CHAPTER 1

INTRODUCTION

In January 2018, a desktop application called FakeApp was launched, “Deepfake” word is the combination of “Deep learning” and “fake”, these are the AI generated videos in which a person in a video or image is replaced with someone else. The idea of swapping faces on photo is not new, we can find examples of such photos made in 19th century, back then these photos were made by hand. However, when the idea of neural networks became popular and with advancement in computational field, people began to use this technology to create such deepfake videos and images. Nowadays we can download and run such programs which can swap our face with others. Today, none of us will be surprised by apps like FaceApp, Snapchat that have an ability to swap faces with good quality and make us funny.

His app allows users to easily create and share videos with their faces swapped with each other, after this app many such apps were launched like Face Swap, DeepFace Lab. Larger companies also started to use deepfakes. The Japanese AI company DataGrid made a full body deepfake that can create a person from scratch, they intend to use these for fashion and apparel. A mobile deepfake app, Impressions, was launched in March 2020. It was the first app for the creation of celebrity deepfake videos from mobile phones. Today we can find various desktop as well as mobile apps which we can use to create a good quality deepfake.

The rise of deep fake technology represents one of the most significant challenges in the digital age, as it allows for the creation of highly realistic videos that can distort reality and manipulate perceptions. Deep fakes leverage advanced artificial intelligence techniques, particularly deep learning, to generate convincing images and sounds that can mislead audiences and undermine trust in visual media. With applications ranging from entertainment to misinformation campaigns, the implications of deep fakes are far-reaching, necessitating effective detection mechanisms to combat their spread.

The rapid advancement of artificial intelligence (AI) and machine learning (ML) has led to the emergence of sophisticated video manipulation techniques, commonly known as "deepfakes." These AI-generated videos can convincingly mimic real-world footage, allowing malicious actors to create fake content that is increasingly difficult to distinguish from reality. As a result, deepfakes pose a significant threat to authenticity, trust, and security, with far-reaching implications for individuals, organizations, and society as a whole.

Therefore, developing effective methods for detecting and mitigating deepfakes is crucial, and deep fake video detection has become a critical task that requires the development of sophisticated algorithms and techniques to identify and flag manipulated content.

Deepfake video detection is a critical area of research and development aimed at identifying and mitigating the risks posed by hyper-realistic, manipulated videos created using artificial intelligence (AI) and machine learning techniques. Deepfakes employ advanced algorithms, particularly generative adversarial networks (GANs), to alter or fabricate videos in ways that are often indistinguishable from real footage. This technology has raised significant concerns due to its potential for misinformation, identity theft, and the erosion of trust in media. The challenge of detecting deepfakes lies in their ability to seamlessly blend altered content with authentic visuals, making it difficult for both humans and traditional detection systems to identify manipulations. As a result, the field of deepfake detection is rapidly evolving, with researchers focusing on developing sophisticated tools and techniques that can detect subtle anomalies in video content, such as inconsistencies in facial expressions, lighting, or audio synchronization. Effective detection methods are crucial to preserving the integrity of digital media and ensuring the accuracy of information in an increasingly digital world.

Deepfake video detection has become an urgent field of research as the rise of artificial intelligence (AI) and machine learning has enabled the creation of hyper-realistic videos that manipulate or fabricate real footage to mislead viewers. These manipulated videos, created through techniques such as Generative Adversarial Networks (GANs), can alter facial expressions, voices, and even entire scenes, making it nearly impossible to distinguish them from actual recordings. The proliferation of deepfakes poses significant challenges to various sectors, including journalism, law enforcement, politics, and social media, where the authenticity of content is crucial.

These technologies have been exploited for malicious purposes, such as spreading misinformation, creating fake celebrity videos, and impersonating individuals, leading to serious consequences in terms of reputation, security, and trust. The complexity of deepfake detection lies in the sophistication of the AI models that produce these videos, as they continuously improve to mimic human behavior more convincingly.

As a result, deepfake detection techniques are evolving to identify subtle inconsistencies in videos, such as abnormal facial movements, inconsistent lighting, or artifacts in the background. Approaches range from traditional machine learning models to more advanced.

AI-based solutions that analyze videos at a granular level, seeking to detect tampering. As deepfake technology advances, the development of robust detection systems is essential not only for protecting public trust but also for securing digital media against malicious exploitation.

Deepfake media, such as videos and images, are the greatest form of crime created from artificial intelligence. The idea of deepfakes was first proposed on Reddit by a user called “deepfakes” who uploaded videos of his work which happened to be deepfake pornographic material of famous Hollywood actresses. This eventually resulted in Reddit banning the sub reddit deepfakes" due to a policy change concerning involuntary pornography.

Effective deep fake detection methods are required to combat the threats that deepfakes pose, and more importantly deepfake detection methods that can remain effective against newer deepfake creation methods and thus staying at least one-step ahead. Deepfakes are mostly created using Generative Adversarial Networks (GANs). GANs utilize two neural networks working against each other, where the “generator” network creates material trying to fool the second network, and the second network which is the “discriminator” attempts to spot the real or fake instances. This is the adversarial component of the network. The discriminator is essentially a classifier. The generator and discriminator learn from each other, since the generator's output is linked directly to the discriminators input, and when backpropagation is occurring, the discriminators prediction/output is used as a signal for the generator to update.

Deepfakes are typically created using Generative Adversarial Networks (GANs) or autoencoders, which train on vast amounts of data to generate highly realistic altered videos or images. GANs consist of two neural networks — a generator and a discriminator — that work together to produce fake content that is difficult to distinguish from real images or videos. The rise of deepfake technology has created significant concerns regarding its potential for misuse, especially in politics.

Detecting deepfakes is complex because the technology behind creating them is rapidly advancing. As deepfake generation tools become more sophisticated, deep learning models need to be continually updated and trained on larger, more diverse datasets to stay effective. additionally, adversarial attacks, where fake content is specifically designed to deceive detection algorithms, can make it difficult for existing models to keep up. Deepfake videos may also alter voice patterns, which can be analyzed through spectrograms and deep learning models trained to detect speech anomalies.

1.1 Defining Deep Learning

Deep learning is a subset of machine learning, which is itself a branch of artificial intelligence (AI). It involves the use of artificial neural networks to model and understand complex patterns and representations in large datasets. Deep learning is called "deep" because it involves neural networks with many layers (often called deep neural networks), where each layer is responsible for extracting increasingly abstract features of the data. This process allows deep learning models to learn from raw data directly, without the need for manual feature extraction.

In theory, deep learning algorithms are inspired by the structure and function of the human brain, where interconnected neurons process and pass information. These artificial neural networks are made up of layers of nodes (or "neurons") that each perform simple mathematical computations. The first layer of the network receives raw data (such as an image or text), and each subsequent layer progressively learns more complex features or patterns in the data. For example, in image recognition, the initial layers might learn to detect simple edges, while deeper layers might identify more complex objects like faces or animals.

The training of deep learning models involves feeding large amounts of labeled data into the network, adjusting the weights of connections between neurons through a process called back propagation, and optimizing the network to minimize the difference between its predictions and the actual outcomes. This training process is computationally intensive and often requires specialized hardware like graphics processing units (GPUs) for efficiency.

Deep learning has been highly successful in tasks such as image and speech recognition, natural language processing, and game playing, making it a cornerstone of many AI advancements in recent years. Deep learning is a specialized subset of machine learning that focuses on using multi-layered artificial neural networks to model complex patterns in data. It is fundamentally rooted in the concept of learning hierarchical representations of data. This means that deep learning algorithms automatically learn to represent data in a hierarchy of increasingly abstract levels, making it capable of handling tasks that were once difficult for traditional machine learning algorithms, such as image recognition, natural language processing, and autonomous driving.

These artificial neural networks are made up of layers of nodes (or "neurons") that each perform simple mathematical computations. This training process is computationally intensive and often requires specialized hardware like graphics processing units (GPUs) for efficiency. It is fundamentally rooted in the concept of learning hierarchical representations of data.

1.2 Deep Learning Application

- **Computer Vision**

  Deep learning, especially through Convolutional Neural Networks (CNNs), has significantly advanced the field of computer vision, enabling machines to interpret and understand visual information. Applications include:

- **Natural Language Processing (NLP)**

  Deep learning has revolutionized NLP, allowing machines to understand, interpret, and generate human language in a much more sophisticated way than earlier methods. Some key applications are:

  - **Speech Recognition:** Converting spoken language into text. Used in virtual assistants like Siri, Google Assistant, and transcription services.
  - **Language Translation:** Machine translation systems like Google Translate rely heavily on deep learning to translate text between languages with high accuracy.
  - **Sentiment Analysis:** Determining the sentiment (positive, negative, or neutral) expressed in text. This is used in social media monitoring, brand management, and customer feedback analysis.
  - **Chat bots and Virtual Assistants:** Deep learning powers more advanced chatbots and virtual assistants that can understand and generate natural conversations.
  - **Text Summarization:** Automatically generating concise summaries of longer texts, such as news articles or academic papers.

- **Healthcare and Medicine**

  Deep learning has significant applications in the healthcare industry, improving diagnostic accuracy, personalized treatment, and patient care. Key applications include:

  - **Medical Imaging:** Detecting diseases and abnormalities in medical images (e.g., identifying cancers in X-rays, CT scans, and MRIs).
  - **Predictive Analytics:** Predicting patient outcomes, such as the likelihood of disease progression or the risk of complications.
  - **Drug Discovery:** Deep learning models analyze vast datasets of molecular structures and predict potential drug candidates, speeding up the drug discovery process.

- **Personalized Medicine:** Developing treatment plans tailored to individual patients based on genetic information and health data.

- **Autonomous Vehicles**

  Deep learning is integral to the development of self-driving cars, enabling them to perceive their environment, make decisions, and navigate safely. Applications in autonomous vehicles include:

  - **Object Detection and Tracking:** Identifying pedestrians, other vehicles, traffic signs, and obstacles.
  - **Path Planning:** Determining the best route and making real-time decisions to avoid collisions or take detours.
  - **Sensor Fusion:** Combining data from multiple sensors (cameras, LiDAR, radar) to get a comprehensive understanding of the surrounding.

- **Finance and Banking**

  Deep learning is transforming the finance industry by improving decision-making, risk management, and customer services. Key applications include:

  - **Fraud Detection:** Identifying fraudulent transactions or account activities by analyzing transaction patterns and detecting anomalies.
  - **Algorithmic Trading:** Deep learning models can predict stock price movements based on historical data and market trends, enabling high-frequency trading strategies.
  - **Credit Scoring:** Analyzing financial behaviors and credit histories to assess the creditworthiness of individuals or businesses.
  - **Customer Service Automation:** Virtual assistants and chatbots are used for providing personalized financial advice and customer support.
  - **Risk Management and Financial Transactions:** In financial disputes, deepfake videos could potentially be used to create fraudulent evidence. For example, falsifying a video of a contract signing or altering footage of a transaction could cause legal complications or financial losses.

- **Potential for AI-Driven Financial Products:** Financial institutions could invest in developing and offering AI-driven tools that help individuals and businesses detect deepfakes, potentially as part of a broader cybersecurity service offering.
- **Autoencoders for Anomaly Detection:** Reconstruction of Realistic Video Content Autoencoders are neural networks trained to reconstruct input data. When a deepfake is introduced, the autoencoder struggles to reconstruct it accurately, revealing inconsistencies or artifacts.
- **Highlighting Key Features**: Attention mechanisms help models focus on the most relevant parts of the image or video that are likely to contain deepfake artifacts. This can include specific facial regions, texture patterns, or inconsistencies in reflections, shadows, or backgrounds.
- **Improved Detection Performance:** By focusing on the most important features, attention-based models can improve the accuracy and efficiency of deepfake detection.
- **Leveraging Pre-trained Models**: Deep learning models can leverage pre-trained networks (e.g., trained on large datasets for general image recognition) and fine-tune them on deepfake-specific datasets. This helps in effectively detecting deepfakes even with limited labeled data.

1.3 Key Features of Deep Learning

Deep learning has several key features that distinguish it from traditional machine learning approaches. One of the most important features is its **layered architecture**, where neural networks consist of multiple layers of neurons that automatically learn hierarchical representations of data. This allows deep learning models to process and extract complex patterns without manual feature engineering. Another critical feature is **non-linearity**, achieved through activation functions like ReLU, which enables deep learning models to capture intricate, non-linear relationships in data. The use of **backpropagation**and **gradient descent** allows for efficient training of these deep networks, adjusting the weights to minimize errors. Additionally, deep learning thrives with **large datasets**, automatically learning useful features from vast amounts of raw data.

- **Multi-Layered Neural Networks**: Deep learning models are built on multiple layers of neurons, allowing them to learn hierarchical features and capture intricate patterns in data. This depth enables the model to automatically extract relevant features from raw input data.
- **Feature Hierarchy**: Deep learning models automatically learn to detect features at multiple levels of abstraction. For example, in image recognition, the first layers might detect edges, and deeper layers combine these into more complex representations like shapes or objects.
- **Non-Linearity**: Activation functions such as ReLU (Rectified Linear Unit) or sigmoid introduce non-linearity to the network, enabling it to model complex, non-linear relationships in data. This is essential for tasks like image and speech recognition.
- **Back propagation and Gradient Descent**: Deep learning relies on back propagation to calculate the gradients of errors and adjust the weights of the model using optimization techniques like gradient descent. This process allows the model to gradually improve its performance.
- **End-to-End Learning**: Deep learning models can learn directly from raw data to output predictions, without needing manual feature extraction or preprocessing. This end- to-end learning simplifies the workflow and allows the model to learn complex relationships autonomously.

1.4 Advantages of Deep Learning

Deep learning offers several advantages that make it a powerful tool for solving complex problems across various domains. Some of the key advantages include:

- **Automatic Feature Extraction**

  Deep learning models can automatically learn relevant features from raw data, eliminating the need for manual feature engineering. This ability is especially useful in domains like image recognition and speech processing, where it would be challenging or time-consuming to manually define features.

- **High Accuracy**

  Deep learning algorithms, particularly with large and well-labeled datasets, often outperform traditional machine learning models in terms of accuracy. For tasks like image classification, speech recognition, and natural language processing, deep learning has consistently achieved state-of-the-art results.

- **Scalability with Large Datasets**

  Deep learning models excel at handling large datasets. Unlike traditional machine learning models, which may struggle to scale with more data, deep learning algorithms improve as they are exposed to larger datasets. This feature is particularly valuable as more and more industries generate vast amounts of data that need to be analyzed.

- **End-to-End Learning**

  Deep learning models are capable of end-to-end learning, where they learn directly from raw input to output. This means that deep learning can take unprocessed data (e.g., raw images or audio) and automatically transform it into the desired result, such as classifying objects or generating text. This reduces the need for manual data pre-processing.

- **Ability to Handle Unstructured Data**

  Deep learning is particularly well-suited for working with unstructured data, such as images, audio, and text.

1.5 Disadvantages of Deep Learning

While deep learning offers numerous advantages, it also has several disadvantages and challenges that need to be considered when applying it to real-world problems. These include:

- **Data Hungry**

  Deep learning models require vast amounts of labeled data to perform well. For many tasks, obtaining such large datasets can be expensive, time-consuming, or even impractical. In the absence of large datasets, deep learning models may not perform optimally or may require significant data augmentation techniques.

- **High Computational Cost**

  Deep learning models, especially deep neural networks with many layers, demand substantial computational resources. Training these models typically requires high-performance hardware like Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs), which can be expensive and energy-consuming. This can be a barrier for organizations with limited resources.

- **Long Training Times**

  Training deep learning models can take a long time, especially with large datasets and complex architectures. Even with powerful hardware, training can take days or weeks, depending on the model's complexity. This makes it less suitable for scenarios requiring quick iterations or real-time adaptation.

- **Lack of Interpretability**

  Deep learning models are often considered "black boxes," meaning it is difficult to understand or explain how they make specific decisions. This lack of interpretability poses challenges, especially in sensitive areas such as healthcare, finance, and legal systems, where understanding the reasoning behind a model's decision is crucial for trust and accountability

- **Over fitting**

  Deep learning models, particularly those with a large number of parameters, are prone to over fitting—especially when trained on small datasets.

CHAPTER 2

LITERATURE SURVEY

Deep fake technology, which uses artificial intelligence (AI) and machine learning algorithms to create highly convincing fake videos, has become a significant challenge in various fields, including cybersecurity, media, and law enforcement. The ability to detect deep fake videos is critical to prevent their malicious use, such as for spreading misinformation, defamation, or fraud. A comprehensive literature survey of deep fake video detection highlights various approaches, challenges, and advancements in this area.

Early methods of deep fake detection focused on identifying inconsistencies or artifacts in the videos, such as unnatural blinking, irregular head movements, or mismatches in facial features. These methods often relied on **image processing techniques** and **handcrafted features.** Techniques such as **Optical Flow Analysis** and **Wavelet Transform** were used to analyze motion inconsistencies. These methods, however, were not robust enough to handle the increasingly sophisticated nature of deep fakes...

- **Survey Paper 1**
  - **Title:** A Comprehensive Literature Survey on Deep Fake Video Detection Techniques and Challenges
  - **Published Year**: **2024**
  - **Description**

    Provides a detailed review of the rapidly evolving field of deep fake video detection. As deep fake technology, powered by machine learning and artificial intelligence, becomes more sophisticated, the need for reliable detection methods has grown significantly.
  - **Advantage**

    The use of eye blinking patterns in the "Deep Vision" method provides a unique approach to detecting deep fakes, incorporating behavioral indicators into the analysis.
  - **Disadvantage**

    CNN-based methods may struggle with over fitting, leading to reduced effectiveness in real- world scenarios where conditions vary widely.

- **Survey Paper 2**
  - **Title:** provides a detailed review of the rapidly evolving field of deep fake video detection.
  - **Published Year: 2023**
  - **Description**

    Understand and leverage advancements in machine learning to streamline the process of video editing, particularly for deep fake generation.
  - **Advantage**

    Advances in mobile technology and AI make it easier for anyone to create and disseminate videos, enhancing creative expression.
  - **Disadvantage**

    The capability to forge identities and create misleading content poses ethical dilemmas regarding consent and privacy**.**

- **Survey Paper 3**
  - **Title:** Deep Fake Detection Techniques, Challenges, and Solutions for Identifying Fake Videos
  - **Published Year: 2022**
  - **Description**

    Detect resolution inconsistencies and artifacts caused by the face warping process usedin deep fake generation.
  - **Advantage**

    The proposed deep learning method improves the ability to distinguish between real and deepfake videos, addressing the growing problem of misinformation.
  - **Disadvantage**

    The effectiveness of the detection method is heavily reliant on the quality and diversity of the training dataset. Limited or biased data could lead to poor generalization.

- **Survey Paper 4**
  - **Title:** Detection and Mitigation of Deep Fake Videos Using Machine Learning Techniques
  - **Published Year: 2023**
  - **Description**

    Classify and distinguish between different types of facial manipulations—expression transfers and identity swaps—to improve detection specificity...
  - **Advantage**

    Transfer learning allows the model to utilize pre-trained networks, enhancing its ability to recognize complex patterns in deep fakes while reducing training time and resource requirements.
  - **Disadvantage**

    As deep fake creation technology continues to improve, detection methods must constantly adapt, which can be resource-intensive and challenging.

- **Survey Paper 5**
  - **Title:** XceptionNet-Based Deep Fake Detection
  - **Published Year: 2023**
  - **Description**

    Leverage complementary information from different feature domains to improve detection performance against various types of deep fake generation techniques.
  - **Advantage**

    By integrating features from both the spatial and frequency domains, the method captures subtle artifacts introduced during the deepfake creation process, leading to improved detection performance.
  - **Disadvantage**

    Combining features from various domains increases the computational load, potentially leading to longer processing times and higher resource consumption.

CHAPTER 3

PROBLEM STATEMENT

With the rise of sophisticated deep fake technology, there is a growing concern about its potential misuse, ranging from spreading misinformation to creating fraudulent content. In response to this threat, there is a critical need for robust deep fake detection systems capable of accurately discerning manipulated videos from authentic ones. This project aims to address this challenge by employing a combination of ResNext and LSTM architectures for deep fake detection.

- ## Problem Statement Summary

  The proliferation of deepfake technology, which involves using artificial intelligence (AI) to manipulate videos to create highly realistic but fabricated content, poses significant challenges to media authenticity, privacy, and security. Deepfake videos are being increasingly used in malicious ways, such as spreading misinformation, cyberbullying, and political manipulation, making the need for reliable detection systems more urgent.

- ## Objective

  The goal is to develop a robust system for detecting deepfake videos using Convolutional Neural Networks (CNNs). These networks are particularly well-suited for image and video analysis due to their ability to automatically learn hierarchical features from raw data. By leveraging CNNs, the system should be able to identify subtle inconsistencies in deepfake videos that are often invisible to the human eye.

- ## Problem Context

  - **Growing sophistication of deepfakes**: As AI techniques advance, deepfake generation tools have become more accessible and produce more convincing content, making detection increasingly challenging.
  - **Traditional detection methods**: Many existing deepfake detection methods rely on manual feature engineering or require large datasets that are often difficult to obtain and may not generalize well to new types of deepfakes.
  - **CNNs for video analysis**: Convolutional neural networks, particularly 3D-CNNs or CNNs enhanced by recurrent layers, have shown promise in learning spatial and temporal features of videos, which can be used to distinguish real from manipulated footage.

- ## Challenges
  - **Real-time detection**: Deepfake detection systems must operate in real-time to be effective in live streaming scenarios and media sharing platforms.
  - **Data variability**: Deepfake videos can vary greatly in quality and manipulation techniques, requiring the detection system to adapt to a broad range of deepfake generation methods.
  - **Imbalance in datasets**: Training deepfake detection models with balanced data can be difficult, as the availability of genuine and manipulated videos may not be equal.
- ## Scope

The system will utilize CNNs to automatically learn features from both spatial (frame-wise) and temporal (motion-based) aspects of video sequences to identify artifacts introduced during the manipulation process. These features could include discrepancies in lighting, facial movements, audio-visual inconsistencies, or unnatural motion patterns, which are often present in deepfake videos but not in real ones.

The proposed solution will be evaluated on standard deepfake datasets, comparing the detection accuracy of CNN-based methods against traditional machine learning approaches and other neural network architectures like recurrent neural networks (RNNs) or transformers. The final goal is to enhance the detection capability to an extent where it can be used effectively in real-world scenarios, such as social media platforms, news organizations, or security systems.

CHAPTER 4

GOALS OF PROJECT

- **Develop a CNN-based Detection Model**
  - Create a deep learning model using Convolutional Neural Networks (CNNs) to detect deepfake videos.
  - Train the model to recognize subtle inconsistencies or artifacts in deepfake videos that differentiate them from authentic content, using both spatial and temporal information.
- **Improve Detection Accuracy**
  - Achieve high accuracy in distinguishing between real and manipulated videos by fine-tuning the CNN architecture.
  - Optimize the model to minimize false positives and false negatives, ensuring reliable and precise classification.
- **Handle Diverse Deepfake Generation Methods**
  - Ensure the model can detect deepfakes generated using various techniques, including facial manipulation, lip-syncing, and deep learning-based video generation methods.
  - Design the model to generalize well across different types of deepfake videos, including those generated by recent technologies and tools.
- **Real-Time Detection Capabilities**
  - Aim to develop a system capable of detecting deepfake videos in real time, making it suitable for practical applications such as live video streaming, social media monitoring, or media verification.
- **Leverage Spatial and Temporal Features**
  - Exploit both spatial (frame-wise) and temporal (motion-related) features in video data to enhance detection performance.
  - Use techniques such as 3D-CNNs or temporal convolution layers to better analyze the video’s movement, facial expressions, and other dynamic features.
- **Build a Robust Dataset for Training**
  - Compile a large and diverse dataset of real and deepfake videos, ensuring that it includes various sources, contexts, and manipulation techniques to enhance model training.
  - Address the imbalance between genuine and deepfake content by employing data augmentation or synthetic data generation techniques.

- **Evaluate Model Performance**
  - Use performance metrics such as accuracy, precision, recall, F1-score, and AUC (Area Under Curve) to evaluate the effectiveness of the detection model.
  - Compare the performance of the CNN model against traditional detection methods and alternative deep learning architectures like RNNs or hybrid models.
- **Robustness Against High-Quality Deep fakes**
  - Ensure the detection model can accurately identify high-quality deepfakes that are specifically designed to evade traditional detection methods.
  - Improve the model's robustness to deepfake videos with minimal visible artifacts, making it effective even against state-of-the-art manipulation techniques.
- **Develop a Scalable System for Deployment**
  - Build the model to be scalable and easily deployable across different platforms and environments, such as social media, video sharing sites, or security systems.
  - Focus on optimizing the model for efficient processing and low-latency detection in real-world applications.
- **Provide Interpretability and Explain ability**
  - Integrate mechanisms for interpreting and explaining how the CNN model makes its predictions, helping end-users or security professionals understand why a video is classified as real or fake.
  - Enhance the model’s transparency to foster trust in its ability to detect deepfakes.
- **Contribute to the Fight Against Misinformation**
  - Contribute to the broader effort to combat misinformation, fake news, and the spread of malicious deepfake content by providing a reliable detection tool.
  - Enable journalists, media outlets, and social platforms to verify the authenticity of videos more effectively and prevent the misuse of deepfake technology.

CHAPTER 5

PROPOSED SYSTEM

- **Data Collection and Pre-processing**
  - **Dataset Acquisition:** Gather a diverse dataset containing real and deepfake videos from various sources, including popular deepfake datasets like the Deep Fake Detection Challenge (DFDC) or Face Forensics++.
  - **Video Preprocessing:** Extract frames from the videos to create datasets suitable for CNN input. Apply image preprocessing techniques such as normalization, resizing, and augmentation to improve model generalization and prevent overfitting.
- **CNN Architecture Design**
  - **Model Selection:** Use Convolutional Neural Networks (CNNs) to process video frames and learn relevant features. The architecture may include layers such as:
  - **Convolutional Layers:** To extract spatial features from individual frames.
  - **Pooling Layers:** For reducing dimensionality and enhancing important features.
  - **Fully Connected Layers:** To make final classifications (real or fake).
  - **3D-CNNs (Optional):** To capture both spatial and temporal features, analyzing the video over time by considering multiple consecutive frames.
  - **Model Variation:** Consider different variations of CNNs, such as ResNet or Inception Net, to enhance performance and capture intricate details.
- **Temporal Feature Integration**
  - **Time-Series Analysis:** Incorporate temporal information by using methods like 3D convolutions or integrating Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) units after CNN layers to capture the motion and changes across video frames.
  - **Video Segmentation:** The system will segment videos into shorter clips or frames to focus on key segments that may reveal deepfake artifacts (e.g., facial anomalies, motion irregularities).
- **Deepfake Feature Detection**
  - **Spatial Features:** The CNN model will learn to identify inconsistencies in individual frames, such as unnatural textures, lighting, or facial expressions.
  - **Temporal Features:** The model will also detect temporal discrepancies like unnatural

motion, lip-sync errors, or mismatched audio-visual cues across consecutive frames.

- **Deepfake Artifacts:** Focus on artifacts introduced during the manipulation process, such as irregularities in skin texture, eye blinking patterns, facial expressions, or motion inconsistencies.

- **Model Training**
  - **Supervised Learning:** Train the CNN model using labeled datasets of real and deepfake videos, applying techniques like cross-validation to ensure robustness and reduce overfitting.
  - **Transfer Learning:** Utilize pre-trained CNN models on large image datasets (e.g., VGG16, ResNet, or Inception Net) and fine-tune them for deepfake detection to leverage existing features and accelerate training.
  - **Loss Function:** Use binary cross-entropy or other suitable loss functions for classification tasks to optimize the model's performance during training.
- **Evaluation and Performance Metrics**
  - **Metrics:** Evaluate the model using performance metrics such as accuracy, precision, recall, F1-score, and Area Under Curve (AUC) to assess its ability to detect deepfakes.
  - **Generalization:** Ensure the model generalizes well across unseen deepfake techniques and different video types (e.g., different backgrounds, lighting, and environments).
  - **Robustness:** Test the model's robustness by evaluating its ability to detect high-quality deepfakes that aim to mimic realistic human behavior's.
- **Real-Time Detection**
  - **System Optimization:** Optimize the CNN model to allow for real-time video analysis, ensuring it can process and classify video frames quickly for live-streaming applications or media verification systems.
  - **Edge Deployment:** Design the system for deployment on edge devices or cloud platforms, balancing between detection speed and accuracy for scalable deployment.
- **Deep fake Detection System Workflow**
  - **Input Video:** The system will accept a video input (either live stream or uploaded video file).
  - **Frame Extraction:** The system will extract frames or video segments for analysis by the CNN model.
  - **Feature Extraction:** The CNN model will process the frames, analyzing both spatial

and temporal features to detect anomalies.

- **Classification:** After feature extraction, the model will classify the video as either “real” or “fake.”
- **Output:** Display the results (real/fake) with a confidence score, potentially marking which parts of the video raised suspicion.
- **Real Videos**: Authentic videos from various sources (e.g., news clips, public appearances).
- **Fake Videos**: Synthetic videos generated using GANs, face-swapping tools, or other deep fake technologies.
- **Normalization**: Resizing and normalizing the frames to ensure uniformity for training deep learning models.
- **Noise Reduction**: Cleaning the video data to remove any irrelevant noise and artifacts.

- **Explain ability and Visualization**
  - **Feature Visualization:** Implement tools for visualizing the features learned by the CNN model, such as Grad-CAM or saliency maps, to explain why the model flagged certain parts of the video as fake.
  - **Confidence Scores:** Provide users with confidence scores indicating the certainty of the deepfake classification, enhancing trust in the system.
- **Model Deployment and Integration**
  - **Platform Integration**: The detection system should be adaptable to different platforms (e.g., social media, news agencies, video streaming services) for real-time video validation.
  - **Scalability**: Ensure the system can handle large-scale video uploads or streaming while maintaining performance in terms of accuracy and speed.
- **Continuous Improvement**
  - **Model Updating:** Implement a mechanism for continuously updating the detection model with new data, especially as deepfake techniques evolve.
  - **User Feedback Loop:** Allow users to provide feedback on false positives/negatives, enabling the model to learn and adapt over time.

5.1 System Architecture

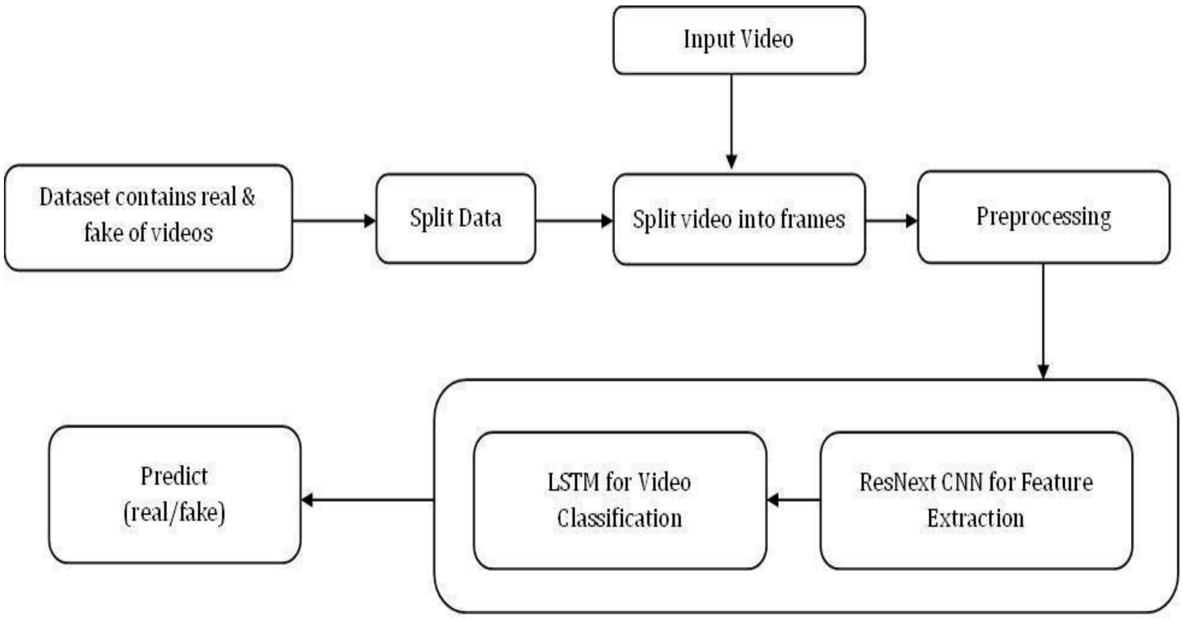

Input Video
Dataset contains real & fake of videos
Split Data
Split video into frames
Preprocessing
ResNext CNN for Feature Extraction
LSTM for Video Classification
Predict (real/fake)

Fig 5.1.1 System Architecture

The above figure 5.1.1 shows the System Architecture. In the suggested framework, the utilization of Res-Next Convolutional Neural Network is employed for the extraction of frame-level features from videos. Subsequently, these features are inputted into an RNN (Recurrent Neural Network) based on LSTM (Long Short-Term Memory) architecture.

This RNN is designed to discern the temporal relationships among individual frames, facilitating the classification of videos into the categories of either authentic or manipulated.

- **Gathering Data:** We have combined data from several datasets, including the Face Forensics ++, Celeb-DF, Deepfake Detection Challenge to create a new dataset.
- **Splitting of data:** To ensure that our model is not biased, we trained it on an equal number of real and fraudulent videos. We took an existing dataset, preprocessed it, and produced a new processes dataset in the development phase that only contained the face-cropped videos.
- **Pre-processing:** Dataset preprocessing comprises the breaking the video into frames. Identifying faces within each frame and precisely cropping the frame to focus solely on the detected face.

To maintain uniformity in the dataset, the mean number of frames per video is calculated. Subsequently, a new dataset is created, containing face-cropped frames with this mean frame count. Frames lacking detected faces are excluded during the preprocessing stage.

- **Model:** The architecture of the model consists of a single LSTM layer after resnext50_32x4d.The videos are divided into train and test sets by the Data Loader after they have been preprocessed and face cropped. During training and testing, the frames from the processed videos are divided into smaller batches. These batches facilitate efficient model training and evaluation.

- **Prediction:** To make predictions, a fresh video is fed into the learned model. To incorporate the trained model's format, a fresh video is additionally preprocessed. Upon dividing the video into frames and cropping them to focus on faces, the face-cropped frames are directly fed to the trained model for identification. There is no local storage involved in this streamlined approach.
- **Output:** The results, from the model will show if the video is real or a deepfake along with the level of certainty, in the model.

5.2 UML Diagrams

A UML diagram shows the unified visual presentation of the UML (Unified Modelling Language) system intending to let developers or business owners understand, analyze, and undertake the structure and behaviors of their system. So far, the UML diagram has become one of the most common business process modelling tools, which is also highly significant to the development of object-oriented software.

UML diagrams have many benefits for both software developers and businesspeople, and the most key advantages are:

- **Problem-Solving -** Enterprises can improve their product quality and reduce cost especially for complex systems in large scale. Some other real-life problems including physical distribution or security can be solved.
- **Improve Productivity -** By using the UML diagram, everyone in the team is on the same page and lots of time is saved down the line.
- **Easy to Understand -** Since different roles are interested in different aspects of the system, the UML diagram offers non-professional developers, for example, stakeholders, designers, or business researchers, a clear and expressive presentation of requirements, functions and processes of their system.

5.3 Use Case Diagram

A use case diagram is the primary form of system/software requirements for a new software program underdeveloped. Use cases specify the expected behavior, and not the exact method of making it happen (as shown fig:5.3.1). Use cases once specified can be denoted both textual and visual representation. A key concept of use case modelling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behavior in the user's terms by specifying all externally visible system behavior.

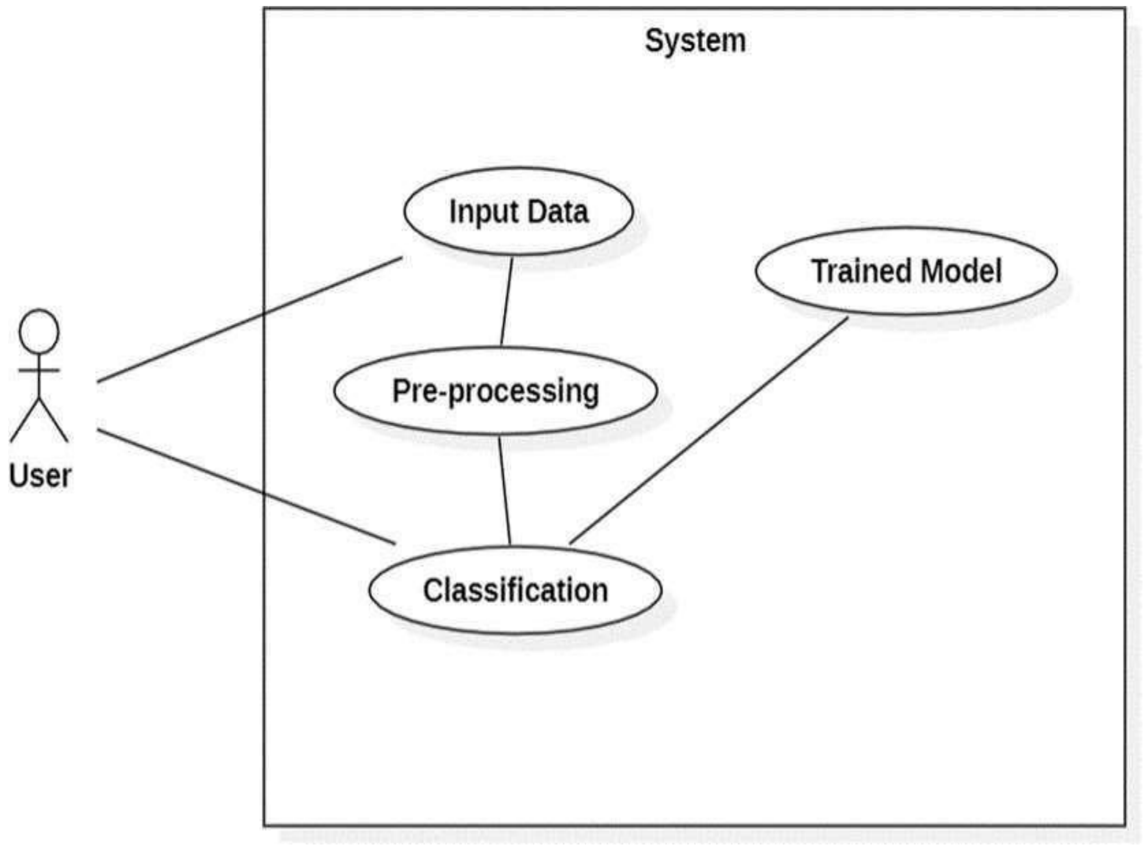


Fig 5.3.1 Use Case Diagram

5.4 Class Diagram

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction. UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as –

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

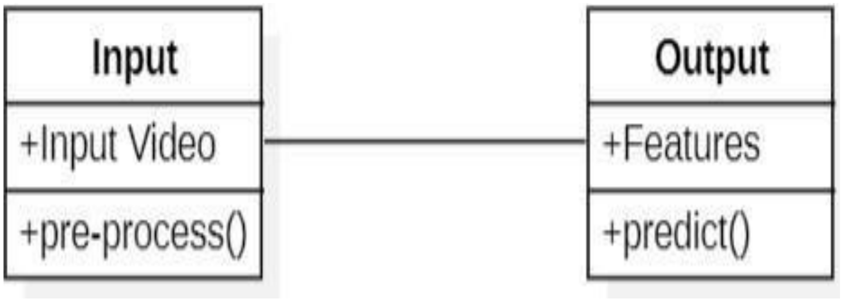


Fig 5.4.1 Class Diagram

5.5 Activity Diagram

The above figure 5.5.1 shows the Activity Diagram. A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioral diagram that illustrates the flow of activities through a system. UML activity diagrams can also be used to depict a flow of events in a business process. They can be used to examine business processes in order to identify its flow and requirements.

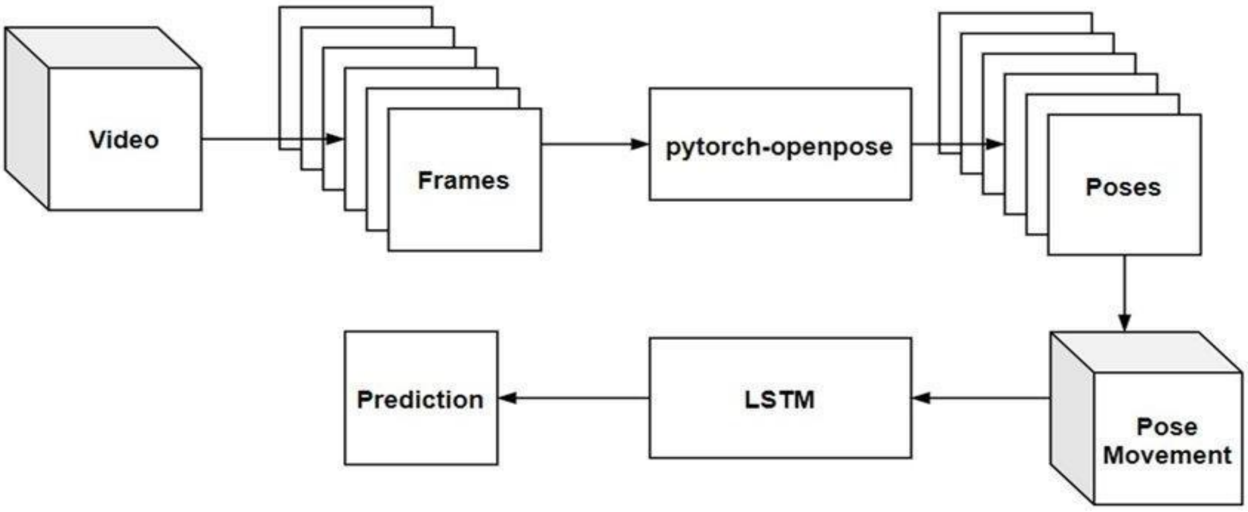


Fig 5.5.1 Activity Diagram

CHAPTER 6

SYSTEM REQUIREMENTS

6.1 Software Requirements

Software Resources Required Platform

- **Operating System:** Windows 7+
- **Programming Language:** Python 3.0
- **Framework:** PyTorch 1.4, Django 3.0
- **Libraries:** OpenCV, Face-recognition

6.2 Hardware Requirements

- **CPU:** A modern CPU with multi-core (Intel Core i7 or AMD Ryzen 7 series)
- **Memory (RAM):** 16GB RAM or above
- **Storage:** A high-performance GPU with CUDA-enabled capabilities such as NVIDIA GeForce GTX or RTX
- **Storage:** A minimum of 100 GB

CHAPTER 7
TECHNOLOGIES

7.1Python

Python is a high level, interpreted and general-purpose dynamic programming language that focuses on code readability. It has fewer steps when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

Python is an interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where asother languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a must for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain.

Advantages:

- Presence of third-party modules
- Extensive support libraries (NumPy for numerical calculations, Pandas for dataanalytics etc...)
- Open source and community development
- Easy to learn
- User-friendly data structures
- High-level language
- Dynamically typed language (No need to mention data type based on value allocated, it takes data type)
- Object-oriented language
- Portable and Interactive
- Portable across Operating systems

7.2 Python Libraries and Frameworks

- face-recognition-models==0.3.0
- filelock==3.0.12
- Flask==2.0.1
- imageio==2.9.0
- itsdangerous==2.0.1
- Jinja2==3.0.1
- joblib==1.0.1
- kiwisolver==1.3.2
- MarkupSafe==2.0.1
- matplotlib==3.4.3
- networkx==2.6.3
- nltk==3.6.6
- numpy==1.21.0
- opencv-python==4.5.3.56
- openpyxl==3.0.7
- pandas==1.3.0
- Pillow==9.0.0
- platformdirs==2.2.0
- pyparsing==2.4.7
- python-dateutil==2.8.2
- pytz==2021.1
- PyWavelets==1.1.1
- regex>=2021.8.3
- scikit-image==0.18.3
- scikit-learn==0.24.2
- scipy==1.7.0
- six==1.16.0
- threadpoolctl==2.2.0

- tifffile==2021.8.30
- torch==1.9.0
- torchvision==0.10.0
- tqdm==4.61.2
- typing-extensions==3.10.0.2
- virtualenv==20.7.2
- Werkzeug==2.0.1

- **PyTorch**

  PyTorch is an open-source deep learning framework developed primarily by Facebook's AI Research lab (FAIR). It provides a flexible and dynamic computational graph, allowing for intuitive model development and experimentation. With its Pythonic interface, PyTorch simplifies the process of building complex neural networks. Its automatic differentiation feature enables efficient gradient computation, crucial for training deep learning models. PyTorch supports both CPU and GPU computation, leveraging CUDA for accelerated training on NVIDIA GPUs. Its extensive ecosystem includes torch vision for computer vision tasks and torch text for natural language processing, making it a popular choice for researchers and practitioners alike.

- **Django**

  Django is a high-level Python web framework known for its simplicity and versatility in building web applications. It follows the "batteries-included" philosophy, providing a wide range of features out-of-the-box, such as authentication, URL routing, and database management. With Django's built-in ORM (Object-Relational Mapper), developers can interact with databases using Python code rather than SQL queries directly. Its robust security features help developers build secure web applications effortlessly. Django's scalability and extensive documentation make it a popular choice for web development projects of any size.

- **NumPy**

  NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- **OpenCV**

  OpenCV (Open-Source Computer Vision Library) is a powerful open-source computer vision and machine learning software library written in C++ and optimized for real-time applications.

It provides a wide range of functionalities for image and video analysis, including object detection, tracking, and manipulation. OpenCV's extensive collection of algorithms makes it a popular choice for academic research, commercial applications, and hobbyist projects.

- **Face Recognition**

  Python's face recognition library is a popular tool for detecting and recognizing faces in images and videos. Built on top of dlib, it provides pre-trained models for face detection, recognition, and clustering. With a simple interface and robust performance, it's widely used for various applications, including security systems, biometrics, and augmented reality.

- **Pandas**

  Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

- **Torch**

  Python's Torch library, based on Lua's Torch framework, offers powerful capabilities for deep learning research and development. It's known for its flexibility, enabling easy experimentation with complex neural network architectures while providing efficient GPU acceleration for faster training.

- **Torch vision**

  Torch vision is a Python package that provides datasets, models, and transforms for computer vision tasks within PyTorch. It simplifies the process of working with image data by offering standard datasets, pre-trained models, and image transformations for training neural networks.

- **Matplotlib**

  Matplotlib is a widely-used Python library for creating static, interactive, and publication-quality visualizations with just a few lines of code. Its versatile functionality and easy integration make it a cornerstone tool for data visualization in various fields, including scientific research, data analysis, and machine learning.

CHAPTER 8 IMPLEMENTATION

8.1 Dataset Details

The below figure 8.1.1 shows the Dataset preparation. For making the model efficient for real time prediction. We have gathered the data from different available data-sets like FaceForensic++(FF) [1], Deepfake detection challenge (DFDC)[2], and Celeb-DF[3]. Further we have mixed the dataset thecollected datasets and created our own new dataset, to accurate and real time detection on different kind of videos. To avoid the training bias of the model we have considered 50% Real and 50% fake videos. Deep fake detection challenge (DFDC) dataset [3] consists of certain audio alerted video, as audio deepfake are out of scope for this paper. We preprocessed the DFDCdataset and removed the audio altered videos from the dataset by running a python script. After preprocessing of the DFDC dataset, we have taken 1500 Real and 1500 Fake videos from the DFDC dataset. 1000 Real and 1000 Fake videos from the FaceForensic++(FF) [1] dataset and 500 Real and 500 Fake videos from the Celeb DF [3] dataset. Which makesource total dataset consisting 3000 Real, 3000 fake videos and 6000 videos in total.

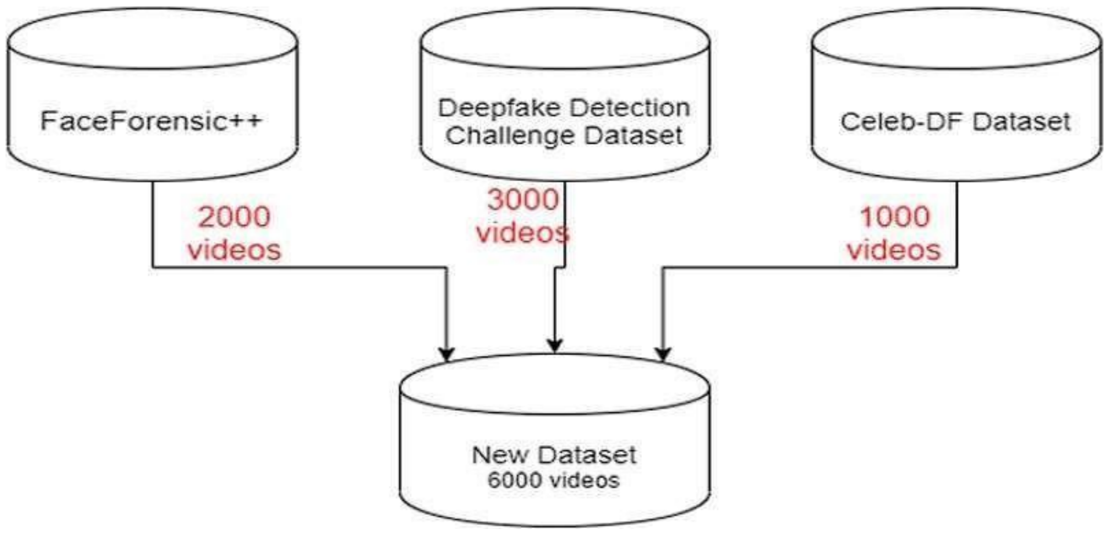


Figure 8.1.1 Dataset preparation

8.2 Preprocessing Details

- Using glob, we imported all the videos in the directory in a python list. cv2.
- Video Capture is used to read the videos and get the mean number of frames in each video.
- To maintain uniformity, based on mean a value 150 is selected as idea value for creating the new dataset.
- The video is split into frames and the frames are cropped on face location.
- The face cropped frames are again written to new video using Video Writer.
- The new video is written at 30 frames per second and with the resolution of 112x 112 pixels in the mp4 format.
- Instead of selecting the random videos, to make the proper use of LSTM for temporal sequence analysis the first 150 frames are written to the new video.

8.3 Model Details

The model consists of following layers:

- ResNext CNN
- Sequential Layer
- LSTM Layer
- ReLU
- Dropout Layer
- Adaptive Average Pooling Layer

8.3.1 ResNext CNN

The pre-trained model of Residual Convolution Neural Network is used. The model’s name is resnext50_32x4d()[22]. This model consists of 50 layers and 32 x 4 dimensions. Figure8.3.1 shows the detailed implementation of model

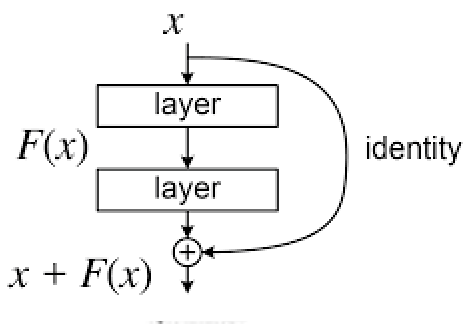
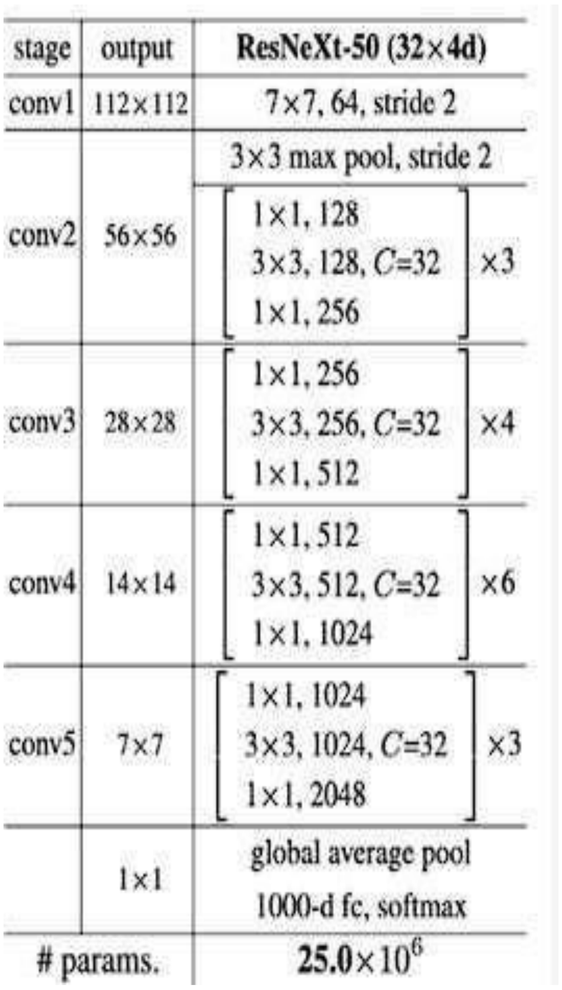


Fig 8.3.1.1: ResNext Working

8.3.2 Sequential Layer

Sequential is a container of Modules that can be stacked together and run at the same time. Sequential layer is used to store feature vector returned by the ResNext model (as shown in Fig:8.3.2) in a ordered way. So that it can be passed to the LSTM sequentially.



stage	output	ResNeXt-50 (32×4d)
conv1	112×112	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.0}\times10^6$

Fig 8.3.2.1 ResNext Architecture

8.3.3 LSTM Layer:

LSTM is used for sequence processing and spot the temporal change between the frames.2048-dimensional feature vectors is fitted as the input to the LSTM. We are using 1 LSTM layer with 2048 latent dimensions and 2048 hidden layers along with 0.4 chance of dropout, which is capable to do achieve our objective. The below figure 8.3.3.1 & 8.3.3.2 shows LSTM is used to process the frames in a sequential manner so that the temporal analysis of the video can be made, by comparing the frame at ‘t’ second with the frame of ‘t-n’ seconds. Where n can be any number of frames before t.

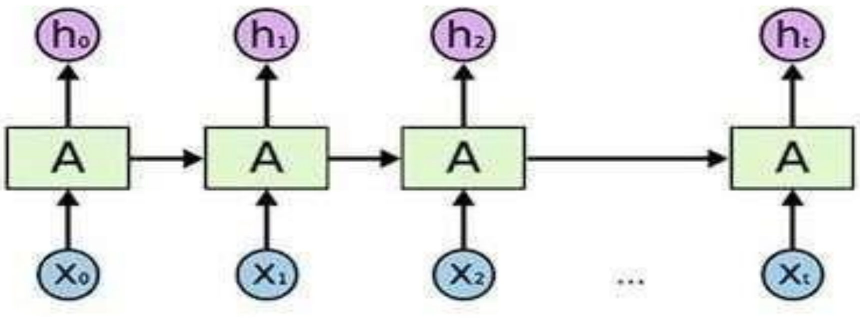


Fig 8.3.3.1 Overview of LSTM Architecture

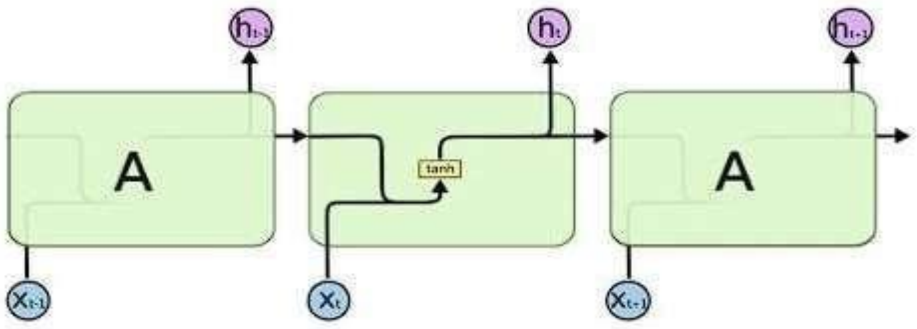


Fig 8.3.3.2 Internal LSTM Architecture

8.3.4 ReLU

A Rectified Linear Unit is activation function that has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. The operation of ReLU is closer to theway our biological neurons work. ReLU is non-linear and has the advantage of not having any backpropagationerrors unlike the sigmoid function, also for larger Neural Networks, the speed of building models based off on ReLU is very fast.

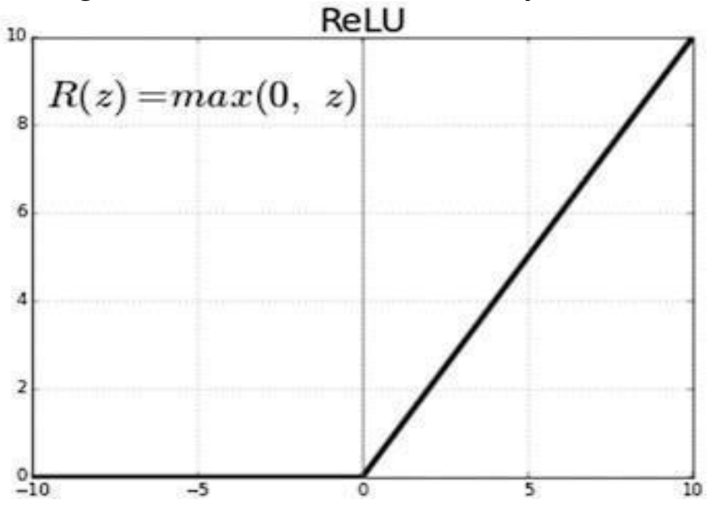


Fig 8.3.4.1 Relu Activation Function

8.3.5 Dropout Layer

Dropoutlayer with the value of 0.4 is used to avoid over fitting in the model and it can help a model generalize by randomly setting the output for a given neuron to 0. In setting the output to 0, the cost function becomes more sensitive to neighboring neurons changing the way the weights will be updated during the process of backpropagation. This process we can see in figure 8.3.5.

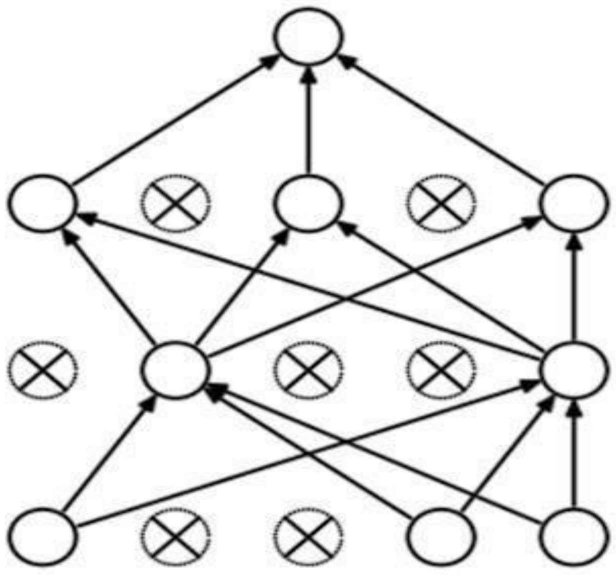


Fig 8.3.5.1 Dropout Layer Overview

8.4 Model Training Details

- **Train Test Split:** The dataset is split into train and test dataset with a ratio of 70%train videos (4,200) and 30% (1,800) test videos. The train and test split are a balanced spliti.e 50% of the real and 50% of fake videos in each split.

- **Data Loader:** It is used to load the videos and their labels with a batch size of 4. **Training:** The training is done for 20 epochs with a learning rate of 1e-5 (0.00001), weight decay of 1e-3 (0.001) using the Adam optimizer.

- **Adam Optimizer:** To enable the adaptive learning rate Adam optimizer with the model parameters is used.

- **Cross Entropy:** To calculate the loss function Cross Entropy approach is used because we are training a classification problem.

- **SoftMax Layer**

  A SoftMax function is a type of squashing function. Squashing functions limit the output of the function into the range 0 to 1. This allows the output to be interpreted directly as a probability. Similarly, Soft M functions are multi-class sigmoid, meaning they are used in determining probability of multiple classes at once. Since the outputs of a SoftMax function can be interpreted as a probability (i.e. They must sum to 1), a SoftMax layer is typically the final layer used in neural network functions. It is important to note that a SoftMax layer must have the same number of nodes as the output later. In our case SoftMax layer has two output nodes i.e. REAL or FAKE, also SoftMax layer provides us the confidence(probability) of prediction.

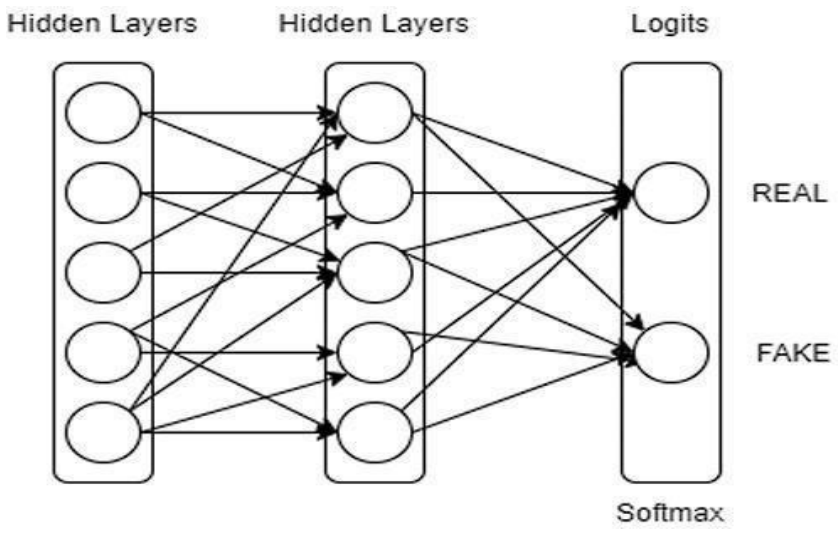


Fig 8.4.1 Software Layer

- **Confusion Matrix:** A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made. Confusion matrix is used to evaluate our model and calculate the accuracy.
- **Export Model:** After the model is trained, we have exported the model. So that it can be used for prediction on real time data.

8.5 Code Module

Code:

```
from flask import Flask, render_template, redirect, request, url_for, send_file
from flask import jsonify, json
from werkzeug.utils import secure_filename

# Interaction with the OS
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# Used for DL applications, computer vision related processes
import torch
import torchvision

# For image preprocessing
from torchvision import transforms

# Combines dataset & sampler to provide iterable over the dataset
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset

import numpy as np
import cv2

# To recognise face from extracted frames
import face_recognition

# Autograd: PyTorch package for differentiation of all operations on Tensors
# Variable are wrappers around Tensors that allow easy automatic differentiation
from torch.autograd import Variable
# 'nn' Help us in creating & training of neural network
from torch import nn
```

```
# Contains definition for models for addressing different tasks i.e. image classification,
object detection e.t.c.
from torchvision import models
from skimage import img_as_ubyte
import warnings
warnings.filterwarnings("ignore")
UPLOAD_FOLDER = 'Uploaded_Files'
video_path = ""
detectOutput = []
app = Flask("__main__", template_folder="templates")
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Creating Model Architecture
class Model(nn.Module):
def __init__(self, num_classes, latent_dim= 2048, lstm_layers=1, hidden_dim=2048,
bidirectional=False):
super(Model, self).__init__()
# returns a model pretrained on ImageNet dataset
model = models.resnext50_32x4d(pretrained= True)
# Sequential allows us to compose modules nn together
self.model = nn.Sequential(*list(model.children())[:-2])
# RNN to an input sequence
self.lstm = nn.LSTM(latent_dim, hidden_dim, lstm_layers, bidirectional)
# Activation function
self.relu = nn.LeakyReLU()
# Dropping out units (hidden & visible) from NN, to avoid overfitting
self.dp = nn.Dropout(0.4)
# A module that creates single layer feed forward network with n inputs and m outputs
self.linear1 = nn.Linear(2048, num_classes)
# Applies 2D average adaptive pooling over an input signal composed of several input planes
self.avgpool = nn.AdaptiveAvgPool2d(1)
def forward(self, x):
batch_size, seq_length, c, h, w = x.shape
# new view of array with same data
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,:]))
im_size = 112
# std is used in conjunction with mean to summarize continuous data
mean = [0.485, 0.456, 0.406]
# provides the measure of dispersion of image grey level intensities
std = [0.229, 0.224, 0.225]
# Often used as the last layer of a nn to produce the final output
sm = nn.Softmax()
# Normalising our dataset using mean and std
inv_normalize = transforms.Normalize(mean=-1*np.divide(mean, std),
std=np.divide([1,1,1], std))
# For image manipulation
def im_convert(tensor):
image = tensor.to("cpu").clone().detach()
image = image.squeeze()
image = inv_normalize(image)
image = image.numpy()
image = image.transpose(1,2,0)
image = image.clip(0,1)
cv2.imwrite('./2.png', image*255)
return image
# For prediction of output
def predict(model, img, path='./'):
# use this command for gpu
# fmap, logits = model(img.to('cuda'))
fmap, logits = model(img.to())
params = list(model.parameters())
weight_softmax = model.linear1.weight.detach().cpu().numpy()
logits = sm(logits)
_, prediction = torch.max(logits, 1)
confidence = logits[:, int(prediction.item())].item()*100
```

```
print('confidence of prediction: ', logits[:, int(prediction.item())].item()*100)
return [int(prediction.item()), confidence]
# To validate the dataset
class validation_dataset(Dataset):
def __init__(self, video_names, sequence_length = 60, transform=None):
self.video_names = video_names
self.transform = transform
self.count = sequence_length
# To get number of videos
def __len__(self):
return len(self.video_names)
# To get number of frames
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)):
faces = face_recognition.face_locations(frame)
try:
top,right,bottom,left = faces[0]
frame = frame[top:bottom, left:right, :]
except:
pass
frames.append(self.transform(frame))
if(len(frames) == self.count):
break
frames = torch.stack(frames)
frames = frames[:self.count]
return frames.unsqueeze(0)

# To extract number of frames
def frame_extract(self, path):
vidObj = cv2.VideoCapture(path)
success = 1
```

```
while success:
success, image = vidObj.read()
if success:
yield image
def detectFakeVideo(videoPath):
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)])
path_to_videos= [videoPath]
video_dataset = validation_dataset(path_to_videos,sequence_length = 20,transform = train_transforms)
# use this command for gpu
# model = Model(2).cuda()
model = Model(2)
path_to_model = 'model/df_model.pt'
model.load_state_dict(torch.load(path_to_model, map_location=torch.device('cpu')))
model.eval()
for i in range(0,len(path_to_videos)):
print(path_to_videos[i])
prediction = predict(model,video_dataset[i],'./')
if prediction[0] == 1:
print("REAL")
else:
print("FAKE")
return prediction@app.route('/', methods=['POST', 'GET'])
def homepage():
if request.method == 'GET':
return render_template('index.html')
return render_template('index.html')
```

```
@app.route('/Detect', methods=['POST', 'GET'])
def DetectPage():
if request.method == 'GET':
return render_template('index.html')
if request.method == 'POST':
video = request.files['video']
print(video.filename)
video_filename = secure_filename(video.filename)
video.save(os.path.join(app.config['UPLOAD_FOLDER'], video_filename))
video_path = "Uploaded_Files/" + video_filename
prediction = detectFakeVideo(video_path)
print(prediction)
if prediction[0] == 0:
    output = "FAKE"
else:
    output = "REAL"
confidence = prediction[1]
data = {'output': output, 'confidence': confidence}
data = json.dumps(data)
os.remove(video_path);
return render_template('index.html', data=data)
app.run(port=3000);
from flask import Flask, render_template, request, jsonify, json
from werkzeug.utils import secure_filename
import os
import torch
import cv2
import face_recognition
from torch import nn
from torchvision import models, transforms
from torch.utils.data import Dataset
import numpy as np

# Define paths for uploaded files
UPLOAD_FOLDER = 'Uploaded_Files'
```

```
video_path = ""
detectOutput = []
# Flask app setup
app = Flask("__main__", template_folder="templates")
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

# Ensure the upload folder exists

```
if not os.path.exists(UPLOAD_FOLDER):
os.makedirs(UPLOAD_FOLDER)
```

# Model Architecture Class Definition

```
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, :]))
```

# Normalization Parameters for Image Preprocessing

```
im_size = 112
```

```
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]
inv_normalize = transforms.Normalize(mean=-1 * np.divide(mean, std), std=np.divide([1, 1,
1], std))
# Image Convert Function
def im_convert(tensor):
image = tensor.to("cpu").clone().detach()
image = image.squeeze()
image = inv_normalize(image)
image = image.numpy()
image = image.transpose(1, 2, 0)
image = image.clip(0, 1)
cv2.imwrite('./2.png', image * 255)
return image

# Prediction Function
def predict(model, img, path='./'):
fmap, logits = model(img.to())
sm = nn.Softmax()
logits = sm(logits)
_, prediction = torch.max(logits, 1)
confidence = logits[:, int(prediction.item())].item() * 100
print('confidence of prediction: ', confidence)
return [int(prediction.item()), confidence]

# Validation Dataset Class
class validation_dataset(Dataset):
def __init__(self, video_names, sequence_length=60, 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)):
faces = face_recognition.face_locations(frame)
try:
    top, right, bottom, left = faces[0]
    frame = frame[top:bottom, left:right, :]
except:
    pass
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)
if not vidObj.isOpened():
raise ValueError(f"Could not open video file {path}")
success = True
while success:
success, image = vidObj.read()
if success:
    yield image
else:
    break
```

# Video Detection Function

```
def detectFakeVideo(videoPath):
train_transforms = transforms.Compose([
transforms.ToPILImage(),
    transforms.Resize((im_size, im_size)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
    ])
    path_to_videos = [videoPath]

    video_dataset=validation_dataset(path_to_videos,sequence_length=20,transform=train_
    transforms)
    model = Model(2)
    path_to_model = 'df_model.pt'

    if not os.path.exists(path_to_model):
    raise FileNotFoundError('Model file not found: df_model.pt')
    model.load_state_dict(torch.load(path_to_model, map_location=torch.device('cpu')))
    model.eval()
    for i in range(0, len(path_to_videos)):
    print(path_to_videos[i])
    prediction = predict(model, video_dataset[i], './')
    if prediction[0] == 1:
    print("REAL")
    else:
    print("FAKE")
    return prediction
    # Flask Route for Home
    @app.route('/', methods=['POST', 'GET'])
    def homepage():
    if request.method == 'GET':
    return render_template('index.html')
    return render_template('index.html')
    # Flask Route for Detecting Fake Videos
    @app.route('/Detect', methods=['POST', 'GET'])
    def DetectPage():
  if request.method == 'GET':
  return render_template('index.html')
  if request.method == 'POST':
```

```
try:
video = request.files['video']
    print(f"Received video: {video.filename}")
    video_filename = secure_filename(video.filename)
    video_path = os.path.join(app.config['UPLOAD_FOLDER'], video_filename)
    video.save(video_path)
        # Detect the video
    prediction = detectFakeVideo(video_path)
    print(f"Prediction: {prediction}")
    # Determine output and confidence
    if prediction[0] == 0:
        output = "FAKE"
    else:
        output = "REAL"
    confidence = prediction[1]
    data = {'output': output, 'confidence': confidence}
    data = json.dumps(data)
    # Clean up the uploaded video
    os.remove(video_path)
    return render_template('index.html', data=data)
    except Exception as e:
    print(f"Error processing the video: {e}")
    return "Error processing the video", 500

    # Run the Flask app
    if __name___== '_main_':
    app.run(debug=True, port=3000)
```

CHAPTER 9

TESTING

System testing involves user training system testing and successful running of the developed proposed system. The user tests the developed system and changes are made according to their needs. The testing phase involves the testing of developed system using various kinds of data.
An elaborate testing of data is prepared and the system is tested using the test data. While testing, errors are noted and the corrections are made. The corrections are also noted for the future use. The users are trained to operate the developed system.
Testing involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should .it is oriented to 'detection'.
System testing is the stage of implementation that is aimed at ensuring that the system works accurately before live operation commences. Testing is vital to the success of the system. System testing makes logical assumption that if all the parts of the system are correct, then the goal will be successfully achieved.
A series of testing are done for the proposed system before the system is ready for the user.

• Testing Methodologies

There are two major types of testing. They are

- White Box Testing
- Black Box Testing

• White Box Testing

White box testing (also known as clear box testing, glass box testing, and transparent box testing and structural testing) is a method of testing software that tests internal structures are working of an application, an opposed to its functionality. In white box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tests are choosing inputs to exercise paths through the code and determine outputs.

While white box testing can be applied at the unit, integration and system level of the software testing process, it is usually done at the unit level. It can test paths within a unit, path between units during integration, and between sub systemsduring system level tests.

This method of test design can uncover many of our problems it might not detect unimplemented parts of the specification or missing requirements.

White box test design techniques include:

- Control flow testing
- Data flow testing
- Branch testing
- Statement coverage
- Decision coverage

- **Black Box Testing**

Black box testing is a method of software testing that examines the functionality of an (see white box testing). This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically compares most if not all higher-level testing but can also dominate unit testing as well. The following are the types of testing:

- Unit Testing
- Integration Testing
- Validation Testing
- Verification Testing
- User Acceptance Testing

- **Unit Testing:** Unit testing focuses verification efforts and the smallest unit of the software design, the module. This is also known as “module testing”. The modules are tested separately, this testing was carried out during programming stage itself, in this testing each module is found to be working satisfactory as regards to the expected output from the module.
- **Integration Testing:** Data can be lost across an interface: one module can have adverse efforts on another. Integration testing is the systematic testing for construction of program structure, while at the same time conducting test to uncover errors associated with in the interface. A correction is difficult because the isolation of the cause is complicated by the cast expanse of the entire program.

- **Validation Testing:** At the conclusion of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final series of software tests begins validation test has been conducted of the two possible conditions exists. One is the function or performance characteristics confirmed to specifications and are accepted and the other is deviation from specifications in uncovered and efficiency list is created.
- **Verification Testing:** Verification is a fundamental concept in software design. This is the bridge between customer requirements and an implementation that specifies those requirements. This is verifiable if it can be demonstrated that the testing will result in an implementation that satisfies the customer requirements.
- **User Acceptance Testing:** User acceptance testing of a system is the key factor of the success of the nay system. The system under study is tested for the user acceptance by constantly keeping in touch with their susceptive system users at any time of developing and making changes whenever required.

9.1 Test Cases and Test Results

- **Test Cases**

Case ID	Test Case Description	Expected Result	Actual Result	Status
1	Upload the video inputs of various formats (e.g. MP4, AVI, etc.)	Real/Fake	Real/Fake	Pass
2	Upload corrupted or invalid video files	Error Message: Only video files allowed	Error Message: Only video files allowed	Pass
3	Upload the files with video of different resolution and frame rates	If frames less than100 and resolution greater than 200MB an error message is raised else the output will be Real/Fake	If frames less than 100 and resolution greater than 200MB an error message is raised else the output will be Real/Fake	Pass
4	Upload video with multiple faces	Real/Fake	Fake	Pass
5	Upload video with no faces	An error message will be shown on the	An error message shown on the scree	Pass

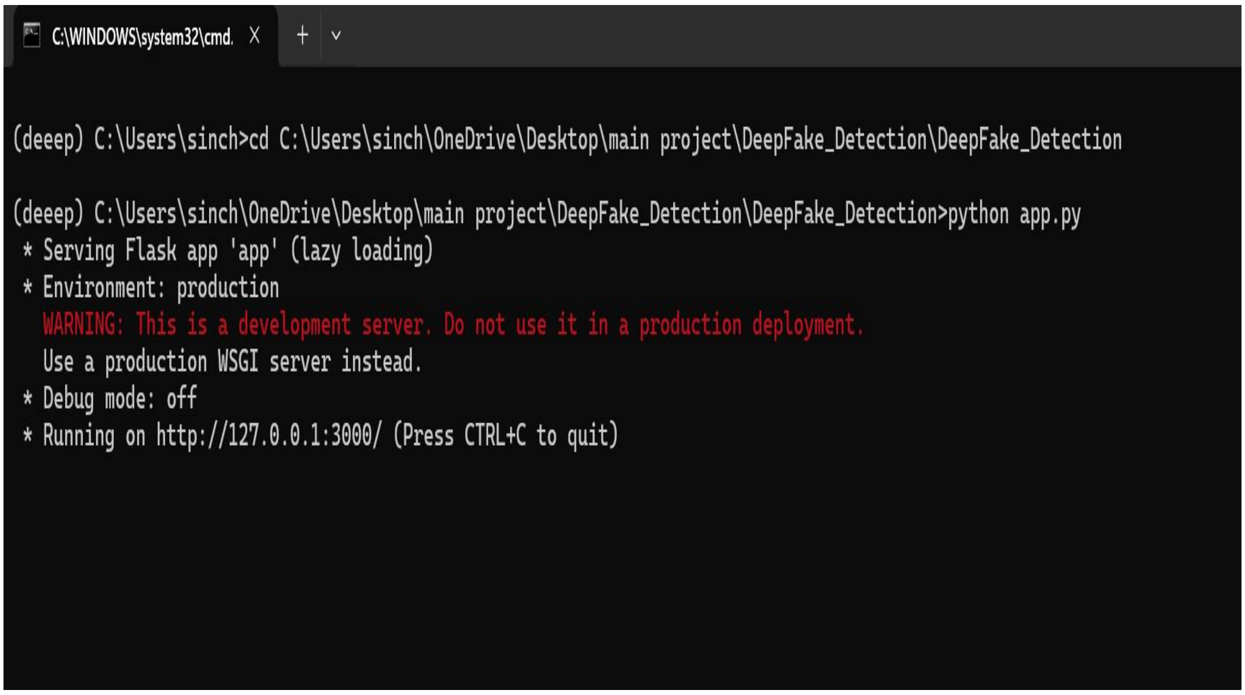
		screen		
6	Upload a real video	Real	Real	Pass
7	Upload a fake video	Fake	Fake	Pass

Table 9.1.1: Test Cases

CHAPTER 10

RESULTS

- **Screenshots**

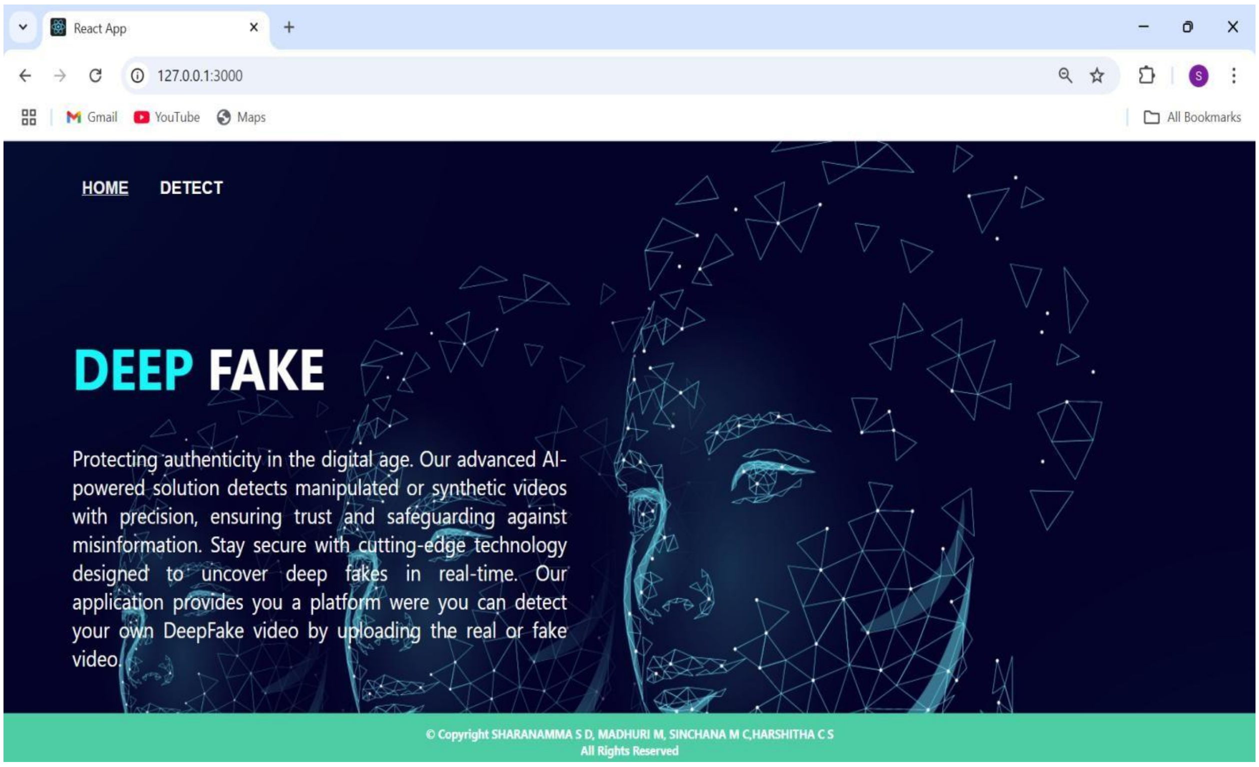


```
C:\WINDOWS\system32\cmd.

(deeep) C:\Users\sinch>cd C:\Users\sinch\OneDrive\Desktop\main project\DeepFake_Detection\DeepFake_Detection

(deeep) C:\Users\sinch\OneDrive\Desktop\main project\DeepFake_Detection\DeepFake_Detection>python app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)
```

Fig 10.1 Address link in Terminal

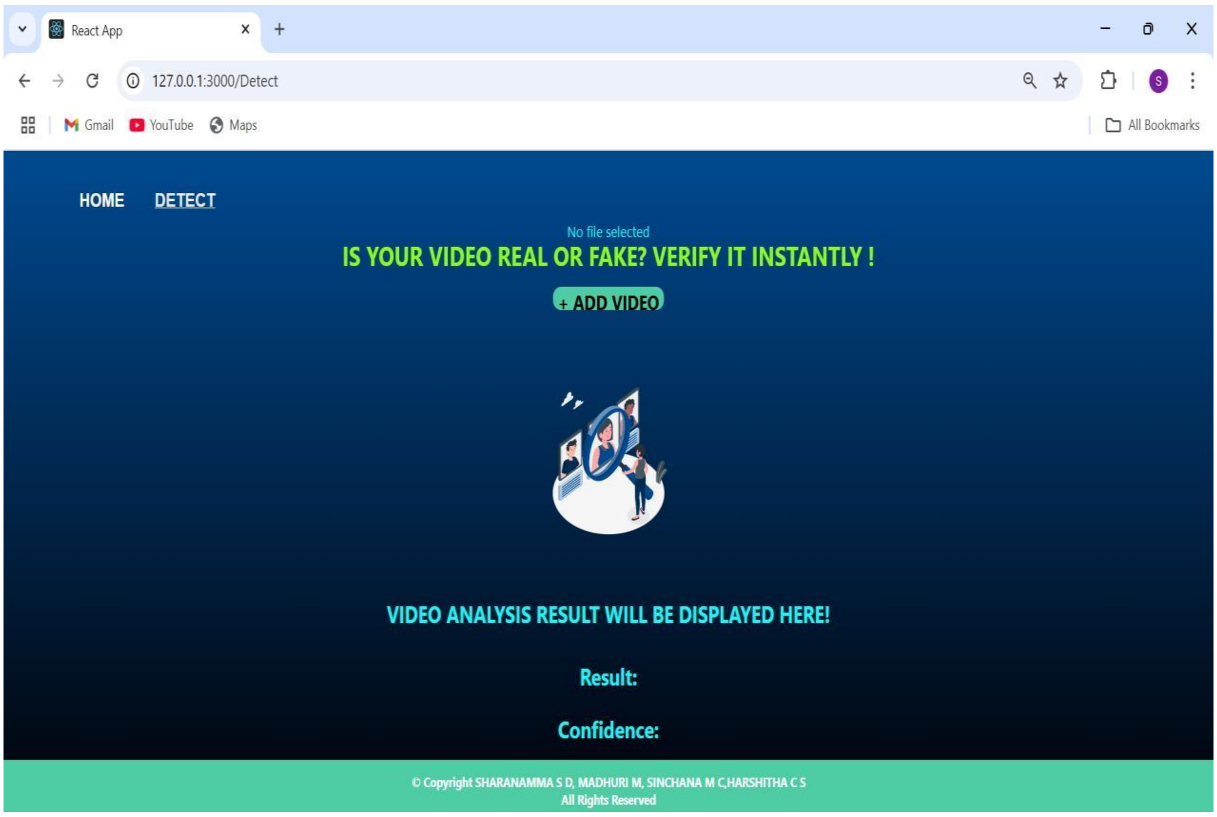


DEEP FAKE

Protecting authenticity in the digital age. Our advanced AI-powered solution detects manipulated or synthetic videos with precision, ensuring trust and safeguarding against misinformation. Stay secure with cutting-edge technology designed to uncover deep fakes in real-time. Our application provides you a platform were you can detect your own DeepFake video by uploading the real or fake video.

© Copyright SHARANAMMA S D, MADHURI M, SINCHANA M C,HARSHITHA C S
All Rights Reserved

Fig 10.2 Home page



IS YOUR VIDEO REAL OR FAKE? VERIFY IT INSTANTLY !

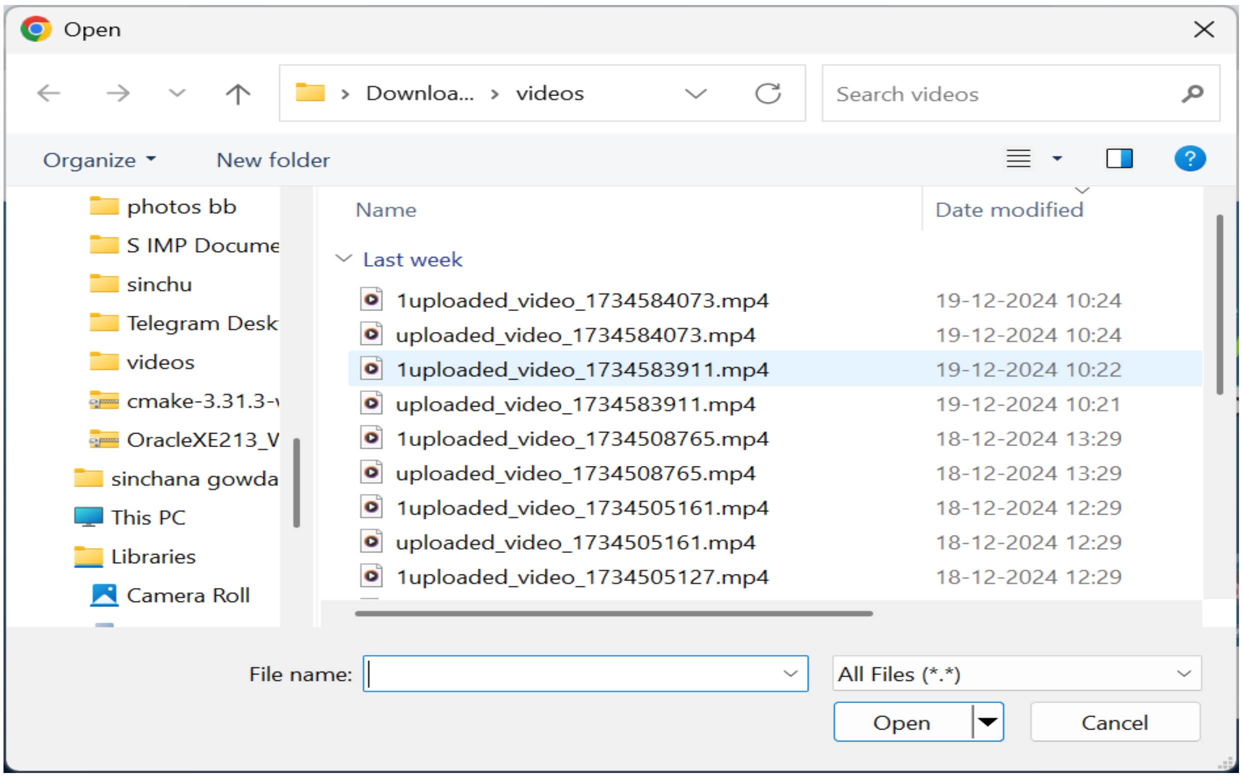
VIDEO ANALYSIS RESULT WILL BE DISPLAYED HERE!

Result:

Confidence:

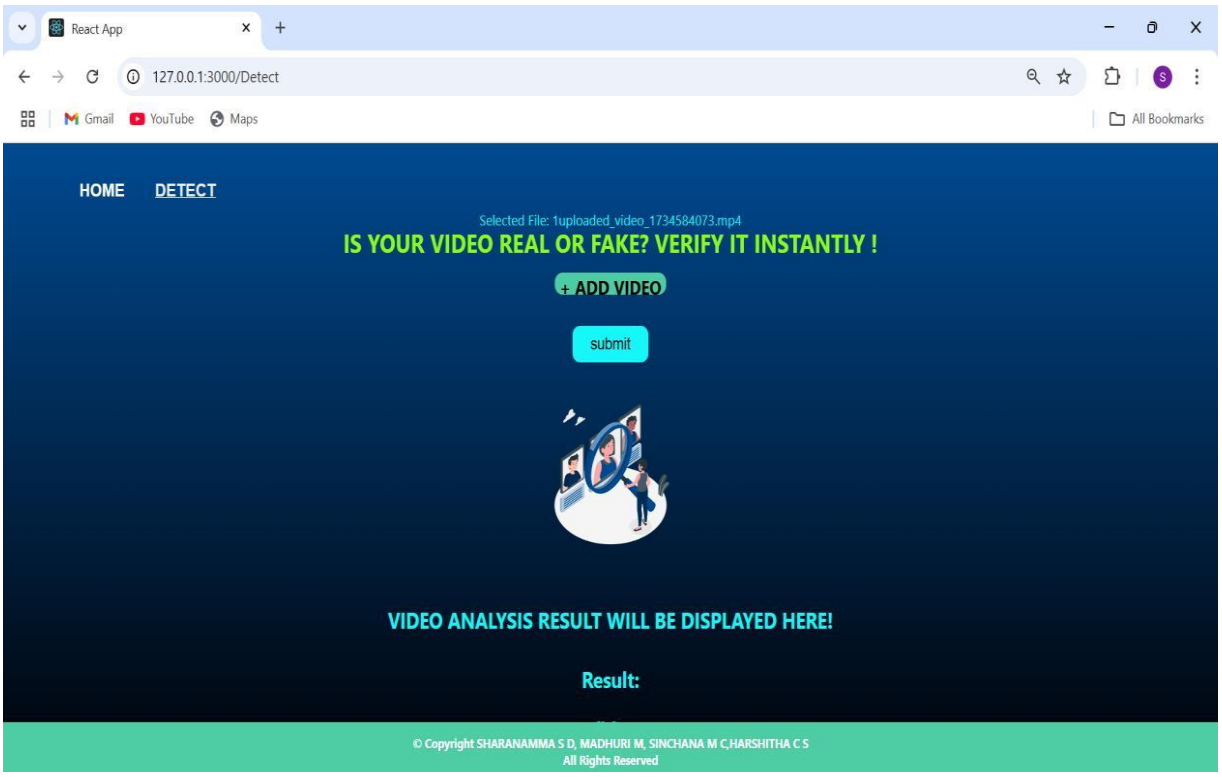
© Copyright SHARANAMMA S D, MADHURI M, SINCHANA M C,HARSHITHA C S
All Rights Reserved

Fig 10.3 Detect page



Name	Date modified
1uploaded_video_1734584073.mp4	19-12-2024 10:24
uploaded_video_1734584073.mp4	19-12-2024 10:24
1uploaded_video_1734583911.mp4	19-12-2024 10:22
uploaded_video_1734583911.mp4	19-12-2024 10:21
1uploaded_video_1734508765.mp4	18-12-2024 13:29
uploaded_video_1734508765.mp4	18-12-2024 13:29
1uploaded_video_1734505161.mp4	18-12-2024 12:29
uploaded_video_1734505161.mp4	18-12-2024 12:29
1uploaded_video_1734505127.mp4	18-12-2024 12:29

Fig 10.4 Selecting videos



Selected File: 1uploaded_video_1734584073.mp4

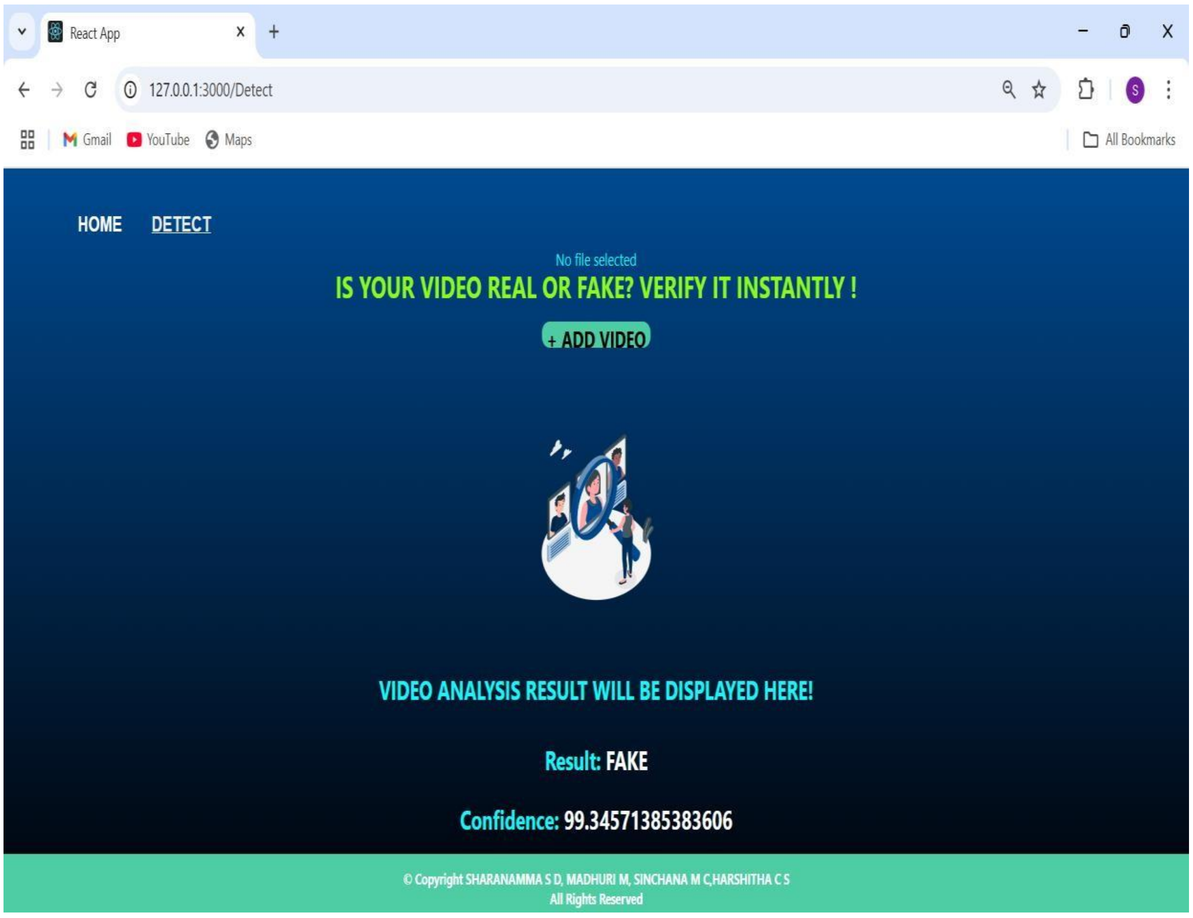
IS YOUR VIDEO REAL OR FAKE? VERIFY IT INSTANTLY !

VIDEO ANALYSIS RESULT WILL BE DISPLAYED HERE!

Result:

© Copyright SHARANAMMA S D, MADHURI M, SINCHANA M C,HARSHITHA C S
All Rights Reserved

Fig 10.5 Video analysis



No file selected

IS YOUR VIDEO REAL OR FAKE? VERIFY IT INSTANTLY !

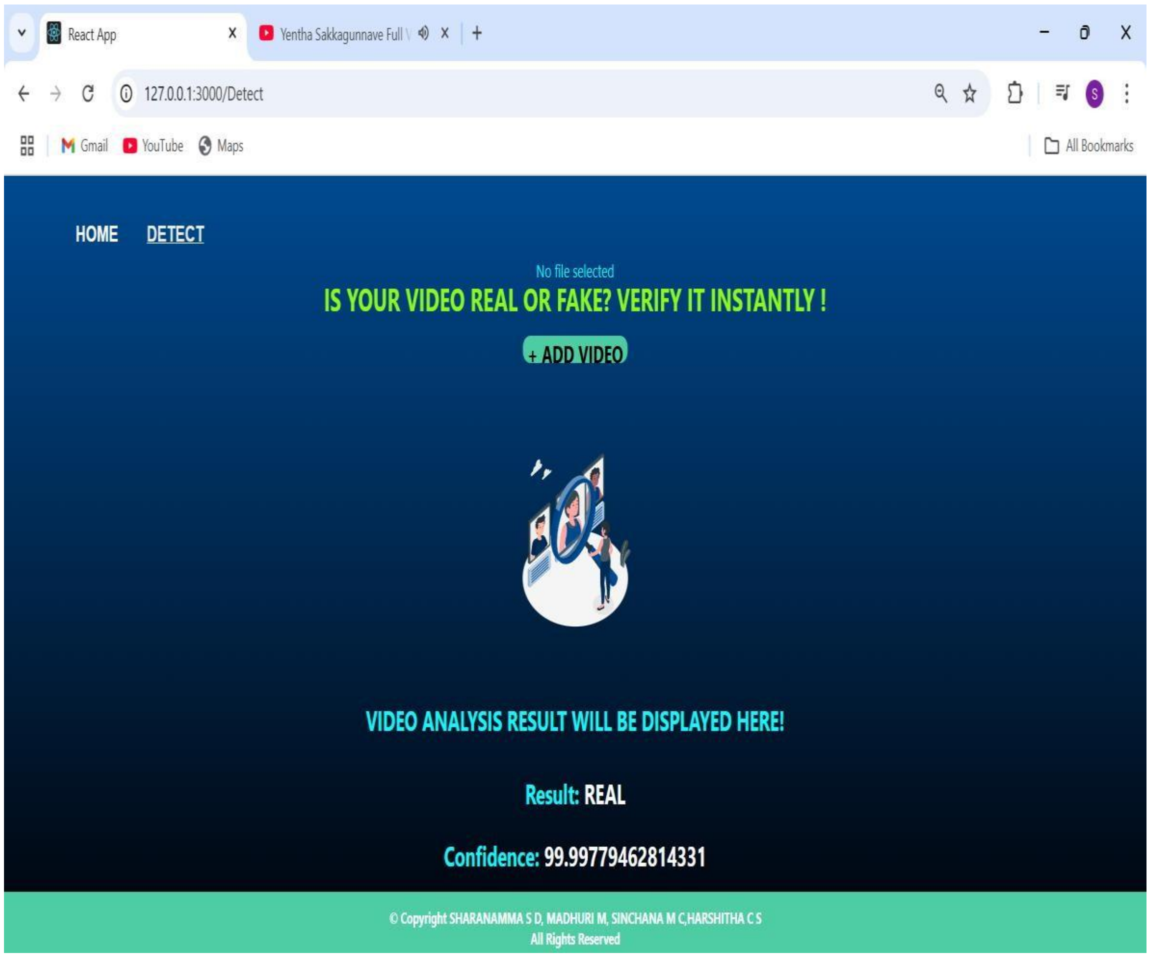
VIDEO ANALYSIS RESULT WILL BE DISPLAYED HERE!

Result: FAKE

Confidence: 99.34571385383606

© Copyright SHARANAMMA S D, MADHURI M, SINCHANA M C,HARSHITHA C S
All Rights Reserved

Fig 10.6 Result of Fake Video



IS YOUR VIDEO REAL OR FAKE? VERIFY IT INSTANTLY !

VIDEO ANALYSIS RESULT WILL BE DISPLAYED HERE!

Result: REAL

Confidence: 99.99779462814331

© Copyright SHARANAMMA S D, MADHURI M, SINCHANA M C,HARSHITHA C S
All Rights Reserved

Fig 10.7 Result of Real video analysis

CONCLUSION

We presented a neural network-based approach to classify the video as deep fake or real, along with the confidence of proposed model. Our method is capable of predicting the output by processing 1 second of video (10 frames per second) with a good accuracy. We implemented the model by using pre-trained ResNext CNN model to extract the frame level features and LSTM for temporal sequence processingto spot the changes between the t and t-1 frame. Our model can process the video in the frame sequence of 10,20,40,60,80,100.Deep fake videos, generated through advanced AI techniques, have raised significant concerns in various domains, such as security, entertainment, and politics. Detecting these manipulated videos is crucial to prevent misinformation and other malicious activities. Convolutional Neural Networks (CNNs) have emerged as one of the most effective methods for detecting deep fake videos due to their ability to process and analyze visual data efficiently.

REFERENCES

[1] Karandikar, A. (2020). Deepfake video detection using convolutional neural network. International Journal of Advanced Trends in Computer Science and Engineering, 9(2), 1311–1315. https://doi.org/10.30534/ijatcse/2020/62922020

[2] Koçak, A., Alkan, M., & Arıkan, S. M. (2024). Deepfake video detection using convolutional neural network-based hybrid approach. Journal of Polytechnic.
https://doi.org/10.2339/politeknik.1523983

[3] Yuezun Li, and SiweiLyu, "Exposing DeepFake Videos By Detecting Face Warping Artifacts", arXiv:1811.00656v3 [cs.CV] 22 May 2019
https://doi.org/10.1016/S0969-4765(19)30137-7

[4] FalkoMatern , Christian Riess and Marc Stamminger, "Exploiting Visual Artifacts to Expose Deepfakes and Face Manipulations ", 2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)
https://doi.org/10.1109/WACVW.2019.00020

[5] David Güera,Edward J. Delp, "Deepfake Video Detection Using Recurrent Neural Networks", 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS).
https://doi.org/10.1109/AVSS.2018.8639163

[6] Shuo Yuan,Xinguo Yu,Abdul Majid, "Deepfake Video Detection Using Neural Networks", 4th International Conference on Control and Robotics Engineering (ICCRE), 20-23 April 2019. https://doi.org/10.1109/ICCRE.2019.8724212

[7] Harsh Vardhan1, Naman Varshney2, Manoj Kiran R3, Pradeep R4, Dr. Latha N.R5, "Deep Fake Video Detection" International Research Journal on Advanced Engineering Hub (IRJAEH) e ISSN: 2584-2137
https://doi.org/10.47392/IRJAEH.2024.0117

[8] Awotunde JB, Jimoh RG, Imoize AL, Abdulrazaq AT, Li C-T, Lee C-C. An Enhanced Deep Learning-Based Deep Fake Video Detection and Classification System. Electronics.2023; 12(1):87.
https://doi.org/10.3390/electronics1

[9] L. Minh Dang, Syed Ibrahim Hassan, Suhyeon Im, Hyeonjoon Moon, Face image manipulation detection based on a convolutional neural network, Expert Systems with Applications, Volume 129, 2019, Pages 156-168, ISSN 0957-4174.
https://doi.org/10.1016/j.eswa.2019.04.005.

[10] Lin Zhang, Xin Wang, Erica Cooper, Nicholas Evans, and Junichi Yamagishi. 2022. The Partial Spoof Database and Countermeasures for the Detection of Short Fake Speech Segments Embedded in an Utterance. IEEE/ACM Trans. Audio, Speech and Lang. Proc. 31 (2023), 813–825.
https://doi.org/10.1109/TASLP.2022.3233236