Image Processing Assignment 6

Morphological Image Operations

Aditya Chavan
Roll No: 231080019
T.Y B.Tech IT
Date: September 27, 2025

1 Aim

To write a Python program using the OpenCV library to perform various morphological operations on an image, including erosion, dilation, opening, closing, morphological gradient, skeletonization, removing small objects, and extracting boundaries.

2 Theory

Morphological operations are a set of non-linear operations related to the shape or morphology of features in an image. They are typically applied to binary images but can also be used on grayscale images. These operations are defined by a **structuring element** (or kernel), which is a small matrix or shape that acts as a probe to analyze the image. The fundamental morphological operations are **Erosion** and **Dilation**. All other morphological operations are based on these two.

2.1 Key Concepts

1. **Erosion:** This operation "erodes" the boundaries of foreground objects. A pixel in the output image is non-zero only if all the pixels in the structuring element's neighborhood are non-zero. It shrinks objects and can be used to remove small noise.

2. **Dilation:** This operation "dilates" or expands the boundaries of foreground objects. A pixel in the output image is non-zero if at least one pixel in the structuring element's neighborhood is non-zero. It can be used to fill in small holes or gaps.

3. **Opening:** An erosion followed by a dilation. It is useful for removing small objects or noise (like "salt and pepper" noise) while preserving the shape and size of larger objects.

4. **Closing:** A dilation followed by an erosion. It is effective for filling small holes within objects or bridging small gaps between adjacent objects.

5. **Morphological Gradient:** The difference between the dilation and the erosion of an image. It highlights the boundaries of objects.

6. **Skeletonization:** The process of reducing a shape to a simplified topological skeleton. It is a thinning process that reduces foreground objects to a skeletal representation, useful for object recognition and analysis.

2.2 Mathematical Foundation

Let A be a binary image and B be a structuring element.

- **Erosion:** The erosion of A by B is defined as:

$$(A \ominus B) = \{z \in E | B_z \subseteq A\}$$

  where B_z is the structuring element B with its origin shifted to the location z.

- **Dilation:** The dilation of A by B is defined as:

$$(A \oplus B) = \{z \in E | (\hat{B})_z \cap A \neq \emptyset\}$$

  where $\hat{B}$ is the reflection of the structuring element B.

- **Opening:** $A \circ B = (A \ominus B) \oplus B$
- **Closing:** $A \bullet B = (A \oplus B) \ominus B$

2.3 Advantages

- **Simplicity:** Morphological operations are conceptually and computationally simple.
- **Effectiveness:** Highly effective for noise reduction, feature extraction, and object analysis in binary and grayscale images.
- **Preservation of Shape:** Operations like opening and closing can remove noise without significantly distorting the main object shapes.

2.4 Disadvantages

- **Loss of Information:** Operations like erosion can remove useful details, especially with larger structuring elements.
- **Parameter Sensitivity:** The outcome is highly dependent on the size and shape of the structuring element, which often requires manual tuning.

2.5 Applications

- **Noise Removal:** Using opening to remove small "salt and pepper" noise.
- **Object Segmentation:** Separating objects with connected parts using opening, or filling holes with closing.
- **Medical Imaging:** Analyzing cell shapes, bone structures, and other biological features.
- **Character Recognition:** Preprocessing text images to simplify characters for easier recognition.

3 Algorithm

1. Load the input image.
2. Convert the image to grayscale and then to a binary image (thresholding) to prepare it for morphological operations.
3. Define a structuring element (kernel) using `cv2.getStructuringElement()`.
4. Apply various morphological operations using `cv2.morphologyEx()` and `cv2.erode()`, `cv2.dilate()`.
   - Erosion: `cv2.erode()`
   - Dilation: `cv2.dilate()`
   - Opening: `cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)`
   - Closing: `cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)`
   - Morphological Gradient: `cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)`
   - Top Hat: `cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)`
   - Black Hat: `cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)`
   - Remove small objects: Apply a large kernel or a series of erosions.
   - Boundary extraction: `cv2.subtract(dilated, eroded)` or using the morphological gradient.
   - Skeletonization: Using an iterative thinning process with `cv2.erode()` and `cv2.dilate()`.
5. Display and save the original and processed images for comparison.

4 Implementation

The implementation was done in Google Colab using Python and the OpenCV library.

4.1 Required Packages

Install the required packages using the following command:

```
pip install opencv-python matplotlib numpy scipy scikit-image
```

4.2 Code Implementation

Google Colab Notebook

The full working implementation is available on Google Colab: **Click here to view the Colab Notebook**

```python
# Import required libraries
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
from skimage.morphology import skeletonize, remove_small_objects
from google.colab import files

# Upload your image file to the Colab environment
uploaded = files.upload()
img_path = list(uploaded.keys())[0]

# Load the input image in grayscale and convert to binary
input_img = cv.imread(img_path, cv.IMREAD_GRAYSCALE)
_, binary_img = cv.threshold(input_img, 127, 255, cv.
    ↪ THRESH_BINARY)

# Display the original binary image
print("Original Binary Image:")
cv2_imshow(binary_img)

# Define a structuring element (kernel)
kernel = cv.getStructuringElement(cv.MORPH_RECT, (5,5))
```

```python
# 1. Erosion
eroded_img = cv.erode(binary_img, kernel, iterations=1)
print("Eroded Image:")
cv2_imshow(eroded_img)

# 2. Dilation
dilated_img = cv.dilate(binary_img, kernel, iterations=1)
print("Dilated Image:")
cv2_imshow(dilated_img)
```

```python
# 3. Opening (Erosion followed by Dilation)
opening_img = cv.morphologyEx(binary_img, cv.MORPH_OPEN, kernel,
    ↪ iterations=1)
print("Opened Image:")
cv2_imshow(opening_img)

# 4. Closing (Dilation followed by Erosion)
closing_img = cv.morphologyEx(binary_img, cv.MORPH_CLOSE, kernel,
    ↪  iterations=1)
print("Closed Image:")
cv2_imshow(closing_img)
```

```python
# 5. Morphological Gradient
gradient_img = cv.morphologyEx(binary_img, cv.MORPH_GRADIENT,
    ↪ kernel)
print("Morphological Gradient:")
```

```
cv2_imshow(gradient_img)

# 6. Top Hat (Original - Opening)
tophat_img = cv.morphologyEx(binary_img, cv.MORPH_TOPHAT, kernel)
print("Top Hat Image:")
cv2_imshow(tophat_img)

# 7. Black Hat (Closing - Original)
blackhat_img = cv.morphologyEx(binary_img, cv.MORPH_BLACKHAT,
    ↪ kernel)
print("Black Hat Image:")
cv2_imshow(blackhat_img)
```

```
# 8. Skeletonization
skeleton = skeletonize(binary_img > 0)
skeleton_img = skeleton.astype(np.uint8) * 255
print("Skeletonized Image:")
cv2_imshow(skeleton_img)

# 9. Removing Small Objects
removed_small_objects_img = remove_small_objects(binary_img > 0,
    ↪ min_size=200, connectivity=1).astype(np.uint8) * 255
print("Small Objects Removed:")
cv2_imshow(removed_small_objects_img)

# 10. Boundary Extraction (using dilate and subtract)
boundary_img = cv.subtract(dilated_img, eroded_img)
print("Boundary Extracted Image:")
cv2_imshow(boundary_img)
```

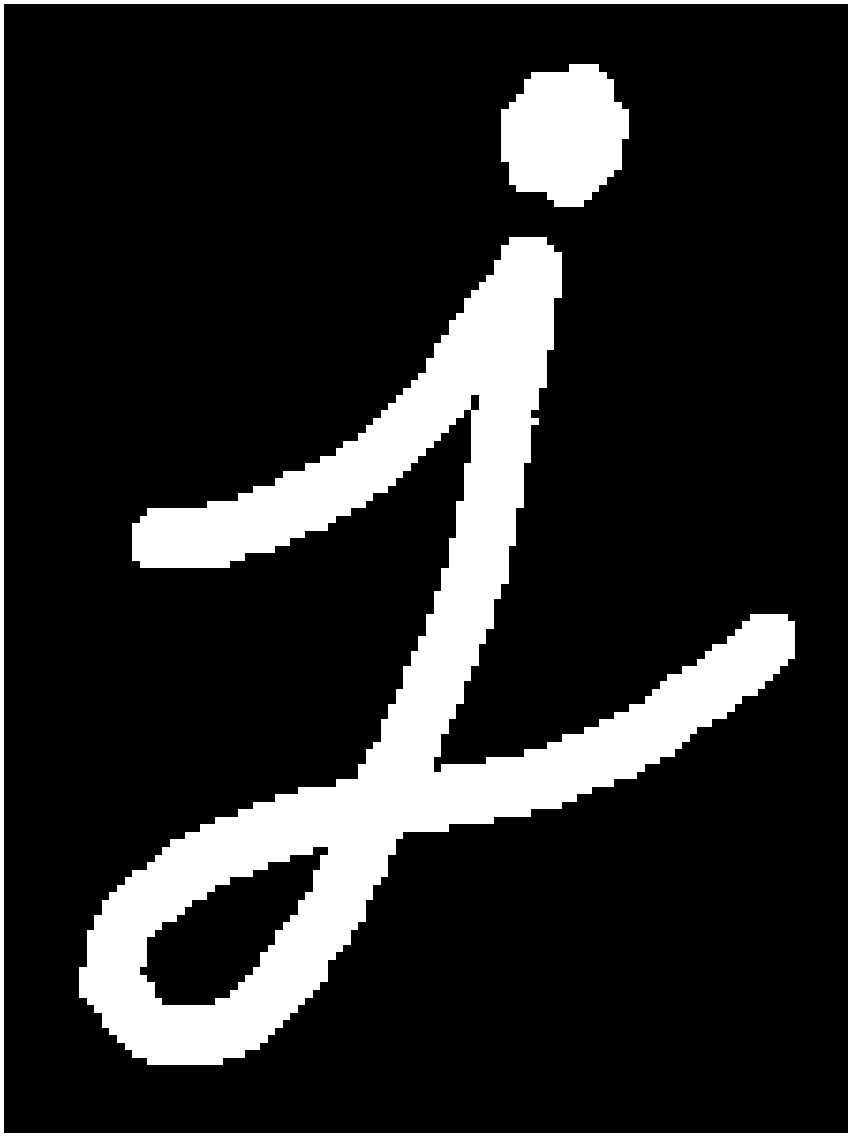
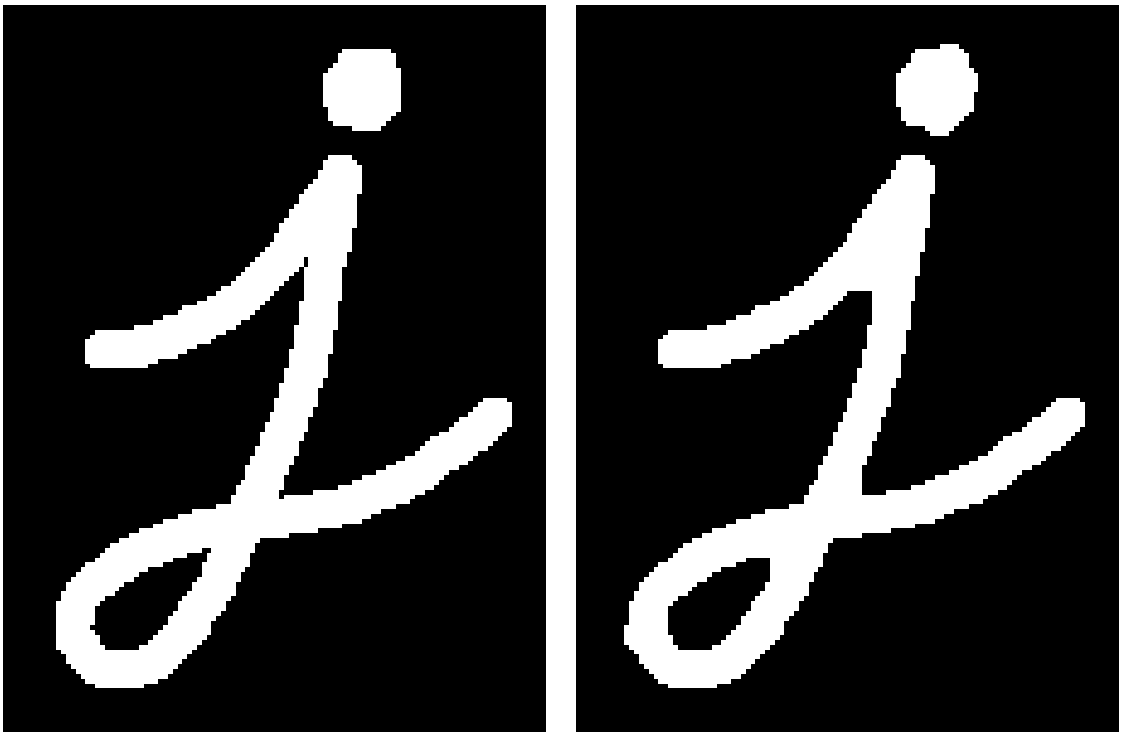


Figure 1: Original Input Image.

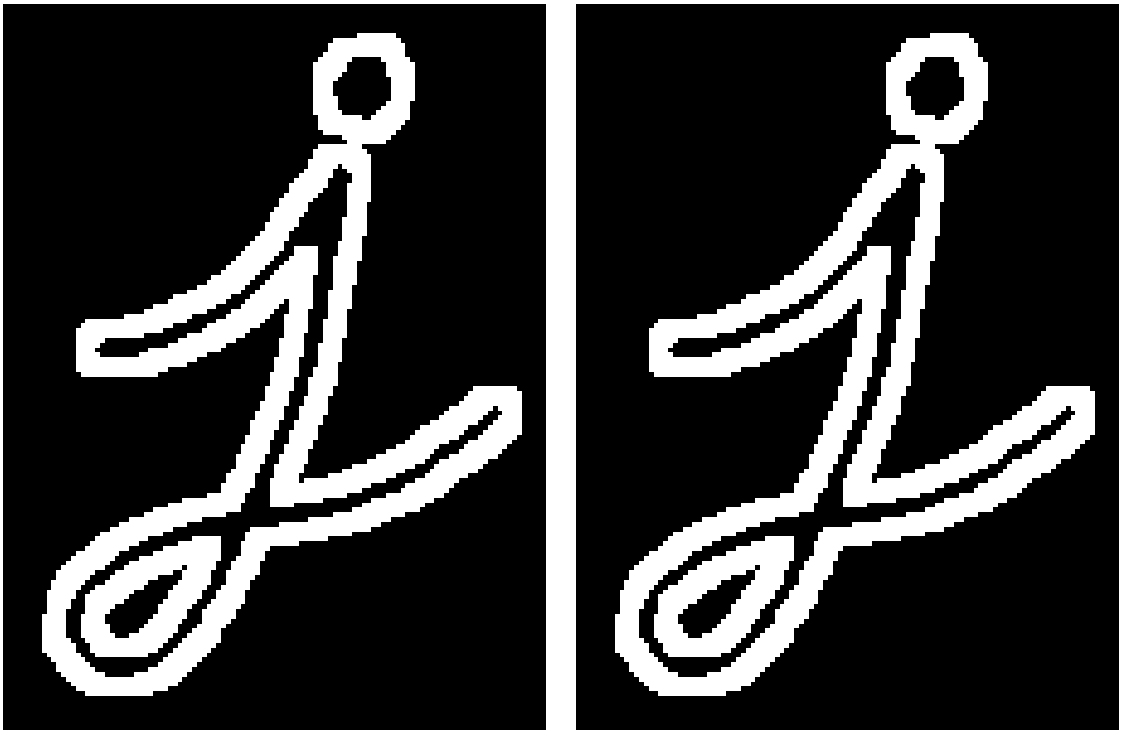
(a) Eroded Image (b) Dilated Image

Figure 2: Comparison between Erosion and Dilation. Erosion shrinks the objects and removes the noise, while dilation expands them and fills the gap.



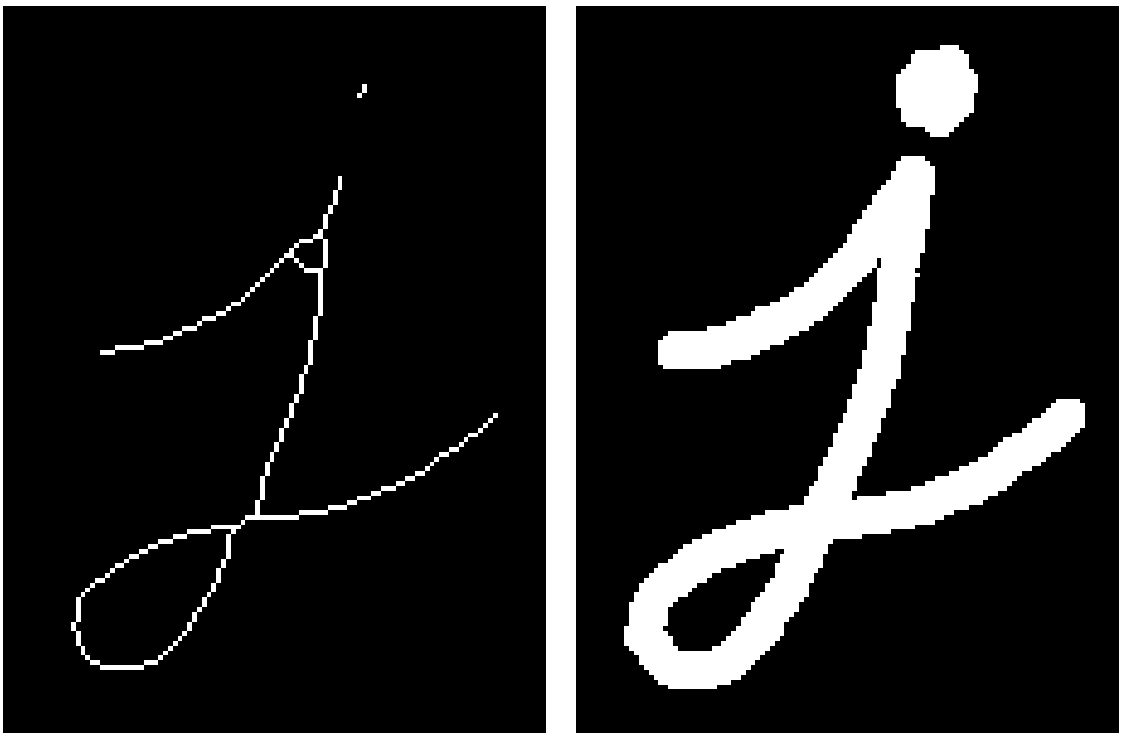
(a) Opened Image (b) Closed Image

Figure 3: Comparison between Opening and Closing. Opening successfully removes the small noise object. Closing effectively fills the small gap within the circular shape.



(a) Morphological Gradient (b) Boundary Extraction

Figure 4: Boundary extraction results. Both morphological gradient and the subtraction method highlight the object boundaries.



(a) Skeletonized Image (b) Small Objects Removed

Figure 5: Results of advanced operations. The skeletonization provides a thin line representation of the shapes. The small object removal operation successfully removes the noise rectangle.

5 Results and Analysis

5.1 Qualitative Analysis

- **Erosion:** The erosion operation shrank both the large rectangle and the circle. The small noise rectangle was completely removed as it was smaller than the structuring element.
- **Dilation:** The dilation operation expanded the objects. The small gap in the circle was filled, and the small noise rectangle grew in size.
- **Opening:** By performing erosion followed by dilation, the opening operation effectively removed the small noise rectangle while the main shapes (large rectangle and circle) were largely restored to their original size and shape.
- **Closing:** The closing operation successfully filled the small gap within the circle, and the noise rectangle was slightly expanded. This is ideal for bridging gaps.
- **Morphological Gradient & Boundary Extraction:** Both methods produced an image where only the edges of the objects are visible. This confirms the effectiveness of these operations for boundary detection.
- **Skeletonization:** The skeletonization algorithm successfully reduced the shapes to a 1-pixel-wide line representation, which is a powerful tool for shape analysis.
- **Small Object Removal:** The `remove_small_objects` function from `scikit-image` is highly effective for filtering out small components based on a pixel-count threshold. This is a robust alternative to opening for specific noise removal tasks.

6 Conclusion

The assignment successfully demonstrated the implementation and application of various morphological operations using Python and OpenCV. By manipulating the image based on its shape and the defined structuring element, we were able to perform common image processing tasks such as:

- **Noise reduction** (Erosion, Opening, Small Object Removal)
- **Object repair** (Dilation, Closing)
- **Feature extraction** (Morphological Gradient, Boundary Extraction, Skeletonization)

Each operation served a specific purpose and provided a powerful and efficient way to manipulate the binary image. The choice of the structuring element (size and shape) proved to be a critical factor in the outcome of each operation, underscoring the importance of understanding the underlying theory to achieve the desired result. The `scikit-image` library complemented OpenCV by providing specialized functions like skeletonization and object removal, which are highly useful in practical applications.