1. Write a Python code to read and write a color/gray image.

```
import cv2

def read_and_write_image(input_path, output_path, is_color=True):
    # Read the image
    if is_color:
        image = cv2.imread(input_path, cv2.IMREAD_COLOR)
    else:
        image = cv2.imread(input_path, cv2.IMREAD_GRAYSCALE)

    # Check if the image was successfully read
    if image is None:
        print(f"Failed to read the image from {input_path}")
        return

    # Write the image to the output path
    success = cv2.imwrite(output_path, image)

    if success:
        print(f"Image successfully written to {output_path}")
    else:
        print(f"Failed to write the image to {output_path}")


input_image_path = 'apple.jpeg'
output_image_path_color = 'output.jpg'

read_and_write_image(input_image_path, output_image_path_color, is_color=True)

read_and_write_image(input_image_path,output_image_path_color, is_color=False)
```

Output:

Color

Gray

2. Write a Python code for bit plane slicing of an image.

```
import cv2
import numpy as np
def bit_plane_slicing(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print(f"Failed to read the image from {image_path}")
        return
    rows, cols = image.shape
    bit_planes = []
    # Extract each bit plane
    for i in range(8):
        bit_plane = np.zeros((rows, cols), dtype=np.uint8)
        bit_plane[:, :] = (image[:, :] & (1 << i)) >> i
        bit_plane *= 255
        bit_planes.append(bit_plane)
        output_path = f'bit_plane_{i}.png'
        cv2.imwrite(output_path, bit_plane)
        print(f"Bit plane {i} saved as {output_path}")
    return bit_planes
# Example usage
input_image_path = 'apple.jpeg'
bit_planes = bit_plane_slicing(input_image_path)
for i, bit_plane in enumerate(bit_planes):
    cv2.imshow(f'Bit Plane {i}', bit_plane)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:

3. Write a Python code for intensity level slicing of an image.

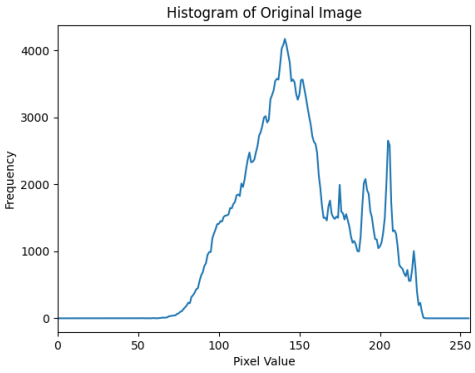
```
import cv2
import numpy as np
def intensity_level_slicing(image_path, lower_bound, upper_bound,
high_intensity=255, low_intensity=0):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print(f"Failed to read the image from {image_path}")
        return None
    sliced_image = np.zeros_like(image)
    sliced_image[(image >= lower_bound) & (image <= upper_bound)] =
high_intensity
    sliced_image[(image < lower_bound) | (image > upper_bound)] =
low_intensity
    return sliced_image
input_image_path = 'apple.jpeg'
output_image_path = 'sliced_image.jpg'
lower_bound = 100
upper_bound = 200
sliced_image = intensity_level_slicing(input_image_path, lower_bound,
upper_bound)
if sliced_image is not None:
    cv2.imwrite(output_image_path, sliced_image)
    print(f"Intensity level sliced image saved as {output_image_path}")
    input_image = cv2.imread(input_image_path, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('Original Image', input_image)
    cv2.imshow('Intensity Level Sliced Image', sliced_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

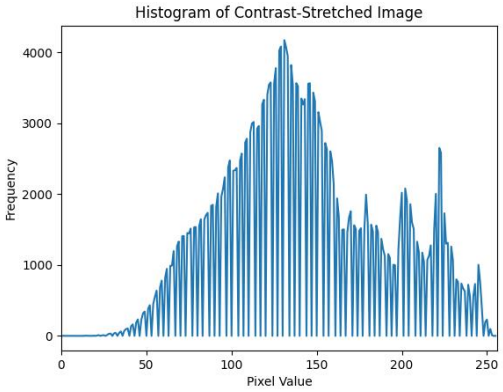
Output:

4. Write a Python code for contrast stretching of an image.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def contrast_stretching(image):
    min_val = np.min(image)
    max_val = np.max(image)
    stretched_image =(image-min_val)/(max_val-min_val)*255).astype(np.uint8)
    return stretched_image

def plot_histogram(image, title):
    hist, bins = np.histogram(image.flatten(), 256, [0, 256])
    plt.figure()
    plt.title(title)
    plt.xlabel("Pixel Value")
    plt.ylabel("Frequency")
    plt.plot(hist)
    plt.xlim([0, 256])
    plt.show()
input_image_path = 'forest.png'
output_image_path = 'path/to/your/output/contrast_stretched_image.jpg'
image = cv2.imread(input_image_path, cv2.IMREAD_GRAYSCALE)
if image is None:
    print(f"Failed to read the image from {input_image_path}")
else:
    plot_histogram(image, "Histogram of Original Image")
    stretched_image = contrast_stretching(image)
    plot_histogram(stretched_image, "Histogram of Contrast-Stretched Image")
    cv2.imwrite(output_image_path, stretched_image)
    print(f"Contrast-stretched image saved as {output_image_path}")
    cv2.imshow('Original Image', image)
    cv2.imshow('Contrast-Stretched Image', stretched_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Output:





5. Write a Python code to generate a negative image from a gray image.

```
import cv2
import numpy as np
def generate_negative_image(image_path, output_path):
    # Read the image in grayscale
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print(f"Failed to read the image from {image_path}")
        return
    negative_image = 255 - image

    cv2.imwrite(output_path, negative_image)
    print(f"Negative image saved as {output_path}")
    cv2.imshow('Original Image', image)
    cv2.imshow('Negative Image', negative_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage
input_image_path = 'apple_gray.jpeg'
output_image_path = 'negative_image.jpg'

generate_negative_image(input_image_path, output_image_path)
```

Output:

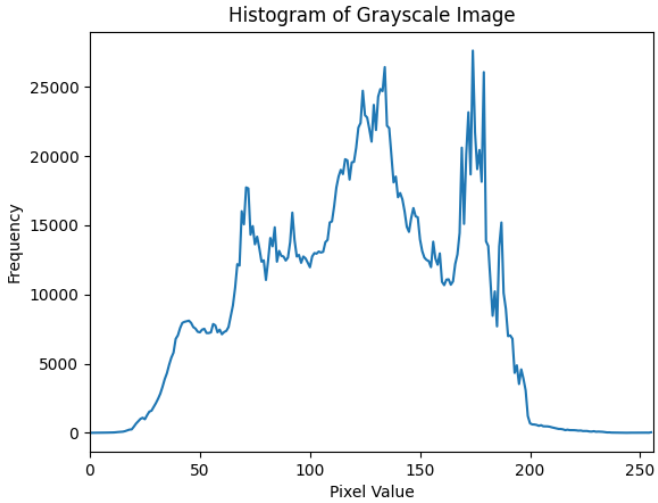
Original

Negative

6. Write a Python code to represent the histogram of an image.

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

def plot_histogram(image_path):
    # Read the image in grayscale
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    if image is None:
        print(f"Failed to read the image from {image_path}")
        return

    # Calculate the histogram
    hist, bins = np.histogram(image.flatten(), 256, [0, 256])

    # Plot the histogram
    plt.figure()
    plt.title("Histogram of Grayscale Image")
    plt.xlabel("Pixel Value")
    plt.ylabel("Frequency")
    plt.plot(hist)
    plt.xlim([0, 256])
    plt.show()

# Example usage
input_image_path = 'apple.jpeg'
plot_histogram(input_image_path)
```

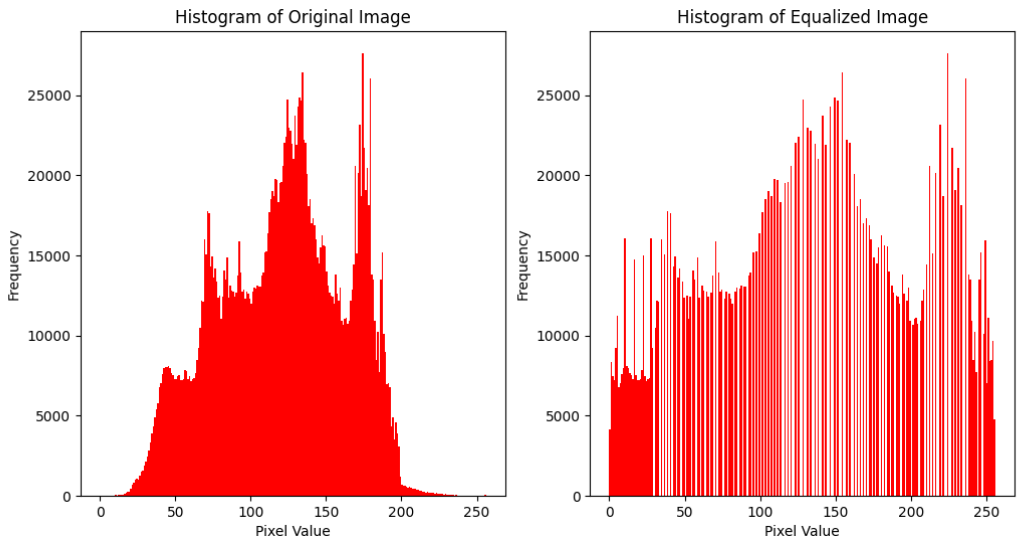
Output:



7. Write a Python code for histogram equalization of an image.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def histogram_equalization(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    equalized_image = cv2.equalizeHist(image)
    cv2.imwrite(output_path, equalized_image)
    print(f"Equalized image saved as {output_path}")
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.hist(image.flatten(), 256, [0, 256], color='r')
    plt.title('Histogram of Original Image')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.subplot(1, 2, 2)
    plt.hist(equalized_image.flatten(), 256, [0, 256], color='r')
    plt.title('Histogram of Equalized Image')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.show()
    cv2.imshow('Original Image', image)
    cv2.imshow('Equalized Image', equalized_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
# Example usage
input_image_path = 'path/to/your/input/apple.jpeg'
output_image_path = 'path/to/your/output/equalized_image.jpg'

histogram_equalization(input_image_path, output_image_path)
```

Output:



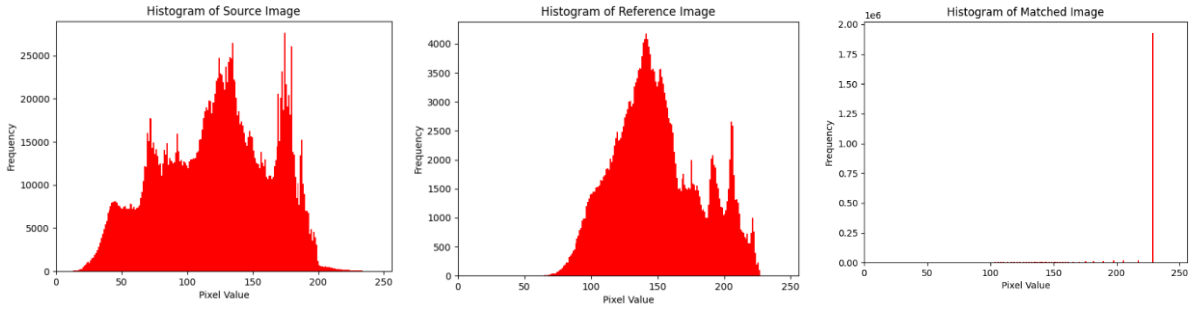
8. Write a Python code to perform the Histogram matching of an image with respect to a reference image.

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

def plot_histogram(image, title):
    plt.figure()
    plt.title(title)
    plt.xlabel("Pixel Value")
    plt.ylabel("Frequency")
    plt.hist(image.ravel(), bins=256, range=[0, 256], color='r')
    plt.xlim([0, 256])
    plt.show()

def calculate_cdf(hist):
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()
    return cdf_normalized

def histogram_matching(source, template):
    src_hist, bins = np.histogram(source.flatten(), 256, [0, 256])
    src_cdf = calculate_cdf(src_hist)

    # Compute the histogram and cumulative distribution function (CDF) of the
template image
    tmpl_hist, bins = np.histogram(template.flatten(), 256, [0, 256])
    tmpl_cdf = calculate_cdf(tmpl_hist)
    lookup_table = np.zeros(256)
    tmpl_cdf_min = tmpl_cdf[tmpl_hist > 0].min()  # Avoid division by zero
    src_cdf_min = src_cdf[src_hist > 0].min()     # Avoid division by zero

    for src_pixel_val in range(256):
        src_val = src_cdf[src_pixel_val]
        closest_val = np.argmin(np.abs(tmpl_cdf - src_val))
        lookup_table[src_pixel_val] = closest_val
    matched = lookup_table[source]

    return matched

def histogram_matching_cv2(source_image_path, reference_image_path,
output_image_path):
    source_image = cv2.imread(source_image_path, cv2.IMREAD_GRAYSCALE)
    reference_image = cv2.imread(reference_image_path, cv2.IMREAD_GRAYSCALE)
    if source_image is None or reference_image is None:
        print(f"Failed to read the source or reference image")
        return
```

```
    matched_image = histogram_matching(source_image,
reference_image).astype(np.uint8)
    cv2.imwrite(output_image_path, matched_image)
    print(f"Matched image saved as {output_image_path}")
    plot_histogram(source_image, "Histogram of Source Image")
    plot_histogram(reference_image, "Histogram of Reference Image")
    plot_histogram(matched_image, "Histogram of Matched Image")
    cv2.imshow('Source Image', source_image)
    cv2.imshow('Reference Image', reference_image)
    cv2.imshow('Matched Image', matched_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

source_image_path = 'apple.jpeg'
reference_image_path = 'forest.png'
output_image_path = 'path/to/your/output/matched_image.jpg'

histogram_matching_cv2(source_image_path, reference_image_path,
output_image_path)
```

Output:

Histograms



Source Images: Reference Image Matched Image

9. Write a Python code for implementing Log transformation of an image.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def log_transformation(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print(f"Failed to read the image from {image_path}")
        return
    c = 255 / np.log(1 + np.max(image))
    log_image = c * (np.log(1 + image))
    log_image = np.array(log_image, dtype=np.uint8)
    cv2.imwrite(output_path, log_image)
    print(f"Log transformed image saved as {output_path}")
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(log_image, cmap='gray')
    plt.title('Log Transformed Image')
    plt.axis('off')
    plt.show()
    cv2.imshow('Original Image', image)
    cv2.imshow('Log Transformed Image', log_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
input_image_path = 'forest.png'
output_image_path = 'log_transformed_image.jpg'

log_transformation(input_image_path, output_image_path)
```

Output:

10. Write a Python code for power law transformation of an image.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def power_law_transformation(image_path, output_path, gamma):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print(f"Failed to read the image from {image_path}")
        return
    normalized_image = image / 255.0
    c = 1.0  # Scaling constant
    transformed_image = c * (normalized_image ** gamma)
    transformed_image = np.uint8(transformed_image * 255)
    cv2.imwrite(output_path, transformed_image)
    print(f"Power-law transformed image saved as {output_path}")
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(transformed_image, cmap='gray')
    plt.title(f'Power-law Transformed Image (gamma={gamma})')
    plt.axis('off')
    plt.show()
    cv2.imshow('Original Image', image)
    cv2.imshow('Power-law Transformed Image', transformed_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
input_image_path = 'apple.jpeg'
output_image_path = 'power_law_transformed_image.jpg'
gamma_value = 2.0  # Example gamma value
power_law_transformation(input_image_path, output_image_path, gamma_value)
```

Output:

11. Write a Python code for identify the edge of an image using Sobel operator.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def sobel_edge_detection(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    sobel_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
    sobel_magnitude = np.uint8(np.absolute(sobel_magnitude))
    cv2.imwrite(output_path, sobel_magnitude)
    print(f"Sobel edge-detected image saved as {output_path}")
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(sobel_magnitude, cmap='gray')
    plt.title('Sobel Edge Detected Image')
    plt.axis('off')
    plt.show()
    cv2.imshow('Original Image', image)
    cv2.imshow('Sobel Edge Detected Image', sobel_magnitude)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
input_image_path = 'apple.jpeg'
output_image_path = 'sobel_edge_detected_image.jpg'

sobel_edge_detection(input_image_path, output_image_path)
```

Output:

12. Write a Python code for identify edge of an image using Canny Edge Detector.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def canny_edge_detection(image_path, output_path, low_threshold,
high_threshold):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    edges = cv2.Canny(image, low_threshold, high_threshold)
    cv2.imwrite(output_path, edges)
    print(f"Canny edge-detected image saved as {output_path}")
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(edges, cmap='gray')
    plt.title('Canny Edge Detected Image')
    plt.axis('off')
    plt.show()
    cv2.imshow('Original Image', image)
    cv2.imshow('Canny Edge Detected Image', edges)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
input_image_path = 'apple.jpeg'
output_image_path = 'canny_edge_detected_image.jpg'
low_threshold = 50  # Lower bound for the hysteresis thresholding
high_threshold = 150  # Upper bound for the hysteresis thresholding

canny_edge_detection(input_image_path, output_image_path, low_threshold,
high_threshold)
```

Output:

13. Write a Python code for identify edge of an image using Roberts Edge Detection Operator.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def roberts_cross_edge_detection(image_path, output_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    kernel_x = np.array([[1, 0], [0, -1]], dtype=int)
    kernel_y = np.array([[0, 1], [-1, 0]], dtype=int)
    roberts_x = cv2.filter2D(image, cv2.CV_64F, kernel_x)
    roberts_y = cv2.filter2D(image, cv2.CV_64F, kernel_y)
    magnitude = np.sqrt(roberts_x**2 + roberts_y**2)
    magnitude = np.uint8(np.absolute(magnitude))
    cv2.imwrite(output_path, magnitude)
    print(f"Roberts edge-detected image saved as {output_path}")
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(magnitude, cmap='gray')
    plt.title('Roberts Edge Detected Image')
    plt.axis('off')
    plt.show()
    cv2.imshow('Original Image', image)
    cv2.imshow('Roberts Edge Detected Image', magnitude)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
input_image_path = 'forest.png'
output_image_path = 'roberts_edge_detected_image.jpg'

roberts_cross_edge_detection(input_image_path, output_image_path)
```

Output: