EX.No: 3	Execution of matrix operations using NumPy to simulate basic image processing functions
DATE:	

AIM:

To implement and understand fundamental image processing operations using NumPy in Python.

ALGORITHM:

- **Step 1:** Start the program.
- **Step 2:** Import the required libraries:

  "NumPy" is used for numerical operations and matplotlib.pyplot is used for displaying images.

- **Step 3:** Load the input image and convert it to a grayscale NumPy array.
- **Step 4:** Display the original image.
- **Step 5:** Perform the following image processing operations:

a) **Image Inversion:**

- For each pixel value, subtract it from 255 to get the negative image.

b) **Thresholding:**

- Choose a threshold value (e.g., 128).
- If a pixel value > threshold, set it to 255; else set it to 0.

c) **Image Rotation (90° Clockwise):**

- Transpose the matrix and reverse the order of rows.

d) **Image Scaling (Downsampling):**

- Select every alternate row and column to reduce resolution.

e) **Brightness Adjustment:**

- Add a constant value (e.g., +50) to each pixel.
- Clip the result to stay within the range 0 to 255.

f) **Blurring (Mean Filter):**

- Define a 3×3 averaging kernel.
- Apply 2D convolution of the kernel over the image.

g) **Edge Detection (Sobel Filter):**

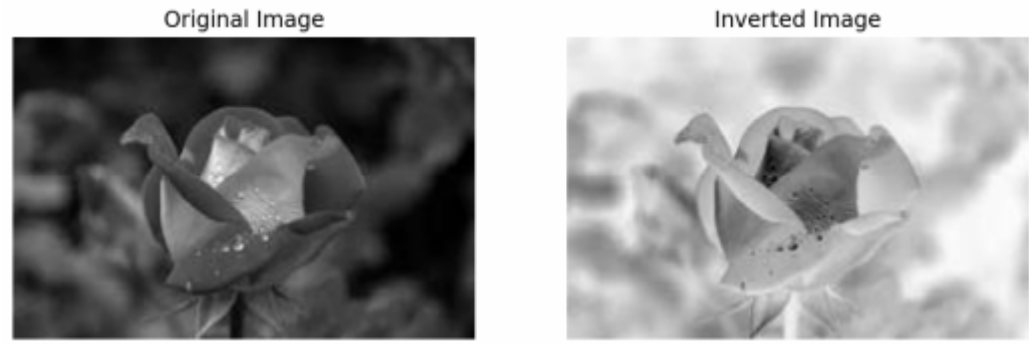
- Define horizontal and vertical Sobel kernels.
- Convolve both kernels with the image.
- Compute the gradient magnitude using both results.

- **Step 6:** Display each processed output image.
- **Step 7:** End the program.

PROGRAM:1 **Image Inversion using NumPy**

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
# Step 1: Load the image and convert it to grayscale
img = Image.open('example.jpg').convert('L')  # 'L' for grayscale
img_array = np.array(img)
# Step 2: Perform image inversion
inverted_img = 255 - img_array
# Step 3: Display original and inverted images
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img_array, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(inverted_img, cmap='gray')
plt.title('Inverted Image')
plt.axis('off')
plt.show()
```

OUT PUT



PROGRAM:2 Code for Thresholding using NumPy

```
import numpy as np
import matplotlib.pyplot as plt
# Step 1: Create a simulated grayscale image (gradient pattern)
img_array = np.tile(np.arange(0, 256, dtype=np.uint8), (100, 1))  # shape: (100, 256)
# Step 2: Apply thresholding using NumPy
threshold = 150
binary_img = np.where(img_array > threshold, 255, 0).astype(np.uint8)
# Step 3: Display original and thresholded images
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img_array, cmap='gray')
plt.title("Original Image")
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(binary_img, cmap='gray')
plt.title(f"Thresholded Image (T = {threshold})")
plt.axis('off')
plt.show()
```

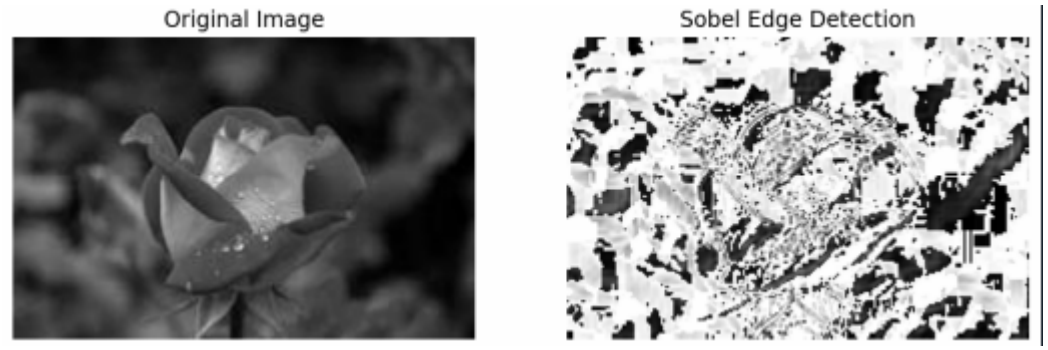
OUT PUT

PROGRAM:3 Edge Detection using Sobel Filter

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import convolve
# Step 1: Load and convert image to grayscale
img = Image.open('example.jpg').convert('L')
img_array = np.array(img)
# Step 2: Define Sobel kernels
sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])
sobel_y = np.array([[-1, -2, -1],
                    [ 0,  0,  0],
                    [ 1,  2,  1]])
# Step 3: Apply convolution with Sobel filters
gx = convolve(img_array, sobel_x)
gy = convolve(img_array, sobel_y)
# Step 4: Compute gradient magnitude
sobel_edges = np.hypot(gx, gy)
sobel_edges = np.clip(sobel_edges, 0, 255).astype(np.uint8)
# Step 5: Display original and Sobel edge image
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img_array, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(sobel_edges, cmap='gray')
plt.title('Sobel Edge Detection')
```

```
plt.axis('off')
plt.show()
```

OUT PUT



Result:

Hence using NumPy, we simulated essential image processing techniques by directly manipulating pixel matrices. This forms the basis for more advanced image analysis and computer vision tasks.

EXERCISE:

1. Constuct a Python program to write a Code for Thresholding using NumPy at 90° clockwise rotation.
2. Construct a Python program to Image Blurring using a Mean Filter
3. Write Python program to Image Brightness Adjustment using NumPy.