EN3160 - Image Processing and Machine Vision
Assignment 01
Intensity Transformations and Neighborhood Filtering

September 1, 2023

Name: Wijetunga W.L.N.K Index Number: 200733D

1 Answer for Question 1

The following intensity transformations were applied to the original image.

```
c = np.array([(50,50),(50,100),(150,255),(150,150)])

t1 = np.linspace(0, c[0,1], c[0,0] ).astype('uint8')
t2 = np.linspace(c[1,1] + 1, c[2,1] ,c[2,0] - c[1,0] ).astype('uint8')
t3 = np.linspace(c[3,1] + 1, 255 , 255 - c[3,0] + 1 ).astype('uint8')
```

Listing 1: Intensity Transformations - Question 1

The results before and after applying the linear transformation is as follows.

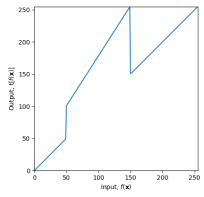
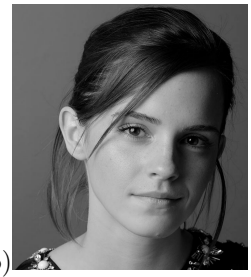
(a)  (b)  (c) 

Figure 1: (a) The Transformation (b) Original Image (c) Image After Transformation

2 Answer for Question 2

2 separate intensity transformations were applied to the image of the brain and from the 1st transformation, grey matter was accentuated and the 2nd transformation accentuated the white matter.

```
#Linear transformation fuctions for the image enhancement 1
t1 = np.linspace(0, 10, 140 ).astype('uint8')
t2 = np.linspace(11, 230 ,90 ).astype('uint8')
t3 = np.linspace(231, 255 , 26).astype('uint8')

#Linear transformation fuctions for the image enhancement 2
ta = np.linspace(0, 1, 120 ).astype('uint8')
tb = np.linspace(11, 235 ,25 ).astype('uint8')
td = np.linspace(236, 200 , 27).astype('uint8')
tc = np.linspace(199, 0 , 9).astype('uint8')
te = np.linspace(0, 1 , 75).astype('uint8')
```

Listing 2: Intensity Transformations - Question 2

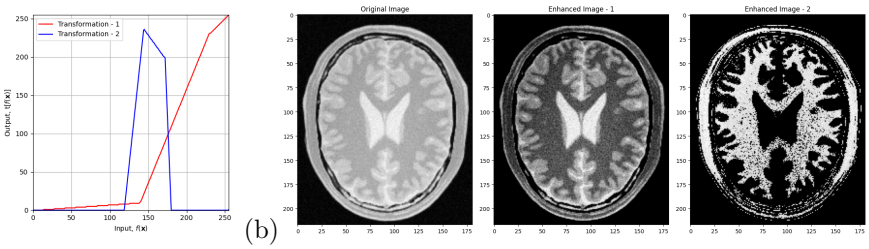


Figure 2: (a)Transformation Plots (b) Original and After transformation

3 Answer for Question 3

3.1 Part (a)

Gamma correction was applied only to the L plane of the L*a*b colour space of the image and the used value for gamma was **1.4** for the gamma correction.

```
# Convert to CIELAB color space
img_lab = cv.cvtColor(img_orig, cv.COLOR_BGR2Lab)

# Define the gamma value
gamma = 1.4

# Apply gamma correction to L plane
plane = img_lab[:, :, 0]
corrected_plane = np.power(plane / 255.0, gamma) * 255.0
img_lab[:, :, 0] = corrected_plane.astype(np.uint8)
```

Listing 3: Gamma Correction - Question 3

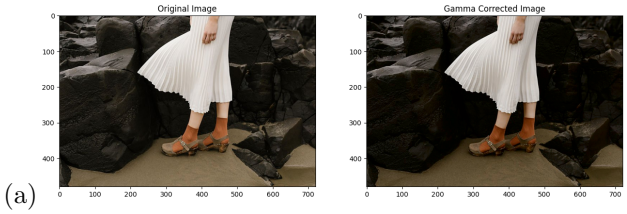


Figure 3: (a) The Transformation (b) Original Image (c) Image After Transformation

3.2 Part (b)

The histogram of the original image and the gamma corrected image is as follows.

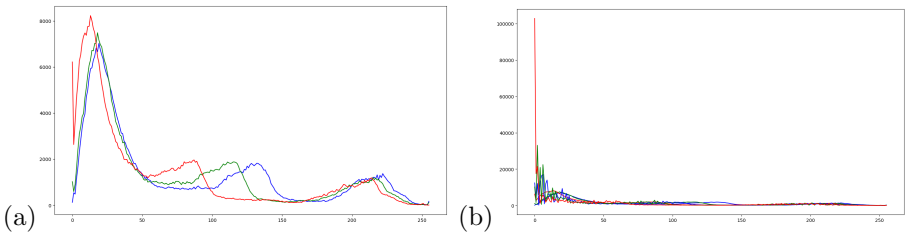


Figure 4: Histogram (a)Before Gamma Correction (b) After Gamma Correction

4 Answer for Question 4

4.1 Part (a)

```
# Convert the image to the HSV color space
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

# Split the image into hue, saturation, and value planes
hue_plane = hsv_image[:,:,0]
saturation_plane = hsv_image[:,:,1]
value_plane = hsv_image[:,:,2]
```

Listing 4: Split the image into HSV planes - Question 4

4.2 Part (b)

```
# Intensity transformation function
def intensity_transformation(x, a, sigma):
    return np.minimum(x + (a * 128) * np.exp(-(x - 128)**2 / (2 * sigma**2)), 255)
```

Listing 5: Apply the transformation to the saturation plane - Question 4

4.3 Part (c)

When the value of 'a' changes, the intensity transformation applied to the saturation plane changes. The value of 'a' was changed in the interval [0,1] and a vibrance-enhanced image could be obtained when **when a = 0.4.**

4.4 Part (d)

```
# Create the enhanced HSV image
enhanced_hsv_image = np.stack([hue_plane, enhanced_saturation_plane, value_plane],
    axis=-1)
```

Listing 6: Combining the Planes - Question 4

4.5 Part (e)

Original image , enhanced image and the intensity transformation applied to the original image is as follows.

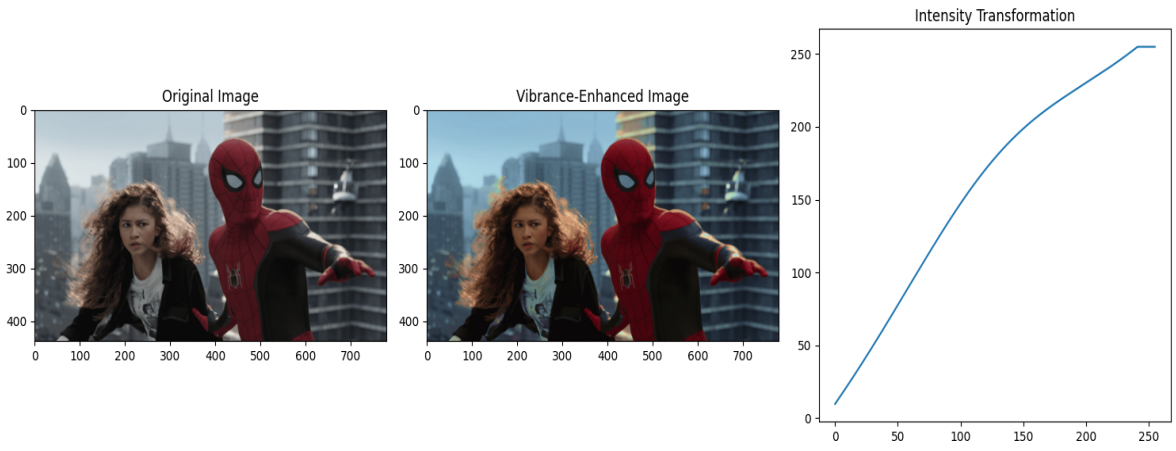


Figure 5: Image Comparison and the Intensity Transformation

5 Answer for Question 5

The custom function created for histogram equalization first get image and check whether it is a gray-scale image. Then the histogram of the image is obtained using **np.histogram** and the range of pixels is set to [0,256). Then the cumulative sum of the histogram values are obtained and it is normalized. The histogram equalization process is performed using the normalized CDF. For each pixel value in the gray-scale image, the code uses **np.interp** to map the pixel value to its corresponding value in the CDF. This essentially stretches the range of pixel values to achieve a more uniform distribution. The equalized image is flattened, processed using the interpolation, and then reshaped back to the original gray-scale image shape.

```
1
2  def histogram_equalization(image):
3      # Convert the image to grayscale if it's not already
4      if len(image.shape) == 3:
5          gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
6      else:
7          gray_image = image.copy()
8
9      # Calculate the histogram of the grayscale image
10     hist, bins = np.histogram(gray_image.flatten(), bins=256, range=[0,256])
11
12     # Calculate the cumulative distribution function (CDF)
13     cdf = hist.cumsum()
14     cdf_normalized = cdf * hist.max() / cdf.max()
15
16     # Perform histogram equalization
17     equalized_image = np.interp(gray_image.flatten(), bins[:-1], cdf_normalized)
18     equalized_image = equalized_image.reshape(gray_image.shape)
19
20     return equalized_image
```

Listing 7: Function for Histogram Equalization - Question 5

The histograms of the original and the equalized image are as follows.

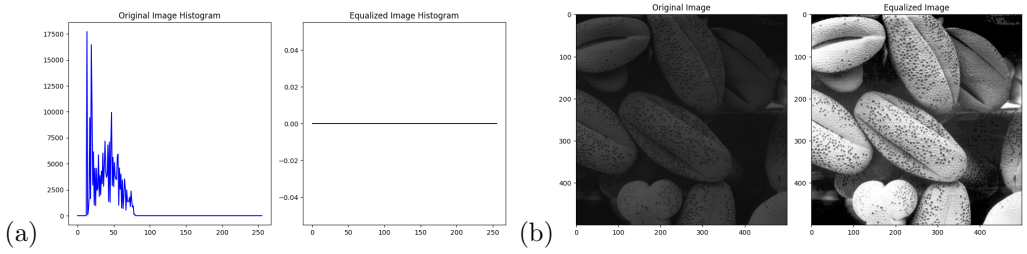


Figure 6: (a) The Transformation (b) Original Image (c) Image After Transformation

6 Answer for Question 6

6.1 Part (a) and Part (b)

```
1  # Split the image into hue, saturation, and value planes
2  hue_plane = hsv_image[:,:,0]
3  saturation_plane = hsv_image[:,:,1]
4  value_plane = hsv_image[:,:,2]
5
6  threshold_value = 75
7
8  _,binary_mask_1 = cv.threshold(hue_plane, threshold_value, 255, cv.THRESH_BINARY)
9  _,binary_mask_2 = cv.threshold(saturation_plane, threshold_value, 255, cv.
       THRESH_BINARY)
10 _,binary_mask_3 = cv.threshold(value_plane, threshold_value, 255, cv.THRESH_BINARY)
```

Listing 8: Code for Part(a) and (b)- Question 6

For part(a),the image was split it into hue, saturation, and values and displayed these planes in grayscale. For part(b), to select the appropriate plane to extract the foreground mask, the foreground mask was extracted from all the planes and it was then decided to use **the value plane** to extract the foreground mask.

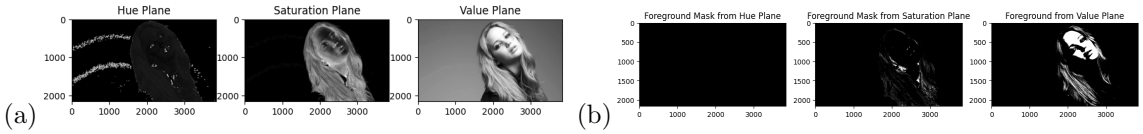


Figure 7: (a)Planes - Part(a) (b)Foreground Extracted - Part(b)

6.2 Part (c) and Part (d)

```
# Apply bitwise_and to extract the foreground pixels
foreground = cv.bitwise_and(image, image, mask=binary_mask_3)

# Compute the histogram of the foreground image
histogram = cv.calcHist([foreground], [0], None, [256], [0, 256])

# Compute the cumulative histogram
cumulative_histogram = np.cumsum(histogram)
```

Listing 9: Code for Part(c) and (d) - Question 6

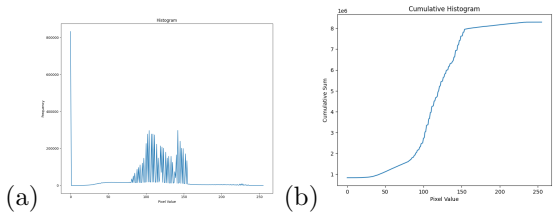


Figure 8: (a)Histogram (b)Cumulative Histogram

6.3 Part (e) and Part (f)

After performing histogram equalization for the foreground, the background was extracted and added with the histogram equalized foreground. The results after performing the above are as follows.

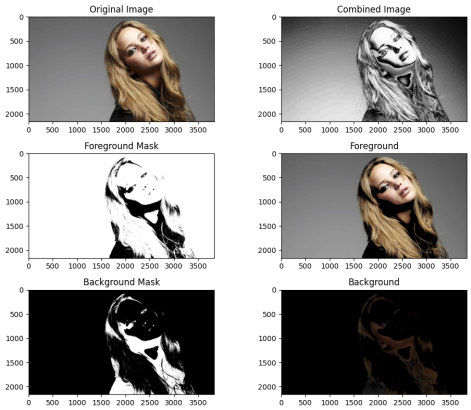


Figure 9: The Results

7 Answer for Question 7

7.1 Part (a)

```
#Kernel For Verticle edge detection
kernel_v = np . array([(-1,-2,-1),(0,0,0),(1,2,1)], dtype = 'float')
#Kernel For Horizontal edge detection
kernel_h = np . array([(-1,0,1),(-2,0,2),(-1,0,1)], dtype = 'float')
```

Listing 10: Using Filter2D - Question 7

7.2 Part (b)

```
# Convolution operation for gradient computation
def convolve(image, kernel):
    kernel_size = kernel.shape[0]
    print(kernel_size)
    image_height, image_width = image.shape

    # Initialize the result image
    result = np.zeros((image_height - kernel_size + 1, image_width - kernel_size + 1))

    # Perform convolution
    for i in range(result.shape[0]):
        for j in range(result.shape[1]):
            result[i, j] = np.sum(image[i:i+kernel_size, j:j+kernel_size] * kernel)

    return result
```

Listing 11: Without Filter2D - Question 7

7.3 Part (c)

```
kernel_1 = np.array([(1,2,1)], dtype = 'float')
kernel_2 = np.array([(-1),(0),(1)], dtype = 'float')
```

Listing 12: Using Convolution Property of the Kernel - Question 7

Comparison of the results.

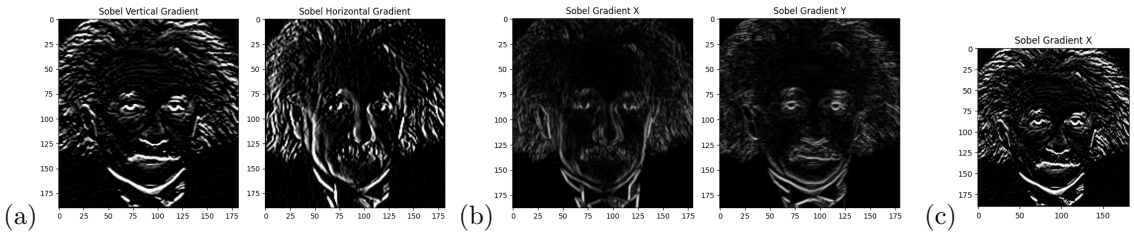


Figure 10: Histogram (a)Using filter2D (b)Without filter2D (c)Convolution Property

8 Answer for Question 8

Two separate functions were created to handle nearest-neighbor and bilinear interpolation when zooming images.

```
def zoom_nearest_neighbor(image, zoom_factor):

    new_height = int(image.shape[0] * zoom_factor)
    new_width = int(image.shape[1] * zoom_factor)

    # Create the zoomed image with the new dimensions
```

```
    zoomed_image = np.zeros((new_height, new_width, 3), dtype=np.uint8)


    # Fill in the values of the zoomed image
    for i in range(new_height):
        for j in range(new_width):
            # Find the nearest neighbor for each pixel
            y = int(i / zoom_factor)
            x = int(j / zoom_factor)

            # Assign the nearest neighbor to the zoomed image
            zoomed_image[i, j] = image[y, x]

    return zoomed_image

def zoom_bilinear_interpolation(image, zoom_factor):
    new_height = int(image.shape[0] * zoom_factor)
    new_width = int(image.shape[1] * zoom_factor)

    # Create the zoomed image with the new dimensions
    zoomed_image = np.zeros((new_height, new_width, 3), dtype=np.uint8)

    # Fill in the values of the zoomed image
    for i in range(new_height):
        for j in range(new_width):
            y = i / zoom_factor
            x = j / zoom_factor

            # Find the nearest neighbors of the pixel
            y_low = int(np.floor(y))
            y_high = min(y_low + 1, image.shape[0] - 1)
            x_low = int(np.floor(x))
            x_high = min(x_low + 1, image.shape[1] - 1)

            # Calculate the interpolation weights
            dy = y - y_low
            dx = x - x_low

            # Interpolate the pixel value
            for c in range(3):
                interpolated_value = (1 - dx) * (1 - dy) * image[y_low, x_low, c] + \
                                     dx * (1 - dy) * image[y_low, x_high, c] + \
                                     (1 - dx) * dy * image[y_high, x_low, c] + \
                                     dx * dy * image[y_high, x_high, c]
                zoomed_image[i, j, c] = int(interpolated_value)

    return zoomed_image
```

Listing 13: Function for Handling Nearest-neighbor and Bilinear Interpolation Zooming - Question 8

After performing the zoom function, the zoomed images were compared with the scaled image provided. The normalized sum of squared difference (SSD) is a common technique used to evaluate the quality of image processing algorithms. It was observed that the normalized sum of squared difference of nearest neighbour method is less than the the normalized sum of squared difference of bilinear interpolated image indicating nearest neighbour image scaling method preserves the original pixel values.

9 Answer for Question 9

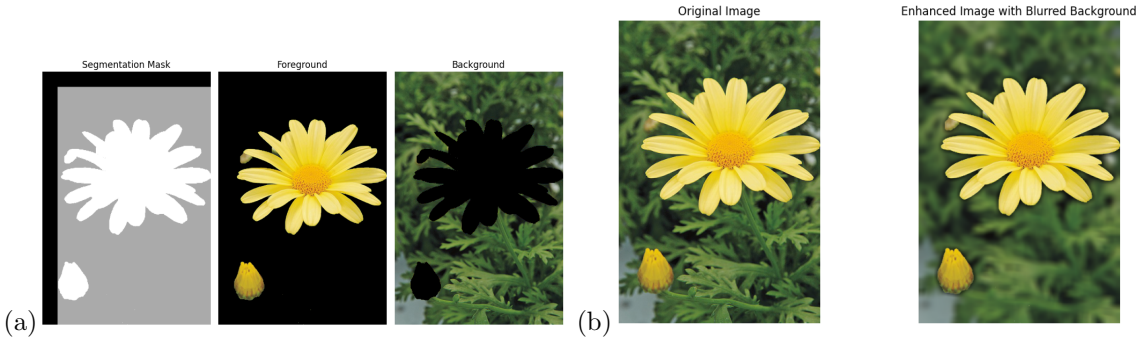
9.1 Part (a) and Part (b)

```
# Create a mask for the foreground and background
mask = np.zeros(image.shape[:2], np.uint8)

# Define rectangle for initial segmentation
rect = (50, 50, 40000, 8000)

# Initialize the background and foreground models
```

```
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)

# Apply GrabCut algorithm
cv.grabCut(image, mask, rect, bgdModel, fgdModel, 5, cv.GC_INIT_WITH_RECT)

# Modify the mask to separate foreground and background
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Create segmented images
foreground = image * mask2[:, :, np.newaxis]
background = image * (1 - mask2[:, :, np.newaxis])

# Create a copy of the original image for the enhanced image
enhanced_image = image.copy()

# Apply Gaussian blur to the background
blurred_background = cv.GaussianBlur(background, (21, 21), sigmaX=30)

# Replace the background of the enhanced image with the blurred background
enhanced_image[np.where(mask2 == 0)] = blurred_background[np.where(mask2 == 0)]
```

Listing 14: Code for Part(a) and (b) - Question 9

Final segmentation mask, foreground image, and background image and comparison of the images are as follows.



(a) (b)

Figure 11: (a)Final segmentation mask, Foreground Image and Background image (b)Comparison of the images

9.2 Part (c)

The background just beyond the edge of the flower is quite dark in the enhanced image, because the GrabCut algorithm has assigned those pixels to the background class during the segmentation process. This can happen due to several reasons.

- The initial estimation of the foreground model has not covered the edges of the flower correctly.
- The flower's edge has variations in color and texture.So the algorithm might have difficulty accurately separating the foreground from the background.
- Since GrabCut uses a Gaussian Mixture Model to represent the foreground and background, the interaction between these models can sometimes lead to incorrect assignments if the two models overlap in certain regions specially at edges.

Full Versions of the Codes and Results

All the full versions of the codes used along with more explanations in this assignment and results of question 8 can be found in the following GitHub repository. Click here to access the repository.