EN3160 – Image Processing and Machine Vision

Assignment 1 - Intensity Transformations and Neighborhood Filtering

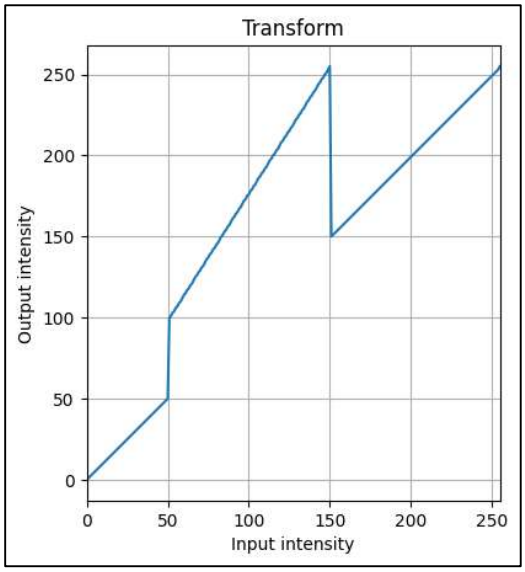
Name: Clarance L.G.S.

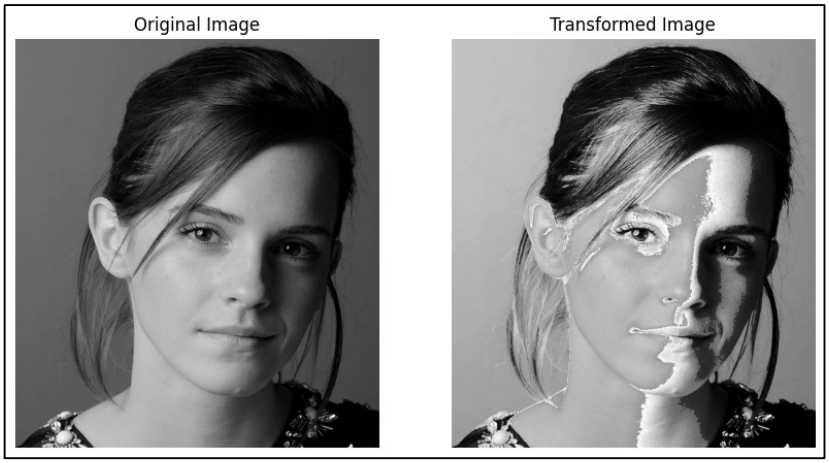
Index No: 200094R

GitHub Repository: https://github.com/ClaranceLGS/EN3160-Assignment01

Question 1 - Implementing an Intensity Transformation

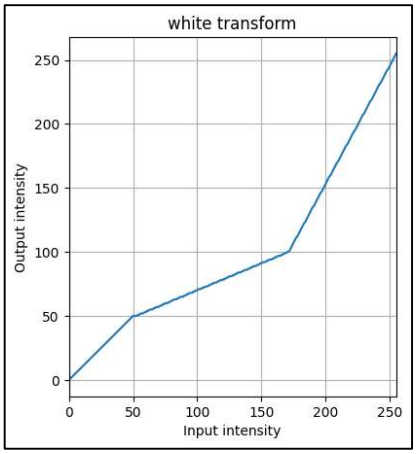
```
# Intensity Transformation
c = np.array([(50, 50), (50, 100), (150, 255),(150, 150)])
t1 = np.linspace(0, c[0, 1], c[0, 0]+1).astype('uint8')
t2 = np.linspace(c[1, 1], c[2, 1], c[2, 0]-c[1, 0]).astype('uint8')
t3 = np.linspace(c[3, 1], 255, 255-c[3, 0]).astype('uint8')
transform = np.concatenate((t1, t2, t3), axis=0).astype('uint8')
```

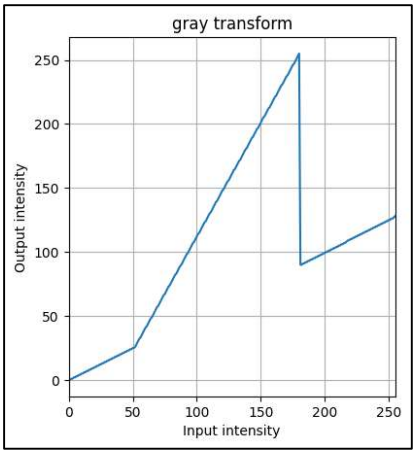




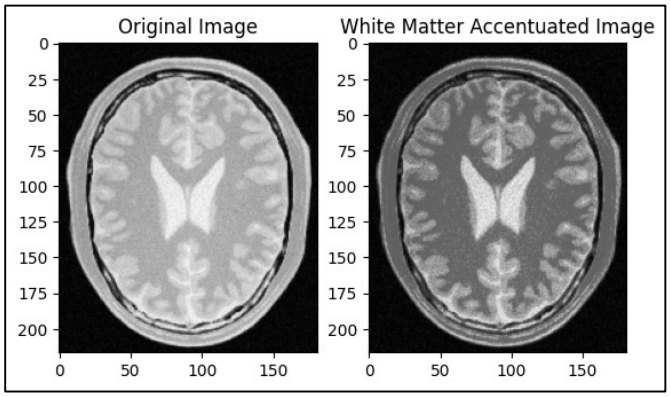
Question 2 – Enhancing Parts of an Image Using Intensity Transformation

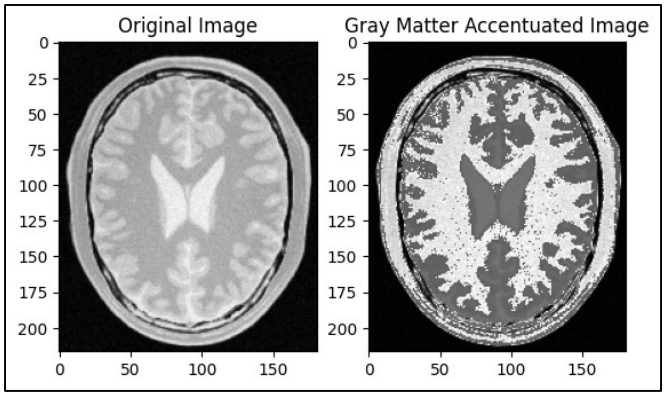
```
# White Matter Accentuation
c = np.array([(50, 50), (170, 100)])
t1 = np.linspace(0, c[0, 1], c[0, 0]+1).astype('uint8')
t2 = np.linspace(c[0, 1], c[1, 1], c[1, 0]-c[0, 0]).astype('uint8')
t3 = np.linspace(c[1 ,1], 255, 255-c[1, 0]).astype('uint8')
white_transform = np.concatenate((t1, t2, t3), axis=0).astype('uint8')
# Gray Matter Accentuation
c = np.array([(50, 25), (180, 255), (180, 90)])
t1 = np.linspace(0, c[0, 1], c[0, 0]+1).astype('uint8')
t2 = np.linspace(c[0, 1], c[1, 1], c[1, 0]-c[0, 0]).astype('uint8')
t3 = np.linspace(c[2 ,1], 128, 255-c[2, 0]).astype('uint8')
gray_transform = np.concatenate((t1, t2, t3), axis=0).astype('uint8')
```





To enhance required part of the image, the relevant intensity ranges are mapped to are larger range.

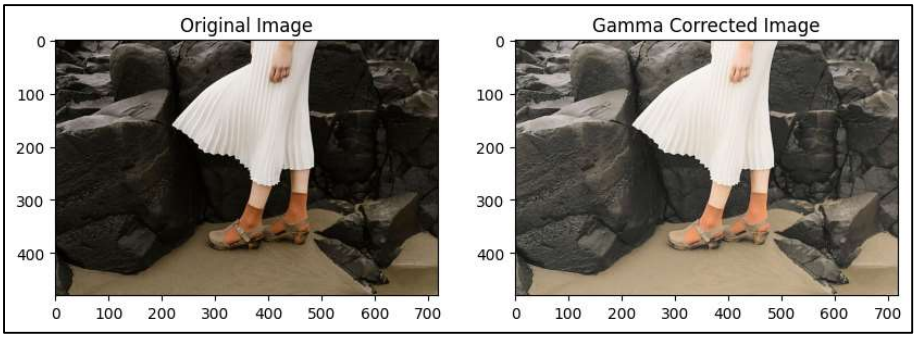


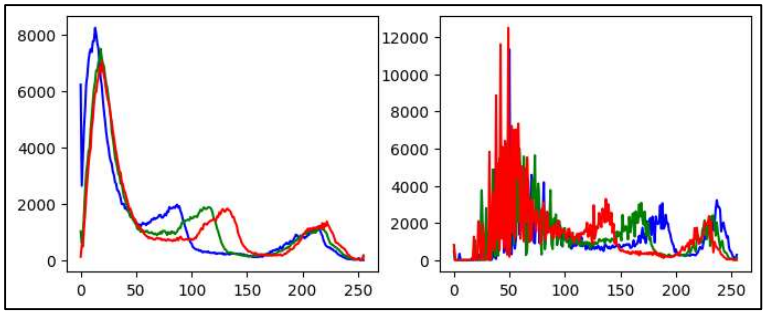


Question 3 – Applying Gamma Correction

```
# Gamma Value
gamma = 0.5
# Applying Gamma Correction to the L* Channel
img_corrected = np.power(img_l/ 255.0, gamma) * 255.0
# Puting the Processed L* Channel Back into the Image
img_lab[:,:,0] = img_corrected.astype('uint8')
```

gamma value = 0.5

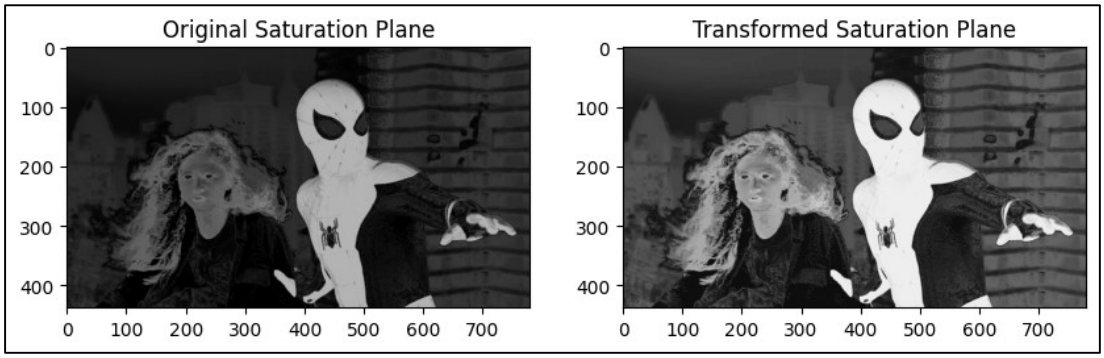




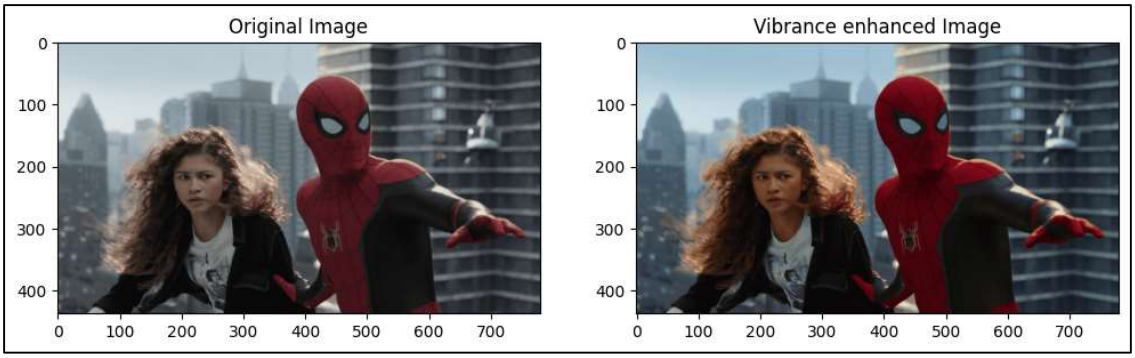
Question 4 – Vibrance Enhancement

```
# Defining the Intensity Transformation Function
def intensity_transform(x, a, sigma):
    return min(x+a*128*np.exp((x-128)**2/(-2*sigma**2)), 255)
a = 0.6 # 0 <= a <= 1
sigma = 70
# Applying the Intensity Transformation to the Saturation Plane
s_transformed = np.zeros(s.shape, dtype='uint8')
for i in range(s.shape[0]):
    for j in range(s.shape[1]):
        s_transformed[i,j] = intensity_transform(s[i,j], a, sigma)
```

The value of a for a visually pleasing output is 0.6.



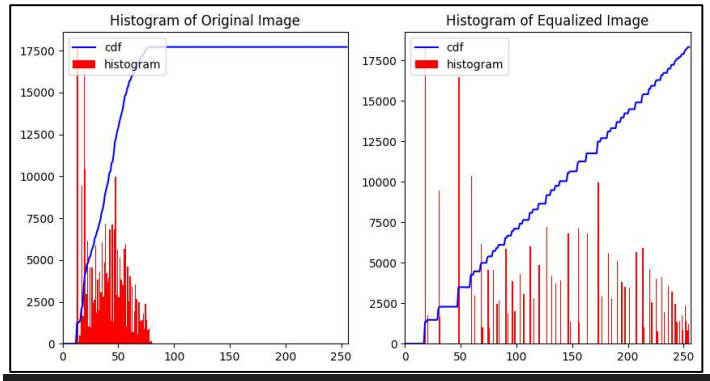

```
# Reconstructing the Image
img_hsv_transformed = cv.merge((h, s_transformed, v))
```

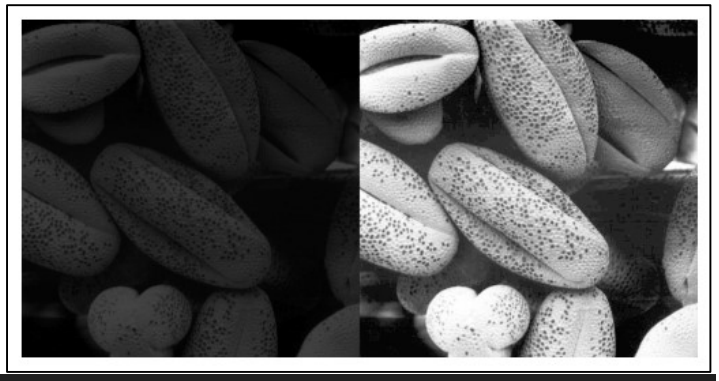


Question 5 – Histogram Equalisation

Own function to carry out Histogram Equalisation

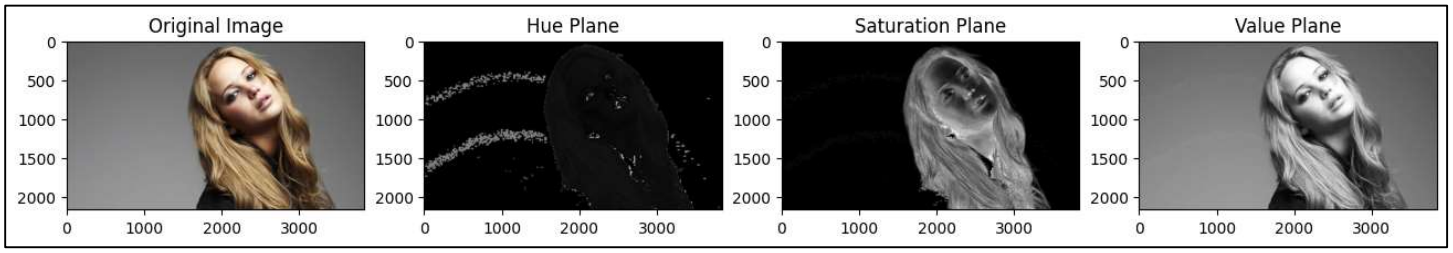
```
# Probabilities for each intensity level
prob = hist / np.sum(hist)
# Cumulative sum of pixels
cum_sum=np.zeros(256)
for i in range(len(cum_sum)):
    cum_sum[i] = np.sum(hist[:i+1])
# Equalised cumulative sum of pixels
equ_cum_sum = np.zeros(256)
for x in range(len(equ_cum_sum)):
    equ_cum_sum[x] = (cum_sum[x] * 255) / img_org.size
```





Question 6 – Histogram Equalisation for the Forground of an Image

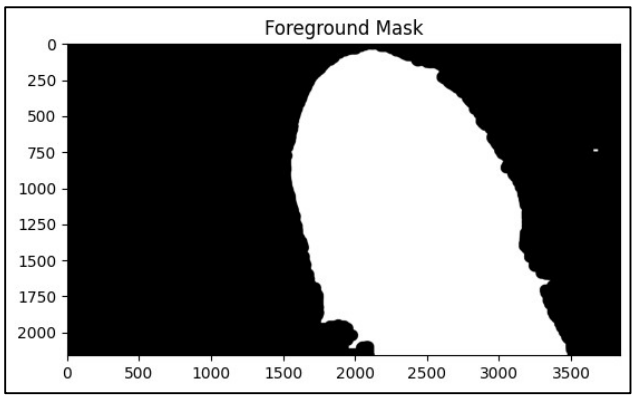
```
# Converting the Image to HSV Color Space
img_hsv = cv.cvtColor(img_org, cv.COLOR_BGR2HSV)
# Splitting the Image into Hue, Saturation and Value Planes
h, s, v = cv.split(img_hsv)
```

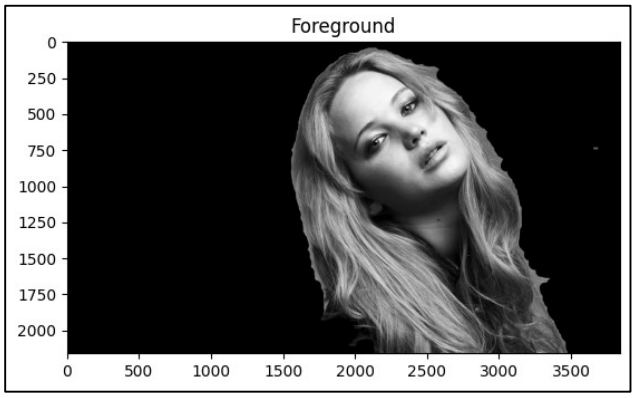


Saturation Plane was selected for extracting the foreground because the background is clearly darker than foreground.

```
# Extracting Forground Mask
threshol_value = 12
foreground_mask = cv.threshold(s, threshol_value, 255, cv.THRESH_BINARY)[1]
foreground_mask = cv.morphologyEx(foreground_mask, cv.MORPH_CLOSE, cv.getStructuringElement(cv.MORPH_ELLIPSE,
(80, 80)))
```

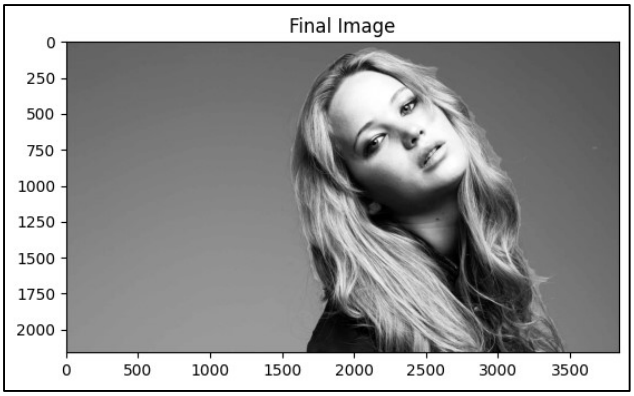
```
# Performing the masking operation using 'cv.bitwise_and' function
foreground = cv.bitwise_and(img_org, img_org, mask=foreground_mask)
# Convert the foreground to grayscale for histogram calculation
foreground_gray = cv.cvtColor(foreground, cv.COLOR_BGR2GRAY)
```





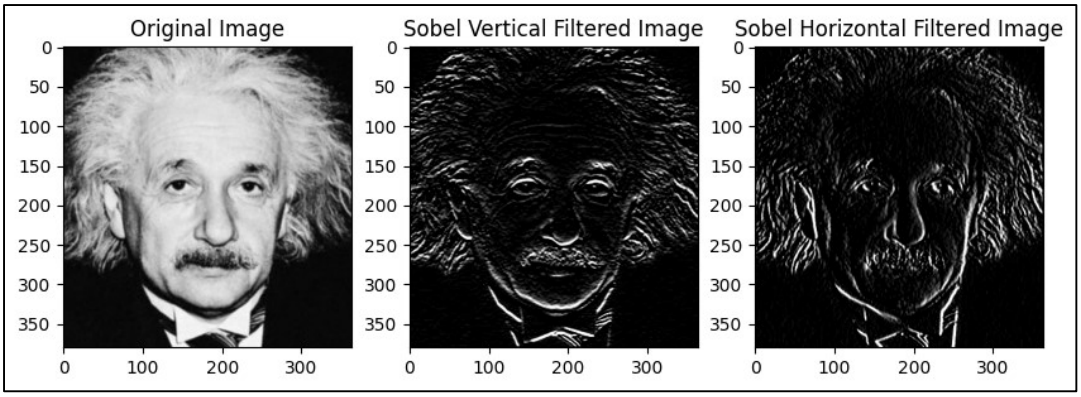

```
# Extracting the Background
```

```
background_mask = cv.bitwise_not(foreground_mask)
img_background = cv.bitwise_and(img_org, img_org, mask=background_mask)
img_background_gray = cv.cvtColor(img_background, cv.COLOR_BGR2GRAY)
# Adding the Background with Equalized Foreground
img_final = cv.add(img_background_gray, img_forground_equ)
```

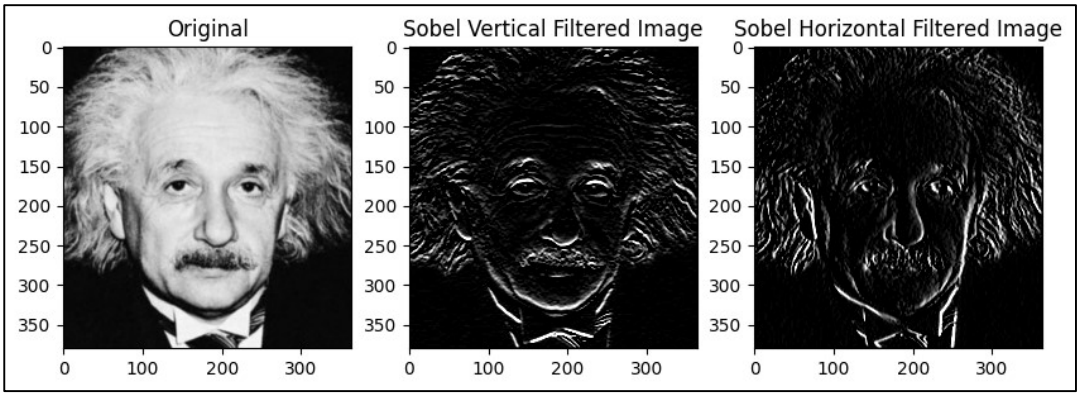


Question 7 – Sobel Filtering

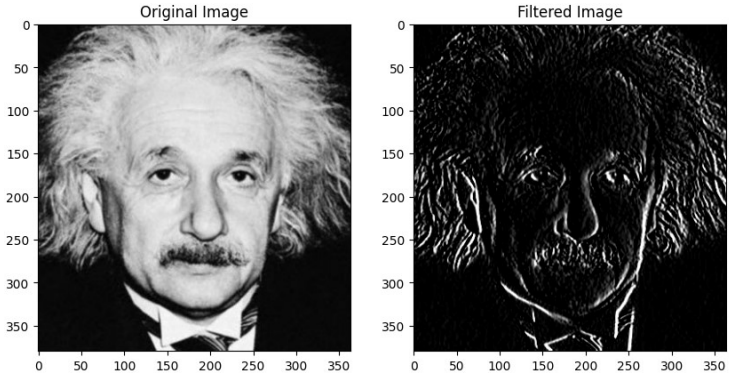
```
# Creating a sobel vertical kernel
kernel_v = np.array([[1, 2, 1], [0, 0, 0], [-1, -2 , -1]])
img_filter_v = cv.filter2D(img_org7, -1, kernel_v)
```




```
    # Sobel Filter using Custom Function
def filter(image, kernel):
    assert kernel.shape[0] % 2 == 1 and kernel.shape[1] % 2 == 1
    k_hh, k_hw = kernel.shape[0] // 2, 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((h, w), dtype='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())
    result = result * 255   # Undo normalization
    # result = np.minimum(255, np.maximum(0, result)).astype(np.uint8) # Limit between 0 and
255
return result
```



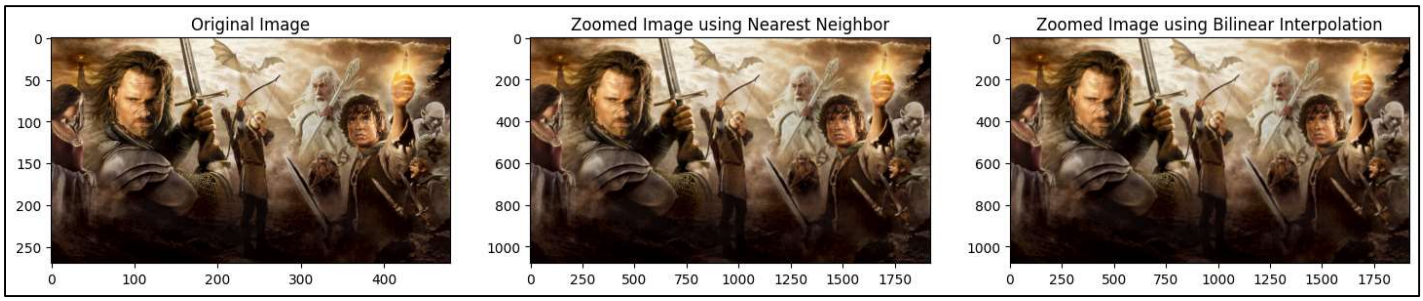

```
kernel_1 = np.array([[1], [2], [1]])
kernel_2 = np.array([[1, 0, -1]])
kernel = np.dot(kernel_1, kernel_2)
img_conv = cv.filter2D(img_org7, -1, kernel)
```



Question 8 – Zooming an Image

```
# Nearest Neighbour Function
def zoom_nearest_neighbor(image, scale_factor):
    height, width = image.shape[:2]
    new_height = int(height * scale_factor)
    new_width = int(width * scale_factor)
    zoomed_image = np.zeros((new_height, new_width, 3), dtype=np.uint8)

    for i in range(new_height):
        for j in range(new_width):
            orig_i = int(i / scale_factor)
            orig_j = int(j / scale_factor)
            zoomed_image[i, j] = image[orig_i, orig_j]

    return zoomed_image
# Bilinear Interpolation Function
def zoom_bilinear_interpolation(image, scale_factor):
    height, width = image.shape[:2]
    new_height = int(height * scale_factor)
    new_width = int(width * scale_factor)
    zoomed_image = np.zeros((new_height, new_width, 3), dtype=np.uint8)

    for i in range(new_height):
```

```
        for j in range(new_width):
            orig_i = i / scale_factor
            orig_j = j / scale_factor
            x1, y1 = int(orig_j), int(orig_i)
            x2, y2 = min(x1 + 1, width - 1), min(y1 + 1, height - 1)
            dx, dy = orig_j - x1, orig_i - y1

            # Bilinear interpolation
            interpolated_pixel = (
                (1 - dx) * (1 - dy) * image[y1, x1] +
                dx * (1 - dy) * image[y1, x2] +
                (1 - dx) * dy * image[y2, x1] +
                dx * dy * image[y2, x2]
            )
            zoomed_image[i, j] = interpolated_pixel.astype(np.uint8)

    return zoomed_image
```



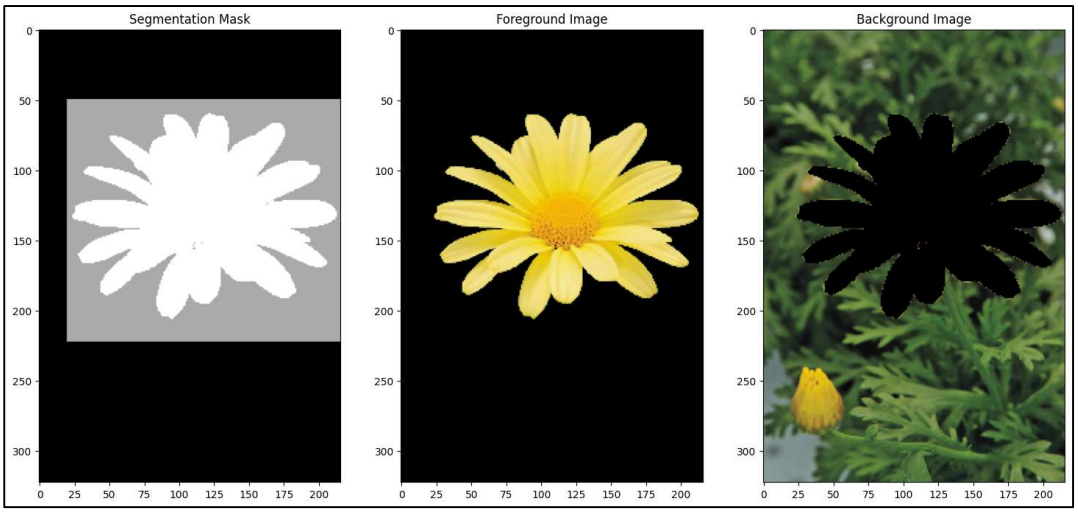
Normalized SSD (Nearest Neighbor): 31.284316486625514

Normalized SSD (Bilinear Interpolation): 39.257033179012346

Bilinear Interpolation provide better result than Nearest Neighbor Interpolation. Both zoomed images are less sharp than the original image.

Question 9 – Segmentation

```
# Initializing a Mask
mask = np.zeros(img_org9.shape[:2], dtype=np.uint8)
# Creating a Rectangles to Define the Foreground and Background
rectangles = (20, 50, img_org9.shape[1] - 20, img_org9.shape[0] - 150)
# Applying a grapCut algorithm
cv.grabCut(img_org9, mask, rectangles, None, None, 5, cv.GC_INIT_WITH_RECT)
# Creating a mask to extract the foreground and background
mask2 = np.where((mask==2)|(mask==0), 0, 1).astype('uint8')
mask3 = 1 - mask2
# Multiplying the mask with the input image to extract foreground
img_forground = img_org9 * mask2[:,:,np.newaxis]
# Background Image
img_background = img_org9 - img_forground
```




```
# Applying Gaussian Blur to the Background Image
img_background_blur = cv.GaussianBlur(img_background, (51, 51), 5)
img_background_blur = img_background_blur * mask3[:,:,np.newaxis]
# Combining the Foreground and Blurred Background Images
img_enhanced = img_background_blur + img_forground
```

The background just beyond the edge of the flower appears dark in the enhanced image because it was initially classified as probable background during the segmentation process and was subsequently blurred, creating a smooth and darkened background transition.