Multiple Convolution Neural Networks for an Online Handwriting Recognition System

Phạm Việt Dũng
Computer Network Centre
Vietnam Maritime University
Haiphong City, Vietnam
e-mail: vietdungiitb@vimaru.edu.vn

***Abstract*— This paper focuses on a specific word recognition technique for an online handwriting recognition system which uses Multiple Component Neural Networks (MCNN) as the exchangeable parts of the classifier. As the most recent of approaches, the system proceeds by segmenting handwriting words into smaller pieces (usually characters), which are recognized separately. The recognition results are then a composition of individually recognized characters. They are sent to the input of a word recognition module to choose the most suitable one by applying some dictionary search algorithms. The proposed classifier overcomes obstacles and difficulties of traditional ones to large character classes. Furthermore, the proposed classifier also has expandable capacity, which can recognize other character classes by adding or changing component networks and built-in dictionaries dynamically.**

Keywords- online handwriting; recognition; convolution; neural network.

I. INTRODUCTION

Nowadays, Touch User Interfaces (TUI) devices are becoming increasingly popular and already play an important role in human-computer interaction. Tablets, smartphones and TUI computers accepting finger touch or pen based input are becoming a crucial part of many everyday life activities. Using fingers or a pen as an input device covers more and improves many functions as compared to the conventional mouse and keyboard. One major advantage of the pen over the mouse is the fact that it is a natural writing tool for humans while the computer's mouse is very hard to use as a writing tool. However, it needs a reliable transformation of handwritten text into a coding that can be directly processed by a computer, e.g., ASCII [6]. A traditional transformation model usually includes a preprocessor which extracts each word from image or input screen and divides it into segments. A neural network classifier then finds the likelihoods of each possible character class given the segments. These likelihoods are used as the input to a special algorithm which recognizes the entire word. In recent years, research in handwriting recognition has advanced to a level that makes commercial applications. Nevertheless, significant disadvantages of such single neural network classifiers are their complexity in big network organizations and expandable capacity [4].

A highly reliable recognition rate neural network can be easily built to recognize a small character class, but not larger ones. The larger inputs and outputs increase the number of layers of the neural network, neurons, and connections. Hence, it makes the network training process more difficult and especially the recognition rate, which is significantly decreased [8]. Furthermore, a single neural network classifier only works on a particular character class. It is not exchangeable or expandable to recognize additional character classes without recreating or retraining the neural network.

This paper presents a new online handwriting recognition system that is based on multiple Convolutional Neural Networks (CNNs). As is well known, CNNs are efficient for various applications [9]. They are presented in Section 2. Section 3 presents a new classifier that includes a collection of very high recognition rate component CNNs working together. Each CNN can only correctly recognize a part of the big character class (digits, alphabet, etc.), but when these networks are combined by programing algorithms, they can create a flexible classifier which can recognize differential big character classes by simply adding or removing component CNNs and language dictionaries. The computer simulation results are shown in Section 4. Finally, the conclusion is presented in Section 5.

II. CONVOLUTION NEURAL NETWORK

CNNs are a special kind of multi-layer neural networks. Like almost every other neural network, they are trained with a version of the back-propagation algorithm. Where they differ is in their architecture. CNNs are designed to recognize visual patterns directly from pixel images with minimal preprocessing. They can recognize patterns with extreme variability (such as handwritten characters), and with robustness to distortions and simple geometric transformations.

The LeNET 5 (see Fig. 1) for handwritten digit recognition has allowed a reliable recognition rate of up to 99% to MNIST dataset [1].

 ISBN: 978-1-61208-371-1

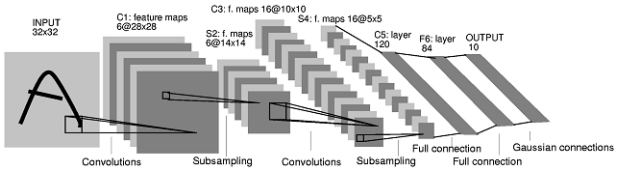

Figure 1. A Typical Convolutional Neural Network (LeNET 5) [1]

The input layer is of size 32 x32 and receives the gray-level image containing the digit to recognize. The pixel intensities are normalized between −1 and +1. The first hidden layer C1 consists of six feature maps, each having 25 weights, constituting a 5x5 trainable kernel, and a bias. The values of the feature map are computed by convolving the input layer with the respective kernel and applying an activation function to obtain the results. All values of the feature map are constrained to share the same trainable kernel or the same weights values. Because of border effects, the feature maps' size is 28x28, smaller than the input layer.

Each convolution layer is followed by a sub-sampling layer which reduces the dimension of the respective convolution layer's feature maps by a factor of two. Hence, the sub-sampling maps of the hidden layer S2 are of size 14x14. Similarly, layer C3 has 16 convolution maps of size 10x10 and layer S4 has 16 sub-sampling maps of size 5x5. The functions are implemented exactly as layers C1 and S2 perform. The S4 layer's feature maps are of size 5x5, which is too small for a third convolution layer. The C1 to S4 layers of this neural network can be viewed as a trainable feature extractor. Thereafter, a trainable classifier is added to the feature extractor, in the form of 3 fully connected layers (a universal classifier).

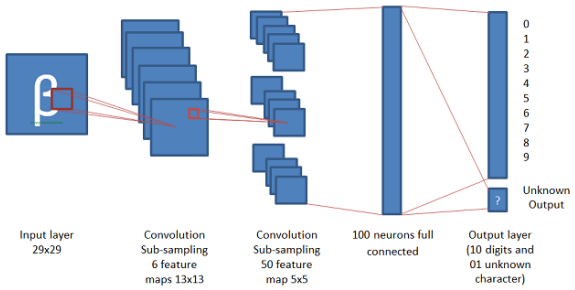

Figure 2. A convolution network based on P. Simard's model [3]

Another model of CNN for handwritten digit recognition that integrates convolution and sub-sampling processes into a single layer also grants recognition rates of over 99% [3]. This model, presented in Fig. 2, extracts simple feature maps at a higher resolution, and then converts them into more complex feature maps at a coarser resolution by sub-sampling a layer by a factor of two. The width of the trainable kernel is chosen to be centered on a unit (odd size), to have sufficient overlap to not lose information (3 would be too small with only one unit overlap), but yet not to have redundant computation (7 would be too large, with 5 units or over 70% overlap). Padding the input (making it larger so that there are feature units centered on the border) does not improve performance significantly. With no padding, a sub-sampling of two, and a trainable kernel of size 5x5, each convolution layer reduces the feature map size from n to (n-3)/2. Since the initial MNIST input used in this model is of size 28x28, the nearest value which generates an integer size after 2 layers of convolution is 29x29. After 2 layers of convolution, the feature of size 5x5 is too small for a third layer of convolution. The first two layers of this neural network can be viewed as a trainable feature extractor after which a trainable classifier is added to the feature extractor in the form of 2 fully connected layers (a universal classifier).

III. MULTIPLE COMPONENT NEURAL NETWORKS CLASSIFIER

CNN can secure a significantly higher recognition rate than traditional multilayer perceptron neural network to small character classes such as digits [3] or the English alphabet (26 characters). However, creating a larger neural network that can reliably recognize a bigger collection (62 characters) is still a challenge. Finding an optimized and large enough network becomes more difficult. Training networks by large input patterns takes a much longer time. Convergent speech of the network is slower and the accuracy rate is significantly decreased because of bigger badly written characters, similar and confusable characters etc.

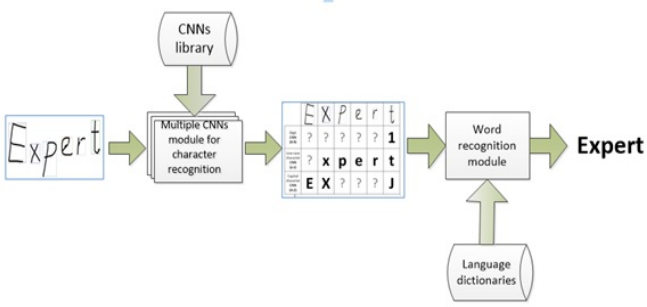

Figure 3. A MCNNs online handwriting recognition system

The proposed solution to the above problem is replacing a unique complex neural network with multiple smaller networks which have a high recognition rate to these own output sets [4]. Fig. 3 illustrates explicitly the working process of this new system. Each component network has an additional unknown output (unknown character) beside the official output sets (digit, letters, etc.). This means that, if the input pattern is not recognized as a character of official outputs, it will be understood as an unknown character.

The character recognition module of the classifier is a collection of multiple component neural networks, which work simultaneously with the input patterns. A handwritten

 ISBN: 978-1-61208-371-1

word is pre-processed by segmenting the isolated character visual patterns [4]. These patterns are then given to the inputs of all the component neural networks, which will recognize likelihoods of each character class. A visual pattern can be recognized by one, some, or all component networks because there are several similar characters in different classes. If a network cannot recognize the pattern as a likelihood of its own character class, it will return an unknown character (null character). The module's output result, presented in Fig. 4, is a table of possible characters which is composed of possible words, such as "Exper1, Expert, ExperJ, EXper1, EXpert, EXperJ" in the above example. Unknown characters (null characters) are not used in word composition. These words are then given to the next word recognition module in turn to choose the most suitable one to become the output of overall classifier. In this example the word "Expert"will be chosen.

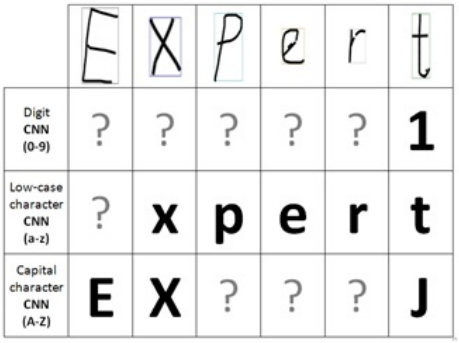

Figure 4. Output of MCNNs classifier module

The algorithm of word composition uses in character recognition module:

Global variables:

- *charMatrix = List<List<Char>> {{E},{x,X},{p},{e},{r},{1,t,J}} // character table*
- *words =List<string> //list of composed word.*
- *startIndex: default is 0*
- *baseWord: default is "*

```
void GetWords(int startIndex, String baseWord)
{
        String newWord = "";
        if (startIndex == charMatrix.Count - 1)
        {
           for (int i = 0; i < charMatrix[startIndex].Count; i++)
           {
              newWord = String.Format("{0}{1}", baseWord,
charMatrix[startIndex][i].ToString());
              words.Add(newWord);
           }
        }
        else
        {
           for (int i = 0; i < charMatrix[startIndex].Count; i++)
           {
              newWord = String.Format("{0}{1}", baseWord,
charMatrix[startIndex][i].ToString());
              int newIndex = startIndex + 1;
              GetWords(newIndex, newWord);
           }
        }
}
```

Word recognition module is in fact a spell checker which uses several dictionary search algorithms and word correction techniques to get the best meaning of the word. All possible words from character recognition modules are given to the dictionary search sequentially. If one of the words is found in built-in dictionaries, it will become the output word of the classifier. Otherwise, some other word correction technique will be applied for choosing the most suitable word in automatic mode or showing a list of similar words to user in manual mode. Some of these techniques are:

- Swap each character one by one and try all the chars in its place to see if that makes a good word.

```
private bool ReplaceChars(String word, out String result)
{
    result = "";
    bool isFoundWord = false;
    foreach (WordDictionary dictionary in Dictionaries)
    {
       ArrayList replacementChars =
dictionary.ReplaceCharacters;
       for (int i = 0; i < replacementChars.Count; i++)
       {
          int split = ((string)replacementChars[i]).IndexOf(' ');
          string key = ((string)replacementChars[i]).Substring(0,
split);
          string replacement =
((string)replacementChars[i]).Substring(split + 1);
          int pos = word.IndexOf(key);
          while (pos > -1)
          {
             string tempWord = word.Substring(0, pos);
             tempWord += replacement;
             tempWord += word.Substring(pos + key.Length);
             if (this.TestWord(tempWord))
             {
                result = tempWord.ToString();
                isFoundWord = true;
                return isFoundWord;
             }
              pos = word.IndexOf(key, pos + 1);
             }
          }
       }
       return isFoundWord;
    }
```

- try swapping adjacent chars one by one.

```
private bool SwapChar(String word, out String result)
{
        result = "";
        bool isFoundWord = false;
        foreach (WordDictionary dictionary in Dictionaries)
        {
           for (int i = 0; i < word.Length - 1; i++)
           {
```

```
            StringBuilder tempWord = new
StringBuilder(word);
            char swap = tempWord[i];
            tempWord[i] = tempWord[i + 1];
            tempWord[i + 1] = swap;
            if (this.TestWord(tempWord.ToString()))
            {
               result = tempWord.ToString();
               isFoundWord = true;
               return isFoundWord;
            }
         }
      }
      return isFoundWord;
 }
```

- Try inserting a new character before every letter.

```
private bool ForgotChar(String word, out String result)
    {
      result = "";
      bool isFoundWord = false;
      foreach (WordDictionary dictionary in Dictionaries)
      {
         char[] tryme =
dictionary.TryCharacters.ToCharArray();
         for (int i = 0; i <= word.Length; i++)
         {
            for (int x = 0; x < tryme.Length; x++)
            {
               StringBuilder tempWord = new
StringBuilder(word);
               tempWord.Insert(i, tryme[x]);
               if (this.TestWord(tempWord.ToString()))
               {
                  result = tempWord.ToString();
                  isFoundWord = true;
                  return isFoundWord;
               }
            }
         }
      }
      return isFoundWord;
    }
```

- Split the string into two pieces after every char. If both pieces are good words make them a suggestion etc.

```
private bool TwoWords(String word, out String result)
    {
      result = "";
      bool isFoundWord = false;
      for (int i = 1; i < word.Length - 1; i++)
      {
         string firstWord = word.Substring(0, i);
         string secondWord = word.Substring(i);
         if (this.TestWord(firstWord) &&
this.TestWord(secondWord))
         {
            string tempWord = firstWord + " " + secondWord;
            result = tempWord;
            isFoundWord = true;
            return isFoundWord;
         }
      }
      return isFoundWord;
    }
```

By using multiple language dictionaries simultaneously in the spell checker, the proposed classifier can correctly recognize different languages, if we supplement component neural networks being trained these languages' character classes.

IV. EXPERIMENTS AND RESULTS

The demo program uses three well trained component CNNs which can recognize 97 % of the digit class and 90% of the upper or the lower alphabet classes of the UNIPEN online handwriting dataset [12], respectively, to identify 62 English characters set. The initial experiment, which took 450 words from 45 students who were required to carefully write 10 different words to a windows 8 touch screen device, had shown an extremely satisfactory results. Without word recognition module, the system could not recognize words properly due to the randomized of the process when choosing a possible word from a collection of recognized characters at MCNN module's outputs. The rate of exactly recognized words was lower than 30%, although the system could identify most of written characters. The main reason for this low rate was a misunderstanding of the system to similar characters "o","0" or "i","I", "l","1", etc.

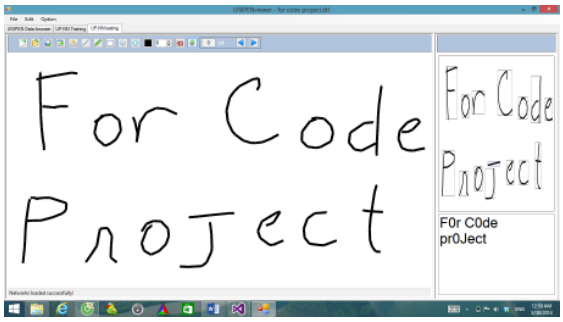

Figure 5. Recognized words without a spell checker

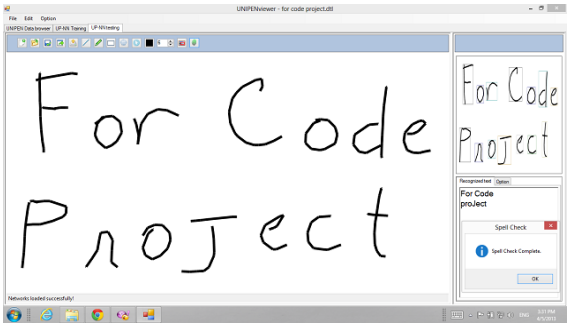

Figure 6. Recognized words with a spell checker

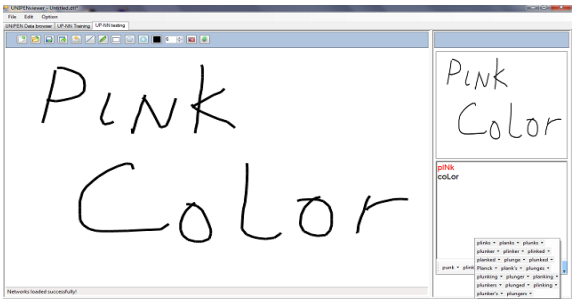

Figure 7. Word correction mode

The recognized word rate dramatically increased when the word recognition module was activated. 352 over 450 words in total were recognized exactly. The result should be higher if the system used correction function for faulty words. Figures 5 to 7 show different results to an image when the system works in different modes.

V. CONCLUSIONS AND FUTURE WORK

Although there are many limits such as writing tools, character segmentation techniques, the experimental program has shown satisfactory results to the proposed approach. Instead of using a traditional neural network, the new system uses several smaller well trained networks to recognize a large character set. The primary advantage of this approach is that the system can adapt to recognize different languages by changing component networks and dictionaries. Also, the new classifier will overcome obstacles and difficulties of traditional ones when faced with large character classes. The full project and source code can be downloaded from [13].

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition", Proceedings of the IEEE, vol. 86, no. 11, Nov. 1998, pp. 2278-2324.

[2] Y. LeCun, L. Bottou, G. Orr and K. Muller, "Efficient BackProp", in Neural Networks: Tricks of the trade, (G. Orr and Muller K., eds.), 1998.

[3] P. Y. Simard, D. Steinkraus and J. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," International Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society, Los Alamitos, 2003, pp. 958-962.

[4] P. V. Dung, “Online handwriting recognition using multi convolution neural networks”. In Proceedings of The Ninth International Conference on Simulated Evolution And Learning (SEAL 2012), Hanoi, Vietnam, December 2012, pp. 310-319.

[5] F. Lauer, Ching Y. Suen and G. Bloch, "A Trainable Feature Extractor for Handwritten Digit Recognition", Elsevier Science, February 2006.

[6] I. Guyon, L. Schomaker, R. Plamondon, R. Liberman and S. Janet, “Unipen project of on-line data exchange and recognizer benchmarks”. In proceedings of the 12th International Conference on Pattern Recognition, ICPR’94, Jerusalem, Israel, October 1994. IAPRIEEE, pp. 29–33.

[7] L. Vuurpijl, R. Niels, M. V. E. Nijmegen, “Verifying the UNIPEN devset”. In proceedings of Ninth International Workshop on Frontiers in Handwriting Recognition, IWFHR-9 2004, pp. 586 – 591.

[8] M. Parizeau, A. Lemieux, and C. Gagné, “Character Recognition Experiments using Unipen Data”. Parizeau & al., Proc. of ICDAR 2001, September 10-13. 2001 , pp. 481 – 485.

[9] List of publications by Dr. Yann LeCun. http://yann.lecun.com/exdb/publis/index.html

[10] Mike O'Neill, “Neural Network for Recognition of Handwritten Digits”. http://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi

[11] Modified NIST ("MNIST") database (11,594 KB total). http://yann.lecun.com/exdb/mnist/index.html [retreived July 2014]

[12] The UNIPEN Project. http://unipen.nici.kun.nl/ [retrived July 2014]

[13] Codeproject website: http://www.codeproject.com/Members/Vietdungiitb [retrieved July 2014]

 ISBN: 978-1-61208-371-1