Received November 7, 2020, accepted November 14, 2020, date of publication December 1, 2020,
date of current version January 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3041676

Effective Deep Learning Models for Automatic Diacritization of Arabic Text

MOKHTAR ALI HASAN MADHFAR[1] AND ALI MUSTAFA QAMAR[1,2]

[1]Department of Computer Science, College of Computer, Qassim University, Buraydah, Saudi Arabia
[2]Department of Computing, School of Electrical Engineering and Computer Science, National University of Sciences & Technology, Islamabad 44000, Pakistan

Corresponding author: Mokhtar Ali Hasan Madhfar (mokhtarmodhfer@gmail.com)

ABSTRACT While building a text-to-speech system for the Arabic language, we found that the system synthesized speeches with many pronunciation errors. The primary source of these errors is the lack of diacritics in modern standard Arabic writing. These diacritics are small strokes that appear above or below each letter to provide pronunciation and grammatical information. We propose three deep learning models to recover Arabic text diacritics based on our work in a text-to-speech synthesis system using deep learning. The first model is a baseline model used to test how a simple deep learning model performs on the corpora. The second model is based on an encoder-decoder architecture, which resembles our text-to-speech synthesis model with many modifications to suit this problem. The last model is based on the encoder part of the text-to-speech model, which achieves state-of-the-art performances in both word error rate and diacritic error rate metrics. These models will benefit a wide range of natural language processing applications such as text-to-speech, part-of-speech tagging, and machine translation.

INDEX TERMS Arabic language, Tacotron, diacritization, deep learning, text-to-speech.

I. INTRODUCTION

The Arabic language is a Semitic language and a native language for 22 countries. It is the liturgical language for over a billion Muslims throughout the world. The Arabic alphabet consists of 28 letters, as shown in Figure 1.

While these letters are enough, in most cases, for the native Arabic to resolve the ambiguity of homographs words from context, it is challenging for new language learners. For this reason, diacritics have been introduced to the Arabic language to provide grammatical and pronunciation information. Figure 2 shows the Arabic diacritics as they appear on the letter (t).[1]

The diacritic Shadda (∼) can be combined with other diacritics:

- Shadda + Fatha: (∼ a).
- Shadda + Damma: (∼ u).
- Shadda + Kasra: (∼ i).
- Shadda + Tanween al-Fatha: (∼ F).
- Shadda + Tanween al-Damma: (∼ N).
- Shadda + Tanween al-Kasra: (∼ K).

The total number of diacritics that should be recovered by the diacritization model is 15, including the no-diacritic option, which is abundant in Arabic writing. All diacritics that are composed of two diacritics, such as (∼ a), are treated as one diacritic.

أ	ب	ت	ث	ج	ح	خ	د	ذ	ر
ز	س	ش	ص	ض	ط	ظ	ع	غ	ف
ق	ك	ل	م	ن	ه	و	ي		

FIGURE 1. The Arabic Alphabet Letters.

Many of the Arabic words are homographs, which means that multiple words have the same writing form. Let us take an example of the Arabic word (Elm); searching through a corpus, we found 27 variations of this word where multiple variations can have the same meaning, but each variation has its unique pronunciation. Figure 3 shows a few sentences that use the word (Elm) with various meanings.

The diacritics are small strokes that may appear above or below each letter to identify its pronunciation. The diacritics also add grammatical information by using different diacritics at the last character of each word to indicate its inflectional. Figure 4 shows the sentences in Figure 3 in their diacritized form.

The majority of the Arabic text characters can be diacritized with either one or two diacritics, but some characters should not be diacritized all, such as the first and last characters of the first word of the last example in Figure 4.

The associate editor coordinating the review of this manuscript and approving it for publication was Shuihua Wang.

[1]Since the latex template does not support writing Arabic letters, the Buckwalter Transliteration is used in this paper.

Diacritic	Name	Pronunciation	Buckwalter Transliteration
		Short Vowels	
تَ	Fatha	/a/	a
تُ	Damma	/u/	u
تِ	Kasra	/i/	i
		Doubled Case Ending	
تً	Tanween al-Fatha	/an/	F
تٌ	Tanween al-Damma	/un/	N
تٍ	Tanween al-Kasra	/in/	K
		Syllabification Marks	
تّ	Shadda	Consonant Doubling	~
تْ	Sukuun	Vowel Absence	o

FIGURE 2. Arabic diacritics as they appear on the letter (t).

English Meaning	Example
Knowledge	ولقد كان لي بمخالطتكم علم
Knowledge	ما لدى الإنسان من علم
Know	فمن علم فيه صبرا
Knowledge	يحيط به علم إنسان
Mountain	كأنه علم في رأسه نار
Have been taught	من علم علما
Flag	علم الكويت رفع في ٢٤ نوفمبر ١٩٦١
Teach	الذي علم بالقلم

FIGURE 3. Various meanings of the word (Elm). The word has a different pronunciation in each example, even if it has the same meaning.

English Meaning	Example
Knowledge	وَلَقَدْ كَانَ لِي بِمُخَالَطَتِكُمْ عِلْمٌ
Knowledge	مَا لَدَى الْإِنْسَانِ مِنْ عِلْمٍ
Know	فَمَنْ عَلِمَ فِيهِ صَبْرًا
Knowledge	يُحِيطُ بِهِ عِلْمُ إِنْسَانٍ
Mountain	كَأَنَّهُ عَلَمٌ فِي رَأْسِهِ نَارُ
Have been taught	مَنْ عُلِّمَ عِلْمًا
Flag	عَلَمُ الْكُوَيْتِ رُفِعَ فِي ٢٤ نُوفَمْبِرِ ١٩٦١
Teach	الَّذِي عَلَّمَ بِالْقَلَمِ

FIGURE 4. The same sentences of Figure 3 in their diacritized forms. Diacritics help in resolving the ambiguity of Arabic words by providing additional information for pronunciation and grammar.

Given the correct diacritics for each word, there will not be much ambiguity in Arabic text, which can help many applications such as Text-to-Speech (TTS) and Part-of-Speech (PoS) tagging. Unfortunately, Modern Standard Arabic (MSA) text, the standard language of writing these days, is mostly written without diacritics. Recovering diacritics has gained much attention in the last few years. The researchers have employed many techniques such as Markov models, machine learning, and lately, deep learning, but the results, especially for TTS systems, are still far from perfect.

In this paper, we propose three deep learning models based on the recent advances in deep learning. The first model is a simple model used for comparison purposes. The second model is based on the encoder-decoder architecture [1], [2], which resembles the Tacotron TTS model [3]. The last one, which achieves state-of-the-art results, is based on the encoder of Tacotron with a few modifications to suit the diacritization problem.

II. BACKGROUND

Given a sequence of characters $x = (x_1, \ldots, x_n)$, our goal in diacritization is to generate a sequence of diacritics $y = (y_1, \ldots, y_n)$, where y_1 is the diacritic of the character x_1, which is chosen from 15 possible values. From a probabilistic perspective, the goal is to find a target sequence that maximizes the conditional probability of y given a source sentence x.

Automatic diacritization of Arabic text is one of the most challenging tasks in Arabic natural language processing. The researchers have employed several approaches to address this problem, including rule-based, statistical, hybrid, and deep learning ones.

The rule-based technique requires a deep understanding of the language to build a set of rules to recover the diacritics. Many systems use this technique alongside other techniques, such as statistical and deep learning techniques. This approach has been used by [4] to build a text-to-speech system and [5] to build several NLP systems such as machine translation and named entity recognition.

Many systems are purely based on statistical approaches, such as [6]–[9]. In an early work to address the diacritization problem, Gal [6] built a Hidden Markov Model (HMM) for Hebrew and Arabic languages. To train the Arabic language model, he used the Quran corpus and reported a 14% Word Error Rate (WER) with Case-Ending (CE). Nelken and Shieber [7] used weighted finite-state transducers. The system used a combination of three probabilistic language models: a word-based, a letter-based, and an orthographical language model. Their best model used a combination of 3-gram words, a clitics concatenation, and 4-gram letter models. The model got 23.61% WER, 12.79% Diacritic Error Rate (DER) with CE; and 7.33% WER, 6.35% DER without CE. Ananthakrishnan *et al.* [9] built a diacritization system for acoustic modeling of speech recognition. The system used acoustic information in combination with different levels of morphological and contextual constraints. The system attained a WER of 41.1% with CE. Elshafei *et al.* [10] proposed a system based on the HMM approach, along with the Viterbi Algorithm. The basic form of the algorithm achieved 4.1% DER. Alghamdi *et al.* [11] proposed the KACST Arabic Diacritizer (KAD). The system used an extensive list of frequently used quad-gram diacritics (68378 quad-grams) and achieved 8.52% WER. Hifny [12] built a system based on a statistical language model and dynamic programming. The system used two models: a bi-gram-based model that is first used for vocalization and a 4-gram character-based model used to handle the words that remain not-diacritized (also known as Out-Of-Vocabulary-OOV words). The system achieved 11.53% WER and 4.30% DER with CE; and 6.28% WER and 3.18% DER without CE. Schlippe *et al.* [8] treated the diacritization problem as a monotone machine translation problem. Using the same design scheme as machine translation, they built several models, including a character-based model, a word-based model, and a model that combined both character-level and word-level models. They also built a model based on the sequence labeling approach showing that the machine translation approach performed better. Their best model with post-editing achieved 15.6% WER, 5.5% DER with CE; and 10.3% WER, 3.5% DER without CE.

Many proposed diacritization systems use a hybrid approach, which combines linguistic knowledge with other techniques. Zitouni *et al.* [13] proposed a model based on a maximum entropy approach, which integrates lexical and PoS tagging features. The model achieved 17.3% WER, 5.1% DER with CE; and 7.2% WER, 2.2% DER without CE. Habash and Rambow [14] proposed a diacritization system that combined a tagger and a lexeme language model. The Buckwalter Arabic Morphological Analyzer (BAMA) is used to produce a list of possible analyses for a word, including the diacritization form. A set of taggers are trained to choose the best possible analysis. The system achieved 14.9% WER, 4.8% DER with CE; and 5.5% WER, 2.2% DER with CE. Shaalan *et al.* [15] integrated three techniques: 1) lexicon retrieval, 2) diacritized bi-gram, and 3) SVM statistical-based diacritizer. CE is treated as a separate post-processing task. The system achieved 12.16% WER and 3.78% DER with CE. Metwally *et al.* [16] used a multi-layered approach. The first layer used a first-order HMM to select the most probable sequence of morphological diacritized words along with their corresponding POS tags. The second layer used the Standard Arabic Morphological Analyser (SAMA) to produce a list of possible words for the OOV words. The last layer used a sequence labeling approach using Conditional Random Field (CRF) to annotate every word with one syntactic diacritic. The system achieved 13.7% WER with CE. Chennoufi and Mazroui [17] presented a hybrid system that combined linguistic rules and statistical treatments, which is based on four phases. The first phase consists of a morphological analysis using the second version of the morphological analyzer, known as Alkhalil morphological system. The second phase eliminated the invalid word transitions according to the syntactic rules. The third phase used a discrete HMM and Viterbi algorithm to determine the most probable diacritized sentence. Finally, the fourth phase used statistical treatments for words that were not analyzed by the Alkhalil analyzer. The system achieved 6.22% WER, 1.98% DER with CE; and 2.53% WER, 0.90% DER without CE, which are better than most of the previous systems. Darwish *et al.* [18] used a Viterbi decoder at word-level with back-off to stem, morphological patterns, transliteration, and sequence labeling based diacritization of named entities. They used Support Vector Machine (SVM) based ranking along with morphological patterns and linguistic rules to guess CE. They reported 12.76% WER, 3.54% DER with CE; and 3.29% WER, 1.06% without CE.

Deep Neural Networks (DNN) techniques have recently been used to solve this problem with significant improvements over the previous techniques. Many deep learning models are simple sequential models consisting of Recurrent Neural Networks (RNNs) and fully-connected (FC) layers. Al Sallab *et al.* [19] designed a system based on DNN and Confused Sub-Classes Resolution (CSR). The system attained 12.7% WER and 3.8% DER with CE. Abandah *et al.* [20] proposed a deep learning model based on Long short-term memory (LSTM) layers. The system significantly improved the diacritization over the previous works, achieving 5.82% WER, 2.09% DER with CE; and 3.54% WER, 1.28% DER without CE. Another system based on deep learning is designed by [21]. They examine several deep learning models with different architectures and different numbers of layers. Their best model, a three-layer bidirectional LSTM model, achieved 8.14% WER and 5.08% DER with CE. Fadel *et al.* [22] used two deep learning approaches: Feed-Forward Neural Network (FFNN) and RNN, with several techniques such as 100-hot encoding, embeddings, CRF, and Block-Normalized Gradient (BNG). Their best settings got 7.69% WER, 2.60% DER with CE; and 4.57% WER, 2.11% DER without CE. Mubarak *et al.* [23] presented a character-level sequence-to-sequence deep learning model. The model used a Neural Machine Translation (NMT) setup on overlapping windows of words. The system achieved state-of-the-art results of 4.49% WER and 1.21% DER with CE. The encoder-decoder model that we propose in this paper, resembles this model to a great extent, but our model adapted a TTS model, which requires significant modifications to fit the diacritization problem. Moreover, the encoder-decoder model uses the location-sensitive attention [24], whereas Mubarak *et al.* used the content-based attention [25]. Al-Thubaity *et al.* [26] built a model using bidirectional LSTM networks with CRF. The model

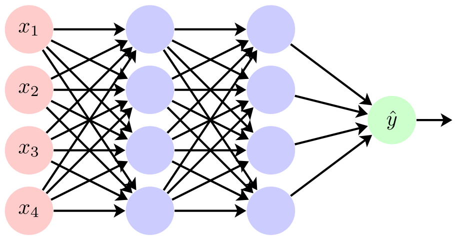


FIGURE 5. The structure of a simple deep learning model that consists of three fully-connected layers. Each layer can consist of any number of neurons, depending on the task.

achieved 4.92% WER and 1.34% DER in the Holy Quran corpus.

III. NEURAL NETWORKS

Neural networks have achieved excellent results in many complex learning tasks such as neural machine translation [1], [2], text-to-speech synthesis [3], [27], image captioning [28], and speech recognition [24]. This section introduces neural network layers and architectures that are used for building the diacritization models.

A. FULLY-CONNECTED LAYER

A Fully-Connected layer (FC) is a type of feed-forward layer in which each neuron is connected to all neurons in the next layer (Figure 5). These are powerful layers that are used in almost all deep learning models. In many cases, a fully-connected layer is required as a projection layer to change the output dimension. The output of one fully-connected layer is calculated as follows:

$$\hat{y} = xW^T + b \tag{1}$$

where W is the weight parameter and b is the bias parameter.

B. RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs) are powerful extensions of feed-forward layers that can be used with sequence data [29], [30]. It is extensively used in many applications such as NMT [1], [2], image captioning [28], and text-to-speech synthesis [3], [27].

RNNs layers add the previous hidden state h_{t-1} as an input to the unit at time step t (Figure 6). At each time step t, an RNN layer outputs h_t and $\hat{y}_t$ using the following equations:

$$h_t = g(W_{hx}x_t + W_{hh}h_t + b_h) \tag{2}$$
$$\hat{y}_t = g(W_{yh}h_t + b_y) \tag{3}$$

where h_t is the hidden state at time t, $\hat{y}_t$ is the output at time t, x_t is the input at time t, h_{t-1} is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time 0, and g is the activation function. An RNN layer can learn the probability distribution over a sequence, where the output at time step t is the conditional probability $p(x_t|\{x_{t-1}, \ldots, x_1\})$.

C. LONG SHORT-TERM MEMORY

One of the most popular RNN types is Long Short-Term Memory (LSTM) [31]. LSTM has been used in many state-of-the-art models in many tasks, such as [2], [27]. LSTM uses several gates to control the flow of information

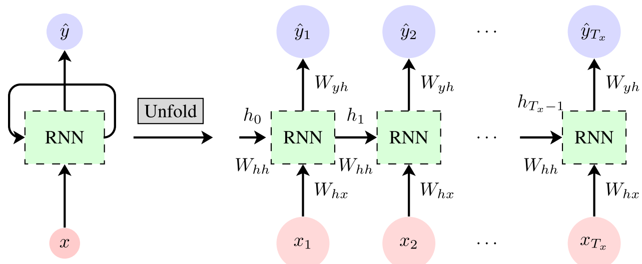


FIGURE 6. The structure of an RNN layer. The layer takes as input a sequence x and outputs a sequence $\hat{y}$. The unfolded structure of an RNN, on the right side, shows that an RNN receives, at each time step, the previous hidden state h_{t-1} and the current input x_t to generate $\hat{y}_t$ and h_t.

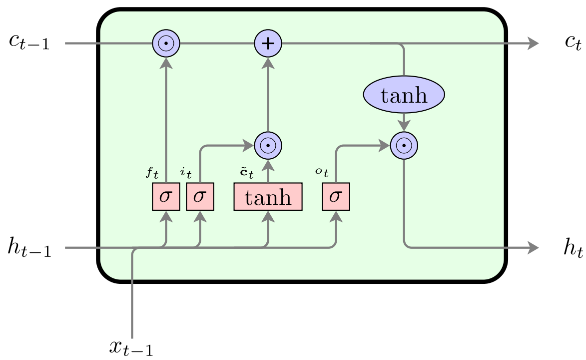


FIGURE 7. LSTM cell with gating mechanisms to control the flow of information.

in the network, which helps avoid the gradient's vanishing problem [32]. Figure 7 illustrates an LSTM unit. LSTM uses three gates: the input gate, the forget gate, and the output gate. All gates use the sigmoid activation function to output a value in the range [0, 1].

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \tag{4}$$
$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \tag{5}$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \tag{6}$$

where f_t is the forget gate, i_t is the input gate, o_t is the output gate, and σ is the sigmoid activation function. LSTM calculates the candidate memory cell $\tilde{c}_t$ just like the gates, but instead of using the sigmoid activation function, it uses the tanh activation function, which outputs values in the range $[-1, 1]$.

$$\tilde{c}_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \tag{7}$$

To calculate the output cell c_t, it uses the input and forget gates, as shown in Eq. 8.

$$c_t = f_t \odot c_{(t-1)} + i_t \odot \tilde{c}_t \tag{8}$$

where $\odot$ is the Hadamard product. The forget gate controls how much of the old memory cell c_{t-1} is retained, and the input gate i_t controls how much of the current candidate cell $\tilde{c}_t$ is taken into account. That is, if we have f_t close to 1 and i_t is close to zero, we mostly use the previous cell c_{t-1}, and vice versa. The output hidden state is calculated using the output gate o_t and the cell state c_t.

$$h_t = o_t \odot \tanh(c_t) \tag{9}$$

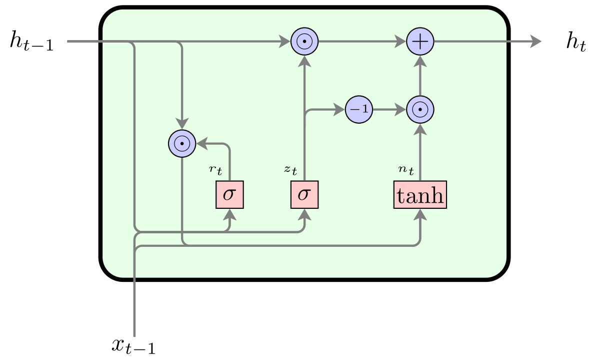


FIGURE 8. GRU cell with gating mechanisms and simpler architecture than LSTM.

D. GATED RECURRENT UNITS

A more recent but less complicated cell that performs as good as LSTM is Gated Recurrent Unit (GRU) [2] (Figure 8). Unlike LSTM, GRU consists of only two gates:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \tag{10}$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \tag{11}$$

where r_t is the rest gate, and z_t is the update gate. It then calculates the candidate hidden state as shown in Eq. 12.

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn})) \tag{12}$$

where n_t is the candidate hidden state. If the rest gate r_t is close to 1, the candidate hidden state will be calculated as normal RNN. If the rest gate r_t is close to 0, the previous hidden state will not be used, and n_t will be calculated like a Multilayer Perceptron (MLP). Lastly, it calculates the hidden state, which depends on the update gate.

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{(t-1)} \tag{13}$$

If the update gate z_t is close to zero, the candidate state is used. If the update gate is close to 1, then the current hidden state is equal to the previous hidden state.

E. BIDIRECTIONAL RECURRENT NEURAL NETWORKS

As discussed earlier, RNNs use the previous hidden state at time $t-1$ to generate h_t and $\hat{y}_t$. But for some problems, there is a need to look into future inputs to generate more accurate results. Bidirectional Recurrent Neural Networks (BRNN) were introduced by [33] to use information from both past and future inputs. BRNN adds a hidden layer in a backward direction: from the last input to the first as depicted in Figure 9.

$$\overrightarrow{h}_t = \phi(x_t W_{xh}^{(f)} + \overrightarrow{h}_{t-1} W_{hh}^{(f)} + b_h^{(f)}) \tag{14}$$
$$\overleftarrow{h}_t = \phi(x_t W_{xh}^{(b)} + \overleftarrow{h}_{t+1} W_{hh}^{(b)} + b_h^{(b)}) \tag{15}$$

where $\overrightarrow{h}_t$ is the forward hidden state, $\overleftarrow{h}_t$ is the backward hidden state, and ϕ is the activation function. The output of the backward and forward hidden states are concatenated to form the output of BRNN at each time step t.

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t] \tag{16}$$

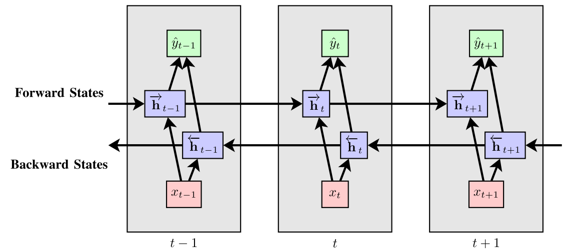


FIGURE 9. The structure of a bidirectional recurrent neural network unfolded in time for three time steps. At each time step, the input is passed to both forward and backward layers. Then, the output of the forward and backward layers are concatenated to form the output of BRNN.

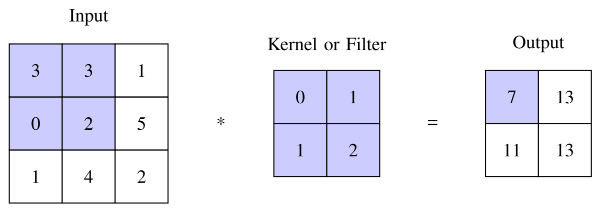


FIGURE 10. Two-dimensional cross-correlation operation, where the shaded portions represent the first pass of the kernel to calculate the first output. The output is calculated by summing the multiplications of each entry in the input portion with its corresponding value in the kernel: $3 \times 0 + 3 \times 1 + 0 \times 1 + 2 \times 2 = 7$.

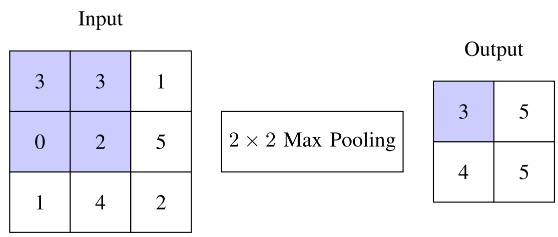


FIGURE 11. Maximum pooling using a 2×2 pooling window. The shaded portions represent the first pass of the 2×2 pooling window to calculate the first output: $\max(3, 3, 0, 2)$.

F. CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are a powerful family of deep neural networks used extensively in computer vision and have also been adapted to many other fields, such as natural language processing and text-to-speech.

Convolution Layer: The convolution layer consists of a set of learnable filters with multiple sizes that slide over the input multiplying the filters' values with the portion's values. This layer performs a cross-correlation operation, as depicted in Figure 10. *Pooling Layer:* This layer does not contain any learnable parameters. It reduces the input size by passing a filter over the input and taking either the maximum (max pooling) or the average (average pooling) of that portion. Figure 11 shows an example of maximum pooling using a 2×2 maximum pooling window.

G. EMBEDDING LAYER

Embedding is a technique in which a set of values (words or characters) are mapped into vectors of real numbers [34], [35]. It captures each input's semantics and syntactic, where similar inputs are grouped together in the embedding space. It is one of the most popular techniques in natural language processing used by many applications such as [1], [3]. We use this technique in all models at the character-level, where the embedding layer maps a sequence of characters $(x_1, \ldots, x_{T_x})$ to a sequence of character embeddings of- dimensionality d.

$$X = (C(x_1), \ldots, C(x_{T_x})) \tag{17}$$

where $X \in \mathbb{R}^{d \times T_x}$, d is the dimensionality of the embeddings, T_x is the length of the source characters, and C is the embedding lookup table.

H. HIGHWAY NETWORKS

Highway Networks were developed by [36] to overcome the difficulty of training deep neural networks. The architecture of Highway networks is inspired by LSTM recurrent networks. The architecture is a modified version of feed-forward networks with a gating mechanism that allows for computation paths along which information can flow across multiple layers without attenuation.

$$\hat{y} = H(x, W_H) \tag{18}$$

$$\hat{y} = H(x, W_H).T(x, W_T) + x.C(x, W_C) \tag{19}$$

where H is an affine transform followed by an activation function, T is the transform gate, and C is the carry gate. The carry gate C is set to $1 - T$ as shown in Eq. 20.

$$\hat{y} = H(x, W_H).T(x, W_T) + x.(1 - T(x, W_C)) \tag{20}$$

I. THE ENCODER-DECODER ARCHITECTURE

The encoder-decoder is a powerful architecture that has been used to solve many problems such as neural machine translation [1], [2], image captioning [28], and text-to-speech synthesis [3], [27].

The need for this new architecture comes from the fact that DNNs cannot be efficiently applied to variable-length sequences, which is a major problem since many tasks such as machine translation, text-to-speech conversion, and speech recognition vary in their input and output lengths.

The architecture consists of an encoder and a decoder. The encoder reads the input sequence $x = (x_1, \ldots, x_{T_x})$ to obtain a fixed-length vector c, which is used by the decoder to generate the output sequence $(y_1, \ldots, y_{T_y})$, one output at a time. Here, T_x is the length of the input sequence and T_y is the length of the target sequence. The most common approach in the encoder is to use RNNs to read the source sequence and generate a sequence of hidden states $h = (h_1, \ldots, h_{T_x})$ where h_t is computed as follows:

$$h_t = f(x_t, h_{t-1}) \tag{21}$$

The encoder summarizes the inputs into a context vector c to be used by the decoder to predict the target sequence. The vector c is the last hidden state in RNN or the concatenation of the left and right hidden states in BRNN. The decoder predicts the output at each time step conditioned on the previously predicted output and the context vector c.

$$p(y_1, \ldots, y_{T_y} | x_1, \ldots, x_{T_x}) = \prod_{t=1}^{T_y} p(y_t | c, \{y_1, \ldots, y_{t-1}\}) \tag{22}$$

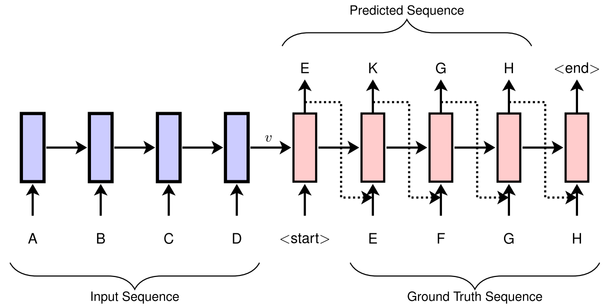


FIGURE 12. An example of the encoder-decoder architecture when used to generate the output sequence (E, K, G, H) from the input sequence (A, B, C, D). The decoder uses the special tokens <start> and <end> to indicate the start and end of decoding. In training, the ground truth input is passed to the next time step, while, in inference, the output of the decoder at time t is passed to the next time step $t + 1$ in a process called *teacher-forcing*.

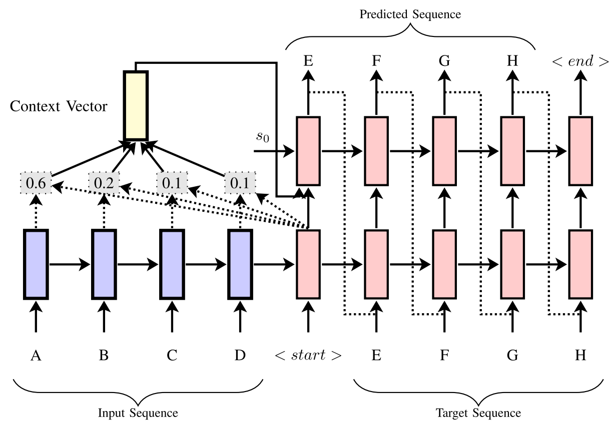


FIGURE 13. The structure of the attention mechanism for the first decoding step. At each time step, a unique context vector is calculated using the previous decoder hidden state s_{i-1}, and the output of the encoder h. The context vector c_i and the previous hidden state s_{i-1} are used to output $\hat{y}_i$.

When using RNN as a decoder, each conditional probability is calculated as follows:

$$p(y_t | c, \{y_1, \ldots, y_{t-1}\}) = g(y_{t-1}, s_{t-1}, c) \tag{23}$$

where g is a non-linear function that outputs the probability of y_t, and s_t is the hidden state of the decoder RNN. Figure 12 illustrates the encoder-decoder architecture.

J. ATTENTION MECHANISMS

One major issue with the encoder-decoder architecture is that the entire input sequence is compressed into a single fixed-length vector c. This fixed-length vector may not be able to summarize all the necessary information of the input sequence, especially for long sequences. In fact, [37] showed that the performance of NMT based on an encoder-decoder architecture gets worse rapidly as the input sequence increases. The attention mechanism extends the encoder-decoder architecture to solve this issue by selecting the portions of the input sequence that are most relevant to the decoded output [25]. Figure 13 illustrates the process of attention for the first time step. When using RNN as a decoder, the conditional probability of each output y_t using attention is calculated as shown in Eq. 24.

$$p(y_t|\{y_1,\ldots,y_{t-1}\},\{x_1,\ldots,x_{T_x}\}) = g(y_{t-1},s_{t-1},c_i) \tag{24}$$

where y_{t-1} is the previous output, s_{t-1} is the previous decoder hidden state, and c_i is the context vector. The major change from the encoder-decoder (Eq. 23) is the use of a distinct vector c_i for each output y_t, instead of a fixed-length vector c that is used for all output in the encoder-decoder architecture. To calculate the context vector c_i at each time step, we first calculate the score using each encoder output h_j and the previous decoder hidden state s_{i-1} as follows:

$$e_{ij} = score(s_{i-1}, h_j) \tag{25}$$

The score is modeled using a feed-forward neural network, which is trained jointly with the other components of the model.

$$e_{ij} = w^T \tanh(Ws_{i-1} + Vh_j + b) \tag{26}$$

The attention weights are calculated using a softmax as follows:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j=1}^{T_x} exp(e_{ij})} \tag{27}$$

where $\alpha_i \in R^{T_x}$ is the attention weights of the i-th decoder step. The context vector is calculated by multiplying the attention weights with the encoder output.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{28}$$

There are many variants of the attention mechanism that mainly differ in how they calculate the score. One type of attention is called location-sensitive attention proposed by [24] for speech recognition. The main purpose of this attention is to deal with long sequences. This attention is also used in text-to-speech [27].

The attention becomes location-aware by taking into account the previous attention weights or alignment. Eq. 25 is extended to use the previous alignment.

$$e_{ij} = score(s_{i-1}, \alpha_i, h_j) \tag{29}$$

This is done by extracting k vectors $f_{ij} \in R^k$ from the previous alignment α_{i-1}, by convolving it with a matrix $F \in R^{k\times r}$. These vectors $f_{i,j}$ are then used in the scoring equation:

$$e_{i,j} = w^T tanh(Ws_{i-1} + Vh_j + Uf_{i,j} + b) \tag{30}$$

where W, V, and U are the weight parameters.

K. CBHG MODULE

The CBHG module (1-D Convolution Bank + Highway network + Bidirectional GRU) was proposed by [38] for a character-level NMT model. It was adapted by [3] for building the Tacotron text-to-speech model. The CBHG module is illustrated in Figure 14. It consists of 1-D convolution filters, followed by highway networks, and bidirectional gated recurrent networks.

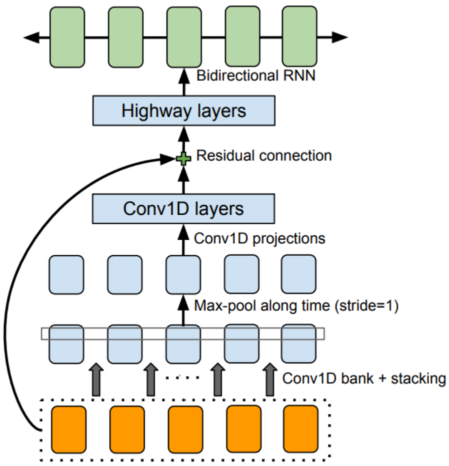


FIGURE 14. The CBHG module architecture, taken from [3]. It consists of a 1-D convolution bank, a multi-layer highway network, and a bidirectional GRU layer.

The module starts by convolving the input sequence with K sets of 1-D filters, where the k-th set contains C_k filters of length $(1, 2, \ldots, K)$. These filters model uni-grams, bi-grams, up to k-grams. The output of the convolutional layer is passed to a max-pooling layer over time with a stride of 1. The max-pooling layer's output is passed to several fixed-width 1-D convolution layers with residual connections to the original inputs. The output of these convolution layers is passed to a multi-layer highway network. The last layer is a bidirectional GRU RNN that extracts the sequential features of the input sequence.

IV. METHODOLOGY

In this section, we will discuss our implementation of three character-level deep learning models for restoring Arabic text diacritics. The first model is a baseline model that consists of an embedding layer and three bidirectional LSTM layers. The purpose of this model is to show how a simple deep learning model performs on the corpora. The second model is an encoder-decoder model with attention, which resembles NMT translation models [25], adapted from Tacotron [3]. The last model uses only the encoder part of the encoder-decoder model with a few modifications to output the diacritics. We will also discuss the corpus, the preprocessing of the corpus, and the evaluation metrics of the diacritization models.

A. DIACRITIZATION CORPORA

The crucial part of any deep learning model is the training data. Unfortunately, the Arabic language lacks any standard corpus that can be used for diacritics recovery. Two corpora are available: Tashkeela [39] (free) and ATL (paid). There are other corpora mentioned in other papers, such as the corpus used by [18], but they did not cite the corpora sources. We used the Tashkeela corpus, which consists of 75 million fully vocalized words from 97 classical Arabic books. While this corpus is large enough for training deep learning models, it has some issues as mentioned next:

- More than 99% of the corpus is from CA books, which may limit the generalization of the models when using MSA text.
- It contains many partially diacritized sentences, which can hurt the model learning of correct diacritization by learning to skip diacritization of some letters or words.
- The corpus has some wrong diacritics: It appears in many ways, such as some characters have more than two diacritics (which is impossible in the Arabic language), or they may have correct diacritics but in the reverse order.
- It is highly probable that the corpus also contains many other errors, especially errors that depend on grammatical rules, but we do not have any method to detect them.

B. CORPUS PREPROCESSING

The data preprocessing can be divided into two stages:

Data Cleaning: All characters except the Arabic letters, diacritics, and punctuation are removed from the corpus, which is essential as it keeps only the characters that contribute to the diacritization to be learned by the models. In inference, we designed algorithms that can restore the removed characters and put them back in their right places.

Data Splitting: In this stage, the corpus is divided into sentences. The absence of strict punctuation rules in Arabic makes recognizing sentence boundaries much harder than other languages such as English [40]. The sentences in Arabic can be conjoined with Arabic coordinators such as (w) wa and (f) fa, which are merged with the first word in the second sentence. What makes it even harder is that many Arabic words can contain these coordinators as their initial letter, making it challenging to split the text using these coordinators.

Moreover, the obvious indicator of the end of sentences in Arabic is a period, like many other languages. However, if we use a period as our primary indicator of a sentence ending, we will have very long sentences since much of Arabic writings, especially classical writings, conjoin the sentences with coordinators and use a period only at the end of each paragraph. For these reasons, we split the corpus using multiple delimiters to generate as many sentences as possible:

- For reasons of computational efficiency, we only use sentences of lengths at most 600 characters. This choice comes after experimenting with various models where a larger length will require a smaller batch size (< 16).
- Split the corpus using a period, which will generate many sentences, some of which are larger than 600 characters. All sentences with a length less than 601 are directly used in training; other sentences are processed further in the following step.
- Iterate over all sentences that are larger than 600 characters from the previous step, and split on punctuation: starting with ",", ";", followed by ":".
- If there are still some sentences larger than 600 characters after the last two steps, they are excluded from the training data.

To study the relationship between MSA and CA, and how the models trained using CA text can diacritize MSA text and vice versa, we created multiple versions of the corpus; each one is divided into training, validation, and test sets:

Classical Arabic Corpus (CA Corpus): All MSA sentences are removed from the corpus. This corpus constitutes more than 99.97% of the Tashkeela corpus. The corpus is divided into training (94%, 2,333,825 sentences), test (5%, 124,139 sentences), and validation (1%, 24,827 sentences) sets.

Modern Standard Arabic Corpus (MSA Corpus): All CA sentences are removed from the corpus. This corpus is tiny compared with the CA corpus, about 0.0026% or 6552 sentences. The purpose of using this corpus is to study the relationship between the CA and MSA language variations. The corpus is divided into training (82%, 5373 sentences), test (15%, 983 sentences), and validation (3%, 196 sentences) sets.

CA_MSA Corpus: The training sentences and validation sentences of the CA and MSA corpora are merged together, while the test sentences of both corpora are kept separate to study how much the models can improve using these additional MSA training sentences. Table 1 shows the statistics for CA and MSA corpora.

C. DEALING WITH DIACRITIZATION ERRORS

The task of annotating Arabic text with diacritics is challenging and time-consuming because diacritics appear on the text as small strokes above or below each letter. Additionally, the annotators need to be experts in the language since annotation of many characters depend on grammatical rules. These are probably the two main reasons why only minimal text of MSA is fully diacritized. As mentioned before, the Tashkeela corpus has many diacritization errors, some of which can be easily detected, while others are hard to detect since they depend on grammatical rules. Next, we discuss how we dealt with various corpus errors:

- Many sentences use wrong diacritics, such as using wrong combinations of diacritics or having characters with more than two diacritics (which is impossible in Arabic). We dealt with these types of errors by eliminating any sentence that contains them (Table 2).
- Many sentences are not fully diacritized, which is a common problem that can highly affect the models' performance. Since we do not have a way to check if a character can be diacritized or not, we checked all words of each sentence and eliminated any sentence that contains a word that is not fully diacritized. It is a partial solution but can get rid of many partially diacritized sentences as shown in Table 2.
- There are many errors in the choice of diacritics either on the core word or because of grammatical rules' violations. This type of error can only be detected and fixed with a manual review by experts.

Table 2 shows the number of removed sentences and the reasons for removal. It is also critical to note that diacritics

TABLE 1. Statistics of the CA and MSA corpora showing the number of sentences, the number of unique words, and the number of unique diacritized words.

No.	CA Corpus Train	CA Corpus Test	CA Corpus Eval	MSA Corpus Train	MSA Corpus Test	MSA Corpus Eval
Sentences	$2.33 \cdot 10^6$	$1.24 \cdot 10^5$	24,829	5,370	984	198
Unique Words	$7.95 \cdot 10^5$	$1.95 \cdot 10^5$	77,628	29,383	7,929	1,864
Unique Diacritized Words	$4.35 \cdot 10^5$	$1.25 \cdot 10^5$	54,057	22,552	6,916	1,745

TABLE 2. Number of removed sentences from Classical Arabic (CA) and Modern Standard Arabic (MSA) corpora with reasons for removal.

Removal Reason	CA Corpus	MSA Corpus
Small Sentences	$4.4 \cdot 10^5$	2,719
Partially Diacritized Sentences	$3.53 \cdot 10^5$	1,135
Sentences With No Diacritics	1,152	178
Large Sentences	59,140	4
Wrong Diacritics	3,483	65

TABLE 3. The frequencies of diacritics in the CA and MSA corpora, which shows significant variations that may lead the models to favor diacritics with higher frequencies.

Diacritics	CA Corpus	MSA Corpus
No Diacritic	$9.55 \cdot 10^7$	$1.93 \cdot 10^5$
Fatha	$6.79 \cdot 10^7$	$1.27 \cdot 10^5$
Kasra	$3.05 \cdot 10^7$	66,770
Sukun	$2.76 \cdot 10^7$	52,258
Damma	$1.84 \cdot 10^7$	32,626
Fathatah	$1.4 \cdot 10^6$	2,895
Dammatan	$1.25 \cdot 10^6$	2,371
Kasratan	$1.71 \cdot 10^6$	4,150
Shaddah	$2.18 \cdot 10^5$	283
Shaddah + Fatha	$7.73 \cdot 10^6$	15,769
Shaddah + Damma	$1.12 \cdot 10^6$	2,328
Shaddah + Kasra	$1.57 \cdot 10^6$	4,605
Shaddah + Kasratan	$1.02 \cdot 10^5$	244
Shaddah + Dammatan	88,267	172
Shaddah + Fathatah	77,380	271

differ significantly in their frequencies, which can affect the models' performances by favoring diacritics with higher frequencies. Table 3 shows the frequencies of diacritics in the CA and MSA corpora. The confusion matrix can be used to analyze the errors made by the models for each diacritic separately.

D. DIACRITIZATION SYSTEMS EVALUATION

The two popular metrics for evaluating diacritization systems are Diacritic Error Rate (DER) and Word Error Rate (WER). These metrics can be calculated either with Case-Ending (CE) or without CE. The calculation without CE excludes each word's last character from error calculation since they mostly depend on grammatical rules. Error rates without CE show the performance on the core word, while error rates with CE show the overall performance. It is almost always the case that both metrics' error rates without CE are lower (better) than the error rates using CE.

Diacritization Error Rate (DER): Given all characters and their correct corresponding diacritics, this metric calculates the percentage of characters that were not correctly diacritized. We calculate this metric by extracting diacritics from both the original and the predicted files and apply Eq. 31:

$$\mathrm{DER} = \frac{D_W}{D_W + D_C} \times 100 \tag{31}$$

where D_W is the number of wrong diacritics, and D_C is number of correct diacritics.

Eq. 31 calculates the errors of all characters, including spaces and punctuations. The characters that can be diacritized with more than one diacritic are treated as one diacritic. In the case of error rate without CE, the last character of each word is excluded from the calculation.

Word Error Rate (WER): This metric calculates the percentage of words that contain at least one diacritization error. In this calculation, we compare all words from both the original and the predicted files to calculate the percentage of unequal words.

$$\mathrm{WER} = \frac{W_W}{W_W + W_C} \times 100 \tag{32}$$

where W_W is the number of unequal words, and W_C is the number of equal words.

In the case of calculation without CE, the two compared words are equal even if both words' last diacritics are different.

E. RESTORING CLEANED CHARACTERS IN INFERENCE

In the data preprocessing stage, all characters except punctuation, Arabic characters, and diacritics are removed from the training data. However, in inference, the model should not modify the text in any way other than annotating each character with its corresponding diacritic. We designed simple algorithms that clean the text before feeding it to the diacritization model and then restore the cleaned characters after predicting the diacritics. The first algorithm is *CleanText* (Algorithm 1), which takes T, the text to be diacritized as input, and returns two texts: C is the text after cleaning, and R is the text that contains all characters that are removed. It is a simple algorithm that iterates the characters of the input text T and checks if each character is in the set V (a set that contains all possible characters that are accepted by the model: Arabic characters, punctuation, and diacritics) or not. It appends the character to C if it is in V; otherwise, it appends it to R.

Algorithm 2 shows how we use the *CleanText* function to clean the unwanted characters and restore them after the model predicts the diacritics sequence. The first step in diacritization is to remove any diacritics from the text. Then, it calls the *CleanText* function to output two texts: C and R. The *diacritizationModel* function outputs the diacritics of the

Algorithm 1 Text Cleaning Algorithm

Input: T - The Text to Be Cleaned
Output: C - The Cleaned Text, and **R** - The Removed Text

function CleanText(**T**)
  Let **V** be a set of all characters accepted by the model (Arabic characters, Punctuation, and Diacritics)
  Let **C** be an empty text
  Let **R** be an empty text
  for $c = 0$ **to T**.length **do**
    if T[c] **in V then**
      add **T**[c] to **C**
    else
      add **T**[c] to **R**
    end if
  end for
  return C, **R**
end function

Algorithm 2 Diacritize and Recover Removed Characters

Input: T - The Text to Be Diacritized
Output: TD - The Diacritized Text

function Diacritize(**T**)
  Remove any diacritics from the text **T**
  C,**R** = *CleanText*(**T**)
  D = *DiacritizationModel*(**C**)
  Let **TD** be an empty text
  Let $i = 0$
  for $c = 0$ **to T**.length **do**
    if T[c] **in R then**
      Append **T**[c] to **TD**
    else
      Append **T**[c] to **TD**
      Append **D**[i] to **TD**
      $i = i + 1$
    end if
  end for
  return TD
end function

cleaned text (D), where the length of D is equal to the length of C. The algorithm defines an empty text *TD*, which is the diacritized form of the input text. It iterates over the original text T and appends the characters to *TD* if it is in R; otherwise (if the character is in C), it performs the following three steps: 1) appends the character to *TD*, 2) appends the i^{th} diacritics to *TD*, and 3) increments i so that the next diacritics will be used.

F. THE BASELINE MODEL

The purpose of this model is to show how a simple deep learning model performs on the corpora. This model resembles in its architecture the model created for the same purpose by [21].

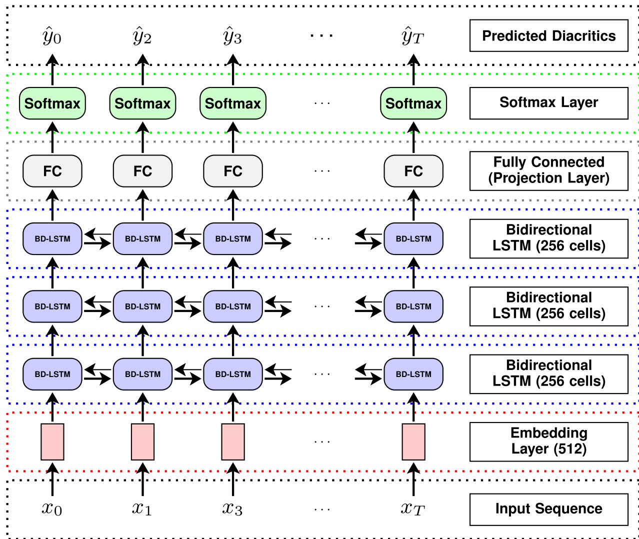


FIGURE 15. The structure of the baseline model. It consists of an embedding layer of 512 dimensions, followed by three bidirectional LSTM layers, each consisting of 256 cells, then a fully connected layer to project to the output size, and lastly, a softmax layer to output the probability for each diacritic.

It consists of an embedding layer (512 dimensions), followed by three bidirectional 256 cells LSTM layers, then a fully-connected layer used as a projection layer to project down the output of the last LSTM layer to the size of the diacritics vocabulary (15 including no-diacritic option). Lastly, a softmax layer is used to output a probability distribution over the output diacritics as shown in Figure 15. The model has 4.8 Million trainable parameters, which is the least compared with the CBHG and encoder-decoder models.

G. THE ENCODER-DECODER MODEL

The encoder-decoder model is adapted from Tacotron [3]. It has the same encoder with the same parameters but differs from Tacotron in the type of attention and the output of the decoder. This model uses the location-sensitive attention [24], which is more appropriate for long sequences. Moreover, the output of the decoder is a sequence of diacritics instead of a sequence of frames.

The encoder converts the input sequences into hidden representations, which are then used by the decoder to generate the diacritics. It represents the input sequences with an embedding layer of 256 dimensions. These vectors are passed to two layers of linear transformations called pre-net, which consists of two fully-connected layers with a tanh activation function and a dropout rate of 0.5. Its output is consumed by the CBHG module to generate the final representation of the input sequence.

The decoder consumes the output of the encoder and generates diacritics one at a time. During training, the decoder, in each time step, uses the ground truth output instead of the predicted output in a process called *teacher forcing*. However, in inference, it uses the predicted output instead. The decoder input is first processed by a pre-net, which has the same structure as the encoder pre-net. The pre-net output is passed

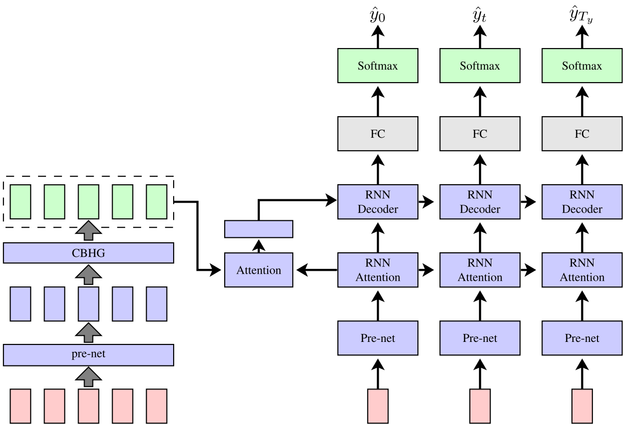


FIGURE 16. **The structure of the encoder-decoder model with attention, which is adapted from Tacotron [3]. The encoder part is the same as Tacotron, but the decoder and the attention are different.**

to an RNN attention layer, which uses the location-sensitive attention to generate the context vector in each time step. The context vector and the previous hidden state are then passed to two layers of RNN, each consisting of 256 cells with residual connections. The output of the decoder RNN layers is projected down to a fully-connected layer. We then used a softmax layer to output the probability distribution of the output diacritics. Figure 16 shows the architecture of this model.

The model has 5.2 Million trainable parameters, which is greater than the baseline model but less than the CBHG model. This model is the slowest to train since its decoder outputs one diacritic at a time, which is then passed to the next time step to generate the next output until the whole predicted sequence is generated.

H. THE CBHG MODEL

The encoder-decoder architecture is designed for problems in which input and output sequences are different. In the diacritization problem, however, the length of input and output sequences are the same. This property allows us to design a model based on only the encoder part of the encoder-decoder. The model has 14 Million trainable parameters, much more than the encoder-decoder model, but will be much faster since all the diacritics are predicted at once, unlike the encoder-decoder, where for each time step, the decoder predicts only one diacritic. We called this model the CBHG model since it used the CBHG module as its core architecture. We added a fully-connected projection layer and a softmax layer on top of the CBHG module to output the diacritics (Figure 17).

The CBHG model works as follows: a 512-dimensions embedding layer first processes the input sequence. The embedding output is passed to two layers of non-linear transformation called pre-net: the first consists of 512 units with a RELU activation function and a dropout probability of 0.5, and the second consists of 256 units with the same activation and dropout probability. The pre-net output is then fed to the CBHG module, which outputs the input sequence's final representation. We added a fully-connected layer to project

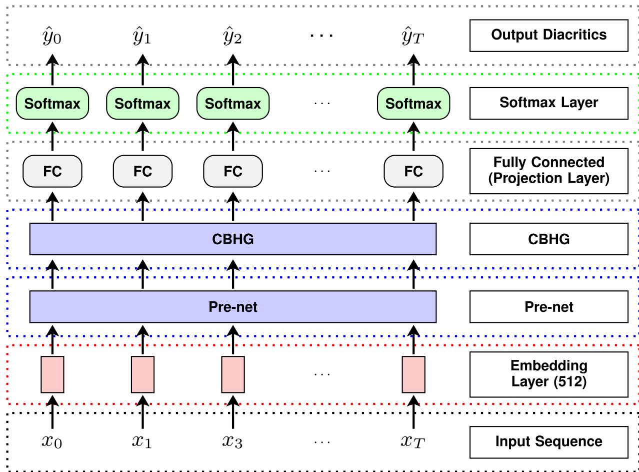


FIGURE 17. **The CBHG model architecture. It is implemented using only the encoder of the encoder-decoder model with more robust parameters. We added a fully-connected layer and a softmax layer on top of the encoder to output the probability distribution for each character.**

TABLE 4. **The results of training the three models for 500k steps using the CA corpus.**

	With CE		Without CE	
Model	WER	DER	WER	DER
Baseline Model	8.74	2.24	5.75	1.83
CBHG Model	4.47	1.14	2.49	0.85
Encoder-decoder	4.84	1.34	2.76	1.02

down the CBHG module's output to the number of possible diacritics. Lastly, we used a softmax layer to output the probability distribution for each diacritic.

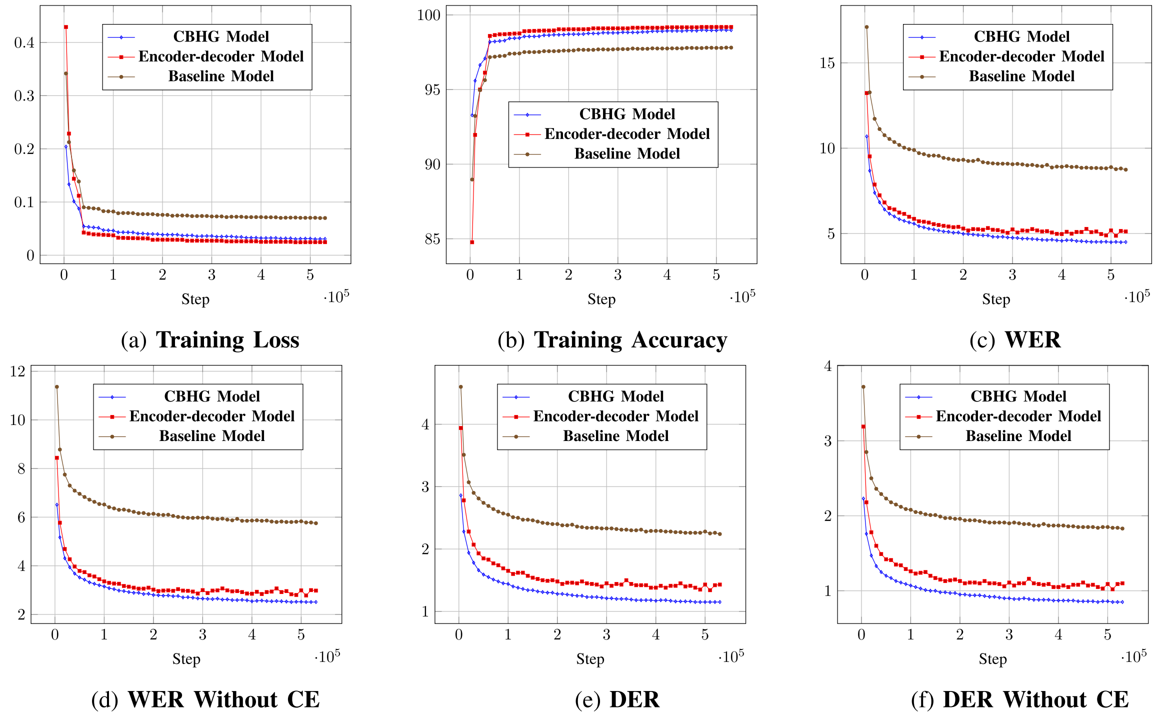
V. RESULTS

We trained the three models using the three corpora that we extracted from the Tashkeela corpus. We trained both the baseline and the CBHG models using RTX 2070 GPU with 64 batch size. The encoder-decoder model requires larger GPU memory, so we used the same GPU with a mixed precision and a batch size of 32. We used the Adam optimizer [41] with $\beta_1 = 0.9$, and $\beta_2 = 0.99$. We varied the learning rate during training, using the formula given by [42].

A. USING THE CA CORPUS

Our first experiment is to train the three models using the CA corpus for 500k steps (Table 4).

The CBHG model performs better than the other two models in every metric. Surprisingly, the CBHG performs better than the encoder-decoder model, given that it trains five times faster. We do not use *teacher forcing* during testing for the encoder-decoder model. The baseline model achieves the worst results, but its results are comparable with many previous results, such as [21]. Figure 18 illustrates the results of the three models during training using the validation set. Figures 18a and 18b show that the encoder-decoder model performs better than the other two models, but this is due to the *teacher forcing* mechanism that is used during training. When the model is tested without *teacher forcing*, the CBHG model performs better, as it is apparent in Figures 18 (c-f).



(a) **Training Loss** (b) **Training Accuracy** (c) **WER**

(d) **WER Without CE** (e) **DER** (f) **DER Without CE**

FIGURE 18. The results of the models during training using the validation set. The CBHG model learns faster and smoother than the encoder-decoder model. The training accuracy in b shows that the encoder-decoder model results are better than the CBHG model, but this is due to the *teacher forcing* that is used during training.

TABLE 5. The results of the models that were trained using the CA corpus when tested using the training sentences of the MSA corpus.

	With CE		Without CE	
Model	WER	DER	WER	DER
Baseline Model	34.95	9.86	25.69	8.33
CBHG Model	23.01	6.23	15.82	5.11
Encoder-decoder Model	24.0	6.92	16.84	5.77

TABLE 6. The results of the models after training for 50k steps using the MSA corpus.

	With CE		Without CE	
Model	WER	DER	WER	DER
Baseline Model	30.75	8.64	22.9	7.33
CBHG Model	23.48	6.41	16.76	5.37
Encoder-decoder Model	23.55	6.86	17.11	5.87

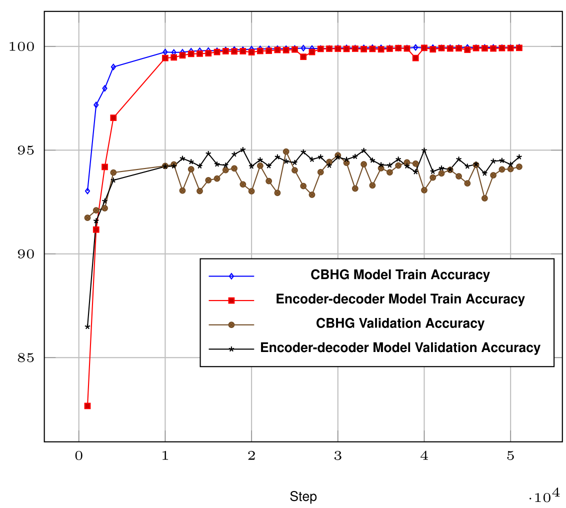


FIGURE 19. The models trained with the MSA corpus overfit the data since the training accuracies are close to 100%, while the validation accuracies keep fluctuating and are always less than 95%.

To study how the models that are trained with the CA corpus perform in MSA sentences, we tested the previous models with the training set of the MSA corpus. As Table 5 shows, the performance is much worse than the CA sentences in all metrics, which suggests significant differences between the two language variations.

B. USING THE MSA CORPUS

We replicated the previous experiment using the MSA corpus. As Table 6 shows, the MSA corpus could be considered small for training deep learning models, reflected in the far worse results produced by the models compared with the models that were trained using the CA corpus. We only trained the models for 50k steps since the models quickly overfit the data after 10k steps (Figure 19). However, the models produced results that were closer to the models that were trained with the larger CA corpus and tested with the MSA training data (Table 5). These results, again, suggest that there are significant differences in the MSA and CA language variations.

C. USING THE CA_MSA CORPUS

In this experiment, we investigate whether the additional MSA sentences can improve the results of the models. As discussed earlier, we merged the training and validation

TABLE 7. The results of the models after training for 500k steps using the CA_MSA corpus and tested using the CA test data.

Model	With CE		Without CE	
	WER	DER	WER	DER
CBHG Model	4.43	1.13	2.47	0.84
Encoder-decoder Model	4.71	1.29	2.68	0.98

TABLE 8. The results of the models after training for 500k steps using the CA_MSA corpus and tested using the MSA test data.

Model	With CE		Without CE	
	WER	DER	WER	DER
CBHG Model	21.28	5.65	14.75	4.64
Encoder-decoder Model	22.7	6.35	15.64	5.25

TABLE 9. Comparing the CBHG model and the encoder-decoder models with previous results.

System	With CE		Without CE	
	WER	DER	WER	DER
Shieber [7]	23.61	12.79	7.33	6.35
Hifny [12]	11.53	4.30	6.28	3.18
Schlippe et al. [8]	15.6	5.5	10.3	3.5
Zitouni et al. [13]	17.3	5.1	7.2	2.2
Habash and Rambow [14]	14.9	4.8	5.5	2.2
Shaalan et al [15]	12.16	3.78		
Chennoufi and Mazroui [17]	6.22	1.98	2.53	0.90
Darwish et al. [18]	2.76	3.54	3.29	1.06
Al Sallab et al. [19]	12.7	3.8		
Abandah et al. [20]	5.8	2.09	3.5	1.28
Belinkovand Glass [21]	8.14	5.08		
Fadel et al. [22]	7.69	2.60	4.57	2.11
Mubarak et al. [23]	4.49	1.21		
Al-Thubaity et al. [26]	4.92	1.34		
Encoder-decoder Model	4.71	1.29	2.68	0.98
CBHG Model	**4.43**	**1.13**	**2.47**	**0.84**

sentences of both CA and MSA corpora and kept the test sentences separate to test each variation. We trained only the CBHG and encoder-decoder models using this corpus. Table 7 shows the results when testing the models using the CA test data.

Both models have improved in all metrics using the additional MSA data compared with using only the CA corpus (Table 4). These are interesting results that require more investigation as we show earlier that there is a considerable difference between the CA and MSA language variations.

Lastly, we tested the models with the MSA corpus test data (Table 8). The results are also better than both the results when training using only the CA corpus (Table 5) and only the MSA corpus (Table 6).

Table 9 shows the results of the CBHG and encoder-decoder models compared with the state-of-the-art methods. The CBHG model achieves better results than all other methods in all metrics. While the results of Mubarak *et al.* [23] are very close to the CBHG model, the advantage of the CBHG model is that it is simpler and much faster.

VI. DISCUSSION

To study how the models perform for each diacritic, we used the confusion matrix to show in percentage how each diacritic

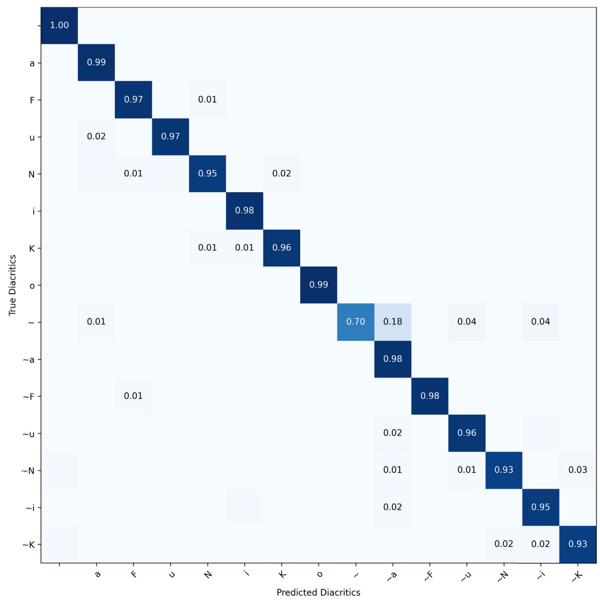


FIGURE 20. Confusion matrix of the CBHG model that was trained with the CA_MSA corpus and tested with the CA test data. For each diacritic, we show the percentage in which it was predicted. The empty cells indicate that the percentage is less than 0.01%.

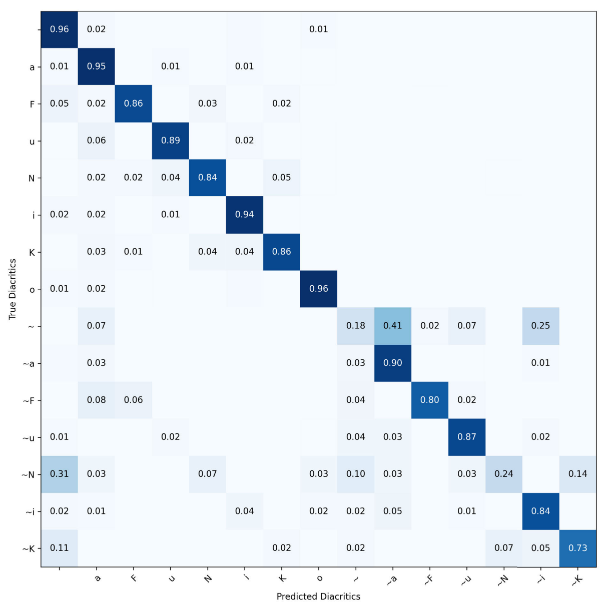
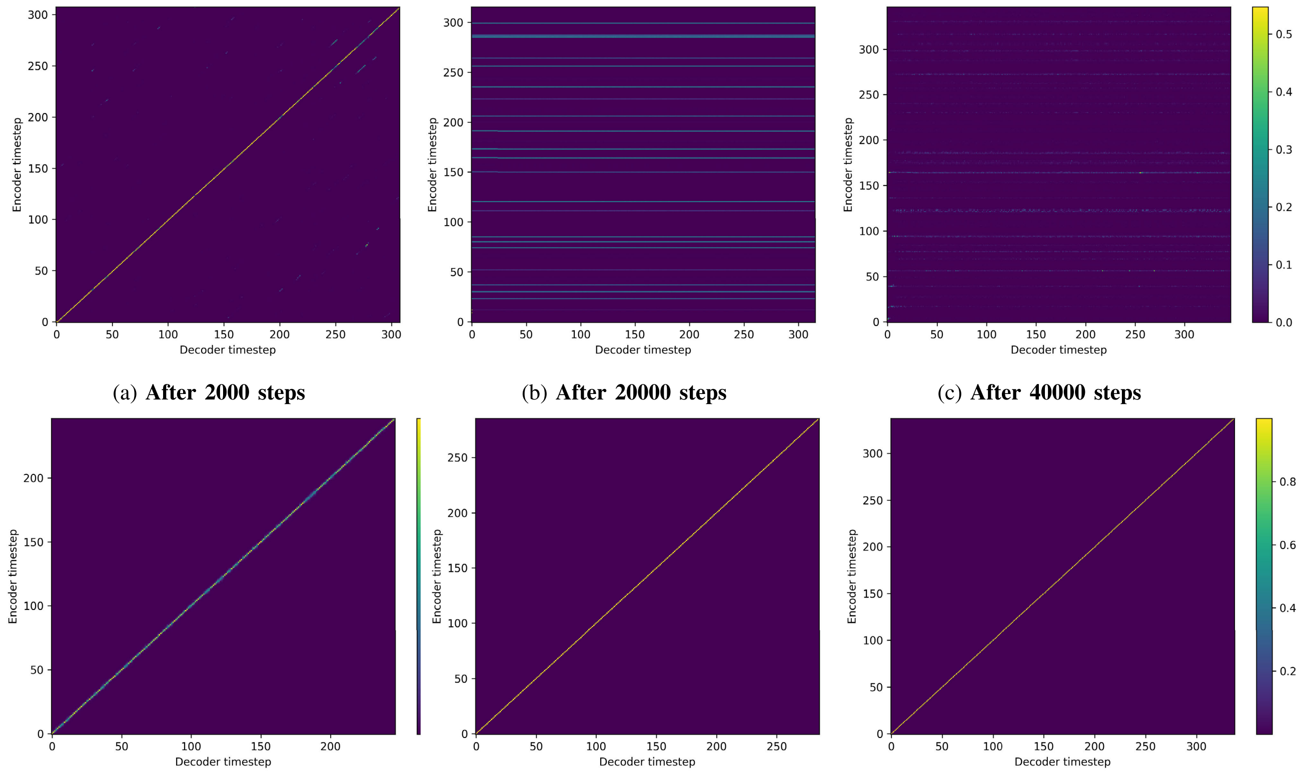


FIGURE 21. Confusion matrix of the CBHG model trained with the CA_MSA corpus and tested with the MSA test data. For each diacritic, we show the percentage in which it was predicted. The empty cells indicate that that the percentage is less than 0.01%.

is predicted. Figure 20 shows the confusion matrix of the CBHG model when trained using the CA_MSA corpus and tested using the CA test data. It shows that the model predicted no-diacritic with 100% accuracy, which is the most frequent option in the corpus. The diacritic Shadda (∼) has the lowest accuracy of 70%, where it was predicted as (Shadda + Fatha, ∼ a) for 18%, (Shadda + Damma, ∼ u) for 4%, and (Shadda + Kasra, ∼ i) for 4% of the predictions.



(a) **After 2000 steps** (b) **After 20000 steps** (c) **After 40000 steps**

(d) **After 1000 steps** (e) **After 20000 steps** (f) **After 40000 steps**

FIGURE 22. The alignments of the encoder-decoder model using the content-based attention [25] versus the location-based attention [24]. The content-based attention is visualized in a, b, and c. It learns the alignment after 2000 steps (a) but then forgets them in the future steps (b and c). The location-based attention, however, learns the attention faster after 1000 steps (d) and keeps improving the alignment in future steps (e and f).

When the model is tested with the MSA test data, it performed poorly for all diacritics. Figure 21 shows the confusion matrix of the CBHG model when trained with the CA_MSA corpus and tested with the MSA corpus test data. Compared with Figure 20, the no-diacritic option has an accuracy of 96%, which means that the model predicted diacritics for many characters that should not be diacritized. Here, the Shadda diacritic also has the lowest accuracy, but it was able to score only 18% in this test. It is mostly confused with (Shadda + Fatha, ∼ a) for 40%, and (Shadda + Kasra, ∼ i) for 25% of the predictions.

The encoder-decoder model confusion matrix has the same pattern as the confusion matrix of the CBHG model. When trained with the CA_MSA corpus and tested with the CA corpus test data, it predicted no-diacritic with 100% accuracy. The Shadda was predicted with lowest accuracy of 38%, which is much worse than the CBHG model (70%). The Shadda is mostly predicted as (Shadda + Fatha, ∼ a) for 43% of the predictions.

One critical aspect of the encoder-decoder model is the attention mechanism. We first tried the content-based attention [25], but we find that the model learns the alignments and forgets them in future steps as shown in Figure 22 (a-c). When we used the location-based attention [24], we found that the model learns the alignment very fast (after 1k steps) and improves the alignment in future steps, as depicted in Figures 22 (d-f).

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed three deep learning models to recover Arabic text diacritics. While the encoder-decoder works for this problem and performs better than many systems in the literature, we found that the CBHG model trains much faster and achieves state-of-the-art results for all metrics. The results clearly show that there are significant differences in Modern Standard Arabic (MSA) and Classical Arabic (CA) language variations. These differences require a large amount of data from both variations, not as common in most corpora, including the Tashkeela corpus, where CA data are much more than the MSA data.

We consider that the most critical future work is to collect more diacritized MSA text, which will be significant for this work and other works in the literature. There are plenty of ways to improve the models, such as trying different hyper-parameters, changing the encoder-decoder model's attention mechanisms, and trying recent architecture such as the transformer language model.

REFERENCES

[1] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*. Doha, Qatar: Association Computational Linguistics, 2014, pp. 1724–1734. [Online]. Available: https://www.aclweb.org/anthology/D14-1179

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 2. Cambridge, MA, USA: MIT Press, 2014, pp. 3104–3112.

[3] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. Le, Y. Agiomyrgiannakis, R. Clark, and R. A. Saurous, ''Tacotron: Towards end-to-end speech synthesis,'' in *Proc. Interspeech*, Aug. 2017, pp. 4006–4010.
[4] Y. A. El-Imam, ''Phonetization of Arabic: Rules and algorithms,'' *Comput. Speech Lang.*, vol. 18, no. 4, pp. 339–373, Oct. 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0885230803000354
[5] K. Shaalan, ''Rule-based approach in Arabic natural language processing,'' *Int. J. Inf. Commun. Technol.*, vol. 3, no. 3, pp. 11–19, 2010.
[6] Y. Gal, ''An HMM approach to vowel restoration in Arabic and Hebrew,'' in *Proc.ACL-Workshop Comput. Approaches Semitic Lang.* Philadelphia, PA, USA: Association for Computational Linguistics, Jul. 2002, pp. 1–7. [Online]. Available: https://www.aclweb.org/anthology/W02-0504
[7] R. Nelken and S. M. Shieber, ''Arabic diacritization using weighted finite-state transducers,'' in *Proc. ACL Workshop Comput. Approaches Semitic Lang.* Stroudsburg, PA, USA: Association Computational Linguistics, 2005, pp. 79–86. [Online]. Available: http://dl.acm.org/citation.cfm?id=1621787.1621802
[8] T. Schlippe, T. Nguyen, and S. Vogel, ''Diacritization as a translation problem and as a sequence labeling problem,'' in *Proc. 8th Conf. Assoc. Mach. Transl. Amer.*, Waikiki, HI, USA, 2008, pp. 21–25. [Online]. Available: https://www.csl.uni-bremen.de/cms/images/documents/publications/AMTA-2008-Schlippe.pdf
[9] S. Ananthakrishnan, S. Bangalore, and S. S. Narayanan, ''Automatic diacritization of Arabic transcripts for automatic speech recognition,'' in *Proc. Int. Conf. Natural Lang. Process. (ICON)*, Kanpur, India, Dec. 2005, pp. 47–54.
[10] M. Elshafei, H. Al-Muhtaseb, and M. Alghamdi, ''Statistical methods for automatic diacritization of Arabic text,'' in *Proc. Saudi 18th Nat. Comput. Conf.*, Riyadh, Saudi Arabia, vol. 18, 2006, pp. 301–306.
[11] M. Alghamdi and Z. Muzafar, ''Kacst Arabic diacritizer,'' in *Proc. 1st Int. Symp. Comput. Arabic Lang.*, Riyadh, Saudi Arabia, 2007, pp. 1–8. [Online]. Available: https://www.researchgate.net/publication/242693866_KACST_Arabic_Diacritizer
[12] Y. Hifny, ''Smoothing techniques for Arabic diacritics restoration,'' in *Proc. 12th Conf. Lang. Eng. (ESOLEC)*, Egypt, 2012, pp. 6–12.
[13] I. Zitouni, J. S. Sorensen, and R. Sarikaya, ''Maximum entropy based restoration of Arabic diacritics,'' in *Proc. 21st Int. Conf. Comput. Linguistics 44th Annu. Meeting ACL* Stroudsburg, PA, USA: Association Computational Linguistics, 2006, pp. 577–584, doi: 10.3115/1220175.1220248.
[14] N. Habash and O. Rambow, ''Arabic diacritization through full morphological tagging,'' in *Proc. Hum. Lang. Technol., Conf. North Amer. Chapter*. Rochester, NY, USA: Association Computational Linguistics, Apr. 2007, pp. 53–56. [Online]. Available: https://www.aclweb.org/anthology/N07-2014
[15] K. Shaalan, H. M. Abo Bakr, and I. Ziedan, ''A hybrid approach for building Arabic diacritizer,'' in *Proc. EACL Workshop Comput. Approaches Semitic Lang.* Athens, Greece: Association Computational Linguistics, Mar. 2009, pp. 27–35. [Online]. Available: https://www.aclweb.org/anthology/W09-0804
[16] A. S. Metwally, M. A. Rashwan, and A. F. Atiya, ''A multi-layered approach for Arabic text diacritization,'' in *Proc. IEEE Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Jul. 2016, pp. 389–393.
[17] A. Chennoufi and A. Mazroui, ''Morphological, syntactic and diacritics rules for automatic diacritization of Arabic sentences,'' *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 29, no. 2, pp. 156–163, Apr. 2017.
[18] K. Darwish, H. Mubarak, and A. Abdelali, ''Arabic diacritization: Stats, rules, and hacks,'' in *Proc. 3rd Arabic Natural Lang. Process. Workshop*. Valencia, Spain: Association Computational Linguistics, Apr. 2017, pp. 9–17. [Online]. Available: https://www.aclweb.org/anthology/W17-1302
[19] A. Al Sallab, M. Rashwan, H. M. Raafat, and A. Rafea, ''Automatic Arabic diacritics restoration based on deep nets,'' in *Proc. EMNLP Workshop Arabic Natural Lang. Process. (ANLP)*. Doha, Qatar: Association Computational Linguistics, Oct. 2014, pp. 65–72. [Online]. Available: https://www.aclweb.org/anthology/W14-3608
[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,'' *Int. J. Document Anal. Recognit.*, vol. 18, no. 2, pp. 183–197, Jun. 2015.
[21] Y. Belinkov and J. Glass, ''Arabic diacritization with recurrent neural networks,'' in *Proc. Conf. Empirical Methods Natural Lang. Process.* Lisbon, Portugal: Association Computational Linguistics, Sep. 2015, pp. 2281–2285. [Online]. Available: https://www.aclweb.org/anthology/D15-1274
[22] 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, Nov. 2019, pp. 215–225. [Online]. Available: https://www.aclweb.org/anthology/D19-5229
[23] H. Mubarak, A. Abdelali, H. Sajjad, Y. Samih, and K. Darwish, ''Highly effective Arabic diacritization using sequence to sequence modeling,'' in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1. Minneapolis, MN, USA: Association Computational Linguistics, Jun. 2019, pp. 2390–2395. [Online]. Available: https://www.aclweb.org/anthology/N19-1248
[24] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, ''Attention-based models for speech recognition,'' in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 1. Cambridge, MA, USA: MIT Press, 2015, pp. 577–585.
[25] D. Bahdanau, K. Cho, and Y. Bengio, ''Neural machine translation by jointly learning to align and translate,'' in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds. San Diego, CA, USA, May 2015, pp. 1–15. [Online]. Available: https://arxiv.org/pdf/1409.0473.pdf
[26] A. Al-Thubaity, A. Alkhalifa, A. Almuhareb, and W. Alsanie, ''Arabic diacritization using bidirectional long short-term memory neural networks with conditional random fields,'' *IEEE Access*, vol. 8, pp. 154984–154996, 2020.
[27] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, R. A. Saurous, Y. Agiomvrgiannakis, and Y. Wu, ''Natural TTS synthesis by conditioning wavenet on MEL spectrogram predictions,'' in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 4779–4783.
[28] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, ''Show, attend and tell: Neural image caption generation with visual attention,'' in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, 2015, pp. 2048–2057.
[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, ''Learning representations by back-propagating errors,'' *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.
[30] P. J. Werbos, ''Backpropagation through time: What it does and how to do it,'' *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[31] S. Hochreiter and J. J. Schmidhuber, ''Long short-term memory,'' *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[32] R. Pascanu, T. Mikolov, and Y. Bengio, ''On the difficulty of training recurrent neural networks,'' in *Proc. 30th Int. Conf. Mach. Learn.*, vol. 28, 2013, pp. III–1310–III–1318.
[33] M. Schuster and K. K. Paliwal, ''Bidirectional recurrent neural networks,'' *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.
[34] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, ''Distributed representations of words and phrases and their compositionality,'' in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, vol. 2. Red Hook, NY, USA: Curran Associates Inc., 2013, pp. 3111–3119.
[35] J. Pennington, R. Socher, and C. D. Manning, ''GloVe: Global vectors for word representation,'' in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*. Doha, Qatar: Association Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: https://www.aclweb.org/anthology/D14-1162
[36] R. K. Srivastava, K. Greff, and J. Schmidhuber, ''Training very deep networks,'' in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 2. Cambridge, MA, USA: MIT Press, 2015, pp. 2377–2385.
[37] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, ''On the properties of neural machine translation: Encoder–decoder approaches,'' in *Proc. SSST 8th Workshop Syntax, Semantics Struct. Stat. Transl.* Doha, Qatar: Association Computational Linguistics, Oct. 2014, pp. 103–111. [Online]. Available: https://www.aclweb.org/anthology/W14-4012
[38] J. Lee, K. Cho, and T. Hofmann, ''Fully character-level neural machine translation without explicit segmentation,'' *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 365–378, Dec. 2017. [Online]. Available: https://www.aclweb.org/anthology/Q17-1026
[39] 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.
[40] A. Farghaly and K. Shaalan, ''Arabic natural language processing: Challenges and solutions,'' *ACM Trans. Asian Lang. Inf. Process.*, vol. 8, no. 4, pp. 1–22, Dec. 2009.
[41] D. P. Kingma and J. Ba, ''Adam: A method for stochastic optimization,'' in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds. Ithaca, NY, USA: arXiv.org, May 2015, pp. 1–15. [Online]. Available: http://arxiv.org/abs/1412.6980

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 6000–6010.

MOKTHAR ALI HASAN MADHFAR received the bachelor's degree in computer science from Qassim University, Saudi Arabia, in 2015, where he is currently pursuing the master's degree. Along with his master's study, he works as a Programmer and Data Analyst with Qassim University. His research interests include data analysis and deep learning.

ALI MUSTAFA QAMAR received the B.E. degree (Hons.) in computer software engineering from the National University of Sciences & Technology, Islamabad, Pakistan, in 2005, the M.S. degree in computer science from University Joseph Fourier (UJF), Grenoble, France, in 2007, and the Ph.D. degree in computer science from the University of Grenoble, France, in 2010. He worked as a Temporary Assistant Professor with UJF from Nov 2010 till June 2011. He has been an Assistant Professor of computer science with the College of Computer, Qassim University, Buraydah, Saudi Arabia, since 2014. He has also been an Assistant Professor of computer science with the School of Electrical Engineering and Computer Science, National University of Sciences & Technology, since 2011. His research interests include machine learning, data mining, deep learning, social networks, and information filtering.