6 Steps to Handwritten Text Segmentation using Image Processing

GROUP 7:

1. NUR HAZIRAH BINTI AZMAN [076506]
2. NURUL ATIEQAH AMIERA BINTI IBRAHIM [079443]

Problem Statement

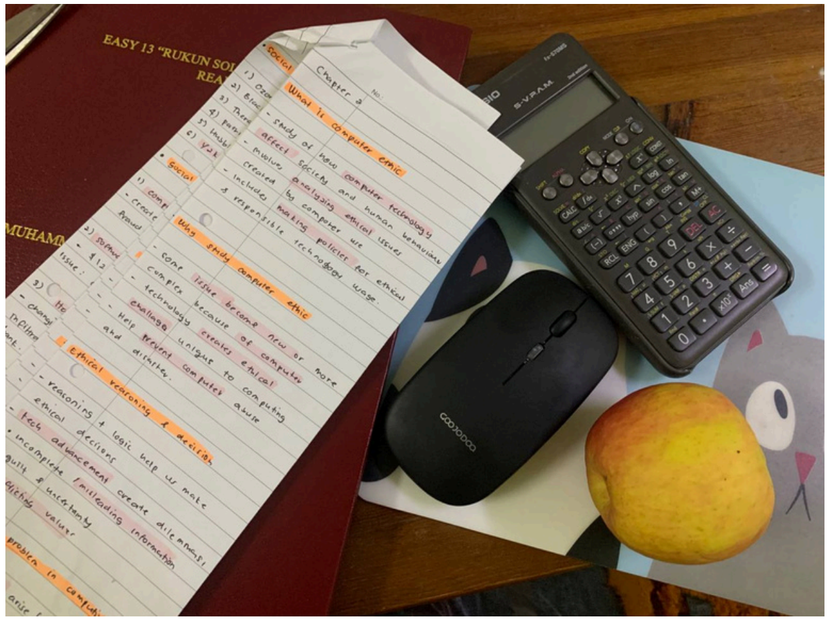
- In real-life, handwritten documents like exam scripts, forms, or notes often need to be digitized.

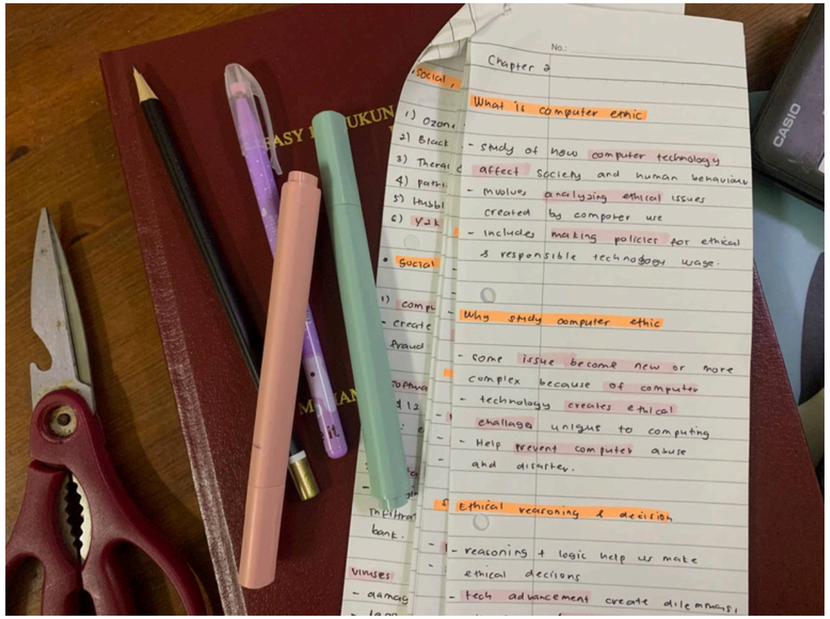
- These images usually have uneven lighting, background noise, or unclear handwriting, making automated analysis difficult.
  This project focuses on extracting handwritten text from such real-world images for reliable processing and recognition.

Project Context

- This project focuses on extracting handwritten text from real-world images.
- The objective is to separate handwriting from the background and enhance its quality for further analysis such as OCR and document digitization.

01





Load Handwritting Image

```
img = cv2.imread("handwriting.png")
```

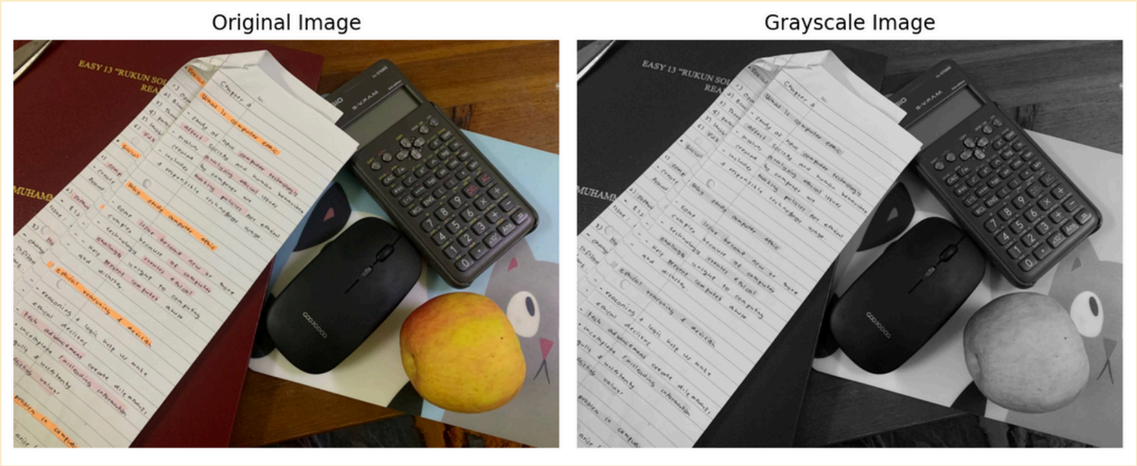
- Reads the handwritten image from the computer in OpenCV's default BGR format.
- There is no visual change at this stage as the image is only prepared for processing.

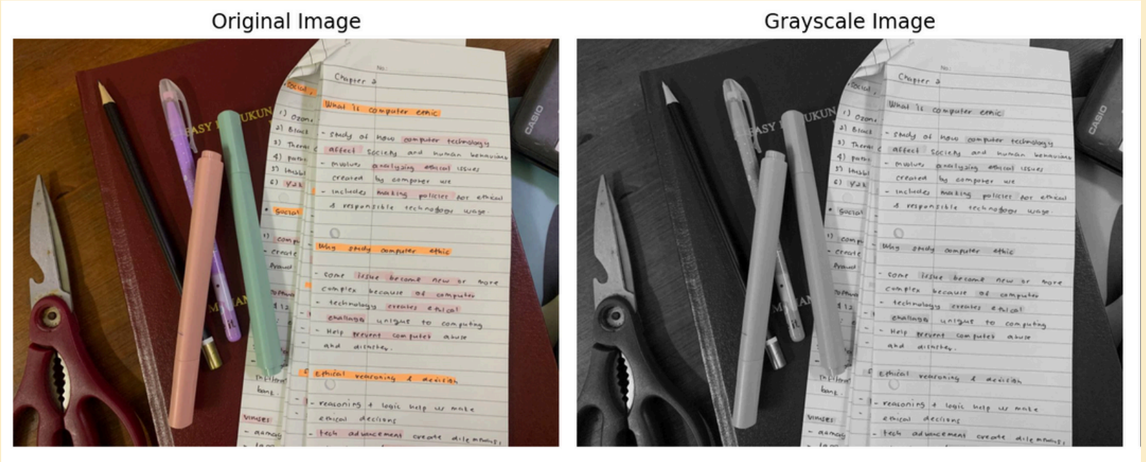
02

Convert to Grayscale

- Converts the image into a single intensity channel by removing color information.
- This simplifies processing and improves thresholding performance.
- Handwriting strokes become clearer against the background.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```





03

Static Thresholding

- Separates handwriting from the background using a fixed intensity value.
- Pixels brighter than the threshold become white, while darker pixels become black.
- Uneven lighting may cause loss of some handwriting details.

```
_, thresh_static = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
```

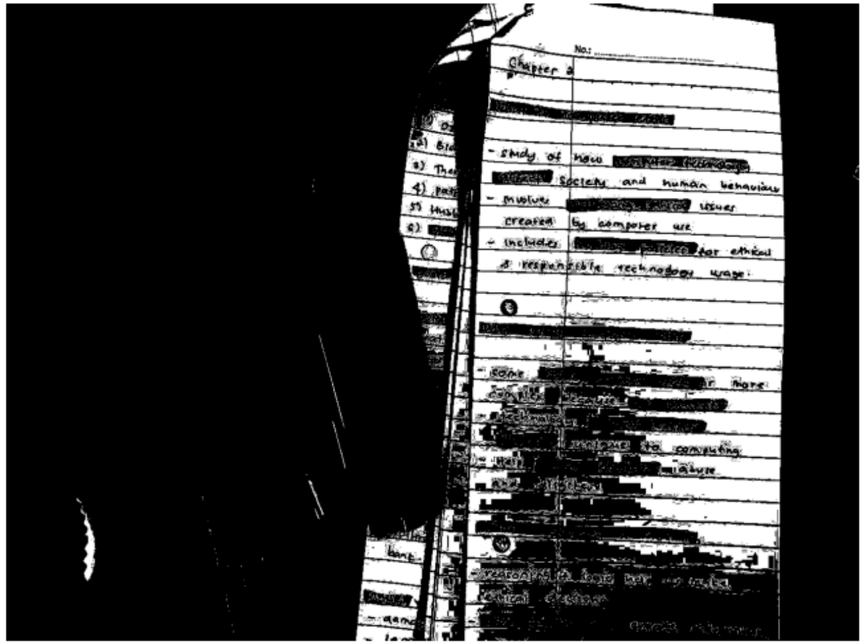
Uses a fixed threshold value (127). Pixels brighter than 127 → white Pixels darker than 127 → black

03

Static Thresholding

Static Threshold

Static Threshold

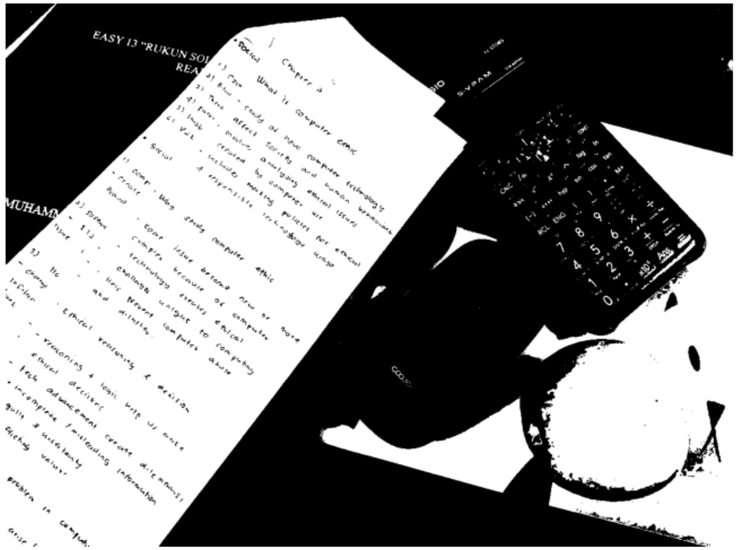


04 Otsu Thresholding

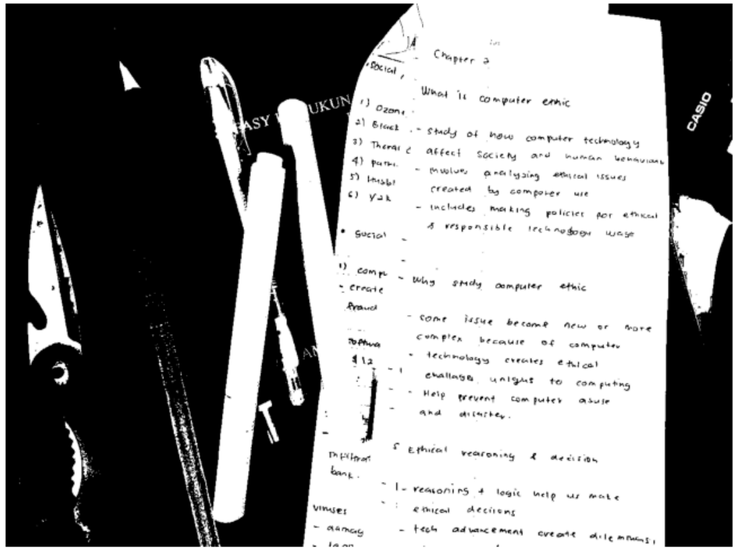
```
_, thresh_otsu = cv2.threshold(
gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU
)
```

- Automatically determines the optimal threshold value using histogram analysis.
- It adapts to lighting variations, producing cleaner separation and less noise compared to static thresholding.

Otsu Threshold



Otsu Threshold



05 Morphology Enhancement

Morphology modifies shapes in binary images to improve handwriting quality.

Step 5.1 – Define Kernel

Creates a structural element for image processing.

```
kernel = np.ones((3,3), np.uint8)
```

Step 5.2 – Opening (Noise Removal)

Removes small noise while keeping handwriting intact.

```
opening = cv2.morphologyEx(thresh_otsu, cv2.MORPH_OPEN, kernel)
```

Step 5.3 – Closing (Stroke Enhancement)

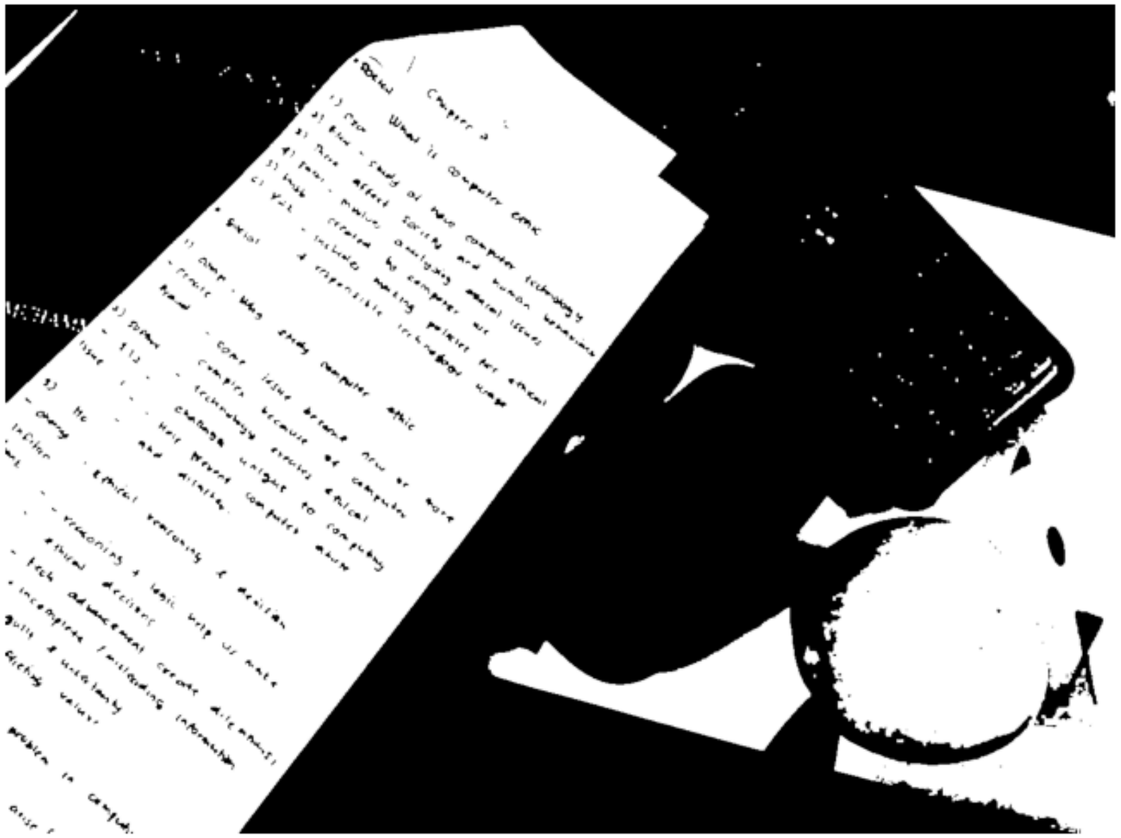
Fills gaps and reconnects broken handwriting strokes.

```
closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)
```

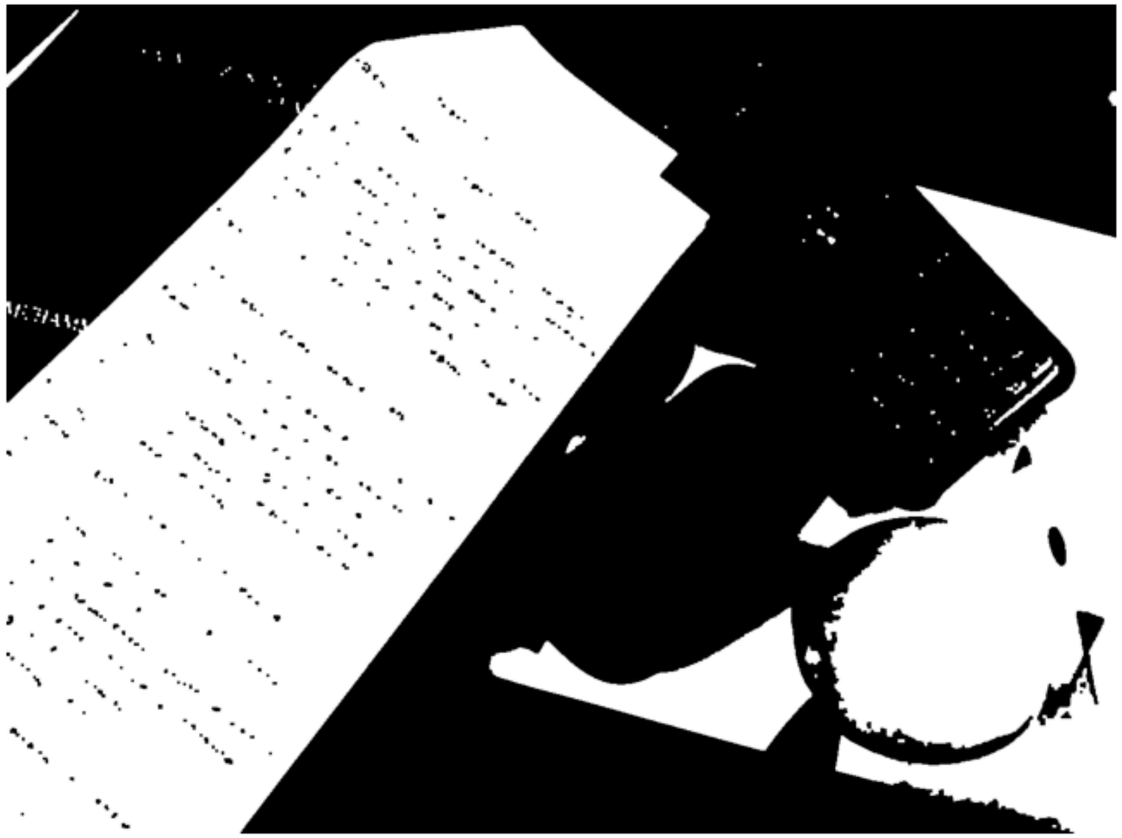
05 Morphology Enhancement

Morphology modifies shapes in binary images to improve handwriting quality.

After Opening



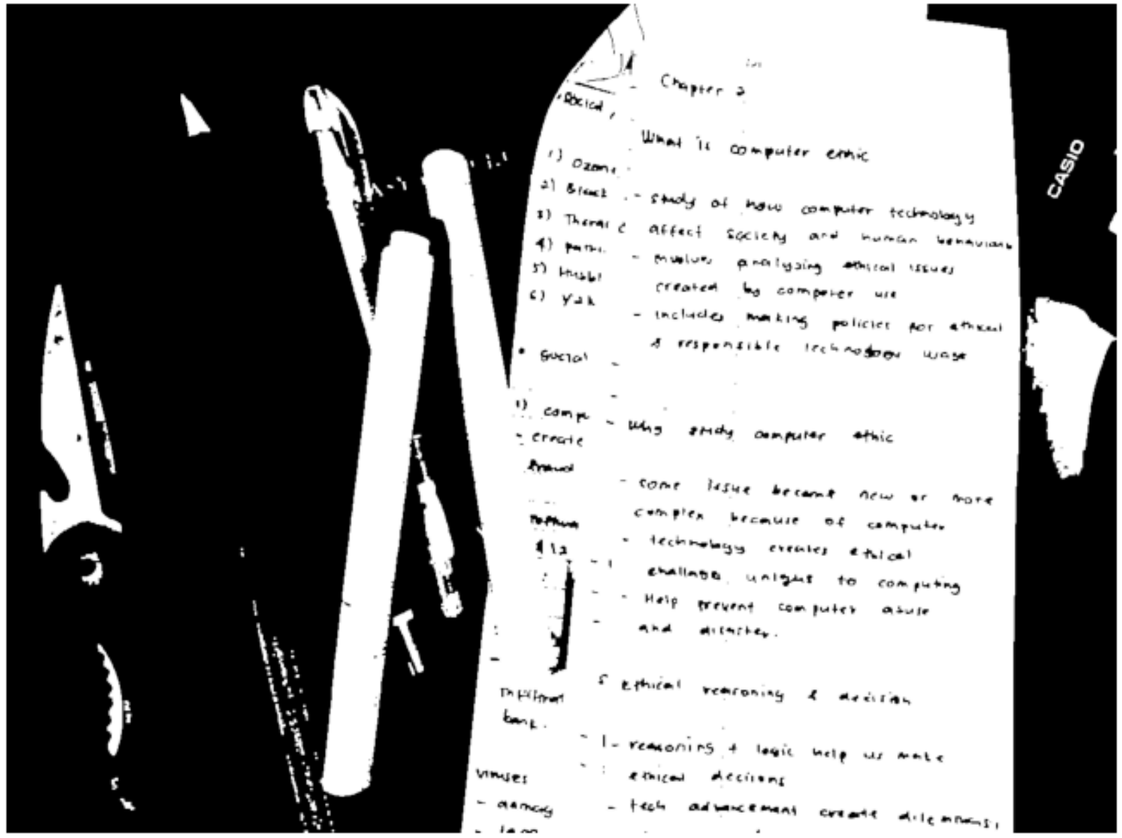
After Closing (Final)



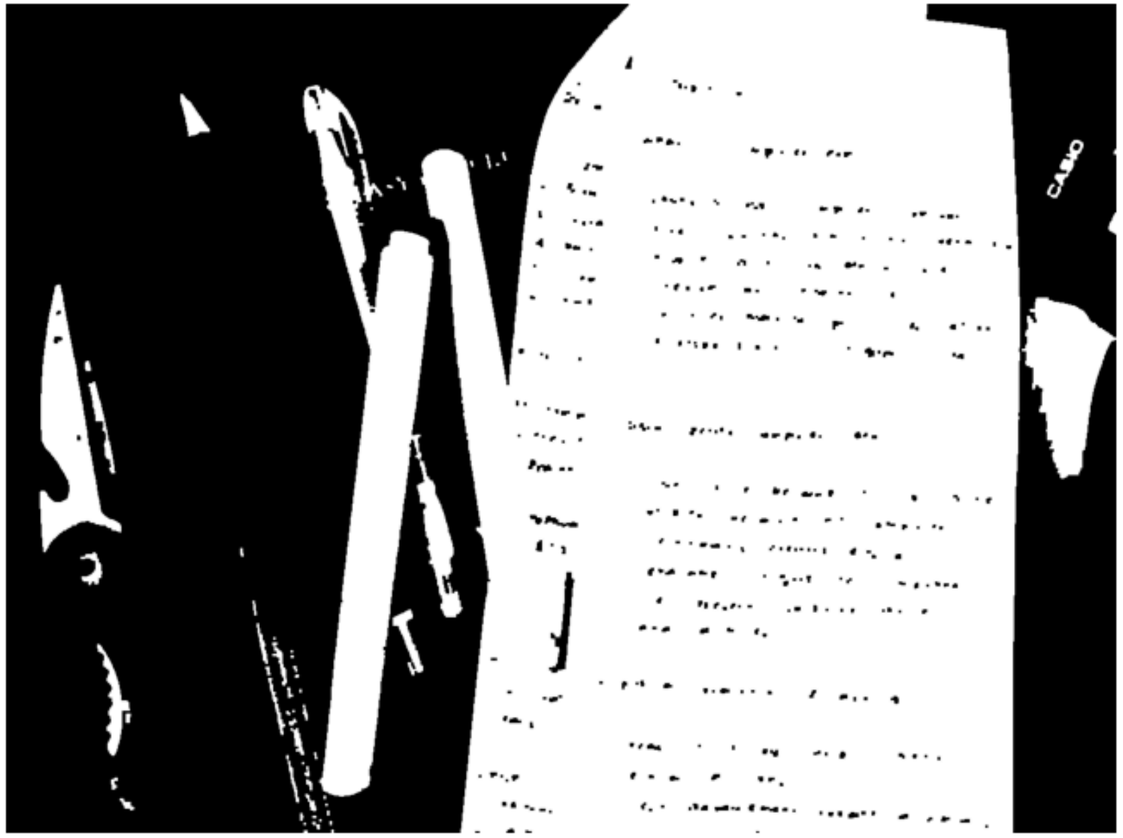
05 Morphology Enhancement

Morphology modifies shapes in binary images to improve handwriting quality.

After Opening



After Closing (Final)



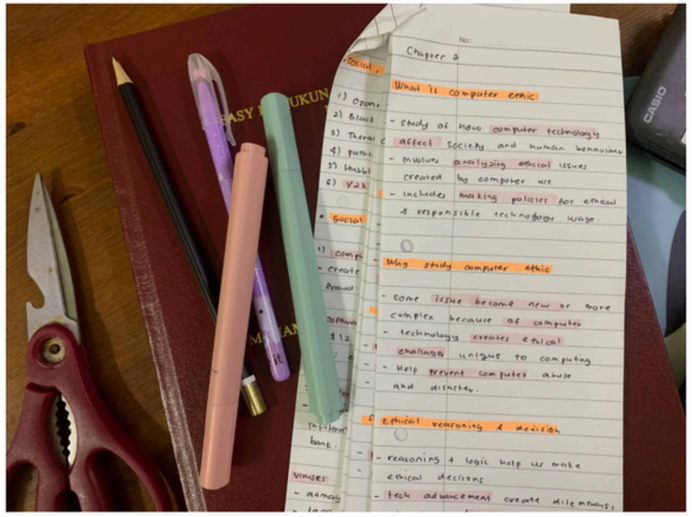
06 Display / Compare Results

Displays the processed image clearly for visualization and comparison.
Using cmap='gray' ensures correct grayscale display.

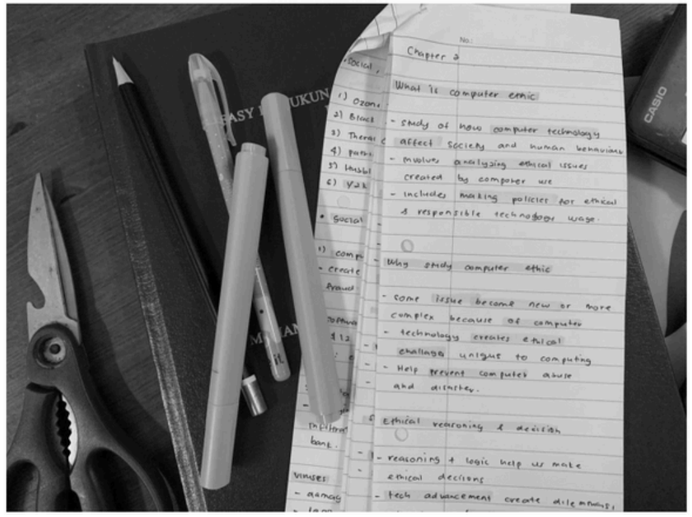
```
plt.imshow(result, cmap='gray')
plt.title("Processed Image")
plt.axis("off")
plt.show()
```

06

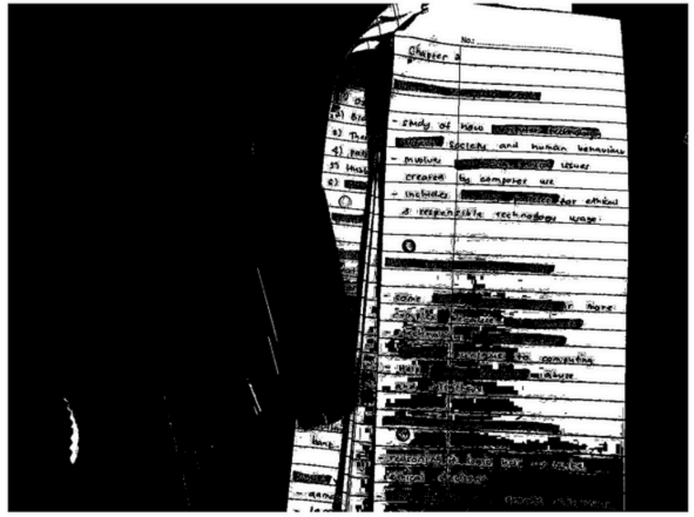
Original Image



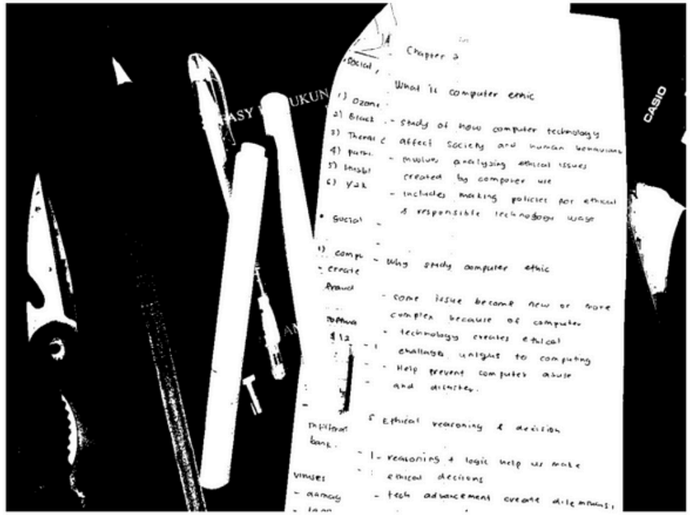
Grayscale Image



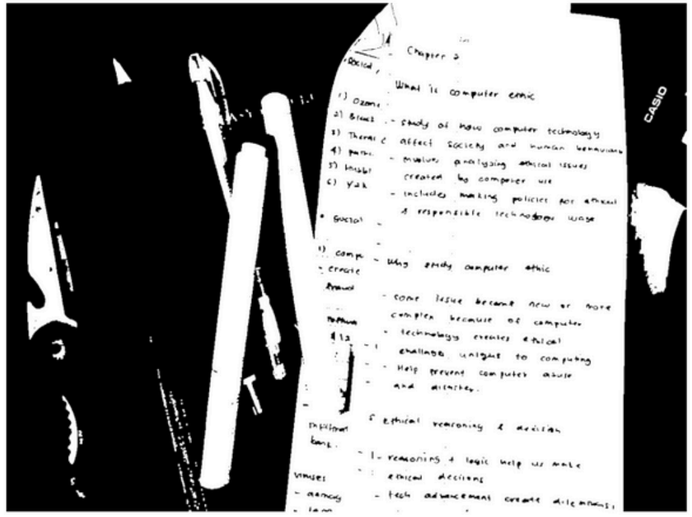
Static Threshold



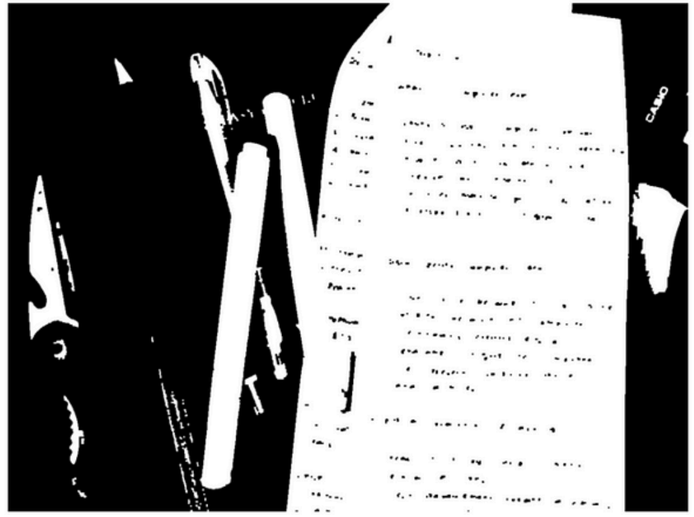
Otsu Threshold



After Opening

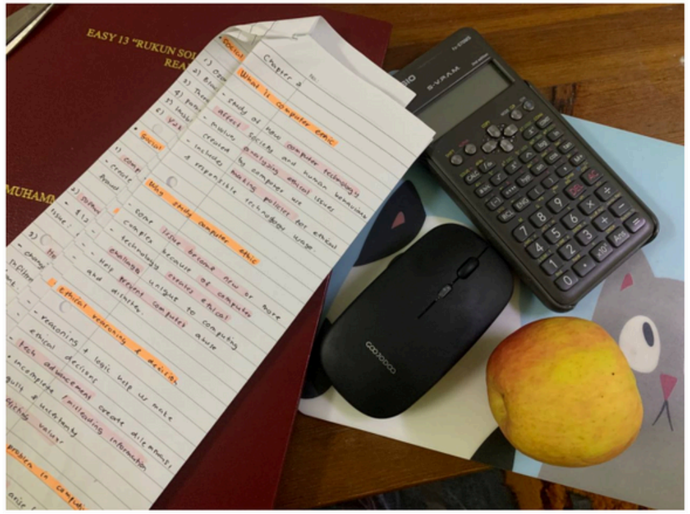


After Closing (Final)

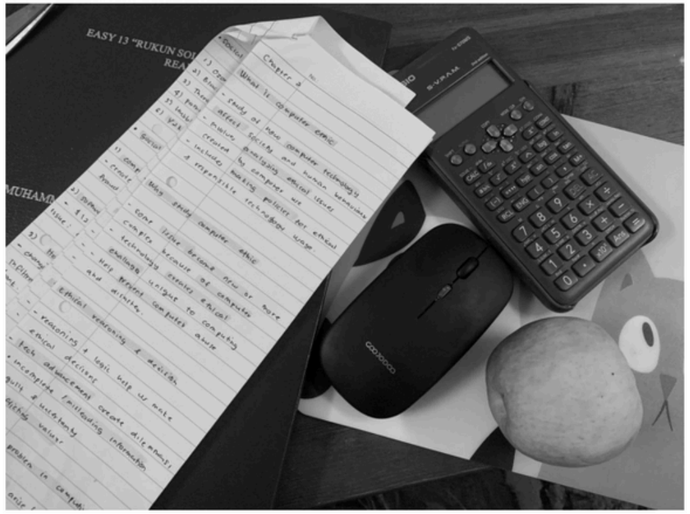


06

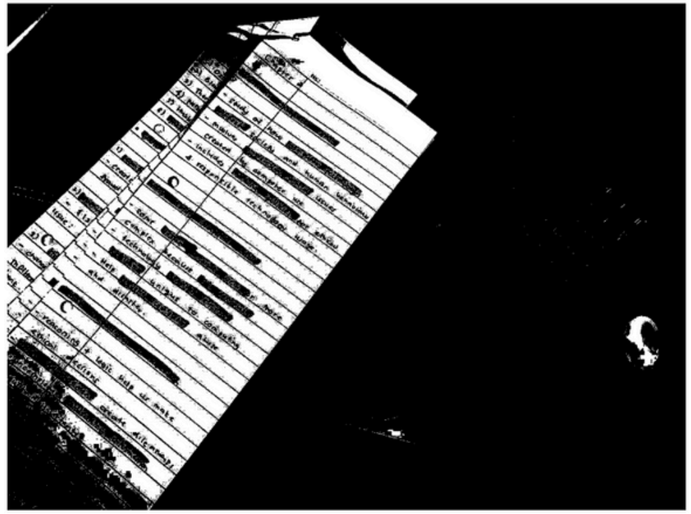
Original Image



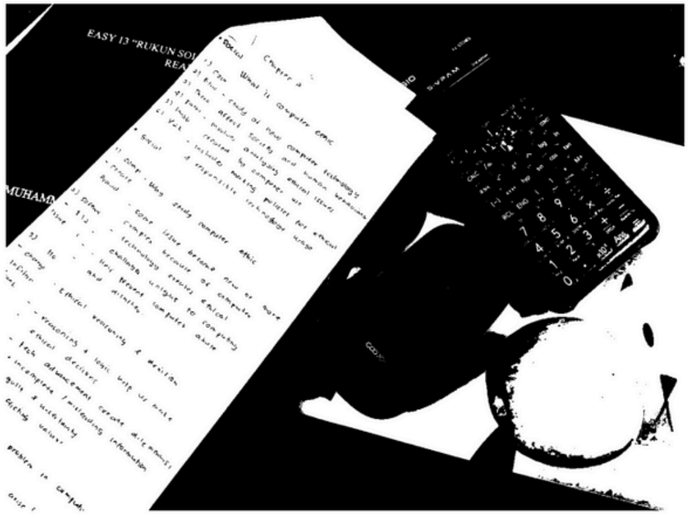
Grayscale Image



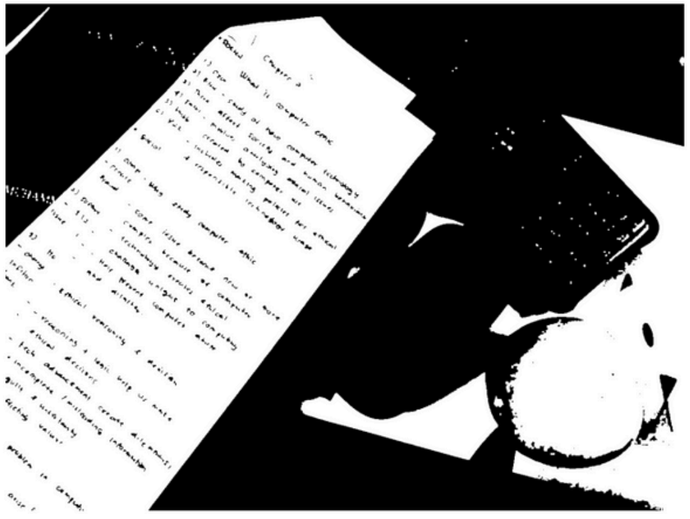
Static Threshold



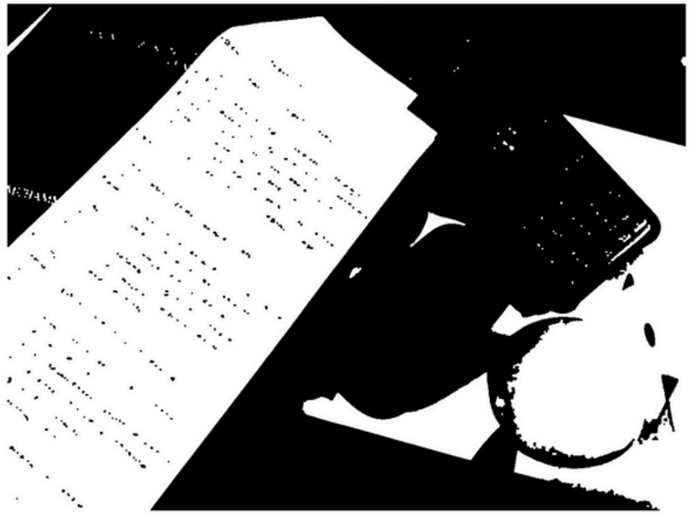
Otsu Threshold



After Opening



After Closing (Final)



Conclusion

The project successfully segments handwritten text from real-world images using image processing techniques.

Grayscale conversion simplifies the image, thresholding separates text from the background, and morphological operations clean noise and enhance stroke continuity.

Otsu thresholding adapts to lighting, making this method suitable for diverse handwriting styles and real-world conditions.

This approach is valuable for OCR, digitizing documents, and any application requiring reliable handwriting recognition.

Thank you!

github:
https://github.com/hazirahazman00-web/dipfinalassignment