EN2550 - Image Processing and Computer Vision

Name :- ADIKARI A.M.A.D.

Index No :- 190021A

❖ Question 01

○ Implement the intensity transformation

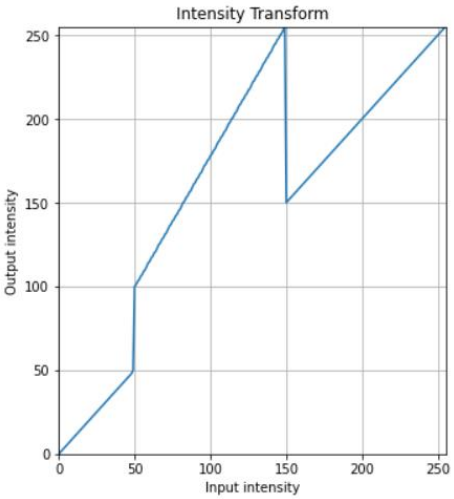
- An intensity transformation is used to change the intensities of pixels in the given grayscale image. The transformation array is as follows.

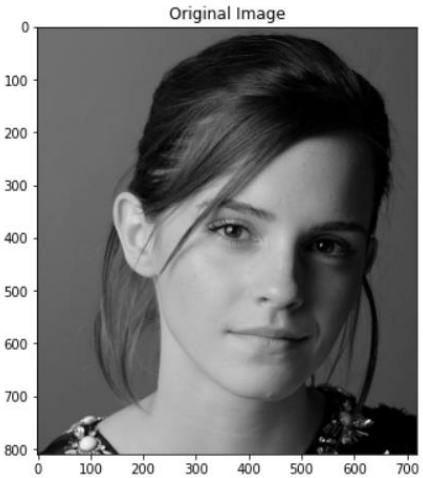
```
t1 = np.linspace(0, 50, 50)
t2 = np.linspace(100, 255, 100)
t3 = np.linspace(150, 255, 106)
t = np.concatenate((t1, t2, t3), axis=0).astype(np.uint8)
filtered_image = cv.LUT(image, t)
```

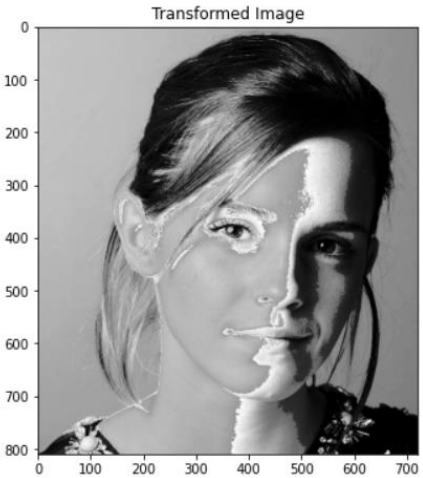
- color values in ranges 0 – 50 & 150 – 255 in original picture are not changed while transforming intensities. Color values in range 50 – 150 are transformed according to the formula $\frac{155}{100}$ * x + 22.5.

- LUT() function is used to map input values to output values for this intensity transformation.

- After this transformation we can observe the intensity difference between the input and output images as below. A side of the Emma's face and neck has been fully grayed while the intensity of other side has comparatively less change.







❖ Question 02

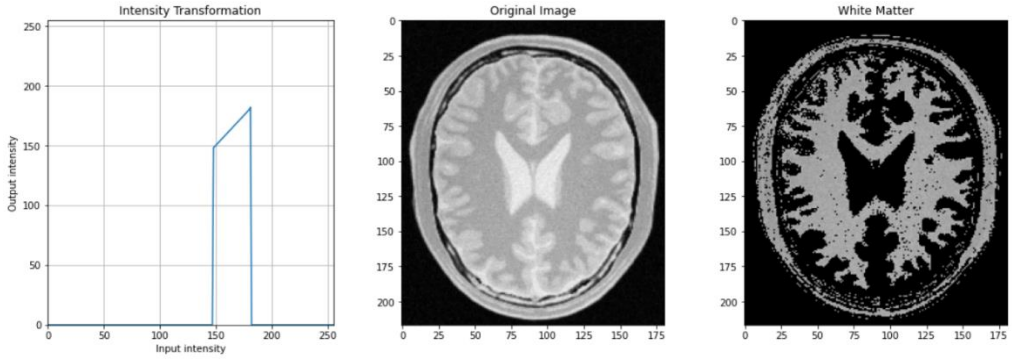
○ Implement the intensity transformations

- Intensity transformations are used to accentuate white matter and gray matter in the given brain proton density slice image.

- intensity transformation array for the white matter is as follows. The range 148 – 182 of intensity is not changed and other intensities are reduced into black to accentuate white matter area in the image.

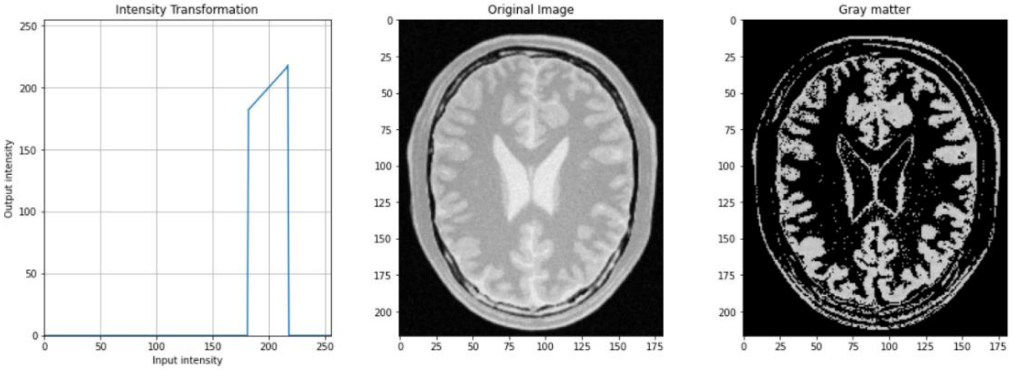
```
white_begin = 148
white_end = 182
t1 = np.linspace(0, 0, white_b (variable) white_end: Literal[182]
t2 = np.linspace(white_begin, white_end, white_end - white_begin)
t3 = np.linspace(0, 0, 256 - white_end)
t = np.concatenate((t1, t2, t3), axis=0).astype(np.uint8)
filtered_image = cv.LUT(image, t)
```

- Using this transformation with LUT() function, we can accentuate the white matter area.



- intensity transformation array for the gray matter is as follows. The range 182 - 218 of intensity is not changed and other intensities are reduced into black to accentuate gray matter area in the image. Using this transformation with LUT() function, we can accentuate the gray matter area.

```
gray_begin = 182
gray_end = 218
t1 = np.linspace(0, 0, gray_begin)
t2 = np.linspace(gray_begin, gray_end, gray_end - gray_begin)
t3 = np.linspace(0, 0, 256 - gray_end)
t = np.concatenate((t1, t2, t3), axis=0).astype(np.uint8)
filtered_image = cv.LUT(image, t)
```



- The input intensity value ranges for white matter and gray matter are found from the histogram of the given image.

❖ Question 03

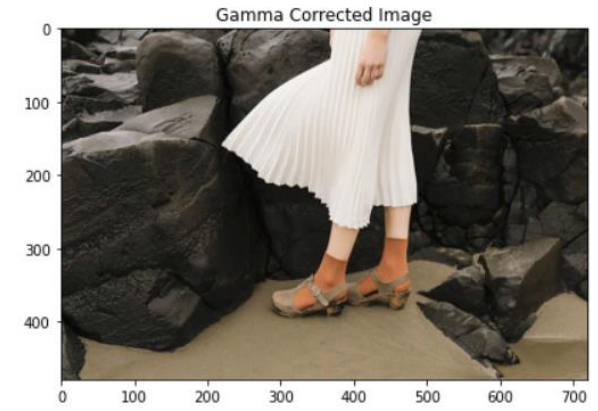
- Gamma correction

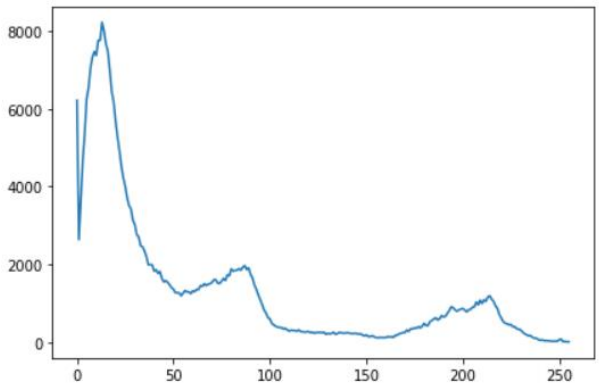
  - In this question, gamma correction is used to the L plane in L*a*b color space.

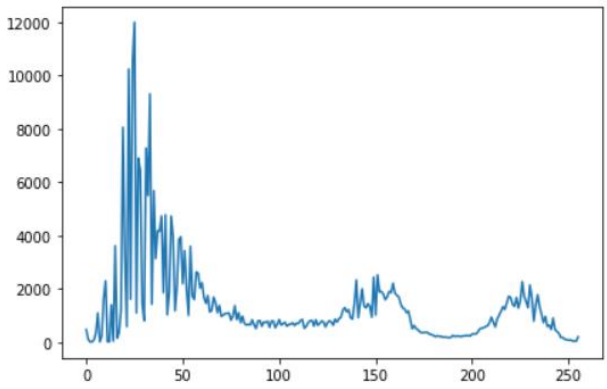
  - In the code, L plane is split first from the color space and then gamma correction is applied to it. Here, gamma value is selected as 0.75 to get an image with a clear change. After that color space is merged again.

```
Lab_image = cv.cvtColor(image, cv.COLOR_BGR2Lab)
img = np.array(Lab_image)
hist_img = cv.calcHist([image], [0], None, [256], [0, 256])
L,a,b = cv.split(img)
ga = 0.75
t = np.array([(p/255)**ga*255 for p in range(0, 256)]).astype(np.uint8)
L = cv.LUT(L, t)
modified_image = cv.merge([L, a, b])
```

- After this gamma correction, we can see clearly that the shadows in the original image have been reduced in the output image and it has been clearer and brighter than the original one. The histograms after and before gamma correction are shown below.







- According to histograms, we can see output image has more brighter pixels than the input image. And the output histogram has more vibrant effect than the input one.

❖ Question 04

○ Equalization

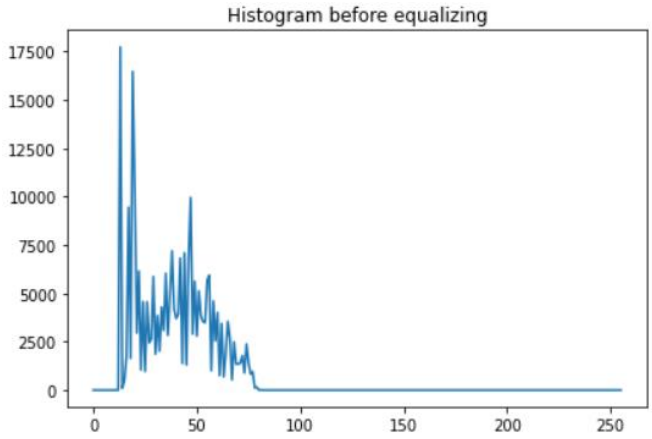
- Histogram equalization is used to get more vibrant images. Due to the equalization, the image will be clear, and contrast will be increased.

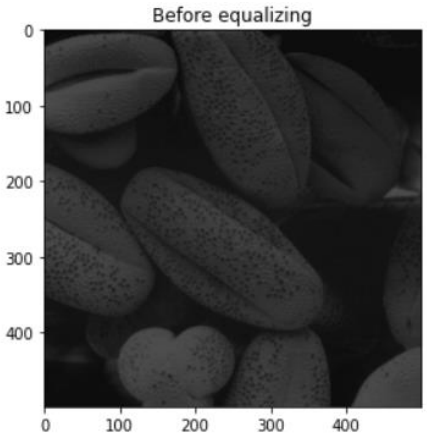
- The Equalizer function gets the image as a cumulative sum and normalized it. After that it fills the full histogram.

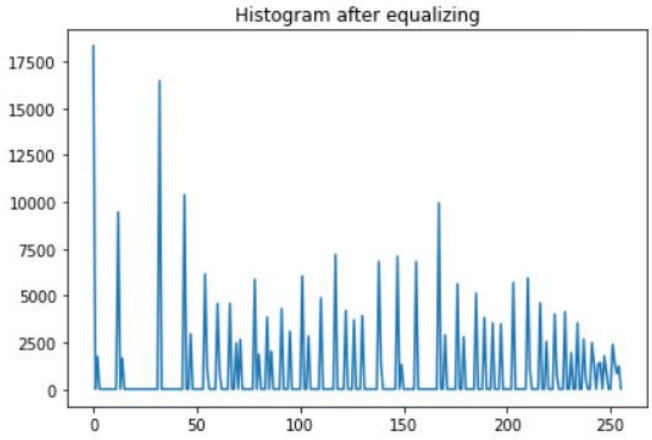
- The code for the equalizer is as follows

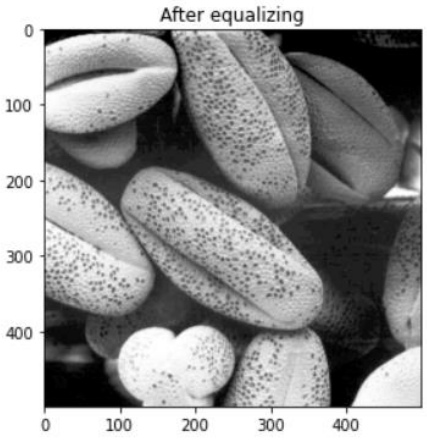
```
def Equalizer(img):
    histo, _ = np.histogram(img.flatten(), 256, [0, 255])
    cdf = histo.cumsum()
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min()) / (cdf_m.max() - cdf_m.min()) * 255
    cdf = np.ma.filled(cdf_m, 0)
    imgEq = cdf[img.astype('uint8')]
    histEq, _ =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

- By using this equalizer, we can observe clear output image with many intensity levels. The histogram after equalizing shows how the intensity levels are made for the output image.









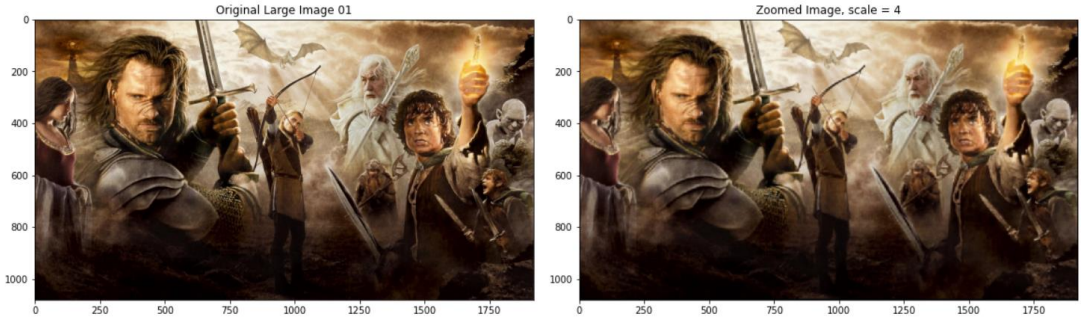
- Here, a gray scale image has been used for equalization but If we use a RGB image, we have to equalize the image in all three planes.

❖ Question 05

- Zooming
  - a) Nearest-neighbor method

- In this method, a color value in a pixel in zoomed image is selected from one pixel in the original image. First, we make an array with zeros of suitable height and width according to the scale and then substitute values to those. We select the color value of (i/s, j/s, k) pixel (rounded) in the original image as the color value of (i, j, k) pixel in the zoomed image.

```python
def nearest_zoom(img, s):
    rows, cols = img.shape[0] * s, img.shape[1] * s
    zoomed_img = np.zeros((rows, cols, 3), dtype=img.dtype)
    for i in range(0, rows-4):
        for j in range(0, cols-4):
            for k in range(0, 3):
                zoomed_img[i, j, k] = img[int(round(i/s)), int(round(j/s)), k]
    return zoomed_img
```



- The above small size image has dimensions (270, 480, 3) and it has been zoomed with scale=4 by using the above nearest_zoom function. Therefore, the zoomed image has dimensions (1080, 1920, 3).

  - b) Bilinear interpolation method

- In this method, color values of the zoomed image are calculated by interpolating color values in the original image.

- First, we make an array with zeros of suitable height and width according to the scale and then substitute values to those. Here, we calculate the color value of (i, j, k) pixel in the zoomed image by linear interpolating color values of four neighbor pixels around (i/s, j/s, k) pixel in the original image.

```python
def bilinear_zoom(img, s):
    rows = img.shape[0]*s
    cols = img.shape[1]*s

    zoomed_img = np.zeros((rows, cols, 3), dtype=img.dtype)
    for i in range(0, rows-4):
        for j in range(0, cols-4):
            x_diff, y_diff = i/s - np.floor(i/s), j/s - np.floor(j/s)
            for k in range(0, 3):
                tl_val = img[int(np.floor(i/s)), int(np.floor(j/s)), k]
                tr_val = img[int(np.floor(i/s)), int(np.ceil(j/s)), k]
                bl_val = img[int(np.ceil(i/s)), int(np.floor(j/s)), k]
                br_val = img[int(np.ceil(i/s)), int(np.ceil(j/s)), k]
                x_l_val = tl_val*(1-x_diff) + bl_val*x_diff
                x_r_val = tr_val*(1-x_diff) + br_val*x_diff
                color_val = x_l_val*(1-y_diff) + x_r_val*y_diff
                zoomed_img[i, j, k] = color_val
    return zoomed_img
```

- The above small size image has dimensions (270, 480, 3) and it has been zoomed with scale=4 by using the above bilinear_zoom function. Therefore, the zoomed image has dimensions (1080, 1920, 3).

- This method is more accurate than the nearest-neighbor method due to the interpolation of the color values. We can compare these two methods by using SSD (sum of squared difference).

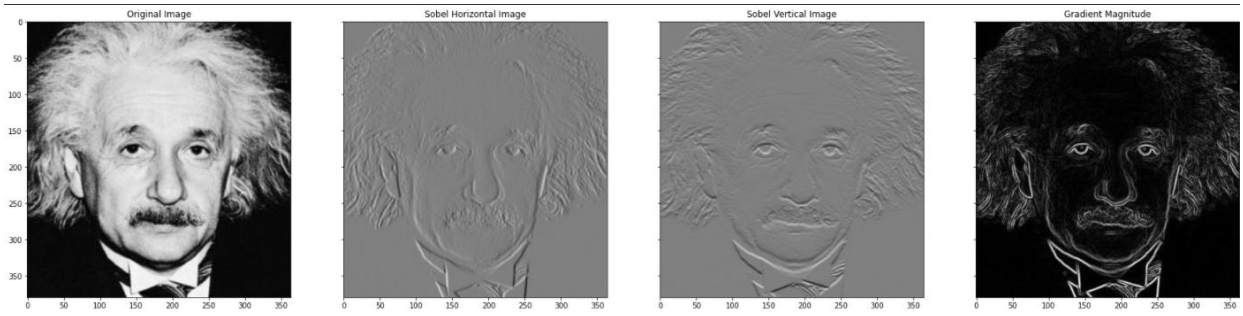
```
def SSD(img1, img2):
  ssd = 0
  rows = int(img1.shape[0])
  cols = int(img1.shape[1])

  for i in range(0,rows):
    for j in range(0,cols):
      for k in range(0,3):
        ssd +=  (img1[i,j,k]-img2[i,j,k])**2
  return(ssd)
```

❖ Question 06

○ Filtering with the Sobel operator

- Generally, we use filter2D function with suitable kernels to get the Sobel vertical and horizontal images. By them we can obtain the gradient magnitude image as well.



- When we manually create this function, first we define the kernel and an array with zeros of height and width of the original image. Then, we must pad a suitable number of layers with zeros to this array according to the size of the kernel. The pad size should be the floor value of the kernel size/2. After that we convolve the kernel with the original image. This gives the Sobel images.

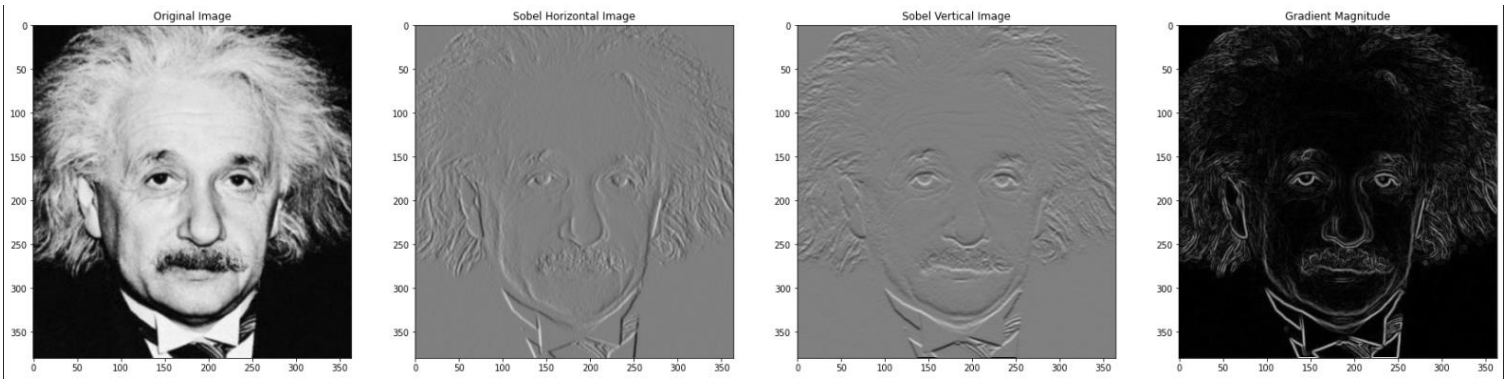
```python
def sobel_filter(image, kernal):
    h, w = np.shape(image)
    k_size = len(kernal)
    pad_size = k_size//2
    upper_pad = np.zeros((pad_size, w + 2*pad_size))
    lower_pad = upper_pad
    mid_pad = np.zeros((h, pad_size))
    mid_img = np.append(mid_pad, image, axis = 1)
    mid_img = np.append(mid_img, mid_pad, axis = 1)
    new_image = np.concatenate((upper_pad, mid_img, lower_pad))

    convoluted_img = np.zeros((h, w))
    for row in range(pad_size, h+pad_size):
        for col in range(pad_size, w+pad_size):
            neighbourhood = np.array([[new_image[row-1][col-1], new_image[row-1][col], new_image[row-1][col+1]], [new_image[row][col-1], new_image[row][col], new_image[row][col+1]],
            [new_image[row+1][col-1], new_image[row+1][col], new_image[row+1][col+1]]])
            convoluted_img[row - pad_size][col - pad_size] = int(sum(sum(np.multiply(neighbourhood, kernal))))
    return convoluted_img
```

- We can obtain the gradient magnitude image by getting the square root of the sum of squares of Sobel vertical and Sobel horizontal values.

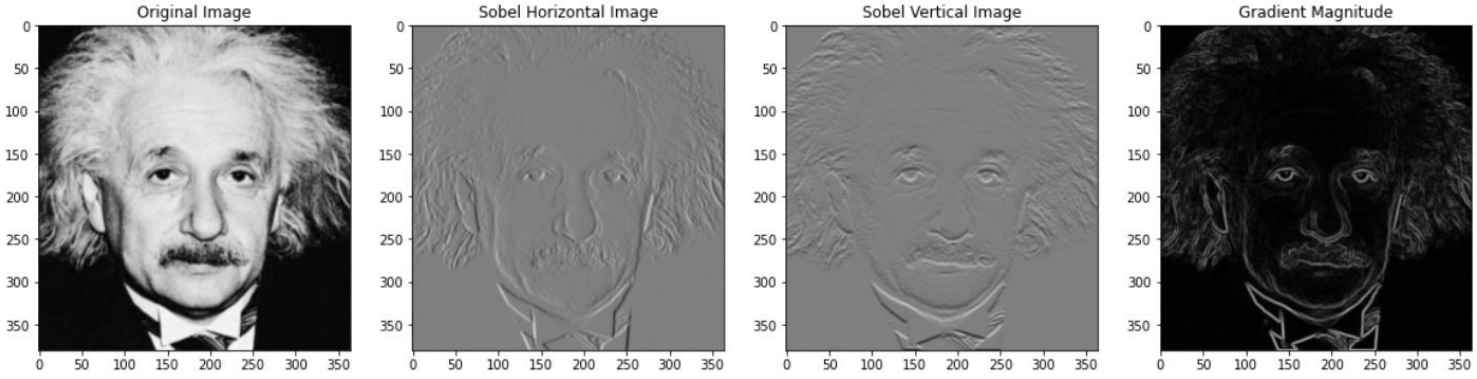
```python
sobel_h_img = sobel_filter(image, sobel_h_kernal)
sobel_v_img = sobel_filter(image, sobel_v_kernal)
gradient_img = (sobel_h_img**2 + sobel_v_img**2) ** 0.5
```

- The below output images are obtained from this sobel_filter function and they are almost similar to the outputs obtained by filter2D function.



- The associative property of convolution is used in part (c). The Sobel vertical filter can be produced by convolving the two arrays mentioned in the question. Using the filter2D function, we can first convolve the picture with the first array, then the result with the second array. Because of the associative property of convolution, this will be the same as applying a Sobel vertical filter.

- Similarly, by convolving two arrays, we can make a Sobel horizontal filter and we can obtain a Sobel horizontal filtered picture using these two arrays and the associative property of convolution. These outputs are also much similar to the outputs which are obtained by using filter2D function.



❖ Question 07

○ Segmentation

- Here we use grabcut algorithm for the segmentation of the given image.

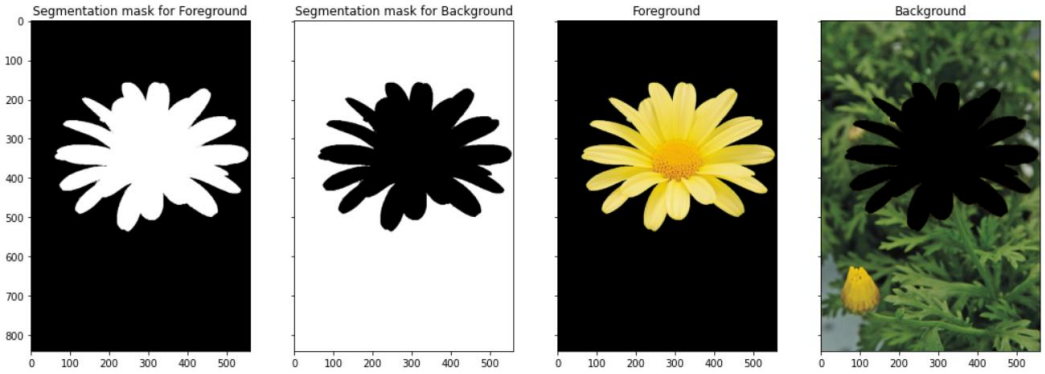
- First, we give a rectangle to as an input and it takes the outside of the rectangle surely as background. Then it estimates the color distribution on inside and outside the rectangle and make separate pdfs for foreground and background. These pdfs help to calculate a probability to identify that a pixel inside the rectangle should be inside or outside the rectangle. After that a min cut algorithm is used to segment the graph and obtain the final mask. The code and output segmented images for this process are as follows.

```
mask1 = np.zeros(image.shape[:2], dtype=np.uint8)

bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)

rect = (50, 150, 520, 400)
cv.grabCut(image, mask1, rect, bgdModel, fgdModel, 5, cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask1 == 0) | (mask1 == 2), 0, 1).astype(np.uint8)
mask3 = np.where((mask1 == 1) | (mask1 == 3), 0, 1).astype(np.uint8)
foreground = image * mask2[:,:,np.newaxis]
background = image * mask3[:,:,np.newaxis]
```



- To enhance the image, we use GaussianBlur function to add a blur effect on the background to focus on foreground. We can clearly see this effect on output image. The flower has been focused on the output image.

```
blur = cv.GaussianBlur(background, (7, 7), 10)
enhanced_image = cv.add(blur, foreground)
```

- Due to the gaussian blurring, the black area will also blur and then the green part will blend in with the black area at the edges of the flower. Therefore, the edges will get distorted and sharp, so these are shown as dark edges

- **GitHub repository link**
  https://github.com/ashend99/Image-Processing