University of Moratuwa

Department of Electronics and Telecommunication

EN3160 – Image Processing and Machine Vision

Report Activity 01

200356A

Q1.

Figure 1 Image after the intensity transform

```python
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np

points = np.array([(1,1),(50,50),(51,100),(150,255),(151,150),(255,255)])
#print(len(points))
transform = np.linspace(0,0,1)

for i in range(0,int(len(points)/2)):
    t1 = np.linspace(points[0+2*i,1], points[1+2*i,1], points[1+2*i,0]-points[0+2*i,0]+1)
    transform = np.concatenate((transform , t1), axis=0).astype('uint8')
    #print(len(t1))
#print(len(transform))

fig, ax = plt.subplots()
ax.plot(transform)
ax.set_xlabel('input intensity')  #, $f(\mathbf{x})$'
ax.set_ylabel('output intensity') #, $\\mathrm{T}[f(\mathbf{x})]$
ax.set_xlim(0,255)
ax.set_ylim(0,255)
ax.set_aspect('equal')
plt.savefig('transform1.png')
plt.show()

img = cv.imread('../emma.jpg', cv.IMREAD_GRAYSCALE)
cv.namedWindow('Emma Image', cv.WINDOW_NORMAL)
cv.imshow('Emma Image', img)
cv.waitKey(0)

transf = cv.LUT(img, transform)
cv.namedWindow('Emma Image', cv.WINDOW_NORMAL)
cv.imshow('Emma Image', transf)
cv.waitKey(0)
cv.destroyAllWindows()
```

Figure 2 Code for the given transformation

Q2.

a. White matter enhanced

Figure 3 Original image

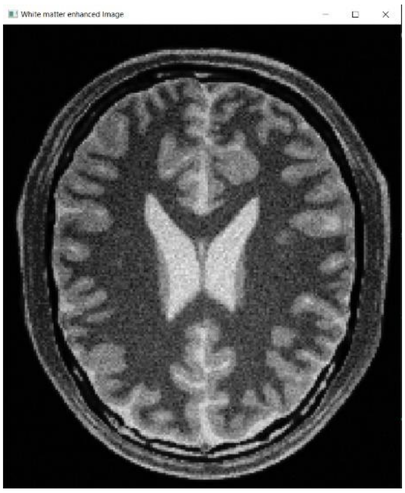


Figure 4 White matter enhanced image

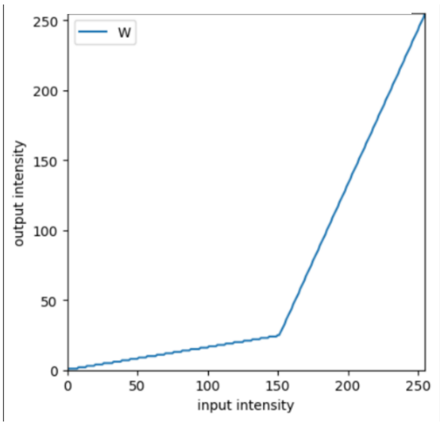


Figure 5 White matter enhancing transform

```python
Question 2.a white matter

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

points = np.array([(1,1),(100,50),(101,75),(255,255)])
#print(len(points))
transform = np.linspace(0,0,1)

for i in range(0,int(len(points)/2)):
    t1 = np.linspace(points[0+2*i,1], points[1+2*i,1], points[1+2*i,0]-points[0+2*i,0]+1)
    transform = np.concatenate((transform , t1), axis=0).astype('uint8')
    #print(len(t1))
#print(len(transform))
```

Figure 6 White matter enhancing transform code

b. Gray matter enhanced

Figure 7 Original image

Figure 8 Gray matter enhanced image

```python
points = np.array([(1,1),(75,25),(76,26),(200,225),(201,226),(255,255)])
#print(len(points))
transform = np.linspace(0,0,1)

for i in range(0,int(len(points)/2)):
    t1 = np.linspace(points[0+2*i,1], points[1+2*i,1], points[1+2*i,0]-points[0+2*i,0]+1)
    transform = np.concatenate((transform , t1), axis=0).astype('uint8')
    #print(len(t1))
#print(len(transform))
```

Figure 10 Gray matter enhancing transform code

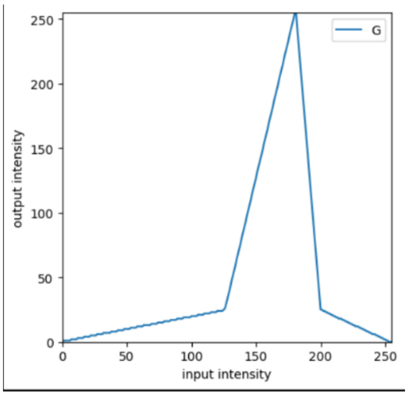


Figure 9 Gray matter enhancing transform

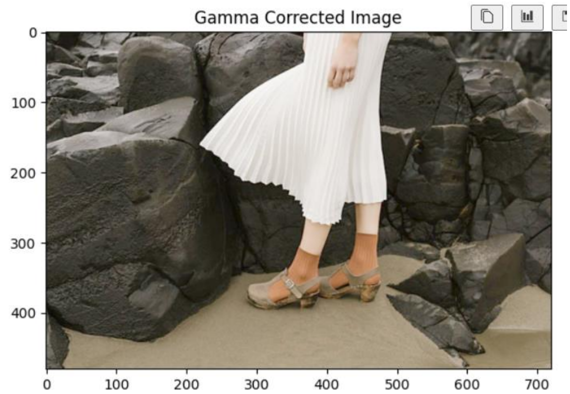
Here, most black parts are not in gray matter.

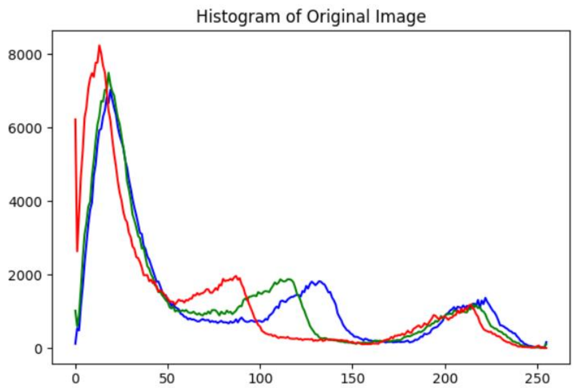
Therefore, enhancement is done only for the region where gray matter is.

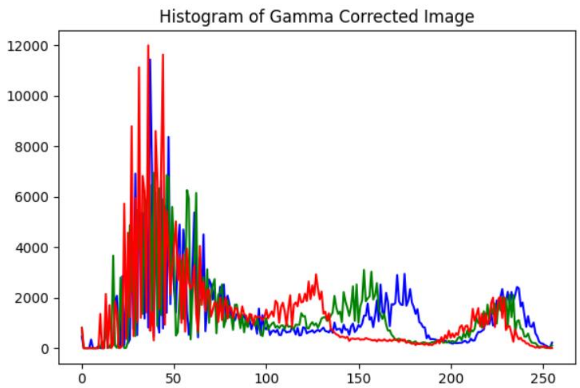
This intensity transform will enhance the gray matter area.

Q3.

Gamma value used = **0.6**



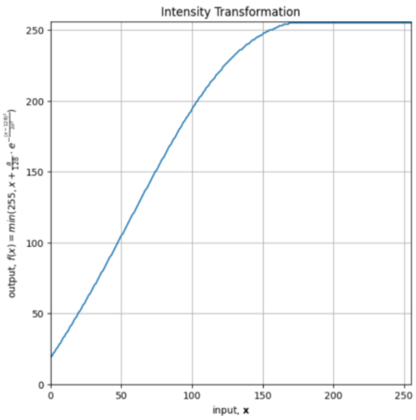




Q4.

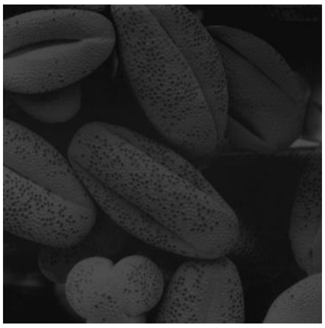
Original Image

Saturation Transformed Image

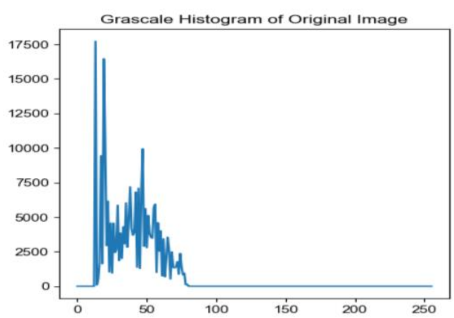


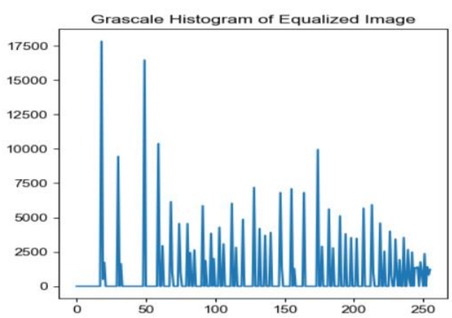

```python
img_orig = cv.imread('../spider.png', cv.IMREAD_COLOR)
sigma = 70
a = 8000
def fn(x):
    return int(min(255,x+(a/128)*np.exp(-((x-128)**2)/(2*(sigma**2)))))
img_o = cv.cvtColor(img_orig, cv.COLOR_BGR2RGB)
img_cvt = cv.cvtColor(img_orig, cv.COLOR_BGR2HSV)
h_img, s_img, v_img = cv.split(img_cvt)

#print(s_img)

for i in range(len(s_img)):
    for j in range(len(s_img[0])):
        s_img[i][j] = fn(s_img[i][j])

img_edited = cv.merge((h_img, s_img, v_img))
img = cv.cvtColor(img_edited, cv.COLOR_HSV2RGB)

print('Value of a is ',a)
```

Q5.

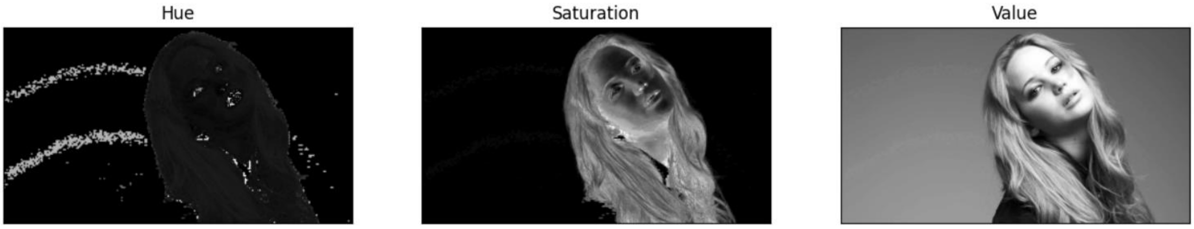


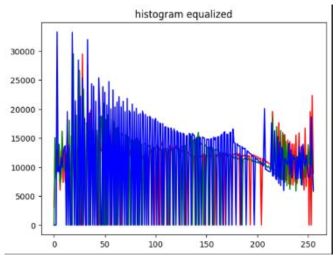

```python
def histogram_equalization(img_orig):
    histogram = np.zeros(256)
    cumilative = np.zeros(256)
    h,w = img_orig.shape #height and width
    pixels = img_orig.size
    #print(pixels)

    for i in range(h):
        for j in range(w):
            histogram[img_orig[i][j]] += 1

    cumilative[0] = histogram[0]
    for i in range(1,256):
        cumilative[i] = cumilative[i-1] + histogram[i]

    for i in range(256):
        cumilative[i] = cumilative[i]/pixels*255

    table = [np.clip(round(number), 0, 255).astype(np.uint8) for number in cumilative]
    #print(type(cumilative))
    img_done = cv.LUT(img_orig, np.array(table))

    return img_done
```

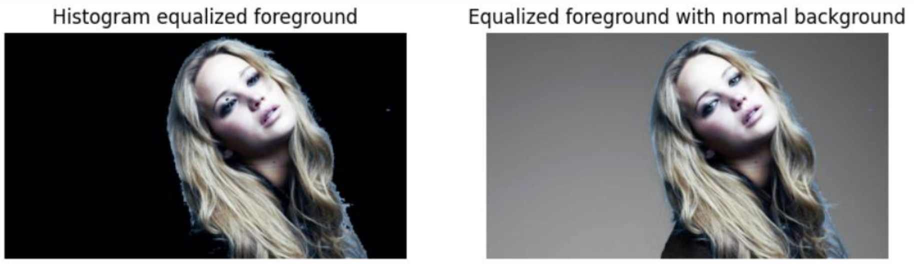




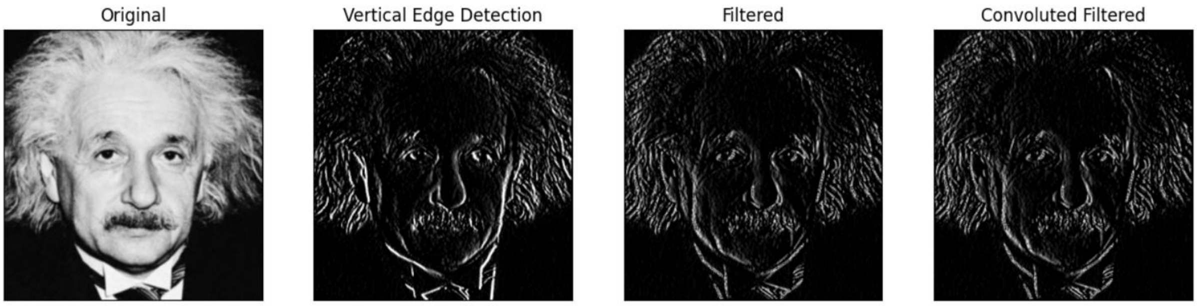
Q6.







Q7.




```python
def k_filter(image, kernel):
    h,w = image.shape[:2]
    k_h,k_w = kernel.shape[0], kernel.shape[1]
    assert k_h%2 == 1 and k_w%2 == 1, "Kernel is not accurate please check it"
    h_start, w_start = math.floor(k_h/2), math.floor(k_w/2)

    normalized_image = cv.normalize(image.astype('float'), None, 0.0, 1.0, cv.NORM_MINMAX)
    kernel_sum = np.sum(kernel)
    normalized_kernel = kernel

    #result image
    result = np.zeros(image.shape, dtype='float')

    #for convolution
    normalized_kernel = np.flipud(np.fliplr(normalized_kernel))

    for i in range(h_start, h-h_start):
        for j in range(w_start, w-w_start):
            result[i][j] = max(0,min(255.0,np.dot(normalized_image[i-h_start : i+h_start+1, j-w_start: j+w_start+1].flatten(), normalized_kernel.flatten())))
    #print(image)
    #print(result)
    result = 255.0 * result
    answer = result.astype(np.uint8)
    return answer
```

```python
def conv(image, kernel):
    h,w = image.shape[:2]
    k_h,k_w = kernel.shape[0], kernel.shape[1]
    assert k_h%2 == 1 and k_w%2 == 1, "Kernel is not accurate please check it"
    h_start, w_start = math.floor(k_h/2), math.floor(k_w/2)
    kernel = np.flipud(np.fliplr(kernel))

    #result image
    result = np.zeros(image.shape, dtype='float')

    for i in range(h_start, h-h_start):
        for j in range(w_start, w-w_start):
            result[i][j] = np.dot(image[i-h_start : i+h_start+1, j-w_start: j+w_start+1].flatten(), kernel.flatten())

    return result

img = cv.imread('../einstein.png', cv.IMREAD_GRAYSCALE)


# Define a kernel
kernel = np.array([
    (1, 0, -1),
    (2, 0, -2),
    (1, 0, -1)
], dtype='float32')

# Apply 2D filtering with the defined kernel
imgc = cv.filter2D(img, -1, kernel)

fig, axes = plt.subplots(1, 4, sharex='all', sharey='all', figsize=(15, 15))
```

```python
normalized_img = cv.normalize(img.astype('float'), None, 0.0, 1.0, cv.NORM_MINMAX)
conv1 = conv(normalized_img, kernel1)
#norm_img = np.maximum(0,np.minimum(255.0,conv2))
conv2 = conv(conv1, kernel2)
norm_img = np.maximum(0,np.minimum(255.0,conv2))


img_after = norm_img * 255
img_after = img_after.astype(np.uint8)
```

Q8.

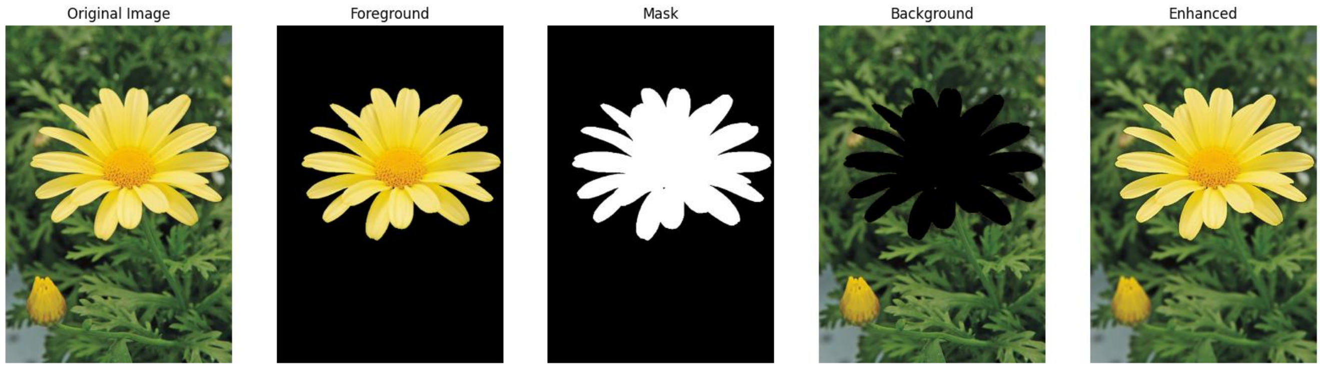
```python
def nearest_neighbour(image, scale):
    height, width, channels = image.shape
    h_zoom, w_zoom = height*scale, width*scale

    image_zoom = np.zeros((h_zoom, w_zoom, channels), dtype = np.float32)

    for h in range(h_zoom):
        for w in range(w_zoom):
            for c in range(channels):

                image_zoom[h][w][c] = image[int(math.floor(h/scale))][int(math.floor(w/scale))][c]

    return image_zoom.astype(np.uint8)
```

```python
def bilinear_interpolation(image, scale):
    height, width, channels = image.shape
    h_zoom, w_zoom = height*scale, width*scale

    image_zoom = np.zeros((h_zoom, w_zoom, channels), dtype = np.float32)

    for h in range(h_zoom):
        for w in range(w_zoom):
            for c in range(channels):

                if ((h+1)%scale==0 and (w+1)%scale==0):
                    image_zoom[h][w][c] = image[int((h+1)/scale) - 1][int((w+1)/scale) - 1][c]

                elif ((h+1)%scale==0):
                    if (w<scale):
                        image_zoom[h][w][c] = (w+1)/scale * image[int((h+1)/scale) - 1][0][c]
                    else:
                        image_zoom[h][w][c] = (1 - (w+1)%scale/scale) * image[int((h+1)/scale) - 1][int((w+1)/scale) - 1][c] + (w+1)%scale/scale * image[int((h+1)/scale) - 1][int((w+1)/scale)][c]

                elif ((w+1)%scale==0):
                    if (h<scale):
                        image_zoom[h][w][c] = (h+1)/scale * image[0][int((w+1)/scale) - 1][c]
                    else:
                        image_zoom[h][w][c] = (1 - (h+1)%scale/scale) * image[int((h+1)/scale) - 1][int((w+1)/scale) - 1][c] + (h+1)%scale/scale * image[int((h+1)/scale)][int((w+1)/scale) - 1][c]

                else:
                    if (h<scale and w<scale):
                        image_zoom[h][w][c] = (w+1)%scale/scale * (h+1)%scale/scale * image[0][0][c]
                    elif (h<scale):
                        image_zoom[h][w][c] = (h+1)%scale/scale * ((1 - (w+1)%scale/scale) * image[int((h+1)/scale) - 1][int((w+1)/scale) - 1][c] + (w+1)%scale/scale * image[int((h+1)/scale) - 1][int((w+1)/scale)][c])
                    elif (w<scale):
                        image_zoom[h][w][c] = (w+1)%scale/scale * ((1 - (h+1)%scale/scale) * image[int((h+1)/scale) - 1][int((w+1)/scale) - 1][c] + (h+1)%scale/scale * image[int((h+1)/scale)][int((w+1)/scale) - 1][c])
                    else:
                        a = (1 - (h+1)%scale/scale) * image[int((h+1)/scale) - 1][int((w+1)/scale) - 1][c] + (h+1)%scale/scale * image[int((h+1)/scale)][int((w+1)/scale) - 1][c]
                        b = (1 - (h+1)%scale/scale) * image[int((h+1)/scale) - 1][int((w+1)/scale)][c] + (h+1)%scale/scale * image[int((h+1)/scale)][int((w+1)/scale)][c]
                        image_zoom[h][w][c] = (1 - (w+1)%scale/scale) * a + (w+1)%scale/scale * b

    return image_zoom.astype(np.uint8)
```

```python
def normalized_ssd(image1, image2):

    assert image1.shape == image2.shape, "Images must have the same shape"

    ssd = np.sum((image1 - image2) ** 2)

    num_pixels = image1.shape[0] * image1.shape[1]
    max_pixel_value = 255
    nssd = ssd / (num_pixels * max_pixel_value ** 2)

    return nssd
```

```
For image 1
Normalized SSD using Nearest Neighbour:  0.0014433364007670365
Normalized SSD using Bilinear Transform:  0.0014433364007670365

For image 2
Normalized SSD using Nearest Neighbour:  0.0005491124941261907
Normalized SSD using Bilinear Transform:  0.0005491124941261907
```

09.




```python
#get image shape
h, w, _ = img_orig.shape

#define mask
mask = np.zeros((h, w), dtype=np.uint8)
print(h, w)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)

# rectangle to be given
rect = (50, 100, w, h-350)

cv.grabCut(img_orig, mask, rect, bgdModel, fgdModel, 7, cv.GC_INIT_WITH_RECT)

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

img_foreground = img_orig * mask2[:, :, np.newaxis]

img_background = cv.subtract(img_orig, img_foreground)
```

```python
kernel_size = 7
#blurred_background = cv.GaussianBlur(img_background, (kernel_size, kernel_size), 0)
blurred_background = cv.blur(img_background, (kernel_size, kernel_size), 0)
```

Once the background image gets blur, edge near the missing flower gets closer values to 0 as most of the pixels are 0. Therefore, image gets little darker.

GitHub Link repo : https://github.com/Dulan24/S5_EN3061_Image-processing

Notebook link : https://github.com/Dulan24/S5_EN3061_Image-processing/blob/master/Activity%2001/answers/codes.ipynb