University of Moratuwa, Sri Lanka

Faculty of Engineering

Department of Electronics and Telecommunication Engineering
Semester 5 (Intake 2021)

EN3160 - Image Processing and Machine Vision

Assignment 01
Intensity Transformations and Neighborhood Filtering

Sehara G.M.M - 210583B

This report is submitted as a partial fulfillment for the moduleEN3160 - Image Processing and Machine Vision, Department of Electronic and Telecommunication Engineering, University of Moratuwa.

1 Intensity Transformation

Here, a piece wise curve is used for the intensity transformation. The code snippet below was implemented to apply the equation, and the resulting output is shown.

Listing 1: Piece-wise intensity curve

```
# Define control points for intensity windowing based on the given image
c = np.array([(50, 50), (50, 100), (150, 255), (150, 150)])

# Create piecewise linear transformation
t1 = np.linspace(0, c[0, 1], c[0, 0] + 1 - 0).astype('uint8')
t2 = np.linspace(c[0, 1], c[1, 1], c[1, 0] - c[0, 0] + 1).astype('uint8')
t3 = np.linspace(c[1, 1], c[2, 1], c[2, 0] - c[1, 0] + 1).astype('uint8')
t4 = np.linspace(c[2, 1], c[3, 1], c[3, 0] - c[2, 0] + 1).astype('uint8')
t5 = np.linspace(c[3, 1], 255, 255 - c[3, 0] + 1).astype('uint8')  # Adjust to include 255

# Concatenate the segments to form the transformation function
transform = np.concatenate((t1, t2[1:], t3[1:], t4[1:], t5[1:]), axis=0)
```

Figure 1: Original and Output Image

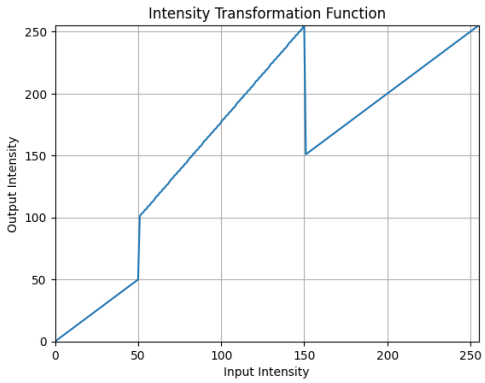

Figure 2: Transformation function

Figure 3

The darker areas of the image gets more brighter and some brightest parts of the image get more dark.

2 Intensity Transformations of White Matter and Gray Matter

The image above illustrates the brain's white and gray matter. The white matter is located in the center, while the outer cells consist of gray matter. The following method can be used to extract these regions. I applied sigmoid and inverse sigmoid functions for the extraction process.

```
# Define the intensity transformation function to attenuate gray matter
def intensity_transformation(value):
    if 180 <= value <= 255:  # Gray matter intensities
        return int(1.5* value)  # Attenuate gray matter (reduce intensity by half)
    else:
```

```
        return value  # Leave other regions unchanged

# Define the intensity transformation function to accentuate gray matter
def intensity_transformation(value):
    if 180 <= value <= 255:  # Gray matter intensities
        return min(255, int(2 * value))  # Enhance gray matter (boost
           intensities)
    else:
        return value  # Leave other regions unchanged
```

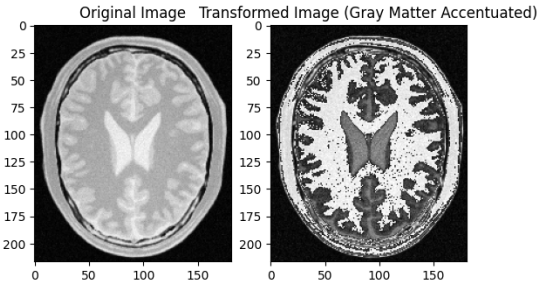

Figure 4: Gray Matter

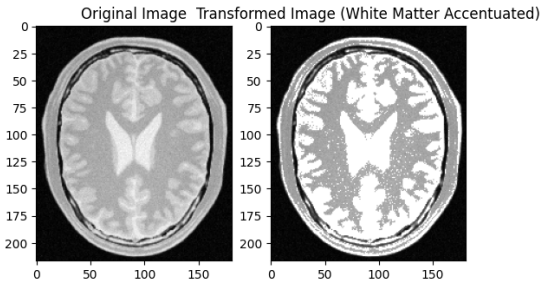

Figure 5: White Matter

Figure 6: Comparison of Gray and White Matter

3 Gamma Correction

Here, we apply linear plane Gamma Correction using gamma value = 2. A gamma value of 1 corresponds to the "Original Image."

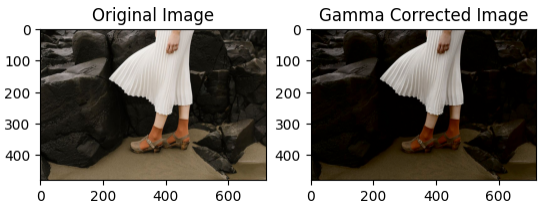

Figure 7: Original and Gamma Corrected Images

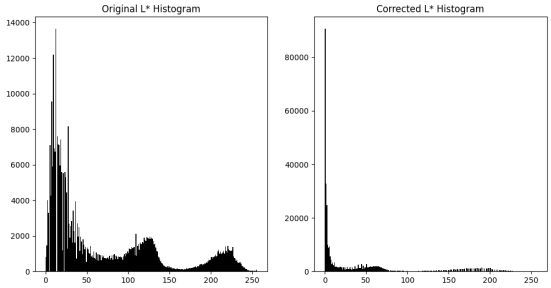

Figure 8: Histograms

Figure 9

4 Vibration Enhancement

Here, the image is split into Hue, Saturation, and Value planes. An intensity transformation is applied to the Saturation plane to enhance the vibrance of the image. The code snippet below performs the intensity transformation using alpha = 0.75 and sigma = 70.

```
#Split the image into hue, saturation, and value planes.
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

h, s, v = cv.split(hsv_image)
```

```
5
6  # Define constants for the transformation
7  a = 0.75  # example value for scaling factor a
8  sigma = 70
9
10 # Apply the transformation to the saturation plane
11 x = np.array(s, dtype=np.float32)
12 transformed_s = np.minimum(x + a * 128 * np.exp(-((x - 128) ** 2) / (2 *
       sigma ** 2)), 255)
```

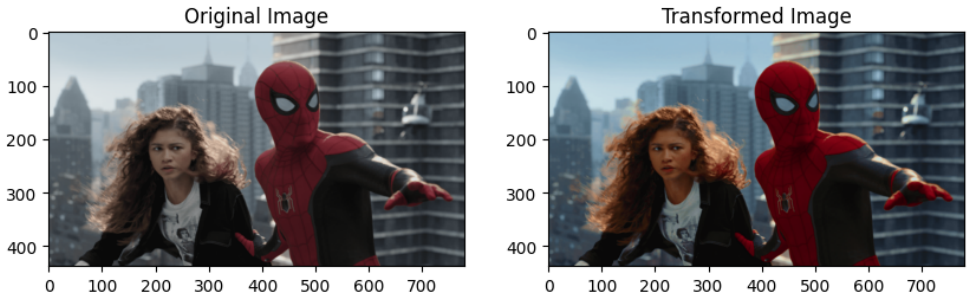

Figure 10: Original and Transformed Image

5 Histogram Equalization

A histogram represents the pixel intensity distribution of an image. When histogram equalization is applied, the pixel intensities are redistributed across the full range of 0-255. The code snippet below demonstrates the histogram equalization function. After applying it, darker areas become brighter, resulting in a clearer image. The histograms will be spread evenly across the 0-255 range.

```
1  # Step 1: Calculate the histogram of the original image
2  hist, bins = np.histogram(image.flatten(), 256, [0, 256])
3
4  # Step 2: Compute the cumulative distribution function (CDF)
5  cdf = hist.cumsum()
6
7  # Normalize the CDF for visualization (Optional)
8  cdf_normalized = cdf * hist.max() / cdf.max()
9
10 # Step 3: Mask CDF values where the CDF is 0 and normalize the rest
11 cdf_m = np.ma.masked_equal(cdf, 0)  # Mask pixels where CDF is 0
12 cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())  #
       Normalize to 0-255 range
13 cdf = np.ma.filled(cdf_m, 0).astype('uint8')  # Fill the masked values
       with 0 and convert to uint8
14
15 # Step 4: Use the CDF to remap pixel values in the original image
16 equalized_image = cdf[image]
17
18 # Step 5: Calculate the histogram of the equalized image
19 equalized_hist, bins = np.histogram(equalized_image.flatten(), 256, [0,
       256])
```

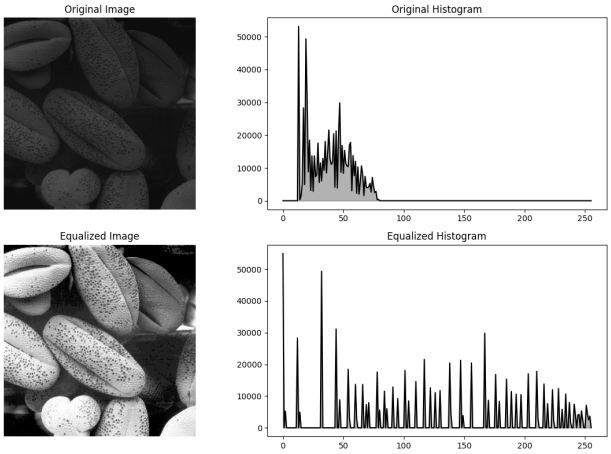

Figure 11

6 Applying histogram equalization only to the foreground of an image

In this process, we import the image and split it into Hue, Saturation, and Value planes, selecting the Saturation plane for mask extraction. We create a mask using a threshold value of 11 to extract the foreground with a bitwise AND operation. The background is extracted using the inverse mask. After applying histogram equalization to enhance the brightness of the foreground, we merge it with the background to obtain the final brightened image of the character.

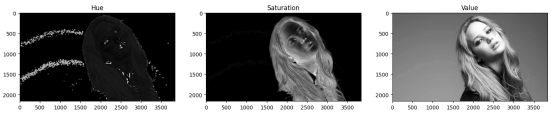

Figure 12

Figure 13

Figure 14

7 Sobel Filtering

7.1 Filter using existing filter2D method

When apply the sobel filtering with filter2D method for the following sobal kernals we can see the following output.

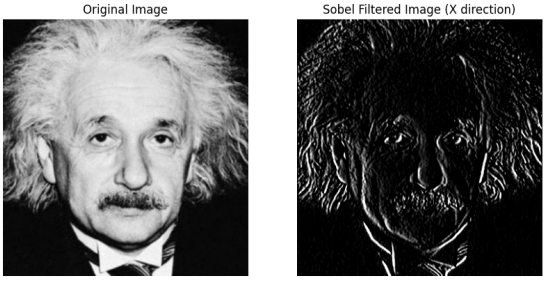

Figure 15: Applying Sobel Filter

7.2 Filtering using a numerical method

Here we multiply the matrices and obtain the output of the Sobel filtering. The following code snippet derives how the filtering happens

```

# Get image dimensions
height, width = albert_img.shape

# Define Sobel kernels
sobel_kernel_x = np.array([[1, 0, -1],
                           [2, 0, -2],
                           [1, 0, -1]])

sobel_kernel_y = np.array([[1, 2, 1],
                           [ 0,  0,  0],
                           [ -1, -2,  -1]])

# Initialize empty arrays to store results
sobel_x = np.zeros_like(albert_img, dtype=np.float64)
sobel_y = np.zeros_like(albert_img, dtype=np.float64)

# Padding the image to handle the edges
padded_img = np.pad(albert_img, ((1, 1), (1, 1)), mode='constant')

# Perform convolution with the Sobel kernel manually
for i in range(1, height+1):
    for j in range(1, width+1):
        # Extract the 3x3 region from the padded image
        region = padded_img[i-1:i+2, j-1:j+2]

        # Apply the Sobel kernels
        gx = np.sum(sobel_kernel_x * region)
        gy = np.sum(sobel_kernel_y * region)

        # Store the result in the output arrays
        sobel_x[i-1, j-1] = gx
        sobel_y[i-1, j-1] = gy

# Combine the horizontal and vertical Sobel results to get the gradient
   magnitude
sobel_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
sobel_magnitude = np.clip(sobel_magnitude, 0, 255)  # Clip values to valid
    range

# Convert the result back to uint8 format
sobel_x = np.uint8(np.abs(sobel_x))
sobel_y = np.uint8(np.abs(sobel_y))
sobel_magnitude = np.uint8(sobel_magnitude)
```

7.3 Using the given property

```

# Define the 1D Sobel kernels (separable)
```

```
sobel_kernel_v = np.array([1, 2, 1])  # Vertical 1D kernel (for smoothing)
sobel_kernel_h = np.array([1, 0, -1])  # Horizontal 1D kernel (for edge
    detection)

# Initialize empty arrays to store intermediate and final results
sobel_x_intermediate = np.zeros_like(albert_img, dtype=np.float64)
sobel_x = np.zeros_like(albert_img, dtype=np.float64)

# Padding the image to handle the edges (for 1D vertical filter)
padded_img = np.pad(albert_img, ((1, 1), (0, 0)), mode='constant')

# Step 1: Apply vertical 1D kernel (along the rows)
for i in range(1, height+1):
    for j in range(0, width):
        # Apply the vertical kernel to the 3-pixel neighborhood in the
            column
        sobel_x_intermediate[i-1, j] = np.sum(padded_img[i-1:i+2, j] *
            sobel_kernel_v)

# Padding the intermediate result for horizontal filtering
padded_intermediate = np.pad(sobel_x_intermediate, ((0, 0), (1, 1)), mode=
    'constant')

# Step 2: Apply horizontal 1D kernel (along the columns)
for i in range(0, height):
    for j in range(1, width+1):
        # Apply the horizontal kernel to the 3-pixel neighborhood in the
            row
        sobel_x[i, j-1] = np.sum(padded_intermediate[i, j-1:j+2] *
            sobel_kernel_h)

# Convert the result back to uint8 format, normalizing the values to 0-255
sobel_x = np.uint8(np.clip(np.abs(sobel_x), 0, 255))
```

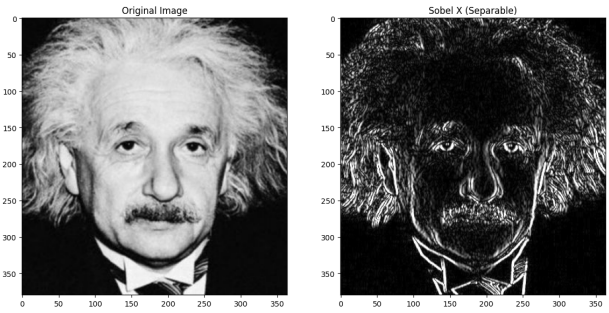

Figure 16

7.4 Image Zooming

The following function will zoom the image by faster 4. Compared to the given large images, the zoomed images have little bit of noise. The following results were also obtained.

Figure 17

Figure 18

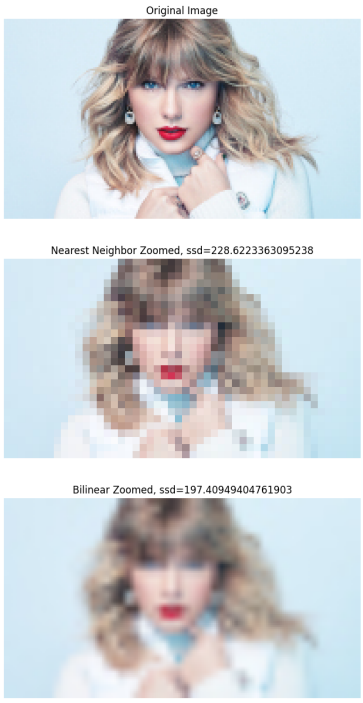

Figure 19

Figure 20: Comparison of All Images

7.5 Image Segmentation

Here we use grabCut segmentation to mask the foreground and the background of the image. After masking we add a gaussian blur to the background and then add the foreground and the background to produce a blurred-background enhanced image.

Figure 21: Enter Caption