International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Deep Feature-Based Writer-Dependent Classifiers for Offline Signature Verification

D S Guru[1], H Annapurna[1], K S Manjunatha[2]

[1]Department of Studies in Computer Science, University of Mysore, Mysuru – 570 006, Karnataka, India
Email: *dsg[at]compsci.uni-mysore.ac.in*

[1]Yuvaraja's College, A Constituent Autonomous College of the University of Mysore, Mysuru – 570 005, Karnataka, India
Email: *annapurna_h[at]ycm.uni-mysore.ac.in*

[2]Maharani's Science College for Women, Mysuru-570001, Karnataka, India
Email: *kowshik.manjunath[at]gmail.com*

Abstract: ***In this work, we proposed a novel scheme based on deep architectures for offline signature verification. The proposed method introduces the notion of writer-dependent deep architectures for offline signature verification. Compared to the current signature verification techniques that use the same architecture for all writers, the proposed model based on applying deep architecture which may vary from a writer to writer. In this work, writer-dependency has been exploited at two stages: In the first stage, writer-dependent deep architectures are selected for each writer. In the second stage, writer-dependent deep architectures are used as feature extractors, and then the dimensionality of a feature vector is reduced through the application of linear dimensionality reduction technique. Finally, writer-dependent classifiers are fixed for each writer. At the verification stage, to establish the authenticity of the test signature, features are extracted from the writer-dependent architecture and fed into the writer-dependent classifier of the claimed writer. Extensive experiments are carried out on two benchmark offline signature datasets: CEDAR and MCYT, to validate the performance of the proposed model. The obtained results clearly indicate the efficacy of the proposed methodology.***

Keywords: Offline signature verification, Deep features, Writer-dependent deep architecture, Writer-dependent classifiers

1. Introduction

As technology advances, secure and reliable authentication systems are becoming increasingly essential. In biometric-based authentication, individuals are identified based on their physiological characteristics, such as the face, iris, hand geometry, and fingerprint, or behavioral characteristics, including signature, voice, and gait [27]. Among these, handwritten signatures are one of the most widely used behavioral biometric traits, playing a crucial role in sectors such as banking, finance, legal document verification, forensic analysis, security systems, and various industries.

With the growing adoption of digital signatures, automatic signature verification has become a vital aspect of biometric security. The primary objective of an automatic signature verification system is to authenticate an individual by determining whether a given signature is genuine or forged [15].

In biometric-based authentication systems, signatures can be collected in either online or offline modes. In offline mode, signature images are captured using optical scanners or cameras. In contrast, online mode utilizes specialized hardware devices such as digitizing tablets, electronic pens, and personal digital assistants, which record various dynamic features during the signing process. These features include velocity, starting point, number of strokes, acceleration, writing speed, and pressure, all of which are stable and distinctive for each individual [27].

However, offline signature verification remains challenging due to high intrapersonal variability. Factors such as available signing space, pen and ink type, and the signer's mental and physical state contribute to variations in signatures. Additionally, compared to online signature verification, offline verification is more complex because it lacks dynamic information and typically has a limited number of training samples [15].

A deep convolutional neural network (DCNN) is a machine learning tool inspired by the structure and function of the human brain's neural networks. In this work, we highlight the significance of writer-dependent characteristics for offline signature verification using DCNNs. We propose a novel approach that leverages writer-dependent deep architectures to enhance verification accuracy. By incorporating writer dependency at both the deep architecture and classifier levels, we achieve a significant improvement in verification performance.

To represent signature samples, we utilize deep features. Instead of relying on a single pre-trained deep architecture, we employ AlexNet, Inception-V2, Inception-V3, ResNet50, SqueezeNet, VGG-16, and VGG-19 to extract features for each writer. Additionally, writer-dependent classifiers are determined by feeding these deep features into various conventional classifiers, including Support Vector Machine (SVM) with different kernel functions (linear, radial basis function, sigmoid, and polynomial), K-Nearest Neighbor (KNN), Decision Tree, Random Forest, and Naïve Bayes algorithms.

To assess the robustness of the proposed method, we conduct extensive experiments on two standard benchmark offline signature datasets: CEDAR and MCYT. The results demonstrate that our approach is not only highly effective but also simple and efficient.

2. Related work

Over the past four decades, extensive research has been conducted on offline signature verification. In the literature, signature verification approaches are broadly categorized into writer-independent and writer-dependent methods [15]. In the writer-independent approach, a single model is trained using the same parameters and matching algorithms for all writers. The primary advantage of this method is its efficiency, training the system requires less time, and adding or removing writers from the database does not affect overall performance. In contrast, the writer-dependent approach requires training a separate model for each writer with specific parameters and matching techniques. This method closely resembles manual verification performed by human experts. As a result, writer-dependent models generally achieve higher verification accuracy compared to writer-independent models.

Researchers have proposed numerous writer-independent models for offline signature verification [12], [30], [41], [44], [48], [49], [52], [59]. Offline signature verification involves analyzing the features of a handwritten signature image to determine its authenticity.

In the context of offline signatures, various types of features are used to represent signatures. These features are broadly classified into three categories: local features, which capture fine-grained details of specific signature regions [3], [4], [11], [19], [28]; global features, which represent the overall structure and shape of the signature [1], [11], [31], [35], [45], [52], [63]; and deep features, which are extracted using deep learning models to capture complex patterns and representations [5], [14], [16], [21], [23], [44], [51], [60].

The authenticity of a signature is determined during the verification step using various pattern recognition algorithms, such as simple distance measures [2], [9], [14], [28]; neural networks [9], [22], [40], [55]; support vector machines (SVMs) [7], [19], [38], [52], [64]; hidden Markov models (HMMs) [10], [32], [57]; naïve Bayes [47], [50]; and fuzzy functions [3], [17], [24].

Pre-trained deep convolutional neural networks have been used for offline signature verification in recent years, including shallow convolutional neural networks (sCNNs) [31]; VGG-16 [5]; Signet [14], [23]; deep convolutional generative adversarial networks (DCGANs) [61]; hierarchical one-class convolutional neural networks [51]; recurrent neural networks (RNNs) [18]; and Siamese neural networks [58]. Although deep architectures improve model accuracy, they require significant time and spatial complexity.

In the aforementioned works, the models are trained using a common feature set and classifier for all writers. However, only a few researchers have proposed writer-dependent models for offline signature verification. In these models, writer dependency is exploited at various levels, such as the classifier level [8], [12], [13], [49], feature level [36], and threshold level [20].

2.1 Findings

However, due to high intra-class variations, a handwritten signature is considered a complex biometric trait [15]. When a human expert manually verifies an individual's signatures, they use different features and matching criteria for each writer. Therefore, using the same set of features and classifiers for all writers may not be effective [37]. In the literature, no attempts have been made to utilize writer-dependent deep architectures and writer-dependent classifiers for offline signature verification. These challenges motivated us to design an effective writer-dependent automatic signature verification model based on deep convolutional neural networks (DCNNs).

The main contributions of this work are as follows:

- A novel deep convolutional neural network-based writer-dependent approach for offline signature verification.
- Exploration of pre-trained deep convolutional neural network architectures for writer-dependent offline signature verification.
- Investigation of deep architectures and adaptation of writer-dependent classifiers.
- Extensive experiments conducted on two standard offline signature databases.
- Achievement of high accuracy through the use of writer-dependent characteristics.

The structure of this paper is as follows: Section 1 provides an introduction, a brief literature review. Section 2 includes a brief literature survey, and the contributions of this work. Section 3 describes the proposed methodology. Section 4 presents the experimental setup and the obtained results. Section 5 offers a comparative analysis, and Section 6 concludes the paper.

3. Proposed Model

The proposed model consists of six steps: pre-processing, feature extraction, writer-dependent deep architecture selection, dimensionality reduction, writer-dependent classifier selection, and verification. The block diagram illustrating the proposed model based on deep architecture is presented in Figure 1.

3.1 Pre-processing

The preprocessing of signature images is an essential step performed before feature extraction. Initially, the input signature image is converted into a binary image to reduce computational complexity [43]. After binarization, a median filter is applied to remove noise [56]. Once the filtered image is obtained, morphological operations such as closing and thinning are applied, followed by cropping the signature image area.

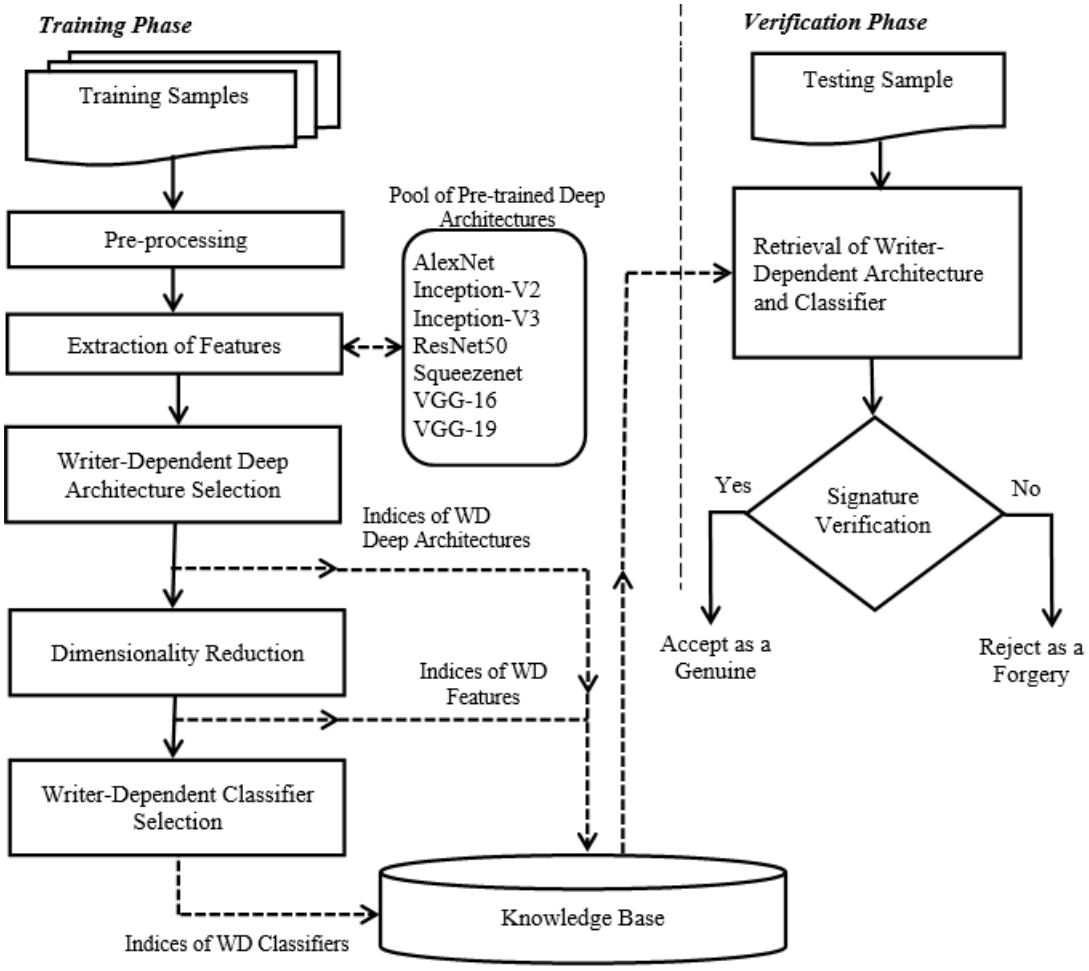

Figure 1: The block diagram of the proposed model

3.2 Extraction of Features

Once the image has been pre-processed, we utilize pre-trained deep convolutional neural networks (DCNNs) to extract deep features from signature images. A pre-trained model has already been trained on a dataset, with weights and biases that capture essential features from the original dataset. These learned features are often transferable to different datasets, making pre-trained models efficient for feature extraction while reducing computational time.

In this work, we employ various pre-trained deep architectures for feature extraction, including AlexNet [34], Inception-V2 [54], Inception-V3 [54], ResNet50 [25], SqueezeNet-1.0 [26], VGG-16 [53], and VGG-19 [53]. Further details on these pre-trained architectures can be found in their respective studies.
The AlexNet architecture [34] consists of eight layers with approximately 60 million learnable parameters. It comprises five convolutional layers with a 7×7 convolution kernel size and three fully connected layers. The first, second, and fifth convolutional layers include max-pooling layers to enhance feature extraction.

The pre-processed input signature image, with a size of 227×227×3, is fed into the first convolutional layer, which contains 96 filters of 11×11 with a stride of 4, producing an output feature map of 55×55×96. This is followed by a max-pooling layer of 3×3 with a stride of 2, reducing the feature map size to 27×27×96.

The second convolutional layer applies 256 filters of 5×5 with a stride of 1 and padding of 2, generating an output of 27×27×256. Another max-pooling layer (3×3, stride 2) follows, reducing the feature map size to 13×13×256. The third convolutional layer consists of 384 filters of 3×3 with a stride of 1 and padding of 1, resulting in a feature map of 13×13×384. The fourth convolutional layer also has 384 filters of 3×3 with the same stride and padding, maintaining the output size of 13×13×384. The fifth convolutional layer applies 256 filters of 3×3, with the same stride and padding, producing an output of 13×13×256.

A third max-pooling layer of 3×3 with a stride of 2 is applied, reducing the feature map to 6×6×256. This is followed by a dropout layer with a dropout rate of 0.5. The first fully connected layer outputs 4,096 neurons, followed by another dropout layer. The second fully connected layer also consists of 4,096 neurons. Finally, the third fully connected layer extracts 1,000 features for each input signature image.

Inception-V2 [54] is the second generation of the Inception convolutional neural network (CNN) architecture, comprising 42 layers. It is built using repeated components called Inception modules, each containing multiple filters of varying sizes at the same level. Unlike conventional CNNs, where deeper layers are used to capture hierarchical features, Inception-V2 utilizes parallel layers, making the model wider rather than deeper. Additionally, it incorporates 1×1 convolutional filters, which do not capture spatial patterns but

effectively learn patterns across the depth (cross-channel correlations) of the input image. To enhance network stability and performance, Inception-V2 employs batch normalization layers after each convolutional layer, reducing internal covariate shifts and improving training efficiency. The obtained output consists of 1000 features for each signature sample input.

Inception-V3 [54] is the third iteration of the Inception network architecture, building upon its predecessors with enhanced efficiency and performance. Like previous versions, Inception-V3 employs a modular design, consisting of multiple Inception modules that utilize filters of varying sizes to capture diverse feature representations. A key feature of Inception-V3 is the incorporation of batch normalization layers after each convolutional layer, which improves network stability, accelerates training, and enhances overall performance. Compared to other deep CNN architectures, Inception-V3 is designed to train faster and more efficiently, making it well-suited for complex computer vision tasks. It has been widely used in image classification, object detection, and face recognition applications. This gives 1000 features for each input signature image.

ResNet-50 [25], short for Residual Network, is a deep convolutional neural network (CNN) designed to address the vanishing gradient problem in deep networks through residual learning. It consists of 50 layers, including 48 convolutional layers, one max-pooling layer, and one average-pooling layer. ResNet-50 is built using residual blocks, which enable deeper network architectures without degradation in performance. The first convolutional layer applies 64 distinct kernels of size 7×7 with a stride of 2, followed by max pooling with a stride of 2.

Subsequent convolutional layers are structured as follows:
Three repeated blocks containing 64 filters of size 1×1, 256 filters of size 3×3, and 256 filters of size 1×1, totaling 9 layers. Four repeated blocks containing 128 filters of size 1×1, 128 filters of size 3×3, and 512 filters of size 1×1, totaling 12 layers.

Six repeated blocks containing 256 filters of size 1×1, 256 filters of size 3×3, and 1024 filters of size 1×1, totaling 18 layers. Three repeated blocks containing 512 filters of size 1×1, 512 filters of size 3×3, and 2048 filters of size 1×1, totaling 9 layers. After the convolutional layers, an average pooling layer is applied, followed by a fully connected layer with 1,000 nodes and a softmax function, producing an output of 1,000 features for each input signature sample.

SqueezeNet_1.0 [26] is a lightweight convolutional neural network (CNN) designed to minimize model size while maintaining high performance, making it ideal for embedded systems and resource-constrained applications. It achieves this efficiency through Fire modules, which consist of a squeeze layer that reduces input channels using 1×1 convolutions, acting as a bottleneck, and an expand layer that increases channels using a combination of 1×1 and 3×3 convolutions, ensuring the feature map size remains unchanged. The architecture begins with an initial convolutional layer (conv1), followed by eight Fire modules (fire2–fire9) and concludes with a final convolutional layer (conv10). To downsample feature maps, max pooling with a stride of 2 is applied after conv1, fire4, fire8, and conv10, and a 50% dropout is introduced after fire9 to prevent overfitting. Finally, 1000 features are extracted for each input signature image. Unlike traditional deep CNNs, SqueezeNet eliminates fully connected layers, significantly reducing memory and computational costs while preserving classification accuracy.

VGG-16 [53] is a 16-layer convolutional neural network (CNN) developed by the Visual Geometry Group (VGG), consisting of 13 convolutional layers and three fully connected layers. It follows a structured design with five convolutional blocks, each followed by a max-pooling layer. The input signature images (224×224×3) are processed through stacked convolutional layers, where all hidden layers use ReLU activation for non-linearity. The first block contains two 3×3 convolutional layers with 64 filters, producing a 224×224×64 feature map, followed by a 2×2 max-pooling layer (stride 2) that reduces it to 112×112×64. The second block follows the same pattern but with 128 filters, reducing the size to 56×56×128. The third block consists of three 3×3 convolutional layers with 256 filters, followed by max-pooling, yielding a 28×28×256 feature map. The fourth and fifth blocks each contain three 3×3 convolutional layers with 512 filters, and after max-pooling, the feature map size is reduced to 7×7×512. This final feature map is flattened and passed through two fully connected layers with 4,096 neurons each, followed by a final fully connected layer with 1,000 neurons using a softmax activation function, extracting 1,000 features per input signature image. The use of small 3×3 filters, progressive max-pooling, and fully connected layers makes VGG-16 an effective model for image classification, object recognition, and feature extraction.

VGG19 is an extension of VGG16, sharing the same overall architecture, particularly in its final three fully connected layers. The key difference lies in the number of convolutional layers, VGG19 consists of 19 layers, including 16 convolutional layers and 3 fully connected layers, whereas VGG16 has 13 convolutional layers. This means that VGG19 incorporates three additional convolutional layers compared to VGG16, potentially allowing for a more detailed feature extraction process while maintaining the same fully connected structure.

Let S_k^i be the kth signature sample of a writer W_i, $(1 \leq k \leq n)$ where n is the number of samples available for training purposes of a writer $W_i (i = 1,2,3, \ldots, N)$. Here N indicates the total number of writers in a database. From each signature sample, P number of deep features are extracted. This results in a feature matrix for each writer of dimension $n \times P$. The feature matrix of size $N \times n \times P$ will be generated for the whole dataset.

3.3 Writer-dependent deep architecture selection

The proposed model utilizes a writer-dependent architecture, meaning the chosen deep architecture varies for each writer. After extracting deep features from signature images, the most suitable deep architecture is selected for each writer to optimize verification accuracy. The selection process is based on an evaluation of how well different deep architectures capture the unique characteristics of a writer's signature. This

adaptive approach ensures that the model tailors its feature extraction and classification process to the specific variations and intricacies of each writer's signature, enhancing overall verification performance.

Let n_g and n_f are the number of genuine and forgery signature samples of each writer available for training purposes, i.e., $n = n_g + n_f$. Hence, for each writer, we have feature matrices of size $n_g \times P$ and $n_f \times P$, extracted from the genuine and forgery signature samples, respectively. Let there be D number of pre-trained deep architectures. Using $n_g \times P$ and $n_f \times P$, verification accuracy (Ac) is estimated for each writer to select the suitable deep architecture. That is for the writer W_i we have,

$$Ac = \{Ac_1^i, Ac_2^i, Ac_3^i, \ldots\ldots, Ac_D^i\}$$

Where Ac_x^i refers to accuracy obtained from x^{th} deep architecture for writer W_i, $(1 \leq x \leq D)$.

In this approach, the deep architecture that achieves the highest verification accuracy for a particular writer is identified as the most suitable architecture for that writer. The corresponding deep features extracted from this architecture are then designated as the appropriate features for that writer. This process establishes the first writer-dependent parameter, the selection of a deep architecture based on verification accuracy.

3.4 Dimensionality reduction

The deep features are selected for each writer using the writer-dependent DCNN, we apply a linear dimensionality reduction technique to minimize computational complexity. Since DCNN-extracted feature vectors tend to be high-dimensional, directly processing them can be computationally expensive. To address this, we use Principal Component Analysis (PCA) to transform the feature vectors into a lower-dimensional space while retaining the most significant information. This reduction helps improve efficiency without significantly affecting the discriminative power of the features. As a result, each signature sample is represented using P key features in the transformed space, optimizing both performance and computational cost.

For instance, the features extracted from the signature sample S_k^i of a writer W_i is denoted as

$$F_k^i = \{f_{k_1}^i, f_{k_2}^i, f_{k_3}^i, \ldots\ldots, f_{k_P}^i\}$$

Here, $f_{k_j}^i$ denotes the index of feature obtained from the signature sample S_k^i. That is $f_{k_j}^i$ represents the j^{th} feature of k^{th} sample. The dimension of the feature vector F_k^i will be reduced to d number of features after the application of PCA and it is denoted as

$$RF_k^i = \{f_{k_1}^i, f_{k_2}^i, f_{k_3}^i, \ldots\ldots, f_{k_d}^i\},\ 1 \leq d \leq P.$$

3.5 Writer-dependent classifier selection

Since signature samples exhibit high intra-class variability and do not follow a uniform distribution across all writers, using a single classifier for all writers may not be practical. To address this, we have explored the idea of employing a writer-dependent classifier, where different classifiers are used for different writers. Once the dimensionality of the feature vectors is reduced, the selection of the most suitable classifier is performed. In this work, we consider multiple classifiers, including Support Vector Machine (SVM) with different kernel functions (linear, radial basis function, sigmoid, and polynomial), K-Nearest Neighbor (KNN), Decision Tree, Random Forest, and Naïve Bayes algorithms. The verification accuracy for each writer is calculated across all classifiers, and the classifier yielding the highest accuracy is selected as the writer-dependent classifier for that individual. In cases where multiple classifiers achieve the same accuracy, a predefined priority order is used to resolve the selection.

Let $\mathrm{C} = \{\mathrm{C}_1, \mathrm{C}_2, \mathrm{C}_3, \ldots\ldots, \mathrm{C}_\varphi\}$ be the list of classifiers, where φ be a number of classifiers and each writer is represented by $n_g \times d$ and $n_f \times d$, where $n_g \times d$ denotes the feature matrix obtained from the genuine training samples and $n_f \times d$ denotes the feature matrix obtained from the forgeries of each writer. Using $n_g \times d$ and $n_f \times d$ the verification accuracy (Ac) is calculated with the set of classifiers C. Finally, accuracy is obtained from each of the classifier. That is for the writer W_i we have

$$Ac^i = \{Ac_1^i, Ac_2^i, Ac_3^i, \ldots\ldots, Ac_\varphi^i\}$$

Where $Ac_{cl,}^i$ refers to accuracy obtained from cl^{th} classifier for wirter W_i . $1 \leq cl \leq \varphi$. A classifier with the highest accuracy is identified as suitable for a specific writer. For instance, the classifier selected for a writer W_i is denoted by, $\mathrm{C}_{sel}^i = min\{Ac\}$.

The details of the writer-dependent architecture, extracted features, and selected classifiers for each writer are stored in a knowledge base. This knowledge base serves as a reference for verifying future signature samples of the respective writers. By maintaining this information, the system ensures that each writer's verification process leverages the most suitable deep architecture and classifier, optimizing accuracy and efficiency.

3.6 Verification

TP	FP
FN	TN

Once the writer-dependent characteristics are selected for each writer and stored in the knowledgebase, the authenticity of a test signature is decided as follows. Given a test signature S_t claimed to be of writer W_i. Initially, the information of write-dependent characteristics viz., deep architecture and writer-dependent classifier of writer W_i are retrieved from the knowledge base, which is stored during training. Then, the deep features of S_t are extracted by using the writer-dependent deep architecture of the writer W_i. During verification, the test signature is represented as a d-

dimensional feature vector. Then only the d features of the test signature are compared with the corresponding d features of reference signatures of the writer W_i. To compute the similarity between the test signature and to every other reference signature of the writer W_i, writer-dependent classifier of writer W_i is employed. Finally, the test signature sample is verified whether it is genuine or forgery.

4. Experimental setup and Results

In this section, we present the details of the datasets used, the experimental setup, and the experiments conducted along with the results obtained.

4.1 Datasets used

All experiments are conducted on two benchmark offline signature datasets: CEDAR [33] and MCYT [42]. The CEDAR dataset consists of signature samples from 55 writers, with each writer contributing 24 genuine signatures and 24 skilled forgeries, totaling 1,320 genuine and 1,320 forged signatures. The signature images in this dataset are available in grayscale format. The MCYT dataset comprises signatures from 75 writers, with each writer providing 15 genuine signatures and 15 skilled forgeries, resulting in a total of 2,250 signature samples. These datasets provide a diverse and challenging testbed for evaluating the effectiveness of the proposed writer-dependent signature verification approach.

4.2 Experimental setup

For experimentation, the signature samples of each writer are divided into two sets: a training set and a testing set. Both genuine and forged signatures are included in the training process to ensure the model learns distinguishing features effectively. The selection of training and testing samples is done randomly, with the percentage of training data varying from 30% to 70%. This variation allows for evaluating the model's performance under different levels of training data availability, ensuring a robust assessment of its effectiveness in writer-dependent offline signature verification. These experiments helped in understanding the impact of writer dependency on signature verification accuracy. To assess the accuracy of the proposed model, we used a confusion matrix, as illustrated in Figure 2. Using this matrix, various performance measures, such as the false acceptance rate (FAR), false rejection rate (FRR), average error rate (AER), and accuracy, were calculated.

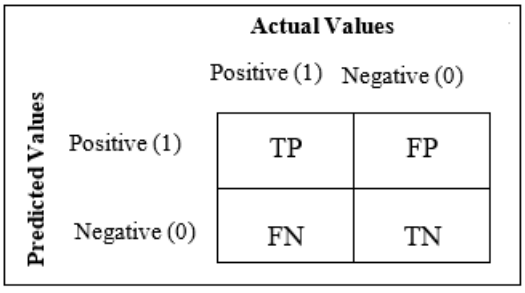

Figure 2: The structure of confusion matrix

In Figure 2, TP (True Positive) indicates that the signature sample is genuine and correctly predicted as genuine, while FP (False Positive) represents a forged signature that is incorrectly accepted as genuine. FN (False Negative) denotes a genuine signature that is mistakenly identified as a forgery. Finally, TN (True Negative) indicates a forged signature that is correctly recognized as a forgery.

FAR (False Accept Rate) is the proportion of forged signatures that are mistakenly accepted as genuine signatures and is calculated as

$$FAR = \frac{FP}{(FN+TP)} \quad (1)$$

The FRR (False Rejection Rate) is the proportion of genuine signature samples that are mistakenly rejected as forgeries and is determined as

$$FRR = \frac{FN}{(FN+TP)} \quad (2)$$

Both FAR and FRR are used to calculate AER (Average Error Rate), by using

$$AER = \frac{(FAR+FRR)}{2} \quad (3)$$

On the other hand, accuracy is a measure of correct predictions made by the system and is determined as

$$Accuracy = \frac{(TP+TN)}{(TP+FP+TN+FN)} \quad (4)$$

4.3 Results

During experimentation, we extracted deep features using seven pre-trained architectures: AlexNet, Inception-V2, Inception-V3, ResNet50, SqueezeNet, VGG-16, and VGG-19.

To evaluate the performance of our approach, we conducted the following five sets of experiments:

1) Using a common deep architecture across all writers.
2) Using writer-dependent deep architectures.
3) Using common deep features and a common classifier for all writers.
4) Using common deep features and writer-dependent classifiers.
5) Using writer-dependent deep features and writer-dependent classifiers.

4.3.1 Experimentation - 1

In this set of experiments, we utilized deep architectures for both feature extraction and classification. Initially, 1000 features were extracted from each pre-trained architecture for every writer. These extracted features were then passed through the dense layers of the respective deep architectures for classification. The results obtained from the CEDAR and MCYT datasets are presented in Table 1 and Table 2, respectively.

Table 1: Accuracy obtained using common pre-trained deep architectures on the CEDAR dataset

Pre-trained Deep Architecture	Percentage of Training Samples				
	30%	40%	50%	60%	70%
Alexnet	99.50	99.57	99.59	99.63	99.58
Inception-V2	99.58	99.67	99.68	99.74	99.69
Inception-V3	99.70	99.76	99.79	99.82	99.82
Resnet50	99.62	99.66	99.72	99.78	99.76
Squeezenet	99.20	99.24	99.33	99.31	99.27
VGG-16	99.50	99.69	99.60	99.65	99.62
VGG-19	99.51	99.44	99.56	99.61	99.61

Table 2: Accuracy obtained using common pre-trained deep architectures on the MCYT dataset

Pre-trained Deep Architecture	Percentage of Training Samples				
	30%	40%	50%	60%	70%
Alexnet	99.53	99.63	99.61	99.60	99.64
Inception-V2	99.50	99.63	99.61	99.68	99.64
Inception-V3	99.60	99.68	99.69	99.71	99.75
Resnet50	99.56	99.64	99.67	99.66	99.67
Squeezenet	99.35	99.33	99.40	99.29	99.38
VGG-16	99.53	99.61	99.59	99.65	99.64
VGG-19	99.51	99.58	99.64	99.66	99.65

4.3.2 Experimentation – 2

This set of experiments is based on writer-dependent deep architecture, where deep architectures are used for both feature extraction and verification. Initially, features are extracted for each writer using all available deep architectures. For each writer, n × 1000 features are obtained, where n denotes the number of training samples for that writer. The verification accuracy is then calculated for each deep architecture per writer. The deep architecture that yields the highest accuracy is selected as the writer-dependent deep architecture for that writer. The corresponding features extracted from this architecture are considered the writer-dependent features.

To verify the authenticity of a signature, the writer-dependent deep features of the test signature are compared with the reference features. This comparison is performed by passing the deep features through the dense layers of the corresponding writer-dependent deep architecture. The results obtained from the CEDAR and MCYT datasets using this approach are presented in Table 3. As indicated in Table 3, the CEDAR dataset achieves higher accuracy compared to the MCYT dataset.

Table 3: Accuracy obtained using writer-dependent deep architectures on the CEDAR, and MCYT datasets

Percentage of Training Samples	Dataset	
	CEDAR	MCYT
30%	99.79	99.68
40%	99.87	99.81
50%	99.88	99.83
60%	99.92	99.86
70%	99.93	99.87

4.3.3 Experimentation – 3

This set of experiments is conducted using common deep features and a common classifier across all writers. A single pre-trained deep architecture is employed for feature extraction across all writers. For classification, various conventional classifiers such as Support Vector Machine (SVM), Naive Bayes (NB), K-Nearest Neighbor (K-NN), Random Forest (RF), and Decision Tree (DT) are considered. The features extracted from each pre-trained deep architecture are individually fed into each of these classifiers. The results obtained from the CEDAR and MCYT datasets are presented in Table 4 and Table 5, respectively.

4.3.4 Experimentation – 4

In this stage, we conducted experiments using common deep features for all writers along with writer-dependent classifiers. PCA was applied to the deep features to reduce their dimensionality, as explained in Subsection 3.4. Through PCA, each feature vector was reduced from 1000 to 100 dimensions. These reduced feature vectors for each writer were then fed into various classifiers. For each writer, accuracy was computed using each classifier, and the classifier that achieved the highest accuracy was selected as the writer-dependent classifier, as described in Section 3.5.

During verification, deep features were extracted from the chosen deep architecture for each test signature sample, and matching was performed using the writer-dependent classifier corresponding to the claimed writer. The results obtained using common deep features and writer-dependent classifiers on the CEDAR and MCYT datasets are presented in Table 6 and Table 7, respectively.

Table 4: The accuracy obtained using common deep features and a common classifier on the CEDAR dataset

Pre-trained Deep Architecture	% of training samples	Support vector machines with different kernel functions				K- nearest neighbor	Decision Tree	Random Forest	Naive Bayesian
		Linear	Polynomial	Radial Basis Function (RBF)	Sigmoid				
Alexnet	30	99.62	99.22	99.34	99.04	99.58	98.62	99.11	99.55
	40	99.67	99.24	99.47	98.99	99.61	98.67	99.13	99.64
	50	99.7	99.27	99.53	98.94	99.64	98.61	99.14	99.69
	60	99.71	99.27	99.54	98.88	99.71	98.84	99.15	99.74
	70	99.73	99.31	99.6	98.83	99.67	98.83	99.18	99.68
Inception-V2	30	99.66	99.24	99.29	99.02	99.56	98.55	98.11	99.38
	40	99.71	99.29	99.43	98.96	99.63	98.59	99.12	99.49
	50	99.75	99.31	99.47	98.93	99.59	98.64	99.13	99.49
	60	99.76	99.33	99.52	98.9	99.66	98.74	99.15	99.41
	70	99.76	99.35	99.57	98.89	99.67	98.79	99.18	99.42
Inception-V3	30	99.71	99.21	99.37	99.04	99.65	98.65	98.1	99.45
	40	99.77	99.24	99.56	98.99	99.72	98.6	99.11	99.66
	50	99.76	99.27	99.61	98.62	99.74	98.76	99.12	99.68

	60	99.8	99.26	99.69	98.88	99.75	98.77	99.12	99.74
	70	99.84	99.34	99.74	98.81	99.76	98.74	99.16	99.76
Resnet50	30	99.67	99.19	99.32	99.03	99.6	98.45	99.1	99.44
	40	99.74	99.22	99.4	98.94	99.69	98.54	99.11	99.63
	50	99.75	99.25	99.57	98.94	99.71	98.58	99.11	99.69
	60	99.77	99.25	99.62	98.92	99.71	98.69	99.11	99.7
	70	99.79	99.29	99.7	98.83	98.76	98.71	99.14	99.72
Squeezenet	30	99.69	99.12	99.28	99.05	99.61	98.56	99.09	99.52
	40	99.7	99.13	99.45	98.99	99.65	98.56	99.1	99.64
	50	99.78	99.21	99.51	98.99	99.68	98.68	99.11	99.68
	60	99.76	99.17	99.56	98.97	99.69	98.71	99.11	99.74
	70	99.8	99.19	99.63	98.91	99.74	98.76	99.12	99.77
VGG-16	30	99.6	99.19	99.29	99.04	99.54	98.35	99.1	99.42
	40	99.69	99.21	99.4	98.98	99.61	98.34	98.11	99.61
	50	99.71	99.23	99.46	98.96	99.64	98.57	99.11	99.67
	60	99.71	99.26	99.5	98.93	99.61	98.48	98.12	99.69
	70	99.72	99.28	99.57	98.86	99.69	98.56	99.12	99.69
VGG-19	30	99.56	99.14	99.3	99.04	99.53	98.23	99.1	99.39
	40	99.62	99.2	99.4	99.01	99.56	98.41	99.11	99.6
	50	99.63	99.2	99.46	98.94	99.59	98.52	99.11	99.66
	60	99.68	99.22	99.48	98.87	99.65	98.44	99.11	99.69
	70	99.7	99.24	99.57	98.89	99.63	98.61	99.12	99.71

Table 5: The accuracy obtained using common deep features and a common classifier on the MCYT dataset

Pre-trained Deep Architecture	% of Training Samples	Support vector machines with different kernel functions				k nearest neighbor	Decision Tree	Random Forest	Naive Bayesian
		Linear	Polynomial	Radial Basis Function (RBF)	Sigmoid				
Alexnet	30	99.61	99.36	99.37	99.33	99.59	98.76	99.33	99.38
	40	99.69	99.39	99.43	99.31	99.65	98.78	99.34	99.56
	50	99.70	99.39	99.50	99.28	99.65	98.94	99.34	99.64
	60	99.74	99.41	99.52	99.27	99.71	98.85	99.35	99.71
	70	99.74	99.43	99.56	99.26	99.73	98.98	99.35	99.80
Inception-V2	30	99.56	99.37	99.35	99.33	99.49	98.58	99.33	99.36
	40	99.64	99.38	99.37	99.31	99.56	98.69	99.34	99.50
	50	99.69	99.41	99.43	99.28	99.60	98.83	99.34	99.59
	60	99.71	99.42	99.44	99.27	99.63	98.81	99.34	99.58
	70	99.74	99.38	99.48	99.25	99.56	98.81	99.34	99.61
Inception-V3	30	99.65	99.36	99.35	99.33	99.62	98.76	99.33	99.36
	40	99.68	99.38	99.41	99.30	99.66	98.80	99.34	99.47
	50	99.75	99.39	99.48	99.30	99.69	98.79	99.34	99.61
	60	99.72	99.38	99.52	99.28	99.71	98.84	99.34	99.65
	70	99.75	99.38	99.55	99.27	99.75	98.97	99.34	99.71
Resnet50	30	99.61	99.36	99.36	99.33	99.58	98.52	99.33	99.36
	40	99.69	99.40	99.42	99.32	99.64	98.52	99.34	99.49
	50	99.73	99.41	99.48	99.28	99.71	98.71	99.34	99.61
	60	99.74	99.41	99.53	99.27	99.72	98.65	99.34	99.68
	70	99.75	99.40	99.57	99.26	99.74	98.75	99.34	99.73
Squeezenet	30	99.66	99.24	99.38	99.33	99.58	98.76	99.34	99.36
	40	99.68	99.24	99.41	99.31	99.62	98.72	99.34	99.50
	50	99.74	99.24	99.46	99.28	99.68	98.86	99.34	99.65
	60	99.75	99.32	99.48	99.27	99.69	98.91	99.35	99.67
	70	99.79	99.22	99.52	99.27	99.71	98.93	99.35	99.71
VGG-16	30	99.56	99.35	99.34	99.32	99.53	98.30	99.33	99.34
	40	99.61	99.38	99.38	99.30	99.59	98.53	99.33	99.45
	50	99.64	99.39	99.44	99.29	99.61	98.67	99.33	99.61
	60	99.65	99.39	99.47	99.25	99.65	98.69	99.34	99.63
	70	99.68	99.40	99.49	99.26	99.63	98.78	99.34	99.67
VGG-19	30	99.56	99.36	99.36	99.32	99.52	98.65	99.33	99.35
	40	99.61	99.37	99.38	99.31	99.57	98.57	99.34	99.45
	50	99.64	99.37	99.43	99.27	99.59	98.74	99.33	99.56
	60	99.67	99.41	99.45	99.27	99.62	98.70	99.34	99.62
	70	99.70	99.39	99.47	99.23	99.62	98.66	99.34	99.68

Table 6: Accuracy obtained using common deep features and writer-dependent classifiers on the CEDAR dataset

Pre-trained Deep Architecture	Percentage of Training Samples				
	30%	40%	50%	60%	70%
Alexnet	99.69	99.76	99.77	99.81	99.81
Inception-V2	99.70	99.77	99.79	99.81	99.81
Inception-V3	99.74	99.82	99.83	99.87	99.9
Resnet50	99.71	99.79	99.81	99.83	99.88
Squeezenet	99.74	99.79	99.82	99.82	99.88
VGG-16	99.64	99.76	99.78	99.79	99.83
VGG-19	99.62	99.7	99.75	99.77	99.83

Table 7: Accuracy obtained using common deep features and writer-dependent classifiers on the MCYT dataset

Pre-trained Deep Architecture	Percentage of Training Samples				
	30%	40%	50%	60%	70%
Alexnet	99.66	99.74	99.66	99.81	99.81
Inception-V2	99.60	99.68	99.74	99.76	99.78
Inception-V3	99.69	99.73	99.80	99.79	99.81
Resnet50	99.67	99.74	99.80	99.81	99.82
Squeezenet	99.69	99.71	99.79	99.80	99.83
VGG-16	99.63	99.68	99.73	99.74	99.75
VGG-19	99.62	99.68	99.70	99.74	99.76

4.3.5 Experimentation - 5

First, we identify the writer-dependent architecture for all the writers in the system, as discussed in Section 3.3. After selecting a writer-dependent deep architecture for each writer, deep features are extracted. These features are then reduced from 1000 to 100 using PCA, as described in Subsection 3.4. The procedure explained in Section 3.5 is used to determine the writer-dependent classifier. Finally, verification is performed using the writer-dependent features and writer-dependent classifiers. The results are presented in Table. 8.

Table 8: Results obtained using writer-dependent deep features and writer-dependent classifiers on the CEDAR, and MCYT datasets

Percentage of Training Samples	Dataset	
	CEDAR	MCYT
30%	99.79	99.74
40%	99.87	99.84
50%	99.88	99.86
60%	99.92	99.89
70%	99.93	99.90

The accuracy achieved using writer-dependent deep features and writer-dependent classifiers outperforms that of other approaches. This underscores the importance of incorporating writer dependency at both the architectural and classifier levels to achieve optimal verification performance. The best results obtained from Experimentation-1 to Experimentation-5 on the CEDAR, and MCYT datasets are given in Table 9.

Table 9: The best results obtained from Experimentation-1 to Experimentation-5 on the CEDAR, and MCYT datasets

Experimentation Details	Datasets	
	CEDAR	MCYT
With a common deep architecture across all writers	99.82	99.75
With writer dependent deep architectures	99.93	99.87
With a common deep architecture and a common classifier for all writers	99.84	99.75
With a common deep architecture and writer dependent classifiers	99.90	99.83
With writer dependent deep architectures and writer dependent classifiers	**99.93**	**99.89**

5. Comparative Analysis

In this section, we compare the verification performance of the proposed approach with state-of-the-art methods in terms of accuracy. Table 10 presents the accuracy of the proposed model based on writer-dependent deep architecture and writer-dependent classifiers on the CEDAR dataset, alongside other existing models.

Table 10: Performance Comparison of Offline Signature Verification Approaches on the CEDAR Dataset

Model	Accuracy (%)
[46]	93.25
[35]	94.10
[12]	94.50
[51]	95.06
[18]	95.31
[52]	95.33
[23]	95.37
[49]	96.46
[62]	97.24
Proposed (With writer-dependent deep architecture and writer-dependent classifier)	99.93

Table 11 presents the results obtained on the MCYT dataset using the proposed writer-dependent deep architecture and writer-dependent classifier, along with a comparison to existing approaches.

Table 11: Comparison of the performance of various offline signature verification approaches on the MCYT dataset

Model	Accuracy (%)
[20]	87.56
[29]	88.49
[12]	90.74
[52]	94.04
[39]	94.15
[51]	94.54
[59]	97.42
[30]	98.93
[18]	99.66
Proposed (With writer-dependent deep architecture and writer-dependent classifier)	99.90

Tables 10 and 11 demonstrate that the proposed model, which incorporates writer-dependent characteristics, achieves higher accuracy compared to other existing models.

6. Conclusion

This work highlights the importance of writer-dependent characteristics in offline signature verification using deep learning approaches. It introduces the concept of writer-dependent deep architectures. The proposed method follows

a two-stage approach: (i) selecting a writer-specific deep architecture and (ii) selecting a writer-dependent classifier for verifying the claimed signature. Experiments are conducted on two offline signature datasets: CEDAR, and MCYT. The results demonstrate a significant improvement in verification accuracy by incorporating writer dependency at both the deep feature and classifier levels.

7. Future Work

In future, we can dynamically exploit the writer-dependent characteristics by applying deep learning.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Abdelrahaman, A. A. A., & Abdallah, M. E. A. (2013). K-nearest neighbor classifier for signature verification system. 2013 International Conference on Computing, Electrical and Electronic Engineering (ICCEEE), 58–62.

[2] Al-Maqaleh, B. M., & Musleh, A. M. Q. (2015). An efficient offline signature verification system using local features. International Journal of Computer Applications, 131(10).

[3] Alaei, A., Pal, S., Pal, U., & Blumenstein, M. (2017). An Efficient Signature Verification Method Based on an Interval Symbolic Representation and a Fuzzy Similarity Measure. IEEE Transactions on Information Forensics and Security, 12(10), 2360–2372.

[4] Alsuhimat, F. M., & Mohamad, F. S. (2023). Offline signature verification using long short-term memory and histogram orientation gradient. Bulletin of Electrical Engineering and Informatics, 12(1), 283–292.

[5] Alvarez, G., & Bryant, M. (2016). Offline Signature Verification with Convolutional Neural Networks. 8.

[6] Alvarez, G., Bryant, M., Sheffer, B., & Bryant, M. (2016). Offline signature verification with convolutional neural networks. Technical Report, Stanford University, 8.

[7] Arab, N., Nemmour, H., & Chibani, Y. (2023). A new synthetic feature generation scheme based on artificial immune systems for robust offline signature verification. Expert Systems with Applications, 213, 119306.

[8] Avola, D., Bigdello, M. J., Cinque, L., Fagioli, A., & Marini, M. R. (2021). R-SigNet: Reduced space writer-independent feature learning for offline writer-dependent signature verification. Pattern Recognition Letters, 150, 189–196.

[9] Baltzakis, H., & Papamarkos, N. (2001). New signature verification technique based on a two-stage neural network classifier. Engineering Applications of Artificial Intelligence, 14(1), 95–103.

[10] Batista, L., Granger, E., & Sabourin, R. (2012). Dynamic selection of generative-discriminative ensembles for off-line signature verification. Pattern Recognition, 45(4), 1326–1340.

[11] Batool, F. E., Attique, M., Sharif, M., Javed, K., Nazir, M., Abbasi, A. A., Iqbal, Z., & Riaz, N. (2020). Offline signature verification system: a novel technique of fusion of GLCM and geometric features using SVM. Multimedia Tools and Applications, 1–20.

[12] Bhunia, A. K., Alaei, A., & Roy, P. P. (2019). Signature verification approach using fusion of hybrid texture features. Neural Computing and Applications, 31(12), 8737–8748.

[13] Bouamra, W., Djeddi, C., Nini, B., Diaz, M., & Siddiqi, I. (2018). Towards the design of an offline signature verifier based on a small number of genuine samples for training. Expert Systems with Applications, 107, 182–195. https://doi.org/10.1016/j.eswa.2018.04.035

[14] Dey, S., Dutta, A., Toledo, J. I., Ghosh, S. K., Llados, J., & Pal, U. (2017). SigNet: Convolutional Siamese Network for Writer Independent Offline Signature Verification. 1, 1–7. http://arxiv.org/abs/1707.02131

[15] Diaz, M., Ferrer, M. A., Impedovo, D., Malik, M. I., Pirlo, G., & Plamondon, R. (2019). A perspective analysis of handwritten signature technology. ACM Computing Surveys, 51(6).

[16] Engin, D., Kantarci, A., Arslan, S., & Ekenel, H. K. (2020). Offline signature verification on real-world documents. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 808–809.

[17] Ferrer, M. A., Alonso, J. B., & Travieso, C. M. (2005). Offline geometric parameters for automatic signature verification using fixed-point arithmetic. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(6), 993–997.

[18] Ghosh, R. (2021). A Recurrent Neural Network based deep learning model for offline signature verification and recognition system. Expert Systems with Applications, 168, 114249.

[19] Goon, L. W., & Eng, S. K. (2021). Offline Signature Verification System Using SVM Classifier with Image Pre-processing Steps and SURF Algorithm. Journal of Physics: Conference Series, 2107(1), 12069.

[20] Hafemann, L. G., Oliveira, L. S., & Sabourin, R. (2018). Fixed-sized representation learning from offline handwritten signatures of different sizes. International Journal on Document Analysis and Recognition, 21(3), 219–232. https://doi.org/10.1007/s10032-018-0301-6

[21] Hafemann, L. G., Sabourin, R., & Oliveira, L. S. (2016a). Analyzing features learned for Offline Signature Verification using Deep CNNs. Proceedings - International Conference on Pattern Recognition, 0, 2989–2994. https://doi.org/10.1109/ICPR.2016.7900092

[22] Hafemann, L. G., Sabourin, R., & Oliveira, L. S. (2016b). Writer-independent feature learning for Offline Signature Verification using Deep Convolutional Neural Networks. Proceedings of the International Joint Conference on Neural Networks, 2016-Octob, 2576–2583. https://doi.org/10.1109/IJCNN.2016.7727521

[23] Hafemann, L. G., Sabourin, R., & Oliveira, L. S. (2017). Learning features for offline handwritten signature verification using deep convolutional neural networks. Pattern Recognition, 70, 163–176.

[24] Hanmandlu, M., Yusof, M. H. M., & Madasu, V. K. (2005). Off-line signature verification and forgery detection using fuzzy modeling. Pattern Recognition, 38(3), 341–356. https://doi.org/10.1016/j.patcog.2004.05.015
[25] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770–778.
[26] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. ArXiv Preprint ArXiv:1602.07360.
[27] Impedovo, D., & Pirlo, G. (2008). Automatic signature verification: The state of the art. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 38(5), 609–635. https://doi.org/10.1109/TSMCC.2008.923866
[28] Jadhav, T. (2019). Handwritten signature verification using local binary pattern features and KNN. Int. Res. J. Eng. Technol.(IRJET), 6(4), 579–586.
[29] Jagtap, A. B., Sawat, D. D., Hegadi, R. S., & Hegadi, R. S. (2020). Verification of genuine and forged offline signatures using Siamese Neural Network (SNN). Multimedia Tools and Applications, 79(47–48), 35109–35123.
[30] Jain, A., Singh, S. K., & Singh, K. P. (2020). Handwritten signature verification using shallow convolutional neural network. Multimedia Tools and Applications, 79(27–28), 19993–20018. https:
[31] Jain, A., Singh, S. K., & Singh, K. P. (2021). Signature verification using geometrical features and artificial neural network classifier. Neural Computing and Applications, 33(12), 6999–7010. https://doi.org/10.1007/s00521-020-05473-7
[32] Justino, E. J. R., Bortolozzi, F., & Sabourin, R. (2001). Off-line signature verification using HMM for Random, simple and skilled forgeries. Proceedings of the International Conference on Document Analysis and Recognition, ICDAR, 2001-Janua(c), 1031–1034.
[33] Kalera, M. K., Srihari, S., & Xu, A. (2004). Offline Signature Verification and Identification. International Journal of Pattern Recognition and Artificial Intelligence, 18(7), 1339–1360.
[34] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84–90.
[35] Maergner, P., Pondenkandath, V., Alberti, M., Liwicki, M., Riesen, K., Ingold, R., & Fischer, A. (2019). Combining graph edit distance and triplet networks for offline signature verification. Pattern Recognition Letters, 125, 527–533. https://doi.org/10.1016/j.patrec.2019.06.024
[36] Manjunatha, K. S., Annapurna, H., & Guru, D. S. (2019). Offline signature verification: An approach based on user-dependent features and classifiers. In Lecture Notes in Networks and Systems (Vol. 43). Springer Singapore. https://doi.org/10.1007/978-981-13-2514-4_20
[37] Manjunatha, K. S., Manjunath, S., Guru, D. S., & Somashekara, M. T. (2016). Online signature verification based on writer dependent features and classifiers. Pattern Recognition Letters, 80, 129–136. https://doi.org/10.1016/j.patrec.2016.06.016
[38] Manna, S., Chattopadhyay, S., Bhattacharya, S., & Pal, U. (2022). SWIS: Self-Supervised Representation Learning for Writer Independent Offline Signature Verification. 1, 1411–1415. https://doi.org/10.1109/icip46576.2022.9897562
[39] Masoudnia, S., Mersa, O., Araabi, B. N., Vahabie, A. H., Sadeghi, M. A., & Ahmadabadi, M. N. (2019). Multi-representational learning for Offline Signature Verification using Multi-Loss Snapshot Ensemble of CNNs. Expert Systems with Applications, 133, 317–330. https://doi.org/10.1016/j.eswa.2019.03.040
[40] Odeh, S. M., & Khalil, M. (2011). Off-line signature verification and recognition: Neural network approach. INISTA 2011 - 2011 International Symposium on INnovations in Intelligent SysTems and Applications, 34–38. https://doi.org/10.1109/INISTA.2011.5946065
[41] Okawa, M. (2018). From BoVW to VLAD with KAZE features: Offline signature verification considering cognitive processes of forensic experts. Pattern Recognition Letters, 113, 75–82. https://doi.org/10.1016/j.patrec.2018.05.019
[42] Ortega-Garcia, J., Fierrez-Aguilar, J., Simon, D., Gonzalez, J., Faundez-Zanuy, M., Espinosa, V., Satue, A., Hernaez, I., Igarza, J.-J., & Vivaracho, C. (2003). MCYT baseline corpus: a bimodal biometric database. IEE Proceedings-Vision, Image and Signal Processing, 150(6), 395–401.
[43] Otsu, N. (1979). A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics, 9(1), 62–66.
[44] Parcham, E., Ilbeygi, M., & Amini, M. (2021). CBCapsNet: A novel writer-independent offline signature verification model using a CNN-based architecture and capsule neural networks. Expert Systems with Applications, 185(July), 115649. https://doi.org/10.1016/j.eswa.2021.115649
[45] Parziale, A., Diaz, M., Ferrer, M. A., & Marcelli, A. (2019). SM-DTW: Stability Modulated Dynamic Time Warping for signature verification. Pattern Recognition Letters, 121, 113–122. https://doi.org/10.1016/j.patrec.2018.07.029
[46] Ren, J.-X., Xiong, Y.-J., Zhan, H., & Huang, B. (2023). 2C2S: A two-channel and two-stream transformer based framework for offline signature verification. Engineering Applications of Artificial Intelligence, 118, 105639.
[47] Ruiz-Del-Solar, J., Devia, C., Loncomilla, P., & Concha, F. (2008). Offline signature verification using local interest points and descriptors. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5197 LNCS, 22–29.
[48] Ruiz, V., Linares, I., Sanchez, A., & Velez, J. F. (2020). Off-line handwritten signature verification using compositional synthetic generation of signatures and Siamese Neural Networks. Neurocomputing, 374(xxxx), 30–41. https://doi.org/10.1016/j.neucom.2019.09.041
[49] Serdouk, Y., Nemmour, H., & Chibani, Y. (2016). New off-line Handwritten Signature Verification method based on Artificial Immune Recognition System. Expert

Systems with Applications, 51, 186–194. https://doi.org/10.1016/j.eswa.2016.01.001
[50] Shah, A. S., Khan, M. N. A., Subhan, F., Fayaz, M., & Shah, A. (2016). An Offline Signature Verification Technique Using Pixels Intensity Levels. International Journal of Signal Processing, Image Processing and Pattern Recognition, 9(8), 205–222.
[51] Shariatmadari, S., Emadi, S., & Akbari, Y. (2019). Patch-based offline signature verification using one-class hierarchical deep learning. International Journal on Document Analysis and Recognition, 22(4), 375–385. https://doi.org/10.1007/s10032-019-00331-2
[52] Sharif, M., Khan, M. A., Faisal, M., Yasmin, M., & Fernandes, S. L. (2020). A framework for offline signature verification system: Best features selection approach. Pattern Recognition Letters, 139(February), 50–59. https://doi.org/10.1016/j.patrec.2018.01.021
[53] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 1–14.
[54] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2818–2826.
[55] Tahir, N. M. T., Ausat, A. N., Bature, U. I., Abubakar, K. A., & Gambo, I. (2021). Off-line Handwritten Signature Verification System: Artificial Neural Network Approach. International Journal of Intelligent Systems and Applications, 13(1), 45–57.
[56] Wang, Z., & Zhang, D. (1999). Progressive switching median filter for the removal of impulse noise from highly corrupted images. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 46(1), 78–80.
[57] Wen, J., Fang, B., Tang, Y. Y., & Zhang, T. P. (2009). Model-based signature verification with rotation invariant features. Pattern Recognition, 42(7), 1458–1466.
[58] Xing, Z.-J., Yin, F., Wu, Y.-C., & Liu, C.-L. (2018). Offline signature verification using convolution Siamese network. Proc.SPIE, 10615, 106151I. https://doi.org/10.1117/12.2303380
[59] Yapıcı, M. M., Tekerek, A., & Topaloğlu, N. (2021). Deep learning-based data augmentation method and signature verification system for offline handwritten signature. Pattern Analysis and Applications, 24(1), 165–179. https://doi.org/10.1007/s10044-020-00912-6
[60] Zhang, Z., Liu, X., & Cui, Y. (2016a). Multi-phase offline signature verification system using deep convolutional generative adversarial networks. 2016 9th International Symposium on Computational Intelligence and Design (ISCID), 2, 103–107.
[61] Zhang, Z., Liu, X., & Cui, Y. (2016b). Multi-phase offline signature verification system using deep convolutional generative adversarial networks. Proceedings - 2016 9th International Symposium on Computational Intelligence and Design, ISCID 2016, 2, 103–107.
[62] Zheng, Y., Iwana, B. K., Malik, M. I., Ahmed, S., Ohyama, W., & Uchida, S. (2021). Learning the micro deformations by max-pooling for offline signature verification. Pattern Recognition, 118, 108008. https://doi.org/10.1016/j.patcog.2021.108008
[63] Zois, E. N., Alexandridis, A., & Economou, G. (2019). Writer independent offline signature verification based on asymmetric pixel relations and unrelated training-testing datasets. Expert Systems with Applications, 125, 14–32.
[64] Zois, E. N., Zervas, E., Tsourounis, D., & Economou, G. (2020). Sequential motif profiles and topological plots for offline signature verification. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 13245–13255.

Author Profile

D. S. Guru received his B.Sc., M.Sc., and Ph.D. degrees in Computer Science and Technology from the University of Mysore, Mysuru, India, in 1991, 1993, and 2000, respectively. He is currently a Senior Professor in the Department of Studies in Computer Science at the University of Mysore, India. He was a BOYSCAST Fellow and a visiting research scientist at Michigan State University. He has authored over 80 journal articles and more than 270 peer-reviewed conference papers at both international and national levels. His research interests include image retrieval, object recognition, shape analysis, sign language recognition, biometrics, and symbolic data analysis.

H. Annapurna received her B.Sc., MCA, and Ph.D. degrees in Computer Science from the University of Mysore, Mysuru, India, in 1996, 2000, and 2024, respectively. She is currently an Associate Professor in the Department of Computer Science at Yuvaraja's College, University of Mysore, Mysuru, Karnataka, India. Her research interests include image processing, pattern recognition, and biometrics.

K. S. Manjunatha received his B.Sc., M.Sc., and Ph.D. degrees in Computer Science from the University of Mysore, Mysuru, India, in 1991, 1993, and 2016, respectively. He is currently a Professor in the Department of Computer Science at Maharani's Science College for Women, Mysuru, Karnataka, India. His research interests include biometrics and symbolic data analysis.