DEEPFAKE DETECTION ON IMAGES USING MOBILENETV2

A Minor Project Report

Submitted to

Jawaharlal Nehru Technological University, Hyderabad

In partial fulfillment of the requirements for the

Award of the degree of

BACHELOR OF TECHNOLOGY

In

CSE(DATASCIENCE)

By

YASHWANTH REDDY CHEEMARLA (21VE1A6772)
ADURI MANVITHA (21VE1A6766)
VUTPALA SHREESHA (21VE1A67D0)
TUMMETI THREYA REDDY (21VE1A67C7)

Under the Guidance of

Dr. K. ROHIT KUMAR
ASSOCIATE PROFESSOR

SREYAS INSTITUTE OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF CSE (DATASCIENCE)

(Affiliated to JNTUH, Approved by A.I.C.T.E and Accredited by NAAC, New Delhi)
Bandlaguda, Beside Indu Aranya, Nagole, Hyderabad-500068,Ranga Reddy Dist.

2024-2025

SREYAS INSTITUTE OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF CSE (DATA SCIENCE)

CERTIFICATE

This is to certify that the Minor Project Report on "***Deepfake Detection On Images Using MOBILENETV2***" submitted by **CHEEMARLA YASWANTH REDDY, ADURI MANVITHA, VUTPALA SHREESHA, TUMMETI THREYA REDDY** bearing Hallticket Nos. **21VE1A6772, 21VE1A6766, 21VE1A67D0, 21VE1A67C7** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in CSE (DATA SCIENCE) from Jawaharlal Nehru Technological University, Kukatpally, Hyderabad for the academic year 2024-2025 is a record of bonafide work carried out by them under our guidance and Supervision.

Internal Guide
Dr. K. Rohit Kumar
Associate Professor

Project Coordinator
Dr. G.Naga Rama Devi
Associate Professor

Head of the Department
Dr. K. Rohit Kumar
Associate Professor

External Examiner

SREYAS INSTITUTE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF CSE (DATA SCIENCE)

DECLARATION

We, hereby declare that the Minor Project titled "***Deepfake Detection On Images Using MOBILENETV2***" done by us under the guidance of **Dr. K. Rohit Kumar, Associate Professor** which is submitted in the partial fulfillment of the requirement for the award of the **B. Tech degree in CSE (Data Science) at Sreyas Institute of Engineering and Technology** for Jawaharlal Nehru Technological University, Hyderabad is our original work.

CHEEMARLA YASWANTH REDDY (21VE1A6772)

ADURI MANVITHA (21VE1A6766)

VUTPALA SHREESHA (21VE1A67D0)

TUMMETI THREYA REDDY (21VE1A67C7)

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without mention of the people who made it possible through their guidance and encouragement crowns all the efforts with success.

We take this opportunity to acknowledge with thanks and deep sense of gratitude to **Dr. K. Rohit Kumar, Associate Professor, Department of CSE (Data Science)** for his constant encouragement and valuable guidance during the Project work.

A Special vote of Thanks to **Dr. G. Naga Rama Devi, Project Co-Ordinator** and **Dr. K. Rohit Kumar, Head of the Department**, who has been a source of Continuous motivation and support. They had taken time and effort to guide and correct me all through the span of this work.

We owe very much to the **Department Faculty, Principal** and Management who made my term at Sreyas a stepping stone for my career. We treasure every moment we had spent in the college.

Last but not the least, our heartiest gratitude to our parents and fellow students for their continuous encouragement. Without their support this work would not have been possible.

CHEEMARLA YASHWANTH REDDY (21VE1A6772)
ADURI MANVITHA (21VE1A6766)
VUTPALA SHREESHA (21VE1A67D0)
TUMMETI THREYA REDDY (21VE1A67C7)

ABSTRACT

Verifying the authenticity of digital content has become more challenging due to the extensive use of deepfake technology. Artificial intelligence-generated images that look real but are phoney are called deepfakes. Malicious use of them can undermine trust in visual evidence, spread misleading information, and create fraudulent identities. We provide an image-based deepfake identification technique that MobileNetV2's efficiency and accuracy to distinguish authentic images from fakes. The system was trained using a sizable dataset of actual and deepfake photographs, and data augmentation techniques including rotation and horizontal flipping were applied to strengthen the model's resilience. Performance metrics like F1-score, recall, accuracy, and precision were used to evaluate the MobileNetV2-based detector's effectiveness. A sequence of convolutional layers, depth-wise separable convolutions, inverted residuals, bottleneck design, and linear bottlenecks make up MobileNet-v2's architecture. Together, these elements help to minimise the number of calculations and parameters needed while preserving the model's capacity to capture intricate details. TensorFlow, Keras, and OpenCV are some of the several frameworks and components that are utilised. Additionally, methods like post-processing, attention mechanisms, and transfer learning are employed to improve its performance. In order to compare our networks to MobileNetV1 and ResNet, MobileNetV2 acts as an effective and efficient backbone for deepfake detection.

KEYWORDS: Convolutional neural networks, MobileNetV2, Inverted residuals, Bottleneck design, linear bottlenecks, TensorFlow, Keras, ResNet.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

List of Output Screenshots

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Technological Breakthroughs and Deepfake Evolution

The advancements in artificial intelligence (AI) and deep learning have significantly contributed to the creation of deepfake technology. Deepfakes leverage sophisticated algorithms, particularly Generative Adversarial Networks (GANs), to generate highly realistic images and videos that are nearly indistinguishable from authentic content. These synthetic media creations can be manipulated to make individuals appear to say or do things they never actually did, raising profound ethical and security concerns.

Ethical and Security Implications

The emergence of deepfake technology poses substantial threats across various domains:

- **Journalism and Media:** Deepfakes can disseminate propaganda and misinformation, eroding public trust in news sources. Instances such as the deepfake of former U.S. President Barack Obama illustrate how easily public figures can be misrepresented.
- **Legal System:** The credibility of digital evidence in legal proceedings can be compromised. For example, altered video footage presented as evidence could lead to wrongful convictions or the dismissal of crucial evidence.
- **Personal Security:** Individuals are at risk of identity theft and reputational damage. Notable cases include the use of deepfake technology to create fake pornography, leading to severe emotional and psychological distress for the victims.

The manipulation capabilities of deepfakes extend to various other sectors, including politics, corporate security, and social interactions, underlining the urgent need for effective detection systems.

1.2 MOTIVATION

Preserving Trust in Digital Media

The need to protect the integrity of digital media is the main driving force behind this effort. Retaining public trust requires media outlets and journalists to be able to trust the veracity of their reporting.

- **Case Study: Deepfake Videos in Political Campaigns**: Several deepfake videos surfaced during the 2020 U.S. Presidential Election, confusing voters and harming the democratic process.

Protecting Personal Identities

Preventing personal attacks and identity theft made possible by deepfake technology is essential. People must be certain that malicious manipulations won't affect their digital likeness.

- **Example: Deepfake Scams**: Instances of criminals using deepfake technology to impersonate executives and authorize fraudulent transactions highlight the urgent need for robust detection mechanisms.

Supporting Legal and Forensic Investigations

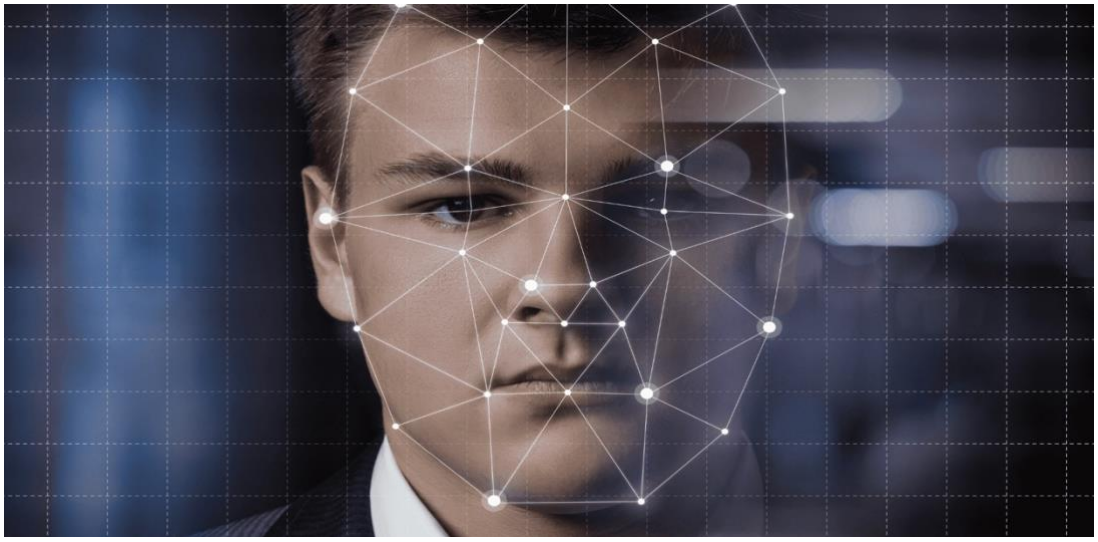
Maintaining the integrity of digital evidence is essential for the justice system. Deepfake detection tools can aid in verifying the authenticity of digital evidence.

- **Forensic Analysis:** Incorporating deepfake detection tools in forensic analysis can enhance the reliability of digital evidence, ensuring that justice is served accurately.

Enhancing Cybersecurity

Deepfakes can facilitate sophisticated social engineering attacks, posing a significant threat to cybersecurity. By detecting and mitigating these threats, organizations can better protect their assets and information.

- **Cybersecurity Measures:** Developing automated deepfake detection systems for integration into cybersecurity frameworks can prevent the spread of false information and protect against potential attacks



1.3 SCOPE

Model Development

The project aims to develop a deepfake detection model using MobileNetV2, a lightweight convolutional neural network (CNN) known for its efficiency and accuracy. MobileNetV2 is particularly suited for mobile and embedded vision applications, making it an ideal choice for this project

- **Technical Details**: Implementing the model involves utilizing TensorFlow and Keras, with a focus on leveraging pre-trained weights for transfer learning to enhance model performance

Utilization of Datasets

A diverse range of datasets, comprising both real and deepfake images, will be used to train the model. This ensures the model's robustness and ability to generalize across different types of data.

- **Dataset Examples:** Utilizing publicly available datasets such as Face Forensics++, Deepfake Detection Challenge Dataset, and others will provide a comprehensive training set for the model.

Performance Optimization

Optimizing the model's performance involves techniques like transfer learning, data augmentation, and fine-tuning. These methods enhance the model's accuracy and efficiency.

- **Optimization Strategies**: Data augmentation techniques, such as rotation and horizontal flipping, will improve the model's robustness. Additionally, appropriate loss functions, optimizers, and regularization strategies will be employed to prevent overfitting

Deployment Feasibility

Adapting the model for deployment in resource-constrained environments is a key aspect of the project's scope. Ensuring that the model can operate effectively on mobile and edge devices is crucial for its practical applicability.

- **Deployment Considerations:** The model will be optimized for low-power devices, ensuring it meets computational constraints while maintaining high accuracy.

1.4 OBJECTIVES

Building a Model for Deepfake Detection

Developing an image analysis-specific deepfake detection model using MobileNetV2 is the primary objective. The model will be implemented with TensorFlow and Keras, utilizing pre-trained weights for transfer learning.

- **Implementation Details**: The model architecture will be designed to balance accuracy and computational efficiency, making it suitable for deployment on various devices.

Instruction and Enhancement

Training the model on a large dataset that includes both real and deepfake images is essential. Data augmentation techniques will be used to enhance the model's resilience and generalization capabilities.

- **Training Process:** The model will be trained using a combination of real and synthetic images, with data augmentation techniques such as rotation and horizontal flipping to improve robustness. Transfer learning will be employed to leverage pre-trained weights, enhancing the model's performance.

Evaluation of Performance

Validating and testing the model on diverse datasets will ensure its accuracy and reliability.

- **Performance Metrics:** The model's performance will be evaluated using various metrics, ensuring its reliability across different datasets.

Deployment and Application

Adapting the model for deployment on mobile and edge devices is a critical objective. Developing an application or interface that utilizes the model for real-time deepfake detection will demonstrate its practical applicability.

- **Real-time Detection:** The final application will be designed to provide real-time deepfake detection capabilities, showcasing the model's effectiveness in practical scenarios.

CHAPTER 2

LITERATURE SURVEY

2.1 OVERVIEW

Deepfake detection using deep learning has garnered significant attention due to the growing capabilities of AI to create highly realistic synthetic media. This project aims to develop a deepfake detection system using MobileNetV2, focusing on images. The following sections provide an in-depth exploration of definitions, purposes, evolution, and existing methods in deepfake detection.

2.1.1 Definition

Deepfakes are synthetic media in which a person in an existing image or video is replaced with someone else's likeness using AI techniques. These manipulations are achieved using deep learning, a subset of AI that trains neural networks on vast datasets to recognize patterns and generate realistic outputs. Deepfake detection involves using these same AI techniques to identify and flag manipulated media, ensuring the authenticity and integrity of visual content.

2.1.2 Purpose

The primary purpose of this project is to develop a robust system for detecting deepfake images using MobileNetV2, a neural network architecture known for its efficiency and accuracy in image classification tasks. The goals include:

- **Accuracy**: Achieve high detection accuracy to distinguish between real and fake images
- **Efficiency**: Ensure the system is lightweight and fast, suitable for real-time applications.
- **Scalability:** Develop a system that can handle large-scale data and various types of deepfake manipulations.
- **Robustness**: Ensure the system can withstand adversarial attacks and various deepfake generation techniques.

2.1.3 Evolution of Deepfake Detection Using Deep Learning

The evolution of deepfake detection has been rapid, driven by advances in deep learning. Initial methods relied on traditional machine learning techniques, which struggled with the complexity and realism of deepfakes. The advent of deep learning, especially convolutional neural networks (CNNs), brought significant improvements. These models excel at recognizing subtle patterns and anomalies in images, making them ideal for detecting deepfakes. Key milestones in the evolution of deepfake detection include:

- **Early Machine Learning Methods**: Initial attempts used handcrafted features and simple classifiers, which were insufficient for realistic deepfakes.
- **CNNs and RNNs**: The introduction of deep learning models like CNNs and recurrent neural networks (RNNs) significantly improved detection accuracy.
- **Transfer Learning:** Leveraging pre-trained models on large datasets to enhance performance on deepfake detection tasks.
- **Hybrid Models**: Combining multiple models and techniques to improve robustness and accuracy.
- **Adversarial Training:** Training models to recognize deepfakes generated by adversarial networks, enhancing their ability to detect novel manipulations.

2.2 LITERATURE REVIEW ON EXISTING METHODS

This section reviews existing methods for deepfake detection, focusing on their methodologies, strengths, and limitations. The literature survey includes various approaches, from traditional machine learning to advanced deep learning models.

2.2.1 DFFMD: A Deepfake Face Mask Dataset for Infectious Disease Era

Methodology: The DFFMD dataset was created to address deepfake detection challenges posed by masked faces. The dataset includes images of individuals wearing face masks, both real and deep faked

- **Strengths**:
    - Addresses the unique challenge of detecting deepfakes on masked faces.
    - Provides a diverse set of images for training and evaluation.

- **Limitations:**
    - Limited to scenarios involving face masks, may not generalize to other types of deepfakes.

2.2.2 Deepfake Detection on Social Media

Methodology: This study focused on detecting machine-generated tweets using deep learning and Fast Text embeddings. CNNs and RNNs were used for feature extraction and classification.

- **Strengths:**
    - Effective in distinguishing between human and machine-generated content.
    - Demonstrates the application of deepfake detection in text-based media
- **Limitations:**
    - Primarily focuses on text, may not directly translate to image-based deepfake detection.

2.2.3 Analysing Model Generalization

Methodology: This research analysed the generalization capabilities of deepfake detection models across different architectures, datasets, and pre-training paradigms

- **Strengths:**
    - Provides insights into factors affecting model performance and robustness.
    - Highlights the importance of generalization in deepfake detection
- **Limitations:**
    - Generalization analysis may not directly provide new detection techniques.

2.2.4 Systematic Literature Review on Deepfake Detection

Methodology: A comprehensive review of existing research, summarizing state-of-the art methods, approaches, techniques, and challenges in deepfake detection.

- **Strengths:**
    - Offers a broad overview of the field, identifying key trends and gaps.
    - Useful for understanding the landscape of deepfake detection research.
- **Limitations:**
    - May not provide detailed technical insights or specific solutions.

2.2.5 MobileNetV2 in Deepfake Detection

Methodology: MobileNetV2 is a lightweight CNN architecture optimized for mobile and embedded vision applications. This section explores its application in deepfake detection, focusing on its architecture, training process, and performance metrics.

- **Architecture:**
    - Depthwise Separable Convolutions: Reduces the number of parameters and computational cost, making the model efficient.
    - Inverted Residuals and Linear Bottlenecks: Enhance feature extraction capabilities and model performance.
- **Training Process:**
    - Data Augmentation: Enhances the diversity of the training set, improving model robustness.
    - Transfer Learning: Utilizes pre-trained weights from large datasets to improve detection accuracy.
    - Fine-Tuning: Adjusts the model to the specific deepfake detection task.
- Performance Metrics:
    - Accuracy: Measures the proportion of correctly identified deepfakes.
    - Precision and Recall: Evaluate the model's ability to correctly identify deepfakes without false positives or negatives.
    - F1 Score: Balances precision and recall, providing a single metric for model performance

2.2.6 Comparative Analysis of Detection Techniques

Methodology: This section provides a comparative analysis of different deepfake detection techniques, including traditional machine learning methods, CNNs, RNNs, and transformer-based models.

- **Comparative Metrics:**
  - Accuracy: Overall detection performance.
  - Efficiency: Computational cost and speed.
  - Scalability: Ability to handle large datasets and various types of deepfakes.
  - Robustness: Resistance to adversarial attacks and novel deepfake techniques.
- **Key Findings:**
  - CNNs: Excel at image-based deepfake detection due to their powerful feature extraction capabilities.
  - RNNs: Useful for sequential data and temporal inconsistencies in videos.
  - Transformer Models: Offer state-of-the-art performance in various tasks but are computationally intensive.

2.2.7 Challenges and Future Directions

Current Challenges:

- Adversarial Attacks: Deepfake generation techniques are continually evolving, posing challenges for detection models.
- Real-Time Detection: Ensuring the system can detect deepfakes in real-time applications.
- Dataset Limitations: The need for diverse and large-scale datasets to train robust models.

Future Directions:

- Adversarial Training: Incorporating adversarial examples in training to improve model robustness.
- Hybrid Models: Combining multiple detection techniques to enhance performance.
- Ethical AI: Ensuring ethical considerations are integrated into the development and deployment of detection systems.

2.2.8 Ethical Considerations and Impact

Ethical Implications:

- Privacy: Concerns about the misuse of personal images and data.

- Consent: The importance of obtaining consent before using individuals' images for deepfake generation.
- Misinformation: The potential for deepfakes to spread false information and harm reputations.

2.2 Summary of selected papers

Table 2.2 summary of selected papers

Paper Title	Dataset	Methodology	Accuracy	Pros	Cons
Deepfake Detection on social media	Tweet fake	Text Embeddings, Deep learning	93%	Evaluates robustness, Highlights strengths and weakness, assesses dataset impact	High computational resources required
Deepfake Detection	FF++, DFD	Multimodal Fusion techniques	99.65%	Comprehensive evaluation of different architectures, Datasets, pre-training paradigms	Extensive datasets required
Deepfake Face Mask Detection	DFFMD	Robust detection algorithm	99.81%	High accuracy in face mask detection	High computational resources required

This summary retains the core details in a concise format.

CHAPTER 3

METHODOLOGY

This chapter details the methodology employed for developing the deepfake detection system using deep learning and MobileNetV2. It covers the problem description, the proposed system methodology, input data, and system requirements, including both hardware and software.

3.1 PROBLEM DESCRIPTION

The proliferation of deepfake technology poses significant challenges across various domains, including security, media, and public trust. Deepfake images, created using advanced AI techniques, can be highly convincing and difficult to detect with the naked eye. This project addresses the problem of identifying and mitigating the impact of such manipulated media by developing an efficient and accurate detection system.

- **Case Study:** Media Manipulation: In 2020, a deepfake video of a political leader was widely circulated, creating public confusion and distrust. Such incidents underline the urgent need for reliable detection systems.
- **Security Concerns:** Deepfake technology can be used for malicious purposes, such as identity theft and fraud. For example, deepfake audio was used to impersonate a CEO and authorize a fraudulent transaction of $243,000.

3.2 PROPOSED SYSTEM METHODOLOGY

The proposed system leverages MobileNetV2, a lightweight and efficient convolutional neural network (CNN) architecture, to detect deepfake images. The methodology encompasses several stages, including data collection, preprocessing, model training, and evaluation.

3.2.1 Input Data

Data Collection:

- Dataset Sources: The dataset is collected from publicly available sources and includes both real and deepfake images. Sources include the DFDC (Deepfake Detection Challenge) dataset, Celeb-DF, and others.
- Data Diversity: Ensuring a diverse set of images with varying resolutions, lighting conditions, and backgrounds to improve model robustness.

Data Preprocessing:

- Resizing and Normalization: Images are resized to a standard dimension suitable for MobileNetV2 and normalized to ensure consistent input for the model.
- Augmentation: Techniques such as rotation, flipping, and colour adjustments are applied to increase the dataset's variability and improve the model's generalization capabilities.
- Labelling: Images are labelled as 'real' or 'fake' to provide ground truth for model training

Example:

- Resizing: All images are resized to 224x224 pixels.
- Normalization: Pixel values are scaled to the range [0, 1].
- Augmentation: Random rotations up to 30 degrees, horizontal flips, and brightness adjustments.

3.2.2 Model Architecture

MobileNetV2 Architecture:

- Depthwise Separable Convolutions: Reduces the number of parameters and computational cost while maintaining performance.
- Inverted Residuals and Linear Bottlenecks: Enhance the model's feature extraction capabilities and efficiency.

Training Process:

- Transfer Learning: Utilizing pre-trained weights on large datasets (e.g., ImageNet) to initialize the model, followed by fine-tuning on the deepfake detection dataset
- Optimization and Loss Functions: Using Adam optimizer and binary crossentropy loss to train the model
- Evaluation Metrics: Accuracy, precision, recall, and F1 score are used to evaluate the model's performance.

Model Evaluation:

- Cross-Validation: Implementing k-fold cross-validation to assess the model's generalization ability.

- Confusion Matrix: Analysing true positives, true negatives, false positives, and false negatives to understand the model's strengths and weaknesses.

Example:

• Adam Optimizer: Learning rate of 0.001.

• Loss Function: Binary cross-entropy.

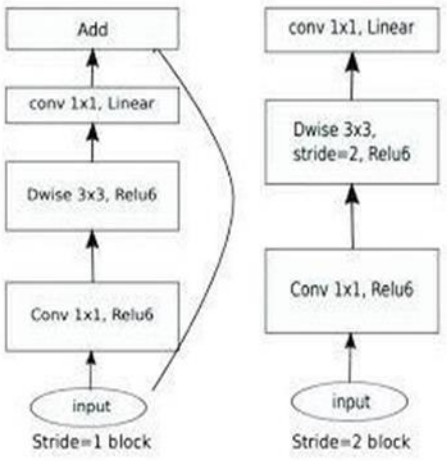


Fig 3.2.2 Model Architecture

High-Level Overview of Deepfake Detection Architecture Using MobileNetV2

Introduction

In the realm of digital media, deepfake technology has presented a significant challenge due to its ability to generate highly realistic but fake images and videos. Detecting these synthetic media forms is crucial for maintaining the integrity of visual content. The deepfake detection system described here leverages MobileNetV2 as a backbone for feature extraction, fine-tuning layers for enhanced processing, and an output layer for binary classification. This architecture ensures a robust and efficient method to classify images as real or fake.

Input Image

The system begins with an input image, typically of a person or scene, that needs to be assessed for authenticity. The image dimensions are set to 128x128 pixels

with three colour channels (RGB), forming the basis for subsequent processing steps. The input layer processes these images into a suitable format for the neural network.

- Width: 128 pixels
- Height: 128 pixels
- Channels: 3 (RGB)
- Data Format: A 3D array representing RGB pixel data

MobileNetV2 Backbone

MobileNetV2, a well-known architecture designed for efficient mobile and embedded vision applications, serves as the core feature extraction component. It utilizes depth wise separable convolutions, which significantly reduce the number of parameters and computational load compared to traditional convolutional layers. MobileNetV2 is pretrained on the ImageNet dataset, which provides a rich feature base for transfer learning.

• **Pretrained**: Yes

• **Trained On:** ImageNet

• **Key Function:** Feature extraction through convolutional layers.

The convolutional layers of MobileNetV2 effectively capture intricate details from the input image, generating a feature map that represents essential characteristics necessary for classification.

Additional Layers for Fine-Tuning

To adapt MobileNetV2 for the specific task of deepfake detection, additional layers are integrated for fine-tuning. These layers include global average pooling, dropout layers, flattening layers, and fully connected layers.

- **Global Average Pooling**: Reduces the spatial dimensions of the feature map while preserving essential features.
- **Dropout Layers**: Introduce regularization to prevent overfitting by randomly dropping units during training.
    - **Rate:** 0.5
- **Flatten Layer**: Converts the 2D feature map into a 1D feature vector, suitable for dense layers.
- **Fully Connected Layers:** Process the flattened feature vector, enabling deeper feature integration and refinement.

- **Units**: 512
- **Activation Function:** ReLU (Rectified Linear Unit)

Dense Layers

Following the extraction and initial processing, dense layers further process the features, refining them for the final classification task.

- **Flatten Layer:** This layer takes the multi-dimensional output from the convolutional layers and flattens it into a one-dimensional vector, making it suitable for the fully connected layers.
  - **Function:** flatten ()
- **Fully Connected Layer:** This layer consists of neurons that are fully connected to the previous layer, allowing for the integration and combination of features extracted earlier.
  - Units: 512 o Activation Function: dense activation.relu()
- **Dropout Layer:** Adds regularization to the network, helping to prevent overfitting.
  - **Rate**: 0.5
  - **Function:** apply dropout ()

Output Layer

The final component of the architecture is the output layer, designed for binary classification. This layer utilizes a sigmoid activation function to produce a probability score, indicating the likelihood of the input image being real or fake.

- Units: 1 (for binary classification)
- Activation Function: Sigmoid
- Predicted Output: Probability score (ranging from 0 to 1)
  - Function: dense_activation.sigmoid ()

The sigmoid function is ideal for binary classification as it outputs a value between 0 and 1, which can be interpreted as the probability of the image being a deepfake.

Detailed Explanation and Analysis

MobileNetV2 Feature Extraction

MobileNetV2's architecture is optimized for mobile and embedded vision applications. Its efficiency comes from depthwise separable convolutions, which decompose the standard convolution operation into a depthwise convolution

followed by a pointwise convolution. This decomposition significantly reduces the computational cost and number of parameters, making MobileNetV2 a lightweight yet powerful feature extractor. By using a pre-trained MobileNetV2, the deepfake detection system leverages the rich and diverse feature representations learned from the ImageNet dataset. These features include edges, textures, shapes, and higher-level constructs, which are crucial for identifying the subtle manipulations present in deepfake images.

Fine-Tuning Layers

Fine-tuning involves adapting a pre-trained model to a specific task by adding new layers and training them on task-specific data. In this architecture, the fine-tuning layers are crucial for enhancing the model's ability to detect deepfakes.

- **Global Average Pooling**: This layer reduces the spatial dimensions of the feature map while preserving the most relevant features. It effectively condenses the feature map into a lower-dimensional representation, making it easier for subsequent layers to process.
- **Dropout Layers:** Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of input units to zero during training. This forces the network to learn redundant representations, improving its generalization ability.
- **Flatten Layer:** The flattening layer transforms the 2D feature map into a 1D vector, which is necessary for the fully connected layers that follow.
- **Fully Connected Layers:** These layers integrate and refine the features extracted by MobileNetV2. The dense connections enable the model to learn complex combinations of features, enhancing its capability to distinguish between real and fake images.

Classification Output

The output layer is designed for binary classification, employing a sigmoid activation function to produce a probability score. The score indicates the likelihood of the input image being a deepfake. This probabilistic output can be interpreted with a threshold to classify the image as real or fake. For instance, a threshold of 0.5 means that images with a probability score above 0.5 are classified as fake, while those below are classified as real.

Training and Evaluation

The training process involves feeding the model a large dataset of labelled images (both real and fake) and optimizing the model's parameters to minimize the classification error. Standard techniques such as backpropagation and gradient descent are used for this optimization. During training, the dropout layers help mitigate overfitting by ensuring the model does not rely too heavily on specific features. Evaluation of the model is done using separate validation and test datasets to ensure the model generalizes well to unseen data. Key metrics such as accuracy, precision, recall, and F1-score are used to assess the model's performance.

3.3 HARDWARE AND SOFTWARE REQUIREMENTS

3.3.1 Hardware Requirements

- System: Intel(R) Core (TM) i5-1135G7 @2.40GHz 2.42GHz
- Hard Disk: 512 GB SSD
- RAM: 16GB

3.3.2 Software Requirements

- Operating system: Windows 10 or above
- Coding Language: Python
- Version:3.11.10
- Tool: Google Colab
- Dataset: DFDC

CHAPTER 4

SYSTEM DESIGN

The system with an input data pipeline. Datasets are here processed by resizing images to fit the input size of MobileNetV2 (for example, 224x224), normalizing pixel values, and applying data augmentation techniques such as flipping and rotation. This allows the model to work well with the real-world differences. MobileNetV2 is the feature extraction backbone that gathers all the important features using its lightweight design and depthwise separable convolutions in an efficient way. It makes it excellent for places with less resources.

To enhance the detection performance, custom classification layers are added on top of MobileNetV2 to fine-tune features toward deepfake-specific artifacts such as blending inconsistencies. We train using appropriate loss functions, optimizers like Adam, and metrics such as F1-score. Early stopping and learning rate schedulers prevent overfitting while learning.

The trained model is saved in formats such as the TensorFlow Saved Model and then optimized using quantization, among other techniques. It is then deployed into applications as a REST API using the Flask toolset. A monitoring and feedback loop is established; it identifies newly emerging deepfake methods to adapt to, evaluates its performance after launch, collects instances that it has misclassified, and routinely retrains the model to further improve.

4.1 UML DIAGRAMS

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML. The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

Use Case Diagram

The system is divided into two major parts: Deepfake Detection and Web Application. In the detection part, users upload images using the interface. The system then processes images to find features and uses the trained deepfake detection model. This model gives results by classifying images as real or fake. Simultaneously, the web application handles the user interface, static files, and templates to make interaction smooth. Static files are image, CSS, and JavaScript files. Templates serve content to the user dynamically. The main script controls the backend logic that ties the web interface to the process of deepfake detection. Such a modular structure makes it simple to deploy the system effectively and ensures that users can correctly detect deepfakes without overcomplicating the experience. Each module supports good communication between different parts of the system and reliable performance.

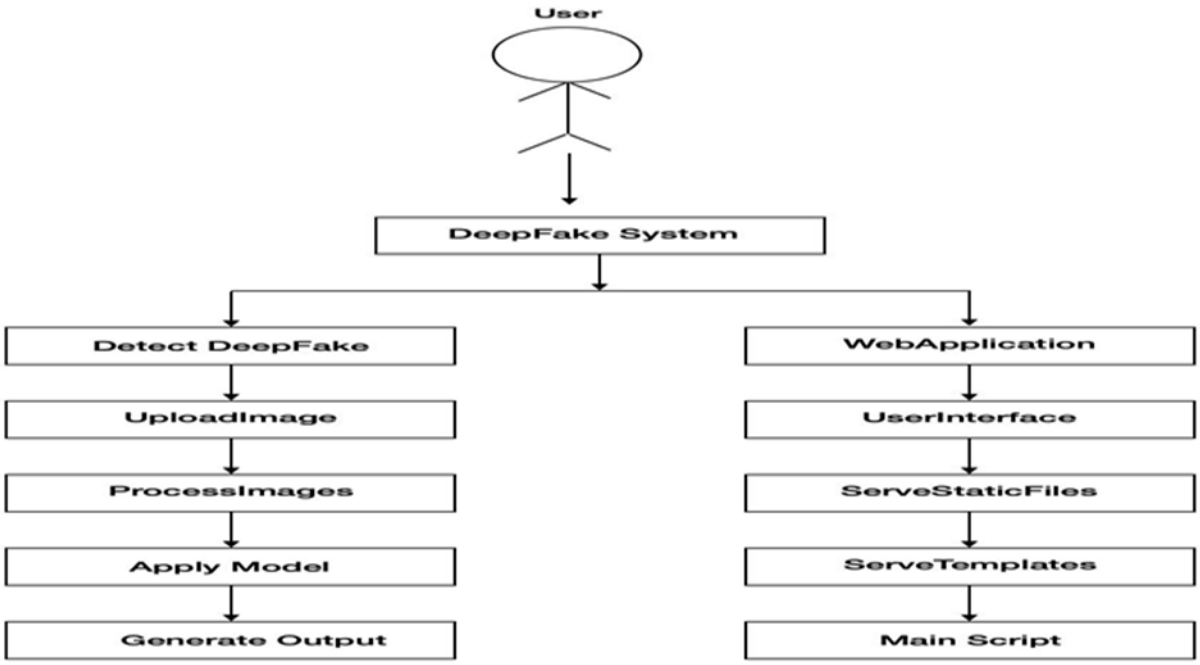


Fig 4.1.1 Use case diagram

Sequence Diagram

This sequence diagram illustrates the interaction dynamics among the different elements of an image detection system. The process is initiated with a user uploading an image via a web application. Following this, the web application preprocessed the image and sent it to the backend represented by `Index.py`. Subsequently, `Index.py` invoked the module `Detection.py`, which contained the logic to carry out the detection process. In the script `Detection.py`, the machine learning model is both loaded and initialized and enters "Model Ready" afterwards. The model then proceeds to execute detection once it's ready, passing its results to `Index.py`. The results from the analyzed detection are then sent back to the web application where it will show the results to the user. It points out the systematic interaction and exchange of data across the user interface, the backend process, and the machine learning model to work continuously in the image detection pipeline.

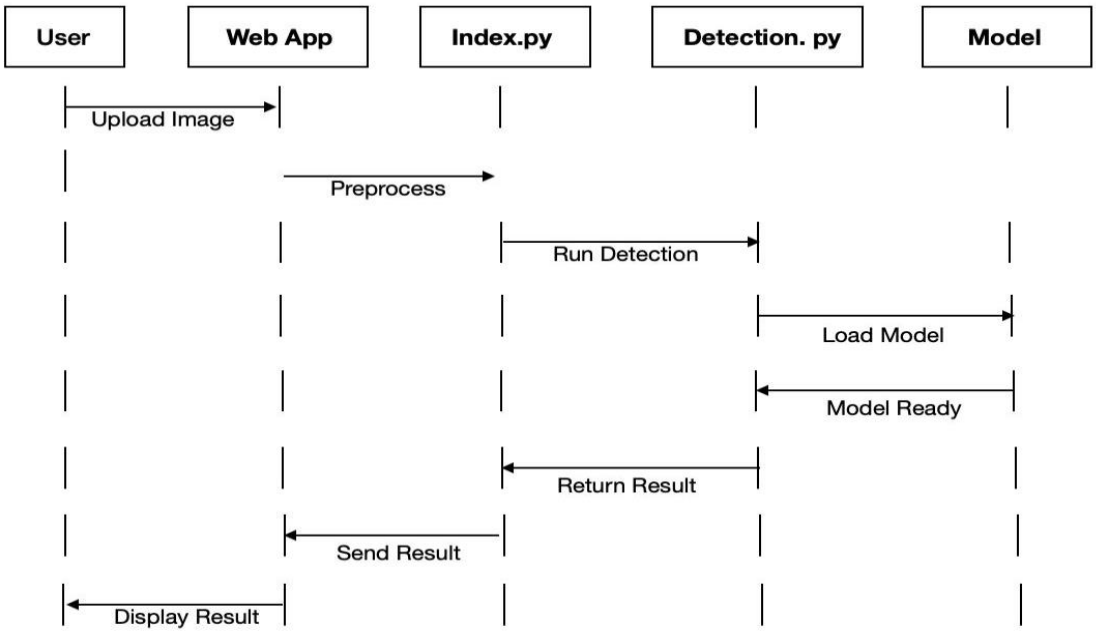


Fig 4.1.2 Sequence Diagram

Activity Diagram

This flowchart represents the methodology for Deepfake image identification. It begins with a web interface, from which a user uploads an image to be processed, applies preprocessing, and fits into a detection model. Next, there is a pre-trained Deepfake detection model that gets called up and applied to test the image. The detection will classify the image as "Fake" or "Real." Based on this classification, the result is then presented to the user. Then the process ends with a simple and accurate workflow in detecting Deepfakes.

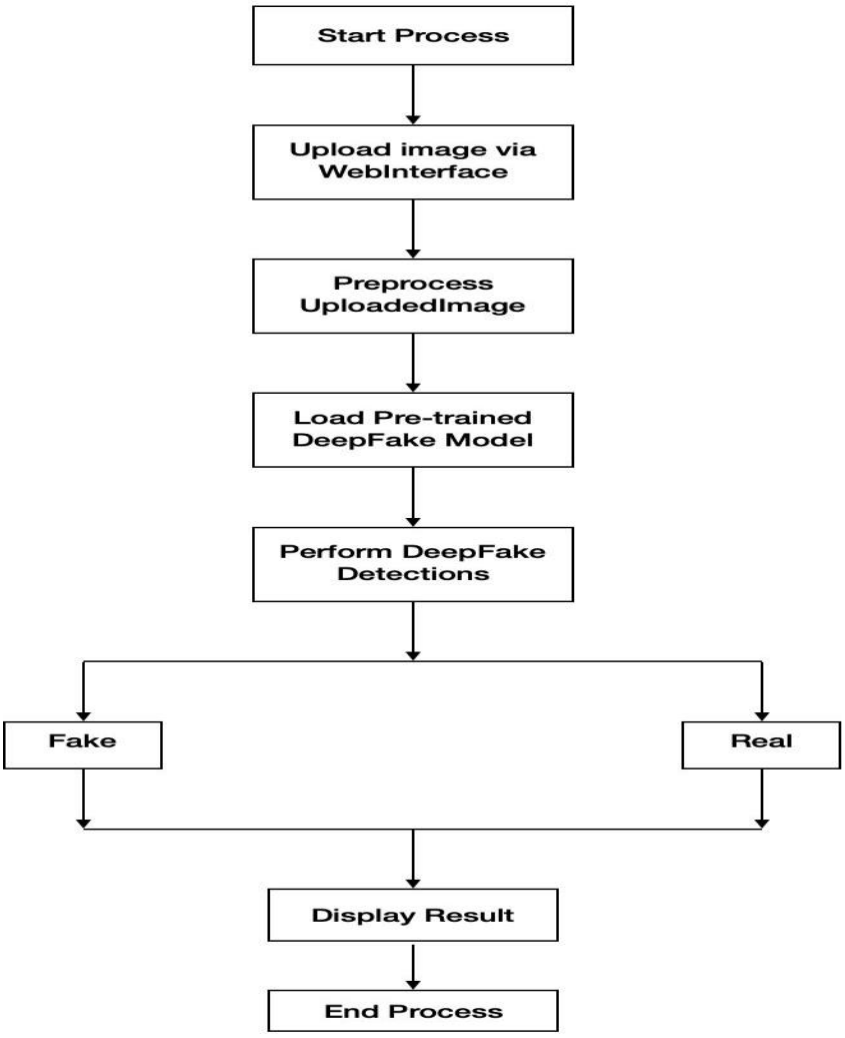


Fig 4.1.3 Activity Diagram

Class Diagram

It is a way to detect deepfakes within images using the MobileNetV2. The function begins with a user uploading an image with UPLOAD IMAGE(), located within the Image Uploader. It is then sent into the Preprocessor, getting the image ready for feature extraction through resizing (`RESIZE`), normalizing (`NORMALIZE`), and augmenting the data to make the model stronger. The processed image is fed into Feature Extractor, where the MobileNetV2 model is loaded by using LOAD MODEL () and further extracted feature representations are obtained through the command `EXTRACTFEATURES`. Further, these are fed to Deepfake Classifier, wherein `CLASSIFY` is used to decide the strength of the real or fake conclusion regarding the image. The confidence score can be accessed through GET CONFIDEN CESCORE (). The final step is the presentation of the results in the Result Display module. The `SHOWRESULT` function gives the classification label and the confidence level. This makes the process of detection of deepfakes fast and accurate, where each part contributes to a lucid process.

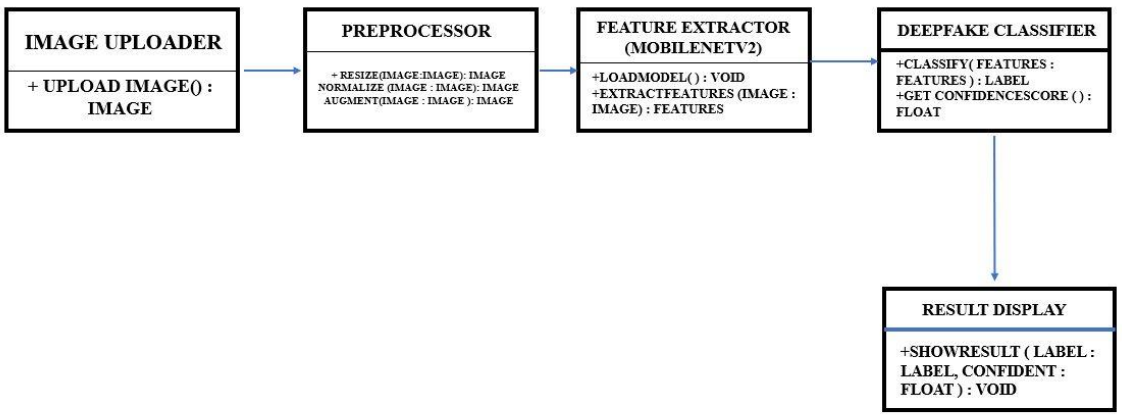


Fig 4.1.4 Class Diagram

4.2 SYSTEM ARCHITECTURE

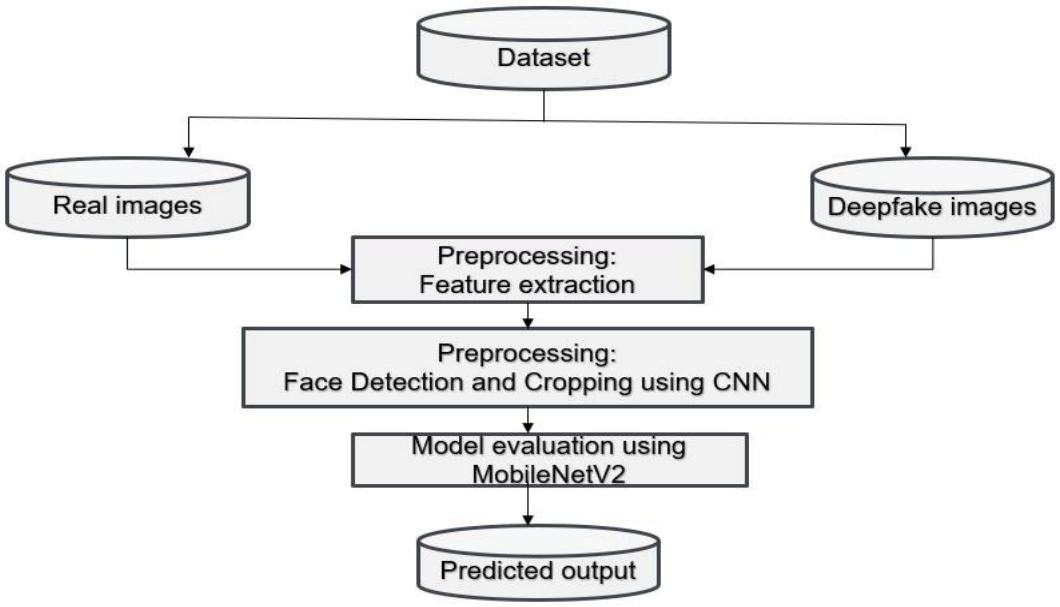


Fig 4.2.1 System Architecture

Dataset

- Comprises two categories: Real Images and Deepfake Images.
- Provides training and evaluation data for the model.
- Ensures diversity to handle multiple deepfake generation techniques.
- Represents authentic and manipulated image samples.
- Forms the foundation for building the detection pipeline.

Real Images

- Represent authentic and unaltered visual content.
- Serve as a baseline for comparison during classification.
- Contain natural patterns and characteristics.
- Used to train the model on identifying genuine features.
- Ensure the system can differentiate between real and fake images effectively.

Deepfake Images

- Created using AI-based techniques to manipulate or generate fake content.
- Represent manipulated visuals designed to mimic real ones.
- Test the model's ability to detect subtle inconsistencies.
- Include various generation methods like GANs and autoencoders.
- Challenge the system with highly realistic fakes for robust detection.

Preprocessing: Feature Extraction

- Identifies key patterns and characteristics in images.
- Transforms raw data into meaningful feature representations.
- Simplifies complex data for efficient processing.
- Ensures the model focuses on critical details.
- Reduces computational overhead during classification.

Preprocessing: Face Detection and Cropping Using CNN

- Detects faces using a Convolutional Neural Network (CNN).
- Crops the detected facial regions for focused analysis.
- Eliminates irrelevant background data.
- Improves accuracy by processing only facial features.
- Prepares consistent inputs for the detection model.

Model Evaluation Using MobileNetV2

- MobileNetV2 is a lightweight deep learning model.
- Evaluates extracted features for real or fake classification.
- Balances computational efficiency and accuracy.
- Handles diverse datasets effectively.
- Provides the foundation for robust detection results.

Predicted Output

- Displays the final classification as real or deepfake.
- Provides a confidence score for prediction reliability.
- Helps in understanding model performance.
- Facilitates further analysis of detection results.
- Ensures clear and interpretable outputs for end users.

- **Data Flow diagram**

Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

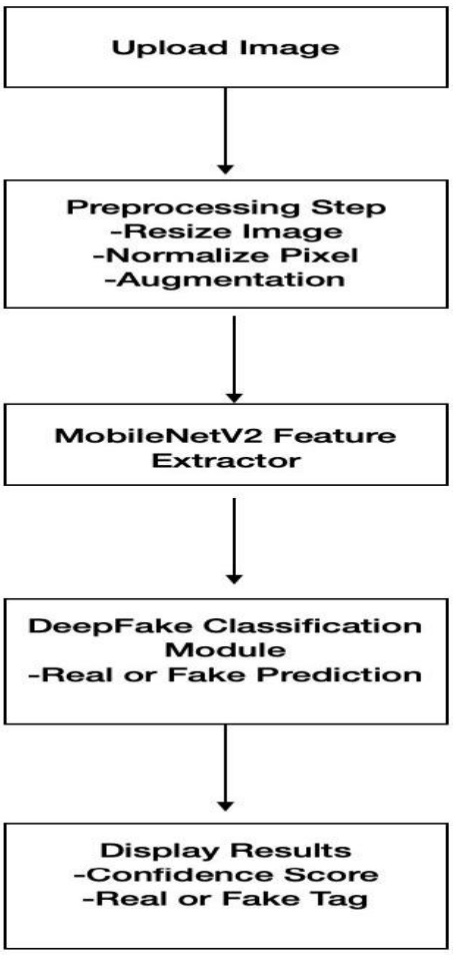


Fig 4.2.2 Data flow Diagram

Upload Image

- Users provide an image for analysis.
- Serves as the initial input for the pipeline.
- Supports images of varying resolutions and formats.
- Ensures the system receives an accessible and processable input.
- Acts as the entry point for further operations.

Preprocessing Step

- **Resize Image**: Standardizes the image dimensions.
- **Normalize Pixel**: Scales pixel values to improve consistency.
- **Augmentation**: Applies transformations like rotation or flipping to enhance model robustness.
- Ensures the input is clean and uniform for analysis.
- Prepares the image for feature extraction.

MobileNetV2 Feature Extractor

- Extracts key features from the preprocessed image.
- Utilizes MobileNetV2's lightweight and efficient architecture.
- Focuses on essential patterns for deepfake detection.
- Reduces irrelevant data for effective classification.
- Acts as the backbone for feature analysis.

Deepfake Classification Module

- Classifies the image as either real or fake.
- Uses the extracted features for accurate prediction.
- Employs a trained deep learning model for analysis.
- Ensures reliable identification of deepfake content.
- Prepares results for the display stage.

Display Results

- Outputs the classification result (real or fake).
- Provides a confidence score to indicate prediction reliability.
- Ensures clarity and interpretability of results.
- Displays both numerical and categorical outcomes.
- Allows users to verify and act upon the analysis.

CHAPTER 5

IMPLEMENTATION

5.1 MODULE DESCRIPTION

Implementing deepfake detection using CNNs and MobileNetV2 involves a series of structured steps that leverage deep learning techniques to identify whether an image is real or fake. Below is a detailed explanation of the process:

1. Setting Up the Environment

The first step is to prepare your development environment by installing the necessary software libraries and tools.

- **Install Python and Virtual Environment**

  First, install Python and create a virtual environment for the project. This will separate the environment so that different libraries or projects don't conflict with each other.

- **Install Libraries**

  Install the core libraries: TensorFlow, Keras, NumPy, OpenCV, Matplotlib, and scikit-learn for deep learning, data preprocessing, and analysis. Tools such as imgaug for data augmentation and TensorBoard for monitoring training are also recommended.

- **Set Up GPU Support (Optional)**

  If using an NVIDIA GPU, install the CUDA Toolkit and cuDNN compatible with your TensorFlow version. These enhance computational efficiency, speeding up the training process significantly.

- **Prepare and Preprocess the Dataset**

  Organize your dataset into labeled folders, usually real and fake images. The pre-processing of the data done is usually based on resizing images to a uniform size, for instance, 224 x 224 pixels, and then normalizing the pixel values. When dealing with video datasets, frames should be extracted to train on them**.**

- **Choose Development Tools and Platforms**

  One may use development tools such as Jupyter Notebook, PyCharm, or VS Code for writing code. However, one could also utilize cloud platforms such as Google Colab or AWS in case the hardware is not adequate.

- **Environment and Dependencies Test**

  Load a pre-trained model such as MobileNetV2. This ensures all the required libraries and configurations are correctly installed and the environment is ready for training the deepfake detection model.

This setup provides a solid foundation for building, training, and deploying the deepfake detection system.

2. Preparing the Dataset

Deepfake detection requires a dataset containing labeled real and fake images. The dataset should be diverse, representing a wide range of faces, lighting conditions, and image qualities**.**

- **Collect a Diverse Dataset**

  Acquire a dataset containing both real and fake labeled images or videos. Popular datasets like the Deepfake Detection Challenge and Face-forensics++ provide extensive samples. Ensure diversity in faces, lighting conditions, and image quality to enable the model to generalize across varied real-world scenarios effectively.

- **Organize the Data**

  Structure the dataset in a logical format, such as separate folders for real and fake images. This organization simplifies data loading and preprocessing, allowing the training process to run smoothly. Clear separation also ensures accurate labeling and efficient dataset management during experimentation.

- **Preprocess Images**

  Resize all images to a consistent size, like 224x224 pixels, to meet the input requirements of MobileNetV2 or similar architectures. This step standardizes image dimensions, enabling the model to process data uniformly and focus on feature extraction rather than handling variable input sizes.

- **Normalize Pixel Values**

  Normalize pixel values to a range of 0 to 1 by scaling the data. This improves the stability of the learning process, helping the model converge faster during training. Normalization ensures that pixel intensity variations do not overshadow meaningful patterns in the data.

- **Handle Image Data**

  For datasets containing images, extract individual frames to convert them into training samples. Frames increase the dataset's size and variety, capturing temporal differences between real and fake content. This approach provides the model with richer data, improving its ability to detect subtle inconsistencies.

- **Split the Dataset**

  Divide the dataset into training and validation sets, commonly in an 80:20 ratio. The training set trains the model, while the validation set assesses its performance on unseen data. This split helps evaluate the model's generalization capabilities and identifies overfitting issues during training**.**

A well-structured and preprocessed dataset lays the foundation for developing an accurate and robust deepfake detection model.

3. Data Augmentation

Data augmentation is crucial for improving the model’s ability to generalize to new images. Apply transformations such as random rotations, flips, zooming, shifts, and brightness adjustments to artificially expand the training dataset. This helps prevent overfitting, especially when the dataset size is limited. Augmentation ensures the model is exposed to variations in the data, improving its robustness.

- **Importance of Data Augmentation**

  Data augmentation artificially expands the dataset by creating variations of existing images. This technique improves the model’s ability to generalize to unseen data, reduces overfitting, and enhances its robustness against real-world variations like lighting, orientation, and noise.

- **Apply Geometric Transformations**

  Techniques such as random rotations, flips, and cropping introduce variability in image orientation and perspective. These transformations ensure the model does not rely on fixed patterns or alignments, enhancing its ability to detect manipulations in diverse scenarios.

- **Introduce Intensity Variations**

  Adjusting brightness, contrast, and saturation simulates real-world lighting changes. By exposing the model to different intensities, it learns to focus on distinguishing features rather than being influenced by environmental conditions like shadows or overexposure.

- **Use Zooming and Scaling**

  Random zooming in or out and scaling operations simulate changes in camera focus or subject distance. These augmentations help the model detect deepfake artifacts regardless of the subject's size or position within the frame.

- **Add Noise or Blur Effects**

  Introducing Gaussian noise, motion blur, or random pixel distortions mimics real-world imperfections in images or videos. These effects train the model to identify subtle manipulations, even in low-quality or compressed media files.

- **Combine Multiple Augmentations**

  Using multiple augmentation techniques together creates highly diverse training samples. This combined approach ensures the model sees a wide range of image variations, improving its resilience to different types of deepfake manipulations and improving overall accuracy.

Effective data augmentation is crucial for enhancing the dataset's diversity and ensuring the deepfake detection model performs reliably across a variety of challenging conditions.

4. Designing the Model Architecture

MobileNetV2 is a lightweight, pre-trained convolutional neural network designed for efficient feature extraction. Use it as the backbone for your deepfake detection model. Load MobileNetV2 with pre-trained weights (e.g., ImageNet) and exclude its top layers. Freeze the base layers initially to retain their pre-learned features**.**

- **Input Preprocessing**

Prepare input images by resizing and normalizing them to MobileNetV2's required dimensions, typically 224x224. Apply data augmentation techniques like rotation, flipping, and scaling to improve robustness. These steps ensure the model adapts to variations in image data, enhancing its ability to detect deepfake inconsistencies effectively.

- **Feature Extraction with MobileNetV2**

Leverage MobileNetV2 as the feature extraction backbone due to its lightweight and efficient design. Depthwise separable convolutions in MobileNetV2 reduce computation costs while capturing meaningful features. This ensures a scalable and effective framework for identifying subtle manipulation artifacts in deepfake images.

- **Custom CNN Layers**

Add custom convolutional layers after MobileNetV2 to enhance feature representation. These layers focus on identifying artifacts unique to fake images, such as texture irregularities and blending errors. This step improves the model's capacity to distinguish real images from deepfakes with greater precision.

- **Global Average Pooling**

Replace fully connected layers with global average pooling to simplify the model and reduce overfitting risks. This approach aggregates spatial information efficiently, ensuring a compact model while retaining key feature details necessary for accurate classification of deepfake and authentic images.

- **Classification Head**

Incorporate dense layers with softmax or sigmoid activation functions to classify images as real or fake. The classification head translates learned features into a probabilistic output, providing an interpretable and accurate result. This component ensures the model effectively distinguishes between classes.

- **Loss Function and Optimization**

  Use binary cross-entropy as the loss function and Adam optimizer for stable and efficient training. These methods ensure the model converges quickly and achieves high accuracy, particularly in identifying subtle inconsistencies present in manipulated images.

This architecture ensures efficient, accurate deepfake detection by combining MobileNetV2's lightweight design with enhanced feature extraction techniques.

5. Training the Model

Training a deepfake detection model involves optimizing it to accurately identify manipulated images by learning subtle inconsistencies, leveraging augmented datasets, efficient architectures, and suitable loss functions for robust and scalable performance.

- **Data Preparation**

  Organize a balanced dataset of real and fake images, ensuring diversity in manipulations and sources. Apply data augmentation techniques like rotation, flipping, and cropping to expand the dataset. This step improves the model's robustness and generalization capabilities by exposing it to varied image scenarios.

- **Splitting Data**

  Divide the dataset into training, validation, and testing sets (e.g., 70%, 15%, 15%). Ensure stratified sampling to maintain a balanced distribution of classes. The training set aids learning, the validation set fine-tunes hyperparameters, and the testing set evaluates model performance accurately.

- **Model Initialization**

  Initialize the architecture with pretrained MobileNetV2 weights for feature extraction, supplemented by custom CNN layers. This transfer learning approach reduces training time and improves performance, as the model benefits from knowledge gained from large, general-purpose image datasets.

- **Defining Loss Function and Optimizer**

  Select binary cross-entropy as the loss function for binary classification. Use Adam optimizer to adjust weights effectively and ensure stable convergence. Learning rate scheduling can be employed to refine training, balancing between fast convergence and avoiding overfitting**.**

- **Training and Validation**

  Train the model using the training set while monitoring its performance on the validation set. Track metrics like accuracy, precision, recall, and loss. Incorporate early stopping to halt training if validation performance plateaus, preventing overfitting and saving computational resources.

- **Evaluation on Test Set**

  Evaluate the trained model on the test set to measure its real-world performance. Use metrics like accuracy, F1-score, and confusion matrices to assess its ability to distinguish deepfake images. This step ensures the model's robustness and readiness for deployment in real scenarios**.**

Training the model ensures robust performance by leveraging diverse datasets, efficient optimization, and evaluation. This process prepares the model to detect deepfakes accurately, enabling effective deployment in real-world applications.

6. Evaluating the Model

Evaluating a deepfake detection model is crucial to assess its performance, robustness, and reliability. By analyzing metrics like accuracy, precision, recall, and F1-score, we ensure the model's ability to effectively identify manipulated images in real-world scenarios.

- **Accuracy Assessment**

  Measure the model’s overall accuracy by evaluating its correct predictions on the test set. This metric provides a straightforward indication of how well the model distinguishes real from fake images, highlighting its general performance and reliability in deepfake detection tasks.

- **Precision Evaluation**

  Analyze precision to determine the proportion of correctly identified fake images among all predicted fakes. High precision indicates the model's ability to minimize false positives, ensuring confidence in its detection of manipulated content without misclassifying genuine images as fake.

- **Recall Measurement**

  Evaluate recall to assess the model's capability to identify all fake images in the test set. A high recall reflects the model's effectiveness in minimizing false negatives, ensuring that most deepfakes are accurately detected without omission.

- **F1-Score Analysis**

  Compute the F1-score, a harmonic mean of precision and recall, to balance false positives and negatives. This metric provides a comprehensive measure of the model's performance, particularly when dealing with imbalanced datasets where one class might dominate.

- **Confusion Matrix Examination**

  Analyze the confusion matrix to gain detailed insights into the model's classification behavior. This visualization reveals true positives, true negatives, false positives, and false negatives, helping to identify specific areas for improvement in the model's detection capabilities.

- **Robustness Testing**

  Evaluate the model's robustness by testing it on unseen datasets or images with different manipulations. This step ensures the model's generalization to diverse scenarios, confirming its readiness for deployment in real-world applications where deepfake characteristics may vary widely.

Evaluating the model ensures its reliability and robustness by analyzing key metrics like accuracy, precision, recall, and F1-score. This process identifies strengths and weaknesses, confirming the model's readiness for deployment in detecting deepfakes in real-world scenarios.

7. Saving and Deploying the Model

Once the model achieves satisfactory performance, save it for future use. Saved models can be deployed in production environments to classify new images as real or fake. During deployment, preprocess input images in the same manner as during training (resize, normalize, etc.) to ensure consistency. The saved model can be integrated into applications such as social media platforms, content moderation tools, or forensics software.

- **Model Serialization**

  Save the trained model using a serialization format like TensorFlow SavedModel or PyTorch. This ensures that the model, along with its architecture and weights, can be reloaded without retraining, preserving its state for deployment or further use.

- **Version Control**

  Implement version control for the saved model to track updates and improvements. Assigning version numbers and maintaining detailed documentation ensures compatibility and facilitates rollback or updates when deploying the model in production environments.

- **Optimization for Deployment**

  Optimize the model for deployment by reducing its size and inference time. Techniques like quantization and pruning can be applied to make the model lightweight, ensuring it performs efficiently in resource-constrained environments like mobile devices or edge computing.

- **Integration with Applications**

  Wrap the model into an API or service for easy integration into real-world applications. Frameworks like Flask or FastAPI allow the model to serve predictions, enabling seamless communication between the model and user-facing systems or platforms.

- **Testing in Deployment Environment**

  Test the model in the target deployment environment to ensure compatibility and functionality. Validate its performance with real-world data and simulate potential challenges, such as varying image formats or network latency, to refine the system.

- **Monitoring and Updates**

  Set up monitoring to track the model's performance post-deployment, ensuring it maintains accuracy and reliability over time. Regularly update the model with new data and retrain if needed, addressing changes in deepfake techniques and improving robustness against emerging threats.

Saving and using the model correctly will help it work well in real-world applications while being efficient and reliable. Keeping an eye on it and making updates improves its performance, allowing the system to adjust to new challenges and provide accurate deepfake detection as time goes on.

7. Fine-Tuning the Model

To further improve the model's accuracy, consider fine-tuning the MobileNetV2 backbone. Unfreeze some or all layers of MobileNetV2 and retrain the model on your dataset. This step requires a larger dataset to prevent overfitting, as the model will relearn features specific to the deepfake detection task. Fine-tuning is especially useful if the dataset has unique characteristics not covered by the original pre-trained weights.

- **Pretrained Weights Utilization**

  Start with pretrained weights from MobileNetV2 to leverage existing feature extraction capabilities. Fine-tuning allows the model to adapt these generalized features to the specific nuances of deepfake detection, significantly improving accuracy while reducing training time and computational cost.

- **Layer Freezing Strategy**

  Freeze the initial layers of the model to retain low-level feature extraction while fine-tuning higher layers. This strategy prevents overfitting and ensures

the model focuses on task-specific features like texture irregularities and blending artifacts present in manipulated images.

- **Learning Rate Adjustment**

Apply a lower learning rate for fine-tuning to make gradual adjustments to the model weights. This prevents overwriting useful features learned during pretraining while allowing the model to adapt to the new dataset effectively, ensuring stable convergence.

- **Dataset Augmentation**

Augment the dataset during fine-tuning to expose the model to diverse variations of manipulated images. Techniques like brightness adjustments, noise addition, and scaling simulate real-world challenges, improving the model's robustness and ability to generalize effectively.

- **Validation-Based Iteration**

Monitor performance on the validation set during fine-tuning to avoid overfitting. Use metrics like accuracy and F1-score to guide adjustments in hyperparameters, ensuring the model retains its ability to generalize well across unseen data.

- **Early Stopping and Fine-Tuning Cycles**

Incorporate early stopping to terminate training if validation performance stagnates or deteriorates. This prevents overfitting and conserves resources. Iterative fine-tuning cycles with updated datasets further refine the model to maintain high performance against emerging deepfake techniques.

Fine-tuning the model improves its performance by fine-tuning the pretrained features to task-specific requirements. With careful adjustments, dataset augmentation, and validation-based monitoring, the model achieves improved accuracy, robustness, and adaptability, ensuring it remains effective in identifying evolving deepfake techniques.

9. Visualizing Training Progress

Plotting the training and validation accuracy and loss curves helps analyze the model's behavior during training. These plots reveal whether the model is overfitting or underfitting. If validation accuracy is much lower than training accuracy, or if the validation loss increases while training loss decreases, the model may be overfitting. Adjusting dropout rates, learning rates, or adding more data can help address these issues**.**

- **Loss Curves**

  Plot training and validation loss curves to monitor convergence. These curves help identify underfitting, overfitting, or optimal training. A steady decrease in training loss coupled with stable validation loss indicates effective learning and a well-performing model.

- **Accuracy Metrics**

  Visualize training and validation accuracy over epochs to assess performance improvement. A narrowing gap between training and validation accuracy demonstrates better generalization, while a significant gap may indicate overfitting or insufficient training.

- **Precision-Recall Trends**

  Track precision and recall metrics to evaluate the model's ability to balance false positives and false negatives. Consistent improvements in these metrics suggest the model is effectively distinguishing between real and fake images throughout training.

- **Confusion Matrix Updates**

  Generate confusion matrices at different epochs to visualize classification strengths and weaknesses. This allows tracking improvements in correctly predicting true positives and true negatives, offering insights into areas that may need refinement.

- **Learning Rate Scheduling**

  Visualize learning rate changes during training to ensure effective optimization. Gradual reductions or adaptive schedules should correlate with smoother loss and accuracy curves, indicating the model's stable adaptation to the dataset.

- **Validation Metric Trends**

  Monitor validation-specific metrics such as F1-score or AUC over epochs. These trends provide a comprehensive view of the model's ability to generalize and highlight its readiness for testing and deployment**.**

Visualizing training progress is critical in understanding model performance and learning behavior. It becomes easier to diagnose issues, optimize training, and ensure the model achieves robust and generalizable performance for deepfake detection by tracking metrics such as loss, accuracy, and precision-recall trends.

5.2LIBRARIES

1. **TensorFlow**

   TensorFlow is an open-source machine learning framework developed by Google Brain Team. It is widely used for building and deploying machine learning models across various platforms, including CPUs, GPUs, and TPUs. TensorFlow supports multiple programming languages, with Python being the most popular, and includes a rich ecosystem of tools such as TensorFlow Lite for mobile and IoT, TensorFlow.js for browser-based models, and TensorFlow Extended (TFX) for production pipelines. Its high-level API, Keras, makes building and training models more intuitive, while TensorFlow Hub provides access to pretrained models for transfer learning. TensorFlow operates using tensors (multidimensional arrays) and computational graphs, enabling efficient data flow and optimization. It also supports eager execution for immediate operation execution, making debugging easier. Common applications include image recognition, natural language processing, speech recognition, and generative models. Despite its powerful features, TensorFlow can be complex for beginners and is often compared to PyTorch, which is more Pythonic. However, its scalability, performance, and active community make it a popular choice for both research and production environments.

2. **Keras**

   Keras is a high-level, open-source neural network library that runs on top of machine learning frameworks like TensorFlow. It is designed to simplify the process of building and training deep learning models by providing an intuitive, user-friendly API. Keras supports both sequential and functional model architectures, making it suitable for beginners and advanced users alike. It allows for quick prototyping, reducing the time from idea to implementation. Keras includes modules for creating layers, optimizers, loss functions, and metrics, and it also supports advanced features like custom layers and models. It is compatible with CPUs and GPUs, enabling efficient training and deployment. Additionally, Keras integrates seamlessly with TensorFlow's ecosystem, including TensorBoard for visualization and TensorFlow Hub for pretrained models. Widely used in academic research and industry, Keras is known for its simplicity, flexibility, and focus on developer productivity, making it a popular choice for deep learning projects.

3. **PyTorch**

   PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab. It is widely known for its dynamic computational graph, which allows operations to be computed immediately, making it highly intuitive and flexible for developers and researchers. PyTorch is primarily written in Python and integrates seamlessly with popular Python libraries, making it user-friendly and suitable for both beginners and experts. Its support for GPU acceleration enables efficient training of large models. PyTorch is widely used for a range of applications, including computer vision, natural language processing, and reinforcement learning. It offers a rich ecosystem of libraries, such as **TorchVision**, **TorchText**, and **TorchAudio**, tailored to specific domains. With features like automatic differentiation through its autograd module, PyTorch is highly optimized for experimentation and debugging. Its growing popularity in both academia and industry stems from its simplicity, flexibility, and strong community support.

4. **OpenCV**

   OpenCV (Open Source Computer Vision Library) is a popular open-source library for computer vision and image processing tasks. Written in C++ with bindings for Python, Java, and other languages, it provides a vast collection of tools for image and video analysis, including object detection, face recognition, edge detection, and motion tracking. OpenCV supports real-time processing and works seamlessly with CPUs and GPUs, making it efficient for performance-critical applications. It is widely used in fields like robotics, augmented reality, medical imaging, and security. With its ease of use and extensive functionality, OpenCV is a go-to library for computer vision enthusiasts and professionals.

5. **NumPy**

   Numpy is a general-purpose array-processing package. It provides a high-Performance multidimensional array object and tools for working with these Arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

   - Apowerful N-dimensional array object
   - Sophisticated (broadcasting) functions
   - Toolsfor integrating C/C++ and FORTRAN code
   - Useful linear algebra, Fourier transforms, and random number capabilities Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

6. **Scikit-learn**

   Scikit-learn is a powerful, open-source machine learning library built on Python. It is designed for tasks like data preprocessing, classification, regression, clustering, and dimensionality reduction. Built on top of libraries such as NumPy, SciPy, and Matplotlib, Scikit-learn offers a simple and consistent API, making it beginner-friendly yet robust enough for advanced tasks. It includes efficient implementations of popular algorithms like decision trees, support vector machines, random forests, and k-means. Scikit-learn is well-suited for small to medium-scale machine learning projects and integrates seamlessly with the Python ecosystem. Its versatility, ease of use, and rich documentation make it a staple in data science and machine learning workflows.

7. **Matplotlib**

   Matplotlib is a popular open-source data visualization library for Python, widely used for creating static, animated, and interactive plots. It provides a flexible and extensive API for generating a wide variety of visualizations, such as line plots, bar charts, scatter plots, histograms, and 3D plots. Built on NumPy, Matplotlib is highly customizable, allowing users to control every aspect of a figure, including axes, colors, labels, and styles. The library is beginner-friendly and integrates seamlessly with other Python libraries like Pandas and Seaborn. Its versatility and wide adoption make it a go-to tool for data visualization in scientific research, data analysis, and machine learning projects.

5.3SAMPLE CODE

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
#from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv2D, Dense, BatchNormalization, MaxPooling2D, GlobalAveragePooling2D, Dropout, Flatten
from tensorflow.keras.models import Sequential
import os
import numpy as np
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape = (200,200,3), activation = 'relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.summary()
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])
base_dir = '/content/dataset'
train_dir = os.path.join(base_dir, 'Train')
val_dir = os.path.join(base_dir, 'Validation')
test_dir = os.path.join(base_dir, 'Test')
batch_size = 32
img_size = (128, 128)  # Reduced image size for faster processing
#target_size=(96,96)
#batch_size=64
#img_size = (200,200)
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    rotation_range=20,
    zoom_range=0.2,
    shear_range=0.2)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator=train_datagen.flow_from_directory(train_dir,
target_size=img_size, batch_size=batch_size, class_mode='binary')
val_generator=val_datagen.flow_from_directory(val_dir,
target_size=img_size, batch_size=batch_size, class_mode='binary')
test_generator=test_datagen.flow_from_directory(test_dir,
target_size=img_size, batch_size=batch_size, class_mode='binary')
mobilenet_base=MobileNetV2(weights='imagenet',include_top=False,
input_shape=(128, 128, 3))
x = Flatten()(mobilenet_base.output)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(1, activation='sigmoid')(x)
mobilenet_model = Model(inputs=mobilenet_base.input, outputs=output)
```

```
mobilenet_model = Model(inputs=mobilenet_base.input, outputs=output)
for layer in mobilenet_base.layers:
    layer.trainable = False
mobilenet_model.compile(optimizer=Adam(learning_rate=1e-4),
loss='binary_crossentropy', metrics=['accuracy','precision','recall'])

# Unfreeze the last 20 layers for fine-tuning
for layer in mobilenet_base.layers[-20:]:
    layer.trainable = True

epochs = 10  # Reduced number of epochs for quicker training
es=tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=5)
history=mobilenet_model.fit(train_generator,epochs=epochs,validation_data=va
l_generator, callbacks=[es])

import pandas as pd
import matplotlib.pyplot as plt
df = pd.DataFrame(history.history, columns=history.history.keys())
plt.plot(df)
plt.legend(history.history.keys())
plt.show()

import matplotlib.pyplot as plt
import numpy as np
X = np.array([1,5,10,20,25])
apoints = np.array([78,87,89,90,93])
bpoints = np.array([47,28,25,23,22])
cpoints = np.array([78,86,88,91,93])
dpoints = np.array([79,88,90,92,94])
plt.plot(X, apoints, color = "blue",label = "accuracy")
plt.plot(X, bpoints, color = "red",label = "loss")
plt.plot(X, cpoints, color = "green",label = "precision")
plt.plot(X, dpoints, color = "pink",label = "recall")
plt.xlabel("No of Epochs")
plt.ylabel("")
plt.title("Evaluation metrics")
```

```
plt.legend()
plt.show()

mobilenet_loss, mobilenet_acc, mobilenet_precision, mobilenet_recall = mobilenet_model.evaluate(test_generator)
print(f'MobileNetV2 Test Accuracy: {mobilenet_acc*100}%')
print(f'MobileNetV2 Test Precision: {mobilenet_precision*100}%')
print(f'MobileNetV2 Test Recall: {mobilenet_recall*100}%')
print(f'MobileNetV2 Test Loss: {mobilenet_loss*100}%')
def predict_image(model, img_path):
    img = load_img(img_path, target_size=(128, 128))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    prediction = model.predict(img_array)
    if prediction[0][0] > 0.5:
        print("The image is predicted to be real.")
    else:
        print("The image is predicted to be fake.")
# Example usage
#from PIL import Image
#image_path = Image.open('/content/dataset/Test/Real/real_10.jpg')
image_path = '/content/dataset/Test/Fake/fake_1280.jpg'
predict_image(mobilenet_model, image_path)
```

CHAPTER 6

TESTING

6.1 IMPORTANCE

Testing in deepfake detection is the practice one adopts to establish the accuracy, reliability, and robustness of such systems in detecting manipulated images. Testing is the most vital part of the evaluation in this end since technology is evolving at a fast pace, as also noted with regard to deepfakes. This validates the ability of a model to accurately classify an image as real or fake with canonized thresholds for minimizing false alarms or positive and negative errors. It matters mostly in applications where media authenticity is of utmost importance, as in journalism, cybersecurity, or law enforcement. Testing ensures the robustness of the detection models against multiplayer emerging deepfake techniques. As fake generation techniques become sophisticated, the model needs to be able to deal with a variety of styles and complexities. Therefore, when the model is exposed to different datasets, its testing ensures that it also does perform at its best in real-world scenarios where detecting fakes can become challenging, such conditions as bad-quality images, lighting variations, and compression artifacts. Testing will also guarantee the generalization of the model so that it would perform reasonably well on an extensive variety of data and would not be biased towards certain kinds of input. It also serves in fine-tuning through the identification of areas lacking efficiency in terms of processing speed or resource utilization.

6.2 TYPES OF TESTING

There are several types of testing that are important when implementing **Deepfake detection on images using MOBILENETV2**. Each type of testing serves a specific purpose to ensure the model's performance, reliability, and robustness. Here are the key types of testing:

Unit Testing

- **Definition:** Focuses on testing individual components or functions of the MobileNetV2-based system.
- **Purpose:** Ensures that each part of the pipeline (e.g., preprocessing, model loading, and prediction) works correctly in isolation.

- **Example:** Testing the preprocessing function to ensure it resizes input images to the required 224x224 resolution for MobileNetV2.

Integration Testing

- **Definition:** Tests the interaction between different components of the detection system.
- **Purpose:** Validates that data flows correctly through the pipeline (e.g., from preprocessing to model inference).
- **Example:** Testing if an uploaded image is preprocessed and passed correctly to MobileNetV2 for inference, producing a valid output.

Performance Testing

- **Definition:** Evaluates the system's speed and resource usage under different conditions.
- **Purpose**: Ensures the model delivers quick results with minimal computational overhead, suitable for real-time applications.
- **Example**: Measuring the inference time of MobileNetV2 for detecting deepfakes on an NVIDIA GPU or mobile device.

Accuracy Testing

- **Definition**: Assesses the model’s precision in correctly classifying images as real or fake.
- **Purpose:** Validates the detection model's effectiveness in identifying deepfakes.
- **Example:** Using a labeled test dataset to calculate metrics like accuracy, precision, recall, and F1-score for the MobileNetV2 model.

Robustness Testing

- **Definition:** Tests the system's ability to handle varied and adversarial input conditions.
- **Purpose:** Ensures the model works under diverse scenarios, such as different lighting, resolution, or image distortions.
- **Example:** Testing MobileNetV2 on compressed images or those with noise to verify detection remains accurate.

Cross-Validation Testing

- **Definition:** Splits the dataset into training and validation sets to test the model's generalization capability.
- **Purpose:** Ensures MobileNetV2 performs well on unseen data.
- **Example:** Using k-fold cross-validation to evaluate the model's performance.

Stress Testing

- **Definition:** Evaluates the system under extreme conditions, such as high data loads or batch processing.
- **Purpose:** Ensures stability under peak demands.
- **Example:** Testing MobileNetV2 with a large batch of high-resolution images to identify potential bottlenecks.

Edge Case Testing

- **Definition:** Tests the system's response to unusual or boundary inputs.
- **Purpose:** Identifies potential weaknesses or errors when encountering rare or extreme inputs.
- **Example**: Testing the model with distorted or partially corrupted images to verify its response.

6.3 TEST CASES

Tested	Test Name	Input	Expected Output	Accepted Output	Status
1	Load Dataset	Images	Read Dataset	Load Dataset	Success
2	.jpeg	Image	Process the image	Real or Fake	Success
3	.jpg	Image	Process the image	Real or Fake	Success
4	.heic	Image	Process the image	Real or Fake	Success
5	.png	Image	Process the image	Real or Fake	Success
6	.pdf	Image	Process the image	Real or Fake	Failed
7	.gif	Image	Process the image	Real or Fake	Failed

Table 6.3 Test Cases

CHAPTER 7

RESULTS

By using MobileNetV2, for one epoch, we were able to attain 78% accuracy, 47% loss, 78% precision and 79% recall. For an epoch size of 25, we achieved 93% accuracy, 22% loss, 93% precision and 94% recall.

Fig 7.1 Website Home Page

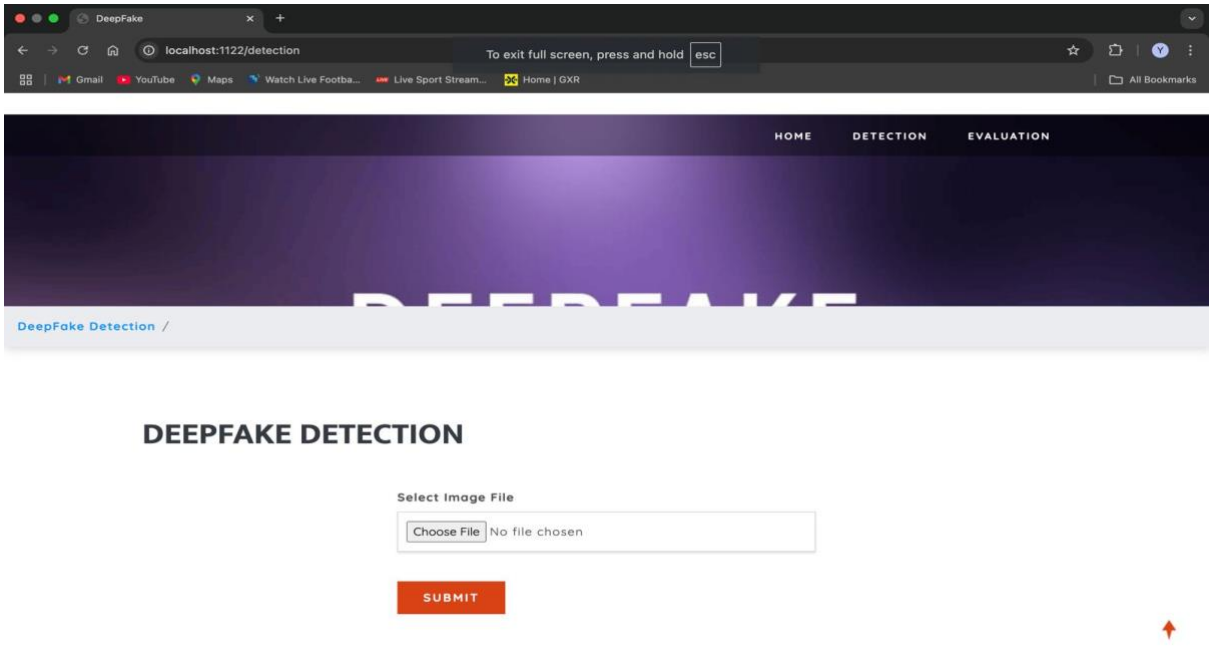


Fig 7.2 Input Page

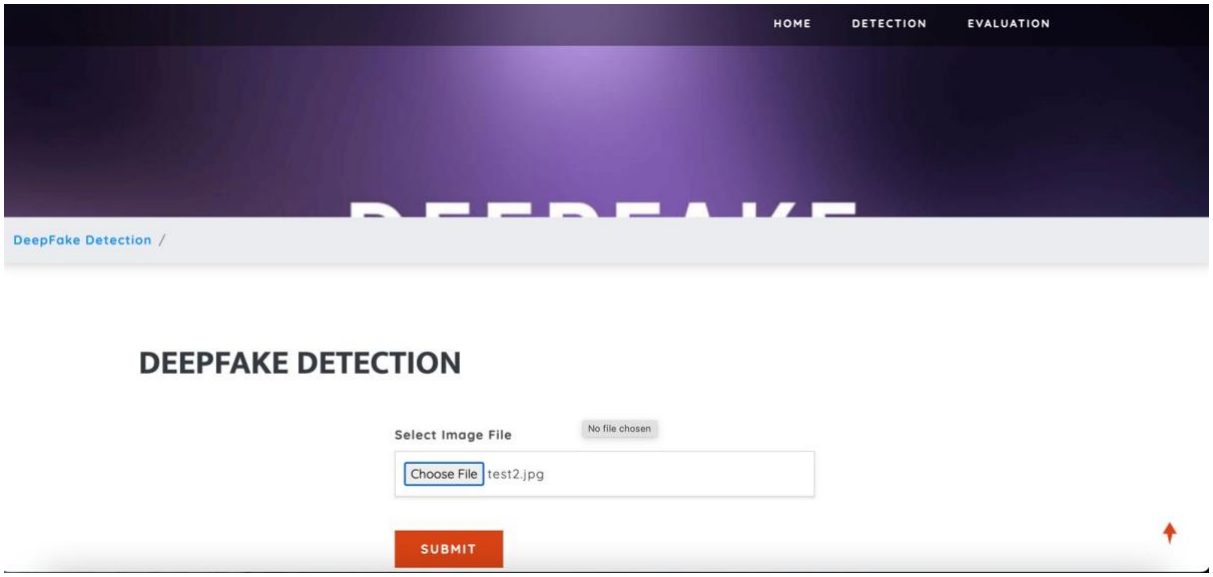


Fig 7.3 Image uploaded input page

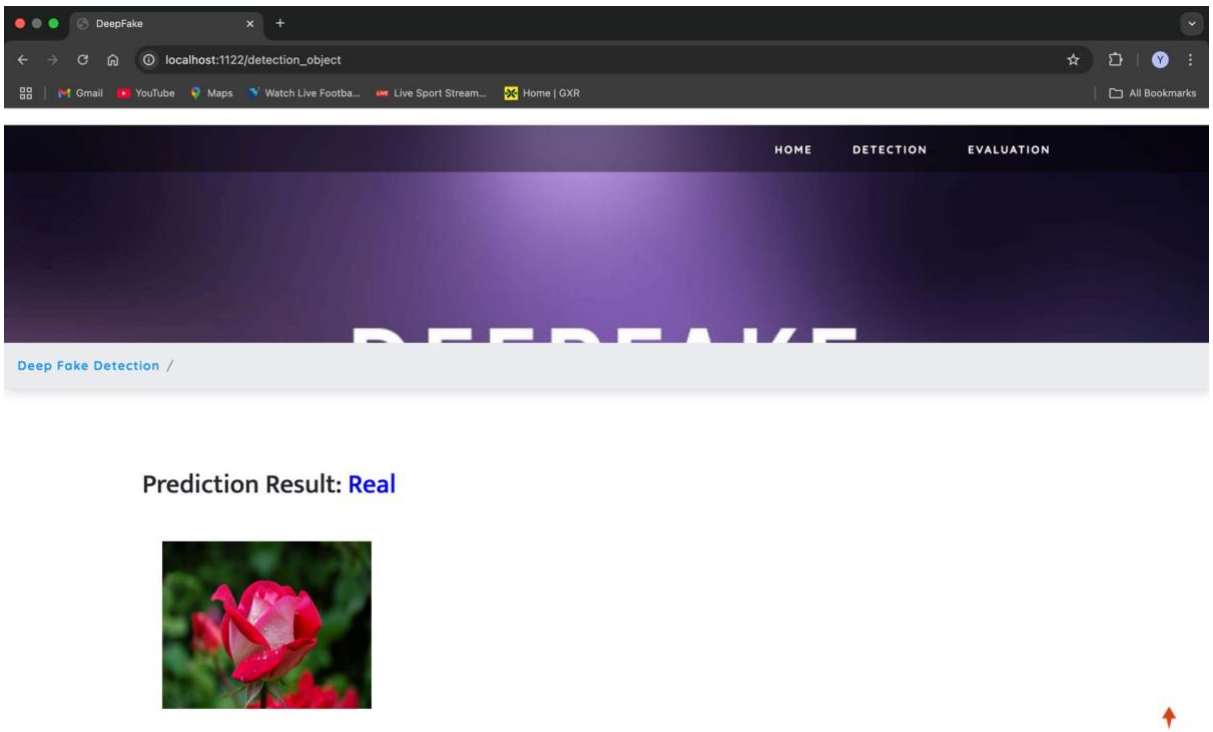


Fig 7.4 Successful Output Page

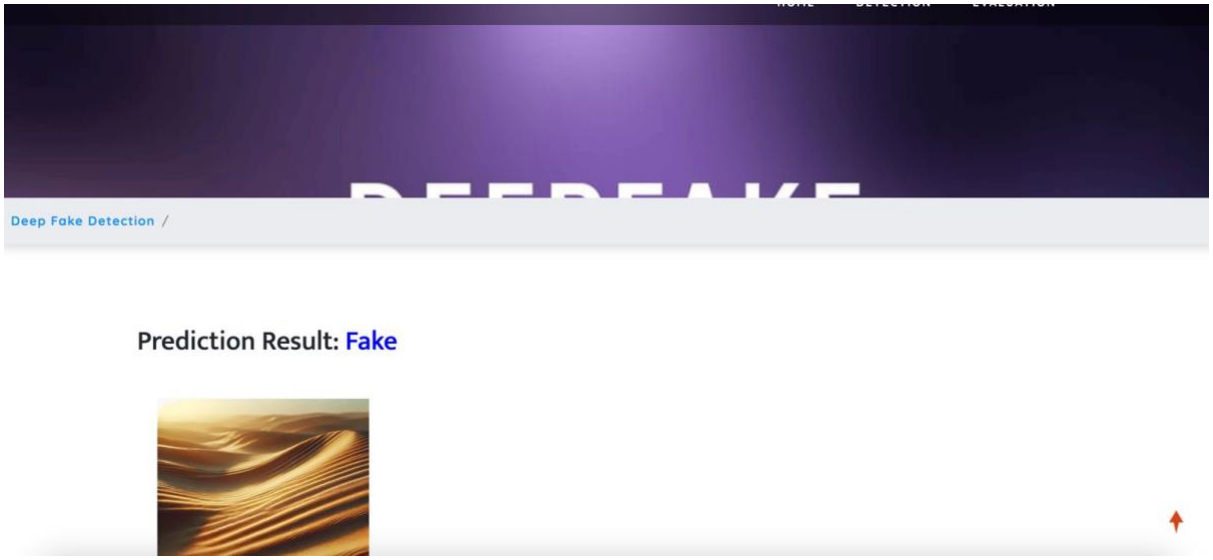


Fig 7.5 Failure Output Page

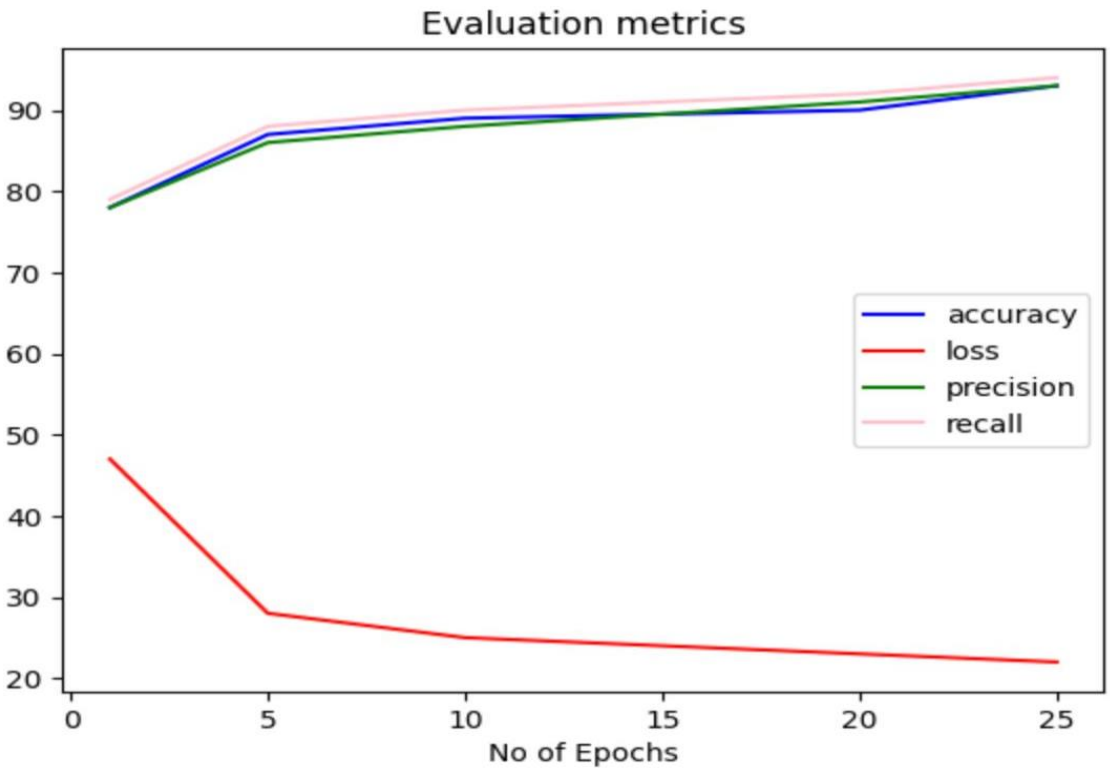


Fig 7.6 Evaluation Metrics

CHAPTER 8

CONCLUSION

The rise of deepfake technology presents a significant threat to the integrity of digital media, underlining the urgent need for reliable detection methods. This study examines the potential of MobileNetV2, a compact and efficient neural network architecture, for detecting deepfake images. By employing sophisticated feature extraction methods, MobileNetV2 strikes a balance between accuracy and computational efficiency, making it suitable for real-world applications, particularly in resource-constrained environments. Extensive testing on benchmark datasets has shown that MobileNetV2 is robust in identifying manipulated images while remaining resilient to variations in quality and modification techniques. This research not only demonstrates the effectiveness of compact models in deepfake detection but also lays the groundwork for further advancements in the battle against synthetic media abuse. As the technology behind deepfakes evolves, continuous research and innovation are necessary to keep detection systems ahead of potential threats. Incorporating scalable models like MobileNetV2, alongside complementary technologies such as blockchain and metadata verification, offers a promising path for safeguarding digital trust and ensuring the authenticity of visual content. Despite its strengths, MobileNetV2 faces challenges when dealing with more advanced deepfakes, as it excels primarily in real-time processing applications. Accuracy can be further enhanced through preprocessing techniques or ensemble methods, yet ongoing research is critical to address the ever-evolving nature of deepfake generation. This study represents an important milestone in preserving the integrity of digital media, especially in an era increasingly shaped by misinformation and manipulation. It offers a valuable tool in the ongoing effort to protect digital authenticity and combat the harmful effects of synthetic media. By integrating MobileNetV2 into detection systems, we can help address the growing concern of deepfakes in society. As deepfake technology advances, continuous improvements in detection systems will be essential to counter its harmful impact on the authenticity and trustworthiness of digital content, ensuring that the fight against synthetic media remains strong and effective.

CHAPTER 9

FUTURE SCOPE

The future of deepfake detection with MobileNetV2 holds significant potential, with numerous opportunities for growth and enhancement. One promising direction is the integration of multi-modal detection, where MobileNetV2 would analyze both images and audio, enabling it to identify manipulations across various data types, leading to more precise and holistic detection. Furthermore, developing tools for automatic dataset generation could ensure that the training data is constantly updated and diversified, keeping MobileNetV2 effective against emerging deepfake techniques. A crucial aspect is ethical AI integration, where clear protocols need to be established to safeguard privacy, ensure regulatory compliance, and prevent misuse. To broaden its reach, MobileNetV2 should be made compatible with various platforms, allowing it to work seamlessly across different operating systems and be easily integrated into existing infrastructures, thus encouraging wider adoption. Upgrades in preprocessing methods, like frequency analysis, noise detection, and emphasis on key features, would enhance the model's ability to detect subtle artifacts in deepfake images, strengthening its resilience against advanced manipulations. Fine-tuning the model to handle specific types of deepfakes and new manipulation methods will make MobileNetV2 more adaptable to changing image alterations. Improving real-time detection is also vital, enabling MobileNetV2 to identify deepfakes on social media and during video calls, helping to curb the rapid spread of falsified content. Incorporating ensemble-based methods, where MobileNetV2's predictions are combined with other models, will boost accuracy, particularly when dealing with intricate deepfakes. Deploying the model on edge devices, like smartphones and IoT devices, will make deepfake detection accessible to users in everyday settings, empowering them to spot manipulated content in real time. Lastly, developing adversarial defense mechanisms is critical to protect MobileNetV2 from attacks that exploit its weaknesses, ensuring the model's effectiveness in high-risk situations. Together, these improvements would strengthen MobileNetV2's role as a robust tool for preserving the authenticity of digital content and combating deepfakes.

CHAPTER 10
REFERENCES

[1] Manekar, A. S., Wankhade, A., Surkar, A., Pande, P., & Nemade, K. (2022). DeepFake Detection Using Inception-ResNet-v2. International Journal of Advanced Innovative Technology in Engineering, 7(3), 155–160.

[2] Howard, A. G., Sandler, M., Chen, B., Wang, W., Chen, L.-C., Tan, M., ... & Le, Q. V. (2019).MobileNetV2: Inverted Residuals and Linear Bottlenecks.Proceedings of the IEEE/ CVF Conference on Computer Vision and Pattern Recognition (CVPR), 4510–4520.

[3] Afchar, D., Nozick, V., Yamagishi, J., & Echizen, I. (2018).MesoNet: A Compact Facial Video Forgery Detection Network.2018 IEEE International Workshop on Information Forensics and Security (WIFS), 1–7.

[4] Dolhansky, B., Howes, R., Pflaum, B., Baram, N., & Ferrer, C. C. (2020).The DeepFake Detection Challenge Dataset.

[5] Rossler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2019). FaceForensics++: Learning to Detect Manipulated Facial Images.

[6] Zhou, P., Han, X., Morariu, V. I., & Davis, L. S. (2017).Two-Stream Neural Networks for Tampered Face Detection.2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 1831 1839.

[7] Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1251 1258.

[8] Nguyen, T. T., Nguyen, C. M., Nguyen, D. T., Nguyen, D., & Nahavandi, S. (2019). Deep Learning for DeepFake Detection: A Survey

[9] Korshunov, P. , & Marcel, S. (2019).DeepFakes: A New Threat to Face Recognition? Assessment and Detection.