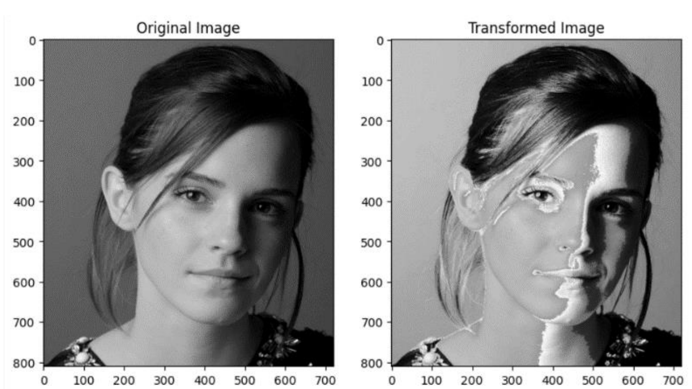
In20-S5-EN3160 – Assignment 1

Index Number – 200709K Wickramanayake R.S.D

Github Repository :- [RuchchaSD/EN3160 (github.com)]

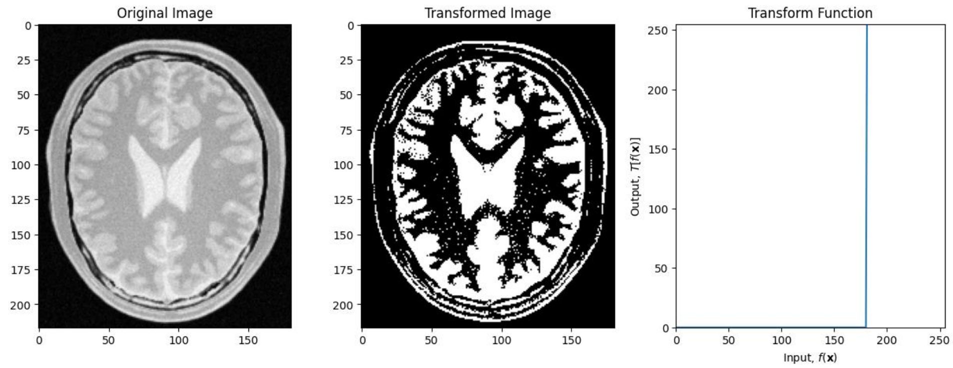
Question 1

```
%matplotlib inline
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

# Prepare Transform Function
t1 = np.linspace(0, 50, 51).astype('uint8')
t2 = np.linspace( 100, 255, 150 - 50).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')
#open original image
img_orig = cv.imread('emma.jpg', cv.IMREAD_GRAYSCALE)
#transform image
image_transformed = cv.LUT(img_orig, transform)

#plot results
fig, ax = plt.subplots(1,3, figsize=(15, 5))
ax[0].imshow(img_orig, cmap='gray')
ax[0].set_title('Original Image')
ax[1].plot(transform)
ax[1].set_xlabel(r'Input, $f(\mathbf{x})$')
ax[1].set_ylabel(r'Output, $T[f(\mathbf{x})]$')
ax[1].set_xlim(0, 255)
ax[1].set_ylim(0, 255)
ax[2].imshow(image_transformed, cmap='gray')
ax[2].set_title('Transformed Image')
plt.show()
```



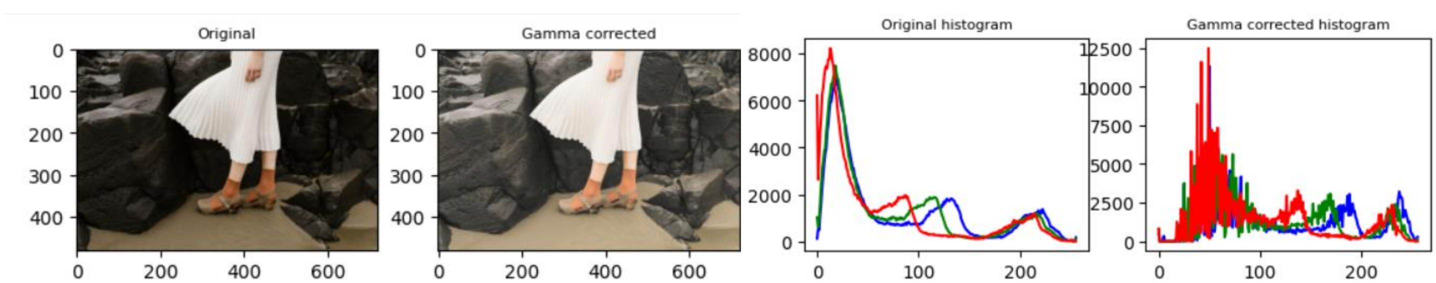
Question 2 (Only Image transformation code snippet)

```
t1 = np.linspace(0, 0, 181).astype('uint8')
t2 = np.linspace( 255, 255, 255 - 180).astype('uint8')
transform = np.concatenate((t1, t2), axis=0).astype('uint8')
#open original image
img_orig = cv.imread('BrainProtonDensitySlice9.png', cv.IMREAD_GRAYSCALE)
#transform image
image_transformed = cv.LUT(img_orig, transform)
```



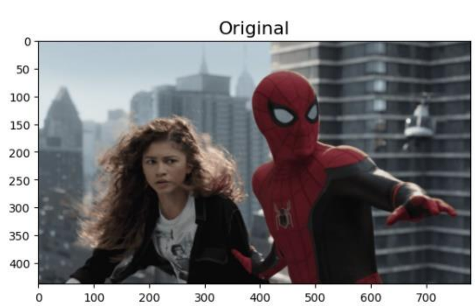
Question 3 (Only Image transformation code snippet)

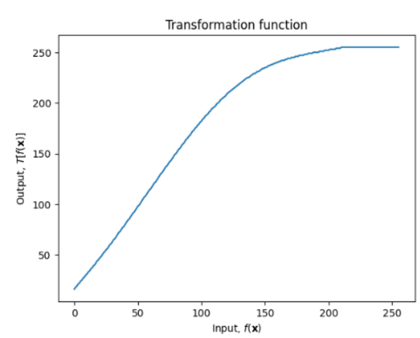
```
# read the image
img_orig = cv.imread('highlights_and_shadows.jpg', cv.IMREAD_COLOR)
# convert to LAB color space
img_gamma = cv.cvtColor(img_orig, cv.COLOR_BGR2LAB)
img_L = img_gamma[:,:,0]
# apply gamma correction to the L channel
gamma = 2 # A gamma value of 2 gave a visually pleasing output
table = np.array([(i/255.0)**(1.0/gamma)*255.0 for i in
np.arange(0,256)]).astype('uint8')
img_L_gamma = cv.LUT(img_L, table)
img_gamma[:,:,0] = img_L_gamma
img_gamma = cv.cvtColor(img_gamma, cv.COLOR_LAB2RGB)
img_orig = cv.cvtColor(img_orig, cv.COLOR_BGR2RGB)
```



Question 4 (Only Image transformation code snippet)

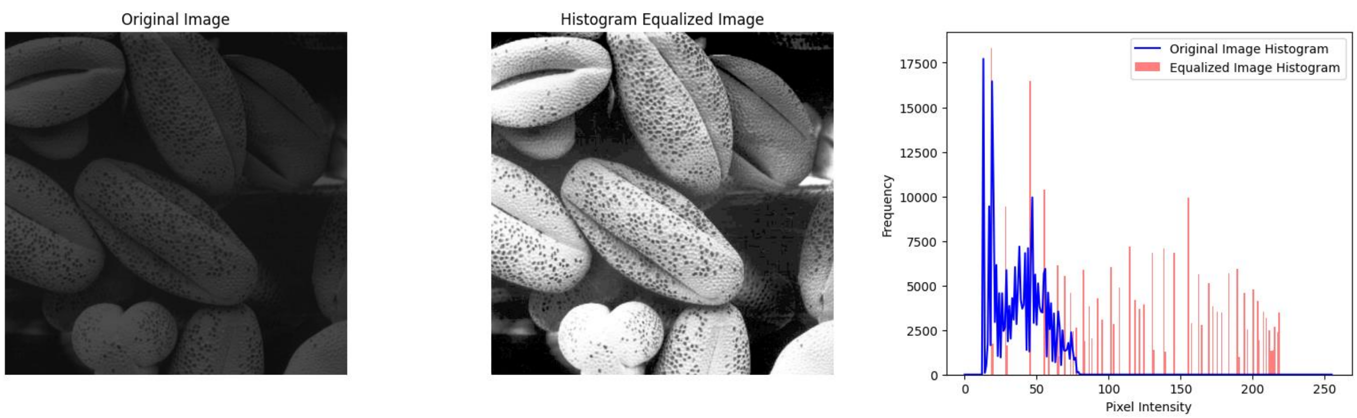
```
img_orig = cv.imread('spider.png', cv.IMREAD_COLOR)
# Convert img_orig to HSV
img_hsv = cv.cvtColor(img_orig, cv.COLOR_BGR2HSV)
# Get the saturation channel
img_s = img_hsv[:,:,1]

# Define the transformation function
def transform(x, a, sigma):
    return min(x + a * 128 * np.exp(-((x - 128) ** 2) / (2 * sigma ** 2)), 255)

a = 0.7
sigma = 70
# Generate the LUT using the transformation function
table = np.array([transform(i, a, sigma) for i in np.arange(0,
256)]).astype('uint8')
# Apply the LUT to the saturation channel
img_s_transformed = cv.LUT(img_s, table)

#recombine image
img_hsv[:,:,1] = img_s_transformed
img_enhanced = cv.cvtColor(img_hsv, cv.COLOR_HSV2BGR)

# Plot original and transformed image
f, axarr = plt.subplots(1, 2, figsize=(15, 10))
axarr[0].imshow(cv.cvtColor(img_orig, cv.COLOR_BGR2RGB))
axarr[0].set_title('Original', size=16)
axarr[1].imshow(cv.cvtColor(img_enhanced, cv.COLOR_BGR2RGB))
axarr[1].set_title('Enhanced', size=16)
plt.show()
fig, ax = plt.subplots()
ax.plot(table)
ax.set_xlabel(r'Input, $f(\mathbf{x})$')
ax.set_ylabel(r'Output, $T[f(\mathbf{x})]$')
ax.set_title('Transformation function')
```





Question 5 (Only Image transformation code snippet)

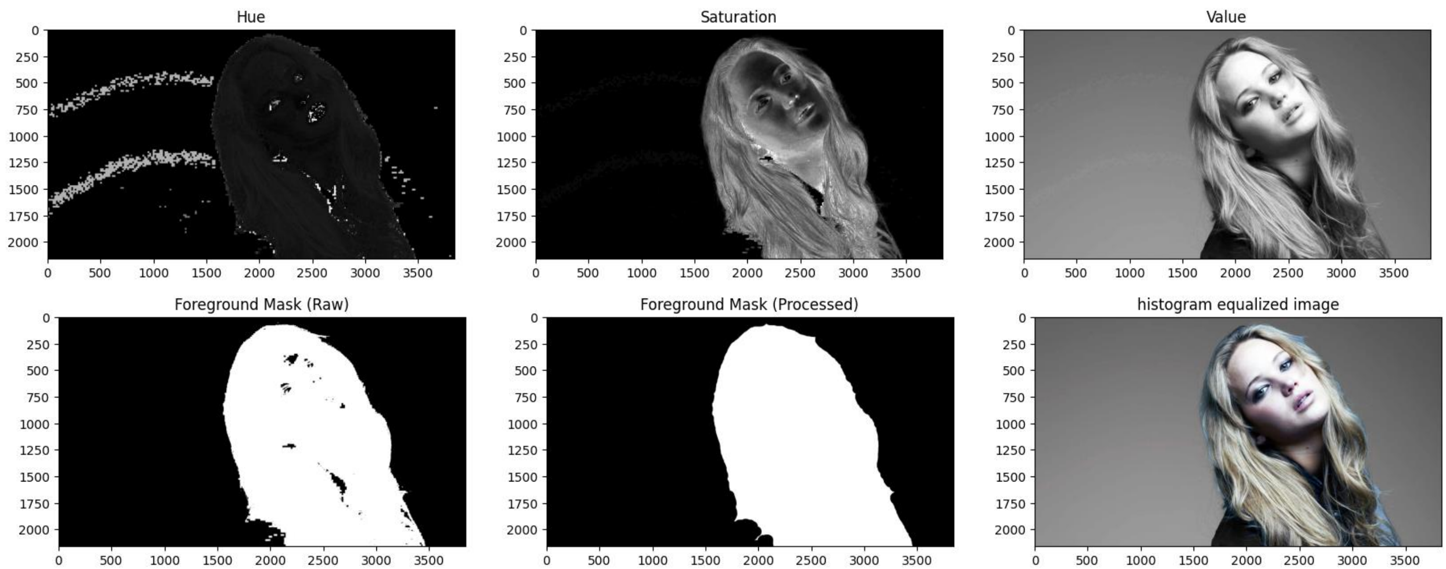
```
input_image = cv.imread("shells.tif", cv.IMREAD_GRAYSCALE)
# Calculate histogram
rows, cols = input_image.shape
histogram = np.zeros((256,), dtype=np.uint16)
for i in range(rows):
    for j in range(cols):
        intensity = input_image[i, j]
        histogram[intensity] += 1
# calculate cdf
cdf = np.zeros((256,), dtype=np.uint16)
for i in range(256):
    for j in range(i + 1):
        cdf[i] += histogram[j] * (255 / (rows * cols))
    cdf[i] = round(cdf[i], 0)
cdf = cdf.astype(np.uint16)

output_image = np.zeros_like(input_image)
for i in range(rows):
    for j in range(cols):
        intensity = input_image[i, j]
        output_image[i, j] = cdf[intensity]
```

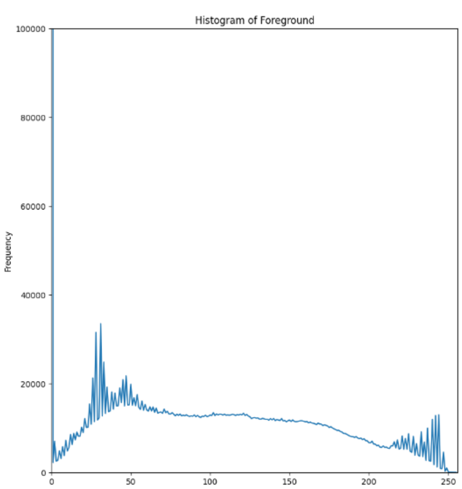


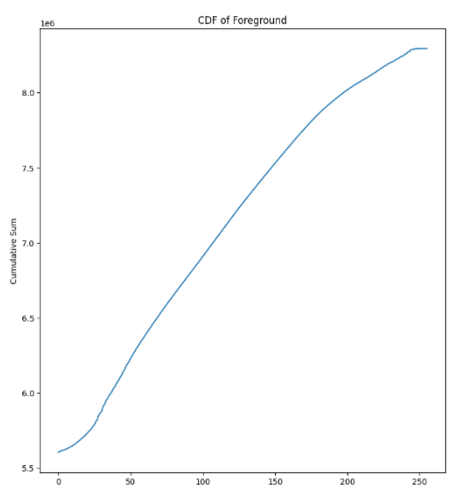
Question 6 (Only Image transformation code snippet)

```
img = cv2.imread('jeniffer.jpg', cv2.IMREAD_COLOR)
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
h, s, v = cv2.split(img_hsv)
# Define a threshold value for saturation
threshold_value = 20
# Create a binary mask for the foreground based on the saturation channel
foreground_mask = np.zeros_like(s)
foreground_mask[s > threshold_value] = 255
foreground_raw = foreground_mask.copy()
foreground_mask = cv2.morphologyEx(foreground_mask, cv2.MORPH_CLOSE,
cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (80, 80)))
foreground_mask = cv2.erode(foreground_mask,
cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (20, 20)), iterations=1)
background_mask = cv2.bitwise_not(foreground_mask)
```

```
# Apply the mask to image to extract foreground objects
img= cv2.cvtColor(img, cv2.COLOR_BGR2RGB);
foreground_r = cv2.bitwise_and(img[:,:,0], foreground_mask)
foreground_g = cv2.bitwise_and(img[:,:,1], foreground_mask)
foreground_b = cv2.bitwise_and(img[:,:,2], foreground_mask)
img_foreground = cv2.merge((foreground_b, foreground_g, foreground_r))
img_foreground = cv2.cvtColor(img_foreground, cv2.COLOR_BGR2GRAY)
#get the histogram of the foreground
hist_foreground = cv2.calcHist([img_foreground],[0],None,[256],[0,256])
#calculate the cumulative distribution function
cdf_foreground = hist_foreground.cumsum()
#histogram equalize the foreground
foreground_r = cv2.equalizeHist(foreground_r)
foreground_g = cv2.equalizeHist(foreground_g)
foreground_b = cv2.equalizeHist(foreground_b)
img_masked = cv2.merge((foreground_r, foreground_g, foreground_b))
background_mask = cv2.bitwise_not(foreground_mask)
#extract the background
background_r = cv2.bitwise_and(img[:,:,0],background_mask)
background_g = cv2.bitwise_and(img[:,:,1],background_mask)
background_b = cv2.bitwise_and(img[:,:,2],background_mask)
background = cv2.merge((background_b, background_g, background_r))
img_new = cv2.add(img_masked, background)
```

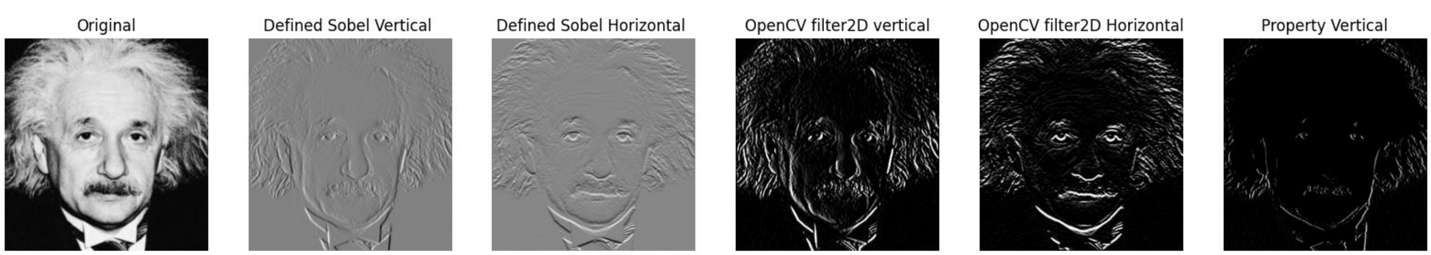






Question 7

```
# write a function to fiter the given image with given kernel
def filterImg(image, kernel):
    img_h, img_w = image.shape
    kernel_h, kernel_w = kernel.shape
    pad_h = kernel_h // 2
    pad_w = kernel_w // 2
    image_float = cv.normalize(image.astype('float'), None, 0.0, 1.0,
cv.NORM_MINMAX)
    output = np.zeros(image.shape,'float')
    for y in range(pad_h,img_h-pad_h):
        for x in range(pad_w,img_w-pad_w):
            output[y, x] = np.dot(image_float[y - pad_h:y + pad_h + 1, x -
pad_w:x + pad_w + 1].flatten(), kernel.flatten())
    return output

# Read image
img = cv.imread('einstein.png', cv.IMREAD_GRAYSCALE)

# Sobel horizontal
kernel_h = np.array([(-1, -2, -1), (0, 0, 0), (1, 2, 1)], dtype='float')
# Sobel vertical
kernel_v = np.array([(-1, 0, 1), (-2, 0, 2), (-1, 0, 1)], dtype='float')
# Using Propertry of Convolution
kernel_1 = np.array([(1), (2), (1)], dtype='float')
kernel_2 = np.array([(1, 0, -1)], dtype='float')
# Using OpenCV
imgcv = cv.filter2D(img,-1,kernel_v)
imgch = cv.filter2D(img,-1,kernel_h)
# Using Defined Function
imgv = filterImg(img, kernel_v)
imgh = filterImg(img, kernel_h)
# Using Property of Convolution
imgpv = cv.filter2D(img, -1, kernel_1)
imgpv = cv.filter2D(imgpv, -1, kernel_2)
```



Question 8

```python
def zoom_nearest_neighbor(image, zoom_factor):
    height, width, _ = image.shape
    new_height = int(height * zoom_factor)
    new_width = int(width * zoom_factor)
    new_image = np.zeros((new_height, new_width, 3), dtype=np.uint8)

    for y in range(new_height):
        for x in range(new_width):
            src_x = int(x / zoom_factor)
            src_y = int(y / zoom_factor)
            new_image[y, x] = image[src_y, src_x]

    return new_image

def zoom_bilinear(image, zoom_factor):
    height, width, _ = image.shape
    new_height = int(height * zoom_factor)
    new_width = int(width * zoom_factor)
    new_image = np.zeros((new_height, new_width, 3), dtype=np.uint8)

    for y in range(new_height):
        for x in range(new_width):
            src_x = x / zoom_factor
            src_y = y / zoom_factor

            x1, y1 = int(src_x), int(src_y)
            x2, y2 = x1 + 1, y1 + 1

            if x2 >= width:
                x2 = width - 1
            if y2 >= height:
                y2 = height - 1

            dx = src_x - x1
            dy = src_y - y1

            new_image[y, x] = (
                (1 - dx) * (1 - dy) * image[y1, x1] +
                dx * (1 - dy) * image[y1, x2] +
                (1 - dx) * dy * image[y2, x1] +
                dx * dy * image[y2, x2]
            ).astype(np.uint8)

    return new_image
```

```
Image[ 1 ] ssd_nearest :  31.284316486625514  ssd_bilinear :  39.257033179012346
Image[ 2 ] ssd_nearest :  11.902013310185184  ssd_bilinear :  16.21177662037037
Image[ 3 ] ssd_nearest :  17.171342909907853  ssd_bilinear :  22.376757387099232
Image[ 4 ] ssd_nearest :  78.73781724215534  ssd_bilinear :  81.65874063625257
Image[ 5 ] ssd_nearest :  50.57724609375  ssd_bilinear :  53.70764858217593
Image[ 6 ] ssd_nearest :  30.553995949074075  ssd_bilinear :  35.51769129372428
Image[ 7 ] ssd_nearest :  27.964142659505207  ssd_bilinear :  30.22005997721354
Image[ 8 ] ssd_nearest :  14.37148410910911  ssd_bilinear :  19.219769269269268
Image[ 9 ] ssd_nearest :  21.148100694444445  ssd_bilinear :  26.669085590277778
Image[ 10 ] ssd_nearest :  21.42844935276991  ssd_bilinear :  25.21551477169893
Image[ 11 ] ssd_nearest :  94.7471351111111  ssd_bilinear :  94.56599555555556
```

<u>**Question 9**</u>

```
rect = (50,150,500,380)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
mask = np.zeros(img.shape[:2],np.uint8)
cv.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
foreground = img * mask2[:, :, np.newaxis]
background = img - foreground
blurred_background = cv.GaussianBlur(background, (13, 13), 0)
final_image = blurred_background + foreground
final_image = cv.cvtColor(final_image, cv.COLOR_BGR2RGB)
```

Q9.(c) The dark edges were introduced due to gaussian blur applied to the background image. The part were the foreground flower was present was black when gaussian blur was applied. So the kernel for gaussian blur would have taken the black pixels in the cut part for calculating values for the edges in the final image.