Lab 02

Name: Faiza

Student ID: 2021-1-60-089

Course Code: CSE438

Section 02

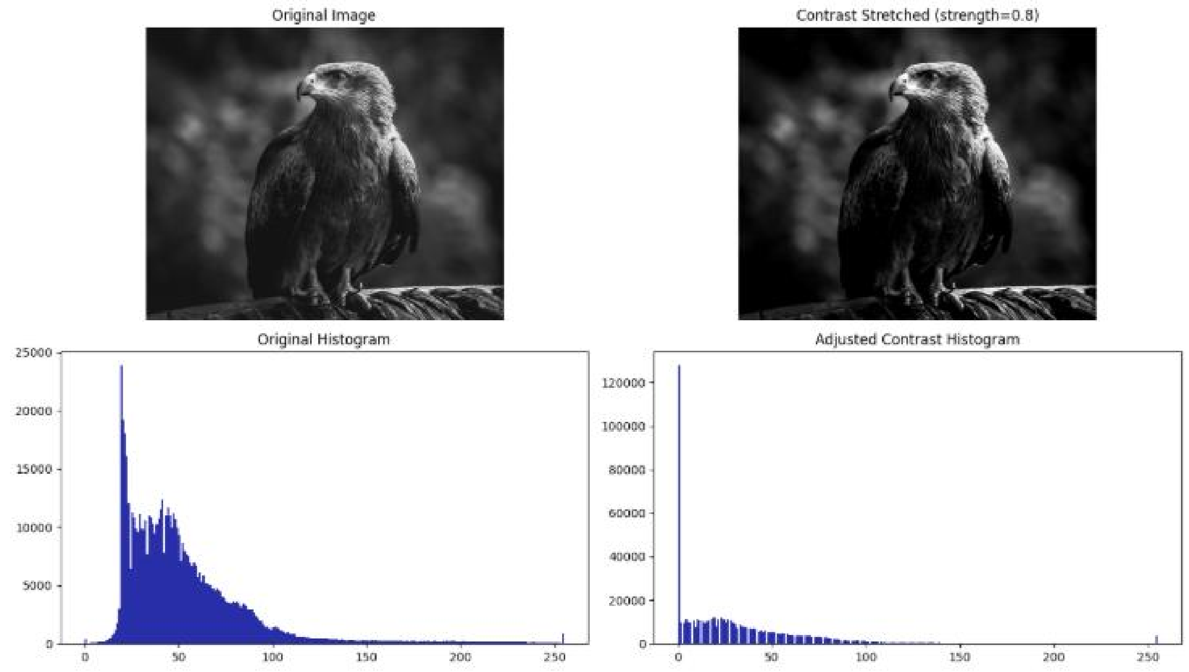
Course name: Digital Image Processing

1. Use contrast stretching on the image.

Code:

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

file_path = r'/kaggle/input/eagle10/eagle1.png'
img = cv2.imread(file_path)

if img is None:
    raise FileNotFoundError("Image not found!")

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Contrast control factor (0 = no change, 1 = full stretch)
contrast_strength = 0.8  # 0.3 → low, 0.8 → medium, 1.0 → full stretch

stretched = np.zeros_like(img_rgb)
for i in range(3):  # R, G, B
    ch = img_rgb[:, :, i]
    min_val, max_val = np.min(ch), np.max(ch)
    # adjust the range
    range_min = min_val + (1 - contrast_strength) * (max_val - min_val) / 2
    range_max = max_val - (1 - contrast_strength) * (max_val - min_val) / 2
    stretched[:, :, i] = np.clip((ch - range_min) / (range_max - range_min) * 255, 0, 255).astype(np.uint8)

# Visualization
plt.figure(figsize=(14, 8))

plt.subplot(2, 2, 1)
plt.imshow(img_rgb)
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(stretched)
plt.title(f'Contrast Stretched (strength={contrast_strength})')
plt.axis('off')

# Histograms
colors = ('r', 'g', 'b')
plt.subplot(2, 2, 3)
for i, col in enumerate(colors):
    plt.hist(img_rgb[:, :, i].ravel(), bins=256, color=col, alpha=0.6)
plt.title('Original Histogram')

plt.subplot(2, 2, 4)
for i, col in enumerate(colors):
    plt.hist(stretched[:, :, i].ravel(), bins=256, color=col, alpha=0.6)
plt.title('Adjusted Contrast Histogram')

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

Output:

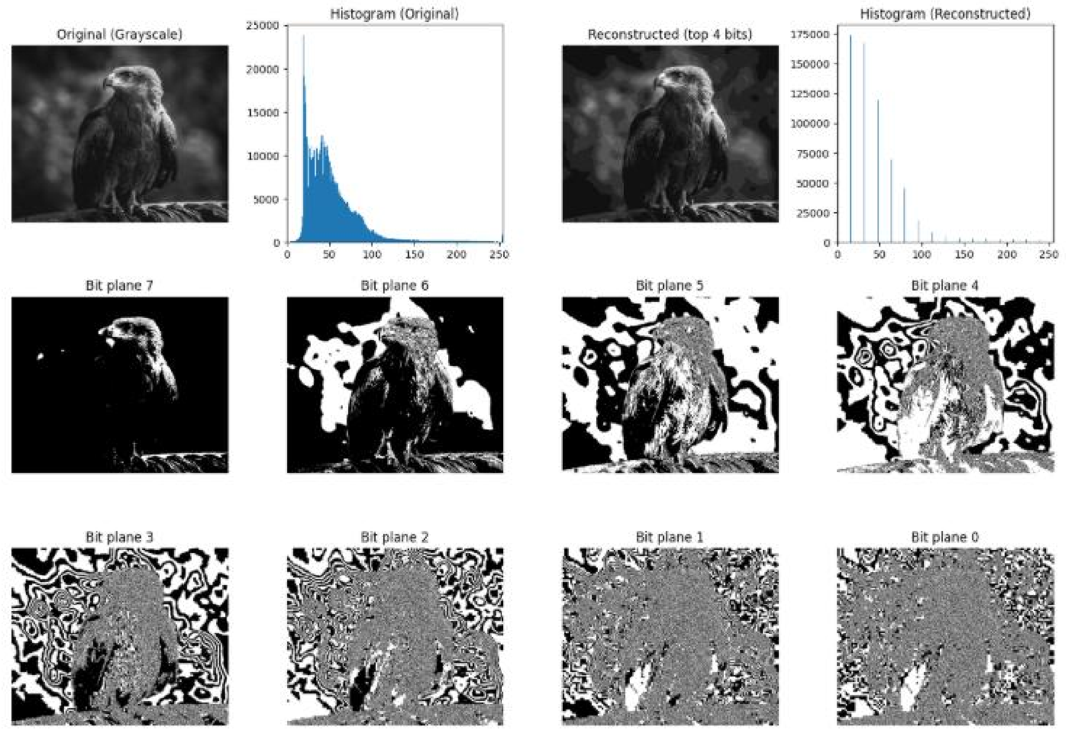


2. Apply bit plane slicing on the image.

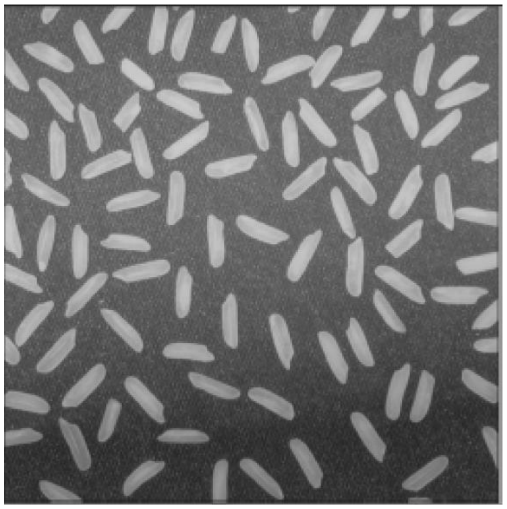
Code:

```
SAVE_OUTPUT = True
OUTPUT_DIR = "/kaggle/working/bitplane_outputs"
RECONSTRUCT_TOP_N = 4   # reconstruct using top N bits (e.g. 4 -> bits 7,6,5,4)
SHOW_COLOR_CHANNELS = False  # If True will also show R/G/B bit-planes separately
# ----------------------------

os.makedirs(OUTPUT_DIR, exist_ok=True)

# Load image (grayscale)
img_bgr = cv2.imread(file_path)
if img_bgr is None:
    raise FileNotFoundError(f"Image not found at {IMG_PATH}. Upload the image or change
IMG_PATH.")

img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
h, w = img_gray.shape

# Function to extract bit planes (0 = LSB, 7 = MSB)
def extract_bit_planes(gray_img):
    bit_planes = []
    for bit in range(8):
        # shift right by `bit` and mask LSB
        plane = ((gray_img >> bit) & 1).astype(np.uint8) * 255
        bit_planes.append(plane)
    return bit_planes

bit_planes = extract_bit_planes(img_gray)

# Reconstruct using top N bits (most significant)
def reconstruct_from_top_n(bit_planes, top_n):
    # top_n=1 -> only MSB (bit 7), top_n=8 -> full image
    out = np.zeros_like(bit_planes[0], dtype=np.uint8)
    start_bit = 8 - top_n
    for bit in range(start_bit, 8):
        out = out + ((bit_planes[bit] // 255) << bit).astype(np.uint8)
    return out

reconstructed = reconstruct_from_top_n(bit_planes, RECONSTRUCT_TOP_N)

# Visualization: Original, reconstructed, histograms, and bit planes grid
plt.figure(figsize=(14, 10))

# Original image
plt.subplot(3, 4, 1)
plt.imshow(img_gray, cmap='gray')
plt.title('Original (Grayscale)')
plt.axis('off')

# Histogram of original
plt.subplot(3, 4, 2)
plt.hist(img_gray.ravel(), bins=256)
plt.title('Histogram (Original)')
plt.xlim([0,255])

# Reconstructed image
plt.subplot(3, 4, 3)
plt.imshow(reconstructed, cmap='gray')
plt.title(f'Reconstructed (top {RECONSTRUCT_TOP_N} bits)')
plt.axis('off')

# Histogram of reconstructed
plt.subplot(3, 4, 4)
plt.hist(reconstructed.ravel(), bins=256)
plt.title('Histogram (Reconstructed)')
plt.xlim([0,255])

# Bit-plane panels (bits 7 -> 0)
for i in range(8):
    plt.subplot(3, 4, 5 + i)
    plt.imshow(bit_planes[7 - i], cmap='gray')   # show MSB first
    plt.title(f'Bit plane {7 - i}')
    plt.axis('off')

plt.tight_layout()
plt.show()

# Save bit planes and reconstructions if requested
if SAVE_OUTPUT:
    cv2.imwrite(os.path.join(OUTPUT_DIR, "original_gray.png"), img_gray)
    cv2.imwrite(os.path.join(OUTPUT_DIR, f"reconstructed_top_{RECONSTRUCT_TOP_N}.png"),
reconstructed)
    for b, plane in enumerate(bit_planes):
        cv2.imwrite(os.path.join(OUTPUT_DIR, f"bitplane_{b}.png"), plane)
    print(f"Saved outputs to: {OUTPUT_DIR}")

# -------- Optional: Show color-channel bit-planes ----------
if SHOW_COLOR_CHANNELS:
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    channels = ('R','G','B')
    for c_idx, ch_name in enumerate(channels):
        ch = img_rgb[:,:,c_idx]
        planes = extract_bit_planes(ch)
        fig, axes = plt.subplots(2,4, figsize=(12,6))
        fig.suptitle(f'{ch_name} channel bit planes (MSB -> LSB)')
        for i in range(8):
            ax = axes[i//4, i%4]
            ax.imshow(planes[7-i], cmap='gray')
            ax.set_title(f'bit {7-i}')
            ax.axis('off')
        plt.tight_layout()
        plt.show()
        # optionally save channel bit planes
        for b, plane in enumerate(planes):
            fname = f"{ch_name}_bit_{b}.png"
            cv2.imwrite(os.path.join(OUTPUT_DIR, fname), plane)
    print("Saved color-channel bit planes (if enabled).")
```

Output:



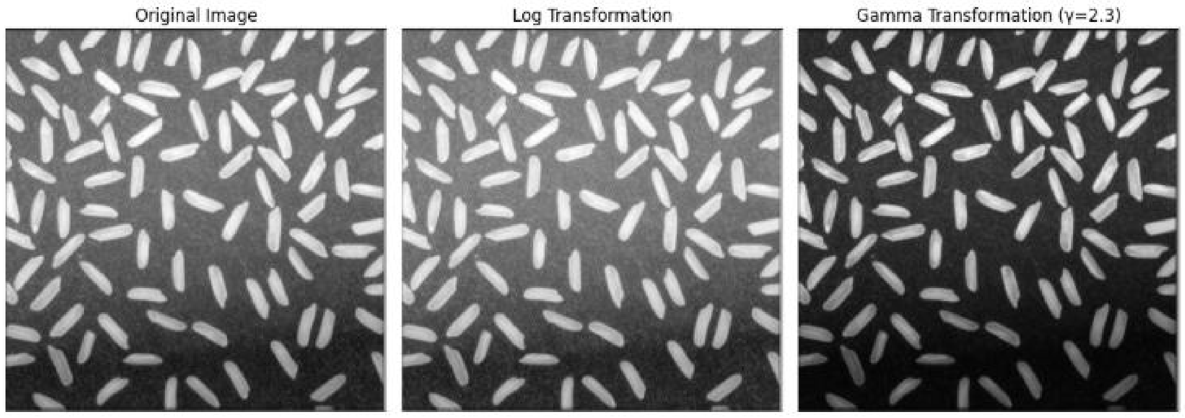
3. Change the contrast of the image using Logarithmic Transformation and Power-law Transformation.



Code:

```
img = cv2.imread('/kaggle/input/eagle10/eaagle3.png', cv2.IMREAD_GRAYSCALE)

# Normalize image to range 0–1 for transformations
img_norm = img / 255.0

# 2. Logarithmic Transformation
c_log = 1 / np.log(1 + np.max(img_norm))  # scaling constant
log_transformed = c_log * np.log(1 + img_norm)
log_transformed = np.uint8(255 * log_transformed)  # scale back to 0–255

# 3. Power-law (Gamma) Transformation
gamma = 2.3
gamma_transformed = np.power(img_norm, gamma)
gamma_transformed = np.uint8(255 * gamma_transformed)  # scale back to 0–255

plt.figure(figsize=(12, 6))

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

plt.subplot(1, 3, 2)
plt.imshow(log_transformed, cmap='gray')
plt.title('Log Transformation')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(gamma_transformed, cmap='gray')
plt.title(f'Gamma Transformation (γ={gamma})')
plt.axis('off')
plt.tight_layout()
plt.show()
```

Output:

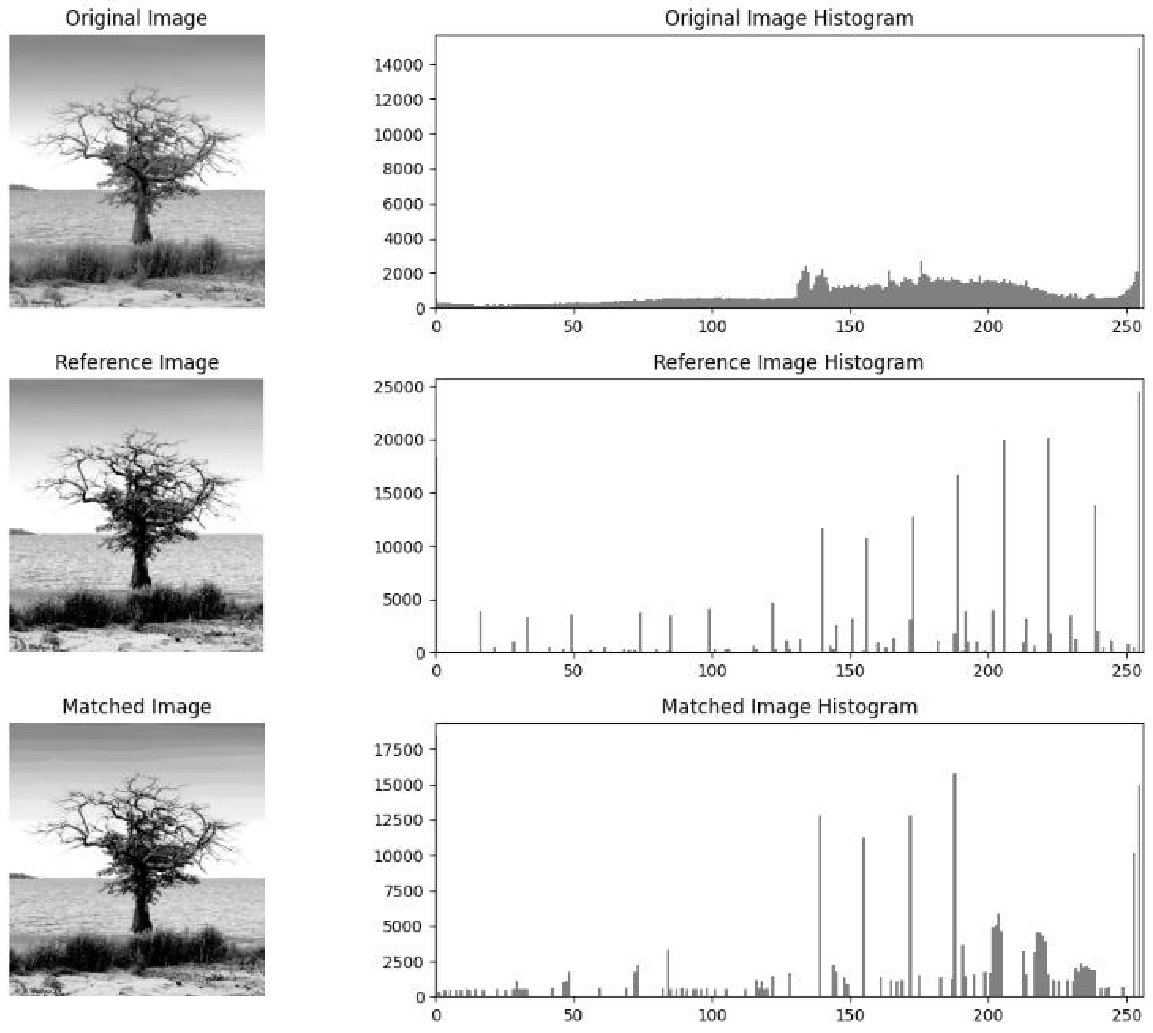


4. Adjust the histogram of the following image to match the reference image using histogram matching. Show the histogram of original, reference, and output images.

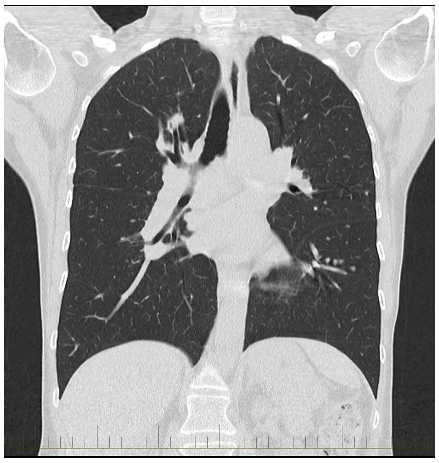
Code:

```
from skimage.exposure import match_histograms

source = cv2.imread('/kaggle/input/eagle10/eagle4.png', cv2.IMREAD_GRAYSCALE)
reference = cv2.imread('/kaggle/input/eagle10/eagle5.png', cv2.IMREAD_GRAYSCALE)

# Perform histogram matching
matched = match_histograms(source, reference, channel_axis=None).astype(np.uint8)

# Plot helper function
def plot_image_and_histogram(img, title, subplot_idx):
    plt.subplot(3, 2, subplot_idx)
    plt.imshow(img, cmap='gray')
    plt.title(title)
    plt.axis('off')

    plt.subplot(3, 2, subplot_idx+1)
    plt.hist(img.ravel(), bins=256, color='gray')
    plt.title(f'{title} Histogram')
    plt.xlim([0, 256])

# Plot all images and their histograms
plt.figure(figsize=(12, 9))

plot_image_and_histogram(source, "Original Image", 1)
plot_image_and_histogram(reference, "Reference Image", 3)
plot_image_and_histogram(matched, "Matched Image", 5)

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

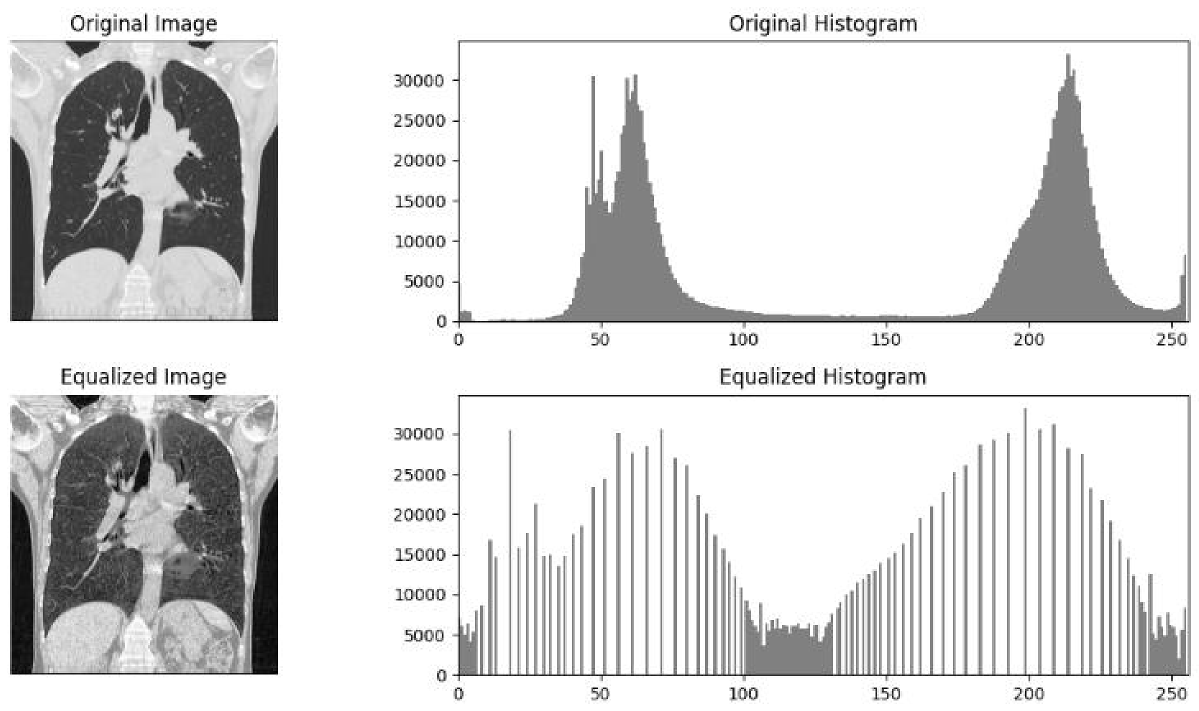
Output:



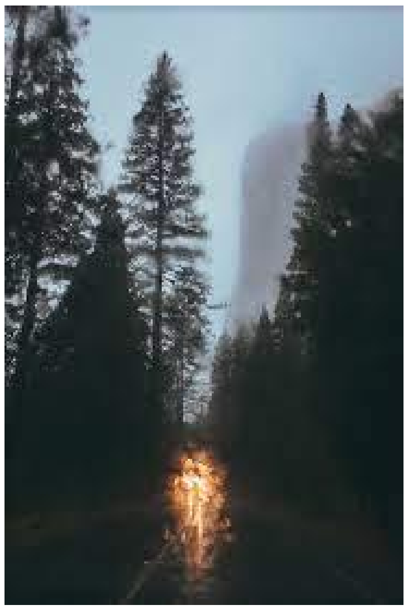
5. Change the contrast of the image using histogram equalization. Show the histogram of both input and output images.



```
img = cv2.imread('/kaggle/input/eagle10/eagle6.png', cv2.IMREAD_GRAYSCALE)

# Histogram Equalization
equalized_img = cv2.equalizeHist(img)

plt.figure(figsize=(12, 6))

# Original image and histogram
plt.subplot(2, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.hist(img.ravel(), bins=256, color='gray')
plt.title('Original Histogram')
plt.xlim([0, 256])

# Equalized image and histogram
plt.subplot(2, 2, 3)
plt.imshow(equalized_img, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.hist(equalized_img.ravel(), bins=256, color='gray')
plt.title('Equalized Histogram')
plt.xlim([0, 256])

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



6. Brighten the following image



Code:

```
img_bgr = cv2.imread('/kaggle/input/eagle10/eagle7.png')
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

# Brightness offset (try different values like 30, 50, 100)
brightness_offset = 50
```

```
# Convert to float to avoid overflow during addition
brightened = img_rgb.astype(np.int16) + brightness_offset

# Clip values to keep them in the valid range [0, 255]
brightened = np.clip(brightened, 0, 255).astype(np.uint8)

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

plt.subplot(1, 2, 1)
plt.imshow(img_rgb)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(brightened)
plt.title(f'Brightened Image (+{brightness_offset})')
plt.axis('off')

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

Output: