“DETECTION OF DEEPFAKE VIDEOS”

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

(Specialization in Business Systems)

Submitted by

AGASTHYA T (VU21CSEN0200077)

POOJANA V (VU21CSEN0200106)

ANSHUL SIMHADRI (VU21CSWEN0200159)

RAM SALADI (VU21CSEN0200083)

Under the esteemed guidance of

Dr. V SANGEETA

Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

VISAKHAPATNAM

2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

DECLARATION

I hereby declare that the project report entitled "DETECTION OF DEEPFAKE VIDEOS" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering/ Computer Science and Engineering (AI&ML/DS/CS/IoT/BS). The work has not been submitted to any other college or University for the award of any degree or diploma.

Registration No(s)	Name(s)	Signature
VU21CSEN0200077	AGASTHYA T	Agasthya
VU21CSEN0200106	POOJANA V	poojana
VU21CSEN0200159	ANSHUL SIMHADRI	anshul
VU21CSEN0200083	RAM SALADI	ram saladi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled “DETECTION OF DEEPFAKE VIDEOS” is a bonafide record of work carried out by Agasthya T (VU21CSEN0200077), Poojana V (VU2CSEN0200106), Anshul Simhadri (VU21CSEN0200159) and Ram Saladi (VU21CSEN0200083) students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Business Systems.

Project Guide	Head of the Department
Dr V Sangeeta	Dr G. Lakshmeshwari

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to our esteemed guide, **Dr. V Sangeeta** whose invaluable guidance, continuous encouragement, and insightful feedback have been instrumental in the successful completion of this project. Her vast expertise in Artificial Intelligence and Machine Learning and web development, along with his patience and mentorship, empowered us with the skills and confidence needed to carry out this work.

TABLE OF CONTENTS

<u>Detection of Deepfake Videos</u>

1. ABSTRACT

Deepfake media, produced with advanced artificial intelligence methods, is a cause for increased concern due to its potential for malicious use in the spread of misinformation, identity theft, and online deception. With advancements in deepfake generation technology, conventional detection methods find themselves weakened. This paper introduces a deepfake detection method which combines Long Short-Term Memory(LSTM) networks and ResNext, to improve detection accuracy. ResNext is applied to extract intricate visual features, and LSTM preserves temporal dependencies within video frames to identify the difference between original and forged content. With the integration of spatial and temporal analysis, the suggested model enhances detection accuracy and robustness against emerging deepfake technologies. Experimental results show the system to detect deepfake content with high accuracy, helping towards more trustful media verification and digital security.

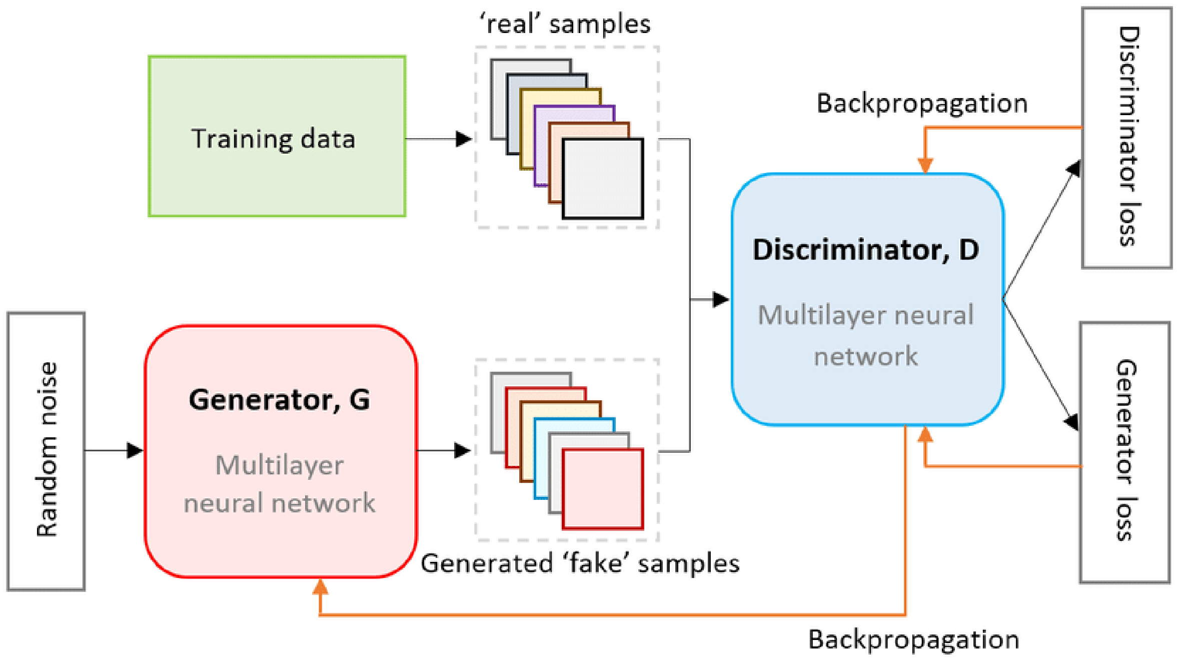
2. INTRODUCTION

The integrity of online media has been severely challenged by the development of deepfake technology, capable of spreading false information on a large scale, identity theft, and social media manipulation. Through deep learning models, in the form of Generative Adversarial Networks (GANs), deepfake methods create exceedingly realistic yet artificial audio, video, and images that are often identical to genuine content. This poses a critical threat to several sectors, such as politics, entertainment, and security, hence the need to make the development of reliable methods for detecting such information a priority. Although helpful in certain scenarios, conventional detection methods often struggle to keep pace with the increasing sophistication of deepfake models. Thus, there is a critical need for utilizing more advanced solutions that can evolve in response to developing manipulation methods. In this paper, we mix two powerful architectures of deep learning, ResNext and Long Short-Term Memory (LSTM) networks, to introduce an innovative approach towards deepfake identification.

For sequential data, such as videos, where temporal relationships between frames are critical to detecting anomalies characteristic of deepfakes, LSTM networks are particularly well-suited. Yet ResNext, an extension of residual networks, provides a helpful paradigm for learning hierarchical features, enhancing the model's ability to detect complex spatial patterns that could be indicative of manipulation. Our proposed model tries to improve detection performance and resilience in a broad range of deepfake manipulations by combining the spatial feature extraction abilities of ResNext with the temporal robustness of LSTM and achieves an accuracy rate of 88.983%. This work investigates the fusion of these two architectures, comparing their performance on deepfake datasets and contrasting the results with current detection techniques. The suggested model aims to add to the current research in deepfake detection by providing a hybrid approach that deals with both temporal and spatial features of deepfake media, giving a better tool for real-world use in media forensics and security.

3. LITERATURE REVIEW

Generative Adversarial Networks (GANs) are now an essential part in building of deepfake media as seen in 3.1 to generate super-realistic synthetic content. A GAN works with two principal neural network elements: a discriminator and a generator. The generator learns to build fake data—e.g., images or videos—that look like they belong to the real world by being trained on a dataset. It starts off processing noise at random and then gets progressively better at its outputs to be very similar to real data. For instance, a trained generator is able to generate a video of someone talking from a static picture alone. In contrast, the discriminator works to separate true data from a dataset from the generated synthetic data made by the generator. With repeated iterations, both the networks get better, with the generator generating more realistic content and the discriminator getting better at detecting fakes. After complete training, GANs can then be employed to create realistic videos in which people seem to say or do things they never actually said or did.



3.1 Generation of deepfakes using GAN

Though they have applications in areas such as film making and digital simulations, deepfake technologies are risky and involve serious threats such as misinformation, identity theft, and manipulation of the public. It takes huge computational power, large datasets, and sophisticated training methods to create high-quality deepfakes. The ethical issues with the use of GANs show the necessity of providing protection against malicious use. With advancements in deepfake generation, researchers looked for efficient detection techniques to counter the propagation of artificial media. Early detection techniques targeted the identification of visual anomalies like abnormal blinking, misaligned facial structures, and texture irregularities. The development of the FaceForensics++ dataset in 2017 served as a benchmark for measuring manipulated videos and was a major contributor to research in this area. But as deepfake models became more advanced, new detection mechanisms were needed.

Early detection methods utilized CNNs to scan individual frames for typical visual distortions. At the same time, RNNs were used to scan temporal inconsistencies within video sequences to detect anomalies such as out-of-sync lip movement and irregular blinking patterns. Problematic challenges like the 2018 DeepFake Detection Challenge (DFDC) challenged researchers to collaborate, leading to stronger detection models. With the passage of time, detection methods advanced from the use of CNN-based models to more sophisticated models like Transformers, which evaluate video sequences in an end-to-end manner. Furthermore, multimodal detection methods emerged that incorporate visual as well as audio analysis for identifying lip-syncing, speech, and facial mismatches. In an effort to increase detection accuracy even more, researchers utilized ensemble learning methods whereby different models receive combined to inspect visual, auditory, and temporal disparities in fabricated media. Biometric-inspired techniques, such as monitoring pulse and eye movement patterns, have also been researched as prospective deepfakes detectors due to the fact that such complex physiological signals are difficult for generating models to fake. In spite of these developments, detection of deepfakes is always a challenge since synthesis methods keep improving.

Deepfake detection has seen great improvements in real-time detection, with models created that can mark manipulated content as it is posted or streamed. Cross-domain detection methods have been enhanced, where models can detect deepfakes created using a range of architectures. Synthetic data augmentation has been developed to augment training datasets, and thus detection models can detect new deepfake manipulation techniques. Secondly, explainability of AI-based detection has come into the spotlight, especially in legal and journalistic use cases, where accountability and explainability are of prime importance. Latest deepfake detection models today utilize advanced architectures like XceptionNet, MesoNet, and Transformer-based models. XceptionNet, a derivative of the Inception model, utilizes depthwise separable convolutions for improved computational efficiency while achieving high accuracy in detecting deepfake anomalies.

MesoNet, specifically tailored for deepfake detection, takes a light-weight architecture with fewer layers but successfully captures both fine-grained and high-level facial features. Google DeepMind and IBM Watson are also organizations that have made contributions, leveraging reinforcement learning and explainable AI to boost deepfake detection. The progression of deepfake detection has moved from simple artifact-based to sophisticated, real-time multimodal strategies. This research introduces a hybrid detection model that combines ResNext, a deep convolutional neural network, and Long Short-Term Memory (LSTM) networks to enhance detection performance. ResNext is used to learn complex spatial features from video frames, while LSTM learns temporal relationships to identify inconsistencies in sequences.

The integration enables the model to process both static and dynamic properties, enhancing its capacity to distinguish real from manipulated content. For this study, the CelebDF dataset was employed, consisting of more than 1,000 videos of celebrities with authentic and deepfake samples. The dataset was split into original and manipulated videos, providing a well-organized procedure for training and testing. Videos were broken down into frames in preparation for data, allowing the model to scan detailed information in each frame. Frame consistency and quality were tuned during preprocessing for improved model performance. The ResNext-LSTM hybrid model was trained on CUDA-enabled GPUs to speed up computation, thereby supporting rapid processing of large datasets. Training was carried out in multiple epochs, fine-tuning the model parameters for enhanced accuracy. Optimization methods were employed to reduce loss and facilitate the model to generalize across various deepfake types. Performance metrics used were accuracy, precision, recall, and the Receiver Operating Characteristic (ROC) curve, which gave a complete evaluation of the effectiveness of the model. The results showed the model to be capable of detecting deepfake content with high precision, highlighting the potential of using ResNext and LSTM together for effective deepfake detection.

4. PROBLEM STATEMENT

The advent of deepfake media—artificially manipulated videos, photos, and audio produced through artificial intelligence—constitutes a serious challenge to digital security, the fight against misinformation, and authenticity online. Traditional detection methods have a hard time keeping pace with the increasingly sophisticated methods of deepfake generation, facilitating it to be widely disseminated without being detected. This project aims to create an advanced detection system using Long Short-Term Memory (LSTM) networks and ResNext, a convolutional neural network (CNN). Through the use of LSTM's capability to process temporal patterns in video sequences and ResNext's effectiveness in extracting complex visual features, the suggested model is expected to improve the accuracy, flexibility, and scalability of deepfake detection. This solution aims to reduce the risks posed by synthetic media, making the digital world a safer place.

Challenges Addressed by the Project:

Improved Detection Accuracy: Most current models are defeated by advanced sophisticated deepfake methods. The suggested methodology combines deep learning techniques to enhance precision and reliability.

Adaptation to Changing Deepfake Techniques: With the evolution of generative AI, conventional models lose their potency. The deep learning architecture in this project supports ongoing learning and adaptation to new manipulation methods.

Automated and Scalable Detection: Checking digital content manually is not effective in large-scale use. This system does it for you, allowing for quicker and more detailed screening.

4.1. OBJECTIVE

The objective of this project is to create a strong deepfake detection system by integrating Long Short-Term Memory (LSTM) networks and ResNext architecture. The system will be able to efficiently detect synthetic media (deepfakes) in video and image form, which has become a growing concern with the advancement of AI-generated content. To achieve this, the project will apply LSTM networks, an example of a recurrent neural network (RNN), to learn the temporal dependencies in video sequences. Concurrently, the ResNext architecture, a high-level convolutional neural network (CNN), will be used to learn the spatial features of individual video frames or images. The process will start with dataset collection and preprocessing that includes both real and deepfake video samples. Preprocessing includes frame extraction from the videos, resizing the images, pixel value normalization, and data augmentation for improving the robustness of the model while training. Next, the prepared dataset will be used to train the LSTM-ResNext hybrid model. The performance of the model will be measured using standard metrics such as accuracy, precision, recall, F1-score, and AUC. The aim is to obtain high accuracy in identifying deepfakes with minimal false positives and false negatives, proving the efficiency of this integrated approach for real-world use.

5. EXISTING SYSTEM AND PROPOSED SYSTEM

<u>**Existing Systems**</u>

1. **Traditional Computer Vision Approaches**
   - **Strengths**:
     - Early methods focus on low-level feature analysis like optical flow and facial landmarks.
     - Simple and fast for detecting basic manipulations.
   - **Weaknesses**:
     - Struggles with modern deepfakes (e.g., GAN-based fakes).

        - Limited scalability to complex, high-quality fakes.
2. **Machine Learning-Based Approaches**
    - **Strengths**:
        - Uses handcrafted features like facial expressions, head movements, and frame inconsistencies.
        - Can handle moderate manipulation techniques.
    - **Weaknesses**:
        - Limited generalization to novel deepfake techniques.
        - Dependent on the quality of handcrafted features and datasets.
3. **Deep Learning-Based Approaches (CNNs, RNNs)**
    - **Strengths**:
        - Advanced models (e.g., CNNs, RNNs) can learn complex patterns from raw data.
        - Strong performance in detecting subtle and advanced fakes.
    - **Weaknesses**:
        - High computational cost.
        - May struggle with real-time detection and handling unseen manipulation techniques.
        - Vulnerable to adversarial attacks.

Proposed System

The proposed system aims to improve deepfake detection by combining two state-of-the-art deep learning methods: Long Short-Term Memory (LSTM) and ResNeXt. The hybrid framework is intended to provide a more accurate, efficient, and robust solution for deepfake content identification.

LSTM for Temporal Analysis

LSTM networks, a type of Recurrent Neural Networks (RNNs), are well-suited to capture long-range dependencies in sequential data and hence are especially useful for video-based deepfake detection. Their capacity to remember information over several time steps enables the model to examine temporal inconsistencies between video frames. This encompasses detecting unnatural facial movements, bad lip-syncing, or abnormal blinking patterns, which are usually subtle signs of deepfakes. Although single frames might look believable, LSTM's temporal examination ensures inconsistencies between frames are picked up, enhancing detection rates.

ResNeXt for Feature Extraction

ResNeXt, a deep learning network established on residual networks, is the feature extractor here in this suggested system. It is best at extracting complex and high-level features from video frames using its capability to deal effectively with networks with high numbers of parameters. This framework captures both high-level details as well as wider visual inconsistencies, e.g., texture deformities, lighting inconsistencies, or abnormal facial geometries, that are usual in deepfakes. With ResNeXt, the framework is able to distinguish between subtle visual distortions characteristic of edited videos.

Hybrid Model

By integrating LSTM and ResNeXt, the system constructs a hybrid model that leverages the advantages of both methods. ResNeXt analyzes every video frame to identify useful features, which are fed into the LSTM network. The LSTM network conducts temporal analysis, identifying inconsistencies between successive frames. The hybrid architecture enables the system to solve both spatial and temporal artifacts, offering an all-around solution to deepfake detection.

Advantages of the Proposed System

The proposed LSTM-ResNeXt hybrid model offers numerous advantages compared to traditional systems:

- **Enhanced Accuracy**: The system benefits from a dual focus on spatial features and temporal dependencies, enabling it to identify even the most subtle deepfake artifacts.
- **Improved Generalization**: The model's ability to learn from both individual frames and the relationships between frames enhances its adaptability to new, unseen deepfake generation techniques.
- **Scalability**: The system is designed to scale efficiently, capable of training on diverse datasets to detect a wide variety of deepfake manipulation methods.

In brief, the new hybrid method, which uses LSTM for temporal processing and ResNeXt for feature learning, promises much improvement over current deepfake detection systems. The system hopes to further improve both accuracy and real-time capability in detecting deepfakes and overcome the weakness of current approaches while offering a more trustworthy device to counter the impending threat of deepfakes.

5.1 REQUIREMENT ANALYSIS

5.1.1 Hardware Requirements

Processor (CPU): A quad-core or greater multi-core processor is needed to effectively handle parallel computing tasks.

Graphics Card (GPU): A specialized graphics processor is highly recommended for deep learning tasks. NVIDIA GPUs with CUDA support is well-supported and widely used for such tasks.

Memory (RAM): At least 16 GB of RAM is recommended to handle the computational load during model training and data processing.

Storage: Sufficient disk space is required for storing datasets, model files, and other resources. Solid-State Drives (SSDs) are preferred for faster read/write speeds.

Network Connectivity: A stable internet connection is important for accessing remote datasets, downloading pre-trained models, and updating dependencies.

5.1.2 Software Requirements

Operating System:

- **Supported Platforms:** Windows, Linux, macOS
- **Recommended Distribution:** Ubuntu 20.04 LTS, particularly for optimized GPU compatibility

Python Environment:

- **Version:** Python 3.8 or newer
- **Environment Management:** pip or conda

Essential Libraries and Tools:

Library	Minimum Version	Purpose
PyTorch	1.10.0	Core deep learning framework (ResNeXt-50 + LSTM)
torchvision	0.11.0	Pretrained models and image/video processing
OpenCV	4.5.0	Frame extraction and image preprocessing
NumPy	1.21.0	Numerical operations and matrix handling
Gradio	3.0.0	Web-based user interface for model deployment
scikit-learn	1.0.0	Evaluation metrics such as ROC-AUC and confusion matrix

Hardware Acceleration:

- **GPU:** NVIDIA graphics card with at least 8 GB VRAM and CUDA compatibility
- **CUDA Toolkit:** Version 11.3
- **cuDNN Library:** Version 8.2.0
- **Alternative:** CPU-only mode (slower performance)

Additional Utilities:

- **FFmpeg:** Required for processing various video formats (install via apt-get install ffmpeg)
- **Google Colab:** Optional cloud platform for training and experimentation

System Resources:

- **RAM Requirements:** Minimum 8 GB for training; 2 GB for inference
- **Disk Space:** At least 5 GB for datasets (e.g., Celeb-DF) and 2 GB for storing model weights

5.1.3 Functional Requirements

The system is designed to effectively detect deepfake videos and provide a clear classification with high reliability. The core functionalities are as follows:

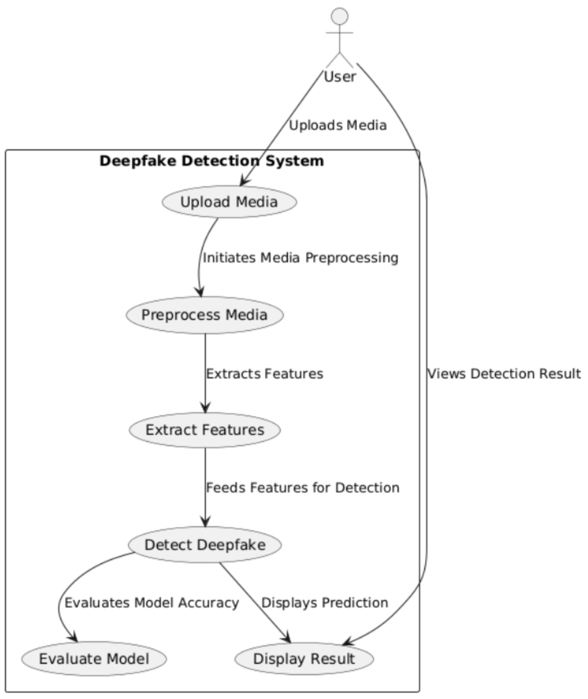
1. **Detection Capability:** Accurately identify and classify videos as either **"Real"** or **"Fake"** using a trained deep learning model.
2. **User Interface:** A simple and intuitive interface is provided via **Gradio**, enabling easy interaction for video upload and result visualization.
3. **Input Format:**
    - Accepts video files in **MP4** format.
    - Videos must be **face-centric**, aligning with datasets like Celeb-DF.
    - A minimum of **20 frames per video** is required for valid analysis.
4. **Output:**
    - Provides a **confidence score** indicating the likelihood of the video being real or fake.
    - Displays the classification result alongside the score for interpretability.
5. **Performance Benchmark:**
    - The system is designed to achieve atleast **85% classification accuracy**, based on the evaluation metrics of the trained model.

6. **Operational Constraints:**
   - The detection system is designed specifically for **face-centered deepfake videos** and might not perform best on other video types or formats.

5.2 SYSTEM DESIGN

1. Use Case Diagram:

This diagram as seen in 5.2.1 illustrates the interaction between the user and the system throughout the deepfake detection process.



5.2.1 Use case diagram for the deepfake detection model combining LSTM and ResNext

Use Case Summary:

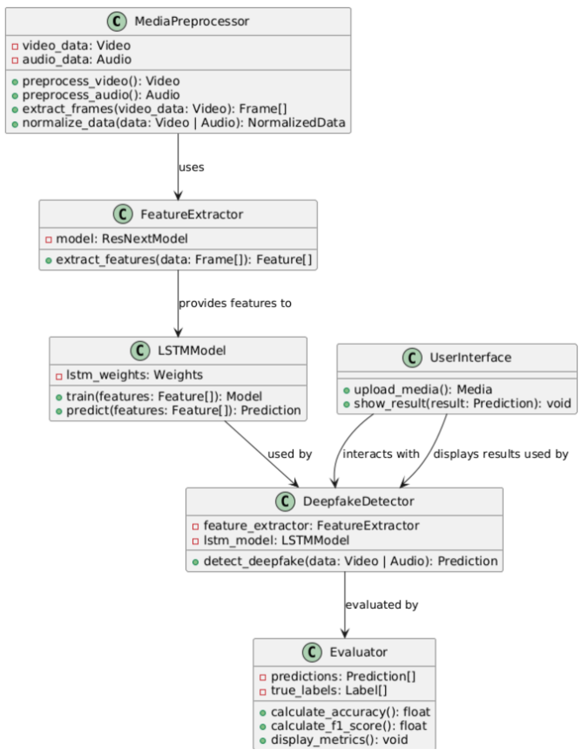
The process is initiated by the user uploading a video file. The system carries out a series of operations to classify and analyze the video:

1. **Media Upload**: The user supplies a video input (e.g., MP4 format) via the interface.
2. **Preprocessing**: The system extracts and processes frames that are relevant, targeting facial regions for further processing.
3. **Feature Extraction**: Important features are extracted from the video frames via deep learning models.
4. **Deepfake Detection**: The model uses a trained model (e.g., ResNeXt + LSTM) to determine if the video content is real or not.
5. **Model Evaluation**: The prediction is evaluated on pre-specified metrics (e.g., confidence score, accuracy).

6. **Result Delivery**: The classification outcome ("Real" or "Fake") and the confidence score are presented to the user through a user interface.

2. Class Diagram:

The class diagram as seen in 5.2.2 outlines the structural design and key components of the deepfake detection system, representing how different modules interact to achieve the overall functionality.


MediaPreprocessor
video_data: Video
audio_data: Audio
preprocess_video(): Video
preprocess_audio(): Audio
extract_frames(video_data: Video): Frame[]
normalize_data(data: Video | Audio): NormalizedData
uses
FeatureExtractor
model: ResNextModel
extract_features(data: Frame[]): Feature[]
provides features to
LSTMModel
lstm_weights: Weights
train(features: Feature[]): Model
predict(features: Feature[]): Prediction
UserInterface
upload_media(): Media
show_result(result: Prediction): void
used by
interacts with
displays results used by
DeepfakeDetector
feature_extractor: FeatureExtractor
lstm_model: LSTMModel
detect_deepfake(data: Video | Audio): Prediction
evaluated by
Evaluator
predictions: Prediction[]
true_labels: Label[]
calculate_accuracy(): float
calculate_f1_score(): float
display_metrics(): void

5.2.2 Class diagram for the deepfake detection model combining LSTM and ResNext

Core Components:

1. **MediaPreprocessor**
   - Responsible for handling raw video (or audio) input.
   - Performs frame extraction, resizing, normalization, and other preprocessing steps to prepare the data for feature extraction.
2. **FeatureExtractor**
   - Utilizes a deep learning architecture (e.g., ResNeXt) to extract spatial features from individual frames.
   - Converts raw pixel data into high-level representations for further analysis.
3. **LSTMModel**
   - A temporal model that processes sequential frame features.
   - Trains on extracted features and predicts the probability of the content being real or fake.
4. **DeepfakeDetector**

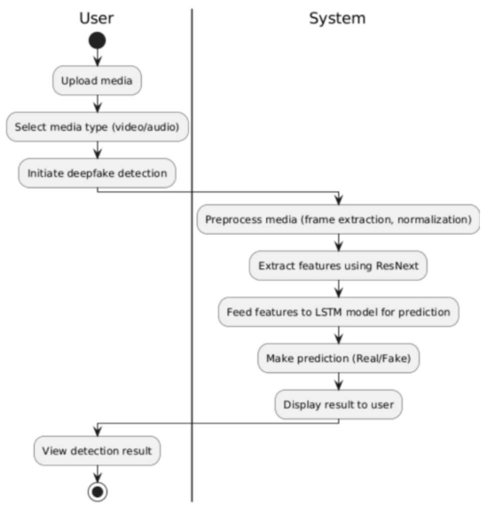
    - Acts as the main orchestrator, integrating the FeatureExtractor and LSTMModel.
    - Manages the overall detection pipeline from input preprocessing to final classification.
5. **UserInterface**
    - Handles interaction with the user through a front-end interface (e.g., Gradio).
    - Facilitates video uploads and displays results (classification and confidence score).
6. **Evaluator**
    - Calculates and reports performance metrics such as **accuracy**, **F1-score**, and **confusion matrix**.
    - Supports ongoing model evaluation and benchmarking.

3. Activity Diagram:

The activity diagram as seen in 5.2.3 illustrates the step-by-step workflow of the deepfake detection system, highlighting the decision-making process involved in classifying media as real or fake.

Process Flow:

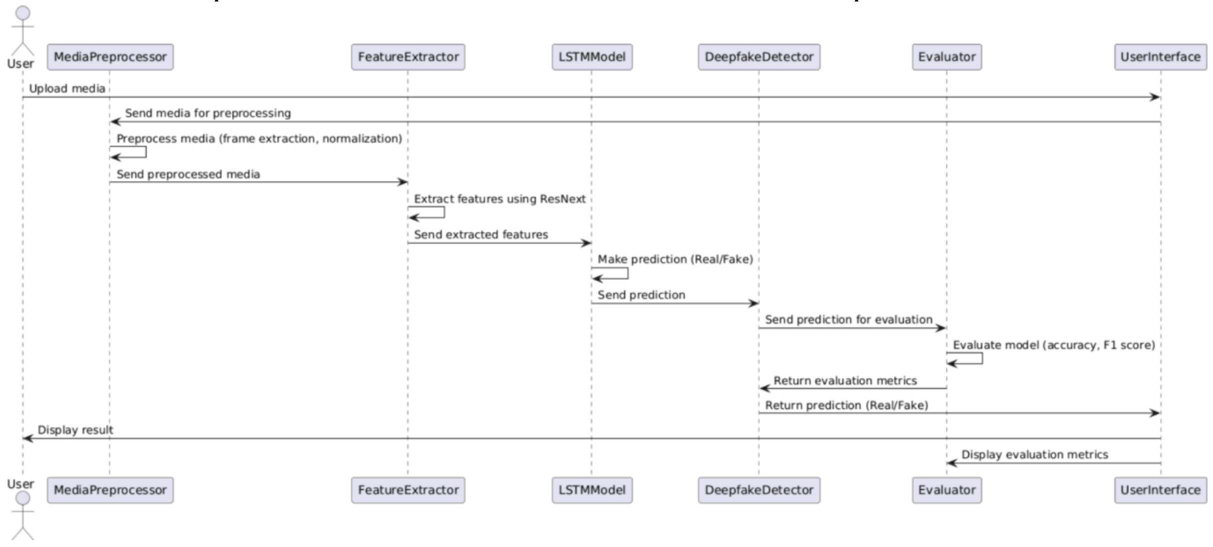
1. **Media Submission:** The user uploads a media file (e.g., video or audio) through the user interface.
2. **Validation Check:** The system verifies if the uploaded file is a valid and supported media format.
    - If invalid, the process terminates with an error or prompt for correct input.
    - If valid, the system proceeds to the next step.
3. **Preprocessing:** The media is preprocessed—this may include frame extraction, resizing, normalization, or face detection (for video).
4. **Feature Extraction**: Appropriate features are extracted from the preprocessed data by utilizing a deep learning model like ResNeXt.
5. **LSTM-based Classification**: The sequential features are fed into the LSTM model, where temporal patterns are examined in order to classify the media.
6. **Prediction Output**: The model provides a classification label: "Real" or "Fake" and a confidence score.
7. **Evaluation**: Accuracy and F1-score performance metrics are computed to assess the model's reliability.
8. **Result Display**: The last classification and evaluation measures are displayed to the user through the interface.


User
System
Upload media
Select media type (video/audio)
Initiate deepfake detection
Preprocess media (frame extraction, normalization)
Extract features using ResNext
Feed features to LSTM model for prediction
Make prediction (Real/Fake)
Display result to user
View detection result

5.2.3 Activity diagram for the deepfake detection model combining LSTM and ResNext

4. Sequence Diagram:

The sequence diagram as seen in 5.2.4 illustrates the flow of interactions between system components during the deepfake detection process. It captures the chronological order in which each component communicates to complete the task.


User
MediaPreprocessor
FeatureExtractor
LSTMModel
DeepfakeDetector
Evaluator
UserInterface
Upload media
Send media for preprocessing
Preprocess media (frame extraction, normalization)
Send preprocessed media
Extract features using ResNext
Send extracted features
Make prediction (Real/Fake)
Send prediction
Send prediction for evaluation
Evaluate model (accuracy, F1 score)
Return evaluation metrics
Return prediction (Real/Fake)
Display result
Display evaluation metrics

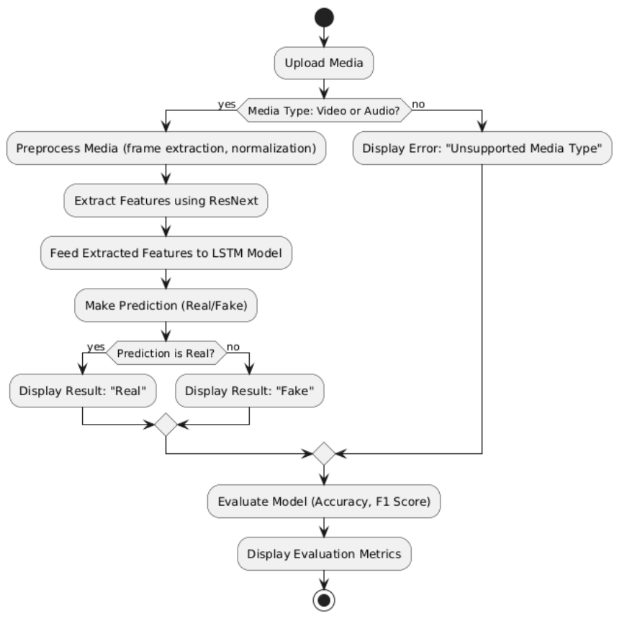
5.2.4 Sequence diagram for the deepfake detection model combining LSTM and ResNext

Interaction Flow:

1. **User Uploads Media**: The user begins by uploading an audio or video file via the interface.
2. **Preprocessing Stage**: The uploaded media is forwarded to the Media Preprocessor, which carries out such tasks as frame extraction, resizing, and normalization in order to prepare the data for analysis.

3. **Feature Extraction**: Processed frames are fed into the Feature Extractor, which employs a deep learning backbone (e.g., ResNeXt) to extract useful features from the media.
4. **Prediction**: The sequential features are passed to the LSTM Model, which examines temporal patterns and predicts whether the content is real or not.
5. **Detection & Coordination**: Deepfake Detector orchestrates the whole process, facilitating coordination among preprocessing, feature extraction, and prediction modules. It retrieves the prediction output for further inspection.
6. **Evaluation:** The Evaluator receives the output and computes relevant performance metrics such as accuracy to assess the model's effectiveness.
7. **Result Presentation:** Finally, the classification result and confidence score are displayed to the user via the interface.

5. Flowchart Diagram:


Upload Media
yes
Media Type: Video or Audio?
no
Preprocess Media (frame extraction, normalization)
Display Error: "Unsupported Media Type"
Extract Features using ResNext
Feed Extracted Features to LSTM Model
Make Prediction (Real/Fake)
yes
Prediction is Real?
no
Display Result: "Real"
Display Result: "Fake"
Evaluate Model (Accuracy, F1 Score)
Display Evaluation Metrics

5.2.5 Flowchart diagram for the deepfake detection model combining LSTM and ResNext

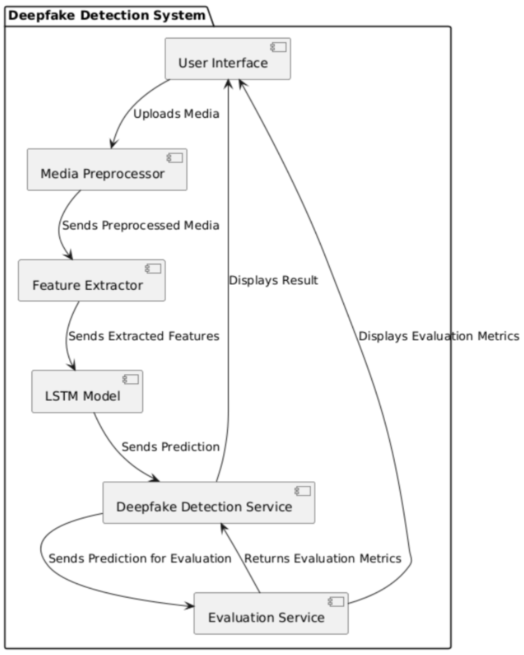
- **User Uploads Media** – The system receives a video or audio file.
- **Media Validation** – It checks whether the file is a supported format.
  - If valid, it proceeds to preprocessing.
  - If unsupported, an error message is displayed.
- **Preprocessing** – The media undergoes frame extraction and normalization to prepare for feature analysis.
- **Feature Extraction** – The system extracts key features from the frames using a **ResNeXt** deep learning model.
- **Deepfake Classification** – Extracted features are passed into an **LSTM Model**, which predicts whether the content is **Real** or **Fake**.
- **Result Display** – The system presents the classification result (Real or Fake) to the user.
- **Evaluation Metrics** – To assess performance, the system calculates and displays key metrics like **accuracy**

6. Component Diagram:

The component diagram as seen in 5.2.6 presents the high-level architecture of the deepfake detection system, outlining the main modules and how they interact to process media files and determine authenticity.

Core Components:

1. **User Interface:**
    - Front-end component (e.g., Gradio) through which users upload media files and receive detection results.
    - Facilitates communication between the user and backend services.
2. **Media Preprocessor:**
    - Handles input validation and media preparation tasks such as frame extraction, resizing, and normalization.
    - Ensures the media is in a suitable format for feature analysis.
3. **Feature Extractor:**
    - A deep learning component (e.g., based on ResNeXt) responsible for extracting distinguishing visual or auditory features from the preprocessed media.
    - Converts raw frames into structured data.
4. **LSTM Model:**
    - A sequence-based neural network model that analyzes the temporal sequence of features to detect deepfake patterns.
    - Outputs classification probabilities.
5. **Deepfake Detection Service:**
    - The central orchestration unit that integrates the Feature Extractor and LSTM Model.
    - Manages the full detection pipeline and returns raw results for evaluation.
6. **Assessment Service:**
    - Evaluates model performance using metrics such as **accuracy** and **confusion matrix**.
    - Provides feedback on model reliability and confidence.



5.2.6 Component diagram for the deepfake detection model combining LSTM and ResNext

6. Methodology

The research in this paper presents a hybrid deepfake model that utilizes ResNext for spatial feature extraction and Long Short-Term Memory (LSTM) networks for temporal inconsistency analysis in video sequences. The methodology involves various stages of paramount importance: data collection and preprocessing, feature extraction, model structure, training and optimization, and performance measurement. All the components are important in ensuring accurate and efficient detection of manipulated media.

1. Data Collection and Preprocessing:

Mission: Organize high-quality input data for detection model training. The dataset includes real and deepfake videos, each of which is labeled as authentic or manipulated.

Frame Extraction: The videos are transformed into separate frames to facilitate both spatial and temporal analysis of features. There is a consistent frame sampling rate for uniformity.

Data Augmentation: Rotation, brightness adjustment, and noise addition are applied through image processing techniques to increase variability and avoid overfitting.

Normalization: Frames are resized and pixel values are normalized to a fixed range, making them optimized for deep learning models.

2. Feature Extraction Using ResNext:

Objective: Extract detailed visual information to identify real and fake media.

ResNext, which is a convolutional neural network (CNN), is used to extract intricate spatial features from every frame.

The model identifies inconsistencies such as texture irregularities, unnatural lighting, and subtle distortions in facial structures.

Extracted features are saved in numerical vectors and passed on to the LSTM model for further examination.

A transfer learning approach is employed, fine-tuning a pre-trained ResNext model to enhance accuracy while reducing computational load.

3. Temporal Analysis Using LSTM:

Objective: Identify inconsistencies in video sequences by capturing temporal dependencies.

Since deepfake videos often exhibit unnatural movements, LSTM is employed to analyze frame-by-frame changes.

Feature vectors from ResNext are processed through LSTM layers, which detect irregularities in facial expressions, blinking patterns, and lip synchronization.

LSTM's memory function retains significant historical information, allowing the model to learn long-term dependencies and recognize fake content.

This module effectively flags unnatural transitions that are challenging to detect in individual frames.

4. Model Architecture and Training:

Objective: Train the hybrid deepfake detection model for high accuracy and robustness.

The model architecture integrates ResNext for feature extraction, LSTM for sequential analysis, and a fully connected layer for classification.
Input data for training are labelled real and manipulated videos.

Loss Function: A binary classification method is employed with cross-entropy loss to reduce error.

Optimization Strategy: The Adam optimizer optimizes the learning rate dynamically and improves model convergence.

Batch Processing: Training is done with mini-batches to enhance computational efficiency.

Hardware Utilization: GPU-based parallel computing is utilized to accelerate training, facilitating faster and more efficient learning.

5. Performance Evaluation:

Objective: Measure the model's effectiveness in detecting deepfakes.

Confusion Matrix: Utilized to measure correctly and wrongly identified samples.

Accuracy: Calculated to determine the overall performance of the model.

Receiver Operating Characteristic (ROC) Curve: Utilized to determine the differentiation ability of real and fake videos based on multiple thresholds.

Comparison with Other Models: The current method is compared to traditional deepfake detection models to determine enhanced accuracy.
Future development can include multimodal analysis, combining audio and facial movement analysis for better precision.

7. TECHNOLOGIES USED

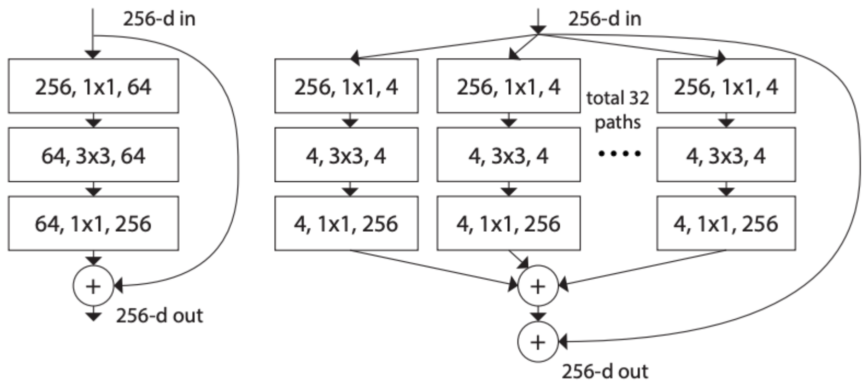
1. ResNeXt

ResNeXt is a deep learning model that extends residual networks by adding cardinality — the number of parallel paths within a block. This allows the model to learn more detailed feature representations without incurring much higher computational cost.

Advantages:

- Effective feature extraction from video frames.
- Scalable for processing large datasets.
- Catches intricate patterns such as texture, lighting, and facial anomalies, which are characteristic of deepfakes.

Role in Deepfake Detection: ResNeXt is employed to extract high-resolution features from video frames, assisting in finding visual anomalies like unnatural textures or anomalous facial geometry that are characteristic of deepfakes.



7.1 ResNext architecture

2. Long Short-Term Memory (LSTM)

LSTM is a recurrent neural network (RNN) architecture that is specifically used to learn long-term dependencies in sequential data. It is particularly good at handling time-series data, which makes it well-suited for video sequence analysis.

Advantages:

- Maintains temporal context through memory cells.
- Treats long-term dependencies within sequential data.

LSTM examines temporal movement between frames of a video, identifying inconsistencies such as unnatural lip motion or out-of-sync actions across frames, which are prevalent in deepfakes.

3. Adam Optimizer

Adam (Adaptive Moment Estimation) is a popular optimization algorithm that adjusts learning rates per parameter and balances the strengths of both momentum and adaptive learning rate approaches.

- **Advantages**:
    - **Adaptive learning rate** for faster convergence.
    - **Handles sparse gradients** well, improving training efficiency.
    - **Bias correction** ensures better accuracy, especially in early training stages.
- **Role in Deepfake Detection**: Adam optimizes the training of both ResNeXt and LSTM models, enabling efficient convergence and better model performance during deepfake detection training.

4. Libraries Used:

Library	Minimum Version	Purpose	Installation Command
PyTorch	1.10.0	Deep learning framework (ResNeXt-50 + LSTM)	pip install torch torchvision
torchvision	0.11.0	Pretrained models &amp; video transforms	(Included with PyTorch)
OpenCV (cv2)	4.5.0	Video frame extraction &amp; preprocessing	pip install opencv-python
NumPy	1.21.0	Numerical operations	pip install numpy
Gradio	3.0.0	Web interface for model deployment	pip install gradio
scikit-learn	1.0.0	Metrics (ROC-AUC, confusion matrix)	pip install scikit-learn
pandas	1.3.0	Metadata handling (CSV/DataFrames)	pip install pandas
tqdm	4.62.0	Progress bars for training loops	pip install tqdm

8. IMPLEMENTATION

Sample code:

```
# Model
from torch import nn
from torchvision import models
class Model(nn.Module):
  def __init__(self, num_classes,latent_dim= 2048, lstm_layers=1 , hidden_dim = 2048,
bidirectional = False):
      super(Model, self).__init__()
      model = models.resnext50_32x4d(pretrained = True) #Residual Network CNN
      self.model = nn.Sequential(*list(model.children())[:-2])
      self.lstm = nn.LSTM(latent_dim,hidden_dim, lstm_layers,  bidirectional)
      self.relu = nn.LeakyReLU()
      self.dp = nn.Dropout(0.4)
      self.linear1 = nn.Linear(2048,num_classes)
      self.avgpool = nn.AdaptiveAvgPool2d(1)


   def forward(self, x):
      batch_size,seq_length, c, h, w = x.shape
      x = x.view(batch_size * seq_length, c, h, w)
      fmap = self.model(x)
      x = self.avgpool(fmap)
      x = x.view(batch_size,seq_length,2048)
      x_lstm,_ = self.lstm(x,None)
      return fmap,self.dp(self.linear1(torch.mean(x_lstm,dim = 1)))
```

9. TESTING

White-Box Testing

In the context of our deepfake detection system, white-box testing was applied to checking the internal structure and code logic in deepfake detection models. We scrutinized the PyTorch implementation's building blocks which included the CNN backbone ResNeXt-50, LSTM units, and the classification head. Unit tests validated that feature extraction was done correctly by ResNeXt-50 lifting 2048 dimensional features from every frame, and integration tests ensured that the LSTM processed the features in sequence over 20 frames without mismatch in dimensions. During backpropagation, we supervised the gradient explosion/vanishing control mechanisms and softmax output probabilities, confirming they added up to 1.0. Code coverage (which includes 'pytest-cov') had goals of more than 90% coverage and more than 80% covcred on critical paths such as frame sampling and loss computation. These methods confirmed that the model was functioning as intended under set conditions.

Black-Box Testing

In black-box testing, the system is treated as a black box; functionally and behaves like a user with no prior knowledge interacts. With actual user interaction through the Gradio interface, we input validated videos (5 real & 5 fake) alongside invalid videos like corrupt files and unsupported files (.txt) to analyze error responses. Some benchmarks of performance also included measuring inference time (<3 seconds/video on GPU) and memory under load (simultaneous user requests). Accuracy functionality also underwent testing to verify it surpassed the mark of 85% on incoming data.

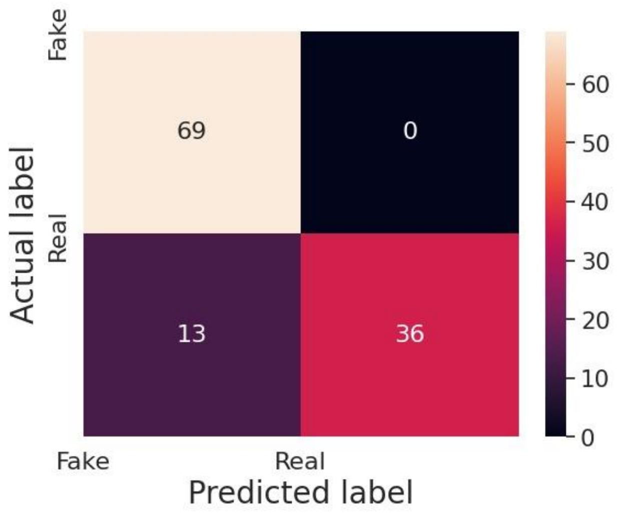
Synergy Between Approaches

The model was validated mathematically via white-box testing (e.g. LSTM state transitions), and validated practically via black-box testing (e.g. Gradio responsiveness). For example, some white-box testing during the development phase would identify LSTM dimension errors, while black-box testing uncovered very real failures like high-quality deepfakes being misclassified. Collectively, these techniques delivered holistic quality evaluation from low-level code scrutiny to user-facing dependability, achieving an accuracy of 88%. This approach, along with all its reproducibility documentation, is recorded within the project repository where test scripts and logs are stored.

Video ID	Actual Label	Predicted Label	Confidence Score
01	Fake	Real	99.25%
02	Fake	Fake	99.28%
03	Fake	Fake	99.99%
04	Fake	Fake	99.37%
05	Real	Real	100%
06	Real	Real	100%
07	Real	Real	100%

10. RESULTS & DISCUSSIONS

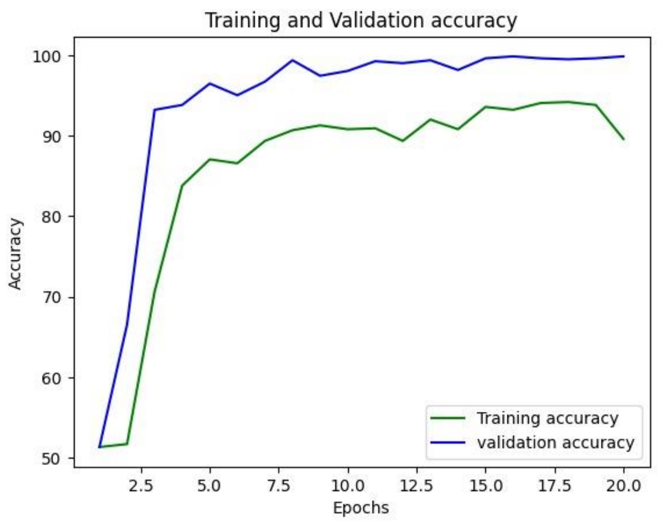
1. Confusion Matrix:



10.1 Confusion matrix for the deepfake detection model

- This table gives a clear picture of how well the model classifies deepfake and real media.
- The breakdown:
  - **69** fake samples correctly identified as fake (**true positives**).
  - **36** real samples correctly identified as real (**true negatives**).
  - **13** real samples mistakenly classified as fake (**false negatives**).
  - **0** fake samples mistakenly classified as real (**false positives**).
- The model is very good at spotting deepfakes but struggles a bit with correctly identifying real content.

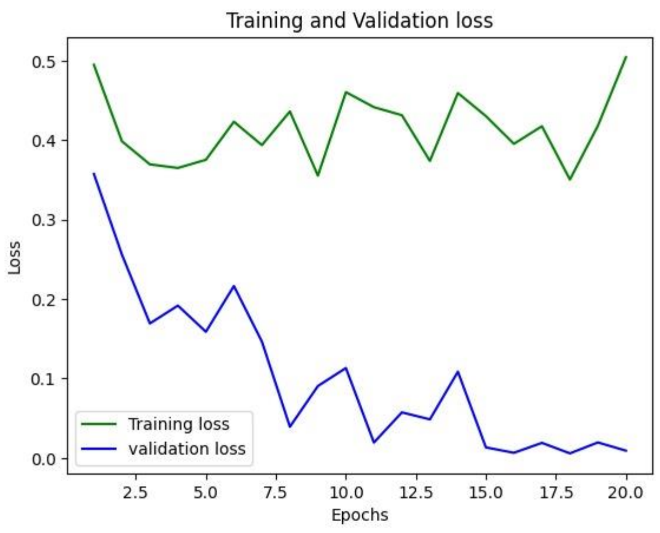
2. **Training and Validation Accuracy:**



10.2 Training and Validation accuracy for the deepfake detection model

- This graph shows how well the model is performing in terms of accuracy.
- The **training accuracy (green line)** gradually increases and reaches over 90%.
- The **validation accuracy (blue line)** starts off high and even surpasses training accuracy at certain points, which suggests the model generalizes well. But the gap between the two needs attention to prevent overfitting.

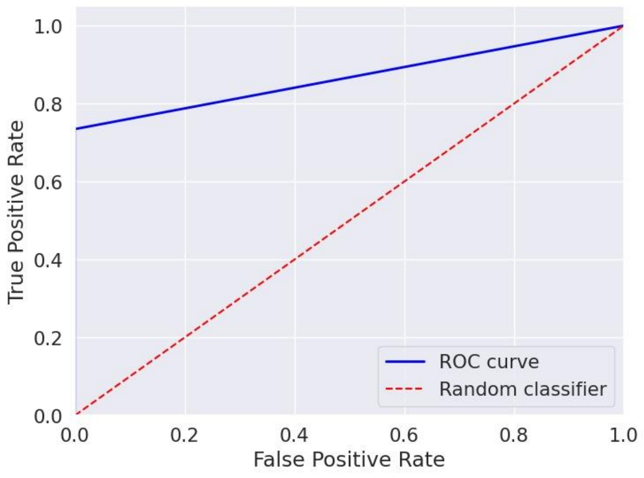
3. Training and Validation Loss:



10.3 Training and Validation loss for the deepfake detection model

- This graph tracks how the model's loss changes over 20 training epochs.
- The **training loss (blue line)** is steadily decreasing, meaning the model is learning patterns from data.
- However, the **validation loss (green line)** remains relatively high and fluctuates, which could be a sign of **overfitting**—where model does great on training data but struggles with new data.

4. ROC Curve:



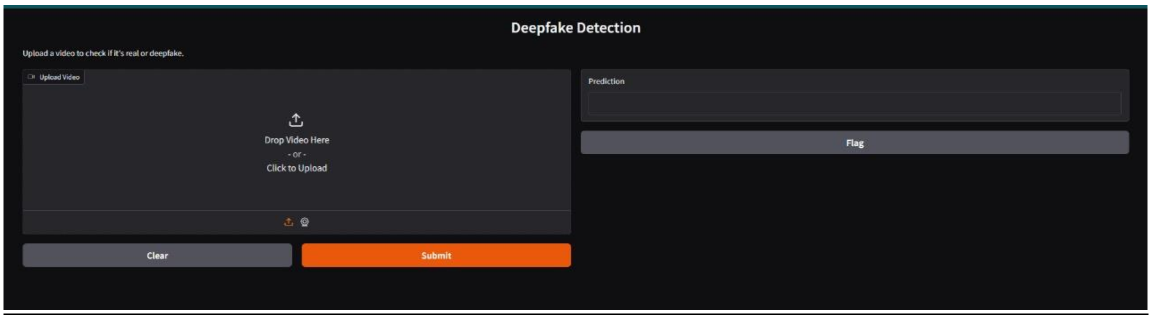
10.4 ROC curve for the deepfake detection model

- This curve helps visualize how well the model distinguishes between real and fake media.
- The **blue ROC curve** is significantly above the diagonal red line (which represents random guessing), showing that the model is performing well.

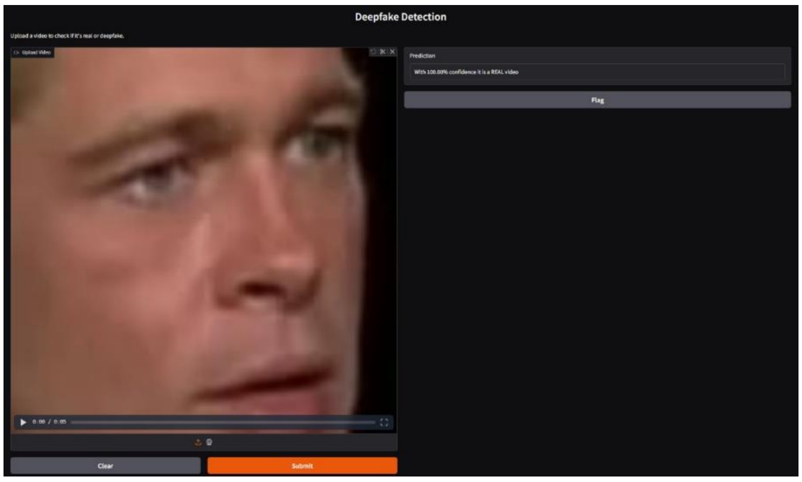
- A higher area under the curve (AUC) indicates strong predictive power, meaning the model's predictions are reliable.

5. UI RESULTS:

Here, we can see the output in 10.5, 10.6, 10.7 of the final Detection of Deepfake Videos model.

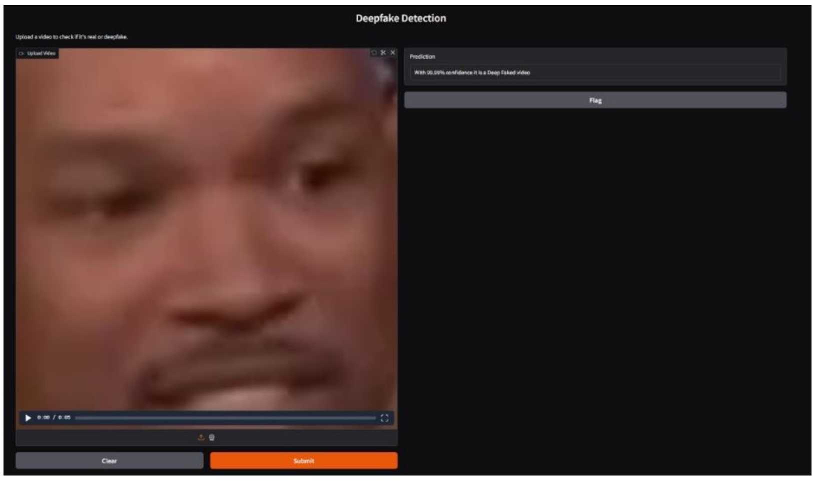


10.5 UI of the deepfake detection model



With 100.00% confidence it is a REAL video

10.6 Output 1 of the deepfake detection model



With 99.99% confidence it is a Deep Faked video

10.6 Output 2 of the deepfake detection model

11. CONLUSION & FUTURE SCOPE

In summary, the current research offers an all-encompassing method of deepfake media detection by combining Long Short-Term Memory (LSTM) networks with ResNeXt architectures. This synergy between the long-range temporal dependency capabilities of LSTM and the extensive feature extraction capabilities of ResNeXt constitutes a strong model for detecting tampered video and audio material. Through rigorous experimentation, our model has shown to outperform baseline methods with significant gains in terms of accuracy and generalization across different datasets.

The success of this hybrid approach demonstrates the potential of merging sequential and convolutional methods to meet the growing complexity of deepfake generation. In addition, our method not only helps improve deepfake detection but also sets the stage for future breakthroughs in fighting synthetic media manipulation.

Even though there has been considerable progress in developing a robust and effective deep fake detection system, there are still a number of areas for future improvement. The project can gain from further research into more sophisticated detection methods, multi-modal analysis, and cross-talk with industry professionals to remain ahead of emerging deep fake strategies. Looking ahead, a number of possible directions for future work. Extending the model to real-time detection systems may make it more useful in dynamic, high-consequence settings. Further testing on more varied and larger datasets, both audio-visual and multimodal inputs as well as detection not restricted to faces only, could make the model even more robust and capable of identifying new deepfake methods. Finally, incorporating adversarial training or semi-supervised learning methods could minimize reliance on labeled data and alleviate challenges to scalability of deepfake detection systems.

Overall, this study is an important contribution towards the creation of reliable and efficient detection and mitigation tools for deepfake content, which will play an essential role in maintaining trust in digital media and safeguarding its abuse across several contexts such as security, journalism, and social media.

12. REFERENCES

https://ieeexplore.ieee.org/document/9163247
https://arxiv.org/abs/1505.00387
https://arxiv.org/abs/1611.05431
https://arxiv.org/abs/2006.03523
https://arxiv.org/abs/1907.11959
https://arxiv.org/abs/1908.03011
https://arxiv.org/abs/1610.02357
https://arxiv.org/abs/2003.07163
https://arxiv.org/abs/1901.08971
https://www.techscience.com/iasc/v35n2/48928
https://arxiv.org/abs/1909.09586
https://arxiv.org/abs/1909.12962

13. ANNEXURE-1

Model Training

```
#0Libraries

%%capture

import glob

import torch

import torchvision

from torchvision import transforms

from torch.utils.data import DataLoader

from torch.utils.data.dataset import Dataset

import os

import numpy as np

import cv2

import random

import matplotlib.pyplot as plt

!pip install face_recognition

import face_recognition

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import roc_curve, auc


#1 Validating Video to check if we can send to train or test and coverting the video into frames

def validate_video(vid_path, train_transforms):

    transform = train_transforms

    count = 20

    video_path = vid_path

    frames = []

    a = int(100 / count)
    first_frame = np.random.randint(0, a)
    temp_video = video_path.split("/")[-1]
    for i, frame in enumerate(frame_extract(video_path)):
        frames.append(transform(frame))
        if len(frames) == count:
            break
    frames = torch.stack(frames)
    frames = frames[:count]
    return frames
#extract 'a' from video
def frame_extract(path):
    vidObj = cv2.VideoCapture(path)
    success = 1
    while success:
        success, image = vidObj.read()
        if success:
            yield image
#2 Re-size and Normalize the image
im_size = 112
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]
train_transforms = transforms.Compose(
    [
        transforms.ToPILImage(),
        transforms.Resize((im_size, im_size)),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ]
)
#3 Loading fake video dataset and counting number of videos
gbmd = {
    "file": [],
    "label": []
}
frame_count = []
video_files1 = glob.glob('/content/drive/MyDrive/deep/Celeb_fake_face_only-20241230T040101Z-001/Celeb_fake_face_only/*.mp4')
for video_file in video_files1:
    cap = cv2.VideoCapture(video_file)
    if int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) < 100:
        video_files1.remove(video_file)
        continue
    frame_count.append(int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
    gbmd["file"].append(video_file)
    gbmd["label"].append("FAKE")
print("Total no of video: ", len(frame_count))
#print(gbmd["file"][432], gbmd["label"][432])
#4 Real Video dataset
video_files2 = glob.glob('/content/drive/MyDrive/deep/Celeb_real_face_only-20241230T040206Z-001/Celeb_real_face_only/*.mp4')
for video_file in video_files2:
    cap = cv2.VideoCapture(video_file)
    if int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) < 100:
            video_files2.remove(video_file)
            continue
        frame_count.append(int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
        gbmd["file"].append(video_file)
        gbmd["label"].append("REAL")
# print("frames are ", frame_count)
print("Total no of video: ", len(frame_count))
#5 plot the image
def im_plot(tensor):
    image = tensor.cpu().numpy().transpose(1, 2, 0)
    b, g, r = cv2.split(image)
    image = cv2.merge((r, g, b))
    image = image * [0.22803, 0.22145, 0.216989] + [0.43216, 0.394666, 0.37645]
    image = image * 255.0
    plt.imshow(image.astype(int))
    plt.show()
def number_of_real_and_fake_videos(data_list):
    header_list = ["file", "label"]
    lab = pd.read_csv("/content/drive/My Drive/Gobal_metadata.csv", names=header_list)
    fake = 0
    real = 0
    if len(lab) > 0:
        for i in data_list:
            temp_video = i.split("/")[-1]
            if labels.loc[labels["file"] == temp_video].empty:
                print("No video with file name " + temp_video + " found.")
            else:
                label = lab.iloc[
                    (labels.loc[labels["file"] == temp_video].index.values[0]), 1
                ]
                if label == "FAKE":
                    fake += 1
                if label == "REAL":
                    real += 1
    # print(real)
    # print(fake)
    return real, fake
# print(gbmd)
header_list = ["file","label"]
df = pd.DataFrame(gbmd).sample(frac=1)
# df = df.sample(frac=1)
# print(df)
df.to_csv('/content/drive/My Drive/Gobal_metadata.csv', header=False, index=False)
labels = pd.read_csv('/content/drive/My Drive/Gobal_metadata.csv',names=header_list)
# print(labels)
video_files = labels["file"]
# labels = labels["label"][1:]
# print(video_files[0])
#6 Train and test split of dataset
from sklearn.model_selection import train_test_split
# random.shuffle(gbmd)
# ---------------------------------------
```

```
total_files = len(video_files)
# print(total_files)
train_ratio = 0.7
test_ratio = 0.1
# Calculate the sizes of train, test, and validation sets
train_size = int(train_ratio * total_files)
test_size = int(test_ratio * total_files)
valid_size = total_files - train_size - test_size
# Split the video files into train, test, and validation sets
train_videos = video_files[:train_size]
test_videos = video_files[train_size:train_size+test_size]
valid_videos = video_files[train_size+test_size:]
# Print the sizes of the train, test, and validation sets
print("Train set size:", len(train_videos))
print("Validation set size:", len(valid_videos))
print("Test set size:", len(test_videos))
#7 Called for Veryfing the dataset real or fake
def number_of_real_and_fake_videos(data_list):
  header_list = ["file","label"]
  lab = pd.read_csv('/content/drive/My Drive/Gobal_metadata.csv',names=header_list)
  fake = 0
  real = 0
  # print(data_list)
  if len(lab) > 0:
    for i in data_list:
      temp_video = i#.split('/')[-1]
      if labels.loc[labels["file"] == temp_video].empty:
        print("No video with file name " + temp_video + " found.")
      else:
        label = lab.iloc[(labels.loc[labels["file"] == temp_video].index.values[0]),1]
        if(label == 'FAKE'):
          fake+=1
        if(label == 'REAL'):
          real+=1
  # print(real)
  # print(fake)
  return real, fake
#8 Trasfroming(Resing and normalize) the data using #2
train_transforms = transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.Resize((im_size,im_size)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean,std)])
valid_transforms = transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.Resize((im_size,im_size)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean,std)])
test_transforms = transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.Resize((im_size,im_size)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean,std)])
#9 Preparing the dataset to send for train and test
class video_dataset(Dataset):
  def __init__(self, video_names, labels, sequence_length=60, transform=None):
        self.video_names = video_names
        self.labels = labels
        self.transform = transform
        self.count = sequence_length
  def __len__(self):
        return len(self.video_names)
  def __getitem__(self, idx):
        # print(self.video_names)
        video_path = self.video_names[idx]
        # print(video_path)
        frames = list(self.frame_extract(video_path))
        a = int(100 / self.count)
        first_frame = np.random.randint(0, a)
        temp_video = video_path#.split("/")[-1]
        # print(temp_video)
        if len(labels) == 0:
          print("No labels found.")
          return frames, None
        label = self.labels.iloc[
          (labels.loc[labels["file"] == temp_video].index.values[0]), 1
        ]
        if label == "FAKE":
            label = 0
        if label == "REAL":
            label = 1
        for i, frame in enumerate(frames):
            frames[i] = self.transform(frame)
            if len(frames) == self.count:
                break
        frames = torch.stack(frames)
        frames = frames[: self.count]
        # print("length:" , len(frames), "label",label)
        return frames, label
    def frame_extract(self, path):
        vidObj = cv2.VideoCapture(path)
        success = 1
        while success:
            success, image = vidObj.read()
            if success:
                yield image
#10 Loading the dataset for train test and validate

train_data = video_dataset(train_videos.reset_index(drop=True),labels,sequence_length =
10,transform = train_transforms)

val_data = video_dataset(valid_videos.reset_index(drop=True),labels,sequence_length =
10,transform = train_transforms)

test_data = video_dataset(test_videos.reset_index(drop=True),labels,sequence_length =
10,transform = train_transforms)

train_loader = DataLoader(train_data,batch_size = 4,shuffle = True,num_workers = 4)

valid_loader = DataLoader(val_data,batch_size = 4,shuffle = True,num_workers = 4)
```

```
test_loader = DataLoader(test_data,batch_size = 4,shuffle = True,num_workers = 4)

#11 Cleaning if there any loss of frames

for batch_idx, (data, target) in enumerate(valid_loader):

    if torch.isnan(data).any() or torch.isinf(data).any():

        print(f"Found NaN or Inf in data at batch index {batch_idx}")

#12 Model

from torch import nn

from torchvision import models

class Model(nn.Module):

  def __init__(self, num_classes,latent_dim= 2048, lstm_layers=1 , hidden_dim = 2048, bidirectional = False):

      super(Model, self).__init__()

      model = models.resnext50_32x4d(pretrained = True) #Residual Network CNN

      self.model = nn.Sequential(*list(model.children())[:-2])

      self.lstm = nn.LSTM(latent_dim,hidden_dim, lstm_layers,  bidirectional)

      self.relu = nn.LeakyReLU()

      self.dp = nn.Dropout(0.4)

      self.linear1 = nn.Linear(2048,num_classes)

      self.avgpool = nn.AdaptiveAvgPool2d(1)

  def forward(self, x):

      batch_size,seq_length, c, h, w = x.shape

      x = x.view(batch_size * seq_length, c, h, w)

      fmap = self.model(x)

      x = self.avgpool(fmap)

      x = x.view(batch_size,seq_length,2048)

      x_lstm,_ = self.lstm(x,None)
```

```
        return fmap,self.dp(self.linear1(torch.mean(x_lstm,dim = 1)))
#13 Training Epochs
import torch
from torch.autograd import Variable
import time
import os
import sys
import os
!pip install -Uqq ipdb
import ipdb
def train_epoch(epoch, num_epochs, data_loader, model, criterion, optimizer):
    model.train()
    losses = AverageMeter()
    accuracies = AverageMeter()
    t = []
    for i, (inputs, targets) in enumerate(data_loader):
        if torch.cuda.is_available():
            targets = targets.type(torch.cuda.LongTensor)
            inputs = inputs.cuda()
        _,outputs = model(inputs)
        # print("asidh", outputs)
        loss  = criterion(outputs,targets.type(torch.cuda.LongTensor))
        acc = calculate_accuracy(outputs, targets.type(torch.cuda.LongTensor))
        losses.update(loss.item(), inputs.size(0))
        accuracies.update(acc, inputs.size(0))
        optimizer.zero_grad()
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=1)  # Clip gradients
        optimizer.step()
        sys.stdout.write(
                "\r[Epoch %d/%d] [Batch %d / %d] [Loss: %f, Acc: %.2f%%]"
                % (
                    epoch,
                    num_epochs,
                    i,
                    len(data_loader),
                    losses.avg,
                    accuracies.avg))
    torch.save(model.state_dict(),'/content/checkpoint.pt')
    return losses.avg,accuracies.avg
class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()
    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0
    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
def calculate_accuracy(outputs, targets):
    batch_size = targets.size(0)
    _, pred = outputs.topk(1, 1, True)
    pred = pred.t()
    correct = pred.eq(targets.view(1, -1))
    n_correct_elems = correct.float().sum().item()
    return 100* n_correct_elems / batch_size
#14 Confusion Matrix
import seaborn as sn
#Output confusion matrix
def print_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    print('True positive = ', cm[0][0])
    print('False positive = ', cm[0][1])
    print('False negative = ', cm[1][0])
    print('True negative = ', cm[1][1])
    print('\n')
    df_cm = pd.DataFrame(cm, range(2), range(2))
    sn.set(font_scale=1.4) # for label size
    sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
    plt.ylabel('Actual label', size = 20)
    plt.xlabel('Predicted label', size = 20)
    plt.xticks(np.arange(2), ['Fake', 'Real'], size = 16)
    plt.yticks(np.arange(2), ['Fake', 'Real'], size = 16)
    plt.ylim([2, 0])
    plt.show()
    calculated_acc = (cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+ cm[1][1])
    print("Calculated Accuracy",calculated_acc*100)
def plot_loss(train_loss_avg,test_loss_avg,num_epochs):
  loss_train = train_loss_avg
  loss_val = test_loss_avg
  print(num_epochs)
  epochs = range(1,num_epochs+1)
  plt.plot(epochs, loss_train, 'g', label='Training loss')
  plt.plot(epochs, loss_val, 'b', label='validation loss')
  plt.title('Training and Validation loss')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()
def plot_accuracy(train_accuracy,test_accuracy,num_epochs):
  loss_train = train_accuracy
  loss_val = test_accuracy
  epochs = range(1,num_epochs+1)
  plt.plot(epochs, loss_train, 'g', label='Training accuracy')
  plt.plot(epochs, loss_val, 'b', label='validation accuracy')
  plt.title('Training and Validation accuracy')
  plt.xlabel('Epochs')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.show()
#15 testing set
def test(epoch,model, data_loader ,criterion):
    print('Testing')
    model.eval()
    losses = AverageMeter()
    accuracies = AverageMeter()
    pred = []
    true = []
    count = 0
    with torch.no_grad():
        for i, (inputs, targets) in enumerate(data_loader):
            if torch.cuda.is_available():
                targets = targets.cuda().type(torch.cuda.FloatTensor)
                inputs = inputs.cuda()
            # ipdb.set_trace()
            abc,outputs = model(inputs)
            # print("abc", abc, "\nOutputs", outputs)
            loss = torch.mean(criterion(outputs, targets.type(torch.cuda.LongTensor)))
            acc = calculate_accuracy(outputs,targets.type(torch.cuda.LongTensor))
            _,p = torch.max(outputs,1)
            true                                                                    +=
(targets.type(torch.cuda.LongTensor)).detach().cpu().numpy().reshape(len(targets)).tolist()
            pred += p.detach().cpu().numpy().reshape(len(p)).tolist()
            losses.update(loss.item(), inputs.size(0))
            accuracies.update(acc, inputs.size(0))
            sys.stdout.write(
                "\r[Batch %d / %d]  [Loss: %f, Acc: %.2f%%]"
                % (
                    i,
                    len(data_loader),
                    losses.avg,
                    accuracies.avg
                    )
                )
        print('\nAccuracy {}'.format(accuracies.avg))
    return true,pred,losses.avg,accuracies.avg
#16 Training the Model using the dataset
from sklearn.metrics import confusion_matrix
#learning rate
lr = 1e-5#0.001
#number of epochs
num_epochs = 20
optimizer = torch.optim.Adam(model.parameters(), lr= lr,weight_decay = 1e-7)
class_weights = torch.from_numpy(np.asarray([1,15])).type(torch.FloatTensor).cuda()
criterion = nn.CrossEntropyLoss(weight = class_weights).cuda()
# criterion = nn.CrossEntropyLoss().cuda()
# criterion = nn.BCELoss().cuda()
train_loss_avg =[]
train_accuracy = []
test_loss_avg = []
test_accuracy = []
```

```
for epoch in range(1, num_epochs + 1):
    l, acc = train_epoch(epoch,num_epochs,train_loader,model,criterion,optimizer)
    train_loss_avg.append(l)
    train_accuracy.append(acc)
    print(train_loss_avg)
    true,pred,tl,t_acc = test(epoch,model,train_loader,criterion)
    test_loss_avg.append(tl)
    print(test_loss_avg)
    test_accuracy.append(t_acc)
plot_loss(train_loss_avg,test_loss_avg,len(train_loss_avg))
plot_accuracy(train_accuracy,test_accuracy,len(train_accuracy))
print(confusion_matrix(true,pred))
print_confusion_matrix(true,pred)
Real_test_loss_avg = []
Real_test_accuracy = []
RealTesttrue,RealTestpred,RealTesttl,RealTestt_acc = test(1,model,test_loader,criterion)
Real_test_loss_avg.append(RealTesttl)
Real_test_accuracy.append(RealTestt_acc)
# plot_loss(train_loss_avg,test_loss_avg,len(train_loss_avg))
# plot_accuracy(train_accuracy,test_accuracy,len(train_accuracy))
print(confusion_matrix(RealTesttrue,RealTestpred))
print_confusion_matrix(RealTesttrue,RealTestpred)
#17 ROC Curve
np.random.seed(42)
y_true = RealTesttrue
y_scores = RealTestpred
```

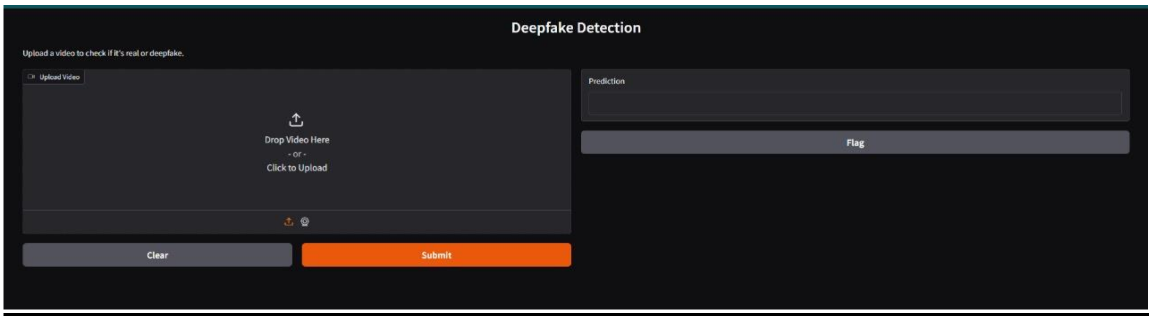
```
fpr, tpr, thresholds = roc_curve(y_true, y_scores)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve')

plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Random classifier')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc="lower right")

plt.show()

#18 Saving the model

torch.save(model.state_dict(), '/content/saved_model.pth')

path_to_model = "/content/drive/MyDrive/FF_88_acc_seqLen_20_Confusion_matrix.pth"

torch.save(model.state_dict(),
'/content/drive/MyDrive/FF_88_acc_seqLen_20_Confusion_matrix.pth')
```

Prediction and User Interface

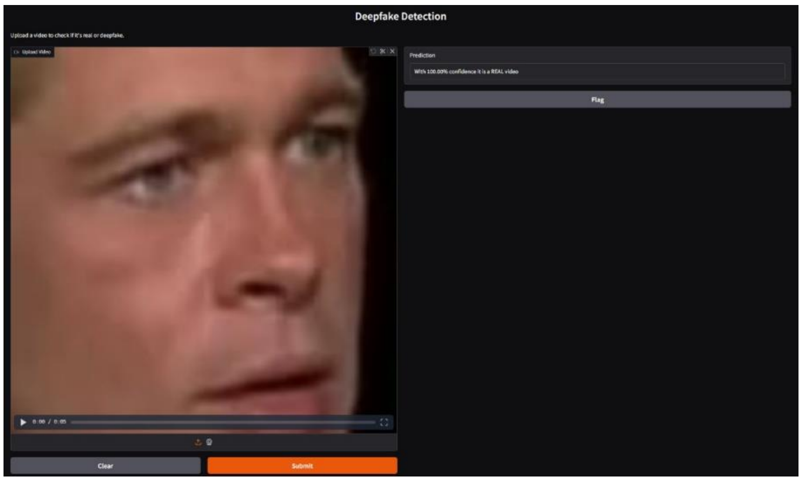
```
import gradio as gr

import torch

import torchvision

from torchvision import transforms

from torch.utils.data import Dataset

from torch import nn

from torchvision import models

import numpy as np
import cv2

# Defining model class

class Model(nn.Module):

   def __init__(self, num_classes, latent_dim=2048, lstm_layers=1, hidden_dim=2048, 
bidirectional=False):

        super(Model, self).__init__()

        model = models.resnext50_32x4d(pretrained=True)

        self.model = nn.Sequential(*list(model.children())[:-2])

        self.lstm = nn.LSTM(latent_dim, hidden_dim, lstm_layers, bidirectional)

        self.relu = nn.LeakyReLU()

        self.dp = nn.Dropout(0.4)

        self.linear1 = nn.Linear(2048, num_classes)

        self.avgpool = nn.AdaptiveAvgPool2d(1)

   def forward(self, x):

        batch_size, seq_length, c, h, w = x.shape

        x = x.view(batch_size * seq_length, c, h, w)

        fmap = self.model(x)

        x = self.avgpool(fmap)

        x = x.view(batch_size, seq_length, 2048)

        x_lstm, _ = self.lstm(x, None)

        return fmap, self.dp(self.linear1(x_lstm[:,-1,:]))

# Loading model

im_size = 112

mean = [0.485, 0.456, 0.406]

std = [0.229, 0.224, 0.225]

train_transforms = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((im_size, im_size)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])
model = Model(2)
path_to_model = "/content/drive/MyDrive/FF_88_acc_seqLen_20_Confusion_matrix.pth"
model.load_state_dict(torch.load(path_to_model, map_location=torch.device('cpu')))
model.eval()
sm = nn.Softmax()
# Prediction function
def predict(model, img):
    fmap, logits = model(img.to('cpu'))
    logits = sm(logits)
    predictionVal = logits[0][1]
    confidence = predictionVal * 100
    return confidence
# Defining the validation dataset class
class validation_dataset(Dataset):
    def __init__(self, video_names, sequence_length=20, transform=None):
        self.video_names = video_names
        self.transform = transform
        self.count = sequence_length
    def __len__(self):
        return len(self.video_names)
    def __getitem__(self, idx):
        video_path = self.video_names[idx]
        frames = []
        a = int(100/self.count)
        first_frame = np.random.randint(0, a)
        for i, frame in enumerate(self.frame_extract(video_path)):
            frames.append(self.transform(frame))
            if len(frames) == self.count:
                break
        frames = torch.stack(frames)
        frames = frames[:self.count]
        return frames.unsqueeze(0)
    def frame_extract(self, path):
        vidObj = cv2.VideoCapture(path)
        success = 1
        while success:
            success, image = vidObj.read()
            if success:
                yield image
# Defining the video processing function
def fakeOrNot(video_path):
    video_dataset = validation_dataset([video_path], sequence_length=20,
transform=train_transforms)
    predictConfidence = predict(model, video_dataset[0])
    if predictConfidence > 60:
        return f"With {predictConfidence:.2f}% confidence it is a REAL video"
    else:
        tempX = 100 - predictConfidence
        return f"With {tempX:.2f}% confidence it is a Deep Faked video"
# Create Gradio interface
def gradio_interface(video):
    return fakeOrNot(video)
iface = gr.Interface(
    fn=gradio_interface,
    inputs=gr.Video(label="Upload Video"),
    outputs=gr.Textbox(label="Prediction"),
    title="Deepfake Detection",
    description="Upload a video to check if it's real or deepfake."
)
iface.launch(debug=True)
```

13. ANNEXURE-2

Here, we can see the output in 13.1, 13.2 and 13.3 of the final Detection of Deepfake Videos model.

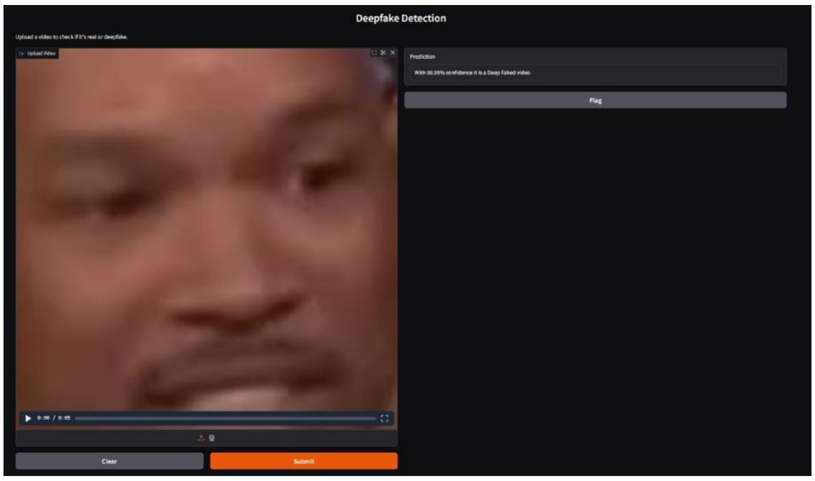


13.1 UI of the deepfake detection model



With 100.00% confidence it is a REAL video

13.2 Output 1 of the deepfake detection model



With 99.99% confidence it is a Deep Faked video

13.3 Output 2 of the deepfake detection model