International Journal on Document Analysis and Recognition (IJDAR) (2018) 21:219–232
https://doi.org/10.1007/s10032-018-0301-6

SPECIAL ISSUE PAPER

Fixed-sized representation learning from offline handwritten signatures of different sizes

Luiz G. Hafemann[1] · Luiz S. Oliveira[2] · Robert Sabourin[1]

Received: 6 July 2017 / Revised: 29 January 2018 / Accepted: 30 March 2018 / Published online: 23 April 2018

Abstract
Methods for learning feature representations for offline handwritten signature verification have been successfully proposed in recent literature, using deep convolutional neural networks to learn representations from signature pixels. Such methods reported large performance improvements compared to handcrafted feature extractors. However, they also introduced an important constraint: the inputs to the neural networks must have a fixed size, while signatures vary significantly in size between different users. In this paper, we propose addressing this issue by learning a fixed-sized representation from variable-sized signatures by modifying the network architecture, using spatial pyramid pooling. We also investigate the impact of the resolution of the images used for training and the impact of adapting (fine-tuning) the representations to new operating conditions (different acquisition protocols, such as writing instruments and scan resolution). On the GPDS dataset, we achieve results comparable with the state of the art, while removing the constraint of having a maximum size for the signatures to be processed. We also show that using higher resolutions (300 or 600 dpi) can improve performance when skilled forgeries from a subset of users are available for feature learning, but lower resolutions (around 100dpi) can be used if only genuine signatures are used. Lastly, we show that fine-tuning can improve performance when the operating conditions change.

Keywords Handwritten signature verification · Representation learning · Convolutional neural networks · Transfer learning · Domain adaptation

1 Introduction

The handwritten signature is a behavioral biometric trait that is extensively used to verify a person's identity in legal, financial and administrative areas. Automating the verification of handwritten signatures has been a subject of research since the decade of 1970 [1–4], considering two scenarios: online (dynamic) and offline (static). In the online case, signatures are captured using a special device, such as a pen tablet, that records the dynamic information of the writing process (e.g., position of the pen over time). For offline signature verification, we consider signatures written on paper, that are subsequently scanned to be used as input.

✉ Luiz G. Hafemann
luiz.gh@gmail.com; lghafemann@livia.etsmtl.ca

[1] Laboratoire d'imagerie, de vision et d'intelligence artificielle (LIVIA), École de technologie supérieure, 1100, rue Notre-Dame Ouest, Montreal, QC H3C 1K3, Canada

[2] Department of Informatics, Federal University of Parana (UFPR), Rua Cel. Francisco Heraclito dos Santos, 100, Curitiba, PR, Brazil

Most of the research effort in offline signature verification has been devoted to finding good feature representations for signatures, by proposing new feature descriptors for the problem [4]. Recent work, however, showed that learning features from data (signature images) can improve system performance to a large extent [5–8]. These work rely on training deep convolutional neural networks (CNNs) to learn a hierarchy of representations directly from signature pixels.

Although these methods present good performance, they also introduce some issues. Signatures from different users vary significantly in size, while a feature descriptor should provide a fixed-sized representation for classification. This is not a problem in many feature descriptions used for signature verification, that by design are able to accommodate signatures of different sizes. Neural networks, on the other hand, in general require fixed-sized inputs, and thus, these methods require preprocessing the signatures such that they all have the same size. Most commonly, signatures are either (a) resized to a common size or (b) first centered in a blank image of a "maximum signature size," and then resized. Figure 1

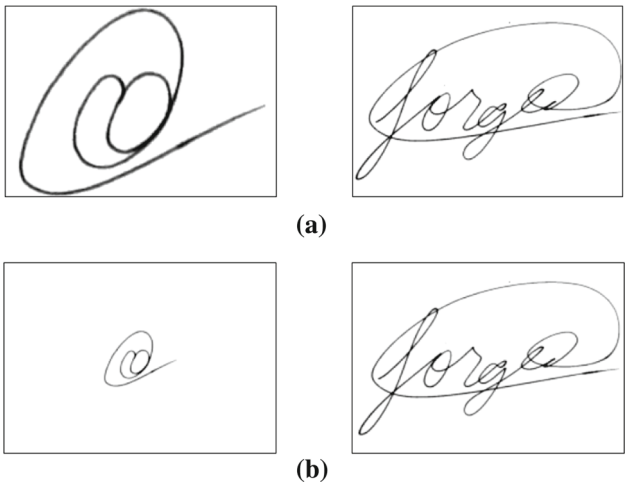

Fig. 1 Two approaches for normalizing the signatures to a common size. The signature on the left is small (176 × 229 pixels), while the signature on the right is large (484 × 819 pixels). **a** Directly resizing the signatures to the input size of the network (170 × 242); **b** centering the signatures in a canvas of a "maximum size" (600 × 850) and then resizing to 170 × 242 pixels

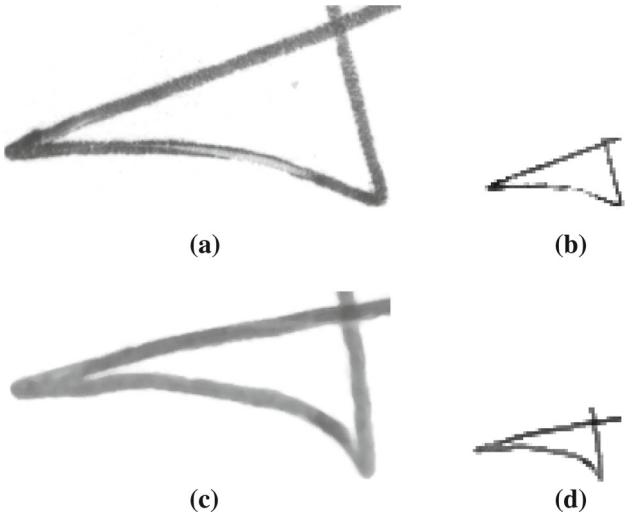

Fig. 2 Details of a genuine signature and a skilled forgery for user 244 in the GPDS dataset. At 300 dpi, we can notice the limp strokes of the skilled forgery, most likely due to slow hand movements while attempting to reproduce the overall shape of the genuine signature. At 100 dpi, information about line quality is mostly lost. **a** Genuine, 300 dpi, **b** genuine, 100 dpi, **c** forgery, 300 dpi, **d** forgery, 100 dpi

illustrates the problems with these approaches. In alternative (a), the width of the strokes becomes very different depending on the size of the original image, while in alternative (b) the width of strokes is not affected, but instead we may lose detail on small signatures, that would otherwise be preserved in the first alternative. Empirically, alternative (b) presented much better results [5], but it also creates the problem that now a "maximum size" is defined, and if a new signature is larger than this size, it would need to be reduced (causing similar problems to (a) regarding the width of the strokes).

Another problem in learning the representations from signature images is the selection of the resolution of the input images. The methods proposed in the literature use small images (e.g., 96 × 192 in [7], 170 × 242 in [9]). For the signatures used in these papers, this is equivalently of using a resolution around 100 dpi. However, as illustrated in Fig. 2, the distinction of genuine signatures and skilled forgeries often rely on the line quality of the strokes (in particular for slowly traced forgeries, as noted in [9]). This suggests that using higher resolutions may improve performance on this task.

In this paper, we propose learning a fixed-sized representation for signatures of variable size, by adapting the architecture of the neural network, using spatial pyramid pooling (SPP) [10,11]. Our contributions are as follows: we define and evaluate different training protocols for networks with SPP applied to offline handwritten signatures. After training, signatures of any size can be fed to the network in order to obtain a fixed-sized representation. We also evaluate the impact of the image resolution on the classification accuracy, and the generalization of features learned in one dataset to other operating conditions (e.g., different acquisition protocols, signatures from people of different locations), by using transfer learning to other datasets.

For feature learning, we used the problem formulation presented in [6], where Writer-Independent features are learned using a subset of users, and subsequently used to train Writer-Dependent classifiers for another set of users. We also use the architecture defined in this work as baseline (SigNet). We adapt this architecture to learn fixed-sized representations (proposing different training protocols) and modifying the architectures to handle images of higher resolution. We conducted experiments on four widely used signature verification datasets: GPDS, MCYT, CEDAR and the Brazilian PUC-PR dataset, and two synthetic datasets (Bengali and Devanagari scripts). Using the proposed architecture, we obtain a similar performance compared to the state of the art, while removing the constraint of having a fixed maximum signature size. We also note that using higher resolutions (300dpi) greatly improves performance when skilled forgeries (from a subset of users) is available for training. On the other hand, if only genuine signatures are used for feature learning, higher resolutions did not improve performance. We verify that the learned features generalize to different operating conditions (by testing them on other datasets) and that fine-tuning the representation for the different conditions further improves performance. We observed that the features learned on GPDS generalize better to other western signature datasets (MCYT, CEDAR and Brazilian PUC-PR) than to other types of scripts (Bengali and Devanagari) and that fine-tuning also largely addresses this problem.

2 Related work

The problem of offline signature verification is either formulated as Writer-Dependent, with one classification task defined for each user enrolled to the system, or as a Writer-Independent problem, where we consider a single problem, of comparing a questioned signature to a reference signature. In the literature, Writer-Dependent classification is most commonly used: for each user, a set of reference (genuine) signatures are used as positive samples, and a set of genuine signatures from other users (in this context called "Random forgeries") are used as negative samples, and a binary classifier is trained. Alternatively, some authors propose using one-class classifiers for the Writer-Dependent formulation, using only genuine signatures from the user as positive samples (e.g., [12]). Writer-Independent classification, on the other hand, is often used by training a binary classifier on a dissimilarity space, where the inputs are the absolute difference of two feature vectors: $\mathbf{x} = |\mathbf{f}_1 - \mathbf{f}_2|$, where $\mathbf{f}_1$ and $\mathbf{f}_2$ are feature vectors extracted from two signatures, and we consider a binary label: $y = 1$ if both signatures are from the same user, and $y = -1$ otherwise [13–15].

After training the classifiers, we verify the performance of the system in distinguishing genuine signatures from forgeries. We adopt the following definitions of forgery, which are the most common in the Pattern Recognition community: "Random Forgeries" are forgeries made without any knowledge of the user's genuine signature, where the forger uses his own signature instead. In the case of "Simple forgeries", the forger has access to the person's name. In this case, the forgery may present more similarities to the genuine signature, in particular for users that sign with their full name, or part of it. Lastly, for "Skilled Forgeries," the forger has access to the user's signature, and often practices imitating it. This results in forgeries that have higher resemblance to the genuine signature and therefore are harder to detect. While discriminating random and simple forgeries are relatively simpler tasks (as reflected in lower error rates in the literature), discriminating genuine signatures and skilled forgeries remains a challenging task.

A critical aspect of designing signature verification systems is how to extract discriminant features from the signatures. A large part of the research efforts on this field addresses this question, by proposing new feature descriptors for the problem. These features range from simple descriptors such as the size of the signature and inclination [16], graphometric features [17,18], texture-based [19,20], interest point-based [21], among others. Recent advancements in this field include using multiple classifiers trained with different representations [20], using interval symbolic representations [22] and augmenting datasets by duplicating existing signatures or creating synthetic ones [23–25]. More recently, methods for learning features from signature images have been proposed [5–8]. Although these methods demonstrated improved performance, they introduced some issues, notably by requiring that all signature images have the same size, which is the problem addressed in this paper. We note that this problem is not present in many handcrafted feature descriptions used for signature verification: for instance Local Binary Patterns (LBP) [26] and Histogram of Oriented Gradients (HOG) [27] use histograms over the entire image, therefore resulting in feature vectors of the same size regardless of the input size; Extended Shadow Code (ESC) [28] divides the image in the same number of windows (adapting the size the windows), therefore also working with signatures of variable sizes.

The problem of requiring inputs of a fixed size for neural networks also affects other applications, such as object recognition. This problem is often handled by simply resizing and cropping images. While these are common operations for object recognition, we argue that they are less interesting for signature verification. In object recognition, the classification task considers objects at different scales. Therefore, resizing an image to fit a particular size is a reasonable action to take, since it is aligned with the invariance to scale that we expect from the classifiers (as long as the change in scale does not distort the image, such as scaling height and width by different factors). On the other hand, for signature verification we have control of how the signatures are acquired: all signatures are usually scanned at the same resolution, usually 300 or 600 dpi. Therefore, changes in scale, introduced by resizing the image, alter the signal is ways that would not otherwise be present. In this case, a better solution would not require resizing the signature images by different factors depending on their original size.

In the context of object recognition, He at al. proposed a solution for working with inputs of variable size, by using spatial pyramid pooling [11]. However, the training procedure still requires fixed-sized images. In [11], the authors proposed using two image sizes, resizing each image to the these sizes (i.e., duplicating the dataset in two different scales). Learning is then conducted by alternating between the two sets in each training epoch. This is sub-optimal for signature images, since we would like to avoid resizing the images entirely. In this work, we propose and test other training protocols for training networks with SPP on signature data.

3 Proposed method

In this work, we consider the two-stage approach described in [5,6], where we train Writer-Dependent classifiers on a set of users, using a feature representation learned on another set of users. We note, however, that the methods described in

this paper can be used for other feature learning strategies, such as the ones used in [7,8].

We consider two disjoint sets of users: a development set $\mathcal{D}$, where we learn feature representations, and an exploitation set $\mathcal{E}$ that consists of the users "enrolled to the system," for whom we train Writer-Dependent classifiers. The first phase consists in learning a function $\phi(X)$, using the data from $\mathcal{D}$, that takes a signature X as input, and returns a fixed-sized feature vector. In the second phase, we use this learned function to "extract features" for the signatures in $\mathcal{E}$, and train a binary classifier for each user. While we could use all users for learning the representations, this separation in two sets allows us to estimate the generalization performance of using this learned representation for new users. This is important since the set of users in a system is not fixed—new users may enroll at any time, and in this formulation we simply use the learned function $\phi(X)$ to obtain a representation for the signatures of this new user and train a binary classifier.

In order to handle signatures of different sizes, we change both the feature learning process and the process to obtain representations for new signatures using the learned network.

3.1 Network architecture and objective function

The definition of a convolutional neural network architecture usually specifies the input size of the images for training and testing. However, as noted in [11], this constraint is not caused by the usage of convolution and pooling layers, but rather by the usage of fully connected layers at the end of network architectures. The reason is that the convolution and pooling operations are well defined for inputs of variable sizes, simply resulting in an output of a larger size. This presents a problem between the last pooling layer and the first fully connected layer of the network (layer FC1 in Fig. 3): the last pooling layer is "flattened" to a vector of dimensionality K (e.g., a pooling output of size $32 \times 3 \times 2$ becomes a vector of $K = 192$ elements), and the fully connected layer uses a weight matrix of size $K \times M$, where M is the output size of the fully connected layer. If we use the network to process an input $\hat{X}$ of a different size, the output of the last pooling layer will have a different size. Flattening the representation results in a vector of dimensionality $\hat{K} \neq K$, and therefore, the vector–matrix product in the fully connected layer will not be defined.

The central idea of spatial pyramid pooling (SPP) [11] is to obtain a fixed-size representation for variable-sized input images. This is done by adapting the size of the pooling region (and strides) for each image size, such that the output of the last pooling operation has a fixed size and therefore can be used as input to fully connected layers. In SPP, a set of fixed-sized outputs is chosen, and the result of them is concatenated. This is illustrated in the "SPP Layer" box in Fig. 3: we consider pooling with output sizes 1×1, 2×2 and 4×4, which has a total of $1 + 4 + 16 = 21$ outputs for each channel. Each image would therefore output a fixed representation of size $21 \times C$, where C is the number of feature maps/channels in the last convolutional layer.

Figure 3 illustrates a CNN architecture used in this work. The network contains a series of convolutions and max-pooling operations, with a spatial pyramid pooling layer between the last convolutional layer and the first fully connected layer. This layer outputs a fixed-sized output regardless of the size of the input signature.

We consider two application scenarios, as in [6]: one in which we have only genuine signatures available for training, and one in which we also have access to skilled forgeries for a subset of users. In the first scenario, we consider a training objective of distinguishing between different users in the development set: the network outputs $P(\mathbf{y}|X)$: the probability of the signature belonging to one of the users in $\mathcal{D}$. Therefore, the network learns to identify the users that produced the signatures in $\mathcal{D}$. In the second scenario, we would like to leverage the information of forgeries in the feature learning process, and we use the multi-task approach defined in [6]. In this formulation, the network also predicts whether or not the signature is a forgery: $P(f|X)$. We simultaneously train the network to optimize both objectives (distinguish between different users, and between genuine signatures and skilled forgeries), by using the loss function defined in Eq. 1:

$$\begin{aligned} L &= (1 - f_i)(1 - \lambda)L_c + \lambda L_f \\ &= -(1 - f_i)(1 - \lambda) \sum_j y_{ij} \log P(y_j|X_i) \\ &\quad + \lambda\big(- f_i \log(P(f|X_i)) - (1 - f_i) \log(1 - P(f|X_i))\big) \end{aligned} \quad (1)$$

where λ is a hyperparameter that trades-off between the two objectives, X_i is the signature, $\mathbf{y}_i$ is the actual user of signature ($y_{ij} = 1$ if the signature i belongs to user j), and f_i indicates whether or not the signature i is a forgery. L_c and L_f indicate the loss functions for user classification and forgery classification, respectively, which are expanded in the second and third lines. We refer the reader to [6] for more details on this formulation.

Table 1 lists the CNN architectures used in this paper. We consider a total of six architectures, considering three different resolutions (around 100 dpi, 300 dpi and 600 dpi), and with or without spatial pyramid pooling. Each line in the table represents a layer of the CNN. For convolutional layers, we specify the size and number of feature maps (filters), the stride and the padding. For instance, conv11-32-s4-p5 refers to a convolutional layer with 32 filters of size 11×11, with stride $s = 4$ and padding $p = 5$. For pooling operations, we inform the pool size, the stride and padding. When not specified, we use stride $s = 1$. After each learnable layer (with the exception of the output layers) use a batch normal-

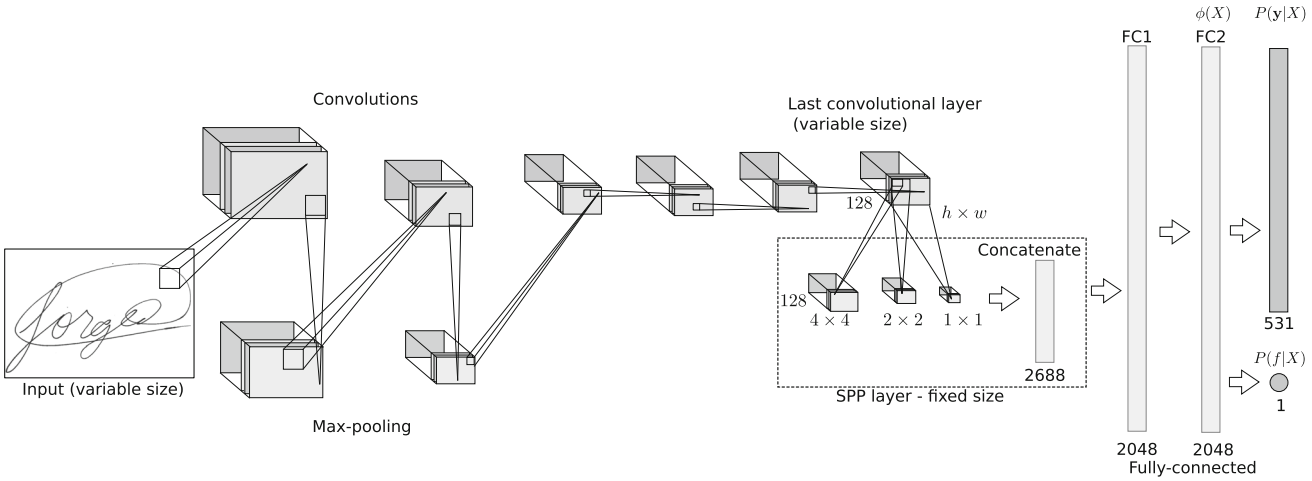

Fig. 3 One of the CNN architectures used in this work. The input signature (of variable size) is transformed in a sequence of convolution and max-pooling operations. The last convolutional layer results in 128 maps of size $h \times w$ (the actual size varies according to the signature size). The spatial pyramid pooling layer (SPP) is then used to obtain a fixed-sized representation, by adapting the size of pooling regions, to obtain pooled results in three sizes: 4×4, 2×2 and 1×1. These are concatenated in a single vector of size $21 \times 128 = 2688$ units, which is then used as input to the fully connected layers. During training, the network outputs $P(\mathbf{y}|X)$ (and, if forgeries are used during training, also $P(f|X)$), and the network is trained to minimize the cross-entropy with respect to the training dataset. For obtaining representations for new signatures, we perform forward propagation until the last layer before softmax, obtaining $\phi(X)$, a vector of 2048 dimensions, regardless of the signature size

Table 1 CNN architectures used in this paper

SigNet	SigNet-SPP	SigNet-300dpi	SigNet-SPP-300dpi	SigNet-600dpi	SigNet-SPP-600dpi
conv11-96-s4-p0	conv11-96-s4-p0	conv11-32-s3-p5	conv11-32-s3-p5	conv11-32-s4-p5	conv11-32-s4-p5
pool3-s2-p0	pool3-s2-p0	pool3-s2-p0	pool3-s2-p0	pool3-s3-p0	pool3-s3-p0
conv5-256-p2	conv5-256-p2	conv5-64-p2	conv5-64-p2	conv5-64-p2	conv5-64-p2
pool3-s2-p0	pool3-s2-p0	pool3-s3-p0	pool3-s3-p0	pool3-s2-p0	pool3-s2-p0
conv3-384-p1	conv3-384-p1	conv3-128-p1	conv3-128-p1	conv3-128-p1	conv3-128-p1
conv3-384-p1	conv3-384-p1	conv3-128-p1	conv3-128-p1	conv3-128-p1	conv3-128-p1
		pool3-s2-p0	pool3-s2-p0	pool2-s2-p0	pool2-s2-p0
conv3-256-p1	**conv3-180-p1**	conv3-128-p1	conv3-128-p1	conv3-128-p1	conv3-128-p1
pool3-s2-p0	**spp-4-2-1**	pool3-s3-p0	**spp-4-2-1**	pool4-s4-p0	**spp-4-2-1**
FC1-2048	FC1-2048	FC1-2048	FC1-2048	FC1-2048	FC1-2048
FC2-2048	FC2-2048	FC2-2048	FC2-2048	FC2-2048	FC2-2048
FC-M + softmax; FC-1 + sigmoid	FC-M + softmax; FC-1 + sigmoid	FC-M + softmax; FC-1 + sigmoid	FC-M + softmax; FC-1 + sigmoid	FC-M + softmax; FC-1 + sigmoid	FC-M + softmax; FC-1 + sigmoid

ization layer [29]. The SigNet architecture was defined in previous work [6], while the other architectures are adapted versions to handle larger images. For higher resolutions, we notice that images are very large (e.g., 780×1095 pixels for a 600 dpi signature). To handle these larger images, we used a smaller number of feature maps, and a more rapid reduction in size across the layers, by using a more aggressive pooling. For each of the three resolutions, we consider both a version with SPP (that accepts inputs of any size), and without SPP (that accepts inputs of a fixed size). The two versions have the same overall structure, but diverge on the last pooling layer. The network SigNet-SPP has another difference (lower number of convolutional maps in the last convolutional layer) to keep the number of parameters between the SPP and non-SPP version similar. In the table, the differences between the non-SPP and SPP versions are highlighted in bold. In all architectures, the last layer outputs M neurons, which estimate $P(\mathbf{y}|X)$, the probability of a signature X belonging to a particular user in $\mathcal{D}$. For the experiments using forgeries during feature learning, the network also outputs a single neuron that predicts $P(f|X)$, the probability that the signature is a forgery. We report experiments with both scenarios

(with and without forgeries for feature learning). In the cases where forgeries are used, we append a suffix **-F** to the architecture name. For example, *SigNet-300dpi-F* refers to using the architecture SigNet-300dpi using both genuine signatures and skilled forgeries for training, while *SigNet-300dpi* refers to using the same architecture, but trained with only genuine signatures.

The spatial pyramid pooling layer was implemented as in [11]: we use pooling regions of sizes 4×4, 2×2 and 1×1, resulting in a total of 21 outputs for each feature map. The pooling region and strides are dynamically determined for each input size. Let h and w be the output size of the last convolutional layer. For the pyramid level of size $n \times n$, the pooling region of the unit (j, i) is defined as rows between: $\lfloor \frac{j-1}{n} h \rfloor$ and $\lceil \frac{j}{n} h \rceil$ and columns between $\lfloor \frac{i-1}{n} w \rfloor$ and $\lceil \frac{i}{n} w \rceil$, where $\lfloor . \rfloor$ and $\lceil . \rceil$ denote the floor and ceiling operations. Similarly to max-pooling, we take the max of this pooling region, that is, the output of unit (j, i) is the maximum value of the pooling region defined above. The implementation of this layer has been made publicly available in the Lasagne library.[1]

3.2 Training protocol

The neural networks are initialized with random weights following [30] and training is performed with stochastic gradient descent to minimize the loss function defined in Sect. 3.1. We use mini-batches of data (which is required in order to use batch normalization and also speeds up training), and we consider different protocols for generating the mini-batches, as described below.

The networks without SPP require a fixed input size for all images. In this case, we preprocess all signatures by centering them in a canvas of a "maximum size," and then resizing to the desired input size for the network.

When using spatial pyramid pooling, we can process signatures of any size, but during training we need to design a protocol that provides batches of signatures having the exact same size. We consider two alternatives:

1. *Fixed size* Using a single "maximum signature size" (as in the training for networks without SPP);
2. *Multiple sizes* Defining multiple canvas sizes, and centering each signature on the smallest canvas that fit the signature.

In the first case, all the images on the training set have the same size, and we simply process the images in mini-batches in a random order.

For the second alternative, we define different image sizes based on statistics of the development set (the set of signatures used to train the CNN). Consider the following definitions:

- $\mu_h, \sigma_h, \max_h$: height of the signatures in the development set (mean, standard deviation and maximum, respectively);
- μ_w, σ_w, $\max_w$: width of the signatures in the development set (mean, standard deviation and maximum, respectively).

We divide the dataset into 5 different parts, as follows:

1. Images larger than 3 standard deviations are considered "outliers." In particular, images taller than $\tau_h = \mu_h + 3\sigma_h$ or wider than $\tau_w = \mu_w + 3\sigma_w$ are all assigned to the largest canvas, of size $(\max_h \times \max_w)$;
2. The remaining signatures are split into four groups, by using the medians of the height and width. Given the medians $\tilde{H}$ and $\tilde{W}$ for the height and width, respectively, we consider the following canvas sizes: $(\tilde{H} \times \tilde{W})$, $(\tilde{H} \times \tau_w)$, $(\tau_h \times \tilde{W})$, $(\tau_h \times \tau_w)$.

Each signature is centered (not resized) in the smallest canvas size that fits the signature. Therefore, this creates a total of 5 datasets, one for each canvas size.

During training, we create an iterator for each of the 5 datasets: each iterator returns batches of signatures of the same size (within the batch). We then train the model by taking batches of different image sizes, alternating the sizes after each mini-batch (contrary to [11] that alternated after an entire epoch). This procedure is detailed in Algorithm 1. In this algorithm, the *train* function is a single step of Stochastic Gradient Descent, with Nesterov Momentum.

```
function TrainWithMultipleSizes(S, iterators):
/* train the network for one epoch                    */
    Data: S: set of image sizes; iterators: list of data iterators,
          for each image size
    active ← S ;
    while active ≠ ∅ do
        for s ∈ active do
            if iterator[s].has_next_batch() then
                mini_batch ← iterator[s].next_batch() ;
                train(mini_batch) ;
            end
            else
                active ← active \ s
            end
        end
    end
```

Algorithm 1: Training algorithm for "multiple sizes," for one epoch.

We trained the networks for 60 epochs, using mini-batches of size 32, L2 penalty with weight decay set to 10^{-4} and

[1] https://github.com/Lasagne/Lasagne/.

momentum factor of 0.9. Training started with a learning rate of 10^{-3}, which was decreased twice (at epochs 20 and 40) by dividing it by 10 each time.

3.2.1 Data augmentation

In previous work [9], we performed data augmentation by performing random crops of the input images. We adopt the same protocol for the "fixed-size" training. However, in the "Multiple sizes" protocol defined above, where we use smaller canvas sizes, cropping the images could result in cropping part of the signature, not only the background. Instead, we use the opposite strategy, of enlarging the signature images, by padding the signatures with the background color. We use this strategy to avoid losing part of the signal due to cropping. For example, consider an image of size 300×300, and a padding of size 20×20: we pad the image so that it has size 320×320, positioning the original image to randomly start between 0 and 20 pixels in height and width (i.e., not necessarily in the center of this new image). For even greater variability, we consider a maximum value of padding, and in each mini-batch we randomly select the padding between 0 and this maximum value.

3.3 Fine-tuning representations

When considering the generalization of the learned features to new operating conditions (e.g., new acquisition protocol), it is possible to fine-tune the representations to the new conditions. In order to evaluate the impact of fine-tuning the representations, we consider a network trained in one dataset as a starting point and subsequently train it for users of another dataset.

Similarly to previous work on transferring representations [31,32], we perform the following steps for fine-tuning representation to a new dataset:

- Duplicate the network that was trained in the first dataset;
- Remove the last layer (that correspond to $P(\mathbf{y}|X)$ for the users in the first dataset);
- Add a new softmax layer, with M_2 units, corresponding to $P(\mathbf{y}|X)$, the probability of a signature image belonging to one of the M_2 users of the second dataset;
- Train the network on the second dataset with a reduced learning rate (5×10^{-4}).

The training procedure during fine-tuning is similar to the training algorithm used for learning the features in the first dataset. The exception is for the SPP models trained with a "fixed size." In this case, we consider two distinct sizes: the original maximum signature size from the first dataset, and the maximum signature size from the target dataset.

Since different datasets have different acquisition protocols (e.g. type of writing instrument, instructions for the forgers, and the resolution of scanned images), we expect that fine-tuning the representations to a set of users from the same domain should improve performance.

3.4 Training WD classifiers

After we learn the CNN in one set of users, we use it to obtain representations for signatures of users in the exploitation set $\mathcal{E}$ and train Writer-Dependent classifiers. The procedure to obtain the representations vary slightly depending on the training method:

- Networks without SPP: The signatures from $\mathcal{E}$ are centered in a canvas of maximum size ($H_{\max} \times W_{\max}$). During transfer learning, we consider the maximum size of the target dataset, and resize all images to the size of the original dataset;
- SPP trained with "fixed size": The signatures from $\mathcal{E}$ are centered in a canvas of size ($H_{\max} \times W_{\max}$). During transfer learning, signatures that are larger than this canvas are processed in their original size;
- SPP trained with "multiple sizes": The signatures are processed in their original size.

The differences among the three training alternatives are summarized in Table 2.

For each user in the set $\mathcal{E}$, we build a dataset consisted of r genuine signatures from the user as positive samples, and genuine signatures from other users as negative samples. We then train a binary classifier for the user: a linear SVM or an SVM with the RBF kernel. We usually have many more negative than positive samples for training, since we only have a few genuine signatures for the user, while we can use samples from many users as negative samples. For this reason, we correct this skew by giving more weight to the positive samples, as described in [6]. After the classifiers are trained, we measure their capability of classifying genuine signatures and different types of forgery.

4 Experimental protocol

We conducted experiments on four offline handwritten signature datasets: GPDS-960 [33], MCYT-75 [34], CEDAR [35] and Brazilian PUC-PR [36], and two synthetic datasets, for Bengali and Devanagari scripts [25]. We used a subset of the GPDS-960 dataset for learning feature representations, using the different architectures and training methods described in this article. We then evaluate the performance of Writer-Dependent classifiers trained with these feature rep-

Table 2 Summary of differences between the training/testing protocols

	Without SPP	SPP (training with fixed size)	SPP (training with multiple sizes)
Training images	Centered in a fixed size	Centered in a fixed size	Consider 5 different sizes
CNN architecture	Use pooling with fixed pooling size	Use SPP (variable pooling size, fixed output)	Use SPP (variable pooling size, fixed output)
Generalization (extracting features)	Center images in a fixed size. Larger images are resized	Center images in a fixed size. Larger images are processed in their original size	All images are processed in their original size
Fine-tuning	Center images in the maximum size of the target dataset. All images are resized to the maximum size of the source dataset	Center images in two canvases: maximum size of the target dataset, and maximum size of the source dataset	Consider 5 different sizes (defined in the target dataset)

resentations, on a disjoint subset of GPDS, as well as the other datasets.

In order to allow comparison with previous work, we used the development set $\mathcal{D}$ as the last 531 users of GPDS (users 350-881) for training the CNNs. For the training protocol using "multiple sizes," we followed the procedure detailed in Sect. 3.2 to process the development dataset into 5 different canvas sizes. For instance, at 600 dpi we used canvases of size 338×684, 338×1183, 619×684, 619×1183 and 778×1212. The networks were then trained as defined in Sect. 3.2.

The images were preprocessed to remove noise, by applying OTSU's algorithm to find the threshold between background and foreground. The background pixels were set to white, leaving the signature pixels in grayscale. The images were then inverted by subtracting them from the maximum pixel intensity: $I_P(x, y) = 255 - I(x, y)$. In the resulting images, the background is therefore zero-valued. The OTSU algorithm was not applied to the two synthetic datasets, since they do not contain any noise.

In the literature, slightly different protocols are used for each dataset, in particular regarding how many reference signatures are used for training, and which signatures are used as negative samples. We use the following protocols: In the GPDS dataset, we trained Writer-Dependent classifiers for the first 300 users (to compare to results using the GPDS-300 dataset), using $r = 12$ reference signatures as positive samples. We used 12 signatures from each user in the development $\mathcal{D}$ as negative samples ($12 \times 531 = 6372$ signatures). This protocol is similar to the Brazilian dataset, where we have a separate development set $\mathcal{D}$. We train classifiers for the first 60 users using $r = 15$, and 15 signatures from each of the remaining 108 users as negative samples ($15 \times 108 = 1620$ signatures). In the MCYT and CEDAR datasets, we used $r = 10$ and $r = 12$, respectively, and the same number of signatures from each other user in the exploitation set $\mathcal{E}$ as negative samples. In all cases, we trained a binary SVM, with an RBF kernel. We used the same hyperparameters as previous research [9]: $C = 1$ and $\gamma = 2^{-11}$, that were selected using a subset of the GPDS validation set. In this paper, we did not explore optimizing these hyperparameters for each dataset (or even each user), but rather keep the same set of parameters for comparison with previous work.

For the experiments generalizing to different conditions (datasets), we considered two scenarios: using the CNN trained on GPDS to extract features without any changes and fine-tuning the representation on these datasets. In these experiments, we used the network trained on GPDS images of the same resolution of the datasets (300 dpi for CEDAR and Brazilian, 600 dpi for other datasets).

In order to assess the generalization performance of the fine-tuned representations, we conducted cross-validation experiments as follows:

1. The dataset is randomly split in two sets of users (50%/50%). Following the same terminology as before, we can consider them to be a development set $\mathcal{D}$ and exploitation set $\mathcal{E}$;
2. We fine-tune the CNN (originally trained on GPDS) for the development set $\mathcal{D}$;
3. We use the fine-tuned CNN to extract features for the exploitation set $\mathcal{E}$ and train WD classifiers.

This protocol allows for an unbiased estimation of the performance on new users, whose signatures match the same operating characteristics of the dataset. We performed cross-validation running the steps above 10 times, each time randomly splitting the dataset in half, fine-tuning the CNN and training WD classifiers. For each fine-tuned CNN, we performed 10 runs on the WD classifier training with different signatures used for training/testing. Therefore, we fine-tuned a total of 10 CNNs for each dataset and trained a total of 100 WD classifiers for each user in each dataset, and for each architecture. We then report the mean and standard deviation across these 100 runs.

We evaluate the generalization of the learned representations to different scripts by training WD classifiers on synthetic signatures for two Indian scripts: Bengali and Devanagari [25]. For these datasets, since skilled forgeries are not available (the generation procedure is only defined for genuine signatures in [25]) we evaluate the performance of the system on random forgeries. To allow for comparison with previous work, we train the WD classifiers with $r = 5$ genuine signatures as positive samples. We also evaluated the errors on random forgeries on the other four datasets, which allows us to verify the generalization performance to other western scripts (on MCYT, CEDAR and Brazilian PUC-PR) and for other types of scripts (Bengali and Devanagari).

We evaluate the performance primarily using the equal error rate (EER): which is the error when false acceptance (misclassifying a forgery as being genuine) is equal to false rejection (misclassifying a genuine as being a forgery). We considered two forms of calculating the EER: $\text{EER}_{\text{user thresholds}}$: using user-specific decision thresholds, and $\text{EER}_{\text{global threshold}}$: using a global decision threshold. For most experiments, we report the equal error rates using only skilled forgeries. In the experiment where we compare the generalization to different scripts, we report the equal error rates calculated with random forgeries.

For the Brazilian PUC-PR dataset, we used the same metrics as previous research in this dataset and also report the individual errors (false rejection rate and false acceptance rate for different types of forgery) and the average error rate, calculate as $\text{AER} = (\text{FRR} + \text{FAR}_{\text{random}} + \text{FAR}_{\text{simple}} + \text{FAR}_{\text{skilled}})/4$. We also reported the average error rate considering only genuine signatures and skilled forgeries: $\text{AER}_{\text{genuine+skilled}} = (\text{FRR} + \text{FAR}_{\text{skilled}})/2$.

For the comparison between different training types, and to measure the impact of fine-tuning, we use t tests to compare the classifiers (using the $\text{EER}_{\text{user thresholds}}$ metric). We considered results significantly different for $p < 0.01$.

5 Results

We first present our analysis on using different image resolutions, followed by the analysis of the methods trained with SPP for handling signatures of variable size, and a comparison with the state of the art.

The results with varying the image resolution are summarized in Fig. 4. This figure shows the classification performance (EER) of Writer-Dependent classifiers trained on the GPDS-300 dataset, as we increase the resolution of the images. For these experiments, we consider the models trained without SPP and consider two training scenarios: when we only use genuine signatures, and when skilled forgeries from a subset of users are used for feature learning (note that for training the WD classifiers, no skilled forgeries are used). The objective of this experiment is to verify the hypothesis that higher-image resolutions are required to discriminate skilled forgeries. We notice an interesting trend in this figure: when using both genuine signatures and skilled forgeries, increasing the resolution greatly improves performance, reducing errors from 2.10% using 100 dpi to 0.4% using 300 dpi. On the other hand, increasing resolution did not improve performance when only genuine signatures are used for feature learning. We argued in the introduction (in Fig. 2) that low resolutions lose information about the line quality. These results suggest that, although fine details are present in higher-resolution images, they are not taken into account when only genuine signatures are used for training the CNN. In other words, since the network does not have access to any skilled forgery, it does not learn features that discriminate line quality. Therefore, when only genuine signatures are available for training, low resolutions (100 dpi) are sufficient, but if forgeries from a subset of users are available, higher resolutions (e.g., 300 dpi) greatly improve performance.

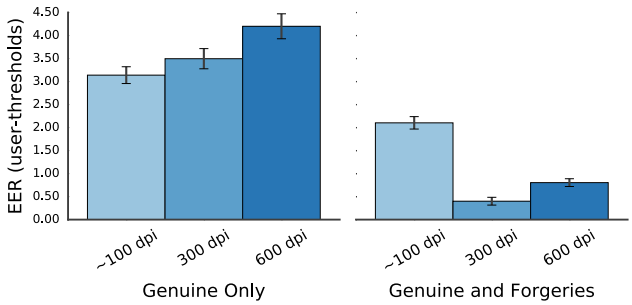

Fig. 4 Impact of the image resolution on system performance: EER of Writer-Dependent classifiers trained on GPDS-300, with representations learned in $\mathcal{D}$ at different resolutions. Left: using only genuine signatures for feature learning; right: using genuine signatures and skilled forgeries for feature learning. Error bars indicate one standard deviation of the mean error (across 10 replications)

We now consider the experiments using SPP for learning a fixed-sized representation for signatures of different sizes. Table 3 compares the performance of the WD classifiers on the GPDS dataset, as we change the training method. We consider both the baseline (network without SPP), and the two proposed training protocols for using SPP: with a single fixed canvas for training (denoted "Fixed" in the table), and using the 5 different canvases, defined in the development set (denoted "Multi" in the table). We compare these different training protocols on the same resolution and training data, and the results that are significantly better than the baseline (at $p < 0.01$) are marked with a bullet point (•). We notice that the performance between the baseline and SPP Fixed is very similar, while the method using multiple canvases during training performs a little worse. The proposed method using SPP Fixed keeps about the same level of performance

Table 3 Performance of WD classifiers on GPDS-300, using 12 reference signatures (Errors and standard deviations in %)

Feature	Training Algorithm	$EER_{global\ threshold}$	$EER_{user\ thresholds}$
SigNet-300dpi		5.72 (±0.21)	3.5 (±0.22)
SigNet-SPP-300dpi	Fixed	5.63 (±0.22)	3.15 (±0.14) •
SigNet-SPP-300dpi	Multi	7.75 (±0.28)	4.86 (±0.24)
SigNet-300dpi-F		1.78 (±0.12)	0.4 (±0.08)
SigNet-SPP-300dpi-F	Fixed	1.69 (±0.1)	0.41 (±0.05)
SigNet-SPP-300dpi-F	Multi	2.52 (±0.09)	0.8 (±0.07)
SigNet-600dpi		7.11 (±0.17)	4.2 (±0.27)
SigNet-SPP-600dpi	Fixed	7.06 (±0.13)	4.02 (±0.18)
SigNet-SPP-600dpi	Multi	6.36 (±0.16)	3.96 (±0.23)
SigNet-600dpi-F		2.46 (±0.09)	0.8 (±0.08)
SigNet-SPP-600dpi-F	Fixed	2.27 (±0.18)	0.65 (±0.11) •
SigNet-SPP-600dpi-F	Multi	2.85 (±0.16)	0.86 (±0.1)

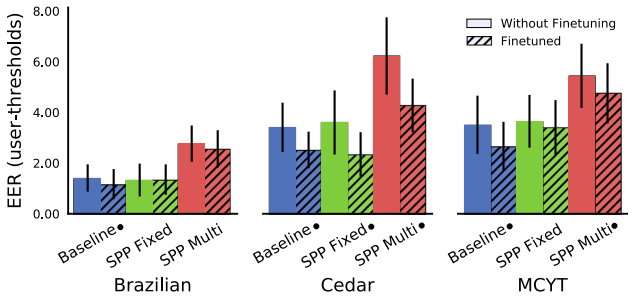

Fig. 5 Classification performance of Writer-Dependent classifiers trained with representations learned in the GPDS dataset. The hatched bars denote results with features fine-tuned in each particular dataset. Error bars denote the standard deviation across 100 replications

as the baseline, while removing the constraint of having a maximum signature size (since both SPP methods accept larger signatures for processing).

The results on transferring representations to different operating conditions are summarized in Fig. 5. We considered models trained on the GPDS dataset, and used these models to extract features and train WD classifiers on other operating conditions, that is, three other datasets: Brazilian PUC-PR, CEDAR and MCYT. In all cases, we verify the impact of fine-tuning the representations for the new operating conditions, following the procedure detailed in Sect. 3.3. For the first two datasets, that were scanned in 300 dpi, we used the representations learned in GPDS at 300 dpi, while for MCYT we used the representations learned at 600 dpi. In this experiment, we did not use any forgeries for training (neither from GPDS nor the target dataset). We performed *t* tests to verify if fine-tuning the representation significantly improved the classification performance (marked with a bullet point next to the dataset name). We can see that the baseline (without SPP) and the SPP model trained with fixed image sizes performed similarly, while SPP trained on multiple canvas sizes performed worse for transfer. We also consistently see that fine-tuning representations on the target datasets helps the domain adaptation, reducing the errors on average.

Table 4 Generalization performance on other datasets, with and without fine-tuning (for *random forgeries*)

Dataset	Fine-tuned	$EER_{global\ threshold}$	$EER_{user\ thresholds}$
Bengali		5.07 (±0.8)	3.41 (±0.81)
Bengali	Yes	0.77 (±0.27)	0.16 (±0.14) •
Devanagari		4.65 (±0.92)	2.93 (±0.8)
Devanagari	Yes	0.33 (±0.2)	0.06 (±0.09) •
MCYT		0.19 (±0.39)	0.03 (±0.13)
MCYT	Yes	0.04 (±0.12)	0.0 (±0.0)
CEDAR		1.14 (±0.75)	0.37 (±0.42)
CEDAR	Yes	0.23 (±0.26)	0.08 (±0.19) •
Brazilian		0.47 (±0.3)	0.2 (±0.25)
Brazilian	Yes	0.5 (±0.38)	0.16 (±0.24)
Bengali (results from [25])	–	0.67	–
Devanagari (results from [25])	–	0.47	–

Table 5 Comparison with state of the art on the GPDS dataset (errors in %)

References	Dataset	#samples per user	Features	EER
Hu and Chen [37]	GPDS-150	10	LBP, GLCM, HOG	7.66
Guerbai et al. [12]	GPDS-160	12	Curvelet transform	15.07
Serdouk et al. [38]	GPDS-100	16	GLBP, LRF	12.52
Yilmaz [20]	GPDS-160	5	LBP, HOG, SIFT	7.98
Yilmaz [20]	GPDS-160	12	LBP, HOG, SIFT	6.97
Soleimani et al. [39]	GPDS-300	10	LBP	20.94
Hafemann et al. [6]	GPDS-300	12	SigNet-F	1.69 ($\pm$ 0.18)
Present work	GPDS-300	12	SigNet-SPP-300dpi	3.15 ($\pm$ 0.14)
Present work	GPDS-300	12	SigNet-SPP-300dpi-F	0.41 ($\pm$ 0.05)

Table 6 Comparison with the state of the art in MCYT (errors in %)

References	#samples per user	Features	EER
Gilperez et al. [40]	5	Contours (chi squared distance)	10.18
Gilperez et al. [40]	10	Contours (chi squared distance)	6.44
Wen et al. [41]	5	RPF (HMM)	15.02
Vargas et al. [19]	5	LBP (SVM)	11.9
Vargas et al. [19]	10	LBP (SVM)	7.08
Ooi et al [42]	5	DRT + PCA (PNN)	13.86
Ooi et al [42]	10	DRT + PCA (PNN)	9.87
Soleimani et al. [39]	5	HOG (DMML)	13.44
Soleimani et al. [39]	10	HOG (DMML)	9.86
Hafemann et al [6]	10	SigNet (SVM)	2.87 ($\pm$ 0.42)
Present work	10	SigNet-SPP-600dpi	3.64 ($\pm$ 1.04)
Present work	10	SigNet-SPP-600dpi (fine-tuned)	3.40 ($\pm$ 1.08)

Table 7 Comparison with the state of the art in CEDAR (errors in %)

References	#samples per user	Features	AER/EER
Chen and Srihari [43]	16	Graph Matching	7.9
Kumar et al. [44]	1	morphology (SVM)	11.81
Kumar et al. [45]	1	Surroundness (NN)	8.33
Bharathi and Shekar [46]	12	Chain code (SVM)	7.84
Guerbai et al. [12]	4	Curvelet transform (OC-SVM)	8.7
Guerbai et al. [12]	8	Curvelet transform (OC-SVM)	7.83
Guerbai et al. [12]	12	Curvelet transform (OC-SVM)	5.6
Hafemann et al. [6]	12	SigNet-F (SVM)	4.63 ($\pm$ 0.42)
Present work	10	SigNet-SPP-300dpi	3.60 ($\pm$ 1.26)
Present work	10	SigNet-SPP-300dpi (fine-tuned)	2.33 ($\pm$ 0.88)

Table 4 shows the results of the experiments on transferring the representation to other types of scripts. The objective of this experiment was to verify if the features learned on the GPDS dataset generalizes to other types of script (in particular, we tested for Bengali and Devanagari). Differently from the previous analysis, for these datasets we consider the performance on discriminating genuine signatures and random forgeries (signatures from other users), since skilled forgeries are not available in these synthetic datasets. We consider experiments using the network trained on GPDS with no changes, and experiments where we fine-tune the representation to the particular dataset, following the protocol from Sect. 3.3. Results with finetuning that are significantly better (at $p < 0.01$) are shown with a bullet. We noticed an interesting trend in these results, where without fine-tuning, the performance on other datasets that contain western-style signatures is already good (around or less than 1% for MCYT, CEDAR and Brazilian PUC-PR), but for the Indian scripts the

Table 8 Comparison with the state of the art on the Brazilian PUC-PR dataset (errors in %)

References	#samples per user	Features	FRR	FAR_{random}	FAR_{simple}	$FAR_{skilled}$	AER	$AER_{genuine+skilled}$	$EER_{genuine+skilled}$
Bertolini et al. [13]	15	Graphometric	10.16	3.16	2.8	6.48	5.65	8.32	–
Batista et al. [47]	30	Pixel density	7.5	0.33	0.5	13.5	5.46	10.5	–
Rivard et al. [14]	15	ESC + DPDF	11	0	0.19	11.15	5.59	11.08	–
Eskander et al. [15]	30	ESC + DPDF	7.83	0.02	0.17	13.5	5.38	10.67	–
Present Work	15	SigNet	1.22 (± 0.63)	0.02 (± 0.05)	0.43 (± 0.09)	10.70 (± 0.39)	3.09 (± 0.20)	5.96 (± 0.40)	2.07 (± 0.63)
Present Work	15	SigNet-SPP-300dpi	0.69 (± 0.51)	0.04 (± 0.07)	0.14 (± 0.2)	9.51 (± 1.27)	2.59 (± 0.35)	5.1 (± 0.69)	1.33 (± 0.65)
Present Work	15	SigNet-SPP-300dpi (fine-tuned)	0.63 (± 0.57)	0.03 (± 0.07)	0.14 (± 0.2)	8.78 (± 1.55)	2.39 (± 0.39)	4.7 (± 0.77)	1.35 (± 0.6)

performance was much worse (3–5% EER). By fine-tuning the representation for the scripts, we obtain good performance (comparable with the previously reported in [25]). This suggests that the learned representation generalize better to users with western scripts than to other scripts. Multi-script learning approaches could be considered to improve performance on all scripts.

Lastly, Tables 5, 6, 7 and 8 compare the results we obtained with SPP Fixed (considering $EER_{user\ thresholds}$ using genuine signatures and skilled forgeries) with the state of the art in GPDS, MCYT, CEDAR and Brazilian PUC-PR, respectively. We observe results competitive to the state of the art in all datasets. In particular, in the GPDS dataset we notice big gains in performance (0.41% EER compared to 1.69% EER).

It is also worth noting that the MCYT dataset contains both offline and online signature data (for the same users). Historically, performance on online systems was greatly superior, but recent work on offline signature verification is closing the gap between the two strategies. The best results on the literature achieve 2.85% EER [48] and 3.36% EER [49] on the online MCYT dataset, while for offline signature verification, performance is achieving around 3% $EER_{user\ thresholds}$. Although these results are not directly comparable (both [48,49] implement per-user score normalization with a single global threshold), it shows that the gap between the two approaches is being reduced.

6 Conclusion

In this work, we proposed and evaluated two methods for adapting the CNN architectures to learn a fixed-size representation for signatures of different sizes. A simple method, of training a network with SPP in images of a fixed sized (and generalizing to signatures of any size), showed similar performance to previous methods, while removing the constraint of having a maximum signature size that could be processed.

Our experiments with different resolutions showed that using larger image resolutions do not always lead to improved performance. In particular, we empirically showed that using resolution higher than 100 dpi greatly improves performance if skilled forgeries (from a subset of users) are used for feature learning, but does not improve performance if only genuine signatures are used. This suggests that when learning features from skilled forgeries, the network can use detailed information about the signature strokes (e.g., if the writing is shaky, with limp strokes), while this information is ignored when only genuine signatures are used for training the CNN (when the network is only attempting to distinguish between different users).

Lastly, our experiments with transfer learning confirm previous results that features learned in one signature dataset

generalize to other operating conditions. Our results also suggest that fine-tuning the representations (on a subset of the users in the new dataset) is useful to adapt the representations to the new conditions, improving performance. Especially for signatures from different styles than used for training (e.g., CNN trained on western signatures and generalizing to other types of script), fine-tuning showed to be particularly important. Other techniques, such as multi-script learning, are also promising for this scenario.

Acknowledgements This work was supported by the Fonds de recherche du Québec - Nature et technologies (FRQNT), the CNPq Grant #206318/2014-6 and by the Grant RGPIN-2015-04490 to Robert Sabourin from the NSERC of Canada. In addition, we gratefully acknowledge the support of NVIDIA with the donation of the Titan Xp GPU used in this research.

References

1. Plamondon, R., Lorette, G.: Automatic signature verification and writer identification the state of the art. Pattern Recognit. **22**(2), 107–131 (1989)
2. Leclerc, F., Plamondon, R.: Automatic signature verification: the state of the art—1989–1993. Int. J. Pattern Recognit. Artif. Intell. **8**(03), 643–660 (1994)
3. Impedovo, D., Pirlo, G.: Automatic signature verification: the state of the art. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **38**(5), 609–635 (2008)
4. Hafemann, L.G., Sabourin, R., Oliveira, L.S.: Offline handwritten signature verification—literature review. In: International Conference on Image Processing Theory, Tools and Applications (IPTA) (2017)
5. Hafemann, L.G., Sabourin, R., Oliveira, L.S.: Writer-independent feature learning for offline signature verification using convolutional neural networks. In: The 2016 International Joint Conference on Neural Networks (2016)
6. Hafemann, L.G., Sabourin, R., Oliveira, L.S.: Learning features for offline handwritten signature verification using deep convolutional neural networks. Pattern Recognit. **70**, 163–176 (2017). https://doi.org/10.1016/j.patcog.2017.05.012
7. Rantzsch, H., Yang, H., Meinel, C.: Signature embedding: writer independent offline signature verification with deep metric learning. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Porikli, F., Skaff, S., Entezari, A., Min, J., Iwai, D., Sadagic, A., Scheidegger, C., Isenberg, T. (eds.) Advances in Visual Computing. Lecture Notes in Computer Science, pp. 616–625. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50832-0_60
8. Zhang, Z., Liu, X., Cui, Y.: Multi-phase offline signature verification system using deep convolutional generative adversarial networks. In: 2016 9th International Symposium on Computational Intelligence and Design (ISCID), vol. 02, pp. 103–107 (2016). https://doi.org/10.1109/ISCID.2016.2033
9. Hafemann, L.G., Oliveira, L.S., Sabourin, R.: Analyzing features learned for offline signature verification using Deep CNNs. In: 23rd International Conference on Pattern Recognition (2016)
10. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: European Conference on Computer Vision, pp. 346–361. Springer (2014)
11. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Trans. Pattern Anal. Mach. Intell. **37**(9), 1904–1916 (2015). https://doi.org/10.1109/TPAMI.2015.2389824
12. Guerbai, Y., Chibani, Y., Hadjadji, B.: The effective use of the one-class SVM classifier for handwritten signature verification based on writer-independent parameters. Pattern Recognit. **48**(1), 103–113 (2015). https://doi.org/10.1016/j.patcog.2014.07.016
13. Bertolini, D., Oliveira, L.S., Justino, E., Sabourin, R.: Reducing forgeries in writer-independent off-line signature verification through ensemble of classifiers. Pattern Recognit. **43**(1), 387–396 (2010). https://doi.org/10.1016/j.patcog.2009.05.009
14. Rivard, D., Granger, E., Sabourin, R.: Multi-feature extraction and selection in writer-independent off-line signature verification. Int. J. Doc. Anal. Recognit. **16**(1), 83–103 (2013)
15. Eskander, G., Sabourin, R., Granger, E.: Hybrid writer-independent-writer-dependent offline signature verification system. IET Biom. **2**(4), 169–181 (2013). https://doi.org/10.1049/iet-bmt.2013.0024
16. Nagel, R.N., Rosenfeld, A.: Computer detection of freehand forgeries. IEEE Trans. Comput. **100**(9), 895–905 (1977)
17. Justino, E.J., El Yacoubi, A., Bortolozzi, F., Sabourin, R.: An off-line signature verification system using HMM and graphometric features. In: Fourth IAPR International Workshop on Document Analysis Systems (DAS), pp. 211–222. Rio de, Citeseer (2000)
18. Oliveira, L.S., Justino, E., Freitas, C., Sabourin, R.: The graphology applied to signature verification. In: 12th Conference of the International Graphonomics Society, pp. 286–290 (2005)
19. Vargas, J.F., Ferrer, M.A., Travieso, C.M., Alonso, J.B.: Off-line signature verification based on grey level information using texture features. Pattern Recognit. **44**(2), 375–385 (2011). https://doi.org/10.1016/j.patcog.2010.07.028
20. Ylmaz, M.B., Yankolu, B.: Score level fusion of classifiers in off-line signature verification. Inf. Fusion **32**, 109–119 (2016). https://doi.org/10.1016/j.inffus.2016.02.003
21. Pal, S., Chanda, S., Pal, U., Franke, K., Blumenstein, M.: Off-line signature verification using G-SURF. In: 12th International Conference on Intelligent Systems Design and Applications, pp. 586–591. IEEE (2012)
22. Alaei, A., Pal, S., Pal, U., Blumenstein, M.: An efficient signature verification method based on an interval symbolic representation and a fuzzy similarity measure. IEEE Trans. Inf. Forensics Secur. **12**(10), 2360–2372 (2017)
23. Ferrer, M., Diaz-Cabrera, M., Morales, A.: Static signature synthesis: a neuromotor inspired approach for biometrics. IEEE Trans. Pattern Anal. Mach. Intell. **37**(3), 667–680 (2015). https://doi.org/10.1109/TPAMI.2014.2343981
24. Diaz, M., Ferrer, M.A., Eskander, G.S., Sabourin, R.: Generation of duplicated off-line signature images for verification systems. IEEE Trans. Pattern Anal. Mach. Intell. **39**(5), 951–964 (2016). https://doi.org/10.1109/TPAMI.2016.2560810
25. Ferrer, M.A., Chanda, S., Diaz, M., Banerjee, C.K., Majumdar, A., Carmona-Duarte, C., Acharya, P., Pal, U.: Static and dynamic synthesis of Bengali and Devanagari signatures. IEEE Trans. Cybern. **PP**(99), 1–12 (2017). https://doi.org/10.1109/TCYB.2017.2751740
26. Ojala, T., Pietikainen, M., Maenpaa, T.: Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Trans. Pattern Anal. Mach. Intell. **24**(7), 971–987 (2002)
27. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005, vol. 1, pp. 886–893. IEEE (2005)
28. Sabourin, R., Cheriet, M., Genest, G.: An extended-shadow-code based approach for off-line signature verification. In: Proceedings of the Second International Conference on Document Analysis

and Recognition 1993, pp. 1–5 (1993). https://doi.org/10.1109/ICDAR.1993.395795
29. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift (2015). arXiv preprint arXiv:1502.03167
30. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: International conference on Artificial Intelligence and Statistics, pp. 249–256 (2010)
31. Oquab, M., Bottou, L., Laptev, I., Sivic, J.: Learning and transferring mid-level image representations using convolutional neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1717–1724(2014). https://doi.org/10.1109/CVPR.2014.222
32. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: delving deep into convolutional nets (2014). arXiv preprint arXiv:1405.3531
33. Vargas, J., Ferrer, M., Travieso, C., Alonso, J.: Off-line handwritten signature GPDS-960 corpus. In: 9th International Conference on Document Analysis and Recognition, vol. 2, pp. 764–768 (2007). https://doi.org/10.1109/ICDAR.2007.4377018
34. Ortega-Garcia, J., Fierrez-Aguilar, J., Simon, D., Gonzalez, J., Faundez-Zanuy, M., Espinosa, V., Satue, A., Hernaez, I., Igarza, J.J., Vivaracho, C., et al.: MCYT baseline corpus: a bimodal biometric database. IEE Proc. Vis. Image Signal Process. **150**(6), 395–401 (2003)
35. Kalera, M.K., Srihari, S., Xu, A.: Offline signature verification and identification using distance statistics. Int. J. Pattern Recognit. Artif. Intell. **18**(07), 1339–1360 (2004). https://doi.org/10.1142/S0218001404003630
36. Freitas, C., Morita, M., Oliveira, L., Justino, E., Yacoubi, A., Lethelier, E., Bortolozzi, F., Sabourin, R.: Bases de dados de cheques bancarios brasileiros. In: XXVI Conferencia Latinoamericana de Informatica (2000)
37. Hu, J., Chen, Y.: Offline signature verification using real adaboost classifier combination of pseudo-dynamic features. In: 12th International Conference on Document Analysis and Recognition, pp. 1345–1349 (2013). https://doi.org/10.1109/ICDAR.2013.272
38. Serdouk, Y., Nemmour, H., Chibani, Y.: New gradient features for off-line handwritten signature verification. In: 2015 International Symposium on Innovations in Intelligent Systems and Applications (INISTA), pp 1–4 (2015). https://doi.org/10.1109/INISTA.2015.7276751
39. Soleimani, A., Araabi, B.N., Fouladi, K.: Deep multitask metric learning for offline signature verification. Pattern Recognit. Lett. **80**, 84–90 (2016). https://doi.org/10.1016/j.patrec.2016.05.023
40. Gilperez, A., Alonso-Fernandez, F., Pecharroman, S., Fierrez, J., Ortega-Garcia, J.: Off-line signature verification using contour features. In: 11th International Conference on Frontiers in Handwriting Recognition, Montral, Qubec-Canada, 19–21 Aug 2008, CENPARMI, Concordia University (2008)
41. Wen, J., Fang, B., Tang, Y.Y., Zhang, T.: Model-based signature verification with rotation invariant features. Pattern Recognit. **42**(7), 1458–1466 (2009). https://doi.org/10.1016/j.patcog.2008.10.006
42. Ooi, S.Y., Teoh, A.B.J., Pang, Y.H., Hiew, B.Y.: Image-based handwritten signature verification using hybrid methods of discrete Radon transform, principal component analysis and probabilistic neural network. Appl. Soft Comput. **40**, 274–282 (2016)
43. Chen, S., Srihari, S.: A new off-line signature verification method based on graph. In: 18th International Conference on Pattern Recognition (ICPR'06), vol. 2, pp. 869–872 (2006). https://doi.org/10.1109/ICPR.2006.125
44. Kumar, R., Kundu, L., Chanda, B., Sharma, J.D.: A writer-independent off-line signature verification system based on signature morphology. In: Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia, ACM, New York, NY, IITM '10, pp. 261–265 (2010). https://doi.org/10.1145/1963564.1963610
45. Kumar, R., Sharma, J.D., Chanda, B.: Writer-independent off-line signature verification using surroundedness feature. Pattern Recognit. Lett. **33**(3), 301–308 (2012). https://doi.org/10.1016/j.patrec.2011.10.009
46. Bharathi, R., Shekar, B.: Off-line signature verification based on chain code histogram and support vector machine. In: 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 2063–2068 (2013). https://doi.org/10.1109/ICACCI.2013.6637499
47. Batista, L., Granger, E., Sabourin, R.: Dynamic selection of generative–discriminative ensembles for off-line signature verification. Pattern Recognit. **45**(4), 1326–1340 (2012). https://doi.org/10.1016/j.patcog.2011.10.011
48. Rua, E.A., Castro, J.L.A.: Online signature verification based on generative models. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **42**(4), 1231–1242 (2012). https://doi.org/10.1109/TSMCB.2012.2188508
49. Fierrez, J., Ortega-Garcia, J., Ramos, D., Gonzalez-Rodriguez, J.: HMM-based on-line signature verification: feature extraction and signature modeling. Pattern Recognit. Lett. **28**(16), 2325–2334 (2007). https://doi.org/10.1016/j.patrec.2007.07.012