

Minor Project Report

On

**Convolutional Siamese Network for
Writer Independent Signature Verification**

Submitted by

ABHISHEK KUMAR

(202IT001)

I Sem M.Tech (IT)

MOHD ASIF KHAN KHAISHAGI

(202IT013)

I Sem M.Tech (IT)



DEPARTMENT OF INFORMATION TECHNOLOGY

National Institute of Technology Karnataka, Surathkal

Introduction

Signature verification is one of the most challenging tasks in biometrics. Here the minute details of signatures should be matching with the original writers signatures. Signatures are a very important part of our life and in a way define the identity of a person. Many places require one's signatures as a verifiable object of one's identity . Its examples are banks, government institutions where signatures are a mandatory part for performing transactions. So there is a need to verify the signatures since signatures can be copied. So a signature verification system is needed to verify the signatures such that it can identify one's signature minute features which can be used to verify it against some other signature.

We can use deep learning to identify the signatures. But if you make a classification model then it will be difficult since most classifiers learn data with respect to classes given to them. For example in signature verification problems we want to train the model independent of the writer i.e if a new writer comes we can identify his signatures also. This problem statement needs to be changed a bit instead of directly classifying the user class from signature. We need to change our problem to finding similarity of 2 images and from similarity verify whether the images are similar or not. Siamese networks are one such technique of converting classification problem to similarity problem.

The term Siamese means twins. The two Convolutional Neural Networks shown above are not different networks but are two copies of the same network, hence the name Siamese Networks. Basically they share the same parameters. The two input images (x_1 and x_2) are passed through the ConvNet to generate a fixed length feature vector for each ($h(x_1)$ and $h(x_2)$). Assuming the neural network model is trained properly, we can make the following hypothesis: If the two input images belong to the same character, then their feature vectors must also be similar, while if the two input images belong to the different characters, then their feature vectors will also be different. Thus the element-wise absolute difference between the two feature vectors must be very different in both the above cases. And hence the similarity score generated by the output sigmoid layer must also be different in these two cases. This is the central idea behind the Siamese Networks.

Literature:

OSVNet: Convolutional Siamese Network for Writer Independent Online Signature Verification

Computer Vision and Pattern Recognition - 2019

Online signature verification (OSV) is one of the most challenging tasks in writer identification and digital forensics. Owing to the large intra-individual variability, there is a critical requirement to accurately learn the intra-personal variations of the signature to achieve higher classification accuracy. To achieve this, in this paper, we propose an OSV framework based on a deep convolutional Siamese network (DCSN). DCSN automatically extracts robust feature descriptions based on metric-based loss function which decreases intra-writer variability (Genuine-Genuine) and increases inter-individual variability (Genuine-Forgery) and directs the DCSN for effective discriminative representation learning for online signatures and extend it for one shot learning framework. Comprehensive experimentation conducted on three widely accepted benchmark datasets MCYT-100 (DB1), MCYT-330 (DB2) and SVC-2004-Task2 demonstrate the capability of our framework to distinguish the genuine and forgery samples. Experimental results confirm the efficiency of deep convolutional Siamese network based OSV by achieving a lower error rate as compared to many recent and state-of-the art OSV techniques.

Offline Signature Verification on Real-World Documents -

April 2020

CVPR 2020 Biometrics Workshop

Research on offline signature verification has explored a large variety of methods on multiple signature datasets, which are collected under controlled conditions. However, these datasets may not fully reflect the characteristics of the signatures in some practical use cases. Real-world signatures extracted from the formal documents may contain different types of occlusions, for example, stamps, company seals, ruling lines, and signature boxes. Moreover, they may have very high intra-class variations, where even genuine signatures resemble forgeries. In this paper, we address a real-world writer independent offline signature verification problem, in which a bank's customers' transaction request documents that contain their occluded signatures are compared with their clean reference signatures. Our proposed method consists of two main components, a stamp cleaning method based on CycleGAN and signature representation based on CNNs. We extensively evaluate different verification setups, fine-tuning strategies, and signature representation approaches to have a thorough analysis of the problem. Moreover, we conduct a human evaluation to show the challenging nature of the problem. We run experiments both on our custom dataset, as well as on the publicly available Tobacco-800 dataset. The experimental results validate the difficulty of offline signature verification on real-world documents. However, by employing the stamp cleaning process, we improve the signature verification performance significantly

Dataset

BengaliHindiSignature (BHSig260) dataset

100 Bengali Signatures

160 Hindi Signatures

Here we worked on Hindi Signatures.

160 Signers

24 Genuine Signatures and 30 Forged Signatures

Total genuine to genuine pairs = 276

So we also took 276 genuine to forged pairs to balance the dataset.

For 1 signer we had around 552 images (both genuine to genuine and genuine to forged in equal amounts)

For training we took data of 100 Signers = 55200 Image combinations

For testing we took data of 60 Signers = 33120 Image combinations

METHODOLOGY

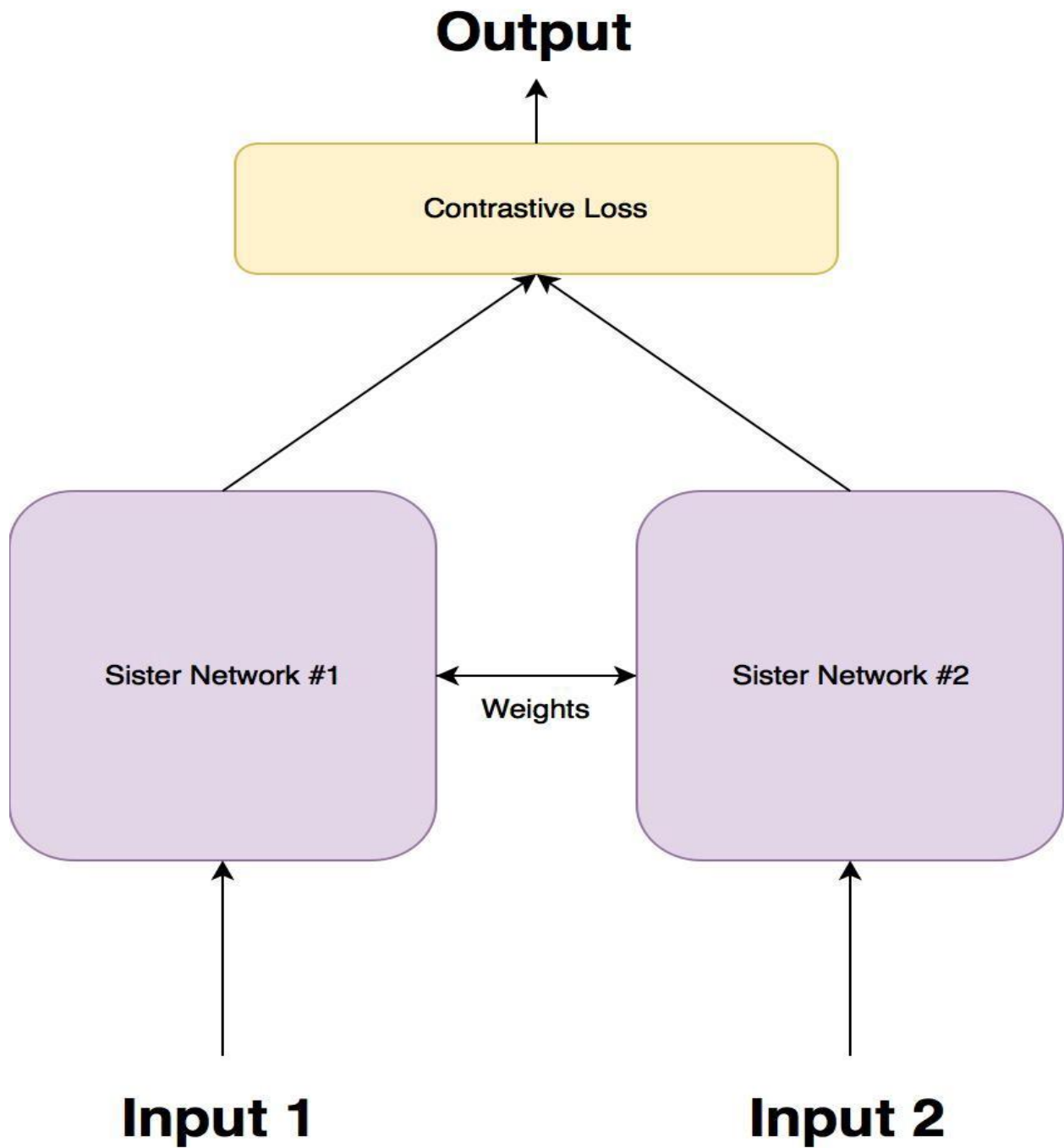
Preprocessing :

- Divide dataset into 100 for train and 60 for test
- For 24 genuine images make 276 genuine combination
- For each genuine image take 23 forged images
- Randomly generate pairs from genuine to forged images i.e. 276 forged combinations
- Make pairs of similar images
- Make pairs of non similar (forged images)
- Open in grayscale format.
- Resize image into 155 * 220 since there can be images of different sizes as input.

Model :

The term Siamese means twins. The two Convolutional Neural Networks shown above are not different networks but are two copies of the same network, hence the name Siamese Networks. Basically they share the same parameters. The two input images (x_1 and x_2) are passed through the ConvNet to generate a fixed length feature vector for each ($h(x_1)$ and $h(x_2)$). Assuming the neural network model is trained properly, we can make the following hypothesis: If the two input images belong to the same character, then their feature vectors must also be similar, while if the two input images belong to the different characters, then their feature vectors will also be different. Thus the element-wise absolute difference between the two feature vectors must be very different in both the above cases. And hence the similarity score generated by the output sigmoid

layer must also be different in these two cases. This is the central idea behind the Siamese Networks.



Reference for image

-<https://hackernoon.com/one-shot-learning-with-siamese-networks-in-pytorch-8ddaa>

Contrastive Loss :

- ▶ Distance Function $d(A,B) = ||f(A) - f(B)||^2 = D$
- ▶ Training : learn the parameter such that
 - if A and B depict the same person signature(genuine), $d(A,B)$ is small
 - if A and B depict the different person signature(forged), $d(A,B)$ is large
- ▶ Loss function for a positive pair:
 - ▶ If A and B depict the genuine pair , $d(A,B)$ is small
 - ▶ $L(A,B) = ||f(A) - f(B)||^2$
- ▶ Loss function for a negative pair:
 - ▶ If A and B depict the forged pair , $d(A,B)$ is large.
 - ▶ Better use a hinge loss:
 - ▶ $L(A,B) = \max(0, m - ||f(A) - f(B)||)^2$
- ▶ Contrastive Loss :

$$L(A,B) = y^* ||f(A)-f(B)||^2 + (1- y^*)\max(0,m - ||f(A) - f(B)||)^2$$

CNN Architecture:

Layer	Size	Parameters
Convolution	$96 \times 11 \times 11$	stride = 1
Local Response Norm.	-	$\alpha = 10^{-4}$, $\beta = 0.75$ $k = 2$, $n = 5$
Pooling	$96 \times 3 \times 3$	stride = 2
Convolution	$256 \times 5 \times 5$	stride = 1, pad = 2
Local Response Norm	-	$\alpha = 10^{-4}$, $\beta = 0.75$ $k = 2$, $n = 5$
Pooling + Dropout	$256 \times 3 \times 3$	stride = 2, $p = 0.3$
Convolution	$384 \times 3 \times 3$	stride = 1, pad = 1
Convolution	$256 \times 3 \times 3$	stride = 1, pad = 1
Pooling + Dropout	$256 \times 3 \times 3$	stride = 2, $p = 0.3$
Fully Connected + Dropout	1024	$p = 0.5$
Fully Connected	128	

We used this architecture to train the model on above dataset:

Hyperparameter Tuning:

Initial Learning Rate	1e-5	1e-4	1e-3	1e-6	1e-7	1e-8
Learning Rate Schedule	0.1	0.1	0.1	0.1	0.1	0.1
Weight Decay	9e-4	9e-4	9e-4	9e-4	9e-4	9e-4
Momentum	0.9	0.9	0.9	0.9	0.9	0.9
Fuzz Factor()	1e-8	1e-8	1e-8	1e-8	1e-8	1e-8
Batch Size	39	39	39	39	39	39
Accuracy	72%	52%	50%	71%	73%	60%

Initial Learning Rate	1e-5	1e-5	1e-5	1e-5	1e-5
Learning Rate Schedule	0.01	0.001	0.0001	0.01	0.01
Weight Decay	9e-4	9e-4	9e-4	5e-4	4e-4
Momentum	0.9	0.9	0.9	0.9	0.9

Fuzz Factor()	1e-8	1e-8	1e-8	1e-8	1e-8
Batch Size	39	39	39	39	39
Accuracy	72%	71%	70%	75%	71%

Initial Learning Rate	1e-5	1e-5	1e-5
Learning Rate Schedule	0.01	0.01	0.01
Weight Decay	5e-4	5e-4	5e-4
Momentum	0.8	0.9	0.9
Fuzz Factor()	1e-8	1e-7	1e-9
Batch Size	39	39	39
Accuracy	68%	73%	70%

With hyperparameter tuning on current model we could achieve max accuracy of 75%

Then we tried modifying the model

Experiment 1 : Changing size of layers

Layer	Size	Parameters
Convolution	$96 \times 11 \times 11$	stride = 1
Local Response Norm.	-	$\alpha = 10^{-4}$, $\beta = 0.75$ $k = 2$, $n = 5$
Pooling	$96 \times 3 \times 3$	stride = 2
Convolution	$128 \times 5 \times 5$	stride = 1, pad = 2

Local Response Norm	-	$\alpha = 10^{-4}$, $\beta = 0.75$ $k = 2$, $n = 5$
Pooling + Dropout	$128 \times 3 \times 3$	stride = 2, $p = 0.3$
Convolution	$384 \times 3 \times 3$	stride = 1, pad = 1
Convolution	$128 \times 3 \times 3$	stride = 1, pad = 1
Pooling + Dropout	$128 \times 3 \times 3$	stride = 2, $p = 0.3$
Fully Connected + Dropout	256	$p = 0.5$
Fully Connected	128	

With this architecture on the same hyperparameters we got 71% accuracy.

Experiment 2 :Making the model smaller

Layer	Size	Parameters
Convolution	$96 \times 11 \times 11$	stride = 1
Local Response Norm.	-	$\alpha = 10^{-4}$, $\beta = 0.75$ $k = 2$, $n = 5$
Pooling	$96 \times 3 \times 3$	stride = 2
Convolution	$128 \times 5 \times 5$	stride = 1, pad = 2
Local Response Norm	-	$\alpha = 10^{-4}$, $\beta = 0.75$ $k = 2$, $n = 5$
Pooling + Dropout	$128 \times 3 \times 3$	stride = 2, $p = 0.3$
Convolution	$256 \times 3 \times 3$	stride = 1, pad = 1
Convolution	$128 \times 3 \times 3$	stride = 1, pad = 1

Pooling + Dropout	$32 \times 3 \times 3$	stride = 2, p = 0.3
Fully Connected + Dropout	256	p = 0.5
Fully Connected	128	

With this architecture we got max accuracy 70%

Experiment 3 : Embedding size as 256

Layer	Size	Parameters
Convolution	$96 \times 11 \times 11$	stride = 1
Local Response Norm.	-	$\alpha = 10^{-4}$, $\beta = 0.75$ k = 2, n = 5
Pooling	$96 \times 3 \times 3$	stride = 2
Convolution	$256 \times 5 \times 5$	stride = 1, pad = 2
Local Response Norm	-	$\alpha = 10^{-4}$, $\beta = 0.75$ k = 2, n = 5
Pooling + Dropout	$256 \times 3 \times 3$	stride = 2, p = 0.3
Convolution	$256 \times 3 \times 3$	stride = 1, pad = 1
Convolution	$384 \times 3 \times 3$	stride = 1, pad = 1
Pooling + Dropout	$32 \times 3 \times 3$	stride = 2, p = 0.3
Fully Connected + Dropout	1024	p = 0.5
Fully Connected	256	

Here we got max accuracy as 75%

Matching our own signatures :

localhost:8888/notebooks/signature-verification-deploy/signature_verify.ipynb

Jupyter signature_verify Last Checkpoint: 12/20/2020 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code Voila

```
pred = "Genuine"
lbl_pred.value = "Dissimilarity {:.2f} Predicted is {}".format(distance.item(),pred)
btn_run.on_click(on_click_verify)
```

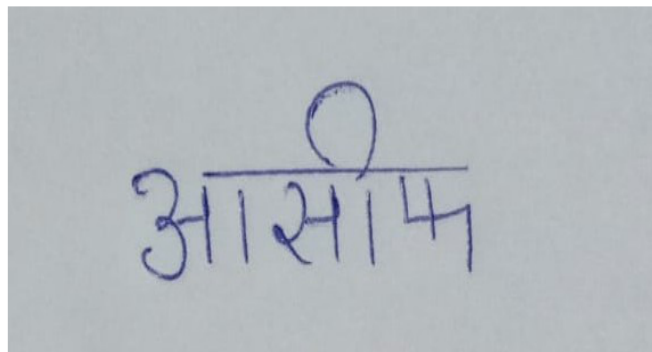
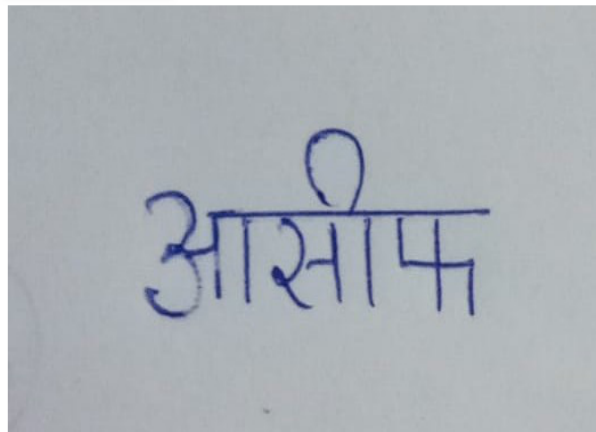
In [7]: widgets.VBox([widgets.Label("Select two signatures to verify!"),btn_upload,btn_upload_2,btn_verify])

Select two signatures to verify!

Upload (1)

Upload (1)

Verify



Dissimilarity 0.01 Predicted is Genuine

← → ↻ ⓘ localhost:8888/notebooks/signature-verification-deploy/signature_verify.ipynb

jupyter signature_verify Last Checkpoint: 12/20/2020 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

📁 + 🔍 📄 ⬆️ ⬆️ ▶ Run ⏏️ 🔁 ⏩ Code ▾ 📄 🗨 Voilà

```
x1, x2 = model(x1, x2)
distance = torch.pairwise_distance(x1, x2, p=2)
if distance > 0.05:
    pred = "Forged"
else:
    pred = "Genuine"
lbl_pred.value = "Dissimilarity {:.2f} Predicted is {}".format(distance, pred)
btn_run.on_click(on_click_verify)
```

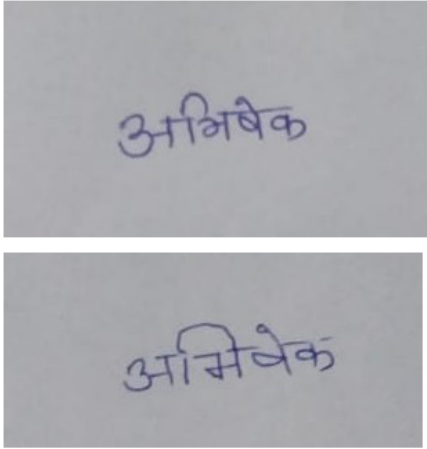
In [7]: widgets.VBox([widgets.Label("Select two signatures to verify!"),

Select two signatures to verify!

📁 Upload (4)

📁 Upload (4)

Verify



Dissimilarity 0.01 Predicted is Genuine

In []:

Case of genuine forged pair :

← → ↻ localhost:8888/notebooks/signature-verification-deploy/signature_verify.ipynb

jupyter signature_verify Last Checkpoint: 12/20/2020 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

📄 + 🔍 📄 📄 ⬆️ ⬆️ ▶️ Run 🛑 ↺ ▶️ Code 📄 🗨️ Voilà

```

x1, x2 = model(x1, x2)
distance = torch.pairwise_distance(x1, x2, p=2)
if distance > 0.05:
    pred = "Forged"
else:
    pred = "Genuine"
lbl_pred.value = "Dissimilarity {:.2f} Predicted is {}".format(distance, pred)
btn_run.on_click(on_click_verify)

```

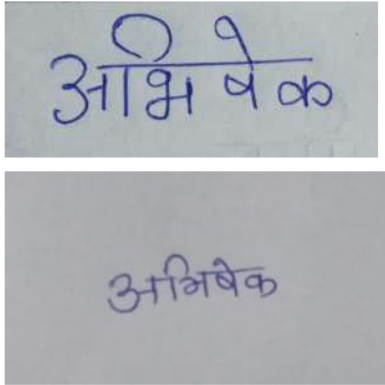
In [7]: widgets.VBox([widgets.Label("Select two signatures to verify!"),btn_upload,btn_verify])

Select two signatures to verify!

📄 Upload (3)

📄 Upload (3)

Verify



Dissimilarity 0.06 Predicted is Forged

In []:

Try it yourself : We deployed the model on binder so you can use it to verify signatures :

https://mybinder.org/v2/gh/asifk1997/SignatureVerification/HEAD?urlpath=%2Fvoila%2Frender%2Fsignature_verify.ipynb

Binder Demo :

Here we have deployed the Convolutional Siamese model to verify signatures. You can use this to match your signature and check whether this model is able to predict genuine or forged signatures.

By - MOHD ASIF KHAN AND ABHISHEK KUMAR

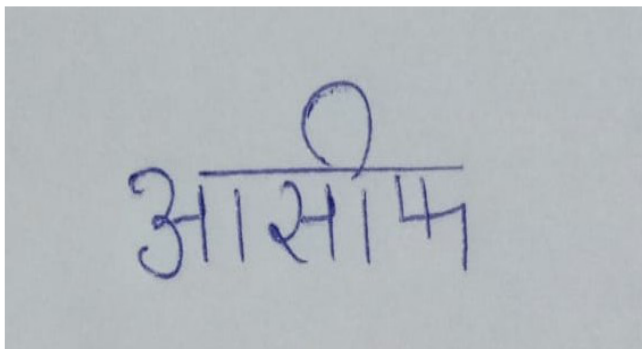
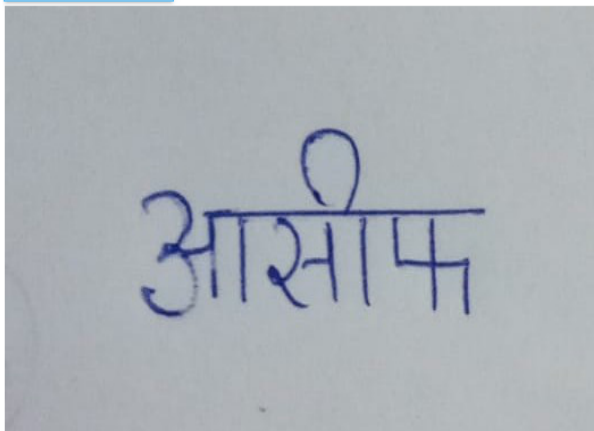
Device: cpu

Select two signatures to verify!

Upload (1)

Upload (1)

Verify



Dissimilarity 0.01 Predicted is Genuine

Conclusion and Future Work

Convolutional siamese networks are able to detect the minute details such as difference in signatures.

While Training We had following observations :

The model overfitting is a major problem since after some epochs the loss keeps oscillating between some value so we have to stop training if the model is learning nothing.

Accuracy increases in initial epochs then start degrading after some epochs.

We tried with multiple hyperparameters
Best hyperparameters were

Initial Learning Rate	1e-5
Learning Rate Schedule	0.01
Weight Decay	5e-4
Momentum	0.9
Fuzz Factor()	1e-8
Batch Size	39
Accuracy	75%

Here we could achieve 75% which is less than other state of the art accuracy 85.90% on Hindi Dataset.

We can use triplet loss, or take inspiration from other state of the art models to improve the accuracy further.

References :

- A. Dutta, U. Pal, J. Lladós, Compact correlated features for writer independent signature verification, in: ICPR, 2016, pp. 3411–3416.
- Sounak Dey, Anjan Dutta, Suman K. Ghosh, Josep Lados, Umapada Pal, Convolutional Siamese Network for Writer Independent Offline Signature Verification ,ELSEVIER
- Harshvardhan Gupta, Facial Similarity with Siamese Network, HackerNoon .
- Elad Hoffer, Nir Ailon Deep metric learning using Triplet network