Department of Electronic & Telecommunication Engineering,
University of Moratuwa,
Sri Lanka.

Assignment 1

Student Details:

210285M Kavishan G.T.

EN3160 - Image Processing and Machine Vision

Date: 2024.09.29

1 Question 01

```
# Original Image Histogram
plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap='gray')

# Define the intensity transformation function based on the given plot
def intensity_transform(value):
    if value < 50:
        return value  # Linear for [0, 50)
    elif value < 150:
        return (1.55*value +22.5) # Linear for [50, 150)
    elif value < 255:
        return value    # Mapping 100-150 to 255

# Apply the transformation to each pixel in the image
transformed_image = np.vectorize(intensity_transform)(image)
# Define input intensity levels
input_intensity = [0, 50, 50, 150, 150, 255]

# Define corresponding output intensity levels
output_intensity = [0, 50, 100, 255, 150, 255]
```

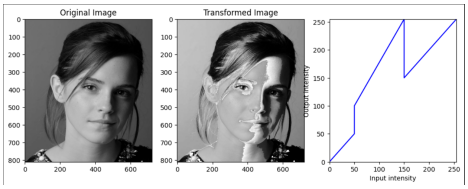


Figure 1: Intensity Transformed Image

2 Question 03

Figure 2: Original Image

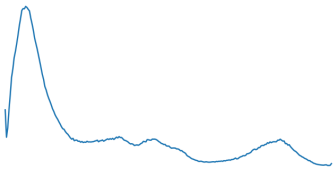


Figure 3: Histogram for Original Image

Figure 4: Gamma corrected Image

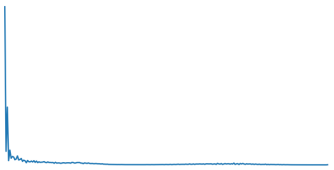


Figure 5: Histogram for Gamma Corrected Image

In this correction I have used gamma = 2 for the gamma correction. Main part of the code is given below.

```
cvt_img = cv.cvtColor(image3, cv.COLOR_BGR2Lab)
fig, ax = plt.subplots(1,4, figsize = (18,4))
L,a,b = cv.split(cvt_img)
# Apply gamma correction to the L channel
gamma = 2  # You can modify the gamma value
L_float = L / 255.0  # Normalize to [0, 1]
L_gamma_corrected = np.power(L_float, gamma) * 255  # Gamma correction
L_gamma_corrected = np.uint8(L_gamma_corrected)

# Merge the corrected L with original a, b channels
lab_gamma_corrected = cv.merge([L_gamma_corrected, a, b])

# Convert back to BGR color space
corrected_image = cv.cvtColor(lab_gamma_corrected, cv.COLOR_LAB2BGR)

#plot histogram for original image
hist1, bins = np.histogram(image3.ravel(), 256, [0, 256])
```

3 Question 04

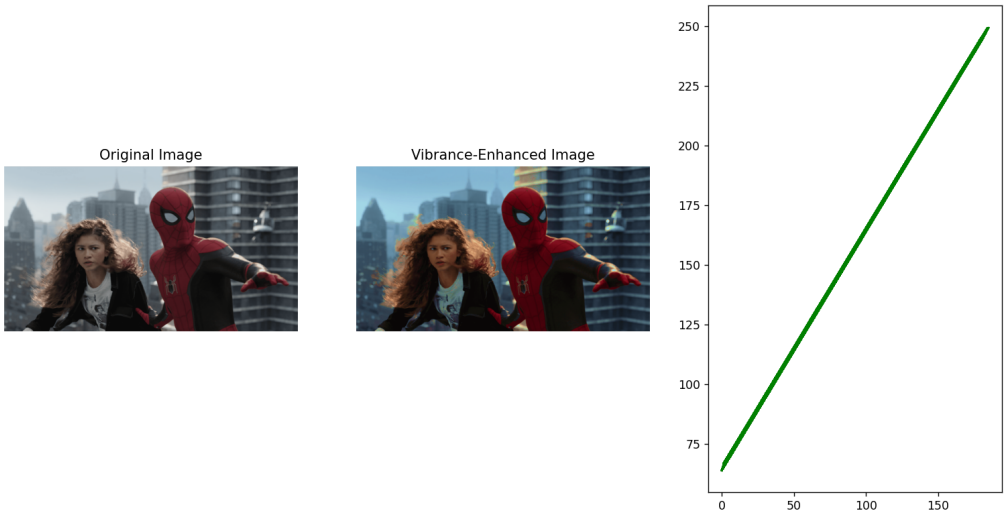


Figure 6: Enter Caption

```
#convert image to hsv type
hsv_image = cv.cvtColor(image4, cv.COLOR_BGR2HSV)
#split the image
hue, saturation, value = cv.split(hsv_image)
saturation_transform = np.minimum(saturation + a* 128* np.exp(-((saturation- 128)**2)/
# Convert the result back to an 8-bit format
saturation_transformed = np.uint8(saturation_transform)
hsv_transform = cv.merge([hue, saturation_transformed, value])
# Convert back to BGR color space for display
image_transformed = cv.cvtColor(hsv_transform, cv.COLOR_HSV2BGR)
```

4 Question 05

The function that I used to histogram equalization is given below.

```
def histogram_equalization(image):
    """
    Perform histogram equalization on a grayscale image.
    """
    # Get the histogram of the original image
    hist, bins = np.histogram(image.flatten(), 256, [0, 256])

    # Compute the cumulative distribution function (CDF)
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()

    # Masking and normalizing the CDF (ignores zero values)
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
    cdf_final = np.ma.filled(cdf_m, 0).astype('uint8')

    # Map the original image pixels based on the CDF
    image_equalized = cdf_final[image]

    return image_equalized, hist, cdf_final
```

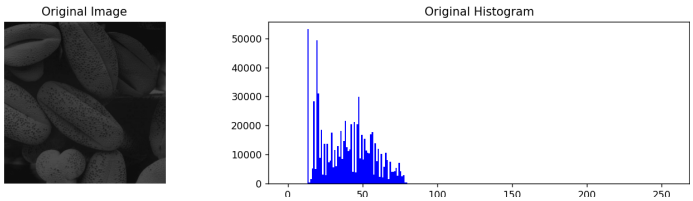


Figure 7: Histogram for original Image

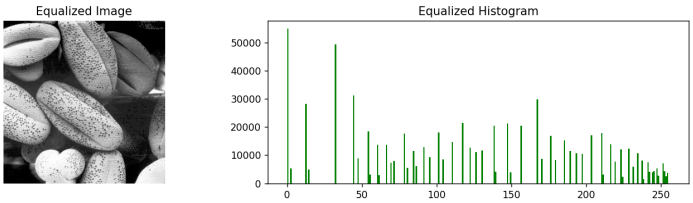


Figure 8: Histogram for after equalization

Histogram equalization is a technique used to enhance the contrast of an image by redistributing the pixel intensities to achieve a more uniform distribution. The primary goal is to spread out the most frequent intensity values, so the image appears clearer, with better contrast between different areas.

5 Question 06

5.1 Display images in grayscale for hue, saturation and value plane

Figure 9: HSV image in separate planes

```
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

```
# Split the image into Hue, Saturation, and Value planes
hue, saturation, value = cv2.split(hsv_image)
```

5.2 Select the appropriate plane to threshold in extract the foreground mask.

```
# Threshold the value plane to extract foreground
_, mask = cv2.threshold(value, 120, 255, cv2.THRESH_BINARY)
```

In this case we use value plane to threshold in foreground mask.

Figure 10: Foreground mask for value plane

5.3 obtain the foreground only using cv.bitwise-and and cumulative sum of the histogram

```
# Use mask to extract the foreground
foreground = cv2.bitwise_and(value, value, mask=mask)
# Compute the histogram of the foreground
foreground_hist = cv2.calcHist([foreground], [0], mask, [256], [0, 256])

#Compute the cumulative sum of the histogram
cdf = np.cumsum(foreground_hist)
cdf_normalized = cdf * (foreground_hist.max() / cdf.max())
```

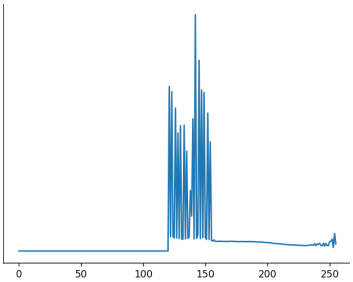


Figure 11: Histogram for foreground mask

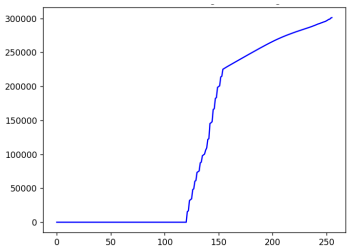


Figure 12: Cumulative distribution for histogram

These are the code and outputs for the original image, and the result with the histogramequalized foreground.

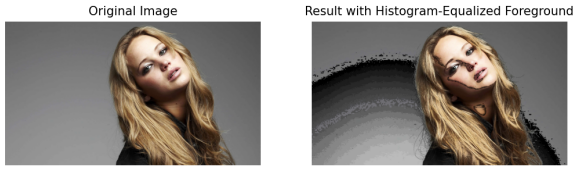


Figure 13: original image, and the result with the histogramequalized foreground.

```
#Extract the background
background = cv2.bitwise_and(value, value, mask=cv2.bitwise_not(mask))
# Combine the equalized foreground and background
result = cv2.add(foreground_equalized, background)
# Merge back into the HSV image
hsv_image_equalized = cv2.merge([hue, saturation, result])
# Convert back to BGR for display
image_equalized = cv2.cvtColor(hsv_image_equalized, cv2.COLOR_HSV2BGR)
```

6 Question 07

6.1 Using the existing filter2D to Sobel filter the image.

```
# Define the Sobel kernels for X and Y directions
sobel_x = np.array([[1, 0, -1],
                    [2, 0, -2],
                    [1, 0, -1]])

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

# Apply Sobel filter using filter2D for both X and Y directions
sobel_x_filtered = cv2.filter2D(image, -1, sobel_x)
sobel_y_filtered = cv2.filter2D(image, -1, sobel_y)

# Compute the gradient magnitude
sobel_combined = np.sqrt(sobel_x_filtered**2 + sobel_y_filtered**2)
sobel_combined = np.uint8(sobel_combined)
```

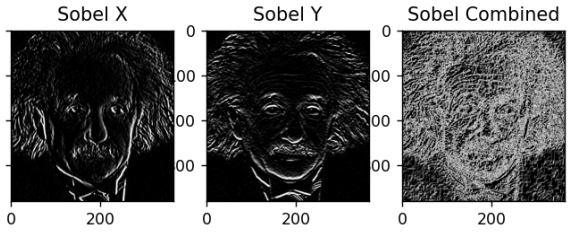


Figure 14: Sobel fitering using filter2D

The following code is my own code for the sobel filtering which is used convolution for the filtering the image.

```
def custom_sobel_filter(image, kernel):
    """
    Custom implementation of 2D convolution using a Sobel kernel.
    """
```

```
    # Get image dimensions
    rows, cols = image.shape
    kernel_size = kernel.shape[0]
    pad_size = kernel_size // 2

    # Pad the image to handle borders
    padded_image = np.pad(image, pad_size, mode='constant', constant_values=0)

    # Initialize output image
    output = np.zeros_like(image)

    # Perform 2D convolution
    for i in range(rows):
        for j in range(cols):
            # Extract the region of interest
            region = padded_image[i:i+kernel_size, j:j+kernel_size]
            # Perform element-wise multiplication and sum
            output[i, j] = np.sum(region * kernel)

    return output
```

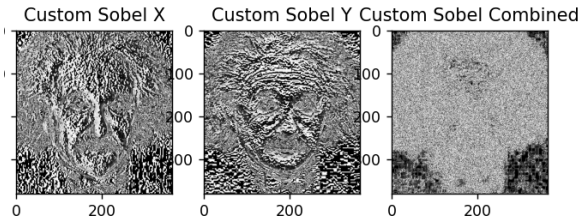


Figure 15: Sobel filter

6.1.1 Part C

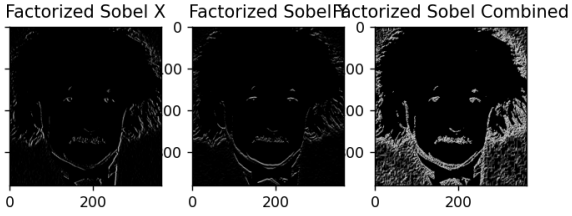


Figure 16: Factorized Sobel Filtering

7 Question 08

The code is consisted with two functions for zooming image and calculate the ssd value. We use zoom factor = 4.

```
def zoom_image(image, s, method='nearest'):
    if method == 'nearest':
        zoomed_image = cv2.resize(image, None, fx=s, fy=s, interpolation=cv2.INTER_NEA
    elif method == 'bilinear':
        zoomed_image = cv2.resize(image, None, fx=s, fy=s, interpolation=cv2.INTER_LIN
    else:
        raise ValueError("Method must be 'nearest' or 'bilinear'.")
    return zoomed_image
```

```
def normalized_ssd(image1, image2):
    # Ensure images are the same shape
    if image1.shape != image2.shape:
        raise ValueError("Images must have the same dimensions.")
    # Compute the SSD
    diff = image1.astype(float) - image2.astype(float)
    ssd = np.sum(diff**2) / np.prod(image1.shape)
    return ssd
```

Normalized SSD for nearest neighbor : 136.290
Normalized SSD for bilinear : 115.0919

Normalized SSD for nearest neighbor : 26.4461
Normalized SSD for bilinear : 18.3459

8 Question 09

8.1 Use grabCut to segment the image

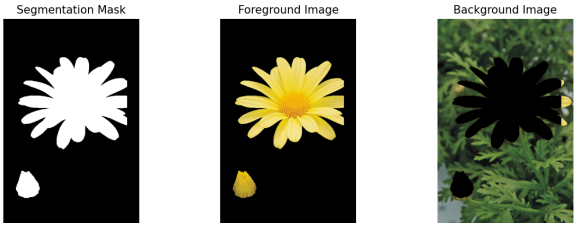
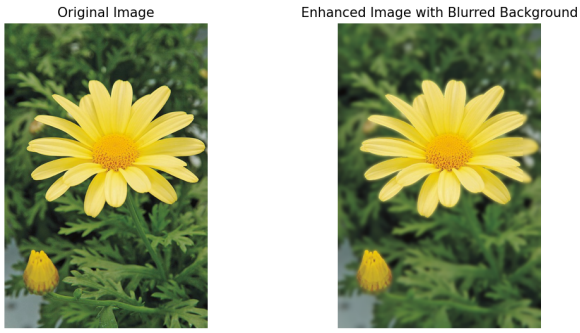


Figure 17: Grabcut segmentation

```
# Step 1: Initialize the mask (0s for background, 1s for probable background, 2s
mask = np.zeros(image.shape[:2], np.uint8)
# Step 2: Create background and foreground models (used by GrabCut internally)
bgd_model = np.zeros((1, 65), np.float64)
fgd_model = np.zeros((1, 65), np.float64)
# Step 3: Define a bounding box around the flower (foreground region)
# For demonstration, we assume the flower is in the center of the image.
rect = (50, 50, image.shape[1] - 100, image.shape[0] - 100)
# Step 4: Apply GrabCut
cv2.grabCut(image, mask, rect, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)
# Post-processing the mask
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
# Extract the foreground
foreground = image * mask2[:, :, np.newaxis]
# Extract the background
background = image * (1 - mask2[:, :, np.newaxis])
```

8.2 Produce an enhanced image with a substantially blurred background.




```
# Step 1: Blur the background using GaussianBlur
blurred_background = cv2.GaussianBlur(image, (25, 25), 0)

# Step 2: Combine the blurred background with the original foreground
enhanced_image = blurred_background * (1 - mask2[:, :, np.newaxis]) + foreground
```

The darker regions near the edges of a flower in an image after GrabCut segmentation and Gaussian blur can arise from several factors. GrabCut may misclassify parts of the background near the flower as foreground, causing those areas to remain unblurred and appear sharper and darker. Additionally, Gaussian blur averages nearby pixels, so if the flower's edges are close to dark background regions, this blending can darken the edges. Low contrast between the flower and background can also obscure the distinction, making the edges seem darker. Finally, blurring can introduce edge artifacts, creating halo effects around the flower that further darken the transition areas.

Check out my project on GitHub: Image-processing-assignment