**Advanced Ocr and Handwriting System with Correction**

|  |
| --- |
| Riham Mohammed Hassan, Rahma Ali Ali Ali, Mohammed Essam Mohammed, Omar Sherif Soliman, Omar Sherif Soliman,Moamen Said Mohammed, Dr. Ahmed Ezzat |
| *Ain Shams University Faculty of Computer & Information Sciences Information Systems Department* |
|  |
|  |

***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.***

# Introduction

Optica 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.

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.

## 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

So, we develop 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.

# Literature Review

This part 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.

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.

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. 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).

In this section, we compare systems that are similar to our project and describe 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.

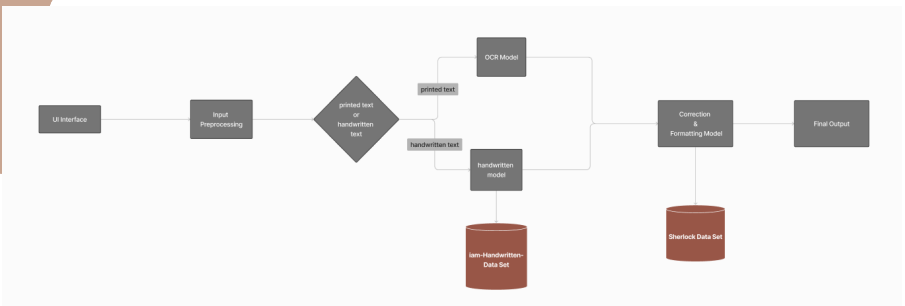
# Methodology

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 inforamation and Uploaded Images.

A screenshot of a computer

Description automatically generated  
 Fig .1: System Architecture

## **Methods and Procedures Used:**

*Fig .2: System Architecture (models and Procedures)*

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 these main steps: data input, preprocessing, and OCR model implementation, handwritten recognition,textcorrection. Below,integration and evaluation, 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 and Handwriting Model Implementation:**

1. Ocr model:

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.

2. Handwritten Text Recognition Models:

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).

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.

Easy-OCR:

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

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

# Results

### **OCR Model:**

### 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.

### - OCR Model Components;

### OpenCV2:

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.

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.

1. .NumPy;

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.**

1. Pytesseract;

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.

1. .EAST Model;

Efficient and Accurate Scene Text model designed for detecting text in natural scene images.

**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.

Summery and Result:

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.

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.

### **Handwritten Text Recognition model:**

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

1.Pytesseract:

Python library providing an interface to Google's Tesseract-OCR.Capable of recognizing text from various textual images and scanned documents.

Cross-platform compatibility.

Supports multiple languages.

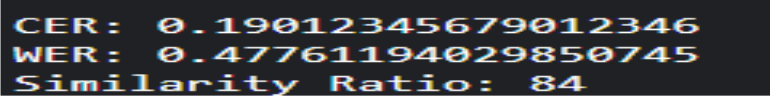
Customizable output (e.g., plain text or HTML).

Accuracy Measures:

Character Error Rate (CER).

Word Error Rate (WER).

Similarity.

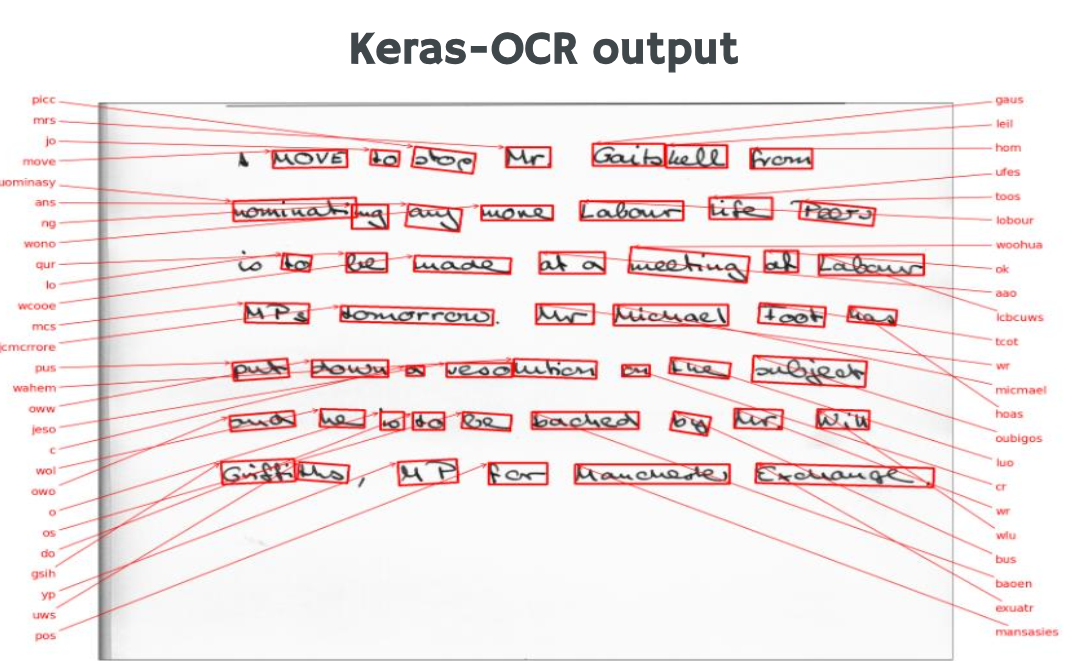
  
*Fig .3: Pytesseract Accuracy measurement*

2.Keras-OCR;

Open-source library providing a pretrained model for OCR tasks.

**Pros:**Pretrained model.Supports various input formats (e.g., PNG, JPEG)

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

Fig .4: Keras-OCR Output

3. Easy-OCR;

**Pros:Supports multiple languages.**

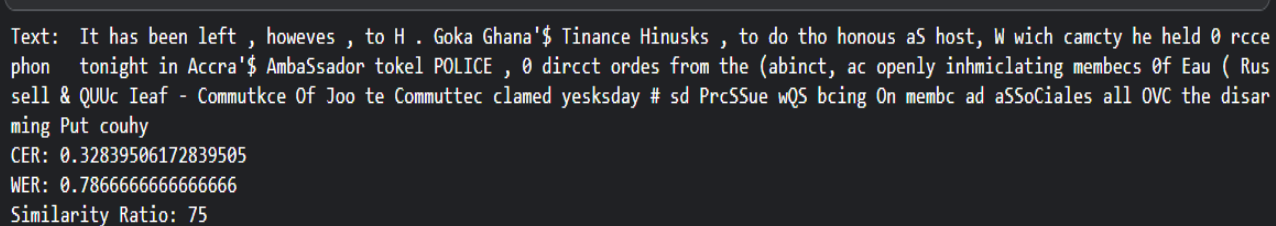
Cross-platform compatibility.

Efficient performance.

**Cons:Less efficient than Pytesseract.**

Requires specifying the language beforehand.

**Accuracy output varies.**

*Fig .5: Easy-OCR Output*

**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.**

* *****Text Correction:*****

**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.**

****1. PySpellChecker and TextBlob:****

**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. 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****

****-**how to measure**

Test Data 1 (printed):

Before: WER = (280/1200)x100=17.9%

After: WER = (165/1000)x100=16.5%

Test Data 2 (handwritten):

Before: WER = (179/1000)x100=23.34%

After: WER = (296/1200)x100=22.4%

Test Data 3:

**Before: WER = (82/800)x100=10.26%**

**After: WER = (74/800)x100=9.3%**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test\_data 1(printed) | Test\_data 2(handwriten) | Tets\_data 3 |
| AvgErrorrate (before) | 17,9 % | 23.34 % | 10.26 % |
| AvgError rate (after) | 16.5 % | 22.45 % | 9.3 % |

*Table ‎1 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(interconnected) | Doc\_3 (contextual) |
| Error rate(before) | 30 % | 38.45% | 25 % |
| Error rate (after) | 6 % | 42,43 % | 23.65% |

*Table ‎2 Results 2 for Error rate before and after PySpellChecker*

**2. 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.

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.

TestData1(printed):

Before:WER = (280/1000)x100=17.9%

After: WER = (134/1000)x100=13.4%

Test Data 2 (handwritten):

Before: WER = (179/1200)x100=23.34%

After: WER = (156/1200)x100=13.0%

Test Data 3:

Before: WER = (82/800)x100=10.26%

After: WER = (46/800)x100=9.3%

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test\_data 1(printed) | Test\_data 2(handwriten) | Test\_data 3 |
| AvgErrorrate (before) | 17,9 % | 23.34 % | 10.26 % |
| AvgError rate (after) | 13.4 % | 13.0 % | 5.75 % |

*Table ‎3. 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. Results 2 for Errorrate before and after Symspell*

**3. 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. JamSpell has their limitations and may not always produce the desired output, especially when dealing with complex sentences or text that contains multiple errors. and we can see the performance of jamspell 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)x100=17.9%

After: WER = (144/1000)x100=16.5%

Test Data 2 (handwritten):

Before: WER = (179/1200)x100=23.34%

After: WER = (100/1200)x100=8.3%

Test Data 3:

Before: WER = (82/800)x100=10.26%

After: WER = (30/800)x100=9.3%

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test\_data 1(printed) | Test\_data 2(handwriten) | Test\_data 3 |
| AvgErrorrate (before) | 17,9 % | 23.34 % | 10.26 % |
| AvgError rate (after) | 14.4 % | 8.3 % | 3.75 % |

Table ‎5 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(interconnected) | Doc\_3 (contextual) |
| Error rate(before) | 30 % | 38.45% | 25 % |
| Error rate (after) | 17% | 30.16 % | 12.74% |

Table ‎6 Results 2 for Error rate before and after JamSpell

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. 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.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)x100=17.9%

After: WER = (44/1000)x100=16.5%

Test Data 2 (handwritten):

Before: WER = (179/1000)x100=23.34%

After: WER = (91/1200)x100=22.4%

Test Data 3:

Before: WER = (82/800)x100=10.26%

After: WER = (22/800)x100=9.3%

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test\_data 1(printed) | Test\_data 2(handwriten) | Test\_data 3 |
| AvgErrorrate (before) | 17,9 % | 23.34 % | 10.26 % |
| AvgError rate (after) | 4.4 % | 7.5 % | 2.75 % |

Table ‎7. Results for AvgErrorrate before and after combination

And in 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) | 6% | 7.16 % | 8.74% |

Table ‎8. Results for Error rate before and after combination

We can observe an improvement after combining the three methods, as evidenced by the data in the tables

# Conclusion and Discussion

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.

# **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, Zhanzhan 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
9. A. Graves, S. Fernandez, F. J. Gomez, and J. Schmidhu- ber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent eural networks. In ICML, 2006.
10. Baek, Y.; Lee, B.; Han, D.; Yun, S.; and Lee, H. 2019. Character region a wareness 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.
11. 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.
12. Xu, Y.; Wang, Y.; Zhou, W.; Wang, Y.; Yang, Z.; and Bai, X. 2019. Textfield: Learning a deep direction field for irregular scene textdetection. IEEE Trans.   
    Image Processing 28(11):5566–5579
13. 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.
14. 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.