



**Faculty of Computers
& Artificial Intelligence**



Benha University

Deepfake Detection

كشف التزييف العميق

A senior project submitted in partial fulfillment of the requirements for the degree of Bachelor of Computers and Artificial Intelligence.

“Information Security and Digital Forensics” Program,

Project Team

- 1- Ahmed Hesham
- 2- Gehad Mohammed
- 3- Mostafa Ebrahim
- 4- Mostafa Mohammed Abdelaziz
- 5- Robert Ayman
- 6- Samaa Hamdan

Under Supervision of

Dr. Ezz Eldin Badawy

Benha, **July 2022**

DECLARATION

We hereby certify that this material, which we now submit for assessment on the program of study leading to the award of Bachelor of Computers and Artificial Intelligence in *(Deepfake Detection)* is entirely our work, that we have exercised reasonable care to ensure that the work is original, and does not to the best of our knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of our work.

Signed: _____

Date: Monday, 18 Jul 2022.

ABSTRACT

In recent months, free deep learning-based software tools have facilitated the creation of credible face exchanges in videos that leave few traces of manipulation, in what they are known as "Deepfake"(DF) videos.

Manipulations of digital videos have been demonstrated for several decades through the good use of visual effects, recent advances in deep learning have led to a drastic increase in the realism of fake content and the accessibility in which it can be created.

Creating the DF using artificially intelligent tools is a simple task. But, when it comes to detection of these DF, it is a major challenge, because training the algorithm to spot the DF is not simple.

We have taken a step forward in detecting the DF using EfficientNetB0. The system uses EfficientNetB0 to extract features at the frame level. These features are used to classify if a video has been subject to manipulation or not and can detect the visual artifacts within frames introduced by the DF creation tools.

TABLE OF CONTENTS

LIST OF FIGURES.....	III
LIST OF ACRONYMS/ABBREVIATIONS.....	5
1. INTRODUCTION.....	6
1.1 OVERVIEW	6
1.2 History of deepfake	6
1.3 PROBLEM STATEMENT	8
1.4 GROWTH OF DEEPFAKE VIDEOS.....	8
1.5 SOLUTION.....	10
1.6 MOTIVATION	10
1.7 OBJECTIVE.....	11
1.8 IMPACT OF THE PROJECT	11
1.9 PROJECT BOOK OUTLINE.....	11
2. THEORETICAL BACKGROUND AND LITERATURE REVIEW	12
2.1 THEORETICAL BACKGROUND	12
2.1.1 Types of deepfakes	12
2.1.2 How deepfakes are made?	12
2.1.3 How many pictures do you need to create a deepfake?	15
2.2 LITERATURE REVIEW	15
2.2.1 General-network-based methods.....	16
2.2.2 Temporal-consistency-based methods	16
2.2.3 Visualartefacts-based methods.....	17
2.2.4 Camera-fingerprints-based methods	17
2.2.5 Biological-signals-based methods.....	18
2.3 CHAPTER SUMMARY.....	19
3. PROPOSED SYSTEM	21
3.1 OVERVIEW	21
3.2 SYSTEM DESIGN	21
3.3 EXPERIMENT STUDY	22
3.3.1 Dataset:	22
3.3.2 Preprocessing:.....	23
3.3.3 Model:.....	25
3.3.4 Predict	29
3.4 IMPLEMENTATION OF MODEL.....	30
3.5 RESULTS	32
3.6 CHAPTER SUMMARY	34
4. SYSTEM GRAPHICAL USER INTERFACE.....	35
4.1 WEBSITE	35
4.1.1 Flask Framework	35
4.1.2 The Flask story	36

4.1.3	Advantages of the Flask framework	36
4.1.4	Flask API	37
4.1.5	Implementation of API	38
4.1.6	Web Graphical User Interface	41
4.2	Android Application.....	45
4.2.1	Application Graphical User Interface	45
5.	REFERENCES.....	50

LIST OF FIGURES

Figure 1-1 Growth of deepfake videos since December 2018 ^[2]	9
Figure 1-2 the growth of deepfake papers ^[2]	9
Figure 2-1 Training phase ^[3]	13
Figure 2-2 Generating deepfakes phase ^[3]	14
Figure 2-3 GAN ^[4]	15
Figure 3-1 System Architecture.....	22
Figure 3-2 Dataset	23
Figure 3-3 Preprocessing flow on video.....	24
Figure 3-4 Preprocessing flow on images	25
Figure 3-5 (a) ^[5]	26
Figure 3-5 (b) ^[5]	26
Figure 3-5 (c) ^[5]	27
Figure 3-5 (d) ^[5]	27
Figure 3-5 (e) ^[5]	27
Table 3-1 EfficientNet-B0 framework	28
Figure 3-6 Prediction flow.....	29
Figure 3-7 Loading Data	30
Figure 3-8 Preprocessing	30
Figure 3-9 Model Creation	31
Figure 3-10 Model Training	31
Figure 3-11 Prediction.....	31
Table 3-2 Model accuracy	32
Figure 3-12 Efficient net with FF++	32
Figure 3-13 Efficient net with FF++ and 140k images	33
Figure 3-14 Efficient net with 140k images	33
Figure 4-1 Website context diagram	37
Figure 4-2 Website activity diagram	38
Figure 4-3 Import some libraries that use in API	38
Figure 4-4 Display Home page and redirect to display upload image, video and about us pages	39
Figure 4-5 Prediction Function.....	39
Figure 4-6 Video prediction and Display Result	40
Figure 4-7 Image prediction and display result	40
Figure 4-8 Loading Trained Model and Run API	40
Figure 4-9 Home page	41
Figure 4-10 Upload image page	42

Figure 4-11 Image Prediction Page	42
Figure 4-12 Upload Video Page	43
Figure 4-13 Video Prediction Page	43
Figure 4-14 About us page	44
Figure 4-15 GitHub page.....	44
Figure 4-16 Application icon.....	45
Figure 4-17 Start screen.....	46
Figure 4-18 Home page	46
Figure 4-19 Upload image or video.....	47
Figure 4-20 Browse	47
Figure 4-21 Prediction result	48
Figure 4-22 Menu navigation	48
Figure 4-23 About us page	49
Figure 4-24 GitHub page.....	49

LIST OF ACRONYMS/ABBREVIATIONS

DF	Deepfake
GAN	Generative Adversarial Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
SVM	Support-vector machine
AI	Artificial Intelligence
PRNU	Photo Response NonUniformity
LTRC	Long-Term Recurrent CNN
rPPG	remote PhotoPlethysmoGraphy
RGB	Red, Green and Blue
FF++	Face Forensics ++
2D	2 Dimension
BN	Batch Normalization
BSD	Berkeley Source Distribution
IRC	Internet Relay Chat
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface

Chapter One

1. INTRODUCTION

1.1 OVERVIEW

Deepfake is a term for videos and presentations enhanced by artificial intelligence and other modern technology to present falsified results. One of the best examples of deepfakes involves the use of image processing to produce video of celebrities, politicians or others saying or doing things that they never actually said or did

Anyone with basic computing skills and a home computer can create a deepfake. Some methods to detect deepfakes use AI to analyze videos for details that deepfakes fail to imitate realistically such as blinking or facial tics.

However, detection is challenging because deepfake technologies evolve rapidly.

deepfakes are not real; they're fake. The deep in deepfake is a bit murkier. Deep has long-established use to describe what is difficult, complicated, intense, etc., and that use applies here, but deep in deepfake is also likely related to its use in deep learning, another term that has yet to meet our criteria for entry. The meaning of deep learning is still settling, but most often it refers to a kind of machine learning (that is, a complex process by which a computer is able to improve its own performance) that uses layered neural networks (that is, computer architecture in which processors are interconnected in a way that suggests neural connections in the human brain) to enhance the accuracy of the machine learning algorithms.

1.2 HISTORY OF DEEPPFAKE

The origin story of deepfakes goes all the way back to the year 1997. It was the Video Rewrite program by Bregler et al. that first published a study about it as a result of a conference on computer graphics and interactive techniques. The experts explained how to modify existing video footage of a person speaking with a different audio track.

It wasn't a new concept (photo manipulation already happened back in the 19th century), but the seemingly insignificant study on altering video content was the first of its kind that fully automated the process of facial re-animation. The help of machine learning algorithms was used to achieve this. It was a huge milestone and quite possibly the real starting point of deepfake video development around the globe.

However, it wasn't until years later that the concept of deepfakes really started to catch on. Just like many new technologies, the mass-adoption comes quite a bit of time later after the invention. First came the Face2Face program in 2016, in which Thies et al. showcased how real-time face capture technology could be used to re-enact videos in a realistic way.

It wasn't until July 2017 that more people started to be interested in the applications of deep learning and media. We were introduced to a highly realistic deepfake video featuring former United States president Barack Obama. The study by Suwajanahorn et al. showed for the first time how audio could be lip synced in a frighteningly realistic way on politicians. A dangerous precedent that could potentially transform the course of the political future.

After that, things went very quickly. The adoption and introduction of the technology by the masses quite literally exploded, as more and more deepfake videos and similar applications of the machine learning technique started to be implemented around the globe. The Obama deepfake was a popular one, and a year later it was re-iterated in the now infamous BuzzFeed YouTube video titled "You Won't Believe What Obama Says In This Video!".

This video, among several similar ones that came out around that time (the spring of 2018), were among the first viral deepfake videos that introduced the wider public to the idea that videos could be fakes in such a realistic way, that they would be indistinguishable from the real thing. Soon after, you had the Snapchat face swap filters, AI voice changer apps, open-source deepfake video creation software, and many more initiatives.

The 'deepfake virus' has now become uncontrollable and is most likely here to stay. For some, that's a good thing. But we shouldn't underestimate the enormous adverse impacts that deepfakes can have on our global society as a whole. We are now slowly entering a new era, in which AI deepfakes will slowly start to be a factor of relevance in many types

of industries. From the technology sector to political election processes, and from revenge pornography to voice phishing scams. The threat of deepfakes is more apparent than ever.

1.3 PROBLEM STATEMENT

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 social media platforms has become very common leading to spamming and peddling wrong information over the platform. These types of deepfake will be terrible and lead to threatening, misleading of common people.

Deepfakes pose a significant problem for public knowledge. Their development is not a watershed moment altered images, audio, and video have pervaded the internet for a long time, but they will significantly contribute to the continued erosion of faith in digital content. The best artificial intelligence tools are open for anyone to use, and many deepfake-specific technologies are freely available. This means that creating convincing fake content is easier than ever before and will become more accessible for the foreseeable future.

It is certainly possible that a single convincing, spectacular, and effectively timed deepfake video could temporarily crash the stock market, cause a riot, or throw an election. However, these are not the most likely scenarios. More likely, a large number of deepfakes uploaded by amateurs (either with satirical or political aims) and influence campaigns (of foreign origin or those driven by advertising monetization) will slowly disperse through the internet, further clouding the authenticity of the digital world.

1.4 GROWTH OF DEEFAKE VIDEOS

In July 2019 it was found over 14,678 videos online – that's an increase of 100 percent over 7,964 videos found in 2018. It also notes that 96 percent of these videos are pornography.

In late 2019, there were a huge increase in less than a year! This trend is still going strong as deepfake software hit mainstream.

The growth of deepfake videos is increased Remarkably, As of June 2020, deepfake videos identified online doubled to 49,081 in just six months since January, according to Deep Trace Lab, a deepfake detection technology firm.

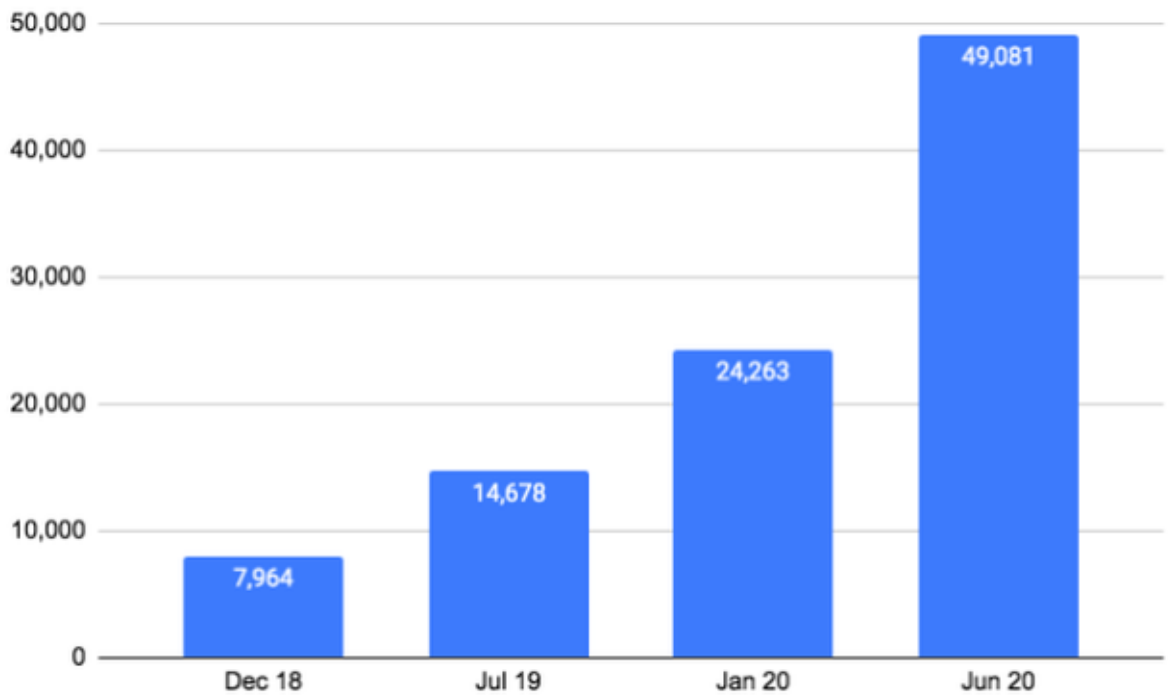


Figure 1-1 Growth of deepfake videos since December 2018 ^[2]

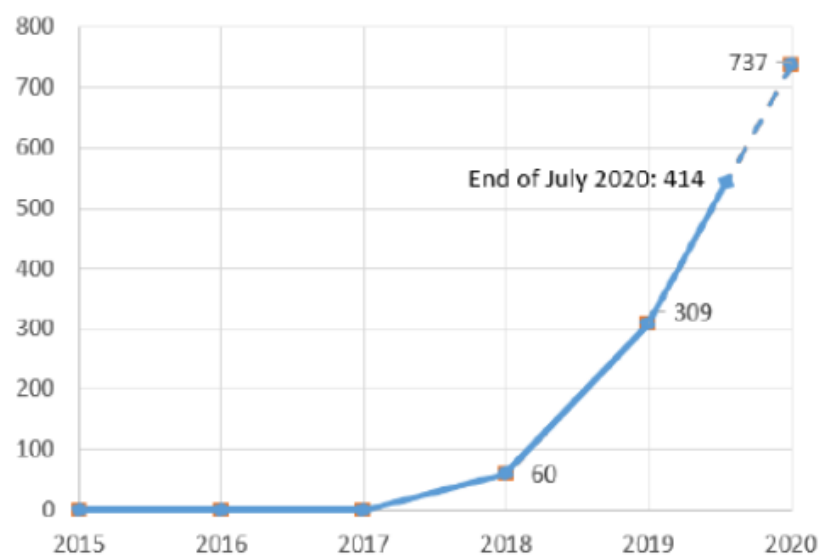


Figure 1-2 the growth of deepfake papers^[2]

1.5 SOLUTION

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

1.6 MOTIVATION

The impetus behind this project is to recognise these distorted media, which is technically demanding, and which is rapidly evolving. Many engineering companies have come together to unite a great deal of dataset. Competitions and actively include data sets to counter deepfakes. Deepfake videos are now so popular that multiple political parties utilise this tool to produce faked images of the leader of their opposing party to propagate hate against them. Fake political videos telling or doing things that have never happened is a threat to election campaigns. These images are the primary source of false media controversies and propagate misleading news.

Also more technological devices are using facial recognition or voice recognition than ever before. While it seems like one of the most secure ways to protect a device, the creation of deepfakes calls that into question.

For example, perhaps your Apple iPhone, Google Pixel phone, or Microsoft Windows phone uses one of these features to unlock the device. Maybe your Amazon Alexa, Google Nest, or other smart home device recognizes your voice when acting as an assistant. But what if someone could recreate your likeness or voice? There are increasing concerns that this new technology puts device security at risk.

In order to expose the forgery in extremely detailed facial expression, such details to be investigated frame by frame in the production of a deepfake picture.

The goal of this research is to create a deep learning model that is capable of recognising deepfake images. A thorough analysis of deepfake video frames to identify slight imperfections in face head and the model will learn what features differentiate a real image from a deepfake.

1.7 OBJECTIVE

The Mechanism is the way by which minor imperfections are found in doctored videos and exposes. The creation of a doctored video needs information to be examined in order to reveal the dishonest, particularly facial expression variables. It is defined as an in-depth video study to find minor imperfections such as boundary points, background incoherence, double eyebrows or irregular twitch of the eye.

1.8 IMPACT OF THE PROJECT

we expect that deceive and blackmail people is reduced using deepfake because anyone can use our project to detect if video or image is real or fake

1.9 PROJECT BOOK OUTLINE

This documentation will discuss all aspects of our deepfake detection system in each chapter, we cover both the benefits and challenges and discover that our system is the generally appropriate solution for dealing with many problems caused by deepfake.

Chapter 1: provides a brief introduction to the project, problem statement, solution of problem ,motivation ,objective , and impact of the project.

Chapter 2: presented the literature and theoretical background.

Chapter 3: provided overview about project, system design and experiment study that include (dataset ,pre-processing model ,training model and prediction model).

Chapter 4: introduced the system GUI including the website and Android design.

2. THEORETICAL BACKGROUND AND LITERATURE REVIEW

2.1 THEORETICAL BACKGROUND

Deepfakes are false media content created by manipulating a person in an existing image or video using powerful machine learning methods.

For example, the fake video that showed Barack Obama scolding Donald Trump, which became viral. The video's main purpose was to show the consequences of Deepfakes and how powerful they are. In 2019, Mark Zuckerberg appeared in a fake video about how Facebook controls billions of user data.

They are created by training using autoencoders or GANs to create highly deceiving media. Deepfakes (from “deep learning” and “fake”) are created by techniques that can set face images of a target person onto a video of a source person to make a video of the target person doing or saying things the source person does.

Nowadays deepfake videos are so easy to create that anyone can create one. There are several deepfake creation software like FakeApp, DeepFaceLab, FaceSwap, and you may find a plethora of tutorials available for creating them with few easy steps.

2.1.1 Types of deepfakes

Deepfakes mainly fall into different categories like:

- Face-swapping: where two images swap together to a fake image
- Face reenactment: changing the facial features of a person
- Audio Deepfakes: which are fake audio of a particular person
- Lip-syncing Deepfakes: videos with consistent mouth movements with an audio.
- Puppet-master: videos of a particular person (puppet) are animated, always using movements of another person (master) sitting in front of a camera.

2.1.2 How deepfakes are made?

Autoencoders and GANs are the two deep learning technologies behind the deepfake applications that have developed so far.

Autoencoders

Autoencoders mainly use face-swapping Deepfakes. To make a Deepfake video of someone, first, you need to train an autoencoder with two parts: an encoder and a decoder. This technique usually uses two encoder and decoder pairs. Using the encoder, you need to run many images of the two people you want to swap. To make these images more realistic, images need to consist of face shots from different angles and lighting.

During this training, the encoder extracts the latent features of the images or reduces them to a latent representation compressing the images. Then the decoder will reconstruct and recover the images from this latent image representation.

For example, suppose you trained the image of a person number 1 using decoder A. Then you can use decoder B to recover that image. Decoder A then reconstructs the image of person number two using the features of person number 1.

When you complete the training, swap the two decoders to recover two different images, ultimately swapping the images as shown on Figure 2.1 and Figure 2.2. The FakeApp software is popular swap-based which is very effective in the realistic generation of images.

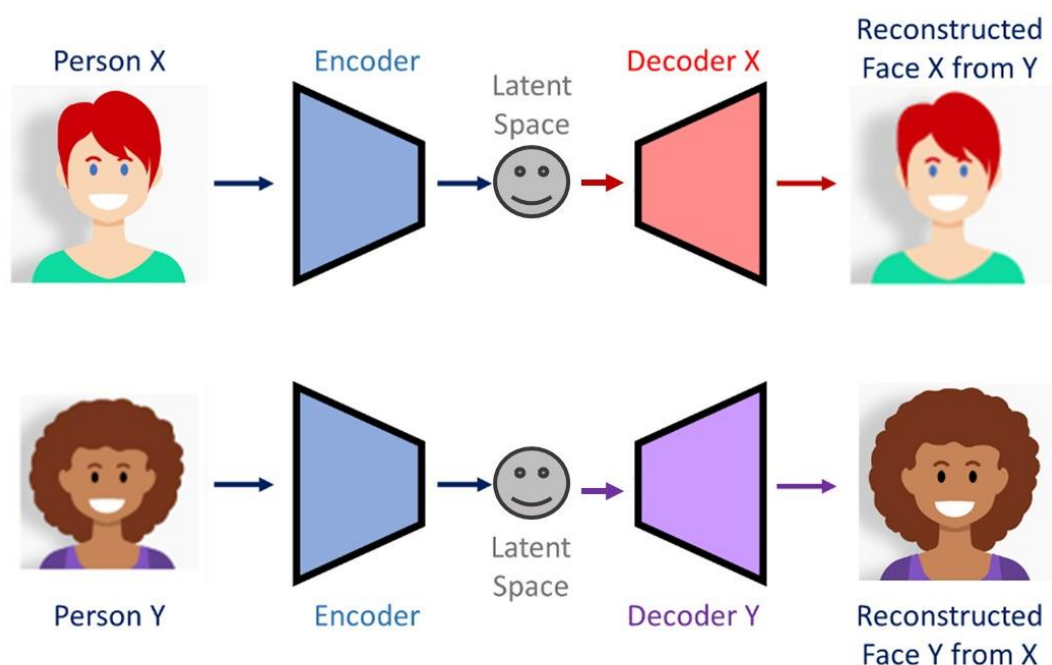


Figure 2-1 Training phase^[3]

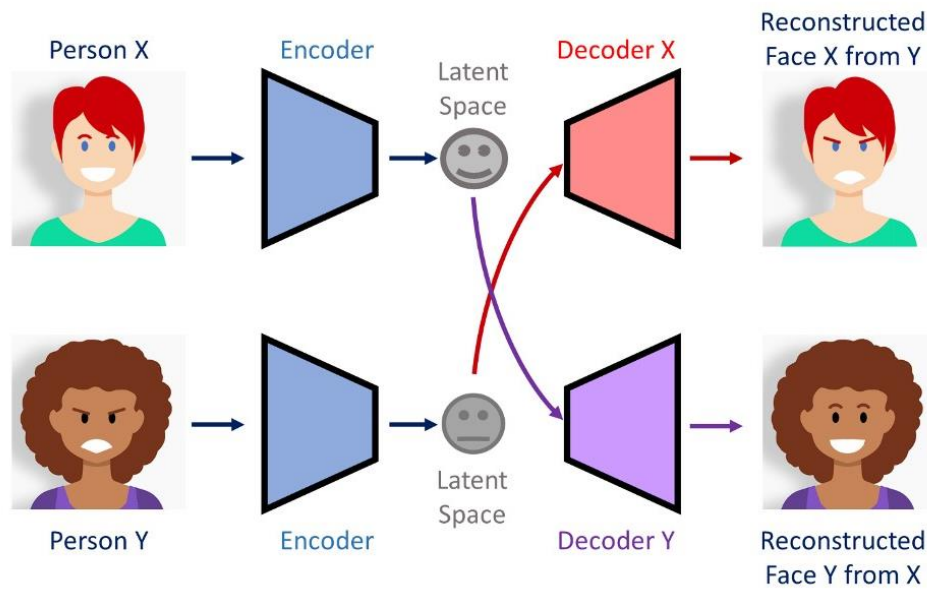


Figure 2-2 Generating deepfakes phase^[3]

Generative Adversarial Networks (GANs)

The objective of the generative adversarial network is to create something new based on previous data. For example, it can come up with a human face after studying hundreds of pictures. Or it can generate a painting resembling a particular artist's style using their work as reference material.

GANs set two neural networks in direct competition with one another – generator and discriminator. Generator produces a new image as an output based on the knowledge that the neural network has been taught. Discriminator determines whether the image is real or fake.

Both components stay in constant interaction. Generator learns how to create images that will deceive the discriminator and make it classify a produced image as a real one. Discriminator, on the other hand, learns how not to be deceived. The better the discriminator is, the harder it will be for the generator to make realistic images and, ultimately, the better job it will accomplish. This method is shown in Figure 2.3.

To illustrate it in simple terms, let's use the following analogy. The better the teacher (discriminator) is at teaching, the more it will help the student (generator) learn and improve

their academic performance. The student submits their work, and the teacher marks their mistakes. Only then can the student realize what they have done wrong and correct their errors. Because of this reason, the generator will create images as real as possible.

Training Discriminator

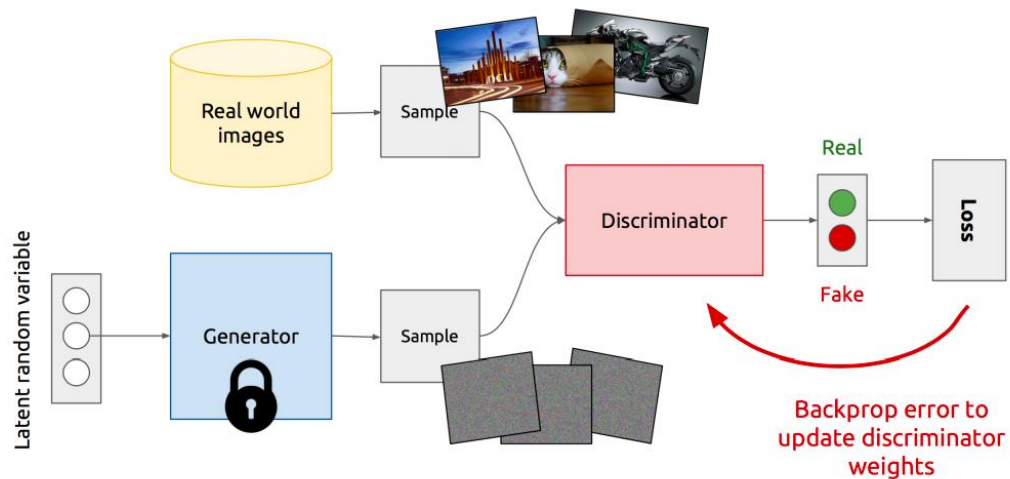


Figure 2-3 GAN^[4]

2.1.3 How many pictures do you need to create a deepfake?

The accuracy and quality of a Deepfake image heavily depend on the number of target images used to train the deep learning model. It is better to use 300–2000 images of their face to recreate the images properly. The images also need to have a wide range of facial features.

2.2 LITERATURE REVIEW

Deepfake videos are increasingly harmful to personal privacy and social security. Various methods have been proposed to detect manipulated videos. Early attempts mainly focused on inconsistent features caused by the face synthesis process while current detection methods mostly target at fundamental features. These methods fall into five categories based on the features they use.

To begin with, detection based on general neural networks is commonly used in literature, where deepfake detection task is considered as regular classification tasks. Temporal

consistency features are also exploited to detect discontinuities between adjacent frames of fake video. To find distinguishable features, visual artefacts generated in blending process are exploited in detection tasks.

Recently proposed approaches focus on more fundamental features, where camera fingerprint and biological signal-based schemes show great potential in detection tasks. In the following sections, we will review detection methods mentioned above.

2.2.1 General-network-based methods

Recent advances in image classification have been applied to improve the detection of deepfake videos. In this method, face images extracted from the detected video are used to train the detection network. Then, the trained network is applied to make predictions for all frames of this video. The predictions are finally calculated by averaging or voting strategy. Consequently, the detection accuracy is highly dependent on the neural networks, without the need to exploit specific distinguishable features.

Existing network-based methods are divided into two types:

- Transfer learning-based methods
- Detection approaches based on specially designed networks.

2.2.2 Temporal-consistency-based methods

Time continuity is a unique feature for videos. Unlike images, video is a sequence composed of multiple frames, where adjacent frames have a strong correlation and continuity. When video frames are manipulated, the correlation between adjacent frames will be destroyed due to defects of deepfake algorithms, specifically expressed in the shift of face position and video flickering. According to this phenomenon, researchers have proposed CNN-RNN architecture.

CNN-RNN

Considering the time continuity in videos, it was first proposed to use RNN to detect deepfake videos. In that work, autoencoder was found to be completely unaware of previously generated faces because faces were generated frame-by-frame. This lack of

temporal awareness results in multiple anomalies, which are crucial evidence for deepfake detection.

To check the continuity between adjacent frames, an end-to-end trainable recurrent deepfake video detection system was proposed. The proposed system is mainly composed of a convolutional long short-term memory (LSTM) structure for processing frame sequences.

Two essential components are used in a convolutional LSTM structure, where CNN is used for frame feature extraction and LSTM is used for temporal sequence analysis. Specifically, a pretrained inceptionV3 is adapted to output a deep representation for each frame. The 2048-dimensional feature vectors extracted by the last pooling layers are applied as the sequential LSTM input, characterizing the continuity between image sequences. Finally, a fully connected layer and a softmax layer are added to compute forgery probabilities of the frame sequence tested.

2.2.3 Visual artefacts-based methods

In most existing deepfake methods, the generated face has to be blended into an existing background image, causing existing intrinsic image discrepancies on the blending boundaries. Faces and background images come from different source images, giving rise to the abnormal behaviour of the synthetic image, such as boundary anomaly and inconsistent brightness. These visual artefacts make deepfake videos fundamentally detectable.

There are three main visual artefacts:

- Face warping artefacts
- Blending boundary
- Head pose inconsistency

2.2.4 Camera-fingerprints-based methods

Camera fingerprints are a kind of noise with very weak energy, which plays an important role in forensic fields, especially source identification tasks. In general, camera fingerprint based approaches have gone through three processes: the photo response nonuniformity (PRNU) patterns, noiseprint and recent video noise pattern.

2.2.5 Biological-signals-based methods

Detection based on biological signals is an interesting scheme that emerged in recent years. The core observation is that even though GAN is able to generate faces with high realism, the naturally hidden biological signals are still not easily replicate, making it difficult to synthesize human faces with reasonable behaviour. Taking advantage of this abnormal behaviour, several studies have been proposed.

There are two approaches based on biological signals:

- Blinking frequency-based
- Heart rate-based detection approaches

Eye blinking

Abnormalities with blink frequency were earlier identified as discriminable features in deepfake detection tasks. This could be attributed to the fact that deepfake algorithms train models using a large number of face images obtained online.

Most of the images show people with their eyes open, causing that a closed-eye view is difficult to generate in a manipulated video. Based on this finding, a deep neural network model, known as long-term recurrent CNN (LRCN), was introduced to distinguish open and close eye states.

Heart rate

Except for blink frequency, heart rate was also found the difference between real and manipulated videos. It has been proved that colour changes of skin in the video could be applied to infer heart rate. Based on these findings, a detector based on biological signals named FakeCatcher was designed to detect deepfake videos. Specifically, remote photoplethysmography (rPPG) was used to extract heart rate signals according to subtle changes of colour and motion in RGB videos.

2.3 CHAPTER SUMMARY

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. All of that directed us towards Deepfake detection.

More and more network- based methods have begun to introduce multitask learning, that is, not only to classify real and fake faces, but also to generate pixel-level tampering masks.

Compared with general-network-based approaches, temporal consistency-based detection methods consider the continuity between adjacent frames, thereby improving the detection performance. However, many models tend to destroy the spatial structure of original frames when extracting temporal features while the motivation for designing such methods is precisely to extract the inconsistency of spatial features in the temporal domain. CNN-RNN architectures pool the intraframe features into vectors, thus cannot capture spatial features while detecting temporal consistency.

Visual-artefacts-based methods often obtain better generalization performance because they target more general artefacts existing in most deepfake contents. However, these algorithms can only detect specific forgery traces due to paying more attention to specific artefacts. With the progress of deepfake algorithms, these artefacts are gradually disappearing. Nevertheless, visual artefacts-based approaches obtain better performance in the latest version of deepfake video datasets.

Camera fingerprints have been proved to be effective in deepfake detection tasks. However, accurate estimation of camera fingerprints requires a large number of images captured by different types of cameras. Thus, there would be a decrease in accuracy when detecting images captured by unknown cameras.

On the other hand, camerafingerprint-based methods are not robust to simple image postprocessing such as compression, noise and blur. Since GAN images are generated without any image capture process, there is no camera fingerprint in the output image, so that the camerafingerprintbased methods are very suitable for detecting images generated by GANs.

Although biologicalsignals-based detection approaches have shown good performance on various datasets, the natural flaw of this kind of method is that the detection process cannot be performed in an end-to-end way. Also, the information reflected by biological signal is seriously affected by video quality, so there are natural flaws and limited application range for biologicalsignals-based approaches.

3. PROPOSED SYSTEM

3.1 OVERVIEW

We have developed one pipeline of efficient detection of forged video and image footage using deepfake detection technology for face identify swap.

After our preprocessing, faces are cropped from the original image-level dataset, while the original video-level dataset is first simplified to image-level then faces are cropped, and the real face is marked as label 0 and the fake face is marked as label 1, so that the detection results of the image to be detected are displayed with a score of 0 to 1 (the closer the score is to 0, the closer the image is to the real; the closer the score is to 1, the closer the image is to fake), which greatly reduces the workload of the detection process and improves detection efficiency.

The video-level dataset used is the face forensics datasets (FF++), which are widely used in the detection of various fake videos, while 140k Real and Fake Faces is the image-level dataset used. The training and evaluation of detection models are performed on these two large-scale fake face image/video dataset.

3.2 SYSTEM DESIGN

Our method is focusing on detecting all types of deepfakes like replacement deepfakes, retrenchment deepfakes, and interpersonal deepfakes. Figure 3-1 represents the simple system architecture of the proposed system:

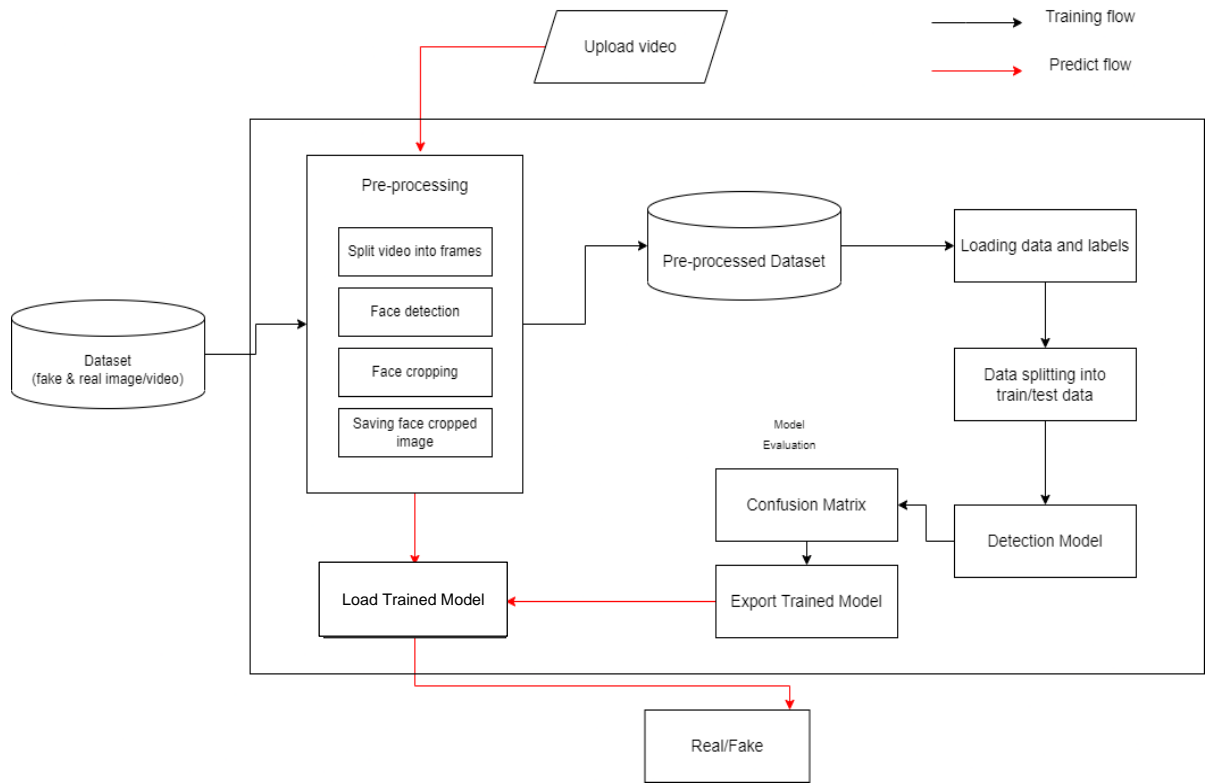


Figure 3-1 System Architecture

3.3 EXPERIMENT STUDY

3.3.1 Dataset:

Video dataset: we are using a mixed dataset which consists of an equal amount of videos from different dataset sources like YouTube, FaceForensics++. Videos are 10 seconds long on average with 30 fpm frame rate.

The average frame dimension is 720*1280. Our newly prepared dataset contains 50% of the original video and 50% of the manipulated deepfake videos. The dataset is split into 80% train and 20% test set.

Image-level dataset: 140k Real and Fake Faces dataset consists of all 70k REAL faces from the Flickr dataset collected by Nvidia, as well as 70k fake faces sampled from the 1 Million

FAKE faces (generated by Style GAN). In this dataset, all the images are resized into 256px. The dataset is split into 80% train and 20% test set.

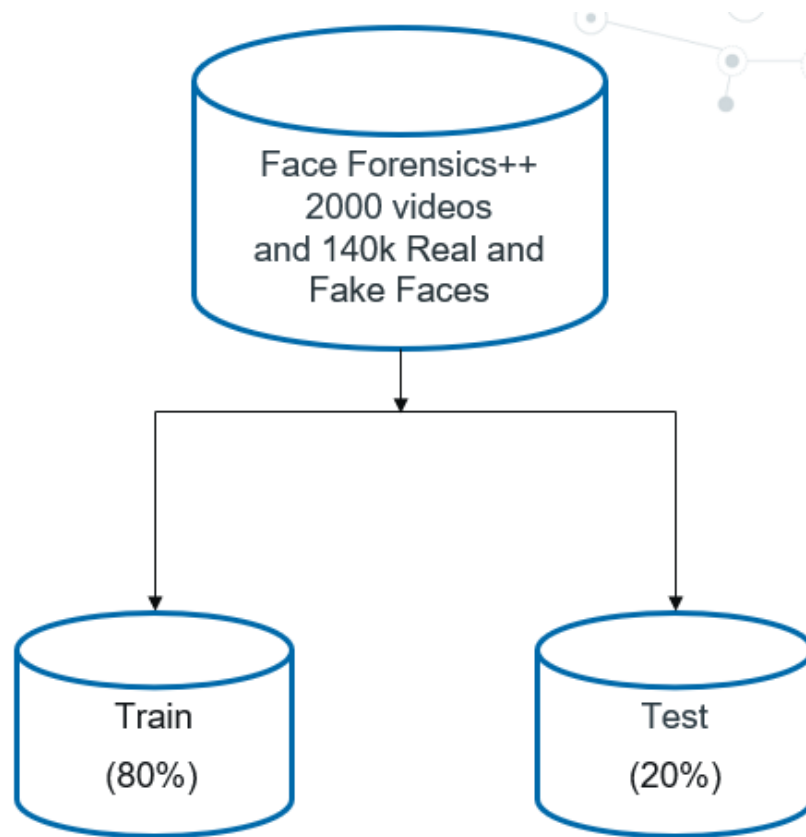


Figure 3-2 Dataset

3.3.2 Preprocessing:

After selecting the datasets, which is convenient for the training and verification of the network, we prepare the datasets for model training by pre-processing them. First, we upload the label file. Then the video-level dataset is converted into a picture-level, using “cv2” to split videos in the dataset, respectively into a group of frames.

In order to improve the diversity and richness of the image dataset as well as memory usage and runtime, we choose to extract a frame image every 30 frames when the video is divided into frames.

In order to enable the network to fully learn the faces in the real and fake pictures, face recognition and cropping work are performed on the video frame images. Using the face 68 feature point detection script in Dlib, all faces in the picture can be accurately identified.

Finally, the processed real and fake face pictures are uniformly scaled ($224*224*3$) and numbered in turn, respectively, saved to the real face pictures and forged face pictures in two fold.

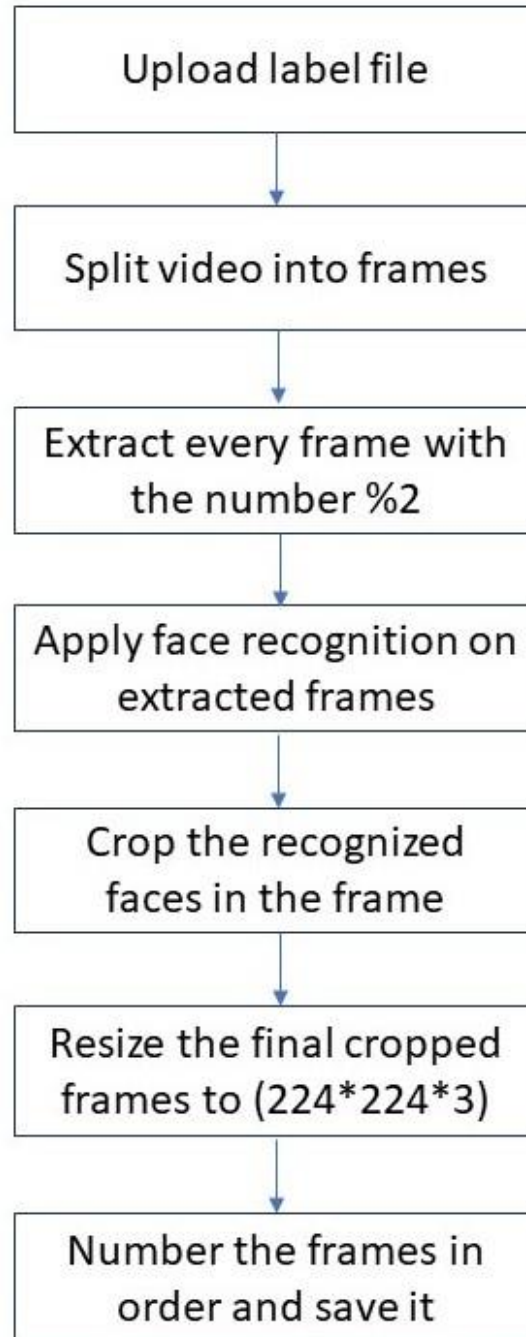


Figure 3-3 Preprocessing flow on video

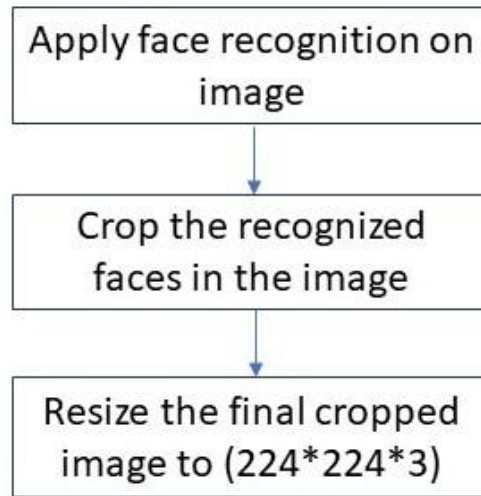


Figure 3-4 Preprocessing flow on images

3.3.3 Model:

In this section, we present our end-to-end trainable re-current deepfake video detection model. The proposed system is composed of ExceptionNetB0 network followed by 2D average pooling layer.

ExceptionNet

The multidimensional mixed model scaling method (EfficientNet series network) proposed by Google in 2019 has attracted extensive attention in the academic community. In order to explore a model scaling method that takes both speed and accuracy into account, the EfficientNet series network that simultaneously scales the three dimensions of network depth, network width, and image resolution is first proposed.

As shown in Figure 5, Figure 5(a) represents the baseline network model based on a convolutional neural network. The input is a three-channel colour image with a width of W and a height of H .

After the layer-by-layer convolution, the network could learn corresponding features in the picture; Figures 5(b)–5(d) represent three common methods of unilaterally scaling the network, respectively: Figure 5(b) is to improve the network from the perspective of the resolution of the input image and improve the network learning efficiency by proportionally

enlarging or reducing the size of the input image; Figure 5(c) is to improve the network and improve network performance by changing the number of channels in each layer of the network; Figure 5(d) is to increase or decrease the number of network layers so that the network can learn more specific feature information and improve network efficiency.

Figure 5(e) shows that, in the EfficientNet network, the composite parameters are used to simultaneously perform the scaling of the above three dimensions, thereby improving the overall performance of the network.

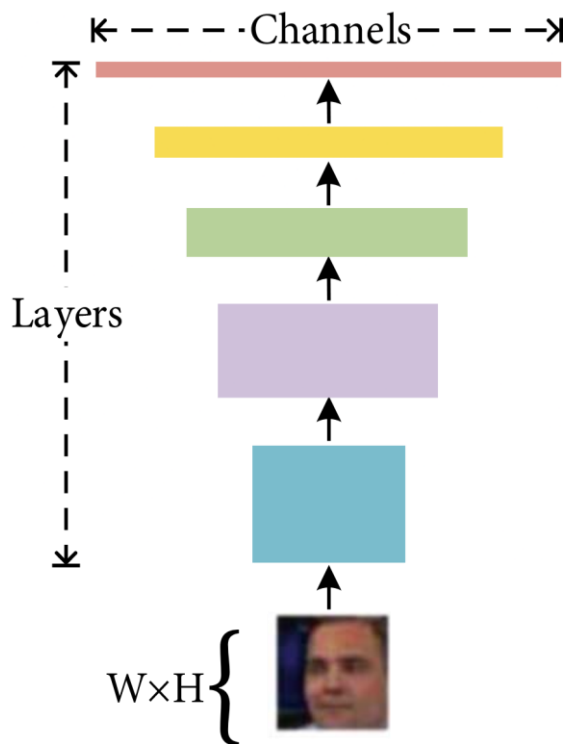


Figure 3-5 (a)^[5]

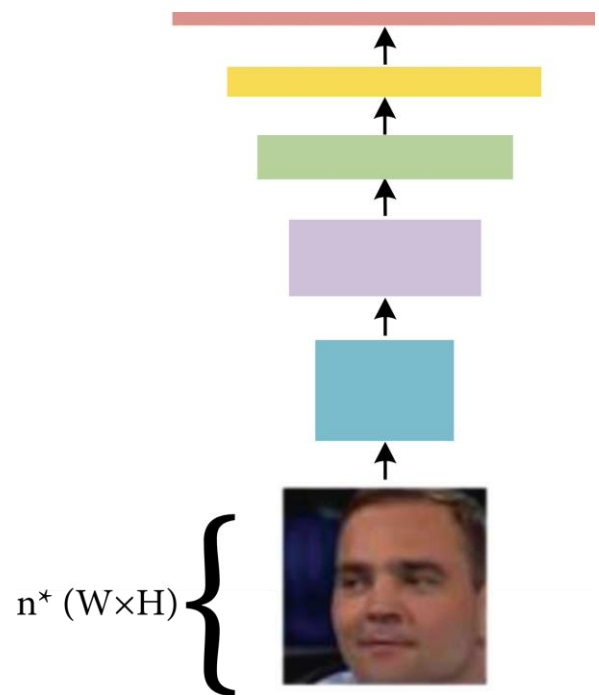


Figure 3-5 (b)^[5]

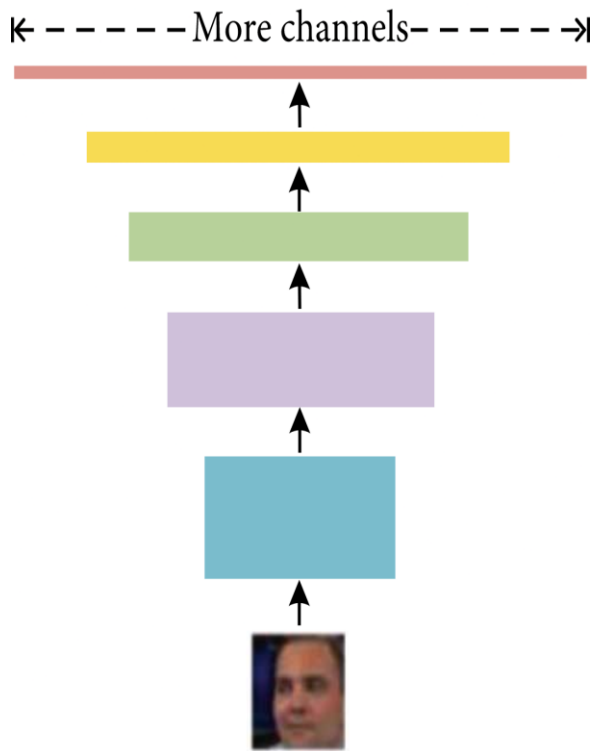


Figure 3-5 (c) ^[5]

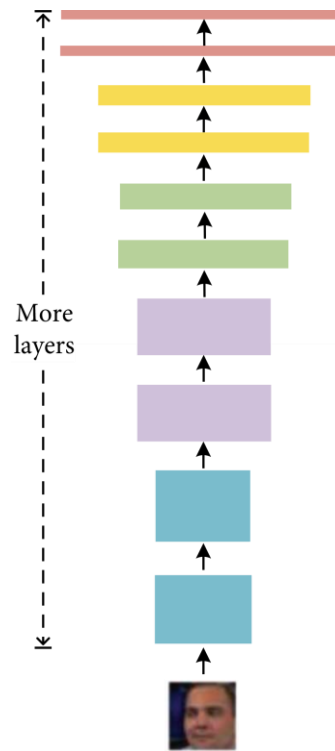


Figure 3-5 (d) ^[5]

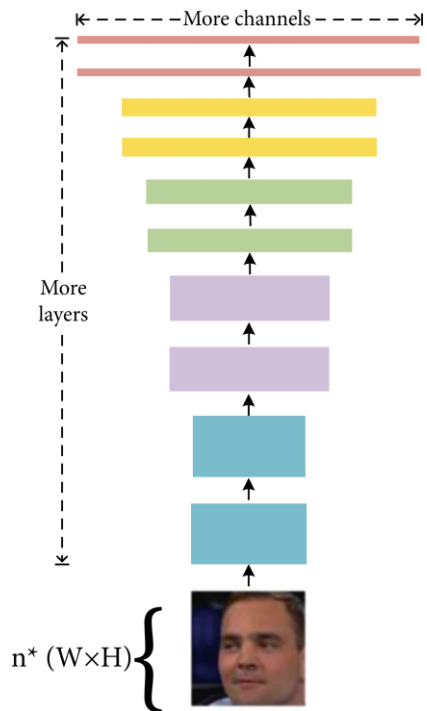


Figure 3-5 (e) ^[5]

The network is similar to the traditional convolutional neural network. Table 3-1 shows the network framework of EfficientNet-B0. It can be seen that the network is divided into 9 stages in total.

Stage1 is an ordinary convolutional layer with a convolution kernel size of 3×3 and a stride of 2, which includes BN (Batch Normalization) and the Swish activation function.

Stage 2–Stage 8 are all in repeated stacking of MBConv structures. The layers in the last column of the table indicate how many times the stage repeats the MB Conv structure, while Stage 9 consists of an ordinary 1×1 convolutional layer, an average pooling layer, and a fully connected layer, which contains BN and the Swish activation function.

In the following table, each MBConv will be followed by a number 1 or 6, where 1 or 6 is the multiplication factor n ; that is, the first 1×1 convolutional layer in MBConv will expand the channels of the input feature matrix to n times, where $k \ 3 \times 3$ or $k \ 5 \times 5$ represents the size of the convolution kernel used by Depthwise Conv in MBConv.

The channels represent the channels that output the feature matrix after passing through the stage.

Stage	Operator	Resolution	Channels	Layers
1	Conv 3×3	224×224	32	1
2	MBConv1, $k \ 3 \times 3$	112×112	16	1
3	MBConv6, $k \ 3 \times 3$	112×112	24	2
4	MBConv6, $k \ 3 \times 3$	56×56	40	2
5	MBConv6, $k \ 3 \times 3$	28×28	80	3
6	MBConv6, $k \ 5 \times 5$	14×14	112	3
7	MBConv6, $k \ 5 \times 5$	14×14	192	4
8	MBConv6, $k \ 3 \times 3$	7×7	320	1
9	Conv 1×1 &Pooling&FC	7×7	1280	1

Table 3-1 EfficientNet-B0 framework

The expression of the Swish activation function is shown in the following formula (1), where β is a constant or set as a trainable parameter. The Swish activation function is lower bound, smooth, and nonmonotonic.

$$f(x) = x \cdot \text{sigmoid}(\beta x) \quad (1)$$

3.3.4 Predict

As shown in Figure 3-6 a new video/image 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 recognition, then frame cropping and instead of storing the video into local storage, the cropped frames are directly passed to the trained model for detection. The prediction is identified by using the mode of all frames prediction. In case of image detection it is treated as a single frame video.

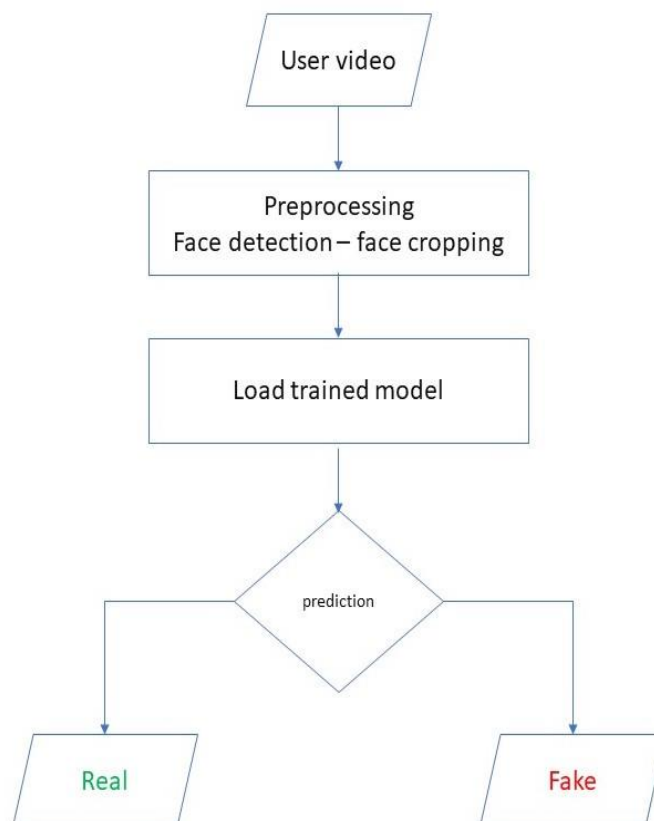
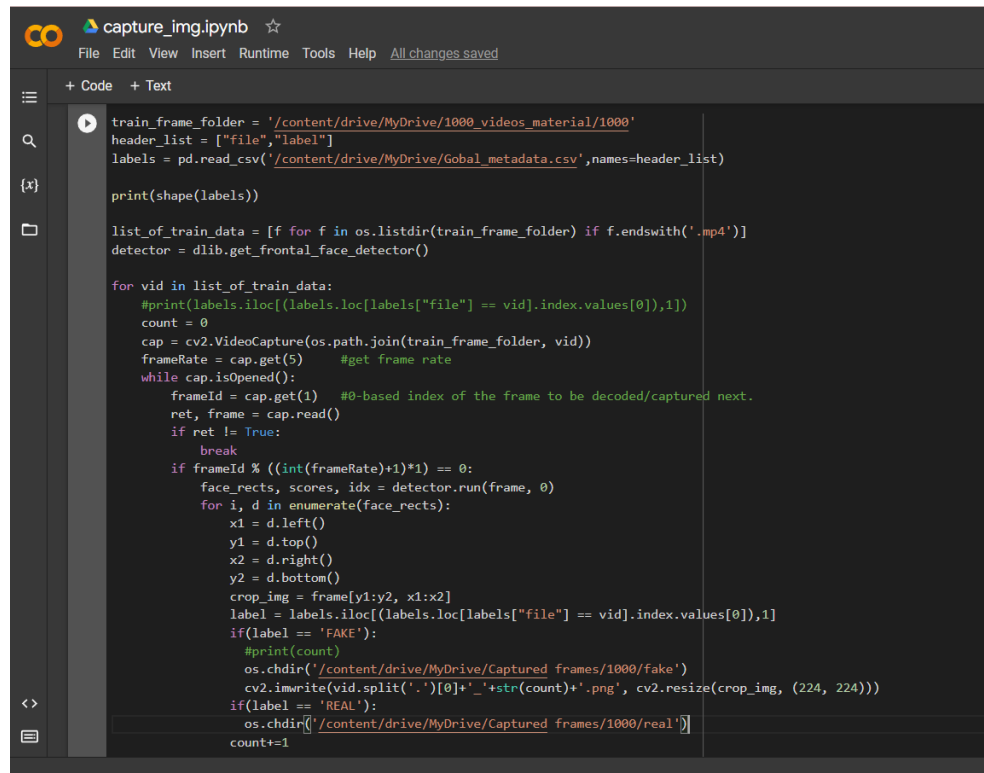


Figure 3-6 Prediction flow

3.4 IMPLEMENTATION OF MODEL



```
capture_img.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

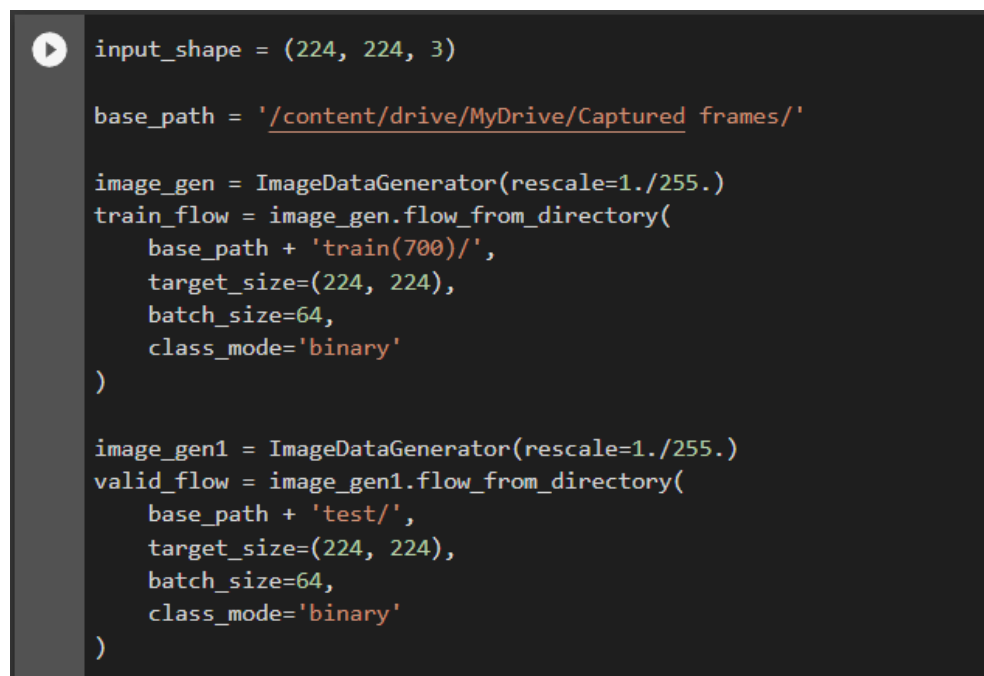
train_frame_folder = '/content/drive/MyDrive/1000_videos_material/1000'
header_list = ["file", "label"]
labels = pd.read_csv('/content/drive/MyDrive/Gobal_metadata.csv', names=header_list)

print(shape(labels))

list_of_train_data = [f for f in os.listdir(train_frame_folder) if f.endswith('.mp4')]
detector = dlib.get_frontal_face_detector()

for vid in list_of_train_data:
    #print(labels.iloc[(labels.loc[labels["file"] == vid].index.values[0]),1])
    count = 0
    cap = cv2.VideoCapture(os.path.join(train_frame_folder, vid))
    frameRate = cap.get(5) #get frame rate
    while cap.isOpened():
        frameId = cap.get(1) #0-based index of the frame to be decoded/captured next.
        ret, frame = cap.read()
        if ret != True:
            break
        if frameId % ((int(frameRate)+1)*1) == 0:
            face_rects, scores, idx = detector.run(frame, 0)
            for i, d in enumerate(face_rects):
                x1 = d.left()
                y1 = d.top()
                x2 = d.right()
                y2 = d.bottom()
                crop_img = frame[y1:y2, x1:x2]
                label = labels.iloc[(labels.loc[labels["file"] == vid].index.values[0]),1]
                if(label == 'FAKE'):
                    #print(count)
                    os.chdir('/content/drive/MyDrive/Captured frames/1000/fake')
                    cv2.imwrite(vid.split('.')[0]+'_'+str(count)+'.png', cv2.resize(crop_img, (224, 224)))
                if(label == 'REAL'):
                    os.chdir('/content/drive/MyDrive/Captured frames/1000/real')
            count+=1
```

Figure 3-7 Loading Data



```
input_shape = (224, 224, 3)

base_path = '/content/drive/MyDrive/Captured frames/'

image_gen = ImageDataGenerator(rescale=1./255.)
train_flow = image_gen.flow_from_directory(
    base_path + 'train(700)/',
    target_size=(224, 224),
    batch_size=64,
    class_mode='binary'
)

image_gen1 = ImageDataGenerator(rescale=1./255.)
valid_flow = image_gen1.flow_from_directory(
    base_path + 'test/',
    target_size=(224, 224),
    batch_size=64,
    class_mode='binary'
)
```

Figure 3-8 Preprocessing

```
[ ] googleNet_model = EfficientNetB0(include_top=False, weights=None, input_shape=input_shape)
    googleNet_model.trainable = True
    model = Sequential()

    model.add(googLeNet_model)
    model.add(GlobalAveragePooling2D())
    model.add(Dense(units = 1, activation = 'sigmoid'))
    #model.add(Flatten())

    model.compile(loss='binary_crossentropy',
                  optimizer=optimizers.Adam(lr=1e-5, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False),
                  metrics=['accuracy'])
    model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4049571
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 1)	1281

Figure 3-9 Model Creation

```
early_stopping = EarlyStopping(monitor='val_loss',
                               min_delta=0,
                               patience=2,
                               verbose=0, mode='auto')

EPOCHS = 15
BATCH_SIZE = 100
history = model.fit(train_flow, batch_size = BATCH_SIZE, epochs = EPOCHS, validation_data = valid_flow, verbose = 1)
```

Figure 3-10 Model Training

```
[ ] input_shape = (224, 224, 3)
    pr_data = []
    detector = dlib.get_frontal_face_detector()
    cap = cv2.VideoCapture('/content/drive/MyDrive/Graduation project/test(original deepfakes)/015_919.mp4')
    frameRate = cap.get(5)
    res=[]
    while cap.isOpened():
        frameId = cap.get(1)
        ret, frame = cap.read()

        if ret != True:
            break
        if frameId % ((int(frameRate)+1)*1) == 0:
            face_rects, scores, idx = detector.run(frame, 0)
            for i, d in enumerate(face_rects):
                x1 = d.left()
                y1 = d.top()
                x2 = d.right()
                y2 = d.bottom()
                crop_img = frame[y1:y2, x1:x2]
                data = img_to_array(cv2.resize(crop_img, (224, 224))).flatten() / 255.0
                data = data.reshape(-1, 224, 224, 3)
                #print(model.predict(data)[0][0])
                res.append(round(model.predict(data)[0][0]))
                #res.append((model.predict(data) > 0.5).astype("int32"))

    print(mode(res))
```

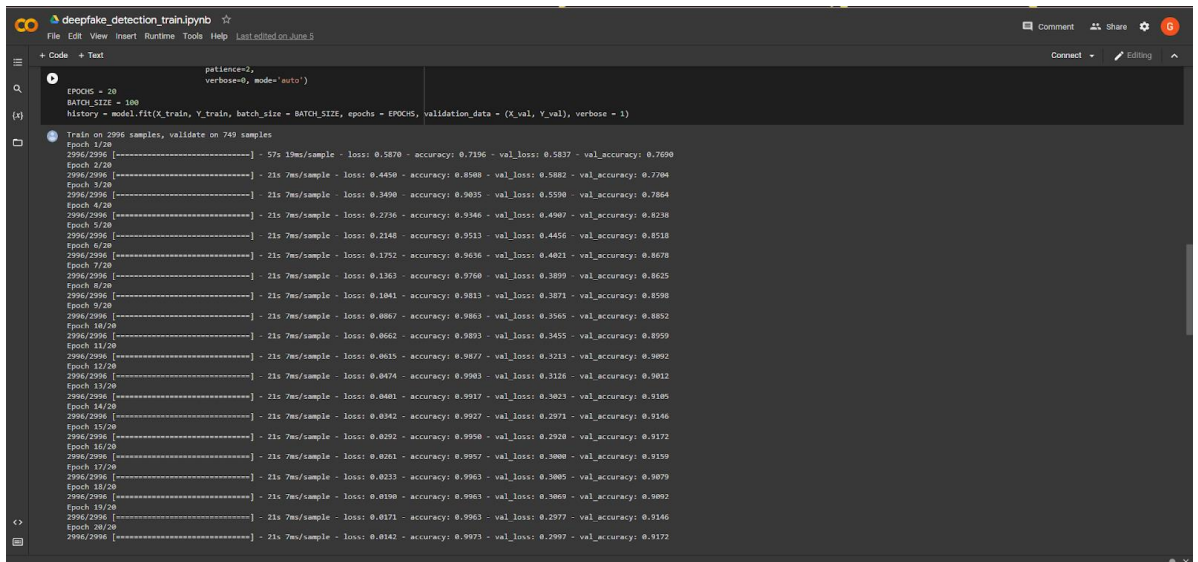
Figure 3-11 Prediction

3.5 RESULTS

At first we used the RESNext deep learning model but it has a lower accuracy and consumes a lot of resources, we then use the EfficientNet deep learning model on two datasets faceforensics++ and 140k real and fake faces. We made a combination of the two datasets and used them to train the model.

Deep learning Model	Dataset	Accuracy
RESNext	Faceforensics++(500 video)	48.0%
EfficientNet	Combination of 140k real and fake faces and Faceforensics++	96.79%
	Faceforensics++	91.72%
	140k real and fake faces	92.87%

Table 3-2 Model accuracy



```

deepfake_detection_train.ipynb
File Edit View Insert Runtime Tools Help Last edited on June 5
+ Code + Text
petience=,
verbose=0, mode='auto')
EPOCHS = 20
BATCH_SIZE = 100
History = model.fit(X_train, Y_train, batch_size = BATCH_SIZE, epochs = EPOCHS, validation_data = (X_val, Y_val), verbose = 1)

Train on 2096 samples, validate on 749 samples
Epoch 1/20
2096/2096 [-----] - 57s 10ms/sample - loss: 0.5870 - accuracy: 0.7196 - val_loss: 0.5837 - val_accuracy: 0.7690
Epoch 2/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.4450 - accuracy: 0.8508 - val_loss: 0.5882 - val_accuracy: 0.7704
Epoch 3/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.3490 - accuracy: 0.9035 - val_loss: 0.5590 - val_accuracy: 0.7864
Epoch 4/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.2736 - accuracy: 0.9346 - val_loss: 0.4907 - val_accuracy: 0.8238
Epoch 5/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.2148 - accuracy: 0.9513 - val_loss: 0.4456 - val_accuracy: 0.8518
Epoch 6/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.1792 - accuracy: 0.9636 - val_loss: 0.4021 - val_accuracy: 0.8678
Epoch 7/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.1363 - accuracy: 0.9760 - val_loss: 0.3899 - val_accuracy: 0.8625
Epoch 8/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.1041 - accuracy: 0.9813 - val_loss: 0.3871 - val_accuracy: 0.8598
Epoch 9/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0867 - accuracy: 0.9863 - val_loss: 0.3565 - val_accuracy: 0.8852
Epoch 10/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0662 - accuracy: 0.9893 - val_loss: 0.3455 - val_accuracy: 0.8959
Epoch 11/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0615 - accuracy: 0.9877 - val_loss: 0.3213 - val_accuracy: 0.9092
Epoch 12/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0474 - accuracy: 0.9903 - val_loss: 0.3126 - val_accuracy: 0.9012
Epoch 13/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0401 - accuracy: 0.9917 - val_loss: 0.3023 - val_accuracy: 0.9185
Epoch 14/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0342 - accuracy: 0.9927 - val_loss: 0.2971 - val_accuracy: 0.9146
Epoch 15/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0292 - accuracy: 0.9950 - val_loss: 0.2928 - val_accuracy: 0.9172
Epoch 16/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0201 - accuracy: 0.9997 - val_loss: 0.3000 - val_accuracy: 0.9159
Epoch 17/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0233 - accuracy: 0.9963 - val_loss: 0.3005 - val_accuracy: 0.9079
Epoch 18/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0100 - accuracy: 0.9963 - val_loss: 0.3069 - val_accuracy: 0.9002
Epoch 19/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0171 - accuracy: 0.9963 - val_loss: 0.2977 - val_accuracy: 0.9146
Epoch 20/20
2096/2096 [-----] - 21s 7ms/sample - loss: 0.0142 - accuracy: 0.9973 - val_loss: 0.2997 - val_accuracy: 0.9172

```

Figure 3-12 Efficient net with FF++

```

)

Epoch 1/15
390/390 [=====] - 210s 538ms/step - loss: 0.2658 - accuracy: 0.8895 - val_loss: 0.2961 - val_accuracy: 0.8738
Epoch 2/15
390/390 [=====] - 198s 506ms/step - loss: 0.2172 - accuracy: 0.9120 - val_loss: 0.3963 - val_accuracy: 0.8450
Epoch 3/15
390/390 [=====] - 191s 489ms/step - loss: 0.1827 - accuracy: 0.9282 - val_loss: 0.1743 - val_accuracy: 0.9267
Epoch 4/15
390/390 [=====] - 189s 484ms/step - loss: 0.1527 - accuracy: 0.9407 - val_loss: 0.2236 - val_accuracy: 0.9131
Epoch 5/15
390/390 [=====] - 188s 481ms/step - loss: 0.1338 - accuracy: 0.9480 - val_loss: 0.2158 - val_accuracy: 0.9147
Epoch 6/15
390/390 [=====] - 187s 478ms/step - loss: 0.1195 - accuracy: 0.9530 - val_loss: 0.1616 - val_accuracy: 0.9355
Epoch 7/15
390/390 [=====] - 186s 477ms/step - loss: 0.1015 - accuracy: 0.9597 - val_loss: 0.1283 - val_accuracy: 0.9525
Epoch 8/15
390/390 [=====] - 185s 474ms/step - loss: 0.0942 - accuracy: 0.9649 - val_loss: 0.1879 - val_accuracy: 0.9287
Epoch 9/15
390/390 [=====] - 185s 474ms/step - loss: 0.0812 - accuracy: 0.9693 - val_loss: 0.1664 - val_accuracy: 0.9409
Epoch 10/15
390/390 [=====] - 185s 473ms/step - loss: 0.0753 - accuracy: 0.9716 - val_loss: 0.1309 - val_accuracy: 0.9525
Epoch 11/15
390/390 [=====] - 185s 473ms/step - loss: 0.0675 - accuracy: 0.9756 - val_loss: 0.1510 - val_accuracy: 0.9473
Epoch 12/15
390/390 [=====] - 186s 476ms/step - loss: 0.0636 - accuracy: 0.9764 - val_loss: 0.1530 - val_accuracy: 0.9443
Epoch 13/15
390/390 [=====] - 185s 474ms/step - loss: 0.0629 - accuracy: 0.9767 - val_loss: 0.1048 - val_accuracy: 0.9623
Epoch 14/15
390/390 [=====] - 183s 468ms/step - loss: 0.0567 - accuracy: 0.9802 - val_loss: 0.1216 - val_accuracy: 0.9525
Epoch 15/15
390/390 [=====] - 184s 472ms/step - loss: 0.0512 - accuracy: 0.9815 - val_loss: 0.1148 - val_accuracy: 0.9629

+ Code + Markdown

[7]:
model.save('trained_model.h5')

```

Figure 3-13 Efficient net with FF++ and 140k images

```

)

Epoch 1/15
390/390 [=====] - 162s 415ms/step - loss: 0.6413 - accuracy: 0.6310 - val_loss: 1.0558 - val_accuracy: 0.5587
Epoch 2/15
390/390 [=====] - 159s 408ms/step - loss: 0.5578 - accuracy: 0.7174 - val_loss: 0.7229 - val_accuracy: 0.6080
Epoch 3/15
390/390 [=====] - 160s 411ms/step - loss: 0.4797 - accuracy: 0.7702 - val_loss: 0.8718 - val_accuracy: 0.5178
Epoch 4/15
390/390 [=====] - 158s 406ms/step - loss: 0.4183 - accuracy: 0.8092 - val_loss: 0.4091 - val_accuracy: 0.8109
Epoch 5/15
390/390 [=====] - 158s 405ms/step - loss: 0.3701 - accuracy: 0.8327 - val_loss: 0.4341 - val_accuracy: 0.8051
Epoch 6/15
390/390 [=====] - 159s 407ms/step - loss: 0.3148 - accuracy: 0.8622 - val_loss: 0.3871 - val_accuracy: 0.8239
Epoch 7/15
390/390 [=====] - 158s 404ms/step - loss: 0.2641 - accuracy: 0.8897 - val_loss: 1.1632 - val_accuracy: 0.5821
Epoch 8/15
390/390 [=====] - 158s 403ms/step - loss: 0.2245 - accuracy: 0.9067 - val_loss: 0.6209 - val_accuracy: 0.7644
Epoch 9/15
390/390 [=====] - 158s 405ms/step - loss: 0.1886 - accuracy: 0.9247 - val_loss: 0.4688 - val_accuracy: 0.8107
Epoch 10/15
390/390 [=====] - 158s 405ms/step - loss: 0.1672 - accuracy: 0.9332 - val_loss: 1.1850 - val_accuracy: 0.6468
Epoch 11/15
390/390 [=====] - 158s 406ms/step - loss: 0.1362 - accuracy: 0.9451 - val_loss: 0.4608 - val_accuracy: 0.8323
Epoch 12/15
390/390 [=====] - 157s 402ms/step - loss: 0.1256 - accuracy: 0.9510 - val_loss: 0.2733 - val_accuracy: 0.8864
Epoch 13/15
390/390 [=====] - 159s 407ms/step - loss: 0.1047 - accuracy: 0.9601 - val_loss: 0.2081 - val_accuracy: 0.9169
Epoch 14/15
390/390 [=====] - 159s 408ms/step - loss: 0.0994 - accuracy: 0.9622 - val_loss: 0.3022 - val_accuracy: 0.8934
Epoch 15/15
390/390 [=====] - 159s 408ms/step - loss: 0.0837 - accuracy: 0.9679 - val_loss: 0.1951 - val_accuracy: 0.9287

model.save('./trained_model.h5')

```

Figure 3-14 Efficient net with 140k images

3.6 CHAPTER SUMMARY

We presented a deep neural network-based approach to classify the video/image as deep fake or real, along with the confidence of the proposed model. The proposed method is inspired by the way the deep fakes are created by the GANs with the help of Autoencoders. Our method does the frame-level detection using EfficientNet0. The proposed method is capable of detecting the video/image as a deep fake or real based on the listed parameters in the paper. We believe that it will provide a very high accuracy on real-time data.

Chapter Four

4. SYSTEM GRAPHICAL USER INTERFACE

4.1 WEBSITE

Our approach is detecting the DF over the world wide web. We will be providing a web and android-based platform for the user to upload the video and classify it as fake or real.

This project can be scaled up from developing a web-based platform to a browser plugin for automatic DF detections. Even big applications like WhatsApp, Facebook can integrate this project with their application for easy pre-detection of DF 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.

4.1.1 Flask Framework

Flask is a lightweight microframework for web applications built on top of Python, which provides an efficient framework for building web-based applications using the flexibility of python and strong community support with the capability of scaling to serve millions of users. Flask has excellent community support, documentation, and supporting libraries; it was developed to provide a barebone framework for developers, giving them the freedom to build their applications using their preferred set of libraries and tool

Flask is a BSD licensed, Python microframework based on Werkzeug and Jinja2. Being a microframework doesn't make it any less functional; Flask is a very simple yet highly extensible framework. This gives developers the power to choose the configuration they want, thereby making writing applications or plugins easy. Flask was originally created by Poccoo, a team of open-source developers in 2010, and it is now developed and maintained by The Pallets Project who power all the components behind Flask. Flask is supported by an active and helpful developer community including an active IRC channel and a mailing list.

4.1.2 The Flask story

Starting as an April Fool's joke in 2010, Flask was created by Armin Ronacher - a member of an international Python enthusiast group known as Pocoo. The project became a quick success, and the Pocoo team managed the development of Flask until 2016. After the team disbanded, the management of Flask was transferred to the Pallets Projects group.

Flask is named after an earlier Python micro-framework called Bottle. It is easily one of the most popular Python web-development frameworks, coming in just slightly behind Django in terms of the number of stars on GitHub.

4.1.3 Advantages of the Flask framework

1. Easy to understand development

The Flask framework is easy to understand, that is why it is best for beginners. Its simplicity gives you the opportunity to understand it better and learn from it. There are interesting features to use in the framework. The simplicity in the flask framework enables the developer to navigate around and create the application easily.

Unlike other web application frameworks, flask let you be in total control in web development taking full creative control of the application and web development. The developer has the chance of being "in the drivers seat", taking charge of what you want to do like adding external features.

2. It is very flexible and easy.

There are only a handful of parts of flask that cannot be changed or altered because of its simplicity and minimalism. This means that almost all the parts of flask are open to change, unlike some other web frameworks. Flask comes with a template engine that lets you use the same user interface for multiple pages. Python can insert variables into the templates.

3. Testing

Using Flask for web development allows for unit testing through its integrated support, built-in development server, fast debugger, and restful request dispatching. It is lightweight to enable you to transit into a web framework easily with some extension.

4.1.4 Flask API

Flask API is defined as a methodology to make HTTP calls to the server for getting the data to populate the dynamic parts of the application.

Flask is a framework that allows users to fetch data for the server, and corresponding to the use case, API is the most preferred methodology.

The data that Flask API retrieves can be represented in a database, stored in a file, or accessible by network protocols.

The job of API is to decouple the data and the application and take care of hiding the data implementation details.

Website context diagram:

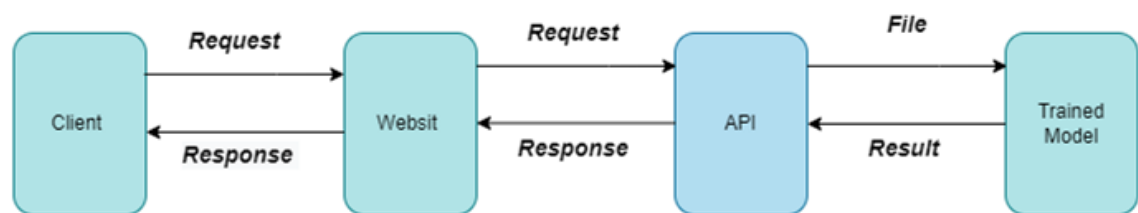


Figure 4-1 Website context diagram

Website activity/sequence diagram:

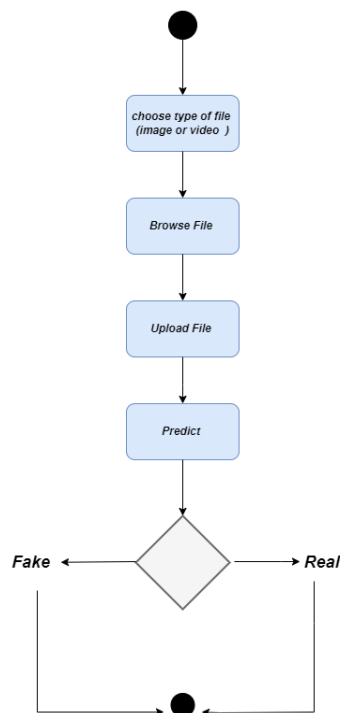


Figure 4-2 Website activity diagram

4.1.5 Implementation of API

```
# Import libraries

from flask import Flask, render_template, request, redirect, url_for
from keras.models import load_model
import numpy as np
import cv2
import os
from flask import Flask
from werkzeug.utils import secure_filename
import os
from tensorflow import keras
from flask import Flask, request, render_template
from werkzeug.utils import secure_filename
import dlib
import tensorflow as tf
from tensorflow.keras.preprocessing.image import img_to_array, load_img
```

Figure 4-3 Import some libraries that use in API

```

app = Flask(__name__ , template_folder='static/templates')
@app.route('/')
def main():
    return render_template ("Home.html")

#*****

@app.route('/upload_video', methods=['POST'])
def Video():
    return render_template ("Upload_Video.html")

@app.route('/upload_image', methods=['POST'])
def image():
    return render_template('Upload_image.html')
@app.route('/About-us', methods=['POST'])
def About_us():
    return render_template('About-us.html')

```

Figure 4-4 Display Home page and redirect to display upload image, video and about us pages

```

def prediction (filepath):
    input_shape = (128, 128, 3)
    pr_data = []
    detector = dlib.get_frontal_face_detector()
    cap = cv2.VideoCapture(filepath)
    frameRate = cap.get(5)
    while cap.isOpened():
        frameId = cap.get(1)
        ret, frame = cap.read()
        if ret != True:
            break
        if frameId % ((int(frameRate)+1)*1) == 0:
            face_rects, scores, idx = detector.run(frame, 0)
            for i, d in enumerate(face_rects):
                x1 = d.left()
                y1 = d.top()
                x2 = d.right()
                y2 = d.bottom()
                crop_img = frame[y1:y2, x1:x2]
                data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
                data = data.reshape(-1, 128, 128, 3)
                print((model.predict(data) > 0.5).astype("int32"))
                outt=''
            if ((model.predict(data) > 0.5).astype("int32")[0][0] == 0 ):
                return 'Real'
            else:
                return 'Fake'

```

Figure 4-5 Prediction Function

```

@app.route('/predict_video', methods=['POST'])
def upload_video():
    file = request.files['file']
    filename = secure_filename(file.filename)
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
    filepath = "static/"+filename
    preds = prediction(filepath)
    return render_template("Display_Video.html",prediction =preds ,video_path = filename)

```

Figure 4-6 Video prediction and Display Result

```

@app.route('/Predict_image', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        f = request.files['file']
        file_path = os.path.join (app.config['UPLOAD_FOLDER'], secure_filename(f.filename))
        f.save(file_path)
        # Make prediction
        preds = prediction(file_path)
    return render_template("Display_image.html",prediction = preds, img_path= f.filename )

```

Figure 4-7 Image prediction and display result

```

model = tf.keras.models.load_model('model/deepfake-detection-tensor.h5')

"""# Run Flask Application """

if __name__ == '__main__':
    app.run()

```

Figure 4-8 Loading Trained Model and Run API

4.1.6 Web Graphical User Interface

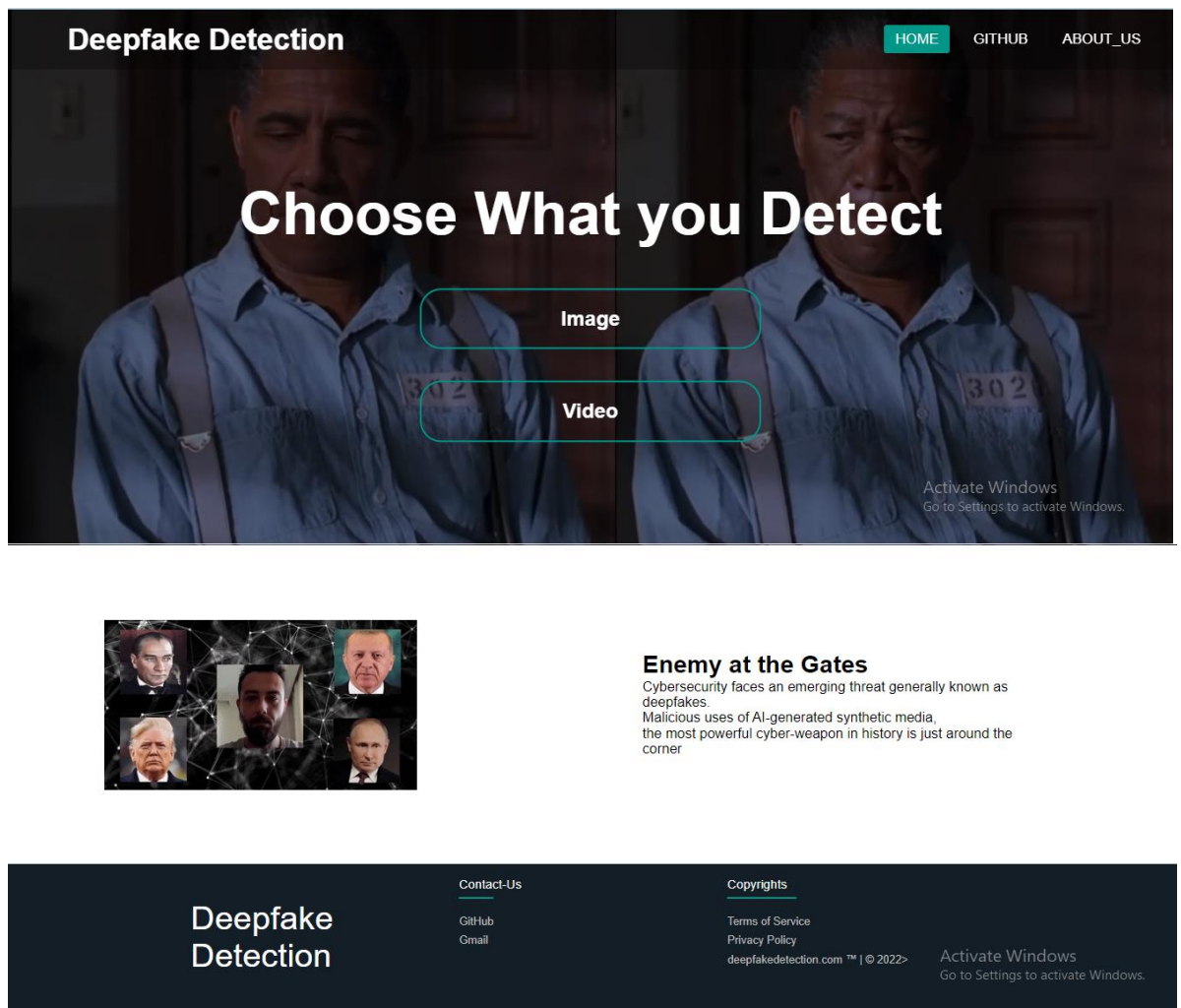


Figure 4-9 Home page

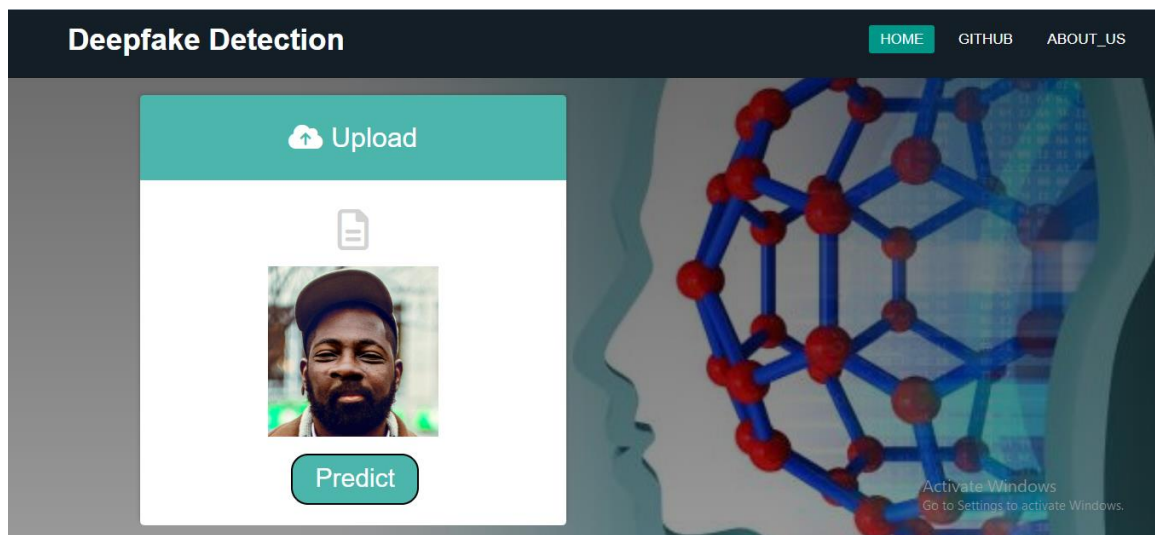


Figure 4-10 Upload image page

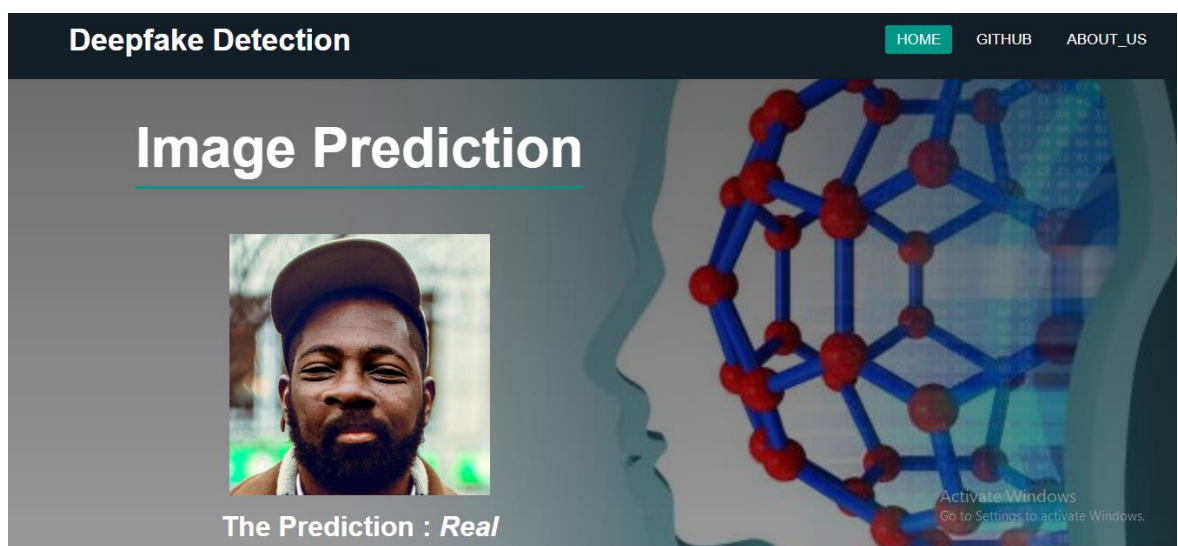


Figure 4-11 Image Prediction Page

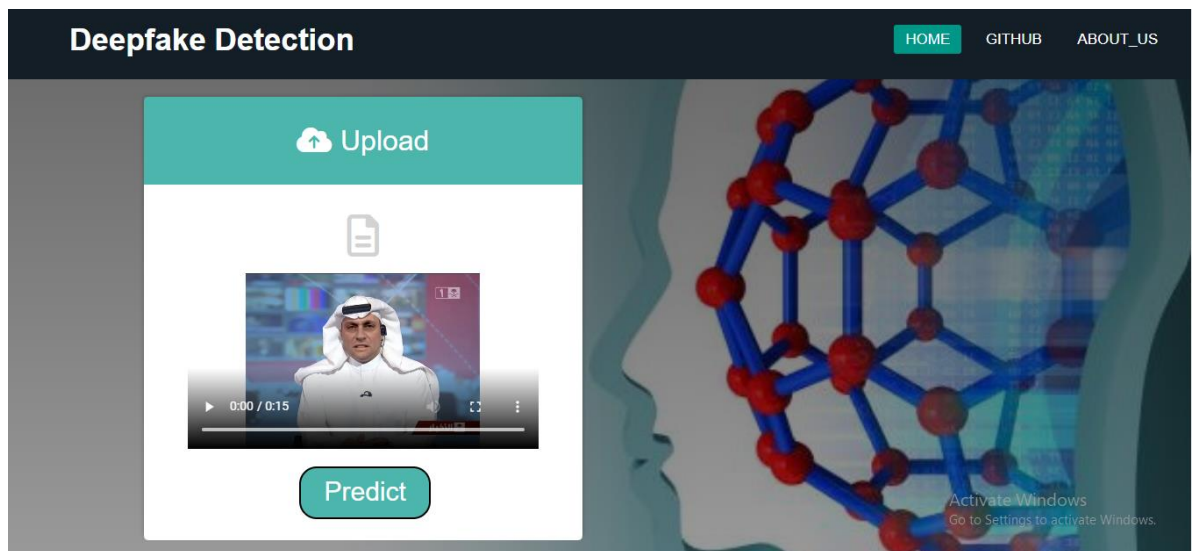


Figure 4-12 Upload Video Page



Figure 4-13 Video Prediction Page

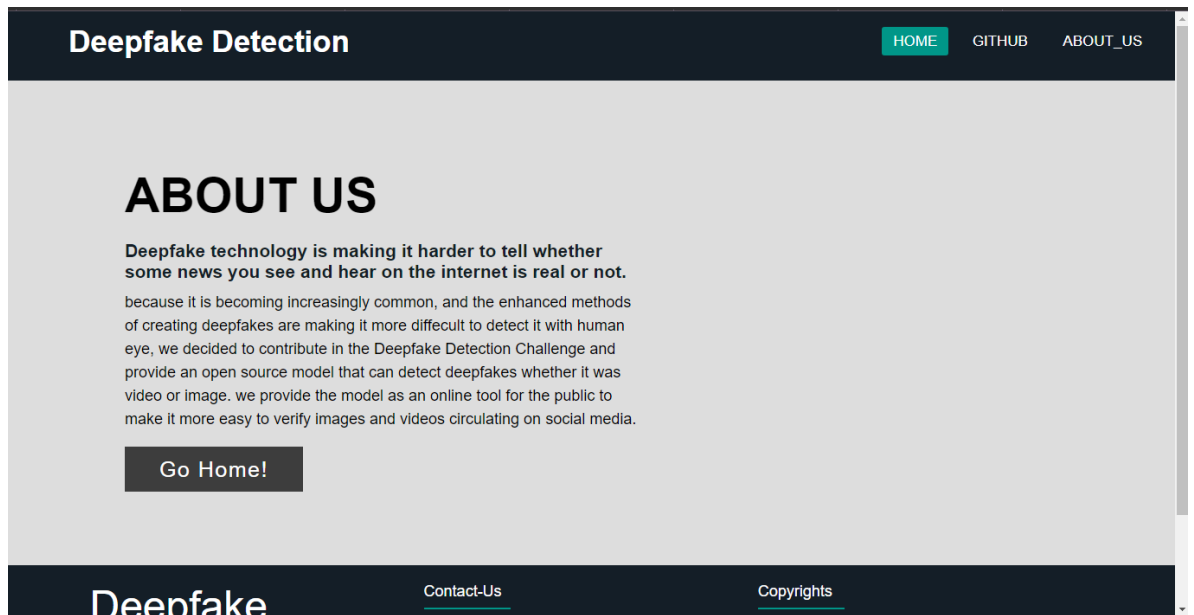


Figure 4-14 About us page

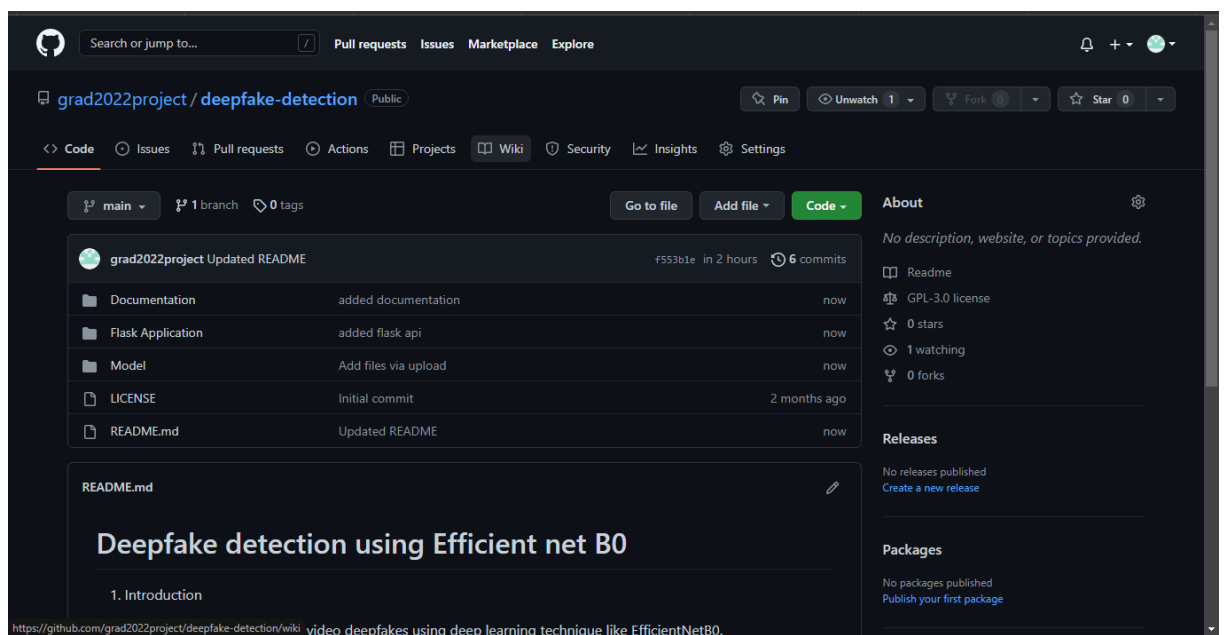


Figure 4-15 GitHub page

4.2 ANDROID APPLICATION

We developed our application with webview. Android WebView is a Chrome-powered system component that enables Android applications to access web pages it can display web pages inside your application. You can also specify HTML string and can show it inside your application using WebView.

It is a significant feature of Android to increase speed and security. Any third-party app can use Android WebView to combine applications' calls efficiently. It makes the application lighter and fast.

It prefers to offer its apps the ability to access and interact with online content inside the applications themselves.

First, the WebView library will be activated, and a WebView class instance is generated. The Android Declaration file consists of the code and web permissions. This enables the launch of web pages from inside applications. These steps will help the browser to enable the rendering of web pages and JS code.

Our application is made with Java language using Android Studio.

4.2.1 Application Graphical User Interface

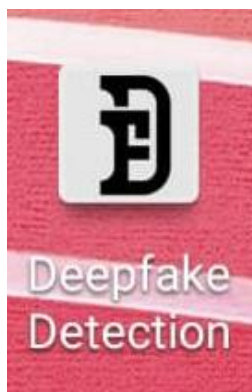


Figure 4-16 Application icon

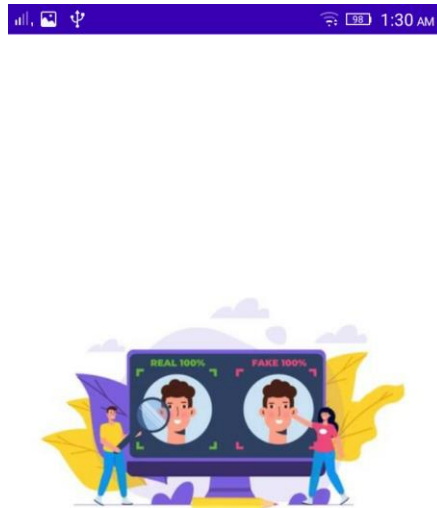


Figure 4-17 Start screen

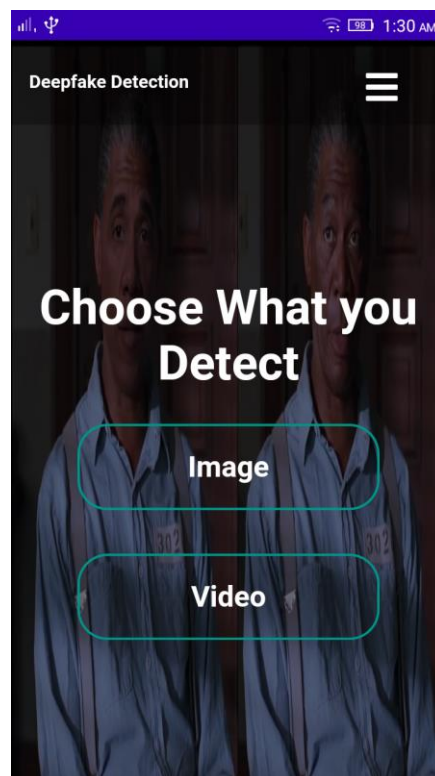


Figure 4-18 Home page

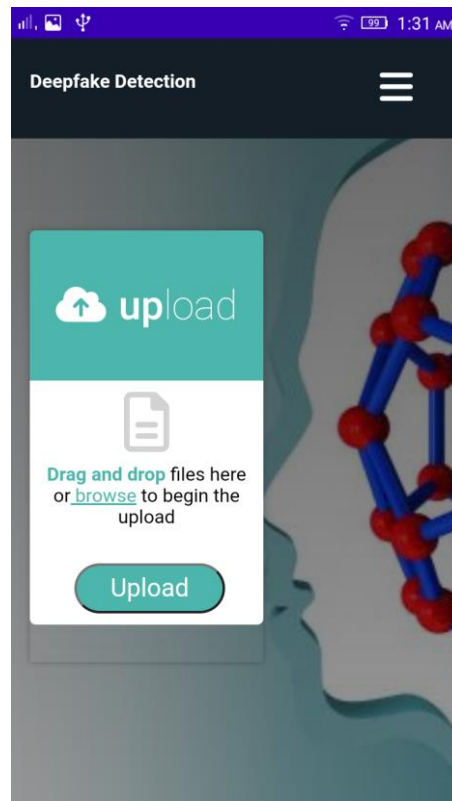


Figure 4-19 Upload image or video

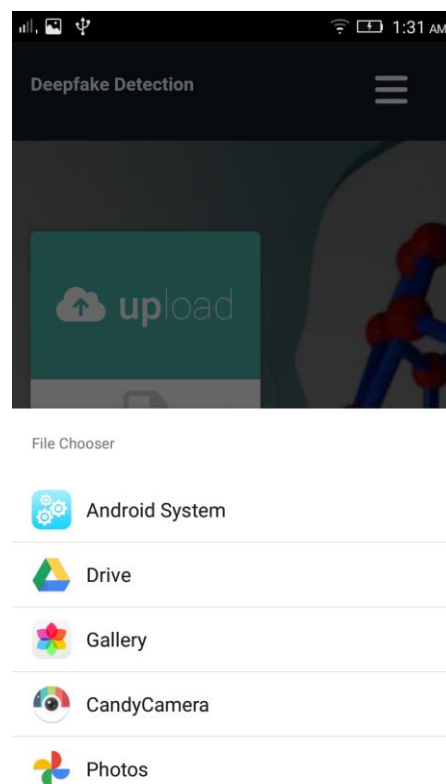


Figure 4-20 Browse

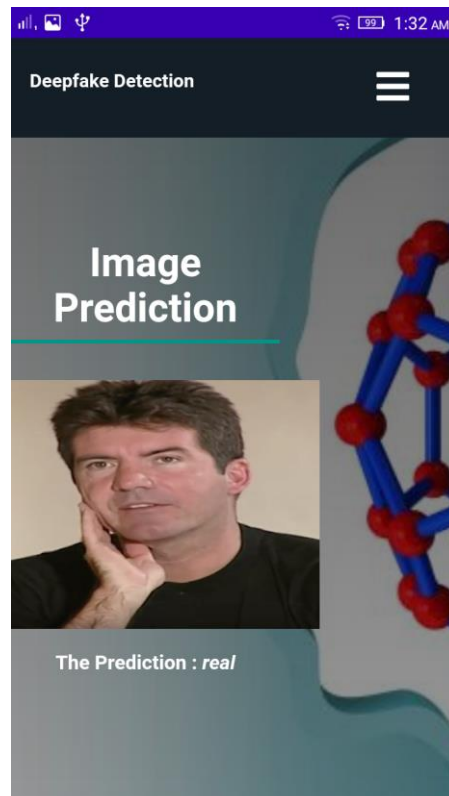


Figure 4-21 Prediction result

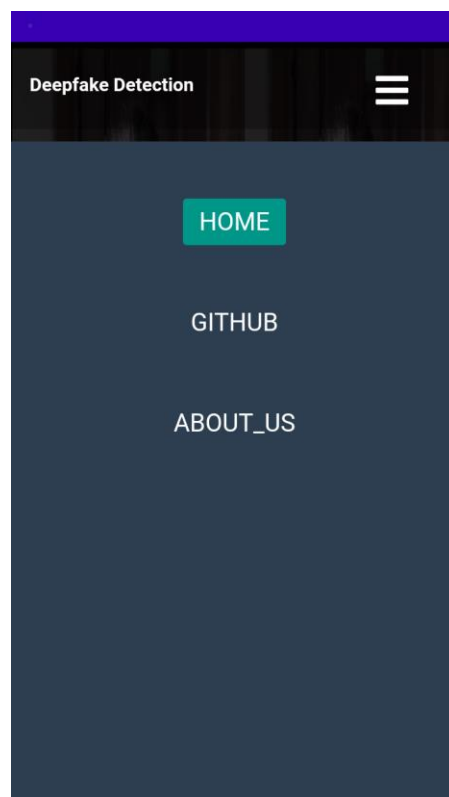


Figure 4-22 Menu navigation

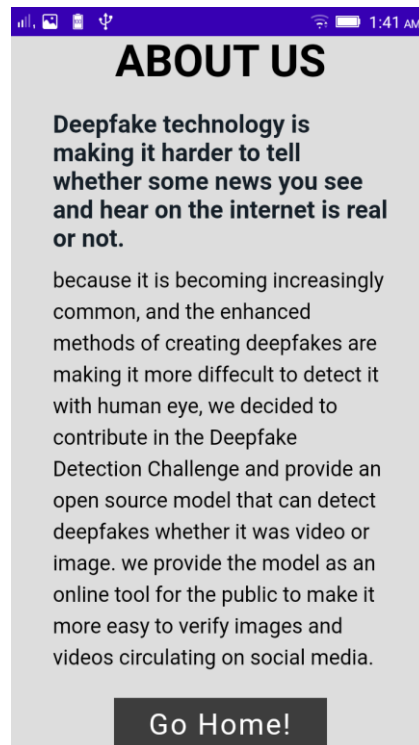


Figure 4-23 About us page

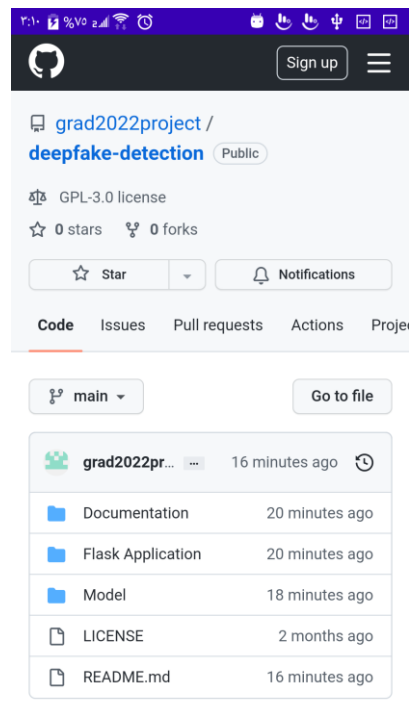


Figure 4-24 GitHub page

5. REFERENCES

- [1] Yuezun Li, Siwei Lyu, "ExposingDF Videos By Detecting Face Warping Artifacts," in arXiv:1811.00656v3.
- [2] Nguyen, Thanh Thi, et al. "Deep learning for deepfakes creation and detection." arXiv preprint arXiv:1909.11573 1 (2019): 2.
- [3] <https://science-union.org/articlelist/category/Review>
- [4] <https://towardsdatascience.com/art-of-generative-adversarial-networks-gan-62e96a21bc35>
- [5] <https://www.hindawi.com/journals/cin/2022/3441549/fig4/>
- [6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In NIPS, 2014.
- [7] David G'uera and Edward J Delp. Deepfake video detection using recurrent neural networks. In AVSS, 2018.
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, 2016.
- [9] An Overview of ResNet and its Variants: <https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035>
- [10] Sequence Models And LSTM Networks
https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html
- [11] <https://www.kaggle.com/c/deepfake-detection-challenge/data>
- [12] <https://github.com/ondyari/FaceForensics>
- [13] Y. Qian et al. Recurrent color constancy. Proceedings of the IEEE International Conference on Computer Vision, pages 5459–5467, Oct. 2017. Venice, Italy.

- [14] R. Raghavendra, Kiran B. Raja, Sushma Venkatesh, and Christoph Busch, “Transferable deep-CNN features for detecting digital and print-scanned morphed face images,” in CVPRW. IEEE, 2017.
- [15] Nicolas Rahmouni, Vincent Nozick, Junichi Yamagishi, and Isao Echizen, “Distinguishing computer graphics from natural images using convolution neural networks,” in WIFS. IEEE, 2017.
- [16] F. Song, X. Tan, X. Liu, and S. Chen, “Eyes closeness detection from still images with multi-scale histograms of principal oriented gradients,” *Pattern Recognition*, vol. 47, no. 9, pp. 2825–2838, 2014.
- [17] D. E. King, “Dlib-ml: A machine learning toolkit,” *JMLR*, vol. 10, pp. 1755–1758, 2009.
- [18] Yu, Peipeng, et al. "A survey on deepfake video detection." *Iet Biometrics* 10.6 (2021): 607-624.
- [19] <https://www.privacyaffairs.com/deepfakes/>
- [20] Yuezun Li, Ming-Ching Chang, and Siwei Lyu “Exposing AI Created Fake Videos by Detecting Eye Blinking” in arxiv.
- [21] Umur Aybars Ciftci, İlke Demir, Lijun Yin “Detection of Synthetic Portrait Videos using Biological Signals” in arXiv:1901.02212v2.
- [22] Hyeongwoo Kim, Pablo Garrido, Ayush Tewari and Weipeng Xu “Deep Video Portraits” in arXiv:1901.02212v2.