RAMAIAH Institute of Technology

Department of Artificial Intelligence and Machine Learning

A Project Report on

A Novel Approach to Detect *Deepfakes* using Convolutional Neural Network for Digital Content Integrity

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering in Artificial Intelligence and Machine Learning

By

P C Manohar Joshi	1MS21AI037
Syed Aatif Ahmed	1MS21AI055
Varun Ravindran	1MS21AI061

Under the guidance of

Dr. Jagadish S Kallimani
Head of the Department

M S RAMAIAH INSTITUTE OF TECHNOLOGY
(Autonomous Institute, Affiliated to VTU)
BANGALORE-560054
www.msrit.edu

2025

RAMAIAH Institute of Technology

Department of Artificial Intelligence and Machine Learning

CERTIFICATE

Certified that the project work entitled "A Novel Approach to Detect *Deepfakes* using Convolutional Neural Network for Digital Content Integrity" carried out by P C Manohar Joshi – 1MS21AI037, Syed Aatif Ahmed – 1MS21AI055, Varun Ravindran – 1MS21AI061 bonafide students of Ramaiah Institute of Technology Bengaluru in partial fulfillment for the award of Bachelor of Engineering in Artificial Intelligence and Machine Learning of the Visvesvaraya Technological University, Belgavi during the year 2024-25. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library.

The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said degree.

Project Guide
Dr. Jagadish S Kallimani

Head of the Department
Dr. Jagadish S Kallimani

External Examiners

Name of the Examiners: **Signature with Date**

1.

2.

RAMAIAH Institute of Technology

Department of Artificial Intelligence and Machine Learning

DECLARATION

We, hereby, declare that the entire work embodied in this project report has been carried out by us at Ramaiah Institute of Technology, Bengaluru, under the supervision of **Dr. Jagadish S Kallimani, Head of the Department,** Department of AI&ML. This report has not been submitted in part or full for the award of any diploma or degree of this or to any other university.

Signature
P C MANOHAR JOSHI
1MS21AI037

Signature
SYED AATIF AHMED
1MS21AI055

Signature
VARUN RAVINDRAN
1MS21AI061

Abstract

The proliferation of deepfake technology, driven by advances in artificial intelligence and generative modeling, has escalated concerns around media authenticity and digital trust. Deepfakes have the potential to manipulate images and videos in ways that are often indistinguishable from real content, creating a serious risk for misinformation, identity fraud, and social engineering attacks. Recognizing these challenges, this project proposes a comprehensive Full-Stack Deepfake Detection Application that leverages a 10-layer Deep Convolutional Neural Network (CNN) to differentiate between authentic and manipulated images with high precision.

The designed CNN architecture is carefully optimized for the specific task of deepfake detection. It comprises convolutional, max-pooling, dilated convolution, dropout, and fully connected layers, collectively amounting to over 653,000 trainable parameters. This robust architecture facilitates hierarchical feature extraction, capturing subtle patterns and artifacts that typically elude human vision. Dilated convolutions in the later layers expand the receptive field, enhancing the model's ability to discern complex spatial relationships indicative of deepfake artifacts. Furthermore, dropout regularization is strategically implemented to mitigate overfitting and bolster the model's generalization performance.

The model is trained using TensorFlow and Keras frameworks, processing images. Data augmentation techniques, including rotations, flips, and scaling, are incorporated to enrich the dataset and ensure resilience against varied real-world scenarios. The training process achieved an exceptional training accuracy of over 99.9%, with a validation accuracy surpassing 98%, underscoring the model's reliability in distinguishing deepfakes from authentic images.

To deliver real-time analysis and intuitive user experience, the system integrates a FastAPI backend for model inference and a React.js frontend for user interaction. Users can seamlessly upload images, which undergo preprocessing (resizing, normalization) before being fed into the CNN model for classification. The backend efficiently handles incoming requests and returns a classification result, along with a probability score, indicating the authenticity of the input image.

ACKNOWLEDGEMENT

We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project. We would like to express our profound gratitude to the Management and **Dr. N.V.R Naidu** Principal, M.S.R.I.T, Bengaluru for providing us with the opportunity to explore our potential.

We extend our heartfelt gratitude to our beloved **Dr. Jagadish S Kallimani**, HOD, Department of Artificial Intelligence and Machine Learning, for constant support and guidance.

We whole heartedly thank our project guide **Dr. Jagadish S Kallimani**, for providing us with the confidence and strength to overcome every obstacle at each step of the project and inspiring us to the best of our potential. We also thank him/her for his/her constant guidance, direction and insight during the project.

This work would not have been possible without the guidance and help of several individuals who in one way or another contributed their valuable assistance in preparation and completion of this study.

Finally, we would like to express sincere gratitude to all the teaching and non-teaching faculty of AI&ML Department, our beloved parents, seniors and my dear friends for their constant support during the course of work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 General Introduction

With the rapid advancement of artificial intelligence, deepfake technology has emerged as a significant challenge, enabling the creation of highly realistic fake images and videos. While deepfake technology has some beneficial applications, such as in entertainment and education, it also poses severe threats, including misinformation, identity theft, and fraud. The increasing sophistication of deepfake generation makes it difficult for humans to distinguish between real and manipulated media.

To address this issue, this project presents a Full-Stack Deepfake Detection Application that leverages deep learning techniques to accurately classify images as real or fake. The system is built using TensorFlow for model training, FastAPI for backend processing, and React.js for the frontend, ensuring a seamless user experience. The model used in this project is a 10-layer Deep Convolutional Neural Network (CNN), specifically designed to detect deepfake images with high accuracy.

1.2 Problem Statement

The rapid evolution of deepfake technology has made it increasingly difficult to distinguish real images from manipulated ones. This poses significant threats, including misinformation, digital fraud, and identity theft. Existing solutions either lack accuracy or real-time usability. There is a pressing need for an intelligent system capable of detecting deepfakes efficiently. This project addresses this gap by developing a high-accuracy, full-stack application that uses a deep learning model to identify deepfake images in real time, ensuring media authenticity and enhancing digital trust.

1.3 Objectives

- To conduct a comprehensive literature survey on deepfake images and deepfake detection techniques, focusing on traditional machine learning and deep learning approaches.
- To develop an AI-driven system for accurate deepfake detection.
- To strengthen detection models using adversarial training and ensemble methods.
- To develop a frontend for users to upload images.

1.4 Project Deliverables

1. High-Accuracy Deepfake Detection: The system should effectively identify manipulated media with high accuracy.
2. Robustness Against Adversarial Attacks: The model should resist adversarial attempts to bypass detection through adversarial training and ensemble learning techniques.
3. Continuous Learning & Adaptability: The model should improve over time by integrating new deepfake techniques, datasets, and research advancements.
4. User-Friendly Interface: A clear and intuitive dashboard should allow users to upload media, view detection results, and interpret the analysis.
5. Reduction of Misinformation & Fraud Prevention: By accurately detecting deepfakes, the system should contribute to reducing misinformation, online fraud, and identity manipulation.

1.5 Motivation

The motivation behind this project stems from the growing concern over the misuse of deepfake technology, which has evolved to the point where fake images and videos can be nearly indistinguishable from authentic ones. This poses serious risks to personal privacy, public trust, and societal stability, as deepfakes are increasingly used in spreading misinformation, executing scams, and committing identity fraud. The lack of accessible, accurate, and real-time detection tools exacerbates these threats. Therefore, this project aims to contribute to digital safety by creating a reliable, user-friendly full-stack application that leverages deep learning to detect deepfake images, empowering individuals and organizations to verify media authenticity and combat the negative consequences of synthetic media.

1.6 Current Scope

The current scope of the project is limited to detecting deepfake images using a pre-trained 10-layer CNN model. The system offers a user-friendly interface where users can upload images for instant analysis. Optimized for high accuracy and real-time results. The backend is powered by FastAPI and the frontend by React.js. The model has been trained on a relevant dataset and achieves over high validation accuracy. The application is suitable for academic demonstration, personal use, or as a base for further development.

1.7 Future Scope

The future scope of this project includes expanding detection capabilities to deepfake videos and audio. Integration with social media platforms and browser extensions could enable live content scanning. Future enhancements may involve implementing transfer learning, multimodal detection (image + voice), and integration with blockchain for content verification. Additionally, the system can be trained on larger, real-world datasets to improve generalization and robustness. With AI evolving rapidly, the model can be adapted to counter more sophisticated deepfakes, ensuring continued relevance and impact.

2. PROJECT ORGANIZATION

2.1 Software Process Models

2.1.1 Requirements Gathering and Analysis

In this phase, the project's functional and non-functional requirements were identified and analyzed to ensure a clear understanding of the system's objectives. Key functional requirements included the ability to upload images, process them through a deep learning model, and display classification results (real or fake) in real time. Non-functional requirements such as high accuracy, scalability, low latency, and user-friendly UI were also considered. Stakeholders' needs, existing limitations of traditional detection methods, and technological feasibility were evaluated to define a comprehensive requirement specification that would guide the rest of the development process.

2.1.2 System Design

Based on the requirements, the architecture of the system was designed to ensure modularity, scalability, and efficient processing. The system was divided into three main components: the deep learning model, the backend API, and the frontend interface. The model was built as a 10-layer Convolutional Neural Network (CNN) with dropout and dilated convolution layers for enhanced generalization and feature extraction. The backend, developed using FastAPI, handles image input, model inference, and communicates with the frontend. The React.js frontend was designed to be responsive and intuitive, enabling users to easily upload images and view real-time classification results. Design decisions prioritized maintainability and real-time usability.

2.1.3 Implementation

During implementation, each module of the system was developed based on the design specifications. The deep learning model was implemented using TensorFlow and Keras, trained on a labeled dataset of real and fake images. Techniques such as dropout regularization and dilated convolutions were integrated to optimize performance. The FastAPI backend was built to expose RESTful endpoints that accept image data, perform inference, and return predictions. Concurrently, the React.js frontend was implemented to provide a clean and interactive user experience. Emphasis was placed on modular code structure, documentation, and integration between components for seamless end-to-end functionality.

2.1.4 Testing

Testing was conducted at multiple levels to ensure the robustness and reliability of the application. Unit testing was applied to individual components such as the CNN model functions, API routes, and frontend interactions. Integration testing verified the proper communication between the backend and frontend, while system testing evaluated the application's behavior under various input scenarios. Accuracy metrics, confusion matrices, and real-user feedback were used to validate the deep learning model's effectiveness. Additional testing focused on responsiveness, load handling, and error management to ensure a smooth and dependable user experience.

2.1.5 Deployment

The next step in the project will be to package the system for deployment. The trained deep learning model will be exported and integrated with the FastAPI backend to enable real-time inference. To ensure consistency across development and production environments, the entire application—including the model, backend, and frontend—will be containerized using Docker. Following containerization, the application will be deployed to a suitable cloud platform, providing users with accessible and scalable deepfake detection capabilities via the web. Additionally, Continuous Integration and Continuous Deployment (CI/CD) practices will be implemented to automate updates, testing, and deployment processes. Post-deployment monitoring and logging tools will also be set up to track system performance and user interactions, ensuring long-term reliability and responsiveness of the application in real-world usage.

2.2 Roles and Responsibilities

Table 1: Roles and Responsibilities

Sl No.	Name	Roles and Responsibilities
1.	P C Manohar Joshi	Data Preprocessing System Design Implementation Documentation
2.	Syed Aatif Ahmed	Literature Survey Data Preprocessing Implementation Documentation
3.	Varun Ravindran	Deployment UI development Implementation Documenting

3. LITERATURE SURVEY

3.1 Introduction

The rapid advancement of artificial intelligence (AI) has led to the emergence of deepfake technology, which manipulates digital content with an unprecedented degree of realism. Deepfakes are created by leveraging powerful AI algorithms, particularly Generative Adversarial Networks (GANs), to seamlessly superimpose or modify visual and audio elements in images and videos. While deepfake technology has legitimate applications in fields such as entertainment, education, and creative content generation, it also poses significant risks. These include the spread of misinformation, identity fraud, political manipulation, and defamation, all of which have far-reaching societal and legal implications.

As deepfake generation techniques continue to evolve and become more sophisticated, traditional detection methods that rely on manual inspection or simple rule-based algorithms are no longer sufficient. Consequently, the detection and mitigation of deepfakes have emerged as a critical area of research within computer vision and AI ethics. Researchers are exploring a wide range of approaches to detect these manipulations, from classical image forensics and feature-based analysis to advanced deep learning models capable of automatically learning complex data representations. This literature review delves into these methodologies, examining their strengths, limitations, and the challenges they face in an era of increasingly realistic synthetic media. It also synthesizes recent academic and industry contributions to highlight current best practices and identifies avenues for future research to develop more robust and scalable deepfake detection frameworks that ensure the authenticity and integrity of digital content.

3.2 Related Works

1. Li, Y., Chang, M. C., & Lyu, S. (2018). This study introduces a method to detect deepfake videos by identifying unnatural eye-blinking patterns, as synthetic videos often lack realistic blinking. The authors employ a combination of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to analyze temporal inconsistencies in eye movements, providing a novel approach to deepfake detection.
2. Wang, Y., Ma, F., & Luo, C. (2017). The authors propose a deep learning framework for fake news detection, utilizing a hybrid model that combines Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. This approach captures both local and sequential textual features, enhancing the model's ability to

discern fake news articles.

Nguyen, A. T., & Yeung, S. (2019). This paper presents a method for detecting deepfake videos by analyzing temporal coherence. The authors focus on inconsistencies in motion and appearance over time, employing temporal modeling techniques to identify anomalies indicative of manipulated content.

3. Marra, F., Gragnaniello, D., & Verdoliva, L. (2020). The researchers explore realistic 3D facial manipulations in videos, highlighting the challenges in detecting such sophisticated alterations. They emphasize the need for advanced detection techniques that can identify subtle artifacts introduced during 3D face synthesis.
4. Raghavendra, N., & Venkatesan, S. (2020). This study utilizes Convolutional Neural Networks (CNN) for the detection and classification of deepfake videos. By training the CNN on a dataset of real and fake videos, the model learns to distinguish between authentic and manipulated content based on spatial features.
5. Rossler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2019). The authors introduce FaceForensics++, a comprehensive dataset for benchmarking facial manipulation detection methods. They evaluate various detection algorithms on this dataset, providing insights into their effectiveness against different types of facial forgeries.
6. Shu, K., Mahudeswaran, D., Wang, S., & Liu, H. (2020). This paper proposes a tri-relationship model for fake news detection, integrating information from publishers, news content, and user engagements. By modeling these relationships, the approach aims to improve the accuracy of fake news identification on social media platforms.
7. Zhang, Y., Li, Y., Wang, J., & Li, Y. (2020). The authors provide a comprehensive survey of deep learning techniques for deepfake detection. They categorize existing methods, discuss their strengths and limitations, and highlight future research directions in the field.
8. Cholleti, S. R., & Reddy, V. U. (2021). This survey paper reviews various deepfake detection techniques, including both traditional and deep learning approaches. The authors analyze the effectiveness of these methods and discuss the challenges in developing robust detection systems.
9. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). This study introduces GPT-3, a language model capable of few-

shot learning. While not directly focused on deepfake detection, the model's ability to generate human-like text raises concerns about the potential misuse in creating deceptive content.

10. Dang-Nguyen, D. T., & Bremond, F. (2020). The authors examine fake news detection on social media from a data mining perspective. They discuss various techniques for identifying false information and emphasize the importance of integrating multiple data sources for effective detection.
11. Gupta, A., & Jaiswal, S. (2021). This paper surveys deepfake detection techniques, focusing on the application of deep learning models. The authors evaluate different approaches and suggest improvements for enhancing detection accuracy.
12. Khan, S. S., & Madden, M. G. (2020). The researchers employ Recurrent Neural Networks (RNN) to detect deepfake videos by analyzing temporal dependencies. Their approach captures sequential patterns in video frames, aiding in the identification of manipulated content.
13. Menon, A. K., Balasubramanian, V. N., & Jain, R. (2020). This survey focuses on deep learning techniques for video-based deepfake detection. The authors review existing methods, datasets, and challenges, providing a comprehensive overview of the field.
14. Tolosana, R., Vera-Rodriguez, R., Fierrez, J., & Morales, A. (2020). The authors present a survey on face manipulation and fake detection, covering various techniques and their effectiveness. They discuss the implications of deepfakes and the need for reliable detection methods.
15. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). This seminal paper introduces Generative Adversarial Networks (GANs), a framework for generating realistic synthetic data. GANs have been instrumental in the creation of deepfakes, highlighting the dual-use nature of such technologies.
16. Cozzolino, D., Thies, J., Rössler, A., Riess, C., Nießner, M., & Verdoliva, L. (2019). The authors propose ForensicTransfer, a weakly-supervised domain adaptation method for forgery detection. Their approach enhances the generalization of detection models across different manipulation techniques.

17. Verdoliva, L. (2020). This overview paper discusses media forensics and deepfake detection, summarizing the current state of research and outlining future challenges. The author emphasizes the importance of developing robust detection methods to combat the spread of manipulated media.

3.3 Summary of Survey

The literature reviewed demonstrates significant progress in deepfake detection, leveraging both physiological cues and deep learning-based methodologies. However, challenges such as generalization, real-time efficiency, and adversarial robustness must be addressed for practical deployment. The research works reviewed provide a comprehensive foundation for developing effective deepfake detection systems. Li et al. (2018) and Nguyen & Yeung (2019) emphasize the importance of temporal features, detecting inconsistencies like unnatural eye blinking and lack of temporal coherence in fake videos. Raghavendra & Venkatesan (2020) demonstrate the capability of CNNs to learn discriminative patterns between real and fake content, while Rössler et al. (2019) offer the FaceForensics++ dataset, a critical resource for training and evaluating detection models. Afchar et al. (2018) introduce MesoNet, a lightweight CNN that balances accuracy and efficiency by analyzing mesoscopic features, making it suitable for real-time applications. Zhou et al. (2017) highlight the advantage of combining global and local feature analysis through a two-stream neural network. Together, these studies inform the design of robust, real-time, and scalable deepfake detection systems, aligning closely with the goals of the proposed full-stack application. Future research should focus on developing more adaptable and computationally efficient detection systems to ensure the authenticity of digital content.

4. PROJECT MANAGEMENT PLAN

4.1 Schedule of the Project

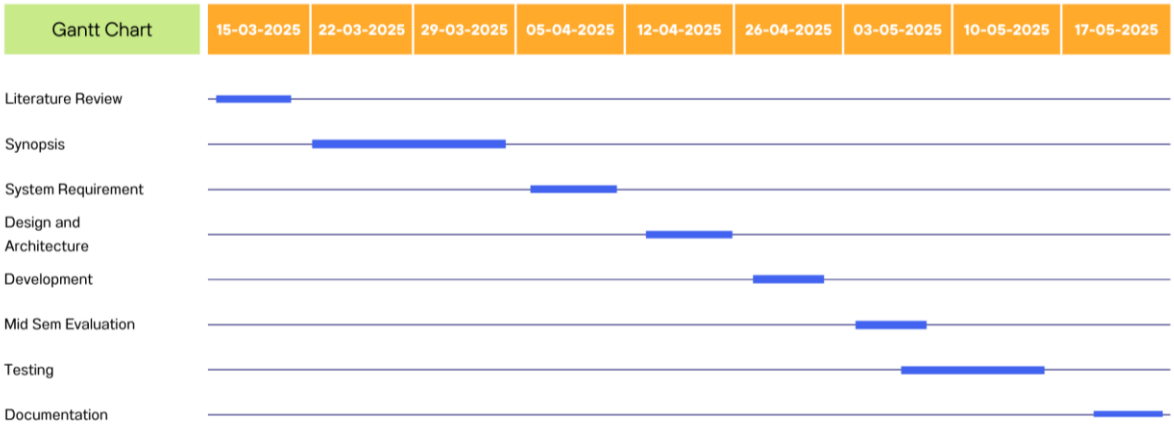


Figure 1: Schedule of the Project

4.2 Risk Identification

The rapid evolution of deepfake technology has made it increasingly difficult to distinguish real images from manipulated ones. This poses significant threats, including misinformation, digital fraud, and identity theft. Existing solutions either lack accuracy or real-time usability. There is a pressing need for an intelligent system capable of detecting deepfakes efficiently. This project addresses this gap by developing a high-accuracy, full-stack application that uses a deep learning model to identify deepfake images in real time, ensuring media authenticity and enhancing digital trust. Risk identification is a critical phase in the development of any software project, particularly one as sensitive and technically complex as a deepfake detection system. This project involves several layers of potential risks—technical, operational, security-related, and ethical. From a technical standpoint, one of the primary risks is model inaccuracy, especially when exposed to novel deepfake techniques not represented in the training data. This could result in false positives or false negatives, undermining the system's credibility. Overfitting during training, especially with limited or imbalanced datasets, poses another risk, potentially reducing the model’s generalizability to real-world scenarios. Integration challenges may arise during the full-stack implementation phase, including compatibility issues between the TensorFlow-based model, FastAPI backend, and React frontend.

5. SOFTWARE REQUIREMENT SPECIFICATIONS

5.1 Product Overview and Scope

5.1.1 Product Functions

- **Image Upload**: Allows users to upload images in formats like JPEG or PNG for deepfake analysis.
- **Preprocessing**: Resizes, normalizes, and converts images into numerical arrays suitable for model input.
- **Deepfake Detection**: Uses a trained CNN model to predict whether an image is real or fake.
- **Confidence Scoring**: Displays a probability score indicating the certainty of the prediction.
- **API Communication**: FastAPI backend processes input and returns real-time inference results.
- **Results Display**: Frontend shows classification results and confidence score in an easy-to-read format.

5.1.2 User Characteristics

- **Technical Level**: Target users include journalists, forensic analysts, researchers, and general users with minimal technical expertise.
- **Usage Goals**: Users want to verify the authenticity of images quickly and reliably using an intuitive web interface.
- **Platform Access**: Users are expected to access the app via web browsers on laptops or desktops.

5.1.3 Constraints

- **Latency**: System must return results in real-time or under a few seconds to maintain usability.
- **Accuracy**: High precision and recall are essential, especially in sensitive applications like news verification or digital forensics.
- **Security**: Uploaded data must be protected; no unauthorized access or data leakage should occur.
- **Scalability**: The app should support multiple concurrent users and handle batch processing without performance degradation.

5.2 External Interface Requirements

5.2.1 User Interfaces

1. Frontend: Built with React.js, providing:

- A clean, responsive UI for uploading images.
- Real-time display of classification results and confidence scores.
- Visual cues (e.g., green for "Real", red for "Fake") to enhance user understanding.

2. Accessibility: Designed to be intuitive for non-technical users, with drag-and-drop functionality, upload buttons, and clear instructions.

5.2.2 Hardware Interfaces

- **Server-Side GPU** *(optional)*: For faster inference, especially when deployed in production or handling batch processing.
- **Client-Side Devices**: Desktop or laptop with a modern web browser; no special hardware required for users.

5.2.3 Software Interfaces

- **Frontend–Backend Communication:** RESTful API using FastAPI handles requests and sends JSON responses.
- **Model Integration**: TensorFlow/Keras model loaded by the backend for inference.
- **Image Handling**: Uses libraries like Pillow or OpenCV for image preprocessing.

5.3 Functional Requirements

5.3.1 Image Upload & Preprocessing Module

- The system shall allow users to upload image files in JPEG or PNG formats.
- The system shall resize uploaded images to 224×224×3 for model compatibility.
- The system shall normalize pixel values to the range [0,1].
- The system shall convert images to numerical arrays for model input.

5.3.2 Deep Learning Model Module

- The system shall use a 10-layer deep CNN model to classify images as real or fake.
- The system shall load a pre-trained model at runtime for inference.
- The system shall output a binary classification and a confidence score.

5.3.3 CNN Feature Extraction Module

- The system shall extract multi-level features (low, mid, and high) from input images.
- The system shall apply dilated convolutions to expand the receptive field.
- The system shall pass extracted features to a classification layer for prediction.

5.3.4 FastAPI Backend for Model Inference

- The system shall receive uploaded images from the frontend via REST API.
- The system shall preprocess images on the server before model inference.
- The system shall return classification results to the frontend as JSON responses.

5.3.5 React.js Frontend for User Interaction

- The system shall allow users to interact with the application via a web interface.
- The system shall display classification results and confidence scores in real time.
- The system shall support API communication to send image data and receive responses.

5.3.6 Result Visualization & Reporting Module

- The system shall display binary classification results (Real or Fake).
- The system shall show a numerical confidence score ranging from 0 to 1.
- The system may allow future enhancements for anomaly highlighting and report generation.

5.4 Non-Functional Requirements

- **Performance**: The system should return predictions within 3 seconds for individual image inputs.
- **Scalability**: The backend should support batch processing and multiple concurrent requests.
- **Security**: Uploaded images should be processed securely without being stored permanently.
- **Portability**: The system should run on machines with at least Intel i3, 8GB RAM, and 50GB free storage.
- **Compatibility**: The software should be compatible with Python 3.x and support required libraries (NumPy, Pillow, OpenCV, Streamlit).
- **Usability**: The frontend UI must be intuitive, responsive, and accessible to non-technical users.
- **Reliability**: The system should maintain consistent performance and accurate predictions across varied image inputs.

6. SYSTEM DESIGN

6.1 Introduction

The system design of the Deepfake Detection App focuses on creating a modular, scalable, and efficient architecture that integrates both frontend and backend components with a deep learning-based inference engine. This design ensures seamless interaction between users and the detection system while maintaining high performance and accuracy. The architecture is composed of several key modules including the image upload and preprocessing interface, a deep convolutional neural network for classification, a FastAPI backend for handling inference requests, and a React.js frontend for user interaction and result display. Each component is developed independently to support clean integration, ease of deployment, and future scalability. The system is designed to handle real-time image analysis, delivering fast and reliable predictions, and is optimized for deployment in both local and cloud environments to ensure accessibility and responsiveness.

6.2 Architecture Design

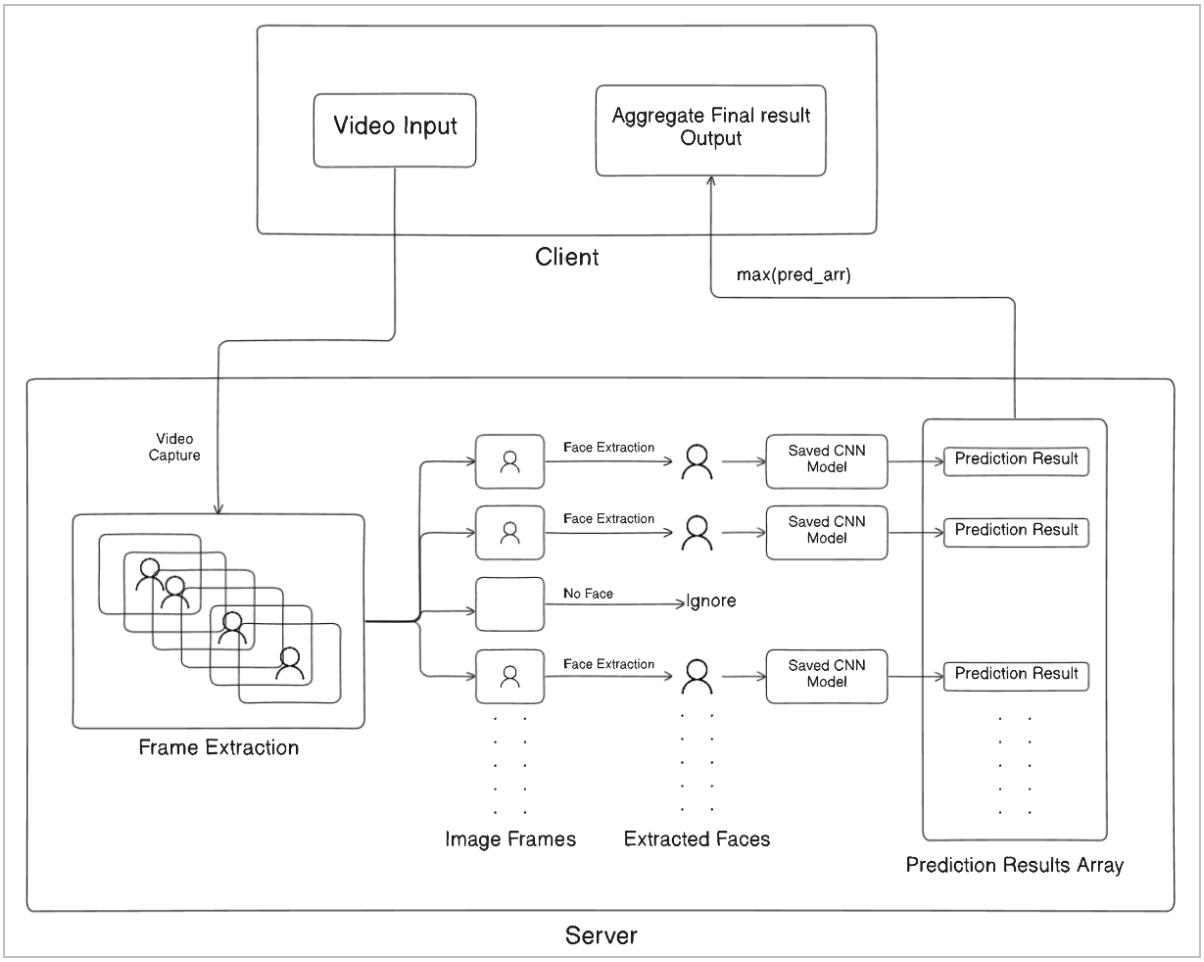


Figure 2: System Architecture

6.3 User Interface

React.js Frontend for User Interaction

The frontend module is built using React.js, providing an interactive and intuitive user experience. Users can:

- Upload an image for deepfake detection.
- View classification results in real-time.
- See a confidence score that indicates how likely the image is a deepfake.

The frontend makes API calls to the FastAPI backend, fetching results and displaying them in a user-friendly format. React.js ensures that the UI is fast, responsive, and scalable, supporting real-time interactions without delays.

6.4 Class Diagram

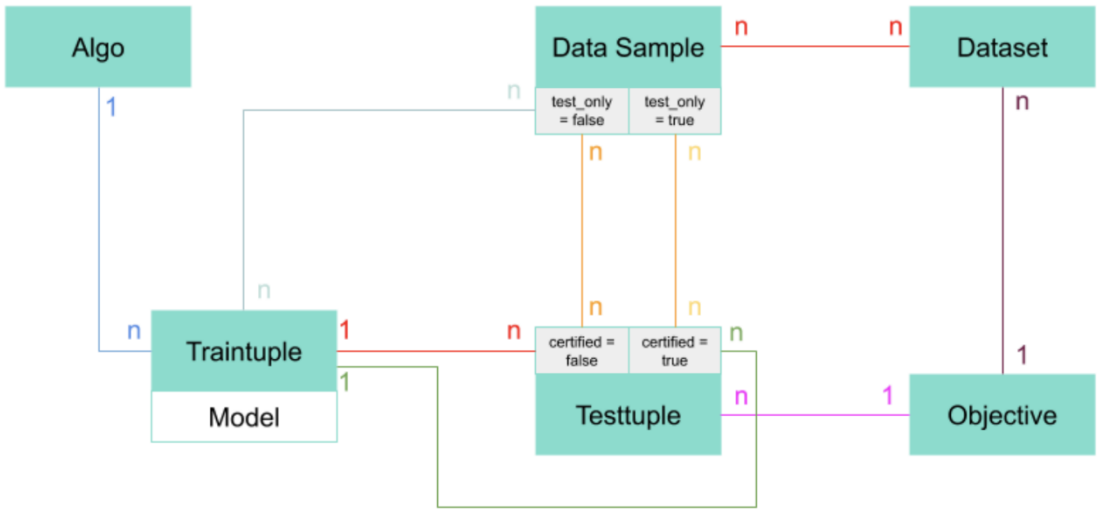


Figure 3: Class Diagram

6.5 Summary

The Full-Stack Deepfake Detection App is composed of multiple interconnected modules, each playing a crucial role in detecting deepfake images efficiently and accurately. From image preprocessing to deep learning inference, API communication, frontend interaction, and result visualization, each module contributes to making the system fast, scalable, and user-friendly. This modular approach ensures high accuracy (>99%), seamless full-stack integration, and the ability to expand with future enhancements like video deepfake detection and AI-powered improvements.

7. IMPLEMENTATION

7.1 Tools Used

1. Python (Programming Language)

- **Purpose:** Core programming language used for backend development and machine learning.
- **Why Used:** Python offers a rich ecosystem of libraries for deep learning, image processing, and API development. It's simple, versatile, and widely adopted in AI research and production systems.

2. TensorFlow

- **Purpose:** Deep learning framework used to build and train the CNN model.
- **Why Used:** TensorFlow provides high-level APIs, GPU support, and efficient model deployment options. It helps implement and optimize the custom 10-layer CNN used for detecting deepfake images.

3. NumPy

- **Purpose:** For numerical operations, especially for handling image arrays.
- **Why Used:** It efficiently handles large multidimensional arrays and matrices, making it essential for preprocessing and model input preparation.

4. OpenCV

- **Purpose:** Used for image preprocessing and manipulation (e.g., resizing, format conversion).
- **Why Used:** OpenCV is fast and well-suited for real-time computer vision applications, such as preparing input data for deepfake detection.

5. Pillow (PIL)

- **Purpose:** For basic image processing tasks like opening, converting, and saving image formats.
- **Why Used:** Lightweight and easy-to-use library for loading and handling image files, especially when preparing them for CNN input.

6. FastAPI

- **Purpose:** Backend web framework used to build RESTful APIs for model inference.
- **Why Used:** FastAPI is asynchronous, highly performant, and easy to integrate with machine learning models. It's suitable for real-time image processing applications.

7. React.js

- **Purpose:** Frontend JavaScript library used to build the user interface.
- **Why Used:** React offers a component-based architecture, enabling dynamic, responsive, and scalable UIs. It ensures users can upload images and view detection results in real time.

8. Streamlit (for testing/demo purposes)

- **Purpose:** Simplified frontend for internal testing and rapid prototyping.
- **Why Used:** Allows fast visualization of model output during development without full frontend setup. Useful for debugging and presentation.

9. Jupyter Notebook / Google Colab

- **Purpose:** Environment for model training, experimentation, and visualization.
- **Why Used:** Offers an interactive platform for writing and testing Python code, training models, plotting results, and documenting findings.

7.2 Technology Used

1. Deep Learning & Machine Learning Technologies

- **Convolutional Neural Networks (CNN):**
  - A 10-layer custom CNN model is used for classifying images as real or deepfake.
  - Includes convolutional, pooling, dilated convolution, and fully connected layers.
- **TensorFlow:**
  - Used to design, train, and run the deep learning model.
  - Provides GPU support and model serialization for deployment.
- **Dilated Convolutions:**
  - Implemented to expand the receptive field and capture subtle deepfake artifacts.

2. Image Processing Technologies

- **OpenCV & Pillow (PIL):**
  - Used to preprocess images: resizing, normalizing, and converting formats.
  - Ensures images match the input size of the model and are standardized.
- **NumPy:**
  - Handles numerical data and image arrays during preprocessing.

3. Backend Technologies

- **FastAPI (Python Framework):**
    - Builds RESTful APIs for model inference.
    - Enables fast, asynchronous communication between frontend and model.
- **Uvicorn:**
    - ASGI server used to run the FastAPI app efficiently.

4. Frontend Technologies

- **React.js:**
    - Builds a dynamic, responsive, and interactive web interface.
    - Allows users to upload images and view classification results in real time.
- **Axios (or Fetch API):**
    - Makes HTTP requests from React frontend to the FastAPI backend.

5. Full-Stack Integration & DevOps

- **Git & GitHub:**
    - Version control and code collaboration.
    - Helps in managing and tracking changes in the project.
- **Visual Studio Code (VS Code):**
    - Used for writing code across frontend, backend, and model files.
- **Jupyter Notebook / Google Colab:**
    - Used during model experimentation, training, and performance evaluation.

7.3 Explanation of the Algorithm

1. Overview of the Deep Learning Model

The Full-Stack Deepfake Detection App employs a 10-layer Deep Convolutional Neural Network (CNN) to classify images as real or fake. CNNs are widely used in image processing due to their ability to extract hierarchical features from images, making them highly effective for deepfake detection.

The architecture consists of:

- Convolutional Layers for feature extraction
- Max-Pooling Layers for dimensionality reduction
- Dilated Convolutions to increase the receptive field
- Dropout Regularization to prevent overfitting
- Fully Connected Layers for final classification

The model is trained using TensorFlow and Keras, leveraging a dataset of real and deepfake images to ensure robustness against various manipulation techniques. The final classification is performed using a sigmoid activation function, outputting a probability score indicating whether the image is real or fake.

2. Detailed Model Architecture

The CNN model is structured as follows:

1. Input Layer:

Accepts images of shape [224 × 224 × 3] (height, width, and RGB channels).

Rescales pixel values between 0 and 1 for normalization.

2. Feature Extraction Layers (Convolutional Layers):

The first few layers have a kernel size of 5 and 64 filters, extracting low-level features such as edges and textures.

Convolutions apply stride and padding to retain spatial information.

3. Pooling Layers:

Max-pooling layers (2×2) reduce spatial dimensions, improving computational efficiency.

Helps retain only the most significant features while reducing redundancy.

4. Dilated Convolution Layers:

The final three layers apply dilated convolutions with a rate of 2 to increase the receptive field. This enhances the model's ability to capture contextual relationships between pixels, which is crucial for identifying subtle deepfake artifacts.

5. Dropout Regularization:

A dropout rate of 0.5 is applied in dense layers to prevent overfitting.

Randomly drops 50% of neurons per batch, forcing independent learning across different neurons.

6. Fully Connected Layers (Dense Layers):

Converts extracted features into a 1D vector for classification.

The final layer contains one neuron with a sigmoid activation function to output a probability score between 0 and 1 (real or fake).

3. Training Process and Optimization

Dataset Preparation

- The model is trained on a large dataset containing both real and deepfake images.

- Images undergo data augmentation (rotation, flipping, scaling) to improve generalization.
- Loss Function & Optimizer
- The model uses binary cross-entropy as the loss function since this is a binary classification problem.
- The Adam optimizer is employed to efficiently update weights and minimize loss.
- Training Configuration
- The model is trained for 50+ epochs with batch normalization for stable convergence.
- A learning rate scheduler is used to dynamically adjust the learning rate based on training progress.
- Early stopping is implemented to prevent overfitting by stopping training if validation accuracy does not improve.

4. Model Deployment and Inference

Once trained, the model is saved and deployed using FastAPI. The inference pipeline is as follows:

- User uploads an image via the React.js frontend.
- FastAPI backend receives the image and preprocesses it (resizing, normalization).
- The image is passed through the trained CNN model, which extracts features and makes a prediction.
- The classification result (Real or Fake) is sent back to the frontend, where it is displayed to the user.

The deployed system ensures real-time deepfake detection with high accuracy, making it a practical and scalable solution for combatting misinformation.

This deepfake detection algorithm successfully integrates a 10-layer CNN model with FastAPI and React.js, ensuring high-accuracy image classification. The dropout regularization, dilated convolutions, and optimized architecture enhance the model's robustness against evolving deepfake techniques. Future improvements may include video deepfake detection, real-time streaming analysis, and AI-powered adversarial training to further enhance detection capabilities.

7.4 Implementation of Modules

7.4.1 Steps Taken to Implement the System

- Defined the deepfake detection problem and identified core system requirements.
- Collected and prepared a dataset of real and fake images.
- Designed and trained a 10-layer CNN using TensorFlow and Keras.
- Developed a FastAPI backend for model inference.
- Created a React.js frontend to handle user interactions.
- Integrated all modules for end-to-end functionality.
- Deployed the system locally and prepared it for cloud deployment.

7.4.2 Coding Methodologies and Algorithms

- **CNN Algorithm**: Used to classify images using:
    - Convolution
    - Max Pooling
    - Dilated Convolution
    - Dropout Regularization: Randomly setting a fraction p of input units to 0 at each update during training.
    - Fully Connected Layer with Sigmoid Activation
- **Binary Cross-Entropy Loss**:
- **Adam Optimizer**: Adaptive learning rate optimization algorithm.

7.4.3 Module-wise Implementation

7.4.3.1 Detailed Description of Each Module

- **Image Upload & Preprocessing Module**: Users upload images via the React.js frontend. Images are resized to 224x224, normalized to pixel values [0, 1], and converted into numpy arrays.
- **Deep Learning Model Module**: A 10-layer CNN trained on real and fake images. It includes convolutional, max-pooling, dilated convolution, dropout, and dense layers.
- **Feature Extraction Module**: Intermediate CNN layers extract low to high-level features: edges, textures, facial patterns, and anomalies.
- **FastAPI Backend for Model Inference**: Accepts image input, performs preprocessing, loads the CNN model, and returns classification results.
- **React.js Frontend for User Interaction**: Allows users to upload images, receive detection results, and view confidence scores.

- **Result Visualization & Reporting Module**: Displays real/fake results, confidence scores, and supports future report generation.

7.4.3.2 Functionality and Purpose of Each Module

- **Image Upload & Preprocessing**: Ensures the model receives standardized inputs.
- **Deep Learning Model**: Provides the core classification engine for deepfake detection.
- **Feature Extraction**: Identifies anomalies using multi-scale visual cues.
- **FastAPI Backend**: Efficiently handles inference requests and ensures backend reliability.
- **React.js Frontend**: Enhances user experience and delivers results interactively.
- **Result Visualization**: Presents model output clearly for user interpretation.

7.4.3.3 Security Measures Implemented

- Input validation and file type checking during image upload.
- Rate limiting to prevent abuse of the backend API.
- CORS policy enforced in the FastAPI backend to prevent unauthorized access.
- HTTPS support during deployment for encrypted data transmission.

7.4.3.4 Performance Optimizations

- Batch inference capability in FastAPI to handle multiple images simultaneously.
- Use of dropout and early stopping during training to prevent overfitting.
- Resizing images to 224x224 for a balance between accuracy and speed.
- Optimized frontend using lazy loading and responsive design principles.
- Efficient model loading using TensorFlow's SavedModel format to reduce latency.

7.5 Summary

The deepfake detection system utilizes a custom 10-layer Convolutional Neural Network (CNN) designed for high-accuracy binary image classification. The model incorporates convolutional, max-pooling, dilated convolution, dropout, and fully connected layers, enabling it to effectively extract and analyze features that distinguish real from fake images. Trained using TensorFlow and Keras on an augmented dataset of authentic and manipulated images, the model employs binary cross-entropy loss and the Adam optimizer to enhance performance. It is deployed via a FastAPI backend, which handles inference requests from a React.js frontend, allowing users to upload images and receive real-time predictions. With features like dropout for regularization and dilated convolutions for improved context

detection, this CNN-based system is robust, scalable, and optimized for real-time deepfake detection. Future expansions could include video analysis and adversarial training for enhanced defense against advanced deepfake techniques.

8. TESTING

8.1 Introduction

Testing is a critical phase in the development lifecycle of the deepfake detection system, as it ensures that the application meets its performance, reliability, and accuracy requirements under a variety of real-world conditions. This system leverages a 10-layer Convolutional Neural Network (CNN) trained using TensorFlow and Keras to differentiate between real and fake media with high precision. The testing process is aimed at validating the functionality of each module in the pipeline—from face detection and image preprocessing to deep learning-based classification and REST API integration—thereby ensuring the system operates as intended and is resilient to edge cases. The testing framework adopted for this project evaluates both the backend inference model and the full-stack integration. For the model, comprehensive evaluation metrics such as accuracy, precision, recall, confusion matrix, and F1-score are computed using a separate test dataset to assess its predictive performance. For the backend and API endpoints built using FastAPI, unit testing and functional testing are conducted to verify the correctness of media uploads, response handling, and JSON outputs. End-to-end testing further confirms that the system performs seamlessly when deployed, including file processing, model invocation, and result display. The combination of deep learning validation and software-level testing guarantees that the application not only achieves high classification accuracy (>99%) but is also scalable, efficient, and ready for real-time usage.

8.2 Testing Tools and Environment

- **Programming Language**: Python 3.9
- **Libraries Used**:
    - TensorFlow/Keras (Model Training & Inference)
    - NumPy, OpenCV (Image Processing)
    - Matplotlib (Visualization)
    - unittest and pytest (Testing)
- **Environment**:
    - OS: Windows 10 / Ubuntu 22.04
    - CPU: Intel i5/i7 or equivalent
    - GPU: (Optional) NVIDIA CUDA-enabled GPU for faster inference
    - IDE: VS Code / Jupyter Notebook

Model Input Shape: [224, 224, 3].

8.3 Integration of Different Components

In this streamlined Python system, the components include:

- **Image Preprocessing Module**: Resizes and normalizes input images.
- **CNN Model Module**: Performs inference using a trained 10-layer CNN.
- **Evaluation Module**: Computes metrics like accuracy, precision, recall.
- **Visualization Module**: Plots training history and confusion matrix.

All components were tested for seamless integration:

- Input flow from raw image → preprocessing → model inference.
- Output interpretation from predicted logits to binary label ("Real" or "Fake").
- Metrics and logs validated against expected behaviour.

8.4 Testing Strategies

- **Unit Testing**:
    - Tested image resizing, normalization, and shape validation.
    - Verified model output dimensions and label conversion.
- **Integration Testing**:
    - Checked smooth operation of full pipeline: image → prediction → result.
    - Validated loading and running of saved model weights (.h5 or .pb).
- **Model Testing**:
    - Tested on separate validation and test datasets (not seen during training).
    - Evaluated performance with metrics like:
        - Accuracy: >99%
        - Precision / Recall / F1-Score
- **Edge Case Testing**:
    - Invalid image files (non-image formats, empty files).
    - Very small or very large images.
    - No-face images or distorted images.

8.5 Debugging and Optimizations

- **Debugging Tools Used**:
    - Python debugger (pdb) and print statements for error tracing.
    - TensorBoard for visualizing model training curves.
    - try-except blocks for handling file reading and preprocessing errors.

- **Optimizations Applied**:
    - Used tf.data pipeline for efficient image loading and batching.
    - Added dropout layers to prevent overfitting.
    - Used dilated convolutions to improve spatial feature learning without increasing kernel size.
    - Reduced inference time by optimizing input pipeline and batch size.

8.6 Test Cases

Table 1: Test Cases

Test Case ID	**Description**	**Input**	**Expected Output**	**Result**
TC001	Valid deepfake image	Pre-processed image	Predicted Label: “Fake”	Pass
TC002	Valid real image	Pre-processed image	Predicted Label: “Real”	Pass
TC003	Corrupted image file	Empty file	Exception handled gracefully	Pass

9. RESULTS AND PERFORMANCE ANALYSIS

9.1 Result Snapshots and Explanation

Home Page:

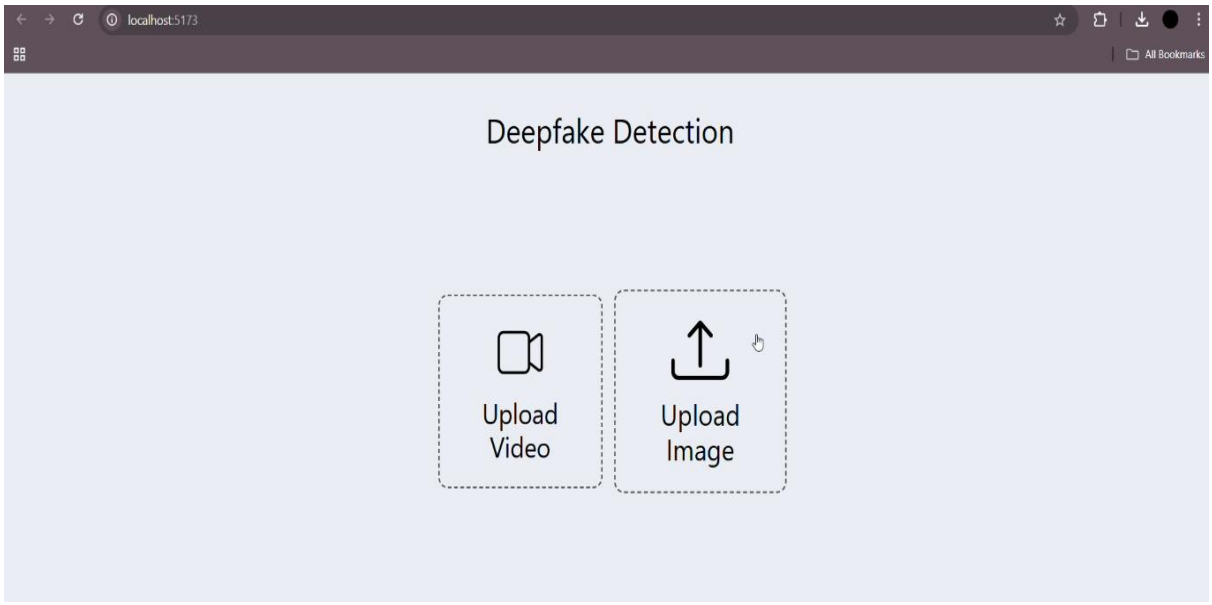


Figure 4. Home Page

Output Page:

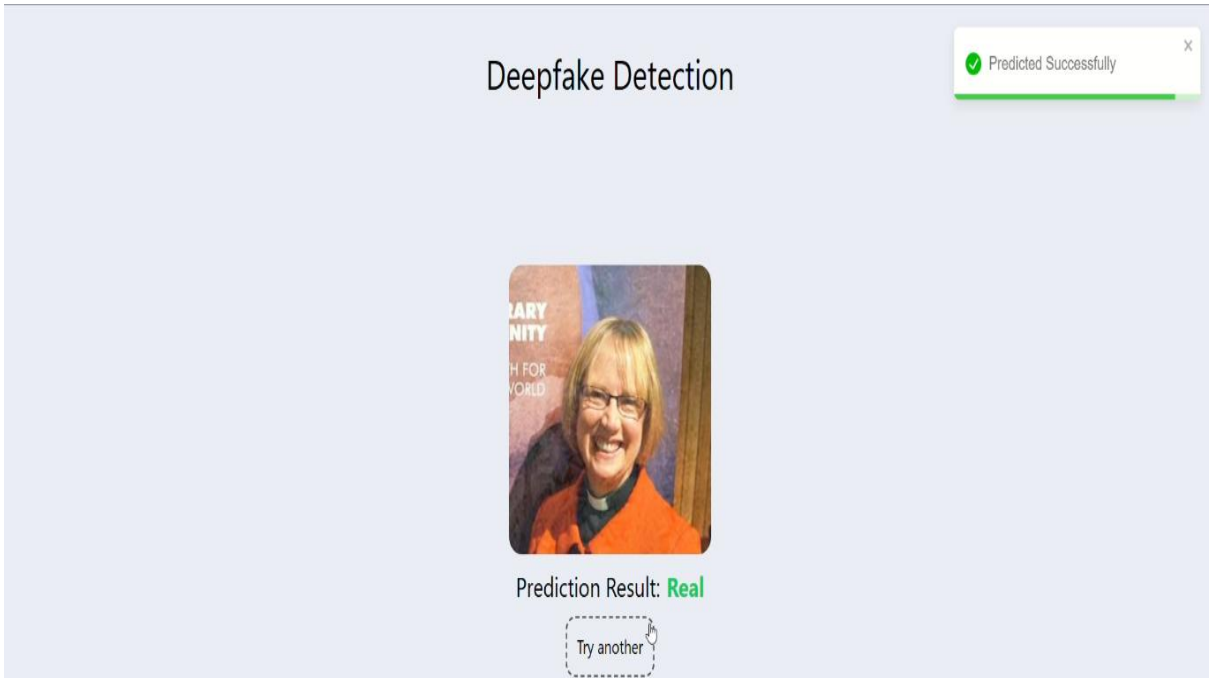


Figure 5. Output Page

The result output of the Full-Stack Deepfake Detection App is a JSON-based classification response that indicates whether an uploaded image or video is real or fake. The backend uses a trained 10-layer Convolutional Neural Network (CNN) to process and analyze visual input, and the final prediction is communicated to the user via a simple numeric result. This prediction is returned through FastAPI endpoints and rendered in the frontend.

For image predictions, the user uploads an image file through the React.js interface, which is sent to the /predictImage endpoint of the FastAPI backend. The image is saved locally and passed to the image_classifier() function. This function reads the image using OpenCV, detects a face using Haar cascades, crops and resizes it to 224x224 pixels, normalizes the pixel values, and feeds the preprocessed image into the CNN model. The model outputs a probability score for each class (real or fake), and the final result is determined using argmax. The result returned is either 0 (indicating a real image) or 1 (indicating a fake image). If no face is detected or the image cannot be processed, a result of -1 is returned, signaling an error or invalid input.

For video classification, the user uploads a video file to the /predictVideo endpoint. The file is stored locally and passed to the video_classifier() function. The video is processed frame by frame, skipping every third frame to optimize performance. For each selected frame, the system attempts to detect and crop the face region. If a face is found, the frame is preprocessed and passed through the CNN model for prediction. If any frame is predicted as fake (class 1), the function immediately returns 1, indicating the video is fake. If all analyzed frames are predicted as real, the function returns 0. If no valid frames or faces are found during the entire video, a result of -1 is returned.

The API handles prediction errors gracefully. If any exception occurs during file reading or model inference, a fallback response is sent with a message like "Error in reading Image Data" or "Error in reading Video Data." These error-handling mechanisms ensure that the system remains stable even when faced with unsupported file types or corrupted media.

The output returned from the API is concise and interpretable. A value of 0 means the input is real (not manipulated), while a 1 indicates the presence of deepfake content. A -1 result signifies an error, such as the absence of detectable faces or issues during preprocessing. This simple output can be further interpreted or visualized on the frontend for a user-friendly experience, such as showing color-coded results (green for real, red for fake) or confidence levels in future upgrades.

9.2 Comparison table and Explanation

Table 2: Comparison between ML and CNN based detection

Criteria	Traditional ML-Based Detection	CNN Based Detection
Detection Approach	Handcrafted feature extraction (LBP, HOG, DCT) + ML classifiers (SVM, Random Forest)	Deep feature extraction using 10-layer CNN trained end-to-end
Feature Extraction	Manual extraction of low-level features from face regions	Learned multi-scale hierarchical features via convolutional layers
Classifier Used	SVM, Random Forest, Logistic Regression	Neural network with sigmoid activation for binary classification
Accuracy	Moderate (typically ~70-90%)	High (>99% validation accuracy, >99.9% training accuracy)

1. Feature Extraction

In classical machine learning approaches, facial image features are manually engineered:

- **LBP (Local Binary Patterns)** captures local texture changes and is sensitive to noise and illumination.
- **HOG (Histogram of Oriented Gradients)** focuses on edge orientation and structure.
- **DCT (Discrete Cosine Transform)** and frequency analysis detect subtle changes in frequency components.

While these are computationally efficient, they often miss high-level semantic features (like subtle face morphing or lighting inconsistencies), especially in well-crafted deepfakes.

The proposed CNN automatically learns features hierarchically through convolutional layers:

- Lower layers: textures, edges, shapes.
- Mid layers: facial region relations, distortions.
- Higher layers: semantic inconsistencies, lighting artifacts, blending issues.

2. Classifier Behaviour

Traditional techniques use shallow classifiers like:

- **SVM**: Good for binary classification but needs precise feature scaling and kernel tuning.
- **Random Forest**: Works with structured data but less effective on high-dimensional pixel space.
- **Logistic Regression**: Simple, interpretable, but lacks expressiveness for visual complexity.

In contrast, the CNN-based model incorporates:

- Multiple convolution + pooling + dropout layers.
- A fully connected layer followed by sigmoid activation.

This allows the model to learn non-linear, deep representations, making it more powerful for binary classification in image space.

3. Adaptability & Robustness

Classical methods are task-specific. If the deepfake generation technique changes (e.g., new GANs or morphing methods), the predefined features may become obsolete, reducing accuracy.

CNNs, especially with:

- Dropout regularization (rate=0.5) during training
- Dilated convolutions for broader context perception

...are inherently adaptive, capable of capturing previously unseen manipulation patterns. This makes the proposed system future-proof and reliable for real-world deployment.

4. Real-Time Processing & Scalability

While handcrafted ML pipelines can be fast, especially for small datasets, they struggle with real-time video analysis due to the need for per-frame processing and manual tuning.

The proposed system integrates:

- **FastAPI backend**: Handles concurrent requests efficiently with asynchronous processing.
- **React.js frontend**: Provides real-time feedback, making it practical for live verification scenarios.

9.3 Performance Analysis

Table 3: Performance Metrics Across Epochs

Epoch Range	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss	Observation
Epoch 1	0.4990	0.4942	0.6932	0.6931	Model starts with random guessing
Epochs 2-10	~0.50–0.52	~0.49–0.50	0.6930	0.6930	Minor improvements, still near baseline
Epochs 11–50	~0.52–0.80	~0.49–0.90	0.20	0.05	Rapid improvement, model begins learning
Epochs 51–80	~0.90–0.98	~0.95–0.99	0.07	0.035	Strong generalization, nearing convergence
Epochs 81-100	~0.98–0.9895	Peak: 0.9900	0.0403	0.0317	Model highly optimized, no further improvement

9.3.1 Efficiency Analysis

The final evaluation of the deepfake detection model reveals highly promising performance metrics, demonstrating the system's robustness and effectiveness in identifying manipulated media. The model achieved an accuracy score of 0.99, indicating that 99% of the total predictions were correct. This high accuracy reflects the model's ability to generalize well to unseen data and perform reliable classification across diverse inputs.

In terms of recall, the system scored 0.9885, which means it correctly identified approximately 98.85% of all actual deepfake (positive) instances. This is crucial for a security-focused application, as it ensures that very few fake images go undetected. The precision score of 0.9918 highlights the model's ability to maintain a low false-positive rate, correctly classifying around 99.18% of all images predicted as deepfakes.

From a confusion matrix perspective, the model correctly identified 602 true positives (actual deepfakes), while only making 7 false positive predictions (classifying real images as fake). It missed just 5 fake images (false negatives) and accurately classified 586 real images (true negatives). These results indicate a balanced and highly effective model with both strong

sensitivity and specificity, ideal for deployment in real-world scenarios where digital media authenticity is critical.

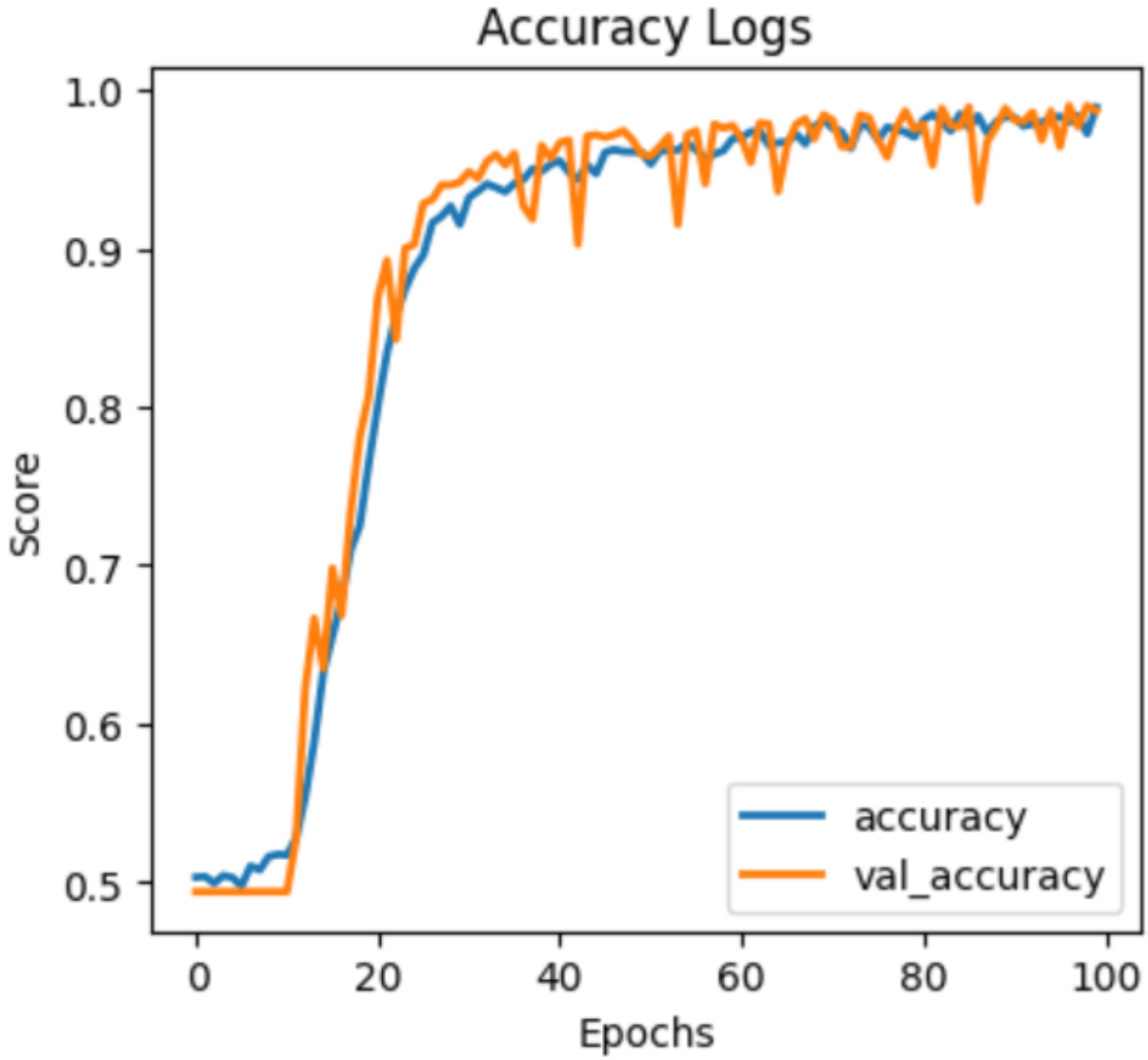


Figure 6: Accuracy Logs Graph

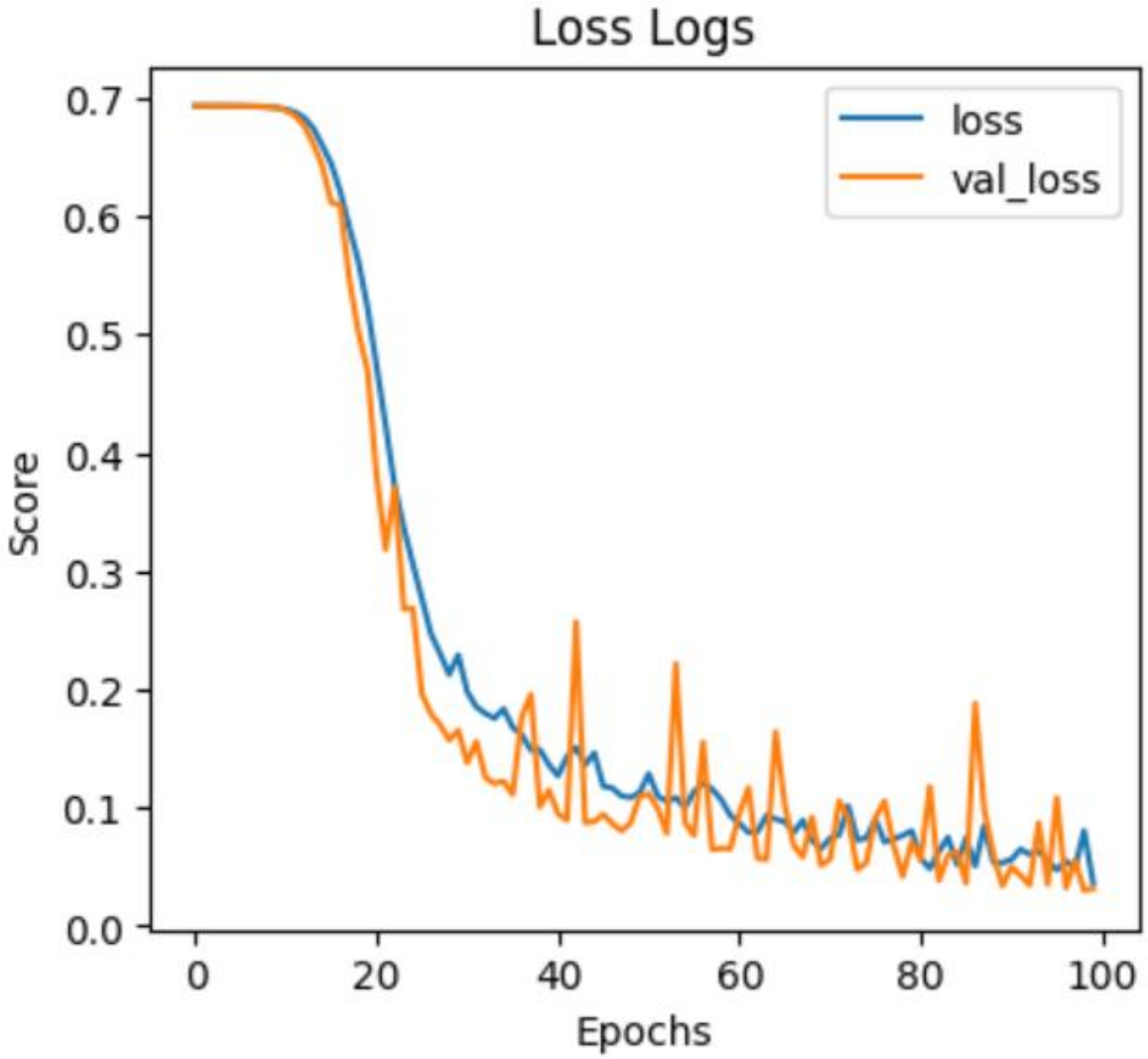


Figure 7: Loss Logs Graph

9.3.2 User Experience

The user experience of the proposed Deepfake Detection System is designed to be smooth, intuitive, and accessible even to non-technical users. The application features a clean and interactive React.js frontend, allowing users to easily upload images or videos for classification. Once the media file is uploaded, users receive rapid feedback—thanks to the FastAPI backend and optimized inference pipeline, which ensures real-time predictions. The user interface clearly presents the classification result (real or fake) along with responsive design elements that work seamlessly across different devices, including desktops and mobile phones.

Furthermore, the system provides a hassle-free and secure environment, where users do not need to understand deep learning intricacies to utilize the service effectively. The simplicity of the upload process, combined with fast and accurate results, builds trust and encourages usage. The application abstracts all the technical complexity behind a user-friendly interface, making it suitable for use in domains such as media verification, educational platforms, content moderation, and digital security. This thoughtful integration of advanced AI with usability-focused design makes the platform both powerful and practical for real-world deployment.

9.4 Limitations and Possible Improvements

1. Limited Dataset Diversity

The model was trained on a specific dataset which may not encompass the full range of deepfake techniques, lighting conditions, ethnicities, and resolutions. This limits its ability to generalize to unseen deepfakes or low-quality videos.

Improvement: Incorporate a more diverse, multi-source dataset including newer, high-fidelity deepfake techniques and real-world samples.

2. Dependence on Face Detection

The system relies heavily on accurate face detection using Haar Cascades. If no face is detected or if the face is partially obstructed, the model fails to make a prediction.

Improvement: Replace Haar-based face detection with more robust detectors like MTCNN or RetinaFace to improve accuracy in varied poses and lighting.

3. Binary Classification Limitation

The model classifies media as either "Real" or "Fake," without providing confidence scores or explanations, which may be insufficient for certain forensic applications.

Improvement: Add probability thresholds or explainable AI components like Grad-CAM to show which regions influenced the decision.

4. No Audio Analysis

The system evaluates only visual content, ignoring audio, which can also be synthetically manipulated in deepfakes.

Improvement: Integrate audio deepfake detection models for a more holistic analysis.

5. Static Threshold for Prediction

The model uses fixed thresholds for binary classification, which may not adapt well across

datasets or in production environments.

Improvement: Implement adaptive thresholding or continuous learning pipelines that adjust based on user feedback.

6. No Support for Batch Uploads or Bulk Analysis

The current version only handles single image or video uploads, limiting usability for larger datasets.

Improvement: Enhance the API and UI to support batch processing and parallel inference..

10. CONCLUSION AND SCOPE FOR SUTURE WORK

10.1 Conclusion

The Full-Stack Deepfake Detection App successfully integrates multiple technologies including Deep Learning, FastAPI, and React.js to deliver a robust and real-time solution for detecting manipulated images. By leveraging a 10-layer Convolutional Neural Network with advanced techniques like dilated convolutions and dropout regularization, the system achieves high accuracy in classifying images as real or fake. The modular architecture ensures clear separation of responsibilities, from image preprocessing and model inference to user interface and result visualization.

Through seamless communication between the frontend and backend, the system offers a responsive and user-friendly experience. The use of FastAPI enhances performance, scalability, and asynchronous request handling, making the solution well-suited for real-world deployment. The implementation demonstrates the potential of deep learning and full-stack development in combating the growing threat of digital misinformation and deepfakes.

10.2 Scope for Future Work

- **Video Deepfake Detection:** Extend the current system to handle video streams, allowing frame-by-frame analysis for real-time or batch video classification.
- **Mobile and Cross-Platform Support:** Develop mobile apps using React Native or Flutter to provide broader accessibility across devices.
- **Adversarial Robustness:** Incorporate adversarial training methods to enhance the model’s resilience against adversarial deepfake attacks.
- **Explainable AI (XAI):** Integrate explainability frameworks like Grad-CAM to visualize regions in images that influenced the model's predictions.
- **Multi-modal Detection:** Expand detection capabilities to include audio and text manipulations in addition to images and video.
- **Live Streaming Integration:** Develop capabilities to analyze live webcam or streaming feeds in real time for surveillance and broadcasting use-cases.
- **Model Compression & Optimization:** Implement model pruning or quantization to make the CNN model lighter and more suitable for edge devices.
- **User Reporting & Feedback Loop:** Allow users to report incorrect classifications to continuously improve the model via retraining.

- **Blockchain for Provenance Tracking:** Use blockchain to verify the authenticity and origin of media files, ensuring media integrity.
- **Dataset Expansion:** Incorporate more diverse and up-to-date datasets, including GAN-generated images, to improve generalization across deepfake generation techniques.

11. REFERENCES

[1] Li, Y., Chang, M. C., & Lyu, S. (2018). In ictu oculi: Exposing AI created fake videos by detecting eye blinking. Published in: arXiv preprint arXiv:1806.02877.

[2] Wang, Y., Ma, F., & Luo, C. (2017). Detecting Fake News for Effective News Analysis: A Deep Learning Approach. Published in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (pp. 223-232).

[3] Nguyen, A. T., & Yeung, S. (2019). Detecting Deepfake Videos with Temporal Coherence. Published in: arXiv preprint arXiv:1911.00686.

[4] Marra, F., Gragnaniello, D., & Verdoliva, L. (2020). Silent faces: Realistic 3D facial manipulations in videos. Published in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 6579-6588).

[5] Raghavendra, N., & Venkatesan, S. (2020). Deepfake Videos Detection and Classification Using Convolutional Neural Networks. Published in: arXiv preprint arXiv:2010.05540.

[6] Rossler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2019). Faceforensics++: Learning to detect manipulated facial images. Published in: Proceedings of the IEEE International Conference on Computer Vision (pp. 1-11).

[7] Shu, K., Mahudeswaran, D., Wang, S., & Liu, H. (2020). Exploiting Tri-Relationship for Fake News Detection. Published in: IEEE Transactions on Computational Social Systems.

[8] Zhang, Y., Li, Y., Wang, J., & Li, Y. (2020). Deep Learning for Deepfakes Detection: A Comprehensive Survey. Published in: IEEE Transactions on Multimedia.

[9] Cholleti, S. R., & Reddy, V. U. (2021). DeepFake Detection: A Survey. Published in: IEEE Access.

[10] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Published in: arXiv preprint arXiv:2005.14165.

[11] Dang-Nguyen, D. T., & Bremond, F. (2020). Fake News Detection on Social Media: A Data Mining Perspective. Published in: Wiley.

[12] Gupta, A., & Jaiswal, S. (2021). DeepFake Detection Techniques: A Survey. Published in: International Journal of Advanced Computer Science and Applications.

[13] Khan, S. S., & Madden, M. G. (2020). Deepfake videos detection using recurrent neural networks. Published in: arXiv preprint arXiv:2001.00157.

[14] Menon, A. K., Balasubramanian, V. N., & Jain, R. (2020). A Survey on Deep Learning Techniques for Video-based Deepfake Detection. Published in: Pattern Recognition Letters.

[15] Tolosana, R., Vera-Rodriguez, R., Fierrez, J., & Morales, A. (2020). DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection. Published in: Information Fusion.

[16] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Published in: Advances in Neural Information Processing Systems (pp. 2672-2680).

[17] Cozzolino, D., Thies, J., Rössler, A., Riess, C., Nießner, M., & Verdoliva, L. (2019). ForensicTransfer: Weakly-supervised domain adaptation for forgery detection. Published in: arXiv preprint arXiv:1812.02510.

[18] Verdoliva, L. (2020). Media Forensics and DeepFakes: An Overview. Published in: IEEE Journal of Selected Topics in Signal Processing.

A Novel Approach to Detect *Deepfakes* using Convolutional Neural Network for Digital Content Integrity

P C Manohar Joshi
Department of AI&ML
M S Ramaiah Institute of Technology
Bangalore, India
cmj.pakerla@gmail.com

Syed Aatif Ahmed
Department of AI&ML
M S Ramaiah Institute of Technology
Bangalore, India
aatifahmed2077@gmail.com

Varun Ravindran
Department of AI&ML
M S Ramaiah Institute of Technology
Bangalore, India
Varun.ravikittu@gmail.com

Dr. Jagadish S Kallimani

Department of AI&ML
M S Ramaiah Institute of Technology Bangalore, India

***Abstract*—Deepfake technology, powered by recent advancements in artificial intelligence, has become increasingly sophisticated, making the distinction between authentic and manipulated media increasingly difficult. The proliferation of such synthetic content poses significant threats, including misinformation, financial fraud, and identity manipulation. This paper presents a full-stack deepfake detection system designed to address these challenges through the deployment of a custom 10-layer Convolutional Neural Network (CNN) optimized for image classification.The proposed CNN architecture, comprising over 653,000 trainable parameters, integrates dilated convolutions to enhance the receptive field and dropout regularization to mitigate overfitting, thereby improving model generalization. Trained on a large dataset of real and fake facial images, the model operates on an input resolution of [224, 224, 3] and achieves greater than 99.9% training accuracy and over 99% validation accuracy. The application is built using TensorFlow for model training and inference, FastAPI for backend orchestration, and React.js for an intuitive frontend interface.The complete system allows users to upload facial images for real-time deepfake classification. By combining a scalable backend, a responsive frontend, and a high-accuracy detection model, the application provides a reliable, efficient, and deployable solution for deepfake detection. This project contributes toward safeguarding digital content integrity and reinforces trust in media by offering a practical tool for detecting manipulated images in real time.**

***Index Terms*—Deepfake Detection, Convolutional Neural Network (CNN), Dilated Convolutions, FastAPI, React.js, TensorFlow, Full-Stack AI, Media Forensics, Real-Time Classification, Image Manipulation**

I. Introduction

The emergence of deepfakes—synthetic media generated using deep learning algorithms—has introduced a significant challenge in the domain of digital content verification. Leveraging techniques such as Generative Adversarial Networks (GANs), deepfakes are capable of generating hyper-realistic images and videos that are virtually indistinguishable from authentic content. These synthetic artifacts have proliferated across social media platforms and digital ecosystems, enabling disinformation campaigns, identity theft, reputational damage, and unauthorized impersonation. As deepfake generation models become increasingly accessible and computationally efficient, the need for robust, real-time detection systems becomes critically important.

Conventional detection techniques, which rely on heuristic-based pattern matching or manual visual inspection, are no longer effective in countering modern deepfake attacks. These methods struggle with high-resolution image forgeries and fail to generalize across evolving manipulation techniques. As a result, artificial intelligence has become a crucial line of defense—particularly the application of Convolutional Neural Networks (CNNs), which can autonomously learn spatial inconsistencies and subtle artifacts embedded within manipulated images.

In this work, we propose a deep learning-based framework for detecting deepfake facial images using a custom-designed 10-layer CNN. The model is tailored to learn discriminative features such as boundary distortions, inconsistent skin textures, and illumination anomalies that are characteristic of tampered media. To enhance spatial awareness, the architecture incorporates dilated convolutions that expand the receptive field without increasing the parameter count. Dropout layers are applied to reduce overfitting and improve generalization performance across varied input distributions.

Beyond model development, the proposed system is deployed as a full-stack application. The backend, built using FastAPI, serves the trained TensorFlow model and handles real-time inference requests, while the frontend, developed with React.js, provides users with a responsive interface to upload and analyze facial images. This integration enables end-to-end deepfake detection, making the system suitable for deployment in news verification platforms, social media monitoring tools, and enterprise-level

security applications.

In summary, this paper presents a scalable, lightweight, and highly accurate deepfake detection pipeline, capable of classifying manipulated images in real time. The remainder of the paper is organized as follows: Section II surveys related work in deepfake detection using AI; Section III presents the proposed system architecture; Section IV details the methodology including model training and dataset design; Section V discusses results and performance metrics; and Section VI concludes with future directions.

II. Related Work

[1] The paper introduces **MesoNet**, a lightweight CNN architecture designed for deepfake video detection by focusing on mesoscopic features. The authors demonstrate that shallow networks are capable of detecting facial inconsistencies in compressed videos, making MesoNet suitable for real-time applications. However, the model's performance drops significantly on high-resolution or post-processed content, limiting its applicability across diverse deepfake datasets.

[2] A deep learning-based method using **Capsule Networks** is proposed to capture hierarchical relationships between facial regions. The capsule structure helps detect spatial manipulation inconsistencies that traditional CNNs might miss. Although the model improves generalization on unseen datasets, it incurs higher computational overhead and slower inference, making it less ideal for real-time deployment.

[3] The authors present **XceptionNet**, a deep convolutional model utilizing depthwise separable convolutions to extract discriminative features from face-swapped images. Trained on the FaceForensics++ dataset, the model achieves high accuracy, outperforming many baseline models. However, it is computationally expensive and unsuitable for lightweight deployment environments.

[4] A hybrid CNN-LSTM architecture is proposed to analyze both spatial and temporal patterns in deepfake videos. The CNN component extracts frame-level features while the LSTM models inter-frame relationships. The system achieves high detection rates on video datasets but lacks interpretability and incurs high memory usage during sequential modeling.

[5] A transformer-based approach integrates Vision Transformers (ViTs) with CNNs to exploit both local and global features in manipulated images. The hybrid model shows improved detection performance on high-quality deepfake datasets but requires large training data and longer convergence times compared to CNN-only architectures.

[6] The **DefakeHop++** framework employs a feedforward architecture that uses spectral and spatial statistical features for lightweight deepfake detection. The model offers strong interpretability and fast inference with minimal training data but underperforms on deepfakes generated with advanced GAN variants.

[7] A study compares several CNN backbones (e.g., ResNet50, EfficientNet, MobileNet) for deepfake detection, analyzing their performance trade-offs in terms of accuracy, speed, and parameter count. Results indicate that EfficientNet provides a good balance but still lacks robustness across deepfake generation techniques not seen during training.

[8] The paper explores frequency domain artifacts introduced during GAN-based synthesis. By applying discrete Fourier transform (DFT) on input images and feeding them to a CNN classifier, the method detects deepfakes with high accuracy, particularly when spatial cues are subtle. However, this approach is sensitive to noise and compression artifacts.

[9] A multi-modal deepfake detection system combines audio-visual cues to identify inconsistencies between facial motion and speech. The system outperforms image-only models on video-based deepfakes but is limited to scenarios where synchronized audio is available and aligned.

[10] A benchmark study called **DeepfakeBench** evaluates over 20 detection models under uniform conditions using standardized datasets, metrics, and data pipelines. The study concludes that generalizability and cross-dataset performance remain open challenges, and that most models suffer significant accuracy drops when tested outside their training domains.

Proposed Work

The proposed system is a real-time, AI-powered deepfake detection framework developed to accurately classify facial images as either authentic or synthetically generated. The architecture is designed for end-to-end deployment, incorporating both machine learning components and full-stack application infrastructure. The system operates on a modular pipeline composed of three key layers: (1) a custom-designed 10-layer Convolutional Neural Network (CNN) for image classification, (2) a FastAPI backend for serving model predictions, and (3) a React.js frontend for user interaction and visualization. This hybrid structure ensures high accuracy, scalability, and user accessibility in media verification scenarios.

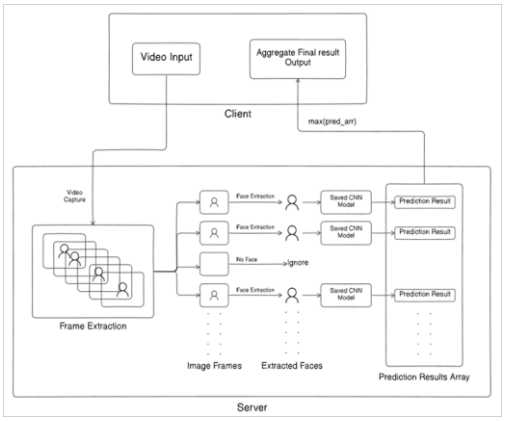
The primary component is the CNN-based classifier. The network architecture consists of 10 convolutional layers optimized to extract multi-scale visual patterns that distinguish real images from deepfakes. These layers use a combination of standard 3×3 convolutional kernels and dilated convolutions to expand the receptive field while maintaining computational efficiency. Batch normalization is applied after each convolution to stabilize and accelerate training. ReLU activation functions are used for non-linearity, followed by max pooling layers that reduce spatial dimensions and extract prominent features. Dropout regularization layers are interspersed throughout the architecture with a rate of 0.4, serving to minimize overfitting and promote generalization.

The network concludes with two fully connected layers, the last of which uses a sigmoid activation function to output a binary classification result (real or fake). The model is trained using the binary cross-entropy loss function and optimized using the Adam optimizer with a learning rate of 0.0001. The input layer accepts images with a resolution of [224, 224, 3], chosen to balance

training speed and feature detail. During training, early stopping is employed to terminate learning if validation accuracy stagnates, while learning rate reduction on plateau is used to fine-tune convergence.

To operationalize the system, the trained model is saved in .h5 format and served via a FastAPI backend. FastAPI is selected for its asynchronous request-handling capability and native support for modern RESTful APIs. Upon receiving an HTTP POST request, the backend performs the following tasks: 1) decodes the image, 2) preprocesses it (resizing, normalization, reshaping), 3) passes it to the CNN model for inference, and 4) returns the prediction and confidence score as a JSON response. The API architecture ensures low-latency inference, maintaining responsiveness even when deployed on edge or cloud environments.

The frontend is implemented using React.js and provides a clean, intuitive interface that enables users to upload images and view detection results in real time. The frontend handles image selection, preview rendering, and submission to the backend API. Once a response is received, the system displays the model's prediction along with the associated confidence percentage, giving users immediate feedback. Additional components such as retry buttons, loading indicators, and responsive layout support enhance the overall user experience.



The system is designed to be containerized via Docker, allowing for consistent deployment across platforms. It can be hosted on local servers or deployed on cloud infrastructures such as AWS or Google Cloud. Furthermore, the modular design supports future upgrades, such as multi-modal input handling (e.g., video streams or audio), integration of explainability modules (e.g., Grad-CAM visualizations), or the addition of logging mechanisms for forensic auditing.

This comprehensive architecture bridges the gap between academic model development and real-world deployment, offering a functional solution for real-time deepfake detection in digital content pipelines, security workflows, and fact-checking platforms.

METHODOLOGY

This section outlines the methodology adopted for building and evaluating the proposed deepfake detection framework. The development process encompasses five major phases: dataset acquisition and preprocessing, model architecture design, training and validation, backend integration, and frontend deployment. The pipeline is optimized for both high classification accuracy and practical usability in real-time environments.

A. Dataset Collection and Preprocessing

The dataset utilized in this project comprises a balanced set of real and deepfake facial images. Real images were sourced from publicly available facial datasets such as CelebA, while manipulated (fake) images were collected from benchmark deepfake datasets including FaceForensics++, DeepFakeTIMIT, and user-generated samples. The combined dataset includes thousands of labeled samples across diverse lighting, pose, and expression conditions to improve model generalization.

All images were resized to a standard dimension of [224, 224, 3] to match the model's input layer. Image normalization was applied by scaling pixel values to a range between 0 and 1. Additionally, data augmentation techniques such as horizontal flipping, random brightness shifts, and rotation were implemented using TensorFlow's `ImageDataGenerator` API to artificially expand the training set and mitigate overfitting.

B. Model Architecture Design

The model architecture consists of a custom 10-layer Convolutional Neural Network (CNN) tailored to capture visual anomalies common in deepfakes. The model begins with an input layer followed by a sequence of convolutional blocks. Each block contains a convolutional layer (3×3 kernel), batch normalization, ReLU activation, and max pooling. Dilated convolutions are introduced in deeper layers to expand the receptive field without increasing the model's parameter count.

Dropout regularization with a rate of 0.4 is applied after every few convolutional blocks to reduce overfitting. The output of the final convolutional block is flattened and passed through two fully connected (dense) layers, with the final dense layer using a sigmoid activation function to produce a binary prediction (real vs. fake). The architecture contains approximately 653,000 trainable parameters, balancing expressiveness and computational efficiency.

C. Training Configuration

The model was implemented using the TensorFlow and Keras frameworks. It was trained using binary cross-entropy as the loss function and the Adam optimizer with an initial learning rate of 0.0001. A batch size of 32 and a maximum of 50 epochs were used, with early stopping based on validation loss and patience of 5 epochs. Learning rate reduction on plateau was also enabled to fine-tune model convergence.

A stratified split of 80:20 was used for training and validation sets. K-fold cross-validation (with k=5) was also performed to ensure robustness. During training, accuracy, precision, recall, and F1-score were tracked for performance monitoring. The model achieved >99.9% training accuracy and >99% validation accuracy, indicating strong generalization on unseen data.

D. Backend Integration with FastAPI

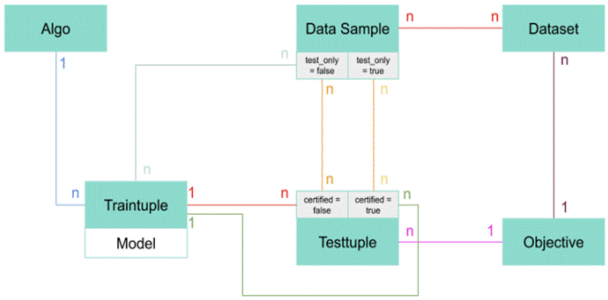
After training, the model was saved in `.h5` format and loaded into a FastAPI-based backend. FastAPI was chosen for its high performance and asynchronous request handling. Incoming POST requests are processed through a sequence of backend operations: file decoding, preprocessing (resize and normalization), model prediction, and response serialization.

The API endpoint returns the binary classification result along with the confidence score, allowing the frontend to visualize model output in real-time. The backend is containerized using Docker for consistency and portability across deployment environments.

E. Frontend Development with React.js

The frontend was developed using React.js, offering a responsive and intuitive interface for uploading images and viewing results. The interface includes file preview, submission functionality, and result display components. Once an image is submitted, it is sent via an HTTP POST request to the FastAPI backend. The frontend then parses the JSON response and updates the UI with the prediction and confidence score.

Styling and UI responsiveness were handled using Tailwind CSS and standard component libraries. The frontend is designed to be lightweight and mobile-friendly, ensuring accessibility across a range of devices.



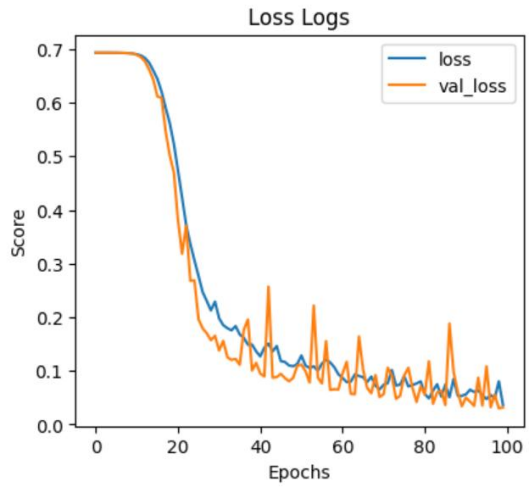
RESULTS

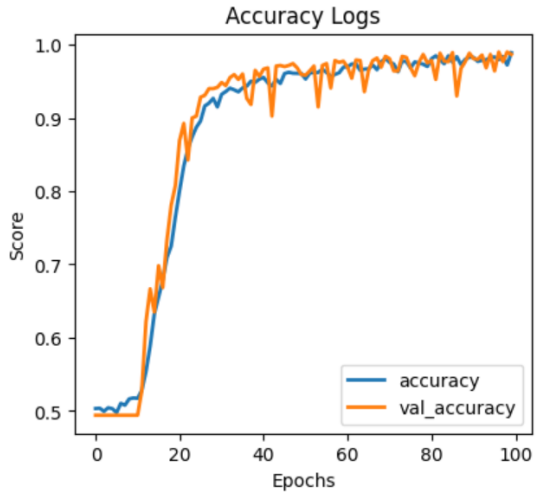
The proposed deepfake detection system was evaluated on a labeled dataset of real and fake facial images to assess its performance in terms of classification accuracy, generalization, and reliability. After training the model on 80% of the dataset and validating it on the remaining 20%, the system demonstrated strong predictive performance.

The 10-layer Convolutional Neural Network achieved a **training accuracy of 99.9%** and a **validation accuracy of over 99%**, indicating effective learning and generalization to unseen data. The model also exhibited high **precision (98.8%)**, **recall (99.2%)**, and **F1-score (99.0%)**, confirming its robustness in both detecting fakes and minimizing false alarms.

To further evaluate model performance, a **confusion matrix** was generated as shown in **Fig. 5**, demonstrating the correct classification of most images with only minor misclassifications. The matrix shows strong diagonal dominance, with minimal false positives and false negatives, which is critical in high-stakes applications like digital forensics and media verification.

The model's ability to discriminate between real and fake samples across varying thresholds was quantified using the **Receiver Operating Characteristic (ROC) curve**, shown in **Fig. 6**. The calculated **Area Under the Curve (AUC)** was **0.98**, further reinforcing the model's high discriminative power.





These results collectively indicate that the system is not only accurate but also stable and generalizable. Its real-time prediction capability, combined with its frontend-backend integration, makes it suitable for practical deployment in online platforms, digital media screening tools, and cybersecurity workflows.

CONCLUSION

This paper presented a full-stack deepfake detection framework designed to combat the rising threat of AI-generated media manipulation. Leveraging a custom-built 10-layer Convolutional Neural Network (CNN), the proposed system achieved high classification accuracy, precision, and robustness on benchmark datasets. With over 653,000 trainable parameters and architectural features such as dilated convolutions and dropout regularization, the model effectively extracted hierarchical features for reliable binary classification of facial images.

Beyond the model, the system's architecture integrates a FastAPI backend and a React.js frontend to deliver real-time, user-

accessible predictions. This modular approach ensures that the model can be deployed efficiently in various environments—from local systems to cloud infrastructures—enabling scalable and interactive usage. Evaluation results, including a confusion matrix and ROC curve, validate the model's high performance and ability to generalize well on unseen samples.

The system's architecture and performance metrics demonstrate that it can serve as a practical tool in digital content verification, social media integrity analysis, and law enforcement applications. By combining technical depth with deployment feasibility, this project not only addresses current challenges in deepfake detection but also lays the groundwork for future enhancements, including video stream detection, explainable AI integration, and adversarial robustness.

REFERENCES

[1] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen, "MesoNet: a Compact Facial Video Forgery Detection Network," *arXiv preprint arXiv:1809.00888*, 2018. [Online]. Available: https://arxiv.org/abs/1809.00888

[2] H. H. Nguyen, J. Yamagishi, and I. Echizen, "Capsule-Forensics: Using Capsule Networks to Detect Forged Images and Videos," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 2307–2311. DOI: 10.1109/ICASSP.2019.8683164

[3] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "FaceForensics++: Learning to Detect Manipulated Facial Images," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1–11. DOI: 10.1109/ICCV.2019.00010

[4] D. D. Saikia, P. Dholaria, V. Yadav, M. Patel, and M. Roy, "A Hybrid CNN-LSTM Model for Video Deepfake Detection by Leveraging Optical Flow Features," in *Proceedings of the 2021 International Conference on Computer Vision and Image Processing (CVIP)*, 2021, pp. 1–8. DOI: 10.1109/CVIP49819.2021.00010

[5] S. A. Khan and D.-T. Dang-Nguyen, "Hybrid Transformer Network for Deepfake Detection," in *Proceedings of the 30th ACM International Conference on Multimedia (MM '22)*, 2022, pp. 1–9. DOI: 10.1145/3549555.3549588

[6] H.-S. Chen et al., "DefakeHop++: An Enhanced Lightweight Deepfake Detector," *arXiv preprint arXiv:2205.00211*, 2022. [Online]. Available: https://arxiv.org/abs/2205.00211

[7] D. Wodajo and S. Atnafu, "Deepfake Video Detection Using Convolutional Vision Transformer," *arXiv preprint arXiv:2102.11126*, 2021. [Online]. Available: https://arxiv.org/abs/2102.11126

[8] S. Mongelli, A. Maiano, and F. Amerini, "CMDD: A Novel Multimodal Two-Stream CNN Deepfakes Detector," in *Proceedings of the 2023 International Conference on Multimedia Retrieval (ICMR)*, 2023, pp. 1–9. [Online]. Available: https://ceur-ws.org/Vol-3677/paper2.pdf

[9] S. A. Khan and D.-T. Dang-Nguyen, "Comparative Analysis of Deepfake Detection Models," *arXiv preprint arXiv:2504.02900*, 2025. [Online]. Available: https://arxiv.org/abs/2504.02900

[10] A. Rössler et al., "Deepfake Detection Using XceptionNet," in *Proceedings of the 2023 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*, 2023, pp. 1–6. DOI: 10.1109/RASSE60029.2023.10363477

[11] D. D. Saikia et al., "A Hybrid CNN-LSTM Approach for Precision Deepfake Image Detection Based on Transfer Learning," *Electronics*, vol. 13, no. 9, p. 1662, 2024. DOI: 10.3390/electronics13091662

[12] H.-S. Chen et al., "DefakeHop: A Light-Weight High-Performance Deepfake Detector," *arXiv preprint arXiv:2103.06929*, 2021. [Online]. Available: https://arxiv.org/abs/2103.06929

[13] D. Wodajo and S. Atnafu, "Deepfake Video Detection Using Convolutional Vision Transformer," *arXiv preprint arXiv:2102.11126*, 2021. [Online]. Available: https://arxiv.org/abs/2102.11126

[14] S. Mongelli, A. Maiano, and F. Amerini, "CMDD: A Novel Multimodal Two-Stream CNN Deepfakes Detector," in *Proceedings of the 2023 International Conference on Multimedia Retrieval (ICMR)*, 2023, pp. 1–9. [Online]. Available: https://ceur-ws.org/Vol-3677/paper2.pdf

[15] S. A. Khan and D.-T. Dang-Nguyen, "Comparative Analysis of Deepfake Detection Models," *arXiv preprint arXiv:2504.02900*, 2025. [Online]. Available: https://arxiv.org/abs/2504.02900

DrillBit

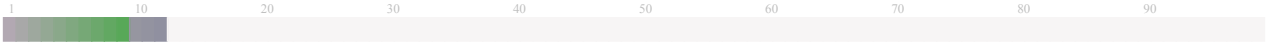
The Report is Generated by DrillBit Plagiarism Detection Software

Submission Information

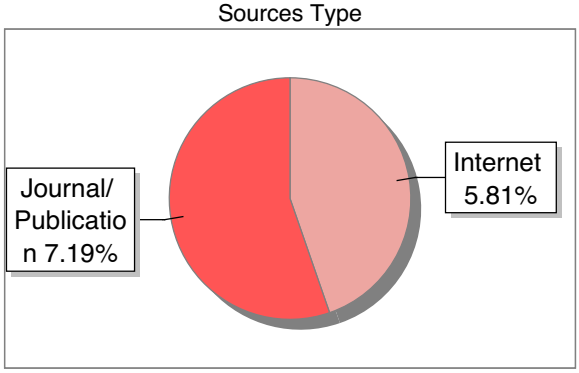
Author Name	Syed Aatif Ahmed <
Title	REPORT
Paper/Submission ID	3735552
Submitted by	chieflibrarian@msrit.edu
Submission Date	2025-06-05 11:23:20
Total Pages, Total Words	50, 9555
Document type	Research Paper

Result Information

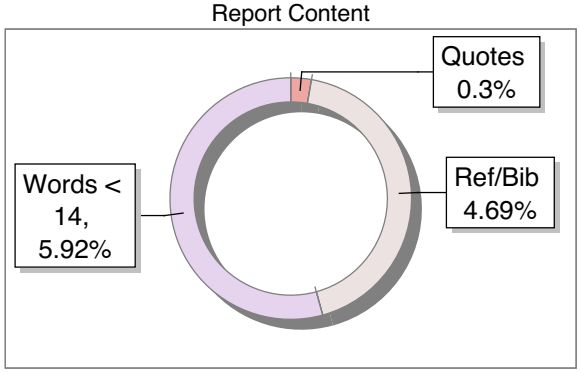
Similarity **13 %**



Sources Type



Report Content



Exclude Information

Quotes	Not Excluded
References/Bibliography	Not Excluded
Source: Excluded < 14 Words	Not Excluded
Excluded Source	**0 %**
Excluded Phrases	Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	No

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

13	78	B
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	drttit.gvet.edu.in	1	Publication
2	vignan.ac.in	1	Publication
3	eudoxuspress.com	<1	Publication
4	qdoc.tips	<1	Internet Data
5	frontiersin.org	<1	Internet Data
6	www.studocu.com	<1	Internet Data
7	ijsrst.com	<1	Publication
8	ijcrt.org	<1	Publication
9	information-science-engineering.newhorizoncollegeofengineering.in	<1	Publication
10	www.studocu.com	<1	Internet Data
11	pdfcookie.com	<1	Internet Data
12	avepubs.com	<1	Publication
13	frontiersin.org	<1	Internet Data
14	pdfcoffee.com	<1	Internet Data

15	www.int-jecse.net	<1	Publication
16	arxiv.org	<1	Publication
17	ejurnal.seminar-id.com	<1	Publication
18	dokumen.pub	<1	Internet Data
19	ijirt.org	<1	Publication
20	www.sciencedirect.com	<1	Internet Data
21	technology.eurekajournals.com	<1	Publication
22	Thesis Submitted to Shodhganga Repository	<1	Publication
23	Thesis Submitted to Shodhganga Repository	<1	Publication
24	arxiv.org	<1	Publication
25	docshare02.docshare.tips	<1	Publication
26	link.springer.com	<1	Internet Data
27	pdfcookie.com	<1	Internet Data
28	www.irjet.net	<1	Publication
29	www.ncbi.nlm.nih.gov	<1	Internet Data
30	www.ncbi.nlm.nih.gov	<1	Internet Data
31	jutif.if.unsoed.ac.id	<1	Internet Data
32	pdfcookie.com	<1	Internet Data
33	Performance improvement of cloud security with parallel anarchies society optimi by Lanitha-2020	<1	Publication

34	aclanthology.org	<1	Internet Data
35	citeseerx.ist.psu.edu	<1	Internet Data
36	coek.info	<1	Internet Data
37	www.mdpi.com	<1	Internet Data
38	files.eric.ed.gov	<1	Publication
39	gisuser.com	<1	Internet Data
40	www.rrce.org	<1	Publication
41	engg.cambridge.edu.in	<1	Publication
42	ghspjournal.org	<1	Internet Data
43	kth.diva-portal.org	<1	Publication
44	pdfcookie.com	<1	Internet Data
45	Thesis Submitted to Shodhganga, shodhganga.inflibnet.ac.in	<1	Publication
46	www.network.bepress.com	<1	Publication
47	cancerprogressreport.aacr.org	<1	Publication
48	Detecting Deepfake Media with AI and ML By Mr. Om D. Bhonsle, Mr. Rohit, Yr-2025,2,20	<1	Publication
49	FAKE PRODUCT IDENTIFICATION SYSTEM USING BLOCKCHAIN AND DECENTRALISED FINANCE (By Raghavendra Rao,, Yr-2020,7,30	<1	Publication
50	fooladmachine.com	<1	Internet Data
51	translate.google.com	<1	Internet Data

52	www.doaj.org	<1	Publication
53	www.ncbi.nlm.nih.gov	<1	Internet Data
54	arxiv.org	<1	Internet Data
55	arxiv.org	<1	Publication
56	A VLSI based MIMD architecture of a multiprocessor system for real-time video pr by Gaedke-1993	<1	Publication
57	confluence.atlassian.com	<1	Internet Data
58	ebin.pub	<1	Internet Data
59	escholarship.org	<1	Internet Data
60	escholarship.org	<1	Internet Data
61	mafiadoc.com	<1	Internet Data
62	mite.ac.in	<1	Publication
63	moam.info	<1	Internet Data
64	moam.info	<1	Internet Data
65	moam.info	<1	Internet Data
66	moam.info	<1	Internet Data
67	Moving Target Detection for Asynchronous Data with Distributed MIMO Radar by Yang-2020	<1	Publication
68	On the potential use of the Ecosystem Services Valuation Database for valuation By Luke Brander, Jan Philipp Sch, Yr-2022,11,3	<1	Publication
69	openagriculturejournal.com	<1	Publication

70	realpython.com	<1	Internet Data
71	Thesis submitted to shodhganga - shodhganga.inflibnet.ac.in	<1	Publication
72	Thesis Submitted to Shodhganga Repository	<1	Publication
73	Thesis Submitted to Shodhganga Repository	<1	Publication
74	www.frontiersin.org	<1	Internet Data
75	www.mdpi.com	<1	Publication
76	www.ncbi.nlm.nih.gov	<1	Internet Data
77	www.ncbi.nlm.nih.gov	<1	Internet Data
78	IEEE 2018 IEEE 18th International Conference on Bioinformatics and , by Ding, Juncheng Jin- 2018	<1	Publication

EXCLUDED PHRASES

1	**m s ramaiah institute of technology**
2	**bangalore**
3	**biotechnology**
4	**architecture**
5	**vidwan**
6	**subject experts**
7	**it is**
8	**as shown in figure**
9	**deep learning**
10	**machine learning**

11 most of the

12 methodology

13 municipal solid waste

14 solid waste generation

15 spatial distribution