Department of Electronic & Telecommunication Engineering

University of Moratuwa

EN3160 - Image Processing and Machine Vision

Intensity Transformations and Neighborhood Filtering

(Assignment 01 - Report)

Image-Processing/Intensity Transformations and Neighborhood Filtering at main · kavindukalinga/Image-Processing (github.com)

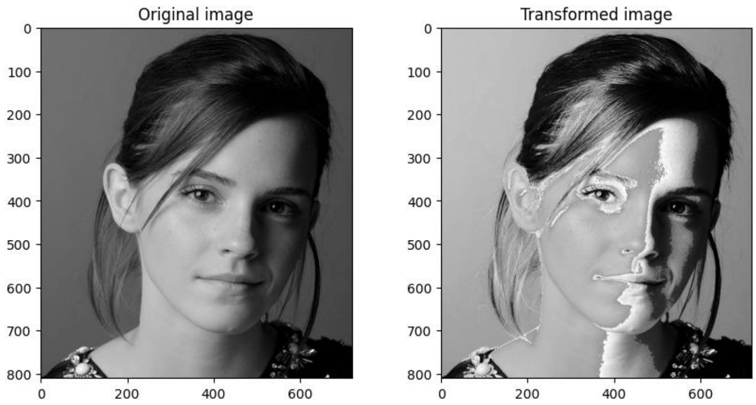
200087A Chandrasiri Y.U.K.K.

Question 01

Intensity Transformation Function:

```python
t1 = np.linspace(0 , 50 , 51)
t2 = np.linspace(100 , 255 , 150-50)
t3 = np.linspace(150 , 255 , 255-150)
transformation_function = np.concatenate((t1,t2,t3), axis=0)

Transformed_image = cv.LUT(Original_image, transformation_function)
```

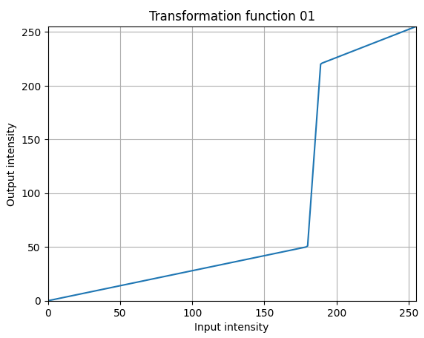
Results:

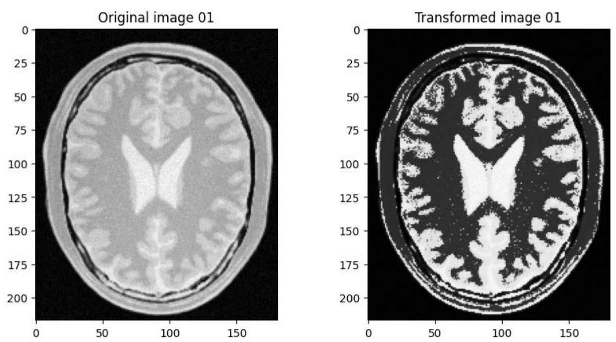


Question 02

Transform 01:

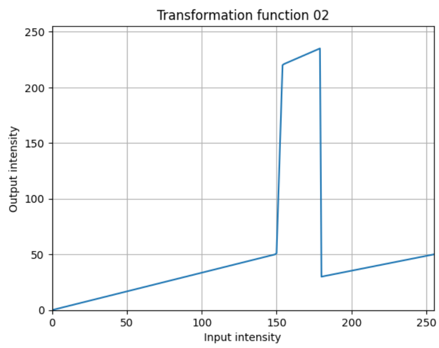
```python
t1 = np.linspace(0 , 50 , 180)
t2 = np.linspace(51 , 220 , 10)
t3 = np.linspace(221 , 255 , 66)
transform1 = np.concatenate((t1,t2,t3), axis=0)
```

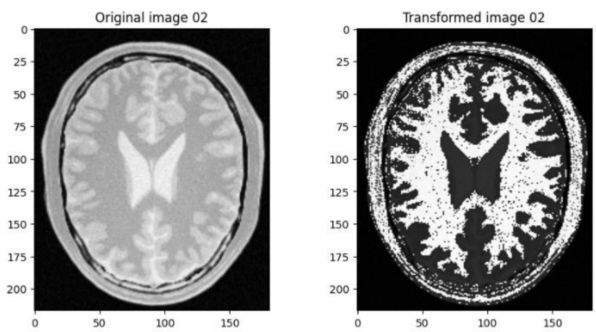




Transform 02:

```python
t1 = np.linspace(0 , 50 , 150)
t2 = np.linspace(51 , 220 , 5)
t3 = np.linspace(221 , 235 , 25)
t4 = np.linspace(30 , 50 , 76)
transform2 = np.concatenate((t1,t2,t3,t4), axis=0)
```



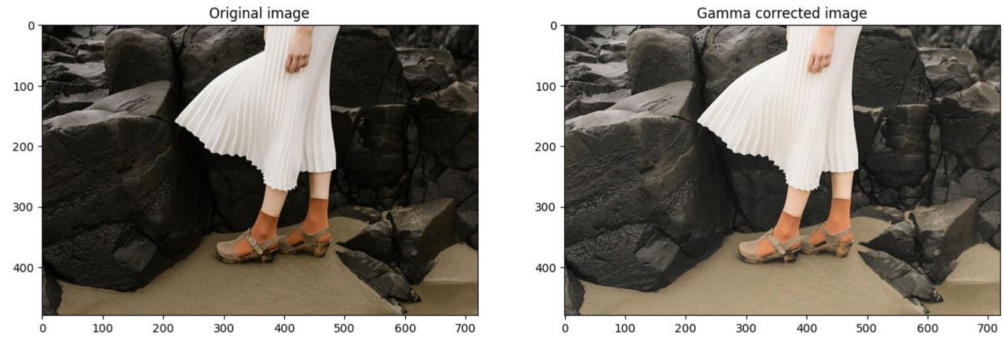


Question 03

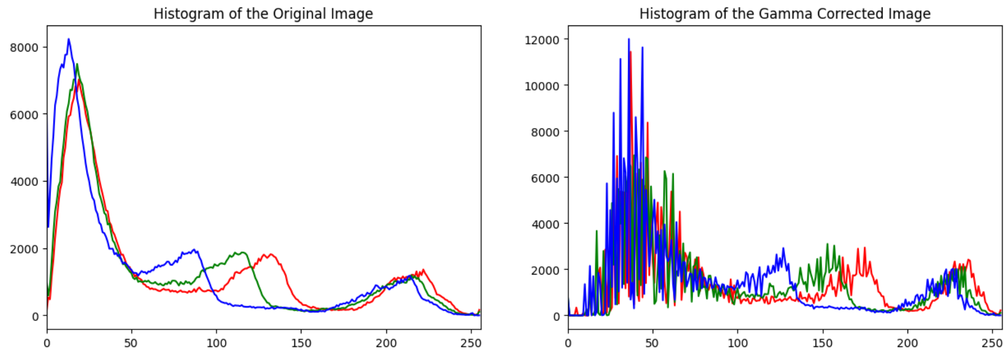
$\gamma = 0.6$

```python
image3 = cv.cvtColor(original_image, cv.COLOR_BGR2LAB)
Lplane, Aplane, Bplane = cv.split(image3)

gamma = 0.6
gamma_correction = np.array([((i/255)**gamma)*255 for i in np.arange(0,256)]).astype('uint8')

Lplane  = cv.LUT(Lplane,gamma_correction)
gamma_corrected_image = cv.merge((Lplane,Aplane,Bplane))
gamma_corrected_image = cv.cvtColor(gamma_corrected_image, cv.COLOR_LAB2RGB)
```

Image Transformation:

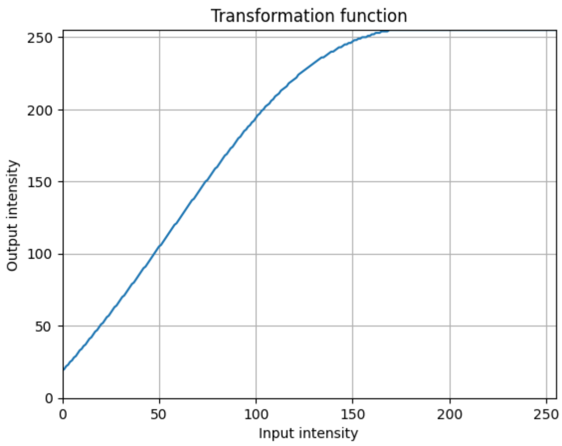


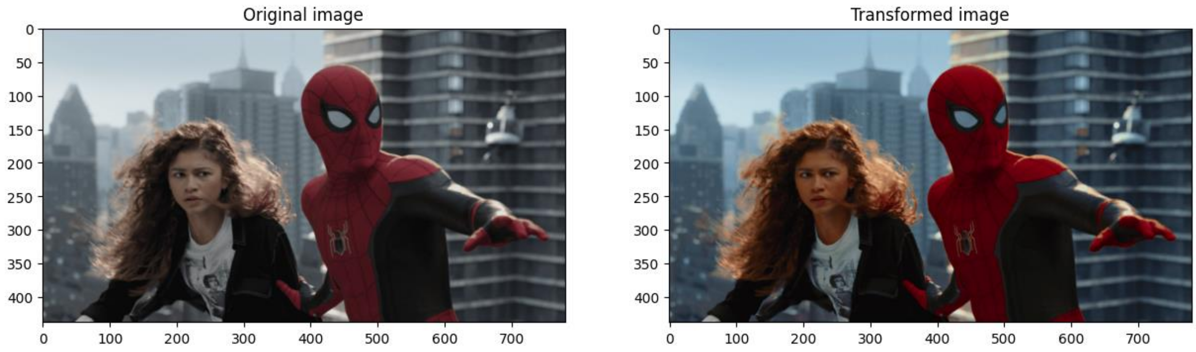
Histograms of the original and corrected images:



Question 04

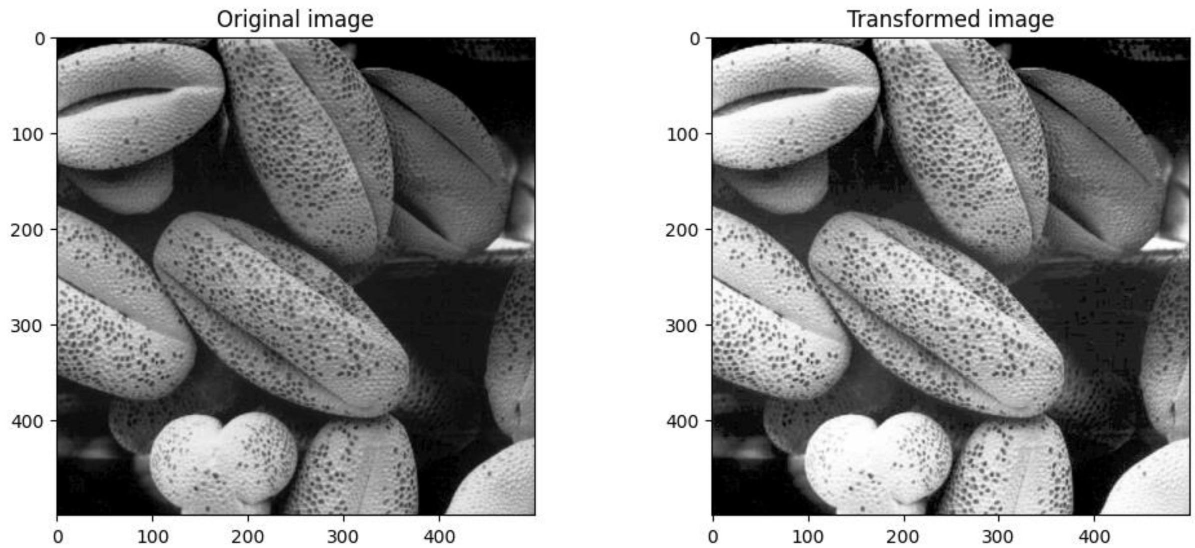
```python
Original_image = cv.cvtColor(Original_image, cv.COLOR_BGR2RGB)
converted_image = cv.cvtColor(Original_image, cv.COLOR_RGB2HSV)
h, s, v = cv.split(converted_image)

def f(x , a ):
    zigma=70
    y = int(x + (128*a)*(np.exp(-(((x-128)**2)/(2*(zigma**2))))))
    return (min(y,255))
a=0.8

transformation = [f(x, a) for x in np.arange(0, 256)]

transformed_s = np.vectorize(f)(s, a).astype(np.uint8)
print(transformed_s)

vibrantHSVimage = cv.merge([h, transformed_s, v])
vibrantBGRimage = cv.cvtColor(vibrantHSVimage, cv.COLOR_HSV2RGB)
```



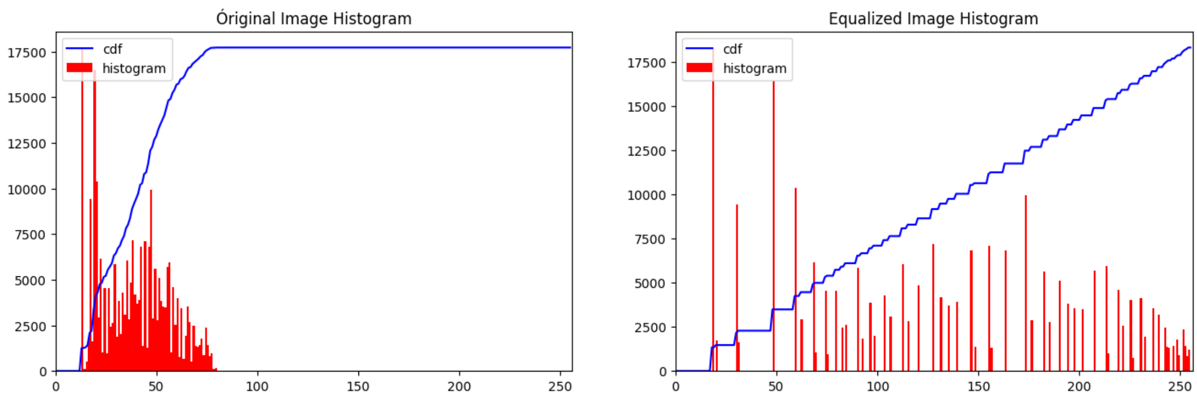


Question 05

```python
def Histogram_Equalizer(image):
    hist , bins = np.histogram(image.ravel(),256,[0,256])
    cdf=hist.cumsum()
    dim = image.shape
    eq = (255*cdf/dim[0]/dim[1]).astype(np.uint8)
    return cv.LUT(image, eq)
```

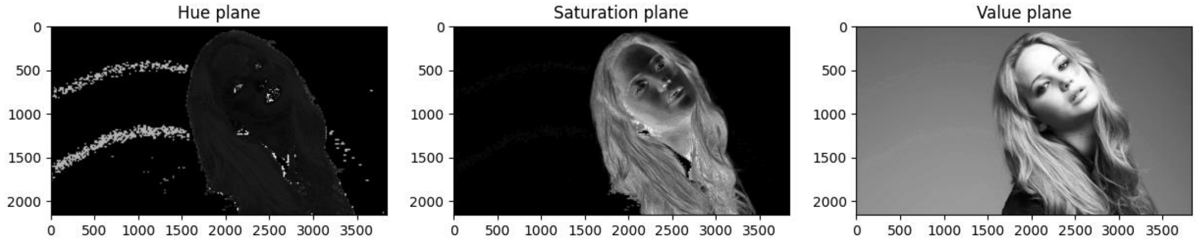


The histograms before and after equalization:

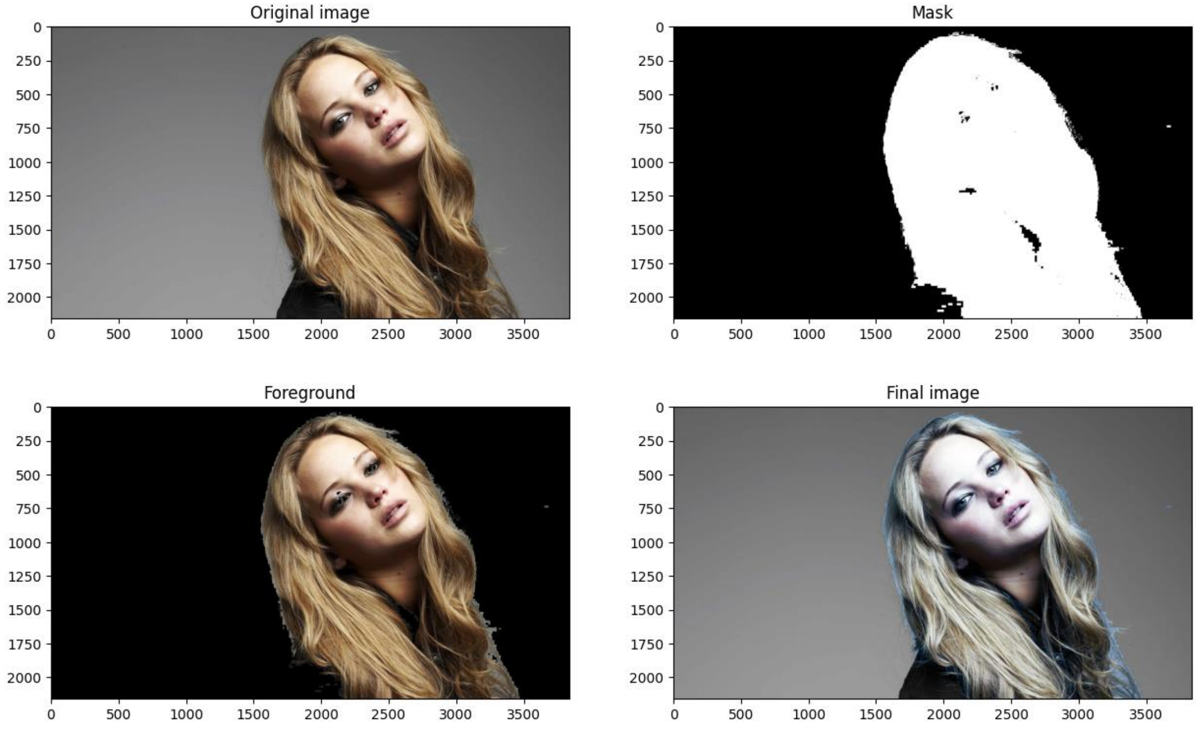


Question 06

```
Converted_image = cv.cvtColor(Original_image , cv.COLOR_BGR2HSV)
h,s,v = cv.split(Converted_image)
```



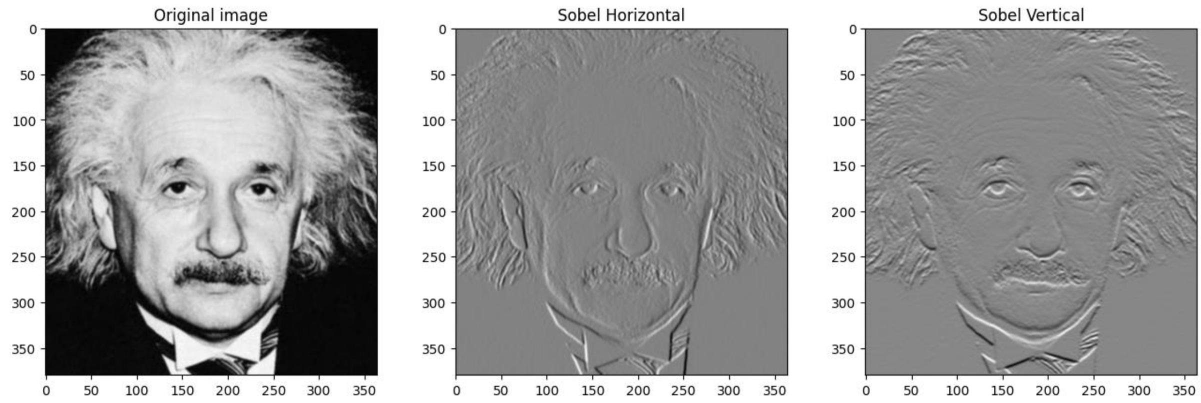

```
fg_mask = cv.threshold(s, 11, 255, cv.THRESH_BINARY)[1]

fg = cv.bitwise_and(Original_image, Original_image, mask=fg_mask)

equalized_fg = cv.merge([red, green, blue])
bg  = cv.bitwise_and(Original_image, Original_image, mask=cv.bitwise_not(fg_mask))
final_image = cv.add(cv.cvtColor(bg, cv.COLOR_BGR2RGB), equalized_fg)
```



Question 07

Using the existing $filter2D$ to Sobel filter the image:

```
sobel_filterH = np.array([(-1, 0 ,1),(-2 , 0 , 2),(-1 ,0 ,1)],dtype='float')
sobel_filterV = np.array([(-1, -2 ,-1),(0 , 0 , 0),(1 ,2 ,1)],dtype='float')

Hfiltered_image = cv.filter2D(original_image, cv.CV_32F, sobel_filterH)
Vfiltered_image = cv.filter2D(original_image, cv.CV_32F, sobel_filterV)
```



Own code to Sobel filter the image:

```
dim=original_image.shape
Hfiltered_image=np.zeros(dim,np.float32)
Vfiltered_image=np.zeros(dim,np.float32)
for row in range(1,dim[0]-1):
    for coloumn in range(1,dim[1]-1):
        Hfiltered_image[row,coloumn] = np.sum(original_image[row-1:row+2,coloumn-1:coloumn+2]*sobel_filterH)

for row in range(1,dim[0]-1):
    for coloumn in range(1,dim[1]-1):
        Vfiltered_image[row,coloumn] = np.sum(original_image[row-1:row+2,coloumn-1:coloumn+2]*sobel_filterV)
```

Using the property

```
hr1=np.array([[1,0,-1]],dtype='float32')
hc1=np.array([[1],[2],[2]],dtype=float32)
image1=cv.filter2D(original_image,cv.CV_32F,hc1)*cv.filter2D(original_image,cv.CV_32F,hr1)

vr1=np.array([[1,2,1]],dtype='float32')
vc1=np.array([[1],[0],[-1]],dtype=float32)
image2=cv.filter2D(original_image,cv.CV_32F,vr1)*cv.filter2D(original_image,cv.CV_32F,vc1)
```

Question 08

Nearest Neighbor

```python
def nearest_neighbor(image, k):
    h1, w1, channels = image.shape
    h2 = int(h1 * k)
    w2 = int(w1 * k)
    image2 = np.zeros((h2, w2, channels), dtype=np.uint8)

    for i in range(h2):
        for j in range(w2):
            si = int(i / k)
            sj = int(j / k)
            image2[i, j] = image[si, sj]

    return image2
```

Bilinear Interpolation

```python
def bilinear_interpolation(image1, k):
    h, w, channels = image1.shape
    h1 = int(h * k)
    w1 = int(w * k)
    image2 = np.zeros((h1, w1, channels), dtype=np.uint8)

    for i in range(h1):
        for j in range(w1):
            src_i = i / k
            src_j = j / k
            x1 = int(src_i)
            x2 = min(x1 + 1, h - 1)
            y1 = int(src_j)
            y2 = min(y1 + 1, w - 1)
            dx = src_i - x1
            dy = src_j - y1
            interpolated = (1 - dx) * (1 - dy) * image1[x1, y1] + dx * (1 - dy) * image1[x2, y1] + (1 - dx) * dy * image1[x1, y2] + dx * dy * image1[x2, y2]
            image2[i, j] = interpolated

    return image2
```

Image	SSD of nearest_neighbor	SSD of bilinear_interpolation
Im01	31.28	39.26
Im02	11.90	16.21
Im04	78.73	81.66
Im05	50.57	53.71
Im06	30.55	35.52
Im07	27.95	30.22
Im09	21.15	26.67

For Images im03, im08, im10, im11:

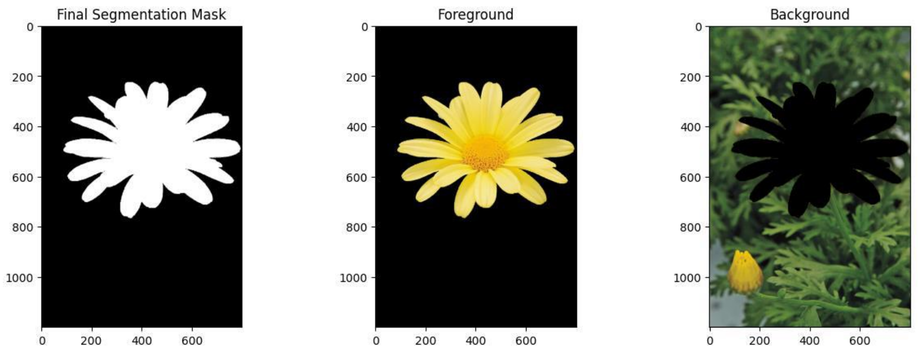
```
ValueError: operands could not be broadcast together with shapes (1460,2400,3) (1459,2400,3)
```

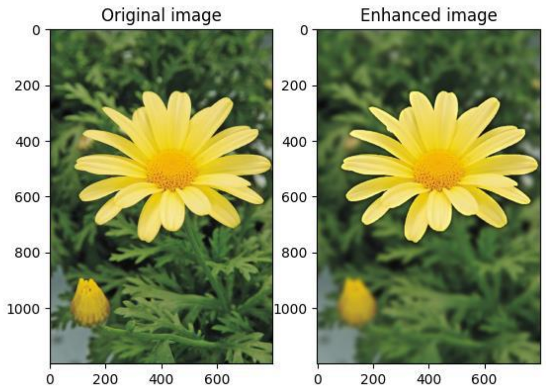
Question 09

```
roi = (70, 178, 730, 619)

mask = np.zeros(image.shape[:2], dtype = "uint8")
fgm = np.zeros((1, 65), dtype = "float")
bgm = np.zeros((1, 65), dtype = "float")

(mask , bgm , fgm) =  cv.grabCut(image, mask, roi, bgm, fgm, 5 , cv.GC_INIT_WITH_RECT)
```

```
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

cv.grabCut(image, mask2, None, bgm, fgm, 5 , cv.GC_INIT_WITH_MASK)

fg = cv.bitwise_and(image, image, mask = mask2)
bg = image - fg
```

Image Results






```
blurred_bg = cv.GaussianBlur(bg, (0, 0), 7)

plt.imshow(cv.cvtColor(blurred_bg, cv.COLOR_BGR2RGB))
enhanced_image = cv.add(blurred_bg, fg)
```

The Gaussian Blur filter's application extends its influence to neighboring pixels. Consequently, when we employ this filter on the background, the black pixels generated by the removal of the foreground will impact nearby pixels situated immediately beyond the foreground's edges. This influence causes these neighboring pixels to adopt a black hue. Consequently, even after we reintroduce the foreground, we may still observe this phenomenon as darkened edges.