DEPARTMENT OF ELECTRONIC & TELECOMMUNICATION ENGINEERING
UNIVERSITY OF MORATUWA

Assignment 01 - Intensity Transformation and Neighbourhood Filtering

By:

220399B M. M. M. Muftee

Submitted in partial fulfillment of the requirements for the module
EN3160 Image Processing and Machine Vision

Date: 12th of August 2025

Github Repository:

`https://github.com/Muftee-Mysan/EN3160---Image-Processing-and-Machine-Vision/tree/main/Assignments/Assignment_01`

Contents

1 Question 1: Basic Intensity Transformation

```python
# Step 1: Define piecewise transformation function
t1 = np.linspace(0, 50, num=51).astype('uint8')
t2 = np.linspace(100, 255, num=100).astype('uint8')
t3 = np.linspace(150, 255, num=105).astype('uint8')

# Step 2: Concatenate all segments to create the transformation array
t = np.concatenate((t1, t2, t3), axis=0).astype('uint8')
print(t.shape)
```

Figure 1: Code

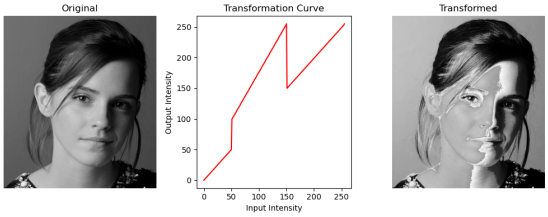

Figure 2: Intensity transformation on emma.jpg image

Interpretation: The transformation can be regarded as an identity mapping with a modification that selectively increases the intensity of mid-gray pixels. The key observation is the presence of jump discontinuities in the transformation function, which generate high-contrast regions in the resulting image. In contrast, the high-intensity (white) and low-intensity (dark) pixel values remain unaffected.

2 Question 2: Accentuating Brain Matter

Approach: A Gaussian function was employed as the intensity transformation function in order to enhance only the desired range of pixel intensities. The choice of a Gaussian function ensures smooth variations in contrast within the output image, thereby avoiding the blocky artifacts that arise from jump discontinuities in the transformation. Furthermore, the mean and standard deviation of the Gaussian can be adjusted, providing precise control over the range of pixel intensities to be emphasized. The transformation function was implemented using the following code:

```python
# Step 2: Define the gaussian pulse for white matter
mu = 150
sigma = 20
x = np.linspace(0, 255, 256)
white = 255 * np.exp(-((x - mu)**2) / (2 * sigma**2))

# Step 3: Define the gaussian pulse for white matter
mu = 200
sigma = 15
x = np.linspace(0, 255, 256)
gray = 255 * np.exp(-((x - mu)**2) / (2 * sigma**2))
```

Figure 3: Code

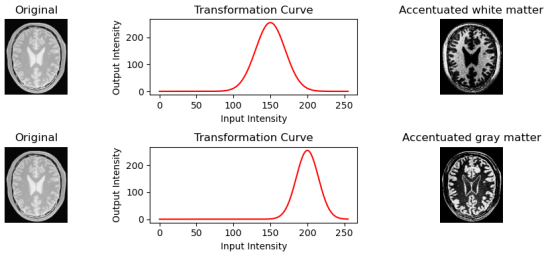

Figure 4: Intensity transformations on brain MRI image

Interpretation of Results: The transformation applied to highlight white matter is centered around an intensity value of approximately 150, producing the output image. Similarly, the transformation designed to emphasize gray matter is centered around an intensity value of approximately 200, yielding the output image. It is important to note that the inherent noise present in the original image remains unaffected and is not eliminated by the transformation.

3 Question 3: Gamma Correction in L*a*b* Space

Gamma Correction: The following code was used to perform gamma correction on the L-plane of the input image in order to adjust its overall lightness. The OpenCV utility `cv.COLOR_BGR2LAB` was employed to convert the image into the L*a*b* color space. The gamma value was fine-tuned to $\gamma = 0.75$, which effectively lightens the shadow regions while preserving the intensity of the highlights.

$$I_{\text{out}} = I_{\text{in}}^{\gamma} \tag{1}$$

where I_{in} and I_{out} denote the input and output pixel intensities, respectively, and γ is the gamma correction parameter.

```python
# Split the image in to L*a*b* color space
L,a,b = cv.split(img3_lab)

# Define and apply gamma transformation
gamma = 0.75
t = np.array([(i/255.0)**(gamma)*255 for i in np.arange(0, 256)]).astype('uint8')
L_modified = cv.LUT(L, t)

# Merge the channels
merged = cv.merge([L_modified, a, b])
img3_modified = cv.cvtColor(merged, cv.COLOR_LAB2RGB)
```

Figure 5: Code

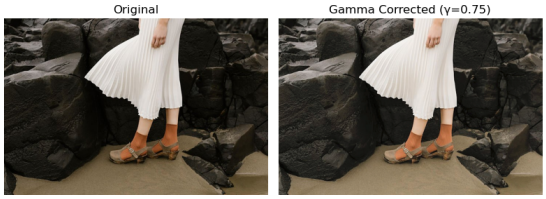

Figure 6: Original and gamma corrected output of highlights and shadows.jpg imag

Interpretation of Results: The histograms of the gamma-corrected image exhibit the presence of noise. Since gamma correction is a non-linear operation, applying it to the L-plane (which represents lightness in color spaces such as L*a*b*) may amplify small variations in pixel intensity. Due to the discrete nature of pixel values, the non-linear transformation redistributes the intensities in a non-uniform manner, thereby introducing quantization noise. As a result, the histogram may appear noisy as pixel intensities are spread unevenly across the range.

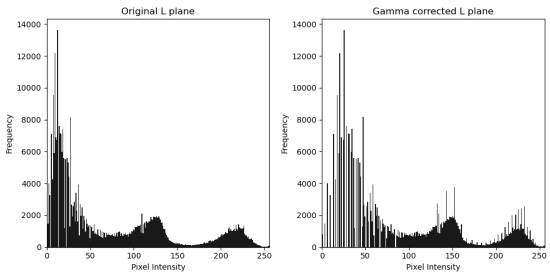

Figure 7: Histograms of the L plane

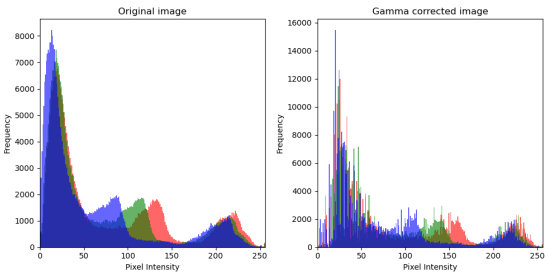

Figure 8: Histograms after converting to RGB

4 Question 4: Vibrance Enhancement

Intensity Transformation in HSV Color Space

Figure 10 illustrates the input image separated into its Hue, Saturation, and Value channels. The following intensity transformation was applied to the Saturation plane:

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

where x represents the input intensity, $a \in [0, 1]$, and $\sigma = 70$. By fine-tuning the parameter, $a = 0.6$ was selected as the optimal value. The resulting transformation is depicted in Figure 6, and the final processed image is shown in Figure 7. The following code snippet was used to apply the transformation exclusively to the Saturation channel.

Figure 9: Image

Interpretation of Results: The Saturation plane exhibits high values for the colorful regions of the image (e.g., the red areas of the spider suit), while the colorless or gray regions (e.g., buildings and sky) have low pixel values. By selectively amplifying the intensity of the colorful areas while leaving the colorless regions unchanged, the overall vibrance of the image is enhanced.

```python
# Split into planes
H, S, V = cv.split(spider_hsv)
# Define the transformation function
a = 0.6
sigma = 70.0
x = np.arange(0, 256)
f = np.minimum(x + a * 128 * np.exp(-((x - 128)**2) / (2 * sigma**2)), 255).astype('uint8')
# Apply transformation to Saturation plane
S_modified = cv.LUT(S, f)
# Merge
merged = cv.merge([H, S_modified, V])
spider_modified = cv.cvtColor(merged, cv.COLOR_HSV2RGB)
```

Figure 10: Code

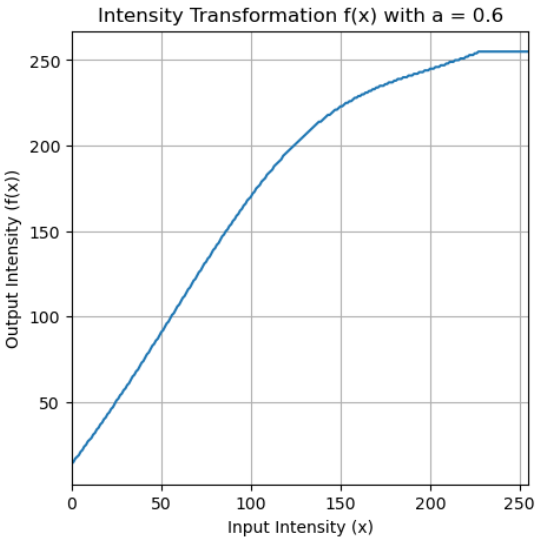

Figure 11: Plot

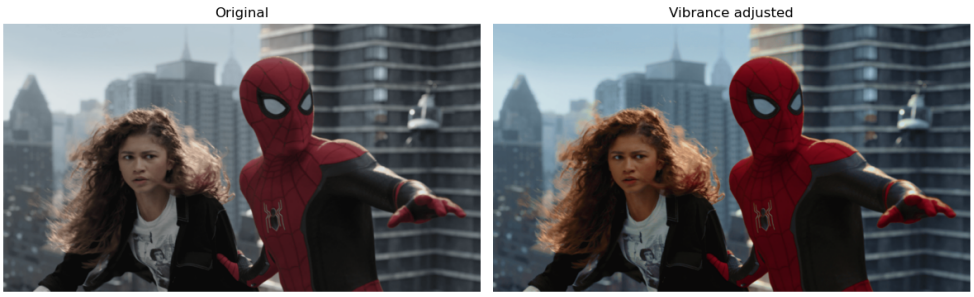

Figure 12: Enhanced Image

5 Question 5: Histogram Equalization

```python
def histogram_equalization(f):
    # Get image details
    L = 256
    M, N = f.shape

    # Get histogram
    hist = cv.calcHist([f], [0], None, [L], [0, L])
    cdf = hist.cumsum()

    # Define transformation
    t = np.array([(L-1)/(M*N)*cdf[k] for k in range(256)]).astype("uint8")

    return t[f]
```

Figure 13: Code

Interpretation of Results

As shown in Figure 15, the histogram of the image has become more evenly distributed following histogram equalization. However, this process has also introduced some noise, attributable to quantization effects. Notably, the low-intensity peaks remain largely unchanged.

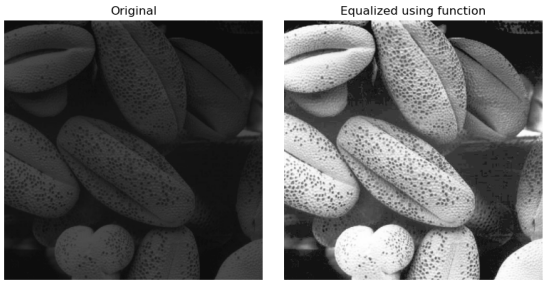

Figure 14: Output image after histogram equalization

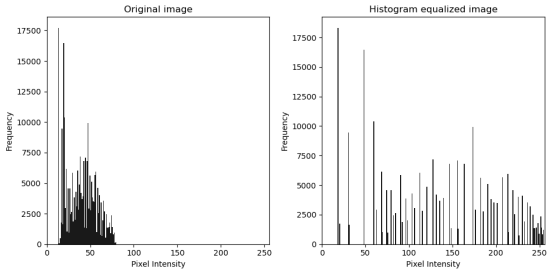

Figure 15: Histograms after equalization

6 Question 6: Foreground Histogram Equalization

```python
# Apply thresholding on Saturation plane
_, mask = cv.threshold(S, 12, 255, cv.THRESH_BINARY)

# Use bitwise_and to extract the foreground using the mask
foreground = cv.bitwise_and(jennifer, jennifer, mask=mask)
```

Figure 16: Code

Foreground Extraction and Histogram Analysis

Figure 17 illustrates the decomposition of the image `jennifer.jpg` into its Hue, Saturation, and Value channels. The Saturation plane is particularly suitable for generating a mask for foreground extraction. The following code was used to extract the foreground using this mask.

The extracted foreground is shown in Figure 18. The initial histogram of the foreground is presented in Figure 19, while the corresponding cumulative distribution function (CDF) is shown in Figure 20. Figure 21 depicts the histogram after applying histogram equalization to the foreground.

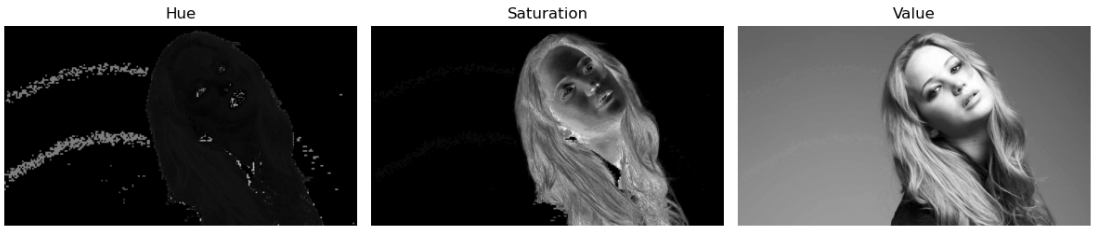

Figure 17: Figure

Figure 18: Results of histogram equalization to the foreground of jennifer.jpg image

Figure 19: Using cv.filter2D()

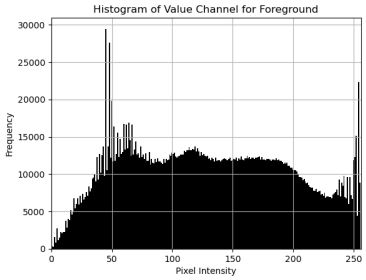

Figure 20: Using custom function

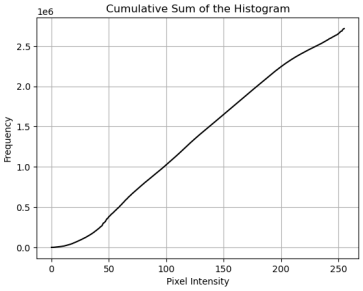

Figure 21: Using cv.filter2D()

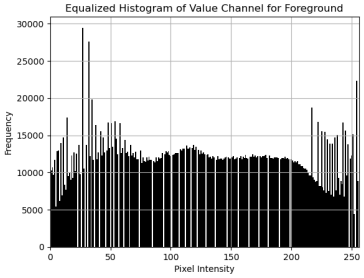

Figure 22: Using custom function

Interpretation of Results: The result of histogram equalization applied to the foreground is shown in Figure 22. The equalization process has increased the number of dark pixels, as can be observed by comparing Figures 22(a) and 22(b). This effect is also evident in the output image, where the shadows beneath the face appear darker.

7 Question 7: Sobel Filtering

```python
def zoom(img, technique, scale=4):
    if technique == 'nn':
        return cv.resize(img, None, fx=scale, fy=scale, interpolation=cv.INTER_NEAREST)
    elif technique == 'bilinear':
        return cv.resize(img, None, fx=scale, fy=scale, interpolation=cv.INTER_LINEAR)
```

Figure 23: Code

Sobel Edge Detection Using 2D Convolution

The following custom function was implemented to apply a 2D Sobel kernel using 2D convolution. Figure 24 shows the result obtained using OpenCV's `filter2D()` function, while Figure 25 presents the corresponding result produced by the custom function. Figure 26 illustrates the intermediate steps during the separable convolution of the Sobel kernels.

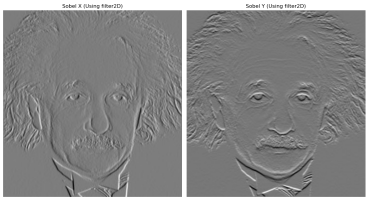

Figure 24: Using cv.filter2D()

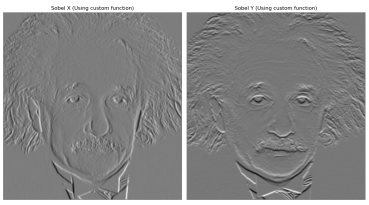

Figure 25: Using custom function

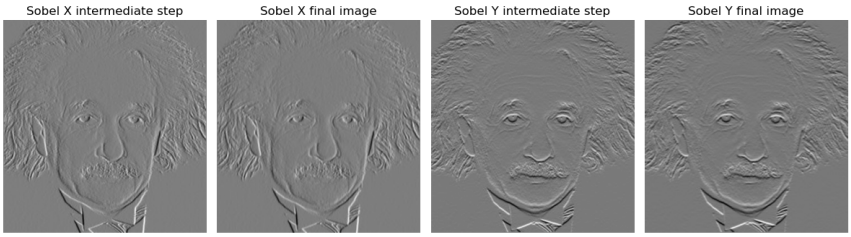

Figure 26: Using Seperable Convolution to filter the einstein.png image

8 Question 8: Image Zooming

The following function was used to zoom the images using a given interpolation method. The SSD values calculated for each image is given above it.

```
def zoom(img, technique, scale=4):
    if technique == 'nn':
        return cv.resize(img, None, fx=scale, fy=scale, interpolation=cv.INTER_NEAREST)
    elif technique == 'bilinear':
        return cv.resize(img, None, fx=scale, fy=scale, interpolation=cv.INTER_LINEAR)
```

Figure 27: Code

Figure 28: Image 01

Figure 29: Image 02

9 Question 9: Background Blurring

```
mask = np.zeros(daisy.shape[:2],np.uint8)
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
rect = (50,100,550,490)
cv.grabCut(daisy,mask,rect,bgdModel,fgdModel,5,cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
foreground = daisy * mask2[:, :, np.newaxis]
background = cv.subtract(daisy, foreground)
```

Figure 30: The following code utilizes Grabcut to segment the foreground.

Figure 31: Seperation using Grabcut

Figure 32: Background blurred

Reason for the Dark Background Beyond the Edge of the Flower: When the background image is blurred using a Gaussian kernel, the pixels belonging to the foreground are replaced with zero values. Since Gaussian blur functions as an averaging filter, the regions near the object boundaries are blended with these zero-valued pixels. As a result, the border areas appear darker.