University of Reading

Department of Computer Science

Comparative Analysis of Writer-Independent Verification of Offline Signatures

John Daven Dela Cruz

Supervisor:

A report submitted in partial fulfilment of the requirements of the University of Reading for the degree of
Master of Science in *Data Science and Advanced Computing*

October 14,2022

Declaration

I, John Daven Dela Cruz, of the Department of Computer Science, University of Reading, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of UoR and public with interest in teaching, learning and research.

John Daven Dela Cruz October 14, 2022

Abstract

Signature is widely used as a method of authenticating an individual's identity in any documents of business, governmental, legal, and other transaction. Given its importance, the project aims to distinguish two signatures from each other, verifying whether the signature is genuine or forged. This paper forms a comparative analysis on the application of machine learning algorithms to writer-independent verification of offline signatures.

First, we use a simple matching algorithm which uses mean squared error to classify the binary problem. Then we use machine learning techniques such as logistic regression, random forest, support vector machine, multi-layer perceptron and a deep learning method with convolutional neural network. Each signature goes through a pre-process stage to enhance the appearance of the image, feature extraction methods such as texture analysis, corner and edge detection are applied and then feature selection techniques such as PCA.

The system will be training the model on the CEDAR Dataset. The evaluation method for different algorithms was a plot of the Receiver Operating Characteristic (ROC) curve and the Equal Error Rate (EER). The results of the simple matching performed similarly to a random classifier. The logistic regression performed with an EER of 31%. This result is then followed by multi-layer perceptron, and support vector machine. The final model approach with deep learning convolutional neural network achieved an EER of 10.6%. Despite the impressive results, there are still room for improvement.

Contents

List of Figures

List of Abbreviations

SVM	Support Vector Machines
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
BPNN	Back-Propagation Neural Networks
DRT	Discrete Radon Transform
PNN	Probabilistic Neural Network
HMM	Hidden Markov Model
MLP	Multi-layer Perceptron
PCA	Principal Component Analysis
ROC	Receiver operating characteristic
EER	Equal Error Rate
FAR	False Acceptance Rate
FRR	False Rejection Rate
TPR	True Positive Rate
FPR	False Positive Rate

Chapter 1 Introduction

1.1 Background

The capability to recognize and identify people is a social function which we use in our daily lives. It is a complex skill we have developed, use on instinct, and requires us to make use of our perception and memory to accomplish. For centuries, we have used body characteristics such as face, voice, and gait to identify each other; and this notion was formally reinforced through biometrics in the 19th century. Although it stems from law enforcement to identify criminals, it is now widely used to establish person recognition in many civilian applications [1]. Any human characteristics can be of two types: physiological and behavioural. In this research we wish to focus on the latter, more specifically on handwritten signature.

The signature of an individual is one of the most accepted practices which we use to authenticate a person's identity in any documents of business, governmental, legal, and other transactions. Given its popularity in authentication and as well as our growing technological advancements, this situation motivates the development of a system that will automate the signature verification process.

1.1.1 Types of Signature Verification

Signature verification can be classified into two types: online (dynamic) and offline (static). In online approach, a device is used to extract information about the individual's signature. This takes in dynamic data like pressure, velocity, acceleration, and pen pressure (Ali Karouni, 2011) for authentication use that makes the signature harder to recreate. However, the practicality of this method is difficult in certain situations, as it needs the individual at both times of obtaining the signature and verification process. In offline approach, the signature is taken after the writing process has been completed and is represented as a digital image. The challenge here is that signatures are sensitive to geometric transformations, complex background of signatures, skilled forgery, noise introduced when scanned and pen width.

1.1.2 Types of Forgery

Forgery can be defined as the imitation of an object with the intent to deceive another. They can be categorized into three types: random forgery, simple forgery, and skilled forgery. In random forgery, the forger lacks knowledge of the person's name and signature, and so uses their signature instead. In the case of simple forgery, the forger has access to the person's name but not their signature. This may lead to more similarities to the legitimate signature, i.e., those who sign with their full name or parts of it. In skilled forgery, the forger has access to both the person's name and signature for practice. This results in imitations very similar to the genuine signatures, making it difficult to distinguish between the two.

1.2 Problem statement

The main issue we face is that compared to biometric characteristics such as fingerprint, face, or iris; handwritten signatures from a person can be inconsistent at times between each signature. This becomes more problematic when skilled forgery is considered, where the forger has practiced imitating the targeted person's signature, as their imitation can closely resemble the genuine signature.

1.3 Aims and objectives

The aims of this project is to investigate and form a comparative analysis of different machine learning classifiers such as logistic regression and support vector machines combined with feature extraction methods on signatures taken from the online available signature database CEDAR. Considering the variations in the signature images such as illumination or scale, we will eliminate any form of potential biases that can affect model performance and training through a pre-processing stage such the classifiers can be trained on any dataset provided. Our objective for the classifiers is to minimize the Equal Error Rate and maximise the Area Under ROC. With the combination of these techniques we aim to see which classifier performs better and see how they compare to a deep learning convolutional neural network model.

1.4 Solution approach

In this research the primary focus is on writer-independent offline signature verification, hence compared to training a model that will be able to distinguish genuine or forged signature on an individual, we consider the training the model on the whole dataset to have a generalised classification method.

1.4.1 Pre-process

During the pre-processing stage we aim to enhance the quality of the image using Gaussian blurring and thresholding to binarize the image and reduce the noise. We then find the contours of the image and crop it to get the region of interest.

1.4.2 Feature extraction and selection

We then extract features using a combination of techniques such as local binary patterns, Gabor, Sobel and Prewitt filters. Given the large amount of features extracted, we aim to reduce the model training time by implementing feature selection using principal component analysis.

1.4.2 Hyperparameter tuning

In order to find the best classification model, each classifier goes through hyperparameter tuning to maximised the Area under the ROC Curve and minimise the Equal Error rate.

1.5 Summary of contributions and achievements

In this report several machine learning classification algorithms were developed and tuned in order to form the best model in dealing with the signature verification problem. In order to do this we used a database of signature, different classifiers and an evaluation method. The final model we developed was a deep learning convolutional neural network (CNN) that takes raw images as input and had resulted in an EER of 10.6% and test accuracy of 79.4%. Surprisingly with the combination of our feature extraction and selection techniques, we found that the support vector machine (SVM) model outperformed the CNN with a test accuracy of 85.1% and a similar EER of 11.3%. Although the hardware used to create and train the CNN model may be it's limiting factor, we can still see that the SVM model is comparable to a deep learning architecture such as CNN.

1.6 Organization of the report

This report structure is as follows:

Chapter 1 is an introduction of the project which briefly introduced simple concepts and the problem statement in signature verification. This section focuses on a presenting the goal, aims contribution and motivation of the research, as well as a small structural overview of the report.

Chapter 2 is a focuses on related works and literature review of the different state-of-the-art methods that has already been established.

Chapter 3 presents the theoretical background needed for the proposed system. This will start with introducing image pre-processing techniques and methods for feature extraction and selection. It will then provide a brief explanation of the different classification algorithms and its derivation. We consider key concepts of logistic regression, support vector machines and neural networks.

Chapter 4 discusses the proposed methodology, its implementation, and any problems that we encountered.

Chapter 5 focuses on the findings based on the proposed system and analyses the performance of the signature dataset collection.

Chapter 6 is the discusses suggestions for future work and a conclusion.

Chapter 7 is a reflection on the learning experience of the project.

Chapter 2 Literature Review

2.1 (J.A Du Perez, 2004)

A system was developed that authenticates offline handwritten signatures using the Discrete Radon Transform (DRT) and a Hidden Markov Model (HMM). A database of 924 signatures from 22 writers was considered and the proposed systems achieved an Equal Error Rate (EER) of 18% when only skilled forgeries were considered and an EER of 4.5% in simple forgeries. When compare to a database of 4800 signatures from 51 writers, the proposed system achieved an EER of 12.2% when only skilled forgeries were considered.

2.2 (K. R. Radhika, 2010)

The system uses Hu moments as an extracted feature, which is invariant to rotation, translation and scaling. The experiment was conducted on MCYT signature database of 75 subjects. This consists of 15 genuine samples and 15 forged samples for each subject. Classifiers applied were SVM, FFT, Bayes, PCA and LDA. The best model performance with an accuracy of 90% was a SVM classifier which used a radial basis kernel, achieving a FRR of 8% and FAR of 10% .

2.3 (Hemant B Kekre, 2010)

The feature vector based on Gabor filtering for dynamic signature recognition is fed into a verification system that evaluates the Euclidian distance between the feature vectors of signature templates. 250 signatures from 25 different users were used for testing and the classifier achieved an EER of 10%

2.4 (Ali Karouni, 2011)

The ANN is trained through a given input data in which during the learning process, the output is compared to the target and a network weight correction via a learning algorithm is performed to minimize an error function between the two values. The system is then tested on a database of about 100 signatures from users containing genuine and skilled signatures. The pre-processing methods used were colour inversion, filtering and binarization. The global features that were extracted were: Area, Centroid Coordinates, Eccentricity, Kurtosis and Skewness, afterwards a threshold was taken as 90% in the study (i.e., where the signature is considered forged if below it). This resulted in a False Acceptance Ratio of 1.6% and a False Rejection Ratio of 3%, and a classification ratio of about 93% was obtained.

2.5 (Mustafa Berkay Yilmaz, 2011)

The paper presented an automatic offline signature verification system based on signature's local histogram representations. The signature is divided into zones using both fixed size rectangular or polar grids, where HOG and LBP features are calculated. For either of the representations, features obtained from grid zones are concatenated to form the final feature vector. Two different types of SVM classifiers are trained, namely writer dependent and independent SVMs, to perform verification. The system performance is measured using the skilled forgery tests of the GPDS-160 signature dataset. The SVM classifier fusion where the two types are combined giving the best result of 15.41% and 17.65% equal error rate on skilled forgery test with 12 and 5 references, respectively.

2.6 (Hurieh Khalajzadeh, 2012)

A variety of experiments are performed, and results are presented in the paper. Different numbers of feature maps and dimensions of convolutional and subsampling filters are considered and the best of them is selected for the CNN model. Afterwards, a MLP network is used to classify the features which are extracted with the applicability of the CNN model. The experiments were performed with 176 original Persian signatures from 22 people, with an image size of 640x480. There was no overlap between the training and testing sets. The training was stopped when the minimum error for the validation dataset was achieved. Experiments are performed 10 times for 1000 epochs. The average of 99.86 is acquired in validation performance and the error is fixed after the average of 785 epochs.

2.7 (Ahmed Abdelrahaman, 2013)

The dataset contains 2250 signatures from 75 writers. Each writer has 15 genuine signature and 15 skilled forgeries. For each individual enrolment 5 genuine signatures are used as a reference set and the rest of the signatures are used. The system proposed to use DRT as a form of feature extraction and feed the dataset into a KNN Classifier which achieve an accuracy of 80%

2.8 (Shih Yin Ooi, 2015)

The system proposed an offline signatures verification depend on DRT, principle component s analysis (PCA) and Probabilistic Neural Network (PNN). The proposed methods was tested on a database contains 1000 genuine signatures, 500 skilled forgeries, and 500 casual forgeries which were collected from 100 writers and 10 forgers. The proposed method was able to achieve 3.23%, 1.51%, and 13.07% equal error rate (EER) for casual, random, and skilled forgeries

2.9 (Pallavi V. Hatkar, 2015)

The system uses image processing, geometric features extraction, neural networks technique. For training and testing off the system many signatures are used. For training and testing of the system many signatures are used. The results provided in this research used a total of 1000 signatures. Those 1000 signatures are comprised of 100 sets (i.e. from 100 different people) and, for each person there are 5 samples of genuine signatures and 5 samples offline forgeries. To train the system, a subset of this database was taken comprising of 5 genuine samples taken from each of the 100 different individuals and 5 forgeries made by different person for one signature. The features extracted from 5 genuine signatures and 5 forged signatures for each person were used to train a neural network. After applying a feature vector of test signature if the output neuron generates value close to +1 test signature is declared as genuine or if it generates values close to -1 it is declared as forged. This method was able to achieve Accuracy of 86.25%

2.10 (M. Thenuwara, 2017)

Random forest classifier was applied on ICDAR database to construct an offline signature verification system. 800 signatures were used for training and 200 signature for testing. The system accuracy was 67%.

2.11 (Jahandada, 2019)

The paper worked on two Convolutional Neural Network Architectures; Inception-v1 and Inception-v3 and proved that these two algorithms can be successfully used to verify individuals in an organization using handwritten signatures. These two models have performed better than other results available in

literature which uses he same GPDS Synthetic Signature Database which achieved lowest 26.74% EER whilst this study achieved the lowest at 26% EER.

2.12 (Mohammed, 2019)

This paper involves converting images to the frequency space using Gabor converter and then extracting the statistical and engineering characteristics of the images to form the matrix of the input to the back-propagation neural networks BPNN. A set of live images were collected as a database of people affiliated with the University of Mosul. The proposed work showed a 88.57% success rate

2.13 (Jivesh Poddar, 2020)

The test signature is recognized with the given input training set using both CNN and Crest-Trough methods. Then forgery detection algorithms (Harris Algorithmic followed by Surf Algorithm) are enforced on this classified image. The identification with neural networks yielded an accuracy of 94% and the proposed forgery detection works with associate accuracy of 85-89%.

Chapter 3 Theoretical Background

3.1 Pre-processing

Pre-processing plays an essential role in ensuring extraction and obtainment of features are accurate from the digital images. As discussed, signature images may possess variations in terms of pen thickness, scale, rotation, etc. even if it's the genuine signature of the individual.

3.1.1 Grayscale

The image is converted to a grayscale image by eliminating the hue and saturation while preserving the brightness. (Ali Karouni, 2011)

3.1.2 Noise Filtering

Scanned signature images often contain noise which can be reduced through a noise removal filter to the image. (Jivesh Poddar, 2020)

3.1.3 Thresholding

The image is filtered by a low pass FIR filter to remove high-frequency components. Then the image is segmented to get a binary image where each pixel is one of two discrete values: 1 or 0. (Ali Karouni, 2011)

3.1.4 Region Of Interest (ROI)

The aim is to locate the actual region where the signature image lies and discard any region which posses irrelevant information.

3.1.5 Rescaling

Signatures dimensions may vary due to the irregularities in the images scanning and capturing process may causes signatures dimensions to vary.

3.2 Feature Extraction

Features can be divided into two types: global or local. Global features take the signature image as a whole – for example, features such as the height or width of the signature. Local features describe parts of the image by dividing the image in a grid and applying the feature extractor in each part.

3.2.2 Texture features

Local binary pattern (LBP) is a powerful feature proposed to capture the texture in objects. In the basic LBP method, a grayscale image is processed such that a binary code is generated for each pixel in the image (Mustafa Berkay Yilmaz, 2011). Gabor filters are special classes of band pass filters, i.e., they allow a certain 'band' of frequencies and reject the others. By applying properly tuned Gabor filters to a signature image, the texture information can be generated. These accentuated texture information can be used to generate feature vector. (Hemant B Kekre, 2010)

3.2.3 Geometric Feature

Geometric features measure the overall shape of the signature. The various shape features that can be used are Area, Centroid Coordinates, Eccentricity, Kurtosis and Skewness (Ali Karouni, 2011):

- Area: Number of pixels in the region.
- Centroid: Horizontal and vertical centres of gravity of the signature.
- Eccentricity: The ratio of the distance between the foci of the ellipse and its major axis length
- Kurtosis: A measure of the flatness of distribution.
- Skewness: A measure of the asymmetry of the distribution.

3.3 Feature Selection

3.3.1 Feature scaling

Feature scaling allows each features to follow the same scale. This removes any potential biases from one feature to another and potentially increase training speed of the model.

- Normalization makes the value of each feature lie in the range [0,1]. $\boldsymbol{z = \frac{x - \min(x)}{\max(x) - \min(x)}}$
- Standardization makes the value of each feature in the dataset to have zero mean and unit variance. $\boldsymbol{z = \frac{x - \mu}{\sigma}}$

3.3.2 Principal Component Analysis (Ian T. Jolliffe, 2016)

Variance is a numerical description that refers to the spread of values within a sample, defined by the formula $\boldsymbol{\sigma^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}}$ where $\boldsymbol{\sigma^2}$ is sample variance, $\boldsymbol{x_i}$ is the value of one observation, $\boldsymbol{\bar{x}}$ being the mean, and n the number of observations. When we wish to find the relationship of two sets of data x and y, we find the covariance which measures the relationship between them, defined by $\boldsymbol{Cov(x, y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}}$.

Expanding on to multiple sets data, which we denote as $\boldsymbol{\vec{X}}$ = ($\boldsymbol{X_1, .., X_n}$), the covariance between each sets of data $\boldsymbol{X_1, .., X_n}$ can be defined with the covariance matrix:

$$\boldsymbol{A = Cov(\vec{X}) = \begin{pmatrix} Cov(X_1, X_1) & Cov(X_1, X_2) & \cdots & Cov(X_1, X_n) \\ Cov(X_2, X_1) & Cov(X_2, X_2) & \cdots & Cov(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(X_n, X_1) & Cov(X_n, X_1) & \cdots & Cov(X_n, X_n) \end{pmatrix}}$$

Principal components analysis (PCA) is a technique utilizes the covariance matrix A of the dataset to transform it into a lower dimensional form, without losing too much information. This is done by first finding the eigenvalues $\boldsymbol{\lambda}$ and eigenvectors $\boldsymbol{v}$ of the covariance matrix of A, by solving $\boldsymbol{Av = \lambda v}$.

This can be written as $\boldsymbol{Av - \lambda v = 0 \Rightarrow (A - \lambda I)v = 0}$

$\boldsymbol{I}$ is the identity matrix of the same dimension as $\boldsymbol{A}$. If $\boldsymbol{v}$ non-zero, the equation holds true if $\boldsymbol{(A - \lambda I)}$ is not invertible, i.e. $\boldsymbol{det(A - \lambda I) = 0}$

The eigenvalues gives the explained variance of the data, so the eigenvectors (principal components) with highest eigenvalues would be considered the most important and are used to transform the dataset. Explained variance can be used to choose the number of dimensions to keep in a reduced dataset. For example, a dataset with 10 features can be reduced to 3 dimensions if 3 principal components gives 90% of the explained variance of the original data.

3.4 Machine Learning Algorithms

3.4.2 Linear Regression

In linear regression we obtain an estimate of the unknown variable y by computing the sum of our know variables x to which we add a bias term, defined as

$$y = b + \sum_{i=1}^{n} w_i * x_i$$

This can be vectorised in the form of $y = [x_0, x_1, \ldots, x_n] \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix} + b$

This equation is only for a data point $\boldsymbol{x}$ and some weight vector $\boldsymbol{w}$, but if we considered a collection of data points and concatenate it into a matrix $\boldsymbol{X}$, we get the general equation as:

$$y = \boldsymbol{Xw} + \boldsymbol{b}$$

3.4.2 Logistic Regression

The logistic regression model uses the logistic function to transform an input of (-∞,∞) to an output of a linear (0,1), defined by: $f(x) = \frac{1}{1 + exp(-x)}$.

Logistic regression fuses together linear regression and the logistic function to be able to output a probability [0,1]. Consider the linear regression model where we have modelled the relationship between the outcome and features with a linear equation

$$y = b + +w_1 * x_1 + \ldots + w_p * x_p$$

Then inputting this into the logistic function we can output a probability p(x) [0,1], i.e.

$$p(x) = \frac{1}{1 + exp(-(b + +w_1 * x_1 + \ldots + w_p * x_p))}$$

The loss function for logistic regression is Log Loss defined as:

$$Log\ Loss = \sum_{(x,y)\in D} -ylog(y') - (1-y)log(1-y')$$

where:

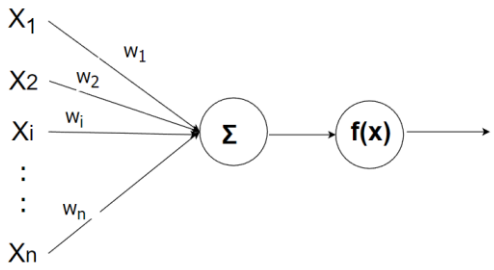
- $(x, y) \in D$ is the data set containing many labelled examples, which are (x, y) pairs.
- y is the label in a labelled example
- y' is the predicted value, give the set of features in x

3.4.3 Support Vector Machine

The objective of the support vector machine is to find a hyperplane in an N-dimensional space that classifies the data points. The aim is to find a plane which maximises the distance between the data points of both classes. In a non-linear case where it cannot be separated with a line, we use kernels that will transform and map the data into higher dimensional space to classify.

3.5 Neural Networks

The neural network architecture is inspired by the interconnected network of neurons in the human brain. These neurons receive and transmit information to help humans process information. Similarly neural networks consists of nodes that are connected by direct links. This link associated with a weight, is used to transmit some out put from one node to another. This can be generalised with the equation:



$$output = f(\sum(x_n * w_n) + b)$$

Figure 1: Perceptron

where:

- f(x) is some activation function which lets the output through
- x is the input (from other nodes)
- w is the associated weights
- b is the bias within the node

3.5.1 Structure of Neural Network

In a standard neural network there are three different types of layers:

- Input layer: Information from the outside world enters the artificial neural network from the input layer. Input nodes process the data, analyse or categorize it, and pass it on to the next layer.
- Hidden layer: Hidden layers take their input from the input layer or other hidden layers. Artificial neural networks can have a large number of hidden layers. Each hidden layer analyses the output from the previous layer, processes it further, and passes it on to the next layer.
- Output layer: The output layer gives the final result of all the data processing by the artificial neural network and can have single or multiple nodes. For binary classification, the output layer will have one output node which gives the result as 1 or 0.

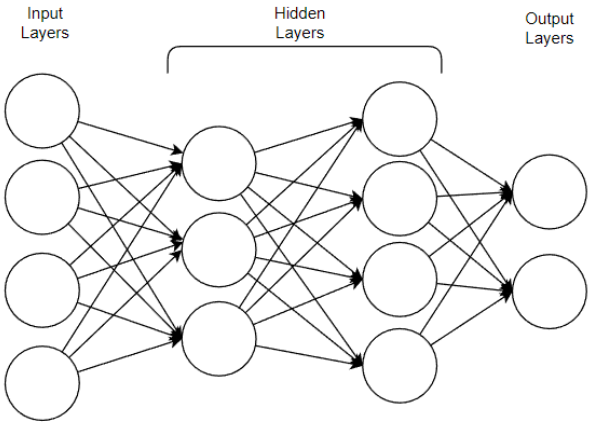


Figure 2: Layer Structure in Neural Network

3.5.3 Activation Function

An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations

Common Activation Functions:

- Sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$

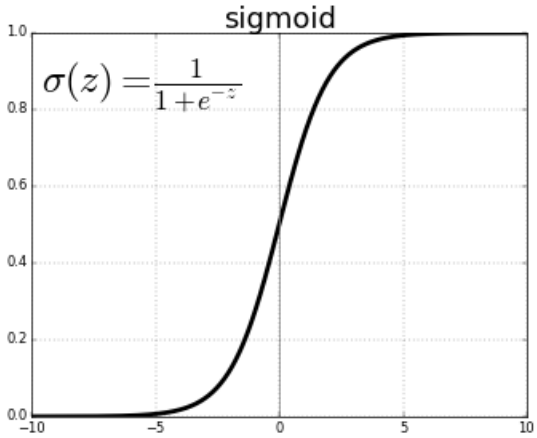


Figure 3: sigmoid activation

- ReLU (Rectified Linear Unit): $R(z) = max(0, z)$

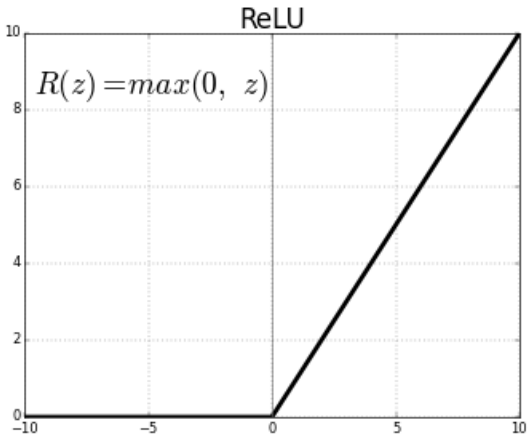


Figure 4: ReLU activation

3.5.4 Loss Function

The objective is to make the model output be as close as possible to the desired output (truth values. Each predicted class probability is compared to the actual class desired output 0 or 1 and a score/loss is calculated that penalizes the probability based on how far it is from the actual expected value. Within this classification problem we use binary cross-entropy defined as:

$$f(x) = -\sum_{i}^{2} t_i \log p_i,$$

$$where\ t_i\ is\ the\ target\ label\ \ and\ p_i\ is\ the\ calcualted\ probability$$

Given that it uses the logarithmic function, it applies a greater penalty to the calculated probability the further it is from the target label. Binary cross-entropy loss is used when adjusting model weights during training. The aim is to minimize the loss until model training is over.

3.5.5 Optimizers

Optimization algorithms or strategies are responsible for reducing the losses and to provide the most accurate results possible. They do this by changing the weights and learning rate of the model.

- Batch Gradient Descent – calculates the next point using the gradient at the current position and scaling it by the learning rate and subtracting it from the current position.
- Stochastic Gradient Descent – instead of calculating the gradient for the entire training data set like in batch gradient descent, it does it for an observation picked at random. This performs better for larger training data.
- Adam – a further extension of stochastic gradient descent which is used to update the network weights during training. The key difference is that the learning rate for each network is updated individually

3.5.6 Convolutional Neural Networks (Anirudha Ghosh, 2020)

3.5.6.1 Convolutional Layers

The convolution layer contains convolutional kernels/filters which get convolved with the input image to generate an output feature map. This kernel can be describe as a (n, n) matrix with discrete values know as the weight. During training this is initialized with random numbers and are then tuned to ensure the kernels extract meaningful features.

3.5.6.2 Pooling Layers

The pooling layers are used to sub-sample (reduce dimension) the feature maps whilst preserving the most dominant features. An example would be max pooling where it outputs the max value of an (n, n) pooling region. This helps reduce the network complexity and computational cost.

3.5.6.3 Fully-Connected Layers

The Fully-Connected Layers take input from the final convolutional or pooling layer which are then flattened to create a vector that is then fed into the either FC layer to generate the final output.

3.5.6.4 Dropout Layers

The main purpose of the dropout later is to nullify contributions of some neurons toward the next layer, the reason is to reduce the changes of overfitting the network.

3.5.6.5 Stride and Padding

The stride indicates the number of steps we are moving in each convolution, increasing this value reduces the dimension of image. To prevent this we use padding. This increases the dimension by adding zeros to the border of the input image.

3.6 Datasets

3.6.1 Privacy

Databases must be built according to ethical aspects such as privacy of the donors (Department for Digital, Culture, Media & Sport and Home Office, 2018). This hinders the acquisition of signatures and database maintenance. To alleviate this problem, an alternative has been to develop databases with disguised genuine signatures, where the signers invent a pretended genuine signature (Moises Diaz, 2019). However, for this research we are using publicly available databases meaning that the dataset is already acquired and maintained by the issuer, and so our only concern is to ensure that it is only used for its intended purpose of research.

3.6.2 CEDAR

The collection contains 1,320 genuine signatures from 55 different individuals. Some were asked to forge three other writers' signatures, eight times per subject, thus creating 1,320 skilled forgeries. Each signature was scanned at 300 dpi grayscale and binarized using a grayscale histogram. Salt pepper noise removal and slant normalization were two steps involved in image pre-processing. Hence the database has 24 genuine and 24 skilled forgeries per each writer

Chapter 4 Proposed Methodology

4.1 Hardware and Software

Hardware

During the entire development of this project, the author's personal computer was used.

- CPU: AMD Ryzen 9 5900HS, 3301 MHz, 8 Core(s), 16 Logical Processor(s)
- GPU: Nvidia RTX 3060 (Laptop), with 6GB RAM
- Size of RAM: 32.0 GB

A problem that occurred was when the models were ran for tuning the CPU usage was 100% utilised and the Python runtime was long causing overheating problems. We were able to tune the models in the end, but such issues made it a bit difficult.

Software

- The project is mainly developed in Python 3.9
- TensorFlow and CUDA was used in developing neural network models.
- Python libraries such as Pandas, NumPy, ScikitLearn, Matplotlib and Seaborn was used.

4.2 Train, Test and Validation

The total data set was divided into two separate sets, the training set and validation set. The ratio used between them was 7:3, meaning that if a person gave 10 signatures, 7 would be in the training set and the 3 would be in the validation set. During the process we implemented stratified 5 fold partitioning in non neural network models, meaning the training set would be split once more into 5 different partitions for training and testing on the classifier.

4.3 Pre-processing of Data

Firstly, we reduce the noise in the image by applying gaussian blurring. Then afterwards we binarize it using Otsu's threshold. We want to remove any unnecessary components of the image and so we locate a bounding rectangle in which the image lies and crop it. Considering that each signature image isn't of the same size, we rescale the image to 96x96 pixel for faster processing.

```
DIRECTORY ="C:/Users/jdeed/Desktop/project/dataset/ICDAR_train"
CATEGORY = ["Forged","Genuine"]

def create_training_data(img_size):
    training_data=[]
    for category in CATEGORY:
        path=os.path.join(DIRECTORY,category)
        class_num=CATEGORY.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path,img), cv2.IMREAD_GRAYSCALE)
                # Otsu's thresholding after Gaussian filtering
                blur_img=cv2.GaussianBlur(img_array,(5,5),0)
                ret,thresh = cv2.threshold(blur_img,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
                #finds the region of interest
                x,y,w,h = cv2.boundingRect(thresh)
                ROI = thresh[y:y+h, x:x+w]
                #scales the data to the same size
                new_array = cv2.resize(ROI, (img_size[0],img_size[1]))
                training_data.append([new_array,class_num])
            except Exception as e:
                pass
    return training_data
```

Figure 5 : Implementation of preprocessing

Figure 6: Before and after data pre-processing

4.4 Feature Extraction

4.4.1 Gabor Filter

Gabor is a convolution filter representing a combination of gaussian and a sinusoidal term. The gaussian component provides the weights and sine component provides the directionality. This can be used to generate features that represent texture and edges.

$$g(x,y,\sigma,\theta,\lambda,\gamma,\psi) = exp(-\frac{x'^2+\gamma^2y'^2}{2\sigma^2})exp(i(2\pi\frac{x'^2}{\lambda}+\psi))$$

$$where\ x' = xcos\theta + \boldsymbol{ysin\theta},\qquad \boldsymbol{y'} = \boldsymbol{-xcos\theta} + \boldsymbol{ysin\theta}$$

- λ — Wavelength of the sinusoidal component.
- Θ — The orientation of the normal to the parallel stripes of Gabor function.
- Ψ — The phase offset of the sinusoidal function.
- σ — The sigma/standard deviation of the Gaussian envelope
- ɣ — The spatial aspect ratio and specifies the ellipticity of the support of Gabor function.

```
#gabor filter
num=1
kernels=[]
for theta in range(2):
    theta = theta/4.*np.pi
    for sigma in (1,3):
        lamda=np.pi/4
        gamma=0.5
        phi=0
        gabor_label="Gabor " +str(num)
        kernel_size=9
        kernel = cv2.getGaborKernel((kernel_size,kernel_size),sigma,theta,lamda,gamma,phi,ktype=cv2.CV_32F)
        kernels.append(kernel)
        fimg=cv2.filter2D(img,cv2.CV_8UC3,kernel)
        filtered_img=fimg.reshape(-1)
        df[gabor_label]=filtered_img
        num +=1
```

Figure 7 implementation of Gabor filter using opencv library

4.4.2 Local Binary Patterns (LBP)

LBPs compute a local representation of texture. This is done by comparing each pixel with its surrounding neighbourhood of pixels. The central pixel p_c will be set as a threshold for the 8 neighbouring pixel p_n, i.e. if the neighbouring pixel is greater or equal to the central pixel. Then the value will be set to one and 0 otherwise. (Pietikäinen, 2010)

An example of how it is calculated is show in the figure below:

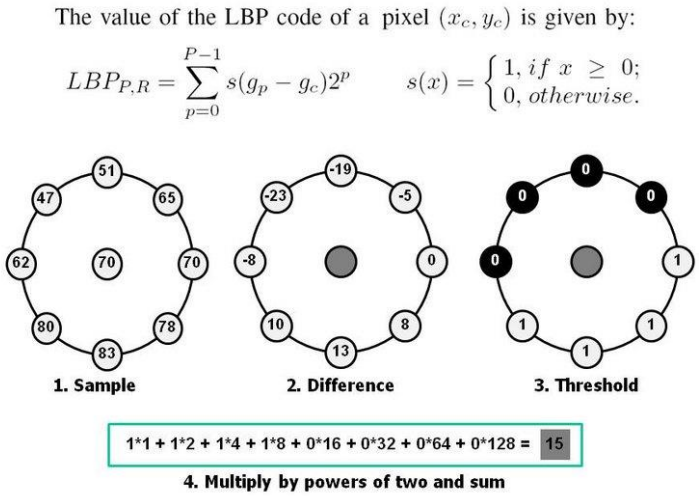


Figure 8: LBP Computation

```
#local binary patterns
lbp_ft = lbp(img,8,1)
ft_lbp = lbp_ft.reshape(-1)
df['Local Binary Patterns'] = ft_lbp
```

Figure 9 implementation of LBP using skimage library

4.4.4 Edge Detection

Filters in the Edge Detection class are designed to detect boundaries between image areas that have distinctly different brightness and to reveal other aspects of image texture. Gradient filter uses two 3 by 3 convolutional kernels to detect gradients in the horizontal and vertical directions.

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

Figure 10: Gx and Gy of Sobel Filter

-1	0	+1
-1	0	+1
-1	0	+1

+1	+1	+1
0	0	0
-1	-1	-1

Figure 11: Gx and Gy of Prewitt filter

The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation. This can be combined together to give the absolute magnitude of the gradient at each point and the orientation of that gradient

$$|G| = \sqrt{{G_x}^2 + {G_y}^2}$$

```
#sobel edge filter
sobel_ft = sobel(img)
edge_s = sobel_ft.reshape(-1)
df['Sobel'] = edge_s

#prewitt edge filter
prewitt_ft = prewitt(img)
edge_p = prewitt_ft.reshape(-1)
df['Prewitt'] = edge_p
```

Figure 12: Implementation of edge filter using skimage library

4.5 Feature Selection

4.5.1 PCA

Extracting features from the image leaves us with a large high dimensional data set which can drastically increase the model training time. In order to select the most important features we plot the PCA fit of the data in order to see how many dimensions it can be reduced down into. In the figure below, we see that about 90% of the variance can be explained by approximately 1200 dimensions. This seems a lot but it's a major improvement from the original dimension of 80,000+ features we have to use.

```
Eigenvalues:
[1.83834921e+07 1.44369163e+07 1.19291848e+07 ... 1.09486329
e-24
 1.07703720e-24 8.74776217e-26]

Variances (Percentage):
[4.33092864e+00 3.40116306e+00 2.81037181e+00 ... 2.57936565
e-31
 2.53736951e-31 2.06086708e-32]
```

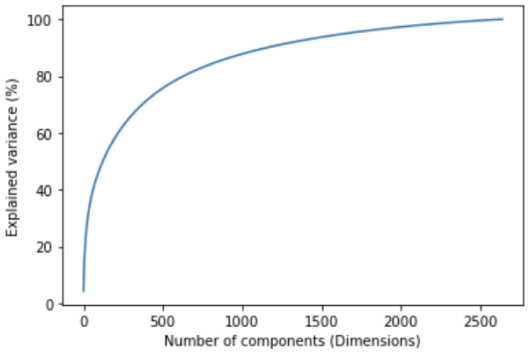


Figure 13: PCA plot of the explained variance

Before we implement the PCA transform we create a standardized training and test set to eliminate any feature scaling that can form a bias on the covariance of dataset.

```
def normalized_training_test_data(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                        test_size=0.3,
                                                        stratify=y,
                                                        random_state=random_state)
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    return X_train, X_test, y_train, y_test
```

Figure 14 Implementation of standardizing data and splitting

```
def PCA_transform(X_train,X_test):
    pca = PCA(0.9,random_state=random_state)
    X1 = pca.fit_transform(X_train)
    X2 = pca.transform(X_test)
    return pd.DataFrame(X1), pd.DataFrame(X2)
```

Figure 15: Transforming the data set into a lower dimension

4.6 Simple Matching Framework

4.6.1 Implementation

A fast and simple approach is to use the raw pixel values of the two images as a metric for measuring their similarity. In this case, we consider the equation of the Mean Squared Error as a basis for computing the similarity metric. We will assume that the lower the value of error, the closer to similarity the images posses.

$$MSE(I_A, I_B) = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}|I_A - I_B|^2 \quad 4.5.1$$

```
def mean_squared_error(data,no_writer,combination):
    MSE=[[] for i in range(0,no_writer)]
    for i in range(0,no_writer):
        for j in range(0,len(combination)):
            MSE[i].append(np.square(np.subtract(data[i][combination[j][0]][0], data[i][combination[j][1]][0])).mean())
    return MSE

MSE_data=mean_squared_error(new_data,no_writers,pairings)
```

Figure 16: MSE function code

Since the classification problem is binary, meaning it is either genuine or forged, we need to establish a threshold in which we accept whether or not there is similarity between the image. Hence we create a range of threshold that the model can use to determine which would give the best accuracy.

```
def threshold_range(data):
    mx=[]
    mn=[]
    for i in range(0,len(data)):
        mx.append(max(data[i]))
        mn.append(min(data[i]))
    return np.linspace(min(mn),max(mx),100)

threshold=threshold_range(MSE_data)
```

Figure 17: Threshold ranged based on the min/max MSE calculated for the image

```
def model_evaluation(data,threshold,no_writer,pairings,mse):
    tpr=[]
    fpr=[]
    accuracy=[]
    maximum=[]
    for x in threshold:
        tp=0
        fp=0
        tn=0
        fn=0
        for i in range(0,no_writer):
            for j in range(0,len(pairings)):
                if mse[i][j]<=x:
                        #matches
                    if data[i][pairings[j][0]][1] == data[i][pairings[j][1]][1]:
                        tp+=1
                    else:
                        fp+=1
                else:
                    if not(data[i][pairings[j][0]][1] == data[i][pairings[j][1]][1]):
                        tn+=1
                    else:
                        fn+=1
        thr_accuracy=(tp+tn)/(tp+tn+fp+fn)
        thr_tpr=tp/(tp+fn)
        thr_fpr=fp/(fp+tn)

        accuracy.append(thr_accuracy)
        tpr.append(thr_tpr)
        fpr.append(thr_fpr)

        current_max=max(accuracy)
        if thr_accuracy==current_max:
            maximum=[x,thr_accuracy,thr_fpr,thr_tpr]

    return [fpr,tpr,accuracy,maximum]

fpr,tpr,accuracy,optimum=model_evaluation(new_data,threshold,no_writers,pairings,MSE_data)
```

Figure 18: Code for evaluation method

4.6.2 Problem encountered

This framework is a writer-dependent classification algorithm as it uses the mean squared error between two similar images. The only issue is that we would require to separate the loaded data set into the individual writers and then consider all the possible combinations of the image to make the matching.

```
#seperating signatures by writers
def seperate_writers(data,no_signatures,no_writers):
    chunks=[data[i:i+no_signatures] for i in range(0, len(data), no_signatures)]
    writer=[]
    for i in range(0,no_writers):
            writer.append(chunks[i]+chunks[i+no_writers])
    return writer

new_data=seperate_writers(training_data,no_signatures,no_writers)
```

Figure 19: separating the signatures

```
#finding all combinations of pairs for image matching
def pair_combination(n):
    combinations=[]
    for i in itertools.combinations(range(0,n), 2):
        combinations.append(i)

    return combinations

pairings=pair_combination(48)
```

Figure 20: finding combinations

4.6 Classifier Framework

4.6.1 Implementation

The classifier model all follow the same process. They all require the input of the training and test set. Which is then further separated to another training set and validation set using stratified k-fold to reduce any form of overfitting. We stratify the sets to ensures that each time the model is trained it is balanced. We then define the parameters we want to tune the model for. The classifier would then be iterated 20 times between a random set of combinations of the parameter and results for each model training. In each iteration the model is attempting to maximise the area under the ROC curve. From there we will be able to select the best classifier model with the lowest EER percentage.

```
def svm_tuner_model(X_train,X_test,y_train,y_test):

    stratified_kfold = StratifiedKFold(n_splits=5,
                                       random_state=random_state,
                                       shuffle=True)

    kernel=['rbf','poly','sigmoid']
    c_values = [0.001,0.1,1,10,100,1000]
    gamma=[1000,100,10,1,0.1,0.01,0.001]

    param_grid = {'kernel':kernel,
                  'C':c_values,
                  'gamma':gamma}

    random_search = RandomizedSearchCV(svm.SVC(cache_size=500,probability=True,random_state=random_state),
                                       param_distributions=param_grid,
                                       scoring='roc_auc',
                                       cv=stratified_kfold,
                                       n_jobs=-1,
                                       n_iter=20)

    random_result = random_search.fit(X_train, y_train)
    cv_results=random_result.cv_results_
    y_predict=random_search.predict(X_test)
    y_preds=random_search.predict_proba(X_test)[:,1]
    cv_score = random_search.best_score_

    accuracy_table=pd.DataFrame((cv_results["split0_test_score"],
                                 cv_results["split1_test_score"],
                                 cv_results["split3_test_score"],
                                 cv_results["split3_test_score"],
                                 cv_results["split4_test_score"]))

    return random_result,cv_results,cv_score,y_predict,y_test,accuracy_table,y_preds
```

Figure 21: SVM model tuner framework

4.7 Multi-Layer Perceptron

4.7.1 Implementation

The model builder aims to a small multi-layer model that is an extension of the logistic regression model. Here it starts with an input layer which will take in the normalized 96x96 image we pre-processed. This will be followed by a Dropout layer which will nullify certain nodes contribution towards the next layer. This is applied to reduce the overfitting nature of the model. We then implement 3 fully connected hidden layers which will use the ReLU function as an activation function. Fully connected means that the output of the hidden layer is a linear combination of its input. The hidden layers will then be followed by a dropout layer once again. Finally the network is passed onto an output layer which will make a binary classification using the sigmoid function as an activation layer.

```
def build_mlp_model(hp):
    model = Sequential()

    model.add(InputLayer(input_shape=(X_dim,), name='Input_Layer'))
    model.add(Dropout(hp.Choice("Input_Dropout_layer",[0.5,0.6,0.7,0.8])))

    for i in range(3):
        model.add(Dense(hp.Int(f'Hidden_{i}_layer',32,512,32),activation='relu',name=f'Hidden_layer_{i}'))
        model.add(Dropout(hp.Choice(f'Dropout_{i}_layer',[0.5,0.6,0.7,0.8]),name=f'Dropou_layer_{i}'))

    model.add(Dense(1, activation='sigmoid', name='Output_Layer'))

    opt=Adam(learning_rate=4e-5)
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

Figure 22: MLP model Builder

4.7.1 Hyperparameter

The model has many hyperparameters that can be varied and it can drastically affect the performance of the classification model if we find a good choice of parameters. There isn't any optimisation algorithms for this except through brute force trial and error. In order to reduce overfitting in the model, we create a validation split in the training data we feed the model. The accuracy of the validation set will be used as a scoring objective in the function that will find us the best combination of choices of hyperparameters. This will be trialled 100 times, training it for 20 epochs with 16 batch size. We created a stopping criterion that minimises validation loss, this metric tells us how well it models new data, if it's increasing that means we are overfitting the training data.

```
def best_network_hyperparameter(model_builder,trials,filename):
    tuner = RandomSearch(
        model_builder,
        objective='val_accuracy',
        max_trials=trials,
        executions_per_trial=2,
        directory='project',
        project_name=filename)
    criterion=EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
    tuner_result = tuner.search(X_train, y_train,epochs=20,batch_size=16,validation_split=0.1,callbacks=[criterion])

    return tuner.get_best_hyperparameters()[0]
```

Figure 23: MLP parameter tuning

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dropout (Dropout)           (None, 9216)              0

 Hidden_layer_0 (Dense)      (None, 384)               3539328

 Dropou_layer_0 (Dropout)    (None, 384)               0

 Hidden_layer_1 (Dense)      (None, 320)               123200

 Dropou_layer_1 (Dropout)    (None, 320)               0

 Hidden_layer_2 (Dense)      (None, 448)               143808

 Dropou_layer_2 (Dropout)    (None, 448)               0

 Output_Layer (Dense)        (None, 1)                 449

=================================================================
Total params: 3,806,785
Trainable params: 3,806,785
Non-trainable params: 0
_________________________________________________________________
```

Figure 24: The best MLP model combination the tuner gave after 100 trials

```
def network_fit(network_model,X_train,y_train):
    criterion = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=50)
    model_fit = network_model.fit(X_train, y_train, batch_size=16, epochs=200, verbose=2,
                    validation_split=0.1,callbacks=[criterion])
    return model_fit
```

Figure 25: Training the model

4.8 Deep Convolutional Network

4.8.1 Implementation

The model builder aims to a build a simple CNN model that is inspire by the VGG16 architecture.

The first convolutional layer takes the input shape of the image.

```
def build_cnn_model(hp):
    CNN_model = Sequential()

    CNN_model.add(Conv2D(hp.Int("input_layer",32,256,32),
                         3,padding='same', activation="relu",
                         input_shape=X[1,:].shape, name="Input_Layer"))
    CNN_model.add(MaxPooling2D(pool_size=(2, 2)))
    CNN_model.add(BatchNormalization())
    CNN_model.add(Dropout(0.8))

    for i in range(3):
        CNN_model.add(Conv2D(hp.Int(f'conv_{i}_layers',32,512,32),3,
                             padding='same', activation="relu",name=f"Convolutional_Layer_{i}"))
        CNN_model.add(BatchNormalization())
        CNN_model.add(MaxPooling2D(pool_size=(2, 2)))

    CNN_model.add(Flatten())
    CNN_model.add(Dense(32,activation="relu",name='FullyConnected_Layer'))
    CNN_model.add(Dropout(0.6))

    CNN_model.add(Dense(1, activation='sigmoid', name='Output_Layer'))

    opt=Adam(learning_rate=1e-4)
    CNN_model.compile(optimizer=opt,
                      loss="binary_crossentropy",
                      metrics=['accuracy'])

    return CNN_model
```

Figure 26: CNN model builder

```
def best_network_hyperparameter(model_builder,trials,filename):
    tuner = RandomSearch(
        model_builder,
        objective='val_accuracy',
        max_trials=trials,
        executions_per_trial=1,
        directory='project',
        project_name=filename)
    criterion=EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
    tuner_result = tuner.search(X_train, y_train,epochs=20,batch_size=64,validation_split=0.1,callbacks=[criterion])

    return tuner.get_best_hyperparameters()[0]
```

Figure 27: CNN parameter tuning

```
def network_fit(network_model,bsize,epoch,X_train,y_train):
    criterion=EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=70)
    model_fit = network_model.fit(X_train, y_train, batch_size=bsize, epochs=epoch, verbose=2,
                        validation_split=0.1,callbacks=[criterion])
    return model_fit
```

Figure 28: CNN Model training

Chapter 5 Results

5.1 Performance Evaluation

A standardized metric of evaluation is require in order to make a comparative analysis of each methods we've discussed. A method would be to consider the percentages of correct predictions i.e. accuracy.

For binary classification there are four different types of prediction. This can be modelled in a confusion matrix where we have the actual values against the predicted values.

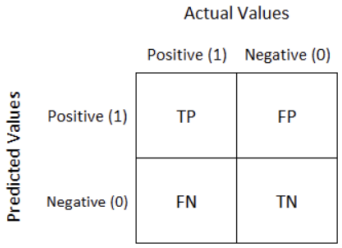


Figure 29: General confusion Matrix

Here we have the predicted values which are given by the model, and the true values from the original data set. Understanding this model we get 4 values:

- True Positives (TP): values that are predicted true and are actually true
- False Positives (FP): values that are predicted true but are actually false
- True Negatives (TN): values that are predicted false but are actually false
- False Negatives (FN): values that are predicted false but are actually true

The Accuracy is defined by: $Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$

An ROC Curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This plots the parameters:

- True Positive Rate : $TPR = \frac{TP}{TP + FP}$
- False Positive Rate: $FPR = \frac{FP}{FP + TN}$

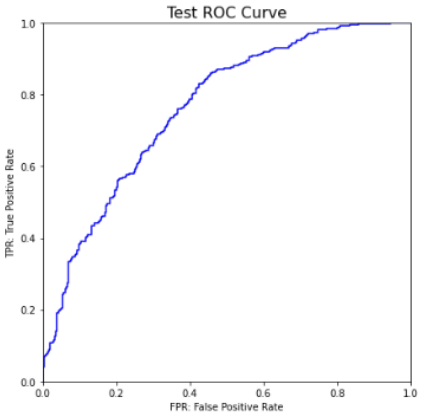


Figure 30: ROC Curve

5.2 Simple Matching Framework

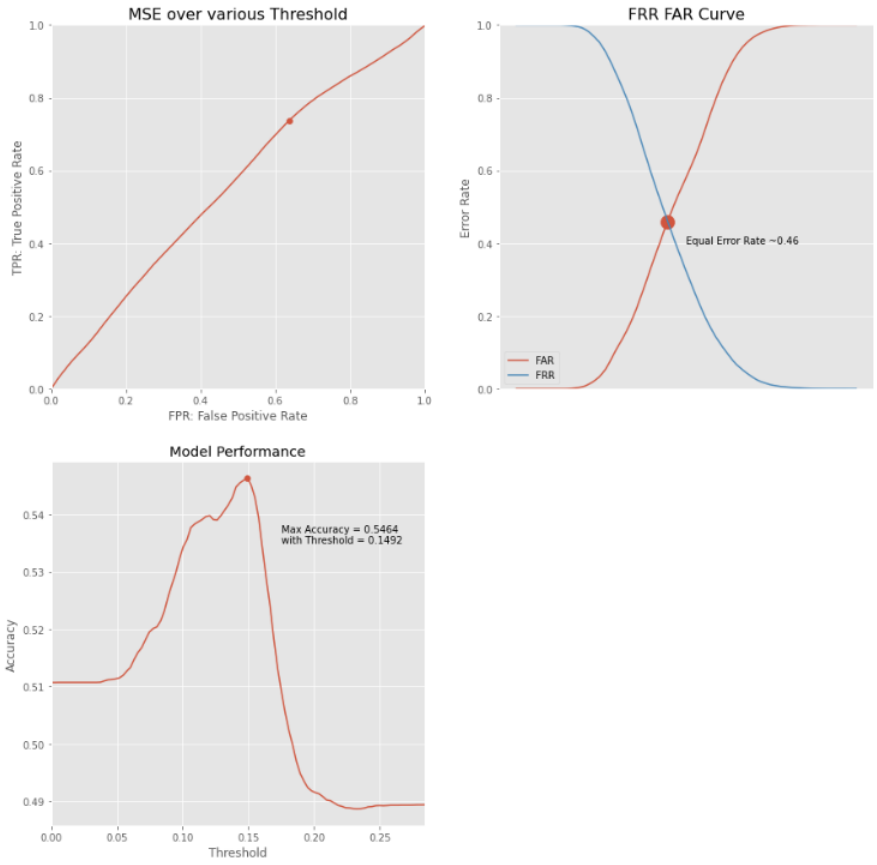


Figure 31: Results for Simple Matching Algorithm

The intention of the simple matching algorithm is to consider a naïve approach to signature verification. Since it is trained on the entire dataset it's not a learning model and isn't comparable to the other models that we explored. In reality this algorithm is no different to randomly guess if a signature is genuine or forged, hence we established a good metric on what a bad classifier is. This will be a basis on what we do not want to achieve for our future classification methods.

5.3 Classifier Framework

Logistic Regression

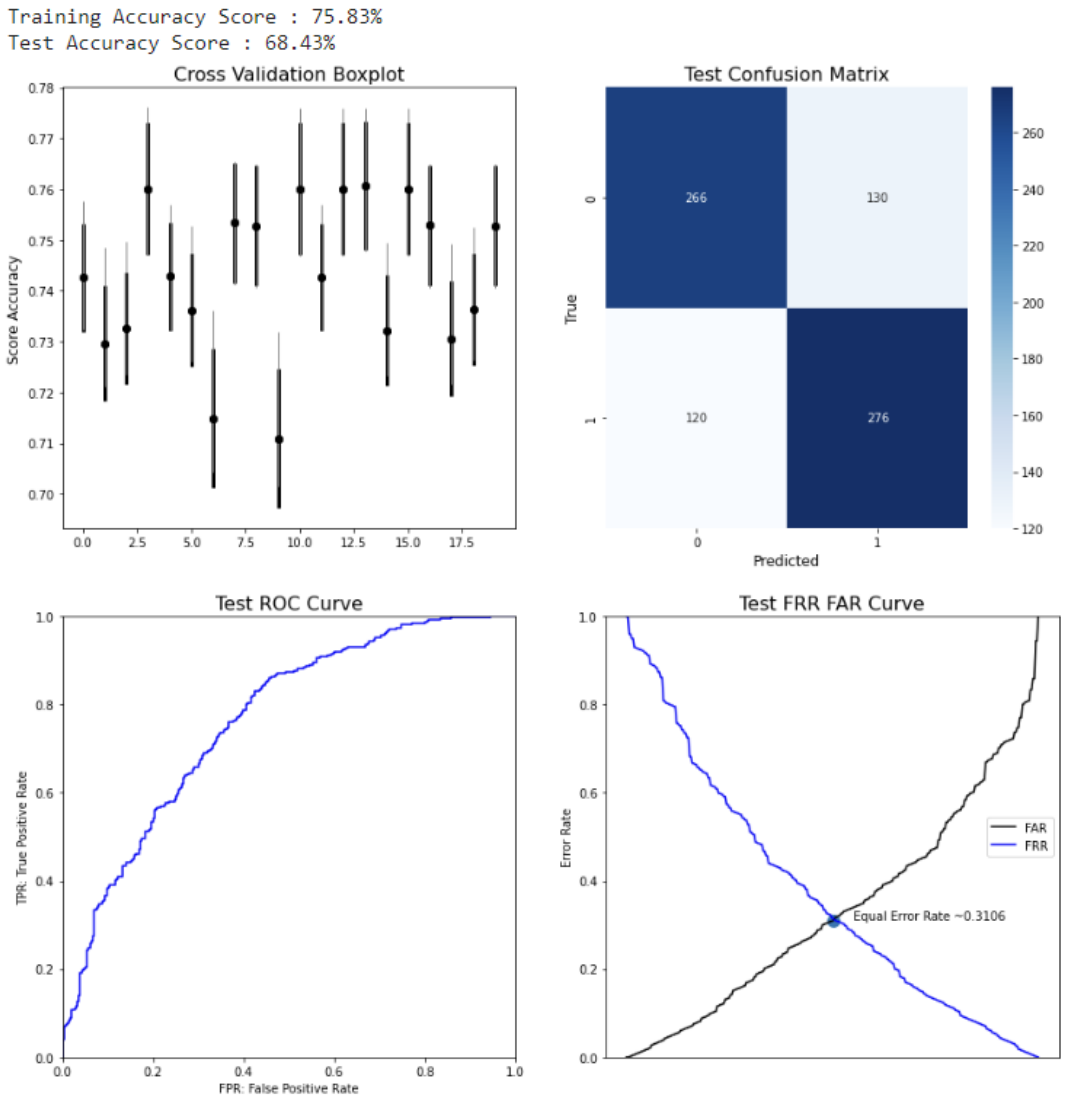


Figure 32: Results for Logistic Regression

When

Support Vector Machine

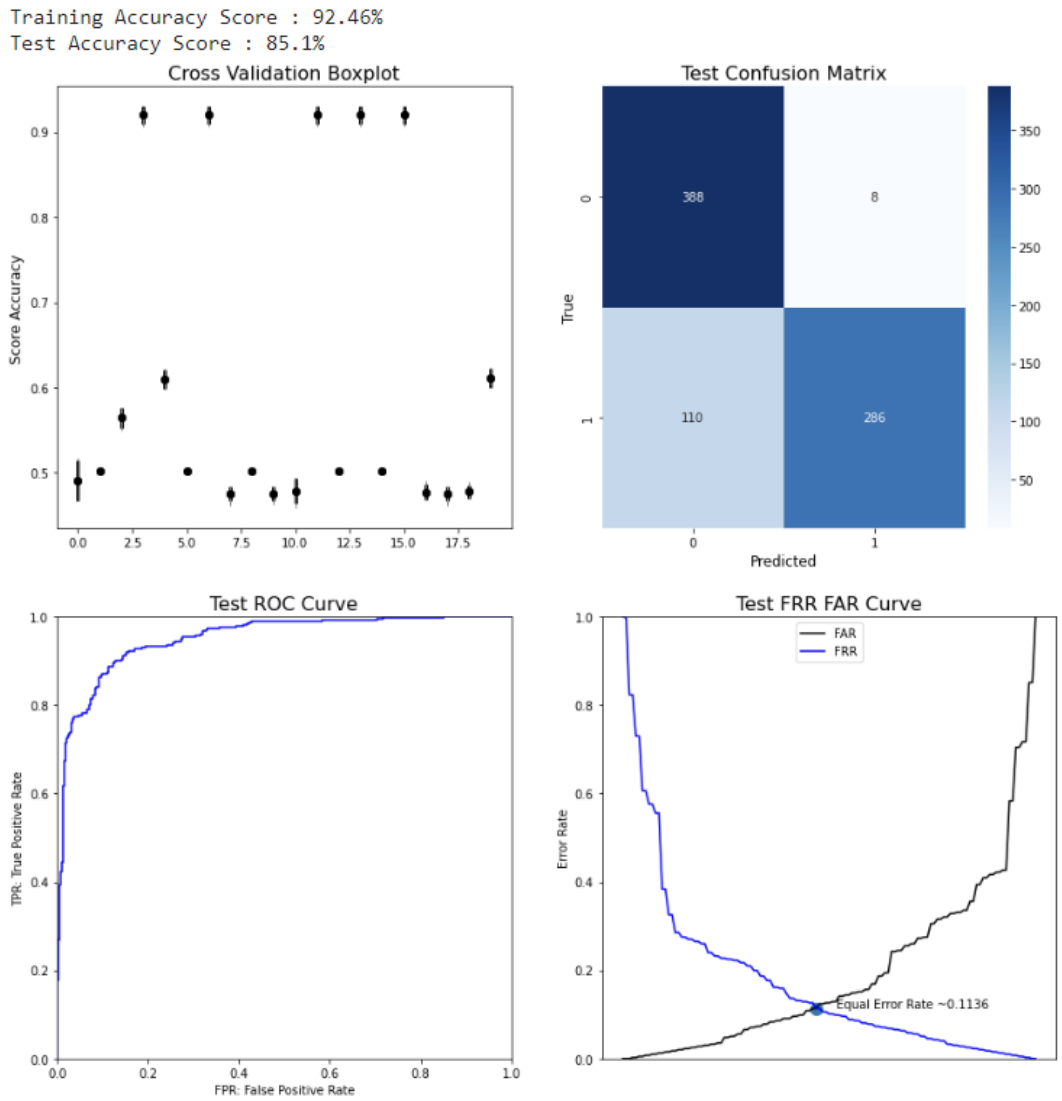


Figure 33: Results for SVM

5.4 Multi-Layer Perceptron

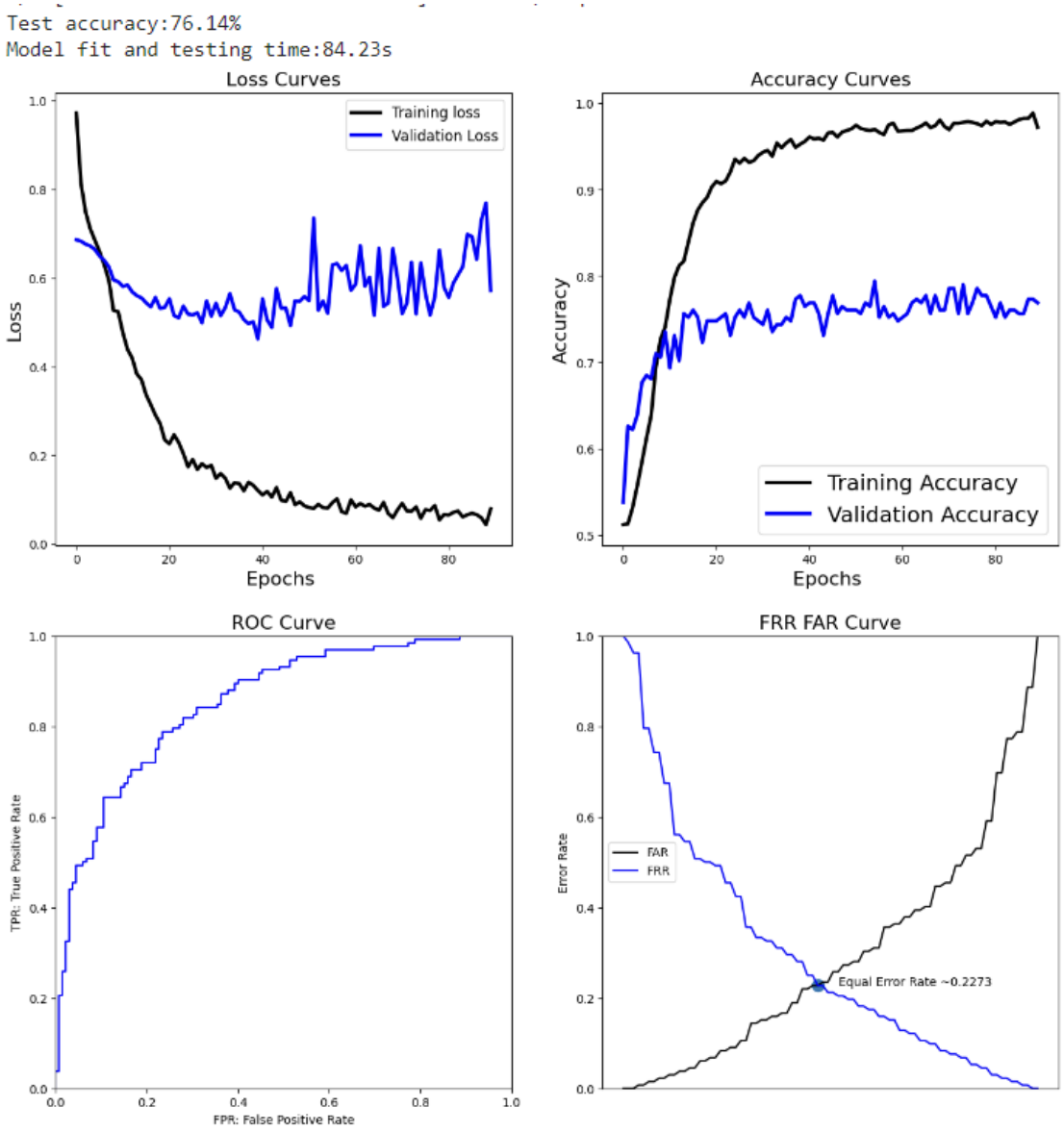


Figure 34: Results for MLP

5.5 Deep Convolutional Network

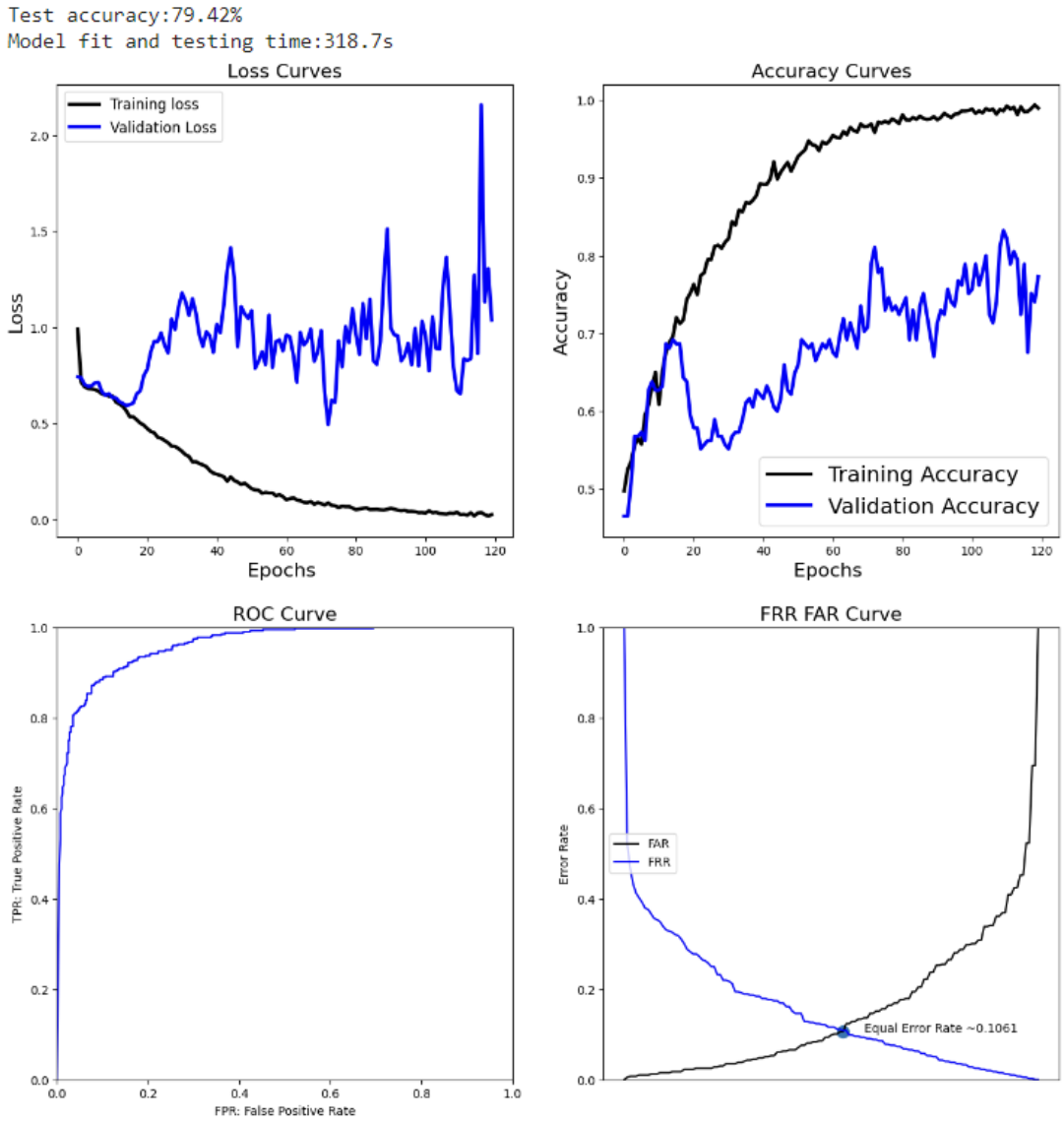


Figure 35: Results for CNN

6.2 Significance of the findings

6.3 Limitations

Chapter 6 Conclusions and Future Work

6.1 Conclusions

In this project writer-independent offline signature verification is implanted and tested. A comparison between multiple machine learning classifiers we established. Each of which implemented data pre-processing stage which improved the image appearance. Feature extraction using texture analysis such as Gabor filters and Local Binary Patterns, and as well as edge detection kernels like Sobel and Prewitt. After extracting these numerous features we selected the most important ones using principal component analysis, limiting the data to about 1200 dimensions from around 80000. We found that logistic regression performed with an accuracy of 68% on the test data with an EER of 31%. Followed by the MLP model with 77% accuracy and EER of 23%. The best model performance belonged to SVM with an accuracy of 85.1% on test set with an EER of 11.3%. This model performed similar to the CNN architecture we've developed, which has an accuracy of 79.4% and an EER of 10.61%

6.2 Future work

This section is intended to provide suggestion of further work and studies which can be a continuation of this project.

6.2.1 Data Augmentation

Signature databases are limited in what they can collect, the entire process require a lot of time and effort to do. To compensate for that we can consider applying mathematical transformations on the data set to augment the data and increase our training sample. This can act as a regulizer and help reduce overfitting when training the machine learning model. Another technique that can be considered is by implement generative adversarial networks (GAN) to create new synthetic images for augmentation.

6.2.2 Using Transfer Learning for Feature Extraction

The process of extracting features from the images takes a long time due to it being limited to CPU processing. We can't also guarantee that the combination of feature extraction methods I implemented extracted the best information to train our classifiers on. In future work, it would be interesting to use an established deep learning CNN architecture to extract features for us and combine it with the SVM Classifier.

6.2.3 Deeper Network Architecture

In this project there wasn't enough time to variate the structure of the final model and analyse further. Despite tuning the CNN model builder to create a better architecture, we still could only create one has 79% accuracy and is comparable to a SVM model we made. This is likely due to our hardware being a limiting factor in training and building a model. In future work, it would be interesting to see a deeper structure CNN that is able to utilise GPU parallel computing effectively.

Chapter 7 Reflection

Coming from a mathematical orientated degree where research and writing were close to non-existent, I was confused and uncertain on how to approach the dissertation process at first. I lacked the ability of research skills and didn't know where to look or how to reference a paper. Although, throughout the journey there were time I found very enjoyable, most specifically the coding aspects of the work and learning more about machine learning models. At first I was introduced to these algorithms but never fully understood the in depth theory of it's derivation.

References

Ahmed Abdelrahaman, A. A. (2013). K-Nearest Neighbor Classifier for Signature .

Ali Karouni, B. D. (2011). Offline signature recognition using neural networks approach. *Procedia Computer Science, 3*, 155-161.

Anil K. Jain, A. R. (2004). An Introduction to Biometric Recognition. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, 14*(1), 4-20.

Department for Digital, Culture, Media & Sport and Home Office. (2018, May 23). *https://www.gov.uk/government/collections/data-protection-act-2018*. (GOV.UK) Retrieved from https://www.gov.uk/government/collections/data-protection-act-2018

Hemant B Kekre, V. A. (2010). Gabor Filter Based Feature Vector for Dynamic Signature Recognition. *International Journal of Computer Applications*.

Hurieh Khalajzadeh, M. M. (2012). Persian signature verification using convolutional neural networks. *International Journal of Engineering Research and Technology, 1*(2), 7-12.

Ian T. Jolliffe, J. C. (2016). Principal Component Analysis.

Ian T. Jolliffe, J. C. (2016). Principal component analysis: a review and recent developments.

IBM Cloud Education. (2020, August 17). *Neural Networks*. (IBM) Retrieved from https://www.ibm.com/cloud/learn/neural-networks

Image-Based Handwritten Signature Verification Using Hybrid Methods of Discrete Radon Transform, P. C. (2015). Shih Yin Ooi, Andrew Beng Jin Teoh, Ying Han Pang, Bee Yan Hiew.

J.A Du Perez, J. C. (2004). Offline Signature Verification Using the Discrete Radon Transform and a Hidden Markov Model. *EURASIP Journal on Advances in Signal Processing*.

Jahandada, S. M. (2019). Offline Signature Verification using Deep Learning Convolutional Neural Network (CNN) Architectures GoogLeNet Inception-v1 and and Inception-v3 . *The Fifth Information Systems International Conference*.

Jivesh Poddar, V. P. (2020). Offline Signature Recognition and Forgery Detection using Deep Learning. *Procedia Computer Science, Volume 170*, Pages 610-617.

K. R. Radhika, M. K. (2010). Off-Line Signature Authentication Based on Moment Invariants Using Support Vector Machine. *Journal of Computer Science*.

M. Thenuwara, H. R. (2017). Offline handwritten signature verification system using random forest classifier. *Advances in ICT for Emerging Regions (ICTer)*, pp. 1–6.

Melo, V. &. (2017). *Datasets for handwritten signature verification: A survey and a new dataset.*

Mohammed, I. S. (2019). Handwritten signature recognition with Gabor filters and neural network. *AIP Conference Proceedings 2096*.

Moises Diaz, M. A. (2019). A Perspective Analysis of Handwritten Signature Technology. *ACM Computing Surveys, 51*(6), 1-39.

Mustafa Berkay Yilmaz, B. Y. (2011). Offline Signature Verification Using Classifier Combination of HOG and LBP Features.

Pallavi V. Hatkar, Z. J. (2015). Image Processing for Signature Verification . *International Journal of Innovative Research in Computer Science & Technology (IJIRCST)*, Volume 3.

Pietikäinen, M. (2010). Local Binary Patterns. *Scholarpedia*, Volume 5, Number 3. Retrieved from http://www.scholarpedia.org/.

Shih Yin Ooi, A. B. (2015). Image-Based Handwritten Signature Verification Using Hybrid Methods of Discrete Radon Transform, Principal Component Analysis and Probabilistic Neural Network.