END-TO-END MACHINE LEARNING SOLUTION FOR RECOGNIZING HANDWRITTEN ARABIC DOCUMENTS

By
Reem Emad Shtaiwi

Supervisor
Dr. Gheith Ali Abandah, Prof.

This Thesis was submitted in Partial Fulfillment of the Requirements for the master's degree of Science in Computer Engineering and Networks

School of Graduate Studies
The University of Jordan

Jan, 2022

Committee Decision

This Thesis (End-to-End Machine Learning Solution for Recognizing Handwritten Arabic Documents) was successfully defended and approved on ----------

Examination Committee	**Signature**
Dr. Gheith A. Abandah, (Supervisor) Prof. of Computer Engineering	_____
Dr. Iyad F. Jafar, (Member) Prof. of Computer Engineering	_____
Dr. Mohammad R. Abdel-Majeed, (Member) Assoc. Professor of Computer Engineering	_____
Dr. Ahmad F. Al-Jaafreh, (Member) Assoc. Professor of Computer Engineering, Tafila Technical University	_____

Dedication

To my parents for their prayers and endless support

To my sisters Wafa'a and Noor

To my brothers Mohammad, Saif, Hashem, and Adel

To my close friends who supported me all the time

To my supervisor, Prof. Gheith A. Abandah, who spared nothing towards my success

Acknowledgement

I would like to thank my supervisor Prof. Gheith Abandah for his advice, suggestions, help, and patience during our work on this thesis. I highly appreciate his efforts. He is an excellent mentor and he taught me how to do research correctly. My gratitude is beyond words.

I would also like to thank my colleague Eng. Safa'a Swalhah for her thesis that has helped me. I would also like to thank the authors of the Start, Follow, Read code and the Handwriting recognition model code for making their work available for researchers like me.

Table of Contents

Subject **Page**

List of Figures

List of Tables

List of Abbreviations

1D-RNN	One-Dimensional Recurrent Layers
2D-LSTM	Two-Dimensional- Long Short-Term Memory
AI	Artificial Intelligence
API	Application Programming Interface
BLEU	Bilingual Evaluation Understudy
BLSTM	Bidirectional LSTM
BN	Batch Normalization
CER	Character Error Rate
CNN	Convolutional Neural Network
CNN-LSTM	Convolutional Neural Network - Long Short-Term Memory
ConvNets	Convolutional Network
CPU	Central Processing Unit
CRNN	Convolutional Recurrent Neural Network
CTC	Connectionist Temporal Classification
CUDA	Compute Unified Device Architecture
DDR3	Double Data Rate 3
DPI	Dots Per Inch
GPU	Graphics Processing Unit
GT	Ground Truth
HMM	Hybrid Hidden Markov Model
HTR	Handwriting Text Recognition
HWR	Handwriting Recognition

IAM	Handwriting Database contains forms of offline handwritten English text
ICDAR	International Conference on Document Analysis and Recognition
IDE	Integrated Development Environment
IFN / ENIT dataset	Institute of Communications Technology / Ecole Nationale d'Ingénieurs de Tunis
JPG	Joint Photographic Experts Group
JU-OCR2	Jordan university- optical character recognition 2
LDC	Linguistic Data Consortium
LF	Line Follower
LR	Learning Rate
LSTM	Long Short-Term Memory
MADCAT	Multilingual automatic document classification analysis and translation Dataset
MDLSTM-RNNs	Multi-Dimensional Long Short-Term Memory Recurrent Neural Networks
MP	Max Pooling
MSE	Mean Square Error
OpenHaRT'13	Open Handwriting Recognition and Translation 2013
PAWs	Parts of Arabic Words (sub-word)
PIL	Library Python Imaging Library
PNG	Portable Network Graphic
READ	Recognition and Enrichment of Archival Documents
ReseNet18	Residual Convolutional Network
RNN	Recurrent Neural Network
RNNLIB	Recurrent Neural Network Library

RPN	Regional Proposed Network
RU-net	Recurrent Residual Convolutional Neural Network based on U-Net
SFR	Start, follow, read
SOL	Start of Line
STN	Spatial Transformation Layer
Tiff	Tagged Image File Format
VGG	Very Deep Convolutional Network for Large-Scale Image Recognition
XML	Extensible Markup Language

END-TO-END MACHINE LEARNING SOLUTION FOR RECOGNIZING HANDWRITTEN ARABIC DOCUMENTS

By
Reem Emad Shtaiwi

Supervisor
Dr. Gheith Ali Abandah, Prof.

Abstract

Nowadays, the research in offline handwriting recognition (HWR) for various languages has gained rising attention, especially in the Arabic language. This is connected to the growing necessity to digitize Arabic documents in several applications such as exploring large documents, automated sorting of express mail, editing of earlier printed documents, and bank control processing. Regrettably, notwithstanding decades of research, there is no satisfactory solution for recognizing cursive handwriting like the Arabic language because of its difficulty.

Different techniques have been introduced to address HWR problems for multiple different languages. The modern advancements in deep learning and recognition algorithms supported building systems able of handling both the two-dimensional phase of the input and the subsequent phase of the prediction. Among these algorithms are the multi-dimensional long short-term memory recurrent neural networks (MDLSTM-RNNs) with the objective function of the connectionist temporal classification (CTC). Such methods give low error rates and are becoming state-of-the-art of HWR.

This thesis aims to study the possibility of implementing an end-to-end machine learning solution by applying a deep learning system using the MADCAT dataset, to accurately recognize Arabic handwritten documents through simultaneously learning text detection, segmentation, and finally recognizing the Arabic documents and converting them to editable text.

Our algorithm is an integration of several distinct algorithms for Latin languages, achieved high accuracy in parsing the full page, converting it to lines, and predicting the writing within each line. Our algorithm consists of three models, the first is to determine the start of the line (SOL model), the second is to follow the line from beginning to end, cut it and normalize it (LF model), and finally recognize the written text (HTR model). Our solution has been applied to a large dataset that contains a variety of lines and different problems that need to be addressed.

We determine work efficiency using the loss for the first two models and character error rate (CER) for the last model. The SOL model achieves a loss of 34.62 and the LF model achieves 73.92 loss. The HTR model achieves a CER of only 3.96%.

CHAPTER I

Introduction

This chapter concisely explains the research field, clarifies the value of recognizing Arabic handwritten documents, and describes the common important methods used lately to recognize HWR documents in the Arabic language and other languages. Also, it briefly explains the concept of machine learning and deep learning, including the main differences between them. It additionally clarifies the research goals of this thesis, its contributions, and lastly gives the outline of the thesis.

1.1 Machine Learning and Pattern Recognition

For the last ten years, artificial intelligence and machine learning has made extraordinary progress in many tasks that were considered mathematically impossible to solve. The existence of a huge number of datasets and their availability for science, the power of computers available in our present and the remarkable improvement of algorithms, and high-quality open-source libraries that allow researchers to quickly encode and test models, all of this led to the development of the machine learning system tremendously and remarkably in many areas. These areas include (1) speech analysis, (2) games analysis, (3) image recognition, and (4) pattern recognition (Došilović *et al*., 2018).

Pattern recognition is a very important field of machine learning, concerned with the automatic detection of patterns using artificial intelligence (Stepney *et al*., 2006). It consists of several types of recognition, the most important of these types is handwritten recognition (HWR) (El-Sawy *et al*., 2016). HWR is the ability of a machine to recognize handwritten letters and words, so that an image containing text is transferred to editable text to avoid rewriting (Lawgali *et al*., 2015).

1.2 Handwritten Recognition

The HWR has attracted the attention of many researchers in the field of pattern recognition. Despite of this, the HWR has faced serious challenges, such as the lake of information about the boundaries of individual characters easily, because the handwriting is not restricted to a specific style of writing, the drawing of letters or the distance of words from each other in one line. All these features depend on the amount of lifting the pen at an angle to the paper, and this depends on the personality of the writer (Mukherjee *et al*., 2017).

The traditional HWR method includes several stages that start with preprocessing the scanned images to prepare them for the final stages. Then, the image is divided into lines and then into words, here the words are divided into letters, and then the features are extracted from the character fragmentation. The characters are then recognized by trained models, and post-processing is preferred to improve recognition accuracy (Abandah and Jamour, 2010).

HWR is divided into two types: online and offline. In the online HWR process, letters are recognized during writing using a digital trace of the pen. In contrast, offline HWR is done by scanning the handwritten paper and then recognizing and analyzing handwritten text (Lawgali *et al*., 2015). Usually, online handwritten recognition is easier than offline handwritten recognition (Abandah and Jamour, 2010). In our current research, We interested in the offline HWR.

The input in the offline HWR process is a two-dimensional, variable-size image, and the output for it is a series of letters, words, and spaces between them. Offline HWR plays an important role in many practical areas such as automatic mail sorting, editing of old documents, *etc*., and so far, there are no satisfactory solutions available for offline

HWR solution. Many languages have been studied in the field of HWR, such as Persian, Urdu, Chinese, and Latin, and a comparison between them. Little research has been done to identify handwritten sentences in Arabic (Lawgali *et al*., 2015).

1.3 What Are the Differences Between Machine Learning and Deep Learning?

Deep learning is a branch of artificial intelligence (AI), and Machine learning (ML) is a section of it. Machine learning algorithms separate data, learn from it, and then determine the answers using what they have learned. Popular algorithms of ML are based on a particular collection of features extracted from raw data (Magnimind, 2019).

Deep learning can be trained using supervised backpropagation techniques if the labeled training set has enough information to determine network parameters. Notwithstanding the flexibility of deep learning, it can only be employed on queries whose inputs and objects can be coded intelligently with vectors of determining dimensions. It is a major limitation, as many significant difficulties are abundantly represented in sequences whose lengths are not identified in advance (Sutskever *et al*., 2014).

Classical machine learning algorithms break queries into multiple sectors and resolve them separately. As for deep learning, it resolves the query from end to end. The important feature of deep learning is that the more extra training data that supplies into deep learning algorithms, the more helpful these algorithms enhance to resolve problems (Magnimind, 2019). The main differences among machine learning and deep learning can be reviewed as follow:

- Deep learning algorithms need a large quantity of data to produce the best results. The general machine learning algorithms do not depend on the quantity of data prepared to develop the performance of them.

- Deep learning needs advanced hardware to execute it, such as graphics processing units (GPUs). General machine learning can be prepared on low-end machines such as the central processing unit (CPU) with usual specs.

- The general machine learning algorithms require handcrafted features as inputs like pixel values, shape, arrangements, adjustment, location, and color. The efficiency of these algorithms depends on the length to which these features are determined and extracted. But deep learning does not depend on handcraft features.

- Deep learning needs a long time to train due to many parameters. The general machine learning algorithms need less time for training, of a few seconds to a few hours.

- The query must be separated into several parts to be resolved using the general machine learning algorithm. For example, things must first be identified, as the task requires determining what the purpose is and wherever it is in the image. On the opposite, in deep learning, the process will necessitate place from end-to-end (Magnimind, 2019).

1.4 Importance of Recognizing Arabic Handwritten Documents

The study toward offline handwritten Arabic character recognition has got larger attention in current years, because of the expanding demand for Arabic document digitization (Wahdan *et al*., 2020), like word search in bulky documents, automatic mail sorting, convenient editing of pre-printed documents, bank check processing, post office self-regulation for mail sorting (Belabiod and Belaïd, 2018). HWR technologies package is additionally used to create and accomplish modern classrooms, explore cultural videos, and analyze data for medicinal writings (Dutta *et al*., 2018). Also, it provides an automatic translation of antique Arabic documents (Mohammed *et al*., 2014).

The Arabic language is one of the six original communication languages in the

universe. Its application have increased over several countries around the globe, and it is the heart of the Arab world. Also, islamic peoples use the Arabic language as their basic language because it is the language of the Qur'an (Wahdan *et al*., 2020). Also, Arabic is the 5th spoken language in the world, approximately 422 million people speak Arabic as a native language (Ali and Suresha, 2019).

The Arabic language contains 28 letters, including hamzas and elmodod. Every character has several structures that depend on the location of the letter on the word, whether it is at the start of the word, in the center of the word, or at the ending of the word (Wahdan *et al*., 2020). Table 1 describes the Arabic alphabet structure depends on the location.

There are no upper-case and lower-case letters in the Arabic language, unlike English or other languages. The morphology of the Arabic language is not simple. It may be difficult because it includes radical words, prefixes, and suffixes (Wahdan *et al*., 2020).

Arabic HWR is a difficult and challenging work for two reasons: (1) the main differences and very few similarities compared to the differences with other languages in the world, (2) the writing style from right to left. Hence, high-quality features must be extracted to develop HWR on handwritten Arabic characters (Ali and Suresha, 2019).

As discussed earlier, the HWR has spread, and the largest studies cover the analysis of other communication languages like English. Little studies on HWR have been directed on the Arabic language (Wahdan *et al*., 2020).

Table 1: The Arabic Alphabet (Lorigo and Govindaraju, 2006)

Name	Isolated	Initial	Medial	Final
Alif	ا	ا	ـا	ا
Baa	ب	بـ	ـبـ	ـب
Taa	ت	تـ	ـتـ	ـت
Thaa	ث	ثـ	ـثـ	ـث
Jiim	ج	جـ	ـجـ	ـج
Haa	ح	حـ	ـحـ	ـح
Khaa	خ	خـ	ـخـ	ـخ
Daal	د	د	ـد	ـد
Thaal	ذ	ذ	ـذ	ـذ
Raa	ر	ـ	ـر	ـر
Zaay	ز	ـ	ـز	ـز
Siin	س	سـ	ـسـ	ـس
Shiin	ش	شـ	ـشـ	ـشـ
Saad	ص	صـ	ـصـ	ـص
Daad	ض	ضـ	ـضـ	ـض
Taa	ط	طـ	ـطـ	ـط
Dhaa	ظ	ظـ	ـظـ	ـظ
Ayn	ع	عـ	ـعـ	ـع
Ghayn	غ	غـ	ـغـ	ـغ
Faa	ف	فـ	ـفـ	ـف
Qaaf	ق	قـ	ـقـ	ـق
Kaaf	ك	كـ	ـكـ	ـك
Laam	ل	لـ	ـلـ	ـل
Mim	م	مـ	ـمـ	ـم
Nuun	ن	نـ	ـنـ	ـن
Haa	ه	هـ	ـهـ	ـه
Waw	و	ـ	ـ	و
Yaa	ي	يـ	ـيـ	ـي

1.5 Problem Statement

Lately, the growing need to digitize Arabic documents for various purposes has expanded the research for obtaining comprehensive solutions to recognize offline handwritten Arabic documents. But notwithstanding decades of study, there are no adequate solutions to recognize cursive handwriting same Arabic texts.

Cursive handwriting recognition is challenging to recognize. To develop effective segmentation algorithms for recognizing cursive handwriting texts, it requires significant scientific knowledge.

Many methods have been recommended in the literature to address HWR problems. Given the modern advancements in deep learning and new algorithms, it is possible to develop systems able to handle both the two-dimensional nature of the inputs and the sequential aspect of the recognition. In the machine learning techniques, we need to have large training data to get sufficient training.

To get good results from ML algorithms, we need to train it with a large amount of data. So, we selected a big Arabic handwritten dataset, which is the multilingual automatic document classification analysis and translation dataset (MADCAT) (Géron *et al.*, 2017).

MADCAT dataset was collected by the Linguistic Data Consortium (LDC) to support the training, testing, and evaluation techniques used for pre-processing, recognition, and translation of Arabic handwritten images. It was built in a controlled setting. The data was written by original Arabs who are fluent in the Arabic language. Original authentic Arabic texts were selected for the dataset from reliable sources of texts from newspapers and news sites. MADCAT contains many styles of handwriting in training and testing images (Abandah and Al-Hourani, 2018).

There are several difficulties in the MADCAT dataset: (1) external graphical elements, (2) varying text bodies, (3) pepper noise, (4) writing errors, (5) irrelevant text or numerals (non-Arabic characters), (6) vertical lines, and many more. Figure 1 presents an example of MADCAT document (Abandah and Al-Hourani, 2018).

In this thesis, we are investigating the possibility of implementing an end-to-end machine learning solution by applying the state-of-the-art systems of Latin HWR, using the MADCAT dataset to accurately recognize the Arabic handwritten documents.

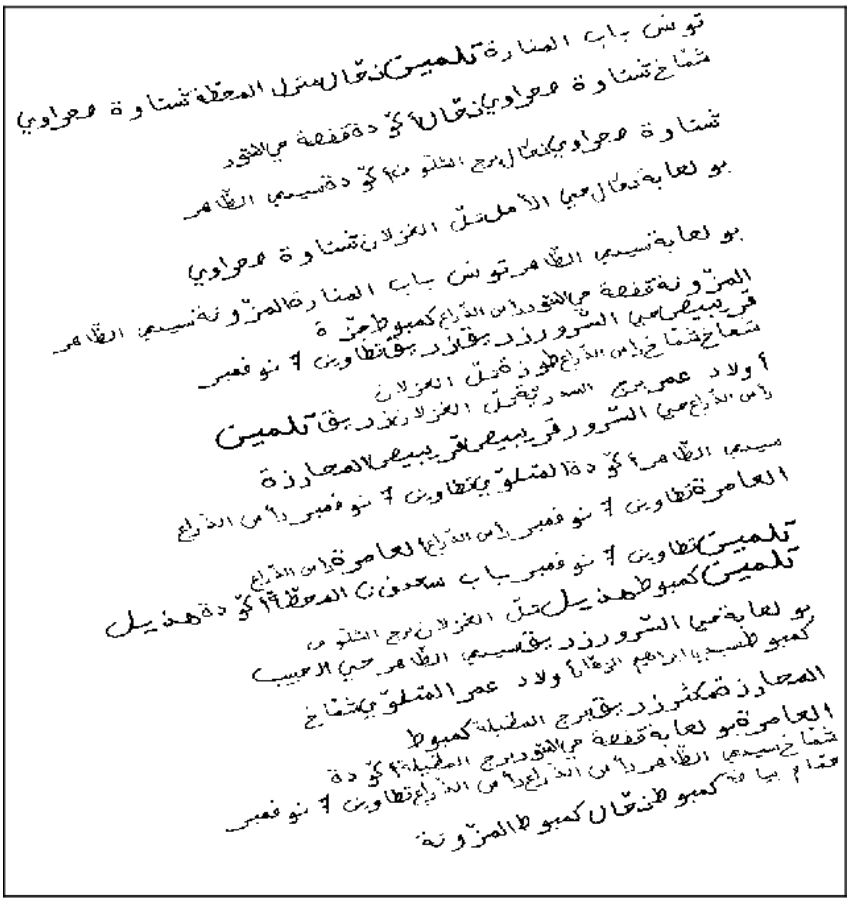


Figure 1: Example MADCAT document (Abandah and Al-Hourani, 2018)

1.6 Modern Technology Used in HWR

The continuous nature of writing makes it difficult to divide into individual letters to be identified individually, each one separately. So, character isolation and prediction methods were widely used in the '90s and gradually replaced by the sliding window so that features are derived from vertical images of images containing handwritten sentences (Lawgali *et al*., 2015). This method transforms the problem into a sequence of the first transformation sequence, by using convolutional neural networks can encoding the 2D image or by extracting the features (Bluche *et al*., 2017).

Currently, advanced studies and developments in deep learning have allowed the building of robust systems that can cope with the 2D images inputs with the sequential output. Specifically, MDLSTM - RNN is the multi-dimensional long short-term memory

recurrent neural networks, directly related to CTC, which is the connectionist temporal classification, use these technologies led us to a decrease in the error rate of the recognition. Then, it became the adopted model for the HWR. Even today, current systems require images with segmented lines, which are rare in the real world. Therefore, to reproduce the document, automatic line segmentation algorithms must be used for pipeline processing (Bluche *et al*., 2017).

All previous research focused on the work of dividing the document into lines and discovering the line before the recognition process. So that, a few researchers tried to integrate all the processes together. The idea of splitting a document into lines is the most difficult in the HWR process, and most errors in prediction are caused by incorrect prediction of the beginning and end of the line (Wigington *et al*., 2018).

The presence of great interest through international competitions that are interested in Handwritten Recognition using ML for multiple languages led to good solutions emerged that we could benefit from in our issue, which are: (1) the Start, Follow, Read (SFR) system and (2) Handwriting Recognition (HTR) system.

The Start, follow, read (SFR) system was the state-of-the-art solution which designed to realize historical handwritten documents in German language. SFR is a deep learning system that simultaneously completes text detection, segmentation, and recognition. It exceeded the champion of the handwriting recognition competition of the international conference on document analysis and recognition (ICDAR) 2017 on the READ dataset. After 2018, this system was applied in a study to distinguish the handwritten Arabic documents (Sawalhah and Abandah, 2020).

SFR system consists of three interfaces: (1) the start of the line (SOL), (2) the line follower (LF), (3) the line-level handwritten recognition (HWR). SFR achieves great

results on large and complex datasets: READ and MADCAT datasets (Sawalhah and Abandah, 2020).

The handwriting recognition (HTR) system is also a deep learning model designed to realize historical handwritten documents in German language in the ICDAR 2017 competition. HTR system achieved the second-best accuracy during the competition.

These state-of-the-art algorithms have been merged and modified to accept the handwritten Arabic documents and recognize it, and several achievements were made in the field of Arabic HWR.

SFR algorithm was modified in terms of two interfaces: (1) SOL network, (2) LF network. It's developed the SOL network architecture and its training strategy. Also, it's offered a novel LF network that learns to fragment and normalizes handwriting lines for data to the HTR system to recognize the Arabic text. The developers handled the pre-training step of SFR algorithm, and the training framework is forthwith ready to enter the final step of full training, using documents that simply include the line transcription level (Sawalhah and Abandah, 2020).

1.7 Research Objectives and Contributions

The chief goals of this thesis are as follows:

- Studying, analyzing, and preparing the MADCAT dataset.
- Complete adapting and combining the SFR algorithm with HTR algorithm to build an accurate system to detect, segment, and read the handwritten Arabic documents.
- Testing the model on the MADCAT dataset after completing the remaining adjustment of the proposed algorithm.
- Tuning the model to give the best readings on the MADCAT dataset.

Ultimately, we conclude that this thesis is a good source to use state-of-the-art end-to-end machine learning methods to segment handwritten Arabic documents. It additionally presents a good approach to encourage the use of machine learning methods, particularly deep learning, in recognizing handwritten Arabic documents.

1.8 Thesis Outline

The rest of the thesis are organized as follows. Chapter Two presents a literature review from early methods developed to determine and solve the difficulty of recognizing handwritten documents for Arabic and other languages. Chapter Three illustrates the technologies that we have used to segment Arabic handwritten documents into lines and recognize them. Furthermore, it presents the networks and sub-models employed in this solution. Chapter Four explains the methods used for data processing stage. Chapter Five explains the experiments that we did in this solution, the results of our system, and analysis of the results. Lastly, Chapter Six presents the conclusions and recommendations for future work.

CHAPTER II

Literature Review

The segmentation of the text handwritten line can be classified toward three methods: 1) bottom-up methods, 2) top-down methods, and 3) machine learning methods. The first two methods need high-level knowledge of documents, like document adjustment, inter-spaces, *etc*. In contrast, machine learning methods prepare the image without any earlier knowledge. Some of the current systems are end-to-end combined systems, with no additional support or preprocessing, while others use document information (Belabiod and Belaïd, 2018). The following sections reviews the related work of the Arabic HWR.

2.1 Bottom-Up and Top-Down Methods

The bottoms-up and top-down methods require advanced knowledge of documents, such as document orientation, inter-spaces, *etc*. These systems must combine all types of processing methods or use segmentation algorithms to devide the documents. These methods have good results on handwritten Latin or Germanic documents (Belabiod and Belaïd, 2018). Below are examples of the most important work used in this field.

Muallim and Yamaguchi (1987) one of the first solutions that provided a structural recognition process of Arabic cursively handwritten words. In this solution, words at the beginning are segmented into strokes. These strokes are listed using their geometrical and topological features. Lastly, the corresponding position of the recorded strokes are considered, and the strokes are mixed in several levels into a string of characters that describes the approved word. The ability of the strokes to be practiced as a central character to represent Arabic scripts, and the ability of their extraction and recognition, did exactly consider in the determination of the strokes. This system is thoroughly

effective against various kinds of distortions occurring in Arabic scripts. The implementation of this design leads to great recognition accuracy, almost 91% on a little data set of 400 words recorded by two persons.

Yusufi and Udpa (1992) performed a statistical method toward the recognition of Arabic characters. As an initial action, the character is segmented into primary and secondary characters "i.e., dots and zigzags". The trivial parts of the character are then hidden and identified individually, by decreasing the number of classes from 28 to 18. single measures of the shape are received from the normalized notes and a 9-D feature vector is concerned for every character. The results increased to 99.5%, but the test data involved handwritten and machine-printed characters, and the handwritten just 10 units were used.

Sulaimani and Razazi (2003) proposed a method to obtain private handwritten characters on forms. The sides of the letter did find in the behavior of noise. The central body isolated from every one of the different symbols, and design was selected. Within a dataset of 220,000 samples from higher than 50,000 authors, the accuracy was 96.4% of the characters.

Haj *et al.* (2005) performed the UOB method that got the chief competition in ICDAR. The method determines the word baseline and selects the features of a strong objective language employing a sliding window, externally character segmentation. The method uses a simple HMM recognizer designed for speech recognition. They used the IFN/ENIT dataset and achieve an accuracy (86.51 %).

Abandah *et al.* (2005) introduced an off-line recognition system based on the chosen feature. The authors classified these features toward quantitative and qualitative features. Quantitative features cover dot numbers, character length, character width,

character range, and character weight over and under the baseline. Qualitative features cover loops, branches, topological information, topological relationships, dot location, link points, and crossing points. The features were used in this research are width, height, aspect ratio, density, *etc*.

This system was trained and tested with practical examples of handwritten Arabic characters. The recognition based on the chosen features provides good accuracies of numbers equal to 88% and for letters equal to 70%.

Abandah and Malas (2010) applied feature selection methods to estimate a loose range of features extracted from handwritten Arabic characters. In this solution, the authors selected 96 features of the secondary parts of the characters, the central body, the skeleton, and the edges. To chooses the most useful feature subsets, they employed five separate feature selection methods. Then, they estimated those feature subsets to choose a subset of features that provides the highest accuracy. Later, the accuracy was investigated employing three classifiers. Figure 2 shows the strategies employed in this method.

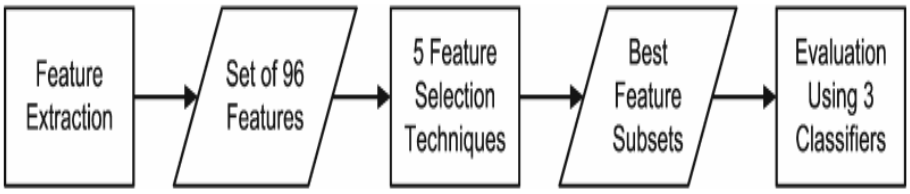


Figure 2: Feature Extraction, Selection, and Evaluation (Abandah and Malas, 2010)

2.2 Machine Learning Methods

For most HWR designs, text lines need to be recognized and segmented of the image before recognition can happen. This is challenging for HWR documents, specifically the historical documents because they may include notable significances of noise, such as spots, holes, remaining decoration, ink fade, and leakage (Wigington *et al.*, 2018).

Notwithstanding this, line discovery and segmentation are usually independently operated by separate algorithms, and multiple HWR models are created, trained, and estimated exclusively on the line segmentations (Wigington *et al.*, 2018).

The segmentation and recognition are important parts of HWR, most maximum previous works resolve these problems separately: text lines are identified, segmented, and pre-processed into orthogonal image parts before being copied by the recognition model. And as we know, errors in the discovery, segmentation, or pre-processing levels usually drive to bad recognition (Wigington *et al.*, 2018).

Deep learning in line segmentation is employed more than word segmentation. Traditional methods of word segmentation are bottom-up methods are based on relevant component analysis and extraction of structural features or indeed both. These methods have excellent results on handwritten Latin or Germanic documents (Belabiod and Belaïd, 2018).

Ding and Liu (2006) introduced works on offline handwritten Chinese and Arabic script recognition employing a segmentation-driven method. Two fundamental problems are: (1) isolated character recognition and (2) establishment of the probabilistic segmentation design. To develop the hidden character recognition accuracy, they suggest a heteroscedastic linear discriminant analysis algorithm to obtain more further

discrimination information of real character features and achieve the smallest classification error learning design to optimize classifier parameters. Also, in the segmentation step, data from three different roots, namely (1) geometrical layout, (2) character recognition confidence, and (3) semantic model are combined into a probabilistic structure to provide the most useful script analysis. Accurate recognition rate equals 59.2% performed on a real handwritten Arabic script.

Menasri *et al.* (2007) performed this study that represented an off-line handwritten Arabic word recognition system. Both specific grapheme segmentation and feature extraction are produced for Latin cursive handwriting. The recognizer here is the hybrid HMM/NN. The authors proposed a novel shape-based alphabet during handwriting Arabic recognition which is aimed to profit from some specificities of Arabic writing.

They presented several analyses using IFN/ENIT benchmark dataset to validate their method. The recognizer shows as close as the state-of-the-art recognition rate with 87%. Those results are quite promising as many aspects and developments.

Abandah *et al.* (2008) proposed research that investigated the most useful sets of feature extraction methods and analyzing the accuracy of popular classifiers for Arabic letters. The primary key study technique is applied to choose the most helpful subset of features out of a huge number of selected features. They applied OCR algorithms to extract the features and used 5 classifiers to predict the HWR Arabic letters which are: quadratic discriminant analysis (QDA) classifier, linear discriminant analysis (LDA) classifier, diagonal quadratic discriminant analysis (DQDA) classifier, diagonal linear discriminant analysis (DLDA) classifier, and the k-nearest neighbor (kNN) classifier.

The authors applied the parametric and non-parametric classifiers and discovered out that a subset of 25 features is required to receive 84% recognition accuracy applying

a linear discriminant classifier and applying more extra features does not essentially enhance this accuracy and for features less than 25 features, a quadratic discriminant classifier is extra accurate than the linear classifier.

Graves and Schmidhuber (2009) proposed MDLSTM-RNNs in the meaning of handwriting recognition, which succeeded in the ICDAR 2009 competition. Figure 3 presents the design of this system. In this design, the LSTM layers study the input in the four possible directions and measure the inner state of the LSTM cell and the output of the states and outputs of the past positions in the horizontal plus vertical directions. All MDLSTM layers follow a convolutional layer. Here there is an individual feature map by character, at the top of the network. Those maps remained collapsed into a series of prediction vectors. The CTC algorithm is employed to train the network to identify text lines. The transformation of 2D to 1D happens during the collapse layer. The collapse layer measures a simple gathering of the characteristic maps in the vector sequences. After that, all data is decreased in the vertical dimension to one vector.

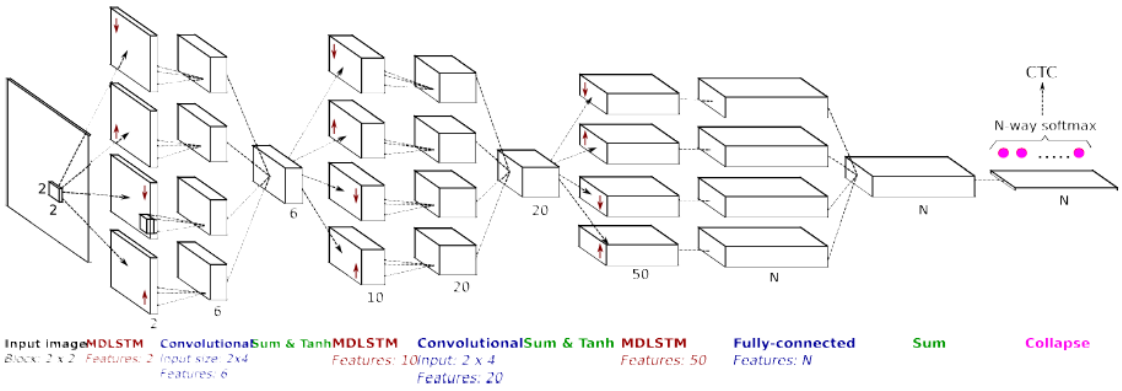


Figure 3: Graves and Schmidhuber MDLSTM-RNN Architecture

Jamal *et al.* (2014) introduced a different segmentation methodology of Arabic handwritten text line toward words. Their proposed method of text segmentation employs the principles of Arabic writing characteristics. The main difference between their

segmentation approaches from earlier methods is using the knowledge of Arabic writing by shape analysis. This strategy for segmentation is two stages approach: (1) metric-based segmentation, (2) recognition-based segmentation.

First, the connected components (CCs) of a text line were selected. CCs consist of associated black pixels. Accordingly, the initial step in segmenting an Arabic handwritten word is detecting and defining its CCs. The 8-connectivity design was adopted. Later, metric based, and ESL based segmentation was involved. The analyses show hopeful outcomes. Figure 4 shows the overall methodology as a block diagram.

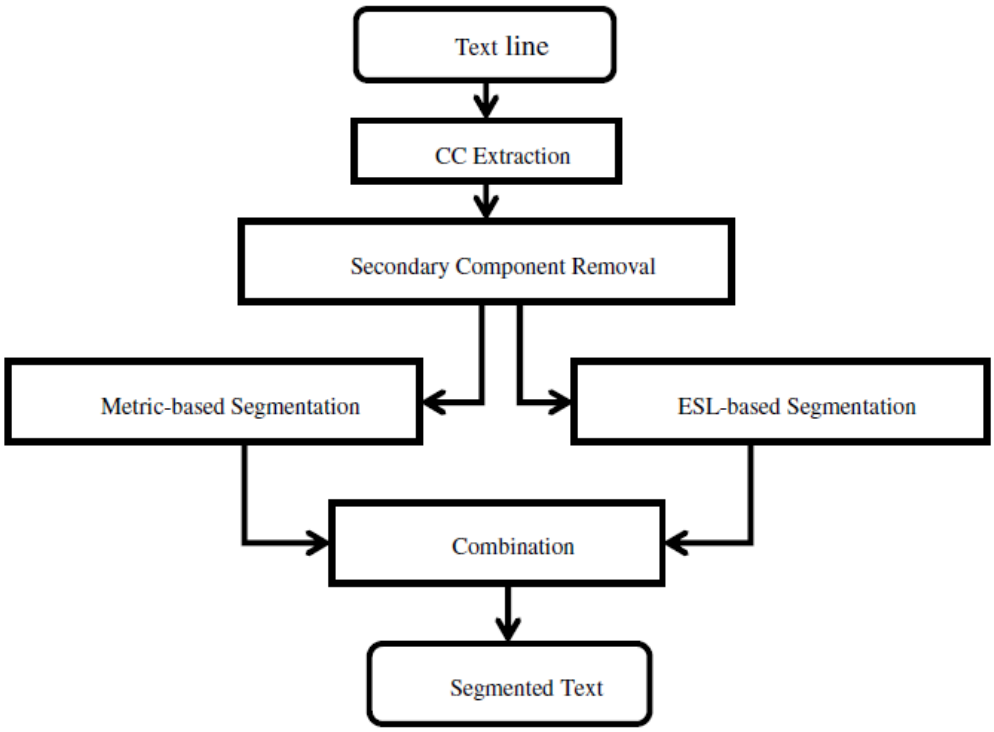


Figure 4: Overall methodology of Jamal search (Jamal *et al.*, 2014)

Abandah *et al*. (2014) presented the word matching algorithm in the JU-OCR2 optical character recognition system of handwritten Arabic words. This system performed state-of-the-art accuracy over many techniques including an effective word matching algorithm. In this solution, they decreased the average sequence error in the IFN/ENIT

dataset of handwritten Arabic words of 32.3% to an average word error of only 5.0%. Utilizing many sequences, rather than one, decreases the average error by 27.0%. Matched with an algorithm used in a leading system, this algorithm proposed a 6.7% lower average word error for the foremost two test sets.

This system is based on the segmentation of cursive words to graphemes. Most parts are segmented to one grapheme by character. Then, some characters are segmented into many graphemes. A collection of features is picked from a strong set of features of the segmented groups. To achieve high recognition accuracy, authors make a feature vector passed to a BLSTM network. The CTC is employed in the output layer. Additionally, The RNN was established applying the RNNLIB library. Figure 5 reviews the chief phases employed in this system and describes wherewith the system improves the miss-transcribed latest character applying word matching.

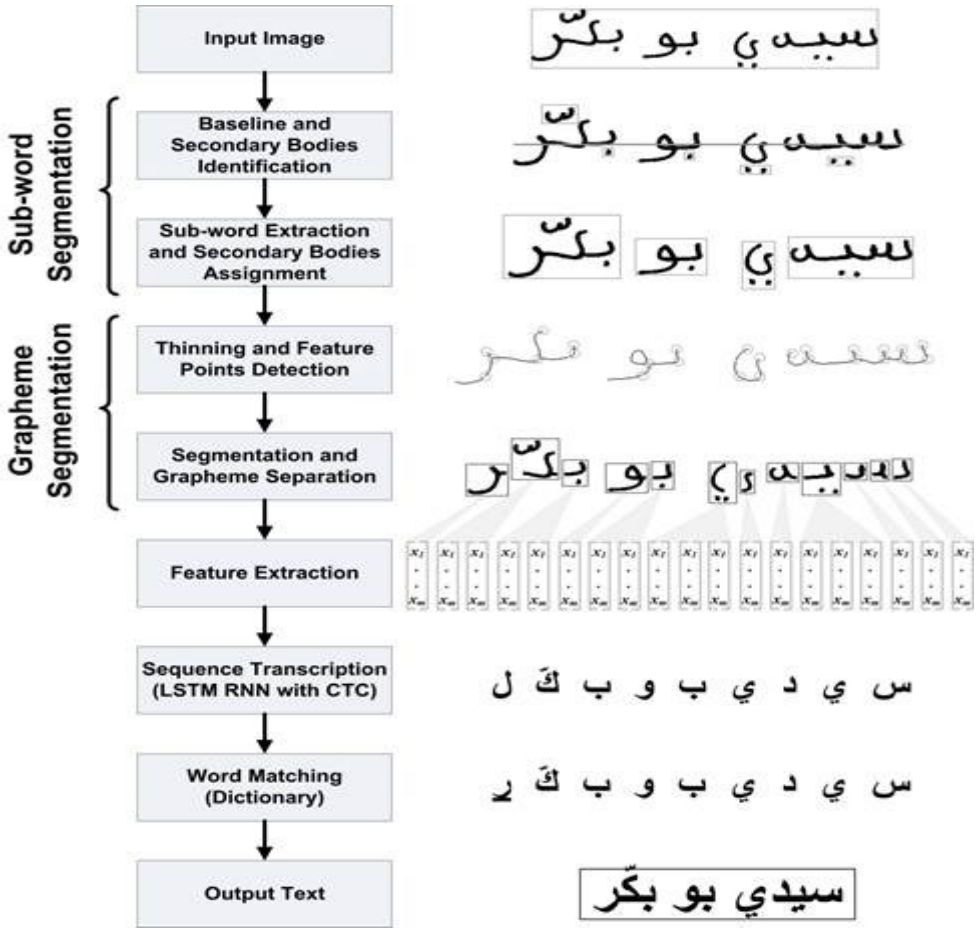


Figure 5: JU-OCR2 Phases (Abandah *et al.*, 2014)

Abandah *et al.* (2015) introduced a different feature extraction procedure of handwritten Arabic letters. Authors believed that extracting and selecting analytical features of handwritten Arabic letters, their mainframes, and their subsequent segments produced feature subsets that provide higher recognition accuracies compared to the subsets of the entire letter's individual. So, they targeted improving the feature extraction step in recognizing handwritten Arabic text. Firstly, pre-segmented letters were partitioned into central and secondary body parts. Then, moment features were obtained from the entire letter as quite as from the main body and the secondary components. With adopting a multi-objective genetic algorithm, dynamic feature subsets were elected. Finally, different feature subsets were estimated according to their classification error

using an SVM classifier. The recommended approach improved the classification error in all cases studied. For instance, the developments of 20-feature subsets of normalized mean moments equal to 15% and Zernike moments equal to 10%.

Bluche (2016) presented an adjustment to MDLSTM-RNN to allow the end-to-end process handwritten paragraphs. The author replaced the collapse layer by converting two-dimensional representation into a sequence of predictions including a recurrent version that could choose an individual line in time. Figure 6 shows the modification performed to the collapse layer. While the regular collapse (left, top) is a simple amount.

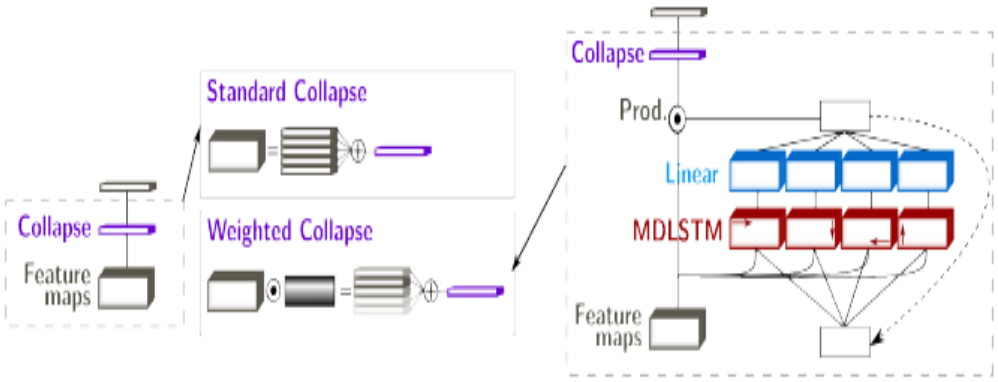


Figure 6: Bluche Modification of the Collapse Layer (Bluche, 2016)

Boufenar *et al.* (2016) introduced the artificial immune system as a supervised learning method that includes the concepts of natural immunity to cope with complicated classification queries. The goal of this study is to examine the applicability of an artificial immune system in offline isolated handwritten Arabic letters. The advanced system was included in three main modules: preprocessing, feature extraction, and recognition. Figure 7 presents the proposed model of recognition scheme. The method was trained and tested with a ten-fold cross-validation technique on an absolute realistic dataset that they made from the popular IFN/ENIT benchmark, and the parameter tuning was performed among a grid-search algorithm including leave-one-out cross-validation. The achieved

results of the advanced system with an accuracy rate of 93.25% and often exceed the most famous classifiers of scikit learn library.

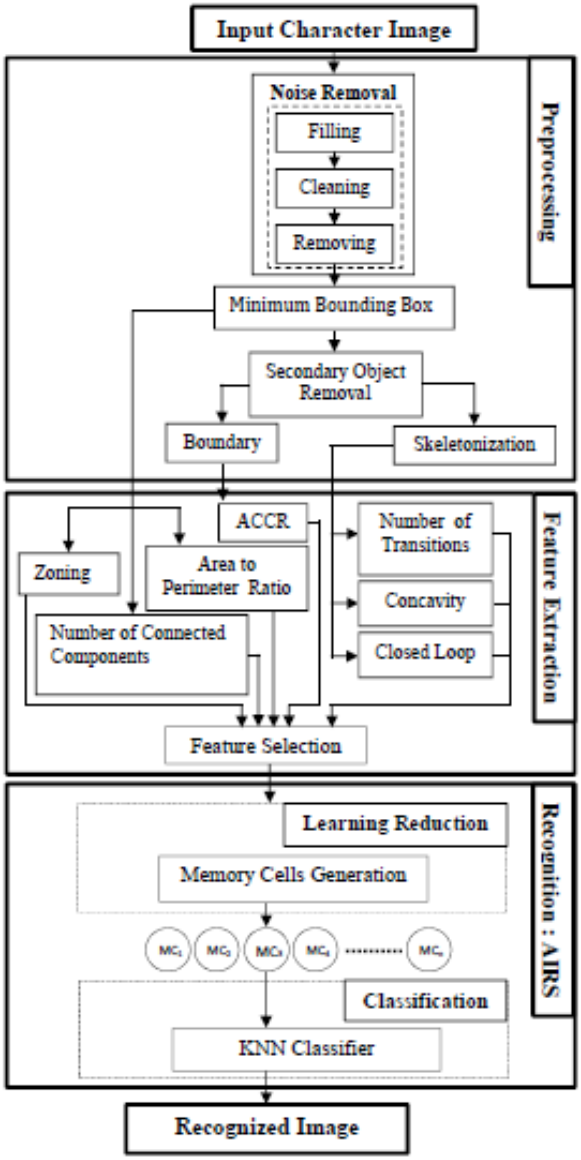


Figure 7**: (Boufenar *et al.* 2016) HWR model**

Bluche *et al.* (2017) proposed a mechanism system for end-to-end multi-line handwriting recognition and did not require any segmentation of the input paragraph. The solution is stimulated by the various attention models applied to recognize image. The principal difference between them and the advanced system is the implementation of the MDLSTM. The quality of this system is that it is the first proposed system for handwriting

recognition in the automated transcription externally a previous segmentation in lines. This is a huge important step in previous systems. The iterative algorithm determines the next attention feature on a sequence of reduced forms produced into the deep state of a recurring network. The system delivered high-grade results when referred to the IAM Dataset.

Moysse *et al.* (2017) introduced a method to realize the whole pages of the heterogeneous documents that can recognize also detect text in various languages. This design focuses on predicting the offset point (left) of the line. The endpoint (right) is predicted by 2D-LSTM. The whole text of the page is identified in a couple of steps: first, a neural network detects where to start, to recognize a text line. In the next step, the network offers the text recognition process including determines when to stop the process. In the neural network, the earlier grid detects the side effects for every text line, by predicting the state of the object's location coordinates as a regression query to recognize the entire text of the page. The localization of text lines depends on gradients with sufficiently convolutional neural networks and MDLSTM as contextual layers. Also, to improve the effectiveness of this system of compensation, just the left side of the text lines is demanded. This system has given great outcomes for recognizing the full text of the page on the heterogeneous Maurdor dataset.

Puigcerver (2017) introduced a design that depends exclusively on the convolutional neural network (ConvNets, CNN) and one-dimensional recurrent layers (1D-RNN). Figure 8 presented the layout of the system. He insists that he realizes results better or like the results obtained by applying MDLSTM-RNNs with quicker performance time because of the reduced estimate.

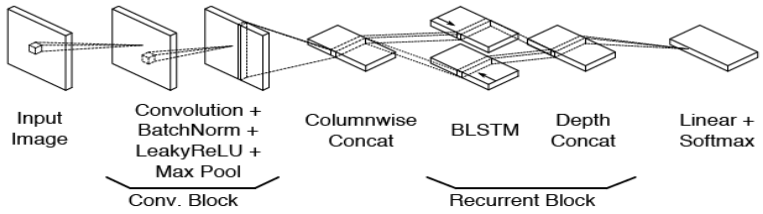


Figure 8: (Puigcerver *et al.*, 2017) Architecture

Belabiod and Belaïd (2018) offered an end-to-end system on the line segmentation employing the RU net. It is a complete system for word segmentation, applying BLSTM, and lastly add a CTC function. The system automatically detects the alignment among text line images and the words in the transcription.

Castro *et al.* (2018) introduced handwriting recognition at the line level. The system is based on (MDLSTM) inside the framework of the hybrid hidden Markov model (HMM). The advanced design transforms the regular MDLSTM structure by changing the position of the layers also the number of pooling layers. Unlike earlier systems, which concentrated just on the adjustment of the width of the network, the proposed design was estimated between the width of the network and its depth to obtain the most suitable topology. The entire system including a decoder including scientific data gives competitive results. Figure 9 shows the architecture of the system.

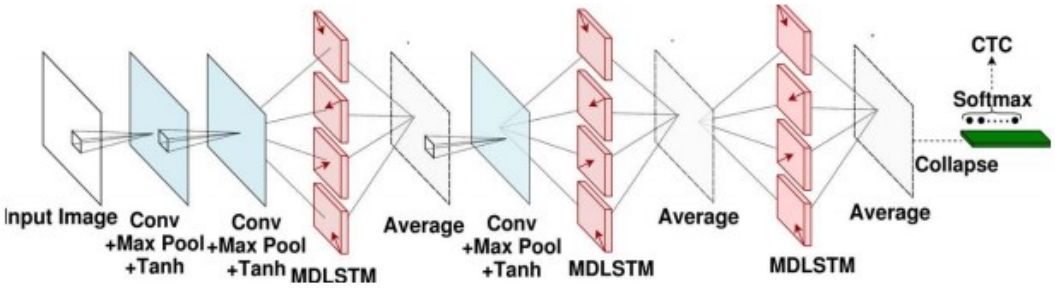


Figure 9: (Castro *et al.*, 2018) MDLSTM architecture

Chowdhury and Vig (2018) introduced an end-to-end neural network for recognizing handwritten text at the line level. The network consists of a deep CNN to extract features and the frequent encoder-decoder network with consideration. The encoder-decoder network is essentially applied to create the output flow of the input sequence. The system was prepared using Focal loss. The beam search algorithm was applied to improve the decoding ability of the design. Figure 10 describes the system.

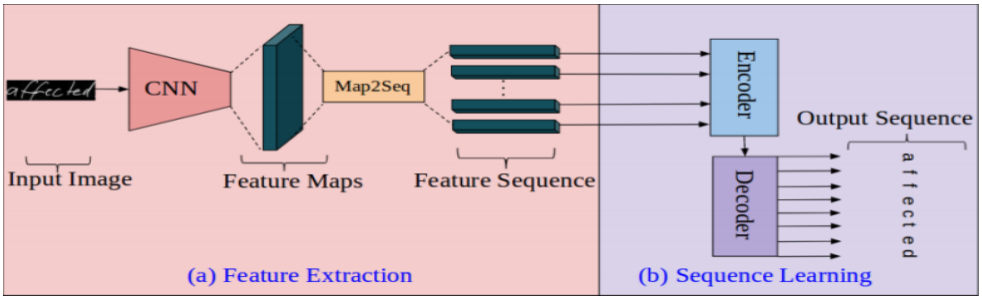


Figure 10: (Chowdhury and Vig, 2018) model Architecture

Dutta *et al.* (2018) presented a system that employed a hybrid CNN-RNN structure to recognize handwritten text. The system consists of a spatial transformation layer (STN), later a collection of residual convolutional (ReseNet18) blocks, supported by BLSTM stacks, and a linear layer for copying the labels. The STN is an end-to-end trainable layer that geometrically transforms inputs to improve handwriting distortions due to turning hand changes. The ReseNet18 is applied to learn the sequence of feature maps. The last ReseNet18 is delivered as input to BLSTMs as a series of feature vectors. The loss function is CTC. Figure 11 shows the architecture of the hybrid network CNN-RNN utilized in this system.

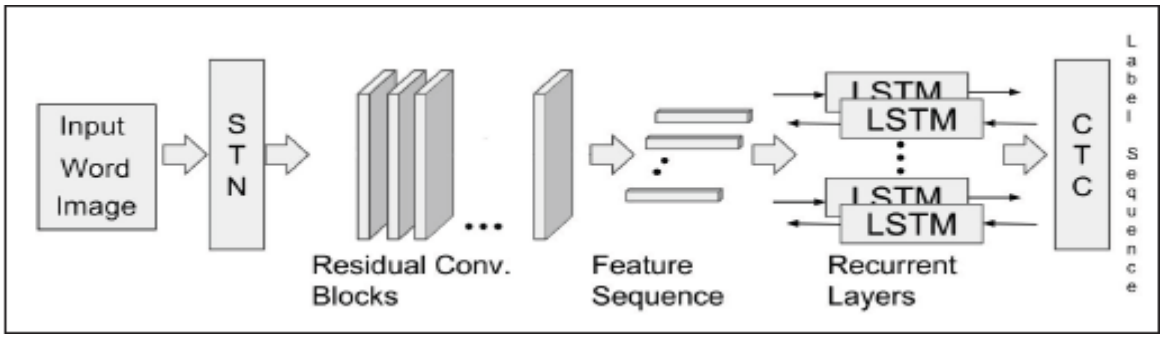


Figure 11: (Dutta *et al*., 2018) Architecture of Hybrid Network CNN-RNN

Ali and Suresha (2019) performed this paper to examine the automatic and suitable selection of classifiers and features collections for recognition of character in Arabic handwritten text. This manuscript examines a unique model that is focused on the fusion strategy. To complete this aim, the fusion of multiple classifiers and features has been completed.

This manuscript focuses on a combination of features and fusion of classifiers, introduced a system as presented in Figure 12 to understand handwritten characters of Arabic script with multi-fonts. This method integrates various features and classifiers to improve the recognition rate of classifiers.

It is the unique method proposed for the fusion of classifier and features for Arabic script recognition by recognizing different font types as igaza, sh roqa, farsi, naskh, and multi-fonts. In this system determined a classifier and features by the algorithm. The proposed work has been performed on AHCD plus IFN/ENIT datasets. The experimentation outcomes show that fusion procedures produce results better than conventional methods.

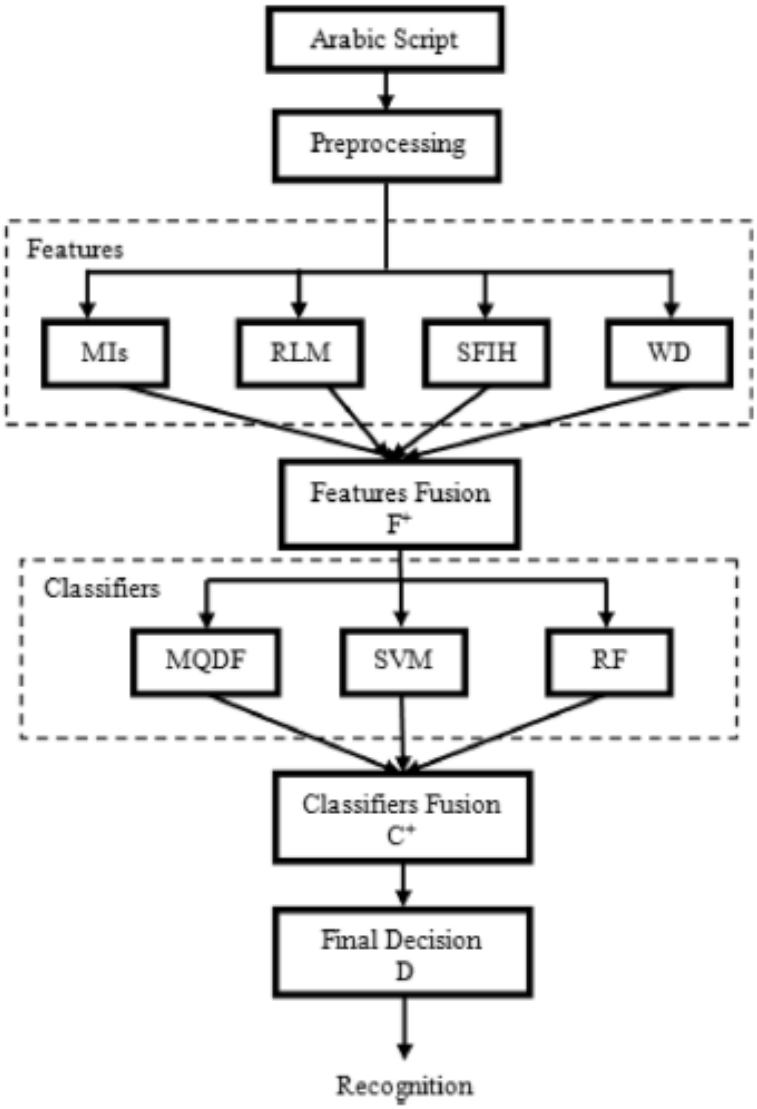


Figure 12: Feature and classifier level fusion (Ali and Suresha, 2019)

Sawalhah and Abandah (2020) produced a thesis that aims to obtain a system that applies end-to-end machine learning methods to recognize handwritten Arabic scripts. Hence, they have analyzed the possibility of applying the (SFR) system to recognize handwritten Arabic texts. The authors adjusted the SOL and LF networks. Also, they chose a large handwritten dataset in Arabic, (MADCAT) which is the multilingual automatic document classification analysis and translation dataset. They need to change the MADCAT dataset area truth information to be proper for the SFR Algorithm. Also,

they made many images preprocessing before employing the SFR system on the MADCAT dataset to identify the Arabic handwritten documents.

CHAPTER III

Technologies Used

While employing neural networks in various applications, several researchers have achieved astounding results (Jamro *et al.*, 2016). Neural networks hold pair kinds of learning algorithms: (1) supervised learning, and (2) unsupervised learning (Svozil *et al.*, 1997). In this design, we applied supervised learning because it could achieve a generic function depending on the training data and it would be capable to test the data reasonably. Many researchers applied end-to-end machine learning methods particularly deep learning to recognize handwritten documents in Latin language, the best technologies are: (1) Start, Follow, Read model (SFR) model, and (2) handwriting recognition model (HTR) model.

In this chapter, we shortly explain the technologies that we applied to recognize the handwritten Arabic documents. The main supplements for the models which we used for recognizing the Arabic HWR. And eventually describe the methodology and the networks that the system consists of them.

3.1 Start, Follow, Read (SFR) Model

The authors of this model present a deep learning model that simultaneously learns text detection, segmentation, and recognition utilizing frequently images without disclosure or segmentation explanations. Start, follow, read (SFR) model is made of a region proposal network to get the start location of text lines, a novel line follower network that incrementally tracks and preprocesses lines of (perhaps curved) text into text-line images fitting for recognition by a CNN-LSTM network. It can likewise be utilized in other areas of HWR. SFR exceeded first place in the ICDAR2017 handwriting recognition competition (Sawalhah and Abandah, 2020).

SFR is an end-to-end full-page handwriting recognition model that aims to identify offline deteriorating handwritten ancient documents, then convert them to editable text (Sawalhah and Abandah, 2020). SFR contains three networks:

1. Start-of-Line Network: to find the start position of lines of text. The SOL network is RPN, and the regions proposed in this network represent the beginning of the positions and directions of the lines in the text in a specific handle image.
2. Line Follower Network: starts at every position divined by SOL, then steps forward the text line, follows the lines of the text (possibly curved), and transforms them into relevant normalized text-line images.
3. Line-level HWR Network: this network predicts the transcription of the normalized text-line images.

Figure 13 shows how SOL, LF, and HWR networks work to control handwritten manuscript images. The red circles and arrows observe how the SOL network works. And the blue lines inspect how the LF network follows the text lines, and whereby it can manage the curved text lines and normalized them as explained in case 2. The transcriptions survey how the HWR network works (Wigington *et al.*, 2018).

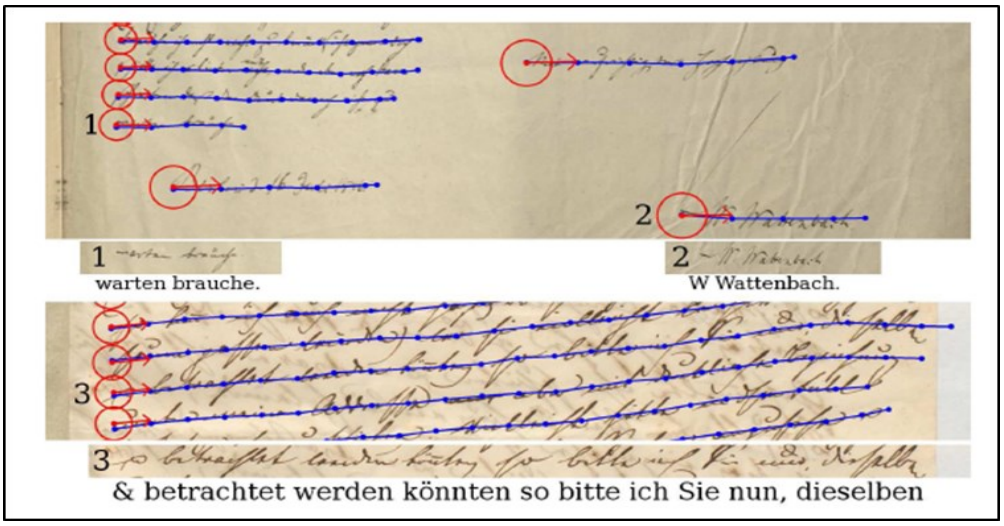


Figure 13: Examples illustrated how SFR recognize handwritten documents from READ dataset (Wigington *et al.*, 2018)

3.1.1 Major Contributions of SFR

The SFR model was chosen after searching in many systems. This model preferred to recognize Arabic handwritten documents following studying numerous systems that use machine learning methods to recognize handwritten documents in different languages, uniquely in the Latin languages (Sawalhah and Abandah, 2020). The SFR model achieves many contributions on the Latin handwritten documents and the most powerful additions are:

- Improved SOL network design, and it's a training system that improves recognition performance.
- LF network design, which can segment and normalize any pattern of the text lines.
- The mixed training of the three parts is mostly a set of images that include just transcriptions that support the SOL, LF, and HWR to adjust to all other and control each other.
- Evidence that LF and HWR networks can be applied to determine and clarify inherent SOL targets; this process needs only pre-training in a little number of images (e.g., 50) with extra hash marks (Sawalhah and Abandah, 2020).

3.1.2 Network Description

To simultaneously learn text disclosure, segmentation, and recognition, the SFR model presented with three parts: the start of line (SOL) network, the line follower (LF) network, and the handwriting recognition (HWR) network. We will describe the most important models for our work with their functionality.

3.1.2.1 Start-Of-Line Network (SOL)

The basic construction of the SOL network is the RPN network. RPN detects the

starting locations of the text lines. It demands any image with any size as an input and the outputs are a set of coordinates of x and y from a square object, all with a degree of interference. This helps to describe the complete convolutional network (Ren *et al.*, 2017).

RPN is the most reliable way to recognizing objects using Faster RCNN. The chief goal of RPN is to recommend multiple recognizable objects inside a given image. Figure 14 displays the faster R-CNN architecture and how it works. It includes a regression to determines the coordinates of the proposals. It's also having a classifier, which defines the probability of the start of the line having the target object (Karmarkar, 2018).

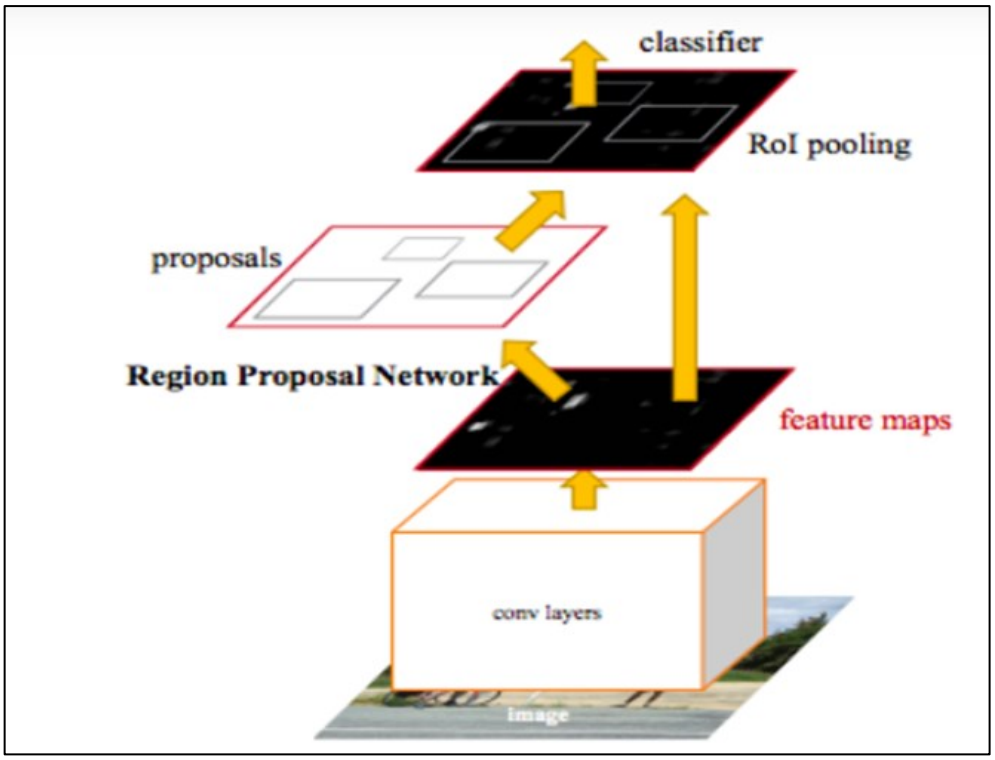


Figure 14**: (Karmarkar, 2018) R-CNN architecture**

the SOL network employed a trimmed deep convolutional network for large-scale image recognition (VGG-11) architecture rather than the MDLSTM (Kaggle, 2020). Therefore, to obtain the correct starting positions, the VGG-11 was integrated into the SOL network instead of MDLSTM (Moysset *et al*., 2017). As exposed in Figure 15, the

SOL network regresses the coordinates ($x0$, $y0$), scale($s0$), and rotation ($\theta 0$), as well as the probability of occurrence ($p0$). The scale and rotation equal the handwriting area and text line slant. The expected coordinates (x, y) describe the offset corresponding to the patch center (Wigington *et al.*, 2018).

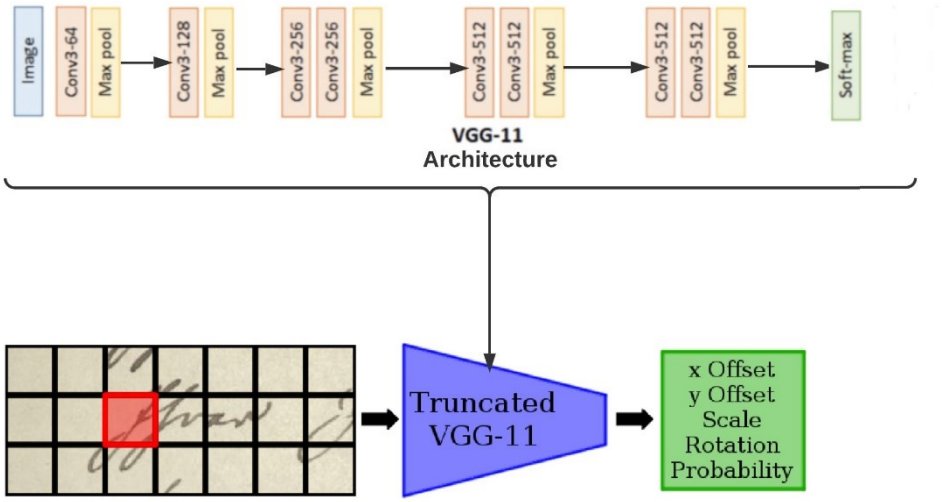


Figure 15**: The SOL network predicts x and y, scale s, and rotation angle θ (Wigington *et al.*, 2018)**

3.1.2.2 Line Follower Network (LF)

Following the SOL network get the start point of each line, the LF network starts to work. It follows the handwriting line in incremental steps and then gives output as a collapsed text line drawing be suitable for HWR. LF network is segmented polygonal areas and can arbitrarily level and following the curved text. The LF implemented the current location and rotation angle (xi, yi, θi). As displayed in Figure 16, describes a small display window, which is the red square in Figure 16a. It is then supplied to the CNN network for regression (xi+1, yi+1, θi+1). Figures 16b, 16c, and 16d explains that this process revolves up to the end of the image. Also, to discover the end of the text line, SFR practices the HWR network during the training process. The predicted SOL is

employed to determine the initial position and rotation and the SOL scale is furthermore utilized to determine the size of the display window and endures constant to the end of the line (Wigington *et al*., 2018).

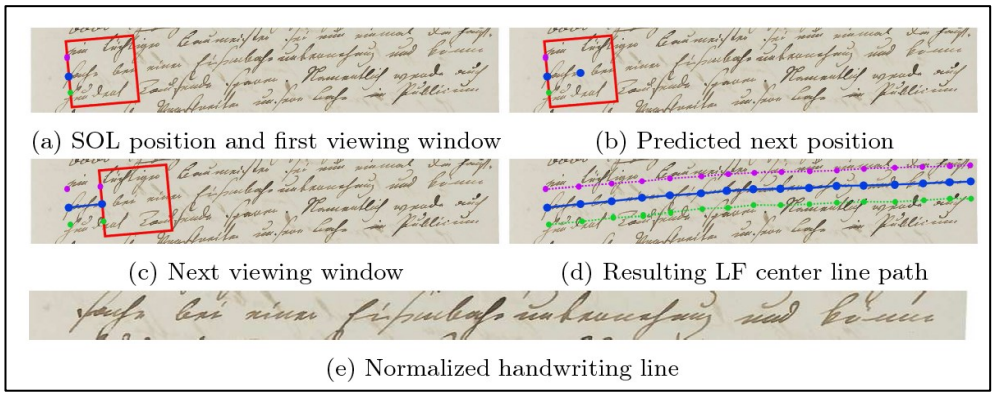


Figure 16**: The** LF **network (Wigington *et al*., 2018)**

In the next step, as presented in Figure 17, the input image is reshaped to get the display window using an affine transformation matrix as the input coordinates of the image is defined to display the image coordinates. This concedes the LF network to be backpropagated its failures by inspecting the windows. The first display window matrix ($W0$) can be determined as given in Equation 1 (Wigington *et al*., 2018). It consists of the mapping established using the transformative SOL matrix (W_{sol}) which is determined by the SOL prediction network values and the matrix of look-ahead (A)

$$W_0 = AW_{SOL} \qquad (1)$$

the (W_{sol}) can be measured as given in Equation 2 (Wigington *et al*., 2018):

$$W_{sol} = \begin{bmatrix} \frac{1}{s_0} & 0 & 0 \\ 0 & \frac{1}{s_0} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta_0) & -sin(\theta_0) & 0 \\ sin(\theta_0) & cos(\theta_0) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (2)$$

also, the value of (A) matrix can be calculated as shown in Equation 3 (Wigington *et al*., 2018):

$$A = \begin{bmatrix} 0.5 & 0 & -1 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3)$$

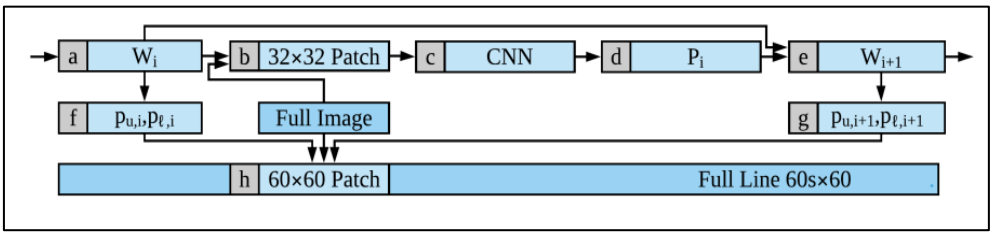


Figure 17**: (Wigington *et al.*, 2018) images workflow**

The look-ahead presents satisfactory context to LF network to accurately follow the lines. For every step (i), the SFR system selects a (32 × 32) display window by reconfiguring it utilizing the next display window (Wi). After reconfiguration, the coordinates of (x, y) in the area are normalized to the scale $(-1, 1)$. Given the correction of the display window $(i - 1)$, the regression of the network $(xi, yi, \theta i)$ is employed in the formulation of the prediction matrix (Pi). Wherever (Pi) can be determined as explained in Equation 4 (Wigginton *et al.*, 2018).

$$P_i = \begin{bmatrix} cos\,(\theta_i) & -sin(\theta_i) & 0 \\ sin(\theta_i) & cos\,(\theta_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (4)$$

next, we can measure (Wi) as given in Equation 5 (Wigington *et al.*, 2018):

$$W_i = P_i W_{i-1} \qquad (5)$$

To get the output image for the HWR network, the SFR system firstly predicts the path of the normalized handwriting line marked as a range of upper and lower coordinate's pairs, (pu, i, pl, i), which is the purple and green lines in Figure 16d. This is determined using the multiplier points down and middle of the predicted windows into their opposite transformations. Equation 6 explains how (pu, i) including (pl, i) can be determined (Wigington *et al.*, 2018).

$$p_{u,i}, p_{l,i} = \begin{bmatrix} x_{u,i} & x_{l,i} \\ y_{u,i} & y_{l,i} \\ 1 & 1 \end{bmatrix} = W_i^{-1} A \begin{bmatrix} 0 & 0 \\ -1 & 1 \\ 1 & 1 \end{bmatrix} \quad (6)$$

The handwriting line extracted using mapping of (pu, i, pl, i, pu, i+1, pl, i+1) to (60 × 60) patch corners. Then all these patches are connected to create a complete 60s × 60 handwriting lines where (s) is the amount of LF moves. As presented in Figure 18 the structure of the LF is consists of 7 CNN layers with (3x3) kernels, and the feature maps are (64,128,256,256,512 and 512) on the six convolution layers. After layers 4 and 5, batch normalization (BN) was applied, and subsequent layers (1, 2, 4, and 6) a (2x2) max-pooling (MP) was implemented. BN was utilized to normalize the activation in the middle layers of deep neural networks. BN is ideal in deep learning because it tends to develop accuracy and speed training (Bjorck *et al*., 2018). And lastly, a fully connected layer was applied in the LF network to regress the output (i.e., X, Y, θ) with initial of (X) bias parameters initialized to 1 and for (Y) and (θ) the biases initialized to 0 (Wigington *et al*., 2018).

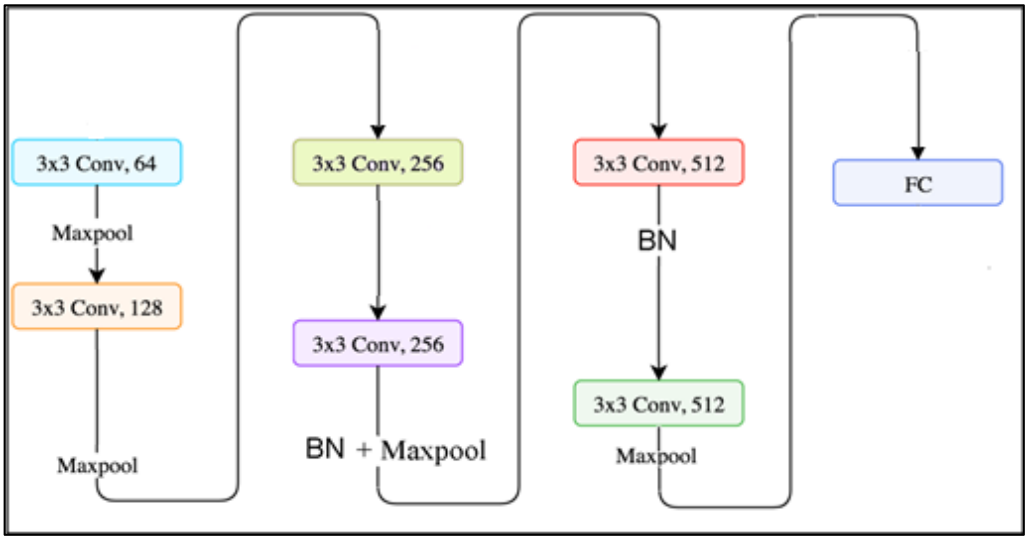


Figure 18: T**he LF architecture** (Sawalhah **and Abandah, 2020)**

3.1.3 End-to-End Training Through SFR Model

The training through SFR is end-to-end differentiable in that the CTC loss can backpropagate into the HWR and LF networks to the SOL network. End-to-end training is much slower and the SFR model is estimated on the 2017 ICDAR HWR full-page competition dataset of 1800s German handwriting, which has a couple of training sets. The initial set has 50 fully annotated images with line-level segmentations and transcriptions. The following set of 10,000 images has only transcriptions. This dataset is the most comprehensive and most challenging free HWR benchmark with 206,161 handwriting lines and 1,769,195 words. The SFR model gets results with CER equals 2.1 and WER equals 9.3 in RIMES (Wigington *et al*., 2018).

3.2 Handwriting Recognition System of Historical Documents (HTR) Model

In this work, the authors describes how to train an HTR system with some labeled data. Clearly, they trained a deep convolutional recurrent neural network (CRNN) system on simply 10% of manually labeled text-line data of a dataset and offer an incremental training scheme that includes the remainder of the data. The performance is additionally improved by increasing the training set with uniquely crafted multiscale data. They furthermore introduce a model-based normalization scheme that reflects the variability in the writing scale at the recognition stage. Also, they apply this method to the publicly available READ dataset like the SFR model (Chammas *et al*., 2018).

The state-of-the-art offline handwriting text recognition (HTR) systems serve at the line level by modifying the text-line image into a series of feature vectors. These features filled toward an optical model to recognize the handwritten text. New works on text discovery and localization at the document level, and joint-line segmentation and recognition at the paragraph level gave encouraging outcomes. Nevertheless, the most

immeasurable recognition results are furthermore performed by the systems working at the line level (Chammas *et al*., 2018).

The automatic segmentation of paragraphs into lines is even more challenging on historical documents. Supervised (or at least semi-supervised) paragraph segmentation is needed to label each text line to train an HTR system. When transcriptions are primarily provided at the paragraph level, the first challenge consists in aligning the training transcription data with the corresponding lines in the image. HTR model proposes to perform such an adjustment after training a first recognition system on a limited number of labeled data (Chammas *et al*., 2018).

The first system helps to bootstrap the whole method. HTR model also suggests growing the amount of data by generating multiscale artificial data to fully examine the scale factor in the test images. HTR system achieved the second-best result during the ICDAR2017 competition (Chammas *et al*., 2018). And as a code and dependencies that was keeping up to date with the developments of support on the latest python programming language.

3.2.1 CRNN System Discerption

HTR model is a deep convolutional recurrent neural network (CRNN) motivated from the VGG16 design utilized for image recognition. It utilizes a stack of 13 convolutional (3 × 3 filters, 1 × 1 stride) layers supported by three bidirectional LSTM layers including 256 parts through the layer. The individually LSTM unit produces one cell with enabled aperture connections. Spatial pooling (max) is applied following some convolutional layers. To launch non-linearity, the rectified linear unit (LeakyReLU) activation function was applied after every convolution (Chammas *et al*., 2018).

It has the benefit of being resistant to the fading gradient difficulty problems while being easy in the duration of computation and was determined to work strongly than sigmoid and tanh activation functions. A square-shaped sliding window is utilized to examine the text-line image in the direction of the writing. The height of the window is like the maximum of the text-line image, which has been obtained to be normalized to 64 pixels (Chammas *et al*., 2018).

The window overlap is equivalent to 2 pixels to provide constant development of the convolution filters. For every analysis window of 64 × 64 pixels in dimension, 16 feature vectors are obtained from the feature maps created by the current convolutional layer and filled into the measurement chain. It is deserving remarking that the number of feature vectors obtained from every sliding window is necessary. The number needs to be fair as to present a good sampling for the image. 16 feature vectors were determined to work best for HTR model architecture. Because for each of the 16 columns of the last 512 feature maps, the columns of length 2 pixels are concatenated inside a feature vector of dimension 1024 (512×2) (Chammas *et al*., 2018).

The HTR system is end-to-end trainable and the convolutional filters also the LSTM unit weights are consequently learned together inside the back-propagation method. The network in this model is manageable with a comparatively little number of parameters. The LSTM unit weights remained initialized to established strong work and help the model concentration stable. This enables the network to keep a constant variation across the network layers which holds the signal from collapsing to a huge value or fading to zero. The weight matrix Wij was initialized with a stable distribution given as Wij $\sim$ U $(-\frac{\sqrt{6}}{n}, \frac{\sqrt{6}}{n})$, where n is the absolute number of input and output neurons at the layer (Chammas *et al*., 2018).

Adam optimizer was employed to train the HTR network with an original learning rate of 0.001. This algorithm could be evaluated as an upgrade version for RMSProp, submitting bias correction and momentum. It provides adaptive learning rates for the stochastic gradient descent update computed from the first and second moments of the slopes. It additionally saves an exponentially fading average of the prior squared angles and the past angles (Chammas *et al*., 2018).

Batch normalization was combined following every convolutional layer to stimulate the training manner. It essentially controls by normalizing per batch by both mean and variance. The HTR network was trained in an end-to-end fashion with the CTC loss role. A token-passing algorithm was applied for decoding. It combines a bigram writing model with qualified Kneser-Ney reducing, created from the available training data. It is deserving remarking that no preprocessing is required. The system operates immediately on new images (Chammas *et al*., 2018).

3.2.2 Incremental Training with Less Training Data

Few labeled text lines are required to bootstrap the training process in the HTR model. HTR network practiced an automated segmentation algorithm to obtain line images from the paper images. The algorithm chooses aspirant baselines by examining contours distribution. It then allows every contour to one of the baselines based on several criteria, compared to the average distance among two lines and the distance among the contour center and the line. Just 10% of the pages were manually established, getting certain the line segmentation is correct and managed to bootstrap the training process. Besides the 10,000 training pages, 50 commented pages at the line level were implemented throughout the ICDAR competition and were employed for validation in the training process (Chammas *et al*., 2018).

The following action is the system remained determined to identify the remaining 90% of the segmented line images in the training set. The acknowledged lines were mapped to lines in the ground-truth data for every page, based on the levenshtein distance among the text lines. Mapping is recognized valid when the edit range is less than or equal to half the length of the source line. Understanding this manner, and according to this threshold on the levenshtein distance, 80% of the provided text lines were chosen to retrain the system, while the rest (20%) were rejected. The process could have been restarted following having trained the system with the new data or even emphasized. An enhanced recognition administration could have produced more extra training lines. Nevertheless, HTR noticed that most maximum of the discarded line images in the first repetition was produced from incorrect segmentation (e.g., two text lines in a single image, cropped text-line, *etc*.), because the algorithm is susceptible to the writing skew. Consequently, more excellent segmentation algorithms are required to develop the training process. The entire process can be reviewed in Algorithm 1 (Chammas *et al*., 2018).

Algorithm 1 Incremental alignment process

Require: $TrainSet$: Set of all training pages
Require: $RefText$: Ground-truth text paragraph for each page
  for each page P in $TrainSet$ **do**
    $Lines[] \leftarrow Segment(P)$
    $RefLineIndex \leftarrow 0$
    for each line L in $Lines$ **do**
      $RecSeq \leftarrow Recognize(L)$
      while $RefLineIndex < length(RefText[P])$ **do**
        $RefSeq \leftarrow RefText[P][RefLineIndex]$
        $EditDistance \leftarrow Levenshtein(RecSeq, RefSeq)$
        if $EditDistance < 0.5 \times length(RefSeq)$ **then**
          $Map(L, RefSeq)$
          $RefLineIndex \leftarrow RefLineIndex + 1$
        end if
      end while
    end for
  end for

3.2.3 Integration of Multi-Scale Training Data

To additional intensify the performance of the system, the HTR model utilized the variability in the writing scale to expand the training set with text-line images at various scales. Based on the vertical scale average, the training lines held first categorized into 3 categories (Large, Medium, and Small) via Jenks natural breaks optimization algorithm as mentioned in Figure 19. To determine the scaling portions by which a certain image of a given rule class is increased or decreased, the average scale grade score is estimated on the data (Chammas *et al*., 2018).

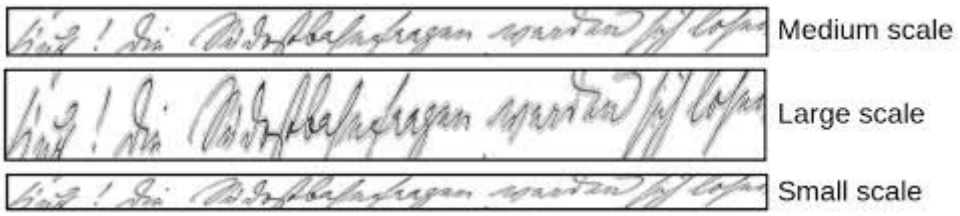


Figure 19: An illustration of the READ dataset wherever a text-line listed as medium scale is transformed into large- and small-scale transcriptions (Chammas *et al*., 2018)

3.2.4 Model-Based Normalization Design

To enhance the performance of the HTR model, the authors intended to reflect the variability in the writing scale in a model-based normalization system, where the test data are adjusted to adequately match the kernel model. Overall, analyze the recognition phase wherever a test image represented by a particular variability is presented at the input of a system qualified on a usual training set (Chammas *et al*., 2018).

According to the statistical decision theory, the recognition responsibility identifies the common likely word sequence given the measurements as:

$$\hat{\delta} = \arg\max \Pr\left(\delta \middle| \underline{\underline{X}}\right) \qquad (7)$$

where (s) represents a word order, and X is the measurement order. To cope with a variability factor θ in a test image, it is assumed that a transmutation Tθ(.) exists with contextual parameter vector θ allowing to overcome this variability to a shadow. It is expected that this parameter is esoteric and cannot be included (Chammas *et al*., 2018). A normalized variant of the input image X can be described as:

$$\hat{\underline{\underline{X}}} = T\theta(\underline{\underline{X}}) \qquad (8)$$

considering the contextual parameter vector θ refers to a limited set, Equation 7 can integrate the normalization described in Equation 8 to become:

$$\hat{\delta} = \operatorname{argmax} s \sum_{\theta} \Pr(\delta|T\theta(\underline{\underline{X}}))\Pr(\theta|\underline{\underline{X}}) \qquad (9)$$

For all potential normalizations of the input X, the system provides clarifications with the identical scores, regarded as next probabilities. A mixture of the scores allows to re-select the optimal clarification as shown in Figure 20. This is recognized as an estimate of the right-hand term of Equation 9. HTR model generated many versions of the test data by vertically scaling every text-line image to multiple scales (0.7, 0.8..., 1.3) (Chammas *et al*., 2018).

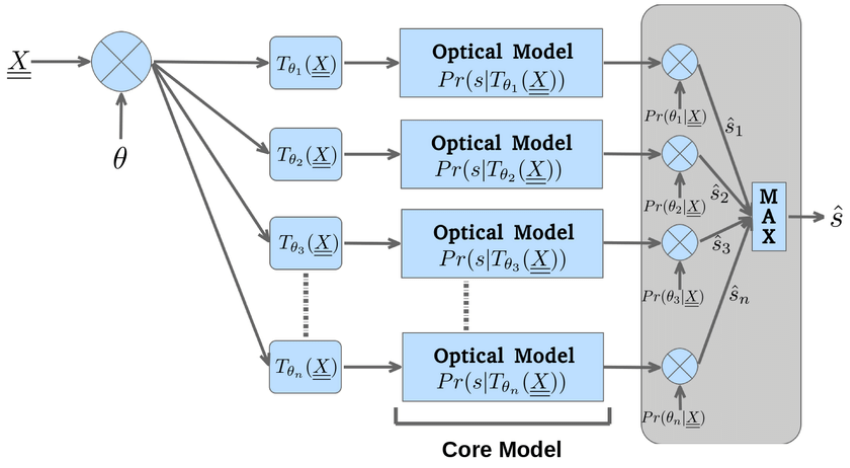


Figure 20**: (Chammas *et al*., 2018) Model-based normalization scheme**

3.2.5 HTR Model Results

HTR model achieved the second position in the ICDAR2017 competition, with relative review to the winning system (SFR model), while remarking that the overall performance depends on both segmentation and recognition responsibilities. The HTR model gets results with CER equals 7.01 and WER equals 19.06. The results can be increased by developing the segmentation algorithm which will allow using of more training data (Chammas *et al*., 2018).

CHAPTER IV

Dataset Preparation

Most machine learning algorithms require data to be formatted in a highly specialized way, so datasets frequently need some amount of preparation before they can generate valuable insights. This chapter attempts to introduce, study, investigate, and preprocess the MADCAT dataset to be fitting for our work. Dataset preparation is the second step of the seven steps developed in end-to-end machine learning behind the data aggregation process (Géron, 2017).

4.1 Getting, Studying, and Analyzing The MADCAT Dataset

At first, we need to describe the dataset so that the rest of our steps are clear. Our experimental data was from the MADCAT dataset. MADCAT dataset includes several styles of handwriting Arabic documents.

4.1.1 What is The MADCAT Dataset?

The MADCAT (multilingual automatic document classification analysis and translation) training collection includes all training data created by the Linguistic Data Consortium of the DARPA MADCAT Program. The data consists of handwritten Arabic documents, scanned at high resolution, and annotated for the X Y coordinates per line and word. They presented digital records and English translations of every image, including the various content and annotation layers integrated into a single MADCAT XML output (Abandah and Al-Hourani, 2018).

The data production process for MADCAT begins with Arabic source documents in three genres: newswire, weblog, and newsgroup text, basically raised as part of the DARPA GALE Program. Then, Arabic-speaking writers copy the record by hand, following specific guidance as to the writing style (fast, normal, careful), writing

implement (pen, pencil), and paper description (lined, unlined). Also, each resulting handwritten page is assigned to up to 5 independent writers, using different writing conditions (Abandah and Al-Hourani, 2018).

Once handwritten data has been collected from the writers, it's marked for quality and completeness, then each page is scanned at a high resolution (600 dpi, greyscale) to build a digital version of the handwritten document. The scanned images are later annotated to symbolize the X Y coordinates of each line. The specific reading scheme is also labeled, as are any errors devised by the writers when copying the text (Abandah and Al-Hourani, 2018).

There are 3 Phases for the MADCAT. The releases include 42,490 notes files of MADCAT XML format along with their corresponding scanned image files in TIFF format. MADCAT dataset have many problems like: (1) slant texts, (2) different types of line spacing (3) sparse and dense noise distributions, and (4) skew lines. Figures (21, 22 and 23) shows different examples of MADCAT documents have variant problems (Abandah and Al-Hourani, 2018).

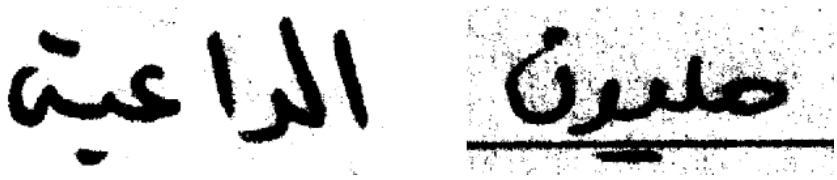


Figure 21: Sparse and dense noise distribution examples from MADCAT dataset (Abandah and Al-Hourani, 2018)

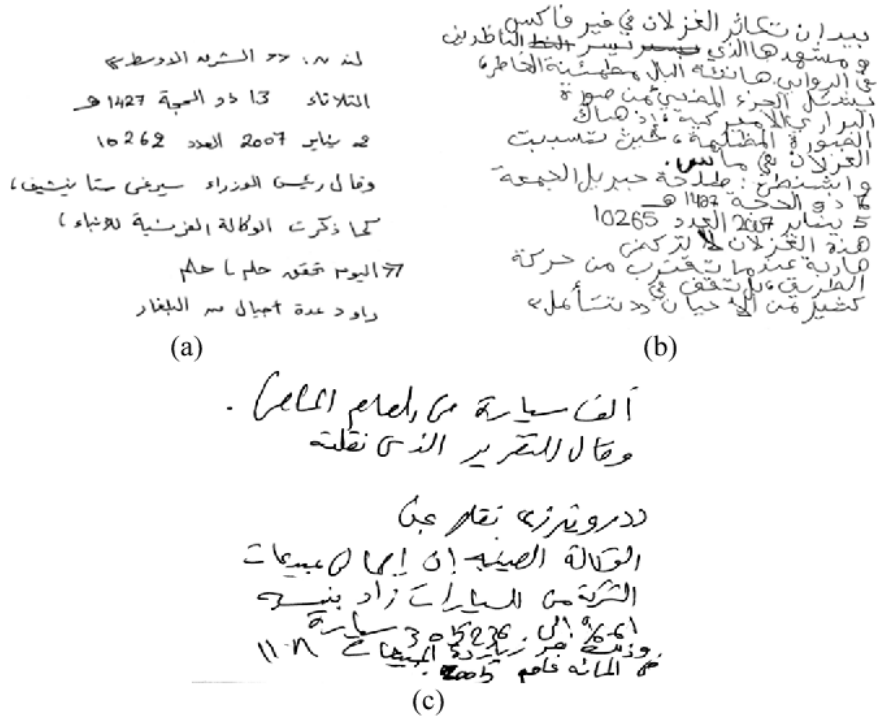


Figure 22: Different types of line spacing example from MADCAT dataset (Abandah and Al-Hourani, 2018)

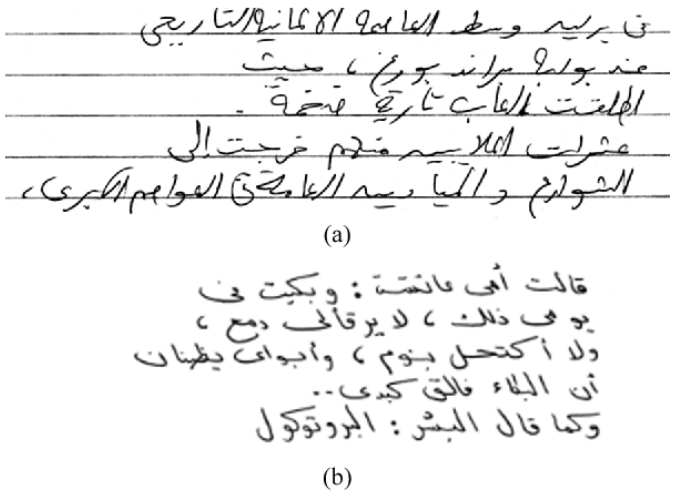


Figure 23 : Slant texts example from MADCAT dataset (Abandah and Al-Hourani, 2018)

4.1.2 Problems of MADCAT Dataset

MADCAT dataset holds multiple difficulties in its samples such as vertical lines, external graphical elements, pepper noise, irrelevant text, or numerals (nonArabic

characters), varying text body, short text lines, slant and skew texts, and bleed through text (Abandah and Al-Hourani, 2018).

There are many difficulties that we encountered when we decided to use the SFR system to realize the Arabic handwritten documents start lines and to perform the MADCAT dataset harmonious with it like:

- Lot of noise in the images in a MADCAT dataset.
- There are many differences between READ and MADCAT datasets. As the format of the images and the extensible markup language (XML) records of the MADCAT are different than the READ dataset and necessary to be modified to be equivalent to the form of the images and XML files in the READ dataset.
- The direction of reading documents. The document's direction is from left to right in the READ dataset in the German language; while in the MADCAT dataset, it is read from right to left in the Arabic language (Sawalhah and Abandah, 2020).

4.2 Prepare Data for Start, Follow, Read Algorithm

We made many pre-processing developments in the images and XML records of the MADCAT dataset to make them fit for the proposed system. Figure 24 describes the steps that we done.

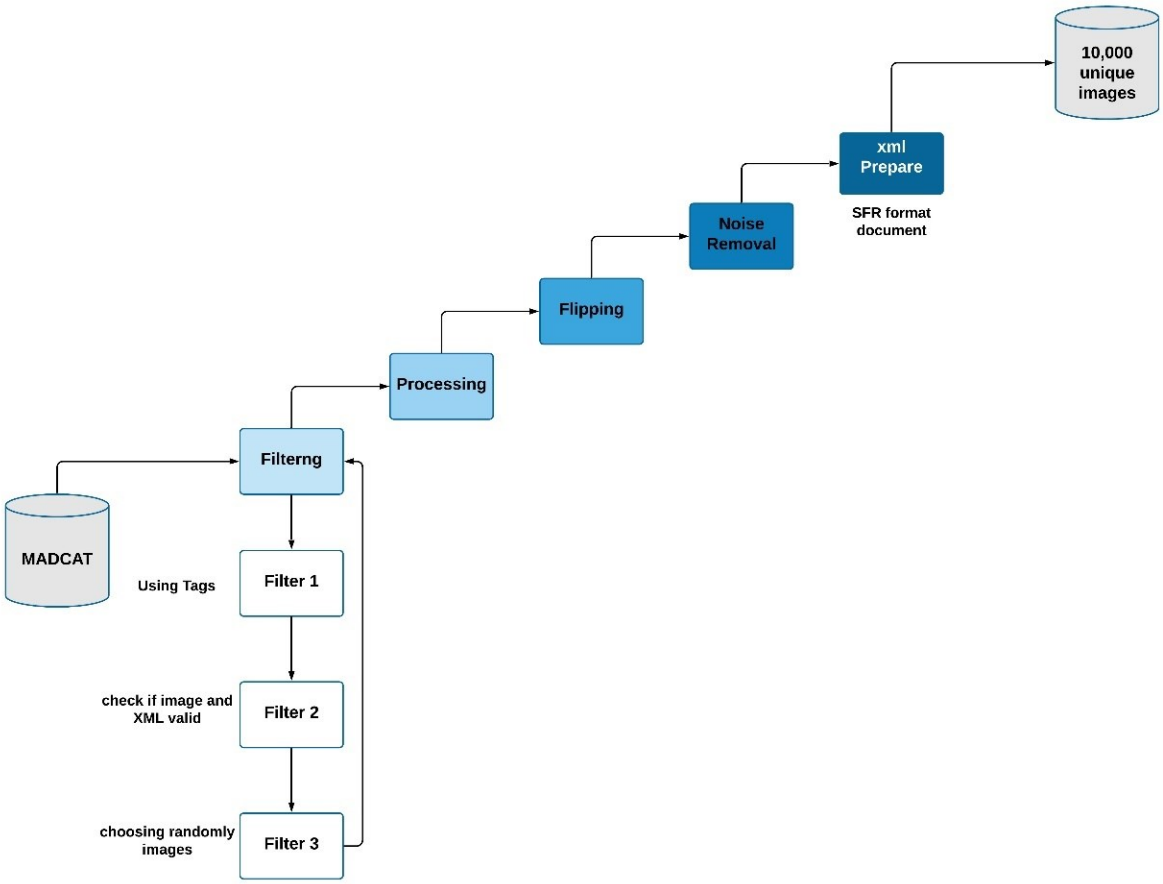


Figure 24: Preparation steps flowchart for MADCAT dataset

4.2.1 Filtering The MADCAT Dataset

After get all the data of MADCAT dataset, we started by filtering the data set to reach the final 10,000 images which will be ready for training our models by applying three filters on the dataset as the following:

(1) **Filter 1.** Filter the images using tags:

- IUC: write the document using pen, unlined page, and careful speed writing conditions
- IUN: write the document using pen, unlined page, and normal speed writing conditions.

In total, we have 22,682 images from the three phases that have “IUC” and “IUN” tags.

(2) **Filter 2.** The aim of this filter is to check the valid images and if the XML design exists. After that, we need to flip the images, because the images are read in the Arabic language from right to left. While the SFR model expected that the images are seen from left to right because it is designed to recognize the Latin language. So, we flipped the direction of the images on the MADCAT dataset to be fit with the SFR system using the horizontal flip (techtutorialsx, 2019). The results we have are 13,774 flipped images with their XML design from MADCAT data.

(3) **Filter 3.** We need to choose randomly from the 13,774 flipped images 10,000 with their XML files, to ensure that our pre-processed dataset has multiple styles of Arabic handwriting patterns.

4.2.2 Image Processing

As we mentioned before, the images collected in the MADCAT dataset are in the tagged image file format (tiff) with dots per inch (DPI) equal to ($600x600$) dpi. But the SFR model requires that the images relinquished to it are in the joint photographic expert's group (JPG) format. So, we used a python code to convert the images in the MADCAT dataset to be cooperative with images practiced in the SFR system. This code uses the Python imaging library (PIL). It is a package from a Python script that contains many functions for processing images (geeks3d Page, 2020). The code in appendix A defines the functions that we used to transform the MADCAT images to be fit with the images used in the SFR system (Sawalhah and Abandah, 2020).

4.2.3 Noise Removal

To advance the system's effectiveness, we implemented some image processing operations to remove the noise and we use the median filter. Median filter is known as a non-linear digital filtering technology. The median filter is normally used to eliminate noise from signals or images, and it's widely used in digital image processing as it maintains edges under certain conditions during noise removal (Sawalhah and Abandah, 2020). How do you do median filtering?

The median filtering process is accomplished by sliding a window over the image. The filtered image is obtained by placing the median of the values in the input window, at the location of the center of that window, at the output image. The "medianBlur" function from the Open-CV Library can be practiced achieving a median filter (Geeksforgeeks Page, 2020). The code at appendix A represents the function that we handled to remove noise from the MADCAT images (Sawalhah and Abandah, 2020).

4.2.4 Processing Extensible Markup Language (XML) Files

The data in XML is formatted hierarchically, it presents the information as a tree (Python page, 2019). The central value of the tree structure to XML is that it performs navigation, correction, and removal programmatically manageable. The XML files are generally used in web-scraping and general practice in structured document analysis (Sawalhah and Abandah, 2020).

Also, the XML files contain many sections called elements, known by the beginning (<) and end (>) tag. The value pair attributes represent the name inside the start tag or the empty element tag. An XML attribute can contain only one value and each attribute can appear once on each element (Steph Howson, 2018).

Therefore, we employed the previous information in our work to extract the necessary information and convert it from MADCAT XML format to the required format for the SFR algorithm. To make XML files in the MADCAT dataset compatible for formatting XML files in Train-B set in the READ dataset, we have developed a Python code depends on several libraries like ElementTree and MiniDom.

These libraries implementation is efficient, simple application programming interface (API) for analyzing and creating XML data, they contain functions for analyzing and managing XML files and additional similar structured files (Python Page, 2019). The resulting code represents what we used to regenerate the XML files on the MADCAT dataset to be cooperative with the form of the XML files used in the SFR model. Firstly, we divided the desired XML file design into 4 terms:

1. **Head and tail of the XML.** They have the first collection of XML tags that we don't need to change.

2. **The Region of text.** It contains the tags that describe the whole text region.

3. **The text lines.** It contains the tags of every line and the information for it like the coordinates and Baseline, with the Arabic text of the line. This region has a varied number of tags depends on the number of text lines in the image. Also, the coordinates of MADCAT are based on the image read from right to left, so we need to flip the coordinates. Overall, we must decrease the coordinates according to the previous program so that the dimensions of the image match the coordinates of X and Y inside the XML file.

4. **Full text region.** It contains the tags of the full text of the image. Figure 25 describes the workflow of the code.

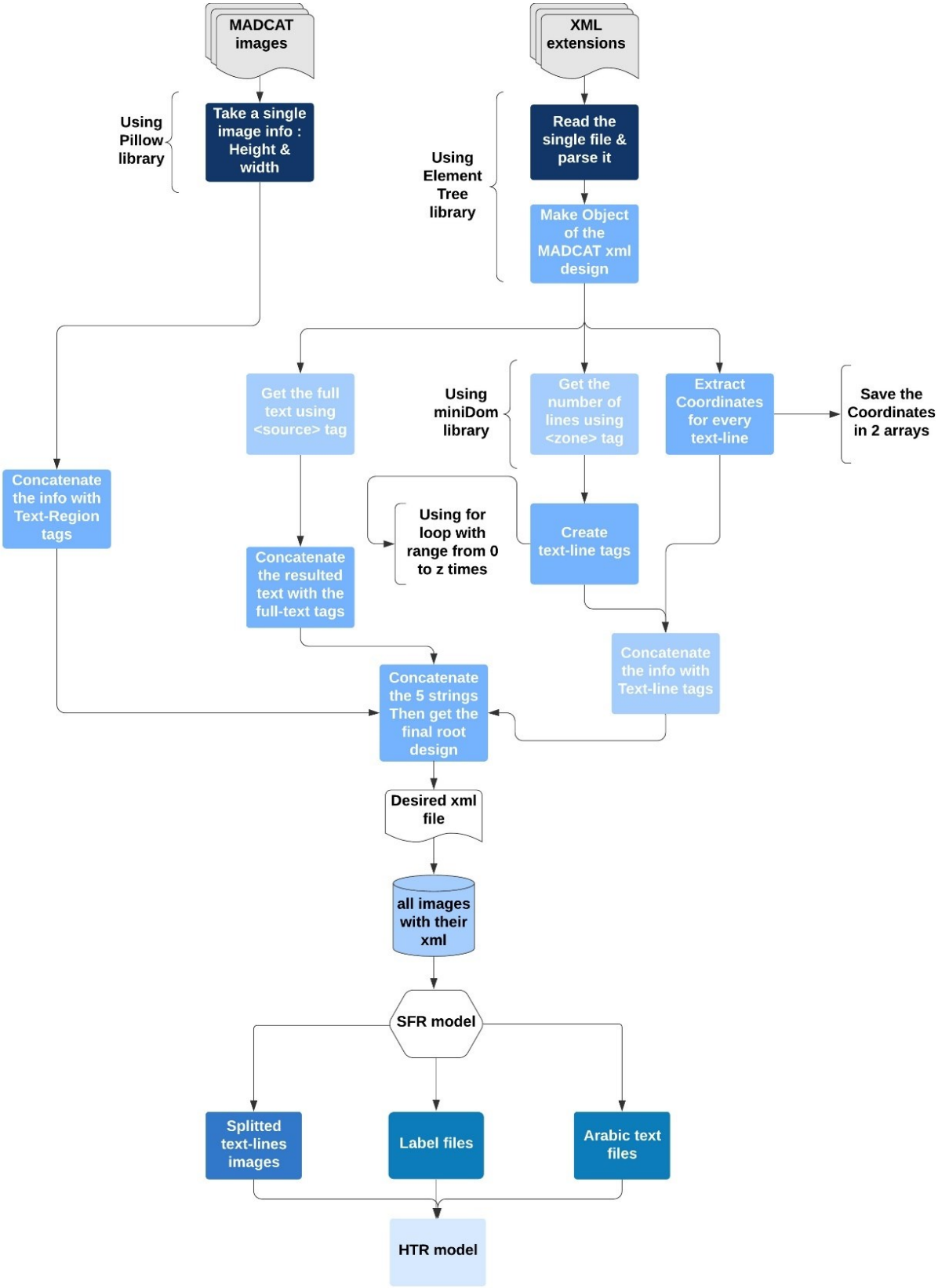


Figure 25: XML Preparation steps flowchart for MADCAT dataset

Consequently, the dataset has been prepared for Arabic handwritten documents containing 10,000 images have multiple styles in writing conditions. The code at appendix A represents the functions and libraries that we used to get the right XML file for each image in our final dataset.

CHAPTER V

Experiments and Results

Our proposed model for recognizing Arabic handwritten documents depends on two models: (1) the SFR model and (2) the HTR model. These models are applied to recognize historical handwritten documents in the Latin language. So, we decided to combine the two models to produce an integrated algorithm to recognize our dataset. This chapter tries to describe how we merge these models together to give us best results, the working set that we employed to complete our design, and the measures we employed to estimate our design. Also, analysis concerning the results we achieved when we performed our proposed model on the MADCAT dataset was presented.

5.1 General Idea of the Proposed Algorithm

Our algorithm is a result of the combination of two algorithms that achieved amazing results in the READ dataset for the Latin language. They are SFR and HTR. As mentioned before, we prepared the dataset to make the SFR accept it. Now, we will make the SFR algorithm work to receive the full image and then determine the beginning of each line in it and identify it through the first model, which is the SOL model.

And as a second step, the results will be transferred to the second model to trace, cut, and normalize each line to the beginning of each line using the LF model. Then we will export these results by extracting all the text for each text-line image with its ground truth table and converting it to the last stage, which is the stage of highlighting the image and converting it to text that can be edited in the Arabic language using HTR model. Figure 26 briefly explains what the expected model will go through in a simplified form.

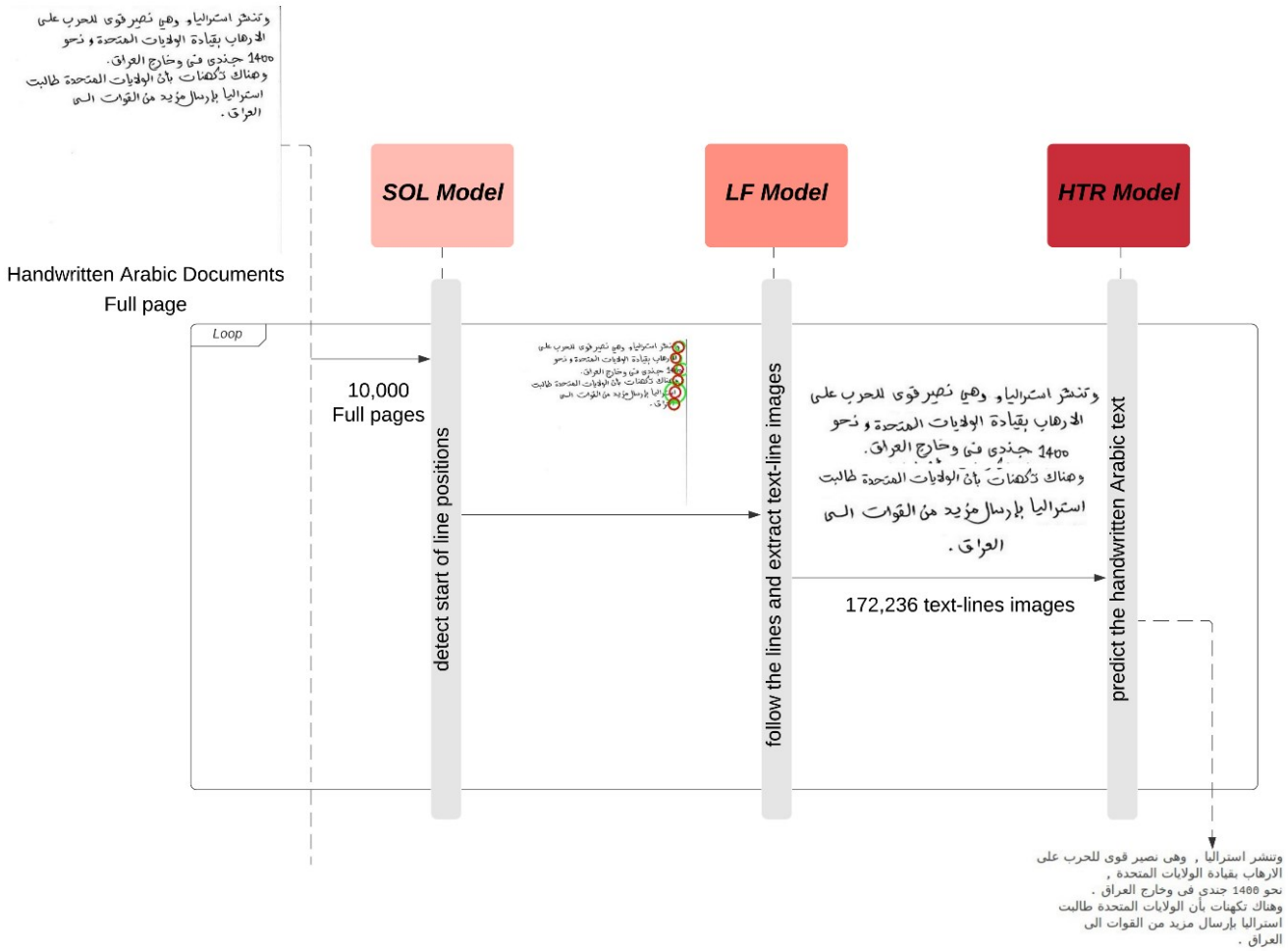


Figure 26: The general idea of the proposed model

5.2 Work Environment

Our proposed design was divided into two parts in the field of the environment used in our experimental work. The first environment contains the first two models, namely, the start of the line model (SOL) and the line follower model (LF) with an environment: a workstation with Intel(R) processor Core-i7 9460 CPU 3.20 GHz; it has an 8GB DDR3 RAM, Ubuntu 18.04.2 LTS amd64, and 64-bit operating system. We have used GPU Gtx-1050 to speed up the training rule. The application was implemented using Python 2.7 language on the PyCharm Professional IDE.

And the other one that has been trained in the last built-in model, which is the HTR

model with an environment: a workstation with Intel(R) processor Core-i7 6400 CPU 3.40 GHz; it has a 32GB DDR3 RAM, Ubuntu 18.04.5 LTS amd64, and 64-bit operating system. We have used GPU GeForce RTX 2080 to speed up the training rule. The application was implemented using Python 3.7 language.

5.3 Loss and Character Error Rate Evaluation

To evaluate the development of our proposed model in training and testing, we applied the loss as a benchmark for assessing the machine. Loss is a technique used for evaluating the 37 qualities of the designated algorithm for the data manifested. If forecasts differ too enormously from original results, the loss function hacks too many. By using the optimization function, the loss function learns to continuously decrease the prediction failure. No individual loss function is suitable for everyone in machine learning algorithms. There are many constituents included in determining the loss function for a particular query before-mentioned as the kind of machine learning algorithm taken, the rest of computing derivatives, and to unusual measure the rate of outliers in the dataset. There are a couple of types of loss functions according to the kind of learning: (1) task classification losses and (2) regression losses (Ravindra Parmar, 2018) and this amount applies to the three models.

In this work has been applied regression loss to evaluate the three networks. Loss functions measure whereby far an expected value is from its true value. A loss function maps determinations to their associated values. Loss functions are not fixed, they vary depending on the job in support and the purpose to be satisfied. The model, while learning on the training dataset, calculates the loss rate, and based on the stop after no improvement parameter, which is sets before, it stops learning (Lund *et al*., 2013).

It is worthwhile to point out that one of the best ways to evaluate the based model

for text recognition in addition to the regression loss scale is the Character Error Rate (CER). Character error rate (CER) is a popular metric of the performance of an automated text recognition system in the ML. CER is comparable to Word Error Rate (WER) and Sentence Error Rate (SER) but uses on character instead of word or sentence (Lund *et al*., 2013). The character error rate can be measured using the following equation:

$$\mathrm{CER} = \frac{(\mathrm{S} + \mathrm{D} + \mathrm{I})}{\mathrm{N}} = \frac{(\mathrm{S} + \mathrm{D} + \mathrm{I})}{(\mathrm{S} + \mathrm{D} + \mathrm{C})} \qquad (10)$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, C is the number of correct characters, N is the number of characters in the reference.

CER's output is not forever a number among 0 and 1, in critical when there is a huge number of inserts. This rate is often correlated with the percentage of characters that were wrongly predicted. The more under the state, the greater the achievement of the system with a CER of 0 being an ideal summary. So, we implemented the CER metric in the last model (Ravindra Parmar, 2018).

5.4 Proposed Model Implementation

In continuation to the previous part, we will explain here the most important basic steps to run and extract results from the codes that we used. Then our attempts to fine-tune the model and develop its results to get the best results for reading handwritten documents in the Arabic language.

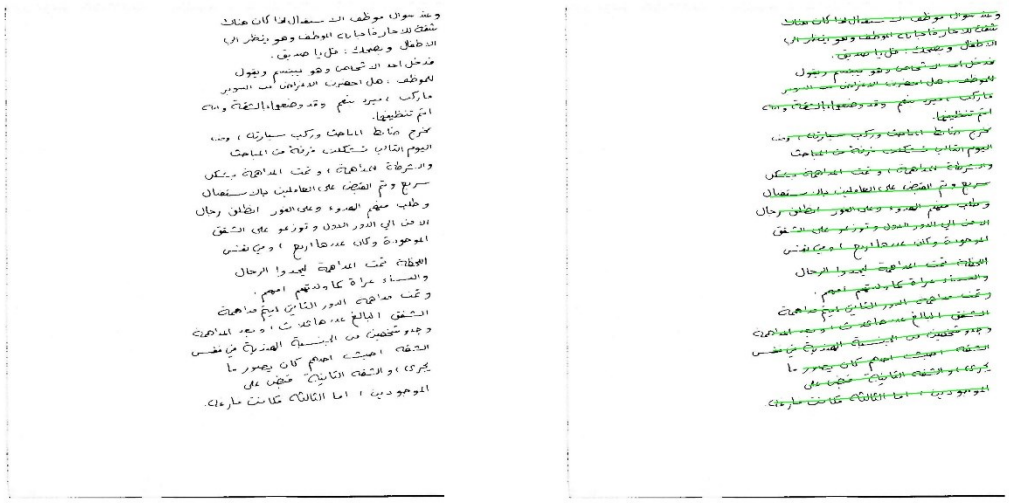
5.4.1 Start, Follow, Read Implementation

To develop the SFR system on our MADCAT dataset, we must complete rare operations before we get the results. The following is an explanation of the steps that were taken during the pre-preprocessing and pre-training steps:

Step 1: Using Anaconda environment on PyCharm IDE, installed all the dependencies available on environment. yaml as follows: (Start follow read github Page, 2020).

Step 2: Installed the warp-ctc library, after Pytorch 0.4.1 implementation as a main library for training and testing the model. (SeanNaren github Page, 2020).

Step 3: Dataset Preprocessing. At this stage, we extract the coordinates of the beginning of the line and the coordinates of the line tracking with its content. This stage is slow but essential to prepare the basic coordinates for each model to learn them and then take a test through the test dataset to make sure of the extent of its learning. Figure 27 shows an example of the output sequence that we get when we utilized to preprocess step on the test data of the MADCAT dataset.



(a) MADCAT document example

(b) GT baselines results by applying preprocessing step

Figure 27: Sample of applying preprocessing step on the MADCAT dataset

Step 4: Pre-Training Step. This stage reviews the effects of the first stages of implementing the SFR model when it is employed to verify the Arabic handwritten documents in the MADCAT dataset.

We did the pre-training process only because the pre-training process is like the real training process, but its goal is to deliver the model to a certain percentage of learning

with a very small number of data sets compared to its actual size. And because it was enough with its amazing results, we didn't need to apply the full training to the models we needed. Besides that, in our various experiments, when using a small number of images for training and the rest of the data for testing, we get results in less time by half. And what matters to us in work in addition to good results is the short time.

We used a small number of images (100 images) with additional segmentation labels in the pre-training phase, Previous experiences have proven that the model does not require a high percentage of data to learn. A small percentage of the data is sufficient for it to perform and then be ready to extract the beginning of any line from any document. After that, we give the trained model all the documents (i.e., 10,000 images) to extract the start of the line for all documents.

As Swalhah and Abandah mentioned before in their previous work on the SFR algorithm. They did many tests until they got the most suitable way to perform the SFR system ready to accept Arabic documents and to find the best configurations that would present the most beneficial results. They chose to change the direction of reading text in the SFR system or change the direction of reading in the images by flipping them on the MADCAT dataset.

Step 5: Start of the Line model (SOL). SOL is the RPN as the proposed regions are the start positions and directions of text lines in the form of a specific image.

Equation 11 shows how we can calculate the loss function which is used to evaluate the SOL network:

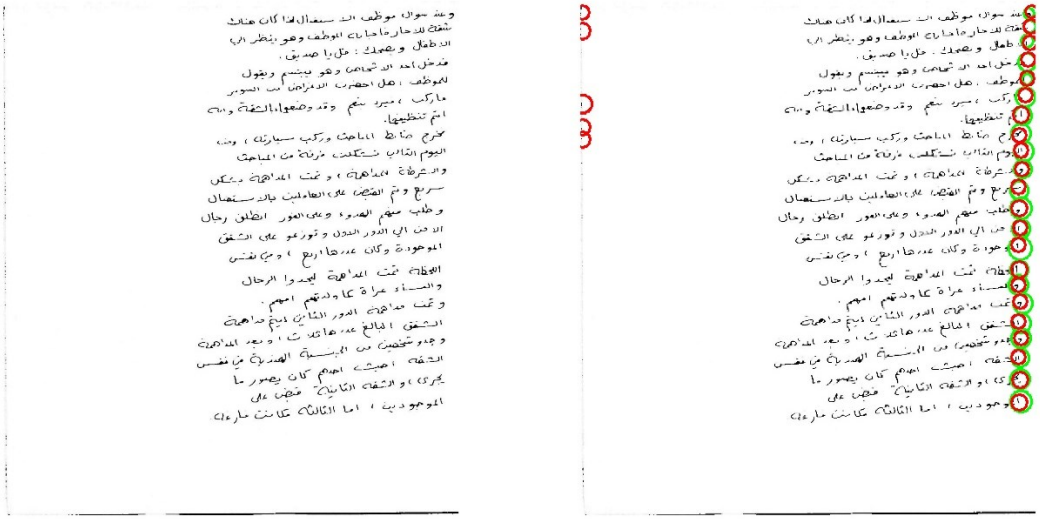
$$L(l, p; t) = \sum_{n=0}^{N} \sum_{m=0}^{M} X_{nm}(\alpha \| l_n - t_m \|_2^2 - \log(p_n)) - (1 - X_{nm}) \log(1 - p_n) \quad (11)$$

the target locations are(t_m), and the probability of occurred SOL is(p_n) , (l_n) is the predicted coordinates (x_n, y_n, s_n, θ_n). Equation 12 describes how we find(l_n) :

$$l_n = (-\sin(\theta_n)\, s_n + x_n, -\cos(\theta_n)\, s_n + y_n, \sin(\theta_n)\, s_n + x_n,\ \cos(\theta_n)\, s_n + y_n) \quad (12)$$

the binary alignment matrix between the (N) expected and the target positions (M) is the(X_{nm}). (α) Weighs is the relative importance of local loss and confidence loss. The SFR authors used $(\alpha = 0.01)$ and we used the same value of it (Wigington *et al*., 2018).

It is worthy to mention that the pre-training process stops in the SOL network after 10 epochs pass without any improvement in the loss (Start follow read github Page, 2020). Figures 28 show sample of the results that we get when we apply SOL network on the test data from the MADCAT datasets. Because we used a small dataset, we can visualize our results on the test data. The green circles in the Figures represent the start of the lines in the GT data, while the red circles represent the start of the lines in the prediction data.



(a) MADCAT document example

(b) GT labels for SOLs & predected SOLs by applying SOL model in pre-training step

Figure 28: Sample of applying SOL pertaining result on the MADCAT dataset

Step 6: Line Follower model (LF). The LF model works at every point predicted by SOL, with the following steps forward the text line, following the hook, and creating a normalized text-line image. To assess the results achieved by applying the LF

network, as mentioned before, we used the loss function, which is the same function that was used by the SFR authors to evaluate the LF network.

The loss function used to train the LF is a Mean Square Error (MSE). Equation 13 explains how we can measure the loss function applied to estimate the LF model:

$$\text{loss} = \sum_{i=0} \left\| p_{u,i} - t_{u,i} \right\|_2^2 + \left\| p_{l,i} - t_{l,i} \right\|_2^2 \qquad (13)$$

where $(t_{u,0})$ and $(t_{l,0})$ are the first target points. These points are resetting every fourth step to the corresponding target points. The goal of this method is to make the LF network to be able to fix without entering major errors during the training step if it deviates from the text line. To help LF fix previous incorrect predictions, the LF position is randomly disabled after resetting to the target position, by converting(Δx, Δy) to $[-2,2]$ pixels and ($\Delta\theta$) rotation to $[-0.1, 0.1]$ radians (Wigington *et al*., 2018).

The pre-training process stops in the LF network after 10 epochs pass without any improvement in the loss. The results of LF model are more than perfect for HTR model. Nevertheless, Figure 29 presented a Sample of applying LF pertaining result on the test data from the MADCAT dataset.

(a) MADCAT document example

(b) GT baseline xy values & predicted xy values using LF model in pre-training step

(c) GT baseline xy values in green points

(d) Predicted xy output using LF model in red points

Figure 29**: Sample of applying LF pertaining result on the validation data from the MADCAT dataset**

Step 7: Extract the resulting data from the SFR algorithm. After the LF produces the image normalization line, we will feed the resulting data to the HTR model, to produce the Arabic transcription for every line. It is worthwhile to mention that we used the HWR model from the SFR algorithm to extract the ground truth table (GT) notes and the Arabic transcription for every text-line image after extracting the text-line images from LF best model. We used the Arabic-reshape function from PyArabic library to return the Unicode according to the position of the letter

within the word and the way it was drawn, not the general Unicode for the letter. After that, use this specific Unicode to get the GT labels notes for every Arabic transcription.

PyArabic library is a special Arabic language library for Python implements basic functions to handle Arabic letters and text, like detecting Arabic letters, Arabic letters groups and characteristics, removing diacritics, *etc*. We got 172,236 text-line images with their GT label notes and the Arabic transcription files. Figure 30 describes the way that we extracted the resulted text-line images from SFR algorithm in easy way.

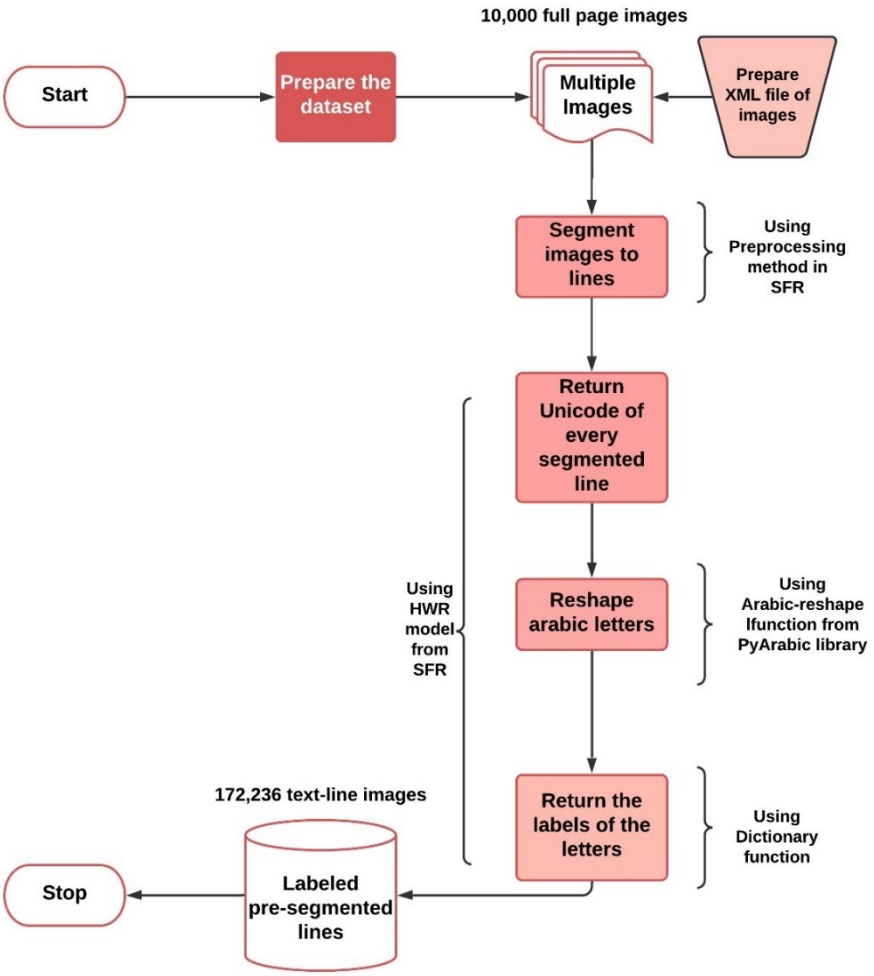


Figure 30: Extracted the resulted text-line images from SFR

5.4.2 Handwriting Recognition of Documents with Few Labeled Data

Now, to implement the HTR system on the resulted 172,236 text-lines handwritten Arabic documents from MADCAT dataset, we must apply some operations before we get the results of the HTR model. The following is an explanation of the steps that were taken during the training steps:

Step 1: Using Anaconda environment on PyCharm, installed all the dependencies we needed, this model use Tensorflow 2.4.1 as a base library for the training (HTR system github Page, 2021).

Step 2: Edit config.py page to take the lists of train, validation, test, and character list have all the Arabic characters which the HTR model needs to train on it.

Step 3: Train the model. After getting results, we tested the model.

Step 4: We generated, for each image, the posterior probabilities at each timestep. Files will be stored in probs by default.

Step 5: Using Tensorbard 2.4.1, Get the plots for train and test loss results. Figure 31 presented the train loss phase, this graphic extracted using Tensorboard Summary.

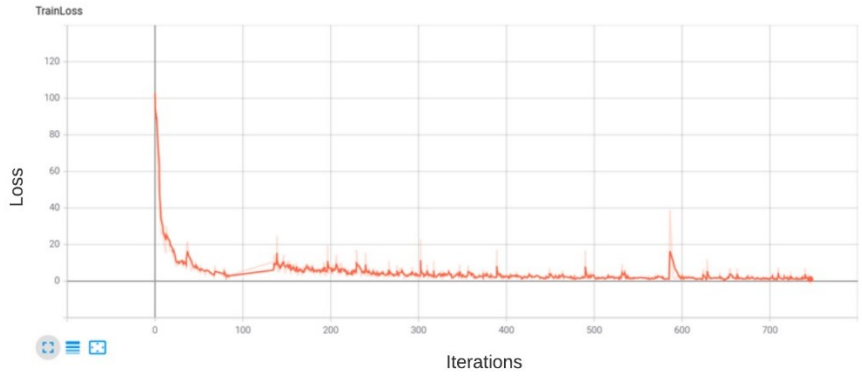


Figure 31**: HTR model train loss phase graph**

Figure 32 describes the full architecture of the HTR model while it works on the flipped raw images. It is worth noting that no preprocessing is needed in this architecture.

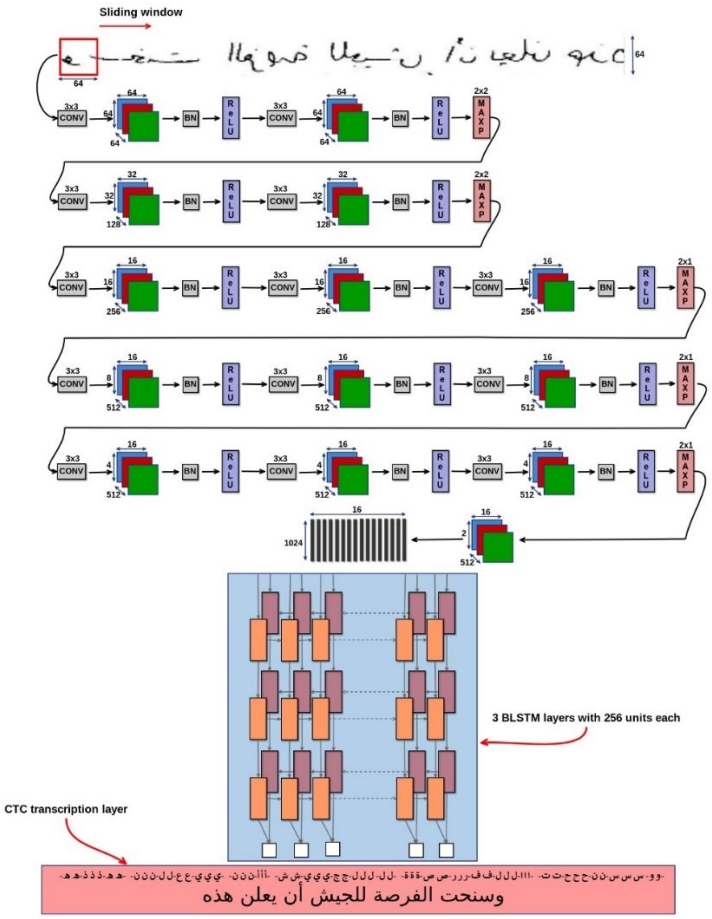


Figure 32: HTR model recognition system architecture on Arabic text-line image

In the next section, we will show the results for the HTR model and how we try to fine-tunning the model with multiple trials.

5.4.3 Fine-Tunning the Model

Fine-tuning is making small adjustments to a process to deliver the wanted output or performance. The fine-tuning process significantly reduces the time required for programming and treating a unique deep learning algorithm as it previously includes essential information of a pre-existing deep learning algorithm. Fine-tuning uses a model that has previously been trained for a particular task and then tweaking it to make it

perform more accurately. There are many ways to fine-tune the model, for example, modifying the hyperparameters of the algorithm (Lund *et al*., 2013). In this section, we will describe what we do with the dataset and the model to get the best results in the HWR for Arabic documents.

5.4.3.1 Dataset Split Way

After applying the first two models from the SFR algorithm (i.e., SOL and LF models), We got 172,236 text-line images with their GT labels and Arabic transcriptions. As we mentioned before in the dataset preparation chapter, the MADCAT dataset consists of three phases that contain a huge variety of fonts, and to ensure that this diversity is preserved in the validation and test sets, we have used a stratified method when we chose the validation and test sets randomly. The stratification method ensures that the test samples are representative of the target categories. We used 2,000 text-line images for validation with 2,000 for testing.

When we study the HTR research, we discovered that the authors decided to study the behavior of the model using different amounts of datasets. They give the model in the first step 20,000 line-text images for the train. Next, train the model on the rest of the dataset using the same validation set in the two experiments.

So, for our trials to study and train the HTR model on the MADCAT dataset and fine-tuning it, and furthermore, to save time from wasting and reach the best results in the shortest time, we decided to use 20,000 for training and fine-tuning trials on the model, and when we reach the best qualification, we will apply it to most of the dataset while maintaining the same validation and test sets.

5.4.3.2 Our Trials for Fine-Tuning

The following are our attempts to develop the model and make it obtain the highest results:

1) **Clean up the dataset.** The system is sensitive for the writing skew, so HTR model authors make a filtration step before they trained the model. They discarded the lines that have big issues like two text-line in the same image and cropped images and white lines. Therefore, if we discarded the skewed lines images from our dataset maybe it will give more good results. There are very small or overlapping fonts are observed due to scribe style or very poor fonts, the following images in Figure 33 are an example of the poor cutting or poor fonts.

However, what we noticed along the way is that while having a big dataset is important but having a clean one is even more. But also, get the efficiency of the model when it recognizes the lines of the data with all its problems without interfering or cleaning the data is a wonderful development for the model. So, we decided to study the behavior of the model with cleaned and not cleaned dataset. Figure 34 shows the numbers of the data and how we distribute it in these two experiments.

Figure 33: Arabic text-line image examples with poor cutting or poor fonts

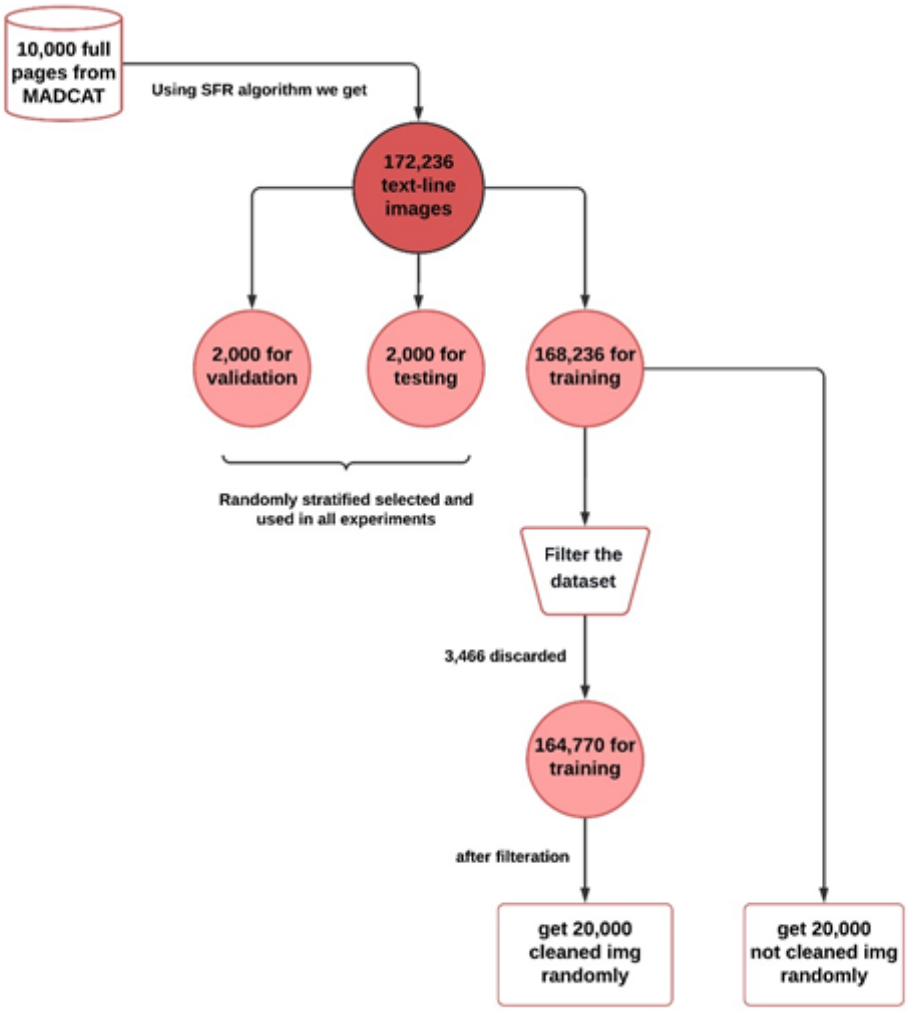


Figure 34: distributed way for resulted MADCAT dataset

So firstly, we cleaned the resulting images from the SFR algorithm, after taking the validation and test set in a stratified random way. We got a very small percentage of poorly cropped images, and when we reviewed it, we discovered the reason behind it is the SOL positions that fed to LF, is limited to a dimension that is too large and includes two lines in one image. The resulting images after filtration are stated later in table 2 compared to the filtration results from the HTR model on READ dataset.

Table 2: Cleaning results comparison of READ and MADCAT datasets

Dataset Name	Number of text-lines for training before filtration	Number of text-lines after filtration	Number of discarded text-lines
READ dataset	200,000	180,000	20,000
MADCAT dataset	168,236	164,770	3,466

After the two experiments was done, we discovered that there are not much different between using cleaned or not cleaned dataset in the training step, Not cleaned dataset gives us a good result without any filtration step. For this reason, we decided to use the dataset without any cleaning and to transform it as output from SFR algorithm to HTR model. Table 3 presents the results of the couple experiments, also Figure 39 represents the results of the experiments.

Table 3: HTR model results with cleaned and not cleaned dataset from MADCAT

Training Trials	Test Set		
	CER	WER	SER
Not cleaned 20k dataset	8.43%	34.19%	80.85%
Cleaned 20k dataset	8.53%	33.77%	79.65%

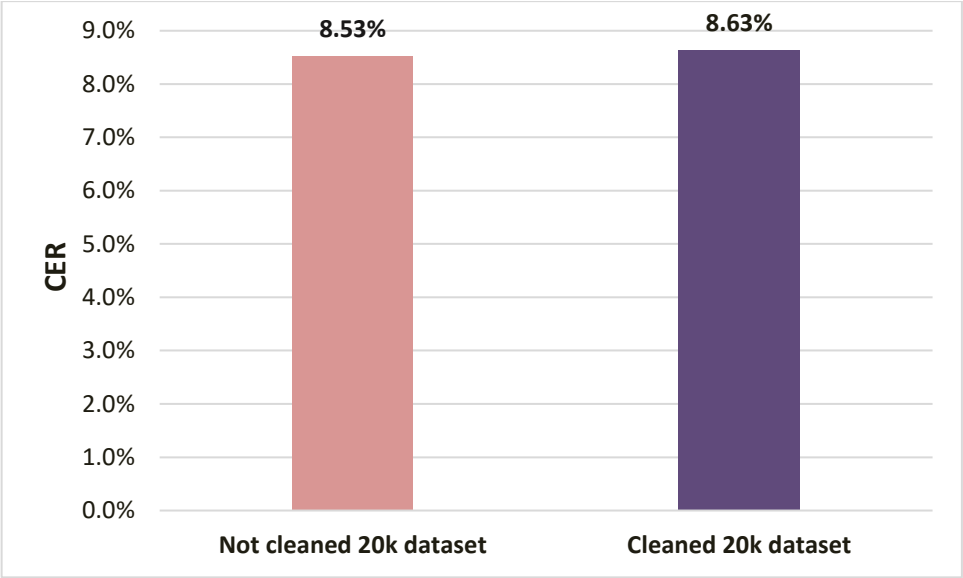


Figure 35: Results of cleaned and not cleaned dataset from MADCAT

2) **Tune the hyperparameters.** The hyperparameter is a parameter whose content is applied to control the learning manner. We have many Hyperparameters to tune it according to our work with HTR model. The following is the hyperparameters that we used to tune our model and enhance the results for the not cleaned dataset experiment:

- **Number of layers.** Authors of the HTR model inspired the deep CRNN-BLSTM architecture from the VGG16 structure utilized for image recognition. It's containing a stack of 13 convolutional (3 × 3 filters, 1 × 1 stride) layers followed by three Bidirectional LSTM layers with 256 units per layer. Nevertheless, Spatial pooling (max) was applied following some convolutional layers. And to include non-linearity, the leaky rectified linear unit (LeakyReLU) activation function was employed following each convolution.

After studying the VGG architecture we found a newer version called VGG19 that gives more accurate predictions among image recognition. We examine if we upgrade the

HTR architecture, which is like VGG16, to the VGG19 architecture, it will give us more good results for loss and CER measures in our dataset. Figure 36 describes the difference between the two architectures.

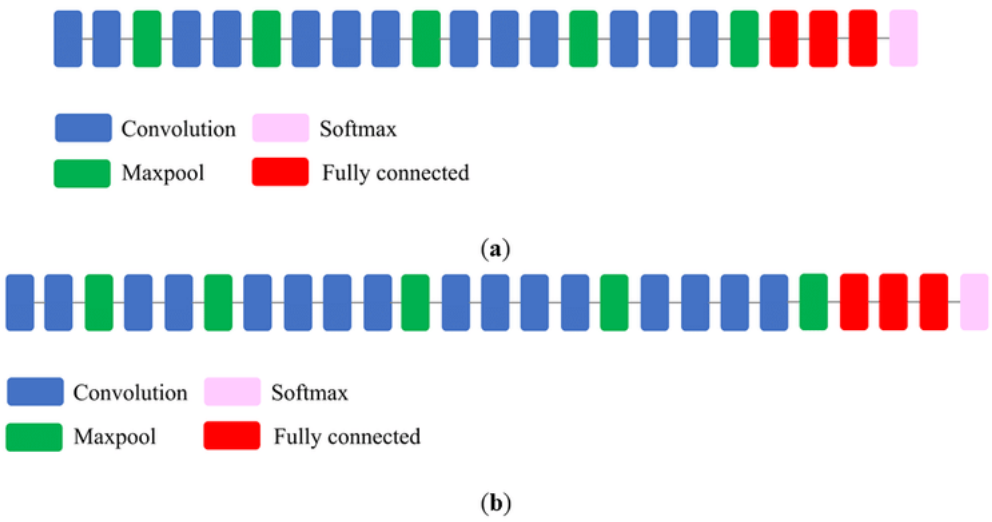


Figure 36: Architecture comparison (a) HTR-16 model and (b) HTR-19 model

The following two experiments were applied to not-cleaned 20,000 text-line images, and we noticed that the architectural development of the model led to a decline in the results. Sometimes the issue we need to solve does not require more complexity in the architecture, and the rise in the number of layers leads to poor analysis, even if it is manageable. Table 4 presents the results of the two experiments.

Table 4: HTR model results in the number of layers trial

Model architecture	Test Set		
	CER	WER	SER
HTR-16	8.43%	34.19%	80.85%
HTR-19	8.89%	34.79%	80.95%

- **Learning rate.** Learning rate (LR) is a hyper-parameter that manages the weights of the neural network concerning the loss gradient. It determines how swiftly the neural network updates the notions it has learned. A good learning rate is low enough that

the network meets something valuable, but important enough that it can be trained in a moderate amount of time (Anusha and Avadhani, 2019).

After searching in several references, we discovered that the most suitable learning rate for architecture like VGG is 0.0001(Anusha and Avadhani, 2019). So, we decided to try it in addition to the learning rate adopted in the HTR model (0.0005) and try it in the two architectures (HTR-16 and HTR-19) aspiring to enhance the performance of the second architecture. Table 5 shows the results assuming to the architecture and the learning rate and Figure 37 shows a chart for the results.

Table 5: HTR model results in the learning rate trials

Model Architecture	Learning Rate	Test Set		
		CER	WER	SER
HTR-16	**0.0005**	8.43%	34.19%	80.85%
HTR-19	**0.0005**	8.89%	34.79%	80.95%
HTR-16	**0.0001**	12.00%	45.34%	88.55%
HTR-19	**0.0001**	11.78%	44.41%	88.50%

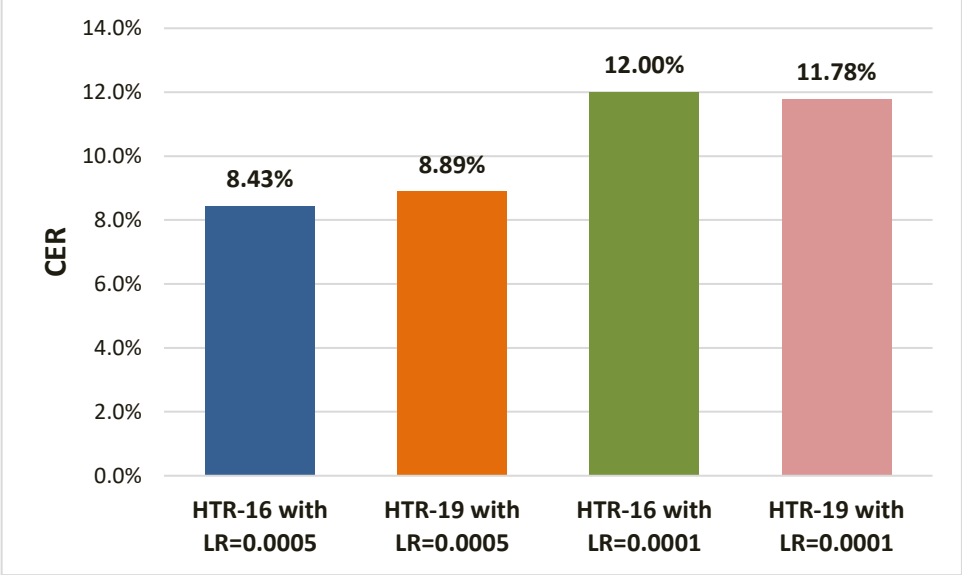


Figure 37: HTR model results in the LR trials

From the results presented in the previous table, we conclude that the best learning rate that gives the lowest results is 0.0005 with HTR-16 architecture.

- **The size of CNN layers.** Convolutional layers are the layers where filters are implemented to the primary image, or to other feature maps in the deep CNN. This is where most of the user-specified parameters are in the network. The most important parameter is the size of the layers in the model architecture (Wang *et al*., 2021). The size of CNN layers in the HTR model is typical of VGG16, it has (64x2, 128x2, 256x3, 512x6). In the next trial, we tried to upgrade the size of the last three layers in the architecture by making it (64x2, 128x2, 256x3, 512x3, 1024x3). Table 6 and Figure 38 present the results of the new size of CNN layers. We found good progress when developing the size of CNN layers in our model.

Table 6: HTR model results in the size of CNN layers trial

Size of CNN layers	Test Set		
	CER	WER	SER
64x2, 128x2, 256x3, 512x6	8.43%	34.19%	80.85%
64x2, 128x2, 256x3, 512x3, 1024x3	8.14%	32.87%	79.50%

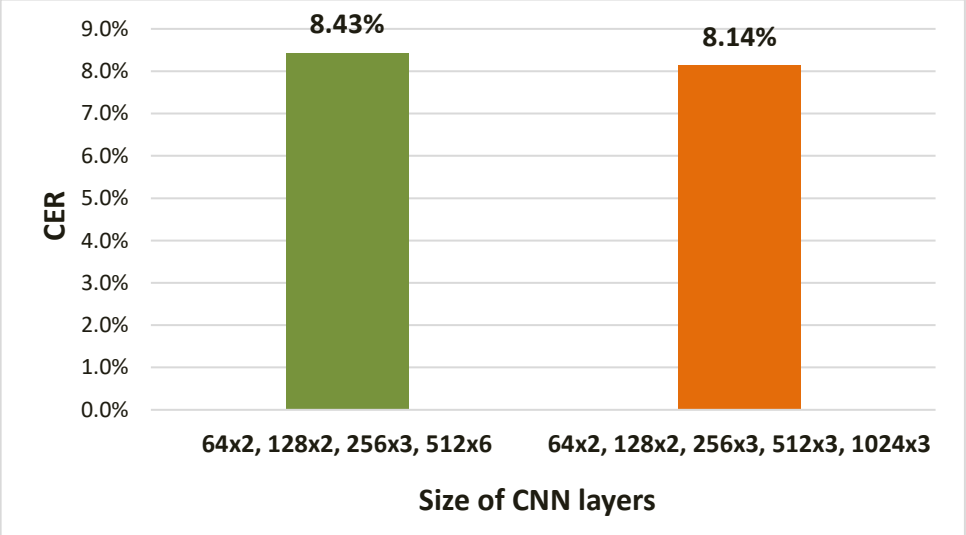


Figure 38: Results in the size of CNN layers trials

- **Number of Units per BLSTM**. Long short-term memory (LSTM) networks are a kind of recurrent neural network able of learning order dependency in sequence prediction queries. This is a response required in complicated problem areas like machine translation, speech recognition, and higher. It processes data crossing on information as it generates forward. The variations are the operations inside the LSTM's cells. These procedures are used to provide the LSTM to hold or ignore the information. Bidirectional long-short term memory (Bi-LSTM) is the process of making any neural network have the sequence information in both directions backward (future to past) or forward (past to future). In bidirectional, our information moves in two directions, making a Bi-LSTM different from the regular LSTM (Wang *et al*., 2021).

We decided to join more units per BLSTM layer in our network to make improvements in the model results by changing the number of units equals (2^8 = 256) units to (2^9 = 512). Table 7 and Figure 39 describes the results of this experiment. We didn't find any progress when developing the number of BLSTM units in our model.

Table 7: HTR model results in Number of Units per BLSTM layers trials

Number of units per 3 BLSTM	Test Set		
	CER	WER	SER
256	8.43%	34.19%	80.85%
512	8.65%	34.26%	80.75%

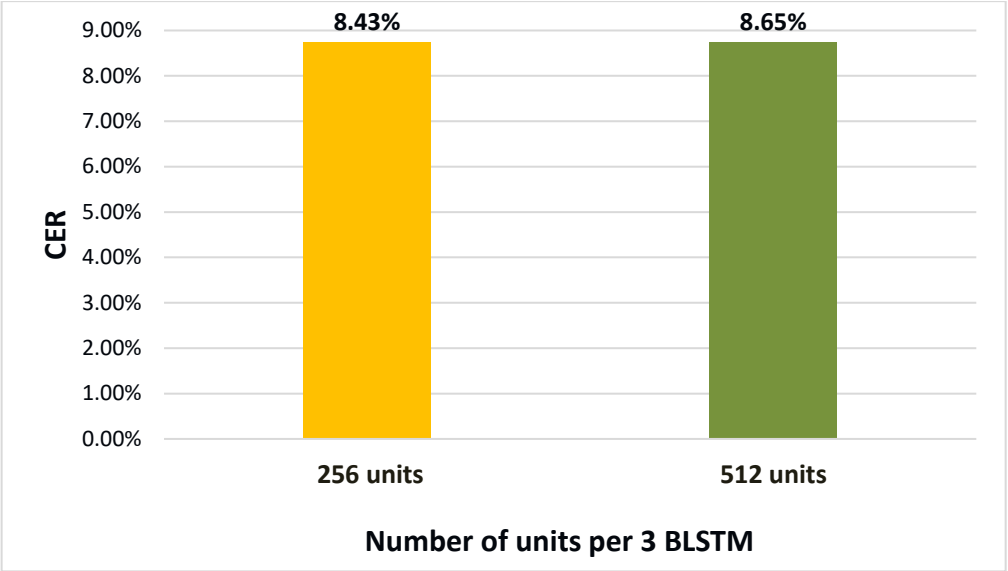


Figure 39: Results in number of units per BLSTM layers trial

3) **Multi-stage Learning.** Multi-stage learning means reusing the best model of a neural network trained on the same dataset to continue the learning with a huge new dataset, instead of training the neural network from scratch. So, if we used the training step best model as a pre-trained model for the next train stage it will have fewer weights to update and can reach a high level of accuracy with our large dataset in much less time. We will use the best architecture with the best hyperparameters that give us the best results, in our case, which give us less CER. Table 8 describes the final model architecture that we used to train the full dataset in the end and table 9 presents the results of training our proposed model on different data size. Also, Figure 40 presents the line-level dataset results on various data size.

Table 8: Full training model information

Criteria	Value
Model Architecture	VGG16
Learning Rate	0.0005
CNN layers	13
Size of each layer	64x2, 128x2, 256x3, 512x3, 1024x3
Number of BLSTM layers	3
Number of LSTM units per forward/backward layer	256
Optimizer	Adam optimizer
Activation functions	LeakyReLU
Training set	168,236 not cleaned
Validation set	2,000 not cleaned
Test set	2,000 not cleaned
Randomize the order of training batches each epoch	True
Stop the training after n epochs with no improvement on validation	5

Table 9: Line-level dataset results for our contribution on various data size

Dataset size	Test set		
	CER	**WER**	**SER**
20k text-line images	8.14%	32.87%	79.50%
168k text-line images	3.96 %	17.27 %	55.55 %

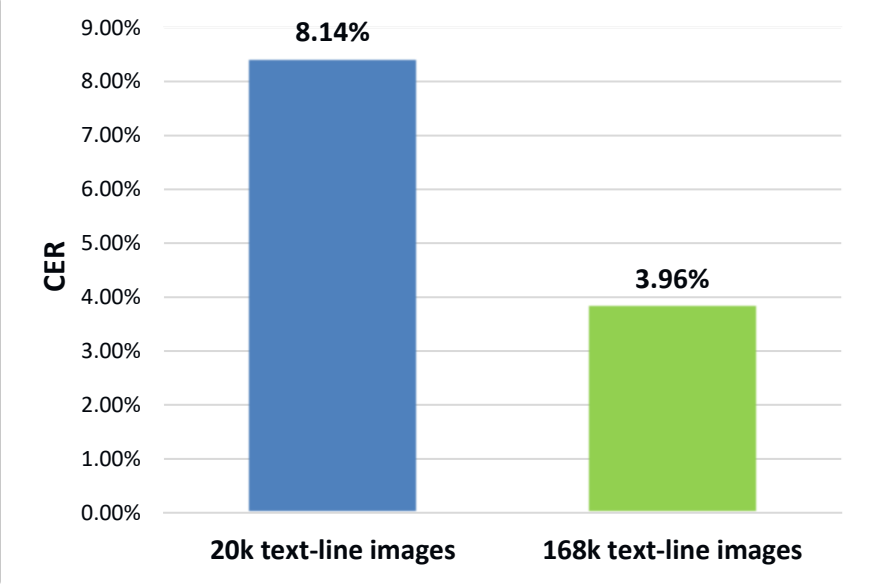


Figure 40: Line-level dataset results on various data size

5.5 Experiments Results and Discussion

In this section, we will present a summary of the completion of the three networks presented in part 5.3 and the result for training the model on the full MADCAT dataset in part 5.4. In this thesis, we implemented an integrated algorithm on the MADCAT dataset for the Arabic language.

To investigate the extent of the success of merging the SFR algorithm with the HTR algorithm in the HWR of the full-page Arabic documents, we estimated the value of the loss on the MADCAT dataset for the SOL model and LF model and CER on HTR model.

As we stated earlier, we used Swalhah and Abandah research as a state-of-the-art solution for our work. We make some major changes and adopt the HTR model with the first algorithm to get more accurate results and use 10,000 full images from the MADCAT dataset. For our contributions, we used the first two models from the SFR algorithm (SOL and LF) with 100 images fully annotated for training and the rest of the data from 10,000 full images for testing, after getting the results from the first algorithm we fed it to the HTR model for HWR step with 172,236 test-line images. Using 168,236 for training and 2,000 for validation and 2,000 for testing. Our results for every stage are mentioned in table 9.

We got results with SOL model equals 34.62. Figure 41 explains how to train loss and test loss shrinkage when SOL network is implemented on the MADCAT dataset. Additionally, for the LF network, the loss equals 73.92. Figure 42 shows how the training loss and test loss decrease when the LF network is applied to the MADCAT dataset. Nevertheless, we got with HTR model CER equals 3.85 % and loss equals 4.44, and that's an incredible result in HWR for Arabic documents have a big variety of writers fonts. Figures 43 presents the train and test CER when the HTR model is applied to our dataset.

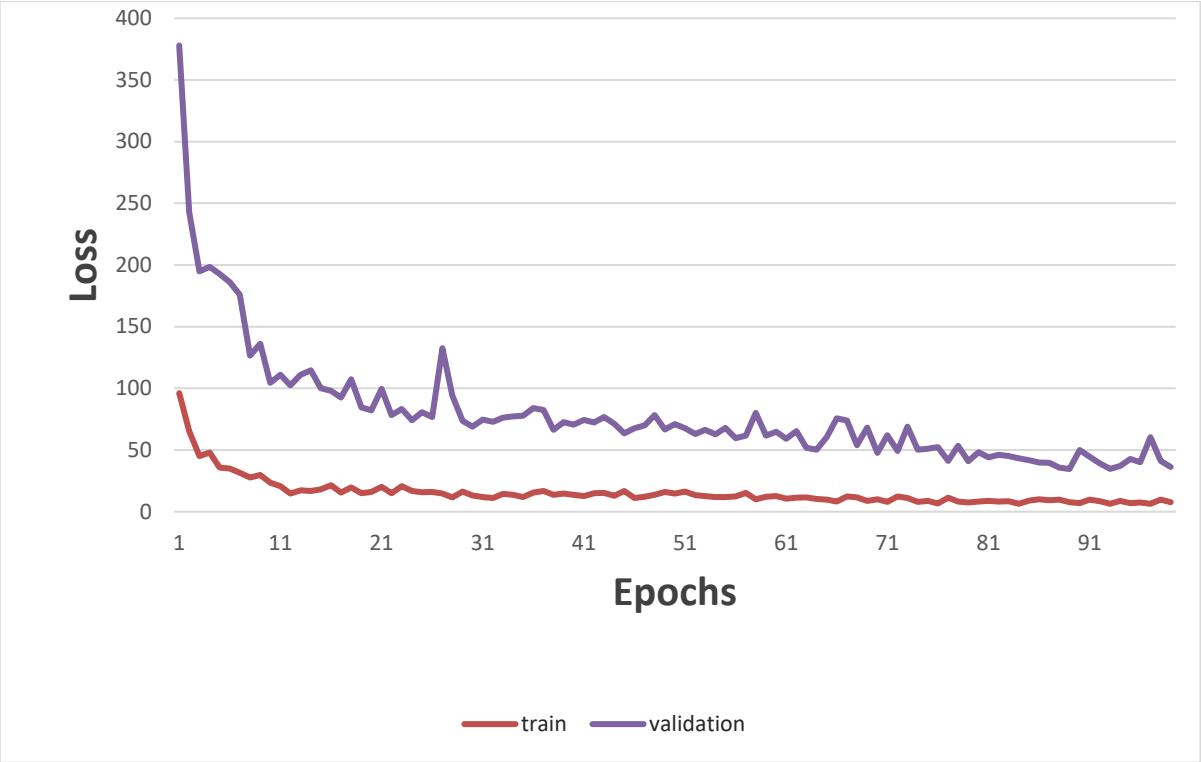


Figure 41: SOL model results plot on train and validation sets

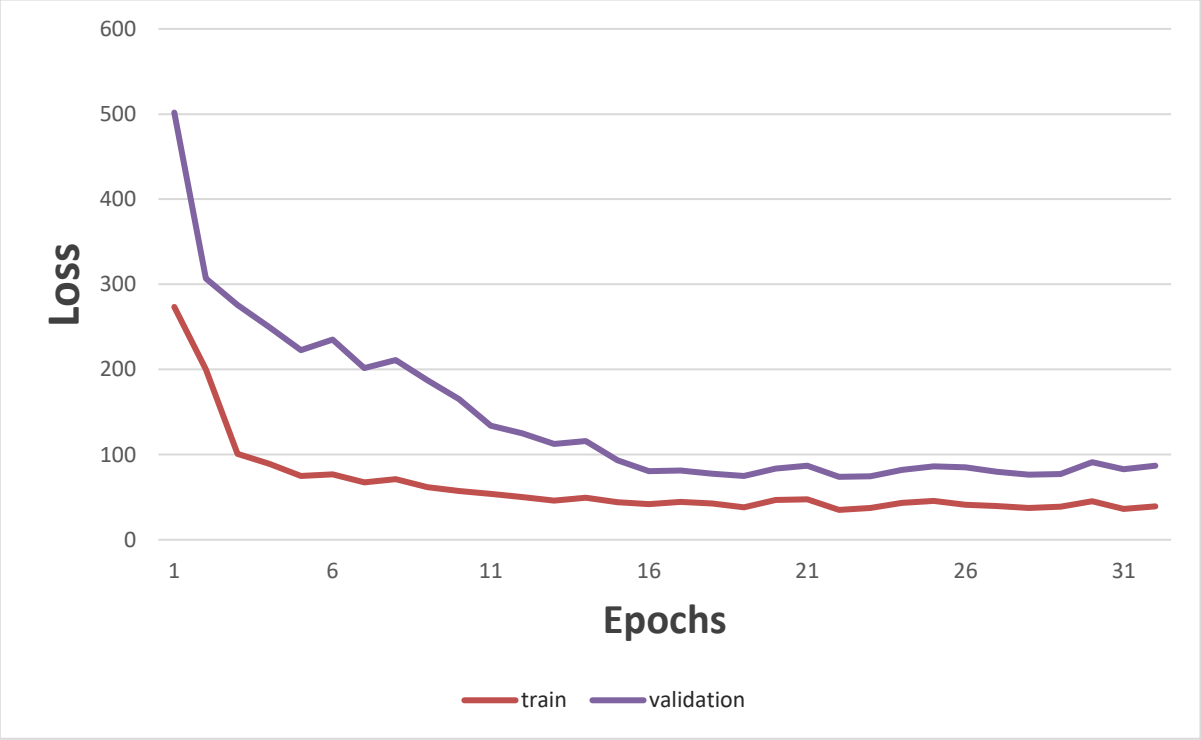


Figure 42: LF model results plot on train and validation sets

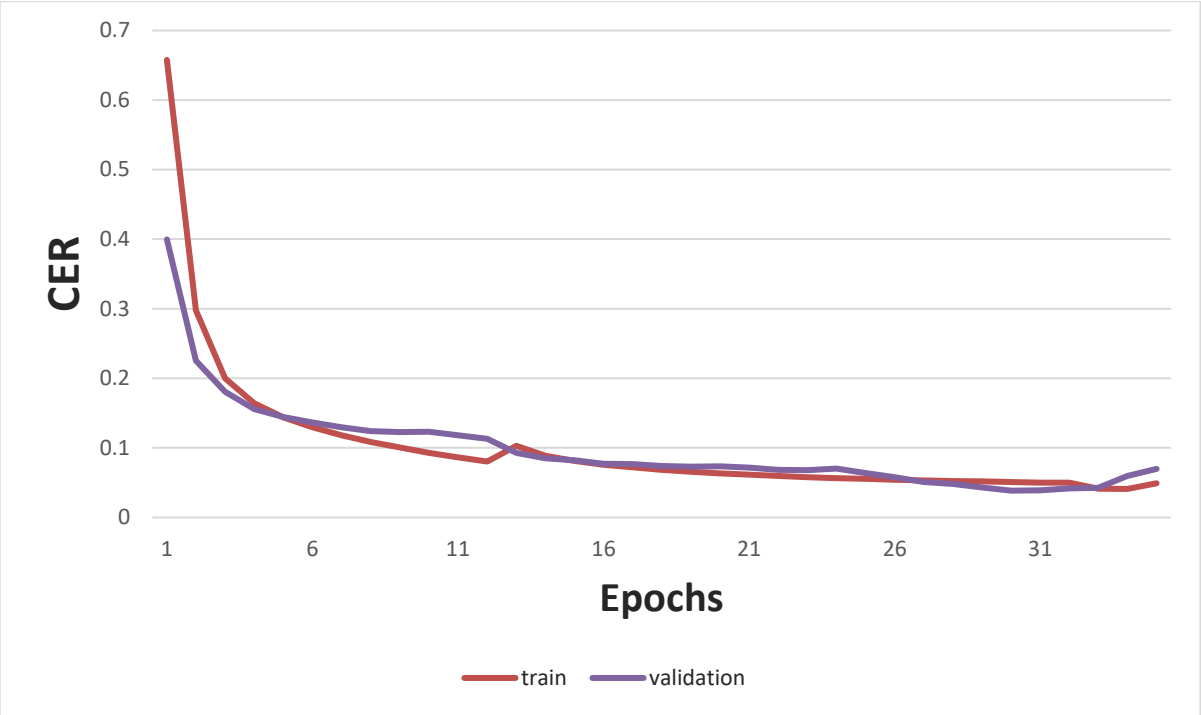


Figure 43: HTR model results plot on train and validation sets

5.5.1 Comparison With the State-of-art Algorithms

HWR designs are often bounded by the accuracy of the preceding steps of text detection and segmentation. Motivated by this, we perform an end-to-end deep learning model that collectively learns text detection, segmentation, and recognition using the MADCAT dataset. Through our attempts to make that integrated system to detect HWR for Arabic documents, both algorithms were employed: (1) SFR algorithm and (2) HTR algorithm.

While the state-of-art research focus discussion on one of the most challenging HWR fields, i.e., READ dataset of Latin documents which contains 10,000 images, these proposed methods are both suitable to other HWR domains. Based on this, these algorithms were used as a nucleus for our research to identify Arabic handwritten documents. Table 10 and Figure 44 present a comparison between our solution result and the other algorithms results.

Table 10: Line-level dataset results for our contribution compared to ICDAR 2017 HWR competition results algorithms

Method	Dataset	Results %	
		CER	WER
SFR model	READ	2.10	9.30
HTR model	READ	7.01	19.06
SFR with HTR models (Ours)	MADCAT	3.96	17.27

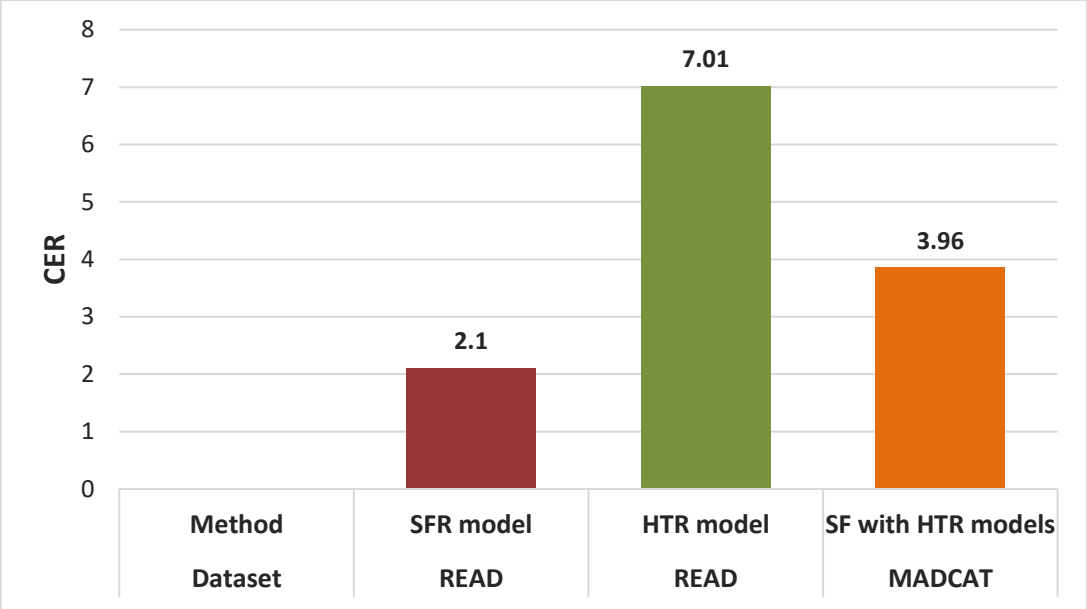


Figure 44**: Results compared to the state-of-art algorithms**

5.5.2 Comparison With the OpenHaRT'13 Evaluation Algorithms

To determine the efficiency of our algorithm on a huge dataset with a diversity in fonts such as our dataset, results should be compared with several algorithms that are used MADCAT dataset to predict the HWR on Arabic documents.

The NIST 2013 Open Handwriting Recognition and Translation (OpenHaRT'13) evaluation series was established from the DARPA MADCAT program to build knowledge in the Arabic language by making the participation available to all researchers so that the most complex tasks can be tackled to achieve the ability to understand handwritten Arabic documents (Tong *et al*., 2014).

OpenHaRT'13 focuses on increasing the essence of document text recognition and translation technologies for document images including mostly handwritten Arabic scripts. Three tasks existed to define the performance of individual algorithmic techniques, which are: (1) The Document Image Recognition (DIR), (2) The Document Image Translation (DIT), and (3) The Document Text Translation (DTT) (Tong *et al*., 2014).

The Document Image Recognition (DIR) task estimated the system's ability to recognize the text in the document images. OpenHaRT'13 was offered a collection of Arabic document images and text line segmentation and was asked to output the Arabic transcription located in every image. In the recognition task (DIR), WER (Word Error Rate) was used to estimate the system performance. WER is an edit distance metric that estimates the number of revisions needed so that the system transcription will check exactly the reference transcription (Tong *et al*., 2014).

The image scanned step in the algorithms is processed by reading its pixel data in four column-first "directions" that arise from combining top-down and bottom-up column traversals with left-to-right and right-to-left row traversals. Then for each line, a certain polygon around the whole line was constructed connecting the given word polygons, and this gives the initial text-line image. While the decision for image pre-processing and normalization of the lines is decided by the teams, they can correct the slants and normalize the images to a specific height to make them perfect for their algorithms (Leifert *et al*., 2013).

Compared to our work with the rest of the OpenHaRT'13 teams' work, we find that we achieved better results in terms of using machine learning techniques in segmenting the images and converting them into printed texts. By using the word error rate measurement, we achieved a better result in a subset of the MADCAT dataset, than the rest of the algorithms in the competition. Figure 45 presents a comparison between our results and the OpenHaRT'13 teams' results.

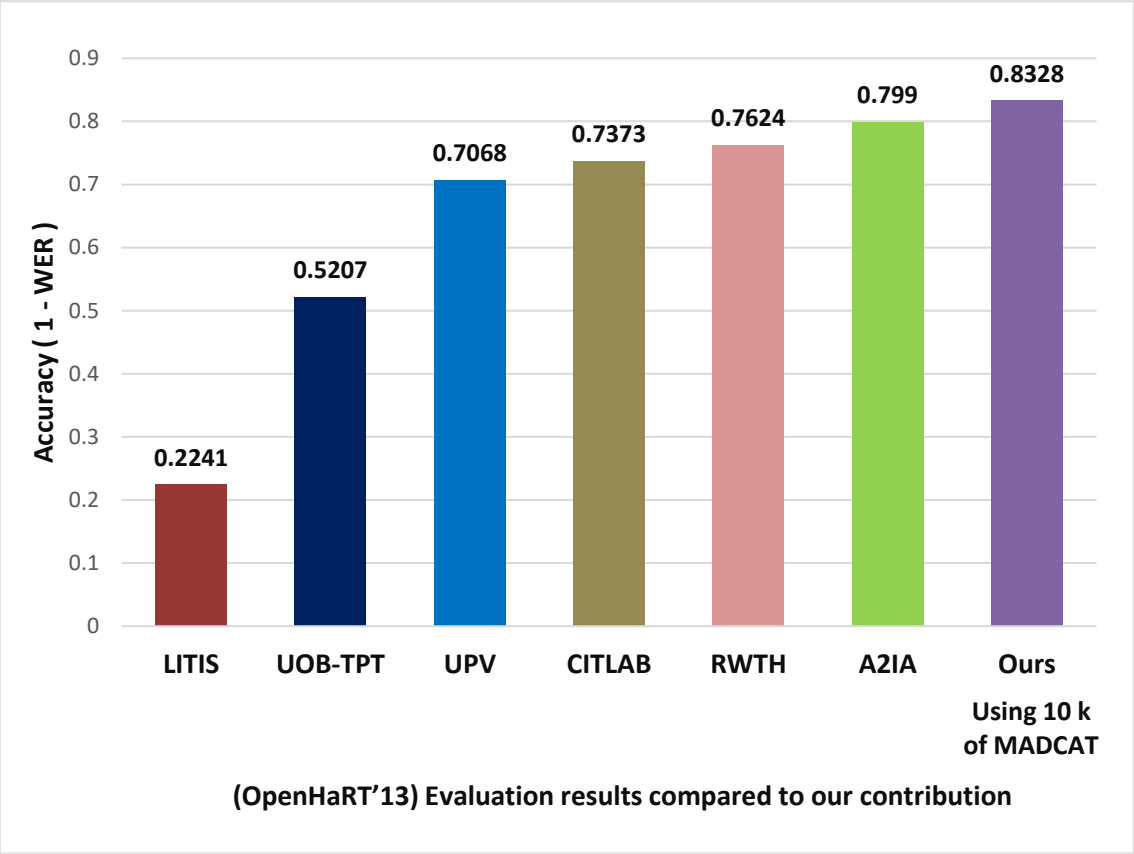


Figure 45: Our results Compared to the OpenHaRT'13 Evaluation Algorithms using Accuracy (1-WER)

5.5.3 Results Discussion

We remarked from the results shown before, that our model gives great results with the MADCAT dataset after trying to join the two algorithms (i.e., SFR and HTR). In this work, besides the major target which is to develop an end-to-end machine learning

algorithm for HWR Arabic documents, is find the latest optimal solution with conventional resources that can any machine learning software developer have it.

The idea for joining the two algorithms together is due to the appearance of the hardware gap between the SFR algorithm proposed from the first contribution of Swalhah and Abandah, and our attempts to complete and develop this work. During our work, we realized that we need to do the full training phase a 4 GBUs, each of which has a memory equals 12 GB, which means we need for the algorithm to run in parallel a GPU memory equals 48 GB, and what we had was just a 2 GB 1050 Gtx, we have the types of equipment that any developer has it. Besides that, many releases for SFR dependencies libraries have been removed their support from the official GBUs sites, so we can't create the same environments of the first contribution.

SFR algorithm achieves two main goals, which are segmenting the full document and predicting a good result due to the full use of the dataset. But the hardware problem has become an obstacle to its use. Also, SFR algorithm has become an old technology that relies on very old versions of programming that no one uses anymore, and support for it has been cancelled. Furthermore, as mentioned by the authors of the HTR model, the main problem in their research is that a huge number of lines have been discarded due to their poor segmentation method. Inspired by these points, we decided to combine them and adapt to the existing and try to complete the work and find a powerful end-to-end model that together learns to discover text, segmentation, and recognition frequently using images.

CHAPTER VI

Conclusions and Future Work

In this thesis, we performed a study on the feasibility of using and applying machine learning methods used in recognizing handwritten documents in the Arabic language. We chose the SFR system and HTR system and merged them together to recognize handwritten Arabic documents because they recorded good performance in the difficult handwritten dataset from the German history dataset after the nineteenth century.

Our proposed approach is classified as end-to-end for recognizing handwritten documents from the top to the bottom of the page. Our algorithm is used for offline handwriting recognition (HWR), it's a deep learning model that learns to discover text, segmentation, and recognition frequently using images. Our algorithm consists of 3 sub-models: SOL is used to find the starting position of a line. The LF network begins at every position predicted by SOL, then the LF network predicts step-by-step along the line, and the LF network also follows the curved lines and corrects them to appropriate regular text images. And finally, the HTR network predicts the transcription of the text.

We got a loss result with the SOL model equals 34.62, Additionally, For the LF network, the loss equals 73.92. Nevertheless, we got with HTR model CER equals 3.96 % and loss equal 4.44, and that’s an incredible result in HWR for Arabic documents have a big variety of writers fonts.

After completing the full training phase of the model and completing its modification, some ways can be developed and delivered CER results closer to zero within some development points:

1) **Add more data.** through our work in the dataset filtration step, we used some tags

were used to select the dataset and start our experiments with it. We recommend adding extra tags to add new writing styles and add new features to Arabic fonts, which helps the model learn more styles.

2) **Clean start of line XML annotations.** While we were working on the segment the page for lines with the LF model, we noticed a small percentage of the lines that have the problem of overlapping or containing two lines in the same text-line image. After following this problem, we found that the reason behind this is wrong information from the XML design that is entered into the SOL model and is being trained on it, and this information is incorrectly labeled from the main source of the MADCAT dataset. We recommend refining or correcting it with Median equations for images.

3) **Optimizing training time performance of the HTR model.** One of the most important factors while working in deep learning is the speed of the algorithm. When working on a large amount of data, such as images, learning is slow compared to other types of data such as texts or numbers. Therefore, we recommend accelerating the algorithm through various acceleration methods such as the Dataset API from Tensorflow.

References

Abandah, G. A., and Al-Hourani, A. S. (2018). Challenges and Preprocessing Recommendations for MADCAT Dataset of Handwritten Arabic Documents. **International Congress on Image and Signal Processing, BioMedical Engineering, and Informatics (CISP-BMEI)**, 1-9. IEEE, 2018.

Abandah, G. A., and Jamour, F. T. (2010). Recognizing handwritten Arabic script through efficient skeleton-based grapheme segmentation algorithm. In 2010 10th **International Conference on Intelligent Systems Design and Applications**, 977-982. IEEE, 2010.

Abandah, G. A., and Jamour, F. T. (2014, a). A Word Matching Algorithm in Handwritten Arabic Recognition Using Multiple-Sequence Weighted Edit Distances. **International Journal of Computer Science Issues (IJCSI**), 11(3), p.18.

Abandah, G. A., and Malas, T. M. (2010). Feature selection for recognizing handwritten Arabic letters. **Dirasat Engineering Sciences Journal**, *37*(2).

Abandah, G. A., Graves, A., Al-Shagoor, B., Arabiyat, A., Jamour, F., and Al-Taee, M. (2015). Automatic diacritization of Arabic text using recurrent neural networks**. International Journal on Document Analysis and Recognition (IJDAR)**, *18*(2), 183-197.

Abandah, G. A., Jamour, F. T., and Qaralleh, E. A. (2014, b). Recognizing handwritten Arabic words using grapheme segmentation and recurrent neural networks. **International Journal on Document Analysis and Recognition (IJDAR)**, 17(3), 275-291.

Abandah, G. A., Younis, K. S., & Khedher, M. Z. (2008, February). Handwritten Arabic character recognition using multiple classifiers based on letter form. **In Proceedings of the 5th International Conference on Signal Processing, Pattern Recognition, and Applications (SPPRA)** (pp. 128-133).

Abandah, G., & Anssari, N. (2009). Novel moment features extraction for recognizing handwritten Arabic letters. **Journal of Computer Science**, 5(3), 226.

Abandah, Gheith A. (2015) "Feature Extraction–Arabic OCR Case Study."

Ali, A. A. A., & Suresha, M. (2019). A novel features and classifiers fusion technique for recognition of Arabic handwritten character script. **SN Applied Sciences**, 1(10), 1286.

Almuallim, H., and Yamaguchi, S. (1987). A method of recognition of Arabic cursive handwriting. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, (5), 715-722.

Alom, M. Z., Hasan, M., Yakopcic, C., Taha, T. M., and Asari, V. K. (2018). Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation. **ArXiv preprint arXiv:1802.06955 :1802. 06955..**

Al-Yousefi, H., and Upda, S. S. (1992). Recognition of Arabic characters. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, (8), 853-857.

Anusha, C., Avadhani, P. S., (2019). Optimal Accuracy Zone Identification in Object Detection Technique - A Learning Rate Methodology. **International Journal of Engineering and Advanced Technology (IJEAT)**, ISSN: 2249 – 8958.

Arvanitopoulos, N. and Süsstrunk, S., 2014, September. Seam carving for text line extraction on color and grayscale historical manuscripts. **In 2014 14th International Conference on Frontiers in Handwriting Recognition**, 726-731, IEEE.

Balaha, H. M., Ali, H. A., & Badawy, M. (2020). Automatic recognition of handwritten Arabic characters: a comprehensive review. Neural Computing and Applications, 1-24.

Belabiod, A., & Belaïd, A. (2018). Line and Word Segmentation of Arabic handwritten documents using Neural Networks. [Research Report] LORIA - Université de Lorraine. 2018. <hal-01910559>

Bjorck, N., Gomes, C.P., Selman, B. and Weinberger, K.Q., (2018). Understanding batch normalization. **In Advances in Neural Information Processing Systems**, 7694-7705.

Bluche, T. (2016). Joint line segmentation and transcription for end-to-end handwritten paragraph recognition. **In Advances in Neural Information Processing Systems**, 838-846.

Bluche, T., Louradour, J., and Messina, R. (2017). Scan, attend and read: End-to-end handwritten paragraph recognition with mdlstm attention. **In Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference**, Vol. 1, 1050-1055, IEEE.

Boufenar, C., Batouche, M., & Schoenauer, M. (2018). An artificial immune system for offline isolated handwritten Arabic character recognition. Evolving Systems, 9(1), 25-41.

Castro, D., Bezerra, B. L., and Valença, M. (2018). Boosting the deep multidimensional long-short-term memory network for handwritten recognition systems. **In International Conference on Frontiers in Handwriting Recognition (ICFHR)**, Vol. 1, 1050-1055, 2018 16th, IEEE.

Chammas, E., Mokbel, C., and Likforman-Sulem, L. (2018). Handwriting Recognition of Historical Documents with few labeled data. **In IAPR International Workshop on Document Analysis Systems (DAS)**, 43-48, 2018 13th, IEEE.

Chowdhury, A., and Vig, L. (2018). An efficient end-to-end neural model for handwritten text recognition. **ArXiv preprint arXiv:1807.07965**.

Cohen, Rafi, Itshak Dinstein, Jihad El-Sana, and Klara Kedem. (2014). Using scale-space anisotropic smoothing for text line extraction in historical documents. **In International Conference Image Analysis and Recognition**, 349-358, Springer, Cham.

Darryl K. Taft , 2010, Eweek, JetBrains Strikes Python Developers with PyCharm 1.0 IDE. Available from **https://www.eweek.com/development/jetbrains-strikes-python-developers-with-pycharm-1.0-ide.**

Das, A., Li, J., Zhao, R., and Gong, Y. (2018, April). Advancing connectionist temporal classification with attention modeling. **In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**, 4769-4773, IEEE.

Ding, X., & Liu, H. (2006, September). Segmentation-driven offline handwritten Chinese and Arabic script recognition. **In Summit on Arabic and Chinese Handwriting Recognition** (pp. 196-217). Springer, Berlin, Heidelberg.

Došilović, F. K., Brčić, M., & Hlupić, N. (2018, May). Explainable artificial intelligence: A survey. **In 2018 41st International convention on information and communication technology, electronics, and microelectronics (MIPRO)** (pp. 0210-0215). IEEE.

Dutta, K., Krishnan, P., Mathew, M., and Jawahar, C. V. (2018). Improving cnn-rnn hybrid networks for handwriting recognition. **In International Conference on Frontiers in Handwriting Recognition (ICFHR)**, 80-85, 2018 16th, IEEE.

El-Sawy, A., Hazem, E. B., and Loey, M. (2016). CNN for handwritten Arabic digits recognition based on LeNet-5. **In International Conference on Advanced Intelligent Systems and Informatics**, 566-575, Springer, Cham.

Geeks3D, First Steps with PIL: Python Imaging Library Page. Last accessed November 21th Available from **https://www.geeks3d.com/20100930/tutorial-first-steps-with-pil-python-imaging-library/#p01**.

GeeksforGeeks, Python | Image blurring using OpenCV Page. Last accessed November 21th. Available from **https://www.geeksforgeeks.org/python-image-blurring-using-opencv**.

Géron, A. (2017). **Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems**. O'Reilly Media, Inc.

Graves, A., and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. **In Advances in neural information processing systems**, 545-552.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. **In Proceedings of the 23rd international conference on Machine learning**, 369-376.

ICDAR2017 Competition on Handwritten Text Recognition on the READ Dataset (ICDAR2017 HTR) Page, 2017. Last accessed November 21st. Available from **https://scriptnet.iit.demokritos.gr/competitions/8/**.

IFN/ENIT-database Page. Last accessed November 21st. Available from **http://www.ifnenit.com/**.

Jamal, A. T., Nobile, N., & Suen, C. Y. (2014, October). End-Shape recognition for Arabic handwritten text segmentation. **In IAPR Workshop on Artificial Neural Networks in Pattern Recognition** (pp. 228-239). Springer, Cham.

Jamro, W. A., Shaikh, H., and Mahar, J. A. (2016). Comprehensive Analysis of Neural Network Techniques in Computational Linguistic Applications. **Asian Journal of Engineering, Sciences and Technology**, 15-22.

Jason Brownlee, 2017, A Gentle Introduction to Calculating the BLEU Score for Text in Python, Available from **https://machinelearningmastery.com/calculate-bleu-score-for-text-python/**

Kaggle, vgg11 Page. Last accessed November 21st. Available from **https://www.kaggle.com/pytorch/vgg11**.

Karmarkar, (2018), Region Proposal Network (RPN) — Backbone of Faster R-CNN. Last accessed November 21st. Available from **https://medium.com/egen/region-proposal-network-rpn-backbone-of-faster-r-cnn-4a744a38d7f9**.

Khedher, M. Z., Abandah, G. A., & Al-Khawaldeh, A. M. (2005, February). Optimizing Feature Selection for Recognizing Handwritten Arabic Characters. **In WEC** (2) (pp. 81-84).

Lawgali, A. (2015). A survey on Arabic character recognition.

Leifert, G., Labahn, R., & Strauß, T. (2013, August). CITlab ARGUS for arabic handwriting: Description of CITlab's system for the OpenHaRT 2013 Document Image Recognition task. **In Proceedings of the NIST 2013 OpenHaRT Workshop [Online]**.

Lewis, M. P. (2009). Ethnologue: Languages of the world. SIL international. (http://www.ethnologue.com/ethno_docs/distribution.asp?by=size).

Lorigo, L. M., and Govindaraju, V. (2006). Offline Arabic handwriting recognition: a survey. **IEEE transactions on pattern analysis and machine intelligence**, 28(5), 712-724.

Lund, W. B., Kennard, D. J., & Ringger, E. K. (2013, February). Combining multiple thresholding binarization values to improve OCR output. **In Document Recognition and Retrieval XX** (Vol. 8658, p. 86580R). International Society for Optics and Photonics.

Magnimind, 2019, What are the differences between deep learning and usual machine learning?. Available from **https://becominghuman.ai/what-are-the-differences-between-deep-learning-and-usual-machine-learning-a822b11ada01**.

Martin Heller, 2018, What is CUDA? Parallel programming for GPUs. Available from **https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html**

Menasri, F., Vincent, N., Cheriet, M., & Augustin, E. (2007, September). Shape-based alphabet for off-line Arabic handwriting recognition. **In Ninth international conference on document analysis and recognition** (ICDAR 2007) (Vol. 2, pp. 969-973). IEEE.

Messina, R., and Louradour, J. (2015). Segmentation-free handwritten Chinese text recognition with LSTM-RNN. **In Document Analysis and Recognition (ICDAR), 2015 13th International Conference**. 171-175, IEEE.

Mohamad, R. A. H., Likforman-Sulem, L., & Mokbel, C. (2008). Combining slanted-frame classifiers for improved HMM-based Arabic handwriting recognition. **IEEE transactions on pattern analysis and machine intelligence**, 31(7), 1165-1177.

Mohammed, M. A., Kumar, M. R., and Pradeep, R. (2014). Text Line Segmentation of Arabic Handwritten Documents using Line Height Method. **International Journal**, *4*(11).

Mouhcine, R., Mustapha, A., and Zouhir, M. (2018). Recognition of cursive Arabic handwritten text using embedded training based on HMMs. **Journal of Electrical Systems and Information Technology**, 245-251.

Moysset, B., Kermorvant, C., and Wolf, C. (2017, November). Full-page text recognition: Learning where to start and when to stop. **In Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference**, Vol. 1, 871-876, IEEE.

Mukherjee, P. S., Chakraborty, B., Bhattacharya, U., & Parui, S. K. (2017, November). A hybrid model for end-to-end online handwriting recognition. **In 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)** (Vol. 1, pp. 658-663). IEEE.

Opencv-python tutorials, Image Denoising Page. Last accessed March 8th, 2020. Available from **https://opencv-python tutroals.readthedocs.io /en/latest /py_tutorials/ py_photo/py_non_local_means/py_non_local_means.html.**

Puigcerver, J. (2017). Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition? **In Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference**, Vol. 1, 67-72, IEEE.

Python, xml.etree.ElementTree — The ElementTree XML API Page. November 21th. Available from **https://docs.python.org/3/library/xml etree.elementtree.html#module xml.etree.ElementTree.**

Qaralleh, E., Abandah, G., and Jamour, F. T. (2013). Tuning recurrent neural networks for recognizing handwritten Arabic words. **Journal of Software Engineering and Applications**.

Ravindra Parmar, 2018, towardsdatascience, Common Loss functions in machine learning, Last accessed November 21st, Available from **https://towardsdatascience .com/ common-loss-functions-in-machine-learning- 46af0ffc4d23**.

Ren, S., He, K., Girshick, R., Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. **IEEE Transactions on Pattern Analysis and Machine Intelligence** 39(6), 1137–1149.

Simonyan, K., Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. **CoRR abs/1409.1556http://arxiv.org/abs/1409. 1556**

Soleymani, M., and Razzazi, F. (2003). An efficient front-end system for isolated Persian/Arabic character recognition of handwritten data-entry forms. **International Journal of Computational Intelligence**, 193-196.

Steph Howson , 2018, Python XML with ElementTree: Beginner's Guide, Available from **https://www.datacamp.com/community/tutorials/python-xml-elementtree**.

Stepney, S., Braunstein, S. L., Clark, J. A., Tyrrell, A., Adamatzky, A., Smith, R. E., ... and Milner, R. (2006). Journeys in non-classical computation II: initial journeys and waypoints. **The International Journal of Parallel, Emergent and Distributed Systems**, 97-125.

Svozil, D., Kvasnicka, V., and Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. **Chemometrics and intelligent laboratory systems**, 39(1), 43-62.

Swalhah, S., Abandah, G., (2020). Towards end-to-end machine learning solution for recognizing handwritten Arabic documents. **Unpublished master's degree thesis, University of Jordan, Amman.**

TECHTUTORIALSX, 2019, Python OpenCV: Flipping an image. Available from **https://techtutorialsx.com/2019/04/21/python-opencv-flipping-an-image/**

Tong, A., Przybocki, M., Märgner, V., & El Abed, H. (2014, April). NIST 2013 Open Handwriting Recognition and Translation (Open HaRT'13) Evaluation. **In 2014 11th IAPR International Workshop on Document Analysis Systems (pp. 81-85).** IEEE.

Wahdan, K. A., Hantoobi, S., Salloum, S. A., & Shaalan, K. (2020). A systematic review of text classification research based ondeep learning models in Arabic language. **Int. J. Electr. Comput. Eng**, 10(6), 6629-6643.

Wang, Y., Xiao, W., & Li, S. (2021, April). Offline Handwritten Text Recognition Using Deep Learning: A Review. **In Journal of Physics: Conference Series** (Vol. 1848, No. 1, p. 012015). IOP Publishing.

Wigington, C., Tensmeyer, C., Davis, B., Barrett, W., Price, B., and Cohen, S. (2018). Start, follow, read: End-to-End Full-Page Handwriting Recognition. **In Proceedings of the European Conference on Computer Vision (ECCV)**, 367-383.

Wikipedia, List of countries where Arabic is an official language Page. Last accessed March 30th 2020, b. Available from **https://en.wikipedia.org/wiki/List _of_countries _where_Arabic_is_an_official_language**.

Wikipedia, PyCharm Page. Last accessed November 21st, a. Available from **https://en.wikipedia.org/wiki/PyCharm**.

Younes, M., and Abdellah, Y. (2015). Segmentation of Arabic handwritten text to lines. **Procedia Computer Science**, 73,115-121.

Appendix A

Dataset Preparation Codes

The following is a summary of the codes that were used in the dataset preparation part in the images and XML records of the MADCAT dataset to make them fit for the proposed system:

1) **Filtering The MADCAT Dataset.** After collecting all the data of MADCAT dataset, we started by filtering the data set to reach the final 10,000 images which will be ready for training of our models by done 3 filters on the dataset. The following code describes the libraries and functions that we used to filter the dataset and get unique 10,000 images with their xml design.

```
from os import path
import cv2
import shutil, os
Img_Directory = "/home/eng-reememad/Documents/MADCAT DVDs/phase
1/madcat_p1_tr_d1/data/images/"
XML_Directory = "/home/eng-reememad/Documents/MADCAT DVDs/phase
1/madcat_p1_tr_d1/data/madcat/"
Img_Save_Path = "/home/eng-reememad/PycharmProjects/pyxmlcode/Train-A"
XML_Save_Path = "/home/eng-reememad/PycharmProjects/pyxmlcode/Train-A/page/"
# define empty list
places = []
# open file and read the content in a list
with open('writing_conditions.tab', 'r') as filehandle:
    filecontents = filehandle.readlines()
    for line in filecontents:
        # remove linebreak which is the last character of the string
        current_line = line[:-1]
        # add item to the list
        places.append(current_line)
matches = []
for match in places:
    if "IUN" in match:
        matches.append(match)
    if "IUC" in match:
        matches.append(match)
for line in matches:
    img_string = line
    first_word = img_string.split()[0]
    if path.exists(XML_Directory + first_word +".madcat.xml"):
        original_xml = XML_Directory + first_word +".madcat.xml"
```

```
            if path.exists(Img_Directory + first_word +".tif"):
                originalImage = cv2.imread(Img_Directory + first_word +".tif")
                image = cv2.flip(originalImage, 1)
                cv2.imwrite(os.path.join(Img_Save_Path, first_word +".tif"), image)
                shutil.copy(original_xml, XML_Save_Path)

#Choose 10000 unique samples from a list
all_lines = img_list.readlines()
for i in range(10000):
    line = random.sample(all_lines, 10000)
    all_lines.write(line)
img_list.close()
final_images = all_lines.readlines()
#Get img and xml from list
i = 1
for i in range(final_images):
    img_name = random.choice(os.listdir(Img_Directory))
    if path.exists(Img_Directory + img_name):
        imgname_without_extension = img_name[:-4]
        original_xml = XML_Directory + imgname_without_extension + ".madcat.xml"
        originalImage = cv2.imread(Img_Directory + img_name)
        if originalImage is not None:
            cv2.imwrite(os.path.join(Img_Save_Path, img_name), originalImage)
            shutil.copy(original_xml, XML_Save_Path)
```

2) **Image Processing.** The following code defines the functions that we used to transform the MADCAT images to be fitting with images used in the SFR system (Sawalhah and Abandah, 2020).

```
from PIL import Image
import os
TIF_path = './Tif_1/'
for image_fn in os.listdir(TIF_path):
  if image_fn.endswith('.tif'):
    image = Image.open(TIF_path + image_fn)
    width, height = image.size
    image = image.resize((int(width / 6), int(height / 6)))
    image_n, image_ext = os.path.splitext(image_fn)
    image.save('Train-A/{}.jpg'.format(image_n), dpi=(300, 300))
```

3) **Noise Removal.** To advance the system's effectiveness, we implemented some image processing operations to remove the noise and we use the median filter. The following code represents the function that we handled to remove noise from the MADCAT images (Sawalhah and Abandah, 2020).

```
import os
from PIL import Image
import cv2
import numpy as np
path = './ noise-image/'
for image_fn in os.listdir(path):
    image = Image.open(path + image_fn)
    # https://www.geeksforgeeks.org/python-image-blurring-using-opencv/

    image = cv2.imread(path + image_fn)
    # Median Blur
    median = cv2.medianBlur(image, 3)
    image_n, image_ext = os.path.splitext(image_fn)
    new_name = './no-noise/{}.jpg'.format(image_n)
    cv2.imwrite(new_name, median)
```

4) **Processing Extensible Markup Language (XML) Files.** We employed several libraries like ElementTree and MiniDom to extract the necessary information and convert it from MADCAT XML format to the required format for the SFR algorithm. To make XML files in the MADCAT dataset compatible for formatting XML files in the READ dataset. The following describes the developed code for XML design extraction.

```
 from pathlib import Path
from PIL import Image
from xml.dom.minidom import parseString
import xml.etree.ElementTree as ET
import glob
import math
image_list = []
for filename in glob.glob('/home/eng-reememad/PycharmProjects/pyxmlcode/Train-Ajpg/*.jpg'):
        im = Image.open(filename)
        image_list.append(im)
        im_name = Path(filename).stem
        #Get heigh and width of image
        width, height = im.size

        Coords_region = ''' <Page imageFilename="'''+ str(im_name)+'''.png" imageWidth="''' + str(width) + '''"
imageHeight="''' + str(height) + '''">
                    <ReadingOrder>
                        <OrderedGroup id="ro_1621344993249" caption="Regions reading order">
                            <RegionRefIndexed index="0" regionRef="r1"/>
                        </OrderedGroup>
                    </ReadingOrder>
                    <TextRegion orientation="0.0" id="r1" custom="readingOrder {index:0;}">
                        <Coords points="0,0 ''' + str(width-1) + ''',0 ''' + str(width-1) + ''',''' + str(height-1) + '''
0,''' + str(height-1) + '''"/>  '''
# The code is completed on the next page
```

```
        # Get number of lines using zones
        file = open('/home/eng-reememad/PycharmProjects/pyxmlcode/page/'+ im_name +'.xml','r')
        print(file)
        data = file.read()
        file.close()
        dom = parseString(data)
        Z = len(dom.getElementsByTagName('zone'))
        #make object of xml pages
        madcattree = ET.parse('/home/eng-reememad/PycharmProjects/pyxmlcode/page/'+ im_name
+'.xml')
        #print(madcattree)
        SFRtree = ET.parse('010001.xml')
        Madcatroot = madcattree.getroot()
        SFRroot = SFRtree.getroot()
        input_line = """<TextLine id="r1l1" custom="readingOrder {index:0;}">
                                <Coords points="0,0 0,0 0,0 0,0"/>
                                <Baseline points="0,0 0,0"/>
                                <TextEquiv>
                                        <Unicode></Unicode>
                                </TextEquiv>
                        </TextLine>"""
        head = """<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
        <PcGts xmlns="http://schema.primaresearch.org/PAGE/gts/pagecontent/2013-07-15"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schema.primaresearch.org/PAGE/gts ">
                <Metadata>
<Creator>Transkribus</Creator>
                        <Created>2021-05-17T14:33:20.298+03:00</Created>
                        <LastChange>2021-05-18T16:36:33.203+03:00</LastChange>
                </Metadata>
                """
        tail = """ </TextRegion>
          </Page>
        </PcGts>"""
        # Create The lines in the page
        Text_lines_str = ''
        for i1 in range(1, Z+1):
                X = []
                Y = []
                zonepoints = madcattree.find('.//zone[@id="z' + str(i1) + '"]')
                for point in zonepoints.iter('point'):
                        X.append(point.attrib['x'])
                        Y.append(point.attrib['y'])
                del X[0:4]
                del Y[0:4]
                newX=[]
                for elem in X:
                        elem = int(elem) / 6
                        elem =int(width) - int(elem) - 1
                        newX.append(elem)
                inversX = newX[::-1]
                x1 = inversX[::2]
                x2 = inversX[1::2]
                # get the length of the list
                mx = len(x2)

# The code is completed on the next page
```

```
            # flip the elements in the second array
            for i2 in range(0, mx - 2, 2):
                index1 = x2.index(x2[i2])
                index2 = x2.index(x2[i2 + 1])
                x2[index1], x2[index2] = x2[index2], x2[index1]
            x2inverse = x2[::-1]
            index3 = x2inverse.index(x2inverse[0])
            index4 = x2inverse.index(x2inverse[1])
            x2inverse[index3], x2inverse[index4] = x2inverse[index4], x2inverse[index3]
            x2inverse=x2inverse[::-1]
            x1 = x1 [::-1]
            X = x1 + x2inverse
            newYYY = []
            for elem in Y:
                elem = int(elem) / 6
                elem = math.ceil(elem)
                newYYY.append(elem)
            inversY = newYYY[::-1]
            m = len(inversY)
            newY1 = []
            newY2 = []
            for v1 in range(0, m, 4):
                newY1.append(inversY[v1])
                newY1.append(inversY[v1 + 1])
            newY1=newY1[::-1]
            for v2 in range(2, m, 4):
                newY2.append(inversY[v2])
                newY2.append(inversY[v2 + 1])
            #newY2 = newY2[::-1]
            Y = []
            Y = newY1 + newY2
            zoneref2 = madcattree.find('.//zone[@id="z' + str(i1) + '"]')
            contentstr = ''
            Coords_linef = ''
            Baselinef = ''
            m = len(X)
            for tockennumber in zoneref2.iter('token-image'):
                        zonecontent = madcattree.find('.//token[@ref_id="' + tockennumber.attrib['id'] + '"]')
                        contentstr += ' '
                        for content in zonecontent.iter('source'):
                            contentstr += content.text
                        input_data1 = '''<TextLine id="r1l''' + str(i1) + '"" custom="readingOrder {index:''' +
str(i1) + ''';}">
                                                    <Coords points=""''
            for i4 in range(0, m):
                Coords_line = str(X[i4]) + ''',''' + str(Y[i4]) + ''' '''
                Coords_linef += Coords_line
            for i4 in range(0, m//2):
                Baseline = str(X[i4]) + ''',''' + str(Y[i4]) + ''' '''
                Baselinef += Baseline
            input_data3 = '""/>
                            <Baseline points=""''
            input_data5 = '""/>
            <TextEquiv>
            <Unicode>''' + contentstr + '''</Unicode>
            </TextEquiv>
            </TextLine>'''
# The code is completed on the next page
```

```
            input_data = input_data1 + Coords_linef + input_data3 + Baselinef + input_data5
            Text_lines_str += input_data
        # print the full text using for loop
        contentfullstr = ''
        for source in Madcatroot.iter('source'):
            word = source.text
            contentfullstr += word
            contentfullstr += " "
        full_text_tag1 = """<TextEquiv>
                                <Unicode>"""
        full_text_tag2 = """</Unicode>
                            </TextEquiv>
        """
        finalroot = head + Coords_region + Text_lines_str + full_text_tag1 + contentfullstr + full_text_tag2 +
tail
        #print(finalroot)
        SFRxml = open("page_sfr/"+ im_name +".xml", "w")
        SFRxml.write(finalroot)
        SFRxml.close()
```

حل متكامل للتعلم الآلي للتعرف على الوثائق العربية المكتوبة بخط اليد

إعداد
ريم عماد شتيوي

المشرف
الأستاذ الدكتور غيث علي عبندة

ملخص

في الوقت الحاضر، اكتسب البحث عن حلول للتعرف الآلي على النصوص المكتوبة للغات المختلفة اهتمامًا متزايدًا. وهذا مرتبط بالحاجة المتزايدة لرقمنة المستندات العربية في العديد من التطبيقات مثل استكشاف المستندات الكبيرة، والفرز الآلي للبريد السريع، والتحرير للوثائق المطبوعة السابقة، ومعالجة الرقابة المصرفية. للأسف، على الرغم من عقود من البحث، لا يوجد بعد حل مرضٍ للتعرف على الخط المتصل مثل اللغة العربية لصعوبة ذلك.

تم إدخال تقنيات مختلفة في مجالات البحث العلمي في لغة الألة لشرح مشاكل هذا التعرف لعدة لغات مختلفة. التطورات الحديثة في التعلم العميق وخوارزميات التعرف تدعم أنظمة البناء القادرة على التعامل مع كل من المرحلة ثنائية الأبعاد للمدخلات والمرحلة اللاحقة من التنبؤ. وتجدر الإشارة إلى أن هذه الخوارزميات هي الشبكات العصبية المتكررة للذاكرة طويلة المدى متعددة الأبعاد (MDLSTM-RNNs) مقارنة بالوظيفة الموضوعية للتصنيف الزمني للوصلة (CTC). مثل هذه الأساليب تعطي معدلات خطأ منخفضة وأصبحت من أحدث تقنيات هذا النوع من التعرف.

تهدف هذه الأطروحة إلى دراسة إمكانية تنفيذ حل التعلم الآلي الشامل من خلال تطبيق نظام التعلم العميق باستخدام مجموعة بيانات MADCAT، للتعرف بدقة على المستندات المكتوبة بخط اليد باللغة العربية بعد التعلم المتزامن لاكتشاف النص وتقسيمه والتعرف أخيرًا على المستندات العربية وتحويلها إلى نص قابل للتحرير.

حققت الخوارزمية المتكاملة لدينا، الناتجة عن دمج عدة خوارزميات متميزة لتحديد اللغة اللاتينية، دقة عالية في تحليل الصفحة كاملة وتحويلها إلى نصوص إلكترونية، ثم توقع الكتابة داخل كل سطر. تتكون الخوارزمية الخاصة بنا من ثلاثة نماذج، الأول هو تحديد بداية السطر (نموذج SOL)، والثاني هو متابعة الخط من البداية إلى النهاية، وقصه وتطبيعه (نموذج LF)، وأخيراً توقع

النص المكتوب (نموذج HTR). من خلال تطبيق حلنا على مجموعة البيانات العملاقة التي تحتوي على مجموعة كبيرة ومتنوعة من الخطوط والمشكلات المختلفة التي يجب معالجتها.

حقق نظام البدء والمتابعة والقراءة نتائج متطورة على مجموعة بيانات MADCAT. حيث بلغ معدل الخسارة لشبكة اكتشاف بداية السطر في مجموعة بيانات مجموعة تحليل الوثائق وتصنيفها آلياً متعدد اللغات، وصل معدل الخسارة إلى 34.62. أما بالنسبة لشبكة تتبع السطر، فقد بلغ معدل الخسارة 73.92. وأخيرًا، بلغ معدل الخسارة في شبكة التعرف على الكتابة 3.96 %.