Offline Signature Recognition with Convolutional Neural Networks

Diego Kaulen
dkaulen@stanford.edu

Kaitlyn Baab
kbaab@stanford.edu

Abstract

Signature recognition is an important technique, especially in financial and legal contexts, to prevent fraud and verify an individual's identity. Applying convolutional neural networks (CNNs) to the signature recognition problem has recently shown very promising results. Our project aims to implement an existing CNN (LS2Net from Çalik et al.) and iterate on this approach by adjusting aspects of the network. We used 2,160 signatures from the GPDS Synthetic Offline Signature dataset in our approach. When identifying whether a signature was genuine or forgery, we achieved accuracies of ~99% and 54% in the training and test sets, respectively. These results show overfitting is a challenge of our model and therefore explored techniques to reduce overfitting and ultimately found only regularization improved the performance.

1. Introduction

Handwritten signatures have an important role in society today. Signatures are a common and widely accepted method to authenticate an individual's identity and to bind an individual to the contents in a document. They are used in many different settings but have prevalence in financial and legal contexts. Signature fraud, such as forging someone's signature on a check, is an issue today. Signature recognition is important to prevent fraud and verify someone's identity.

There are two main signature recognition methods: online / dynamic (electronic system with pressure sensitive pen or similar device to record dynamic information) and offline / static (scanned document consisting of 2D pixel data)[1]. Though online signature recognition can provide more verification information, they are harder to implement and therefore offline signatures are a more prevalent form of verification. Given the scale and importance of offline signatures, we decided to focus on the offline signature verification method.

With the advancements in image recognition, convolutional neural networks (CNN) have been applied to improve existing signature recognition methods and appear to be the best approach. In this work, we implement an existing and promising CNN model for signature recognition and explore modifications to this model.

2. Related work

Even with the recent research and algorithm development for signature recognition, there is no one standard for performance evaluation. In addition, performance is still lacking for large-scale signature datasets [1]. When evaluating the existing literature, we found there are many examples of different approaches to signature recognition, including: Hidden Markov Models [2], Support Vector Machines [3], Neural Networks [1,4], and other machine learning models. A significant amount of the work to-date has depended on feature extraction and only recently has feature learning been used. Çalik et al. [1] and Poddar et. al. [4] are two recent works (within the last 2 years) where feature learning has been implemented for signature recognition.

The first published use of CNNs for signature recognition was by Khalajzadeh et al. [5] in 2012 where CNN was developed for Persian Signature Recognition. Since then, several additional examples of using CNNs for signature recognition have been published with Çalik et al. and Poddar et. al. being two promising examples. Both papers focused on offline signature recognition. Poddar et. al. [4] obtained 90-94% accuracy for signature recognition when using a dataset comprising of 1320 pictures, while Çalik et al. [1] achieved 96.91% accuracy score but used a much larger dataset (closer to 200k images with 4k unique signers).

Though the results from both of these papers are impressive in the field, we decided to focus on Calik et. al. for our project due to the clarity and detail of their approach. In this paper, we take the existing CNN approach developed by Çalik et al. [1] and iterate on it by testing additional features (e.g., hyperparameters, CNN layers).

3. Dataset and Features

In our work we have focused on using one main dataset - GPDS Synthetic Offline Signature dataset [6], the same dataset used by Çalik et al. This dataset contains 4,000 synthetic individuals with 24 genuine signatures for each individual and 30 forgeries of his/her signature, for a total of 54 images associated with each individual. See example signature from dataset in Figure 1 and corresponding forgery in Figure 2. This example also shows the similarity between real signatures and forgeries highlighting the

difficulty to distinguish between the two. These signatures were generated with different pens (e.g., pen thickness) and have different pixel sizes. The signature files are in "jpg" format and have a resolution of 600 dpi. Given these differences across signatures, we must apply pre-processing to images so they can be appropriate inputs to the CNN model.

Figure 1. Example genuine signature from GPDS Synthetic Offline Signature dataset.

Figure 2. Example of forgery signature from GPDS Synthetic Offline Signature dataset

Due to memory limitations, we were not able to use entire dataset and instead our work focused on signatures from 40 individuals, with 24 genuine signatures for each individual and 30 forgeries of his/her signature. This resulted in a total dataset of size of 2,160 images (54x40). The dataset was then randomized and split 75% for training and 25% for testing.

4. Methods

We based our model on the work by Çalik et al. [1], in which they propose a new convolutional neural network (CNN) structure named Large-Scale Signature Network (LS2Net). The main structure is composed of sequential blocks of 2D Convolutional layers, Batch Normalization, ReLU Activations and Max Pooling. It is preceded by a Pre-Processing block and succeeded by a Fully Connected Layer and a Softmax Score set of neurons.

We implemented a smaller version of LS2Net and explored several modifications detailed in the following subsections. The general structure of the model we developed is depicted in Figure 3.

4.1. Pre-processing

Instead of following the comprehensive pre-processing described in [1] (Otsu Thresholding, Image Opening, Crop and Skeletonization and Dilating and Square Reshaping), we standardized all images to the same size (288 x 720). We arrived at this image size as it provided a good balance: it was not either too big to trigger memory errors, or too small to prevent the algorithm to capture the details in each of the signatures. The aspect ratio (2:5) is an approximate average of the aspect ratios of each of the 2,160 images we

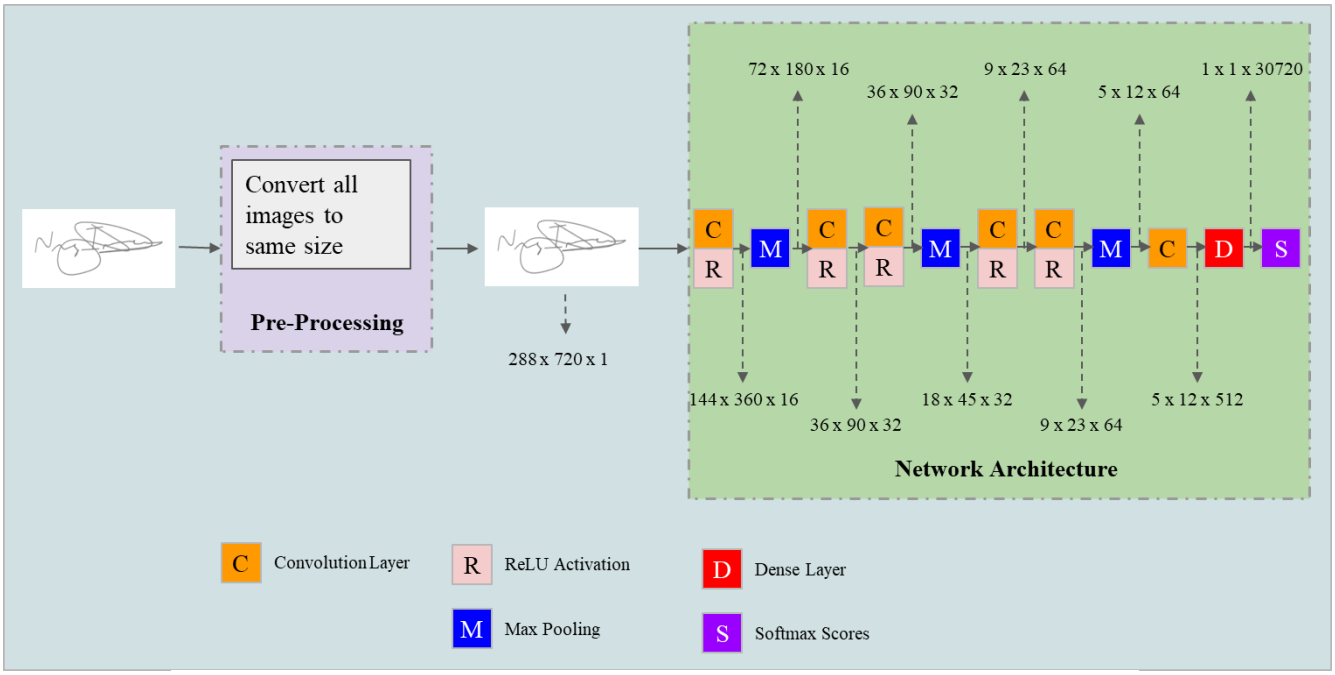


Figure 3. Signature recognition system. Layers of network and volume dimensions noted in diagram.

processed. This resulted in some of the signatures being slightly misshapen as a result of the pre-processing but still found the pre-processing sufficient for our purposes.

This change in our approach significantly impacted the performance of our algorithm. In the previous version, we were padding all images to make them the same size. As signatures from the same person had similar sizes, the algorithm could rely on the size of the signature to effectively detect whom the signature was from. However, this is not the way in which the algorithm should work: Real applications would demand users to sign in a clearly delimited space, in which all signatures would be approximately the same size.

4.2. Network Architecture

Compared with the approach taken by Çalik et al. [1], a smaller number of channels in each of the Conv2D layers was used. The original LS2Net uses 1, 128, 128, 256, 256, 512, 512 and 4096 channels in each layer, while our algorithm uses only 1, 16, 16, 32, 32, 64, 64 and 512, respectively. See Table 1 for parameter information. We ultimately decided to not increase the size of the network to more closely resemble LS2Net due to the presence of overfitting (see loss function for more details). Furthermore, we did not include a Batch Normalization step between each Conv2D layer and the corresponding activation layer as the inclusion of this layer only resulted in a decreasing performance of the algorithm.

Even though Çalik et al. [1] use a LeakyReLU activation function for hidden layers, we chose the simpler ReLU activation function.

Table 1. Parameter information for final network. h, w, # of channels in, # of channels out p: padding size s: stride size

Input: 288 x 720 x 1	
conv	5, 5, 1, 16 p:same s:2
maxpool	2, 2 p:1 s:2
conv	3, 3, 16, 32 p:same s:2
conv	3, 3, 32, 32 p:same s:1
maxpool	3,3 p:1 s:2
conv	3, 3, 32, 64 p:same s:2
conv	3, 3, 64, 64 p:same s:1
maxpool	3, 3 p:1 s:2
conv	7, 7, 64, 512 p:same s:1
dense	30720
label	41, activation: softmax

4.2.1 Class Scoring

Softmax is the final layer of the neural network. This layer computes the class score or 'y' vector which will then be fed into loss function or outputted during testing. In our model the 'y' vector is a one hot 1 x 41 vector. There will be a '1' in one of the first 40 positions if the signature corresponds to one of the 40 real signatures or there will be a '1' in the 41st position if the signature corresponds to any of the forgeries.

4.2.2 Loss function

Both the original LS2Net algorithm and the simplified version presented in this paper use a cross entropy loss based on the softmax activation function. One change we made to the loss function was the addition of a L2-regularization term to address overfitting. When the L2-regularization was added the test set accuracy increased suggesting this addition does in fact decrease overfitting and improve the algorithm. However, the addition of this L2-regularization component did not eliminate overfitting completely, as will be discussed in the results section of this report.

5. Experiments/Results/Discussion

After trialing many different hyperparameters, we selected the hyperparameters listed in Table 2 for our final algorithm since they led to the best performance.

Table 2. Hyperparameters for final algorithm.

Hyperparameter list	
Learning rate	3×10^{-4}
Minibatch size	5
Weight initialization	Truncated Normal ($\sigma = 0.1$)
Optimizer	Adam
Number of epochs	51
Regularization factor (λ)	0.0011

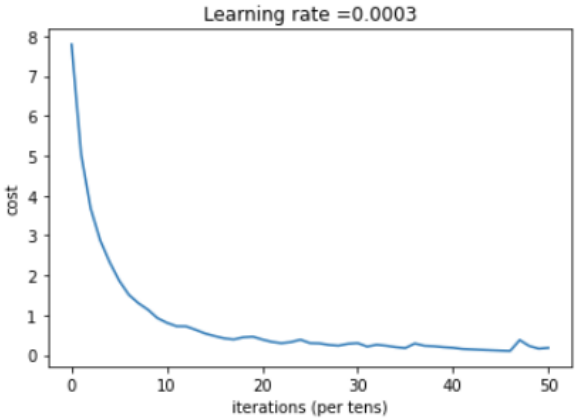
With these hyperparameters, the algorithm achieved accuracies of ~99% and 54% in the training and test sets, respectively. The cross-entropy loss showed a generally decreasing pattern when plotted against the epoch number, as depicted in Figure 4. The evolution is not monotonously decreasing, as mini batches were used to make the convergence faster.

From our work, we have an operating convolutional network that can successfully recognize signatures from 40 different individuals in 54% of the cases. Although the accuracy is not satisfactory for a real application, we attained a result that is significantly better than a random draw (which would have an accuracy of 2.5%). Based on these results, we consider our model to have learned a fair amount from the data provided.

The low accuracy obtained in the training set relative to that of the test set is an evidence of overfitting. This result motivated us to try a few alternatives to reduce the gap between accuracies of training and test sets. These

measures include:

- Optimizing the regularization term in the loss function, which did help to slightly improve the accuracy of the algorithm. We reached a point, however, where any increase of the hyperparameter lambda led to lower training and test set accuracies, and a decrease of it led to an increase in the gap between both accuracies.
- Modifying the size of the network. Increasing the size of the network by expanding the number of neurons in the hidden layers only increased overfitting. On the other hand, decreasing the number of neurons in hidden layers did reduce overfitting, but at the cost of reducing the accuracy of the whole algorithm. We then chose to maintain the network structure.
- Stopping early. We also thought of stopping the training earlier (by epoch ~25, for example), to try to prevent the model from overfitting. However, this did not have the desired effect and the accuracy of both the training and test sets decreased when attempting this.




```
Train Accuracy: 0.99814814
Test Accuracy: 0.53518516
```

Figure 4. Final results of algorithm.

6. Conclusions

In this work, we built an operational convolutional network similar to the network developed by Çalik et al. with the following main modifications: smaller network, simpler pre-processing step, removed Batch Normalization, ReLU vs LeakyReLU activation function. Our algorithm achieved accuracies of ~99% and 54% in the training and test sets, respectively, when using a 75-25 split on the 2,160 images. These results are consistent with overfitting and therefore we explored regularization, modifying network size, and stropping early to improve the model. Of these approaches we found only regularization helped reduce overfitting without compromising accuracy of the whole algorithm. Although the testing accuracy is not satisfactory for real signature recognition application, we consider our model to have learned a fair amount from the data provided and performs significantly better than a random draw.

As a future work, a further exploration of the overfitting challenge identified would be key to improving the training accuracy of the CNN network. Additionally, additional modifications to the algorithm such as adding Batch Normalization or additional pre-processing steps could be explored, to investigate the impact on the model performance. Finally, memory limitations should be addressed to incorporate more training data to the model.

7. Contributions

Our team consisted of two individuals, Diego Kaulen and Kaitlyn Baab. Both of us worked together to determine project topic and overall project approach. We also collaborated very closely on code implementation. When we found it was more efficient to break up parts of the project, Kaitlyn focused on paper writing while Diego focused on implementing edits to code. This phase consisted of frequent back-and-forth conversation to make sure we were aligned on the approach. Overall, we worked very closely together for the entirety of the project.

References

[1] N. Çalik, O. C. Kurban, A. R. Yilmaz, T. Yildirim, and L. Durak Ata, “Large-scale offline signature recognition via deep neural networks and feature embedding,” Neurocomputing, vol. 359, pp. 1–14, Sep. 2019, doi: 10.1016/j.neucom.2019.03.027.

[2] J. Fierrez, J. Ortega-Garcia, D. Ramos, and J. Gonzalez-Rodriguez, “HMM-based on-line signature verification: Feature extraction and signature modeling,” Pattern Recognition Letters, vol. 28, no. 16, pp. 2325–2334, Dec. 2007, doi: 10.1016/j.patrec.2007.07.012.

[3] L. Nanni and A. Lumini, “Advanced methods for two-class problem formulation for on-line signature verification,” Neurocomputing, vol. 69, no. 7–9, pp. 854–857, Mar. 2006, doi: 10.1016/j.neucom.2005.08.007.

[4] J. Poddar, V. Parikh, and S. K. Bharti, “Offline Signature Recognition and Forgery Detection using Deep Learning,” Procedia Computer Science, vol. 170, pp. 610–617, 2020, doi: 10.1016/j.procs.2020.03.133.

[5] 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.

[6] M.A. Ferrer, J.F. Vargas, A. Morales, A. Ordonez, Robustness of offline signature verification based on gray level features, IEEE Trans. Inf. Forens. Secur., vol. 7, no. 3, pp. 966-977, 2012, doi: 10.1109/TIFS.2012.2190281.