Synergizing Optical Character Recognition: A Comparative Analysis and Integration of Tesseract, Keras, Paddle, and Azure OCR

Yuchen Li
SID: 490236424
yuli5818@uni.sydney.edu.au

Supervisors
Prof. Josiah Poon
Prof. Uwe Roehm

University of Sydney
Faculty of Engineering
School of Computer Science

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Data Science.

THE UNIVERSITY OF SYDNEY

February 11, 2024

Acknowledgement

I would like to first express my gratitude to Professors **Josiah Poon** and **Uwe Roehm** for their guidance across the entire 6 month of my project duration. Their instruction and professionalism on OCR topic is invaluable to the completion of my research and writing endeavors. Their readiness to address any quries has been nothing but supportive and enlightening.

I am also deeply indebted to my parents, Mr. **Naizhong Li** and Mrs. **Li Liu**, for their endless love and encouragement on my academic path. Some unfortunate events happened during my journey and I could not present the thesis if it were not their encouragement and understanding. Their steadfast belief in my potential and their sacrifices have been the bedrock of my perseverance. I am more than grateful for their support.

Last but not least, my truthful appreciation goes to **Siyao Li**, my partner and dearest confidant, whose companionship has been a source of immense comfort and joy throughout my college years. The encouragement, our shared late-night study sessions, and the moments of laughter and respite have been the highlights of this remarkable journey.

ABSTRACT

This study presents an extensive comparative evaluation of Optical Character Recognition (OCR) performance across various image preprocessing techniques such as Edge Detection and Image Thresholding, and various datasets including the IAM dataset and FUNSD dataset, to investigate the strengths and weaknesses of each OCR engine under different circumstances such as recognizing different hand-writings and extracting key information from complex layout forms under poor image resolution.

The OCR engines presented in the study include Tesseract OCR, Keras OCR, Paddle OCR, and Microsoft Azure Computer Vision. An innovative design and implementation of an integrated OCR Scoring System which, intricately assesses each engine's output, integrating confidence scores and Character Error Rates (CER) to determine the most accurate text extraction, is developed and evaluated.

The selection of evaluation metrics—accuracy, precision, recall, F1 score, Character Error Rate (CER), Word Error Rate (WER), and running time per image (RT)—was strategic to comprehensively distinguish the performance of individual OCR engines. Precision and recall offer insights into an OCR engine's inclusivity versus its focus on detail, which bears significance in handling complex data sets. Running time was tracked to evaluate processing speed, providing a pragmatic lens for understanding the trade-offs between speed and accuracy in real-world applications. demonstrate that while individual OCR engines have their strengths, the integrated

Our findings reveal that the Integrated Method, which amalgamates the strengths of individual OCR engines, consistently outperforms its constituent systems across all tested conditions. It demonstrates remarkable adaptability and resilience, achieving superior accuracy, precision, and recall rates, particularly in the face of the image quality fluctuations inherent in real-world tasks. Azure OCR emerged as a significant contributor to this integrated success, showing exceptional processing speed and precision, with notable proficiency in deciphering complex handwriting—a capability that suggests a high degree of immunity to image quality variations.

Contents

List of Figures

List of Tables

1 Introduction

The rapidly evolving landscape of Optical Character Recognition (OCR) technology has become a linchpin in the digital transformation of various sectors. From historical archives in the public sector to daily transactional documents in retail, OCR stands at the forefront of converting vast paper-based information into digital formats. This transition not only facilitates enhanced accessibility and analysis of data but also plays a crucial role in preserving historical records and streamlining operational workflows. This thesis aims to delve deeply into the nuances of OCR technology, exploring its potential and limitations across diverse applications. The introductory section sets the stage for a thorough investigation, outlining the research objectives and the structural roadmap of the thesis, thereby guiding readers through the anticipated discoveries and contributions of this study.

1.1 Research Objectives

The primary objective of this study focuses both on the comparative analysis between three open-source OCR engines, Tesseract OCR, Keras OCR, and Paddle OCR, and a commercial product Microsoft Azure Computer Vision, and also on an innovative integrated scoring system to evaluate individual OCR outputs and generate the optimal result. The goal is to discover the OCR engines' performance and robustness under varying circumstances.

1.2 Project Scope

Commencing with a comparative evaluation, this research aims to demonstrate the unique strengths, limitations, and characteristics of all engines under various circumstances. Thus IAM dataset and FUNSD dataset which, present a spectrum of text recognition challenges, from varied handwriting styles to texts in complex visual environments, were confined to the examination of the OCR systems.

The study also compares the impact of data preprocessing and post-processing brought to the recognition result, aiming to discover the robustness of each OCR engine towards image quality. Specifically, Canny Edge Detection and Otsu's Thresholding Method were applied in the hope of improving general accuracy by simplifying input images and increasing contrast.

The proposed integrated framework will incorporate a novel scoring mechanism, assessing the performance of each OCR tool based on metrics such as confidence levels and character error rate (CER), assisted by hugging face [15], [23], evaluated over every word recognized.

1.3 Structure Overview

The study starts by summarizing the empirical evidence from previous studies in section 2, literature review. It delves into the technological underpinnings and historical evolution of OCR, with a particular focus on Tesseract, Keras OCR, PaddleOCR, and Azure OCR. It addresses advancements in machine learning and AI that have propelled OCR forward. Canny Edge Detection, Otsu's Thresholding Method, and Autocorrect processing techniques are also included in this section to showcase their potential and limitations by existing experimental results.

In the methodology section, a rigorous comparative evaluation approach is laid out, detailing the criteria for assessing OCR performance in theory. This includes a thorough

explanation of the selected evaluation metrics, background introduction for datasets, and Image Pre-processing methods chosen for their relevance and diversity in real-world applications.

The experimental results and discussion provide a critical analysis of the performance of each OCR engine under various conditions. This section meticulously compares and contrasts their accuracy, processing speed, and adaptability, drawing on the research's findings to highlight the distinct capabilities and limitations of each system. The conclusions shed light on the future potential focuses on the research of the computer vision filed especially for character recognition tasks.

2 Literature Review

This literature review aims to dissect the chronological advancement and technological progression of OCR systems. It delves into the nuances of various OCR technologies, with a particular focus on the comparative analysis of leading systems: Tesseract OCR, Keras OCR, Paddle OCR, and Microsoft Azure Computer Vision. The exploration extends to critical pre-processing techniques, such as the Canny Edge Detector and Otsu's Method for Thresholding, evaluating their impact on OCR accuracy and robustness. The purpose here is firstly, to contextualize the current state of OCR technology within its historical trajectory, and secondly, to align this contextualization with the objectives of this research. By doing so, this review seeks to establish a comprehensive understanding of OCR's capabilities, limitations, and expectations for the experiment, particularly in the realm of system integration and enhancement through advanced image pre-processing methodologies.

2.1 Optical Character Recognition

2.1.1 OCR Technology

OCR technology converts different forms of text-containing documents into editable digital formats, a process facilitated by a series of computational phases, each addressing specific challenges associated with text recognition from images [6]. Although many methodologies and structures are applied in different OCR systems, a widely accepted OCR process will go through the following phases:

Segmentation Phase Novel techniques, such as those based on the Hough Transform, have been proposed for text line extraction from images. For example, a piecewise water flow technique has shown high success rates in extracting text lines from multi-skewed handwritten document images [17]. Additionally, segmentation accuracy is improved by methods like the vertical and horizontal projection profile method [13], which can achieve up to 98% accuracy in line and word segmentation.

Feature Extraction Phase In this phase, pertinent features from objects or characters are extracted to construct feature vectors. These vectors facilitate the classification process, making it easier to distinguish between dissimilar classes [6], [7]. Two main classes of features are recognized: statistical features, which are derived from the statistical distribution of points within the character matrix, and structural features, which relate to the geometrical properties of the character set.

Classification Phase OCR systems utilize various pattern recognition methodologies to assign each character to a predefined class [7]. Statistical decision theory underlies several OCR classification techniques, including Nearest Neighbor, Bayes classifiers, and Hidden Markov Models, among others. Neural networks are widely applied as the main backbone algorithm in many OCR engines for character classification due to their parallel processing capabilities and ability to learn from experience[6].

2.1.2 Individual Analysis of Selected OCR Engines

Tesseract OCR: A Multifaceted Tool with Language Versatility Tesseract OCR, initially developed by Hewlett-Packard and later evolved under the management of Google, has been a cornerstone in the development of OCR technology. The latest Tesseract 4, adding a Long-Short-Term-Memory (LSTM) neural network, focuses more

on line recognition and still supports the legacy Tesseract 3 which works by recognizing character patterns.

The combination of rule-based and statistical learning algorithms ensures the ability to handle a multitude of languages and scripts. Supported by Unicode (UTF-8), this multilingual capability is one of Tesseract's most lauded strengths, enabling its application in diverse linguistic contexts.

Studies focusing on Tesseract, such as those conducted by Ray Smith [20], have consistently highlighted its adaptability and accuracy in processing printed text. However, its performance is expected to be heavily dependent on the input image quality. The dependency on image quality posits a significant limitation, particularly in solving real-world problems that often involve recognizing texts in noisy or low-resolution images.

Keras OCR: Customizability at the Expense of Efficiency Keras OCR, recognized for its high-level Customizability, offers a broad avenue for specialized tasks using customized engines. This system's design, centered around user-defined customization, allows for the tailoring of OCR models to specific text types or image conditions.

The default Keras API provides two key architectures for OCR tasks: the Detector and the Recognizer. The Detector uses the ***CRAFT*** architecture for text detection, primarily supporting ***'clovaai_general'*** weights and a ***'vgg'*** backbone. It's designed for the effective detection of text regions in images. The Recognizer employs the ***CRNN*** architecture, suitable for recognizing text from image regions containing text. It allows customization of the alphabet and weights, providing flexibility for different OCR applications. These architectures are specifically tailored for handling the distinct challenges of detecting and recognizing text in varied image formats.

The study by Madhugiri and Devashree underscores the trade-offs in Keras OCR [12], particularly the computational overhead accompanying its customization flexibility. While it excels in image text extraction, its processing speed is somewhat mitigated by the extensive computational resources required, particularly evident in real-time processing scenarios. Thus, Keras OCR emerges as a powerful tool for customized solutions, albeit with considerations regarding efficiency and resource allocation.

Paddle OCR: Speed and Efficiency in Text Style Recognition Paddle OCR, developed by the Chinese AI firm PaddlePaddle, distinguishes itself with its rapid processing capabilities and efficient deep learning algorithms.

Paddle OCR's detection module primarily utilizes DB (Differentiable Binarization) for text detection, which is effective in handling various text scenarios. For the recognition part, it employs the CRNN (Convolutional Recurrent Neural Network) architecture. This combination of DB for detection and CRNN for recognition allows Paddle OCR to efficiently and accurately process text in images. The system is designed to be adaptable, supporting multiple languages and scripts, which is a testament to its versatility.

As evidenced in the evaluation by Liu et al. (2020), Paddle OCR's proficiency lies in its ability to quickly process and recognize a diverse array of text styles [3]. This makes it particularly effective in applications where text presentation varies significantly. However, its limitation surfaces in the form of restricted language support and a relatively smaller community base, factors that could potentially constrain its applicability across various linguistic domains.

Microsoft Azure Computer Vision: Out-of-the-Box Usability with Ecosystem Dependence Microsoft Azure Computer Vision, as a commercial OCR tool, offers a comprehensive service with high accuracy and easy API usage.

One of the key aspects that gives Azure OCR an edge over other OCR systems is its ability to handle both printed and handwritten text, including mixed languages and writing styles. However, the specific details about the internal architecture, including the exact models and backbones used for recognition and detection tasks, are proprietary to Microsoft and are not publicly disclosed in detail.

As appraised by William Ughetta and Brian W. Kernighan [21], the strength of this tool lies in its ready-to-use capabilities, which necessitate minimal setup, making it ideal for rapid deployment in diverse environments such as IoT and enterprise solutions. The flip side of this convenience is its potential reliance on the broader Microsoft ecosystem, which might impose limitations on its utility in non-Microsoft environments or applications requiring extensive customization outside the Microsoft suite.

2.2 Data Pre-processing

2.2.1 Otsu's Method for Thresholding

Otsu's method for thresholding is a key technique in character recognition, designed to effectively separate text from the background in images. This method calculates an optimal threshold, crucial for minimizing intra-class variance between black and white pixels. It operates on the principle that document images consist of two primary pixel classes: text and background, and by reducing variance within these classes, Otsu's method creates a clear distinction between them facilitating the OCR recognition,

Gupta et al. (2007) conducted a notable study that elucidates the efficacy of Otsu's method in the realm of OCR [5]. Their research, focusing on historical documents, demonstrates that Otsu's method effectively models the binarization process, catering to the unique challenges posed by historical texts. This method adeptly adapts to varied document conditions, reinforcing its suitability for documents where lighting and contrast inconsistencies are prevalent. The study underscores the method's capacity to generate a clear and discernible delineation between text and its backdrop, a fundamental prerequisite for reliable OCR.

Despite its demonstrated advantages, Otsu's method is not without limitations. The study by Gupta et al. (2007)[5] points out that the enhancements in OCR accuracy from image preprocessing, such as binarization, have their bounds, particularly when the OCR system operates independently without any feedback mechanism. This observation leads to the hypothesis that significant advancements in OCR accuracy might be achievable through systems that incorporate feedback loops into the binarization process. Such systems could potentially leverage human input for training or integrate natural language processing post-OCR to refine and contextualize the results.

2.2.2 Edge Detection Using Canny Edge Detector

The Canny edge detection algorithm helpful improving the accuracy of OCR systems by delineating the textual boundaries within images. This algorithm, established by John Canny, initially employs a Gaussian filter, pivotal in suppressing noise and smoothing the image. This step ensures that the potential distortions in images are minimized.

Subsequently, the algorithm computes the intensity gradient of the image, a precursor to the pivotal stage of non-maximum suppression. It then identify the directional change

in intensity across the image, which is the reason that the algorithm can highlight the image edges. Here, the algorithm refines these edges, ensuring that the resultant boundaries are as narrow and distinct as possible, a critical factor in enhancing text recognition accuracy.

Lastly the application of a double threshold technique discerns potential boundaries by categorizing pixels into strong and weak edges. This final phase, involves solidifying these detections, ensuring that only the most probable edges are retained, thereby culminating the edge detection process.

In an research consucted by Wu et al. (2017), the efficacy of the Canny edge detection algorithm was evaluated within the OCR domain, using the ICDAR2013 dataset [24]. The study's findings showed that: it achieved a recall rate of 72.4%, a precision of 88.3%, an F-score of 75.9%, and an accuracy of 90.5%. These metrics are indicative of the algorithm's robustness in enhancing text recognition, particularly when combined with clustering techniques like k-means, which further refine the text isolation process.

On the other hand, the limitation cannot be neglected. Firstly, it's sensitive to noise. Although incorporating a noise reduction step, canny edge detector remains susceptible to high noise levels. This sensitivity can lead to the false detection of edges and thus affecting the accuracy of text recognition. Secondly, the effectiveness depends largely on the parameter selection. The choice of parameter will affect the precision of boundaries. If the boundaries are either too generic (omit crucial edges) or too specific (include irrelevant ones), it may prevent the OCR from correctly recognizing.

2.3 Result Post-processing

In OCR systems, the role of post-processing can sometimes imoprove the results a lot more reasonable. Despite their advancement, these algorithms often encounter difficulties with complex text layouts, diverse fonts, and varied image qualities. The Python autocorrect library, utilized in this study, addresses this by correcting common spelling errors in OCR outputs. Operating through approximate string matching, as detailed in Chen (2019) [2], it compares each word in the OCR output against a comprehensive dictionary of correctly spelled words. When it finds an mismatch, the library proactively suggests the most probable correction based on lexical similarity analyses.

One of its advantages is its straightforward implementation and wide adaptability meaning that it can be easily integrated into OCR workflows without adjusting or tuning OCR configurations. It is proficient in rapidly processing text which gives it advantageous in large-scale or time-sensitive OCR applications. The library's efficacy in handling common spelling mistakes in English aligns well with general OCR tasks, making it a valuable tool for such applications.

However, there are notable constraints to this library. Its utility is predominantly limited to the English language, which restricts its application in OCR tasks involving multiple languages. Moreover, it lacks contextual understanding, potentially overlooking errors in cases where words, though spelled correctly, are misapplied. There is also a risk of over-correction, especially for less common or specialized terms, which the library might mistakenly modify to more frequently occurring words. Additionally, its limited customization options restrict its adaptability for specialized vocabularies, often crucial in specific OCR applications.

3 Methodology

The methodology for evaluating OCR technologies in this research involves a systematic approach to assess the capabilities of four OCR engines: Tesseract OCR, Keras OCR, Paddle OCR, and Microsoft Azure Computer Vision. The effectiveness of these engines is analyzed using two specific datasets:

- IAM Dataset: This collection provides various handwriting styles, enabling the evaluation of the OCR engines' proficiency in recognizing handwritten text.
- FUNSD Dataset: This dataset focuses on assessing the OCR engines' ability to extract information from documents with complex layouts and varying qualities.

The study incorporates image preprocessing techniques like the Canny Edge Detector and Otsu's Method for Thresholding to enhance text recognition, with a comparative analysis of OCR results before and after applying these methods. Post-processing includes the use of the Python autocorrect package for correcting spelling errors, crucial for practical usability assessment.

A unique aspect of the methodology is the development of an integrated OCR system. This system combines the strengths of the selected OCR engines through a scoring mechanism that employs a hybrid scoring mechanism based on each engine's confidence score and Character Error Rate (CER) for each recognized word. The final output for each text block is selected based on the highest score, thus capitalizing on the collective strengths of the engines.

This approach ensures a thorough and comparative analysis of the OCR technologies, utilizing advanced preprocessing and post-processing techniques to maximize text recognition accuracy in various document processing scenarios.

3.1 Evaluation Metrics and Analysis

In the assessment of OCR systems, several metrics are employed to gauge performance, including accuracy, F1 score, precision, recall, Character Error Rate (CER), and Word Error Rate (WER). These metrics provide a comprehensive evaluation, each focusing on different aspects of OCR performance.

Accuracy: This metric is a fundamental measure of the OCR system's performance, quantifying the proportion of total number of words recognized accurately to the total number of words. Mathematically, it is expressed as:

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision and Recall: Precision is the ratio of correctly predicted positive observations to the total predicted positives, while recall (or sensitivity) measures the ratio of correctly predicted positive observations to all observations in the actual class. They are calculated as:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

F1 Score: The F1 Score is the weighted average of Precision and Recall, thus taking both false positives and false negatives into account. It is a more robust measure than accuracy in scenarios where the class distribution is imbalanced. The F1 Score is defined as:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

In the experiment section 4, *'accuracy_score'*, *precision_score*, *recall_score* and *f1_score* from *'sklearn.metrics'* is utilized for easier computation.

Word Error Rate (WER): The challenge in evaluating OCR performance arises when the length of the word sequence recognized by the system differs from that of the reference sequence, which is assumed to be correct. Word Error Rate (WER) is a critical metric in this context, calculated based on the Levenshtein distance, but at the level of words rather than phonemes [15]. WER is instrumental in both comparing various systems and assessing improvements within a single system. However, while WER provides a quantitative measure of error, it does not offer insights into the specific nature of these errors. Consequently, additional analysis is necessary to pinpoint the primary sources of inaccuracies and direct research efforts effectively.

To address the disparity in sequence lengths, the recognized word sequence is aligned with the reference sequence using a technique known as dynamic string alignment. This alignment is crucial for an accurate calculation of WER. Theoretical frameworks, such as the power law, elucidate the relationship between perplexity and word error rate, offering a deeper understanding of how changes in system complexity impact performance. This connection underscores the importance of optimizing OCR systems not only for accuracy but also for the complexity of the tasks they undertake [23]. WER is computed as:

$$WER = \frac{(S + D + I)}{N} = \frac{(S + D + I)}{(S + D + C)}$$

Where S represents the number of substitutions, D denotes the deletions, I stands for insertions, C signifies the correct words, and N, is the total number of words in the reference $N = S + D + C$. A lower WER is indicative of better recognition performance, with few errors detected.

Character Error Rate (CER): The application of Word Error Rate (WER) in evaluating speech recognition systems might appear somewhat stringent, particularly when an entire word is deemed incorrect due to a single letter's error. This perceived rigor stems from the word-level evaluation approach of WER, where errors are annotated on a per-word basis. To address this, the Character Error Rate (CER) offers a more granular assessment by focusing on the character level. In CER, words are decomposed into constituent characters, and errors are identified and annotated for each character. This method provides a more nuanced evaluation, instrumental in scenarios where minor errors at the character level significantly impact the overall performance assessment.

CER is a metric particularly pertinent to OCR, representing the ratio of character-level errors (insertions, deletions, or substitutions) to the total number of characters in the reference text. It is calculated as follows:

$$CER = \frac{(S + D + I)}{N} = \frac{(S + D + I)}{(S + D + C)}$$

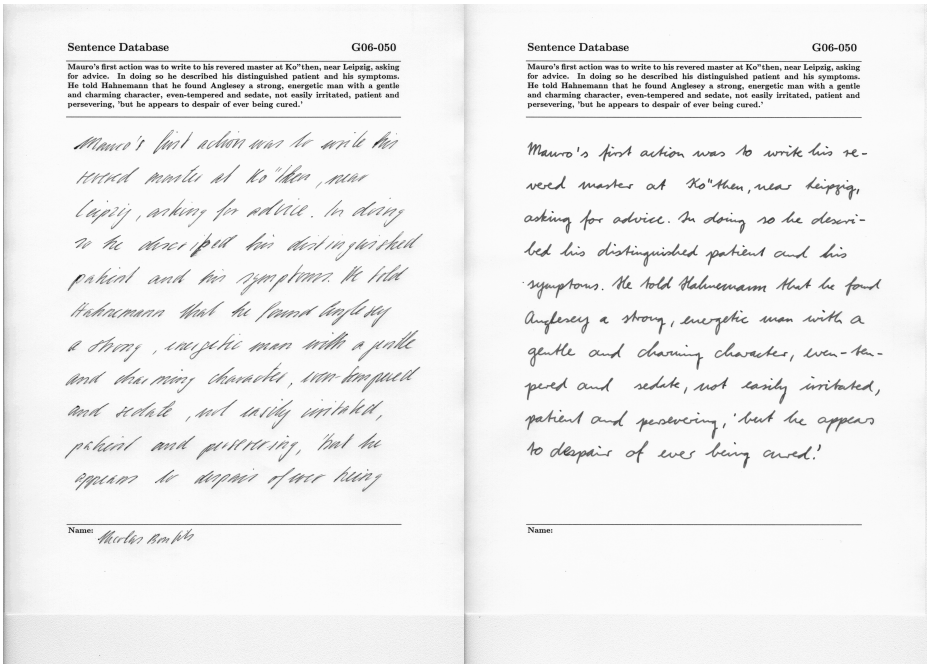
In the experimental implementation in section 4, The calculation is obtained using the *wer* and *cer* metrics from Hugging Face's *datasets* library.

3.2 Dataset Background

The efficacy of OCR systems hinges significantly on the datasets used for their evaluation. In this study, two distinct datasets are employed: the IAM Dataset and the FUNSD Dataset.

3.2.1 IAM Dataset

The IAM dataset, a cornerstone in handwriting recognition research, comprises a diverse collection of handwritten English text. Developed by the IAM Database Group at the University of Bern by Marti and Bunke [14], this dataset contains forms written by numerous writers, offering various handwriting styles. The text within the dataset is sourced from the Lancaster-Oslo/Bergen Corpus, ensuring a wide range of vocabulary and sentence structures. Examples are shown in figure 1a and 1b:



(a) Style 1 (b) Style 2

Figure 1: First Example of IAM Dataset

It can be seen from the above figures that the handwriting is a transcript of the printed text of different styles. The rich hand-writing styles enable us a solid foundation for testing and showcasing OCR engines' capability to recognize hand-writings in the real world. To achieve such a goal, the following preparation steps will ensure a targeted evaluation of each OCR engine's capabilities in handling different text types.

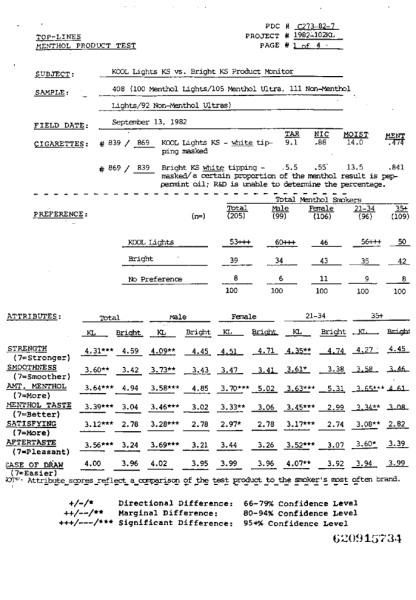
3.2.2 FUNSD Dataset

The Form Understanding in Noisy Scanned Documents (FUNSD) dataset, designed for the task of form understanding, comprises scanned images of noisy, unstructured forms, including annotations for semantic understanding [4].

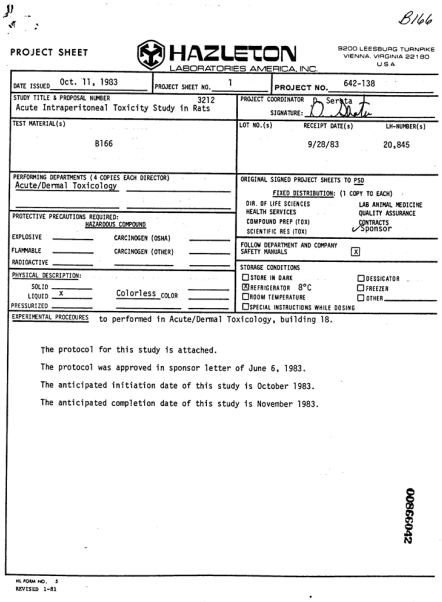
In pursuit of a robust and authentic evaluation of OCR system performance, this study utilizes a carefully curated subset of the RVL-CDIP dataset. The RVL-CDIP dataset—comprising an extensive collection of 400,000 grayscale images of assorted document types from the 1980s and 1990s—provides a realistic representation of documents.

The documents encapsulated are characterized by their modest resolution, averaging around 100 dots per inch (dpi). The images, predominantly of low quality, embody an assortment of noise artifacts attributable to repeated scanning and printing—a common occurrence in real-world archival scenarios. This attribute of the dataset is particularly significant as it presents OCR systems with practical challenges, closely mimicking the conditions under which many historical documents exist.

The development of the Form Understanding in Noisy Scanned Documents (FUNSD) dataset was an exercise in meticulous selection from the 'form' category of the RVL-CDIP collection. From an initial pool of 25,000 form images, a stringent review process was undertaken. This review eliminated forms that were deemed unreadable or excessively similar to others, yielding a pool of 3,200 eligible documents. A random sampling of these documents produced a final annotated set of 199 forms, which now comprises the FUNSD dataset utilized for OCR assessment in this study [9]. Examples are shown in figure 2:



(a) Form Style 1



(b) Form Style 2

Figure 2: Examples of FUNSD Data

The above figures demonstrated a fraction of many variations of styles and structures of forms in FUNSD Dataset. Some texts are too distorted or small to recognize which provided a ground basis for a comprehensive test on OCR engines' form understanding.

3.3 Image Pre-processing

An integral aspect of this research involves meticulous data preprocessing and the subsequent impact analysis on OCR performance. The study will rigorously examine how different image preprocessing techniques, namely Canny Edge Detector and Otsu's Method for Thresholding can help improve OCR performance.

3.3.1 Canny Edge Detector

The Canny Edge Detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986 [25]. The process of the Canny edge detection algorithm can be broken down into the following steps:

- Noise Reduction: Since edge detection is susceptible to noise in an image, the first step is to remove the noise with a Gaussian filter. The image is convolved with a Gaussian filter $G(x, y, \sigma)$, which smooths the image to reduce the impact of obvious noise.
- Gradient Calculation: The gradient of the smoothed image is calculated to find the intensity gradients. The edges should be marked where the gradients of the image have large magnitudes.
- Non-maximum Suppression: To get rid of spurious response to edge detection, the gradients of the image are thinned by applying non-maximum suppression, which means only the local maxima are kept as the edge pixels, and all others are suppressed.
- Double Threshold: Potential edges are determined by thresholding. A double threshold is applied to distinguish between strong, weak, and non-edge pixels, making the detector more robust to noise and variations in the image.
- Edge Tracking by Hysteresis: The final step involves converting weak edges into strong ones if and only if they are connected to strong edges. This step is based on the assumption that edges are long lines in the image and hence, should be connected.

Mathematically, the Gaussian filter applied during the noise reduction step can be represented as:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation of the Gaussian filter. An illustrative comparison example of before and after application of Cany Edge Detector is as follows:

Figure 3: Example of Cany Edge Detector

The Canny Edge Detector helps in OCR by highlighting the edges of the characters in an image, which can be particularly useful in segmenting characters from the background and each other in the case of cursive or connected scripts.

3.3.2 Otsu's Method for Thresholding

Otsu's method is an automated technique in image processing that is utilized to transform a grayscale image into a binary image. The method assumes a bimodal histogram to represent two distinct classes of pixels, foreground and background. It seeks to determine the threshold that optimally separates these classes by maximizing the inter-class variance. The algorithm comprehensively examines each potential threshold, assessing the variance of the foreground and background pixel intensities. The objective is to ascertain the threshold value where the combined spread of the target region's and background's pixel intensities is at its minimum, thereby facilitating the segmentation of the image into meaningful regions [18]. The method is based on the idea of finding the threshold that minimizes the intra-class variance, defined as a weighted sum of variances of the two classes:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Where $q_1(t)$ and $q_2(t)$ are probabilities of the two classes separated by a threshold t and $\sigma_1^2(t)$ and $\sigma_2^2(t)$ are variances of these two classes.

Figure 4 better demonstrates the comparison before and after the usage of Otsu's Method:

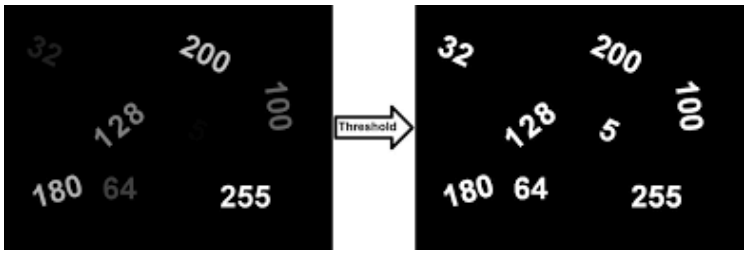


Figure 4: Example of Otsus Thresholding

Otsu's Method improves OCR by providing a clear demarcation of text against the background, especially useful in images with varying illumination and contrast. Adapting to the specific content of each image ensures that the text is properly segmented for recognition by the OCR engine.

3.4 Integrated OCR System Development

3.4.1 Conceptual Framework

The Hybrid Approach in OCR system design marks a significant departure from traditional methodologies by integrating outputs from multiple OCR engines, such as Keras, Paddle, Tesseract, and Microsoft Azure Computer Vision, at a more refined level. This approach pivots on the principle of selective amalgamation, wherein the optimal fragments from each engine's output are concatenated to form a superior composite result. This strategy, focusing on individual words or smaller text units, aims to leverage the unique strengths of each OCR engine, thereby enhancing the overall accuracy and reliability of text recognition. Figure 5 shows the conceptual flowchart:

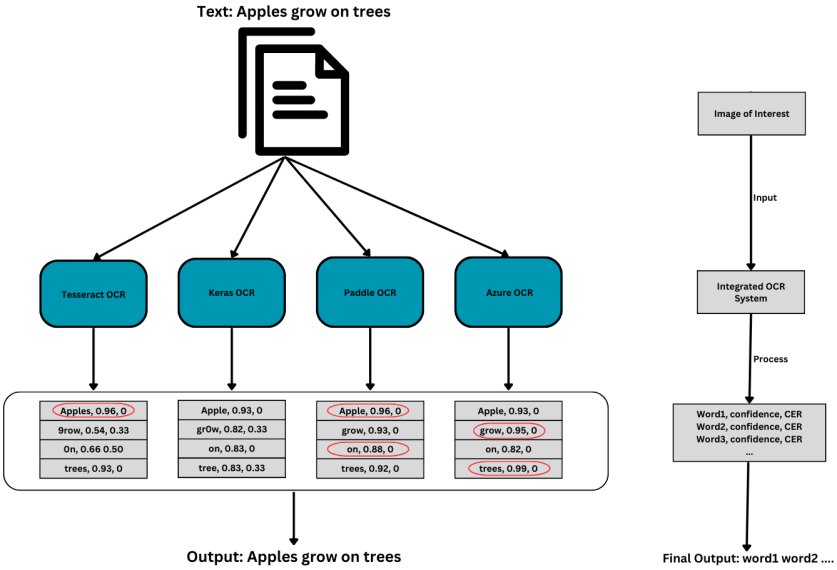


Figure 5: Integrated System Flowchart

3.4.2 Implementation Steps

The flowchart depicts the operational workflow of an integrated OCR scoring system designed to process a given text from an image. Firstly, an image containing the text *"Apples grow on trees"* serves as the input. The image is then simultaneously processed by four different OCR engines: Tesseract OCR, Keras OCR, Paddle OCR, and Microsoft Azure Computer Vision. Each OCR engine analyzes the image and provides its output, along with a confidence score and a Character Error Rate (CER) for each word detected.

For example, Tesseract OCR identifies "*Apples*" with a high confidence of 0.96 but mistakes "*grow*" for "*9row*" with a lower confidence and CER. Keras OCR recognizes "*Apple*" without the '*s*', and similarly has varying confidence levels and CERs for other words. Paddle OCR correctly identifies "Apples" and "grow" with high confidence and has lower CERs. Lastly, Azure OCR shows high confidence and low CER for most words, with "trees" having the highest confidence at 0.99.

The outputs of all OCR engines are then aggregated and passed on to the integrated scoring system to evaluate each word based on the provided confidence scores and CER from each OCR engine. If an OCR engine consistently performs well on certain types of texts, more weight is given to its output in the scoring process. The final output is a string of words that have been selected based on the highest score from the aggregated results, aiming to form the most accurate rendition of the text from the image.

4 Experiment

4.1 Hardware Specificaion

The experimental phase of the thesis is conducted using Google Colab, which provides a robust cloud-based platform ideal for high-performance computing tasks. The specifications of the environment are as follows:

- RAM: 51.0 GB, offering ample memory for large-scale data processing and model training.
- GPU: Tesla T4 with 15.0 GB RAM, providing powerful computational capabilities essential for deep learning models.
- Disk Space: A total of 166.8 GB is available, indicating sufficient storage for datasets, models, and intermediate outputs.

4.2 OCR Engine Setup

This section provides an in-depth description of how the OCR engines—Tesseract OCR, Keras OCR, Paddle OCR, and Microsoft Azure Computer Vision—are configured and employed in the study.

4.2.1 Tesseract OCR Configuration

Tesseract OCR is employed using its Python interface, *pytesseract*. The configuration parameter ***'–psm 6'*** is set to assume a single uniform block of text, enhancing the engine's ability to process structured text efficiently. The most recent versions of Tesseract, particularly from version 4.0 with LSTM were used in my experiment.

The architecture of this LSTM network within Tesseract 4 is built to be compute-intensive, utilizing multiple CPU cores in parallel where possible, and leveraging SIMD (Single Instruction, Multiple Data) parallelization on Intel/AMD processors that support SSE and/or AVX to handle the core matrix multiplications efficiently.

4.2.2 Keras OCR Configuration

Keras OCR is employed through the Python package *'keras-ocr'*: A package for OCR using Keras, which utilizes pre-trained weights for both detection and recognition tasks.

For detection, the CRAFT (Character Region Awareness for Text detection) architecture is employed with hyperparameters such as **weights** set to ***'clovaai_general***' and **backbone** architecture set to ***'vgg***'. detection_threshold is set to 0.7 to avoid including boxes that may have represented large regions of low-confidence text predictions. With **text_threshold** and **link_threshold** both set to 0.4, it means that the higher this value is, the less likely it is that characters will be merged into a single word. The lower this value is, the more likely it is that non-text will be detected.

For recognition, the CRNN (Convolutional Recurrent Neural Network) architecture is used, which allows customization through alphabet specification and build parameters. The model is equipped with **weights** pre-trained on the ***'kurapan'*** dataset, and the recognizer can be fine-tuned for specific text recognition tasks with various batch sizes and lowercase conversions.

4.2.3 Paddle OCR Configuration

Global Settings: Paddle OCR is implemented using Python packages *'paddlepaddle-gpu'*, *'paddleocr'*. The experiment utilizes the Paddle OCR version en_PP-OCRv4 detection model with GPU acceleration, setting the ***'use_gpu'*** parameter to true for enhanced performance. Epochs are set to 500, with a logging window of 20 and checkpoint saving every 50 epochs.

The ***DistillationModel*** with DB algorithm is employed, using a ***PPLCNetNew*** backbone at a scale of 0.75 and an ***RSEFPN*** neck with 96 out-channels. The ***DBHead*** is utilized with a kernel size of 50 for both the student and teacher models. An ***Adam*** optimizer with a Cosine learning rate starting at 0.001 and a warmup epoch of 2 is used. ***DistillationDBPostProcess*** is applied with specific thresholds for text detection. The ***DistillationMetric*** with ***DetMetric*** as the base metric is used to evaluate the model's performance during the training and evaluation process.

4.2.4 Microsoft Azure OCR Configuration

Global Settings: The Azure OCR engine is accessed via the ComputerVisionClient, with the necessary subscription key and endpoint. Implementation required packages *'google-cloud-vision (3.4.4)'*, *and 'azure-cognitiveservices-vision-computervision (0.9.0)'* which provide Python access to APIs for Google Cloud Vision and Azure Computer Vision services. The *read_in_stream* method is used for image processing, and results are fetched asynchronously. Each detected line of text is passed through an autocorrect function if enabled.

Architecture: Microsoft Azure's Read OCR, uses deep learning models, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), and a combination of both in architectures like CNN-RNN. It contains specialized neural network architectures to capture the nuances of handwritten texts better. For handling sequences, such as text in images, Long Short-Term Memory (LSTM) networks, are used to capture the context and dependencies in sequential data, which is crucial for text recognition tasks. Azure OCR employs transfer learning, where a model trained on a large dataset is fine-tuned for specific OCR tasks.

4.3 Preprocessing

4.3.1 Label Processing

IAM dataset: The preprocessing pipeline for the IAM dataset involves several key steps, initiated by the definition of a function *'process_ocr_data'*. This function is designed to facilitate the processing of OCR output through a series of structured and methodical steps, ensuring systematic data handling.

An essential aspect of this pipeline is the integration of XML data, which provides a structured format for storing and manipulating the OCR data. The inclusion of XML significantly enhances the flexibility and scalability of the data processing. Figure 6 below showed what it looks like in the original IAM label

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE form SYSTEM 'http://www.iam.unibe.ch/~fki/iamdb/form-metadata.dtd'>

<form created="2002-01-14" height="3542" id="a01-000u" last-modified="2002-07-17" skew="287" status="final" version="3.0_beta" width="2479" writer-id="000">
  <machine-printed-part>
    <machine-print-line text="A MOVE to stop Mr. Gaitskell from nominating any more Labour life Peers is to" />
    <machine-print-line text="be made at a meeting of Labour M Ps tomorrow. Mr. Michael Foot has put down" />
    <machine-print-line text="a resolution on the subject and he is to be backed by Mr. Will Griffiths, M P for" />
    <machine-print-line text="Manchester Exchange." />
  </machine-printed-part>
  <handwritten-part>
    <line ass="-187" asx="0" asy="739" character-width="999" dss="-187" dsx="0" dsy="831" fd0="12129" fd1="2994" fd2="-339" filter-width="5" id="a01-000u-00"
segmentation="ok" slant="90000" stroke-width="7333" threshold="154" text="A MOVE to stop Mr. Gaitskell from" ubs="-347" ubx="0" uby="769">
      <word id="a01-000u-00-00" sentence-start="yes" tag="AT" text="A">
        <cmp x="408" y="768" width="27" height="51" />
      </word>
      <word id="a01-000u-00-01" tag="NN" text="MOVE">
        <cmp x="507" y="768" width="63" height="46" />
        <cmp x="568" y="770" width="56" height="41" />
        <cmp x="631" y="768" width="38" height="41" />
        <cmp x="676" y="772" width="31" height="36" />
        <cmp x="691" y="766" width="29" height="12" />
      </word>
      <word id="a01-000u-00-02" tag="TO" text="to">
        <cmp x="796" y="764" width="26" height="43" />
        <cmp x="826" y="780" width="40" height="34" />
      </word>
      <word id="a01-000u-00-03" tag="VB" text="stop">
        <cmp x="919" y="783" width="36" height="28" />
        <cmp x="958" y="757" width="77" height="57" />
        <cmp x="1041" y="789" width="44" height="46" />
      </word>
```

Figure 6: Example of IAM Original Label

The pipeline segregates the OCR output into distinct directories, categorizing them based on the nature of the text, such as printed or handwritten. This segregation is instrumental in refining the data analysis, as it allows for tailored processing techniques for different text types. Figure 7 below demonstrates the processed label ready for use as reference.

Label:

```
Northern Rhodesia is a member of the Federation. Mr. Macleod was not at the week-end meeting. But he told M
Ps yesterday: "I have no knowledge of secret negotiations." He said Britain had an obligation to consult the
Federal Government. But the final decision remained with the British Government. Mr. James Callaghan,
Labour's Colonial spokesman, said Sir Roy had no right to delay progress in the talks by refusing to sit
round the conference table.
```

Image:

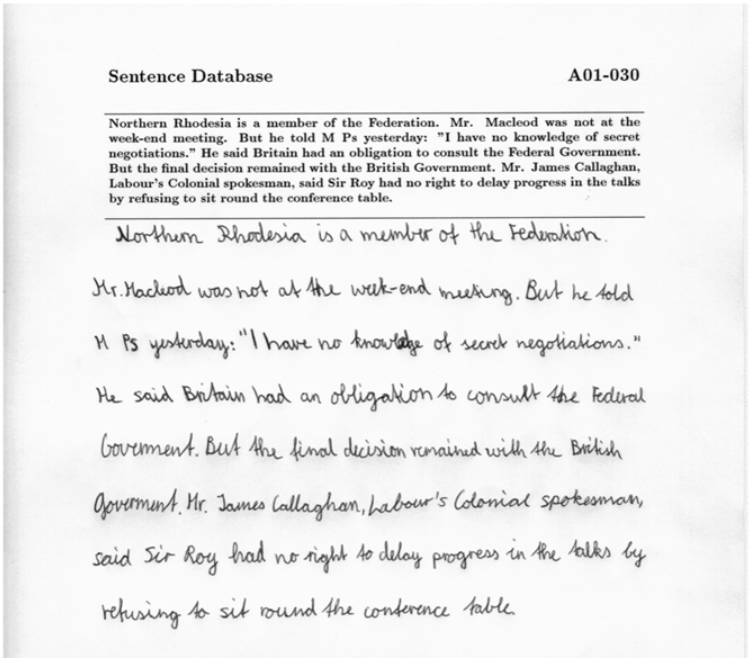


Figure 7: Processed IAM Label

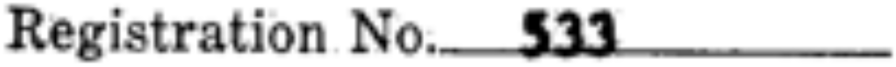
The implementation is designed to ensure precision in text categorization and accuracy in matching with the corresponding ground truth.

FUNSD dataset: The FUNSD dataset employs a structured JSON format that contains the semantic and spatial characteristics of forms. These entities, defined as coherent clusters of words that share semantic and spatial proximity, form the fundamental units of the dataset's structured representation and .

Each semantic entity within the JSON file is comprehensively described through several attributes:

- Unique Identifier: An identifier that serves as a distinct reference to the semantic entity within the form.
- Label: A categorical tag defining the role of the entity within the form's context, typically classified as 'question', 'answer', 'header', or 'other'.
- Bounding Box: The spatial domain of the entity is specified by a bounding box, articulated through coordinates following the schema $[x_{left}, y_{top}, x_{right}, y_{bottom}]$.
- Links: Directed connections denoted by [id_{from}, id_{to}] format that establishes the relational context between different semantic entities, where 'id' corresponds to the entity's unique identifier.
- Word List: An enumeration of constituent words, each word represented by its textual content and individual bounding box coordinates.

An example of such annotation is as shown by figure 8 below:




```
    {
    "form": [
    {
        "id": 0,
        "text": "Registration No.",
        "box": [94,169,191,186],
        "linking": [
            [0,1]
        ],
        "label": "question",
        "words": [
            {
                "text": "Registration",
                "box": [94,169,168,186]
            },
            {
                "text": "No.",
                "box": [170,169,191,183]
            }
        ]
    },
    {
        "id": 1,
        "text": "533",
        "box": [209,169,236,182],
        "label": "answer",
        "words": [
            {
                "box": [209,169,236,182
                ],
                "text": "533"
            }
        ],
        "linking": [
            [0,1]
        ]
    }
]
}
```


Figure 8: Example of FUNSD Annotation

It can be seen from the above figure that for each *'text'*, a label is assigned to describe its nature in the form context. For example *'Registration No.'* is labelled as question and *'533'* is labelled as answer. This provided a sound environment for us to test the integrity of information in a form.

In the preparation of the FUNSD annotation pipeline, the process commences with the extraction of data from a JSON file, parsing it to delineate semantic entities. Each entity is scrutinized to glean vital attributes: a unique identifier, label, bounding box coordinates, inter-entity links, and the list of constitutive words. The bounding box data is pivotal, facilitating spatial analysis to understand the layout intricacies of the form, essential for associating linked questions and answers spatially.

A relational map is then constructed, connecting questions to their answers using the Links attribute, essentially forming a dictionary where questions direct to their respec-

tive answers. Following this, the reconstruction of text for each linked pair ensues, extracting and ordering words accurately as they appear in the form. This reconstructed text is then elegantly formatted into a structured text file, preserving the question-answer sequence, differentiated by delimiters or new lines for clarity.

To enhance accuracy, this setup is integrated with Optical Character Recognition (OCR) outputs, ensuring that the text for entities is sourced directly from OCR results, matched with the corresponding identifiers. A subsequent post-processing step ensures the reconstructed text adheres to quality and format standards. The culmination of this pipeline is the saving of these meticulously generated labels in a text file or another format, tailored for testing the OCR's prowess in extracting information. Figure 9 illustrates the result of processed annotation file:

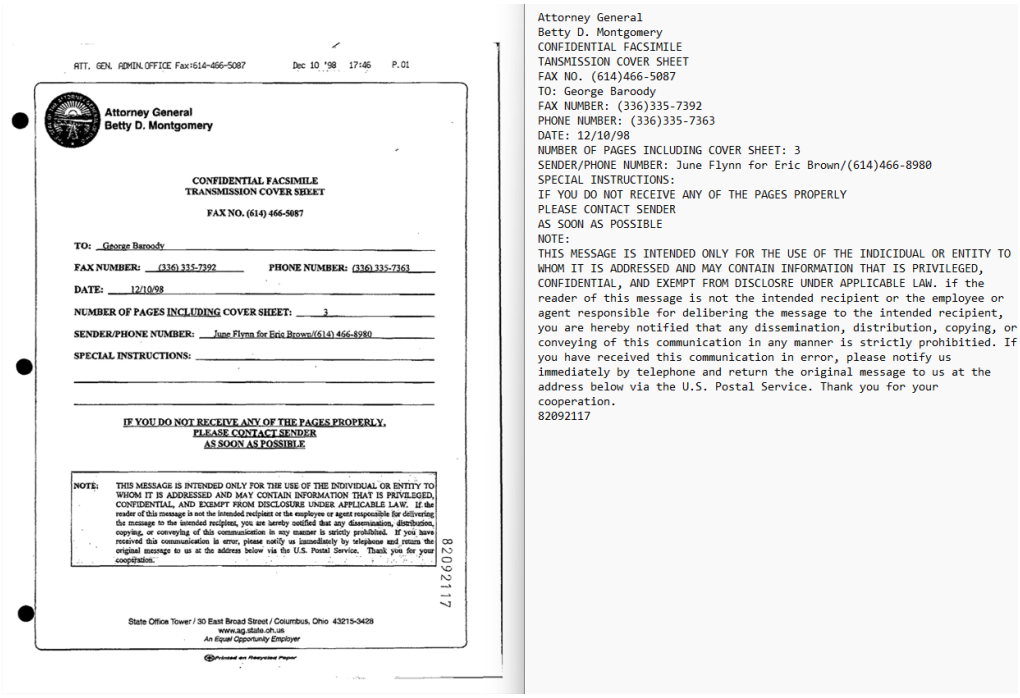


Figure 9: Result of FUNSD annotation

4.3.2 Image Processing

Canny Edge Detection: In the presented study, the *'cv2.Canny'*function within OpenCV is employed. This function implements the Canny Edge Detector, an algorithm well-regarded for its ability to detect a wide range of edges in images. The algorithm operates in a multi-stage process to detect edges: The procedure is executed in a controlled environment, where images are systematically converted to grayscale to conform to the algorithm's requirements.

The Canny Edge Detection is parameterized by two threshold values that govern the detection of edges, set at 100 and 200 respectively in this instance. The selection of these thresholds is critical, as it influences the sensitivity of the edge detection process. A lower threshold may result in the detection of spurious edges, while a higher threshold may overlook significant yet subtle edges.

Upon the application of the Canny Edge Detector, the resultant edge-highlighted images are stored in a dedicated subdirectory, thereby maintaining an organized structure for the processed data. This edge-enhanced representation of images is anticipated to facilitate the OCR's ability to discern and extract textual elements by providing a clearer distinction between text and background, thereby potentially enhancing the accuracy of subsequent data extraction and recognition tasks.

The images processed using the Canny Edge Detector from OpenCV are expected to have enhanced edges, which is essential for the OCR's task of detecting and recognizing characters. Examples are shown by figure 10 below:

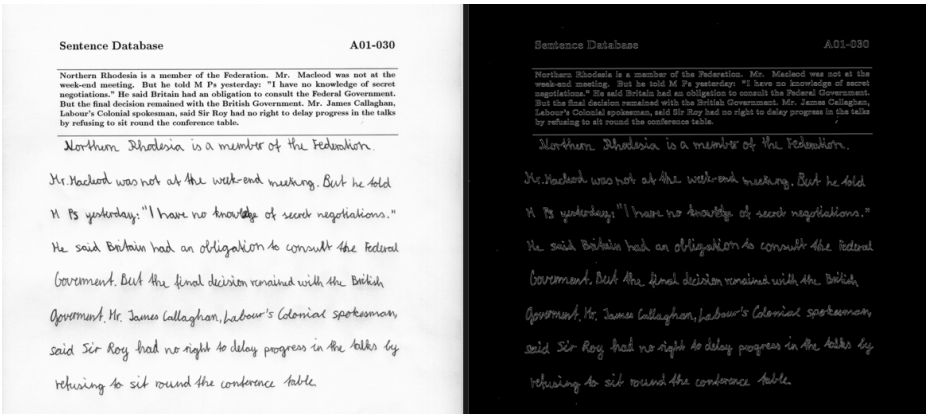


Figure 10: Canny-Edge Processed Comparison

The sharpness of edges directly influences the OCR's ability to segment characters from their backgrounds, a fundamental step in text recognition. By utilizing OpenCV's comprehensive suite of functions, the preprocessing stage is optimized to produce images that are primed for efficient and accurate OCR performance.

Otsu's Thresholding Method: Otsu's Thresholding method is adeptly integrated as a binarization technique to enhance the feature extraction capabilities of OCR systems when applied to the IAM and FUNSD datasets. Leveraging the *OpenCV* library, the implementation systematically applies Otsu's method to convert grayscale images into a binary format. This approach is predicated on the algorithm's capacity to compute an optimal threshold value that segregates the pixel intensity into two classes, foreground, and background, thus amplifying the contrast between textual content and its surrounding space.

The procedural execution begins with the creation of a targeted directory for the thresholded images, ensuring an organized storage structure. The os.*'listdir'* function is employed to enumerate through the images in the specified directories, selectively processing those with a *'.png'* extension, indicative of image files.

Upon loading an image in grayscale, the *'cv2.threshold'* function is invoked with the flag *'cv2.THRESH_BINARY'* and *'cv2.THRESH_OTSU'*. The former flag indicates the binary nature of the thresholding process, while the latter automatically determines the optimal threshold value based on the bimodal histogram of pixel intensities, a distinguishing feature of Otsu's algorithm. This automatic thresholding is particularly ad-

vantageous as it obviates the need for manual calibration and is therefore adaptable to varying document conditions with differing lighting and contrast scenarios.

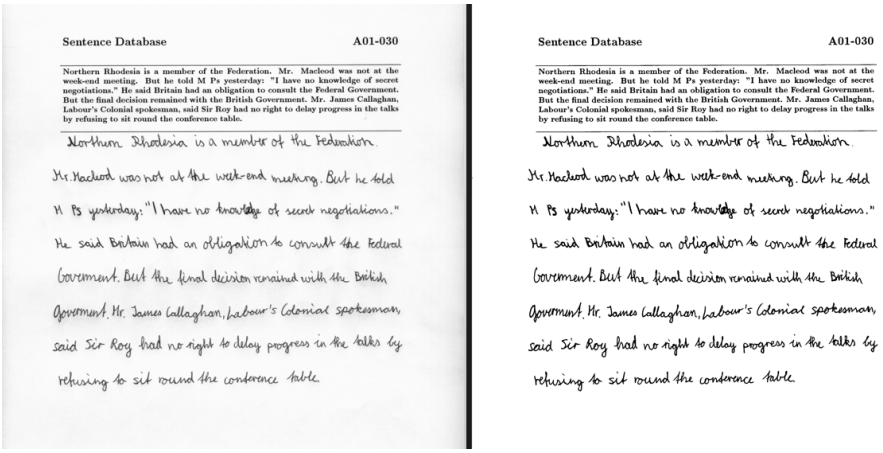


Figure 11: Otsu's Processed Comparison

The resultant images, demonstrated by figure11 now rendered in stark binary contrast, are then saved to the designated directory. This binary representation is pivotal in reducing the computational complexity for OCR systems by delineating the text more distinctly from the background, thereby facilitating improved accuracy in text detection and recognition.

Otsu's Thresholding, through its self-adjusting nature, presents a robust preprocessing step within the realm of document analysis and OCR processing, enhancing the capability of OCR technology to perform reliably across diverse document types and conditions.

4.4 Development of Integrated OCR System

The design of the integrated OCR system

4.5 Performance Analysis

In this study, we have designed a comprehensive experimental framework to evaluate the performance of various Optical Character Recognition (OCR) engines in recognizing printed and handwritten text, as well as their ability to extract key information from complex layouts. The experiments leverage two distinct datasets: the IAM dataset, which consists of identical content in both printed and handwritten formats to assess OCR capabilities in font robustness and handwriting recognition, and the FUNSD dataset, which is utilized to evaluate OCR performance in the context of complex document layouts.

The evaluation pipeline for Optical Character Recognition (OCR) adopts a comprehensive and systematic methodology to analyze the accuracy and efficiency of text extraction from images. This analysis spans various OCR engines and preprocessing methods, employing datasets such as IAM and FUNSD. The process begins by selecting the OCR engine, output type, and preprocessing technique (raw, canny, or otsu), thereby standardizing the input data quality.

In this pipeline, the core function `calculate_tp_fp_fn` computes true positives (TP), false positives (FP), and false negatives (FN) by comparing the predicted and reference strings character by character, identifying TPs through exact matches. FP and FN are then calculated by accounting for the discrepancies in predicted and reference strings. Following this, the `calculate_metrics` function calculates precision, recall, F1 score, and accuracy for 106 images in the IAM dataset (both printed and handwriting parts) and 50 images in the FUNSD dataset. These metrics, alongside the Character Error Rate (CER) and Word Error Rate (WER), provide a quantifiable measure of OCR performance, reflecting the system's capacity for accurate text extraction.

Crucially, the Seconds per File (spf) metric, computed by dividing the total processing time by the number of images in each dataset, assesses the processing efficiency of the OCR engines. The `evaluate_ocr` function interfaces with structured file systems based on OCR output conventions, facilitating a comparative analysis between predictions and references. Performance metrics are aggregated, and the final output presents a statistical overview of each OCR engine's performance, integrating standard metrics such as Accuracy, Precision, Recall, F1 Score, CER, WER, and spf. This methodical approach ensures a comprehensive and fair evaluation of OCR systems, providing insights into their effectiveness in different scenarios and conditions.

4.5.1 IAM Results

For the IAM dataset, the experiments are conducted into two sections: printed and handwriting. This division, detailed earlier in the article, aims to investigate each OCR engine's proficiency in discerning between different font styles and handwritten text.

Experiment	**IAM Dataset – Printed**						
Raw without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	60.65	64.21	63.91	64.06	0.08	0.1	8.4
Keras	45.39	47.77	45.27	46.49	0.12	0.15	10.8
Paddle	67.81	75.49	74.43	74.96	0.03	0.05	4.8
Azure	79.92	82.91	82.6	82.75	0.01	0.02	3
Integrated Method	81.27	85.33	84.68	85	0.01	0.01	NA
Edge Detection without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	58.43	61.14	60.77	60.95	0.09	0.1	2.4
Keras	42.92	45.28	44.6	44.94	0.13	0.15	7.2
Paddle	68.24	73.91	72.84	73.37	0.03	0.04	3.6
Azure	78.92	82.91	82.6	82.75	0.01	0.02	1.8
Integrated Method	80.58	83.62	83.29	83.45	0.01	0.01	NA
Thresholding without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	64.22	68.75	68.23	68.49	0.07	0.09	1.8
Keras	52.4	54.53	53.79	54.16	0.09	0.12	10.8
Paddle	68.21	76.13	75.84	75.98	0.03	0.05	3.6
Azure	80.12	83.04	83.29	83.16	0.01	0.02	3
Integrated Method	81.33	85.84	85.13	85.48	0.01	0.01	NA
Raw with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	61.03	64.82	64.32	64.57	0.08	0.09	8.4
Keras	46.57	48.23	47.71	47.97	0.12	0.14	11.4
Paddle	67.9	75.52	75.13	75.32	0.03	0.05	4.8
Azure	80.02	83.14	82.6	82.87	0.01	0.02	3.6
Integrated Method	81.27	85.33	84.68	85	0.01	0.01	NA
Edge Detection with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	59.39	62.47	62.15	62.31	0.08	0.09	3
Keras	44.23	46.51	45.92	46.21	0.12	0.13	8.4
Paddle	68.35	73.35	72.69	73.02	0.03	0.04	3.6
Azure	78.95	83.06	82.84	82.95	0.01	0.02	2.4
Integrated Method	80.58	83.75	83.46	83.6	0.01	0.01	NA
Thresholding with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	65.48	70.13	69.75	69.94	0.06	0.08	2.4
Keras	55.17	55.82	56.19	56	0.08	0.11	10.8
Paddle	71.15	77.23	76.81	77.02	0.03	0.04	4.2
Azure	82.64	84.75	84.98	84.86	0.01	0.02	3
Integrated Method	83.19	85.34	85.61	85.47	0	0.01	NA

Table 1: Performance metrics for different experiments on the printed texts on IAM dataset.

Experiment	**IAM Dataset – Handwriting**						
Raw without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	10.32	14.63	14.27	14.45	0.58	1.04	13.2
Keras	8.91	10.33	10.7	10.51	0.68	1.02	15.6
Paddle	15.48	20.09	20.77	20.42	0.48	0.79	6
Azure	59.32	62.15	62.54	62.34	0.08	0.24	4.2
Integrated Method	60.13	63.52	63.98	63.75	0.07	0.23	NA
Edge Detection without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	9.48	12.37	11.72	12.04	0.65	1.28	7.8
Keras	6.85	9.42	9.6	9.51	0.73	1.24	15
Paddle	16.32	20.57	20.96	20.76	0.39	0.68	5.4
Azure	58.65	61.37	61.84	61.6	0.08	0.23	4.2
Integrated Method	59.75	62.08	62.39	62.23	0.08	0.22	NA
Thresholding without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	13.58	17.93	18.12	18.02	0.42	0.96	6
Keras	10.32	14.67	14.3	14.48	0.57	0.99	13.2
Paddle	16.83	22.36	22.85	22.6	0.36	0.62	4.8
Azure	60.83	63.52	63.88	63.7	0.08	0.23	3.6
Integrated Method	61.02	63.95	64.37	64.16	0.07	0.22	NA
Raw with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	12.15	15.29	14.87	15.08	0.53	0.98	13.8
Keras	9.85	12.56	12.03	12.29	0.63	1.02	16.2
Paddle	16.07	21.51	22.16	21.83	0.41	0.73	6.6
Azure	60.28	62.69	63.01	62.85	0.08	0.23	4.8
Integrated Method	60.54	63.52	64.3	63.91	0.07	0.22	NA
Edge Detection with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	10.02	12.68	12.72	12.7	0.55	1.17	8.4
Keras	7.21	9.84	10.19	10.01	0.69	1.15	15.6
Paddle	16.51	21.06	21.24	21.15	0.38	0.64	6
Azure	58.87	61.52	62.03	61.77	0.08	0.22	4.8
Integrated Method	59.81	61.87	62.48	62.17	0.08	0.22	NA
Thresholding with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	14.69	18.42	19.27	18.84	0.39	0.91	6.6
Keras	12.15	15.49	16.06	15.77	0.51	0.93	13.8
Paddle	17.64	22.95	23.47	23.21	0.34	0.59	5.4
Azure	61.98	64.25	64.56	64.4	0.07	0.22	4.2
Integrated Method	62.44	64.72	65.28	65	0.07	0.22	NA

Table 2: Performance metrics for different experiments on the hand-writing texts on IAM dataset.

4.5.2 FUNSD Results

The FUNSD dataset is employed to test the OCR engines' effectiveness in key information extraction from documents with complex layouts. This experiment aims to quantify the impact of layout complexity on the performance of different OCR engines.

Experiment	**FUNSD Dataset**						
Raw without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	57.73	51.62	84.03	63.95	0.11	0.14	10.2
Keras	51.24	45.42	68.03	54.47	0.13	0.14	13.4
Paddle	72.93	70.46	84.82	76.98	0.06	0.09	5.7
Azure	75.25	79.83	76.04	77.89	0.04	0.06	4.8
Integrated Method	79.33	79.83	84.82	82.25	0.02	0.02	NA
Edge Detection without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	54.26	49.74	80.65	61.53	0.13	0.17	5.7
Keras	49.75	43.06	64.71	51.71	0.15	0.19	12.8
Paddle	70.68	71.32	82.96	76.7	0.07	0.1	3.2
Azure	73.19	77.35	73.18	75.21	0.05	0.08	2.5
Integrated Method	77.33	78.12	83.68	80.8	0.04	0.06	NA
Thresholding without Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	59.16	54.38	85.86	66.59	0.09	0.12	4.3
Keras	54.62	48.71	70.96	57.77	0.11	0.13	12.8
Paddle	75.32	71.28	85.09	77.58	0.05	0.08	3.6
Azure	77.83	80.26	78.14	79.19	0.03	0.05	2.1
Integrated Method	80.15	80.97	85.74	83.29	0.01	0.02	NA
Raw with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	58.21	52.58	84.06	64.69	0.11	0.13	10.8
Keras	52.67	46.07	69.14	55.3	0.12	0.14	13.4
Paddle	73.43	71.92	84.82	77.84	0.06	0.08	6.2
Azure	75.88	80.05	76.89	78.44	0.04	0.05	5.1
Integrated Method	79.33	80.05	84.82	82.37	0.02	0.02	NA
Edge Detection with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	56.73	51.28	81.29	62.89	0.13	0.16	6.3
Keras	51.28	44.15	65.03	52.59	0.15	0.18	13.2
Paddle	71.32	72.54	83.18	77.5	0.06	0.09	3.6
Azure	74.03	77.94	74.26	76.06	0.04	0.07	2.8
Integrated Method	77.33	78.35	83.92	81.04	0.04	0.06	NA
Thresholding with Autocorrect	Accuracy	Precision	Recall	F1	CER	WER	RT (spf)
Tesseract	60.38	55.81	86.02	67.7	0.09	0.11	4.8
Keras	55.14	49.06	71.38	58.15	0.11	0.12	13.2
Paddle	75.68	71.82	85.26	77.97	0.05	0.07	4.1
Azure	78.21	80.93	79.62	80.27	0.03	0.04	2.6
Integrated Method	80.28	81.06	85.92	83.42	0.01	0.02	NA

Table 3: Performance metrics for different experiments on the FUNSD dataset.

4.6 Discussion

4.6.1 Overview

The experimental findings indicated by all result tables illustrated that the Integrated Method powered by the scoring system consistently outperforms other OCR engines across different preprocessing techniques. Azure generally holds the second-best position, followed by Paddle, Tesseract, and Keras, respectively. The Integrated Method demonstrates notable stability in its performance, suggesting robustness that may be attributed to advanced algorithmic design capable of handling image variabilities.

4.6.2 Impact of Image Preprocessing

Recognition Ability

Edge detection preprocessing, specifically the Canny edge detection method, results in a general performance decline across all OCR engines and all datasets. However, Paddle shows a slight improvement, while Azure and the Integrated Method are less impacted, indicating their robustness to image quality fluctuations, possibly due to their underlying architectures designed to be less sensitive to input noise and variance.

Otsu's thresholding method positively impacts performance metrics across all OCR engines, with Tesseract and Keras exhibiting the most substantial gains. This improvement may stem from the method's binary thresholding process, which reduces input variability and simplifies character recognition, particularly for engines that might struggle with nuanced grayscale values.

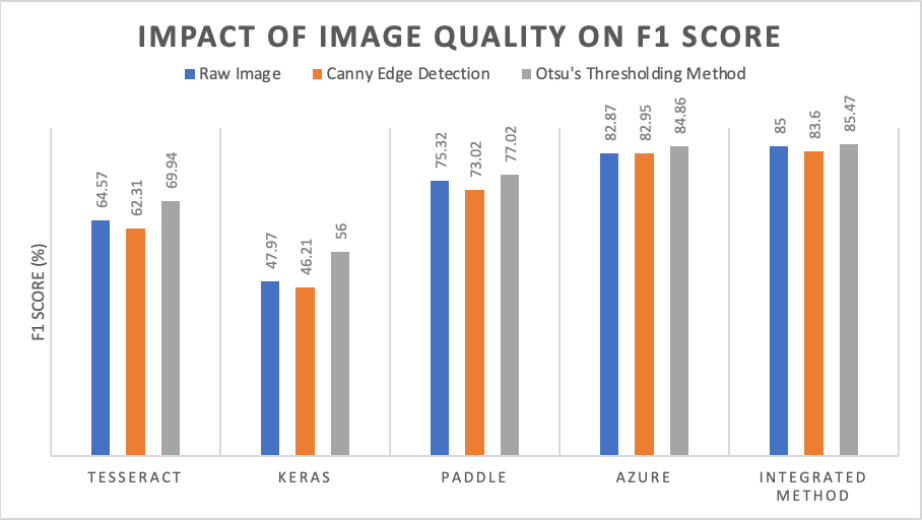


Figure 12: Impact of Image Quality on F1 Score

Figure 12 presents a comparative analysis of the F1 scores achieved by various Optical Character Recognition (OCR) engines, including Azure, Paddle, Tesseract, Keras, and an Integrated System, when tested on the IAM Printed dataset. The data indicates that the Integrated System exhibits a marginally superior performance compared to the other OCR engines. Additionally, the application of Otus's Thresholding method has

been shown to enhance the performance of all evaluated OCR engines, with particularly notable improvements observed in the cases of Tesseract and Keras.

Running Time per Image

Figure 13 elucidates the average OCR recognition time for various OCR engines. According to the data, Azure emerges as the swiftest engine in processing images, showing minimal sensitivity to variations in image quality. On the other hand, Keras is identified as the slowest among the engines. Tesseract notably demonstrates the highest sensitivity to image quality. This sensitivity can be technically attributed to Tesseract's reliance on image preprocessing methods that are more affected by image quality variations. For instance, Tesseract's performance is significantly influenced by factors such as noise, contrast, and resolution, which can dramatically alter its recognition speed and accuracy.

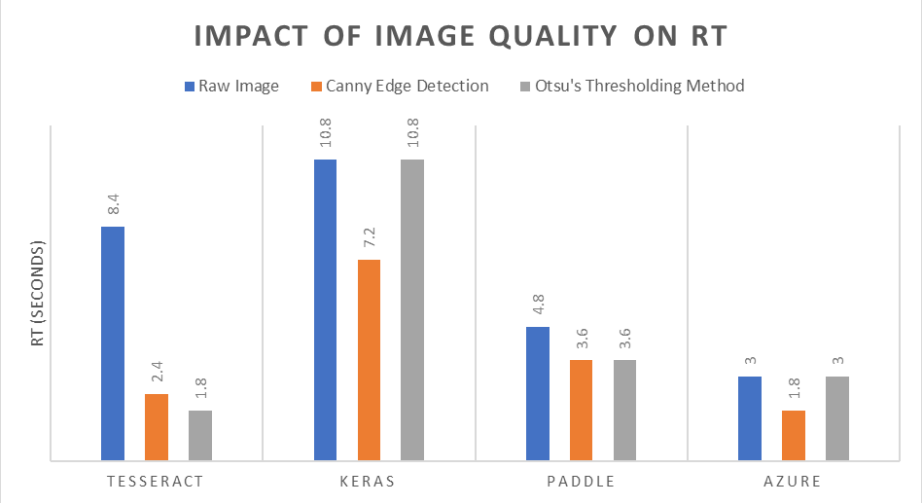


Figure 13: Image Quality on RT

The observed reduction in OCR engines' processing time, owing to preprocessing techniques, is largely expected. This reduction can be technically rationalized by considering that these techniques, such as binarization and noise reduction, simplify the image by reducing its dimensionality and complexity. As a result, the OCR system encounters fewer variations and complexities, facilitating faster processing. Preprocessing methods like Otsu's Thresholding and adaptive binarization typically convert a grayscale image to a binary image, which simplifies the text recognition process.

However, the exception of Otsu's Thresholding on Keras OCR highlights an interesting anomaly. This could be due to Keras OCR's underlying architecture, which does not synergize well with the binarization method employed by Otsu's algorithm. Keras OCR relies on a more detailed analysis of pixel intensity and color variations, which Otsu's Thresholding, by converting images to binary form, might oversimplify. This oversimplification can lead to the loss of critical details necessary for Keras OCR's algorithms, ultimately resulting in increased processing time.

4.6.3 Impact of Auto Correct

Recognition Ability

Autocorrect features prove to be moderately beneficial, particularly in reducing the Word Error Rate (WER), but do not substantially influence other metrics. Notably, the impact of autocorrect is more pronounced for Keras and Tesseract compared to Paddle, Azure, and the Integrated System. This differential effect is due to the former OCR engines' reliance on exact character matches, which are more prone to error without the assistance of a context-aware autocorrect system.

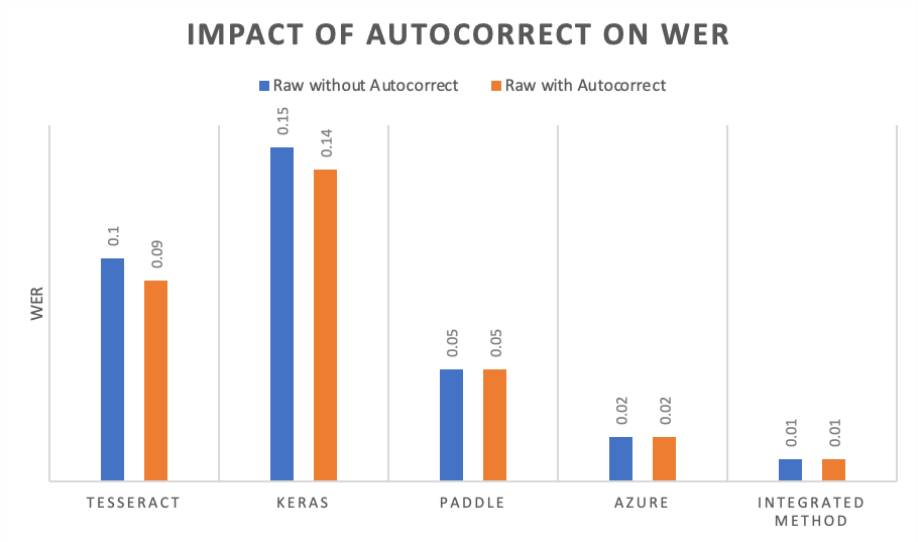


Figure 14: Impact of Autocorrect on WER

Engines with a more robust approach to character recognition, capable of contextual understanding and error correction, will inherently exhibit higher stability and performance, even in the face of varying input quality. This is evident in the performance metrics, where engines with advanced image preprocessing and error correction algorithms, such as Azure and the Integrated Method, consistently demonstrate superior and more stable results.

Running Time per Image

Figure 15, utilizing the IAM printed datasets on non-pre-processed images, demonstrates an expected trend. This pattern holds true across various datasets, including FUNSD, IAM Printed, and IAM Handwriting. It is observed that the integration of an additional autocorrect step in the OCR process tends to increase the processing time by approximately 0.5 seconds per image. This increment, however, does not seem to confer any significant benefits.

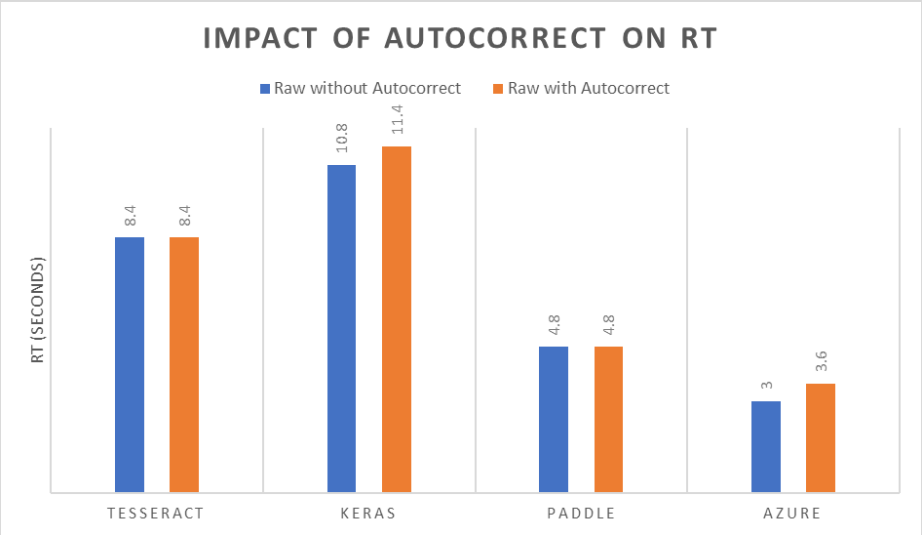


Figure 15: Impact of Autocorrect on Processing Speed

The redundancy of the autocorrect feature in some scenarios can be attributed to the advanced capabilities of many OCR algorithms like Tesseract, Paddle, and Azure. These OCR engines are equipped with built-in libraries that aid in text recognition by referencing a vast database of words and common text patterns. Consequently, the incorporation of an autocorrect step doesn't substantially enhance the accuracy or efficiency of these systems but merely affect the processing speed.

Given this context, a more sophisticated post-processing approaches, such as Natural Language Processing (NLP) powered contextual post-processing in OCR would be more appropriate. NLP techniques can analyze the context surrounding the recognized text, enabling a more nuanced understanding and correction of errors. Unlike standard autocorrect, which typically relies on dictionary-based word replacement, NLP-driven contextual processing can interpret the syntax and semantics of the text. This allows for more accurate corrections based on sentence structure, grammar, and even thematic elements. For instance, an NLP system could correctly interpret and process homonyms based on sentence context, a task beyond the capability of standard autocorrect algorithms.

4.6.4 Impact of Handwriting

Recognition Ability

The bar chart 16 illustrates the F1 scores for different OCR engines when applied to printed text and handwriting. It shows that all OCR engines tend to perform a lot less effectively on handwritten text compared to printed text. Tesseract shows a significant difference in performance, with a much higher F1 score for printed text. Keras OCR and Paddle OCR also display better performance for printed text, but the difference is not as pronounced as with Tesseract. Azure OCR exhibits a relatively smaller disparity between printed and handwritten text performance.

This disparity is primarily because OCR works by pattern recognition which is easier for printed text recognition. Howver, human handwriting, especially in cursive form, typically lacks consistent patterns. Moreover, handwriting is usually more diverse and

complex, often featuring variations in character shapes, sizes, and spacings, which challenges the pattern recognition capabilities of general OCR technologies.

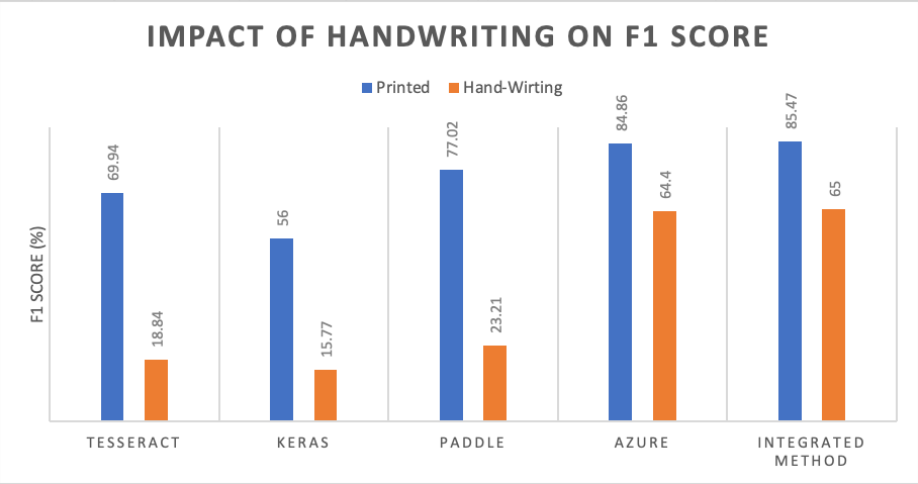


Figure 16: Impact of Handwriting on F1 Score

Azure OCR, however, stands out in handwriting recognition compared to other OCR models. One of the key reasons is its advanced machine-learning algorithms trained on a wide range of both handwriting and printed samples. This extensive training allows Azure OCR to better adapt to the variability and nuances of handwritten text, resulting in more accurate recognition. Additionally, Azure OCR has employed sophisticated techniques like Intelligent Character Recognition (ICR), which is specifically designed to handle the complexities of handwritten characters and cursive writing.

By combining the outputs, the integrated system leverages diverse recognition capabilities, likely leading to a more balanced and accurate performance across different text types. The slight margin of improvement suggests that while integration can harness the strengths of each OCR engine, the overall gain is dependent on how well each engine complements the others in handling the variations inherent in handwritten text.

Running Time per Image Figure 17 indicates that OCR engines generally take longer to process handwritten text compared to printed text. This increased processing time can be attributed to the inherent complexity and variability of handwriting mentioned above.

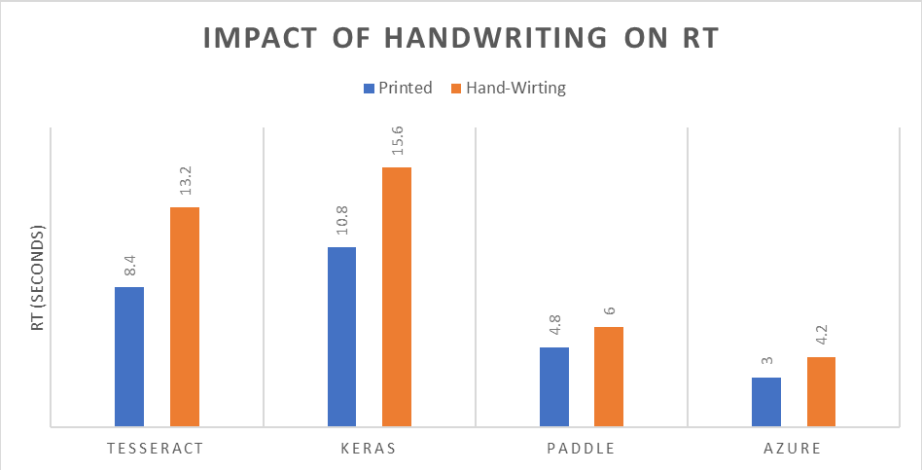


Figure 17: Impact of Handwriting on RT

In terms of specific OCR engines, Azure OCR demonstrates superior performance in handwriting recognition. On the other hand, Tesseract and Keras show a larger decrease in efficiency. The effectiveness of OCR engines largely depends on the datasets they were trained on. If Tesseract and Keras were trained predominantly on printed text datasets, their algorithms might not be well-tuned to recognize the nuances of handwritten text.

4.6.5 Impact of Complex Layout

Recognition Performance In the context of the FUNSD dataset, which comprises complex, real-world scanned forms, the performance metrics of OCR engines like Tesseract and Paddle indicate specific tendencies. Tesseract and Keras, exhibiting high recall but low precision, suggest a tendency to overgeneralize. As Figure 18 shows, this means they identify the most relevant text (high recall) but also incorrectly classify irrelevant elements as text (low precision), resulting in numerous false positives. Tesseract's performance, in particular, is affected by image quality; it requires a minimum of 300 dpi to function optimally. Lower-quality images, such as the ones in the FUNSD dataset, lead to the misidentification of noise as text.

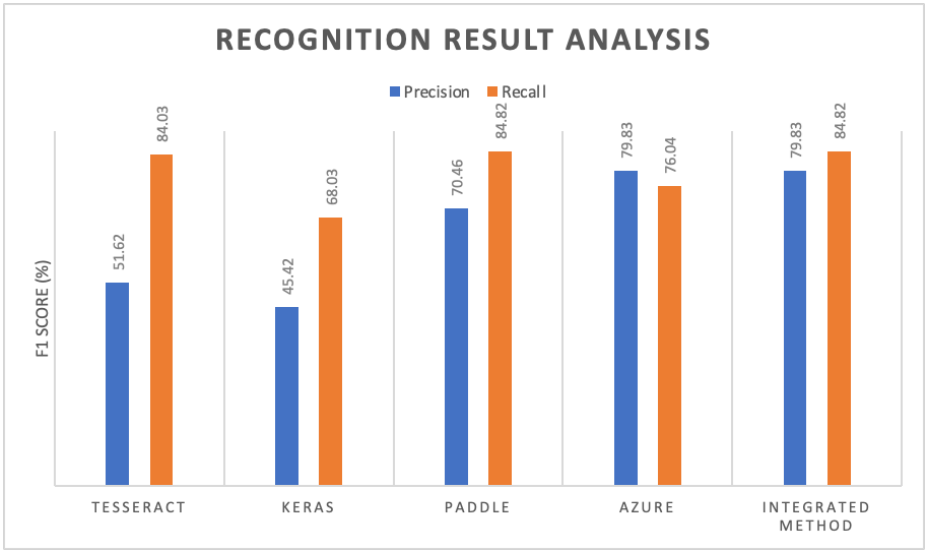


Figure 18: Recognition Result Analysis

In contrast, Azure demonstrates high precision with slightly lower recall. This implies that while it accurately identifies relevant text (high precision), it may miss some relevant instances (lower recall). Azure's approach tends to be more selective, focusing on accuracy at the expense of comprehensiveness. This difference in performance characteristics between Tesseract/Keras/Paddle and Azure highlights the variance in OCR technologies' capabilities when dealing with complex documents like those in the FUNSD dataset.

Running Time per Image

The increased running time for OCR engines on the FUNSD dataset, as observed in Figure 19, can be attributed to the complex layouts and lower image quality prevalent in this dataset. OCR technology is sensitive to image quality, and when faced with low-resolution, poorly lit, or low-contrast images, its text recognition capability is significantly diminished. Additionally, OCR struggles with documents that have complex or unconventional layouts, such as multiple columns or embedded graphical elements. These complexities require more advanced processing, which can increase the time needed for accurate text recognition. Therefore, the combination of intricate layouts and suboptimal image quality in the FUNSD dataset naturally leads to longer processing times per image compared to the simpler layouts of the IAM dataset.

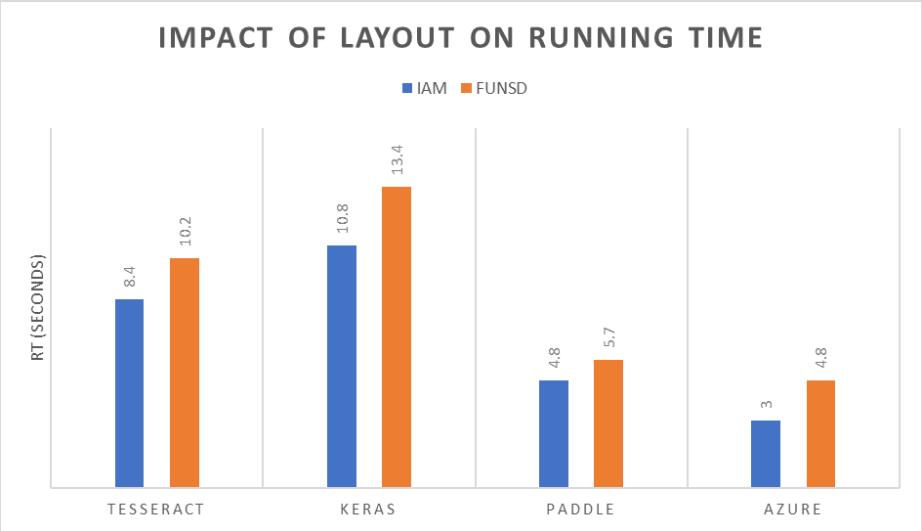


Figure 19: Impact of Layout on RT

5 Conclusion

5.1 Observation

To conclude the experimental findings from previous section, the empirical analysis continuously positions the Integrated System as the superior method across various settings, demonstrating exceptional robustness and consistently high performance across various preprocessing conditions and datasets. In a closely contended second place, and the biggest contributor to the Integrated System, Azure OCR excels in terms of processing speed and precision, exhibiting particular adeptness in deciphering complex handwriting scenarios, a testament to its resilience against fluctuations in image quality.

We can conclude that even though the gap of F1 score between OCR models may be not as distinctive, the more complex dataset, such as FUNSD, reveales the larger gap between recall and precision. Such difference pattern was not clearly showed in IAM dataset possibly due to the simple layout. These performance metrics measuring false positives and false negatives shed light on the real-world applicatio of OCRs. To be specific, Tesseract, while achieving high recall, does so at the expense of precision, indicating a higher likelihood for overgeneralization that manifests in a higher rate of false positives. This behavior may be beneficial for use cases where capturing the totality of text is paramount, despite the risk of inaccuracies. Conversely, Azure's predilection for precision, even at the slight sacrifice of recall, underscores a more conservative recognition strategy that could be favored in applications where accuracy is critical.

The impact of preprocessing techniques such as Otsu's thresholding is notably significant, especially for Tesseract and Keras, which marked performance enhancements by reducing the complexity of the input images. However, this improvement is not universally applicable, as evidenced by the Integrated Method's consistent performance irrespective of such preprocessing, indicating a possible intrinsic optimization within its composite algorithms.

The addition of autocorrect features introduces a modest enhancement in reducing the Word Error Rate (WER), yet it unevitably introduces extra processing time for all OCR models by roughly 1 second per image. This increment suggests a diminishing return for engines already equipped with advanced error-correction capabilities, pointing to a potential redundancy of the autocorrect step in such sophisticated systems.

In conclusion, the investigation underscores the imperative for tailored OCR solutions, capable of adapting to the specific challenges posed by diverse text types and document intricacies. The Integrated Method exemplifies the efficacy of a synergistic approach, leveraging the collective strengths of multiple OCR systems to elevate overall text recognition performance. Looking ahead, there is a compelling argument for the integration of context-aware post-processing techniques, such as advanced Natural Language Processing (NLP), to further refine OCR accuracy and efficiency. This study lays the groundwork for such future explorations, potentially catalyzing advancements in the field of OCR technology.

5.2 Limitation

The result of comparative analysis and the choice of dataset may not completely reveal or indicate the superiority of one OCR engine to another. Due to the limitation of the project scope, no fine-tuning of weights or any customizations were done to the OCR engines selected which potentially could cause bias to OCR algorithms such as Keras.

The integrated post-processing approach, designed to enhance OCR output by amalgamating results from multiple engines, achieves high accuracy but at the expense of increased processing time and computational resources. This system waits for all OCR engines to complete their tasks before it commences its analysis, which, while beneficial for accuracy, introduces delays that may not be suitable for time-sensitive applications.

The scoring system's complexity adds computational overhead, particularly when handling larger, more complex datasets. Additionally, the scoring mechanism, based on word-level accuracy, may lead to context ignorance, potentially preventing the system to understand context of a sentence and thus generate a high confident but low accuracy sentences. Integration challenges also arise from the assumption that outputs from different OCR engines are compatible, which may not hold across different scales or confidence measures, potentially complicating the validation process.

A notable limitation is the system's underperformance in recognizing handwriting, heavily dependent on Azure's adoption of Intelligent Recognition (IR) techniques. The overreliance on a single OCR component exposes a vulnerability in the integrated system. Incorporating Intelligent Character Recognition (ICR) may address this shortfall by providing nuanced recognition of handwritten text, adapting to various styles, and improving through machine learning over time. Unlike traditional OCR, ICR does not rely on template matching and instead utilizes AI and neural networks to learn from the data it processes, continually refining its ability to decipher handwriting.

To maintain professional standards and conciseness, future iterations of this system could benefit from an assessment of each OCR engine's contribution to the overall performance and a more dynamic scoring system that accounts for the context of the text, potentially through the integration of advanced NLP techniques. This would not only streamline the process but also enhance the system's contextual understanding, providing a more robust solution for real-world applications.

5.3 Future Work

To stress the limitations mentioned before, four potential future paths are listed below including fine-tunning OCR weights for a more specialized performance, incorporating parallel OCR for a more efficient process, utilizing Contextual Analysis such as NLP as a post-processing technique to improve the OCR performance, and focusing on Intelligent Character Recognition (ICR) for hand-writing and unstructured data forms.

5.3.1 OCR Fine-tuning

It has been proven useful to fine-tune OCR models using transfer-learning for specific tasks. Although some OCR engines may not be available for fine-tuning, certain OCRs especially open-source OCR engines will benefit a lot from task-specific transfer learning by adopting a pre-trained weight while train and evaluate on prepared dataset. The difficulties of such should not be neglected, for instance, the preperation of datasets is complex consider different OCR engines require different forms of annotations or image sizes [8].

5.3.2 Parallel OCR:

To address computational overheads, parallel processing techniques could be employed to handle the OCR outputs simultaneously, reducing processing time in the future studies. For implementing parallel processing in OCR systems, a map-reduce approach can be adopted. This approach involves decomposing OCR tasks and distributing them across multiple processors to be executed concurrently, leading to a substantial reduction in time-to-completion for image recognition tasks. Specific patterns like voting, predictive selection, tessellation, and speculative parallelism can be applied to improve accuracy and are robust even with sparse or non-Gaussian training data [19].

5.3.3 Contextual Analysis:

More sophisticated post-processing technique can be discussed in the future such as incorporating NLP techniques. It could enable the system to consider the context of words, reducing the chance of selecting out-of-context words that may have a high confidence score and thus generate a more reasonable results especially for localities, technical terms or names.

Techniques like n-grams, noun phrase extraction, and theme extraction with relevancy scoring are central to contextual analysis in NLP [16]. N-grams can capture key phrases and themes that provide context, while noun phrase extraction focuses on identifying relevant parts of speech patterns. Theme extraction further refines this by assigning relevancy scores to noun phrases, allowing for the identification of the most contextually significant terms.

5.3.4 Intelligent Character Recognition:

Intelligent Character Recognition, known as ICR, is a technique that addresses and surpasses the limitations commonly affects OCR systems, particularly in the processing of handwritten texts and other unstructured data forms. It harnesses the collective strength of artificial intelligence (AI), machine learning (ML), and natural language processing (NLP) to understand their context and variations in handwriting styles [10] and not just pattern recogniton in OCR.

What is interesting in ICR's technical aspect is its self-learning capabilities, where it employs neural networks that improve the recognition database with each new piece of data processed [1]. This continuous learning aspect allows ICR systems to adapt to new handwriting styles, making them increasingly accurate over time. Unlike OCR, which is highly efficient with printed text, ICR excels in interpreting a wide range of handwriting, including cursive and informal styles, without requiring predefined templates or formats.

ICR also integrates contextual understanding, a feature that enables it to decipher the meaning and context of words within a document, further refining its accuracy. This is particularly beneficial in situations where characters may be ambiguous or difficult to discern based on appearance alone [11]. By analyzing the shapes and features of characters rather than relying solely on pixel matching, ICR reduces errors and improves precision.

ICR has been successfully deployed in various industries for tasks such as automating data entry from handwritten forms, streamlining invoice and receipt processing, and accurately capturing information from handwritten notes and annotations. Its versatility and multilingual support extend its use to a broad spectrum of document types across different languages.

References

[1] Saptak Chakraborthy. Intelligent character recognition vs ocr: Overcoming ocr limitations in document processing, Nov 2023.

[2] Honglie Chen, Weidi Xie, Andrea Vedaldi, and Andrew Zisserman. Autocorrect: Deep inductive alignment of noisy geometric annotations. *arXiv preprint arXiv:1908.05263*, 2019.

[3] 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, 2020.

[4] Jean-Philippe Thiran Guillaume Jaume, Hazim Kemal Ekenel. Funsd: A dataset for form understanding in noisy scanned documents. In *Accepted to ICDAR-OST*, 2019.

[5] Maya R Gupta, Nathaniel P Jacobson, and Eric K Garcia. Ocr binarization and image pre-processing for searching historical documents. *Pattern Recognition*, 40(2):389–397, 2007.

[6] Karez HAMAD and Mehmet KAYA. A detailed analysis of optical character recognition technology. *International Journal of Applied Mathematics Electronics and Computers*, (Special Issue-1):244–249, 2016.

[7] Santosh Kumar Henge and B. Rama. Comprative study with analysis of ocr algorithms and invention analysis of character recognition approched methodologies. In *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, pages 1–6, 2016.

[8] Aravind Inbasekaran, Rajesh Kumar Gnanasekaran, and Richard Marciano. Using transfer learning to contextually optimize optical character recognition (ocr) output and perform new feature extraction on a digitized cultural and historical dataset. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 2224–2230. IEEE, 2021.

[9] Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. Funsd: A dataset for form understanding in noisy scanned documents. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, volume 2, pages 1–6, 2019.

[10] Renuka Kajale, Soubhik Das, and Paritosh Medhekar. Supervised machine learning in intelligent character recognition of handwritten and printed nameplate. In *2017 International Conference on Advances in Computing, Communication and Control (ICAC3)*, pages 1–5. IEEE, 2017.

[11] Yuliia Kniazieva. Intelligent character recognition, Aug 2023.

[12] Devashree Madhugiri. Extract text from images quickly using keras-ocr pipeline, Apr 2023.

[13] HR Mamatha and K Srikantamurthy. Morphological operations and projection profiles based segmentation of handwritten kannada document. *International Journal of Applied Information Systems (IJAIS)*, 4(5):13–19, 2012.

[14] U-V Marti and H Bunke. The iam-database: An english sentence database for off-line handwriting recognition. *International Journal on Document Analysis and Recognition*, 5:39–46, 2002.

[15] Andrew Morris, Viktoria Maier, and Phil Green. From wer and ril to mer and wil: improved evaluation measures for connected speech recognition. 01 2004.

[16] James Mutinda, Waweru Mwangi, and George Okeyo. Lexicon-pointed hybrid n-gram features extraction model (lenfem) for sentence level sentiment analysis. *Engineering Reports*, 3(8):e12374, 2021.

[17] Ram Sarkar, Nibaran Das, Subhadip Basu, Mahantapas Kundu, and Mita Nasipuri. Extraction of text lines from handwritten documents using piecewise water flow technique. *Journal of Intelligent Systems*, 23(3):245–260, 2014.

[18] Snehil Saxena, Sidharth Jain, Saurabh Tripathi, and Kapil Gupta. Comparative analysis of image segmentation techniques. In *Advances in Communication and Computational Technology: Select Proceedings of ICACCT 2019*, pages 317–331. Springer, 2021.

[19] Steven Simske. Parallel processing considerations for image recognition tasks. *HP Laboratories Technical Report*, 01 2011.

[20] Ray Smith. An overview of the tesseract ocr engine. In *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR)*, pages 629–633, 2007.

[21] William Ughetta and Brian W. Kernighan. The old bailey and ocr: Benchmarking aws, azure, and gcp with 180,000 page images. *Proceedings of the ACM Symposium on Document Engineering 2020*, 2020.

[22] William Ughetta and Brian W. Kernighan. The old bailey and ocr: Benchmarking aws, azure, and gcp with 180,000 page images. In *Proceedings of the ACM Symposium on Document Engineering 2020*, DocEng '20, New York, NY, USA, 2020. Association for Computing Machinery.

[23] J.P. Woodard and year = 1982 journal = Workshop on standardisation for speech I/O technology, Naval Air Development Center, Warminster, PA title = An information theoretic measure of speech recognition performance Nelson, J.T.

[24] Fangsheng Wu, Changan Zhu, Jinxiu Xu, Mohammed Wasim Bhatt, and Ashutosh Sharma. Research on image text recognition based on canny edge detection algorithm and k-means algorithm. *International Journal of System Assurance Engineering and Management*, pages 1–9, 2021.

[25] Zhao Xu, Xu Baojie, and Wu Guoxin. Canny edge detection based on open cv. In *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, pages 53–56, 2017.