Fine-Tuning an Arabic OCR Model using Tesseract 5.0

Omar Samir, Youssef Waleed, Ibrahim Ahmed, Omar El Gohary
Mayar Emam, Farida Fawzy, Malak Emam, Shaden El Naggar
Faculty of Computing and Information Sciences
Egypt University of Informatics, New Administrative Capital

Supervised by: Dr. Mohamed Taher Al-Refaie

***Abstract*— Optical Character Recognition (OCR) is a crucial technology for the digital processing and preservation of textual information. While significant progress has been made in OCR for commonly used languages, the recognition of Arabic script poses unique challenges due to its complex linguistic characteristics. This research paper presents a comprehensive approach to collecting data and training an Arabic OCR system using the latest version of the Tesseract OCR engine 5.0 [1]. Integrating the customized Tesseract 5.0 model, the study achieved high accuracy in text recognition, as measured by character error rate (CER) and word error rate (WER). The findings highlight the complexities of the Arabic language, including diacritics and ligatures, and discuss the limitations of the current approach. This study contributes to the advancement of optical character recognition, providing a robust Arabic OCR solution that can be leveraged in various applications, from digitizing historical documents to automating text extraction.**

I. INTRODUCTION

Optical Character Recognition (OCR) technology has significantly evolved, enabling the conversion of various types of documents, such as scanned paper documents, PDFs, or images captured by a digital camera, into editable and searchable data. While OCR systems have achieved high levels of accuracy for many widely used languages, the development of effective OCR for the Arabic language remains a challenging endeavor. Arabic script is inherently complex due to its cursive nature, contextual character shaping, and the use of diacritics that alter the meaning and pronunciation of words.

This research focuses on addressing these challenges by leveraging Tesseract 5.0, the latest version of the widely adopted open-source OCR engine. Tesseract 5.0 incorporates numerous enhancements over its predecessors, offering improved recognition accuracy and expanded language support. The primary objective of this study is to train an Arabic OCR system using Tesseract 5.0, aiming to achieve high accuracy in text recognition and to contribute to the broader field of OCR technology.

II. KEY AREAS OF FOCUS

Our research encompasses several key areas essential to achieving our objectives. These areas include data collection and preprocessing, dataset composition, data augmentation, training and evaluation, challenges and limitations, results, and future directions. Recognizing the pivotal role of high-quality and diverse training data in OCR system success, we emphasize data collection and preprocessing as foundational steps.

III. METHODOLOGY

A. Data Collection and Preprocessing

The success of any OCR system is heavily dependent on the quality and diversity of the training data. For this study, we focused on building a comprehensive Arabic dataset to support the development of the Tesseract 5.0-based OCR model **[1]**. We leveraged Hugging Face to collect a large corpus of Arabic text data from various online sources, including websites, books, and social media platforms. This initial dataset consisted of over 150 million Arabic words, covering a wide range of topics and genres. To ensure the quality and consistency of the data, we implemented a series of preprocessing steps, including removing any non-Arabic characters, normalizing the text, and handling common issues such as inconsistent formatting.

B. Dataset Composition

We decided to limit our first training data to about 1 million Arabic sentences and proceeded to segment the Arabic sentences into different length categories, ranging from 0 to 10 words per sentence. This was done because Tesseract primarily accepts training on individual text lines rather than entire pages. The training process involves feeding Tesseract with images containing single lines of text along with their corresponding ground truth text.

We have also paid close attention to the inclusion of Arabic text with and without diacritical marks (tashkeel) in the dataset. This was a crucial consideration, as the presence of tashkeel can significantly impact the performance of Arabic OCR systems.

We split the initial 1 million Arabic sentences into two distinct subsets:

- 60% of the sentences with no diacritical marks (tashkeel-free)
- 40% of the sentences with diacritical marks (tashkeel-included)

This deliberate partitioning of the dataset was intended to ensure that the OCR model would be trained on a balanced representation of Arabic text, allowing it to handle both tashkeel-free and tashkeel-included scenarios effectively. We hypothesized that by exposing the model to a substantial amount of tashkeel-free data, it would be better equipped to handle the recognition of Arabic text in real-world applications, where tashkeel may often be omitted. Conversely, the inclusion of tashkeel-included data would enable the model to learn the nuances of diacritical marks and improve its overall accuracy in handling fully vocalized Arabic text.

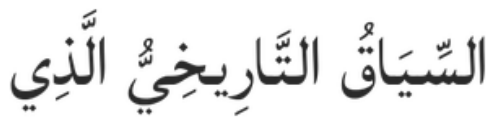
C. Image Generation

A crucial aspect of the data preparation process for this study was the generation of images from the preprocessed Arabic sentences. To achieve this, we leveraged the Text Recognition Data Generator (TRDG) library [2], which is a popular open-source repository hosted on GitHub and a powerful tool that provides a convenient way to generate synthetic text images for training OCR models.

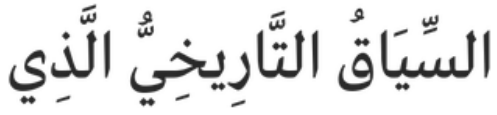
The TRDG library [2] provided us with the flexibility to generate high-quality images from the Arabic text in a programmatic and scalable manner. This was particularly important given the size of the dataset, which consisted of 1 million preprocessed Arabic sentences.

We selected three distinct Arabic font families to ensure a diverse representation of the Arabic script in the generated images and distributed them over the dataset in the following percentages:

1. Baghdad (35%): A traditional Arabic font with distinctive character shapes and stroke patterns.



2. Bahij (30%): A more modern Arabic font with a refined and elegant aesthetic.



3. Rakkas (35%): A calligraphic-inspired Arabic font that captures the artistic elements of the script.

By incorporating these three font choices, we aimed to expose the OCR model to Arabic script variations, enhancing its ability to generalize and perform well on diverse real-world documents.

We ensured that the generated images and their corresponding ground truth text were formatted in the file types commonly used for training Tesseract. We produced the data in .tif image format and .gt.txt ground truth text format. This attention to file format compatibility was a critical step in preparing the training data.

During the image generation process, we deliberately did not apply any additional noise or distortions to the synthetic images. The decision to forgo noise introduction at this stage was intentional, as we wanted to establish a clean and high-quality baseline dataset for the initial training of the OCR model.

This approach allowed us to assess the inherent capabilities of the Tesseract 5.0 OCR engine **[1]** in recognizing Arabic text, without the potential confounding factors introduced by extensive data augmentation. The inclusion of noise and other augmentation techniques would be crucial for improving the model's robustness and performance in later iterations of the research, which we started researching and will address in the next section. However, we chose to first focus on building a strong foundation by training the model on the clean and diverse set of Arabic sentence images generated using the TRDG library [2].

IV. DATA AUGMENTATION

As a crucial component of our ongoing research into training an Arabic OCR model, we have explored advanced data augmentation techniques to enhance the robustness and generalizability of our OCR models in the future. Data augmentation is a powerful strategy that can significantly improve the performance of machine learning models, especially in domains like computer vision where the availability of large, diverse training datasets is often a challenge.

A. Data Augmentation Methods

To enhance the robustness of our OCR model in the future, we researched several data augmentation techniques aimed at simulating real-world variations and imperfections in text images. We will be expanding the breadth and variability of our training data beyond the 1 million preprocessed text samples we had previously generated by applying carefully designed data augmentation methods, to expose our models to a wider range of realistic scenarios and variations that they may encounter in real-world Arabic document analysis tasks. This approach is critical for improving the model's ability to generalize across diverse and noisy input data in future versions.

1. Noise Injection

A. TRDG Library:

- Skew: Introduces random angular distortions to simulate misaligned text.
- Blur: Applies Gaussian blur to mimic out-of-focus images.
- Distortion: Warps the text to create irregular shapes and deformations.
- Background Image: Superimposes text on varied background images to replicate real-world conditions.

B. Text Types:

- Normal Text: Standard text without additional distortions.
- Dilate Text: Expands the text characters to simulate bold or smudged writing.
- Erosion Text: Erodes the text characters to mimic faded or worn-out text.

C. Custom Functions:

- Gaussian Noise: Adds random Gaussian noise to the images.
- Random Shadow: Introduces shadows to mimic lighting inconsistencies.
- Random Rain: Simulates rain streaks on the text images.
- Pixel Dropout: Randomly removes pixels to create text gaps or degradation.

In the future, we plan to augment our data significantly, incorporating various forms of noise, backgrounds, and more sentences with diacritics (tashkeel) to create a more robust dataset.

The rationale behind these data augmentation methods is to simulate a wide range of real-world conditions, thereby improving the OCR model's robustness and generalization capabilities. By exposing the model to diverse text variations and noise types, we aim to enhance its performance in recognizing Arabic text under various circumstances.

While these methods have shown promise, they have not yet been fully tested and validated. Future research will focus on systematically evaluating the impact of each augmentation technique on OCR accuracy, further refining the dataset to continually improve model performance.

V. ABOUT TESSERACT OCR

Tesseract OCR is an open-source OCR engine **[1]**, it extracts printed or written text from images and converts them to editable text. It was originally developed by Hewlett-Packard, and later on, the development was taken over by Google making it one of the most widely used OCR tools available today.

Two important features of Tesseract OCR are the OCR Engine Mode (OEM) and the Page Segmentation Mode (PSM), these two features are crucial in determining how text is recognized and segmented from the image.

A. OCR Engine Mode (OEM)

The OCR Engine Mode determines which type of OCR engine Tesseract should use when processing an image. There are several OEM options available, therefore we had to decide on what the best option is for our use case:

- OEM 0 - Legacy Engine Only: Uses the original Tesseract OCR engine.
- OEM 1 - Neural Network LSTM Engine Only: Uses the newer Long Short-Term Memory (LSTM) neural network-based engine.
- OEM 2 - Legacy + LSTM Engines: Combines both the legacy and the LSTM engines to leverage the strengths of both.
- OEM 3 - Default: Default, based on what is available.

We decided to use OEM 3 as we wouldn't need to test each mode, which would in turn save us more time and therefore allow us to invest more time and effort into fine-tuning the model for the best overall performance.

B. Page Segmentation Mode (PSM)

Page Segmentation Mode determines how Tesseract segments the image into text regions. There are several PSM options, each suitable for different types of document layouts, and therefore we had to decide on the most suitable option:

- PSM 0 - Orientation and Script Detection (OSD) Only.
- PSM 1 - Automatic Page Segmentation with OSD.
- PSM 2 - Automatic Page Segmentation but No OSD or OCR.
- PSM 3 - Fully Automatic Page Segmentation, but No OSD.
- PSM 4 - Assume a Single Column of Text of Variable Sizes
- PSM 5 - Assume a Single Uniform Block of Vertically Aligned Text
- PSM 6 - Assume a Single Uniform Block of Text.
- PSM 7 - Treat the Image as a Single Text Line.
- PSM 8 - Treat the Image as a Single Word.
- PSM 9 - Treat the Image as a Single Word in a Circle.
- PSM 10 - Treat the Image as a Single Character.
- PSM 11 - Sparse Text.
- PSM 12 - Sparse Text with OSD.
- PSM 13 - Raw Line: Processes the image as a raw line of text.

We decided to use Page Segmentation Mode 13 (PSM 13) for our project. This mode processes the image as a raw line of text. We found that this was the most suitable way for our project as we split the text into segments of 1-10 words and then generate images for each segment, (this was previously explained in more detail in the Dataset composition section). Therefore processing the image as one line of raw text was the best option.

VI. TRAINING

When starting off with training using Tesseract OCR we had to decide on a way of training, Tesseract has 3 ways of training as follows:

- Fine-tune Starting with an existing trained language, train on your specific additional data. This may work for problems that are close to the existing training data, but different in some subtle way, like a particularly unusual font. It may work with even a small amount of training data.
- Cut off the top layer (or some arbitrary number of layers) from the network and retrain a new top layer using the new data. If fine-tuning doesn't work, this is most likely the next best option. Cutting off the top layer could still work for training a completely new language or script if you start with the most similar-looking script.
- Retrain from scratch. This is a daunting task unless you have a very representative and sufficiently large training set for your problem. If not, you are likely to end up with an over-fitted network that does really well on the training data, but not on the actual data.

We decided to go with fine-tuning a model. We chose this method since training from scratch requires an enormous dataset and therefore would take days, weeks, or even months. To fine-tune, a pre-trained model is needed, the pre-trained model used was taken from the official Tesseract repository [2], this repository includes a list of pre-trained models for a range of different languages, we used the 'ara.traineddata' model, which is the Arabic pre-trained model. Next, we needed to decide on the best values for each parameter, The parameter values used were as follows:

- Maximum iterations: 400. This value was used as we found that using a value greater than 400 causes the model to overfit, especially if the dataset is small.
- Lang type: RTL. This value was used as we are fine-tuning an Arabic OCR model and the Arabic language is read from right to left, unlike English which is read from left to right.
- PSM: 13. Tesseract 5.0 requires a line of text as input, the value 13 indicates that the model will receive an image containing one line of text.

VII. EVALUATION

Upon fine-tuning our model, we had to move on to evaluating its performance to make sure that it met our desired accuracy and reliability standards. Evaluating the model involves assessing the model's ability to recognize the text and convert it to editable text correctly.

First, we had to find a dataset that we could use to evaluate the model and compare our results to the pre-trained model results, in our case we used the dataset used by Thomas Hegghammer[3] as we found that it is the most suitable dataset for our case. In this process, we also compared our results to those obtained by Thomas Hegghammer[3], who conducted a benchmarking experiment on OCR with Tesseract, Amazon Textract, and Google Document AI. Since our research focuses only on fine-tuning the Arabic model, we used the results from Tesseract and Google Document AI, as Amazon Textract was only used in English and not Arabic. This comparison provides a benchmark to assess the improvements we achieved through fine-tuning.

Steps Taken to Evaluate:

To evaluate the model, we used the Ground Truth Comparison evaluation method. First, the model receives as input the images from the benchmarking dataset and generates an output that contains the text that is extracted from the image. Next, we compare the text generated from the image to the ground truth using the ocreval tool. This tool employs the word error accuracy metric to provide statistics about the model's performance. Finally, we visualize the output received from the ocreval tool by calculating the mean word error for each noise.

VIII. CHALLENGES AND LIMITATIONS

Developing a robust Arabic OCR system using Tesseract 5.0 presented several significant challenges and limitations. These issues stemmed from both the inherent complexities of the Arabic language and the technical aspects of OCR technology.

A. Dataset Diversity and Quality

Balanced Dataset Composition: One of the primary challenges was ensuring a diverse and high-quality dataset. Arabic script includes both tashkeel (diacritical marks) and non-tashkeel text, which significantly affect OCR performance. Our dataset was composed of 60% tashkeel-free and 40% tashkeel-included text. Finding the optimal ratio required extensive experimentation. Applying tashkeel accurately was initially difficult due to insufficient data, which we addressed by utilizing the libtashkeel library [4] to add diacritics to 40% of our dataset.

Font Availability and Selection: At first, finding Arabic fonts that supported tashkeel was a significant challenge. Ensuring a varied representation of the Arabic script was critical, so we utilized three distinct font families: Baghdad, Bahij, and Rakkas. This diversity aimed to enhance the model's ability to generalize across different text styles.

B. OCR Engine and Segmentation Modes

Optimizing OEM and PSM: Selecting the appropriate OCR Engine Mode (OEM) and Page Segmentation Mode (PSM) was another challenge. These settings are crucial for optimizing the model's performance. Finding the best combination required multiple iterations and extensive testing. Despite our efforts, the model still struggled with recognizing certain characters, indicating the need for further refinement.

C. Character Recognition Issues

Challenges with Arabic Script Recognition: Despite having a well-prepared dataset, the model faced significant challenges in accurately recognizing certain Arabic characters. This was particularly evident with the cursive nature of the script and the complex contextual character shaping inherent in the Arabic language. These difficulties were not due to limitations in Tesseract's default unicharset. In fact, attempts to customize and replace the unicharset with alternative versions led to poorer performance. Our findings indicated that the best results were achieved using the unicharset generated during the training process. This underscores the intricate nature of Arabic OCR and the need for further refinements in model training and data preprocessing to improve recognition accuracy.

D. Image Resolution and Font Selection

The resolution of training images (DPI) and the selection of suitable fonts were crucial for creating effective training data. We utilized three distinct Arabic font families—Baghdad, Bahij, and Rakkas—to capture the variability in Arabic script. However, maintaining high image quality while ensuring diversity remained a challenging task.

When initially training the model, we encountered CTC (Connectionist Temporal Classification) errors. After investigating, we determined that the low DPI of the training images was the root cause. When generating images, TRDG [2] sets the DPI to 96. After further research we found out that Tesseract works best on high-resolution images (at least 300 dpi). So we implemented a function that adjusts the DPI of the generated images to 300 to ensure consistent image quality across the training data.

E. Data Augmentation and Noise

Establishing a Baseline: While data augmentation methods, such as adding noise, are known to improve OCR accuracy by increasing robustness, we initially opted for a clean dataset to establish baseline performance. This lack of augmented data might have limited the model's ability to handle real-world documents with various imperfections. Future iterations will incorporate noise and other augmentation techniques to enhance robustness.

F. Integration and Customization

Fine-Tuning for Arabic OCR: Integrating and customizing Tesseract 5.0 for optimal performance with the Arabic language required significant effort. This process involved fine-tuning the

pre-trained model to better handle Arabic script's unique characteristics, such as contextual character shaping and diacritics. Despite these customizations, achieving perfect accuracy remained elusive, highlighting the complexity of the task.

G. Evaluation Metrics and Benchmarking

Performance Assessment: Evaluating the model's performance involved using the word error rate (WER) with the ocreval library. This provided a clear insight into how well the model recognized and converted text. Benchmarking against existing models, like Hegghammer's Tesseract model, revealed that while our customized model showed improvements, there was still room for enhancement, particularly in handling noisy and diverse text samples.

IX. MODEL

The model name ioy_mi_1000_nn_fn_nns provides a short description of everything important in how we trained the model using the following parameters Maximum Iterations, Page Segmentation Mode (PSM), and Starting Model. This naming strategy makes it trivial to know at a glance what the particular setup and specifications a named model would be trained on. In the setup we provide, we try to attain a significant number of iterations: 1000 to hit the ideal balance between a good enough scope of learning and not overfitting the model. In our experiments, we found that going above 1000 iterations does not show a marked improvement.

PSM is one of the Tesseract properties that specify in what way it is going to segment an image into text areas. The current application uses PSM 13; the image is treated as a raw line of text. The latter is, of course, the primary purpose of our dataset since they were images already pre-segmented from lines of written Arabic text. The model treated each line of text independently, leading to higher recognition accuracies.

Then, training the pre-trained ara model of Tesseract regarding the Arabic language was initiated. Reutilization of the pre-trained model initiates a solid base for the very reason that it already has much information concerning the Arabic script. The pre-trained model was fine-tuned for increased performance with our dataset with regard to our specific requirements.

Quite a few practices were initiated in the training of the model. First is the initialization of the area model as a base with its pre-trained weights and knowledge of the Arabic script. Set up the parameters of training based on the study objective and the characteristics of the data, which were the number of iterations, about a maximum of 1,000 iterations, and PSM. We Ran the Training: we reached a maximum number of text samples within our 1,000 iterations. This experiential practice gave it a fine-tuning to regain its power of deciphering Arabic text. Track performances from the various metrics of WER (Word error rate), CER (Character error rate), and other performance metrics, which are set to monitor improvements in the accuracy and robustness of failures against the model.

Therefore, we hope to develop a very accurate and robust Arabic OCR model through scrupulous training parameter selection and configuration. This was helpful to the enormous task of dealing with the difficulties in Arabic text recognition, together with the pre-trained model of ARA, 1000 max iteration, and PSM 13.

In the next section, we will present the results of our model's performance compared to other models. This comparison will highlight the differences in accuracy and robustness across various noise conditions, providing a comprehensive evaluation of our model's capabilities relative to existing solutions.

X. RESULTS AND COMPARISONS

Having trained our Arabic OCR model, we benchmarked its performance by testing it against a variety of noise conditions, even though we did not train our model with these specific noises. The noise types included:

- col: "Color" - Variations in text color, including differences in hue, saturation, and brightness within the text images.
- bin: "Binary" - A binary version of each image was created, resulting in two versions: "color" and "greyscale." This binary conversion helps in assessing the model's performance under different image preprocessing conditions.
- blur: "Blur" - Blurred text images, which can occur due to motion, out-of-focus captures, or low-quality imaging.
- weak: "Weak Ink" - Text that is faint or lightly printed, making it difficult for the OCR system to distinguish the characters clearly.
- snp: "Salt and Pepper" - Noise in images characterized by the presence of random white and black pixels, resembling salt and pepper sprinkled over the image.
- wm: "Watermark" - The presence of watermarks on the text images, which are often semi-transparent or patterned overlays that can interfere with text recognition.
- scrib: "Scribbles" - Handwritten annotations or random scribbles present on the text images, which can obstruct the OCR system's ability to read the main text.
- ink: "Ink Stains" - Smudges or stains from ink that can obscure parts of the text, making it difficult for the OCR system to accurately interpret the characters.

By testing our model under these conditions, we were able to evaluate its robustness and ability to handle various real-world text recognition challenges. Despite not having trained the model with such noises, this benchmarking process provided valuable insights into the model's strengths and areas for improvement, guiding future enhancements and refinements.

Noise	ioy_mi_1000_nn_fn_nns (Our Model)	Tesseract V4	Hegghammer's Tesseract Benchmark
col	***10***	11.2	16.2
bin	***10.7***	11.2	14.4
col + blur	14.3	***11.8***	16
col + weak	***8.5***	10	18.6
col + snp	99.9	***88.6***	97.3
col + wm	14.6	***12.6***	17.7
col + scrib	17.1	***16.5***	23.3
col + ink	***21.6***	22.1	25
bin + blur	14.2	***12***	14.9
bin + weak	12.0	***10.4***	18.6
bin + snp	100	***79.8***	87.9
bin + wm	11.5	***11.3***	15.7
bin + scrib	***17***	17.6	23.3
bin + ink	***22***	24.3	26.8
col + blur + weak	21.2	***15.4***	19
col + blur + snp	100	***97.5***	100
col + blur + wm	37.2	***34.3***	37.3
col + blur + scrib	21.8	***18.7***	22.1
col + blur + ink	68.8	***68.5***	70.6
col + weak + snp	100	***88***	97.3
col + weak + wm	11.9	***10.2***	16.6
col + weak + scrib	15.9	***13.9***	22.1
col + weak + ink	21.2	***22.2***	28.8
col + snp + wm	100	***70.4***	86
col + snp + scrib	100	***69.9***	81.8
col + snp + ink	100	***86.1***	96.1
col + wm + scrib	23	***17.1***	20.7
col + wm + ink	36.4	***36.4***	38
col + scrib + ink	35.6	***34.6***	39.2
bin + blur + weak	20.3	***15.5***	17.2
bin + blur + snp	100	***98.4***	100
bin + blur + wm	25.3	***21.6***	23.5
bin + blur + scrib	21.5	***19***	22.4
bin + blur + ink	30.4	***26.7***	27.9
bin + weak + snp	87.8	***49.4***	71.4
bin + weak + wm	10.2	***9.3***	15.4
bin + weak + scrib	16.4	***15***	22.3
bin + weak + ink	***21.7***	23.1	28.4
bin + snp + wm	80.8	***41.7***	55.2
bin + snp + scrib	86.2	***38.9***	49.3
bin + snp + ink	100	***95.6***	92.2
bin + wm + scrib	18.3	***16.4***	18.6
bin + wm + ink	***32.9***	***32.9***	35.2
bin + scrib + ink	***34.9***	***34.9***	38.7

Fig. 1 **ioy_mi_1000_nn_fn_nns** (Our Model) vs. **Tesseract V4** vs.**Hegghammer's Tesseract Benchmark** using Hegghammer's benchmarking dataset evaluated using ocreval by calculating the mean of word error (percent), you will find that we ***bolded and italicized*** the ocreval of the model in which it outperformed the other models in this certain type of noise

XI. FINAL RESULTS AND CONCLUSION

Our model performance was compared against two models — the Tesseract V4 and Hegghammer's Tesseract Benchmark. Some are presented below for comparison, from which it is clear that our model worked relatively better than the noisy condition of Hegghammer's Tesseract Benchmark in:

1. col
2. bin
3. col + weak
4. col + ink
5. bin + scrib:
6. bin + ink:
7. bin + weak + ink
8. bin + wm + ink
9. bin + scrib + ink

These comparisons are focused on the areas where our model needed to improve, so that in noisy conditions it at least equals, or surpasses, the existing benchmarks. In the future, the works of the research group will be directed towards closing these gaps to ensure robustness and enhance the accuracy of the Arabic OCR system.

XII. FUTURE DIRECTIONS

To further improve our Arabic OCR system, future work will focus on the following:

- Enhancing Dataset Robustness: Expanding the dataset with more diverse and high-quality samples, including additional fonts and more sentences.
- Data Augmentation: Incorporating various forms of noise, backgrounds, and augmented text to improve the model's robustness and generalization capabilities.
- Advanced Customization: Further refining OCR configurations and potentially modifying the unicharset to better handle the complexities of Arabic script.
- Extensive Evaluation: Systematically evaluating the impact of each augmentation technique on OCR accuracy and exploring additional strategies to continually improve model performance.

In conclusion, while our research achieved notable progress in developing an Arabic OCR system using Tesseract 5.0, it also highlighted several areas for improvement. Addressing these challenges through enhanced data robustness, refined OCR configurations, and advanced augmentation techniques will be essential steps in advancing the accuracy and reliability of Arabic OCR technology.

XIII. ACKNOWLEDGMENT

We would like to thank Dr. Mohamed Taher for his valuable comments and suggestions to improve the quality of the paper and for helping us review our work regularly.

XIV. REFERENCES

[1]TesseractOCR:https://github.com/tesseract-ocr/tessdata_best

[2]TextRecognitionDataGeneratorLibray:
https://github.com/Belval/TextRecognitionDataGenerator

[3]Thomas Heghhammer paper:https://osf.io/preprints/socarxiv/6zfvs

[4]LibtashkeelLibrary:
https://github.com/mush42/libtashkeel?tab=MIT-1-ov-file