DETECTION OF DEEP FAKE MORPHING IN VIDEO FILES USING LONG SHORT-TERM MEMORY (LSTM) BASED ON RECURRING NEURAL NETWORKS (RNN)

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

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

Submitted by

MEDAPATI HARSHAVARDHAN REDDY	**122010325001**
PAMARTHI NITHIN	**122010325005**
CHILUKURI KARTHEEK SAI KUMAR	**122010325009**
SAMBANA TEJA SAI	**122010325011**

Under the esteemed guidance of

Mr. SAI PRATHEEK CHALAMALASETTY
ASSISTANT PROFESSOR, Dept. of CSE,
GITAM School of Technology,
GITAM Deemed to be University,
VISAKHAPATNAM

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM
(Deemed to be University)
VISAKHAPATNAM
MARCH 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM
(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled **"Detection of Deep Fake morphing in Video files using Long Short-Term Memory(LSTM) based Recurring Neural Networks(RNN)"** is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No(s).	Name(s)	Signature(s)
122010325001	MEDAPATI HARSHAVARDHAN REDDY	
122010325005	PAMARTHI NITHIN	
122010325009	CHILUKURI KARTHEEK SAI KUMAR	
122010325010	SAMBANA TEJA SAI	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM
(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled **"Detection of Deep Fake morphing in Video files using Long Short-Term Memory(LSTM) based Recurring Neural Networks(RNN)"** is a bona fide record of work carried out by students **SAMBANA TEJA SAI (122010325011), PAMARTHI NITHIN (122010325005), MEDAPATI HARSHAVARDHAN REDDY (122010325001), CHILUKURI KARTHEEK SAI KUMAR (122010325009),** students submitted in partial fulfilment of requirement for the award of degree of Bachelor of Technology in Computer Science and Engineering.

Project Guide	**Head of the Department**
Mr. SAI PRATHEEK CHALAMALASETTY	**Dr. G. Lakshmeeswari**
Assistant Professor, Dept. of CSE,	**Professor, Dept. of CSE,**
GST, GITAM Deemed to be University,	**GST, GITAM Deemed to be University,**
Visakhapatnam	**Visakhapatnam**

TABLE OF CONTENTS

1. ABSTRACT

Growing processing power has made deep learning algorithms so powerful that creating a synthetic movie that resembles a human being a technique referred to as a "deep fake" has become comparatively simple. It is easy to imagine scenarios in which people are blackmailed, manipulated into believing they are victims of revenge porn, or exploited to incite political unrest through realistic face swapping deepfakes. In this study, we provide a cutting-edge deep learning method that can reliably discriminate between real videos and artificial intelligence-generated phony ones.

Our technique can recognize whether a deep fake is being replaced or reenacted automatically. Our aim is to use artificial intelligence (AI) to fight artificial intelligence (AI). Our method employs a Res-Re-enactment of Option Neural Network to extract structure-level attributes, which are subsequently used to train the Long Short-Term Memory (LSTM). based Recurrent Neural Network (RNN) to categorize whether or not the video has been altered, that is, if it is a deepfake or authentic video.

We test our method on a large, balanced, and mixed dataset made by using the many accessible datasets, such as FF++, DFD challenge, and Celebrity-DF. We try our method on a real-time event and improve the model's performance on real-time data. We also show how our approach may produce competitive outcomes using a rather simple and dependable methodology.

2. INTRODUCTION

PROJECT IDEA

Deep learning algorithms have gotten so strong due to advances in processing capacity that making a human-like synthetic movie a process known as a "DF" has become quite easy. It is simple to imagine situations where people are used to provoke unrest in politics through realistic face swapping deepfakes, blackmailed, or tricked into thinking they are the victims of revenge porn. In this work, we provide a revolutionary deep learning-based method that can accurately distinguish between fake videos produced by artificial intelligence and real ones. Our method can automatically discern whether a deep fake is being replaced or reenacted.

Our objective is to combat artificial intelligence (AI) by means of AI. Using a Res-Re-enactment of Option Neural Network, our method extracts frame-level characteristics. These features are then used to train an LSTM-based Recurrent Neural Network (RNN) to determine whether the video is authentic or a deepfake. To mimic the real-time scenarios and enhance the model's performance on real-time data, we evaluate our approach on a large-scale balanced and mixed dataset created by merging the multiple existing datasets, such as Face-Forensic++, DFD Challenge, and Celeb-DF. We also demonstrate the relatively straightforward and reliable output that our system is capable of.

LITERATURE SURVEY

The identification of face warping artefacts was accomplished through the utilisation of a customised Convolutional Neural Network model, which involved comparing the generated face areas with their corresponding surrounding regions. The aforementioned item comprised of two distinct Face Artefacts. The approach employed by the researchers is founded upon the recognition that the existing deepfake algorithm exhibits constraints in its capacity to produce high-resolution images. Consequently, additional processing is necessary to align the faces that necessitate substitution in the original film. The methodology employed by the authors fails to consider the temporal analysis of the frames.

In order to distinguish between pristine and deepfake movies, the article "Identification via Eyes Flashing" describes an innovative method for doing so. This method involves analysing a person's eye blinking patterns. An eye blinking clipped frame was subjected to temporal analysis using the Long-term Recurrent Convolution Network (LRCN). The deepfake creation algorithms of today are so strong that even not blinking your eyes won't be enough to identify a deepfake. In order to identify deepfakes, additional factors need to be taken into account, such as crooked teeth, creases on the face, misaligned eyebrows, etc.

Using an enclosed system, this method may be used in a variety of scenarios, including those generated by computer video identification and replay attack detection, in order to detect changed pictures and videos.

In the learning phase of their technique, they used random noise, which is not advised. However, the model performed admirably on their dataset; however, noise in the training set could cause it to perform poorly on real-time data. We suggest using real-time and noiseless datasets to train our technique.

3. SYNOPSIS

A method called "deep fake" is used to simulate human images using neural network tools such as auto encoders and GANs (Generative Adversarial Networks). These applications overlay target images onto source movies using deep learning techniques to create deepfake videos that appear realistic. The variations between these deepfake videos and the real ones cannot be noticed with an unaided eye.

In this paper, we provide a revolutionary deep learning-based technique that can accurately distinguish between authentic videos and fake ones produced by AI. Our effective method of differentiating between pristine and deep fake videos is based on the limitations of the deep fake generating tools. The existing deep fake creation techniques leave some recognizable artifacts in the frames during the deep fake production process.

Our system Res-Next Convolution Neural Networks are used by our system to retrieve frame-level characteristics. These characteristics is utilized to instruct a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) to decide whether that video was altered, or put another way, if it is a deepfake or authentic video.

We suggested testing our approach on a sizable collection of deepfake films gathered from several online video portals. Our goal is to improve the performance of the deep fake detection model using real-time data. We used a variety of accessible datasets to train our model to accomplish this. for our model to be able to extract characteristics from many types of photos. From Face-Forensic++, we sufficiently retrieved films for DFD.

4. TECHNICAL KEYWORDS

AREA OF PROJECT

Our project uses neural network technology inspired by the human brain and falls under the category of deep learning, a subfield of artificial intelligence. Our project involves a significant amount of computer vision. With the aid of Open-CV, it assists in processing the video and frames. A model trained on Python Torch capable of determining whether a source video is pristine or deepfake.

TECHNICAL KEYWORDS

- Deep Learning
- Computer Vision
- Res-Next CNN
- Long Short Term Memory (LSTM)
- OpenCV, PyTorch
- Face Recognition
- GAN (Generative Adversarial Network)

5. DEFINITION AND SCOPE OF THE PROBLEM

A STATEMENT OF THE PROBLEM

Visual effects have been used for a while to prove that digital photographs and movies have been altered convincingly, but recent advances in deep learning have greatly increased the realism of false material and made it easier to create. These purportedly artificially generated materials are also known as deep fakes. Artificially intelligent tools may be used to quickly construct Deep Fakes. However, it might be difficult to spot these Deep Fakes. Deep fakes have been used historically to incite political unrest, stage terrorist attacks, produce revenge pornography, extort individuals, and other things. Therefore, it is essential to spot these deepfakes and prevent percolation. Social networking websites are used to propagate deep false. By using LSTM-based artificial neural networks, we have achieved strides in the detection of deep fakes.

GOALS AND OBJECTIVES

- The goal of our endeavour is to unearth the deeply faked versions of reality.
- 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 deep fake or pristine.
- Offer a simple method for uploading videos and determining if they are authentic or phony.

METHODOLOGIES OF PROBLEM SOLVING

ANALYSIS

- Solution Requirement

  We analysed the problem statement and found the feasibility of the solution of the problem. We read different research papers as mentioned in 3.3. After checking the feasibility of the problem statement. The next step is the data- set gathering and analysis.

We analysed the data set in a different approach of training like negatively or positively trained i.e., training the model with only fake or real video's but found that it may lead to addition of extra bias in the model leading to inaccurate predictions. Thus, after doing extensive study, we discovered that the optimal method for avoiding bias and variation in the technique and achieving acceptable accuracy is to train it in a balanced manner.

- Limitations on the Solution

We examined the answer with regard to expenses, processing speed, needs, degree of experience, and equipment availability.

- Parameter Identifiers:

• Moves of one's eyes	• Skin tone
• Enchanting teeth	• Facial Posture
• Larger eye distance	• Lighting
• Mustaches	• Different Pose
• Twice the edges, eyes, ears, nose	• Double chins
• Segmenting the Iris	• Hairstyle
• Lines on the Face	• Higher cheekbones
• Variable head posture	• Face Angle

Fig 5.1: Parameter Identifiers

DESIGN

After research and analysis, we developed the system architecture of the solution as mentioned in Chapter 6. We decided the baseline architecture of the Model which includes the different layers and their numbers.

DEVELOPMENT

After researching, we decided to use the PyTorch framework and the Python 3 platform for our programming. PyTorch is the choice due of its solid assistance for CUDA, or the Graphics Processing Unit, and its high degree of customization. To train an enormous amount of data sets and generate the final model, use the Google Cloud Platform.

EVALUATION

In order to evaluate our methodology, we analysed various real-time datasets, which encompassed YouTube videos. This study employs the Confusion Matrix methodology to assess the accuracy of the trained model.

OUTCOME

The system yields trained deepfake detection models that enable users to verify the authenticity of new videos, distinguishing between deep fakes and genuine ones.

APPLICATIONS

The user will submit the video for processing by uploading it using web-based applications. The uploaded video will be pre-processed by the model to determine if it is a real or deepfake.

6. PROJECT PLAN

MODEL ANALYSIS OF PROJECT

The application development model places emphasis on the individuals involved in the task, their collaborative efforts, and the management of risks. Consequently, we opt to employ the spiral model instead. We utilise the spiral model because it ensures adjustments can be made fast and throughout the development process by having constant evaluations to assess the product with the desired outputs requested. The spiral model is ideally suited for this type of application because we created it in modules. The utilisation of a Spiral approach facilitates the active participation of clients throughout the entirety of the project, encompassing various stages such as feature prioritisation, iterative planning and review sessions, and the regular integration of new features into algorithms. Nevertheless, clients must acknowledge that they are observing a work in progress in return for the enhanced advantage of transparency. We are utilizing the spiral model for product development because our model has a lot of risk, and the spiral model is capable of addressing the hazards.

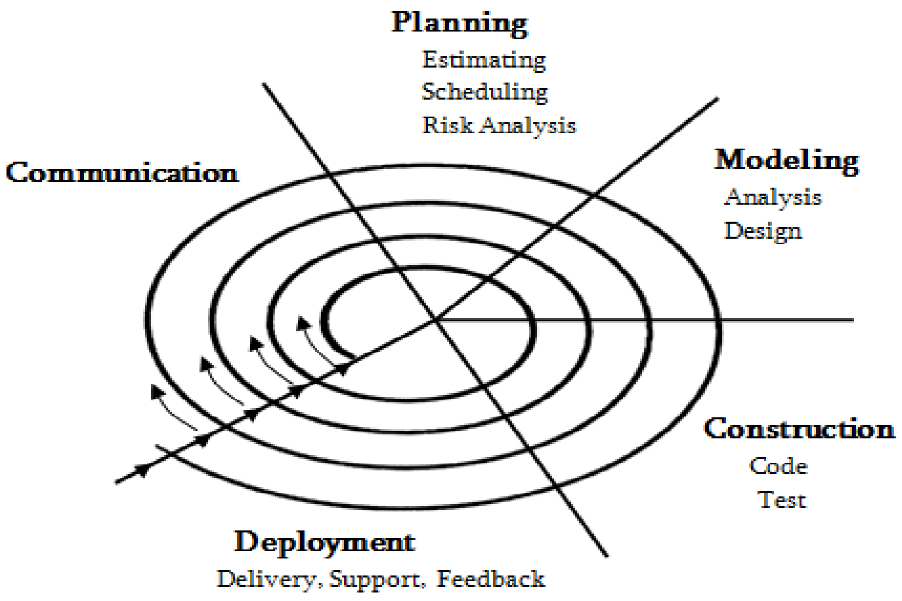


Fig 6.1: Project Plan

PROJECT SCHEDULE

Major Tasks in the Project stages are.

- Task 1: Dataset gathering and analysis

  This task consists of downloading the dataset. Analysing the dataset and making the dataset ready for the preprocessing.

- Task 2: Module 1 implementation

  Module 1 implementation consists of splitting the video to frames and cropping each frame consisting of faces.

- Task 3: Pre-processing

  Pre-processing includes the creation of the new dataset which includes only face cropped videos.

- Task 4: Module 2 Implementation

  Module 2 implementation consists of implementation of Data Loader for load- Ing the video and labels. Training a baseline model on a small amount of data.

- Task 5: Hyper parameter tuning

  This task includes the changing of the Learning rate, batch size, weight decay and model architecture until the maximum accuracy is achieved.

- Task 6: Training the final model

  The final model on a large dataset is trained based on the best hyper parameter identified in the Task 5.

- Task 7: Testing

  The complete application is tested using unit testing.

7. SOFTWARE REQUIREMENT SPECIFICATION

INTRODUCTION

THE AIM AND EXTENT OF THE RECORD

This paper outlines a project proposal for developing a neural network-based Deep Fake video detection system. The project sponsors and present and future developers working on deepfake video detection using neural networks are the intended readers of this publication. The project scope from the perspective of the "Deep Fake video detection" team (my mentors and I), use case diagrams, data flow diagrams, activity diagrams, functional and non-functional requirements, project risks and how those risks will be mitigated, the methodology by which we will develop the project, and metrics and measurements that will be recorded throughout the project are just a few of the things that will be included in the plan.

USE CASE VIEW:

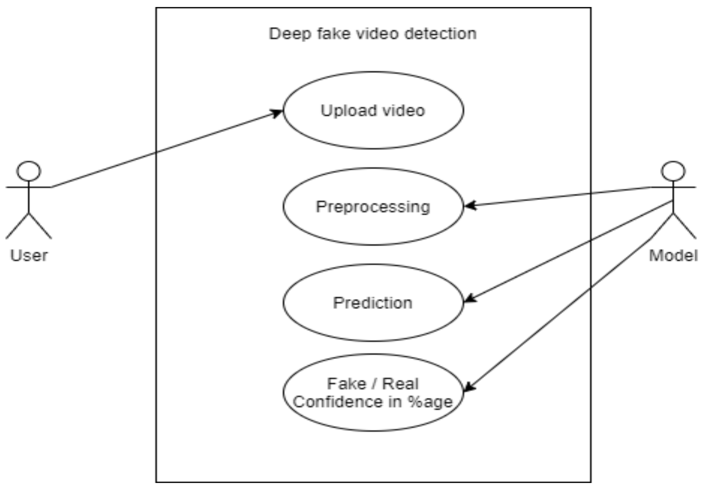


Fig 7.1: Use Case Diagram

DATA FLOW DIAGRAM:

DFD LEVEL – 0:

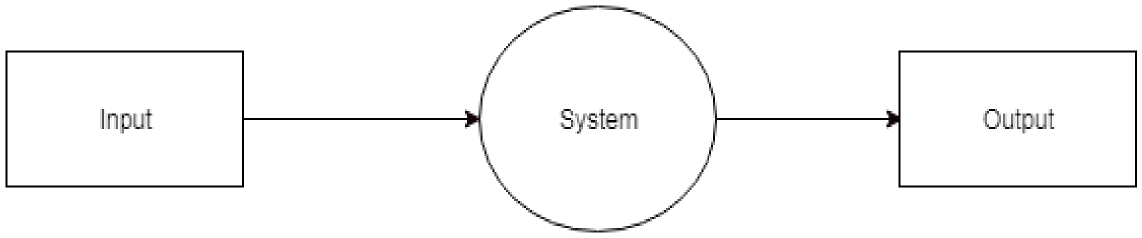


Fig 7.2: Data Flow Diagram - 1

Indicates the basic flow of data in the system. In this System Input is given equal importance as that for Output.

- Input: Here input to the system is uploading video.
- System: In system it shows all the details of the Video.
- Output: Output of this system shows the fake video or not.

Hence, the data flow diagram indicates the visualization of a system with its input and output flow.

DFD LEVEL-1

- DFD Level – 1 gives more in and out information of the system.
- Where the system gives detailed information of the procedure taking place

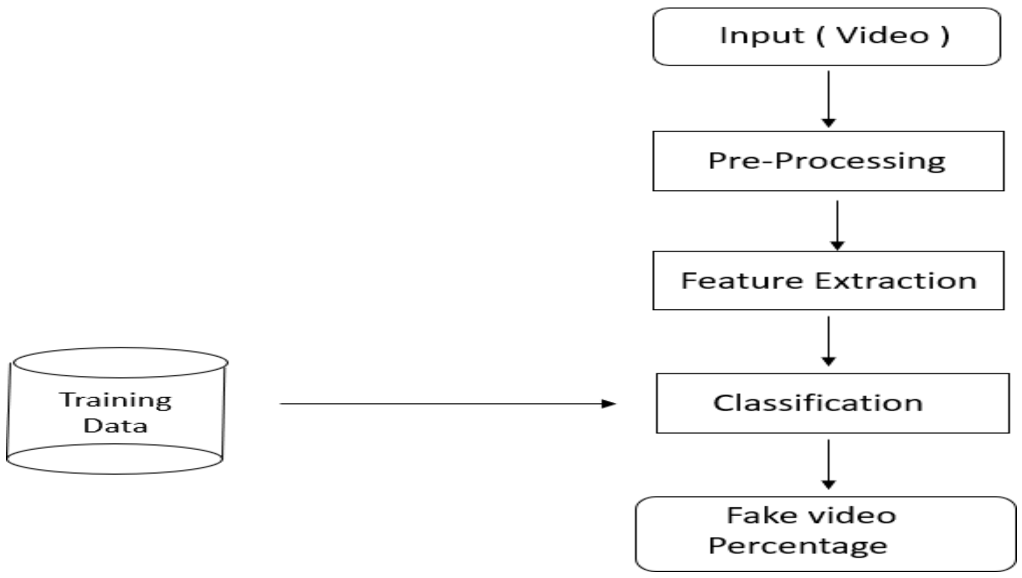


Fig 7.3: Data Flow Diagram 2

DFD LEVEL-2

- DFD level-2 enhances the functionality used by user etc.

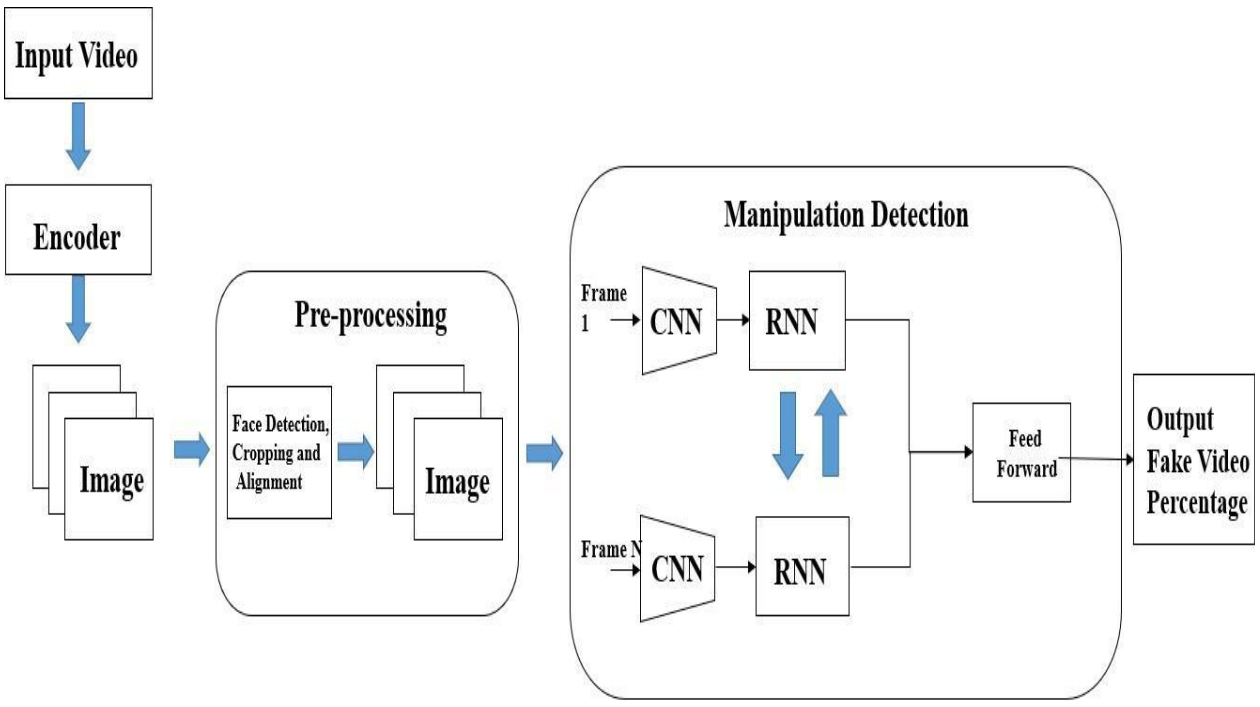


Fig 7.4: Data Flow Diagram 3

ACTIVITY DIAGRAMS:

1)TRAINING WORKFLOW:

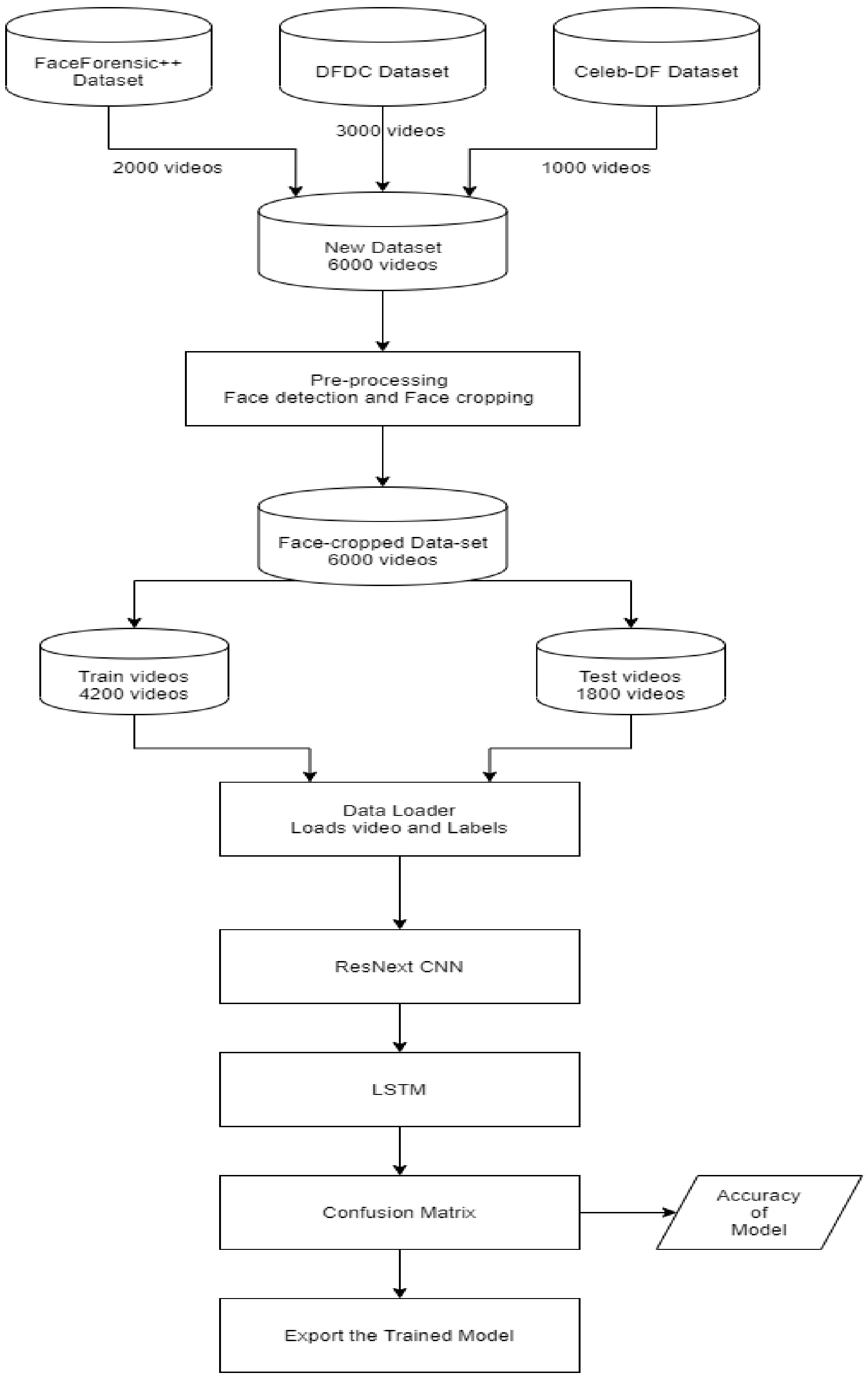


Fig 7.5: Training Workflow Diagram

TESTING WORKFLOW:

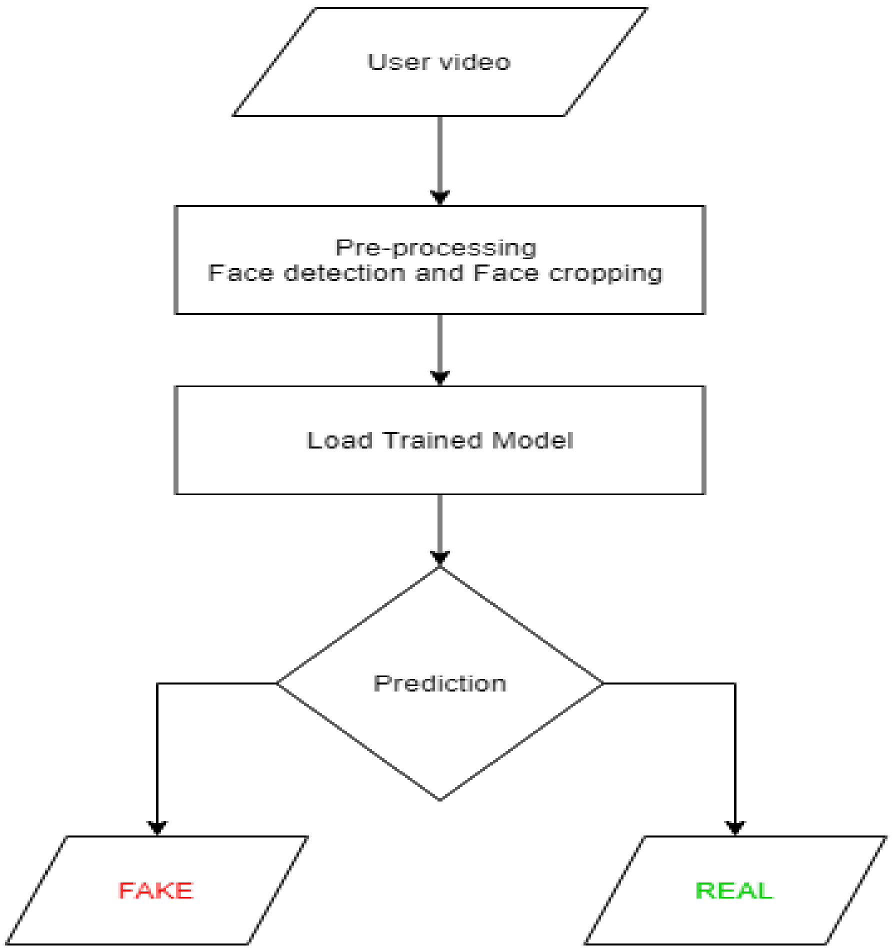


Fig 7.6: Testing Workflow Diagram

NON-FUNCTIONAL REQUIREMENTS:

PERFORMANCE REQUIREMENT:

The software should have an effective construction that allows it to be utilized for more real-world uses and provides consistent identification of fake videos.

- The design is versatile and quick to operate.
- The application reduces time, and it was quick and reliable.
- The system has adaptations for all regions.
- The system is easily integrated and compatible with future upgrades.

SAFETY REQUIREMENT:

- The Data integrity is preserved. Once the video is uploaded to the system. It is only processed by the algorithm. The videos are kept secure from human interventions, as the uploaded video is not able for human manipulation.

- To extend the safety of the videos uploaded by the user will be deleted after 30 min from the server.

SECURITY REQUIREMENT:

- While uploading the video, the video will be encrypted using a certain symmetric encryption algorithm. On the server also the video is in encrypted format only. The video is only decrypted from preprocessing till we get the output. After getting the output the video is again encrypted.

- SSL certification is made mandatory for Data security.

SEQUENCE DIAGRAM:

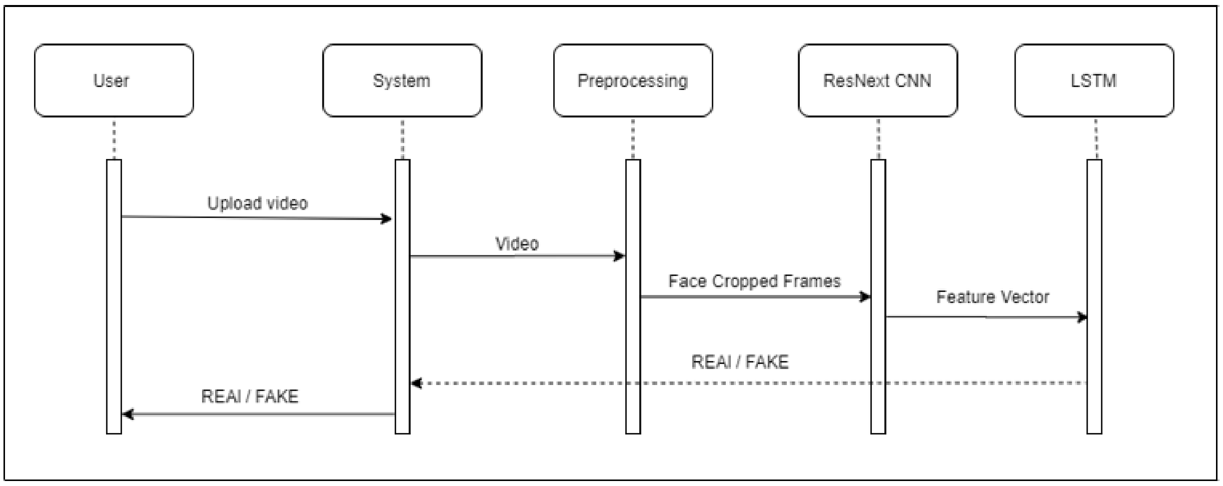


Fig 7.7: Sequence Diagram

8. DETAILED DESIGN DOCUMENT

INTRODUCTION:

SYSTEM ARCHITECTURE:

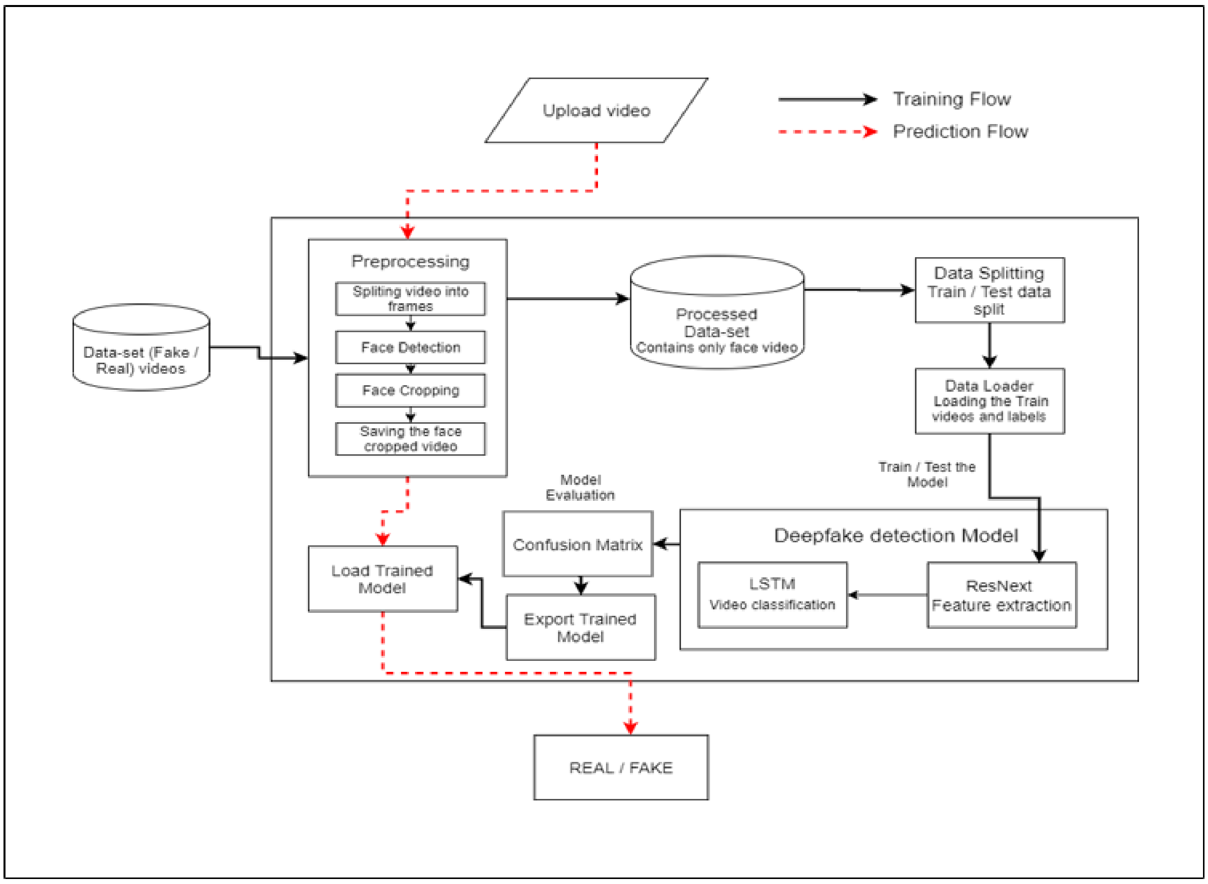


Fig 8.1: System Architecture

CREATING DEEP FAKE VIDEOS:

To identify deep fake films, it is essential to comprehend the process of deep fake video generation. Numerous techniques, like GANs and autoencoders, accept input from both the source image and the target video. These programs divide a video into frames, locate faces within the film, and then, on each frame, switch from the source face to the target face. The updated frames are then integrated with several pre-trained models. These models help improve video quality by erasing traces left over by the deepfake production technique. As a result, the deepfake appears realistic in nature. We also used the same method to detect deepfakes. Deepfakes built with pre-trained neural network models. Deepfakes generated with pre-trained neural network models are so lifelike that it is nearly hard to tell them apart with the naked eye. However, the deepfakes generating techniques leave some traces or artifacts in the video that are not visible to the naked eye. The goal of this work is to find invisible traces and recognizable artifacts in these videos and classify them as deepfake or real footage.

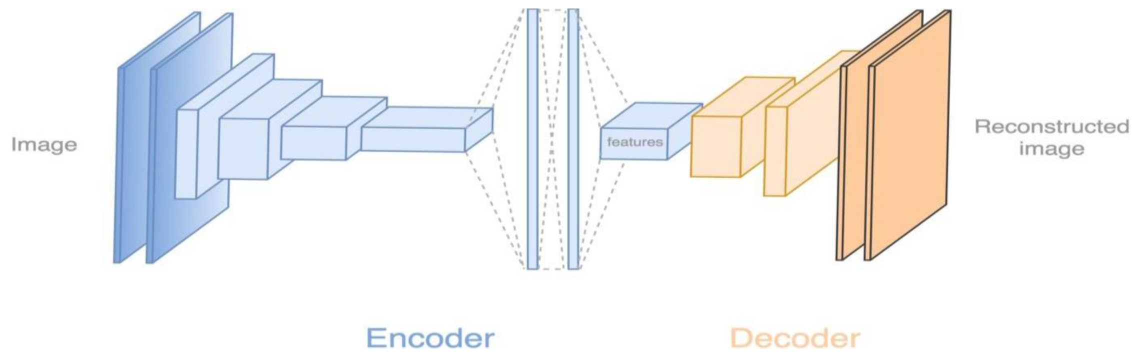


Fig 8.2: Image encoder and Decoder

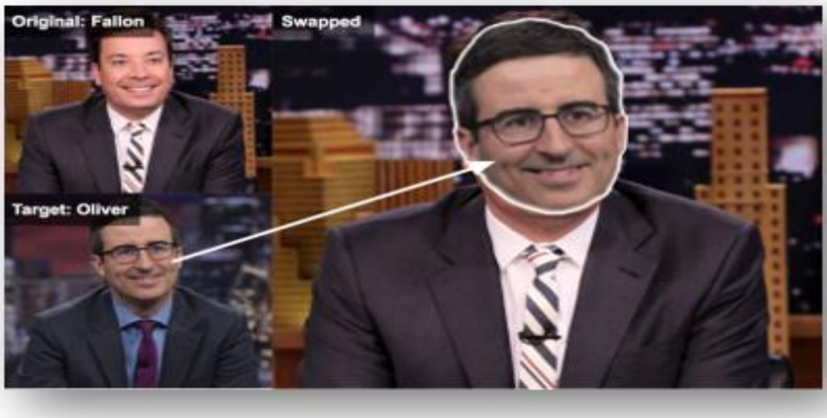


Fig 8.3: Example of Morphed Image

TOOLS FOR DEEP FAKE CREATION

1. Face swap
2. Face it
3. Deep Face Lab
4. Deepfake Capsule GAN
5. Large resolution face masked

ARCHITECTURAL DESIGN

MODULE 1: DATA COLLECTION

To enhance the model's capacity to make quick predictions. We acquired data from many publicly accessible data sets, including Face Forensic++ (FF), Deepfake Detection Challenge (DFDC), and Celeb-DF. In order to develop our own original dataset for precise and prompt identification on a range of movies, the dataset was then combined with the gathered datasets. To eliminate the model's training bias, we used a 50/50 mix of real and phony videos.
Since audio deepfake is outside the scope of this article, it is included in the dataset for the Deep Fake Detection Challenge (DFDC). Using a Python script, we pre-processed the DFDC dataset and eliminated the audio-altered videos.
The DFDC dataset was pre-processed, and 1500 Real and 1500 Fake films were extracted from it. The Face Forensic++(FF) dataset features 1000 Real and 1000 Fake movies, in contrast to the Celeb-Dataset's 500 Real and 500 Fake videos. As a result, our dataset has a total of 6000 videos, of which 3000 are real and 2000 are fake.

MODULE 2: PRIOR ANALYSIS

In this phase, the films have been prepared to remove any extraneous material and background noise. The video is only identified and edited to the relevant parts, focusing on the face. Frame-by-frame segmentation of the video is the first stage in video preprocessing.
The face is identified in each frame once the movie has been segmented into frames, and the picture is cut along the face. The chopped frame is later combined with other frames from the video to generate a new frame. The method is repeated for each film, resulting in a processed dataset consisting only of face-containing videos. In the initial processing stage, the faceless frame is ignored.
Based on the average of every movie's total number of frames, we establish a threshold value in order to maintain consistency in the number of frames. The absence of computer power is another justification for choosing a threshold number. It is quite difficult computationally to analyse all 300 frames at once in the experimental setting since a 10-second video at 30 frames per second (fps) comprises 300 frames. We selected 150 frames as our threshold number because of our Graphics Processor Unit (GPU) processing capability in an experimental environment. To show appropriate use, we only included the first 150 frames of the movie when storing the frames to the new dataset.
We examined the frames one after the other to demonstrate how Long Short-Term Memory (LSTM) should be used, i.e. the first 150 frames, rather than randomly. The freshly generated video has a frame rate of 30 frames per second and a resolution of 112 x 112.

9. PROJECT IMPLEMENTATION

INTRODUCTION:

There are many examples where deepfake creation technology is used to mis- lead the people on social media platforms by sharing the false deepfake videos of the famous personalities like Mark Zuckerberg Eve of House A.I. Hearing, Donald Trump's Breaking Bad series where he was introduced as James McGill, Barack Obama's public service announcement and many more. These types of deep fakes create a huge panic among the normal people, which raises the need to spot these deepfakes accurately so that they can be distinguished from the real videos.

Recent breakthroughs in technology have revolutionised the field of video manipulation. The advances in the modern open-source deep learning frameworks like TensorFlow, Keras, PyTorch along with cheap access to the high computation power has driven the paradigm shift. The Conventional autoencoders and (GAN) pretrained models have made the tampering of the realistic videos and images very easy. Moreover, access to these pretrained models through the smartphones and desktop applications like Face App and Face Swap has made the deepfake creation a childish thing. These applications generate a highly realistic synthesized transformation of faces in real videos. These apps also provide the user with more functionalities like changing the face hair style, gender, age and other attributes. These apps also allow the user to create a very high quality and indistinguishable deepfakes. Although some malicious deepfake films exist, they are a minority.

So far, the available tools that make deepfake films have been widely utilised to create fake celebrity pornographic videos. Some of the examples are Brad Pitt, Angelina Jolie nude videos. The real looking nature of the deepfake videos makes the celebrities and other famous personalities the target of pornographic material, fake surveillance videos, fake news and malicious hoaxes.

The Deepfakes are very much popular in creating political tension. Due to which it becomes very important to detect the deepfake videos and avoid the percolation of the deepfakes on the social media platforms.

TOOLS AND TECHNOLOGIES USED:

PLANNING

1. Open Project

UML TOOLS

2. draw.io

PROGRAMMING LANGUAGES

3. Python3

IDE

4. Google Collab
5. Jupyter Notebook
6. Visual Studio Code

VERSIONING CONTROL

7. Git

APPLICATION AND WEB SERVERS:

8. Google Cloud Engine

LIBRARIES:

9. torch
10. torch vision
11. OS
12. NumPy
13. cv2
14. matplotlib
15. face recognition
16. Json
17. pandas
18. copy, glob, random, sklearn

ALGORITHM DETAILS:

DATASET DETAILS:

- Face Forensics++
- Collab-DF, Deepfake Detection Challenge

PREPROCESSING DETAILS:

- Using glob, we imported all the videos in the directory in a python list.
- To read the clips and get the average number of frames, we use cv2.VideoCapture.
- To guarantee consistency, 180 is determined to be the ideal quantity for the newly formed dataset.
- The motion picture is divided into frames, and each frame has its facial location trimmed.
- Use Video Writer to add face-cropped frames to a new video. The latest film is an mp4 file with a resolution of 112 × 112 pixels and is produced at 30 frames per second.
- Instead of selecting the random videos, to make the proper use of LSTM for temporal sequence analysis the first 150 frames are written to the new video.

PRE-PROCESSING STEPS:

DATASET:

Some of the dataset we used are listed below:

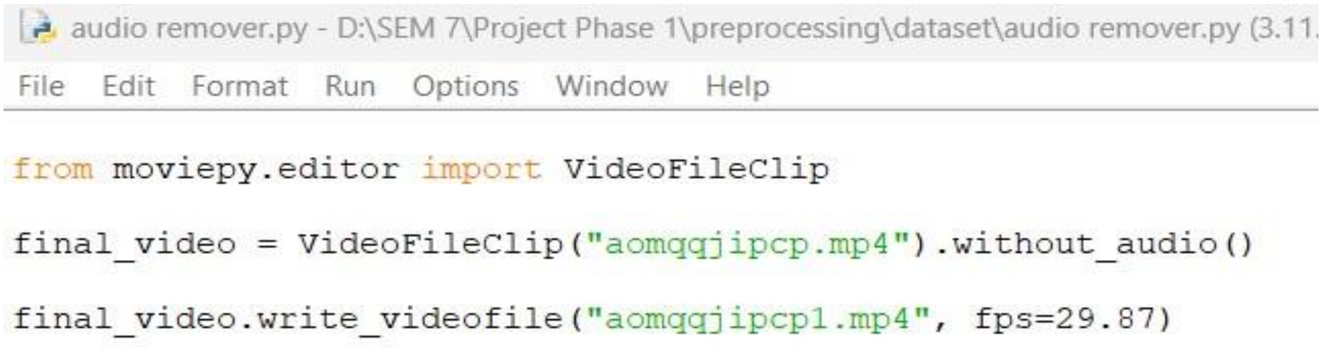
- Face Forensics++
- Celeb-DF
- Deepfake Detection Challenge

PREPROCESSING:

- Loading the dataset
- Split the motion picture up into frames.
- Take out the face off each frame.
- Face cropped video is saved.

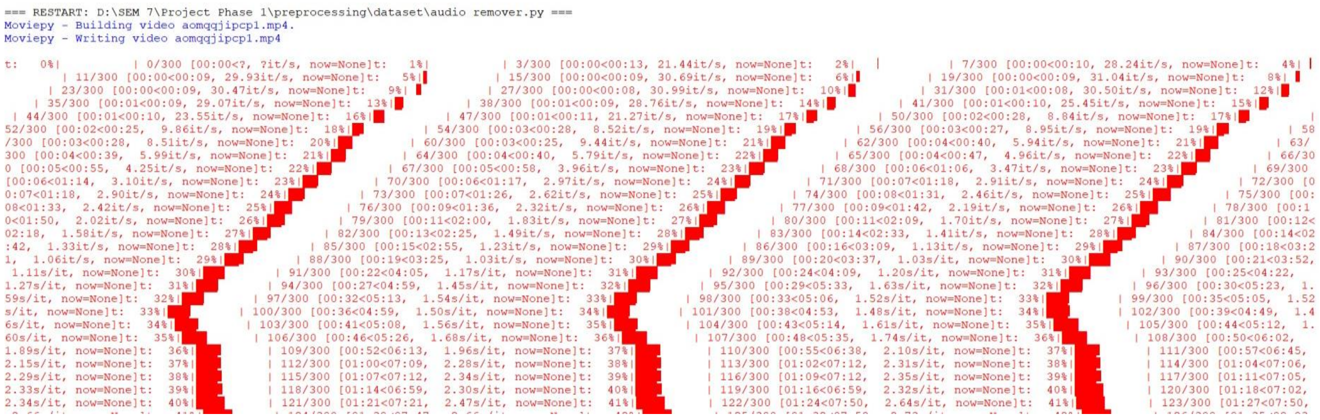
LOAD THE DATASET:

- The dataset which is downloaded is uploaded into google drive.


audio remover.py - D:\SEM 7\Project Phase 1\preprocessing\dataset\audio remover.py (3.11
File Edit Format Run Options Window Help

```python
from moviepy.editor import VideoFileClip

final_video = VideoFileClip("aomqqjipcp.mp4").without_audio()

final_video.write_videofile("aomqqjipcp1.mp4", fps=29.87)
```

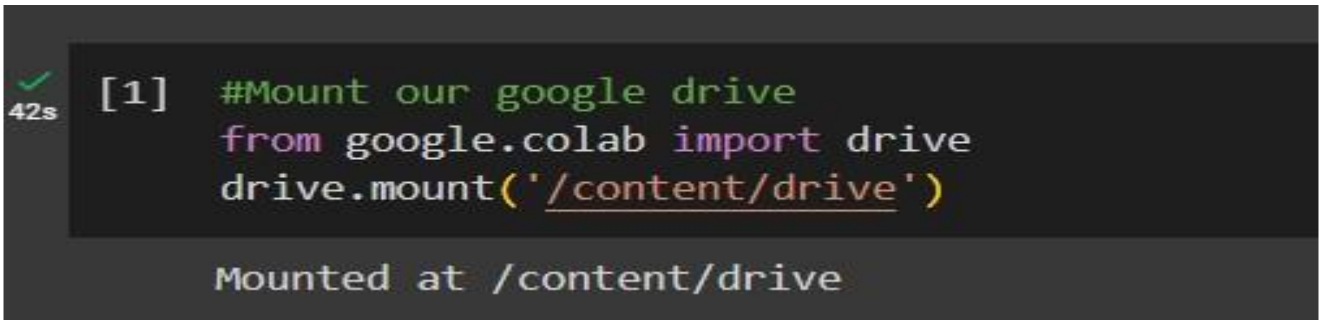
Fig 9.1.1: Code for removing audio from video.



```
=== RESTART: D:\SEM 7\Project Phase 1\preprocessing\dataset\audio remover.py ===
Moviepy - Building video aomqqjipcp1.mp4.
Moviepy - Writing video aomqqjipcp1.mp4

t:   0%|          | 0/300 [00:00<?, ?it/s, now=None]t:   1%|          | 3/300 [00:00<00:13, 21.44it/s, now=None]t:   2%|  |          | 7/300 [00:00<00:10, 28.24it/s, now=None]t:   4%| |
        | 11/300 [00:00<00:09, 29.93it/s, now=None]t:   5%|█         | 15/300 [00:00<00:09, 30.69it/s, now=None]t:   6%|█          | 19/300 [00:00<00:09, 31.04it/s, now=None]t:   8%| █
      | 23/300 [00:00<00:09, 30.47it/s, now=None]t:   9%| █        | 27/300 [00:00<00:08, 30.99it/s, now=None]t:  10%|█         | 31/300 [00:01<00:08, 30.50it/s, now=None]t:  12%|█
    | 35/300 [00:01<00:09, 29.07it/s, now=None]t:  13%|█        | 38/300 [00:01<00:09, 28.76it/s, now=None]t:  14%|█        | 41/300 [00:01<00:10, 25.45it/s, now=None]t:  15%|█
 | 44/300 [00:01<00:10, 23.55it/s, now=None]t:  16%|█        | 47/300 [00:01<00:11, 21.27it/s, now=None]t:  17%|█        | 50/300 [00:02<00:28,  8.84it/s, now=None]t:  17%|█        |
52/300 [00:02<00:25,  9.86it/s, now=None]t:  18%|█        | 54/300 [00:03<00:28,  8.52it/s, now=None]t:  19%|█        | 56/300 [00:03<00:27,  8.95it/s, now=None]t:  19%|█        | 58
/300 [00:03<00:28,  8.51it/s, now=None]t:  20%|██       | 60/300 [00:03<00:25,  9.44it/s, now=None]t:  21%|██       | 62/300 [00:04<00:40,  5.94it/s, now=None]t:  21%|██       | 63/
300 [00:04<00:39,  5.99it/s, now=None]t:  21%|██      | 64/300 [00:04<00:40,  5.79it/s, now=None]t:  22%|██      | 65/300 [00:04<00:47,  4.96it/s, now=None]t:  22%|██      | 66/30
0 [00:05<00:55,  4.25it/s, now=None]t:  22%|██     | 67/300 [00:05<00:58,  3.96it/s, now=None]t:  23%|██     | 68/300 [00:06<01:06,  3.47it/s, now=None]t:  23%|██     | 69/300
[00:06<01:14,  3.10it/s, now=None]t:  23%|██    | 70/300 [00:06<01:17,  2.97it/s, now=None]t:  24%|██    | 71/300 [00:07<01:18,  2.91it/s, now=None]t:  24%|██    | 72/300 [0
0:07<01:18,  2.90it/s, now=None]t:  24%|██   | 73/300 [00:07<01:26,  2.62it/s, now=None]t:  25%|██   | 74/300 [00:08<01:31,  2.46it/s, now=None]t:  25%|██   | 75/300 [00:
08<01:33,  2.42it/s, now=None]t:  25%|██   | 76/300 [00:09<01:36,  2.32it/s, now=None]t:  26%|██   | 77/300 [00:09<01:42,  2.19it/s, now=None]t:  26%|██   | 78/300 [00:1
0<01:50,  2.02it/s, now=None]t:  26%|██  | 79/300 [00:11<02:00,  1.83it/s, now=None]t:  27%|██  | 80/300 [00:11<02:09,  1.70it/s, now=None]t:  27%|██  | 81/300 [00:12<
02:18,  1.58it/s, now=None]t:  27%|██  | 82/300 [00:13<02:25,  1.49it/s, now=None]t:  28%|██  | 83/300 [00:14<02:33,  1.41it/s, now=None]t:  28%|██  | 84/300 [00:14<02
:42,  1.33it/s, now=None]t:  28%|██  | 85/300 [00:15<02:55,  1.23it/s, now=None]t:  29%|██  | 86/300 [00:16<03:09,  1.13it/s, now=None]t:  29%|██  | 87/300 [00:18<03:2
1,  1.06it/s, now=None]t:  29%|██  | 88/300 [00:19<03:25,  1.03it/s, now=None]t:  30%|██  | 89/300 [00:20<03:37,  1.03s/it, now=None]t:  30%|██  | 90/300 [00:21<03:52,
 1.11s/it, now=None]t:  30%|███  | 91/300 [00:22<04:05,  1.17s/it, now=None]t:  31%|███  | 92/300 [00:24<04:09,  1.20s/it, now=None]t:  31%|███  | 93/300 [00:25<04:22,
1.27s/it, now=None]t:  31%|███ | 94/300 [00:27<04:59,  1.45s/it, now=None]t:  32%|███ | 95/300 [00:29<05:33,  1.63s/it, now=None]t:  32%|███ | 96/300 [00:30<05:23,  1.
59s/it, now=None]t:  32%|███ | 97/300 [00:32<05:13,  1.54s/it, now=None]t:  33%|███ | 98/300 [00:33<05:06,  1.52s/it, now=None]t:  33%|███ | 99/300 [00:35<05:05,  1.52
s/it, now=None]t:  33%|███ | 100/300 [00:36<04:59,  1.50s/it, now=None]t:  34%|███ | 101/300 [00:38<04:53,  1.48s/it, now=None]t:  34%|███ | 102/300 [00:39<04:49,  1.4
6s/it, now=None]t:  34%|███ | 103/300 [00:41<05:08,  1.56s/it, now=None]t:  35%|███ | 104/300 [00:43<05:14,  1.61s/it, now=None]t:  35%|███ | 105/300 [00:44<05:12,  1.
60s/it, now=None]t:  35%|███ | 106/300 [00:46<05:26,  1.68s/it, now=None]t:  36%|███ | 107/300 [00:48<05:35,  1.74s/it, now=None]t:  36%|███ | 108/300 [00:50<06:02,
1.89s/it, now=None]t:  36%|███  | 109/300 [00:52<06:13,  1.96s/it, now=None]t:  37%|███  | 110/300 [00:55<06:38,  2.10s/it, now=None]t:  37%|███  | 111/300 [00:57<06:45,
2.15s/it, now=None]t:  37%|███  | 112/300 [01:00<07:09,  2.28s/it, now=None]t:  38%|███  | 113/300 [01:02<07:12,  2.31s/it, now=None]t:  38%|███  | 114/300 [01:04<07:06,
2.29s/it, now=None]t:  38%|███  | 115/300 [01:07<07:12,  2.34s/it, now=None]t:  39%|███  | 116/300 [01:09<07:12,  2.35s/it, now=None]t:  39%|███  | 117/300 [01:11<07:05,
2.33s/it, now=None]t:  39%|███  | 118/300 [01:14<06:59,  2.30s/it, now=None]t:  40%|███  | 119/300 [01:16<06:59,  2.32s/it, now=None]t:  40%|███  | 120/300 [01:18<07:02,
2.34s/it, now=None]t:  40%|███  | 121/300 [01:21<07:21,  2.47s/it, now=None]t:  41%|███  | 122/300 [01:24<07:50,  2.64s/it, now=None]t:  41%|███  | 123/300 [01:27<07:50,
```

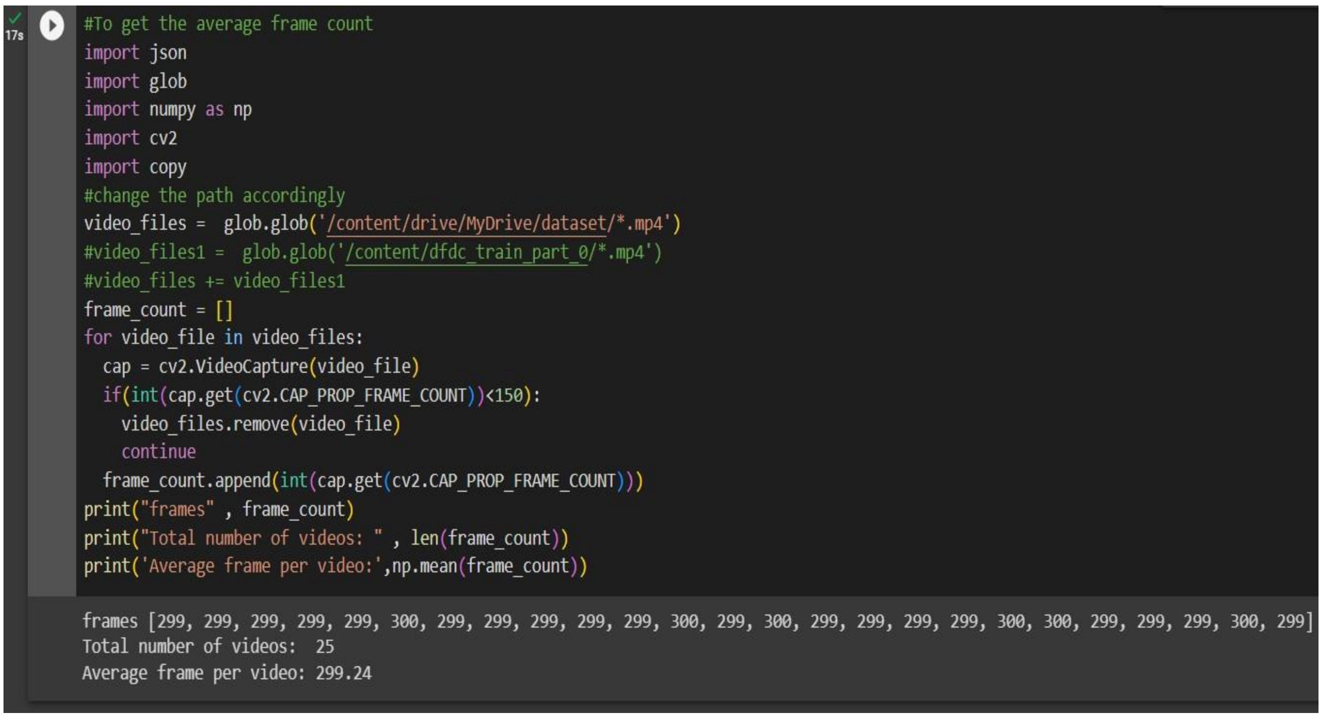
Fig 9.1.2: Process while the code is executing.

FROM HERE WE WILL USE GOOGLE COLLAB TO PROCEED:



```
42s  [1]  #Mount our google drive
          from google.colab import drive
          drive.mount('/content/drive')

          Mounted at /content/drive
```

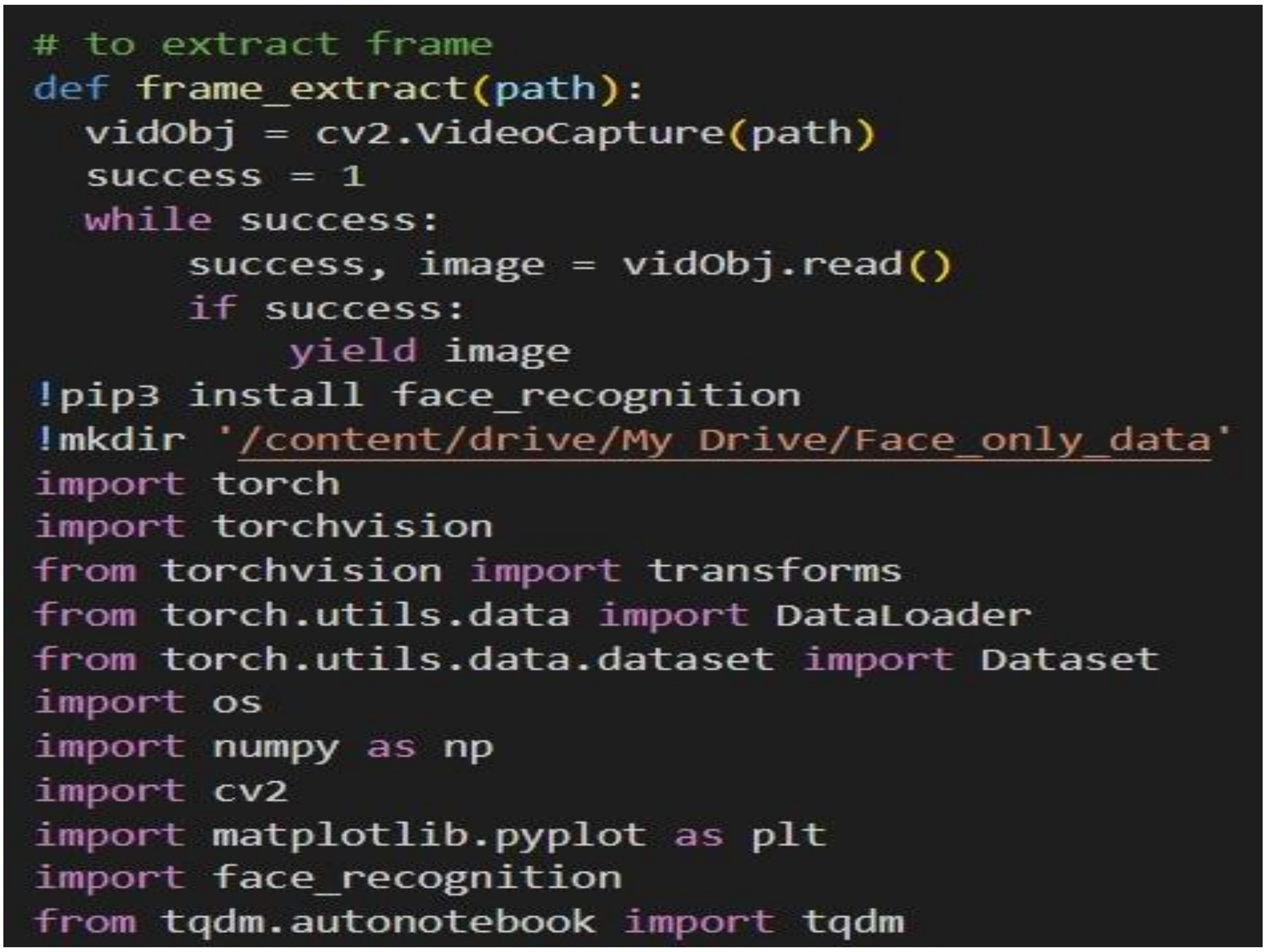
Fig 9.1.3: To mount our google drive.



```
#To get the average frame count
import json
import glob
import numpy as np
import cv2
import copy
#change the path accordingly
video_files =  glob.glob('/content/drive/MyDrive/dataset/*.mp4')
#video_files1 =  glob.glob('/content/dfdc_train_part_0/*.mp4')
#video_files += video_files1
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))
```

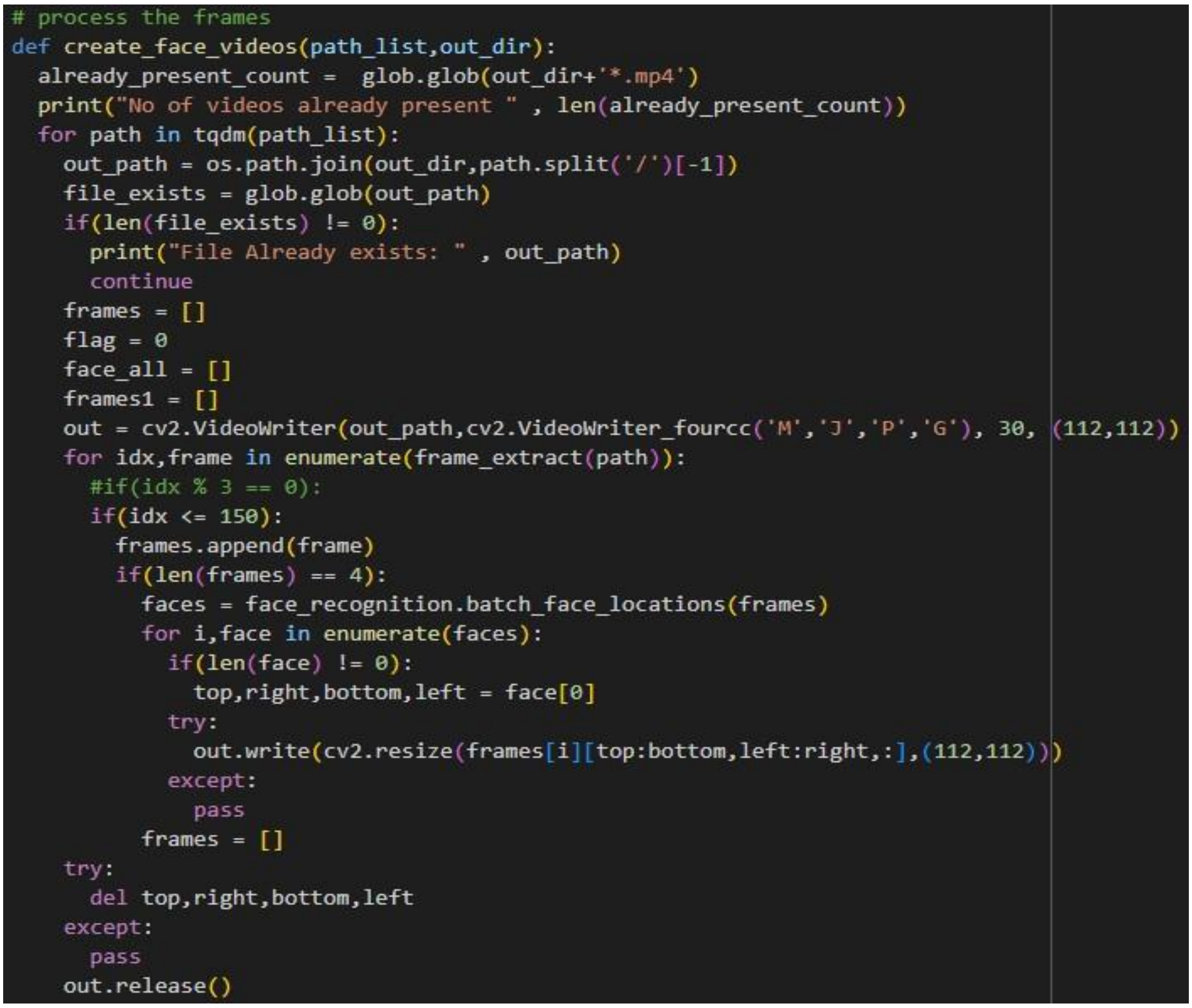
```
frames [299, 299, 299, 299, 299, 300, 299, 299, 299, 299, 299, 300, 299, 300, 299, 299, 299, 299, 300, 300, 299, 299, 299, 300, 299]
Total number of videos:  25
Average frame per video: 299.24
```

Fig 9.1.4: To get the average frame count.



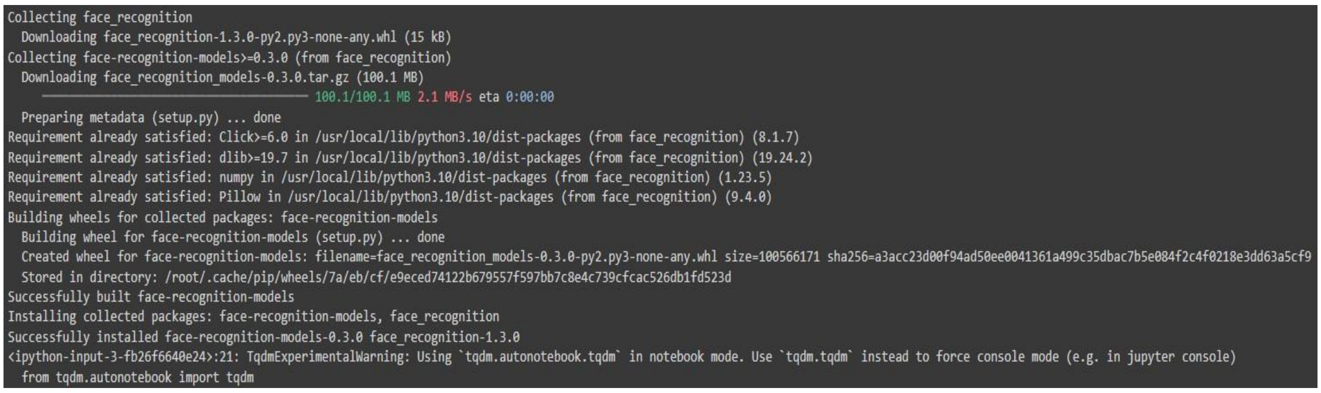
```
# to extract frame
def frame_extract(path):
  vidObj = cv2.VideoCapture(path)
  success = 1
  while success:
      success, image = vidObj.read()
      if success:
          yield image
!pip3 install face_recognition
!mkdir '/content/drive/My Drive/Face_only_data'
import torch
import torchvision
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
from tqdm.autonotebook import tqdm
```

Fig 9.1.5: Code to extract frame.



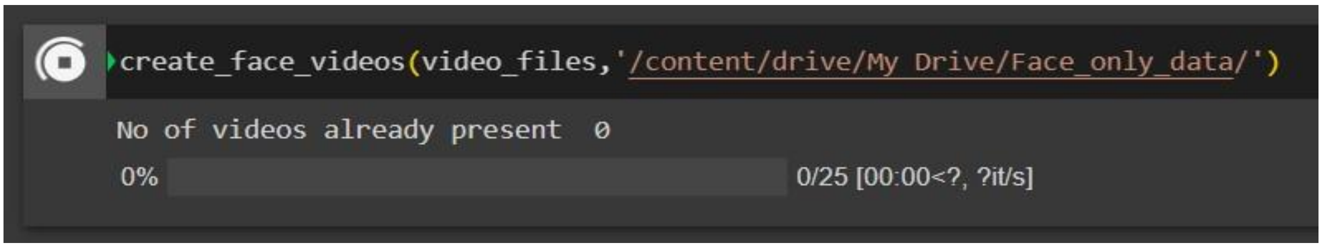
```python
# process the frames
def create_face_videos(path_list,out_dir):
  already_present_count =  glob.glob(out_dir+'*.mp4')
  print("No of videos already present " , len(already_present_count))
  for path in tqdm(path_list):
    out_path = os.path.join(out_dir,path.split('/')[-1])
    file_exists = glob.glob(out_path)
    if(len(file_exists) != 0):
      print("File Already exists: " , out_path)
      continue
    frames = []
    flag = 0
    face_all = []
    frames1 = []
    out = cv2.VideoWriter(out_path,cv2.VideoWriter_fourcc('M','J','P','G'), 30, (112,112))
    for idx,frame in enumerate(frame_extract(path)):
      #if(idx % 3 == 0):
      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 = []
    try:
      del top,right,bottom,left
    except:
      pass
    out.release()
```

Fig 9.1.6: Code to process the frames.



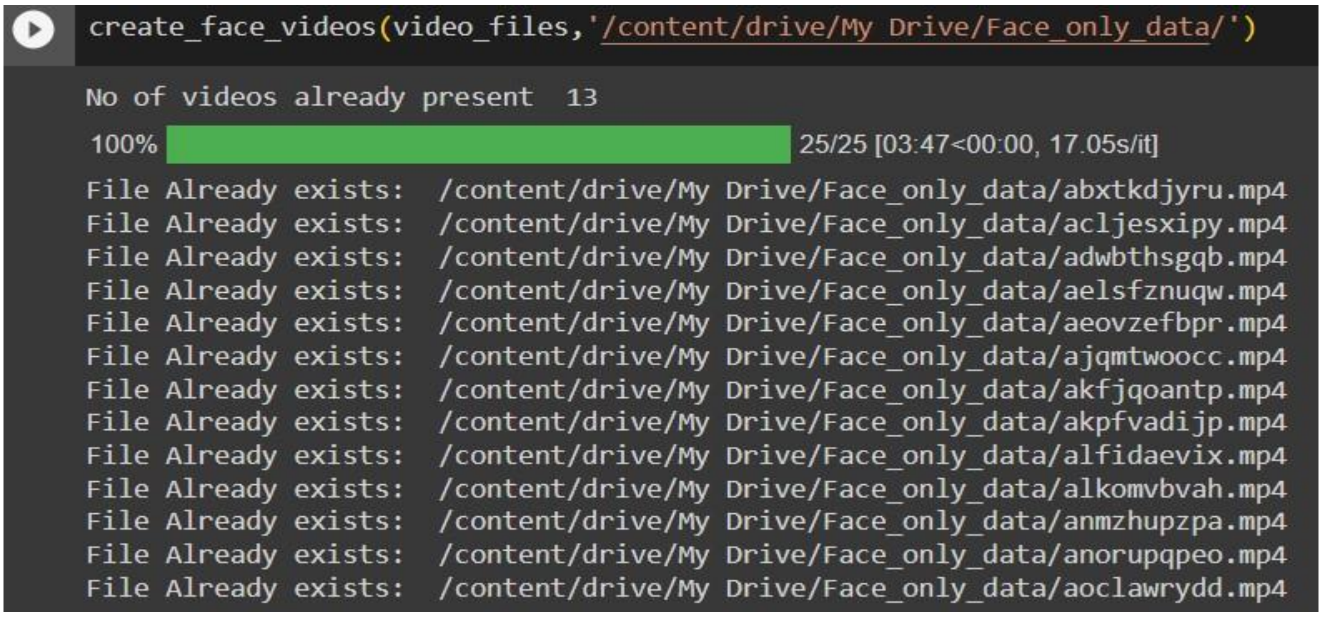
```
Collecting face_recognition
  Downloading face_recognition-1.3.0-py2.py3-none-any.whl (15 kB)
Collecting face-recognition-models>=0.3.0 (from face_recognition)
  Downloading face_recognition_models-0.3.0.tar.gz (100.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.1/100.1 MB 2.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.10/dist-packages (from face_recognition) (8.1.7)
Requirement already satisfied: dlib>=19.7 in /usr/local/lib/python3.10/dist-packages (from face_recognition) (19.24.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from face_recognition) (1.23.5)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from face_recognition) (9.4.0)
Building wheels for collected packages: face-recognition-models
  Building wheel for face-recognition-models (setup.py) ... done
  Created wheel for face-recognition-models: filename=face_recognition_models-0.3.0-py2.py3-none-any.whl size=100566171 sha256=a3acc23d00f94ad50ee0041361a499c35dbac7b5e084f2c4f0218e3dd63a5cf9
  Stored in directory: /root/.cache/pip/wheels/7a/eb/cf/e9eced74122b679557f597bb7c8e4c739cfcac526db1fd523d
Successfully built face-recognition-models
Installing collected packages: face-recognition-models, face_recognition
Successfully installed face-recognition-models-0.3.0 face_recognition-1.3.0
<ipython-input-3-fb26f6640e24>:21: TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to force console mode (e.g. in jupyter console)
  from tqdm.autonotebook import tqdm
```

Fig 9.1.7: Installing required packages for processing the frames.



```
create_face_videos(video_files,'/content/drive/My Drive/Face_only_data/')

No of videos already present  0
0%                                              0/25 [00:00<?, ?it/s]
```

Fig 9.1.8: Code to create videos from the extracted frames and save in given path.

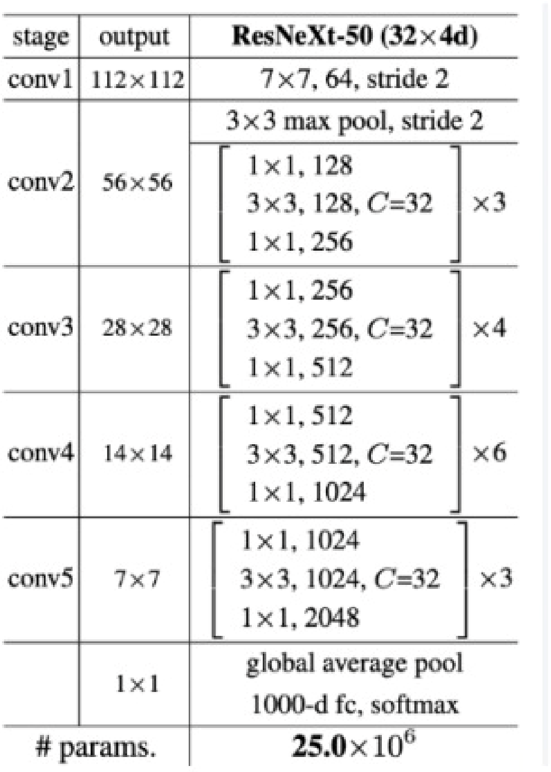


```
create_face_videos(video_files,'/content/drive/My Drive/Face_only_data/')

No of videos already present  13
100%                                            25/25 [03:47<00:00, 17.05s/it]
File Already exists:  /content/drive/My Drive/Face_only_data/abxtkdjyru.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/acljesxipy.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/adwbthsgqb.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/aelsfznuqw.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/aeovzefbpr.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/ajqmtwoocc.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/akfjqoantp.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/akpfvadijp.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/alfidaevix.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/alkomvbvah.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/anmzhupzpa.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/anorupqpeo.mp4
File Already exists:  /content/drive/My Drive/Face_only_data/aoclawrydd.mp4
```

Fig 9.1.9: After completion of saving videos in the given path.

MODEL DETAILS:

ResNext CNN : A previously trained version of RCNN has been used. The model’s name is resnext50_32x4d()[22]. This model has 50 layers and 32 x 4 dimensions. The figure displays a comprehensive implementation of the model.



stage	output	ResNeXt-50 (32×4d)
conv1	112×112	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2
		$\left[\begin{array}{l}1\times1, 128\\ 3\times3, 128, C{=}32\\ 1\times1, 256\end{array}\right]\times3$
conv3	28×28	$\left[\begin{array}{l}1\times1, 256\\ 3\times3, 256, C{=}32\\ 1\times1, 512\end{array}\right]\times4$
conv4	14×14	$\left[\begin{array}{l}1\times1, 512\\ 3\times3, 512, C{=}32\\ 1\times1, 1024\end{array}\right]\times6$
conv5	7×7	$\left[\begin{array}{l}1\times1, 1024\\ 3\times3, 1024, C{=}32\\ 1\times1, 2048\end{array}\right]\times3$
	1×1	global average pool 1000-d fc, softmax
# params.		**25.0**×10^6

Diagram: Res-Next Architecture

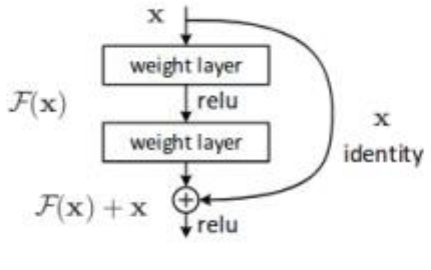


Diagram: Res-Next functioning

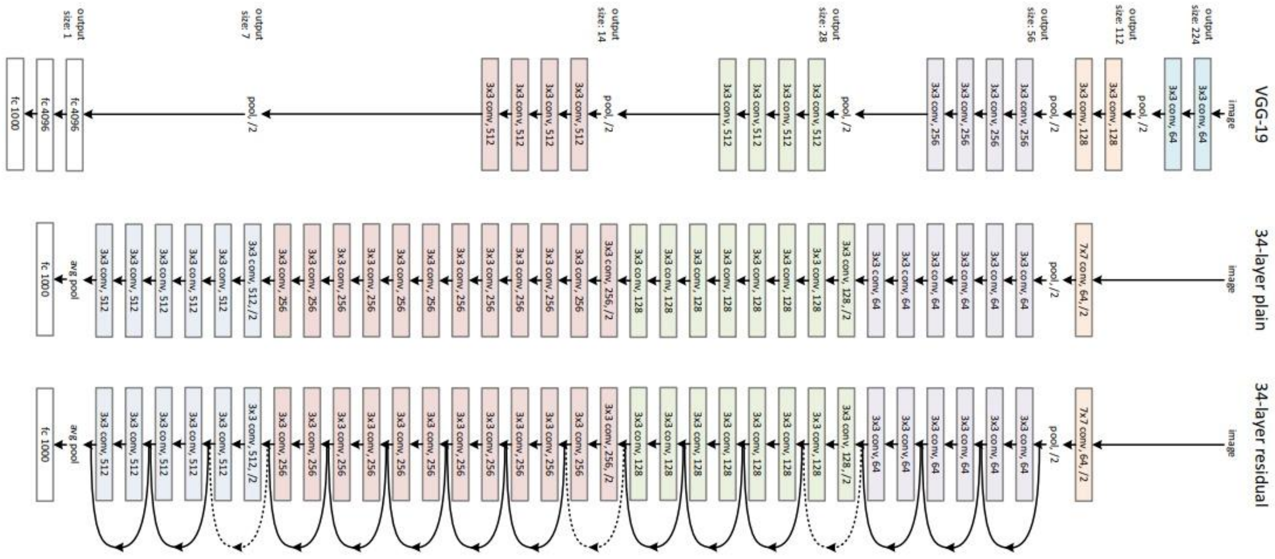


Diagram: Overview of Res-Next Architecture

Sequential Layer : Sequential serves as a container for Modules, enabling their stacking and simultaneous execution. Within this structure, Sequential layer organizes the feature vector from the ResNext model in a sequential manner, facilitating its consecutive transmission to the LSTM.

LSTM Layer : The LSTM layer is utilized for sequence analysis, enabling the detection of temporal changes between frames. It operates on 2048-dimensional feature vectors as input. Our approach involves a single LSTM layer with 2048 latent dimensions, 2048 hidden units, and a dropout probability of 0.4. By comparing the frame at time "t" with the frame at time "t-n", where 'n' can be any number of frames preceding 't', the LSTM sequentially analyzes frames to provide temporal insights into the video.

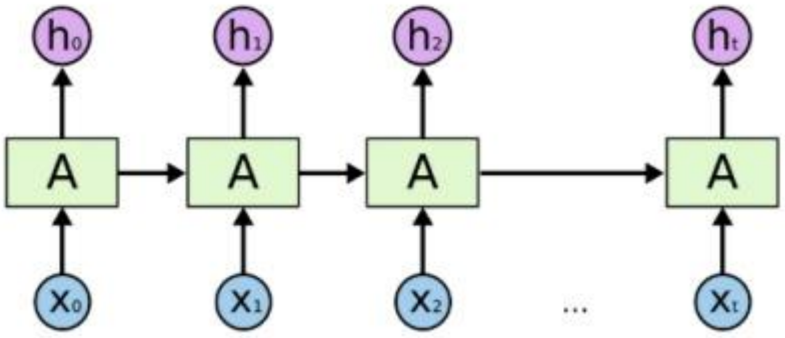


Diagram: Overview of LSTM Architecture

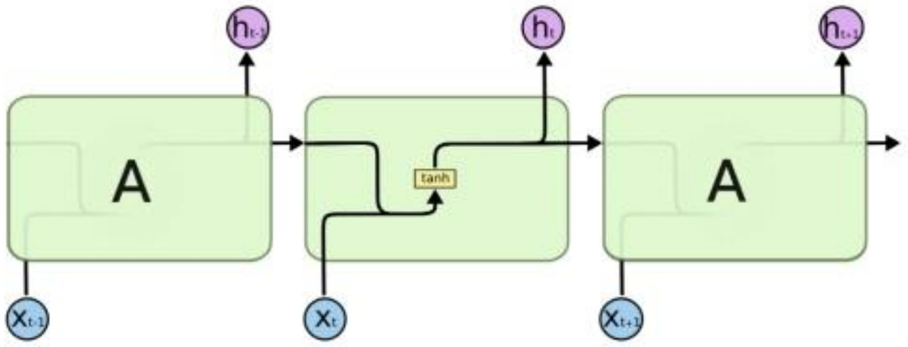


Diagram: Internal LSTM Architecture

ReLU: ReLU, short for Rectified Linear Unit, is an activation function commonly used in neural networks. It produces an output of 0 when the input is less than 0, and otherwise outputs the input value directly. This means that if the input is greater than 0, the output is identical to the input. ReLU operates similarly to biological neurons, making it a popular choice in deep learning. It is non-linear and avoids issues with back propagation that can occur with functions like sigmoid. Moreover, when constructing large neural networks, ReLU facilitates rapid model development.

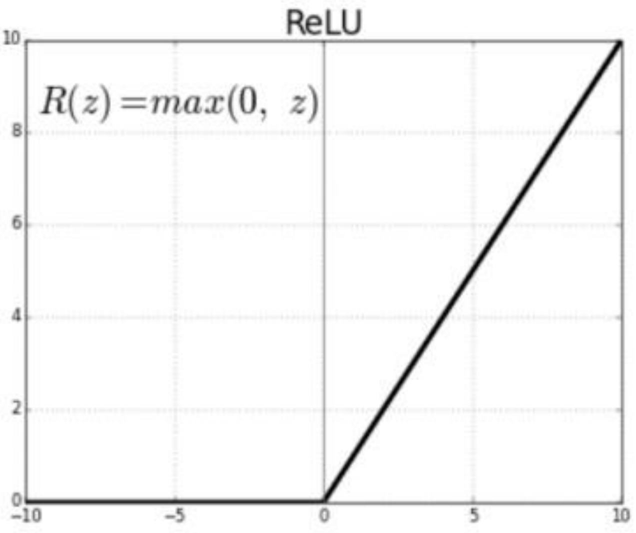


Diagram: ReLU activation function.

Dropout Layer: The Dropout layer, set with a dropout rate of 0.4, serves to mitigate overfitting within the model. This technique assists in enhancing the model's ability to generalize by randomly deactivating the output of individual neurons, effectively setting them to zero during training. This process introduces a form of redundancy, as neighboring neurons must compensate for the dropped ones, thus making the model more robust. Consequently, during the backpropagation process, the cost function becomes more responsive to the influence of neighboring neurons, influencing how the model's weights are adjusted.

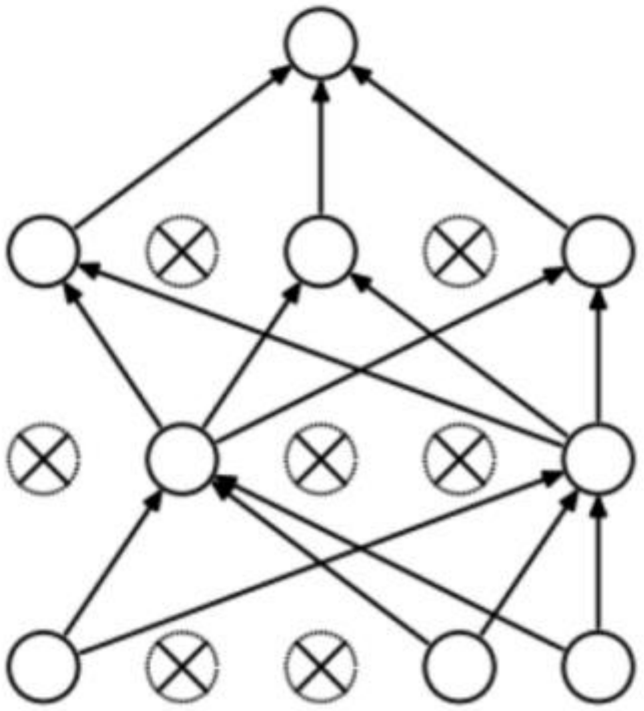


Diagram: Dropout layer overview.

Adaptive Average Pooling Layer: It is employed to limit variation, reduce computation cost and retrieve low level features from neighbourhood.2 dimensional Adaptive Average Pooling Layer is employed in the model.

TRAINED MODEL DETAILS:

• **Train Test Split:** The dataset is separated between train and test subsets, containing 4,200 train videos and 1,800 test movies, respectively. The training and examination split is balanced, meaning that each split contains 50% authentic videos and 50% fake videos.

• **Data Loader:** It loads the videos with a set size of (4) together with their labels.

• **Training:** During the training phase, the Adam optimizer is employed for a total of 20 epochs. The learning rate for the optimizer is set at 1e-5 (0.00001), indicating the rate at which the model adjusts its parameters based on the gradient of the loss function. Additionally, a weight decay rate of 1e-3 (0.001) is applied, which helps prevent overfitting by penalizing large weights in the model.

• **Adam optimizer:** To allow the flexible learning rate Adam optimizer with the model parameters is employed.

• **Cross Entropy:** To determine the loss function the cross entropy approach is utilized since the problem we are training is one of categorization.

• **SoftMax Layer:** Squashing functions are similar to SoftMax functions. Squashing functions limit the function's output to the interval between 0 and 1. This makes it possible to read the result as a probability straight away. SoftMax functions are also multi-class sigmoid, which means they may be used to estimate several classes' probabilities simultaneously. A SoftMax layer is frequently the last layer used in neural networks since its outputs may be thought of as probabilities (i.e., they must total to 1). In this case, the SoftMax layer has two output nodes: REAL and FAKE. It also gives us the prediction's confidence, or probability.

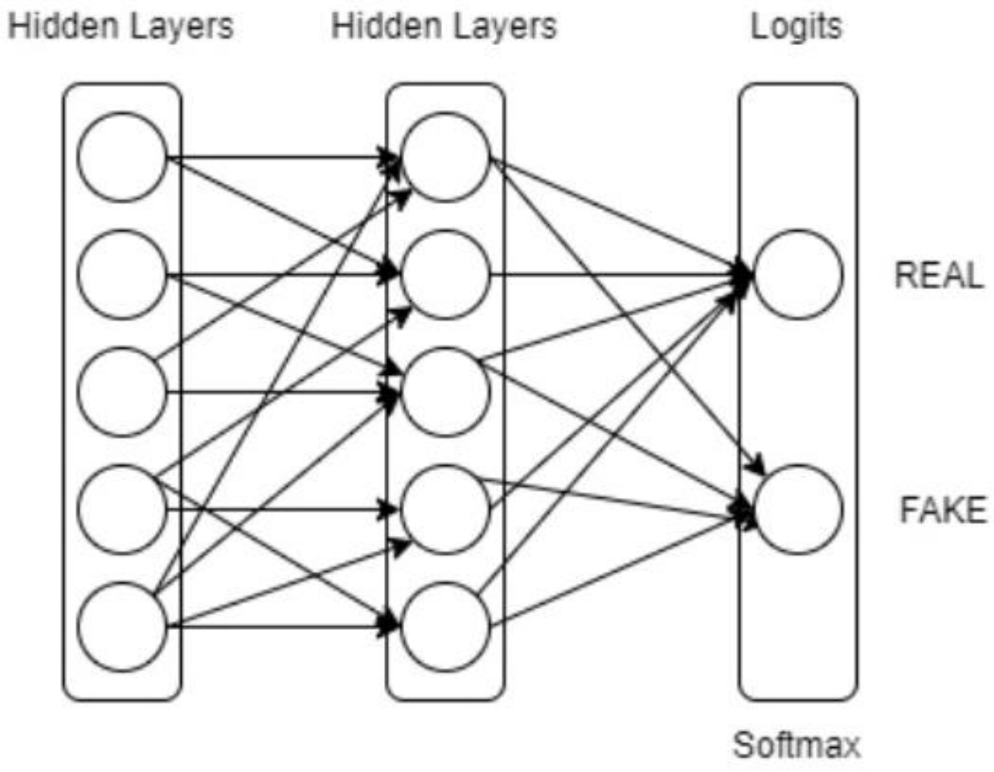


Diagram: SoftMax Layer

Confusion Matrix:

A confusion matrix is a table commonly utilized to assess the classification model's performance. It offers visualization of the performance of an algorithm by giving a breakdown of the predictions versus the actual outcomes. The confusion matrix is particularly useful for examining the performance of models in binary classification tasks, however it can also be applied to multi-class classification issues.

Here are the components of a confusion matrix:

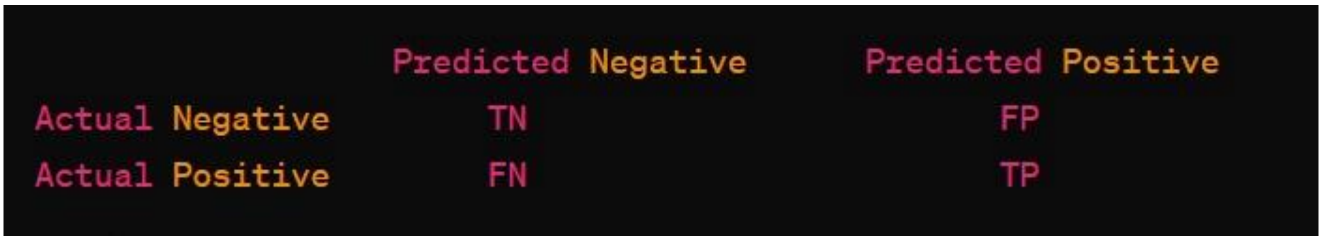
True Positives (TP): These are the occurrences where the model successfully predicted the positive class (e.g., accurately identified instances of a specific class as belonging to that class).

True Negatives (TN): These are the occurrences when the model successfully predicted the negative class (e.g., accurately classified instances of not belonging to a given class as not belonging to that class).

False Positives (FP): Also known as Type I mistakes, these are the occurrences when the model mistakenly predicted the positive class when it was actually negative (e.g., falsely classifying instances as belonging to a given class when they do not).

False Negatives (FN): Also known as Type II errors, these are the occurrences when the model mistakenly predicted the negative class when it was actually positive (e.g., failing to recognise instances as belonging to a given class when they do).

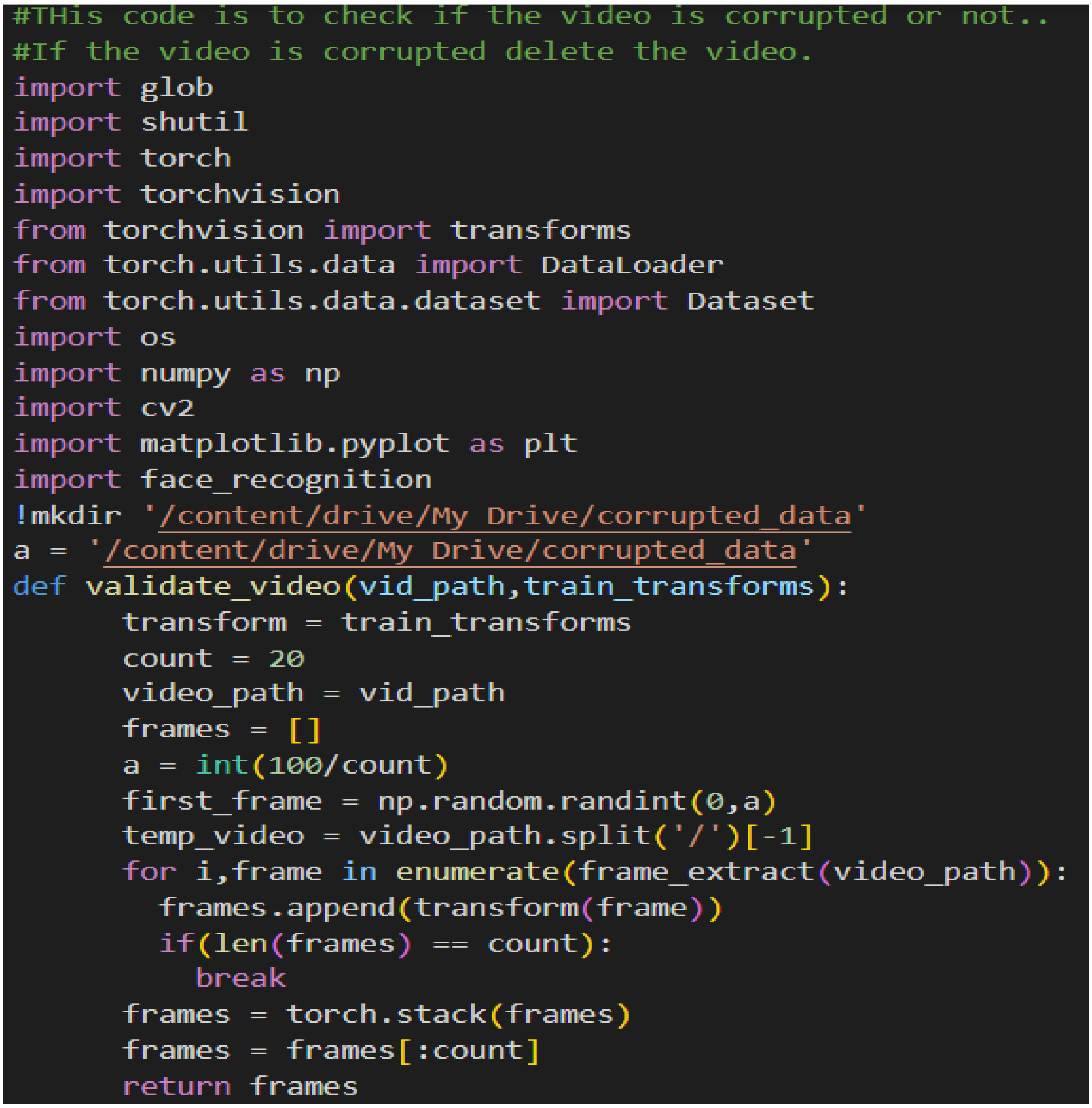
The confusion matrix is often arranged in a table format like this:



```
                   Predicted Negative      Predicted Positive
Actual Negative          TN                       FP
Actual Positive          FN                       TP
```

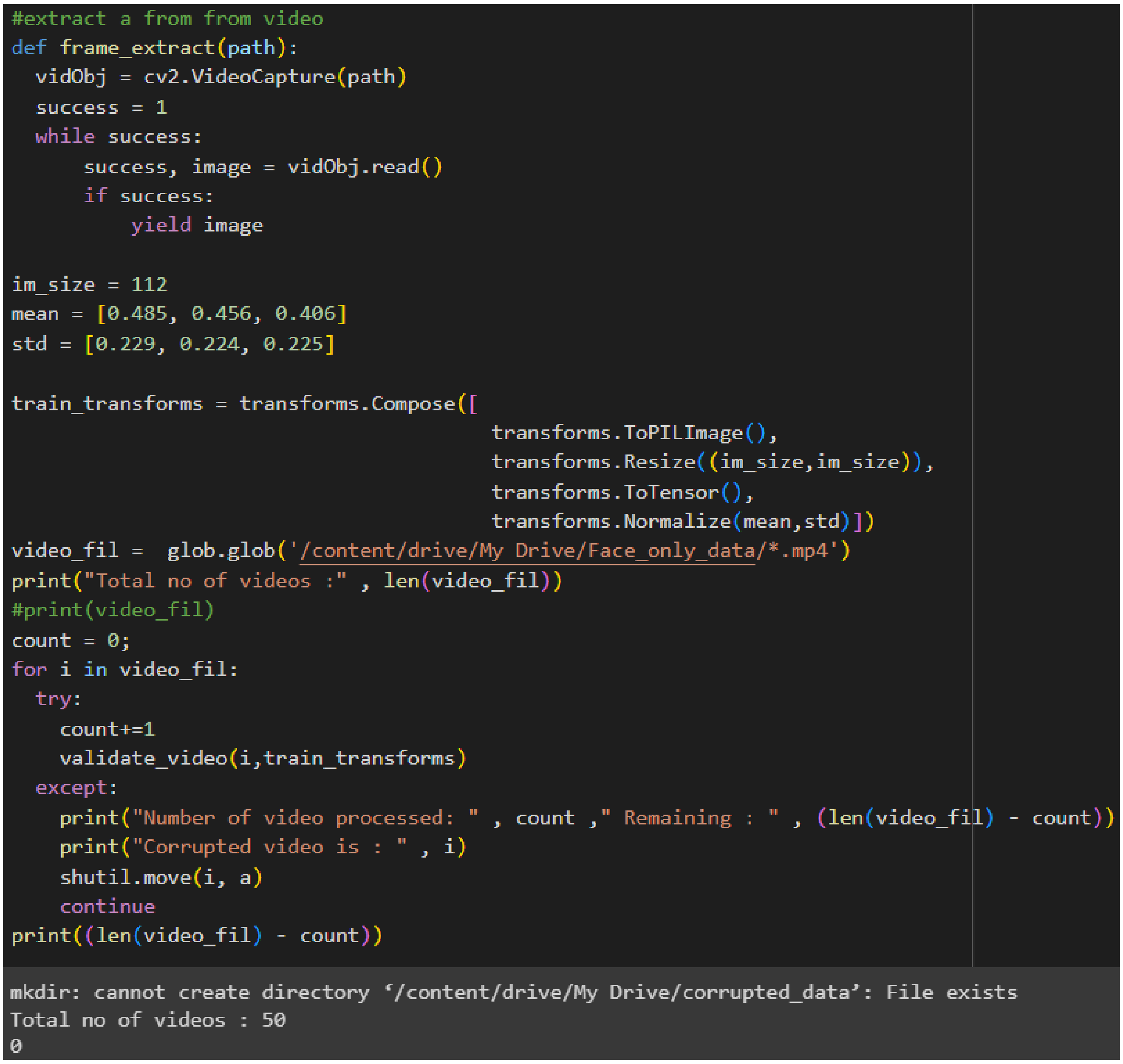
These metrics provide insights into multiple aspects of the model's performance, such as its capacity to produce right predictions, its ability to avoid false positives, and its ability to capture all positive cases.

MODEL CREATION CODE STEPS:



```
#THis code is to check if the video is corrupted or not..
#If the video is corrupted delete the video.
import glob
import shutil
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
!mkdir '/content/drive/My Drive/corrupted_data'
a = '/content/drive/My Drive/corrupted_data'
def validate_video(vid_path,train_transforms):
      transform = train_transforms
      count = 20
      video_path = vid_path
      frames = []
      a = int(100/count)
      first_frame = np.random.randint(0,a)
      temp_video = video_path.split('/')[-1]
      for i,frame in enumerate(frame_extract(video_path)):
        frames.append(transform(frame))
        if(len(frames) == count):
          break
      frames = torch.stack(frames)
      frames = frames[:count]
      return frames
```

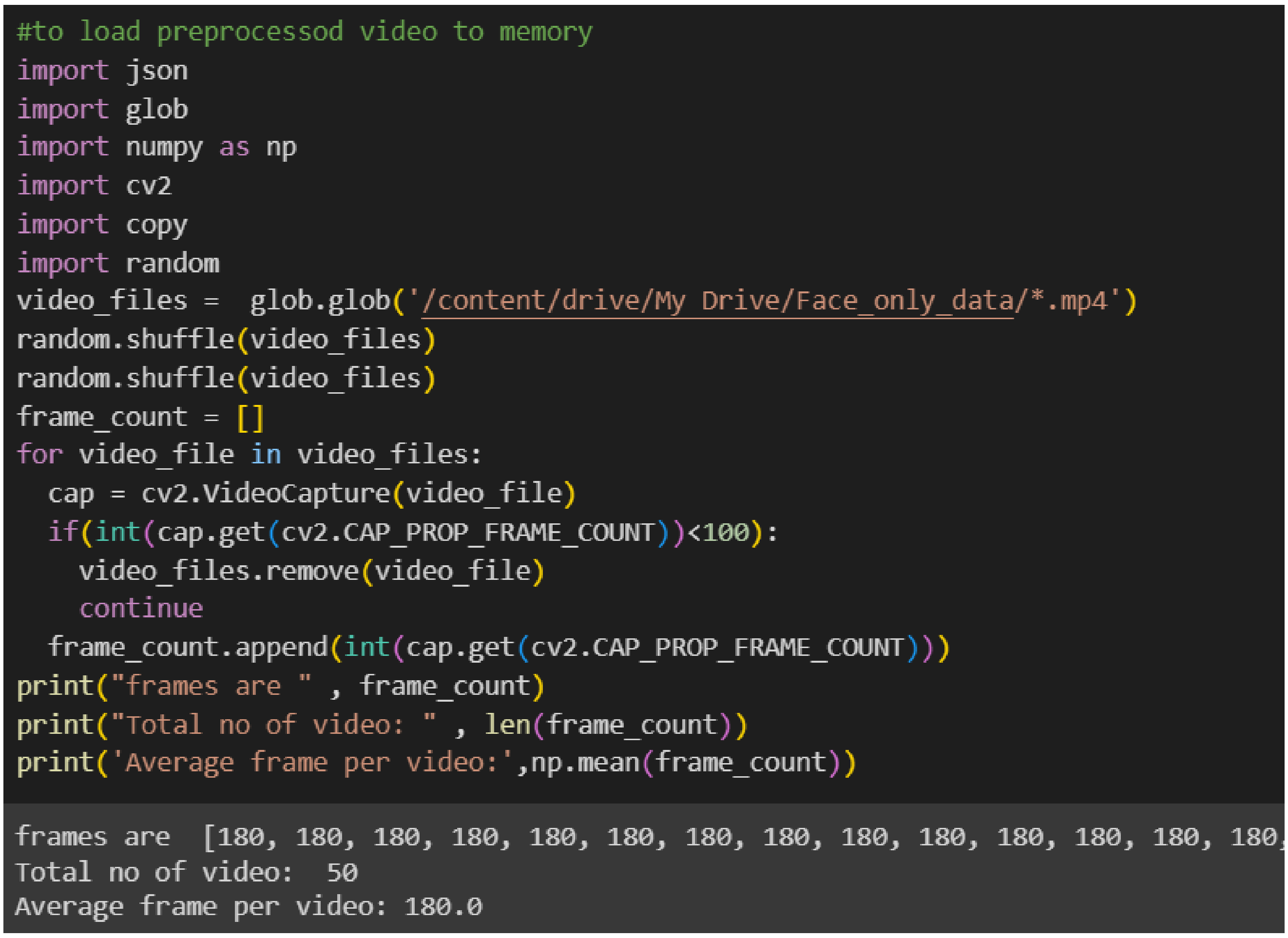
Fig 9.2.1: This code is to check whether the video is corrupted or not



```python
#extract a from from video
def frame_extract(path):
  vidObj = cv2.VideoCapture(path)
  success = 1
  while success:
      success, image = vidObj.read()
      if success:
          yield image

im_size = 112
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]

train_transforms = transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.Resize((im_size,im_size)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean,std)])
video_fil =  glob.glob('/content/drive/My Drive/Face_only_data/*.mp4')
print("Total no of videos :" , len(video_fil))
#print(video_fil)
count = 0;
for i in video_fil:
  try:
    count+=1
    validate_video(i,train_transforms)
  except:
    print("Number of video processed: " , count ," Remaining : " , (len(video_fil) - count))
    print("Corrupted video is : " , i)
    shutil.move(i, a)
    continue
print((len(video_fil) - count))
```

```
mkdir: cannot create directory '/content/drive/My Drive/corrupted_data': File exists
Total no of videos : 50
0
```

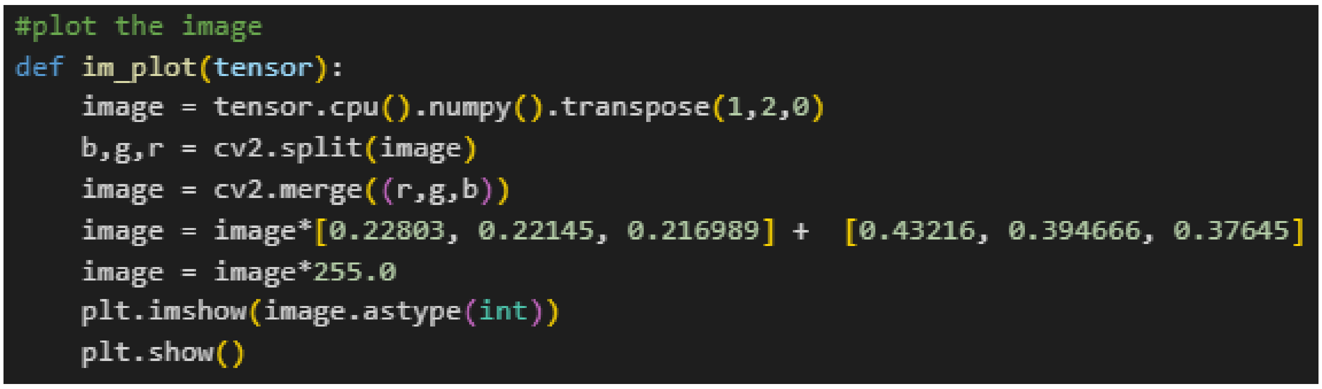
Fig 9.2.2: To extract a frame from video and to send corrupted video to corrupted folder



```
#to load preprocessod video to memory
import json
import glob
import numpy as np
import cv2
import copy
import random
video_files =  glob.glob('/content/drive/My Drive/Face_only_data/*.mp4')
random.shuffle(video_files)
random.shuffle(video_files)
frame_count = []
for video_file in video_files:
  cap = cv2.VideoCapture(video_file)
  if(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))<100):
    video_files.remove(video_file)
    continue
  frame_count.append(int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
print("frames are " , frame_count)
print("Total no of video: " , len(frame_count))
print('Average frame per video:',np.mean(frame_count))
```

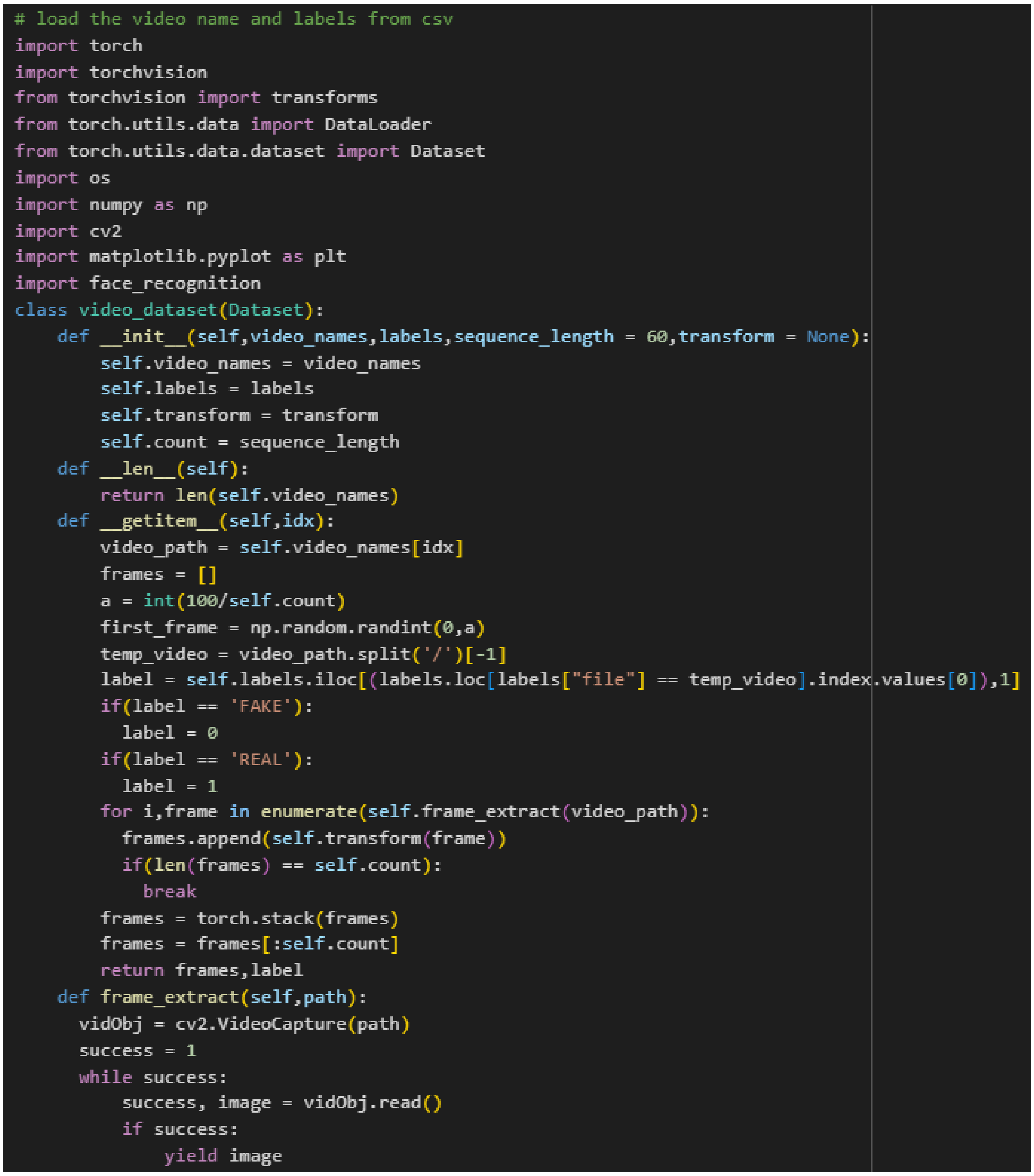
```
frames are  [180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180,
Total no of video:  50
Average frame per video: 180.0
```

Fig 9.2.3: To load pre-processed video to memory



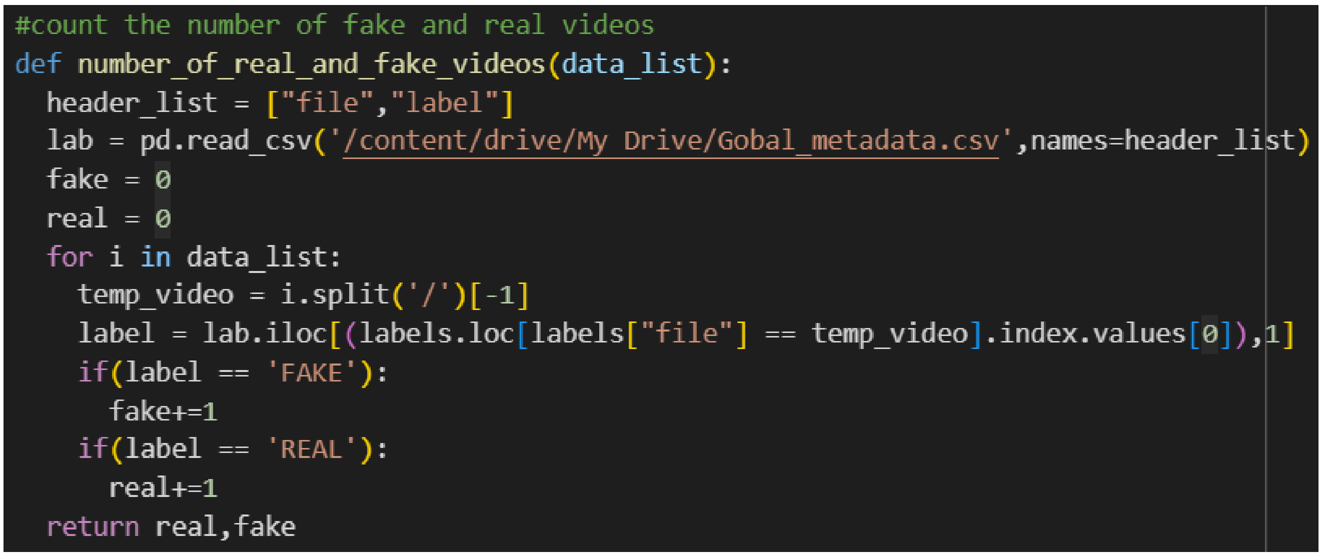
```
#plot the image
def im_plot(tensor):
    image = tensor.cpu().numpy().transpose(1,2,0)
    b,g,r = cv2.split(image)
    image = cv2.merge((r,g,b))
    image = image*[0.22803, 0.22145, 0.216989] +  [0.43216, 0.394666, 0.37645]
    image = image*255.0
    plt.imshow(image.astype(int))
    plt.show()
```

Fig 9.2.4: To plot the image



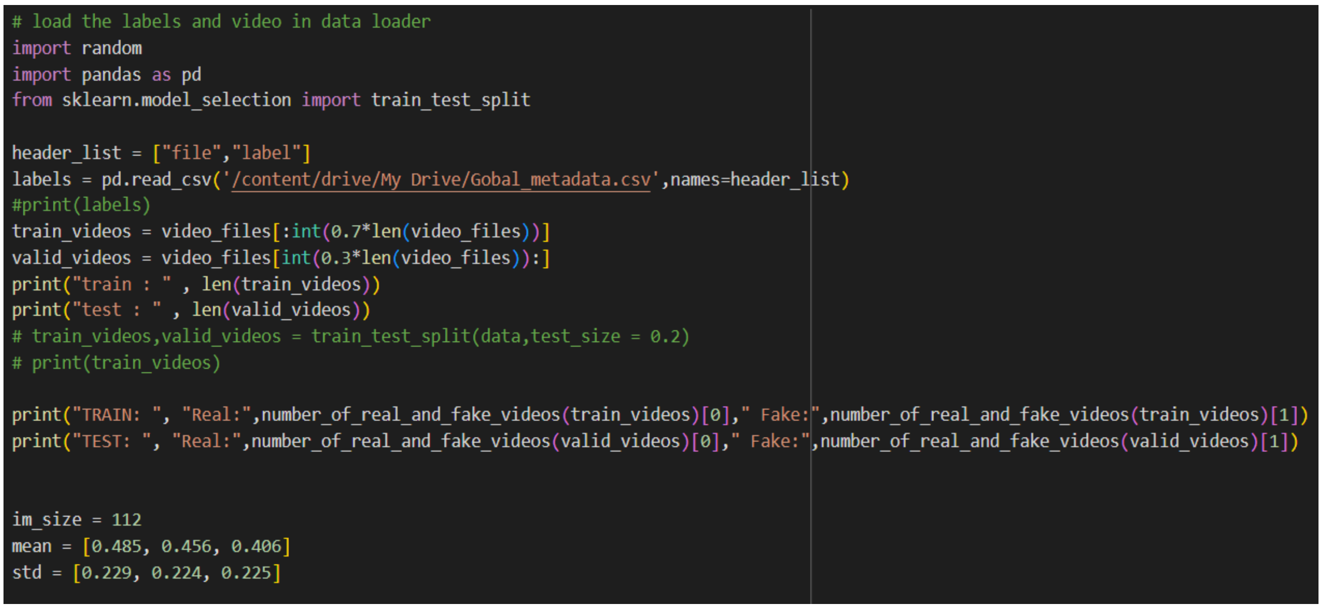
```python
# load the video name and labels from csv
import torch
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import face_recognition
class 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]
        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]
        return frames,label
    def frame_extract(self,path):
      vidObj = cv2.VideoCapture(path)
      success = 1
      while success:
          success, image = vidObj.read()
          if success:
              yield image
```

Fig 9.2.5: To load the video name and labels from csv file

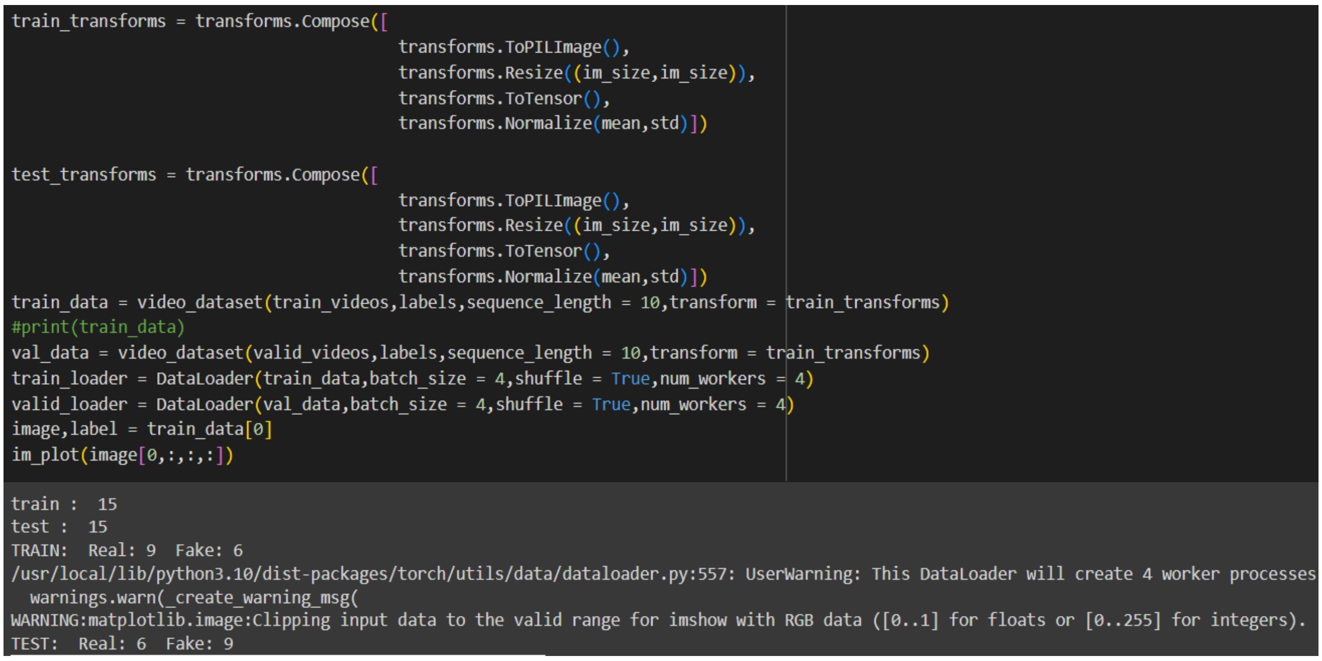


```python
#count the number of fake and real videos
def number_of_real_and_fake_videos(data_list):
  header_list = ["file","label"]
  lab = pd.read_csv('/content/drive/My Drive/Gobal_metadata.csv',names=header_list)
  fake = 0
  real = 0
  for i in data_list:
    temp_video = i.split('/')[-1]
    label = lab.iloc[(labels.loc[labels["file"] == temp_video].index.values[0]),1]
    if(label == 'FAKE'):
      fake+=1
    if(label == 'REAL'):
      real+=1
  return real,fake
```

Fig 9.2.6: To count the number of fake and real videos



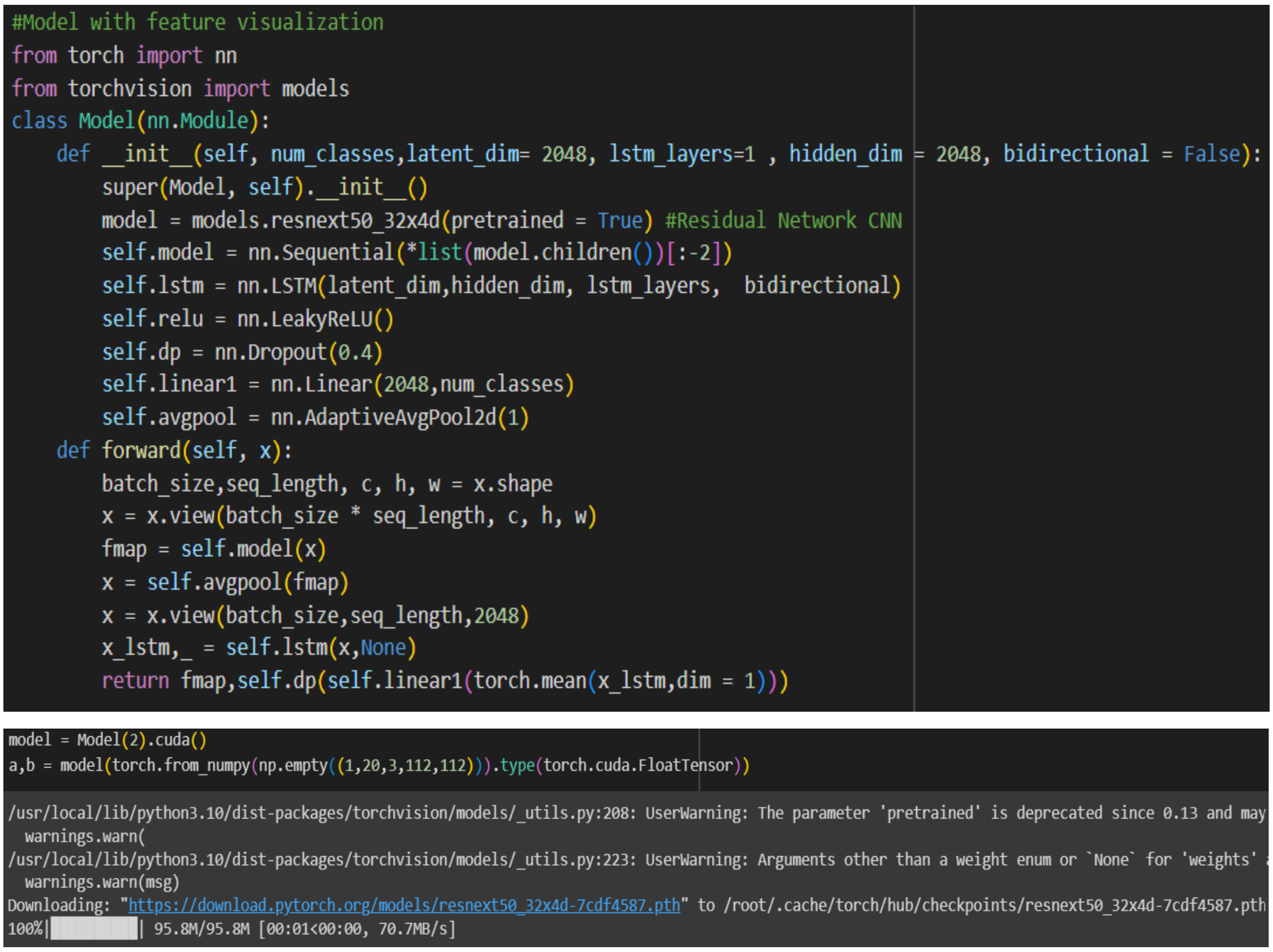
```python
# load the labels and video in data loader
import random
import pandas as pd
from sklearn.model_selection import train_test_split

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(0.7*len(video_files))]
valid_videos = video_files[int(0.3*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]
```



```python
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,:,:,:])
```

```
train :  15
test :  15
TRAIN:   Real: 9  Fake: 6
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes
  warnings.warn(_create_warning_msg(
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
TEST:   Real: 6   Fake: 9
```

Fig 9.2.7: To load the labels and video in data loader

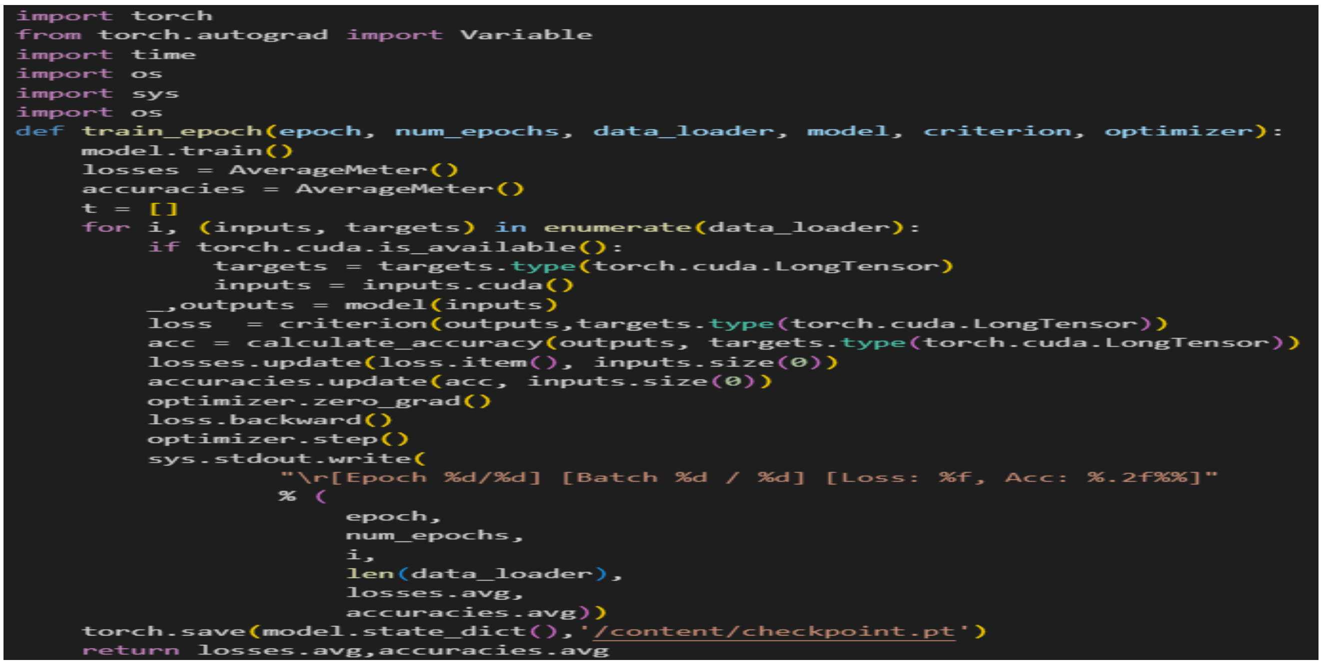


```
#Model with feature visualization
from torch import nn
from torchvision import models
class Model(nn.Module):
    def __init__(self, num_classes,latent_dim= 2048, lstm_layers=1 , hidden_dim = 2048, bidirectional = False):
        super(Model, self).__init__()
        model = models.resnext50_32x4d(pretrained = 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)))
```

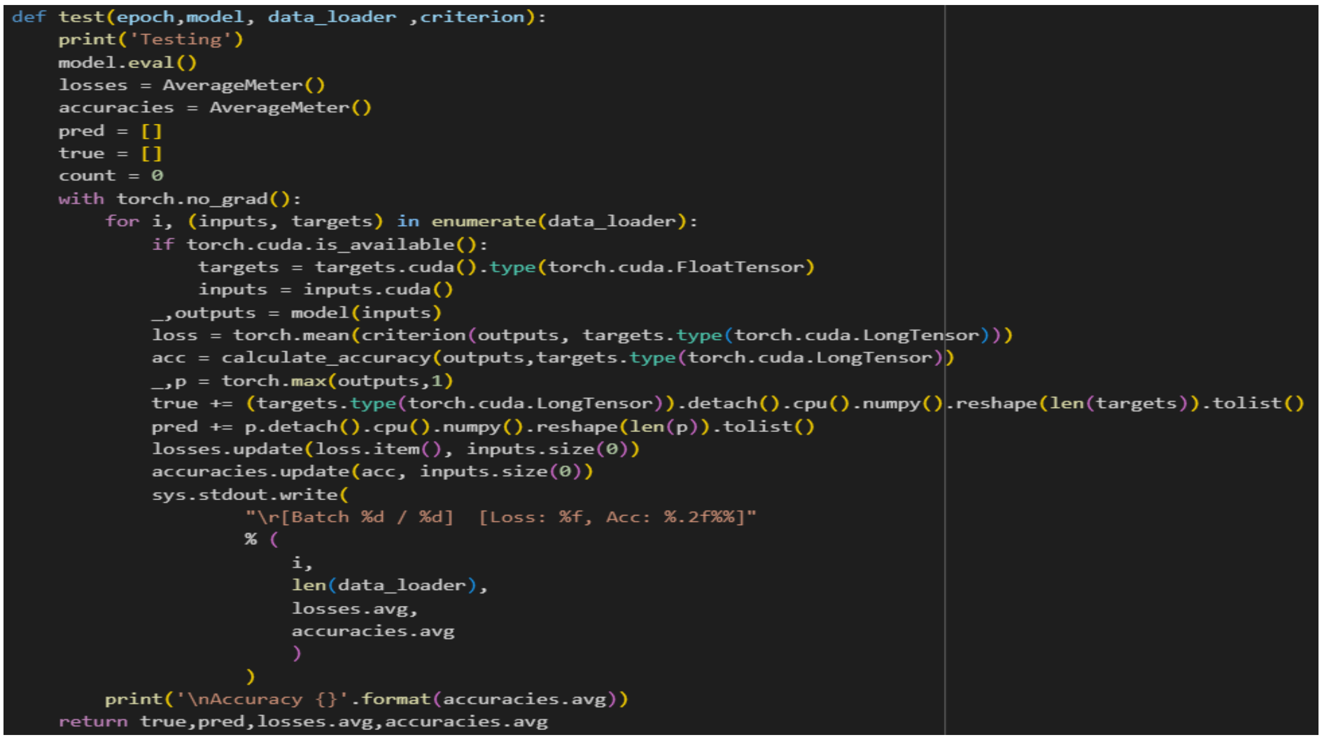
```
model = Model(2).cuda()
a,b = model(torch.from_numpy(np.empty((1,20,3,112,112))).type(torch.cuda.FloatTensor))
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights'
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnext50_32x4d-7cdf4587.pth" to /root/.cache/torch/hub/checkpoints/resnext50_32x4d-7cdf4587.pth
100%|██████████| 95.8M/95.8M [00:01<00:00, 70.7MB/s]
```

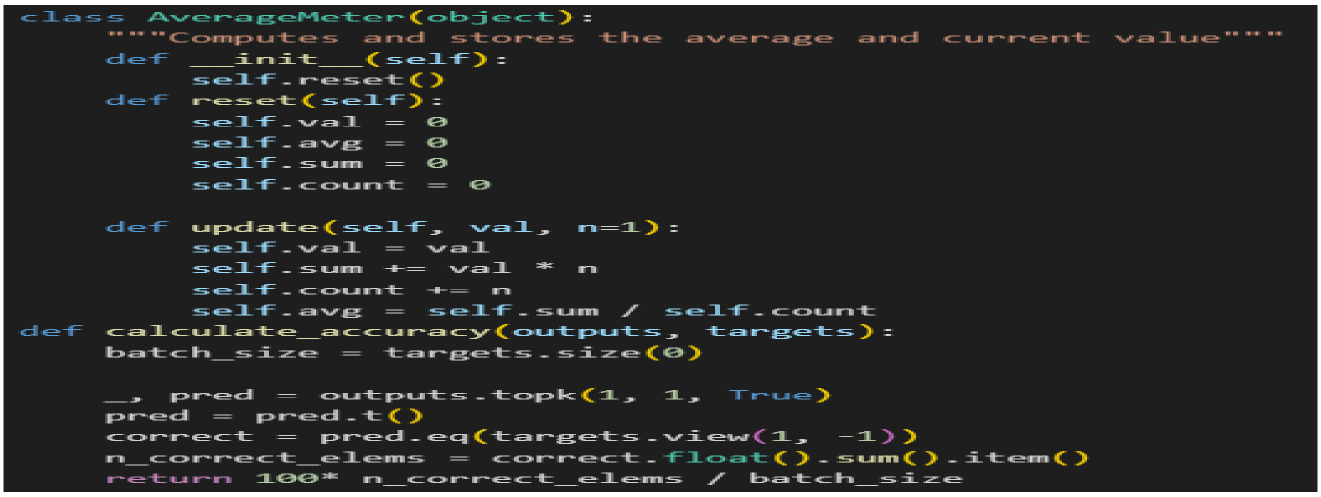
Fig 9.2.8: Code tells the model about feature visualization



```python
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
```

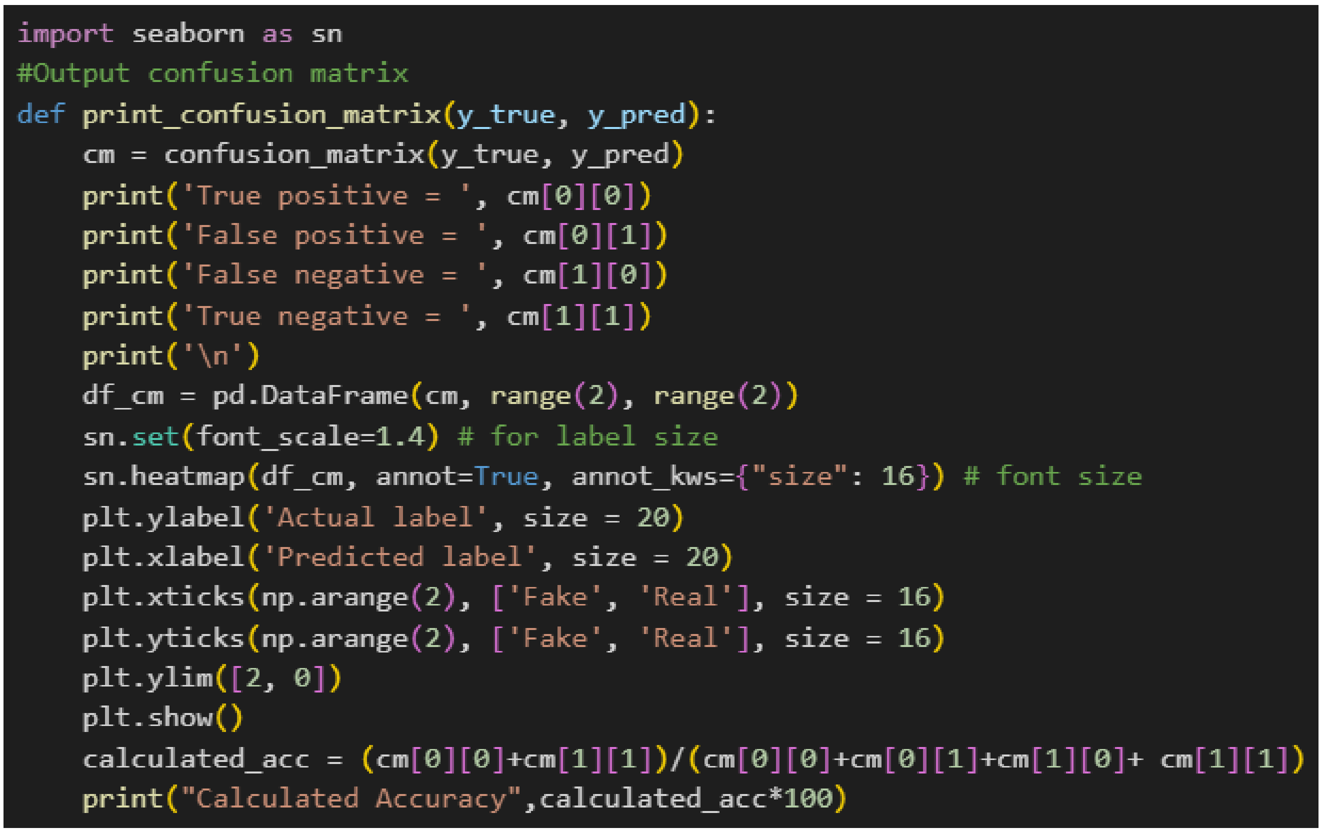


```python
def test(epoch,model, data_loader ,criterion):
    print('Testing')
    model.eval()
    losses = AverageMeter()
    accuracies = AverageMeter()
    pred = []
    true = []
    count = 0
    with torch.no_grad():
        for i, (inputs, targets) in enumerate(data_loader):
            if torch.cuda.is_available():
                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)).tolist()
            pred += p.detach().cpu().numpy().reshape(len(p)).tolist()
            losses.update(loss.item(), inputs.size(0))
            accuracies.update(acc, inputs.size(0))
            sys.stdout.write(
                    "\r[Batch %d / %d]  [Loss: %f, Acc: %.2f%%]"
                    % (
                        i,
                        len(data_loader),
                        losses.avg,
                        accuracies.avg
                        )
                    )
        print('\nAccuracy {}'.format(accuracies.avg))
    return true,pred,losses.avg,accuracies.avg
```



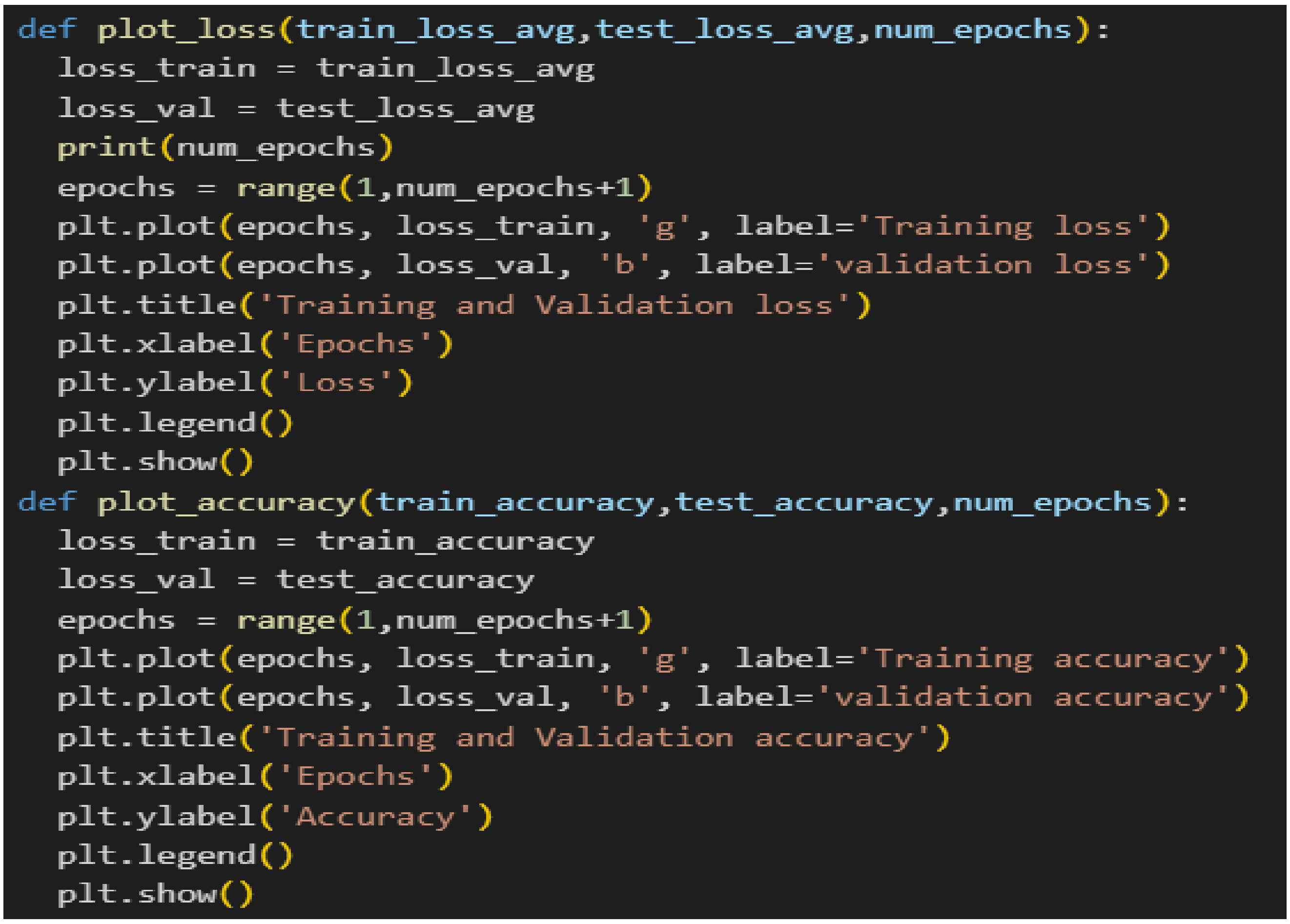
```python
class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()
    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    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
```

Fig 9.2.9: For training epochs, testing, calculate accuracy



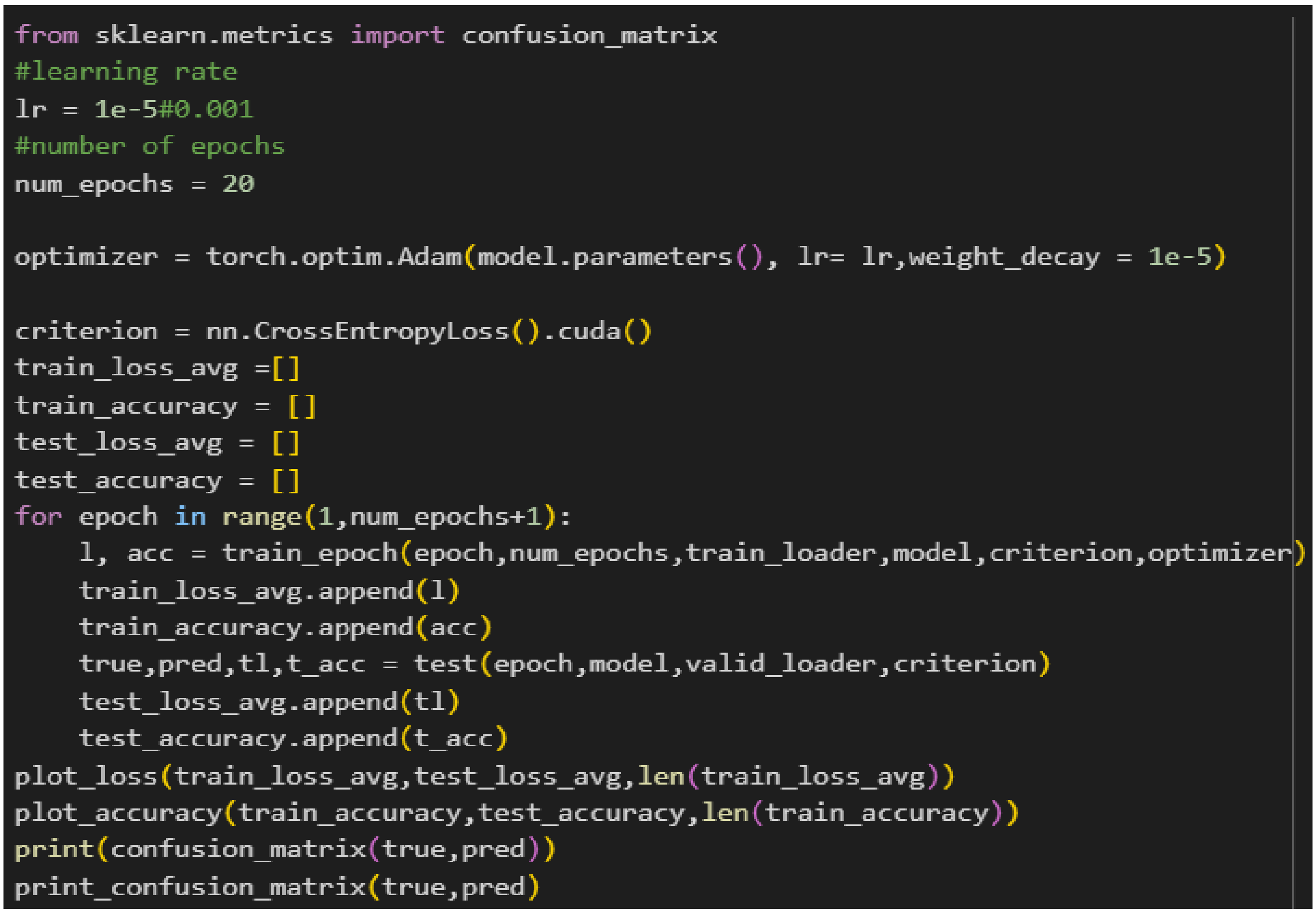
```python
import seaborn as sn
#Output confusion matrix
def print_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    print('True positive = ', cm[0][0])
    print('False positive = ', cm[0][1])
    print('False negative = ', cm[1][0])
    print('True negative = ', cm[1][1])
    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)
```

Fig 9.2.10: For printing calculated accuracy



```python
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()
```

Fig 9.2.11: To plot and accuracy



```
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)

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)
```

Fig 9.2.12: To print confusion matrix

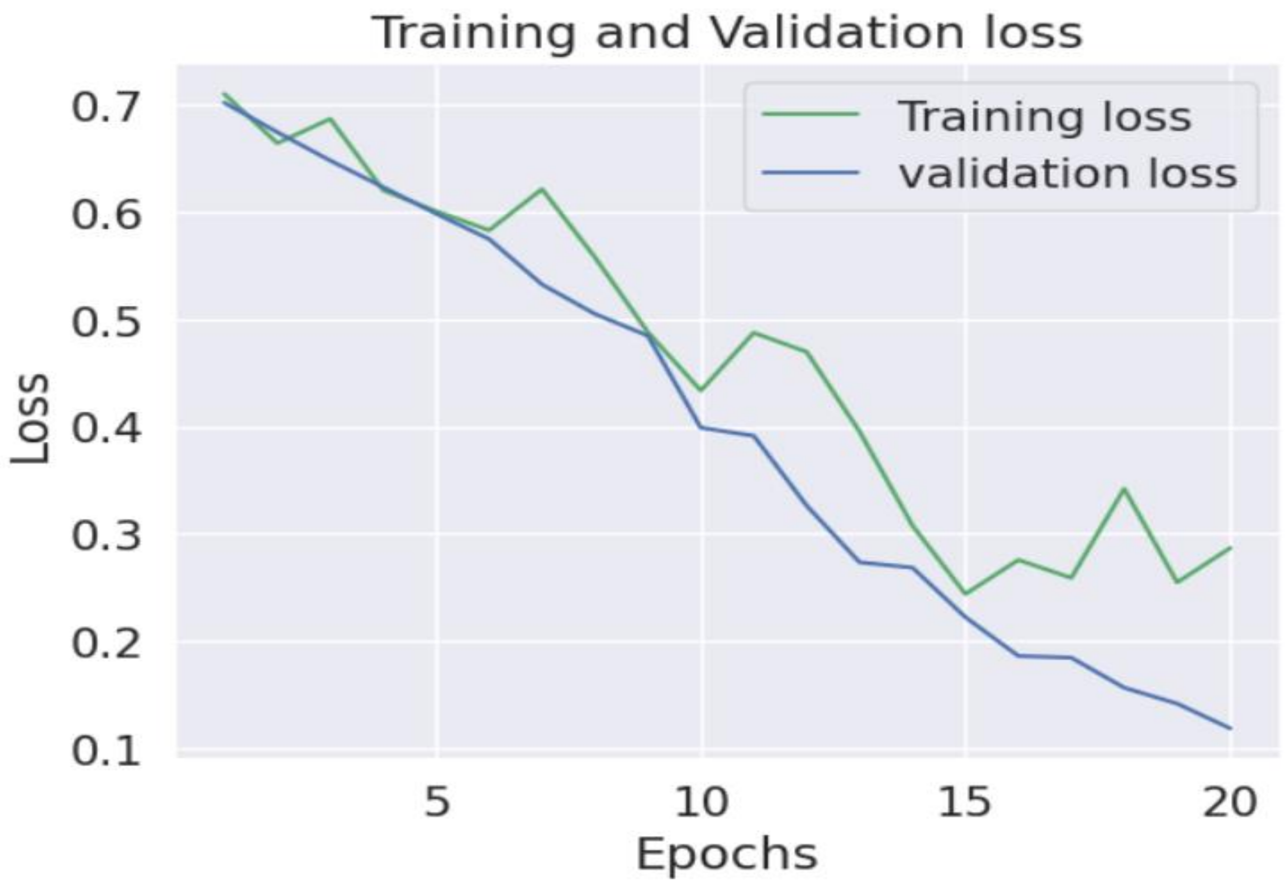


Fig 9.2.13: Train and Validity Loss

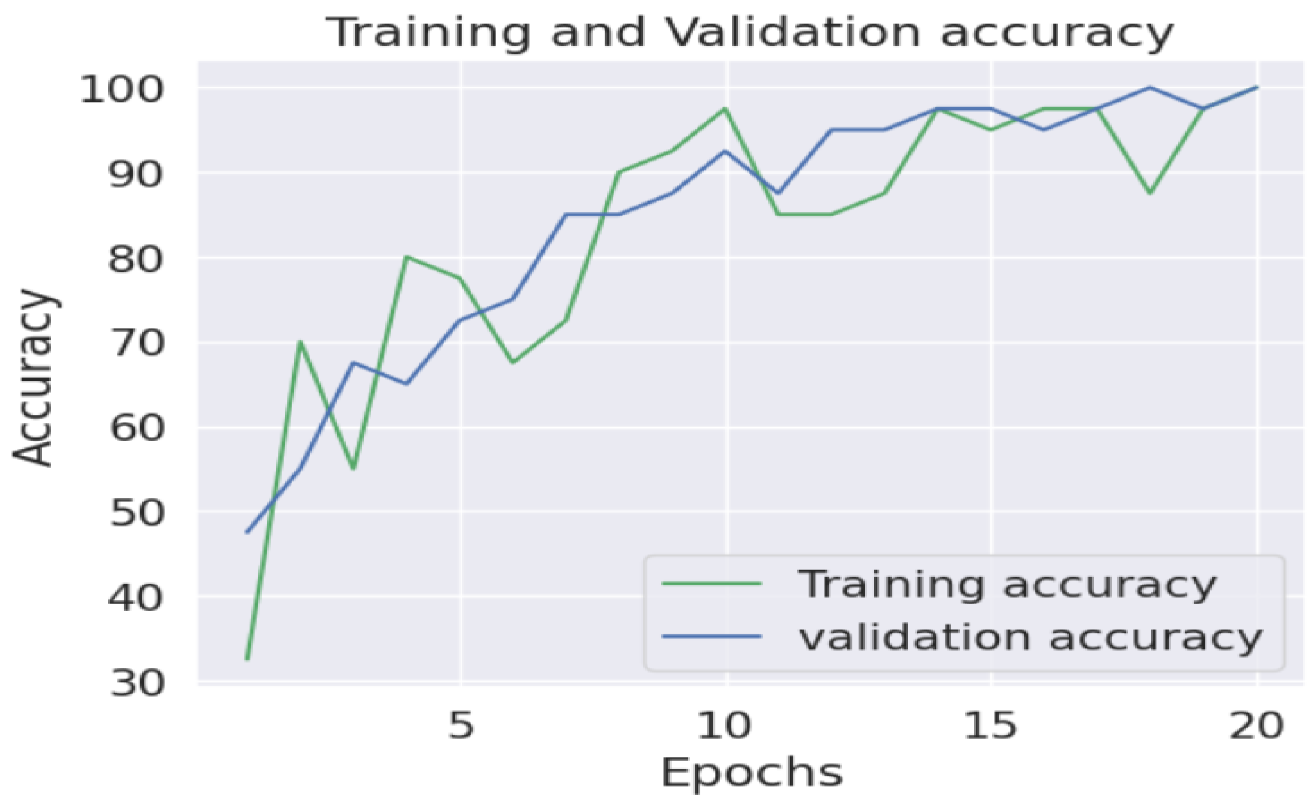
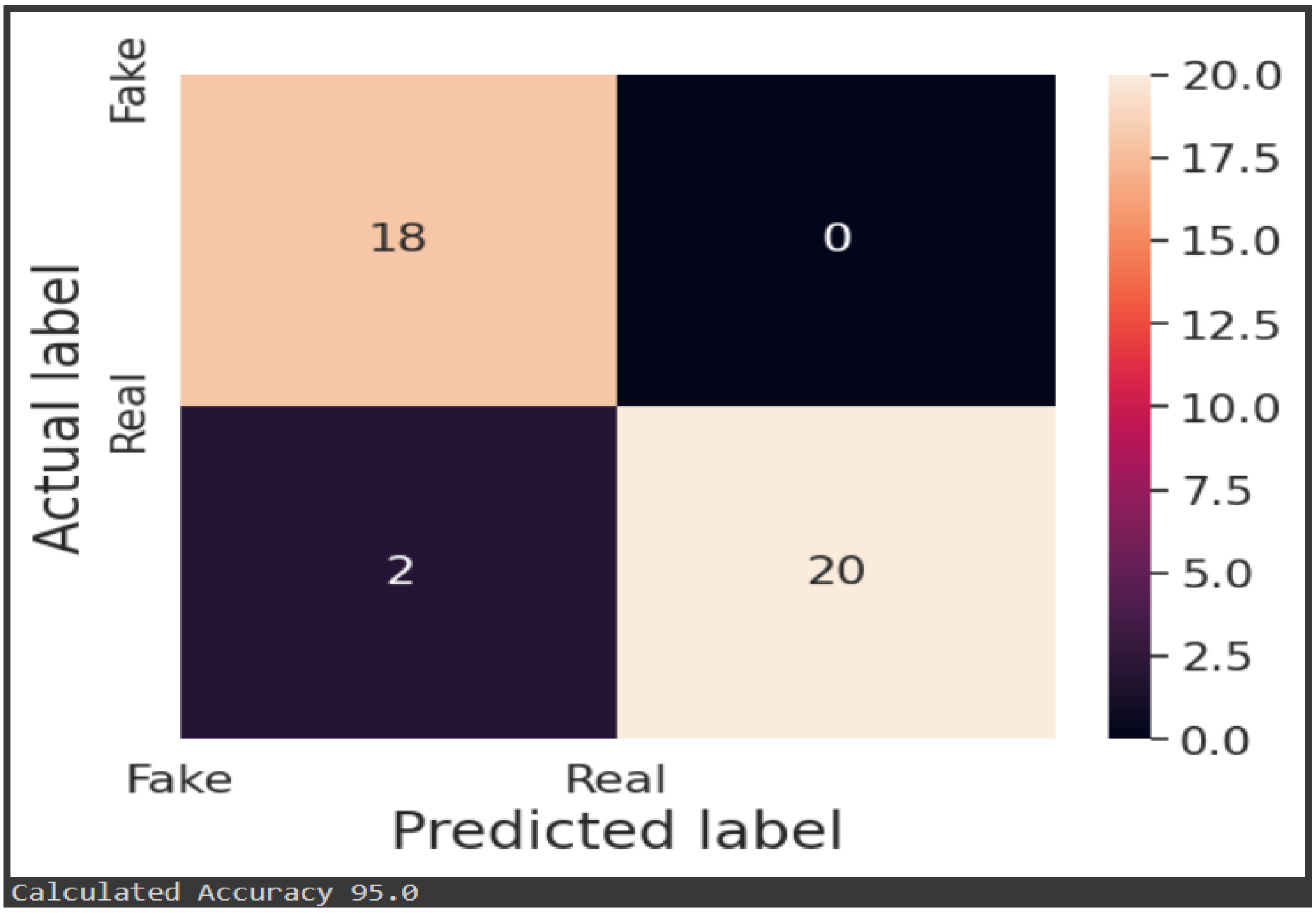


Fig 9.2.14: Training and Validation Accuracy



```
Calculated Accuracy 95.0
```

Fig 9.2.15: Confusion matrix

PREDICTION DETAILS:

- **Mounting Google Drive:** The notebook starts by mounting Google Drive to access files stored there.

- **Importing Libraries:** Various libraries are imported, including PyTorch, OpenCV, Matplotlib, and face recognition, among others.

- **Defining the Model:** A pre-trained ResNeXt-50 architecture with LSTM layers made especially for sequence analysis is used to build the deep learning model.

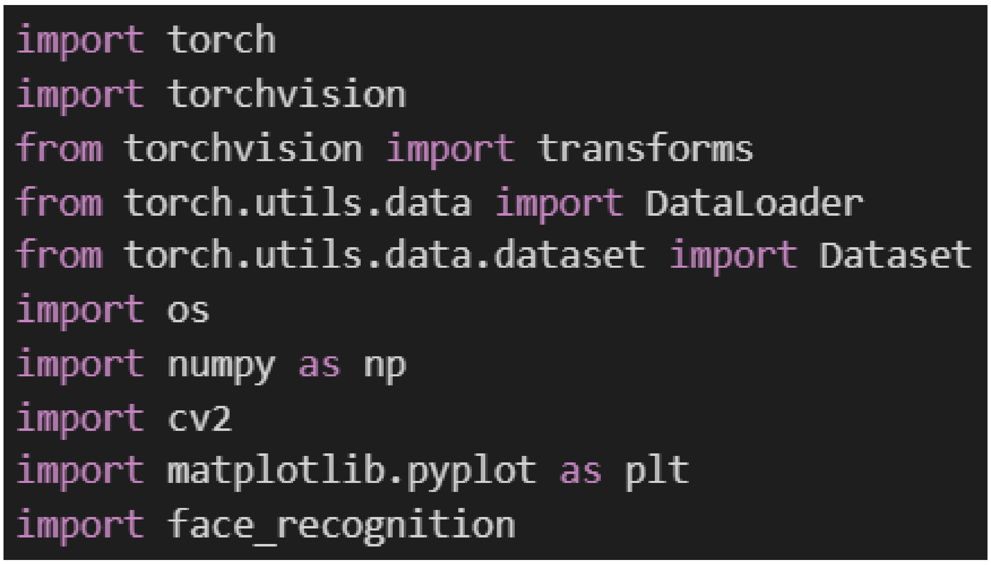
- **Image Processing Utilities:** Utility functions for image conversion, prediction, and graphing are defined.

- **Dataset Class:** A dataset class is established to facilitate the loading of video frames, incorporating the capability to crop faces in the event that face recognition is accessible.

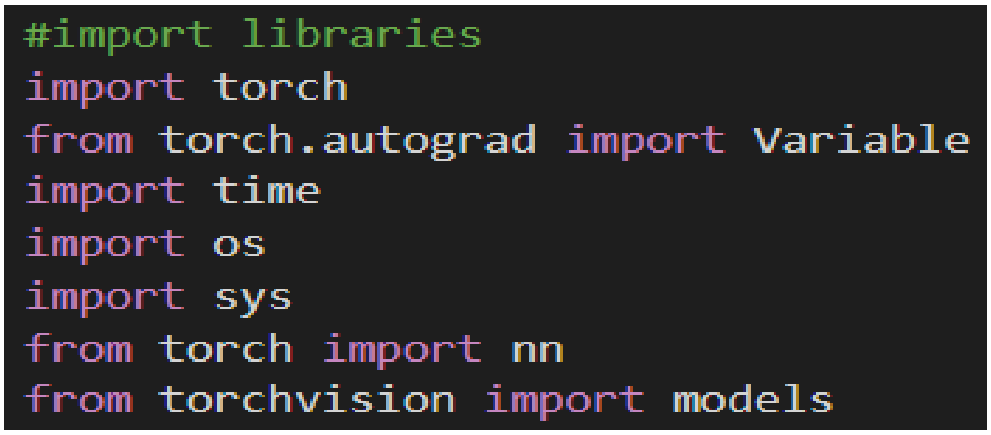
- **Prediction Code:** The primary prediction function is implemented, which imports the trained model, analyses video frames, and generates predictions regarding the presence or absence of a deepfake in the video.

- **Optional entire Frame Processing:** An optional component for processing entire frames instead of cropped faces is given.

PREDICTION STEPS:

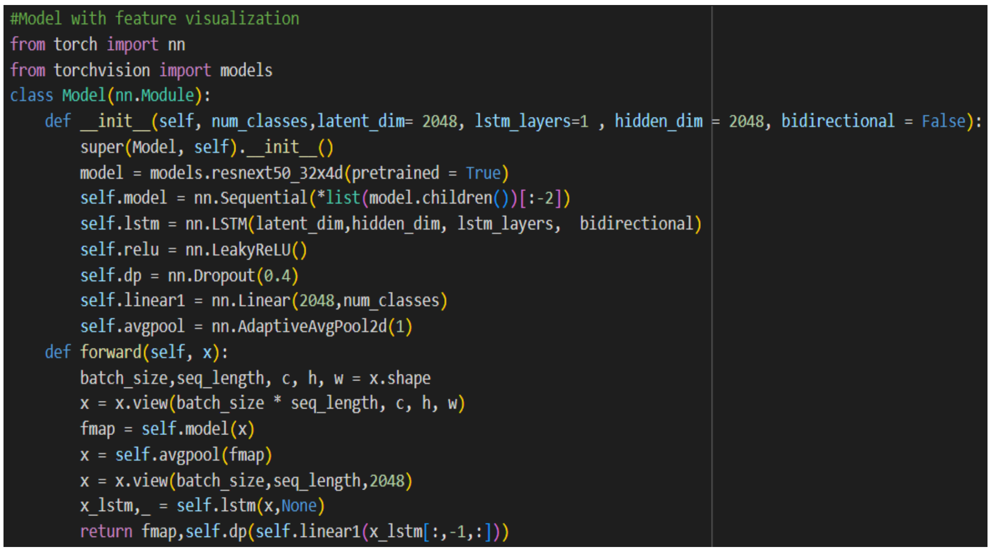


```python
import torch
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import face_recognition
```



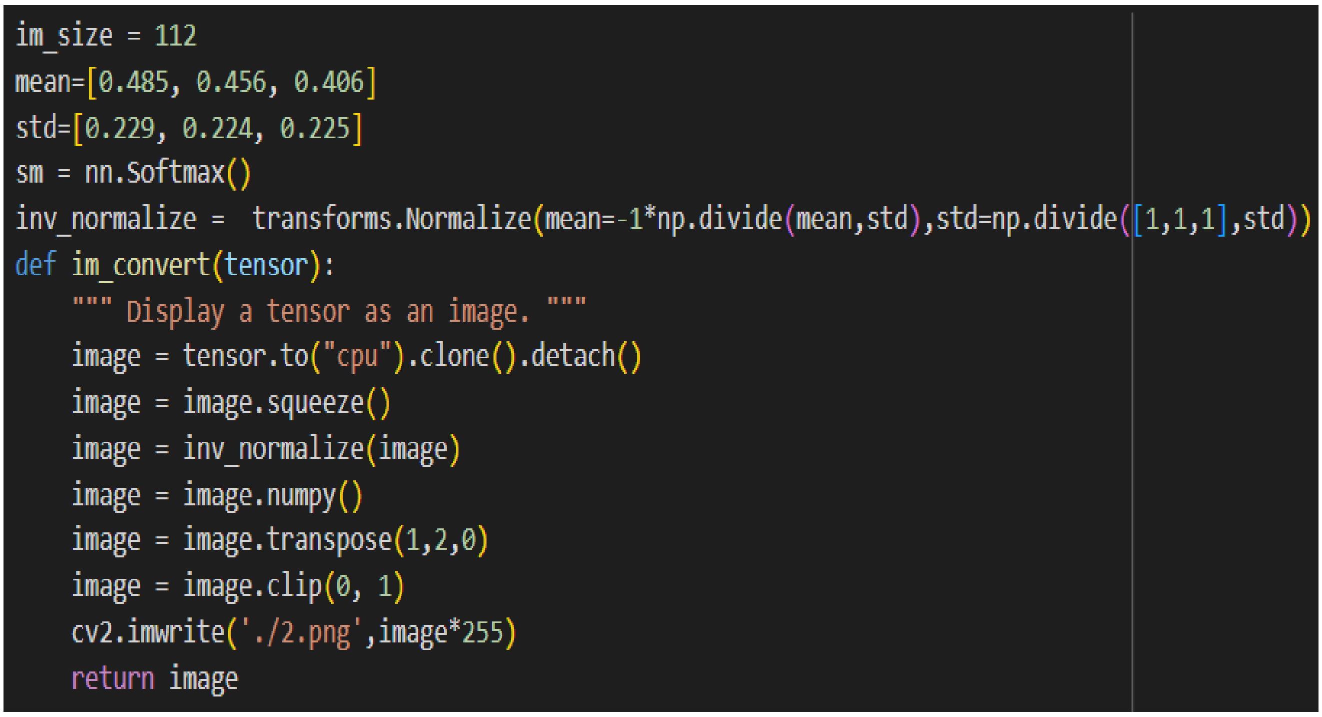
```python
#import libraries
import torch
from torch.autograd import Variable
import time
import os
import sys
from torch import nn
from torchvision import models
```

Fig 9.3.1: To import libraries

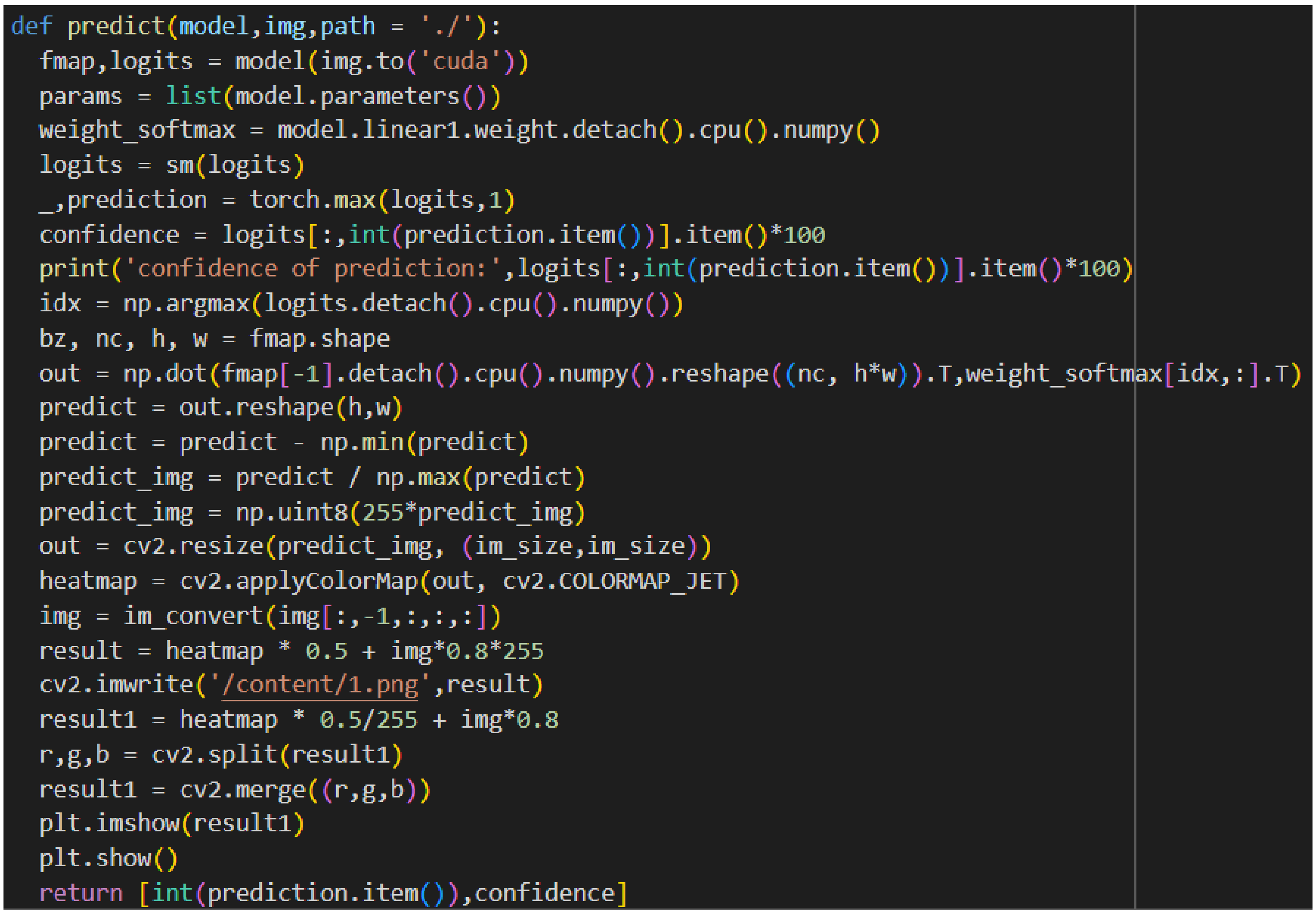


```python
#Model with feature visualization
from torch import nn
from torchvision import models
class Model(nn.Module):
    def __init__(self, num_classes,latent_dim= 2048, lstm_layers=1 , hidden_dim = 2048, bidirectional = False):
        super(Model, self).__init__()
        model = models.resnext50_32x4d(pretrained = True)
        self.model = nn.Sequential(*list(model.children())[:-2])
        self.lstm = nn.LSTM(latent_dim,hidden_dim, lstm_layers,  bidirectional)
        self.relu = nn.LeakyReLU()
        self.dp = nn.Dropout(0.4)
        self.linear1 = nn.Linear(2048,num_classes)
        self.avgpool = nn.AdaptiveAvgPool2d(1)
    def forward(self, x):
        batch_size,seq_length, c, h, w = x.shape
        x = x.view(batch_size * seq_length, c, h, w)
        fmap = self.model(x)
        x = self.avgpool(fmap)
        x = x.view(batch_size,seq_length,2048)
        x_lstm,_ = self.lstm(x,None)
        return fmap,self.dp(self.linear1(x_lstm[:,-1,:]))
```

Fig 9.3.2: Code for feature visualization

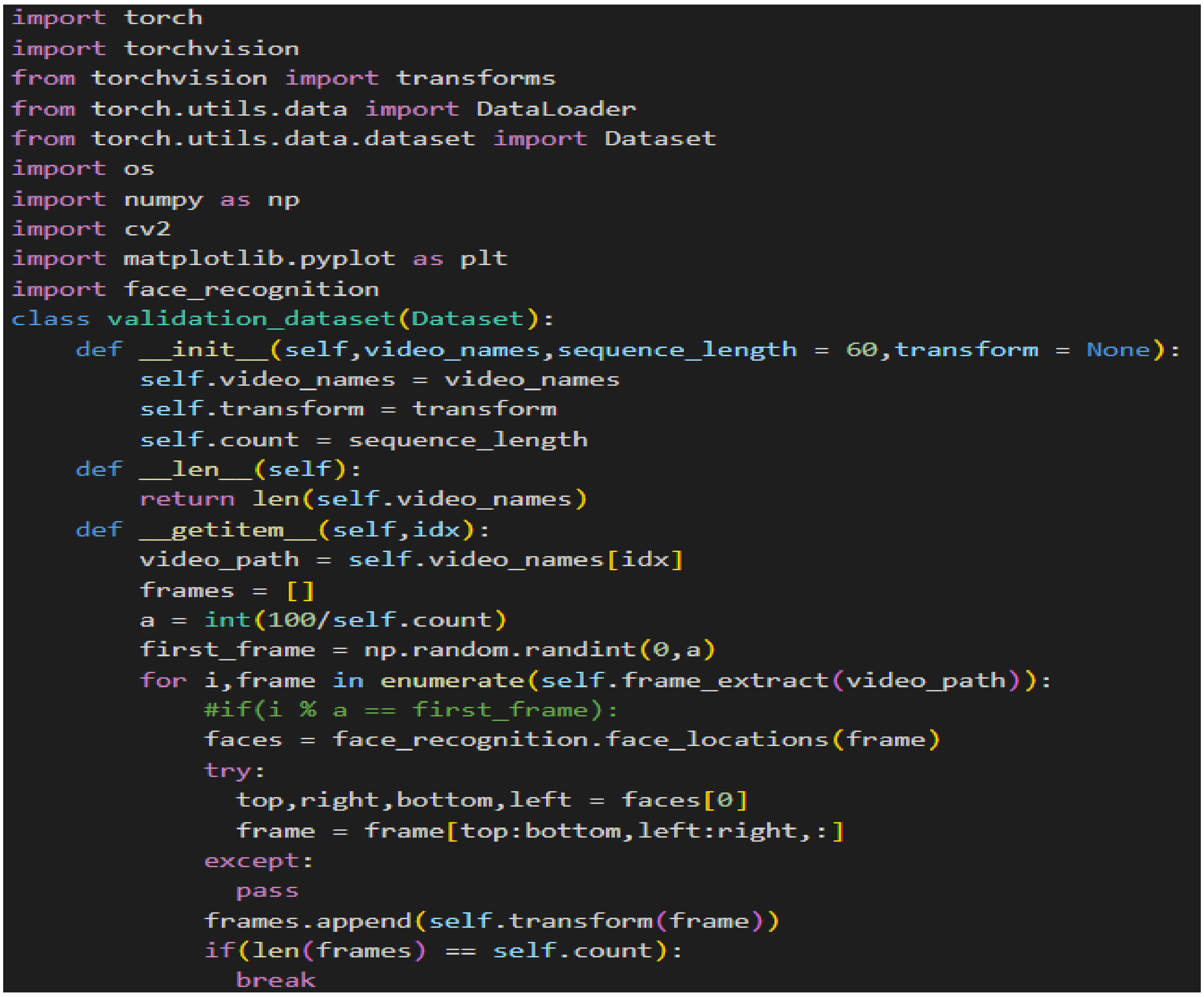


```python
im_size = 112
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
sm = nn.Softmax()
inv_normalize =  transforms.Normalize(mean=-1*np.divide(mean,std),std=np.divide([1,1,1],std))
def im_convert(tensor):
    """ Display a tensor as an image. """
    image = tensor.to("cpu").clone().detach()
    image = image.squeeze()
    image = inv_normalize(image)
    image = image.numpy()
    image = image.transpose(1,2,0)
    image = image.clip(0, 1)
    cv2.imwrite('./2.png',image*255)
    return image
```

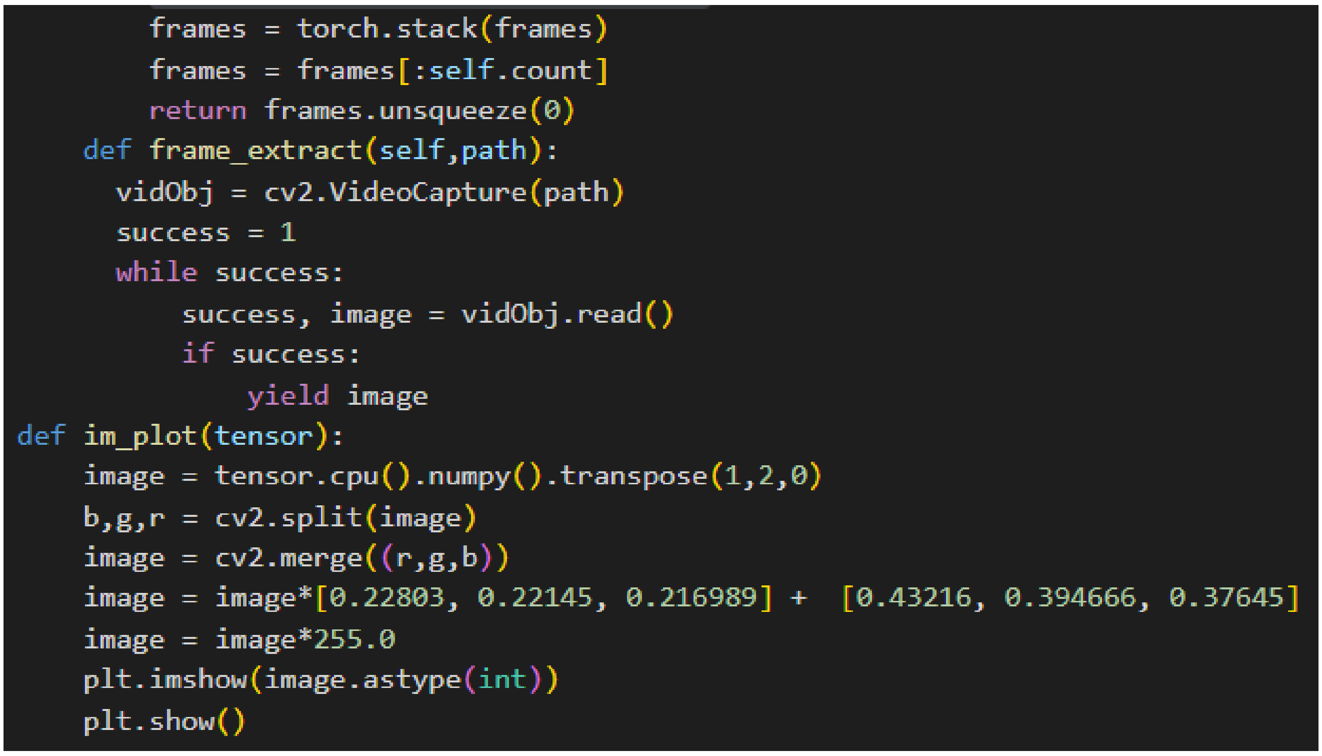


```python
def predict(model,img,path = './'):
  fmap,logits = model(img.to('cuda'))
  params = list(model.parameters())
  weight_softmax = model.linear1.weight.detach().cpu().numpy()
  logits = sm(logits)
  _,prediction = torch.max(logits,1)
  confidence = logits[:,int(prediction.item())].item()*100
  print('confidence of prediction:',logits[:,int(prediction.item())].item()*100)
  idx = np.argmax(logits.detach().cpu().numpy())
  bz, nc, h, w = fmap.shape
  out = np.dot(fmap[-1].detach().cpu().numpy().reshape((nc, h*w)).T,weight_softmax[idx,:].T)
  predict = out.reshape(h,w)
  predict = predict - np.min(predict)
  predict_img = predict / np.max(predict)
  predict_img = np.uint8(255*predict_img)
  out = cv2.resize(predict_img, (im_size,im_size))
  heatmap = cv2.applyColorMap(out, cv2.COLORMAP_JET)
  img = im_convert(img[:,-1,:,:,:])
  result = heatmap * 0.5 + img*0.8*255
  cv2.imwrite('/content/1.png',result)
  result1 = heatmap * 0.5/255 + img*0.8
  r,g,b = cv2.split(result1)
  result1 = cv2.merge((r,g,b))
  plt.imshow(result1)
  plt.show()
  return [int(prediction.item()),confidence]
```

Fig 9.3.3: Prediction Code

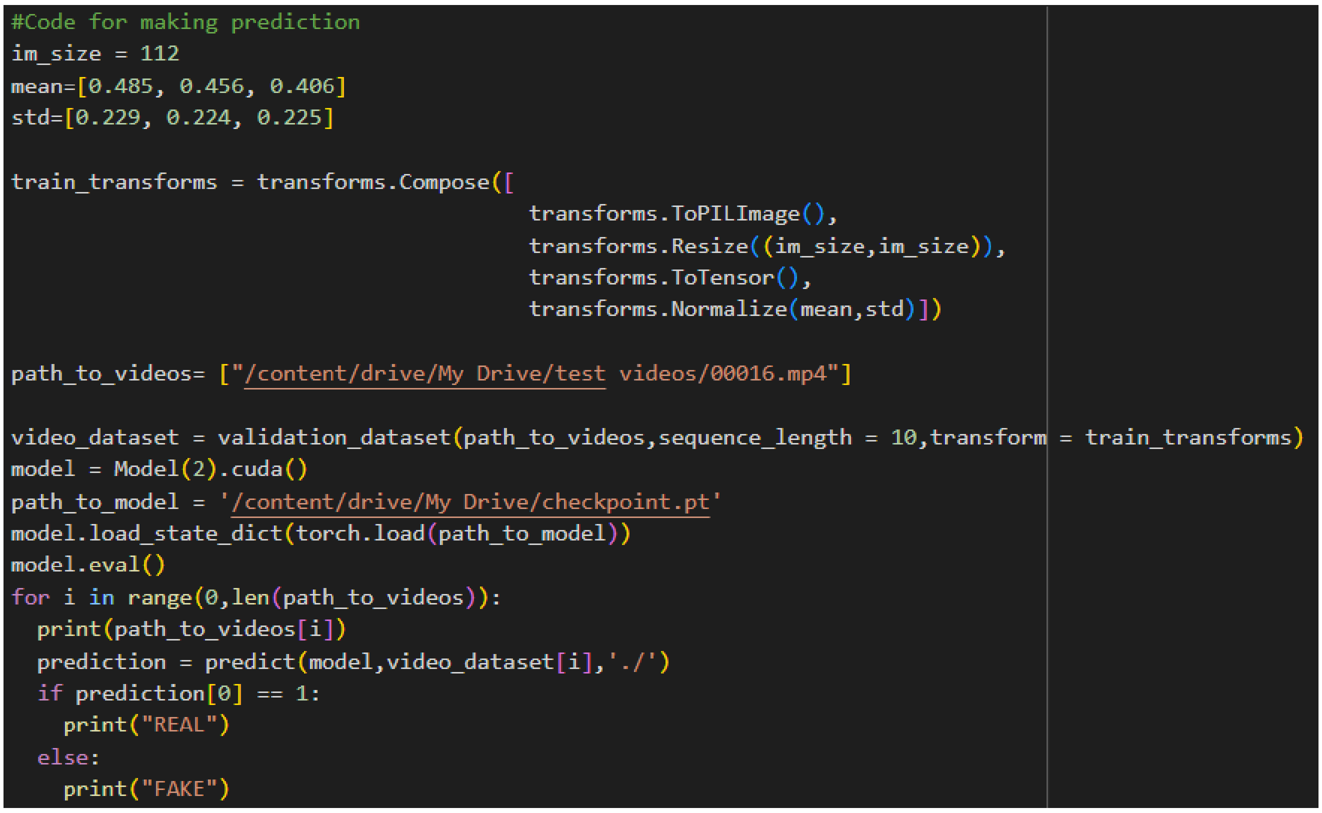


```python
import torch
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import face_recognition
class validation_dataset(Dataset):
    def __init__(self,video_names,sequence_length = 60,transform = None):
        self.video_names = video_names
        self.transform = transform
        self.count = sequence_length
    def __len__(self):
        return len(self.video_names)
    def __getitem__(self,idx):
        video_path = self.video_names[idx]
        frames = []
        a = int(100/self.count)
        first_frame = np.random.randint(0,a)
        for i,frame in enumerate(self.frame_extract(video_path)):
            #if(i % a == first_frame):
            faces = face_recognition.face_locations(frame)
            try:
              top,right,bottom,left = faces[0]
              frame = frame[top:bottom,left:right,:]
            except:
              pass
            frames.append(self.transform(frame))
            if(len(frames) == self.count):
              break
```



```python
        frames = torch.stack(frames)
        frames = frames[:self.count]
        return frames.unsqueeze(0)
    def frame_extract(self,path):
      vidObj = cv2.VideoCapture(path)
      success = 1
      while success:
          success, image = vidObj.read()
          if success:
              yield image
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()
```

Fig 9.3.4: Code for frame extract, plotting.



```
#Code for making prediction
im_size = 112
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]

train_transforms = transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.Resize((im_size,im_size)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean,std)])

path_to_videos= ["/content/drive/My Drive/test videos/00016.mp4"]

video_dataset = validation_dataset(path_to_videos,sequence_length = 10,transform = train_transforms)
model = Model(2).cuda()
path_to_model = '/content/drive/My Drive/checkpoint.pt'
model.load_state_dict(torch.load(path_to_model))
model.eval()
for i in range(0,len(path_to_videos)):
  print(path_to_videos[i])
  prediction = predict(model,video_dataset[i],'./')
  if prediction[0] == 1:
    print("REAL")
  else:
    print("FAKE")
```

Fig 9.3.5: Code for prediction of video

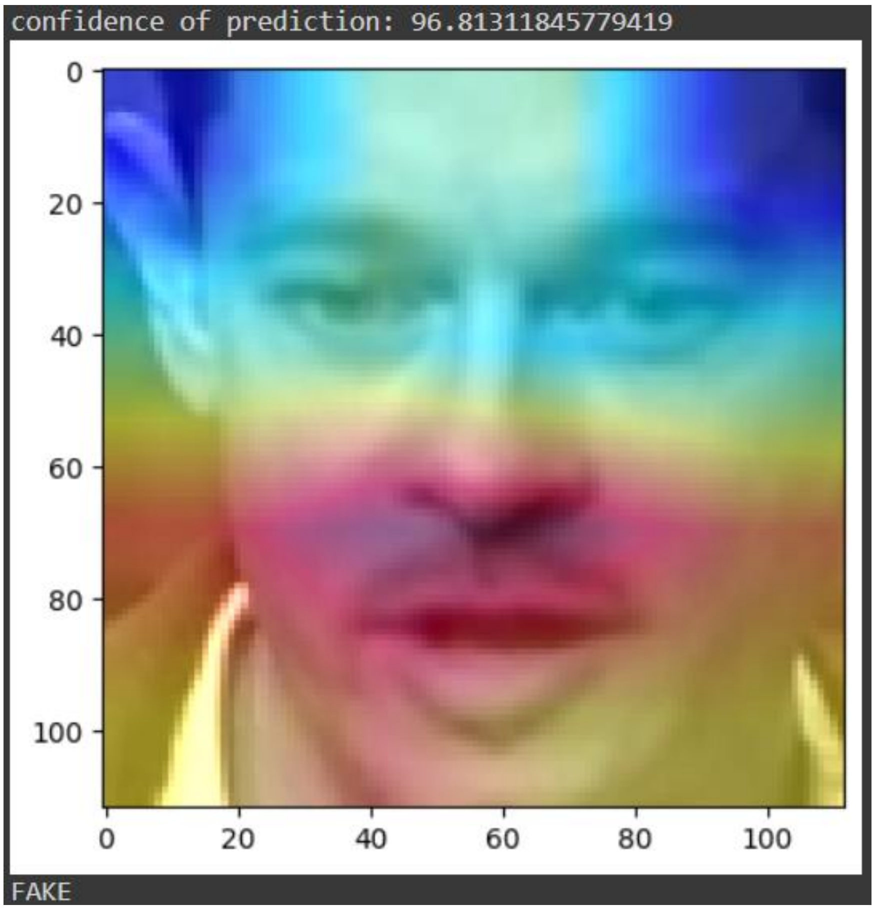


Fig 9.3.6: Showing result that given video is fake

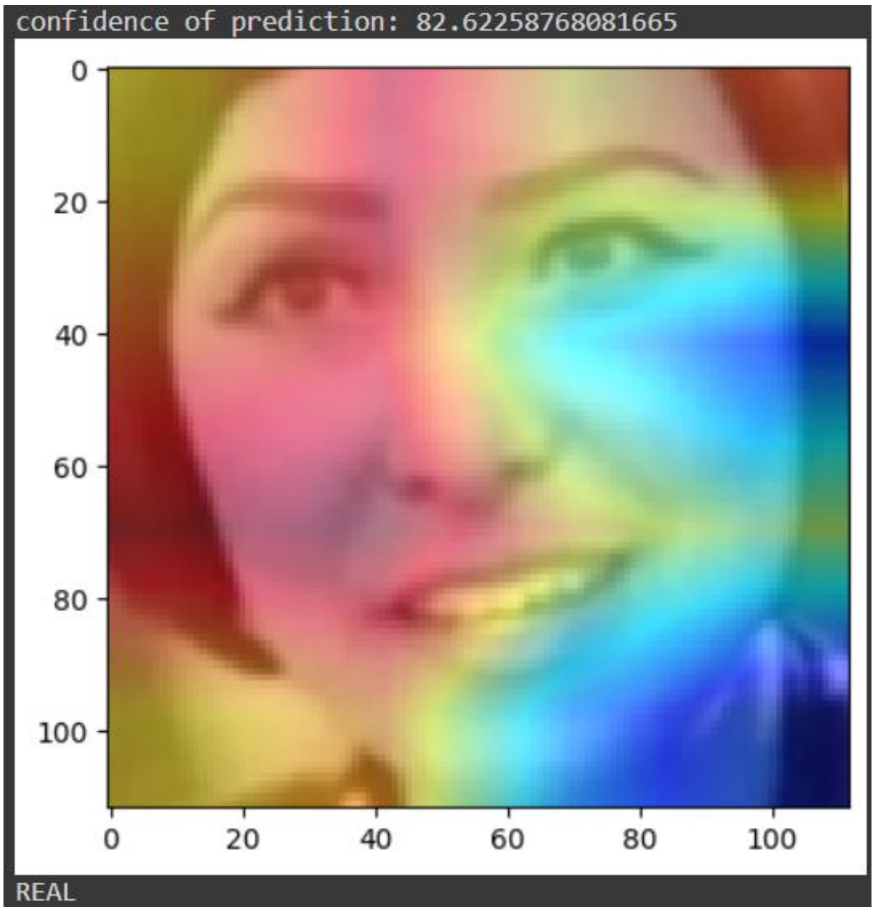


Fig 9.3.7: Showing result that given video is real

10. SOFTWARE TESTING

TYPE OF TESTING USED:

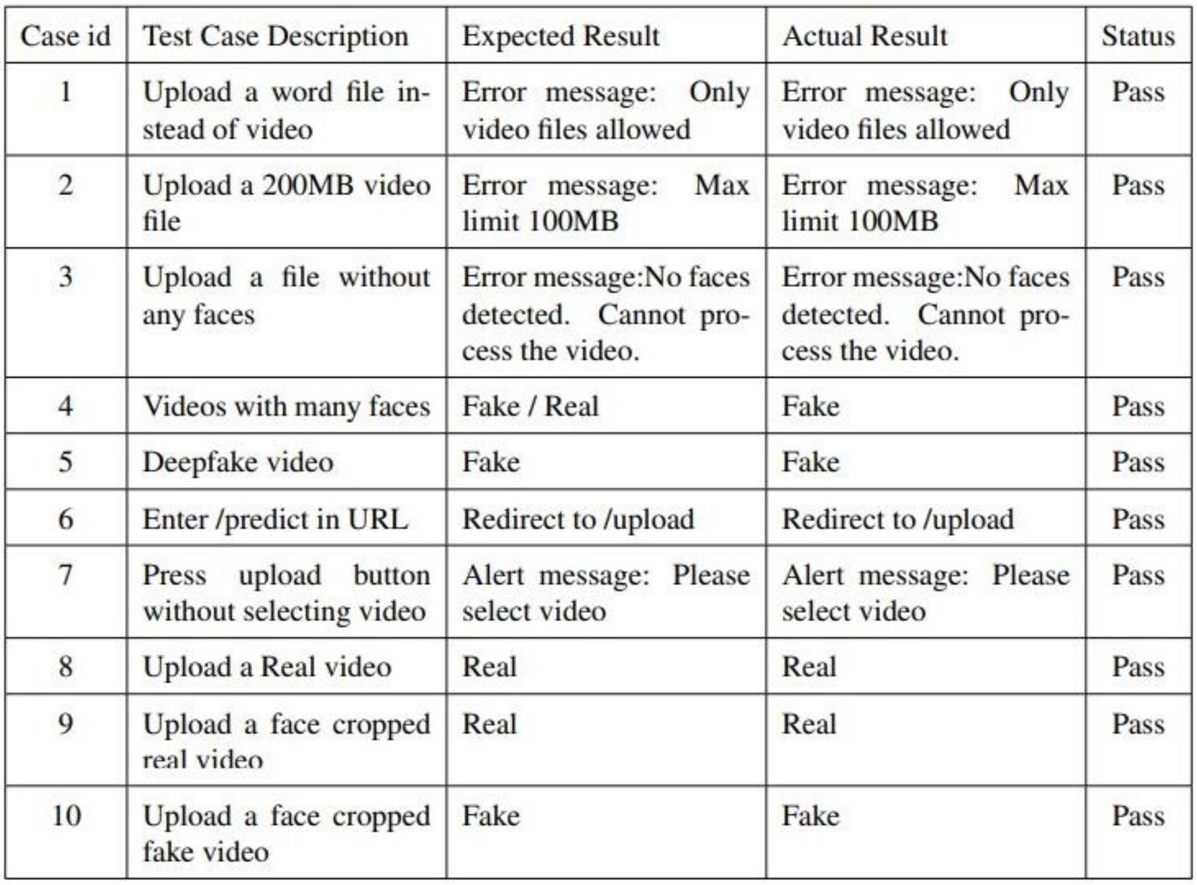
FUNCTIONAL TESTING:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Interface Testing

NON-FUNCTIONAL TESTING:

1. Performance Testing
2. Load Testing
3. Compatibility Testing

TEST CASE AND TEST RESULTS:



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 limit 100MB	Error message: Max limit 100MB	Pass
3	Upload a file without any faces	Error message:No faces detected. Cannot process the video.	Error message:No faces detected. Cannot process the 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: Test Case Report.

11. CONCLUSIONS AND FUTURE SCOPE

RESULTS:

We provided a neural network approach to discerning the authenticity of a video, distinguishing between deep fakes and genuine content, while also indicating the model's confidence level. After analysing 1 second of video (10 fps), our method can predict the outcome with excellent accuracy. We utilized a pre-trained ResNext CNN model and an LSTM to extract frame-level features and analyse temporal sequences, thereby identifying differences between the frames at time "t" and "t-1". Our model can handle video frames in the following sequences: 10, 20, 40, 60, 80, 100.

FUTURE PERSPECTIVE:

Any system that has been constructed may always be improved, especially if it was created utilizing the most recent technology and has a promising future.

- Web based platforms can be upscaled to a browser plugin for ease of access to the user.
- Currently only Face Deep Fakes are being detected by the algorithm, but the algorithm can be enhanced in detecting full body deep fake

12. REFERENCES

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

[2.] Deepfake detection challenge dataset : https://www.kaggle.com/c/deepfake-detection-challenge/data Accessed on 26 March, 2020

[3.] 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

[4.] Deepfake Video of Mark Zuckerberg Goes Viral on Eve of House A.I. Hearing : https://fortune.com/2019/06/12/deepfake-mark-zuckerberg/ Accessed on 26 March, 2020

[5.] 10 deepfake examples that terrified and amused the internet : https://www.creativebloq.com/features/deepfake-examples Accessed on 26 March, 2020

[6.] TensorFlow: https://www.tensorflow.org/ (Accessed on 26 March, 2020)

[7.] Keras: https://keras.io/ (Accessed on 26 March, 2020)

[8.] PyTorch: https://pytorch.org/ (Accessed on 26 March, 2020)

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

[10.] 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.

[11.] Face app: https://www.faceapp.com/ (Accessed on 26 March, 2020)

[12.] Face Swap : https://faceswaponline.com/ (Accessed on 26 March, 2020)

[13.] 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/

[14.] The rise of the deepfake and the threat to democracy : https://www.theguardian.com/technology/ng-interactive/2019/jun/22/the-rise-of- the-deepfake-and-the-threat-to-democracy(Accessed on 26 March, 2020)

[15.] https://www.theguardian.com/technology/ng-interactive/2019/jun/22/the-rise-of- the-deepfake-and-the-threat-to-democracy(Accessed on 26 March, 2020)

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

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

[18.] Huy H. Nguyen , Junichi Yamagishi, and Isao Echizen “ Using capsule net- works to detect forged images and videos ” in arXiv:1810.11215.

[19.] 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.

[20.] Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic hu- man actions from movies. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8, June 2008. Anchorage, AK

[21.] Umur Aybars Ciftci, ˙Ilke Demir, Lijun Yin “Detection of Synthetic Portrait Videos using Biological Signals” in arXiv:1901.02212v2

[22.] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv:1412.6980, Dec. 2014.

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

[24.] https://www.geeksforgeeks.org/software-engineering-cocomo-model/ Accessed on 15 April 2020

[25.] Deepfake Video Detection using Neural Networks: http://www.ijsrd.com/articles/IJSRDV8I10860.pdf

[26.] International Journal for Scientific Research and Development: http://ijsrd.com/