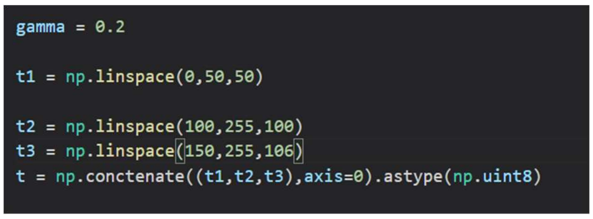
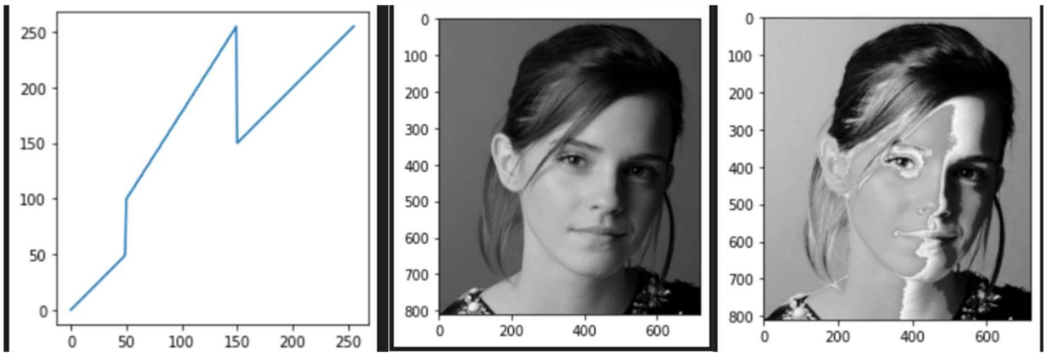
Question 1 Intensity Transformation

To implement the intensity transformation mentioned in give graph, divide color range into three ranges and mapped them into different ranges to fit the graph.



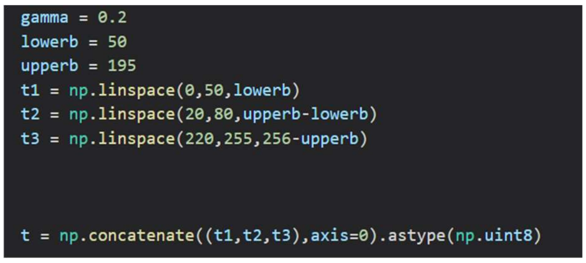
```
gamma = 0.2

t1 = np.linspace(0,50,50)

t2 = np.linspace(100,255,100)
t3 = np.linspace(150,255,106)
t = np.conctenate((t1,t2,t3),axis=0).astype(np.uint8)
```



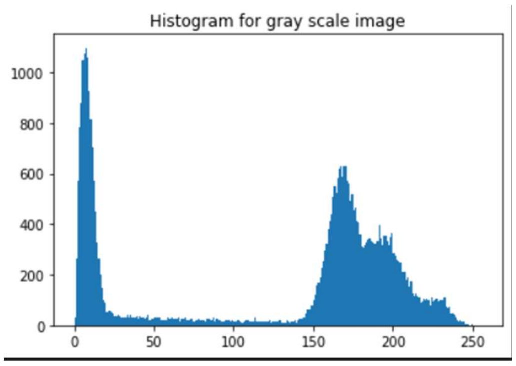
When comparing original image and result, 0-50 and 150-255 ranges are doesn’t changed and 50-150 range is mapped into 100-200(those pixels gets more whiter than before).

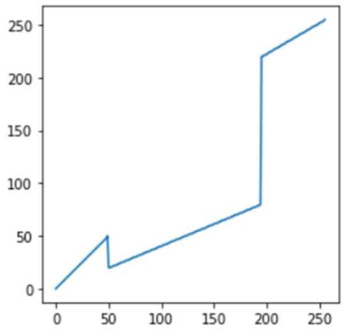
Question 2 – Intensity Transformation for brain images

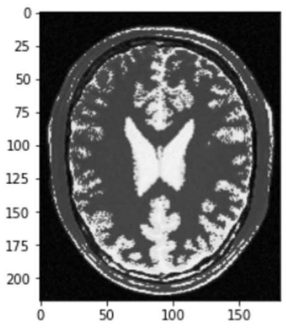
To filter-out the white matter in the brain image, lighter pixel values moved into more lighter range and midrange pixel values decreased with use of histogram of original image.



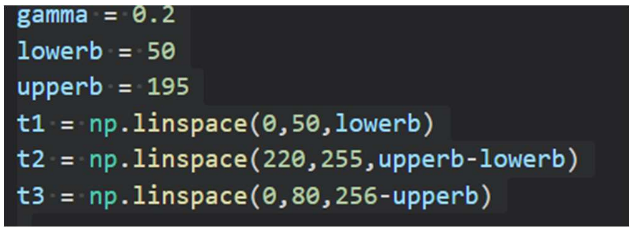
```
gamma = 0.2
lowerb = 50
upperb = 195
t1 = np.linspace(0,50,lowerb)
t2 = np.linspace(20,80,upperb-lowerb)
t3 = np.linspace(220,255,256-upperb)



t = np.concatenate((t1,t2,t3),axis=0).astype(np.uint8)
```



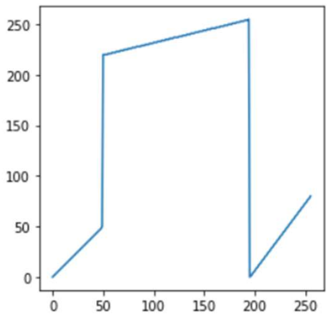


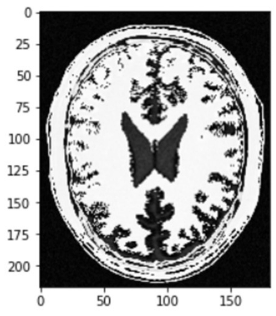


To filter-out the gray matter in the brain image, midrange pixel values moved into more lighter range and lighter pixel values decreased with use of histogram of original image.



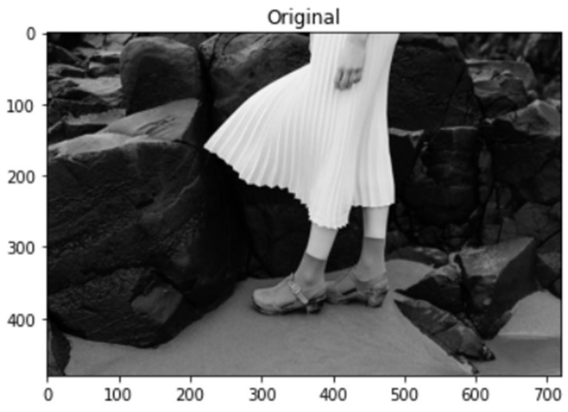
```
gamma = 0.2
lowerb = 50
upperb = 195
t1 = np.linspace(0,50,lowerb)
t2 = np.linspace(220,255,upperb-lowerb)
t3 = np.linspace(0,80,256-upperb)
```

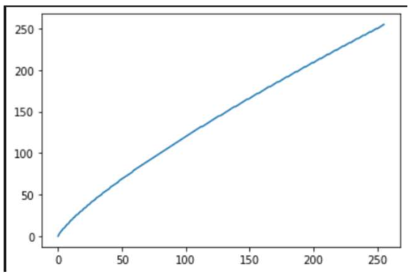


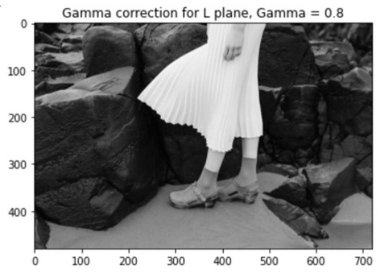


Question 3 – Gamma correction

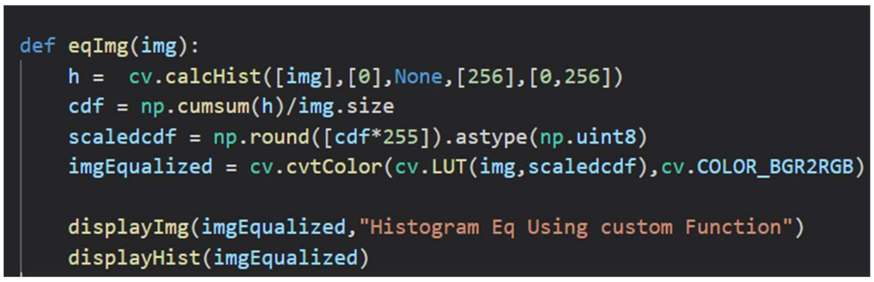
When increasing gamma correction value, darker pixel getting more darker. So to enhance the original image and clear out the stone colors more clearer, gamma value should be lesser than 1.

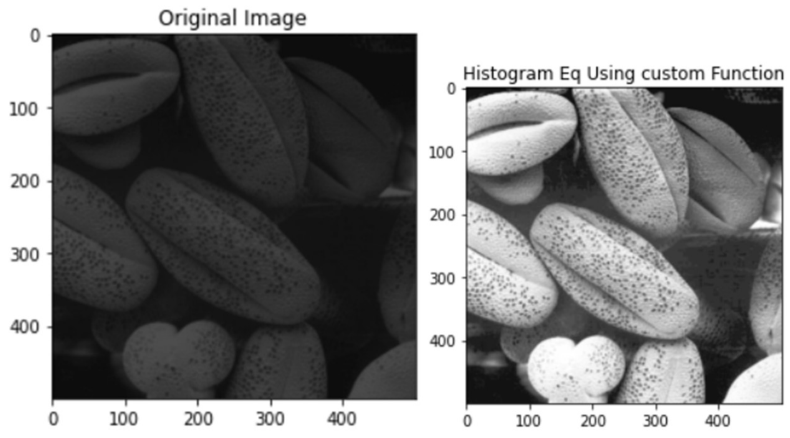




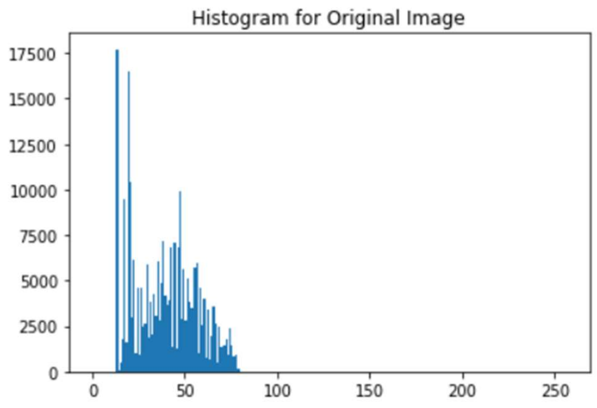


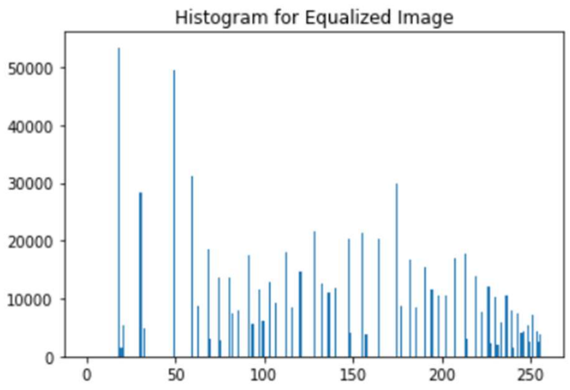
Question 4 Histogram Equalization


```
def eqImg(img):
    h =  cv.calcHist([img],[0],None,[256],[0,256])
    cdf = np.cumsum(h)/img.size
    scaledcdf = np.round([cdf*255]).astype(np.uint8)
    imgEqualized = cv.cvtColor(cv.LUT(img,scaledcdf),cv.COLOR_BGR2RGB)

    displayImg(imgEqualized,"Histogram Eq Using custom Function")
    displayHist(imgEqualized)
```

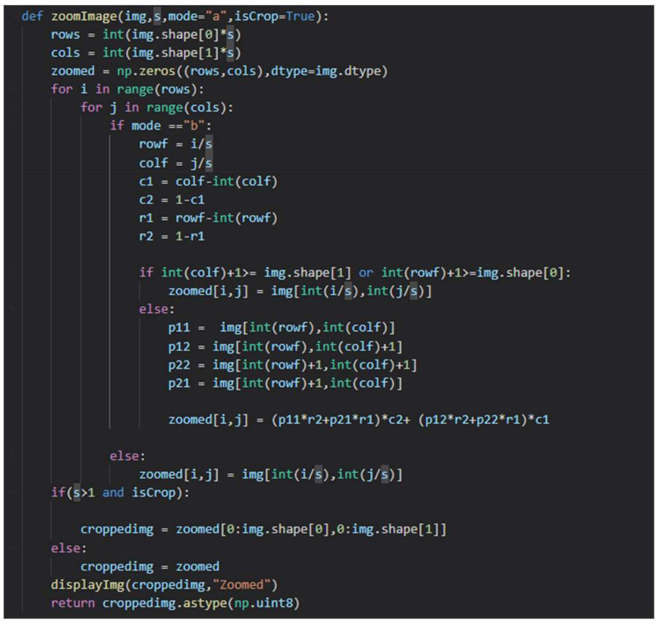


Histogram of the original image is closer two darker side (0-100). In the ideal situation to equalize the histogram, all the color count should be same. It means histogram should be flat. But in practical scenario, histogram only can be nearly flat. To define the equalization function, cumulative distribution function of the image pixel data is used. Then image pixels are replaced using CDF.



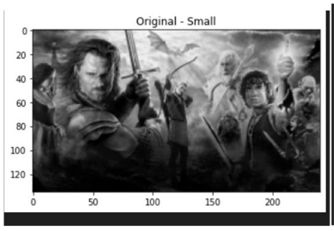


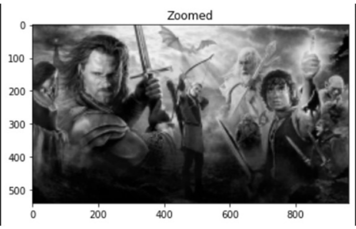
Q5 Zooming Algorithm

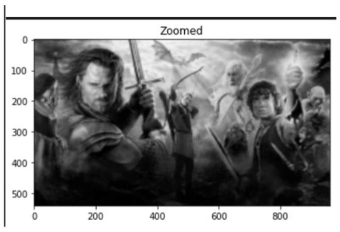


```
def zoomImage(img,s,mode="a",isCrop=True):
    rows = int(img.shape[0]*s)
    cols = int(img.shape[1]*s)
    zoomed = np.zeros((rows,cols),dtype=img.dtype)
    for i in range(rows):
        for j in range(cols):
            if mode =="b":
                rowf = i/s
                colf = j/s
                c1 = colf-int(colf)
                c2 = 1-c1
                r1 = rowf-int(rowf)
                r2 = 1-r1

                if int(colf)+1>= img.shape[1] or int(rowf)+1>=img.shape[0]:
                    zoomed[i,j] = img[int(i/s),int(j/s)]
                else:
                    p11 =  img[int(rowf),int(colf)]
                    p12 = img[int(rowf),int(colf)+1]
                    p22 = img[int(rowf)+1,int(colf)+1]
                    p21 = img[int(rowf)+1,int(colf)]

                    zoomed[i,j] = (p11*r2+p21*r1)*c2+ (p12*r2+p22*r1)*c1

            else:
                zoomed[i,j] = img[int(i/s),int(j/s)]
    if(s>1 and isCrop):

        croppedimg = zoomed[0:img.shape[0],0:img.shape[1]]
    else:
        croppedimg = zoomed
    displayImg(croppedimg,"Zoomed")
    return croppedimg.astype(np.uint8)
```

To define the zoom function, two scaling algorithm is used, in the function argument "a" for nearest-neighbor and "b" for bilinear interpolation.





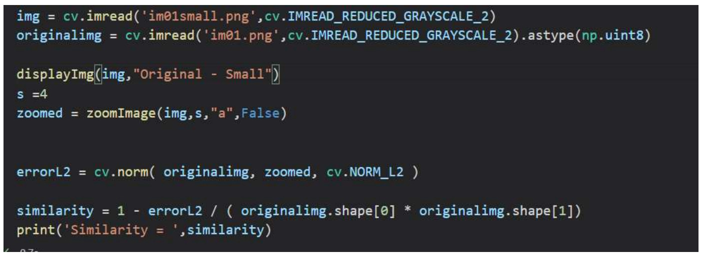


Original Image　　Nearest Neighbor　　Bilinear Interpolation

When comparing those images using normalized sum of squared method, In most of the time bilinear interpolation method is better than Nearest Neighbour method.

0.976 similarity with original image for Nearest Neighbour method

0.979 similarity with original image for interpolation method

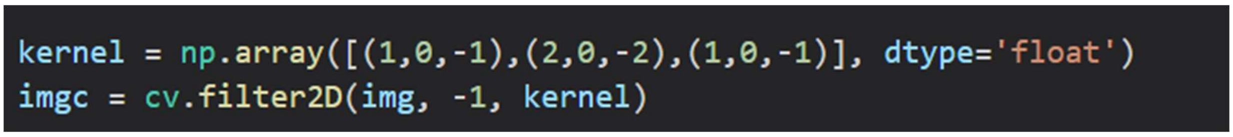


```
img = cv.imread('im01small.png',cv.IMREAD_REDUCED_GRAYSCALE_2)
originalimg = cv.imread('im01.png',cv.IMREAD_REDUCED_GRAYSCALE_2).astype(np.uint8)

displayImg(img,"Original - Small")
s =4
zoomed = zoomImage(img,s,"a",False)


errorL2 = cv.norm( originalimg, zoomed, cv.NORM_L2 )

similarity = 1 - errorL2 / ( originalimg.shape[0] * originalimg.shape[1])
print('Similarity = ',similarity)
```

Q6(a) Sobel Filter using filter2D

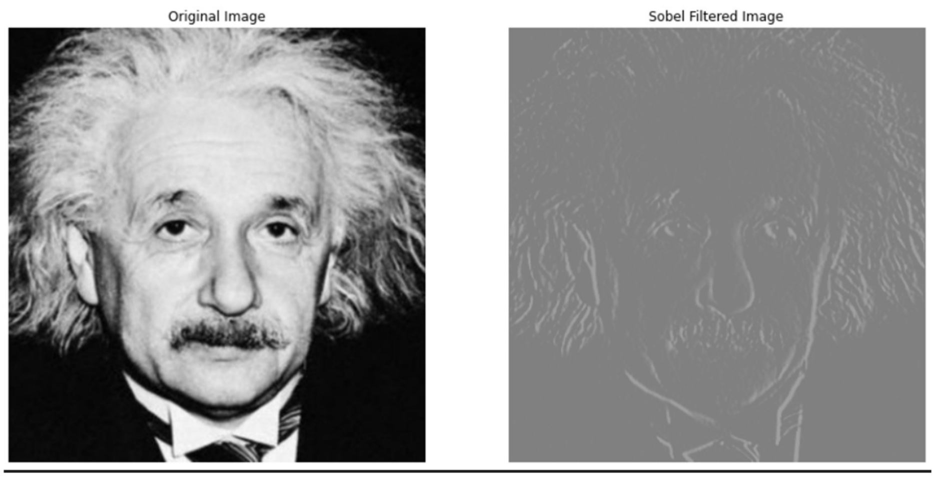
In this technique, convolution with image and a kernel is used to filterout edges of the image, when convoluting with kernel, similar color areas canceled out to to zero value, due to kernel is inversely symmetrical. When kernel is vertically inverse symmetrical, then vertical edges can be detected. Using above technique similarly to horizontally, horizontal edges can be detected. So adding those results. Full edges can be detected.

Code:



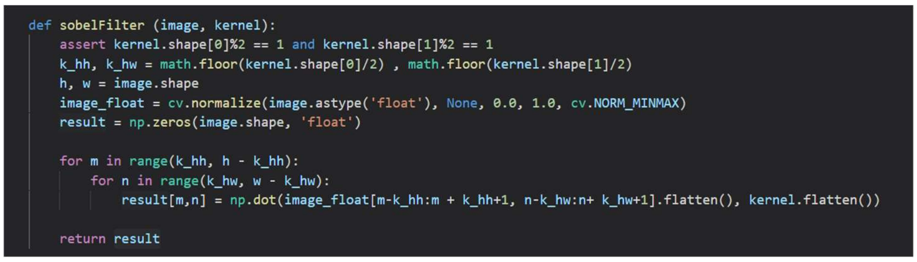
```
kernel = np.array([(1,0,-1),(2,0,-2),(1,0,-1)], dtype='float')
imgc = cv.filter2D(img, -1, kernel)
```

Result:



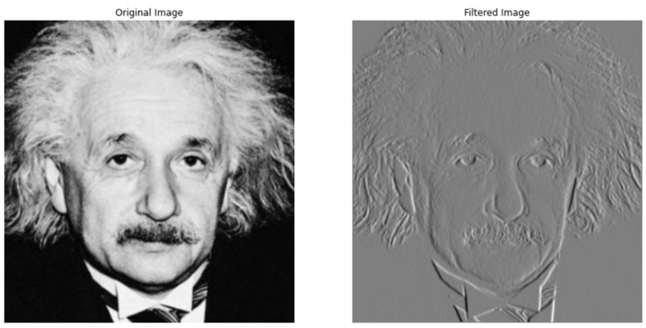
Q6(b) Sobel Filter using custom function

Code:



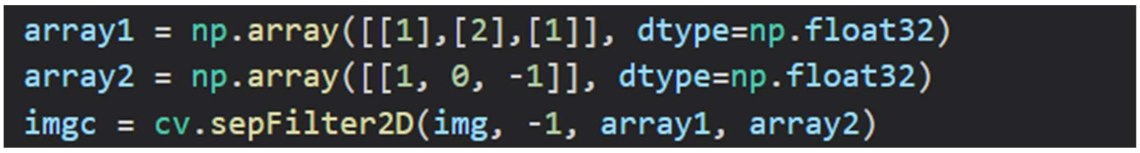
```
def sobelFilter (image, kernel):
    assert kernel.shape[0]%2 == 1 and kernel.shape[1]%2 == 1
    k_hh, k_hw = math.floor(kernel.shape[0]/2) , math.floor(kernel.shape[1]/2)
    h, w = image.shape
    image_float = cv.normalize(image.astype('float'), None, 0.0, 1.0, cv.NORM_MINMAX)
    result = np.zeros(image.shape, '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(), kernel.flatten())

    return result
```

Result:



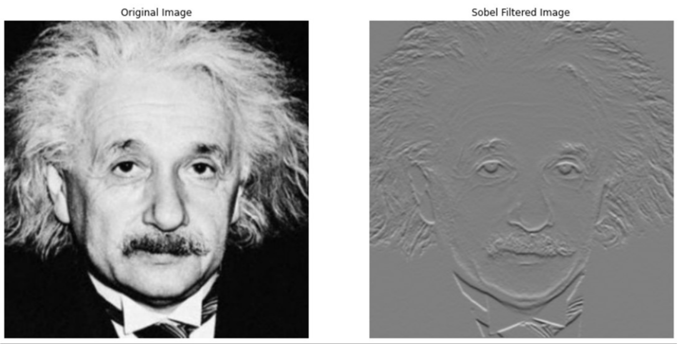
Q6(c) Sobel Filter using sepfilter2D

Code:



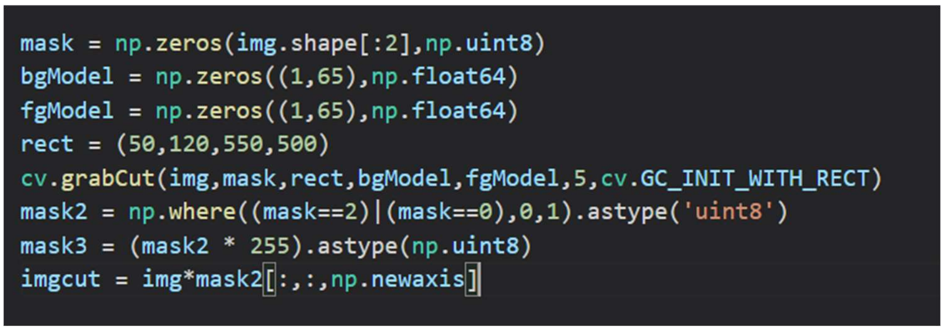
```
array1 = np.array([[1],[2],[1]], dtype=np.float32)
array2 = np.array([[1, 0, -1]], dtype=np.float32)
imgc = cv.sepFilter2D(img, -1, array1, array2)
```

Result:



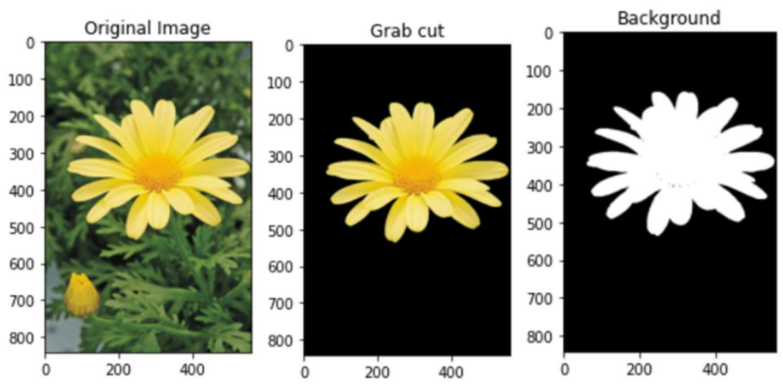
Q7(a) Segmentation of Image

Grabcut function detect the focused area and put a mask on those focused pixels, an then unfocused background and focused foreground can be filterout from image.



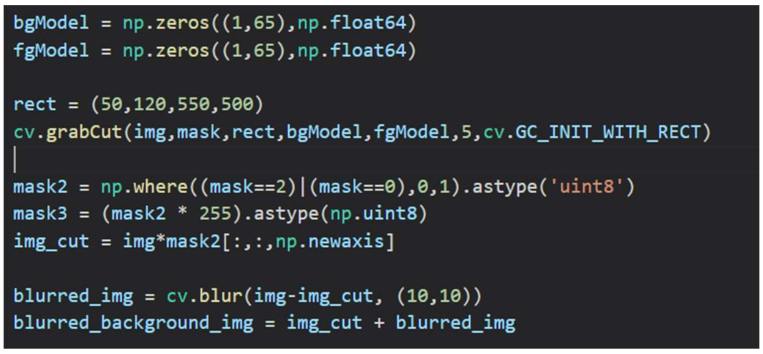
```
mask = np.zeros(img.shape[:2],np.uint8)
bgModel = np.zeros((1,65),np.float64)
fgModel = np.zeros((1,65),np.float64)
rect = (50,120,550,500)
cv.grabCut(img,mask,rect,bgModel,fgModel,5,cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
mask3 = (mask2 * 255).astype(np.uint8)
imgcut = img*mask2[:,:,np.newaxis]
```

Result:

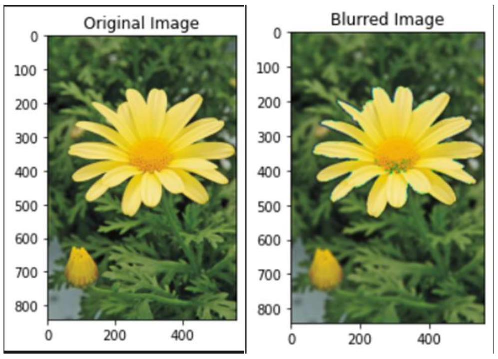


Q7(b) Blur background of Image

To blur the background of the image, at first, put mask on foreground using grabcut method. And then blur the background mask and add to forward mask.



```
bgModel = np.zeros((1,65),np.float64)
fgModel = np.zeros((1,65),np.float64)

rect = (50,120,550,500)
cv.grabCut(img,mask,rect,bgModel,fgModel,5,cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
mask3 = (mask2 * 255).astype(np.uint8)
img_cut = img*mask2[:,:,np.newaxis]

blurred_img = cv.blur(img-img_cut, (10,10))
blurred_background_img = img_cut + blurred_img
```



Q7(c) In the masked background image, foreground flower area is black color. When putting blur effect on background mask, flower edges will mixed up with those black area and these edges getting discontinues with foreground mask and they get little bit darker.

Github-link : https://github.com/PabasaraDilshan/EN2550