University of Moratuwa

Department of Electronic and Telecommunication Engineering

EN2550 - Fundamentals of Image Processing and Machine Vision

Transformations and Neighborhood Filtering

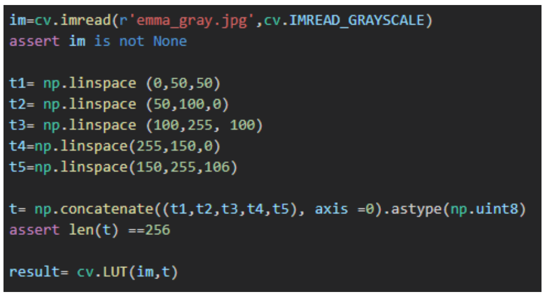
W. A. D. K. De Silva	190128H

github.com/devindi99

March 5 2022

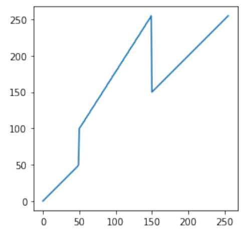
Imported libraries
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

1. QUESTION 1



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

t1= np.linspace (0,50,50)
t2= np.linspace (50,100,0)
t3= np.linspace (100,255, 100)
t4=np.linspace(255,150,0)
t5=np.linspace(150,255,106)

t= np.concatenate((t1,t2,t3,t4,t5), axis =0).astype(np.uint8)
assert len(t) ==256

result= cv.LUT(im,t)
```

(a) Code snippet



(b) Look up table

When we apply this intensity transformation it can be seen that only the pixel values that are of range 50-150 get enhanced and the rest remain as same as before

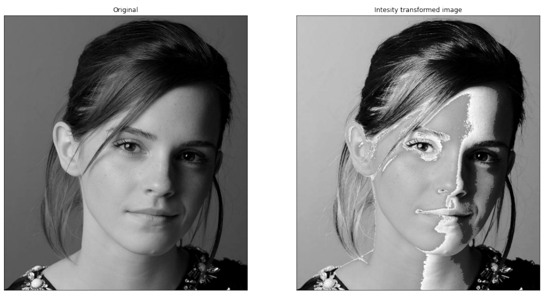
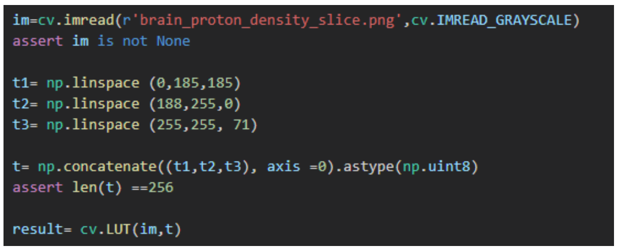


Figure 1: Comparison of original image and resultant image

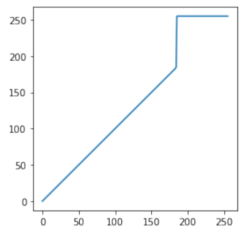
2. QUESTION 2

2.1 Accentuating the white matter



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

t1= np.linspace (0,185,185)
t2= np.linspace (188,255,0)
t3= np.linspace (255,255, 71)

t= np.concatenate((t1,t2,t3), axis =0).astype(np.uint8)
assert len(t) ==256

result= cv.LUT(im,t)
```

(a) Code snippet



(b) Look up table

In order to accentuate the white matter of the image the pixel values that are closer to 256 must be intensified.

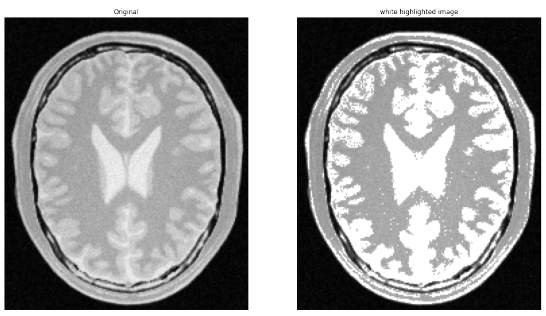
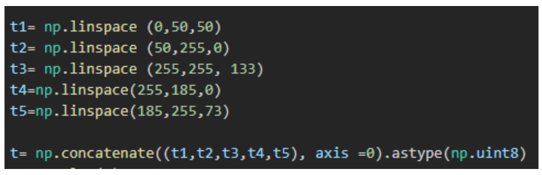


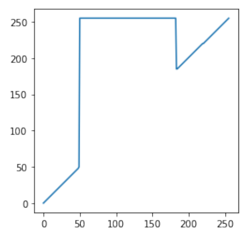
Figure 2: White matter accentuated image

2.2 Accentuating the gray matter



```
t1= np.linspace (0,50,50)
t2= np.linspace (50,255,0)
t3= np.linspace (255,255, 133)
t4=np.linspace(255,185,0)
t5=np.linspace(185,255,73)

t= np.concatenate((t1,t2,t3,t4,t5), axis =0).astype(np.uint8)
```

(a) Code snippet



(b) Look up table

In order to accentuate the gray matter the darker pixels (closer to 0) must be intensified. In this case values in the range 0-50 is not intensified, as it will enhance the black area too.

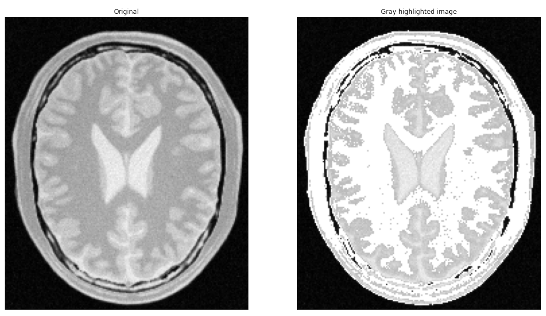
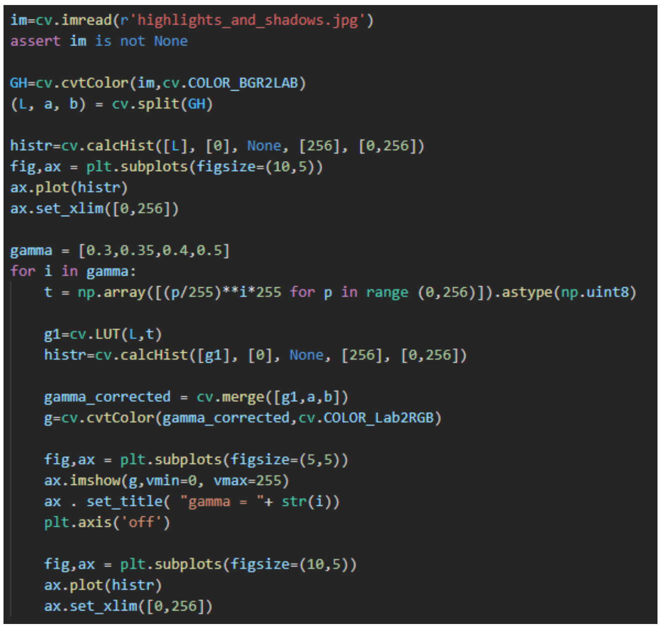


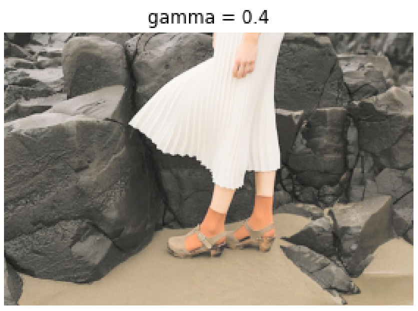
Figure 3: Gray matter accentuated image

3. QUESTION 3



```
im=cv.imread(r'highlights_and_shadows.jpg')
assert im is not None

GH=cv.cvtColor(im,cv.COLOR_BGR2LAB)
(L, a, b) = cv.split(GH)

histr=cv.calcHist([L], [0], None, [256], [0,256])
fig,ax = plt.subplots(figsize=(10,5))
ax.plot(histr)
ax.set_xlim([0,256])

gamma = [0.3,0.35,0.4,0.5]
for i in gamma:
    t = np.array([(p/255)**i*255 for p in range (0,256)]).astype(np.uint8)

    g1=cv.LUT(L,t)
    histr=cv.calcHist([g1], [0], None, [256], [0,256])

    gamma_corrected = cv.merge([g1,a,b])
    g=cv.cvtColor(gamma_corrected,cv.COLOR_Lab2RGB)

    fig,ax = plt.subplots(figsize=(5,5))
    ax.imshow(g,vmin=0, vmax=255)
    ax . set_title( "gamma = "+ str(i))
    plt.axis('off')

    fig,ax = plt.subplots(figsize=(10,5))
    ax.plot(histr)
    ax.set_xlim([0,256])
```

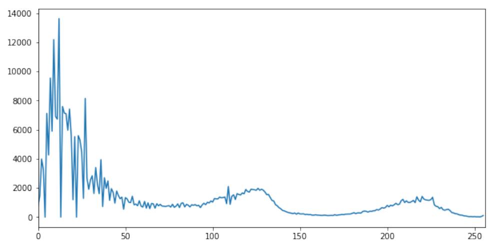
(a) Code snippet



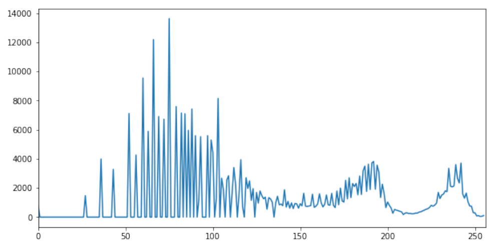
(b) Gamma corrected image

Gamma correction controls the overall brightness of the image. Using gamma correction we can accurately display an image on a computer screen. A computer monitor has an intensity to voltage response function which approximately raises the voltage level by a power of 2.5. Therefore after taking all the matters into consideration 1/2.5 is used as the gamma value.

3.1 Comparing the histograms

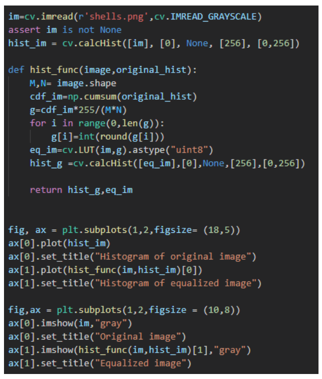


(c) Histogram of L plane of original image



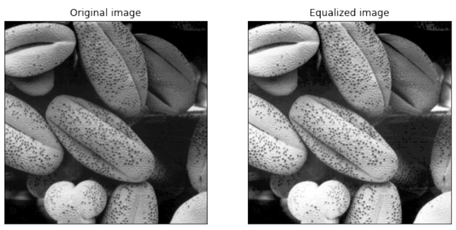
(d) Histogram of L plane of gamma corrected image

4. QUESTION 4



```
im=cv.imread(r'shells.png',cv.IMREAD_GRAYSCALE)
assert im is not None
hist_im = cv.calcHist([im], [0], None, [256], [0,256])

def hist_func(image,original_hist):
    M,N= image.shape
    cdf_im=np.cumsum(original_hist)
    g=cdf_im*255/(M*N)
    for i in range(0,len(g)):
        g[i]=int(round(g[i]))
    eq_im=cv.LUT(im,g).astype("uint8")
    hist_g =cv.calcHist([eq_im],[0],None,[256],[0,256])

    return hist_g,eq_im


fig, ax = plt.subplots(1,2,figsize= (18,5))
ax[0].plot(hist_im)
ax[0].set_title("Histogram of original image")
ax[1].plot(hist_func(im,hist_im)[0])
ax[1].set_title("Histogram of equalized image")

fig,ax = plt.subplots(1,2,figsize = (10,8))
ax[0].imshow(im,"gray")
ax[0].set_title("Original image")
ax[1].imshow(hist_func(im,hist_im)[1],"gray")
ax[1].set_title("Equalized image")
```

(e) Code snippet



(f) Comparing the images

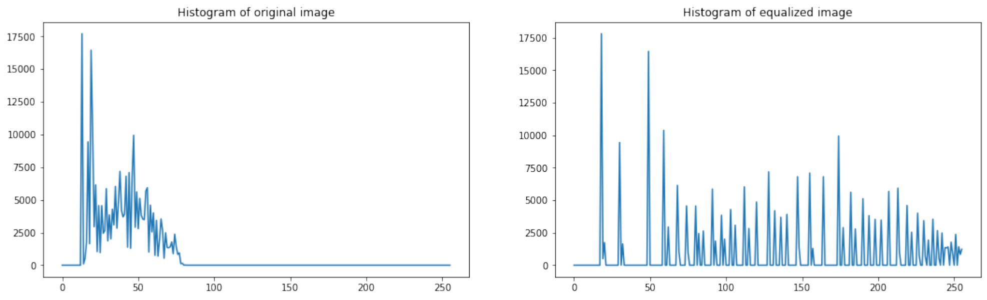
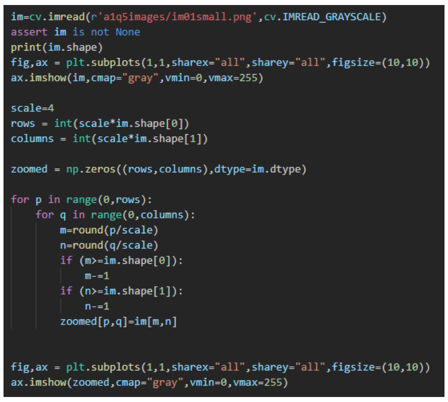


Figure 4: Comparing the histograms

Histogram equalization stretches out the most frequent pixel intensity values of an image which will make the areas of lower contrast to gain a higher contrast.In order to obtain histogram equalization the PMF values are calculated using *calc.hist()* function and the CDF is calculated using *numpy.cumsum()* function. Since this is an 8 bit image the CDF values are multiplied by 255 and divided by the total number of pixels.

5. QUESTION 5

5.1 Nearest neighbour interpolation



```
im=cv.imread(r'a1q5images/im01small.png',cv.IMREAD_GRAYSCALE)
assert im is not None
print(im.shape)
fig,ax = plt.subplots(1,1,sharex="all",sharey="all",figsize=(10,10))
ax.imshow(im,cmap="gray",vmin=0,vmax=255)

scale=4
rows = int(scale*im.shape[0])
columns = int(scale*im.shape[1])

zoomed = np.zeros((rows,columns),dtype=im.dtype)

for p in range(0,rows):
    for q in range(0,columns):
        m=round(p/scale)
        n=round(q/scale)
        if (m>=im.shape[0]):
            m-=1
        if (n>=im.shape[1]):
            n-=1
        zoomed[p,q]=im[m,n]


fig,ax = plt.subplots(1,1,sharex="all",sharey="all",figsize=(10,10))
ax.imshow(zoomed,cmap="gray",vmin=0,vmax=255)
```

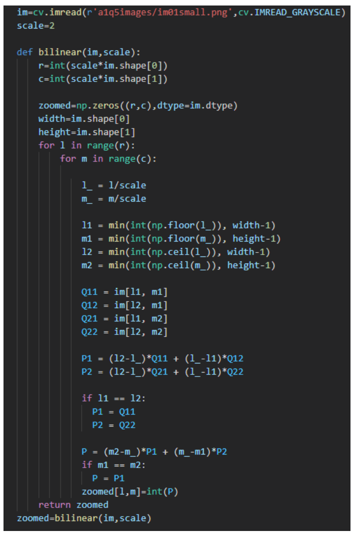
Figure 5: Code snippet

(a) Original image

(b) Zoomed image by a scale of 4

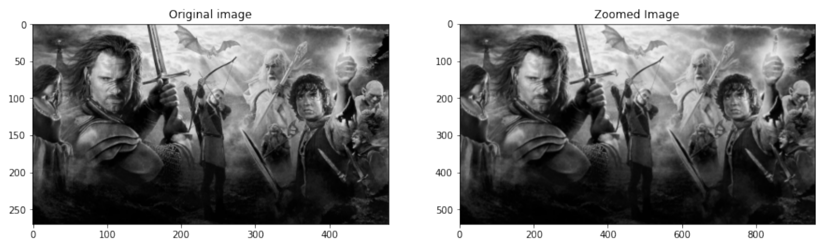
Nearest neighbour interpolation is achieved by interpolating the the calculated pixel value to th nearest neighbour. In order to achieve this *round()* function has been used.

5.2 Bilenear interpolation



```
im=cv.imread(r'a1q5images/im01small.png',cv.IMREAD_GRAYSCALE)
scale=2

def bilinear(im,scale):
    r=int(scale*im.shape[0])
    c=int(scale*im.shape[1])

    zoomed=np.zeros((r,c),dtype=im.dtype)
    width=im.shape[0]
    height=im.shape[1]
    for l in range(r):
        for m in range(c):

            l_ = l/scale
            m_ = m/scale

            l1 = min(int(np.floor(l_)), width-1)
            m1 = min(int(np.floor(m_)), height-1)
            l2 = min(int(np.ceil(l_)), width-1)
            m2 = min(int(np.ceil(m_)), height-1)

            Q11 = im[l1, m1]
            Q12 = im[l2, m1]
            Q21 = im[l1, m2]
            Q22 = im[l2, m2]

            P1 = (l2-l_)*Q11 + (l_-l1)*Q12
            P2 = (l2-l_)*Q21 + (l_-l1)*Q22

            if l1 == l2:
              P1 = Q11
              P2 = Q22

            P = (m2-m_)*P1 + (m_-m1)*P2
            if m1 == m2:
              P = P1
            zoomed[l,m]=int(P)
    return zoomed
zoomed=bilinear(im,scale)
```

(c) Code snippet

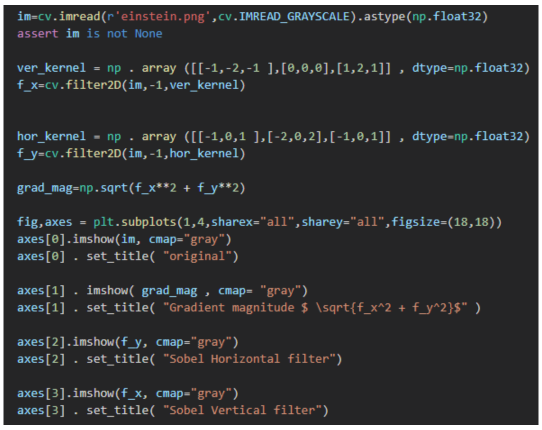


(d) Comparing the images

Bi-linear interpolation applies linear interpolation in two directions. Using the 4 nearest neighbours the weighted average is taken and the output is produced.

6. QUESTION 6

6.1 Using sobel filters



```python
im=cv.imread(r'einstein.png',cv.IMREAD_GRAYSCALE).astype(np.float32)
assert im is not None

ver_kernel = np . array ([[-1,-2,-1 ],[0,0,0],[1,2,1]] , dtype=np.float32)
f_x=cv.filter2D(im,-1,ver_kernel)


hor_kernel = np . array ([[-1,0,1 ],[-2,0,2],[-1,0,1]] , dtype=np.float32)
f_y=cv.filter2D(im,-1,hor_kernel)

grad_mag=np.sqrt(f_x**2 + f_y**2)

fig,axes = plt.subplots(1,4,sharex="all",sharey="all",figsize=(18,18))
axes[0].imshow(im, cmap="gray")
axes[0] . set_title( "original")

axes[1] . imshow( grad_mag , cmap= "gray")
axes[1] . set_title( "Gradient magnitude $ \sqrt{f_x^2 + f_y^2}$" )

axes[2].imshow(f_y, cmap="gray")
axes[2] . set_title( "Sobel Horizontal filter")

axes[3].imshow(f_x, cmap="gray")
axes[3] . set_title( "Sobel Vertical filter")
```

Figure 6: Code snippet

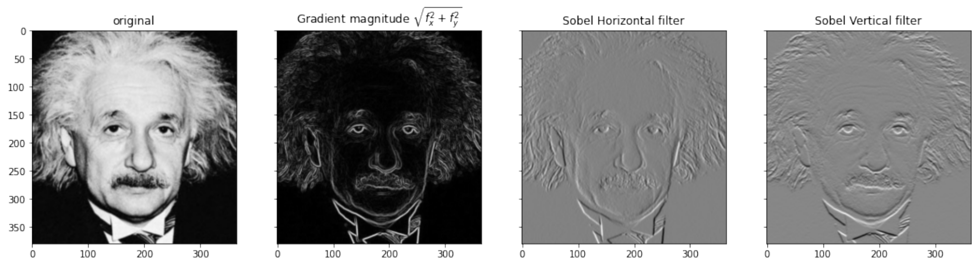
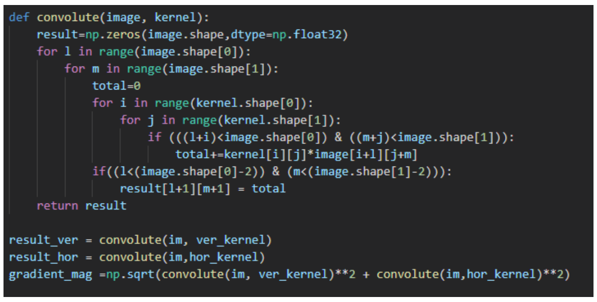


Figure 7: Images

Sobel horizontal filter highlights the vertical edges of an image while sobel vertical filter highlights the horizontal edges of an image. The filtering operation calculates how much similar is the kernel to the image.

6.2 Using my own function



```python
def convolute(image, kernel):
    result=np.zeros(image.shape,dtype=np.float32)
    for l in range(image.shape[0]):
        for m in range(image.shape[1]):
            total=0
            for i in range(kernel.shape[0]):
                for j in range(kernel.shape[1]):
                    if (((l+i)<image.shape[0]) & ((m+j)<image.shape[1])):
                        total+=kernel[i][j]*image[i+l][j+m]
            if((l<(image.shape[0]-2)) & (m<(image.shape[1]-2))):
                result[l+1][m+1] = total
    return result

result_ver = convolute(im, ver_kernel)
result_hor = convolute(im,hor_kernel)
gradient_mag =np.sqrt(convolute(im, ver_kernel)**2 + convolute(im,hor_kernel)**2)
```

Figure 8: Code snippet

When getting the resultant image zero padding have been considered . When using this function the computational time has increased in a large amount(0.5s to 6.4s)

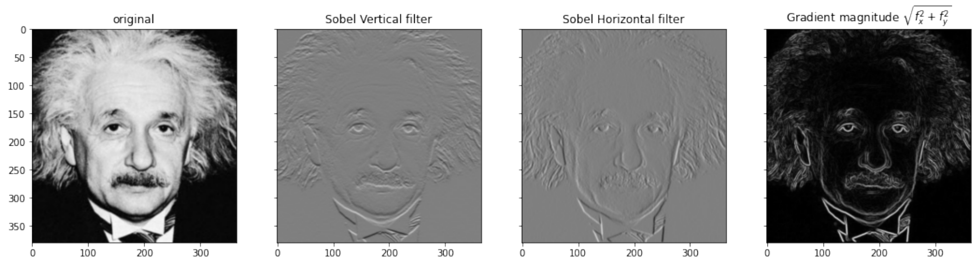
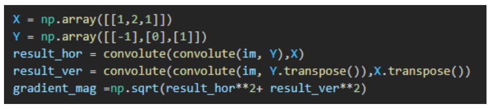


Figure 9: Images

6.3 Using convolution properties



```
X = np.array([[1,2,1]])
Y = np.array([[-1],[0],[1]])
result_hor = convolute(convolute(im, Y),X)
result_ver = convolute(convolute(im, Y.transpose()),X.transpose())
gradient_mag =np.sqrt(result_hor**2+ result_ver**2)
```

Figure 10: Code snippet

By passing arguments as shown above sobel filtering can be implemented using the properties of convolution. As can be seen from the execution time (6.2s) this method is faster than convoluting the whole kernel with the image (6.4s) as the computational complexity of convolution increases when the kernel size increases.

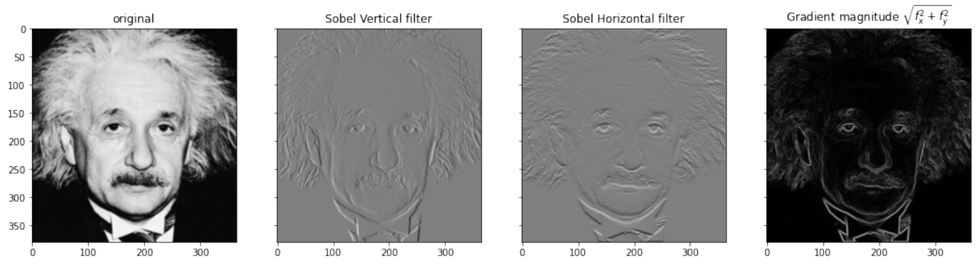
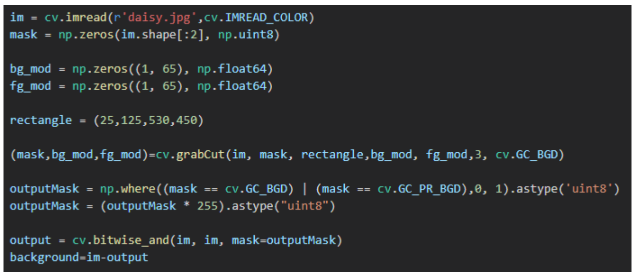


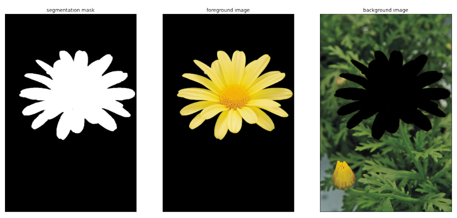
Figure 11: Images

7. QUESTION 7

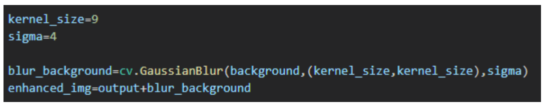


```
im = cv.imread(r'daisy.jpg',cv.IMREAD_COLOR)
mask = np.zeros(im.shape[:2], np.uint8)

bg_mod = np.zeros((1, 65), np.float64)
fg_mod = np.zeros((1, 65), np.float64)

rectangle = (25,125,530,450)

(mask,bg_mod,fg_mod)=cv.grabCut(im, mask, rectangle,bg_mod, fg_mod,3, cv.GC_BGD)

outputMask = np.where((mask == cv.GC_BGD) | (mask == cv.GC_PR_BGD),0, 1).astype('uint8')
outputMask = (outputMask * 255).astype("uint8")

output = cv.bitwise_and(im, im, mask=outputMask)
background=im-output
```

(a) Code snippet

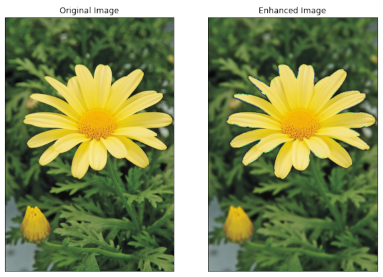


(b) Images



```
kernel_size=9
sigma=4

blur_background=cv.GaussianBlur(background,(kernel_size,kernel_size),sigma)
enhanced_img=output+blur_background
```

(c) Code snippet



(d) Comparing the images

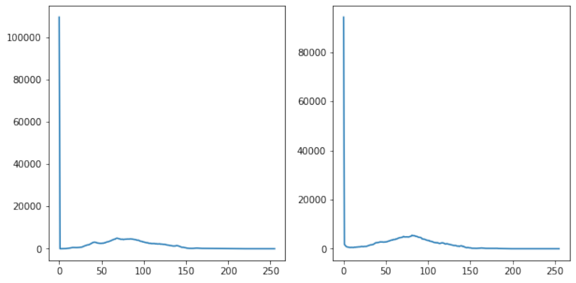


Figure 12: Comparing the histograms

When comparing the two histograms it can be seen that intensity levels of pixel values have changed slightly when Gaussian blur was applied to the background of the image. It can be explained as follows; Gaussian blur is a spatial filtering technique. When Gaussian blur is applied to the background image, which has a black area where the flower had been,the edges of the blackened area get smoothed too. In other words the pixels near the edges also obtains a dark effect.

8. REFERNECES

- OpenCV: Interactive Foreground Extraction using GrabCut Algorithm
- Medical Image Contrast Enhancement based on Gamma Correction
- What is bilinear interpolation?