



**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
KALYANI**

(Autonomous institution under MoE, Govt. of India &

Department of Information Technology & Electronics, Govt. of West Bengal)

Address: (c/o WEBEL IT Park, Opposite of Kalyani Water Treatment Plant,
Near Buddha Park, Dist. Nadia, Kalyani - 741235, West Bengal)

Email: office@iiitkalyani.ac.in, Website: www.iiitkalyani.ac.in

Signature Verification Using CNN Algorithm

PROJECT – REPORT

By

Abhishek Pal/856

Aditya Vishwakarma/860

Amit Chausali/866

Arindam Mondal/880

Under the guidance of:

Dr. Sudeshna Mondal

Contents

Abstract	1
1 Introduction	2
1.1 Background	2
1.2 Problem Statement	2
1.3 Scope	3
1.4 Supervised Learning in Signature Verification	3
2 Literature Review	4
2.1 Signature Verification Methods	4
3 Convolutional Neural Networks (CNN)	5
3.1 Convolutional Layers	5
3.1.1 Example Calculations	6
3.2 Stride and Padding	6
3.3 Pooling Layers	7
3.4 Fully Connected Layers	7
3.5 Forward Propagation	7
4 Dataset and Preprocessing	8
4.1 Loading the Images	8
4.2 Creating Labels	9
4.3 Combining the Data	9
4.4 Splitting the Data	10
4.5 Normalization of Images	10
4.6 Data Augmentation	10
4.7 Building the CNN Model	11

4.8	Modifying CNN for Signature Verification	12
5	Essential Functions Used	13
5.1	Adam Optimizer	13
5.2	ReLU Activation Function	14
5.2.1	<i>Why is ReLU Popular?</i>	14
5.3	Sigmoid Activation Function	14
5.3.1	<i>Derivative of the Sigmoid Function</i>	15
5.4	Binary Cross-Entropy Loss	15
5.4.1	<i>Understanding Binary Cross-Entropy</i>	16
6	Model Training	17
6.1	Training the Model	17
6.2	Model Evaluation	17
6.3	Code Implementation	18
7	Results and Discussion	20
7.1	Table of Accuracy and Loss	20
7.2	Graphical Representation of Results	20
7.3	Performance Metrics and Confusion Matrix	21
7.3.1	<i>Epoch 10 for Kaggle dataset</i>	21
7.3.2	<i>Epoch 50 for Kaggle dataset</i>	22
7.3.3	<i>Epoch 100 for Kaggle dataset</i>	22
7.3.4	<i>Epoch 500 for Kaggle dataset</i>	23
7.3.5	<i>Epoch 500 for our dataset</i>	23
7.4	Discussion	24
8	Conclusion and Future Works	25
8.1	Summary	25
8.2	Future Directions	25

Abstract

This thesis is about using Convolutional Neural Networks (CNN) for offline signature verification. It's part of my third-year project and focuses on providing a secure way to verify a person's identity in fields like banking and other organizations. Recent progress in deep learning, especially with CNNs, has made them useful for tasks like identifying signatures. For this project, we used a combination of our own signature database along with additional samples from a Kaggle dataset. These databases include both real and forged signatures, which were used to train and test the CNN model to recognize genuine signatures and detect forgeries.

Chapter 1

Introduction

Signatures are important for verifying identity in financial services and official documents like checks and letters. Even if someone signs twice, small differences will appear, but certain features like how letters are shaped, writing speed, and the slant of the signature stay consistent. These similarities help confirm if a signature is genuine. In banking, signatures are especially important for tasks like approving checks or verifying actions, making them a key part of ensuring trust and security.

1.1 Background

There are two main types of signature verification: offline and online. Offline verification looks at static images of signatures, while online verification uses dynamic data such as the speed and pressure of the writing. Offline verification is more widely used in real-life situations where only images of signatures are available.

Recently, Convolutional Neural Networks (CNNs) have become popular for processing images. They can automatically extract important features without needing to be manually programmed. CNNs are good at identifying complex patterns, which makes them useful for telling apart genuine signatures from forged ones.

1.2 Problem Statement

Authentic signatures can vary due to factors like writing speed, pressure, or the surface, making verification tricky. Forgeries, from simple copies to skilled fakes, add to the

challenge. Traditional systems often fail to handle these variations, leading to errors. This project uses a CNN-based approach to overcome these issues with improved accuracy and reliability.

1.3 Scope

This study focuses on offline signature verification using CNNs to analyze still images of signatures. By applying supervised learning, we train the model with labeled data—identifying each signature as genuine or forged. Unlike online methods that track signing in real-time, this project targets dependable offline verification in controlled scenarios, leaving out cases like heavily damaged images or complex forgeries.

1.4 Supervised Learning in Signature Verification

Labeled Data: The CNN is trained on a dataset of labeled signature images as "genuine" or "forged," using multiple samples per person to learn individual traits.

Learning Patterns: It identifies unique features in genuine signatures, such as curves and strokes, while detecting inconsistencies in forged ones.

Mapping Inputs to Outputs: The model maps each image to a label through convolutional layers that extract key features.

Binary Classification: Signature verification is treated as a binary problem where the CNN classifies signatures based on a probability score.

Training Process: Through labeled examples, the CNN learns to distinguish between genuine and forged signatures and verify new ones effectively.

This approach ensures reliable verification by focusing on unique traits of genuine signatures and spotting forgery discrepancies.

Chapter 2

Literature Review

Signature verification has become an important tool in security, especially for applications like banking and legal documentation, where verifying a person's identity is essential. Traditional methods relied on manually analyzing features of signatures, such as stroke length, angles, and curvature. Although these techniques can be effective, they often require a high level of expertise and are limited by human judgment.

2.1 Signature Verification Methods

Signature verification has evolved from traditional methods requiring manual feature extraction, such as analyzing curves, angles, and stroke lengths, to advanced machine learning techniques like SVM, k-NN, and decision trees. While these methods improve accuracy, they rely on subjective feature selection.

Deep learning, particularly CNNs, has further advanced the field by automating feature extraction and capturing complex patterns in signature images. However, our current implementation achieves excellent results without specialized algorithms or architectures.

This semester focuses on these results, and next semester, we plan to explore advanced techniques.

Chapter 3

Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a type of deep learning model particularly effective for image data due to their specialized layers and techniques that automatically extract important visual features. Here's an overview of the main components of CNNs, including additional methods and forward propagation calculations.

3.1 Convolutional Layers

- **Purpose:** Convolutional layers apply filters (kernels) that scan across the image to capture details like edges, textures, and patterns.
- **Mathematics:** In each convolution operation, the filter (a small matrix) slides or "convolves" over the input image, performing an element-wise multiplication and summing the results. This output is called the feature map.
- **Formula:**

$$\text{Output}_{i,j} = \sum_k \sum_l (\text{Input}_{i+k,j+l} \cdot \text{Filter}_{k,l})$$

where i, j are positions on the feature map, and k, l are positions within the filter.

We are given the following input image (x) and filter (w):

$$\text{Input Image (x)} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$\text{Filter (w)} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Let the bias (b) be $b = 1$.

3.1.1 Example Calculations

- **For position $(i, j) = (0, 0)$:** The filter covers the top-left 2×2 region:

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix}$$

Convolution operation:

$$(1 \cdot 1) + (0 \cdot 2) + (0 \cdot 4) + (-1 \cdot 5) = -4$$

Adding bias:

$$-4 + 1 = -3$$

- **Final Output Feature Map:** After applying the filter to the entire image:

$$\text{Output} = \begin{bmatrix} -3 & -3 \\ -3 & -3 \end{bmatrix}$$

3.2 Stride and Padding

- **Stride:** Controls the movement of the filter across the image. For stride S , the output size is:

$$\text{Output Size} = \frac{N - F}{S} + 1$$

- **Padding:** Adds pixels around the image to control output size. For padding P ,

the output size is:

$$\text{Output Size} = \frac{N + 2P - F}{S} + 1$$

where N : Input size, P : Padding, F : Filter size, S : Stride.

3.3 Pooling Layers

- **Purpose:** Reduces spatial dimensions while preserving features. Common methods: max pooling and average pooling.
- **Max Pooling:** Selects the maximum value in a defined window.

3.4 Fully Connected Layers

- **Purpose:** Maps extracted features to predictions (e.g., genuine or forged).
- **Mathematics:** Each neuron computes:

$$\text{Output} = f \left(\sum_i (\text{Weight}_i \cdot \text{Input}_i + \text{Bias}) \right)$$

3.5 Forward Propagation

- **Convolutional Layer Output:**

$$\text{Output}_{i,j} = f \left(\sum_{k,l} (\text{Input}_{i+k,j+l} \cdot \text{Filter}_{k,l} + \text{Bias}) \right)$$

- **Pooling Layer Output:** For a region,

$$\text{Output}_{i,j} = \max(\text{Region})$$

Chapter 4

Dataset and Preprocessing

The signature verification project utilizes two datasets: a custom dataset and a Kaggle dataset, each divided into genuine and forged signature categories, stored in separate directories. The preprocessing pipeline ensures the images are correctly formatted and normalized to train a CNN for binary classification, distinguishing between genuine and forged signatures. This approach combines diverse data sources to enhance the model's robustness and performance.

4.1 Loading the Images

The dataset is first loaded using the function `load_images_from_folder`, which scans through the directories of genuine and forged signatures. Each image is read using OpenCV's `cv2.imread` function, and if the image is successfully loaded, it is resized to a standard size of 128×128 pixels. Resizing the images ensures uniformity, as the CNN model requires consistent input dimensions. Resizing also speeds up computation, especially when dealing with large datasets.

- **Input Dimensions:** The resizing to 128×128 pixels helps ensure that the input to the CNN is consistent. This size strikes a balance between image quality and computational efficiency.
- **Handling Missing or Corrupted Images:** Images that fail to load are skipped, ensuring that only valid images are included in the final dataset.

Once all the images are loaded and resized, they are stored in two arrays, one for genuine signatures and another for forged signatures. The next step involves labeling the images: genuine signatures are labeled as 1, and forged signatures are labeled as 0. This labeling is crucial as it allows the model to distinguish between the two categories during training.

4.2 Creating Labels

The next step involves creating labels for the images:

- **Genuine Signatures:** Each genuine signature is assigned a label of 1.
- **Forged Signatures:** Each forged signature is assigned a label of 0.

The labels for the genuine and forged images are stored in separate arrays. These arrays are then combined into a single array `y`, which holds the labels for all the images in the dataset.

4.3 Combining the Data

At this point, the images from both categories are combined into a single array `X`. This array contains all the image data, with genuine and forged signatures mixed together. The corresponding labels are also combined into a single array `y` using `np.concatenate` to align the labels with the images.

- **Shuffling the Data:** To ensure that the model is not biased by the order of the images (e.g., all genuine signatures followed by all forged signatures), the dataset is shuffled using `shuffle` from `sklearn.utils`. This helps prevent the model from learning the order of the samples and encourages generalization.
- **Shuffling Randomization:** The dataset is shuffled using a fixed random state (42) to ensure reproducibility of the results across different runs of the experiment.

4.4 Splitting the Data

Next, the dataset is split into training and testing sets using `train_test_split` from `sklearn.model_selection`. The training set consists of 80% of the data, while the remaining 20% is used for testing the model's performance. This split allows the model to be trained on a large portion of the dataset and evaluated on a separate portion to gauge its generalization ability.

- **Training Set:** The training set contains 80% of the dataset, which is used to teach the model to distinguish between genuine and forged signatures.
- **Test Set:** The test set contains 20% of the dataset and is used to evaluate the model's performance after training.

4.5 Normalization of Images

Before feeding the images into the model, the pixel values are normalized. This step ensures that the pixel values range between 0 and 1, which is important for the training process. The images are divided by 255.0, as the original pixel values range from 0 to 255. Normalization helps the model converge faster and improves its accuracy by preventing the model from being biased by large values during optimization.

- **Normalized Data:** The training and test datasets (`X_train` and `X_test`) are divided by 255.0 to scale the pixel values into a range of $[0, 1]$. This helps the optimizer make faster progress during training.

4.6 Data Augmentation

Data augmentation is applied to artificially increase the size of the training dataset. By performing various transformations on the input images, the model becomes more robust and less prone to overfitting. The following transformations are applied to the training data:

- **Rotation:** Random rotations within a range of 10 degrees.

- **Width and Height Shifts:** Images are shifted horizontally and vertically by up to 10%.
- **Zoom:** Random zooming is applied to the images within a range of 10%.
- **Horizontal Flip:** Images are randomly flipped horizontally.

These transformations are applied using `ImageDataGenerator`, which applies them in real-time during training. This increases the diversity of the training data and helps the model generalize better on unseen data.

Summary

In summary, the dataset for signature verification is carefully prepared by loading, labeling, and combining genuine and forged signature images. After splitting the dataset into training and testing sets, the images are normalized to a consistent range. Data augmentation is applied to increase the variability of the training data and improve the model's ability to generalize. With this preprocessing pipeline, the dataset is now ready for training the signature verification model.

4.7 Building the CNN Model

The CNN architecture is built using the Keras library, which provides a simple interface for defining deep learning models. The model is structured as follows:

- **Input Layer:** The input layer expects images of size $128 \times 128 \times 3$, where 128 is the image size and 3 represents the three color channels (RGB).
- **Convolutional Layers:** The model contains three convolutional layers with increasing filters (32, 64, 128) that automatically learn features from the images during training. These layers use ReLU activation to introduce non-linearity.
- **Pooling Layers:** MaxPooling is applied after each convolutional layer to reduce the spatial dimensions of the feature maps, helping to lower the computational complexity and number of parameters.

- **Flatten Layer:** After the convolutional and pooling layers, the feature maps are flattened into a 1D vector to be passed into the fully connected layers.
- **Fully Connected Layers:** A dense layer with 128 units uses **ReLU** activation for feature learning, followed by a final dense layer with a single neuron and a **sigmoid** activation function for binary classification (genuine or forged).
- **Dropout:** Dropout with a rate of 0.5 is applied to the dense layer to reduce overfitting by randomly setting a fraction of input units to 0 during each update cycle.

The CNN model is compiled using the **Adam** optimizer with a learning rate of 0.001, which ensures efficient and adaptive gradient updates. The loss function used is binary cross-entropy, as this is a binary classification problem.

4.8 Modifying CNN for Signature Verification

- **Custom Architecture:** The CNN uses three convolutional layers with increasing filters (32, 64, 128) and **ReLU** activation, followed by MaxPooling layers for down-sampling. A dense layer with 128 units and a final sigmoid layer handles binary classification (genuine or forged).
- **Regularization:** Dropout with a rate of 0.5 is applied in the dense layer to reduce overfitting.
- **Loss Function and Optimizer:** The model is compiled using Binary Cross-Entropy as the loss function and Adam optimizer with a learning rate of 0.001 for effective training.
- **Evaluation Metrics:** Accuracy is tracked during training, and additional metrics such as confusion matrix, precision, recall, and F1-score are computed using the test set for comprehensive evaluation.
- **Thresholding:** Predictions are converted to binary (0 or 1) using a threshold of 0.5 for decision-making.

Chapter 5

Essential Functions Used

5.1 Adam Optimizer

Adam (Adaptive Moment Estimation) is an optimization technique for gradient descent. The method is highly efficient when working with large problems involving a lot of data or parameters. It requires less memory and is more efficient. Intuitively, it combines the 'gradient descent with momentum' algorithm and the 'RMSProp' algorithm.

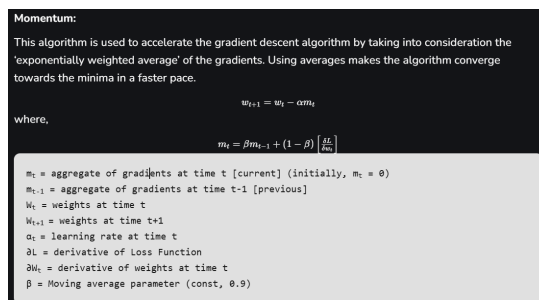


Figure 5.1: Momentum in Adam Optimizer

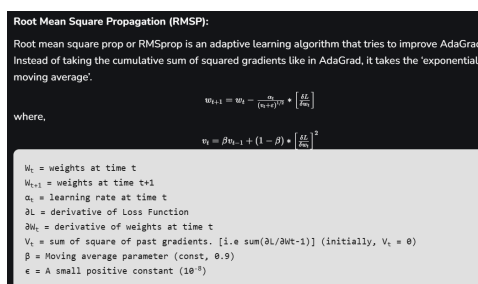


Figure 5.2: RMSP in Adam Optimizer

5.2 ReLU Activation Function

The ReLU (Rectified Linear Unit) activation function allows the model to learn hierarchical features by introducing non-linearity, making it effective for extracting meaningful patterns in data. In simpler terms, ReLU allows positive values to pass through unchanged while setting all negative values to zero.

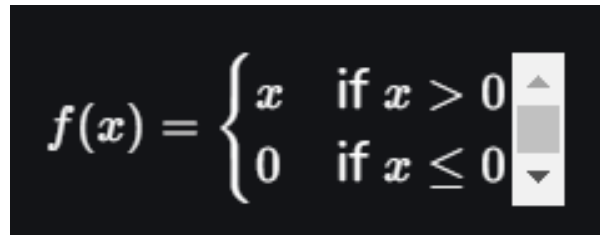

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Figure 5.3: ReLU Activation Function

where,

- x is the input to the neuron.
- The function returns x if x is greater than 0.
- If x is less than or equal to 0, the function returns 0.

5.2.1 Why is ReLU Popular?

1. **Simplicity:** ReLU is computationally efficient as it involves only a thresholding operation. This simplicity makes it easy to implement and compute, which is crucial when training deep neural networks with millions of parameters.
2. **Non-Linearity:** Although it seems like a piecewise linear function, ReLU is still a non-linear function. This allows the model to learn more complex data patterns and model intricate relationships between features.

5.3 Sigmoid Activation Function

The sigmoid function is a mathematical function that produces an S-shaped curve. It maps any real-valued number into a value between 0 and 1.

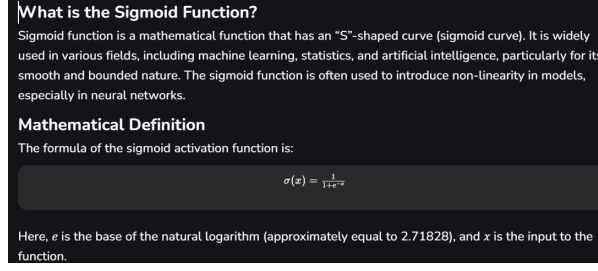


Figure 5.4: Sigmoid Activation Function

5.3.1 Derivative of the Sigmoid Function

The sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$, has the following derivative:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Let's derive this derivative as follows:

1. Let $y = \sigma(x) = \frac{1}{1+e^{-x}}$ 2. Let $u = 1 + e^{-x}$. Thus, $y = \frac{1}{u}$. 3. The derivative of u with respect to x is:

$$\frac{du}{dx} = -e^{-x}$$

4. The derivative of y with respect to u is:

$$\frac{dy}{du} = -\frac{1}{u^2}$$

5. Apply the chain rule:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = -\frac{1}{u^2} \cdot (-e^{-x}) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

Thus, we can express this as:

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x))$$

5.4 Binary Cross-Entropy Loss

Binary Cross-Entropy (BCE), also known as log loss, is a crucial concept in binary classification problems within machine learning and statistical modeling. It measures the

performance of a classification model whose output is a probability value between 0 and 1.

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

- N = Batch size (the number of data points in the batch).
- y_i = True label for the i -th data point (either 0 or 1).
- p_i = Predicted probability that the i -th data point belongs to class 1.
- $\log(p_i)$ = The logarithm of the predicted probability for class 1.
- $(1 - y_i)$ = Complement of the true label for the i -th data point. This is 1 when the true label is 0, and 0 when the true label is 1.
- $\log(1 - p_i)$ = The logarithm of the predicted probability for class 0.
- $\sum_{i=1}^N$ = Summation over all data points in the batch.

Figure 5.5: Binary Cross-Entropy Loss Function

5.4.1 Understanding Binary Cross-Entropy

Binary Cross-Entropy quantifies the difference between two probability distributions—the true labels and the predicted probabilities. It is calculated as follows:

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Where:

- N is the number of batch size.
- y_i is the true label for the i -th sample.
- \hat{y}_i is the predicted probability for the i -th sample.
- \log denotes the natural logarithm.

The BCE loss increases as the predicted probability diverges from the actual label. A perfect model would have a BCE of 0, indicating perfect prediction accuracy.

Chapter 6

Model Training

6.1 Training the Model

The model is trained using the preprocessed training data. The training set consists of images and corresponding labels that are passed into the model in batches. The following steps outline the training process:

- **Batch Size:** The training data is divided into small batches to prevent memory overload and speed up the computation. A typical batch size for training a CNN is 32.
- **Epochs:** An epoch represents one complete pass through the entire training dataset. The model is trained for a predetermined number of epochs (e.g., 20 epochs).
- **Validation Set:** The validation data is used to monitor the performance of the model during training. It helps in tuning the hyperparameters and provides an estimate of how well the model will perform on unseen data.

6.2 Model Evaluation

After training, the model is tested on new data (test set) to evaluate its performance. The key evaluation methods are:

- **Accuracy:** Measures the percentage of correctly classified signatures.

- **Confusion Matrix:** Provides detailed results, including:
 - Genuine signatures correctly identified (*true positives*).
 - Forged signatures wrongly labeled as genuine (*false positives*).
 - Forged signatures correctly identified (*true negatives*).
 - Genuine signatures wrongly labeled as forged (*false negatives*).
- **Precision, Recall, F1-Score:**
 - *Precision*: How often the "genuine" predictions are correct.
 - *Recall*: How well the model catches all genuine signatures.
 - *F1-Score*: Balances precision and recall.

-

6.3 Code Implementation

To provide a clear understanding of the model training process, the following code snippet outlines the implementation using Python and the Keras library. The code highlights the architecture and training pipeline for the CNN model.

```

model = Sequential()

# Convolutional layers
model.add(Conv2D(32, (3, 3), input_shape=(128, 128, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the output
model.add(Flatten())

# Dense layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5)) # To avoid overfitting
model.add(Dense(1, activation='sigmoid')) # Binary classification

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy']) # Use learning_rate instead of lr

```

Figure 6.1: Convolutional Layers for features extraction

```
▶ history = model.fit(datagen.flow(x_train, y_train, batch_size=60),  
                      epochs=500,  
                      validation_data=(x_test, y_test))
```

Figure 6.2: Dataset Parsing according to epoch.

```
▶ acc = history.history['accuracy']  
val_acc = history.history['val_accuracy']  
loss = history.history['loss']  
val_loss = history.history['val_loss']  
  
epochs = range(len(acc))  
  
plt.figure(figsize=(25, 7))  
plt.subplot(1, 2, 1)  
plt.plot(epochs, acc, 'b', label='Training Accuracy')  
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')  
plt.title('Training and Validation Accuracy')  
plt.legend()  
  
plt.subplot(1, 2, 2)  
plt.plot(epochs, loss, 'b', label='Training Loss')  
plt.plot(epochs, val_loss, 'r', label='Validation Loss')  
plt.title('Training and Validation Loss')  
plt.legend()  
  
plt.show()
```

Figure 6.3: Plot creating using Matplotlib.

Chapter 7

Results and Discussion

This chapter presents the results obtained from the model training and evaluation process. The results are analyzed using various performance metrics such as accuracy, precision, recall, F1-score, and the confusion matrix. Additionally, visualizations such as graphs are included to provide a better understanding of the model's performance.

7.1 Table of Accuracy and Loss

Epoch	Accuracy	Loss
10	0.5134	0.6930
50 own	0.5742	0.5525
100	0.92	0.25
500 own	0.9470	0.1309
500 kag	0.9167	0.2049

Table 7.1: Performance Metrics for Different Epochs.

7.2 Graphical Representation of Results

The performance of the CNN model during training and validation is visualized using various graphs, including accuracy and loss curves. These graphs illustrate how the model's performance improved over multiple epochs.

7.3 Performance Metrics and Confusion Matrix

The confusion matrix highlights the model's strengths and weaknesses. For instance, a high number of false positives may indicate difficulty in distinguishing specific forged signatures.

- **Accuracy:** The proportion of correctly classified samples.
- **Precision:** The fraction of relevant instances among the retrieved instances.
- **Recall (Sensitivity):** The model's ability to detect genuine signatures.
- **F1-Score:** The harmonic mean of precision and recall.
- **ROC-AUC:** The Area Under the ROC Curve, measuring the balance between true positives and false positives.

7.3.1 Epoch 10 for Kaggle dataset

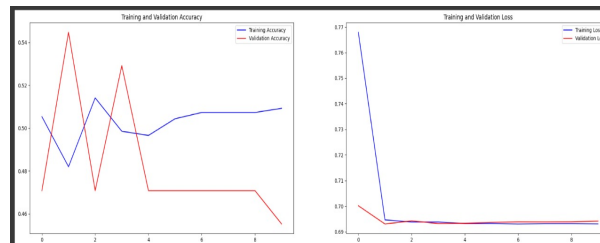


Figure 7.1: Training and Validation Accuracy for Epoch 10.

```

9/9 4s 375ms/step
[[ 81 40]
 [100 36]]

```

	precision	recall	f1-score	support
0.0	0.45	0.67	0.54	121
1.0	0.47	0.26	0.34	136
accuracy			0.46	257
macro avg	0.46	0.47	0.44	257
weighted avg	0.46	0.46	0.43	257

Figure 7.2: Confusion Matrix for Epoch 10.

7.3.2 Epoch 50 for Kaggle dataset

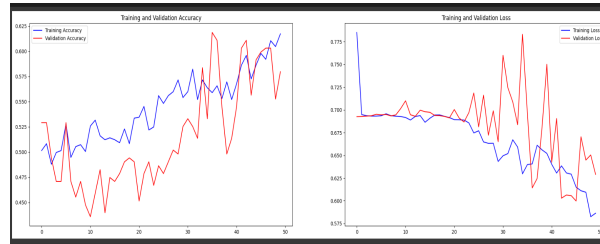


Figure 7.3: Training and Validation Accuracy for Epoch 50.

```
[[83 38]
 [70 66]]
```

	precision	recall	f1-score	support
0.0	0.54	0.69	0.61	121
1.0	0.63	0.49	0.55	136
accuracy			0.58	257
macro avg	0.59	0.59	0.58	257
weighted avg	0.59	0.58	0.58	257

Figure 7.4: Confusion Matrix for Epoch 50.

7.3.3 Epoch 100 for Kaggle dataset

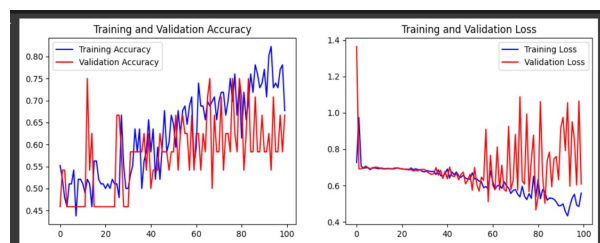


Figure 7.5: Training and Validation Accuracy for Epoch 100.

```
[[11  0]
 [ 8  5]]
```

	precision	recall	f1-score	support
0.0	0.58	1.00	0.73	11
1.0	1.00	0.38	0.56	13
accuracy			0.67	24
macro avg	0.79	0.69	0.64	24
weighted avg	0.81	0.67	0.64	24

Figure 7.6: Confusion Matrix for Epoch 100.

7.3.4 Epoch 500 for Kaggle dataset

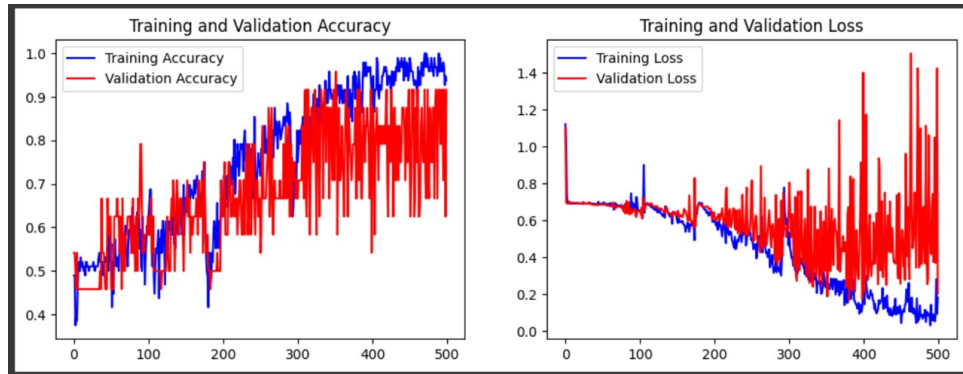


Figure 7.7: Training and Validation Accuracy for Epoch 500.

```
[[11  0]
 [ 2 11]]
```

	precision	recall	f1-score	support
0.0	0.85	1.00	0.92	11
1.0	1.00	0.85	0.92	13
accuracy			0.92	24
macro avg	0.92	0.92	0.92	24
weighted avg	0.93	0.92	0.92	24

Figure 7.8: Confusion Matrix for Epoch 500.

7.3.5 Epoch 500 for our dataset

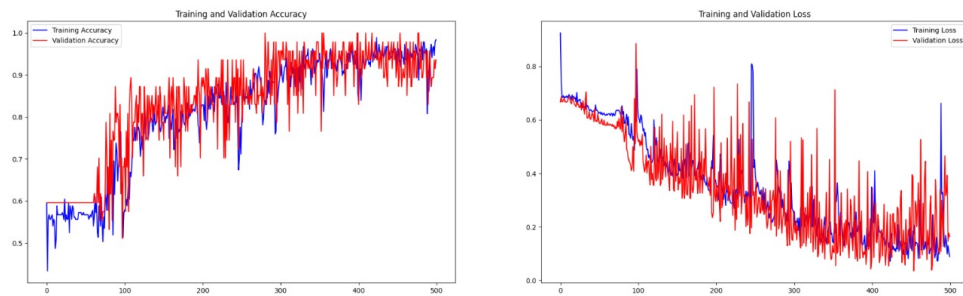


Figure 7.9: Training and Validation Accuracy for Epoch 500.

```
[[25  3]
 [ 0 19]]
```

	precision	recall	f1-score	support
0.0	1.00	0.89	0.94	28
1.0	0.86	1.00	0.93	19
accuracy			0.94	47
macro avg	0.93	0.95	0.94	47
weighted avg	0.94	0.94	0.94	47

Figure 7.10: Confusion Matrix for Epoch 500.

7.4 Discussion

The results indicate that the model improves steadily with each epoch. The training and validation accuracy increase while the loss decreases, demonstrating effective learning. However, slight discrepancies between training and validation metrics suggest potential overfitting as the model learns finer details. Future efforts could focus on:

- Adding regularization techniques, such as dropout.
- Increasing the dataset size to generalize better.
- Exploring alternative architectures or hyperparameters for optimization.

Chapter 8

Conclusion and Future Works

8.1 Summary

This study demonstrated the effectiveness of Convolutional Neural Networks (CNNs) for signature verification. Specifically, the Inception-v1 architecture outperformed traditional methods by achieving higher accuracy and lower error rates in distinguishing between genuine and forged signatures. The results show that CNNs can automatically learn complex features from raw data, eliminating the need for manual feature extraction. This capability makes CNNs a powerful tool for signature verification, offering a more robust solution compared to traditional approaches that often rely on handcrafted features.

8.2 Future Directions

The future scope of CNN-based signature verification models includes the following key areas of improvement and practical application:

1. Expanding the Dataset

- Gather a larger, more diverse dataset of signatures from real-world scenarios to improve the training process.
- Include signatures from different age groups, cultural backgrounds, and environmental conditions to ensure the model performs well in varied contexts.

- Incorporate dynamically generated forged signatures to better prepare the system for real-world forgery attempts.

2. Developing a Web Application for Signature Verification

- Create an intuitive web-based platform to make the signature verification model easily accessible.
- The application will allow users to upload signatures for real-time verification.

3. Exploring Advanced Architectures and Algorithms

- In the next semester, we plan to experiment with well-known architectures such as AlexNet, VGG, ResNet, and Inception for feature extraction and image classification tasks.
- Additionally, traditional machine learning algorithms like Support Vector Machines (SVM) and Random Forest will be evaluated for their performance in signature verification.

4. Real-World Deployment and Testing

- Future efforts will focus on testing the system in real-world applications such as banking, insurance, legal document verification, and e-governance systems.
- Robust feedback mechanisms will be implemented to gather user insights and continuously improve the model's performance.

Bibliography

1. Berkay Yilmaz, M. and Ozturk, K., 2018. Hybrid user-independent and user-dependent offline signature verification with a two-channel CNN. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 526-534).
2. Yapici, M.M., Tekerek, A. and Topaloglu, N., 2018, December. Convolutional neural network based offline signature verification application. In 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT) (pp. 30-34). IEEE.