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

1. 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:



1. 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:



1. 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}")

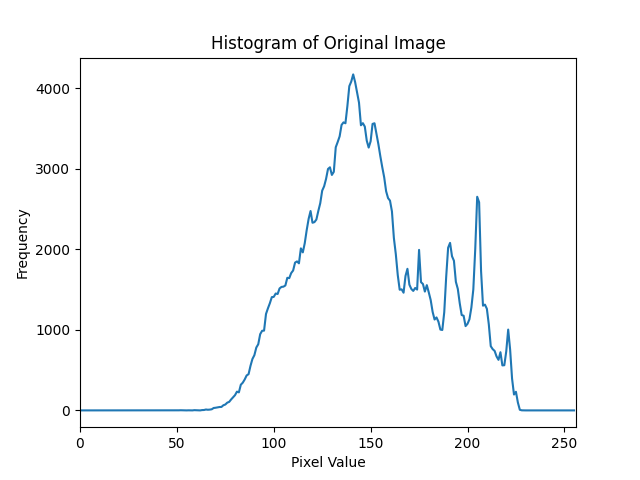
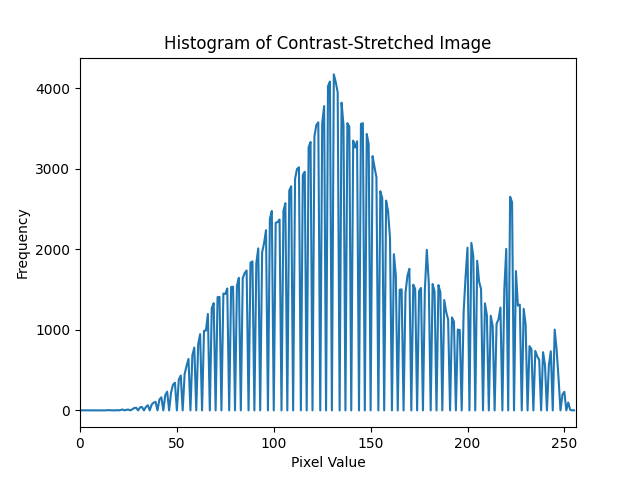
cv2.imshow('Original Image', image)

cv2.imshow('Contrast-Stretched Image', stretched\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

Output:



1. 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

1. 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])

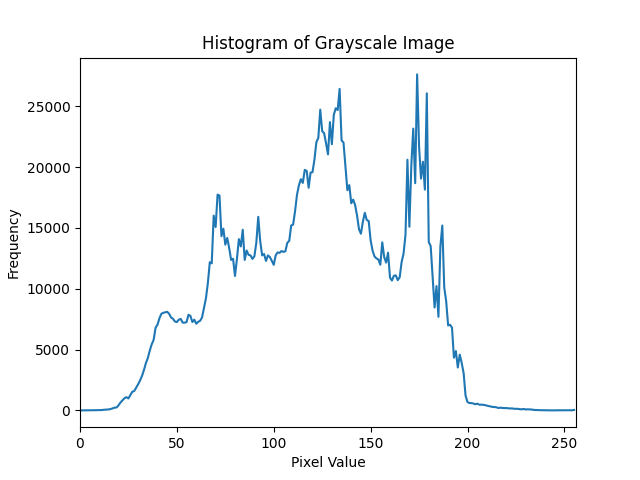
    plt.show()

# Example usage

input\_image\_path = 'apple.jpeg'

plot\_histogram(input\_image\_path)

Output:



1. 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)

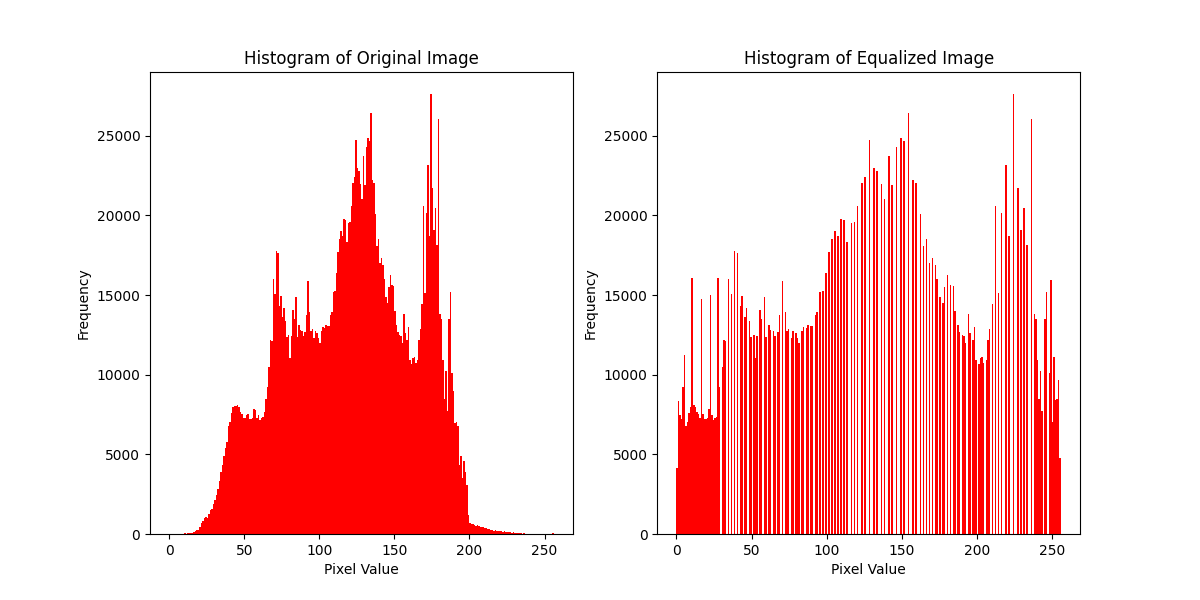
    cv2.destroyAllWindows()

# 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:

1. 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()

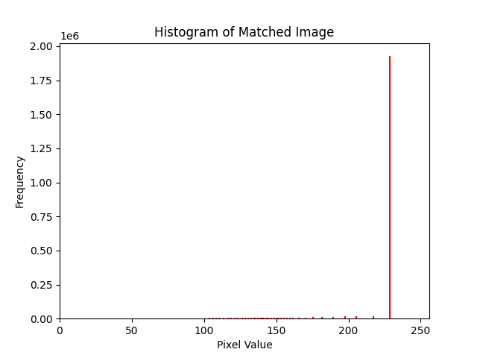
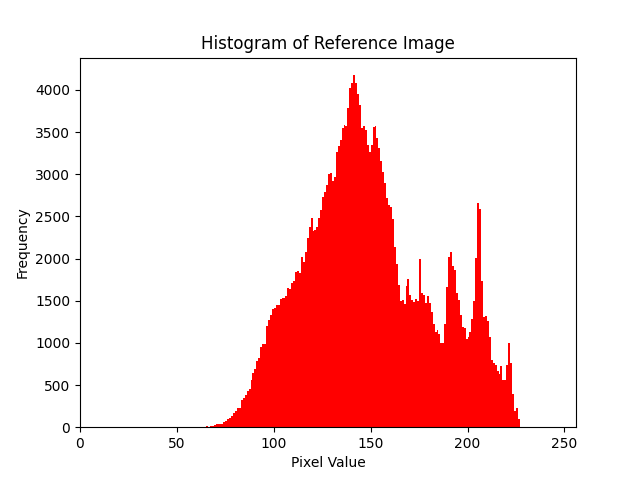
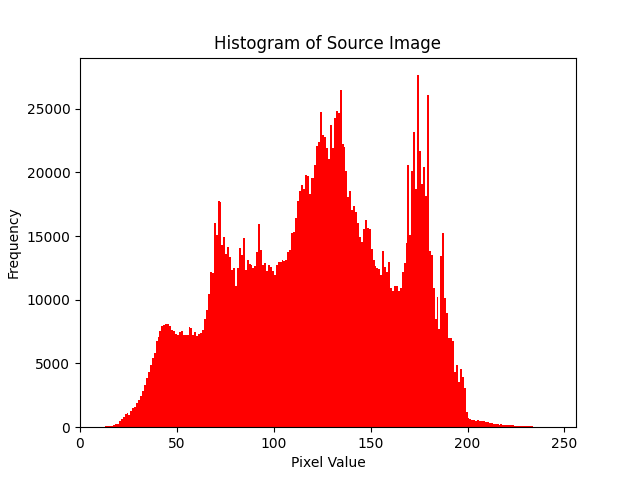
source\_image\_path = 'apple.jpeg'

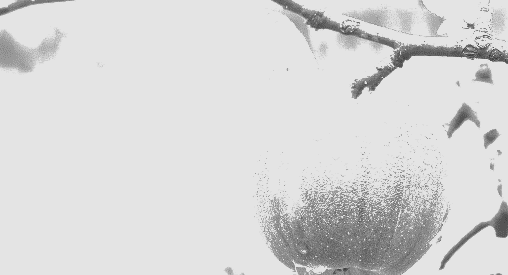
reference\_image\_path = 'forest.png'

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



1. 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:

1. 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)

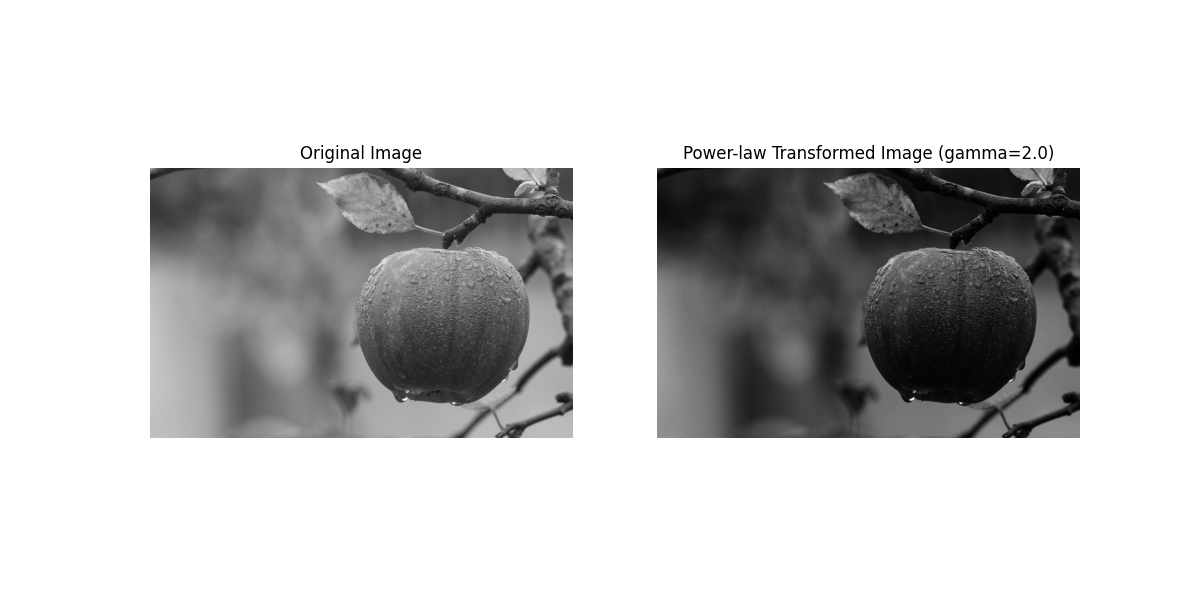
    cv2.destroyAllWindows()

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:

1. 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:

1. 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:

1. 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: