On the Training of Deep Neural Networks for Automatic Arabic-Text Diacritization

Asma Abdel Karim, Gheith Abandah
Computer Engineering Department
The University of Jordan
Amman, Jordan

***Abstract*—Automatic Arabic diacritization is one of the most important and challenging problems in Arabic natural language processing (NLP). Recurrent neural networks (RNNs) have proved recently to achieve state-of-the-art results for sequence transcription problems in general, and Arabic diacritization in specific. In this work, we investigate the effect of varying the size of the training corpus on the accuracy of diacritization. We produce a cleaned corpus of approximately 550k sequences extracted from the full dataset of Tashkeela and use subsets of this corpus in our training experiments. Our base model is a deep bidirectional long short-term memory (BiLSTM) RNN that transcribes undiacritized sequences of Arabic letters with fully diacritized sequences. Our experiments show that error rates improve as the size of training corpus increases. Our best performing model achieves average diacritic and word error rates of 1.45% and 3.89%, respectively. When compared with state-of-the-art diacritization systems, we reduce the word error rate by 12% over the best published results.**

Keywords*—Arabic text; automatic diacritization; bidirectional neural network; long short-term memory; natural language processing; recurrent neural networks; sequence transcription

I. Introduction

The Arabic language is vastly spoken and written in many countries around the world. Arabic scripts mainly exist in two forms: Classical Arabic (CA) represented in holy scripts and old books, and Modern Standard Arabic (MSA) which is a contemporary form of CA used nowadays to write stories, books, newspapers, and formal speeches. Moreover, people use dialects that differ from one region to another, to communicate in their everyday lives [1].

Arabic sentences consist of sequences of words, written from right to left, composed of letters and diacritics. Diacritics are generally zero-width characters that appear in the form of marks added above or below the letters. They provide syntactic and semantic distinction that is essential to pronounce and understand Arabic texts [2]. However, diacritics are optional in most texts, especially MSA texts. This causes problems in understanding the text for non-native speakers and children since they may not be able to infer diacritics from the context. Moreover, it poses challenges on automatic Arabic language processing applications which require text to be diacritized such as automatic speech recognition (ASR), text to speech (TTS), and machine translation (MT) [1].

The Arabic language consists of 28 letters and eight basic diacritics. A total of 36 variants of the Arabic letters result from adding the six Hamza letters (ئ، إ، ؤ، أ، آ، ء), the Teh Marbuta (ة), and the Alef Maksura (ى) to the basic 28 letters. These variants have the Unicode hexadecimal codes 0621–063A and 0641–064A. The eight basic Arabic diacritics are: three short vowel diacritics (Fatha, Damma, Kasra), three nunation (Tanween) diacritics, double consonant diacritic (Shadda), and the no-vowel diacritic (Sukun). Arabic diacritics have the Unicode hexadecimal codes 064B–0652. The nunation diacritics are Fathatan, Dammatan, and Kasratan. They can only appear on the last letter of the word. Shadda diacritic is usually combined with either a short vowel or nunation diacritic. With these combined forms, we get a total of thirteen possible different diacritization of a letter in the Arabic language. Table I show the Arabic diacritics along with their transliterated names and list their shapes and sounds when written on the letter Beh (ب).

Diacritics can be classified into two categories: lexemic diacritics and inflectional diacritics. Lexemic diacritics distinguish between words in Arabic morphology that have the same orthography (spelling) but different pronunciations and meanings [3]. Example 1 in Table II shows how adding diacritics to the word كتب in two different ways results in two different pronunciations and meanings. The diacritized word كَتَبَ, pronounced "kataba", is a verb which means "he wrote". The diacritized word كُتُب, pronounced "kutub", is a plural noun which means books. Specifying which diacritization form to use for a word depends on the context.

TABLE I. Arabic Diacritics, their Transliterated Names, and Pronunciations

Diacritic	Transliterated Name	Shape	Sound
Short Vowels	*Fatha*	بَ	/ba/
	Damma	بُ	/bu/
	Kasra	بِ	/bi/
Nunation (Tanween)	*Fathatan*	بً	/ban/
	Dammatan	بٌ	/bun/
	Kasratan	بٍ	/bin/
Double Consonant Diacritic	*Shadda*	بّ	/bb/
No-vowel Diacritic	*Sukun*	بْ	/b/

Example 2 in Table II shows how diacritizing the word كتب using one of the two mentioned forms depends on the context it appears in. More specifically, it differs based on the third word in the sentence. In the first case, it is diacritized as the verb "kataba" since the third word is the noun lesson (الدرس) indicating that this is a verb-subject-object sentence. In the second case, it is diacritized as the noun "kutubu" since the third word is the adjective useful (مفيدة) indicating that this is a nominal sentence. In more complex sentences, diacritizing a word may expand to depend on words even further away in the sentence.

Inflectional diacritics distinguish different inflected forms of the same word. The diacritic of the last letter in the word depends on the position and role of the word in the sentence [3]. Example 1 in Table III shows how placing the noun كُتُب "kutub" in three different positions changes the last letter ب diacritic between Fatha, Damma, and Kasra. The last letter diacritic is often referred to as end case diacritic. Restoring this diacritic is considered a challenging task even when performed manually since it depends on the way the sentence is formed syntactically. Moreover, words (both nouns and verbs) may be inflected by appending suffixes that add features such as voice, number, person, tense, case, and other categorical information [1].

Example 2 in Table III shows how the diacritic of the last letter changes when the verb كَتَب is inflected in three different ways to represent masculine second narration using Fatha in the word كَتَبْتَ, feminine second narration using Kasra in the word كَتَبْتِ, and first narration using Damma in the word كَتَبْتُ. Inflected words make syntactical position of the word affect the diacritization not only of the last letter, but even the letters before. Example 3 in Table III shows the plural noun كُتُبه which is inflected by adding the possessive masculine pronoun ـه. The diacritization of the letter ب which is the letter before the pronoun ـه is the one affected by the position of the word in the sentence.

Consequently, recovering diacritics of undiacritized Arabic text is a challenging yet an important task. Many models have been proposed to automate the process of diacritizing Arabic texts. The performance of these models has been measured using two main metrics that represent the accuracy of the model in providing correct diacritics for the input undiacritized text. These metrics are the diacritics error rate (DER) and word error rate (WER). DER is computed by finding the percentage of wrong diacritics to the total number of characters in the input sequences. WER is computed by finding the percentage of words with at least one wrong diacritic to the total number of words in the input sequences.

TABLE II. EXAMPLES OF LEXEMIC DIACRITICS

Example Number	Forms	Meaning	Pronunciation
1	كَـتَـبَ	he wrote	/kataba/
	كُتُب	Books	/kutub/
2	كَتَبَ أحمد الدرس.	Ahmad wrote the lesson.	
	كُتُبُ أحمد مفيدة.	Ahmad's books are useful.	

TABLE III. EXAMPLES OF INFLECTIONAL DIACRITICS

Example Number	Forms	Meaning	Pronunciation
1	كُتُبُ أحمدَ مفيدة.	Ahmad's books are useful.	/kutubu/
	قرأتُ كُتُبَ أحمدَ جميعها.	I read all Ahmad's books.	/kutuba/
	أعجبت بكُتُبِ أحمدَ جميعها.	I liked Ahmad's books.	/kutubi/
2	كَتَبْتَ	You wrote (masculine)	/katabta/
	كَتَبْتِ	You wrote (feminine)	/katabti/
	كَتَبْتُ	I wrote	/katabtu/
3	كُتُبُهُ مفيدة.	His books are useful.	/kutubuhu/
	قرأ أحمد كُتُبَهُ.	Ahmad read his books.	/kutubahu/
	اعتنى أحمد بكُتُبِهِ.	Ahmad took care of his books.	/kutubihi/

Although the best previous solutions have shown steady improvement in accuracy over time, we think that the latest accuracies can be improved further using better models and training datasets. In most cases, the accuracy is restricted due to the lack of large, cleaned training dataset with acceptable diacritization to character rate. In this work, we extend the cleaning process performed in [4] to include the entire Tashkeela dataset. We concentrate on finding the effect of the training dataset size on the diacritization accuracy and on reducing the error rates through using larger datasets. Finding the effect of the dataset size on model accuracy and the best training size would hopefully help interested researchers to reach even better accuracies. The cleaning process was performed in steps such that eight corpora are extracted and cleaned with incremental sizes in terms of number of sequences. We perform experiments that use these corpora to explore and analyze the effect of increasing the training dataset size on the accuracy of our baseline model.

We build on our previous experience in designing a model that exploits the efficiency of bidirectional long short-term memory (BiLSTM) recurrent neural networks in automatic diacritization of Arabic texts. These networks are characterized with their ability to utilize long-term past and future contexts to predict diacritics. Our work produces a cleaned dataset of 543,364 sequences with diacritization to character rate of at least 80%. This dataset can be used to experiment training more sophisticated diacritization models. Moreover, our best-performing BiLSTM model achieves DER of 1.45% and WER of 3.89%.

The rest of this paper is organized as follows. The next section reviews systems proposed to automate the diacritization of Arabic text. Section III provides background information of sequence transcription and recurrent neural networks. Section IV illustrates our experimental setup. Section V presents and discusses the results of our experiments and compares our best results with the results of previous best performing models. Finally, we conclude our work in Section VI.

II. Literature Review

Diacritization is the process of adding diacritics to the letters of undiacritized texts. This operation is essential to many applications that involve translation and text-to-speech (TTS) conversion. Many models have been proposed over the years to automate the process of diacritizing Arabic text. These models involve rule-based models, statistical models, and hybrid models. Rule-based natural language processing (NLP) systems depend on using a set of well-defined language-dependent rules which are formed by exploiting solid linguistic knowledge. These systems are based on dictionaries and/or morphological and syntactic analyzers/generators [5][6]. Although rule-based approaches achieve acceptable results, their main drawback is the difficulty of maintaining and including all aspects of the language in a comprehensive set of rules. This is even more significant with a complex language morphologically and syntactically like the Arabic language [7].

Statistical approaches use large corpora of diacritized texts to predict the probability distribution of diacritics for a sequence of characters. The main advantage of these approaches is that they do not depend on a set of rules to solve the problem and hence do not require solid linguistic knowledge. Statistical methods that have been applied to Arabic text diacritization include hidden Markov models (HMM) [8][9], n-grams [10], finite state transducers (FST) [11], conditional random fields (CRF) [12], and neural networks. Recently, most proposed systems combined statistical approaches with linguistic knowledge such that the stochastic process is guided by language specific rules, introducing hybrid approaches [3, 13-19].

More recently, RNNs have been successfully used to solve restoring diacritics of Arabic texts as a sequence transcription problem. Our previous work in [20] proposed, trained, and tested a bidirectional LSTM network that transcribes raw undiacritized Arabic sequences with fully diacritized ones. Error correction techniques were used as a post processing step to the output of the network to overcome some transcription errors. We also experimented preprocessing the RNN input using a morphological and syntactical analyzer in [21]. Mubarak et al. [22] implemented a sequence-to-sequence model using an encoder-decoder LSTM RNN with content-based attention. They used a fixed length sliding window of character-based n-words in the training process and a voting algorithm of n-gram probabilistic estimation to select the most likely diacritic form of a word. They trained their model using 4.5 million tokens and tested it using the freely available WikiNews corpus of 18,300 words.

In [4], Fadel et al. tested and compared a few existing web-based automatic diacritization tools. They produced a cleaned subset of 55K sequences from the Tashkeela dataset which is split into training, testing, and validation sets. In [23], they implemented and tested several neural network models that belong to two main approaches, feed forward neural networks and recurrent neural networks. They explored several models using different types of input layers, using a CRF classifier instead of the softmax layer, and optimizing gradients normalization using block-normalized gradient (BNG).

Darwish et al. [24] proposed an approach to automatic diacritization that consists of two bidirectional LSTM RNNs. The first network is responsible for core-word (i.e., all letters other than the last letter of the word) diacritics and the second is responsible for case-ending (i.e., last letter) diacritics. They trained and tested their approach on two sets: one that represents MSA texts and the other represents CA texts. Their model included post correction using a unigram language model.

In our most recent work [25], we trained and tested RNN models using two datasets: Linguistic Data Consortium's Arabic Treebank part 3 (LDC-ATB3) [26], and the cleaned subset of Tashkeela [4]. We performed extensive experiments to explore and analyze the effect of tuning several network parameters, such as the number of network layers and using dropout, on the accuracy and execution time of the tested models. We also experimented models built using different network architectures, alternative approaches to handle problems in sequence lengths, and multiple encoding methods for the diacritized output sequences.

Madhfar and Qamar [27] implemented and experimented automatic diacritization using three character-level deep learning models. The first model is a network that consists of six layers: an embedding layer, followed by three bidirectional LSTM layers, a projection layer, and finally, a softmax layer. The second model consists of an encoder and decoder with location-based attention. The third model consists only of the encoder part of the second model. Its core architecture is implemented using a 1-D convolution bank, a multi-layer highway network, and a bidirectional GRU network. The model is named CBHG (1-D Convolution Bank + Highway network + Bidirectional GRU).

In this paper, we experiment training a deep BiLSTM model using several datasets with incremental sizes extracted from the Tashkeela dataset. Our goal is to test the accuracy of the trained model in each case, thus investigating the effect of the training set size on the accuracy of diacritization. Moreover, our work includes extracting a cleaned corpus of the full dataset of Taskeela which includes only sequences with diacritization to letter rates greater than 80%.

III. Sequence Transcription

Many machine learning tasks can be implemented as a sequence transcription problem, in which input sequences are translated into corresponding output sequences. These include speech recognition, machine translation, and text to speech [28]. Arabic text diacritization has been expressed successfully as a sequence transcription problem as well [20-27]. In our work, an input sequence X consists of characters $x_1, x_2, x_3, \ldots, x_T$ that represent the undiacritized sequence. The output sequence Y is a sequence of diacritics $y_1, y_2, y_3, \ldots, y_T$ such that y_i is the diacritic of the letter x_i.

Recurrent neural networks (RNNs) have proved to perform best on sequence transcription problems. This is because cells' hidden states are functions of all previous states with respect to time. This provides RNNs with their ability to maintain correlations between data points in the input sequence and the

capability of pointing backward in time [28]. Basic recurrent neural networks are generalization of feedforward neural networks to sequences [29]. Given a sequence of inputs $(x_1, x_2, \dots, x_T)$, a standard RNN computes a sequence of outputs $(y_1, y_2, \dots, y_T)$. At each time step, a recurrent neuron receives the output vector from the previous time step $y_{i(t-1)}$, in addition to the input vector $x_{i(t)}$. Hence, $y_{i(t)}$ is a function of $x_{i(t)}$ and $y_{i(t-1)}$, which is a function of $x_{i(t-1)}$ and $y_{i(t-2)}$, which is a function of $x_{i(t-2)}$ and $y_{i(t-3)}$, and so on. Consequently, $y_{i(t)}$ is a function of all input vectors since $t = 1$ [30].

Sequence transcription problems solved using RNNs can be classified into four categories based on the lengths of input and output sequences [30]. One-to-one networks take an input sequences and produces an output sequence of the same length. Sequence-to-vector networks transcribe input sequences into one final output by ignoring all previous outputs. Vector-to-sequence networks take one input vector and produce an output sequence. The general sequence-to-sequence network has output sequence that is generally not of the same length as the input sequence. This type is often implemented using the encoder-decoder architecture [31]. In this work, we implement automatic Arabic diacritization as a one-to-one sequence transcription problem since for each input sequence of characters; the output sequence of diacritics is of the same length.

Long short-term memory (LSTM) RNNs were first proposed in [32] to deal with the basic RNNs' problem of decaying or slowly changing weights. This results in their disability to learn long dependencies in the input sequences. LSTM networks, on the other hand, which use purpose-built memory cells, can converge faster, and detect long-term dependencies in the sequences [28]. Each memory cell has two states, the short-term state (also used as the cell output) $h_{(t)}$ and a long-term state $c_{(t)}$. These states are updated using an input gate, a forget gate, an output gate, and a cell activation unit. The operation of these gates collectively enables the LSTM cell to capture long term patterns by recognizing important inputs, preserving them as long as they are needed, and extracting them whenever they are needed. Fig. 1 shows a basic RNN cell and an LSTM cell.

Conventional unidirectional RNNs can make use only of previous context. However, many sequence transcription problems, including diacritization, require exploiting future context as well. Bidirectional RNN layers achieve this by comprising two unidirectional layers that process the sequence in both time directions producing two hidden vectors. The output is a function of both vectors and, consequently, exploits past and future contexts [33]. Fig. 2 shows the general structure of the bidirectional neural network unfolded for three time-steps. RNNs are made even more powerful by stacking multiple layers on top of each other forming a deep RNN. Deep networks are necessary to solve complex transcription functions. In such architectures, the output sequence of one-layer acts as the input sequence for the next layer.

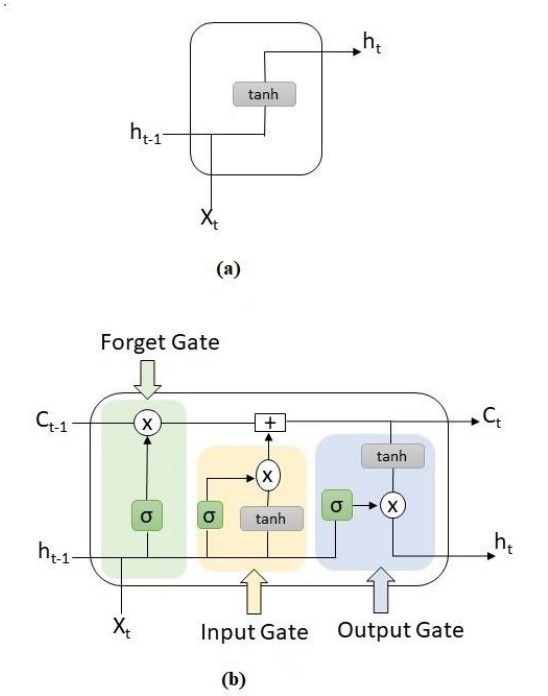

Fig. 1. (a) Basic RNN Cell, (b) LSTM Cell.

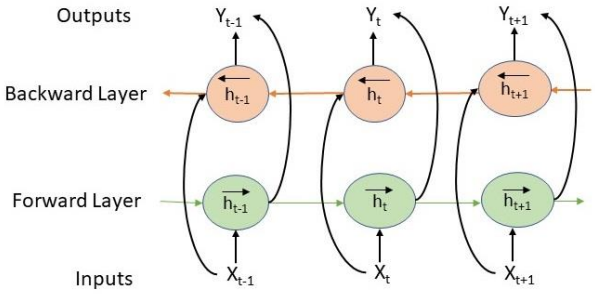

Fig. 2. General Structure of the Bidirectional Neural Network Shown unfolded for Three Time Step.

IV. Experimental Setup

In this section we provide the details of the experiments conducted in this work. We illustrate the methodology used, how datasets were extracted and preprocessed, the scheme used to encode sequences, and the structure of our baseline model. We performed all experiments on the Cyclone supercomputer of the High-Performance Computing Facility of The Cyprus Institute [34]. The processing and memory specifications of the used resources on the platform are listed in Table IV.

TABLE IV. Processing and Memory Specifications of the Experimental Platform

CPU	Intel Xeon Gold 6284 @ 2.5 GHz, 20 cores (40 threads), 27.5 MB cache
GPU	Nvidia Tesla V100-SXM2 @ 1.53 GHz, 5120 CUDA cores, 32 GB memory
Memory	192 GB DDR4-SDRAM

A. Methodology

We performed several experiments which involved training of our baseline model using corpora with different sizes. All experiments went through two phases: the first phase is training the model and the second phase is testing its diacritization accuracy. In the training phase, diacritics are removed from diacritized training sequences to generate undiacritized sequences. Generated undiacritized sequences represent the model input sequences whereas diacritic sequences are the model target sequences. Both undiacritized input sequences and diacritic target sequences are fed to the model after being encoded. Fig. 3 shows the steps performed in the training phase of the performed experiments.

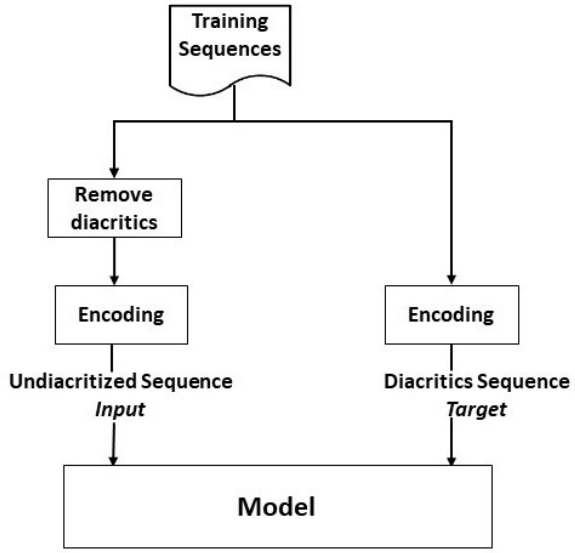

Fig. 3. Training Phase of Experiments.

In the testing phase, diacritics are removed from diacritized testing sequences. The trained model takes the generated undiacritized sequences as input to predict their diacritics. We perform minor corrections to the output sequences according to rules developed in our previous work in [20]. Corrected output sequences are stored in a text file named diacritized_output.txt. We test the accuracy of the model by comparing the model diacritized sequences, in the file diacritized_output.txt, with the correctly-diacritized target sequences, stored in a file named target_output.txt, in measures of DER and WER rates. Fig. 4 shows the steps performed in the testing phase of the performed experiments.

B. Training Datasets

The Tashkeela dataset [35] consists of 75 million diacritized words. In its main part, it is collected from 97 books filtered from 7079 books of Shamela library which is an Islamic electronic library. These books are example of CA text. Only 1.15% of the Tashkeela dataset consists of MSA texts which is drawn from modern books and crawled from the Internet. This makes Tashkeela mainly an example of CA. In [4], Fadel *et al.* extracted a subset of 55,000 sequences from the Tashkeela dataset with diacritization to character rate of at least 80%. The subset was cleaned by removing English letters and extra whitespaces, fixing some diacritization issues, and separating numbers from words, among other techniques. The subset was divided into 50,000 sequences for training, 2,500 sequences for validation, and 2,500 sequences for testing. This subset was used in our previous work in [25] to train and test the developed model.

In this work, we use the cleaning and filtering scripts developed by Fadel *et al.* [4] to extract the larger datasets used in our experiments. In addition, we wrap sequences such that they have maximum lengths of 400 characters. This step is performed to reduce the training time and memory usage and is based on experiments we conducted in our previous work [25]. One of the main goals of this work is to study the effect of incrementing the training data size on the diacritization accuracy. We use the 50k training sequences of Fadel et al. as a base dataset from which smaller training sets are derived and larger training sets are formed by adding more sequences to it. Three smaller subsets are derived by randomly selecting 6,250, 12,500, and 25,000 sequences from the basic dataset. Since the basic dataset is cleaned and filtered to meet the diacritization to character rate of 80%, except for wrapping to 400-character length, no further work was needed for these subsets.

In order to construct larger datasets, we randomly select sequences from the Tashkeela corpora to be added to enlarge our sets, starting with the 50K set. The sequences are selected to have at least 80% diacritics to characters rate. Then, they are processed using the cleaning scripts. To avoid duplication of sequences in our sets, the selected sequences are checked not to be already included in the set to be enlarged. Finally, we wrap sequences lengths to 400 characters. By repeating this process, we extracted datasets that consist of 100,000, 200,000, and 400,000 sequences. We also obtain the largest set used in our experiments which results from including all available sequences from Tashkeela that satisfy the above criterion, which is 543,364 sequences.

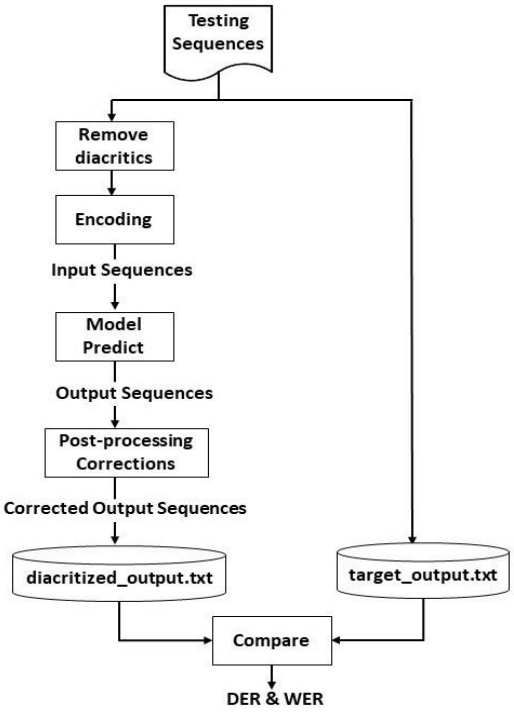

Fig. 4. Testing Phase of Experiments.

Except for the largest dataset, the sizes of the datasets are incremented by doubling the number of sequences from one set to the next. Moreover, the incremental process by which a new set is formed by adding sequences to the current set maintains the inclusion property, such that each dataset is a subset of the next. Table V shows size statistics of the used datasets in terms of word count, letters per word, words per sequence, and the number of sequences after the dataset is wrapped. All used subsets have close letters per word and words per sequence rates. For all experiments, we use the same validation set of 2,500 sequences, and testing set of 2,500 sequences to test the DER and WER of the trained model in each experiment.

TABLE V. SIZE STATISTICS OF THE EXTRACTED DATASETS

Dataset Size (# of sequences)	Word Count	Letters per Word	Words per Sequence	# of Sequences after Wrapping
6,250	259,847	3.98	41.5	7,675
12,500	522,502	3.97	41.7	15,334
25,000	1,059,573	3.97	42.4	30,847
50,000	2,103,071	3.97	42.1	61,453
100,000	4,180,191	3.97	41.8	122,817
200,000	8,410,559	3.97	42.1	245,994
400,000	16,854,689	3.97	42.1	492,288
543,364	22,729,365	3.97	41.8	667,990

C. Data Encoding

Sequences used in our experiments are either undiacritized consisting of letters only, or diacritized consisting of both letters and their diacritics. Undiacritized sequences are encoded using the Unicode representations of their letters. For diacritized sequences, we experimented using different encoding schemes in our previous work [25]. A one-to-one encoding scheme which represents each diacritic produced the best results in all performed experiments. Hence, we use this encoding scheme in this work. This scheme benefits from the fact that letters must not change between the input and the output sequences. Only diacritics must be added. Hence, it limits the classes at the output to the number of possible diacritics codes which is 16. Table VI shows the binary codes used for the eight Arabic diacritics. In Arabic, a letter may have two diacritics if one of them is *Shadda*. In this case, the diacritic code is formed by ORing the *Shadda* code (1000) with the other diacritic code. Fig. 5 shows an example of encoding the diacritized word صَيَّادٌ (hunter) which includes letters with no, one, and two diacritics.

D. Base Model

For building our models, we use Keras (Python deep learning library) with TensorFlow at the backend [36]. Our baseline model is an BiLSTM that consists of an embedding layer of 32 dimensions, four bidirectional LSTM layers each consisting of 256 cells, followed by a 16-cell fully-connected output layer. The Softmax function is used for activating the diacritic class with the highest probability at the output layer. Adam optimizer is used in training, and the sparse categorical cross entropy is used as the loss function. The batch size is set to 128 sequences for all experiments. In addition, the maximum number of epochs used in training is 100 with early stopping such that training stops if the validation accuracy does not improve for five consecutive epochs. Fig. 6 shows the structure of our baseline model.

TABLE VI. BINARY BIT CODES USED TO ENCODE DIACRITICS IN OUR EXPERIMENTS

Diacritic	Bit Code
No diacritic	0000
Fathatan (ً)	0001
Dammatan (ٌ)	0010
Kasratan (ٍ)	0011
Fatha (َ)	0100
Damma (ُ)	0101
Kasra (ِ)	0110
Sukun (ْ)	0111
Shadda (ّ)	1000

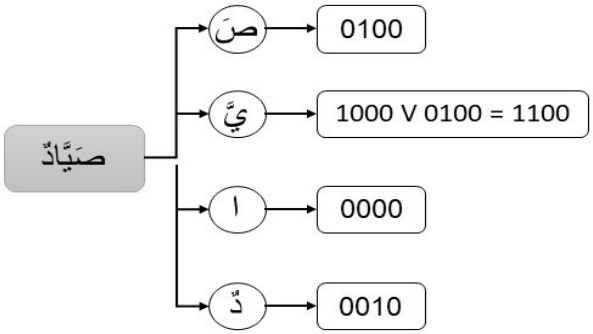

Fig. 5. Example Encoding the Diacritized Word صَيَّادٌ (Hunter).

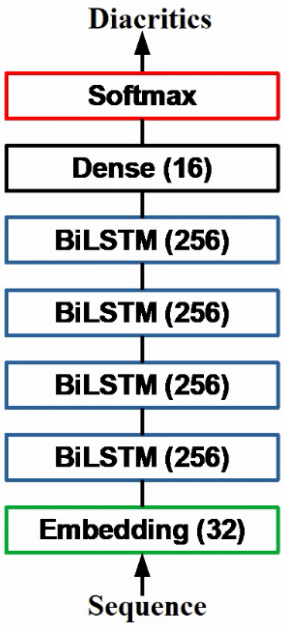

Fig. 6. The Structure of the base Model used in our Experiments.

V. Experiments and Results

The following subsections present the experiments performed and discuss their results. We also compare our best results with previous work.

A. Experiments

We experimented training our baseline model using the eight corpora we extracted from the Tashkeela dataset. We evaluated the trained model in each experiment in terms of time required to train the model and the model accuracy. We report the training time both in terms of the training total execution time and the average training time per epoch for each of the eight experiments. Table VII shows the total training time, average execution time per epoch, and number of executed epochs for each of the eight experiments. As expected, larger corpus size results in longer training time. However, the increase in the execution time is not directly proportional to the increase in number of sequences. This is dependent on the number of epochs which vary from one experiment to another based on when the early stop occurs. On the other hand, it can be observed that the increase in average time per epoch is proportional to the corpus size.

We report the performance of the models during training using the validation set in terms of validation loss and validation accuracy. Fig. 7 shows the validation accuracy and Fig. 8 shows the validation loss as functions of the training epochs for each experiment. In all reported results, we refer to each experiment by the number of sequences of its training dataset (i.e., 6,250, 12,500, 25,000, …). Training using larger number of sequences generally results in slower learning, higher values of accuracy and lower loss values. The best validation accuracy and validation loss achieved are 0.988 and 0.016, respectively, using the largest dataset of 543,364 sequences.

We tested the diacritization accuracy of the trained models using the eight extracted corpora. For all testing experiments, we use the test set of 2,500 sequences defined by Fadel *et al.* [4]. Fig. 9 shows diacritization error rates and word error rates for the eight models. The results show that both DER and WER improves as the number of sequences used in training increases. The best improvement, which is 22%, is observed when the training set increases from 6,250 sequences to 12,500. The improvement decreases gradually as we move towards larger datasets. No improvement is observed in the error rates when increasing the training set from 400,000 to 543,364 sequences. The best DER and WER achieved are 1.45% and 3.89%, respectively.

TABLE VII. Total Training Time, Average Execution Time Per Epoch, and Number of Executed Epochs for the Eight Experiments

Dataset Size (# of sequences)	Total Training Time (hours)	Epoch Average Training Time (hours)	# of Epochs
6,250	5.1	0.05	100
12,500	9.2	0.09	88
25,000	10.6	0.18	58
50,000	21.4	0.36	60
100,000	32.1	0.63	51
200,000	56.3	1.22	46
400,000	191.0	2.27	85
543,364	336.0	3.43	98

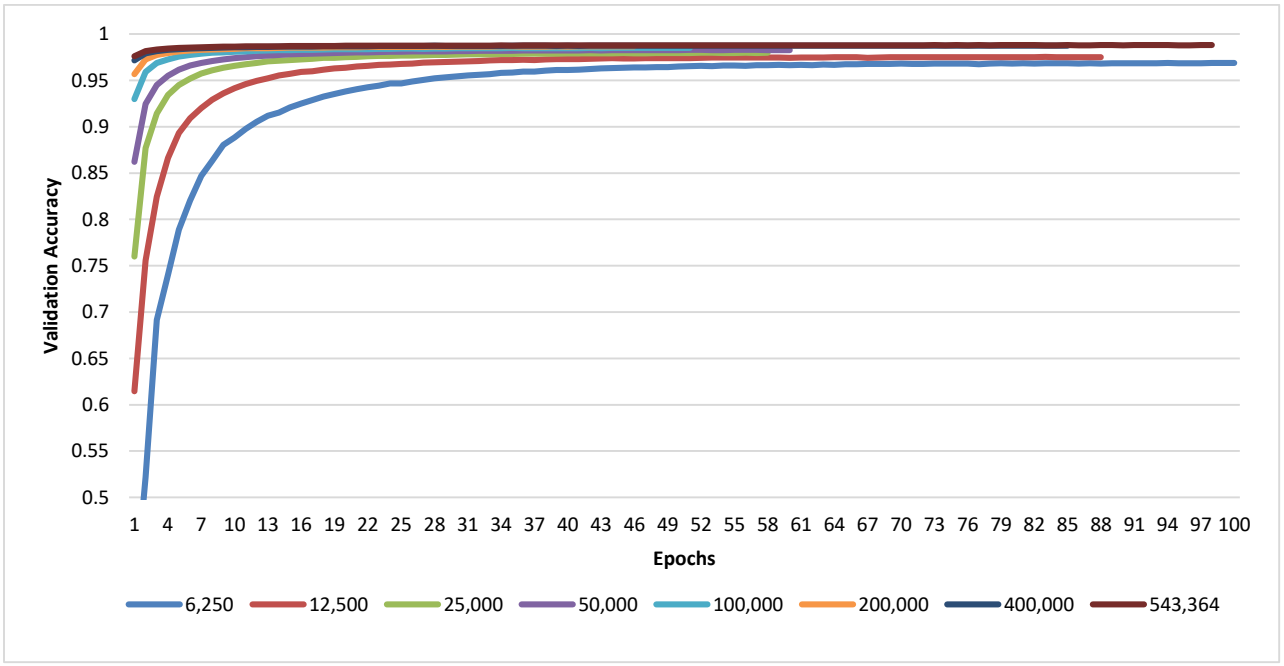

Fig. 7. Validation Accuracy Recorded during Training the Model using Datasets with different Number of Sequences.

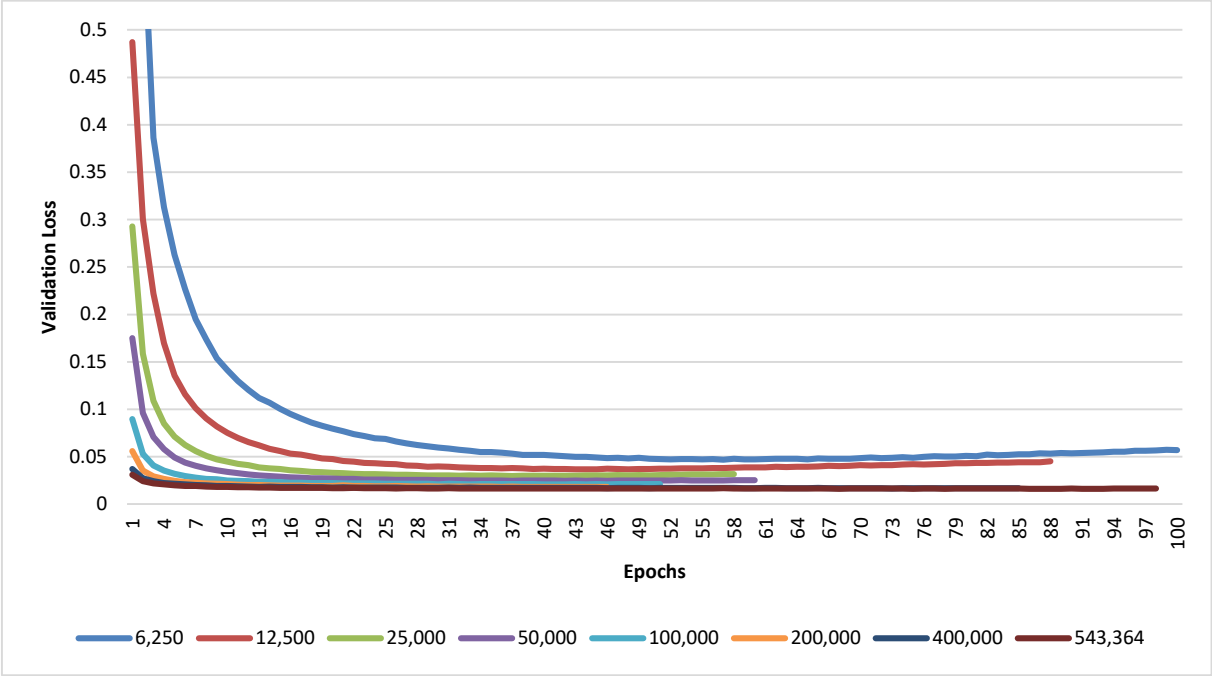

Fig. 8. Validation Loss Recorded during Training the Model using Datasets with different Number of Sequences.

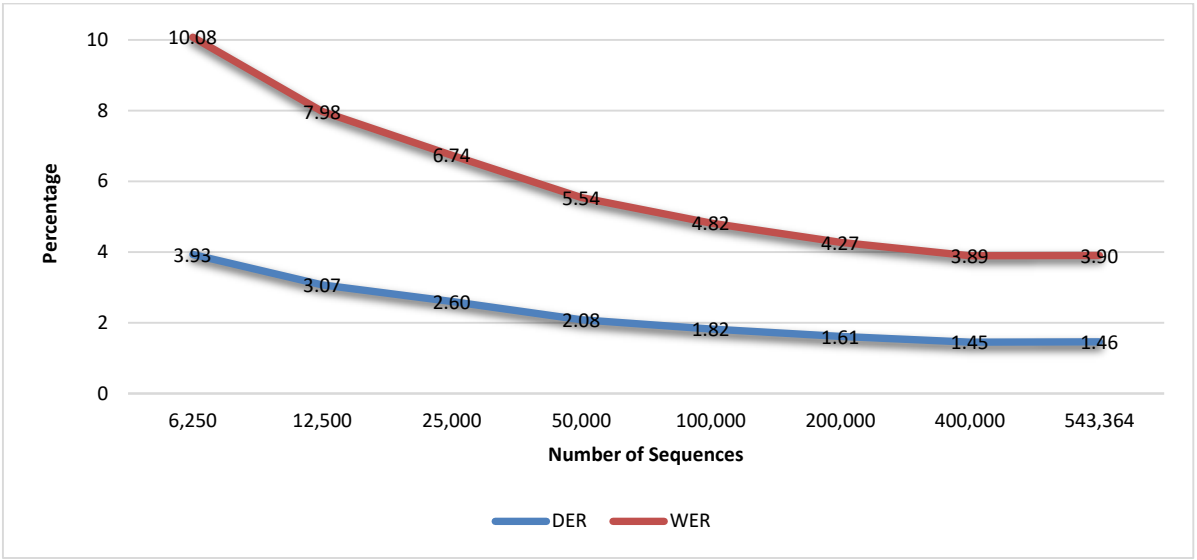

Fig. 9. DER and WER Values for Models Trained with different Number of Sequences.

We analyze errors of our system by enumerating the errors according to the number of errors per word and presence of end-case diacritization errors. The results of this analysis are shown in Fig. 10 and 11. Fig. 10 shows that for all dataset sizes, most of the miss-diacritized words have one diacritic error. Words with three or more diacritic errors are not frequent contributing to less than 6% of the errors in all experiments. Moreover, the ratio of multiple errors per one word decreases with larger datasets. Fig. 11 shows the end-case diacritization errors contribution in the DER and WER ratios. As explained earlier, end-case diacritization depends on the context and is subject to complex inflection rules. In our results, end-case diacritization errors contribute to about half of the word errors in all experiments. The best DER and WER values when ignoring end-case diacritization errors are 0.91 and 1.95, respectively.

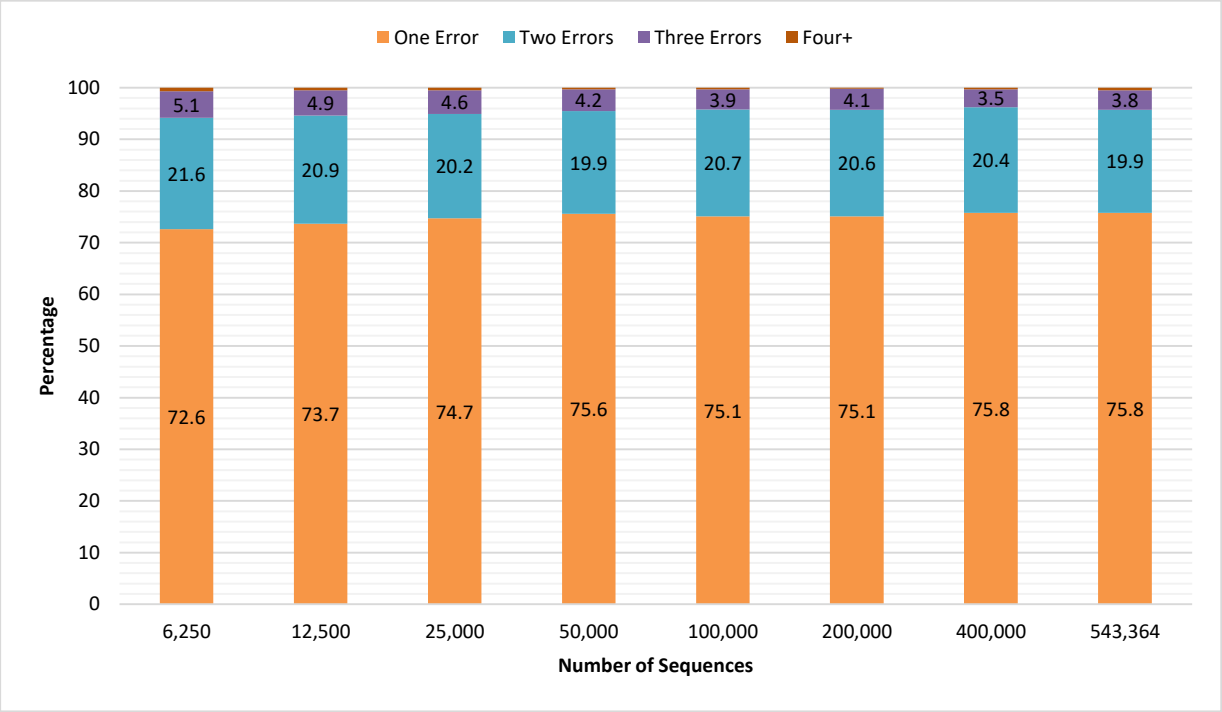

Fig. 10. Number of Errors per Word for each Experiment as a Percentage of the Total Number of Errors.

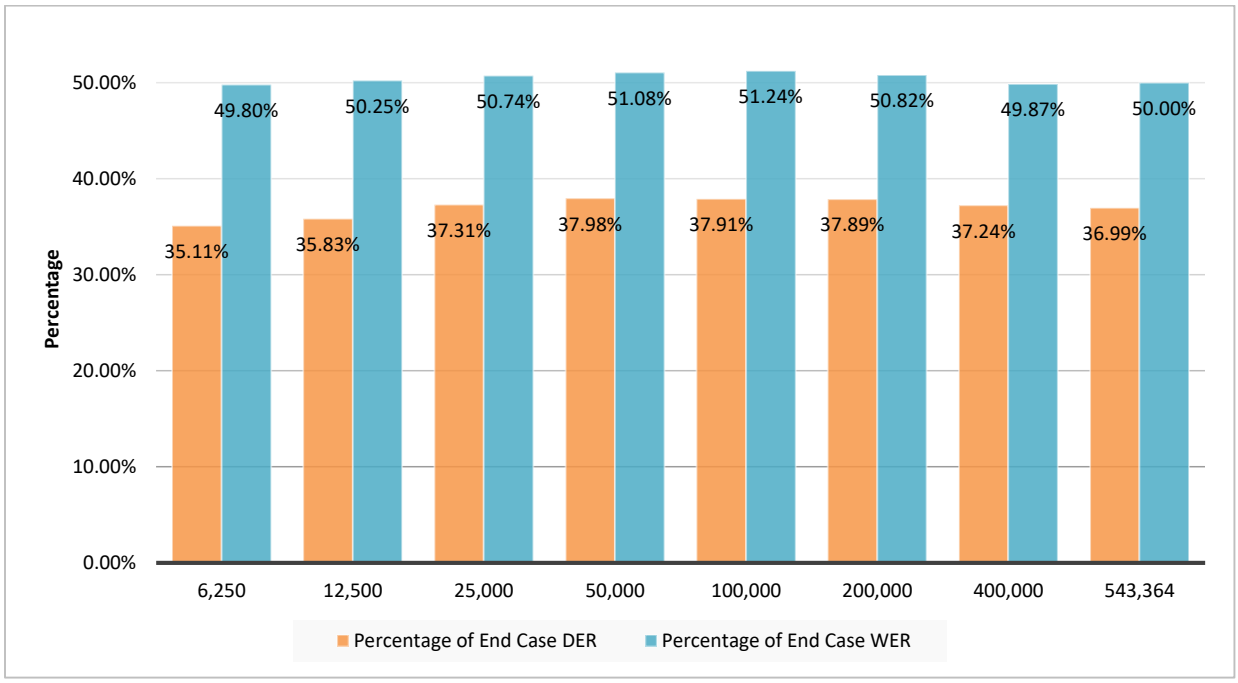

Fig. 11. Contribution of the End-case Diacritization Errors in the DER and WER Ratios.

B. Comparison with Previous Work

Our best results are reported here are for the model trained using 400,000 sequences. Table VIII summarizes the comparison of the best results in this work and best published systems. For each system, the table shows its publication year, the database used in evaluating it, and its DER and WER values both when including all diacritics errors (i.e., with case ending) and when ignoring last-letter diacritization errors (i.e., without case ending). The last column shows DER resulting from last letter diacritization errors only.

TABLE VIII. COMPARISON OF OUR BEST DER AND WER RESULTS WITH PREVIOUS WORK

System	*Dataset*	*All Diacritics*		*Ignore Last*		*DER Last*
		DER	*WER*	*DER*	*WER*	
Zitouni *et al.* (2006) [15]	ATB3	5.5	18	2.5	7.9	3.0
Habash&Rambow (2007) [3]	ATB3	4.8	14.9	2.2	5.5	2.6
Rashwan *et al.* (2011) [17]	ATB3	3.8	12.5	1.2	3.1	2.6
Said *et al.* (2013) [18]	ATB3	3.6	11.4	1.6	4.4	2.0
Fadel *et al*. (2019) [23]	Tashkeela	2.18	4.44	1.76	2.66	0.42
Abandah *et al.* (2020) [25]	ATB3	2.46	8.12	1.24	3.81	1.22
	Tashkeela	1.97	5.13	1.22	3.13	0.75
Madhfar and Qamar (2021) [26]	Tashkeela	1.13	4.43	0.84	2.47	0.29
This work	**Tashkeela**	**1.45**	**3.89**	**0.91**	**1.95**	**0.54**

Most previous work used either LDC's Arabic Treebank Part 3 (ATB3) [26], which represent an example of MSA, or Tashkeela, which represents an example of CA, or both. To the best of our knowledge, our previous work in [25] achieves the best published results for ATB3. The size of the ATB3 dataset is limited to 22,170 training sequences, which makes it unsuitable for the experiments we perform in this work. We do not include the results of Darwish et al. [14] since they use different training and testing datasets in both their MSA and CA experiments and hence comparison would not be fair. They used the training dataset of the RDI diacritizer in [18] and a test set of WikiNews for their MSA experiments. For their CA experiments, they used data from an undefined publisher.

The best DER and WER achieved in this work are 1.45% and 3.89%, respectively. This improves over our previous work which used a subset of the Tashkeela dataset and reported a DER of 1.97% and a WER of 5.13%. We compare our results with the best results of Fadel *et al.* [23] and Madhfar and Qamar [26] since both works use the Tashkeela dataset for training and testing. We outperform the model developed by Fadel *et al.* in DER and WER both with and without case ending. However, they achieve better last letter diacritization error rate.

Among the models they experimented, Madhfar and Qamar report the best DER and WER values for their CBHG model. In our comparison, the CBHG model of Madhfar and Qamar achieves the best DER in all cases. However, our best-performing model word error rates outperform those of the CBHG model indicating that our model results in less percentage of wrongly diacritized words. Noting that they perform their own cleaning and filtering process, but with different rules, to extract datasets used in training their models. It's worth mentioning that our best-performing model outperforms the baseline model of Madhfar and Qamar, which is a deep BiLSTM RNN. DER and WER values reported for their baseline model are 2.24% and 8.74%, respectively.

It can be observed that our base model achieves results that are comparative to more complex models such as the CBHG model proposed by Madhfar and Qamar. This shows that training using a large clean dataset with high diacritization to letter rate provides competitive diacritization accuracy. Training more-sophisticated models using such a dataset would certainly provide even better results. Although this work involves experimentations using a basic BiLSTM RNN, it generates cleaned corpora with incremental sizes that can be used to experiment with several other models. Moreover, it shows that state-of-the-art error rates could be achieved when training using large clean corpora.

VI. CONCLUSION

Automating diacritization of Arabic texts is a crucial operation for many Arabic NLP applications. In this paper, we have conducted several experiments to study the effect of changing the training data size on performance. Our work included generating several cleaned subsets of the Tashkeela corpora with incremental size in terms of number of sequences. Our largest subset, which consists of 543,364 sequences, can be used for training other models and comparing them, such as the model used by Madhfar and Qamar [26]. Our baseline model is a deep LSTM bidirectional RNN. We evaluated the performance of our baseline model during training using each of the generated corpora by monitoring the validation loss and accuracy using the validation set. We tested the diacritization accuracy of the model after being trained using each corpus by finding its DER and WER values when diacritizing the 2,500-sequence testing set.

Our experiments indicate that performance of the trained model improves as training set size increases. However, improvement in DER and WER values decreases as the number of sequences increases. Best achieved DER and WER values are 1.45% and 3.89%, respectively, using a training dataset size of 400,000 sequences (about 17 million words). Our WER value is the best when compared with other state-of-the-art results. In order to further improve the performance, we aim to experiment with other proposed models and to develop a loss function that considers unharmful differences between the output and target sequences when training is performed.

ACKNOWLEDGMENT

This work was supported by computing time granted on the Cyclone supercomputer of the High Performance Computing Facility of The Cyprus Institute.

REFERENCES

[1] Farghaly, and K. Shaalan, "Arabic natural language processing: challenges and solutions," ACM Transaction on Asian Language Information Processing, vol. 8, no. 4, pp. 1–22, Dec. 2009.

[2] N. Y. Habash, "Introduction to Arabic natural language processing," in Synthesis Lectures on Human Language Technologies. Morgan and Claypool Publishers, 2010.

[3] N. Habash and O. Rambow, "Arabic diacritization through full morphological tagging," in Conference on North American Chapter of the Association for Computational Linguistics, Rochester, New York, USA, 2007, pp. 53-56.

[4] A. Fadel, I. Tuffaha, B. Al-Jawarneh and M. Al-Ayyoub, "Arabic text diacritization using deep neural networks," 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 2019, pp. 1-7, doi: 10.1109/CAIS.2019.8769512.

[5] Y. El-Imam, "Phonetization of Arabic: rules and algorithms," Computer Speech and Language, vol. 18, no. 4, pp. 339–373, Oct. 2004.

[6] K. Shaalan, "Rule-based approach in Arabic natural language processing," International Journal on Information and Communication Technologies (IJICT), vol. 3, no. 3, Serial Publications, pp. 11–19, 2010.

[7] A. M. Azmi and R. S. Almajed, "A survey of automatic Arabic diacritization techniques," Natural Language Engineering, vol. 21, no. 3, pp. 477-495, 2013, doi:10.1017/S1351324913000284.

[8] Y. Gal, "An HMM approach to vowel restoration in Arabic and Hebrew," In Proceedings of the ACL-02 Workshop on Computational Approach to Semitic Languages (SEMITIC '02), Philadelphia, Pennsylvania, USA, 2002, pp. 27-33, doi.org/10.3115/1118637.1118641.

[9] E. Elshafei, H. Al-Muhtaseb, and M. Alghamdi, "Statistical methods for automatic diacritization of Arabic text," In Proceedings of Saudi 18th National Computer Conference (NCC18), Riyadh, Saudi Arabia, 2006, pp. 301-306.

[10] Y. Hifny, "Smoothing techniques for Arabic diacritics restoration," In Proceedings of the 12th Conference on Language Engineering (ESOLEC '012), Cairo, Egypt, 2012, pp. 6-12.

[11] R. Nelken, and S. Shieber, "Arabic diacritization using weighted finite-state transducers," In Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages (SEMITIC '05), Ann Arbor, MI, 2005, pp. 79–86.

[12] A. S. Azim, X. Wang, and K. C. Sim, "A Weighted Combination of Speech with Text-Based Models for Arabic Diacritization," In 13th Annual Conference of International Speech Communication Association, Portland, OR, USA, 2012, pp. 2334-2337.

[13] D. Vergyri and K. Kirchhoff, "Automatic Diacritization of Arabic for Acoustic Modelling in Speech Recognition," In Workshop on Computational Approaches to Arabic Script-based Languages, Geneva, Switzerland, 2004, pp. 66-73.

[14] S. Ananthakrishnan, S. Narayanan, and S. Bangalore, "Automatic diacritization of Arabic transcripts for automatic speech recognition," In Proceedings of the International Conference on Natural Language Processing (ICON-05), Kanpur, India, 2005,

[15] I. Zitouni, J. S. Sorensen, and R. Sarikaya, "Maximum entropy based restoration of Arabic diacritics," In 21st International Conference on Computational Linguistics, Sydney, Australia, 2006, pp. 577-584.

[16] K. Shaalan, H. Abo Bakr, and I. Ziedan, "A hybrid approach for building Arabic diacritizer," In Proceedings of EACL 2009 Workshop on Computational Approaches to Semitic Language, Morristown, NJ, 2009, pp. 27–35.

[17] M. Rashwan, M. Al-Badrashiny, M. Attia, S. Abdou, and A. Rafea, "A Stochastic Arabic diacritizer based on a hybrid of factorized and unfactorized textual features," IEEE Trans. Audio Speech Language Proceedings, vol. 19, no. 1, pp. 166-175, Jan. 2011.

[18] A. Said, M. El-Sharqwi, A. Chalabi, E. Kamal, "A hybrid approach for Arabic diacritization," In E. Mtai, F. Mezaine, M. Saraee, V. Sugumaran, and S. Vadera (eds.) Natural Language Processing and Information Systems, Lecture Notes in Computer Science, vol. 7934, pp. 53-64, Springer, 2013.

[19] M. Rashwan, A. Sallab, H. Raafat, and A. Rafea, "Deep learning framework with confused sub-set resolution architecture for automatic Arabic diacritization," IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP), vol. 23, no. 3, pp. 505-516, March 2015.

[20] G. A. Abandah, A. Graves, B. Al-Shagoor, A. Arabiyat, F. Jamour and M. Al-Taee, "Automatic diacritization of Arabic text using recurrent neural networks," International Journal on Document Analysis and Recognition (IJDAR), vol. 18, no. 2, pp. 183-197, March 2015.

[21] S. Alquda, G. Abandah, and A. Arabiyat, "Investigating hybrid approaches for Arabic text diacritization with recurrent neural networks." In Proceedings of the 2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT), Aqaba, Jordan, 2017.

[22] H. Mubarak, A. Abdelali, H. Sajjad, Y. Samih, and K. Darwish, "Highly effective Arabic diacritization using sequence to sequence modeling," In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, Minnesota, USA, 2019, pp. 2390-2395.

[23] A. Fadel, I. Tuffaha, B. Al-Jawarneh, and M. Al-Ayyoub, "Neural Arabic text diacritization: state of the art results and a novel approach for machine translation," in Proc. 6th Workshop Asian Transl.Hong Kong: Association Computational Linguistics, 2019, pp. 215-225.

[24] K. Darwish, A. Abdelali, H. Mubarak, and M. Aldesouki, "Arabic diacritic recovery using a feature-rich biLSTM model," arXiv: 2002.01207v1, 2020.

[25] G. Abandah and A. Abdel-Karim, "Accurate and fast recurrent neural network solution for the automatic diacritization of Arabic text," Jordanian Journal of Computers and Information Technology (JJCIT), vol. 6, no. 2, pp. 103-121, 2020.

[26] M. Maamouri, A. Bies, T. Buckwalter, and W. Mekki, "The Penn Arabic treebank: building a large-scale annotated Arabic corpus," In: NEMLAR Conference on Arabic Language Resources and Tools, Cairo, Egypt, 2004, pp. 102-19.

[27] M. A. H. Madhfar and A. M. Qamar, "Effective Deep Learning Models for Automatic Diacritization of Arabic Text," in IEEE Access, vol. 9, pp. 273-288, 2021, doi: 10.1109/ACCESS.2020.3041676.

[28] A. Graves, "Sequence Transduction with Recurrent Neural Networks," In Proceedings of the 29th International Conference on Machine Learning (ICML 2012), Edinburgh, Scotland, 2012, arXiv: 1211.3711v1.

[29] I. Sutskever, O. Vinyals, Q. V. LE, "Sequence to Sequence Learning with Neural Networks," In Advances in Neural Information Processing Systems (NIPS), Montreal, Canada, 2014, pp. 3104-3112.

[30] A. Geron, "Recurrent neural networks," in Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. USA: O'Reilly, 2017.

[31] K. Cho, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phase representations using RNN encoder-decoder for statistical machine translation," arXiv: 1406.1078v3, 2014.

[32] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[33] M. Schuster, and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673-2681, 1997.

[34] HPC Resources - High Performance Computing Facility (cyi.ac.cy), accessed July 2, 2021.

[35] T. Zerrouki and A. Balla, "Tashkeela: Novel corpus of Arabic vocalized texts, data for auto-diacritization systems,"' Data Brief, vol. 11, pp. 147-151, Apr. 2017.

[36] Google, "TensorFlow," Available: https://www.tensorflow.org/, accessed July 3, 2021.