Department of Electronic and Telecommunication Engineering
University of Moratuwa

EN3160 - Image Processing and Machine Vision

Assignment 1

210490L Prabodha K.P.K.A.

Question 1

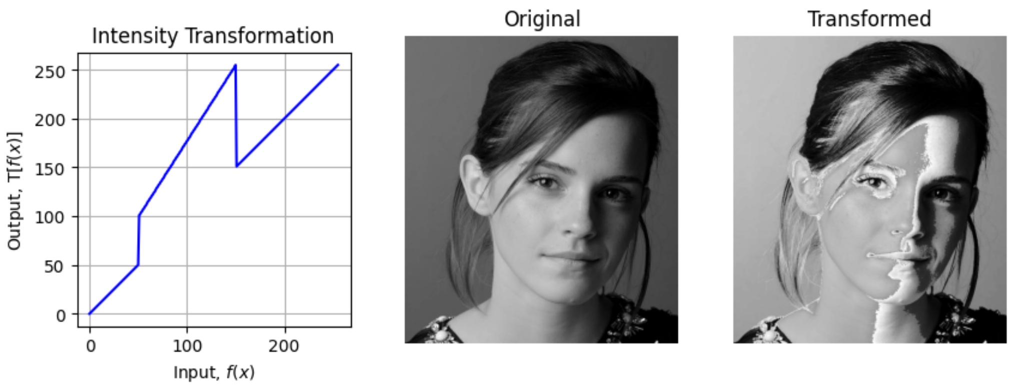

Figure 1: Intensity transformation of `emma.jpg` image

This graph shows the variation in pixel intensity within an image. The input intensity is represented on the x-axis, while the output intensity is shown on the y-axis. The graph's analysis says that for low and high-intensity values (near 0 and 250), the output intensity mirrors the input. In the mid-range intensity values (approximately 100 to 150), there is a significant increase in output intensity. This indicates that mid-tone pixels in the image will exhibit greater brightness, at the same time extremely dark and extremely bright pixels will remain unchanged.

```python
# Compute the piecewise linear intensity transformation based on the control points
t1 = np.linspace(0, c[0, 1], c[0, 0] + 1).astype('uint8')  # from 0 to c[0]
t2 = np.linspace(c[0, 1] + 1, c[1, 1], c[1, 0] - c[0, 0]).astype('uint8')  # from c[0] to c[1]
t3 = np.linspace(c[1, 1] + 1, c[2, 1], c[2, 0] - c[1, 0]).astype('uint8')  # from c[1] to c[2]
t4 = np.linspace(c[2, 1] + 1, c[3, 1], c[3, 0] - c[2, 0]).astype('uint8')  # from c[2] to c[3]
t5 = np.linspace(c[3, 1] + 1, c[4, 1], 255 - c[3, 0]).astype('uint8')  # from c[3] to c[4]
```

Question 2

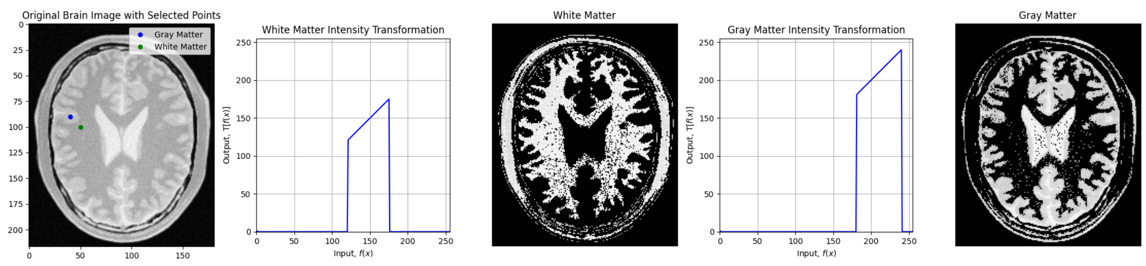
Original Brain Image with Selected Points
Gray Matter
White Matter
White Matter Intensity Transformation
Output, T[f(x)]
Input, f(x)
White Matter
Gray Matter Intensity Transformation
Output, T[f(x)]
Input, f(x)
Gray Matter

Figure 2: Transformations for `White` and `Gray` matters

```python
# Compute the piecewise linear intensity transformation based on the control points
t1 = np.linspace(0, c[0, 1], c[0, 0] + 1).astype('uint8')  # from 0 to c[0]
t2 = np.linspace(c[0, 1] + 1, c[1, 1], c[1, 0] - c[0, 0]).astype('uint8')  # from c[0] to c[1]
t3 = np.linspace(c[1, 1] + 1, c[2, 1], c[2, 0] - c[1, 0]).astype('uint8')  # from c[1] to c[2]
t4 = np.linspace(c[2, 1] + 1, c[3, 1], c[3, 0] - c[2, 0]).astype('uint8')  # from c[2] to c[3]
t5 = np.linspace(c[3, 1] + 1, c[4, 1], c[4, 0] - c[3, 0]).astype('uint8')  # from c[3] to c[4]
t6 = np.linspace(c[4, 1] + 1, c[5, 1], 255 - c[4, 0]).astype('uint8')  # from c[4] to c[5]
```

- In brain imaging, white matter, and gray matter represent two distinct areas that play crucial roles in various brain functions. White matter primarily comprises nerve fibers (axons) that facilitate communication between different brain regions, and it typically appears brighter in MRI scans due to its unique composition. Conversely, gray matter consists of the cell bodies of neurons and is generally observed as darker than white matter in MRI images.

- The graphs and images illustrate intensity transformations that have been applied to enhance the visibility of each type of matter. The initial transformation aims to highlight the white matter by increasing the brightness of the intensity values within the white matter range. Each transformation concentrates on the gray matter, modifying the intensity values to bring this region into sharper focus while maintaining the relative stability of other intensities. These transformations are instrumental in improving the visualization of distinct brain tissues by emphasizing the specific intensity ranges associated with white and gray matter.
- We can use **Gaussian pulse as the intensity transformation function** also.

Question 3

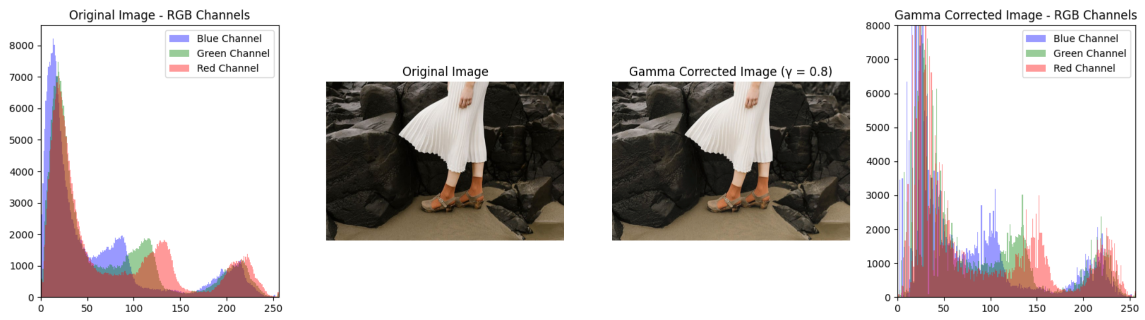

Figure 3: Original image vs Gamma corrected image

- In this process, gamma correction was implemented on the L^* (lightness) channel within the $L^*a^*b^*$ color space of the image to modify its brightness. A gamma value of **0.8** was selected, increasing the image's brightness. This correction alters pixel intensity in a non-linear manner, lightening the darker regions while having a lesser impact on the brighter areas.
- After the correction, a comparison was made between the original and the adjusted images, with their respective histograms being plotted. These histograms illustrate the distribution of pixel intensities across the Red, Green, and Blue channels for both the original and gamma-corrected images. The histogram of the corrected image exhibits a slight shift to the right, signifying an increase in pixel values relative to the original image.

```python
L_channel, a_channel, b_channel = cv.split(lab_img)
L_channel = L_channel / 255.0
L_corrected = np.array(255 * (L_channel ** gamma_value), dtype='uint8')
lab_corrected_img = cv.merge((L_corrected, a_channel, b_channel))
```

Question 4

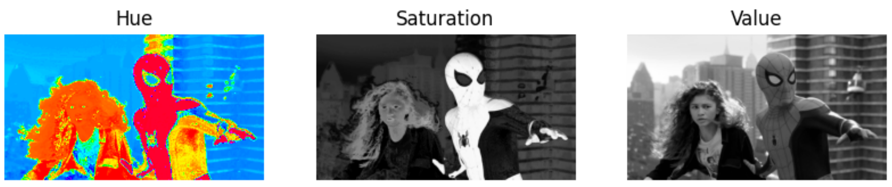

Figure 4: Image split into Hue, Saturation and Value channels

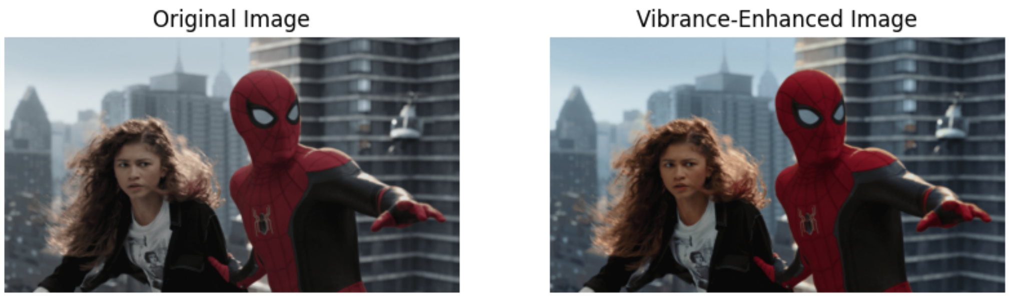

Figure 5: Vibrance adjustment

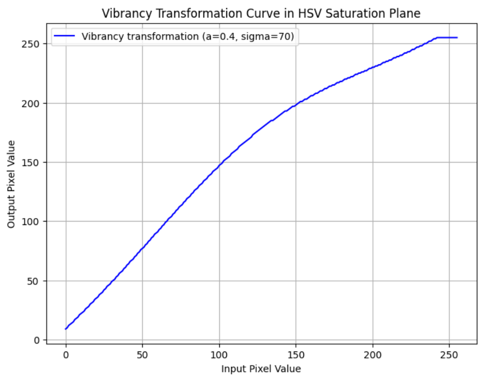

Figure 6: Transformation Curve

Figure 4 shows that the input image is split into Hue, Saturation, and Value channels separately. The equation,

$$f(x) = \min\left(x + \alpha \times 128e^{\frac{-(x-128)^2}{2\sigma^2}}, 255\right),$$

The input intensity is represented by x, with a value of α ranging from 0 to 1, and σ set at 70, which facilitated the intensity transformation illustrated in Figure 6. After careful adjustment, a value of $\alpha = 0.4$ was determined to be optimal. The resulting image can be observed in Figure 7. This transformation was exclusively applied to the Saturation plane, utilizing the code provided below.

The saturation plane exhibits elevated values in the vibrant regions of the image, such as the red in the spider suit. Conversely, the areas devoid of color, like the buildings and sky, display low pixel values. By enhancing solely the colorful sections of the image while maintaining the colorless areas unchanged, we effectively boost the vibrance.

Question 5

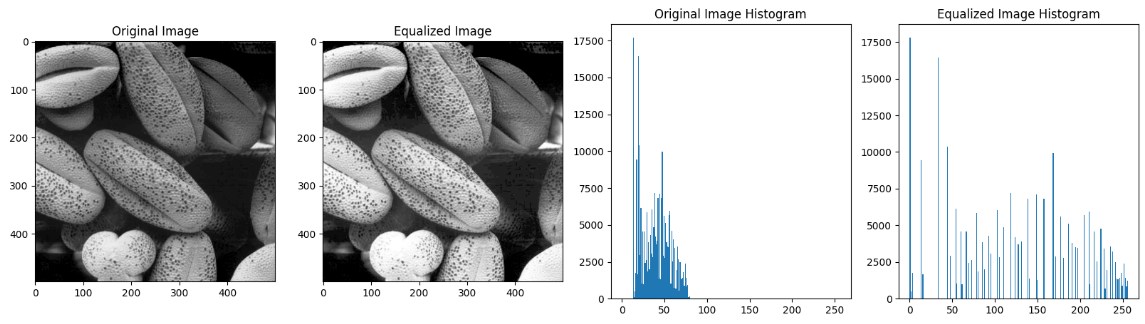

Figure 7: Equalized image with Histogram

Here we execute histogram equalization on a grayscale image. Histogram equalization is a method that is used to improve the contrast of an image by distributing the intensity values more uniformly throughout the available range. Additionally, histograms showing how the pixel brightness values are spread out before and after equalization are created, highlighting the improvement in contrast.

```python
# Custom function for histogram equalization
def custom_histogram_equalization(img):
    # Get the image histogram
    hist, bins = np.histogram(img.flatten(), 256, [0, 256])

    # Compute the cumulative distribution function (CDF)
    cdf = hist.cumsum()

    # Normalize the CDF
    cdf_normalized = cdf * hist.max() / cdf.max()
```

Question 6

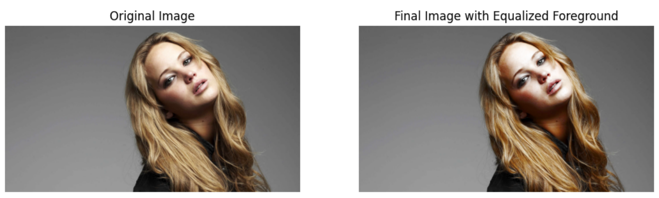

Figure 8: Image split into Hue, Saturation and Value channels

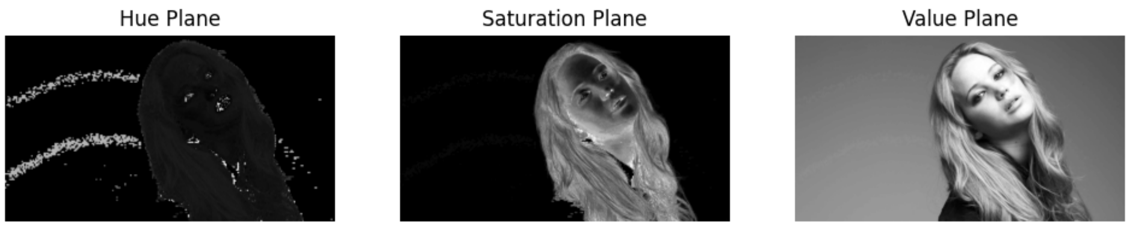

Figure 9: Image split into Hue, Saturation and Value channels

Figure 9 illustrates the decomposition of the image jennifer.jpg into its Hue, Saturation, and Value components. **The saturation component is particularly suitable for use as a mask.**

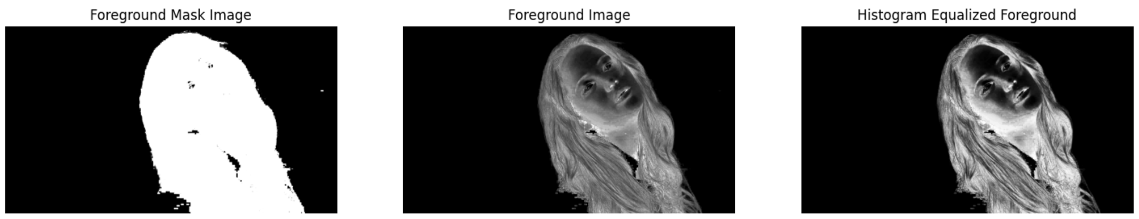

Figure 10: Foreground outputs

- The outcome following the brightness adjustment of the foreground is shown in Figure 8. This modification resulted in an increased number of areas appearing darker. This can be verified by examining the output image, where the shadows beneath the face are noticeably darker.
- There are other middle outputs of this process also shown in Figure 10.

```
# Threshold the value plane to extract the foreground
_, mask = cv2.threshold(value, 80, 255, cv2.THRESH_BINARY)
```

Question 7

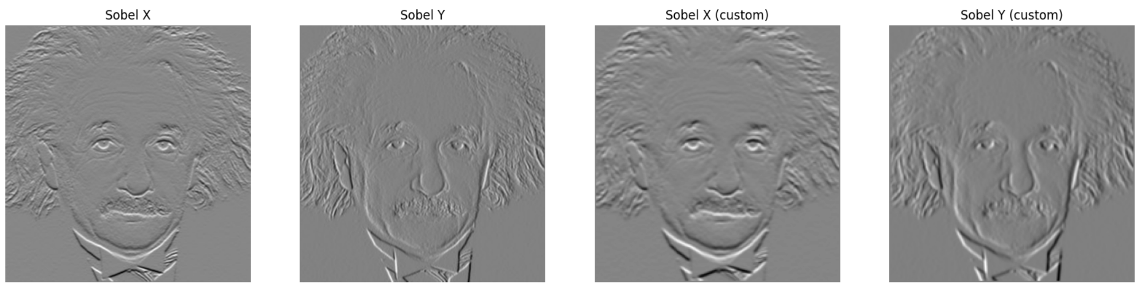

Figure 11: Filtered images using `cv2.filter2D` and Custom code

The first 2 images show the results that get using `cv2.filter2D` and the other 2 images are the outputs that get from custom code. The above images show that these 2 processes give nearly identical results.

```python
def sobel_filter(image, kernel):
    M, N = image.shape
    (k, k) = kernel.shape

    # Sobel filtering without padding
    filtered_image = np.zeros((M, N), dtype=np.float64)

    for i in range(1, M - k + 2):
        for j in range(1, N - k + 2):
            filtered_image[i, j] = np.sum(image[i - 1:i + 2, j - 1:j + 2] * kernel)

    return filtered_image
```

Question 8

```python
def zoom_image(image, zoom_factor, interpolation):
    M, N, c = image.shape
    new_M = int(M * zoom_factor)
    new_N = int(N * zoom_factor)

    if interpolation == cv2.INTER_NEAREST:
        zoomed_image = cv2.resize(image, (new_N, new_M), interpolation=cv2.INTER_NEAREST)

    elif interpolation == cv2.INTER_LINEAR:
        zoomed_image = cv2.resize(image, (new_N, new_M), interpolation=cv2.INTER_LINEAR)
```

The function provided below was employed to enlarge the images utilizing either nearest-neighbor or bilinear interpolation methods. For each image, the Sum of Squared Differences (SSD) was computed to assess the similarity between the scaled-up versions and the original images. The SSD values corresponding to each image are presented above.

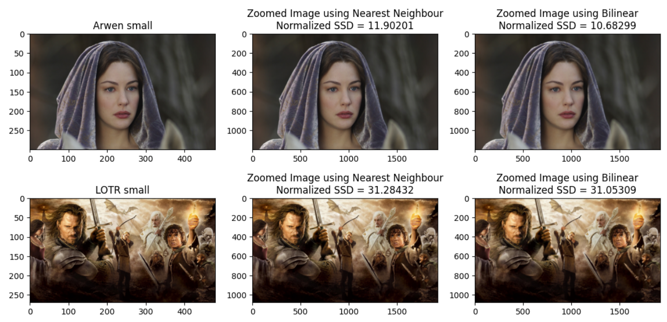
Arwen small
Zoomed Image using Nearest Neighbour
Normalized SSD = 11.90201
Zoomed Image using Bilinear
Normalized SSD = 10.68299
LOTR small
Zoomed Image using Nearest Neighbour
Normalized SSD = 31.28432
Zoomed Image using Bilinear
Normalized SSD = 31.05309

Figure 12: Zooming the images using Nearest Neighbour and Bilinear interpolation

Question 9

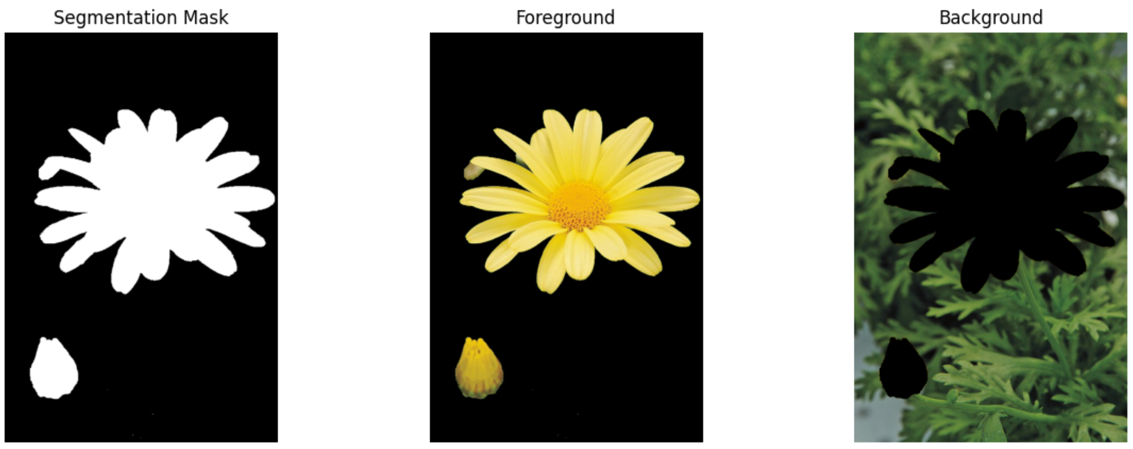

Figure 13: Segmentation using Grabcut

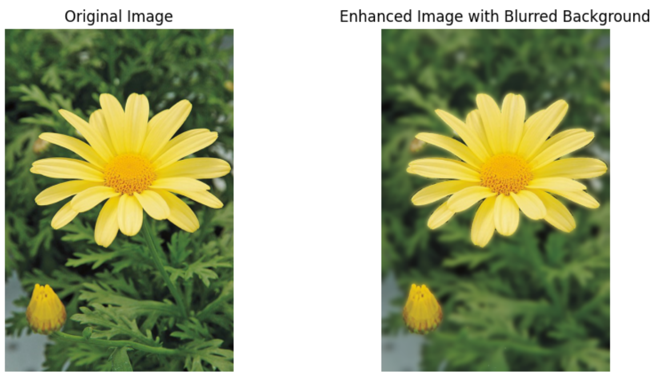

Figure 14: Background blurred

Reason for dark background just beyond the edge of the flower:
When the background image is blurred using a Gaussian filter, the pixels near the edges of the flower are replaced with zero (black) values. Since blurring works by averaging nearby pixels, the areas close to the flower's border get mixed with these black pixels, making the background near the edges appear darker.

```python
# Initialize the mask (all zeros)
init_mask = np.zeros(img.shape[:2], np.uint8)

# Background and foreground models (used internally by GrabCut)
bgd_model = np.zeros((1, 65), np.float64)
```

```python
fgd_model = np.zeros((1, 65), np.float64)

# Define a bounding box around the object (flower)
bounding_box = (50, 50, img.shape[1] - 50, img.shape[0] - 50)

# Apply the GrabCut algorithm
cv2.grabCut(img, init_mask, bounding_box, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)

# Apply Gaussian blur to the background image
blurred_bg = cv2.GaussianBlur(img_rgb, (35, 35), 0)

# Combine the blurred background with the foreground image
combined_image = blurred_bg * (1 - final_mask[:, :, np.newaxis]) + foreground_img
```

Github Repository for Assignment 01 here