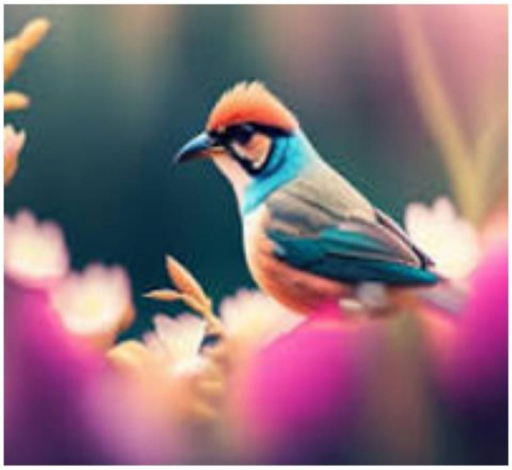
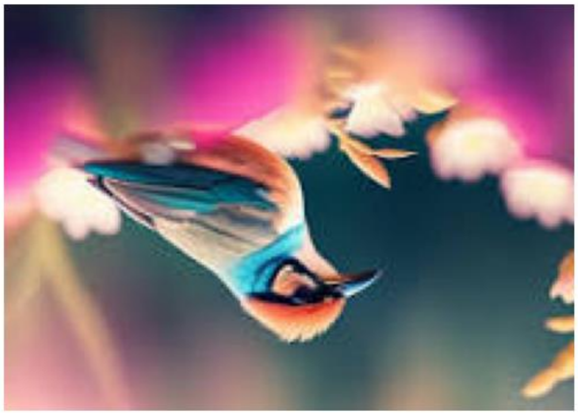
Expt No: 1 Write a python script to perform basic operations on images.

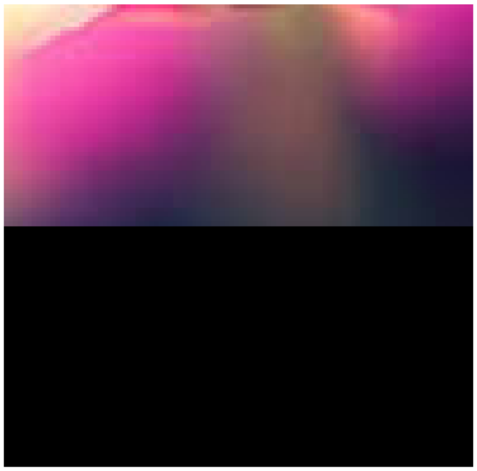
```
from PIL import Image, ImageOps
from matplotlib import pyplot as plt
def resize_image(image_path, output_path, size):
    image = Image.open(image_path)
    resized_image = image.resize(size)
    resized_image.save(output_path)
    print(f"Image resized to {size} and saved to {output_path}")
    return resized_image  # Return the processed image for display

def rotate_image(image_path, output_path, angle):
    image = Image.open(image_path)
    rotated_image = image.rotate(angle)
    rotated_image.save(output_path)
    print(f"Image rotated by {angle} degrees and saved to {output_path}")
    return rotated_image  # Return the processed image for display

def crop_image(image_path, output_path, crop_area):
    image = Image.open(image_path)
    cropped_image = image.crop(crop_area)
    cropped_image.save(output_path)
    print(f"Image cropped with area {crop_area} and saved to {output_path}")
    return cropped_image  # Return the processed image for display

def convert_to_grayscale(image_path, output_path):
    image = Image.open(image_path)
    grayscale_image = ImageOps.grayscale(image)
    grayscale_image.save(output_path)
    print(f"Image converted to grayscale and saved to {output_path}")
    return grayscale_image  # Return the processed image for display

def display_image(image):
    # Use plt.imshow to display the image
    plt.imshow(image)
    plt.axis('off')  # Turn off axis numbers and ticks
    plt.show()

def main():
    image_path = 'C:/Users/ADMIN/Pictures/img1.jfif'  # Replace with your image path
    # Resize the image to 200x200 and display
    resized_image = resize_image(image_path, 'resized_image.jpg', (200, 200))
    display_image(resized_image)
    # Rotate the image by 180 degrees and display
    rotated_image = rotate_image(image_path, 'rotated_image.jpg', 180)
    display_image(rotated_image)
    # Crop the image (left, upper, right, lower) and display
    cropped_image = crop_image(image_path, 'cropped_image.jpg', (100, 100, 400, 400))
    display_image(cropped_image)
    # Convert the image to grayscale and display
    grayscale_image = convert_to_grayscale(image_path, 'grayscale_image.jpg')
    display_image(grayscale_image)
if __name__ == '__main__':
    main()
```



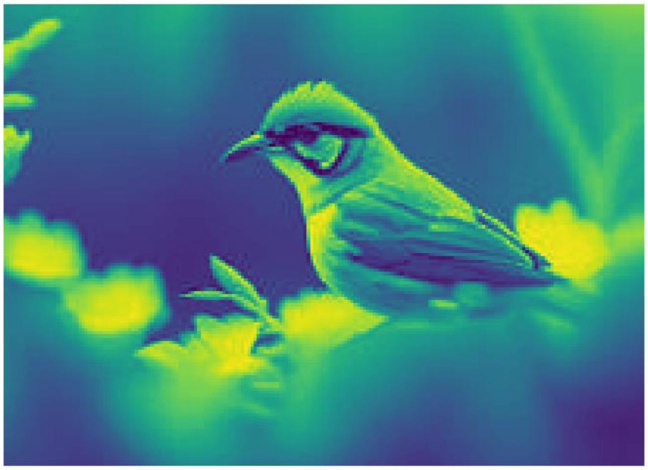
Resized Image



Ratated Image



Cropped Image

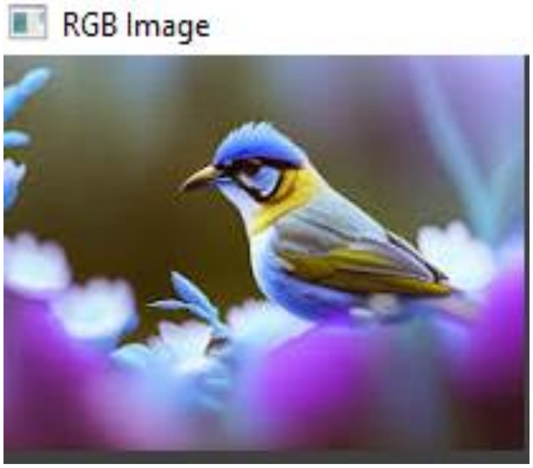


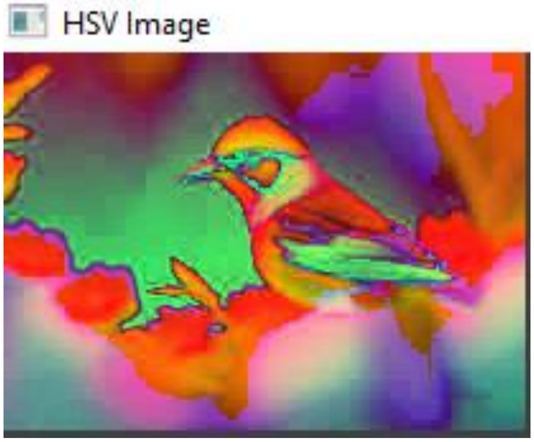
Grayscale Image

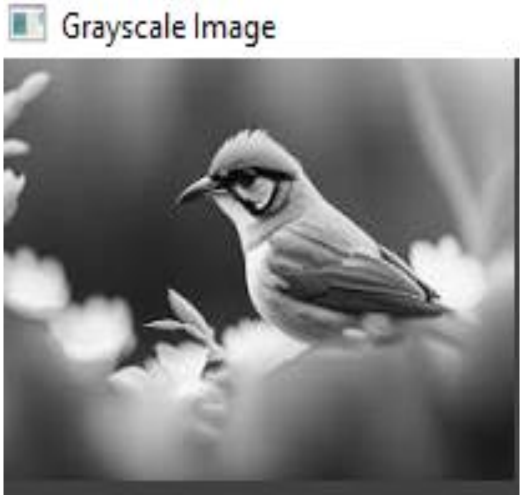
Expt No: 2 Write a python script to perform conversion between color spaces.

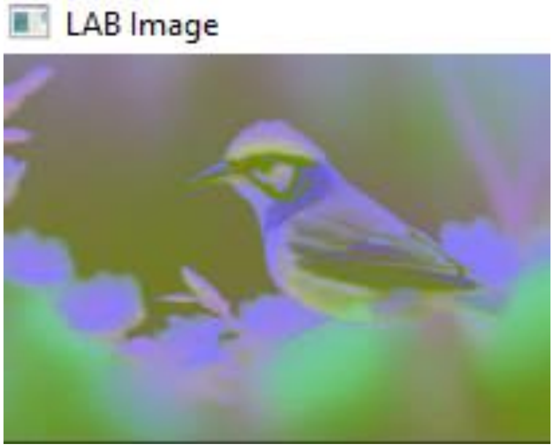
```
import cv2
def convert_color_space(image_path):
    # Read the image
    image = cv2.imread(image_path)

    if image is None:
        print(f"Error: Unable to open image file {image_path}")
        return
    # Convert to RGB
    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    cv2.imwrite('rgb_image.jpg', rgb_image)
    cv2.imshow('RGB Image', rgb_image)  # Display the RGB image
    print("Image converted to RGB and saved as rgb_image.jpg")

    # Convert to HSV
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    cv2.imwrite('hsv_image.jpg', hsv_image)
    cv2.imshow('HSV Image', hsv_image)  # Display the HSV image
    print("Image converted to HSV and saved as hsv_image.jpg")

    # Convert to Grayscale
    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    cv2.imwrite('grayscale_image.jpg', grayscale_image)
    cv2.imshow('Grayscale Image', grayscale_image)  # Display the grayscale image
    print("Image converted to Grayscale and saved as grayscale_image.jpg")

    # Convert to LAB
    lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    cv2.imwrite('lab_image.jpg', lab_image)
    cv2.imshow('LAB Image', lab_image)  # Display the LAB image
    print("Image converted to LAB and saved as lab_image.jpg")

    # Wait for user to press any key before closing windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def main():
    image_path = 'C:/Users/ADMIN/Pictures/img1.jfif'  # Replace with your image path
    convert_color_space(image_path)

if __name__ == '__main__':
    main()
```

OUTPUT:



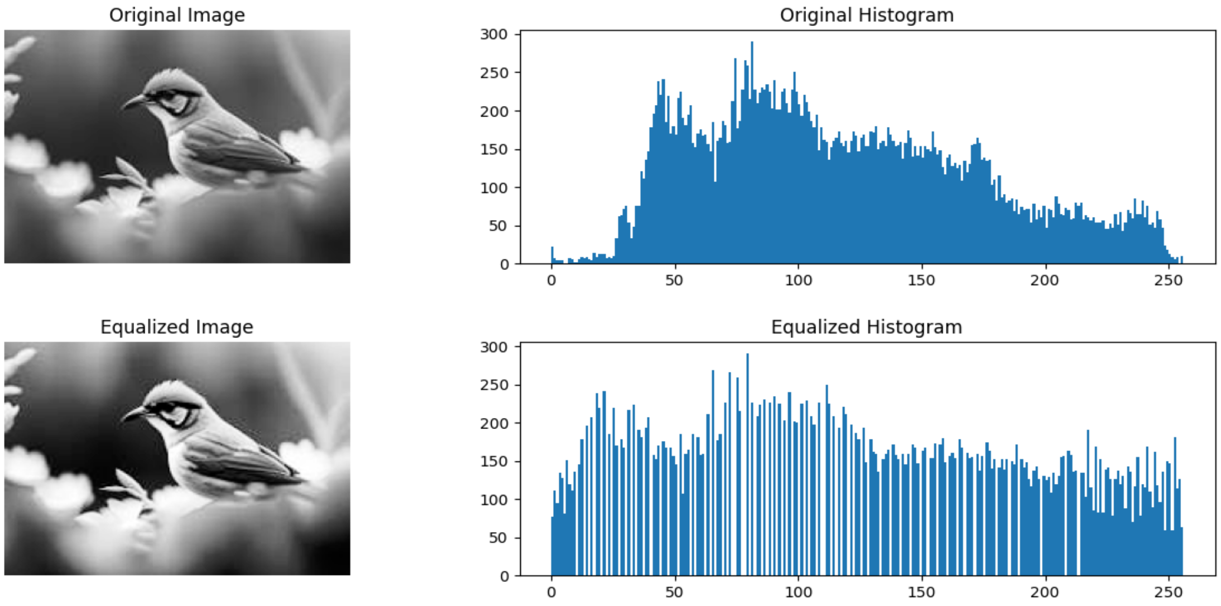






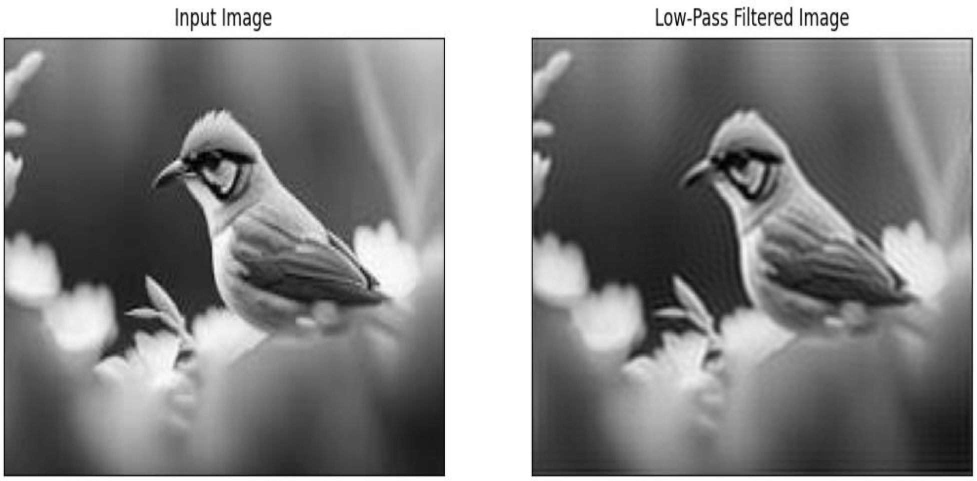
Expt No: 3 Write a python script to perform histogram equalization.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
def histogram_equalization(image_path):
# Read the image in grayscale mode
    img = cv2.imread(image_path,
cv2.IMREAD_GRAYSCALE)
# Check if the image was loaded
    if img is None:
        print("Error: Could not load image.")
        return
# Apply histogram equalization
    equ_img = cv2.equalizeHist(img)
# Plot the original and equalized histograms
    plt.figure(figsize=(10, 5))
    # Original image and its 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(), 256, [0, 256])
    plt.title('Original Histogram')
    # Equalized image and its histogram
    plt.subplot(2, 2, 3)
    plt.imshow(equ_img, cmap='gray')
    plt.title('Equalized Image')
    plt.axis('off')
    plt.subplot(2, 2, 4)
    plt.hist(equ_img.ravel(), 256, [0, 256])
    plt.title('Equalized Histogram')
    plt.tight_layout()
    plt.show()
    # Save the equalized image
    output_image_path =
'equalized_image.png'
    cv2.imwrite(output_image_path,
equ_img)
    print(f"Equalized image saved as
{output_image_path}")
# Example usage
image_path =
'C:/Users/ADMIN/Pictures/img1.jfif'
# Replace with your image path
histogram_equalization(image_path)
```



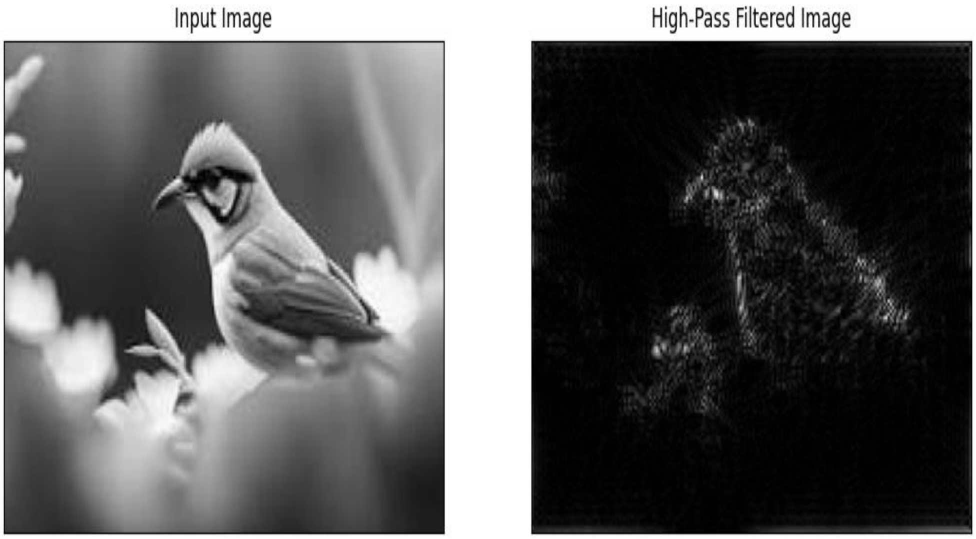
Expt No: 5 Write a python script to perform image filtering in spatial domain.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
# Load the image in grayscale
image = cv2.imread('C:/Users/ADMIN/Pictures/img1.jfif', 0)
# Step 1: Perform DFT
dft = cv2.dft(np.float32(image), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
# Step 2: Create a mask for filtering
rows, cols = image.shape
crow, ccol = rows // 2 , cols // 2
# Low-pass filter mask
mask = np.zeros((rows, cols, 2), np.uint8)
r = 50 # Radius of the low-pass filter
center = [crow, ccol]
cv2.circle(mask, (ccol, crow), r, (1, 1, 1), thickness=-1)
# Apply mask (Low-pass filter)
fshift = dft_shift * mask
# Step 3: Inverse DFT to get filtered image
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])
# Display the original and the low-pass filtered image
plt.subplot(121),plt.imshow(image, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back, cmap = 'gray')
plt.title('Low-Pass Filtered Image'), plt.xticks([]), plt.yticks([])
plt.show()
# To apply a high-pass filter, invert the mask
mask_highpass = np.ones((rows, cols, 2), np.uint8)
cv2.circle(mask_highpass, (ccol, crow), r, (0, 0, 0), thickness=-1)
# Apply mask (High-pass filter)
fshift_highpass = dft_shift * mask_highpass
# Inverse DFT for high-pass filtered image
f_ishift_high = np.fft.ifftshift(fshift_highpass)
img_back_high = cv2.idft(f_ishift_high)
img_back_high = cv2.magnitude(img_back_high[:, :, 0], img_back_high[:, :, 1])
# Display the high-pass filtered image
plt.subplot(121),plt.imshow(image, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_back_high, cmap = 'gray')
plt.title('High-Pass Filtered Image'), plt.xticks([]), plt.yticks([])
plt.show()
```



Input Image

Low-Pass Filtered Image



Input Image

High-Pass Filtered Image

Practical 6. Write a Python script to perform image restoration.

```
import cv2
import numpy as np
from scipy.signal import convolve2d
import matplotlib.pyplot as plt

# Function to perform Wiener filtering
def wiener_filter(image, kernel, noise_variance, estimated_noise_variance):
    """Apply Wiener filter to an image.
  :param image: Input image (grayscale)
    :param kernel: The kernel used for blurring (convolution filter)
    :param noise_variance: Estimated noise variance in the image
    :param estimated_noise_variance: Estimated noise variance
    :return: Restored image
    """
 # Convolve the image with the kernel (blurring)
    blurred_image = convolve2d(image, kernel, mode='same', boundary='wrap')
    # Calculate the noise-to-signal ratio
    noise_to_signal_ratio = noise_variance / estimated_noise_variance
    # Wiener filter formula to restore the image
    restored_image = blurred_image + noise_to_signal_ratio * (image - blurred_image)
   return np.clip(restored_image, 0, 255).astype(np.uint8)
# Read the input image
image = cv2.imread('image_with_noise.jpg', cv2.IMREAD_GRAYSCALE)

# Define a simple blur kernel (e.g., 5x5 Gaussian kernel)
kernel = np.ones((5, 5)) / 25  # simple averaging kernel for demonstration
# Set noise variance and estimated noise variance (example values)
noise_variance = 5.0
estimated_noise_variance = 10.0
# Perform Wiener image restoration
```

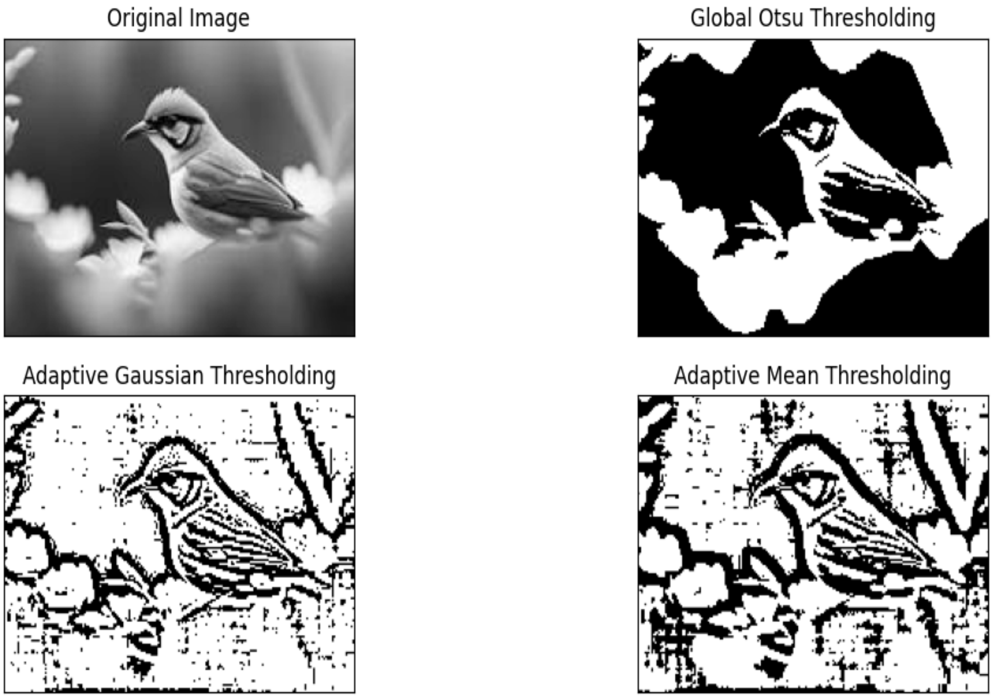
```
restored_image = wiener_filter(image, kernel, noise_variance, estimated_noise_variance)
# Display the original and restored images
plt.figure(figsize=(10, 5))
# Original image
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
# Restored image
plt.subplot(1, 2, 2)
plt.imshow(restored_image, cmap='gray')
plt.title('Restored Image')
plt.axis('off')
plt.show()
# Save the restored image
cv2.imwrite('restored_image.jpg', restored_image)
```

EXP 7. Write a Python script to perform edge detection using various operator

```
import cv2

import numpy as np


# Load the image in grayscale

image = cv2.imread('input_image.jpg', cv2.IMREAD_GRAYSCALE)


if image is None:

    print("Error: Could not load image.")

    exit()

# Sobel Edge Detection

sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)  # Horizontal edges

sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)  # Vertical edges

sobel_combined = cv2.magnitude(sobel_x, sobel_y)  # Magnitude of gradients


# Laplacian Edge Detection

laplacian = cv2.Laplacian(image, cv2.CV_64F)

# Canny Edge Detection

canny_edges = cv2.Canny(image, 50, 150)  # Adjust thresholds as needed

# Display results

cv2.imshow('Original Image', image)

cv2.imshow('Sobel Combined', cv2.convertScaleAbs(sobel_combined))

cv2.imshow('Laplacian', cv2.convertScaleAbs(laplacian))

cv2.imshow('Canny', canny_edges)

# Wait for a key press and close all windows

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Expt No: 8 Write a python script to perform global, adaptive, Ostu’s thresholding.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
# Read the image
image = cv2.imread('C:/Users/ADMIN/Pictures/img1.jfif', 0) # Load the image in grayscale
# Global Otsu's thresholding
ret1, th1 = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
# Adaptive thresholding (Gaussian)
th2 = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
 cv2.THRESH_BINARY, 11, 2)
# Adaptive thresholding (Mean)
th3 = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
 cv2.THRESH_BINARY, 11, 2)
# Plot the results
titles = ['Original Image', 'Global Otsu Thresholding', 'Adaptive Gaussian Thresholding',
'Adaptive Mean Thresholding']
images = [image, th1, th2, th3]
for i in range(4):
 plt.subplot(2, 2, i+1), plt.imshow(images[i], 'gray')
 plt.title(titles[i])
 plt.xticks([]), plt.yticks([])
plt.show()
```



Original Image

Global Otsu Thresholding

Adaptive Gaussian Thresholding

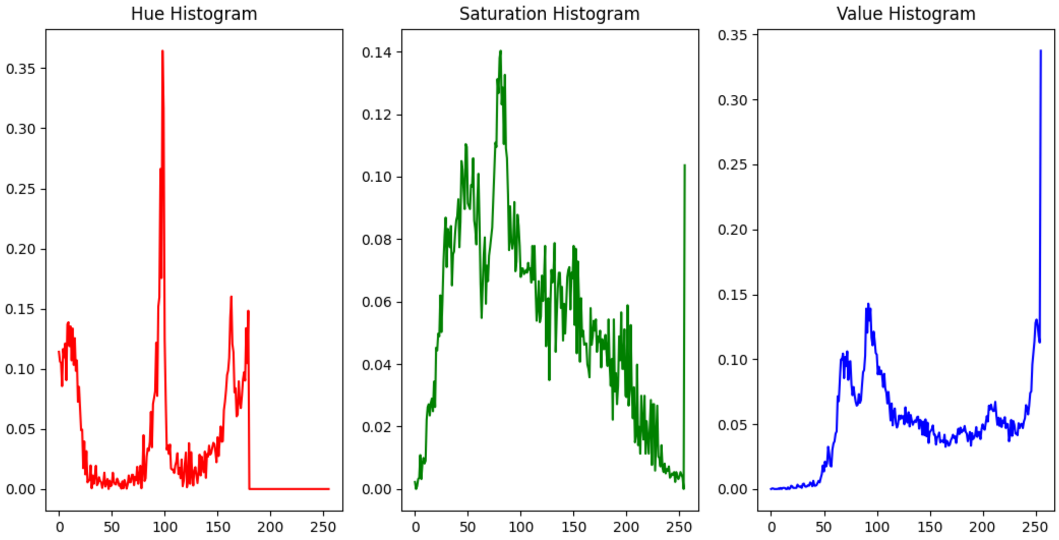
Adaptive Mean Thresholding

9. Write a Python script to apply morphological operations on an image.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# pip install opencv-python

def apply_morphological_operations(image_path):
    """
    Apply morphological operations on an image and display results.

    :param image_path: Path to the input image
    """
    # Load the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Check if the image is loaded
    if image is None:
        print("Error: Could not load image.")
        return

    # Display the original image
    plt.figure(figsize=(12, 8))
    plt.subplot(2, 3, 1)
    plt.title('Original Image')
    plt.imshow(image, cmap='gray')
    plt.axis('off')

    # Define a kernel (structuring element)
    kernel = np.ones((5, 5), np.uint8)

    # Apply Erosion
    erosion = cv2.erode(image, kernel, iterations=1)
    plt.subplot(2, 3, 2)
    plt.title('Erosion')
    plt.imshow(erosion, cmap='gray')
    plt.axis('off')

    # Apply Dilation
    dilation = cv2.dilate(image, kernel, iterations=1)
    plt.subplot(2, 3, 3)
    plt.title('Dilation')
    plt.imshow(dilation, cmap='gray')
    plt.axis('off')
    # Apply Opening (Erosion followed by Dilation)
    opening = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
    plt.subplot(2, 3, 4)
    plt.title('Opening')
    plt.imshow(opening, cmap='gray')
    plt.axis('off')

    # Apply Closing (Dilation followed by Erosion)
    closing = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
    plt.subplot(2, 3, 5)
    plt.title('Closing')
    plt.imshow(closing, cmap='gray')
    plt.axis('off')

    # Apply Morphological Gradient (Difference between Dilation and Erosion)
    gradient = cv2.morphologyEx(image, cv2.MORPH_GRADIENT, kernel)
    plt.subplot(2, 3, 6)
    plt.title('Morphological Gradient')
    plt.imshow(gradient, cmap='gray')
    plt.axis('off')

    # Show all results
    plt.tight_layout()
    plt.show()

# Path to the input image
image_path = 'input_image.jpg'

# Apply morphological operations
apply_morphological_operations(image_path)
```

Expt No: 10 Write a python script to extract texture and color features of an image.

```
import cv2
import numpy as np
from skimage.feature import graycomatrix, graycoprops
from skimage.color import rgb2gray
import matplotlib.pyplot as plt
# Function to extract color histogram features
def extract_color_histogram(image):
 # Convert the image to HSV color space
 hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
 # Calculate the histogram for each channel (Hue, Saturation, Value)
 hist_hue = cv2.calcHist([hsv], [0], None, [256], [0, 256])
 hist_saturation = cv2.calcHist([hsv], [1], None, [256], [0, 256])
 hist_value = cv2.calcHist([hsv], [2], None, [256], [0, 256])
 # Normalize the histograms
 hist_hue = cv2.normalize(hist_hue, hist_hue).flatten()
 hist_saturation = cv2.normalize(hist_saturation, hist_saturation).flatten()
 hist_value = cv2.normalize(hist_value, hist_value).flatten()
 return hist_hue, hist_saturation, hist_value
# Function to extract texture features using GLCM
def extract_texture_features(image):
 # Convert the image to grayscale
 gray_image = rgb2gray(image)
 # Compute GLCM (Gray Level Co-occurrence Matrix)
 glcm = graycomatrix((gray_image * 255).astype(np.uint8),
 distances=[1], angles=[0],
 levels=256, symmetric=True, normed=True)
 # Extract texture properties: contrast, dissimilarity, homogeneity, and energy
 contrast = graycoprops(glcm, 'contrast')[0, 0]
 dissimilarity = graycoprops(glcm, 'dissimilarity')[0, 0]
 homogeneity = graycoprops(glcm, 'homogeneity')[0, 0]
 energy = graycoprops(glcm, 'energy')[0, 0]
 return contrast, dissimilarity, homogeneity, energy
# Load the image
image_path = 'C:/Users/ADMIN/Pictures/img1.jfif'
image = cv2.imread(image_path)
# Extract color features
hue_hist, sat_hist, val_hist = extract_color_histogram(image)
# Extract texture features
texture_features = extract_texture_features(image)
# Output results
print("Color Features (Histograms):")
print("Hue Histogram:", hue_hist)
print("Saturation Histogram:", sat_hist)
print("Value Histogram:", val_hist)
print("\nTexture Features (GLCM):")
print("Contrast:", texture_features[0])
print("Dissimilarity:", texture_features[1])
print("Homogeneity:", texture_features[2])
print("Energy:", texture_features[3])
# Optional: Plot the color histograms
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.plot(hue_hist, color='r')
plt.title('Hue Histogram')
plt.subplot(1, 3, 2)
plt.plot(sat_hist, color='g')
plt.title('Saturation Histogram')
plt.subplot(1, 3, 3)
plt.plot(val_hist, color='b')
plt.title('Value Histogram')
plt.show()
```



```
Texture Features (GLCM):
Contrast: 150.6499735029147
Dissimilarity: 5.099761526232115
Homogeneity: 0.38755492328945096
Energy: 0.028467579643727446a
```

Practical 11. Write a Python code to perform character recognition

```
import cv2
import pytesseract
#pip install pytesseract opencv-python
# Specify Tesseract-OCR executable path (Required for Windows)
# Uncomment and modify the line below for Windows
# pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

def perform_character_recognition(image_path):
    """Perform character recognition on the given image.
    :param image_path: Path to the input image
    :return: Recognized text """
    # Load the image
    image = cv2.imread(image_path)

    # Convert the image to grayscale (improves OCR accuracy)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Perform thresholding to improve contrast (optional)
    _, thresh_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Use pytesseract to extract text
    extracted_text = pytesseract.image_to_string(thresh_image, lang='eng')
    return extracted_text
# Path to the image containing text
image_path = 'text_image.jpg'
# Perform OCR
recognized_text = perform_character_recognition(image_path)
# Output the recognized text
print("Recognized Text:")
print(recognized_text)
# Optional: Display the processed image
cv2.imshow('Processed Image', cv2.imread(image_path))
cv2.waitKey(0)
cv2.destroyAllWindows()
```