Offline Signature Verification with Convolutional Neural Networks

Gabe Alvarez galvare2@stanford.edu
Blue Sheffer bsheffer@stanford.edu
Morgan Bryant mrbryant@stanford.edu

Abstract

Signature verification is an important biometric technique that aims to detect whether a given signature is forged or genuine. It is essential in preventing falsification of documents in numerous financial, legal, and other commercial settings. Our project aims to automate the process of signature verification by using convolutional neural networks (CNNs). Our model is based on the VGG16 architecture, and we use the ICDAR 2011 SigComp dataset to train our model with transfer learning. When classifying whether a given signature was a forgery or genuine, we achieve accuracies of 97% for Dutch signatures and 95% for Chinese Signatures. We also performed several experiments altering the types of training data and prediction task to make it more relevant to real-life applications, for which our methods seem promising but for which we were not able to achieve results much higher than a naive baseline.

1. Introduction

Biometric authentication is the process of verifying the identity of individuals based on their unique biological characteristics. It has become a ubiquitous standard for access to high security systems. Current methods in machine learning and statistics have allowed for the reliable automation of many of these tasks (face verification, fingerprinting, iris recognition). Among the numerous tasks used for biometric authentication is signature verification, which aims to detect whether a given signature is genuine or forged.

Signature verification is essential in preventing falsification of documents in numerous financial, legal, and other commercial settings. The task presents several unique difficulties: high intra-class variability (an individual's signature may vary greatly day-to-day), large temporal variation (signature may change completely over time), and high inter-class similarity (forgeries, by nature, attempt to be as indistinguishable from genuine signatures as possible).

There exist two types of signature verification: online and offline. Online verification requires an electronic signing system which provides data such as the pen's position, azimuth/altitude angle, and pressure at each time-step of the signing. By contrast, offline verification uses solely 2D visual (pixel) data acquired from scanning signed documents. While online systems provide more information to verify identity, they are less versatile and can only be used in certain contexts (e.g. transaction authorization) because they require specific input systems.

Figure 1. Superimposed examples of multiple genuine signatures from the same ID, indicating high intraclass variability (from [10])

We aim to build an offline signature verification system using a Convolutional Neural Network (CNN). Our paper focuses on building systems trained on data with varying degrees of information, as well as experimenting with different objective functions to obtain optimal error rates.

2. Related Work

2.1. Tasks Within Signature Verification

Unfortunately, the task of offline signature verification does not have a uniform standard for performance evaluation. In 2004, the University of Hong Kong hosted the first international signature verification competition [2], but the competition was solely for online signatures. Nonetheless, this paved the way for future competitions for both online and offline verification ([3], [4]). Beyond the online/offline distinction, there are two further distinctions that define the task of signature verification, which we will refer to as "forgery exposure" and "writer-dependence".

The forgery exposure of the task is determined by what kinds of signatures the model is trained on. Some authors train on forgeries and genuine signatures, but test on different forgeries and genuine signatures during testing; some train on forgeries for identities that exist solely in the training set (then test on forgeries and genuine signatures for novel IDs); some authors do not use forgeries at all during training. It is worth noting that only the latter two tasks are

reasonable for practical applications. That is, while a system could have access to a development set that includes forgeries, it is unlikely that forgeries can be acquired for all users of that system.

We have trained systems for the first two of these varieties. In the experiments section, we detail exactly how we use the training and test data to accomplish each task.

The writer-dependence distinction refers to whether the model has a different classifier for each identity, or a single classifier for all identities. Writer-dependent models, which are more common in the literature, use a different classifier for each user. At test time, the model is given the ID of the signature for which it is testing authenticity. Writer-independent models use a single classifier for all identities. We have developed both writer-dependent and writer-independent models in our report.

2.2. Feature Selection

Most existing models in the literature use explicit feature extraction, including geometric [9], graphometric [10], directional [11], wavelet [12], shadow [13], and texture [14] features . Only in recent years has feature learning been explored [15]. We feed in raw pixels to our network, letting the CNN learn the relevant features for signature verification.

2.3. Competing Models

The signature verification literature includes several examples of Hidden Markov Models [18], Neural Networks [17], Support Vector Machines [19], and other machine learning models.

In 2012, Khalajzadeh et al. [16] used CNNs for Persian signature verification, which is the only report of CNNs being used in the offline signature verification literature. Unfortunately, the paper includes very little information about their methodology (i.e. forgery exposure, writer dependence). Based on their explanation, we assume their model is trained on forgeries and genuine signatures for all IDs, and thus can be compared with our work in our main experiment. There results only report an average of 99.86 for validation performance, and mean squared error, making it difficult to fully compare our model to theirs.

3. Methods

3.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have proven successful in recent years at a large number of image processing-based machine learning tasks. Many other methods of performing such tasks revolve around a process of feature extraction, in which hand-chosen features extracted from an image are fed into a classifier to arrive at a classification decision. Such processes are only as strong as the chosen features, which often take large amounts of care and effort to construct.

By contrast, in a CNN, the features fed into the final linear classifier are all learned from the dataset. A CNN consists of a number of layers, starting at the raw image pixels, which each perform a simple computation and feed the result to the next layer, with the final result being fed to a linear classifier. The layers' computations are based on a number of parameters which are learned through the process of backpropagation, in which for each parameter, the gradient of the classification loss with respect to that parameter is computed and the parameter is updated with the goal of minimizing the loss function. Exactly how this update is done and what the loss function is are tunable hyperparameters of the network, discussed in more detail below. For more details on backpropagation, see [5].

3.2. VGG Architecture

The architecture of a CNN determines how many layers it has, what each of these layers is doing, and how the layers are connected to each other. Choosing a good architecture is crucial to successful learning with a CNN. For our main training tasks, we used the VGG-16 CNN architecture [6]. This network contains a total of 16 layers with learnable parameters. These layers are of the following types:

Fully Connected Layers: Fully connected layers apply an affine transformation to their inputs. Mathematically, a fully-connected layer from n inputs to h outputs works as follows:

$$f(X) = WX + b$$

$$W \in \mathbb{R}^{nxh}, b \in \mathbb{R}^h$$

In a fully-connected layer, every output depends on every input according to the weight matrix W, a learnable parameter. Outputs also depend on a bias term b which is learnable but doesn't depend on the inputs.

ReLU Nonlinearity: The Rectified Linear Unit is a commonly used activation function after fully-connected layers. This layer applies the following mathematical operation to input X:

$$f(X) = max(X, 0)$$

Here, the maximum is taken elementwise. ReLU layers do not have any learnable parameters. ReLU is commonly used in modern neural networks instead of other possible activation functions such as sigmoid and tanh for several reasons. One reason is that their computation is very simple, saving time during training that would be spent computing exponentials for sigmoid and tanh. ReLU neurons also do not become saturated at high input values, meaning their gradient does not vanish to zero when receiving such values. This allows the neurons to continue learning in

scenarios where other activation functions would have vanishing gradients. However, because the gradient of a ReLU neuron is 0 for negative inputs, it's also possible for these neurons to stop learning in cases where their input never produces positive values.

Softmax Nonlinearity: The Softmax nonlinearity appears in the final layer of the neural network and computes final class scores that will be fed into the loss function or outputted during testing. It has no learnable parameters. These scores have an interpretation as the neural network's estimated probabilities for each class. Note that all the output values produced by the Softmax function add to one. Mathematically, the i^{th} class probability $f(x)_i$ is computed as follows:

$$f(x)_i = \frac{e^{x_i}}{\sum e^x_j}$$

Using Softmax to compute class scores is an attractive option because of its ease of interpretation.

Convolutional Layers: Convolutional layers process an input image by sliding a number of small filters across each possible region and outputting the dot product of the filter and the image at each region. They are similar to fully-connected layers, but with restrictions on which input neurons are connected to which outputs. Specifically, outputs are only connected to inputs of a small region, and all weights for each filter are tied together rather than being allowed to be learned independently. The learnable parameters for a convolutional layer are the weights of each filter and one bias value for each filter. Convolutional layers lend themselves naturally to understanding of images, in which we often want to extract features by looking at small areas of an image, where we don't care exactly where in the image the feature is. For example, a face is still a face regardless of where in the image it is. Our architecture uses 3x3 filters, with more filters used per layer as the network gets deeper.

Max Pooling Layers: Max pooling layers reduce the size of an image by combining 2x2 regions of the input into a single output value. For each 2x2 region of input, the output value is simply the max value of those 4 input pixels. This layer has no learnable parameters. This layer cuts both the width and height of the image in half as it goes to the next layer. In networks with max pooling layers, the input width and height decrease as the image is forwarded through the network, and the number of filters (depth) tends to increase. This corresponds to processing the image at a higher level of abstraction in which features correspond to larger region of the input image.

Dropout Layers: Dropout layers are a non-deterministic nonlinearity used in many modern neural networks. A dropout layer takes in a number of inputs and for each input, sets it to 0 with probability p and leaves it unchanged with probability (1-p). During test time, dropout layers instead behave deterministically and multiply all input values by (1-p), the average value by which they were multiplied during training. The dropout value p is a tunable hyperparameter for the network. Dropout can be interpreted as a form of regularization, as using dropout during training forces the network to have many ways of computing a correct result rather than just one. This keeps the network from relying too heavily on any single connection.

3.3. Training the Network

As our loss function, we chose the standard Categorical Cross-Entropy loss with L_2 regularization, which is computed from the outputs of the final Softmax layer. If X is the output from the Softmax layer for a given training data point, the unregularized loss is defined as:

$$-log(X_{y_i})$$

Where y_i is the correct class label. This is a standard loss function for CNN's. One important feature of this loss function is that unlike other choices such as the Hinge Loss function, it does not become satisfied with results that are "good enough" and seeks to push the class scores closer and closer to perfection, corresponding to all the probability mass being assigned to the correct class.

Using the gradient of this loss function, we update all parameters in the network using the Nesterov Momentum update. Our reasoning for using this particular update method is explained in our Results section. This update keeps track of a variable "v" that is a function of the magnitude of previous updates, allowing the learning rate to depend on this variable. This type of update is more tolerant to different values of learning rate and tends to converge relatively quickly, as it can adapt over time based on how quickly the parameters are changing.

3.4. Transfer Learning

VGG-16 is a large neural network with a huge number of learnable parameters. Effectively training a network this large from scratch requires a large dataset and access to substantial computational resources. However, this problem can be averted by leveraging similarities between different image datasets. Specifically, for most reasonable image classification tasks, the low-level features learned by the first few layers of a network will be roughly the same regardless of the dataset. This means that we can initialize our neural network with parameter values learned from a different dataset and expect that the values for the first layers of the network will work well without having to be trained. This process is known as transfer learning.

For our task, we downloaded pretrained weights trained on ImageNet for the VGG-16 architecture by following the directions here: (https://github.com/Lasagne/Recipes/blob/master/modelzoo/vgg16.py).

We initialized our model with the pretrained weights and allowed only the fully-connected layers at the end to train, keeping the convolutional layers fixed throughout training (this is a decision we came to based on testing different options, see results for further discussion).

3.5. Comparison Networks

In our final experiment, we make use of a model proposed in [7] for image patch comparison. The paper proposes several models for image patch comparison, including siamese (similar to [8]), pseudo siamese, and 2-channel networks. In the 2-channel network, two images are stacked (i.e. each image serves as a channel for the composite, paired image) as the input to a convolutional network. The convolutional network then leads to a full connected linear decision layer with 1 output that indicates the similarity of the two patches. They found the the 2-channel network outperforms the other proposed architectures. Further details about our adaptation are in the Results section. The following is a sketch of a 2-channel architecture:

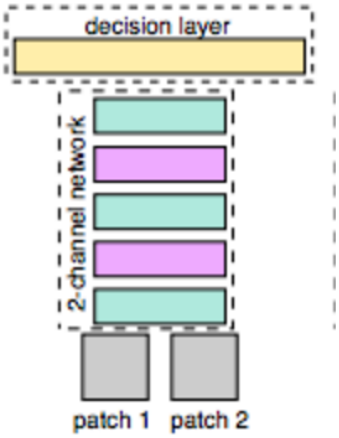

Figure 2. 2-channel network for image patch comparison (from [7])

4. Dataset and Features

4.1. Dataset

Our dataset comes from the International Conference on Document Analysis and Recognition (ICDAR) 2011 SigComp international signature verification competition [3]. The dataset includes both online and offline signatures (of which we only use the latter) for both Chinese and Dutch signers. The dataset is split into a training set and testing set of nonoverlapping IDs. The Dutch training set includes a total of 366 images for 10 IDS, with about 25 genuine signatures and 11 forged signatures for each ID. The Chinese

Figure 3. Example images from the SigComp 2011 dataset

training set contains 576 images for 10 IDs, with about 25 genuine signatures and 30 forgeries for each ID.

Both testing sets include a “reference” and“questioned” set, where the references are known genuine signatures, and the questioned signatures are either genuine or forged. The The Dutch testing set had 45 IDs, each with 10 reference signatures and about 25 questioned signatures. The Chinese testing set had 10 IDs, each with about 12 reference signatures and about 50 questioned signatures.

We conformed to the prescribed training testing split for our last experiment (Writer-Dependent Comparison Task), but modified the split to increase the number of examples in our training set for the other tasks.

5. Results

5.1. Evaluation Metrics

During training and model tuning, we held out a test set which was not tested on until the final submission of our project. We report accuracy on this set as well as on a validation set, which we used to tune our hyperparameters as we iterated on our models. The model we consider ”best” is the one which achieved the best performance on the validation set, and we ran this best model on the test set to achieve the results for the test set.

All of our tasks involve the test-time behavior of classifying whether a given signature is forged or genuine. We evaluate our performance using three metrics: classification accuracy, False Acceptance Rate, and False Rejection Rate. They are defined as follows, where genuine signatures are considered ”positive” examples:

$$Accuracy = \frac{true\_positives + true\_negatives}{num\_data}$$

$$FAR = \frac{false\_negatives}{true\_positives + false\_negatives}$$

$$FRR = \frac{false\_positives}{false\_positives + true\_negatives}$$

Higher accuracy is better, and lower FAR and FRR are better. These metrics can be interpreted in different ways, but it is arguable that in applications it is most important to achieve low FAR scores, as a good signature verification system should not let forgeries through, whereas accidentally classifying a genuine signature as a forgery is less of a problem because one can simply ask the person to sign their signature again.

5.2. Main Task

Our first task was to train a CNN to recognize whether individual signatures are forged or genuine, having seen examples of both forged and genuine versions of that same person's signature during training.

For this task, we label all data points in our SIGCOMP 2011 Dataset as either genuine or forged, discarding the identity of the signer. We train one network on signatures from all identities together. For each language, all the data for that language is split into training, validation, and test data randomly, meaning validation and test data consist of new copies of signatures of the same people whose signatures are in the training set. For both Chinese and Dutch, we put 80% of our data in the training set and 10% in each of the validation and test sets.

Our results are summarized in the following table:

Category	Dutch Result	Chinese Result
Validation Accuracy	97%	95%
Validation FAR	3.6%	4.7%
Validation FRR	3.6%	5.6%
Test Accuracy	94%	88%
Test FAR	13.32%	8.2%
Test FRR	3.13%	18.2%

Our focus when tuning the network was to achieve the best possible classification accuracy, FAR, and FRR scores for the Dutch dataset. Using these same parameters, we then tested on the Chinese dataset as well, achieving the results above. This process explains in part why the results for the Chinese dataset are significantly lower than for Dutch.

Looking at the plots of training and validation loss below, we conclude that we have overfit our training set to at most a small extent. When we train for longer than 20 epochs, the training loss does not become significantly less thant the validation loss until after roughly 15-20 epochs, and after this point, training loss continues to decrease but validation loss begins to increase. At this point, validation accuracy peaks as well and then begins to decrease.

Our test accuracies are in the same ballpark as the validation accuracies but are somewhat lower. We tuned several hyperparameters using the validation dataset, including learning rate, regularization constant, and number of layers to train. Because the settings we chose reflected the best performance on the validation set, it's likely that the hyperparameters were to some extent fitting noise in the validation set.

5.3. Parameter Updates

One important aspect of our training is which parameter update algorithm we use. We considered a number of different parameter update schemes as trials. We tested various update rules on our main task. Unsurprisingly, SGD was consistently slower to converge than any of the other methods. We had trouble finding hyperparameters that got Adam to converge, although it's possible that this is because we did not try enough combinations. This result was surprising since Adam is often recommended as an update rule instead of simpler methods, but we were unable to get Adam to achieve the same accuracies as RMSprop or Nesterov. Our final choice of update algorithm was Nesterov Momentum, which performed similarly to the other best algorithms in terms of validation accuracy and which was attractive due to its simplicity. The following indicates validation accuracy over time with various update algorithms:

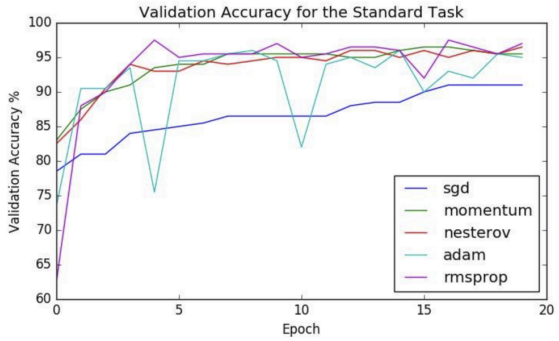

Figure 4. Validation accuracies on test sets for each kind of parameter updates

5.4. Hyperparameter Choices for Main Task

For this task, our best model used L_2 regularization with a regularization constant of 2e-3. We found this hyperparameter using a coarse-to-fine search approach and selecting values that produced the best validation accuracy. Using a nonzero regularization constant was essential to our results, as running the network with no regularization produces severe overfitting.

We used a learning rate of 1e-4, which was the default learning rate for the Nesterov Momentum update. We conducted a few training runs with values near this on both sides but did not see better results.

For our dropout layers, we used a dropout value of 0.5. This was the value that was coded into the VGG-16 architecture that we downloaded.

We trained for a total of 16 epochs to find our best model on the Dutch validation dataset. The Chinese dataset achieved its maximum validation performance after 20 epochs. For both these training runs, validation set performance began to decrease after this point.

We chose to let only the fully-connected layers at the end of the network train, keeping the pretrained weights fixed for the convolutional layers. We made this choice because it led to better performance and faster convergence, and also because allowing more layers to train made our training slower and made it use more computational resources.

5.5. Main Task Training Plots

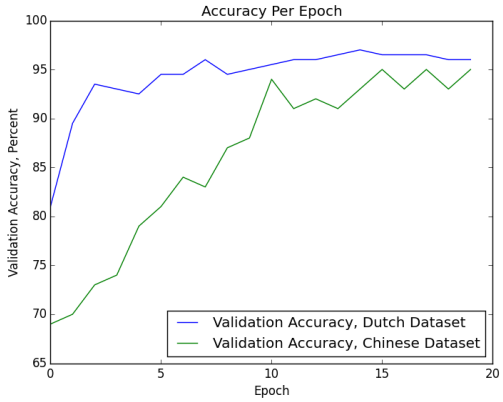

Figure 5. Accuracy Per Epoch for the Main Task

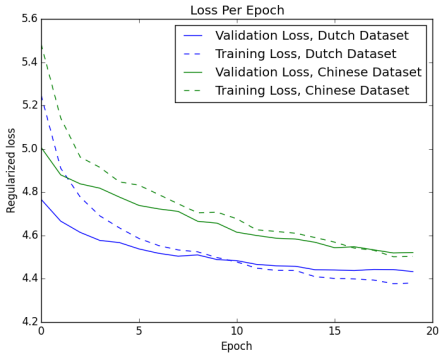

Figure 6. Loss Per Epoch for the Main Task

5.6. Unseen Identities Experiment

We are able to achieve high results on this main task because the test data resembles the training data, in that it contains signatures from the same people. We wished to expand our results to more difficult tasks in which the datasets were less similar. One such task is to test the network on a set of signatures that are by entirely different people than the training set.

For this experiment, we split our data set into training, test, and validation sets containing 80%, 10%, and 10% of the data respectively. The difference, however, is that the validation and test sets consist only of signatures by new people and forgeries by new people which are not present in the training set. A network that is successful at this task would be learning some sort of information about distinguishing signatures from their forgeries that is true across the signatures of all people. It is unclear whether such information exists, and as a result we did not expect very high performance at this task. For this task, our best model used L_2 regularization with a regularization constant of 3e-3, a learning rate of 1e-4, and a dropout value of 0.6. Our results are summarized in the following table:

Category	Dutch Result
Validation Accuracy	73%
Validation FAR	22.2%
Validation FRR	28.3%
Test Accuracy	76%
Test FAR	28.6%
Test FRR	21.8%

Because of the class skew of the Dutch dataset, a net that predicted genuine for every signature would achieve a classification accuracy of 67% (with an FAR of 100% and FRR of 0%). Our performance is somewhat better than this, and our mistakes are more evenly spread out between both types of misclassifications. However, these results are not much better than guessing all genuine. This indicates that our network was able to pick up on some information that remains constant across the signatures of all people, but this information was only enough to achieve an accuracy slightly better than a naive baseline.

Our training runs for this task fell victim to overfitting. Increasing the regularization parameter and dropout percentage were helpful, but only for small increases - for large increases, the network stopped learning at all. Training for fewer epochs was also helpful to combat overfitting – our best models were trained for only 5 epochs, and performance started to decline sharply after this due to overfitting. Training loss continued to decrease but validation loss began to increase at this point. It is reasonable to expect that overfitting would be a problem with this type of task, as it is easy to learn information that distinguishes signatures from their forgeries in a way that is specific to the person, but more difficult to get this to generalize to new people.

In terms of classification accuracy, the results from this experiment are little better than guessing the same, most common class for every single test example. This indicates that there is not enough information present in this task to

make reasonable predictions, at least with our CNN method. We decided not to continue this experiment with the Chinese dataset, moving on instead to a more tractable problem detailed below.

5.7. Writer-Dependent Comparison Task

We adapted the 2-channel network from [CITE] for our task. We train our network on (genuine, genuine) pairs and (genuine, forged) pairs for all IDs in our training set (n = 10 IDs). We test solely on unseen IDs. Thus, the network should learn to tell when two signatures are the same or different. For each test ID, there are 10 reference signatures that are known to be genuine, and 25 questioned signatures, which are either genuine or forged. At test time, we send every questioned signature into the network paired with each reference signature. Thus, each questioned-reference pairing serves as a vote, and if the network deems that the questioned signature differs from too many of the reference signatures, it is classified as a forgery.

Category	Dutch Result
Validation Accuracy	67.1%
Validation FAR	33.0%
Validation FRR	33.0%

Unfortunately, this experiment had comparable results to the unseen identities experiment. However, our results suffered from a peculiar error that gives room for hope in future experiments. The success of our model varied significantly from ID to ID. On some IDs, the model perfectly classified the forgeries and genuine. When our model failed, it tended to fail for most examples within that ID, usually by either guessing uniformly genuine or forged.

6. Conclusions

We experimented with several variations on signature verification tasks. We showed that convolutional neural networks do an excellent job of verifying signatures when allowed access during training to examples of genuine and forged signatures of the same people whose signatures are seen at test time. We then conducted an experiment where we tested our network on the signatures of new people whose signatures had not been seen at all during training, resulting in performance little better than a naive baseline due to the inherent difficulty of this task. Finally, we proposed a novel architecture for the comparison of signatures which has promise for future work in signature verification, specifically in situations where a possibly-forged signature can be compared to known genuine signatures of a specific signer.

We propose two directions for future work. First, access to more resources would allow us to achieve better performance on our main task. Specifically, being able to train on a larger dataset with more signature examples per person could achieve higher accuracies, as well as training a larger network for more epochs, which we were not able to do due to time and computational resource constraints. We were also constrained here by the fact that our dataset was relatively small, only on the order of thousands of examples, and that it is difficult to find good publically available signature datasets.

The Writer Dependent Comparison Task also appears to be a promising direction for future exploration. This task is attractive because it mirrors the situation of a real-world application of signature verification. Although we were unable to achieve high results at this task, the literature on signature verification indicates that our method is promising. Having access to more data and more computational resources would likely allow us to achieve stronger results at this task as well. With access to these, we would train our model on larger datasets and allow more layers to train for more epochs. This problem would also require more time spent carefully tuning the network, which we were unfortunately unable to do for this problem.

References

[1] E. J. Justino, A. El Yacoubi, F. Bortolozzi, and R. Sabourin, An off-line signature verification system using HMM and graphometric features, in Fourth IAPR International Workshop on Document Analysis Systems (DAS), Rio de. Citeseer, 2000, pp. 211222.

[2] Yeung, D.-Y., Chang, H., Xiong, Y., George, S.E., Kashi, R.S., Matsumoto, T., Rigoll, G.: SVC2004: "First International Signature Verification Competition" in ICBA 2004. LNCS, vol. 3072, pp. 16-22. Springer, Heidelberg

[3] Liwicki, M., Malik, M., van den Heuvel, C., Chen, X., Berger, C., Stoel, R., Blumenstein, M., Found, B.: Signature Verification Competition for Online and Offline Skilled Forgeries (SigComp2011). in 2011 International Conference on Document Analysis and Recognition

[4] J. Vargas, M. Ferrer, C. Travieso, and J. Alonso, Off-line Handwritten Signature GPDS-960 Corpus, in Ninth International Conference on Document Analysis and Recognition, 2007. ICDAR 2007, vol. 2, Sep. 2007, pp. 764768.

[5] A. Karpathy "Backpropagation, Intuitions" from Stanford's CS231n, http://cs231n.github.io/optimization-2/, 2015

[6] K. Simonyan, A. Zisserman "Very Deep Convolutional Networks for Large-Scale Image Recognition" in ICLR 2015

[7] S. Zagoruyko, N. Komodakis "Learning to Compare Image Patches via Convolutional Neural Networks" CVPR 2015, arXiv:1504.03641v1

[8] J. Bromley, J. W. Bentz, L. bottou, I. Guyon, Y. LeCun, C. Moore, E. Sa ckinger, and R. Shah, Signature verification using a siamese time delay neural network, Int. J. Pattern Recognit. Artificial Intell., vol. 7, no. 4, pp. 669687, Aug. 1993.

[9] H. Baltzakis and N. Papamarkos, A new signature verification technique based on a two-stage neural network classifier, Engineering applications of Artificial intelligence, vol. 14, no. 1,

pp. 95103, 2001.

[10] L. S. Oliveira, E. Justino, C. Freitas, and R. Sabourin, The graphology applied to signature verification, in 12th Conference of the International Graphonomics Society, 2005, pp. 286 290

[11] J.-P. Drouhard, R. Sabourin, and M. Godbout, A neural network approach to off-line signature verification using directional PDF, Pattern Recognition, vol. 29, no. 3, pp. 415424, 1996.

[12] P. S. Deng, H.-Y. M. Liao, C. W. Ho, and H.-R. Tyan, Wavelet-Based Off-Line Handwritten Signature Verification, Computer Vision and Image Understanding, vol. 76, no. 3, pp. 173190, Dec. 1999

[13] R. Sabourin and G. Genest, An extended-shadow-code based approach for off-line signature verification. I. Evaluation of the bar mask definition, in Conference on Pattern Recognition, 1994. Vol. 2 - Conference B: Computer Vision amp; Image Processing., Proceedings of the 12th IAPR International, vol. 2, Oct. 1994, pp. 450453 vol.2.

[14] T. Ojala, M. Pietikinen, and D. Harwood, A comparative study of texture measures with classification based on featured distributions, Pattern Recognition, vol. 29, no. 1, pp. 5159, Jan. 1996.

[15] N. A. Murshed, R. Sabourin, and F. Bortolozzi, A cognitive approach to off-line signature verification, International Journal of Pattern Recognition and Artificial Intelligence, vol. 11, no. 05, pp. 801825, 1997.

[16] H. Khalajzadeh, M. Mansouri, and M. Teshnehlab, Persian Signature Verification using Convolutional Neural Networks, in International Journal of Engineering Research and Technology, vol. 1. ESRSA Publications, 2012.

[17] K. Huang and H. Yan, Off-line signature verification based on geometric feature extraction and neural network classification, Pattern Recognition, vol. 30, no. 1, pp. 917, Jan. 1997.

[18] L. Batista, E. Granger, and R. Sabourin, Dynamic selection of generativediscriminative ensembles for off-line signature verification, Pattern Recognition, vol. 45, no. 4, pp. 13261340, Apr. 2012.

[19] E. zgndz, T. entrk, and M. E. Karslgil, Off-line signature verification and recognition by support vector machine, in European signal processing conference, EUSIPCO, 2005.