DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

UNIVERSITY OF MORATUWA

EN3160 - Image Processing and Machine Vision

Assignment 1

Intensity Transformations and Neighborhood Filtering

Name:	Mirihagalla M.K.D.M.
Index:	200397A
GITHUB:	click here to visit.

Question 1

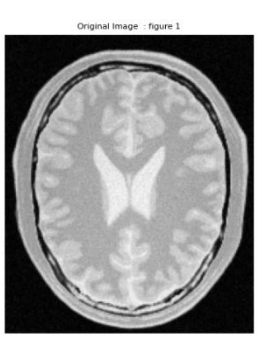

```
#  Creating the transformation function
t1 = np.arange(0, 50, 1)
t2 = np.arange(101, 256, 1.55)
t3 = np.arange(150, 256, 1)
T = np.concatenate((t1, t2, t3), axis=0)
#  Reading the image
img_original = cv.imread('emma.jpg',cv.IMREAD_GRAYSCALE)
#  Transforming the image
img_transformed =  cv.LUT(img_original, T)
```

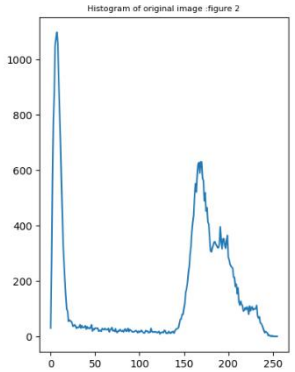
Darker areas and brighter regions in the input image have not been changed after applying the intensity transformation. The mid intensity region has been shifted up in terms of the intensity.

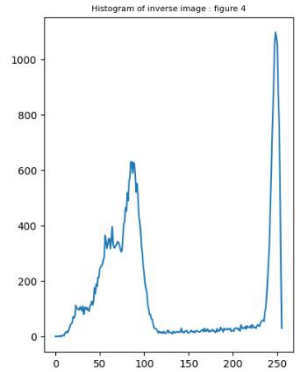
Question 2

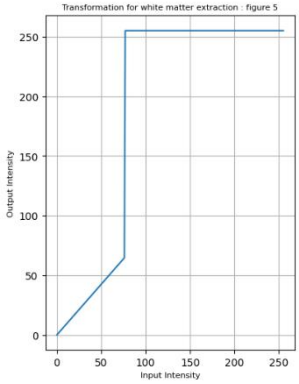
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('BrainProtonDensitySlice9.png', cv2.IMREAD_GRAYSCALE)
# Analyze histogram
hist = cv2.calcHist([img],[0],None,[256],[0,256])
#inverse of image
img1 = 255 - img
# Analyze histogram to find intensity transformation
hist = cv2.calcHist([img1],[0],None,[256],[0,256])

#definiting a intenity transformation array to use cv.LUT
threshold_white_matter = 65
t1 = np.arange(0, threshold_white_matter, 0.85)
t2 = 255 - np.zeros(256 - len(t1))
T = np.concatenate((t1, t2))

# histogram of white matter highlighted image
hist = cv2.calcHist([img2],[0],None,[256],[0,256])

threshold_gray_matter_start = 80
threshold_gray_matter_end = 120

t_mid = np.arange(threshold_gray_matter_start, threshold_gray_matter_end, 1)
t_left = 255-np.zeros(threshold_gray_matter_start)
t_right = 255 - np.zeros(256 - threshold_gray_matter_end)
T = np.concatenate((t_left, t_mid, t_right))
img3 = cv2.LUT(img1,T).astype(np.uint8)

# histogram of gray matter highlighted image
hist = cv2.calcHist([img3],[0],None,[256],[0,256])
```

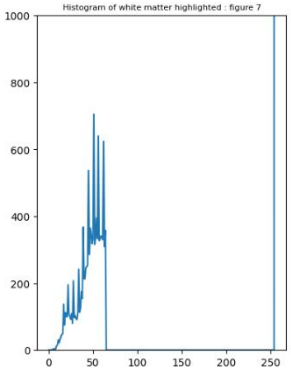
Instead of just changing the image's brightness directly, I made a few adjustments to the image in a specific way. When I reversed the intensities, it made certain details stand out better. This made the gray and white parts of the brain look darker, helping us notice them more easily. It is easy to analyze them with histograms.

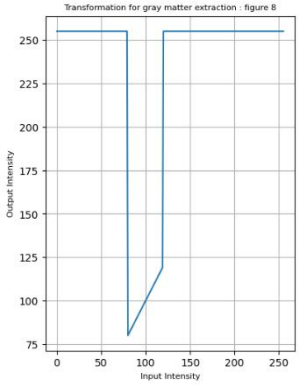


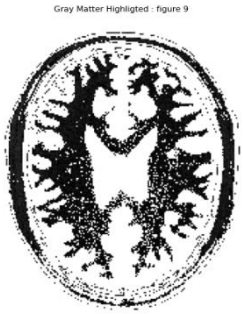


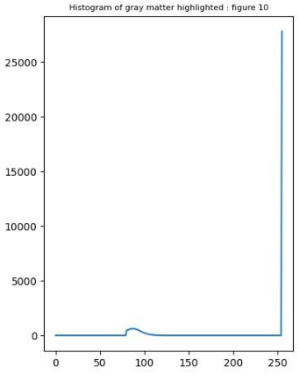








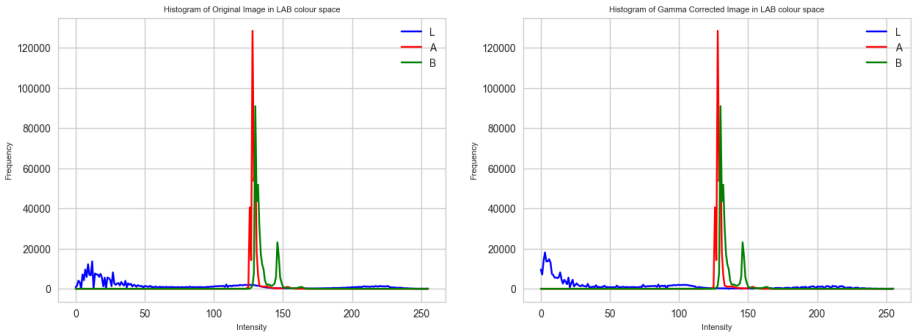




Process followed to extract white and gray matter. (Highlighted as dark areas)

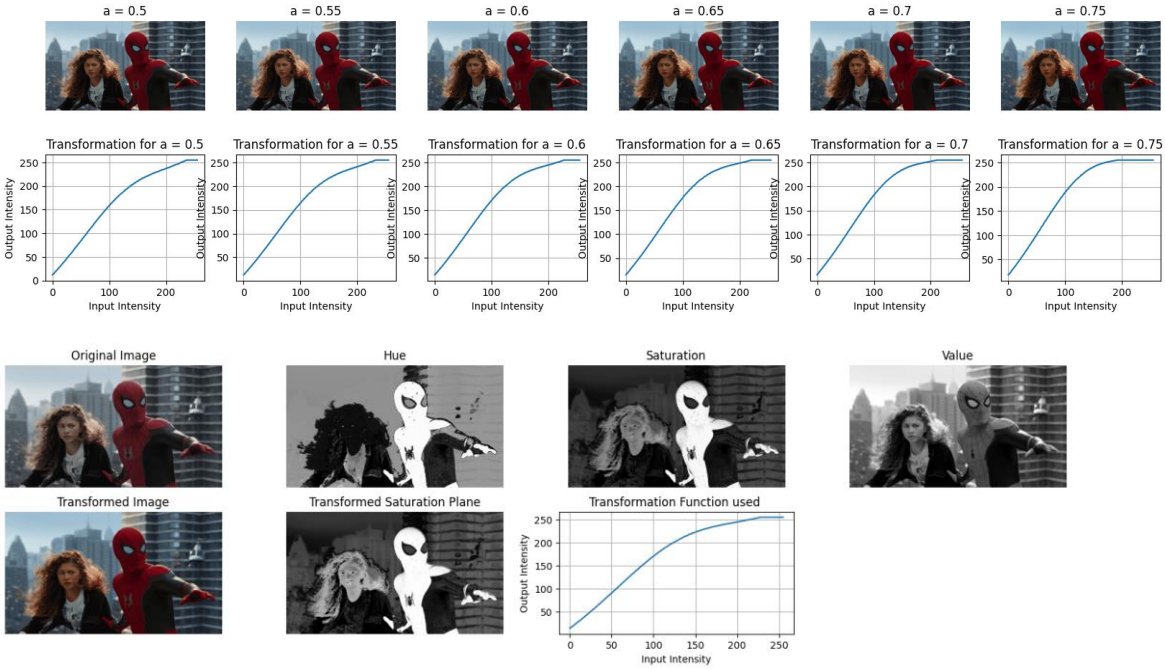
Question 3

Applying gamma correction for couple of values it looks like $\gamma = 1.28$ is fairly good for the correction.

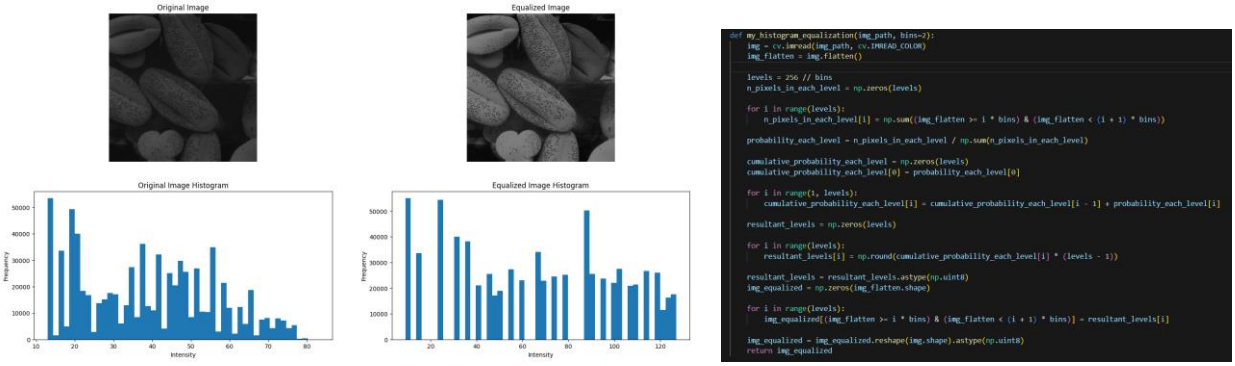


Question 4

I have tasted several a value and it seems to be $a = 0.6$ seems pleasant. i.e., it has improved the vibrance of the image.

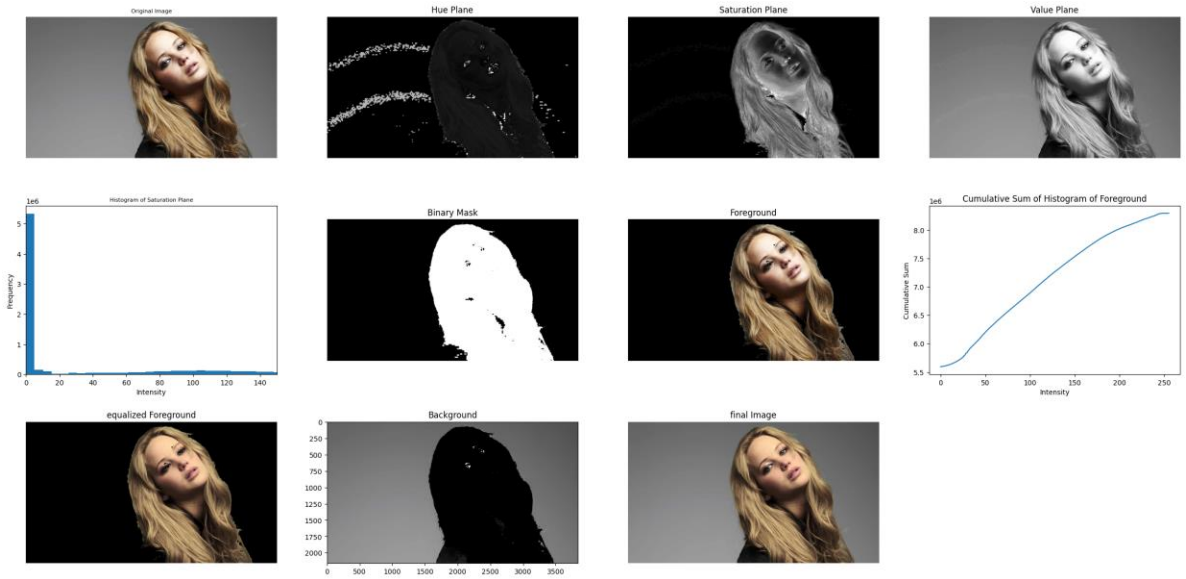


Question 5



I have use the formular $\frac{L-1}{MN}\sum_{j=0}^{k} n_j \quad where\ k = 0,1,\dots(L-1)$

Question 6

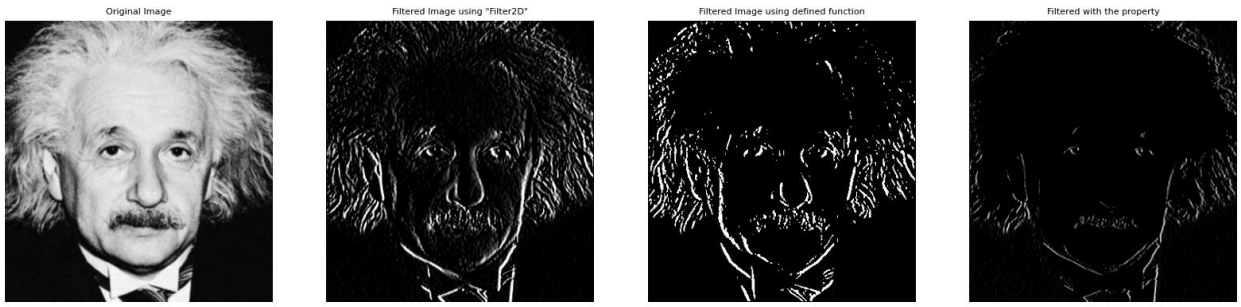


The process I have followed according to the question given above (left to right top to bottom.)

The most critical parts of the code collected together like this.

```
# creating binary mask using the threshold value
threshold = 15
mask = np.zeros(s.shape)
mask[s>threshold] = 255
#bitwise operation
foreground = cv.bitwise_and(img,mask_cvt_bgr)
#histogram equalization of the saturation plane
fg_s_equalized = my_histogram_equalization(fg_s,bins=2)
#merge
fg_hsv_equalized = cv.merge([fg_h,fg_s_equalized,fg_v])
#background mask invert foreground mask
bg_mask = 255 - mask
#extract the background
background = cv.bitwise_and(img,cv.cvtColor(bg_mask.astype(np.uint8), cv.COLOR_GRAY2BGR))
```

Question 7




```
def filter(image, kernel):
    assert kernel.shape[0] % 2 == 1 and kernel.shape[1] % 2 == 1
    k_hh, k_hw = math.floor(kernel.shape[0] / 2), math.floor(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(image.shape, '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())
    return result
```

Question 8

```python
def nearest_neighbor_zoom(image,scaling_factor):
    height, width , channels = image.shape

    new_height = int(height*scaling_factor)
    new_width = int (width * scaling_factor)

    new_image = np.zeros((new_height,new_width,channels),dtype=image.dtype)

    for y in range(new_height):
        for x  in range(new_width):
            u = int (x/scaling_factor)
            v = int (y/scaling_factor)
            new_image[y,x,:] = image[v,u,:]

    return new_image
```

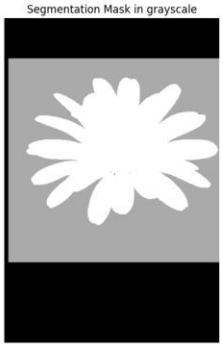
```python
def SSD_of_an_image(image1,image2):
    print(image1.shape,image2.shape)
    assert image1.shape == image2.shape
    height, width , channels = image1.shape
    ssd = np.int64(0)
    for y in range(height):
        for x in range(width):
            for channel in range(channels):
                ssd += (image1[y,x,channel] - image2[y,x,channel])**2
        #print(ssd)

    return ssd/(height*width*channels)
```

```python
def zoomed_image_bilinear(image,scaling_factor):
    height, width , channels = image.shape
    new_height = int(height*scaling_factor)
    new_width = int (width * scaling_factor)

    new_image = np.zeros((new_height,new_width,channels),dtype=image.dtype)

    for y in range(new_height):
        for x in range(new_width):
            u = x/scaling_factor
            v = y/scaling_factor
            u1,u2 = math.floor(u),math.ceil(u)
            v1,v2 = math.floor(v),math.ceil(v)

            if u2 >= width:
                u2 = width - 1
            if v2 >= height:
                v2 = height - 1
            A = image[v1,u1,:]
            B = image[v1,u2,:]
            C = image[v2,u1,:]
            D = image[v2,u2,:]
            # Calculate bilinearly interpolated pixel values for each channel
            for channel in range(channels):
                new_image[y, x, channel] = (
                    (u2 - u) * (v2 - v) * A[channel] +
                    (u - u1) * (v2 - v) * B[channel] +
                    (u2 - u) * (v - v1) * C[channel] +
                    (u - u1) * (v - v1) * D[channel]
                )

    return new_image
```

```
Image 01 - Nearest Neighbor SSD: 29376.036127186213, Bilinear SSD: 20229.320792984825
Image 02 - Nearest Neighbor SSD: 28443.24245775463, Bilinear SSD: 15977.82180367477
Image 04 - Nearest Neighbor SSD: 26317.328341933513, Bilinear SSD: 30529.760674511315
Image 05 - Nearest Neighbor SSD: 29667.387542390046, Bilinear SSD: 26223.14325998264
Image 06 - Nearest Neighbor SSD: 28136.714284014917, Bilinear SSD: 24797.95592319316
```

It is obvious that the SSD of nearest neighbor zooming is fairly high compared to bilinear zooming. i.e.the quality of the image is high when we use bilinear zooming.

Question 9



Reason for background and just beyond the edge of the flower quite dark in the enhanced image.

The flower's edge became sharper due to blur and contrast changes when we apply gaussian filters. This made the adjacent background appear darker as the enhanced flower's colors contrasted with the background's slight dimming caused by the blur.