A Comparative Study of Transfer Learning Models for Offline Signature Verification and Forgery Detection

Manikantha K[1], Aishwarya R Bhat[2], Pavani Nerella[3], Pooja Baburaj[4]* and Sharvari K S[5]

Department of Computer Science and Engineering, B.N.M Institute of Technology, Bengaluru
[1]*manikanthak@bnmit.in*, [2]*bhat.aishwarya99@gmail.com*,
[3]*pavanin862@gmail.com*, [4]*baburaj.pooja@gmail.com*,
[5]*sharvarisuresh9@gmail.com*

Abstract: *Recognising one's identity to enter a system is called authentication. This process can take various forms where users input the system with a set of identifying credentials to access the system. Signatures belong to behavioural biometric, where the distinct features of every individual are considered in order to corroborate the person's identity. The act of falsely imitating one's signature biometric to impersonate and leverage access to their assets is called signature forgery. Our paper presents a comparative study of various deep learning models using the Siamese architecture, over a wide catalogue of signature images. Openly available datasets like CEDAR, Handwritten Signatures dataset from Kaggle, ICDAR 2011 SigComp, and BH-Sig260 signature corpus are used to train the models. A set of classifiers – Support Vector Classifiers (SVC), Gaussian Naïve Bayes (GNB), Logistic Regression (LR) and K-Nearest Neighbours (KNN) are applied sequentially to classify the signature as genuine or forged.*

Keywords: Convolutional Neural Networks, Offline Signature Verification, Signature Forgery Detection, Deep Learning, Transfer Learning

1. Introduction

Signatures belong to behavioural biometric, where the distinct features of every individual are considered in order to corroborate the person's identity. The act of falsely imitating one's signature to impersonate and leverage access to their assets is called signature forgery. The forgers use different methods like tracing and optical transfer to fabricate false signatures. Random/blind forgery, unskilled forgery and skilled forgery are the different types of forgery encountered when trying to discern genuine signatures from fake ones.

Handwritten signatures play a vital role in our social and legal life. It is used for verification and authentication and when a person places a signature on a document, it implies his/her intent to agree with conditions or terms stated by that document. Thus, signature verification becomes a very important security aspect.

According to how the signatures are acquired, the verification is done in online and offline signature verification modes. In Online signature, the signer is made to sign on a digital device with the help of a stylus in real time to acquire the signature. The system captures dynamic features, such as position, velocity, pressure, etc. Offline Signature Verification is used when physically signed documents have to be inspected, and only a two-dimensional image is available for verifying the signature.

Offline signature verification can be addressed with two approaches – writer dependent approach where the system is updated every time a new signer is introduced and writer independent approach where a generic system is built to distinguish between genuine and forged signatures [1].

The offline signature verification system discussed in this paper is implemented using a Siamese Network containing twin Convolutional Neural Networks (CNN). The Siamese neural network consists of two identical CNNs, where each of them are capable of learning the hidden features of an input vector. Both networks have the same structure and parameters such that they can compare their outputs at the end. Training a signature verification system under a writer independent scenario, divides the available signers into train and test sets and for a particular signer, signatures are coupled as similar (genuine, genuine) or dissimilar (genuine, forged) pairs [1].

Our paper focuses on comparing systems trained on data from various dataset, using different network models and classifiers. Experiments are also conducted with two different distance functions namely, cosine and Euclidean distances to obtain the similarity or dissimilarity score of the feature embeddings from the two signature images.

2. Literature Survey

Recent works on handwritten signature verification have explored different CNN architectures with or without a separate classifier. In [2], Jerome et al. propose the use of a basic CNN architecture with three convolution and max pooling layers in an alternating fashion. The model achieves 98.23% validation accuracy but a new class has to be created for the genuine and forged signatures of each signer. Sultan et al. [3] propose a three-layer CNN architecture for feature extraction and classification with data augmentation to improve performance. Krishnaditya et al. [4] compare two optimizers: Adam and RMSProp, on their CNN model for three different datasets.

An architecture using both CNN and Crest Trough method for signature recognition along with Harris and Surf Algorithms for forgery detection is proposed by Jivesh et al. [5]. Harris corner detection algorithm and Surf feature extraction algorithm is seen again in the model proposed by Debasree et al. in [6].

Hanmandlu et al. [7] have used two CNN architectures for feature extraction: LeNet and AlexNet. An SVM classifier using the Cubic kernel in conjunction with AlexNet performed to provide a 96.6% recognition rate on the GPDS960 database. GoogLeNet Inception V1 and Inception V3 CNN architectures are used by Jahandad et al. in [8], where the Inception V1 model outperformed Inception V3. Hsin-Hsiung Kao et al. [9] propose a signature verification method based on explainable deep learning and local feature extraction on a single reference sample. They used two architectures: VGG19 and Inception V3. In [10], G. Alvarez et al. based their model on the VGG16 architecture. They trained and tested using the ICDAR 2011 SigComp dataset. Atefeh et al. [11] use pretrained models of VGG16, VGG19, ResNet50, Inception V3, and signature verification networks SigNet and SigNet-F to review existing CNN based models.

The SigNet model proposed in [1] uses a convolutional Siamese network. S. Dey et al. propose the first implementation of a CNN–Siamese network for offline signature verification and their results are used as a benchmark in this paper. OSVNet by C. Sekhar et al. [12] is another CNN–Siamese based writer independent model which produces 78.6% to 100% accuracy for MCYT-100 and MCYT-330 datasets.

3. Siamese Neural Network and Deep Transfer Learning

Bromley et al. [13] introduced the Siamese network in the early 1990s, and it was used to solve signature verification using the image matching concept. A Siamese network, as the word suggests, is a twin network architecture that usually contains two identical

networks. Each one of the twins have the same configuration with the same weights, which can be trained to learn features where similar observations are kept in proximity. A pair of signatures are input to the twin CNNs and the high-level features in the images are learnt by each network. At the end the two networks are joined by a distance layer, where a distance function calculates the difference of the extracted features in the last output layer. A widely known loss function – contrastive loss, receives the distance calculated using the similarity/dissimilarity metric – Cosine or Euclidean distance.

This paper deploys different types of Convolutional Neural Network architectures for the twins and presents a comparative study of each model against various types of datasets and distance functions. The architectures in study are implemented using pre-trained models via transfer learning, which helps in adapting the weights from previous image classification tasks to help in faster model convergence and better feature extraction.

4. Experiments

4.1. Datasets: A variety of datasets with unique structure and features of the signatures are taken for studying the model performance when the input is diverse. We have gathered different well-known datasets to evaluate the models with the aim of establishing a comparative study in this paper. The datasets considered are CEDAR, BHSig260[14], a dataset named Handwritten Signatures from Kaggle for which the original source is unknown, UTSig[15] and ICDAR 2011 Signature Verification Competition datasets [16]. Only the offline samples are used in each dataset. Table 1 and Table 2 shows the details of the datasets used.

Table 1. Datasets Description

Dataset		No. of Signers	No. of Genuine Signatures per Signer	No. of Forged Signatures per Signer	Language
CEDAR		55	24	24	English
BHSig260 (Hindi)		160	24	30	Hindi
BHSig260 (Bengali)		100	24	30	Bengali
Kaggle (Source unknown)		30	5	5	English
UTSig		115	27	42	Persian
ICDAR 2011 SigComp (Chinese)	Train Set	10	21 to 24	23 to 36	Chinese
ICDAR 2011 SigComp (Dutch)	Train Set	10	23 to 24	8 to 16	Dutch

ICDAR 2011 SigComp test set has a different structure from the previous datasets as shown in Table 2. Each dataset, Chinese and Dutch, have reference and questioned signatures from different signers. The image file is named such that genuine and forged pairs can be distinguished for creating the test labels. The number of reference and questioned signatures are not consistent for each signer unlike other datasets.

Table 2. ICDAR 2011 SigComp Test Sets

Dataset	No. of Signers	No. of Reference Signatures per Signer	No. of Questioned Signatures per Signer
Chinese	10	10 to 12	46 to 59
Dutch	10	12	20 to 24

A few of the signature images are shown in Figure 1. The signatures are taken from different datasets showing a genuine and forged pair for each language.
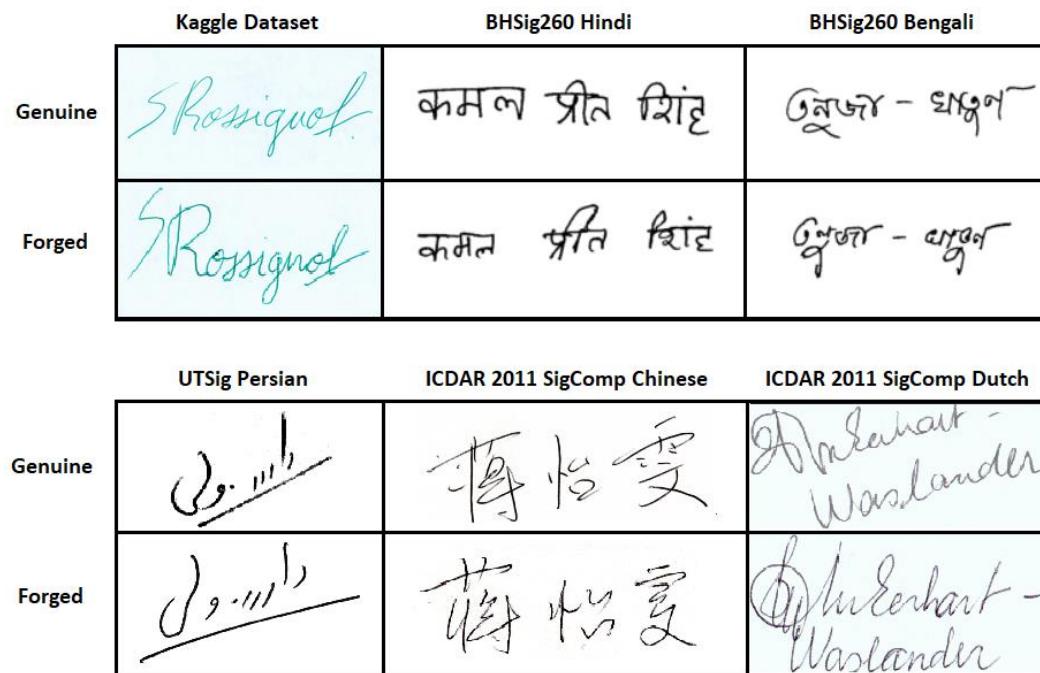

Figure 1. Signature Images in Datasets

4.2. Preprocessing

The images are taken from a wide gamut of datasets, the aim of pre-processing is to prepare all the signatures for further operations and increase the feasibility of learning. A size S=H*W is maintained as a default for training the network. Signature images inherently have noise present around the strokes which may decrease the model performance.

The images undergo three essential steps after being resized. I) Gray Scaling: Grey Scaling includes converting the RGB image into a grayscale image by inverting the pixels based off a weighted average threshold. To provide a 3-channel input image to the pretrained model, the one channel image resulting from the grey scale operation is stacked up to form three layers of the same pixel values. II) Binary processing: To suppress noise and the grey remains, the image is converted to a binary image to have only black and white pixel values by global/local thresholding. The background pixel intensities above a threshold are converted to 255 and below to 0 transforming the images using Otsu's algorithm. III) Denoising: Maintaining the original features of a signature and removing noise without waning the image requires usage of image denoising filters. OpenCV avails a tool fastNlMeansDenoising which we use to fine tune and remove noises with template

window size=7 pixels and search window size= 21 pixels. It theoretically performs a non-local mean sorting pixels and the replace mean values to smoothen out grainy noise of grey-images. Finally, the image is sharpened for resolution and the pixel values are normalised as a precondition for optimal network training.

Figure 2. Signature Image Pre-processing

4.3. Models

SigNet[1] signature verification model was used as base for the Siamese architecture in this paper. SigNet hyperparameters and final dense layers were combined with pretrained models downloaded from the Keras Applications API. The base network layers were frozen to preserve the weights essential for feature extraction. The different pretrained architectures used are VGG16, ResNet50, MobileNetV2, DenseNet121 and Xception. A few of the model details are mentioned in Table 3. The dense layers and dropout layers along with the base network make up the feature extractor. Input pairs are passed to this network and their final feature embeddings are given to a distance function to calculate similarity. A loss function receives the calculated distance and adjusts parameters so as to decrease distance between (genuine, genuine) pairs and increase distance between (genuine, forged) pairs.

Table 3. Pre-trained Models Description

Pre-trained model	Depth	No. of Parameters	Output Feature Layer	Output Feature Size
VGG16	16	138M	block5_pool (MaxPool2D)	7* 7* 512
ResNet50	50	23M	conv5_block3_out (Activation)	7*7*2048
MobileNetV2	53	3.4M	out_relu (ReLu)	7*7*1280
DenseNet121	121	20M	relu/Relu:0 (ReLu)	7*7*1024
Xception	71	22M	block14_sepconv2_act (Activation)	7*7*2048

4.4. Classifiers

Classification of the distance vector output from the predict() function of the model can be done using different classifiers like SVM, KNN, Gaussian Naïve Bayes and Logistic Regression. All classifiers used in our experiments are trained on the distance vectors of the embeddings of the (genuine, genuine) and (genuine, forged) image pairs along with the class labels (similar/dissimilar). Initially, the accuracy and best distance threshold is calculated by looping through the minimum and maximum distance values in steps of 0.001 and checking the number of samples correctly classified using that distance threshold. Classifiers do this job by using linear and non-linear techniques and sometimes provide more accurate results. Hence, the accuracies using each classifier is recorded and compared. KNN was configured at n=2. SVC was tested on different kernels–RBF, Poly, Linear and Sigmoid and RBF was selected as it consistently produced maximum test accuracy. The classifier models were imported from the scikit-learn library.

4.5. Experimental Setup

To study the performance of different pre-trained models on detecting forgery, our experiments follow the given design.
1. Load data and generate pairs of similar and dissimilar classes
2. Pre-process the generated image pairs which yield batches of grey-scaled, noise free and normalized arrays of pixel values
3. Split the data into train, test and validation sets after investigating the dataset structure: datasets like SigComp provide separate train and test sets for original and forged signatures whereas BHSig has one folder for original signatures and one for forged; train, test and validation sets must be extracted from these folders.
4. Create the final model by adding dense layers on the loaded pre-trained model
5. Train the network on each dataset separately.
6. Compute its accuracy for the batches of the test data using different classifiers.

Pretrained models are loaded from Keras Applications API, two dense layers are added after flattening the base model's output. The first Dense layer has 1024 neurons with a Dropout rate of 0.5, whereas the second Dense layer has 128 neurons. Rectified Linear Units (ReLU) activation function is used on the output of the fully connected layers. The twin CNNs with the above structure output image encodings whose distance is calculated using a Euclidean distance function and again using a cosine distance function for comparative study.

This Siamese network is trained using RMSprop for 15 epochs using contrastive loss, with momentum rate equal to 0.9 and batch size equal to 128. The initial learning rate (LR) is set to $1e-4$ with hyper parameters $\rho = 0.9$ and $\varepsilon = 1e-8$. Early stopping with patience set as 12 is used to reduce overfitting. ReduceLROnPlateau is called to reduce learning rate when model metric has stopped improving, with factor set as 0.1, patience as 5 and minimum learning rate as 1×10^{-6} [1]. The entire framework is implemented using Keras library with the TensorFlow as backend. Trained was done using a single 12GB NVIDIA Tesla K80 GPU, and the overall model training took approximately 2 to 5 hours depending on the dataset.

The trained Siamese model outputs a distance vector which is used to decide if the image pair belongs to the (genuine, genuine) class or (genuine, forged) class, this is fed to a separate binary classifier which yields the target label. The classifiers used are SVC using RBF kernel, KNN, Gaussian Naïve Bayes and Logistic Regression. Final kernel for SVC was chosen on a trial-and-error basis.

4.6. Results and Discussions

The accuracies for each model with all the datasets and both distance functions are shown in Table 4, Table 5, Table 6, Table 7 and Table 8. The test accuracy refers to the accuracy calculated by looping through different threshold values and checking the number of correctly classified image pairs.

Table 4. VGG16 Accuracies for Different Distance Functions & Classifiers

Dataset	Distance Function	Test	LR	GNB	KNN	SVC (RBF)
Kaggle	Euclidean	100	100	100	100	100
	Cosine	89.03	96.0	96.0	89.5	97
CEDAR	Euclidean	100	100	100	99.1	100
	Cosine	50.0	65.1	65.4	64.3	65.1
BHSig260 (Hindi)	Euclidean	64.54	63.75	62.26	59.87	64.02
	Cosine	93.6	93.08	93.08	88.04	93.1
BHSig260 (Bengali)	Euclidean	83.0	81.74	74.56	80.7	81.2
	Cosine	71.04	70.11	69.62	62.08	70.86
UTSig	Euclidean	79.28	71.5	72.11	65.16	73.01
	Cosine	87.47	87.33	87.17	83.31	87.37
ICDAR 2011 SigComp (Chinese)	Euclidean	64.0	58.6	57.8	69.5	67.2
	Cosine	63.59	55.72	51.04	55.98	55.98
ICDAR 2011 SigComp (Dutch)	Euclidean	67.9	62.18	61.48	64.53	66.4
	Cosine	60.47	55.0	55.0	52.18	54.92

The VGG16 architecture produces best results for the smallest dataset, which was obtained from Kaggle and the second smallest dataset CEDAR using the Euclidean distance. The Support Vector Classifier on RBF kernel performs the best consistently following which are the Gaussian Naïve Bayes and Logistic Regression classifiers. Both give similar accuracies. K-Nearest Neighbors classifier does not produce up to par accuracies but is still acceptable for most datasets. Cosine distance works well for BHSig260 Hindi and UTSig Persian datasets while Euclidean distance performs best for the remaining datasets.

Table 5. ResNet50 Accuracies for Different Distance Functions & Classifiers

Dataset	Distance Function	Test	LR	GNB	KNN	SVC (RBF)
Kaggle	Euclidean	78.0	75.0	78.0	75.5	75.0
	Cosine	90.5	89.5	90.0	90.5	89.0
CEDAR	Euclidean	66.75	64.93	59.17	62.03	64.53
	Cosine	76.01	76.6	76.0	75.0	75.8
	Euclidean	82.82	82.59	82.67	77.49	82.79

BHSig260 (Hindi)	Cosine	84.15	84.1	84,08	82.17	84.13
BHSig260 (Bengali)	Euclidean	84.20	83.41	83.97	82.98	83.41
	Cosine	78.3	50.5	78.8	50.0	70.25
UTSig	Euclidean	78.71	66.44	66.49	62.71	66.34
	Cosine	82.23	80.0	80.21	79.86	81.48
ICDAR 2011 SigComp (Chinese)	Euclidean	65.61	62.6	57.79	98.29	70.29
	Cosine	63.80	64.04	65.13	69.47	65.04
ICDAR 2011 SigComp (Dutch)	Euclidean	62.65	61.32	60.6	60.07	62.15
	Cosine	63.31	58.16	61.63	61.8	60.07

ResNet50 produces the best accuracy for the ICDAR 2011 SigComp Chinese dataset on Euclidean distance, but only for the KNN classifier. Looking at the overall accuracies, Gaussian Naïve Bayes classifier works well consistently but not better than the direct threshold-based classification used to produce the test accuracy. Cosine distance works well for most datasets except BHSig260 Bengali and ICDAR 2011 SigComp Chinese.

Table 6. MobileNetV2 Accuracies for Different Distance Functions & Classifiers

Dataset	Distance Function	Test	LR	GNB	KNN	SVC (RBF)
Kaggle	Euclidean	78.0	78.0	75.0	76.0	78.0
	Cosine	80.0	78.0	79.0	80.0	79.5
CEDAR	Euclidean	92.0	92.0	73.0	88.0	89.0
	Cosine	96.0	96.0	95.9	96.0	96.0
BHSig260 (Hindi)	Euclidean	75.0	75.0	74.0	75.0	75.0
	Cosine	70.73	69.0	68.0	68.8	68.6
BHSig260 (Bengali)	Euclidean	85.63	85.0	73.4	80.3	83.4
	Cosine	75.51	75.0	75.0	74.6	74.5
UTSig	Euclidean	74.6	74.0	74.5	74.5	49.0
	Cosine	73.56	74.0	73.5	73.4	50.0
ICDAR 2011 SigComp (Chinese)	Euclidean	57.25	50.1	64.9	57.9	56.7
	Cosine	54.46	59.1	49.7	59.1	59.1
ICDAR 2011 SigComp (Dutch)	Euclidean	59.52	59.5	56.5	57.3	57.5
	Cosine	64.83	64.2	64.3	63.3	64.2

MobileNetV2 produces the best accuracies using the Cosine distance on the CEDAR dataset. Logistic Regression classifier and SVC on RBF kernel seem to do well for almost all the datasets but the test accuracy gives equally good results. Here, Cosine and Euclidean

distances perform well on alternative datasets with Cosine outperforming Euclidean for the Kaggle, CEDAR, BHSig260 Bengali and ICDAR 2011 SigComp Dutch datasets.

Table 7. DenseNet121 Accuracies for Different Distance Functions & Classifiers

Dataset	Distance Function	Test	LR	GNB	KNN	SVC (RBF)
Kaggle	Euclidean	62.5	96.5	97.0	96.5	96.0
	Cosine	98.5	62.5	61.5	60.0	62.5
CEDAR	Euclidean	69.63	68.0	65.0	65.3	68.65
	Cosine	67.82	65.94	65.39	64.6	64.92
BHSig260 (Hindi)	Euclidean	77.47	77.0	72.0	76.0	76.7
	Cosine	81.08	80.8	80.3	79.4	80.5
BHSig260 (Bengali)	Euclidean	80.83	81.0	70.8	78.6	79.6
	Cosine	74.18	74.0	73.8	73.7	73.5
UTSig	Euclidean	76.83	69.3	68.7	63.9	69.9
	Cosine	74.29	70.8	68.9	61.0	66.0
ICDAR 2011 SigComp (Chinese)	Euclidean	63.31	66.1	75.6	71.3	71.3
	Cosine	64.63	62.0	66.5	71.0	64.0
ICDAR 2011 SigComp (Dutch)	Euclidean	64.22	62.5	63.0	60.0	62.4
	Cosine	60.73	64.5	62.2	64.3	64.3

DenseNet121 has the best accuracy for Kaggle dataset when used with the distance thresholding test classifier. Logistic Regression, Gaussian Naïve Bayes and Support Vector Classifier on RBF perform fairly well in comparison to KNN and the Test accuracy. Logistic Regression produces slightly better accuracies for most of the datasets and Euclidean distance does well in the majority of the datasets.

Table 8. Xception Accuracies for Different Distance Functions & Classifiers

Dataset	Distance Function	Test	LR	GNB	KNN	SVC (RBF)
Kaggle	Euclidean	54.5	52	52.5	51	52.5
	Cosine	66.5	62.5	60.5	71.5	60
CEDAR	Euclidean	62.13	60.97	61.63	48.36	61.63
	Cosine	50.57	54.4	56.9	55.9	56.3
BHSig260 (Hindi)	Euclidean	75.32	72.95	74.81	73.57	74.48
	Cosine	73.96	73.76	73.86	72.66	73.86
BHSig260 (Bengali)	Euclidean	71.7	68.84	63.83	66.18	68.84
	Cosine	65.31	64.32	64.21	62.97	63.5
UTSig	Euclidean	58.3	53.87	54.94	51.28	50.77

	Cosine	53.24	51.61	51.78	50.19	49.79
ICDAR 2011 SigComp (Chinese)	Euclidean	59.94	54.94	77.6	69.01	69.01
	Cosine	59.63	61.16	67.18	68.30	68.08
ICDAR 2011 SigComp (Dutch)	Euclidean	60.29	57.3	57.6	58.5	57.5
	Cosine	60.90	58.12	58.98	58.51	58.28

Xception has its best accuracy of 77.6% on the ICDAR 2011 SigComp Chinese dataset using Euclidean distance. None of the classifiers perform well consistently and the test accuracy seems sufficient for the first few datasets. Euclidean distance brings good accuracies in most cases except for the Kaggle dataset.

The VGG16 model using Euclidean distance performs best for the Kaggle, CEDAR and ICDAR 2011 SigComp Dutch datasets. VGG16 on Cosine distance gives the best accuracies for BHSig260 Hindi and UTSig datasets. MobileNetV2 works the best using Test accuracy for the BHSig260 Bengali dataset while ResNet50 gives the best accuracy for ICDAR 2011 Chinese dataset on KNN classifier.

5. Conclusion

A comparative analysis is made using various pretrained models–VGG16, ResNet50, MobileNetV2, DenseNet121 and Xception on various datasets–CEDAR, Kaggle Dataset, BHSig260, UTSig Persian dataset, and ICDAR 2011 SigComp. A variety of classifiers– Support Vector Classifier, K-Nearest Neighbours, Gaussian Naïve Bayes and Logistic Regression, are used to see their effect on the overall performance. The implementation is inspired from the SigNet model. It adopts transfer learning methods using various Convolutional neural network architectures to implement the base twin networks in the Siamese model.

Based on the analysis of the results obtained, it can be concluded that Siamese network using VGG16 measured on Euclidean distance and Gaussian Naïve Bayes as the binary classifier best detects signature forgery with a maximum accuracy of 100% on CEDAR and Kaggle datasets. ResNet50 has the highest accuracy of 98.29% for detecting forgery in Chinese signatures obtained from ICDAR 2011 SigComp dataset of handwritten signatures. MobileNetV2 performed best for the CEDAR dataset using cosine distance with 96% accuracy and DenseNet121 produced 98.5% accuracy for the Kaggle dataset using the cosine distance function. Xception saw its best accuracy of 77.6% on the ICDAR 2011 SigComp Chinese dataset using the Euclidean distance function.

Future works include implementing fine tuning, exploring more loss functions, models and datasets. Curriculum learning would be another interesting area to look into and different hyperparameters can be tested for each model.

6. Acknowledgements

REFERENCES

[1]   Dey, Sounak & Dutta, Anjan & Toledo, J. & Ghosh, Suman & Lladós, Josep & Pal, Umapada. (2017). SigNet: Convolutional Siamese Network for Writer Independent Offline Signature Verification.

[2] Gideon, S & Kandulna, Anurag & Kujur, Aron & Diana, A & Raimond, Kumudha. (2018). Handwritten Signature Forgery Detection using Convolutional Neural Networks. Procedia Computer Science. 143. 978-987. 10.1016/j.procs.2018.10.336.

[3] S. Alkaabi, S. Yussof, S. Almulla, H. Al-Khateeb and A. A. AlAbdulsalam, "A Novel Architecture to verify Offline Hand-written Signatures using Convolutional Neural Network," 2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 2019, pp. 1-4, doi: 10.1109/3ICT.2019.8910275.

[4] K. Kancharla, V. Kamble and M. Kapoor, "Handwritten Signature Recognition: A Convolutional Neural Network Approach," 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), 2018, pp. 1-5, doi: 10.1109/ICACAT.2018.8933575.

[5] Poddar, Jivesh & Parikh, Vinanti & Bharti, Drsantosh. (2020). Offline Signature Recognition and Forgery Detection using Deep Learning. Procedia Computer Science. 170. 610-617. 10.1016/j.procs.2020.03.133.

[6] Mitra, Debasree & Baksi, Aurjyama & Modak, Alivia & Das, Arunima & Das, Ankita. (2019). Machine Learning Approach for Signature Recognition by HARRIS and SURF Features Detector. International Journal of Computer Sciences and Engineering. 7. 73-80. 10.26438/ijcse/v7i5.7380.

[7] Hanmandlu, M. & Sronothara, A. & Vasikarla, Shantaram. (2018). Deep Learning based Offline Signature Verification. 732-737. 10.1109/UEMCON.2018.8796678.

[8] Jahandad, & Sam, Suriani & Kamardin, Kamilia & Sjarif, Nilam & Mohamed, Norliza. (2019). Offline Signature Verification using Deep Learning Convolutional Neural Network (CNN) Architectures GoogLeNet Inception-v1 and Inception-v3. Procedia Computer Science. 161. 475-483. 10.1016/j.procs.2019.11.147.

[9] Kao, Hsin-Hsiung & Wen, Che-Yen. (2020). An Offline Signature Verification and Forgery Detection Method Based on a Single Known Sample and an Explainable Deep Learning Approach. Applied Sciences. 10. 3716. 10.3390/app10113716.

[10] Alvarez, Gabe, Blue Sheffer, and Morgan Bryant. "Offline signature verification with convolutional neural networks." Technical report, Stanford University, 2016.

[11] A. Foroozandeh, A. Askari Hemmat and H. Rabbani, "Offline Handwritten Signature Verification and Recognition Based on Deep Transfer Learning," 2020 International Conference on Machine Vision and Image Processing (MVIP), 2020, pp. 1-7, doi: 10.1109/MVIP49855.2020.9187481.

[12] C. S. Vorugunti, G. Devanur S, P. Mukherjee and V. Pulabaigari, "OSVNet: Convolutional Siamese Network for Writer Independent Online Signature Verification," 2019 International Conference on Document Analysis and Recognition (ICDAR), 2019, pp. 1470-1475, doi: 10.1109/ICDAR.2019.00236.

[13] Bromley, Jane & Bentz, James & Bottou, Leon & Guyon, Isabelle & Lecun, Yann & Moore, Cliff & Sackinger, Eduard & Shah, Rookpak. (1993). Signature Verification using a "Siamese" Time Delay Neural Network. International Journal of Pattern Recognition and Artificial Intelligence. 7. 25. 10.1142/S0218001493000339.

[14] Alaei, Alireza & Pal, Srikanta & Pal, Umapada & Blumenstein, Michael. (2017). An Efficient Signature Verification Method Based on an Interval Symbolic Representation and a Fuzzy Similarity Measure. IEEE Transactions on Information Forensics and Security. PP. 1-1. 10.1109/TIFS.2017.2707332.

[15] Soleimani Bajestani, Amir & Fouladi, Kazim & Araabi, Babak. (2016). UTSig: A Persian offline signature dataset. Preprint, submitted to IET biometrics. 6. 10.1049/iet-bmt.2015.0058.

[16] M. Liwicki et al., "Signature Verification Competition for Online and Offline Skilled Forgeries (SigComp2011)," 2011 International Conference on Document Analysis and Recognition, 2011, pp. 1480-1484, doi: 10.1109/ICDAR.2011.294.