Offline Signature Verification using Support Vector Machines and Multilayer Perceptron

Abstract

In recent years, along with the growing need for personal verification in many daily applications, the automatic signature verification is being considered with renewed interest. Signature verification is the process of authenticating and verifying the signature of a subject. The signature verification is a very efficient way of identifying forgeries. There are two major methods of signature verification. One is an *on-line* method to measure the sequential data such as handwriting and pen pressure with a special device. The other is the *off-line* method that uses an optical scanner to obtain handwriting data written on paper. Off-line systems work on the scanned image of a signature. In this project we describe a grid-based feature extraction technique that utilizes directional information extracted from the signature contour, i.e. the chain code histogram. The Gaussian Grid feature extraction technique was employed for feature extraction and Support Vector Machines (SVMs) were considered for classification. Once the image contour is obtained, it is divided into 12 x 12 zones, and by tracing the contours in each block the 4-direction chain code histogram is created for each block. And then for each block in the contour, a smoothening 2-D Gaussian filter is applied to generate the feature matrix on which SVM was applied.

Contents

1 Introduction

Biometrics consists of methods for uniquely recognizing humans based upon one or more intrinsic physical or behavioral traits. Handwritten signatures have long been established as the most widespread means of personal verification. Signatures are generally recognized as a legal means of verifying individual's identity by administrative and financial institutions. Considering the large number of signatures verified daily through visual inspection by people, the construction of a robust and accurate automatic signature verification system has many potential benefits for ensuring authenticity of signatures and reducing fraud and other crimes. Handwritten signature verification has been extensively studied in this decade. Its many applications include cheques, credit card validation, security systems, certificates, contracts, etc...

For several reasons, the task of verifying human signatures cannot be considered a trivial pattern recognition problem. It is a difficult problem because signature samples from the same person are similar but not identical. In addition, a person's signatures changes often radically during their lifetime. In fact, great variability can be observed in signatures according to country, age, time, habits, psychological or mental state, physical and practical conditions.

There are two major methods of signature verification: one is the On-line method to measure the sequential data such as handwriting and pen pressure with a special device. The other is an off-line method that uses an optical scanner to obtain handwriting data written on paper. Off-line data is a 2-D image of the signature. The Off-line signature verification is in a way better than the On-line model, the latter requires complex devices to process the signatures dynamically and to extract the required feature set from them which are then used to classify them as either genuine or forgeries. Processing Off-line is complex due to the absence of stable dynamic characteristics. Difficulty also lies in the fact that it is hard to segment signature strokes due to highly stylish and unconventional writing styles. Also, great variability can be observed in signatures according to country, age, time, habits, psychological or mental state, physical and practical conditions. All these coupled together cause large intra-personal variation. A robust system has to be designed which should not only be able to consider these factors but also detect various types of forgeries. The system should neither be too sensitive nor too coarse. It should have an acceptable trade-off between a low False Acceptance Rate (FAR), a low False Rejection Rate (FRR), A high True Acceptance Rate(TAR) and a high True Rejection Ratio(TRR).

The False Rejection Rate (FRR) is the percentage of genuine signatures rejected as false, and False Acceptance Rate (FAR) is the percentage of false signatures accepted as genuine. The FRR is the ratio of the number of genuine test signatures rejected to the total number of genuine test signatures submitted. The FAR is the ratio of the number of forgeries accepted to the total number of forgeries submitted.

Similarly, the True Acceptance Ratio is the percentage of genuine signatures accepted as genuine and the True Rejection Ratio is the percentage of forged signatures rejected as false. TAR is the ratio of number of genuine test signatures accepted to the total number of genuine test signatures submitted and TRR is the ratio of the number of forgeries rejected to the total number of forgeries submitted.

In signature verification systems, forgeries may be classified in two classes: Casual, skilled or traced. Casual forgeries are produced by only knowing the name of the person, whose signature is being forged, with no previous knowledge of the appearance of the genuine signature. Casual forgeries are the most commonly found forgeries. In skilled forgeries the forger knows the signature very well and has practiced the simulation process. Therefore the skilled forgery is very similar to the genuine signature.

In this project we consider the verification of both Kannada and English offline signatures. For the Kannada signatures there was no specific database, so we had to take the signatures by our own manually from 19 persons. For the English signatures, we use an already created database, therefore there was no need for us to create the database manually.

We approach the problem in two steps. Initially a set 20 of signatures are obtained from the subject and fed to the system. We have chosen 19 subjects for the formation of our database. Each of the subjects has provided 20 sample signatures in Kannada language. A standard database containing 20 signatures per person for 55 persons is used for verifying English signatures. 20 forgeries of the signatures of each person are collected in both the language. These signatures are pre-processed, and the *Gaussian Grid Feature Extraction* technique is used for extracting the various features of the signatures. A contour representation of the image is first developed and on this representation the Gaussian Grid Feature Extraction technique is applied.

First, the contour of the image is divided into *m x n* zones. Then each of the blocks are traced in all the eight directions, the 4-direction chain code histogram is generated. Every step from a pixel to its adjacent four direction (Horizantal, Vertical, Left-Diagonal and Right-Diagonal) are counted and four matrices *H, V, L and R* are created. Each of these matrices is of dimension *m x n* respectively. Over these four matrices obtained, a Gaussian smoothing filter is applied. Then the values in each of the matrices are adjusted by dividing its value by the maximum of each of the four matrices. From these four matrices two more matrices called *addmod* and *mulmod* are obtained. Finally, the feature vector is formed by combining all the six matrices *Horizontal, Vertical, Left Diagonal, Right Diagonal, addmod* and *mulmod.*

After creating a feature matrix for all the signatures of all the persons, the next step is to train the signatures of each persons and then testing the signatures to find whether they are genuine or forged, and finally to find the False Acceptance Ratio(FAR), False Rejection Ratio(FRR), True Acceptance Ratio(TRR) and True Acceptance Ratio(TAR).

Two different approaches are used for training and testing of signatures. One is the use of **Support Vector Machines (SVM)**.

Support Vector Machine is a classification tool. It takes a set of input data and predicts, for each given input, which of two possible classes forms the input. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

The other approach for training and testing of the signatures is the use of **Multi Layer Perceptrons (MLP).**

A **Multilayer Perceptron** (MLP) is an artificial neural network model that maps sets of input data onto a set of appropriate output. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one.

An **artificial neural network** or simply **neural network** is a mathematical model or computational model that is inspired by the structure and/or functional aspects of biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation.

In our project we make use of both SVM and MLP for training and testing.

In SVM approach, for training, a training matrix is formed by using the feature matrix. For every person, ten of his genuine signatures and ten of his forgeries are taken for training. And then for testing, the remaining signatures of each person are taken. The tested signatures belong to either one of the two groups- a genuine or a forgery, as trained in the training phase. A count is kept as to which signatures are accepted or rejected. Based on these count values, the values of FAR, FRR, TAR and TRR are calculated.

In the MLP approach, the training matrix is also constructed using the feature matrix. Each row of the feature matrix is put into a cell of the training matrix, and each row of the training matrix represents the training signatures of an individual, and there are as many rows as twice the number of individuals, one for the genuine and the other for forgeries. The parameters such as the number of epochs, the minimum gradient and type of function are set and then the trained matrix is trained. For testing, an array similar to that of each row of the training matrix is used for each signature that is to be tested. Similar to that of SVM, a count is kept as to which signatures are accepted or rejected. Based on these count values, the values of FAR, FRR, TAR and TRR are calculated.

2 Literature Survey

2.1 Biometrics:

Biometrics consists of methods for uniquely recognizing humans based upon one or more intrinsic physical or behavioural traits. In computer science, in particular, biometrics is used as a form of identity access management and access control. It is also used to identify individuals in groups that are under surveillance.

Biometric characteristics can be divided in two main classes

- **Physiological** are related to the shape of the body. Examples include, but are not limited to fingerprint, face recognition, DNA, Palm print, hand geometry, iris recognition, which has largely replaced retina, and odour/scent.
- **Behavioral** are related to the behavior of a person. Examples include, but are not limited to typing rhythm, gait and voice. Some researchers have coined the term **Behavio-metrics** for this class of biometrics.

2.2 Signature:

A signature is a person's name written as a form of identification in authorizing a check or document or concluding a letter.

2.3 Types of signatures

There are two types of signatures: on line signatures and off-line signatures.

On-line signatures are sequential data such as handwriting and various features like pen pressure are obtained by special devices, which play a vital role.
Off-line signatures are obtained through scanned copies of the handwritten signature on paper.

2.4 Forgeries

Forgery is the act of producing a copy of a document, signature, banknote or a work of art. A copied document, signature or a work note can also be called a forgery.

2.5 Types of forgeries

In signature verification systems, forgeries may be classified into two classes: casual, skilled or traced. *Casual forgeries*, also referred to as random forgeries are produced by only knowing the name of the person, whose signature is being forged, with no prior knowledge of the appearance of the genuine signature. In some cases, the forger may even use their own signature as a forgery. Casual forgeries are the most commonly found forgeries. In *skilled* or *traced forgeries,* the forger knows the signature very well and has practiced the simulation process. Therefore the skilled forgery is very similar to the genuine signature and some effective features in casual forgery verification becomes ineffective in skilled signature verification.

2.6 Types of signature verification

There are two types of signature verification- offline and online signature verification.

2.6.1 Offline Signature Verification

Offline signature verification is an important form of biometric identification that can be used for various purposes. Similar to other biometric measures, signatures have inherent variability and so pose a difficult recognition problem. In this paper, we explore a novel approach for reducing the variability associated with matching signatures based on curve warping. Existing techniques, such as the dynamic time warping approach, address this problem by minimizing a cost function through dynamic programming. This is by nature a 1-D optimization process that is possible when a 1-D parameterization of the curves is known. In this paper, we propose a novel approach for solving the curve correspondence problem that is not limited by the requirement of 1-D parameterization. The proposed approach utilizes particle dynamics and minimizes a cost function through an iterative solution of a system of first-order ordinary differential equations. The proposed approach is, therefore, capable of handling complex curves for which a simple parameterization is not available. The proposed approach is evaluated by measuring the precision and recall rates of documents based on signature similarity. To facilitate a realistic evaluation, the signature data we use were collected from real-world documents, spanning a period of several decades.

2.6.2 Online Signature Verification

Online signature verification is the process of verification of sequential data such as handwritings and pen pressure with a special device. It is more easier compared to offline signature verification and is generally more successful. But they require access to special processing devices when the signatures are produced.

2.7 Recent Contributions

2.7.1 OFFLINE SIGNATURE VERIFICATION

a) Offline Signature Verification Using Local Radon Transform and Support Vector Machines

-Vahid Kiani, Reza Pourreza, Hamid Reza Pourreza 2009, International Journal of Image Processing.

In this paper, we propose a new method for signature verification using local Radon Transform. The proposed method uses Radon Transform locally as feature extractor and Support Vector Machine (SVM) as classifier. The main idea of our method is using Radon Transform locally for line segments detection and feature extraction, against using it globally

b) Offline Signature Verification Using Critical Region Matching

-Abhay Bansal, Bharat Gupta, Gaurav Khandelwal, and Shampa 2009, International Journal of Signal Processing, Image Processing and Pattern

The paper is primarily focused on skilled forgery detection. It emphasizes on the extraction of the critical regions which are more prone to mistakes and matches them following a modular graph matching approach. The technique is robust and takes care of the inevitable intra- personal variations. The results show significant improvement over other approaches for detecting skilled forgery

c) Offline signature verification using structural feature correspondence

- Kai Huangand Hong Yan accepted 13 September 2001

This paper presents an off-line signature verification method using a model-based approach. In this method, statistical models are constructed for both pixel distribution and structural layout description. In addition to simple geometric handwriting features, it is proposed to use the directional frontier feature as a structural descriptor of the signature. The statistical verification algorithm based on the geometric handwriting feature is used to accept signatures which closely resemble the reference samples, and to reject random and less skilled forgeries

d) Offline signature verification using DTW

-A.Piyush Shankar and A.N.Rajagopalan 2007, Pattern Recognition Letters

In this paper, we propose a signature verification system based on Dynamic Time Warping (DTW). The method works by extracting the vertical projection feature from signature images and by comparing reference and probe feature templates using elastic matching. Modifications are made to the basic DTW algorithm to account for the stability of the various components of a signature. The basic DTW and the modified DTW methods are tested on a signature database of 100 people. The modified DTW algorithm, which incorporates stability, has an equal-error-rate of only 2% in comparison to 29% for the basic DTW method.

e) Offline Signature Verification: An Approach Based on Score Level Fusion

-H.N.Prakash and D.S.Guru 2010, International Journal of Computer Applications

In this paper, we propose a new approach for offline signature verification based on score level fusion of distance and orientation features of centroids. The proposed method employs symbolic representation of offline signatures using bi-interval valued feature vector. Distance and orientation features of centroids of offline signatures are used to form bi interval valued symbolic feature vector for representing signatures.

f) Performance analysis of the gradient feature and the modified direction feature for off-line signature verification

- V. Nguyen, Y. Kawazoe, T. Wakabayashi, U. Pal, and M. Blumenstein

the performance of two feature extraction techniques, the Modified Direction Feature (MDF) and the gradient feature are compared on the basis of similar experimental settings. In addition, the performance of Support Vector Machines (SVMs) and the squared Mahalanobis distance classifier employing the Gradient Feature are also compared and reported. Without using forgeries for training, experimental results indicated that an average error rate as low as 15.03% could be obtained using the gradient feature and SVMs.

2.7.2 ONLINE SIGNATURE VERIFICATION

a) A New On-Line Signature Verification Algorithm Using Variable Length Segmentation and Hidden Markov Models

- Mohammad M. Shafiei and Hamid R. Rabiee Document Analysis and Recognition, 2003.

In this paper, a new on-line handwritten signature verification system using Hidden Markov Model (HMM) is presented. The proposed system segments each signature based on its perceptually important points and then computes for each segment a number of features thatare scale and displacement invariant. The resulted sequence is then used for training an HMM to achieve signature verification. Our database includes 622 genuine signatures and 1010 forgery signatures that were collected from a population of 69 human subjects. Our verification system has achieved a false acceptance rate (FAR) of 4% and a false rejection rate (FRR) of 12%.

b) Online signature verification algorithm with a user-specific global-parameter fusion model

-Muramatsu, D.; Matsumoto, T. s<u>ystems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference</u>

Fusion is a promising strategy to improve performance in biometrics, and many fusion methods have been proposed. Most of them are user-generic fusion strategies, because generating user-specific strategies for each user is difficult. In this paper, we propose an online signature verification method using a user-specific global-parameter fusion model. The basic fusion model is a user-generic (global-parameter) fusion model, but by introducing a user-dependent mean vector, we can generate a user-specific fusion model. To evaluate the proposed algorithm, several experiments were performed by using three public databases. The proposed algorithm yielded equal error rates (EERs) of 4.0%, 8.6%, and 6.1% for the MCYT, SVC2004 task2, and My Idea databases, respectively.

c) Online slant signature algorithm analysis

-Rohayu Yusof Published in 2009.

A vector rule-based approach and analysis to on-line slant signature recognition algorithm is presented. Extracting features in signature is an intense area due to complex human behaviour, which is developed through repetition. Features such as direction, slant, baseline, pressure, speed and numbers of pen ups and downs are some of the dynamic information signature that can be extracted from an online method. This paper presents the variables involved in designing the algorithm for extracting the slant feature. Signature Extraction Features System (SEFS) is used to extract the slant features in signature automatically for analysis purposes. The system uses both local and global slant characteristics in extracting the feature. Local slant is the longest slant among the detected slant while the global slant represents the highest quantity of classified slant whether the slant are leftward, upright or rightward. Development and analysis are reported on a database comprises of 20 signatures from 20 subjects. The system is compared to human expert evaluation. The results demonstrate a competitive performance with 85% accuracy.

d) Online Signature Verification Algorithm Using Hill-Climbing Method.

-Daigo Muramatsu, <u>Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on</u>

Attacks using hill-climbing methods have been reported as a vulnerability of biometric authentication systems. In this paper, I propose a robust online signature verification algorithm against attacks using the hill-climbing method. Specifically, the attack considered in this paper is a hill-climbing forged data attack. Artificial forgeries are generated by using the hill-climbing method and the forgeries are input to a target system to be attacked. In order to generate a robust algorithm, I propose incorporating the hill-climbing method into an online signature verification algorithm. Preliminary experiments were performed using several online signature databases. The results show that the proposed algorithm improved the performance against this kind of attack. Equal Error Rate (EER) was improved from 88.3% to 1.2% when using a private database for evaluation.

e) Online Signature Verification Algorithm based on time sequence.
- Fangjun Lua, Shillang, INTERNATIONAL JOURNAL OF INFORMATION AND SYSTEMS SCIENCES 2005.

On-line handwritten signature verification is a topic in the field Of biometric authentication now. Time sequence match and result judgement are the key to this problem. This paper presents a time sequence match algorithm for handwritten signature verification. The algorithm is eventual to be examined by signature data from SVC 2004(First International Signature Verification Competition). The paper also introduces distance calculation formula based on weighting H1 module to calculate the differences between input signatures and genuine signatures.

. Parameter Features:

The below table shows some recent contributions towards the parameter features of a signature.

Parameters	Category	References
Total signature time duration	Online	R.S. Kashi et al., J.Lee et al, L.L.Lee et al, W. Nelson et al, T.Qu et al, W.S.Wijesoma et al
Pen-down time ratio	Online	R.S. Kashi et al, W. Nelson et al, W.S.Wijesoma et al
Cosine transform	Online	T.Matsuura and T.S.Yu
Random transform	Offline	J.Coetzar et al
Contour based	Offline	R.Bajaj and S.Chaudhury, H.Cardot et al, S.Chen and S.N.Srihari, M.A.Ferrer et al, F.Nouboud, F.Nouboudand R.Plamondon, I.Pavlidis et al, V.E.Ramesh and M.Narasimha Murty
Peripheral-based	Offline	B.Fang and Y.Y.Tang
Fractal transform	Offline	k.Huang and H.Yan, S.Mozattari et al
Smoothening based	Offline	B.Fang et al

3 System Requirements and Analysis

3.1 Hardware Specifications

1. Any Processor with a clock speed of >=2GHz

2. RAM more than 1GB.

3. High definition Scanner.

4. Storage capacity of more than 10GB.

3.2 Software Specifications

1. Windows 7/vista 32 bit/64 bit.

2. MATLAB 7.10.00 R2010a 32 bit/64 bit.

3. Graphic card.

4.GhostScript version 2.8 or higher

4 REQUIREMENT SPECIFICATION

4.1 FUNCTIONAL REQUIREMENTS

4.1.1 PREPROCESSING

Since a standard signature database is not available in the public domain for the Kannada signatures, we built our own database to test the effectiveness of our system. Our signature database comprises of 380 signatures collected from 19 people. Each individual was asked to provide 20 samples of his/her signature for training. We collected 380 samples of forgeries.

Figure 1: Signatures of individual persons on a4 paper

For the English signatures, a standard database, containing 20 signatures for each of the 55 persons was already available, so we made use of that database.

After extracting each of the signatures, the images have to be converted to grayscale as a preprocessing process:

Conversion of the image to GrayScale

The first pre-processing step is conversion of image into grayscale. This is done because in all later steps of the program we are using grayscale images only.

4.1.2 EXTRACTION

After converting the image to grayscale, we have to extract each of the 20 signatures for each of the signer and store them as individual images. This is done by the extraction phase.

In this phase we extract the entire 20 genuine signature and 20 forged signature of an individual from the scanned paper and store them as images in TIFF format in their respective folders. They are later used in other phases.

Some extracted signatures and their forgeries are shown below:

Figure 2: An extracted genuine signature

Figure 3: Its forgery, extracted

Figure 4: Another genuine signature

Figure 5: Forgery of Figure 4

4.1.3 CONTOUR REPRESENTATION OF THE IMAGE

After extracting each signature from the database, the next step is to obtain the contour plot representation of each of the signatures (A contour is an outline of the image). After the contours are created for each image, each of the contour representations are cropped to the boundaries of the signature so as to eliminate extra white spaces in the image. The resulting contour plot is stored in a TIFF image. The images below show contour plots of a few images:

Using ghostScript to save the contour image

The contour representation obtained is a plot. In MatLab, it is difficult to save plots to images. Hence , an external program called *ghostScript* was used to save the contour representation cropped to the boundaries of the signature. Ghostscript must be first configured properly before being used.

Figure 6: Contour plot of Figure 2

Figure 7: Contour plot of Figure 3

Figure 8: Contour plot of Figure 4

Figure 9: Contour plot of Figure 5

4.1.4 APPLYING THE CHAIN CODE HISTOGRAM

A **chain code** is a lossless compression algorithm for monochrome images. The basic principle of chain codes is to separately encode each connected component (a region of pixels with the same value) or a blob in the image. For each such region, a point on the boundary is selected and its coordinates are transmitted. The encoder then moves along the boundary of the region and, at each step, transmits a symbol representing the direction of this movement. This continues until the encoder returns to the starting position, at which point the blob has been completely described, and encoding continues with the next blob in the image. In this project we use the 4-directional chain code. The 8-direction code represents the eight neighbors of a pixel along four principal stroke directions: Horizontal, Vertical, Left diagonal and Right Diagonal. Each direction represents the percentage of respective direction in an individual's writing.

In this project, from the contour representation of a signature image, the Gaussian Grid feature extraction technique first divides this image into *m x n* zones. In the next step, the contours in each block are traced and a 4-direction chain code histogram is then generated and four matrices called H, V, L and R are generated for the four directions Horizontal, Vertical, Left Diagonal and Right Diagonal respectively, each of dimension m x n.

4.1.5 APPLYING THE GAUSSIAN FILTER

In this technique, a smoothening Gaussian filter[2] is applied on each of the four matrices H, V, L and R each of dimension *m x n*. The Gaussian filter is given by:

$$A_{ij}^{*} = \sum_{di=-\infty}^{\infty} \sum_{dj=-\infty}^{\infty} A_{i+di,j+dj} \frac{1}{2\pi\sigma^2} e^{-\frac{di^2+dj^2}{2\sigma^2}}$$

The value of each element of each matrix is adjusted by dividing its value by the maximum value of the four matrices:

$$A_{ij} = \frac{A_{ij}}{\max(Hxy, Vxy, Lxy, Rxy)}$$

From the two-matrix pairs horizontal (*H*) and vertical (*V*) matrices, left-diagonal (*L*) and right diagonal (*R*) matrices, two new matrices $\oplus$ and $\otimes$ are established using the following two equations:

$$\oplus_{ij} = \begin{cases} \dfrac{\max(H_{ij}, V_{ij})}{\min(H_{ij}, V_{ij})} & if \max(H_{ij}, V_{ij}) \neq 0 \\ 0 & if \max(H_{ij}, V_{ij}) = 0 \end{cases}$$

$$\otimes_{ij} = \begin{cases} \dfrac{\min(L_{ij}, R_{ij})}{\max(L_{ij}, R_{ij})} & if \max(L_{ij}, R_{ij}) \neq 0 \\ 0 & if \max(L_{ij}, R_{ij}) = 0 \end{cases}$$

Finally, the feature vector is formed by merging the six matrices H, V, L, R, $\oplus$ and $\otimes$. The dimension of the output feature matrix is *m x n x 6.*

There are as much rows in the feature matrix as the total number of signatures in the database, and each cell of the feature matrix contains another matrix of size *m x n x 6.*
Once the feature matrix is computed, the signatures are verified as to whether they are genuine or forged.

4.1.6 VERIFICATION OF SIGNATURES

The process of verifying the signatures comprise of two steps- training and testing. Two different approaches can be followed. The first one being the Support Vector Machine (SVM) approach and the second being the Multilayer Perceptrons (MLP) approach.

For both of the techniques, a process called training has to be performed. Training is the process where we use some of the signatures to train the system to make it distinguish between the genuine signatures and the forged ones for all of the signatures present in the system. For both SVM and MLP approaches, we build a training matrix from the feature matrix. The training matrix is built every time for each of the signatures, both genuine and forged. The training matrix is the same for both SVM and MLP approaches, but the training matrix for SVM is a column matrix where as for MLP the training matrix is the transpose of that of for SVM(a row matrix).

For SVM, the column matrix is of the dimensions *n x 1* where n is the total number of samples used for training, including both genuine and forged signatures. In our project, n takes the value 5, 10 or 15.

After constructing the training matrix, the matrix data is trained using different kernel functions. The experiment is repeated for different scaling factors and the results compared for each scaling facto. The signatures belong to either of the two groups, 0 or one. If a signature belongs to the group 0, it is accepted as a genuine. If the signature belongs to the group 1, then it is rejected as forgery.

In the case of MLP, the row matrix is of dimensions *m x 1* where m is the total number of samples used for training, including both genuine and forged signatures. In our project m takes the value 5 or 15.

After constructing the training matrix, the matrix data is trained for different values for the parameters like error goals, epochs, etc. The target values (0 or 1) specify the group the signature belongs to. The target matrix is a matrix containing 0’s and 1’s and having the same dimensions as the training matrix.

In the testing process, a matrix is constructed out of the data that is to be tested (the signature to be tested) of the same dimensions as the training matrix. A new matrix is constructed out of the values obtained for each signatures that are tested. If the value is closer to zero then it is considered accepted as genuine. If the value is closer to 1 then it is considered rejected as forgery.

4.1.7 ACCURACY FACTORS

FAR

Accepting the forgery signature thinking it is a genuine signature is called as False Acceptance Rate (FAR).

It is given by

$$FAR = \frac{Total\ number\ of\ Accepted\ forgeries}{Total\ number\ of\ tested\ forgeries} \times 100$$

FRR

Rejecting a signature even though it is genuine signature thinking it is a forgery signature is called as False Rejection Rate (FRR).

It is given by:

$$FRR = \frac{Total\ number\ of\ genuine\ rejected}{Total\ number\ of\ tested\ genuines} \times 100$$

TAR

Accepting a genuine signature as true is called as True Acceptance Rate (TAR).

It is given by:

$$TAR = \frac{Total\ number\ of\ genuine\ accepted}{Total\ number\ of\ tested\ genuines} \times 100$$

TAR

Rejecting a forged signature as false is called as True Rejection Rate (TRR).

It is given by:

$$TAR = \frac{Total\ number\ of\ forgeries\ rejected}{Total\ number\ of\ tested\ forgeries} \times 100$$

Generally, the FAR and FRR should be low and TAR and TRR should be high for a signature verification system.

4.2 NON-FUNCTIONAL REQUIREMENTS

- The signatures were collected on A4 sheets with boxes of size 2 cm · 8 cm, big enough to accommodate large signatures.
- An HP Scanjet 4400C was used to scan and digitize the images. The images were scanned at 500 dpi resolution and stored in bitmap format.
- The cutpic images were not properly extracted. Hence, they required some manual intervention in removal of the boxes which helped in pre-processing.
- The program is written in MATLAB platform. The programs files are stored with **.m** extension. The project is done using windows 7 operating system.
- The accuracy factors FAR, FRR and ERR should be as low as possible

4.3 System Design

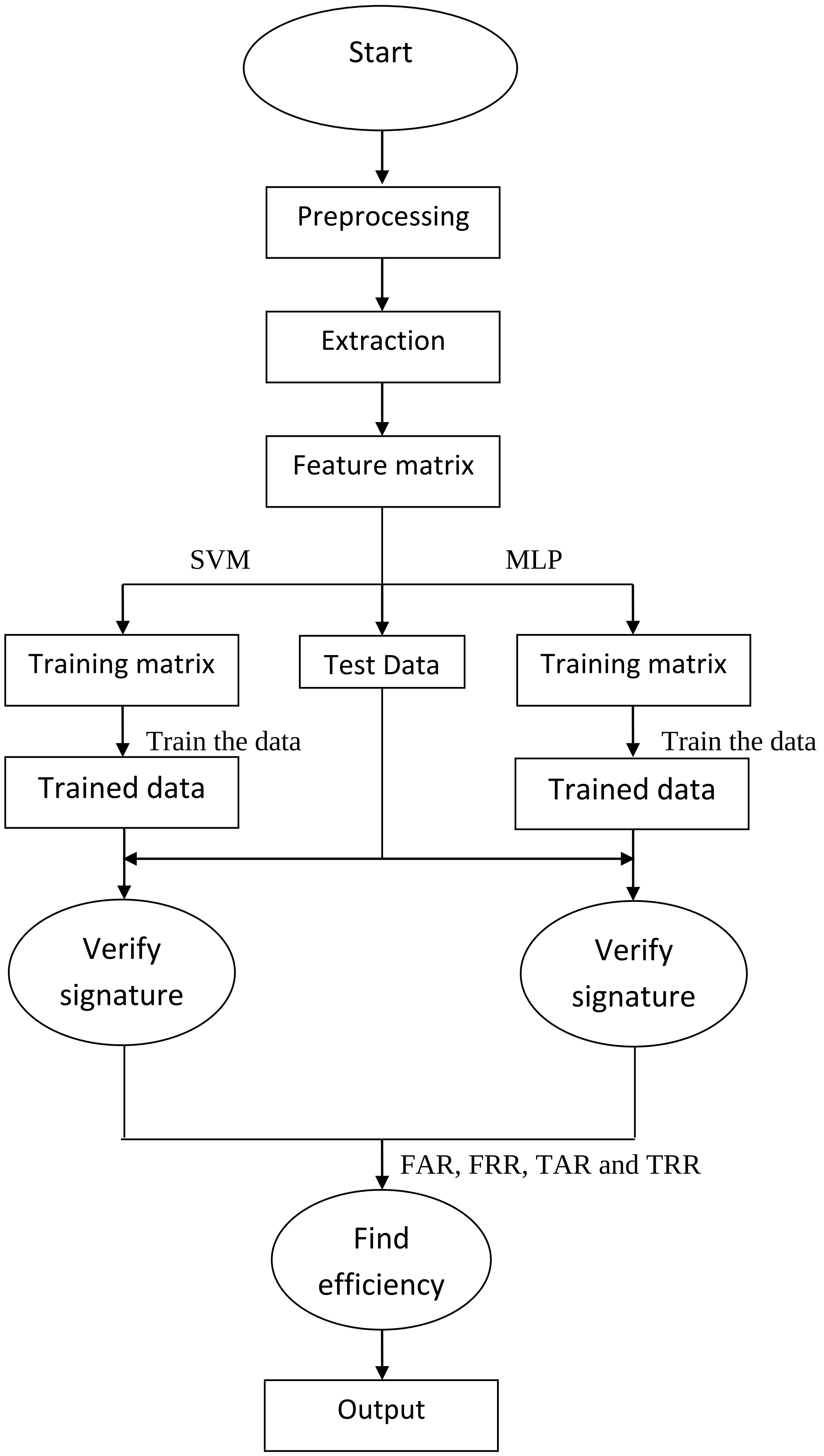

Figure 10: Flowchart of the approach used

The above flowchart is a pictorial representation of the modules that are used in the project. Each phase of the design explained in detail in the next chapter i.e. Implementation. The algorithms of these modules are provided in this chapter.

4.3.1 Algorithms

1. Extraction and pre processing of genuine and forged signatures.
2. Contour representation of the signatures.
3. Constructing feature matrix of the signatures.
4. Verification of signatures.
5. Calculation of efficiency.

Extraction and pre processing of genuine and forged signatures

Input: JPG image sheet containing the signatures.

Output: JPG images of the pre processed and cropped signatures.

```
For i=1 to 40
From the corresponding pixel value to the end value
Extract the signature
Name the signature
Convert the image to grayscale
Write the signature image to a file
Store the signature in the Genuine/Forged folder according to the image
End for
```

The signature images are extracted in this step and they are stored in 40 folders for 40 users.

Contour representation of signatures

Input: JPG image of extracted and pre processed signatures

Output: A contour representation of each of the signatures which are stored as images.

```
For j=1 to 40                    //40 persons
For k=1 to 19          //19 signature samples per person
Apply contour on the signature
Save the contour plot by using ghostScript
End for
End for
```

Constructing feature matrix of the signatures

Input: The contour plots of each signature of all persons

Output: The feature matrix of the signatures which is further used for verification

```
For i=1 to 40
For j=1 to 19
Find the size of the image.
Resize the image such that the length and width are multiples of 12. (12 x 12 x 6  feature matrix)
Convert the resulting matrix of the image into a cell array.
Define four matrices, H, V, L and R. and four variables hi, vi, li, ri, and four temporary variables h, v, l, r.
Trace the boundaries of the signature using 8 connectivity and store it into a variable.
Find the chain codes of the variable obtained from the previous step.
Based on the values of chain codes, increment the values of h, v, l or r.
Compute the values hi, vi, li and ri as h/n, v/n, l/n and r/n respectively.
If the values of hi, vi, li or ri are Nan, then set them to zero.
Append hi, vi, li and ri values to the matrices H, V, L and R respectively.
Reshape the H, V, L and R arrays into 12 x 12 matrices.
Apply a smoothening Gaussian filter on each of the four matrices.
Adjust the value of each element in each matrix by dividing it by the maximum value of the four matrices.
For the two matrix pairs H, V and L, R obtain two more matrices addmod and mulmod of same dimensions.
The feature vector is formed by merging all the six matrices H, V, L, R, addmod and mulmod. It is of dimensions m x n x 6.
```

Verification of signatures

Input: The feature matrix obtained in the previous step.

Output: An array containing the values 0 or 1 indicating the corresponding signatures in the training matrix genuine or forged.

There are two different methods we consider here for verification- the SVM method and MLP method.

SVM method

```
For i=1 to 40
If i greater than 11 or I greater than 20 and less than 31 then, copy the feature matrix values to the training cell array
Convert this cell array to a matrix
Append this matrix to training matrix
Create a matrix with 0 in the index of corresponding genuine signatures and 1 in the corresponding forged signature
Find the transpose of these matrices
```

Supply these 2 matrices to the training function as arguments
For i=1 to 40
If i greater than 11 and less than 21 and i greater than 30 then, copy the feature matrix values to the testing cell array
Convert this cell array to a matrix
Test this matrix and save the results

MLP method

For i=1 to 40
If i greater than 11 or I greater than 20 and less than 31 then, copy the feature matrix values to the training cell array
Convert this cell array to a matrix
Append this matrix to training matrix
Create a matrix with 0 in the index of corresponding genuine signatures and 1 in the corresponding forged signature
Supply these 2 matrices to the training function as arguments
For i=1 to 40
If i greater than 11 and less than 21 and i greater than 30 then, copy the feature matrix values to the testing cell array
Convert this cell array to a matrix
Test this matrix and save the results

---

5 Implementation

5.1 Database Creation

The creation of a database was the first step in the project as we were not working on any other available databases on the internet. First we had to draw a table with 2 columns and 10 rows with equidistant spaces for the people to sign in it. We made 19 people sign 20 times each, collecting a total of 19 sheets of signed data which made us create a database of 360 signatures in total. The forgeries were made by random people. We made 20 forgeries of each person.

Scanning these images was an issue as the scanned images were having small noise elements which caused a few problems while processing the image in later stages. As a result we had to manually clean some images in order to remove some noisy parts in the images.

As for the English signatures, the database was available already, containing 20 signatures each of 55 persons, a total of 1100 genuine signatures. 20 forged signatures were available for each person.

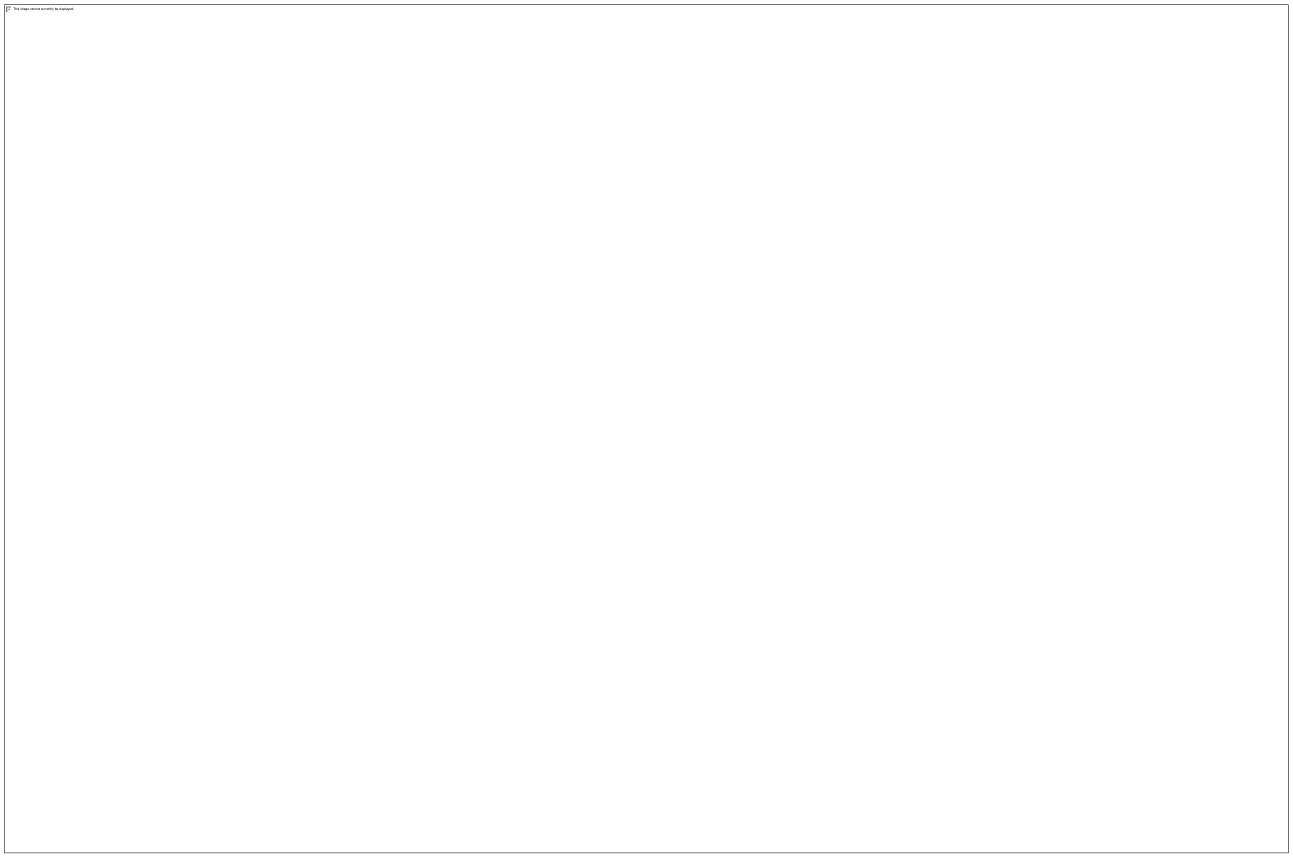

Figure 11: Genuine and forged signatures of a person

The above image has two parts, the first being the genuine signatures as collected by us in the stage of data collection and the second being the forged signatures. Like this the data from 19 users are collected and scanned using a high definition scanner to obtain a better image to process in the later stages. These collected images form our database and are used in the later stages of our project. As for the English signatures, the database was available already itself, so there was no need to create a new database.

5.2 Extraction of signatures

The next phase is the extraction of each signature in the images present in the database created earlier. The approach we chose was not suitable because the images of the signatures obtained had the lines segments in them which were in the scanned images for separation of each signature from the other. This was of major concern as we had to extract features from these obtained images and those line segments were causing problems in doing so.

Thus, we decided to remove those tables from the scanned images using a suitable technique, but later decided to do it manually using the MS paint software. This helped us in extracting the required signature image of a given subject in an easier manner. We made use of the pixel positions and decided a rectangular box which would fit all the signature images in the given image and extracted all the individual samples using a loop. An example image of the extracted signature is shown below. Both genuine and forged signatures are extracted in the same way.

Figure 12: Example of extracted image from the database

The genuine and forged signatures were stored together alternatively. i.e., if a genuine signature sheet had the name ‘IMG_0001’ then its forged signature sheet was the next. i.e., its name was ‘IMG_0002’. This was followed for all of the 19 genuine signatures.

The following MATLAB code was used to extract the genuine and forged Kannada signature images to the ‘signatures/IMG_00(i)’, and ‘signatures/IMG_00(j)’ folders respectively, where i and j were taking the values from 01 to 19 and were odd and even respectively.

```
%this program extracts all the genuine and forged signatures from the %each of
the 38 signature sheets, and store them into the folders of the %corresponding
name of the sheet they are extracted from.
for x=1:38
    f=strcat('IMG_00',num2str(x));
    fname=strcat(f,'.jpg');
    mkdir(f);
    im=imread(fname);
    im = im2double(im);
    f_makebw = @(I) im2bw(I.data, double(median(I.data(:)))/1.45);
    I0 = blockproc(im, [128 128], f_makebw);

    imf=imfinfo(fname);

    h=imf.Height;
    w=imf.Width;


    for i=0:9
        r=0;
        for j=(i*round(h/10)+1):((i+1)*round(h/10))
            if(j<=h)
                r=r+1;
                for c=1:w
                    if(c<round(w/2))
                        im1(r,c)=I0(j,c);
                    else
                        c1=c-round(w/2)+1;
                        im2(r,c1)=I0(j,c);
                    end
                end
            end
        end
        name1=strcat(f,'/',num2str(2*i),'.tiff');
        name2=strcat(f,'/',num2str(2*i+1),'.tiff');
        imwrite(im1,name1);
        imwrite(im2,name2);
    end
end
```

As the English signature was available, there was no need to extract each of the English signatures from the sheets.

5.3 Contour representation of images

After extracting each of the genuine and forged signatures, the next step is to find the contour representation of each of the signatures. In this step, we extract the contour representation of each of the signature extracted from the previous step. A contour is nothing but an image showing only the boundaries of an image. It is on the contour of the image we apply the chain code histogram to. The contour of a signature in the Figure 12 is shown below. Again, the contour representation is the same for both the genuine and forged signatures.

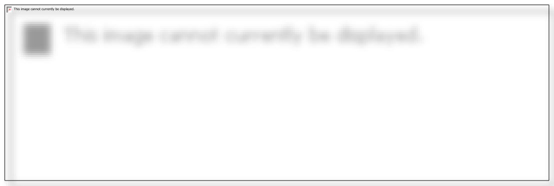
Figure13: Signature to which the contour is applied

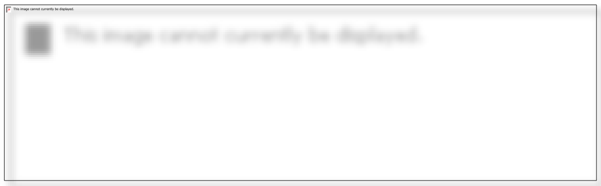
Figure 14: The contour plot of Fig 13

The contour of the image which is generated cannot be saved like a normal image in Matlab. To save the contour image, an external program called ‘ghostScript’ is used. After applying the contour to the image, we use a ghostScript function in Matlab called ‘savefig’. This function saves the contour plot by cropping it to its boundaries. The image file format and the type of image (grayscale, binary, etc) are specified by the user as parameters. In this program we use the ‘.tiff’ file format and the grayscale type of image.

The below Matlab code shows the process of creating contour plots for all the signatures. The contours of the signatures of a specific person are stored in the same folder as the person’s signatures are present, with the same name as the signature names. Only the format is changed from ‘.jpg’ to ‘.tiff’.

```
for e=1:38
      f=strcat('IMG_00',num2str(e),'/');
      f1=f;
      for z=0:19
            f=strcat(f1,num2str(z),'.tiff');
            I1=imread(f);
            x=[];
            y=[];
            img=imcontour(I1);
            set(gca,'xcolor',get(gcf,'color'));
            set(gca,'ycolor',get(gcf,'color'));
            set(gca,'ytick',[]);
            set(gca,'xtick',[]);
            name=strcat(f1,'contour',num2str(z));
            name1=strcat(f1,'contouru',num2str(z));
            savefig(name,'tiff','-gray')
      end
end
```

This shows how to create contour plots for the 38 Kannada signatures including genuine and forgeries. To apply the contour for the 110 English signatures (55 genuine and 55 forgery), all that has to be done is that the outer loop in the above code has to iterate from 1 to 110, instead of from 1 to 38.

5.4 Construction of the feature matrix

The next step after constructing the contour plot of each of the signatures is to construct the feature matrix from the contour plots of the signatures. First the length and width of each of the signature is made a multiple of 12. After this process, the contour image matrix is divided into 12 x 12 zones. Then the contours in each block are traced and the 4-direction chain code histogram of each block is created. Every step from a pixel to its adjacent one of the four directions (horizontal, vertical, left-diagonal, and right-diagonal) are counted. There are four matrices of size *12×12* for each direction, namely *hmat*, *vmat* , *lmat* and *rmat*. Then a smoothening Gaussian filter is applied on each of the four matrices, after which the values in each matrix are adjusted and two more matrices *addmod and mulmod* are constructed out of the four matrices.

The function *boundaries()* which traces the signature's boundaries is as given below.

```
function B=boundaries(BW,conn,dir)
if nargin<3

    dir='cw';
end
if nargin<2
    conn=8;
end
L=bwlabel(BW,conn);
numObjects=max(L(:));
if numObjects>0
    B={zeros(0,2)};
    B=repmat(B,numObjects,1);
else
    B={};
end
%Pad label matrix with zeros.This lets us write the boundary_following loop
%without worrying about going off the edge of the image.
Lp=padarray(L,[1 1],0,'both');
%Compute the linear indexing offsets to take us from a pixel to its
%neighbors.
M=size(Lp,1);%SIZE(X,1) returns the number of rows.

if conn==8
    %Order is N NE E SE S SW W NW.
    offsets=[-1,M-1,M,M+1,1,-M+1,-M,-M-1];
else
    %Order is N E S W.
    offsets=[-1,M,1,-M];
end
%next_search_direction_lut is a lookup table.Given the direction from pixel
%k to pixel k+1,what is the direction to start with when examining the
%neighborhood of pixel k+1?
if conn==8
    next_search_direction_lut=[8 8 2 2 4 4 6 6];
else
    next_search_direction_lut=[4 1 2 3];
end
%next_direction_lut is a lookup table.Given that we just looked at neighbor
%in a given direction,which neighbor do we look at next?
if conn==8
  next_direction_lut=[2 3 4 5 6 7 8 1];
else
  next_direction_lut=[2 3 4 1];
end
%Values used for marking the starting and boundary pixels.
START=-1;
BOUNDARY=-2;
```

```
%Initialize scratch space in which to record the boundary pixels as well as
%follow the boundary.
scratch=zeros(100,1);
%Find candidate starting locations for boundaries.
[rr,cc]=find((Lp(2:end-1,:)>0)&(Lp(1:end-2,:)==0));
rr=rr+1;
for k=1:length(rr)
    r=rr(k);
    c=cc(k);
    if (Lp(r,c)>0)&(Lp(r-1,c)==0)&isempty(B{Lp(r,c)})
        %We've found the start of the next boundary.Compute its linear
        %offset,record which boundary it is,mark it,and initialize the
        %counter for the number of boundary pixels.
        idx=(c-1)*size(Lp,1)+r;
        which=Lp(idx);
        scratch(1)=idx;
        Lp(idx)=START;
        numpixels=1;
        currentpixel=idx;
        initial_departure_direction=[];
        done=0;
        next_search_direction=2;
        while ~done
            %Find the next boundary pixel.
            direction=next_search_direction;
            found_next_pixel=0;
            for k=1:length(offsets)
                neighbor=currentpixel+offsets(direction);
                if Lp(neighbor)~=0
                    %Found the next boundary pixel.
                    if (Lp(currentpixel)==START)&...
                        isempty(initial_departure_direction)
                    %We are making the initial departure from the starting
                    %pixel.
                    initial_departure_direction=direction;
                    elseif (Lp(currentpixel)==START)&...
                            (initial_departure_direction==direction)
                       % We are about to retrace our path.
                       %That means we're done.
                       done=1;
                       found_next_pixel=1;
                       break;
                    end
                    %Take the next step along the boundary.
                    next_search_direction=...
                        next_search_direction_lut(direction);
                    found_next_pixel=1;
                    numpixels=numpixels+1;
                    if numpixels>size(scratch,1)
                        %Double the scratch space.
                        scratch(2*size(scratch,1))=0;
                    end
                    scratch(numpixels)=neighbor;
                    if Lp(neighbor)~=START
                       Lp(neighbor)=BOUNDARY;
                    end
                    currentpixel=neighbor;
                    break;
                end
                direction=next_direction_lut(direction);
            end
            if ~found_next_pixel
                %If there is no next neighbor,the object must just have a
                %single pixel.
                numpixels=2;
                scratch(2)=scratch(1);
                done=1;
            end
        end
        %Convert linear indices to row_column coordinates and save in the
        %output cell array.
        [row,col]=ind2sub(size(Lp),scratch(1:numpixels));
        B{which}=[row-1,col-1];
    end
end
if strcmp(dir,'ccw')
    for k=1:length(B)
        B{k}=B{k}(end:-1:1,:);
    end
end
```

After extracting the boundaries in each of the block, we have to apply a 4- direction chain code on each of the blocks to populate the matrices hmat, vmat, lmat and rmat.

The function *createcode()* below shows the creation of the 4 direction chain code over an image.

```
function Code=creatcode(B)

[nr,nc]=size(B{1});
Code=[];
for i=2:nr
        x=B{1}(i,1)-B{1}(i-1,1);
        y=B{1}(i,2)-B{1}(i-1,2);

    if x==1&y==0
        Code=[Code,0];
    elseif x==1&y==1
        Code=[Code,1];
    elseif x==0&y==1
        Code=[Code,2];
    elseif x==-1&y==1
        Code=[Code,3];
    elseif x==-1&y==0
        Code=[Code,4];
    elseif x==-1&y==-1
        Code=[Code,5];
    elseif x==0&y==-1
        Code=[Code,6];
    elseif x==1&y==-1
        Code=[Code,7];
    end
end
```

After applying the chain code to the four matrices, a smoothening Gaussian filter is applied on them. Them the value of each element in all the matrices is adjusted by dividing its value by the maximum value of the four matrices. Later, two more matrices are computed using the four matrices.

The following code shows the whole process of constructing a feature matrix. The feature matrix is stored in a matrix called *clarr*.

```
clc
incr=0;
clarr=cell(760,6);
for e=1:38

f=strcat('IMG_00',num2str(e),'/');
f1=f;
for z=0:19
    incr=incr+1;

    f=strcat(f1,'contour',num2str(z),'.tiff');
im=imread(f);
[m n]=size(im);

w=ceil(m/12);
x=ceil(n/12);

m=w*12;
n=x*12;

im4=imresize(im,[m,n]);
[m n]=size(im4);
c=mat2cell(im4,[m/12 m/12 m/12 m/12 m/12 m/12 m/12 m/12 m/12 m/12 m/12 m/12],
[n/12 n/12 n/12 n/12 n/12 n/12 n/12 n/12 n/12 n/12 n/12 n/12]);

hmat=[];vmat=[];lmat=[];rmat=[];

for i=1:12
    for j=1:12

        J=boundaries(c{i,j},8,'cw');
        if(isempty(J))
            hi=0;vi=0;li=0;ri=0;
        else
        Code=creatcode(J);

        h=0;v=0;l=0;r=0;
        [m n]=size(Code);
        for p=1:n
            switch Code(p)
                case {0,4}
                    h=h+1;
                case {1,5}
                    r=r+1;
                case {2,6}
                    v=v+1;
                case {3,7}
                    l=l+1;
            end
        end

        hi=h/n;
        if(isnan(hi))
            hi=0;
        end

        vi=v/n;
        if(isnan(vi))
            vi=0;
        end

        li=l/n;
        if(isnan(li))
            li=0;
        end

        ri=r/n;
        if(isnan(ri))
            ri=0;
        end
        end

        hmat=[hmat,hi];
        vmat=[vmat,vi];
        lmat=[lmat,li];
        rmat=[rmat,ri];

    end

end
hmat=reshape(hmat,12,12);hmat=hmat';
vmat=reshape(vmat,12,12);vmat=vmat';
rmat=reshape(rmat,12,12);rmat=rmat';
lmat=reshape(lmat,12,12);lmat=lmat';

fil=fspecial('gaussian',[3 3],1.2);
realfil1=imfilter(hmat,fil);
realfil2=imfilter(vmat,fil);
realfil3=imfilter(lmat,fil);
realfil4=imfilter(rmat,fil);

 m1=max(max(realfil1));
 m2=max(max(realfil2));
 m3=max(max(realfil3));
 m4=max(max(realfil4));

 allmax=[m1 m2 m3 m4];

 maxi=max(allmax);

 for i=1:12
     for j=1:12
         realfil1(i,j)=realfil1(i,j)/maxi;
         realfil2(i,j)=realfil2(i,j)/maxi;
```

```
            realfil3(i,j)=realfil3(i,j)/maxi;
            realfil4(i,j)=realfil4(i,j)/maxi;
        end
    end

    addmod=[];
    mulmod=[];
    for i=1:12
        for j=1:12
            max1=max(realfil1(i,j),realfil2(i,j));
            max2=max(realfil3(i,j),realfil4(i,j));
            min1=min(realfil1(i,j),realfil2(i,j));
            min2=min(realfil3(i,j),realfil4(i,j));
            if(max1==0)
            addmod(i,j)=0;
            else
                addmod(i,j)=max1/min1;
            end
            if(max2==0)
                mulmod(i,j)=0;
            else
                mulmod(i,j)=min2/max2;
            end

        end
    end

    clarr(incr,1)={realfil1};
    clarr(incr,2)={realfil2};
    clarr(incr,3)={realfil3};
    clarr(incr,4)={realfil4};
    clarr(incr,5)={addmod};
    clarr(incr,6)={mulmod};
end

end
save('feature.mat','clarr');
```

the feature matrix is stored in the disk for use by the verification algorithms.

5.5 Verification of signatures

The verification is done by following two different approaches- the SVM (Support Vector Machines) and the MLP (Multi Layer Perceptrons) approaches.

5.5.1 SVM method for verifying the signatures.

The SVM method of signature verification is employed by first training some set of data and by testing the training data with all the others. Training is performed for different samples and the results at each trial are observed. The below code shows the process of training and testing with SVM:

```
function [classes]=train5svm(incx)
```

```
load feature.mat;
clc;
train7=cell(2,20);
j=1;
i=incx;
while i<=incx+39

    if(i==incx+5)
        i=i+15;

    end
    if(i==incx+25)
        break;
    end
    cc11=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} clarr{i,6}];
    B5=reshape(cc11,864,1);
    train7{1,j}=[B5; ];
    j=j+1;
    i=i+1;
end

train7(2,1)={0};train7(2,2)={0};train7(2,3)={0};train7(2,4)={0};train7(2,5)={0};
train7(2,6)={1};train7(2,7)={1};train7(2,8)={1};train7(2,9)={1};train7(2,10)={1}
;
P11 = cell2mat(train7(1,:));
T11 = cell2mat(train7(2,:));
[R,Q] = size(P11);
[S2,Q] = size(T11);
options = optimset('maxiter',100000);

net =
svmtrain(P11',T11','Kernel_Function','linear','Polyorder',2,'quadprog_opts',opti
ons);
train8=cell(2,20);
j=1;
i=incx;
while i<=incx+39
    if(i==incx||i==incx+20)
        i=i+5;

    end

   cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} clarr{i,6}];
   B6=reshape(cc,864,1);
    train8{1,j}=[B6; ];
    j=j+1;
    i=i+1;
end
P12 = cell2mat(train8(1,:));
classes = svmclassify(net,P12');
```

The above code shows taking only 5 genuine and 5 forged samples for training. If we want to consider 10 or 15 samples for training, we have to make some minor changes to the above code to consider 10 and 15 samples respectively.

Taking random samples for verification in SVM

The below code shows the verification of signatures taking 10 genuine samples and 10 forged samples for training and the 10 genuine samples and 10 forged samples are generated randomly for testing:

```
function [classes]=train10svmrandom(incx)
%% code to train using svm and test the samples
% here we take 10 genuine and 10 forgeries to train using svm
load feature.mat;
clc;
train7=cell(2,20);
j=1;
i=incx;
gen=randperm(20);
% loop to generate training matrix
for k=1:10
    i=gen(k);

    cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} clarr{i,6}];
    B=reshape(cc,864,1);
    train7{1,j}=[B; ];
    j=j+1;
    if(k==10)
        for l=1:10
            i=gen(l)+20;
            cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5}
clarr{i,6}];
            B=reshape(cc,864,1);
            train7{1,j}=[B; ];
            j=j+1;
        end
    end
end

train7(2,1)={0};train7(2,2)={0};train7(2,3)={0};train7(2,4)={0};train7(2,5)={0};
train7(2,6)={0};train7(2,7)={0};train7(2,8)={0};train7(2,9)={0};train7(2,10)={0}
;train7(2,11)={1};train7(2,12)={1};train7(2,13)={1};train7(2,14)={1};train7(2,15
)={1};train7(2,16)={1};train7(2,17)={1};train7(2,18)={1};train7(2,19)={1};train7
(2,20)={1};
%extract training matrix
P11 = cell2mat(train7(1,:));
T11 = cell2mat(train7(2,:));
[R,Q] = size(P11);
[S2,Q] = size(T11);
options = optimset('maxiter',100000);

%training using svm
net =
svmtrain(P11',T11','Kernel_Function','linear','Polyorder',2,'quadprog_opts',opti
ons);
train8=cell(2,20);
j=1;
i=incx;
%============Testing using 10 genuine and 10 forgery samples============%
```

```
for k=11:20
    i=gen(k);

    cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} clarr{i,6}];
    B=reshape(cc,864,1);
    train8{1,j}=[B; ];
    j=j+1;
    if(k==20)
        for l=11:20
            i=gen(l)+20;
            cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5}
clarr{i,6}];
            B=reshape(cc,864,1);
            train8{1,j}=[B; ];
            j=j+1;
        end
    end
end

P12 = cell2mat(train8(1,:));
classes = svmclassify(net,P12');
```

5.5.2 MLP method for verifying the signatures

MLP is another method for verifying the signatures. The processes of training and testing are also present in MLP. The training is performed for different training samples and the results at each trial are observed. The below code shows the process of training and testing with MLP.

```
function [A1]=mlp5train(incx)
clc;
load feature.mat;
train1=cell(2,20);
j=1;
i=incx;
while i<=incx+39

    if(i==incx+10)
        i=i+5;
        end
    if(i==incx+25)
        break;
    end
    cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} clarr{i,6}];
    B=reshape(cc,864,1);
    train1{1,j}=[B; ];
    j=j+1;
    i=i+1;
end
train1(2,1)={0};train1(2,2)={0};train1(2,3)={0};train1(2,4)={0};train1(2,5)={0};
train1(2,6)={0};train1(2,7)={0};train1(2,8)={0};train1(2,9)={0};train1(2,10)={0}
;train1(2,11)={1};train1(2,12)={1};train1(2,13)={1};train1(2,14)={1};train1(2,15
)={1};train1(2,16)={1};train1(2,17)={1};train1(2,18)={1};train1(2,19)={1};train1
(2,20)={1};
```

```
P = cell2mat(train1(1,:));
T = cell2mat(train1(2,:));
[R,Q] = size(P);
[S2,Q] = size(T);



S1 = 15;   % hidden layers
net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;

net.performFcn = 'sse';        % Sum-Squared Error performance function
net.trainParam.goal = 0.001;   % Sum-squared error goal.
net.trainParam.show = 20;      % Frequency of progress displays (in epochs).
net.trainParam.epochs = 4500;   % Maximum number of epochs to train.
net.trainParam.mc = 0.95;
trainParam.min_grad = 1e-5;

net.trainParam.lr = 0.05;
net.trainParam.lr_inc = 1.01;

 [net,tr] = train(net,P,T);

%=================Testing=================

train2=cell(2,20);
j=1;
i=incx;
while i<=incx+39
    if(i==incx||i==incx+20)
        i=i+5;

    end

   cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} clarr{i,6}];
   B=reshape(cc,864,1);
    train2{1,j}=[B; ];
    j=j+1;
    i=i+1;
end
P1 = cell2mat(train2(1,:));
A1 = sim(net,P1);
end
```

The above code shows taking only 5 genuine and 5 forged samples for training. If we want to consider 10 or 15 samples for training, we have to make some minor changes to the above code to consider 10 and 15 samples respectively.

Taking random samples for verification in MLP

The below code shows the verification of signatures taking 10 genuine samples and 10 forged samples for training and the 10 genuine samples and 10 forged samples are generated randomly for testing.

```
function [A1]=testerrandom(incx)
```

```
%% code to train using mlp and test the samples
% here we take 10 genuine and 10 forgeries to train using mlp
clc;
load feature.mat;
train1=cell(2,20);
j=1;
i=incx;
gen=randperm(20);
k=1;

% loop to generate training matrix
for k=1:10

    i=gen(k);

    cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} clarr{i,6}];
    B=reshape(cc,864,1);
    train1{1,j}=[B; ];
    j=j+1;
    if(k==10)
        for l=1:10
            i=gen(l)+20;
            cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5}
clarr{i,6}];
            B=reshape(cc,864,1);
            train1{1,j}=[B; ];
            j=j+1;
        end
    end
   end

train1(2,1)={0};train1(2,2)={0};train1(2,3)={0};train1(2,4)={0};train1(2,5)={0};
train1(2,6)={0};train1(2,7)={0};train1(2,8)={0};train1(2,9)={0};train1(2,10)={0}
;train1(2,11)={1};train1(2,12)={1};train1(2,13)={1};train1(2,14)={1};train1(2,15
)={1};train1(2,16)={1};train1(2,17)={1};train1(2,18)={1};train1(2,19)={1};train1
(2,20)={1};
%extract training matrix
P = cell2mat(train1(1,:));
T = cell2mat(train1(2,:));
[R,Q] = size(P);
[S2,Q] = size(T);



S1 = 15;   % hidden layers
net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;

net.performFcn = 'sse';         % Sum-Squared Error performance function
net.trainParam.goal = 0.001;    % Sum-squared error goal.
net.trainParam.show = 20;       % Frequency of progress displays (in epochs).
net.trainParam.epochs = 4500;    % Maximum number of epochs to train.
net.trainParam.mc = 0.95;
trainParam.min_grad = 1e-5;

net.trainParam.lr = 0.05;
net.trainParam.lr_inc = 1.01;

%training using mlp
[net,tr] = train(net,P,T);

%============Testing using 10 genuine and 10 forgery samples============%


train2=cell(2,20);
j=1;
i=incx;

for k=11:20
    i=gen(k);

    cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} clarr{i,6}];
    B=reshape(cc,864,1);
    train2{1,j}=[B; ];
    j=j+1;
    if(k==20)
        for l=11:20
            i=gen(l)+20;
            cc=[clarr{i,1} clarr{i,2} clarr{i,3} clarr{i,4} clarr{i,5} 
clarr{i,6}];
            B=reshape(cc,864,1);
            train2{1,j}=[B; ];
            j=j+1;
        end
    end
end

P1 = cell2mat(train2(1,:));

A1 = sim(net,P1);
end
```

5.6 Calculation of efficiency

This process is done based on the results of the signature verification phase, whether SVM or MLP. In this method the FAR, FRR, TAR and TRR are calculated for all the signatures. In the SVM method, the signatures are grouped into two groups, 0 or 1. 0 meaning that the signature is a genuine and 1 meaning that the signature is forged.

5.6.1 Using SVM for verification

The below code shows the calculation of values of FAR, FRR, TAR and TRR when SVM is used:

```
clc;
```

```
incx=1;
A3=[];
for iter=1:19
   A2=train5svm(incx);
    A3=[A3;A2];
    incx=incx+40;
end
true=0;
truef=0;
false=0;
falsef=0;
for i=1:570
    for j=1
    if(mod(i,30)~=0&&mod(i,30)<16&&A3(i,j)==0)
        true=true+1;
    elseif(mod(i,30)~=0&&mod(i,30)<16&&A3(i,j)==1)
        truef=truef+1;
    elseif(mod(i,30)==0||mod(i,30)>=16&&A3(i,j)==0)
        falsef=falsef+1;
    elseif(mod(i,30)==0||mod(i,30)>=16&&A3(i,j)==1)
        false=false+1;
    end
    end
end
TAR5s=(true/285)*100;
FRR5s=(truef/285)*100;
TRR5s=(false/285)*100;
FAR5s=(falsef/285)*100;
save('result5svm.mat','A3','TAR5s','FRR5s','TRR5s','FAR5s');
```

The above code shows taking only five training samples. For taking 10 and 15 samples, we have to make some minor chances to the above code to consider 10 and 15 samples respectively.

Taking random samples for testingin SVM

The below code shows the calculation of FAR, FRR TAR and TRR taking 10 genuine samples and 10 forged samples for training and the 10 genuine samples and 10 forged samples are generated randomly for testing.

```
clc;
%% iterator function which repeatedly calls the train10svm function for
signature of each individual and generates the training matrix and testing for
genuine or forgery
% Also here we calculate True acceptance ratio,false acceptance ratio,true
% rejection ratio and false rejection ratio
% loop to generate training and testing matrix for each individual
incx=1;
A3=[];
for iter=1:19
   A2=train10svmrandom(incx);
    A3=[A3;A2];
    incx=incx+40;
end
true=0;
truef=0;
```

```
false=0;
falsef=0;
% To calculate the TAR,FRR,TRR and FAR based on the testing
for i=1:380
    for j=1
    if(mod(i,20)~=0&&mod(i,20)<11&&A3(i,j)==0)
        true=true+1;
    elseif(mod(i,20)~=0&&mod(i,20)<11&&A3(i,j)==1)
        truef=truef+1;
    elseif(mod(i,20)==0||mod(i,20)>=11&&A3(i,j)==0)
        falsef=falsef+1;
    elseif(mod(i,20)==0||mod(i,20)>=11&&A3(i,j)==1)
        false=false+1;
    end
    end
end
%calculating the pecentage accuracies
TAR10sr=(true/190)*100;
FRR10sr=(truef/190)*100;
TRR10sr=(false/190)*100;
FAR10sr=(falsef/190)*100;
% saving the result
save('result10svmr.mat','A3','TAR10sr','FRR10sr','TRR10sr','FAR10sr');
```

5.6.2 Using MLP for verification

In the MLP method, a target is specified along with the training matrix which contains the values 0s or 1s, corresponding to genuine and forged signatures respectively in the training case.

During testing, the testing signature has the same dimensions as that of the training matrix and another matrix of same dimensions as the signature contains the results of the test. If the result corresponding to a specific entry in the testing signature matrix is closer to 0, then it is accepted. If it is closer to 1, then it is rejected.

The below code shows the calculation of values of FAR, FRR, TAR and TRR when MLP is used:

```
clc;
incx=1;
A3=[];
for iter=1:19
    A2=mlp5train(incx);
    A3=[A3;A2];
    incx=incx+40;
end

tar=0;far=0;trr=0;frr=0;
for i=1:19
    for j=1:30
        if(j<16 && A3(i,j)<=0.6000)
            tar=tar+1;
        elseif(j<16 && A3(i,j)>0.6000)
            frr=frr+1;
        elseif(j>=16 && A3(i,j)<=0.6000)
```

```
            far=far+1;
        else
            trr=trr+1;
        end
    end
end

tarp=tar/285;
farp=far/285;
trrp=trr/285;
frrp=frr/285;
TAR5m=tarp*100;
FAR5m=farp*100;
TRR5m=trrp*100;
FRR5m=frrp*100;
save('result5mlp.mat','A3','TAR5m','FAR5m','TRR5m','FRR5m');
```

The above code shows taking only five training samples. For taking 10 and 15 samples, we have to make some minor chances to the above code to consider 10 and 15 samples respectively.

Taking random samples for testingin MLP

The below code shows the calculation of FAR, FRR TAR and TRR taking 10 genuine samples and 10 forged samples for training and the 10 genuine samples and 10 forged samples are generated randomly for testing.

```
clc;
%% iterator function which repeatedly calls the tester function for signature of
each individual and generates the training matrix and testing for genuine or
forgery
% Also here we calculate True acceptance ratio,false acceptance ratio,true
% rejection ratio and false rejection ratio
% loop to generate training and testing matrix for each individual
incx=1;
A3=[];
for iter=1:19
    A2=testerrandom(incx);
    A3=[A3;A2];
    incx=incx+40;
end

tar=0;far=0;trr=0;frr=0;
% To calculate the TAR,FRR,TRR and FAR based on the testing
for i=1:19
    for j=1:20
        if(j<11 && A3(i,j)<=0.6000)
            tar=tar+1;
        elseif(j<11 && A3(i,j)>0.6000)
            frr=frr+1;
        elseif(j>=11 && A3(i,j)<=0.6000)
            far=far+1;
        else
            trr=trr+1;
        end
    end
```

```
end
%calculating the pecentage accuracies
tarp=tar/190;
farp=far/190;
trrp=trr/190;
frrp=frr/190;
TAR10mr=tarp*100;
FAR10mr=farp*100;
TRR10mr=trrp*100;
FRR10mr=frrp*100;
% saving the result
save('result10mlpr.mat','A3','TAR10mr','FAR10mr','TRR10mr','FRR10mr');
```

6 Verification: Experiments and Results

Testing has to be done twice; once for both SVM and MLP. The testing is done by taking 5,10 and 15 samples of the signatures on both SVM and MLP.
Some of the cases employed are as given as below

SVM

Case 1

In this case is we train 5 genuine and 5 forged samples, 10 genuine and 10 forged samples and 15 genuine and 15 forged samples, and 15,10 and 55 are the number of samples taken for testing . This is depicted as shown in the figure below:

Training signatures used	Number of test samples	FRR	FAR	TAR	TRR
5	15	11.22	12.28	88.77	87.71
10	10	5.78	12.63	94.21	87.36
15	5	7.36	21.05	92.63	78.95

Case 2

In this case we train 10, 15 samples each of genuine and forgeries. The testing samples are taken at random. This is depicted as shown in the table below

Training signatures used	Number of test samples	FRR	FAR	TAR	TRR
10	Random	12.63	3.16	96.84	87.71

15	Random	3.684	5.78	89.47	87.36

MLP

Case 1

In this case is we train 5 genuine and 5 forged samples, 10 genuine and 10 forged samples and 15 genuine and 15 forged samples, and 15,10 and 55 are the number of samples taken for testing . This is depicted as shown in the figure below:

Training signatures used	Number of test samples	FRR	FAR	TAR	TRR
5	15	39.29	12.28	60.70	19.64
10	10	5.26	8.94	94.73	91.05
15	5	9.47	15.78	90.52	84.21

Case 2

In this case we train 10, 15 samples each of genuine and forgeries. The testing samples are taken at random. This is depicted as shown in the table below

Training signatures used	Number of test samples	FRR	FAR	TAR	TRR
10	Random	12.63	19.64	96.84	87.71
15	Random	3.684	5.78	89.47	87.36

7 Conclusion and future work

Signature verification is a very attractive field of research from both scientific and commercial points of view. In recent years, along with the continuous growth of internet and the increasing security requirements for the development of e society, the field of signature verification is being considered with renewed interest since it uses a customary prrsonal authentication method that is accepted at both legal and social levels.

Our signature verification with the Gaussian grid feature extraction technique has better results compared to the Modified directional feature (MDF) as claimed by the paper [3] .

In our experimentation with different number of testing & training samples we made the following observation

- SVM is good classifier when there are less number of training samples as compared to the MLP
- MLP is good classifier when there are less number of training samples as compared to the SVM
- Training and testing with random samples resulted in random changes.

In the near future, our work can be extended for more datasets of Kannada off-line signatures.

8 References

[1] Jukka Iiverinen and Ari Visa “Shape recognition of irregular objects “

[2] Vu Nguyen and Michael Blumenstein “An Application of the 2D Gaussian Filter for Enhancing Feature Extraction in Off-line Signature Verification” , 2011 International Conference on Document Analysis and Recognition

[3] V. Nguyen, Y. Kawazoe, T. Wakabayashi, U. Pal, and M. Blumenstein,“Performance analysis of the gradient feature and the modified direction feature for off-line signature verification,” in 12*th ICFHR*, 2010,
[4]Digital Image Processing using MATLAB