Student: Aditya Sandhu

University: Colorado State University Global

Course: 25FC - CSC515 - 1 [Module 5 – Morphology]

Critical Thinking Assignment [OpenCV in Morphology Operations for Handwritten Text Enhancement]

Instructor: Dr Dong Nguyen

Date – 10/8/2025

GIT LINKS

Document Link – [25FC-CSC515-1/MODULE5/ csc515-1-module5-critical-thinking-aditya-sandhu00.docx at main · 65AR645ASAN/25FC-CSC515-1](https://github.com/65AR645ASAN/25FC-CSC515-1/blob/main/MODULE5/%20csc515-1-module5-critical-thinking-aditya-sandhu00.docx)

Python File – [25FC-CSC515-1/MODULE5/ csc515-1-module5-critical-thinking-aditya-sandhu03.py at main · 65AR645ASAN/25FC-CSC515-1](https://github.com/65AR645ASAN/25FC-CSC515-1/blob/main/MODULE5/%20csc515-1-module5-critical-thinking-aditya-sandhu03.py)

Morphology in the context of image processing refers to a set of operations that process images based on their shapes. It works by probing an image with a structuring element, a (kernel) which is a small shape or mask. It modifies the original image depending on how this shape fits or overlaps with the image’s features. Morphological operations are primarily used on binary images (black and white) but can also be applied to grayscale images. Binary images contain only two-pixel values, typically 0 (black) and 255 (white) which represent the complete absence or presence of a feature, such as text or shapes. They are often used for object detection and morphological processing. Grayscale images represent intensity variations using shades of gray, where pixel values range from 0 (black) to 255 (white), capturing brightness information without any color. They help in refining shapes, removing noise, filling gaps, and extracting structures.

Conceptually, an image can be imagined as a grid of 1s (white) and 0s (black), where morphological operations modify these values to refine shapes and structures. Erosion removes the outer layers of 1s, effectively shrinking objects in the image. Dilation adds layers of 1s around the shapes, making them thicker or more connected. Opening, which combines erosion followed by dilation, smooths object boundaries and removes small white specks or noise. Conversely, Closing, which performs dilation followed by erosion, fills small black holes and gaps inside white areas. These operations have wide-ranging applications such as enhancing handwritten text, reducing noise, detecting edges, analyzing shapes, and preprocessing images for segmentation tasks.

Figure 1 - Python script demonstrating morphological operations (dilation, erosion, opening, and closing) on a scanned image of handwritten text using OpenCV to enhance readability and prepare the image for Handwritten Text Recognition (HTR) or Optical Character Recognition (OCR).

A screenshot of a computer program

AI-generated content may be incorrect.

Importing Required Libraries - The script begins by importing the essential Python libraries used for image processing and visualization. cv2 (OpenCV) provides functions for reading, processing, and displaying images. matplotlib.pyplot (as plt) allows visualization of intermediate and final image results. These libraries together form the foundation for performing morphological operations efficiently.

The descriptive comment block explains the purpose of the script. It states that the program demonstrates morphological operations, specifically dilation, erosion, opening, and closing on a scanned image of handwritten text. The main objective is to enhance handwritten text for Handwriting Recognition (HWR) or Optical Character Recognition (OCR) by reducing noise, smoothing boundaries, and filling gaps in characters. Users are instructed to replace 'handwritten\_sticky.jpg' with their own scanned image file path.

original = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

This code simplifies processing by converting the image into shades of gray (values between 0 and 255), removing unnecessary color information while retaining brightness and texture details. This is an essential preprocessing step before performing binary thresholding or morphological transformations. Its Greyscale creation.

*if* original *is None*:  
 *raise* FileNotFoundError(f"Image not found at {img\_path}. "  
 f"Please provide a valid path to a scanned handwritten sticky note image.")

Here we have an if statement that is catch code for missing images.

*# Apply binary thresholding to create a binary image (dark text = 0, light background = 255)*\_, binary = cv2.threshold(original, 127, 255, cv2.THRESH\_BINARY\_INV)

This line converts the grayscale image into a “binary image” by applying a threshold value of “127”, where pixel intensities below this value become black (0) and those above become white (255). The `cv2.THRESH\_BINARY\_INV` flag inverts the result, making the “handwritten text white on a black background” to improve visibility for morphological operations.

# Define a structuring element (3x3 rectangle for basic enhancement)

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))

This line creates a 3×3 rectangular structuring element (kernel) used to probe and modify the image during morphological operations. It defines the shape and size of the neighborhood that influences how pixels are added or removed when performing dilation, erosion, opening, or closing.

# Apply morphological operations

dilated = cv2.dilate(binary, kernel, iterations=1) # Expands text lines

eroded = cv2.erode(binary, kernel, iterations=1) # Shrinks text lines, removes thin noise

opened = cv2.morphologyEx(binary, cv2.MORPH\_OPEN, kernel) # Erosion then dilation: removes noise, separates characters

closed = cv2.morphologyEx(binary, cv2.MORPH\_CLOSE, kernel) # Dilation then erosion: fills gaps, connects broken parts

This section applies the four main morphological operations to the binary image using the defined kernel. First, is dilation (cv2.dilate) expands bright regions, making text lines thicker and connecting nearby components. Then there is erosion (cv2.erode) shrinks bright regions, removing small white noise and refining thin strokes. Opening (cv2.morphologyEx with MORPH\_OPEN) performs erosion followed by dilation to eliminate small noise while preserving text structure and Closing (cv2.morphologyEx with MORPH\_CLOSE) performs dilation followed by erosion to fill small gaps and join broken or incomplete characters.

Figure 2 - Visualization of morphological operations applied to a handwritten text image using a 2x3 subplot grid in OpenCV and Matplotlib, displaying the original grayscale, binary thresholded, dilated, eroded, opened, and closed images for comparison and analysis.

A computer screen with colorful text

AI-generated content may be incorrect.

This section visualizes the results of all morphological operations in a 2x3 grid using Matplotlib, allowing easy comparison between the original, binary, and processed images. Each subplot displays a different stage. grayscale, thresholded, dilated, eroded, opened, and closed, for visual analysis of enhancement quality. It then applies a final closing operation to the opened image for combined noise removal and gap filling. Lastly, the enhanced image is saved as 'enhanced\_handwritten.jpg' for later use in OCR or handwriting recognition.

*# Visualize results in a 2x3 subplot grid*fig, axes = plt.subplots(2, 3, figsize=(15, 10))  
images = [original, binary, dilated, eroded, opened, closed]  
titles = ['Original Grayscale', 'Binary Thresholded', 'Dilation', 'Erosion', 'Opening', 'Closing']

This piece of code creates a 2x3 subplot grid to display different stages of image processing side by side for visual comparison. It organizes the images and corresponding titles, “original, binary thresholded, dilated, eroded, opened, and closed”, into labeled subplots for clarity.

*for* i, (ax, img, title) *in* enumerate(zip(axes.flat, images, titles)):  
 ax.imshow(img, cmap='gray')  
 ax.set\_title(title)  
 ax.axis('off')

Then, this for loop iterates through each subplot, image, and title simultaneously using zip(). For each image, it displays it in grayscale (cmap='gray'), sets an appropriate title for identification, and hides the axis lines for a cleaner visual presentation.

plt.tight\_layout()  
plt.show()

plt.tight\_layout() adjusts spacing between subplots for a neat display, and plt.show() renders the visualization.   
*# Optional: Save enhanced image (e.g., after opening + closing for combined enhancement)*enhanced = cv2.morphologyEx(opened, cv2.MORPH\_CLOSE, kernel)  
cv2.imwrite('enhanced\_handwritten.jpg', enhanced)  
print("Enhanced image saved as 'enhanced\_handwritten.jpg'")

The final lines apply a closing operation to the previously opened image for smoother enhancement, then save the result as 'enhanced\_handwritten.jpg' and print a confirmation message.

Figure 3 - Visualization of morphological processing stages applied to a handwritten note, showing the progression from the raw grayscale image through binarization, dilation (widened), erosion (narrowed), and combined opening and closing (polished) operations, illustrating how each step refines text clarity and reduces noise for improved OCR readability.

A screenshot of a graph

AI-generated content may be incorrect.

The output image illustrates the sequential stages of morphological processing applied to handwritten text. The **raw grayscale** image initially shows uneven brightness and faint writing. After **binarization**, the text becomes more defined against the background. The **widened (dilated)** and **narrowed (eroded)** stages adjust text thickness to strengthen or refine characters. Finally, the **polished open** and **polished close** stages remove residual noise and fill gaps, producing a smoother, clearer image ideal for accurate OCR recognition.

This program effectively demonstrates the role of morphological operations in improving the clarity and structure of handwritten text for OCR applications. Through sequential processing steps, “dilation, erosion, opening, and closing” , where the image becomes cleaner, smoother, and more uniform. The visual results highlight how each stage contributes to reducing noise and enhancing character definition. These techniques emphasize the importance of preprocessing in computer vision workflows. Overall, morphology serves as a fundamental tool for refining handwritten or scanned images before text recognition.

References

GeeksforGeeks. (n.d.). *OpenCV Python tutorial*. Retrieved October 8, 2025, from <https://www.geeksforgeeks.org/python/opencv-python-tutorial/>

PythonGeeks. (n.d.). *What is OpenCV?*. Retrieved October 8, 2025, from <https://pythongeeks.org/what-is-opencv/>

OpenCV Documentation. (n.d.). *KNN in OpenCV (Python)*. *OpenCV 3.4 Documentation*. Retrieved October 8, 2025, from <https://docs.opencv.org/3.4/d8/d4b/tutorial_py_knn_opencv.html>

HandwritingOCR. (n.d.). *How to convert handwriting to text using Python*. Retrieved October 8, 2025, from <https://www.handwritingocr.com/handwriting-to-text/how-to-convert-handwriting-to-text-using-python>