| **EXERCISE NO:4**  **DATE:06/02/2025** | **IMAGE FILTERING** |
| --- | --- |

**AIM:**

To implement various image processing techniques using OpenCV, including image reading, color space conversions, histogram equalization, manual grayscale conversion, histogram calculation, and image resizing, to enhance and analyze images effectively.

**NATURE IMAGE**

**ALGORITHM:**

1. Import OpenCV and necessary libraries
2. Read Image → cv2.imread()
3. Convert BGR to RGB → cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)
4. Display RGB Image
5. Convert RGB to Grayscale → cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)
6. Manual RGB to Grayscale Conversion → Compute weighted sum of channels
7. Histogram Equalization
8. Using cv2.equalizeHist()
9. Compute manually using CDF table
10. Histogram Calculation
11. Using np.ravel() and cv2.calcHist()
12. Resize Image → cv2.resize(image, (width, height))

**CODE:**

import cv2

import matplotlib.pyplot as plt

import numpy as np

image = cv2.imread("Downloads/cv2.jpg")

image\_rgb= cv2.cvtColor(image,cv2.COLOR\_BGR2RGB)

plt.title("original image")

plt.imshow (image\_rgb)

plt.axis ("off")

plt.show ()

**OUTPUT:**



**CODE:**

image\_gray = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

plt.title("Original Image")

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

plt.axis('off')

plt.show()

**OUTPUT:**



**IMAGE FILTERING:**

**CODE:**

def apply\_filter(image, filter\_mask):

if len(image.shape) == 3:

image = np.mean(image, axis=2)

image\_height, image\_width = image.shape

filter\_height, filter\_width = filter\_mask.shape

pad\_height, pad\_width = filter\_height // 2, filter\_width // 2

padded\_image = np.pad(image, ((pad\_height, pad\_height), (pad\_width, pad\_width)), mode='constant', constant\_values

filtered\_image = np.zeros\_like(image, dtype=float)

for i in range(image\_height):

for j in range(image\_width):

region = padded\_image[i:i + filter\_height, j:j + filter\_width]

filtered\_image[i, j] = np.sum(region \* filter\_mask)

filtered\_image = np.clip(filtered\_image, 0, 255).astype(np.uint8)

return filtered\_image

def median\_filter(image, kernel\_size=3):

assert kernel\_size % 2 == 1

height, width = image.shape

pad = kernel\_size // 2

padded\_image = np.pad(image, pad, mode='constant', constant\_values=0)

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

# Apply median filtering

for i in range(height):

for j in range(width):

neighborhood = padded\_image[i:i+kernel\_size, j:j+kernel\_size]

filtered\_image[i, j] = np.median(neighborhood)

return filtered\_image

#box\_blur\_filter = np.ones((5, 5)) / 25

mask\_size = 25

box\_blur\_filter=np.ones((mask\_size,mask\_size),np.float32)/(mask\_size\*mask\_size)

sharpening\_filter = np.array([[ 0, -1, 0],

[-1, 5, -1],

[ 0, -1, 0]], dtype=np.float32)

# Apply sharpening filter to grayscale image

sharpened\_image = apply\_filter(image\_gray, sharpening\_filter)

smoothed\_image\_box = apply\_filter(image\_gray, box\_blur\_filter)

smoothed\_image\_median = median\_filter(image\_gray, kernel\_size=25)

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

plt.subplot(2, 3, 1)

plt.title("Original Image")

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

plt.axis('off')

plt.subplot(2, 3, 2)

plt.title("Smoothing (Box Blur)")

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

plt.axis('off')

plt.subplot(2, 3, 3)

plt.title("Sharpening Image (Laplacian)")

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

plt.axis('off')

plt.subplot(2, 3, 4)

plt.title("Smoothing (Median Filter)")

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

plt.axis('off')

plt.tight\_layout()

plt.show()

**OUTPUT:**

****

**CODE:**

plt.figure(figsize=(8, 4))

plt.subplot(2,3,1)

plt.hist(image\_gray.ravel(),bins=256,range=(0,256),color='maroon')

plt.title("Original Image Histogram")

plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.subplot(2,3,2)

plt.hist(sharpened\_image.ravel(),bins=256,range=(0,256),color='maroon')

plt.title("Sharpened Image Histogram")

plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.subplot(2,3,3)

plt.hist(smoothed\_image\_box.ravel(),bins=256,range=(0,256),color='maroon')

plt.title("Smoothed Box Image Histogram")

plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.subplot(2,3,4)

plt.hist(smoothed\_image\_median.ravel(),bins=256,range=(0,256),color='maroon')

plt.title("Median Filtered(Smoothed) Image Histogram")

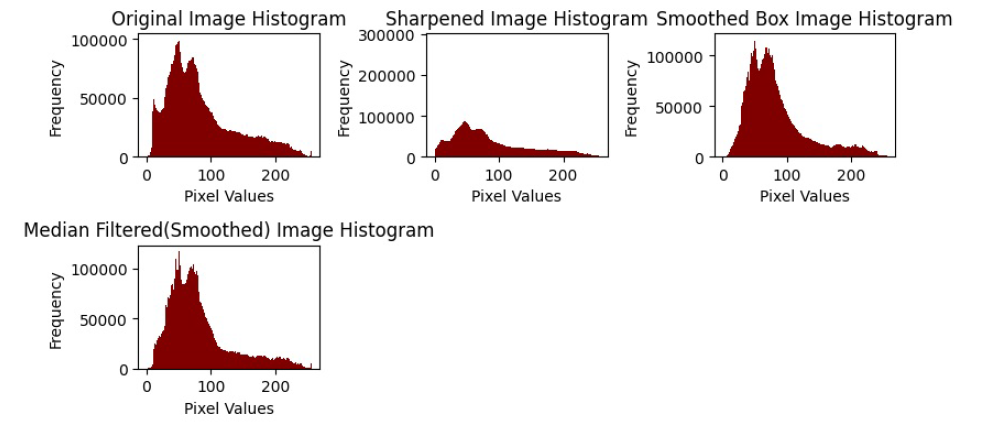
plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.tight\_layout()

plt.show()

**OUTPUT:**



**CODE:**

dft = np.fft.fft2(image\_gray)

dft\_shift = np.fft.fftshift(dft)

magnitude\_spectrum = 20\*np.log(np.abs(dft\_shift)+1)

plt.figure(figsize=(10,6))

plt.subplot(1,2,1)

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

plt.title("Original Image")

plt.axis('off')

plt.subplot(1,2,2)

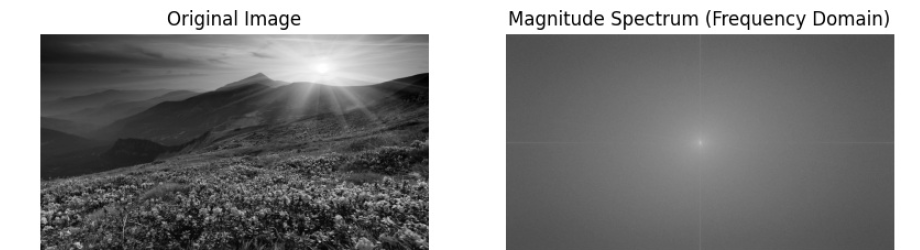
plt.imshow(magnitude\_spectrum,cmap='gray')

plt.title("Magnitude Spectrum (Frequency Domain)")

plt.axis('off')

plt.show()

**OUTPUT:**

****

**CODE:**

shifted\_image\_add = np.clip(image\_rgb + 50, 0, 255).astype(np.uint8) # Brighten

shifted\_image\_sub = np.clip(image\_rgb - 50, 0, 255).astype(np.uint8)

plt.figure(figsize=(8,4))

plt.subplot(2,3,1)

plt.hist(image\_rgb.ravel(),bins=256,range=(0,256),color='maroon')

plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.title("Original Image")

plt.subplot(2,3,2)

plt.hist(shifted\_image\_add.ravel(),bins=256,range=(0,256),color='maroon')

plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.title("Histogram of Shifted Image+50")

plt.subplot(2,3,3)

plt.hist(shifted\_image\_sub.ravel(),bins=256,range=(0,256),color='maroon')

plt.xlabel("Pixel Values")

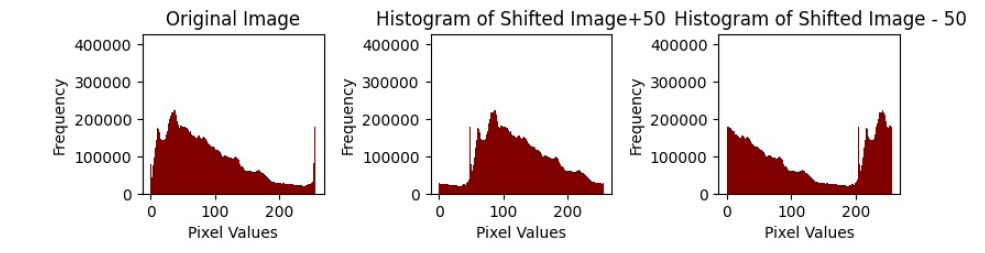
plt.ylabel("Frequency")

plt.title(" Histogram of Shifted Image - 50")

plt.tight\_layout()

plt.show()

**OUTPUT:**

****

**CODE:**

plt.subplot(2,3,1)

plt.imshow(image\_rgb)

plt.title("Original Image")

plt.axis('off')

plt.subplot(2,3,2)

plt.imshow(shifted\_image\_add)

plt.title("Shifted Image +50 ")

plt.axis('off')

plt.subplot(2,3,3)

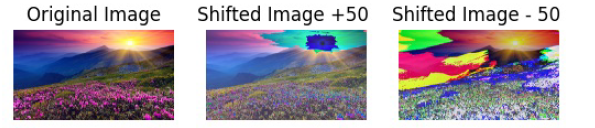
plt.imshow(shifted\_image\_sub)

plt.title("Shifted Image - 50 ")

plt.axis('off')

plt.show()

**OUTPUT:**

****

**CODE:**

I\_min = np.min(image\_gray)

I\_max = np.max(image\_gray)

stretched\_image = np.clip(((image\_gray - I\_min) \* 255) / (I\_max - I\_min), 0, 255).astype(np.uint8)

#stretched\_image = contrast\_stretch(image\_gray)

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

plt.subplot(1,2,1)

plt.title("Original Image")

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

plt.axis("off")

plt.subplot(1,2,2)

plt.title("Contrast Stretched Image")

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

plt.axis("off")

plt.tight\_layout()

plt.show()

**OUTPUT:**

****

**CODE:**

plt.figure(figsize=(8,4))

plt.subplot(2,3,1)

plt.hist(image\_gray.ravel(),bins=256,range=(0,256),color='maroon')

plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.title("Original Image")

plt.subplot(2,3,2)

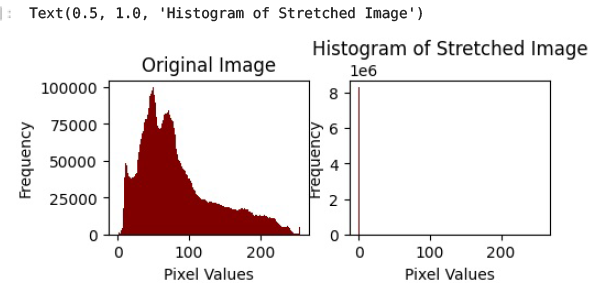
plt.hist(stretched\_image.ravel(),bins=256,range=(0,256),color='maroon')

plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.title("Histogram of Stretched Image")

**OUTPUT:**

****

**CODE:**

threshold = 127

binary\_image = np.zeros\_like(image\_gray,dtype=np.uint8)

binary\_image[image\_gray>threshold] = 255

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

plt.title("Original Image")

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

plt.axis("off")

plt.subplot(1, 2, 2)

plt.title(f"Binary Image (Threshold = {threshold})")

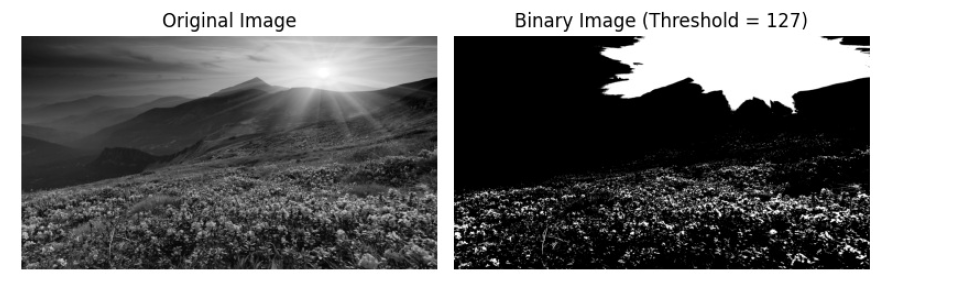
plt.imshow(binary\_image, cmap='gray')

plt.axis("off")

plt.tight\_layout()

plt.show()

**OUTPUT:**

****

**CODE:**

image\_negative = 255 - image\_rgb

plt.subplot(1,2,1)

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

plt.title("Original Image")

plt.axis('off')

plt.subplot(1,2,2)

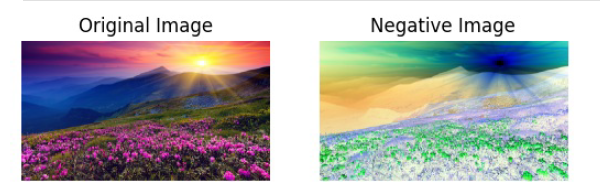
plt.axis('off')

plt.title("Negative Image")

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

plt.show()

**OUTPUT:**

****

**CODE:**

plt.figure(figsize=(10,6))

plt.subplot(2,2,1)

plt.hist(image\_rgb.ravel(),bins=256,range=(0,256),color='maroon')

plt.title("Original Image")

plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

plt.subplot(2,2,2)

plt.hist(image\_negative.ravel(),bins=256,range=(0,256),color='maroon')

plt.title("Negative Image")

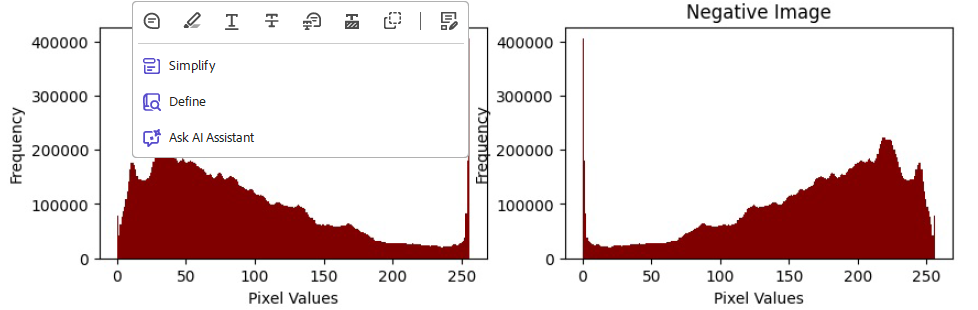
plt.xlabel("Pixel Values")

plt.ylabel("Frequency")

#plt.tight\_layout()

plt.show()

**OUTPUT:**

****

**RESULT:**

**We have successfully executed the code…**