Online Handwritten Signature Verification using Deep Learning

Wittmajer Dávid, GitHub: wittmajerd, email: wittmajerd@edu.bme.hu

Consultant: Szücs Cintia Lia

INTRODUCTION

There is a lot of methods for online handwritten signature verification and most of them can show quite good result. One of them is to extract features from the signatures and compare them based on these. This paper presents a method using deep learning for the comparison and the system achieved 93% test accuracy. I extracted 26 features from the signatures and used a fully connected neural network to classify a signature as genuine or forged with a genuine signature as a reference.

DATA

From 870 users, I used 34700 signatures from the DeepSignDB dataset. I calculated a feature vector for each signature and then created difference vector from a user's genuine signature and another signature from the same user. This difference vector is the input for the model and the output is 0 if the tested signature is genuine and 1 if it is forged [2].

For the difference vector calculation there is a few methods I tried. First there is the simple subtraction, which gave a good result. Next the absolute and squared difference performed the same or little worse. The best result came from the relative difference calculation, which doesn't work by default because there are zero values in the feature vectors and this method requires a division, so I added a small epsilon to the divider. Here is the formula for it where v_g is the genuine basis of the comparison and v_t is the tested signature's vector and *eps* is a very small number.

$$V = \frac{v_g - v_t}{\frac{v_g + v_t}{2} + eps} \qquad (1)$$

This relative difference vector doesn't even need to be scaled, because is worked the same or even better without standard scaling.

In the DeepSignDB dataset there are different databases so the number of signatures per user varies. Some has 50 signatures, and some has 28. So, I shuffled the users and then used 700 users for training, 100 for validation and 70 for testing. With this difference vector calculation method, I got 124880 difference vectors.

FEATURES

I implemented the 26 following features from article [1]:

Duration: The time taken to perform the signature, in seconds.

Width: The horizontal distance of the signature.

Height: The distance between the lowest and the highest point in a signature.

Aspect ratio: This is the ratio of the writing length to the writing height.

Pen-down ratio: This is the ratio of the pen-down time to the total writing time. Calculation is performed by taking the ratio of the number of non-zero points to the total number of points.

Number of pen-ups: The number of times the pen is lifted while signing after the first contact with the tablet and excluding the final pen-lift.

Component physical spacing: Calculation involves taking the Euclidean distance between the last point sampled in a component and the first point sampled in the following component. This value is calculated for each pen-up instance and averaged to obtain the final feature value.

Component time spacing: This is the average duration of a pen-up instances in a signature.

Pressures: Mean, standard deviation, minimum and maximum of the pressure values.

Velocities: Mean, standard deviation and maximum of the velocities.

Horizontal velocity: This is the average velocity over the X direction. It measures how fast the signature moves horizontally. This feature is calculated as the ratio of signature length to the duration of the signature.

Accelerations: Max, Mean and standard deviation of acceleration.

Number of strokes: This feature is indicative of how many segments or states the handwriting goes through during the signature's production.

Curvature: This is a measure of how "flat" or how "curved" the handwriting is. Curvature is calculated as the ratio of the signature path length to the word length. The path length is the sum of distances between each consecutive point in the sample (large number in the order of 10,000 pixels). The word length is the physical, or Euclidean, distance between the captured writing's first and last point.

Average curvature per stroke: The curvature value is calculated for each individual stroke in the handwriting sample, then averaged.

Cursivity: The ratio of the number of strokes to the number of pen-downs.

Cursiveness: The ratio of the horizontal length of the handwriting to the number of pen-downs.

We can also perform some preprocessing on the signatures before the feature extraction. There are two methods mentioned in the paper [1] that could be useful, but I didn't try them. One is to restore the rotation of the signatures and the other is to remove small points and lines (like the dot of the "i" letter or the cross of the "t" letter).

Also, there are pen-ups in the signatures and for some features we can delete these points (where the pressure is zero) and maybe get better results.

Model

I used a fully connected neural network. The input layer has 26 neurons, one for each feature and the output layer has 1 neuron with sigmoid activation, which outputs the probability of the signature being a forgery. The activation function in the hidden layer(s) is ReLU, which performed a little better than sigmoid. I chose Adam as optimizer because I achieved better results with it. SGD only worked well in half of the test cases and even then, it gave results a bit worse than Adam.

I used binary cross entropy as the loss function and accuracy as the metric.

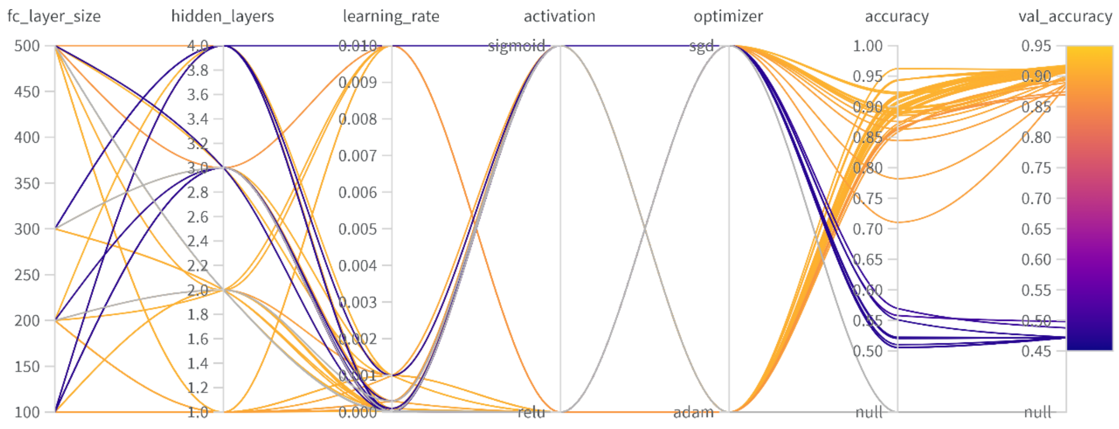

Figure 1: Hyperparameter optimalization with wandb sweeps. As the graph shows ReLU and Adam is most of the time better than their counterparts and the size of the network do not change the result much.

The layer size and the number of neurons in the hidden layers aren't that important, I tried different configurations, and they all gave similar results. Basically, it works with a few hidden layers with a few hundred neurons in each layer. I think it depends on how much data is there to train the model on. My model has a single 500 neuron hidden layer to keep it simple. I used a dropout layer after each hidden layer to prevent overfitting with a value of 0,4. I used early stopping with model checkpointing to save the best model. The validation set is to determine when to stop training and to save the best model. Early stopping is not necessary because the model reaches the best validation accuracy within a few epochs. And the test set is for the evaluation of the model.

I also tried a model which compares the tested signature to four different genuine ones. Because of this I got quarter of the training data of the single comparison method, but still, it gives a little bit better result. The best test accuracy I achieved with this model is 93,24% which is 2% higher than the single comparison one. This route has potential, but I didn't have time to experiment with it enough.

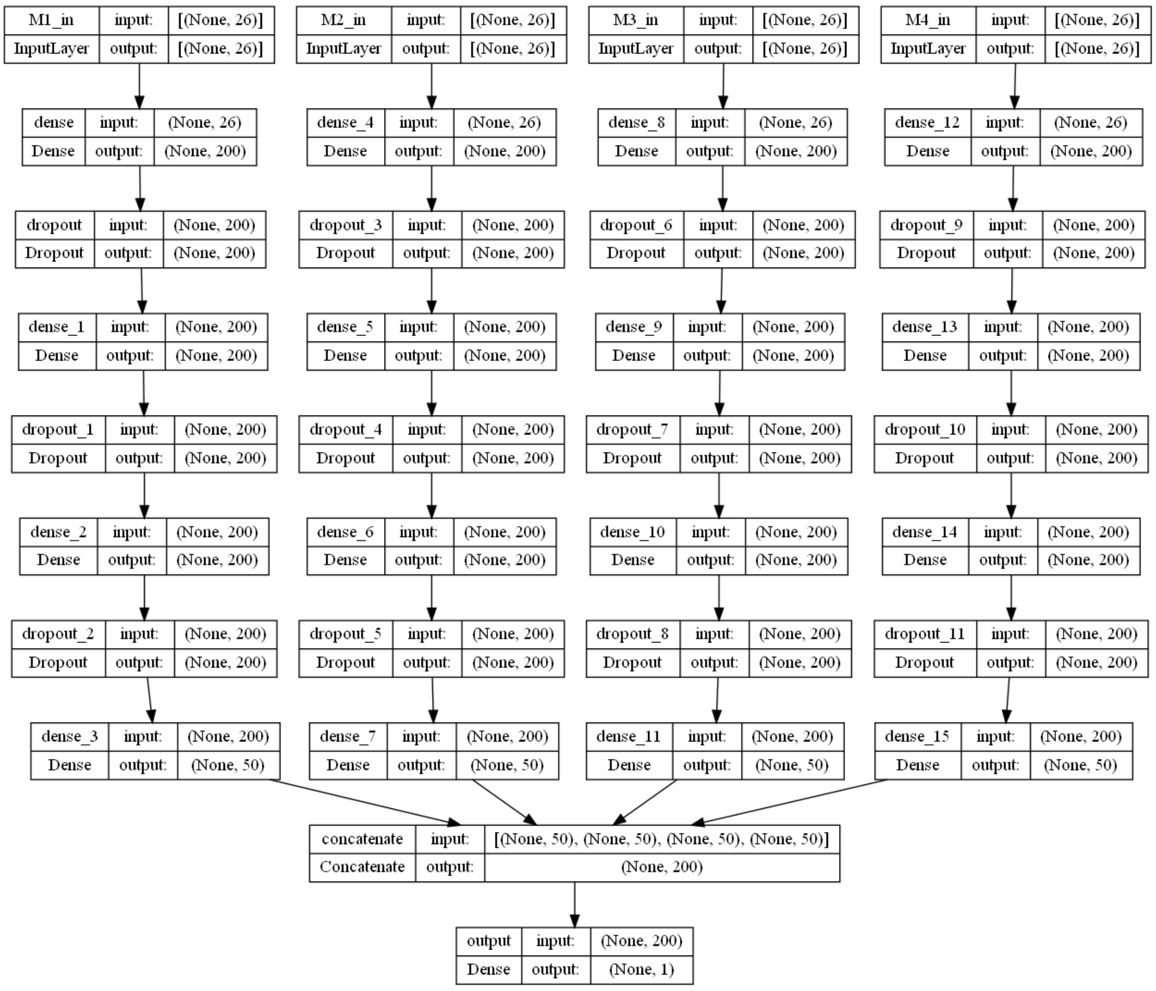
M1_in input: [(None, 26)]
InputLayer output: [(None, 26)]
M2_in input: [(None, 26)]
InputLayer output: [(None, 26)]
M3_in input: [(None, 26)]
InputLayer output: [(None, 26)]
M4_in input: [(None, 26)]
InputLayer output: [(None, 26)]
dense input: (None, 26)
Dense output: (None, 200)
dense_4 input: (None, 26)
Dense output: (None, 200)
dense_8 input: (None, 26)
Dense output: (None, 200)
dense_12 input: (None, 26)
Dense output: (None, 200)
dropout input: (None, 200)
Dropout output: (None, 200)
dropout_3 input: (None, 200)
Dropout output: (None, 200)
dropout_6 input: (None, 200)
Dropout output: (None, 200)
dropout_9 input: (None, 200)
Dropout output: (None, 200)
dense_1 input: (None, 200)
Dense output: (None, 200)
dense_5 input: (None, 200)
Dense output: (None, 200)
dense_9 input: (None, 200)
Dense output: (None, 200)
dense_13 input: (None, 200)
Dense output: (None, 200)
dropout_1 input: (None, 200)
Dropout output: (None, 200)
dropout_4 input: (None, 200)
Dropout output: (None, 200)
dropout_7 input: (None, 200)
Dropout output: (None, 200)
dropout_10 input: (None, 200)
Dropout output: (None, 200)
dense_2 input: (None, 200)
Dense output: (None, 200)
dense_6 input: (None, 200)
Dense output: (None, 200)
dense_10 input: (None, 200)
Dense output: (None, 200)
dense_14 input: (None, 200)
Dense output: (None, 200)
dropout_2 input: (None, 200)
Dropout output: (None, 200)
dropout_5 input: (None, 200)
Dropout output: (None, 200)
dropout_8 input: (None, 200)
Dropout output: (None, 200)
dropout_11 input: (None, 200)
Dropout output: (None, 200)
dense_3 input: (None, 200)
Dense output: (None, 50)
dense_7 input: (None, 200)
Dense output: (None, 50)
dense_11 input: (None, 200)
Dense output: (None, 50)
dense_15 input: (None, 200)
Dense output: (None, 50)
concatenate input: [(None, 50), (None, 50), (None, 50), (None, 50)]
Concatenate output: (None, 200)
output input: (None, 200)
Dense output: (None, 1)

Figure 2: The architecture of the more complex model's

Conclusion

The best test accuracy the single comparison model achieved is 93,78% and the results are not bad considering the feature extraction and the prediction is very inexpensive, requiring little computation power and time. To get better performance with this method there is a few possibilities. The model needs more or features better quality features. There are quite a few that could be implemented but I didn't have time to do it. Also, an autoencoder can be used for feature extraction to get much more and maybe better features.

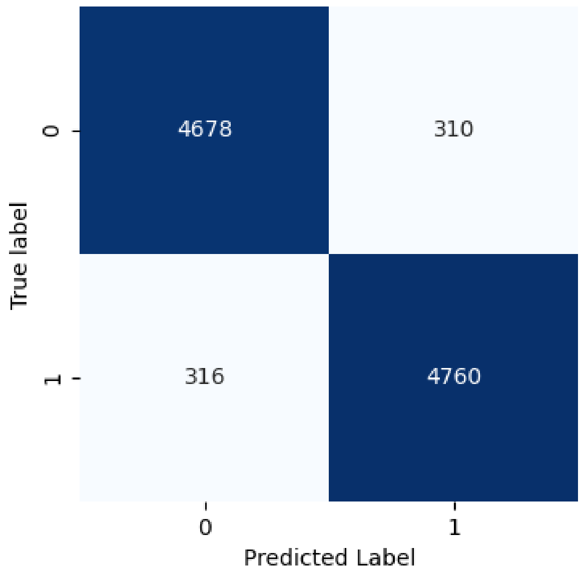

Figure 3: The confusion matrix of the best result

Sources

[1] A. McCabe, J. Trevathan and W. Read, (2008) "Neural Network-based Handwritten Signature Verification", https://www.researchgate.net/publication/235993403_Neural_Network-based_Handwritten_Signature_Verification

[2] P. Drobintsev, D. Sergeev and N. Voinov (2017) "Online signature verification system for mobile devices with multilevel pressure sensor", https://ieeexplore.ieee.org/abstract/document/7970606