A Novel Word Spotting Algorithm Using Bidirectional Long Short-Term Memory Neural Networks

Volkmar Frinken, Andreas Fischer, and Horst Bunke

Institute of Computer Science and Applied Mathematics, University of Bern,
Neubrückstrasse 10, CH-3012 Bern, Switzerland
{frinken,afischer,bunke}@iam.unibe.ch

Abstract. Keyword spotting refers to the process of retrieving all instances of a given key word in a document. In the present paper, a novel keyword spotting system for handwritten documents is described. It is derived from a neural network based system for unconstrained handwriting recognition. As such it performs template-free spotting, i.e. it is not necessary for a keyword to appear in the training set. The keyword spotting is done using a modification of the CTC Token Passing algorithm. We demonstrate that such a system has the potential for high performance. For example, a precision of 95% at 50% recall is reached for the 4,000 most frequent words on the IAM offline handwriting database.

1 Introduction

The automatic recognition of handwritten text – such as letters, manuscripts or entire books – has been a focus of intensive research for several decades [1,2]. Yet the problem is far from being solved. Particularly in the field of unconstrained handwriting recognition where the writing styles of various writers must be dealt with, severe difficulties are encountered.

Making handwritten texts available for searching and browsing is of tremendous value. For example, one might be interested in finding all occurrences of the word “complain” in the letters a company receives. As another example, libraries all over the world store huge numbers of handwritten books that are of crucial importance for preserving the world's cultural heritage. Making these books available for searching and browsing would greatly help researchers and the public alike. Finally, it is worth mentioning that Google and Yahoo have announced to make handwritten books accessible through their search engines as well [3].

Transcribing the entire text of a handwritten document for searching is not only inefficient as far as computational costs are concerned, but it may also result in poor performance, since misrecognized words cannot be found. Therefore, techniques especially designed for the task of keyword spotting have been developed.

Current approaches to word spotting can be split into two categories, viz. query-by-example (QBE) and query-by string (QBS). With the former approach, all instances of the search word in the training set are compared with all word images in the test set. Among the most popular approaches in this category are dynamic time warping (DTW) [4,5,6] and classification using global features [7,8]. Word shape methods using Gradient, Structural and Concavity features (GSC) have been shown to outperform DTW in [9,10]. Algorithms based on QBE suffer from the drawback that they can only find words appearing in the training set. The latter approach of QBS models the key words according to single characters in the training set and searches for sequences of these characters in the test set. The approach proposed in [11,12] requires a character-position based ground truth for the training set. Consequently, not only bounding boxes around each word are required, but around each single character. In addition to expensive manual preprocessing, this imposes a problem since in cursive handwriting it is often not clear how to segment a word into individual characters.

The approach proposed in this paper uses a neural network based handwriting recognition system which has several advantages compared to the above mentioned approaches. First, by treating an entire text line at a time, it is not necessary to split the text lines of the test set into separate words. Secondly, being derived from a general neural network based handwritten text recognition system, any arbitrary string can be searched for, not just the words appearing in the training set. Thirdly, it is not required to have the bounding box of each word or character included in the training set. The ASCII transcription of the text lines in the training set is sufficient to train the neural network.

The rest of this paper is organized as follows. In Section 2, document preprocessing procedures and a neural network for handwritten text recognition are introduced. In Section 3, we describe how the proposed system can be adopted to the task of word spotting. An experimental evaluation of this system is presented in Section 4, and conclusions are drawn in Section 5.

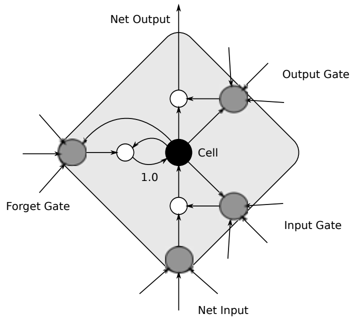
2 Neural Network Based Handwritten Text Recognizer

2.1 Preprocessing

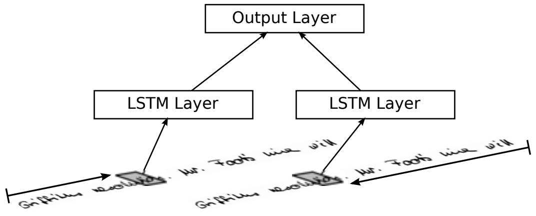
We follow the common approach to offline handwriting recognition by first automatically segmenting an input document into individual text lines. From each line, a sequence of feature vectors is extracted, which is then submitted to the neural network.

The words used in the experiments described in this paper come from the IAM database [13]. They are extracted from pages of handwritten texts, which were scanned and separated into individual text lines. After binarizing an image with a suitable threshold on the gray scale value, the slant and skew of each textline is corrected and the width and height of the handwriting are normalized [14].

Given the image of a single word, a horizontally sliding window with a width of one pixel is used to extract nine geometric features at each position from left to right, three global and six local ones. The global features are the 0^{th}, 1^{st} and 2^{nd} moment of the black pixels' distribution within the window. The local features are the position of the top-most and bottom-most black pixel, the inclination of



(a) The LSTM Node



(b) The entire BLSTM Neural Networks

Fig. 1. The schematics of the BLSTM Neural Network. LSTM memory cells (a) in two distinct recurrent layers process the text line from different directions.

the top and bottom contour of the word at that position, the number of vertical black/white transitions, and the average gray scale value between the top-most and bottom-most black pixel. For details on these steps, we refer to [14].

2.2 BLSTM Neural Network

The recognizer used in this paper is a recently developed recurrent neural network, termed *bidirectional long short-term memory* (BLSTM) neural network [15]. In general, recurrent NN offer a natural way for neural networks to process sequential data by reading the sequence one step at a time. Due to recurrent connections within the hidden layer, information from previous times steps can be accessed. Unfortunately, recurrent neural networks suffer from the *vanishing gradient problem*, which describes the exponential increase or decay of values as they cycle through recurrent network layers.

A way to circumvent this problem is the introduction of so-called *long short-term memory* blocks. In Fig. 1(a) such a LSTM node is displayed. At the core of the node, a simple cell, which is connected to itself with a recurrence multiplication factor of 1.0 stores the information. New information via the *Net Input* enters only if the *Input Gate* opens and leaves the cell into the network when the *Output Gate* is open. The activation of the *Forget Gate* resets the cell's value to 0. The gates and the *Net Input* are conventional nodes using an arctan activation function. This architecture admits changes to the cell's memory only when one of the gates is open and is therefore able to carry information across arbitrarily long sequence positions. Thus, at any point in the sequence, the usage of contextual information is not restricted to the direct neighborhood.

The input layer contains one node for each of the nine geometrical features and is connected with two distinct recurrent hidden layers. The hidden layers are both connected to the output layer. The network is *bidirectional*, i.e. a sequence of feature vectors is fed into the network in both forward and backward modes. One hidden layer deals with the forward sequence, and the other layer with the

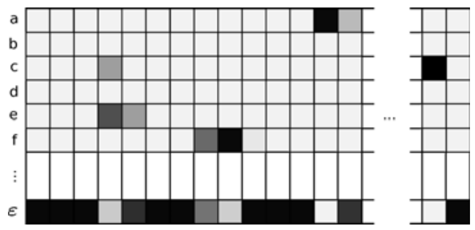


Fig. 2. The activation level for all nodes in the output layer. The activation is close to 0 most of the time for normal letters and peaks only at distinct position. In contrast, the activation level of the ε node is nearly constantly 1.

backward sequence. An illustration of the network can be seen in Fig. 1(b). At each position k of the input sequence of length t, the output layer sums up the values coming from the hidden layer that has processed positions 1 to k and the hidden layer that has processed positions t down to k. The output layer contains one node for each possible character in the sequence plus a special ε node, to indicate "no character". At each position, the output activations of the nodes are normalized so that they sum up to 1, and are treated as probabilities that the node's corresponding character occurs at this position. A visualization of the output activations along a text line can be seen in Fig. 2. For more details about BLSTM networks, we refer to [15,16].

The neural network produces a sequence of probabilities for each letter and each position in the text line. This sequence can be efficiently used for word and text line recognition as well as for word spotting as shown in the present paper, where the Connectionist Temporal Classification (CTC) Token Passing algorithm is utilized for the latter task.

3 Word Spotting Using BLSTM

The neural network described in the previous section produces a sequence of probabilities for each letter and each position in the text line. This sequence can be efficiently used for word and text line recognition [15] as well as for word spotting as shown in the present paper, where the Connectionist Temporal Classification (CTC) Token Passing algorithm is adapted to the latter task. To the knowledge of the authors, this is the first attempt to spot keywords using the CTC algorithm in conjunction with BLSTM neural networks.

3.1 CTC Token Passing Algorithm

The CTC Token Passing algorithm for single words expects a sequence of letter probabilities of length t as output by the neural network, together with a word w as a sequence of ASCII characters, and returns a matching score, i.e. the probability that the input to the neural network was indeed the given word (Algorithm 1).

Let $n(l,k)$ denote the probability of the letter l occurring at position k according to the neural network output[1] and let $w = l_1 l_2 \dots l_n$ denote the word to be matched. The algorithm first expands w into a sequence

$$w' = \varepsilon l_1 \varepsilon l_2 \dots \varepsilon l_n \varepsilon = c_1 c_2 c_3 \dots c_{2n+1}$$

and creates for every character c_i $(i = 1, \dots, 2n+1)$ and every position $j = 1, \dots, t$ in the text line a token $\vartheta(i,j)$ to store the probability that character c_i is present at position j together with the probability of the best path from the beginning to position j. The tokens are initialized so that their probability is 0 except for the first ε (c_1) and the first letter (c_2), which are initialized to hold the value of ε and the probability value of c_1 at the first position of the sequence, respectively (Lines 3–5).

During the following loop over all input sequence positions j, the tokens $\vartheta(\cdot,j)$ are updated, so that a) the token's corresponding letter l occurs at position j, b) in the best path, all letters of the word occur in the given order, c) between two subsequent letters of the word, only ε-node activations are considered and d) if two subsequent letters of a given word are the same (e.g. positions 3 and 4 in "Hello"), at least one ε node must lie between them. To compute the value of the token $\vartheta(i,j)$, a set $\mathbb{T}_{best}$ is created in which all valid tokens are stored that can act as predecessor to the token $\vartheta(i,j)$ according to the above mentioned constraints. If at sequence position j the letter c_i is considered (which might be a real letter or ε), the token corresponding to the same letter c_i at sequence position $j-1$ is valid (Line 9). The token corresponding to the letter c_{i-1} (ε if c_i is a real letter and a real letter if $c_i = \varepsilon$) at sequence position $j-1$ is valid for each but the first letter (Line 10 and 11). Since two different letters might follow each other without an ε-node activation, the token corresponding to the letter c_{i-2} is valid for these cases, too (Line 12 and 13). Afterwards, the probability of the best token in $\mathbb{T}_{best}$ is multiplied with $n(i,j)$ to give the probability of $\vartheta(i,j)$. Algorithm 1 is a slightly simplified version of the one given in [15] and only suitable for single word recognition, but it is sufficient for our task of keyword spotting.

The main contribution of this paper is the modification of Algorithm 1 to search for any given word in a text line s of arbitrary length. First, we add a virtual node to the output nodes, called the *any*- or $*$-node and set $\forall j \; n(*,j) = 1$. Then, assuming that the considered word actually occurs in the given text line, we distinguish three different cases: a) the word to be searched occurs at the beginning of the sequence, b) the word occurs at the end of the sequence, and c) the word occurs in the given text line, but neither at the end nor at the beginning. This distinction is important in view of whitespace characters $'\_'$ following or preceding the word to be searched.

Consider again the given word $w = l_1 l_2 \dots l_n$. To cope with case a), we append a whitespace character to the word, followed by the *any*-character. For case b)

[1] Due to our normalization procedure, the following statement holds $\forall k : \sum_l n(l,k) = 1$. Therefore, the output values of the neural network can be indeed considered as probabilities.

Algorithm 1. The CTC Token Passing Algorithm for single word recognition

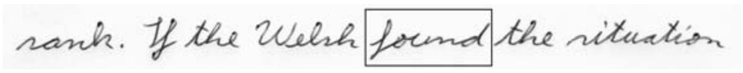
```
Require: input word $w = l_1 l_2 \dots l_n$
Require: sequence of letter probabilities, accessible via $n(\cdot,\cdot)$
 1: Initialization:
 2: expand $w$ to $w' = \varepsilon l_1 \varepsilon l_2 \varepsilon \dots \varepsilon l_n \varepsilon = c_1 c_2 \dots c_{2n+1}$
 3: $\vartheta(1,1) = n(\varepsilon, 1)$
 4: $\vartheta(2,1) = n(l_1, 1)$
 5:
 6: Main Loop:
 7: for all sequence positions $2 \leq j \leq t$ do
 8:     for all positions $i$ of the extended word $1 \leq i \leq 2n+1$ do
 9:         $\mathbb{T}_{best} = \{\vartheta(i, j-1)\}$
10:         if $i > 1$ then
11:             $\mathbb{T}_{best} = \mathbb{T}_{best} \cup \vartheta(i-1, j-1)$
12:             if $c_i \neq \varepsilon$ and $c_i \neq c_{i-2}$ then
13:                 $\mathbb{T}_{best} = \mathbb{T}_{best} \cup \vartheta(i-2, j-1)$
14:             end if
15:         end if
16:         $\vartheta(i,j) = \max(\mathbb{T}_{best}) \cdot n(i,j)$    ▷ multiply the best token's probability with
    the letter probability
17:     end for
18: end for
        return $\max\{\vartheta(2n+1, t),\ \vartheta(2n, t)\}$ ▷ The word can either end on the last $\varepsilon$
    ($c_{2n+1}$) or on the last regular letter ($c_{2n}$)
```

we prefix the word with the *any*-character and a whitespace. Finally, for case c), we prefix and append a whitespace and an *any*-character:

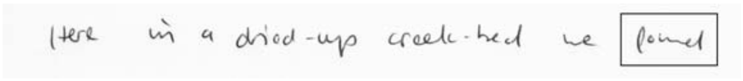
$$w_a = l_1 l_2 \dots l_n \_ *$$
$$w_b = * \_ l_1 l_2 \dots l_n$$
$$w_c = * \_ l_1 l_2 \dots l_n \_ *$$

If we now use the CTC-Algorithm for single word recognition to compute the probability of the word being w_a, we compute in fact the probability that the text line starts with the first letter of the word w, followed by the second letter, and so on until the word's last letter, followed by a whitespace and then by *any* character. Obviously, the size and content of the text following the whitespace after word w is irrelevant, since $n('*', j) = 1$. Similarly, if we run the CTC-Algorithm with the word w_b, we compute the probability that the textline ends with word w. If the CTC Token Passing algorithm is run with w_c, we get the probability that the word w occurs somewhere in the middle. We can now easily combine the output of the three runs of the algorithm with w_a, w_b and w_c by using the maximum

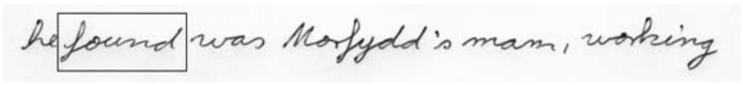
$$p_{CTC}(w|s) = \max\{p_{CTC}(w_a|s),\ p_{CTC}(w_b|s),\ p_{CTC}(w_c|s)\} \ .$$



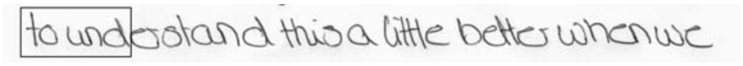
(a) Returned log Likelihood: -1.7125



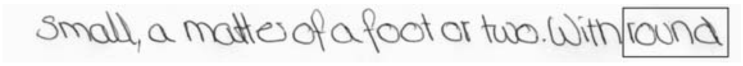
(b) Returned log Likelihood: -8.4097



(c) Returned log Likelihood: -9.0727



(d) Returned log Likelihood: -11.0900



(e) Returned log Likelihood: -11.6082

Fig. 3. Search results for the word "found"

Of course, the returned probability of a word still depends upon the word's length. To receive a normalized value which can then be thresholded, we divide $p_{CTC}(w|s)$ by the search word's length[2]

$$p'_{CTC}(w|s) = \frac{p_{CTC}(w|s)}{|w|} \ .$$

How the results of such a search may look like can be seen in Fig. 3. Note that the system just returns a likelihood of the word being found. Afterwards, this likelihood can be compared to a threshold to decide whether or not this is a true match.

4 Experimental Evaluation

4.1 Setup

For testing the proposed keyword spotting method, we used the IAM offline database[3]. This database consists of 1,539 pages of handwritten English text, written by 657 writers. From this database, we used 6,161 lines as a training set, 920 lines as a writer independent validation set, and an additional 920 lines as

[2] We define the length according to the number of letters.

[3] `http://www.iam.unibe.ch/fki/databases/iam-handwriting-database`

a test set. Using the training set, we trained seven randomly initialized neural networks and used the validation set to stop the back propagation iterations in the training process. See [15] for details on the neural network training algorithm. Then we selected the 4,000 most frequent words from all three sets and performed keyword spotting using these words. Note that by far not all the keywords used in the test occur in every set.

4.2 Results

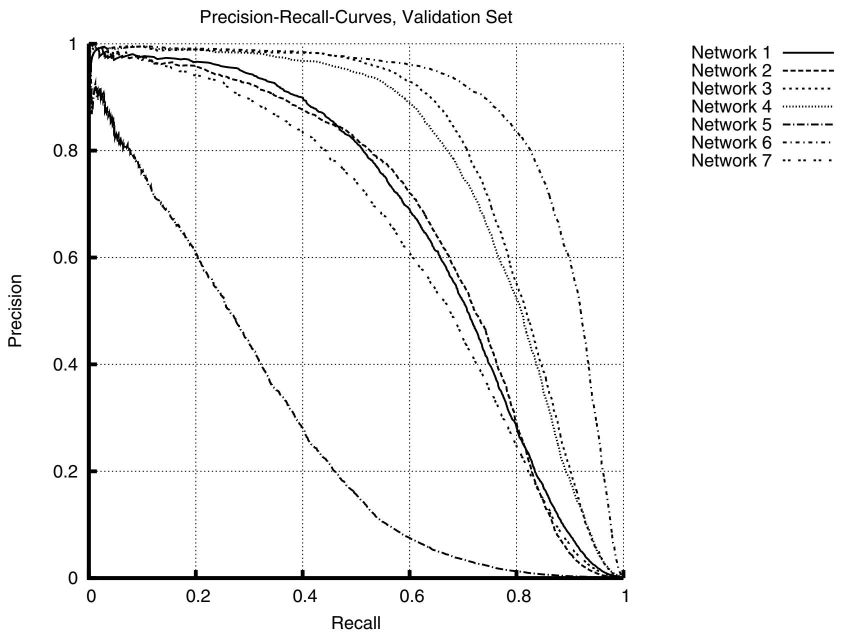
Every word w tested on each text line s returns a probability $p'_{CTC}(w|s)$. The word spotting algorithm compares this probability against a global threshold to decide whether or not it is a match. We used all returned values $p'_{CTC}(w|s)$ as a global threshold in oder to make the results as precise as possible. For each of these thresholds, we computed the number *true positives* (TP), *true negatives* (TN), *false positives* (FP), and *false negatives* (FN). These number were then used to plot a *precision-recall* curve for each neural network (see Fig. 4(a)). *Precision* is defined as number of relevant objects found by the algorithm divided by the number of all objects found $\frac{TP}{TP+FP}$, while *recall* is defined as the number of relevant objects found divided by the number of all relevant objects in the test set $\frac{TP}{TP+FN}$.

A precision-recall curve therefore gives us an idea about the noise in the returned results, given the percentage of how many true elements are found. It can be seen that the performance of the different networks varies greatly. The network that performed best on the validation set achieves an average precision rate of 87.6%. At a recall rate of 50%, the precision is 97.3%.

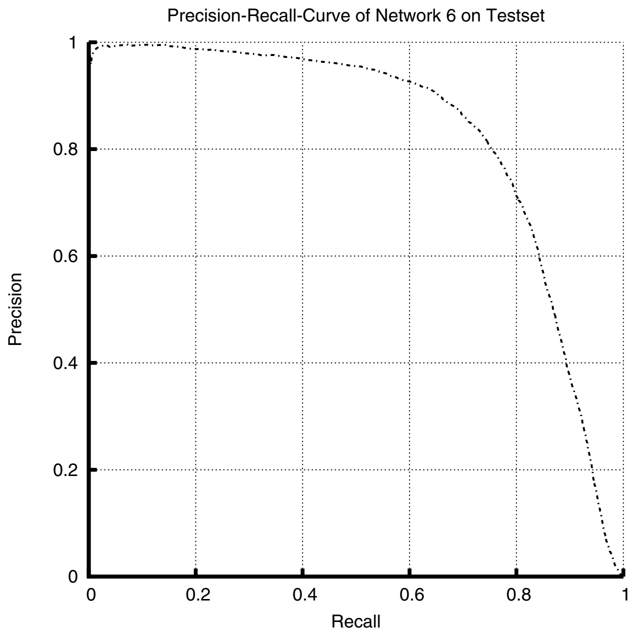
Based on the validation set, it is possible to pick good networks from the ensemble of all networks. The network that performed best (Network 6) was further analyzed. Its *precision-recall* curve on the independent test set is shown in Fig. 4(b). The average precision rate is 82.8% and its precision at a recall rate of 50% is 95.5%. This performance rivals those of the best existing systems, e.g. [9,8], although on a different, but not harder data set.

In Fig. 5, a *rank plot* is shown. A rank plot visualizes the quality of the search results. Each row corresponds to one keyword and dots correspond to positions of the right occurrences of keywords in the rank. Keywords are sorted according to the number of occurrences in the test set. Consequently, an optimal rank plot would have all the black dots on the left hand side of the plot, more black dots in the lower left hand corner than in the upper left hand corner, while the rest of the plot should be white.

The rank plot using all search words actually occurring in the test set can be seen in Fig 5. Although there are few black dots spread over the entire plot, it can be seen that it comes quite close to the ideal form. Since rank plots cannot easily be converted into a single number, it is not straight forward to compare different rank plots. However, they give a good impression about a system's performance. The keyword spotting ability of the proposed approach seems to be independent of the frequency of search word. Both common and uncommon



(a) Precision-Recall curves for seven different neural networks on the validation set.



(b) Precision-Recall curves on the test set for network 6.

Fig. 4. The Precision-Recall curves of the neural networks

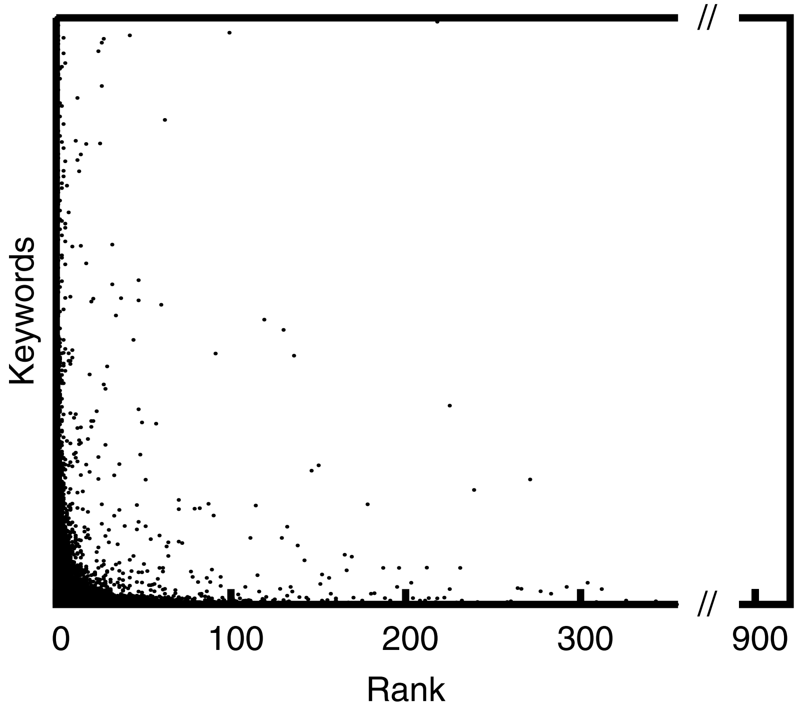


Fig. 5. The best network's rank plot for all search words that appear in the test set

words can be spotted since the black dots are aligned on the left side in the lower and upper part of the rank plot.

5 Conclusion

I this paper, we developed a word spotting algorithm that is derived from a recurrent neural network based handwriting recognition system. We were able to demonstrate that it not only performs very well with respect to the word spotting task, but it also overcomes some of the drawbacks of existing system. Not only words that appear in the training set can be searched for as in the query-by-example approach, but any character string. Secondly, to train the neural network, it is not necessary to have a bounding box for each character in the training set; just the correct transcription is needed. The performance varies greatly from one individual neural network to the other; however, an independent validation set can be used to identify the best performing network.

A future line of research will evaluate the proposed form of keyword spotting on historical documents. Desirable is also a direct experimental comparison to other available methods. Combining different neural networks to build one single improved system as well as modifying different handwriting recognition systems, such as HMM based recognizers for the task of keyword spotting, are further aspects worth to be considered in future research.

Acknowledgments

This work has been supported by the Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (IM2) and the the Swiss National Science Foundation (Project CRSI22_125220/1). We thank Alex Graves for kindly providing us with the BLSTM Neural Network source code.

References

1. Vinciarelli, A.: A Survey On Off-Line Cursive Word Recognition. Pattern Recognition 35(7), 1433–1446 (2002)
2. Plamondon, R., Srihari, S.N.: On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. IEEE Transaction on Pattern Analysis and Machine Intelligence 22(1), 63–84 (2000)
3. Levy, S.: Google's two revolutions, Newsweek (December 27/January 3, 2004)
4. Kołcz, A., Alspector, J., Augusteijn, M.F., Carlson, R., Popescu, G.V.: A Line-Oriented Approach to Word Spotting in Handwritten Documents. Pattern Analysis and Applications 3, 153–168 (2000)
5. Manmatha, R., Rath, T.M.: Indexing of Handwritten Historical Documents - Recent Progress. In: Symposium on Document Image Understanding Technology, pp. 77–85 (2003)
6. Rath, T.M., Manmatha, R.: Word Image Matching Using Dynamic Time Warping. Computer Vision and Pattern Recognition 2, 521–527 (2003)
7. Ataer, E., Duygulu, P.: Matching Ottoman Words: An Image Retrieval Approach to Historical Document Indexing. In: 6th Int'l. Conf. on Image and Video Retrieval, pp. 341–347 (2007)
8. Leydier, Y., Lebourgeois, F., Emptoz, H.: Text Search for Medieval Manuscript Images. Pattern Recognition 40, 3552–3567 (2007)
9. Srihari, S.N., Srinivasan, H., Huang, C., Shetty, S.: Spotting Words in Latin, Devanagari and Arabic Scripts. Indian Journal of Artificial Intelligence 16(3), 2–9 (2006)
10. Zhang, B., Srihari, S.N., Huang, C.: Word Image Retrieval Using Binary Features. In: Proceedings of the SPIE, vol. 5296, pp. 45–53 (2004)
11. Edwards, J., Whye, Y., David, T., Roger, F., Maire, B.M., Vesom, G.: Making Latin Manuscripts Searchable using gHMM's. In: Advances in Neural Information Processing Systems (NIPS), vol. 17, pp. 385–392. MIT Press, Cambridge (2004)
12. Cao, H., Govindaraju, V.: Template-free Word Spotting in Low-Quality Manuscripts. In: 6th Int'l. Conf. on Advances in Pattern Recognition (2007)
13. Marti, U.V., Bunke, H.: The IAM-Database: An English Sentence Database for Offline Handwriting Recognition. Int'l. Journal on Document Analysis and Recognition 5, 39–46 (2002)
14. Marti, U.V., Bunke, H.: Using a Statistical Language Model to Improve the Performance of an HMM-Based Cursive Handwriting Recognition System. Int'l. Journal of Pattern Recognition and Artificial Intelligence 15, 65–90 (2001)

15. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A Novel Connectionist System for Unconstrained Handwriting Recognition. IEEE Transaction on Pattern Analysis and Machine Intelligence 31(5), 855–868 (2009)
16. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist Temporal Classification: Labelling Unsegmented Sequential Data with Recurrent Neural Networks. In: 23rd Int'l. Conf. on Machine Learning, pp. 369–376 (2006)