Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN3160
Image Processing and Machine Vision

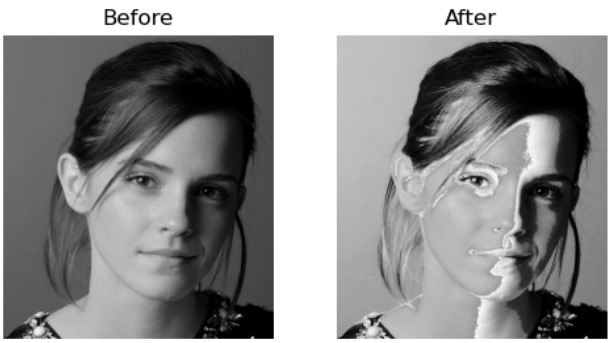
Assignment 01
Intensity Transformations and Neighborhood Filtering

210642G - Thennakoon T. M. K. R.

1.

Code of the intensity function:

```
t1 = np.linspace(0, 50 ,51).astype('uint8')
t2 = np.linspace(100, 255, 100).astype('uint8')
t3 = np.linspace(150, 255, 255-150).astype('uint8')
transform =
np.concatenate((t1,t2),axis=0).astype('uint8')
transform = np.concatenate((transform,
t3),axis=0).astype('uint8')
transformed_img = transform[img]
```



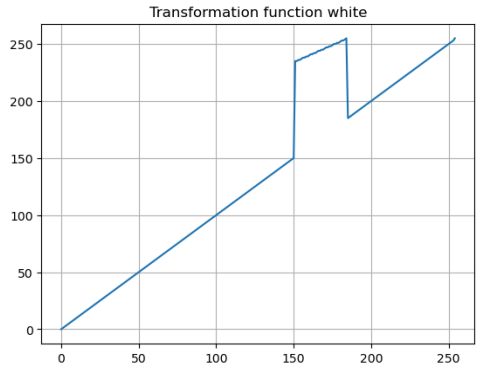
2.a

Foundout that white matters have a intensity value 164. Therefor choosed a range of 150 to 185 to accentuate.

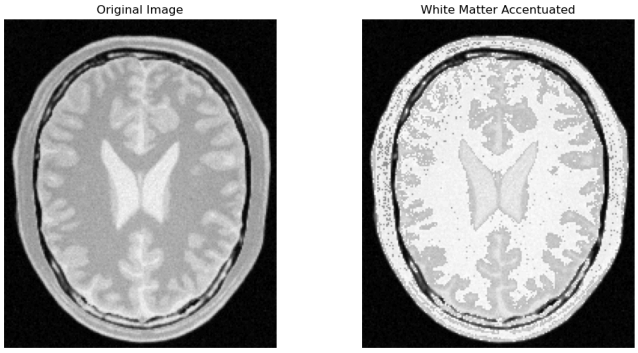
Code of the intensity transformation:

```
t1 = np.linspace(0, 150, 151).astype('uint8')
t2 = np.linspace(235, 255, 185-151).astype('uint8')
t3 = np.linspace(185, 255, 255-185).astype('uint8')
transform = np.concatenate((t1, t2),  axis=0).astype('uint8')
transform = np.concatenate((transform, t3),  axis=0).astype('uint8')
transformed_img = transform[img]
```

Plot if intensity transformation:



Output:



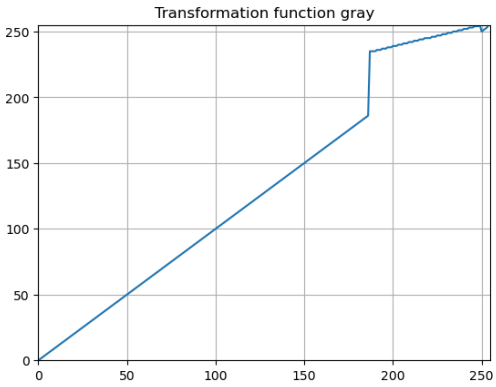
2.b

Foundout that gray white matters have a intensity value 172. Therefor choosed a range of 150 to 185 to accentuate.

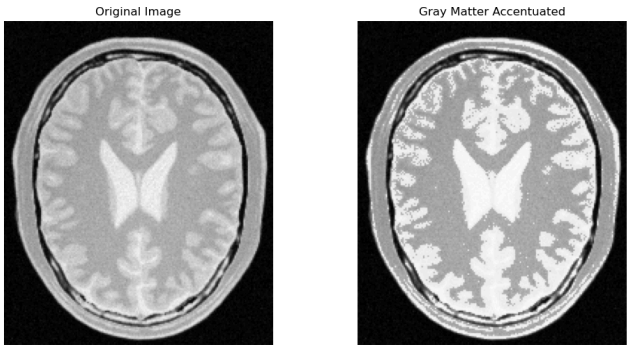
Code of the intensity transformation:

```
t1 = np.linspace(0, 186, 187).astype('uint8')
t2 = np.linspace(235, 255, 250-187).astype('uint8')
t3 = np.linspace(250, 255, 255-250).astype('uint8')
transform = np.concatenate((t1, t2), axis=0).astype('uint8')
transform = np.concatenate((transform, t3),  axis=0).astype('uint8'); transformed_img = transform[img];
```

Plot if intensity transformation:



Output:

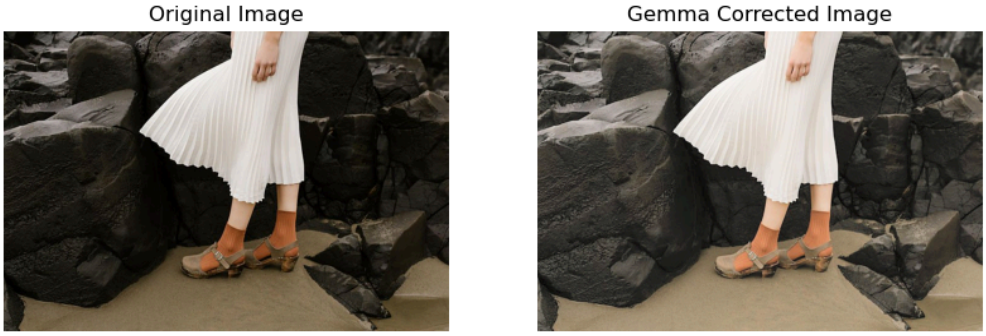


3.a

Used 0.7 as the gemma value. The image is bit of darker. Therefore increasing the range of darker pixel is better. Since, I have choosed values lower than 1 as gemma.

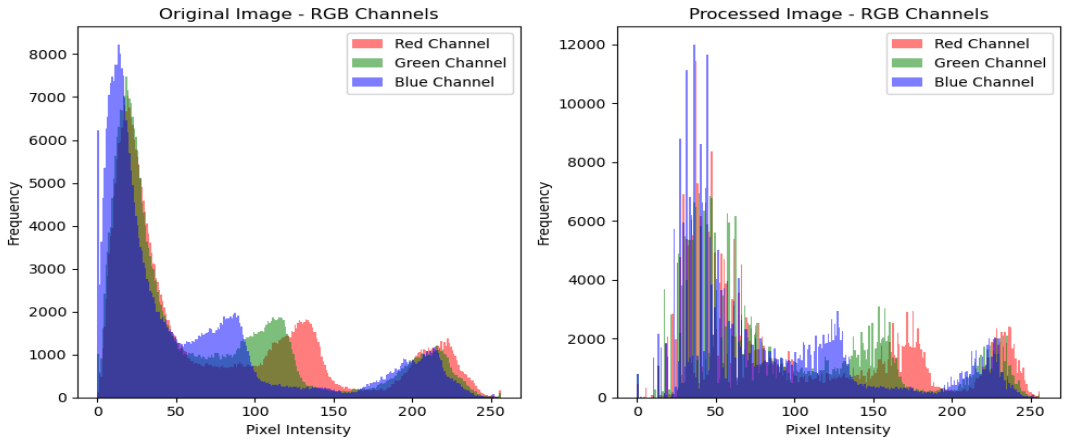
Code of applying gemma transformation to L plane **:**

```
L, a, b = cv.split(lab_image)
gemma = 0.7
t = np.array([((i/255.0)**gemma)*255 for i in  range(256)]).astype(np.uint8)
g = t[L]
```



3.b

The range of low intensity values have been increased.



4.a

Code of Dividing HSV planes:

```
img = cv.imread('Images\spider.png')
hs_img = cv.cvtColor(img, cv.COLOR_BGR2HSV)
hue, saturation, value = cv.split(hsv_img)
```

4.b

Code of applying given f(x) transformation to saturation plane:

```
a = 0.55
sigma = 70
f = np.array([i + a*128*np.exp(((i-128)**2)/(2*sigma**2)) for i in saturation]).astype(np.uint8)
f = np.clip(f,0,255)
```

4.c

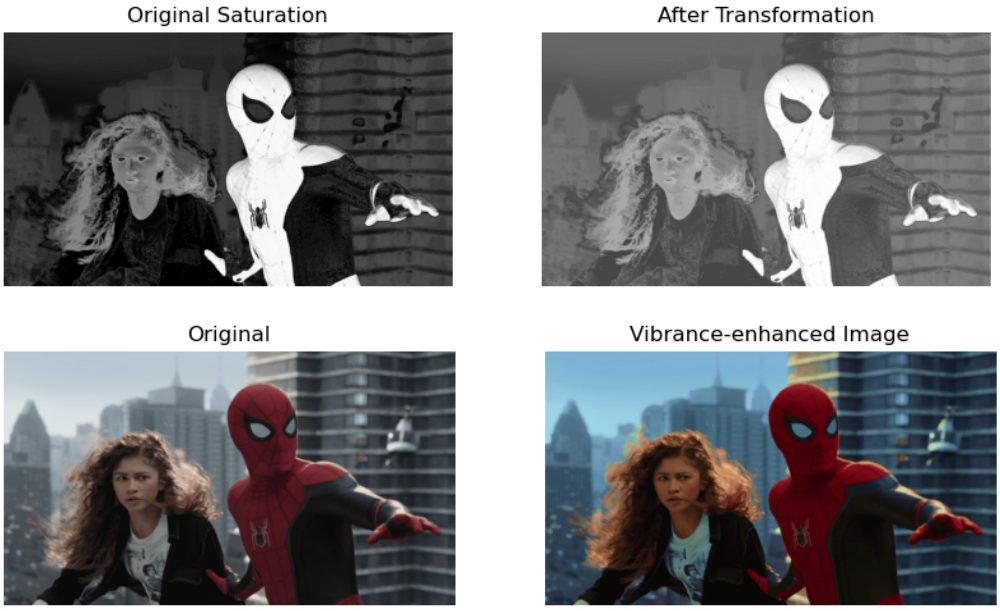
Choosed **a =0.55** for visually pleasant output after testing with different a values.

4.d

Code of Recombining planes:

```
merged_hsv = cv.merge([hue, f, value])
merged_bgr = cv.cvtColor(merged_hsv, cv.COLOR_HSV2BGR)
```

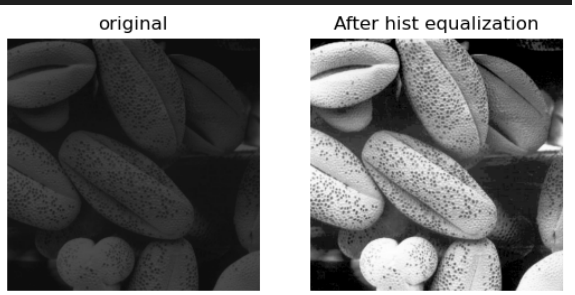
4.e



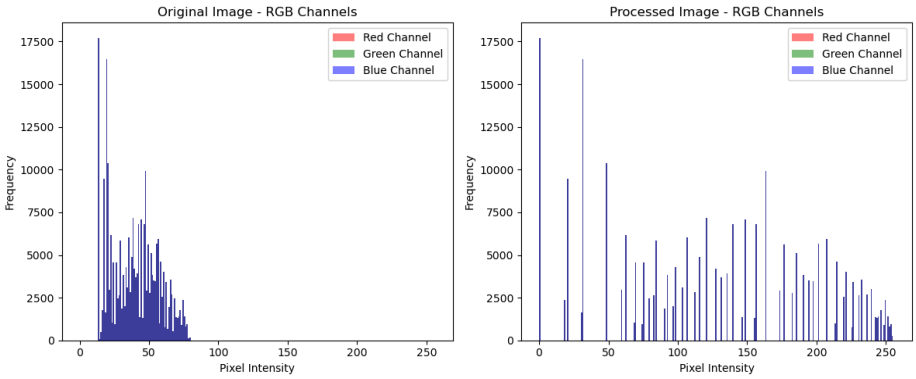
5.

Code of histogram equalization function:

```
img = cv.imread('Images\shells.tif')
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
hist = cv2.calcHist([img_gray], [0], None, [256], [0, 256])
def hist_eq(X):
    L = 256
    MN = X.shape[0]*X.shape[1]
    hist = cv.calcHist([X], [0], None, [256], [0, 256])
    s = []
    for i in range(256):
        s.append((L-1)*np.sum(hist[:i])/MN)
    eq = np.array(s).astype(np.uint8)
    return eq[X]
eq_img = hist_eq(img)
```



Histogram comparison:



6.a

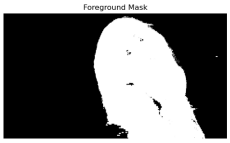
Code of dividing planes:

```
img = cv.imread("Images\jeniffer.jpg")
hsv_img = cv.cvtColor(img, cv.COLOR_BGR2HSV)
hue, saturation, value = cv.split(hsv_img)
```

6.b

Code of binary mask creation:

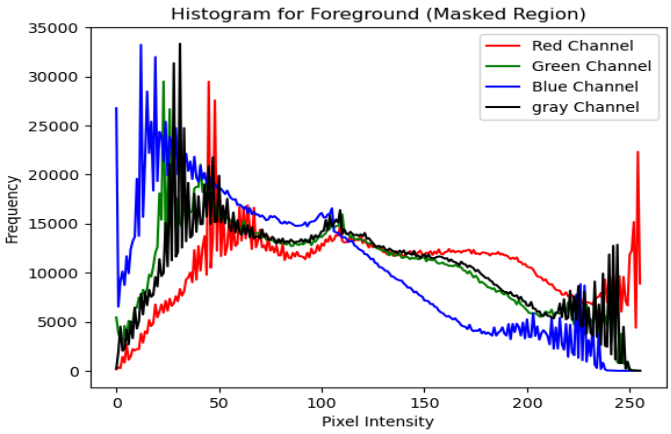
```
_,binary_mask = cv.threshold(saturation, 11, 255, cv2.THRESH_BINARY)
```

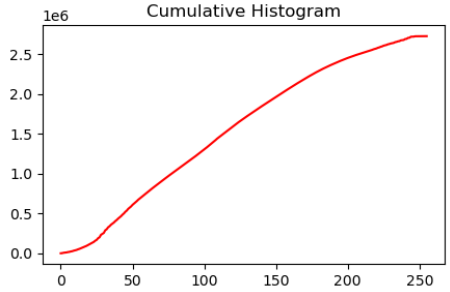


6.c

Code for creating foreground image and it's histogram:

```
foreground = cv2.bitwise_and(img, img,  mask=binary_mask)
b, g, r = cv.split(foreground)
r_hist = cv.calcHist([r], [0], binary_mask, [256], [0, 256])
g_hist = cv.calcHist([g], [0], binary_mask, [256], [0, 256])
b_hist = cv.calcHist([b], [0], binary_mask, [256], [0, 256])
gray_hist = cv.calcHist([cv.cvtColor(foreground,  cv.COLOR_BGR2GRAY)], [0], binary_mask, [256], [0,  256])
```





6.d

Code:

```
cumulative_hist = np.cumsum(hist)
```

6.e

Code for equalizing foreground histogram:

```
def hist_eq(X):
    L = 256
    MN = X.shape[0]*X.shape[1]
    s = []
    for i in range(256):
        s.append((L-1)*cumulative_hist[i]/MN)
    eq = np.array(s).astype(np.uint8)
    return eq[X]
foreground_eq = hist_eq(foreground)
```

6.f

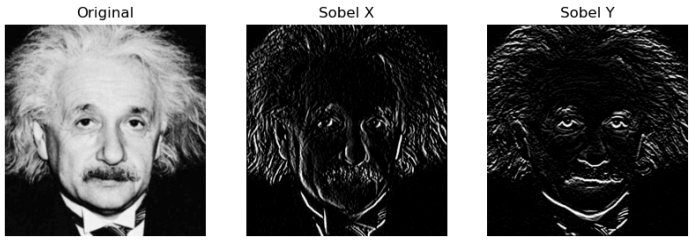
Code for getting background and combining with equalize foreground:

```
background_mask = ~binary_mask
background = cv2.bitwise_and(img, img,
                             mask=background_mask)
full_image = background + foreground_eq
```

7.a

Code:

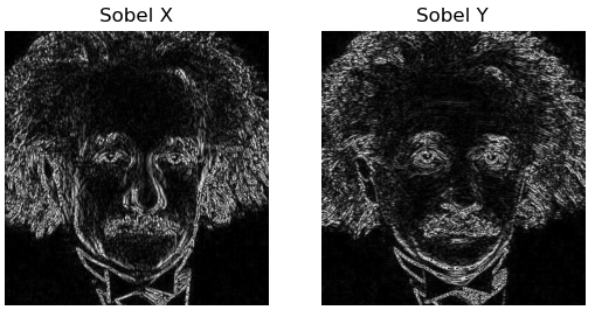
```
sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]], dtype=np.float32)
sobel_filtered_x = cv2.filter2D(img, -1, sobel_x)
```



7.b

Code for sobel filtering with custom convolution function:

```
def convolve(img, kernal):
    kernal_hw = int(kernal.shape[0]/2)
    padded_img = cv.copyMakeBorder(  img, top=kernal_hw, bottom=kernal_hw, left=kernal_hw,  right=kernal_hw,
                                     borderType=cv.BORDER_REFLECT)
    img_convolved = np.zeros_like(img, dtype=np.float64)
    kernal_flipped = cv.flip(kernal, -1)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            region = padded_img[i:i+kernal.shape[0],
j:j+kernal.shape[1]]
            img_convolved[i, j] = np.sum(region *
kernal_flipped)
    return np.uint8(np.abs(img_convolved))
```



7.c

Code and the output:

```
a = np.array([[-1, 0, 1]])
b = np.array([[1,2,1]]).reshape(-1,1)
sobel_filtered_x1 = cv.filter2D(img, -1, b)
sobel_filtered_x2 = cv.filter2D(sobel_filtered_x1, -1, a)
```

8.

Code to zoom images using mentioned two methods and calculating ssd:

```
def zoom_image(img, scale, interpolation='bilinear'):
    height, width = img.shape[:2]  ;   new_width = int(width * scale) ; new_height = int(height * scale);
    if interpolation == 'nearest':
        interpolation_method = cv.INTER_NEAREST
    elif interpolation == 'bilinear':
        interpolation_method = cv.INTER_LINEAR
    else:
        raise ValueError("Interpolation method must be   'nearest' or 'bilinear'")
    zoomed_img = cv.resize(img, (new_width,  new_height),  interpolation=interpolation_method)
    return zoomed_img
```

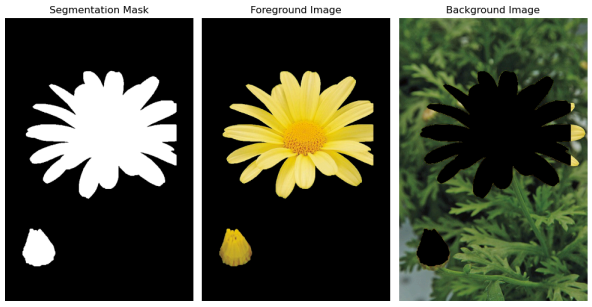
```
def normalize_ssd(original_img, scaled_img):
    assert original_img.shape == scaled_img.shape, "Images   must be the same size."
    ssd = np.sum((original_img - scaled_img) ** 2);  n_pixels = original_img.size;  normalized_ssd = ssd / n_pixels;
    return round(normalized_ssd, 2)
```

We can observe that bilinear interpolation's SSD is lower than nearest neighbours meaning bilinear method is good for zooming than nearest neighbour method.

9.a

Code for grabcut which is used to differentiate foreground and background:

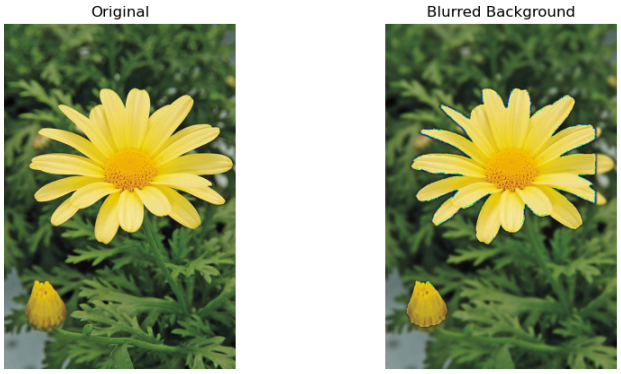
```
img = cv.imread('Images\daisy.jpg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rect = (50, 50, img.shape[1] - 100, img.shape[0] - 100)
mask = np.zeros(img.shape[:2], dtype=np.uint8)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 20,
                cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 2) | (mask == 0), 0,  1).astype('uint8')
foreground = img_rgb * mask2[:, :, np.newaxis]
background = img_rgb.copy()
background[mask2 == 1] = [0, 0, 0]
```



9.b

Code for blurring background using gaussian kernel and combining background and foreground:

```
blurred_bkg = cv2.GaussianBlur(background, (5, 5), 3)
final_img = foreground + blurred_bkg
```



9.c

When applyin gaussian blur kernel, it consider every pixel under the kernel. When it is convolved with background image, it also consider black patch shape flower also. Therefor thin black line is applyied around the edge.

Github Link for the codes:

https://github.com/kisalthennakoon/Assignments/tree/main/Computer%20Vision/Assignment1