Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Handwritten digit segmentation: Is it still necessary?

A.G. Hochuli[a], L.S. Oliveira[a,*], A.S. Britto Jr[b], R. Sabourin[c]

[a] *Federal University of Parana (UFPR), Rua Cel. Francisco H. dos Santos, 100, Curitiba, PR 81531-990, Brazil*
[b] *Pontifical Catholic University of Parana (PUCPR), R. Imaculada Conceição, 1155, Curitiba, PR 80215-901, Brazil*
[c] *Ecole de Technologie Superieure, 1100 rue Notre Dame Ouest, Montreal, Quebec, Canada*

ARTICLE INFO

Article history:
Received 3 January 2017
Revised 31 December 2017
Accepted 7 January 2018
Available online 10 January 2018

ABSTRACT

Over the last decades, a great deal of research has been devoted to handwritten digit segmentation. Algorithms based on different features extracted from the background, foreground, and contour of images have been proposed, with those achieving the best results usually relying on a heavy set of heuristics and over-segmentation. Here, the challenge lies in finding a good set of heuristics to reduce the number of segmentation hypotheses. Independently of the heuristic over-segmentation strategy adopted, all algorithms used show their limitations when faced with complex cases such as overlapping digits. In this work, we postulate that handwritten digit segmentation can be successfully replaced by a set of classifiers trained to predict the size of the string and classify them without any segmentation. To support our position, we trained four Convolutional Neural Networks (CNN) on data generated synthetically and validated the proposed method on two well-known databases, namely, the Touching Pairs Dataset and NIST SD19. Our experimental results show that the CNN classifiers can handle complex cases of touching digits more efficiently than all segmentation algorithms available in the literature.

1. Introduction

The design of most systems built to recognize unconstrained numerical strings includes image acquisition, pre-processing, segmentation, representation, and classification. One of the main bottlenecks in such a system is the segmentation module, which reads a string of digits and segments them into isolated characters. The challenge here is that a context is lacking; i.e., usually we do not know the number of digits in the string and so the optimal boundary between them is unknown.

In the last two decades several segmentation algorithms have been proposed, which rely on several heuristics, background information, foreground information, and sometimes the combination of these in order to generate potential segmentation cuts [20]. However, finding optimal segmentation cuts is difficult due to their variability in the location. To guarantee that the optimal segmentation point is generated, a strategy commonly used is the heuristic over-segmentation, whose basic idea is to segment the image as much as is necessary to produce the optimal segmentation cuts. While over-segmentation does indeed maximize the chances of generating good segmentation points, it does on the other hand considerably increase the computational cost, since the number of hypotheses that must be assessed by a classifier increases exponentially with the number of segmentation cuts.

The problem is then how to reduce the use of heuristics without increasing the number of segmentation hypotheses, or vice versa, given the lack of general rules to describe points, as well as the variability of points location. A more elaborate strategy to reduce the impacts of over-segmentation was proposed by Vellasques et al. in [25], where the goal was to filter unnecessary segmentation cuts using an SVM classifier. They succeeded in reducing the number of segmentation cuts in about 83%.

Ribas et al. [20] compared various segmentation algorithms and evaluated them in terms of performance, number of segmentation hypotheses and processing time. They also discussed the performance of the segmentation algorithms in different types and locations of connections. This characterization aimed to identify the strengths and weaknesses of different segmentation algorithms, thereby allowing the selection of the best algorithm, given a touching pair of digits. Implementing such a strategy is problematic because of the huge variability of touching pairs, owing to the unlimited number of different overlapping and touching types present.

This variability has led some authors to attempt to avoid segmentation. To the best of our knowledge, the seminal work in this regard was published by Matan et al. [17], who replicated Convolutional Neural Networks over large input fields containing unsegmented characters. This approach was named SDNN (Spatial Displacement Neural Network). Instead of producing a single output

* Corresponding author.
E-mail address: luiz.oliveira@ufpr.br (L.S. Oliveira).

https://doi.org/10.1016/j.patcog.2018.01.004
0031-3203/© 2018 Elsevier Ltd. All rights reserved.

vector, SDNN produces a series of output vectors that are used by a post-processor to pull out the best possible label sequence from the vector sequence. The authors reported a performance of 66% of correct classification on 3000 images of ZIP Codes. As stated by LeCun et al. [14], SDNN is an attractive technique, but has not yielded better results than heuristic over-segmentation methods.

Another strategy to avoid segmentation was presented by Choi and Oh [3], who trained a modular neural network composed of 100 separate subnetworks. They reported a 95.3% recognition rate on 1374 pairs of digits extracted from the NIST database. A similar notion was presented by Ciresan [4], who trained a 100-class CNN using 200,000 images, and reported a 94.65% recognition rate. The author also presented some experiments on 3-digit strings using two CNNs, one for isolated digits and the other for touching pairs. Notwithstanding the fact that three overlapping digits were not considered, a 93.4% performance was reported on 1476 3-digit strings from the NIST dataset. These above strategies were supported by the fact that most touching occurs between two consecutive digits.

With respect to unconstrained digit string recognition, we note a dominance of heuristic over-segmentation methods [2,16,19,23]. These works feature different pre-processing, segmentation, classification, and post-processing schemes. However, the common thread running through all the systems is a strong dependence on the segmentation algorithm. To avoid missing the correct segmentation point, over-segmentation is usually employed, even with its added burden of higher computational cost.

A deeper perusal of the technical literature shows that the advances in the field of machine learning, especially with the popularization and better understanding of deep learning techniques [1,10], provided great advances in different areas of handwriting recognition, such as digit recognition [6,22], character recognition [26], word recognition [21,24,27], script identification [28], and signature verification [11]. However, the recognition of handwritten digit strings is still limited by the pitfalls presented by segmentation algorithms.

In this regard, the following questions may be pertinent: Do we still need to rely on segmentation algorithms? Why do we not take advantage of the advancements that have occurred in the machine learning field and make handwriting digit recognition less dependent on segmentation algorithms? Some works in the literature [5,15,22] show that deep neural networks were able to achieve near-human performances on the traditional MNIST handwriting benchmark and other problems such as object recognition [12].

In this paper we postulate that strings of digits of any size, composed of isolated or touching digits, can be recognized without a segmentation module. Instead of relying on a segmentation algorithm and a general-purpose classifier to assess a huge number of segmentation hypotheses, we propose a framework based on four task-specific classifiers. The first one is responsible for estimating the number of touching components in the string while the remaining three are designed to discriminate 10 [0...9], 100 [00...99], and 1000 [000...999] classes. To avoid the laborious task of feature engineering, we learn the representation from synthetic data using Deep CNNs.

In order to validate such a concept and evaluate the robustness of the framework, we present experiments on the Touching Pair (TP) dataset of 79,464 touching digits proposed in [20], as well as on 11,585 numerical strings extracted from the NIST SD19 database. The experimental results on the TP dataset show that the proposed strategy surpasses all segmentation algorithms published in the literature by a fair margin, while avoiding the cost of creating and filtering segmentation hypotheses. The framework also shows its efficacy when used to classify the numerical strings of NIST SD19 ranging from two to six digits. In this case, the method achieves state-of-the-art performance without suffering the heavy burden of segmentation.

Table 1
Distribution of the data used for training and testing the classifiers. Samples are uniformly distributed among the classes.

Length/Classes	Samples	Authors	Purpose
1 (Isolated digits)	197,784	0000-2099	Training
10 classes	23,384	3850-4099	Validation
	23,621	3600-3849	Testing
2-Digit String	161,563	1000-1599	Training
100 classes	53,907	1600-1799	Validation
	55,091	1800-1999	Testing
3-Digit String	1,448,680	1000-1599	Training
1000 classes	484,346	1600-1799	Validation
	491,749	1800-1999	Testing
4-Digit String	100,000	1000-1599	Training
[a]	20,000	1600-1799	Validation
	20,000	1800-1999	Testing

[a] Data used to train the Length classifier.

2. Synthetic data

In order to efficiently learn representation from data, we had to rely on a considerable amount of samples. We thus created a synthetic dataset composed of numerical strings of sizes 2, 3, and 4. The strings are built by concatenating isolated digits of NIST SD19 [9] through the algorithm described by Ribas et al. in [20]. The SD19 database, which is an update of SD3 and SD7, is provided by the American National Institute of Standards and Technology (NIST). This database contains the full page binary images of 3699 Handwriting Sample Forms (HSFs) and 814,255 segmented handprinted digits and alphabetic characters from the forms.

To avoid building a biased dataset, we used the information on the authors available on the NIST SD19, such that digits from different authors were used exclusively for training, validation, and testing. Table 1 shows the purpose (training, validation, and testing), as well as the amount of data created[1]. Isolated digits were extracted from NIST SD19. No data augmentation was necessary since more than 240,000 isolated digits are available in this dataset.

3. Proposed framework

The system discussed in this work takes a segmentation-free approach based on three main modules: Pre-processing, Length Classifier, and Classification. Initially, an image I goes through a pre-processing module (Section 3.1) that identifies all connected components (CCs). Each CC is then classified by the Length Classifier (Section 3.2.1) which will assign to it a probability of having 1, 2, 3 or 4 touching digits. As stated earlier, most of touching occurs between two digits and sometimes between three. Strings composed of more than three touching digits are very rare. For example, we scanned the entire NIST SD19 and found very few strings with more than three touching digits. Therefore, if the Length Classifier assigns 4 to the CC, the string is rejected.

The classification module (Section 3.2.2), comprises three classifiers ($\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$) designed to discriminate 10 [0...9], 100 [00...99], and 1000 [000...999] classes. The classifiers that will be used for a given CC depends on the output of the Length Classifier, and the decisions are made in the fusion module described in Section 3.3. Depending on the confidence of the Length Classifier, more than one digit classifier may be invoked to mitigate any

[1] All the synthetic data is available upon request for research purposes at https://web.inf.ufpr.br/vri/databases-software/touching-digits/.

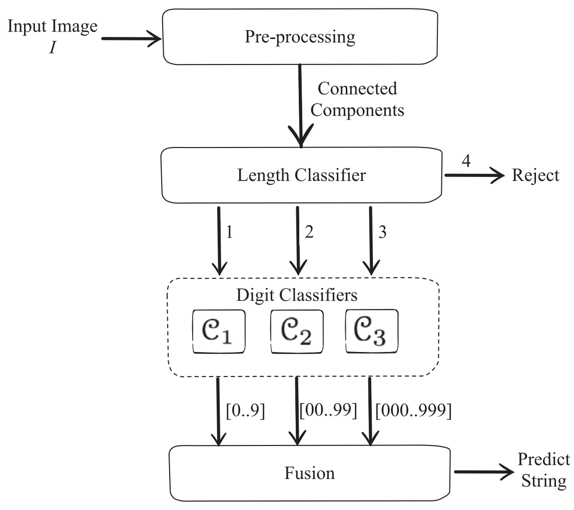


Fig. 1. Proposed framework.

possible confusions. Then, the final decision is made by combining the classification scores produced for all *CC*s found in the image *I*. Fig. 1 depicts the proposed framework which is detailed in the next subsections.

3.1. Pre-processing

The input image first goes through a pre-processing step that divides the image into groups of *CC*s. This allows us to convert the recognition of a string image to that of its partial images thus reducing the complexity of the subsequent tasks. The last pre-processing step tries to overcome the effects of fragmentation. A *CC* can represent either an integer number of characters or not. The second case is critical and can compromise the performance of the system, and should therefore be avoided. To group broken digits, we have adopted the strategy used in [19]. For the sake of completeness, the grouping technique is described in this section.

Essentially, the grouping step tries to group a character composed of several *CC*s by detecting potential parts and grouping each of them to its nearest neighbor. The median line of the image (SI_{mediam}) is used as reference. A *CC* is considered as a broken part if at least one of the following two conditions is met:

1. The *CC* does not intersect the median line (SI_{mediam}) of the numeral string.
2. $\frac{\max(CC_{above},CC_{below})}{\min(CC_{above},CC_{below})} > 5$
   where CC_{above} and CC_{below} denote the vertical height of the part above and below SI_{mediam}, respectively.

Even when a *CC* intersects with the median line, it may still be considered as broken part by the second condition if the intersection point is near to the top or bottom part of the *CC*. Thereafter, those *CC*s which are deemed to be broken parts are grouped to their neighborhood. We have to decide the neighboring *CC* to which a broken part (CC_{broken}) should be grouped. The decision is based on the following rule:

```
IF  $CC_{left} < CC_{right}$  THEN
  Group($CC_{previous}, CC_{broken}$)
ELSE
  Group($CC_{broken}, CC_{next}$)
```

If a broken *CC* is on the left or right end of the numeral string, CC_{left} or CC_{right} is set to a very high value to achieve the correct grouping. Fig. 2 defines the preceding geometric quantities. The resulting partial images are ordered from left to right according to their horizontal positions in the SI. In the case of Fig. 2, we would expect three partial images corresponding to "4", "5", and "6", respectively.

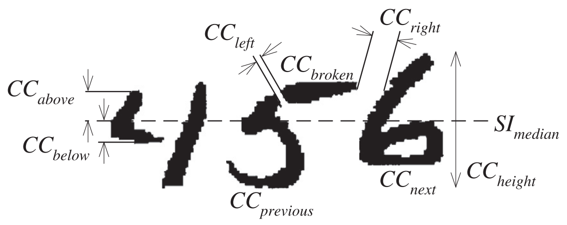


Fig. 2. Definition of various geometric quantities.

All thresholds described in this section were determined based on experimentation reported in [19]. It is worth of remark that if this module fails, all other modules will be compromised. In this context, it is natural to argue that this pre-processing module should be a robust one. However the frequency of broken digits on NIST SD19 does not demand such robustness. In this dataset, less than 5% of the digit strings contain broken digits. The strategy described in this section was able to group correctly broken components in 85% of them.

We carried out a further analysis in this module by creating 800 digit strings with poorer handwriting quality by i) cropping the images at the bottom and top (to simulate a line removal), ii) creating broken digits manually, and iii) adding random noise. Figs. 3 and 4 show some examples of the images where the pre-processing succeeded and failed, respectively.

The performance of the pre-processing module in these experiments is about the same we have observed on the NIST SD19 for images containing broken digits created manually and caused by cropping (about 85%). For the images with random noise, the performance dropped to 70%. However, with some image enhancement (e.g., morphological closing) the performance improves to the same 85% observed in the NIST SD19. In summary, if the dataset contains very poor quality handwriting with a considerable number of broken digits, a more robust pre-processing will be necessary.

3.2. Classifiers

Let x be an input image (*CC*) that should be assigned to one of the ω classes. $\mathcal{C}$ means the classifier and $\mathcal{C}(x) = p^i(x) | \forall_i (1 \leqslant i \leqslant \omega)$ means that the classifier $\mathcal{C}$ assigns the input x to each class i with a probability value $p^i(x)$. This definition is used for all classifiers of the system.

As mentioned earlier, all the classifiers used in this work are CNNs that are constructed using multiple layers considering the following operations: convolutions, max-pooling, and dot products (fully-connected layers), where convolutional layers and fully connected layers have learnable parameters that are optimized during training. With the exception of the last layer in the network, after each learnable layer we apply ReLU non-linearity. The last layer uses the softmax non-linearity.

Training is performed with the Stochastic Gradient Descent (SGD) using back-propagation with mini-batches of 256 instances, a momentum factor of 0.9 and a weight decay of 5×10^{-4}. The learning rate is set to 10^{-2} in the beginning to allow the weights to quickly fit the long ravines in the weight space, after which it is reduced over the time (until 5×10^{-4}) to make the weights fit the sharp curvatures. The network makes use of the well known cross-entropy loss function.

In the present work, regularization was implemented through early-stopping, which prevents overfitting from interrupting the training procedure once the performance of the network on a validation set deteriorates. During training, the performance of the

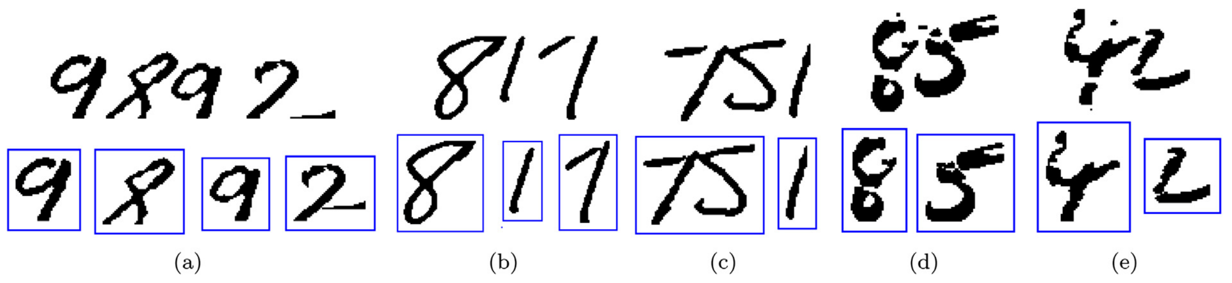


Fig. 3. Broken digits correctly grouped: (a) [9892] bottom part cropped removing part of the digits that may cause broken digits (b) [817] upper part cropped removing part of the digits that may cause broken digits, (c) [751] broken digits created manually (d) and (e) [85,42] random noise causing broken digits.

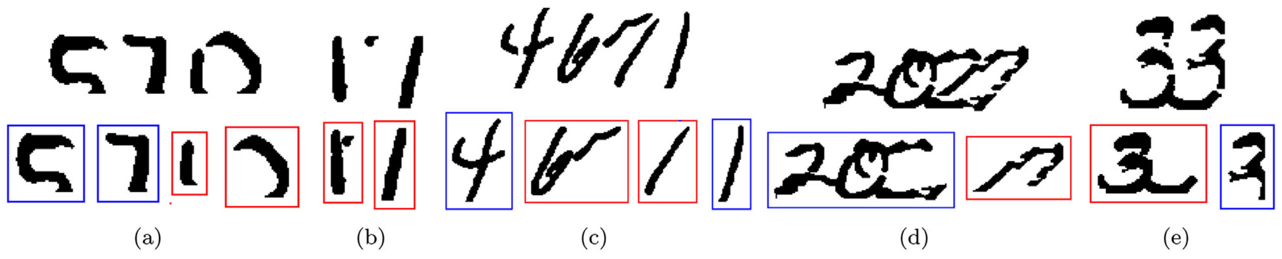
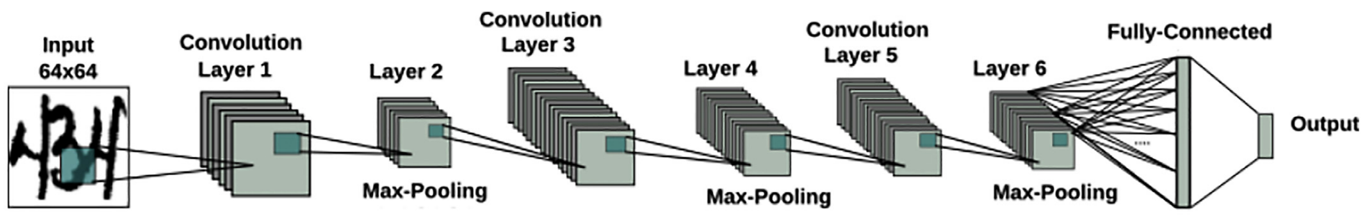


Fig. 4. Broken digits not grouped correctly: (a) [570] bottom part cropped removing part of the digits that may cause broken digits (b) [17] upper part cropped removing part of the digits that may cause broken digits, (c) [4671] broken digits created manually (d) and (e) [2027,33] random noise causing broken digits.



Models	Input	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Fully Connected	Output
Length	64x64 Binary	Convolution 5x5@1@6	MaxPooling 2x2@2@6	Convolution 5x5@1@32	MaxPooling 2x2@2@32	Convolution 3x3@1@20	MaxPooling 2x2@2@20	72	4

Fig. 5. CNN architecture for $\mathcal{L}$. Layer parameters are represented as Kernel Size @ Stride @ Feature Maps.

network on the training set will continue to improve, but its performance on the validation set will only improve up to a certain point, where the network starts to overfit the training data; at that point, the learning algorithm is terminated. To implement the CNN models we have used the Caffe framework [13] on an NVidia GeForce GTX Titan Black GPU[2].

3.2.1. Length classifier

The length classifier ($\mathcal{L}$) was designed to predict the length of the *CC*. We have tested several different architectures for this classifier but the one that yielded the best results was based on the well-known LeNet 5 [14]. The final architecture contained three convolutional layers followed by max pooling layers. This architecture, which was defined empirically on the validation set, is depicted in Fig. 5.

The classifier was trained using the protocol described in Section 3.2 using 400,000, 79,157 and 79,742, samples (uniformly distributed) for training, validation, and testing, respectively. Using the Caffe framework and the hardware mentioned in Section 3.2, it took about 90 minutes to train this model over 30,000 iterations.

Table 2
Confusion matrix (%) for the $\mathcal{L}$ on the testing set.

	(1)	(2)	(3)	(4)
(1)	99.9	0.01		
(2)	0.02	99.2	0.07	
(3)		0.9	96.9	2.3
(4)			2.3	97.7

Classifying a single input image takes about 0.4 milliseconds (ms). In our experiments, the best results were achieved when the input image was resized to 64×64 pixels. The recognition rate on the testing set was 98.4% and 99.9% for Top-1 and Top-2, respectively. Table 2 shows the confusion matrix.

Analyzing the confusions resulting from $\mathcal{L}$ we conclude that the number and location of the vertical strokes seem to bear important information needed to determine the size of the string. For example, single digits that are classified as 2-digit string are often slashed zeros, zeros with missing parts, and the digit “6” similar to those presented in Fig. 6a and b. Digits that are almost overlapping such as the “3” and “9” in Fig. 6c and strings with several vertical strokes close together such as in the “44” in Fig. 6d are also sources of confusion.

[2] All trained classifiers are available for research purposes at https://web.inf.ufpr.br/vri/databases-software/touching-digits/

Table 3
Data used to train the digit classifiers.

Classifier	Number of	Amount of data (×1000) for			Source	Training	Classification
	Classes	Train	Validation	Testing		Time (min)	Time (ms)
($\mathcal{C}_1$)	10	197	23	23	NIST SD 19	70	0.57
($\mathcal{C}_2$)	100	161	53	55	Synthetic data	90	0.60
($\mathcal{C}_3$)	1000	1448	484	491	Synthetic data	200	0.63

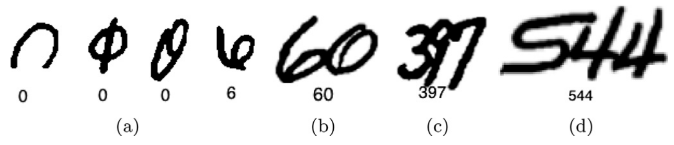


Fig. 6. Some images misclassified by $\mathcal{L}$: (a) single digit classified as 2-digit string, (b) 2-digit classified as 3-digit string, (c) 3-digit classified as 2-digit string, and (d) 3-digit classified as 4-digit string.

3.2.2. Digit classifiers

To recognize isolated digits, and 2- and 3-digit strings, we used an architecture depicted in Fig. 7. The three CNNs, which also are based on the LeNet 5 [14], share the same structure but with different numbers of filters, kernel sizes, and strides. Fig. 7 summarizes the parameters used in all three classifiers, which were defined empirically on the validation set.

Table 3 shows the amount of data used for training, validation, and testing for all three classifiers. It also shows training (30,000 iterations) and classification time using the Caffe framework and the hardware mentioned in Section 3.2.

All three classifiers were trained using the protocol described in Section 3.2 and yielded the accuracies reported in Table 4. As we can observe, all three classifiers achieved high performances on the testing set, showing that the CNN is able to learn good representation from data for the three different classes of problems.

Table 4
Recognition rate of the digit classifiers on testing set.

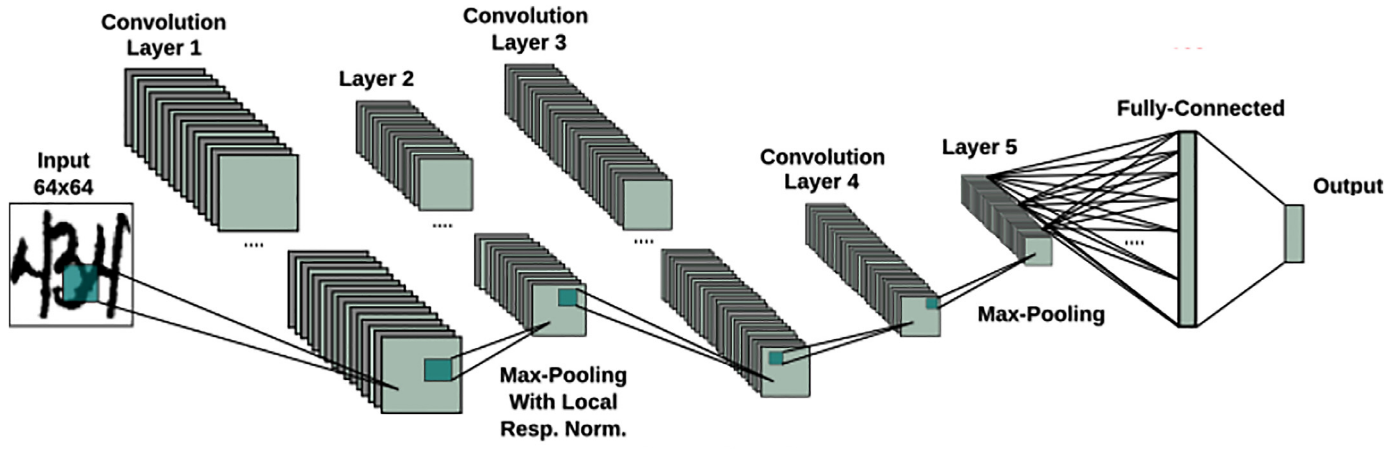
Classifier	Top 1	Top 2
Isolated Digits ($\mathcal{C}_1$)	99.6	99.9
2-Digit ($\mathcal{C}_2$)	99.7	100.0
3-Digit ($\mathcal{C}_3$)	97.7	98.9

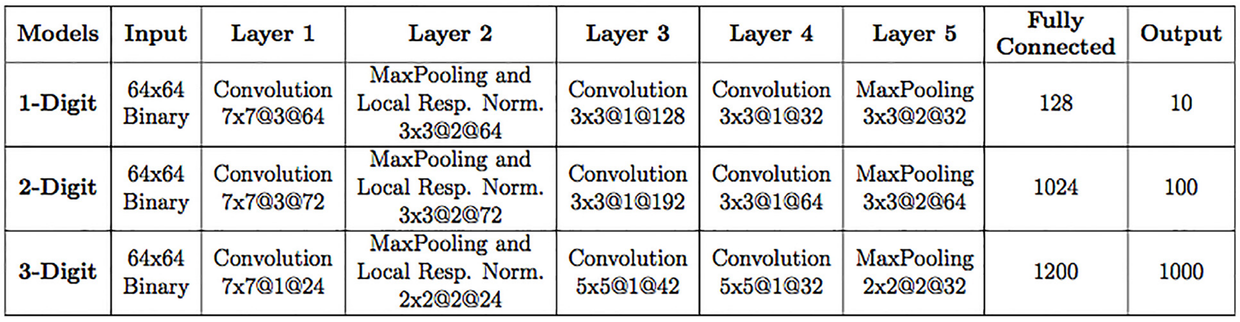
3.3. Fusion

The confusion matrix presented in Table 2 shows that some of the confusions caused by the classifier $\mathcal{L}$ can also be recovered by considering its second highest output (Top-2). With that in mind, we have proposed a fusion rule that consider the Top-2 outputs of $\mathcal{L}$.

Let $\mathcal{L}^i(x) = p^i(x)$ be the probability of the input pattern x be composed of i, $(i = 1, 2, 3, 4)$ digits. Let $\mathcal{C}_1(x) = \max_{0\le i\le 9} p^i(x)$, $\mathcal{C}_2(x) = \max_{0\le i\le 99} p^i(x)$, and $\mathcal{C}_3(x) = \max_{0\le i\le 999} p^i(x)$ be the probability produced by 10-class, 100-class, and 1000-class classifiers, respectively, for the input pattern x. Let Top1($\mathcal{C}$) and Top2($\mathcal{C}$) be the functions that return the classes with first and second highest scores of a given classifier $\mathcal{C}$, respectively. Then, x is assigned to the class ω_j according to Eq. (1),

$$P(\omega_j|x)\begin{cases}\text{if } \mathcal{L}(x) < T, & \max(\mathcal{C}_{Top1(\mathcal{L})}(x), \mathcal{C}_{Top2(\mathcal{L})}(x)) \\ \text{otherwise}, & \mathcal{C}_{Top1(\mathcal{L})}(x)\end{cases} \quad (1)$$





Models	Input	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Fully Connected	Output
1-Digit	64x64 Binary	Convolution 7x7@3@64	MaxPooling and Local Resp. Norm. 3x3@2@64	Convolution 3x3@1@128	Convolution 3x3@1@32	MaxPooling 3x3@2@32	128	10
2-Digit	64x64 Binary	Convolution 7x7@3@72	MaxPooling and Local Resp. Norm. 3x3@2@72	Convolution 3x3@1@192	Convolution 3x3@1@64	MaxPooling 3x3@2@64	1024	100
3-Digit	64x64 Binary	Convolution 7x7@1@24	MaxPooling and Local Resp. Norm. 2x2@2@24	Convolution 5x5@1@42	Convolution 5x5@1@32	MaxPooling 2x2@2@32	1200	1000

Fig. 7. CNN architecture for isolated, 2- and 3-digit strings. Layer parameters are represented as Kernel Size @ Stride @ Number of Feature Maps.

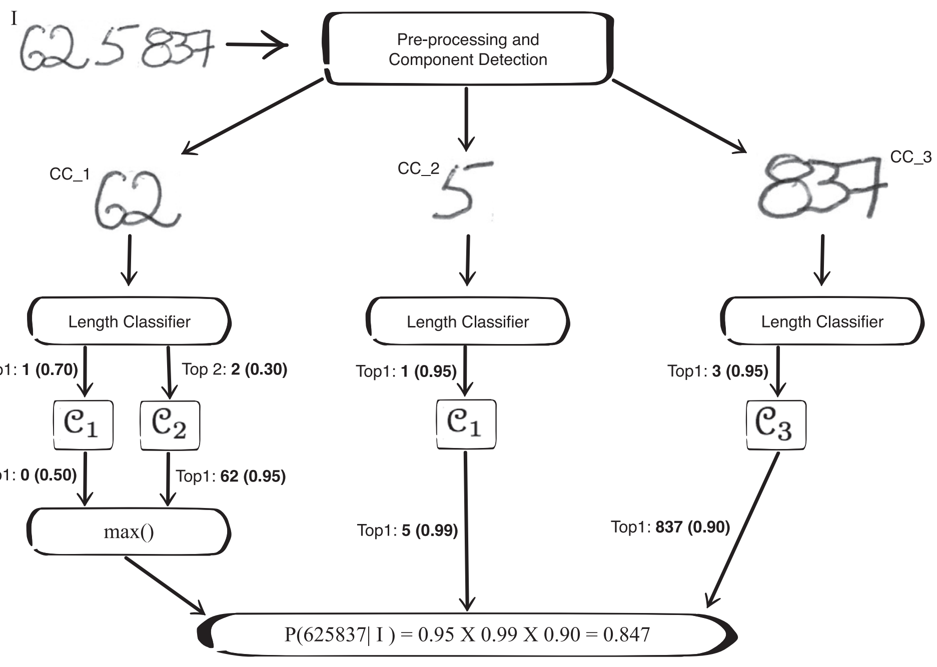


Fig. 8. Example of the fusion strategy.

where T is a threshold defined empirically on the validation set.

Considering that the input image I may contain n connected components, the most probable interpretation of the written amount M is given by Eq. (2).

$$P(M|I) = \prod_{i=1}^{n} P(\omega_j|x_i) \tag{2}$$

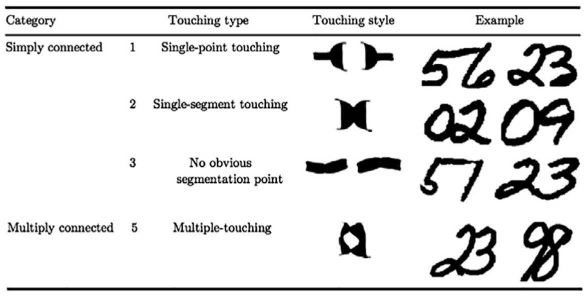
Fig. 8 exemplifies the entire process from pre-processing to the final decision. In this case, the string consists of four CCs. After pre-processing, the broken digit “5” composed of two strokes is grouped, thus creating in this way three CCs (“62”, “5”, and “837”). Then, they are classified by $\mathcal{L}$, which assigns two hypotheses to them for their lengths. These two hypothesis (Top-1 and Top-2) may be used to select the digit classifiers, as depicted in Fig. 8. In this example, $\mathcal{L}$ misclassifies CC_1 by assigning it to class 1 (digits) instead of 2 (digits). However, the output of $\mathcal{L}$ is smaller than T, which means that the final output for CC_1 will be given by $\max(\mathcal{C}_1(CC_1), \mathcal{C}_2(CC_1))$. For the other CCs, only the Top-1 classifiers are used since $\mathcal{L}$ produces scores greater than T.

4. Experiments

In order to validate the proposed framework, we elaborate two sets of experiments. In the first one, the goal is to compare the proposed segmentation-free approach with traditional segmentation-recognition algorithms. To accomplish that, we have used the 79,464 images of touching digits available in the Touching Pairs (TP) database. In the second scenario, we tested the method on more than 11,000 numerical strings ranging from 2 to 6 digits extracted from NIST SD19.

4.1. Synthetic data

When evaluating the segmentation algorithms, the authors in [20] were interested in knowing whether or not the segmentation cuts produced by the algorithms were the good ones, independently of their quantity. For the algorithms based on the segmentation-recognition approach, this task is straightforward,



Category		Touching type	Touching style	Example
Simply connected	1	Single-point touching		
	2	Single-segment touching		
	3	No obvious segmentation point		
Multiply connected	5	Multiple-touching		

Fig. 9. Types of connected numeral string (extracted from [20]).

since there is only one hypothesis to be assessed. For those algorithms based on over-segmentation, all the cuts must be assessed. In the latter case, the strategy used is as follows: if there are two digits among the hypotheses (using a classification engine) that match to the ground truth, the segmentation is considered successful. It is clear that this strategy considers the best case scenario since all misclassifications due to over- and under-segmentation are not considered.

As indicated earlier, we assume that the size of the string is unknown. So, $\mathcal{C}_2$ is only used to classify the images that were assigned as 2-digit string by $\mathcal{L}$. Otherwise, we count as an error. Thus, there may be two sources of errors, i.e., a wrong estimation of the number of digits in the string ($\mathcal{L}$) or a misclassification of the string ($\mathcal{C}_2$).

Table 5 summarizes the results reported in [20] and [7] where the authors compare several segmentation algorithms in terms of correct segmentation on the TP database. Besides the overall performance, this table also shows the performance depending on the connection types depicted in Fig. 9.

Table 5 also allows us to draw some conclusions. Algorithms based on a single segmentation hypothesis (segmentation cuts = 1) usually fail in more complex touching cases (e.g., type V) since a single segmentation cut is very often not enough to correctly split the digits. Algorithms based on multiple cuts, on the other hand, achieve better performance in terms of finding the cor-

Table 5
Performance of the segmentation algorithms (reported in [20] and [7]), in terms of correct segmentation, on the TP database.

Method	Performance	Connection type (%)				Segmentation
	%	I	II	III	V	Cuts
Shi and Govindaraju (1997)	59.30	68.31	59.72	60.35	25.44	1
Congedo et al. (1995)	63.07	62.88	67.51	59.40	40.45	1
Lacerda and Mello (2013)	65.79	71.75	71.21	63.64	56.57	1
Elnagar and Alhajajj (2003)	67.34	63.88	71.51	56.40	58.73	1
Pal et al. (2003)	71.21	73.96	74.69	80.09	41.52	1
Oliveira et al. (2000)	88.03	90.40	90.78	89.01	64.88	1
Fusijawa et al. (1992)	89.85	95.45	91.27	83.57	63.72	3.66
Fenrich and Krishnamoorthy (1990)	92.37	97.54	93.79	99.45	65.57	4.07
Gattal and Chibani (2015)	93.24	96.67	93.75	99.68	77.58	24.11
Chen and Wang (2000)	93.80	97.87	94.23	97.55	76.76	45.40
Proposed method	97.12	97.02	97.89	98.97	93.03	0

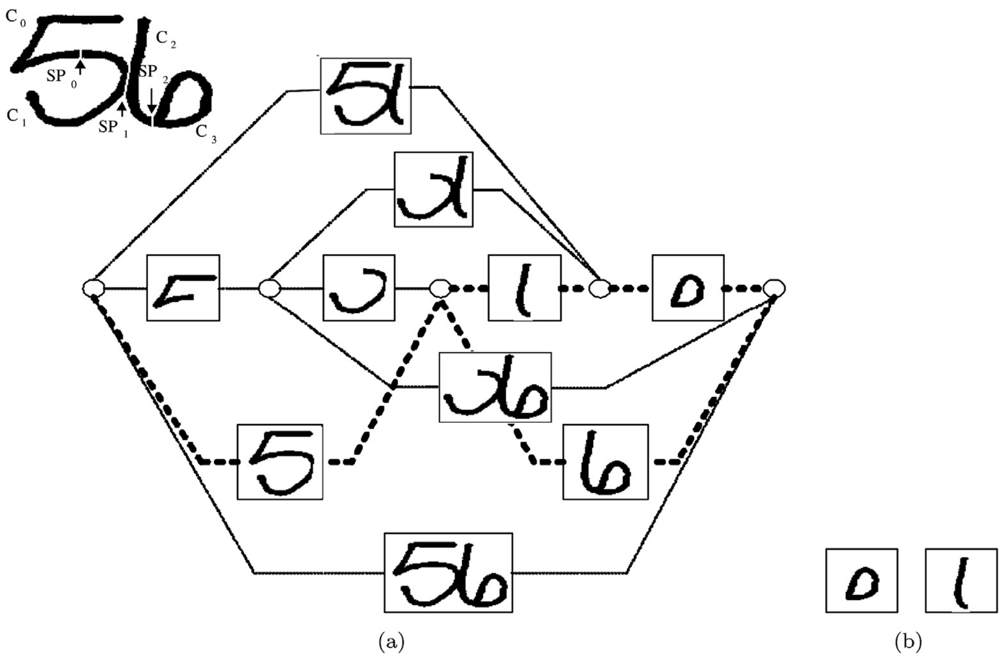


Fig. 10. (a) Segmentation paths for the string “56” and (b) Images that can be easily confused with digits “0” and “1” [25].

rect segmentation cut, but with the computational cost of having to evaluate several hypotheses.

This problem is exemplified in Fig. 10. In this example the segmentation algorithm produced three segmentation hypotheses. Accordingly, we may have to evaluate up to 10 different segmentation hypotheses, and then find the one that maximizes the output in the segmentation graph. What happens very often is that some over-segmented pieces, such as those depicted in Fig. 10b can be misclassified with high scores. In that case, the path “510” may produce a higher score than the path “56”. To tackle this, authors make use of heuristics or even more elaborated filters to reduce the number of segmentation hypotheses [25]. In summary, dissecting the image into several pieces by creating a huge number of segmentation points may produce a good segmentation cut at some point; however, the cost of filtering and dealing with over-segmentation may be prohibitive in real cases.

In this context, the proposed approach shows only advantages, compared to traditional segmentation algorithms. The expensive process of finding the segmentation cuts, filtering out unlikely hypotheses, and classifying the remaining ones is replaced by two classifier calls ($\mathcal{L}$ and $\mathcal{C}_2$). Furthermore, this segmentation-free approach achieves the highest performance (97.12% recognition rate) when compared to all methods reported in the literature. The total error (2.88%) can be divided into length classification (1.78%) and string classification (1.10%). Fig. 11 shows some images that were misclassified by $\mathcal{C}_2$. As reported in Table 5, the poorest performance (93%) is achieved on type V (multiple touching), which shows the highest variability. However, when compared to others, such a performance is outstanding.

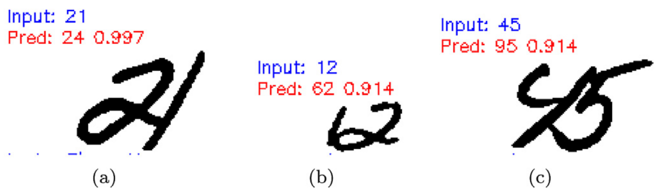


Fig. 11. Some images not recognized - Input:Prediction[Probability]: (a) 21:24[0.997], (b)12:62[0.914], (c)45:95[0.914].

4.2. NIST SD19

The experiments using numeral strings are based on 11,585 numeral strings extracted from the hsf_7 series and distributed into five classes: 2_digit (2,370), 3_digit (2,385) 4_digit (2,345), 5_digit

Table 6
Recognition rates for the NIST strings.

Length	Samples	Recognition	Error			4-digit
		Rate (%)	Pre-Processing	Length	Misclassification	Rejection
2	2370	97.6	0.2	0.5	1.7	0.0
3	2385	96.2	0.5	0.6	2.7	0.0
4	2345	94.6	0.8	0.8	3.5	0.3
5	2316	94.1	1.1	1.5	3.2	0.1
6	2169	93.3	1.3	1.5	3.6	0.3
	Average	95.2	0.8	1.0	2.9	0.1

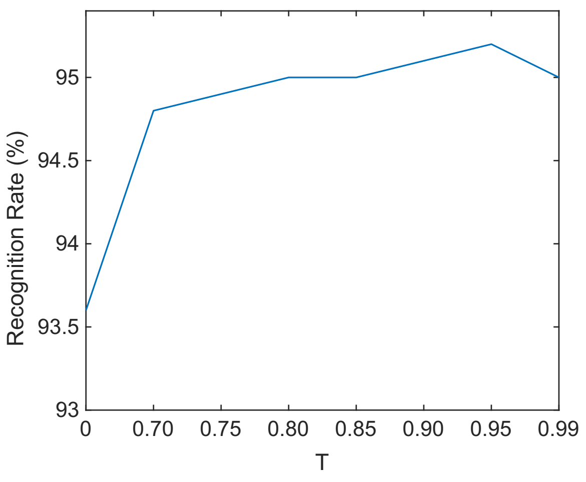


Fig. 12. Sensitivity analysis of parameter T from Eq. (1).

(2,316), and 6_digit (2,169) strings, respectively. These data exhibit different problems such as touching and fragmentation and they were also used as a test set in [2,16,19,23]. It is important to mention that at any moment the authors of hsf_7 were not used for training.

Table 6 summarizes the results for this experiment where the threshold value T from Eq. (1) was set to 0.95 (defined empirically on the validation set). Fig. 12 shows how sensitive the performance is to changes in T. As can be seen, a very similar performance is achieved for T ranging from 0.85 to 0.95. Besides the recognition rates, Table 6 also shows how the error is distributed among the three modules of the system, i.e., pre-processing (grouping), length classifier, and digit classifiers. The last column shows the percentage of strings classified as 4-digit that were rejected by the system.

As we can see, the main sources of error are i) digit misclassification, ii) length misclassification, and iii) pre-processing. The number of strings classified as 4-digit, hence rejected, is very small. Figs. 13(a), (b), and (c) show some examples of misclassified images, while Figs. 13(d), (e), and (f) show examples of images that were correctly recognized by the system.

Fig. 14 depicts the number of classifier calls per string size. Since most of the strings of NIST SD19 contains only isolated digits, the number of requests for classifiers $\mathcal{L}$ and $\mathcal{C}_1$ is roughly equal to the number of characters to be recognized. Classifiers $\mathcal{C}_2$ and $\mathcal{C}_3$, on the other hand, are only requested a few times by the system, including the times they are used as the second option (Top2) in order to resolve some confusion. The results show that, although all classifiers were trained on synthetic data (except $\mathcal{C}_1$), the performance of the system on real data is compelling.

Table 7 compares the recognition rates of several systems published in the literature on NIST SD19. In the first part of the table we group those works that used the same number of strings. Britto Jr. et al. [2] used a two-stage HMM-based recognition method to compensate for any possible loss in terms of recognition performance caused by the necessary trade-off between segmentation and recognition in an implicit segmentation-based strategy. Oliveira et al. [19] proposed a system based on over-segmentation,

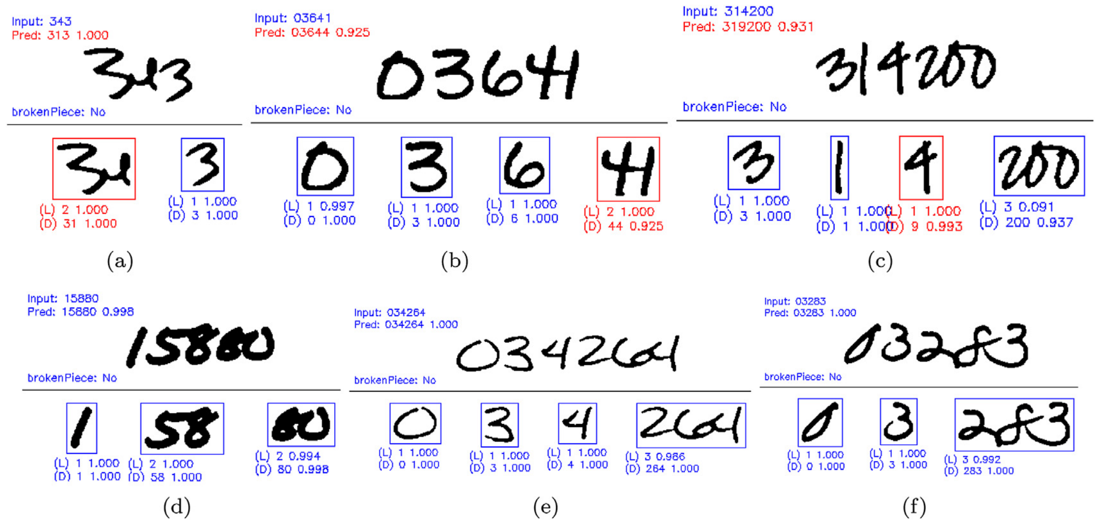


Fig. 13. Examples of digit strings (NIST SD19): (a), (b) and (c) Not recognized and (d), (e) and (f) recognized. The upper-part of each figure contains the original string with its label (input) and the output of the system (pred) with is final probability. The lower-part, contains the connected components and the results of the length classifier and digit classifiers. A red bounding box indicates those components misclassified. For example, in (a), the first connected component, which is a 34, was classified correctly the $\mathcal{L}$ but $\mathcal{C}_2$ assigned 31 with a very high score instead of 34. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

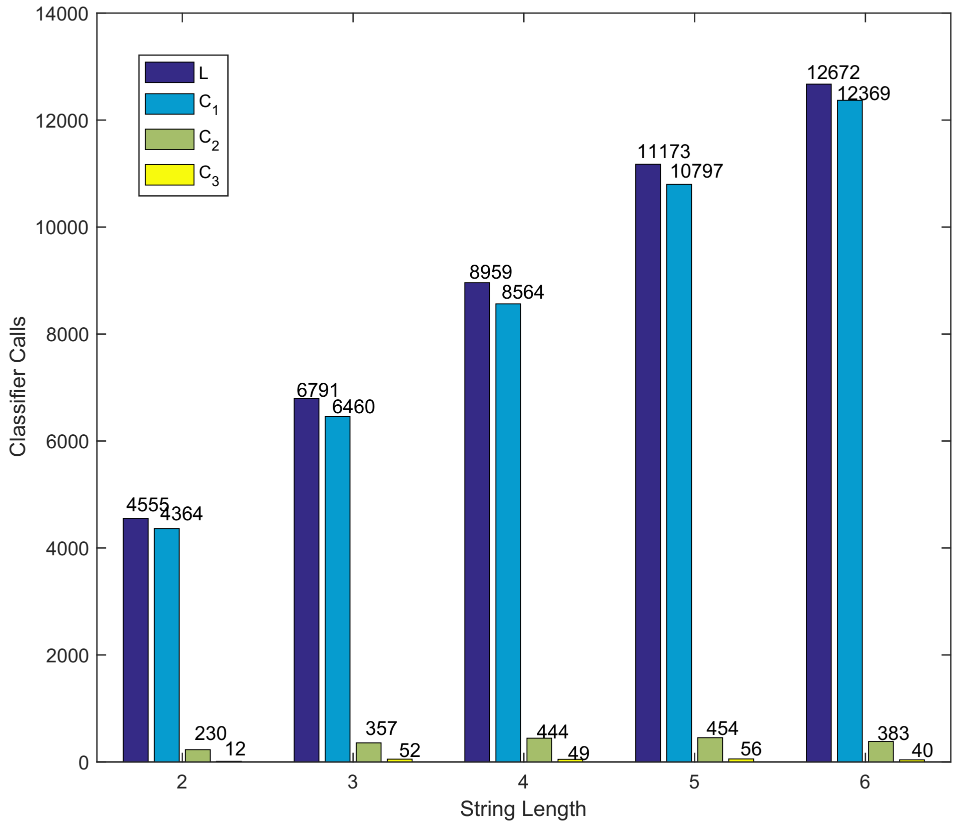


Fig. 14. Number of classifier calls per string size.

Table 7
Comparison of the recognition rates on NIST SD19.

Length	Samples	Britto et al. [2]	Oliveira et al. [19]	Oliveira et al. [18]	Sadri et al. [23]	* Sadri et al. [23]	Gattal et al. [8]	Proposed method	Samples	Liu et al. [16]	Ciresan et al. [4]
2	2370	94.8	96.8	97.6	95.5	98.9	99.0	97.6			
3	2385	91.6	95.3	96.2	91.4	97.2	97.3	96.2	1476	96.8	93.4
4	2345	91.3	93.3	94.2	91.0	96.1	96.5	94.6			
5	2316	88.3	92.4	94.0	88.0	95.8	95.9	94.1			
6	2169	89.0	93.1	93.8	88.6	96.1	96.6	93.3	1471	96.7	

with some modules designed to reduce the number of segmentation hypotheses. All classifiers were multi-layer perceptrons (MLP). The same authors in [18] replaced the MLPs by Support Vector Machines (SVMs) and obtained an improvement of one percentage point in general.

Sadri et al. [23] also proposed a system based on over-segmentation, in which they deal with multiple segmentation hypotheses as an optimization problem. To solve the optimization problem, they applied a genetic algorithm, thereby adding in this way another layer of complexity to the system. The authors in [23] show a second set of experiments (marked with * in Table 7) where they define a set of heuristics, at the top of the Genetic Algorithm, to deal with multiple segmentation hypotheses. They show an important improvement, but the results are somehow biased since the heuristics were defined using a subset of the testing set.

Another costly over-segmentation algorithm has been recently proposed by Gattal et al. [8], and combines contour information, skeleton, and Radon transform on a sliding window. The authors report very interesting recognition rates on NIST SD19; however, a considerable number of segmentation parameters appear to be adjusted on the testing set, which makes it difficult to assess the results. The authors only mention a validation set composed of isolated digits used to train the Support Vector Machines used for classification.

A deeper analysis of the aforementioned systems may show that they all face two bottlenecks, namely, creating multiple segmentation hypotheses and then proposing some strategy to reduce them in order to allow the string to be recognized in a reasonable amount of time. Both problems, which are byproducts of the segmentation, are avoided by the proposed method. The non-dependence of segmentation, coupled with representation learning, make the system design simpler since there is no need to design hand-crafted features for either segmentation or recognition. Furthermore, the performance of the system is comparable to what obtains in the literature.

5. Conclusion

Our aim in this work was to answer the title question, i.e., Is digit string segmentation still necessary in order to build a robust reading system for unconstrained numerical strings? We have demonstrated through a series of comprehensive experiments on two datasets that digit segmentation can be successfully replaced by CNN classifiers trained on synthetic data for two specific tasks, i.e., string length classification and digit classification.

The digit classification comprises three classifiers that are responsible for classifying isolated digits, and 2-digit, and 3-digit strings. These classifiers are used according to the output of the length classifier. To avoid a hard decision and overcome some of the confusion caused by the length classifier, the fusion method may use the outputs of two digit classifiers to produce the final decision.

Experiments on the TP database highlight the advantages of the proposed method by achieving a 97% of recognition rate. The closest result reported in the literature reaches 93.8% of correct segmentation but with an associated cost of having to assess a huge number of hypotheses that are created by more than 45 segmentation cuts. In the experiment on NIST SD19, where most of the strings contain only isolated digits, the method compares favorably against others published in the literature that are based on some kind of heuristic over-segmentation technique.

For future works, we plan to deploy the system on a real dataset of numerical string that we are currently building.

Acknowledgements

This research has been supported by The National Council for Scientific and Technological Development (CNPq) grant 303513/2014-4. In addition, we gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] Y. Bengio, A. Courville, P. Vincent, Representation learning: a review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1798–1828.
[2] A. Britto Jr, R. Sabourin, F. Bortolozzi, C.Y. Suen, The recognition of handwritten numeral strings using a two-stage HMM-based method, Int. J. Doc. Anal. Recognit. 5 (2) (2003) 102–117.
[3] S. Choi, I. Oh, A segmentation-free recognition of two touching numerals using neural networks, in: Proc. of 5th International Conference on Document Analysis and Recognition, 1999, pp. 253–256. Bangalore, India
[4] D. Ciresan, Avoiding segmentation in multi-digit numeral string recognition by combining single and two-digit classifiers trained without negative examples, in: 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2008, pp. 225–230.
[5] D. Ciresan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 3642–3649, doi:10.1109/CVPR.2012.6248110.
[6] R.S.N. Das, A. K.Saha, M. Nasipuri, A multi-objective approach towards cost effective isolated handwritten Bangla character and digit recognition, Pattern Recognit. 58 (2016) 172–189.
[7] A. Gattal, Y. Chibani, SVM-based segmentation-verification of handwritten connected digits using the oriented sliding window, Int. J. Comput. Intell. Appl. 14 (1) (2015) 1–17.
[8] A. Gattal, Y. Chibani, B. Hadjadji, Segmentation and recognition system for unknown-length handwritten digit strings, Pattern Anal. Appl. 20 (2) (2017) 307–323.
[9] P.J. Grother, NIST Special Database 19 - Handprinted forms and characters database, NIST, 2016.
[10] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, T. Shen, Recent advances in convolutional neural networks, Pattern Recognit. (2017).
[11] L.G. Hafemann, R. Sabourin, L.S. Oliveira, Learning features for offline handwritten signature verification using deep convolutional neural networks, Pattern Recognit. (2017) 163–176.
[12] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, arXiv preprint arXiv:1512.03385(2015).
[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: convolutional architecture for fast feature embedding, arXiv preprint arXiv:1408.5093(2014).
[14] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[15] Z. Liao, G. Carneiro, A deep convolutional neural network module that promotes competition of multiple-size filters, Pattern Recognit. 71 (2017) 94–105.
[16] C.-L. Liu, H. Sako, H. Fujisawa, Effects of classifier structures and training regimes on integrated segmentation and recognition of handwritten numeral strings, IEEE Trans. Pattern Anal. Mach. Intell. 26 (11) (2004) 1395–1407.
[17] O. Matan, J.C. Burges, Y. LeCun, J.S. Denker, Multi-digit recognition using a space displacement neural network, in: J.E. Moody, S.J. Hanson, R.L. Lippmann (Eds.), Advances in Neural Information Processing Systems, 4, Morgan Kaufmann, 1992, pp. 488–495.
[18] L.S. Oliveira, R. Sabourin, Support vector machines for handwritten numerical string recognition, in: 9th International Workshop on Frontiers in Handwriting Recognition, 2004, pp. 39–44.
[19] L.S. Oliveira, R. Sabourin, F. Bortolozzi, C.Y. Suen, Automatic recognition of handwritten numerical strings: a recognition and verification strategy, IEEE Trans. Pattern Anal. Mach. Intell. 24 (11) (2002) 1438–1454.
[20] F.C. Ribas, L.S. Oliveira, A.S. Britto, R. Sabourin, Handwritten digit segmentation: a comparative study, Int. J. Doc. Anal. Recognit. 16 (2) (2013) 567–578.
[21] P. Roy, A. Bhunia, A. Das, P. Dey, U. Pal, HMM-based Indic handwritten word recognition using zone segmentation, Pattern Recognit. 60 (2016) 1057–1075.
[22] S. Sabour, N. Frosst, G. Hinton, Dynamic routing between capsules, Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017.
[23] J. Sadri, C.Y. Suen, T.D. Bui, A genetic framework using contextual knowledge for segmentation and recognition of handwritten numeral strings, Pattern Recognit. 40 (2007) 898–919.
[24] Z. Tamen, H. Drias, D. Boughaci, An efficient multiple classifier system for Arabic handwritten words recognition, Pattern Recognit. Lett. 93 (123–132) (2017).
[25] E. Vellasques, L.S. Oliveira, A.S.B. Britto Jr, A. Koerich, R. Sabourin, Filtering segmentation cuts for digit string recognition, Pattern Recognit. 41 (10) (2008) 3044–3053.
[26] X. Xiao, L. Jin, Y. Yang, W. Yang, J. Sun, T. Chang, Building fast and compact convolutional neural networks for offline handwritten Chinese character recognition, Pattern Recognit. 72–81 (2017).
[27] F. Yina, Y. Wua, C.L. Liu, Improving handwritten Chinese text recognition using neural network language models and convolutional neural network shape models, Pattern Recognit. 65 (2017) 251–264.
[28] F. Ziyong, Y. Zhaoyang, J.L.H. Shuanping, S. Jun, Robust shared feature learning for script and handwritten/machine-printed identification, Pattern Recognit. Lett. 100 (6–13) (2017).

Andre. G. Hochuli received the B.S. and M.Sc. degrees in Computer Science from PUCPR Pontifical Catholic University of Paraná (PUCPR), Curitiba, Brazil, in 2004 and 2007, respectively. Currently he is a Ph.D. candidate in the Department of Informatics of the Federal University of Parana, Curitiba, Brazil. His interests include pattern recognition and machine learning.

Luiz S. Oliveira received the B.S. degree in Computer Science from UnicenP, Curitiba, PR, Brazil, the M.Sc. degree in electrical engineering and industrial informatics from the Centro Federal de Educacao Tecnologica do Parana (CEFET-PR), Curitiba, PR, Brazil, and Ph.D. degree in Computer Science from Ecole de Technologie Superieure, Universite du Quebec in 1995, 1998, and 2003, respectively. From 2004 to 2009 he was a professor of the Computer Science Department at Pontifical Catholic University of Parana, Curitiba, PR, Brazil. In 2009 he joined the Federal University of Parana, Curitiba, PR, Brazil, where he is a professor of the Department of Informatics and Head of the Graduate Program in Computer Science. His current interests include Pattern Recognition, Machine Learning, and Image Analysis.

Alceu S. Britto Jr. received M.Sc. degree in Industrial Informatics from the Centro Federal de Educação Tecnológica do Paraná (CEFET-PR, Brazil) in 1996, and Ph.D. degree in Computer Science from the Pontifícia Universidade Católica do Paraná (PUCPR, Brazil) in 2001. In 1989, he joined the Informatics Department of the Universidade Estadual de Ponta Grossa (UEPG, Brazil). In 1995, he also joined the Computer Science Department of the Pontifícia Universidade Católica do Paraná (PUCPR) and, in 2001, the Post-graduate Program in Informatics (PPGIa). His research interests are in the areas of document analysis and handwriting recognition.

Robert Sabourin joined in 1977 the Physics Department of the Montreal University where he was responsible for the design, experimentation and development of scientific instrumentation for the Mont Mégantic Astronomical Observatory. His main contribution was the design and the implementation of a microprocessor-based fine tracking system combined with a low-light level CCD detector. In 1983, he joined the staff of the École de Technologie Supeérieure, Université du Québec, in Montréal where he cofounded the Department of Automated Manufacturing Engineering where he is currently Full Professor and teaches Pattern Recognition, Evolutionary Algorithms, Neural Networks and Fuzzy Systems. In 1992, he joined also the Computer Science Department of the Pontifícia Universidade Católica do Paraná (Curitiba, Brazil) where he was co-responsible for the implementation in 1995 of a master program and in 1998 a Ph.D. program in applied computer science. Since 1996, he is a senior member of the Centre for Pattern Recognition and Machine Intelligence (CENPARMI, Concordia University). Dr. Sabourin is the author (and co-author) of more than 260 scientific publications including journals and conference proceedings. He was a co-chair of the program committee of CIFED'98 (Conférence Internationale Francophone sur l'Écrit et Le Document, Québec, Canada) and IWFHR'04 (9th International Workshop on Frontiers in Handwriting Recognition, Tokyo, Japan). He was nominated as a Conference co- chair of ICDAR'07 (9th International Conference on Document Analysis and Recognition) that has been held in Curitiba, Brazil, in 2007. His research interests are in the areas of handwriting recognition, signature verification, intelligent watermarking systems and bio-cryptography.