



**Ain Shams University**

**Faculty of Computer & Information Sciences**

**Information Systems Department**

# **Advanced Ocr and Handwriting System with Correction**

**This documentation is submitted as required for the degree of bachelors  
in Computer and Information Sciences**

**By**

Riham Mohammed Hassan (20201700296)

Rahma Ali Ali Ali (20201700258)

Mohammed Essam Mohammed (20201700713)

Omar Sherif Soliman (20201700536)

Sara Mohammed Abd-Elmonaem(20201700334)

Moamen Said Mohammed(20191700914 )

**Supervised by:**

**Dr. Ahmed Ezzat**

**June 2024**

## Acknowledgments

We extend our heartfelt gratitude to all those who have contributed to the success of our project. We would like to express our special appreciation to the Graduation Project Unit for their unwavering support and provision of valuable information, paving the way for students to apply their educational knowledge in real-world project design and analysis. We are sincerely grateful to our professors and industry experts for delivering engaging lectures that have greatly benefited us throughout our academic journey. Their insights and expertise have been invaluable. A special word of thanks goes to our project supervisor, Dr. Ahmed Ezzat,, for her enthusiastic encouragement and valuable feedback during this research. Her guidance has been instrumental in shaping our work. We also wish to acknowledge our parents and the members of Ain Shams University for their kind cooperation and unwavering support throughout this project.

# Abstract

Our project addresses the complex task of Optical Character Recognition (OCR) for both printed and handwritten text, aiming to enhance accuracy and efficiency in digitizing textual information from diverse sources like scanned documents and images. This development is crucial across multiple sectors: in business, it improves data entry efficiency and enhances digital workflows by accurately processing documents and invoices; in education, it aids in digitizing handwritten notes, making them searchable and accessible for both students and educators; in healthcare, it streamlines medical record management and prescription processing, ensuring more efficient documentation; and in the legal and financial sectors, it automates contract processing and improves accuracy in handling financial documents. The project employs a multifaceted approach, including image preprocessing techniques like noise reduction and binarization to enhance image quality, advanced text detection methods to identify text regions within complex images, and a combination of traditional OCR and modern algorithms to ensure effective recognition of both printed and handwritten text. Additionally, the system integrates multiple correction models to enhance accuracy by rectifying spelling errors and refining outputs. Through these strategies, the project achieves significant improvements in OCR accuracy and reliability, offering a robust solution for accurate text recognition and digitization to meet the increasing demand for efficient data handling across various industries.

# Table of Contents

|  |                              |
|--|------------------------------|
| ABSTRACT .....   | III                          |
| TABLE OF CONTENTS .....                                    | IV                           |
| LIST OF FIGURES .....                                      | VI                           |
| LIST OF TABLES .....                                       | VIII                         |
| LIST OF ABBREVIATIONS .....                                | ERROR! BOOKMARK NOT DEFINED. |
| <b>CHAPTER 1 : INTRODUCTION .....</b>                      | <b>1</b>                     |
| 1.1 INTRODUCTION .....                                     | 2                            |
| 1.2 PROBLEM DEFINITION .....                               | 2                            |
| 1.3 MOTIVATION .....                                       | 2                            |
| 1.4 OBJECTIVES .....                                       | 2                            |
| 1.5 METHODOLOGY .....                                      | 3                            |
| 1.6 TIME PLAN .....  | 4                            |
| 1.7 THESIS OUTLINE .....                                   | 4                            |
| <b>CHAPTER 2 : LITERATURE REVIEW .....</b>                 | <b>6</b>                     |
| 2.1 INTRODUCTION .....                                     | 7                            |
| 2.2 THEORETICAL BACKGROUND .....                           | 7                            |
| 2.2.1 Traditional Approaches .....                         | 7                            |
| 2.2.2 Modern Techniques in OCR .....                       | 7                            |
| 2.2.3 Preprocessing Techniques .....                       | 7                            |
| 2.2.4 Text Detection and Localization .....                | 8                            |
| 2.3 RELATED WORK .....                                     | 8                            |
| <b>CHAPTER 3 : SYSTEM ARCHITECTURE AND METHODS .....</b>   | <b>10</b>                    |
| 3.1 INTRODUCTION .....                                     | 11                           |
| 3.2 SYSTEM ANALYSIS AND DESIGN .....                       | 11                           |
| 3.2.1 System Architecture .....                            | 11                           |
| 3.2.2 Functional Requirements .....                        | 12                           |
| 3.2.3 Non-Functional Requirements .....                    | 12                           |
| 3.2.4 System Users .....                                   | 12                           |
| 3.2.5 UML Diagrams .....                                   | 13                           |
| 3.2.5.1 Use Case Diagram .....                             | 13                           |
| 3.2.5.2 Class Diagram .....                                | 14                           |
| 3.2.5.3 Sequence Diagrams .....                            | 15                           |
| 3.3 METHODS AND PROCEDURES USED .....                      | 16                           |
| <b>CHAPTER 4 : SYSTEM IMPLEMENTATION AND RESULTS .....</b> | <b>22</b>                    |
| 4.1 INTRODUCTION .....                                     | 23                           |
| 4.2 MATERIALS USED .....                                   | 23                           |
| 4.3 SOFTWARE TOOLS .....                                   | 24                           |
| 4.4 HARDWARE SPECIFICATIONS .....                          | 25                           |
| 4.5 EXPERIMENTAL AND RESULTS .....                         | 26                           |
| 4.5.1 OCR MODEL .....                                      | 26                           |
| 4.5.2 OCR MODEL COMPONENTS .....                           | 26                           |
| 4.5.2.1 OpenCV2 .....                                      | 26                           |

|  |           |
|--|-----------|
| 4.5.2.2 NumPy.....   | 27        |
| 4.5.2.3 Pytesseract.....   | 27        |
| 4.5.2.4 EAST Model.....  | 27        |
| 4.6 SUMMARY AND RESULTS .....  | 27        |
| 4.7 PERFORMANCE AND CONCLUSION .....                                     | 28        |
| 4.7.1 Handwritten Text Recognition model.....                            | 28        |
| 4.7.2 OCR Methods .....  | 28        |
| 4.7.2.1 Pytesseract.....   | 28        |
| 4.7.2.2 Keras-OCR.....   | 29        |
| 4.7.2.3 Easy-OCR.....  | 30        |
| 4.7.2.4 Merging Keras and Pytesseract.....                               | 30        |
| 4.7.2.5 Summary.....   | 31        |
| 4.7.3 Text Correction.....   | 31        |
| 4.7.3.1 Correction Models Evaluation.....                                | 31        |
| 4.7.3.1.1 TextBlob .....   | 31        |
| 4.7.3.1.2 PySpellChecker .....   | 33        |
| 4.7.3.1.3 SymSpell .....   | 34        |
| 4.7.3.1.4 JamSpell .....   | 35        |
| 4.7.3.1.5 The combination of SymSpell, JamSpell, and Pyspellchecker..... | 36        |
| 4.7.4 Conclusion.....  | 37        |
| <b>CHAPTER 5 : RUN THE APPLICATION .....</b>                             | <b>39</b> |
| 5.1 INTRODUCTION.....  | 40        |
| 5.2 RUN THE APPLICATION.....   | 40        |
| <b>CHAPTER 6 : CONCLUSION AND FUTURE WORK .....</b>                      | <b>55</b> |
| 6.1 CONCLUSION .....   | 56        |
| 6.2 FUTURE WORK.....   | 57        |
| <b>REFERENCES .....</b>  | <b>85</b> |

## List of Figures

|  |    |
|--|----|
| Figure 1:1 Time Plan.....  | 4  |
| Figure 3:1 System Architecture .....   | 11 |
| Figure 3:2 Use Case Diagram .....  | 13 |
| Figure 3:3 Class Diagram .....   | 14 |
| Figure 3:4 Sequence Diagram for App .....  | 15 |
| Figure 4:1 output example Pytesseract .....  | 29 |
| Figure 4:2 Ground truth .....  | 29 |
| Figure 4:3 Accuracy Measures.....  | 29 |
| Figure 4:4 Keras-OCR output .....  | 30 |
| Figure 4:5 Easy-OCR output.....  | 30 |
| Figure 4:6 Merging Keras and Pytesseract Accuracy .....                                | 31 |
| Figure 5:1 After running the application, it will redirect you to the login page. .... | 40 |
| Figure 5:2 Sign up.....  | 41 |
| Figure 5:3 login in.....   | 41 |
| Figure 5:4 The home page.....  | 42 |
| Figure 5:5 take a picture .....  | 43 |
| Figure 5:6 upload a picture .....  | 44 |
| Figure 5:7 select model .....  | 45 |
| Figure 5:8 select printed model.....   | 46 |

|  |    |
|--|----|
| Figure 5:9 select handwritten model .....              | 47 |
| Figure 5:10 text extraction.....                       | 48 |
| Figure 5:11 text extraction_2 .....                    | 48 |
| Figure 5:12 Trying Printed Text Model .....            | 49 |
| Figure 5:13From setting change the font style .....    | 50 |
| Figure 5:14 From setting change the font family .....  | 51 |
| Figure 5:15 from setting change the font size.....     | 52 |
| Figure 5:16 from setting switch to the dark mode ..... | 54 |

## List of Tables

|   |    |
|---|----|
| Table 4.1 Software Tools .....  | 24 |
| Table 4.2 the system's hardware requirements.....                         | 25 |
| Table 4.3 Results 1 for AvgErrorrate before and after TextBlob .....      | 32 |
| Table 4.4 Results 2 for Error rate before and after TextBlob .....        | 32 |
| Table 4.5 Results 1 for AvgErrorrate before and after PySpellChecker..... | 33 |
| Table 4.6 Results 2 for Error rate before and after PySpellChecker.....   | 33 |
| Table 4.7 Results 1 for AvgErrorrate before and after Symspell.....       | 34 |
| Table 4.8 Results 2 for Errorrate before and after Symspell .....         | 34 |
| Table 4.9 Results 1 for AvgErrorrate before and after JamSpell .....      | 35 |
| Table 4.10 Results 2 for Error rate before and after JamSpell .....       | 36 |
| Table 4.11 Results for AvgErrorrate before and after combination.....     | 37 |
| Table 4.12 Results for Error rate before and after combination.....       | 37 |



# **Chapter 1 : Introduction**

## 1.1 Introduction

Optical Character Recognition (OCR) is a transformative technology that converts text from images or scanned documents into machine-readable data. This process enhances efficiency, facilitates data analytics, and integrates seamlessly into modern workflows. OCR plays a crucial role in various industries, such as finance, healthcare, legal, and education, streamlining processes and improving accessibility.

## 1.2 Problem Definition

The problem with traditional OCR systems lies in their limitations in recognizing and extracting content from handwritten text. Inaccuracies and misinterpretations often occur without smart correction algorithms. Our goal is to address these challenges by developing an OCR system that demonstrates proficiency in accurately recognizing both printed and handwritten text.

## 1.3 Motivation

Developing advanced OCR (Optical Character Recognition) and handwritten text recognition systems is crucial for enhancing accessibility and efficiency across diverse domains. These systems accurately convert printed text and handwritten notes into digital formats, ensuring universal access to information, including for individuals with visual impairments or language barriers. They also automate document management by swiftly converting physical documents into searchable, editable digital formats, thus saving time and reducing errors.

In summary, advanced OCR and handwritten text recognition systems significantly contribute to a more interconnected, efficient, and inclusive society by improving information accessibility, preserving cultural heritage, fostering innovation, and supporting educational and business requirements.

## 1.4 Objectives

Developing an OCR and handwritten system with using correction :

- Accurate text recognition .

- Robust text detection.
- Multi algorithm integration.
- Efficiency and speed.

## 1.5 Methodology

This project develops an advanced OCR system for accurate recognition of both printed and handwritten text using a multi-stage methodology. The process begins with data acquisition, including collecting and annotating diverse text images. Preprocessing steps like noise reduction and binarization enhance image quality. Text detection algorithms, such as EAST, identify text regions, while OCR models (e.g., Pytesseract) handle printed text, and specialized models (e.g., Keras-OCR, Easy-OCR) address handwritten text. These models are trained and fine-tuned on custom datasets to improve accuracy. A hybrid approach merges outputs from different models to leverage their strengths. Text correction models, including PySpellChecker and custom contextual models. By following this methodology, the project aims to develop a comprehensive OCR system that use diverse techniques and continuous improvement processes ensures the system's robustness and reliability across various applications.

## 1.6 Time plan



Figure 1.1 time plan

. The timeline, as depicted in Figure 1.1, was created to outline the key dates and milestones for the project's execution.

## 1.7 Thesis Outline

This project consists of six chapters in addition to appendixes. These chapters are designed to represent the scientific steps we've taken toward our core goal. The following paragraphs provide a summary of the contents of each chapter:

### Chapter 1: Introduction

This chapter provides an introduction to the project, encompassing the project overview, problem definition, motivation, objectives, methodology, and time plan.

### Chapter 2: Literature Review

This chapter offers an exploration of the theoretical background, as well as a review of previous studies and works relevant to the project. It establishes a foundation of knowledge and understanding within the field, providing context for the current study, and highlighting the existing research and contributions made by others.

### Chapter 3: System Architecture and Methods

This chapter encompasses the system architecture, methods, and procedures employed in the project. It presents a detailed description and visualization of the system architecture.

#### **Chapter 4: System Implementation and Results**

This chapter includes the dataset used in the project, the software tools employed, the setup configuration, and a comprehensive account of the experiments conducted along with their corresponding results.

#### **While chapter 5: Run the Application.**

This chapter provides a step-by-step guide on how to run the application.

#### **Finally, in Chapter 6: Conclusion.**

The conclusion and future work section of this report serve as a reflection on the project's outcomes, offering insights into the project's significance and potential for further improvement and expansion.

## **Chapter 2 : Literature Review**

## 2.1 Introduction

In this chapter, we will conduct a literature review to compare, analyze, explore, and comprehend the various efforts and directions taken to identify research gaps. Throughout this review, we aim to shed light on the future potential and scope of our project.

## 2.2 Theoretical Background

The theoretical foundation of the project revolves around the principles and procedures of OCR and handwritten text recognition. OCR involves extracting text from images, which traditionally relied on pattern recognition and template matching. However, advancements in machine learning, particularly deep learning techniques like convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have revolutionized OCR by enabling more robust and accurate text recognition. Handwritten text recognition, a more complex subset of OCR, addresses the variability in handwriting styles and structures. Techniques such as transfer learning and hybrid model architectures have been pivotal in improving the accuracy and versatility of OCR systems.

### 2.2.1 Traditional Approaches

**Early OCR Systems:** Review the foundational methods of OCR, including template matching, pattern recognition, and feature extraction.

**Handwritten Text Recognition:** Discuss early attempts at recognizing handwritten text and the challenges faced, such as variability in handwriting and cursive writing.

### 2.2.2 Modern Techniques in OCR

**Machine Learning-Based Methods:** Explore the shift from traditional methods to machine learning approaches, focusing on models like support vector machines (SVMs) and random forests.

**Deep Learning Approaches:** Analyze the impact of deep learning on OCR, including the use of convolutional neural networks (CNNs), recurrent neural networks (RNNs), and long short-term memory networks (LSTMs).

### 2.2.3 Preprocessing Techniques

**Noise Reduction:** Review methods for enhancing image quality by removing noise.

**Binarization and Normalization:** Discuss techniques for converting images to binary format and standardizing their size and resolution.

## 2.2.4 Text Detection and Localization

**Algorithms and Models:** Examine different algorithms used for text detection, such as the EAST (Efficient and Accurate Scene Text) detector.

**Challenges:** Identify challenges in text detection, particularly in complex backgrounds and varying lighting conditions.

## 2.3 Related Work

In this section, we compare systems that are similar to our project and describe the tools, processes, and results of each system.

**In [1][4],** provides valuable insights into the challenges, methodologies, and outcomes related to OCR and information extraction from scanned receipts. Incorporating these lessons into your project can guide the development of a more effective and robust OCR system capable of handling both printed and handwritten texts with high accuracy and efficiency.

**In[2][6],** provides valuable insights, such as lightweight model design, and open-source collaboration, can enrich the development of OCR and handwritten text recognition project. These strategies not only enhance performance and efficiency but also broaden the scope of potential applications across different domains.

**In[3][5],** provides valuable insights, such as leveraging proven neural network architectures, addressing challenges in irregular text recognition, optimizing annotation strategies, and fostering openness in code and model sharing, you can enhance the effectiveness and applicability of your OCR and handwritten text recognition system across diverse use cases and scenarios.

**In our proposed system,** In our project focused on OCR and handwritten text recognition systems, our primary goal was to develop an Android application that efficiently performs these functions. we utilized techniques such as data acquisition, preprocessing, noise reduction, and binarization to enhance image quality. Text detection



algorithms like EAST were employed to identify text regions, while OCR models such as Pytesseract handled printed text, and specialized models like Keras-OCR and Easy-OCR addressed handwritten text. These models were trained and fine-tuned on custom datasets to boost accuracy. Additionally, text correction was implemented using models such as PySpellChecker and custom contextual models. We employed these techniques to efficiently perform the functions of OCR and handwritten text recognition in our project.

## **Chapter 3 :SystemArchitecture and Methods**

### 3.1 Introduction

This chapter will mainly focus on system analysis and design, covering various aspects such as UML diagrams, system architecture, system users, functional and non-functional requirements, and database schema. Additionally, we will delve into the methods and procedures utilized in our project.

#### 3.2.1 System Architecture

Our proposed system comprises three modules, as depicted in Figure 3.1. The initial module encompasses the user interface, where user upload picture has text , and receive text in this picture The second module is the application layer, which takes picture as input, processes it and extract the text in it . The results are then presented as output to the user. The third module involves user information and Uploaded Images.

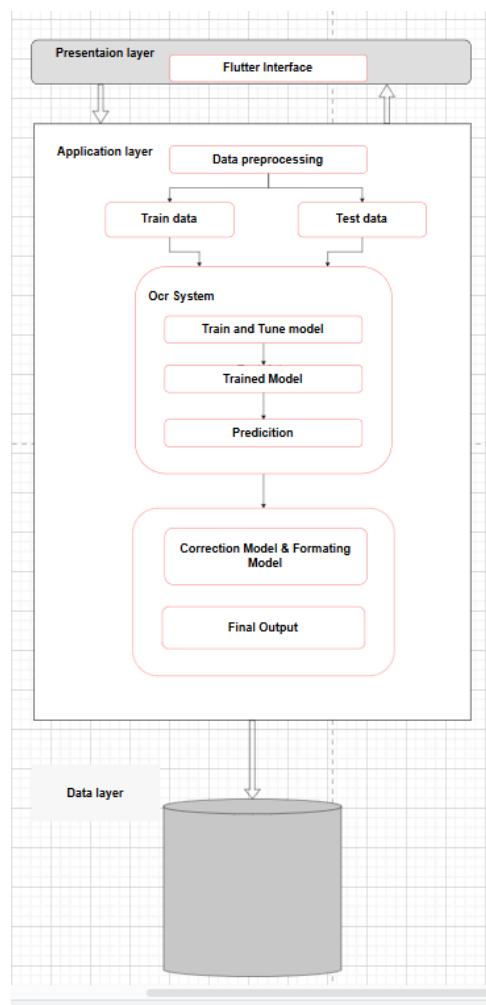


Figure 3:1 System Architecture

### 3.2.2 Functional Requirements

The proposed system provides functionalities

#### 1) **user Functionality:**

- The user must be able to register on the app.
- The user must be able to log in and log out of the app.
- The user must Upload/Take a photo.
- The user must be able to crop the picture to select the desired area for OCR.
- The user must choose between printed or handwritten text recognition model
- The user can change the font style.
- The user can change the font size.
- The user can change the font family.
- The user can choose between dark and light modes.

#### 2) **The Model Functionality:**

- OCR Processing
- Text Extraction from Images
- Smart Correction

### 3.2.3 Non-Functional Requirements

Non-functional requirements are the characteristics and features that must be available in the system.

- 1) **Usability:** The destinations must be clear to the users and there is no ambiguity in them. There are explanatory texts for the operation of each of the system buttons.
- 2) **Efficiency:** The system must be efficient, work well, and serve the largest number of users
- 3) **Security:** The data in the system must have a high degree of protection against penetration or loss.
- 4) **Reliability:** The system must operate continuously and reliably and prevent incorrect data from being entered.

### 3.2.4 System Users

**Intended Users:**

- **General Consumers :** Individuals who need to digitize documents,
- **Business Professionals** Office workers who need to scan and digitize documents for archiving, sharing, and editing
- **Educational Institutions:** Teachers and administrators who need to digitize educational materials, exams, and assignments.

## 3.2.5 UML Diagrams

### 3.2.5.1 Use Case Diagram

The key requirement for a use case diagram is "how a system interacts with its environment involves a diagram and a description to illustrate the discrete activities that the users conduct," which can be created based on the functional requirements. as depicted in figure (3.2):

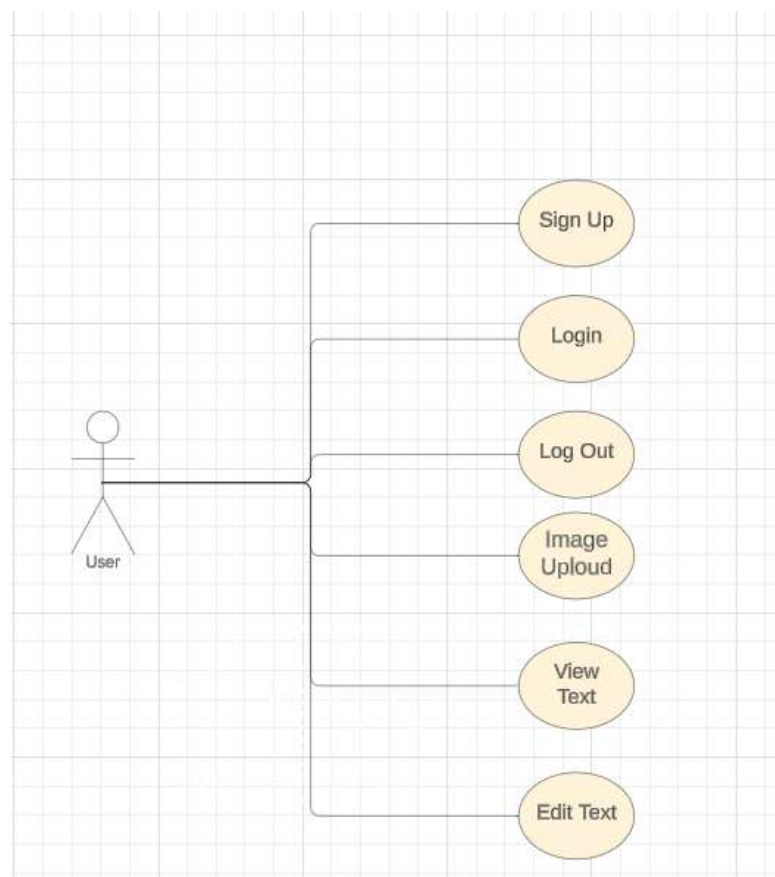


Figure 3:2 Use Case Diagram

### 3.2.5.2 Class Diagram

A class diagram in software engineering is a form of a static structure diagram that displays the classes, their characteristics, actions (or methods), and relationships between objects to illustrate the structure of a system. As depicted in Figure (3.3).

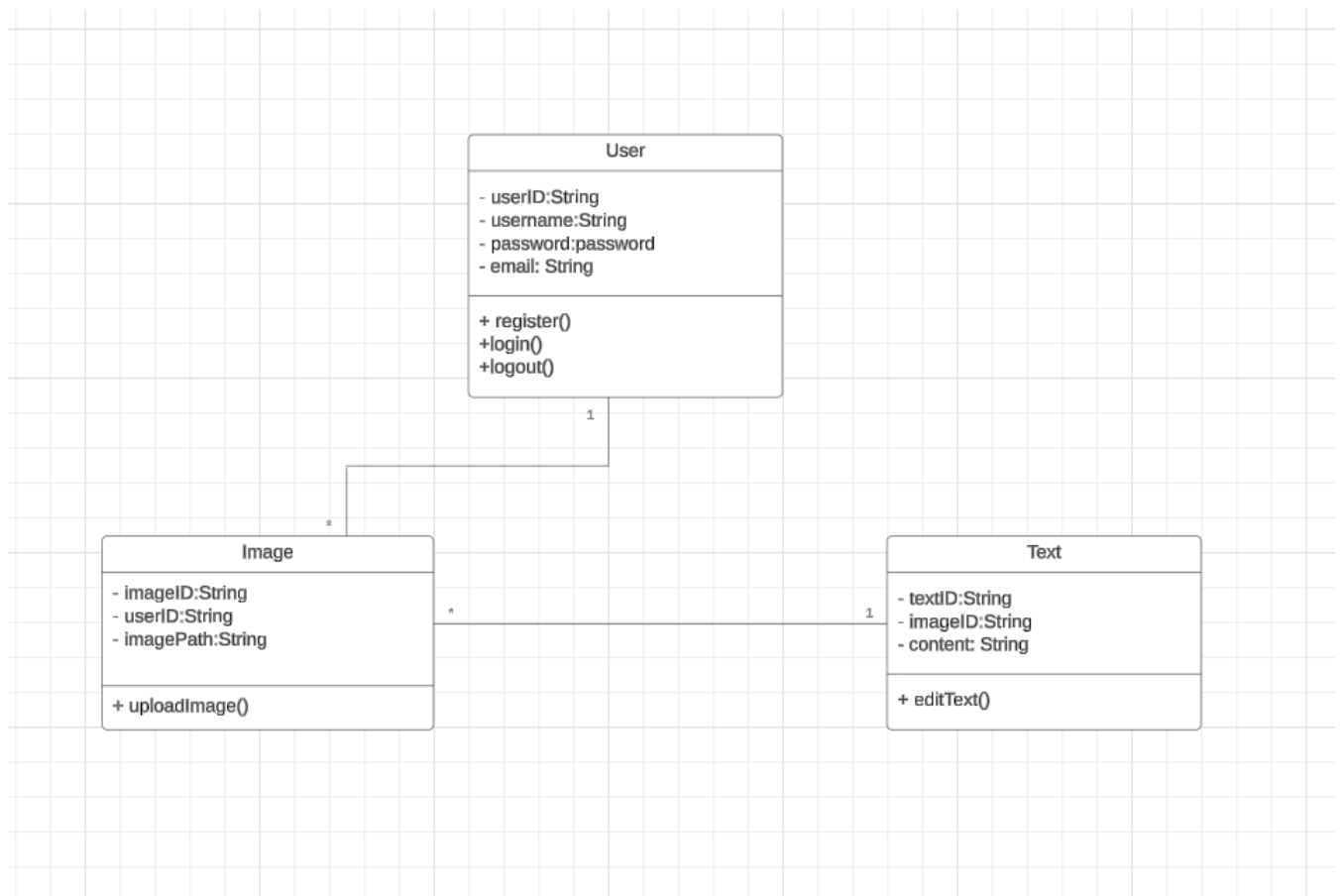


Figure 3:3 Class Diagram

### 3.2.5.3 Sequence Diagrams

Developers frequently use sequence diagrams to model the interactions between items in a single use case. They demonstrate the interactions that take place when a specific use case is executed and the order in which various system components interact with one another to perform a function. As depicted in Figure (3.4),(3,5).

#### 1. Sequence Diagram for App

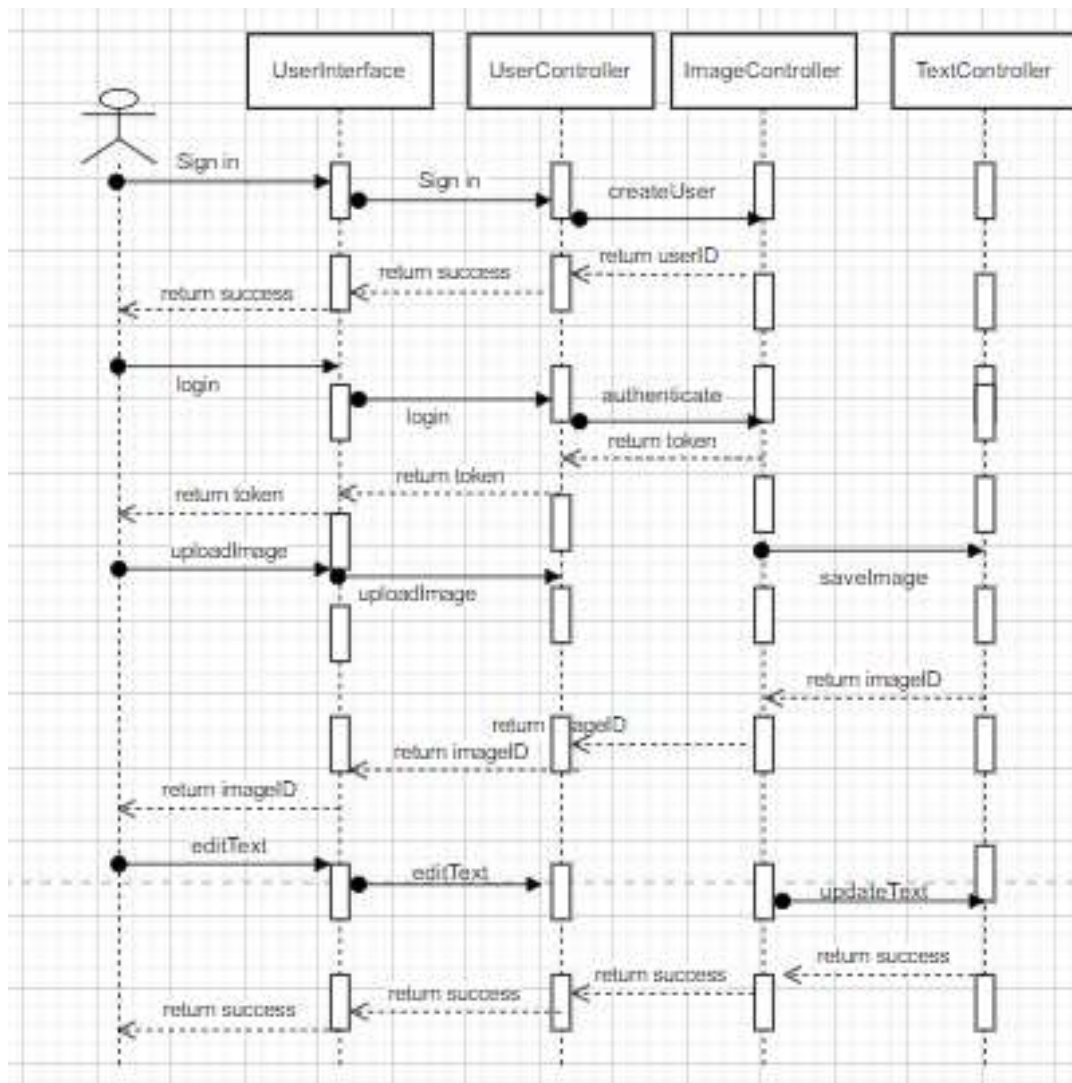
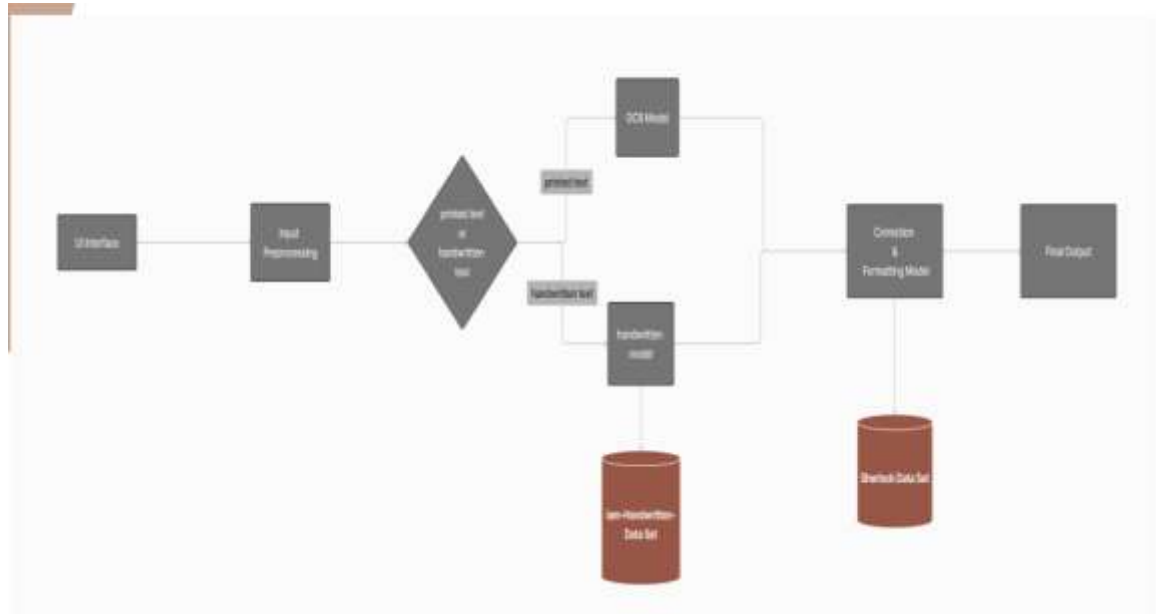


Figure 3:4 Sequence Diagram for App

### 3.3 Methods and Procedures Used



*Figure 3:5 Methods and Procedures Used*



In our OCR and handwritten text recognition project, we employed a variety of well-established and customized techniques to ensure efficient and accurate performance. The project consists of three main steps: data input, preprocessing, and OCR model implementation. Below, we describe these methods and procedures in detail:

### **1. Data Input :**

We collected a comprehensive dataset of printed and handwritten documents to train and evaluate our models. This dataset includes various printed and handwriting styles to ensure robustness and versatility in our system.

### **2. Preprocessing:**

Preprocessing is a crucial step to enhance image quality and prepare data for accurate text recognition.

**We used the following techniques:**

#### **OpenCV2 (Open Source Computer Vision Library):**

- **Loading and Running Model:** We utilized OpenCV2 to load and run our OCR models. OpenCV2's real-time image processing capabilities are essential for tasks such as character detection, text extraction, and document analysis.
- **Image Preprocessing:** Techniques such as thresholding, Gaussian Blur, and conversion to grayscale were applied to improve image quality and facilitate better text detection.
- **Rotated Bounding Boxes and NMS:** We applied rotated bounding boxes and Non-Maximum Suppression (NMS) to accurately detect and delineate text regions, reducing redundant and overlapping boxes.
- **Drawing Detected Bounding Boxes:** OpenCV2 was used to draw bounding boxes around detected text regions, visually highlighting the text areas.
- **NumPy (Numerical Python):**
  - 1-Creating Image Arrays: NumPy was used to efficiently handle and manipulate numerical data, particularly in creating image arrays.

2-Array Operations: NumPy's array operations facilitated the creation of blobs used as input for the models, streamlining the preprocessing workflow.

3-Image Arrays Preprocessing: Additional preprocessing steps were implemented using NumPy to ensure the data fed into our models was of high quality.

### **Preprocessing on Handwritten Images:**

- **Convert Image to Grayscale:** Converting images to grayscale helps to better separate text from the background.
- **Apply Threshold:** Thresholding converts grayscale images to binary (black and white) images, enhancing clarity by turning higher intensity pixels to white and lower intensity pixels to black.
- **Contours (Not Used):** While contours can help find object boundaries in an image, they were not used as they might mislead the model.

### **3. OCR Model Implementation:**

Our OCR model implementation involved several key components and methods:

EAST (Efficient and Accurate Scene Text) Model:

- **Usage:** The EAST model is employed for detecting text regions in natural scene images. It predicts the geometry (bounding box and rotation angle) and a confidence score for each text region.
- **Architecture:** The model architecture combines feature extraction, geometry prediction, and score prediction, processing the entire image in a single forward pass.
- **Geometry Prediction:** This component predicts the bounding box coordinates and the rotation angle of the text.
- **Score Prediction:** This predicts a confidence score for each detected text region, indicating the likelihood of containing text.
- **Non-Maximum Suppression (NMS):** NMS is applied as a post-processing step to remove redundant and overlapping bounding boxes, improving the precision of detected text regions.

## Handwritten Text Recognition Models:

### 1- Pytesseract:

- Usage: Pytesseract, an OCR tool based on Tesseract, provides an interface to Google's Tesseract-OCR for recognizing text from images and scanned documents.
- Pros: Simple interface, cross-platform compatibility, support for multiple languages, and customizable output formats (e.g., plain text, HTML).

### 2- Keras-OCR:

- Usage: An open-source library providing pretrained models for OCR tasks.
- Pros: Pretrained model, supports various input formats (e.g., PNG, JPEG).
- Cons: Output is not sorted, making sentiment analysis difficult, and the processing time is longer than other models.

### 3- Easy-OCR:

- Pros: Multi-language support, cross-platform compatibility, efficient performance.
- Cons: Less efficient than Pytesseract, requires language specification before use, variable accuracy.

**Merge Between Pytesseract and Keras-OCR:** To improve similarity and reduce character error rate (CER), we applied a hybrid approach by running Keras-OCR first and then using Pytesseract to replace similar text. This approach leverages Pytesseract's sorted output and higher accuracy.

## 4. Text Correction Methods

In the realm of text processing and natural language understanding, ensuring the accuracy and clarity of textual content is paramount. We explored various popular and effective text correction methods to enhance the accuracy of the recognized text:

### 1- TextBlob:

- Usage: TextBlob is a Python library for processing textual data, built on top of NLTK and Pattern. It provides a simple API for NLP tasks such as part-of-speech

tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

- Pros: Effective for spell checking and correction based on word frequencies in the English language.
- Cons: May struggle with context-based errors due to its reliance on dictionary lookups and language-specific rules.

#### 2- PySpellChecker:

- Usage: PySpellChecker is a Python package built on the Hunspell library, providing efficient spelling correction based on word frequencies in the English language.
- Pros: Simple and efficient, cross-platform, and supports multiple languages.
- Cons: May not always produce perfect results, especially for context-based errors or words not in the dictionary.

#### 3- JamSpell:

- Usage: JamSpell is a spell checking and correction library leveraging Hunspell and various techniques like language models, Levenshtein distance, and phonetic similarity to suggest contextually appropriate corrections.
- Pros: Strong ability to make context-based corrections using surrounding words in a sentence.
- Cons: Still faces challenges with complex sentences or multiple errors.

#### 4- SymSpell:

- Usage: SymSpell is a Python library using the Symmetric Delete spelling correction algorithm, efficient and accurate for handling compound words and various types of errors.
- Pros: Efficient compound-aware spelling correction, useful for languages with compound words.
- Cons: Limited by the quality of the dictionary and may not handle all types of errors.

## Combined Text Correction Approach

To create a comprehensive text correction system, we combined the strengths of SymSpell, JamSpell, and PySpellChecker:

- SymSpell: Used first to address compound word errors, handling splitting, concatenation, substitution, transposition, deletion, and insertion errors.
- JamSpell: Applied next for context-based corrections, leveraging language models, edit distance, and phonetic similarity.
- PySpellChecker: Used finally to catch any remaining spelling errors, ensuring individual words are correct.

By combining these techniques, our system can handle a wide range of errors, from simple spelling mistakes to complex context-based errors, providing a robust and accurate text correction solution.

## 5. Evaluation

The final evaluation step involved integrating the OCR and handwritten text recognition capabilities into an Android application

## **Chapter 4 : System Implementation and Results**

## 4.1 Introduction

This chapter will cover various aspects of our study, including the dataset used, the software programs utilized, the configuration setup, as well as the experimental procedures, and the obtained results.

## 4.2 Materials Used

In developing our OCR and handwritten text recognition system, we utilized a variety of materials and datasets to ensure comprehensive training and testing. The materials used include

### **Printed and Handwritten Datasets:**

We gathered diverse datasets of printed and handwritten documents from several reputable sources to train and evaluate our models effectively.

Sites and Sources:

**MNIST Database:** A widely used database for handwritten digits, providing a foundational dataset for training handwritten recognition models.

**COCO Text Dataset:** A dataset containing complex scenes with dense text annotations, used to train our scene text recognition models.

**IAM Handwriting Database:** A collection of handwritten English text samples, including forms, letters, and historical documents, used to train and test handwritten text recognition.

**Synthetic Data Generation:** Additionally, synthetic datasets were generated using tools like Python scripting and augmentation techniques to diversify the training data and improve model generalization.

These datasets were selected for their diversity, ensuring our system's robustness across different text recognition tasks.

## 4.3 Software Tools

This section provides an overview of the software tools used in our proposed system, as listed in Table 4.2. Each tool is selected for its specific features and contributions to the system's overall functionality. We will briefly describe each tool's purpose and its role within the system architecture.

Table 4.1 Software Tools

| Tool / Language               | Reason for Using This Program  |
|-------------------------------|--|
| python                        | Python is chosen for its versatility and extensive libraries in machine learning, image processing, and natural language processing (NLP). It provides robust support for developing and training OCR and text recognition models.   |
| VS Code (Visual Studio Code): | VS Code is a lightweight and powerful code editor with excellent support for Python and Dart. It offers extensions for Flutter development, making it ideal for writing and debugging Flutter apps efficiently.  |
| Text Recognition Libraries:   | These libraries (such as TensorFlow Lite, Tesseract-OCR, or specialized OCR libraries) are crucial for accurately extracting text from images. They provide pre-trained models and APIs for integrating OCR and handwritten text recognition capabilities into your app.   |
| Image Processing Libraries:   | Libraries like OpenCV or Dart/Flutter equivalents are used for preprocessing images before text recognition. They help enhance image quality, apply filters (like thresholding or blurring), and detect regions of interest, improving the accuracy of OCR results.  |
| Google Colab:                 | Google Colab is a cloud-based Jupyter notebook environment that provides free access to GPU resources.is commonly used by data scientists, researchers, and developers for building AI models and performing machine learning tasks. It is available for free and provides access to a high-performance GPU and TPU (Tensor Processing Unit). This allows users to leverage powerful hardware resources for training deep learning models and executing computationally intensive tasks It's used for training machine learning models, including OCR models, with TensorFlow or other |



|                    |   |
|--------------------|---|
|                    | frameworks. It offers collaborative development and easy integration with Google services for dataset management and model evaluation.  |
| Visual Studio Code | <b>Visual Studio Code (VS Code):</b> is an open-source source code editor developed by Microsoft. but it is an open-source project with contributions from a large community of developers worldwide. Visual Studio Code was initially released in April 2015. Since then, it has been continuously updated with new features, bug fixes, and performance improvements.VS Code is available for Windows, macOS, and Linux, making it a versatile choice for developers working on different operating systems |
| Flutter SDK        | Flutter SDK enables cross-platform mobile app development with a single codebase. It's chosen for its fast development cycles, expressive UI components, and performance on Android devices, crucial for building your Android app.   |
| Android Studio     | Android Studio provides a comprehensive IDE tailored for Android app development. It includes tools for Android SDK management, emulator setup, and debugging, essential for testing and deploying your Flutter-based Android app.  |

## 4.4 Hardware Specifications

This section provides an overview of the hardware specifications required for our proposed system. Each component is selected to ensure optimal performance and compatibility with the software tools. We will briefly describe the key specifications and their roles within the system.

*Table 4.2 the system's hardware requirements*

| Hardware            | Specifications  |
|---------------------|---|
| Development Machine | <p>Intel Core i5 or AMD Ryzen 5 processor or higher.</p> <p>RAM: 8 GB.</p> <p>Hard Disk 512GB SSD</p> |

|                 |   |
|-----------------|---|
|                 | Graphics Card:<br><br>dedicated GPU (NVIDIA GeForce GTX or RTX series,<br><br>AMD,Radeon RX series) |
| Testing Devices | Android Device (version 5 or above )  |

## 4.5 Experimental and Results

### 4.5.1 OCR Model

Today, we explore the groundbreaking realm of Optical Character Recognition (OCR), which serves as the bridge between the physical and digital worlds. OCR models convert printed text into editable and searchable digital content, streamlining data extraction and document processing. The tools and methods used to extract text play a crucial role in delivering a seamless user experience. Here, we'll summarize some of the most popular and effective methods used in text processing, focusing on key tools and their applications.

### 4.5.2 OCR Model Components

#### 4.5.2.1 OpenCV2

- **Description:** Open Source Computer Vision Library, versatile for real-time image processing and computer vision.
- **Usage:**
  - Loading and running models
  - Applying rotated bounding boxes and Non-Maximum Suppression (NMS)
  - Image preprocessing (thresholding, Gaussian Blur, Grayscale)
  - Drawing detected bounding boxes
- **Why Use OpenCV2:** It provides robust tools for image and video processing, supporting multiple programming languages and integration with machine learning frameworks like TensorFlow and PyTorch. This makes it ideal for tasks like character detection, text extraction, and document analysis.

#### 4.5.2.2 NumPy

- **Description:** Numerical Python, essential for efficient handling and manipulation of numerical data in image processing tasks.
- **Usage:**
  - Creating image arrays
  - Performing array operations to create input blobs for models
  - Image arrays preprocessing
- **Why Use NumPy:** Its array and matrix operations are critical for processing pixel data, enhancing the accuracy of OCR algorithms and facilitating seamless integration with image processing libraries.

#### 4.5.2.3 Pytesseract

- **Description:** A Python tool leveraging Google's Tesseract-OCR Engine for Optical Character Recognition.
- **Usage:**
  - Text extraction from images
- **Why Use Pytesseract:** It is open-source, supports multiple languages, has seamless Python integration, and is customizable, making it a versatile solution for extracting text from various sources.

#### 4.5.2.4 EAST Model

- **Description:** Efficient and Accurate Scene Text model designed for detecting text in natural scene images.
- **Usage:**
  - Geometry prediction (bounding box coordinates and rotation angle)
  - Score prediction (confidence score for text regions)
  - Non-Maximum Suppression (NMS) to remove redundant bounding boxes
- **Why Use EAST:**
  - **Efficiency:** Processes entire images in a single forward pass, enabling real-time text detection.
  - **Accuracy:** Maintains high accuracy in detecting text regions, even in complex scenarios.
  - **Rotation Handling:** Capable of handling rotated text, providing valuable geometry information for scene text recognition tasks.

### 4.6 Summary and Results

- By integrating these components, we developed an Android app that efficiently handles OCR for both printed and handwritten text. Our approach includes image preprocessing techniques like noise reduction and binarization to enhance image quality. Advanced text detection methods identify text regions within complex

images, and a combination of traditional OCR tools (like Pytesseract) and modern algorithms (like the EAST model) ensures effective recognition of both printed and handwritten text. The system also integrates multiple correction models to enhance accuracy by rectifying spelling errors and refining outputs.

## 4.7 Performance and Conclusion

- The resulting system demonstrates significant improvements in OCR accuracy and reliability. In our evaluations, we found that combining these methods reduced error rates and improved text recognition performance across various documents. This robust solution meets the increasing demand for efficient data handling across industries such as business, education, healthcare, legal, and financial sectors, where accurate text recognition and digitization are critical.
- In conclusion, by leveraging OpenCV2, NumPy, Pytesseract, and the EAST model, we created an Android app that provides an efficient and accurate OCR solution, showcasing the power of combining traditional and modern techniques to achieve superior performance in text recognition.

### 4.7.1 Handwritten Text Recognition model

### 4.7.2 OCR Methods

Optical Character Recognition (OCR) methods convert printed or handwritten text from images into machine-readable text. Here are key OCR methods:

#### 4.7.2.1 Pytesseract

- **Description:**
  - Python library providing an interface to Google's Tesseract-OCR.
- **Pros:**
  1. Capable of recognizing text from various textual images and scanned documents.
    2. Cross-platform compatibility.
    3. Supports multiple languages.
  4. Customizable output (e.g., plain text or HTML).

## Output Example

It has been left, however, to Mr. Goka, Ghana's Finance Minister, to do the honours as host, in which capacity he held a reception tonight in Accra's Ambassador Hotel. POLICE, on direct orders from the Cabinet, are openly intimidating members of Earl Russell's nuclear-disarming Committee of 100, the Committee claimed yesterday. It said pressure was being put on members and associates all over the country.

Figure 4:1 output example Pytesseract

## Ground truth

It has been left, however, to Mr. Goka, Ghana's Finance Minister, to do the honours as host, in which capacity he held a reception tonight in Accra's Ambassador Hotel. POLICE, on direct orders from the Cabinet, are openly intimidating members of Earl Russell's nuclear-disarming Committee of 100, the Committee claimed yesterday. It said pressure was being put on members and associates all over the country.

Figure 4:2 Ground truth

### ○ Accuracy Measures:

1. Character Error Rate (CER).
2. Word Error Rate (WER).
3. Similarity.

CER: 0.19012345679012346  
WER: 0.47761194029850745  
Similarity Ratio: 84

Figure 4:3 Accuracy Measures

### 4.7.2.2 Keras-OCR

- **Description:**
  - Open-source library providing a pretrained model for OCR tasks.
- **Pros:**
  1. Pretrained model.
  2. Supports various input formats (e.g., PNG, JPEG).
- **Cons:**

1. Output isn't sorted, complicating sentiment analysis.
  2. Requires tracking the line link between the image and the box around the text to determine word placement.
  3. Takes more time compared to other models.

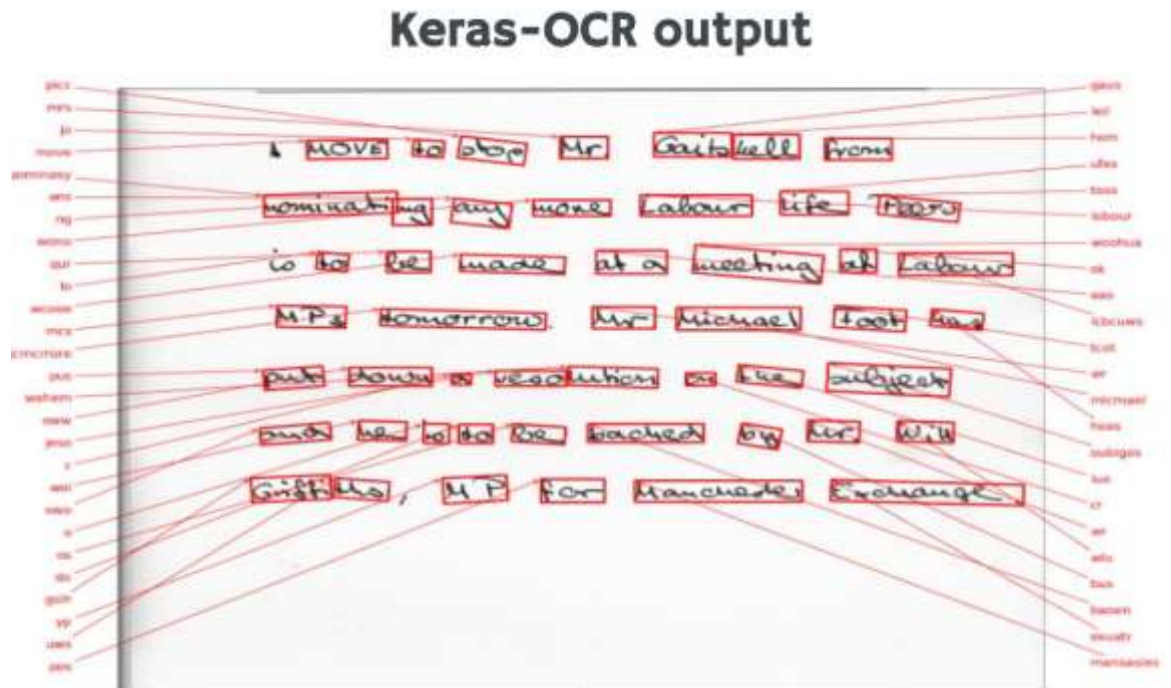


Figure 4:4 Keras-OCR output

#### 4.7.2.3 Easy-OCR

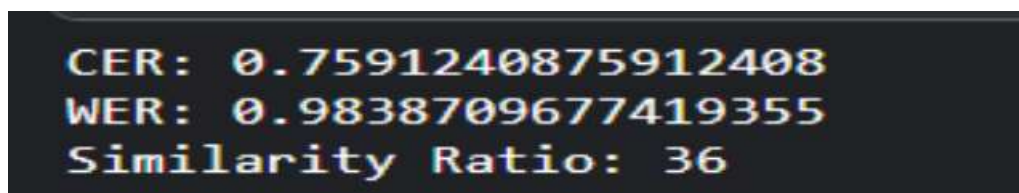
- **Pros:**
  1. Supports multiple languages.
  2. Cross-platform compatibility.
  3. Efficient performance.
- **Cons:**
  1. Less efficient than Pytesseract.
  2. Requires specifying the language beforehand.
  3. Accuracy output varies.

```
Text: It has been left , howeves , to H . Goka Ghana's Finance Hinusks , to do the honous as host, W wich camcty he held 0 rcece
phon tonight in Accra's AmbaSSador tokel POLICE , 0 dirct ordes from the (abinct, ac openly inhmiclating membecs 0f Eau ( Rus
sell & QUUc leaf - Commutke Of Joo te Commutec clamed yesksday # sd PrcSSue wQS being On membc ad aSSoCiales all OVC the disar
ming Put couhy
CER: 0.32839506172839505
WER: 0.7866666666666666
Similarity Ratio: 75
```

Figure 4:5 Easy-OCR output

#### 4.7.2.4 Merging Keras and Pytesseract

- **Purpose:**
  - Increase similarity and reduce character error rate (CER).
- **Method:**
  - Apply Keras first, then Pytesseract, and replace similar text from Pytesseract with Keras.
- **Accuracy:**
- **Why Pytesseract?**
  - The least CER and highest similarity ratio.
  - Sorted output similar to original text.
  - Supports multiple languages.
  - Customizable output (plain text or HTML).



*Figure 4:6 Merging Keras and Pytesseract Accuracy*

- **Why Pytesseract?**
  - The least CER and highest similarity ratio.
  - Sorted output similar to original text.
  - Supports multiple languages.
  - Customizable output (plain text or HTML).
  - More efficient and accurate.

#### 4.7.2.5 Summary

- Combining Keras-OCR and Pytesseract aims to leverage the strengths of both models. By using Keras for its pretrained capabilities and Pytesseract for its sorting and accuracy, this hybrid approach seeks to minimize error rates and improve output similarity to the original text.



## 4.7.3 Text Correction

### 4.7.3.1 Correction Models Evaluation

Evaluating text correction models involves comparing their performance in identifying and correcting errors in text. This process is critical to ensure the models' accuracy, efficiency, and applicability in real-world scenarios.

#### 4.7.3.1.1 TextBlob

is a powerful tool for text processing, particularly for tasks like spell checking and correction. However, like any tool, it has its limitations. One of these limitations is its ability to handle context-based errors, TextBlob's spell checking and correction capabilities are primarily based on dictionary lookups and language specific rules. It may struggle with context-based errors because it does not have a deep understanding of the context in which the text is used, and we can see the performance of that library with different test data, know that the error rate refer to the number of wrong words per document so the less the error rate the more good the method.

To measure the error rate by comparing the number of incorrect words to the total number of words in the text, you can define a metric called **Word Error Rate (WER)**. In this context, WER is slightly different from the traditional usage, focusing on the proportion of incorrect words to the total words. The formula for this metric can be defined as:

**WER = Number of Incorrect Words/Total Number of Words**

Test Data 1 (printed):

Before: WER =  $(280/1200) \times 100 = 17.9\%$  After: WER =  $(165/1000) \times 100 = 16.5\%$

Test Data 2 (handwritten):

Before: WER =  $(179/1000) \times 100 = 23.34\%$  After: WER =  $(296/1200) \times 100 = 22.4\%$

Test Data 3:

Before: WER =  $(82/800) \times 100 = 10.26\%$  After: WER =  $(74/800) \times 100 = 9.3\%$

|                       | Test_data 1(printed) | Test_data 2(handwritten) | Tets_data 3 |
|-----------------------|----------------------|--------------------------|-------------|
| AvgErrorrate (before) | 17,9 %               | 23.34 %                  | 10.26 %     |
| AvgError rate (after) | 16.5 %               | 22.45 %                  | 9.3 %       |



Table 4.1 Results 1 for AvgErrorrate before and afterTextBlob

And now we will see the performance on different documents that contain different types of error text, individual errors words, interconnected words, missing spaces text and contextual errors words

|                    | Doc_1(individual) | Doc_2(interconnect<br>ed) | Doc_3 (contextual) |
|--------------------|-------------------|---------------------------|--------------------|
| Error rate(before) | 30 %              | 38.45%                    | 25 %               |
| Error rate (after) | 8 %               | 42.43 %                   | 23.65%             |

Table 4.4 Results 2 for Error rate before and after TextBlob

TextBlob is effective for correcting individual word errors, significantly reducing the error rate. However, it performs poorly with interconnected and contextual word errors. This limitation is due to its reliance on dictionary lookups and language-specific rules, which do not account for the context in which words are used. As a result, while TextBlob is a valuable tool for basic spell checking and correction, it is less effective for more complex text correction tasks that require contextual understanding.

#### 4.7.3.1.2 PySpellChecker

SpellChecker class from the pyspellchecker library uses a simple spelling correction algorithm based on word frequencies in the English language. It may not always produce perfect results, especially for words that are not in the English dictionary or for words that are spelled correctly but used incorrectly in the context of the text.

Both PySpellChecker and TextBlob are useful tools for spell correction, and they often provide similar outputs for misspelled words. However, both PySpellChecker and TextBlob are useful for spell correction and often provide similar outputs for misspelled words, they may struggle with context-based errors and missing spaces. And they nearly give the same performance so we can see the tables are nearly identical, know that the error rate refer to the number of wrong words per document so the less the error rate the more good the method.

Test Data 1 (printed):

Before: WER =  $(280/1200) \times 100 = 17.9\%$  After: WER =  $(165/1000) \times 100 = 16.5\%$

Test Data 2 (handwritten):

Before: WER =  $(179/1000) \times 100 = 23.34\%$  After: WER =  $(296/1200) \times 100 = 22.4\%$

Test Data 3:

Before: WER =  $(82/800) \times 100 = 10.26\%$  After: WER =  $(74/800) \times 100 = 9.3\%$

|                       | Test_data 1(printed) | Test_data 2(handwritten) | Tets_data 3 |
|-----------------------|----------------------|--------------------------|-------------|
| AvgErrorrate (before) | 17,9 %               | 23.34 %                  | 10.26 %     |
| AvgError rate (after) | 16.5 %               | 22.45 %                  | 9.3 %       |

Table 4.5 Results 1 for AvgErrorrate before and after PySpellChecker

And the docs table we will see the performance on different documents that contain different types of error text, individual errors words, interconnected words, missing spaces text and contextual errors word

|                    | Doc_1(individual) | Doc_2(interconnect ed) | Doc_3 (contextual) |
|--------------------|-------------------|------------------------|--------------------|
| Error rate(before) | 30 %              | 38.45%                 | 25 %               |
| Error rate (after) | 6 %               | 42,43 %                | 23.65%             |

Table 4.6 Results 2 for Error rate before and after PySpellChecker

#### 4.7.3.1.3 Symspell

SymSpell is a Python library for spell checking and correction. It uses the Symmetric Delete spelling correction algorithm, which is highly efficient and accurate. The library provides a SymSpell class with methods for looking up words and compound words in the dictionary and getting suggestions for correcting their spelling. SymSpell uses the LookupCompound function to perform compound-aware automatic spelling correction of multi-word input strings. This function is designed to handle various types of errors that can occur

in compound words, including splitting errors, concatenation errors, substitution errors, transposition errors, deletion errors, and insertion errors. and we can see the performance of symspell with different test data, know that the error rate refere to the number of wrong words per document so the less the error rate the better the method.

Test Data 1 (printed):

Test Data 2 (handwritten):

Before: WER =  $(280/1000) \times 100 = 17.9\%$  After: WER =  $(134/1000) \times 100 = 13.4\%$

Before: WER =  $(179/1200) \times 100 = 23.34\%$  After: WER =  $(156/1200) \times 100 = 13.0\%$

Test Data 3:

Before: WER =  $(82/800) \times 100 = 10.26\%$  After: WER =  $(46/800) \times 100 = 9.3\%$

|                       | Test_data 1(printed) | Test_data 2(handwritten) | Test_data 3 |
|-----------------------|----------------------|--------------------------|-------------|
| AvgErrorrate (before) | 17,9 %               | 23.34 %                  | 10.26 %     |
| AvgError rate (after) | 13.4 %               | 13.0 %                   | 5.75 %      |

Table 4.7 Results 1 for AvgErrorrate before and after Symspell

And the docs table we will see the performance on different documents that contain different types of error text, individual errors words, interconnected words, missing spaces text and contextual errors words.

|                    | Doc_1(individual) | Doc_2(interconnected) | Doc_3 (contextual) |
|--------------------|-------------------|-----------------------|--------------------|
| Error rate(before) | 30 %              | 38.45%                | 25 %               |
| Error rate (after) | 10 %              | 10.16 %               | 20.74%             |

Table 4.8 Results 2 for Errorrate before and after Symspell

SymSpell is a robust spelling correction algorithm renowned for its proficiency in correcting both individual words and entire sentences, especially adept at handling compound words and detecting spacing errors. However, for context-dependent corrections, JamSpell may offer advantages due to its utilization of language models to gauge word correctness within context, enabling more contextually relevant suggestions. JamSpell further leverages phonetic similarity and contextual cues for improved correction accuracy. While SymSpell excels in general word and sentence corrections, JamSpell's contextual approach may be

preferable for tasks heavily reliant on context. Ultimately, the choice between them hinges on the specific needs of the text processing task.

#### 4.7.3.1.4 JamSpell

is a Python library for spell checking and correction, built on the widely used Hunspell library. While primarily statistical in nature, it leverages various techniques to suggest contextually appropriate corrections. Firstly, it utilizes a language model trained on a large corpus of text to calculate the probability of a word being correct in a given context. Additionally, it employs the Levenshtein distance to measure similarity between words, enabling it to suggest corrections even for non-exact matches. Furthermore, JamSpell can utilize phonetic similarity to suggest corrections based on sound rather than spelling. Although lacking deep contextual understanding, it can use surrounding words in a sentence to make more informed corrections. These capabilities collectively enhance its spellchecking accuracy, despite its reliance on statistical probabilities and patterns in language.

and we can see the performance of sjamspell with different test data, know that the error rate refere to the number of wrong words per document so the less the error rate the better the method.

Test Data 1 (printed):

Before: WER =  $(280/1000) \times 100 = 17.9\%$  After: WER =  $(144/1000) \times 100 = 16.5\%$

Test Data 2 (handwritten):

Before: WER =  $(179/1200) \times 100 = 23.34\%$  After: WER =  $(100/1200) \times 100 = 8.3\%$

Test Data 3:

Before: WER =  $(82/800) \times 100 = 10.26\%$  After: WER =  $(30/800) \times 100 = 9.3\%$

|                       | Test_data 1(printed) | Test_data 2(handwritten) | Test_data 3 |
|-----------------------|----------------------|--------------------------|-------------|
| AvgErrorrate (before) | 17,9 %               | 23.34 %                  | 10.26 %     |
| AvgError rate (after) | 14.4 %               | 8.3 %                    | 3.75 %      |

Table 4.9 Results 1 for AvgErrorrate before and after JamSpell

And the docs table we will see the performance on different documents that contain different types of error text, individual errors words, interconnected words, missing spaces text and contextual errors words.

|                    | Doc_1(individual) | Doc_2(interconnect<br>ed) | Doc_3 (contextual) |
|--------------------|-------------------|---------------------------|--------------------|
| Error rate(before) | 30 %              | 38.45%                    | 25 %               |
| Error rate (after) | 17%               | 30.16 %                   | 12.74%             |

*Table 4.10 Results 2 for Error rate before and after JamSpell*

JamSpell demonstrates a stronger ability to make corrections in sentences that rely on context compared to Spello. However, both models still face challenges when dealing with complex sentences or text that contains multiple errors. While JamSpell's language model, edit distance, phonetic similarity, and ability to use contextual information provide it with an advantage in making contextually appropriate corrections, it is important to note that no model is perfect. Both Spello and JamSpell have their limitations and may not always produce the desired output, especially when dealing with complex sentences or text that contains multiple errors.

In summary, SymSpell is a robust spelling correction algorithm renowned for its proficiency in correcting both individual words and entire sentences, especially adept at handling compound words and detecting spacing errors. However, for context-dependent corrections, JamSpell may offer advantages due to its utilization of language models to gauge word correctness within context, enabling more contextually relevant suggestions. JamSpell further leverages phonetic similarity and contextual cues for improved correction accuracy. While SymSpell excels in general word and sentence corrections, JamSpell's contextual approach may be preferable for tasks heavily reliant on context. Ultimately, the choice between them hinges on the specific needs of the text processing task. So we decide to use a combination of these methods to get best performance

#### **4.7.3.1.5 The combination of SymSpell, JamSpell, and Pyspellchecker**

The combination of SymSpell, JamSpell, and Pyspellchecker can provide a comprehensive approach to text correction, leveraging the strengths of each technique to address different types of errors. Here is how the combination could work:

1) SymSpell: Use SymSpell first to address errors where a space has been mistakenly inserted within a correct word, leading to two incorrect terms, or where a space has been mistakenly omitted between two correct words, leading to one incorrect combined term. SymSpell's compound-aware automatic spelling correction capabilities make it well-suited for this task.

2) JamSpell: Next, use JamSpell to address context-based errors. JamSpell's language model, edit distance, phonetic similarity, and ability to use contextual information can help make more contextually appropriate corrections.

3) Pyspellchecker: Finally, use Pyspellchecker to ensure the individual words are correct. Pyspellchecker's spell-checking capabilities can help catch any remaining spelling errors that SymSpell and JamSpell may have missed.

By combining these techniques, you can create a more robust and accurate text correction system that can handle a wide range of errors, from simple spelling mistakes to more complex context-based errors.

So, we can see the results on our test data after the combination

Test Data 1 (printed):

Before: WER =  $(280/1200) \times 100 = 17.9\%$  After: WER =  $(44/1000) \times 100 = 16.5\%$

Test Data 2 (handwritten):

Before: WER =  $(179/1000) \times 100 = 23.34\%$  After: WER =  $(91/1200) \times 100 = 22.4\%$

Test Data 3:

Before: WER =  $(82/800) \times 100 = 10.26\%$  After: WER =  $(22/800) \times 100 = 9.3\%$

|                       | Test_data 1(printed) | Test_data 2(handwritten) | Test_data 3 |
|-----------------------|----------------------|--------------------------|-------------|
| AvgErrorrate (before) | 17,9 %               | 23.34 %                  | 10.26 %     |
| AvgError rate (after) | 4.4 %                | 7.5 %                    | 2.75 %      |

Table 4.11 Results for AvgErrorrate before and after combination

|                    | Doc_1(individual) | Doc_2(interconnected) | Doc_3 (contextual) |
|--------------------|-------------------|-----------------------|--------------------|
| Error rate(before) | 30 %              | 38.45%                | 25 %               |
| Error rate (after) | 6%                | 7.16 %                | 8.74%              |

#### 4.7.4 Conclusion

After integrating the various methods and techniques discussed, we successfully created a mobile application capable of performing robust OCR and handwritten text recognition. By leveraging a combination of preprocessing steps, advanced OCR models, and effective text correction methods, our application delivers accurate text extraction and correction for both printed and handwritten documents.

Key achievements include:

- **Preprocessing:** Improved image quality through techniques like noise reduction, binarization, and grayscale conversion.
- **Text Detection:** Efficiently identified text regions using the EAST model and applied appropriate bounding boxes.
- **OCR Models:** Utilized Pytesseract for printed text and a hybrid approach for handwritten text using Keras-OCR and Easy-OCR, achieving high accuracy.
- **Text Correction:** Implemented multiple text correction libraries such as TextBlob, PySpellChecker, and JamSpell, enhancing the accuracy of the extracted text.

Through the integration of these advanced methods, we have developed a functional and user-friendly mobile application that meets the needs of users requiring reliable OCR and handwritten text recognition. This app is built using Flutter for cross-platform compatibility and ensures a seamless user experience on Android devices (version 5.0 and above).

Overall, our project demonstrates the effectiveness of combining different technologies and methodologies to create a comprehensive solution for text recognition and correction, paving the way for future enhancements and applications.

## **Chapter 5 : Run the Application**

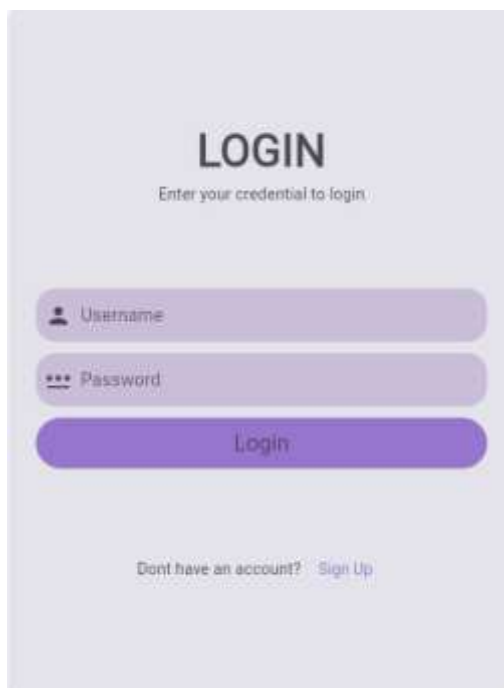


## 5.1 Introduction

This chapter presents a detailed, step-by-step tutorial on how to execute the Mobile application. The following instructions will guide you through the process.

## 5.2 Run the Application

- 1) After running the application, it will redirect you to the login page, you can Sign Up or Log In



*Figure 5:1 After running the application, it will redirect you to the login page.*

- 2) **Sign Up:** If you don't have an account, sign up using your email address..

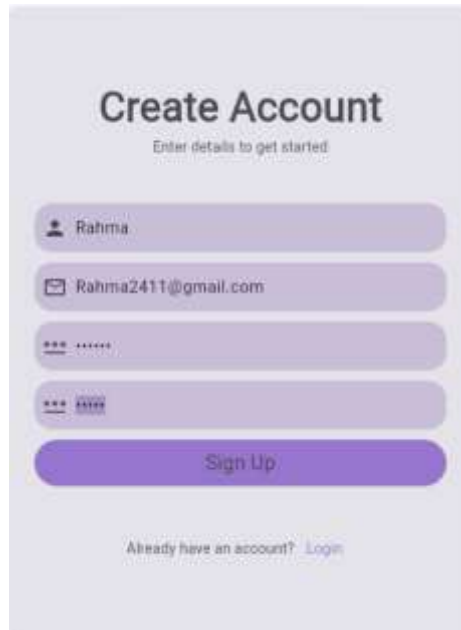
A screenshot of a 'Create Account' form. The title 'Create Account' is at the top, followed by the subtitle 'Enter details to get started'. There are four input fields: a name field with 'Rahma', an email field with 'Rahma2411@gmail.com', and two password fields, each with a strength indicator (three dots and a bar). A purple 'Sign Up' button is below the password fields. At the bottom, there is a link: 'Already have an account? [Login](#)'.

Figure 5:2 Sign up

- 3) **Log In:** If you already have an account, log in using the same email address that you used to sign up.

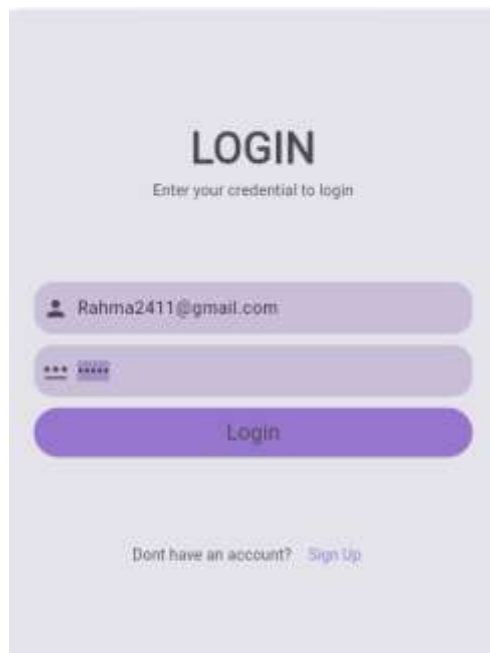
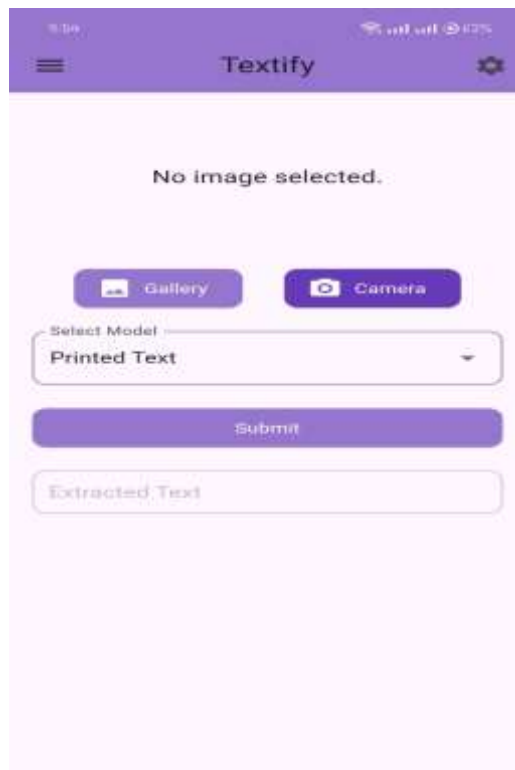
A screenshot of a 'LOGIN' form. The title 'LOGIN' is at the top, followed by the subtitle 'Enter your credential to login'. There are two input fields: an email field with 'Rahma2411@gmail.com' and a password field with a strength indicator (three dots and a bar). A purple 'Login' button is below the password field. At the bottom, there is a link: 'Dont have an account? [Sign Up](#)'.

Figure 5:3 login in

#### 4) Navigate to the Home Page



*Figure 5:4 The home page*

##### 5) Capture or Upload a Picture:

- You can either:
  - **Take a Picture:** Use your device's camera to capture a new image, and, crop the text part



Figure 5:5 take a picture

- **Upload from Gallery:** Select an existing image from your device's gallery.

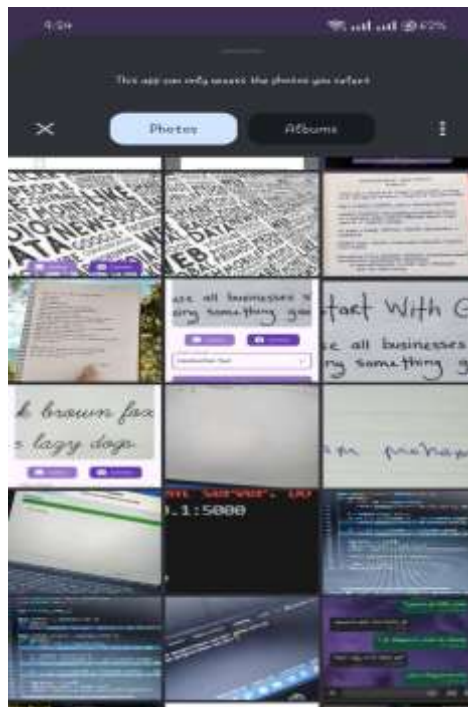


Figure 5:6 upload a picture

6) Select the OCR Model:

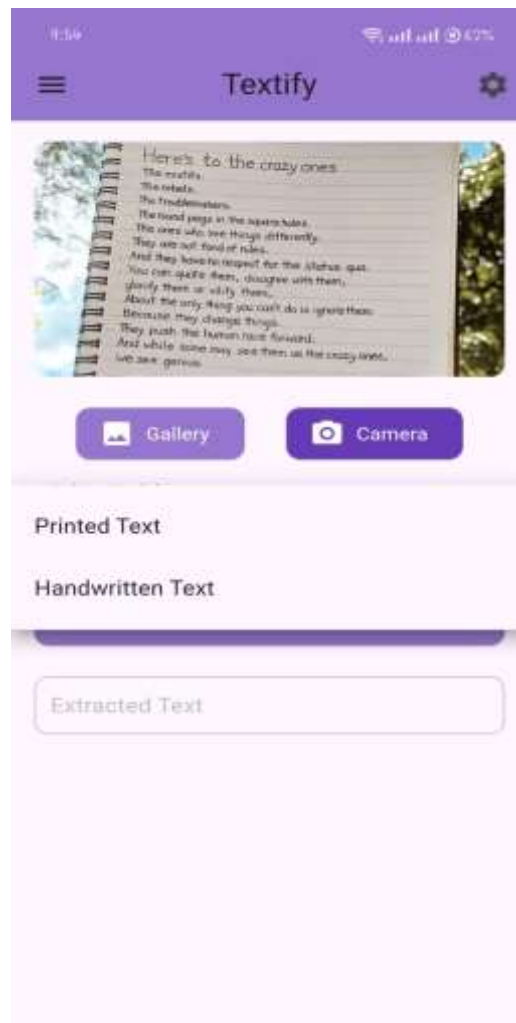


Figure 5:7 select model

Choose which model to use for text extraction:

- **Printed Text Model**

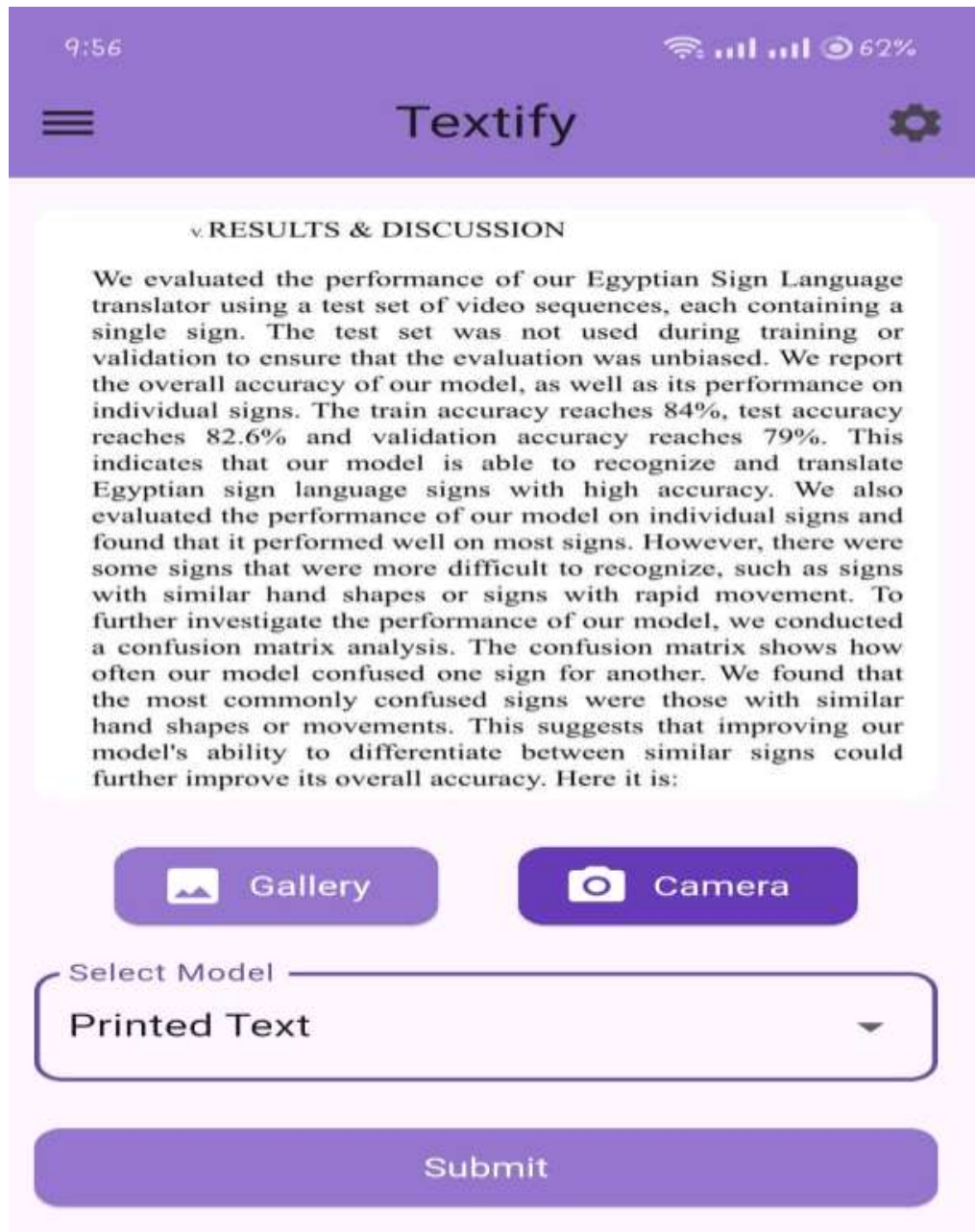


Figure 5:8 select printed model

- **Handwritten Text Model**

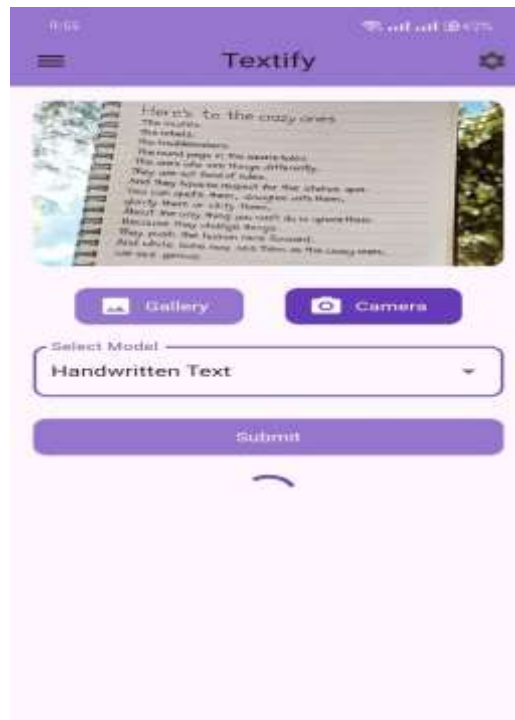


Figure 5:9 select handwritten model

#### 7) Extract and Correct Text:

- The app will process the image and extract the text using advanced models.
- After extraction, the app will apply corrections to ensure the text is accurate and clear.



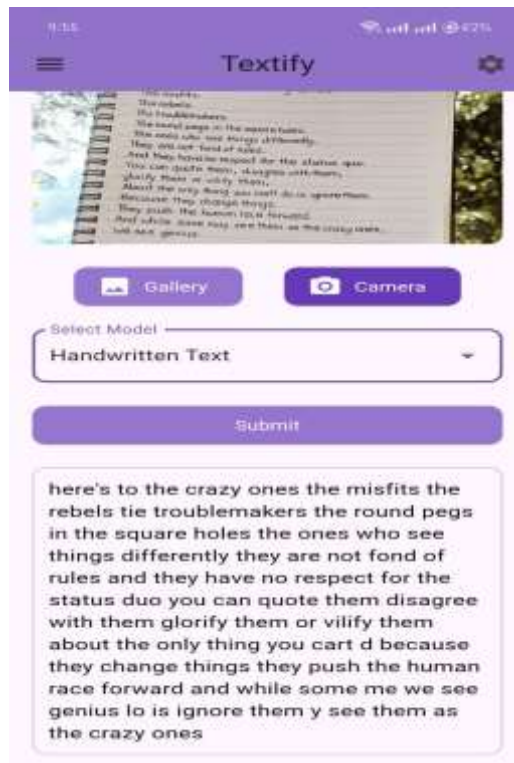


Figure 5:10 text extraction

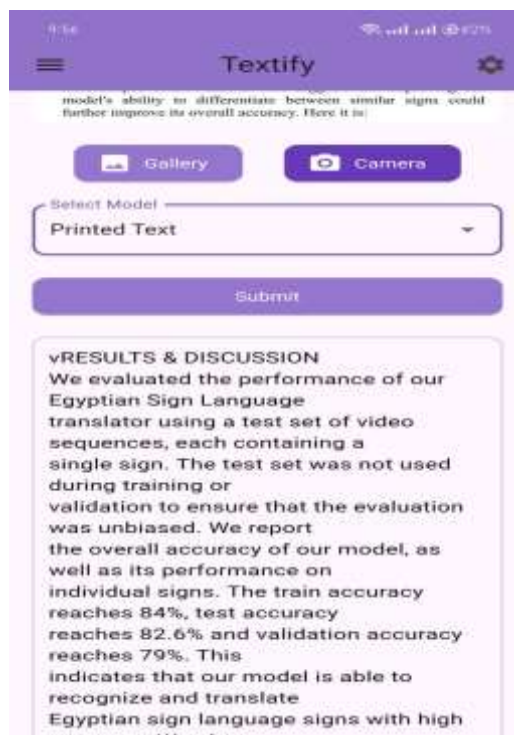





Figure 5:11text extraction\_2

Figure 5:12 Trying Printed Text Model

InstaText



 Gallery

 Camera

Select Model

Printed Text

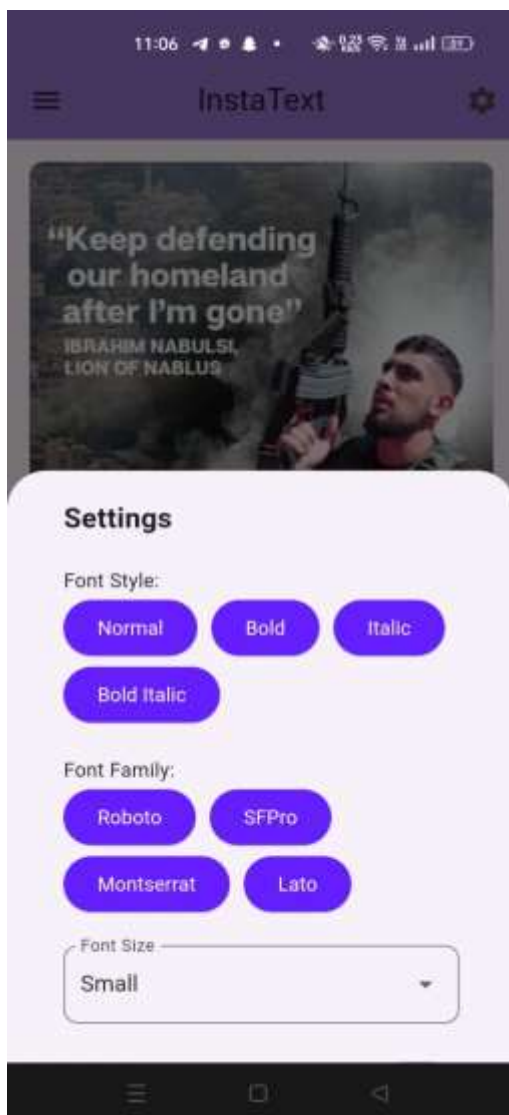
Submit

Keep defending  
our homeland  
after Im gone"  
IBRAHIM NABULSI,  
LION OF NABLUS

- Bold

Figure 5: 13From setting  
change the font style

**Keep defending  
our homeland  
after Im gone"  
IBRAHIM NABULSI,  
LION OF NABLUS**



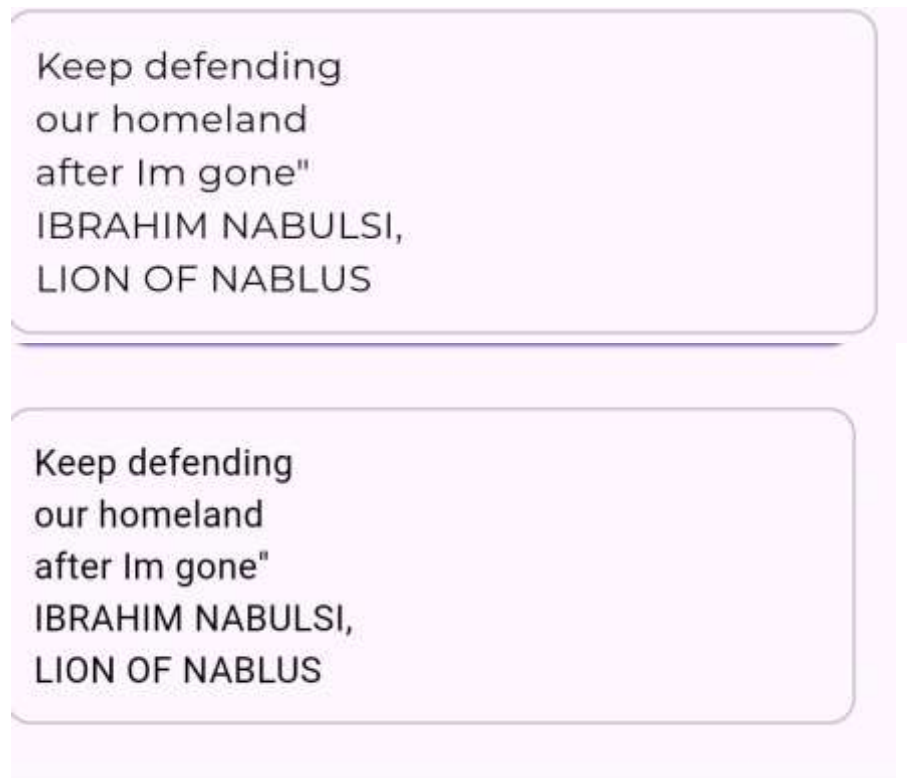


Figure 5: 14From setting change the font family

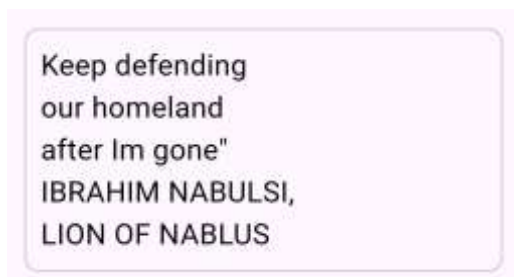
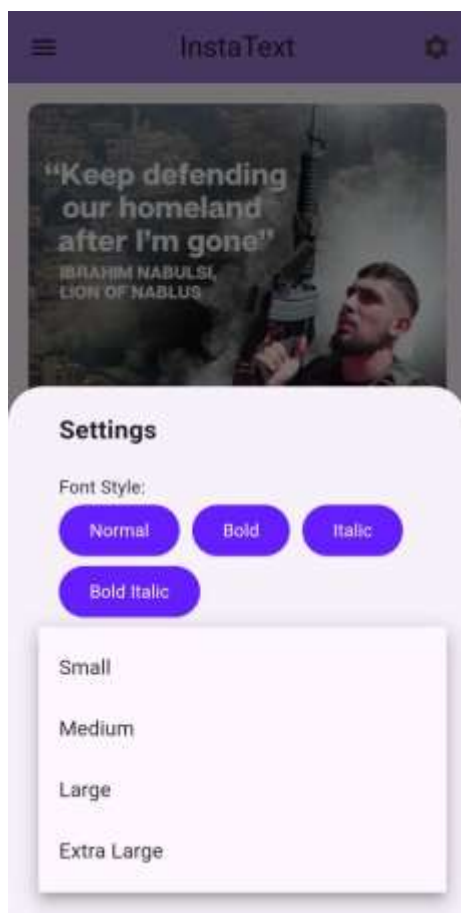


Figure 5: 15from setting change the font size

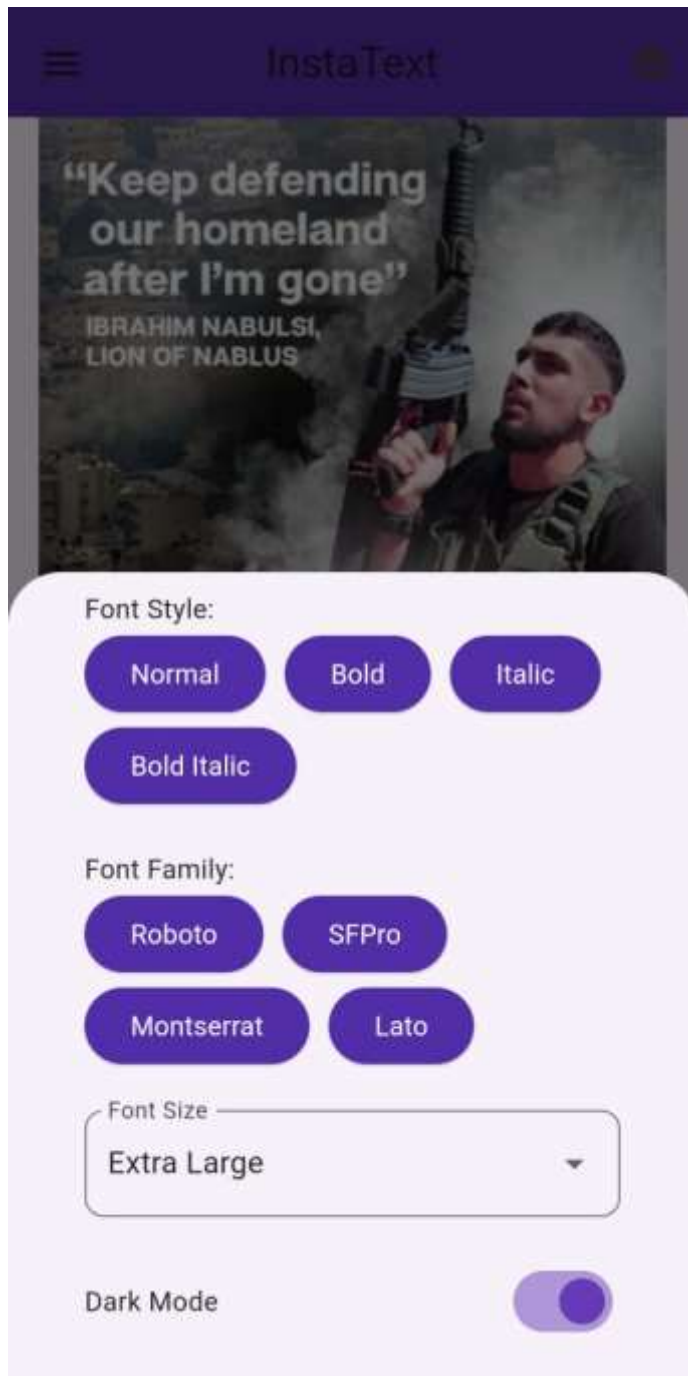


Figure 5: 16 from setting switch to the dark mode

## **Chapter 6 : Conclusion and Future Work**

## 6.1 Conclusion:

the development of advanced OCR (Optical Character Recognition) and handwritten text recognition systems represents a transformative leap in the digital age, impacting multiple facets of society. These systems not only bridge the gap for individuals with visual impairments or language barriers by converting printed and handwritten content into accessible digital formats but also streamline various operational processes across industries. The automation of document management, for instance, results in significant time savings and error reduction, which enhances overall efficiency.

Moreover, the cultural and historical significance of OCR technology cannot be overstated. By digitizing manuscripts and handwritten records, we are preserving invaluable pieces of our heritage, ensuring they are protected from the ravages of time and made available for future generations to study and appreciate. This digital preservation facilitates broader access to historical documents and fosters a deeper understanding of our past.

In the realm of machine learning and artificial intelligence, OCR technology is indispensable. It provides large, diverse datasets necessary for training sophisticated algorithms, thereby improving the accuracy and functionality of AI systems. This, in turn, spurs innovation across various fields, including linguistics, literature, and data science, by enabling more refined and insightful analyses.

For businesses, the implications of OCR technology are profound. Enhanced operational efficiency through faster document processing translates to better customer service and more streamlined workflows. This technological boost is crucial for staying competitive in today's fast-paced market. Similarly, in education, OCR systems democratize access to knowledge by digitizing textbooks and other educational materials, thus supporting students with disabilities and promoting inclusive learning environments.

Ultimately, advanced OCR and handwritten text recognition systems are pivotal in creating a more connected, efficient, and inclusive world. They enhance accessibility, preserve cultural heritage, foster technological and scientific innovation, and support both educational and business needs. As these technologies continue to evolve, their potential to further transform and enrich our society is boundless, promising a future where information is universally accessible and operational processes are increasingly optimized.

## 6.2 Future Work

The current OCR and handwritten text recognition systems have several areas for potential improvement:

1. **Expanding Detection to Include Young Adults:** Enhance the system to identify signs of potential violence among teenagers and young adults, broadening the focus beyond just children.
2. **Detecting Risks of Self-Harm:** Improve the system's ability to recognize signs of self-harm, enabling earlier and more effective support for individuals at risk.
3. **Enhancing Child Interaction Through Indirect Questioning:** Use indirect, child-friendly questions to improve engagement and comfort, facilitating more accurate and comprehensive information gathering.

Addressing these areas will make OCR systems more robust and versatile, ultimately contributing to a safer and more supportive environment for all individuals.



## References

- [1] Christian Bartz, Hendrik Rätz, and Christoph Meinel, "Handwriting Classification for the Analysis of Art-Historical Documents," Hasso Plattner Institute, University of Potsdam, 14482 Potsdam, Germany, Nov. 4, 2020.
- [2] Yuning Du, Chenxia Li, Ruoyu Guo, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Zilin Yu, Yehua Yang, Qingqing Dang, and Haoshuang Wang, "PP-OCR: A Practical Ultra Lightweight OCR System," Baidu Inc, Oct. 5, 2020.
- [3] Chengwei Zhang, Yunlu Xu, Zhazhan Cheng, Shiliang Pu, Yi Niu, Fei Wu, and Futai Zou, "SPIN: Structure-Preserving Inner Offset Network for Scene Text Recognition," 1Shanghai Jiaotong University, China; 2Hikvision Research Institute, China; 3Zhejiang University, China, Oct. 25, 2021.
- [4] unlong Li, Yiheng Xu, and Tengchao Lv, "DiT: Self-supervised Pre-training for Document Image Transformer," Shanghai Jiao Tong University, Shanghai, China, and Microsoft Research Asia, Beijing, China, July 19, 2022.
- [5] Atman Mishra, A. Sharath Ram, and Kavyashree C., "Handwritten Text Recognition Using Convolutional Neural Network," Dept. of AIML, New Horizon College of Engineering, Bangalore, India, Jul. 1, 2023.
- [6] Tianlun Zheng, Zhineng Chen, Bingchen Huang, Wei Zhang, and Yu-Gang Jiang, "MRN: Multiplexed Routing Network for Incremental Multilingual Text Recognition," School of Computer Science, Fudan University, China, and Shanghai Collaborative Innovation Center of Intelligence, July 30, 2023
- [7] Y. Bengio, P. Y. Simard, and P. Frasconi. Learning longterm dependencies with gradient descent is difficult. *NN*, 5(2):157–166, 1994. 3
- [8] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, R. Young, K. Ashida, H. Nagai, M. Okamoto, H. Yamamoto, H. Miyao, J. Zhu, W. Ou, C. Wolf, J. Jolion, L. Todoran, M. Worring, and X. Lin. ICDAR 2003 robust reading competitions: entries, results, and future directions. *IJDAR*, 7(2- 3):105–122, 2005
- [10] A. Graves, S. Fernandez, F. J. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML*, 2006.

- [11] Baek, Y.; Lee, B.; Han, D.; Yun, S.; and Lee, H. 2019. Character region awareness for text detection. In Proc. CVPR, 9365–9374. Chng, C. K., and Chan, C S. 2017. Total-text: A comprehensive dataset for scene text detection and recognition. In Proc. ICDAR, 935–942.
- [12] Lyu, P.; Yao, C.; Wu, W.; Yan, S.; and Bai, X. 2018b. Multioriented scene text detection via corner localization and region segmentation. In Proc. CVPR, 7553–7563.
- [13] Xu, Y.; Wang, Y.; Zhou, W.; Wang, Y.; Yang, Z.; and Bai, X. 2019. Textfield: Learning a deep direction field for irregular scene text detection. IEEE Trans. Image Processing 28(11):5566–5579
- [14] Karatzas, D.; Gomez-Bigorda, L.; Nicolaou, A.; Ghosh, S. K.; Bagdanov, A. D.; Iwamura, M.; Matas, J.; Neumann, L.; Chandrasekhar, V. R.; Lu, S.; Shafait, F.; Uchida, S.; and Valveny, E. 2015. ICDAR 2015 competition on robust reading. In Proc. ICDAR.
- [15] Xie, E.; Zang, Y.; Shao, S.; Yu, G.; Yao, C.; and Li, G. 2019a. Scene text detection with supervised pyramid context network. In Proc. AAAI, volume 33, 9038–9045.