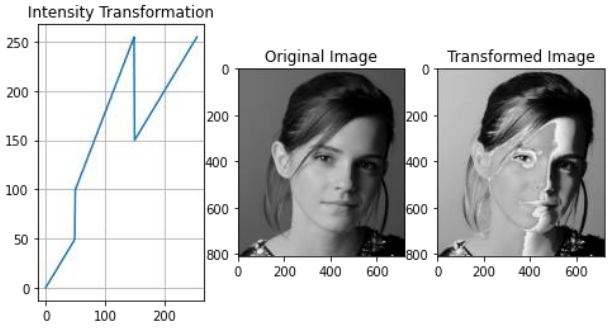
Name: Fernando I.A.M.D.
Index No.: 190172K

- **Question 1**
  - **Code and Results**

```
img = cv2.imread("emma_gray.jpg",cv2.IMREAD_GRAYSCALE)
assert img is not None

t1 = np.linspace(0,50,50)
t2 = np.linspace(100,255,100)
t3 = np.linspace(150,255,106)

t = np.concatenate((t1,t2,t3),axis=0).astype(np.uint8)
assert len(t)==256
transformed_img = cv2.LUT(img,t)
```

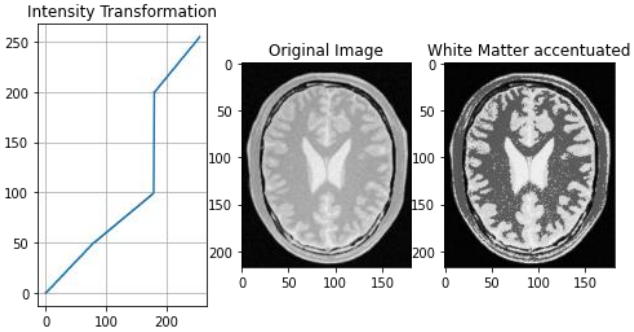


  - **Discussion**

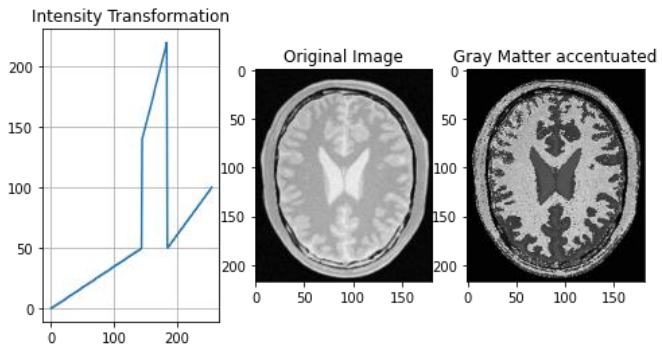
For pixels in the range 0-50 and 150-255 there is no transformation. Input pixel value is taken as output. For the pixels in the range 100-150 there is a linear transformation added. Therefore, input pixel value in x-axis is given by the output value in y-axis.

- **Question 2**
  - **Code and Results**

```
img = cv2.imread("brain_proton_density_slice.png",cv2.IMREAD_GRAYSCALE)
assert img is not None

t1 = np.linspace(0,50,80)
t2 = np.linspace(50,100,100)
t3 = np.linspace(200,255,76)

t = np.concatenate((t1,t2,t3),axis=0).astype(np.uint8)
assert len(t)==256
transformed_img = cv2.LUT(img,t)
```




```
img = cv2.imread("brain_proton_density_slice.png",cv2.IMREAD_GRAYSCALE)
assert img is not None

t1 = np.linspace(0,50,145)
t2 = np.linspace(140,220,40)
t3 = np.linspace(50,100,71)

t = np.concatenate((t1,t2,t3),axis=0).astype(np.uint8)
assert len(t)==256
transformed_img  = cv2.LUT(img,t)
```

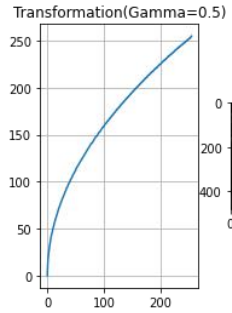


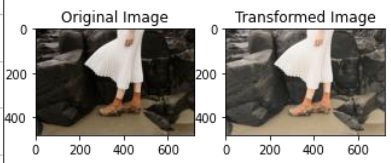
➢ **Discussion**

By considering the histogram of the original image pixel values corresponding to white matter was identified as 186-255 and the pixel values corresponding to gray matter was identified as 146-185. Therefore, to accentuate those pixel values, corresponding pixel ranges were given a transformation to give out a higher pixel value and other pixel values were given a transformation to output a lower pixel value. Accentuated and non-accentuated areas have were given a gradient to not harm the details of the image.

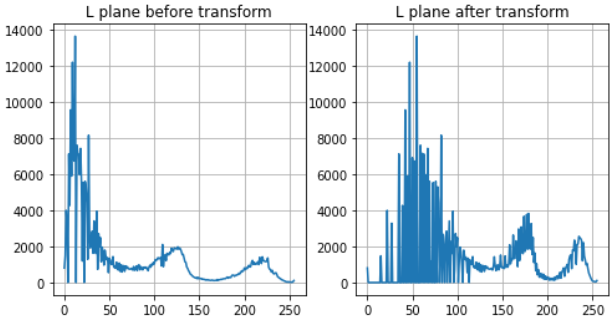
- **Question 3**
  - ➢ Code and results

```
######################  a  ######################
img = cv2.imread("highlights_and_shadows.jpg")
img_RGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
img_Lab = cv2.cvtColor(img,cv2.COLOR_BGR2Lab)
L,a,b = cv2.split(img_Lab)

gamma = 0.5
t = np.array([(p/255)**gamma*255 for p in range(0,256)]).astype("uint8")
transformed_L = cv2.LUT(L,t)

img_L_Increased = cv2.merge([transformed_L,a,b])
img_L_Increased = cv2.cvtColor(img_L_Increased,cv2.COLOR_Lab2RGB)
```






```
######################  b  ######################
hist_f = cv2.calcHist([L],[0],None,[256],[0,256])
hist_g = cv2.calcHist([transformed_L],[0],None,[256],[0,256])
```



➢ **Discussion**

Here Lab color space is considered. Initially the BGR image is converted to Lab color space. Since the gamma correction is done to the L plane the Lab image array is split into separate plane. Then the gamma correction is done to the L plane. Finally, the gamma corrected L plane is merged with a, b planes. And converted back to RGB color space for displaying.

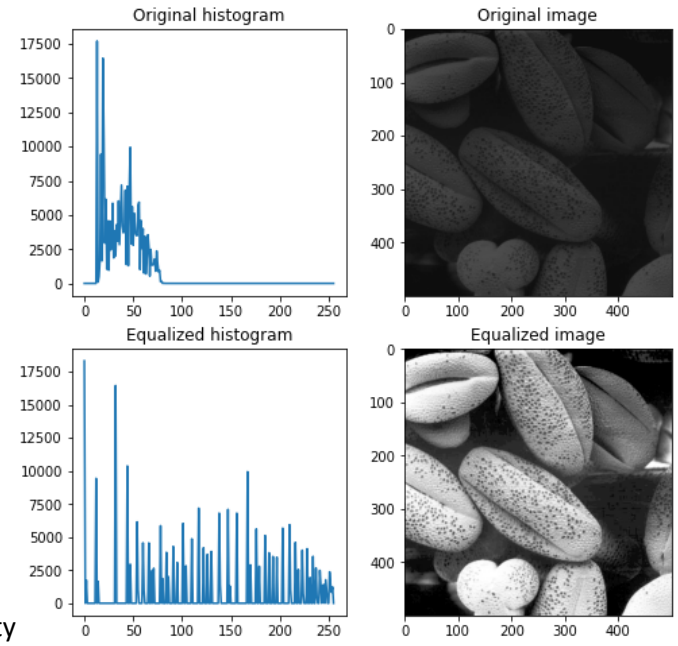
Here by gamma correction darker areas of the image is transformed into a wider region. Therefore, we will be able to see more details in the darker area.

The comparison shows original image and gamma corrected image with gamma=0.5.

By increasing the gamma value beyond 0.5 the detail of darker areas will be more and more identical to the original image since the transformation becomes linear. By further decreasing gamma below 0.5 the low valued pixels will get transformed to higher values resulting in the dark areas being more brighter and less dark.

- Question 4
  - **Code and Results**

```
def hist_eq(img):
    histOrig, bins = np.histogram(img.flatten(), 256, [0, 255])
    cdf = histOrig.cumsum()
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
    cdf = np.ma.filled(cdf_m, 0)
    imgEq = cdf[img.astype('uint8')]
    histEq, bins2 = np.histogram(imgEq.flatten(), 256, [0, 256])
    return imgEq, histOrig, histEq

img1 = cv2.imread("shells.png",cv2.IMREAD_GRAYSCALE)
g,hist_f,hist_g = hist_eq(img1)
```



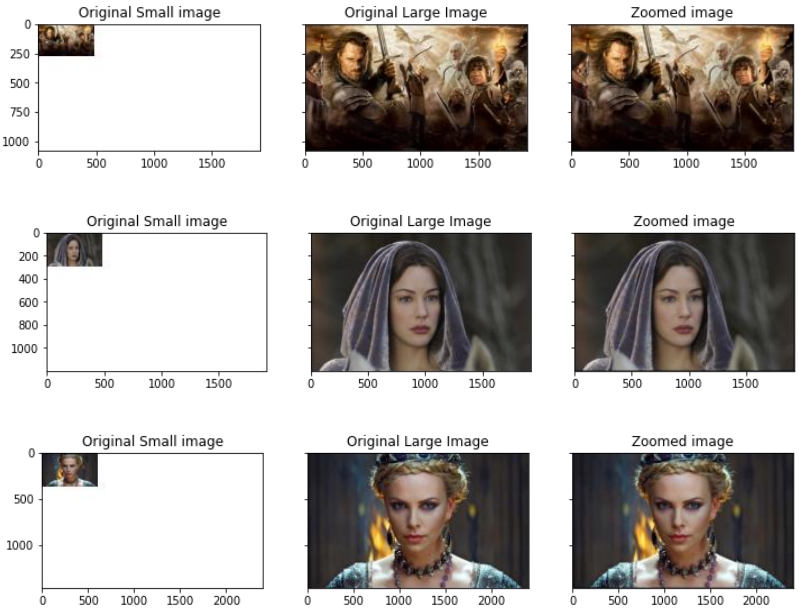
  - **Discussion**

Histogram of an image provides us a visual to the distribution of the pixels values of the image. Pixels values of the image vary in the range 0-255. As seen in the top left plot, some pixel values have a high frequency, and some have a very low frequency. By histogram equalization we spread out the most frequent pixel intensity values. It's like starching the histogram.

Here a gray scale image was considered for equalization. If and RGB image was used equalization has to be done in all three planes.

- **Question 5 a)**
  - **Code and results**

```
################### a - nearest-neighbor ###################
def zoom(img,scale):
    rows = int(scale*img.shape[0])
    cols = int(scale*img.shape[1])
    zoomed = np.zeros((rows,cols,3),dtype=img.dtype)
    gap = scale
    if scale<1:
        gap = 1
    for i in range(0,rows-gap):
        for j in range(0,cols-gap):
            for k in range(0,3):
                zoomed[i,j,k] = img[int(round(i/scale)),int(round(j/scale)),k]
    zoomed_RGB = cv2.cvtColor(zoomed,cv2.COLOR_BGR2RGB)
    return zoomed_RGB

img = cv2.imread("a1q5images\im01small.png")
img_large = cv2.imread("a1q5images\im01.png")
img_RGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
img_large_RGB = cv2.cvtColor(img_large,cv2.COLOR_BGR2RGB)
scale = 4
img_zoomed = zoom(img,scale)
```

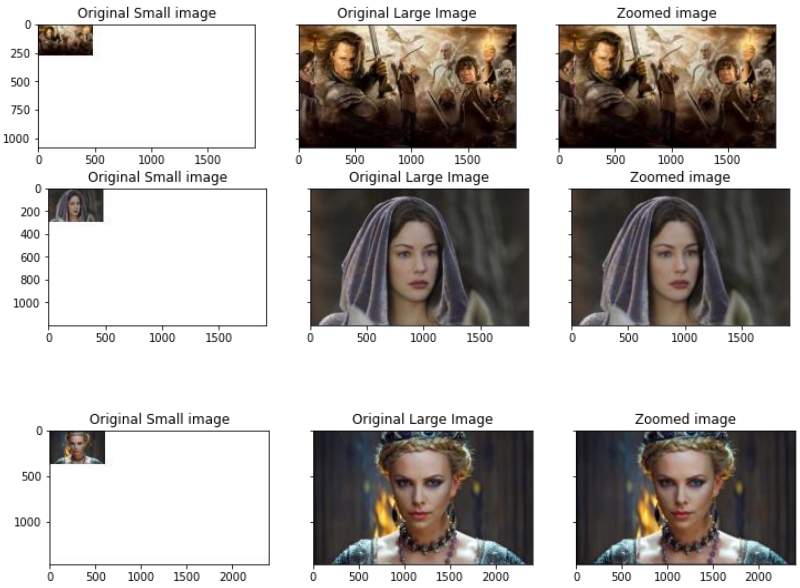


SSD Values = im01 -> 21.72
im02-> 11.45

The zoom function uses nearest neighbor method to zoom the image. Here the pixel values in the zoomed image are decided by considering the nearest pixel. Initially an empty image array is created. The (i, j, k) th pixel is decided by considering (i/scale, j/scale, k)th pixel in the original image, where scale is the scaling factor. When (i/scale, j/scale) are not integers we need to see to which neighboring pixel it is closest to. Therefore, first the (i/scale, j/scale) values are rounded and then integer value is taken. The images shown are zoomed by a factor of 4.

- **Question 5 b)**
  - **Code and results**

```
################### b - bilinear interpolation ###################
import math
def zoom(img,scale,img_large):
    rows = int(scale*img.shape[0])
    cols = int(scale*img.shape[1])
    zoomed = np.zeros((rows,cols,3),dtype=img.dtype)
    gap = scale
    if scale<1:
        gap = 1
    ssd = 0
    pad_img = cv2.copyMakeBorder(img, 0, 1, 0, 1, cv2.BORDER_REPLICATE)
    for i in range(0,rows):
        for j in range(0,cols):
            for k in range(0,3):
                x = i/scale - math.floor(i/scale)
                y = j/scale - math.floor(j/scale)
                tl = pad_img[math.floor(i/scale),math.floor(j/scale),k]
                bl = pad_img[math.floor(i/scale),math.ceil(j/scale),k]
                tr = pad_img[math.ceil(i/scale),math.floor(j/scale),k]
                br = pad_img[math.ceil(i/scale),math.ceil(j/scale),k]
                avg_pix = int((br*(1-x)+bl*(x))*(1-y)+(tr*(1-x)+tl*(x))*(y))
                zoomed[i,j,k] = avg_pix
                ssd += (avg_pix - img_large[i,j,k])**2
    ssd = np.sqrt(ssd/(rows*cols*3))
    zoomed_RGB = cv2.cvtColor(zoomed,cv2.COLOR_BGR2RGB)
    return zoomed_RGB,ssd

img = cv2.imread("a1q5images\im01small.png")
img_large = cv2.imread("a1q5images\im01.png")
img_RGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
img_large_RGB = cv2.cvtColor(img_large,cv2.COLOR_BGR2RGB)
img_zoomed,ssd = zoom(img,4,img_large)
rows = int(scale*img.shape[0])
cols = int(scale*img.shape[1])
print("SSD=",ssd/(rows*cols))
```



SSD Values = im01 -> 6.27
im02-> 4.04

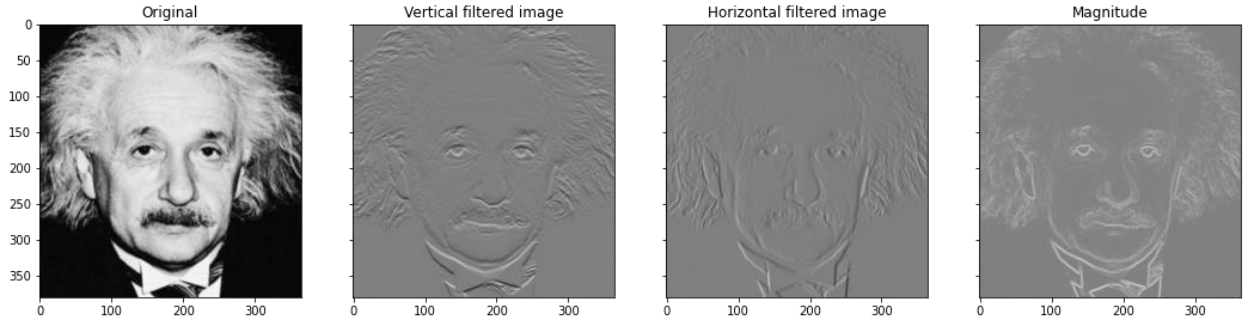
- **Discussion**

The zoom function uses nearest bilinear interpolation method to zoom the image. In this method the (i,j,k)th pixel value is decided by considering the four neighboring pixels of (i/scale,j/scale,k)th pixel. The four neighboring pixel values are interpolated considering its pixel value and distance from the (i/scale,j/scale,k) value. To find the four neighboring pixels floor and ceil functions from the math module is used. Bi-linear interpolation method is more reliable than nearest neighbor method because the pixel values which have float values when divided by scale are being calculated by interpolating. Therefore, sudden pixel value changes will not be visible in the zoomed image.

- **Question 6**
  - **Code and results**

```
######################  a  ######################
img = cv2.imread("einstein.png",cv2.IMREAD_GRAYSCALE).astype(np.float32)
assert img is not None

#sobel verical
kernelver = np.array([(-1,-2,-1),(0,0,0),(1,2,1)],dtype='float')
imgv = cv2.filter2D(img,-1,kernelver)
#sobel horizontal
kernelhor = np.array([(-1,0,1),(0,0,0),(-2,0,2)],dtype='float')
imgh = cv2.filter2D(img,-1,kernelhor)
#sobel magnitude
mag = np.sqrt(imgv**2+imgh**2)
```

Sobel operator is used for edge detection. Here the image is convolved with a 3x3 matrix. Sobel vertical filter is used to detect horizontal edges and Sobel horizontal filter is used to detect vertical edges. By considering magnitude of the two filters we can detect both vertical and horizontal edges. filter2D function is used to do the convolution process between the image and the kernel.

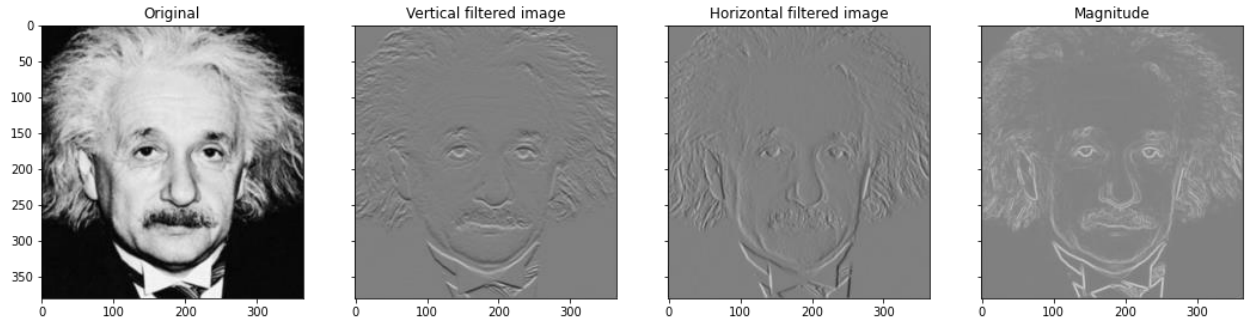


As shown in the outputs its clearly visible that when the vertical filter is used horizontal edges have been detected and when horizontal filter is used vertical edges have been detected. And the magnitude image has detected both types.

```
#####################  b  #####################
def kernalConv(image,kernal):
  rows = image.shape[0]
  cols = image.shape[1]

  if kernal.shape[0]>1:
      gap = int(kernal.shape[0]/2)
  else:
      gap = 0
  conv = np.zeros((rows,cols),dtype=image.dtype)

  for i in range(gap ,rows - gap):
    for j in range( gap, cols - gap):
      conv [i , j] = np.sum( np.multiply( kernal, image[i-gap:i+gap+1, j-gap:j+gap+1]))
  return conv
```

For part **b** a custom function **kernalConv** was built to carry out the process of convolution.

The convolution process is carried out by using two nested for loops.

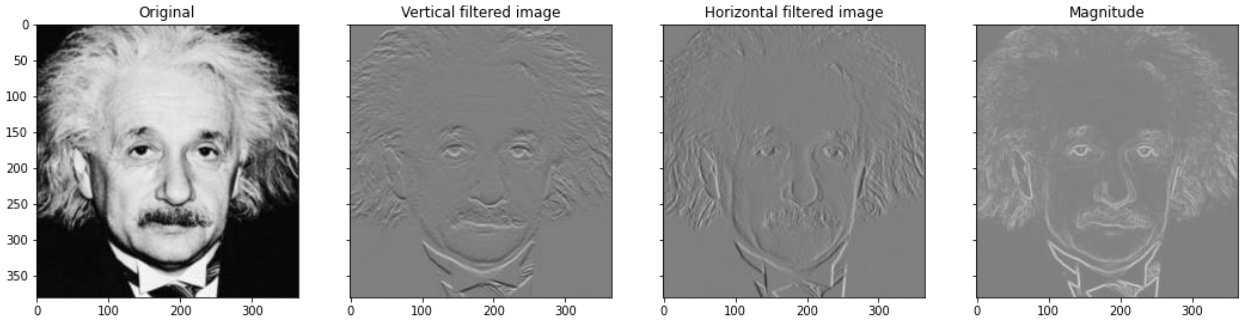


Comparing this set of manually filtered images with the previous set of images which were filtered using filter2D function both are visually almost the same.

```
#####################  c  #####################
img = cv2.imread("einstein.png",cv2.IMREAD_GRAYSCALE).astype(np.float32)

ker1 = np.array([[1],[2],[1]],dtype='float')
img1 = cv2.filter2D(img,-1,ker1)

ker2 = np.array([[1,0,-1]],dtype='float')
img2 = cv2.filter2D(img1,-1,ker2)

ker3 = np.array([[-1],[0],[1]],dtype='float')
img3 = cv2.filter2D(img,-1,ker3)

ker4 = np.array([[1,2,1]],dtype='float')
img4 = cv2.filter2D(img3,-1,ker4)

mag = np.sqrt(img2**2+img4**2)
```

In the c part the associative property of convolution is considered. In the question by convolving the provided two arrays the Sobel vertical filter can be obtained. Initially we can use the filter2D function and convolve the image with the first array and then convolve its output with the second array. This will be same as applying Sobel vertical filter because of the associative property of convolution.

Similarly, we can also create Sobel horizontal filter by convolving two arrays. And with these two arrays and the associative property of convolution we can output a Sobel horizontal filtered image.



As seen here, the results are identical to the results obtained in the previous two parts.

- **Question 7**

```
#######################  a,b  ########################
img = cv2.imread("daisy.jpg")
imgRGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
mask = np.zeros(img.shape[:2],np.uint8)

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

rect = (50,150,520,400)

cv2.grabCut(img,mask,rect,bgModel,fgModel,5,cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2) | (mask==0) , 0,1).astype('uint8')
mask3 = np.where((mask==1)|(mask==3),0,1).astype('uint8')

imgcut = img * mask2[:,:,np.newaxis]
imgback = img * mask3[:,:,np.newaxis]
imgcutRGB = cv2.cvtColor(imgcut,cv2.COLOR_BGR2RGB)
imgbackRGB = cv2.cvtColor(imgback,cv2.COLOR_BGR2RGB)

imgbackBlur = cv2.GaussianBlur(imgback,(21,21),0)
imgenhanced = cv2.add(imgcut ,imgbackBlur)
imgenhancedRGB = cv2.cvtColor(imgenhanced,cv2.COLOR_BGR2RGB)
```

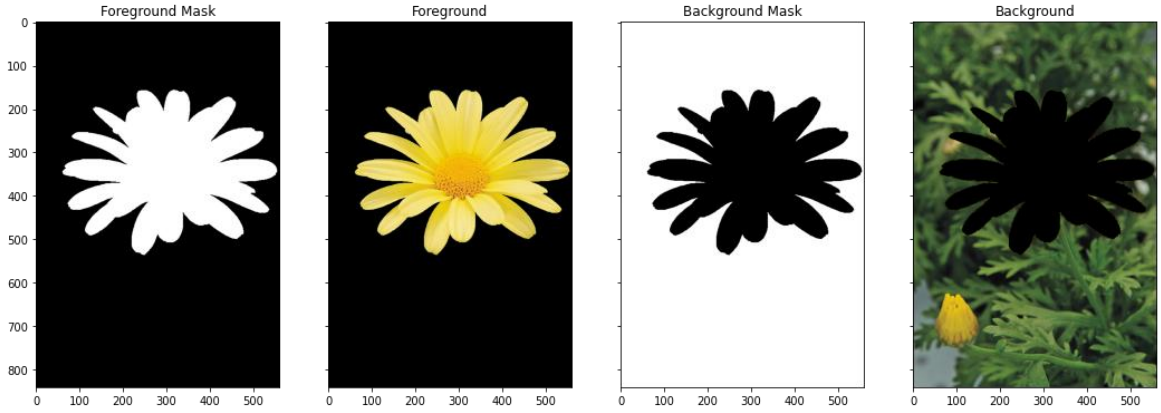
  - **Code and results**

    Here the OpenCV grabCut function is used to obtain the segmentation mask. Here rectangle coordinates need to select such that the parts of the image which needs to be focused (foreground) is covered. The grabCut algorithm considers everything outside the rectangle as sure background. Next by considering the pixel value distribution inside and outside the rectangle two PDFs are created for foreground and background. These two pdf models are used to decide whether each pixel belong to foreground or background.

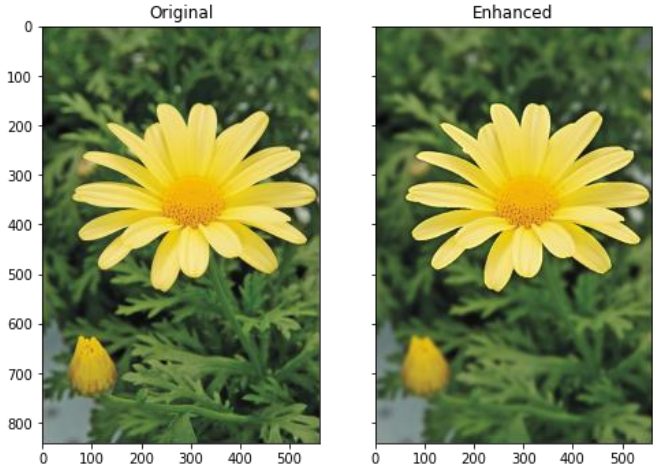
The final output from grabCut function contains only four values. 0 for sure foreground, 1 for sure background, 2 for suspected foreground and 3 for suspected background.

By extracting values where the final output is 0 or 2, the foreground mask is obtained. And by extracting values where the final output is 1 or 3 the background mask is obtained.

By multiplying the original image with the foreground mask, the Foreground image is obtained.
By multiplying the original image with the background mask, the Background image is obtained



Now in order to blur the background, the Background image is blurred using Gaussian blur function with kernel size of (21,21). Then the blurred background image is added to the Foreground image which results in an enhanced image with a substantially blurred background.



Here only the bloomed flower was considered as the foreground. By increasing the size of the rectangle, we can create an output image with the other small buds to in the foreground.

By comparing these two images its clearly visible that the Bloomed daisy has been focused and the background is blurred substantially.

c) Here the background image is getting blurred. The kernel smoothens the edges between colored area and removed foreground area by convolution and giving pixel values closer to majority color near the considered pixel. Therefore, areas in the background closer to foreground also gets darker. When the kernel size for the blur increases this dark area increases because more pixels are now smoothened with the removed foreground parts.