Training a Tesseract module for GDR typewriter

Research Lab

Anu Antony - 220200551
Muhammad Kashan Khalid - 222100695
Sai Venkata Sahith Reddy Kadapaiahgari - 222201978
Aman Singla - 221202770

Universität Koblenz
Department für Informatik,
Institute for Web Science and Technologies

Supervisor(s)
Dr. Jens Dörpinghaus

March 14, 2024

Contents

Abstract

Digitizing vast archives of historical documents like those from the former German Democratic Republic (GDR) poses a significant challenge due to the prevalence of unique typewriter fonts. Standard Optical Character Recognition (OCR) tools like Tesseract often struggle with such unconventional font styles, hindering efficient and accurate text recognition [19]. This research investigates the development of a custom Tesseract module specifically designed to handle German texts written in GDR typewriter fonts. We achieve this by converting PDF documents containing GDR typewriter fonts into images and training a custom module using Tesseract v5.3.3 with "tessdata best" as the base model. By evaluating the performance of our module against standard Tesseract on unseen test documents, we demonstrate its superior accuracy in recognizing GDR typewriter fonts. This research offers a valuable tool for large-scale digitization of GDR archives, facilitating historical research and unlocking valuable information for future generations.

1 Introduction

The Bundesinstitut für Berufsbildung (BIBB) holds a vast archive of documents from the former German Democratic Republic (GDR), estimated to contain millions of pages [7]. These documents, filled with valuable historical and sociological data, currently exist in physical format, limiting their accessibility and hindering detailed analysis. Digitizing these archives would unlock immense research potential, but presents a significant hurdle: the prevalence of typewriter fonts, which pose difficulties for standard Optical Character Recognition (OCR) tools like Tesseract [18]).

Tesseract, a popular open-source OCR engine, struggles with historical documents due to factors like font variations, faded ink, and page imperfections. While recent advancements have improved its accuracy, typewriter fonts often present unique challenges, particularly those from specific regions like the GDR. Existing attempts to train Tesseract on historical documents have yielded mixed results, suggesting the need for a more specialized approach [13].

This research project aims to address this challenge by investigating the development of a custom Tesseract module specifically designed to handle German texts written in GDR typewriter fonts. By leveraging deep learning techniques and training the module on a comprehensive dataset of authentic GDR documents, we seek to achieve accurate and efficient text recognition, enabling the large-scale digitization of this historically significant archive.

This endeavor holds crucial implications for researchers and historians studying the GDR. Digitizing these documents would open doors to new avenues of exploration and analysis, allowing for quantitative and qualitative studies of topics ranging from social structures to economic trends. Additionally, the development of a specialized Tesseract module could contribute to the broader field of document digitization, offering a valuable tool for processing historical archives with unique font characteristics.

In the following sections, we will delve deeper into the specific challenges of recognizing GDR typewriter fonts, explore existing OCR technologies and their limitations, and outline the methodology for developing and evaluating our custom Tesseract module. Through this research, we aim to make a significant contribution to both preserving and unlocking the wealth of knowledge contained within the BIBB's GDR archives.

1.1 Motivation

Exploring GDR history remains a vital task, even decades after German reunification. Its influence continues to shape contemporary German identity. Ongoing research seeks to understand the historical, political, social, and economic complexities of the GDR era, often grappling with the sensitive nature of documents produced under a surveillance state. Digitizing old typewritten archives is crucial, yet presents challenges in preservation, access, and the ethical use of potentially personal information. We are deeply motivated to contribute to this effort, recognizing its potential to enrich our understanding of the past while navigating the complexities involved in making such history accessible.

1.2 Related Work

There are multiple researches conducted in the past related to handwritten, printed as well as typewritten datasets.
John A. et al [12] introduced a flexible OCR system for maintenance documents, countering the inconsistent performance of standard OCR tools, particularly with tabular data. By segmenting the document pages into seperate images and adding them to the OCR engine, the system achieved accurate results. It also provided granular control, enabling fine-tuning of image segmentation and the incorporation of custom regular expressions based on document schema. The scalable and adaptable nature of this approach makes easy to digitalize the documents, reducing labor costs and enhancing advanced analytics capabilities. Overall, it offers an efficient solution to OCR challenges in maintenance document processing.

Anand K. et al [11] evaluated various OCR tools on both handwritten and printed datasets, focusing on a self-made dataset derived from Complete Blood Count Reports. Amazon Textract outperformed Google Vision and PyTesseract, with preprocessing techniques improving overall model performance. The study concluded that printed datasets yield better results, attributing this to the consistent format of printed characters. It achieved a noteworthy 95% accuracy and a low Character Error Rate (CER) of 0.1082 for the CBC blood report dataset using Amazon Textract with preprocessing. The paper suggests that paid OCR services generally offer better results and reliability, but acknowledges that outcomes depend on the dataset and model training. The potential for manual training to surpass existing models is also highlighted.

Another research by Brennan N. et al [14] presented an inventive framework for historical document text recognition in collaboration with the Tesseract OCR system. Operating with labeled character images and a simplified character class file, the system eliminated the need for manual transcription by language experts. The framework, incorporating a training text generation mechanism, explored the impact of different character prototypes on model performance. With dissimilarity measures like Mean Squared Error and Jaccard-Needham-Dissimilarity optimization, various models were generated from selected sample subsets, allowing for reliable calculation of upper and lower bound performance models. Despite its simplicity, the method demonstrated a commendable error rate of approximately 15% on the "Narrenschiff" benchmark, surpassing standard synthetic Tesseract training with an error rate of approximately 27%. The paper concluded by highlighting the significant potential of this approach for recognizing text automatically in historical documents without ground-truth data.

2 Methodology

2.1 System Setup

To carry out this experiment, we configured few items on the machine and pre-processed the data. Let's delve more into those steps,

2.1.1 Installation of tesseract

To install the Tesseract, we followed several steps. The installation process includes downloading the executable file from the Tesseract official website. It depends on the type of operating system being used. We can install Tesseract and its developer tools on Ubuntu by simply executing, [4].

```
sudo apt install tesseract-ocr
sudo apt install libtesseract-dev
```

We installed Tesseract on MacOS, for that homebrew needs to be intalled. Homebrew is a package manager for MacOS that simplifies the installation of various software such as Git, Ruby, and Node. You can install homebrew directly from the source website. After installing homebrew, we can intall the tesseract by executing, [2].

```
brew install tesseract
```

This will install the tesseract in the machine with default **make** file version 4, but to train the model using tesseract, atleast GNU **make** with minimal version 4.2 [1] is required.

2.1.2 Installation of Make File:

As we discussed above, to train the model in tesseract we need atleast Make file 4.2 version. This can be done by running below command [5],

```
wget http://ftp.gnu.org/gnu/make/make-4.2.tar.gz
```

Before that we have to know what is make file, in Tesseract, a "makefile" is a file used with the make utility to simplify the compiling and building of the Tesseract OCR engine and its associated tools from source code. It consists of all the configurations and rules to train the model [1]. After installing the updated version of make, we used below command to unzip it[5],

```
tar -xzvf make-4.2.tar.gz
```

In the next step, run the below command to configure the make file [1],

```
./configure
```

Once./configure has successfully configured the build environment, the make command is used to turn the source code into executable binaries or libraries. It detects the presence or absence of optional program features and dependencies [10]. This consists of libraries, header files, and external applications required for certain capabilities. The./configure script frequently includes options and parameters that let users to modify the build process based on their preferences and needs. These options might involve enabling or removing specific features, defining installation locations, or selecting compiler optimizations. It reads the Makefile prepared by*./configure* and performs the compilation process as defined in the makefile. After this, we executed below command [5].

```
make
```

When the compilation process (make) is finished, *make install* is used to install the built binaries, libraries, and any other required files on your system. This command normally copies compiled files to the system folders provided in the *./configure* step. It may also do other activities, such as granting rights or modifying system configuration files [5],

```
make install
```

After that we configure the changes in **glob.c** file, which is inside Makefile of ***tesstrain*** repository. This repository should be downloaded in the local machine. The below sed command changes the word **`__alloca`** to **alloca** in **glob.c** file in order to make the code compatible according to system requirements.[5],

```
sed -i 's/__alloca/alloca/g' /app/src/tesstrain/
make-4.2/glob/glob.c
```

When the tesseract is completely installed with the updated configuration, we can now train the model. Before that, we processed the data files that are required to train the model. In the next section, we will get to know about how we converted the input documents into the required formats.

2.2 Training the Tesseract

2.2.1 Data Collection

From the universität koblenz cloud repository, total of 28 pdf files are taken. Those documents contained the printed text which is typed from GDR typewriter. Those 28 pdf files consisted 282 pages, each page is converted into individual png files.

2.2.2 Data Pre-Processing

In this section, we describe the data pre-processing methods used to prepare data files for subsequent analysis and model training. The goal of this pre-processing stage is to convert the PDF files into formats appropriate for text recognition and model training.

Conversion of PDF files to TIFF and gt.txt

- **TIFF:** It is abbrievated as Tagged Image File Format. TIFF is uncompressed image file. Which is widely used for various purposes like scanning, OCR, image editing. [8]. These file have **'.tif'** extension.
- **Ground Truth:** Ground truth files, often referred to as.gt.txt files, contain precise text that corresponds to the images used in OCR (Optical Character Recognition) training or evaluation. These files act as an anchor or "ground truth" against which the OCR engine's outputs is measured. During training, the OCR model learns to recognize text patterns through validating output with the ground-truth. [8].

Let's discuss the conversion process step by step

1. Given input documents are pdf files, so we converted the each page of pdf files to png format using 'pdf2images' library in python
2. Once we have png files, The next step is to convert those png files to 'tif' and 'gt.txt'. For this again we used same python library 'pytesseract' to extract the data as a single line text. The extracted data is utf-8 encoded.

The conversion preserved text quality and resolution, allowing for reliable OCR [8].

Box and LSTMF File Generation

Box file creation, Using the TIF image files generated in the previous step, we proceeded to create box files. These box files are required to train the tesseract model. It consists of extracted data and the coordinates of the box where the text is located. The file contains six labeled columns named left, right, bottom, top, symbol and page. In the box file, each character has its own line. The LSTM model can accept either individual character coordinates or an entire text line. Line-level coordinates are much easier to use, Box files are created by executing belo command, [8],

```
cd /home/fine_tune/train
tesseract train_invoice.tiff train_invoice --psm 4 -l
best/deu lstmbox
```

The first parameter is the input image file on which we will perform OCR and the second is the base name for the files that Tesseract will produce. It generates output files named $\text{train}_i nvoice.box..The language argument - l tells Tesseract to utilize the German model for OCR. The - psm argument instructs Tesseract to utilize page segmentation technique number four$[8].

LSTMF file creation, During tuning process, tesseract uses OCR in order to extract text from tif file and validates its projection against the coordinates and symbol in the box files. It does not depends on the tif and box files directly, but instead demands a lstmf file created from the two prior ones. To create 'lstmf' files we executed the below command, [8],

```
cd /home/fine_tune/train
tesseract train_invoice.tiff train_invoice lstm.train
```

2.2.3 Training the model

For training the model, we used already pretrained model from **tessdata_best**. This directory is expected to include the best-trained Tesseract OCR models. These models are tuned and trained to produce the most accurate text recognition results, making them the preferred option for many applications that use Tesseract for OCR purposes. This directory often contains language data files required by Tesseract to recognize text in multiple languages. To start training the model, Run the below command [1]

```
/usr/local/bin/make training MODEL_NAME=test
START_MODEL=deu PSM=7 TESSDATA=./tessdata_best-main
```

/usr/local/bin/make: This part of the command invokes the make utility, which compiles and builds the Tesseract OCR engine and its associated tools from source codecompiling of projects. In this case, it's being used to execute tasks related to training the tesseract OCR model [10].

training: This describes the task or aim that should be carried out. In your situation, it indicates that you desire to complete the training task [10].

MODEL_NAME=test: This is where you specify the name of the model being trained. In the above command we specified the name as test, we can give any name we want [5].

START_MODEL=deu: This parameter indicates the initial model that will serve as the foundation for training. "Deu" relates to the German language model. This means that your model will begin with the parameters and knowledge obtained from the German language model, which can be useful when training a model for German text recognition [5].

PSM=7: This configures the page segmentation mode (PSM) to 7. Page segmentation controls how Tesseract reads the image. Mode 7 handles the image as a single line of text. This parameter is used to determine how the input photos should be treated during training [1].

TESSDATA=./tessdata_best-main: This parameter provides the directory that holds the tessdata files required for training. These files provide linguistic data, trained models, and other resources required by Tesseract to conduct OCR. In this scenario, we are pointing to a directory called "tessdata_best-main". This repository should be inside our main repository **tesstrain-main** [3].

Before starting the training, there is one prerequisite. We have to store all the the files i.e, TIF, Ground truth, Box and LSTM in folder inside our tesstrain-main repository. The folder name should be "MODEL_NAME"-ground-truth. So that the model will train with the files from that folder on the top of tessdata_best model.

2.2.4 Testing the model

Till now the model is trained on top of tessdata_best-main, the next step we did is testing the model by providing an image file. To do so, We used the below command [8],

```
tesseract input_image.png output_text -l
/best/deu-latest -u
```

Let's delve into the above command,

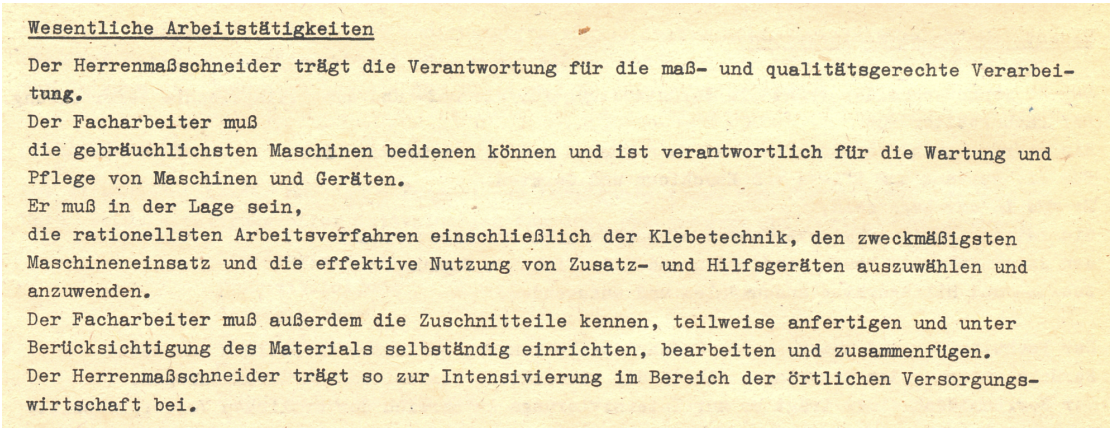
image_2.png: This parameter allows to input the image file on which to do OCR. In this scenario, the input image name is "image_2.png". Tesseract will attempt to identify the text in this image [8].

output_image_2.png: This parameter defines the base name for the output files created by Tesseract. In this scenario, the detected text will be saved in a file called output_image_2.txt.

-l: This flag refers to the language parameter. It guides tesseract to choose a particular language model for OCR [5].

/best/deu-latest-u: This defines which language model to be used. Here is the breakdown,

- **/best:** This specifies the directory path in which Tesseract's language data is stored. It's most likely a convention employed in your system to store language data [3].
- **deu-latest-u:** This defines which language model to be used. The term "deu" relates to the German language, while "latest" implies that the most recent version of the German language model should be employed. The "-u" at the end denotes that this is a unicharset language model, which contains a complete set of characters for the specified language [3].


Wesentliche Arbeitstätigkeiten

Der Herrenmaßschneider trägt die Verantwortung für die maß- und qualitätsgerechte Verarbeitung.
Der Facharbeiter muß
die gebräuchlichsten Maschinen bedienen können und ist verantwortlich für die Wartung und Pflege von Maschinen und Geräten.
Er muß in der Lage sein,
die rationellsten Arbeitsverfahren einschließlich der Klebetechnik, den zweckmäßigsten Maschineneinsatz und die effektive Nutzung von Zusatz- und Hilfsgeräten auszuwählen und anzuwenden.
Der Facharbeiter muß außerdem die Zuschnitteile kennen, teilweise anfertigen und unter Berücksichtigung des Materials selbständig einrichten, bearbeiten und zusammenfügen.
Der Herrenmaßschneider trägt so zur Intensivierung im Bereich der örtlichen Versorgungswirtschaft bei.

Figure 1: The above figure is the input document which contains printed text

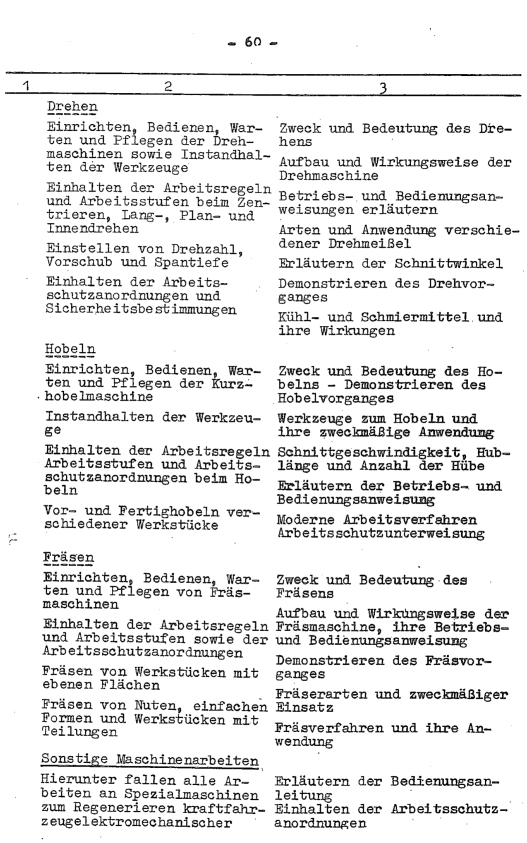
```
Wesentliche Arbeitstätigkeiten

Der Herrenmaßschneider trägt die Verantwortung für die maß- und qualitätsgerechte Verarbeitung. z
Der Facharbeiter muß
die gebräuchlichsten Maschinen bedienen können und ist verantwortlich für die Wartung und
Pflege von Maschinen und Geräten.

Er muß in der Lage sein,

die rationellsten Arbeitsverfahren einschließlich der Klebetechnik, den zweckmäßigsten
Maschineneinsatz und die effektive Nutzung von Zusatz- und Hilfsgeräten auszuwählen und
anzuwenden.

Der Facharbeiter muß außerdem die Zuschnitteile kennen, teilweise anfertigen und unter
Berücksichtigung des Materials selbständig einrichten, bearbeiten und zusammenfügen.

Der Herrenmaßschneider trägt so zur Intensivierung Am Bereich der örtlichen Versorgunge-
wirtschaft bei,.
```

Figure 2: The above is the output text file containing the recognized text.

Let's delve deep into above figures, Figure1 is the "input_image.png" which is the input image file consists of printed text. Our goal is to extract that text from the image.

Figure2 is the "output_text" which is an text file containing the extracted text from Figure1. Every line in this file corresponds to a segment of text extracted from the input_image file. This resultant text file can be further processed or examined as required, such as for indexing, searching, or data extraction.

Let us discuss about above two images in detail, Figure 3 is the image file where the text is aligned in two columns. This is a good example to see how our model will extract text from this image. Figure 4 is a text file which contains the extracted text from figure 3. We can see how the extracted text is aligned one by one in a single column.



- 60 -

1	2	3
	Drehen	
	Einrichten, Bedienen, Warten und Pflegen der Drehmaschinen sowie Instandhalten der Werkzeuge	Zweck und Bedeutung des Drehens
		Aufbau und Wirkungsweise der Drehmaschine
	Einhalten der Arbeitsregeln und Arbeitsstufen beim Zentrieren, Lang-, Plan- und Innendrehen	Betriebs- und Bedienungsanweisungen erläutern
		Arten und Anwendung verschiedener Drehmeißel
	Einstellen von Drehzahl, Vorschub und Spantiefe	Erläutern der Schnittwinkel
		Demonstrieren des Drehvorganges
	Einhalten der Arbeitsschutzanordnungen und Sicherheitsbestimmungen	Kühl- und Schmiermittel und ihre Wirkungen
	Hobeln	
	Einrichten, Bedienen, Warten und Pflegen der Kurzhobelmaschine	Zweck und Bedeutung des Hobelns - Demonstrieren des Hobelvorganges
	Instandhalten der Werkzeuge	Werkzeuge zum Hobeln und ihre zweckmäßige Anwendung
	Einhalten der Arbeitsregeln Arbeitsstufen und Arbeitsschutzanordnungen beim Hobeln	Schnittgeschwindigkeit, Hublänge und Anzahl der Hübe
		Erläutern der Betriebs- und Bedienungsanweisung
	Vor- und Fertighobeln verschiedener Werkstücke	Moderne Arbeitsverfahren Arbeitsschutzunterweisung
	Fräsen	
	Einrichten, Bedienen, Warten und Pflegen von Fräsmaschinen	Zweck und Bedeutung des Fräsens
	Einhalten der Arbeitsregeln und Arbeitsstufen sowie der Arbeitsschutzanordnungen	Aufbau und Wirkungsweise der Fräsmaschine, ihre Betriebs- und Bedienungsanweisung
	Fräsen von Werkstücken mit ebenen Flächen	Demonstrieren des Fräsvorganges
	Fräsen von Nuten, einfachen Formen und Werkstücken mit Teilungen	Fräserarten und zweckmäßiger Einsatz
		Fräsverfahren und ihre Anwendung
	Sonstige Maschinenarbeiten	
	Hierunter fallen alle Arbeiten an Spezialmaschinen zum Regenerieren kraftfahrzeugelektromechanischer	Erläutern der Bedienungsanleitung
		Einhalten der Arbeitsschutzanordnungen

(a) Figure 3

```
= 60 =

Einrichten, Bedienen, War-
ten und Pflegen der Dreh-
maschinen sowie Instandhal-
ten der Werkzeuge

Einhalten der Arbeitsregeln
und Arbeitsstufen beim Zen-
trieren, Lang-, Plan- und
Innendrehen

Einstellen von Drehzahl,
Vorschub und Spantiefe

Einhalten der Arbeits-
schutzanordnungen und
Sicherheitsbestimmungen

Hobeln

Einrichten, Bedienen, War-
ten und Pflegen der Kurz-
.hobelmaschine

Instandhalten der Werkzeu-
ge

Einhaiten der Arbeitsregeln
Arbeitsstufen und Arbeits-

schutzanordnungen beim Ho-
beln

Vor- und Fertighobeln ver-
schiedener Werkstücke
```

(b) Figure 4

Figure 3: The above two images shows the extraction of text from an image file and saving it in new text file

3 Comparison and Evaluation

3.1 Model Comparison

In this research, three OCR models are extensively evaluated: ABBYY FineReader, OCR4all, and a custom model built by us using Tesseract-OCR.

1. ABBYY FineReader

ABBYY FineReader is a renowned OCR tool known for its accuracy and robust feature set. It effectively transforms an image or PDF file into a text file that is both editable and searchable [16]. It excels in handling various document types and languages, making it a popular choice for commercial applications.

2. OCR4all

OCR4all is a relatively newer OCR model that gained attention for its open-source nature and adaptability. Developed by a collaborative community, OCR4all leverages machine learning techniques for enhanced accuracy and has shown promising results in diverse use cases. Another crucial factor involves it's ability to iteratively decrease the Character Error Rate by continuously training the models with more training text generated during the processing of a specific book [17].

3. Custom Tesseract-OCR Model

Tesseract-OCR, an open-source OCR engine, offers flexibility and customization. In this research, a tailored Tesseract-OCR model was developed, incorporating fine-tuned parameters and training on domain-specific datasets. This approach aims to harness the power of open-source tools while tailoring them to specific needs.

The following metrics define the similarities and differences between the three models:

Accuracy

- ABBYY FineReader is known for its high accuracy, especially in challenging scenarios with complex layouts and diverse fonts.
- OCR4all has demonstrated competitive accuracy, leveraging community contributions for continuous improvement.
- The Tesseract-OCR model, although customizable, may require extensive tuning to achieve optimal accuracy.

Language Support

- ABBYY FineReader supports many languages, making it great for use around global applications.
- OCR4all benefits from community-driven language expansions, providing flexibility in accommodating less common languages.
- Tesseract-OCR's language support is vast but may require manual integration for specific languages.

Customization

- ABBYY FineReader provides limited customization options, primarily relying on pre-trained models.
- OCR4all allows for extensive customization due to its open-source nature, enabling users to adapt the model to specific requirements.
- Tesseract-OCR model offers unparalleled customization, enabling fine-tuning of parameters for optimal performance in specialized domains.

While each OCR model has its strengths, the custom Tesseract-OCR model stands out for its adaptability and customizable nature. By tailoring the model to domain-specific requirements, it exhibits competitive accuracy and language support, surpassing the out-of-the-box capabilities of ABBYY FineReader and OCR4all.

3.2 Evaluation

The assessment of the models involved converting PDF files into images, That we have seen it above, Which was then converted into text file. Later this text file is processed by the Python scripts to measure the accuracy of the text extraction by comparing it with the image file. Subsequently, these output files were utilized as sources for evaluation, enabling the comparison and determination of similarity among the three models discussed in the preceding section.

3.2.1 Metrics

The evaluation of an Optical Character Recognition (OCR) system's accuracy hinges on the utilization of various metrics. These metrics serve as foundational criteria for comparative analysis, allowing for a comprehensive assessment of the system's performance. Through the careful examination of these metrics, researchers and practitioners can gain valuable insights into the OCR system's precision, thus facilitating informed comparisons and informed decision-making in the context of document processing and digitization. The key metrics used in our research are mentioned here:

1. **Character Error Rate (CER):**
    - **Definition:** The CER quantifies the precision of an OCR engine by measuring the percent of incorrect recognized characters to the sum total of characters in the ground truth. It is computed using the Levenshtein Distance, which represents the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform the recognized text into the ground truth. [15]
    - **Formula:** $CER = \frac{\text{Levenshtein Distance}}{\text{Total Ground Truth Characters}} \times 100\%$.
    - **Explanation:** A lower CER indicates a higher accuracy, reflecting fewer discrepancies between the recognized and ground truth characters based on Levenshtein Distance calculations.
2. **Word Error Rate (WER):**
    - **Definition:** The WER assesses the precision of word recognition in OCR, measuring the percent of wrongly recognized words to the count of words in the ground truth. It utilizes the Levenshtein Distance to determine the minimum number of edits needed to align the recognized and ground truth words. [15]
    - **Formula:** $WER = \frac{\text{Levenshtein Distance}}{\text{Total Ground Truth Words}} \times 100\%$.
    - **Explanation:** A lower WER indicates a higher accuracy in recognizing entire words, considering the Levenshtein Distance as a measure of similarity between the recognized and ground truth words.
3. **Word Accuracy Rate (WAR):**
    - **Definition:** Word Accuracy Rate (WAR) complements WER and represents the percentage of correctly recognized words. It is calculated as the complement of WER.
    - **Formula:** $WAR = 100\% - WER$.
    - **Explanation:** A higher WAR signifies better overall word recognition accuracy based on the Levenshtein Distance, providing a straightforward measure of the OCR system's success in accurately transcribing words.
4. **Character Accuracy Rate (CAR):**
    - **Definition:** Character Accuracy Rate (CAR) assesses the accuracy of individual characters across the entire document. It represents the percentage of correctly recognized characters compared to total characters in the ground truth, calculated using the Levenshtein Distance.
    - **Formula:** $CAR = \frac{\text{Levenshtein Distance}}{\text{Total Ground Truth Characters}} \times 100\%$.
    - **Explanation:** CAR offers a comprehensive evaluation of the OCR system's ability to accurately identify and transcribe individual characters, with Levenshtein Distance serving as a measure of character-level recognition performance.
5. **Flesch-Kincaid Grade:**

- **Definition:** It measures the readability of a text, based on the average number of words in a sentence to the average number of syllables in a word.
- **Formula:** $0.39 \times \left(\frac{\text{Total Words}}{\text{Total Sentences}}\right) + 11.8 \times \left(\frac{\text{Total Syllables}}{\text{Total Words}}\right) - 15.59$.
- **Explanation:** A lower Flesch-Kincaid Grade indicates easier readability, with lower values corresponding to simpler texts [6].

6. **Gunning Fog Index:**

- **Definition:** It evaluates the complexity of a text by considering the average length of sentence and the percentage of complex words (words with three or more syllables).
- **Formula:** $0.4 \times \left(\frac{\text{Total Words}}{\text{Total Sentences}} + 100 \times \frac{\text{Complex Words}}{\text{Total Words}}\right)$.
- **Explanation:** A lower Gunning Fog Index suggests simpler and more easily comprehensible text [9].

3.2.2 OCR Model Performance Comparison

1. **Document: ahrzeugschlosser 24219 Spezialisierungsrichtungen 01-09-1977.pdf**

Table 1: Document : Fahrzeugschlosser_24219 Spezialisierungsrichtungen 01-09-1977.pdf

Page	Metric	Our Model	ABBYY FineReader Model	OCR4all Model
12	CER	6.43%	6.36%	19.24%
	WER	6.54%	6.64%	20.88%
51	CER	10.15%	44.94%	71.61%
	WER	10.36%	47.85%	73.47%
87	CER	16.13%	47.51%	38.09%
	WER	17.52%	51.76%	42.43%

Table 1 demonstrates the analysis of the document "Fahrzeugschlosser_24219 Spezialisierungsrichtungen 01-09-1977.pdf" using evaluate-tessearct-ocr and evaluate_other_models python scripts present in the code repository. It shows the superior performance of our model in optical character recognition (OCR) tasks. Across three sampled pages our model consistently achieved the lowest character and word error rates. ABBYY FineReader's accuracy was more variable, while OCR4all consistently exhibited the highest error rates. These results suggest that our model may be a more robust OCR solution for challenging documents like the one analyzed.

Figure 4 and Table 2 highlights the similarity scores between three different OCR models: ABBYY FineReader, OCR4all, and Tesseract. This figure has been created using three_mode_results_comprison python script present in the code repository. ABBYY FineReader and Tesseract demonstrate the highest similarity (67.89%), suggesting their outputs are the most closely aligned. ABBYY FineReader and OCR4all exhibit the lowest similarity (44.39%), indicating more significant differences in their OCR results. OCR4all and Tesseract fall in between, with a moderate similarity score.

Table 3 compares the readability of three documents abby_fine_reader_result_fahr.docx, ocr4al_results_fahr.txt and tesseract_result_fahr.txt using the Flesch-Kincaid Grade and

Table 2: Model Similarity Comparisons

Comparison	Similarity
ABBYY FineReader vs OCR4all	44.39%
ABBYY FineReader vs Tesseract	67.89%
OCR4all vs Tesseract	45.30%

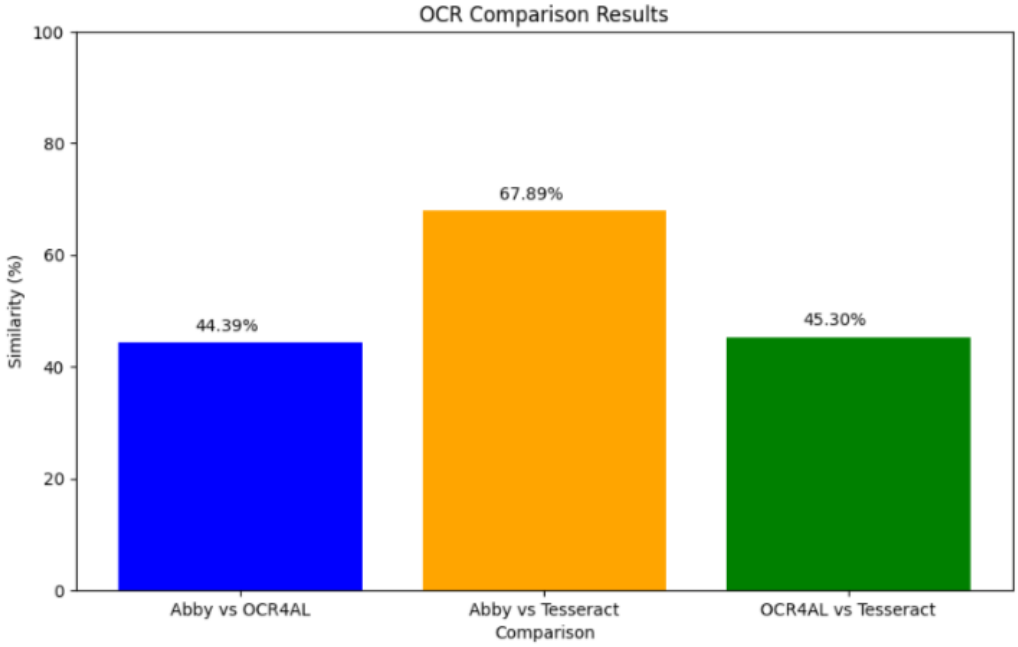


Figure 4: OCR Similarity Comparison

Table 3: Readability Score Comparisons

Document	**Flesch-Kincaid Grade**	**Gunning Fog Index**
abbyy_fine_reader_result_fahr.docx	-15.7	0.0
ocr4al_results_fahr.txt	28.1	29.16
tesseract_result_fahr.txt	16.8	14.51

Gunning Fog Index. These documents were generated as part of the results from ABBYY FineReader, OCR4all and Tesseract OCR models, and evaluated using results_readability_check python script. ABBYY FineReader consistently demonstrates the easiest reading experience, scoring the lowest across all metrics. Conversely, OCR4all scores the highest, indicating it's the most difficult to read Tesseract OCR exhibits moderate readability, with scores falling between the other two.

2. **Document: Kraftfahrzeugelektromechaniker_1850 (30210) 01-09-1976.pdf**

   Table 4 analyzes the performance of three OCR models ("Our Model", "ABBYY FineReader", and "OCR4all") on the document "Kraftfahrzeugelektromechaniker_1850 (30210) 01-09-1976.pdf" using evaluate-tessearct-ocr and evaluate_other_models python scripts present in the code repository. It uses four metrics: Character Error Rate (CER), Word Error Rate (WER), Word Accuracy Rate (WAR), and Character Accuracy Rate (CAR). The analysis focuses on three specific pages of the document. Across most pages and metrics, our model demonstrates the best performance, suggesting it is the most accurate OCR solution for this specific document type.

Table 5: Model Similarity Comparisons

Comparison	**Similarity**
ABBYY FineReader vs OCR4all	49.06%
ABBYY FineReader vs Tesseract	72.11%
OCR4all vs Tesseract	50.08%

Table 4: Document: Kraftfahrzeugelektromechaniker_1850 (30210) 01-09-1976.pdf

Page	Metric	Our Model	ABBYY FineReader Model	OCR4all Model
15	CER	5.85%	5.95%	17.70%
	WER	5.81%	6.08%	19.90%
	WAR	94.19%	93.92%	80.10%
	CAR	94.15%	94.05%	82.30%
38	CER	10.87%	48.90%	40.92%
	WER	11.16%	49.45%	42.05%
	WAR	88.84%	50.55%	57.95%
	CAR	89.13%	51.10%	59.08%
81	CER	10.48%	22.34%	66.27%
	WER	10.48%	22.59%	67.34%
	WAR	89.52%	77.41%	32.66%
	CAR	89.52%	77.66%	33.73%

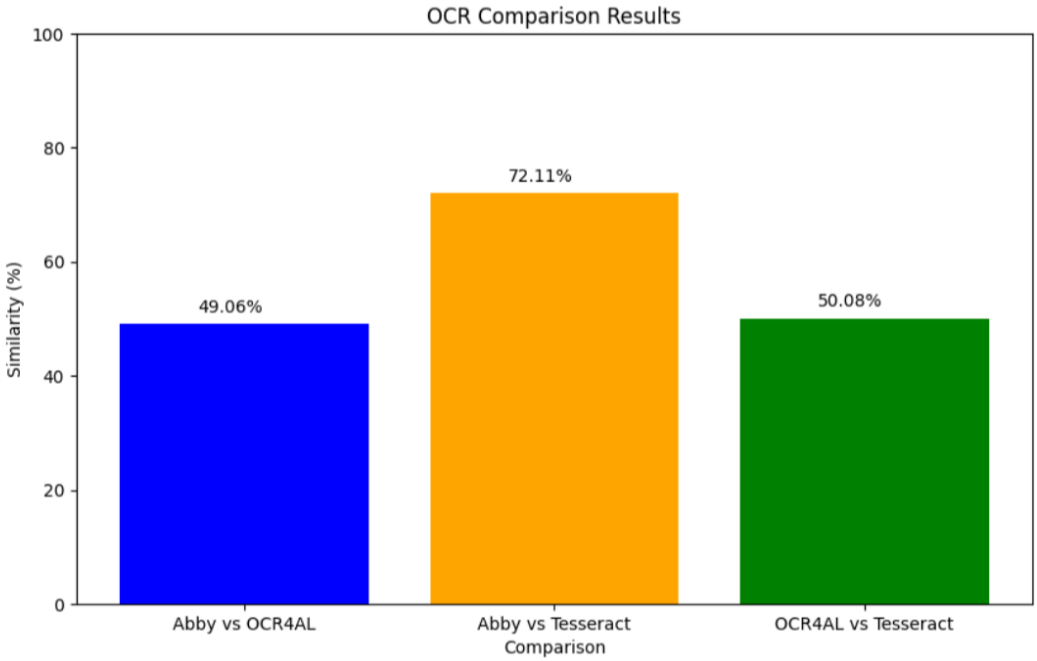


Figure 5: OCR Similarity Comparison

Figure 5 and Table 5 reveals the similarity scores between three OCR models: ABBYY FineReader, OCR4all, and Tesseract. This figure has been created using three_mode_results_comprison python script present in the code repository. The results indicate that ABBYY FineReader and Tesseract have the most similar outputs (72.11% similarity), suggesting their results align closely. The lowest similarity exists between ABBYY FineReader and OCR4all (49.06%), indicating greater differences in how these models process text. OCR4all and Tesseract exhibit a moderate level of similarity.

Table 6: Readability Score Comparisons

Document	Flesch-Kincaid Grade	Gunning Fog Index
abby_fine_reader_result_kraft.docx	-15.7	0.0
ocr4al_results_kraft.txt	24.3	26.71
tesseract_result_kraft.txt	23.8	22.93

Table 6 compares the readability of three documents abby_fine_reader_result_kraft.docx, ocr4al_results_kraft.txt and tesseract_result_kraft.txt using the Flesch-Kincaid Grade and Gunning Fog Index which assess the ease of reading. These documents were generated as part of the results from ABBYY FineReader, OCR4all and Tesseract OCR models, and evaluated using results_readability_check python script. ABBYY FineReader scores significantly lower on both metrics, suggesting it is the easiest to read. In contrast, OCR4all has the highest scores, indicating it's the most difficult to read. The readability of Tessearact OCR falls between the other two models.

4 Conclusion and Discussion

In this study, we set out to train a Tesseract OCR model that was specifically designed to recognize text in the domain of GDR typewriter documents. To achieve best performance, the training procedure required rigorous setup, which included system configuration, data preprocessing, and model training. The system setup entailed installing Tesseract and its dependencies, as well as the tools required for model compilation and training. Data pretreatment was methodically completed, including the conversion of PDF files to TIFF format and the creation of ground truth files (.gt.txt). Additionally, box and LSTMF files were created to provide proper training data for the model. The model was trained on the available data, beginning with the German language model and applying page segmentation techniques appropriate for the job.

To evaluate the effectiveness of our trained model, we ran a thorough study, comparing it to two popular OCR solutions: ABBYY FineReader and OCR4all. The evaluation metrics were character error rate (CER), word error rate (WER), character accuracy rate (CAR) and word accuracy rate (WAR). Our model produced competitive results in terms of CER, WER, CAR and WAR, proving its ability to recognize text in the given documents. A comparison with the ABBYY FineReader and OCR4all models revealed particular findings, such as comparing accuracy, mistake rates, and so on. When compared to other two models, Tesseract OCR model achieved great results with less CER, WER and more. For page 51 of document "ahrzeugschlosser 24219 Spezialisierungsrichtungen 01-09-1977.pdf", the WER for our model is far more good when compared to other two models. There are few limitations we faced while doing this research,

- The amount of documents available for training were limited.
- By looking the above metrics for ABBYY FineReader and OCR4all, Our model got greater results. That does not mean those two models are giving bad results. We did not have original ground truth files for ABBYY FineReader and OCR4all, So we created our custom ground truth files to calculate evaluation metrics. If we get the original ground truth for those two models, Then there will more accurate comparision with our model.

In future, For better results the number of documents could be increased for training, Similarly other OCR models could be tested against our custom tesseract model for better comparision and evaluation. Finally, our trained Tesseract OCR model produces promising results in text recognition. While additional optimization and fine-tuning may be required, the model's performance is comparable to proven commercial solutions such as ABBYY FineReader and OCR4all. Moving ahead, further refining and validation of the model will be required to realize its full potential in real-world applications.

References

[1] https://tesseract-ocr.github.io/tessdoc/tess4/trainingtesseract-4.00.html.

[2] Install tesseract on mac, https://formulae.brew.sh/formula/tesseractl.

[3] Tessdata-best, https://github.com/tesseract-ocr/tessdata_best.

[4] Tesseract installation, https://tesseract-ocr.github.io/tessdoc/installation.html.

[5] Training with tesseract, https://github.com/tesseract-ocr/tesstrain.

[6] R. Flesch. A new readability yardstick. *Journal of Applied Psychology*, 32(3):221–233, 1948.

[7] B. für Berufsbildung (BIBB). Bundesinstitut für berufsbildung, 2023.

[8] D. Gontcharov. Fine-tuning tesseract ocr for german invoices, 2020.

[9] R. Gunning. *The Technique of Clear Writing*. McGraw-Hill, 1971.

[10] S. Idrees and H. Hassani. Exploiting script similarities to compensate for the large amount of data in training tesseract lstm: Towards kurdish ocr. *Applied Sciences*, 11(20), 2021.

[11] A. Kumar, P. Singh, and K. Lata. Comparative study of different optical character recognition models on handwritten and printed medical reports. In *2023 International Conference on Innovative Data Communication Technologies and Application (ICIDCA)*, pages 581–586, 2023.

[12] J. A. Labarga, A. Singh, and V. Z. Moffitt. An extensible system for optical character recognition of maintenance documents.

[13] M. Mühling, M. Meister, N. Korfhage, J. Wehling, A. Hörth, R. Ewerth, and B. Freisleben. Content-based video retrieval in historical collections of the german broadcasting archive, Feb 2017.

[14] B. Nunamaker, S. S. Bukhari, D. Borth, and A. Dengel. A tesseract-based ocr framework for historical documents lacking ground-truth text. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3269–3273, 2016.

[15] N. Patwardhan, S. Marrone, and C. Sansone. Transformers in the real world: A survey on nlp applications. *Information*, 14(4), 2023.

[16] C. Purna Vithlani. Comparative study of character recognition tools. *International Journal of Computer Applications*, 118(9):31–36, May 2015.

[17] C. Reul, D. Christ, A. Hartelt, N. Balbach, M. Wehner, U. Springmann, C. Wick, C. Grundig, A. Büttner, and F. Puppe. Ocr4all—an open-source tool providing a (semi-)automatic ocr workflow for historical printings. *Applied Sciences*, 9(22), 2019.

[18] R. Smith. An overview of the tesseract ocr engine. In *Sixth International Conference on Document Image Analysis for Libraries (DIA)*, pages 629–637. IEEE, 2007.

[19] P. Tripathi. Major limitations of ocr technology and how idp systems overcome them, Dec 2023.