Comparative Analysis of CNN Models for Signature Verification

Aman Chopra[1][0000−0001−6650−4445], Aakash Bhattacharya[2][0000−0001−6671−8128], and Rob Fergus[3]

[1] Computer Science Department, New York University, New York, NY, USA
ac8511@nyu.edu
[2] Computer Science Department, New York University, New York, NY, USA
ab9541@nyu.edu
[3] Computer Science Department, New York University, New York, NY, USA
fergus@cs.nyu.edu

Abstract. Signatures validate the authenticity of an individual. Hence, it becomes extremely important to distinguish a forged signature from a valid signature. Even the signature signed by the same original individual can vary sometimes and hence, it becomes a challenging task to identify the genuine signature. We believe that due to the advancements in image processing, features from the original signatures can be extracted that shows similarities between different original signatures. We have employed different data pre-processing techniques and build and compared Signature Network and our Custom CNN model which can bifurcate original and forged signatures with high accuracy. We have used the CEDAR dataset for our experiments. We have also evaluated the performance of certain models like the Signature Network with different hyper-parameters and have compared the performance, pros, and cons of the Signature Net with our custom CNN Model.

Keywords: Signature Verification · SIGCOMP · UTSIG · CEDAR Signature Dataset · Signature Net · Siamese Networks · Convolutional Neural Network · Receiver operating characteristic.

1 Introduction

Signature is the seal of authenticity for most of the critical transactions like bank transactions. It is one of the prime tool for security. There are two types of signature verification systems i.e., the online signature verification system and the offline. In this paper, we have tried to classify the original off-line signatures from the forged one. We can consider signatures as artistic hand writings which involves flow and movement. Hence, we can consider it as an image. Clearly, we can employ Computer Vision techniques and Convolutional Neural Networks to extract features from the signature like pressure and speed which can help reason the authenticity of the signature.

A lot of features like geometry, topology, variation, and other statistical features can be extracted from the image which can help us solve this problem

efficiently. There are a lot of key researches being already done in this area which have used robust pre-trained models to tackle this problem. In our case, we have designed our own Convolutional Neural Network model which comparatively takes less training time and has shown good results with the CEDAR Signature Dataset. We have also compared it with The SigNet model which uses the Siamese Networks.

The paper has been laid out in the following manner. Section II, the objective section deals with the primary objectives of this paper. Section III of this paper talks about the key researches being already done in this field. Section IV lays down the methodology we have followed to solve this particular problem. The methodology lays the dataset we have chosen, the pre-processing techniques employed, and the details of the hyper-parameters and the architecture of the models used. Sections V and VI mention the results and the conclusion.

2 Objective

In this paper, we have compared and analyzed the models to classify the original offline signatures from the forged one. The prime objective is to compare the accuracy, pros, and cons of the SigNet model with different hyper-parameters with our custom model and suggest a novel one to classify the original offline signature from a forged one. We have experimented with the architecture of our custom model and the hyper-parameters of the SigNet model following the Siamese Network to attain a high accuracy.

3 Related work

There have been studies related to offline signature verification systems in the recent past. In work [7], the authors have explored the different methods for two types of signature verification. The first involving the person independent one where from a set of signatures, the algorithm need to classify if its a genuine or a forged signature and the second one, which involves finding if a particular signature is signed by a particular genuine person. They have used a combination of the Kolmogorov-Smirnov and Kullback-Leibler measure, denoted by KLKS to find the accuracy of their model. In work [2], the authors have explored the power of transfer learning on the pretrained CNN models like VGG16, VGG19, InceptionV3 for offline signature verification system. The have used Support Vector Machine for binary classification. They have received good accuracy on the pre-trained models but the training time is very high. In another research [5], the authors have presented an off-line signature verification system based on moment invariant method and Artificial Neural Networks. The dataset chosen is small and not very varied and they have achieved a decent accuracy.

4 Methodology

Figure 1 shows the work flow chart. It shows the dataset we have chosen, the data pre-processing techniques employed, and the networks we have used.

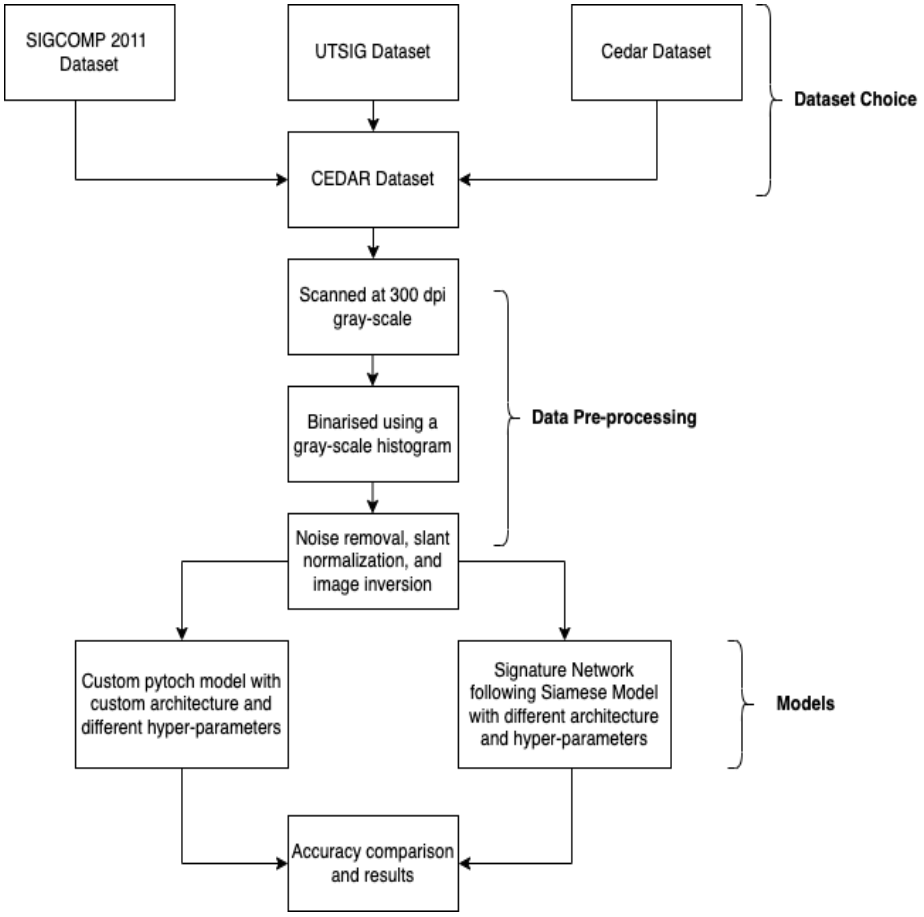

Fig. 1. Research methodology flowchart

4.1 Dataset:

The choice of dataset is extremely important when developing the architectures for classification and regression tasks. Certain algorithms and architecture perform fast and give a higher accuracy when run against certain kinds of data. Certain types of architectures are specialised to work well against a certain kind

of dataset. We have explored three widely used dataset for Signature Verification, the details of which are mentioned below.

SIGCOMP 2011 Dataset: This is a highly used dataset for online and offline signature verification competitions [4]. The offline dataset contains 400 dpi RGB images. It contains 1932 offline dutch signatures and 575 offline Chinese Signatures. The directory structure of this dataset containing both offline and online dataset from different countries makes it difficult to load via the data loader. Besides, there is no prior pre-processing done in the dataset which puts an added pre-processing overhead.

UTSig Dataset: UTSig [6] is a Persian offline signature dataset. It contains 8280 images from 115 classes. Each class has 27 genuine signatures, 3 opposite-hand signatures, and 42 skilled forgeries made by 6 forgers. The dataset is pre-processed, but the division of the dataset in many classes and subclasses makes it difficult to load via the data loader. Hence, we have tried the CEDAR Dataset.

Cedar Dataset: A good number of recent researches have been using the CEDAR dataset because of a well defined directory structure and pre-processed signature. A total of 55 individuals contributed 24 signatures each making a total of 1320 original signatures. Similarly, 1320 forged signatures were also created. The dataset is already pre-processed using gray-scale histogram and noise removal.

4.2 Data pre-processing:

Machine learning data sets usually provide us with copious amounts of data scraped off the Internet or noted down as a result of experimentation. This data needs to be restructured and processed according to the needs of an individual case study. In our attempt to suggest a classification model for bifurcating an offline original signature from that of a forged one, we have taken the following data pre-processing steps:

Grayscale and Normalisation: First the images are converted into gray scale using gray-scale histogram as the images are originally coloured. The smoothing of images is very important as images are often corrupted due to positive and negative impulses because of the noisy channels. Each image was scanned at 300 dpi. Signature normalisation was done to make the height and width of the images consistent. The gray-scale image is then converted to a bitmap. The image was then inverted before feeding to the model.

The data set was then shuffled with a random seed value and was divided into two sets of training and test data. 80% of the data was used for training and the rest was used for testing the performance of the models.

4.3 Models

```
Net(
  (layers): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=133760, out_features=512, bias=True)
    (2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU(inplace=True)
    (4): Dropout(p=0.5, inplace=False)
    (5): Linear(in_features=512, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): ReLU(inplace=True)
    (8): Dropout(p=0.5, inplace=False)
    (9): Linear(in_features=512, out_features=128, bias=True)
  )
)
```

Fig. 2. Details of Custom Convolutional Neural Network Model

Custom Convolutional Neural Network Model: The final model that has been used has 4 Conv2d layers of the input sizes shown in the Figure 2. Batch normalization has been used to normalize the output from every layer. It essentially offsets the "internal covariate shift" of the layers in the network. ReLU has been computed for the outputs of the BatchNorm which then acts as the input for the next layer. Pooling has been incorporated after the second and the fourth convolutional layers to down sample the feature maps so that the network is oblivious to the positions of the features in the images. Technically, it offsets the "internal translation invariance". Subsequently, the network converges to classify the images to the defined 2 binary classes with the help of 3 linear layers used in combination with BatchNorm, Dropout and ReLU.

Optimizing Algorithm and Learning rate: An Adam optimizer has been used instead of SGD optimizer. Adam optimizer uses an adaptive learning rate which essentially speeds up the learning process. The learning rate has not been fixed to a static value. Instead, exponential decay has been used to start of with a higher learning rate and then reduces the rate as and when the network converges so that the loss is optimally minimized. This model has been trained with 10 epochs which resulted in an accuracy of 0.9638.
Figure 3 shows the architecture of our Custom Convolutional Neural Network Model.

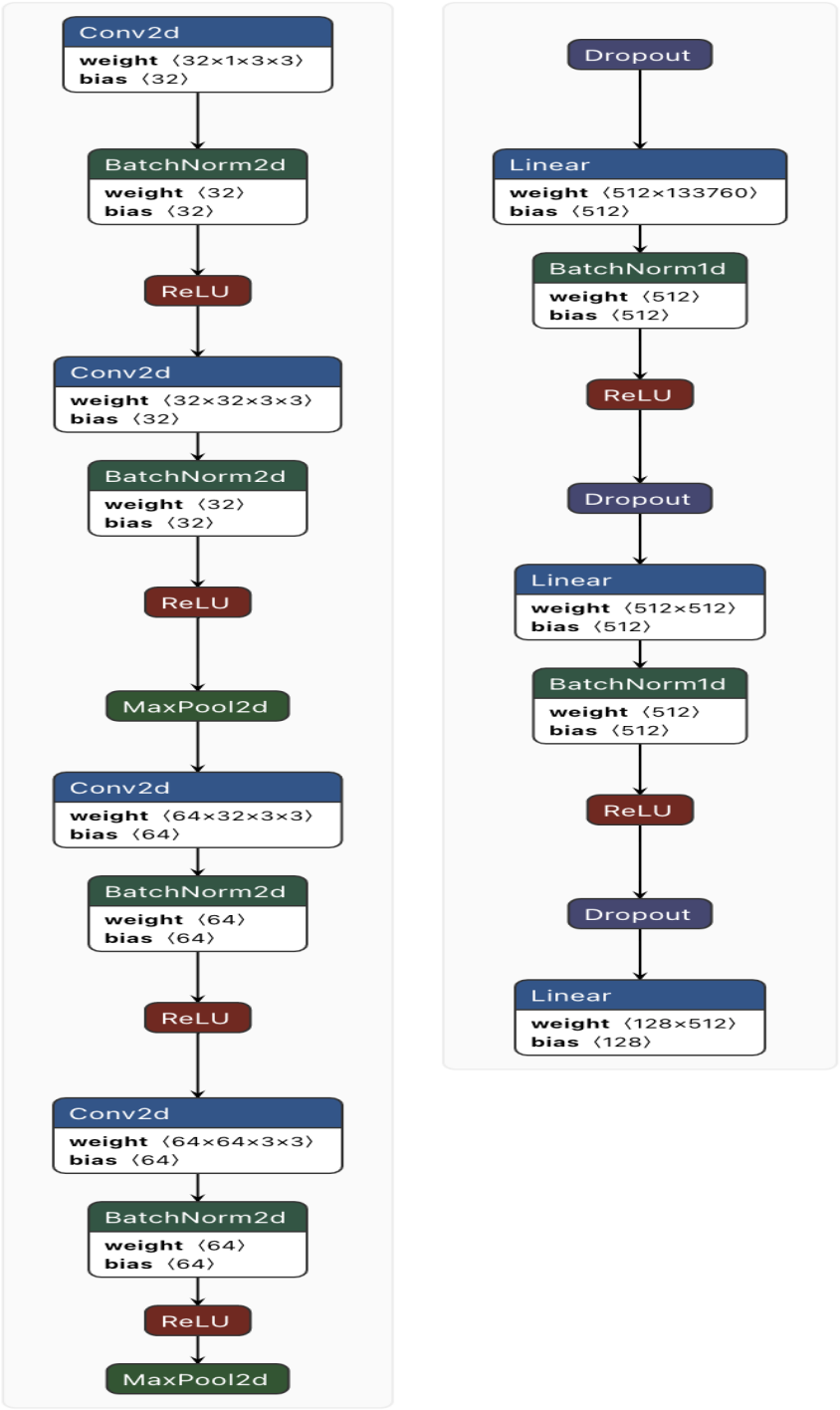

Fig. 3. Custom Convolutional Neural Network Model

```
SigNet(
  (conv1): Conv2d(1, 48, kernel_size=(11, 11), stride=(1, 1))
  (lrn1): LocalResponseNorm(48, alpha=0.0001, beta=0.75, k=2)
  (pool1): MaxPool2d(kernel_size=(3, 3), stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(48, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (lrn2): LocalResponseNorm(128, alpha=0.0001, beta=0.75, k=2)
  (pool2): MaxPool2d(kernel_size=(3, 3), stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout1): Dropout(p=0.3, inplace=False)
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(256, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool3): MaxPool2d(kernel_size=(3, 3), stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout2): Dropout(p=0.3, inplace=False)
  (fc1): Linear(in_features=40800, out_features=1024, bias=True)
  (dropout3): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=1024, out_features=128, bias=True)
)
```

Fig. 4. Details of Signature Network

Signature Network: The final model that has been used is shown in the Figure 4. We have followed the ideology of Siamese Networks here [3]. Siamese Networks is a class of Neural Networks. The ideology is that it contains two or more identical subnetworks which mirrors the parameter updates. It is used to find the similarity of the inputs by comparing its feature vectors, so these networks are used in many verification applications. Siamese networks are more robust to class imbalance, it can also learn Semantic similarity.

The cons of using a Siamese Network is that it does not output probability and takes a lot of time to train. As Siamese Networks are generally used in verification systems, we have made use of Siamese Networks for this problem statement.

We have used this network because it works well with new data. The ROC accuracy achieved can be improved with more number of epochs but we have trained the network for 5 epochs. We have made use of the Stochastic Gradient Descent with an exponential decay learning rate which has given us the maximum accuracy.

4.4 Loss functions:

Cross entropy: In our Custom Convolutional Neural Network Model, we have made use of the binary cross-entropy loss. We have trained the model on the 80% of the dataset and have tested it on the remaining 20%. We have tested the accuracy based on the number of unseen signatures that our model has successfully correctly classified. Figure 5 shows the decay of loss with increasing number of iterations in the case of our Custom Convolutional Neural Network.

Contrastive Loss function: In our signature Network, we have made use of the Contrastive Loss Function. We have made use of the Receiver Operating Characteristic (ROC) curve to calculate the accuracy in this case. ROC tries all of the cutpoint and plots the sensitivity and specificity. Figure 6 shows the decay of loss with increasing number of iterations in the case of Signature Net.

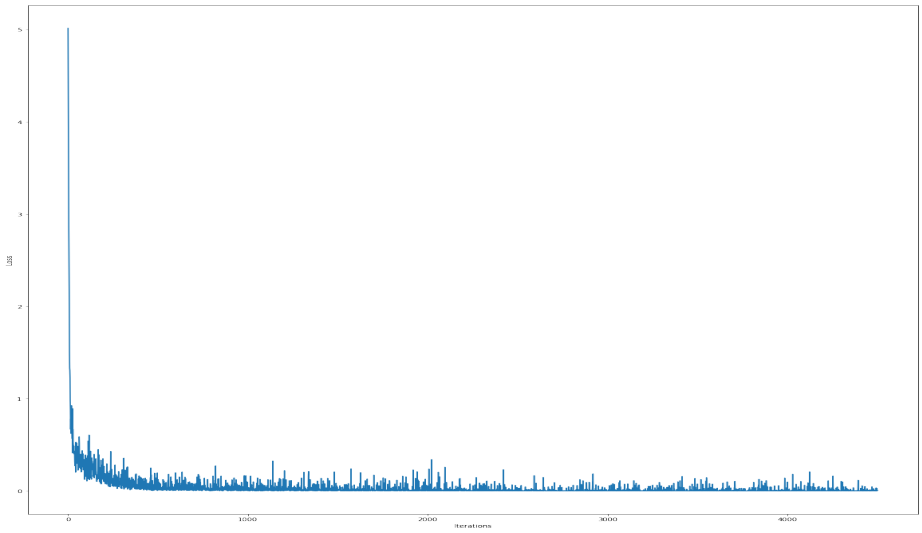

Fig. 5. Loss vs iterations of Custom CNN Model

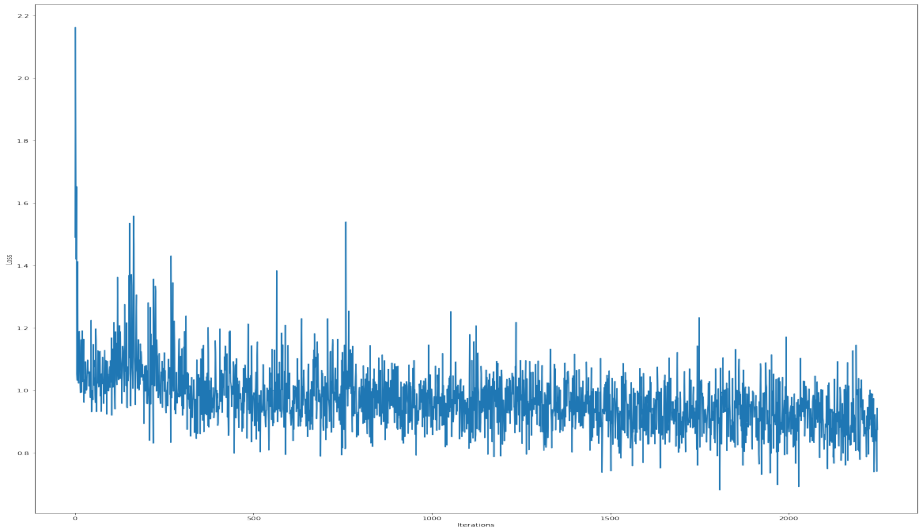

Fig. 6. Loss vs iterations of Signature Network

5 Results

We have compared the accuracy of our custom model with the Signature Network. For our custom model, we have calculated the overall accuracy based on the number of correct predictions out of the total signatures in the test dataset. For signature Network, we have used Receiver Operating Characteristic (ROC) curve. The ROC curve is different from the overall accuracy. Overall accuracy is based on one specific cutpoint, while ROC tries all of the cutpoint and plots the sensitivity and specificity. We have also experimented with different hyper-parameters like optimiser and the learning rates with and without exponential decay. Table 1 shows the final result. We can see that the custom model gives the accuracy of 96.38%. We have used Adam Optimiser with exponential decay learning to get the best accuracy. For the Signature Network, the best ROC accuracy of 82.74% (ROC) is achieved with the Stochastic Gradient Descent Optimiser with an exponential decay learning rate.

Table 1. Results

Accuracy		
Hyper-parameters	**Custom Model**	**Signet Model (ROC)**
Adam Optimiser with exp. decay	96.38%	79.56%
SGD Optimiser with exp. decay	94.26%	82.74%
Adam Optimiser with lr = 0.01	95.58%	78.95%
Adam Optimiser with lr = 0.001	94.96%	79.27%
SGD Optimiser with lr = 0.01	93.27%	81.56%
SGD Optimiser with lr = 0.001	94.20%	81.75%

6 Conclusion and future work

We have analyzed the performance of two models to classify the original offline signature from a forged one. According to our analysis, our custom novel Convolutional Neural Network model performed better in CEDAR dataset but the SigNet model is more robust and can handle different datasets well.

In future, we intend to test our custom model with varied datasets to test the robustness and accuracy of the model. We also intend to fine tune the model by altering layers and hyper-parameters so that it becomes more accommodating to the newer datasets and can continue to perform well. We will also try different deep nets like AlexNet, VGG Neural Network, ResNet etc - which require more computing power. We also intend to extend this work to online signature verification where we can get the result as and when a signature is seen. This

approach can further be extrapolated to a wider array of new topics. We also want to see if the similar model with minor tweaks can be used for similar classification problems. This might bridge the gap between having to have different models for similar classification problems. The GitHub link to the repository containing our code is [1]

References

1. Aakash Bhattacharya, A.C.: Signature Verification. https://github.com/abbh07/Signature-Verification (2021)
2. Foroozandeh, A., Askari Hemmat, A., Rabbani, H.: Offline handwritten signature verification and recognition based on deep transfer learning. In: 2020 International Conference on Machine Vision and Image Processing (MVIP). pp. 1–7 (2020)
3. Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition (2015)
4. Marcus Liwicki, Michael Blumenstein, E.v.d.H.: Sigcomp11: Signature verification competition for on- and offline skilled forgeries. In: 11th Int. Conference on Document Analysis and Recognition (2011)
5. Oz, C.: Signature recognition and verification with artificial neural network using moment invariant method. vol. 3497, pp. 195–202 (05 2005)
6. Soleimani, A., Fouladi, K., Araabi, B.N.: Utsig: A persian offline signature dataset. IET Biometrics 6(1), 1–8 (Aug 2016), http://dx.doi.org/10.1049/iet-bmt.2015.0058
7. Srinivasan, H., Srihari, S.N., Beal, M.J.: Machine learning for signature verification. In: Kalra, P.K., Peleg, S. (eds.) Computer Vision, Graphics and Image Processing. pp. 761–775. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)