Unconstrained Handwritten Word Recognition based on Trigrams Using BLSTM

Xi Zhang, Chew Lim Tan
School of Computing
National University of Singapore, Singapore
Email: \{xizhang, tancl\}@comp.nus.edu.sg

Abstract—To get high recognition accuracy, we should train the recognizer with sufficient training data to capture characteristics of various handwriting styles and all possible occurring words. However, in most of the cases, available training data are not satisfactory and enough, especially for unseen data. In this paper, we try to improve the recognition accuracy for unseen data with randomly selected training data, by splitting the training data into two parts based on trigrams and training two recognizers separately. We also propose a modified version of token passing algorithm, which makes use of the outputs of the two recognizers to improve the recognition accuracy.

I. INTRODUCTION

With the development of computer science and internet, a large number of valuable documents are scanned and stored in image format in digital databases and public access is provided online. Recognition of the content allows us to retrieve these documents using traditional text retrieval methods. However, recognition of unconstrained handwritten documents is always a challenging task and the poor results may cause unreliable and unsatisfactory retrieval output.

Handwriting recognition can be achieved at character level, or word level, even at text line level nowadays. Because of its wide usage and popularity in speech processing tasks, Hidden Markov Models (HMMs) are also applied for handwritten document analysis. In [1], one isolated handwritten word is represented by one HMM, but this word classification approach cannot be used for words which do not appear in the training data and a considerable amount of training data for each word is required. Moreover, it cannot be scaled to large vocabularies, because every distinct occurring word needs an HMM. Therefore, in order to recognize arbitrary words, HMMs are used to represent character models instead of the whole words, and one word or text line are represented by a sequence of HMMs, which are linearly connected [2]. Based on the trained HMMs, for a given text line, we can obtain the most likely character sequence and the beginning and end positions of each character along the text line, using the Viterbi algorithm.

However, methods based on HMMs suffer from several disadvantages. The probability of every observation only depends on the current state, and it is difficult to take the context information into account. Moreover, because handwritten document recognition is always a discriminative task, HMMs, which are generative, may not provide better performance than other discriminative models.

Combining HMMs and neural networks is a kind of hybrid method for handwriting recognition. Many kinds of architectures of neural networks are applied, such as Multilayer Perceptrons (MLP) [3], time delay neural network [4], and Recurrent Neural Networks [5]. Although the hybrid method can capture context information, they also suffer from some drawbacks of HMMs.

In the recent works, Recurrent Neural Network (RNN), with Connectionist Temporal Classification (CTC) output layer is applied for unconstrained handwritten document recognition [6]. Traditional RNN needs presegmented input data, namely, we should label each position of the input data. But, RNN with CTC output layer can map the whole unsegmented sequence of the input data to the output labels directly. Combined with a dictionary, the recognition results for both on-line and off-line data outperform HMMs.

In order to get satisfactory recognition results, sufficient training data is always important. However, in practice, unavailability of enough training data is always the case. What is more, words in the training data may be distributed unevenly, namely, some words appear much more frequently than others. In unconstrained writing styles, consecutive characters are always connected and how one character is connected with other neighboring characters are different for different writers or even the same writer. Therefore, given a set of training data, we split it into two parts based on the occurring trigrams (three consecutive characters), so that the two training sets contain exclusive words with two sets of different trigrams. We aim at training two networks separately on the two sets and make the trained networks capture reliable information about how consecutive characters appear together in the corresponding training set. In the decoding period, instead of calculating the probability of each whole word in the dictionary appearing in the test images, we generate the weighted probabilities of all trigrams of each possible word, and combine them together to get the final score.

II. PREPROCESSING

The database we use in the experiments is IAM offline handwritten database [7], consisting of 657 writers. All documents are segmented into isolated and labeled word images. In order to reduce variations due to different writing styles, the word images are binarized, with skew and slant correction [8], and the heights of ascender, the main body and descender parts are normalized to 20, 40, and 20, respectively, as shown in Fig. 1.

Nine geometrical features are extracted from a sliding
window, with width of 1, moving from left to right along each word image. The features are shown as follows [6]:

1) the number of black pixels
2) the center of gravity of the group of pixels
3) the second order moment of the window
4) the location of the upper-most pixel
5) the location of the lower-most pixel
6) the orientation of the upper-most pixel
7) the orientation of the lower-most pixel
8) the number of black-white transitions
9) the number of black pixels divided by the number of all pixels between the upper- and lower-most pixel

(a) The original word image.

(b) The normalized word image.

Fig. 1. An example of the normalized result for a word image from IAM database.

Each dimension of the nine features in one word image is normalized by subtracting the mean and dividing by the standard deviation.

III. NEURAL NETWORK FOR RECOGNITION

The recognizer we use for off-line handwritten word images is a Recurrent Neural Network (RNN). The hidden nodes of RNN are self-connected and also connected to the hidden nodes in the later time step, so that RNN can capture a certain range of previous input information, as shown in Fig. 2(a). However, only history information can be used by RNN. In order to make full use of all input data, whether in the past or in the future, an additional backward hidden layer is added, which is not connected to the original forward hidden layer, as shown in Fig. 2(b). The forward states are trying to capture the past information, and the backward states are making use of the future information, and together they construct the Bidirectional Recurrent Neural Network (BRNN) and allow each time step to be evaluated based on both the past and future information.

However, the authors in [10] present one drawback of the traditional RNN, named as vanishing gradient problem, which means there is a limitation in the range of contextual information that we can use to train the network, because the influence of each input will be reduced exponentially over time. In order to overcome this limitation, a specially designed architecture is used instead of the traditional one, named as Long Short-Term Memory (LSTM) [11]. The hidden nodes of the traditional RNN in Fig. 2 are replaced by the LSTM memory block, shown in Fig. 3. One LSTM memory block with a single cell has three gates, which control the cell to access information over a long time period.

Fig. 2. Structure of Recurrent Neural Network from [9].

Fig. 3. Structure of LSTM memory block with a single cell from [6]. There are three gates: input gate, output gate, and forget gate. They collect the input from other parts of the network and control the information the cell can accept. The input and output of the cell are controlled by the input gate and output gate, while how the recurrent connection affects the cell is controlled by the forget gate.

IV. SPLITTING OF RANDOMLY SELECTED TRAINING DATA

In practice, we may have insufficient training data on hand, and the recognizer cannot recognize the words very well which do not or rarely appear in the training data. Especially, if the words in the training data are not distributed evenly, namely, some words appear hundreds of times, but others may rarely appear. As a result, the trained network fits very well characteristics which occur much more frequently in the training data, but fail for rarely appearing features. In this
paper, we try to improve the recognition results, especially for the words which do not appear in the training data. Moreover, the training data is randomly selected from a collection of word images, namely, any word and any number of word images for each word can be included in the training data, which is always the case in the real world. We also test our method on the writer-independence dataset, the test set of which is also a kind of unseen data with respect to the training data.

Given a set of training data, we try to split it into two parts and train them separately. Assume train_data is a set of randomly selected training data, and train_data1, train_data2 are two exclusive training data from train_data. From the ground truth of the training data, we can generate a set, train_dict, which contains all distinct occurring words in the training data. We randomly select two words from train_dict, which do not have the same trigrams, and assign their prefix trigrams to tri1 and tri2, respectively, as the initialization. For example, if 'company' and 'special' are selected, then tri1 and tri2 are initialized to 'company' and 'special', respectively. Then, we will generate two subsets set1 and set2 from train_dict as described in Algorithm 1.

Algorithm 1: Splitting the training data into two subsets

1: Random() ∈ (0, 1)
2: for i = 1 → size(train_dict) do
3:   tri ← all trigrams in train_dict[i]
4:   mark1 = 0, mark2 = 0
5:   if ∃t : t ∈ tri and t ∈ tri1 then
6:     mark1 = 1
7:   end if
8:   if ∃t : t ∈ tri and t ∈ tri2 then
9:     mark2 = 1
10:  end if
11:  switch mark do
12:    case 0:
13:      if Random() > 0.5 then
14:        Add tri to tri1, and train_dict[i] to set1
15:      else
16:        Add tri to tri2, and train_dict[i] to set2
17:      end if
18:    case 10:
19:      Add tri to tri1, and train_dict[i] to set1
20:    case 1:
21:      Add tri to tri2, and train_dict[i] to set2
22:  end switch
23: end for
24: W = train_dict – (set1 ∪ set2)
25: for i = 1 → size(W) do
26:   tri ← all trigrams in W[i]
27:   if size(tri ∩ tri1) > size(tri ∩ tri2) then
28:     Add tri to tri1, and W[i] to set1
29:   else
30:     Add tri to tri2, and W[i] to set2
31: end if
32: end for

For each word in train_dict, we first extract all its trigrams and store them into tri. Then we check whether tri1 or tri2 contains one of the trigrams in tri. If a trigram in tri is included in tri1, mark1 is assigned to 1, otherwise 0; if a trigram in tri is included in tri2, mark2 is assigned to 1, otherwise 0. Then, mark is equal to mark1 × 10 + mark2, and Table I shows the meanings of its different values. If both tri1 and tri2 contain none of the trigrams in tri (mark = 0), we will randomly add tri and the word to either tri1 or set1 or tri2 and set2, with equal probability. If only one of tri1 and tri2 contains at least one trigram in tri (mark = 1 or 10), we will add tri to it, and add the word to the corresponding set.

<table>
<thead>
<tr>
<th>mark</th>
<th>0</th>
<th>1</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>tri1</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>tri2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

′✓’ means containing at least one trigram in tri, and ′×’ means containing none of the trigrams in tri.

When we finish checking all the words in train_dict, tri1 and tri2 are two sets of trigrams, and set1 and set2 are two exclusive subsets of train_dict. However, there are some words, subsets of whose trigrams are both included in tri1 and tri2, so if tri1 contains more trigrams of the words than tri2, we will add the trigrams and the words to tri1 and set1 respectively, otherwise, put them into tri2 and set2.

Because of the randomness when mark = 0, we repeat Algorithm 1 for several times, and choose the results, in which the sizes of tri1 and tri2 are nearly the same and they have least common trigrams. At last, based on set1 and set2, we put the word images containing the words in set1 to train_data1, and others to train_data2.

Consequently, we split the training data into two parts, and make them contain two sets of different trigrams. For training, we construct two RNNs with LSTM hidden layers (BLSTM) [6] Net1 and Net2, trained on train_data1 and train_data2 respectively, in order to make the networks capture information of tri1 and tri2 separately. Because of insufficient training data, the more sets we split into, the fewer word images each set contains, and more likely we cannot get well enough trained networks as we expect. Moreover, the computation cost for combining outputs from multiple networks depends on the number of split sets; therefore, we split the training data into two sets instead of three or more.

V. MODIFIED CTC TOKEN PASSING ALGORITHM

A. CTC Token Passing Algorithm

Based on the token passing method for HMMs [12], CTC token passing algorithm is proposed in [6] for text line recognition and also used for keyword spotting in [13]. For single word recognition, we expect the probability of each possible word with a sequence of ASCII characters based on the output of the trained network. Assume we have N different characters and the input has t time steps, then the network will output the probability of each character appearing at each time step, constructing an N × t matrix. Using dynamic programming, the best path through character probabilities is calculated for each input word, and the final accumulated probability value is treated as the score. The word with the highest score is returned as the most likely transcription.
As shown in Fig. 4, the image on the top shows the output of a trained network with the input image at the bottom, containing the word ‘report’. The total number of time steps is the same as the width of the input image, and we assume that the network is trained for the 26 lower-case characters. The darker the block is, the higher the probability of the corresponding character appears with. For example, at time step 25, the 18th character ‘r’ has the highest probability, assuming characters are indexed in lexicographical order. However, in some cases, the trained network cannot recognize every character perfectly, such as the character ‘y’, which has similar probability with ‘n’ at time step 180. Therefore, a dictionary is used to filter out impossible character sequences.

**B. Modification to spot trigrams**

Because we try to spot possible trigrams in the input image, other than the whole word, based on the word spotting algorithm proposed in [13], we propose a modified version in order to get the weighted probability for each trigram of one input word and combine them together to get the final score.

When we finish training Net1 and Net2, which are trained on train_data1 and train_data2 separately, we apply a modified token passing algorithm described in Algorithm 2 to calculate the probability of each word in a closed dictionary Dict. When the trained Net1 (i = 1, 2) accepts a sequence of column features extracted from a word image, it will return Prob \( \{c, t\} \) with two parameters: \( c \) indicates the character (lower-case and capital characters, blank symbol ‘#’), and any-character symbol ‘*’ and \( t \) indicates the time step or position, so the value of Prob \( \{c, t\} \) is the probability of the character \( c \) appearing at time \( t \). We assume ‘w’ can appear at any time step, so Prob \( \{*, t\} \) = 1 for all \( t \).

For each word \( w \) = \( c_1c_2...c_n \) in Dict, we first generate all its trigrams \( \{c_1c_2c_3, c_2c_3c_4, \ldots, c_{n-2}c_{n-1}c_n\} \). We aim at obtaining the probability of each trigram appearing in the input image instead of the whole character sequence as in [13], and we also should take the order of how the trigrams appear one by one into account. So, for each trigram, we define a set \( w' \) based on the set of trigrams, by first adding ‘#’ at the beginning and end of each character, and add ‘*’ to the beginning except for the first trigram, and at the end except for the last trigram. For example, the trigrams of the word ‘limit’ is \{’lim’, ’imi’, ’mit’\}, and \( w' \) is \{’#lim#i#m#i#t’\} for each entry in \( w' \), \( w'[j], j \in [1, \text{size}(w')] \), instead of using all time steps, we only consider the part where the corresponding trigram probably appears. Assume the width of the input image is \( L \), the estimated width of one character in \( w \) is \( L/\text{len}(w) \) \( (\text{len}() \) calculates the number of characters in the given string), denoted as \( len_c \), so the width of one trigram is estimated as \( 3 \times len_c \). We divide the whole time steps into \( len(w) - 2 \) parts, each of which corresponds to one entry in \( w' \). For \( w'[j] \), the time steps we consider are \( \{(j - 1) \times len_c + 1, \ldots, \min(L, (j + 2) \times len_c)\} \).

Because the width of all characters may not be the same, such that the width of two consecutive characters is less than \( 2 \times len_c \) or the width of one character is larger than \( len_c \), so we expand each part by \( len_c + 1 \) at the beginning and the end, and the time steps under consideration are changed to \( \{\max(j - 2, 0) \times len_c + 1, \ldots, \min(L, (j + 3) \times len_c)\} \), where the minimal and maximal value are denoted as \( s_p \) and \( e_p \), respectively. For example, if the width of the input word image is 200, and we want to generate the probability for word ‘lim’, then \( len_c = 40 \) and the divided parts of time steps is \( \{1 : 160, 1 : 200, 41 : 200\} \). Due to the expansion, the time steps for \( w'[j] \) may include other parts of characters, so we allow any-character symbol ‘*’ appearing at the beginning or the end of the trigram as shown in Line 3.4 of Algorithm 2.

**Algorithm 2 Modified CTC Token Passing Algorithm Combining Outputs of Two Trained Networks**

1: \( P = 0 \)
2: Input word \( w = c_1c_2...c_n, w \in \text{Dict} \)
3: \( w' = \{’#c_1#c_2#c_3#*’, ’#c_2#c_3#c_4#*’, \ldots\} \)
4: \( {t} = \text{size}(w') \)
5: for \( i = 1 \rightarrow t\) do
6: \( \text{tri} = w'[i] \)
7: \( L_\text{tri} = \text{len}(\text{tri}) \)
8: \( s_p = \max(i - 2, 0) \times \text{len}_c + 1 \)
9: \( e_p = \min(L, (i + 3) \times \text{len}_c) \)
10: \( l = e_p - s_p + 1 \)
11: \( \text{V}_l(r, c) = 0 \) for all \( r \in [1, L_\text{tri}] \) and \( c \in [1, l] \)
12: \( \text{V}_l(1, 1) = \text{Prob}_1(\text{tri}[1], s_p) \)
13: \( \text{V}_l(2, 1) = \text{Prob}_1(\text{tri}[2], s_p) \)
14: for \( t = s_p + 1 \rightarrow e_p \) do
15: \( \text{p} = 1 \rightarrow L_\text{tri} \) do
16: \( \text{Best} = \text{V}_l(p, t - 1) \)
17: if \( p > 1 \) then
18: \( \text{Best} = \text{Best} \cup \text{V}_l(p - 1, t - 1) \)
19: if \( p > 2 \) and tri[p] \notin \{’, #, tri[p] - 2\} \) then
20: \( \text{Best} = \text{Best} \cup \text{V}_l(p - 2, t - 1) \)
21: end if
22: end if
23: \( \text{V}_l(p, t) = \text{max}(\text{Best}) \times \text{Prob}_1(\text{tri}[p], t) \)
24: end for
25: \( \text{V}_2 \) is constructed by the same manner based on \( \text{Prob}_2 \)
26: \( \text{Score}_1 = \text{log}(\text{max}(\text{V}_1(L_\text{tri}, t), \text{V}_1((L_\text{tri} - 1, l)))/L_\text{tri} \)
27: \( \text{Score}_2 = \text{log}(\text{max}(\text{V}_2(L_\text{tri}, t), \text{V}_2((L_\text{tri} - 1, l)))/L_\text{tri} \)
28: \( P = P + (\text{Score}_1 \times \omega_1 + \text{Score}_2 \times \omega_2) \)
29: end for
30: return \( \text{Score} = P / \text{size}(w') \)
Given \( w'[j], s_p, \) and \( e_p, \) we construct a matrix \( V_i \) \((i = 1, 2),\) with the size of \( \text{len}(w'[j]) \times (e_p - s_p + 1),\) which is initialized to \(-\ln f\) for all elements. \( V_i(t)\) is updated based on the values in \( V_i(t-1)\) to get the optimal character sequence at each time step. \( \text{score}_i\) is obtained based on the output of \( \text{Net}_i\), however, when the length of \( w'[j]\) is bigger, we will accumulate more probabilities in \( V_i\), so \( \text{score}_i\) is normalized by the length of \( w'[j]\). Assume \( n_i\) is the number of \( w'[j]\) appearing in \( \text{train}_i\), dividing by the total number of trigrams in \( \text{train}_i\), then \( \omega_i = (n_i + 1)/(n_i + n_{i+1} + 2),\) so that we give more trust to the network which is trained on the data set containing more occurrences of the trigram. We add the sum of the weighted scores to \( P\). However, the probability of word \( w\) also depends on its length. At last, \( P\) is normalized by the size of \( w',\) as the returning score. Therefore, we calculate the probabilities of all words in the dictionary, and return the one with the highest score.

VI. EXPERIMENTS

A. Experimental Setup

The experiment data consist of word images segmented correctly and containing English words with more than 4 characters, totally 21332 word images and 1601 distinct words. For words containing less than 4 characters, most of them are stop words, so we do not test them in our experiments.

The recognition network is BLSTM, which has 9 input nodes and 53 output nodes, containing 52 lower-case and capital characters and one more node for ‘blank’, the symbol ‘#’ in Algorithm 2. The forward and backward hidden layers both have 100 LSTM memory blocks, and the number of total weights is 99253, which are initialized with a Gaussian distribution of mean zero and standard deviation 0.1. Gradient descent is used for training the network, with learning rate 0.0001 and momentum 0.9. In the decoding, we only use a closed dictionary, without any language model.

The performance is measured on the character error rate (CER):

\[
\text{CER} = 100 \times \left( \frac{\text{insertions} + \text{substitutions} + \text{deletions}}{\text{total number of characters in the testing data}} \right)
\]

where all the counts are summed over the whole test set.

Word error rate (WER) is also recorded, which is defined as the number of words recognized wrongly dividing by the total number of words in the test set.

B. Results on Randomly Selected Training and Testing Data

First, we randomly select 20% of the 1601 distinct words and make the corresponding word images as testing data. Moreover, we try to avoid adding words, which have the same relatively long prefix in the test set simultaneously, such as ‘writes’ and ‘write’. In the all remaining word images, we randomly select 70% for training, and 30% for validating. We repeat the above procedure and do the experiment for five times, and record the average results. The number of distinct words and the corresponding word images for 5 experiments, \( \text{Exp}_i, i = 1, 2, ..., 5\), are shown in Table II.

In Fig. 5, the character error rates for validation dataset of one experiment during the first 100 training iterations are shown, where \( \text{Net}_i\) is trained on \( \text{train}_i\), and \( \text{Net}_i\) is trained on \( \text{train}_i\), \( i = 1, 2\). The error rates of \( \text{Net}_1\) and \( \text{Net}_2\) drop faster than \( \text{Net}\) in the first 10 iterations and \( \text{Net}_2\) is converged much faster than \( \text{Net}\) and \( \text{Net}_1\). We can see that \( \text{Net}_1\) and \( \text{Net}_2\) have higher character error rates than \( \text{Net}\) in the following iterations, because they are trained on only half of the whole training word images. We choose the networks with the best character error rates on validation data for decoding in our experiments, for example, the best networks for \( \text{Net}, \text{net}_1\) and \( \text{Net}_2\) are in the 60, 61, 31 iterations in the figure 5, respectively.

![Fig. 5. Character error rate on the validation data over first 100 iterations.](image)

We do the experiments for 5 times on 5 different sets of training and testing data, as described in Table II. The results for \( \text{Net}\) are based on the CTC token passing algorithm in [13] for single words and the results for \( \text{Net}_1 + \text{Net}_2\) are based on our method. As shown in Table III and Table IV, our methods have better results according to both CER and WER. For our method, the two trained networks are trained on different sets of distinct words, in order to capture characteristics of different sets of trigrams separately, in contrast to training one network with a large set of data, containing large variations. In the decoding, for each possible trigram, we give more trust to the network which is trained on more word images containing the trigram, i.e. the trigram appears much more times in the corresponding training data. Therefore, we get the probability for each word in the dictionary by the outputs of two networks, both of which give their results with more confidence.

C. Results on Writer Independent Training and Testing Data

Besides using randomly selected training and testing data, we also test our algorithm on the public dataset used for Large

---

### Table II. The Number of Distinct Words and the Corresponding Word Images for Each Data Set.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Exp1</th>
<th>Exp2</th>
<th>Exp3</th>
<th>Exp4</th>
<th>Exp5</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>982</td>
<td>975</td>
<td>974</td>
<td>974</td>
<td>974</td>
<td>974</td>
</tr>
<tr>
<td>Valid</td>
<td>1199</td>
<td>1198</td>
<td>1198</td>
<td>1198</td>
<td>1198</td>
<td>1198</td>
</tr>
<tr>
<td>Test</td>
<td>3284</td>
<td>3276</td>
<td>3276</td>
<td>3276</td>
<td>3276</td>
<td>3276</td>
</tr>
</tbody>
</table>

In each entry, the value at the left of ‘/’ is the number of distinct words in the corresponding data set, and the right value is the total number of word images.

---

### Table III. Character Error Rate (CER) (%)

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Exp1</th>
<th>Exp2</th>
<th>Exp3</th>
<th>Exp4</th>
<th>Exp5</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net</td>
<td>12.84</td>
<td>11.24</td>
<td>12.46</td>
<td>13.51</td>
<td>11.28</td>
<td>12.46</td>
</tr>
</tbody>
</table>
TABLE IV. WORD ERROR RATE (WER%)

<table>
<thead>
<tr>
<th>test set</th>
<th>b.exp1</th>
<th>b.exp2</th>
<th>b.exp3</th>
<th>b.exp4</th>
<th>b.exp5</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net1 + Net2</td>
<td>15.35</td>
<td>11.27</td>
<td>10.94</td>
<td>11.57</td>
<td>15.39</td>
<td>12.08</td>
</tr>
</tbody>
</table>

Writer Independent Text Line Recognition Task [7]. We put 21332 word images into the corresponding sets, and totally 16500 words are used. The separation of the dataset is shown in Table V, on which Net will be trained:

TABLE V. WRITER INDEPENDENT DATASET FOR Net

<table>
<thead>
<tr>
<th>Set Name</th>
<th>No. of Word Images</th>
<th>No. of Distinct Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>11642</td>
<td>283</td>
</tr>
<tr>
<td>Validation 1</td>
<td>1389</td>
<td>46</td>
</tr>
<tr>
<td>Validation 2</td>
<td>1339</td>
<td>43</td>
</tr>
<tr>
<td>Test</td>
<td>2200</td>
<td>128</td>
</tr>
</tbody>
</table>

For our algorithm, we split the training data using Algorithm 1, and based on set1 and set2 we get, each validation dataset is also split into two sets, each of which only contains the words in set1 or set2. Table VI shows the dataset we use for our method. In set1, there are 1214 distinct trigrams, and in set2, there are 1231 distinct trigrams. The number of common trigrams in the two sets are 371.

TABLE VI. WRITER INDEPENDENT DATASET FOR Net1 and Net2

<table>
<thead>
<tr>
<th>Set Name for Net1</th>
<th>No. of Word Images</th>
<th>No. of Distinct Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>5856</td>
<td>751</td>
</tr>
<tr>
<td>Validation 1</td>
<td>643</td>
<td>279</td>
</tr>
<tr>
<td>Validation 2</td>
<td>606</td>
<td>277</td>
</tr>
<tr>
<td>Set Name for Net2</td>
<td>No. of Word Images</td>
<td>No. of Distinct Words</td>
</tr>
<tr>
<td>Train</td>
<td>5786</td>
<td>737</td>
</tr>
<tr>
<td>Validation 1</td>
<td>626</td>
<td>278</td>
</tr>
<tr>
<td>Validation 2</td>
<td>783</td>
<td>326</td>
</tr>
</tbody>
</table>

We train Net1 and Net2 on two sets of validation dataset, and test on the same test dataset in Table V. The results are shown in Table VII, which are the average results after doing experiments for 5 times, i.e. different weight initializations for the networks. Because of writer independence in the training and testing dataset and large writing variations among different writers, the CER and WER for Net, Net1 and Net2 are all higher than our experiments in Section VI-B. However, our method, combining two trained networks, also has better recognition results. We also split the training set randomly, i.e. put each training word image into either train_data1 or train_data2 with equal probability, and use the same value 0.5 for both ω1 and ω2 in Algorithm 2. As shown in the last row in Table VII, the results are worst. Because of splitting the training data randomly, in each set, the variations are large, but the training data is half, so that, the trained networks can not fit to the large variations very well.

TABLE VII. RESULTS ON LARGE WRITER INDEPENDENT DATASET

<table>
<thead>
<tr>
<th>Validation 1</th>
<th>validation 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CER(%):</td>
<td>WER(%):</td>
</tr>
<tr>
<td>Net</td>
<td>14.2</td>
</tr>
<tr>
<td>Net1 + Net2</td>
<td>11.7</td>
</tr>
<tr>
<td>Net1 + Net2 (Randomly)</td>
<td>16.2</td>
</tr>
</tbody>
</table>

VII. CONCLUSION

In this paper, we propose a new method combining the outputs of two networks, which are trained on the subset of the training data. The splitting of the training data into two subsets satisfies the condition that the different words in the two sets are exclusive and the two word sets have as few common trigrams as possible. Our method for decoding is a modified version of the Token Passing Algorithm and we only focus on spotting trigrams instead of the whole character sequence for an input word. In the experiment, we select the training data and testing data from a collection of word images randomly and also test on the Writer Independence Recognition Task dataset. Our method has better results both on the character error rate and word error rate. What is more, our modified CTC token passing algorithm can also be used to get better recognition results by combining two trained networks, which are trained on different sets of training data, than using each network individually.

In the future, we try to apply our method directly on text line images, and also try to reduce the time cost for decoding. What is more, we would like to test on other databases, especially for other languages.

REFERENCES


In this paper, we propose a new method combining the outputs of two networks, which are trained on the subset of the training data. The splitting of the training data into two subsets satisfies the condition that the different words in the two sets are exclusive and the two word sets have as few common trigrams as possible. Our method for decoding is a modified version of the Token Passing Algorithm and we only focus on spotting trigrams instead of the whole character sequence for an input word. In the experiment, we select the training data and testing data from a collection of word images randomly and also test on the Writer Independence Recognition Task dataset. Our method has better results both on the character error rate and word error rate. What is more, our modified CTC token passing algorithm can also be used to get better recognition results by combining two trained networks, which are trained on different sets of training data, than using each network individually.

In the future, we try to apply our method directly on text line images, and also try to reduce the time cost for decoding. What is more, we would like to test on other databases, especially for other languages.

REFERENCES