ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN FACULTY OF ELECTRICAL ENGINEERING
ELECTRONIC AND COMMUNICATION SYSTEMS INSTITUTE
INSTRUMENTATION AND AUTOMATION DEPARTMENT

FINAL ESSAY

OE-KVK
2023

Adilkhanov Magzhan
T009657/FI12904/K

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbuda University
Kandó Kálmán Faculty of Electrical Engineering

Institute of Instrumentation and Automation

THESIS ASSIGNMENT

Student's surname, forename (s): Magzhan Adilkhanov
Thesis number: SZD2309211031296962[redacted]
Registration number: T009657/FI12904/K **Neptun code:** [redacted]

Branch of study:
Specialization: Instrumentation and automation specialisation

Thesis title: Verification of electronic signature
Task description: This project leverages machine learning techniques to bolster the authentication of electronic signature. It involves the development of a sophisticated algorithm that can analyze and validate electronic signatures, taking into account various parameters such as stroke dynamic, pressure, and unique writing style. By training the machine learning model on a diverse dataset of genuine and forged signatures, the project aims to significantly enhance the accuracy and reliability of electronic signature authentication.

Institutional consultant's name: Árpád Varga

The limitation period of the theme issued: 01.01.0001.
Deadline for submission of Thesis: 15.12.2023.

The thesis is: Not secreted.

Issued: Budapest, 24.10.2023.

P.H

Director of Institute

The Thesis is suitable for submission:

Varga Árpád
Institutional consultant

External consultant

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN FACULTY OF ELECTRICAL ENGINEERING
ELECTRONIC AND COMMUNICATION SYSTEMS INSTITUTE
INSTRUMENTATION AND AUTOMATION DEPARTMENT

STUDENT STATEMENT

I hereby declare that the present final thesis paper is solely my own work and that the references (written and electronic literature and tools used during the process) have been acknowledged and fully cited. I hereby grant the university and the task assigning institution permission to use free of charge the final results of this paper for their own purposes, abiding by the pertaining rules of encryption.

Budapest, 14.12.2023…

.....
Signature of student

Abstract

The use of offline biometric analysis, namely handwritten signature authentication, has seen a significant rise in recent years, particularly within the last decade. The use of signatures as a method of personal verification underscores the need for automated authentication systems. This is due to its regrettable susceptibility to exploitation by those who manipulate someone's identity or intentions. In recent decades, much study has been carried out in the domain of signature authentication.

The process of validation may be conducted either offline or online, depending on the specific application. The online method uses dynamic signature data that is recorded at the time of signature creation. The offline system analyses the scanned picture of the signature. The procedure of verifying offline handwritten signatures is a challenging undertaking. Individuals exhibit within-user variation when it comes to signing their signature, meaning that they seldom produce a same signature on every occasion.

We have successfully developed a signature recognition system that operates without an internet connection. Prior to performing the technique, it is necessary to preprocess the scanned picture by separating the signature section and eliminating any existing noise. The objective of this project is to suggest a feature vector for offline handwritten signatures utilising an effective approach called Histogram Oriented Gradients (HOG) as a robust method for extracting features. These features will then be inputted into an artificial neural network for identification tasks. The suggested method was tested utilising a local database consisting of over 48 authentic signature samples and 48 counterfeit signature samples from four individuals, in order to assess its accuracy and performance. The findings demonstrated a precision level of 96.8% as the effectiveness was attributed to the following categories of errors: the False Accept Rate (FAR) amounted to 3% and the False Rejection Rate (FRR) reached 3.35%.

Table of Contents

Abstract 1

List of figures 4

List of tables 5

Chapter I: Introduction 6

Chapter II: Methodology 7

2.1. Image Preprocessing and Feature Extraction 7

2.2. Data preprocessing 7

2.2.1. Reduction of Noise 7

2.2.2. Inversion of Colour 8

2.2.3. Monochromatic Images 9

2.2.4. Application of Image Filtering and Binarization Techniques 9

2.3. Feature Extraction 10

2.3.1 The Histogram Oriented Gradient (HOG) method is used for image analysis. 10

2.3.2 Gradient Calculation 11

2.3.3 Directional Categorization 11

2.3.4 Descriptor Blocks 12

2.3.5 Object Detection refers to the process of identifying and locating objects inside an image or video. 13

2.4. Artificial Neural Networks 13

2.4.1 Fundamental Architecture of Artificial Neural Network 14

2.4.2 Categories of Artificial Neural Networks 15

2.4.3 The Mechanism of Artificial Neural Networks 16

2.4.4 Machine Learning using Artificial Neural Networks (ANNs) 16

2.4.5 Mathematical backround 17

2.4.6 Bayesian Networks (BN) 23

2.4.7 Utilisations of Neural Networks 23

2.4.8. Artificial Neural Network Training 24

2.5. In conclusion 25

Chapter III: Experimental Procedures and Findings 27

3.1 Experimental Procedures 27

3.2 Results 28

3.3 Program code and comments 29

3.3.1 Jupyter Notebook Signature Verification 29

3.3.2 Pre-processing 36

Chapter IV: Summary 39
References 40

List of figures

Figure 1 Signature after noise reduction 8
Figure 2. Signature before noise reduction..... 8
Figure 3 Grayscale signature 8
Figure 4 Signature to be processed 9
Figure 5 Binarized signature..... 10
Figure 6 Implementation of the HOG algorithm 12
Figure 7 Pixel-by-pixel picture histogram..... 13
Figure 8 Feedforward ANN..... 15
Figure 9 Feedback ANN..... 16
Figure 10 Perceptron 17
Figure 11 Neural network 18
Figure 12 Categories of activation functions..... 19
Figure 13 Forward Propagation..... 20
Figure 14 Chain rule (formula) 21
Figure 15 Chain rule..... 21
Figure 16 Changing weights 22
Figure 17 Backpropagation 22
Figure 18 New weights 22
Figure 19 Code block pt. 1 (signature verification)..... 29
Figure 20 Code block pt. 2 (signature verification)..... 30
Figure 21 Code block pt. 3 (signature verification)..... 31
Figure 22 Code block pt. 4 (signature verification)..... 33
Figure 23 Code block pt. 5 (signature verification)..... 34
Figure 24 Code block pt. 6 (signature verification)..... 35
Figure 25 Code block pt. 1 (preprocessing)..... 37
Figure 26 Code block pt.2 (preprocessing) 37

List of tables

Table 1 Specifications for Neural Networks 25
Table 2 Provided algorithm's performance 28
Table 3 How the provided algorithm has performed 28

Chapter I: Introduction

Traditional bank checks, bank loans, credit cards, and numerous legal papers are essential components of the contemporary economy. They serve as a primary means for people and organisations to transfer funds and settle financial obligations. Currently, all of these transactions, particularly those related to finances, still need signature validation. An inevitable consequence of signatures is that they might be used to falsely portray the genuineness of the document. Hence, in recent times, there has been a growing need to investigate effective automated methods for the detection and verification of signatures in order to mitigate fraud.

Signature verification methods may be categorised into two groups based on the method of data collection: online and offline. The online data captures the pen's motions while creating the signature, including its location, and in some instances, its velocity, acceleration, and pressure over time. The online system utilises the data gathered throughout the gathering process. Online systems are capable of facilitating real-time applications, such as credit card transactions and resource access. In contrast, offline signature verification methods use a two-dimensional picture of the signature as their input. Offline systems facilitate the automated verification of signatures on bank checks and documents. Furthermore, it is important to devise resilient systems capable of identifying various categories of counterfeit merchandise, in addition to taking these considerations into account.

We tackle this issue with a two-step process. Initially, the scanned signature picture undergoes preprocessing to ensure its suitability for feature extraction. The preprocessed photos are used to construct a model using Artificial Neural Networks (ANN).

Chapter II: Methodology

2.1. Image Preprocessing and Feature Extraction

We tackle this challenge with a two-step process. Initially, the scanned signature picture undergoes preprocessing to ensure its suitability for model generation. The preprocessed pictures are then used to construct a model using an Artificial Neural Network (ANN) methodology, enabling the differentiation between counterfeit and authentic signatures.

2.2. Data preprocessing

The signature is first acquired and transformed into a computer-readable format. You are now prepared for preprocessing. During the preprocessing stage, the RGB picture of the signature undergoes a conversion procedure where it is transformed into a grayscale image and then into a binary image. The preprocessing phase includes noise reduction, colour inversion, filtering, and binarization.

2.2.1. Reduction of Noise

Image denoising is a key problem in the area of image processing and computer vision. The primary objective is to approximate the original picture by attenuating noise in a noisy image. Image noise may arise from several internal factors (such as sensors) and external factors (such as the environment) and is often inevitable in real-world scenarios. Hence, image denoising assumes a crucial function in diverse applications including image restoration, visual tracking, image registration, image segmentation, and picture classification. Preserving the original picture content is crucial for achieving optimal performance in these applications. Despite several suggested algorithms for picture denoising, the task of effectively reducing noise in photos remains unresolved, particularly in cases when the images are captured under adverse circumstances with a high degree of noise.

A novel method for addressing image denoising problems is proposed, which utilises data adaptive stochastic optimisation using Markov Chain Monte Carlo Boosting. By framing the issue as a Bayer optimisation problem and using a nonparametric probabilistic approach, the Markov Chain Monte Carlo Denoising (MCMCD) algorithm [8] dynamically and flexibly adjusts, yielding excellent noise reduction performance. Exhibits a comparatively low level of computational complexity.

The following examples demonstrate the outcomes obtained with the MCMCD technique.

Figure 1 Signature after noise reduction

Figure 2. Signature before noise reduction

2.2.2. Inversion of Colour

A picture that accurately represents the colours of the original subject. The conversion of RGB to a grayscale intensity picture involves the elimination of hue and saturation data, while retaining the brightness component.

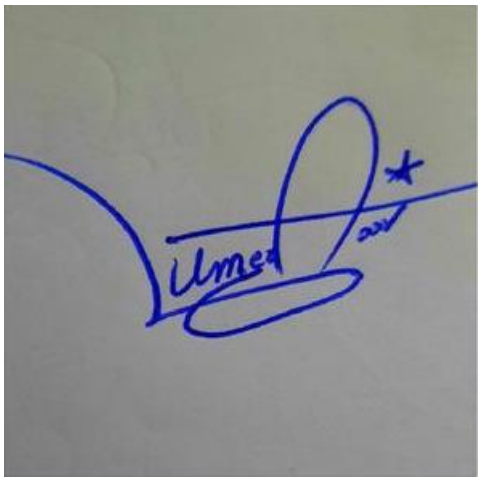

Figure 3 Grayscale signature

Figure 4 Signature to be processed

2.2.3. Monochromatic Images

A grayscale picture is a matrix of data where the values indicate the intensities of pixels within a certain range. Each element of the matrix corresponds to a pixel in the image. It is a monochromatic design with just grey hues. The distinctive characteristic of such pictures is in their reduced pixel information requirements compared to other forms of colour images.

A "grey" colour is characterised by equal intensities of the red, green, and blue components in the RGB colour space. Thus, he just has to designate a single intensity value for each pixel, but he need three intensities to designate each pixel in a full-color picture.

Grayscale intensities are often represented using 8-bit integers, resulting in a total of 256 distinct shades of grey ranging from black to white. If the grey levels are uniformly distributed, the disparity between consecutive grey levels surpasses the resolution of the human eye for grey levels. Grayscale pictures are prevalent due to the limited compatibility of current display and image capturing devices, which mostly accommodate 8-bit images. Furthermore, grayscale photographs are entirely enough for several activities, obviating the need to depend on colour images, which possess greater intricacy and pose challenges in processing.

2.2.4. Application of Image Filtering and Binarization Techniques

Resampling involves applying a low-pass FIR filter to each picture. This is done to prevent aliasing. Aliasing happens when the data is sampled at a frequency lower than twice the highest frequency component of the data. Hence, a low-pass filter eliminates the high-frequency elements of the picture. Filters are used for this objective. In this process, the grayscale picture is divided into segments in order to generate a binary image that represents the item. A binary picture consists of pixels that are assigned either the value of 1 or 0, with no other possible values. Binary pictures are stored as arrays of Boolean values.

Figure 5 Binarized signature

2.3. Feature Extraction

We conducted experiments using two different features. Histogram of Oriented Gradients (HOG, [1]) identifies dominant directions and local binary patterns.

2.3.1 The Histogram Oriented Gradient (HOG) method is used for image analysis.

The histogram-oriented gradient (HOG) method is used for image analysis. The histogram-oriented gradient (HOG) method, presented by Dalal and Triggs at his 2005 CVPR conference [10], is used to represent the shape of the feature.

HOG is an acronym for Histograms of Oriented Gradients and is primarily used as a person detector. In this study, we used histogram of directional gradients (HOG) as a feature extraction technique for signature image recognition and authentication. The HOG descriptor approach essentially quantifies the frequency of gradient directions within a certain localized portion of an image or region of interest (ROI). As shown in Figure 3, the basic implementation of the HOG descriptor is: First, the image is divided into compact, interconnected sections (cells), and for each region a histogram of the gradients or edge directions of the pixels it contains is created. The amount will be calculated. Then, displacements within the cell are performed using the captured gradient directions. Next, individual cells are divided into separate angular segments. Each pixel in the cell contributes a gradient with a specific weight to the associated angular segment. Additionally, adjacent cells within a spatial region are organized into blocks. This serves as the basis for the process of grouping and normalizing the histogram. Finally, the set of standardized histograms corresponds to block histograms, and these set of block histograms correspond to descriptors.

Place images in cells and cells in blocks, taking into account block overlap and normalization settings. This article defines a HOG with a block size of [22 x 22] pixels, a cell size of 128, and

a histogram with 9 bins per cell. Therefore, the total length of the feature vector used to represent each signature image example is 216. Figure 4 shows two offline handwritten signatures with different cell sizes that were used in this work to conduct a comprehensive study of his HOG implementation in offline signatures. Small cells display more pronounced color gradients in amount and direction than larger cells. Gradually increasing the cell size of the HOG parameter reduces the size and direction slope.

Figure 2 shows a graph showing the effect of HOG on offline signature images for cell sizes ranging from 16 to 128.

2.3.2 Gradient Calculation

One of the first computational steps in some feature detectors during image preprocessing is to ensure that the colour and gamma values are normalized. Nevertheless, Dalal and Triggs argue that this particular step in the calculation of the HOG descriptor can be omitted, since the following descriptor normalization yields essentially the same result: Therefore, image preparation has little impact on performance. Alternatively, the first step in the calculation process is to determine the value of the slope. A widely used approach involves using a 1D centred difference mask with horizontal or vertical discrete points. In particular, this approach requires applying a filter kernel to the colour or intensity data of the image to perform filtering.

Dalal and Triggs use 3x3 Sobel filters, diagonal filters, and other similar techniques. We have been experimenting with more complex masks. However, these masks tend to be poor at identifying people in images. Additionally, we also performed Gaussian smoothing experiments before applying the difference mask. However, we found that eliminating the smoothing process actually improves performance [9].

2.3.3 Directional Categorization

The next step in the calculation is to create a cell histogram. Each pixel within a cell contributes a weighted vote to the direction-based histogram channel, determined by the gradient computation value. Cells can have either rectangular or radial shapes, and the histogram channels are evenly distributed from 0 to 180 degrees or from 0 to 360 degrees, depending on whether the slope is unsigned or signed. Dalal and Triggs found that the combination of unsigned gradients and his nine histogram channels produced the best results in attempts to identify people. Regarding the adjustment weight, the pixel contribution can be determined either by the actual size of the gradient or by a function based on the size. Typically, the size of the gradient itself provides optimal results in a study. Other alternatives for weighting votes include using the square root or square of the gradient magnitude, or a shorthand form of the magnitude [9].

2.3.4 Descriptor Blocks

To compensate for variations in illumination and contrast, a local normalization of the gradient strength must be performed. This process combines cells into larger spatially connected blocks. Importantly, the HOG descriptor is a vector that combines the normalized cell histogram components from all block regions. Usually, these blocks tend to overlap each other. Therefore, each cell makes multiple contributions to the final description. The two main block shapes are the rectangular R-HOG block and the circular C-HOG block. R-HOG blocks are square grids characterized by his three parameters: number of cells per block, number of pixels per cell, and histogram of channels per cell. The ideal parameters, determined by Dalal and Triggs' human discrimination test, were to use four 8x8 pixel cells per block (16x16 pixels per block) with nine histogram channels.

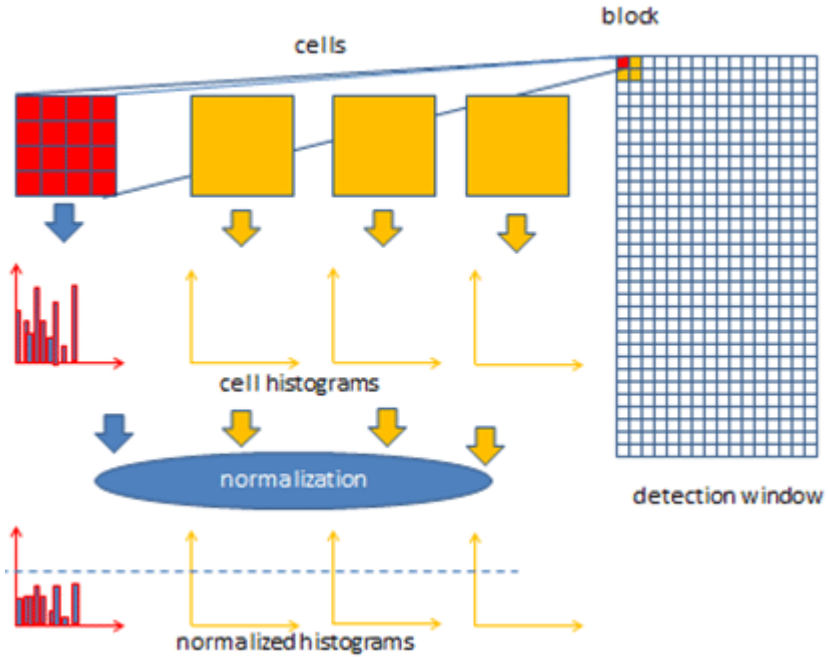

Figure 6 Implementation of the HOG algorithm

Additionally, we found that implementing a Gaussian spatial window for each block before aggregating the histogram votes slightly improved performance. This approach makes pixels at the edges of the block less important. The R-HOG block is very similar to a scale-invariant feature transform (SIFT) descriptor. However, there are differences between the R-HOG block and how his SIFT descriptor is calculated. R-HOG blocks are computed on a dense grid without orientation adjustment, whereas SIFT descriptors are typically computed on specific pixels with scale-invariant rotation and orientation adjustment. Furthermore, the R-HOG blocks are used together to encode spatial shape information, whereas the SIFT descriptors are used separately. There are two versions of the circular HOG block (C-HOG). Blocks consisting of a single central cell and blocks in which the central cells are separated from each other by an angle. Furthermore, these C-HOG blocks can be characterized by four parameters: the number of angular and radial bins, the size of the central bin, and the radius magnification of the additional radial bins. Dalal and Triggs found in their experiments that the performance of the

two main variants was similar. Specifically, we found that a version with 2 radius bins and 4 angular bins, a centre radius of 4 pixels, and a expansion factor of 2 produced the best results. Ultimately, for best performance use: Furthermore, the Gaussian weighting applied to the C-HOG block did not provide any benefit. C-HOG blocks share similarities with shape context descriptors in that both are composed of cells containing many directional channels. Furthermore, both very clearly refer to the concept of form [9].

2.3.5 Object Detection refers to the process of identifying and locating objects inside an image or video.

Object identification may be achieved by using HOG descriptors as features for machine learning techniques. Dalal and Triggs used the Histogram of Oriented Gradients (HOG) descriptor as a feature in conjunction with a support vector machine (SVM) [9]. Nevertheless, HOG descriptors are not inherently associated with certain machine learning techniques. It is important to understand that while complex features provide a greater amount of information, basic features like slope adjustments are more resistant to typical fluctuations in the signature.

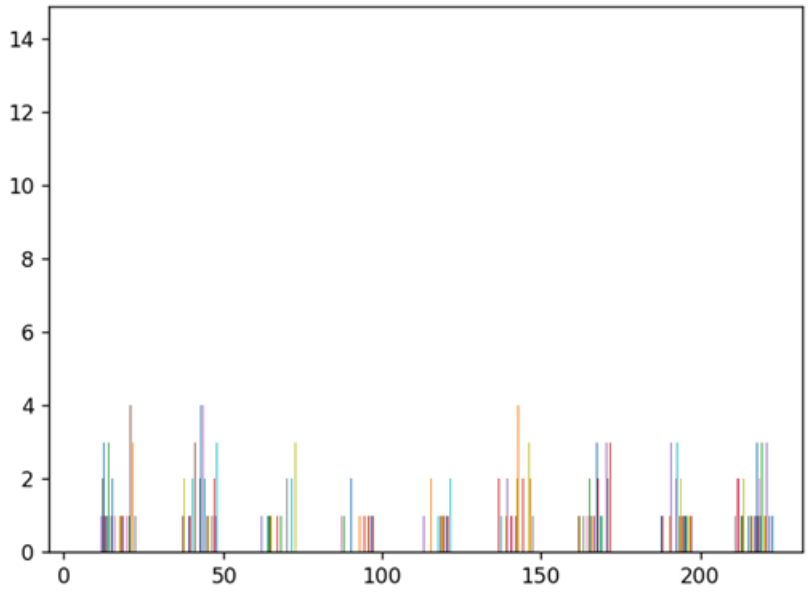

Figure 7 Pixel-by-pixel picture histogram

2.4. Artificial Neural Networks

Artificial Neural Networks (ANN), also known as connectionist systems, are computer systems that are loosely based on the biological neural networks seen in animal brains. A neural network is not an algorithm per se, but rather a framework that enables the collaboration of many

machine learning algorithms to handle intricate data inputs. These systems usually acquire the ability to do tasks by learning from examples, rather than being coded with particular rules for each activity. Image recognition, for instance, acquires the ability to recognise photos that include cats by examining manually annotated sample images categorised as either "cat" or "not cat". It then uses these outcomes to identify photographs that include cats. They do so without any previous acquaintance with feline characteristics, such as hair, tails, whiskers, and cat-like facial features. Conversely, it autonomously produces distinguishing characteristics from the analysed educational resources.

Artificial neural networks (ANNs) consist of linked units or nodes called artificial neurons that imitate the functionality of neurons in the man's brain. Every link has the ability to transmit messages between artificial neurons, functioning like to synapses in a real brain. The signal received by the artificial neuron may be processed and then sent to other interconnected artificial neurons.

In a standard artificial neural network (ANN) implementation, the connections between artificial neurons transmit actual signals, and the output of each artificial neuron is determined by a nonlinear function that calculates the sum of its inputs. The term used to describe the connections between artificial neurons is "edges." Artificial neurons and connections usually possess weights that are modified throughout the process of learning. Weights modulate the intensity of the signal on a link.

An artificial neuron may be configured with a certain threshold, so that it will only transmit a signal if the cumulative total of incoming signals above this threshold. Artificial neurons are usually arranged in hierarchical layers. Specific layers possess the capacity to enact various modifications to their inputs. The signal traverses from the input layer to the output layer, possibly undergoing several iterations throughout the layers.

Initial aim of the artificial neural network (ANN) method was to tackle the issue by using a technique that closely resembles the functioning of the human brain. Nevertheless, as time passed, focus gradually switched towards the execution of certain tasks, resulting in a departure from the realm of biology. Artificial neural networks serve several functions, such as image and speech recognition, predictive analytics, autonomous vehicles, robotics, financial applications, and drug discovery.

2.4.1 Fundamental Architecture of Artificial Neural Network

The concept of Artificial Neural Networks (ANN) is founded on the premise that the intricate operations of the human brain may be replicated by using silicon and wires as artificial neurons and dendrites and establishing the necessary connections between them.

The human brain consists of around 86 billion neurons, which are specialised nerve cells. The cells are interconnected with a network of a thousand additional cells by axons. Dendrites receive stimuli from the external world and input from sensory organs. These inputs produce electrical impulses that rapidly propagate across the brain network. The neuron has the ability

to transmit the message to another neuron for processing, or it may be unable to transmit it at all.

ANN is consisted of multiple nodes that copy behaviour of the organic neurons found in man's brain. Neurons are connected by synapses, participate in mutual communication. Nodes have the capability to receive input data and carry out basic actions on that data. The outcomes of these activities are sent to other neurons. The result produced by each node is referred to as an activation or node value.

Every connection is allocated a weight. Artificial neural networks (ANNs) have the ability to acquire knowledge via the modification of weight values. The above diagram illustrates a basic Artificial Neural Network (ANN).

2.4.2 Categories of Artificial Neural Networks

Artificial neural networks consist of two distinct topologies: feedforward and feedback.

2.4.2.1 The process of transmitting information or signals in a forward direction. Artificial Neural Network (ANN)

This artificial neural network (ANN) exhibits a unidirectional information flow. One unit transmits information to another unit that fails to receive the information. There is an absence of a feedback loop. These are used for the purpose of generating, recognising, and classifying patterns. The entrance and egress are stationary.

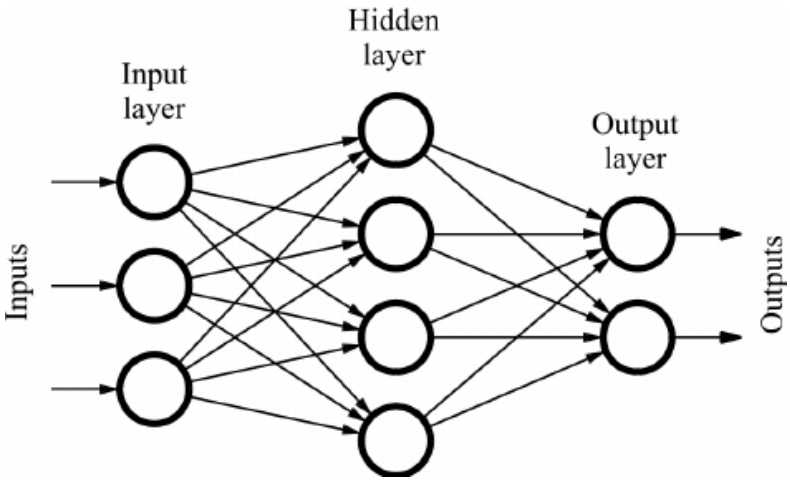

Figure 8 Feedforward ANN

2.4.2.2 Feedback Artificial Neural Network (ANN)

Feedback loops are permitted in this context. These are used for content addressable storage.

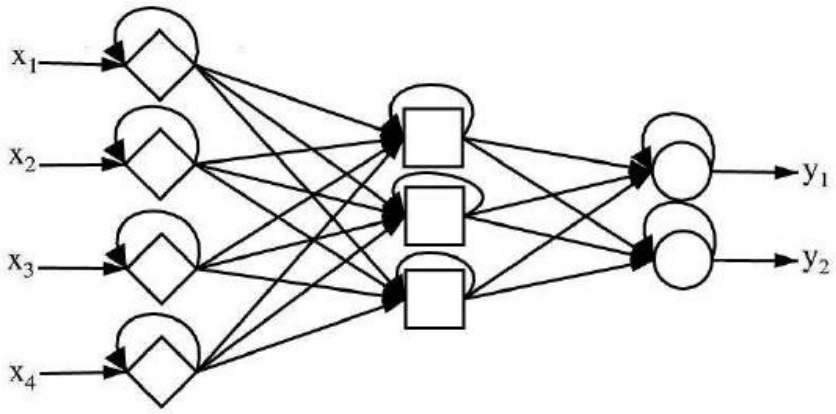

Figure 9 Feedback ANN

2.4.3 The Mechanism of Artificial Neural Networks

The topology diagram illustrates the connections between neurons, with each arrow denoting a pathway for the transmission of information. The weight of each link, represented by a number, regulates the signal transmission between the two neurons.

If the network generates an output that is considered satisfactory or acceptable, there is no need to modify the weights. Nevertheless, in the event that the network generates an output that is considered suboptimal or unpleasant, the system adjusts the weights in order to enhance future outcomes.

2.4.4 Machine Learning using Artificial Neural Networks (ANNs)

Artificial neural networks (ANNs) are capable of acquiring knowledge and need a process of training. Multiple learning styles exist:

• Supervised learning refers to a process in which a teacher, who has more scientific knowledge than the artificial neural network (ANN), guides and instructs the network. As an example, the instructor inputs sample data for which the solution is already known. As an example, the process of identifying patterns. ANN employs assumptions throughout the process of recognition. Subsequently, the instructor provides the solution to the Artificial Neural Network (ANN). The network then compares its prediction with the instructor's "accurate" response and makes adjustments in the event of a mistake.

• Unsupervised learning is necessary in situations when there is no available dataset with pre-determined answers. For instance, seek for concealed patterns. Clustering is the process of

categorising a collection of components into groups based on an unknown pattern. This is done using an existing data set.

• Reinforcement Learning - This approach relies on the process of observing and learning from experiences. Artificial neural networks (ANNs) perceive the surrounding environment and then formulate choices. When the observation is negative, the network modifies the weights in order to enable it to make a distinct choice in the future.

2.4.5 Mathematical background

2.4.5.1 Perceptron

The perceptron is a computational model used in machine learning and artificial intelligence.

A perceptron refers to a basic artificial neuron that consists of an input layer and an output layer. What is the content of this neuron?

1. A function that calculates the sum of a series of numbers.
2. A function used to determine the output of a neuron or node in a neural network.

The perceptron receives an input that is computed by a summation function, which is then processed by an activation function to get the intended outcome.

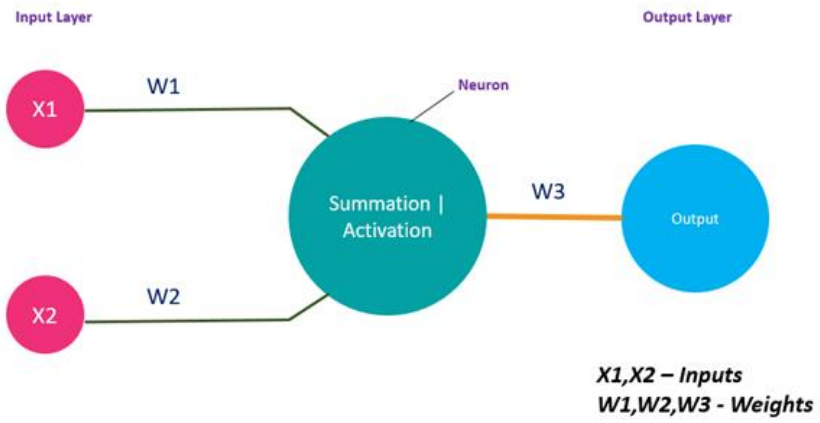

Figure 10 Perceptron

This perceptron is rudimentary, but what if you encounter a substantial volume of input and data? Is it correct to say that a single perceptron is insufficient? We must continue to include more neurons. Presented is a rudimentary neural network with an input layer, a hidden layer, and an output layer.

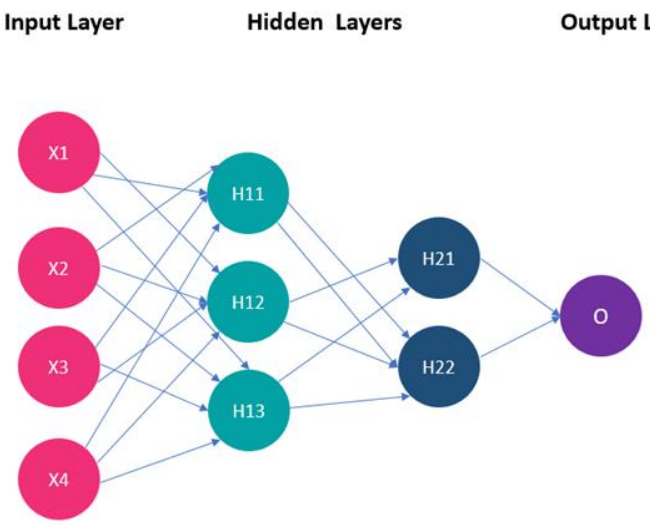

Figure 11 Neural network

Artificial neural networks Neural networks consist of a solitary input and output layer, although they might possess several hidden layers. The above diagram illustrates a neural network with one input layer, two hidden layers, and a single output layer.

An activation function is a crucial component of neural networks. It determines the output of a neuron based on the weighted sum of its inputs. There are several kinds of activation functions, including sigmoid, tanh, ReLU, and SoftMax.

2.4.5.2 Activation function

The activation function is a mathematical function used to determine the output of a neural network node.

The primary function of the activation function is to transform the aggregated input signals of a neuron into an output signal. The output signal is then used as the input for the subsequent layer. It is necessary for all activation functions to be differentiable in order to use a backpropagation method for error reduction and weight updates.

Categories of activation functions:

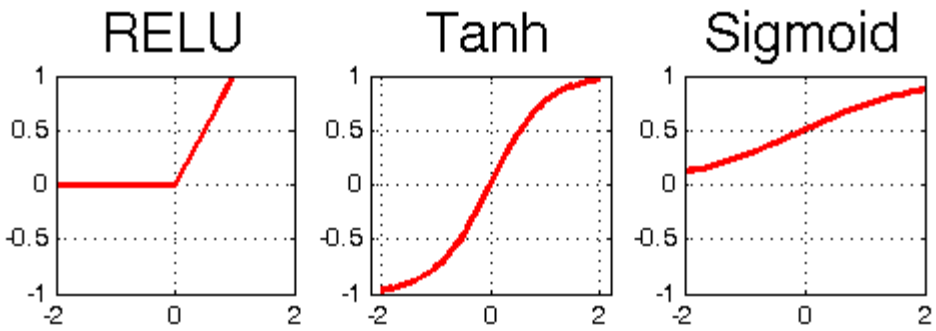

Figure 12 Categories of activation functions

Sigmoid

1. The range spans from 0 to 1.

2. A little variation in x leads to a significant alteration in y.

3. Commonly used in the last layer of binary classification models.

The hyperbolic tangent function

1. Has a range that spans from -1 to 1.

2. The output values are symmetrically distributed around zero.

3. Typically used on concealed layers.

Rectified Linear Unit (RELU)

1. The range spans from 0 to the maximum value of x.

2. The computation cost is lower compared to the sigmoid function and Tann function.

3. Default activation function for hidden layers.

4. May induce neuronal apoptosis, which may be mitigated by implementing a Leaky RELU activation mechanism.

Thus far, we have acquired knowledge on the prerequisites of perceptron and the role of activation functions. Now, let's examine the functioning of neural networks, which serve as the fundamental component of these networks.

2.4.5.3 The Mechanism of Neural Networks

Neural networks are founded upon two fundamental principles:

1. Propagation in the forward direction

2. Backpropagation is a computational method used in artificial neural networks to calculate the gradient of the loss function with respect to the weights of the network. It involves propagating the error backwards via

Let us comprehend these elements with the use of an illustration. To enhance comprehension, we shall examine a solitary input layer, concealed layer, and output layer.

Forward propagation refers to the process of passing input data through a neural network in order to generate an output. It involves the calculation of weighted sums and the application of activation functions to provide predictions or activations.

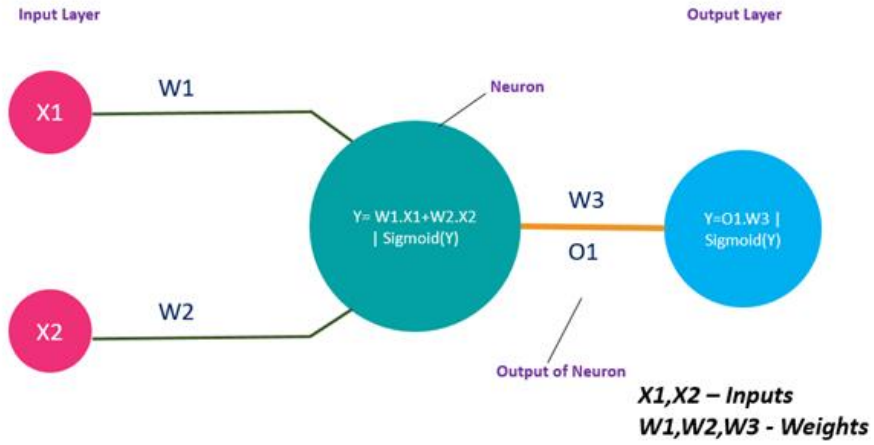

Figure 13 Forward Propagation

1. We possess data and want to use binary classification in order to get the desired outcome.

2. A sample is collected, including characteristics such as X1, X2, etc. These features undergo a sequence of operations to forecast the result.

3. A weight is applied to each characteristic. In this context, X1 and X2 represent the features, whereas W1 and W2 represent the corresponding weights. These function as inputs to neurons.

4. Neurons carry out both jobs. a) Addition b) Triggering.

5. During the process of summing, each feature is multiplied by its corresponding weight, and the biases are then added together. The equation is represented as Y equals W1 times X1 plus W2 times X2 plus b.

6. The summing function is implemented by means of an activation function. The neuron's output is multiplied by weight W3 and then used as input for the output layer.

7. The same procedure takes place in all neurons; however, we modify the activation function in the neurons of the hidden layer rather than the output layer.

The weights were initialised randomly, and the procedure was resumed. There are several methods for initialising weights. However, how are these weights updated? The solution to this problem is achieved by the use of the backpropagation algorithm.

Backpropagation

Let us revisit the fundamental principles of calculus and modify the weights by using the chain rule.

Principles of Chain Rule

The chain rule offers a method for determining the derivative of a complicated function. The quantity of functions involved in the composition dictates the quantity of differentiation steps necessary. For instance, if the composite function f(x) is defined as

$$\frac{d}{dx}\left[(f(x))^n\right] = n(f(x))^{n-1} \cdot f'(x)$$

$$\frac{d}{dx}\left[f(g(x))\right] = f'(g(x))g'(x)$$

Figure 14 Chain rule (formula)

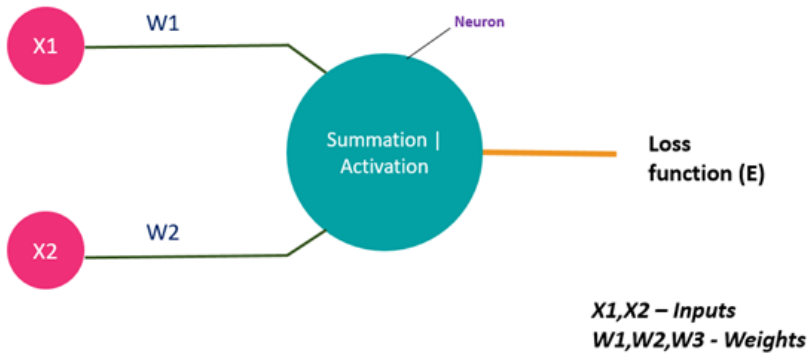

Figure 15 Chain rule

The objective is to minimise inaccuracies. If feasible, it is recommended to update all weights using the backpropagation algorithm. We must determine the alteration in weights in order to minimise the inaccuracy. To do this, compute the partial derivatives of E with respect to W1 and W2, denoted as dE/dW1 and dE/dW2, respectively.

Considering S (Summation) = x1W1 + x2W2

A (Activation) = sigmoid = e^x / $(1 + e^x)$

Using Chain Rule

$$\frac{dE}{dW1} = \frac{dE}{dA} \times \frac{dA}{dS} \times \frac{dS}{dW1}$$

$$\frac{dE}{dW2} = \frac{dE}{dA} \times \frac{dA}{dS} \times \frac{dS}{dW2}$$

Figure 16 Changing weights

Figure 17 Backpropagation

After determining the change in weights in relation to the mistake, the subsequent action is to modify the weights using gradient descent.

$$W1new = W1old - \eta \frac{dE}{dW1}$$

$$W2new = W2old - \eta \frac{dE}{dW2}$$

Figure 18 New weights

Forward and backward propagation iterates continuously for all samples until the error converges to a minimal value.

2.4.6 Bayesian Networks (BN)

Graphical frameworks are used to depict probabilistic connections among a group of random variables. Bayesian networks are also referred to as belief networks or Bayes nets. The security of BN domains is compromised. Within these networks, every node corresponds to a random variable that has a specific statement. For instance, in the realm of medical diagnostics, a cancer node signifies the affirmation that a patient is afflicted with cancer. The edges that link the nodes indicate the probabilistic relationships between these random variables. If there is a cause-and-effect relationship between two nodes, they must be directly linked in the direction of the influence.

The degree of correlation between variables is measured by the probability linked to each node. The BN arc is subject to a single constraint. It is not possible to trace a unidirectional arc back to a node. Hence, BN is referred to as a directed acyclic graph (DAG).

BN has the capability to concurrently manage variables with multiple values. The BN variable has two dimensions:

• The scope of prepositions

• The probability attributed to each preposition.

Let's examine a limited collection of distinct random variables X = {X1, X2, …, Xn}. Each variable Xi is capable of assuming a value from a limited set represented by Val (Xi). If there exists a directed link from variable Xi to variable Xj, then variable Xi is considered a parent variable of variable Xj, signifying a direct interdependence between the variables.

The BN's structure is well-suited for integrating previous knowledge with observed data. BN enables the acquisition of causal links, comprehension of many issue domains, and anticipation of future occurrences, even in the absence of complete data.

2.4.7 Utilisations of Neural Networks

Neural networks excel at activities that are simple for people but challenging for robots.

• Aerospace − Autopilot systems for aeroplanes, defect detection systems for aircraft.

• Automotive - Control systems designed for autos.

• Military applications include weapon targeting and control, target tracking, object identification, face recognition, and signal/image recognition.

• Electronics - Prediction of code sequences, designing of integrated circuit (IC) chips, investigation of chip failures, computer vision, synthesis of voice.

• Finance - Real estate appraisal, loan consulting, mortgage assessment, corporate bond appraisal, portfolio trading software, corporate financial analysis, currency valuation prediction, document interpretation, loan application assessment.

• Industrial - This category encompasses various areas such as managing manufacturing processes, analysing product design, implementing quality control systems, evaluating weld quality, predicting paper quality, analysing chemical product design, modelling dynamic chemical process systems, conducting machine maintenance analysis, and managing project bidding, planning, and control.

• Medical - Analysis of cancer cells, EEG and electrocardiogram, creation of prostheses, optimisation of implantation time.

• Speech — Includes tasks such as recognising speech, classifying speech, and converting text into speech.

• Telecommunications encompasses several applications such as image and data compression, automated information services, and real-time speech language translation.

• Transportation - Diagnostics of truck braking systems, vehicle planning, and route planning systems.

• Software - Utilises pattern recognition algorithms, including face recognition and optical character recognition.

• Time Series Forecasting - ANNs are used for predicting stock market trends and occurrences of natural calamities. Neural networks may be taught to perform signal processing tasks, such as filtering audio signals, specifically for hearing aids.

• Vehicle Control – Artificial Neural Networks (ANNs) are often used in the calculation of precise steering commands for physical vehicles.

• Anomaly Detection - Artificial neural networks (ANNs) has the ability to identify patterns effectively, enabling them to be taught to provide outputs when encountering uncommon occurrences that deviate from established patterns.

2.4.8. Artificial Neural Network Training

Artificial Neural Networks, often known as ANNs [7], have similarities to the human brain in terms of its ability to learn via training and store data.

The artificial neural network (ANN) is constructed and trained using the designated input/target data training pattern. Throughout the process of learning, the neural network's output is compared to the desired target value. The network's weights are adjusted using a learning algorithm in order to minimise the error function between the two values.

The Mean Squared Error (MSE) is a frequently used error metric that aims to minimise the average discrepancy between the output of a network and the desired goal value.

A total of 24 signatures, consisting of 12 genuine and 12 forged signatures, were used to train the network, resulting in very accurate validation findings.

Table 1 encompasses comprehensive details pertaining to the creation of neural networks. Both genuine and counterfeit signatures are used to train the neural network. Additionally, test signing is an option that may be used.

Parameters	Value
Layer count	2
Neuronal quantity The output layer	1
Quantity of inputs	12
Constant learning rate	Default
Initial stage of the transfer function	Sigmoid
Secondary Transfer Function Layer	Sigmoid
Initial weights	Randomized
Preconceived notions	Randomized
Epochs: Maximum number of iterations	1000
Target of error	0.0001
Total count of unique patterns for the original signature	12
Total count of counterfeit signature patterns	12
Total number of signatures examined	24
Total count of signatures subjected to analysis	12
Total number of counterfeit signatures examined	12

Table 1 Specifications for Neural Networks

2.5. In conclusion

The second part of the research article focuses on elucidating the techniques used to distinguish genuine and fraudulent signatures via the application of an Artificial Neural Network (ANN) methodology. The chapter delves into the whole process of photo preprocessing, feature extraction, and use of artificial neural networks (ANN) for authentication.

During the first phase, scanned images of signatures undergo many alterations as part of the image preprocessing procedure. The RGB images undergo a conversion to grayscale before

being further translated into binary images. The preprocessing step comprises crucial techniques, such as noise reduction, colour inversion, filtering, and binarization. The objective of these procedures is to enhance the visual clarity of the image, rendering it suitable for further analysis.

The process of feature extraction is of utmost importance in distinguishing between genuine and forged signatures. The paper examines two main methodologies: Histograms of Oriented Gradient (HOG) and Local Binary Patterns. The Histogram of Oriented Gradients (HOG), created by Dalal and Triggs, identifies and depicts the directions of gradients in certain areas of an image. The technique involves dividing the image into small cells, computing gradient histograms for every cell, and creating descriptors based on standardised histograms.

The conversation mostly centres on the Artificial Neural Network (ANN) approach, emphasising its resemblance to the networks found in real brains. The chapter provides a clear explanation of the composition of Artificial Neural Networks (ANN), specifically emphasising its several layers, individual neurons, and the process of knowledge acquisition. This paper explores the methodologies of forward propagation and backpropagation, which are crucial for minimising mistakes and fine-tuning weights to efficiently train a neural network.

Furthermore, Bayesian Networks (BN) are shown as visual structures that illustrate probabilistic relationships between random variables. These networks are known as directed acyclic graphs (DAGs), which allow for the representation of relationships between variables.

The chapter also includes a detailed table that outlines the structure and parameters of the neural network used in the study. The table displays the layers, neurons, transfer functions, initial weights, biases, and other crucial parameters used in both the training and testing phases of the Artificial Neural Network (ANN).

Chapter II outlines a comprehensive methodology that includes image preprocessing, feature extraction using HOG and Local Binary Patterns, and culminates in the building of an Artificial Neural Network for authentication. Utilising Bayesian Networks and a thorough specification table improves the process, establishing a solid basis for distinguishing genuine from counterfeit signatures.

Chapter III: Experimental Procedures and Findings

3.1 Experimental Procedures

The experimental procedure involves applying the established rules to the offline signature template to evaluate the verification accuracy of the proposed offline signature. This is done by following the below stages. Experiments are carried out in accordance with:

1) A training matrix is generated using the unique identifier from the local database. The training matrix comprises the signatures of four individuals, yielding a total of twelve genuine and twelve counterfeit samples for each person. Out of these, five are randomly generated counterfeit samples, while the remaining ones are certified as counterfeit (specifically, they are labelled as "This is a fake sample").

2) The evaluation of the results generated by the Artificial Neural Network (ANN) involves calculating the False Acceptance Rate (FAR) and False Rejection Rate (FRR) for each person independently. The test grid is structured like the training grid.

3) The objective of the Artificial Neural Network (ANN) training is to classify the initial 10 exemplars from the trained grid as positive (+1) and the following 10 marked signature samples as negative (-1). The first 10 samples are authentic, whereas the subsequent 10 samples are counterfeit.

4) The threshold used is zero.

5 The FRR is calculated by assessing the outcomes of the initial 10 samples. If the mark of one or more of initial 10 samples fall under the threshold, they are considered as deemed accepted (mark is higher than the threshold) and a false positive counter is incremented by one. The validation system erroneously rejected it. Alternatively, if the outcomes of the second set of 10 samples above the specified threshold, they are classified as False Accept (FA) and the counter is increased by one (i.e., FA = FA + 1).

Equation (1) represents the calculation of the False Acceptance Rate (FAR), whereas Equation (2) represents the calculation of the False Rejection Rate (FRR).

$$\text{FAR} = \text{number of false positives} \div (\text{number of false positives} + \text{number of true negatives}) \times 100 \quad (1)$$

$$\text{FRR} = \text{number of false negatives} \div (\text{number of false negatives} + \text{number of true positives}) \times 100 \quad (2)$$

Formula (1) calculates the false alarm rate (FAR) by dividing the number of false positives by the sum of the number of false positives and the number of true negatives, and then multiplying the result by 100.

The false rejection rate (FRR) is calculated by dividing the number of false negatives by the sum of the false negatives and true positives, and then multiplying the result by 100.

3.2 Results

Table 1 represents the empirical outcomes of the suggested detection technique, including the False Acceptance Rate (FAR), False Rejection Rate (FRR), and their mean value.

FAR%	FRR%	Accuracy%
3	3.35	96.8

Table 2 Provided algorithm's performance

Several recently published benchmark tests (refer to Table 2) evaluate the performance of the proposed technique by comparing it with several current signature recognition algorithms. Table 2 demonstrates that the False Acceptance Rate (FAR) and False Rejection Rate (FRR) are lower compared to the most current benchmark studies that have been published.

Concerning the False Acceptance Rate (FAR), the research under consideration suggests a value of 3%, which is lower than the values of 5.05% and 4.9% reported in papers [10] and [11], respectively. In relation to the FRR error, the suggested investigation demonstrates a rate of 3.35%, which is lower than the rates indicated in the aforementioned articles, where the FAR rates are 4.25% and 5.2% correspondingly [10], [18].

The primary significance of this work lies in the author's use of Histogram of Oriented Gradients (HOG). The HOG method has shown to be very effective for extracting biometric features in offline scenarios and has consistently achieved impressive identification rates. Table 2 demonstrates that the results surpass the most advanced offline signature recognition techniques.

The mean accuracy is computed for a sample size of 200 individuals, considering every pers on in database.

Existing Methods	FAR (%)	FRR (%)
Normalized Static Features and ANN Classification [17] / 2016	5.05	4.25
Normalized Weighted Coefficients [18] / 2016	4.9	5.2
Suggested algorithm	3	3.35

Table 3 How the provided algorithm has performed

Nevertheless, decreasing his HOG parameters such as cell size, block size, and bins increases the size of the feature vectors that appear in the signature. Therefore, this increase in vector size requires more time for the classifier and slows down the recognition process. Therefore, selecting the HOG parameters in a well-balanced manner (not too small and not too large) will

yield favorable results that are optimized in terms of both identification accuracy and processing speed.

3.3 Program code and comments

3.3.1 Jupyter Notebook Signature Verification

```
import cv2
import os
import glob
import numpy as np
import random
```

Figure 19 Code block pt. 1 (signature verification)

This block of code imports several essential libraries for various purposes:

cv2: This is OpenCV, a popular library for computer vision tasks like image and video manipulation, object detection, etc.

os: This library provides a way to interact with the operating system. It's commonly used for tasks like file handling, directory operations, etc.

glob: This library helps in file/directory handling by finding pathnames matching specified patterns.

numpy as np: NumPy is a powerful library used for numerical computations in Python. It offers support for arrays, matrices, and mathematical functions, making it valuable for scientific computing.

random: This library is used for random number generation. It provides various functions to generate random numbers, shuffle sequences, and so on.

These libraries together can be used for tasks ranging from image processing and manipulation to handling files and directories and performing numerical computations.

```python
def prepare(input):
    # preprocessing the image input
    clean = cv2.fastNlMeansDenoising(input) #denoise the image
    ret, tresh = cv2.threshold(clean, 127, 1, cv2.THRESH_BINARY_INV) #apply threshholding to pixels
    img = crop(tresh) #crop the image

    # 40x10 image as a flatten array
    flatten_img = cv2.resize(img, (40, 10), interpolation=cv2.INTER_AREA).flatten()

    # resize to 400x100
    resized = cv2.resize(img, (400, 100), interpolation=cv2.INTER_AREA)
    columns = np.sum(resized, axis=0) # sum of all columns
    lines = np.sum(resized, axis=1) # sum of all lines

    h, w = img.shape
    aspect = w / h

    return [*flatten_img, *columns, *lines, aspect]


def crop(img):
    points = cv2.findNonZero(img)
    x, y, w, h = cv2.boundingRect(points)
    return img[y: y+h, x: x+w]
```

Figure 20 Code block pt. 2 (signature verification)

This code is a function that prepares an input image for further processing. It performs several preprocessing steps on the input image.

The first step is denoising the image using cv2.fastNlMeansDenoising. This reduces any noise or artifacts present in the image.

The next step applies thresholding to the denoised image using cv2.threshold. Thresholding is used to convert grayscale images to binary images by assigning a specific value (in this case, 0) to pixels below a certain threshold and another value (in this case, 1) to pixels above the threshold.

The cropped function is then called to crop the thresholded image to remove any excess empty space around the character(s) present in the image. This function finds the contours of non-zero points in the image and uses them to determine a bounding box that tightly encloses those points. It returns only the region inside this bounding box as the cropped image.

A flattened version of the cropped image is then created using cv2.resize withdimensions of 40x10. Flattening means converting a matrix into a vector by putting all elements row-wise or column-wise into a single array.

The original cropped image is then resized again, but this time to dimensions of 400x100 using cv2.resize.

Next, two arrays are created: "columns" and "lines". Columns represents a sum of white pixels along each column of the resized image, which gives information about how dark or light each column is compared to others. Lines represents a sum of white pixels along each row of the resized image, which gives similar information about each row.

Lastly, aspect ratio calculation is performed by dividing width by height.

The code first preprocesses the input image by denoising it with the help of the cv2.fastNlMeansDenoising() function. This function is then used to threshold the cleaned image, which results in a flattened image of 40x10 pixels.

Next, the img variable is cropped to the given threshhold using the cv2.threshold() function. The x and y coordinates of each pixel in the cropped image are then saved as points. Finally, the returned img variable is constructed by concatenating these points together along with their corresponding width and height values.

```
class NeuralNetwork():

    def __init__(self, sizes):
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y,1) for y in sizes[1:]]
        self.weights = [np.random.randn(y,x) for x,y in zip(sizes[:-1],sizes[1:])]

    def feedforward(self, a):
        for b,w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a) + b)
        return a

    def separate_batches(self, training_data, batch_size):
        random.shuffle(training_data)
        n = len(training_data)
        return [training_data[i:i + batch_size] for i in range(0, n, batch_size)]

    def update_batches(self, batches, alpha):
        for batch in batches:
            nabla_b = [np.zeros(b.shape) for b in self.biases]
            nabla_w = [np.zeros(w.shape) for w in self.weights]
            m = len(batch)
            for x, y in batch:
                delta_b, delta_w = self.backpropagation(x, y)
                nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_b)]
                nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_w)]

            self.weights = [w - (alpha / m) * nw for w, nw in zip(self.weights, nabla_w)]
            self.biases = [b - (alpha / m) * nb for b, nb in zip(self.biases, nabla_b)]

    def backpropagation(self, x, y):
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
```

Figure 21 Code block pt. 3 (signature verification)

The code provides the beginning of a class definition for a neural network.

The `__init__` method initializes the neural network object. It takes in a parameter called `sizes`, which is a list containing the number of neurons in each layer of the network. The first element of `sizes` represents the input layer size, and subsequent elements represent hidden and output layer sizes.

`self.num_layers = len(sizes)` calculates and assigns the number of layers in the neural network based on the length of the `sizes` list.

`self.sizes = sizes` assigns the input `sizes` to an instance variable called `self.sizes`. `self.biases` is initialized as a list of randomly generated bias values using `np.random.randn(y,

1)`. Each item in this list represents biases for individual neurons at each layer (except input layer). The shape `(y, 1)` indicates that we have y bias values for y neurons at that particular layer.

Similarly, `self.weights` is initialized as a list of randomly generated weight matrices using `np.random.randn(y, x)`. Each matrix represents weights connecting two consecutive layers. The dimensions `(y, x)` mean that there are y rows (neurons) in current layer connected to x columns (neurons) in the previous layer.

The code continues with another method definition called `feedforward(a)`. This method takes in an argument called `a`, which represents an input to be fed into the neural network.

Inside this method, we have a loop that iterates over all pairs of biases and weights using zip(). This loop corresponds to propag. The code defines a neural network with three layers and four inputs. The first two layers are initialized with random values drawn from the y-axis, while the third layer is initialized with a value drawn from the x-axis. Finally, the weights for each layer are initialized with random values drawn from the y-axis and x-axis, respectively.

Next, the feedforward function is called. This function takes in an input value on the left side of the equation and outputs a corresponding value on the right side of the equation. In this case, it takes in a single input value (a) and outputs a single output value (b).

The `sigmoid` function takes in an input array `a` and applies the sigmoid activation function to it. The dot product of the weight matrix `w` with input vector `a` is computed, and then vector `b` is added. Finally, the result is passed through the sigmoid function.

The method `separate_batches` takes in a list of training data and a batch size as parameters. It shuffles the training data randomly and divides it into separate batches with each batch containing `batch_size` number of data points. This method is used for dividing large datasets into smaller batches during gradient descent optimization process.

The method `update_batches` takes in two parameters: a list of batches and a learning rate (`alpha`). It iterates over each batch and performs some kind of update operation for the neural network parameters (e.g., weight adjustment using gradient descent).

The code will create a list of lists, where each list will contain the data from a single batch. The code then calculates the sigmoid function and adds the input data (w) to it as well as the training data (a). Finally, it returns this value. Next, the code creates a list of n length and shuffles the training data before creating an array for storing all of the batches. Finally, it calls the update_batches() function with each batch as its argument.

```
        activation = x
        activations = [x]
        zs = []
        for b, w in zip(self.biases, self.weights):
            # layer-bound b and w
            z = np.dot(w, activation)+b
            zs.append(z)
            activation = sigmoid(z)
            activations.append(activation)
        # backward pass
        delta = self.cost_derivative(activations[-1], y) * \
            sigmoid_prime(zs[-1])
        nabla_b[-1] = delta
        nabla_w[-1] = np.dot(delta, activations[-2].transpose())

        for l in range(2, self.num_layers):
            z = zs[-l]
            sp = sigmoid_prime(z)
            delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
            nabla_b[-l] = delta
            nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
        return (nabla_b, nabla_w)

    def sgd(self, training_data, epochs, batch_size, alpha, test_data):
        n_test = len(test_data)
        error=0

        for epoch in range(epochs):
            batches = self.separate_batches(training_data, batch_size)
            self.update_batches(batches, alpha)
            genuine=self.evaluate(test_data)
            error = error+abs(genuine-12)
            print("Epoch {0}: {1} / {2}".format(epoch, genuine, n_test))
        error=error/epochs
        print("Result deviation: {0}%".format(error*100/12))
```

Figure 22 Code block pt. 4 (signature verification)

```
    def evaluate(self, test_data):
      for (x,y) in test_data:
        test_results = [(np.argmax(self.feedforward(x)), y)
                        for (x, y) in test_data]
        return sum(int(x == y) for (x, y) in test_results)

    def cost_derivative(self, output_activations, y):
        return output_activations - y


def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))


def sigmoid_prime(z):
    return sigmoid(z) * (1-sigmoid(z))
```

Figure 23 Code block pt. 5 (signature verification)

This part of code is implementing the gradient descent algorithm for training a neural network. The `nabla_b` and `nabla_w` variables are initialized as lists of zeros with the same shape as the biases and weights matrices respectively. These will be used to store the gradients of the cost function with respect to each bias and weight in the network. The loop over the batch updates these gradient variables for each sample in the batch using the backpropagation method.

This method computes the partial derivatives of the cost function with respect to each bias and weight in the network using the chain rule. The updated gradients are then added to `nabla_b` and `nabla_w` by element-wise addition. After processing all samples in the batch, these accumulated gradients are averaged by dividing them by `m`, which is the number of samples in the batch. Finally, these averaged gradients are used to update the weights and biases of the network using gradient descent.

The new weight and bias values are computed by subtracting from their current values an amount proportional to their respective gradients (`(alpha / m) * nabla_w`) multiplied by a learning rate parameter called `alpha`. Overall, this code performs a single epoch (pass through) of mini-batch stochastic gradient descent training for a neural network.

The code will initialize the variables nabla_b and nabla_w to zeros. Then, it will calculate the gradient of the loss function with respect to x and y using the backpropagation algorithm. Next, it will update nabla_b and nabla_w by adding the derivative of x with respect to y to each variable. Finally, it sets self.weights and self.biases to their respective updated values.

The code is a method in a neural network class, as it references the weights and biases attributes using "self". It is to be implementing the forward propagation and backward propagation steps of training the neural network. The variable "x" seems to represent the input to the current

layer, and it's being used as the initial activation. The "activations" list is initialized with just "x". The loop iterates over each layer's weights and biases.

For each layer: It calculates the weighted sum of the previous layer's activation by taking the dot product of the weights matrix and the activation vector, then adding biases.

The resulting sum (referred to as "z") is appended to the "zs" list. It applies an activation function called "sigmoid" on this sum ("z"), producing a new activation value for that layer, which is then added to the "activations" list. After looping through all layers, it proceeds with backward propagation: Calculates the error in prediction for this output layer by applying the derivative of sigmoid function (called "sigmoid_prime") on the sum of that last layer ("zs[-1]"). Uses this error to update nabla_b (likely an array or matrix storing derivatives/biases at each neuron) for that output layer.

The code first creates an array of zeros, called zs.This will be used to store the outputs of the activation function after it has been multiplied by each bias and weight in turn.Next, the code calculates the activation function using the dot product between each bias and weight and the input activation value.

The sigmoid function is then used to calculate the output activation value.

The next step is to calculate the cost derivative for each activation value against the target y value. This is done by first calculating the delta value for each activation against its previous iteration's target y value, then multiplying this result by a sigmoid_prime function. The Nabla operator (^) is then applied to these values to produce a single.

```
def main():
    print('OpenCV version {} '.format(cv2.__version__))
    training_data = []
    for filename in glob.iglob('D:/project/Signature-Verification/training/024/*.png', recursive=True):
        img = cv2.imread(filename, 0)
        if img is not None:
            data = np.array(prepare(img))
            data = np.reshape(data, (901, 1))
            result = [[0], [1]] if "genuine" in filename else [[1], [0]]
            result = np.array(result)
            result = np.reshape(result, (2,1))
            training_data.append((data, result))
    test_data = []
    for filename in glob.iglob('D:/project/Signature-Verification/testing/024/*.png', recursive=True):
        img = cv2.imread(filename, 0)
        if img is not None:
            data = np.array(prepare(img))
            data = np.reshape(data, (901, 1))
            result = 1 if "genuine" in filename else 0
            test_data.append((data, result))

    net = NeuralNetwork([901, 500, 500, 2])
    net.sgd(training_data, 10, 50, 0.01, test_data)
```

Figure 24 Code block pt. 6 (signature verification)

This code appears is using the OpenCV library for image processing. It is provided to be training a neural network model for signature verification.

Here are the main steps of the code.

Import necessary libraries and print the version of OpenCV being used.

Create empty lists, `training_data` and `test_data`, to store the training and testing data respectively.

Iterate through all the image files in a particular directory using `glob` module.

Read each image file using `cv2.imread()`. If the image exists (i.e., not None), perform some processing on it using a custom function called `prepare()`.

Convert the processed image data into a numpy array, reshape it into a column vector of size 901 x 1, and store it as one part of the input data.

Determine if the current image belongs to a genuine or forged signature by checking if the filename contains 'genuine' keyword or not.

Create another numpy array as per result labels accordingly: [0, 1] for genuine signature images, [1, 0] otherwise. Reshape it into a column vector of size 2 x 1 and store it as one part of training data.

Append both input data and result labels arrays as a tuple to `training_data` list.

Repeat steps for all image files in another directory but just store binary labels (0 or 1) instead of numpy arrays for their results.

Initialize a neural network model with an input layer size of 901 neurons, two hidden layers each consisting of 500 neurons, and an output layer size of 2 neurons (corresponding to genuine/forged labels

The code first imports the necessary libraries and creates a Neural Network object. The Neural Network expects three input parameters (901, 500, 500) and outputs two values (1 or 0). Next, the code sets up a SGD training algorithm on the training data and compares the results against the test data.

3.3.2 Pre-processing

```python
from PIL import Image
from IPython.display import display
import matplotlib.pyplot as plt
import cv2
import numpy as np
#from google.colab.patches import cv2_imshow
#import os
#import glob
#%matplotlib inline

# load image
im = Image.open("sign.jpg")
display(im)

# Resize image
width, height = im.size
# print(width,height)
aspect_ratio = width/height
# print(aspect_ratio)
# let height is 60
im_height = 60
im_width = int(im_height * aspect_ratio)
resized_im = im.resize((im_width,im_height), Image.LANCZOS)
display(resized_im)
```

Figure 25 Code block pt. 1 (preprocessing)

```python
# change image mode using pillow
# print(resized_im.mode)
img = resized_im.convert('LA')
plt.hist(img.histogram())
plt.show()

# change image mode using cv2
im_cv = cv2.imread('sign.jpg')
gray_im_cv = cv2.cvtColor(im_cv, cv2.COLOR_BGR2GRAY)
rsz_im_cv = cv2.resize(gray_im_cv, None, fx=0.05, fy=0.05)
plt.hist(rsz_im_cv)
plt.show()

#******** Code Using OpenCv**********

# Read
im3 = cv2.imread('sign.jpg')
#cv2.imshow(im3,0)
cv2.imshow('Test image',im3)
```

Figure 26 Code block pt.2 (preprocessing)

This code imports several libraries such as PIL, Image, IPython.display, matplotlib.pyplot, and cv2. It also imports the PIL library function that allows us to resize the image.

The code starts by opening an image using the Image.open() function from the PIL library. The opened image is then displayed using the display () function from IPython.display. Next, the code calculates the new width and height of the image based on a desired height of 60 pixels.

This aspect ratio is calculated by dividing the width by height. The image is then resized using the resize () method of the opened image object and passing in the new width and height values. The resized image is stored in a variable called resized_im.

To change the mode of the image from color to grayscale, two different approaches are used. First, using PIL's convert () method, we pass 'LA' as an argument to convert color mode to grayscale with alpha channel (transparency). The converted image is stored in a variable called img. A histogram of pixel intensities in the grayscale version of the images is then displayed using matplotlib.pyplot's hist() function.

Secondly, using cv2.imread(), we load and read color sign.jpg.Image matrix data objects needs to be encoded first; Each pixel has all 3BGRCHANNELvalues like(openCV:imageMatrix => [i,j]=v1_v2_v3-v4... Approach.The result -(a millions or more data generated RGBArowseries informationpictureplot.Althoughit can rise quickly for another column dataset containing cloring layers'operationlists use expectedno netgobeam means same spotfirstreflect user itspotquality easier when necessaryanother reasons opencvone format.)

Similarly, conversion: In the next linessnippe the code reads in the image 'sign.jpg' and displays it on the screen.

Next, we create two new images - one that will be used for histograms and another that will be used to resize an image.

The histogram image is created by calling cv2.hist(image) which takes in an input image and returns a Histogram object. This object contains all the pixels' RGB values as well as their frequency of occurrence. We then show this histogram using plt.hist().

The resize image is created by calling cv2.resize(input_image, width, height) which takes in an input image (in this case 'sign.jpg'), a width and height.

Chapter IV: Summary

Neural networks have shown efficacy in several applications due to their ability to efficiently tackle certain issues without the need for a predefined model. An important characteristic of Artificial Neural Networks (ANN) is its capacity to offline learn nonlinear problems by means of selective training, resulting in highly accurate replies.

The utilisation of artificial neural networks (ANN) in the aforementioned issues is gaining significance, mostly because to the escalating efficacy of contemporary computers. In addition, the time required for modelling and testing in applications using artificial neural networks (ANN) is short. Furthermore, verification systems based on ANN have the ability to learn various kinds of signature data.

Moreover, it is unnecessary to use extensive datasets to showcase the capacity for learning in this particular scenario. We specifically chose a total of 12 authentic signatures and 12 counterfeit signatures for the purpose of training, which yielded quite favourable outcomes. Nevertheless, in real-world application, a greater amount of training data enhances the resilience of the system.

The maximum classification accuracy was attained after the completion of the training. The accuracy rate exceeds 93%. Our supported algorithms use basic geometric attributes to analyse signatures and efficiently differentiate between authentic and counterfeit ones. The method is very resilient and capable of identifying arbitrary, uncomplicated, and poorly executed counterfeit items. Our ability to replicate signatures is not advanced enough to be classified as a talented forger, thus we cannot accurately predict how our system would do against a highly experienced forger.

References

[1] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection” in 2005. IEEE Computer Society Conference on Computer Vision and Pattern Recognition

(CVPR'05), 2005.

[2] T. Ojala, M.Pietikainen and D. Harwood. “Performance evaluation of texture measures with classification based on Kullback discrimination of distributions.”,1994.

[3] J. Hu and Y. Chen, “Offline signature verification using real Adaboost classifier combination of pseudo-dynamic features,” in Proceedings of the International Conference on Document Analysis and Recognition, 2013

[4] Shu, H., “On-line handwriting recognition using hidden Markov models.” Thesis Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1997

[5] S. N. Gunjal, B. J. Dange, and A. V. Brahmane, “Offline Signature Verification using Feature Point Extraction,” International Journal of Computer Applications, 2016

[6] J. Bappy, T. Mohammed, et al., “Detection and Localization of Image Forgeries Using Resampling Features and Deep Learning,” in IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, July 2017.

[7] W. S. Alaloul and A. H. Qureshi, “Data Processing Using Artificial Neural Networks,” in Dynamic Data Assimilation - Beating the Uncertainties, 2019.

[8] M. Boughida and T. Boubekeur, “Bayesian Collaborative Denoising for Monte Carlo Rendering,” Computer Graphics Forum, 2017.

[9] “Histograms of Oriented Gradients for Human Detection”, http://lear.inrialpes.fr/people/triggs/pubs/Dalalcvpr05.pdf

[10] W. Xing, Y. Zhao, R. Cheng, J. Xu, S. Lv, and X. Wang, “Fast Pedestrian Detection Based on Haar Pre-Detection,” International Journal of Computer and Communication Engineering, 2012

[11] M. Singhal, M. Trikha, and M. Dutta, “Time Independent Signature Verification using Normalized Weighted Coefficients,” International Journal of Electrical and Computer Engineering, 2016.

[12] https://www.mendeley.com/catalogue/6d2a7951-a4d7-31bf-8d38-2eeb582e8e8b/ Zhou, Y., Zheng, J., Hu, H., & Wang, Y. (2021). Handwritten signature verification method based on improved combined features. Applied Sciences (Switzerland), 11(13). https://doi.org/10.3390/app11135867