SCHOOL OF COMPUTER SCIENCE AND IT

DEPARTMENT OF CS & IT

BCA PROGRAMME

SEMESTER: V

SUBJECT NAME : Image Processing

SUBJECT CODE : 24BCA6IOE02

ACTIVITY 2

MINI PROJECT

Implementation of Spatial, Morphological and Thresholding Operations in Image Processing

Date of Submission: 12th February 2026

Submitted by:

23BCAR0339 – Somnath Gorai

Faculty In-Charge:

Dr. Zion Ramdinthara

Professor

CERTIFICATE

This is to certify that Ms. /Mr. **Somnath Gorai** has satisfactorily completed Activity 2 prescribed by JAIN (Deemed to be University) for the 6th Semester BCA degree course in the year 2026.

Signature of the Faculty In Charge

Assessment Sheet with Rubrics for Grading & Evaluation

Students have to complete the online courses in the given timeline and submit the report as per format given.

Sr. No.	USN No.	Student Name	Report	Presentation	Viva-Voce	Total
			5 Marks	5 Marks	5 Marks	15 Marks
1	23BCAR0339	Somnath Gorai				

ABSTRACT

Image processing plays a crucial role in enhancing, analyzing, and interpreting digital images for various real-world applications such as medical imaging, remote sensing, surveillance, and computer vision. The primary objective of this activity is to study and implement fundamental image processing techniques using spatial filtering, morphological operations, feature detection, and segmentation methods. These techniques form the foundation for understanding how digital images can be manipulated and analyzed mathematically and computationally.

In this activity, Gaussian Blur, Image Sharpening, and Unsharp Masking are applied to enhance image quality and control image smoothness. Gaussian Blur is used to reduce noise and suppress high-frequency components by convolving the image with a Gaussian kernel. Image Sharpening enhances edges and fine details by emphasizing high-frequency components, while Unsharp Masking improves clarity by combining the original image with its blurred version. These techniques demonstrate how spatial domain filters influence image appearance and detail.

Morphological image processing operations such as Dilation, Erosion, Opening, and Closing are implemented to analyze and manipulate binary images based on their shapes. Dilation and erosion modify object boundaries, while opening and closing are used for noise removal and gap filling. These operations are particularly effective in processing images containing text and structured objects.

Further, Line Detection and Point Detection techniques are applied using suitable convolution masks to identify structural features within an image. Line detection highlights linear patterns and edges, whereas point detection identifies isolated pixels and sharp intensity variations. These operations are essential for feature extraction and pattern recognition tasks.

Finally, a Global Thresholding technique is implemented to segment an image into foreground and background regions based on a predefined threshold value. This method demonstrates a simple yet effective approach to image segmentation.

All algorithms are implemented using Python with standard development tools such as Spyder, Jupyter Notebook, Google Colab, PyCharm, and Visual Studio Code. The results obtained validate the theoretical concepts and highlight the practical importance of basic image processing techniques.

Input Image:

Above image is the input image that was being used in all the operation performed bellow.

Table of Contents

CHAPTER 1: GAUSSIAN BLUR, SHARPENING AND UNSHARP MASKING

1.1 Gaussian Blur

Gaussian Blur is a linear smoothing filter used to reduce noise and minor details in an image. It works by averaging pixel values with their neighboring pixels using weights derived from the Gaussian distribution. Pixels closer to the center of the kernel have higher influence than distant pixels. This operation suppresses high-frequency components such as noise and sharp edges, resulting in a smoother image.

Kernal:

Gaussian Blur smooths an image using a weighted averaging kernel derived from the Gaussian distribution. The Gaussian kernel used is:

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Algorithm:

1. Read the input image and convert it to grayscale.
2. Add zero padding around the image borders.
3. Define the 3×3 Gaussian kernel.
4. Slide the kernel over the image pixel by pixel.
5. Multiply kernel values with corresponding image pixels.
6. Sum the results and divide by 16.
7. Assign the computed value to the output pixel.

Code:

```
import cv2
import numpy as np

# Read image
img = cv2.imread("image.png", cv2.IMREAD_GRAYSCALE)

rows, cols = img.shape

# Gaussian kernel (textbook)
kernel = np.array([[1, 2, 1],
```

```
        [2, 4, 2],
        [1, 2, 1]])

# Zero padding
padded = np.zeros((rows + 2, cols + 2), dtype=np.uint8)
padded[1:rows+1, 1:cols+1] = img

# Output image
blurred = np.zeros_like(img)

# Manual convolution
for i in range(rows):
    for j in range(cols):
        region = padded[i:i+3, j:j+3]
        value = np.sum(region * kernel) / 16
        blurred[i, j] = int(value)

# Save output
cv2.imwrite("gaussian_blur.png", blurred)
```

```python
import cv2
import numpy as np

# Read image
img = cv2.imread("image.png", cv2.IMREAD_GRAYSCALE)

rows, cols = img.shape

# Gaussian kernel (textbook)
kernel = np.array([[1, 2, 1],
                   [2, 4, 2],
                   [1, 2, 1]])

# Zero padding
padded = np.zeros((rows + 2, cols + 2), dtype=np.uint8)
padded[1:rows+1, 1:cols+1] = img

# Output image
blurred = np.zeros_like(img)

# Manual convolution
for i in range(rows):
    for j in range(cols):
        region = padded[i:i+3, j:j+3]
        value = np.sum(region * kernel) / 16
        blurred[i, j] = int(value)

# Save output
cv2.imwrite("gaussian_blur.png", blurred)
```

Code Explanation:

Output Image:

Observation:

After applying the Gaussian filter using the 3×3 kernel, the image appears smoother with a noticeable reduction in noise and minor intensity variations. Fine details and sharp edges are slightly softened due to the averaging effect of the filter. The overall brightness of the image is preserved because the kernel is normalized, making Gaussian blur effective for noise reduction without significant loss of image information.

1.2 Image Sharpening

Image sharpening is a spatial domain technique used to enhance edges and fine details in an image. It works by emphasizing high-frequency components such as edges and boundaries, making the image appear clearer and more defined. Sharpening is commonly applied after smoothing operations to restore lost details.

Kernal:

Image sharpening can be achieved using a sharpening mask that highlights intensity differences between a pixel and its neighbors.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Algorithm:

1. Read the input image and convert it to grayscale.
2. Define the sharpening kernel as shown above.
3. Convolve the kernel with the input image.
4. Enhance edges and fine details.
5. Obtain the sharpened output image.

Code:

```
import cv2
import numpy as np

# Read image
img = cv2.imread("image.png", cv2.IMREAD_GRAYSCALE)

# Sharpening kernel
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])

# Apply sharpening
sharpened = cv2.filter2D(img, -1, kernel)

# Save output
cv2.imwrite("sharpened.png", sharpened)
```

Code Explanation:

- The input image is read in grayscale format.
- A 3×3 sharpening kernel is defined to enhance edges.
- Convolution is applied using filter2D() to emphasize high-frequency components.
- The sharpened image is saved as the output.

Output Image:

Observation:

Edges and boundaries in the image appear more prominent after sharpening. Fine details are enhanced, making the image clearer compared to the original, though excessive sharpening may slightly increase noise.

1.3 Unsharp Masking

Unsharp masking is an image enhancement technique used to improve the sharpness of an image by enhancing its edges. The method works by subtracting a blurred version of the image from the original image to extract edge details, and then adding these details back to the original image. This process increases contrast along edges without significantly amplifying noise.

Formula:

Unsharp masking is mathematically represented as:

$$\textbf{Unsharp Image} = \textbf{Original Image} + (\textbf{Original Image} - \textbf{Blurred Image})$$

Algorithm:

1. Read the input image and convert it to grayscale.
2. Apply Gaussian Blur to obtain a blurred version of the image.
3. Subtract the blurred image from the original image to extract edge details.
4. Add the extracted details back to the original image.
5. Obtain the unsharp masked output image.

Code:

```
import cv2
import numpy as np

# Read image
img = cv2.imread("image.png", cv2.IMREAD_GRAYSCALE)

# Gaussian blur using textbook kernel
kernel = (1/16) * np.array([[1, 2, 1],
                    [2, 4, 2],
                    [1, 2, 1]])

blurred = cv2.filter2D(img, -1, kernel)

# Unsharp masking
unsharp = img + (img - blurred)
unsharp = np.clip(unsharp, 0, 255)

# Save output
cv2.imwrite("unsharp_mask.png", unsharp)
```

Code Explanation:

- The input image is read in grayscale format.
- Gaussian blur is applied using the standard 3×3 Gaussian kernel.
- The blurred image is subtracted from the original image to extract edge information.
- The extracted details are added back to the original image to enhance sharpness.

- The result is clipped to maintain valid pixel intensity values and saved.

Output Image:

Observation:

Edges and fine details appear sharper compared to the original image. The image clarity is improved without excessive amplification of noise, making unsharp masking effective for enhancing visual quality.

CHAPTER 2: MORPHOLOGICAL OPERATIONS

2.1 Dilation

Dilation is a morphological operation used to expand foreground regions in a binary image. Using a vertical structuring element, dilation strengthens vertical connections by setting a pixel to foreground if at least one pixel under the kernel matches. This operation is useful for filling gaps and connecting broken vertical structures.

Kernel:

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Rules:

- **FIT** (When all the pixels of input image match with above kernel) → Output pixel = 1
- **HIT** (When some pixels of input image match with above kernel) → Output pixel = 1
- **MISS** (When none of the pixels of input image match with above kernel match) → Output pixel = 0

Algorithm:

1. Read the binary input image.
2. Select the vertical structuring element.
3. Slide the kernel vertically over the image.
4. If all or some pixels match the kernel, set output pixel to 1.
5. Otherwise, set output pixel to 0.

CONCLUSION

REFERENCES