Avoiding Segmentation in Multi-digit Numeral String Recognition by Combining Single and Two-digit Classifiers Trained without Negative Examples (draft - camera ready on 09/01/2008)

Dan Ciresan
Politehnica University of Timisoara
Computer Department
Timisoara, Romania
dan.ciresan@ac.upt.ro

Abstract

The objective of the present work is to provide an efficient technique for off-line recognition of handwritten numeral strings. It can be used in various applications, like postal code recognition or information extraction from fields of different forms. Proposed solution uses Convolutional Neural Networks (CNNs) to implement two classifiers, one for digit recognition and one for numeral strings composed from two digits partially overlapped. Both classifiers are trained without negative examples. By comparing the results of the classifiers it can decide if the image contains one digit or two partially overlapped digits. The use of the two-digit strings classifier completely relieves our method from the usage of segmentation. The method is evaluated on a well-known numeral strings database -NIST Special Database 19- and the results are comparable with the best results from literature, even if those are using elaborate segmentation and training with negative examples.

1. Introduction

Handwritten numerical string recognition with overlapping digits is an actual research topic due to a large number of practical applications, starting from postal code recognition ([5]) to complex document processing ([9]). This problem is more complex than recognizing isolated digits since it deals with problems such as unknown string length, segmentation, partial overlapping and others. Also, significant differences could be found comparing with the problem of recognizing handwritten words. Indeed, there is almost no contextual information available. Many of existing methods use segmentation and complex features extraction in order to obtain individual digits from the string ([14]). Then a general one-digit classifier is used. This classifier is designed to be accurate in character classification and resistant to non-character patterns. However, segmentation is a very difficult task because individual numerals in a string can overlap or touch each other. Generally, this technique is applied under two different approaches ([17]). In the first one the segmentation module transfers subsequences of isolated characters to the classifier ([18]). Oliveira et al. refine the concept by proposing an over-segmentation and under-segmentation verifier ([16]). They also propose a recognition system using heuristic over-segmentation based on support vector machines ([15]). The second approach relies on a probabilistic assumption and the final decision usually expresses the best segmentation-recognition score of the input image ([7]). It is used by Cheong et al. ([3]) to generate several segmentation hypotheses based on stroking sequences that form a possible-digit. Advanced post-processing is possible as absolute or one-to-one verifier's implementation ([10]).

In the following sections we will present the numeral string recognition system, then the experimental results. The last section is reserved for conclusion.

2. Numeral string recognition system

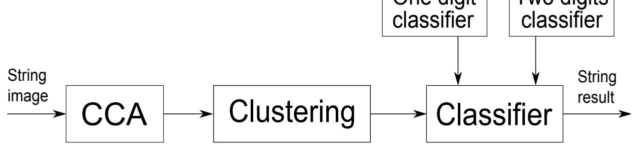

Figure 1. Numeral string recognition system (CCA = connected component analyzer).

A diagram of the numeral string recognition system is depicted in Figure 1. Each numeral string image is first cut to the bounding rectangle and then presented to Connected Component Analyzer (CCA) that extracts all connected components (CCs) from the image. The components will be clustered later to probable digit and multi-digit images in Clustering stage. Recognition is performed by the Classifier using two Convolutional Neural Networks (CNNs).

2.1. Connected Component Analyzer

All the connected components from the input image are extracted using a recursive search in only four directions, horizontally and vertically. Eight-way search is not used because this is the only place where segmentation is performed, and is desirable that the images to be thoroughly separate in as many possible components. Any unwanted segmentation will be corrected in the Clustering stage. For each component there are computed various parameters, like: size, bounding rectangle, width, height, aspect (width/height), distances to other components. The CCA also assigns a color to each component. No segmentation is performed because not even a single connected image is not split up in parts. We extract only already separated components. In contrast to other methods [13, 6, 8], we can avoid complex segmentation by using a classifier for two connected digits.

2.2. Clustering

For badly written or scanned images there are many CCs that form a single digit or a multi-digit. In order to reconstruct the image, the clustering stage performs in sequence the following four operations (the constants in all tests regarding size -i.e. number of pixels- of the components are tailored to NIST images which were scanned at 300dpi, but they can be used to images scanned with other resolution by multiply the constants with the square (because size is an area) of resize factor from one resolution to the other):

- all small components containing maximum four pixels are deleted if the closest component is at more than five pixels. They are too small and too far from any other component, so very probably they are only noise.

- any two components separated by only one pixel are concatenated if they obey one of the following conditions:

  - both have heights greater than 60% of the image and the distance between them or the overlapping distance is less than 35% of the width of the wider component

  - both have sizes greater than 200 pixels and aspects greater than 0.5 and at least one have aspect greater than 1 and the distance between them or the overlapping distance is less than 25% the width of the wider component

  - both have sizes greater than 400 pixels

  Then the components separated by two pixels are concatenated and, finally, those separated by three white pixels. This operation is also useful for poorly scanned images and for digits that were written from more than one stroke (Figure 2).

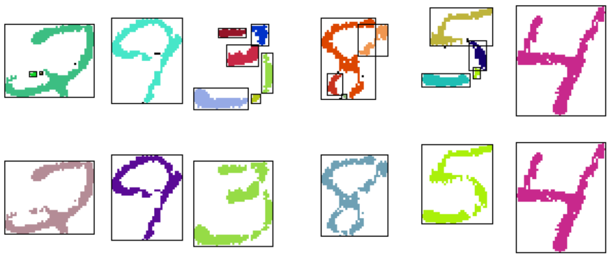

Figure 2. Before (up) and after (down) clustering phase. Each component is color coded and bounding-boxed.

- there are many persons that write the digit five from two strokes, one for the body of the digit (BD) and one for the upper part (UP). Frequently, the two strokes are very distanced horizontally and the previous clustering operations cannot reconnect them. This is a distinct case and it appears only for digit five. In order to reform the digit, we search for the upper part of a five by looking for components that: have at least ten pixels, have a horizontally elongated aspect (greater that 1.4), are not too thick (height less than 40% of the image height), and are placed in the upper half of the image. If a component of this type is detected, then the lower part of digit five is searched. The component that is suitable to represent the lower part of the digit five must have at least 100 pixels. The UP must start

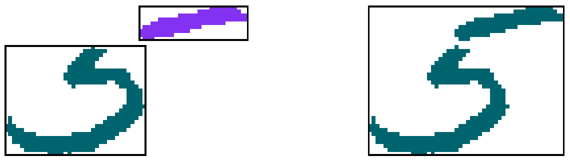

Figure 3. Reconnecting digit five

on the right part of the BD, but not further than 50%

of the width of the BD, after the end of BD. If no component is detected, the component that presumable is the upper part of a five remains un-concatenated. If both parts of a five are detected, then the upper part is translated left until is on the same vertical with the top-right-est pixel of the component that is forming the body part of the digit five (Figure 3).

- this step deals with the relatively few cases of digits written from strokes vertically separated. If two components are completely horizontally overlapped, they are concatenated.

2.3. The Classifier

The Classifier component uses two CNNs, one for single digit recognition and one for numeral strings formed from two partially overlapped digits.

The single digit CNN (1DCNN) is very similar with that from [19]. The only architectural difference is the number of convolution maps (CMs) used on first layer of convolution. We used six instead of five because we have obtained a smaller error rate with six CMs. Like in [12], hyperbolic tangent was chosen for activation function. In order to train the network to values of ± 1.0, the activation function is scaled to vary between ± 1.7159. For training we used standard backpropagation in conjunction with a second order method: stochastic diagonal Levenberg-Marquardt.

The two-digit CNN (2DCNN) is presented in our previous work [4]. 1000 different writers selected from the first four partitions of the NIST database [2] were used for training set, and 1000 for testing set. First, we concatenated the data from partitions hsf_0 to hsf_4 and we have obtained samples from 2100 writers. Then we extracted first 1000 for training set and the following 1000 for testing set. For each writer there are ten digits from what we automatically generated 200 images with two joined digits. In total, training set contains 200000 images and testing set also 200000 images. We scaled the images to 21x13 pixels. Each pixel from the image has a corresponding neuron in the input layer, L0. After experiments with various numbers of convolution maps and different numbers of neurons ([4]), we have found that the best performing NN has ten 9x5 convolution maps on layer L1, one hundred 3x1 convolution maps on layer L2 and one hundred neurons on layer L3.

3. Experimental results

We trained both NNs without negative examples. For NN used on digits we used MNIST database [11]. It is comprised from 60000 28x28 pixels digit images for the training set and 10000 digit images for the test set. In order to use CNN on MNIST data set, all the images were upscaled to 29x29 pixels. Our NN, which is similar to that of Simard [19], trained to 0.66% error rate which is very close to Simard's error rate of 0.6%. We also use elastic distortions: σ = 6.0 vs. 4.0 - Simard, and α = 0.4 vs. 0.34. The implementation details can be found in [19].

The two-digit CNN ([4]) was trained with 21x13 pixels images. The small size was chosen for faster training of the network because the training set was very large, i.e. 200000 samples. From the 200000 test samples, 10708 were unrecognized. This represents 5.35% from the test set. From overall 10708 errors only 663 (0.33%) were completely wrong, i.e. both digits were unrecognized.

All our tools for image processing, training and testing CNNs, clustering etc. were written in C++. We also use open source code [1].

We tested our classifier on images from NIST SD 19 database [2]. The state of the art for multi-digit numeral string recognition is presented in [13]. The authors from [13] compared their methods with the best on literature. In order to compare our method with their methods we have tested our recognition system on exactly the same data set. For this we extracted all three-digit images from 300 writers, nos. 1800-2099. From this 300 form pages with samples from the 300 writers, they discarded four pages written with faded ink: 1965, 1977, 2027 and 2067. We also discarded them. On each form there are five fields with three-digit strings, for a total of 1480 images for 296 writers. They reported that they further excluded four samples that mismatch the ground truth, but they have not specified what the samples are. We found four images that mismatch the ground truth (Figure 4) without any doubt. The false labeled images are: form 1902 - field 7, form 2035 - field 28, form 2056 - field 11, form 2076 - field 11. Each image was cropped to the bounding rectangle that surrounds the digits from the image.

Using the two CNNs, several recognition system were implemented. All of them take the components from the image one by one and try to recognize the digit or the digits from it. Before entering the recognition stage, all components that have less than 80 pixels or their height is smaller than 30% of the image height are discarded.

In all the subsequent algorithms rez_i represents a vector of structures, each structure containing two fields: $class$ and $score$. The index i is 1 for 1CNN and 2 for 2CNN. The vectors rez_i store the output of the CNN. For example, $rez_2[3].class$ represent the forth elements class from the result of 2CNN. The field $class$ is not redundant because is needed after the vector is sorted, in order to know what class has the best score, what class has the second best score and so on.

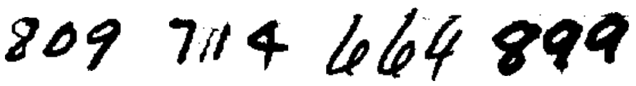

Figure 4. The three-digit images that mismatch the ground truth. First row: original images from NIST forms. Second row: ground truths.

3.1. Recognition system based on maximum score

The simplest recognition system that can be implemented takes an image and present it to 1CNN and to 2CNN. The greatest score of the CNNs are compared, and that who generated the maximum will offer both the class and the number of digits (Algorithm 1).

```
foreach component in image do
    $rez_1[\,] = 1CNN(component)$;
    $rez_2[\,] = 2CNN(component)$;
    descending sorting $rez_1$ and $rez_2$;
    if $rez_1[0].score \geq rez_2[0].score$ then
        return $rez_1[0].class$;
    end
    else
        return $rez_2[0].class$;
    end
end
```

Algorithm 1: Recognition system based on maximum score.

From 1476 three-digit images, 158 were incorrectly recognized. This translates into 89.29% recognition rate. The result for MLP (Multi Layer Perceptron) trained without negative examples from [13] is 82.52%. Considering that out method do not use segmentation at all, 89.29% correct recognized images is promising, but there is room for improvements. It is very important to one observe that this represents the proof of the efficiency of combined use of the one and two-digit CNNs versus only one-digit CNN ([13]), even if the latter it is helped by a complicated segmentation mechanism.

After study the errors of the simple recognition system, we drawn the following conclusions:

- because we trained both CNNs without negative examples, we cannot use them directly to detect if a component contains one, two or more digits. By training a NN with only positive examples, the NN will try to map any input image, even those that are incorrect, to the closest resembling class. Furthermore, probably because of the convolutional nature of the NNs, the most confusing case appear to two-digit numbers of the type $\overline{XX}$. It is very hard for both the one digit NN (1DNN) and two-digit NN (2DNN) to distinguish if an image contain a single digit or a two-digit number. For example, if the image contain only the digit 4, the score of for both NNs can be very big (and similar), the 2DNN detecting a 44!
- many images containing digit 0, 1, 3, 4, 7, 8 or 9 were recognized by the 2DNN to be 01, 11, 31, 41, 71, 81 respectively 91. The problem is attributable to the method of joining ([4]) the digits for the training set of the 2DNN, because if digit 1 is written like a line segment and is joined with a digit that in the right side has pixels on all the height of the digit, the result is very confusing (Figure 5). Similar argument can be offered for cases 10, 16, 18. This cases are denoted by $\overline{1X}$ and $\overline{X1}$.

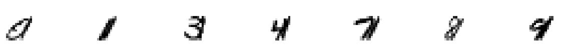

Figure 5. Cases $\overline{X1}$.

Evidently, training CNNs with negative examples would have solved many problems, but manually collecting negative examples, i. e. non-characters and non two-digit strings, is a very time consuming task. The number of negative examples must be sufficiently big to ensure a proper training. In [13], the authors trained the digit classifier with 1000 and 4000 non-characters examples. In both cases the recognition rate of the one digit classifier was reduced by 2-4%, but it had became resistant to non-character images. Our method have two classifiers, therefore we would have needed to collect negative examples for both of them. Of course, automatic generation of the negative examples can be one other way to resolve this problem, but an algorithm to generate negative example would have been devised. After evaluating all these aspects, we decided not to train the classifiers with negative examples and to develop a method that will alleviate the problems presented above in a simple as possible way.

When we looked to the errors, we observed that in the majority of cases the CNNs were right, but the relative score of the classifiers generated the misclassification.

We have trained both CNNs to be less sensitive to translation, but if the image is perfectly centered on the classifier, then the scores are better. Considering this aspect, we tried to move the image in the input field of the classifiers. The 1CNN was trained with MNIST database and the images from MNIST are mass centered. We have applied repeatedly the 1CNN on the input image translated with ± 1 pixel relative to mass center and keep the greatest score of the $3 \times 3 = 9$ tests. The 2CNN was trained with images that were bounding-box centered. Taking account of the generation method [4] of the images, we placed the 18x10 pixel image on all positions in the 21x13 input of the 2CNN. There are $(21-18+1)\times(13-10+1)=16$ possibilities. Again we retain the maximum score for each class. The number of misclassification was raising (Table 1) and the execution time was greater because of the repeated application of the CNNs. One can conclude that moving the classifiers over the images do not help the recognition process.

3.2. Recognition system based on differences of scores

```
foreach component in image do
    $rez_1[\,] = 1CNN(component)$;
    $rez_2[\,] = 2CNN(component)$;
    descending sorting $rez_1[\,]$ and $rez_2[\,]$;
    $dif_1 = rez_1[0].score - rez_1[1].score$;
    $dif_2 = rez_2[0].score - rez_2[1].score$;
    if u=1 or z=1 then
        if $dif_1 \geq dif_2$ then
            return $rez_1[0].class$;
        end
        else
            return $rez_2[0].class$;
        end
    end
    else
        if $dif_1 \geq 2 \times dif_2$ then
            return $rez_1[0].class$;
        end
        else
            return $rez_2[0].class$;
        end
    end
end
```

Algorithm 2: Recognition system based on score differences.

We stated that we cannot trust the maximum score offered by the classifier because it was not trained with negative examples. Moreover, by training a NN with only positive examples, the NN will try to map any input image to the closest resembling class. This implies that if the classifier receives an image that is the representation of one of the classes that it knows, then the corresponding output neuron will have a value close to 1, and the others considerable smaller, close to -1. We can use the fact that the difference between the best and the second best score will be very large. Considering the observation for $\overline{X1}$ cases that we made above, we implemented Algorithm 2, strengthening the condition if at least one of the digits were 1.

Contrary to the first recognition system that used only the maximum score of each CNN, in this case moving the classifiers over the input image improved the recognition rate with more than 6%, to 92.82%.

Table 1. Recognition rates (1C - moving 1CNN over the image, 2C - moving 2CNN over the image, 1+2C moving both CNNs over the image, simple - without moving)

	1C(%)	2C(%)	1+2C(%)	simple(%)
score	89.16	62.73	64.22	89.29
differences	86.85	92.61	92.82	86.65
score&diff	88.14	93.36	**93.49**	88.00

3.3. Recognition system based on both on maximum score and on differences of scores

In order to see if one can obtain better results by combining the previous methods, we implemented Algorithm 3. For each classifier we have simply added the best score with the difference between it and the second best score. The best result, **93.49%** or 96 misclassified numeral strings, was obtained when we applied translation of the classifiers. This is better with $3 \div 18\%$ than all the results from [13] obtained with various architectures of NN also trained without negative examples. pun figura cu date! Even if we compare our method with the best from [13], the difference is only 3.33%, but we do not use segmentation nor training with negative examples.

```
foreach component in image do
    $rez_1[\,] = 1CNN(component)$;
    $rez_2[\,] = 2CNN(component)$;
    descending sorting $rez_1[\,]$ and $rez_2[\,]$;
    $dif_1 = rez_1[0].score - rez_1[1].score$;
    $dif_2 = rez_2[0].score - rez_2[1].score$;
    if u=1 or z=1 then
        if $rez_1.score + dif_1 \geq rez_2.score + dif_2$ then
            return $rez_1[0].class$;
        end
        else
            return $rez_2[0].class$;
        end
    end
    else
        if $rez_1 + dif_1 \geq rez_2 + 2 * dif_2$ then
            return $rez_1[0].class$;
        end
        else
            return $rez_2[0].class$;
        end
    end
end
```

Algorithm 3: Recognition system based both on score and (on ?) differences of scores.

We can further improve our algorithm by complicate the rules for selecting 1CNN or 2CNN, but it will loose its simplicity. The analysis of the errors shows that the CNNs are very good, but the rules that select the result of one of the

CNNs are not always accurate. The 96 errors can be separated in five types:

- 4 segmentation errors. They are all caused by the very poorly scanned images.
- 8 three-digit errors. We have not addressed the cases with more than two partially overlapped digits.
- 23 errors on 1CNN. About half of them are caused by under-representation in the training set and the others by very confusing images.
- 26 errors on 2CNN. The majority of them are generated by the fact that the two digits are very different (even more than 200%) in height, and we trained the 2CNN with pair of digits that were different in height with maximum 10%.
- 63 errors generates by choosing the wrong classifier. They can be corrected by complicating the rules that select the classifier or, preferably, by training the classifiers with negative examples.

4. Conclusion

We have implemented a new method for numeral string recognition. It is based on two convolutional neural networks, one for digit recognition and the other for two-digit string recognition. By using the two-digit CNN we completely avoided the need for segmentation. Even if both CNNs were trained without negative examples, we have succeeded to overcome the majority of the negative results by implementing simple rules based on the biggest score and on the differences between the first two best scores of the networks. We have tested the methods on numeral strings from NIST SD 19. Our recognition rate of **93.49%** is better than all the result from [13] obtained with NNs trained without negative examples, and very close to the best result from [13], even if we have neither used segmentation for images nor negative examples for training the networks.

We plan to implement the CNNs on GPU for speed acceleration, and to add negative examples to the training process in order to increase the recognition rate and to further simplify the recognition system. Also, we will devise a method to resolve the cases with three or more joined digits.

References

[1] Neuralnet recognition project on the codeproject page, www.codeproject.com/kb/library/neuralnetrecognition.aspx.

[2] Nist special database 19 - nist handprinted forms and characters database, www.nist.gov/srd/nistsd19.htm.

[3] C. Cheong, K. Ho-Yon, S. Jang-Won, and J. Kim. Handwritten numeral string recognition with stroke grouping. *Proc. of the Fifth International Conference on Document Analysis and Recognition (ICDAR'99)*, (1):745–748, September 1999.

[4] D. Ciresan and D. Pescaru. Off-line recognition of handwritten numeral strings composed from two-digits partially overlapped using convolutional neural networks. *to appear in Proc. of IEEE 4th International Conference on Intelligent Computer Communication and Processing*. `http://www.cs.utt.ro/~cdanc/ICCP2008.pdf`.

[5] G. Dzuba, A. Filatov, and A. VolguninAuthor. Handwritten zip code recognition. *Proc. of the 4th International Conference on Document Analysis and Recognition (ICDAR'97)*, 2:766–770, August 1997.

[6] T. HA, J. Zimmermann, and H. Bunke. Off-line handwritten numeral string recognition by combining segmentation-based and segmentattion-free methods. *Pattern Recognition*, 31(3):257–272, 1998.

[7] T. M. Ha, M. Zimmermann, and H. Bunke. Off-line handwritten numeral string recognition by combining segmentation-based and segmentation-free methods. *Journal of Pattern Recognition*, 31(3):257–272, 1998.

[8] K. Kim, Y. Ghung, J. Kim, and C. Suen. Recognition of unconstrained handwritten numeral strings using decision value generator. *Proc. Sixth Int'l Conf. Document Analysis and Recognition*, pages 14–17, 2001.

[9] G. Koch, H. L., and P. T. Automatic extraction of numerical sequences in handwritten incoming mail documents. *Pattern Recognition Letters*, 26(8):1118–1127, June 2005.

[10] F. B. L. S. Oliveira, R. Sabourin and C. Y. Suen. Impacts of verification on a numeral string recognition system. *Pattern Recognition Letters*, 27(7):1023–1031, April 2003.

[11] Y. LeCun. The mnist database of handwritten digits http://yann.lecun.com/exdb/mnist/.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[13] C.-L. Liu, H. Sako, and H. Fujisawa. Effects of classifier structures and training regimes on integrated segmentation and recognition of handwritten numeral string. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(11):1395–1407, November 2004.

[14] L. Liu, K. Nakashima, H. Sako, and H. Fujisawa. Integrated segmentation and recognition of handwritten numerals: Comparison of classification algorithms. *Proc. of the International Workshop on Frontiers in Handwritten Recognition (IWFHR'02)*, 8:303–308, 2002.

[15] L. Oliveira and R. Sabourin. Support vector machine for handwritten numerical string recognition. *9th International Workshop on Frontiers in Handwriting Recognition (IWFHR-9)*, pages 39–44, October 2004.

[16] L. Oliveira, R. Sabourin, F. Bortolozzi, and S. C.Y. Automatic recognition of handwritten numerical strings: A recognition and verification strategy. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(11):1438–1454, November 2002.

[17] S. Ouchtati, M. Bedda, and A. Lachouri. Segmentation and recognition of handwritten numeric chains. *Journal of Computer Science*, 3(4):242–248, 2007.

[18] Z. Shi, N. Srihari, C. Y. Shin, and A. V. Ramanaprasad. A system for segmentation and recognition of totally unconstrained handwritten numeral strings. *Proc. of the 4th International Conference on Document Analysis and Recognition (ICDAR'97)*, 2:445–458, August 1997.

[19] P. Y. Simard, D. Steinkraus, and J. Platt. Best practices for convolutional neural networks applied to visual document analysis. *Proc of International Conference on Document Analysis and Recognition*.