Assignment 1 – Intensity Transformations and Neighborhood Filtering
Index – 200117T
Name – Deshan K.A.G.D.
GitHub –https://github.com/Gishandamindu/ImaPro_Ass1

1. The given image's intensity is to be changed in accordance with the intensity transformation diagram. Make a lookup table first, then map the provided image to it. To do that, I employed the function below.

```
def piecewise_linear_transform(image, breakpoints, slopes):
    lookup_table = np.zeros(256, dtype(np.uint8))
    for i in range(len(breakpoints) - 1):
        start, end = breakpoints[i], breakpoints[i+1]
        slope = slopes[i]
        lookup_table[start:end] = np.clip(np.arange(start, end) *
slope, 0, 255)
        transformed_image = cv.LUT (image, lookup_table)
    return transformed_image
```

code segment for question 1

This code enhances intensity values in (50-150) range to (100-255) range.

Intensity transformation Input Output

Figure 1

2.This question is asking to identify gray matters and white matters of brain.

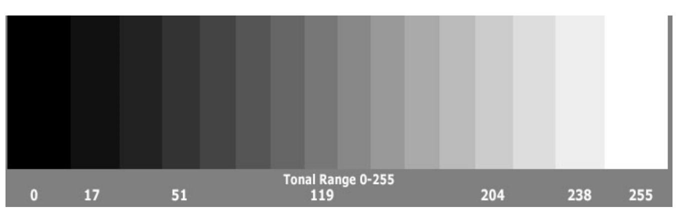
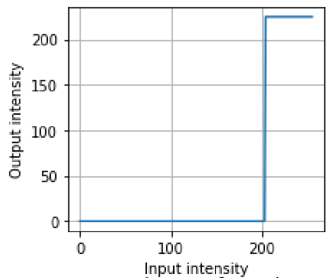


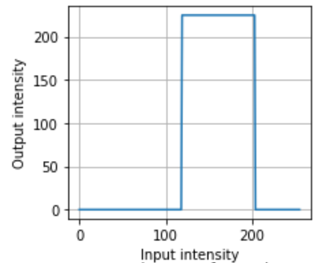
Figure 2 gray scale

Based on the grayscale image provided, we've identified two distinct intensity ranges: gray (119-204) and white (204-255). In the gray matter lookup table, values within the 119-204 range are set to 255 (white), and others to 0 (black). In the white matter lookup table, values within the 204-255 range are set to 255, while the rest are 0. This code is like the one in question 01, with differing lookup tables.



Intensity transformation


White matter

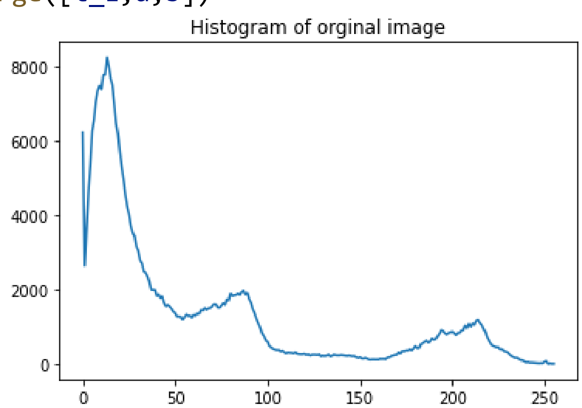


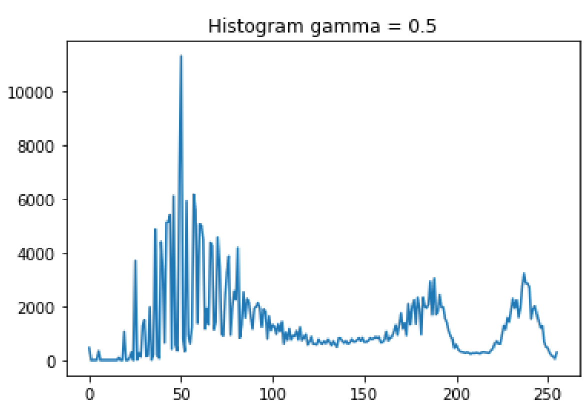
Intensity transformation

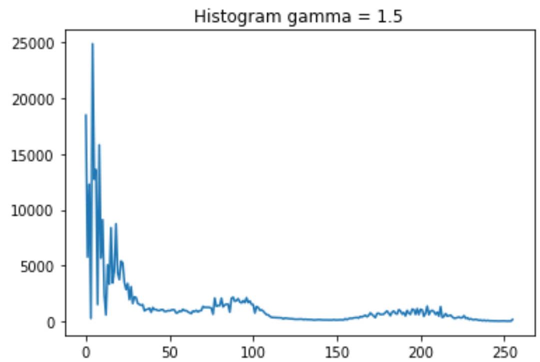

Gray matter

3. In this question, we're tasked with performing gamma correction on the L channel within the L*a*b color space. In Lab, L denotes the perception of brightness, while a and b signify colors like red, green, blue, and yellow. The code begins by converting the image to Lab color space and then proceeds with gamma correction.

```
#convert image to Lab color space
img = cv.cvtColor(img,cv.COLOR_BGR2LAB)
#add gamma correction
for gamma_val in gammaList:
    t_L = np.array([(p/255.0)**gamma_val*255 for p in L]).astype(np.uint8)
    img2= cv.merge([t_L,a,b])
```







The image is brighter for smaller gamma values (<1) and darker for bigger gamma values (>1). Moreover, by looking at histogram distributions, we can draw this conclusion.

4. Enhancing Photo Vibrance with Intensity Transformation for Stunning Results

(a) Split into HSV planes

```
hue_plane, saturation_plane, value_plane = cv.split(hsv_image)
```

(b) Apply the intensity transformation to the saturation plane

```
def apply_intensity_transformation(x, a, sigma=70):
    transformed_x = np.minimum(x + a * 128 * np.exp(-(x - 128) * 2 / (2 * sigma * 2)),
255).astype('uint8')
    return transformed_x
```

(c)select alpha value.

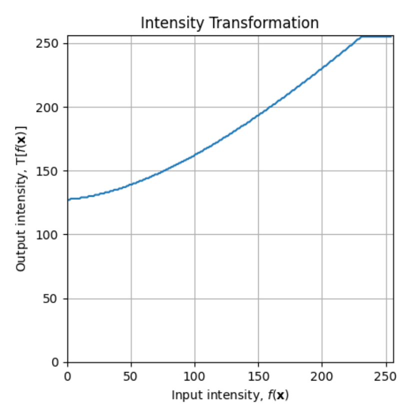
```
a_value = 0.4
transformed_saturation_plane = apply_intensity_transformation(saturation_plane, a_value)
```

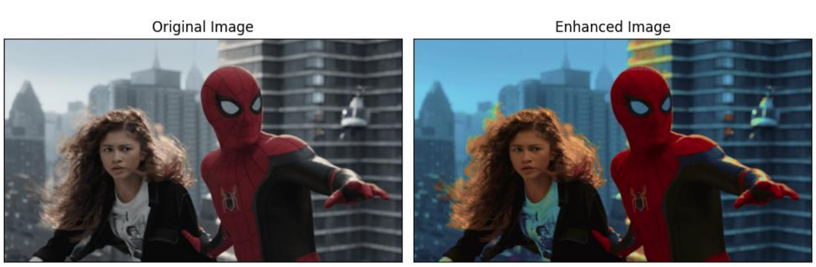
(d) Recombine the three planes

```
enhanced_hsv_image = cv.merge([hue_plane, transformed_saturation_plane, value_plane])
# Convert back to BGR for visualization
enhanced_image = cv.cvtColor(enhanced_hsv_image, cv.COLOR_HSV2BGR)
```

(e) Display the images and intensity transformation

```
plt.figure(figsize=(10, 5))
```

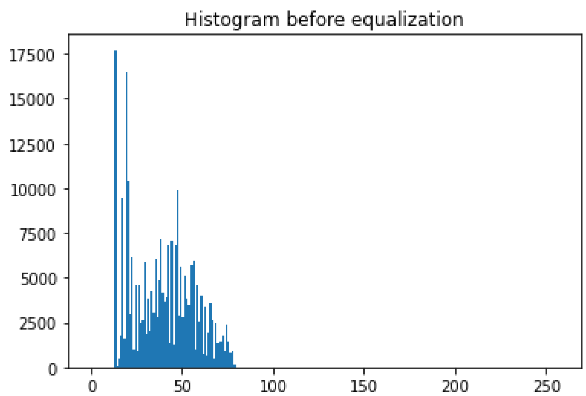


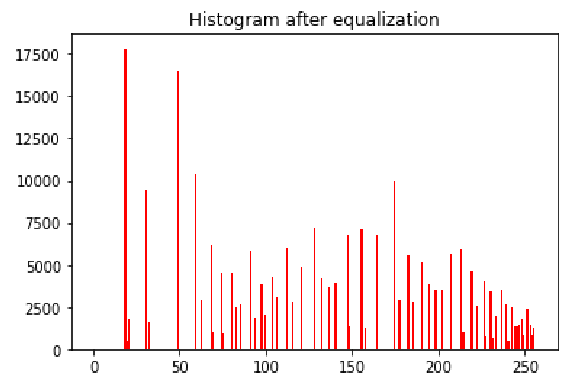


5.This question requests the creation of a function to perform histogram equalization. To achieve this, we can utilize the provided equation to implement the equalization process.

$$S_k = \frac{(L-1)}{Mn}\sum_{j=0}^{j=k} n_j \qquad k = 0,1,\ldots.. L-1$$

```
def Equalization(img):
        L_val=256
        hist,bins = np.histogram(img.ravel(),256,[0,256])
        cdf=hist.cumsum() #calculate cdf
        eqcHist = np.round((L-1)*cdf/cdf.max()) #normalized between 0-255
        return cv.LUT(img,eqcHist) #mapping
```





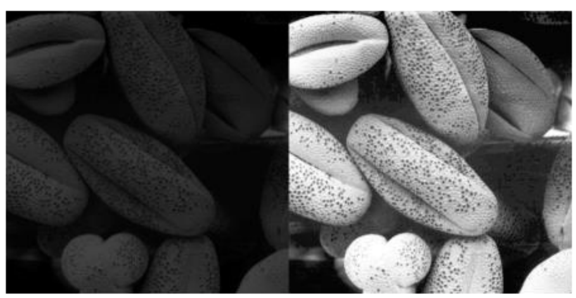


Image Before and after equalization

6.

(a) split it into hue, saturation, and values and display these planes in grayscale.

```
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
hue, saturation, value = cv.split(hsv_image)
```

(b) selected Value plane and made a mask as follows

```
_, foreground_mask = cv.threshold(value, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
```

(c) obtaining foreground using bitwise and

```
foreground = cv.bitwise_and(value, value,   mask=foreground_mask)
```

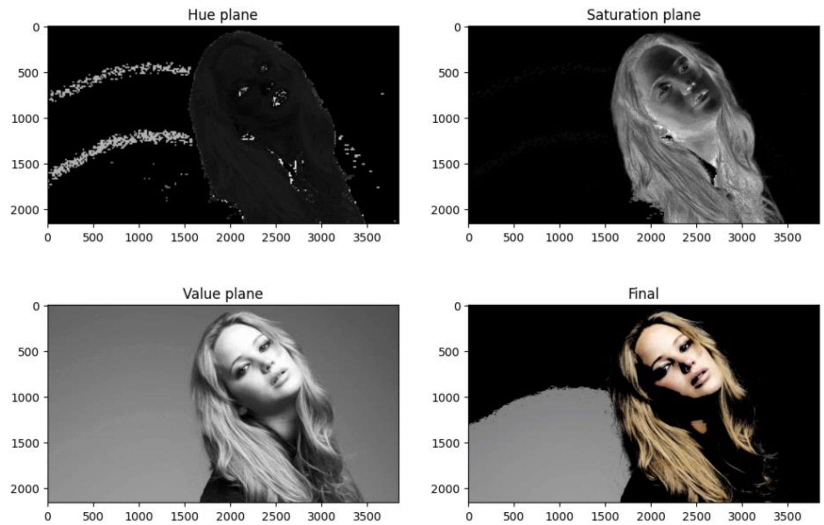
(d), (e), following function is used to equalize the foreground,

```
def histogram_equalization(image):
        hist, bins = np.histogram(image.flatten(), bins=256, range=[0, 256])
        cdf = hist.cumsum() cdf_normalized = ((cdf - cdf.min()) * 255) / (cdf.max() - cdf.min())
        equalized_image = np.interp(image.flatten(), bins[:-1], cdf_normalized)
```

```
return equalized_image.reshape(image.shape).astype(np.uint8) # Convert to uint8
```

(f) adding foreground and background again,

```
enhanced_value = cv.add(background, foreground)
hsv_image_modified = cv.merge((hue, saturation, enhanced_value))
result_image = cv.cvtColor(hsv_image_modified, cv.COLOR_HSV2RGB)
```



7.

(a)Sobel filter using cv. filter2D function

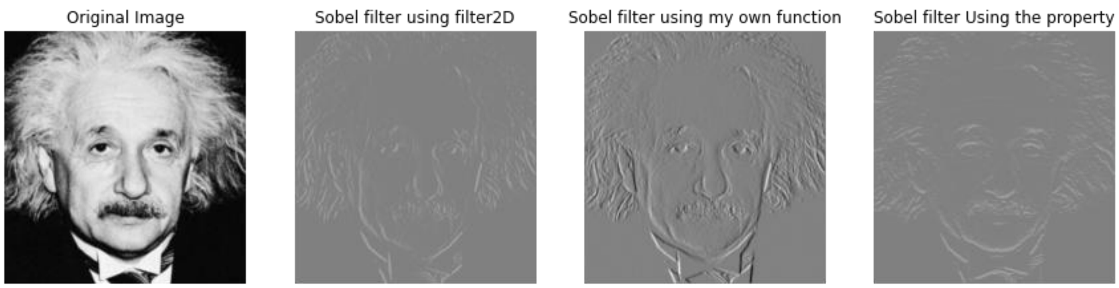
```
sobel_v = np.array([(-1,0,1),(-2,0,2),(-1,0,1)])
image_sobel_v = cv.filter2D(img,-1,sobel_v)
```

(b)Sobel filter using my own function.

```
def sobel_Filter(img):
    sobel_v = np.array([(-1,0,1),(-2,0,2),(-1,0,1)])#sobel vertical kernal
    rows,columns=img.shape
    r_lim,c_lim = sobel_v.shape[0]//2,sobel_v.shape[1]//2 #find limts
    filteredImg = np.zeros(img.shape) #create empty array
    #converlution
    for r in range(r_lim,rows-r_lim):
        for c in range(c_lim,columns-c_lim):
            filteredImg[r][c] = np.dot(img[r-r_lim:r+r_lim+1,c-
c_lim:c+c_lim+1].flatten(),sobel_v.flatten())
    return filteredImg
```

(c)Sobel filter using the property.

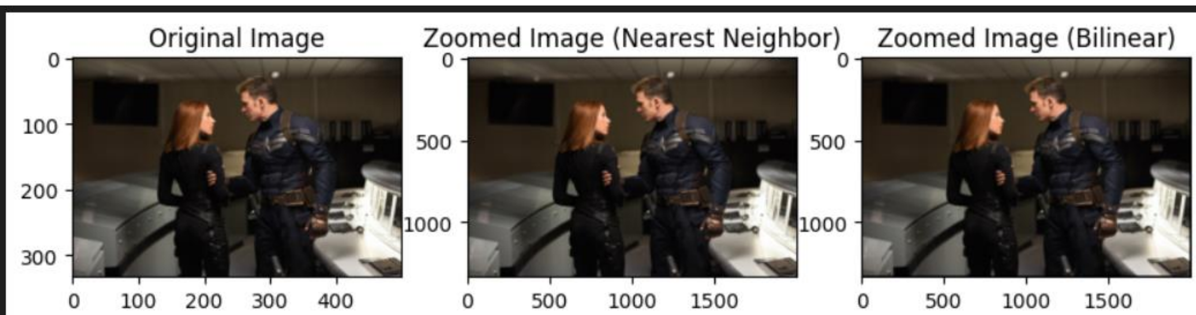
```
kernel1=np.array([[1],[2],[1]])
kernel2=np.array([1,0,-1])
img_s = cv.sepFilter2D(img,-1,kernel1,kernel2)
```



8. First we must implement two functions using the Nearest-Neighbor zooming method and bilinear interpolation method.

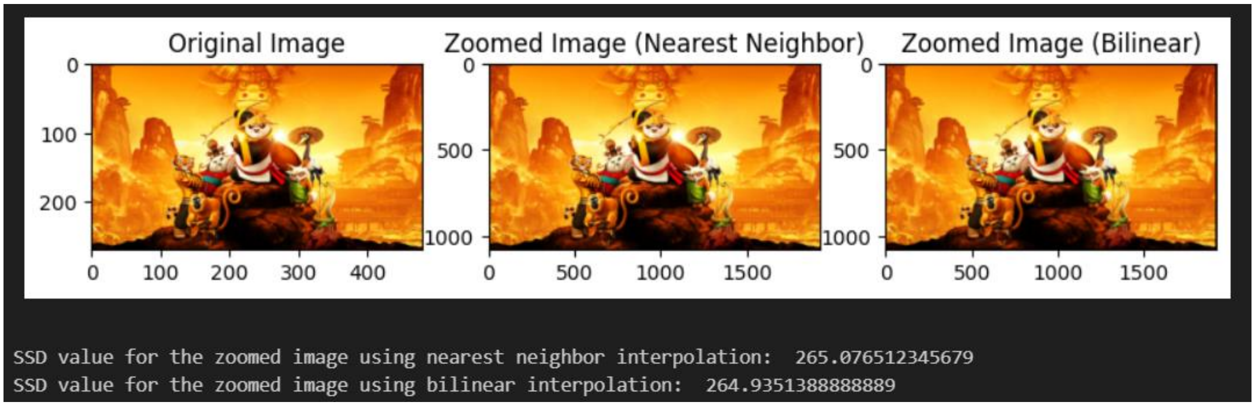
```python
# Zoom in using nearest neighbor interpolation
def zoom_nearest_neighbor(image, factor):
    zoomed_image = np.zeros((int(factor * len(image)), int(factor * len(image[0])), 3), dtype=np.uint8)
    for i in range(len(zoomed_image)):
        for j in range(len(zoomed_image[0])):
            zoomed_image[i, j] = image[int(i / factor), int(j / factor)]
    return zoomed_image

# Define a function to zoom the image with bilinear interpolation
def zoom_bilinear(image, factor):
    zoomed_image = np.zeros((int(factor * len(image)), int(factor * len(image[0])), 3), dtype=np.uint8)
    for i in range(len(zoomed_image)):
        for j in range(len(zoomed_image[0])):
            x = i / factor
            y = j / factor
            x1 = int(x)
            y1 = int(y)
            x2 = min(x1 + 1, len(image) - 1)
            y2 = min(y1 + 1, len(image[0]) - 1)
            zoomed_image[i, j] = (image[x1, y1] * (x2 - x) * (y2 - y) + image[x1, y2] * (x2 - x) * (y - y1) +
                                  image[x2, y1] * (x - x1) * (y2 - y) + image[x2, y2] * (x - x1) * (y - y1)).astype(np.uint8)
    return zoomed_image
```

Using two images,
Results_1




```
SSD value for the zoomed image using nearest neighbor interpolation:  285.5827327327327
SSD value for the zoomed image using bilinear interpolation:  286.3897597597598
```

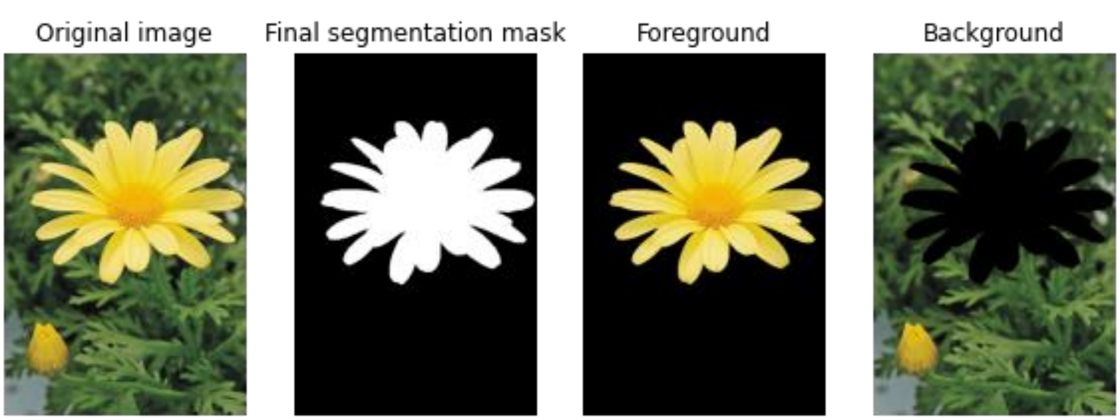
Results_2,



The results obtained from the nearest neighbor technique yield higher values compared to those from the bilinear method. This implies that when striving for improved and more precise image quality, the bilinear method is a superior option compared to the nearest neighbor approach. The SSD metric offers insights into the proximity between our zoomed image and the original one.

9.This question is asking to use grab Cut to segment the image.

```
mask1= np.zeros(img.shape[:2],np.uint8) #create empty mask with same size of image
rect =(55,145,555,490) #coordinate of rectangle which cotain flower
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
mask1,bgdModel,fgdModel = cv.grabCut(img,mask1,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask1==2)|(mask1==0),0,1).astype('uint8')#set 1 to places where flower exist
final_mask = (mask2 * 255).astype(np.uint8)#create segmentation mask
foreground_img = img*mask2[:,:,np.newaxis] #isolate foreground
background_img = img - foreground_img       #isolate background
```



We can add foreground to blurred background to get enhanced image.

```
blurred_Background = cv.blur(background_img,(9,9)) #blue isolated
background
enhanced_image = foreground_img+blurred_Background #add blurred background
and islated foreground
```

The background just beyond the edge of the flower is quite dark in the enhanced image because when blurring the background, some pixel values near the border of the flower convert to value 0.