A Comparison of Neural Networks Architectures for Diacritics Restoration

Eduard Klyshinsky\textsuperscript{1(✉)}, Olesya Karpik\textsuperscript{2}, and Alexander Bondarenko\textsuperscript{3}

\textsuperscript{1} National Research University Higher School of Economics, S. Basmanna Street, 21, Moscow 105066, Russia
klyshinsky@mail.ru
\textsuperscript{2} Keldysh Institute of Applied Mathematics, Miusskaya sq., 4, Moscow 125047, Russia
parlak@mail.ru
\textsuperscript{3} State Research Institute of Aviation Systems, Viktorenko Street, 7, Moscow 125319, Russia
cod@gosniias.ru

Abstract. Neural networks are widely used for the task of diacritics restoration last years. Authors use different architectures of neural network for selected languages. In this paper, we demonstrated that an architecture should be selected according to a language in hand. It also depends on a task one states: low and full resourced languages could use different architectures. We demonstrated that common used accuracy metric should be changed in this task to precision and recall due to the heavy unbalanced nature of the input data. The paper contains results for seven languages: Croatian, Slovak, Romanian, French, German, Latvian, and Turkish.

Keywords: Diacritics restoration · Neural Networks · Out-of-vocabulary words

1 Introduction

Let us consider an extended alphabet that contains characters with diacritics, e.g. the extended Latin or Cyrillic alphabet. Every character with a diacritical mark could be matched with a character without such a mark; e.g. \v{A}→A. In this paper, we do not consider the substitution of a single character with a diacritical mark for several characters, e.g. \v{A}→AE. Let us substitute all characters with diacritics for their corresponding characters without diacritics. After such replacements we could state the task of diacritics restoration: consider the resulting text and restore the diacritics only in those positions where they were omitted.

Originally, the task of diacritics restoration was set for a natural language text. Finding a way to automate the input of diacritics is necessary not only for old valuable texts stored in an electronic format, but also for modern electronic texts, since they continue to be created in non-diacritical form. For some
reasons, such as the lack of keyboards with diacritics, OCR mistakes, ergonomic factors, etc., the diacritics are omitted in a text, and they must be restored. Such situation is typical not only for languages that use the Latin script, but also for Cyrillic and Arabic writing.

Recently, this task has been solved by machine learning methods. Preparing a data set could be done in a following easy way. On the first step, one constructs a set of replacements for a given language. For example, the letter A with different diacritics (ÁÅÁÁÁÁ) will be replaced with the same letter – A. To create training and test sets, one could replace all characters with diacritics in a text. In order to create a test set, one should tag positions of proper characters without diacritics by corresponding classes of diacritical marks. The set of tagged words could be shuffled and separated into the training and test sets.

In such case, the most frequent words have a chance of occurrence in both the training and test sets. Therefore, a model is both trained and tested on the highly intersected sets of words, and the probability of overlap is obviously high. As a result, the algorithm could ‘study answers’ but not ‘generalize data’. Such approach to the construction of training and test sets is similar to diacritics restoration using a dictionary.

The motivation of this paper is the following. According to the ICAO standard specification, all proper names in formal documents should be presented without diacritics. There is no complete dictionary of proper names for all languages yet; therefore this dictionary could not be used to restore diacritics in proper names. There is the same problem with low resourced languages. So, the more correct way to model such situation is to divide a vocabulary of a text and generate non-intersecting training and test sets. In this paper, we investigate the difference between these two approaches and compare several different models based on neural networks.

The rest of the paper is organized as following. In Sect. 2, we make a brief overview of current approaches to restoration of diacritics. In Sect. 3, we describe our data preparation method and machine learning methods used. Section 4 presents our experimental results and their brief discussion. Finally, Sect. 5 concludes this paper with a summary of outcomes.

2 Overview

One of the first papers in the area of diacritics restoration are Yarowsky [1] and Tufiş [2]. The main method used in these papers is searching a word in a dictionary of a given language. The word from a text was compared with the dictionary regardless of the accentuation. However, such approach could not be applied to a language that has majority of accentuated words. For example, in German the diacritical mark can be an indicator of the plural form (der Vater – die Väter). In Vietnamese and Igbo, there are words that differ in diacritics only, but have extremely different meanings. So, for example, the paper [3] reports the following chain:

ákwá (cloth), ákwà (bed/bridge), ákwá (cry), ákwá (egg).
The same problem is presented in the Arabic script, where diacritical marks are mostly omitted despite of having a semantic meaning.

Due to lack of accuracy, the described methods were replaced by machine learning methods. For example, Pauw [8] and Schlippe [4] propose to use character n-grams. The former article used the CRF method to determine the correct n-gram for the current position. The reported quality of this method is about 85\%–95\%.

The development of neural networks shifted the focus of investigations. The state-of-the-art implementations are using two architectures of neural networks - recurrent (RNN) and convolutional (CNN) neural networks. The results of both architectures are comparable. The article [5] reports accuracy metrics for twelve languages as high as 97–99\%. The authors used a network constructed using BiLSTM units with input embedding layer. The authors of [6] used Acausal Temporal Convolutional Neural Network with two hidden layers; precision of diacritics restoration reported as 96.2\% for the Yoruba language, 97\% for Arabic, and 97.5\% for the Vietnamese language. The difference between results of CNN and RNN reported less than 0.3\%. The article [7] uses CNN with an attention layer. Note that all authors use a single neural network to restore all diacritical marks in a given language. All neural networks accept as an input the vector of the character n-gram in various forms.

As it was mentioned in the previous section, low resourced languages have a problem with training a neural network on a small data set. The article [6] demonstrates that though the precision of a neural network learned and tested on the same dictionary is about 96–98\%, the precision of the same network for out-of-vocabulary words dramatically decreased to 70–85\% and even less than 50\%. Most of the authors (e.g. [3, 5, 7]) do not consider the class of out-of-vocabulary words at all. It should be noted that the accuracy metrics could not be used for such a project, since there is a disbalance between amount of words with and without diacritical marks. As it shown in [5], the amount of words with diacritics in different languages varies from 8\% in German to 88\% in Vietnamese. Our experience demonstrates that the relation between amount of occurrences of the same character, with and without diacritical marks, could be correlated as 1:100. In such conditions, the accuracy metric is useless, and one should prefer the precision metric.

According to the written above, we can state that most of the current methods of diacritics restoration have problems with processing of out-of-vocabulary words. To use machine learning methods, it is more correct to create non-intersected training and test sets. The precision and recall metrics should be prioritized for evaluation of the achieved results. Convolutional and recurrent neural networks demonstrate almost the same results in the same language, but these results differ among languages. Therefore, the aim of this article is to investigate the dependence of the neural network architecture on the selected language and the type of test and training sets. Moreover, we want to test a new approach to constructing a neural network solver, where instead of training single neural network that recognizes any character, we train a set of binary solvers where one network is responsible for only one character with a given diacritic.
3 Experimental Setup

This section discusses the used dataset, the data preparation method, and used architectures of neural networks.

3.1 Used Data and Data Preparation

In this paper, we used several news collections. French, German, and Turkish collections were downloaded from the news wire sites. Collections for Croatian, Latvian, Romanian, and Slovak languages were downloaded from the corpus presented in [5] (http://hdl.handle.net/11234/1-2607). The size of the former ranges from 63 to 217 mln of characters, the latter were cut to the first 200 mln characters.

We constructed a list of character replacements with diacritics to their ASCII counterparts for each language. We divided the collection into words not shorter than 4 characters; each word was aligned to the left and right side with three alignment symbols. Then we extracted all 7-grams with the character from the replacements list in the central position\(^1\). The 7-gram was tagged depending on whether the symbol in the central position had a diacritical mark or not. Finally, we replaced all the characters with diacritics with their counterparts.

For example, the Turkish word ‘¨oncesi’ is aligned with three spaces in the starting and final positions: ‘¨oncesi’. Then the algorithm extracts all 7-grams containing an accentuated symbol or its counterpart in the central position; in our case the resulting list is [‘¨once’, ‘¨oncesi’, ‘ncesı’, ‘cesi’]. Finally, symbols with diacritics are replaced by their counterparts, 7-grams are tagged by 0 (no diacritics) and 1 (with diacritics) according to the symbol in the central position: [(‘once’, 1), (‘oncesi’, 0), (‘ncesı’, 0), (‘cesı’, 0)].

We used two types of character embedding. In the first case, we replace the symbol with its position in the alphabet of the given language (dense representation); in the second case, we used one-hot encoding for this alphabet (one-hot representation). The alphabet was shaped automatically by the first million characters in the collection after word tokens were extracted.

3.2 Used Architectures

Unlike other authors, we constructed one binary classifier for each accentuated character in a given language. Such approach can be directly applied to languages like German, which has only three accentuated characters. Therefore, one could just extract all 7-grams with counterpart symbols and pass them to the proper classifier, which takes decision whether the diacritic mark should be placed here. For languages such as French, Romanian, and Latvian, which have several variant of diacritics for the same symbols, our approach should be

\(^{1}\) We extracted 5-grams in our preliminary experiments, but achieved remarkably worse results not published here.
extended to the second step. A 7-gram with a candidate character is passed to several classifiers connected to the given character. After classification, we should find the maximum among classifiers’ answers and make a conclusion about the necessary diacritical mark or its elimination. In this paper, we also compared two approaches: training separate classifiers without taking decision about different diacritics for the same base character vs training one classifier for every character with diacritics.

According to the chosen approach, any neural network contains an output dense layer consisting of two neurons with SoftMax activation function.

We used three different neural network architectures: dense, convolutional, and recurrent\(^2\); we also used the Random Forest method as a baseline. As mentioned above, we used two types of input vectors in our experiments: a vector of seven character codes and one-hot encoded vector of \(7^*(\text{size of an alphabet})\) elements. The used architectures of neural networks are also listed in Table 1.

The dense network consists of three layers of 128 neurons with the ReLU activation function and an output layer of 2 neurons with the SoftMax function; the batch size was 2048. We experimented with the number of neurons and layers, but it decreased the results. We studied this type of networks for 7 character codes vectors only. The network converges in 30 epochs.

The convolutional network consists of two layers with 32 convolution matrices size of \(3 \times 1\), a dense layer with 128 neurons with the ReLU activation function, and an output layer of 2 neurons with the SoftMax function. The batch size was 512; the network converges in 10 epochs. We experimented with the number of neurons, convolution matrices, and their size but it decreased the results.

For the recurrent networks we used three configurations. The first one consists of a layer with 64 biLSTM units and an output layer of 2 neurons with the SoftMax function. The first layer of the second configuration consists of 128 biLSTM units. The third configuration had a dense input layer with 32 neurons and the ReLU activation function, 128 biLSTM units, and an output layer of 2 neurons with SoftMax function. The batch size was 512; all networks converge in 10 epochs.

Table 1. Used architectures of neural networks.

| Dense network | CNN | RNN |
|---------------|-----|-----|
| Output vector: | \((0, 1)^2\) | |
| SoftMax layer | SoftMax layer | SoftMax layer |
| Dense 128 ReLU | Dense 128 ReLU | 32/64/128 biLSTM |
| Dense 128 ReLU | 32 convolution \(3 \times 1\) | |
| Dense 128 ReLU | 32 convolution \(3 \times 1\) | Dense 128 ReLU (optional) |
| Input vector: | \((\text{charcode})^7\) | or \((0, 1)^{7*|\text{alphabet}|}\) |

\(^2\) Links to used data and source codes are placed at GitHub: https://github.com/klyshinsky/diacritics_restoration.
In order to investigate the influence of the intersection of training and test vocabularies, we conducted two series of experiments. For the first series, we separated the input set of 7-grams into two non-intersecting sets; the sum of the test set’s frequencies was approximately 20% of the total frequency sum. Thus, in this series we trained and tested a classifier on different vocabularies and simulated the situation of out-of-vocabulary words. The second series of experiments tested only convolutional and recurrent networks, since they shown better productivity. In this series, we simply shuffled the set of 7-grams and split the input set into training and test in the ratio 80%–20%. Our experiments demonstrated that 80% of the words of the test set could be found in the training set.

4 Results of Experiments

The results of our experiments are shown in the Tables 2, 3, 4, 5, 6, 8. We did not use Word Error Rate (WER) and Diacritics Error rate (DER) since they are introduced as accuracy calculated for words and single diacritics, respectively. Instead of these metrics, we report precision and recall. As shown below, they evaluate results of the methods more correctly.

The tables are organized as following. Each table consists of results for a single language. We measured the results separately for every character with a diacritic mark. The first line for every character contains precision; the second line reports a recall. Figures marked with the bold font demonstrate maximal precision for the character among all results for separated training and test vocabularies. Figures marked with the underlined font demonstrate maximal precision for the character among all results for mixed training and test vocabularies. The last two rows in each table demonstrate the average precision and recall for the given classifier. Note that the best precision does not mean the best F1-value, but we did not calculate the F1-value due to the big size of resulting tables. Mostly, it is the case of low recall; however, such classifiers are not practically applicable. In the case of a high recall and a small difference between the values, both classifiers can be applied and should be compared by other features. In this article, we tried to draw the overall picture, but not to compare different neural networks in detail.

In the case of zero precision and recall, our system either failed to randomly select any 7-grams with a diacritic mark because of lack of words with diacritics, either evaluate all 7-grams as ones without diacritics due to the huge disbalance between classes. In case of Romanian ţ, it was true for every experimental run. For the French È, classes was unbalanced about 1:50. Its precision and recall were equal to zero but the accuracy was equal to 0.98. Thus, the accuracy metrics do not demonstrate the real situation here. That is why we used both precision and recall in this project.

\[3\] We want to thank Google Colab for the provided GPU time. It was not enough for all our calculations, but it was very useful.
Table 2. Results of experiments for the Croatian language (precision, recall). Bold script indicates the best solution for separated train and test, underline - for mixed ones.

| Characters | Separated train and test sets | Mixed train and test sets |
|------------|-------------------------------|--------------------------|
|            | 7 character input vector      | One-hot encoded          |
|            | Rand Forest                   | LSTM x128                |
|            | Dense 3 × 128                 | LSTM x64                 |
|            | Conv 2 × 32                   | LSTM x128                |
|            | Conv 2 × 32                   | One-hot encoded          |
|            | 7 character input vector      |                          |
|            | 3                             | 4                        |
|            | 4                             | 5                        |
|            | 5                             | 6                        |
|            | 6                             | 7                        |
|            | 7                             | 8                        |
|            | 8                             | 9                        |
| D          | 0.163/ 0.185/ 0.556/ 0.186/ 0.709/ 0.581/ 0.802/ 0.973/ 0.963/ 0.998/ | 0.195/ 0.924/ 0.936/ 0.970/ |
| C          | 0.522/ 0.721/ 0.809/ 0.841/ 0.865/ 0.798/ 0.972/ 0.971/ 0.989/ | 0.376/ 0.886/ 0.954/ 0.937/ 0.984/ |
| Č           | 0.694/ 0.649/ 0.850/ 0.892/ 0.860/ 0.831/ 0.881/ 0.969/ 0.972/ 0.968/ | 0.462/ 0.072/ 0.584/ 0.608/ 0.581/ 0.758/ 0.715/ 0.886/ 0.954/ 0.937/ |
| Š           | 0.365/ 0.573/ 0.450/ 0.504/ 0.696/ 0.613/ 0.659/ 0.960/ 0.931/ 0.944/ | 0.297/ 0.185/ 0.560/ 0.530/ 0.560/ 0.934/ 0.936/ 0.876/ 0.977/ |
| Z           | 0.638/ 0.686/ 0.851/ 0.770/ 0.911/ 0.626/ 0.840/ 0.978/ 0.936/ 0.967/ | 0.589/ 0.072/ 0.560/ 0.470/ 0.485/ 0.702/ 0.851/ 0.964/ 0.982/ 0.990/ |
| avg         | 0.476/ 0.727/ 0.772/ 0.729/ 0.810/ 0.726/ 0.816/ 0.970/ 0.954/ 0.973/ | 0.383/ 0.072/ 0.587/ 0.665/ 0.636/ 0.726/ 0.588/ 0.794/ 0.951/ 0.949/ 0.959/ |

Table 3. Results of experiments for the German language (precision, recall). Column names are presented in Table 2. Bold script indicates the best solution for separated train and test, underline - for mixed ones.

| Characters | Separated train and test sets | Mixed train and test sets |
|------------|-------------------------------|--------------------------|
|            | 7 character input vector      | One-hot encoded          |
|            | Rand Forest                   | LSTM x128                |
|            | Dense 3 × 128                 | LSTM x64                 |
|            | Conv 2 × 32                   | LSTM x128                |
|            | Conv 2 × 32                   | One-hot encoded          |
|            | 7 character input vector      |                          |
|            | 3                             | 4                        |
|            | 4                             | 5                        |
|            | 5                             | 6                        |
|            | 6                             | 7                        |
|            | 7                             | 8                        |
|            | 8                             | 9                        |
| A          | 0.065/ 0.219/ 0.338/ 0.278/ 0.314/ 0.270/ 0.139/ 0.916/ 0.899/ 0.655/ | 0.142/ 0.072/ 0.560/ 0.470/ 0.485/ 0.702/ 0.851/ 0.964/ 0.982/ 0.990/ |
| Ó           | 0.039/ 0.290/ 0.218/ 0.241/ 0.168/ 0.169/ 0.957/ 0.917/ 0.889/ 0.860/ | 0.072/ 0.072/ 0.560/ 0.470/ 0.485/ 0.702/ 0.851/ 0.964/ 0.982/ 0.990/ |
| Ü           | 0.704/ 0.472/ 0.577/ 0.446/ 0.322/ 0.278/ 0.761/ 0.949/ 0.938/ 0.991/ | 0.509/ 0.072/ 0.560/ 0.470/ 0.485/ 0.702/ 0.851/ 0.964/ 0.982/ 0.990/ |
| avg         | 0.269/ 0.327/ 0.377/ 0.321/ 0.268/ 0.239/ 0.619/ 0.927/ 0.908/ 0.835/ | 0.241/ 0.320/ 0.369/ 0.253/ 0.157/ 0.346/ 0.430/ 0.826/ 0.836/ 0.733/ |

As we can see, there is no ‘silver bullet’ among the neural network architectures even for a single language. For German and Turkish, the preferred architecture is convolutional network over one-hot encoded input vector, if we consider the number of winning characters (2 of 3 and 4 of 5 respectively). Other languages do not demonstrate any preferences for this feature. In the case of consideration the average precision for the ML model, the same network wins for Croatian, German, and Turkish; but for Romanian, Slovak, and Latvian the leading model is LSTM with 64 units; the convolutional network with dense vectors wins for French. Note that training CNN with a one-hot-encoded input vector takes about 10–50 times more calculation time. For mixed training and test vocabularies, the biLSTM is a default best choice, it wins in 39 out of 56 cases.
Table 4. Results of experiments for the French language (precision, recall). Column names are presented in Table 2. Bold script indicates the best solution for separated train and test, underline - for mixed ones.

|     | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Å   | 0.541/0.0 | 0.0 | 0.865/0.0 | 0.222/0.0 | 0.0/0.0 | 0.0/0.992 | 0.986/0.986 | 0.010/0.986 | 0.781/0.821 | 0.061 |
| Å   | 0.050/0.135/0.052/0.1 | 0.0 | 0.0 | 0.0/0.0 | 0.0/0.347 | 0.0/0.996 | 0.0/0.0 | 0.0/0.0 | 0.950/0.0 |
| Ç   | 0.028/0.001 | 0.247 | 0.011/0.0 | 0.0/0.065 | 0.0/0.289 | 0.0 | 0.0 | 0.0 | 0.950/0.0 |
| È   | 0.595/0.356/0.837/0.446 | 0.110 | 0.564/0.0 | 0.0/0.699 | 0.254/0.868 | 0.0/0.997 | 0.0 | 0.0 | 0.0 | 0.985 |
| È   | 0.846/0.968/0.831/0.846 | 0.910 | 0.948/0.0 | 0.916/0.0 | 0.0/0.0 | 0.0/0.0 | 0.0 | 0.0 | 0.0 | 0.989/0.138 |
| È   | 0.298 | 0.379 | 0.869/0.0 | 0.604/0.423 | 0.385/0.0 | 0.0/0.958 | 0.0 | 0.0 | 0.0 | 0.986/0.603 |

Table 5. Results of experiments for the Romanian language (precision, recall). Column names are presented in Table 2. Bold script indicates the best solution for separated train and test, underline - for mixed ones.

|     | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Å   | 0.541/0.0 | 0.0 | 0.865/0.0 | 0.222/0.0 | 0.0/0.0 | 0.0/0.992 | 0.986/0.986 | 0.010/0.986 | 0.781/0.821 | 0.061 |
| Å   | 0.050/0.135/0.052/0.1 | 0.0 | 0.0 | 0.0/0.0 | 0.0/0.347 | 0.0/0.996 | 0.0/0.0 | 0.0/0.0 | 0.950/0.0 |
| Ç   | 0.028/0.001 | 0.247 | 0.011/0.0 | 0.0/0.065 | 0.0/0.289 | 0.0 | 0.0 | 0.0 | 0.950/0.0 |
| È   | 0.595/0.356/0.837/0.446 | 0.110 | 0.564/0.0 | 0.0/0.699 | 0.254/0.868 | 0.0/0.997 | 0.0 | 0.0 | 0.0 | 0.985 |
| È   | 0.846/0.968/0.831/0.846 | 0.910 | 0.948/0.0 | 0.916/0.0 | 0.0/0.0 | 0.0/0.0 | 0.0 | 0.0 | 0.0 | 0.989/0.138 |
| È   | 0.298 | 0.379 | 0.869/0.0 | 0.604/0.423 | 0.385/0.0 | 0.0/0.958 | 0.0 | 0.0 | 0.0 | 0.986/0.603 |

We compared our approach, training of separate neural networks for each character, to the common one, training one neural network for all the characters of a given language, using convolutional networks over separated training and
Table 6. Results of experiments for the Slovak language (precision, recall). Column names are presented in Table 2. Bold script indicates the best solution for separated train and test, underline - for mixed ones.

| Column | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
|--------|----|----|----|----|----|----|----|----|----|----|
| ĽA     | 0.329/0.492/0.541/0.604/0.525/0.408/0.930/0.910/0.645/0.722/0.843 |
| ĽA     | 0.719/0.362/0.904/0.714/0.843/0.919/0.988/0.945/0.388/0.962/0.258 |
| ĽE     | 0.191/0.481/0.535/0.514/0.499/0.591/0.500/0.873/0.876/0.684/0.854/0.682 |
| Ľi     | 0.188/0.572/0.125/0.402/0.347/0.543/0.513/0.941/0.905/0.497/0.836/0.953 |
| ĽO     | 0.038/0.290/0.344/0.662/0.963/0.618/0.309/0.985/0.981/0.0/0.884/0.834 |
| ĽU     | 0.313/0.375/0.674/0.557/0.519/0.569/0.751/0.948/0.927/0.819/0.884/0.795 |
| ĽY     | 0.328/0.705/0.692/0.530/0.753/0.792/0.917/0.886/0.894/0.921/0.766 |
| ĽÇ     | 0.640/0.849/0.718/0.828/0.849/0.786/0.877/0.979/0.888/0.957/0.962/0.980 |
| ĽÔ     | 0.038/0.300/0.112/0.291/0.647/0.144/0.902/0.817/0.948/0.0 |
| ĽI     | 0.033/0.619/0.0/0.794/0.538/0.525/0.387/0.437/0.965/0.959/0.940/0.993 |
| ĽD     | 0.151/0.0/0.204/0.023/0.689/0.009/0.797/0.829/0.964/0.959/0.940/0.974 |
| ĽL     | 0.425/0.687/0.351/0.876/0.783/0.967/0.0/0.981/0.981/0.988/0.999/0.974 |
| ĽN     | 0.556/0.471/0.758/0.654/0.510/0.089/0.755/0.950/0.974/0.950/0.974 |
| ĽR     | 0.100/0.877/0.567/0.632/0.794/0.671/0.715/0.964/0.903/0.801/0.993/0.935 |
| ĽS     | 0.132/0.339/0.290/0.241/0.250/0.470/0.913/0.944/0.958/0.935 |
| ĽT     | 0.218/0.311/0.360/0.507/0.322/0.609/0.358/0.970/0.892/0.850/0.905/0.776 |
| ĽZ     | 0.289/0.055/0.424/0.374/0.434/0.517/0.125/0.855/0.905/0.977 |
| average| 0.300/0.509/0.612/0.594/0.659/0.631/0.482/0.956/0.926/0.685/0.908/0.740 |

Test sets. In the former case, the output layer consists of a number of neurons with SoftMax activation function equal to the length of the replacement list containing symbols both with and without a diacritical mark. For the Turkish language the average precision over accentuated symbols was as low as 0.645 (the average accuracy for one-hot CNN for our approach was 0.93), for the French language was 0.029 (0.5 for our approach), for the German language - 0.377 (0.619 in our case), 0.491 for Croatian (0.816 in our case), etc. Thus, it is easy to see that our method demonstrates better results. However, it takes more computations, the training of one deep neural network takes up to six hours on...
Table 7. Results of experiments for the Latvian language (precision, recall). Column names are presented in Table 2. Bold script indicates the best solution for separated train and test, underline - for mixed ones.

|   | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
|---|----|----|----|----|----|----|----|----|----|----|
| A | 0.278/ | 0.367/ | 0.425/ | **0.461**/ | 0.385/ | 0.380/ | 0.425/ | 0.913/ | 0.900/ | 0.456/ |
|   | 0.313/ | 0.403/ | 0.490/ | **0.490**/ | 0.447/ | 0.519/ | 0.361/ | 0.829/ | 0.780/ | 0.418/ |
| Č | 0.154/ | 0.633/ | 0.271/ | 0.209/ | **0.948**/ | 0.508/ | 0.809/ | 0.964/ | 0.962/ | 0.821/ |
|   | 0.213/ | 0.422/ | 0.197/ | 0.205/ | **0.789**/ | 0.408/ | 0.443/ | 0.877/ | 0.899/ | 0.996/ |
| Ė | 0.524/ | 0.683/ | 0.710/ | 0.737/ | 0.650/ | 0.784/ | 0.791/ | **0.791**/ | 0.966/ | 0.951/ |
|   | 0.434/ | 0.689/ | 0.706/ | 0.709/ | 0.637/ | 0.777/ | 0.808/ | 0.936/ | 0.941/ | 0.902/ |
| Ğ | 0.892/ | 0.801/ | 0.930/ | 0.935/ | 0.896/ | 0.963/ | 0.911/ | 0.987/ | 0.989/ | **0.996**/ |
|   | 0.530/ | 0.625/ | 0.734/ | 0.487/ | 0.405/ | **0.651**/ | 0.946/ | 0.897/ | 0.981/ | 1.0 |
| Ī | 0.467/ | 0.705/ | 0.895/ | **0.905**/ | 0.874/ | 0.827/ | 0.472/ | 0.985/ | 0.967/ | 0.597/ |
|   | 0.439/ | 0.785/ | 0.828/ | **0.836**/ | 0.837/ | 0.818/ | 0.713/ | 0.973/ | 0.972/ | 0.823/ |
| Ķ | 0.529/ | 0.923/ | 0.724/ | 0.867/ | 0.809/ | 0.773/ | 0.943/ | 0.972/ | 0.963/ | **0.969**/ |
|   | 0.302/ | 0.526/ | 0.594/ | 0.736/ | 0.777/ | 0.703/ | **0.873**/ | 0.949/ | 0.947/ | 0.924/ |
| Ļ | 0.510/ | 0.734/ | 0.751/ | 0.733/ | **0.813**/ | 0.689/ | 0.510/ | 0.973/ | 0.945/ | 0.904/ |
|   | 0.309/ | 0.534/ | 0.482/ | 0.600/ | **0.534**/ | 0.464/ | 0.794/ | 0.930/ | 0.940/ | 0.590/ |
| ķ | 0.367/ | 0.737/ | **0.797**/ | 0.691/ | 0.793/ | 0.740/ | 0.726/ | 0.967/ | 0.950/ | 0.979/ |
|   | 0.264/ | 0.635/ | **0.598**/ | 0.629/ | 0.668/ | 0.538/ | 0.593/ | 0.927/ | 0.919/ | **0.962**/ |
| Ģ | 0.287/ | **0.828**/ | 0.732/ | 0.669/ | 0.780/ | 0.601/ | 0.648/ | 0.967/ | 0.956/ | 0.917/ |
|   | 0.594/ | **0.625**/ | 0.697/ | 0.699/ | 0.772/ | 0.656/ | 0.676/ | 0.935/ | 0.945/ | 0.786/ |
| Ĉ | 0.347/ | 0.596/ | 0.478/ | **0.660**/ | 0.638/ | 0.496/ | 0.648/ | 0.979/ | 0.969/ | 0.720/ |
|   | 0.319/ | 0.617/ | 0.340/ | **0.566**/ | 0.408/ | 0.414/ | 0.821/ | 0.956/ | 0.963/ | 0.797/ |
| Ž | 0.556/ | 0.749/ | 0.724/ | 0.737/ | 0.757/ | 0.861/ | **0.889**/ | 0.978/ | 0.968/ | 0.818/ |
|   | 0.482/ | 0.707/ | 0.631/ | 0.725/ | 0.700/ | 0.744/ | **0.886**/ | 0.938/ | 0.947/ | 0.711/ |
| avg | 0.446/ | 0.705/ | 0.676/ | 0.691/ | 0.758/ | 0.692/ | 0.706/ | 0.968/ | 0.956/ | 0.824/ |
|   | 0.381/ | 0.597/ | 0.572/ | 0.607/ | 0.634/ | 0.608/ | 0.719/ | 0.922/ | 0.930/ | 0.809/ |

Google Colab’s T4. As result, the training of all the models for languages like Slovak takes a whole day and should be done several times.

Considering the difference between results with and without intersection of training and test vocabularies, we can state that results for mixed vocabularies are almost always better than for vocabularies without intersection. It is always true for a dense 7-character input vector, except the French êA with obviously erroneous results for LSTM on separate dictionaries, the French Ô, and the German Õ. Probably, the last two cases can be explained by the lack of statistics, since we had trained neural networks 3–6 times on separate data sets, while for mixed data sets - only once. The average difference here for the RNN network is 0.28, and for CNN is 0.33. Mixed vocabularies demonstrated the best result with the average difference of 0.12. Thus, we can state that methods of investigation of low resource languages should be changed.

Note, that we considered words not shorter than 4 characters. Such languages as Vietnamese, Chinese, and Irish have shorter names and prefixes. Our experiments demonstrated that the introduced method lacks of productivity; we suppose that it is mostly improbable to restore the correct diacritics having just
Table 8. Results of experiments for the Turkish language (precision, recall). Column names are presented in Table 2. Bold script indicates the best solution for separated train and test, underline - for mixed ones.

|    | 0       | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9       |
|----|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| C  | 0.852/  | 0.776/  | **0.946**/ | 0.808/  | 0.457/  | 0.892/  | 0.910/  | 0.978/  | 0.984/  | 0.968/  |
| 0.694/ | 0.762/  | **0.708**/ | 0.652/  | 0.692/  | 0.857/  | 0.876/  | 0.970/  | 0.963/  | 0.946/  |
| O  | 0.880/  | 0.922/  | 0.923/  | 0.904/  | 0.0/    | 0.926/  | **0.976**/ | 0.991/  | 0.989/  | 0.948/  |
| 0.727/ | 0.611/  | 0.629/  | 0.929/  | 0.0/    | 0.959/  | **0.981**/ | 0.979/  | 0.992/  | 0.981/  |
| Ü  | 0.873/  | 0.809/  | 0.923/  | 0.932/  | 0.0/    | 0.919/  | **0.955**/ | 0.991/  | 0.990/  | 0.965/  |
| 0.812/ | 0.833/  | 0.826/  | 0.769/  | 0.0/    | 0.907/  | **0.853**/ | 0.978/  | 0.984/  | 0.853/  |
| G  | 0.974/  | 0.950/  | 0.973/  | 0.990/  | 0.727/  | 0.955/  | 0.953/  | **0.995**/ | 0.997/  | 0.993/  |
| 0.854/ | 0.992/  | 0.995/  | 0.986/  | 0.689/  | 0.994/  | 0.998/  | 0.998/  | **0.999**/ | 0.999/  | 0.996/  |
| Ș  | 0.699/  | 0.637/  | 0.809/  | 0.622/  | 0.451/  | 0.575/  | **0.859**/ | 0.961/  | 0.958/  | 0.790/  |
| 0.549/ | 0.730/  | 0.814/  | 0.713/  | 0.510/  | 0.486/  | **0.752**/ | 0.938/  | 0.921/  | 0.760/  |
| avg | 0.855/  | 0.818/  | 0.914/  | 0.851/  | 0.327/  | 0.853/  | 0.930/  | 0.983/  | 0.983/  | 0.932/  |
|    | 0.727/  | 0.785/  | 0.794/  | 0.809/  | 0.378/  | 0.840/  | 0.892/  | 0.972/  | 0.971/  | 0.907/  |

Moreover, we can not be completely sure that a neural network trained on news wire will correctly restore diacritics in proper names due to specificity of testing data.

Some characters, such as French ˆI, German Ä, Romanian Ș, demonstrate very poor results. The reasons of such behavior need to be investigated in an independent research, but we can presuppose that these characters could be found in the same context as their counterparts. For example, this is true for French I which is used to differentiate similarly sounding words. But this is not always the case of German Ä, since there are many words which do not have counterparts.

5 Conclusion

In this paper we demonstrated that there is no best neural network architecture for each considered language in the area of diacritics restoration. There are differences in the neural network design and their results in case of low and full resourced languages. For a full resourced language, the task is usually stated as searching for a word from the vocabulary and taking a decision about its diacritics. For a low resourced language, we could have in hand an out-of-vocabulary word. These tasks demand different approaches to their solution. In the former case, the slightly better results are achieved with the biLSTM networks; in the later case, the choice depends on the considered language.

We have investigated languages such as Croatian, Slovak, Romanian, French, German, Latvian, and Turkish. We found that there is a large difference between the results for the case of completely separated train and test vocabularies and the random distribution of tokens between these vocabularies. The average difference between precision for these two cases reaches 0.12–0.33.
We have presented a new approach based on several neural networks each trained for its own symbol. This approach wins over the previous one, where only one neural network is trained for all the symbols.

Almost always, there is a good option for any character with diacritics to be restored. However, the results for some characters are quite low. These are French Ê, German Â, and Romanian Ţ. For the last character, any machine training algorithm has decided that it is easier to attribute all the characters as S since there are not so many examples of words with Ţ; the same is true for many other cases. This means that we have to use other methods to form train and test sets and construct batches. The case of the German Â needs a linguistic investigation of this phenomenon.

References

1. Yarowsky, D.: Comparison of corpus-based techniques for restoring accents in Spanish and French text. Natural Language Processing using Very Large Corpora 1999, pp. 99–120. Springer, Dordrecht (1999)
2. Tufiş, D., Chiţu, A.: Automatic diacritics insertion in Romanian texts. In: Proceedings of COMPLEX 1999 International Conference on Computational Lexicography, Pecs, Hungary, pp. 185–194 (1999)
3. Ezeani, I., Hepple, M., Onyenwe, I.: Automatic restoration of diacritics for Igbo language. In: Sojka, P., Horák, A., Kopeček, I., Pala, K. (eds.) TSD 2016. LNCS (LNAI), vol. 9924, pp. 198–205. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45510-5_23
4. Schlippe, T., Nguyen, T.L., Vogel, S. E.: Diacritization as a machine translation problem and as a sequence labeling problem. In: Proceedings of the Eighth Conference of the Association for Machine Translation in the Americas, pp. 270–278 (2008)
5. Náplava, J., Straka, M., Straňák, P., Hajíč, J.: Diacritics restoration using neural networks. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), pp. 1566–1573 (2018)
6. Alqahtani, S., Mishra, A., Diab, M.: Efficient convolutional neural networks for diacritic restoration. In: Proceedings of the 9th International Joint Conference on Natural Language Processing, pp. 1442–1448 (2019)
7. Orife, I.: attentive sequence-to-sequence learning for diacritic restoration of Yorùbá language text. In: Proceedings of the INTERSPEECH 2018, pp. 2848–2852 (2018). https://doi.org/10.21437/Interspeech.2018-42
8. De Pauw, G., Wagacha, P.W., de Schryver, G.-M.: Automatic diacritic restoration for resource-scarce languages. In: Proceedings of the Text, Speech and Dialogue, 10th International Conference, pp. 170–179 (2007). https://doi.org/10.1007/978-3-540-74628-7_24