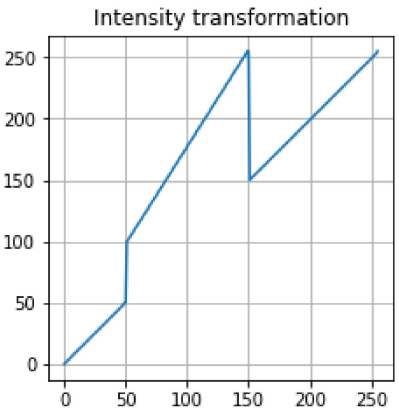
EN2550 Fundamentals of Image Processing and Machine vision

Assignment 1

Name : Jegakumaran P.
Index Number : 190280N

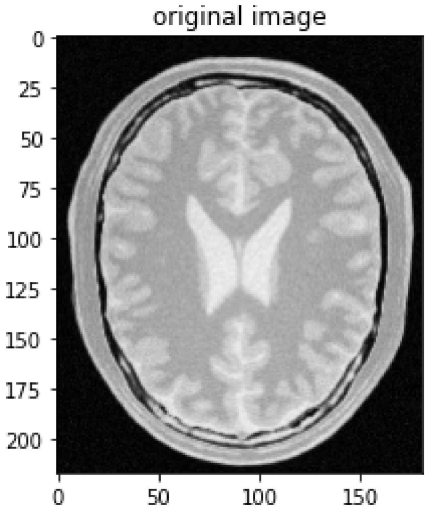
In [ ]:
```
# Question 1
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img= cv.imread(r'Assignment Images/emma_gray.jpg',cv.IMREAD_UNCHANGED)
assert img is not None
t1=np.linspace(0,50,50)
t2=np.linspace(50,100,1)
t3=np.linspace(100,255,99)
t4=np.linspace(255,150,1)
t5=np.linspace(150,255,105)
t=np.concatenate((t1,t2,t3,t4,t5),axis=0).astype(np.uint8)
assert len(t)==256
g=cv.LUT(img,t)
```

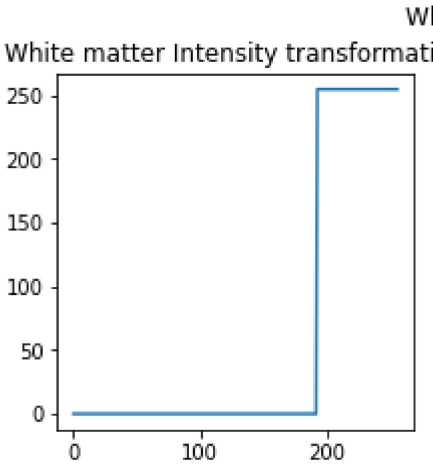


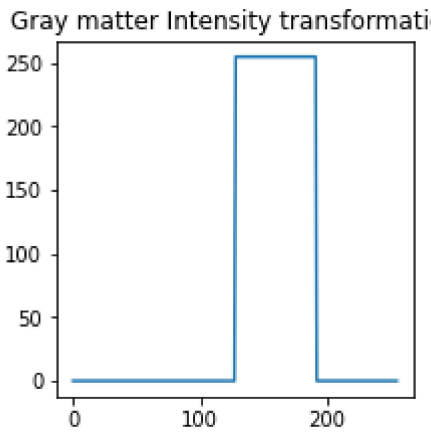
Result Comparision : We can say that intensity of the image within the range of (50,150) has been mapped to higher values. There are sudden changes in the intensity level at 50 and 150. So, we can observe major difference in the intensity transformed image near those values.

In [ ]:
```
# Question 2
img= cv.imread(r'Assignment Images/brain_proton_density_slice.png',cv.IMREAD_UNCHANGED
assert img is not None
# White Matter
t1=np.linspace(0,0,192)
t2=np.linspace(0,255,0)
t3=np.linspace(255,255,64)
t=np.concatenate((t1,t2,t3),axis=0).astype(np.uint8)
assert len(t)==256
g=cv.LUT(img,t)
#Gray Matter
t1= np.linspace(0,0,128)
t2=np.linspace(0,255,0)
t3=np.linspace(255,255,64)
```

```
t4=np.linspace(255,0,0)
t5=np.linspace(0,0,64)
t=np.concatenate((t1,t2,t3,t4,t5),axis=0).astype(np.uint8)
assert len(t)==256
g=cv.LUT(img,t)
```

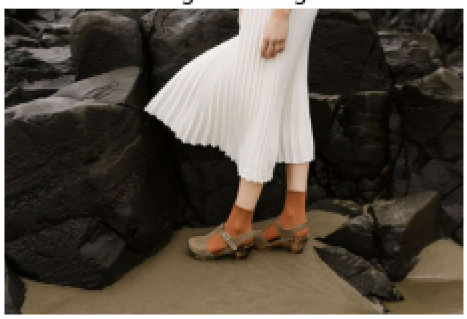


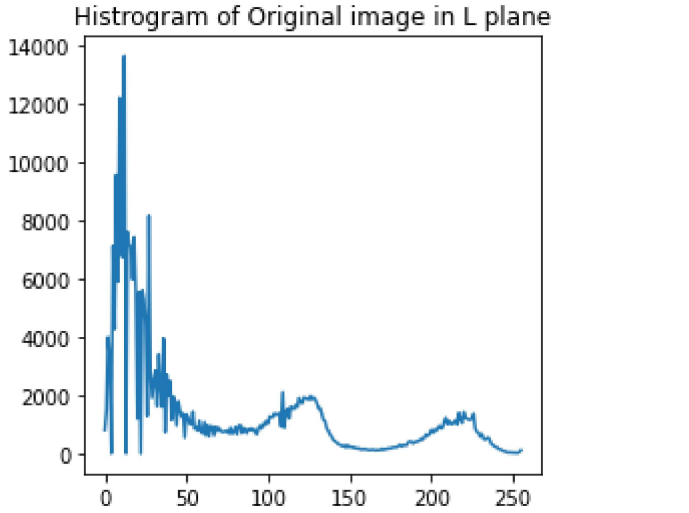


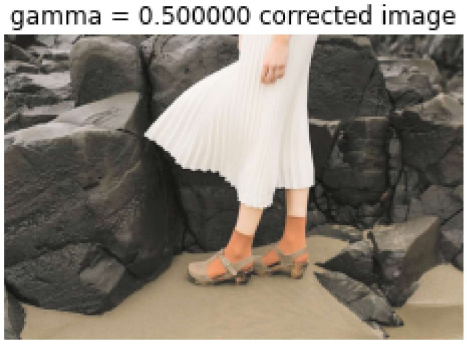


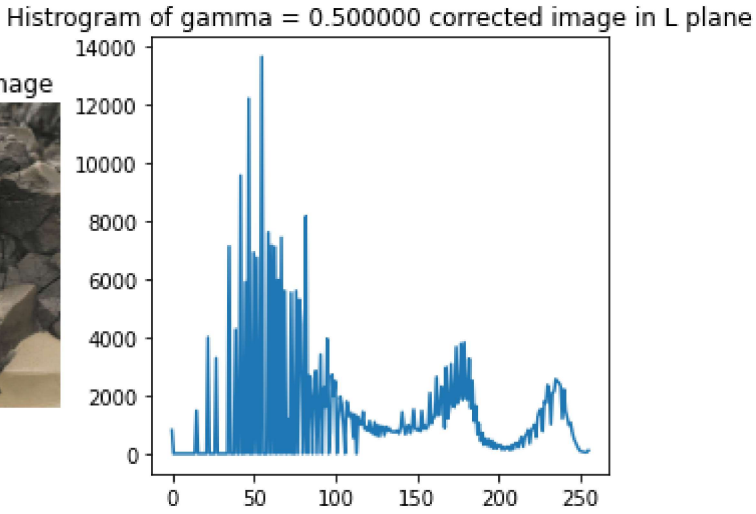
White Matter : According to the graph only the pixels within the white range (nearly 200 to 255) will be shown. Other pixels will be in black colour. Gray Matter : According to the graph only the pixels within the gray range (nearly 125 to 200) will be shown. Other pixels will be in black colour.

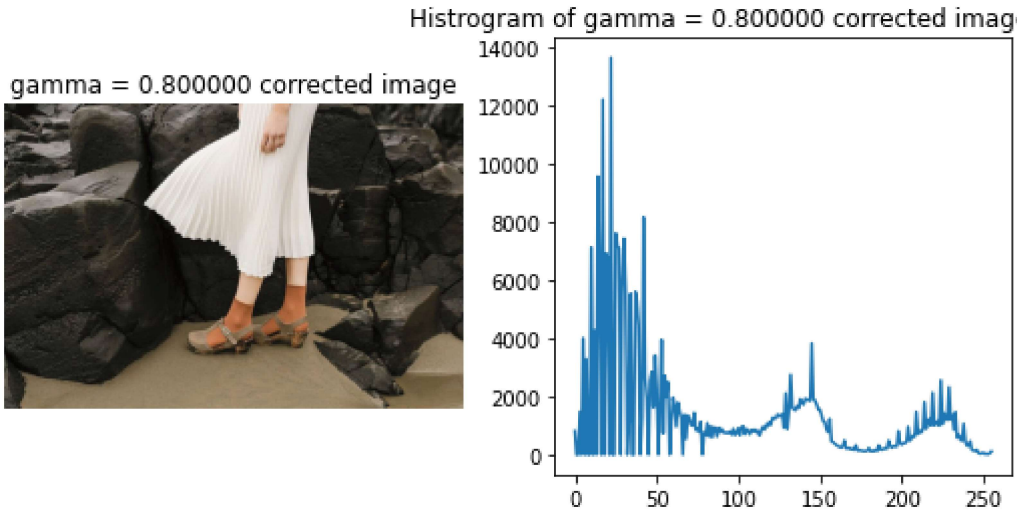
In [ ]:
```
# Question 3
img= cv.imread(r'Assignment Images/highlights_and_shadows.jpg',cv.IMREAD_UNCHANGED)
assert img is not None
lab = cv.cvtColor(img, cv.COLOR_BGR2LAB)
l,a,b = cv.split(lab)
hist_img=cv.calcHist([lab],[0],None,[256],[0,256])
gamma = [0.5,0.8,1.2]
for i in range(len(gamma)):
    t= np.array([(p/255)**gamma[i]*255 for p in range(0,256)]).astype(np.uint8)
    g_l=cv.LUT(l,t)
    g_lab=cv.merge([g_l,a,b])
    new_img=cv.cvtColor(g_lab,cv.COLOR_LAB2BGR)
    hist_new=cv.calcHist([g_lab],[0],None,[256],[0,256])
```

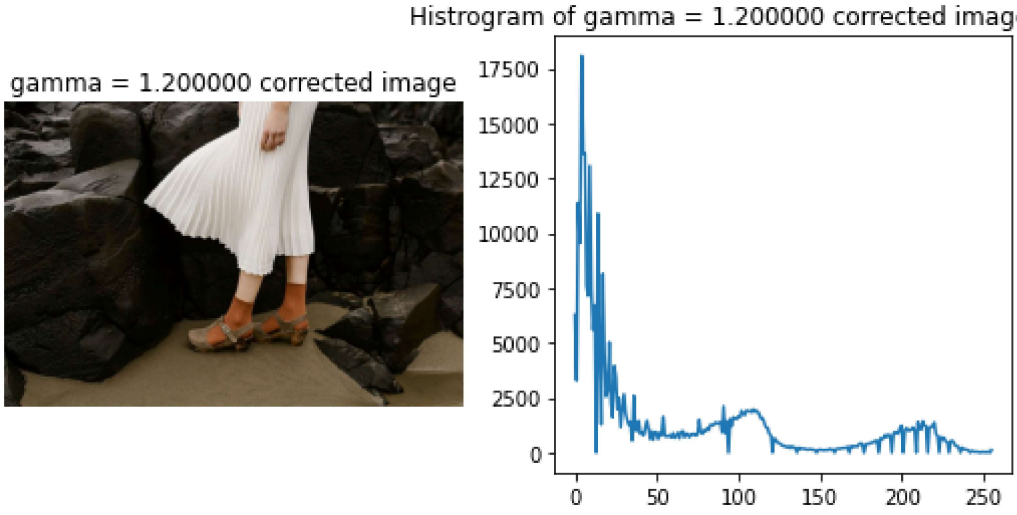








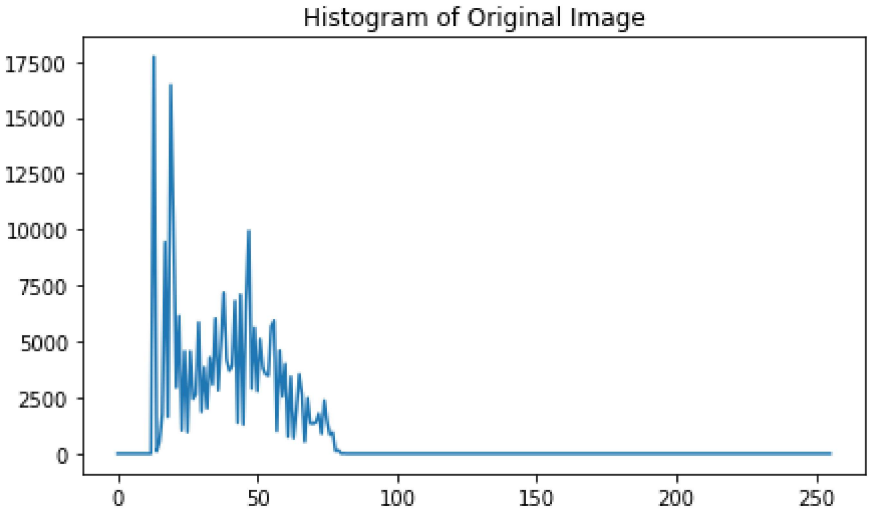


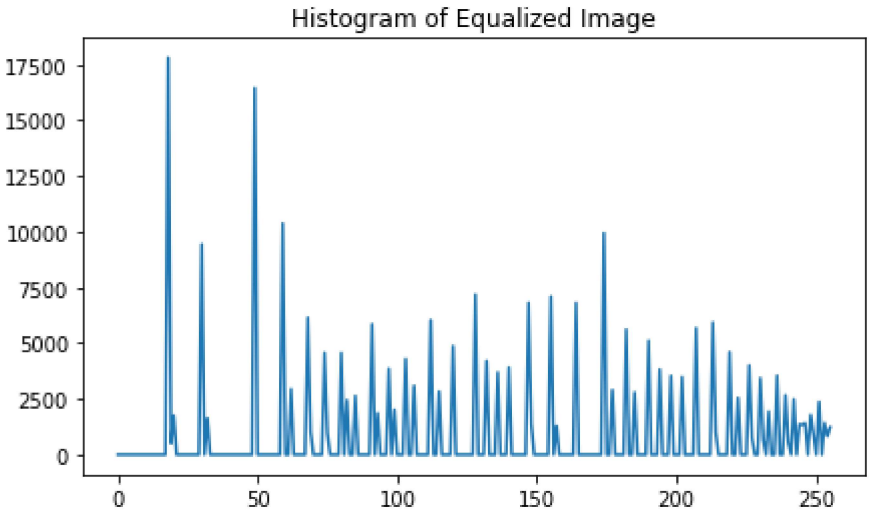


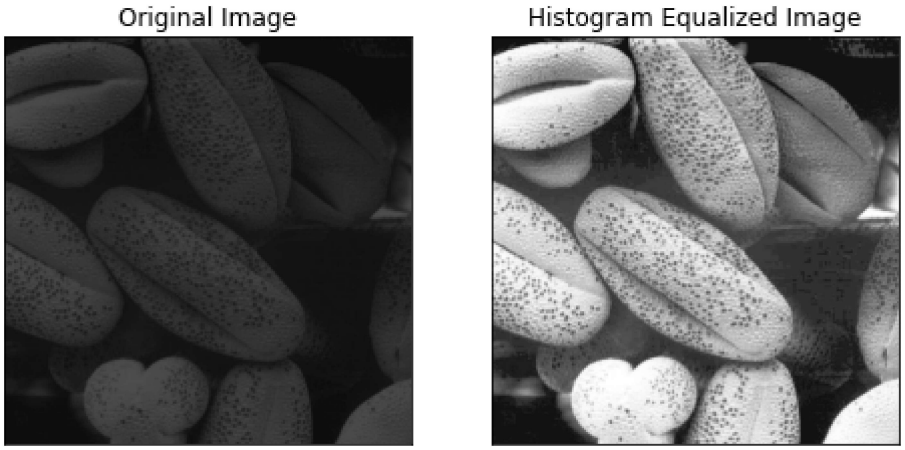
Result comparision: If the gamma value is lesser, then the histogram is spreaded in wider range, but with the increasing gamma values histrogram gets squeezed. (Images with lesser gamma values are more brighter than images with higher gamma values)

In [ ]:
```
# Question 4
img = cv.imread(r"Assignment Images/shells.png",cv.IMREAD_GRAYSCALE)
img_flat=img.flatten()
def histogram_equalize(img,size):
    img_pixel_ =np.zeros((256,1)).astype(int)
    index_=np.linspace(0,255,256).reshape(256,1).astype(int)
    img_pixel=np.append(index_,img_pixel_,axis=1)
    for j in range(size):
        img_pixel[img[j],1]+=1 #increse the intensity count
    #Cumulative sum
    sum=0
    for i in range(256):
        sum+=img_pixel[i,1]
        img_pixel[i,1]=sum
        img_pixel[i,1]= round(img_pixel[i,1]*255/size)
    hist_img=np.zeros((1,size)).astype(np.uint8)
    for i in range(size):
        hist_img[0,i] = img_pixel[img[i],1]
    new_img=hist_img[0,:]
    return new_img
def histogram(img,size,title):
```

```
    img_pixel =np.zeros((256,1))
    index_=np.linspace(0,255,256).reshape(256,1).astype(int)
    img_pixel=np.append(index_,img_pixel,axis=1).astype(int)
    for j in range(size):
        img_pixel[int(img[j]),1]+=1
size=np.shape(img_flat)[0]
histogram(img_flat,size,"Histogram of Original Image") #plot histogram of original ima
hist_image=histogram_equalize(img_flat,size)
histogram(hist_image,size,"Histogram of Equalized Image") #plot histogram of equalized
```







Result comparision : Histogram of equalized image is spreaded in the all range, due to this

histogram equalized image here is more brighter than original image(because image we used here is gray image)

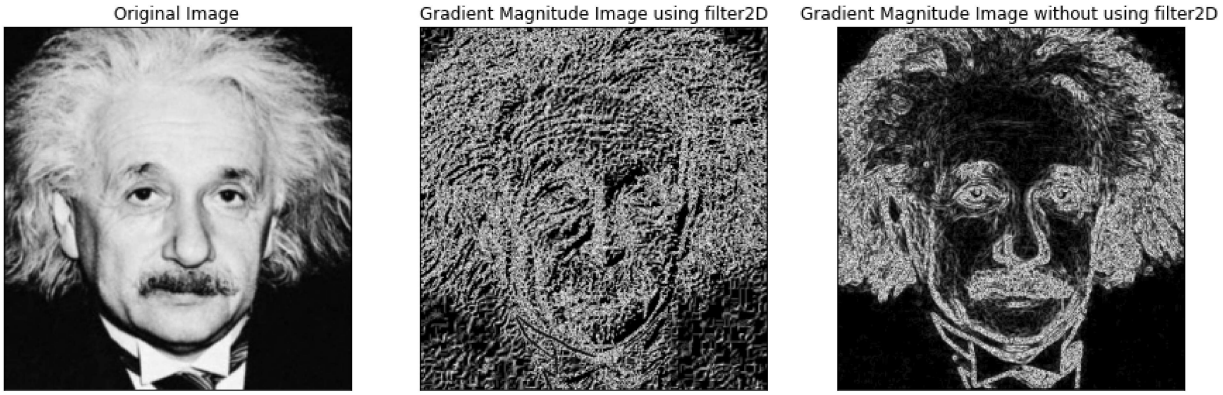
In [ ]:
```
# Question 5
img_1_small=cv.imread(r'Assignment Images/im01small.png',cv.IMREAD_GRAYSCALE)
img_2_small=cv.imread(r'Assignment Images/im02small.png',cv.IMREAD_GRAYSCALE)
img_3_small=cv.imread(r'Assignment Images/im03small.png',cv.IMREAD_GRAYSCALE)

img_1=cv.imread(r'Assignment Images/im01.png',cv.IMREAD_GRAYSCALE)
img_2=cv.imread(r'Assignment Images/im02.png',cv.IMREAD_GRAYSCALE)
img_3=cv.imread(r'Assignment Images/im03.png',cv.IMREAD_GRAYSCALE)
#Zoom image by Nearest Neighbour
def zoomimage_NN(img,zoomfactor):
    zoom_img = cv.resize(img,None, fx = zoomfactor, fy = zoomfactor, interpolation = c
    return zoom_img
#Zoom image by Bilinear Interpolation
def zoomimage_BI(img,zoomfactor):
    zoom_img = cv.resize(img,None, fx = zoomfactor, fy = zoomfactor, interpolation = c
    return zoom_img
img_small,img=[img_1_small,img_2_small,img_3_small],[img_1,img_2,img_3]
for i in range(2):
    zoom_img_NN=zoomimage_NN(img_small[i],4)
    zoom_img_BI= zoomimage_BI(img_small[i],4)
    #Calculating SSD
    difference_NN=np.subtract(zoom_img_NN,img[i])
    squ_NN=np.square(difference_NN)
    SSD_NN=np.sum(squ_NN)
    difference_BI=np.subtract(zoom_img_BI,img[i])
    squ_BI=np.square(difference_BI)
    SSD_BI=np.sum(squ_BI)
    print('SSD Of Nearest Neighbour = ',SSD_NN)
    print('SSD of Bilinear Interpolation = ',SSD_BI)
```

```
SSD Of Nearest Neighbour =  64809605
SSD of Bilinear Interpolation =  64245250
```


```
SSD Of Nearest Neighbour =  27496006
SSD of Bilinear Interpolation =  24595568
```

Result comparision : Zooming image using the bilinear interpolation method is more clear than using the nearest neighbour method. We can clearly state the above statement by calculting SSD. SSD of bilinear interpolation < SSD of nearest neighbour

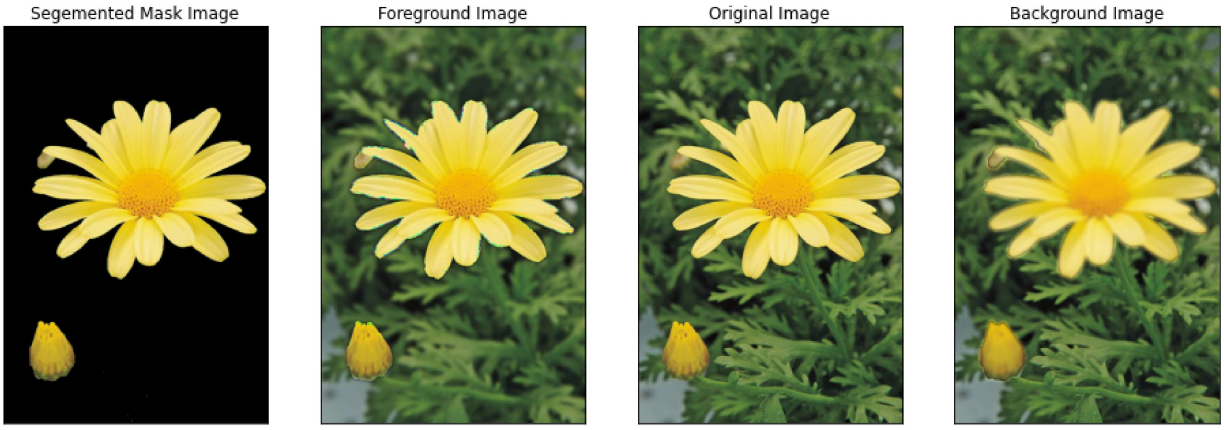
In [ ]:
```
# Question 6(a)
import math
img=cv.imread(r"Assignment Images/einstein.png",cv.IMREAD_GRAYSCALE)
assert img is not None
kernel_v,kernel_h=np.array([[-1,-2,-1],[0,0,0],[1,2,1]]),np.array([[-1,0,1],[-2,0,2],[
img_x,img_y=cv.filter2D(img,-1,kernel_v),cv.filter2D(img,-1,kernel_h)
grad_mag_filter2D=np.sqrt(img_x**2+img_y**2)
def filter(image, kernel):# Filter Function
    assert kernel.shape[0]%2 == 1 and kernel.shape[1]%2==1
    k_hh,k_hw = math.floor(kernel.shape[0]/2),math.floor(kernel.shape[1]/2)
    h,w = image.shape
    image_float= cv.normalize(image.astype("float"),None,0.2,0.3,cv.NORM_MINMAX)
    result= np.zeros( image.shape,"float")
    for m in range(k_hh,h-k_hh):
        for n in range(k_hw,w-k_hw):
            result[m,n]=np.dot(image_float[m-k_hh: m+k_hh+1,n-k_hw: n+k_hw+1].flatten(
    return result
def filter_sep(image,kernel_col,kernel_row):
    k_hh,k_hw = math.floor(kernel_row.shape[1]/2),math.floor(kernel_col.shape[0]/2)
    h,w = image.shape
    image_float= cv.normalize(image.astype("float"),None,0.2,0.3,cv.NORM_MINMAX)
    result= np.zeros( image.shape,"float")
    for m in range(k_hh,h-k_hh):
        for n in range(k_hw,w-k_hw):
            result[m,n]=np.dot(np.dot(image_float[m-k_hh: m+k_hh+1,n-k_hw: n+k_hw+1],k
    return result
imgb_v,imgb_h=filter(img,kernel_v),filter(img,kernel_h)
imgb_v=(imgb_v*255.0).astype(np.uint8)
imgb_h=(imgb_h*255.0).astype(np.uint8)
grad_mag=np.sqrt(imgb_v**2+imgb_h**2)
```



Result comparision : Gradient magnitude image using loop (without filter2D function) is inefficient method comparing with using filter2D function

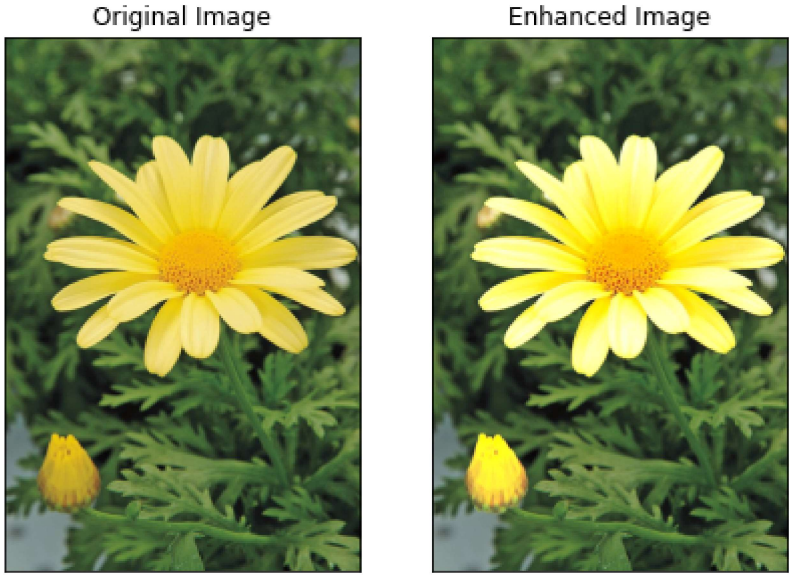
In [ ]:
```
# Question 7
img = cv.imread("Assignment Images/daisy.jpg",cv.IMREAD_COLOR) # Read the original ima
assert img is not None
# Define boundary rectangle containing the foreground object
height, width,_ = img.shape
left_margin_proportion = 0.1
right_margin_proportion = 0.1
up_margin_proportion = 0.1
down_margin_proportion = 0.1
boundary_rectangle = (int(width * left_margin_proportion),int(height * up_margin_propo
    int(width * (1 - right_margin_proportion)),int(height * (1 - down_margin_proportio
cv.setRNGSeed(0)# Set the seed for reproducibility purposes
```

```
mask = np.zeros((height, width), np.uint8) # Initialize GrabCut mask image, that will
background_model = np.zeros((1, 65), np.float64)# Arrays used by the algorithm interna
foreground_model = np.zeros((1, 65), np.float64)# Arrays used by the algorithm interna
number_of_iterations = 5
grab_img = cv.grabCut(img=img,mask=mask,rect=boundary_rectangle,bgdModel=background_mo
    iterCount=number_of_iterations,mode=cv.GC_INIT_WITH_RECT,)
grabcut_mask = np.where((mask == cv.GC_PR_BGD) | (mask == cv.GC_BGD), 0, 1).astype("ui
segmented_mask_image = img.copy() * grabcut_mask[:, :, np.newaxis]
img_2=img-segmented_mask_image
blur_1=cv.GaussianBlur(segmented_mask_image,(19,19),cv.BORDER_CONSTANT)
background_img=img_2+blur_1
blur_2=cv.GaussianBlur(img_2,(19,19),cv.BORDER_CONSTANT)
foreground_img=blur_2+segmented_mask_image
```



In [ ]:
```
blur_img = cv.GaussianBlur(segmented_mask_image, (0, 0), 100)
sharpen = cv.addWeighted(segmented_mask_image, 1.5, blur_img, -0.5, 0)
enhanced_img=sharpen+img_2
```

Out[ ]: ([], [])



c) Flower is the segmented mask of the given image. So when we enhance the image with the blurred background edges will get dark borders.