Deep Fake Analysis using LBP and Sobel based Deep Learning Model

Dr. P. Horsley Solomon, Head, Department of Electronics with Artificial Intelligence, SRM Arts and Science College, Kattankulathur, Chengalpattu District.

Abstract

Deep fake technology poses significant threats by enabling highly realistic synthetic facial videos and images. In this paper, a lightweight yet effective Convolutional Neural Network (CNN) model that combines Local Binary Pattern (LBP) texture features and Sobel gradient edge information to detect deepfakes is presented. The proposed approach leverages handcrafted features to enhance discrimination between real and fake facial imagery. Trained and evaluated on a subset of the RVF10K dataset, the proposed model achieves promising accuracy, making it suitable for real-time or resource-constrained applications.

Keywords: deepfake detection, LBP, Sobel gradient, CNN, image classification.

I. Introduction

Deepfakes, synthetic media generated using deep learning techniques like GANs, are a growing concern in digital security. Their increasing realism challenges both social trust and content authentication. Most existing models rely heavily on deep networks and vast computational resources. This project explores a hybrid approach where classical image processing features—Local Binary Patterns and Sobel filters—are combined and passed into a CNN for deepfake detection, aiming for interpretability and efficiency.

II. Ease of Use

The proposed system is easy to implement and run on modest hardware (e.g., Google Colab or laptops with basic GPUs). The input images are resized and processed into a compact 3-channel format, allowing for quick training and prediction. No pre-trained models or external datasets are required other than a labeled deepfake dataset (e.g., RVF10K). The model can be trained and tested using fewer than 5000 images with good accuracy

III. Related Work

Several deepfake detection models rely on deep CNNs, Vision Transformers, and frequency-domain analysis. While accurate, many of these approaches are resource-heavy and complex. Others focus on face warping artifacts or blinking patterns, but such heuristics fail for improved fakes.

Handcrafted features like LBP and Sobel are simple yet effective in capturing subtle texture and gradient inconsistencies. Previous works have used LBP for face recognition and forgery detection. In this work, LBP is combined with Sobel gradients in a novel 3-channel format to train a CNN for

deepfake detection.

IV. Methodology

Our proposed deepfake detection method involves three key stages: (A) Feature Extraction, (B) Preprocessing, and (C) CNN Model Design and Training. The goal is to convert each image into a 3-channel representation that encodes both texture and edge information, and use this representation to train a CNN that can distinguish between real and fake faces.

A. Feature Extraction

To detect deepfakes, visual clues typically left by generative algorithms are focussed. These include unnatural textures, over-smoothed areas, and edge distortions. Instead of relying purely on raw RGB pixels, handcrafted features that can highlight these anomalies are computed:

1. Local Binary Pattern (LBP):

2. LBP is a texture descriptor that compares each pixel to its surrounding neighbors. It encodes the result as a binary number and captures fine-grained texture irregularities.

- The uniform LBP method with 8 neighbors in a circular pattern (radius = 1) is used.
- The resulting LBP map is normalized to the range [0, 1].

3. Sobel Gradient (X and Y):

4. Sobel operators detect gradients along the x and y directions. They highlight transitions in intensity, which helps capture sharp edges, facial outlines, and potential warping artifacts introduced by generative models.

- Sobel X captures horizontal gradients.
- Sobel Y captures vertical gradients.
- Both are scaled and stacked as separate channels.

Together, these features provide a rich 3-channel input containing:

- Channel 1: LBP map
- Channel 2: Sobel X gradient
- Channel 3: Sobel Y gradient

This fusion allows the CNN to learn both local texture patterns and macro edge structures.

B. Image Processing

To ensure consistent input and efficient training, the following preprocessing steps are applied to all images:

- Resizing: Images are resized to 64×64 pixels to reduce computational load.
- Grayscale Conversion: Required for LBP and Sobel computations.
- Normalization:
  - LBP is normalized using lbp / lbp.max()
  - Sobel gradients are computed as float values and rescaled by dividing by 255
- Stacking: The final shape of each image becomes (64, 64, 3).

These steps are applied to both training and test images.

C. CNN Model Design

A simple CNN model capable of processing the 3-channel handcrafted input is designed. The architecture includes the following layers:

1. Convolutional Blocks:

- Three convolutional layers with filters (32, 64, 128) and kernel size (3×3)
- ReLU activation
- Each followed by:
  - Batch Normalization for stable training
  - Max Pooling (2×2) to downsample features

2. Fully Connected Layers:

- A Flatten layer to convert 2D feature maps to a 1D vector
- Dense layer with 128 units and ReLU
- Dropout (rate = 0.5) to prevent overfitting
- Final output layer with a sigmoid activation (binary classification)

Input shape: (64, 64, 3)

Output: Probability that the input image is a fake face.

D. Data Augmentation

To improve generalization, ImageDataGenerator is used with:

- Rotation: ±10 degrees
- Width/Height Shift: ±10%
- Zoom: ±10%

This helps the model learn better under slight transformations and prevents overfitting.

Training Setup

- Optimizer: Adam with learning rate = 0.0005
- Loss Function: Binary Cross-Entropy
- Batch Size: 32
- Epochs: 20 (early stopping if validation loss stops improving)
- Validation: A separate test set is used to monitor performance.

Early stopping monitors validation loss and restores the best weights after training ends.

E. Dataset

- RVF10K dataset, a curated face dataset is used.
- Real faces (from publicly available datasets)
- Deepfake faces (generated using known synthesis techniques)
- The dataset is split into:
- Training set: Used for model fitting
- Validation/Test set: Used to evaluate generalization

- Total images used:
- ~6000 for training
- ~4000 for validation/testing

V. Local Binary Pattern (LBP) for Texture Analysis

- Deepfake artifacts (especially from face synthesis) often affect the skin texture, such as blurring, unnatural transitions, or loss of micro-details.
- LBP can effectively highlight these anomalies by emphasizing texture differences between real and manipulated regions.
- It is rotation-invariant and computationally efficient, making it suitable for preprocessing large datasets.

VI. Sobel Gradient for Edge Detection

- Deepfake algorithms often fail to maintain consistent edge information — especially around eyes, lips, and hairlines — due to blending errors or resolution mismatches.
- The Sobel filter captures these structural inconsistencies and sharp transitions that might not be detected by texture descriptors alone.
- It enriches the input features with directional intensity changes, improving the model's understanding of image structure.

VII. Model used

The model employed in this work is a Convolutional Neural Network (CNN), tailored to handle both LBP (Local Binary Pattern) and Sobel gradient preprocessed images. This hybrid input approach ensures that both texture and edge-level features are effectively captured.

A. Strong Feature Extraction Capability

CNNs are highly efficient in capturing spatial hierarchies in images through convolutional layers. When combined with LBP and Sobel features, CNNs can detect fine-grained local textures (from LBP) and edge orientation patterns (from Sobel)

B.Complementary Nature of Inputs

- LBP helps highlight micro-textures that often differ subtly between real and fake faces.
- Sobel filters emphasize gradients, which reveal unnatural transitions or blending artifacts typical in deepfake regions.
- By feeding these enhanced inputs into the CNN, the model gains a deeper understanding of both structural and textural inconsistencies.

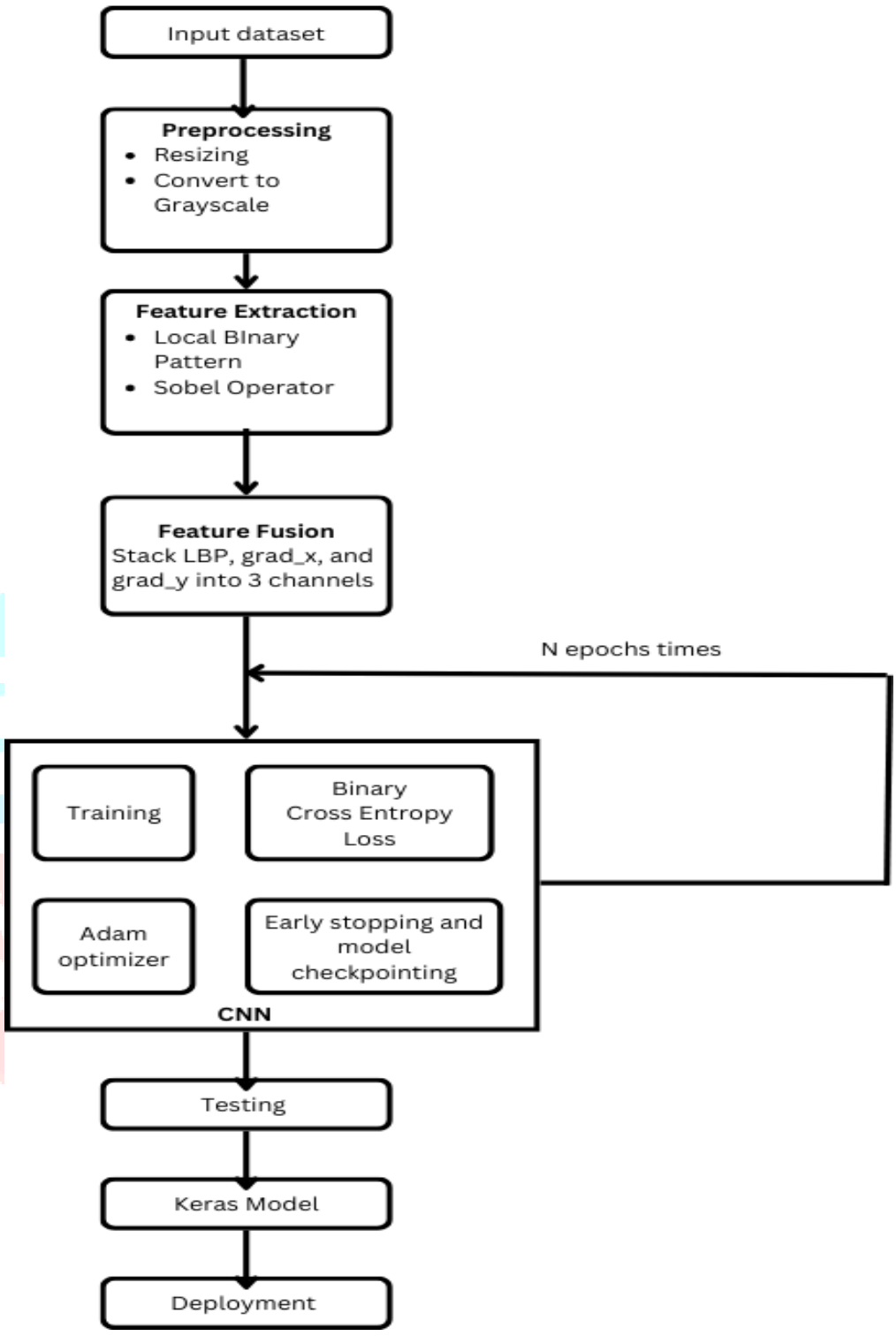
B. Adaptability

- CNNs can learn and adapt filters during training, making them well-suited for generalization across various deepfake manipulation techniques.

C. End-to-End Learning

- The CNN model allows for an end-to-end training pipeline where feature extraction and classification are optimized together, ensuring better performance than handcrafted features followed by traditional classifiers.

VIII. Flowchart



Input Dataset:

& The process begins with the input dataset, which typically consists of images or video frames.

Preprocessing:

& This step includes:

& Resizing the images to a uniform dimension suitable for model input.

& Grayscale Conversion, which simplifies the image data by reducing color channels, helping to focus on structural patterns.

Feature Extraction:

& In this phase, two types of features are extracted:

& Local Binary Pattern (LBP): Captures texture information by encoding the relationship between pixels.

& Sobel Operator: Computes the gradients (grad_x and grad_y) to highlight edges and intensity variations in the image.

Feature Fusion:

& The extracted features (LBP, grad_x, and grad_y) are stacked into three channels, simulating a 3-channel image input. This fusion enhances the feature representation by combining texture and edge information.

CNN Model Training:

& The fused features are fed into a CNN which is trained using:

& Binary Cross Entropy Loss: Used for binary classification tasks to measure the discrepancy between predicted and actual labels.

& Adam Optimizer: An efficient stochastic optimizer that adjusts the learning rate during training.

& Early Stopping and Model Checkpointing: These techniques prevent overfitting and ensure the best model is retained by stopping training once performance no longer improves.

Epoch Loop:

& The training process is repeated over N epochs, where the model iteratively updates weights to minimize the loss function.

Testing:

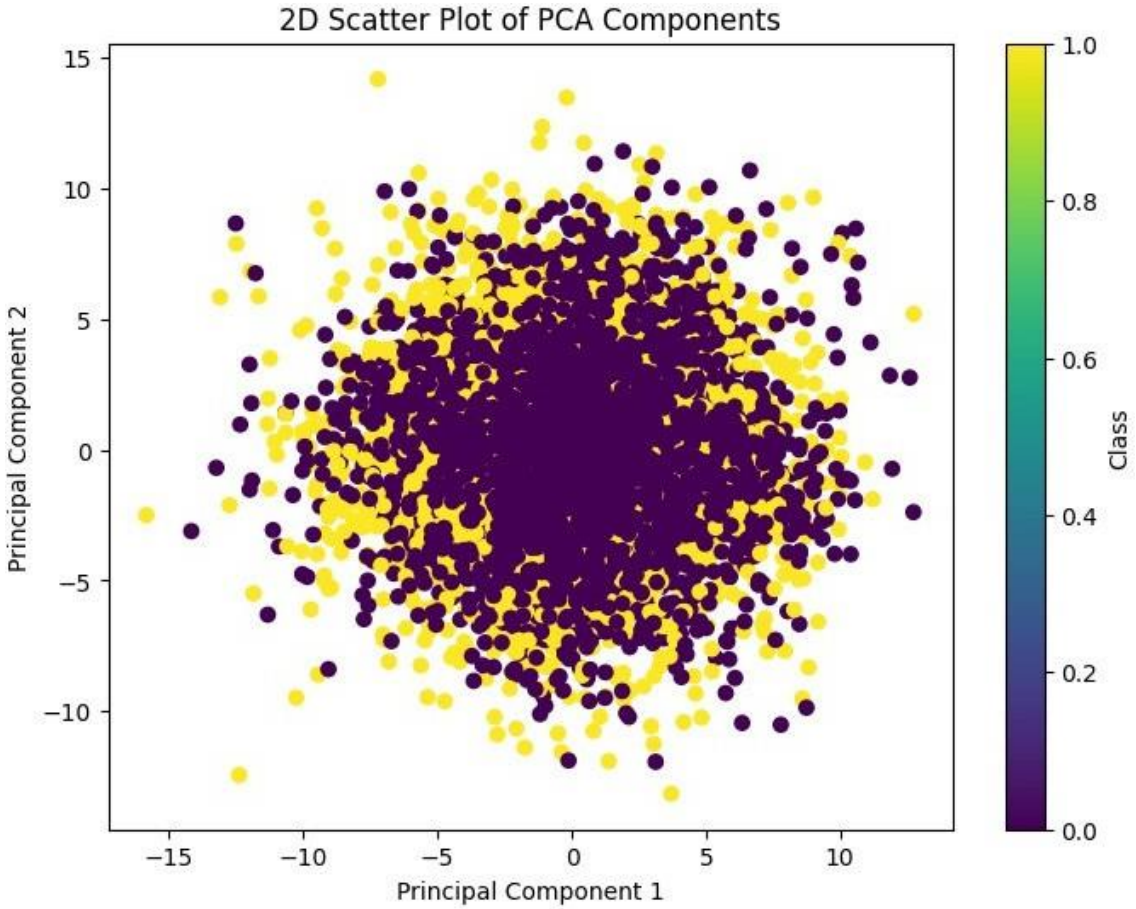
& After training, the model is evaluated on a separate test dataset to assess its generalization capability.

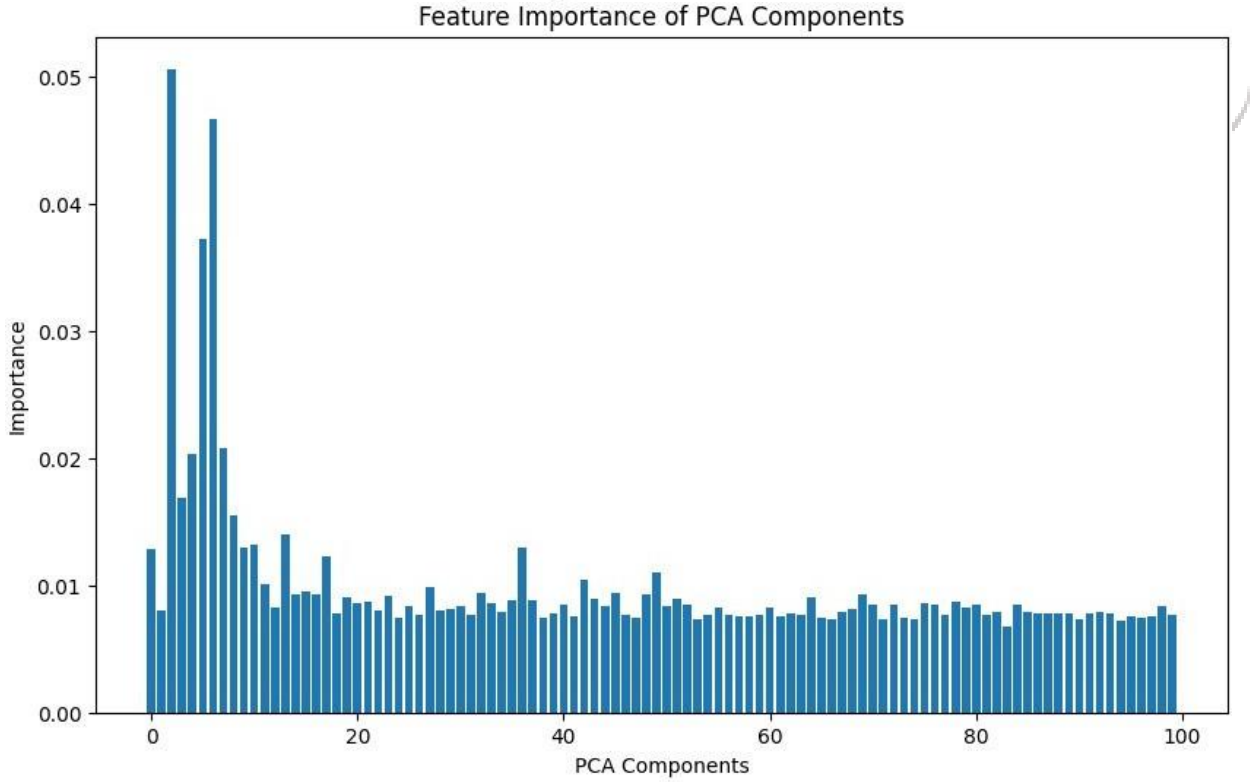
Keras Model & Deployment:

& The final trained model is exported using the Keras framework and deployed in the target environment for real-world inference and application.

IX. PCA-Based Visualization of Feature Distribution

X. Importance of PCA Histogram in Deepfake Detection


2D Scatter Plot of PCA Components
Principal Component 2
Principal Component 1
Class
15
10
5
0
−5
−10
−15
−10
−5
0
5
10
1.0
0.8
0.6
0.4
0.2
0.0


Feature Importance of PCA Components
Importance
PCA Components
0.05
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
100

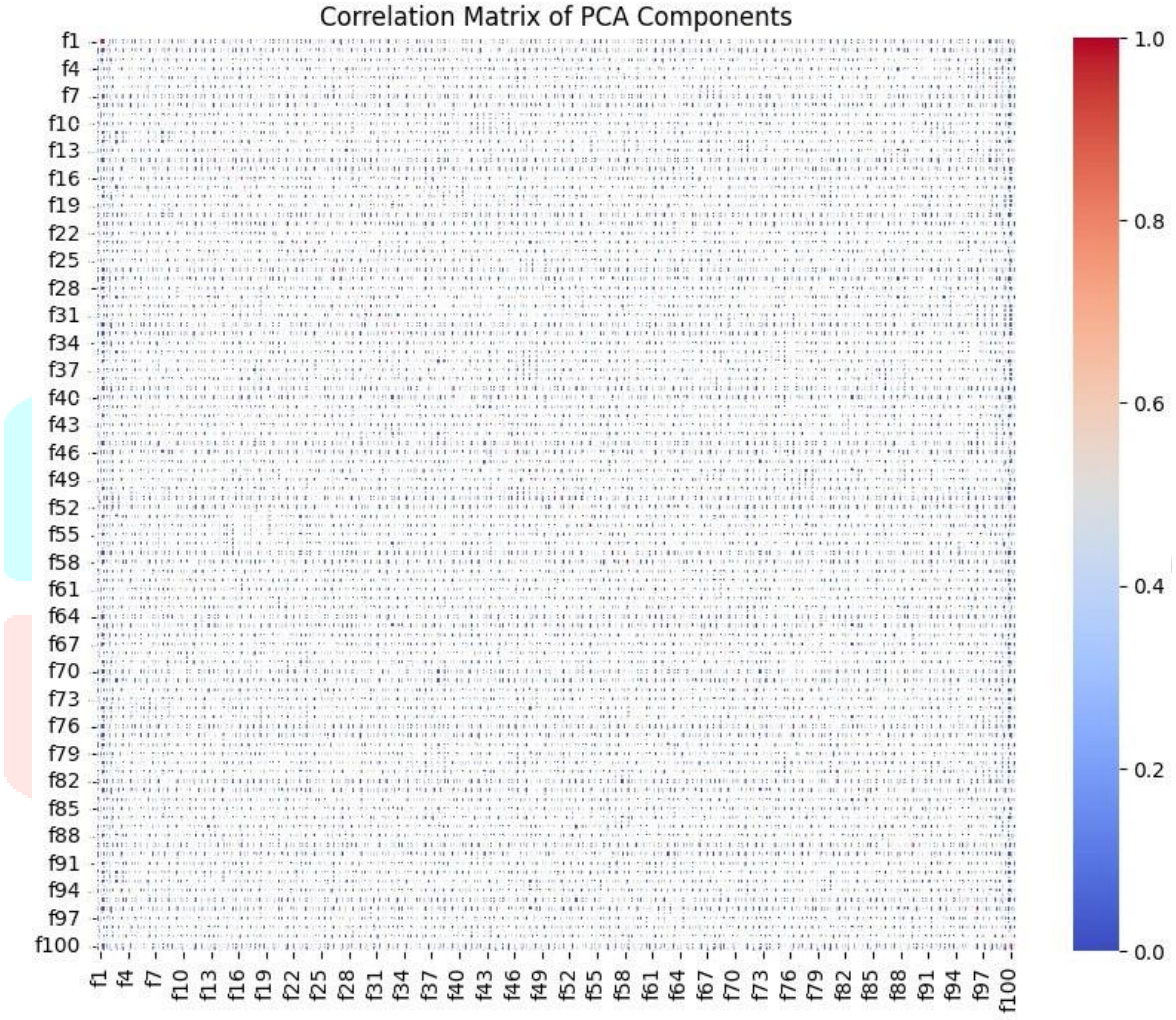
& Dimensionality Reduction with PCA:

& PCA was applied to reduce high-dimensional features to two principal components, enabling visualization of data distribution, where each point represents a sample colored by class label (e.g., real vs. fake).

& Observation and Model Implication:

& The scatter plot shows overlapping classes, indicating limited linear separability. This supports the use of a nonlinear classifier like CNN to effectively capture complex patterns in the data.

XI. PCA-Based Pair Plot Visualization of Feature Distribution



The pairplot helps visualize feature relationships and class separability. It highlights overlapping or clustered patterns between real and fake images, guiding feature selection and model design for better classification. A PCA histogram displays the variance explained by each principal component. It identifies how much information is retained after dimensionality reduction, helping to choose the optimal number of components and assess data complexity for classification tasks.

XII. Results and Discussion

To evaluate the effectiveness of the proposed deepfake detection framework, extensive experiments using the RVF10K dataset were conducted. The performance of the model was assessed using standard evaluation metrics, including accuracy, precision, recall, F1-score, and Area under the ROC Curve (AUC)

A. Accuracy

& Definition: Accuracy is the proportion of total correct predictions to the total number of inputs. It gives an overall performance measure of a classification model.

- Formula: Accuracy=(True Positives+ True Negatives) / Total samples
- Where:
- True Positives (TP): Correctly predicted positive samples.
- True Negatives (TN): Correctly predicted negative samples.
- Total Samples: The total number of samples in the dataset (TP + TN + False Positives (FP) + False Negatives (FN)).

B. Precision

- Definition: Precision is the proportion of true positives among the predicted positives. It shows how many of the predicted positive instances are actually positive.
- Formula:
- Precision= True positives / (True positives + False positives)
- Where:
  - True Positives (TP): Correctly predicted positive samples.
  - False Positives (FP): Incorrectly predicted positive samples (i.e., negative samples that were incorrectly predicted as positive).

C. Recall (Sensitivity or True Positive Rate)

- Definition: Recall is the proportion of true positives among the actual positives. It shows how well the model identifies the actual positive instances.
- Formula:
- Recall= True positives / (True positives + false Negatives)
- Where:
- True Positives (TP): Correctly predicted positive samples.
- False Negatives (FN): Incorrectly predicted negative samples (i.e., positive samples that were predicted as negative).

F1-Score

- Definition: F1-Score is the harmonic mean of precision and recall, representing a balance between them. It is a good metric when you need a balance between precision and recall.
- Formula:
- F1-Score= 2 *((precision* recall) / (precision + recall))
- Where:
- Precision is the ratio of true positives to predicted positives.
- Recall is the ratio of true positives to actual positives.

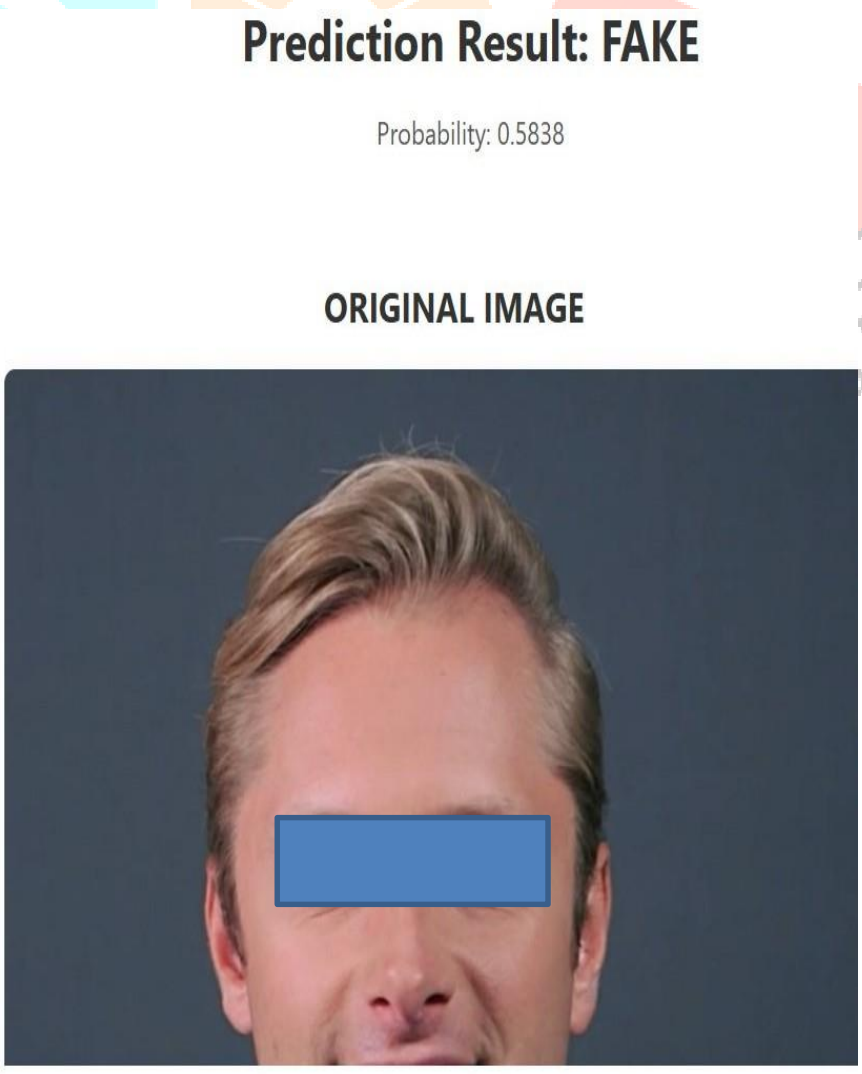
```
95/95 ━━━━━━━━━━━━━━━━━━━━ 24s 240ms/step
Classification Report:
              precision    recall  f1-score   support

         0.0     0.7634    0.6733    0.7156      1500
         1.0     0.7095    0.7927    0.7488      1510

    accuracy                         0.7332      3010
   macro avg     0.7365    0.7330    0.7322      3010
weighted avg     0.7364    0.7332    0.7322      3010

Confusion Matrix:
 [[1010  490]
 [ 313 1197]]
```

XIII. Receiver Operating Characteristic (ROC) curve



X-axis (False Positive Rate): This shows the proportion of

actual negatives that were incorrectly classified as positives. Y-axis (True Positive Rate): This represents the proportion of actual positives that were correctly classified (also known as sensitivity or recall).

Orange Line: This is the ROC curve for your model.

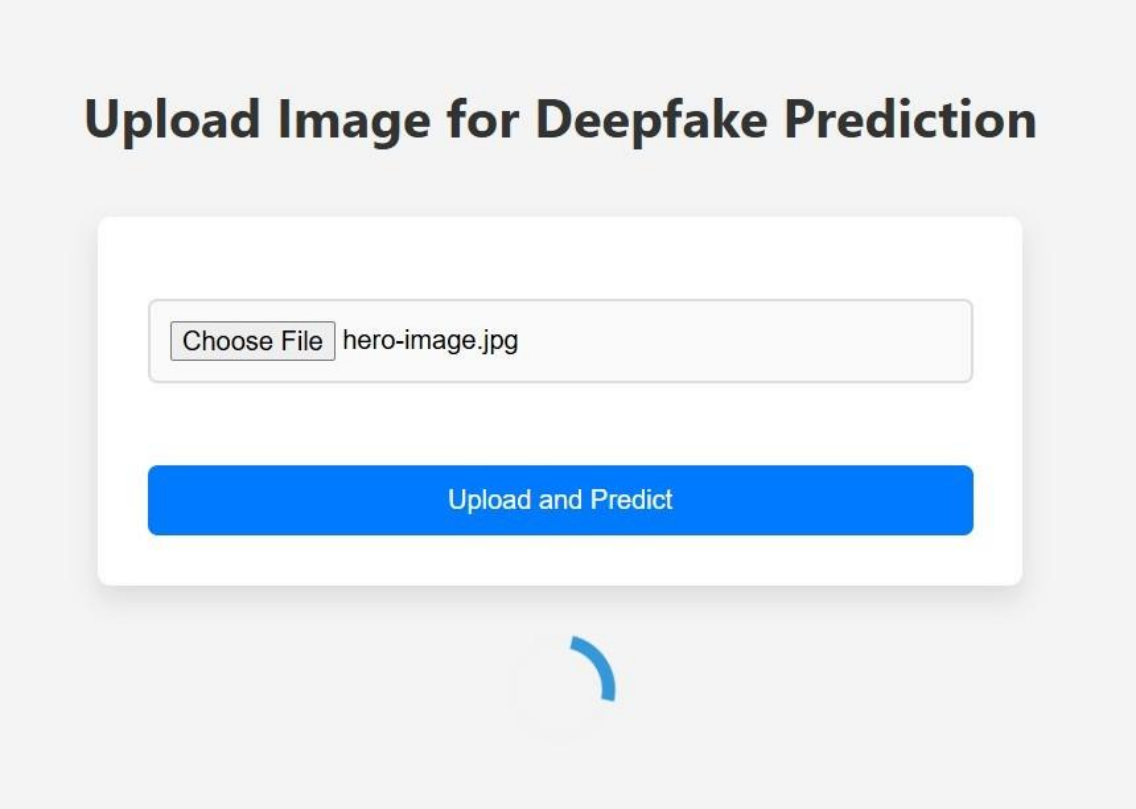
Blue Dashed Line: This is the baseline for a random classifier (i.e., no skill); it follows a 45-degree line from (0,0) to (1,1).

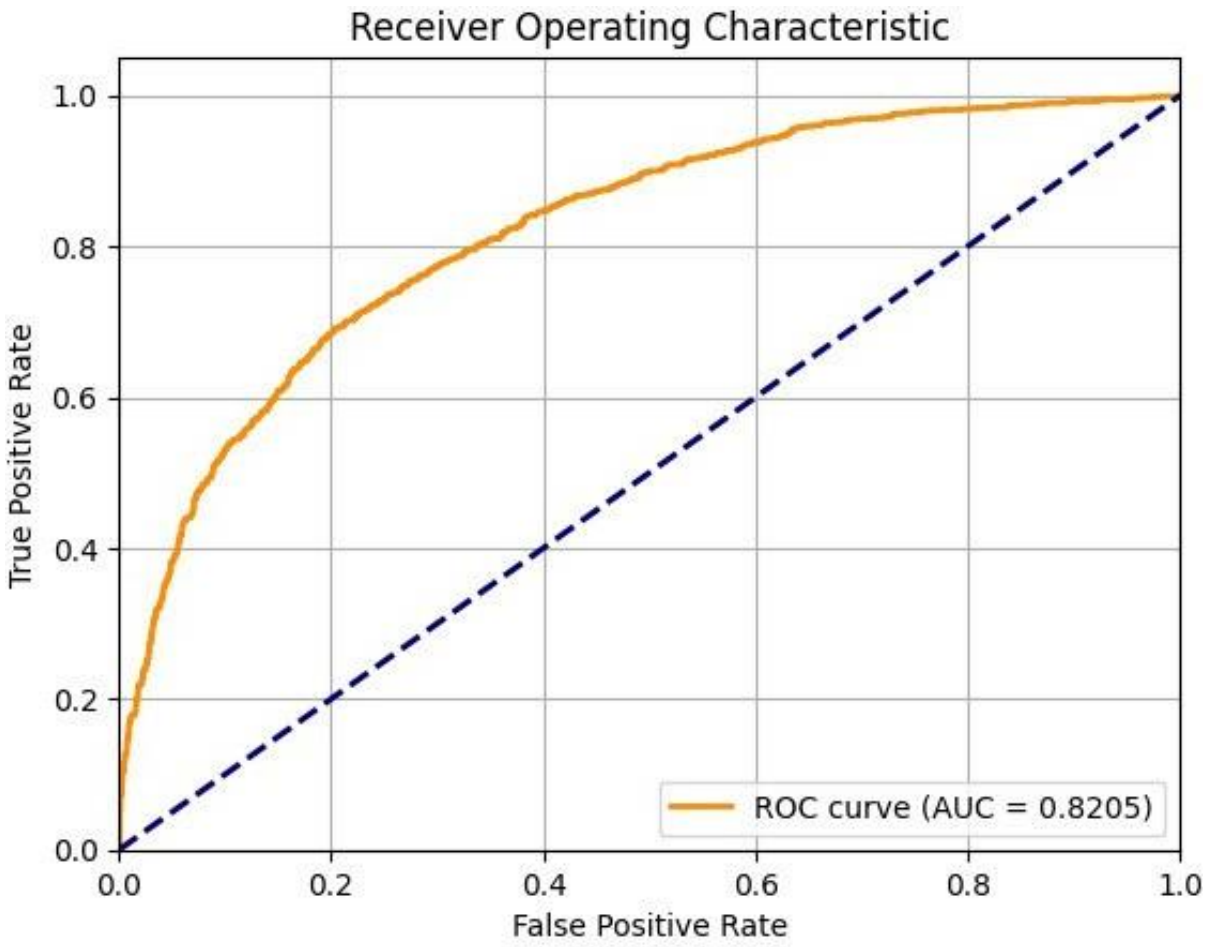
AUC (Area Under the Curve): This is a single scalar value summarizing the ROC curve. In this case:

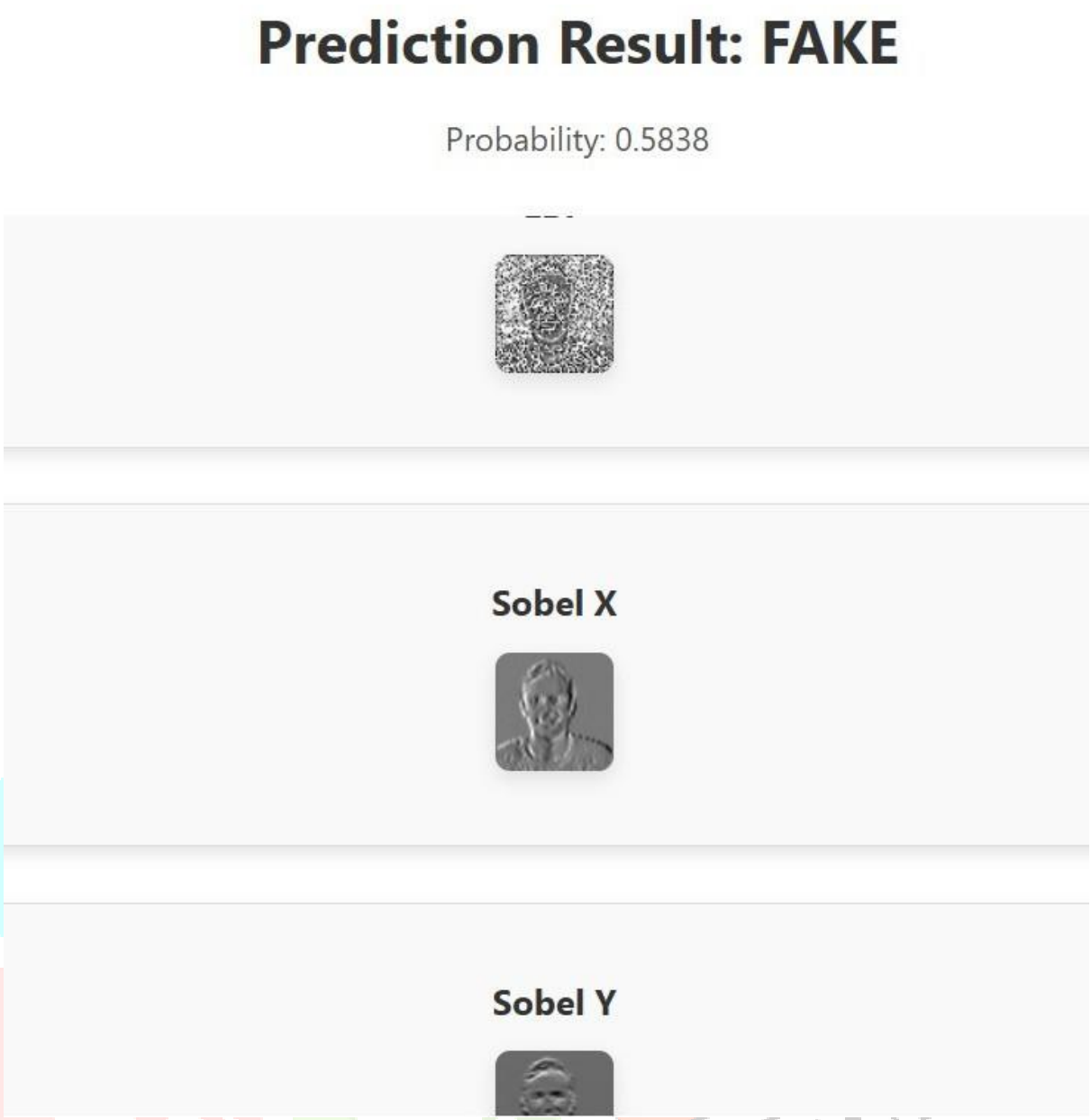
AUC = 0.8205, which indicates good performance.

A perfect classifier would have an AUC of 1.0, and a purely random classifier would have an AUC of 0.5.

XIV. Output




Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.8205)



References

[1] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "FaceForensics++: Learning to detect manipulated facial images," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 1–11, doi: 10.1109/ICCV.2019.00009.

[2] Y. Li and S. Lyu, "Exposing DeepFake videos by detecting face warping artifacts," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, 2019, pp. 46–52.

[3] L. Li, J. Bao, T. Zhang, H. Yang, D. Chen, F. Wen, and B. Guo, "Face X-Ray for more general face forgery detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 5001–5010.

[4] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 971–987, Jul. 2002, doi: 10.1109/TPAMI.2002.1017623.

[5] A. Malik, M. Kuribayashi, S. M. Abdullahi, and A. N. Khan, "DeepFake detection for human face images and videos: A survey," *IEEE Access*, vol. 10, pp. 18757–18775, 2022, doi: 10.1109/ACCESS.2022.3151186.

[6] M. N. Nobi, B. Murali, M. S. Rana, and A. H. Sung, "Deepfake detection: A systematic literature review," *IEEE Access*, vol. 10, pp. 25494–25513, 2022, doi: 10.1109/ACCESS.2022.3154404.

[7] L. Li, J. Bao, H. Yang, D. Chen, and F. Wen, "Advancing high fidelity identity swapping for forgery detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 5074–5083.

[8] L. Li, J. Bao, T. Zhang, H. Yang, D. Chen, F. Wen, and B. Guo, "Face X-ray for more general face forgery detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 5001–5010.

[9] R. Tolosana, R. Vera-Rodriguez, J. Fierrez, A. Morales and J. Ortega-Garcia, "Deepfakes and Beyond: A Survey of Face Manipulation and Fake Detection," Information Fusion, vol. 64, pp. 131–148, 2020. doi: 10.1016/j.inffus.2020.07.007

[10] Y. Li, M.-C. Chang, and S. Lyu, "In Ictu Oculi: Exposing AI Created Fake Videos by Detecting Eye Blinking," IEEE International Workshop on Information Forensics and Security (WIFS), 2018