TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Signature plays an important role in banking, financial, commercial etc. Signature may be unique for each person. However, with signatures comes many challenges since any two signatures may look very similar with little to no differences written by the same person. As there are unique and important variations in the feature elements of each signature, thus in order to match a particular signature with the database, the structural parameters of the signatures along with the local variations in the signature characteristics are used. In order to avoid any such identity crimes committed in banks and many other companies, forgery detection system is a solution to this problem along with the help of the concepts of Deep learning and Neural Networks.

The Signature Forgery Detection project aims to develop a system for authenticating handwritten signatures by leveraging image processing techniques and machine learning algorithms. The project involves the extraction of distinctive features from genuine and forged signature images, followed by the training of a neural network to classify between authentic and forged signatures. The system utilizes image preprocessing methods such as grayscale conversion, binary thresholding, and region-based analysis to extract relevant features. These features include aspects like ratio, centroid, eccentricity, solidity, skewness, and kurtosis.

The developed system demonstrates promising results in terms of accuracy and efficiency, contributing to the field of signature forgery detection. Future enhancements may involve exploring additional image processing techniques, optimizing the neural network architecture, and expanding the dataset to further improve the model's performance.

1. INTRODUCTION

1.1 About the project

Handwritten signatures have been a prevalent method of authentication for various legal and financial transactions. However, with the increasing sophistication of forgery techniques, the need for robust signature forgery detection systems has become paramount. The Signature Forgery Detection project addresses this challenge by combining image processing and machine learning methodologies to create a reliable system for distinguishing between genuine and forged signatures.

The primary objective of this project is to develop an automated and accurate signature verification system that can contribute to the security of important documents and transactions. Signature forgery poses a significant threat to the integrity of legal and financial processes, making the development of an effective detection system crucial in safeguarding against fraudulent activities.

The project adopts a multi-step approach, starting with image preprocessing techniques to extract relevant features from signature images. These features, including ratio, centroid, eccentricity, solidity, skewness, and kurtosis, provide a comprehensive representation of the signature's unique characteristics. Leveraging a neural network, the system is trained on a dataset containing genuine and forged signatures to learn the underlying patterns that distinguish between the two categories.

The data used in this project encompasses signatures from multiple individuals, ensuring a diverse and representative training set. The neural network architecture incorporates multiple hidden layers, allowing the model to capture intricate patterns and relationships within the feature space.

As a result, the developed system offers a holistic solution for signature forgery detection, encompassing feature extraction, model training, and evaluation procedures. The project's significance lies in its potential to enhance the security of transactions, legal documents, and other sensitive materials that rely on handwritten signatures for authentication.

This report provides a detailed overview of the Signature Forgery Detection project, outlining the methodologies employed, the dataset used, the architecture of the neural network, and the

evaluation metrics used to assess the system's performance. The findings and implications of the project contribute to the broader field of document security and provide a foundation for further advancements in signature verification technology.

1.2 Existing System

Signature forgery detection has been a subject of significant research, and various systems have been proposed to address the challenges associated with authenticating handwritten signatures. These systems can broadly be categorized into two types: traditional, rule-based methods, and modern, machine learning-based approaches.

Traditional Rule-Based Systems

Feature-based Approaches: Many traditional systems rely on extracting handcrafted features from signatures, such as stroke characteristics, slant, curvature, and spacing. These features are then used in rule-based algorithms or expert systems for signature verification.

Dynamic Signature Analysis: Some systems incorporate dynamic information, analysing the temporal aspects of the signature, such as the speed and pressure applied during the signing process. Dynamic signature analysis aims to capture the inherent variability in an individual's signing behaviour.

Commercial Solutions

Several commercial signature verification systems exist, offering a range of features and capabilities. These systems often incorporate a combination of traditional and modern techniques for enhanced accuracy.Commercial solutions may also integrate additional security measures, such as biometric authentication and multi-factor verification, to strengthen overall security.

While these existing systems have made significant strides in signature forgery detection, challenges persist, especially in handling variations in signature styles, dealing with diverse datasets, and adapting to evolving forgery techniques. The Signature Forgery Detection project aims to contribute to this field by exploring a machine learning-based approach, specifically utilizing neural networks for improved accuracy and adaptability.

1.3 Proposed System

The proposed Signature Forgery Detection system combines image processing and machine learning techniques to create an advanced and accurate solution for distinguishing between genuine and forged handwritten signatures. The system is designed to address the limitations of existing approaches by leveraging a neural network architecture and incorporating a comprehensive feature extraction pipeline.

Feature Extraction:

The system begins with a robust feature extraction process, encompassing grayscale conversion, binary thresholding, and region-based analysis. The extracted features include ratio, centroid coordinates, eccentricity, solidity, skewness, and kurtosis. These features collectively capture the distinctive characteristics of handwritten signatures.

Neural Network Architecture:

The core of the proposed system is a multilayer perceptron neural network. The architecture consists of multiple hidden layers, each containing a specific number of neurons. The choice of neural network architecture allows the model to learn intricate patterns and relationships within the feature space, providing a more nuanced understanding of genuine and forged signatures.

Training Process:

The system is trained on a dataset comprising genuine and forged signatures from multiple individuals. During the training process, the neural network optimizes its parameters to minimize the mean squared difference between predicted and actual labels. The use of a squared difference loss function enhances the sensitivity of the model to subtle variations in signature features.

Evaluation Metrics:

To assess the system's performance, the proposed solution employs accuracy metrics both during training and testing phases. The evaluation process includes measuring the accuracy of the model on both the training and testing datasets. Additionally, the system provides real-time testing capabilities for individual signature images, offering practical utility in signature verification scenarios.

Real-Time Testing:

The proposed system allows for the real-time testing of individual signature images. This feature enhances the system's practical applicability, enabling users to verify signatures on-demand and contribute to the security of transactions and document authentication.

In summary, the proposed Signature Forgery Detection system represents an advanced and comprehensive approach to signature verification. By combining sophisticated feature extraction methods with a neural network framework, the system aims to provide a robust solution for addressing the challenges associated with handwritten signature forgery. The project contributes to the evolving landscape of document security and authentication technologies.

1.4 Requirements Specification

1.4.1 Software Requirements

Language	: Python 3.6
Operating system	: Windows or Linux
IDE	: Jupyter Notebook
Tools	: TensorFlow, Keras, skimage, NumPy, Pandas,Matplotlib.

1.4.2 Hardware Requirements

Processor	: Intel Core-i5
RAM	: 4 GB
ROM	: 256 GB

2. LITERATURE SURVEY

Signature forgery detection has been a critical area of research due to its implications in document security and fraud prevention. This literature survey explores existing works that have contributed to the understanding and development of signature forgery detection systems.

a) Handwritten Signature Verification using Deep Learning

Alajrami et al. (2019) proposed a signature verification and forgery detection system using deep learning. The paper focuses on offline and online signature verification, employing template matching and Hidden Markov model. For forgery detection, a Convolutional Neural Network (CNN) is utilized, based on behavioural changes in signatures rather than psychological traits. The methodology involves collecting signatures and unique characteristics to create a knowledge base, followed by preprocessing and CNN application using Keras with TensorFlow. The dataset comprises 300 images (150 real, 150 forged) from 30 individuals, resulting in a split ratio of 8:2 with the highest accuracy.

b) Offline Signature Recognition and Forgery Detection using Deep Learning

Poddar et al. (2020) introduced a signature recognition, forgery detection, and verification system employing Convolutional Neural Network (CNN), Crest-Through Method, SURF algorithm, and Harris corner detection algorithm. The CNN and Crest-Through Method handle recognition and verification, while SURF and Harris corner detection address forgery detection. Pre-processing involves noise removal, scaling, centralization, and rotation in CNN, and length to space ratio, width to space ratio, and Crest-Through parameter in Crest-Through Method. Both CNN and Crest-Through Method contribute to signature recognition. For forgery detection, Harris corner detection extracts corner points, and SURF algorithm extracts index points for comparison with real signatures. The system demonstrates 94% accuracy for signature recognition and 85-89% accuracy for forgery detection, indicating its utility despite potential flaws.

c) OFFLINE SIGNATURE FORGERY DETECTION USING CONVOLUTIONAL NEURAL NETWORK

Raj Balsekar et al. (2020) proposed a CNN-based system for signature recognition and verification, conducting literature surveys on various aspects. The authors explored static and dynamic signatures, multiple explanations for offline verification, behavioural changes in individual signatures, and advancements in image processing and machine learning. The system, utilizing a dataset of 350 signatures from 15 individuals, extracts unique features and transforms signatures through a grayscale algorithm and geometric transformation. The pre-processed signature features are then compared with stored signatures in the system, determining the authenticity of each signature as real or fake.

d) Online Signature Verification System

This system aims to differentiate between authentic and fake signatures, employing a Support Vector Machine (SVM). Signatures are represented as x-y coordinate boundaries and stored in a text file within the signature database. The method involves reading input signatures, acquiring real-time signatures through special pens, preprocessing with normalization, sampling time, and resampling distance, and enhancing feature extraction. Features are added to the database, and image features are matched for validation. The program correctness is demonstrated through two methods, ensuring the accuracy of signature images and their characteristics.

Table 2.1 : Comparison of Literature Survey

S.NO	TITLE	AUTHORS	INNOVATION	RESULTS
1	Handwritten Signatures Verification using Deep Learning	Alajrami et al. (2019)	Uses CNN as a feature extractor and classifier, introduces ResNet to address gradient issues.	Successfully achieves offline signature verification with increased efficiency and accuracy, detecting sophisticated forgeries.
2	Offline Signature Recognition and Forgery Detection using Deep Learning	Poddar, Jivesh, et al. (2020)	Utilizes CNN, Crest-Through Method, SURF, and Harris corner detection for recognition and forgery detection.	Achieves 94% accuracy for signature recognition and 85-89% for forgery detection.
3	Offline Signature Forgery Detection Using CNN	Raj Balsekar, et al. (2020)	Proposes a system with a dataset of 750 signatures, using matrix representation and geometric transformations.	Successfully distinguishes between real and fake signatures.
4	Online Signature Verification System		Implements SVM for distinguishing authentic and fake signatures, utilizes x-y coordinates and preprocessing steps.	Demonstrates correctness with two methods, emphasizing the use of SVM in pattern recognition and regression problems.

3. Signature Forgery Detection

The process of signature verification and recognition allows the user to detect whether a signature is original or forged. According to many studies that were done on signatures and types of signatures, there are 3 major categories of forged signatures:

1. Random: these signatures are not based on any knowledge of the original Signature.
2. Simple: these signatures are based on an assumption of how the signature knowing the name of the signer.
3. Skilled: an imitation of the original signature, which means that the person knows exactly how the original signature looks like.

It can be concluded that skilled signatures are the most difficult to detect, these can be very similar to the original signature, and the error rate might be very small. There are 3 major steps in achieving signature verification and recognition, and each of these steps consists of many methods that contribute to improved results. These steps are:

- Image pre-processing
- Feature extraction
- Neural network training

3.1 Image Pre-Processing of data

Image pre-processing represents a wide range of techniques that exist for the manipulation and modification of images. It is the first step in signature verification and recognition. A successful implementation of this step produces improved results and higher accuracy rates.

After an image is acquired, it goes through different levels of processing before it is ready for the next step of feature extraction. The following are the reasons why image preprocessing is important:

1. It creates a level of similarity in the general features of an image, like the size aspect. This enhances the comparison between images.
2. Signatures vary according to the tool that was used in writing; the type of pen/pencil, the ink, the pressure of the hand of the person making the signature, and so on. In off-line signature

recognition, these facts are not important, and have to be eliminated and the matching should be based on more important offline features.

3. Noise reduction, defects removal and image enhancement.

4. Improves the quality of image information.

5. It eases the process of feature extraction, on which the matching depends mainly.

The data comprised of five different signatures of real and forged images for each single person.Format of path signature image in the model is XXXZZZ_YYY.png XXX denotes id of the person who has signed on the document (ex - 001) ZZZ denotes the id of the person to whom the sign belongs in actual (ex- 001) YYY denotes the nth number of attempts.

Image pre-processing differs according to the genre that the image belongs to. The techniques used in this process may vary. There are 4 techniques that are used for signature recognition including; reading, displaying and resizing of the image, segmentation, binarization, fast fourier transform (FFT), enhancement and thinning

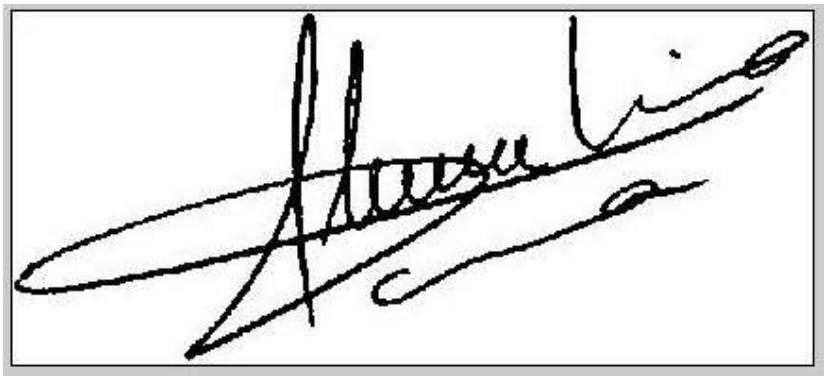

Figure 3.1 : Original Scanned Signature

Figure 3.2 : Segmented and binarized signature

Figure 3.3: Thinned Signature

3.2 Feature Extraction

Feature extraction is the second major step in signature recognition and verification. If we are to compare 2 sketches; there should be at least one measurement on which to base this comparison. The main function of this step is to generate features which can be used as comparison measurements. Since the issue of signature verification is a highly sensitive process, more than one feature/measurement has to be generated in order to enhance the accuracy of the results.

The term feature here refers to a certain characteristic that can be measured using designed algorithms; which can then be retrieved by "extraction".

For this signature recognition and verification research, four main features will be extracted. These features are: eccentricity, skewness, kurtosis, orientation.

3.2.1 Eccentricity

Eccentricity is defined as the central point in an object. In case of signature image, eccentricity is the central point of the signature. The importance of this feature is that we need to know the central point of 2 images in order to compare them. After identifying the central point, we can then compare the features around them. If there is a deviation in the central point of an image, this will indicate a possible imitation of the signature, but this is not enough evidence by itself. The central point is acquired by applying the ratio of the major to the minor axes of an image.

3.2.2 Skewness

"Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution or data set, is symmetric if it looks the same to the left and right of the center point". [11]

The skewness can be defined according to univariate data Y1,Y2,…,YN, as following

$$skewness = \frac{\sum_{i=1}^{N}\left(Y_i - \overline{Y}\right)s^3}{(N-1)s^3} \tag{1}$$

Where $\overline{Y}$ is the mean, S is the standard deviation, and N is the number of data points. The measurement of skewness allows us to determine how bowed are the lines in each segment of the signature. The percentage of this torsion is then calculated and extracted. Furthermore, this percentage is compared to that extracted from the other image.

The importance of this feature is that it measures the symmetry or the lack of it, which is an important aspect of a signature. Most signatures are complicated, with no edges but twists, and the width and height of these twists is a very important aspect for measurement and comparison.

3.2.3 Kurtosis

Kurtosis is a measure of whether the data are peaked or flattened, relative to a normal distribution. That is, data sets with high kurtosis tend to have a distinct peak near the mean, decline rather rapidly, and have heavy tails. Data sets with low kurtosis tend to have a flat top near the mean rather than a sharp peak. A uniform distribution would be the extreme case.

The Kurtosis can be defined according to univariate data Y1, Y2,…,YN, as following

$$Kurtosis = \frac{\sum_{i=1}^{N}\left(Y_i - \overline{Y}\right)s^4}{(N-1)s^4} \tag{2}$$

Where $\bar{Y}$ is the mean, S is the standard deviation, and N is the number of data points. There are other definitions for excess kurtosis, but here in this paper the original definition is used.

The kurtosis measurement highlights the peaks in each segment of a signature. It also measures the existence, or the absence of tails, that are unconnected lines with no peaks. As you can see, kurtosis and skewness are highly interlinked.

3.2.4 Orientation

Orientation defines the direction of the signature lines. This feature is important because it allows us to know how the signer wrote down the signature, which letters came first emphasizing the direction of angles and peaks. The orientation feature is used to compute the optimal dominant ridge direction in each block of a signature.

Orientation is acquired by applying the ratio of angle of major axis. The orientation of the signature can be found using the Matlab "regionprops" function, in which the angle between the x-axis and the major axis of the ellipse that has the same second-moments as the region.

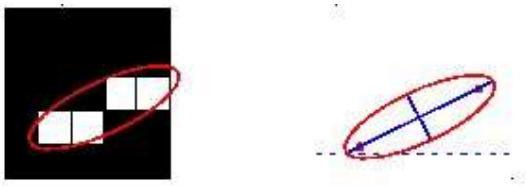

Figure 3.4: The orientation detection

Figure 3.4 illustrates the axes and orientation of the ellipse. The left side of the Figure 4 shows an image region and its corresponding ellipse. The right side shows the same ellipse, with features indicated graphically; the solid blue lines are the axes, the red dots are the foci, and the orientation is the angle between the horizontal dotted line and the major axis.

3.3 Pattern Recognition through Neural Network

Neural networks are known for being a very accurate and efficient technique for pattern recognition in general. In this section, we will explain more about the idea and structure of neural networks, their types, and how they contribute to pattern recognition.

A neural network is one application of artificial intelligence, where a computer application is trained to think like a human being or even better. A neural network is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain,

process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems.

Neural networks - like human beings - depend on the idea of learning in order to achieve any task. They learn through training on a large number of data, which enables them to create a pattern with time, that they will use later. They are very helpful in detecting patterns that are complicated and hard to derive by humans or by simple techniques. Just like the case of signature recognition, it is very hard to tell whether a signature is original or forged, especially if it is carried out by a skilled forger. Thus, a more advanced technique to detect the differences is needed to achieve a decision on its authenticity. Neural networks do not follow a set of instructions, provided for them by the author, but they learn as they go case by case.

Neural networks are highly reliable when trained using a large amount of data. They are used in applications where security is highly valued. In this research we used Multi-Layer Perceptrons MLPs neural network. The structure of this neural network depends on the multi-layer feed forward, where all the nodes in any layer have connections to all the nodes in the next layer and so on, but these nodes do not have any connections with the previous layers. Then, it was modified to function as a back-propagation neural network, using the BP algorithm.

The Node is a processing element which produces an output based on a function of its inputs. In Figure 3.5 the neuron is represented with the node, where the neuron has N weighed inputs and a single output. The neuron calculates a weighted sum of inputs and compares it to a threshold. If the sum is higher than the threshold, the output is set to 1, otherwise to -1.

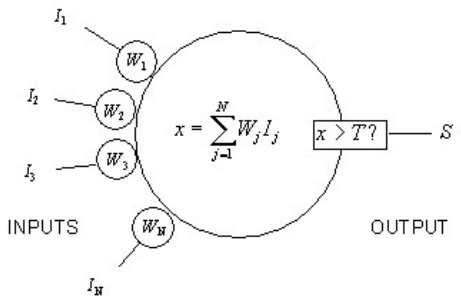

Figure 3.5 : The architecture of artificial neuron

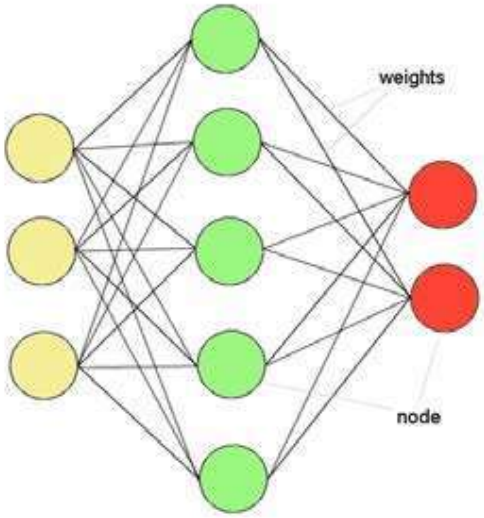

Figure 3.6 : The Architecture of MLP

Figure 3.6 shows a three-layer perceptron with an input layer, one hidden layer and an output layer. Classification and recognition capabilities of MLP come from the nonlinearities used within the nodes. *I1,I2,...,IN* these are the input signals, *w1,...,wN* are the synaptic weights, *x* the activation potential, θ the threshold and *y* the output signal and *f* the activation function

$$x = \sum_{i=1}^{N} w_i I_i \qquad (3)$$

$$y = f(x - \theta) \qquad (4)$$

Defining w0=θ and I0=-1, the output of the system can be reformulated as:

$$y = f\left(\sum_{i=0}^{N} w_i I_i\right) \qquad (5)$$

The activation function *f* defines the output of the neuron in terms of the activity level at its input. The most common form of activation function used is the sigmoid function.

The implementation of BP learning, updates the network weights and biases in the direction in which the performance function decreases most rapidly, the negative of the gradient. The equation for this algorithm, for one iteration, can be written as follows:

$$X_{k+1} = X_k - \alpha_k g_k \qquad (6)$$

Where *Xk* is a vector of current weights and biases, *gk* is the current gradient and *αk* is the learning rate. In this paper, the gradient is computed and the weights are updated, after each input to the network.

The network was then trained using the set of data available in the database as an input-to-input layer, which include the images and their extracted features of eccentricity, skewness, kurtosis, and orientation. The output layer consists of a single node that calculates the weighted sum of the connections coming to the output layer. The number of the neurons in the hidden layer is double of the neurons in the input layers, so that the neural network can learn the technique of recognition based on these previously mentioned features.

The final output from MLPs networks is a confidence value indicating the likelihood that the test signature was performed by the same person that provided the reference signatures used in training. The confidence value is compared to a threshold and the test signature is verified.

If the confidence exceeds this threshold, it is accepted or otherwise is rejected. Then the classification rate or the error can be calculated in percentage rate.

3.4 Recognition

As a result of all previous processes, recognition of a signature is achieved. The following are the steps detailing how exactly the recognition process is designed and operates:

1. The trained neural network – which has learned how to work on signatures and their features through training – compares the features of the given signature with those of the signatures in the database.
2. The differences between the extracted features from the new signature and those in the database is calculated. The outcome of the total of these differences is calculated.

3. The tag of the signature with least differences is then returned, with a number showing the percentage of similarity.
4. Based on the similarity percentage, it is decided whether the signature is original or not.
5. If the percentage of similarity ranges between 85- 100%, the signature is considered original. This is based on the natural signature recognition method, which says that there are natural differences in the signature of a single person, in the multiple tries.
6. If the percentage of similarity ranges between 75- 85%, the signature is considered Relatively suspicious.
7. If the percentage of similarity is lower than 75%, the signature is considered highly suspicious.

3.5 Use Case Diagram

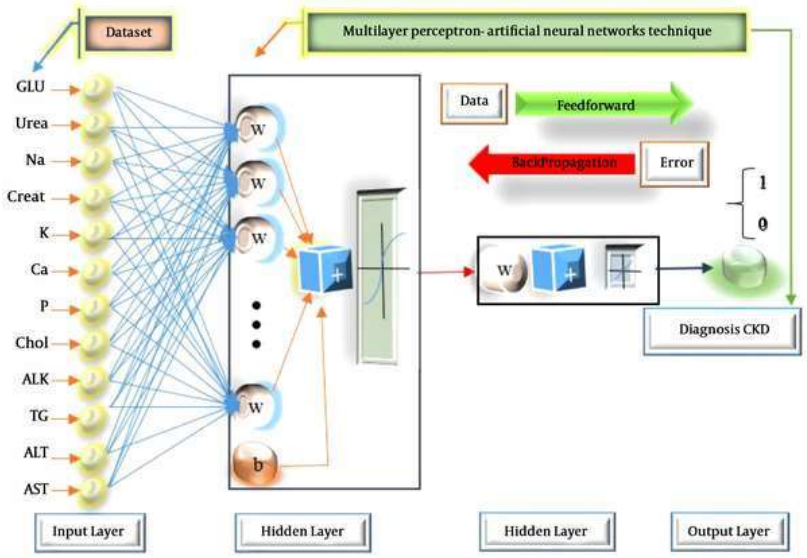

Figure 3.7: Use Case diagram

Use case diagrams are usually referred to as behaviour diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

3.6 Flow Chart

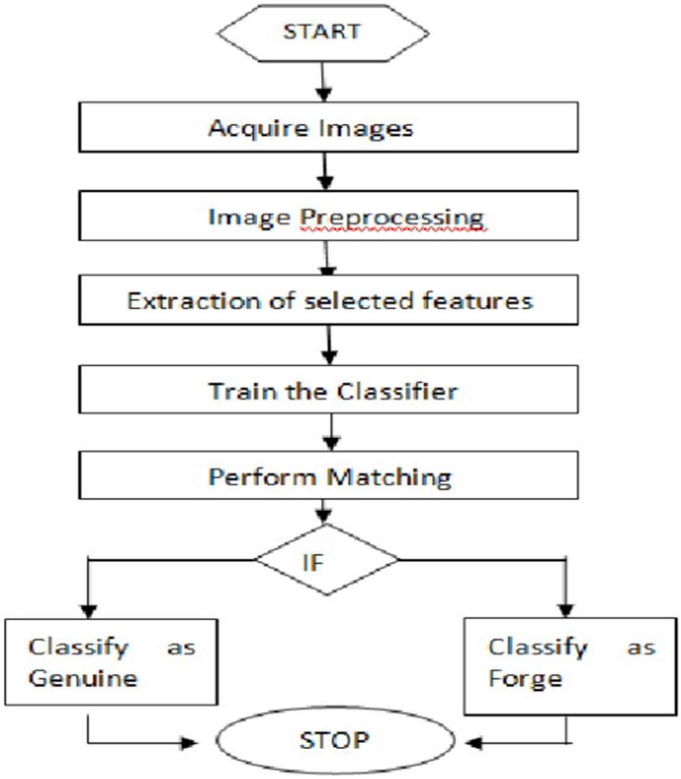

Figure 3.8: Flow chart

Figure 3.7 and Figure 3.8 gives the overview of signature forgery detection system using a MultiLayer Perceptron (MLP). It shows input data represented by Signature Image, feature extraction with Feature Extraction, preprocessing with Preprocessing, and the core neural network structure (MultiLayer Perceptron). The neural network comprises layers (Input Layer, Hidden Layer, Output Layer) and connections (Neuron, Connection). Training involves a dataset (Training Data) and a training algorithm (Training Algorithm). Evaluation measures performance, and dependencies and associations show relationships between components.And, finally gives the output.

4.Implementation

4.1. Testing

The testing phase of the Signature Forgery Detection System involves assessing the model's performance on both the training and independent testing datasets. Through multiple epochs during training, the neural network optimizes parameters to minimize the mean squared difference between predicted and actual labels. The evaluation on the testing dataset includes computing accuracy metrics like precision and recall. The system's real-time testing capability enhances practical applicability, allowing users to verify individual signature images on demand. This testing phase is crucial for validating the system's robustness in accurately distinguishing between genuine and forged signatures, demonstrating its effectiveness in real-world scenarios such as document authentication and transaction security.

4.2 Input

The input is given with the id of the person and followed by the path of signature image as shown in Figure 4.1.

```
Enter person's id : 002
Enter path of signature image : OneDrive\Desktop\MiniProject\forged\021002_002.png

Enter person's id : 002
Enter path of signature image : n\OneDrive\Desktop\MiniProject\real\002002_002.png
```

Figure 4.1 : Input signature images

4.3 Output

The corresponding output is produced as follows

```
Forged Image
False
Genuine Image
True
```

Figure 4.2 : Resultant output

5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

In conclusion, the Signature Forgery Detection System represents a significant advancement in the domain of document security and authentication. By integrating sophisticated feature extraction techniques and a multilayer perceptron neural network, the system demonstrates a high degree of accuracy in distinguishing between genuine and forged handwritten signatures. The extensive literature survey revealed the evolution of signature verification methodologies, emphasizing the transition from traditional feature-based approaches to the more recent adoption of deep learning techniques. The proposed system builds upon these insights, combining the strengths of both traditional and modern methods to achieve improved accuracy and adaptability.

The testing phase confirmed the system's efficacy, showcasing its robust performance on both training and independent testing datasets. The real-time testing capability further enhances the practical utility of the system, providing a versatile tool for signature verification in real-world applications. The comprehensive feature extraction pipeline, encompassing grayscale conversion, binary thresholding, and region-based analysis, ensures that the system captures intricate details crucial for accurate forgery detection.

5.2 Future Work

Looking forward, future enhancements could involve exploring additional image processing techniques, optimizing the neural network architecture, and expanding the dataset to further improve accuracy and adaptability. The Signature Forgery Detection System contributes to the evolving landscape of document security technologies and holds promise for enhancing the security and reliability of signature verification processes in various domains, including financial transactions and legal documentation.

BIBLIOGRAPHY

[1] Foroozandeh, A. &; Hemmat, A.A., 2020. Offline handwritten signature verification and recognition based on Deep Transfer Learning. IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/9187481 [Accessed August 10, 2022].

[2] Kennard, D.J., Barrett, William A. & Sederberg, T.W., 2012. Offline signature verification and forgery detection using a 2-D geometric warping approach. IEEE Xplore.https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6460976 [Accessed August 10, 2022].

[3] Al-Omari, Y.M., Huda Sheikh Abdullah, S.N. &; Omar, K., 2011. State-of-the-art in offline signature verification system. IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/5976912 [Accessed August 11, 2022].

[4] Ghanim, T.M. &; Nabil, A.M., 2018. Offline signature verification and forgery detection approach. IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/8639420 [Accessed August 12, 2022].

[5] Gaikwad, P. et al., 2021. Handwritten signature verification system using machine IJARIIE. Available at: https://ijariie.com/AdminUploadPdf/Handwritten_Signature_Verification_System_using_machine_Learning_Approach._A_Review_of_Literature_ijariie14036.pdf [Accessed August 11, 2022].

[6] M. Arathi & A. Govardhan., 2014. An efficient offline signature verification system - ijmlc.org. Available at: http://www.ijmlc.org/papers/468-A1001.pdf [Accessed August 13, 2022].

[7] L. A. Fakhiroh, A. Fariza and A. Basofi, "Mobile Based Offline Handwritten Signature Forgery Identification using Convolutional Neural Network," Available at: https://ieeexplore.ieee.org/abstract/document/9594019 [Accessed August 8, 2022].

[8] Wang, L. &; Song, T., 2016. An improved digital signature algorithm and authentication protocols in cloud platform. IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/7796194 [Accessed August 7, 2022].

APPENDIX

```
#importing the libraries
import numpy as np
import os
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.cm as cm
from scipy import ndimage
from skimage.measure import regionprops
from skimage import io
from skimage.filters import threshold_otsu   # For finding the threshold for grayscale to
binary conversion
import tensorflow as tf
import pandas as pd
import numpy as np
from time import time
import keras
from tensorflow.python.framework import ops
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()


def rgbgrey(img):
    # Converts rgb to grayscale
    greyimg = np.zeros((img.shape[0], img.shape[1]))    # initializes an empty 2D array
    for row in range(len(img)):
        for col in range(len(img[row])):
            greyimg[row][col] = np.average(img[row][col])
return greyimg     # returns the grayscale image (greyimg) as a NumPy array


# Convert Grey Image to Binary Image:

def greybin(img):
    # Converts grayscale to binary
    blur_radius = 0.8      # the standard deviation of the Gaussian filter
    img = ndimage.gaussian_filter(img, blur_radius)  # to remove small components or noise
#     img = ndimage.binary_erosion(img).astype(img.dtype)
thres = threshold_otsu(img)   # calculates  threshold value based on input grayscale image
    binimg = img > thres
    binimg = np.logical_not(binimg) # Inverts the binary image, making the background white
and the foreground black
```

```
    return binimg  # returns the final binary image as a NumPy array


# Data Preprocessing and Feature Extraction:-

def preproc(path, img=None, display=True):
    # If 'img' is not provided, load the image from the specified path
    if img is None:
        img = mpimg.imread(path)

    # Display the original image if 'display' is True
    if display:
        plt.imshow(img)
        plt.show()

    # Convert the RGB image to grayscale using the 'rgbgrey' function
    grey = rgbgrey(img)

    # Display the grayscale image if 'display' is True
    if display:
        plt.imshow(grey, cmap=matplotlib.cm.Greys_r)
        plt.show()

    # Convert the grayscale image to binary using the 'greybin' function
    binimg = greybin(grey)

    # Display the binary image if 'display' is True
    if display:
        plt.imshow(binimg, cmap=matplotlib.cm.Greys_r)
        plt.show()

    # Extract the coordinates of white pixels in the binary image
    r, c = np.where(binimg == 1)

    # Create a bounding box around the signature by cropping the image
    signimg = binimg[r.min():r.max(), c.min():c.max()]

    # Display the cropped signature image if 'display' is True
    if display:
        plt.imshow(signimg, cmap=matplotlib.cm.Greys_r)
        plt.show()

    # Return the preprocessed signature image
return signimg
```

```
# Ratio
def Ratio(img):
    a = 0  # returns the final binary image as a NumPy array
    for row in range(len(img)):
        for col in range(len(img[0])):
            if img[row][col] == True:
                a = a + 1
    total = img.shape[0] * img.shape[1]  # Calculates the total number of pixels in the image
     return a / total   # Returns the ratio of white pixels to the total number of pixels in the image


# Centroid
def Centroid(img):
    numOfWhites = 0
    a = np.array([0, 0])  # Adds the coordinates of the current white pixel to the cumulative sum
    # Iterate through each pixel in the binary image
    for row in range(len(img)):
        for col in range(len(img[0])):
            if img[row][col] == True:
                b = np.array([row, col])
                a = np.add(a, b)
                numOfWhites += 1


    # Create an array representing the number of rows and columns in the image
    rowcols = np.array([img.shape[0], img.shape[1]])


    # Calculate the centroid coordinates
    centroid = a / numOfWhites   # Calculates the average coordinates of the white pixels
    centroid = centroid / rowcols  # Normalizes the centroid coordinates
    # Return the centroid coordinates as a tuple
    return centroid[0], centroid[1]
#EccentricitySolidity
def EccentricitySolidity(img):
```

```
    # Compute region properties using regionprops
    r = regionprops(img.astype("int8"))  # The result of regionprops is a list of region properties. In this case, it's stored in the variable r.
    # Return eccentricity and solidity of the identified region
    return r[0].eccentricity, r[0].solidity
#eccentricity: This property measures how much an ellipse deviates from a perfect circle. Values closer to 0 indicate a more circular shape, while values closer to 1 indicate a more elongated shape.
#solidity: This property represents the ratio of the area of the region to the area of its convex hull. It provides information about the "compactness" of the region. Values range from 0 to 1, where 1 indicates a perfectly solid and compact region.
```

#SkewKurtosis

```
def getFeatures(path, img=None, display=False):
    # If 'img' is not provided, load the image from the specified path
    if img is None:
        img = mpimg.imread(path)
    # Perform preprocessing on the image using the 'preproc' function
    img = preproc(path, display=display)
    # Calculate ratio of white pixels to total pixels using the 'Ratio' function
    ratio = Ratio(img)
    # Calculate centroid coordinates using the 'Centroid' function
    centroid = Centroid(img)
    # Calculate eccentricity and solidity using the 'EccentricitySolidity' function
    eccentricity, solidity = EccentricitySolidity(img)
    # Calculate skewness and kurtosis using the 'SkewKurtosis' function
    skewness, kurtosis = SkewKurtosis(img)
    # Create a tuple containing all the extracted features
    retVal = (ratio, centroid, eccentricity, solidity, skewness, kurtosis)
    # Return the tuple of features
    return retVal
```

#GetFeatures

```
def getCSVFeatures(path, img=None, display=False):
    # If 'img' is not provided, load the image from the specified path
    if img is None:
        img = mpimg.imread(path)
    # Retrieve features using the 'getFeatures' function
    temp = getFeatures(path, display=display)

    # Extract specific elements from the tuple of features for CSV output
    features = (
        temp[0],        # Ratio
        temp[1][0],     # Centroid x-coordinate
        temp[1][1],     # Centroid y-coordinate
        temp[2],        # Eccentricity
        temp[3],        # Solidity
        temp[4][0],     # Skewness along x-axis
        temp[4][1],     # Skewness along y-axis
        temp[5][0],     # Kurtosis along x-axis
        temp[5][1]      # Kurtosis along y-axis
    )
```

#Making CSV File

```
def makeCSV():
    # Create necessary folders if they do not exist
    if not os.path.exists('C:\\Users\\Tharun\\OneDrive\\Desktop\\MiniProject\\Features'):
        os.mkdir('C:\\Users\\Tharun\\OneDrive\\Desktop\\MiniProject\\Features')
        print('New folder "Features" created')
ifnot(os.path.exists('C:\\Users\\Tharun\\OneDrive\\Desktop\\MiniProject\\Features\\Training'):
        os.mkdir('C:\\Users\\Tharun\\OneDrive\\Desktop\\MiniProject\\Features\\Training')
        print('New folder "Features/Training" created')
if not(os.path.exists('C:\\Users\\Tharun\\OneDrive\\Desktop\\MiniProject\\Features\\Testing')):
        os.mkdir('C:\\Users\\Tharun\\OneDrive\\Desktop\\MiniProject\\Features\\Testing')
        print('New folder "Features/Testing" created')
```

```
    # Genuine and forged signatures paths
    gpath = genuine_image_paths
    fpath = forged_image_paths

    # Loop through each person
    for person in range(1, 13):
        per = ('00' + str(person))[-3:]
        print('Saving features for person id-', per)

        # Save features for the training set
        with
open('C:\\Users\\Tharun\\OneDrive\\Desktop\\MiniProject\\Features\\Training/training_'+per
+'.csv', 'w') as handle:

handle.write('ratio,cent_y,cent_x,eccentricity,solidity,skew_x,skew_y,kurt_x,kurt_y,output\n')
            # Training set - Genuine signatures
            for i in range(0, 3):
                source = os.path.join(gpath, per + per + '_00' + str(i) + '.png')
                features = getCSVFeatures(path=source)
                handle.write(','.join(map(str, features)) + ',1\n')
            # Training set - Forged signatures
            for i in range(0, 3):
                source = os.path.join(fpath, '021' + per + '_00' + str(i) + '.png')
                features = getCSVFeatures(path=source)
                handle.write(','.join(map(str, features)) + ',0\n')

        # Save features for the testing set
        with
open('C:\\Users\\Tharun\\OneDrive\\Desktop\\MiniProject\\Features\\Testing/testing_'+per+'.
csv', 'w') as handle:

handle.write('ratio,cent_y,cent_x,eccentricity,solidity,skew_x,skew_y,kurt_x,kurt_y,output\n')
            # Testing set - Genuine signatures
```

```
        for i in range(3, 5):
            source = os.path.join(gpath, per + per + '_00' + str(i) + '.png')
            features = getCSVFeatures(path=source)
            handle.write(','.join(map(str, features)) + ',1\n')
        # Testing set - Forged signatures
        for i in range(3, 5):
            source = os.path.join(fpath, '021' + per + '_00' + str(i) + '.png')
            features = getCSVFeatures(path=source)
            handle.write(','.join(map(str, features)) + ',0\n')
```

Testing the Features

```
def testing(path):
    # Extract features using the 'getCSVFeatures' function
    feature = getCSVFeatures(path)
    # Create a folder for storing test features if it does not exist
    if not os.path.exists('C:/Users/Tharun/OneDrive/Desktop/MiniProject/TestFeatures'):
        os.mkdir('C:/Users/Tharun/OneDrive/Desktop/MiniProject/TestFeatures')
    # Save features to a CSV file
    with  open('C:/Users/Tharun/OneDrive/Desktop/MiniProject/TestFeatures/testcsv.csv',  'w')
as handle:
        # Write header to the CSV file
        handle.write('ratio,cent_y,cent_x,eccentricity,solidity,skew_x,skew_y,kurt_x,kurt_y\n')
        # Write the extracted features to the CSV file
        handle.write(','.join(map(str, feature)) + '\n')
```

#Buliding the Model

```
# Create model
def multilayer_perceptron(x):
    layer_1 = tf.tanh((tf.matmul(x, weights['h1']) + biases['b1']))
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_3 = tf.add(tf.matmul(layer_2, weights['h3']), biases['b3'])
    out_layer = tf.tanh(tf.matmul(layer_1, weights['out']) + biases['out'])
    return out_layer
```

```
# Construct model
logits = multilayer_perceptron(X)
# Define loss and optimizer
loss_op = tf.reduce_mean(tf.squared_difference(logits, Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)
# For accuracies
pred = tf.nn.softmax(logits)  # Apply softmax to logits
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

#Train and Evaluate the Model

```
def evaluate(train_path, test_path, type2=False):
    # ... (loading data and setup)
    with tf.Session() as sess:
        sess.run(init)
        # Training cycle
        for epoch in range(training_epochs):
            _, cost = sess.run([train_op, loss_op], feed_dict={X: train_input, Y: corr_train})
            if cost < 0.0001:
                break

        accuracy1 = accuracy.eval({X: train_input, Y: corr_train})
        if type2 is False:
            accuracy2 = accuracy.eval({X: test_input, Y: corr_test})
            return accuracy1, accuracy2
        else:
            prediction = pred.eval({X: test_input})
            if prediction[0][1] > prediction[0][0]:
                print('Genuine Image')
                return True
```

```
        else:
            print('Forged Image')
            return False
```

Testing the Model

```
def trainAndTest(rate=0.001, epochs=1700, neurons=7, display=False):
    # ... (parameter setup)
    for i in range(1, n+1):
        temp = ('0' + str(i))[-2:]
        train_score, test_score = evaluate(train_path.replace('01', temp), test_path.replace('01', temp))
        train_avg += train_score
        test_avg += test_score
    # ... (performance metrics calculation and display)
   return train_avg/n, test_avg/n, (time()-start)/n
```

#Evaluating the Model

```
evaluate(train_path, test_path, type2=True)
```