190501V - Ranathunga R.A.C.D.

Github profile : https://github.com/ChamithDilshan

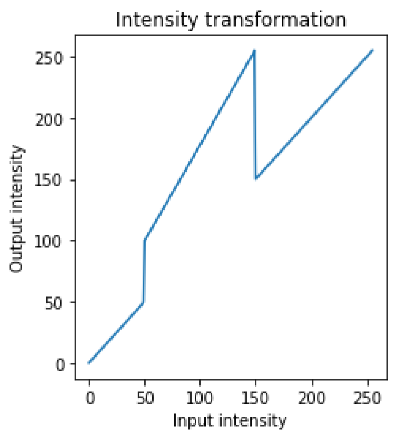
Question 01

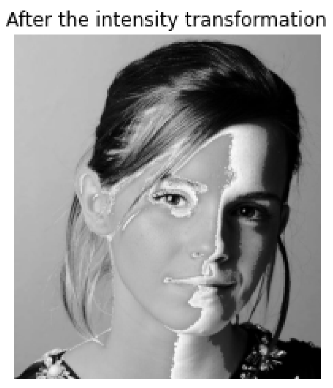
In [206…

```
#import the libraries

import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
%matplotlib inline
```

In [207…

```
original = cv.imread(r'emma_gray.jpg',cv.IMREAD_COLOR)
assert original is not None

transf_1 = np.linspace(0,50,50)  #define the transformation
transf_2 = np.linspace(100,255,100)
transf_3 = np.linspace(150,255,106)

transform = np.concatenate((transf_1,transf_2,transf_3),axis = 0).astype(np.uint8)
output = cv.LUT(original,transform) #apply the transform
fig, ax = plt.subplots(1,3,figsize=(12,4))

ax[0].plot(transform)
ax[0].set_title("Intensity transformation")
ax[0].set_ylabel("Output intensity")
ax[0].set_xlabel("Input intensity")

output = cv.cvtColor(output,cv.COLOR_BGR2RGB)
ax[1].imshow(original)
ax[1].set_title("Original image") ;  ax[1].axis('off')

ax[2].imshow(output)
ax[2].set_title("After the intensity transformation") ;  ax[2].axis('off')

plt.show()
```



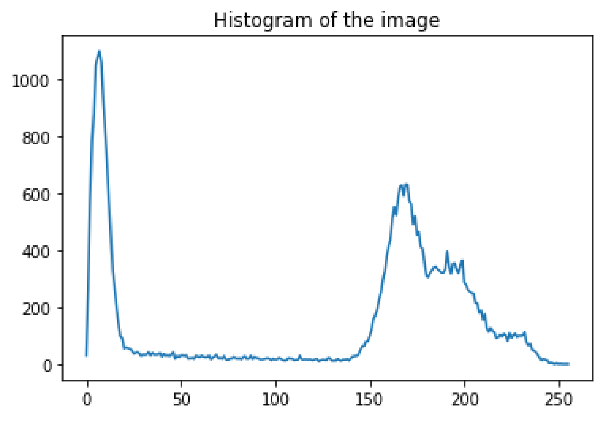


The pixels having input intensity within range (50,150) has mapped to the range (100,250). Therefore the area close to white color is increased in the resulting image

Question 02

In [208…

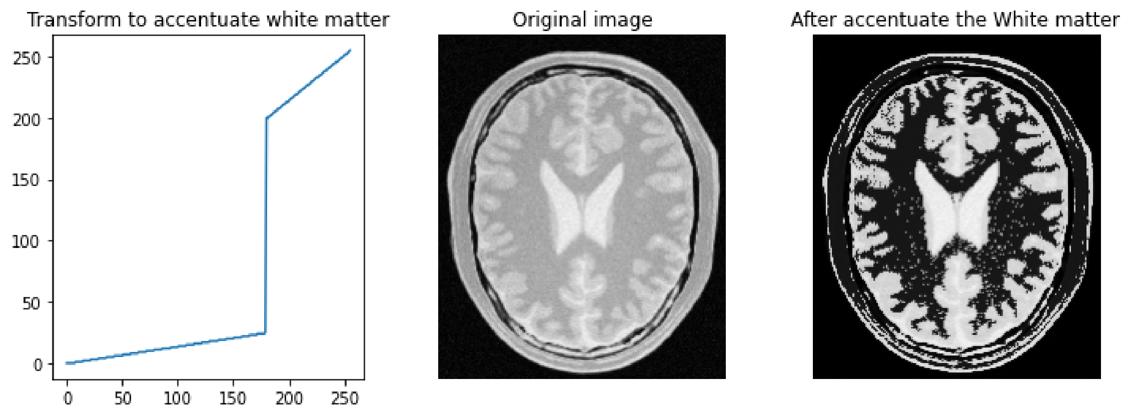
```
image = cv.imread('brain_proton_density_slice.png',cv.IMREAD_GRAYSCALE)
assert image is not None

image = cv.cvtColor(image,cv.COLOR_BGR2RGB)  #I want to get an idea about the color gray level distribution by plotting the histogram
hist = cv.calcHist([image],[0],None,[256],[0,256])
plt.plot(hist)
plt.title("Histogram of the image")
plt.show()
```



In [209…

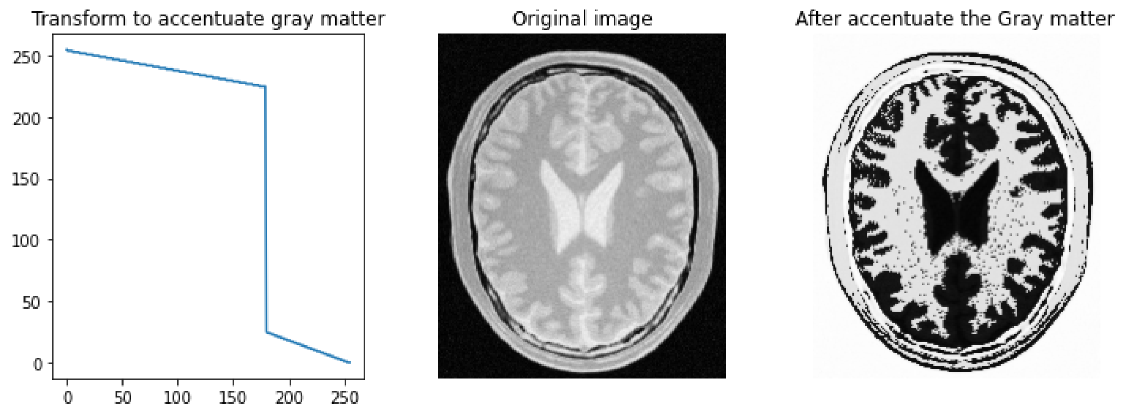
```
transf_1 = np.linspace(0,25,180)
```

```
transf_2 = np.linspace(200,255,76)
transform = np.concatenate((transf_1,transf_2),axis = 0).astype(np.uint8)

output = cv.LUT(image,transform)
output = cv.cvtColor(output,cv.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].plot(transform)
ax[0].set_title('Transform to accentuate white matter') #This transform map the white matter to be close to further white while putting others down

ax[1].imshow(image) ; ax[1].set_title("Original image") ;  ax[1].axis('off')

ax[2].imshow(output) ; ax[2].set_title("After accentuate the White matter") ;  ax[2].axis('off')
plt.show()
```


Transform to accentuate white matter
Original image
After accentuate the White matter

Pixels having intensity closed to white, have mapped to above and lesser are mapped to down. Therefore the white areas are amplified in the resulting image

In [210…
```
transf_1 = np.linspace(255,225,180)
transf_2 = np.linspace(25,0,76)
transform = np.concatenate((transf_1,transf_2),axis = 0).astype(np.uint8)

output = cv.LUT(image,transform)
output = cv.cvtColor(output,cv.COLOR_BGR2RGB)

fig, ax = plt.subplots(1,3,figsize=(12,4))
ax[0].plot(transform)
ax[0].set_title('Transform to accentuate gray matter') #This transform map the gray matter to be close to white while putting others down

ax[1].imshow(image) ; ax[1].set_title("Original image") ;  ax[1].axis('off')

ax[2].imshow(output) ; ax[2].set_title("After accentuate the Gray matter") ;  ax[2].axis('off')

plt.show()
```


Transform to accentuate gray matter
Original image
After accentuate the Gray matter

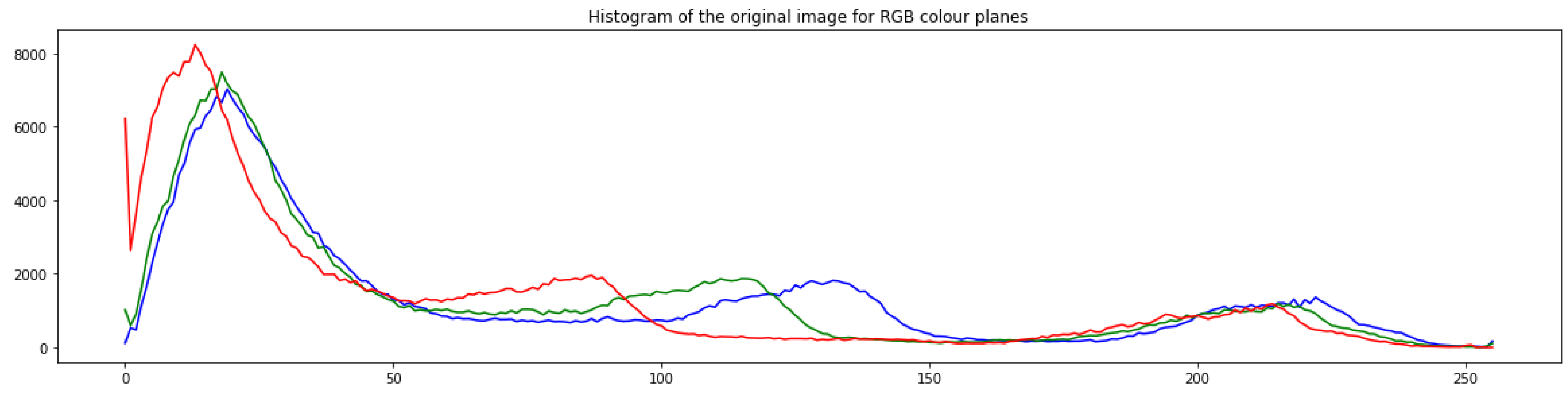
Pixels having intensity closed to gray, have mapped to down and lesser are mapped to above. Therefore the gray areas are amplified in the resulting image

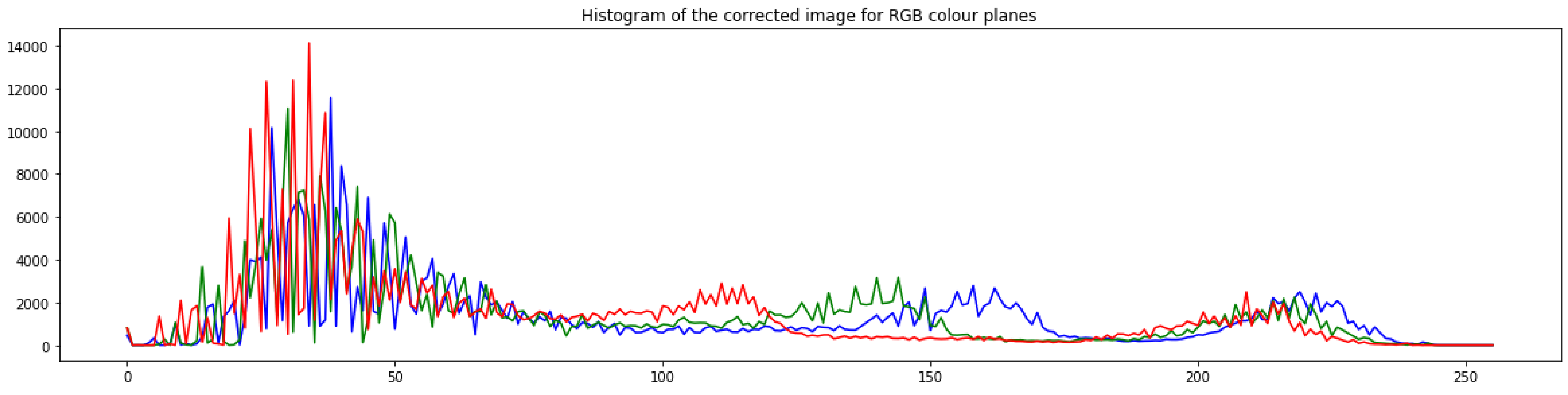
Question 03

In [211…
```
original_image = cv.imread("highlights_and_shadows.jpg", cv.IMREAD_COLOR)
assert original_image is not None

LAB_image = cv.cvtColor(original_image,cv.COLOR_BGR2LAB) #convert image to L*A*B space
gamma = 0.65
transform = np.array([(p/225)**gamma*225 for p in range(0,256)]).astype(np.uint8)  #gamma array

new_L = cv.LUT(LAB_image[:,:,0],transform)   #perform the transform to L plane
LAB_image[:,:,0] = new_L  #replace by new L plane
output = cv.cvtColor(LAB_image,cv.COLOR_LAB2RGB)  #convert image to RGB before show using matplotlib

fig,ax = plt.subplots(1,2,figsize =(10,10))
original_image = cv.cvtColor(original_image,cv.COLOR_BGR2RGB)

ax[0].imshow(original_image) ;ax[0].set_title("Original Image") ; ax[0].axis('off')
ax[1].imshow(output) ; ax[1].set_title("Corrected image") ;  ax[1].axis('off')
plt.show()
```

Since the gamma correction is applied to the lightness plane of L*a*b, the lightness of the output image is higher than the original image. Therefore, the dark areas in the original image are more clearly visible in the resulting image.

In [212…
```python
colours = ['b','g','r']  #represnts three colour planes
fig, ax = plt.subplots(2,1, figsize=(20,10))

for i,c in enumerate(colours):
    hist = cv.calcHist([original_image],[i],None,[256],[0,256])
    ax[0].plot(hist, color = c)
ax[0].set_title('Histogram of the original image for RGB colour planes')

for i,c in enumerate(colours):
    hist = cv.calcHist([output],[i],None,[256],[0,256])
    ax[1].plot(hist, color = c)
ax[1].set_title('Histogram of the corrected image for RGB colour planes')

plt.show()
```

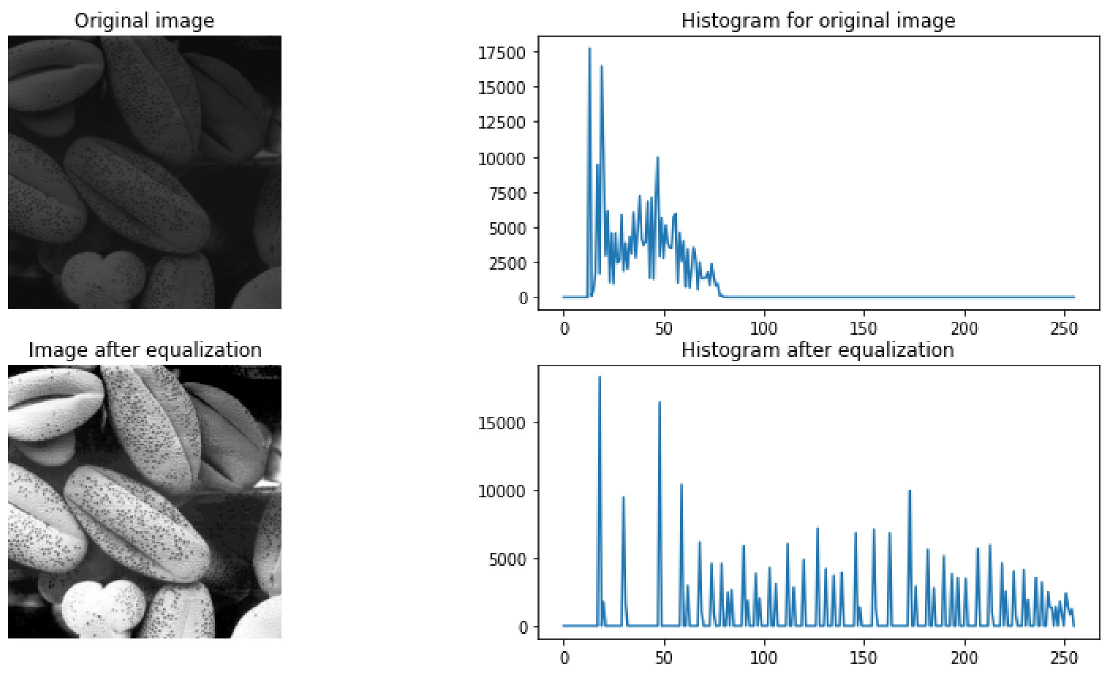




Question 04

In [213…
```python
image = cv.imread('shells.png', cv.IMREAD_GRAYSCALE)
assert image is not None

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

L = 256
image_height = image.shape[0] ; image_width = image.shape[1]

temp = [np.sum(image == i) for i in range(0,256)]
transform = temp*(int((L-1)/(image_height*image_width)))

for i in range(1,256):
    temp[i] = temp[i-1] + temp[i]

temp = [int(j*((L-1)/(image_height*image_width))) for j in temp]

temp = np.array(temp)
output = cv.LUT(image, temp)
histogram_after = [np.sum(output == i) for i in range(0,256)]

fig, ax = plt.subplots(2,2, figsize = (14,7))

ax[0,0].imshow(cv.cvtColor(image,cv.COLOR_GRAY2RGB))
ax[0,0].set_title("Original image") ; ax[0,0].axis('off')
```

```
ax[0,1].plot(hist) ; ax[0,1].set_title("Histogram for original image")

ax[1,0].imshow(output, cmap = 'gray')
ax[1,0].set_title("Image after equalization") ; ax[1,0].axis('off')

ax[1,1].plot(histogram_after) ; ax[1,1].set_title("Histogram after equalization")
plt.show()
```



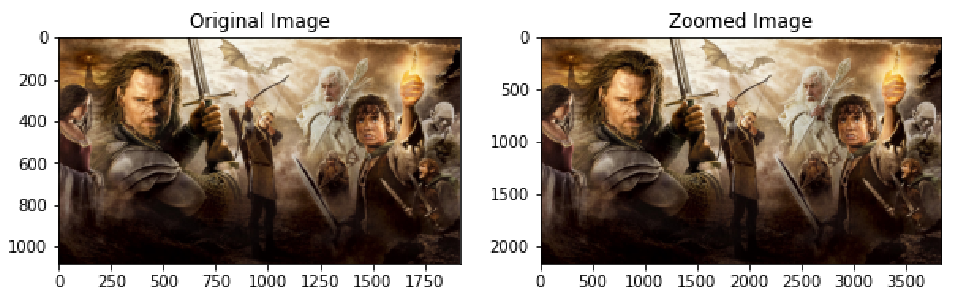
The contrast of the output image is increased. Therefore the image is more clear and vibrant. We can see that the probability of having white pixels in the original image less. But after histogram equalization that probability has increased, therefore image looks vibrant.

Question 05

Question 05)a

In [214…

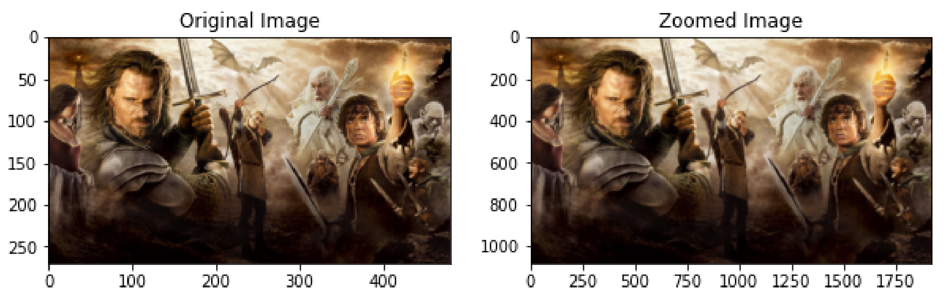
```
image = cv.imread('a1q5images/im01.png')
assert image is not None

im_h = image.shape[0] ; im_w = image.shape[1]
scaling_factor = 2

scaled_h = im_h * scaling_factor ; scaled_w = im_w * scaling_factor
zoomed = np.zeros((scaled_h,scaled_w,3), dtype=image.dtype)

for r in range(0,scaled_h):
     for c in range(0, scaled_w):
        if round(r/scaling_factor) == im_h and round(c/scaling_factor) == im_w:
            zoomed[r,c] = (image[round(r/scaling_factor)-1, round(c/scaling_factor)-1])
        elif  round(r/scaling_factor) == im_h:
            zoomed[r,c] = (image[round(r/scaling_factor)-1, round(c/scaling_factor)])
        elif round(c/scaling_factor) == im_w:
            zoomed[r,c] = (image[round(r/scaling_factor), round(c/scaling_factor)-1])
        else:
            zoomed[r,c] = (image[round(r/scaling_factor), round(c/scaling_factor)])
        #This is because in the last pixel, the index when devided canbe outof original image index limit
        #For an example: if the width of zoomed image is 1000 and if we divide by 2, then 500 is not in original image. Therefore then we have to 
take 499

fig, ax = plt.subplots(1,2, figsize=(10,10))
ax[0].imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB)) ; ax[0].set_title("Original Image")
ax[1].imshow(cv.cvtColor(zoomed, cv.COLOR_BGR2RGB)) ; ax[1].set_title("Zoomed Image")
plt.show()
```

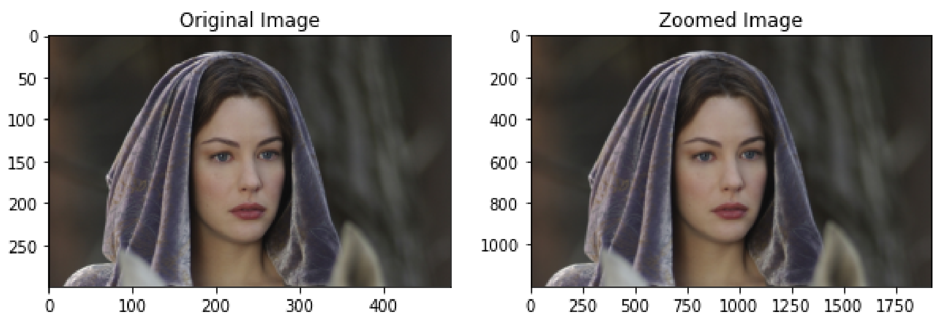


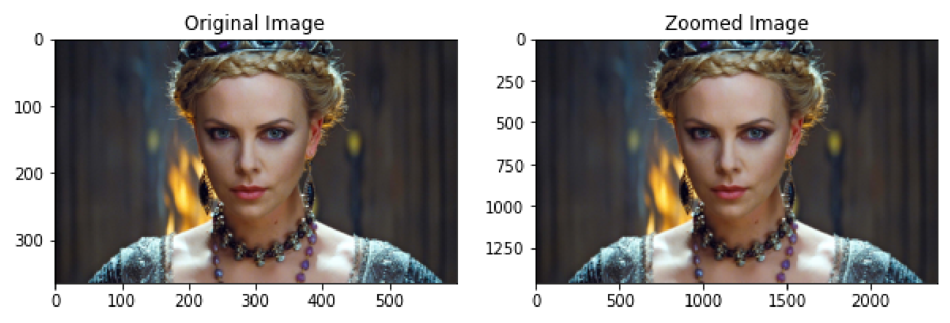
Both the image width and height of output is twice of that of the input image (Because the scaling factor is 2). There could be slight errors in pixels when we use the nearest-neighbor method

In [215…

```
import os
def bilinear_interpolation(image, image_title):
    im_h = image.shape[0] ; im_w = image.shape[1]
    scaled_h = im_h * scaling_factor ; scaled_w = im_w * scaling_factor

    zoomed = np.zeros((scaled_h,scaled_w,3), dtype=image.dtype)
    for i in range(0,scaled_h):
        for j in range(0, scaled_w):

            m = i/ scaling_factor
            n = j/ scaling_factor

            i1 = int(np.floor(m))
            i2 = int(np.ceil(m))

            j1 = int(np.floor(n))
            j2 = int(np.ceil(n))

            if i1 == im_h: i1 = im_h-1
            if i2 == im_h: i2 = im_h-1
            if j1 == im_w: j1 = im_w-1
            if j2 == im_w: j2 = im_w-1

            if i1 == i2 and j1 == j2:
                zoomed[i,j] = image[i1,j1]
            elif i1 == i2:
                left_side = image[i1,j1]
                right_side = image[i1,j2]
                zoomed[i,j] = np.round((n-j1)*right_side + (j2-n)*left_side).astype(int)

            elif j1 == j2:
                upper_side = image[i1,j1]
                bottom_side = image[i2,j1]
                zoomed[i,j] = np.round((m-i1)*bottom_side + (i2-m)*upper_side).astype(int)

            else:
                left_side = (m-i1)*image[i2,j1] + (i2-m)*image[i1,j1]
                right_side = (m-i1)*image[i2,j2] + (i2-m)*image[i1,j2]

                zoomed[i,j] = np.round((n-j1)*right_side + (j2-n)*left_side).astype(int)

    fig, ax = plt.subplots(1,2, figsize=(10,10))

    ax[0].imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
    ax[0].set_title("Original Image")

    ax[1].imshow(cv.cvtColor(zoomed, cv.COLOR_BGR2RGB))
    ax[1].set_title("Zoomed Image")
    plt.show()
    cv.imwrite(os.path.join(path, image_title), zoomed )

path = 'a1q5images/'

images = ["im01small.png", "im02small.png", "im03small.png"]
scaling_factor = 4
for i in range(3):
    image = cv.imread('a1q5images/'+images[i])
    assert image is not None
    save_as = images[i][0:4]+" zoomed_by_%s.png"%(scaling_factor)
    bilinear_interpolation(image, save_as)
```







The accuracy looks high when we use the bilinear interpolation.

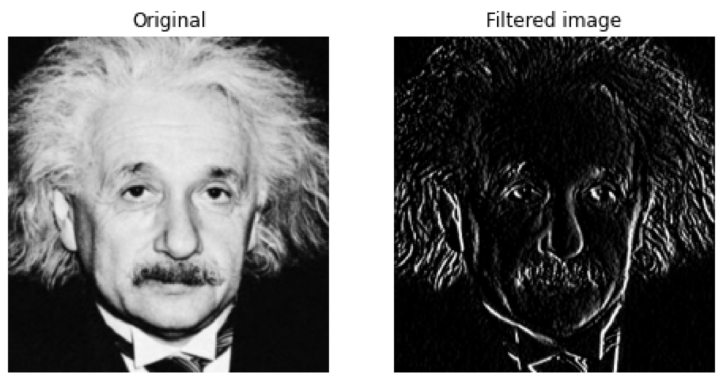
In [216…
```python
def SSD(original_image , second_image):
    original_image_sqr = np.square(original_image)
    second_image_sqr = np.square(second_image)
    normalized_squared_differences = (original_image_sqr - second_image_sqr)/(original_image.size)
    print(np.sum(normalized_squared_differences))



for i in range(1,4):
    original_image = cv.imread("a1q5images/im0%s.png"%i).astype(np.float32)
    second_image = cv.imread("a1q5images/im0%s zoomed_by_4.png"%i).astype(np.float32)
    if i==3: original_image = np.resize(original_image,(1460,2400,3))  #because the given image to compare has missed a one pixel
    print("SSD for image 0%s: "%i, end= ''); SSD(original_image, second_image)
```

```
SSD for image 01: 220.2565134709362
SSD for image 02: 3.183057581018519
SSD for image 03: 63.54420700152209
```

Question 06

Question 06)a

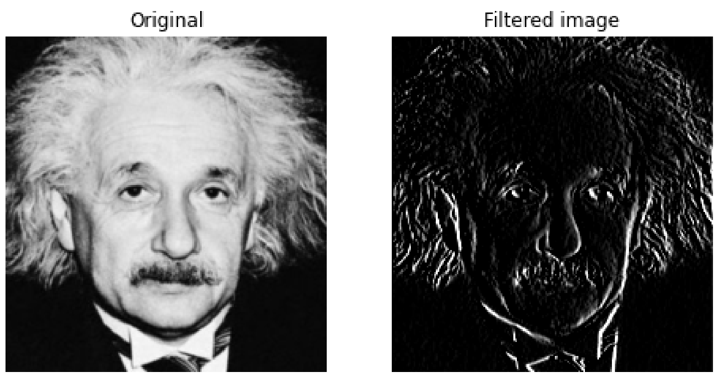
In [217…
```python
einstein_original = cv.imread("einstein.png", cv.IMREAD_GRAYSCALE)  #open the original image in grayscale

sobel_filter = np.array([[1,0,-1],[2,0,-2],[1,0,-1]], dtype = np.float32) #define the sobel filter
filtered_einstein = cv.filter2D(einstein_original, -1, sobel_filter) #apply filter2D function

fig,ax = plt.subplots(1,2, figsize = (8,8))  #show the images
ax[0].imshow(einstein_original, cmap = 'gray')
ax[0].set_title("Original") ; ax[0].axis('off')

ax[1].imshow(filtered_einstein, cmap = 'gray')
ax[1].set_title("Filtered image") ; ax[1].axis('off')

plt.show()
```



The edges are detected and denoted by white color lines

Question 06)b

In [222…
```python
original_image = cv.imread("einstein.png", cv.IMREAD_GRAYSCALE)

kernal = np.array([[1,0,-1],[2,0,-2],[1,0,-1]])
kernal_size = kernal.shape[0]

num_rows = original_image.shape[0]
num_columns = original_image.shape[1]

padding_row= np.zeros((kernal_size//2,num_columns))
padding_column = np.zeros(( 2* int(np.floor(kernal_size/2)) + num_rows, kernal_size//2))

temp = np.concatenate((padding_row,original_image,padding_row),axis=0)
padded_matrix = np.concatenate((padding_column,temp,padding_column),axis=1)

filtered = np.zeros(original_image.shape)

for j in range(0,padded_matrix.shape[0]-kernal_size//2- 1):
    temp2 = padded_matrix[j:j+kernal_size,:]
    array_ = np.zeros((1, num_columns))
    for i in range(0, padded_matrix.shape[1]-kernal_size+1 ):
          temp3 = padded_matrix[j :j +kernal_size,i:kernal_size+i]
          array_[0,i] = np.sum(np.multiply(temp3, kernal))

    filtered[j] = array_

fig,ax = plt.subplots(1,2, figsize = (8,8))  #show the images
ax[0].imshow(original_image, cmap = 'gray')
ax[0].set_title("Original") ; ax[0].axis('off')

ax[1].imshow(filtered, cmap = 'gray', vmin=0, vmax=255)
ax[1].set_title("Filtered image") ; ax[1].axis('off')

plt.show()
```

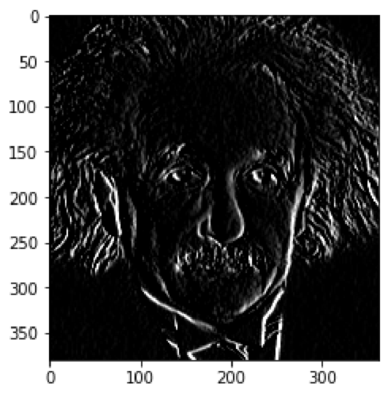


The result when using my own implementation is quite similiar to the result when using filter2D

Question 06)c

In [219…

```
original_image = cv.imread("einstein.png", cv.IMREAD_GRAYSCALE)
num_rows = original_image.shape[0]
num_cols = original_image.shape[1]

kernal1 = np.array([[1],[2],[1]])

temp1 = np.zeros((1,num_cols)).astype(np.uint8)
padded_matrix1 = np.concatenate((temp1,original_image,temp1), axis = 0)

after_kernal1 = np.zeros(original_image.shape).astype(np.float32)

for i in range(1,num_rows+1):
    for j in range(0, num_cols):
        temp2 = np.array([[padded_matrix1[i-1][j]],[padded_matrix1[i][j]],[padded_matrix1[i+1][j]]])
        after_kernal1[i-1][j] = np.sum(np.multiply(temp2,kernal1))

kernal2 = np.array([1,0,-1])
temp3 = np.zeros((num_rows,1)).astype(np.float32)
padded_matrix2 = np.concatenate((temp3,after_kernal1,temp3), axis = 1)

after_kernal2 = np.zeros(after_kernal1.shape).astype(np.float32)

for i in range(0,num_rows):
    for j in range(1, num_cols+1):
        temp4 = np.array([padded_matrix2[i][j-1],padded_matrix2[i][j],padded_matrix2[i][j+1]])
        after_kernal2[i][j-1] = np.sum(np.multiply(temp4,kernal2))

plt.imshow(after_kernal2, cmap = 'gray', vmin=0, vmax=255)
plt.show()
```



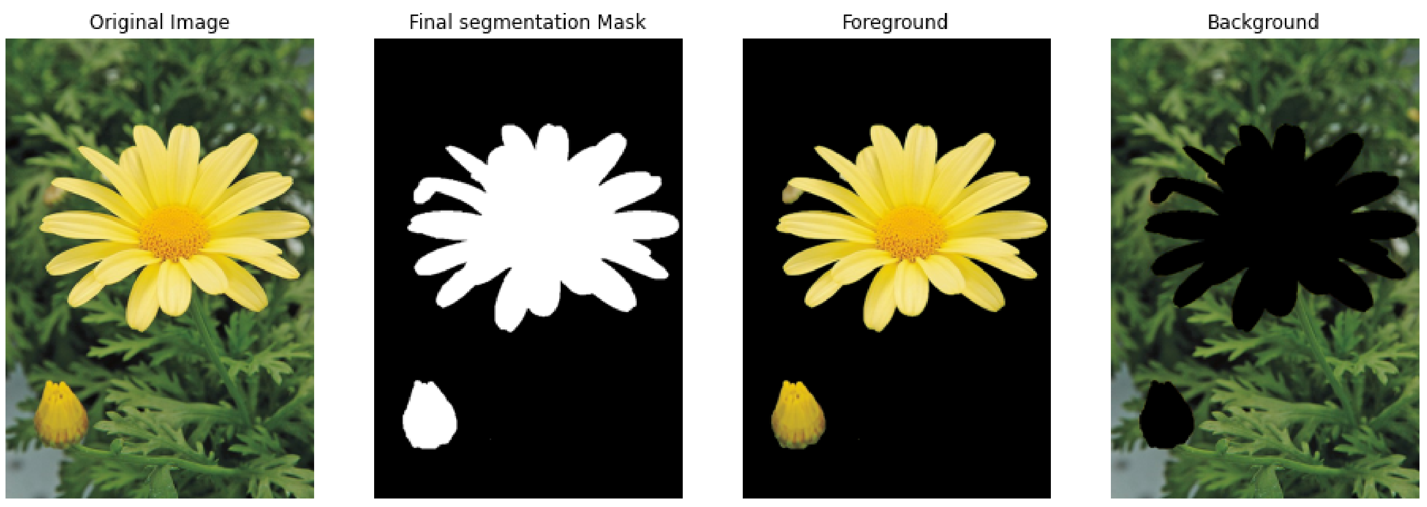
The same result could be obtained by this method. The convolution has run twice

Question 07

In [220…

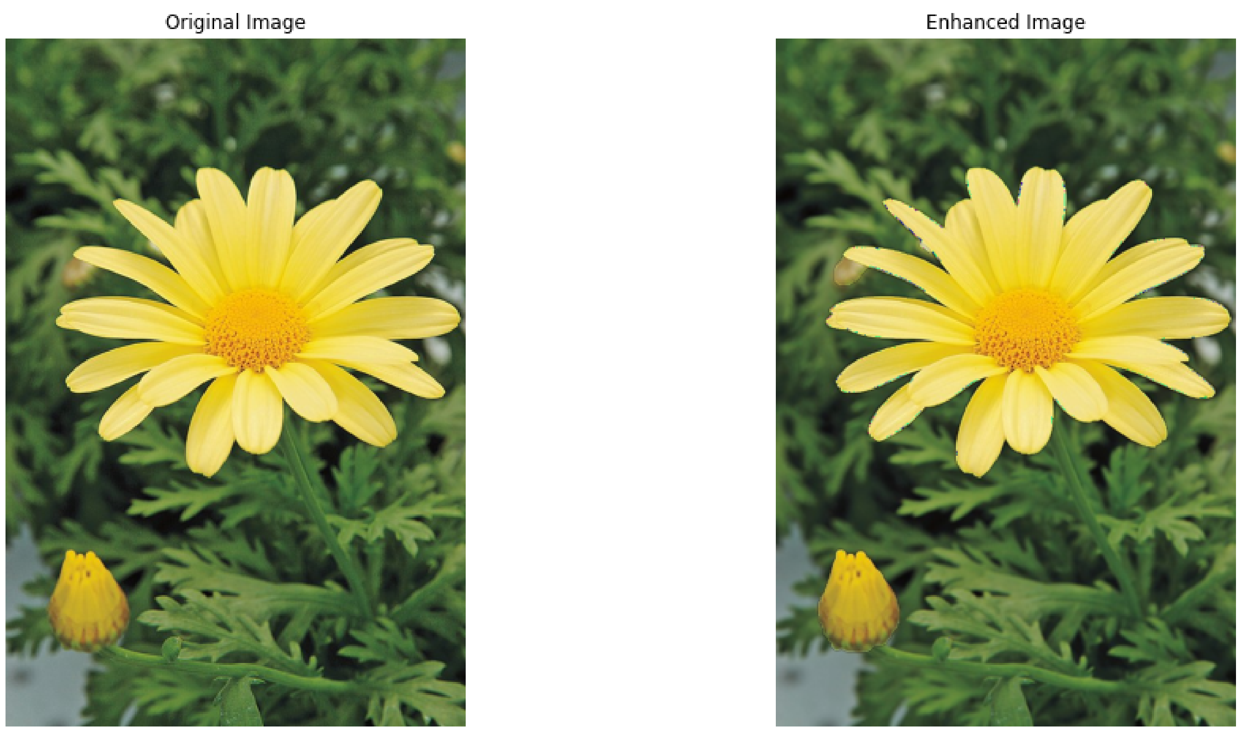
```
original_image = cv.imread('daisy.jpg')

mask = np.zeros(original_image.shape[:2],np.uint8)
```

```
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)

rect = (25,150,535,600)

cv.grabCut(original_image,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
foreground = original_image*mask2[:,:,np.newaxis]

mask3 = np.zeros(original_image.shape[:2],np.uint8)
mask3 = np.where(mask2==1,0,1).astype('uint8')
background = original_image*mask3[:,:,np.newaxis]

foreground = cv.cvtColor(foreground, cv.COLOR_BGR2RGB)
original_image = cv.cvtColor(original_image, cv.COLOR_BGR2RGB)
background = cv.cvtColor(background, cv.COLOR_BGR2RGB)

fig , ax = plt.subplots(1,4, figsize = (16,16))
ax[0].imshow(original_image); ax[0].set_title("Original Image"); ax[0].axis('off');
ax[1].imshow(mask2, cmap = 'gray'); ax[1].set_title("Final segmentation Mask"); ax[1].axis('off');
ax[2].imshow(foreground); ax[2].set_title("Foreground"); ax[2].axis('off');
ax[3].imshow(background); ax[3].set_title("Background"); ax[3].axis('off');

plt.show()
```



The area matches with the white areas in the final segmentation mask is visible as a color image in the foreground. The rest is the background. When we change the size of the rectangle in GrabCut, the area we consider as foreground changes.

In [221…
```
fig , ax = plt.subplots(1,2, figsize = (16,8))

blurred_background = cv.GaussianBlur(background,(5,5),0)
enhanced = blurred_background + foreground

ax[0].imshow(original_image); ax[0].set_title("Original Image"); ax[0].axis('off');
ax[1].imshow(enhanced); ax[1].set_title("Enhanced Image"); ax[1].axis('off');

plt.show()
```



The background is blured. Hence the foreground is more enhanced.

c) Here to enhanced the image, I have blurred the background and added the foreground on top of that. When we blur the background, the black colour areas get shaded and spreaded.Therefore, after we implement the foreground on top of that, we can see a small dark region around that.