How to Use Pytesseract and OpenCV for Image Processing and Text Recognition

Student Name:GUAN Yuqi
Student ID:3160069

Abstract

This report examines the application of OCR technology, specifically through the integration of Pytesseract and OpenCV, to automate data entry, document digitization, and text analysis. It outlines key image preprocessing techniques, including grayscale conversion, binarization, and edge detection, which are crucial for improving text recognition accuracy. Following these preprocessing steps, Pytesseract is employed to extract text from images. The system's performance is evaluated based on accuracy and processing time, revealing high recognition accuracy alongside efficient processing speeds. However, challenges such as image quality and background clutter are noted, highlighting the limitations of the current methodology. In the end, the report concludes by discussing the issues encountered and suggesting potential solutions, emphasizing the need for further research to enhance OCR technology and its applications.

1. Introduction

In the wave of digital transformation, many traditional industries are striving to convert paper documents and information into editable and searchable digital formats. This demand has driven the development of Optical Character Recognition (OCR) technology, which serves as a bridge between physical text and digital information, automating processes such as data entry, document digitization, and text analysis. Pytesseract and OpenCV are two of the most widely used tools in this field.

The main objectives of this report are to explain the text detection methods I have chosen and how they work, provide an overview of how Pytesseract and OpenCV perform image processing and text recognition, and ultimately evaluate the performance and accuracy of these combined methods.

2. Background

First, I will briefly introduce some technical terms relevant to this report.

2.1 Optical Character Recognition (OCR)

Optical character recognition is a science that enables to translate various types of documents or images into analyzable, editable and searchable data(J. M et al.,2020).

The process typically involves several stages, including image preprocessing, character segmentation, feature extraction, and character recognition.

2.2 Pytesseract

Pytesseract is a Python-based OCR tool that can be easily integrated into various applications, providing functionalities such as image-to-text conversion and bounding box extraction for detected text.

2.3 OpenCV (Open Source Computer Vision Library)

OpenCV is a widely used open-source computer vision library that offers a rich set of image processing functionalities, including filtering, edge detection, and morphological transformations. It is often used to prepare images for OCR by enhancing their quality and structuring them for better recognition accuracy (Bradski & Kaehler, 2008).

3. Methodology

3.1 Image Preprocessing Techniques

Image preprocessing is a critical step in improving image quality to achieve better OCR results. After reading the image, we need to preprocess it to enhance text recognition accuracy. The preprocessing techniques involved in this report are as follows:

3.1.1 Grayscale Conversion

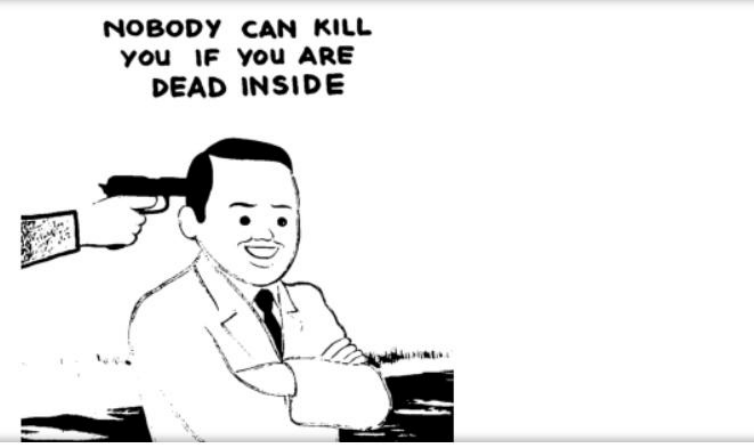
Since processing color images can be complex, we usually need to convert images to grayscale first. This transformation clarifies the structure and details within the image, facilitating subsequent processing. Grayscale conversion simplifies the color information of each pixel to a value ranging from 0 (black) to 255 (white), with intermediate values representing different shades of gray.

3.1.2 Binarization

Binarization is the process of converting a grayscale image into an image with only two colors: black and white. By setting a threshold, pixels above this threshold are set to white (255), while those below are set to black (0). For instance,I set the threshold for the image at 100, meaning that any pixel value greater than 100 would be white, while the rest would be black. The effect is shown in the figure below.

```
# binarization
_, binary_image = cv2.threshold(gray_image, 100, 255, cv2.THRESH_BINARY)
```

```
plt.imshow(binary_image, cmap='gray')
plt.axis('off')
plt.show()
```



3.1.3 Edge Detection

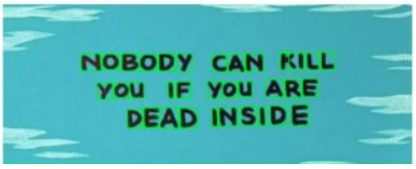
Edge detection identifies the edges of objects within an image. Edges typically represent the most significant color changes in an image and help us understand the shape and position of objects. Using OpenCV, we can perform edge detection through the Canny algorithm. By focusing solely on edge information, we can significantly reduce the data volume for subsequent processing.

3.2 Text Detection Method

After preprocessing, the next step is to use “image_to_boxes( )” of Pytesseract to detect text. This function returns a string containing the location information for each character, which we need to parse and use to draw bounding boxes on the image. The effect is as shown below.

```
# getting boxes around text
img = cv2.imread('picture-1.jpg')
h, w, c = img.shape
boxes = pytesseract.image_to_boxes(img)
for b in boxes.splitlines():
    b = b.split(' ')
    img = cv2.rectangle(img, (int(b[1]), h - int(b[2])), (int(b[3]), h - int(b[4])), (0, 255, 0), 2)
```

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```



3.3 Tesseract Configuration

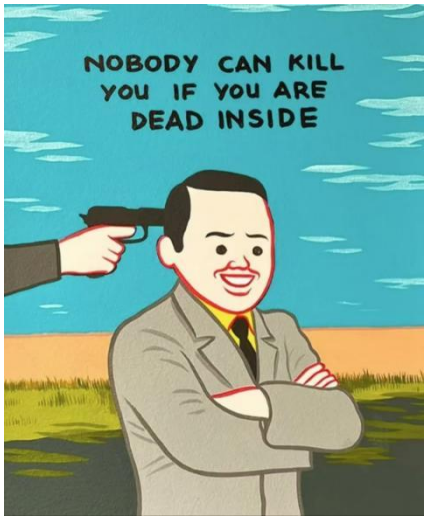
When using Pytesseract for text recognition, we can adjust some settings of the Tesseract OCR through the “custom_config” parameter. This configuration typically

includes two important parameters:--oem and --psm. Specifying the language and OCR engine mode can significantly impact performance. In this report, the images used contain only simple English vocabulary, and the code used was simply, “recognized_text = pytesseract.image_to_string(img)”, without involving adjustments to the aforementioned parameters.

4. Results

4.1 Performance Evaluation

This report evaluates the effectiveness of the implemented system based on accuracy and processing time. The recognition results indicate a high accuracy level, successfully distinguishing between uppercase and lowercase letters without misrecognizing or adding extra letters.




```
# recognize text and output the results
recognized_text = pytesseract.image_to_string(img)
print(recognized_text)
```

```
NOBODY CAN KILL
you IF you ARE
DEAD INSIDE
```

To illustrate the processing speed, we can measure the text detection system's speed using Python's time module. For example, in the threshold processing step (illustrated below), the time taken was approximately 0.284 seconds.

```
import time

start_time = time.time()
recognized_text = pytesseract.image_to_string(thresh)
end_time = time.time()

print(f"Processing Time: {end_time - start_time} seconds")
```

```
Processing Time: 0.2838404178619385 seconds
```

4.2 Results Analysis

Comparing the results before and after image processing, the effectiveness of image processing varies significantly depending on the image. For example, higher-resolution images typically yield better results, while standard fonts with high contrast against the background are easier to detect. Misrecognition often occurs due to cluttered backgrounds.

4.3 Strengths and Weaknesses

4.3.1 Strengths

Since Pytesseract provides a simple API for text recognition, and most of the libraries involved can be called for free, the above operation process can be widely used by the public for various image text recognition needs.

4.3.2 Weaknesses

Although the image processing speed was fast for the images I used, this was due to the clarity of the text and the cleanliness of the background, with no complex languages involved. Processing time will inevitably increase with larger images or more complex preprocessing steps. Furthermore, OCR accuracy may decline due to poor image quality or cluttered backgrounds. Various factors contribute to the challenges faced in this workflow, indicating significant room for improvement.

In the future, I aim to explore this field further and study the latest advancements to enhance text detection and recognition effectiveness.

5. Issues Encountered and Solutions

Initially, I used an image captured while watching a YouTube lecture (as shown in the first figure), without considering that the image had very few colors. When I began preprocessing the image, I realized that no matter what operations were applied, the differences were minimal (ss shown in the next two pictures). Thus, I promptly replaced the image to ensure the subsequent processing would yield more noticeable comparisons.

```
#image preprocessing
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  #convert the image to grayscale
image_rgb = cv2.cvtColor(gray_image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.axis('off')
plt.show()
```

```
#binarization
_, binary_image = cv2.threshold(gray_image, 150, 255, cv2.THRESH_BINARY)
```

```
#display
plt.imshow(binary_image, cmap='gray')
plt.axis('off')
plt.show()
#the picture I used is a screenshot when browsing youtube
```

During the grayscale conversion process, I was surprised to find that the processed image was unexpectedly bright. "Shouldn't it be gray?" I questioned. After Googling, I discovered that my confusion stemmed from not recognizing the difference between RGB and BGR formats, which led to suboptimal results. I then added this line： ”image_rgb = cv2.cvtColor(gray_image, cv2.COLOR_BGR2RGB)”, and the resulting image matched my expectations (see the comparison images below).

References

1. J. Memon, M. Sami, R. A. Khan and M. Uddin, "Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)," in IEEE Access, vol. 8, pp. 142642-142668, 2020, doi: 10.1109/ACCESS.2020.3012542.

2. Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media.

3. Trenton McKinney. (2019), “Enhancing OCR Accuracy in Python with OpenCV and PyTesseract”. https://trenton3983.github.io/posts/ocr-image-processing-pytesseract-cv2/

Tips：The YouTube link to the screen recording is here https://youtu.be/-US-bk46HxA