VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi, Karnataka - 590 018

A PROJECT REPORT

ON

"DEEPFAKE CREATION AND DETECTION USING CYCLE GANs"

SUBMITTED BY

CHARAN K	**1RI16CS013**
NAVEED AHMED	**1RI17CS027**
NIKITH KUMAR N	**1RI17CS028**
SHANKAR R A	**1RI17CS043**

Under the guidance of

Prof. Shruthi S

Assistant Professor, Dept. of CSE, RRIT

In partial fulfillment of the award of degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering
R R INSTITUTE OF TECHNOLOGY
Bengaluru, Karnataka, India – 560 090

2020-2021

R R INSTITUTE OF TECHNOLOGY

No. 67, Raja Reddy Layout, Hesaraghatta Main Road, Chikkabanavara, Bangalore - 560 090

Department of Computer Science and Engineering

CERTIFICATE

Certified that the final year phase-II project work prescribed in 17CSP85 entitled "DEEPFAKE CREATION AND DETECTION USING CYCLE GANs", carried out by **Mr. CHARAN K (1RI16CS013), Mr. NAVEED AHMED (1RI17CS027), Mr. NIKITH KUMAR N (1RI17CS028), Mr. SHANKAR R A (1RI17CS043),** bonafide students of R R Institute Of Technology, Bengaluru in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belagavi during the year 2020 - 21. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The final year project phase-I report has been approved as it satisfies the academic requirements in respect of final year project work prescribed for the said degree.

.....	.....	.....
Signature of Internal Guide	**Signature of HOD**	**Signature of Principal**
[Prof. Shruthi S]	**[Dr. Manjunath R]**	**[Dr. Mahendra K V]**
Assistant professor,	Prof., and Head of Dept	Principal
Dept. of CSE, RRIT	Dept. of CSE, RRIT	RRIT, Bengaluru

Name of the Examiners	**Signature with date**
1……	……..
2……..	…….

ACKNOWLEDGEMENTS

The completion of final year phase-II project work brings a sense of satisfaction, but it is never complete without thanking the persons responsible for its successful completion.

At the outset, We express our most sincere grateful acknowledgment to the holy sanctum "**R R INSTITUTE OF TECHNOLOGY**", the temple of learning, for giving me an opportunity to pursue the degree course in Computer Science and Engineering and thus helping me in shaping my career.

We extend my deep sense of sincere gratitude to **Dr. MAHENDRA K V**, Principal, R R Institute of Technology, Bengaluru, for providing me an opportunity to continue my higher studies.

We express my heartfelt sincere gratitude to **Dr. MANJUNATH R**, Professor, and HOD, Department of Computer Science and Engineering, R R Institute of Technology, Bengaluru, for his valuable suggestions and support.

We extend my special in-depth, heartfelt, sincere gratitude to my guide **Prof. SHRUTHI S**, Assistant Professor, and Co-ordinator **Prof. DHANANJAYA M K,** Assistant Professor, Department of Computer Science and Engineering, R R Institute of Technology, Bengaluru, for her constant support and valuable guidance for completion of the final year phase-II project work.

We would like to thank all the teaching and non-teaching staff members in my Department of Computer Science and Engineering, R R Institute of Technology, Bengaluru, for their support.

Finally, We would like to thank all my friends and family members for their constant support, guidance, and encouragement.

CHARAN K (1RI16CS013)

NAVEED AHMED (1RI17CS027)

NIKITH KUMAR N (1RI17CS028)

SHANKAR R A (1RI17CS043)

DECLARATION

We, **CHARAN K, NAVEED AHMED, SHANKAR R A, NIKITH KUMAR N,** students of 4th Year, Department of Computer Science and Engineering, R R Institute of Technology, Bengaluru, hereby declare that this final year project entitled "**DEEPFAKE CREATION AND DETECTION USING CYCLE GANs**" is the result of our work and has been carried out under the supervision of our guide **Prof. SHRUTHI S**, Assistant Professor, Department of Computer Science and Engineering.

It is not substantially the same as any that we have submitted for a degree or diploma or other qualification at the Visvesvaraya Technological University or any other University or similar institution. We further state that no substantial part of our final year project report has already been submitted, or is being concurrently submitted, for any such degree, diploma, or other qualification during the academic year 2020 - 2021.

CHARAN K (1RI16CS013)

SIGNATURE :

NAVEED AHMED (1RI17CS027)

SIGNATURE :

NIKITH KUMAR N (1RI17CS028)

SIGNATURE :

SHANKAR R A (1RI17CS043)

SIGNATURE :

Place **:** Bengaluru

Date **: …….**

ABSTRACT

Deepfake Creation and Detection Using Cycle GANs is to solve various complex problems, from big data analytics to computer vision and human-level control, deep learning has been successfully applied. However, advances in deep learning have also been used to develop software that can cause threats to privacy, democracy, and national security. A "deepfake" is one of those deep learning-powered applications that have recently emerged. Deepfake algorithms can generate fake images and videos that can not be distinguished by humans from real ones.

Deepfake is a popular technique based on artificial intelligence for image synthesis. As it can produce images without paired training data, it is more powerful than traditional image-to-image translation.

The most common form of deepfakes involves the generation and manipulation of human imagery. This technology has creative and productive applications. For example, realistic video dubbing of foreign films, education through the reanimation of historical figures, and virtually trying on clothes while shopping. There are also numerous online communities devoted to creating deepfake memes for entertainment, such as music videos portraying the face of actors.

CONTENTS

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A Deepfake refers to a specific kind of synthetic media where a person in an image or video is swapped with another person's likeness.

The increasing sophistication of smartphone cameras and the availability of good internet connection all over the world has increased the ever-growing reach of social media and media sharing portals have made the creation and transmission of digital videos more easy than ever before. The growing computational power has made deep learning so powerful that would have been thought impossible only a handful of years ago. Like any transformative technology, this has created new challenges. So-called "Deepfake" is produced by deep generative adversarial models that can manipulate video and audio clips. Spreading of the Deepfake over the social media platforms have become very common leading to spamming and speculating wrong information over the platform. These types of the Deepfake will be terrible, and lead to threatening, misleading common people.

To overcome such a situation, Deepfake detection is very important. So, we describe a new deep learning-based method that can effectively distinguish AI-generated fake videos (Deepfake Videos) from real videos. It’s incredibly important to develop technology that can spot fakes, so that the Deepfake can be identified and prevented from spreading over the internet.

The underlying mechanism for deepfake creation is deep learning models such as autoencoders and generative adversarial networks, which have been applied widely in the computer vision domain. These models are used to examine facial expressions and movements of a person and synthesize facial images of another person making analogous expressions and movements. Deepfake methods normally require a large amount of image and video data to train models to create photo-realistic images and videos.

1.1 About Domain

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

The main machine learning methods used to create deepfakes are based on deep learning and involve training generative neural network architectures, such as autoencoders or generative adversarial networks (GANs).

Deepfakes are synthetic media in which a person in an existing image or video is replaced with someone else's likeness. While the act of faking content is not new, deepfakes leverage powerful techniques from machine learning and artificial intelligence to manipulate or generate visual and audio content with a high potential to deceive.

1.2 Overview

Deepfake is a popular image synthesis technique based on artificial intelligence. It is more powerful than traditional image-to-image translation as it can generate images without given paired training data.

For creation and detection of the deepfake it is very important to understand the way Generative Adversarial Network (GAN) creates the deepfake. GANs takes as input a video and an image of a specific individual (‘target’), and outputs another video with the target’s faces replaced with those of another individual (‘source’). The backbone of deepfakes are deep adversarial neural networks trained on face images and target videos to automatically map the faces and facial expressions of the source to the target. With proper postprocessing, the resulting videos can achieve a high level of realism. The GANs splits the video into frames and replaces the input image in every frame. Further it reconstructs the video. This process is usually achieved by using autoencoders. We describe a new deep learning-based method that can effectively distinguish deepfakes videos from the real ones. Our method is based on the same process that is used to create the deepfake by GANs.

1.3 Objectives

The aim of this project is to provide the reader with a deeper understanding of the creation and detection of deepfakes, the current trends and advances in this field, the weaknesses of current defense solutions, and the areas that require further research and attention.

One of the important objectives is to evaluate its performance and acceptability in terms of security, user friendliness, accuracy and reliability.

Explore the possibility of detecting fake images/videos for any social media platforms. If we can successfully detect fake images/videos from social media platforms, it would significantly reduce the user effort to identify fake images/videos.

1.4 Problem Identification

The explosive growth in deepfake video and its illegal use is a major threat to democracy, justice, and public trust. Due to this there is an increased demand for fake video analysis, detection and intervention.

Most of the current research aimed at combating the impact of deepfakes have focused on automated deepfake detection using algorithms to determine whether a specific image, audio clip, or video has been significantly modified from an original.

The ability to detect must extend to journalists, fact-checkers and civil society groups in an easy to understand interface. Deepfake detection solution development faces a tradeoff between open source datasets and models and deterring adversaries who could use those resources to improve deepfakes.

Our goal is to train the CycleGAN model to learn how to map between images of two domains, and then use the trained models to generate deepfakes for the given input image.

1.4.1 Existing System

There are many tools available for creating the deepfake, but for deepfake detection there is hardly any tool available. Our approach for detecting the deepfake will be a great contribution in avoiding the percolation of the deepfake over the world wide web. Since most of the deepfakes are created via adversarial training (GANs), the creator algorithm's ability to evade AI-based detection methods will improve as they are introduced to new detection systems.

Recently, Facebook Deepfake Detection Challenge (DFDC) Results: The best models got 65% accuracy on real-world data. The results reinforce the difficulty of deepfake detection and emphasize the limitations of AI models to mitigate the synthetic media threat.

1.4.2 Proposed System

Our approach for creating and detecting the deepfake will be a great contribution in avoiding the percolation of the deepfake over the world wide web. The project can be scaled up from developing a web based platform to a browser plugin for automatic deepfake detections. Even big applications like WhatsApp, Facebook can integrate this project with their application for easy pre-detection of deepfake before sending it to another user.

One of the important objectives is to evaluate its performance and acceptability in terms of security, user-friendliness, accuracy and reliability. Our method is focusing on detecting all types of deepfake like replacement deepfake , retrenchment deepfake and interpersonal deepfake.

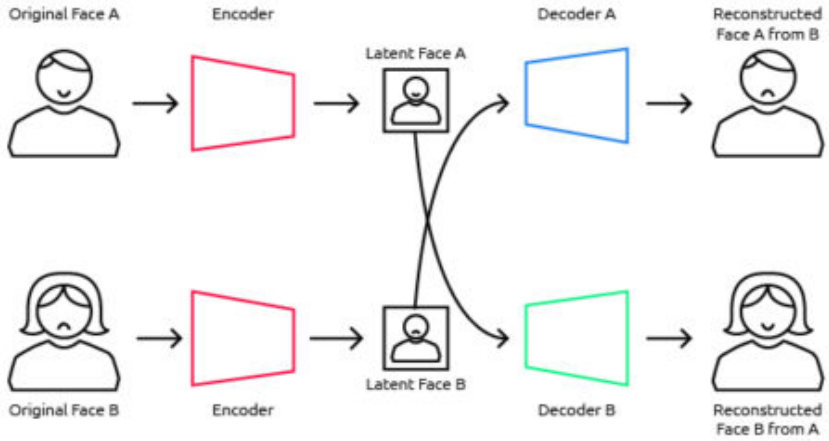

Fig. 1.1 Represents the simple system for creation procedure of deepfake of the proposed system.

CHAPTER 2

LITERATURE SURVEY

Deepfake video's explosive growth and illegal use is a major threat to democracy, justice, and public trust. As a result, there is an increased demand for fake video analysis, detection and intervention. Some of the related word in deep fake detection are listed below:

[1] Title : Exposing Deepfake Videos by Detecting Face Warping Artifacts
Year : 22 May, 2019
Author : Yuezun Li, Siwei Lyu
Methodology : Exposing Deepfake Videos by Detecting Face Warping Artifacts used an approach to detect artifacts by comparing the generated face areas and their surrounding regions with a dedicated Convolutional Neural Network model. In this work there were two-fold Face Artifacts.

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

[2] Title : Exposing AI Generated Fake Face Videos by Detecting Eye Blinking
Year : 11 June, 2018
Author : Yuezun Li, Ming-Ching Chang, Siwei Lyu
Methodology : Exposing AI Created Fake Videos by Detecting Eye Blinking describes a new method to expose fake face videos generated with deep neural network models. The method is based on detection of eye blinking in the videos, which is a physiological signal that is not well presented in the synthesized fake videos. The method is evaluated over benchmarks of eye-blinking detection datasets and shows promising performance on detecting videos generated with Deep Neural Network based software Deepfake.

Their method only uses the lack of blinking as a clue for detection. However certain other parameters must be considered for detection of the deep fake like teeth enchantment, wrinkles on faces etc. Our method is proposed to consider all these parameters.

[3] Title : STGAN: A Unified Selective Transfer Network for Arbitrary Image Attribute Editing

Year : 21 April, 2019

Author : Ming Liu, Yukang Ding, Min Xia, Xiao Liu, Errui Ding, Wangmeng Zuo, Shilei Wen

Methodology : Expression Swap this manipulation, also known as face reenactment, consists of modifying the facial expression of the person. Although different manipulation techniques are proposed in the literature, e.g., at image level through popular GAN architectures , in this group we focus on the most popular techniques Face2Face and NeuralTextures, which replaces the facial expression of one person in a video with the facial expression of another person.

[4] Title : Deepfakes Detection with Automatic Face Weighting

Year : 4 May, 2020

Author : Daniel Mas Montserrat, Hanxiang Hao, S. K. Yarlagadda, Sriram Baireddy, Ruiting Shao

Methodology : One of the latest approaches proposed in the literature is STGAN. In general, attribute manipulation can be tackled by incorporating an encoder-decoder or GAN. However, as commented Liu et al., the bottleneck layer in the encoder-decoder usually provides blurry and low quality manipulation results.

To improve this, the authors presented and incorporated selective transfer units with an encoder-decoder for simultaneously improving the attribute manipulation ability and the image quality. As a result, STGAN has recently outperformed the state of the art in attribute manipulation.

[5] Title : Exploring Adversarial Fake Images on Face Manifold

Year : 8 January, 2021

Author : Dongze Li, Wei Wang, Hongxing Fan, Jing Dong

Methodology : Detection systems based not only on features at image level, but also at temporal level, along the frames of the video, have also been studied in the literature. Guera and Delp proposed ¨ in a temporal-aware pipeline to automatically detect fake videos. They considered a combination of CNNs and RNNs.

[6] Title : DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection

Year : 1 Jan 2020

Author : Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales

Methodology : Fake detection systems based on facial expressions and head movements have also been proposed in the literature. Yanget al. observes that some DeepFakes are created by splicing synthesised face regions into the original image, and in doing so, introducing errors that can be revealed when 3D head poses are estimated from the face images.

Thus, they performed a study based on the differences between head poses estimated using a full set of facial landmarks and those in the central face regions to differentiate DeepFakes from real videos. Once these features are extracted and normalised (mean and standard deviation), a SVM is considered for the final classification. Their proposed approach was originally evaluated with the UADFV database, achieving a final 89.0% AUC. However, this pre-trained model (using UADFV database) seems not to generalise very well to other databases as depicted.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Functional Requirements

3.1.1 Deepfake Creation

Deepfakes have become popular due to the quality of tampered videos and also the easy-to-use ability of their applications to a wide range of users with various computer skills from professional to novice. These applications are mostly developed based on deep learning techniques. Deep learning is well known for its capability of representing complex and high-dimensional data. One variant of the deep networks with that capability is deep autoencoders, which have been widely applied for dimensionality reduction and image compression.

The first attempt of deepfake creation was FakeApp, developed by a Reddit user using autoencoder-decoder pairing structure. In that method, the autoencoder extracts latent features of face images and the decoder is used to reconstruct the face images. To swap faces between source images and target images, there is a need of two encoder-decoder pairs where each pair is used to train on an image set, and the encoder's parameters are shared between two network pairs. In other words, two pairs have the same encoder network. This strategy enables the common encoder to find and learn the similarity between two sets of face images, which are relatively unchallenging because faces normally have similar features such as eyes, nose, mouth positions. Fig. 1.1 shows a deepfake creation process where the feature set of face A is connected with the decoder B to reconstruct face B from the original face A. This approach is applied in several works such as DeepFaceLab, DFaker, DeepFake-tf (tensorflow-based deepfakes).

Tools	Links	Key features
Faceswap	https://github.com/deepfakes/faceswap/	- Using two encoder-decoder pairs. - Parameters of the encoder are shared.
DeepFaceLab	https://github.com/iperov/DeepFaceLab/	- Expand from the Faceswap model with new models.
DeepFake-tf	https://github.com/StromWine/DeepFake-tf/	- Similar to DFaker but implemented based on tensorflow.

Fig 3.1 : Summary of notable deepfake tools

3.1.2 Deepfake Detection

Deepfakes are increasingly detrimental to privacy, society security and democracy. Methods for detecting deepfakes have been proposed as soon as this threat was introduced. Early attempts were based on handcrafted features obtained from artifacts and inconsistencies of the fake video synthesis process. Recent methods, on the other hand, applied deep learning to automatically extract salient and discriminative features to detect deepfakes.

Deepfake detection is normally deemed a binary classification problem where classifiers are used to classify between authentic videos and tampered ones. This kind of method requires a large database of real and fake videos to train classification models. The number of fake videos is increasingly available, but it is still limited in terms of setting a benchmark for validating various detection methods. These videos were then used to test various deepfake detection methods. Test results show that the popular face recognition systems based on VGG and Facenet are unable to detect deepfakes effectively. Other methods such as lip-syncing approaches and image quality metrics with support vector machines (SVM) produce very high error rates when applied to detect deepfake videos from this newly produced data set.

This section presents a survey of deepfake detection methods where we group them into two major categories: fake image detection methods and fake video detection ones (Fig. 3.1). The latter is distinguished into two groups: visual artifacts within video frame-based methods and temporal features across frames based ones. Whilst most of the methods based on temporal features use deep learning recurrent classification models, the methods use visual artifacts within the video frame can be implemented by either deep or shallow classifiers.

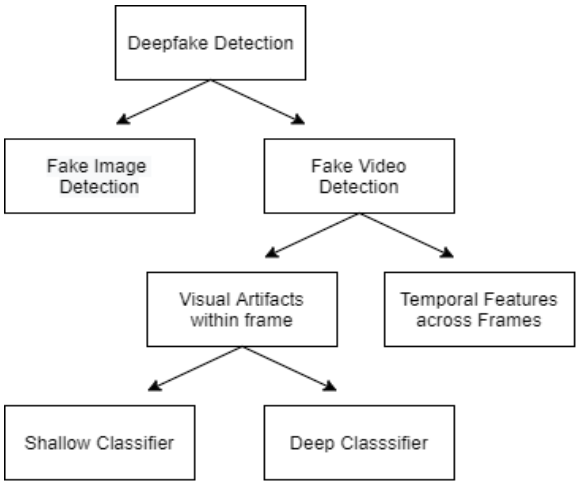

Fig 3.2 Fake image detection and face video detection.

3.2 Non - Functional Requirements

- Access Security

  The extent to which the system is safeguarded against deliberate and intrusive faults from internal and external sources.

- Accessibility

  The extent to which the software system can be used by people with the widest range of capabilities to achieve a specified goal in a specified context of use.

- Efficiency

  The extent to which the software system handles capacity, throughput, and response time.

- Confidentiality

  The degree to which the software system protects sensitive data and allows only authorized access to the data.

- Usability

  The ease with which the user is able to learn, operate, prepare inputs, and interpret outputs through interaction with a system.

- Interoperability

  The extent to which the software system is able to couple or facilitate the interface with other systems.

- Modifiability

  The degree to which changes to a software system can be developed and deployed efficiently and cost effectively.

3.3 Software Requirements

- Programming Language: Python 3
- Programming Framework: PyTorch
- Web Framework / Tools : Django (hosting)
- Operating System : Windows, Linux
- IDE: Google Colaboratory, Jupyter Notebook, VSCode / PyCharm

3.4 Hardware Requirements

- GPU : Clock Speed - 1770MHz, Memory Speed - 8Gbps, GDDR6
- Processor : Core i5 (3.9GHz)
- RAM : 8GB
- WEBCAM: 720p (autofocus, 30fps)

CHAPTER 4

SYSTEM DESIGN

4.1 System Architecture

4.1.1 System Architecture for Deepfake Creation:

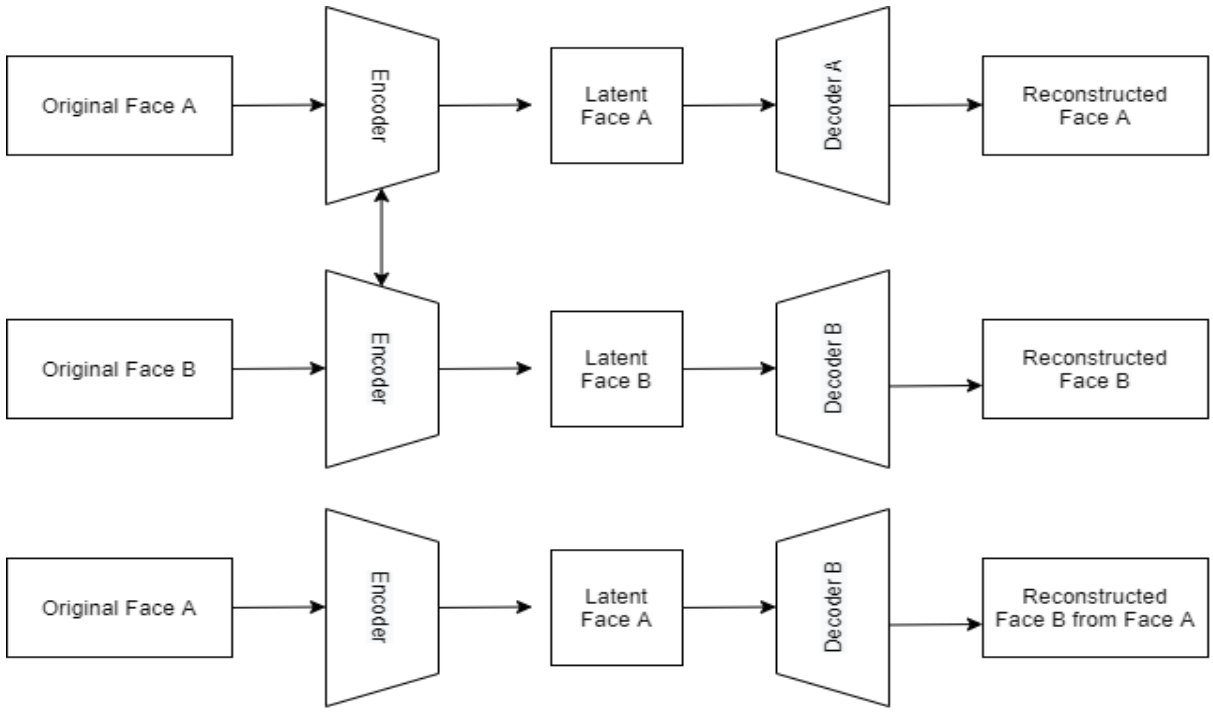

Fig. 4.1.1 : A deepfake creation model using two encoder-decoder pairs. Two networks use the same encoder but different decoders for the training process (top). An image of face A is encoded with the common encoder and decoder with decoder B to create a deepfake (bottom).

Training Phase: There are two phases involved in developing deepfakes. First is the training phase, which deals with the initial image manipulation. (Fig. 4.1.1 top)

Test Phase: Then, the test phase begins, which involves the warped image with another image that produces the specific result the creator is looking for. (Fig. 4.1.1 bottom)

4.1.2 System Architecture for Deepfake Detection:

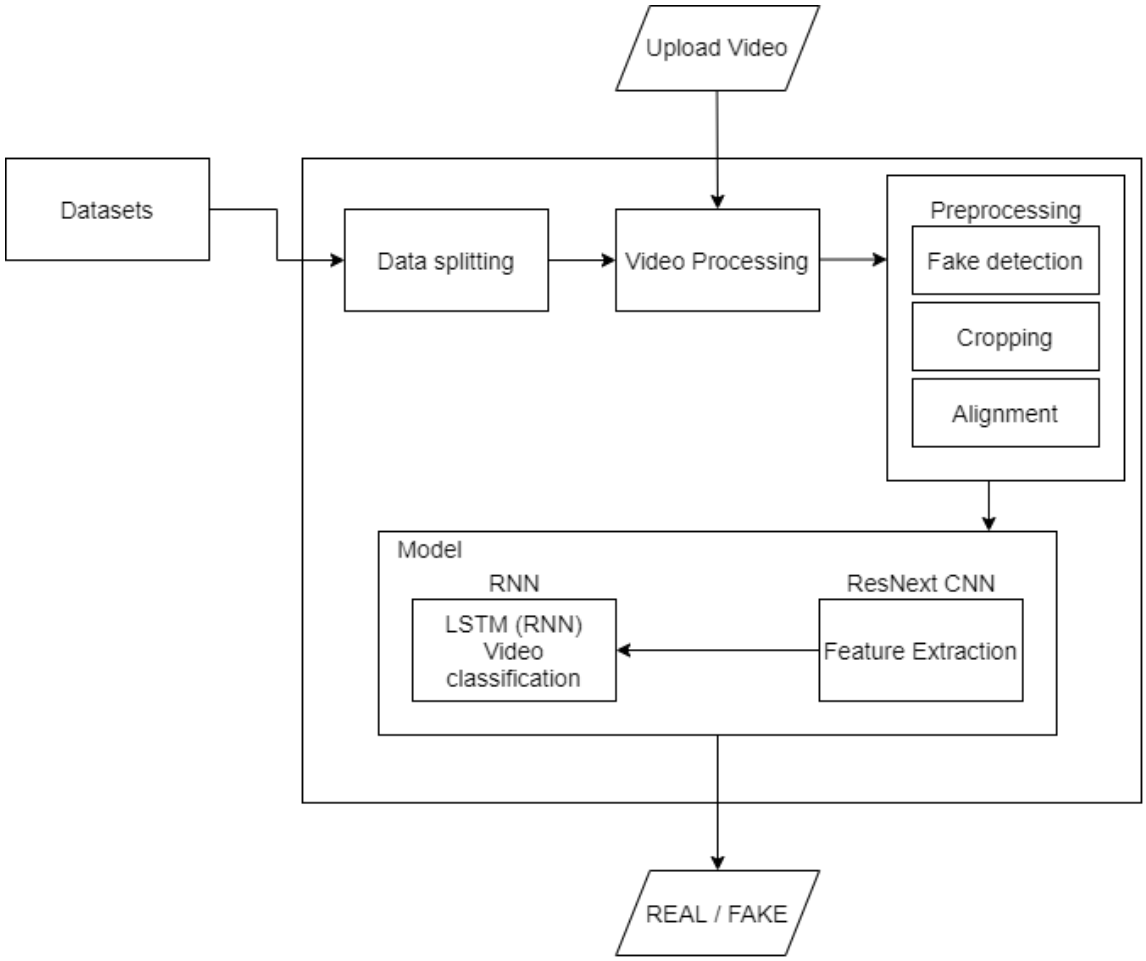

Fig. 4.1.2: System Architecture for Deepfake Detection

Procedure to create model:

Dataset : There are some sources such as YouTube, Face Forensics++, Deep fake detection challenge dataset etc, to train the mixed datasets, consisting of equal numbers of images/videos. Our newly preparing dataset contains 50% of the original video and 50% of the manipulated deepfake videos. The dataset is split into 70% train and 30% test set.

Preprocessing : Dataset preprocessing includes the splitting of the video into frames. Followed by the face detection and cropping the frame with the detected face. To maintain the uniformity in the number of frames the mean of the dataset video is calculated and the new processed face cropped dataset is created containing the frames equal to the mean. The frames that don't have faces in it are ignored during preprocessing.

Model : The Data Loader loads the preprocessed face cropped video and splits the videos into train and test sets. Further the frames from the processed videos are passed to the model for training and testing in mini batches.

Predict : A new video is passed to the trained model for prediction. A new video is also preprocessed to bring in the format of the trained model. The video is split into frames followed by face cropping and instead of storing the video into local storage the cropped frames are directly passed to the trained model for detection.

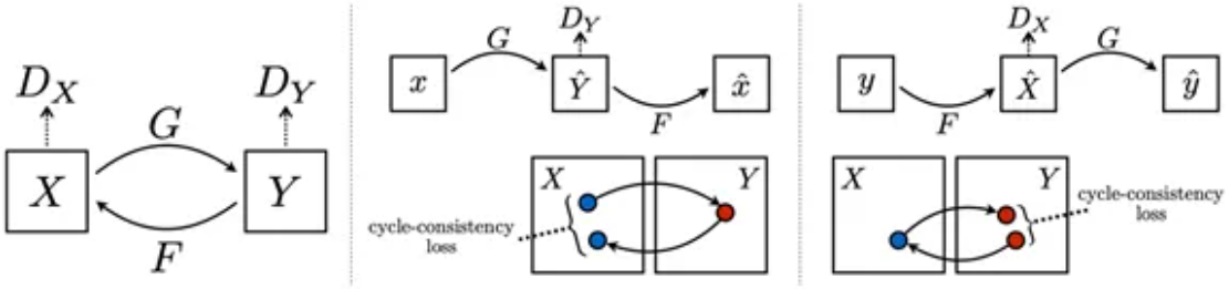

Fig 4.1.3: The training procedure for CycleGAN.

A custom dataset will be created using helper functions to get images of the target and input faces. The network is trained using multiple losses. We use the losses of the two generator-discriminator pairs, just like a general GAN (adversarial loss), but we also add a cyclic loss and identity loss. The cyclic loss is used when the image is cycled back after passing through both generators.

The need for this cyclic loss comes from our requirement that the image translated from one domain to another should retain the distinguishing features from the original domain. The identity loss helps the model to not make changes to the input image if its already in the required domain.

4.2 Data Flow Diagram

1) Data Flow Diagram for Deepfake Creation:

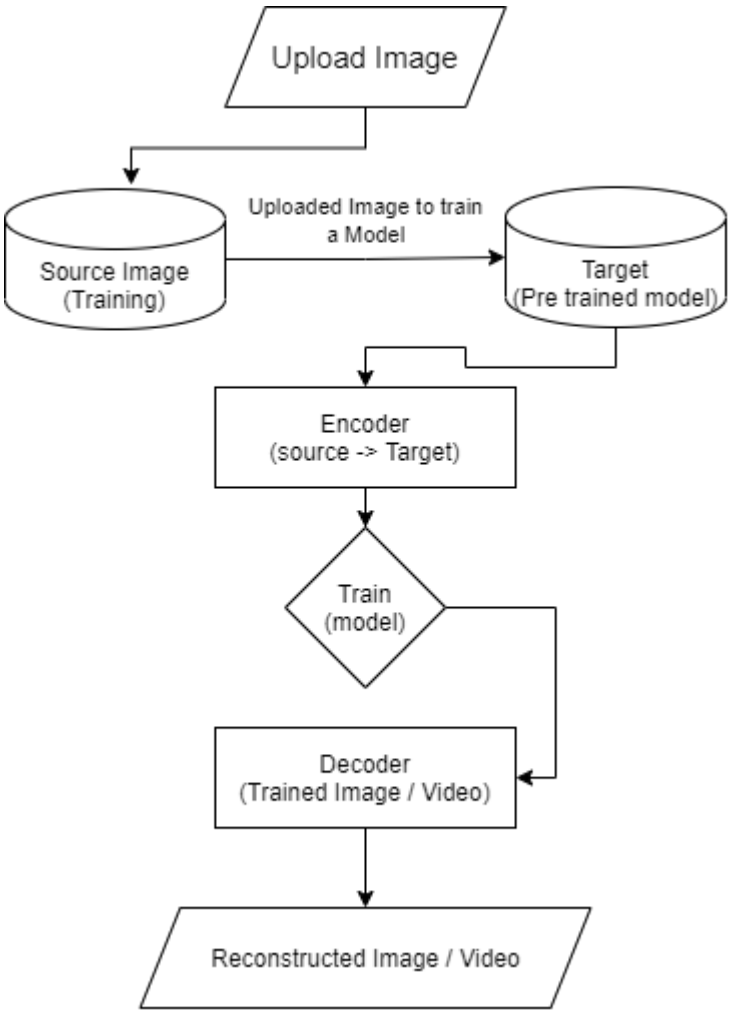

Fig. 4.2.1: Dataflow diagram for Deepfake Creation

The user must first upload the image from their system directory and select the video of the target person, then after some time the model will train the source image from the encoder to the targeted person decoder, resulting in the reconstructed video from the source image.

2) Data Flow Diagram for Deepfake Detection:

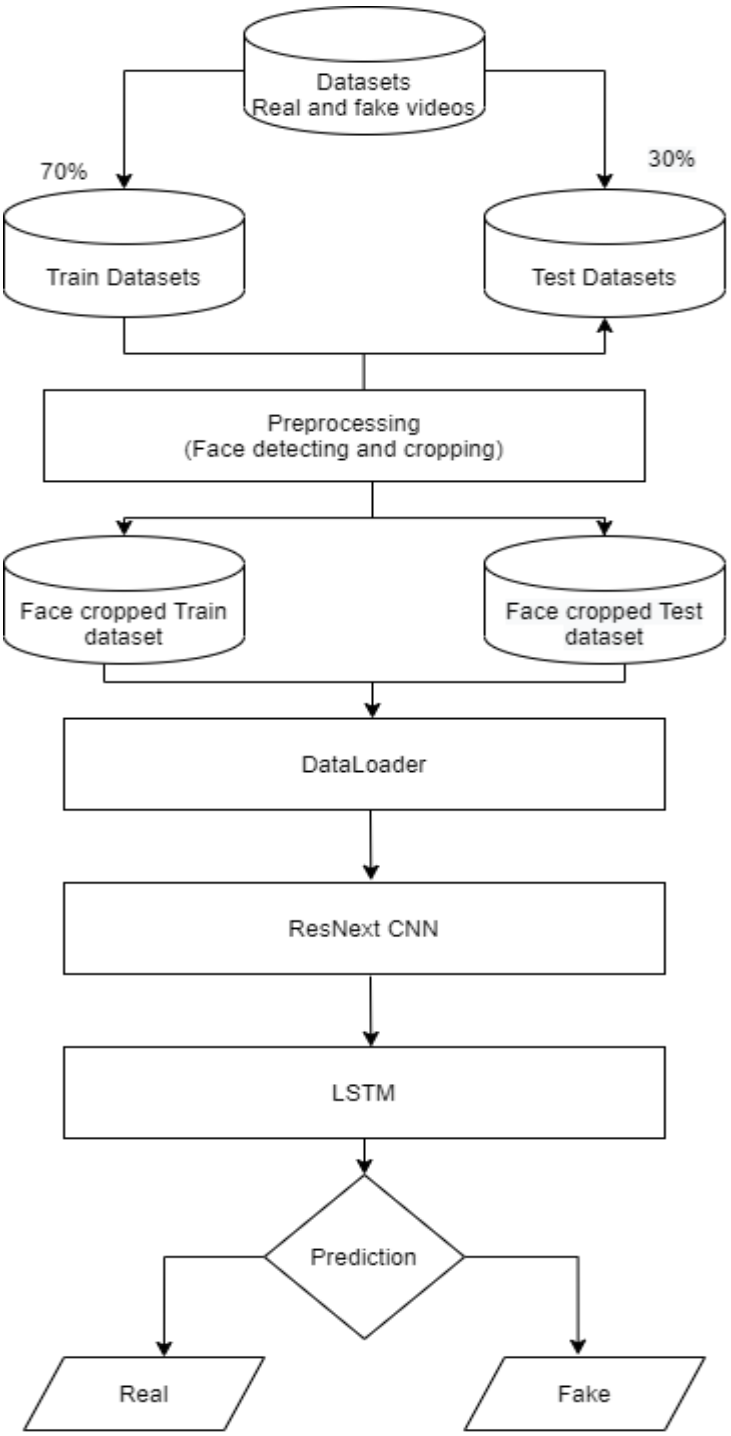

Fig. 4.2.2: Training Flow for Deepfake Detection

The output of the model is going to be whether the video is deepfake or a real video along with the confidence of the model. From fig. 4.2.2

4.3 UseCase Diagram

1) UML Class diagram for Deepfake Creation and Detection of the proposed system

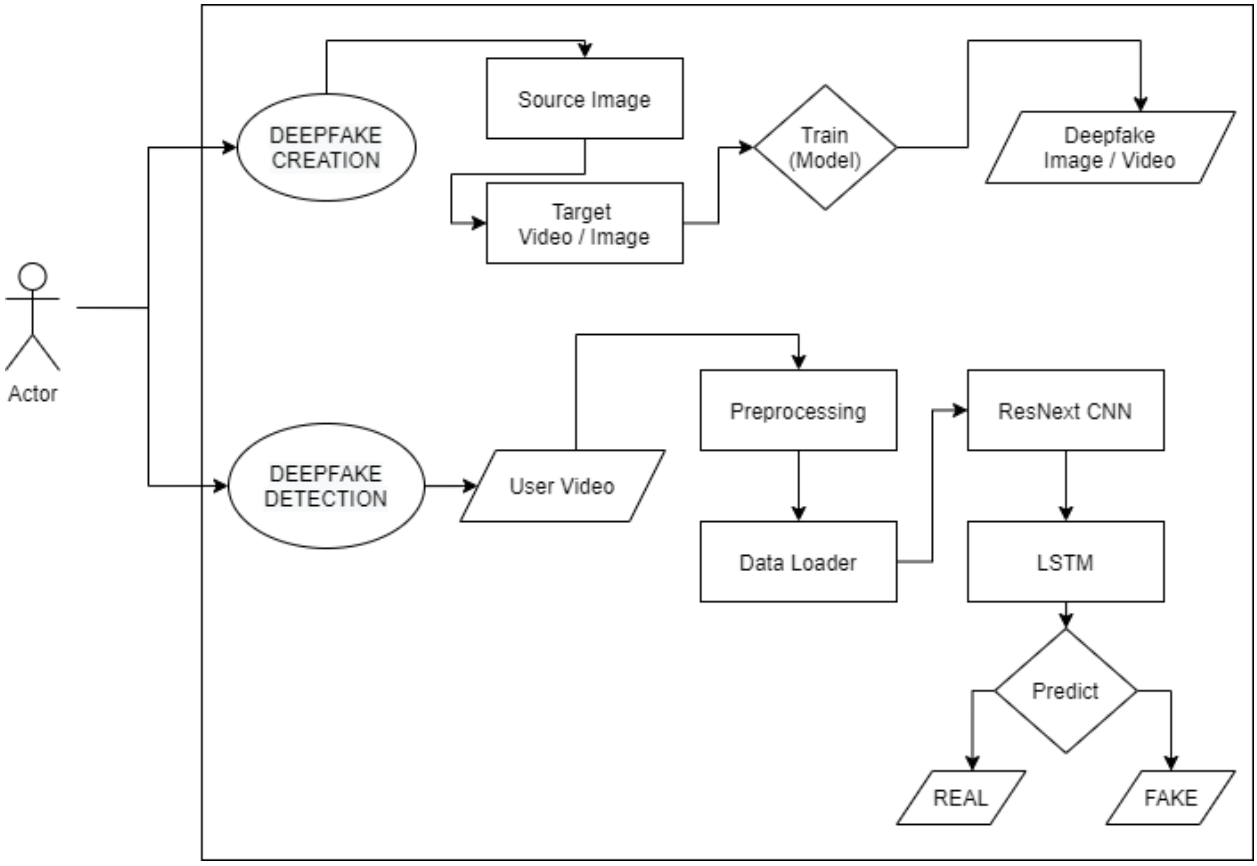

Fig. 4.3.1: UseCase Diagram for deep fake creation and detection

For Deepfake Creation, the user can upload the image from the user's system directory and then select the video for deepfake ie; the target video then the model will start training and the reconstructed video will result from the user source image.

For Deepfake Detection, the user uploads any video from the user's system directory, then the model starts pre-processing the video and splits it into frames, then loads it into a data loader. Then ResNext CNN will classify the image whether it's original or fake, then move it to LSTM, then output of the model is going to be whether the video is deepfake or a real video.

CHAPTER 5

IMPLEMENTATION

5.1 Techniques

Deepfakes rely on a type of neural network called an autoencoder. These consist of an encoder, which reduces an image to a lower dimensional latent space, and a decoder, which reconstructs the image from the latent representation. Deepfakes utilize this architecture by having a universal encoder which encodes a person into the latent space. The latent representation contains key features about their facial features and body posture. This can then be decoded with a model trained specifically for the target. This means the target's detailed information will be superimposed on the underlying facial and body features of the original video, represented in the latent space.

A popular upgrade to this architecture attaches a generative adversarial network to the decoder. A GAN trains a generator, in this case the decoder, and a discriminator in an adversarial relationship.The generator creates new images from the latent representation of the source material, while the discriminator attempts to determine whether or not the image is generated. This causes the generator to create images that mimic reality extremely well as any defects would be caught by the discriminator.Both algorithms improve constantly in a zero sum game.This makes deepfakes difficult to combat as they are constantly evolving; any time a defect is determined, it can be corrected.

Given a training set, this technique learns to generate new data with the same statistics as the training set.

Fig. 5.1.1: Deepfake technology used to create facial morphing

5.2 Model Training

A training model is a dataset that is used to train an ML algorithm. It consists of the sample output data and the corresponding sets of input data that have an influence on the output. The training model is used to run the input data through the algorithm to correlate the processed output against the sample output. The result from this correlation is used to modify the model.

This iterative process is called “model fitting”. The accuracy of the training dataset or the validation dataset is critical for the precision of the model.

There are several types of machine learning models, of which the most common ones are supervised and unsupervised learning. The machine learning model needs the outcomes to determine the features that best predict the outcomes.

During the training process, the data are sorted by outcomes and the algorithm extracts statistical patterns to build the model.

5.3 Developing Deep Learning Models in PyTorch

Before training a model we need some basic setup for the PyTorch library that allows efficient computation and automatic differentiation on graph-based models.

PyTorch is an open-source Python library for deep learning developed and maintained by Facebook. Torch (*Torch7*) is an open-source project for deep learning written in C and generally used via the Lua interface.

The PyTorch API is simple and flexible, making it a favorite for academics and researchers in the development of new deep learning models and applications. The extensive use has led to many extensions for specific applications (such as text, computer vision, and audio data), and many pre-trained models that can be used directly.

The flexibility of PyTorch comes at the cost of ease of use, especially for beginners, as compared to simpler interfaces like Keras. The choice to use PyTorch instead of Keras gives up some ease of use, a slightly steeper learning curve, and more code for more flexibility, and perhaps a more vibrant academic community.

5.3.1 Steps to install and run PyTorch

Before installing PyTorch, ensure that you have Python installed, such as Python 3.6 or higher.

- PyTorch installation with PIP for CPU

```
pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu
torchaudio===0.8.1 -f
https://download.pytorch.org/whl/torch_stable.html
```

- PyTorch installation with PIP for GPU 10.2

```
pip3 install torch==1.8.1+cu102 torchvision==0.9.1+cu102
torchaudio===0.8.1 -f https://download.pytorch.org/whl/torch_stable.html
```

- PyTorch installation with PIP for GPU 11.1

```
pip3 install torch==1.8.1+cu111 torchvision==0.9.1+cu111
torchaudio===0.8.1 -f https://download.pytorch.org/whl/torch_stable.html
```

- To verify installation

```
import torch
print(torch.__version__)
```

- Save the file, then open your command line and change directory to where you saved the file. Then type

```
python versions.py
```

- You should then see output like the following:

```
1.7.1+cpu
```

- This confirms that PyTorch is installed correctly and that we are all using the same version.

5.4 Pseudocode

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm.

It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are not essential for human understanding of the algorithm, such as variable declarations, system-specific code and some subroutines. The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation.

The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm. It is commonly used in textbooks and scientific publications that are documenting various algorithms, and also in planning of computer program development, for sketching out the structure of the program before the actual coding takes place.

No standard for pseudocode syntax exists, as a program in pseudocode is not an executable program. Pseudocode resembles, but should not be confused with skeleton programs, including dummy code, which can be compiled without errors. Flowcharts and Unified Modeling Language (UML) charts can be thought of as a graphical alternative to pseudo code, but are more spacious on paper.

Directory Structure

For ease of understanding the project is structured in below format

```
DEEPFAKE_CREATION_&_ DETECTION_USING_CYCLE_GANs
  |
  |--- Django Application
  |--- Model Creation
       |--- Deepfake_Creation_Model
       |--- Deepfake_Detection_Model
```

1. Django Application: This directory consists of the django made application of our work. Where a user can upload the video and submit it to the model for deepfake creation & detection prediction. The trained model performs the prediction and the result is displayed on the screen.
2. Model Creation: This directory consists of the step by step process of creating and training a deepfake creation & detection model using our approach.

Below shows the structure for this project:

- DEEPFAKE_CREATION_&_ DETECTION_USING_CYCLE_GANs
  - Django_Application
    - ml_app
      - templates
        - index.html
        - detection.html
        - creation.html
        - about.html
        - 404.html
      - __init__.py
      - admin.py
      - apps.py
      - forms.py
      - models.py
      - tests.py
      - urls.py
      - views.py
    - models
      - trained_model_1.pt
      - trained_model_2.pt
      - trained_model_3.pt
      - trained_model_4.pt
    - project_setting
      - __init__.py
      - asgi.py
      - settings.py
      - urls.py
      - wsgi.py
    - static
      - bootstrap
        - bootstrap.min.css
      - css
        - style.css
      - images
        - all_images
      - js
        - scripts.js
        - main.js
      - json
    - templets
      - base.html
    - uploaded_images
    - uploaded_videos
    - manage.py
    - Model_Creation
      - Deepfake_Creation_model
      - Deepfake_Detection_model

5.5 Model Creation

- We will preprocess the dataset, train a pytorch model, predict on new unseen data using a trained model.We used Google Colab for training the models.

Datasets

Some of the dataset we used are listed below:

1. FaceForensics++: FaceForensics++ is a forensics dataset consisting of 1000 original video sequences that have been manipulated with four automated face manipulation methods: Deepfakes, Face2Face, FaceSwap and NeuralTextures. The data has been sourced from 977 youtube videos and all videos contain a trackable mostly frontal face without occlusions which enables automated tampering methods to generate realistic forgeries.
2. Celeb-DF: Celeb-DF dataset includes 590 original videos collected from YouTube with subjects of different ages, ethic groups and genders, and 5639 corresponding DeepFake videos.
3. Deepfake Detection Challenge: A deepfake could be either a face or voice swap (or both). In the training data, this is denoted by the string "REAL" or "FAKE" in the label column. In your submission, you will predict the probability that the video is a fake.

Preprocessing

- Load the dataset
- Split the video into frames
- Crop the face from each frame
- Save the face cropped video

Model and train

- It will load the preprocessed video and labels from a csv file.
- Create a pytorch model using transfer learning with RestNext50 and LSTM.
- Split the data into train and test data
- Train the model
- Test the model
- Save the model in .pt file

Predict

- Load the saved pytorch model
- Predict the output based on trained weights.

Pseudocode for model prediction:

In django, the piece of code that handles a specific URL is called a view. A view function, or *view* for short, is a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image etc.

views.py file is the file where we write our whole logic about what data should come from db and what data should serve from views to server. views are mapped with urls.py file.

Django_application/ml_app/views.py: (pseudocode)

```
from django.shortcuts import render, redirect
import torch
import torchvision
from torchvision import transforms, models
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 torch.autograd import Variable
import time
import sys
from torch import nn
import json
import glob
import copy
from torchvision import models
import shutil
from PIL import Image as pImage
import time
from django.conf import settings
from .forms import VideoUploadForm

index_template_name = 'index.html'
predict_template_name = 'predict.html'

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))
```

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

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,:]))


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
        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_convert(tensor, video_file_name):
    """ 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)
    # This image is not used
    # cv2.imwrite(os.path.join(settings.PROJECT_DIR, 'uploaded_images',
video_file_name+'_convert_2.png'),image*255)
    return image

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


def predict(model,img,path = './', video_file_name=""):
  fmap,logits = model(img.to('cuda'))
  img = im_convert(img[:,-1,:,:,:], video_file_name)
  params = list(model.parameters())
  weight_softmax = model.linear1.weight.detach().cpu().numpy()
  logits = sm(logits) _,prediction = torch.max(logits,1)
```

```
confidence = logits[:,int(prediction.item())].item()*100
  print('confidence of
prediction:',logits[:,int(prediction.item())].item()*100)
  return [int(prediction.item()),confidence]

def plot_heat_map(i, model, img, path = './', video_file_name=''):
  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)
  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)
  out = np.dot(fmap[i].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,:,:,:], video_file_name)
  result = heatmap * 0.5 + img*0.8*255
  # Saving heatmap - Start
  heatmap_name = video_file_name+"_heatmap_"+str(i)+".png"
  image_name = os.path.join(settings.PROJECT_DIR, 'uploaded_images',
heatmap_name)
  cv2.imwrite(image_name,result)
  # Saving heatmap - End
  result1 = heatmap * 0.5/255 + img*0.8
  r,g,b = cv2.split(result1)
  result1 = cv2.merge((r,g,b))
  return image_name

# Model Selection
def get_accurate_model(sequence_length):
    model_name = []
    sequence_model = []
    final_model = ""
    list_models = glob.glob(os.path.join(settings.PROJECT_DIR, "models",
"*.pt"))
    for i in list_models:
        model_name.append(i.split("\\")[-1])
    for i in model_name:
        try:
            seq = i.split("_")[3]
```

```
            if (int(seq) == sequence_length):
                    sequence_model.append(i)
            except:
                pass

    if len(sequence_model) > 1:
        accuracy = []
        for i in sequence_model:
            acc = i.split("_")[1]
            accuracy.append(acc)
        max_index = accuracy.index(max(accuracy))
        final_model = sequence_model[max_index]
    else:
        final_model = sequence_model[0]
    return final_model

ALLOWED_VIDEO_EXTENSIONS =
set(['mp4','gif','webm','avi','3gp','wmv','flv','mkv'])

def allowed_video_file(filename):
    #print("filename" ,filename.rsplit('.',1)[1].lower())
    if (filename.rsplit('.',1)[1].lower() in ALLOWED_VIDEO_EXTENSIONS):
        return True
    else:
        return False
def index(request):
    if request.method == 'GET':
        video_upload_form = VideoUploadForm()
        if 'file_name' in request.session:
            del request.session['file_name']
        if 'preprocessed_images' in request.session:
            del request.session['preprocessed_images']
        if 'faces_cropped_images' in request.session:
            del request.session['faces_cropped_images']
        return render(request, index_template_name, {"form":
video_upload_form})
    else:
        video_upload_form = VideoUploadForm(request.POST, request.FILES)
        if video_upload_form.is_valid():
            video_file = video_upload_form.cleaned_data['upload_video_file']
            video_file_ext = video_file.name.split('.')[-1]
            sequence_length =
video_upload_form.cleaned_data['sequence_length']
            video_content_type = video_file.content_type.split('/')[0]
            if video_content_type in settings.CONTENT_TYPES:
                if video_file.size > int(settings.MAX_UPLOAD_SIZE):
                    video_upload_form.add_error("upload_video_file",
"Maximum file size 100 MB")
```

```
                return render(request, index_template_name, {"form":
            video_upload_form})
              if sequence_length <= 0:
                video_upload_form.add_error("sequence_length", "Sequence
Length must be greater than 0")
                return render(request, index_template_name, {"form":
video_upload_form})

              if allowed_video_file(video_file.name) == False:
                video_upload_form.add_error("upload_video_file","Only video
files are allowed ")
                return render(request, index_template_name, {"form":
video_upload_form})

              saved_video_file =
'uploaded_file_'+str(int(time.time()))+"."+video_file_ext
              with open(os.path.join(settings.PROJECT_DIR, 'uploaded_videos',
saved_video_file), 'wb') as vFile:
                shutil.copyfileobj(video_file, vFile)

              request.session['file_name'] =
os.path.join(settings.PROJECT_DIR, 'uploaded_videos', saved_video_file)
              request.session['sequence_length'] = sequence_length
              return redirect('ml_app:predict')
          else:
              return render(request, index_template_name, {"form":
video_upload_form})

def predict_page(request):
    if request.method == "GET":
          if 'file_name' not in request.session:
              return redirect("ml_app:home")
          if 'file_name' in request.session:
              video_file = request.session['file_name']
          if 'sequence_length' in request.session:
              sequence_length = request.session['sequence_length']
          path_to_videos = [video_file]
          video_file_name = video_file.split('\\')[-1]
          video_file_name_only = video_file_name.split('.')[0]
          video_dataset = validation_dataset(path_to_videos,
sequence_length=sequence_length,transform= train_transforms)
          model = Model(2).cuda()
          model_name = os.path.join(settings.PROJECT_DIR,'models',
get_accurate_model(sequence_length))
          models_location = os.path.join(settings.PROJECT_DIR,'models')
          path_to_model = os.path.join(settings.PROJECT_DIR, model_name)
          model.load_state_dict(torch.load(path_to_model))
          model.eval()
          start_time = time.time()
          # Start: Displaying preprocessing images
```

```
        print("<=== | Started Videos Splitting | ===>")
        preprocessed_images = []
        faces_cropped_images = []
        cap = cv2.VideoCapture(video_file)

        frames = []
        while(cap.isOpened()):
            ret, frame = cap.read()
            if ret==True:
                frames.append(frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                break
        cap.release()

        for i in range(1, sequence_length+1):
            frame = frames[i]
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = pImage.fromarray(image, 'RGB')
            image_name = video_file_name_only+"_preprocessed_"+str(i)+'.png'
            image_path = os.path.join(settings.PROJECT_DIR,
'uploaded_images', image_name)
            img.save(image_path)
            preprocessed_images.append(image_name)
        print("<=== | Videos Splitting Done | ===>")
        print("--- %s seconds ---" % (time.time() - start_time))
        # End: Displaying preprocessing images

        # Start: Displaying Faces Cropped Images
        print("<=== | Started Face Cropping Each Frame | ===>")
        padding = 40
        faces_found = 0
        for i in range(1, sequence_length+1):
            frame = frames[i]
            #fig, ax = plt.subplots(1,1, figsize=(5, 5))
            face_locations = face_recognition.face_locations(frame)
            if len(face_locations) == 0:
                continue
            top, right, bottom, left = face_locations[0]
            frame_face = frame[top-padding:bottom+padding,
left-padding:right+padding]
            image = cv2.cvtColor(frame_face, cv2.COLOR_BGR2RGB)

            img = pImage.fromarray(image, 'RGB')
            image_name =
video_file_name_only+"_cropped_faces_"+str(i)+'.png'
```

```
                image_path = os.path.join(settings.PROJECT_DIR,
        'uploaded_images',
        video_file_name_only+"_cropped_faces_"+str(i)+'.png')
                img.save(image_path)
                faces_found = faces_found + 1
                faces_cropped_images.append(image_name)
            print("<=== | Face Cropping Each Frame Done | ===>")
            print("--- %s seconds ---" % (time.time() - start_time))

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

            # End: Displaying Faces Cropped Images
            try:
                heatmap_images = []
                for i in range(0, len(path_to_videos)):
                    output = ""
                    print("<=== | Started Prediction | ===>")
                    prediction = predict(model, video_dataset[i], './',
video_file_name_only)
                    confidence = round(prediction[1], 1)
                    print("<=== |  Prediction Done | ===>")
                    # print("<=== | Heat map creation started | ===>")
                    # for j in range(0, sequence_length):
                    #     heatmap_images.append(plot_heat_map(j, model,
video_dataset[i], './', video_file_name_only))
                    if prediction[0] == 1:
                        output = "REAL"
                    else:
                        output = "FAKE"
                    print("Prediction : " , prediction[0],"==",output
,"Confidence : " , confidence)
                    print("--- %s seconds ---" % (time.time() - start_time))
                return render(request, predict_template_name,
{'preprocessed_images': preprocessed_images, 'heatmap_images':
heatmap_images, "faces_cropped_images": faces_cropped_images,
"original_video": video_file_name, "models_location": models_location,
"output": output, "confidence": confidence})
            except:
                return render(request, 'cuda_full.html')
    def about(request):
        return render(request, about_template_name)
    def handler404(request,exception):
        return render(request, '404.html', status=404)
    def cuda_full(request):
        return render(request, 'cuda_full.html')
```

CHAPTER 6

TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. The system has been verified and validated by running the test data and live data.

6.1 Levels of Testing

6.1.1 Unit Testing

Unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In object-oriented programming a unit is often an entire interface, such as a class, but could be an individual method.

For unit testing first we adopted the code testing strategy, which examined the logic of the program. During the development process itself all the syntax errors etc. got rooted out. For this developed test case that resulted in executing every instruction in the program or module i.e. every path through the program was tested. Test cases are data chosen at random to check every possible branch after all the loops.

6.1.1.1 Test Cases

Steps	Test Actions	Results
Step 01	Django command for running server : python manage.py runserver	Starting development server and loaded successful without errors
Step 02	Click on URL http://127.0.0.1:8000/	Main page(index page) loaded successfully
Step 03	Checking loaded django web-server	All pages are working and scrollable
Step 04	Checking hyperlinked buttons	Buttons are working & loaded web pages successfully
Step 05	Checking Deepfake Creation page	Page loaded successfully
Step 06	Checking for uploading images & videos	Images & videos are uploaded successfully
Step 07	Uploaded the sample image and click on Run	Trained model started
Step 08	Waiting for Results	Output video displayed successfully
Step 09	Checking the result	Model created deepfake video
Step 10	Verifying deepfake video if the result is correct	Yes it is correct
Step 11	Checking Deepfake Detection page	Page loaded successfully
Step 12	Checking for suitable uploading videos	Videos uploaded successfully
Step 13	Uploaded the sample video and click on predict	Trained model started for prediction
Step 14	Checking videos for frame splits	Video splits into images successfully
Step 15	Checking the result whether it is real/fake video	Model predicted result successfully
Step 16	Verifying if result is correct	Yes it is Correct

Table 6.1.1.1: Test cases for the project

6.1.1.2 Error Handling

In this system, we have tried to handle all the errors that occurred while running the application. The common errors we saw were reading a tuple with an attribute set to null and database connection getting lost.

For Testing we used Top-Down design, a decomposition process which focuses on the flow of control, while latter strategies concern itself with code production. The first step is to study the overall aspects of the tasks at hand and break it into a number of independent modules. The second step is to break one of these modules further into independent sub modules. One of the important features is that each level the details at lower levels are hidden. So unit testing was performed first and then system testing.

6.1.2 Integration Testing

Data can be lost across an interface, one module can have an adverse effect on the other sub function, when combined may not produce the desired functions. Integrated testing is the systematic testing to uncover the errors with an interface. This testing is done with simple data and the developed system has run successfully with this simple data. The need for an integrated system is to find the overall system performance.

Steps to perform integration testing:

Step 1: Create a Test Plan
Step 2: Create Test Cases and Test Data
Step 3: Once the components have been integrated execute the test cases
Step 4: Fix the bugs if any and re test the code
Step 5: Repeat the test cycle until the components have been successfully integrated

Name of the Test	**Integration testing**
Test plan	To check whether the system works properly when all the modules are integrated.
Test data	Sample Image and Videos

Table 6.1.2: Test cases for integration testing

6.1.3 System Testing

Ultimately, software is included with other system components and the set of system validation and integration tests are performed. System testing is a series of different tests whose main aim is to fully exercise the computer-based system. Although each test has a different role all work should verify that all system elements are properly integrated and formed allocated functions.

Name of the Test	**System Testing**
Items being tested	Overall functioning of web pages with all functions properly linked.
Sample Inputs	Sample Image
Expected Outputs	All the modules are working as expected
Actual Output	Application reacts to user inputs in expected manner.
Remarks	Successful

Table 6.1.3: Test cases for system testing

6.1.4 Validation Testing

At the culmination of black box testing, software is completely assembled as a package. Interfacing errors have been uncovered and the correct and final series of tests, i.e., validation tests begins. Validation test is defined with a simple definition that validation succeeds when the software functions in a manner that can be reasonably accepted by the customer.

6.1.5 Output Testing

After performing validation testing, the next step is output testing of the proposed system. Since the system cannot be useful if it does not produce the required output. Asking the user about the format in which the system is required tests the output displayed or generated by the system and tests the output displayed or generated by the system under consideration. The output format is considered in two ways, one is on screen format and the other is printed format. The output format on the screen is found to be corrected as the format was designated in the system according to the user needs. As for the hard copy the output comes according to the specification requested by the user. The output testing does not result in any correction in the system.

6.1.6 Test data and Output:

Taking various kinds of soft data plays a vital role in system testing. After preparing the test data system under study is tested using the test data. While testing, errors are again uncovered and corrected by using the above steps and corrections are also noted for future use.

6.1.7 User acceptance Testing

User acceptance testing of the system is the key factor for the success of the system. A system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system at the time of development and making changes whenever required. This is done with regard to the input screen design and output screen design.

6.1.8 GUI Testing

GUI testing is used to ensure the visual clarity of the system, flexibility of the system, user friendliness of the system. The various components which are to be tested are:

- Relative layout
- Various Links and Buttons

CHAPTER 7

RESULTS

- Deepfakes have begun to erode trust of people in media contents as seeing them is no longer commensurate with believing in them. They could cause distress and negative effects to those targeted, heighten disinformation and hate speech, and even could stimulate political tension, inflame the public, violence or war. This is especially critical nowadays as the technologies for creating deepfakes are increasingly approachable and social media platforms can spread those fake contents quickly. Sometimes deepfakes do not need to be spread to a massive audience to cause detrimental effects. People who create deepfakes with malicious purposes only need to deliver them to target audiences as part of their sabotage strategy without using social media.

- On the other hand, current detection methods mostly focus on drawbacks of the deepfake generation pipelines, i.e. finding weakness of the competitors to attack them. This kind of information and knowledge is not always available in adversarial environments where attackers commonly attempt not to reveal such deepfake creation technologies. Recent works on adversarial perturbation attacks to fool DNN-based detectors make the deepfake detection task more difficult

- Another research direction is to integrate detection methods into distribution platforms such as social media to increase its effectiveness in dealing with the widespread impact of deepfakes. The screening or filtering mechanism using effective detection methods can be implemented on these platforms to ease the deepfakes detection

- Using detection methods to spot deepfakes is crucial, but understanding the real intent of people publishing deepfakes is even more important. This requires the judgement of users based on social context in which deepfake is discovered, e.g. who distributed it and what they said about it

7.1 Screenshots

7.1.1 Startup Page

Fig 7.1.1 Starter Page

7.1.2 About Page

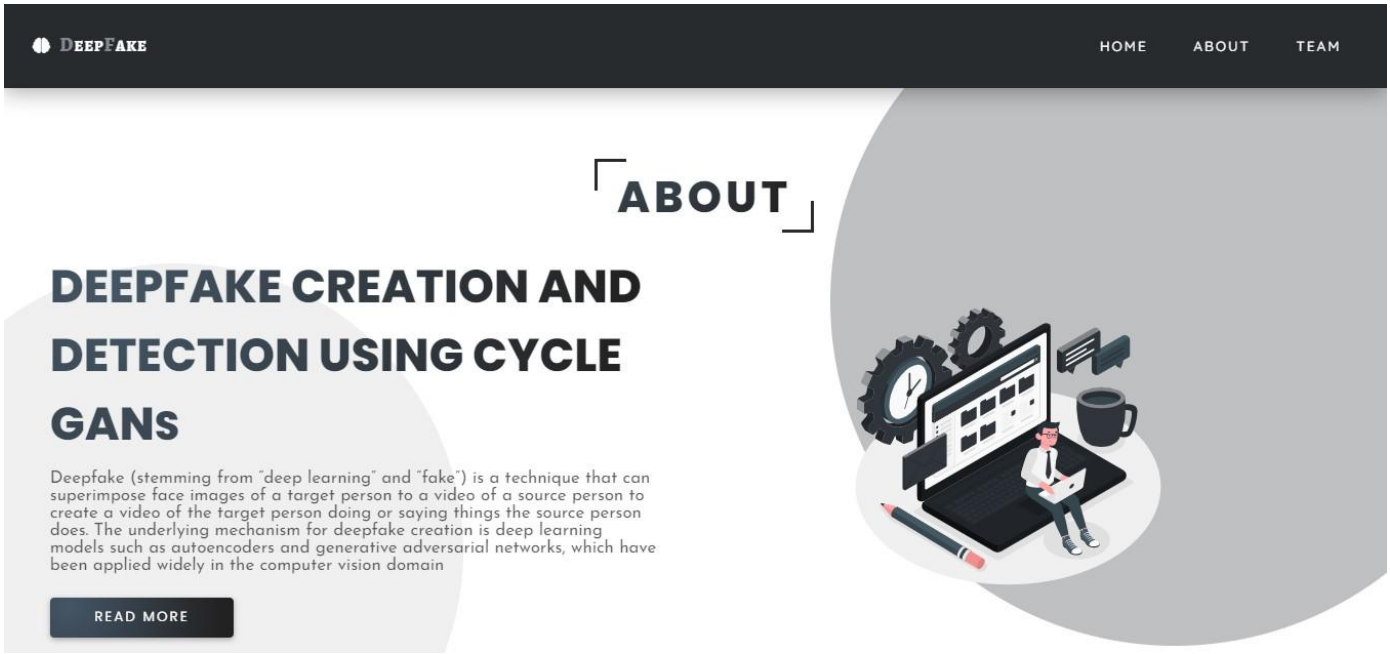

Fig 7.1.2 About Page

7.1.3 Deepfake Creation Page

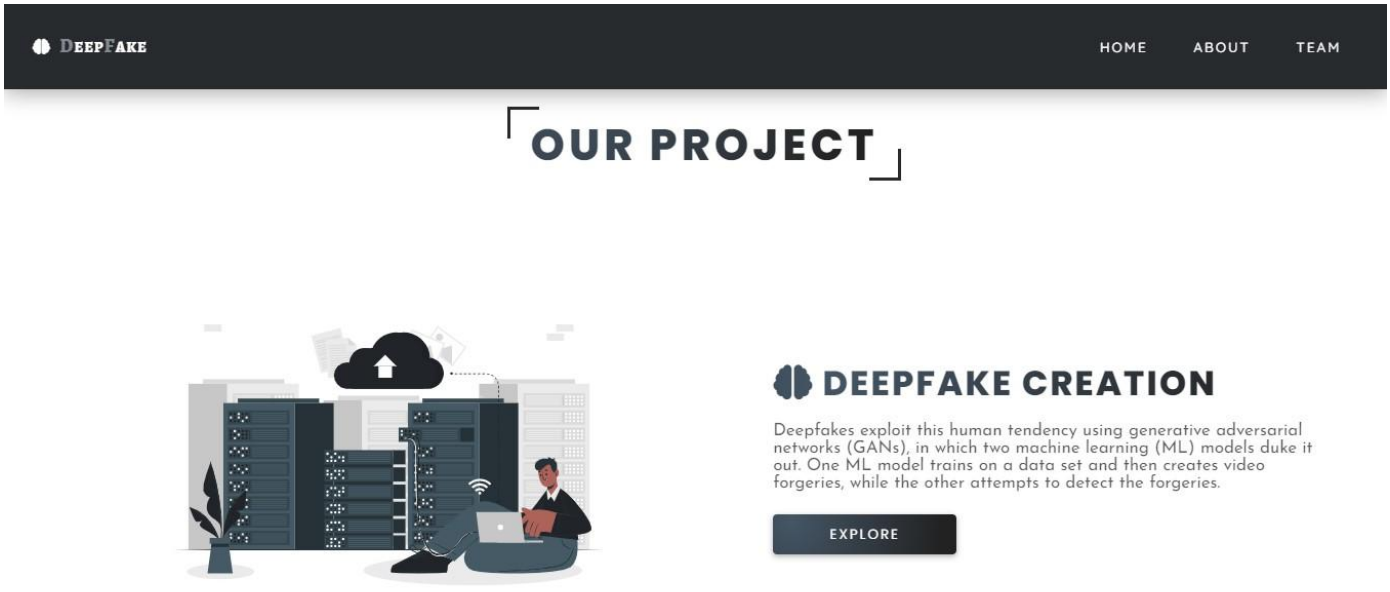

Fig 7.1.3 Deepfake Creation Page

7.1.4 Deepfake Detection Page

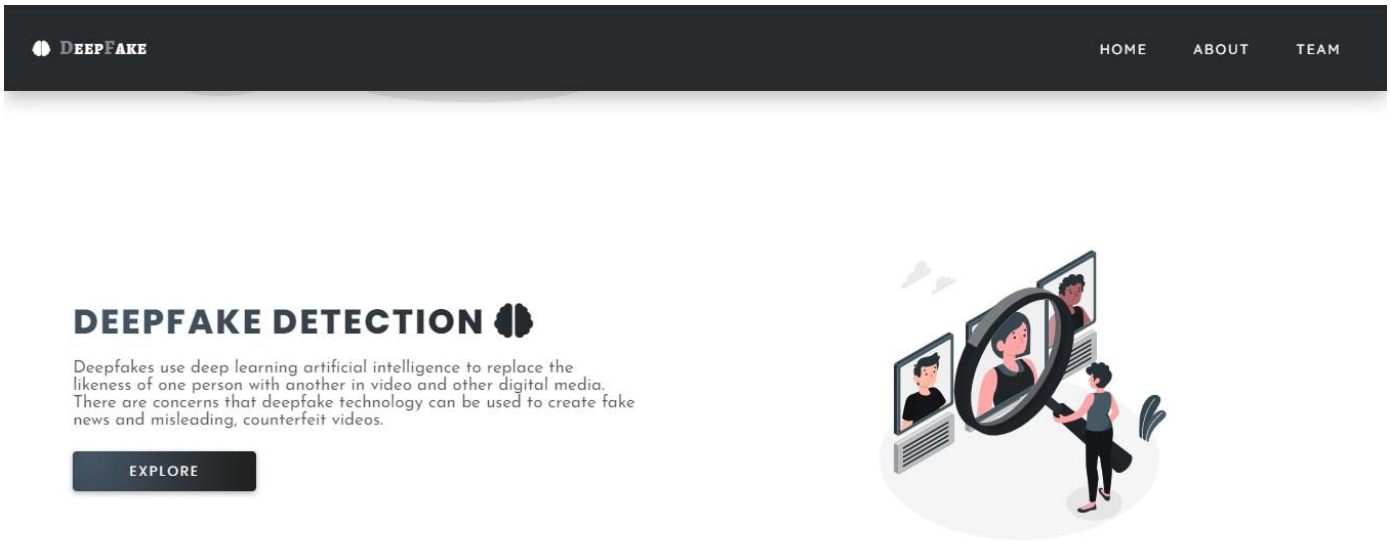

Fig 7.1.4 Deepfake Detection Page

7.1.5 Deepfake Creation for Source Image & Destination Video uploading page

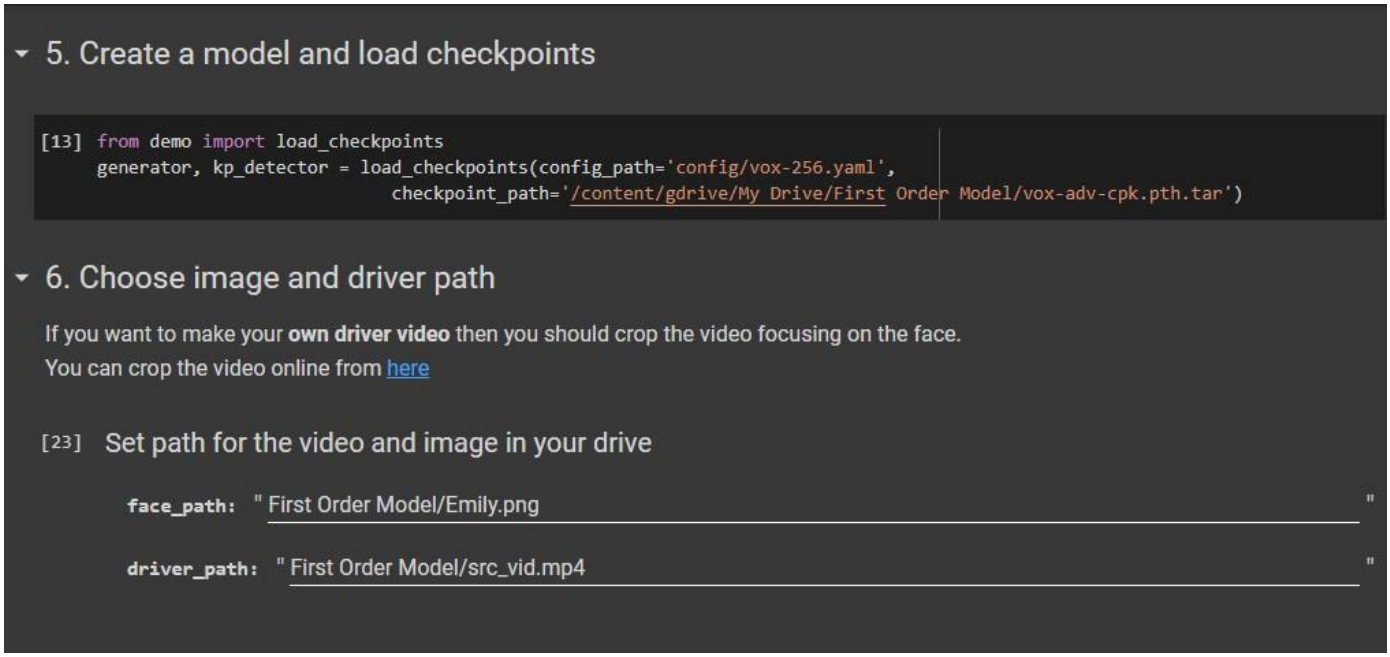

Fig 7.1.5 Deepfake Creation for Source Image & Destination Video uploading page

7.1.6 Deepfake Creation Output Page

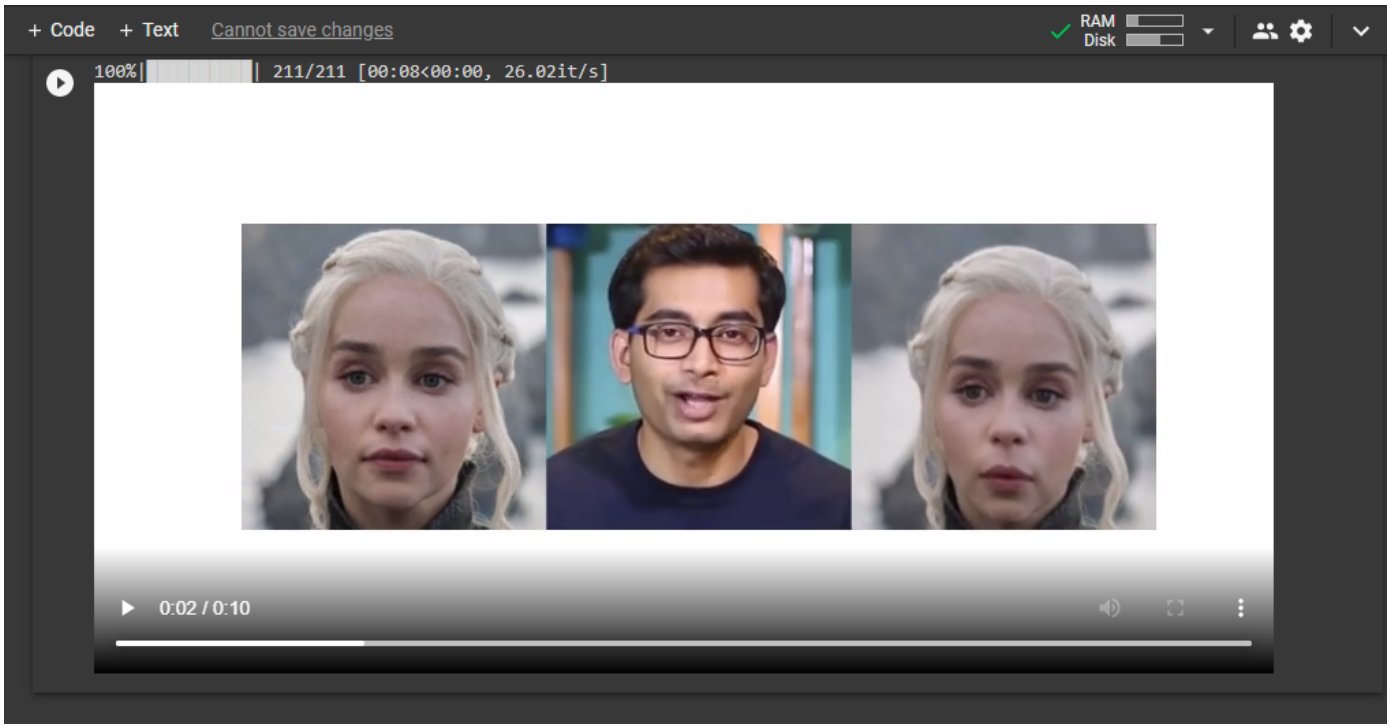

Fig 7.1.6 Deepfake Creation Output Page

7.1.7 Deepfake Detection Video Uploading Page

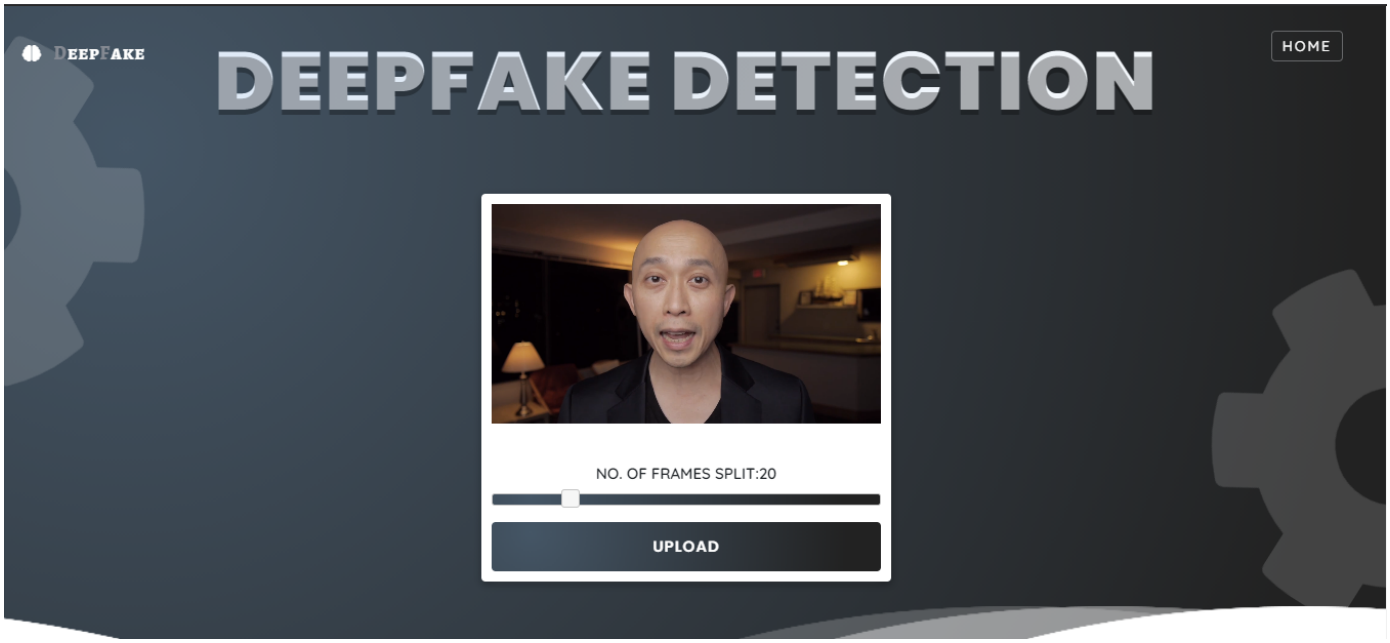

Fig 7.1.7 Deepfake Detection Video Uploading Page

7.1.8 Deepfake Detection Frame Splitting Page

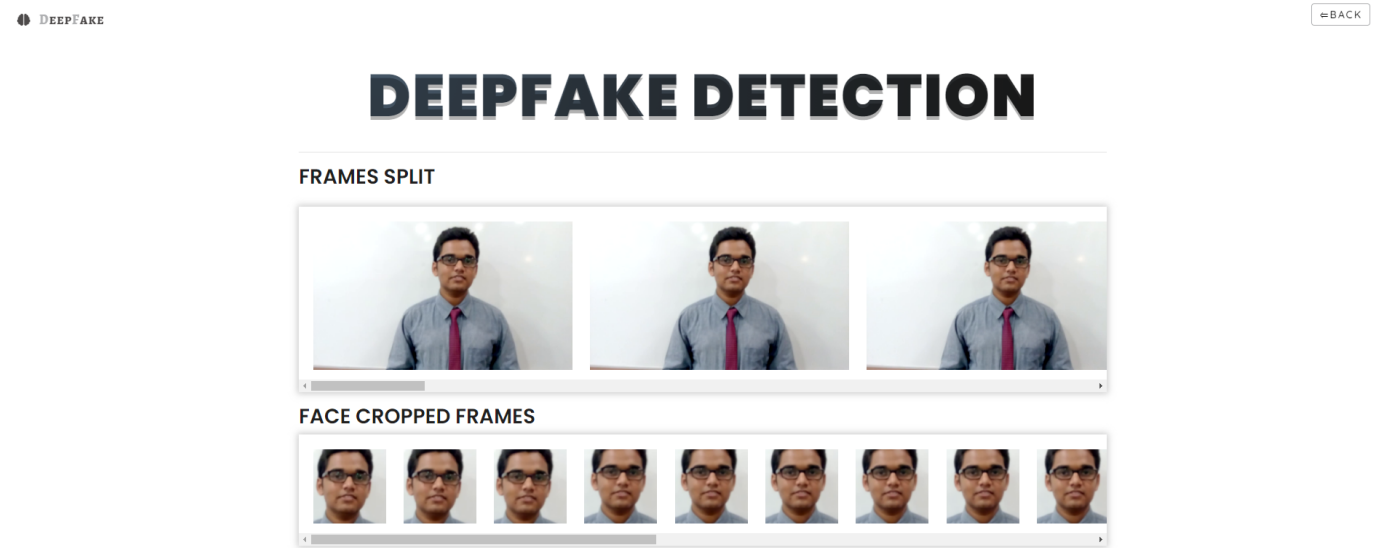

Fig 7.1.8 Deepfake Detection Frame Splitting Page

7.1.9 Deepfake Detection Prediction Page for Real Video

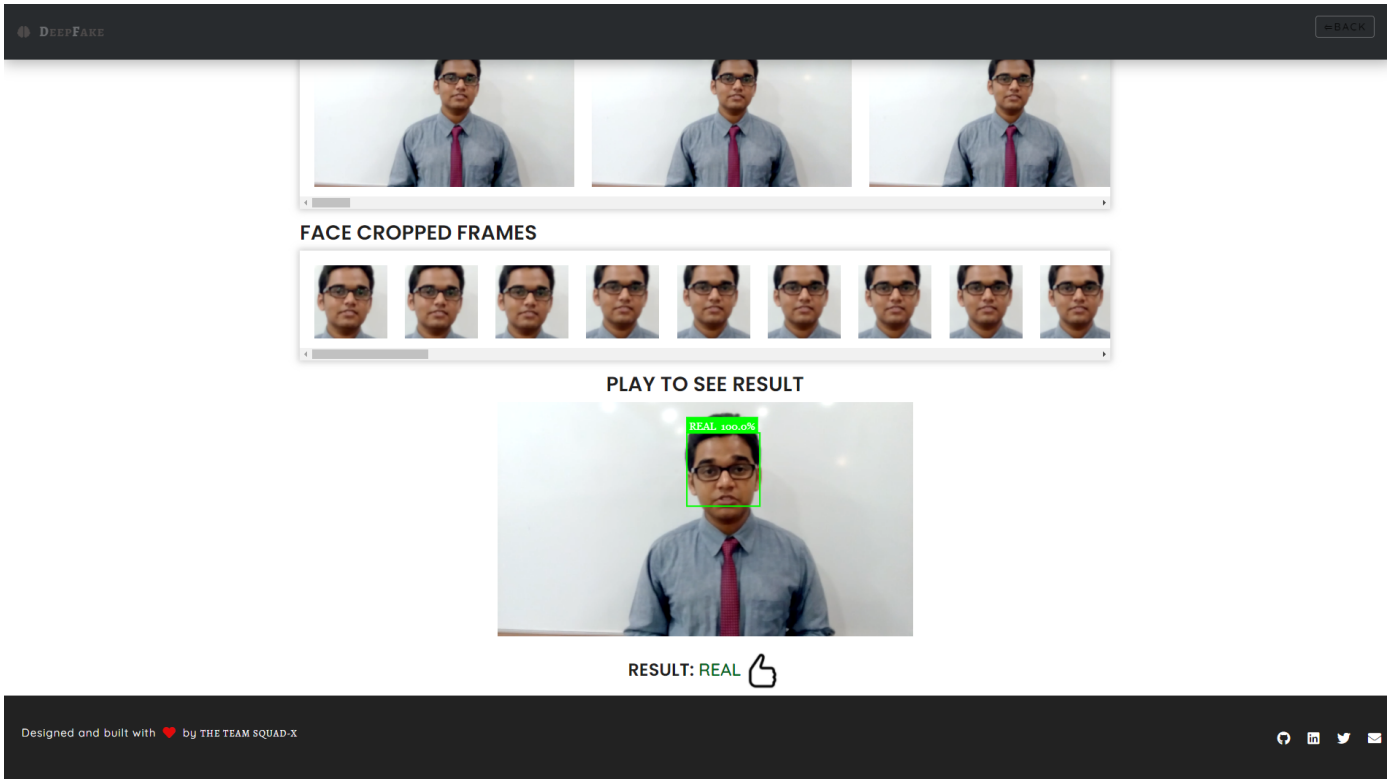

Fig 7.1.9 Deepfake Detection Prediction Page for Real Video

7.1.10 Deepfake Detection Prediction Page for Fake Video

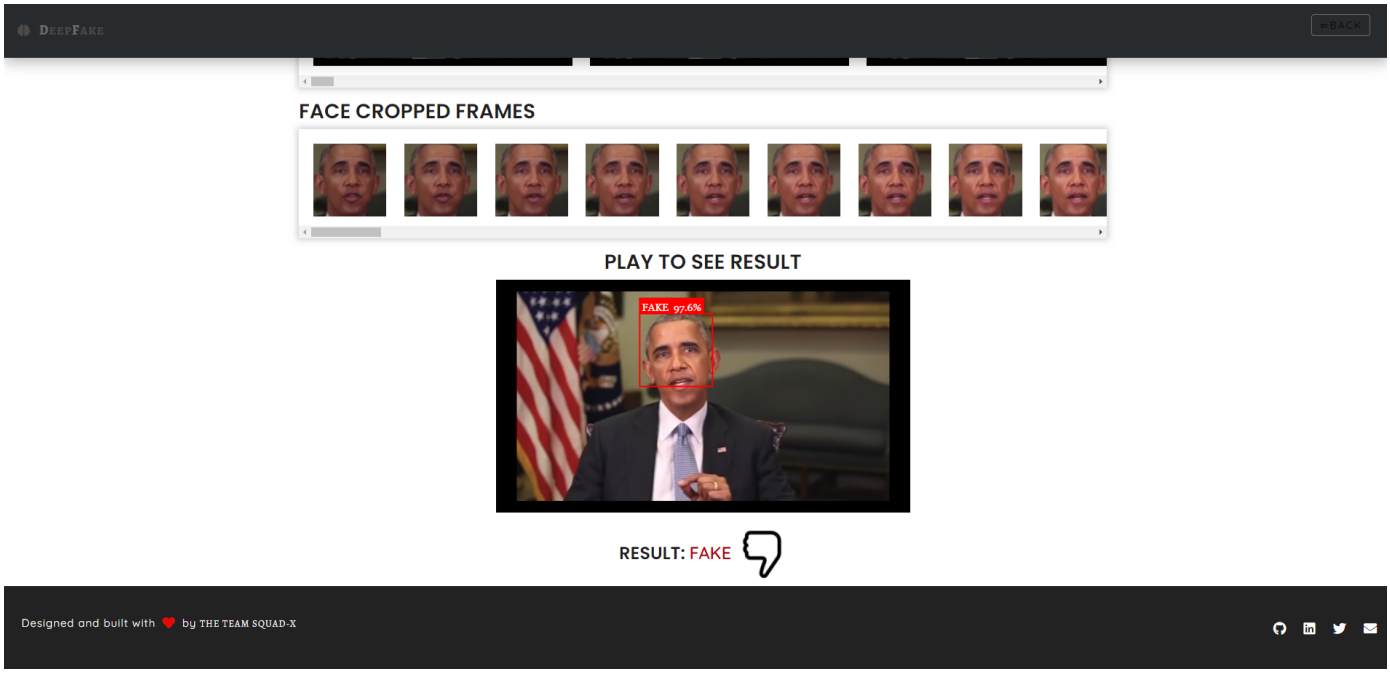

Fig 7.1.10 Deepfake Detection Prediction Page for Fake Video

7.1.11 Our Team Page

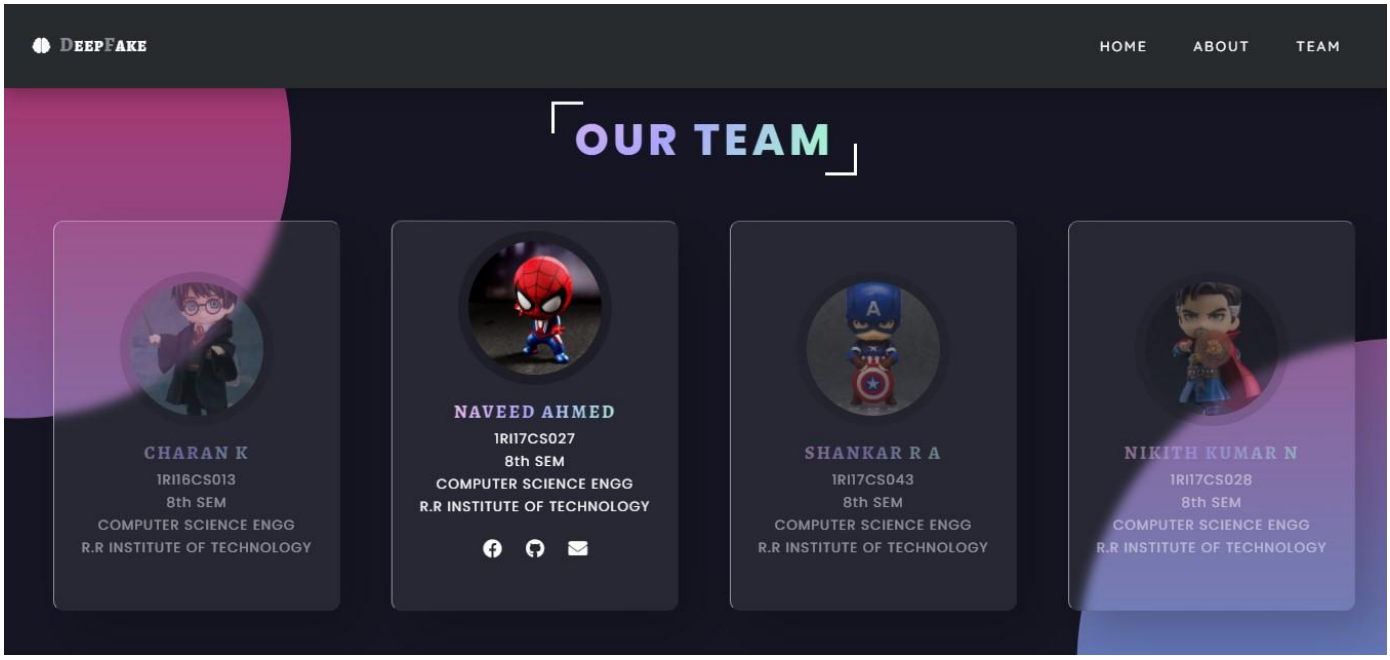

Fig 7.1.11 Our Team Page

CHAPTER 8

CONCLUSION

- Deepfakes' quality has been increasing and the performance of detection methods needs to be improved accordingly. The inspiration is that what AI has broken can be fixed by AI as well. Detection methods are still in their early stage and various methods have been proposed and evaluated but using fragmented data sets. An approach to improve performance of detection methods is to create a growing updated benchmark data set of deepfakes to validate the ongoing development of detection methods. This will facilitate the training process of detection models, especially those based on deep learning, which requires a large training set

- The project is designed with a neural network-based approach to classifying the video as deep fake or real. The proposed method is inspired by the way in which deep fakes are created by GANs with the help of Autoencoders.

- Our method is used to detect frame level using ResNext CNN and to classify video using RNN along with LSTM. The proposed method is capable of detecting a video as a deep fake or real based on the parameters.

- A web based platform for the user to upload the video and classify it as fake or real as well as the user can create their own deepfake image or video. The immediate goal of this project is to focus on detecting all types of deepfake like replacements deepfake, retrenchment deepfake and interpersonal deepfake.

BIBLIOGRAPHY

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

[2] Yuezun Li, Ming-Ching Chang and Siwei Lyu “Exposing AI Created Fake Videos by Detecting Eye Blinking” in arxiv.

[3] Huy H. Nguyen , Junichi Yamagishi, and Isao Echizen “ Using capsule networks to detect forged images and videos ”.

[4] Daniel Mas Montserrat, Hanxiang Hao, S. K. Yarlagadda, Sriram Baireddy, Ruiting Shao Janos Horv ´ ath, Emily Bartusiak, Justin Yang, David G ´ uera, Fengqing Zhu, Edward J. Delpin “Deepfakes Detection with Automatic Face Weighting” in arXiv:2004.12027.

[5] Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales and Javier Ortega-Garcia “DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection” in arXiv:2001.00179v3.

[6] Guo, Y., Jiao, L., Wang, S., Wang, S., and Liu, F. (2017). Fuzzy sparse autoencoder framework for single image per person face recognition. IEEE Transactions on Cybernetics, 48(8), 2402-2415.

[7] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). SegNet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12), 2481-2495.

[8] Liu, F., Jiao, L., and Tang, X. (2019). Task-oriented GAN for PolSAR image classification and clustering. IEEE Transactions on Neural Networks and Learning Systems, 30(9), 2707-2719.

[9] Kaliyar, R. K., Goswami, A., and Narang, P. (2020). Deepfake: improving fake news detection using tensor decomposition based deep neural network. Journal of Supercomputing, doi: https://doi.org/10.1007/s11227-020-03294-y.

[10]] Lyu, S. (2020, July). Deepfake detection: current challenges and next steps. In IEEE International Conference on Multimedia and Expo Workshops (ICMEW) (pp. 1-6). IEEE.