BCSE417 -L43-L44 - MACHINE VISION LAB 1

Nihar Thampi - nihar.thampi2021@vitstudent.ac.in - 21BAI1027
August 16th,2024

By turning in this assignment, I declare that all of this is my own work

---

Assignment 1

Task 1: Basic Image Statistics and Color Space Conversion
Objective: Compute basic statistics and convert an image into different color spaces.
Steps:
1. Read the Image: Load an image using OpenCV.
2. Compute Basic Statistics: Calculate the mean, standard deviation, and histogram of each color channel.
3. Convert Color Spaces: Convert the image to HSV and Lab color spaces and display the results.

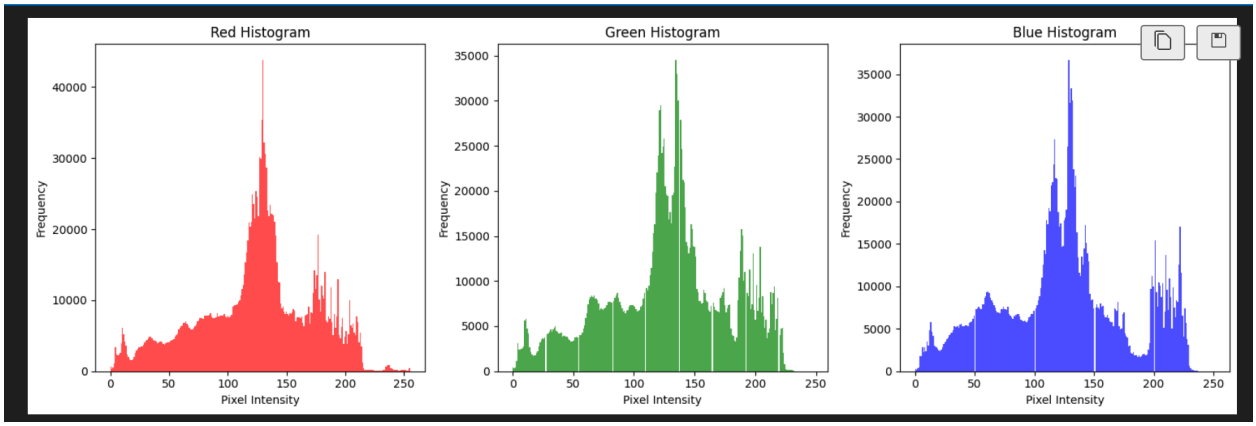
Code:

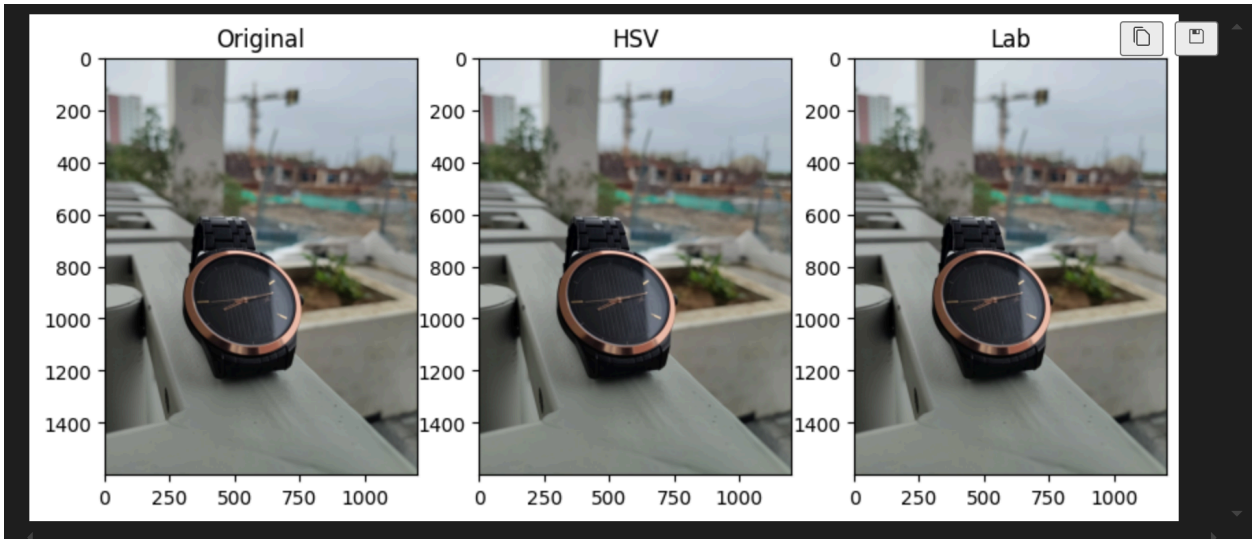
```
import numpy as np
import os
import cv2 as cv
import matplotlib.pyplot as plt

path = r"watch.jpg"
color_image = cv.imread(path)
plt.title('Original Image')
plt.imshow(cv.cvtColor(color_image, cv.COLOR_BGR2RGB))
plt.axis('off')
img_rgb = cv.cvtColor(color_image, cv.COLOR_BGR2RGB)
channels = cv.split(img_rgb)
channel_names = ['Red', 'Green', 'Blue']

stats = {}
for i, channel in enumerate(channels):
    mean = np.mean(channel)
    std = np.std(channel)
    stats[channel_names[i]] = {'mean': mean, 'std': std}
stats
```

```
plt.figure(figsize=(15, 5))
for i, channel in enumerate(channels):
    plt.subplot(1, 3, i+1)
    plt.hist(channel.ravel(), bins=256, color=channel_names[i].lower(),
alpha=0.7)
    plt.title(f'{channel_names[i]} Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

hsv_image = cv.cvtColor(color_image, cv.COLOR_BGR2HSV)
lab_image = cv.cvtColor(color_image, cv.COLOR_BGR2Lab)

plt.figure(figsize=(10, 5))
plt.subplot(131), plt.imshow(cv.cvtColor(color_image, cv.COLOR_BGR2RGB)),
plt.title('Original')
plt.subplot(132), plt.imshow(cv.cvtColor(hsv_image, cv.COLOR_HSV2RGB)),
plt.title('HSV')
plt.subplot(133), plt.imshow(cv.cvtColor(lab_image, cv.COLOR_Lab2RGB)),
plt.title('Lab')
plt.show()
```

Output:

```
{'Red': {'mean': 124.17683492922565, 'std': 47.93543480435334},
 'Green': {'mean': 127.06718880099916, 'std': 50.32875271596533},
 'Blue': {'mean': 123.84299385928394, 'std': 53.819046166915406}}
```





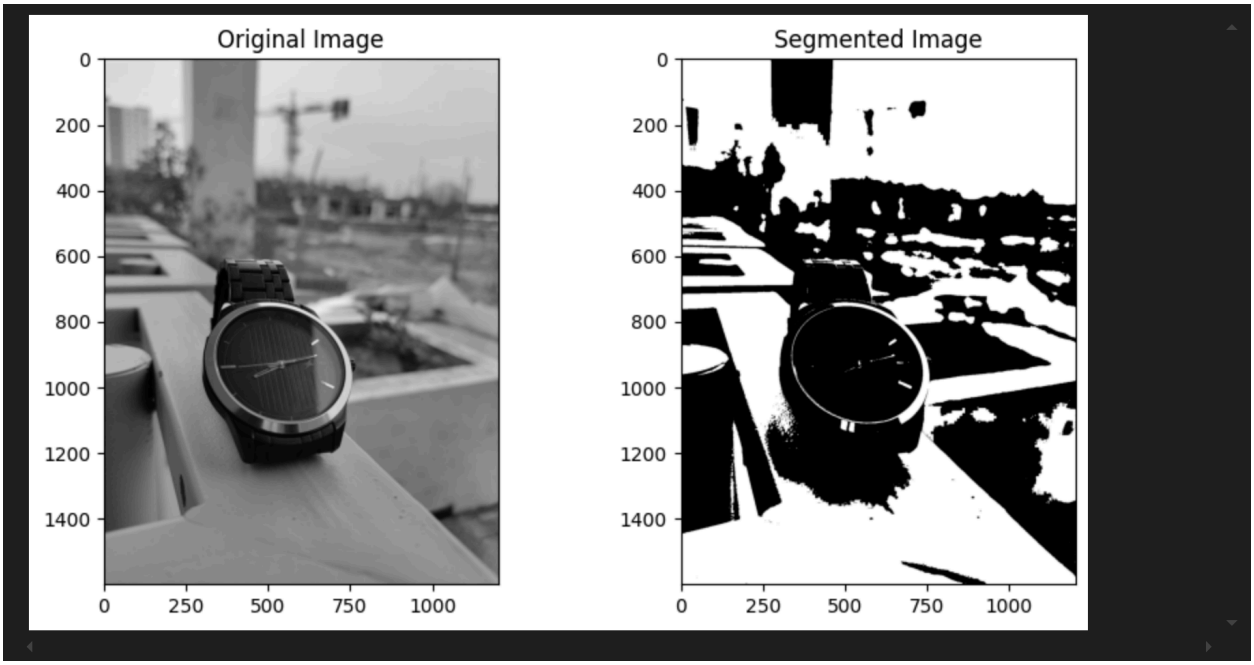
Task 2: Simple Image Segmentation Using Thresholding
Objective:
Segment an image into foreground and background using global thresholding.
Steps:
Read the Image: Load a grayscale image.
1.
Apply Thresholding: Use a fixed threshold value to segment the image.
2.
Display Results: Show the original and segmented images.

Code:

```
image_gray = cv.imread(r"watch.jpg", cv.IMREAD_GRAYSCALE)
threshold_value = 125
_, segmented_image = cv.threshold(image_gray, threshold_value, 255,
cv.THRESH_BINARY)
plt.figure(figsize=(10, 5))
```

```
plt.subplot(121), plt.imshow(image_gray, cmap='gray'), plt.title('Original
Image')
plt.subplot(122), plt.imshow(segmented_image, cmap='gray'),
plt.title('Segmented Image')
plt.show()
```

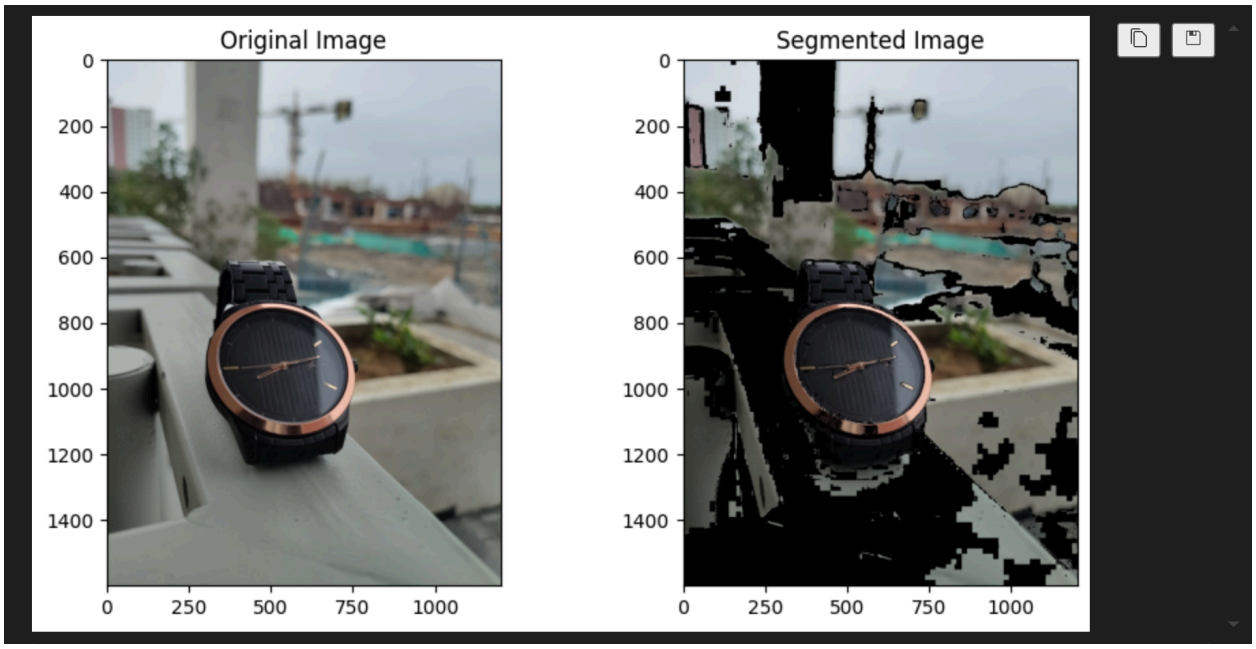
Output:



Task 3: Color-Based Segmentation
Objective:
Segment specific objects in an image based on their color.
Steps:
Read the Image: Load an image with objects of different colors.
1.
Convert to HSV: Convert the image to HSV color space.
2.
Apply Color Thresholding: Use color thresholds to segment objects of a specific color.
3.
Display Results: Show the original and segmented images.

```
image = cv.imread('watch.jpg')
hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
lower_red = np.array([5, 14, 11])
upper_red = np.array([255,255, 255])
```

```
mask = cv.inRange(hsv, lower_red, upper_red)
segmented_object = cv.bitwise_and(image, image, mask=mask)
plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB)),
plt.title('Original Image')
plt.subplot(122), plt.imshow(cv.cvtColor(segmented_object,
cv.COLOR_BGR2RGB)), plt.title('Segmented Image')
plt.show()
```



Assignment 2

Task 1: Image Negative Transformation
Objective: Create an image negative using Python.
Steps:

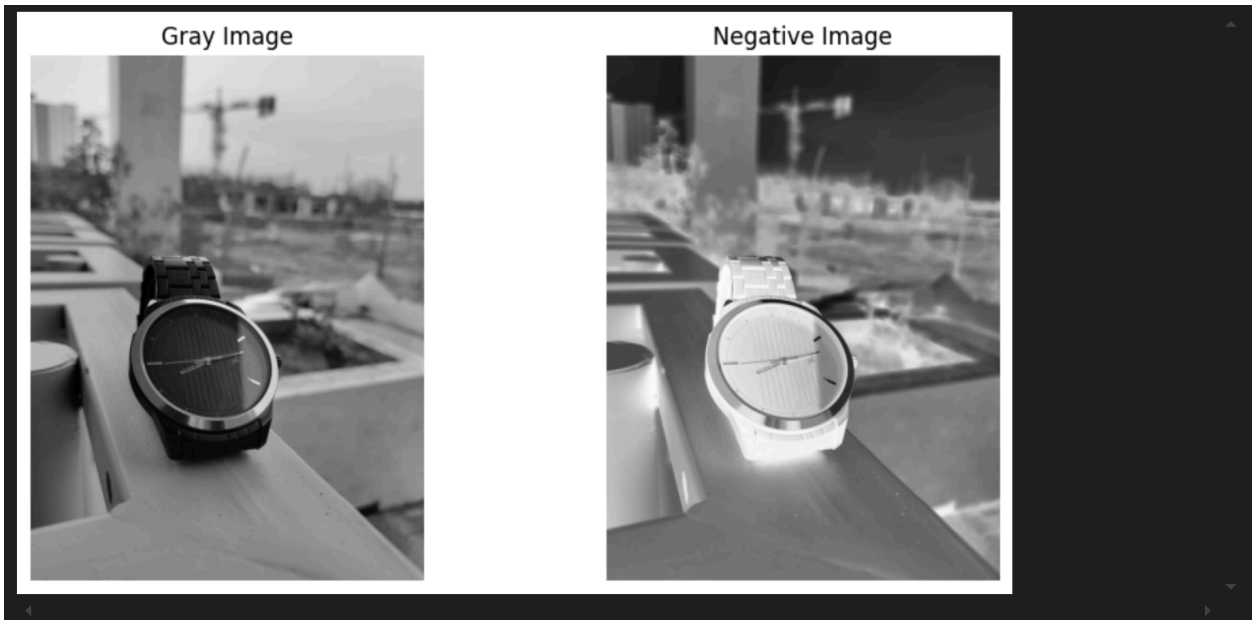
1. Load a grayscale image.
2. Apply the image negative transformation.
3. Display the original and the negative image

Code:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
path = r"watch.jpg"
gray_image = cv.imread(path, cv.IMREAD_GRAYSCALE)
def apply_negative_transformation(image):
```

```
    return 255 - image
negative_image = apply_negative_transformation(gray_image)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Gray Image')
plt.imshow(gray_image, cmap='gray')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.title('Negative Image')
plt.imshow(negative_image, cmap='gray')
plt.axis('off')

plt.show()
```

Output:



Task 2: Gamma Correction

Objective: Apply gamma correction with different gamma values.

Steps:

1. Load a grayscale image.
2. Apply gamma correction with γ=0.5\gamma = 0.5γ=0.5, γ=1.0\gamma = 1.0γ=1.0, and γ=2.0\gamma = 2.0γ=2.0.
3. Display the original and the gamma-corrected images.

Code:

```
def apply_gamma_correction(image, gamma):
    normalized_image = image / 255.0
    gamma_corrected_image = np.power(normalized_image, gamma)
    gamma_corrected_image = np.uint8(gamma_corrected_image * 255)
    return gamma_corrected_image

gamma_0_5 = apply_gamma_correction(gray_image, 0.5)
gamma_1_0 = apply_gamma_correction(gray_image, 1.0)
gamma_2_0 = apply_gamma_correction(gray_image, 2.0)

plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
plt.title('Original Image')
plt.imshow(gray_image, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title('Gamma Correction (gamma = 0.5)')
plt.imshow(gamma_0_5, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title('Gamma Correction (gamma = 1.0)')
plt.imshow(gamma_1_0, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.title('Gamma Correction (gamma = 2.0)')
plt.imshow(gamma_2_0, cmap='gray')
plt.axis('off')

plt.show()
```

Output:

Original Image

Gamma Correction (gamma = 0.5)

Gamma Correction (gamma = 1.0)

Gamma Correction (gamma = 2.0)

Task 3: Log Transform

Objective: Apply log transformation to enhance an image.

Steps:

1. Load a grayscale image
2. Apply log transformation.
3. Display the original and the log-transformed image.

Codee:

```
def apply_log_transformation(image):
    float_image = np.float32(image)
    log_transformed_image = np.log1p(float_image)
    log_transformed_image = cv.normalize(log_transformed_image, None, 0,
255, cv.NORM_MINMAX)
    log_transformed_image = np.uint8(log_transformed_image)
```

```
    return log_transformed_image

log_transformed_image = apply_log_transformation(gray_image)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Gray Image')
plt.imshow(gray_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Log-Transformed Image')
plt.imshow(log_transformed_image, cmap='gray')
plt.axis('off')

plt.show()
```

Output:

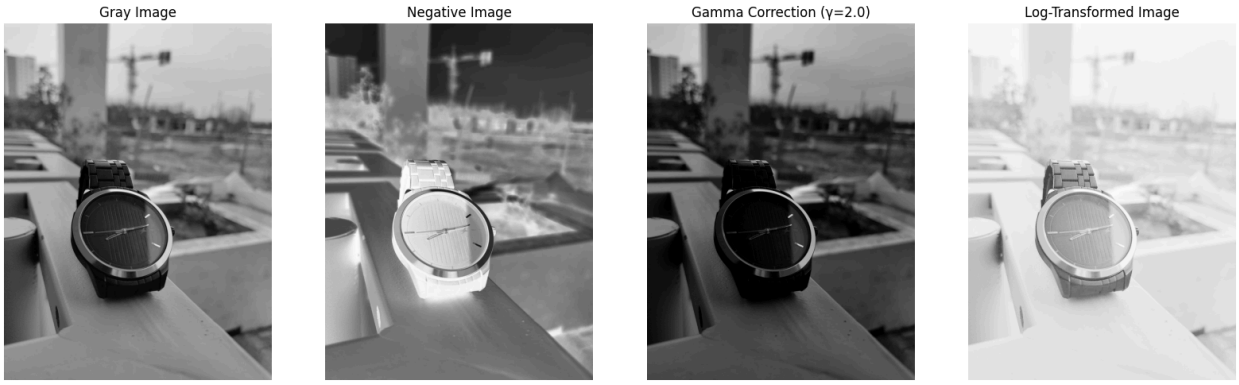
Task 4: Compare Transformations
Objective: Compare the effects of different transformations.
Steps:

1. Load a grayscale image.
2. Apply image negative, gamma correction (γ=2.0\gamma = 2.0γ=2.0), and log transformation.
3. Display the original image alongside the transformed images for comparison.

Code:

```
negative_image = apply_negative_transformation(gray_image)
gamma_2_0_image = apply_gamma_correction(gray_image, 2.0)
log_transformed_image = apply_log_transformation(gray_image)
plt.figure(figsize=(20, 10))

plt.subplot(1, 4, 1)
plt.title('Gray Image')
plt.imshow(gray_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 2)
plt.title('Negative Image')
plt.imshow(negative_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 3)
plt.title('Gamma Correction (γ=2.0)')
plt.imshow(gamma_2_0_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 4)
plt.title('Log-Transformed Image')
plt.imshow(log_transformed_image, cmap='gray')
plt.axis('off')

plt.show()
```

Output:



Task 5: Apply Transformations to Color Images

Objective: Apply the transformations to a color image by processing each channel separately.

Steps:

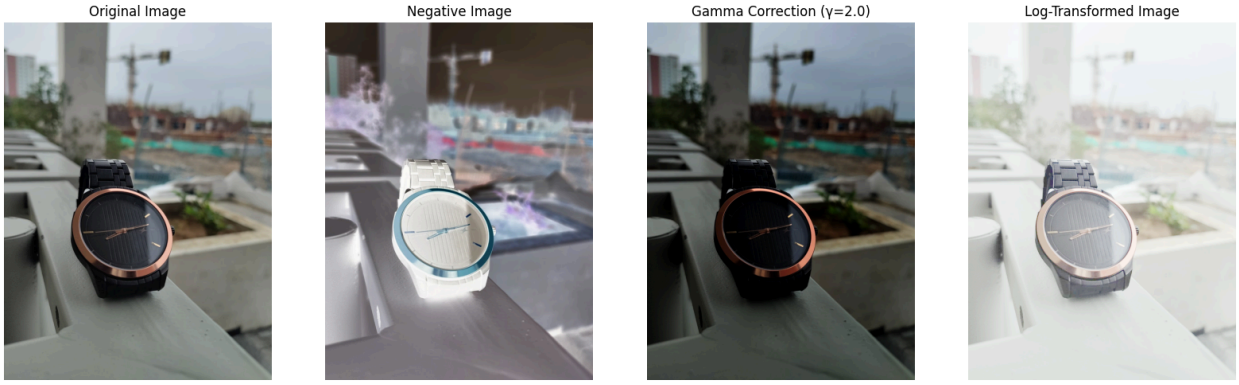
1. Load a color image.
2. Split the image into its R, G, and B channels.
3. Apply image negative, gamma correction, and log transformation to each channel.
4. Merge the channels back together.
5. Display the original and the transformed images.

Code:

```
path = r"watch.jpg"
color_image = cv.imread(path)

B, G, R = cv.split(color_image)



negative_B = apply_negative_transformation(B)
negative_G = apply_negative_transformation(G)
negative_R = apply_negative_transformation(R)
negative_image = cv.merge([negative_B, negative_G, negative_R])

gamma_B = apply_gamma_correction(B, 2.0)
gamma_G = apply_gamma_correction(G, 2.0)
gamma_R = apply_gamma_correction(R, 2.0)
gamma_image = cv.merge([gamma_B, gamma_G, gamma_R])

log_B = apply_log_transformation(B)
log_G = apply_log_transformation(G)
log_R = apply_log_transformation(R)
log_image = cv.merge([log_B, log_G, log_R])
```

```
plt.figure(figsize=(20, 10))

plt.subplot(1, 4, 1)
plt.title('Original Image')
plt.imshow(cv.cvtColor(color_image, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 4, 2)
plt.title('Negative Image')
plt.imshow(cv.cvtColor(negative_image, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 4, 3)
plt.title('Gamma Correction (γ=2.0)')
plt.imshow(cv.cvtColor(gamma_image, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 4, 4)
plt.title('Log-Transformed Image')
plt.imshow(cv.cvtColor(log_image, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.show()
```

Output:



Assignment 3

Task 1: Introduction to Histogram Equalization

Objective: Understand the basics of histogram equalization and its impact on image contrast.

1. Load and Display an Image:
    - Load a grayscale image and display it.
    - Plot the histogram of the original image to show the distribution of pixel intensities.
2. Calculate and Plot Histogram:
    - Calculate the histogram of the image manually.
    - Plot the histogram to visualize the pixel intensity distribution.
3. Calculate Cumulative Distribution Function (CDF):
    - Compute the CDF from the histogram.
    - Normalize the CDF to the range [0, 255].
4. Apply Histogram Equalization:
    - Map the original pixel values to equalized pixel values using the CDF.
    - Display the equalized image and plot its histogram.
5. Compare Results:
    - Compare the original and equalized images.
    - Discuss the differences in visual quality and histogram distributions.

Code:

```
image_path = 'watch.jpg'
image_gray = cv.imread(image_path, cv.IMREAD_GRAYSCALE)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Histogram')
plt.hist(image_gray.ravel(), bins=256, range=[0, 256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

histogram = np.zeros(256, dtype=int)
for pixel_value in image_gray.ravel():
```

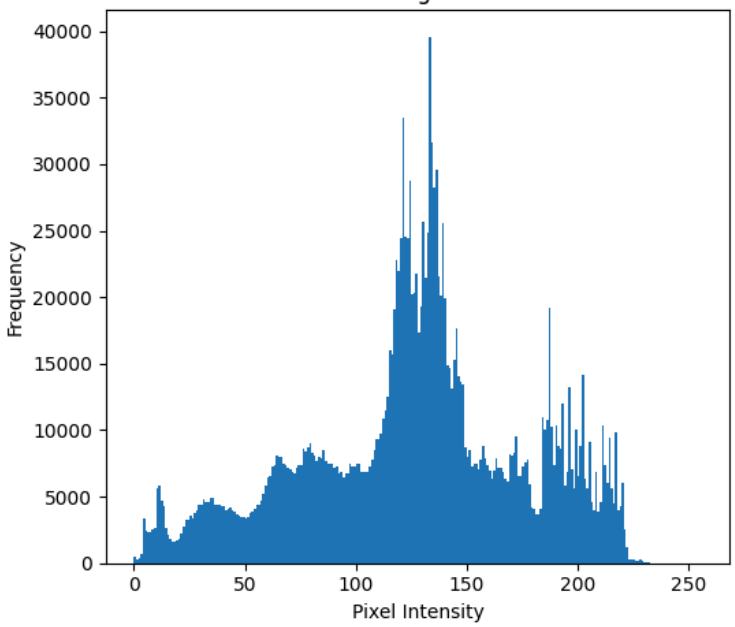
```
    histogram[pixel_value] += 1

plt.figure(figsize=(10, 5))
plt.bar(range(256), histogram)
plt.title('Manually Calculated Histogram')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.show()
cdf = histogram.cumsum()
cdf_normalized = cdf * 255 / cdf[-1]
cdf_normalized = cdf_normalized.astype(np.uint8)

plt.figure(figsize=(10, 5))
plt.plot(cdf_normalized)
plt.title('Cumulative Distribution Function (CDF)')
plt.xlabel('Pixel Intensity')
plt.ylabel('Normalized Cumulative Frequency')
plt.show()
equalized_image = cdf_normalized[image_gray]

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Equalized Image')
plt.imshow(equalized_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Histogram of Equalized Image')
plt.hist(equalized_image.ravel(), bins=256, range=[0, 256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
```

```
plt.imshow(image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Equalized Image')
plt.imshow(equalized_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Histogram of Original Image')
plt.hist(image_gray.ravel(), bins=256, range=[0, 256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.title('Histogram of Equalized Image')
plt.hist(equalized_image.ravel(), bins=256, range=[0, 256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

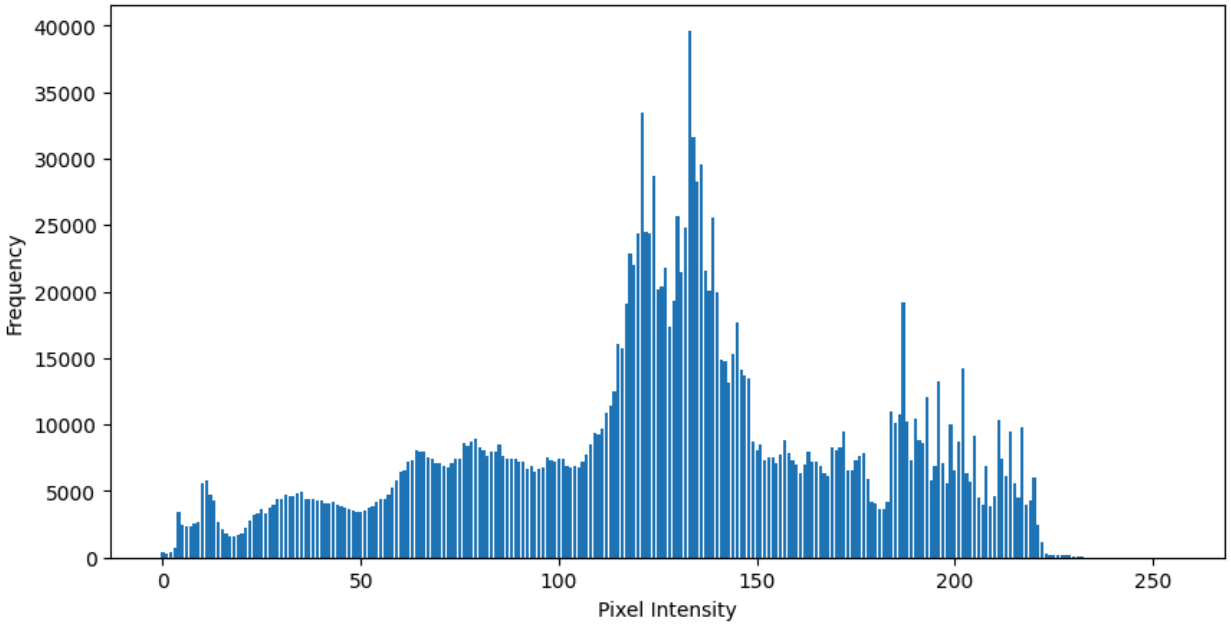
Output:

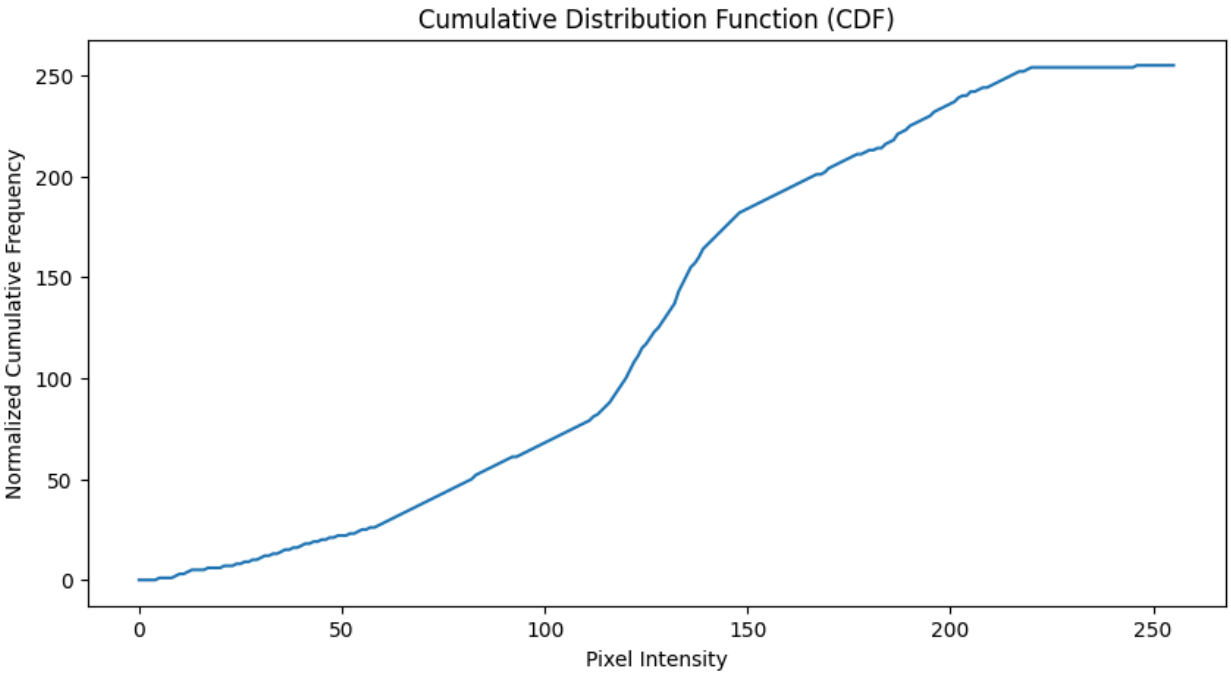
Original Image

Histogram

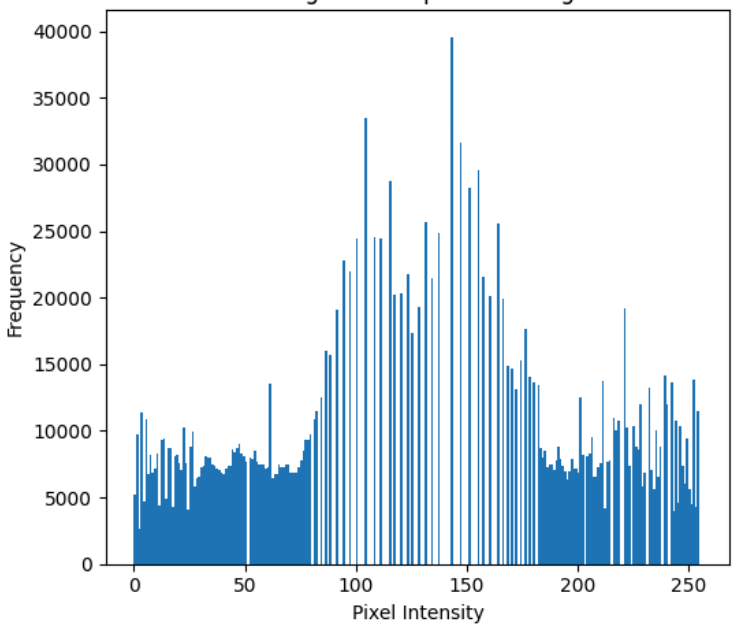


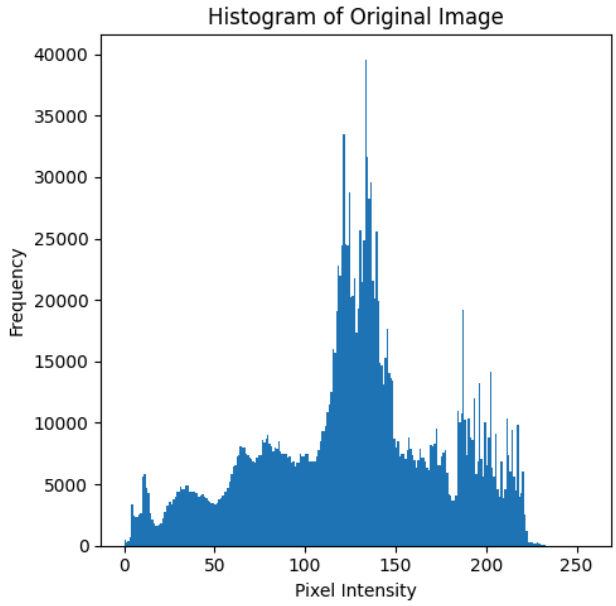
Manually Calculated Histogram

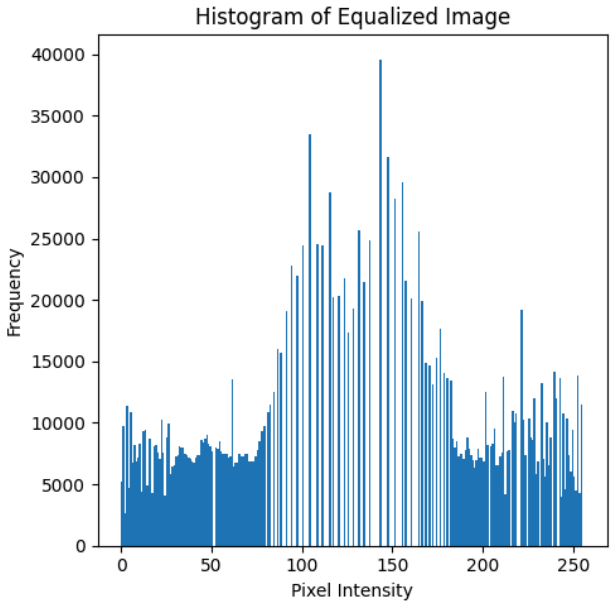




Equalized Image







Task 2: Comparing Histogram Equalization Techniques

Objective: Compare standard histogram equalization with adaptive histogram equalization.

1. Implement Adaptive Histogram Equalization (AHE):
    - Divide the image into smaller regions (tiles).
    - Apply histogram equalization to each tile separately.
    - Combine the tiles to form the final image.

2. Implement Contrast Limited Adaptive Histogram Equalization (CLAHE):
    - Enhance the AHE implementation by limiting the contrast.
    - Apply CLAHE to the image and display the result.
3. Compare Results:
    - Display the original image, standard histogram equalized image, AHE image, and CLAHE image side by side.
    - Discuss the differences in visual quality and histogram distributions.

Code:

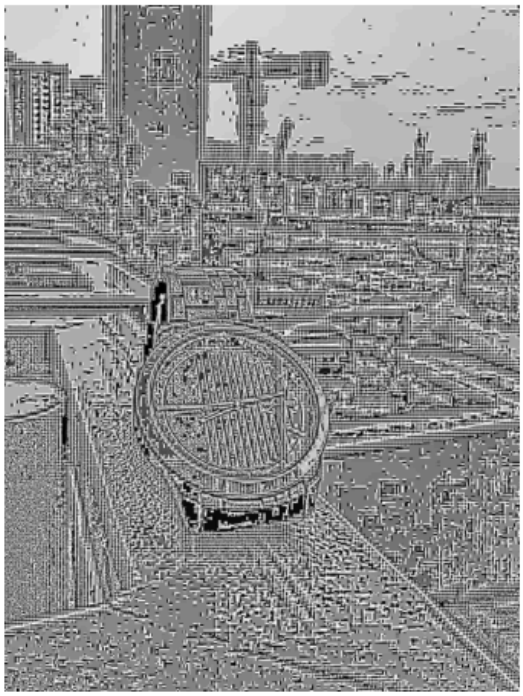
```
image_gray = cv.imread(image_path, cv.IMREAD_GRAYSCALE)

def adaptive_histogram_equalization(img, tile_size=(8, 8)):
    height, width = img.shape
    result = np.zeros_like(img)

    for i in range(0, height, tile_size[0]):
        for j in range(0, width, tile_size[1]):
            tile = img[i:i+tile_size[0], j:j+tile_size[1]]
            tile_equalized = cv.equalizeHist(tile)
            result[i:i+tile_size[0], j:j+tile_size[1]] = tile_equalized

    return result
ahe_image = adaptive_histogram_equalization(image_gray)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('AHE Image')
plt.imshow(ahe_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
clahe_image = clahe.apply(image_gray)
```

```
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('CLAHE Image')
plt.imshow(clahe_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

plt.figure(figsize=(15, 5))

plt.subplot(1, 4, 1)
plt.title('Original Image')
plt.imshow(image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 2)
plt.title('Standard HE Image')
plt.imshow(equalized_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 3)
plt.title('AHE Image')
plt.imshow(ahe_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 4)
plt.title('CLAHE Image')
plt.imshow(clahe_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Output:

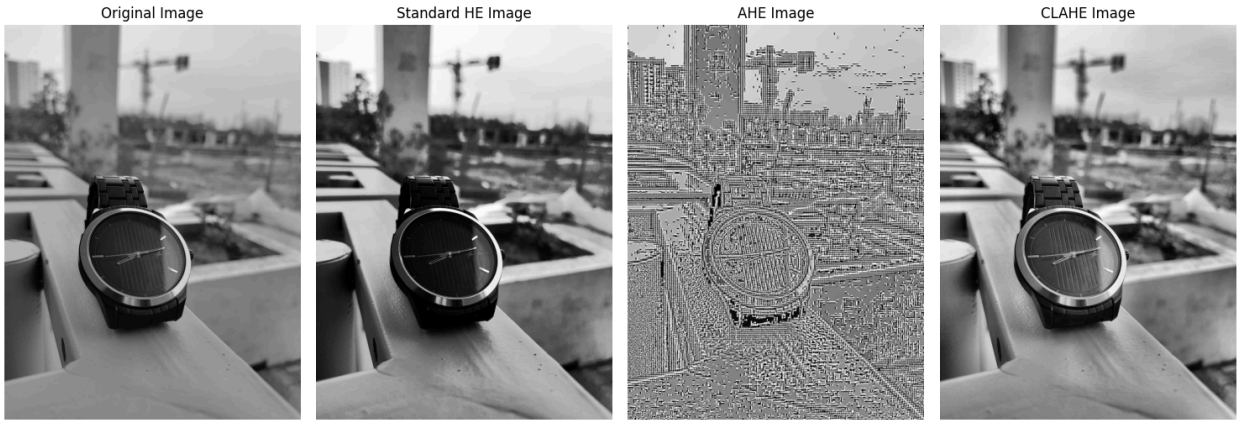
Original Image

AHE Image



Original Image

CLAHE Image



Task 3: Implementing Histogram Equalization on Color Images

Objective: Apply histogram equalization to color images.

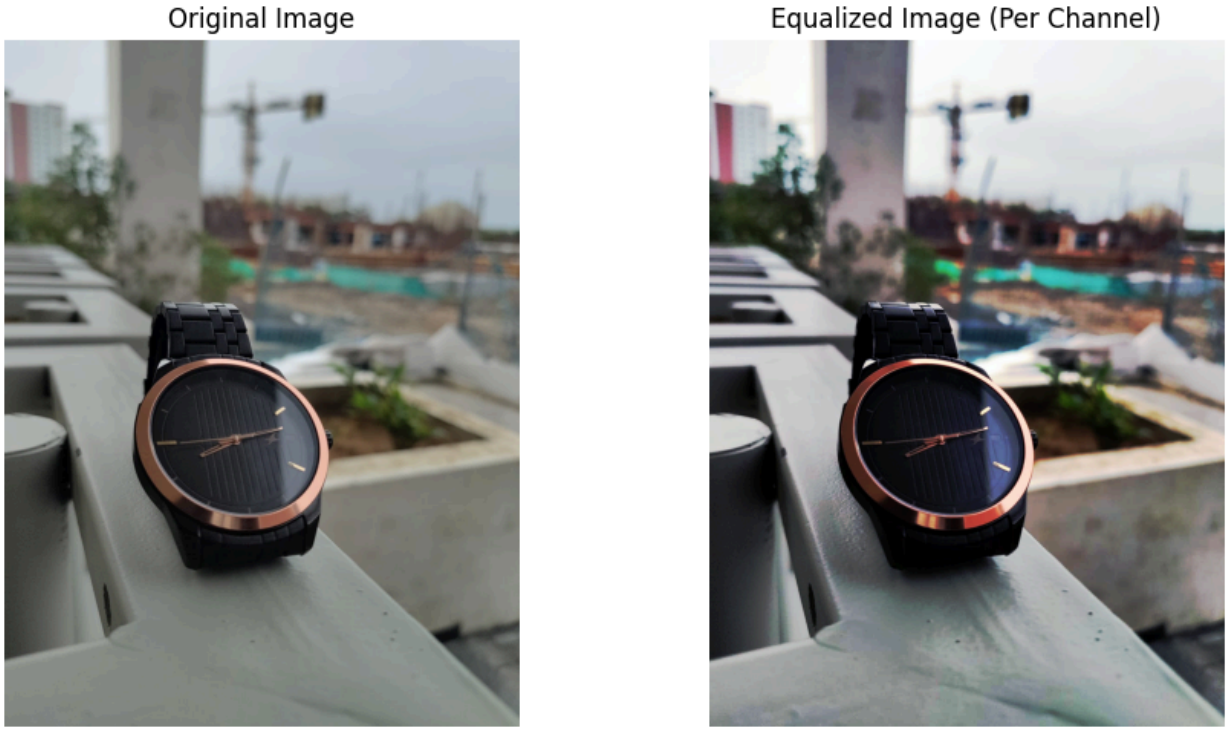
1. Separate Color Channels:
    - Split the color image into its Red, Green, and Blue (RGB) channels.
2. Apply Histogram Equalization to Each Channel:
    - Perform histogram equalization on each color channel separately.
3. Reconstruct the Color Image:
    - Combine the equalized color channels to form the final image.
4. Compare Results:
    - Display the original and equalized color images.
    - Discuss the visual differences and effects of equalization on color images.

Code:

```
color_image = cv.imread(image_path)
B, G, R = cv.split(color_image)
plt.figure(figsize=(15, 5))

plt.subplot(1, 4, 1)
plt.title('Original Image')
plt.imshow(cv.cvtColor(color_image, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 4, 2)
plt.title('Blue Channel')
plt.imshow(B, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 3)
```

```
plt.title('Green Channel')
plt.imshow(G, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 4)
plt.title('Red Channel')
plt.imshow(R, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

equalized_B = cv.equalizeHist(B)
equalized_G = cv.equalizeHist(G)
equalized_R = cv.equalizeHist(R)
equalized_color_image = cv.merge([equalized_B, equalized_G, equalized_R])
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv.cvtColor(color_image, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Equalized Image (Per Channel)')
plt.imshow(cv.cvtColor(equalized_color_image, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.tight_layout()
plt.show()
```

Output:



Task 4: Histogram Equalization in Different Applications

Objective: Explore the use of histogram equalization in various applications.

1. Medical Imaging:
    - Apply histogram equalization to medical images (e.g., X-rays, MRIs).
    - Discuss how equalization can enhance the visibility of important features.
2. Satellite Imagery:
    - Apply histogram equalization to satellite images.
    - Analyze how equalization improves the contrast and details in the images.
3. Document Scanning:

- Apply histogram equalization to scanned documents.
- Evaluate how equalization enhances text readability and clarity.

4. Night Vision:
   - Apply histogram equalization to night vision images.
   - Discuss how equalization improves the visibility of objects in low-light conditions.

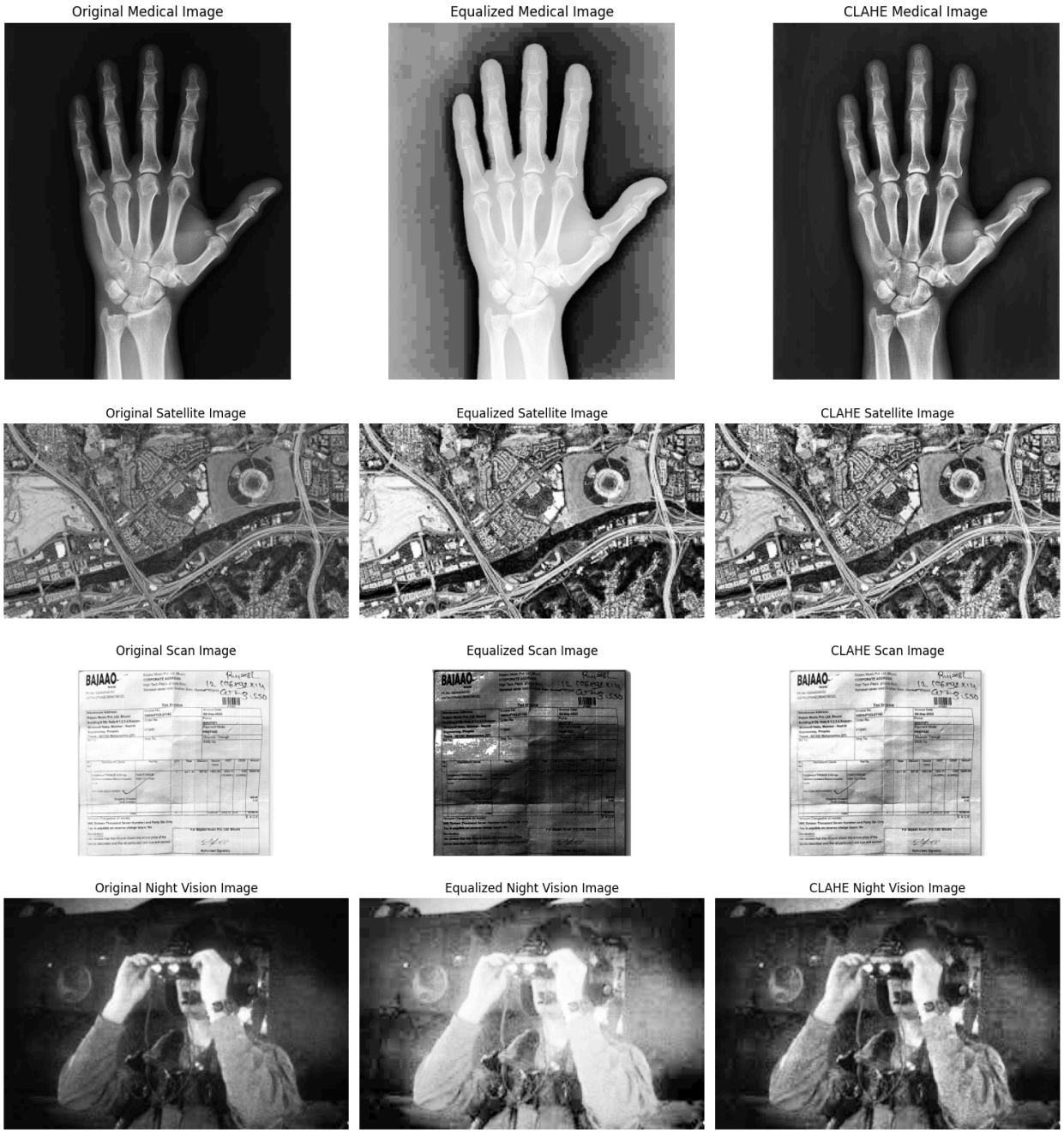
Code:

```
medical_image_path = 'xray.jpg'
medical_image_gray = cv.imread(medical_image_path, cv.IMREAD_GRAYSCALE)

equalized_medical_image = cv.equalizeHist(medical_image_gray)

clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
clahe_medical_image = clahe.apply(medical_image_gray)

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title('Original Medical Image')
plt.imshow(medical_image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Equalized Medical Image')
plt.imshow(equalized_medical_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('CLAHE Medical Image')
plt.imshow(clahe_medical_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

satellite_image_path = 'satelite.jpeg'
satellite_image_gray = cv.imread(satellite_image_path,
cv.IMREAD_GRAYSCALE)
```

```
equalized_satellite_image = cv.equalizeHist(satellite_image_gray)

clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_satellite_image = clahe.apply(satellite_image_gray)

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title('Original Satellite Image')
plt.imshow(satellite_image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Equalized Satellite Image')
plt.imshow(equalized_satellite_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('CLAHE Satellite Image')
plt.imshow(clahe_satellite_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

scan_image_path = 'scan.jpeg'
scan_image_gray = cv.imread(scan_image_path, cv.IMREAD_GRAYSCALE)

equalized_scan_image = cv.equalizeHist(scan_image_gray)

clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_scan_image = clahe.apply(scan_image_gray)

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title('Original Scan Image')
plt.imshow(scan_image_gray, cmap='gray')
plt.axis('off')
```

```
plt.subplot(1, 3, 2)
plt.title('Equalized Scan Image')
plt.imshow(equalized_scan_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('CLAHE Scan Image')
plt.imshow(clahe_scan_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
night_vision_image_path = 'night.jpg'
night_vision_image_gray = cv.imread(night_vision_image_path,
cv.IMREAD_GRAYSCALE)

equalized_night_vision_image = cv.equalizeHist(night_vision_image_gray)

clahe = cv.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_night_vision_image = clahe.apply(night_vision_image_gray)

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title('Original Night Vision Image')
plt.imshow(night_vision_image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Equalized Night Vision Image')
plt.imshow(equalized_night_vision_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('CLAHE Night Vision Image')
plt.imshow(clahe_night_vision_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Output:



Task 5: Histogram Equalization for Low Contrast Images

Objective: Improve the visibility of features in images with low contrast.

1. Select a Low Contrast Image:
    - Choose an image with low contrast or uniform intensity distribution.
2. Apply Histogram Equalization:

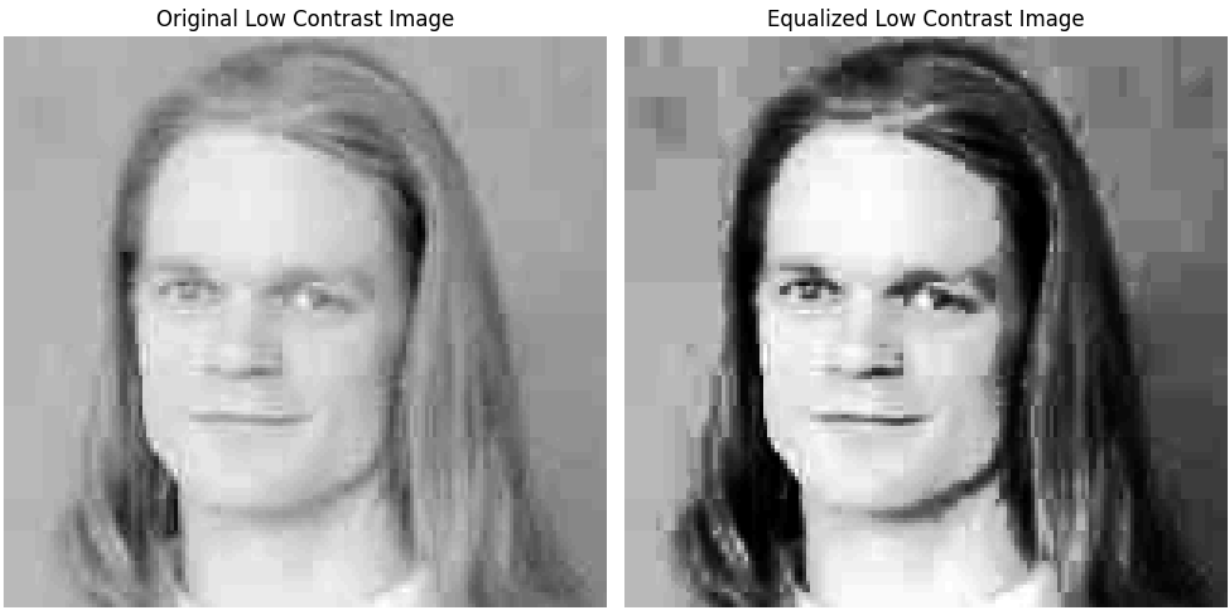
- Perform histogram equalization on the low contrast image.
- Display and compare the results to the original image.

3. Analyze Results:
   - Evaluate the improvement in contrast and feature visibility.
   - Discuss how histogram equalization addresses low contrast issues.

Code:

```
low_contrast_image_path = 'lowcontrast.jpeg'
low_contrast_image_gray = cv.imread(low_contrast_image_path,
cv.IMREAD_GRAYSCALE)

equalized_low_contrast_image = cv.equalizeHist(low_contrast_image_gray)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Low Contrast Image')
plt.imshow(low_contrast_image_gray, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Equalized Low Contrast Image')
plt.imshow(equalized_low_contrast_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Output:



Task 6: Multi-Scale Histogram Equalization

Objective: Explore the effects of applying histogram equalization at different scales.

1. Image Rescaling:
    - Resize the image to different scales (e.g., 50%, 100%, 200%).
2. Apply Histogram Equalization:
    - Perform histogram equalization on the resized images.
3. Compare Results:
    - Compare the equalized images at different scales.
    - Discuss how the scale affects the equalization results and image details.

Code:

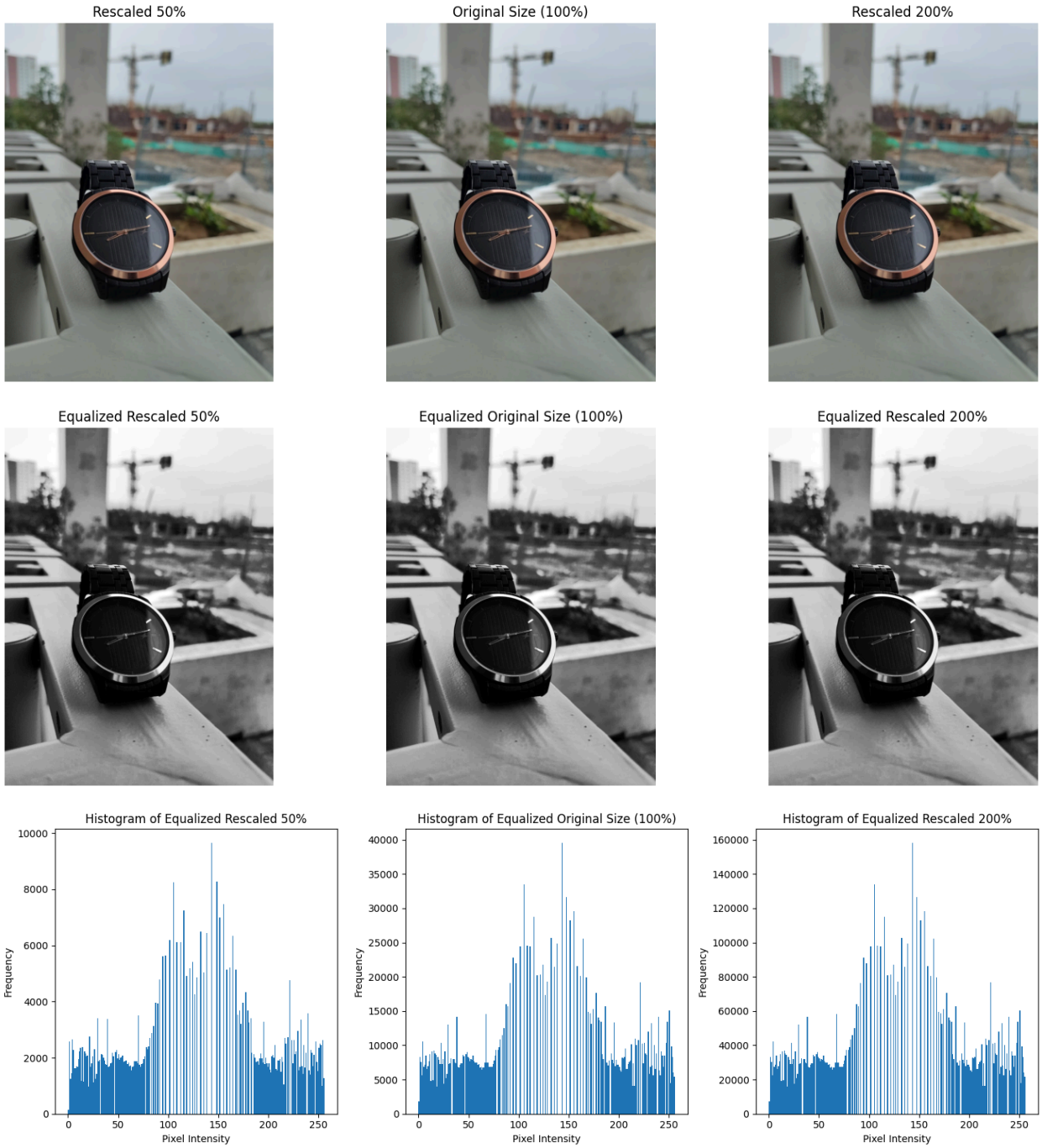
```
image_path = 'watch.jpg'
image = cv.imread(image_path)

scale_percent_50 = 50
width_50 = int(image.shape[1] * scale_percent_50 / 100)
height_50 = int(image.shape[0] * scale_percent_50 / 100)
dim_50 = (width_50, height_50)
resized_50 = cv.resize(image, dim_50, interpolation=cv.INTER_AREA)

scale_percent_100 = 100
width_100 = int(image.shape[1] * scale_percent_100 / 100)
```

```
height_100 = int(image.shape[0] * scale_percent_100 / 100)
dim_100 = (width_100, height_100)
resized_100 = cv.resize(image, dim_100, interpolation=cv.INTER_AREA)

scale_percent_200 = 200
width_200 = int(image.shape[1] * scale_percent_200 / 100)
height_200 = int(image.shape[0] * scale_percent_200 / 100)
dim_200 = (width_200, height_200)
resized_200 = cv.resize(image, dim_200, interpolation=cv.INTER_AREA)

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title('Rescaled 50%')
plt.imshow(cv.cvtColor(resized_50, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Original Size (100%)')
plt.imshow(cv.cvtColor(resized_100, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('Rescaled 200%')
plt.imshow(cv.cvtColor(resized_200, cv.COLOR_BGR2RGB))
plt.axis('off')

plt.tight_layout()
plt.show()


gray_resized_50 = cv.cvtColor(resized_50, cv.COLOR_BGR2GRAY)
equalized_resized_50 = cv.equalizeHist(gray_resized_50)

gray_resized_100 = cv.cvtColor(resized_100, cv.COLOR_BGR2GRAY)
equalized_resized_100 = cv.equalizeHist(gray_resized_100)

gray_resized_200 = cv.cvtColor(resized_200, cv.COLOR_BGR2GRAY)
equalized_resized_200 = cv.equalizeHist(gray_resized_200)
```

```
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title('Equalized Rescaled 50%')
plt.imshow(equalized_resized_50, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Equalized Original Size (100%)')
plt.imshow(equalized_resized_100, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('Equalized Rescaled 200%')
plt.imshow(equalized_resized_200, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title('Histogram of Equalized Rescaled 50%')
plt.hist(equalized_resized_50.ravel(), bins=256, range=[0, 256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.subplot(1, 3, 2)
plt.title('Histogram of Equalized Original Size (100%)')
plt.hist(equalized_resized_100.ravel(), bins=256, range=[0, 256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.subplot(1, 3, 3)
plt.title('Histogram of Equalized Rescaled 200%')
plt.hist(equalized_resized_200.ravel(), bins=256, range=[0, 256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
```

```
plt.tight_layout()
plt.show()
```

Output:



Task 7: Histogram Equalization for Image Enhancement

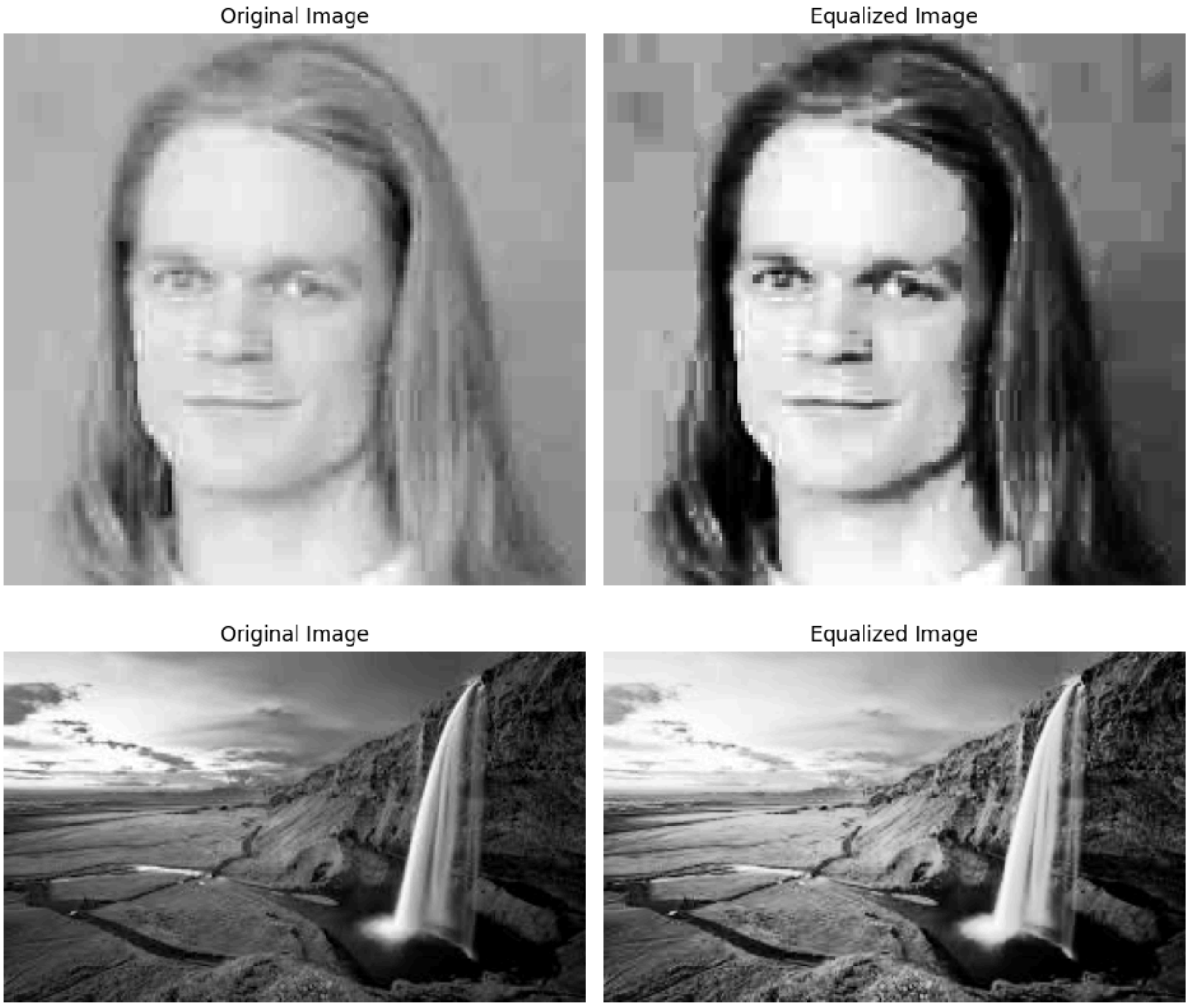
Objective: Use histogram equalization to enhance images for better visual analysis.

1. Select Images:
    - Choose images with varying levels of detail and contrast.
2. Apply Histogram Equalization:
    - Perform histogram equalization on the selected images.
3. Evaluate Enhancement:
    - Compare the equalized images to the originals in terms of detail enhancement and clarity.
    - Discuss how histogram equalization improves visual analysis in different scenarios.

Code:

```
image_paths = ['lowcontrast.jpeg', 'highcontrast.jpeg']

for image_path in image_paths:
    image_gray = cv.imread(image_path, cv.IMREAD_GRAYSCALE)

    equalized_image = cv.equalizeHist(image_gray)

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.title('Original Image')
    plt.imshow(image_gray, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.title('Equalized Image')
    plt.imshow(equalized_image, cmap='gray')
    plt.axis('off')

    plt.tight_layout()
    plt.show()
```

Output:



Conclusion

Each and every function done using opencv affects the images in different ways. The imread function has many formats of reading data and it is possible to get the image in different color formats. There are also different methods to perform equalizations and transforms and different images. The images can be rescaled and can have their quality improved for further processing. The OpenCV commands provides us, the users a versatile way to improve the image and understand it better

Github:
https://github.com/Nihar-Thampi/Machine-Vision.git