BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution
(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)
"Jnana Gangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur,
Ballari-583 104 (Karnataka) (India)
Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE-ARTIFICIAL INTELLIGENCE

NEURAL NETWORKS AND DEEP LEARNING (22CA71)
MINI PROJECT ON

"DEEPFAKE DETECTION"

Submitted By

ASHISH KUMAR JHA 3BR22CA006

Under the Guidance of

Mr. AZHAR BIAG ASST.PROF.

Mr. VIJAY KUMAR TEACHING ASST.

Mr. PAVAN KUMAR ASST.PROF.

Visvesvaraya Technological University

Belagavi, Karnataka 2025-2026

BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution

(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)

"Jnana Gangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballari-583 104 (Karnataka) (India)

Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE- (ARTIFICIAL INTELLIGENCE)

CERTIFICATE

This is to certify that the mini-project for **Neural Networks and Deep Learning Lab** entitled **"DEEPFAKE DETECTION"** has been successfully presented by **ASHISH KUMAR JHA** bearing USN **3BR22CA006** of VII semester B.E for the partial fulfilment of the requirements for the award of **Bachelor Degree in CSE-(Artificial Intelligence)** of the **BALLARI INSTITUTE OF TECHNOLOGY& MANAGEMENT** during the academic year 2025-2026.

Signature of guides

MR. AZHAR BAIG

MR. VIJAY KUMAR

MR. PAWAN KUMAR

Signature of HOD

DR. YERESIME SURESH

ACKNOWLEDGEMENT

The satisfactions that accompany the successful completion of our mini project on "**DEEPFAKE DETECTION**" would be incomplete without the mention of people who made it possible, whose noble gesture, affection, guidance, encouragement and support crowned my efforts with success. It is our privilege to express our gratitude and respect to all those who inspired us in the completion of our mini-project.

We are extremely grateful to our Guides **Mr. Azhar Baig, Mr. Vijay Kumar and Mr. Pavan Kumar** for their noble gesture, support, co-ordination and valuable suggestions given in completing the mini-project. We also thank **DR. YERESIME SURESH**, H.O.D. Department of CSE-AI, for his co-ordination and valuable suggestions given in completing the mini-project.

ASHISH KUMAR JHA 3BR22CA006

TABLE OF CONTENTS

ABSTRACT

Deepfake technology has rapidly advanced due to progress in generative adversarial networks (GANs), diffusion models, and other deep learning techniques, enabling highly realistic manipulation of audio-visual content. While these synthetic media offer creative opportunities, they also pose significant risks, including misinformation, identity abuse, and threats to digital trust. Consequently, developing robust and generalizable deepfake-detection methods has become a critical research area. This work reviews and analyzes state-of-the-art detection approaches, focusing on spatial, temporal, physiological, and frequency-domain cues. We highlight emerging trends such as transformer-based architectures, multimodal fusion, self-supervised learning, and adversarial robustness. Additionally, we examine the challenges of dataset bias, cross-domain generalization, real-world degradation, and the evolving arms race between deepfake generation and detection. Our findings emphasize the need for explainable, scalable, and resilient detection systems capable of operating in unconstrained environments. The paper concludes with open research problems and future directions for building trustworthy AI-driven media authentication.

CHAPTER 1

INTRODUCTION

Deepfake Technology has rapidly advanced in recent years, enabling highly realistic manipulation of audio, images, and videos. While this technology offers creative possibilities, it also poses serious threats, including misinformation, identity fraud, privacy violations, and political manipulation. As deepfakes become increasingly difficult for humans to recognize, the need for reliable automated detection systems has become critical. With the growth of Artificial Intelligence, Neural Networks and Deep Learning models have proven highly effective in analyzing visual and audio patterns. Convolutional Neural Networks (CNNs), along with advanced architectures such as Recurrent Neural Networks (RNNs) and 3D CNNs, can automatically learn subtle facial inconsistencies, unnatural movements, frame-level artifacts, and texture abnormalities commonly found in deepfake media. These capabilities make deep learning a powerful solution for deepfake detection. This project focuses on building a deep learning-based **Deepfake Detection System** that classifies video frames or images as Real or Fake. The system includes key stages such as face extraction, preprocessing, feature learning using CNN-based architectures, and classification. Performance is evaluated using standard metrics like accuracy, precision, recall, and F1-score to ensure reliable detection. By leveraging AI, the system aims to provide fast, accurate, and scalable deepfake identification. Such automated tools can support social media platforms, legal authorities, cybersecurity teams, and digital content verification systems. Overall, this project demonstrates how deep learning can enhance digital media security, reduce risks associated with manipulated content, and contribute to a safer and more trustworthy digital ecosystem.

1.1 Problem Statement

To design and implement a deep learning–based system using CNN (and CNN-based) models to detect deepfake images and video frames and classify them as **Real** or **Fake**, improving detection speed, robustness, and explainability for real-world content-verification workflows.

1.2 Scope of the Project

- Automating deepfake detection using image and video data.
- Enhancing the reliability and consistency of digital content verification.
- Assisting media platforms, cybersecurity teams, and authorities in identifying manipulated content.
- Demonstrating the application of CNNs and advanced deep learning techniques (e.g., transfer learning, frame-level analysis) in detecting facial and visual artifacts.
- Providing a foundation for future integration into real-time detection systems for social media, legal investigations, and digital forensics.

1.3 Objectives

- To collect and preprocess deepfake datasets, including real and manipulated images or video frames, for model training.
- To build and train a CNN-based or CNN-enhanced model for deepfake classification.
- To evaluate model performance using accuracy, precision, recall, F1-score, and other relevant metrics.
- To provide prediction functionality for new, unseen images or video frames to determine whether they are real or fake.
- To demonstrate the effectiveness of deep learning in automated detection of manipulated digital content.

CHAPTER 2

LITERATURE SURVEY

[1]. The study titled **"FaceForensics++: Learning to Detect Manipulated Facial Images"** by Rossler et al. (2019) introduced a large-scale dataset for training and evaluating deepfake detection models. Using CNN-based architectures such as XceptionNet, the research demonstrated that deep learning can effectively identify subtle facial inconsistencies in manipulated videos. The results highlight that CNNs enable high-accuracy automated detection, offering strong potential for combating misinformation and digital fraud.

[2]. In the research **"DeepFake Detection Using Convolutional Neural Networks"** by Afchar et al. (2018), the authors proposed the MesoNet architecture, designed to detect deepfake artifacts through lightweight CNN layers. The findings showed that MesoNet achieves strong performance with low computational cost, making it suitable for real-time applications. The study emphasizes that targeted deep learning architectures can successfully capture manipulation traces such as texture abnormalities and blending artifacts.

[3]. A study by Dang et al. titled **"On the Detection of Digital Face Manipulation"** evaluated the effectiveness of deep learning models including Capsule Networks, ResNet, and EfficientNet for classifying real and fake facial images. The analysis revealed that transfer learning improved accuracy significantly, while data augmentation enhanced robustness against compression and noise. The research also showed that CNN-based models generalize better when trained across multiple manipulation techniques.

[4]. The research paper **"Deep Learning for Deepfake Detection: A Survey"** by Tolosana et al. (2020) reviewed the evolution of AI-based methods for detecting manipulated audiovisual content. The authors concluded that CNNs and hybrid architectures outperform traditional forensic techniques in detecting inconsistencies in facial expressions, eye blinking patterns, and motion dynamics. The study provides strong evidence that deep learning enables scalable, reliable detection tools for digital media forensics.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware Requirements

- Processor: Intel i5/i7 or equivalent
- RAM: Minimum 8 GB (16 GB recommended for faster training)
- Storage: 10 GB free space
- GPU: Optional, but NVIDIA GPU recommended for faster training

3.2 Software Requirements

- Operating System: Windows / Linux / macOS
- Programming Language: Python 3.x
- Libraries:
  - TensorFlow / Keras
  - NumPy
  - Matplotlib
  - OpenCV
  - Scikit-learn
- Jupyter Notebook / Google Colab

3.3 Functional Requirements

- Ability to load and preprocess images or video frames for deepfake analysis.
- A CNN-based model to classify content as **Real** or **Fake**.
- Performance evaluation and reporting using relevant metrics.
- User interface/input functionality to upload new images or video frames and receive deepfake prediction results.

.

3.4 Non-Functional Requirements

- **Accuracy:** Model should provide high classification accuracy.
- **Efficiency:** Predictions should be fast and computationally feasible.
- **Reliability:** Consistent performance across various image types.
- **Usability:** Easy-to-use interface for uploading and analyzing images.
- **Scalability:** Should support larger datasets in future adaptations.

CHAPTER 4

DESCRIPTION OF MODULES

The Deepfake Detection System is divided into several interconnected modules. Each module plays a specific role in ensuring that the system operates efficiently—from dataset acquisition and preprocessing to model training, evaluation, and final prediction. These modules work together to automate the detection of real and synthetic facial images using deep learning. The detailed description of each module is provided below.

4.1 Dataset Acquisition and Loading Module

This module is responsible for obtaining and loading the deepfake detection dataset used for training and evaluation. In the implemented system, two publicly available Kaggle datasets are utilized:

- **FFHQ Face Dataset** – contains real human face images.
- **ThisPersonDoesNotExist Dataset** – contains AI-generated synthetic (fake) face images.
- These datasets together form the Real vs Fake classification dataset required for training the CNN model.

Key functions:

- **Access and load datasets** from the `/kaggle/input` directory.
- **Count and verify image availability**, ensuring both real and fake categories contain adequate data.
- **Organize and load image paths** for further processing.
- **Extract and convert images into arrays**, storing them in suitable NumPy structures (`X.npy`, `Y.npy`).

- **Label encoding** such that:

- Real images → label **0**
- Fake images → label **1**

4.2 Image Preprocessing Module

The preprocessing module transforms raw face images into a standardized format suitable for deep learning–based deepfake classification. Since images from both datasets vary in size and quality, preprocessing is essential for ensuring consistency and improving learning efficiency.

Operations performed:

- **Resizing images to 128×128×3**, matching the input shape required by the custom CNN model.
- **Normalization of pixel values to the range [0,1]** to ensure stable and faster gradient updates during training.
- **Conversion of loaded images into NumPy arrays** using `img_to_array()` for compatibility with TensorFlow models.

- **Shuffling and splitting the dataset** into:

1. 80% training data
2. 20% validation data

4.3 Model Construction Module (Custom CNN Architecture)

This module is responsible for building the deep learning architecture used to classify images as *Real* or *Fake*. Instead of using transfer learning, the system employs a **custom-built Convolutional Neural Network (CNN)** designed specifically for deepfake detection. This allows the model to learn manipulation artifacts and texture patterns directly from the dataset.

Components:

- **Convolutional Layers:** Extract spatial and texture features using multiple Conv2D blocks with ReLU activation.
- **Pooling Layers:** MaxPooling layers reduce image dimensions and prevent overfitting.
- **Dilated Convolutions:** Capture wider contextual information to detect subtle deepfake artifacts.
- **Global Feature Extraction:** GlobalAveragePooling and Flatten layers convert feature maps into 1D vectors.
- **Fully Connected Layers:** Dense layers (400, 512, 400) with ReLU activation for high-level classification.
- **Dropout Layers:** Reduce overfitting by randomly disabling neurons during training.
- **Output Layer:** Final Dense layer with 2 units and sigmoid activation for binary classification (Real vs Fake).
- **Optimizer & Loss:** Adam optimizer (1e-5) and binary crossentropy for stable training.

4.4 Model Training Module

This module manages the complete training process of the custom CNN model using the preprocessed Real and Fake image datasets.

Functions:

- Configure training parameters, including:

- Optimizer: Adam (learning rate = 1e-5)
- Loss function: Binary Crossentropy
- Metrics: Accuracy
- Batch size: 32
- Number of epochs: 100

- Train the model using:

- The training dataset (X_train, Y_train)
- The validation dataset (X_val, Y_val) for monitoring generalization and detecting overfitting

- Monitor model performance through:

- Loss curve (training vs validation)
- Accuracy curve (training vs validation)
- ModelCheckpoint callback to save the best-performing model automatically

4.5 Model Evaluation Module

This module is used to assess the performance of the trained model on unseen test data.

Evaluation Metrics:

- Accuracy
- Loss
- Confusion Matrix
- Precision, Recall, and F1-Score

This module helps determine how well the model generalizes to new images and reveals any overfitting or underfitting issues.

4.6 Prediction Module

This module enables users to upload a new image and obtain a classification on whether the face is Real or Fake.

Process:

- **Accept a user-uploaded image** (from system upload, Kaggle environment, or Colab).

- **Preprocess the image** by resizing it to **128×128×3** and normalizing pixel values to the range 0,10,10,1.
- **Pass the processed image through the trained deepfake detection model** (`deepfake.keras` or `deepfake.h5`).
- **Display the predicted output:**
  - **Real Face**
  - **Fake (AI-Generated) Face**
- **Show a confidence score** to ensure prediction transparency and user trust.
- This module provides the final functionality of the system, allowing real-time deepfake identification using the trained CNN model.

CHAPTER 5

IMPLEMENTATION

The implementation of the Deepfake Detection System involves a structured series of steps that combine software tools, deep learning techniques, image processing methods, and computational resources. This section explains how each part of the system is executed in practice—from setting up the environment to generating the final Real/Fake predictions.

5.1 Hardware Implementation

The system can be implemented on a local machine or cloud-based platform. For efficient model training, GPU support is highly recommended.

Hardware Components Used

- Processor: Intel Core i5/i7 or equivalent
- RAM: Minimum 8 GB (16 GB preferred for faster processing)
- Storage: At least 10 GB free space for dataset and model files
- GPU (Optional but recommended):
    - NVIDIA GPU such as Tesla T4, GTX 1650, or RTX series
    - Cloud services like Google Colab or Kaggle Notebooks provide free GPU access

Role of Hardware

- CPU: Handles preprocessing, file management, and general operations
- GPU: Accelerates matrix computations required for training the CNN
- Memory (RAM): Loads batches of images and supports model training
- Storage: Saves dataset, model weights, and training outputs

Using GPU-backed environments significantly reduces training time and improves model performance.

5.2 Software Implementation

The software component involves setting up the development environment, installing necessary libraries, and executing the machine learning pipeline.

Tools and Libraries Used

- **Python 3.x** – Programming language
- **TensorFlow / Keras** – For building and training CNN models
- **NumPy** – Numerical operations
- **OpenCV / PIL** – Image manipulation

- **Matplotlib** – Visualization
- **KaggleHub** – Automatic dataset download
- **Google Colab (optional)** – Cloud-based environment with free GPU

Software Workflow Overview

1. Install dependencies
2. Import required libraries
3. Download dataset using KaggleHub
4. Build preprocessing pipeline
5. Construct and train the CNN model
6. Evaluate using test dataset
7. Predict using new images

Each step is executed systematically within the Jupyter/Colab notebook environment.

CHAPTER 6

CODE IMPLEMENTATION

6.1 IMPLEMENTATION STEPS

Step 1: The implementation begins by importing essential libraries such as TensorFlow, Keras, NumPy, Pandas, Matplotlib, OpenCV, and tqdm. These libraries support deep learning operations, dataset handling, visualization, and image preprocessing.

Step 2: The Real (FFHQ) and Fake (ThisPersonDoesNotExist) face datasets are loaded directly from the Kaggle input directory. The paths for real and fake image folders are stored for processing.

Step 3: The system reads the filenames from both datasets, counts the total images, and prepares lists for image paths and labels to ensure proper dataset structuring.

Step 4: Image preprocessing is performed by resizing images to **128×128×3**, converting them to arrays, and normalizing pixel values to the range 0,10, 10,1. This ensures uniformity and stabilizes training.

Step 5: The real and fake image arrays are combined into a single dataset, with labels assigned as:

- **0 → Real**
- **1 → Fake**

The dataset is then split into **80% training** and **20% validation** using `train_test_split()`.

Step 6: A custom CNN model is constructed consisting of multiple Conv2D, MaxPooling2D, dilated convolution layers, GlobalAveragePooling, Dense, and Dropout layers. No transfer learning is used in this implementation.

Step 7: The final output layer uses 2 neurons with sigmoid activation to perform binary classification (Real vs Fake). The model is compiled with the **Adam optimizer** and **binary crossentropy** loss.

Step 8: Model training is performed using `model.fit()` on the training dataset, with validation monitored at each epoch. A `ModelCheckpoint` callback saves the best-performing model.

Step 9: Training performance is visualized using accuracy and loss curves for both training and validation sets. This helps identify overfitting or underfitting.

Step 10: After training, the best saved model is loaded and evaluated using accuracy, precision, recall, confusion matrix, and ROC curve to measure classification performance.

Step 11: For prediction, a new image is uploaded or selected from the dataset. The system preprocesses the image (resize → normalize → expand dimensions) to match the model input format.

Step 12: The trained model predicts whether the image is **Real or Fake**, and the confidence score is displayed to provide prediction transparency.

6.2 TRAINING CODE

```
import os
import numpy as np
import pandas as pd
import cv2
from tqdm.notebook import tqdm_notebook
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
img_to_array, load_img
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split


input_dir = '/kaggle/input'
output_dir = '/kaggle/working'
working_dir = '/kaggle/working'


real_img_dir = f"{input_dir}/ffhq-face-data-set/thumbnails128x128"
fake_img_dir = f"{input_dir}/person-face-dataset-
thispersondoesnotexist/thispersondoesnotexist.10k"

input_shape = (128,128,3)


X = []
Y = []

for img in tqdm_notebook(os.listdir(real_img_dir)[:3000]):
    X.append(img_to_array(load_img(f"{real_img_dir}/{img}",
target_size=input_shape)) / 255.0)
    Y.append(0)

for img in tqdm_notebook(os.listdir(fake_img_dir)[:3000]):
    X.append(img_to_array(load_img(f"{fake_img_dir}/{img}",
target_size=input_shape)) / 255.0)
    Y.append(1)


# normalization
X = np.array(X)
X = X.reshape(-1, 128, 128, 3)
Y = to_categorical(Y, 2)
# Y = np.array(Y)

# saving processed data arrays
os.chdir(output_dir)
np.save('X.npy',X)
```

```
np.save('Y.npy',Y)
   os.chdir(working_dir)
X = np.load('X.npy')
Y = np.load('Y.npy')

# train test split
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size = 0.2,
random_state=5)

# info
print(f"\ndata shape: {X.shape}; labels shape: {Y.shape}")
print(f"X_train shape: {X_train.shape}\nY_train shape: {Y_train.shape}")
print(f"X_val shape: {X_val.shape}\nY_val shape: {Y_val.shape}")
)


# Not using transfer learning
# made our own model!

inputs=Input(shape=input_shape)
x = Conv2D(64,5,padding='same')(inputs)
x = Activation(activation='relu')(x)
x = MaxPool2D(strides=(2,2))(x)
x = Conv2D(64,5,padding='same')(x)
x = Activation(activation='relu')(x)
x = MaxPool2D(strides=(2,2))(x)
x = Conv2D(64,5,padding='same')(x)
x = Activation(activation='relu')(x)
x = MaxPool2D(strides=(2,2))(x)

 #Decreasing Filters and MAxPool Layers
x = Conv2D(32,3,padding='same',dilation_rate=2)(x)
x = Activation(activation='relu')(x)
# x = BatchNormalization()(x)
x = MaxPool2D(strides=(2,2))(x)
x = Conv2D(16,3,padding='same',dilation_rate=2)(x)
x = Activation(activation='relu')(x)
x = MaxPool2D(strides=(2,2))(x)
# x = BatchNormalization()(x)
x = Conv2D(16,3,padding='same',dilation_rate=2)(x)
x = Activation(activation='relu')(x)
x = MaxPool2D(strides=(2,2))(x)

 #Dense Layers
x = GlobalAveragePooling2D()(x)
x = Flatten()(x)
x = Dense(400)(x)
x = Dropout(0.5)(x)
x = Activation(activation='relu')(x)
x = Dense(512)(x)
x = Dropout(0.5)(x)
x = Activation(activation='relu')(x)
x = Dense(400)(x)
x = Dropout(0.5)(x)
x = Activation(activation='relu')(x)
x = Dense(2)(x)
```

```
#Output
out = Activation(activation='sigmoid')(x)

# final model:
model = Model(inputs,out,name='BaseModel')
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.Adam(learning_rate=1e-5, beta_1=0.9,
beta_2=0.999),
              metrics=['accuracy'])

model.summary()
)



import matplotlib.pyplot as plt
import seaborn as sn


fig, axs = plt.subplots(1, 2, figsize=(10, 4))

axs[0].plot(train_logs.history['loss'], label='loss')
axs[0].plot(train_logs.history['val_loss'], label='val_loss')
axs[0].legend()
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Score')
axs[0].set_title('Loss Logs')

axs[1].plot(train_logs.history['accuracy'], label='accuracy',
linewidth=2)
axs[1].plot(train_logs.history['val_accuracy'], label='val_accuracy',
linewidth=2)
axs[1].legend()
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Score')
axs[1].set_title('Accuracy Logs')

plt.show()
  else:

    print("\nPrediction: NORMAL")


from sklearn.metrics import precision_score, recall_score,
accuracy_score, confusion_matrix
pred = best_model.predict(X_val).round()

predicted = [np.argmax(p) for p in pred]
actual = [np.argmax(p) for p in Y_val]

print(f"Accuracy Score: {accuracy_score(predicted,actual).round(5)}")
print(f"Recall Score: {recall_score(predicted,actual).round(5)}")
print(f"Precision Score:
{precision_score(predicted,actual).round(5)}\n")
```

```
def print_confusion_matrix(y_true, y_pred):
    cm=confusion_matrix(y_true,y_pred)
    cm[0][0],cm[1][1]=cm[1][1],cm[0][0]
    print('True positive = ', cm[0][0])
    print('False positive = ', cm[0][1])
    print('False negative = ', cm[1][0])
    print('True negative = ', cm[1][1])
    print('\n')

cm=pd.DataFrame(cm,index=['True_predicted','False_predicted'],columns=['
True_Actual','False_Actual'])

    plt.subplots(figsize=(4,4))
    sn.heatmap(cm,cmap='BuGn',annot=True, fmt='.0f',cbar=False)
    plt.show()

print_confusion_matrix(actual,predicted)


input_arr = img_to_array(load_img('/kaggle/input/ffhq-face-data-
set/thumbnails128x128/00001.png', target_size=(128,128,3))) / 255.0
input_arr = np.expand_dims(input_arr, axis=0)
print(input_arr.shape)

np.argmax(best_model.predict(input_arr).round()[0])
model.save('/kaggle/working/deepfake.h5')
saved_model = tf.keras.models.load_model('/kaggle/working/deepfake.h5')
```

CHAPTER 7

RESULTS

```
Epoch 1/100
 10/150 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.4713 - loss: 0.6935
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1714710646.541668      79 device_compiler.h:186] Compiled cluster using XLA!  This line is logged at most once
r the lifetime of the process.
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.4989 - loss: 0.6932
Epoch 1: val_accuracy improved from -inf to 0.49417, saving model to /kaggle/working/deepfake.keras
150/150 ━━━━━━━━━━━━━━━━━━━━ 19s 37ms/step - accuracy: 0.4990 - loss: 0.6932 - val_accuracy: 0.4942 - val_loss: 0.6931
Epoch 2/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5165 - loss: 0.6930
Epoch 2: val_accuracy did not improve from 0.49417
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.5162 - loss: 0.6930 - val_accuracy: 0.4942 - val_loss: 0.6931
Epoch 3/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.4958 - loss: 0.6931
Epoch 3: val_accuracy did not improve from 0.49417
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.4959 - loss: 0.6931 - val_accuracy: 0.4942 - val_loss: 0.6931
Epoch 4/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5095 - loss: 0.6931
Epoch 4: val_accuracy did not improve from 0.49417
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.5094 - loss: 0.6931 - val_accuracy: 0.4942 - val_loss: 0.6930
Epoch 5/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5136 - loss: 0.6930
Epoch 5: val_accuracy did not improve from 0.49417
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.5134 - loss: 0.6930 - val_accuracy: 0.4942 - val_loss: 0.6930
Epoch 6/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.5024 - loss: 0.6929
Epoch 6: val_accuracy did not improve from 0.49417
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.5023 - loss: 0.6929 - val_accuracy: 0.4942 - val_loss: 0.6929

 150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9373 - loss: 0.1812 - val_accuracy: 0.9600 - val_loss: 0.1112
 Epoch 37/100
 148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9365 - loss: 0.1775
 Epoch 37: val_accuracy did not improve from 0.96000
 150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9366 - loss: 0.1772 - val_accuracy: 0.9267 - val_loss: 0.1766
 Epoch 38/100
 148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9504 - loss: 0.1491
 Epoch 38: val_accuracy did not improve from 0.96000
 150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9503 - loss: 0.1491 - val_accuracy: 0.9183 - val_loss: 0.1957
 Epoch 39/100
 148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9474 - loss: 0.1529
 Epoch 39: val_accuracy improved from 0.96000 to 0.96500, saving model to /kaggle/working/deepfake.keras
 150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.9474 - loss: 0.1528 - val_accuracy: 0.9650 - val_loss: 0.1007
 Epoch 40/100
 148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9496 - loss: 0.1410
 Epoch 40: val_accuracy did not improve from 0.96500
 150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9497 - loss: 0.1409 - val_accuracy: 0.9567 - val_loss: 0.1149
 Epoch 41/100
 148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9549 - loss: 0.1291
 Epoch 41: val_accuracy improved from 0.96500 to 0.96667, saving model to /kaggle/working/deepfake.keras
 150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.9549 - loss: 0.1290 - val_accuracy: 0.9667 - val_loss: 0.0950
 Epoch 42/100
 148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9569 - loss: 0.1286
 Epoch 42: val_accuracy improved from 0.96667 to 0.96833, saving model to /kaggle/working/deepfake.keras
 150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9567 - loss: 0.1289 - val_accuracy: 0.9683 - val_loss: 0.0895
 Epoch 43/100
 148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9383 - loss: 0.1634
```

```
Epoch 93: val_accuracy did not improve from 0.98917
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9782 - loss: 0.0589 - val_accuracy: 0.9858 - val_loss: 0.0348
Epoch 94/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9804 - loss: 0.0589
Epoch 94: val_accuracy did not improve from 0.98917
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9804 - loss: 0.0590 - val_accuracy: 0.9683 - val_loss: 0.0873
Epoch 95/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9843 - loss: 0.0468
Epoch 95: val_accuracy did not improve from 0.98917
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9843 - loss: 0.0470 - val_accuracy: 0.9867 - val_loss: 0.0353
Epoch 96/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9790 - loss: 0.0525
Epoch 96: val_accuracy did not improve from 0.98917
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9791 - loss: 0.0524 - val_accuracy: 0.9642 - val_loss: 0.1084
Epoch 97/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9810 - loss: 0.0504
Epoch 97: val_accuracy improved from 0.98917 to 0.99000, saving model to /kaggle/working/deepfake.keras
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9809 - loss: 0.0505 - val_accuracy: 0.9900 - val_loss: 0.0323
Epoch 98/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9832 - loss: 0.0478
Epoch 98: val_accuracy did not improve from 0.99000
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9832 - loss: 0.0478 - val_accuracy: 0.9767 - val_loss: 0.0557
Epoch 99/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9573 - loss: 0.1243
Epoch 99: val_accuracy did not improve from 0.99000
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.9576 - loss: 0.1234 - val_accuracy: 0.9900 - val_loss: 0.0302
Epoch 100/100
148/150 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9895 - loss: 0.0404
Epoch 100: val_accuracy did not improve from 0.99000
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9895 - loss: 0.0403 - val_accuracy: 0.9867 - val_loss: 0.0317
```

```
Epoch 1/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 167s 912ms/step - accuracy: 0.7710 - loss: 0.4667 - val_accuracy: 0.8125 - val_loss: 0.3774
Epoch 2/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 106s 648ms/step - accuracy: 0.9200 - loss: 0.2048 - val_accuracy: 0.8125 - val_loss: 0.3506
Epoch 3/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 107s 656ms/step - accuracy: 0.9256 - loss: 0.1724 - val_accuracy: 0.8125 - val_loss: 0.3610
Epoch 4/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 106s 648ms/step - accuracy: 0.9371 - loss: 0.1589 - val_accuracy: 0.7500 - val_loss: 0.5026
Epoch 5/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 107s 657ms/step - accuracy: 0.9283 - loss: 0.1636 - val_accuracy: 0.8125 - val_loss: 0.3885
Epoch 6/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 105s 642ms/step - accuracy: 0.9427 - loss: 0.1411 - val_accuracy: 0.8125 - val_loss: 0.3059
Epoch 7/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 103s 631ms/step - accuracy: 0.9467 - loss: 0.1347 - val_accuracy: 0.8125 - val_loss: 0.4162
Epoch 8/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 104s 634ms/step - accuracy: 0.9416 - loss: 0.1421 - val_accuracy: 0.8125 - val_loss: 0.3061
Epoch 9/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 103s 630ms/step - accuracy: 0.9480 - loss: 0.1327 - val_accuracy: 0.8125 - val_loss: 0.3595
Epoch 10/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 105s 644ms/step - accuracy: 0.9443 - loss: 0.1349 - val_accuracy: 0.8125 - val_loss: 0.4496
Epoch 11/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 101s 619ms/step - accuracy: 0.9484 - loss: 0.1242 - val_accuracy: 0.8125 - val_loss: 0.3960
Epoch 12/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 104s 640ms/step - accuracy: 0.9450 - loss: 0.1418 - val_accuracy: 0.8125 - val_loss: 0.3087
Epoch 13/20
...
Epoch 19/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 102s 624ms/step - accuracy: 0.9523 - loss: 0.1219 - val_accuracy: 0.8125 - val_loss: 0.3841
Epoch 20/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 103s 634ms/step - accuracy: 0.9505 - loss: 0.1250 - val_accuracy: 0.8125 - val_loss: 0.3777
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```
# # video display:

# video = './test_clip.mp4'
# play_video(video)
```

```
# # prediction:
# video = './test_clip.mp4'
# pred = features.video_classifier(video)

# if(pred==1):
#     print("Deepfake")
# elif(pred==0):
#     print("Real")
# else:
#     print("No face")
```

CONCLUSION

The Deepfake Detection System developed using deep learning demonstrates the effectiveness of Convolutional Neural Networks (CNNs) in distinguishing real and AI-generated facial images. The custom CNN model successfully classifies images with high accuracy and reliability, proving that deep learning techniques can capture subtle visual artifacts that are often undetectable to the human eye.

Through systematic preprocessing, normalization, and model optimization, the system achieves strong generalization performance on unseen data. The evaluation metrics—including accuracy, precision, recall, confusion matrix, and ROC curve—confirm that the model is capable of robust and consistent deepfake detection.

While not a complete solution to all forms of digital manipulation, the system serves as a valuable tool for enhancing digital content verification. It can support cybersecurity teams, social media platforms, and forensic analysts by reducing manual inspection effort, improving detection speed, and strengthening trust in digital media.

REFERENCES

[1] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, et al., **"FaceForensics++: Learning to Detect Manipulated Facial Images,"** IEEE International Conference on Computer Vision (ICCV), 2019.

[2] Darius Afchar, Vincent Nozick, Junichi Yamagishi, et al., **"MesoNet: A Compact Facial Video Forgery Detection Network,"** IEEE WIFS, 2018.

[3] R. Tolosana, R. Vera-Rodriguez, J. Fierrez, et al., **"DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection,"** Information Fusion, 2020.

[4] Huy H. Nguyen, Junichi Yamagishi, Isao Echizen, **"Capsule-Forensics: Using Capsule Networks to Detect Forged Images and Videos,"** ICASSP, 2019.

[5] TensorFlow Documentation: **"Image Classification and CNN Model Development.**