Project Report

Handwritten Text Detection and Recognition

Course Name : Digital Image Processing
Course Code : CS7.404
Semester : Monsoon'24
Group Name: PixelPals
Instructor Name : Anoop Namboodiri

Janaksinh Ven
Roll: 2023702018
Evani Lalitha
Roll : 2024701023

November 30, 2024

Contents

1 Introduction

Handwritten Text Recognition (HTR) is a sub-domain of Optical Character Recognition (OCR) that focuses on automatically converting handwritten text into digital text. HTR is a challenging problem that intersects with image processing, requiring the handling of various complexities inherent in handwritten documents. Some of the key challenges include the uniqueness of handwriting, noise, blurring, smudges, incomplete characters, variations in ink density, the orientation of characters, and the scaling of characters. In this project, we aim to leverage traditional image processing techniques for line segmentation and word segmentation, which are critical steps in the overall process of recognizing handwritten text.

The relavent files can be found in this github link: ***PixelPals github***

1.1 Report Structure

The subsequent sections of this report are organized as follows:

- **Objectives**: Presents the main goals and the problems this project aims to address.
- **Dataset**: Describes the dataset used, including its source, characteristics, and relevance to the project.
- **Methodology**: Outlines the core techniques and algorithms employed to solve the problem.
- **Implementation**: Provides a detailed explanation of the steps taken to implement the proposed methods.
- **Results**: Discusses both the qualitative and quantitative outcomes obtained during the project.
- **Conclusion**: Summarizes the project, highlights the achieved solutions, and suggests directions for future work.

1.2 Objectives

The primary objective of this project is to utilize the image processing techniques to perform segmentation of a handwritten document image and subsequently move on to recognition. Specifically, we aim to:

- Use basic image processing techniques to pre-process the handwritten documents at hand.
- Use traditional image processing algorithms like piece-wise painting algorithm to perform line segmentation and word segmentation.
- Perform word recognition of the resultant images and state the resultant metrics.

2 Dataset

The IAM dataset [3] is a benchmark dataset for handwritten text recognition and writer identification. It contains 1,539 scanned pages of handwritten English text contributed by 657 writers. Each page is annotated at the line, word, and character levels, making it suitable for segmentation and recognition tasks. The dataset includes ASCII transcriptions, XML metadata, and a lexicon, supporting research in optical character recognition (OCR) and handwriting analysis.

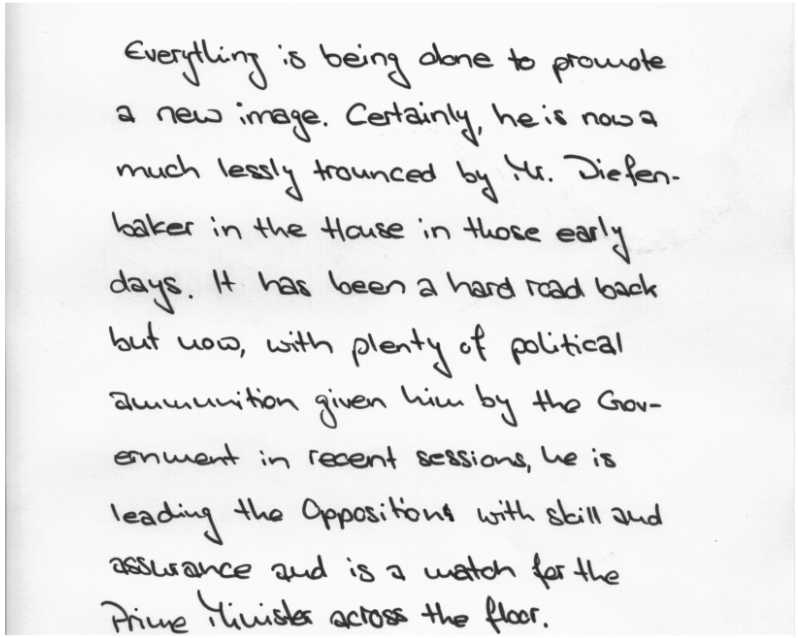
The IAM dataset in this project is taken from the Kaggle IAM handwritten dataset. This dataset contains both printed and handwritten text. The printed text comprises of what the author was supposed to write in the blank space provided. For our project we downloaded around 30 scanned pages of handwritten page level text where each page is written by a unique writer to maintain the variablity and authenticity and avoid redundancy. We also went for various handwritten styles like fully, semi, and non cursive. We also selected the pages which contain varying font sizes while writing and handwritten text that is written at various degrees of slant, varying line space and word spaces which makes it challenging for segmentation. The pages are then cropped manually to remove the additional printed text from the image since the focus of this project is handwritten text only. They are then saved and used in the project. Some of the example images can be found in fig 1.

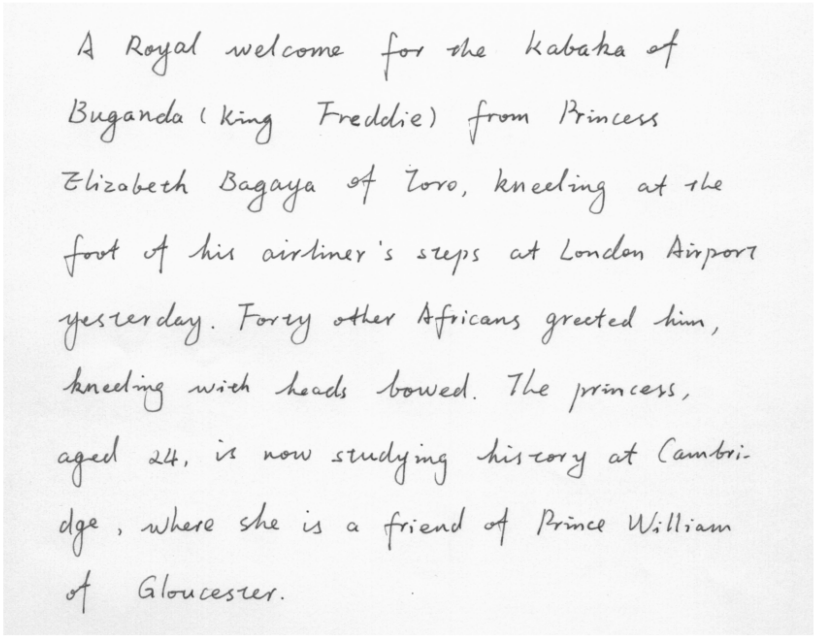
3 Methodology

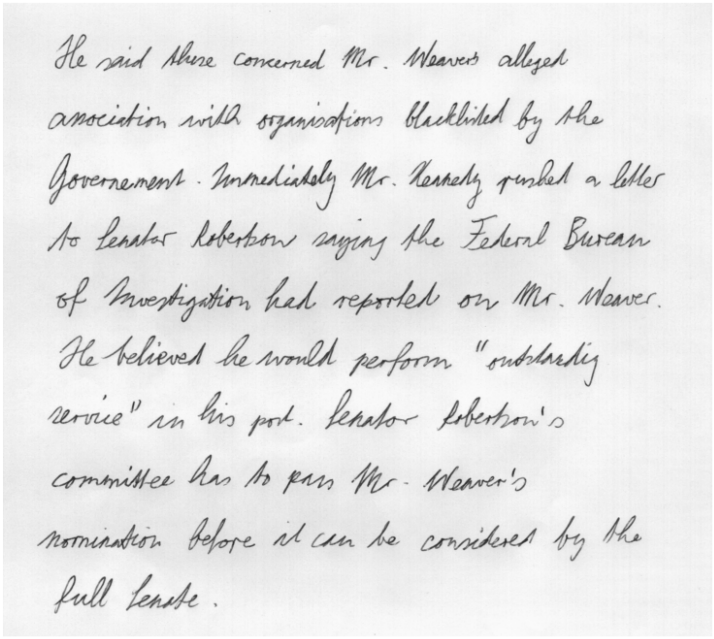
3.1 Line segmentation

The line segmentation algorithm is implemented from [1]. The algorithm introduced in the paper [1] is briefly described below. The goal of the line segmentation algorithm is to identify and separate individual lines of text in an image containing handwritten text. The process involves several stages of image processing, including connected component analysis, thresholding, skeletonization, filtering, and noise removal. Here's how the algorithm works:

1. **Connected Components Analysis**: The first step is to perform a connected components analysis using cv2.connectedComponentsWithStats. This function detects all connected regions in the image, where each component corresponds to a potential text region. The algorithm then calculates the width of each component and computes the average component width across all components. This average width is used to help in the subsequent steps of the algorithm, as it provides an estimate of the width of text characters in the image.
2. **Row-wise Component Width Averaging**: For each row in the image, the average width of the components (i.e., text characters) is calculated. This row-wise averaging helps in determining the most likely spacing between individual characters and lines of text. It also allows the algorithm to adjust to varying text sizes across different rows of the image.







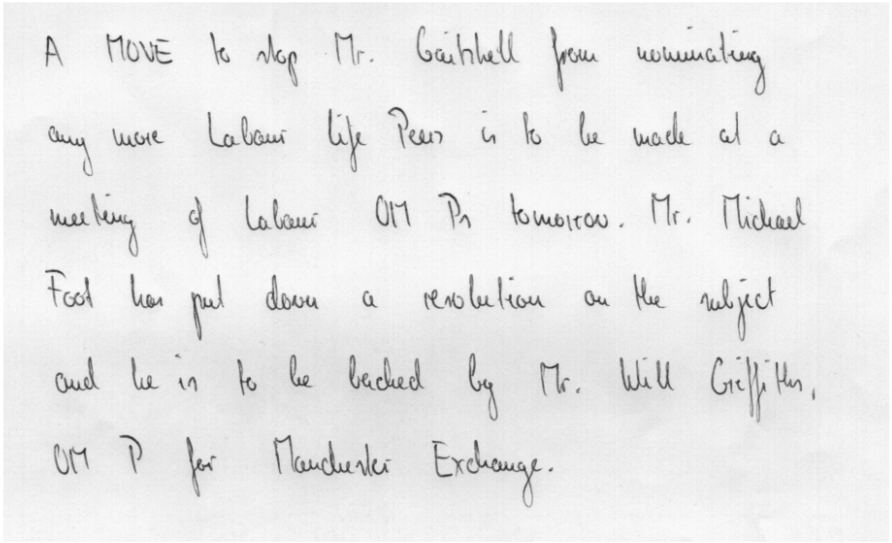


Figure 1: Some sample images of dataset

3. **Thresholding for Text Area Highlighting**: Next, a thresholding technique is applied to the image to highlight the regions containing text. This involves separating the foreground (text) from the background by choosing an appropriate threshold value, which helps in identifying the text areas clearly.
4. **Filling Vertical Gaps**: Once the text regions are highlighted, the algorithm identifies the vertical gaps between these highlighted areas. Based on a threshold value, these gaps are filled to ensure that individual lines of text are properly connected, making it easier to separate them in subsequent steps.
5. **Filling Gaps Between Black Regions**: To further refine the segmentation, the gaps between black regions (representing text) are filled based on a threshold value that is four times the calculated component width. This helps to solidify the lines and separate them more distinctly.
6. **Skeletonization for Line Detection**: After thresholding and filling the gaps, the image is inverted, and skeletonization is applied. Skeletonization reduces the lines to their central axis, helping to detect the exact shape and structure of each line. The output of this step is a clear representation of the individual lines of text.
7. **Noise Reduction with Horizontal Filtering**: To remove noise and small unwanted components that may still be present in the image, a horizontal filter is applied. The filter used is [[1, 2, 1], [0, 0, 0], [-1, -2, -1]], which emphasizes horizontal features while reducing vertical noise. This filter helps disconnect small, noisy components from the lines, ensuring that only the true lines of text remain.
8. **Length Thresholding for Noise Removal**: After applying the horizontal filter, connected components are analyzed based on their size. Any components that do not meet a minimum length threshold are discarded. This step ensures that only significant components, corresponding to actual lines of text, are retained.
9. **Connecting Broken Line Components**: In some cases, the lines may be broken due to gaps in the text or noise. To address this, the algorithm applies additional thresholding to connect broken components and extend them towards their endpoints. This ensures that the individual lines are fully formed and complete.
10. **Extracting and Saving Individual Lines**: Finally, after all the previous steps, the individual lines of text are successfully segmented. These lines are then extracted and saved as separate images in a designated folder.

The resulting images from each stage of the algorithm can be found in the results section.

3.2 Word segmentation

3.2.1 Initial Approach (Piecewise Painting Algorithm - PPA):

The PPA was first implemented for word segmentation using the average connected component width. This method worked well for line segmentation as it captured the average character width within a line. However, it was ineffective for word segmentation, as connected components in words represent individual characters, not the actual word boundaries.

3.2.2 Limitations of PPA:

In word segmentation, the average width of connected components did not accurately reflect the spaces between words, especially with varying character sizes or styles.

3.2.3 New Threshold-Based Approach:

To overcome the limitations of the PPA, we introduced a new threshold-based method for word segmentation. The process involved:

1. **Identifying Columns Between Words:** The first step was to identify the columns of pixels between words. This allowed us to separate blocks of continuous text from gaps.
2. **Determining Continuity of Columns:** We then assessed the continuity of these columns to identify whether they belonged to the same word or represented the gap between different words.
3. **Thresholds Based on Writing Style:** We recognized that the gap between characters is influenced by the writing style, which led to the use of adaptive thresholds for different types of handwriting:
    - **Cursive Writing:** In cursive writing, characters are often connected, resulting in fewer and narrower gaps between words. Based on this observation, we set the threshold for word segmentation in cursive writing at 60% of the average gap width within a line.
    - **Non-Cursive Writing:** For non-cursive writing, where the gaps between characters are more pronounced, we set the threshold to 100% of the average gap width. This accounted for the more distinct word boundaries in non-cursive writing.

The threshold-based approach successfully handles both cursive and non-cursive writing styles by adapting the segmentation threshold to the characteristics of each style. This method improves the accuracy of word segmentation compared to the previous Piecewise Painting Algorithm, making it more reliable for different handwriting styles.

3.3 Word recognition

Word recognition is the final step in our project pipeline. After segmenting the paragraph into individual lines and those lines into words, these words are then used as the input to the word recognizer to predict the words in the image. In order to achieve this there are several deep learning techniques that evolved with time which include CNNs, RNNs, transformers, etc. Considering the latest advancements in handwritten word recognition we choose a transformer based model called TrOCR [2].

TrOCR (Transformer-based OCR) is a state-of-the-art handwritten and printed text recognition model. It uses a Transformer architecture, integrating vision (encoder) and language (decoder) modules. The vision encoder extracts image features, while the text

decoder predicts word sequences. Pretrained on large datasets, TrOCR excels in end-to-end recognition, offering robust accuracy for diverse handwriting styles.

In this project, we have utilized the pretrained TrOCR model available at huggingface. Sticking to the fact that the main focus of this project is to leverage the use of Image processing techniques we use the deep learning model just for inference to give a completion to the pipeline and the project. The Word accuracy of this model is above 95% pretrained on IAM handwriting dataset. Hence, there is no necessity for finetuning the model since the dataset we gathered is a subset of IAM dataset.

4 Implementation

4.1 Pipeline

1. **Dataset and Preprocessing:**The dataset consists of around 32 images that were carefully selected for the project. Initially, a preprocessing step was performed on these images to enhance the quality of the text and minimize noise. One of the key preprocessing techniques used is Gaussian Blur, which helps in removing unnecessary information and irrelevant details from the image, such as noise or background clutter. By applying Gaussian Blur, only the essential features of the image, such as the text and key text structures, are retained, making the subsequent processing steps more effective.
2. **Line Segmentation:** After preprocessing, the next critical step in the pipeline is line segmentation. This process involves detecting and isolating individual lines of text within the image. As outlined in Section 3.1, the segmentation is performed using a series of image processing techniques that include connected component analysis, thresholding, and gap-filling. The output of this step is a set of images where each image contains one line of text. These segmented line images are then saved into a dedicated folder, which serves as the input for the next stage in the pipeline.
3. **Word Segmentation:** Once the individual text lines are successfully segmented, the next task is word segmentation. The segmented line images are individually processed using the word segmentation algorithm described in Section 3.2. This algorithm works by identifying the spatial boundaries between words within each line and separating them accordingly. The resulting images, now containing individual words instead of full lines of text, are saved into another folder for further processing in the pipeline.
4. **Word Recognition:** The final step in the pipeline is word recognition, where the segmented words are converted into machine-readable text. For this purpose, we use Hugging Face's TrOCR (Transformer-based OCR) model, as described in Section 3.3. Each word-segmented image is fed into the model, which processes the image and outputs the recognized text. The results obtained from this recognition process are saved for further evaluation and analysis. These results are also visually described in the Results section.

4.2 Resources

The pipeline was implemented entirely on CPU. Since the project requires the implementation of traditional image processing techniques, the computational resources used were not as significant as latest deep learning models. The inference used in word recognition also was implemented on CPU. Additionally, libraries such as OpenCV, skimage, numpy, PIL, Hugging Face Transformers, PyTorch and other necessary python libraries were used.

5 Results

5.1 Qualitative Analysis

5.1.1 Line Segmentation : Piecewise Painting Algorithm

The line segmentation algorithm is introduced in [1] and described in detail in section 3.1. The resultant output of each step is visualized for an image in fig 2 and fig 3.

5.1.2 Line Segmentation results

In the figure 4 given below, it can be seen that the line segmentation algorithm correctly segments cursive and non-cursive handwritten text images in (a) and (c) respectively whereas it failed in segmenting some lines in case of (b) and (d) for cursive and non-cursive respectively. The failure case can be attributed to the fact that the thresholding done in the final step of the algorithm where the components are connected to form complete lines when it internally assumes that there is no connection as can be seen in (b) or when it assumes more than one complete connection as can be seen in (d) it either gets partially segmented or over segmented.

5.1.3 Word Segmentation results

After applying the word segmentation algorithm as discussed in section 3.2, the resultant word segmentation is as shown in the fig 5. As can be seen the proposed algorithm works fine in almost all of the cases. There are a few cases where it might not work. This can happen when the threshold given is equal to some of the character widths in the words then it might oversegment the word. Other case is when there is presence of noise, the connectivity between the components is lost and the noise is considered as a word and segmented.

5.1.4 Word Recognition Results

The word recognition as described in section 3.3 yields results as shown in fig 6. The Word Recognition Rate of TrOCR is above 95% as a result of which during the inference

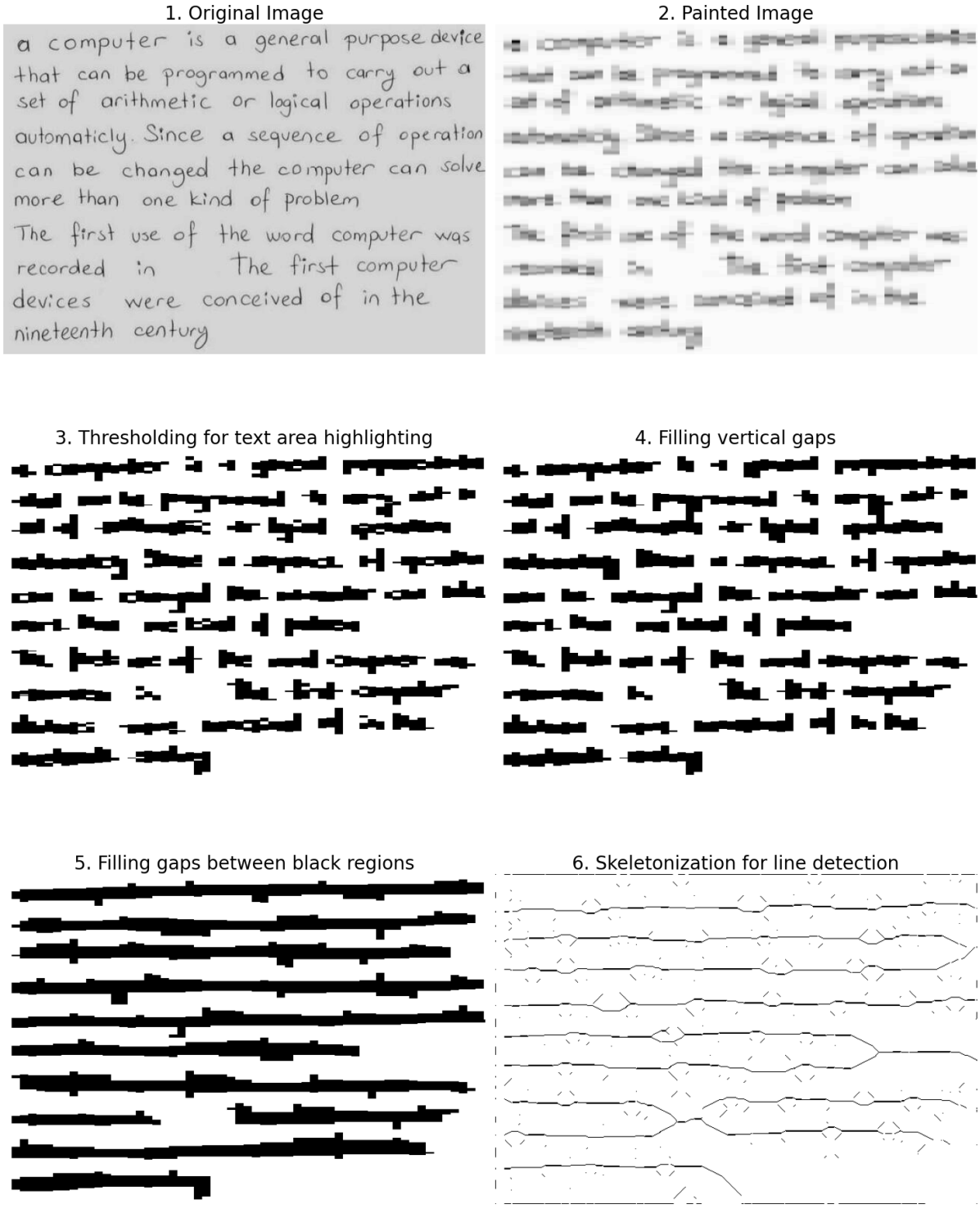


Figure 2: Piecewise Painting Algorithm Process

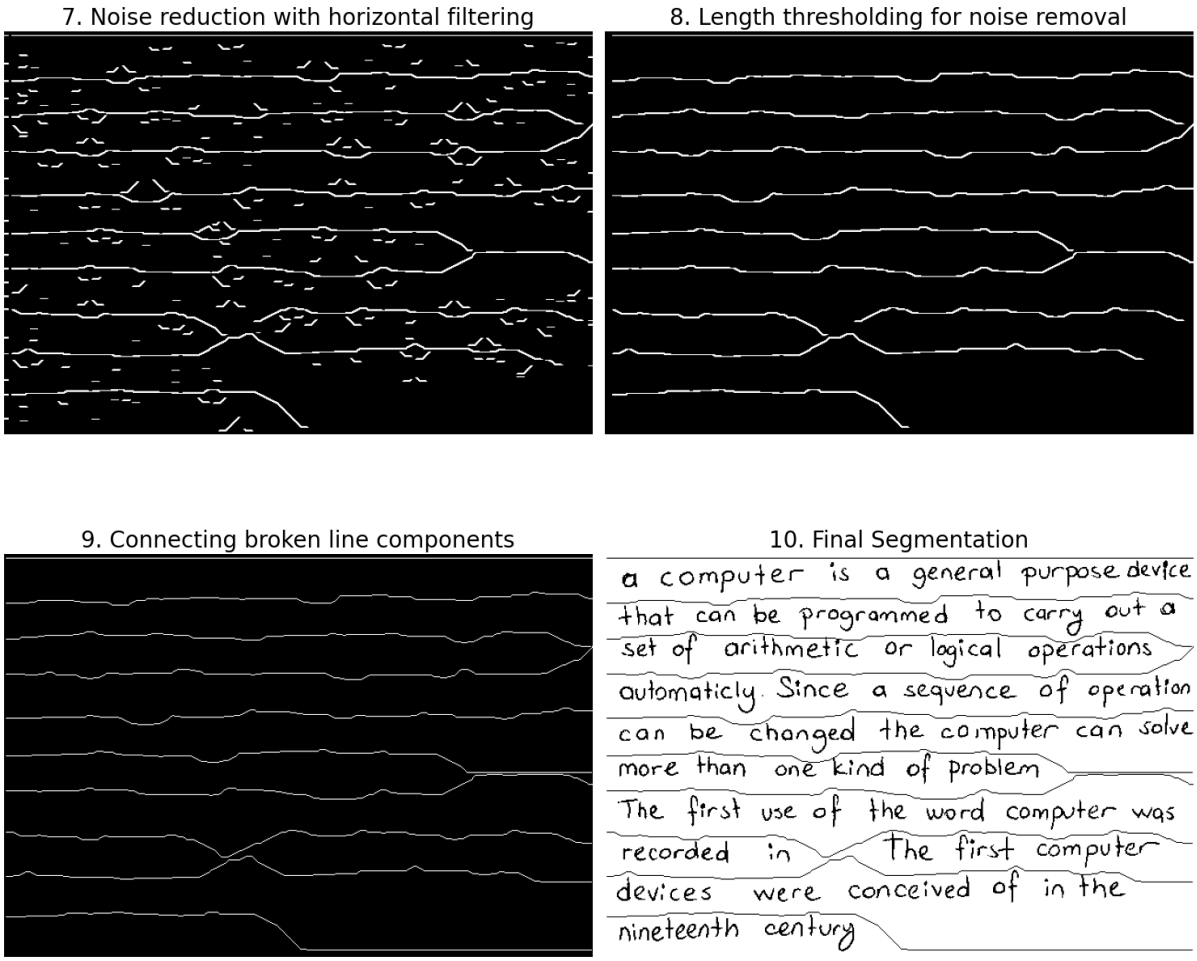


Figure 3: Piecewise Painting Algorithm Process

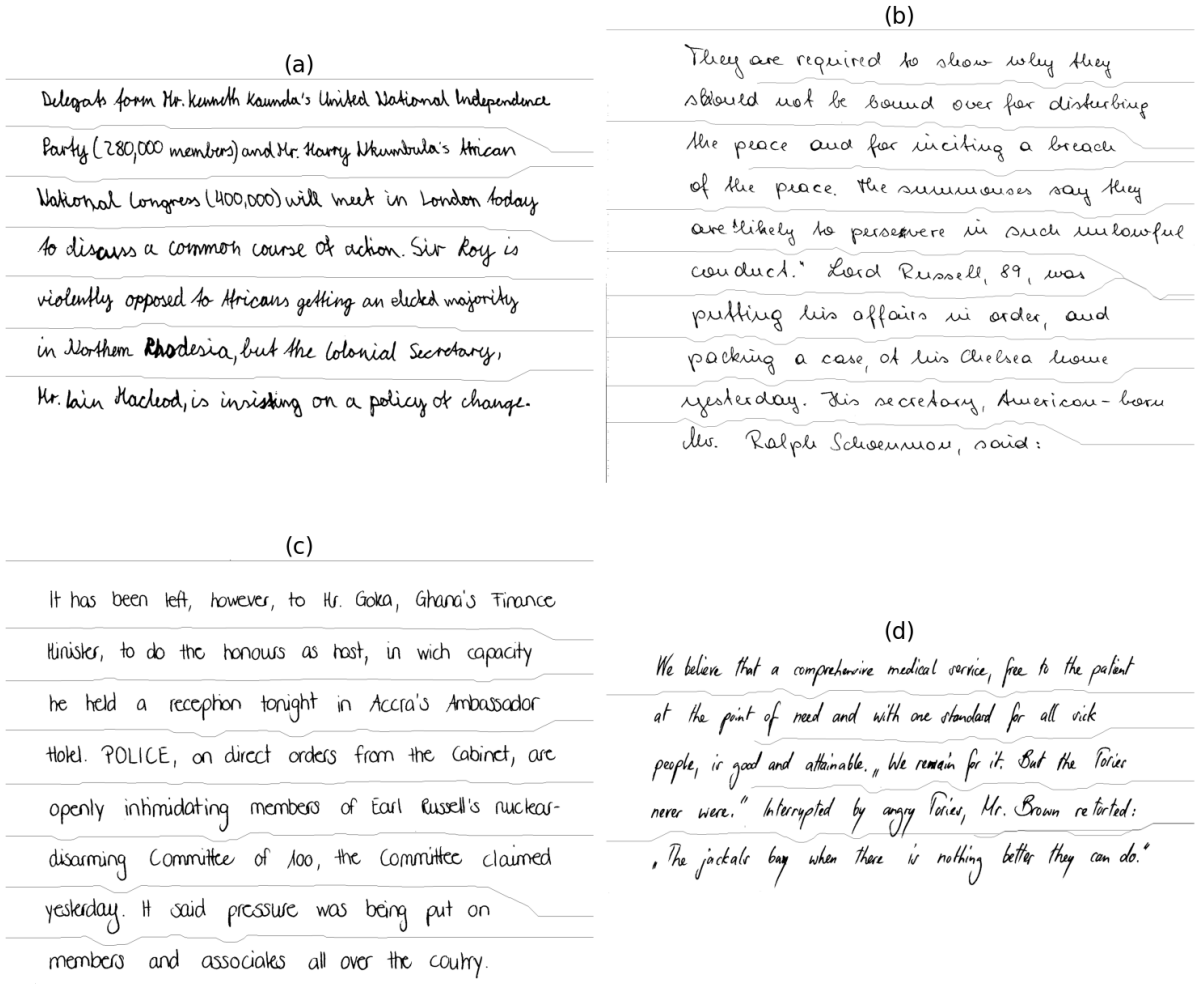


Figure 4: (a) and (c) show correct line segmentation, (b) and (d) show partially correct line segmentation of cursive and non-cursive images respectively

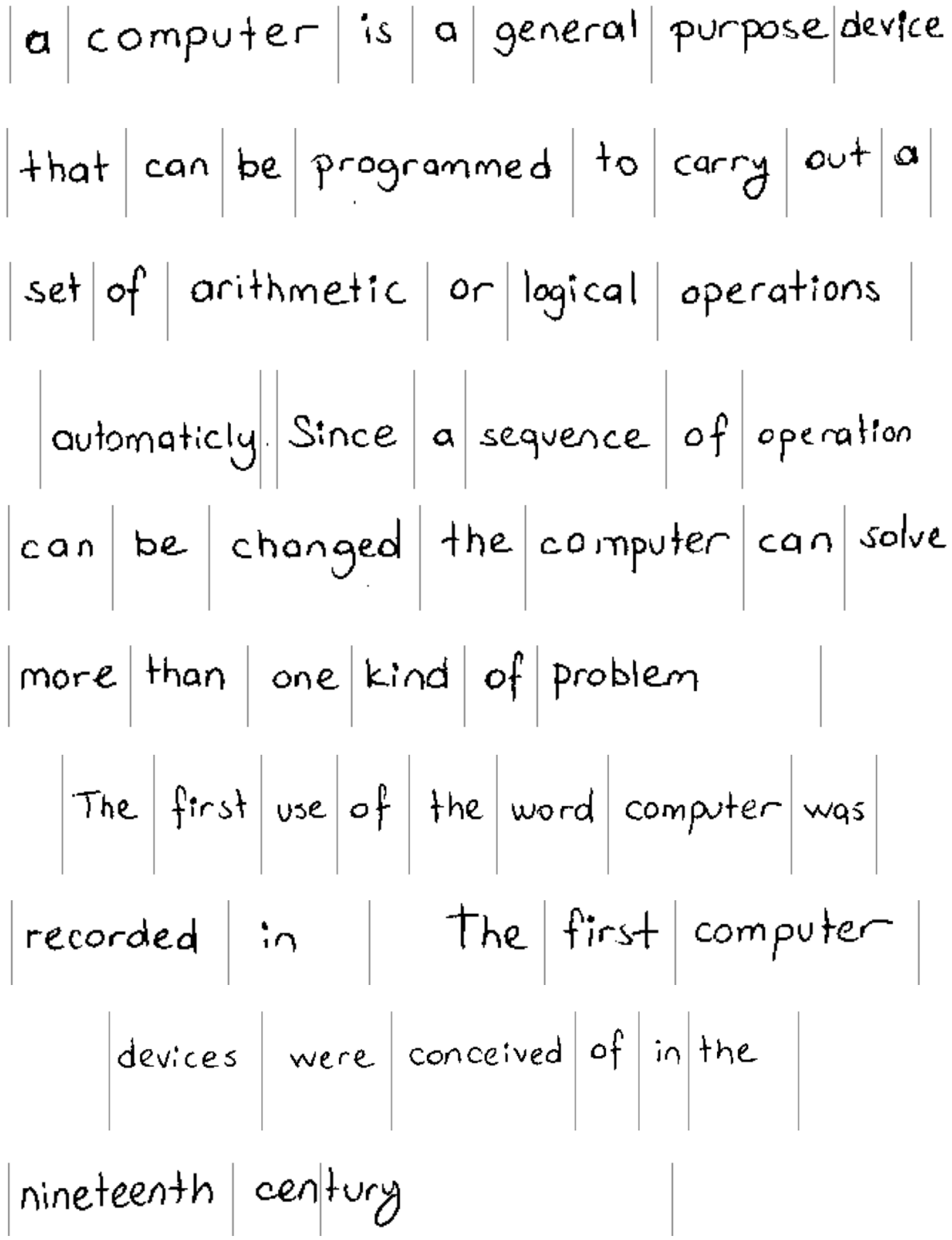


Figure 5: Word segmentation

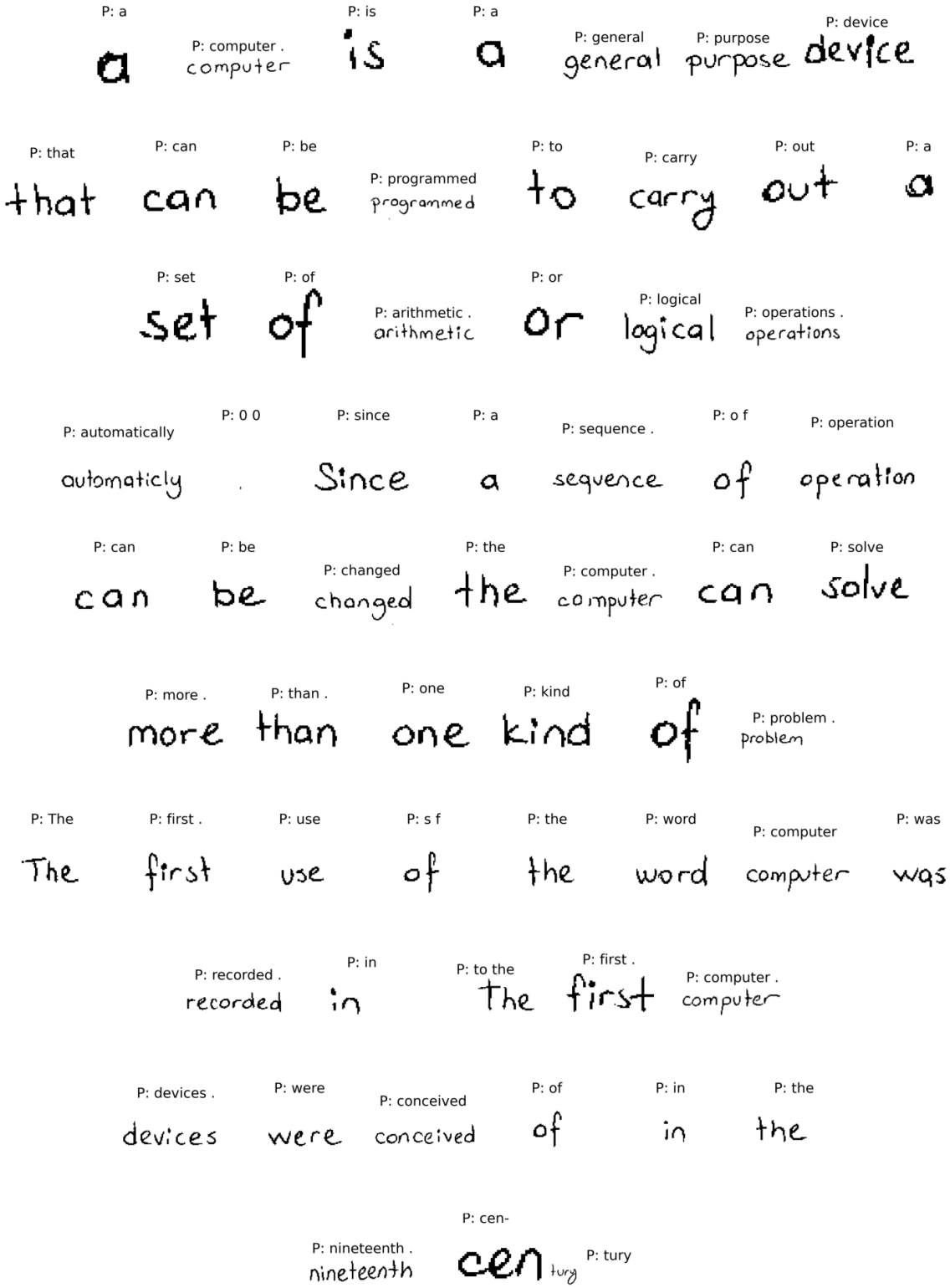


Figure 6: Word recognition

most of the words are correctly predicted. To improve the accuracy further the number of images in the dataset can be increased to increase the number of segmented words and then using these words the pretrained model can be fine-tuned.

5.2 Quantitative Analysis

Since the main focus of this project is to reimplement the Piecewise Painting algorithm [1] we calculated metrics for line segmentation only. The metric used for line segmentation is Line Segmentation Accuracy(LSA).

$$\text{LSA} = \left(\frac{\text{Number of correctly segmented lines}}{\text{Total number of lines in the ground truth}} \right) \times 100$$

Applying this formula on the calculated results obtained in this project the number of correctly segmented lines amounted to 248 and the total number of lines in the ground truth are 275 hence the LSA calculated is 90.18%. This is almost near to the one given in the paper. The given results are not compared with the one given in the paper because the dataset used by them is different from the dataset used in this project. The dataset we chose is more complicated and the number of pages selected are comparatively lower but still it was able to achieve a high LSA.

6 Conclusion

In this project we applied traditional image processing techniques for line and word segmentation, using the Piecewise Painting Algorithm and adaptive thresholds for cursive vs. non-cursive styles. The TrOCR model was integrated for high-accuracy word recognition, addressing challenges like handwriting variations, slanting text, and irregular spacing in the IAM dataset.

One of the key takeaways of this project is the demonstrated effectiveness of traditional image processing methods for segmentation. These methods, while relatively less complex, require significantly fewer computational resources and processing time compared to deep learning-based segmentation models, making them more suitable for applications with limited resources or real-time processing requirements. This simplicity, however, does not compromise their performance, as evidenced by the accurate segmentation results obtained.

This project not only underscores the importance of preprocessing and segmentation in OCR tasks but also highlights the value of combining classical methods with deep learning for end-to-end recognition systems. The results validate the efficacy of the proposed methods, making them suitable for real-world applications.

For future work, we suggest exploring more advanced preprocessing techniques to handle noisy datasets, expanding the scope to multi-lingual handwritten texts, and experimenting with fine-tuning the TrOCR model on custom datasets to further enhance its adaptability.

References

[1] Alireza Alaei, Umapada Pal, and P Nagabhushan. A new scheme for unconstrained handwritten text-line segmentation. *Pattern Recognition*, 44(4):917–928, 2011.

[2] Minghao Li, Tengchao Lv, Jingye Chen, Lei Cui, Yijuan Lu, Dinei Florencio, Cha Zhang, Zhoujun Li, and Furu Wei. Trocr: Transformer-based optical character recognition with pre-trained models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13094–13102, 2023.

[3] U-V Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International journal on document analysis and recognition*, 5:39–46, 2002.