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

By
Safaa Al-Sawalhah

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

May, 2020

COMMITTEE DECISION

This Thesis (Towards 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) Professor of Computer Science and Engineering	_____
Dr. Andraws I. Swidan, (Member) Professor of Computer Engineering	_____
Dr. Mohammad R. Abd-Almajeed, (Member) Assistant Professor of Computer Engineering	_____
Dr. Esam A. AlQaralleh, (Member) Associate Professor of Computer Engineering (Princess Sumaya University for Technology (PSUT))	_____

DEDICATION

To my parents for their prayers and endless support

To my supporter and my beloved Hitham

To my little girl Salma

To my brothers and my sisters

To auntie Um Hitham and uncle Abu Hitham

ACKNOWLEDGEMENT

I would like to thank my supervisor Prof. Gheith Abandah for his advices, 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 the Transkribus-1.7.1 team for their cooperation with us in the data preparation phase. I would also like to thank the authors of the Start, Follow, Read code for making it available for free to researchers and academics for application to their scientific research.

List of Contents

List of Figures

List of Tables

List of Abbreviations

1D-RNN	One-Dimensional Recurrent Layers
2D-LSTM	Two-Dimensional- Long Short-Term Memory
API	Application Programming Interface
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
DDR3	Double Data Rate 3
DPI	Dots Per Inch
GPU	Graphics Processing Unit
GT	Ground Truth
HMM	Hybrid Hidden Markov Model
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
LDC	Linguistic Data Consortium
LF	Line Follower
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
JU-OCR2	Jordan university- optical character recognition 2
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-11	Very Deep Convolutional Network for Large-Scale

	Image Recognition
XML	Processing Extensible Markup Language

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

By

Safaa Al-Sawalhah

Supervisor

Dr. Gheith Ali Abandah, Prof

ABSTRACT

The Arabic language is Islam language and the language of Al-Quran Al-Kreem. The Arabic language is one of the oldest languages in the world and it is also a Semitic language. Handwriting recognition (HWR) is a sub-science of pattern recognition.

Currently, the search for offline Arabic handwriting recognition solution has received increasing attention. This is attributed to the increasing need to digitize Arabic documents in several applications such as searching in huge documents, automatic sorting of postal mail, convenient editing of previously printed documents, and bank check processing. Unfortunately, despite decades of research, there is no satisfactory solution for recognizing the cursive handwriting like Arabic scripts.

Several methods have been proposed in the literature to solve HWR problems. The recent advances in deep learning and recognition algorithms allowed building systems capable of handling both the two-dimensional aspect of the input and the sequential aspect of the prediction. Prominent of these algorithms is the multi-dimensional long short-term memory recurrent neural networks (MDLSTM-RNNs) associated with the objective function of the connectionist temporal classification (CTC). Such approaches give low error rates, and are becoming the state-of the-art of HWR.

This thesis aims to find a system that uses end-to-end machine learning approaches in order to recognize handwritten Arabic documents. Therefore, we have studied the possibility of using the Start, Follow, Read (SFR) system to recognize handwritten Arabic documents. SFR is a deep learning system that jointly learns text detection, segmentation and recognition.

In order to obtain good training using machine learning techniques, it is required to have a huge dataset. Therefore, we chose a large handwritten dataset in Arabic, the Multilingual Automatic Document Classification Analysis and Translation dataset (MADCAT).

SFR was applied on the Recognition and Enrichment of Archival Documents (READ) dataset. The READ dataset consists of two sets: Train-A and Train-B dataset. The Train-A is small set of 50 pages with manually specified baselines and transcripts. The Train-B set does not contain any text-line information or layout. The corresponding texts are available at the page level, with line breaks. The Train-B dataset consists of 10 thousand pages divided into two batches, each of which contains 5 thousand pages.

The SFR system consists of three networks. The start of line (SOL) network is a region proposal network (RPN) to find the starting location of text lines. The line follower (LF) network follows and handles possibly curved lines of blurred images to be recognize by third network: the line-level HWR network which is a convolutional neural network-long short-term memory (CNN-LSTM). SFR outperformed the winner of handwriting recognition competition of the International Conference on Document Analysis and Recognition (ICDAR) 2017, even when the submitted competition annotations were not used.

Unfortunately, after we study the MADCAT dataset we found fundamental differences between the READ dataset and the MADCAT dataset. So we took a number of steps to prepare the data of the MADCAT dataset to be suitable with the SFR. One of these steps is communicating with the Transkribus-1.7.1 team to prepare the data for out Train-A set. In order to prepare Train-B set, we converted the MADCAT dataset ground truth information to be suitable for the SFR. We also performed a number of image preprocessing before applying the SFR system on the MADCAT dataset to recognize the Arabic handwritten documents.

SFR achieves state-of-the-art results when applied to huge and difficult datasets: READ and MADCAT datasets. The loss rate in the Start of Line (SOL) network is (47.9) in the READ dataset, while in the MADCAT dataset, the loss rate is (21.9). As for the Line Follower (LF) network, the loss rate is (39.1) for the READ dataset, while for the MADCAT dataset it is (47.0). Finally, the character error rate (CER) in the handwritten recognition (HW) network on the READ dataset is (0.52) and is (0.25) on the MADCAT dataset.

CHAPTER I

Introduction

This chapter clarifies the importance of recognizing Arabic handwritten documents and briefly explains the research area to which it belongs. The most important methods used today to recognize handwritten documents in other languages. It also clarifies the research objectives of this thesis, its most important contributions, and the outline of this thesis.

1.1. PATTERN RECOGNITION

Pattern recognition is an important field in artificial intelligence and machine learning applications. Pattern recognition is interested with automatic discovery of patterns and data regularities (Stepney, et al., 2006). Pattern recognition consists of many different fields such as fingerprint recognition, face recognition, image recognition, character and number recognition, etc. (El-Sawy, et al., 2016). Extraction of features is one of the most complex issues in pattern recognition, but it is often not needed now through end-to-end machine learning. The machine bears the burden of this problem (Belabiod, et al., 2018).

Handwritten recognition (HWR) is a subfield of pattern recognition. HWR is defined as the ability of the computer to receive and translate handwriting from several sources such as photographs, paper documents, touch screens and other devices (Lorigo, et al., 2006). HWR uses supervised or semi-supervised learning (classification approach), offline or online learning, and instance-based or model-based (Bluche, 2016). HWR is one of the key problems studied by the document community due to its pervasiveness in the places where people interact, communicate and transact (Dutta, et al., 2018).

1.2. HANDWRITING RECOGNITION

There are two types of HWR: 1) Static (offline), which recognizes printed or handwritten text, but is more common with printing. 2) Dynamic "online" recognition that uses electronic pens or touch screens (Belabiod, et al., 2018). Online recognition is usually easier than offline handwriting recognition (Abandah and Malas, 2010).

The methods based on individual characters were extensively used in the 1990s. Then they were gradually replaced by sliding window approaches, where features are derived from the vertical frames of the line image. This led to the transformation of the problem into a sequence to sequence. The convolutional neural networks were used to process the two-dimensional nature of the image, and identify the relevant features (Bluche, 2016).

In traditional methods, HWR consists of a number of stages starting with pre-processing of scanned images to prepare them for the final stages. The scanned page is segmented into lines and words. The words are segmented into characters. Features are extracted from segmented characters. Then characters are recognized using trained models. And finally, post-processing is performed to improve the recognition accuracy (Abandah and Malas, 2010).

Most state-of-the-art HWR systems are worked at the line level by converting the text line image to a series of feature vectors (Chammas, et al., 2018). The input in offline HWR is a variable-sized two-dimensional image, and the output is a sequence of characters (Bluche, 2016). The recognition of offline handwriting is a difficult problem that does not have satisfactory general solutions yet. The key challenges include the interplay of adjacent characters, the dramatic change in both the quality and style of human handwriting and the similarities between character shapes (Abandah, et al., 2014b). The HWR problem is divided into two stages:

- The image (word or line) is given to the recognizer to predict the string of characters.
- Then it is given to the linguistic engine that constrains the characters to form a valid word (Dutta, et al., 2018).

Handwriting segmentation is traditionally a necessary step in the HWR and is designed to try to extract all text line areas. This process becomes difficult in the case of handwriting, with irregular gaps or overlapping lines, etc. (Belabiod, et al., 2018).

The majority of previous research has been trying to focus on the work of segmenting the document into lines and detecting the line before the recognition process and a few of them tried to merge the three processes. The process of segmenting the document into lines is the most difficult step in HWR. Most of the errors in the HWR are caused in the segmentation of the document into lines rather than in the recognition of the word or characters. However, line detection and segmentation is usually done using separate algorithms and in separate ways. Several HWR systems are designed, trained, and evaluated only in the context of segmentation of the ground truth line. But a few works attempted to combine detection, segmentation, and recognition (Wigington, et al., 2018).

1.3. IMPORTANCE OF RECOGNIZING ARABIC HANDWRITTEN DOCUMENTS

Quran language is Arabic, and currently Arabic is the fifth most language prevalent in the world. The number of Arabic speakers is estimated at 422 million (native and non-native) people. The Arabic language is the fifth most languages spoken in the world (Wikipedia b, 2020). Arabic is important in the culture of many countries (Abandah, et al., 2014b). The Arabic alphabet is used in about 27 languages including Arabic, Persian, Kurdish, Urdu, and Jawi (Lewis, 2009).

The Arabic alphabet consists of 28 letters (Abandah, et al., 2015). The Arabic writing is arranged from right to left. The Arabic text is always cursive (Abandah and Malas, 2010). Table 1 shows all the forms of Arabic letters. Each letter has two or four shapes depending on its position in the word (Govindaraju, et al., 2006). The choice of the desired shape depends on the location of the character within the word or part of Arabic word (sub-word) (PAWs) (Abandah, et al., 2014b).

The cursive handwriting recognition such as Arabic script is an open research area in pattern recognition (Mouhcine, et al., 2018). For the cursive handwriting recognition, it is difficult to develop effective segmentation algorithms that need significant linguistic knowledge. However, they make significant gains in recognition accuracy, because they provide a richer feature and allow recognition of open vocabulary (Abandah, et al., 2014a).

HWR techniques can be used to organize and manage modern classrooms, search educational videos, analyze data on medical texts (Dutta, et al., 2018). In addition to that, it is allowing the automatic reading of ancient Arabic manuscripts and many other industries. Despite decades of research, there are no satisfactory solutions for recognizing the cursive handwriting like Arabic text (Abandah and Jamour, 2014).

1.4. PROBLEMS STATEMENT

Recently, the increasing need to digitize Arabic documents in many applications has increased the search for finding radical solutions to recognize offline handwritten Arabic documents. But despite decades of research, there are no satisfactory solutions to recognize cursive handwriting like Arabic texts.

The cursive handwriting recognition is difficult to recognize. In order to develop effective segmentation algorithms for recognizing cursive handwriting texts, it requires significant linguistic knowledge.

Many approaches have been suggested in the literature to solve HWR problems. Given the recent advances in deep learning and new algorithms it was allowed to build systems capable of handling both the two-dimensional side of the inputs and the sequential aspect of the prediction. In the machine learning techniques, we need to have a large amount of training data to get a good training.

In this project we have studied the possibility of applying end-to-end machine learning approaches to recognize handwritten document, especially in Latin languages, to recognize Arabic handwritten.

We used the Start, Follow, Read (SFR) system to recognize Arabic handwritten documents. SFR is a deep learning system. It uses deep learning approaches to recognize historical handwritten documents in German language. SFR outperformed the winner of handwriting recognition competition of the International conference on document analysis and recognition (ICDAR) 2017. SFR used to recognize Recognition and Enrichment of Archival Documents (READ) dataset. SFR jointly learns to discover, segment and recognize text.

SFR includes three models. 1) The Start of Line (SOL) model is used to find the location of the beginning of the text lines. 2) Line Follower (LF) model, used for tracing lines and addressing possible curved lines for blurred images. 3) The line-level HWR model is used for text recognition.

1.5. MODERN TECHNOLOGY USED IN HWR

Recent progress in deep learning and new algorithms has allowed the construction of systems capable of handling both the two-dimensional aspect of the input and the sequential aspect of the prediction. Especially multi-dimensional long short-term memory recurrent neural networks (MDLSTM-RNNs) associated with the objective function of the connectionist temporal classification (CTC) (Das, et al., 2018). Resulting

in low error rates, and becoming the state-of-the-art of HWR (Bluche, et al., 2017).

In order to obtain good training on machine learning techniques, it is required to have a huge amount of data. Therefore, we chose a large Arabic handwritten dataset, the Multilingual automatic document classification analysis and translation dataset (MADCAT) (Géron, 2017). MADCAT dataset was created by the linguistic data consortium (LDC) to standardize the training, testing, and evaluation methods used for pre-processing, recognition and translation of Arabic documents. The MADCAT dataset was created in a controlled environment. The documents were written by native Arabs who are fluent in Arabic. Original authentic Arabic texts were selected for the dataset from reliable sources of texts from newspapers and news sites (Abandah and Al-Hourani, 2018). Figure 1 shows an example from MADCT dataset.

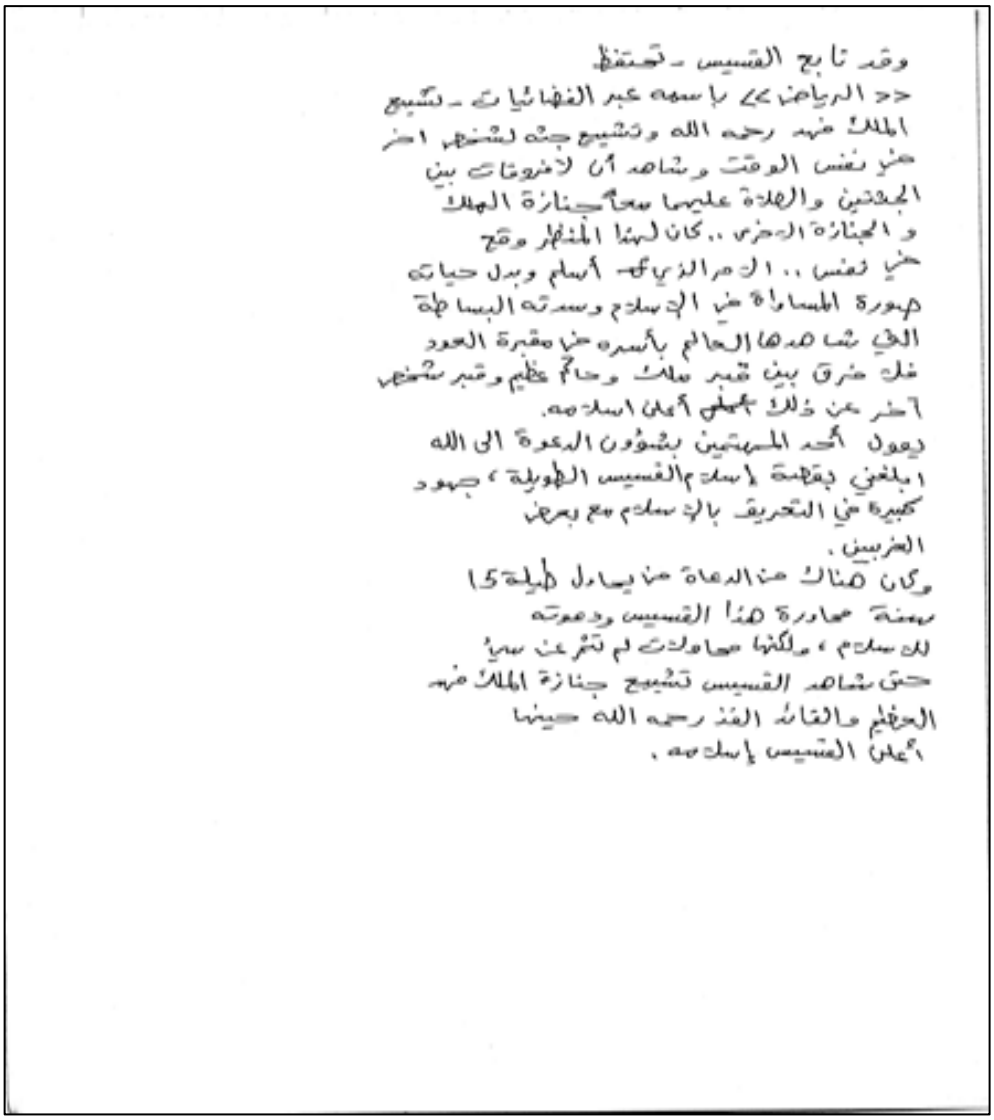

Figure 1: MADCAT Document

1.6. RESEARCH OBJECTIVES AND CONTRIBUTIONS

The main objectives and contributions of this thesis project are summarized as follows:

- Investigating the accuracy of previous systems for segmenting or recognizing handwritten documents in Arabic and Latin languages.
- Studying, analyzing, and preprocessing the MADCAT dataset.
- Investigating the state-of-the-art, end-to-end machine learning approaches especially deep learning to segment handwritten Latin and Arabic documents into lines.

- Investigating the state-of-the-art, end-to-end machine learning approaches especially deep learning to recognize handwritten Latin and Arabic documents.
- Investigating the possibility of applying SFR system especially SOL network to find the start of the line in the handwritten Arabic documents.
- Investigating the possibility of applying SFR system especially LF network for segmenting the handwritten Arabic documents into lines.
- Investigating the possibility of applying SFR system especially line –level HWR network in recognizing the Arabic handwritten documents.

Finally, we believe that this thesis is a good start to use state-of-the-art end-to-end machine learning approaches to segment handwritten Arabic documents. It also provides a good approach to support the use of machine learning methods, especially deep learning, in recognizing handwritten Arabic documents.

1.7. THESIS OUTLINE

The rest of this thesis is organized as follows. Chapter 2 presents a review of the literature from previous approaches developed to solve the problem of recognizing handwritten documents for Arabic and other languages. Chapter 3 describes the solution that we have adopted to segment Arabic handwritten documents into lines and recognize them. Also, it describes the networks and sub-models used in this solution. Chapter 4 explains the software used to perform data processing. Chapter 5 describes the experiments that we did in this work, results of the SFR system when applied on READ and MADCAT datasets, and a comparison with state-of-the-art SFR systems when applied on READ dataset and MADCAT dataset. Finally, Chapter 6 gives the conclusions and suggestions for future work.

Table 1: The Arabic Alphabet (Govindaraju, et al., 2006)

Name	Isolated	Initial	Medial	Final
Alif	ا	_	_	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	د
Thaal	ذ	ذ	ـذ	ذ
Raa	ر	_	ـر	ر
Zaay	ز	_	ـز	ز
Siin	س	سـ	ـسـ	ـس
Shiin	ش	شـ	ـشـ	ـش
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	_	_	و
Yaa	ي	يـ	ـيـ	ـي

CHAPTER II

Literature Review

The segmentation of the handwritten text can be divided into three methods: bottom-up methods, top-down methods, and machine learning methods. Often, line and word segmentation functions are separated because of the difference between line and word (Belabiod, et al., 2018). The following sections survey the best methods used in this field.

2.1 BOTTOM-UP AND TOP-DOWN METHODS

The bottom-up and top-down methods requires advanced knowledge of documents, such as document orientation, inter-spaces, etc. So such systems must combine all types of processing methods. Common methods of word segmentation are bottom-up methods based on connected component analysis and extraction of structural feature or even both. These methods have good results on handwritten Latin or Germanic documents (Belabiod, et al., 2018). Below are examples of the most important work used in this field.

Abandah and others (2010) used feature selection approaches to evaluate wide range of features extracted from handwritten Arabic characters. They extracted 96 features from the secondary components of the characters, the main body, the skeleton, and the borders. In order to select the best feature subsets, they used five different feature selection techniques. Then they evaluated these feature subsets to select a subset of features that gives the highest accuracy. After that, the recognition accuracy was analyzed using three classifiers. Figure 2 summarizes the approaches used in this system.

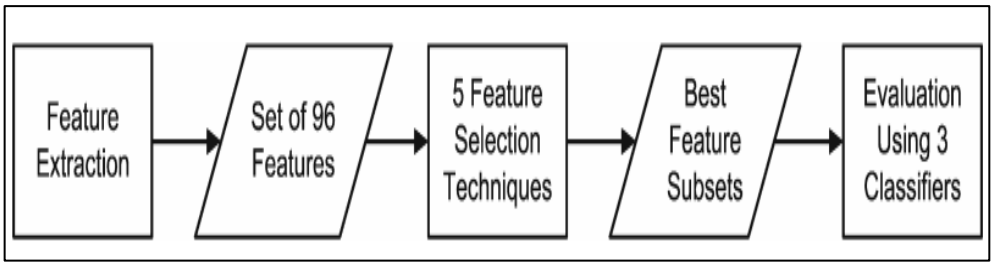

Figure 2: Feature Extraction, Selection, and Evaluation

2.2 MACHINE LEARNING METHODS

The machine learning methods process the image as a whole without any previous knowledge. Some existing systems are end-to-end integrated systems, with no further post or preprocessing, while the others use document knowledge. Deep learning in line segmentation is used more than word segmentation (Belabiod, et al., 2018). Machine learning methods were used in literature to recognize the word, or to fully recognize the text. The following is a review of the most important literature used in machine learning methods to recognize handwritten texts.

2.2.1 MACHINE LEARNING METHODS TO RECOGNIZE WORDS

Graves (2008) introduced MDLSTM-RNNs for the first time in the context of handwriting recognition, which won the ICDAR 2009 competition. Figure 3 shows the architecture of this network. The long short-term memory (LSTM) layers scan the input in the four possible directions, and calculate the internal state of the LSTM cell, and the output of the cases and outputs of the previous positions in the horizontal and vertical directions. Each MDLSTM layer follows a convolutional layer. There is one feature map per character, at the top of this network. These maps were folded into a sequence of prediction vectors, which were normalized using softmax activation. For all possible sequence labeling, the CTC algorithm is applied to train the network to recognize text lines. The conversion from two-dimensional (2D) to one-dimensional (1D) occurs in the collapse layer. The collapse layer computes a simple aggregation of the characteristic

maps in the vector sequences. After that all information is reduced in the vertical dimension to one vector, regardless of its location in the features map.

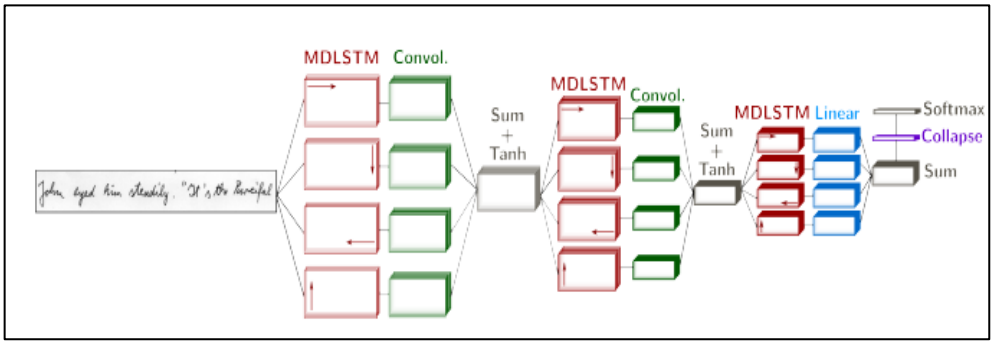

Figure 3: MDLSTM-RNN Architecture for Handwriting Recognition

Abandah and others (2014b) introduced the Jordan university-optical character recognition2 (JU-OCR2) system to recognize handwritten Arabic words. This system achieved high accuracy using efficient segmentation, feature extraction, recurrent neural network (RNN), and word matching. This system is based on the segmentation of cursive words to graphemes. Most characters are segmented to one grapheme per character. Some characters are segmented into multiple graphemes. Also, some letter combination is segmented in one grapheme. A set of features are selected from a rich set of features of the segmented bodies. To achieve high recognition accuracy, the feature vector is passed to a bidirectional LSTM (BLSTM) network. The CTC is used in the output layer. The RNN was built using the recurrent neural network Library (RNNLIB) library. The transcriber maps the vectors of the entities split directly into characters and achieve a low label error rate (averaging 5.55% for the Institute of communications technology / Ecole nationale d'Ingénieurs de Tunis (IFN / ENIT database) without using a dictionary. Figure 4 summarizes the main phases used in this system and demonstrates how the system corrects the miss-transcribed last character using word matching.

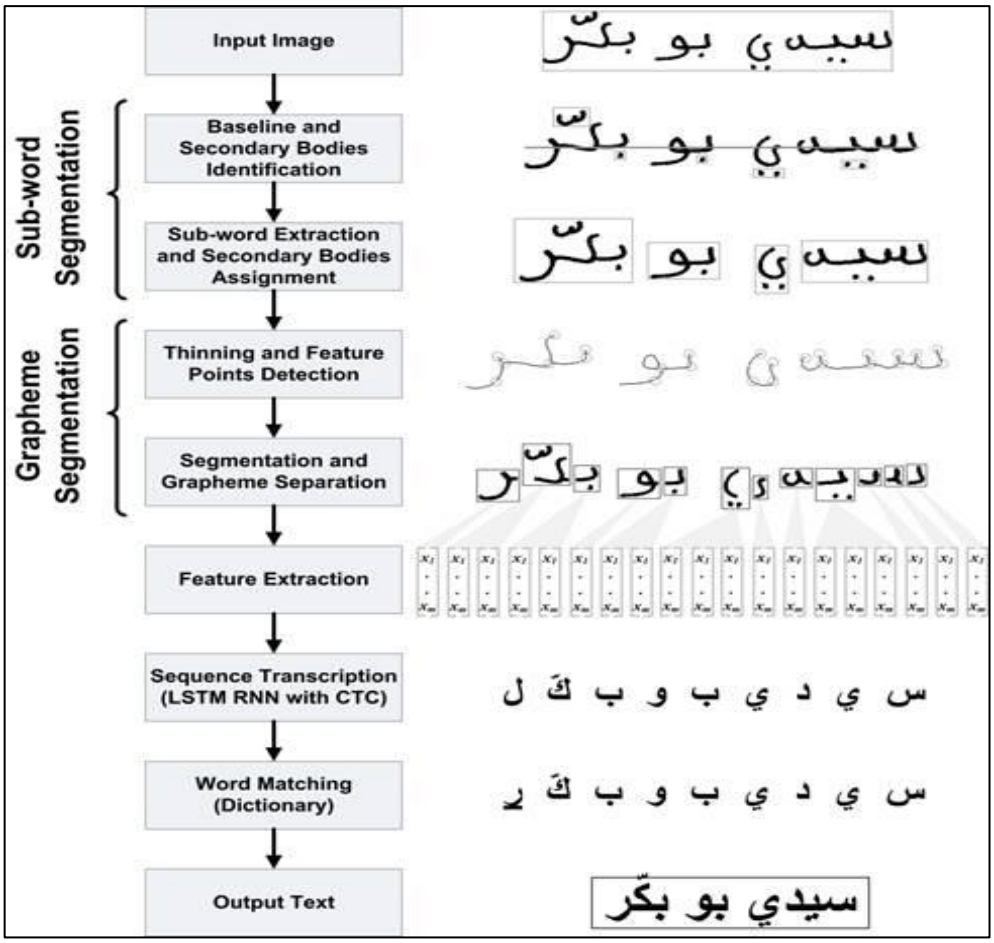

Figure 4: JU-OCR2 Phases

Castro and others (2018) proposed handwriting recognition at the line level. The system is based on MDLSTM within the framework of the hybrid hidden Markov model (HMM). The proposed model modifies the standard MDLSTM structure by modifying the order of the layers and the number of pooling layers. Unlike previous methods, which focused only on the modification of the width of the network, the proposed model was balanced between the width of the network and its depth to obtain the best topology. The system was evaluated using the IAM dataset, and gave a good performance and reduce the execution time. The complete system including a decoder with linguistic knowledge offers competitive results. Figure 5 illustrates the architecture of the system.

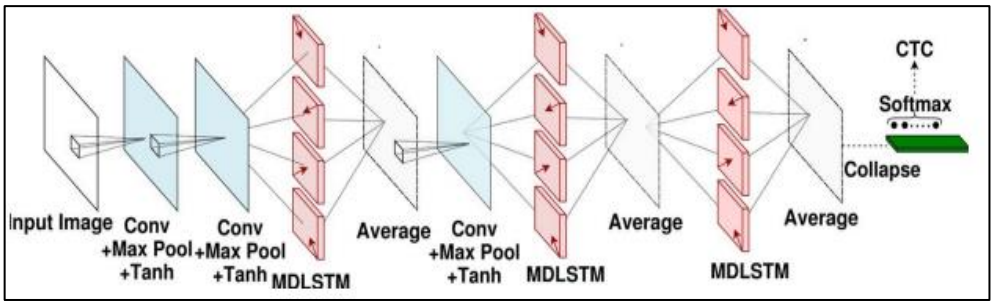

Figure 5: The Proposed MDLSTM Architecture by Castro and others (2018)

2.2.2 MACHINE LEARNING METHODS TO RECOGNIZE TEXT

Bluche proposed (2016) a modification to MDLSTM-RNN in order to enable end to end process handwritten paragraphs. He replaced the collapse layer by converting two-dimensional representation into a sequence of predictions with a recurrent version that could select one line at a time. The neural network is a type of implicit line segmentation by calculating the weights of attention to images representation. Figure 6 shows the adjustment made to the collapse layer. While the standard collapse (left, top) is a simple amount. The weighted collapse includes a neural network to predict weights from a weighted amount.

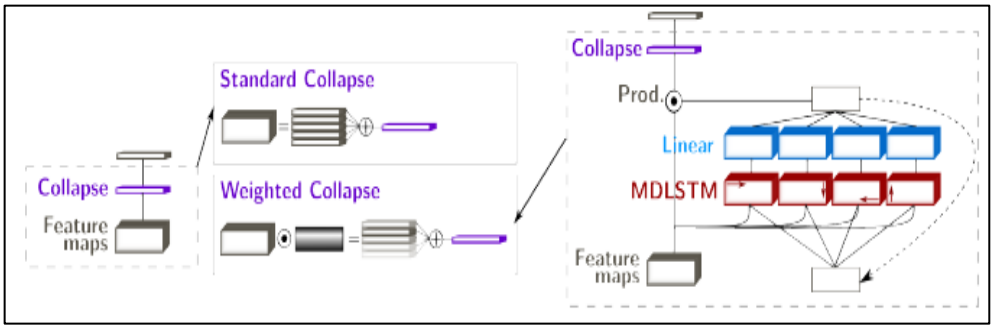

Figure 6: The Modification of the Collapse Layer

Bluche and others (2017) proposed a hard attention system of multi-line handwriting recognition mechanism, requiring no segmentation of the input paragraph. This model is inspired by different interest models used for speech recognition, images, and translation. The main difference between the proposed system and the other system is the implementation of MDLSTM. The primary importance of this system is that it is

the first proposed system for handwriting recognition in automatic transcription without previous segmentation into lines. This step is very important in the previous methods. The iterative algorithm finds the next point of interest on a series of simplified profiles designed through the hidden state of a repetitive network. The system gave good results when applying the IAM dataset.

Moysse and others (2017) proposed a system to recognize the entire pages of the heterogeneous documents that can recognize and detect text in different languages. The method focuses on predicting the starting point (left) of the text. The end point (right) is predicted by to two-dimensional- LSTM (2D-LSTM). The full text of the page is recognized in two steps: First a neural network detects where to start to recognize a text line. A second network performs the text recognition process and decides when to stop the process. In the neural network, the previous network detects the side effects of each text line, by predicting the value of the object's position coordinates as a regression problem to recognize the full text of the page. The localization of text lines depends on gradients with fully CNN and MDLSTM as contextual layers. To increase the efficiency of this method of settlement, only the left side of the text lines is expected. The text recognizer is then responsible for predicting the end of the text to recognize it. This method has shown good results for recognize the full text of the page on the data set of heterogeneous Maurdor. Figure 7 illustrates the architecture of this system.

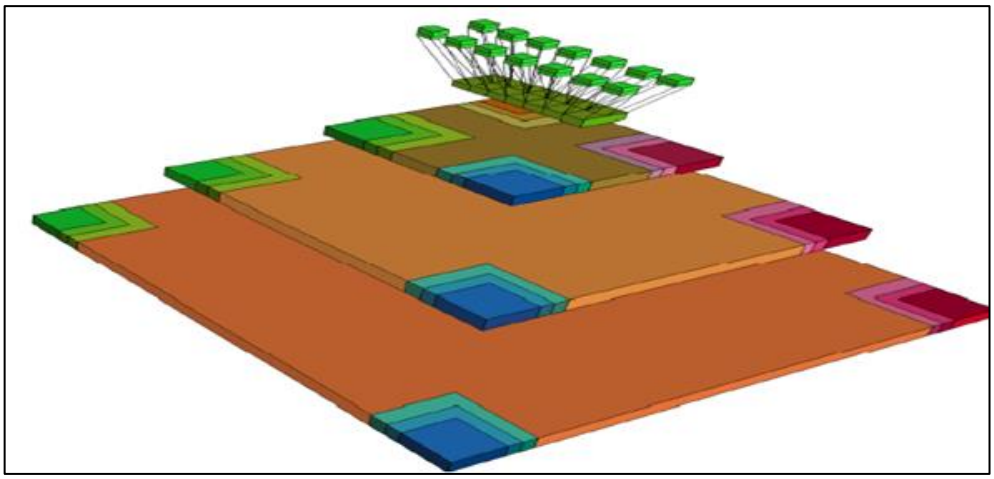

Figure 7: Convolutional Recurrent Neural Network that Locally Predicts the Object Positions

Puigcerver (2017) proposed a model that depends only on the convolutional network (ConvNets, CNN) and one-dimensional recurrent layers (1D-RNN). He claims that he achieves results better or equivalent to the results achieved by using MDLSTM-RNNs with faster execution time as a result of the reduced computation cost. Figure 8 illustrates the architecture of this system.

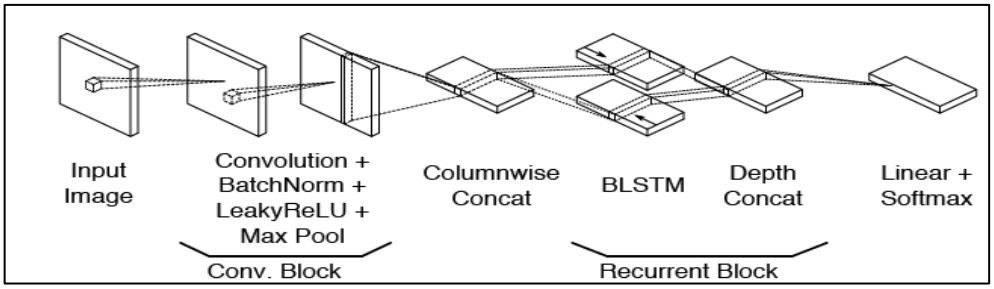

Figure 8: Architecture of the Neural Network using Bidirectional 1D-LSTM

Belabiod and others (2018) proposed end-to-end system on line segmentation using the recurrent residual convolutional neural network based on U-Net (RU-net) (Alom, et al., 2018). It is a comprehensive system for word segmentation, using BLSTM and finally a CTC function. The system automatically learns the alignment between text line images and the words in the transcription.

Chammas and others (2018) proposed the convolutional recurrent neural network (CRNN) system to recognize the text line of historical documents. This system achieved

second place in the competition ICDAR2017. This system was trained using small number of labeled text line data, through a gradual training process with only 10% of the manually labeled text line data from the READ 2017 dataset. In order to improve the performance of the system, they created syntactic data to increase the size of few labeled data in training set submitted by the ICDAR2017competition (i.e. only 10% of the pages are segmented into lines and the rest is transcript in the paragraphs level). This system can be applied to recognize the variance in the write scale. Figure 9 illustrates the model-based normalization approach.

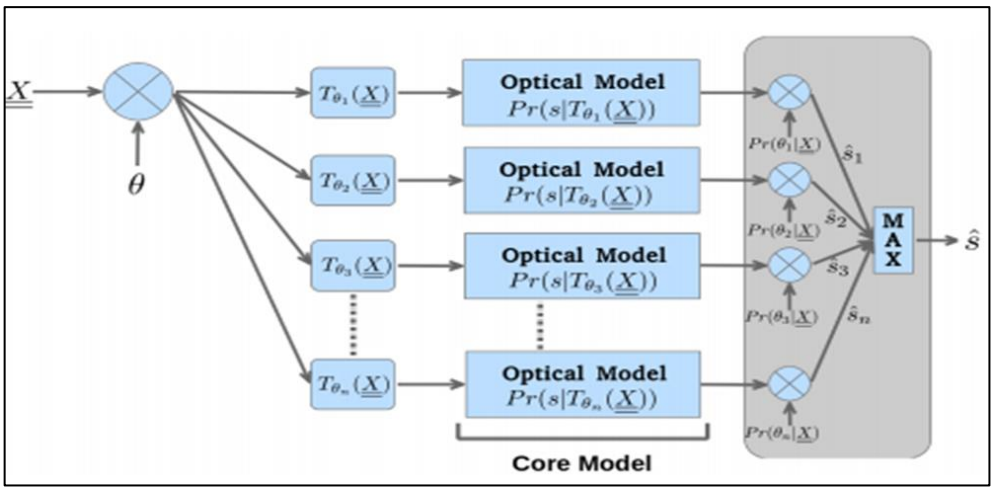

Figure 9: Model-Based Normalization Approach

Wigington and others (2018) proposed the end-to-end SFR system to recognize full-page handwriting. This system consists of 3 sub-models: 1) SOL 2) LF and 3) line-level HWR model. The SOL is a regional proposed network (RPN), where the suggested areas represent the location and direction of text lines in a particular document. The LF model begins in every predicted SOL position. The LF takes incremental steps along the text line, and after bending, produces a normal text image. The LF model can split the curved text and normalize it. After that, the HWR model predicts a transcription of the measured line image.

Chowdhury and others (2018) proposed an end-to-end neural network for recognizing the handwritten text at the line level. The network consists of a deep CNN to extract features and the frequent encoder-decoder network with attention. The encoder-decoder network is primarily used to generate the output sequence from the input sequence. The system was trained using Focal loss. The Beam Search algorithm was used to enhance the decoding capacity of the model. Figure 10 illustrates the system.

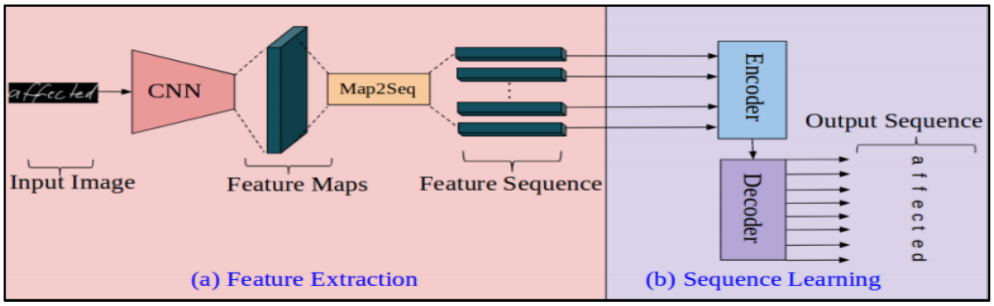

Figure 10: The Proposed Model Architecture in Chowdhury and others (2018)

Dutta and others (2018) proposed a system that used a hybrid CNN-RNN structure to recognize handwritten text. The system consists of a spatial transformation layer (STN), then a set of residual convolutional network (ReseNet18) blocks, followed by BLSTM stacks, and finally a linear layer for transcribing the labels. The STN is an end-to-end trainable layer that geometrically transforms inputs to correct handwriting distortions due to shifting hand movements. The ReseNet18 are used to learn the sequence of feature maps. It is then passed as an input to the stacked BLSTMs. The last ReseNet18 is given as input to BLSTMs as a sequence of feature vectors. The loss function is CTC. Figure 11 illustrates the architecture of a hybrid network CNN-RNN used in this system.

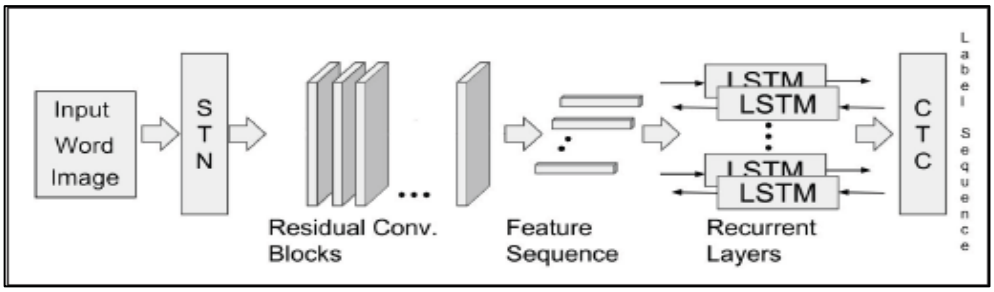

Figure 11: The Architecture of Hybrid Network CNN-RNN Used in the Proposed System

CHAPTER III

Technologies Used

When using neural networks in several applications, many researchers have obtained amazing results (Jamro, et al., 2016). Neural networks have two types of learning algorithms; supervised learning, and unsupervised learning (Svozil, et al., 1997). In this project, we used supervised learning because it could conclude a general function depending on the training data and it would be able to test the data in a reasonable way. There are many researchers who applied end-to-end machine learning approaches especially deep learning to recognize handwritten documents in Latin language.

This chapter briefly clarifies the technologies that we used to recognize the handwritten Arabic documents. The main contributions to the model we used to recognize the Arabic handwritten documents. And finally briefly explain the methodology and the networks that the system consists of them.

3.1 START, FOLLOW, READ (SFR) MODEL

Our proposed system directly depends on the SFR models. SFR is end-to-end full-page handwriting recognition model. SFR aims to recognize offline deteriorating handwritten historical documents. SFR focuses on recognizing historical handwritten documents, which is one of the most difficult HWR tasks. It can also be applied to other ranges of HWR. SFR outperformed the winner of the ICDAR2017 handwriting recognition competition. SFR provided deep learning models that learn together to discover text, segment it, and recognize it using images mostly without disclosure or hash annotations. SFR consists of 3 sub-models:

1. SOL finder to find the start position of lines of text. SOL finder is a RPN. 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. LF network starts at every position predicted by SOL, steps along the text line, follows the lines of the text (possibly curved), and converts them into appropriate normalized text images.
3. Line-level HWR network. The HWR model predicts transcription of the normal line image. Figure 12 shows how SOL, LF and HWR networks work to handle document images.

The SFR model detects all lines of the text efficiently in a single path. It makes use of preprocessing before applying the HWR model to each line independently. This allowed each line to be recognized in parallel.

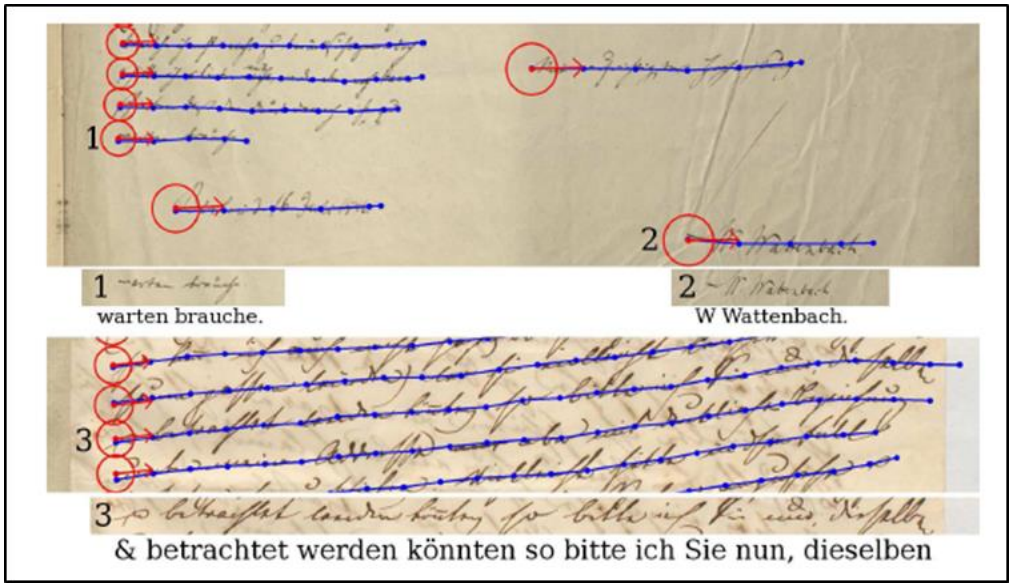

Figure 12: Red circles and arrows view the SOL network work . The blue lines view the LF network work. The transcriptions view the HWR network work

3.2 MAJOR CONTRIBUTIONS OF SFR

We chose the SFR system to recognize Arabic handwritten documents after studying several systems that use machine learning approaches to recognize handwritten documents in other languages, especially in the Latin languages. This is because there are many contributions and achievements of the SFR system. The following is a review of the most important of these contributions and

accomplishments:

- Improved SOL network architecture, and its training system which increases the recognition performance.
- LF network proposal, which can segment and normalize, curved text.
- The combined training of the three components is largely a set of images that contain only transcriptions which allow the SOL, LF, and HWR to adapt to each other and supervise each other.
- Prove that LF and HWR networks can be used to derive and refine potential SOL targets; this method requires only pre-training in a small number of images (e.g. 50) with additional hash marks.

3.3 NETWORK DESCRIPTION

The SFR system is proposed for jointly learning to detect, segment, and recognize the text. SFR consists of three networks: SOL network, LF network, and HWR network. The following is an illustration of these networks.

3.3.1 Start-of-Line Network (SOL)

The SOL is a RPN network. RPN detects the starting points of the text lines. RPN takes any image (of any size) as input. Its outputs are a set of proposals from rectangular object, each with a degree of interception. This represents the entire convolutional network. (Ren, et al., 2017).

RPN is the only true and effective backbone for detecting object using Faster R-CNN. The main goal of RPN is to suggest multiple recognizable objects within a given image. Figure 13 shows the Faster R-CNN architecture. RPN contains a regression and classifier. Regression determines the coordinates of the proposals. The classifier determines the probability of the proposal having the target object (Karmarkar, 2018).

As shown in Figure 14, the SOL network used a truncated very deep convolutional network for large-scale image recognition (VGG-11) architecture

instead of the MDLSTM to predict the density of SOL positions (Kaggle, 2020). VGG-11 is a profound convolutional network of object recognition developed and trained by the renowned Oxford Visual Geometry Group (VGG), which performed very well in the ImageNet dataset (Simonyan, et al., 2014).

In order to obtain a correct image, SOL network returns the coordinates (x0, y0), scale s0, rotation 0, as well as the probability of occurrence p0. As shown in Figure 14 in the red box. The network should predict p0 = 1, otherwise 0. In order to obtain a predictive stride of 16x16, the fully connected and final pooling layers from VGG-11 were removed. Therefore, like Faster R-CNN, the expected coordinates (x, y) represent an offset relative to the correction center. Scale and rotation correspond to handwriting size and text line slant.

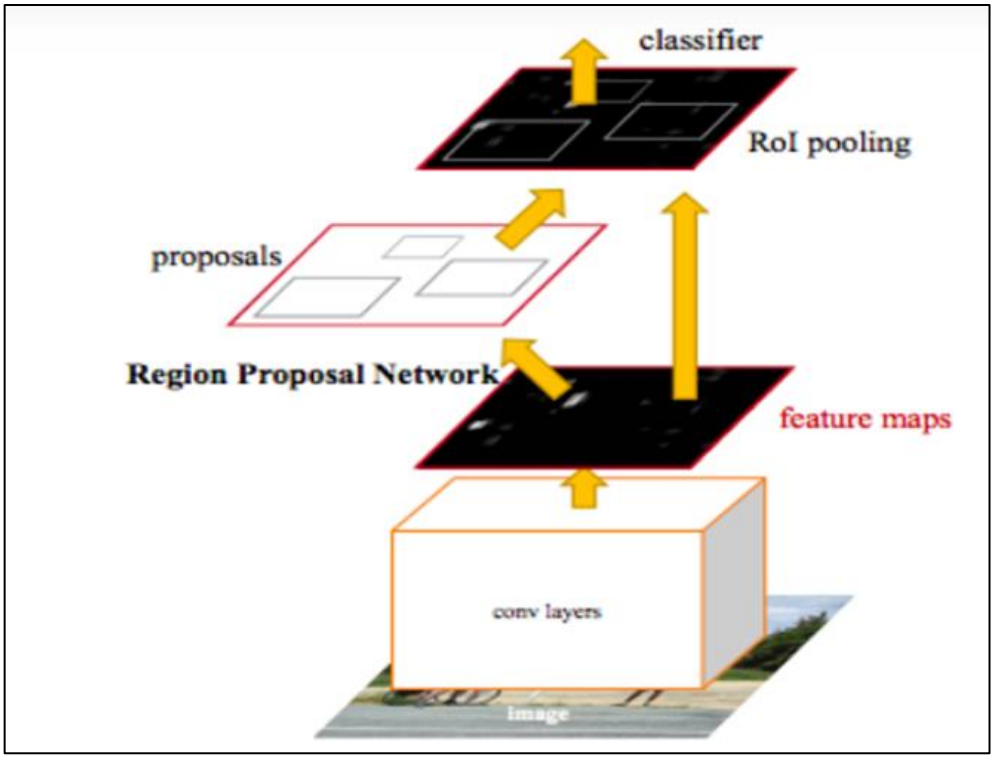

Figure 13: Faster R-CNN architecture

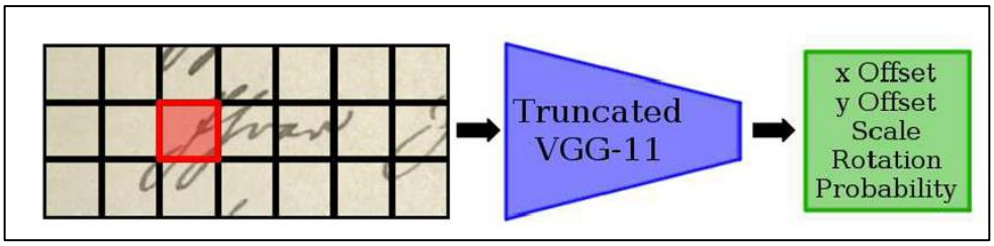

Figure 14: The SOL network predicts x and y, scale, and rotation angle. The probability of each correction are 16x16 input

3.3.2 Line Follower (LF)

As shown in Figure 15, after the SOL network determined the start position of the line, the LF network begins to operate. It traces handwriting in incremental steps and then outputs a distorted text line image be convenient for HWR. LF network is segmented polygonal regions and are able to arbitrarily straighten and following curve text.

The LF is a recurrent network, given the current location and rotation angle$(\chi i, \mathcal{Y} i, \theta_i)$. As shown in Figure 15a it represents a small display window (the red square in the Figure 15a). It is then fed to the CNN network for regression $(\chi i + 1, \mathcal{Y} i + 1, \theta\_i + 1)$. Figures 15b, 15c, and 15d show that this process repeats up to the edge of the image. To determine the end of the text line, SFR uses the HWR network during the training process. A predicted SOL is used to determine the initial position and rotation. The SOL scale is also used to determine the size of the display window and remains constant.

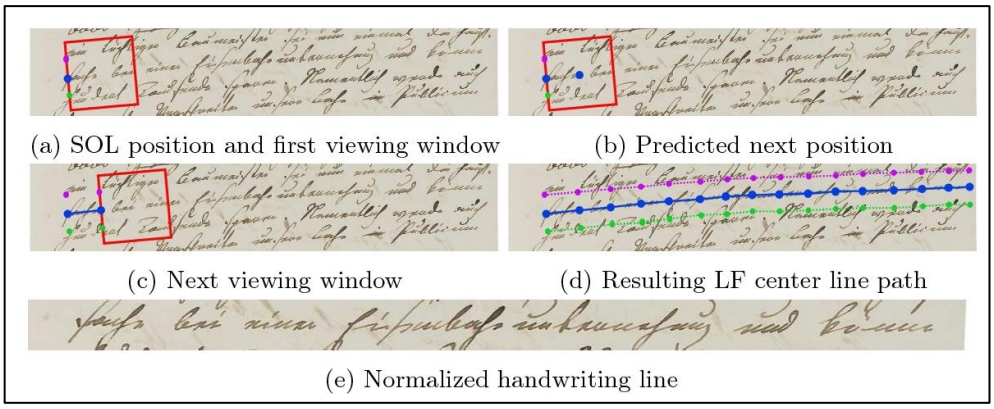

Figure 15: a) LF starts after SOL. b) Retracts from the new position indicated by the second blue dot. c) The following input is a new display window. d) Repeat the process until it reaches the edge of the image. d) The purple and green lines show the segmentation. e) Produce normalized handwriting line

As shown in Figure 16, the input image is reshaped in order to obtain the display window using an affine transformation matrix as the input coordinates of the image are determined to display the image coordinates. This allows the LF network to be back propagated its errors by viewing the windows.

The first display window matrix,(W0 = AWSOL) consists of the mapping defined using the (WSOL) transformative SOL matrix (defined by the SOL prediction network values) and the matrix of look-ahead(A). Equation 1 shows how we can calculate W.

$$W = \begin{bmatrix} \frac{1}{s_0} & 0 & 0 \\ 0 & \frac{1}{s_0} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_0) & -\sin(\theta_0) & 0 \\ \sin(\theta_0) & \cos(\theta_0) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix}, \; A = \begin{bmatrix} 0.5 & 0 & -1 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

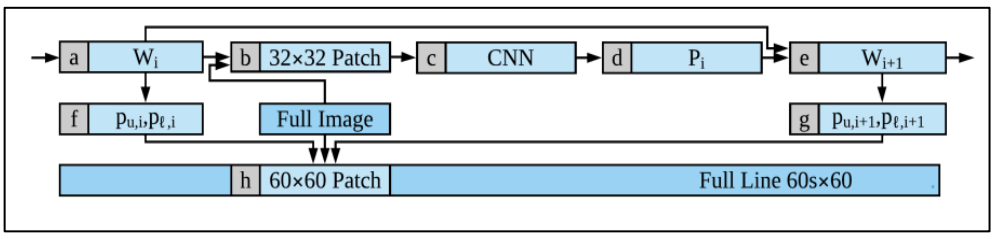

Figure 16: a) The current transformation Wi. b) $\mathbf{32} \times \mathbf{32}$ patches reshaped. c) CNN regresses the transformation change. d) Calculate the following transformation. e) The $\mathbf{60} \times \mathbf{60}$ patch is reconfigured using the upper and lower points $(\boldsymbol{f}, \boldsymbol{g})$ of the LF path.

The look-ahead provides sufficient context to LF network to properly follow the lines. For each step(i), SFR system extracts a 32×32 display window correction by reconfiguring it using(Wi). After reconfiguration, the coordinates of (x, y) in the area

are normalized to the range$(-1, 1)$. Given the correction of the display window$(\mathrm{i}-1)$, the regression of network $(\mathrm{xi}, \mathrm{yi}, \text{and } \theta\mathrm{i})$ are used in the formulation matrix of the prediction(Pi). (Pi) is calculated as shown in equation 2.

$$P_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

To get the output image for the HWR network, the SFR system firstly represents the path of the normalized handwriting line measured as a series of upper and lower coordinate's pairs, $(\mathrm{pu}, \mathrm{i} \text{ and } \mathrm{pL}, \mathrm{i})$ (purple and green lines in Figure 3d. This is calculated using the multiplier points down and middle of the predicted windows through their inverse transformations. Equation 3 shows how $(\mathrm{pu}, \mathrm{i} \text{ and } \mathrm{pL}, \mathrm{i})$ can be calculated.

$$P_{u,i}, P_{l,i} = \begin{bmatrix} x_{u,i} & x_{l,i} \\ y_{u,i} & y_{l,i} \\ 1 & 1 \end{bmatrix} = W_i^{-1} A \begin{bmatrix} 0 & 0 \\ -1 & 1 \\ 1 & 1 \end{bmatrix} \quad (3)$$

The handwriting line extracted using mapping of $(\mathrm{pu}, \mathrm{i}, \mathrm{p}\ell, \mathrm{i}, \mathrm{pu}, \mathrm{i}+1, \mathrm{p}\ell, \mathrm{i}+1)$ to 60×60 patch corners. Then all these patches are joined to form a complete 60×60 handwriting line where (s) is the number of LF steps.

The LF architecture is 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) applied and after layers 1, 2, 4, and 6 (2x2) Max Pooling (MP) applied. The fully connected layer was used for regress the output$(\mathrm{i.e}: \chi, \mathcal{Y}, \theta)$, with initial of (X) bias parameters initialized to 1 and for (Y) and (θ) the biases initialized to 0. Figure 17 shows the architecture of the LF network.

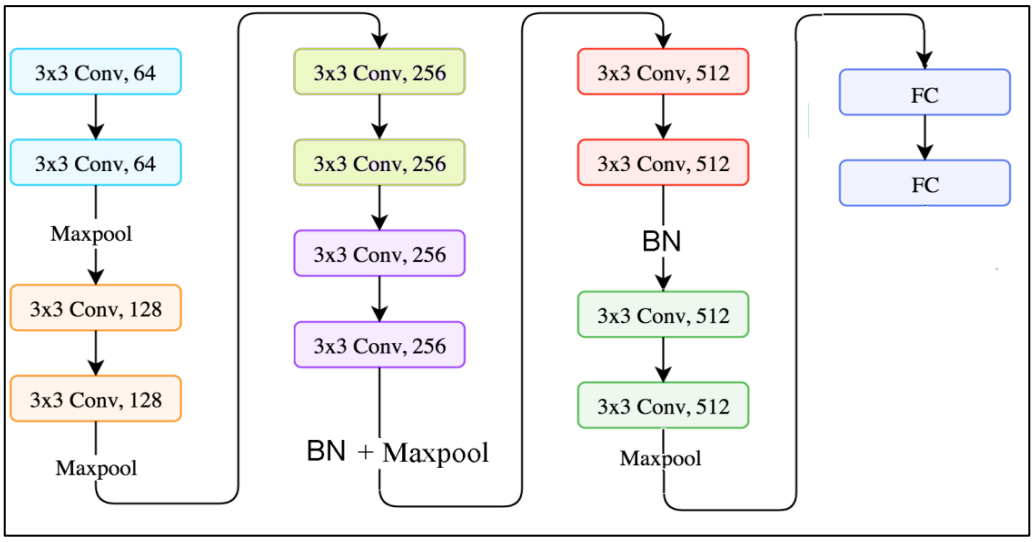

Figure 17: The LF architecture

3.3.3 Handwriting Recognition (HWR)

To produce transcription, the normal line image is fed to the convolutional neural network-long short-term memory CNN-LSTM network after it is produced by the LF network. CNN is a portion from a HWR network as it recognizes high-level features and is folded vertically to create a 1D horizontal sequence that is fed into a BLSTM model. In BLSTM, the context learned features are propagating forward and backward through the length of the sequence before applying the character classifier to every output step time.

The character prediction output sequence is much longer than the transcriptions of the ground truth (GT), because it includes an empty character for use in decoding step (Graves, et al., 2006). Decryption is performed by the first folding non-empty duplicate characters and then removing the blanks.

The HWR network architecture in SFR system consists of the CNN-LSTM HWR network and is similar to the LF network in it. The input size represent as(W x 60), where (W) varies dynamically. The HWR network consists of 6 convolutional layers with 3x3 kernels with feature maps (64, 128, 256, 256, 512 and 512) respectively. After layers 4 and 5 the BN is applied, and after layers 1, 2 apply

(2×2) MP (stride 2). After layers 4 and 6, use the(2×2) MP with a vertical step of 2 and a horizontal step of 1 to collapse the features vertically. To form a series of 1024 dimensional feature vectors, the features are connected vertically, which are fed into two BLSTM layers, each of them consist of 512 hidden nodes and the probability of dropout node is 0.5. Finally, in each time step, the fully connected layer is applied to produce character classifications. Figure 18 shows the HWR network architecture.

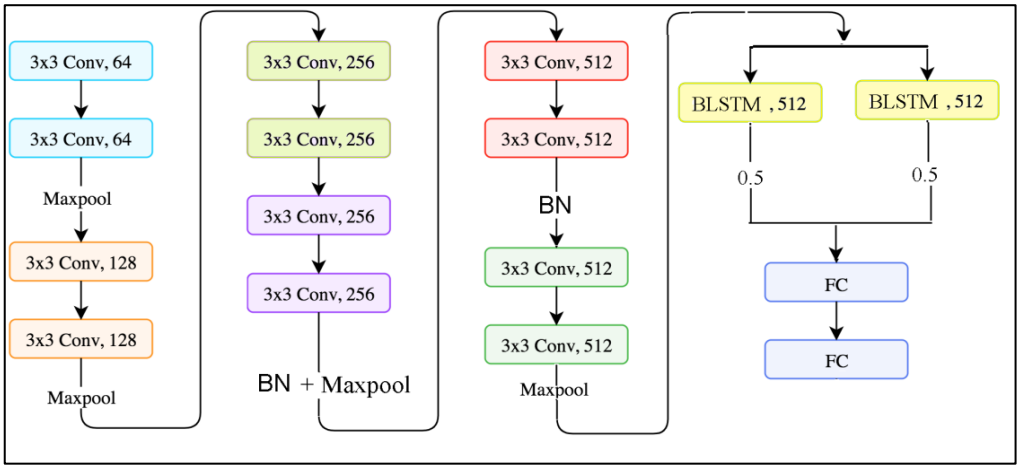

Figure 18: The HWR network architecture

CHAPTER IV

Dataset Preparation

This chapter aims to study, analyze, and preprocess the MADCAT dataset to be suitable to SFR system. The data preparation is the second step of the seven steps followed in end-to-end machine learning after the data gathering process. The following is a review of the most important topics discussed in this chapter.

- Getting, studying and analyzing the MADCAT dataset.
- Prepare the data for SFR model.

4.1 GETTING, STUDYING, AND ANALYSIS OF THE MADCAT DATASET

Our experimental data was from the MADCAT dataset. MADCAT dataset contains many styles of handwriting in training and testing images. Unfortunately, the MADCAT dataset has many problems in its dataset such as pepper noise, vertical lines, external graphical elements, writing errors, varying text body, irrelevant text or numerals (non-Arabic characters), varying text body, short text lines, slant text, skew text, and bleed-through text. (Abandah et al, 2018). So we make many pre-processing steps on the MADCAT dataset, to be suitable for SFR model.

4.2 PREPARE THE DATA FOR START, FOLLOW, READ MODEL

We performed several pre-processing operations in order to make the MADCAT dataset suitable for the SFR model. The following is a review of the most important of these processes.

4.2.1 Image Processing

Images stored in the MADCAT dataset are in the tagged image file format (tiff) format with dots per inch (DPI) equal to 600*600 dpi. But the SFR model assumes that the images pass to it are in the joint photographic experts group (JPG)

format. The training and testing dataset in the SFR system is from the READ dataset, which is divided into two sets. Train_A and Train_B. Train_A consist of 50 images in (JPG) format or portable network graphic (PNG) format with DPI equal to 300*300 dpi. Train_B consists of two batches; each batch includes (5000) images with JPG format and with different DPI from one image to another. So we develop python code to convert the images in the MADCAT dataset to be suitable with images used in the SFR model. This code uses a library Python imaging library (PIL). It is a package from Python script that contains many functions for processing images (geeks3d, 2020). The following code defines a function that we used to convert the MADCAT images to be suitable with images used in the SFR model.

```
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 / 2), int(height / 2)))
        image_n, image_ext = os.path.splitext(image_fn)
        image.save('Train-A/{}.jpg'.format(image_n), dpi=(300, 300))
```

4.2.1.1 Noise Remove

Generally, noise is a random variable with zero mean (opencv, 2020). In order to improve the system's effectiveness, we performed a number of image processing operations to remove noise and we did this using a median filter.

The median filter is known as a non-linear digital filtering technology. Median filter is usually used to remove noise from signals or images. Median filtering is widely used in digital image processing as it maintains edges under certain conditions during noise removal.

The median filter calculates the average pixel intensity that surrounds the central pixel in the (n x n) kernel. The median then replaces the pixel intensity in the center of the pixel. The median filter does a better job of removing salt and pepper noise from mean and Gaussian filters. The average filter retains the edges of the image but does not deal with spot noise.

The "medianBlur" function from the Open-CV Library can be used to implement a median filter (Geeksforgeeks, 2020). The following code defines a function that we used to remove noise from the MADCAT images.

```
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)
    cv2.imshow('image', image)
        # Median Blur
    median = cv2.medianBlur(image, 3)
    image_n, image_ext = os.path.splitext(image_fn)
    cv2.imshow(image_n, median)
    cv2.waitKey(0)
    new_name = './no-noise/{}.jpg'.format(image_n)
    cv2.imwrite(new_name, median)
```

4.2.1.2 Images Rotation

The images are read in the Arabic language from right to left, while the SFR model assumes that the images are read from left to right, as assumed by the Latin language reader. So we flipped the direction of the images in the

MADCAT dataset to be fit with the SFR model. The following code defines a function that we used to rotate the MADCAT images.

```
import cv2
from PIL import Image
import os
originalImage =
cv2.imread('/home/salma/PycharmProjects/SFR_Image_preprocessing/sol_val_2/4.png')
file_path='/home/salma/PycharmProjects/SFR_Image_preprocessing/sol_val_2_R/'
image = cv2.flip(originalImage, 1)
cv2.imwrite(os.path.join(file_path, '4.png'), image)
```

4.2.2 PROCESSING EXTENSIBLE MARKUP LANGUAGE (XML) FILES

The data in XML is formatted hierarchically, and the common way it is represented is the tree (Python Page, 2019). The basic importance of the tree structure to XML is that it makes navigation, modification, and removal programmatically simple. XML files are widely used in web-scraping and general practice in structured document analysis. XML files contain a number of sections called elements, known by the beginning (<) and end (>) tag. Items can contain tags for other sub-items. The first element in the top level is called the root, which contains all the other elements. 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).

The SFR model used XML files from the READ dataset, which also includes two different formats to each of the Train_A and Train_B dataset where they differ from the XML files in the MADCAT dataset.

4.2.2.1 Format of the Train_A Dataset

In order to get format of Train_A dataset, we have contacted a Transkribus-1.7.1 team. Transkribus is presented as a research infrastructure through the H202 project (READ) (Transkribus b Page, 2019). The following steps explain how to convert MADCAT XML files to be appropriate with the images used in the

Train_A dataset in the SFR model are explained below:

1. We register on the website of the Transkribus (Transkribus c Page, 2019).
2. We download Transkribus from the website (Transkribus a Page, 2019).
3. Open Transkribus (Transkribus b Page, 2019). Steps to work on the Transkribus program:
4. We ran the tool and used the Login button on the Server tab. Figure 19 shows the Transkribus login mechanism (Transkribus b Page, 2019).

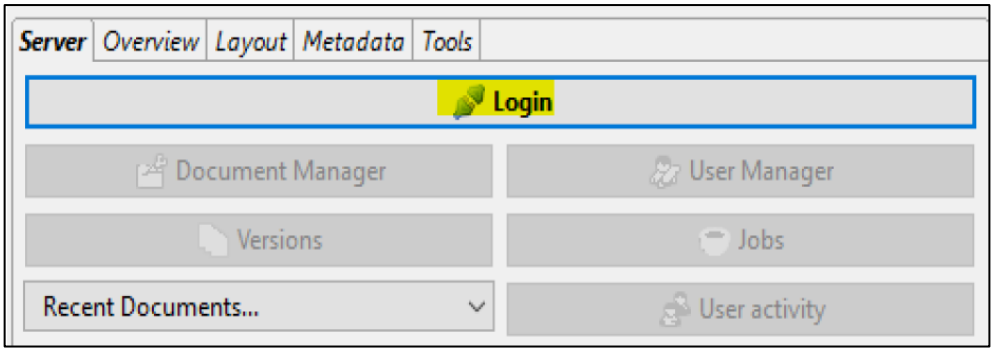

Figure 19: Transkribus login mechanism

5. We have uploaded our documents. This operation can be done either locally or by uploading them to the server.
6. Images must reside in a separate folder on our computer before uploading them to Transkribus.
7. In order to transfer images from our computer to the platform. We click the "Import Document (s)" button. Figure 20 and 21 shows the import document button.

Figure 20: Import Document (s)" button

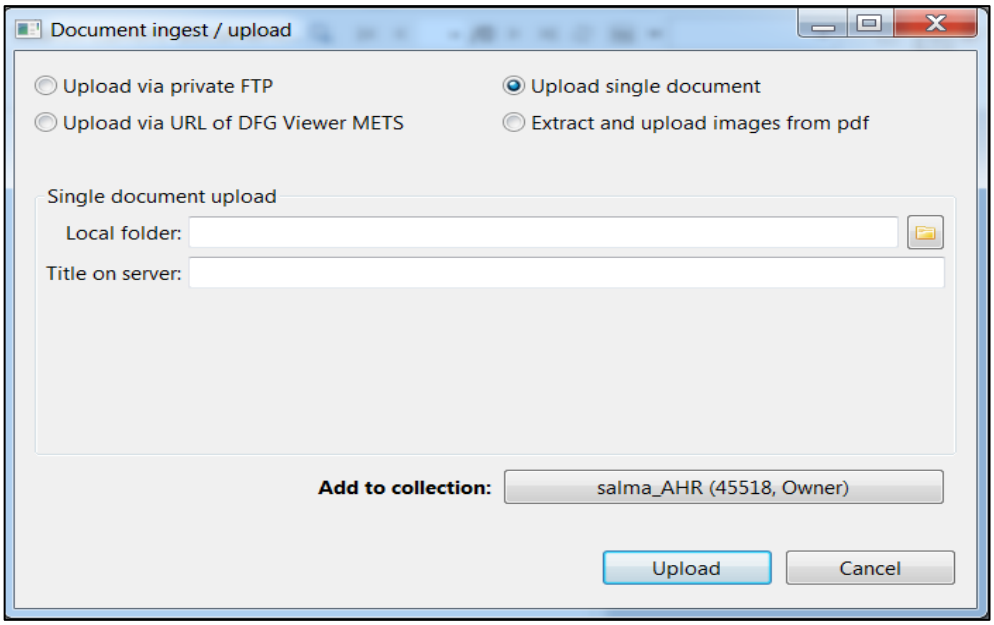

Figure 21: Import document mechanism

8. We create a new collection (Train-A-L-Salma) by clicking the "Collections" button at the bottom of the "Server" box and then clicking "Create". Figure 22 and 23 shows how we can create a new collection mechanism.

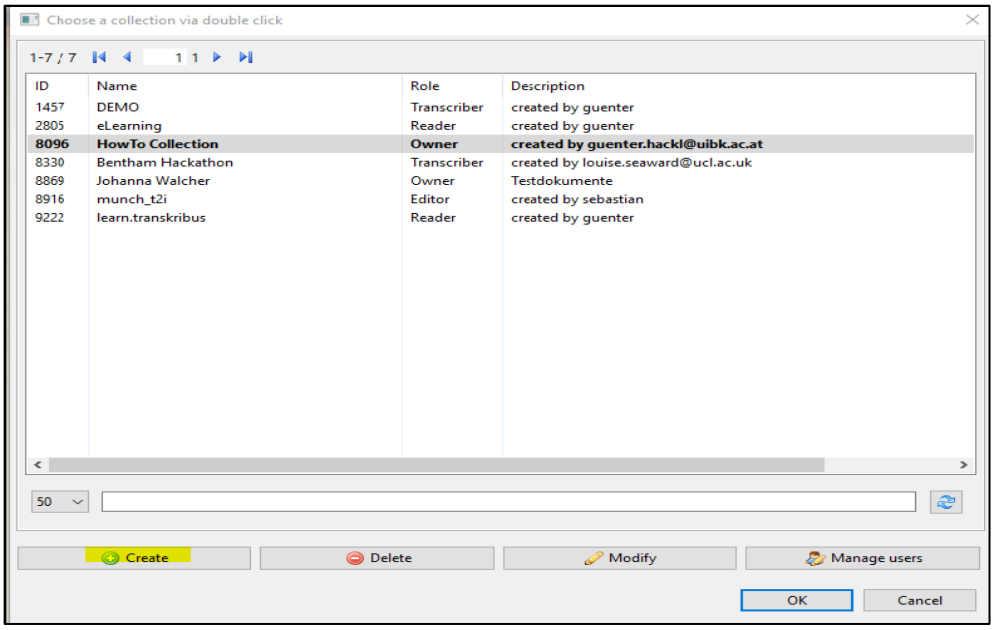

Figure 22: Create new collection mechanism

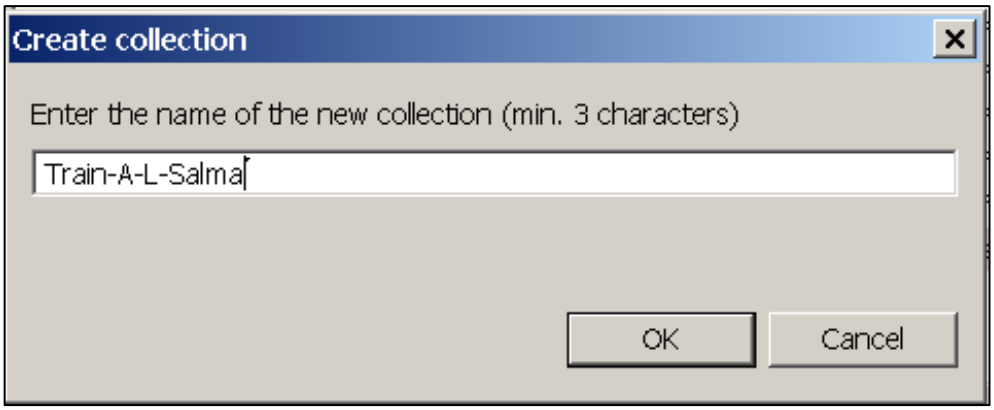

Figure 23: Create Train-A-L-Salma collection mechanism

9. To access our own documents (Train-A-L-Salma), we click on the "Collections" button on the "Server" tab and select our collection. Then double-click the documents in the box below the Server tab to open them. Figure 24 shows how we can access (Train-A-L-Salma) documents.

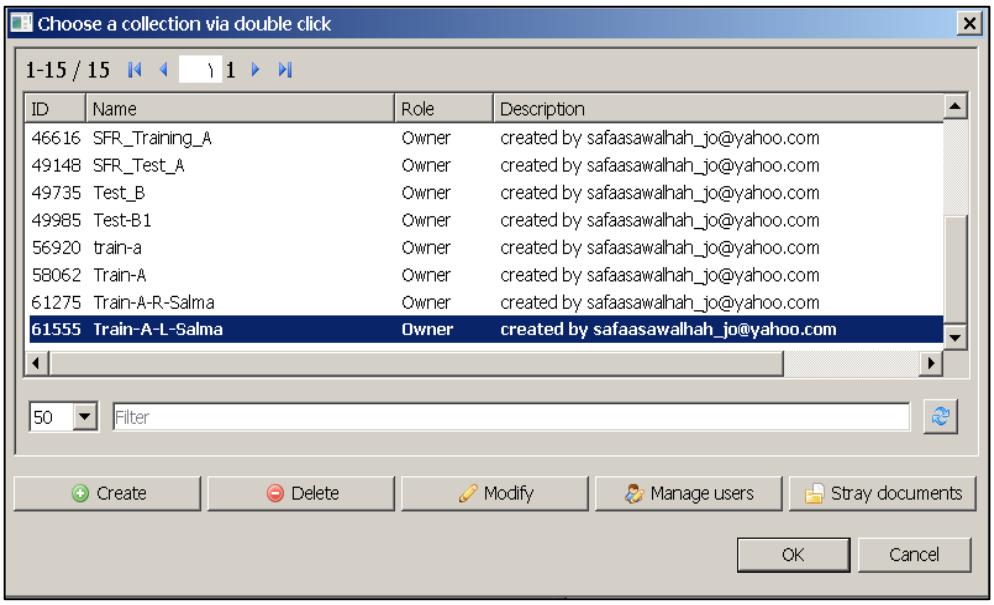

Figure 24: Access (Train-A-L-Salma) documents

10. All documents carried to Transkribus are private by default. It is possible to grant other users permission to view our documents if we wish. It is possible to add users to our collection by using the "User Manager" button on the "Server" tab, so only users with an account can only share the collection. Figure 25 shows the "User Manager" button.

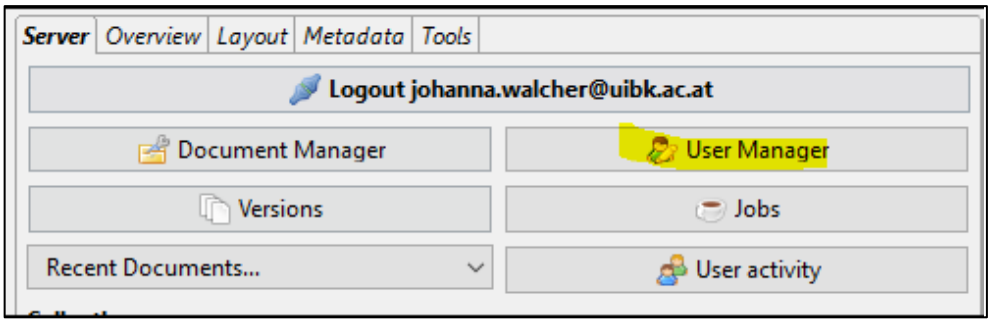

Figure 25: “User Manager” button

11. Documents segmentation into lines. Documents must be segmented into lines so that we can feed the HTR engine with training data, and this can be done automatically in Transkribus. The following is a review of the segmentation mechanism.

    11.1 On the Tools tab, make sure "Find Text Regions" is selected and press the "Run" button. Figure 26 and 27 shows how we can find text regions mechanism and how we can confirm this process.

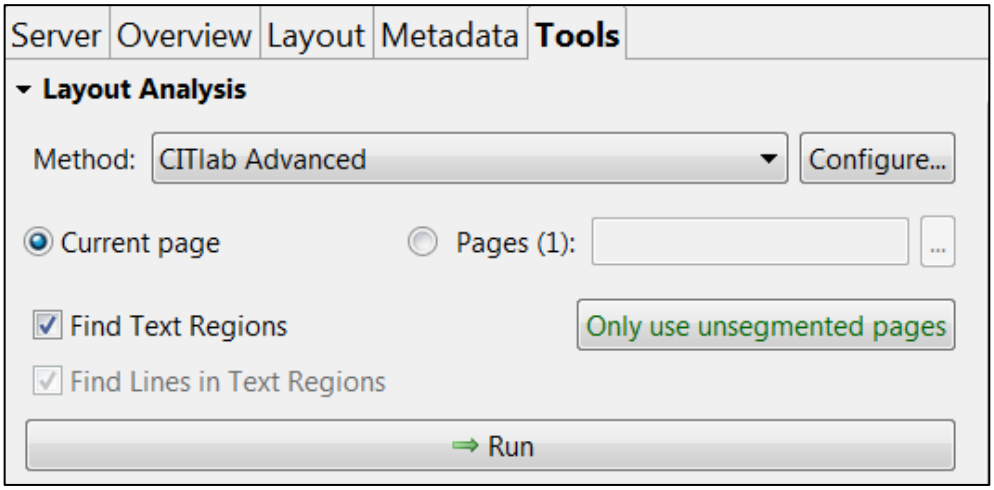

Figure 26: Find Text Regions Mechanism

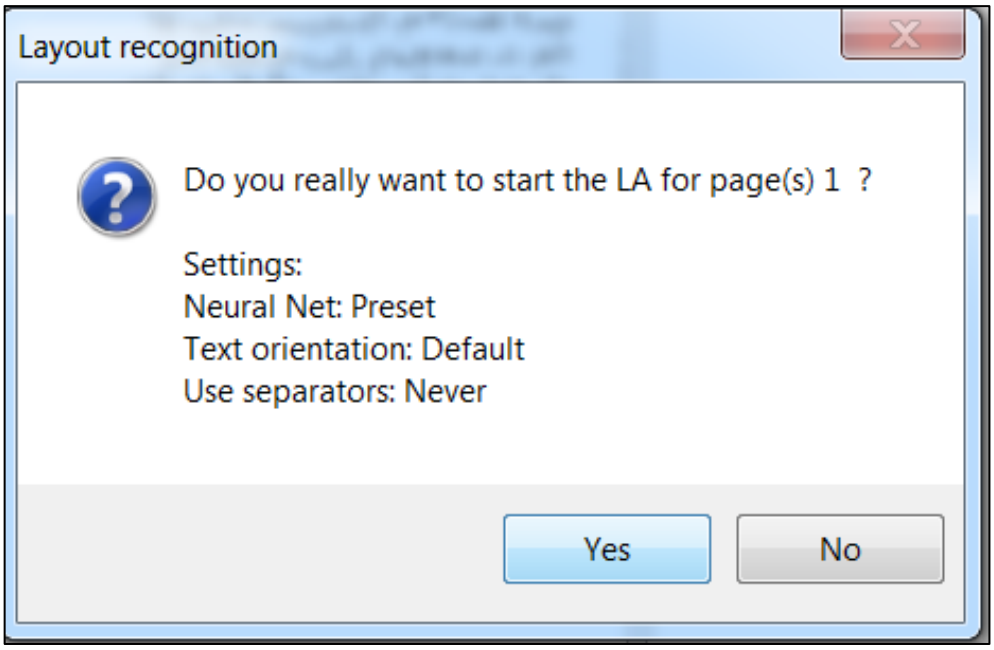

Figure 27: Confirm the find text regions mechanism

11.2 Figure 28 shows how to automatically detect text regions, baselines, and lines in the Transkribus.

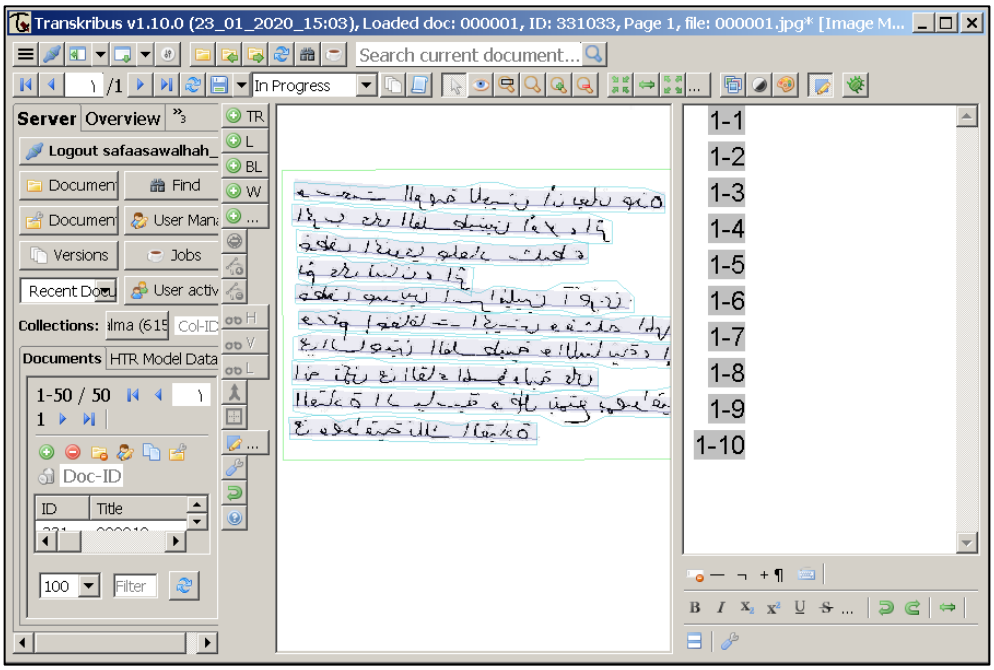

Figure 28: Automatically detect text regions, baselines, and lines

11.3 Figure 29 shows how we can automatically delete text area, baseline, and font detection. If there is a mistake in the automatic recognition by the Transkribus.

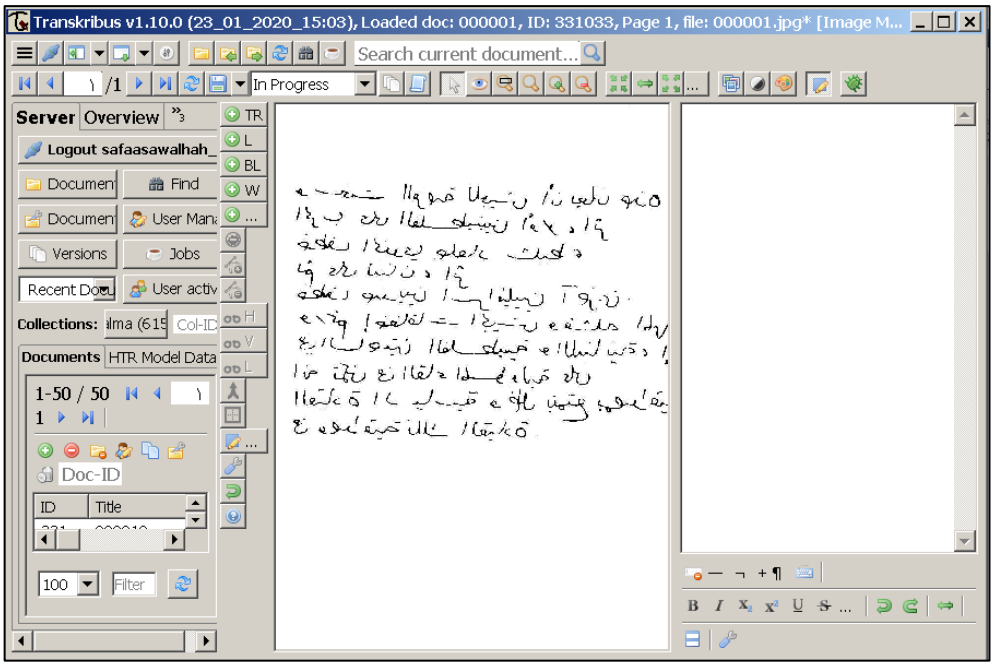

Figure 29: Delete automatically detects text regions, baselines, and lines

11.4 Figure 30 shows how we can add manual text regions.

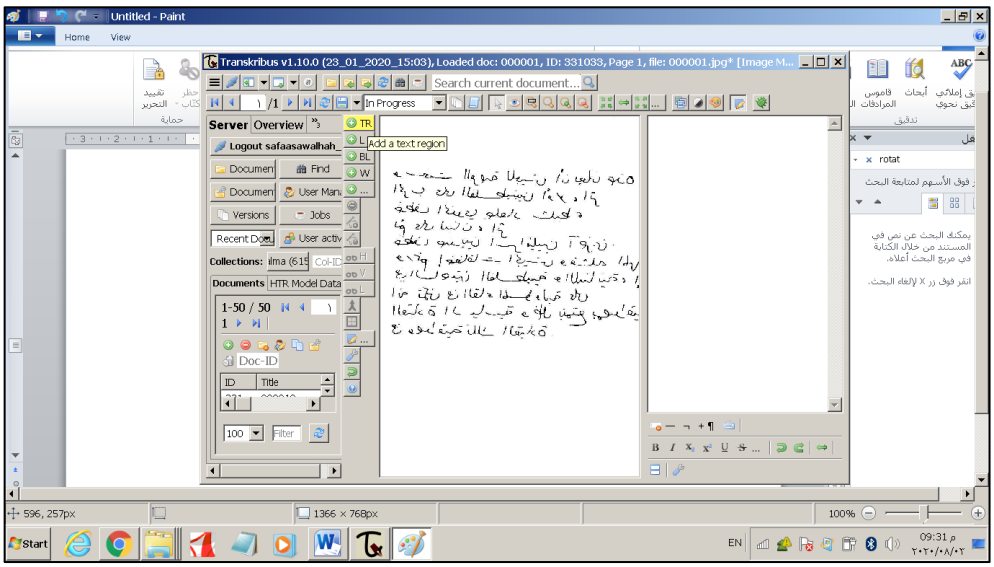

Figure 30: Add manual text regions

11.5 Figure 31 and 32 show how we can add baselines.

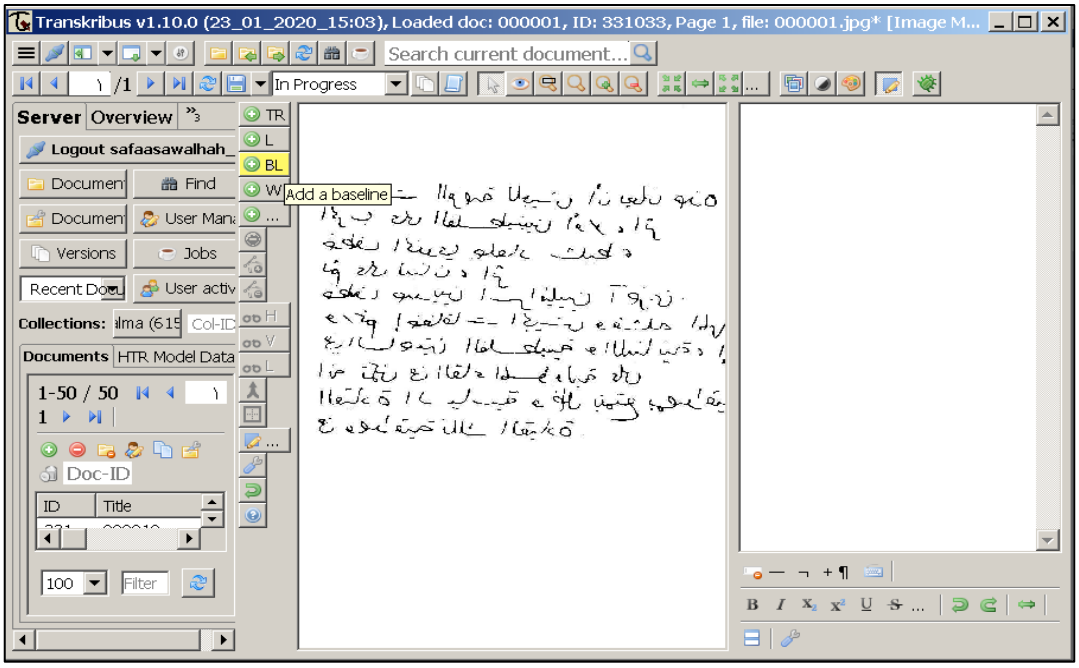

Figure 31: Add manual baselines (1)

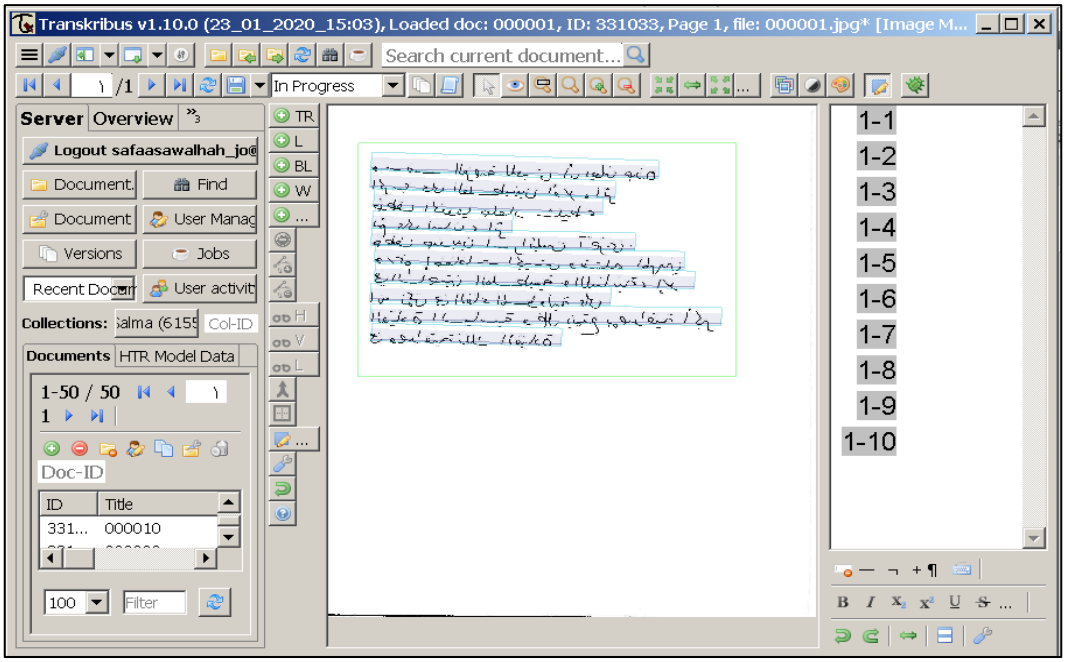

Figure 32: Add manual baselines (2)

12. Figure 33 shows the "View Profiles" button and chooses "Transcription".

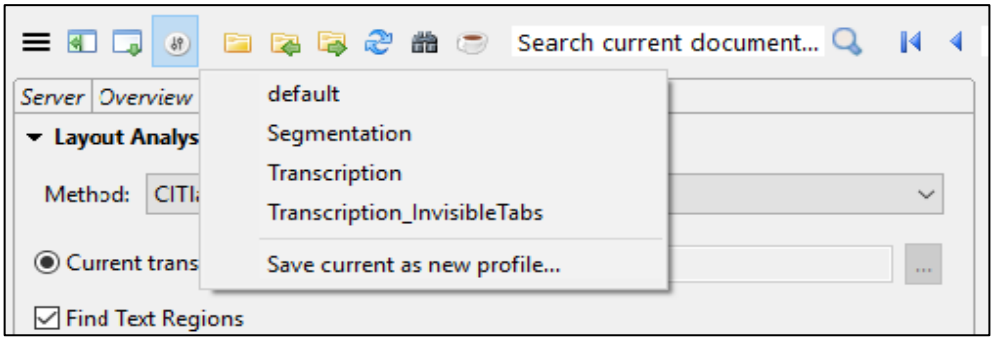

Figure 33: The View Profiles button

13. Documents transcription. After the baselines appear on the images, we start to write text into the text editor field. To each baseline, there is a similar line in the text editor. Figure 34 shows how to make transcript to text.

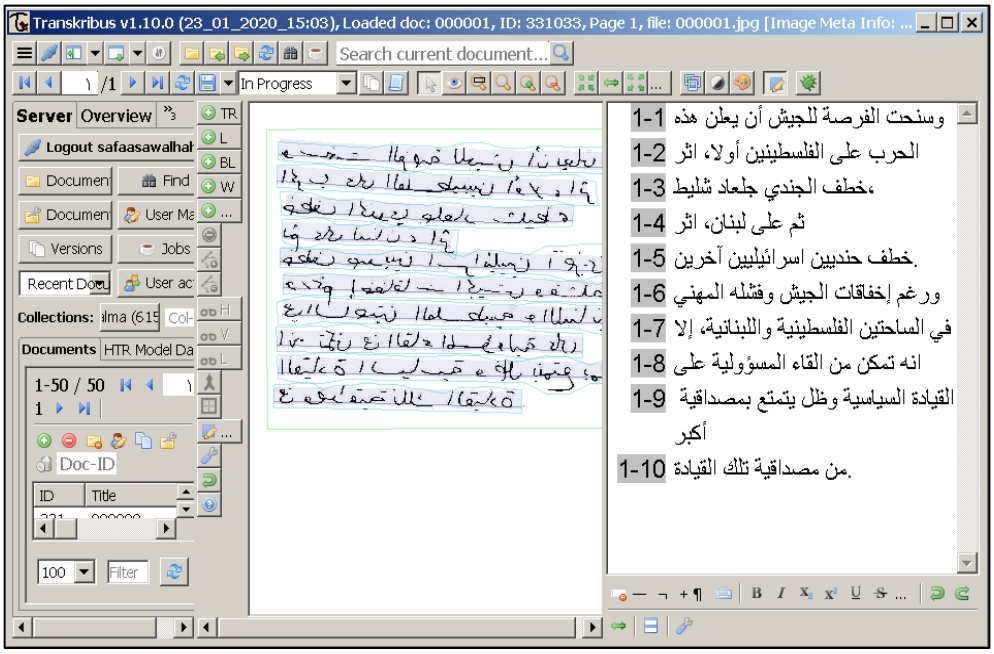

Figure 34: The transcriptions to text mechanism

14. Figure 35 shows how we can save and export our transcription from the Transkribus server.

Figure 35: The save and export buttons

4.2.2.2 FORMAT OF TRAIN_B DATASET

In order to make XML files in MADCAT dataset suitable for formatting XML files in Train_B dataset, we have developed Python code. This code uses a library (ElementTree). Xml.etree.ElementTree implements an efficient, simple API for analyzing and creating XML data. The ElementTree library contains functions for reading and manipulating XML files (and other similar structured files) (Python, 2019). The following code defines a function that we used to convert the XML files on the MADCAT dataset to be suitable with the format of the XML files used in the SFR model.

```
# Read data from madcat xml file
import xml.etree.ElementTree as ET
import os
with open('template.xml', 'r', encoding='utf-8') as f:
contents = f.read()
XML_path = './ml_1/'
CXML_path = './3/'
for XML_fn in os.listdir(XML_path):
    if XML_fn.endswith('.xml'):
        tree = ET.parse(XML_path + XML_fn)
        root = tree.getroot()
        a = ''
        for transcription in root.iter('transcription'):
            a += transcription.text
        for page in root.iter('page'):
            image_name = page.attrib.get('id')
            height = page.attrib.get('height')
            width = page.attrib.get('width')
        W = int(round(int(width)//2,0))
        H = int(round(int(height)//2,0))
        coords='0,0 ' + str(W-1)+',0 ' + str(W-1)+','+str(H-1) + ' 0,' +str(H-1)
        # Parsing xml file from READ dataset
        output_text = xml_template_contents.replace("string_image_name",
image_name+'.jpg')
        output_text = output_text.replace("string_H", str(H))
        output_text = output_text.replace("string_W", str(W))
        output_text = output_text.replace("string_coord", str(coords))
        output_text = output_text.replace("string_text", str(a))
        print(output_text)
        f = open(CXML_path + XML_fn, 'w+', encoding='utf-8')
        f.write(output_text)
        f.close()
```

CHAPTER V

Experiments and Results

Our proposed HWR for recognizing Arabic handwritten documents is directly depending on the SFR system. The SFR system is used to recognize historical handwritten documents in German. The science of recognizing handwritten documents is classified as one of the most difficult branches of HWR science.

SFR outperformed the winner of the ICDAR2017 handwriting competition, even when the submitted competition annotations were not used. SFR is a deep learning system that jointly learns to discover, segment and recognize text with most anonymous images or segmentation annotations.

5.1 Work Environment

Our proposed HWR system was implemented on a Lenovo workstation with Intel(R) processor Intel core-i5 4460 CPU 3.20 GHz; it has a 4GB DDR3 RAM, Ubuntu 18.04.2 LTS amd64, and 64 bit operating system. In order to speed up the training process, we have used GPU of a kind gtx-1060.The program was written using Python 2.7 language on the PyCharm IDE. PyCharm was developed by the Czech company JetBrains (Darryl K, 2010).

We have used PyCharm environment because it provides the ability to code analysis, graphic debugger, integrated unit testing, and integration with version control systems (VCSes) (Wikipedia a Page, 2020). And it supports network development with Django as well as Data Science with Anaconda (Darryl K, 2010).

5.2 Loss Evaluation

To assess the improvement of the program in the learning and training, it was used loss as a benchmark for evaluating the tool. Loss is a method used for assessing the quality of the specified algorithm for the data presented. If predictions deviate too much from actual results, the loss function coughs too many. By using the optimization function, the loss function learns to gradually reduce the error in prediction.

No single loss function is appropriate for everyone in machine learning algorithms. There are many factors involved in choosing a loss function for a specific problem such as the type of machine learning algorithm chosen, ease of calculating derivatives, and to some extent the percentage of outliers in the dataset.

There are two types of loss function according to the kind of learning: task classification losses and regression losses (Ravindra Parmar, 2018).

5.3 Start, Follow, Read Implementation

In order to implement the SFR model, we must perform a number of operations. In this project we did the pre-training process only. Figure 36 shows a summary of the complete training process for the SFR model. The following is an explanation of the steps that were taken during the pre- preprocessing and pre-training process:

1 We installed the dependencies are all available on the environment.yaml and can be downloaded as follows(Start follow read github Page, 2020) :

```
Conda env create -f environment.yaml
```

2 To activate the environment:

```
source activate sfr_env
```

3 We installed the warp-ctc library from the source (warp-ctc) is used in training (SeanNaren github Page, 2020).

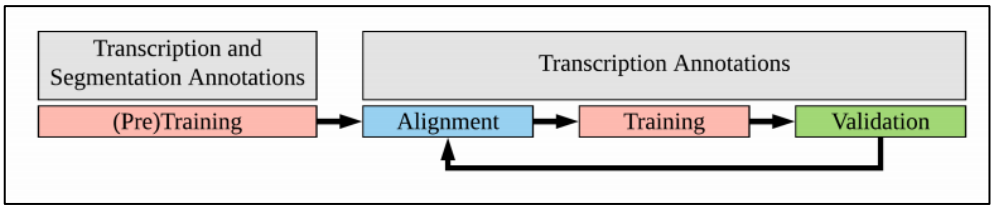

Figure 36: a) SFR pre-training on a small training set. b) The training process is carried out on a much larger training set that contains only annotations in three phases

4 Prepare Train-A. In this step we extract line start positions, line follower, and the normalized line images to give it to the HWR network so the HWR network can recognize it.

Figure 37 shows sample of the output result that we get when we applied preprocessing step on the test data from Train-A on the READ dataset. Figure 38 shows sample of the output result that we get when we applied preprocessing step on the test data from Train-A on the MADCAT dataset. The following code defines a function that we used to plot lines between points that represent the baseline on the Train-A data from READ and MADCAT dataset.

```
def draw_line_points(img, line_points,target_step_size):
    for point1, point2 in zip(line_points, line_points[1:]):
        X0=point1['x0']
        X1 = point1['x1']
        delat_x = X1 - X0
        target_step_size=X0/(X0+X1)
        x0= X0 + abs(delat_x) * target_step_size
        x0=int(x0)
        Y0 = point1['y0']
        Y1 = point1['y1']
        delat_y=Y1-Y0
        target_step_size = Y0 / (Y0 + Y1)
        y0=Y0 + abs(delat_y) * target_step_size
        y0=int(y0)
        start_point = (x0,y0)
        X0 = point2['x0']
        X1 = point2['x1']
        delat_x = X1 - X0
        target_step_size = X0 / (X0 + X1)
        x1 = X0 + abs(delat_x) * target_step_size
        x1=int(x1)
        Y0 = point2['y0']
        Y1 = point2['y1']
        delat_y = Y1- Y0
        target_step_size = Y0 / (Y0 + Y1)
        y1 = Y0 + abs(delat_y) * target_step_size
        y1=int(y1)
        end_point=(x1,y1)
        color = (0, 255, 0)
        thickness = 2
        img = cv2.line(img, start_point, end_point, color, thickness)
    return img
```

We can prepare Train-A data as follow:

```
python preprocessing/prep_train_a.py data/Train-A/page data/Train-A
data/train_a data/train_a_training_set.json data/train_a_validation_set.json
```

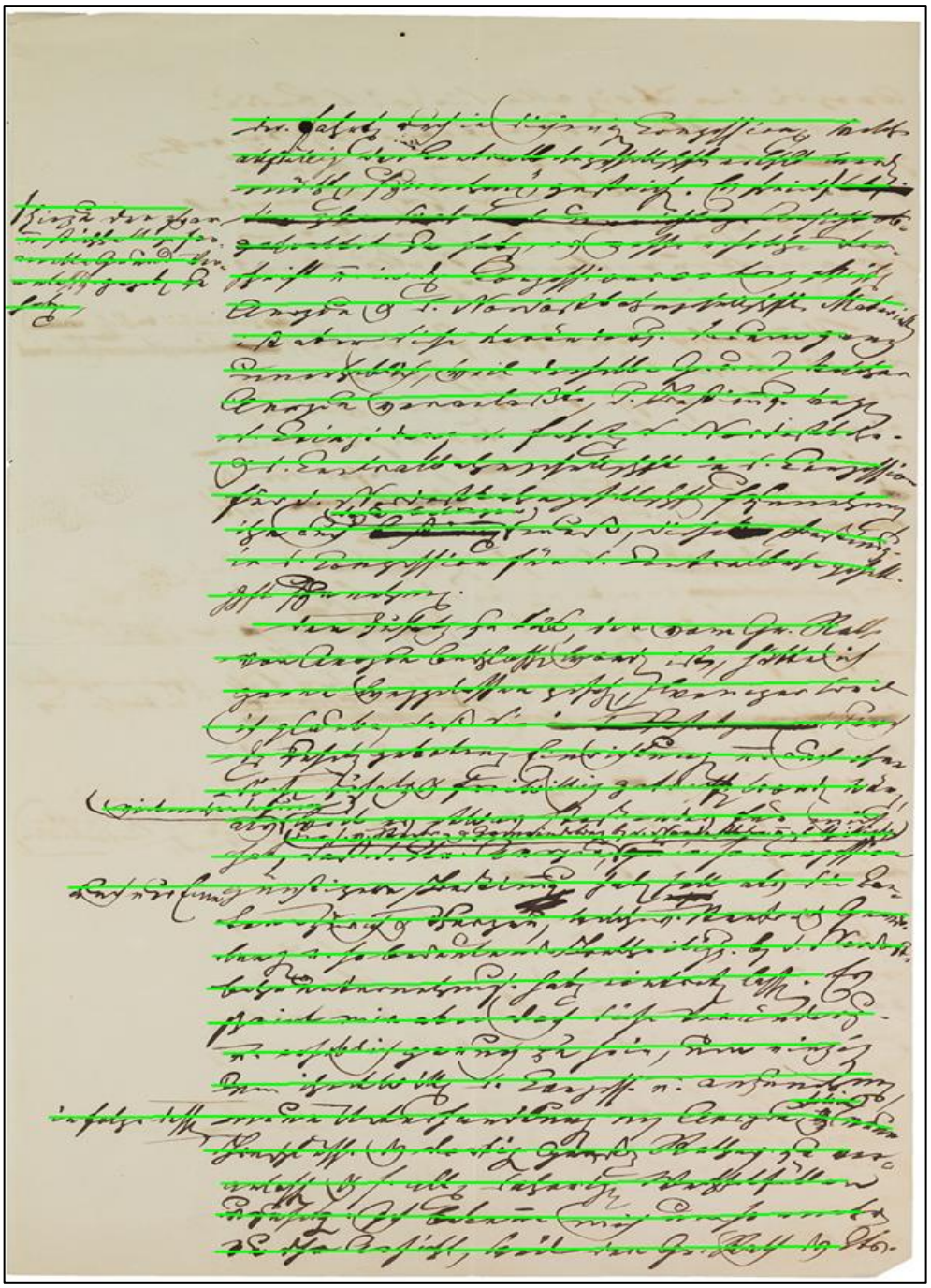

Figure 37: Sample of applying preprocessing step on the test data from the READ dataset

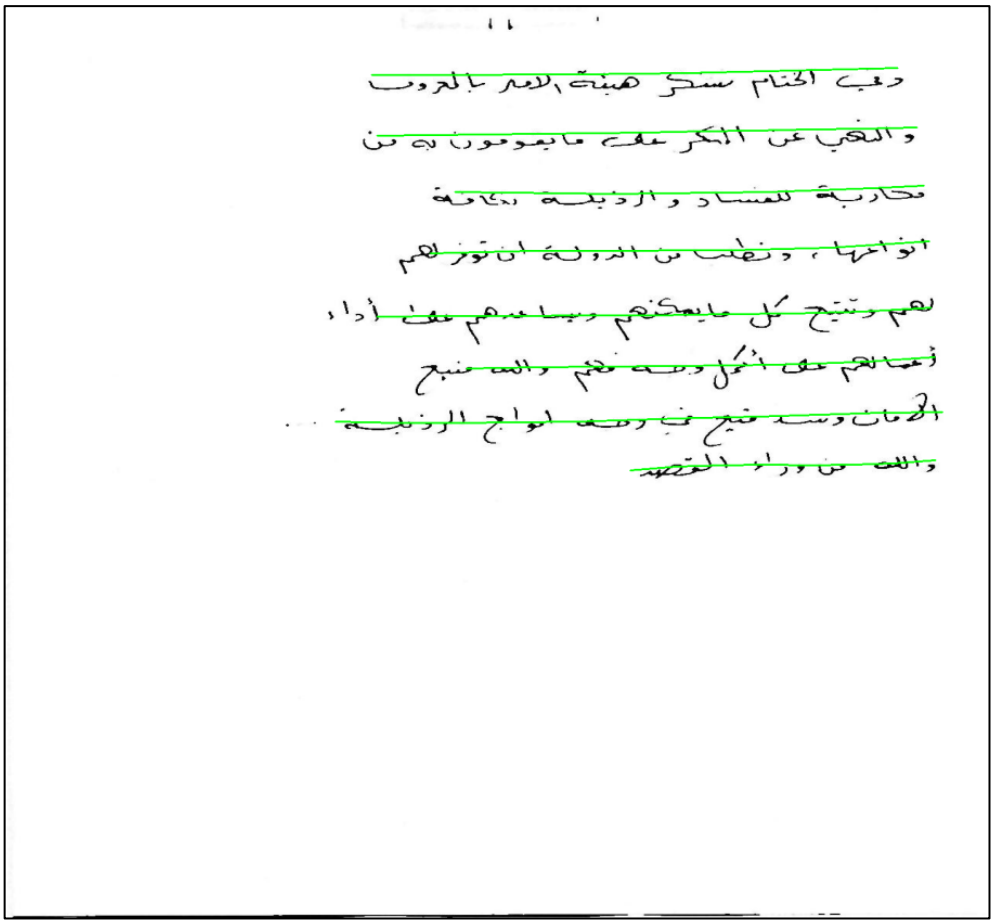

Figure 38: Sample of applying preprocessing step on the test data from the MADCAT dataset

5 Prepare Train-B, in this step, only the ground truth lines are extracted from XML. We can prepare Train-B as follow:

```
python preprocessing/prep_train_b.py data/Train-B data/Train-B
data/train_b data/train_b_training_set.json data/train_b_validation_set.json
```

6 Generate Character Settings, in this step, a group of characters will be created depend on the lines on Train-A and Train-B. There must be 140 unique characters in the MADCAT dataset. So that the network will output 141 characters to include the empty CTC. We can do this as follow:

```
python utils/character_set.py data/train_a_training_set.json
data/train_a_validation_set.json data/train_b_training_set.json
data/train_b_validation_set.json data/char_set.json
```

7 Pre-training, this step summarizes the results of the first stages of applying SFR model when it is applied to recognize the Arabic handwritten documents specifically the MADCAT dataset. And when it is applied on READ dataset. Training is performed in this phase using 32-pixel tall images and it is also good to align as per the author recommendation to increase the speed of training.

We used a small number of images just (50) with additional segmentation labels in the pre-training phase (i.e. Train-A dataset). We do many experiments on the Train-A dataset. We split this dataset into two sets: training and testing. The training set includes 45 images. The validation set is the same as the test set which include 5 images. We conducted several experiments using the Train-A dataset to find the best formations that would provide the best results.

8 Start of the Line (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. Figures 39 and 40 respectively show the results of applying SOL network on the READ and MADCAT datasets. Figures 41 and 42 respectively show sample of the results that we get when we apply SOL network on the test data from the READ and MADCAT datasets. The following code defines a function that we used to plot the circles that represent the truth points and the prediction points on the SOL network.

```
### Write images to file to visualization
sol.load_state_dict(torch.load(os.path.join(pretrain_config['snapshot_path'], 'sol.pt')))
for step_i, x in enumerate(test_dataloader):
        img = Variable(x['img'].type(dtype), requires_grad=False, volatile=True)
        sol_gt = Variable(x['sol_gt'].type(dtype), requires_grad=False, volatile=True)
        predictions = sol(img)
        predictions = transformation_utils.pt_xyrs_2_xyxy(predictions)
        org_img = img[0].data.cpu().numpy().transpose([2, 1, 0])
        org_img = ((org_img + 1) * 128).astype(np.uint8)
        org_img = org_img.copy()
        org_img = drawing.draw_sol_sol_gt_torch_prediction_data(sol_gt, org_img)
        org_img_1 = drawing.draw_sol_torch(predictions, org_img)
        cv2.imwrite("data/sol_val_2/{}.png".format(step_i), org_img_1)
for step_i, x in enumerate(train_dataloader):
                # for each image x, draw it, draw the labels (SOLs), predict, the draw the predicted SOLs
                # in another color
        img = Variable(x['img'].type(dtype), requires_grad=False)
        org_img = img[0].data.cpu().numpy().transpose([2, 1, 0])
        org_img = ((org_img + 1) * 128).astype(np.uint8)
        org_img = org_img.copy()
        sol_gt = None
        if x['sol_gt'] is not None:
                # This is needed because if sol_gt is None it means that there
                # no GT positions in the image. The alignment loss will handle,
                # it correctly as None
                sol_gt = Variable(x['sol_gt'].type(dtype), requires_grad=False)
                org_img = drawing.draw_sol_sol_gt_torch_prediction_data(sol_gt, org_img)
                predictions = sol(img)
                predictions = transformation_utils.pt_xyrs_2_xyxy(predictions)
                org_img_1 = drawing.draw_sol_torch(predictions, org_img)
                cv2.imwrite("data/sol_val_1/{}.png".format(step_i), org_img_1)

else:
#  continue
                print("None in step ", step_i)
```

We can do this as follow:

```
python sol_pretraining.py sample_config.yaml
```

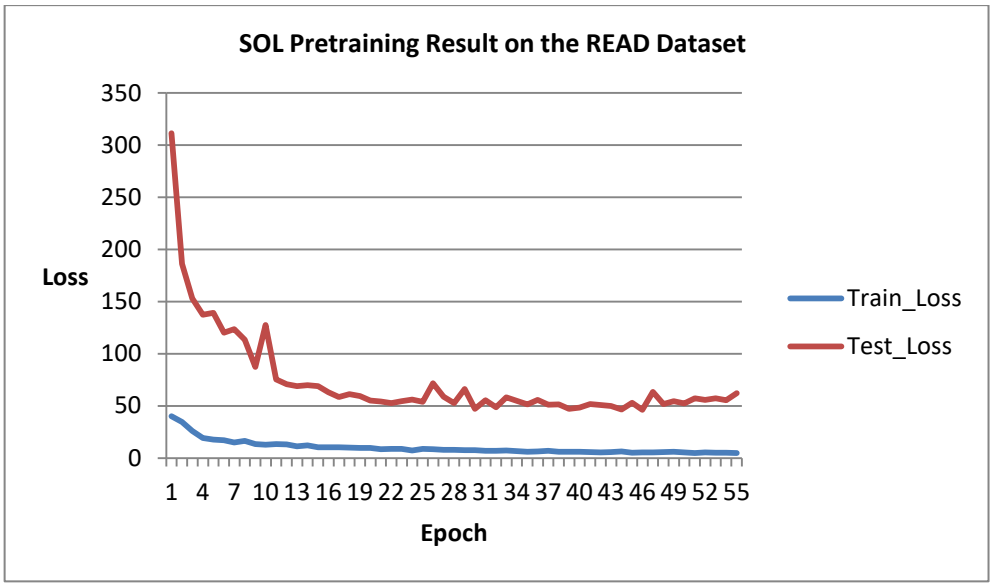

Figure 39: SOL pertaining result on the READ dataset

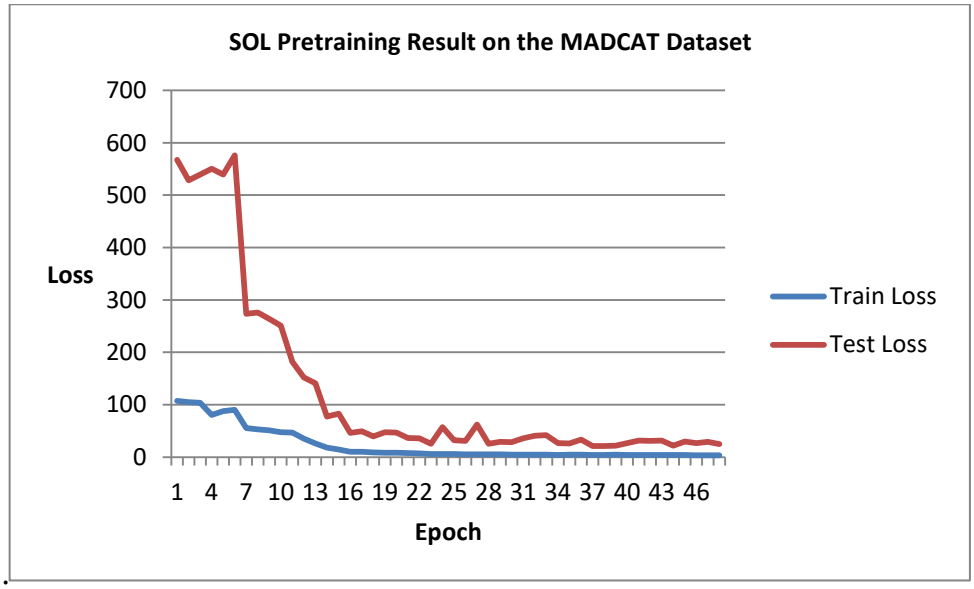

Figure 40: SOL pertaining result on the MADCAT dataset

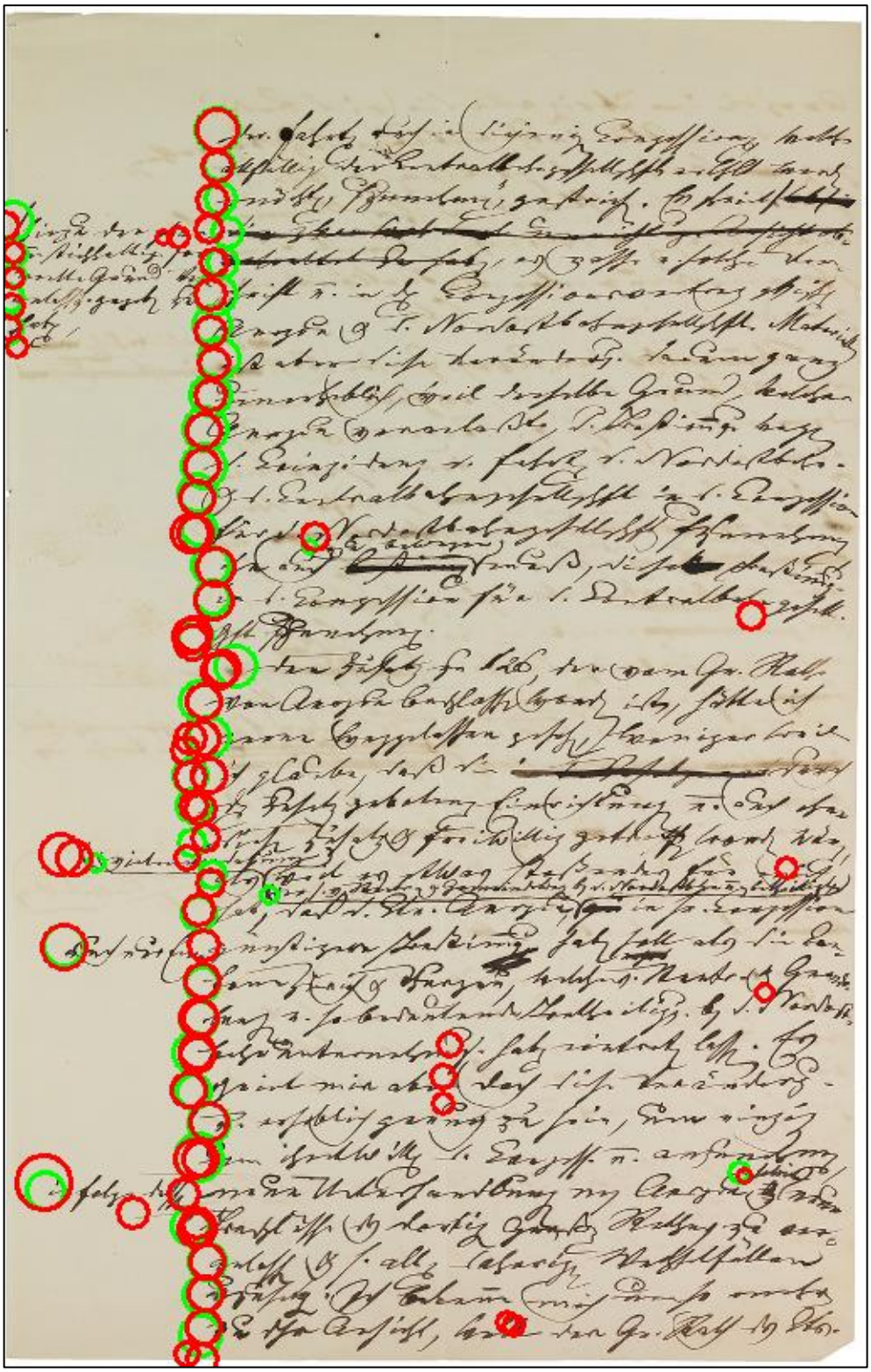

Figure 41: Sample of applying SOL pertaining result on the test data from the READ dataset

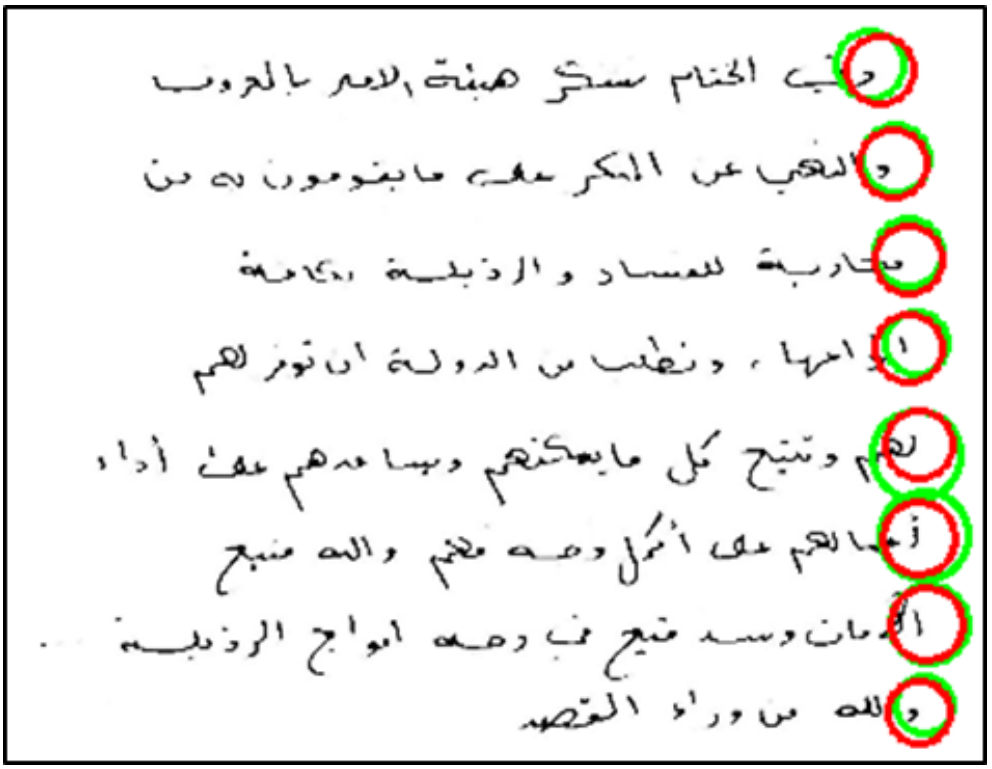

Figure 42: Sample of applying SOL pertaining result on the test data from the MADCAT dataset

9 Line Follower, the LF model begins at every position predicted by SOL, with progressive steps along the text line, following the curve, and producing a normal text image. Figures 43 and 44 respectively show the results of applying LF network on the READ and MADCAT dataset. Figures 45 and 46 respectively show samples of the results that we get when we apply LF network on the test data from the READ and MADCAT dataset. The following code defines a function that we used to visualize the output result from the LF network on the test data.

```
i=0
old_sum = -1
for x in test_dataloader:
        x = x[0]
        positions = [Variable(x_i.type(dtype), requires_grad=False, volatile=True)[None, ...] for x_i
in x['lf_xyrs']]
        xy_positions = [Variable(x_i.type(dtype), requires_grad=False, volatile=True)[None, ...] for
x_i in x['lf_xyxy']]
        img = Variable(x['img'].type(dtype), requires_grad=False, volatile=True)[None, ...]
        org_img = img[0].data.cpu().numpy().transpose([2, 1, 0])
        org_img = ((org_img + 1) * 128).astype(np.uint8)
        org_img = org_img.copy()
        if len(xy_positions) <= 1:
                continue
        grid_line, _, _, xy_output = line_follower(img, positions[:1], steps=len(positions),
skip_grid=True)
        new_sum = sum(sum(sum(org_img.astype(np.uint64))))
        if new_sum != old_sum:
                my_img = org_img
                old_sum = new_sum
                i = i + 1
        my_img = drawing.draw_lf_data_xy_positions(xy_positions, my_img)
        my_img = drawing.draw_lf_data_xy_output(xy_output, my_img)
        cv2.imwrite("data/lf_val/{}.png".format(str(i)), my_img)
```

```
def draw_lf_data_xy_output(xy_output,image):
        for point1,point2 in zip(xy_output, xy_output[1:]):
                pt0 = tuple(point1.data.cpu().numpy().astype(np.int64).tolist())
                x = pt0[0]
                x=x[0]
                X0,X1=x[0],x[1]
                x_avg = int((X0 + X1) / 2
                y = pt0[0]
                y = y[1]
                Y0, Y1 = y[0], y[1]
                y_avg = int((Y0 + Y1) / 2)
                start_point = (x_avg,y_avg)

                pt1 = tuple(point2.data.cpu().numpy().astype(np.int64).tolist())
                x = pt1[0]
                x = x[0]
                X0, X1 = x[0], x[1]
                x1_avg = int((X0 + X1) / 2)
                y = pt1[0]
                y = y[1]
                Y0, Y1 = y[0], y[1]
                y1_avg = int((Y0 + Y1) / 2)
                end_point = (x1_avg, y1_avg)
                # https://www.geeksforgeeks.org/python-opencv-cv2-line-method/
                # Green color in BGR
                color = (0, 0, 255)
                # Line thickness of 1 px
                thickness = 2
                # Using cv2.line() method
                # Draw a  green lines with thickness of 2 px
                image = cv2.line(image, start_point, end_point, color, thickness)
        return image
```

```
def draw_lf_data_xy_positions(xy_positions, image):
    for point1,point2 in zip(xy_positions, xy_positions[1:]):
        pt0 = tuple(point1.data.cpu().numpy().astype(np.int64).tolist())
        X0,X1=x[0],x[1]
        x_avg = int((X0 + X1) / 2
        y = pt0[0]
        y = y[1]
        Y0, Y1 = y[0], y[1]
        y_avg = int((Y0 + Y1) / 2)
        start_point = (x_avg,y_avg)

        pt1 = tuple(point2.data.cpu().numpy().astype(np.int64).tolist())
        x = pt1[0]
        x = x[0]
        X0, X1 = x[0], x[1]
        x1_avg = int((X0 + X1) / 2)
        y = pt1[0]
        y = y[1]
        Y0, Y1 = y[0], y[1]
        y1_avg = int((Y0 + Y1) / 2)
        end_point = (x1_avg, y1_avg)

        # https://www.geeksforgeeks.org/python-opencv-cv2-line-method/
        # Green color in BGR
        color = (0, 255, 0)
       # Line thickness of 1 px
        thickness = 4
        # Using cv2.line() method
        # Draw a  green lines with thickness of 2 px
        image = cv2.line(image, start_point, end_point, color, thickness)
  return image
```

We can do this as follow:

```
python lf_pretraining.py sample_config.yaml
```

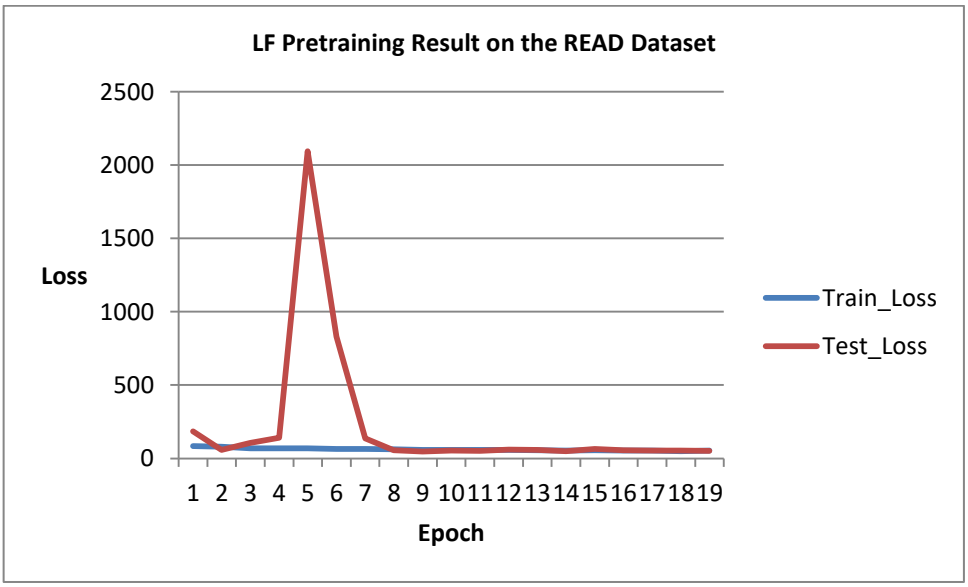

Figure 43: LF pertaining result on the READ dataset

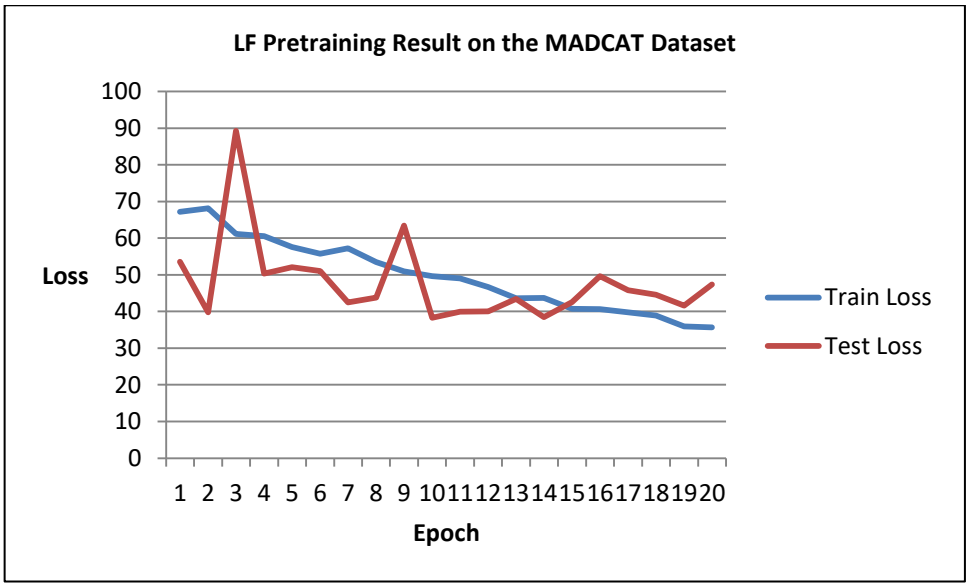

Figure 44: LF pertaining result on the MADCAT dataset

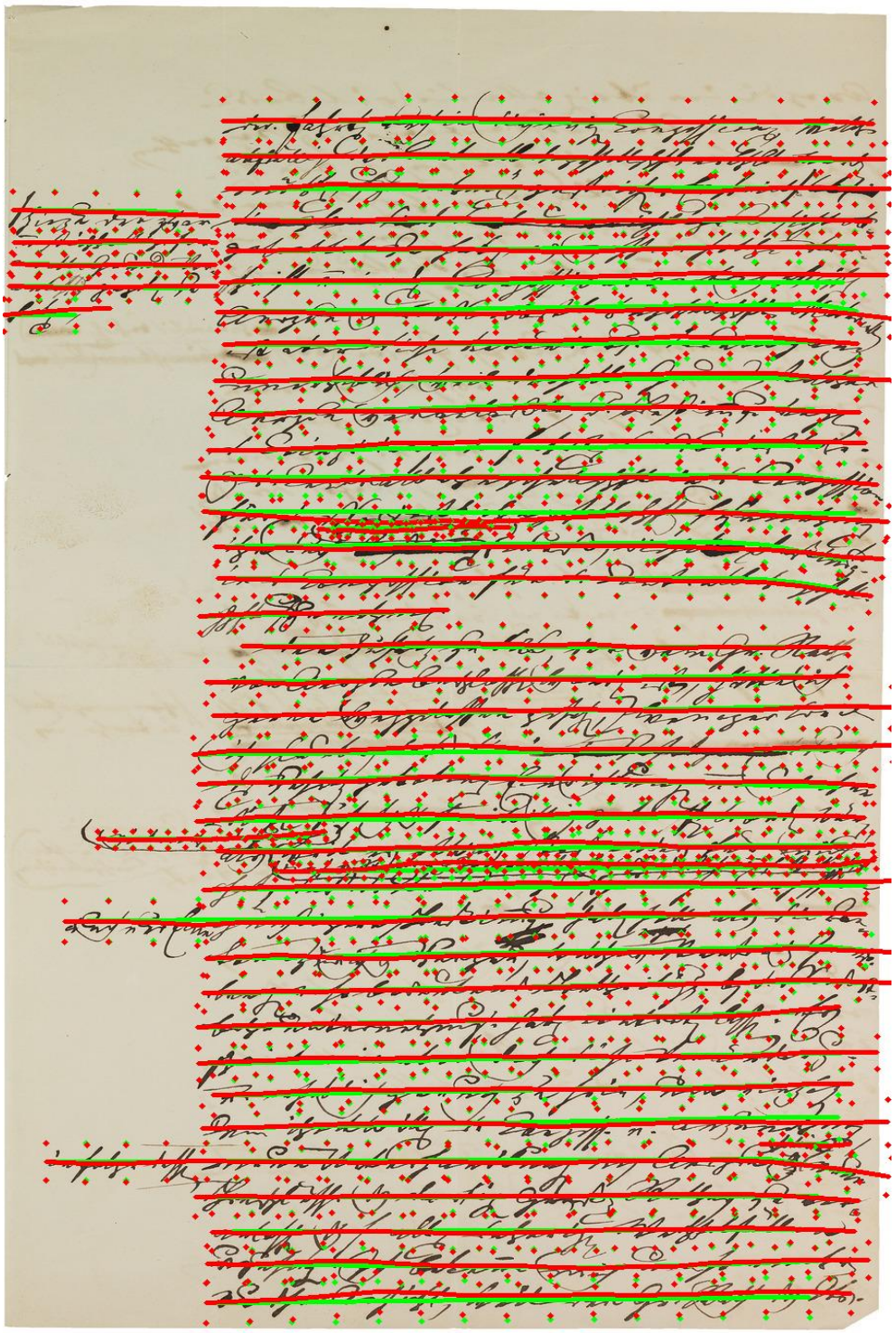

Figure 45: Sample of applying LF pertaining result on the test data from the MADCAT dataset

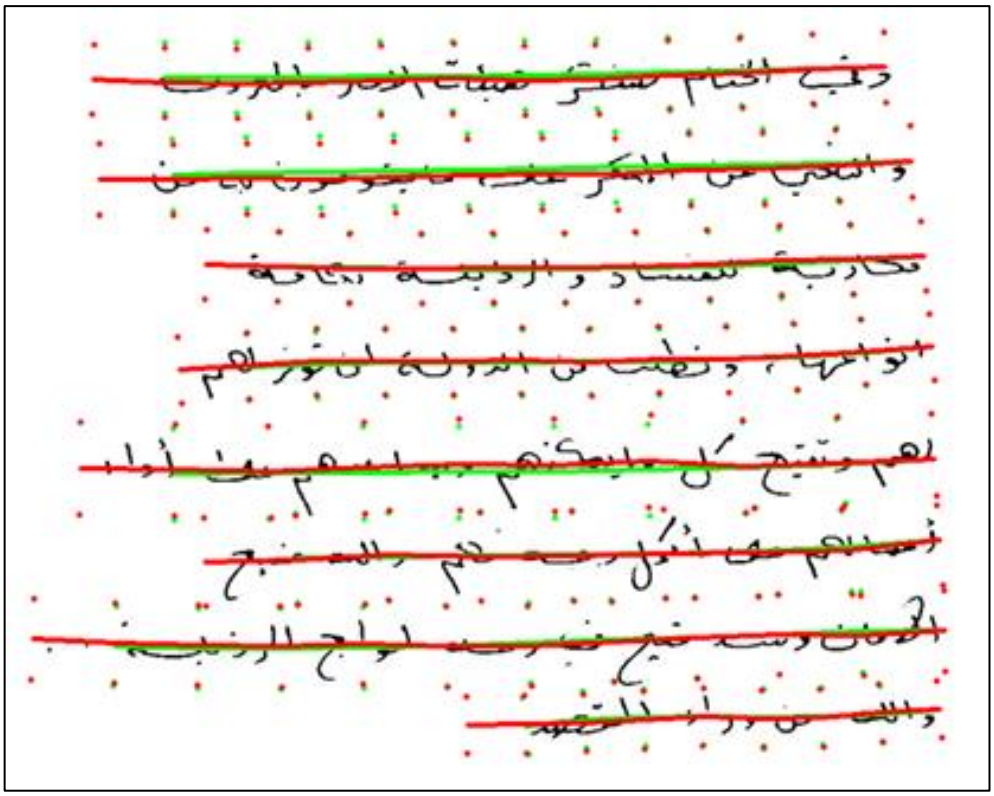

Figure 46: Sample of applying LF pertaining result on the test data from the MADCAT dataset

10 Handwriting Recognition, after the LF produces the image normalization line, they are fed to the CNN-LSTM network, to produce the transcription. CNN-LSTM is part of the HWR network learning high-level features that collapsed vertically to create a 1D horizontal sequence fed to the BLSTM model. Context learned features spread forward and backward on the BLSTM. The sequence before the character classifier is applied to each step in the output.

The character prediction output sequence is much longer than GT, because it includes a blank character for use in CTC decoding step. Decoding is carried out by the first collapsing non-empty of repeating characters, and after that blanks are removed.

There are three metrics that are used in the literature to calculate loss on HWR system: character error rate (CER), word error rate (WER), and

BLUE. CER is a percentage of character with wrong prediction. WER is a percentage of word that has at least one character of wrong prediction. We calculate CER for every epoch on the output sequences of the training and testing data on the HWR network. Figures 47 and 48 respectively show the results of applying HWR network on the READ and MADCAT dataset. The following code shows a function that we used to visualize the output result from the HWR network on the test data.

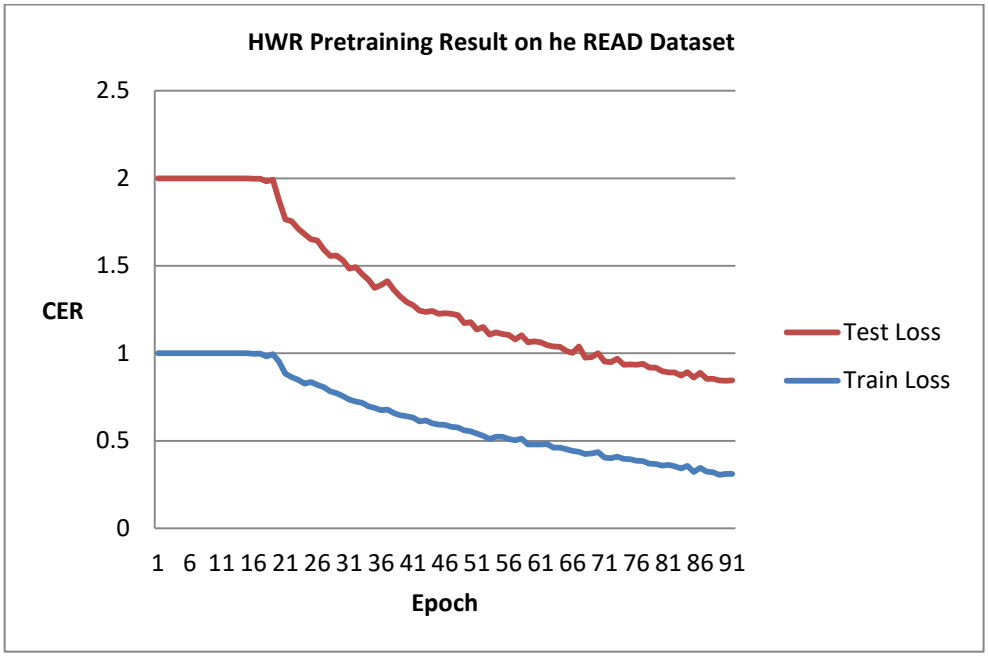

Figure 47: HWR pertaining result on the READ dataset

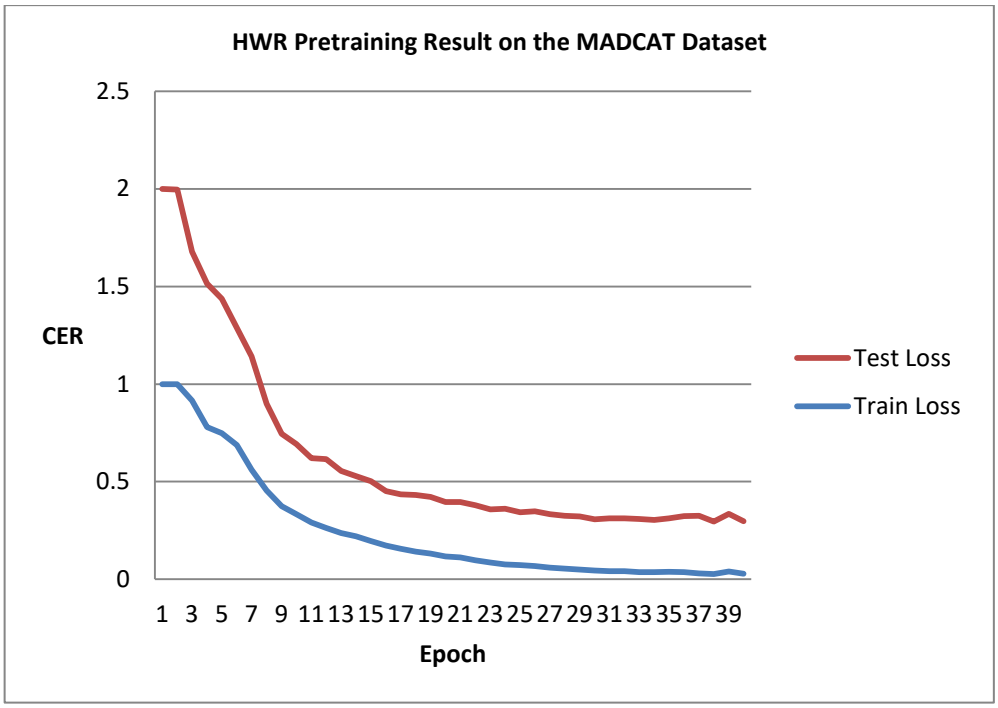

Figure 48: HWR pertaining result on the MADCAT dataset

```
hw.load_state_dict(torch.load(os.path.join(pretrain_config['snapshot_path'], 'hw.pt')))
for x in test_dataloader:
line_imgs = Variable(x['line_imgs'].type(dtype), requires_grad=False, volatile=True)
labels = Variable(x['labels'], requires_grad=False, volatile=True)
label_lengths = Variable(x['label_lengths'], requires_grad=False, volatile=True)
preds = hw(line_imgs).cpu()
output_batch = preds.permute(1, 0, 2)
out = output_batch.data.cpu().numpy()
f = open(str(steps) + 'gt', 'w+', buffering=-1)
a = open(str(steps) + 'pred', 'w+', buffering=-1)
for i, gt_line in enumerate(x['gt']):
logits = out[i, ...]
pred, raw_pred = string_utils.naive_decode(logits)
pred_str = string_utils.label2str_single(pred, idx_to_char, False)
# cer = error_rates.cer(gt_line, pred_str)
# sum_loss += cer
f.write(gt_line.encode('utf-8'))
f.write('\n')
a.write(pred_str.encode('utf-8'))
a.write('\n')
steps=steps+1
f.close()
a.close()
```

We can do this as follow:

```
python hw_pretraining.py sample_config.yaml
```

5.4 Training and Testing Time

In this project, we are interested in training time because in machine learning projects the training time is very long. It is directly proportional to the size of the dataset. The more data processed, the more training time. In machine learning projects, it is important to be able to work on large datasets in order to enhance accuracy and reduce the loss which used as a basic criterion in our project. Also, testing time is important in machine learning projects and one of the most prominent criteria used to diagnose how effective and efficient a project is because we need to know how much time the project will take for evaluation.

As shown in Figures 49 and 50, we measured the training and testing times for the three networks on the READ and MADCAT datasets. The three networks achieved the highest value of training times on the READ dataset. This is because the images in READ are deteriorating and unclear images of historical documents. Also, we noticed the same with the test time of the LF and HWR networks, as it was higher on the READ dataset.

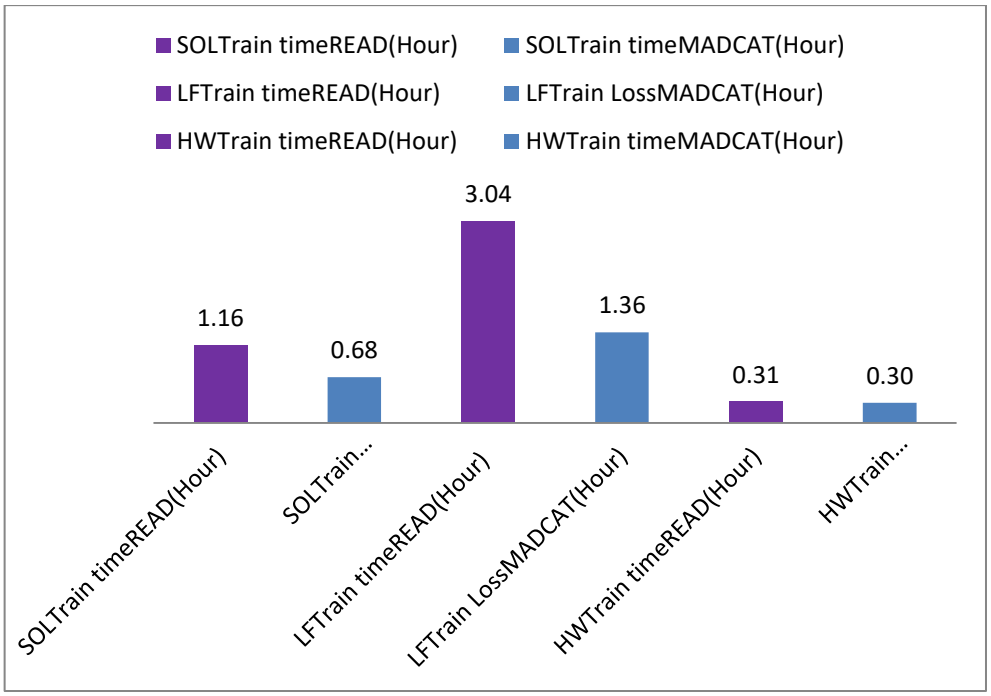

Figure 49: Comparison of total train time between the MADAT and the READ datasets

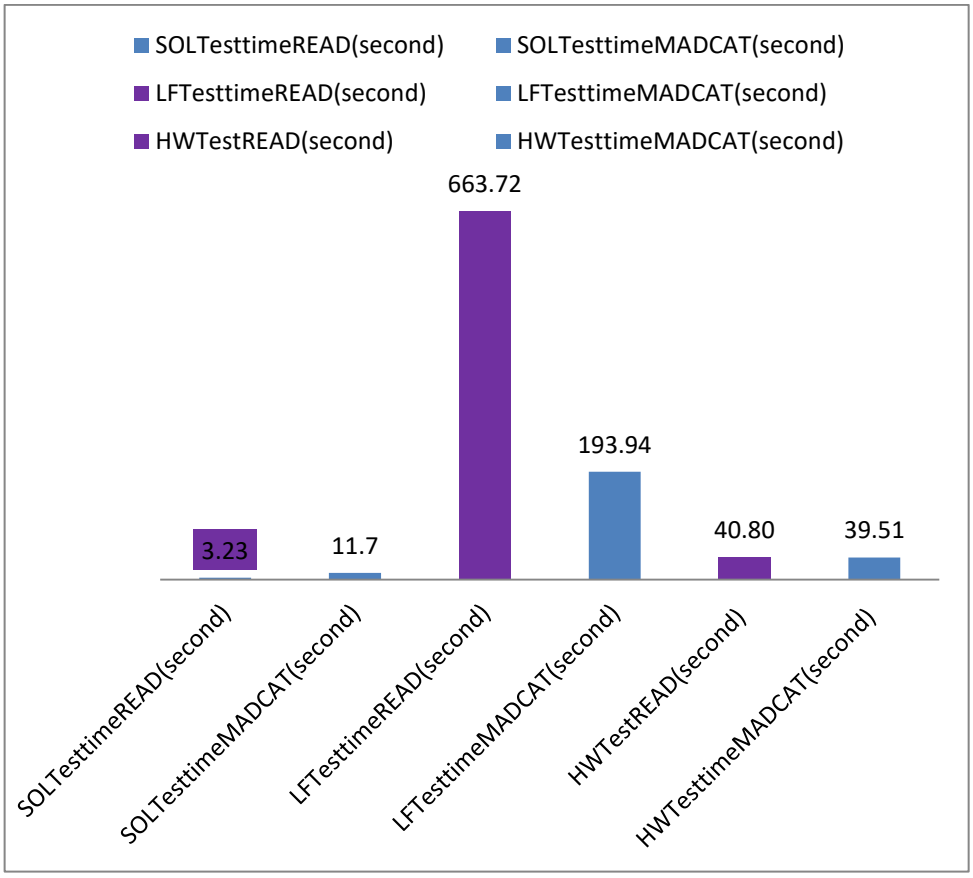

Figure 50: Comparison of the total test time between the MADCAT dataset and the READ dataset

5.5 Experiments Results

In this section we will provide an analysis of the results of the three networks discussed in section 5.5. By providing evaluation to the SFR system, when it applied into two datasets. The first one is the READ dataset which is handwritten dataset in the German language which back to the 19th century. The second one is the MADCAT dataset for the Arabic language. READ contains two sets of training data, the first set contains 50 fully annotated images with line-level segmentations and transcriptions. The second contains 10,000 images containing only transcriptions (containing Line breaks).

In this thesis, we will discuss only the results of applying the SFR on the 50 images, i.e. (Train-A) on the READ dataset or on the MADCAT dataset.

As shown in table 2, we have noticed that the loss rate on the MADCAT dataset in all networks is less than on the READ dataset. This is predictable because the images in the READ dataset are less clear than in the MADCAT dataset. As they are historical images deteriorating back to the nineteenth century.

Table 2: Results of applying SFR on the READ dataset and on the MADCAT dataset

Loss	READ dataset	MADCAT dataset
SOL_pretraining	47.91	21.95
LF_pretraining	39.11	47.06
HWR_pretraining	0.53	0.26

5.6 Post-processing Contribution

We use post-processing techniques to minimize loss rates on the MADCAT dataset. Table 3 illustrates the effect of post- processing techniques on minimize loss. We note that the removal of noise has minimized the loss. As shown in table 3 it is not a great improvement but it contributes on reduce loss rates and reducing the CER on HWR network.

Table 3: The effect of the post-processing techniques on the MADCAT dataset

Loss	READ dataset	MADCAT dataset without post-processing
SOL_pretraining	47.91	22.36
LF_pretraining	39.11	52.68
HWR_pretraining	0.53	0.26

5.7 DISCUSSION

We found that applying SFR on the MADCAT dataset achieved better results than applying it on the READ dataset, because the READ dataset is one of the largest on the HWR systems. It consists of 206,161 handwriting lines and 1769,195 words. It is one of the most difficult HWR bands, i.e. historical documents (Wigington, et al., 2018).

Figures 51and 52 show train and test loss values of the three networks on the READ and on the MADCAT dataset. We note that the train loss and the test loss on the MADCAT dataset are less than on the READ dataset on the SOL network. The train loss and the test loss on the MADCAT dataset are greater than the train loss and the test loss on the READ dataset on the LF network. In the HWR network, CER on the train data and on the test data on the MADCAT dataset are less than on the READ dataset.

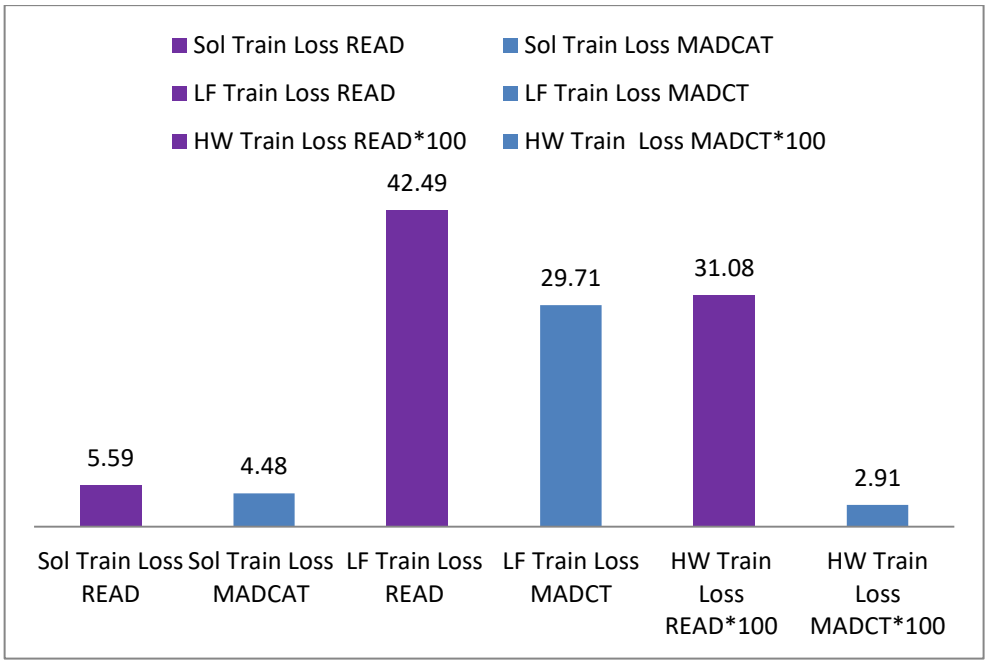

Figure 51: Comparison between MADCAT and READ dataset on the train loss

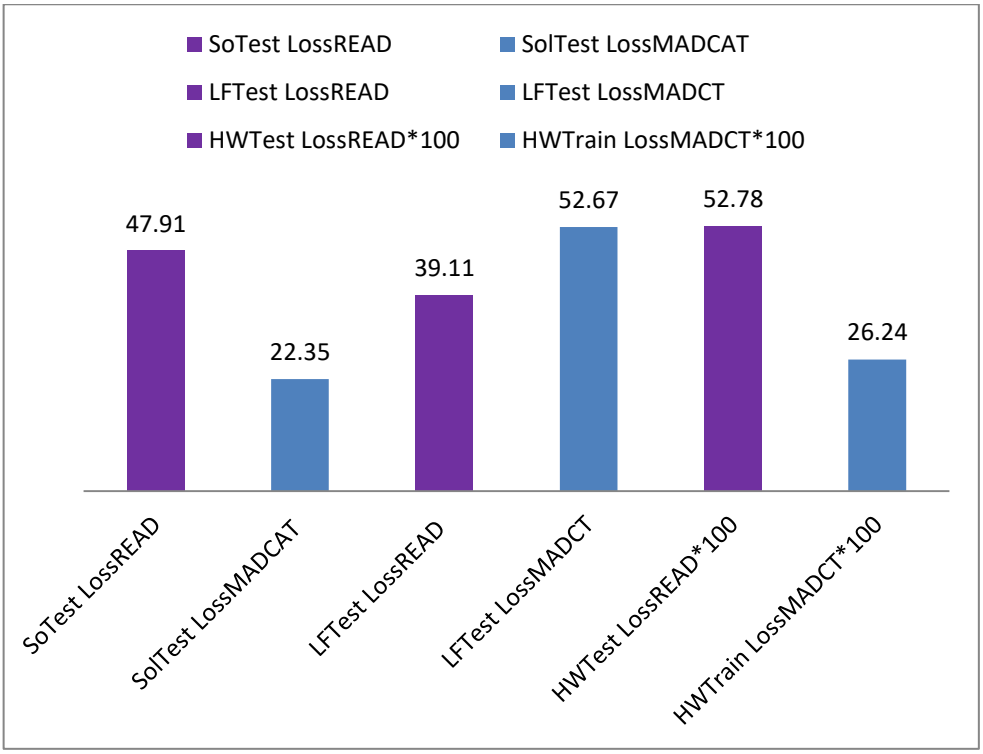

Figure 52: Comparison between MADCAT and READ dataset on the test loss

CHAPTER VI

Conclusions and Future Work

Pattern recognition is a sub field in machine learning and artificial intelligence science. Pattern recognition is regularly concerned with data and automatic pattern detection (Stepney, et al., 2006). Pattern recognition includes many different fields such as image recognition, fingerprint and face recognition, numbers and letters recognition, etc. (El-Sawy, et al., 2016). One of the most common pattern recognition problems is feature extraction. But now, with the use of machine learning methods, it has become an easy process as the machine bears the burden to solve this problem (Belabiod, et al., 2018).

One of the most important pattern recognition sciences is handwriting recognition (HWR) and it is known as the computer ability to receive the handwriting and translate it from many sources such as paper documents, photographs, touch screens and etc. (Lorigo, et al., 2006).

One of the main problems studied in the documentation community is HWR (Dutta, et al., 2018). And machine learning methods have been used in their solution as they are classified as supervised or semi-supervised learning method (classification approach), online or offline learning, and instance or model based learning (Bluche, 2016).

In this thesis, we presented a study on the possibility of using and applying machine learning methods used in recognizing handwritten documents in Latin languages, specifically the Start, Follow, Read system (SFR) to recognize the handwritten documents in the Arabic language.

We chose the SFR system to recognize handwritten Arabic documents because it showed good performance in the difficult handwritten dataset from the German history dataset back to the nineteenth century. As it outperformed the winner of the ICDAR2017 competition for handwriting recognition. It also achieved good results when it applied to the MADCAT handwritten Arabic language dataset.

The SFR is classified as end-to-end for recognizing handwritten documents from the front to the bottom of the page. SFR is used for offline handwriting recognition (HWR). SFR provided a deep learning model that together learns to discover text, segmentation, and appreciation mostly using images without disclosure or annotations. SFR consists of 3 sub-models: SOL 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. HWR predicts the normal line image will be transcript.

SFR achieved a number of contributions and accomplishments such as, it has improved on the previous SOL network. SFR introduced a new LF network which learns to segment and normalize handwriting lines for input to a HWR network.

SFR achieve loss rate on SOL network reaches to (47.9) in READ dataset, while it reaches to (21.9) on MADCAT dataset. the loss rate on LF network, reaches to (39.1) on READ dataset, while it reaches to (47.0) on MADCAT dataset. CER on HW network has reached to (0.52) on READ dataset. But on MADCAT dataset it reaches to (.25).

After we conducted the initial training of SFR (i.e. pretraining step), the training framework is now able to enter the final step of joint training for networks, using documents that contain only the line transcription level. This is important in real life

because when human commentators copy documents, they often do not comment on it any spatial information or segmentation.

SFR can be further improved and improved the segmentation process. This is done by conducting the end-of-line prediction (EOL) process, in addition to predicting the start of line. It is also possible to improve the accuracy and reduce the loss of the system by increasing network architectures on a large scale, therefore the system performance can increase with improved network architectures such as residual network.

REFERENCES

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

Abandah, G. A., & Jamour, F. T. (2010). Recognizing handwritten Arabic script through efficient skeleton-based grapheme segmentation algorithm. Intelligent Systems Design and Applications.

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

Abandah, G. A., Jamour, F. T., & Qaralleh, E. A. (2014). Recognizing handwritten Arabic words using grapheme segmentation and recurrent neural networks. International Journal on Document Analysis and Recognition (IJDAR).

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

Abandah, G. A., & Al-Hourani, A. S. (2018). Challenges and Preprocessing Recommendations for MADCAT Dataset of Handwritten Arabic Documents. In 2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), IEEE.

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

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

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

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>

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

Bluche, T., Louradour, J., & 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, IEEE.

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

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

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

Cwig, start_follow_read Page. Last accessed March 28th. Available from **https://github.com/cwig/start_follow_read.**

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

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.**

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

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

Ernst Haagsman, (2019), Collaboration with Anaconda, Inc., from **https://blog.jetbrains.com/pycharm/2019/04/collaboration-with-anaconda-inc/.**

Geeks3D, First Steps with PIL: Python Imaging Library Page. Last accessed March 8th 2020. 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 March 8th 2020. 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., Fern´andez, S., Gomez, F., Schmidhuber, J. (2006).Connectionist temporal

classification: Labelling unsegmented sequence data with recurrent neural networks. In: 23rd International Conference on Machine Learning. pp. 369–376. ACM

ICDAR2017 Competition on Handwritten Text Recognition on the READ Dataset (ICDAR2017 HTR) Page, 2017. Last accessed March 28th 2020. Available from **https://scriptnet.iit.demokritos.gr/competitions/8/.**

IFN/ENIT-database Page. Last accessed March 30th 2020. Available from **http://www.ifnenit.com/.**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.

Kaggle, vgg11 Page. Last accessed March 31th 2020. Available from **https://www.kaggle.com/pytorch/vgg11.**

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

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

Lorigo, L. M., & Govindaraju, V. (2006). Offline Arabic handwriting recognition: a survey. IEEE transactions on pattern analysis and machine intelligence.

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

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

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

Moysset, B., Kermorvant, C., & 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. 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. IEEE.

Python, xml.etree.ElementTree — The ElementTree XML API Page. Last accessed September 6th 2019. Available from **https://docs.python.org/3/library/xml.etree.elementtree.html#module xml.etree.ElementTree.**

Qaralleh, E., Abandah, G., & 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, 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.

SeanNaren, warp-ctc Page. Last accessed March 30th 2020. Available from **https://github.com/SeanNaren/warp-ctc.**

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., & Razzazi, F. (2003). An efficient front-end system for isolated Persian/Arabic character recognition of handwritten data-entry forms. Int. J. Comput. Intell. Appl.

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., ... & Milner, R. (2006). Journeys in non-classical computation II: initial journeys and waypoints. The International Journal of Parallel, Emergent and Distributed Systems.

Svozil, D., Kvasnicka, V., and Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. Chemometrics and intelligent laboratory systems, 39(1), pp. 43-6.

Transkribus a, Download_and_Installation Page. Last accessed August 20th 2019. Available from **https://.eu/wiki/index.php/Download_and_Installation.**

Transkribus b**,** How_to_Guides Page. Last accessed August 25th 2019. Available from **https://transkribus.eu/wiki/index.php/How_to_Guides.**

Transkribus c**,** Registration Page. Last accessed August 25th 2019. Available from **https://transkribus.eu/wiki/index.php/Registration.**

Wigington, C., Tensmeyer, C., Davis, B., Barrett, W., Price, B., & Cohen, S. (2018). Start, Follow, Read: End-to-End Full-Page Handwriting Recognition. In European Conference on Computer Vision. Springer, Cham.

Wikipedia a, PyCharm Page. Last accessed March 30th 2020. Available from **https://en.wikipedia.org/wiki/PyCharm.**

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

Younes, M., & Abdellah, Y. (2015). Segmentation of Arabic handwritten text to lines. Procedia Computer Science.

نحو حل متكامل للتعلم الآلي للتعرف على الوثائق العربية المكتوبة بخط اليد

إعداد
صفاء الصوالحة

المشرف
الأستاذ الدكتور غيث علي عبندة

ملخــص

تعد اللغة العربية لغة القران الكريم ولغة الإسلام واحدة من أقدم اللغات في العالم وهي لغة سامية. وعلى الرغم من عقود من البحث، لا توجد حلول جذرية للتعرف على النصوص المكتوبة بخط اليد مثل النصوص العربية.

يعد تقسيم الصفحة إلى أسطر خطوة مهمه في مجال التعرف على الوثائق المكتوبة بخط اليد باستخدام طرق تعلم الآلة. وقد قام العديد من الباحثين بعمل العديد من الأبحاث لتطوير أنظمة لتعرف على الوثائق المكتوبة بخط اليد.

سمح التقدم الأخير في التعلم العميق والهياكل الجديدة ببناء أنظمة قادرة على التعامل مع كل من الجانب الثنائي الأبعاد للمدخلات والجانب المتسلسل للتنبؤ والشبكات العصبية المتكررة متعددة الأبعاد خاصة الذاكرة قصيرة المدى المرتبطة بالوظيفة الموضوعية للتصنيف المؤقت للربط بمعدلات أخطاء منخفضة في مجال التعرف على النصوص المكتوبة بخط اليد.

من أجل الحصول على تدريب جيد على تقنيات التعلم الآلي، يجب أن يكون لديك كمية كبيرة من البيانات. لذلك، اخترنا مجموعة بيانات كبيرة مكتوبة بخط اليد باللغة العربية، وهي مجموعة تحليل الوثائق وتصنيفها آلياً متعدد اللغات.

حقق نظام البدء والمتابعة والقراءة نتائج متطورة على مجموعة بيانات صعبة ومجموعة بيانات الاعتراف وإثراء وثائق الأرشيف ومجموعة تحليل الوثائق وتصنيفها آلياً متعدد اللغات. حيث بلغ معدل الخسارة لشبكة اكتشاف بداية السطر في مجموعة بيانات الاعتراف وإثراء وثائق الأرشيف (47.9)، بينما في مجموعة بيانات مجموعة تحليل الوثائق وتصنيفها آلياً متعدد اللغات، وصل معدل الخسارة إلى (21.9). أما بالنسبة لشبكة تتبع السطر، فقد بلغ معدل الخسارة (39.1) على مجموعة بيانات الاعتراف وإثراء وثائق الأرشيف، بينما في مجموعة بيانات مجموعة تحليل الوثائق وتصنيفها آلياً متعدد اللغات وصل (47.0).

وأخيرًا، بلغ معدل الخسارة في شبكة التعرف على الكتابة في مجموعة بيانات الاعتراف وإثراء وثائق الأرشيف (0.52) ويصل إلى (0.25) مجموعة بيانات مجموعة تحليل الوثائق وتصنيفها آلياً متعدد اللغات.