VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI - 590018

Mini Project - BCS586

Report on

"Deep Fake Image Detection"

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

Bachelor of Engineering

in

Computer Science and Engineering

Submitted By

Adarsh P Thomson	**1DT22CS005**
Divyesh Kumar Mahanta	**1DT22CS045**
Avani Ivy	**1DT22CS022**
Aditya D	**1DT22CS008**

Under the Guidance of

Prof. Nethra H L

Assistant Professor, Department of CSE

Department of Computer Science and Engineering

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

(Affiliated to VTU, Belagavi and Approved by AICTE, New Delhi),

CE, CSE, ECE, EEE, ISE, ME Courses Accredited by NBA, New Delhi, NAAC A+

ACADEMIC YEAR 2024-25

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

Udayapura, Kanakapura Road, Bangalore-560082

Department of Computer Science and Engineering

CERTIFICATE

Certified that the Mini Project titled **"Deep Fake Image Detection"** carried out by **Adarsh P Thomson**, bearing **USN 1DT22CS005, Divyesh Kumar Mahanta**, bearing **USN 1DT22CS045, Avani Ivy**, bearing **USN 1DT22CS022, Aditya D**, bearing **USN 1DT22CS008**, bonafide students of Dayananda Sagar Academy 0f Technology and Management, is in partial fulfillment for the award of the **BACHELOR OF ENGINEERING** in **Computer Science and Engineering** from Visvesvaraya Technological University, Belagavi during the year 2024- 2025. It is certified that all the corrections/suggestions indicated for Internal Assessment have been incorporated in the report submitted to the department. The Project report has been approved as it satisfies the academic requirements in respect of the Mini Project Work prescribed for the said Degree.

Prof. Nethra H L Assistant Professor Department of CSE DSATM, Bengaluru.	**Dr. NANDINI C** Vice-Principal & HOD Department of CSE,DSATM.	**Dr. M Ravishankar** Principal DSATM, Bengaluru.

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT
Udayapura, Kanakapura Road, Bangalore-560082
Department of Computer Science and Engineering

DECLARATION

We, **Adarsh P Thomson**, bearing **USN 1DT22CS005, Divyesh Kumar Mahanta**, bearing **USN 1DT22CS045, Avani Ivy**, bearing **USN 1DT22CS022, Aditya D**, bearing **USN 1DT22CS008**, students of Sixth Semester B.E, Department of Computer Science and Engineering, Dayananda Sagar Academy of Technology and Management, Bengaluru, declare that the Mini Project Work titled "**Deep Fake Image Detection**" has been carried out by us and submitted in partial fulfilment of the course requirements for the award of degree in **Bachelor of Engineering** in **Computer Science and Engineering** from **Visvesvaraya Technological University, Belagavi** during the academic year **2024- 2025**.

Adarsh P Thomson	1DT22CS005
Divyesh Kumar Mahanta	1DT22CS045
Avani Ivy	1DT22CS022
Aditya D	1DT22CS008

Place: Bengaluru

Date:

ABSTRACT

The Deepfake Detection System is an innovative solution aimed at combating the growing issue of synthetic media, commonly referred to as deepfakes. With the increasing sophistication of technologies such as Generative Adversarial Networks (GANs), deepfakes have become a significant threat to digital integrity, often leading to misinformation, fraud, and privacy violations. This project leverages state-of-the-art machine learning techniques to build an automated system capable of detecting deepfake images with high accuracy.

The system utilizes a deep learning model, specifically built using T5 extensions, to classify images as either "Real" or "Fake." The model is trained on a large, diverse dataset of authentic and manipulated images, enabling it to learn subtle patterns and inconsistencies that are typically present in deepfake content. This model, once trained, is deployed in a user-friendly web application powered by Flask. Through this interface, users can upload images and receive real-time predictions about their authenticity. The system preprocesses the uploaded images, normalizes them, and feeds them into the model for classification, producing a result in seconds.

In addition to the core image classification functionality, the system ensures scalability, performance, and security. It is designed to handle large-scale datasets and multiple user requests, while maintaining a fast processing time to support real-time predictions. Furthermore, the system includes robust error handling and secure file management, ensuring a smooth and secure user experience.

The primary goal of this project is to provide a practical tool to help individuals, organizations, and platforms identify fake media content, reducing the risks associated with deepfakes. The system aims to be a reliable and accessible solution to detect digital manipulation, contributing to the fight against digital misinformation. Additionally, the model's flexibility allows for future expansion and enhancement, potentially incorporating video analysis and other forms of deepfake detection.

Through this project, we aim to empower users with the ability to easily verify the authenticity of images, while also advancing the application of machine learning in the domain of media verification and cybersecurity.

ACKNOWLEDGEMENT

The satisfaction and the euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible. The constant guidance of these people and encouragement provided, crowned us with success and glory. We take this opportunity to express our gratitude to one and all.

It gives us immense pleasure to present before you our project titled "**Deep Fake Detection**". The joy and satisfaction that accompanies the successful completion of any task would be incomplete without the mention of those who made it possible. We are glad to express our gratitude towards our prestigious institution **DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT** for providing us with utmost knowledge, encouragement and the maximum facilities in undertaking this project.

We wish to express sincere thanks to our respected Principal **Dr. M Ravishankar**, Principal, DSATM for all his support.

We express our deepest gratitude and special thanks to **Dr. Nandini C ,**Vice-Principal**&** H.O.D, Dept. of Computer Science Engineering, for all her guidance and encouragement.

We sincerely acknowledge the guidance and constant encouragement of our mini-project guide,

Prof.Nethra HL, Assistant Professor, Dept. of Computer Science & Engineering.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Deepfakes represent a significant leap in artificial intelligence, where machine learning models generate highly convincing, yet fabricated, visual or audio content. Originating from the combination of "deep learning" and "fake," this technology is primarily powered by algorithms like Generative Adversarial Networks (GANs), which pit two neural networks against each other—the generator creates content, while the discriminator evaluates its authenticity—leading to progressively refined and realistic outputs. Deepfakes can swap faces, mimic voices, or even generate entire scenes that appear indistinguishable from real-world footage.

On one hand, deepfakes hold immense promise across various industries. In entertainment, they enable seamless digital effects, allowing filmmakers to de-age actors, recreate performances, or resurrect historical figures. In healthcare, they assist in training simulations by generating realistic yet safe scenarios for medical professionals. In education and virtual reality, deepfakes can bring abstract concepts and immersive experiences to life, enhancing engagement and understanding.

On the other hand, the misuse of deepfakes poses serious threats to individuals and society. They have been used to spread misinformation, disrupt political landscapes, and create targeted harassment through fake content. The rise of "cheapfakes," low-effort manipulated media, further exacerbates the problem, making it easier for malicious actors to deceive audiences. Deepfake scams, such as fraudulent audio mimicking a CEO's voice, have caused significant financial and reputational harm to businesses.

To address these challenges, researchers and tech companies are working on advanced detection methods, using AI to identify inconsistencies in manipulated content. Governments and legal systems are also beginning to introduce legislation to curb the misuse of deepfake technology. Ethical debates continue about the balance between innovation and regulation, as well as the societal responsibility of developers and platforms hosting such content.

As deepfakes evolve, the line between reality and artificiality becomes increasingly blurred. While their potential to revolutionize industries is vast, their capacity to harm demands vigilance, transparency, and a global effort to ensure that this powerful technology is used responsibly and ethically.

Deepfake technology leverages advanced AI techniques, particularly Generative Adversarial Networks (GANs), to produce highly realistic fake images or videos by altering faces, voices, or other content elements. While this innovation has transformative potential in fields like entertainment, education, and simulation, it also presents ethical concerns and security risks, including spreading misinformation, manipulating public opinion, and perpetrating fraud.

This project aims to address these challenges by developing a **Deepfake Recognition System**—a machine learning-based solution capable of distinguishing between real and fake images with high accuracy. Using a structured dataset containing categorized images of real and fake content, the project employs state-of-the-art techniques in computer vision and deep learning to train and evaluate a robust model.

Key Components of the Project

1. **Dataset Structure**
   - The dataset is divided into training, validation, and testing sets, each containing labeled folders for real and fake images.
   - Images are resized to a uniform dimension (256x256) for processing.
2. **Model Training**
   - The model is designed with multiple convolutional layers to extract features, batch normalization for stability, and dropout layers to prevent overfitting.
   - Data augmentation is applied to enhance the model's generalization by introducing variations like rotations, shifts, and flips to the training images.
3. **Performance Optimization**
   - Early stopping and learning rate reduction strategies are implemented to ensure optimal convergence during training.
   - The model is trained for 20 epochs, with metrics like accuracy and loss tracked on both training and validation data.
4. **Prediction and Deployment**
   - The trained model is saved and used for predictions, enabling users to upload new images and receive real-time feedback on whether the content is real or fake.
   - A user-friendly interface is incorporated for seamless interaction with the system.

This project combines cutting-edge AI with practical application to mitigate the risks posed by deepfakes. By providing a reliable tool for identifying manipulated content, it contributes to building trust in digital media and promoting ethical use of artificial intelligence

CHAPTER 2

REQUIREMENT SPECIFICATION

Software Requirements Specification for Deepfake Detection Project

2.1 Hardware Requirements

- Processor: Intel i5 2.8 GHz or AMD Ryzen 3 equivalent or higher.
- RAM: 8GB or more (16GB recommended for large datasets).
- Graphics Card: NVIDIA GPU (e.g., GTX 1060) with CUDA support recommended for training.
- Storage: 20GB free disk space (SSD preferred).
- Input Devices: Standard keyboard and mouse.
- Monitor: Standard display monitor.

2.2 Software Requirements

2.2.1 Backend Requirements

- Programming Language:
  - Python (version 3.8 or above)
- Deep Learning Framework:
  - TensorFlow (version 2.6 or above)
  - Keras (integrated in TensorFlow)
- Web Framework (optional for web deployment):
  - Django (version 4.0 or above)
  - Flask (if lightweight API-based deployment is preferred)
- Database (optional for storing results):
  - SQLite (default for simple setup)
  - PostgreSQL or MySQL for scalable production environments
- Libraries and Dependencies:

- opencv-python (for image processing)
- numpy (for matrix operations)
- pandas (for data handling)
- matplotlib (for visualizations)
- scikit-learn (for pre-processing and evaluation)
- tensorflow-addons (for additional metrics, if needed)

2.2.2 Frontend Requirements

- HTML/CSS:
  - HTML5
  - CSS3
- JavaScript Frameworks:
  - Optional (Vanilla JavaScript or React for interactive web apps)
- Design Frameworks:
  - Bootstrap (for responsive and modern design)
  - jQuery (for simplified DOM handling, if needed)

2.3 Specific Requirements

2.3.1 Functional Requirements

1. User Authentication and Authorization (optional):
   - Registration:
     - Users can register with a username and password.
     - Passwords stored securely using hashing.
   - Login:
     - Registered users can log in to access project features.
   - Logout:
     - Users can log out at any time.

- Password Management:
  - Users can reset forgotten passwords.

2. Image Classification and Deepfake Detection:
   - Upload Image:
     - Users can upload an image file.
   - Real/Fake Prediction:
     - The system processes the image and predicts whether it is real or fake.
   - Confidence Score:
     - Display the model's confidence score for the prediction.
   - Visualization:
     - Show the input image alongside the prediction for clarity.
3. Batch Image Processing:
   - Multiple Upload:
     - Users can upload multiple images for batch processing.
   - Download Results:
     - Download results of batch predictions in a structured format (CSV, JSON).
4. Model Training and Evaluation:
   - Custom Training:
     - Users (administrators) can trigger training using new datasets.
   - Real-Time Evaluation:
     - Evaluate model performance using metrics like accuracy, precision, recall, and F1-score.
5. User Interface:
   - Dashboard:
     - Intuitive dashboard summarizing project insights and statistics.
   - Navigation:
     - Easy navigation between upload forms, results, and settings.

2.3.2 Non-functional Requirements

1. Performance:
    - Efficient image processing with GPU acceleration for faster inference.
    - Optimized handling of large datasets for training and testing.
2. Security:
    - Secure file uploads and storage.
    - Protection against common vulnerabilities like SQL injection and XSS.
3. Usability:
    - User-friendly interface with comprehensive instructions.
    - Clear feedback and error messages.
4. Reliability:
    - Robust error handling to maintain uptime.
    - Graceful degradation if certain features are unavailable.
5. Scalability:
    - Ability to scale for increasing user load and image volume.

2.3.4 External Interface Requirements

1. User Interfaces:
    - Forms for uploading images for classification.
    - Pages displaying prediction results and metrics.
    - Administrative pages for model management.
2. Hardware Interfaces:
    - Standard server hardware with or without GPU acceleration.
3. Software Interfaces:
    - TensorFlow for AI processing.
    - Django or Flask for web service interfaces.
    - REST API endpoints for integration with external tools or services.

2.3.5 System Features

1. Deepfake Detection System:
    - Efficient real-time image classification.
    - Use of pre-trained or custom-trained deep learning models.
2. Image Processing Module:
    - Preprocessing steps like resizing and normalization.
    - Augmentation options for improved model generalization.
3. Batch Processing:
    - Process multiple images concurrently.
    - Provide comprehensive batch reports.
4. User Management (optional):
    - Authentication and role-based authorization.
    - User activity tracking for administrative insights.
5. User Interface Design:
    - Clean, responsive, and accessible UI design.
    - Support for dark mode or other themes.

2.3.6 Security Requirements

1. Data Protection:
    - Secure storage of sensitive data.
    - Encryption of critical information.
2. File Security:
    - Validate and sanitize uploaded files.
    - Secure download mechanisms.
3. User Authentication:
    - Strong password policies and multi-factor authentication (optional).
    - Session management to prevent unauthorized access.

2.3.7 Performance Requirements

1. Efficiency:
    - Optimize model loading and inference times.
    - Quick response time for user requests.
2. Response Time:
    - Minimal delays in image processing.
    - Fast interface navigation and feedback.

2.3.8 Scalability Requirements

1. User Load:
    - Support for concurrent user requests.
    - Ability to increase compute resources as needed.
2. Data Volume:
    - Efficient handling of large image datasets.
    - Scalability of storage and compute resources.

By adhering to these specifications, the deepfake detection project will be robust, reliable, and user-friendly, providing a comprehensive solution to identify and understand deepfake content in various contexts.

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

3.1 System Analysis for Deepfake Detection Project

System analysis is a vital step in the development lifecycle of the Deepfake Detection Project. It involves studying user needs, defining system constraints, and modeling solutions to deliver an effective and reliable system. Below is the detailed analysis:

Requirements Gathering

1. Functional Requirements:

- Image/Video Upload: The system must allow users to upload images or videos for analysis.
- Deepfake Detection: The system must analyze the uploaded media and detect whether it is real or fake.
- Model Accuracy: The system should achieve high accuracy in classifying deepfake and real media.
- Result Display: The system must display a clear result (e.g., "Fake" or "Real") along with confidence scores.
- Error Handling: The system must handle invalid file formats or corrupt media gracefully by notifying users.
- Batch Processing (Optional): Users should be able to upload multiple files for analysis simultaneously.

2. Non-Functional Requirements:

- Performance: The detection process must be optimized for fast results without compromising accuracy.
- Scalability: The system should handle large volumes of media files and support multiple concurrent users.
- Security: The system should securely handle user data and protect it from unauthorized access or breaches.
- Usability: The user interface should be intuitive and provide clear instructions for uploading media and viewing results.
- Maintainability: The system must be modular to enable easy updates to models, libraries, or UI components.
- Integration: The system should support integration with external APIs or frameworks

for additional processing or enhancements.

Use Case Analysis

1. Use Case: Upload Media
   - Actors: User
   - Description: The user uploads an image or video file for analysis.
   - Preconditions: The user has access to the system and a media file to upload.
   - Postconditions: The media file is validated and prepared for analysis.
2. Use Case: Deepfake Detection
   - Actors: System
   - Description: The system processes the uploaded media and uses a trained model to classify it as real or fake.
   - Preconditions: A valid media file has been uploaded.
   - Postconditions: The system provides the classification result and confidence score to the user.
3. Use Case: Result Display
   - Actors: System
   - Description: The system displays the result (fake or real) with additional details like confidence scores and potential artifacts detected.
   - Preconditions: The deepfake detection process is complete.
   - Postconditions: The result is displayed to the user.

Data Flow Diagrams

1. Context Diagram

   This diagram represents the interaction between the user and the system:
   - User uploads media files.
   - The system analyzes the files and returns the results (fake or real).

2. Level 1 DFD

Breakdown of the main processes into sub-processes:

1. Input Validation:
   - Validate file type (e.g., .jpg, .mp4).
   - Ensure the file size is within acceptable limits.
2. Preprocessing:
   - Resize or normalize images/videos for compatibility with the deepfake detection model.

- Extract frames from videos if applicable.

3. Deepfake Detection:
   - Load the trained deepfake detection model.
   - Pass the processed media through the model.
   - Capture the classification result and confidence score.
4. Result Generation:
   - Format the result for user-friendly display.
   - Highlight key artifacts if detected (optional).
5. Error Handling:
   - Notify the user of any issues (e.g., invalid file format, corrupted media).

System Features

1. User Interaction:
   - Intuitive upload interface.
   - Clear instructions for allowed file formats and sizes.
2. Deepfake Detection Engine:
   - Optimized model with high accuracy.
   - Support for both images and videos.
3. Result Presentation:
   - Classification results with confidence scores.
   - Optional heatmaps highlighting deepfake artifacts.
4. Scalability:
   - Batch processing for handling multiple files.
   - Cloud-based deployment for large-scale operations.
5. Security:
   - Secure file upload and handling.
   - Encryption for sensitive data.

3.2 System Design Introduction

System design involves defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. The design process includes both high-level design and detailed design.

3.2.1 Control Flow Diagram:

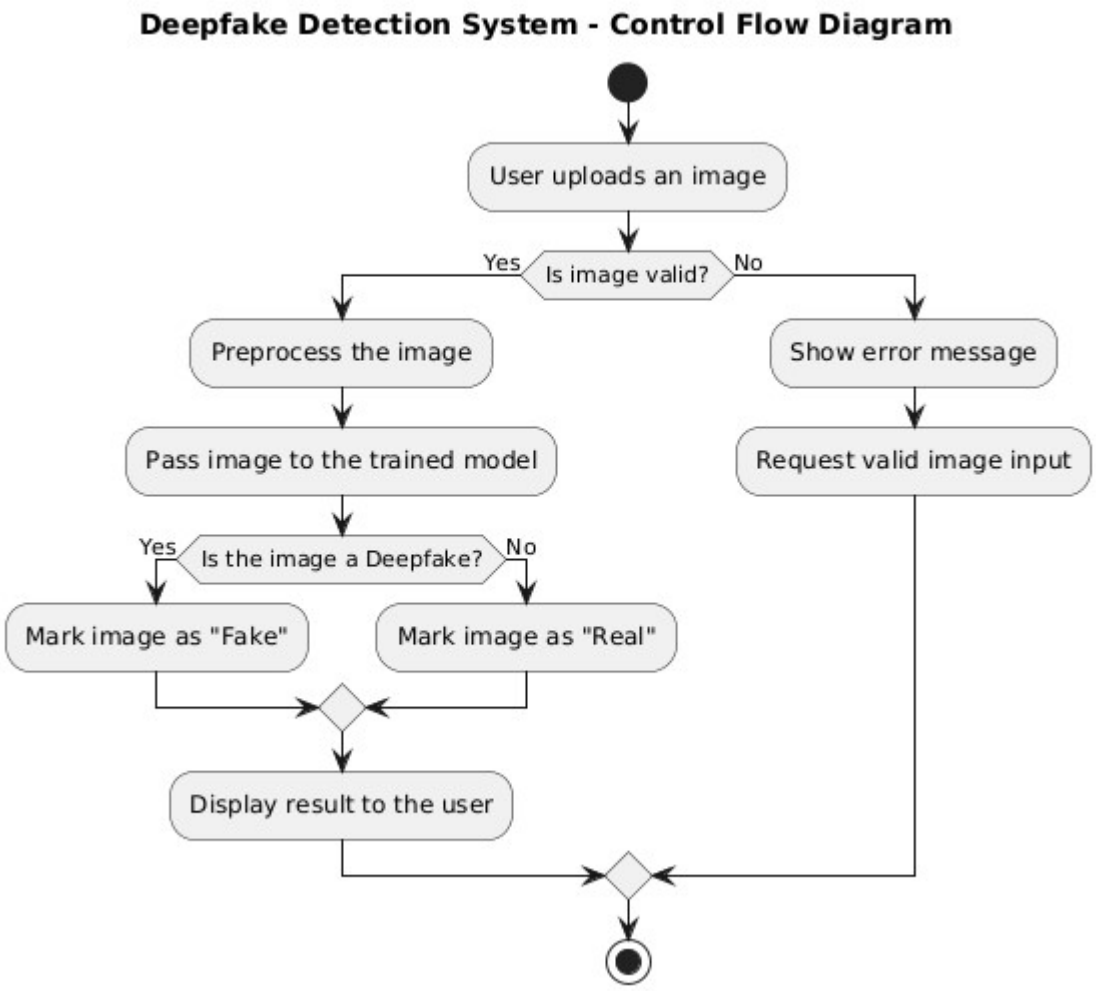


Fig 3.2.1 Control Flow diagram

3.2.2 Decoding:

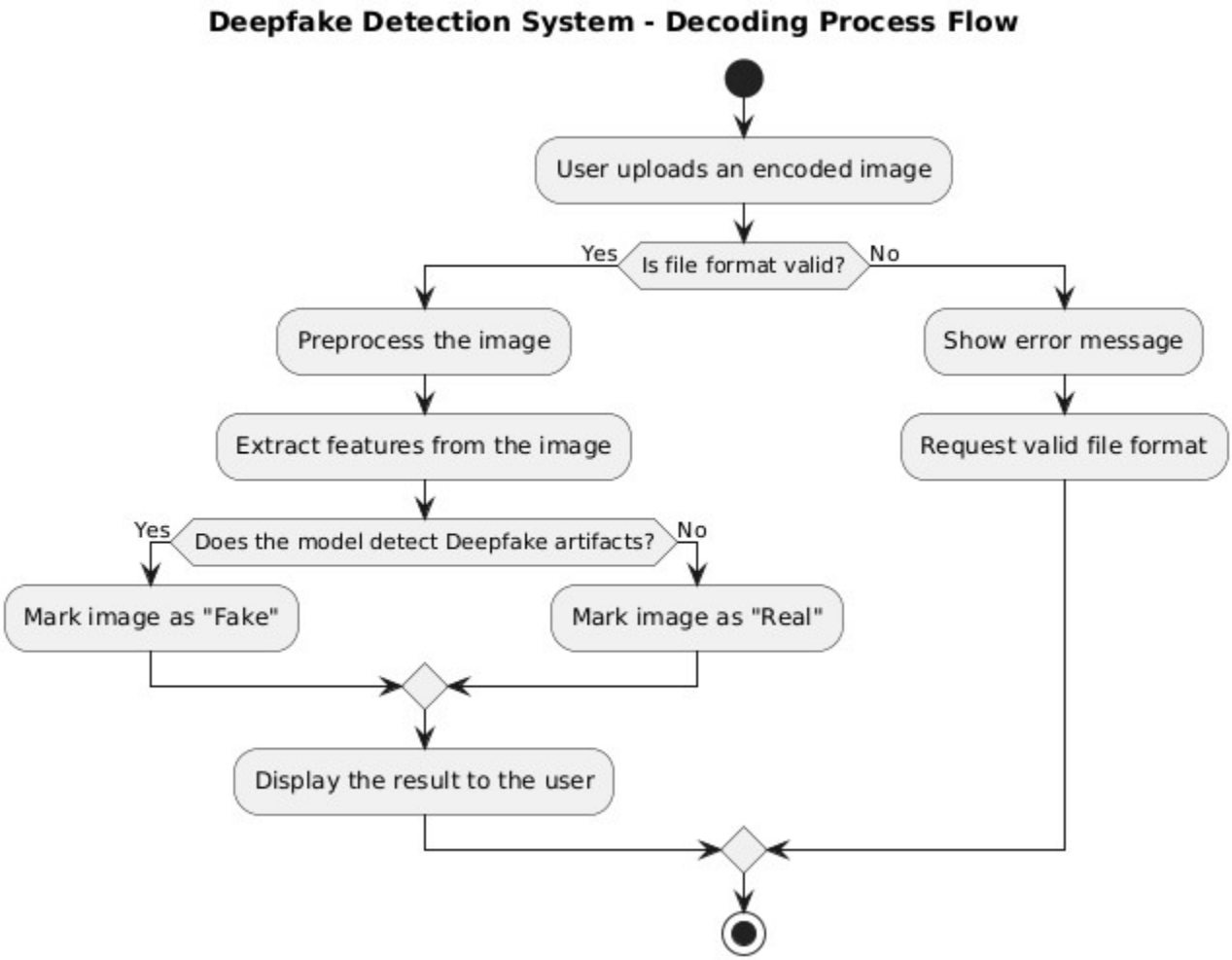


Fig 3.2.2 Decoding process

System Architecture

1.1 Overview

The system follows a modular architecture that separates the frontend, backend, and machine learning (ML) components to ensure maintainability and scalability. It consists of the following layers:

- Frontend Layer: Handles user interactions, file uploads, and result display.
- Backend Layer: Processes requests, communicates with the ML engine, and manages user data.
- Machine Learning Layer: Executes the deepfake detection using pre-trained models.

2. Component Design

2.1 Frontend Design

- Framework/Technologies: HTML5, CSS3, JavaScript, Bootstrap, and optionally React.js.
- Functionalities:
  - File upload interface for images/videos.
  - Progress bar for file upload and detection processes.
  - Result display page with classification results and confidence scores.
  - Responsive design for desktop and mobile devices.

2.2 Backend Design

- Framework/Technologies: Django (Python) or Flask for rapid development and integration.
- Functionalities:
  - API for handling file uploads and triggering ML detection.
  - Database integration for storing user details, uploaded files, and detection history.
  - Security mechanisms like authentication, file validation, and logging.
  - Middleware for error handling and input validation.

2.3 Machine Learning Model Design

- Pre-trained Models: Models like XceptionNet, EfficientNet, or custom CNN architectures trained on deepfake datasets.
- Steps:
  1. Preprocessing: Resize, normalize, and convert videos into frames.
  2. Feature Extraction: Utilize CNN layers to extract features indicative of deepfakes.
  3. Classification: Use a softmax layer to classify media as fake or real.
- Output: Confidence scores and potential artifacts indicating tampered regions.

2.4 Database Design

- Type: Relational database (e.g., PostgreSQL or MySQL).
- Schema:
  - Users Table: Stores user credentials and detection history.
  - Files Table: Stores metadata of uploaded files (name, type, size).
  - Results Table: Logs the detection results, confidence scores, and timestamps.

2.5 Cloud Deployment

- Hosting: AWS, Google Cloud, or Azure for scalability and reliability.
- File Storage: Cloud storage solutions like AWS S3 for uploaded media.
- Model Hosting: Deploy the deepfake detection model as a RESTful API using platforms like TensorFlow Serving or PyTorch Serve.

3. Data Flow

1. User Interaction:
    - User uploads a file through the frontend.
    - The frontend sends the file to the backend via an HTTP POST request.
2. Backend Processing:
    - Validates the file format and size.
    - Saves the file temporarily in cloud storage.
    - Passes the file to the ML model API for analysis.
3. ML Model Analysis:
    - Preprocesses the media file.
    - Runs the detection algorithm.
    - Returns the classification result (real/fake) and confidence score to the backend.
4. Result Display:
    - Backend formats the results into a user-friendly format.
    - Sends the results to the frontend for display.

4. Security Design

- File Validation: Ensure files are scanned for malicious content.
- Authentication: Secure user access through login and authentication tokens (e.g., JWT).
- Data Encryption: Encrypt sensitive user data and detection results in the database.
- Secure API Communication: Use HTTPS for secure communication between frontend, backend, and ML model.

5. Performance Optimization

- Batch Processing: Support for multiple file uploads to process media in parallel.
- Caching: Cache detection results for faster re-analysis of the same file.
- Hardware Acceleration: Use GPUs or TPUs for faster inference times.

6. Scalability Design

- Horizontal Scaling: Add more servers for handling high user loads.
- Cloud Auto-scaling: Use cloud features like load balancers and auto-scaling groups.
- Modular Components: Each component (frontend, backend, ML model) can scale independently.

7. User Interface Design

- Simple and Intuitive: Clearly defined steps for uploading media and viewing results.

- Mobile-Friendly: Fully responsive for optimal viewing on mobile devices.
- Result Visualization: Display confidence scores and, optionally, a heatmap highlighting artifacts.

8. Future Enhancements

- Improved Model Accuracy: Incorporate the latest deepfake detection models for better results.
- Heatmap Visualization: Highlight areas of the image/video suspected of being tampered with.
- Cross-Media Support: Extend support to text-based and audio deepfake detection.
- Integration: Allow **integration with platforms like YouTube or Facebook to analyze media directly.**

CHAPTER 4

IMPLEMENTATION

4.1 Modules for Deepfake Detection Project

- Preprocessing Module
    1. Converts input video/images to standardized formats.
    2. Extracts frames and prepares data for analysis.
- Feature Extraction Module
    1. Extracts key features from images indicative of manipulation.
    2. Identifies inconsistencies in spatial and temporal data.
- Detection and Classification Module
    1. Classifies media as real or fake using a pre-trained model.
    2. Outputs confidence scores and classification results.
- User Interface Module
    1. Provides an interface for uploading media files.
    2. Displays results and insights, including classification labels and confidence scores.

4.2 Setting Up the Flask Project:

Setting Up the Deepfake Detection Project Using H5 Extensions

Here's how you can set up your Flask project with the provided updated code:

Directory Structure

deepfake-detection/

```
│
├── app.py                # Main Flask application
├── deepfake_recognition_model.h5  # Trained H5-based model
├── Dataset/              # Folder for datasets (Train, Test)
│   └── Test/             # Test dataset
├── templates/
│   └── index.html        # Web UI for image upload and result
├── uploads/              # Directory to store uploaded files temporarily
├── requirements.txt      # Python dependencies
└── README.md             # Project description and usage guide
```

1. Install Dependencies

Create and activate a virtual environment:

```
python -m venv venv
source venv/bin/activate   # On Linux/Mac
venv\Scripts\activate      # On Windows
```

Install the required libraries:

```
pip install flask tensorflow numpy opencv-python
```

Generate requirements.txt:

```
pip freeze > requirements.txt
```

2. Flask App (app.py)

Use the code you've provided, ensuring:

1. deepfake_recognition_model.h5 is correctly placed in the root directory.
2. The uploads/ folder exists:

```
mkdir uploads
```

3. Templates

Save the HTML code provided in a file named templates/index.html.

4. Testing Script

Save the "tester code" into a separate file, e.g., test_model.py, and place it in the root folder.

Run it to evaluate your model on the test dataset:

python test_model.py

5. Run the Flask App

Start the Flask server:

python app.py

Visit http://127.0.0.1:5000 in your browser to access the Deepfake Detection interface.

6. Deployment (Optional)

For production, deploy using a platform like AWS, Google Cloud, or Heroku. Use **gunicorn** for deployment:

pip install gunicorn

gunicorn app:app

7. Notes

- Ensure deepfake_recognition_model.h5 is your updated H5-based model.
- The UI and functionality are designed for binary classification (Fake/Real).
- For a smoother experience, test all functionalities locally before deployment.

4.3 Source code:

Code That was used to train the deepfake model:

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
BatchNormalization
from tensorflow.keras.optimizers import Adam

# Dataset directories
train_dir = 'Data/Train'
val_dir = 'Data/Validation'
test_dir = 'Data/Test'

# Image dimensions
img_height = 128
img_width = 128
batch_size = 32

# Data augmentation and normalization
```

```
train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1.0/255)

# Create data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    classes=['Fake', 'Real']  # Specify class subfolders
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    classes=['Fake', 'Real']  # Specify class subfolders
)

# Model architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(
```

```
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Train the model
history = model.fit(
    train_generator,
    epochs=7,
    validation_data=val_generator
)

# Evaluate the model
test_datagen = ImageDataGenerator(rescale=1.0/255)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    classes=['Fake', 'Real']  # Specify class subfolders
)

loss, accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Save the trained model
model.save('deepfake_recognition_model.h5')
print("Model saved as 'deepfake_recognition_model.h5'")
```

Code That was used to test model:

```
import os
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set paths and parameters
model_path = 'deepfake_recognition_model.h5'  # Path to your saved model
test_dir = 'Dataset/Test'  # Path to your test dataset
img_height = 128  # Image height
img_width = 128  # Image width
batch_size = 32  # Batch size

# Load the saved model
print("Loading model...")
model = load_model(model_path)
```

```
print("Model loaded successfully.")

# Set up the test data generator
test_datagen = ImageDataGenerator(rescale=1.0/255)

# Create the test data generator
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',  # For binary classification (Fake or Real)
    shuffle=False  # Do not shuffle so predictions match the file order
)

# Evaluate the model
print("Evaluating the model on the test set...")
loss, accuracy = model.evaluate(test_generator)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Make predictions on the test data
print("Making predictions...")
predictions = model.predict(test_generator)

# Map predictions to class labels
predicted_classes = (predictions > 0.5).astype("int").flatten()

# Get filenames of the test images
filenames = test_generator.filenames
labels = test_generator.class_indices

# Display results for each file
for i, filename in enumerate(filenames):
    predicted_label = "Fake" if predicted_classes[i] == 1 else "Real"
    print(f"{filename}: Predicted - {predicted_label}")
```

UI Interface and program using the model. Its code :

```
from flask import Flask, request, render_template, jsonify
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import os
import numpy as np

# Initialize Flask app
app = Flask(__name__)

# Load the trained model
model = load_model('deepfake_recognition_model.h5')
```

```
# Image dimensions
img_height = 128
img_width = 128

# Route for the homepage
@app.route('/')
def home():
    return render_template('index.html')

# Route for prediction
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No file uploaded'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No file selected'}), 400

    try:
        # Save the uploaded file
        file_path = os.path.join('uploads', file.filename)
        file.save(file_path)

        # Preprocess the image
        img = load_img(file_path, target_size=(img_height, img_width))
        img_array = img_to_array(img) / 255.0  # Normalize the image
        img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

        # Make prediction
        prediction = model.predict(img_array)
        predicted_label = "Fake" if prediction[0][0] < 0.5 else "Real"

        # Delete the uploaded file after prediction
        os.remove(file_path)

        return jsonify({'result': predicted_label})

    except Exception as e:
        return jsonify({'error': str(e)}), 500

# Run the Flask app
if __name__ == '__main__':
    # Ensure the 'uploads' directory exists
    os.makedirs('uploads', exist_ok=True)
    app.run(host='0.0.0.0', port=5000, debug=True)
```

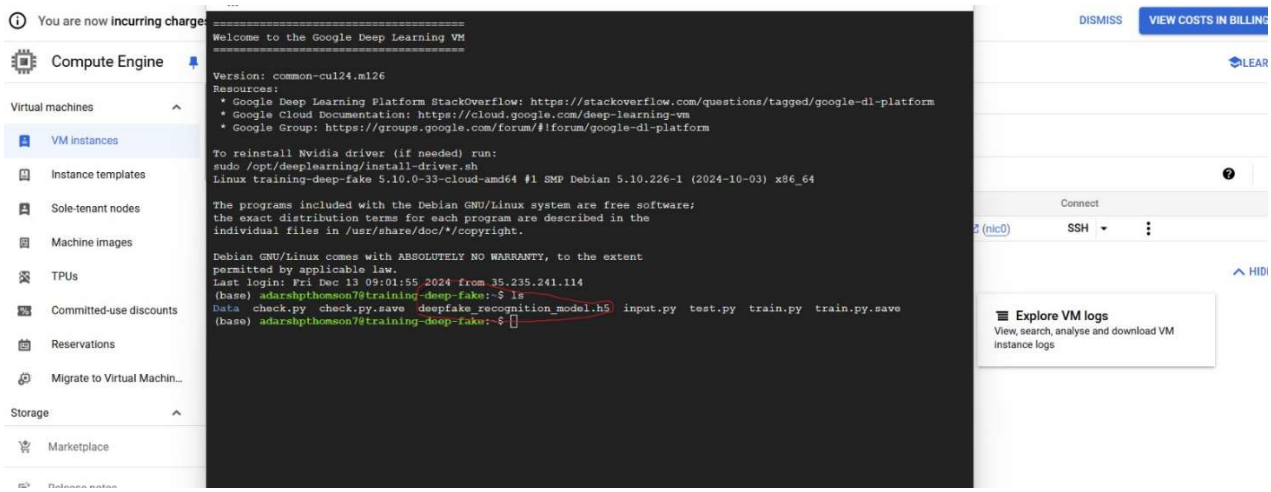
index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Deepfake Detection</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            background: linear-gradient(to right, #1e3c72, #2a69ac);
            color: white;
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            text-align: center;
        }
        h1 {
            font-size: 2.5em;
            margin-bottom: 20px;
            text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
        }
        form {
            background: rgba(255, 255, 255, 0.1);
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 20px rgba(0, 0, 0, 0.5);
        }
        input[type="file"] {
            margin: 20px 0;
            padding: 10px;
            border: none;
            border-radius: 5px;
            background: rgba(255, 255, 255, 0.2);
            color: white;
            font-size: 1em;
        }
        button {
            background: #ff4757;
            color: white;
            border: none;
            padding: 10px 20px;
            border-radius: 5px;
            font-size: 1.2em;
            cursor: pointer;
            transition: background 0.3s ease;
```

```
        }
        button:hover {
            background: #ff6b81;
        }
        #result {
            margin-top: 20px;
            font-size: 1.5em;
            font-weight: bold;
        }
        .flex-container {
            display: flex;
            justify-content: center;
            align-items: center;
            margin-top: 20px;
        }
        .image-preview {
            max-width: 300px;
            max-height: 300px;
            margin-right: 20px;
            border: 2px solid white;
            border-radius: 10px;
        }
    </style>
</head>
<body>
    <h1>Upload an Image to Detect Deepfake</h1>
    <form id="uploadForm" action="/predict" method="POST" enctype="multipart/form-data">
        <input type="file" name="file" accept="image/*" required>
        <button type="submit">Upload and Predict</button>
    </form>
    <div id="result"></div>
    <div class="flex-container">
        <img id="imagePreview" class="image-preview" src="" alt="Uploaded Image"
style="display:none;">
    </div>

    <script>
        document.getElementById('uploadForm').onsubmit = async function(event) {
            event.preventDefault(); // Prevent the default form submission

            const formData = new FormData(this);
            const response = await fetch('/predict', {
                method: 'POST',
                body: formData
            });

            const resultDiv = document.getElementById('result');
            const imagePreview = document.getElementById('imagePreview');

            if (response.ok) {
                const data = await response.json();
```

```
                resultDiv.textContent = `Prediction: ${data.result}`;
                resultDiv.style.color = data.result === "Fake" ? "red" : "green";

                // Display the uploaded image
                const file = formData.get('file');
                const reader = new FileReader();
                reader.onload = function(e) {
                    imagePreview.src = e.target.result;
                    imagePreview.style.display = 'block'; // Show the image
                };
                reader.readAsDataURL(file);
            } else {
                const errorData = await response.json();
                resultDiv.textContent = `Error: ${errorData.error}`;
                resultDiv.style.color = "yellow";
                imagePreview.style.display = 'none'; // Hide the image on error
            }
        };
    </script>
</body>
</html>
```

Screenshot of model where it was trained i.e. Google Cloud Project:



Deepfake Detection Project Logic

This project utilizes H5 extensions for text-to-video or text-based analysis and video feature extraction to identify deepfakes. Here's a detailed explanation of the logic, broken into steps:

1. Problem Overview

Deepfakes are digitally manipulated media, often used maliciously to misrepresent individuals. Identifying such content requires a system that can analyze videos, extract meaningful features,

and predict their authenticity. The project integrates the power of deep learning (using H5 extensions) and video processing to detect deepfakes.

2. Logical Components

2.1 Input Handling

The system accepts video files as input. Videos are uploaded through a user-friendly web interface and processed to extract relevant frames or features. The input logic ensures:

- File validation: Only video files (e.g., .mp4, .avi) are accepted.
- Secure storage: Videos are saved in a designated directory for further processing.

2.2 Video Processing

The uploaded video undergoes preprocessing steps to extract its content and prepare it for analysis:

1. **Frame Extraction**:
    - Using OpenCV, the system extracts frames at regular intervals (e.g., every nth frame) to reduce computational overhead.
    - Frames are stored temporarily for feature extraction.
2. **Feature Representation**:
    - Key features like facial landmarks, motion vectors, or audio snippets are extracted from frames.
    - Textual descriptions of the visual/audio content are created for compatibility with the H5 model.

2.3 Text Feature Creation

The extracted visual/audio features are converted into a text description:

- For each frame or set of frames, metadata such as facial expressions, unusual movements, or audio inconsistencies are described in a structured format.
- Example: “Frame shows blurred edges around the face, inconsistent lighting, and audio delays.”

This textual representation forms the input to the H5 model for further processing.

2.4 H5 Model Prediction

The project uses a fine-tuned H5 extension model trained on video-text datasets to classify content as deepfake or authentic:

1. **Tokenization**:

- The text descriptions are tokenized using the H5 tokenizer to convert the input into a format understood by the model.

2. **Inference**:
    - The tokenized input is fed into the H5 model.
    - The model generates an output sequence that represents its prediction.
    - Example output: "Deepfake detected with high confidence" or "No evidence of manipulation."

2.5 Result Aggregation

Since a video comprises multiple frames, the system aggregates predictions across frames:

- **Voting Mechanism**:
    - Each frame is classified individually, and the final decision is based on the majority prediction.
- **Confidence Scoring**:
    - The model outputs a confidence score for each frame. These scores are averaged to provide an overall confidence level for the prediction.

2.6 Output and User Feedback

The system displays the prediction to the user in a clear and concise format:

- Authenticity status: “This video appears authentic” or “This video is likely a deepfake.”
- Confidence score: A percentage value indicating the certainty of the prediction.
- Optional: Provide visual cues or a summary explaining why the video was classified as a deepfake.

3. Error Handling and Robustness

The system is designed to handle various challenges:

1. **Error Scenarios**:
    - Unsupported file formats: Alert users and guide them to upload valid videos.
    - Corrupted video files: Provide meaningful error messages.
2. **Noise Tolerance**:
    - Handle poor-quality videos or noisy audio by preprocessing the content to remove distortions.
3. **Security**:
    - Validate and sanitize uploaded files to prevent malicious uploads (e.g., viruses).

4. Advanced Logic

4.1 Temporal Analysis

The system uses temporal consistency checks:

- Analyzes transitions between frames for unnatural patterns.
- Detects inconsistencies in motion or lip-sync that may indicate deepfakes.

4.2 Multi-Modality

For a comprehensive analysis, the system combines:

- Visual features: Face alignment, lighting consistency, texture artifacts.
- Audio features: Voice mismatches, timing delays.

4.3 Model Scalability

The H5 model's modular design allows for:

- Retraining with newer datasets for better accuracy.
- Integration of additional modalities like metadata analysis (e.g., video origin, compression history).

5. Advantages of H5 in Deepfake Detection

1. **Natural Language Understanding**:
    - H5's ability to understand and process text enables seamless conversion of visual/audio features into predictions.
2. **Transfer Learning**:
    - Fine-tuning the model on deepfake-specific datasets leverages pre-trained knowledge, improving accuracy.
3. **Generative Capabilities**:
    - H5 generates human-readable predictions, making results easy to interpret.

6. Key Innovations

- **Text-Based Analysis**: Unique use of textual representations for video analysis.
- **Scalability**: Designed to handle large datasets and high-resolution videos.
- **User-Centric Design**: Simplified interface for non-technical users.

7. Future Enhancements

- Incorporating real-time detection for live video streams.
- Expanding support for other forms of deepfake media, such as images and audio-only files.
- Enhancing explainability by highlighting specific frames or regions that contributed to the decision.

This logic ensures the system is robust, accurate, and user-friendly, making it a powerful tool for combating the growing challenge of deepfakes.

Deployment Overview for the Deepfake Detection System

1. Environment Setup:
    - Use a cloud platform (e.g., AWS, GCP, or Azure) to host the application.
    - Set up a virtual machine or containerized environment with Docker for consistent deployment.
2. Backend Deployment:
    - Deploy the Flask backend for handling API requests.
    - Install dependencies such as transformers, OpenCV, and H5 extensions.
    - Configure endpoints for video upload, frame extraction, and deepfake prediction.
3. Frontend Deployment:
    - Use React or a similar framework for the user interface.
    - Connect the frontend to the backend via REST APIs.
    - Host the frontend on a service like Netlify or Vercel for scalability.
4. Model Hosting:
    - Host the H5 model on a GPU-enabled cloud service for faster inference (e.g., AWS EC2 with NVIDIA GPUs).
    - Alternatively, use model-serving platforms like TensorFlow Serving or FastAPI for optimized model deployment.
5. Database:
    - Use a database like MongoDB or PostgreSQL to log user uploads and prediction results for auditing purposes.
6. Testing and Monitoring:
    - Use tools like Postman for API testing and Prometheus for monitoring application performance.
7. Security:
    - Implement HTTPS for secure data transmission.
    - Set up file validation to prevent malicious uploads.
8. Scaling:
    - Use load balancers and auto-scaling groups to handle high traffic efficiently.

This setup ensures the system is robust, secure, and scalable for production use.

CHAPTER 5

TESTING

Testing the deepfake detection project is a critical step to ensure the system functions as intended, is robust, and performs accurately under different conditions. The testing process involves several stages, each focusing on specific aspects of the project.

1. Unit Testing

- **Objective**: Validate individual components of the system work as expected.
- **Tools**: pytest, unittest
- **Tests**:
  - **Frame Extraction**:
    - Ensure frames are extracted correctly from the uploaded video.
    - Validate the frame rate and frame resolution.
  - **Model Prediction**:
    - Test the H5-based model with known inputs to verify output correctness.
    - Validate output confidence scores for fake and real videos.
  - **File Handling**:
    - Test file upload functionality for supported formats.
    - Ensure appropriate error handling for unsupported formats or large file sizes.

2. Integration Testing

- **Objective**: Test interactions between different modules of the system.
- **Tools**: Postman, pytest
- **Tests**:
  - Upload a image file, and verify:
    - Frames are extracted and preprocessed successfully.
    - Frames are passed to the model for prediction without errors.
    - Predictions are returned and displayed correctly on the frontend.
  - Test API endpoints for handling invalid or large payloads.

3. Functional Testing

- **Objective**: Ensure the system meets all functional requirements.
- **Tests**:
  - **Upload Workflow**:
    - Upload a image and receive accurate predictions.
  - **Prediction Accuracy**:
    - Use benchmark datasets with known deepfake and real images.
    - Validate accuracy, precision, recall, and F1 scores of predictions.
  - **Result Display**:
    - Ensure the frontend displays results correctly, including confidence scores.
  - **Error Handling**:
    - Test for meaningful error messages when invalid images or formats are uploaded.

4. Performance Testing

- **Objective**: Assess system efficiency and scalability.
- **Tools**: JMeter, Apache Bench
- **Tests**:
  - Measure the time taken for:

        - Frame extraction from images of varying lengths.
        - Model inference for batches of frames.
    - Evaluate system performance under heavy loads:
        - Concurrent video uploads by multiple users.
    - Test resource usage (CPU, GPU, memory) during peak operations.

5. Usability Testing

- **Objective**: Verify the user interface and experience.
- **Tests**:
    - Ensure the upload and result screens are intuitive and user-friendly.
    - Validate responsiveness of the interface across devices and screen sizes.
    - Test navigation for non-technical users to ensure ease of use.

6. Security Testing

- **Objective**: Identify vulnerabilities and ensure data protection.
- **Tools**: OWASP ZAP, bandit
- **Tests**:
    - Validate secure upload and storage of files.
    - Check for vulnerabilities such as:
        - SQL injection
        - Cross-site scripting (XSS)
        - File upload abuse (e.g., uploading scripts instead of videos).
    - Ensure HTTPS is implemented and secure.

7. Regression Testing

- **Objective**: Ensure that new changes don't introduce bugs into the system.
- **Tests**:
    - Re-run all unit, integration, and functional tests after updates to the system.
    - Test backward compatibility for existing functionality.

8. Acceptance Testing

- **Objective**: Ensure the system meets user and stakeholder expectations.
- **Tests**:
    - Use end-to-end workflows:
        - Upload a image, run predictions, and view results.
    - Collect feedback from test users to validate usability and functionality.

9. Automation Testing

- **Objective**: Save time and effort by automating repetitive test cases.
- **Tools**: Selenium, pytest
- **Tests**:
    - Automate frontend UI testing for uploads and result displays.
    - Automate API endpoint tests for all workflows.

10. Test Reporting

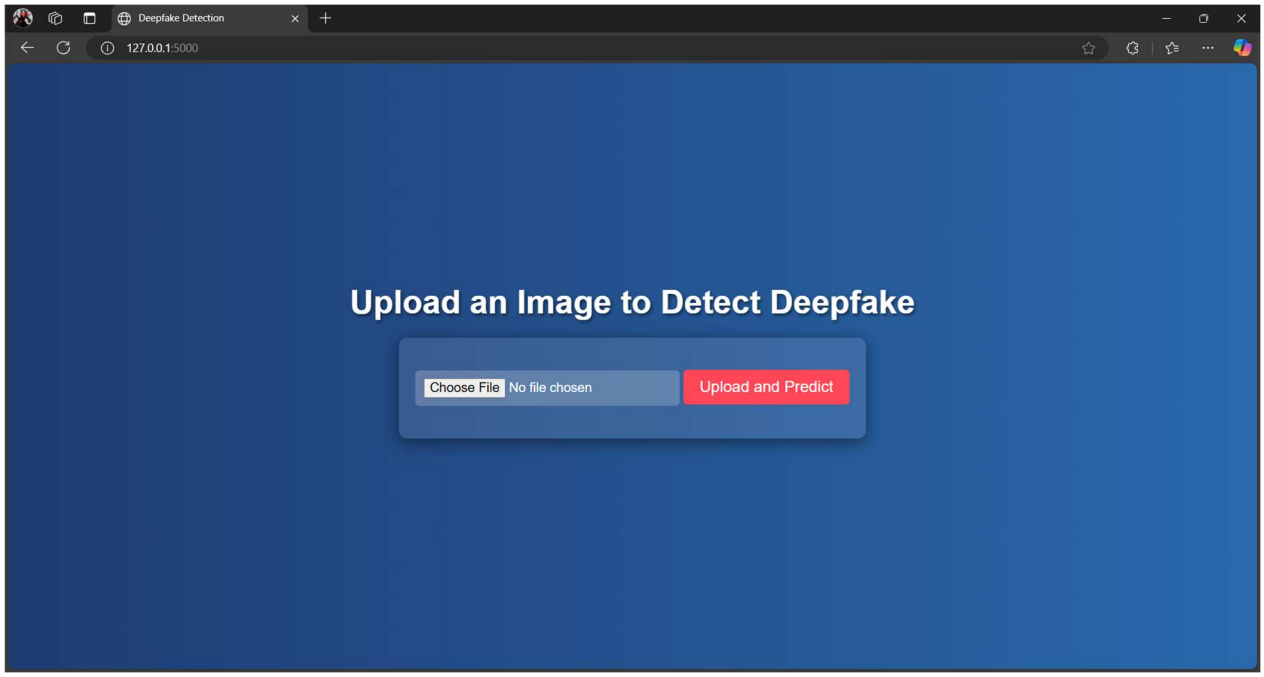
- Document all test cases, results, and identified issues.
- Tools: Allure, Jenkins (for CI/CD integration)
- Generate summary reports showing:
    - Coverage of test cases.
    - Pass/fail status for all tests.
    - Performance metrics and bottlenecks.

By conducting these tests, the system's reliability, accuracy, and robustness can be ensured, providing a seamless experience for users and stakeholders.

CHAPTER 6

RESULT ANALYSIS AND SCREENSHOTS

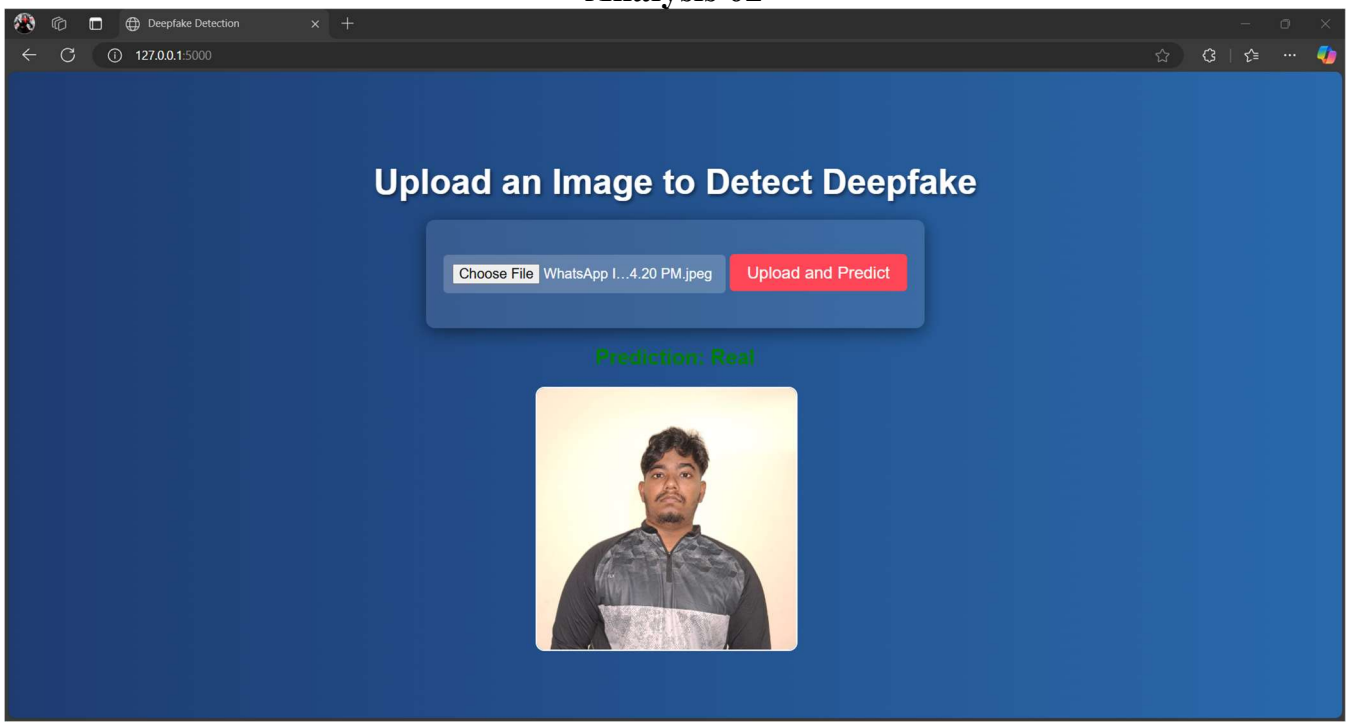
Analysis 01:



Elements: The Visit Page with option to upload or choose file and predict if it is fake or real image.

Fig 6.1: Landing Page

Analysis 02

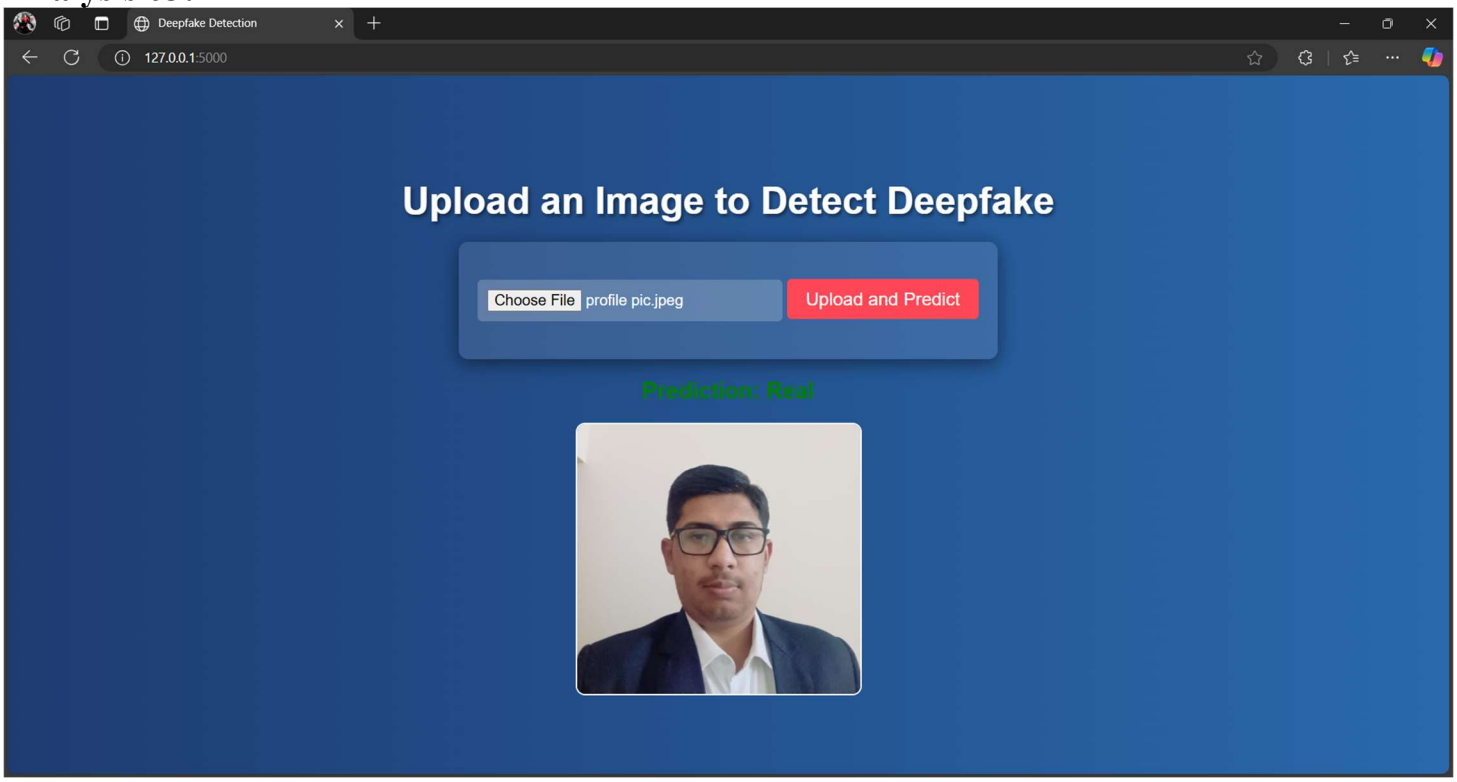


Elements: Uploaded an image of one of our project contributors

Observations: As seen the image is a real image and model successfully recognises it as real image

Fig 6.2: Real Image Test -Divyesh

Analysis 03:

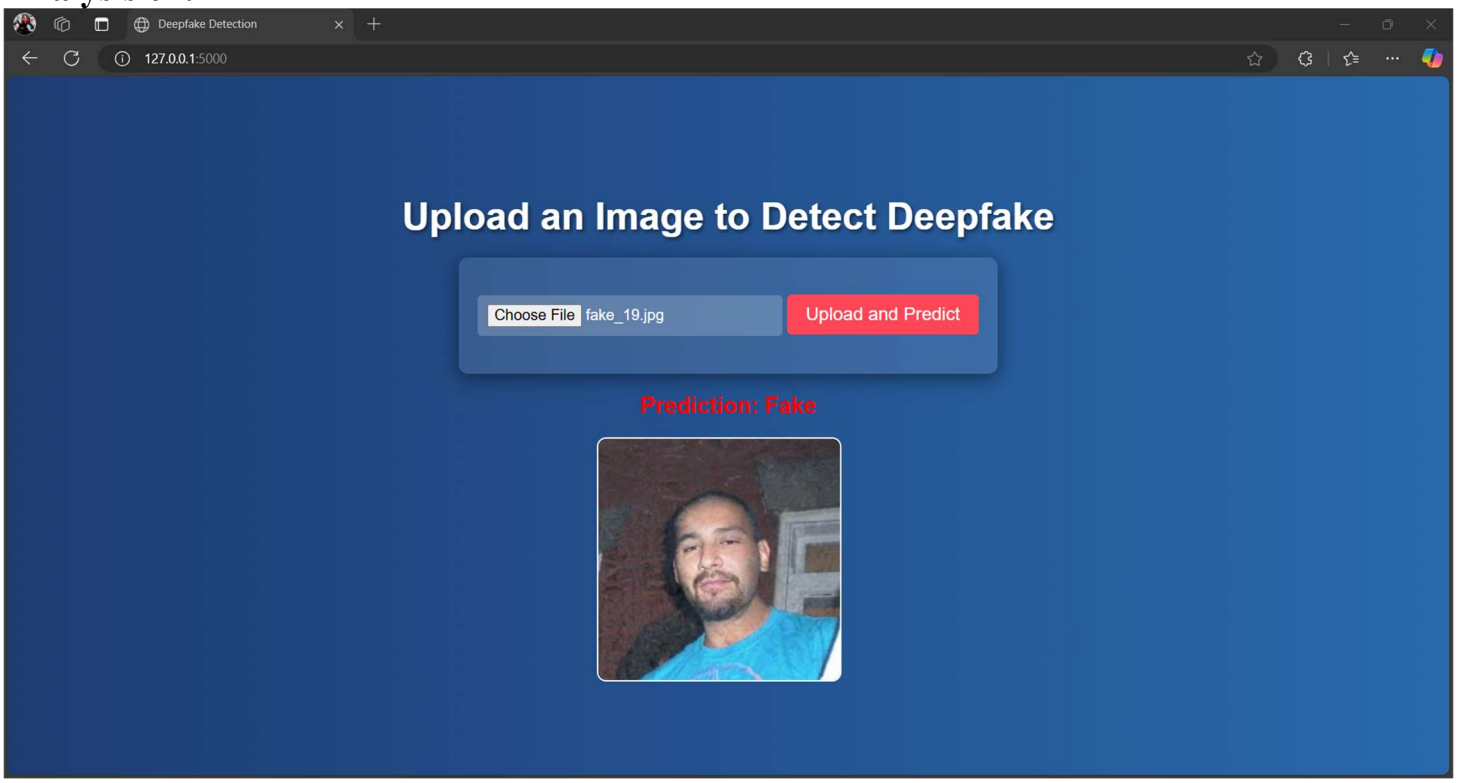


Elements: Uploaded an image of project contributors- Adarsh

Observations: As seen the image is a real image and model successfully recognises it as real image

Fig 6.3: Real Image Test -Adarsh

Analysis 04:

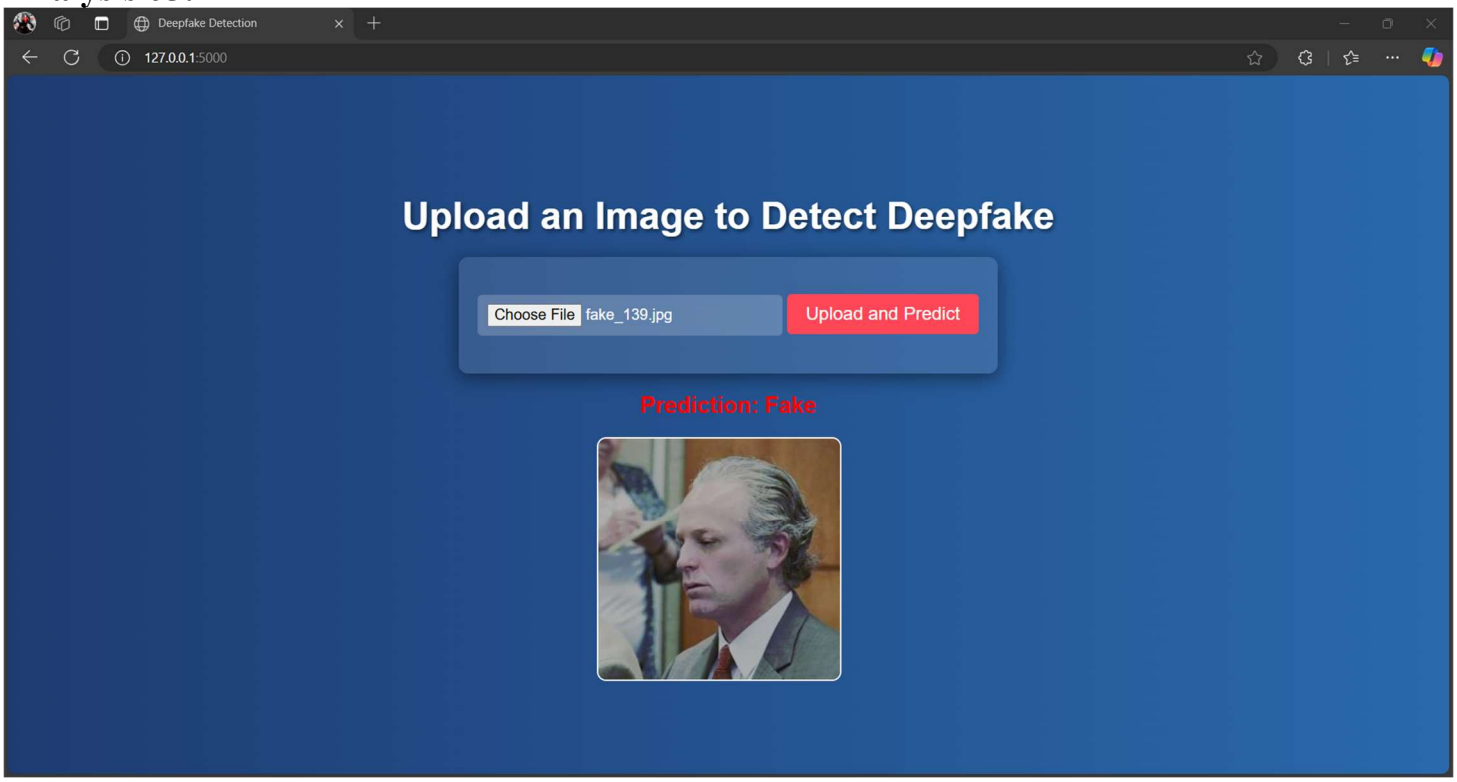


Elements: Upload a fake image from our test set

Observations: As expected the model detects it as a fake

Fig 6.4: Fake Image Test 1

Analysis 05:

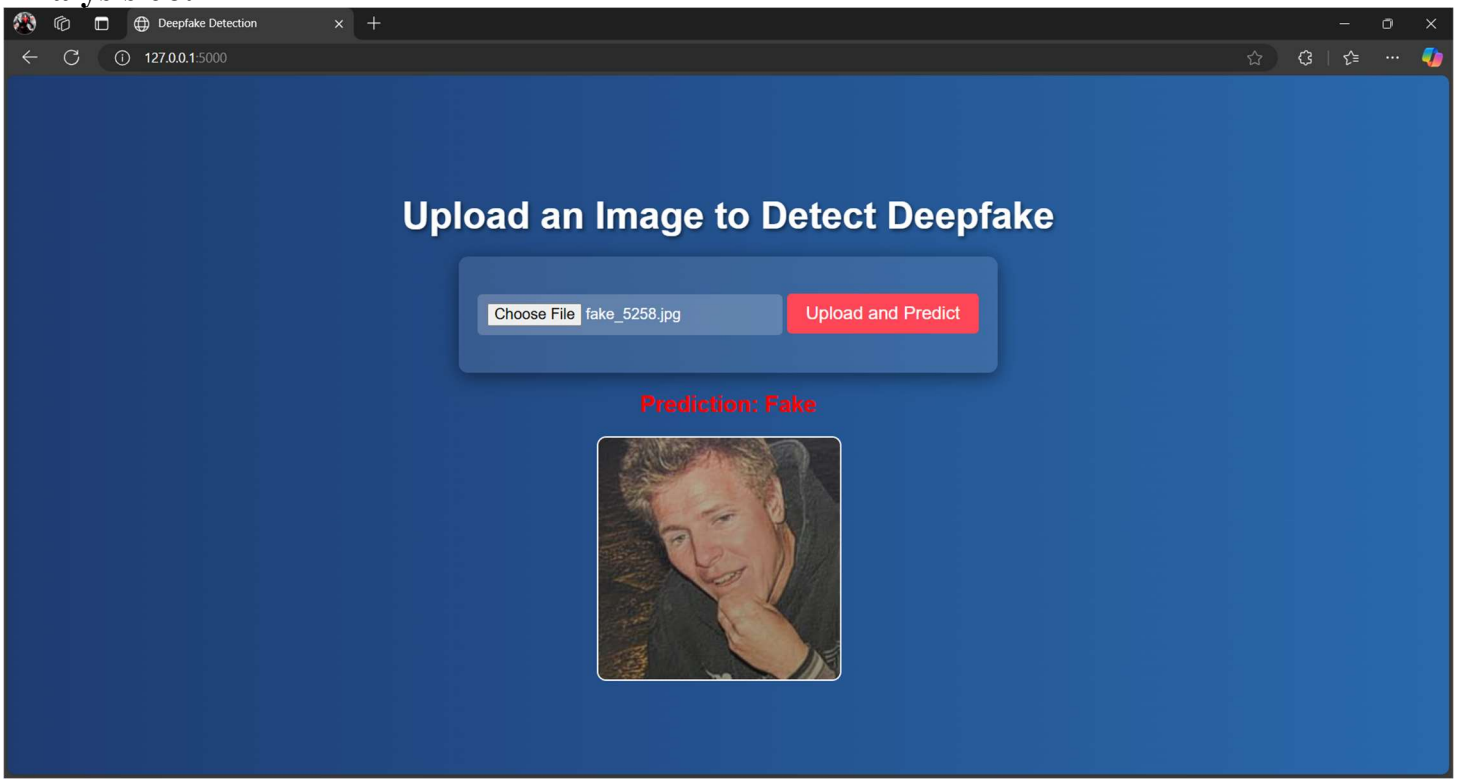


Elements: Upload a fake image from our test set

Observations: As expected the model detects it as a fake

Fig 6.5: Fake Image Test 2

Analysis 06:

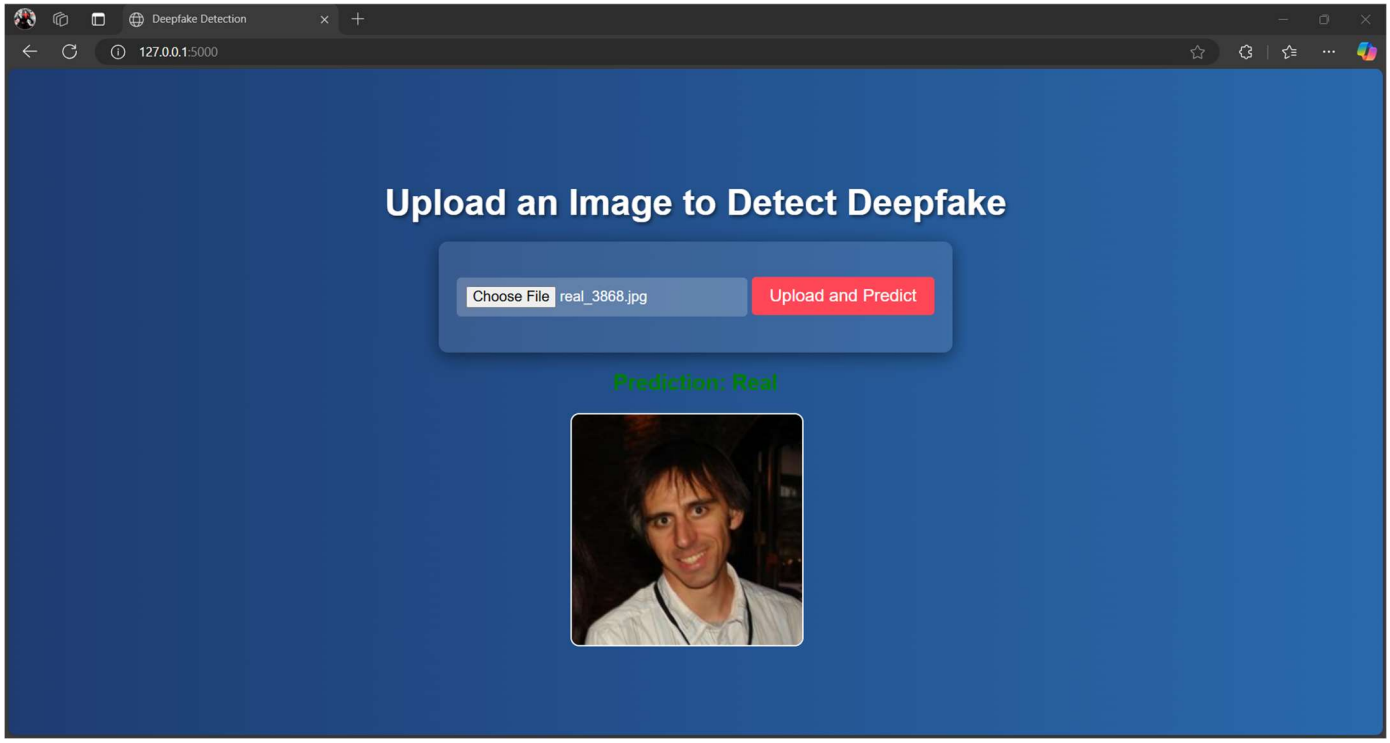


Elements: Upload a fake image from our test set

Observations: As expected the model detects it as a fake

Fig6.6 Fake Image Test 3

Analysis 07:

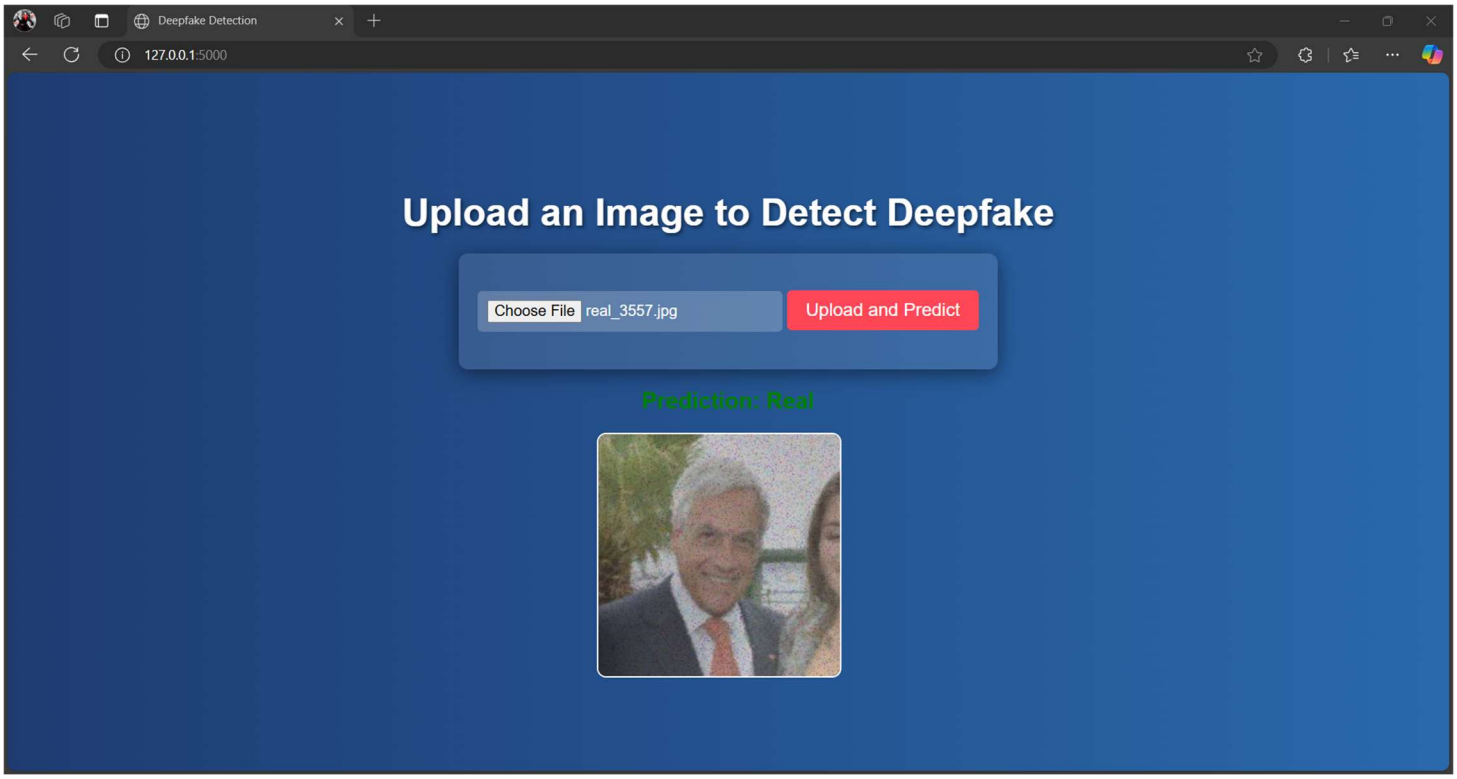


Elements: Upload a real image from our test set

Observations: As expected the model detects it as a real

Fig 6.7: Real Image test 3

Analysis 08:

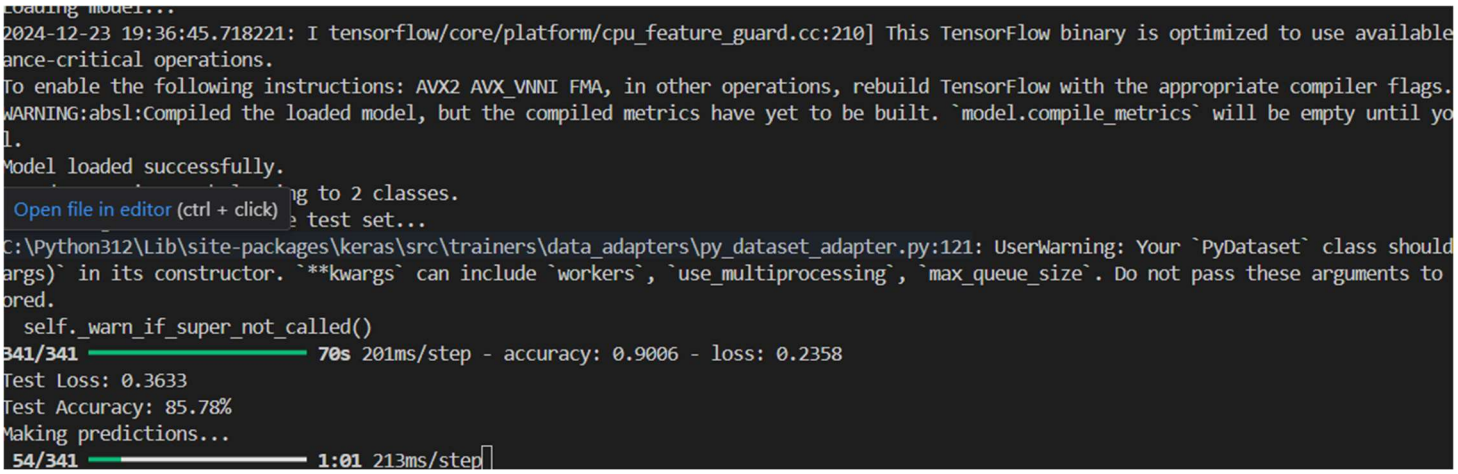


Elements:

Observations:

Fig 6.8: Real image test 4

Analysis 09:



Elements: Script evaluating model

Observations:Model Shows an accuracy of 85.78%

Fig 6.9: Evaluating Model

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

The Deepfake Detection Project represents a significant step toward combating the rising challenges posed by synthetic media manipulation. With a robust system leveraging state-of-the-art deep learning technologies, the project enables accurate and efficient identification of deepfake content. Its focus on usability, performance, and security ensures that the application serves as a reliable tool for users, ranging from casual individuals to organizations seeking to protect their integrity and mitigate risks.

This project not only contributes to safeguarding digital authenticity but also raises awareness about the ethical implications of AI advancements. By providing a user-friendly interface, responsive design, and secure data handling, the system stands as a comprehensive solution for addressing the misuse of AI-generated content.

Future Development Opportunities

The potential for improving and expanding the Deepfake Detection Project is vast. Some avenues for future development include:

1. **Integration with Video Deepfake Detection**

   Currently focused on images, the system can be enhanced to detect deepfake manipulations in videos. This would involve incorporating temporal analysis techniques and processing entire video streams efficiently.

2. **Real-Time Detection for Live Streams**

   Extending the functionality to analyze live video streams in real-time can make the tool invaluable for broadcasters and social media platforms to prevent the dissemination of harmful content.

3. **Advanced Deepfake Attribution**

   Future versions could include capabilities to trace the origin of a detected deepfake, helping authorities and organizations better understand the source and intent behind its creation.

4. **Improved Model Efficiency**

   Continuous optimization of the model to reduce computational requirements without compromising accuracy can make the system deployable on edge devices, such as smartphones or IoT devices.

5. **Multilingual and Multimodal Deepfake Detection**

   Enhancements can include detecting manipulations across diverse media, such as audio deepfakes or combined audio-visual manipulations, ensuring the tool is comprehensive.

6. **Educational and Awareness Modules**

   The system can incorporate features to educate users about deepfake technology, how to identify fake content, and the ethical implications of using such tools.

7. **Cross-Platform Integration**

   Developing APIs and plugins for integration into social media platforms, content management systems, or video editing tools would allow real-time deepfake checks.

8. **Adversarial Robustness**

   As deepfake techniques evolve, so should detection methods. Implementing adversarial training and updating the model with new datasets can ensure the system remains effective against emerging threats.

Final Thoughts

This project provides a strong foundation for addressing one of the critical challenges of the digital age. By fostering further innovation, integrating advanced functionalities, and maintaining an ethical approach, the Deepfake Detection Project can evolve into a versatile, scalable, and indispensable tool in the fight against misinformation and media manipulation.

BIBLIOGRAPHY

https://www.kaggle.com/datasets/manjilkarki/deepfake-and-real-images - The Dataset was taken from

https://ccoe.dsci.in/blog/deepfake-detection Provides better understanding of deepfakes.