Cairo University
Faculty of Computers and Artificial intelligence
Department of Computer Sciences

Verivideo

Implemented by

20200362	*Omar Mahmoud Ali*
20201014	*Esraa Osama Mohamed*
20201134	*Fatma Mahmoud Ramadan*
20201045	*Basmala Magdy Mohamed*
20201065	*Rahma Mohamed Sayed*

Supervised by
Dr. Lamia Abo Zaid
TA. Mahmoud Haddad

Graduation Project
Academic Year 2023-2024
Final Year Document

Abstract

Deepfake technology is an emerging field that combines **artificial intelligence (AI)** and **machine learning** techniques to create highly realistic manipulated media content. The rapid advancement of **deepfake** technology poses a significant threat to the integrity of multimedia content on various platforms, including social media, news outlets, and entertainment industries. In response to this challenge, VeriVideo aims to develop a web application to detect **deepfake** videos, providing an essential tool for ensuring the authenticity of digital media. This documentation introduces a **deep learning-based** method designed to effectively discern AI-generated fake videos from authentic ones. Our approach integrates a **Res-Next Convolutional Neural Network (CNN)** for extracting frame-level features, complemented by a **Long Short-Term Memory (LSTM)** based **Recurrent Neural Network (RNN)** for capturing temporal dependencies. This combination ensures the model's proficiency in identifying subtle alterations introduced during deepfake creation, such as slight changes in facial expressions, inconsistencies in lighting, and unnatural motion patterns. The model was trained and evaluated on a large, balanced dataset namely **FaceForensics++** [1]. By doing so, we aim to create a safer digital environment for users, mitigating the potential impact of manipulated media content.

Keywords:

Res-Next Convolution neural network
Recurrent Neural Network (RNN)
Long Short-Term Memory (LSTM)
Computer vision

Contents

List of Figures

Table of Abbreviations

CNN	Convolutional Neural Network	dlib	C++ toolkit for creating complex software
CUDA	Compute Unified Device Architecture	TP	True Positives
cv2	Open-Source Computer Vision Library	TN	True Negatives
EWC	Elastic Weight Consolidation	FP	False Positives
GAN	Generative Adversarial Network	FN	False Negatives
RNN	Recurrent Neural Network	Res-Next	Residual Next
LSTM	Long Short-Term Memory	ReLU	Rectified Linear Unit
OC-VAE	One-Class Variational Autoencoder	LeakyReLU	Leaky Rectified Linear Unit

Chapter 1: Introduction

The proliferation of high-resolution cameras in smartphones and the pervasive influence of social media have democratized digital video creation and dissemination, making it more accessible and impactful than ever before. Alongside these advancements, deep learning has ushered in transformative capabilities, enabling the generation of highly realistic media content through sophisticated generative models. These models can synthesize images, speech, music, and video with unprecedented fidelity, revolutionizing various sectors from entertainment to healthcare.

However, the rise of these generative models has also introduced significant challenges, particularly in the form of "deepfakes" These AI-generated videos can convincingly depict individuals saying or doing things they never did, blurring the line between reality and fabrication. Since their emergence, deepfakes have proliferated due to accessible creation tools and a growing community of developers.

1.1 Motivation

The implications of unchecked deepfake dissemination are profound. Imagine a deepfake video of a political leader declaring a war that never happened or a beloved celebrity engaging in offensive behavior. Such instances can lead to widespread misinformation, social unrest, and erosion of trust in media platforms. The potential for malicious actors to exploit deepfakes for political manipulation, defamation, or financial gain underscores the urgent need for effective detection and prevention measures.

To mitigate these risks, deepfake detection technologies are crucial. VeriVideo aims to contribute to this effort by developing a robust web application capable of discerning AI-generated fake videos from authentic ones. By leveraging advanced techniques and methodologies, we seek to empower users with tools that enhance the credibility and reliability of digital media content.

In essence, our motivation stems from the imperative to safeguard the integrity of digital media amidst the growing threat posed by deepfake technology. By developing and deploying effective detection solutions, we strive to uphold trust, combat misinformation, and foster a safer online environment for all users. Through continuous innovation and collaboration, we aim to mitigate the negative impacts of deepfakes and preserve the authenticity of information in the digital age.

1.2 Problem Definition

The primary concern is the potential for deepfakes to be exploited for malicious purposes, including but not limited to disinformation campaigns, reputation damage, identity theft, and the erosion of trust in digital media. As deepfakes become more sophisticated and accessible, it is imperative to identify and address the underlying issues to mitigate their negative impact. Effective deepfake detection is essential to protect society from the harmful consequences of manipulated visual content [2].

Deepfakes represent a significant threat due to their ability to convincingly depict individuals saying or doing things they never did. This blurring of the line between reality and fabrication has profound implications. For instance, imagine a deepfake video of a political leader declaring war or a celebrity engaging in offensive behavior. Such instances can lead to widespread misinformation, social unrest, and erosion of trust in media platforms. The potential for malicious actors to exploit deepfakes for political manipulation, defamation, or financial gain underscores the urgent need for effective detection and prevention measures [3].

The rapid advancement of deepfake technology presents significant **challenges**:

- **Detection Difficulty:** Existing deepfake detection algorithms often struggle to keep pace with the evolving sophistication of deepfake generation techniques.
- **Widespread Misinformation:** The ability of deepfakes to convincingly fabricate reality can lead to widespread misinformation and social unrest.
- **Trust Erosion:** Deepfakes contribute to the erosion of trust in digital media, making it difficult for individuals to distinguish between real and manipulated content.
- **Malicious Exploitation:** Deepfakes can be used for political manipulation, defamation, and financial gain, posing serious threats to individuals and society.

1.3 Project Objective

To tackle the challenges outlined in Section 1.2, we propose an approach aimed at developing a robust VeriVideo. The objectives of our solution include:

1- Discovering the Distorted Truth: VeriVideo aims to uncover the distorted truth behind deepfakes, ensuring that manipulated content is accurately identified and flagged.
2- Reducing Misuse and Misleading Information: By effectively detecting deepfakes, VeriVideo will help reduce abuses and the spread of misleading information on the worldwide web, protecting individuals and society at large.
3- Accurate Classification: We aim to develop a system that can accurately distinguish and classify videos as either deepfake or pristine, thereby maintaining the integrity of digital content.
4- User-Friendly System: Provide an easy-to-use system where users can upload videos or enter URLs to determine whether the content is real or fake. This will empower users to verify the authenticity of media quickly and conveniently.

1.4 Gantt Chart of Project Time Plan

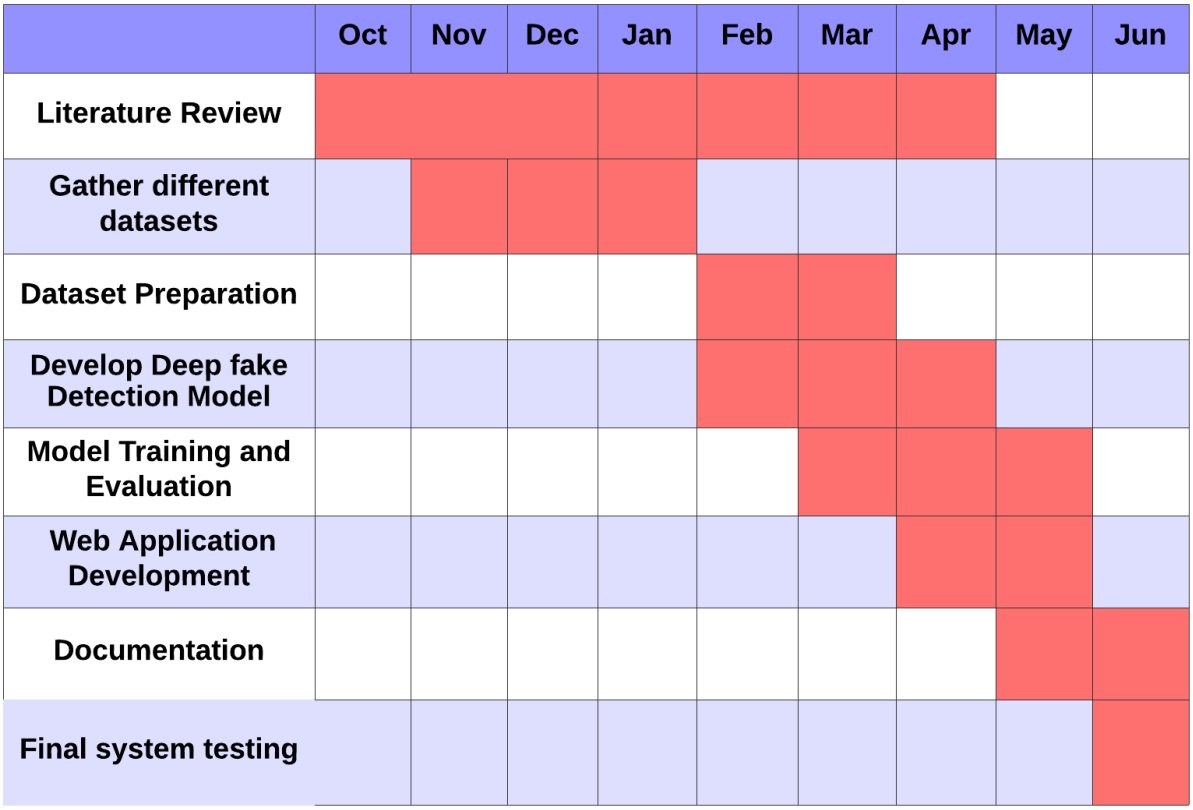

Figure 1. Gantt Chart

1.5 Project Development Methodology

VeriVideo follows a structured development methodology to ensure efficiency and effectiveness in achieving our objectives:

Requirements Gathering: This includes conducting in-depth research on deepfake detection techniques through comprehensive literature review and analysis of existing methodologies. Concurrently, we engage with stakeholders and potential users to gather detailed requirements for the web application. This collaborative approach ensures that our development efforts are closely aligned with user needs and expectations, thereby enhancing usability and maximizing adoption.

Data Preprocessing: constitutes a critical phase in our methodology. We implement rigorous checks to ensure Video Integrity, validating the completeness and correctness of each video file to filter out any corrupted or incomplete data. As part of preprocessing, we develop robust algorithms to Extract Frames from videos. These algorithms are designed not only to efficiently sample frames but also to preserve temporal sequences effectively, essential for capturing the temporal nuances characteristic of deepfake videos. Moreover, to enhance the model's Robustness and Generalization, we employ advanced Data Augmentation techniques. These techniques include random cropping, flipping, and color jittering, which augment the training dataset to expose the model to diverse variations of deepfake scenarios, thereby improving its ability to generalize to unseen data.

Model Selection and Architecture Design: involves making critical decisions to ensure the model's effectiveness in detecting deepfakes. We have selected ResNeXt-50 [4] as the backbone convolutional neural network (CNN) due to its demonstrated capability in feature extraction from images. Integrating Long Short-Term Memory (LSTM) [5] networks enable our model to perform Temporal Sequence Analysis of video frames. This integration is crucial for capturing temporal dependencies and subtle changes over time, which are indicative of deepfake manipulations. The architecture is further refined with Fully Connected Layers, strategically placed after the CNN and LSTM modules, to facilitate accurate classification based on the extracted features. This design balances computational efficiency with detection accuracy, ensuring optimal performance across various scenarios.

Training and Optimization: during this phase we meticulously fine-tune the model's parameters through Hyperparameter Tuning to optimize performance metrics such as accuracy, precision, and recall. [6] We employ Regularization Techniques such as dropout and batch normalization to prevent overfitting and enhance generalization capabilities. Additionally, we implement Gradient Clipping to stabilize training dynamics,

particularly beneficial for LSTM training, ensuring smooth convergence and improved model robustness.

Testing: We conduct thorough testing to make sure that VeriVedio solution works well across different datasets and scenarios. This phase ensures that the model meets specific performance goals and can accurately tell apart real videos from manipulated ones.

Integration and Delivery: In this phase, we focus on integrating the trained model into a user-friendly web application. This involves developing a seamless interface for end-users to upload and analyze videos, ensuring the system's scalability and responsiveness. The delivery phase includes extensive usability testing to ensure the application provides a satisfactory user experience and operates effectively in real-world conditions.

By adhering to this structured and comprehensive methodology, we aim to deliver VeriVideo, a robust and adaptive deepfake detection solution. VeriVideo not only meets current needs but also evolves to address future challenges in digital media authenticity and security. Each phase of our development process is meticulously planned and executed, incorporating advanced technologies and best practices in deep learning and cybersecurity to achieve our project goals effectively.

1.6 The used tools in the project (SW and HW)

In VeriVideo, we utilize a range of software (SW) and hardware (HW) tools to support the development, training, and deployment of our deepfake detection system. These tools are carefully selected to ensure optimal performance, scalability, and efficiency throughout the project lifecycle.

1.6.1 Software Tools (SW)

In this project, we utilized a variety of software and hardware tools to develop and implement our deepfake detection solution effectively:

Web Development:

Front-end:

- Development Environment: VSCode
- Video Source: LipSynthesis channel for slider videos

- Technologies: HTML, CSS, JavaScript
- Bootstrap Code: cdnjs website for importing Bootstrap code

Back-end:

- Flask: A lightweight web framework used for deploying the machine learning model as an API.

-API: Endpoint for uploading videos

Version Control: GitHub for both frontend and the backend

Machine Learning:

- Python: Python served as the primary programming language for its versatility in machine learning and computer vision tasks [7].

-PyTorch: A deep learning framework, provided the foundational libraries and modules for constructing and training neural networks [8].

-Google Colab: a cloud-based platform with free GPU access, was instrumental in executing computationally intensive tasks such as model training and preprocessing of video datasets [9].

-OpenCV: was utilized for video processing tasks, including frame extraction and manipulation, essential for preparing the dataset [10].

-Face Recognition: Libraries such as face_recognition [11] and dlib [11] were employed for detecting and locating faces within video frames, a crucial preprocessing step to isolate facial regions for subsequent analysis.

-Matplotlib and NumPy: Matplotlib [13] aided in visualizing data and model outputs, while NumPy [14] facilitated efficient numerical computations and array operations.

1.6.2 Hardware (HW):

- GPUs (Graphics Processing Units): Accessed via Google Colab for accelerated training of deep learning models. Provides high-performance GPUs, such as NVIDIA Tesla K80s, T4s, P100s, and V100s, depending on availability.

- Cloud Storage: Google Drive for storing large datasets and model checkpoints, ensuring seamless integration with the development environment.

By leveraging these software and hardware tools, we ensured a comprehensive and efficient development process, from initial design to deployment and maintenance. Each tool played a crucial role in addressing specific aspects of the project, contributing to the overall success of VeriVideo.

1.7 Report Organization

The rest of this report discusses the different phases that describe and illustrate the characteristics of the project.

Chapter Two: Related Work

Overview of Related Work: Surveys existing literature and projects in deepfake detection, highlighting methodologies, algorithms, and technological advancements.

Comparative Analysis: Compares the project with existing deepfake detection systems, emphasizing novel contributions and improvements in detection accuracy and efficiency.

Chapter Three: System Analysis

Functional Requirements: Details the specific functionalities of VeriVideo, including real-time detection, robustness to various manipulation techniques, and integration capabilities with existing platforms.

Non-functional Requirements: Specifies performance metrics such as detection accuracy, speed, scalability, and reliability under different environmental conditions for VeriVideo.

Use Case Diagrams: Illustrates the interactions between system components and stakeholders, outlining key use case scenarios for VeriVideo's operation and management.

Chapter Four: System Design

This chapter outlines the architecture of our VeriVideo, which combines state-of-the-art machine learning techniques to identify deepfakes effectively.

Chapter Five: Implementation and Testing

System Implementation: Details the implementation process of VeriVideo, including software development frameworks, integration of machine learning models, and deployment strategies on target hardware.

Testing: Reports on the testing methodologies employed, including performance evaluation metrics, dataset characteristics, and validation results against ground truth labels to assess detection accuracy and robustness.

System Evaluation: Analyzes the performance and effectiveness of the implemented system, discussing strengths, limitations, and areas for future improvement based on experimental results and user feedback.

Chapter 2: Related work

The emergence of deepfake videos as a significant threat to online security and privacy has spurred considerable research into developing methods for detecting them. This document reviews some of the most relevant work in this field, combining insights from various studies and highlighting the methodologies, advancements, and differences between current research and our project.

2.1 The Existing Solutions:

Thanh Thi Nguyen's Paper [7]: A Comprehensive Exploration of **Deepfake** Technology
Overview: Thanh Thi Nguyen's paper provides a thorough examination of deepfake technology, highlighting the dual-edged nature of **deep learning** advancements. While deep learning has propelled various fields forward, it has also brought significant risks to **privacy**, **democracy**, and **national security**. **Deepfakes**, powered by sophisticated algorithms, create highly realistic fake images and videos that are nearly indistinguishable from genuine ones.

Focus: The paper surveys the **algorithms** used in deepfake creation and evaluates existing detection **methods**. Nguyen discusses the **challenges**, **research trends**, and future directions in deepfake technologies. By reviewing the background and analyzing state-of-the-art detection methods, the study offers a comprehensive overview of deepfake techniques.

Objective: The primary goal is to facilitate the development of robust methods to address the increasing sophistication and prevalence of **deepfakes**. Understanding the current landscape and advancements in detection allows researchers and practitioners to tackle the growing challenges posed by deepfake technology.

Edward J. Delp and David Guera's Paper [8]: Detecting **Deepfake Videos** Using **CNNs** and **LSTMs**
Overview: This paper addresses the rising threat of **deepfake videos**, particularly in **facial manipulations**, amidst increased digitalization and AI adoption. The study uses the **HOHA dataset**, which contains 300 videos, to develop a novel deep learning model.

Proposed Solution: The solution synergizes Convolutional Neural Networks (**CNNs**) for feature extraction with Long Short-Term Memory (**LSTM**) Recurrent Neural Networks for classification. This approach achieves an impressive accuracy of **92.49%,** demonstrating its effectiveness in distinguishing manipulated videos from authentic ones.

Hasam Khalid and Simon S. Woo's Paper [9]: Deepfake Classification Using **One-Class Variational Autoencoder (OC-VAE)**
Overview: This paper focuses on deepfake classification with a **One-Class Variational Autoencoder (OC-VAE)**, introducing two variants: **OC-FakeDect-1** and **OC-FakeDect-2.** These models are trained exclusively on real images from **FaceForensics**++ [1].

Performance: **OC-FakeDect** surpasses the baseline **OC-AE** in **precision**, **recall**, and **F1** scores across diverse deepfake datasets. **OC-FakeDect-2** achieves the highest performance, showcasing its superiority. The VAE-based approach excels in learning probability distribution parameters, enhancing its capability to detect various types of fake images.

Darius Afchar et al.'s Paper [10]: Detecting Manipulated Content with **Meso-4** and **MesoInception-4**
Overview: This paper explores deep learning techniques for detecting **manipulated content** created through **Deepfake** and **Face2Face** techniques. **Two network architectures, Meso-4 and MesoInception-4,** are used for microscopic level analysis.

Network Architectures: The **Meso-4** network comprises **four** layers of **convolution** and **sequential pooling**, complemented by a dense network with a single hidden layer using **ReLU** activation for efficient generalization. **MesoInception-4** includes a different version of the initial unit for its first two **convolutional** layers.

Results: **MesoInception-4** achieves a high accuracy of **98%** on Deepfake and **95%** on **Face2Face** datasets. The study underscores the importance of focusing on the eyes and mouth regions in determining the content of manipulated faces.

Ruben Tolosana et al.'s Paper: Evaluating Deepfake Detection Accuracy Across Datasets

Overview: This study extensively evaluates DeepFake detection accuracy across various datasets, including 1st and 2nd generation DeepFakes.

Performance: In 1st generation datasets, **Xception**, Capsule Network, and **DSP-FWA** achieve near-perfect accuracy, especially in UADFV, although FaceForensics++ [1] presents challenges with a higher Equal Error Rate. In 2nd generation datasets, Celeb-DF v2 demonstrates strong performance (**97.90% AUC**), with the Eyes region consistently yielding superior results.

Challenges: **The DFDC Preview dataset** poses challenges, resulting in lower **AUC** values (**88.85%**). Fusion techniques at both the system and facial-region levels further improve accuracy, achieving state-of-the-art results.

Yuezun Li, Ming-Ching Chang, and Siwei Lyu's Paper [11]: **Deepfake** Detection Using **Physiological Signals**

Overview: This paper discusses the ease of **creating, editing, and propagating digital videos** due to advancements in camera technology, widespread cellphone usage, and the popularity of social networks and video-sharing platforms. However, this convenience has also led to the manipulation of videos to spread falsified information.

Deepfake Technology: Deepfake utilizes generative adversarial networks (**GANs**) to replace human faces in original videos with synthesized faces, resulting in seamless splicing and potential identity falsification.

Challenges: Traditional forensic methods based on signal-level, physical-level, or semantic-level cues face challenges in detecting AI-generated fake face videos.

Proposed Method: The paper introduces a novel forensic method focusing on detecting the absence of physiological signals intrinsic to humans, such as eye blinking. The proposed method employs a deep learning model that combines a convolutional neural network (**CNN**) with a recursive neural network (**RNN**) to capture the **phenomenological and temporal regularities** of **eye blinking.**

2.2 Comparing Our Presented Approach to Previous Work:

Our approach stands out from existing methodologies through its integration of innovative elements designed to significantly advance the effectiveness and adaptability of deepfake detection systems. Central to our strategy is the proactive collection of a new dataset, meticulously curated based on direct user feedback and insights derived from continuous model refinement. This iterative process ensures that our detection capabilities evolve alongside emerging deepfake techniques, thereby enhancing our system's resilience against evolving threats.

We have selected 40 sequences from our dataset.These sequences, chosen to represent a diverse range of deepfake scenarios and complexities, serve as critical benchmarks for evaluating our model's performance across various temporal and contextual nuances. We presented a neural network-based approach to classify videos as deepfake or real, along with the confidence of our proposed model. The implementation utilizes a pre-trained ResNeXt CNN model to extract frame-level features and LSTM for temporal sequence processing to detect changes between consecutive frames. Our model effectively processes video sequences with a frame sequence length of 40, chosen to balance computational efficiency and temporal context sensitivity. By focusing on sequence-level validation, we aim to reinforce the robustness and reliability of our detection framework under real-world conditions

Moreover, to cater to the dynamic nature of digital media consumption, we have implemented seamless integration with YouTube video links. This feature empowers users to conveniently submit media content for analysis without the constraints of traditional file uploads, thereby enhancing accessibility and user engagement. This innovation not only simplifies the user experience but also broadens the application scope of our detection system across different online platforms and content formats.

In summary, our approach not only introduces novel methodologies and dataset advancements but also underscores our commitment to user-centric design and continuous improvement. By leveraging cutting-edge techniques and embracing user feedback, we aim to establish VeriVideo as a leading solution in safeguarding digital media integrity against the growing threat of deepfake technology.

Chapter 3: System Analysis

3.1 Project Specification

3.1.1 Functional Requirements

We divide functional requirements into two categories: functional requirements of the deepfake model and functional requirements of the web application that will use the model.

Functional Requirements of the Model:

Users of the application will be able to detect whether the uploaded video is fake or real, along with the model's confidence in the classification. Users can determine the authenticity of uploaded videos and assess the model's classification confidence. They can also provide feedback to facilitate ongoing learning and ensure the model remains up to date with evolving deepfake techniques and variations.

The following parameters have been identified to aid in detection:

- Blinking of eyes
- Teeth enhancement
- Bigger distance between eyes
- Mustaches
- Double edges (eyes, ears, nose)
- Iris segmentation
- Wrinkles on face
- Inconsistent head pose
- Face angle
- Skin tone
- Facial expressions
- Lighting
- Different pose
- Double chins
- Hairstyle
- Higher cheekbones

Dataset Used: The model is trained on the FaceForensics++ dataset [1].

Functional Requirements of the Application:

Video Upload: Users can upload videos through a user-friendly interface that supports the MP4 video format. Additionally, users are allowed to enter a YouTube URL for processing, making it convenient to submit either a local video file or a YouTube video link for analysis.

Result Display: The application presents users with a clear and concise result indicating whether the video is real or fake. This result includes a confidence score to show the probability of the video's authenticity, providing users with an easy-to-understand assessment of the video's integrity.

User Feedback: The application enables users to provide feedback on the analysis results to improve model accuracy over time. This feature allows the system to learn from user inputs and continuously enhance its detection capabilities based on real-world feedback.

3.1.2 Non-Functional Requirements

- **Easy and User-friendly:** Users seem to prefer a more simplified process of Deep Fake video detection. Hence, a straightforward and user-friendly interface is implemented, The UI contains an input field to enter video URL or upload it for processing. It reduces the complications and at the same time enriches the user experience.

- **Cross-platform compatibility:** with an ever-increasing target market, accessibility should be your main priority. By enabling a cross-platform compatibility feature, you can increase your reach to across different platforms. Being a server-side application, it will run on any device that has a web browser installed in it, this approach eliminates the need for users to download and install platform-specific software, reducing barriers to entry and making the application more user-friendly.

- **Reliable:** Reliability is achieved through deploying and using the model from the cloud, which allows for scalable and robust infrastructure. Additionally, our model has an accuracy rate of 95%, which provides a high level of confidence in the detection results.

3.2 Use Case Diagram

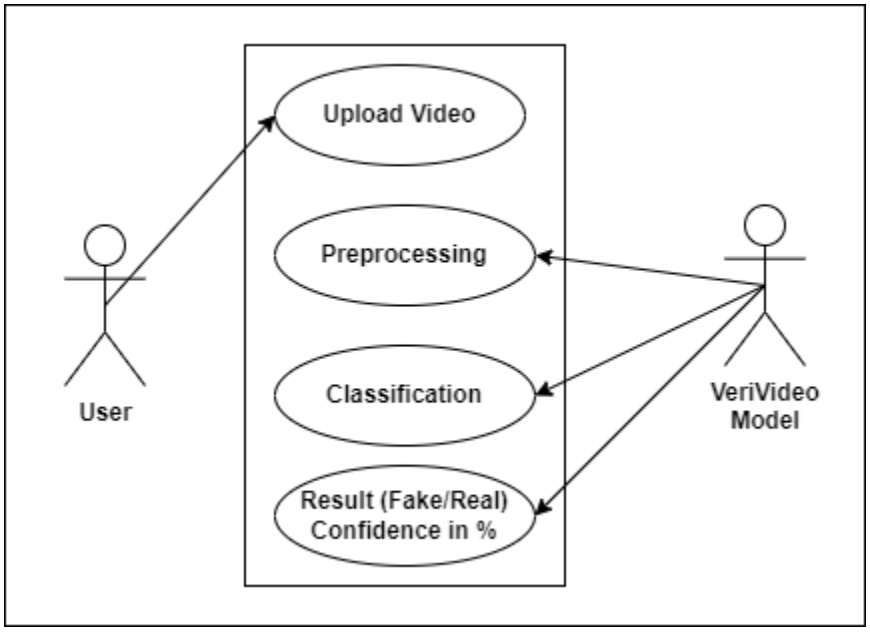

Figure 2. Use case diagram

Chapter 4: System Design

This chapter outlines the architecture of VeriVideo, which combines state-of-the-art machine learning techniques to identify deepfakes effectively. Leveraging a Long Short-Term Memory (LSTM) [5] model with Elastic Weight Consolidation (EWC) for continual learning, the system adapts to evolving deepfake technologies without losing previous knowledge.

VeriVideo features a web application, allowing users to easily detect deepfakes by uploading videos or providing URLs. Video frames are processed using OpenCV, and features are extracted with a ResNeXt [4] convolutional network. The LSTM model analyzes these features, while EWC ensures the model retains its accuracy over time, even with new data.

This chapter presents an overview of the design behind our solution

4.1 Structure Diagram:

4.1.1 Training Flow:

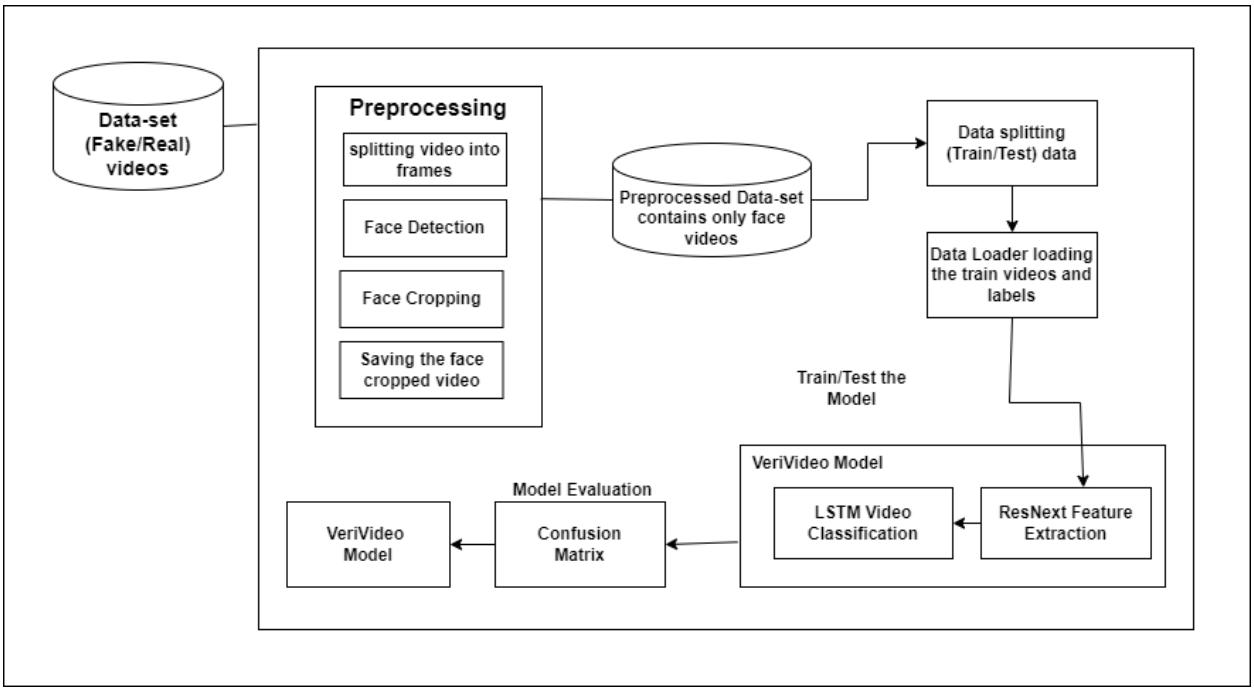

Figure 3. training flow

4.1.2 Classification Flowchart:

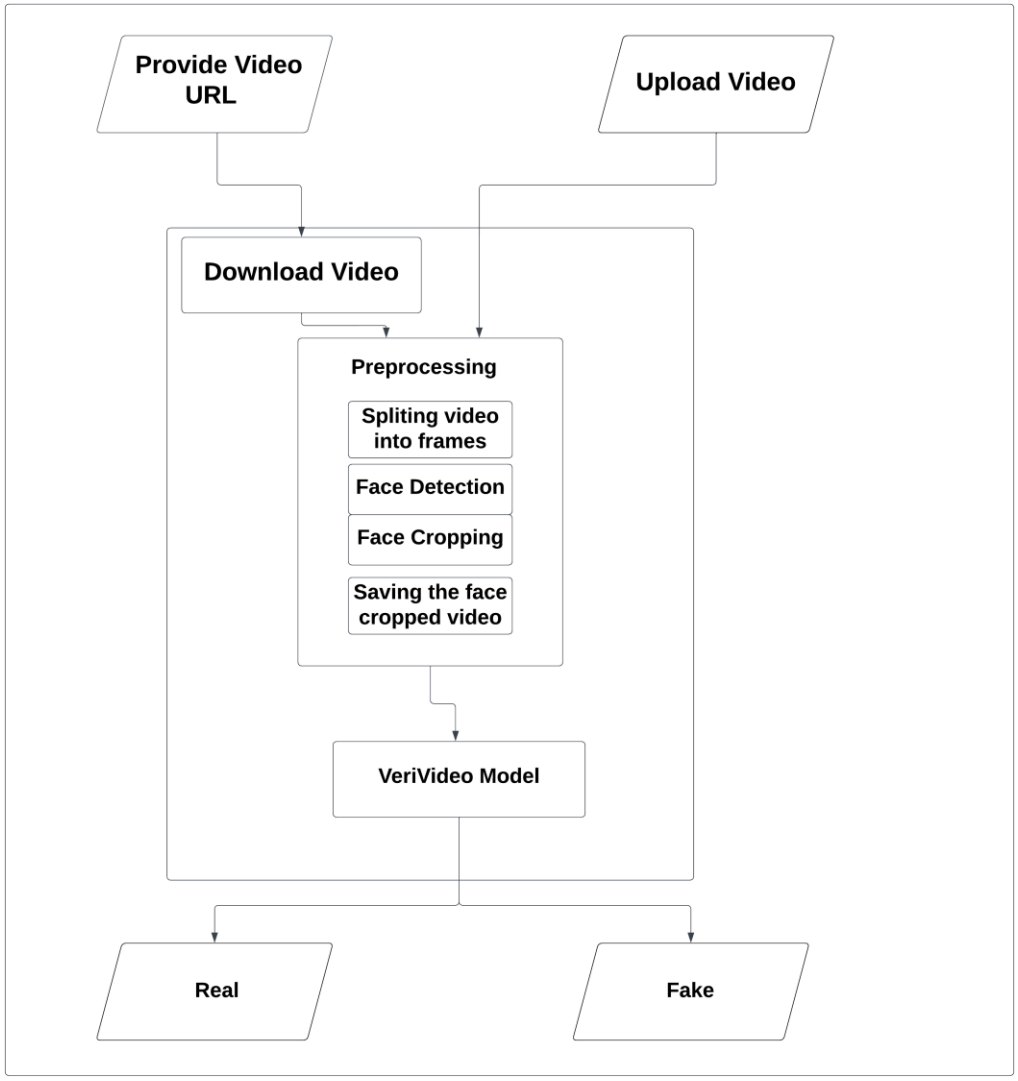

Figure 4. Classification flowchart

4.2 Sequence Diagram:

4.2.1 Upload Video Scenario:

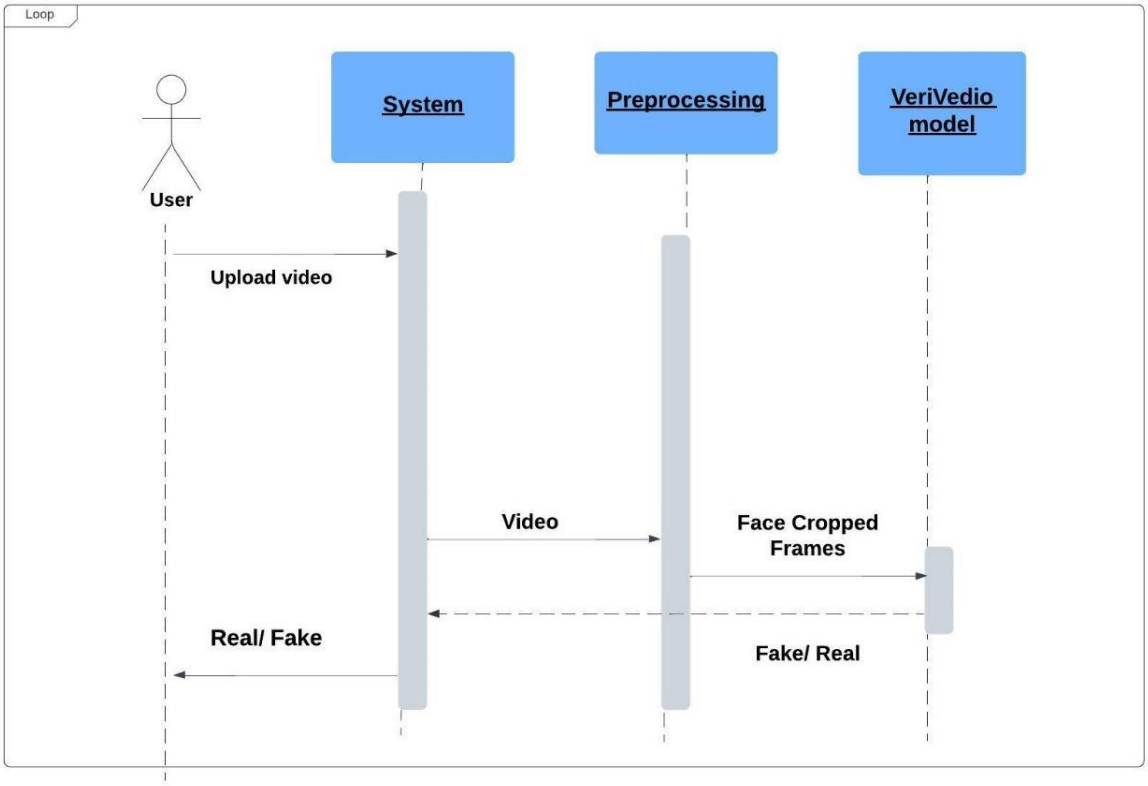

Figure 5. sequence diagram upload video scenario

4.2.2 Provide URL Scenario:

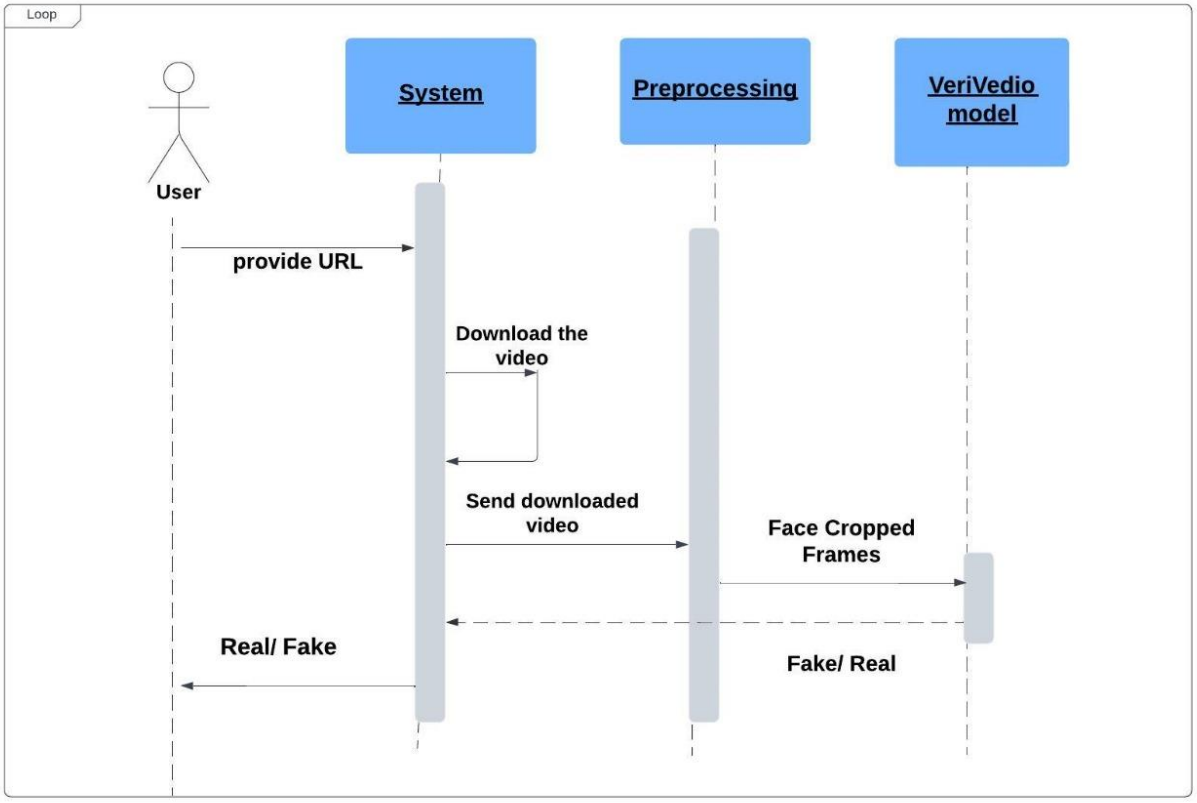

Figure 6. sequence diagram provides URL scenario

4.3 System GUI Design

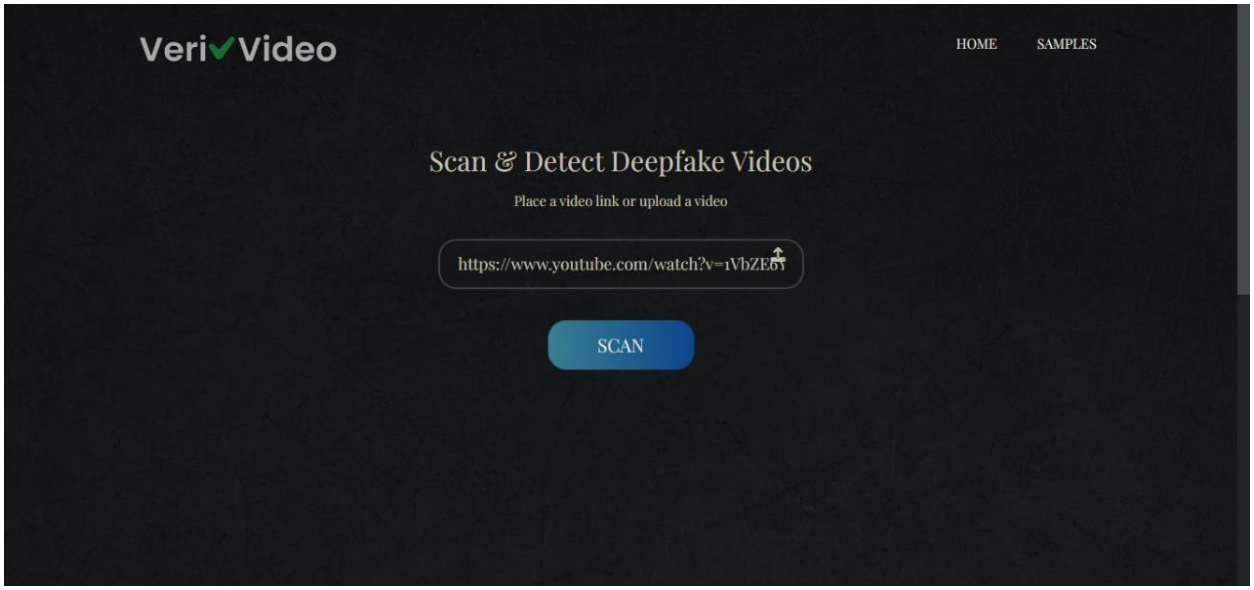

Figure 7. System GUI input view

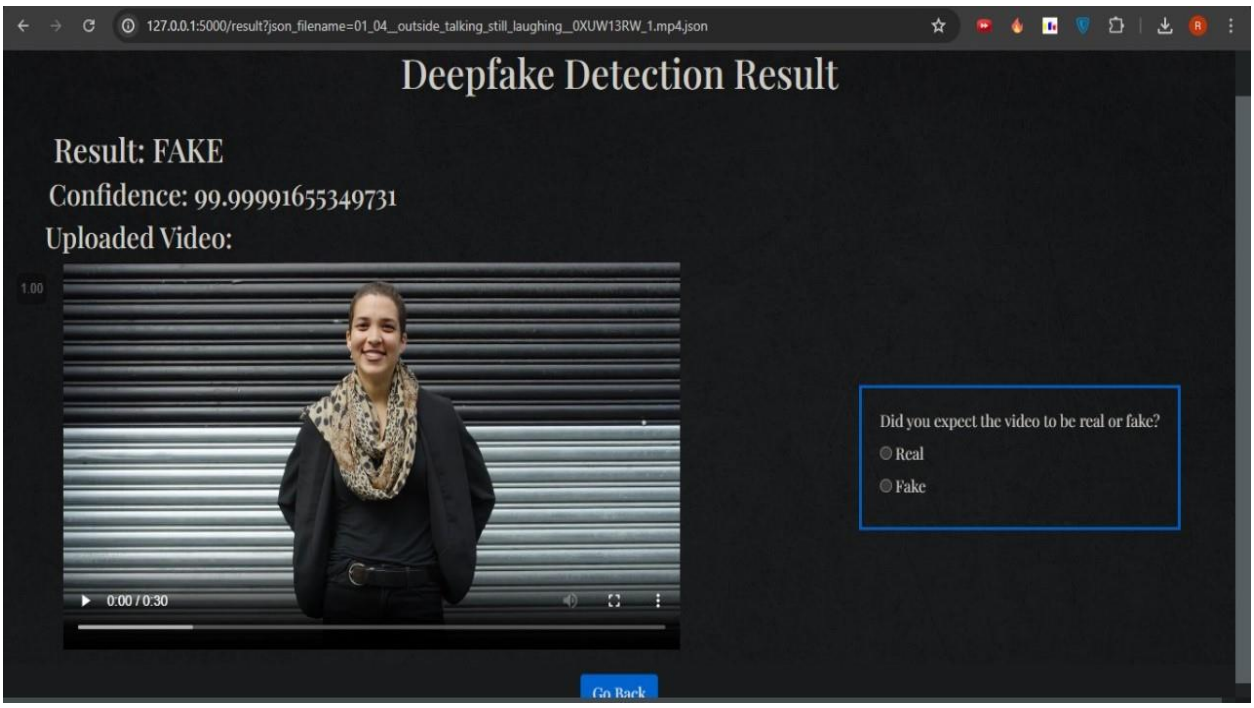

Figure 8. system GUI classification view

Chapter 5: Implementation and Testing

Our project progresses through several critical implementation steps. These steps include preprocessing the video data, ensuring compatibility with CUDA for efficient processing, and integrating essential libraries like dlib for optimal face detection and recognition. Each step is meticulously designed to lay a robust foundation for our VeriVideo model's development and training

5.1 Dataset Details

For our project, we utilized the **FaceForensics++ (FF)** [1] dataset as the main dataset for training and evaluation. The FaceForensics++ dataset is a widely used benchmark specifically designed for deepfake detection research. It consists of a large collection of real videos and deepfake videos generated using various techniques, including **face swapping** and **facial reenactment**. This dataset encompasses different subjects, lighting conditions, and backgrounds, providing a diverse range of scenarios for training and testing our VeriVideo model. We selected 1000 real and 1000 fake videos from the FaceForensics++ dataset, aiming to enhance the efficiency and accuracy of our model for real-time deepfake detection. By using this comprehensive dataset, we ensure our model is well-trained to handle various video manipulation techniques in diverse settings.

5.2 Preprocessing Details:

Preprocessing Steps:

Step 1: Import all videos in the directory into a Python list using glob.

```
def get_video_files(path):
  video_files =  glob.glob(path+'*.mp4')

  # print(video_files)
```

Figure 9. code for get all videos

Step 2: Read the videos using cv2.VideoCapture and calculate the mean number of frames per video.

```
frame_count = []
for video_file in video_files:
  cap = cv2.VideoCapture(video_file)
  if(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))<150):
    video_files.remove(video_file)
    continue
  frame_count.append(int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
print("frames" , frame_count)
print("Total number of videos: " , len(frame_count))
print('Average frame per video:',np.mean(frame_count))

return video_files
```

Figure 10. code for calc mean frames per video

Step 3: Based on the mean number of frames, decide on 150 frames as the ideal value to maintain uniformity in the new dataset.

Step 4: Split each video into individual frames.

```
# to extract frame
def frame_extract(path):
  vidObj = cv2.VideoCapture(path)
  success = 1
  while success:
      success, image = vidObj.read()
      if success:
          yield image
```

Figure 11. code for extracting frames

Step 5: Crop the frames to the face location.

Step 6: Write the cropped face frames to a new video.

Step 7: Set the video parameters:

- 30 frames per second
- 112 x 112-pixel resolution

- MP4 format

```
out = cv2.VideoWriter(out_path,cv2.VideoWriter_fourcc('M','J','P','G'), 30, (112,112))
```

Figure 12. code for set video parameters

Step 8: Instead of selecting random videos, write the first 150 frames to the new video to make better use of LSTM for temporal sequence analysis.

```
for idx,frame in enumerate(frame_extract(path)):
  if(idx <= 150):
    frames.append(frame)
    if(len(frames) == 4):
      faces = face_recognition.batch_face_locations(frames)
      for i,face in enumerate(faces):
        if(len(face) != 0):
          top,right,bottom,left = face[0]
        try:
          out.write(cv2.resize(frames[i][top:bottom,left:right,:],(112,112)))
        except:
          pass
      frames = []
```

Figure 13. code for select first 150 frames

Importing Libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import os
import glob
import json
import cv2
import copy
import face_recognition
from tqdm.autonotebook import tqdm
import torch
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset
```

Figure 14. imported libraries

Checking CUDA Availability:

```
print("Is dlib compiled with CUDA:", dlib.DLIB_USE_CUDA)
print("CUDA Device Count:", dlib.cuda.get_num_devices())
```

Figure 15. checking CUDA Availability

Checking **CUDA** availability ensures that the preprocessing steps in our project can be executed in the most efficient manner possible, leveraging available hardware resources to optimize performance and handle large-scale data processing. By confirming CUDA support and the number of available CUDA devices, we can utilize **GPU** acceleration for faster and more efficient processing of computationally intensive tasks, such as face detection and recognition in videos. This step ensures optimal performance, efficient resource utilization, system compatibility, and flexible code execution, making our preprocessing pipeline robust and capable of running on various hardware setups.

Using **dlib** in our project ensures that the preprocessing steps can be executed in the most efficient manner possible, leveraging available hardware resources to optimize performance and handle large-scale data processing. Dlib provides powerful tools for face detection and recognition, and by confirming **CUDA** support and the number of available CUDA devices, we can utilize **GPU** acceleration for faster and more efficient processing. This capability allows us to perform computationally intensive tasks, such as face detection and recognition in videos, much more quickly. Dlib ensures optimal performance, efficient resource utilization, system compatibility, and flexible code execution, making our preprocessing pipeline robust and capable of running on various hardware setups.

Helper Functions:

- **Function to Retrieve Video Files**

  Role: This function retrieves all video files from a specified directory and filters them based on the number of frames.

  Details:

  It searches for .mp4 files in the given path.

It checks the frame count of each video, and only includes videos with more than 150 frames.

The function also calculates and prints the total number of videos and the average frame count per video.

- **Function to Extract Frames from a Video**

  Role: This function extracts frames from a given video file.

  Details:

  It opens a video file and reads it frame by frame.

  It yields each frame to be processed, allowing for operations on individual frames.Function to Create Face Videos.

- **Function to Create Face Videos**

  Role: This function processes a list of video paths, extracts face from the frames, and saves the processed frames as new videos.

  Details:

  It checks for existing processed videos to avoid duplication.

  For each video, it extracts frames and batches them.

  It uses the face_recognition library to detect faces in the frames.

  Detected faces are resized and saved as a new video file.

5.3 Model Details:

The model, developed with PyTorch [15], benefits from GPU acceleration, enabling efficient processing of extensive datasets. The system's accuracy is evaluated using a Confusion Matrix, providing insights into performance.

The model consists of following layers:

1. ResNeXt CNN:

- The pre-trained ResNeXt50_32x4d model is used as the backbone [4] .
- This model consists of 50 layers with 32 x 4 dimensions.
- The detailed architecture of ResNeXt is shown in Figures 8.1, 8.2, and 8.3.

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

Figure 16. ResNext Architecture

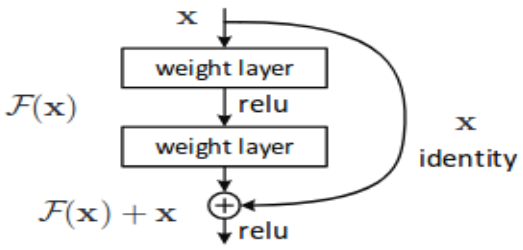

Figure 17. ResNext Working

2. Sequential Layer:

- This layer is used to store the feature vectors returned by the ResNeXt model in an ordered way.
- This allows the features to be passed sequentially to the LSTM layer

3. LSTM Layer:

- The LSTM layer is used for sequence processing and detecting temporal changes between frames.
- The input to the LSTM is a 2048-dimensional feature vector.
- The LSTM layer has 1 layer with 2048 latent dimensions and 2048 hidden layers, along with a dropout rate of 0.4.
- The LSTM processes the frames sequentially, comparing the current frame with the previous 'n' frames to perform temporal analysis.

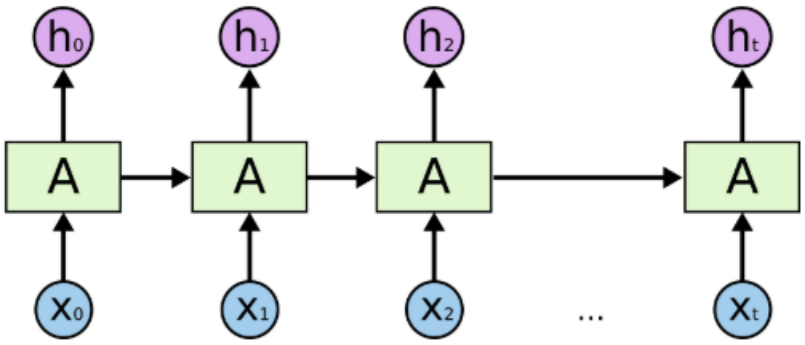

Figure 18. Overview of LSTM Architecture

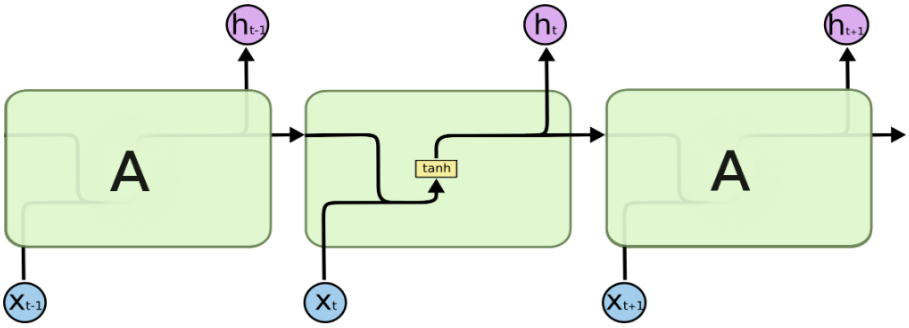

Figure 19. Internal LSTM Architecture

4. ReLU Activation:

5. The **LeakyReLU** activation function is used, which outputs the input directly if it is greater than 0; otherwise, it outputs a small fraction of the input (commonly 0.1 times the input).

6. Like ReLU, **LeakyReLU** introduces non-linearity into the model, allowing it to learn complex relationships in the data.
7. **Advantages over ReLU**: Prevents "**dying neurons**" by maintaining a small, non-zero gradient for negative inputs, which can enhance the model's ability to learn from data and improve overall training performance.

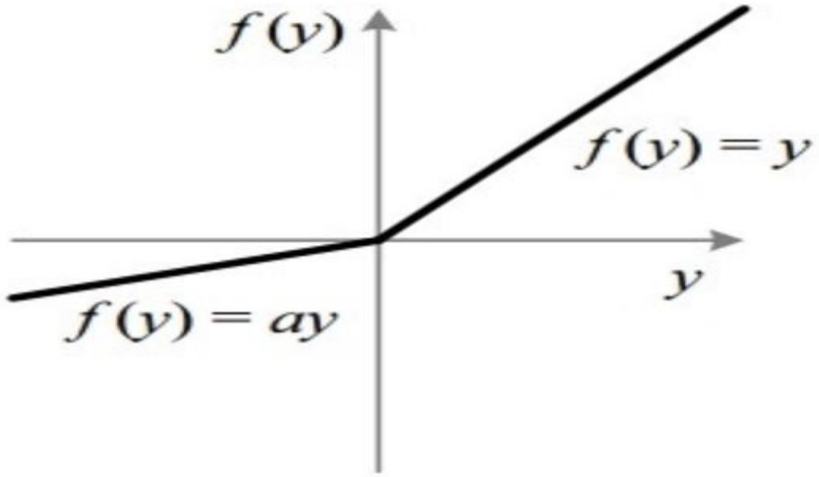

Figure 20. LeakyReLU Activation function

8. **Dropout Layer:**
    - A dropout layer with a value of 0.4 is used to prevent overfitting.
    - Dropout randomly sets the output of a given neuron to 0, making the model more robust and generalized.

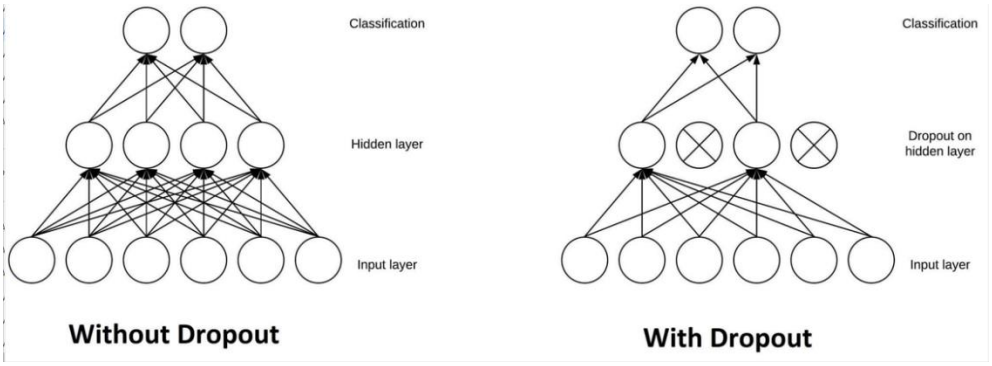

Figure 21. Overview of Dropout layer

5. Adaptive Average Pooling Layer:

- This layer is used to reduce variance, reduce computational complexity, and extract low-level features from the neighborhood.
- A 2-dimensional Adaptive Average Pooling layer is used in the model.

```
#Model with feature visualization
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)))
```

Figure 22. code for model class

5.4 Model Training Details:

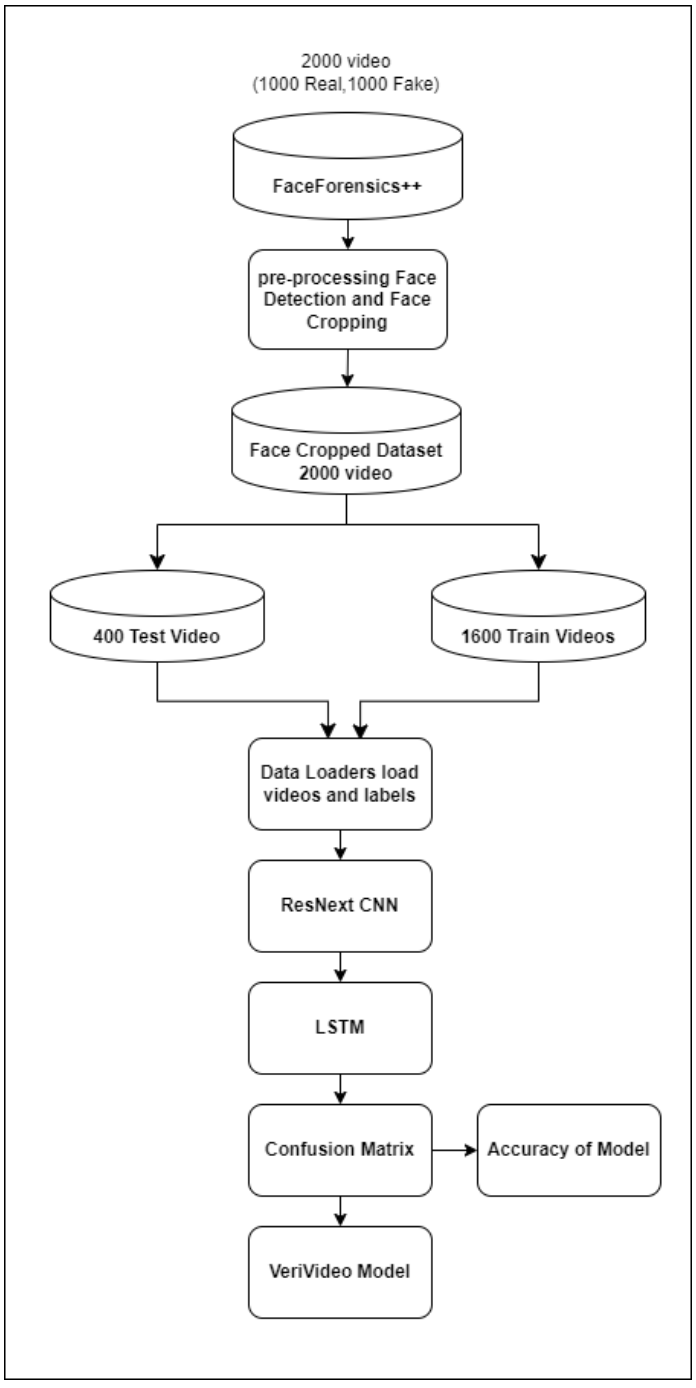

Figure 23. training flow

Train Test Split:

- The full video dataset is split into a training set and a test set using a 80/20 ratio.
- This means the training set contains **1600 videos** (80% of the total), while the test set comprises the remaining **400 videos** (20%).
- Importantly, the split is performed in a balanced manner, ensuring that both the training and test sets have an equal representation of 50% real and 50% fake videos.
- This balanced partitioning is crucial to avoid skewing the model's performance and to ensure a fair evaluation of its ability to distinguish real from fake videos.

```
header_list = ["filename","label"]
labels = pd.read_csv('/content/drive/MyDrive/DatasetsLabels/all_video_labels.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)
```

Figure 24. code for splitting the data

Data Loading:

- A custom data loader is employed to efficiently load the video data and labels in batches of size 4 during the training process.
- The data loader handles tasks such as video decoding, preprocessing, and on-the-fly augmentation to introduce variability and improve the model's generalization.
- By using a batch size of 4, the training process can leverage the benefits of parallel computation on the available hardware resources.

```
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 = 40,transform = train_transforms)
#print(train_data)
val_data = video_dataset(valid_videos,labels,sequence_length = 40,transform = train_transforms)
train_loader1 = DataLoader(train_data,batch_size = 4,shuffle = True,num_workers = 4)
valid_loader1 = DataLoader(val_data,batch_size = 4,shuffle = True,num_workers = 4)
image,label = train_data[0]
im_plot(image[0,:,:,:])
```

Figure 25. code for data loading

Training Regimen:

- The model is trained for a total of **20** epochs using the Adam optimizer, a popular adaptive optimization algorithm.
- The learning rate is set to a relatively low value of 1e-5 to ensure stable and gradual convergence of the model parameters.
- Additionally, a weight decay of 1e-5 is applied as a regularization technique to prevent overfitting and improve the model's generalization to unseen data.
- The loss function employed is **Cross-Entropy**, which is a widely used objective for binary classification tasks like real vs. fake video detection.

```
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)

# Learning rate scheduler
base_scheduler = StepLR(optimizer, step_size=10, gamma=0.1)
warmup_scheduler = GradualWarmupScheduler(optimizer, multiplier=1, total_iters=5, after_scheduler=base_scheduler)

#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_loader1,model,criterion,optimizer)
    train_loss_avg.append(l)
    train_accuracy.append(acc)
    true,pred,tl,t_acc = test(epoch,model,valid_loader1,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)
```

Figure 26. code for training loop

Softmax Layer:

- The final layer of the model is a Softmax layer with two output nodes, representing the predicted probabilities of the input being "Real" or "Fake".
- The Softmax function is a key component that transforms the raw output logits into a probability distribution, where the outputs sum to 1.
- This allows the model to provide well-calibrated probability estimates for each class, which is crucial for making informed decisions about the authenticity of the input videos.

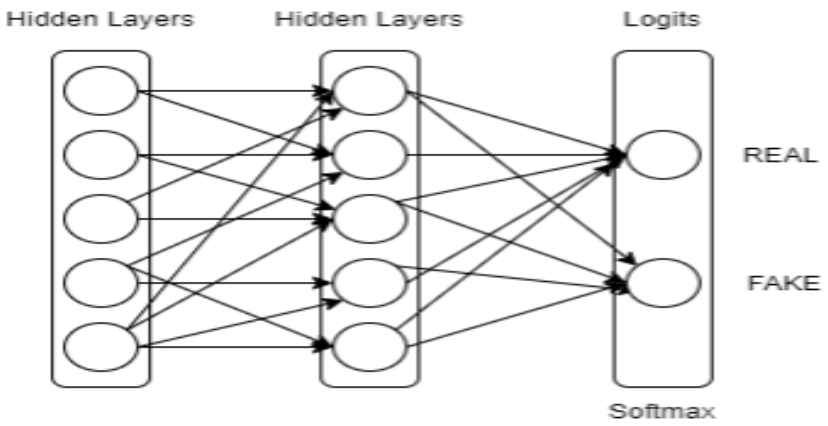

Figure 27. SoftMax Layer

Role of Learning Rate Schedulers in Training:

Learning rate schedulers automate the adjustment of learning rates during training to optimize model learning dynamics and performance.

- **GradualWarmupScheduler**: Initiates training with a gradually increasing learning rate, which stabilizes early training phases by allowing the model parameters to adjust smoothly. This approach can prevent sudden weight adjustments that may lead to instability.

- **StepLR**: Adjusts the learning rate periodically by a specified factor (gamma) at predefined intervals (step_size). This helps the model converge more effectively by reducing the learning rate as training progresses, which can prevent overfitting and improve generalization.

These schedulers play a crucial role in improving model stability, accelerating convergence, and enhancing overall performance in deep learning training workflows.

```python
from torch.optim.lr_scheduler import StepLR
import torch.optim as optim

# Warm-Up Scheduler Implementation
class GradualWarmupScheduler(optim.lr_scheduler._LRScheduler):
    def __init__(self, optimizer, multiplier, total_iters, after_scheduler=None):
        self.multiplier = multiplier
        self.total_iters = total_iters
        self.after_scheduler = after_scheduler
        self.finished = False
        super(GradualWarmupScheduler, self).__init__(optimizer)

    def get_lr(self):
        if self.last_epoch > self.total_iters:
            if self.after_scheduler:
                if not self.finished:
                    self.after_scheduler.base_lrs = [base_lr * self.multiplier for base_lr in self.base_lrs]
                    self.finished = True
                return self.after_scheduler.get_last_lr()
            return [base_lr * self.multiplier for base_lr in self.base_lrs]

        return [base_lr * ((self.multiplier - 1) * self.last_epoch / self.total_iters + 1) for base_lr in self.base_lrs]

    def step(self, epoch=None):
        if self.finished and self.after_scheduler:
            if epoch is None:
                self.after_scheduler.step(None)
            else:
                self.after_scheduler.step(epoch - self.total_iters)
        else:
            return super(GradualWarmupScheduler, self).step(epoch)
```

Figure 28. code for Gradual Warmup Scheduler

Confusion Matrix:

- The performance of the trained model is evaluated using a confusion matrix, a powerful tool for analyzing the classification accuracy.
- The confusion matrix provides a detailed breakdown of the model's classifications, including the number of:
- **True Positives (TP)**: Correctly identified real videos
- **True Negatives (TN)**: Correctly identified fake videos
- **False Positives (FP)**: Fake videos incorrectly identified as real
- **False Negatives (FN)**: Real videos incorrectly identified as fake

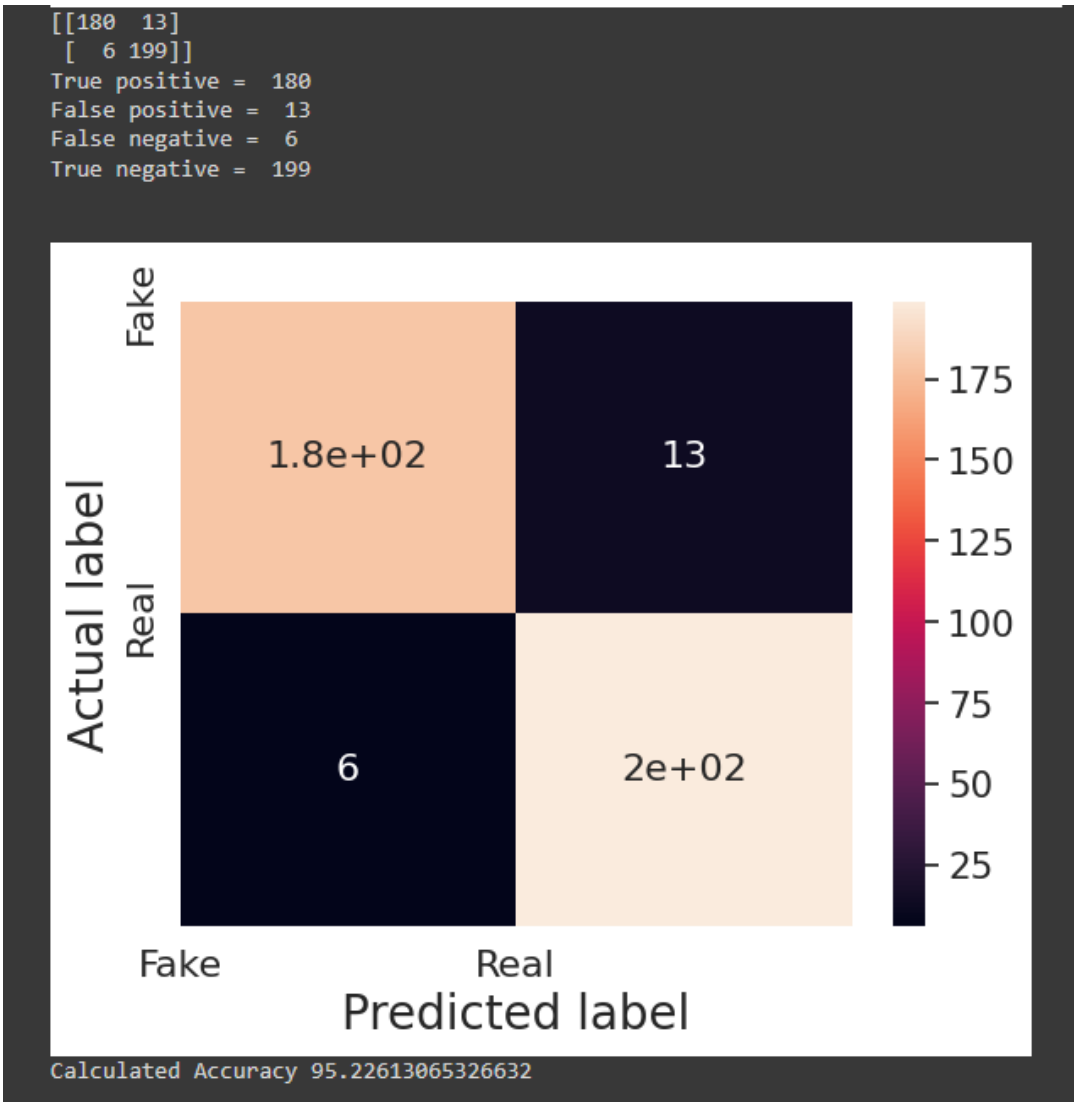

Figure 29. Confusion Matrix

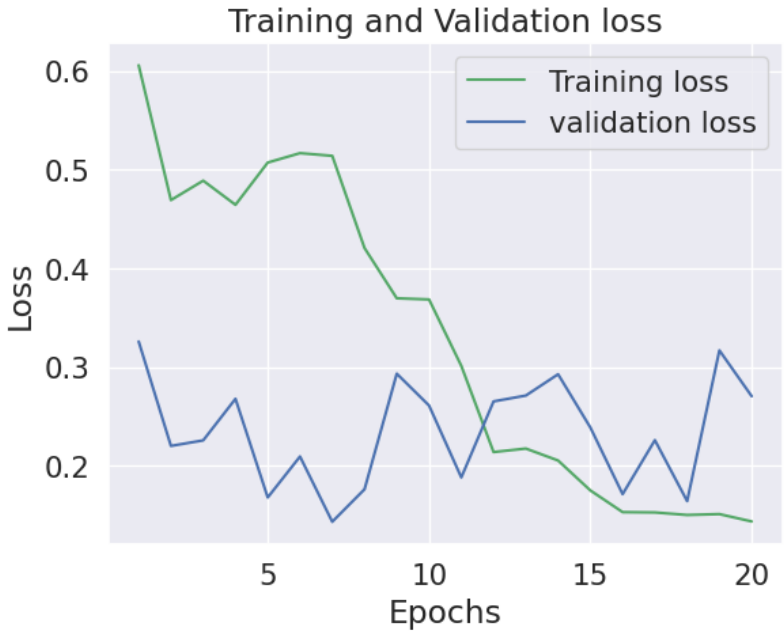

Figure 30. Training and Validation loss

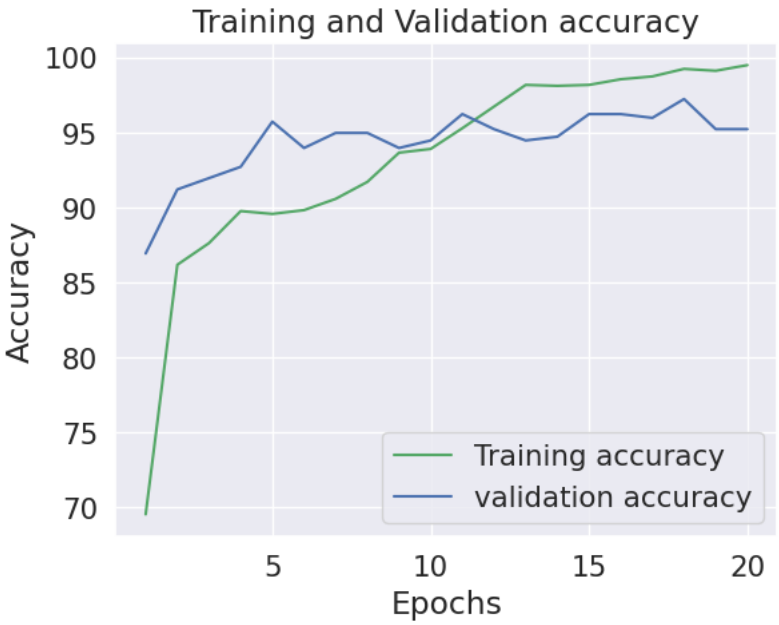

Figure 31. Training and Validation loss

Model Export:

- After the training and evaluation process is complete, the trained model is exported for deployment and use on real-time video data.
- This allows the model to be integrated into a user-facing web application, enabling practical, end-to-end detection of deepfakes.
- The exported model can be used to make classifications on any video inputs, facilitating the detection of deepfake videos in real-world scenarios. The model can be generalized to classify any video.

5.5 Model Classification Details:

Model Deployment: Once the VeriVideo model has been trained and evaluated to achieve the desired accuracy threshold, it undergoes deployment within the application framework. This deployment phase involves configuring the model to operate seamlessly within the application's ecosystem. Key steps include setting up the model's runtime environment, integrating it with the application's existing infrastructure, and optimizing its performance parameters. This ensures that the deployed model can efficiently process incoming video submissions.

Model Integration: The VeriVideo model is integrated into the application using Flask API for real-time video analysis. Upon video submission, Flask manages preprocessing: frames are extracted, faces are detected using OpenCV and face_recognition, and frames are resized for neural network evaluation.

The Flask API endpoints (/upload and /upload-url) handle video uploads and YouTube URL processing respectively. These endpoints facilitate seamless integration of the VeriVideo model, which utilizes a custom PyTorch architecture (ResNeXt-50 for feature extraction, LSTM and FC layers for prediction).

The model scrutinizes each frame to determine authenticity, providing a confidence score for each analysis. This setup enhances the application's capability to detect deepfakes effectively, ensuring robust security in digital media.

Verdict Delivery: The model quickly determines whether the video is real or fake and provides a confidence score for its output result.

Ongoing Vigilance: The application, with the model as its sentinel, remains vigilant in detecting deepfakes, contributing to the fight against manipulated media.

5.6 Model Accuracy:

The VeriVideo model achieves an accuracy of **95.22%** in detecting deepfakes. This high accuracy ensures reliable identification of manipulated media, contributing to the application's robustness in maintaining digital content authenticity.

Here's a comparison of the model's performance metrics for different sequence lengths:

Metric	Sequence Length 10 frames	Sequence Length 40 frames
Accuracy	92.46%	95.22%
Loss	≈ 0.5	≈ 0.26

5.7 Testing:

Our web application is designed to be user-friendly and intuitive, allowing users to easily upload videos or enter URLs to check if the video is real or fake. The main components of the user interface include:

Upload Video or Enter URL

Description: Users have two options to input a video for analysis:

1. **Upload Video:** Users can click on the "Upload" button to select a video file from their device. The application supports videos in MP4 format only. Additionally, the URL must be from YouTube.

2. **Enter URL:** Users can paste the URL of an online video in the designated field. This feature allows the application to fetch and analyze videos hosted on the web.

Scan Button

Description: Once the video is uploaded or the URL is entered, users can click the "Scan" button to initiate the analysis. The button is prominently displayed to ensure easy access.

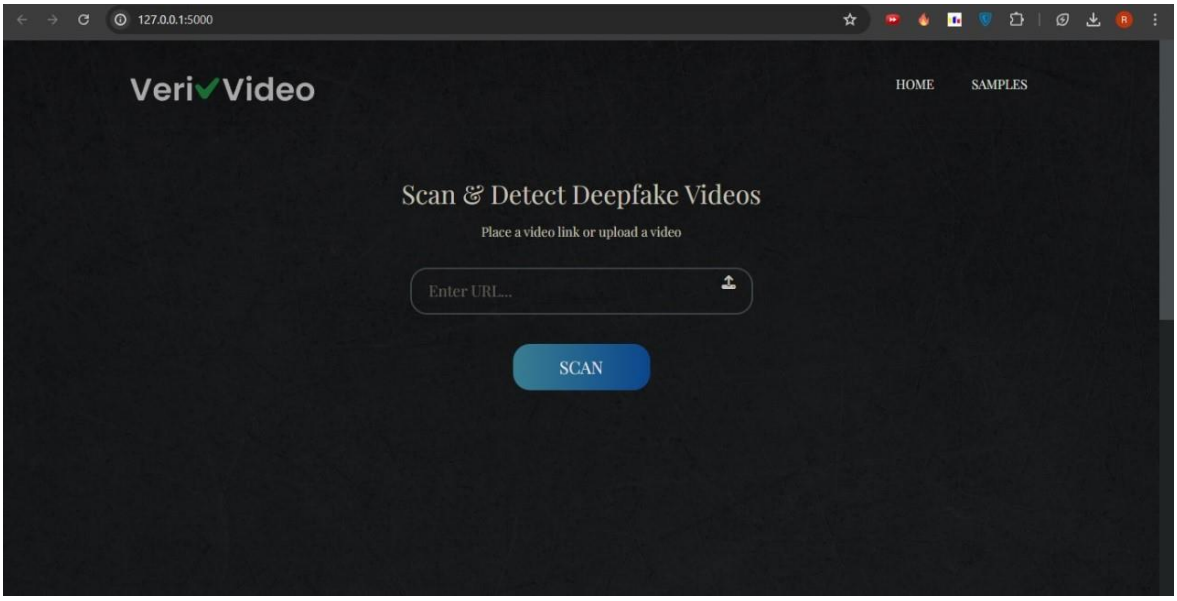

Figure 32. scan button

Uploading a Video from a Device

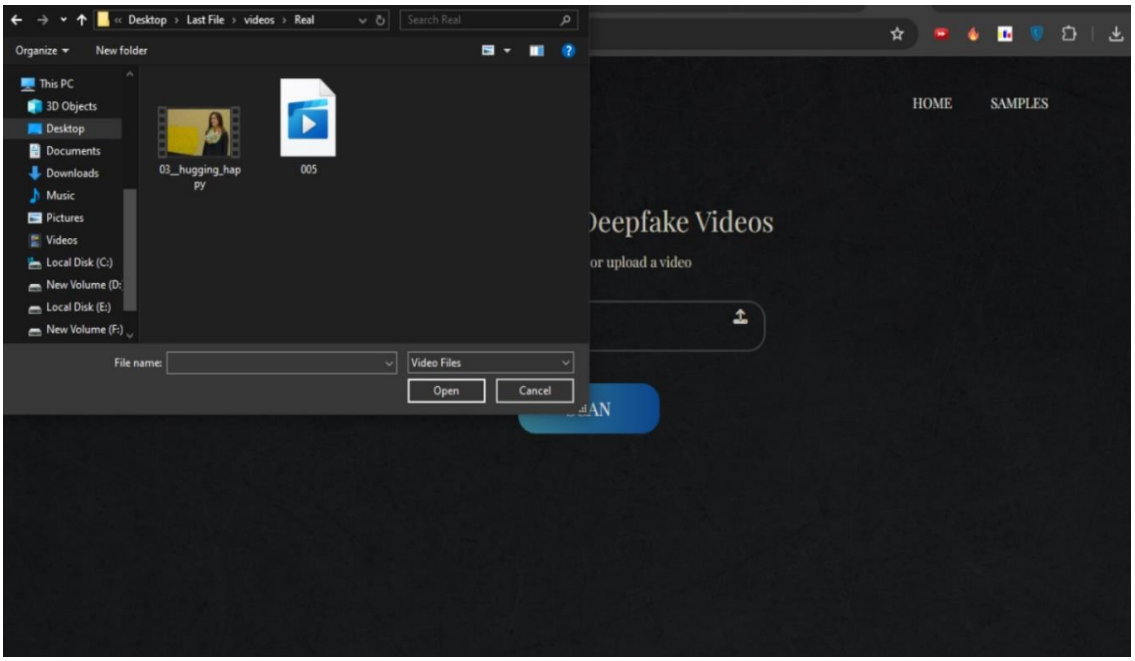

Figure 33. Upload video from device

Enter a URL of a Video

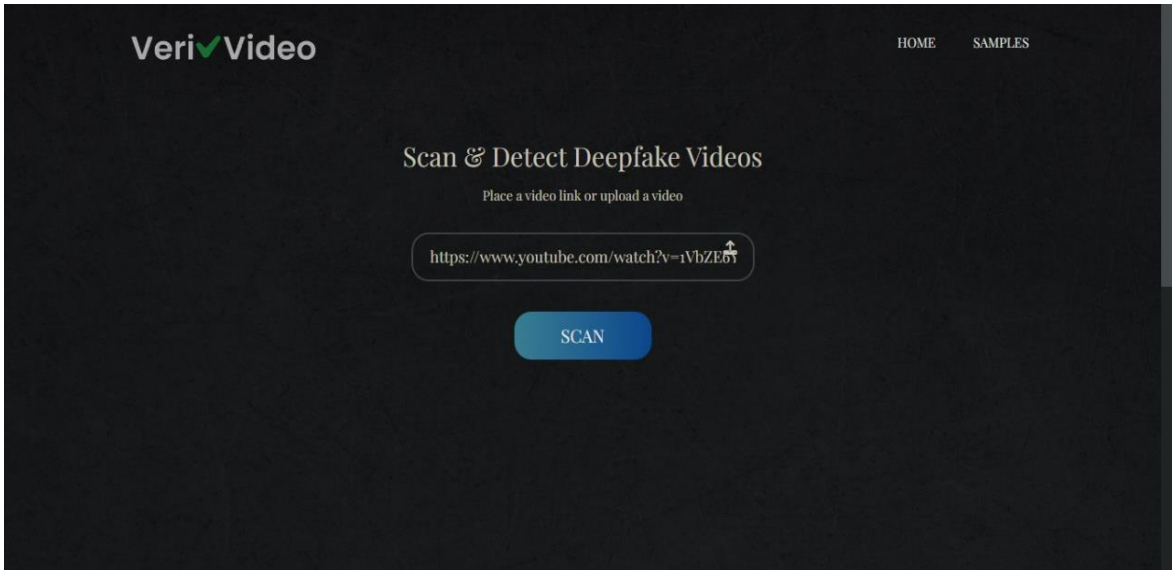

Figure 34. enter URL of a video

The Video after the preprocessing step:

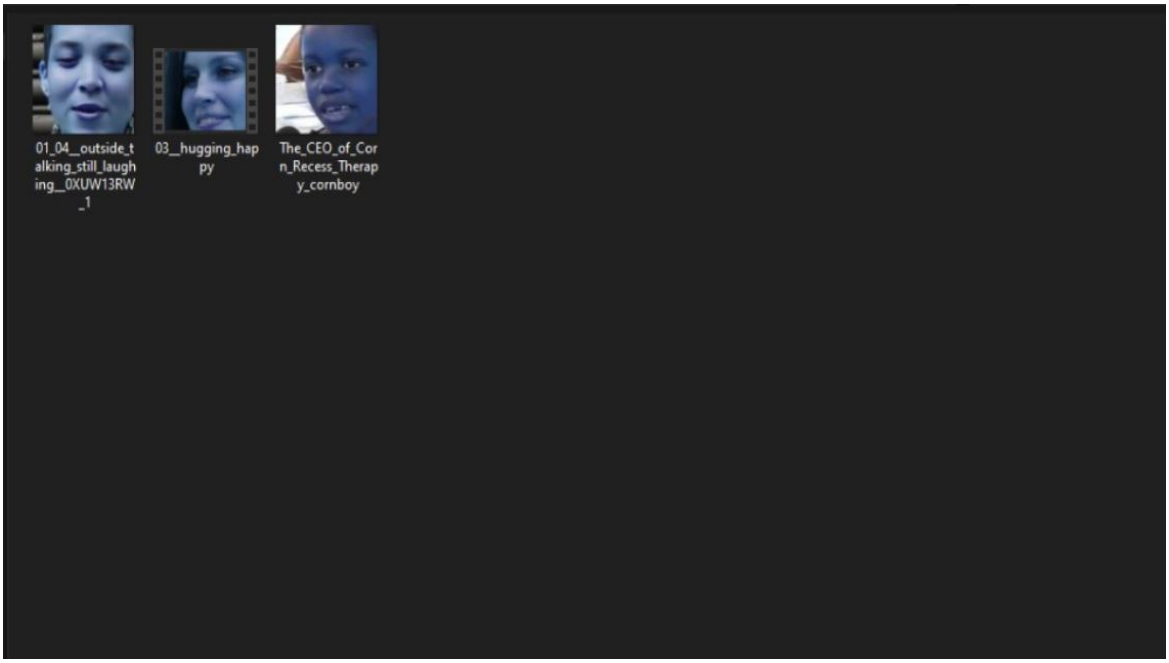

Figure 35. video after preprocessing step

Results

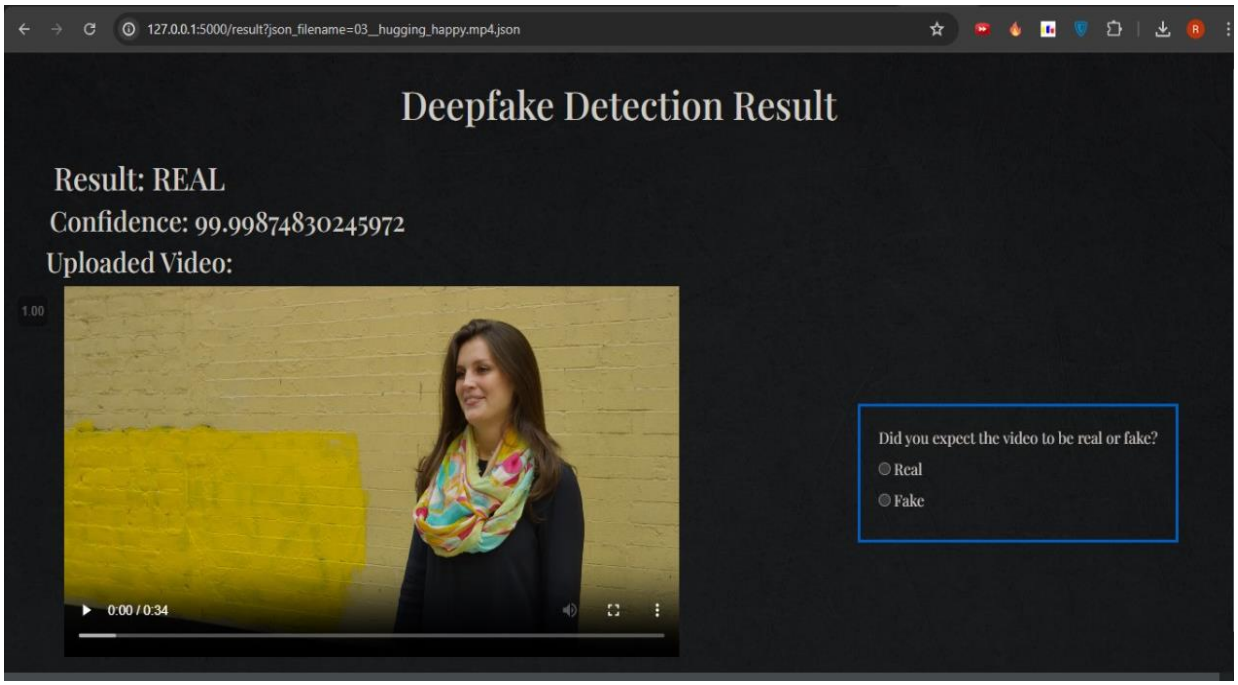

Figure 36. results of real uploaded video

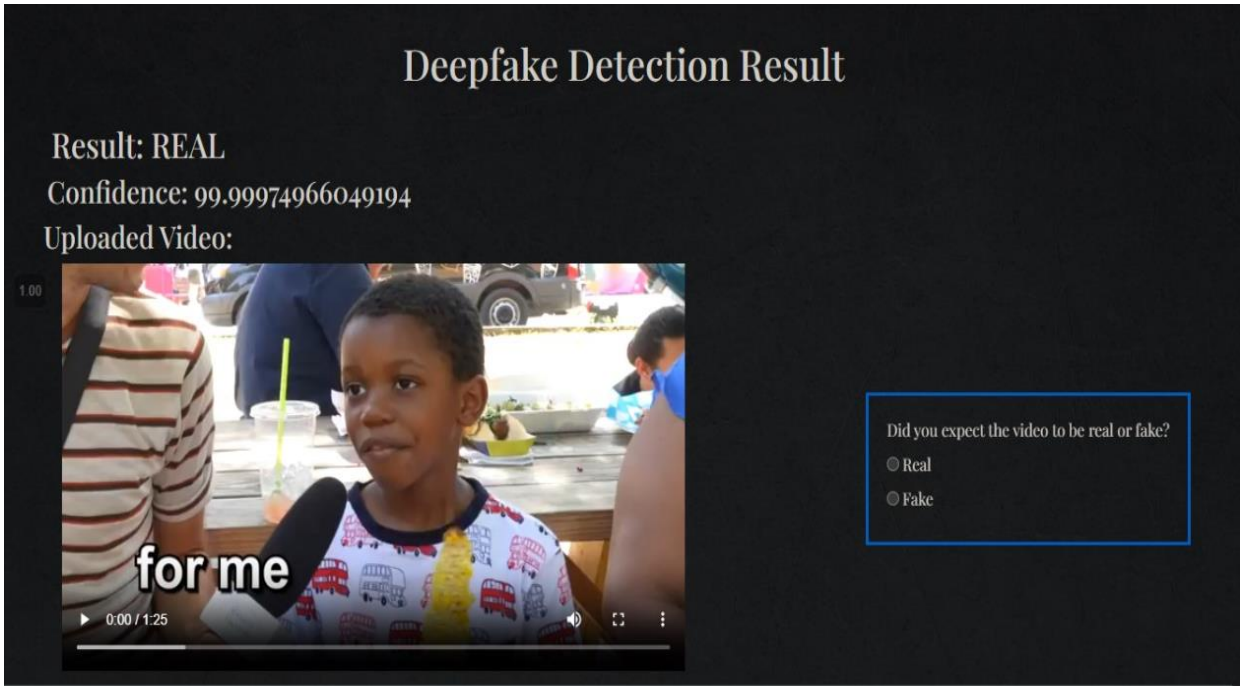

Figure 37. Results of real video after providing its URL

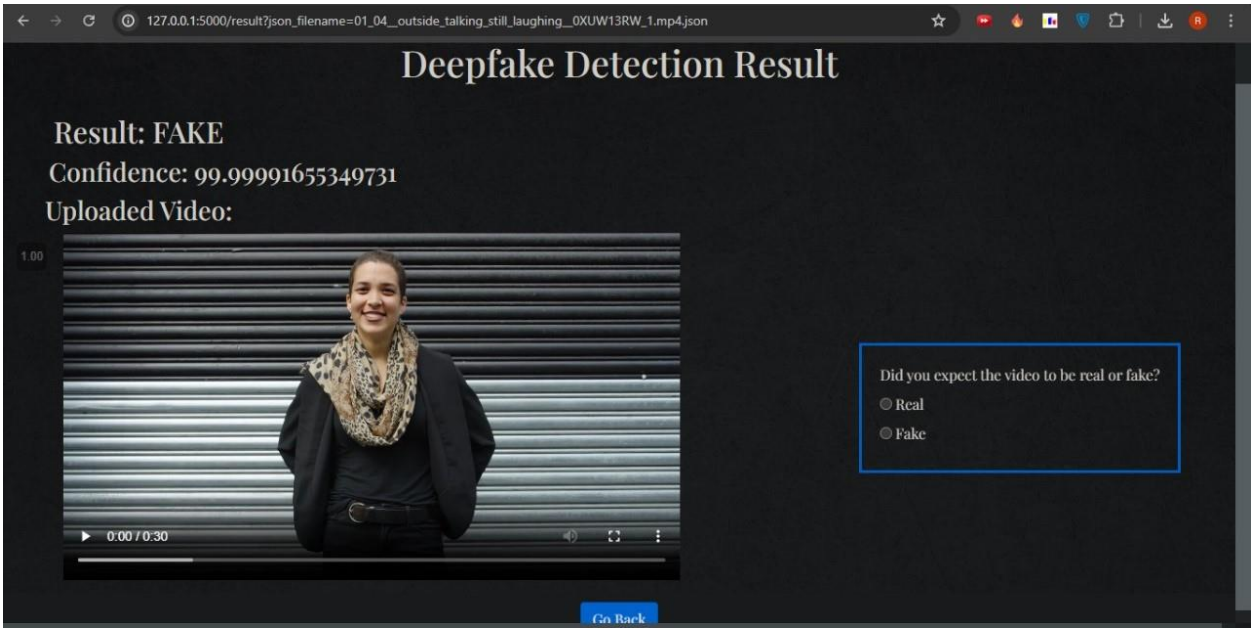

Figure 38. Results of fake uploaded video

Chapter 6: Conclusion and Future work

6.1 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 processing to spot the changes between the t and t+1 frame.

6.2 Future work

There is always a scope for enhancements in any developed system, especially when the project build using latest trending technology and has a good scope in future.

- Web based platform can be upscaled to a browser plugin for ease of access to the user.

- The model can be up to date by make it continually learns from the new data that users add to predict and based on users' feedback

References

[1] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, Matthias Nießner, "Learning to Detect Manipulated Facial Images," *arXiv:1901.08971.*

[2] Hanqing Zhao, Wenbo Zhou, Dongdong Chen, Tianyi Wei, Weiming Zhang, Nenghai Yu, "Multi-attentional Deepfake Detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[3] [Online]. Available: https://www.theguardian.com/technology/ng-interactive/2019/jun/22/the-rise-ofthe-.

[4] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He, "Aggregated Residual Transformations for Deep Neural Networks," *arxiv.org:1611.05431.*

[5] Sepp Hochreiter, Jürgen Schmidhuber, "LONG SHORT-TERM MEMORY," 1997. [Online]. Available: https://www.bioinf.jku.at/publications/older/2604.pdf.

[6] J. Lalor, "Improving Machine Learning Ability with Fine-Tuning," 2017.

[7] [Online]. Available: https://www.python.org/.

[8] [Online]. Available: https://pytorch.org/.

[9] [Online]. Available: https://colab.google/.

[10] [Online]. Available: https://opencv.org/.

[11] [Online]. Available: https://github.com/ageitgey/face_recognition.

[12] [Online]. Available: http://dlib.net/.

[13] [Online]. Available: https://matplotlib.org/.

[14] [Online]. Available: https://numpy.org/.

[15] hanh Thi Nguyen, Quoc Viet Hung Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, Thien Huynh-The, Saeid Nahavandi, Thanh Tam Nguyen, Quoc-Viet Pham, Cuong M. Nguyen, "Deep Learning for Deepfakes Creation and Detection: A Survey," arXiv:1909.11573, 2019.

[16] Edward J. Delp, David Güera, "Deepfake Video Detection Using Recurrent Neural Networks," IEEE, 2018.

[17] Hasam Khalid, Simon S. Woo, "OC-FakeDect: Classifying Deepfakes Using One-class Variational Autoencoder," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020.

[18] Darius Afchar, Vincent Nozick, Junichi Yamagishi, I.Echizen, "MesoNet: a Compact Facial Video Forgery Detection Network," in *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2018.

[19] Yuezun Li, Ming-Ching Chang, Siwei Lyu, "In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking," 2018.

[20] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, Stefan Wermter, "Continual Lifelong Learning with Neural Networks," arXiv:1802.07569, 2018.

[21] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, Raia Hadsell, "Progressive Neural Networks," arXiv:1606.04671, 2016.

[22] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, Raia Hadsell, "Overcoming catastrophic forgetting in neural networks," arXiv:1612.00796, London, 2017.

[23] [Online]. Available: https://pytorch.org/docs/stable/index.html.

[24] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, "Generative Adversarial Networks," arXiv:1406.2661, 2014.

[25] [Online]. Available: https://www.faceapp.com/. [Accessed 26 March 2020].

[26] [Online]. Available: https://faceswaponline.com.

[27] L. Verdoliva. [Online]. Available: https://arxiv.org/abs/2001.06564.