Real-Time Deepfake Detection Platform with Advanced ResNet and Temporal Analysis

A Main Project thesis submitted in partial fulfillment of requirements for the award of degree for VIII Semester

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE SND ENGINEERING

by

B. S. S. DVIJESH (21131A0522)

D. SAI SRUJANA (21131A0566)

CH. AVINASH (21131A0535)

A. GOWTHAM KUMAR (21131A0562)

Under the esteemed guidance of

Ms. G. Sathee Lakshmi,

Assistant Professor,

Department of Computer Science and Engineering

COLLEGE OF ENGINEERING
(Autonomous)

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)

Approved by AICTE & Andhra University, Visakhapatnam 2022-23

(Affiliated to JNTUK, upto 2021-22)

Visakhapatnam

2024-2025

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING
(Autonomous)
Approved by AICTE & Affiliated to Andhra University, Visakhapatnam from 2022-23
(Affiliated to JNTUK, Kakinada upto 2021-22)
Accredited by NAAC at 'A++' Grade for 7 years in the 3rd Cycle

CERTIFICATE

This is to certify that the main project entitled **"Real-Time Deepfake Detection Platform with Advanced ResNet and Temporal Analysis"** being submitted by

B. S. S. DVIJESH (21131A0522)
D. SAI SRUJANA (21131A0566)
CH. AVINASH (21131A0535)
A. GOWTHAM KUMAR (21131A0562)

in partial fulfillment for the award of degree "Bachelor of Technology" in Computer Science and Engineering to the Jawaharlal Nehru Technological University, Kakinada is a record of bonafide work carried under my guidance and supervision during VIII semester of the academic year 2024-2025.

The results embodied in this record have not been submitted to any other university or institute for the award of any Degree or Diploma.

Guide
Ms. Sathee Laxmi
Assistant Professor
Deapartment of CSE
GVPCE(A)

Head of the Department
Dr. C. V. Guru Rao
Professor and
Head of CSE
Department of CSE
GVPCE(A)

External Signature

DECLARATION

We hereby declare that this project entitled **"Real-Time Deepfake Detection Platform with Advanced ResNet and Temporal Analysis"** is a bonafide work done by us and submitted to Department of Computer Science and Engineering, G. V. P College of Engineering (Autonomous) Visakhapatnam, in partial fulfillment for the award ofthe degree of B. Tech is of our own and it is not submitted to any other universityor has been published anytime before.

PLACE: VISAKHAPATNAM
DATE:

B. S. S. DVIJESH (21131A0522)
D. SAI SRUJANA (21131A0566)
CH. AVINASH (21131A0535)
A. GOWTHAM KUMAR (21131A0562)

ACKNOWLEDGEMENT

We would like to express our deep sense of gratitude to our esteemed institute **Gayatri Vidya Parishad College of Engineering (Autonomous)**, which has provided us an opportunity to fulfill our cherished desire.

We express our sincere thanks to our principal **Dr. A. B. KOTESWARA RAO, Gayatri Vidya Parishad College of Engineering (Autonomous)** for his encouragement to us during this project, giving us a chance to explore and learn new technologies in the form of mini projects.

We express our sincere thanks to our principal **Dr. C. V. GURU RAO, Professor, Head of the Department of Computer Science and Engineering, Gayatri Vidya Parishad College of Engineering (Autonomous)** for giving us an opportunity to do the project in college.

We express our profound gratitude and our deep indebtedness to our Coordinator **Dr. CH. SITA KUMARI**, Associate Proffesor, Department of Computer Science and Engineering, whose valuable guidance helped us a lot in realizing our project.

We also thank our Project Guide, **Ms. G. Sathee Laxmi**, Assistant Proffesor, Department of Computer Science and Engineering, for the kind suggestions and guidance for the successful completion of our project work.

B. S. S. DVIJESH (21131A0522)
D. SAI SRUJANA (21131A0566)
CH. AVINASH (21131A0535)
A. GOWTHAM KUMAR (21131A0562)

ABSTRACT

In today's digital landscape, deepfake technology has emerged as a powerful yet concerning tool capable of manipulating media with high precision. The increasing accessibility of AI-driven deepfake generation poses serious challenges to information integrity, personal security, and digital trust. This project presents a deep learning-based approach for detecting deepfake videos, leveraging a combination of convolutional neural networks (CNNs) and long short-term memory (LSTM) networks to analyze and classify video frames. Specifically, we employ a pre-trained ResNext CNN model for feature extraction and an LSTM network for temporal sequence processing, ensuring the detection of subtle frame-by-frame inconsistencies. The system is designed to process short video clips, providing real-time classification results along with confidence scores. The implementation includes a web-based interface for user-friendly access and aims to contribute to the ongoing fight against misinformation by providing a robust deepfake detection mechanism.

Keywords: Deepfake Detection, Machine Learning, Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), ResNext Model

INDEX

1. INTRODUCTION

With the rapid advancement of artificial intelligence, deepfake technology has emerged as a serious threat to digital authenticity. Deepfakes, created using generative models like GANs[1]. and autoencoders, can manipulate video and audio to produce hyper-realistic but entirely fake content[2]. While this technology has positive applications in media and entertainment, it also poses significant risks, including political misinformation[3], identity theft[4], and cybercrimes[5].

To address this challenge, we propose a deep learning-based solution for detecting deepfake videos. Our method leverages a combination of ResNeXt[6]-based convolutional neural networks (CNNs) for frame-level feature extraction and Long Short-Term Memory (LSTM)[7] networks. This approach enables our model to distinguish between real and AI-generated videos effectively. Additionally, a user-friendly interface allows individuals to upload videos for verification, ensuring accessibility and ease of use.

1.1. Purpose

The primary goal of this project is to develop an AI-driven system capable of identifying deepfake videos with high accuracy. By analyzing both spatial and temporal features of video frames, our model aims to provide a reliable method for detecting manipulated media. This technology is crucial in combating misinformation, protecting individuals from malicious content, and ensuring digital integrity in an era where AI-generated content is becoming increasingly sophisticated.

1.2. Scope

This project focuses on detecting deepfake videos using advanced deep learning techniques. Our solution is trained on large-scale datasets, including FaceForensics++[8], Deepfake Detection Challenge[9], and Celeb-DF[10], ensuring robustness across various deepfake generation methods. The scope includes:

- **Real-time Detection** – Providing a practical and efficient system for analyzing videos in real-world scenarios.
- **User Interface** – Developing an interactive front-end application for easy video upload and classification.
- **Model Optimization** – Enhancing accuracy through diverse training data and fine-tuning techniques.

2. Literature Survey

2.1. Literature Survey

2.1.1. No One Can Escape, IEEE Access, Volume 7, September 2019[11]:

→ The paper introduces a general approach for detecting tampered and AI-generated images by leveraging edge information extracted using the Scharr operator, which is further transformed into a Gray-Level Co-occurrence Matrix (GLCM) as a feature descriptor.

→ The extracted GLCM features are processed by a deep neural network to effectively classify images as real or forged. The study highlights the robustness of combining edge detection with deep learning for image forensics.

2.1.2. Face Warping Artifacts[12]:

→ Used an approach to detect artifacts by comparing the generated face areas and their surrounding regions with a dedicated Convolutional Neural Network model. In this work, there were two-fold of Face Artifacts.

→ Their method is based on the observations that the current deepfake algorithm can only generate images of limited resolutions, which then need to be further transformed to match the faces to be replaced in the source video.

→ Their method has not considered temporal analysis of the frames.

2.1.3. Detection by Eye Blinking[13]:

→ They described a new method for detecting deepfakes by eye blinking as a crucial parameter leading to the classification of the videos as deepfake or pristine.

→ The Long-term Recurrent Convolution Network (LRCN) was used for temporal analysis of the cropped frames of eye blinking.

→ As today the deepfake generation algorithms have become powerful that lack of eye blinking can not be the only clue for detection of the deepfakes. There must be certain other parameters must be considered for the detection of deepfakes like teeth enchantment, wrinkles on faces, wrong placement of eyebrows, etc

2.1.4. Deepfake Video Detection Using Recurrent Neural Networks[14]:

→ This research explores use of Recurrent Neural Networks (RNNs) to detect deepfake videos by modeling the temporal dependencies across video frames.

→ Frame-level features are extracted using a pre-trained convolutional model (ImageNet) and subsequently processed by an RNN to capture sequential inconsistencies indicative of deepfakes.

→ The evaluation was conducted on the HOHA dataset, comprising 600 videos, though the limited dataset size and lack of diverse real-world samples pose concerns regarding generalization to real-time applications

2.1.5. Detecting Deepfake Images Using DL Techniques and Explainable AI Methods[15]:

→ The paper investigates the effectiveness of deep learning models to classify deepfake and authentic images, emphasizing the role of explainable AI techniques to enhance model interpretability.

→ CNNs, including InceptionV3 InceptionResNetV2, DenseNet201, are utilized for image classification tasks.

→ The research highlights the importance of model transparency through explainable AI techniques, aiding in understanding the decision-making process behind deepfake detection models

2.2. Literature Survey Summary

The study "No One Can Escape" proposes a method that extracts edge information using the Scharr operator and transforms it into a Gray-Level Co-occurrence Matrix (GLCM) for feature representation. A deep neural network processes these features to classify images as real or forged, demonstrating the effectiveness of combining edge detection with deep learning for image forensics. Similarly, "Face Warping Artifacts" identifies inconsistencies in face-warping techniques by analyzing differences between generated face areas and surrounding regions using a CNN model. However, it does not consider temporal analysis, which is crucial for video-based deepfake detection.

Other approaches focus on video-based deepfake detection, such as the "Detection by Eye Blinking" method, which utilizes Long-term Recurrent Convolution Networks (LRCN) to analyze eye-blinking patterns. However, due to advancements in deepfake generation, additional clues such as teeth misalignment and eyebrow placement need to be considered. The study on "Deepfake Video Detection Using Recurrent Neural Networks" explores RNNs to model temporal inconsistencies across video frames but faces challenges due to limited dataset diversity. Lastly, the research on "Detecting Deepfake Images Using DL Techniques and Explainable AI" examines

CNN models like InceptionV3 and DenseNet201 for image classification while emphasizing explainable AI techniques to enhance model interpretability, highlighting the need for transparency in deepfake detection models.

2.3. **Objectives**

- Our project aims at discovering the distorted truth of the deep fakes.
- Our project will reduce the Abuses' and misleading of the common people on the world wide web.
- Our project will distinguish and classify the video as deepfake or pristine.
- Provide a easy to use system for used to upload the video and distinguish whether the video is real or fake.

3. Existing System

3.1. Methodology

Deepfake detection has seen the development of various models and systems aimed at identifying manipulated media. This section covers the methodologies employed by the prominent existing systems:

3.1.1. XceptionNet[16]

→ Utilizes an advanced Convolutional Neural Network (CNN) architecture based on depth wise separable convolutions.

→ Focuses on spatial inconsistencies by analyzing pixel-level features, especially in facial regions.

→ Trained on large datasets like FaceForensics++ to improve manipulation detection.

3.1.2. ForensicTransfer[17]

→ Implements transfer learning to detect deepfakes with minimal label data.

→ Utilizes few-shot learning, enabling adaptability to new domains with limited training samples.

→ Builds on a pretrained CNN model to enhance cross-dataset generalization.

3.2. Performance Measure

3.2.1. XceptionNet

→ Achieves high accuracy on datasets like FaceForensics++ (>90%).

→ Performance drops when tested on unseen manipulation techniques.

3.2.2. Forensic Transfer

→ Performs well in cross-dataset evaluations due to transfer learning capabilities.

→ Effectiveness is notable even with few labeled samples, though performance varies with the domain gap.

3.3. Drawbacks

Despite advancements, existing systems have several limitations:

3.3.1. Dataset Dependency:

→ Most deepfake detection models are trained on specific datasets and manipulation techniques.

→ These models struggle to generalize when exposed to novel or unseen deepfake methods.

→ Overfitting to particular datasets reduces their effectiveness on real-world data where manipulations differ from the training samples.

3.3.2. Vulnerability to Preprocessing Techniques:

→ Deepfakes often undergo post-processing steps like compression, blurring, and resizing to disguise manipulation artifacts.

→ Such techniques mask key features that detection models rely on, making it difficult to identify tampering.

→ Video platforms like social media compress media files, further degrading detection accuracy.

3.3.3. Difficulty in Detecting Subtle Manipulations

→ While blatant deepfakes can be detected with higher accuracy, subtle manipulations (like slight facial expression changes or minimal alterations in voice) are harder to identify.

→ Advanced deepfake generation techniques create more realistic and less detectable fakes.

4. PROPOSED SYSTEM

4.1. System Architecture

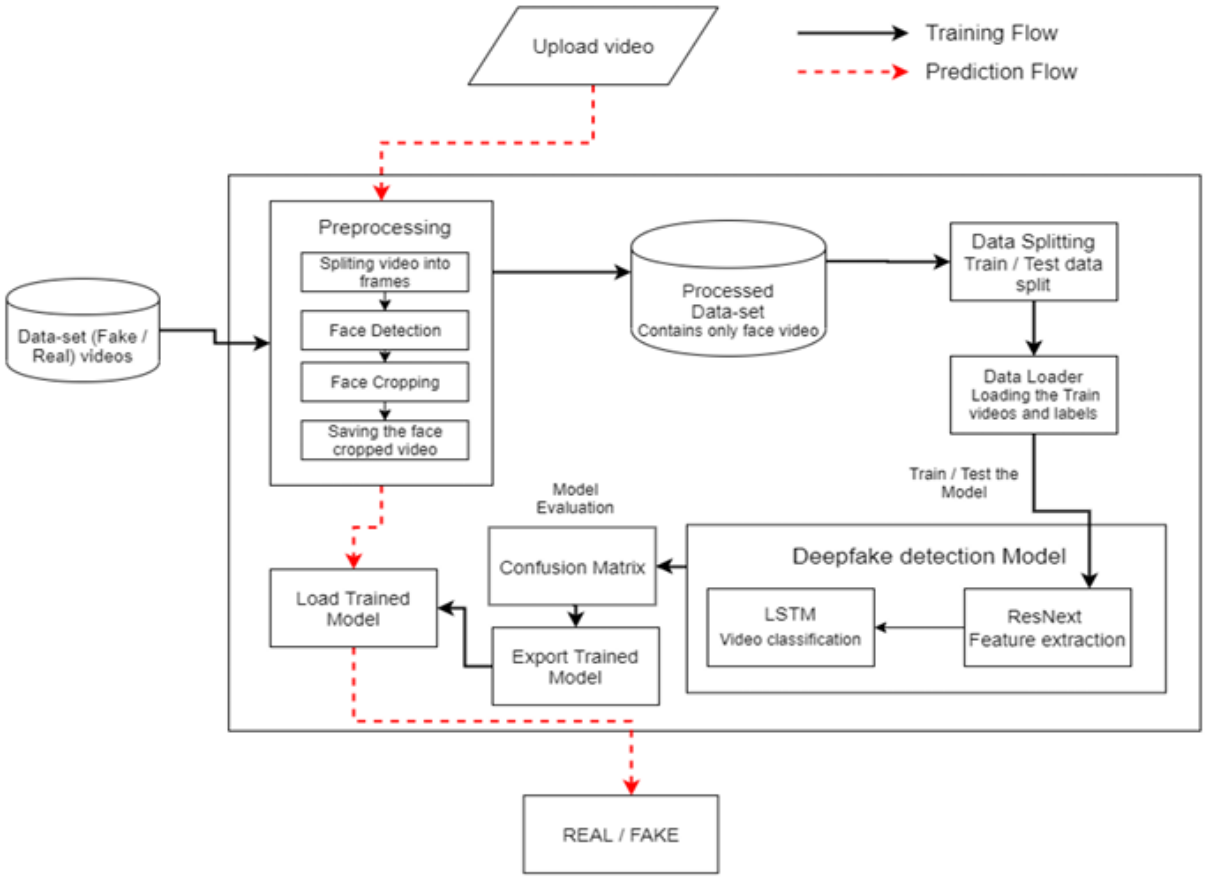

Fig 4.1 - System Architecture

4.2. Modules

4.2.1. Dataset Gathering

To develop a robust deepfake detection model, datasets were collected from multiple sources, including FaceForensics++[8] (FF), Deepfake Detection Challenge[9] (DFDC), and Celeb-DF[10]. A balanced dataset was curated with 50% real and 50% fake videos to prevent model bias. Since DFDC contained some audio-altered videos, these were removed through preprocessing, ensuring that only visual deepfake detection was considered. The final dataset comprised 6,448+ videos (3,310 real and 3,138 fake), maintaining diversity for effective model training.

4.2.2. Preprocessing

Preprocessing aimed at reducing noise and focusing only on the relevant facial regions. The videos were first split into frames, and face detection[18] was applied to extract only the face region, discarding frames without faces[19][20]. To standardize frame counts, a threshold of 150 frames per video was set, ensuring manageable computational requirements while preserving essential sequential information. The preprocessed frames were then reconstructed into new videos at 30 FPS and 112×112 resolution, ready for training.

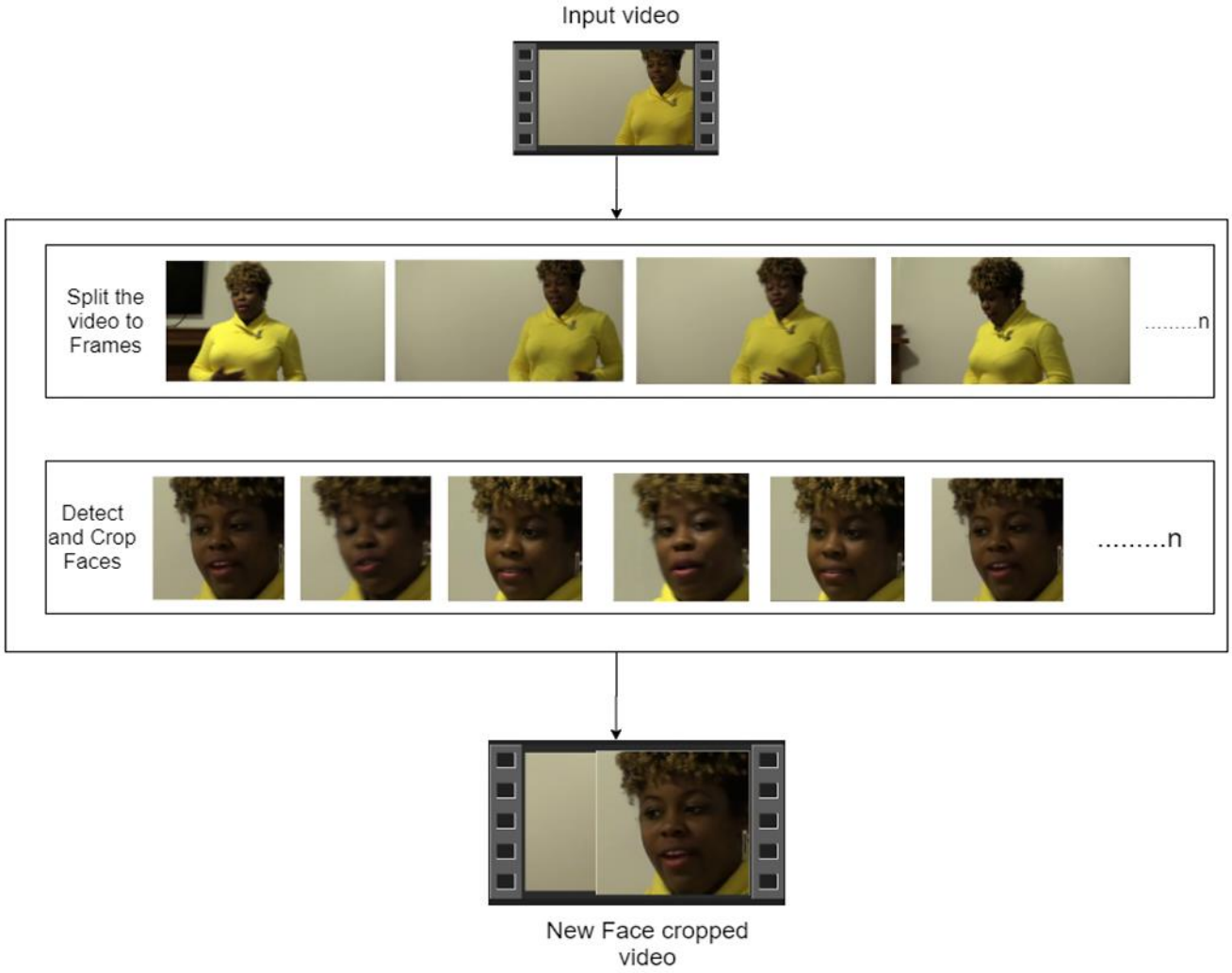

Fig 4.2 : Preprocessing of a sample video

4.2.3. Dataset Split

For effective training and evaluation, the dataset was split into 70% training and 30% testing data, ensuring an equal distribution of real and fake videos across both sets. This balanced approach prevents overfitting and allows the model to generalize better to unseen data. The training set contained 4,200 videos, while the testing set had 1,800 videos, ensuring an appropriate amount of data for model evaluation.

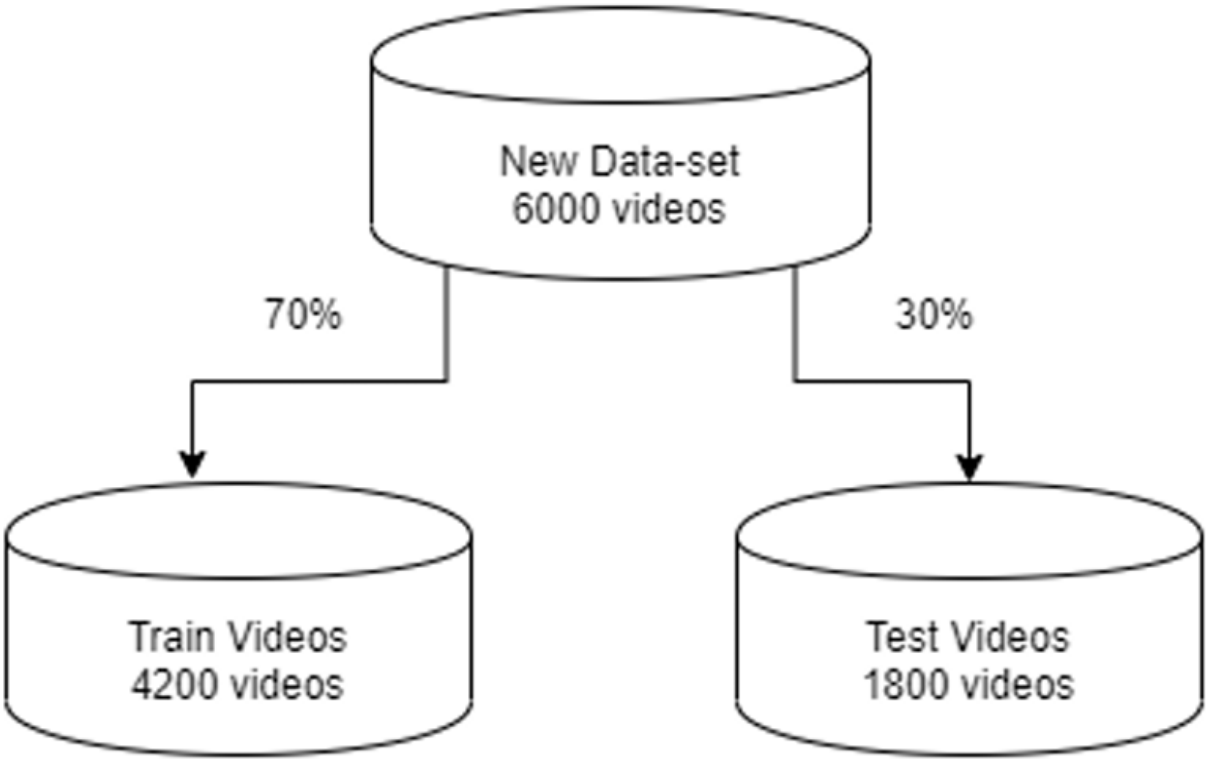

Fig. 4.3 : Dataset split

4.2.4. Model Architecture

The deepfake detection model follows a hybrid architecture combining CNN (ResNeXt) for feature extraction and LSTM for sequential analysis. Pre-trained ResNeXt-50 was used to extract 2048-dimensional feature vectors from video frames, which were then fed into an LSTM network to analyze temporal dependencies. The model was fine-tuned using Leaky ReLU[21] activation, adaptive average pooling, and a SoftMax[22] classifier to predict real vs. fake labels with confidence scores. Batch size was set to 4 to optimize computational efficiency.

4.2.5. Hyper-parameter tuning

The model was fine-tuned using Adam[23] optimizer with a learning rate of 1e-5 and weight decay of 1e-3 for optimal convergence. Cross-entropy loss was chosen due to the classification nature of the task. Batch training was employed with a batch size of 4, ensuring efficient learning given computational constraints.

4.3. Flow Diagram

4.3.1. Data Flow Diagram

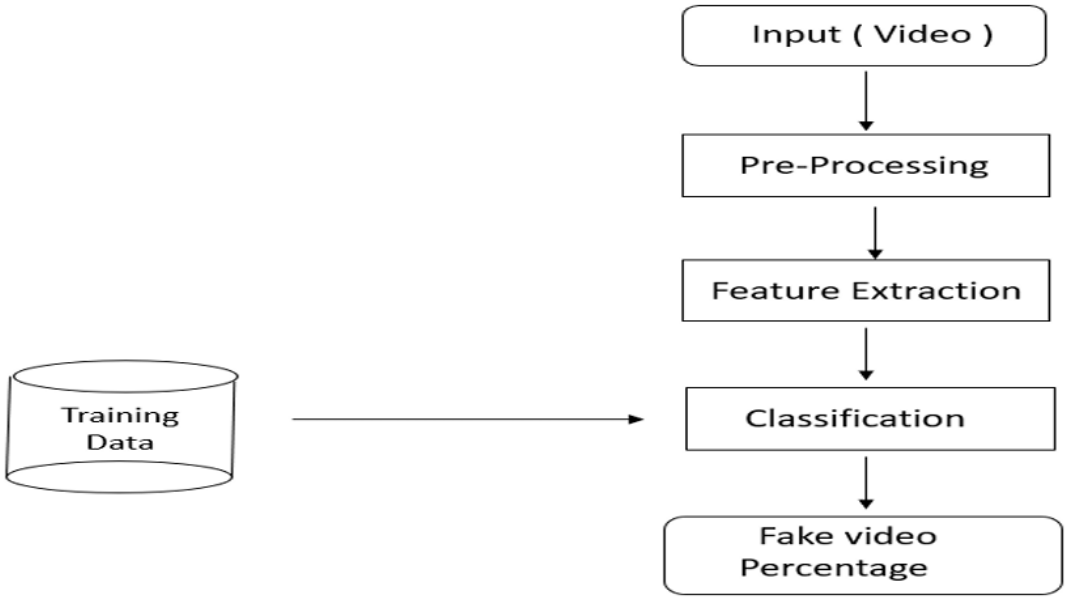

Fig 4.4 Data flow Diagram

Input (Video) - The user provides a video as input to the system.

Pre-Processing - Extract frames of faces from the video at a fixed frame rate

Feature Extraction - Used ResNeXt-50 to extract meaningful features from the face images.

Classification - Use a **Long Short-Term Memory (LSTM) network** to analyze sequential features from frames.

Fake Video Percentage - The model outputs a probability score indicating how likely the video is fake.

- If the score is above a threshold (i.e, ., 50%), the video is classified as fake.
- The final result is displayed to the user.

4.3.2. Training Work Flow Diagram

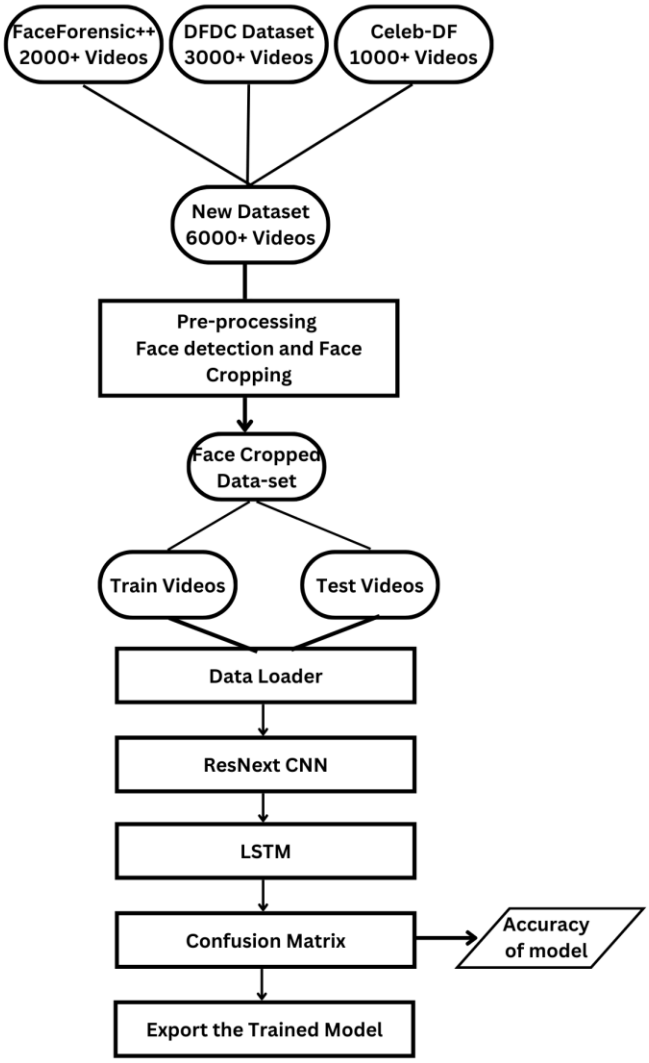

Fig 4.5 Training Workflow

4.4. Methodology

4.4.1. Data Collection & Preprocessing

To build a robust and unbiased model, we carefully curated a large dataset by integrating multiple publicly available deepfake datasets:

- **FaceForensics++ (FF)** – Contains real and manipulated face videos created using different face-swapping techniques.
- **Deepfake Detection Challenge (DFDC)** – A large-scale dataset consisting of real and fake videos, including some with audio alterations.
- **Celeb-DF** – A dataset focused on deepfake detection with high-quality fake videos mimicking real-world scenarios.

Dataset Cleaning & Balancing

- **Frame Extraction & Face Cropping**: Each video was split into frames at a rate of 30 frames per second (fps). Using a face detection algorithm, only the facial region was detected, cropped, and saved for further analysis.
- **Frame Standardization**: To maintain consistency and optimize computation, face-cropped frames were resized to 112×112 pixels and normalized.

4.4.2. Feature Extraction

Feature extraction is crucial for identifying subtle visual patterns distinguishing deepfake videos from real ones. The process includes:

- **CNN-Based Feature Extraction:** We used **ResNeXt-50**, a deep Convolutional Neural Network (CNN), to extract meaningful representations from each frame.
- **2048-Dimensional Feature Vectors:** The CNN processed each image and produced a 2048-dimensional feature vector, capturing important spatial details.

- **Temporal Coherence:** Since deepfake detection involves understanding temporal inconsistencies across frames, these extracted features were further analyzed in sequence.
-

4.4.3. Model Training

Once the feature vectors were extracted, they were used to train a **hybrid deep learning model** combining **CNN (ResNext) and LSTM** for enhanced performance.

- **LSTM for Sequential Learning:** A **Long Short-Term Memory (LSTM) network** was used to process the sequential features extracted from video frames. This allows the model to learn the **temporal inconsistencies** present in deepfake videos.
- **Training Process:**

  → Model parameters were optimized using the **Adam optimizer** with a **learning rate of 1e-5 (0.00001)**.

  → **Cross-entropy loss function** was used since it is well-suited for binary classification tasks.

  → The batch size was set to **4** based on GPU memory constraints.

  → Model is trained for 20 epochs

4.4.4. Evaluation & Fine-Tuning

- **Performance Metrics:** The trained model was evaluated using:
  - **Accuracy** – Measures the overall correctness of predictions.
  - **Precision** – Determines how many predicted fake videos were actually fake.
  - **Recall** – Measures the ability to detect all fake videos correctly.
  - **F1-Score** – A harmonic mean of precision and recall to balance false positives and false negatives.
- **Hyperparameter Tuning:** Multiple experimental iterations were conducted to fine-tune the model parameters, ensuring improved convergence and minimizing overfitting.

4.4.5. User Interface

To make the deepfake detection system easily accessible, a **Django-based web application** was developed.

- **User-Friendly Interface:** The web app allows users to **upload a video file** via an interactive UI.
- **Automated Prediction Pipeline:**

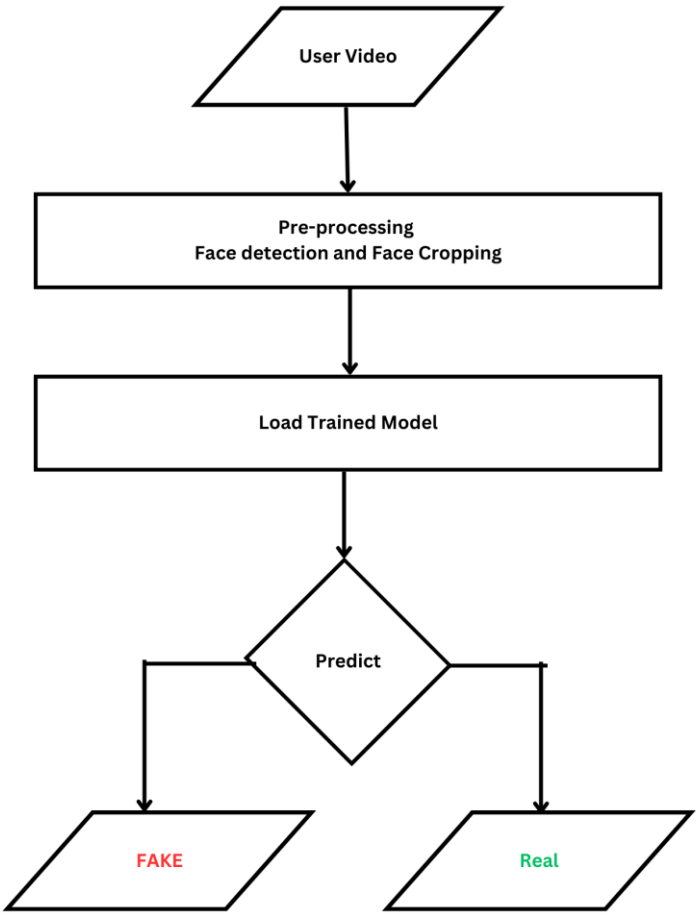

Fig 4.6 : User Input prediction workflow

→ The uploaded video is **processed in real time**, and the **face region is extracted**.

→ The pre-trained model performs feature extraction and classification.

→ The **prediction result (Real or Fake)** is displayed to the user along with the model's **confidence score**.

- **Scalability Considerations:** Django was chosen for its **scalability and robustness**, allowing future enhancements such as API integration for large-scale deployment.

4.5. Performence Measure

- 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.
- The confusion matrix shows how our Deepfake Video Detection 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.

4.6. Advantage

-The combination of **ResNeXt-50 and LSTM** provides a powerful deepfake detection framework by leveraging both spatial and temporal analysis. **ResNeXt-50**, with its grouped convolution architecture, enhances feature extraction by efficiently capturing **fine-grained facial distortions** that are characteristic of deepfake videos. Unlike traditional CNNs like ResNet or VGG, ResNeXt's **cardinality-based structure** improves feature diversity, making it more effective in detecting **subtle visual artifacts** introduced by deepfake techniques.

-**LSTM further enhances deepfake detection** by capturing **temporal inconsistencies** across video frames, a major limitation of frame-based CNN approaches. Deepfake videos often exhibit unnatural motion patterns, such as **inconsistent blinking, facial flickering, or misaligned textures**, which LSTMs can effectively learn by analyzing sequential dependencies

5. DATASET AND MODELS

5.1. Datasets

5.1.1. FaceForensics++ - [link]

FaceForensics++ is a large-scale dataset designed for training and evaluating deepfake detection models. It consists of both real and manipulated videos generated using multiple deepfake creation techniques such as DeepFakes, Face2Face, FaceSwap, and NeuralTextures. The dataset provides videos in different compression levels to simulate real-world scenarios, making it an essential resource for deepfake detection research.

Metadata

→ Total Videos: Over 1,000 original videos with corresponding manipulated versions

→ Manipulation Methods: DeepFakes (DF), Face2Face (F2F), FaceSwap (FS), NeuralTextures (NT)

5.1.2. DeepFake Detection Challenge - [link]

The Deepfake Detection Challenge (DFDC) dataset was created by Facebook AI to aid in developing deepfake detection models. It contains thousands of deepfake videos generated using various techniques, including AI-based facial manipulation and audio alterations. The dataset includes a diverse set of individuals, backgrounds, and lighting conditions to improve real-world applicability.

Metadata

→ **Manipulation Methods**: Multiple AI-based deepfake generation techniques

→ **Audio**: Some videos contain altered audio, we can .filter them for experiments focused on video-only deepfakes)

5.1.3. Celeb-DF - [link]

The **Celeb-DF** dataset is a large-scale deepfake dataset designed to improve the robustness of deepfake detection models. It consists of high-quality deepfake videos generated

using advanced deepfake synthesis techniques, making detection more challenging. Unlike earlier datasets, Celeb-DF focuses on reducing visible artifacts, improving visual realism, and mimicking real-world deepfake scenarios.

Metadata

→ **Manipulation Methods**: High-resolution deepfake synthesis, fewer visual artifacts

→ **Audio**: Most videos retain original audio, but some may have mismatched lip-sync

→ **Diversity**: Videos cover different celebrities, various lighting conditions, and multiple expressions

→ **Format**: MP4 video files with frame rates between 25-30 FPS

Fig 5.1 : Celeb-DF dataset preview

5.2. Models

Our model integrates both **Convolutional Neural Networks (CNNs)** and **Recurrent Neural Networks (RNNs)** to detect deepfake videos efficiently. We leverage a **pre-trained ResNeXt CNN** to extract frame-level features, which are then processed by an **LSTM (Long**

Short-Term Memory) network for temporal analysis. The **DataLoader** is used to feed the labeled training video splits into the model for learning and classification.

5.2.1. ResNeXt for Feature Extraction

→ To streamline feature extraction, we employ a **pre-trained ResNeXt-50 (resnext50_32x4d)** model, a residual CNN optimized for deep neural networks. This model consists of **50 layers** with a **32 × 4-dimensional architecture**, ensuring robust performance. The **2048-dimensional feature vectors** obtained from the final pooling layers of ResNeXt serve as inputs for the LSTM network.

→ This model consists of 50 layers and 32 x 4 dimensions.

stage	output	ResNeXt-50 (32×4d)
conv1	112×112	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2 [1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 256 3×3, 256, C=32 1×1, 512] ×4
conv4	14×14	[1×1, 512 3×3, 512, C=32 1×1, 1024] ×6
conv5	7×7	[1×1, 1024 3×3, 1024, C=32 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax
# params.		**25.0**×10^6

Fig 5.2: ResNext Architecture

5.2.2. LSTM for Sequential Processing

The extracted **2048-dimensional feature vectors** are passed to an **LSTM layer**, which consists of **2048 hidden units** and a **dropout rate of 0.4** to enhance generalization. The LSTM

processes frames sequentially, analyzing temporal dependencies by comparing the frame at time **t** with earlier frames at **t-n**. This approach helps the model capture frame-to-frame inconsistencies indicative of deepfake videos.

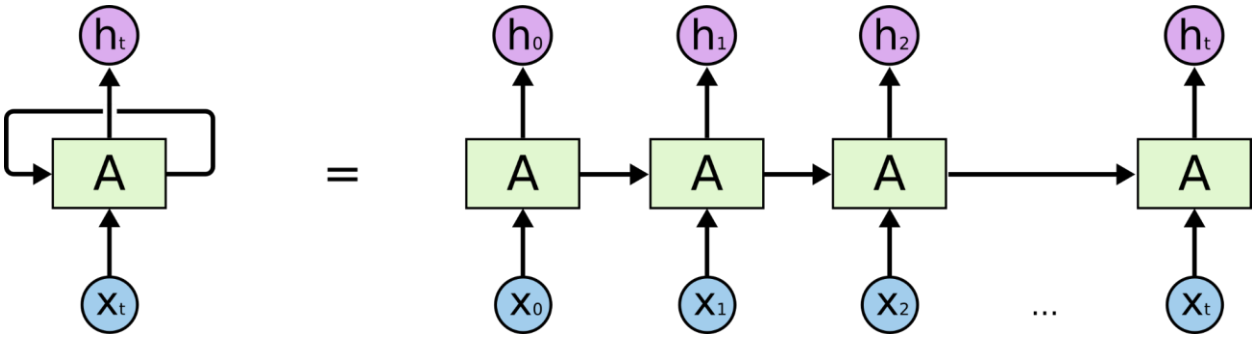

Fig. 5.3 : LSTM architecture

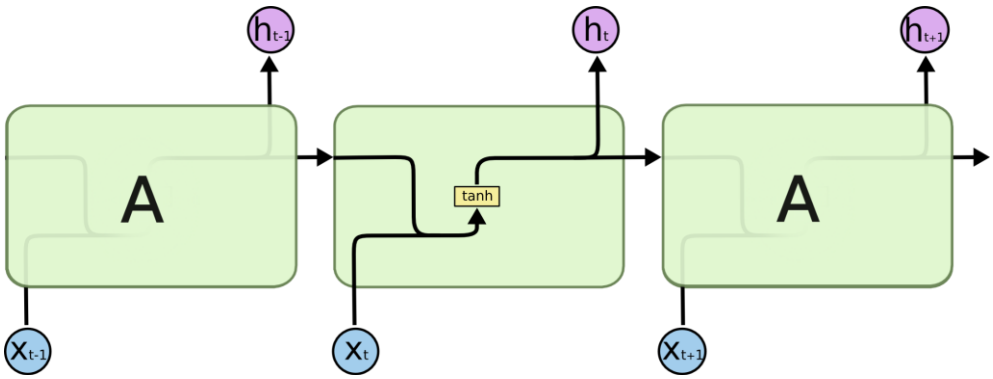

Fig. 5.4 : LSTM internal architecture

5.2.3. 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 in a ordered way. So that it can be passed to the LSTM sequentially.

5.2.4. Relu Activation Function

ReLU is a widely used activation function in neural networks that outputs 0 for any negative input and returns the input itself for positive values. This simple yet effective function closely mimics the behavior of biological neurons. Being non-linear, ReLU enables complex pattern

learning while avoiding the vanishing gradient issue commonly seen in sigmoid functions. Additionally, for larger neural networks, ReLU significantly enhances training speed, making it a preferred choice for deep learning models.

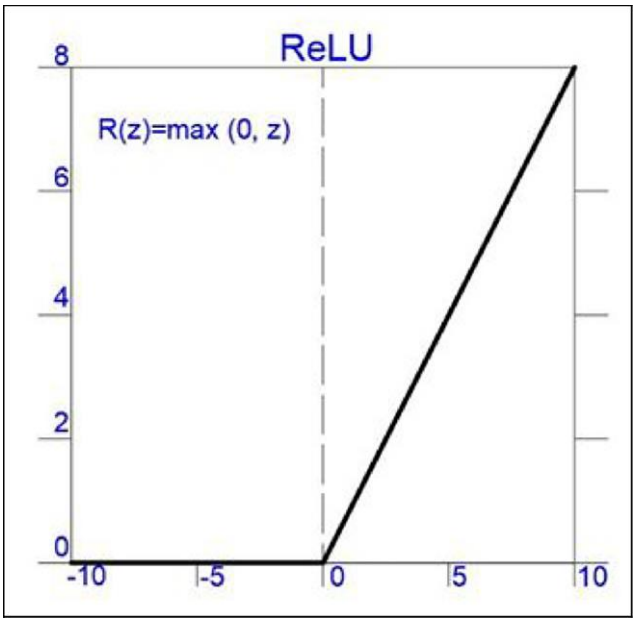

Fig. 5.5 : Relu activation function

5.2.5. SoftMax Activation Function

Softmax is an activation function primarily used in the **final layer of classification models**, especially in **multi-class classification problems**. It converts raw output logits into a **probability distribution** by assigning values between **0 and 1**, where the sum of all output values equals **1**.

Softmax helps interpret model predictions by providing a **confidence score** for each class, allowing for better decision-making in classification tasks.

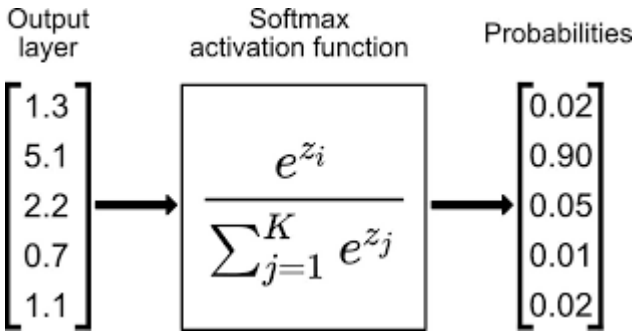

Fig. 5.6: Example of SoftMax activation function working

5.2.6. Dropout Layer[24]

A **Dropout layer** with a probability of **0.4** is used in the model to **reduce overfitting** and improve generalization. Dropout works by **randomly deactivating (setting to 0) a fraction of neurons** during training, preventing the model from relying too much on specific neurons.

By setting **40% of the neurons to 0**, the network is forced to learn **more robust features** from different neuron combinations, making it less prone to memorizing training data. This process **modifies the weight updates** during backpropagation, leading to a more **generalized and stable model** for unseen data.

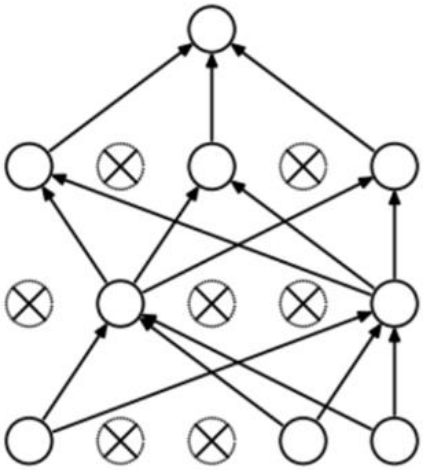

Fig. 5.7 : Dropout Layer overview

5.2.7. Adaptive Average Pooling Layer

A **2D Adaptive Average Pooling Layer** is incorporated into the model to **reduce variance, lower computational complexity, and extract low-level features** from neighboring regions. By dynamically adjusting the pooling window size, it helps retain essential spatial information while making the model more efficient and robust for feature extraction.

6. SAMPLE CODING

6.1. Preprocessing file:

6.1.1. Frame count based elimination:

```
import glob
import numpy as np
import cv2



# Define base path
base_path = 'dataset_folder_path'



# Collect all video files from different subdirectories
video_paths = [
    f'{base_path}/Celeb-DF/real/*.mp4',
    f'{base_path}/Celeb-DF/fake/*.mp4',
    f'{base_path}/DFDC/real/*.mp4',
    f'{base_path}/DFDC/fake/*.mp4',
    f'{base_path}/FF++/real/*.mp4',
    f'{base_path}/FF++/fake/*.mp4'
]



video_files = []
for path in video_paths:
    found_files = glob.glob(path)
    video_files.extend(found_files)
    print(f'Searched in: {path}, Found: {len(found_files)} files')



if not video_files:
```

```
    print("No video files found! Check your paths.")
else:
    print(f'Total videos found before filtering: {len(video_files)}')


# Filter videos and get frame counts
frame_count = []
valid_videos = []


for video_file in video_files:
    cap = cv2.VideoCapture(video_file)
    frame_num = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    cap.release()


    if frame_num >= 150:
        frame_count.append(frame_num)
        valid_videos.append(video_file)


print(f"Total valid videos (with 150+ frames): {len(valid_videos)}")
print("Frame counts:", frame_count)
print('Average frames per valid video:', np.mean(frame_count) if frame_count else 'N/A')
```

This Python script is designed to **collect and filter video files** from multiple subdirectories of a deepfake dataset, ensuring that only videos with a sufficient number of frames are processed further. It leverages the **glob** library to gather files and uses **OpenCV** to analyze frame counts.

6.1.2. Frame extraction :

```
def frame_extract(path):
    vidObj = cv2.VideoCapture(path)
    success = True
    while success:
        success, image = vidObj.read()
        if success:
            yield image
```

This Script is using OpenCV to extract the faces from the video given

6.1.3. Face detection:

```
# Process frames to extract faces and create face-only videos
def create_face_videos(path_list, out_dir):
    # Ensure output directory exists
    if not os.path.exists(out_dir):
        os.makedirs(out_dir)

    # Check already existing videos
    already_present_count = len(glob.glob(os.path.join(out_dir, '*.mp4')))
    print("No of videos already present:", already_present_count)

    for path in tqdm(path_list):
        out_path = os.path.join(out_dir, os.path.basename(path))

        # Skip if the file already exists
        if os.path.exists(out_path):
            print(f"File Already exists: {out_path}")
            continue

        # Initialize video writer
        out = cv2.VideoWriter(out_path, cv2.VideoWriter_fourcc(*'mp4v'), 30, (112,
112))
```

```
        frames = []
        frame_count = 0

        try:
            for idx, frame in enumerate(frame_extract(path)):
                if frame_count >= 150:
                    break

                frames.append(frame)
                frame_count += 1

                if len(frames) == 1:  # Process one frame at a time to reduce issues
                    faces = face_recognition.batch_face_locations(frames,
number_of_times_to_upsample=0, batch_size=1)
                    for i, face in enumerate(faces):
                        if len(face) > 0:
                            top, right, bottom, left = face[0]
                            try:
                                face_crop = cv2.resize(frames[i][top:bottom, left:right], (112,
112))
                                out.write(face_crop)
                            except Exception as e:
                                print(f"Face extraction error in {path}: {e}")

                    # Clear frames buffer
                    frames = []

        except Exception as e:
            print(f"Error processing video {path}: {e}")
```

```
        out.release()
```

This Python function processes input videos to **extract faces from frames and create face-only videos**. It ensures that only relevant facial regions are stored in the output videos, making the dataset more suitable for deepfake detection models that focus on **facial features**.

6.2. Model Training[25]:

6.2.1. Dataset Class to load labels from csv file

```
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 matplotlib.pyplot as plt
import face_recognition
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):
        video_path = self.video_names[idx]
        frames = []
        a = int(100/self.count)
        first_frame = np.random.randint(0,a)
```

```
        temp_video = video_path.split('/')[-1]
        #print(temp_video)
        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(self.frame_extract(video_path)):
          frames.append(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
#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()
```

This script defines a **PyTorch dataset class** (video_dataset) that loads and processes videos for deepfake detection. It extracts frames, applies transformations, and assigns corresponding labels for training deep learning models.

6.2.2. DataLoader class

```
# load the labels and video in data loader
import random
import pandas as pd
from sklearn.model_selection import train_test_split

header_list = ["file","label"]
labels = pd.read_csv('/content/drive/My Drive/Gobal_metadata.csv',names=header_list)
#print(labels)
train_videos = video_files[:int(0.8*len(video_files))]
valid_videos = video_files[int(0.8*len(video_files)):]
print("train : " , len(train_videos))
print("test : " , len(valid_videos))
# train_videos,valid_videos = train_test_split(data,test_size = 0.2)
# print(train_videos)

print("TRAIN: ", "Real:",number_of_real_and_fake_videos(train_videos)[0],"
Fake:",number_of_real_and_fake_videos(train_videos)[1])
print("TEST: ", "Real:",number_of_real_and_fake_videos(valid_videos)[0],"
Fake:",number_of_real_and_fake_videos(valid_videos)[1])



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)])

test_transforms = transforms.Compose([
                        transforms.ToPILImage(),
                        transforms.Resize((im_size,im_size)),
                        transforms.ToTensor(),
                        transforms.Normalize(mean,std)])
train_data = video_dataset(train_videos,labels,sequence_length = 10,transform = train_transforms)
#print(train_data)
val_data = video_dataset(valid_videos,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)
image,label = train_data[0]
im_plot(image[0,:,:,:])
```

This script loads and preprocesses **deepfake detection video datasets** by organizing labeled video data into PyTorch **DataLoaders**. It enables efficient **batch processing** for training deep learning models.

6.2.3. Model Definition[26]

```
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)))
```

This script defines a **PyTorch model** for deepfake detection, utilizing a **ResNeXt-50 CNN** for feature extraction and an **LSTM** for temporal analysis of video frames. It also supports **feature visualization** for interpretability.

6.2.4. Training and Testing loop definition[28]

```
import torch
from torch.autograd import Variable
import time
import os
import sys
import os
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)
        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()
        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
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()
            _,outputs = model(inputs)
            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)).t
olist()
            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
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
```

This script defines functions for **training (`train_epoch`) and testing (`test`)** a **CNN+LSTM deepfake detection model** in PyTorch. It includes accuracy tracking, loss calculation, and checkpoint saving.

6.2.5. Ploting 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)
```

This function **print_confusion_matrix(y_true, y_pred)** generates a confusion matrix for evaluating the model's performance in classifying Fake vs. Real videos.

6.2.6. Ploting Accuracy and Loss

```
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()
```

6.2.7. Training and evaluation

```
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-5)

#class_weights =
torch.from_numpy(np.asarray([1,15])).type(torch.FloatTensor).cuda()
#criterion = nn.CrossEntropyLoss(weight = class_weights).cuda()
criterion = nn.CrossEntropyLoss().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)
    true,pred,tl,t_acc = test(epoch,model,valid_loader,criterion)
    test_loss_avg.append(tl)
    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)
```

This script **trains** and **evaluates** a deepfake detection model using **PyTorch and scikit-learn**.

6.3. User Interface

6.3.1. Prediction Function:

```
def predict_page(request):
    if request.method == "GET":
        # Redirect to 'home' if 'file_name' is not in session
        if 'file_name' not in request.session:
            return redirect("ml_app:home")
        if 'file_name' in request.session:
            video_file = request.session['file_name']
        if 'sequence_length' in request.session:
            sequence_length = request.session['sequence_length']
        path_to_videos = [video_file]
        video_file_name = os.path.basename(video_file)
        video_file_name_only = os.path.splitext(video_file_name)[0]
        # Production environment adjustments
        if not settings.DEBUG:
            production_video_name = os.path.join('/home/app/staticfiles/',
video_file_name.split('/')[3])
            print("Production file name", production_video_name)
        else:
```

```
        production_video_name = video_file_name

    # Load validation dataset
    video_dataset = validation_dataset(path_to_videos, sequence_length=sequence_length, transform=train_transforms)

    # Load model
    if(device == "gpu"):
        model = Model(2).cuda()  # Adjust the model instantiation according to your model structure
    else:
        model = Model(2).cpu()  # Adjust the model instantiation according to your model structure
    model_name = os.path.join(settings.PROJECT_DIR, 'models', get_accurate_model(sequence_length))
    path_to_model = os.path.join(settings.PROJECT_DIR, model_name)
    model.load_state_dict(torch.load(path_to_model, map_location=torch.device('cpu')))
    model.eval()
    start_time = time.time()
    # Display preprocessing images
    print("<=== | Started Videos Splitting | ===>")
    preprocessed_images = []
    faces_cropped_images = []
    cap = cv2.VideoCapture(video_file)
    frames = []
    while cap.isOpened():
        ret, frame = cap.read()
        if ret:
            frames.append(frame)
        else:
```

```
            break
        cap.release()

        print(f"Number of frames: {len(frames)}")
        # Process each frame for preprocessing and face cropping
        padding = 40
        faces_found = 0
        for i in range(sequence_length):
            if i >= len(frames):
                break
            frame = frames[i]

            # Convert BGR to RGB
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            # Save preprocessed image
            image_name = f"{video_file_name_only}_preprocessed_{i+1}.png"
            image_path = os.path.join(settings.PROJECT_DIR, 'uploaded_images',
image_name)
            img_rgb = pImage.fromarray(rgb_frame, 'RGB')
            img_rgb.save(image_path)
            preprocessed_images.append(image_name)

            # Face detection and cropping
            face_locations = face_recognition.face_locations(rgb_frame)
            if len(face_locations) == 0:
                continue

            top, right, bottom, left = face_locations[0]
            frame_face = frame[top - padding:bottom + padding, left - padding:right +
padding]
```

```
            # Convert cropped face image to RGB and save
            rgb_face = cv2.cvtColor(frame_face, cv2.COLOR_BGR2RGB)
            img_face_rgb = pImage.fromarray(rgb_face, 'RGB')
            image_name = f"{video_file_name_only}_cropped_faces_{i+1}.png"
            image_path = os.path.join(settings.PROJECT_DIR, 'uploaded_images',
image_name)
            img_face_rgb.save(image_path)
            faces_found += 1
            faces_cropped_images.append(image_name)

        print("<=== | Videos Splitting and Face Cropping Done | ===>")
        print("--- %s seconds ---" % (time.time() - start_time))

        # No face detected
        if faces_found == 0:
            return render(request, 'predict_template_name.html', {"no_faces": True})

        # Perform prediction
        try:
            heatmap_images = []
            output = ""
            confidence = 0.0

            for i in range(len(path_to_videos)):
                print("<=== | Started Prediction | ===>")
                prediction = predict(model, video_dataset[i], './', video_file_name_only)
                confidence = round(prediction[1], 1)
                output = "REAL" if prediction[0] == 1 else "FAKE"
                print("Prediction:", prediction[0], "==", output, "Confidence:", confidence)
                print("<=== | Prediction Done | ===>")
```

```
                print("--- %s seconds ---" % (time.time() - start_time))

                # Uncomment if you want to create heat map images
                # for j in range(sequence_length):
                #     heatmap_images.append(plot_heat_map(j, model, video_dataset[i], './',
video_file_name_only))

            # Render results
            context = {
                'preprocessed_images': preprocessed_images,
                'faces_cropped_images': faces_cropped_images,
                'heatmap_images': heatmap_images,
                'original_video': production_video_name,
                'models_location': os.path.join(settings.PROJECT_DIR, 'models'),
                'output': output,
                'confidence': confidence
            }

            if settings.DEBUG:
                return render(request, predict_template_name, context)
            else:
                return render(request, predict_template_name, context)

        except Exception as e:
            print(f"Exception occurred during prediction: {e}")
            return render(request, 'cuda_full.html')
```

6.3.2. **Html template:**

```
{% extends 'base.html' %}
{%load static%}
{%block content%}
<div class="bg" >
<div class="container">

   <div class="row align-items-center justify-content-center">
      <div class="col-12 my-auto">
         <div class="logo text-center mb-3"><img src="{% static 'images/logo1.png'%}" alt="Logo" ></div>
         <div class="width-400">
            <video width="100%" controls id="videos">
               <source src="" id="video_source">
               Your browser does not support HTML5 video.
            </video>
            <form class="form" method="POST" enctype="multipart/form-data" name="video-upload" id="video-upload"
               class="text-center mt-3">
               {%csrf_token%}
               <div class="form-group">
                  <label>{{form.upload_video_file.widget}}</label>
                  {{form.upload_video_file}}
                  <!-- <input type="file" id="{{form.upload_video_file.id_for_label}}" name="{{form.upload_video_file.name}}" /> -->
                  {%if form.upload_video_file.errors%}
                  {%for each_error in form.upload_video_file.errors%}
                  <div class="alert alert-danger mt-1 {{form.upload_video_file.id_for_label}}">
                     {{each_error}}
                  </div>
```

```
                {%endfor%}
                {%endif%}
            </div>
            <div class="form-group">
                <label
for="{{form.sequence_length.id_for_label}}">{{form.sequence_length.label}}:
</label><span
                    id="slider-value"></span>
                <input type="number" hidden="hidden"
id="{{form.sequence_length.id_for_label}}"
                    name="{{form.sequence_length.name}}"></input>
                <div id='slider'></div>
                {%if form.sequence_length.errors%}
                {%for each_error in form.sequence_length.errors%}
                <div class="alert alert-danger mt-1
{{form.sequence_length.id_for_label}}">
                    {{each_error}}
                </div>
                {%endfor%}
                {%endif%}
            </div>
            <button id="videoUpload" type="submit" name="submit" class="btn
btn-success mt-3 btn-block">Upload</button>
        </form>
      </div>
    </div>
  </div>
</div>
</div>
{%endblock%}
{%block js_cripts%}
```

```
<script src="{%static 'js/script.js'%}"></script>
<script>
   $(function () {
      var sliderSequenceNumbers = [10,20,40,60,80,100];
      var slider = $("div#slider").slider({
         value: 1,
         min: 0,
         max: sliderSequenceNumbers.length-1,
         slide: function (event, ui) {

$('#{{form.sequence_length.id_for_label}}').val(sliderSequenceNumbers[ui.value]);

$('#{{form.sequence_length.id_for_label}}').val(sliderSequenceNumbers[ui.value]);
            $('#slider-value').html(sliderSequenceNumbers[ui.value]);
         }
      });

$("#{{form.sequence_length.id_for_label}}").val(sliderSequenceNumbers[$("#slide
r").slider("value")]);
      $('#slider-value').html(sliderSequenceNumbers[$("#slider").slider("value")]);
   });
</script>
{%endblock%}
```

7. OUTPUTS

7.1. Home Page

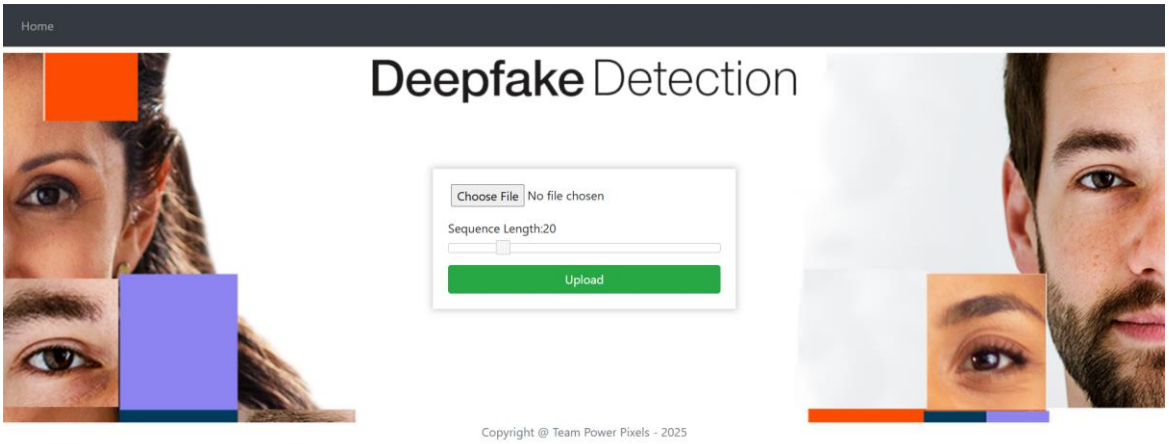

Fig. 7.1 : Home page of the UI

7.2. Real Video

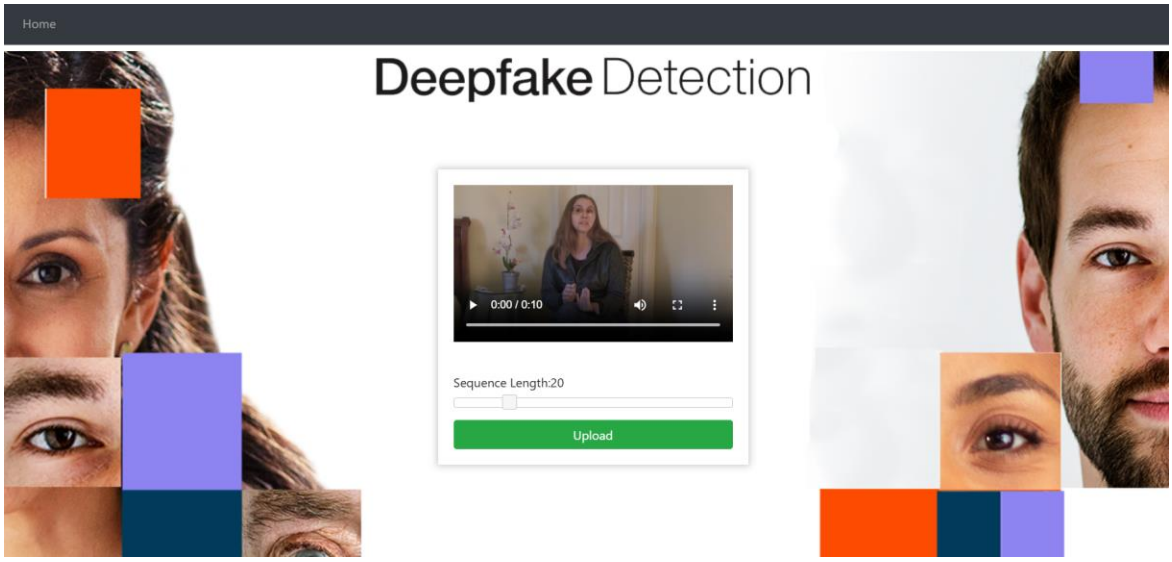

Fig. 7.2 : Uploading a real video

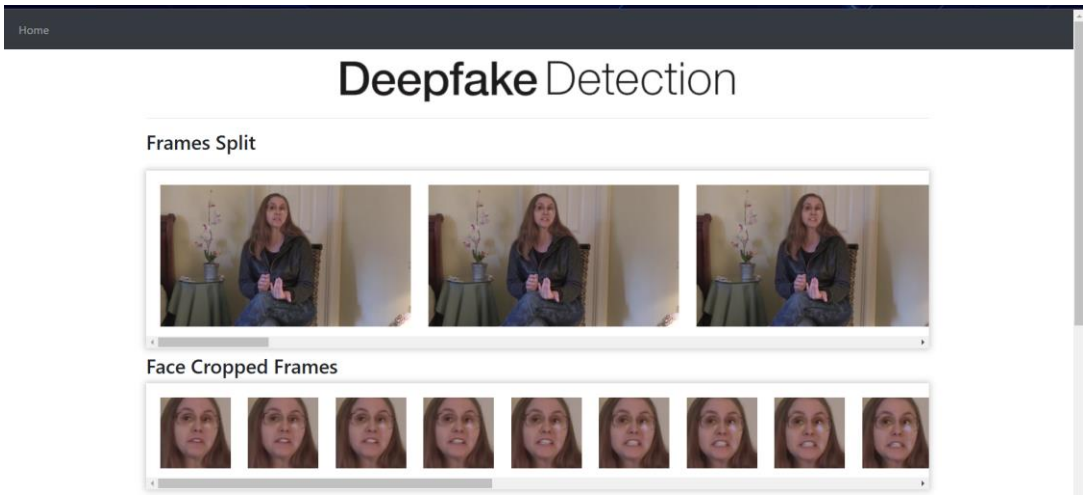

Fig. 7.3 : Frame Split and Face Extraction of Real Video

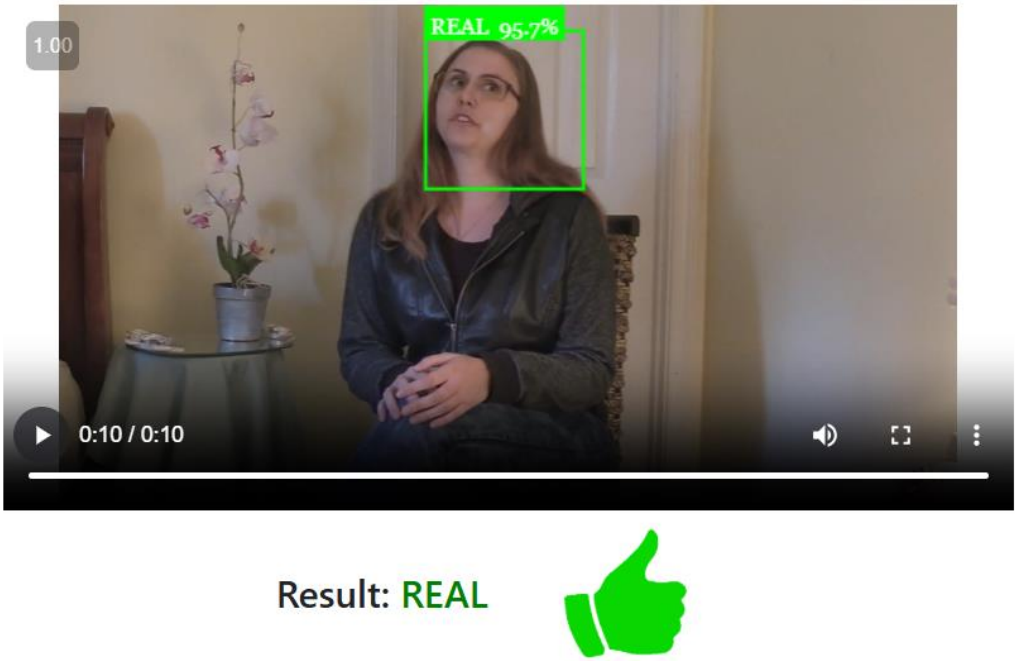

Fig. 7.4 : Real Video Prediction

```
<=== | Started Videos Splitting | ===>
Number of frames: 300
<=== | Videos Splitting and Face Cropping Done | ===>
--- 40.29127597808838 seconds ---
<=== | Started Prediction | ===>
confidence of prediction: 95.65742611885071
Prediction: 1 == REAL Confidence: 95.7
<=== | Prediction Done | ===>
--- 71.78405451774597 seconds ---
```

Fig. 7.5 : CLI Output for real audio (showing the time taken in seconds)

7.3. Fake Video

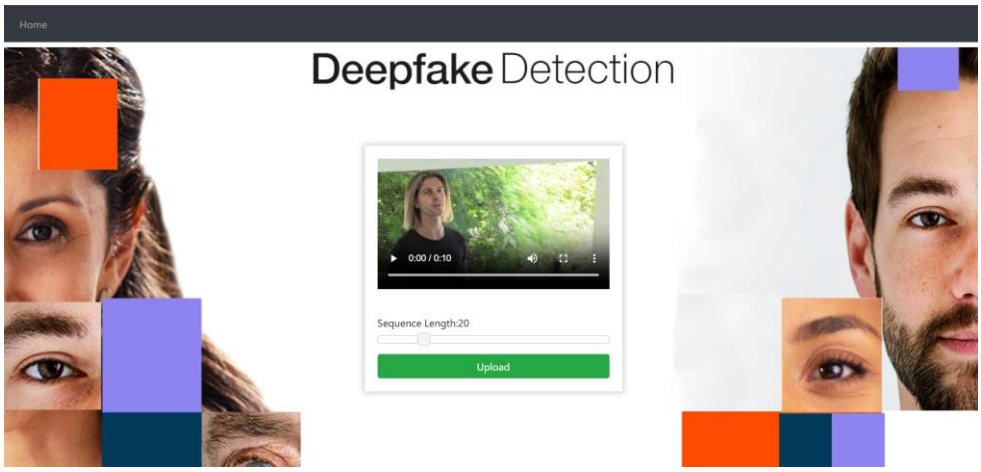

Fig. 7.6 : Uploading a fake video

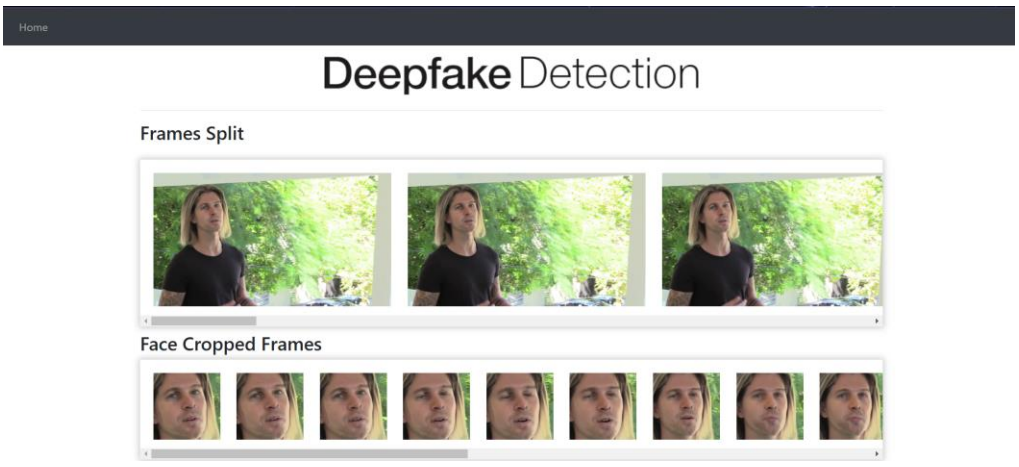

Fig. 7.7 : Frame Split and Face Extraction of Fake Video

```
<=== | Started Videos Splitting | ===>
Number of frames: 300
<=== | Videos Splitting and Face Cropping Done | ===>
--- 42.88801383972168 seconds ---
<=== | Started Prediction | ===>
confidence of prediction: 99.99874830245972
Prediction: 0 == FAKE Confidence: 100.0
<=== | Prediction Done | ===>
--- 71.73545169830322 seconds ---
```

Fig. 7.8 : CLI Output for real audio (showing the time taken in seconds)

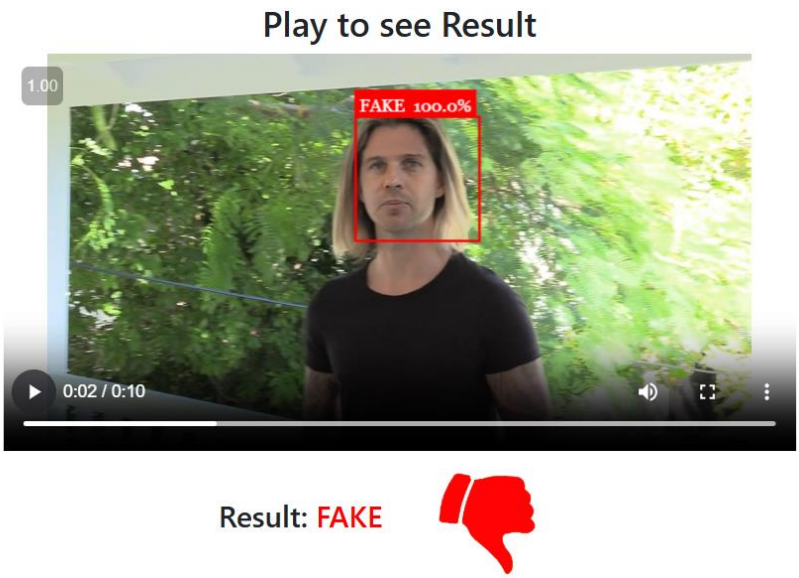

Fig. 7.8 : Fake Video Prediction

8. TESTING

Case Id	Test Case Description	Expected Result	Actual Result	Status
1	Upload a word file instead of video	Error message: Only video files allowed	Error message: Only video files allowed	Pass
2	Upload a 200MB video file	Error message: Max file size 100MB	Error message: Max file size 100MB	Pass
3	Upload a file without any faces	Error message : No faces detected in the uploaded video	Error message : No faces detected in the uploaded video	Pass
4	Videos with many faces	Fake / Real	Fake	Pass
5	Deepfake video	Fake	Fake	Pass
6	Enter /predict in URL	Redirect to /upload	Redirect to /upload	Pass
7	Press upload button without selecting video	Alert message : Please select video	Alert message : Please select video	Pass
8	Upload a real video	Real	Real	Pass
9	Upload a face-cropped real video	Real	Real	Pass
10	Upload a face cropped fake video	Fake	Fake	Pass

Table 8.1 : Test Cases and Reports

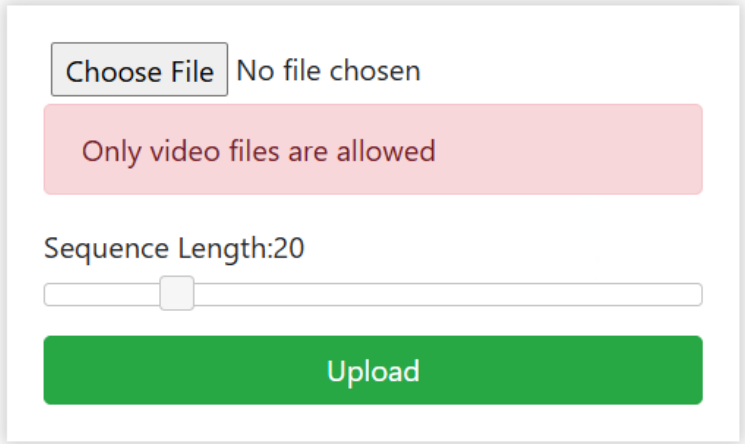

Fig. 8.1 : Test 1 (Uploading non video file)

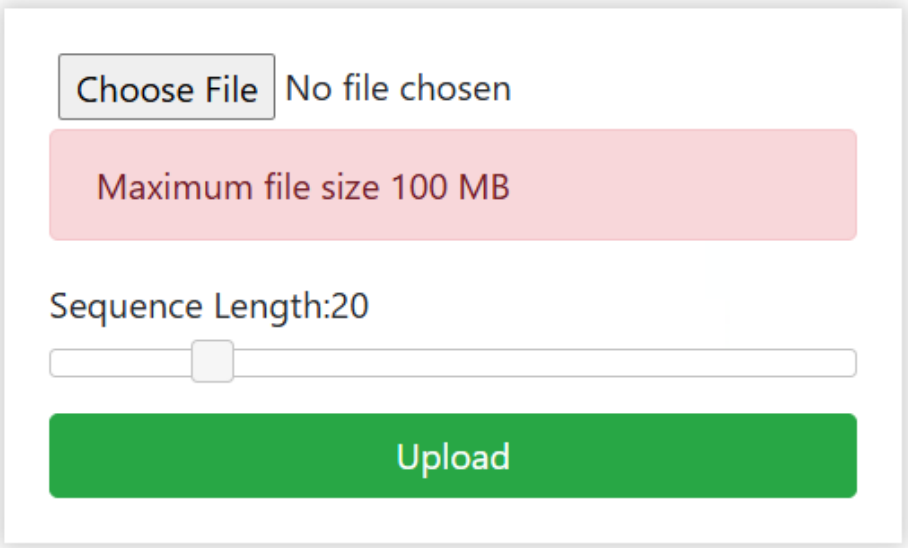

Fig. 8.2 : Test2 (Uploading a 200MB file)

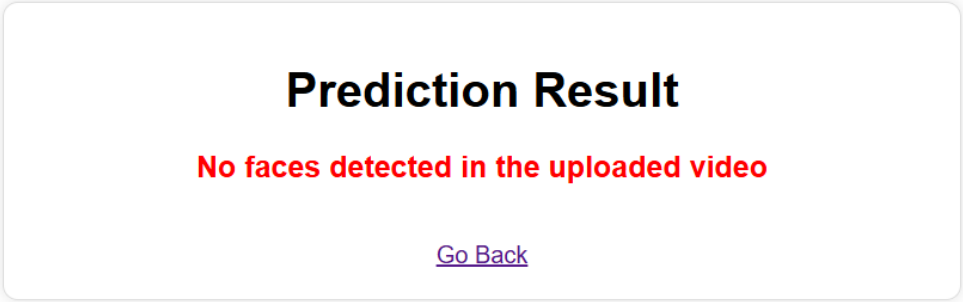

Fig. 8.3: Test 3 (Uploading a video without faces)

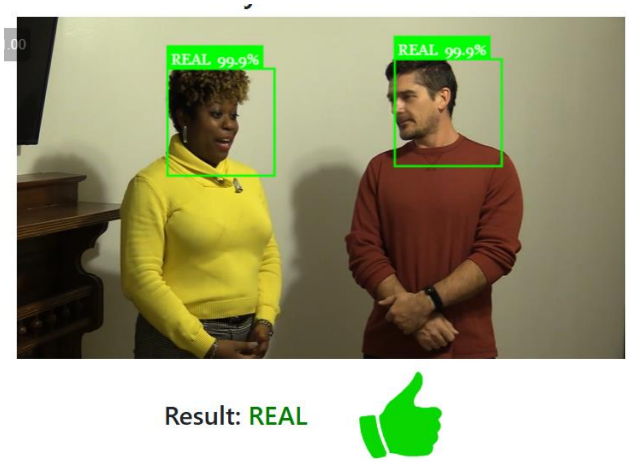

Fig. 8.4 : Test 4 (Upload video with multiple faces i.e, 2)

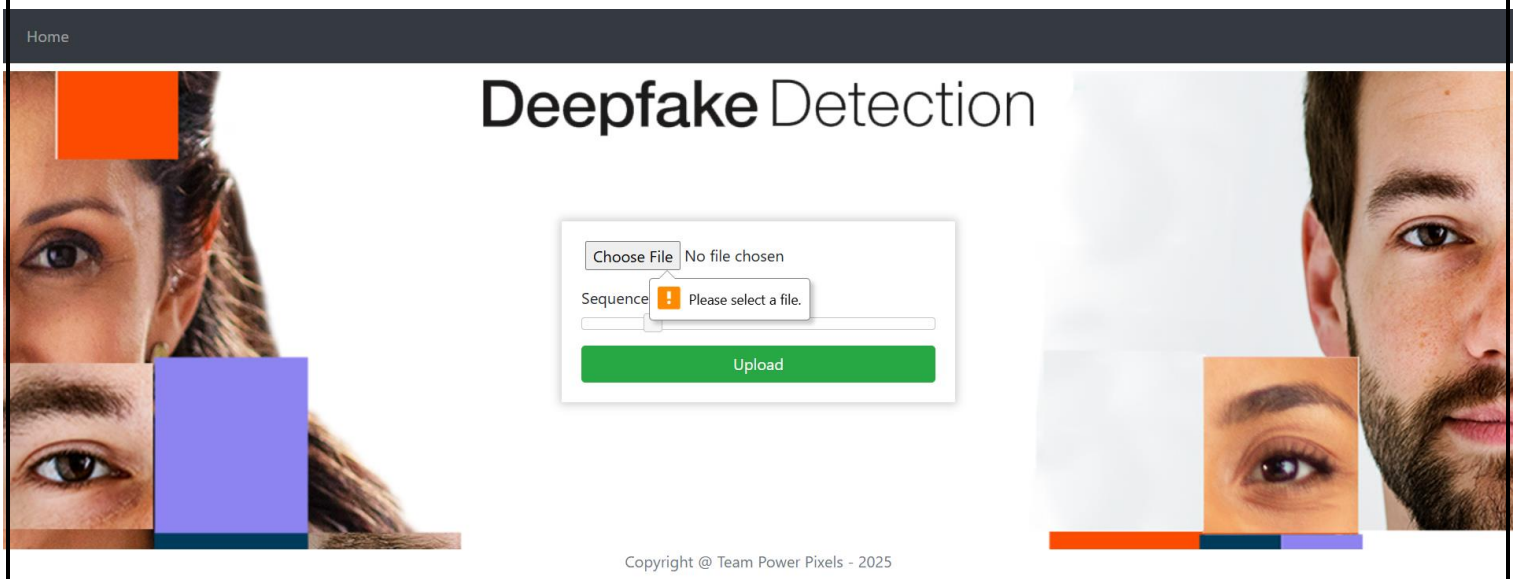

Fig. 8.7 : Test 7 (clicking on upload without a video)

9. Conclusion

In this project, we developed a deep learning-based approach to classify videos as either deepfake or real, along with the model's confidence score. Our approach leverages a combination of convolutional and recurrent neural networks, where the **ResNeXt-50** convolutional neural network (CNN) extracts frame-level features, and a **Long Short-Term Memory (LSTM)** network processes these features to capture temporal inconsistencies between consecutive frames.

With the growing sophistication of AI-generated content, deepfake detection is becoming an essential tool for **media verification, social media platforms, and cybersecurity applications**. Our method provides a reliable and scalable approach to combat the spread of misinformation through manipulated videos.

Furthermore, this study contributes to the ongoing research in **deepfake detection** by demonstrating how a **hybrid deep learning approach** can improve video classification accuracy. By combining spatial and temporal feature extraction, our model is capable of capturing both local inconsistencies in individual frames and global inconsistencies across multiple frames. This enhances the robustness of our approach compared to traditional single-frame detection techniques.

10. Future Scope

While the current system demonstrates **promising results**, several enhancements can be made to improve its performance and extend its applicability:

→ **Enhanced Video Processing** – The model currently processes frames at **10 FPS**, but increasing the frame rate or incorporating **multi-frame attention mechanisms** could further improve accuracy.

→ **Full-Body Deepfake Detection** – Our approach is currently **focused on facial deepfake detection**. Future improvements can expand the scope to detect **full-body manipulations**, including **body posture changes, gait analysis, and limb inconsistencies**.

→ **Robustness Against Adversarial Attacks** – Deepfake techniques are evolving rapidly, with adversarial attacks making fake content harder to detect. Adding **adversarial training**[28] and improving model generalization against such manipulations will enhance reliability.

→ **Real-Time Detection** – Optimizing the model for **real-time inference** on low-power devices (e.g., mobile phones, embedded systems) can allow **instant deepfake detection** in social media applications and video calls.

→ **Generalization** – Training on multiple deepfake datasets and fine-tuning the model to work across **different ethnicities, lighting conditions, and video qualities** will make it **more robust to diverse scenarios**.

→ **Integration with Web and Mobile Applications** – A **browser extension** or **mobile app** could make deepfake detection more **accessible to the general public**, allowing users to verify video authenticity before sharing.

11. User Manual

Step 1: Accessing the Code Repository

→ The complete source code is available in the repository.

→ The directory structure is as follows:

```
DeepFakeDetectionUsingResNet
    ├── Django Application
    ├── Model Creation
```

Step 2: Setting Up the Virtual Environment

Before running the project, create and activate a Python virtual environment. We recommend using ***venv***.

- Creating a Virtual Environment:

  python -m venv myenv

- Activating the Virtual Environment:

  - On Windows (CMD):

    myenv\Scripts\activate

  - On Mac/Linux:

    source myenv/bin/activate

For further details, refer to the official Python documentation on virtual environments.

Step 3: Navigating to the Django Application Directory

Change your working directory to the Django application folder using:

cd DeepFakeDetectionUsingResNet/Django Application

Step 4: Installing Required Dependencies

Run the following command to install all required packages:

pip install -r requirements.txt

Step 5: Preparing the Dataset

Place your dataset files in the *DataSetPreparation* directory before proceeding.

Step 6: Data Preprocessing

Navigate to the *Model Creation folder* and open the *preprocessing.ipynb* file. Modify file paths as necessary and execute all notebook cells to preprocess the dataset.

Step 7: Model Training

Once preprocessing is complete, open *Model_and_training.ipynb* to train the deepfake detection model. Adjust hyperparameters such as:

- Number of epochs
- Learning rate

Run the notebook until training is completed.

Step 8: Saving the Trained Model

After training, save the model in *.pt* format for later use.

Step 9: Deploying the Model in Django

Copy the saved *.pt* model file into the following directory:

DeepFakeDetectionUsingResNet/Django Application/models/

This allows the Django application to use the trained model for predictions.

Step 10: Starting the Django Server

Run the following command inside the Django Application directory:

python manage.py runserver

Step 11: Verifying Successful Setup

Upon successful execution, the server will start, and you should see an output similar to:

Starting development server at *http://127.0.0.1:8000/*

Step 12: Using the Web Application

Open the provided server URL in a web browser.

Upload a video to analyze. The system will classify the video as real or deepfake with confidence scores.

12. References

[1] G. Antipov, M. Baccouche, and J.-L. Dugelay. Face aging with conditional gen erative adversarial networks. arXiv:1702.01983, Feb. 2017

[2] Deepfake Video of Mark Zuckerberg Goes Viral on Eve of House A.I. Hearing : https://fortune.com/2019/06/12/deepfake-mark-zuckerberg/ Accessed on23 February, 2025

[3] 10 deepfake examples that terrified and amused the internet : https://www.creativebloq.com/features/deepfake-examples Accessed on23 February, 2025

[4]Deepfakes, Revenge Porn, And The Impact On Women : https://www.forbes.com/sites/chenxiwang/2019/11/01/deepfakes-revenge-porn-and the-impact-on-women/

[5]The rise of the deepfake and the threat to democracy : GHRCEM-Wagholi,Pune, Department of Computer Engineering 2019-2020 51 Deepfake Video Detection https://www.theguardian.com/technology/ng-interactive/2019/jun/22/the-rise-of the-deepfake-and-the-threat-to-democracy(Accessed on23 February, 2025)

[6] ResNext Model : https://pytorch.org/hub/pytorch_vision_resnext/ accessed on 06 April 2020

[7] Understanding LSTM Networks : https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[8] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies,Matthias Nießner, “FaceForensics++: Learning to Detect Manipulated Facial Images” in arXiv:1901.08971.

[9] Deepfake detection challenge dataset : https://www.kaggle.com/c/deepfake-detection challenge/data Accessed on23 February, 2025

[10] Yuezun Li , Xin Yang , Pu Sun , Honggang Qi and Siwei Lyu “Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics” in arXiv:1909.12962

[11] KEJUN ZHANG,YULIANG, JIANYI ZHANG, AND XINXIN LI1, “No One Can Escape: A General Approach to Detect Tampered and Generated Image” , No One Can Escape: A General Approach to Detect Tampered and Generated Image, August-September 2019, DOI: 10.1109/ACCESS.2019.2939812 https://ieeexplore.ieee.org/document/8826269

[12] Yuezun Li, Siwei Lyu, “ExposingDF Videos By Detecting Face Warping Arti facts,” in arXiv:1811.00656v3.

[13] Yuezun Li, Ming-Ching Chang and Siwei Lyu “Exposing AI Created Fake Videos by Detecting Eye Blinking” in arXiv:1806.02877v2.

[14] D. Güera and E. J. Delp, "Deepfake Video Detection Using Recurrent Neural Networks," 2018 15th IEEE International Conference on Advanced Video and Sig nal Based Surveillance (AVSS), Auckland, New Zealand, 2018, pp. 1-6.

[15]Wahidul Hasan Abir, Faria Rahman Khanam, Kazi Nabiul Alam, Myriam Hadjouni, Detecting Deepfake Images Using Deep Learning Techniques and Explainable AI Methods, Published in Intelligent Automation & Soft Computing by Tech Science Press,Volume 35, Issue 2, 2023, DOI:10.32604/iasc.2023.029653 https://www.techscience.com/iasc/v35n2/48928/html

[16] XCeption Model and Depthwise Separable Convolutions maelfabien.github.io/deeplearning/xception/

[17]A machine learning-based forensic tool for image classification - A design science approach https://www.sciencedirect.com/science/article/abs/pii/S2666281721001827
[18] J. Thies et al. Face2Face: Real-time face capture and reenactment of rgb videos. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2387–2395, June 2016. Las Vegas, NV.
[19] Face app: https://www.faceapp.com/ (Accessed on 23 February, 2025)
[203] Face Swap : https://faceswaponline.com/ (Accessed on 23 February, 2025)
[21]An Introduction to the ReLU Activation Function - https://builtin.com/machine-learning/relu-activation-function (Accessed on 23 February, 2025)
[22]Softmax Activation Function: Everything You Need to Know - https://www.pinecone.io/learn/softmax-activation/ (Accessed on 23 February, 2025)
[23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv:1412.6980, Dec. 2014.
[24] Dropout layer - http://keras.io/api/layers/regularization_layers/dropout/ (Accessed on 23 February, 2025)
[25] TensorFlow: https://www.tensorflow.org/ (Accessed on 23 Febraury, 2025)
[26] Keras: https://keras.io/ (Accessed on 23 February, 2025)
[27] PyTorch : https://pytorch.org/ (Accessed on 23 February, 2025)
[28] International Journal for Scientific Research and Development http://ijsrd.com/
https://www.kaggle.com/code/yasserh/resnext-50-implementation