PROGRAM TO PRINT A IMAGE :

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

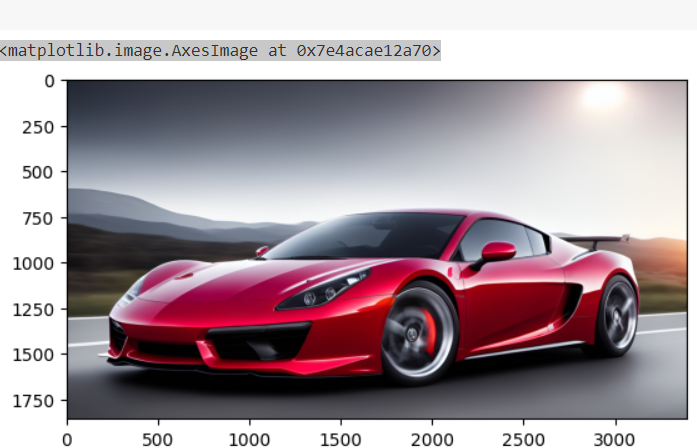
# Read Images

img = mpimg.imread(/content/car3.jpeg')

# Output Images

plt.imshow(img)

OUTPUT :



PROGRAM TO COVERT AN IMAGE INTO GREY SCALE :

from PIL import Image

import matplotlib.pyplot as plt

 # storing image path

fname = r' /content/car3.jpeg’

img = Image.open(‘/content/car3.jpeg’)

image = Image.open(fname).convert("L")

plt.subplot(1,2,1)

plt.imshow(img)

plt.title('Original Image')

# opening image using pil

# mapping image to gray scale

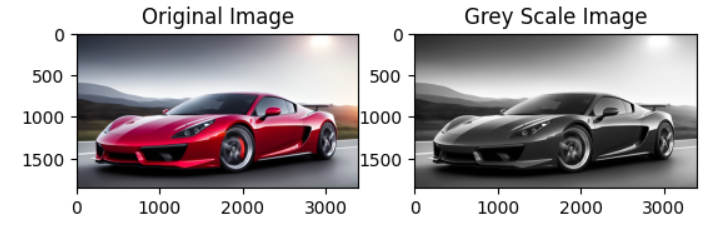
plt.subplot(1,2,2)

plt.imshow(image, cmap='gray')

plt.title("Grey Scale Image")

plt.show()

OUTPUT :



PROGRAM TO PERFORM HISTOGRAM :

import cv2

# importing library for plotting

from matplotlib import pyplot as plt

# reads an input image

img = cv2.imread('/content/beach.jpg')

# find frequency of pixels in range 0-255

histr = cv2.calcHist([img],[0],None,[256],[0,256])

plt.subplot(1,2,1)

plt.imshow(img)

plt.title('Original Image')

# show the plotting graph of an image

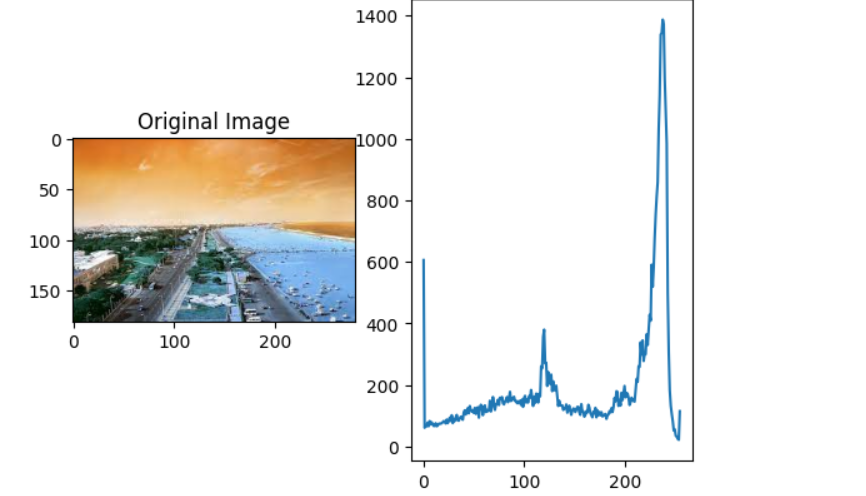
plt.subplot(1,2,2)

plt.plot(histr)

plt.title("histogram")

plt.show()

OUTPUT : Histogram



PROGRAM TO PRINT RGB VALUES OF AN IMAGE :

from PIL import Image

# Open the image

image\_path = "/content/spider-man-homecoming-wallpaper-9.jpg"

image = Image.open(image\_path)

# Get the dimensions of the image

width, height = image.size

# Prompt the user for the number of pixels

num\_pixels =  1

# Loop through each pixel

for i in range(num\_pixels):

    x = int(input("Enter the x-coordinate of pixel {}: ".format(i+1)))

    for j in range(num\_pixels):

      y = int(input("Enter the y-coordinate of pixel {}: ".format(j+1)))

    # Ensure the specified coordinates are within the image boundaries

    if x < 0 or x >= width or y < 0 or y >= height:

        print("Invalid pixel coordinates!")

    else:

        # Get the RGB values of the pixel at (x, y)

        r, g, b = image.getpixel((x, y))

        # Print the pixel values

        print(f"Pixel {i+1} at ({x}, {y}): R={r}, G={g}, B={b}")

OUTPUT :

Enter the x-coordinate of pixel 1: 1

Enter the y-coordinate of pixel 1: 2

Pixel 1 at (1, 2): R=208, G=237, B=245

PROGRAM TO PRINT RED SCALE,BLUE SCALE AND GREEN SCALE OF AN IMAGE :

import cv2

import numpy as np

import matplotlib.pyplot as plt

def display\_channel(image, channel):

    # Create a blank image with the same shape as the original image

    blank\_image = np.zeros\_like(image)

    # Set the channel of the blank image based on user input

    if channel == 'red':

        blank\_image[:, :, 0] = image[:, :, 0]  # Red channel

    elif channel == 'green':

        blank\_image[:, :, 1] = image[:, :, 1]  # Green channel

    elif channel == 'blue':

        blank\_image[:, :, 2] = image[:, :, 2]  # Blue channel

    else:

        print("Invalid channel selection. Displaying the original image.")

        return image

    return blank\_image

def display\_with\_channel\_selection(image):

    # Get user input for channel selection

    print("Select the channel to display:")

    print("1. Red Channel")

    print("2. Green Channel")

    print("3. Blue Channel")

    print("4. All Channels (Red, Green, Blue)")

    choice = input("Enter your choice (1/2/3/4): ")

    if choice == '1':

        channel\_image = display\_channel(image, 'red')

        channel\_name = 'Red Channel'

        plt.figure(figsize=(12, 6))

        plt.subplot(1, 2, 1)

        plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

        plt.title('Original Image')

        plt.axis('off')

        plt.subplot(1, 2, 2)

        plt.imshow(cv2.cvtColor(channel\_image, cv2.COLOR\_BGR2RGB))

        plt.title(channel\_name)

        plt.axis('off')

        plt.tight\_layout()

        plt.show()

    elif choice == '2':

        channel\_image = display\_channel(image, 'green')

        channel\_name = 'Green Channel'

        plt.figure(figsize=(12, 6))

        plt.subplot(1, 2, 1)

        plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

        plt.title('Original Image')

        plt.axis('off')

        plt.subplot(1, 2, 2)

        plt.imshow(cv2.cvtColor(channel\_image, cv2.COLOR\_BGR2RGB))

        plt.title(channel\_name)

        plt.axis('off')

        plt.tight\_layout()

        plt.show()

    elif choice == '3':

        channel\_image = display\_channel(image, 'blue')

        channel\_name = 'Blue Channel'

        plt.figure(figsize=(12, 6))

        plt.subplot(1, 2, 1)

        plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

        plt.title('Original Image')

        plt.axis('off')

        plt.subplot(1, 2, 2)

        plt.imshow(cv2.cvtColor(channel\_image, cv2.COLOR\_BGR2RGB))

        plt.title(channel\_name)

        plt.axis('off')

        plt.tight\_layout()

        plt.show()

    elif choice == '4':

        # Display all channels separately with the same dimensions

        red\_channel = display\_channel(image, 'red')

        green\_channel = display\_channel(image, 'green')

        blue\_channel = display\_channel(image, 'blue')

        plt.figure(figsize=(16, 8))

        # Original Image

        plt.subplot(2, 2, 1)

        plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

        plt.title('Original Image')

        plt.axis('off')

        # Red Channel

        plt.subplot(2, 2, 2)

        plt.imshow(cv2.cvtColor(red\_channel, cv2.COLOR\_BGR2RGB))

        plt.title('Blue Channel')

        plt.axis('off')

        # Green Channel

        plt.subplot(2, 2, 3)

        plt.imshow(cv2.cvtColor(green\_channel, cv2.COLOR\_BGR2RGB))

        plt.title('Green Channel')

        plt.axis('off')

        # Blue Channel

        plt.subplot(2, 2, 4)

        plt.imshow(cv2.cvtColor(blue\_channel, cv2.COLOR\_BGR2RGB))

        plt.title('Red Channel')

        plt.axis('off')

        plt.tight\_layout()

        plt.show()

    else:

        print("Invalid choice. Displaying the original image.")

        channel\_image = image

        channel\_name = 'Original Image'

# Load the original image

original\_image\_path = '/content/redfort1.jpg'

 # Replace with the actual path of the image

original\_image = cv2.imread(original\_image\_path)

display\_with\_channel\_selection(original\_image)

OUTPUT :



PROGRAM TO PERFORM ARITHMETIC OPERATIONS OF AN IMAGE :

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

def load\_image(image\_path):

    try:

        # Open the image

        image = Image.open(image\_path)

        # Convert the image to RGB mode if it's in a different mode (e.g., RGBA)

        if image.mode != 'RGB':

            image = image.convert('RGB')

        return image

    except Exception as e:

        print("Error:", e)

        return None

def perform\_arithmetic\_operation(image1, image2, operation):

    # Convert images to NumPy arrays for easier manipulation

    array1 = np.array(image1)

    array2 = np.array(image2)

    if operation == "add":

        result\_array = np.clip(array1 + array2, 0, 255).astype(np.uint8)  # Ensure values are within [0, 255]

    elif operation == "subtract":

        result\_array = np.clip(array1 - array2, 0, 255).astype(np.uint8)

    elif operation == "and":

        result\_array = np.bitwise\_and(array1, array2)

    elif operation == "or":

        result\_array = np.bitwise\_or(array1, array2)

    elif operation == "not":

        result\_array = np.bitwise\_not(array1)

    elif operation == "xor":

        result\_array = np.bitwise\_xor(array1, array2)

    else:

        print("Invalid operation.")

        return None

    # Create a new image from the NumPy array and return it

    result\_image = Image.fromarray(result\_array)

    return result\_image

def display\_images(image1, image2, result\_image):

    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    # Display the first input image

    axes[0].imshow(image1)

    axes[0].set\_title('Image 1')

    axes[0].axis('off')

    # Display the second input image

    axes[1].imshow(image2)

    axes[1].set\_title('Image 2')

    axes[1].axis('off')

    # Display the result image

    axes[2].imshow(result\_image)

    axes[2].set\_title('Result Image')

    axes[2].axis('off')

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    # Replace "path\_to\_image1.jpg" and "path\_to\_image2.jpg" with the actual paths to your image files

    image1\_path = "/content/halwa.jpg"

    image2\_path = "/content/halwa1.jpg"

    # Load the images

    image1 = load\_image("/content/halwa.jpg")

    image2 = load\_image("/content/halwa1.jpg")

    if image1 and image2:

        print("Select an operation:")

        print("1 - Addition")

        print("2 - Subtraction")

        print("3 - Bitwise AND")

        print("4 - Bitwise OR")

        print("5 - Bitwise NOT (only applied to image1)")

        print("6 - Bitwise XOR")

        choice = int(input("Enter your choice: "))

        operations = {

            1: "add",

            2: "subtract",

            3: "and",

            4: "or",

            5: "not",

            6: "xor"

        }

        if choice in operations:

            operation = operations[choice]

            result\_image = perform\_arithmetic\_operation(image1, image2, operation)

            if result\_image:

                display\_images(image1, image2, result\_image)

        else:

            print("Invalid choice.")

    else:

        print("Unable to load images.")

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

def load\_image(image\_path):

    try:

        # Open the image

        image = Image.open(image\_path)

        # Convert the image to RGB mode if it's in a different mode (e.g., RGBA)

        if image.mode != 'RGB':

            image = image.convert('RGB')

        return image

    except Exception as e:

        print("Error:", e)

        return None

def perform\_arithmetic\_operation(image1, image2, operation):

    # Convert images to NumPy arrays for easier manipulation

    array1 = np.array(image1)

    array2 = np.array(image2)

    if operation == "add":

        result\_array = np.clip(array1 + array2, 0, 255).astype(np.uint8)  # Ensure values are within [0, 255]

    elif operation == "subtract":

        result\_array = np.clip(array1 - array2, 0, 255).astype(np.uint8)

    elif operation == "and":

        result\_array = np.bitwise\_and(array1, array2)

    elif operation == "or":

        result\_array = np.bitwise\_or(array1, array2)

    elif operation == "not":

        result\_array = np.bitwise\_not(array1)

    elif operation == "xor":

        result\_array = np.bitwise\_xor(array1, array2)

    else:

        print("Invalid operation.")

        return None

    # Create a new image from the NumPy array and return it

    result\_image = Image.fromarray(result\_array)

    return result\_image

def display\_images(image1, image2, result\_image):

    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    # Display the first input image

    axes[0].imshow(image1)

    axes[0].set\_title('Image 1')

    axes[0].axis('off')

    # Display the second input image

    axes[1].imshow(image2)

    axes[1].set\_title('Image 2')

    axes[1].axis('off')

    # Display the result image

    axes[2].imshow(result\_image)

    axes[2].set\_title('Result Image')

    axes[2].axis('off')

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    # Replace "path\_to\_image1.jpg" and "path\_to\_image2.jpg" with the actual paths to your image files

    image1\_path = "/content/halwa.jpg"

    image2\_path = "/content/halwa1.jpg"

    # Load the images

    image1 = load\_image("/content/halwa.jpg")

    image2 = load\_image("/content/halwa1.jpg")

    if image1 and image2:

        print("Select an operation:")

        print("1 - Addition")

        print("2 - Subtraction")

        print("3 - Bitwise AND")

        print("4 - Bitwise OR")

        print("5 - Bitwise NOT (only applied to image1)")

        print("6 - Bitwise XOR")

        choice = int(input("Enter your choice: "))

        operations = {

            1: "add",

            2: "subtract",

            3: "and",

            4: "or",

            5: "not",

            6: "xor"

        }

        if choice in operations:

            operation = operations[choice]

            result\_image = perform\_arithmetic\_operation(image1, image2, operation)

            if result\_image:

                display\_images(image1, image2, result\_image)

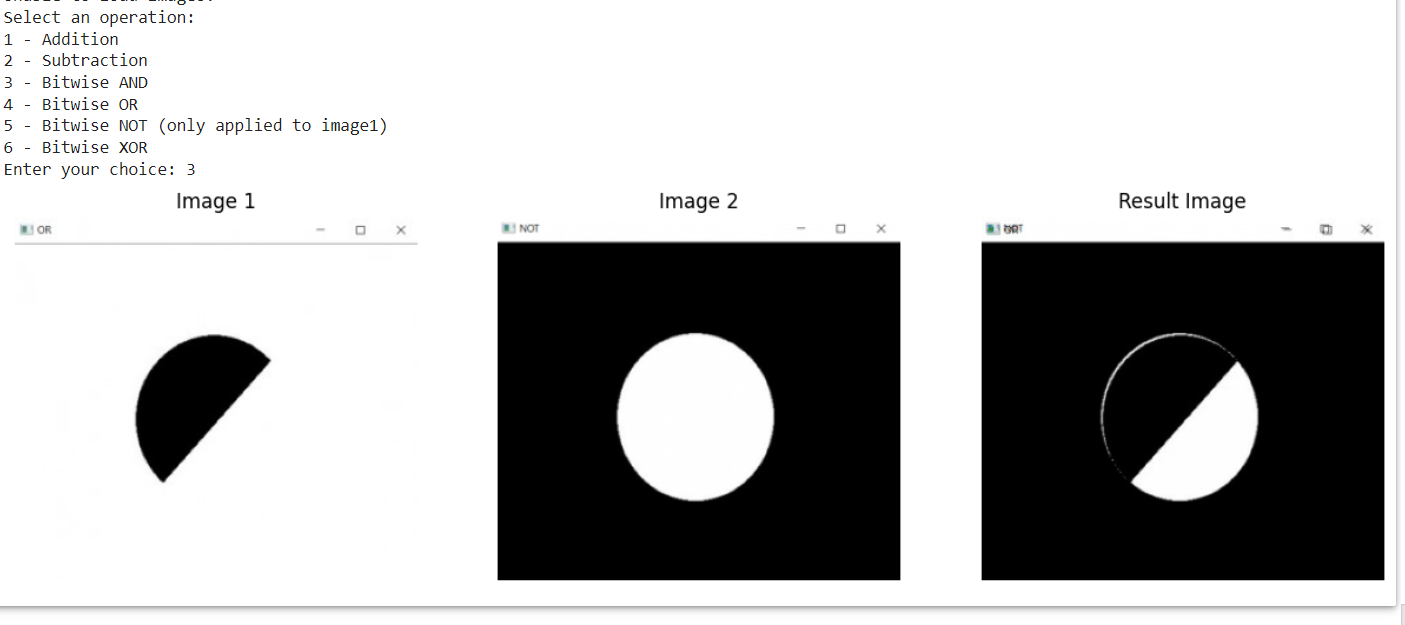
        else:

            print("Invalid choice.")

    else:

        print("Unable to load images.")

OUTPUT :



PROGRAM TO PERFORM WAVELET TRANSFORMATION:

import numpy as np

import matplotlib.pyplot as plt

import pywt

from PIL import Image

def main():

    image\_path = input("Enter the path to the image: ")

    try:

        original = np.array(Image.open(image\_path).convert('L'))  # Convert to grayscale

    except Exception as e:

        print("Error:", e)

        return

    wavelet\_name = input("Enter the wavelet name (e.g., 'haar', 'db2'): ")

    try:

        coeffs2 = pywt.dwt2(original, wavelet\_name)

        cA, (cH, cV, cD) = coeffs2

        reconstructed = pywt.idwt2((cA, (cH, cV, cD)), wavelet\_name)

        fig, axes = plt.subplots(2, 2, figsize=(10, 10))

        axes[0, 0].imshow(cA, cmap=plt.cm.gray)

        axes[0, 0].set\_title('Approximation (cA)')

        axes[0, 1].imshow(cH, cmap=plt.cm.gray)

        axes[0, 1].set\_title('Horizontal Detail (cH)')

        axes[1, 0].imshow(cV, cmap=plt.cm.gray)

        axes[1, 0].set\_title('Vertical Detail (cV)')

        axes[1, 1].imshow(cD, cmap=plt.cm.gray)

        axes[1, 1].set\_title('Diagonal Detail (cD)')

        for ax in axes.ravel():

            ax.axis('off')

        plt.tight\_layout()

        plt.show()

        fig, axes = plt.subplots(1, 2, figsize=(10, 5))

        axes[0].imshow(original, cmap=plt.cm.gray)

        axes[0].set\_title('Original Image')

        axes[1].imshow(reconstructed, cmap=plt.cm.gray)

        axes[1].set\_title('Reconstructed Image')

        for ax in axes:

            ax.axis('off')

        plt.tight\_layout()

        plt.show()

    except ValueError as ve:

        print("Error:", ve)

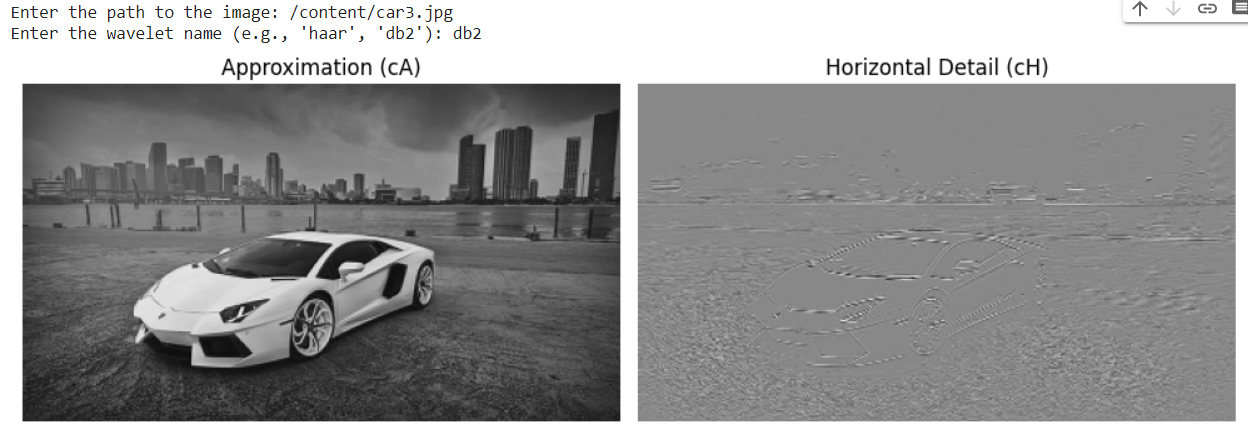
    except Exception as e:

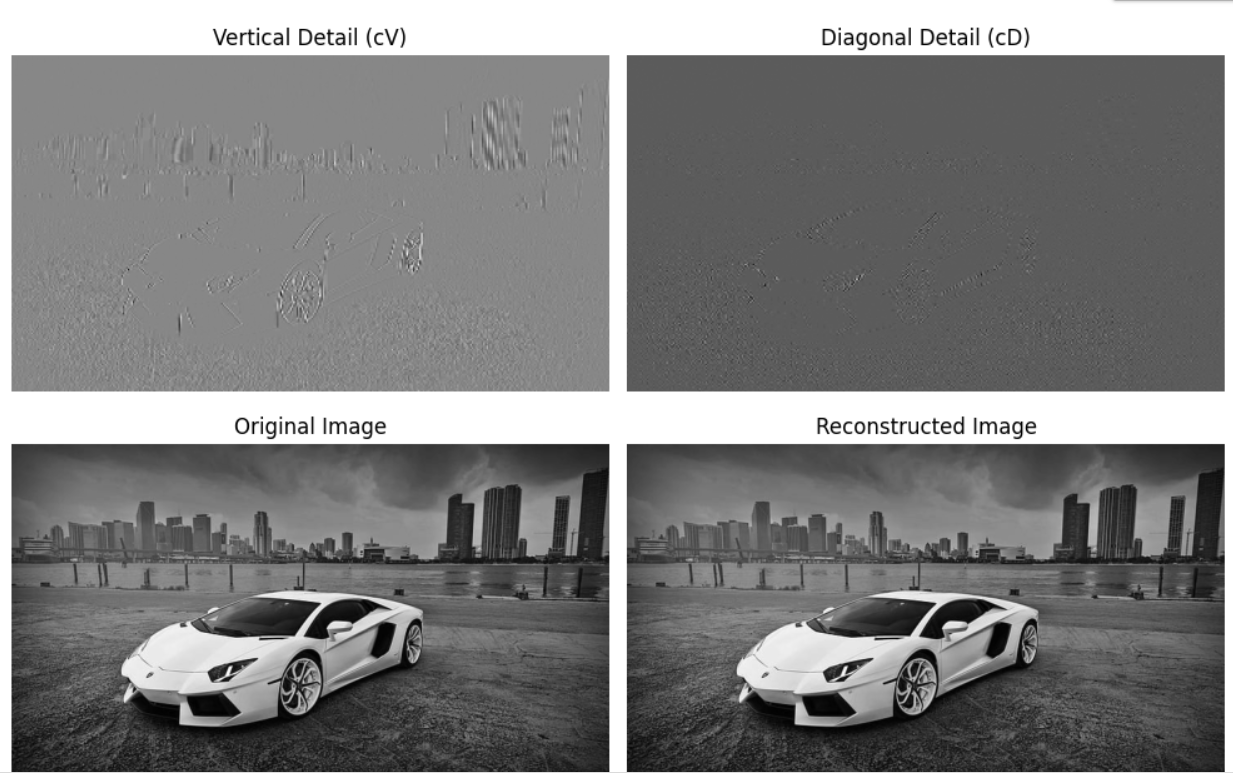
        print("An error occurred:", e)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

OUTPUT :





PROGRAM TO PERFORM IMAGE RESTORATION:

import numpy as np

import matplotlib.pyplot as plt

from scipy.ndimage import convolve, gaussian\_filter1d

def restore\_image(degraded\_image, kernel, noise\_level, regularization\_param):

    # Apply Wiener deconvolution

    kernel\_fft = np.fft.fft2(kernel, s=degraded\_image.shape)

    restored\_image = np.fft.ifft2(

        np.fft.fft2(degraded\_image) / (kernel\_fft + regularization\_param)

    ).real

    # Add noise to the restored image to simulate the noisy environment

    restored\_image += np.random.normal(scale=noise\_level, size=degraded\_image.shape)

    return restored\_image

def main():

    image\_path = input("Enter the path to the image: ")

    try:

        original\_image = plt.imread(image\_path)

        if original\_image.ndim == 3:

            original\_image = original\_image.mean(axis=-1)

    except Exception as e:

        print("Error:", e)

        return

    # Normalize pixel values to [0, 1]

    original\_image = original\_image.astype(np.float32) / 255.0

    # Create a Gaussian kernel for blurring

    kernel = gaussian\_filter1d(gaussian\_filter1d(np.eye(21), 2), 2)

    # Apply Gaussian blur to the original image

    degraded\_image = convolve(original\_image, kernel, mode='wrap')

    # Simulate noise

    noise\_level = 0.05

    # Set the regularization parameter for Wiener deconvolution

    regularization\_param = 0.001

    # Restore the degraded and noisy image

    restored\_image = restore\_image(degraded\_image, kernel, noise\_level, regularization\_param)

    # Display the original, degraded, and restored images

    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)

    plt.imshow(original\_image, cmap='gray')

    plt.title('Original Image')

    plt.subplot(1, 3, 2)

    plt.imshow(degraded\_image, cmap='gray')

    plt.title('Degraded Image')

    plt.subplot(1, 3, 3)

    plt.imshow(restored\_image, cmap='gray')

    plt.title('Restored Image')

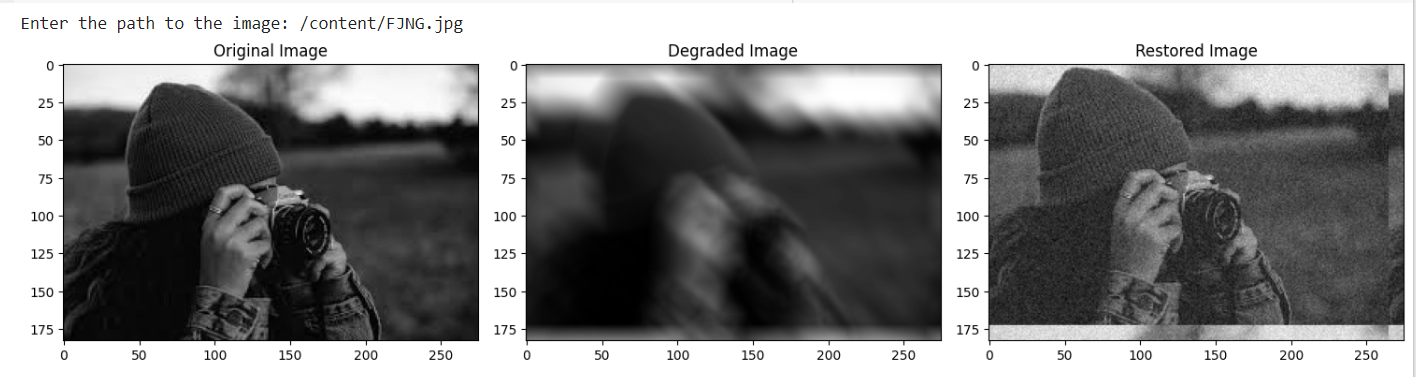
    plt.tight\_layout()

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

OUTPUT :



PROGRAM TO PERFORM MORPHOLOGICAL OPERATIONS:

import cv2

import numpy as np

import matplotlib.pyplot as plt

def main():

    # Get the path to the input image from the user

    image\_path = input("Enter the path to the image: ")

    try:

        image = cv2.imread(image\_path)

        if image is None:

            print("Error: Unable to read the image.")

            return

    except Exception as e:

        print("Error:", e)

        return

    # Define a structuring element (kernel)

    kernel = np.ones((5, 5), np.uint8)

    # Convert the image to grayscale

    gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    # Perform erosion, dilation, opening, and closing on the grayscale image

    erosion = cv2.erode(gray\_image, kernel, iterations=1)

    dilation = cv2.dilate(gray\_image, kernel, iterations=1)

    opening = cv2.morphologyEx(gray\_image, cv2.MORPH\_OPEN, kernel)

    closing = cv2.morphologyEx(gray\_image, cv2.MORPH\_CLOSE, kernel)

    # Display the original and processed color images

    plt.figure(figsize=(12, 12))

    plt.subplot(2, 3, 1)

    plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

    plt.title('Original Image')

    plt.subplot(2, 3, 2)

    plt.imshow(erosion, cmap='gray')

    plt.title('Erosion')

    plt.subplot(2, 3, 3)

    plt.imshow(dilation, cmap='gray')

    plt.title('Dilation')

    plt.subplot(2, 3, 4)

    plt.imshow(opening, cmap='gray')

    plt.title('Opening')

    plt.subplot(2, 3, 5)

    plt.imshow(closing, cmap='gray')

    plt.title('Closing')

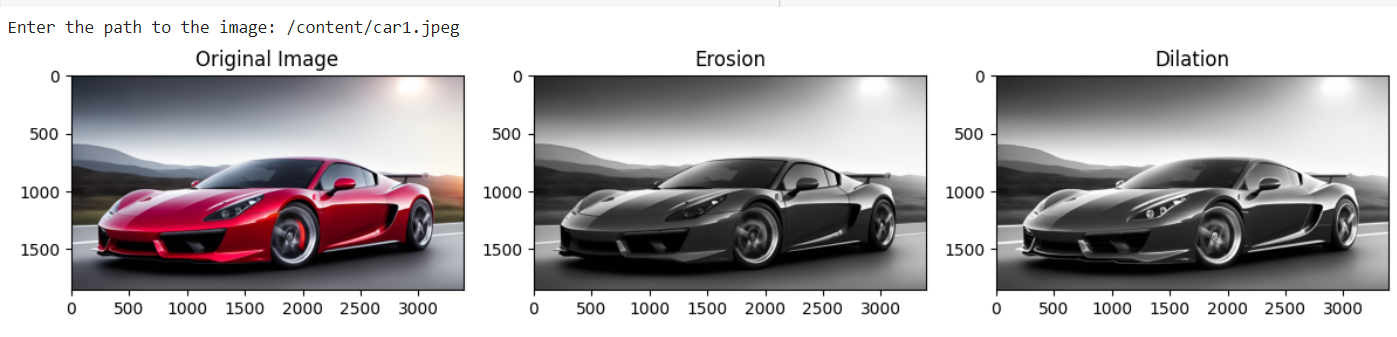
    plt.tight\_layout()

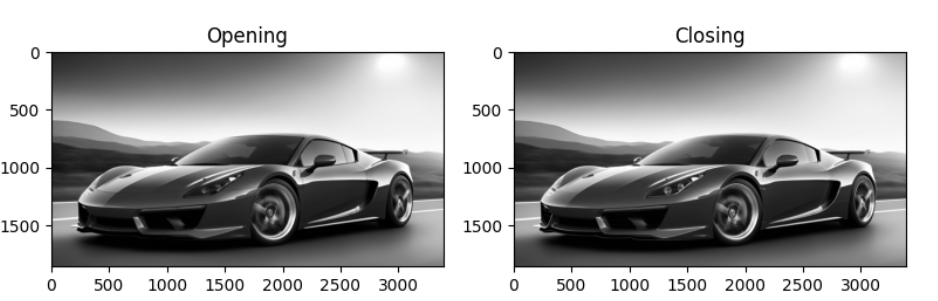
    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

OUTPUT :





PROGRAM TO PERFORM FREQUENCY DOMAIN FILTERATION

(LOW PASS AND HIGH PASS):

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load an image

image\_path = '/content/car1.jpg' # Replace with the path to your image

image = cv2.imread(image\_path)

# Check if the image was loaded successfully

if image is None:

    print("Error: Unable to load the image.")

    exit()

# Convert the image to grayscale

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Define a low-pass filter kernel (e.g., Gaussian kernel)

low\_pass\_kernel = np.array([[1, 4, 6, 4, 1],

                             [4, 16, 24, 16, 4],

                             [6, 24, 36, 24, 6],

                             [4, 16, 24, 16, 4],

                             [1, 4, 6, 4, 1]], dtype=np.float32) / 256

# Apply low-pass filtering to create the low-frequency component

low\_frequency\_component = cv2.filter2D(gray\_image, -1, low\_pass\_kernel)

# Blur the low-frequency component (you can adjust the kernel size for different blurring levels)

blurred\_low\_frequency\_component = cv2.GaussianBlur(low\_frequency\_component, (5, 5), 0)

# Subtract the blurred low-frequency component from the original grayscale image to obtain the high-frequency component

high\_frequency\_component = gray\_image - blurred\_low\_frequency\_component

# Restore the image by adding the blurred low-frequency component to the high-frequency component

restored\_image = high\_frequency\_component + blurred\_low\_frequency\_component

# Display the original image, low-frequency component, high-frequency component, and restored image

plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)

plt.imshow(gray\_image, cmap='gray')

plt.title('Original Grayscale Image')

plt.axis('off')

plt.subplot(2, 2, 2)

plt.imshow(low\_frequency\_component, cmap='gray')

plt.title('Low-Frequency Component')

plt.axis('off')

plt.subplot(2, 2, 3)

plt.imshow(high\_frequency\_component, cmap='gray')

plt.title('High-Frequency Component')

plt.axis('off')

plt.subplot(2, 2, 4)

plt.imshow(restored\_image, cmap='gray')

plt.title('Restored Image')

plt.axis('off')

plt.tight\_layout()

plt.show()

OUTPUT :

