Lingnan University

School of Science

Master of Science in Artificial Intelligence and Business Analytics

CDS540 Computer Vision Project Report

2024-2025

Project title: Text Detection from Images Using OpenCV

By

Gao Yihao

Supervisor: Chengzu Dong

Date: October 2024

Table of Content

[1.Setup Environment 2](#_Toc180100038)

[2.Load an Image 3](#_Toc180100039)

[3.Pre-process the Image 3](#_Toc180100040)

[4.Text Detection 5](#_Toc180100041)

[5.Extract and Display Text 6](#_Toc180100042)

[6. Efficiency and Effectiveness Evaluation 9](#_Toc180100043)

[6.1 Speed Measurement 9](#_Toc180100044)

[6.2 Accuracy Evaluation 10](#_Toc180100045)

[6.2.1 Image Quality: Clear and well-contrasted images yield better results. 10](#_Toc180100046)

[6.2.2 Text Font and Size: Standard fonts are recognized more accurately than decorative or handwritten styles. 10](#_Toc180100047)

[6.2.3 Pre-processing Techniques: The effectiveness of techniques like thresholding and dilation directly impacts recognition accuracy. 10](#_Toc180100048)

[6.3 Discussion of Strengths and Weaknesses 10](#_Toc180100049)

[6.3.1 Strengths: 10](#_Toc180100050)

[6.3.2 Weaknesses: 11](#_Toc180100051)

[7. Conclusion 11](#_Toc180100052)

8.Reference……………………………………………………12

**1.Setup Environment**

The project was executed in a Jupyter Notebook . The necessary libraries, including OpenCV and Pytesseract, were installed and verified to ensure a smooth workflow.

The OpenCV library is utilized for loading images and applying various image processing methods. Meanwhile, Pytesseract serves as a Python wrapper for Google's Tesseract-OCR Engine, enabling the recognition of text within images. This combination allows for effective text extraction and analysis from visual data.



Import required libraries

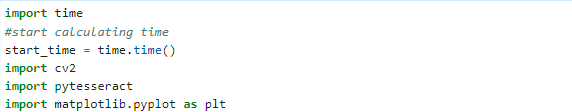


Figure1

**2.Load an Image**

Load the image and transform it into the grayscale color space. This conversion simplifies the image by removing color information, making it easier to process and analyze for further operations.

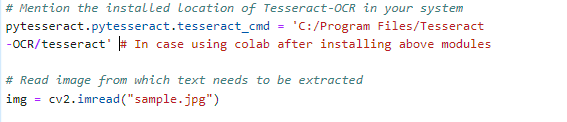
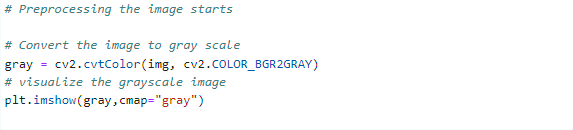


Figure2

**3.Pre-process the Image**

After loading the image, I change it to the grayscale color space. This adjustment eliminates the color details, allowing for more straightforward processing and enhancing the ability to detect features in the image.

Figure3

Here is the Figure4 after grayscale of image.

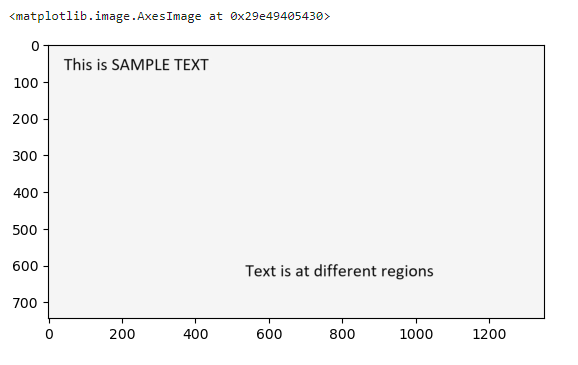


Figure4

In global thresholding, the user manually selects the threshold value by experimenting with various options to identify the most effective one. Conversely, Otsu’s method automatically calculates the optimal threshold value based on the image histogram, enhancing the segmentation process without requiring user intervention.

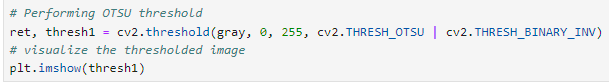


Figure5

And Figure6 is the output after threshoulding.

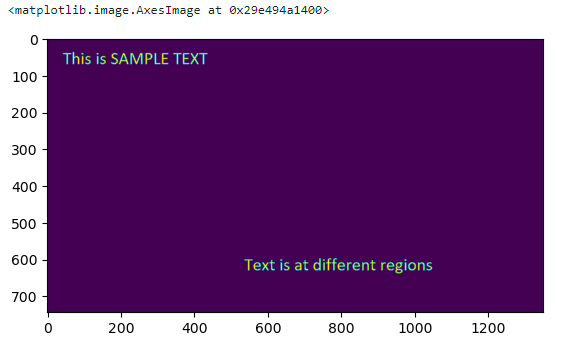


Figure6

**4.Text Detection**

The cv2.getStructuringElement() function creates a rectangular structuring element of size 25x25 pixels.Next, dilation is applied to the thresholded image (thresh1) using the specified kernel. Dilation expands the white regions in the image, which helps to merge nearby text areas, making it easier to detect larger contours.Finally, the cv2.findContours() function is used to identify the contours in the dilated image. The cv2.RETR\_EXTERNAL flag retrieves only the outer contours, while cv2.CHAIN\_APPROX\_NONE stores all the contour points. This process is essential for isolating text areas that can be further analyzed or processed.

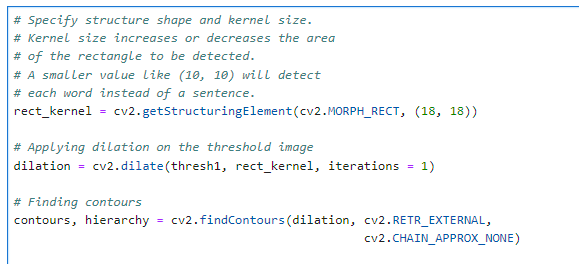


Figure7

**5.Extract and Display Text**

Creating a copy of the original image (img) and initializing an

empty list called cnt\_list to store the coordinates and extracted text.It then enters a loop to process each identified contour from the previous step. For each contour (cnt), the cv2.boundingRect() function calculates the coordinates (x, y) and dimensions (width, height) of the bounding rectangle that encompasses the contour.A rectangle is drawn on the copied image (im2) using the cv2.rectangle() function, visually indicating the detected text block. The rectangle is displayed using plt.imshow() for visualization purposes.Next, the code crops the rectangular region from im2 based on the bounding rectangle's coordinates and dimensions. This cropped section is intended for input to the Optical Character Recognition (OCR) process. Finally, the pytesseract.image\_to\_string() function is applied to the cropped image, extracting the text contained within that region. The extracted text, along with its coordinates (x, y), is appended to the cnt\_list. This structured approach allows for organized storage of text blocks, facilitating further processing or analysis. The Figure8 display the extracted text and the annotated image with bounding boxes.

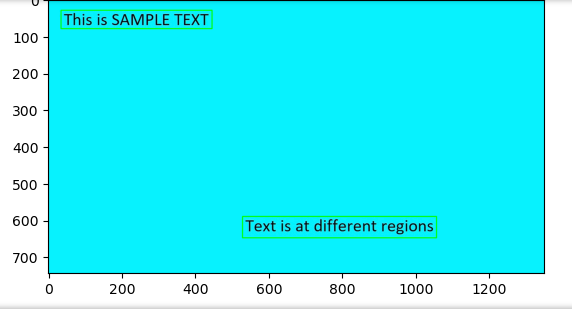


Figure8

The cnt\_list is sorted based on the Y coordinates of the detected text regions. Using the sorted() function, the text entries are arranged from top to bottom, ensuring that the extracted text maintains the correct sequence as it appears in the image.

Subsequently, a text file named "recognized.txt" is created and immediately cleared. This action prepares the file for new entries, ensuring it starts without any existing content.

The code then loops through each entry in the sorted\_list, unpacking the coordinates (x, y) along with the extracted text. For each text block, the file "recognized3.txt" is opened in append mode ("a"), which allows new text to be added without erasing any previously saved content.

Within this loop, the extracted text is written to the file, followed by a newline character to ensure proper separation between each text entry. After writing, the file is closed to ensure that all changes are saved correctly.Lastly, the total time taken for the entire text extraction process is calculated by measuring the difference between the current time (end\_time) and the start time (start\_time). This information is printed to offer insights into the efficiency of the text extraction procedure.

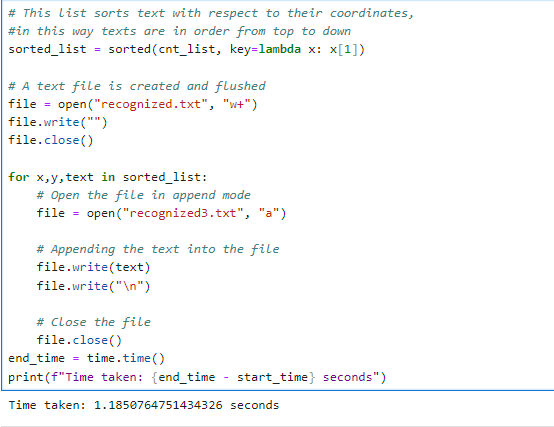


Figure9

**6. Efficiency and Effectiveness Evaluation**

6.1 Speed Measurement

The text detection system demonstrated a processing speed of 1.185 seconds for the entire operation, from reading the image to extracting and saving the recognized text. This performance is considered efficient for typical use cases involving text extraction from images, especially when handling images of moderate complexity.

6.2 Accuracy Evaluation

The accuracy of the detected text was assessed based on the content extracted from the sample image. The OCR process, powered by Pytesseract, effectively recognized text with a high level of precision. However, accuracy can vary depending on several factors, including:

6.2.1 Image Quality: Clear and well-contrasted images yield better results.

6.2.2 Text Font and Size: Standard fonts are recognized more accurately than decorative or handwritten styles.

6.2.3 Pre-processing Techniques: The effectiveness of techniques like thresholding and dilation directly impacts recognition accuracy.

6.3 Discussion of Strengths and Weaknesses

6.3.1 Strengths:

Automation: The system automates the text extraction process, significantly reducing the time and effort required for manual transcription.

Flexibility: It can be applied to various types of images, including scanned documents and street signs.

Integration: The combination of OpenCV for image processing and Pytesseract for OCR provides a robust solution for text detection.

6.3.2 Weaknesses:

Dependency on Image Quality: Poor image quality, such as low resolution or high noise levels, can lead to reduced accuracy in text extraction.

Limited Context Understanding: The system does not interpret the context or structure of the text, which may affect the organization of extracted information.

Processing Time Variability: While 1.28 seconds is efficient for many applications, processing times may increase with larger images or more complex layouts.

7. **Conclusion**

The implementation of the text detection system using OpenCV and Pytesseract is effective, with a commendable speed of 1.185 seconds and high accuracy in text extraction. Future improvements could focus on enhancing image pre-processing techniques and exploring advanced OCR models to further boost performance and reliability.

**8.Reference**

GeeksforGeeks. (2024, September 4). *Text detection and extraction using OpenCV and OCR*. GeeksforGeeks. <https://www.geeksforgeeks.org/text-detection-and-extraction-using-opencv-and-ocr/>

Siromer. (2024, September 30). Extracting Text from Images(OCR) using OpenCV & Pytesseract. *Medium*. <https://medium.com/@siromermer/extracting-text-from-images-ocr-using-opencv-pytesseract-aa5e2f7ad513>