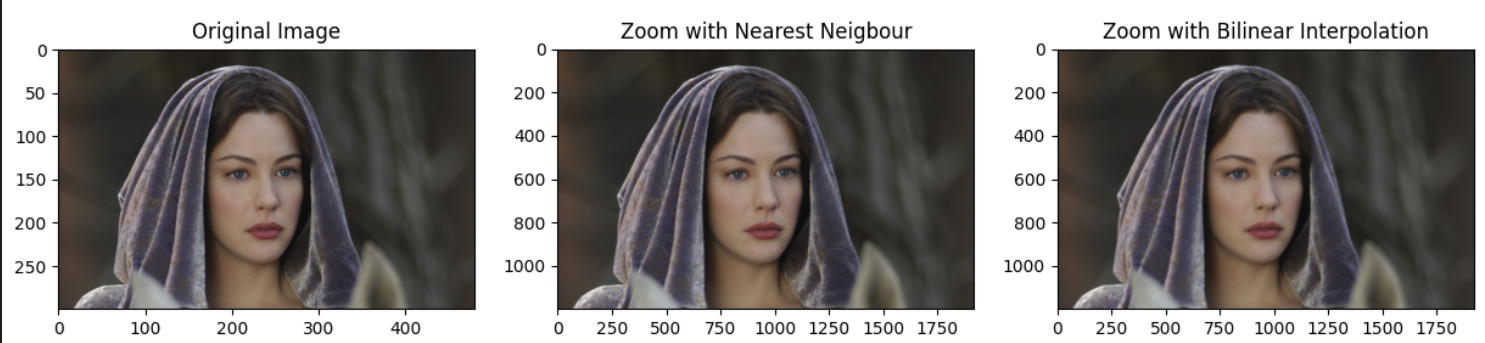
***Part b***

def sobel\_fiter(image, kernel):  
  
 h\_im, w\_im = image.shape[0] , image.shape[1]  
 h\_ke, w\_ke = kernel.shape[0] , kernel.shape[1]  
 h\_ke\_half, w\_ke\_half = h\_ke // 2, w\_ke // 2   
 result\_h=np.zeros(image.shape,dtype=np.float32)  
 image = cv.normalize(image.astype('float'), None, 0.0, 1.0, cv.NORM\_MINMAX)   
   
 for i in range(h\_ke, h\_im - h\_ke):  
 for j in range(w\_ke, w\_im - w\_ke):  
 # indentify the image window to get the 2D convolution with kernal, then convolution  
 image\_window = image[i - h\_ke\_half : i + h\_ke\_half + 1, j - w\_ke\_half : j + w\_ke\_half + 1]   
 result\_h[i, j] = np.sum(image\_window \* kernel)  
 result\_h = (result\_h - np.min(result\_h)) / (np.max(result\_h) - np.min(result\_h)) \* 255  
 return result\_h  
  
img\_h = sobel\_fiter(image7, vertical\_sobel)  
img\_v = sobel\_fiter(image7, horizontel\_sobel)  
final\_image = np.sqrt(img\_horizontal\*\*2, img\_vertical\*\*2)

***Part C***

kernal\_vertical = np.array([-1,0,1],dtype=np.float32)  
kernal\_horizontal = np.array([1,2,1],dtype=np.float32)  
img\_vertical = cv.sepFilter2D(image7, -1, kernal\_vertical, kernal\_horizontal)  
img\_horizontal = cv.sepFilter2D(image7, -1, kernal\_horizontal, kernal\_vertical)  
final\_image = np.sqrt(img\_horizontal\*\*2, img\_vertical\*\*2)

***Question 08***

def zoom(image, scale, method) :  
 h\_image , w\_image , channels = image.shape  
 h\_zoomed = h\_image\*scale  
 w\_zoomed = w\_image\*scale  
 zoomed\_image = np.zeros((h\_zoomed, w\_zoomed, channels), dtype = np.float32)  
 if method == 'nearest neighbour' :   
 for i in range(h\_zoomed) :  
 for j in range(w\_zoomed) :  
 zoomed\_image[i,j] = image[i // scale, j // scale]  
  
 if method == 'bilinear interpolation' :  
 for i in range(h\_zoomed) :  
 for j in range(w\_zoomed) :  
 x\_image , y\_image = (i / scale) , (j / scale )  
 x1 , y1 = int(x\_image) , int(y\_image)  
 x2 , y2 = x1+1, y1+1  
 if x2 >= h\_image : x2 = x1   
 if y2 >= w\_image : y2 = y1  
 dx, dy = x\_image - x1 , y\_image - y1  
 w1,w2,w3,w4 =(1 - dx)\*(1 – dy),dx\*(1 - dy),(1 - dx)\*dy, dx\*dy  
 zoomed\_image[i,j] = w1\*image[x1,y1] + w2\*image[x1,y2] + w3\*image[x2,y1] + w4\*image[x2,y2]  
 return zoomed\_image.astype(np.uint8)  
  
zoomed\_nn = zoom(image8, 4, 'nearest neighbour')  
zoomed\_bi = zoom(image8, 4, 'bilinear interpolation')

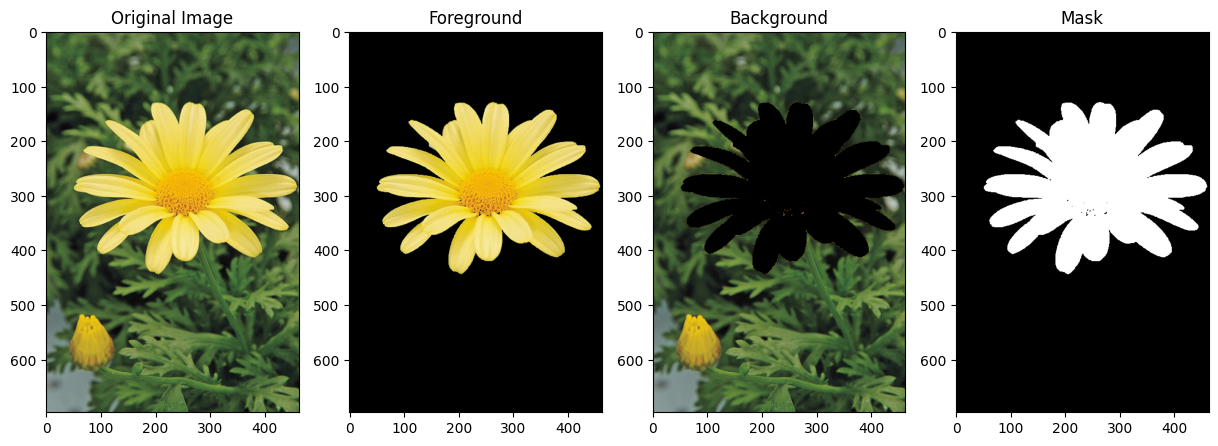
***Discussion***Nearest neighbour method gives a sharp zoomed images while bilinear interpolation gives a smooth zoomed images. It happens because, interpolation method is taking the weighted sum of the four nearest pixel values. Therefore sharp edges will smooth when zooming images using bilinear interpolation.

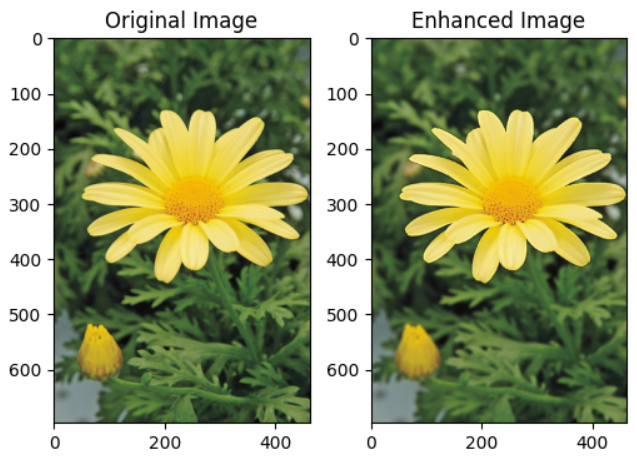
***Question 08 algorithm testing (for image 01)***

zoomed\_image = cv.imread('zooming/im01.png')  
resized = cv.resize(image8, (1920, 1080), interpolation=cv.INTER\_LINEAR)  
squared\_diff1 = np.sum(np.square(zoomed\_image - zoomed\_nn))  
squared\_diff2 = np.sum(np.square(zoomed\_image - zoomed\_bi))  
squared\_diff3 = np.sum(np.square(zoomed\_image - resized))

Squared Difference of zoomed image by Nearest Neighbour : 522633462  
Squared Difference of zoomed image by the Bilinear interpolation : 543747100  
Squared Difference of zoomed image by the resize Function : 522553013

***Question 09***

h, w, \_ = image9.shape  
mask = np.zeros((h, w), dtype = np.uint8)  
bgdModel = np.zeros((1,65),np.float64)  
fgdModel = np.zeros((1,65),np.float64)  
rectangle = (0,0,h,w)  
mask, bg\_model, fg\_model = cv.grabCut(image9, mask, rectangle, bgdModel, fgdModel, 5, cv.GC\_INIT\_WITH\_RECT)  
  
new\_mask = np.where((mask == 0) | (mask == 2), 0, 1).astype(np.uint8)  
foreground = image9 \* new\_mask[:, :, np.newaxis]  
background = image9 \* (1 - new\_mask[:, :, np.newaxis])  


***Part B***blur\_bg = cv.GaussianBlur(background, (11,11), 0,0)  
final\_image = cv.add(foreground, blur\_bg)

***Part C***

* Blurring the background decreases contrast and edge details. Deformations near the foreground edge can lead to darker appearances there.
* Using GrabCut for segmentation might misclassify edge pixels between foreground and background, leading to dark appearances near the foreground edge.

Name : R.M.S.Madhushan  
Index : 200363R  
GitHub Link : https://github.com/SadeepRathnayaka/EN3160\_Assignment1