DETECTION OF DEEPFAKE VIDEOS ON SOCIAL MEDIA

by

Sanathkumar Adavayya Swamy (1096761)

Supervisor: Dr Jorge Goncalves

A thesis submitted in total fulfillment for the
degree of Master of Computer Science

in the
Department of Computing and Information System
Melbourne School of Engineering
THE UNIVERSITY OF MELBOURNE

November 2021

Subject code: COMP90080 and COMP90081.

THE UNIVERSITY OF MELBOURNE

Abstract

Department of Computing and Information System
Melbourne School of Engineering

Master of Computer Science

by Sanathkumar Adavayya Swamy (1096761)
Supervisor: Dr Jorge Goncalves

In the last few years, machine learning algorithms have been developing and improving over time. Deepfake technology is one such application that is built using an efficient machine learning algorithm. Deepfake uses a face-swapping technique that allows anyone to replace faces in a video, resulting in a high-quality video. Users using this technology with malicious intent can make a significant impact on society's values. For instance, users can spread false news about an individual, which can cause reputational damage. Social media has been playing a vital role in spreading false information with the use of deepfake videos. Hence, detecting deepfake videos and preventing them from spreading false news on social media platforms is necessary. In this research, we propose a deepfake detection system that detects videos explicitly on social media platforms to avoid spreading false information. An existing study in deepfake detection suggests that the high-quality deepfake videos spread across social media are difficult to detect. Therefore, we tackle this problem by designing a deepfake detection system based on social media video requirements. This research focuses on building a detection system to overcome existing systems' issues and detect high-quality videos. Inspired by state-of-the-art, we have proposed a system that has used augmented deepfake video frames to train three different CNN models. We have computed the best predictions to classify high-quality deepfake videos on social media using an ensemble method.

Keywords: CNN (Convolutional Neural Network), PRNU (Photo Response Non-Uniformity), GAN (Generative Adversarial Network), Deepfake, Machine Learning, social media, False News, Deep Neural Network (DNN), Augmented.

Declaration of Authorship

I certify that:

- This thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.
- The thesis is fewer than 30000 words in length (excluding text in images, table, bibliographies, and appendices).

Signed: sanathkumar

Date: 01/11/2021

Acknowledgements

I would first like to thank my supervisor, Professors Jorge Goncalves for his guidance, motivation, insightful comments, and sharing his knowledge through the entire research and writing of this dissertation. My sincere thanks to my family and friends for their ongoing support and continuous encouragement throughout my years of study.

Contents

List of Figures

List of Tables

Abbreviations

DNN	**D**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
GAN	**G**enerative **A**dversarial **N**etwork
SVM	**S**upport **V**ector **M**achine
PRNU	**P**hoto **R**esponse **N**on **U**niformity

Chapter 1

Introduction

In recent times, the number of people using social media to gain knowledge has drastically increased. News and various types of content shared on these platforms are numerous, and users tend to consume more information from here than the traditional methods [2]. One of the most common methods used to share information is a video-based format. Most video-based social computing applications use short-length videos to share information. These platforms have a significant advantage in communicating information to a larger audience and sharing it rapidly amongst its users [2].

Users have misused social media to spread false information, which is also known as fake news. In video-based social computing applications, fake news is being generated by manipulating the videos. A method to manipulate videos is known as Deepfake, and it makes the videos look authentic in people's eyes. The deepfake technology swaps faces and modifies various video features to target an individual or an organization. The high quality of videos generated by this method has recently been used to create threats to people worldwide [3]. Deepfake makes people question the trustworthiness of a visual experience on a social media platform. According to an article [4], the recent development in technology is a significant reason why computer scientists, digital media forensics, and others find it challenging to detect deepfake videos. The first awareness of the existence of deepfake was shown in 2018 by a comedian named Peele, where he manipulated former U.S President Obama's video conveying maliciousness of the technology [4]. Since then, deepfake technology has been evolved with various features.

It has been a more significant challenge for social media giants like Facebook, Twitter, and Instagram to track these videos and avoid spreading false information.

Deepfake technology is an increasingly sophisticated machine learning model combined with inexpensive, easy-to-use editing software [3]. This software has been used as a technological weapon to target an individual or an organization. Users of deepfake software can be categorized into two prominent use cases:

1. Malicious Intent: The user intends to use deepfake as a weapon to harm an individual or an organization. The negative consequences of deepfake are non-consensual pornography, political disinformation, and financial fraud [3]. Deepfake can harm an individual by causing reputational damage, or it can also cause harm to society by undermining social values such as trust in an organization. Over the years, few reported examples have seen the effect of deepfake used for malicious intent. One such example is that a shooting survivor in Parkland, U.S.A named Emma had uploaded a video of her tearing down the shooting range target into pieces to show her anger towards gun use in the U.S.A [4]. This video was manipulated and showed the viewers Emma tearing the U.S.A constitution instead of the shooting range target. This video made the viewers create propaganda and defame her. Deepfake has not only meddled with people's reputation but has also been causing deaths [4]. One such example is when a journalist in India who reported corruption against a politician was sexually assaulted and lynched by the local community due to a pornographic deepfake video. The malicious intent of the users makes this an urgent challenge for researchers to find an effective solution.

2. Unharmful Intent: Most social media users have been using face swap techniques to create humorous content for entertaining their users. For Instance, Snapchat, a video-based social computing application, has face-swap filters, which users can use to create content and share it. Figure1.1 illustrates a face swap technique used, swapping my face with Donald Trump's face. Deepfake image shown in figure1.1 can be termed unharmful, as it does not motivate to spread any fake news. Nevertheless, these manipulated contents can easily detect by humans without any effort needed. Researchers have also shown that deepfake can be used for good reasons as well. For example, a research published a solution using the deepfake technology to protect patient's privacy and clinical data by encrypting

it using deepfake[5]. The researchers used the Deepfake algorithm to manipulate the patient's faces in clinical video data. In these examples mentioned above, the intent of the users to use the deepfake application can be termed unharmful.

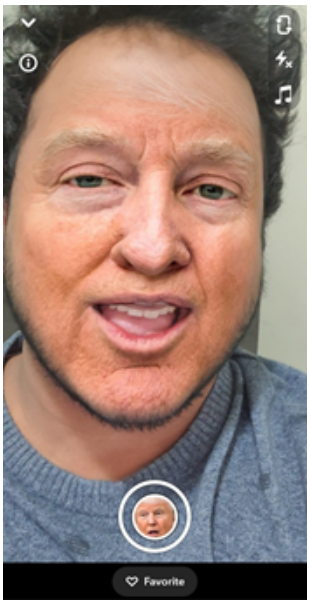

FIGURE 1.1: Non-malicious deepfake created using Snapchat (Social media platform)

The researchers and other experts have been focusing on detecting malicious deepfake videos over the years. However, few researchers and experts have succeeded in presenting various approaches to detecting Deepfake videos [6–14]. These methods mainly focus on flaws present in the deepfake videos. The detection system extract features from videos and makes the DNN algorithms learn to group videos into two different types, deepfake and non-deepfake (original videos).

However, the updated versions of the deepfake software have been generating high-quality videos, making it hard for the detection system to find flaws or group them. Additionally, an inbuilt filter function in social media platforms allows the users to change or augment videos and images before posting them on the forum. These functions on social media platforms make it even more challenging for the existing deepfake detection systems to detect flaws in deepfake videos.

Improvements in deepfake technology and social media applications are proportionally increasing, allowing people to create and spread malicious content. Our primary motivation for the research project is to solve the issues in the existing detection system. Therefore, in this research, we will be proposing a deepfake detection system that can

detect dynamic deepfake videos on social media platforms. Additionally, this research project would also provide an insight into how social media platforms can implement deepfake detection systems in their applications.

Chapter 2

Literature Review

2.1 Understanding Deepfake Technology

Deepfake technology has been an easy-to-use tool for most users, where a complex level of deep learning model is combined with software making it very simple to interact [4]. Researchers have conducted in-depth research on the most used deepfake applications. L.Siwei [7] has stated that DNN (Deep Neural Network), an advanced machine learning algorithm, is mainly used to create deepfake videos. The author has also traced the first deepfake model made, and it was published on Reddit by a user named deepfake, which later gained popularity. Users used this DNN-based face-swapping technology to create fake pornographic videos of celebrities. The development in DNN and deepfake technology have proportionally improved over time. Mahmud et al. [9] have published an article showing a deep insight into deepfake technology. It states that pornographic deepfake videos are the most dangerous and harmful content created using deepfake applications, which causes reputational damages to famous individuals.

Researchers have created a timeline showing improvements in the deepfake applications, where the first open-source deepfake application was FakeApp [7, 9, 15]. FakeApp used a large junk of image data to swap a person's face with another. The application used DNN to swap a soured face with a targeted video by splitting it into frames and generating a fake video. The previous study showed that the initial deepfake application was improvised by implementing the latest developments in DNN based algorithms [7, 9].However, some of the popular open-source and free deepfake software used by the users

are Dfaker [16], Faceswap – GAN [17], face swap [18], deepfakeLab [19], faceswapPytorch [20], and other online services. Research suggests that these deepfake applications have developed over time, as they have been generating flawless and high-quality deepfake videos, making it difficult to detect them by simple observation [9]. One of the most used open-source tools is deepfakeLab [19]. The tool produces faster and high-quality deepfake videos than other deepfake applications and updated versions [9].

A survey published by Yisorel et al. [14] gives a deeper insight into these deepfake applications, showing the technical background of the software and making a detailed comparison amongst them. The authors have categorized the deepfake algorithm into three different attack methods, such as:

- Facial Re-enactment: This manipulation method allows the attacker to tamper with the individual's emotions and expression. It controls a person's actions and tamper with evidence by creating false information [14]. Figure 2.1 shows an example of a re-enactment attack where a person's Gaze, mouth, expression, and pose have tampered with.

FIGURE 2.1: Facial Re-enactment

- Facial Replacement: This method is used to swap a source face to a target face. Previous studies have suggested that this attack category is more dangerous than other attacks [7, 9, 14]. Face swapping is used on famous people to create pornographic videos, as the malicious intent of the attacker here is to damage the reputation of a person or be used for blackmailing. Figure 2.2 shows the replacement methods being used to swap the faces of one person to another.

FIGURE 2.2: Facial Replacement

- Facial Editing and Synthesis: This method of attack is used to change the various aspects, including hair, age, beauty, and ethnicity of a person. As shown in Figure 2.3, a person's multiple aspects have been tampered with using deepfake. Nevertheless, this attack can be used for non-malicious intent as well. For instance, Snapchat uses these features to attract more users to the platform.

FIGURE 2.3: Facial Editing and Synthesis

A detailed survey conducted by researchers suggests that the common and the most used deep neural network for building these attack models is GAN (Generative Adversarial Network) [7, 9, 14]. This neural network is an unsupervised learning model, which is a combination of two neural networks. Generator and discriminator are the two models present in GAN. The generator model is used to create new examples of the input dataset, and the discriminator classifies the dataset as real or fake. This model can be used for fast learning and produce flawless manipulated content with more input data. The researchers are making people more aware of the malicious spread of deepfake content by publishing surveys.

2.2 Detection of Deepfake Videos

While the usage of deepfake videos has made people question the authenticity of the content on social media platforms. Nevertheless, researchers have published various approaches to detect deepfake videos. These approaches mainly focus on detecting flaws

in deepfake videos and classifying them as fake or original videos. Generalizing these approaches presented by various researchers, the survey shows that it can be categorized into three main detection methods [14].

2.2.1 Classic Machine Learning Methods (Classifier)

The classification technique is based on a mathematical formula to categorize the deepfake videos as fake or original. Since detecting deepfake video is a two-group classification problem, one must decide whether the video is fake or original. One such classification algorithm which fits precisely to solve this problem is SVM (Support Vector Machine). Support Vector Machine uses a classification algorithm for two-group classification problems, where it determines if a video is fake or not. The dataset used by various researchers consists of two groups, one group consists of the original videos, and another group is deepfake videos [21–27]. The features from these datasets are extracted accordingly and vectorized to represent the numerical format. The vectorized features used by researchers in their dataset are a person's Gaze, mouth, and expression. Figure 2.4 shows a sample of a graphical representation of these two groups of datasets. The SVM model categorizes the data by forming a decision boundary, as shown in Figure 2.4. Previous studies have stated that the SVM classifier is best suited to detect deepfake videos [21–27]. It uses less computational power to classify the data with various features and produces high accuracy.

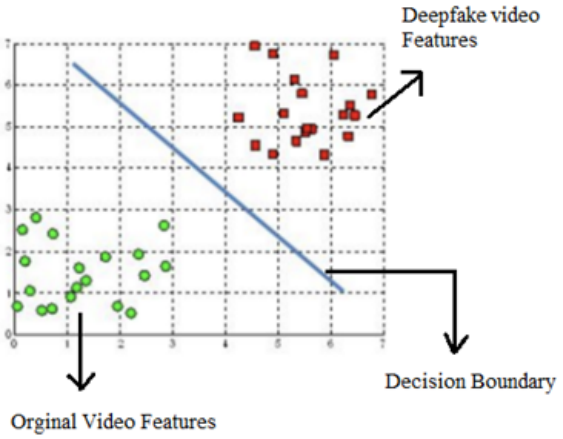

FIGURE 2.4: Graphical representation of SVM

Table 1 gives more details of various researchers using the classification method to detect deepfake videos.

Research	Deepfake Category	Classifier Method	Performance (Accuracy %)
Y. Zhang et al. [28]	Replacement	SVM-RBF	92.9
A. Agarwal et al. [22]	Replacement	SVM	81.8
X. Yang et al. [23]	Replacement and Re-enactment	SVM	92.5
P. Korshunov et al. [24]	Replacement and Re-enactment	SVM	96.6
R. Durall et al. [25]	Replacement	SVM, K-means	100
Z. Akhtar et al. [26]	Replacement and Re-enactment	SVM	86.7
S. Agarwal et al. [27]	Replacement and Re-enactment	SVM	93

TABLE 2.1: Deepfake Detection Systems using Classification Method.

As mentioned in the introduction, the deepfake methods replacement and Re-enactment are the most malicious methods used. Researchers using the classification method mainly focus on detecting replacement and re-enactment approaches of deepfake technology [21–27]. Table 2.1 shows that various researchers use SVM and classification methods to detect deepfake videos [21–27]. R. Durall et al. [25] have trained an SVM model based on the face artifacts, where the features mainly included a person's Gaze, mouth, and expression. The SVM model here is trained to differentiate between the original video and a face-swapped video. Therefore, the author has achieved an accuracy of 96.6% using two publicly available datasets. Whereas researchers [25, 28] have combined two different classification methods to detect deepfake videos, the researchers have used a more significant number of data to train their models to detect the replacement attack method of deepfake technology. However, R. Durall et al. [25] have achieved an accuracy of 100% by training the machine learning model with very little data. Before training the classifier, the research states that the converting the videos in the dataset to frames or images make the system more efficient in grouping the videos.

However, previous studies have suggested that classification method-based deepfake detection systems have major disadvantages [21–27]. Generalizing the drawbacks and limitations faced by the researchers using a classifier are as follows:

- Classification models like SVM tend to find it challenging to detect long-length deepfake videos.
- Most of the classification model-based deepfake detection systems have been trained using a short-length deepfake video dataset. Therefore, these detection systems will find it hard to detect longer-length deepfake videos.

- As deepfake applications use a DNN algorithm to generate deepfake videos, they improve over time. Therefore, classification models would find it challenging to detect high-resolution deepfake videos.

Due to these disadvantages, a previous study suggests that a classification-based detection system has lower performance than other detection systems like a deep neural network [14]. Apart from the disadvantages, one main benefit we observed in these existing systems is that they use an efficient pre-processing method, which converts videos in the dataset to frames. Researchers [21–27] suggest that this pre-processing method can be beneficial in building a low computational and highly efficient detection system.

2.2.2 Steganalysis method

The steganalysis method of detecting deepfake is performed using open-source forensic tools developed by government agencies and forensic experts. These tools primarily analyse subtle features and patterns for the models, and they are created using a machine learning classifier. However, some tools are built using a neural network algorithm and a machine learning classifier. PRNU (Photo Response Non-Uniformity) is the most used forensic tool by researchers and experts to detect deepfake videos. This tool is built using mainly machine learning classifier such as Regression, SVM (Support Vector Machine).

Photo Response Non-Uniformity (PRNU) uses primarily uses pixels to detect manipulation in videos. When uniform light falls on the camera sensor, each pixel shows the same values, a slight difference in output values is detected by PRNU. An analysis is made to determine whether the same digital camera is used to produce the digital images. A PRNU is trained with several video frames to detect deepfake videos. Similar to the classification method of detecting deepfake videos, PRNU can only train the classifier with images rather than videos. However, a few researchers have used PRNU analysis to detect deepfake videos [29–31].

J. Straub [29] has used a forensic tool called FFmpeg to conduct PRNU analysis on deepfake video frame's features. Here, the authors have used two image-based datasets, where the original and deepfake videos are converted into frames and used as datasets. The original video frames are compared with the deepfake video frames to determine the likelihood ratio. The classifier learns these sets of comparisons made using several

video frames before classifying deepfake videos. However, other researchers have used a similar method, but instead of focusing on the feature of the frames, they have focused more on comparing the pixels of images in the facial regions of the frames. Researchers proved that analysing the pixels in the facial region can achieve higher accuracy than analysing the video frame's features [30, 31]. Table 2.2 gives an overview of researchers using the forensic tool to detect deepfake videos.

Research	Deepfake Category	Method	Performance (Accuracy %)
J. Straub [29]	Replacement and Re-enactment	PRNU	34
F. Marra et al. [30]	Replacement and Re-enactment	PRNU	~40
H. Mo et al. [31]	Replacement and Re-enactment	PRNU	90

TABLE 2.2: Deepfake Detection Systems using Forensic Tools.

Table 2.2 shows that researchers have used the PRNU analysis and achieved low accuracy in detecting deepfake videos compared to the machine learning method. However, H. Mo et al. [31] have achieved higher accuracy than other researchers using PRNU to detect deepfake. The author's detection system has primarily focused on detecting deepfake video by analyzing the pixel's brightness values around the eye region of the face. The author then concludes the research by stating using PRNU to analyse pixels can provide better performance than interpreting the feature of the images.

However, researchers using PRNU analysis to detect deepfake videos have clearly stated various drawbacks to their methods [29–31]. They are generalized as follows:

- PRNU can only detect deepfake in images but not videos. As the videos are converted into frames, the forensic tool finds it challenging to identify the frame's source.
- J. Straub [29] specifies that the dataset used to train forensic tools consists of short-length videos, i.e., that the PRNU can only train and predict video with 10 seconds of length.
- The accuracy and efficiency of a PRNU analysis are affected by the video quality. For instance, if the deepfake video has a higher resolution than the original video, the model will find it harder to detect deepfake videos.Therefore, one of the limitations is that the original and deepfake videos should have a similar resolution.
- The Steganalysis method of detecting deepfake would find it hard to detect the high-quality deepfake videos generated from updated versions of deepfake software.

Forensic tool-based deepfake detection systems have a similar drawback to the classification method because these tools are built using classification algorithms. The only difference between a classification-based detection system and forensic tool-based detection systems is that different features train its classifier. In a classification method, as stated before, it uses facial region as its main feature. In contrast, a forensic tool-based detection system considers the pixel rate, brightness, and other video features. Both methods' drawbacks would be a significant setback in detecting deepfake videos on social media platforms due to their low accuracy in detecting high-quality deepfake videos.

2.2.3 Deep Neural Network method

The deep neural network is a set of algorithms inspired by the biological structure and functioning of a brain, which is used to train a machine to do impossible tasks. A survey states that most researchers prefer to use this methodology to develop a deepfake detection system due to its functionalities that work in favour of detecting high-quality deepfake videos [14].

Few researchers have used steganalysis and classification methods to develop detection systems, where they are mainly focusing on detecting flaws in a single feature of the deepfake video [21–31]. However, in deep neural network-based detection systems, the model uses neurons to compute multiple features of deepfake videos to classify a deepfake video.

Figure 2.5 is an example of a deep neural network consisting of multiple neurons. These neurons send a generated output as a signal to other neurons, forming a complex network that keeps developing and learning better each time using a feedback mechanism.

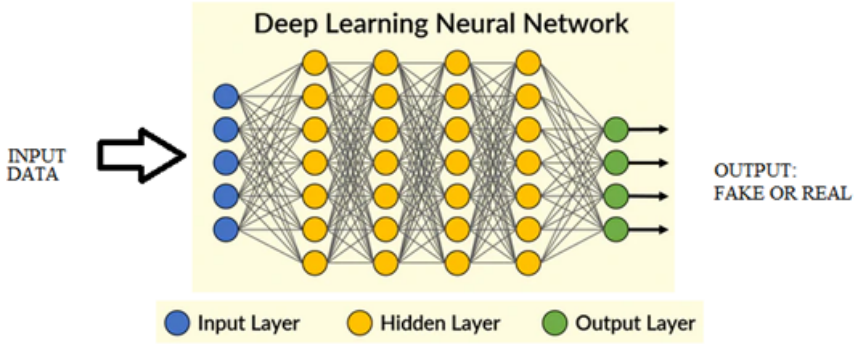

FIGURE 2.5: Deep Neural Network Architecture.

As mentioned before, deepfake technology approaches have been developed using a deep neural network algorithm. Most researchers have used advanced DNN methods to detect deepfake videos and this method tends to achieve higher accuracy than other methods [32–38]. A survey states that researchers have used a convolutional neural network (CNN) algorithm to detect deepfake videos [14]. CNN is the best method that can be used for image or video-based datasets. One of the main reasons for using CNN is that it automatically detects the essential features without human supervision. CNN algorithm does not require any pre-processing methods to be implemented on datasets to identify crucial features in a video. The algorithm's architecture is designed to learn to find flaws in deepfake video without human intervention. The survey gives a detailed analysis of all the deep neural network-based detection systems, comparing them with each other. Based on the study, Table 2.3 shows the best-performing models of a few researchers who have used DNN to detect deepfake videos [32–38].

Research	**Deepfake Category**	**Model**	**Performance (Accuracy %)**
H. H. Nguyen et al. [32]	Replacement and Re-enactment	CNN	99.4
M. Du et al. [33]	Replacement and Re-enactment	Capsule - CNN	99.3
T. Fernando et al. [34]	Replacement and Re-enactment	CNN AE GAN	99.2
J. Li et al. [35]	Replacement and Re-enactment	CNN+HNM	99.4
N. Yu et al. [36]	Replacement and Re-enactment	FCN	98.1
M. S. Rana et al. [37]	Replacement and Re-enactment	CNN	99.6
A. Mitra et al. [38]	Replacement and Re-enactment	Ensemble	99.65

TABLE 2.3: Deepfake Detection Systems using Deep Neural Network method.

H. Mo et al. [31] have implemented a basic CNN algorithm using Google's TensorFlow machine learning framework. The author's dataset consists of local images which are generated using open-source deepfake applications. The facial region of the images on the dataset is cropped and stored. The dataset is then used to train the CNN model, where the model is focused on considering only three facial features from the data.

The authors tend to achieve an accuracy of 99.4%, which is better than classification-based and forensic tool-based detection systems. However, T. Fernando et al. [34] have used a similar detection system method, but the author has a different dataset to train the model. However, researchers [33–36] have used a different neural network with a similar CNN architecture. Few researchers have also used a unique method of detecting deepfake video. Two researchers [34, 35] have used a DNN based ensemble method to detect deepfake videos. An ensemble method is a technique used to combine two or more deepfake detection system algorithms. The outcomes of all the algorithms are compared before classifying if a video is a deepfake. As shown in Table 3, the ensemble method-based detection systems perform better than other CNN algorithm-based detection systems. A. Mitra et al. [38] have outperformed all deep neural network-based detection systems using the ensemble method to achieve an accuracy of 99.65%. However, researchers have also stated that the ensemble-based detection system requires high computations hardware to run the algorithms [34, 35].

According to [14], all the researchers using CNN algorithm-bassed deepfake detection system have used pre-trained models to detect deepfake videos. A pre-trained model is a model created by someone else to solve a similar problem. The CNN-based models like VGG-16 have been trained on a problem to classify cats and dog images. The researchers have used a transfer learning method to re-train the CNN models and detect deepfake videos. The research states that most researchers have used this method to build a detection system because it uses low computational resources, trains the models with fewer data, and produces a well-performing system. The high performing models in Table 2.3 have used CNN pre-trained models to build a deepfake detection system. Therefore, most of the systems have achieved high accuracy compared to the other deepfake detection methods.

Nevertheless, the deep neural network-based deepfake detection systems have various drawbacks, which researchers mention. The drawbacks from each of the systems can be generalised as follows:

- When a video frame is tilted in different angles, the DNN based detection systems find it challenging to classify the video as deepfake or original. Since most detection systems have been trained to detect videos only at one camera angle, the detection system will produce a lower accuracy when tested with different camera angles.

- Few deep learning algorithms, such as Capsule-CNN [32]and FCN [36], would require more time and computational resources to classify videos.
- The deep learning models have been trained with public datasets that contain outdated deepfake videos or low-quality videos. Therefore, these systems would find it challenging to detect high-quality deepfake videos.
- According to [39], pre-trained researchers cannot modify their models to detect the latest version of deepfake videos.

2.2.4 Detection of Deepfake for Social Media

Over the years, social media platforms have been improving their features and functionalities to attract more users. However, as social media applications improve, the concern for creating, spreading, and consuming disinformation and fabricated content increases. According to [40] and [41], the deepfake technology like face replacement of face re-enactment is the primary concern for all the social media companies, as users use these deepfake methods to create fabricated contents that are realistic. Researchers have stated that the deepfake videos uploaded on social media are harder to be detected by the existing detection systems.

Sue et al. [41] have stated that all the existing deepfake detectors are built on an assumption made of the GAN (Generative Adversarial Network) model used to generate deepfake videos. The detectors built by researchers are based on the publicly available deepfake generation algorithms that are currently outdated. The research vividly states that the attackers creating fabricated content to target an individual are more careful, and the algorithms used to generate deepfake videos are possibly hidden. The possibility that the attacker is using an updated version of deepfake software is high.

According to [42, 43] and [40] there is clear evidence that the existing deepfake detection systems find it challenging to detect deepfake videos that are generated by the latest versions of deepfake software. The reasons why the existing systems have outdated can be generalised in a few points as follow:

- The detection systems are trained using various public datasets containing deepfake videos generated from older versions of deepfake software.

- Forensic tool and DNN (Deep Neural Network) based detection systems are trained to find flaws in the standard GAN(Generative Adversarial Network) models used to generate deepfake videos.
- DNN (Deep Neural Network) based detection systems built using pre-trained models cannot be modified to extract any flaws found in the updated versions of deepfake software.

Several recent studies [40–42] have stated that the publicly available dataset played a major role in training the existing detection systems. They proved that the publicly available dataset lacks various visual qualities compared to the deepfake videos circulated on social media platforms. A few reasons why the public dataset and videos on social media differ are as follows:

- The research has stated the social media functionalities such as applying the filter to images and videos before the users can post them on the platforms creates an extra layer of visual quality that makes the deepfake videos flawless.
- The social media applications perform heavy compression and downsize the videos before they are uploaded to the platform to save network bandwidth and protect users' privacy. Researchers have stated that these functions have impacted the visual qualities by improving the resolution or changing various features of the videos. These changes make deepfake videos more flawless when uploaded on a social media platform.

Since various factors have impacted the existing deepfake detection system, few researchers have proposed solutions to overcome these issues and create a detection system suited for social media. Das et al. [44] have published research that analyses how deepfake detection systems can be improved using dynamic face augmentation. Researchers have used a face-cutout method, which cuts out the region of images using facial landmark information. This method applies filters and changes the pixels of the facial regions of the video frames in the dataset—the DNN models further train this dataset by comparing each facial area of the augmented deepfake video frame to the original video frame. The researchers also find similarities between the augmented deepfake videos and deepfake videos found on social media.

Augmenting the deepfake videos do provide a reasonable solution to improve the detection systems to detect videos on social media platforms. However, studies show few researchers have designed unique systems to detect deepfake systems on social media platforms [38, 45, 46]. Mitra et al. [38] have developed an algorithm to detect deepfake videos on social media with a low computational requirement. The model is a keyframe extraction CNN model, which means that the video frames are extracted and focused on the facial area before attempting to detect deepfake video. The research emphasises the importance of converting video to frames and using the frames to train DNN models. The study provides evidence that the models trained with video frames perform better than those trained with videos. However, in this research, the system is built using two deep neural networks, InceptionV3 and Resnet50, which have the same architecture as the CNN model. The detection system has achieved an accuracy of 98.5% on the FaceForensic++ dataset and 92.33% on other public datasets for deepfake detection. However, the research does not explain how this method can be implemented on existing social media applications [46].

Y. Li et al. [45] have developed a similar model, but the research clearly states how social media platforms can implement deepfake detection systems in their application. The authors have created a website where users can upload videos to detect if they are deepfake or not. The web application consists of a back-end system that finds the authentic author of the uploaded video using google API. Additionally, the back end of the application performs deepfake detection using a low computational CNN algorithm. The author also states that this web application can be embedded in social media using the API feature. However, the researcher proposed solution seems to be performing poorly, as it achieved an accuracy of 46%. The authors concluded that the detection system performs badly due to the low computational resources used to run the detection system on a server.

An article published by C. Leibowicz et al. [46] provided insight and inspiration to other researchers on the importance of developing detection systems that prevent the spread of false information on social media. The article emphasises the importance of developing a new approach to detect deepfake videos on social media platforms. The researchers state the flaws in the deepfake videos in two different methods: the facial region in the video and the second is detecting the voice of the video. However, the article concludes

by stating that minimal research is published on how these detection systems can be used in the existing social media platforms.

Most prominent tech companies like Google, Facebook, and I.B.M. are focused on finding an apt solution to prevent the spread of false information via deepfake videos on social media platforms [47]. One of the steps taken by Facebook and Google is to update their community guidelines to prevent the spread of deepfake videos [3]. They also host a public competition for researchers and experts every year to solve the deepfake challenge.

Based on analysing the existing deepfake detection system built for social media platforms, we can find a lack of models built specifically for social media. However, specific observations can be made from these studies that can help build a system suitable to detect deepfake video on social media. The observations are as follows:

- As mentioned in [38], using video frames to train DNN models would provide a more efficient system than training with videos. Therefore, converting the videos in the dataset before training the model is essential to build a well-performing detection system.
- Training the system with the publicly available dataset is not sufficient. As mentioned in [44], training models with augmented video frames by changing the pixels at the facial region would train a model to detect deepfake videos on social media.

We will use these observations to build the most suitable system to detect deepfake videos on social media in our proposed system.

Overall, the deep neural network-based deepfake detection systems have the best overall performance compared to all the existing detection systems. However, there has been a lack of research and proposed approaches to detect deepfake videos on social media platforms. Therefore, in this research, our main objective is to collect all the requirements of various social media platforms and design a system that will help the social media applications to detect deepfake videos and prevent the spread of disinformation. In this researcher, we will also investigate how an ensemble method consisting of three different CNN algorithm-based models will perform compared to a single CNN algorithm model. While designing the system, we will also focus on the existing deep neural network issues, as stated in the previous subsection 2.2.3.

Chapter 3

Problem Statement

Deepfake detection systems built using the DNN (Deep Neural Network) algorithms have outperformed all the other methods of detection systems. As mentioned previously, we will be using the DNN method to build our detection system to detect deepfake such as replacement and re-enactment. However, various surveys have shown that the existing deepfake detection systems are not suitable for detecting deepfake on social media. A few of the significant issues of the existing detection system that we will be focusing on are as follows:

- The existing systems do not focus on building a detection system for social media platforms. There has been a lack of research on the working of detection systems on social media applications.
- Most of the DNN based detection systems use a pre-trained model to detect deepfake. The pre-trained model does not allow the researchers to modify the DNN algorithms. This issue makes the existing systems outdated as the deepfake technology has been improving drastically.
- The public dataset used by most of the existing systems consists of outdated videos from the older versions of deepfake software. Detecting the latest versions of deepfake videos would be challenging for the existing systems.

Apart from these challenges, the improvement in social media applications allows the users to manipulate deepfake videos by adding extra layers of changes such as adding a filter or changing the brightness and contrast of the videos.

Developments in social media applications and deepfake technology are proportionally increasing. These developments have motivated us to propose a new approach to detect deepfake videos explicitly for social media platforms. Another motivation for this research was the challenges and issues with the existing system to detect deepfake videos on social media platforms.

Using these motivations, we have formulated a more specific research question, which is:

- **How can existing social media platforms effectively detect malicious deepfake videos and prevent them from spreading false information?**

In this research, we will be focusing on answering our research question by designing and building a new approach to detect deepfake videos.

Chapter 4

Proposed Approach

4.1 Overview

Detecting deep fake videos on social media is essential to prevent the spread of false information. We propose a new approach to detect deepfake videos using DNN algorithms to answer our research question. Our primary focus should be to detect deepfake videos on social media platforms based on our research question.

As seen in [6], the research states that the best method to control the spread of misinformation is early detection. Giving early alerts to the users of the disinformation is extremely important to minimise the damage to an individual or an organisation. Therefore, we have designed our deepfake detection system based on the early detection mechanism. Figure 4.1 is a flowchart showing the implementation of deepfake detection in a social media application.

The implementation of deepfake detection in social media applications is very tricky. The community guidelines and privacy rules restrict the platforms from accessing all the content uploaded on social media. Therefore the best way to detect deepfake videos is when the content violates the platforms guidelines. Therefore, as shown in figure 4.1, the reported contents are verified from our detection system. As mentioned previously, not all deepfake videos are harmful, so we need a human to review the maliciousness of the contents. A human review of the content is necessary to classify if the deepfake content is harmful or not. So all the contents after detection must be reviewed by a human. However, the importance of this implementation system is that warning of

deepfake videos after verification from the detector can be a way to minimise the damage during the spread of disinformation. The red flag, shown in figure 4.1, can initiate a warning on the content detected as deepfake.

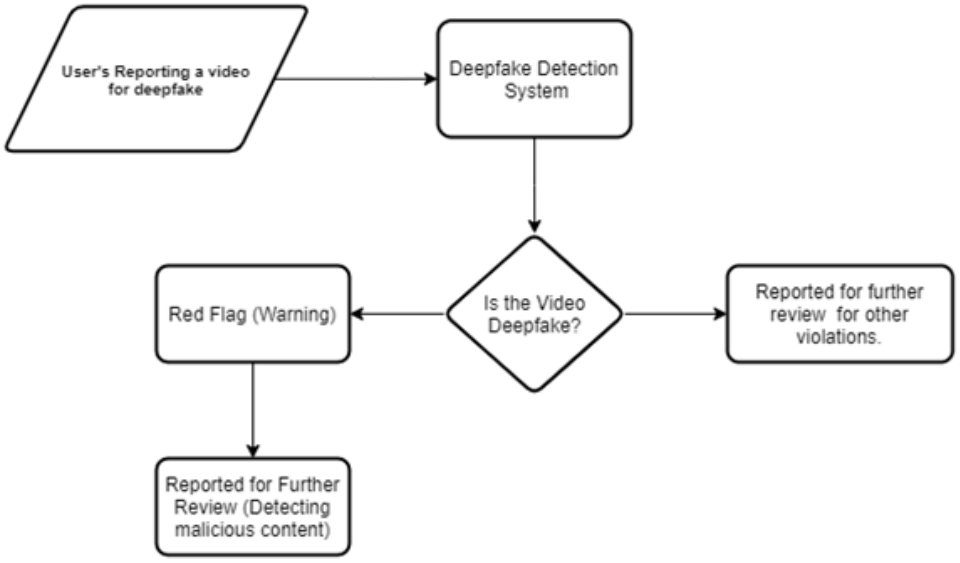

FIGURE 4.1: Overview for social media platforms to manage deepfake content.

Now that we know where our deepfake detection system can be implemented in the social media application, we can design a detector limiting the computational resources. We have designed our model by splitting the process into three major stages. As shown in figure 8, the overall processing of the project has three steps: dataset collection, data pre-processing and training DNN models using the pre-processed data. We are using a standard method to train a deep neural network algorithm, as shown in Figure 4.2.

FIGURE 4.2: Processing Steps to Build our detection system.

This section will propose a new approach for detecting deepfake videos for social media platforms. Existing video-based social media applications such as Facebook, Instagram, TikTok, YouTube, and Twitter requirements are analysed to design and build the proposed solution. The dataset and deep learning models used in the proposed solution are

selected based on the existing deepfake detection systems and the selected social media requirements.

4.2 Dataset

When building a supervised model, selecting a dataset plays a vital role in training and developing a prediction system. Understanding the research question mentioned in chapter 3, the main task of this research is to build a deepfake detection system specifically for social media platforms. Therefore, we have created a set of requirements before collecting the dataset. We created the requirements by observing the functions and features of various social media platforms. The requirements are as follow:

- Videos uploaded on social media generally have a minimum resolution of 32 by 32 pixels and a maximum resolution of 1280 by 1024 pixels. These resolutions are considered for both modes of videos which are landscape and vertical modes. The survey [14] shows that the latest deepfake videos have a resolution ranging from 80 by 80 to 1024 by 720 pixels. Hence, I am considering developing a detection system to detect deepfake videos from 32 by 32 to1280 by 1024 pixels.

- The maximum video file size uploaded on Twitter is 512 MB, which can be considered the minimum video file size one can upload on a social media platform. The maximum video file size users can upload on social media platforms is 4GB. The deep learning model should detect deepfake videos from a minimum file size of 512MB to a maximum of 4 GB.

- The maximum length of a video which can be uploaded on social media platforms can vary accordingly. A survey conducted [47] states that users can upload a maximum of 240 minutes of video on a few platforms like YouTube and Facebook. Nevertheless, considering the surveys [9, 14] conducted by researchers, deepfake applications can generate videos up to a maximum length of 2 minutes videos. Hence, we would like to set the requirement of detecting deepfake videos of a maximum length of 3 minutes long. This requirement is made based on the assumption that most social media platforms like Instagram, LinkedIn, and Twitter allow users to upload short-length videos.

These requirements are tailored based on the selected social media applications (Facebook, Instagram, LinkedIn, YouTube, and Twitter). Since these applications have many users, it would be appropriate to build the requirements based on these applications. Furthermore, the deepfake videos and original videos collected and used in this research are selected based on the requirements mentioned above.

The artificial neural network models used in deepfake detection systems are dependent on qualitative data, which is used to train the algorithm and detect the latest deepfake videos. Based on the requirements mentioned, we have selected a few public deepfake datasets, as shown in Table 4.1, generated from the latest versions of the deepfake application.

Dataset	**Updated Year**	**No. of original videos**
Celeb-DF (v2) [48]	2021	590
FaceForensics++ [49]	2021	363
Deepfake Detection Challenge [50]	2020	77

TABLE 4.1: Public deepfake dataset

4.2.1 Celeb-DF Dataset

Yuezun et al. [48] have collected a large scale deepfake and original video dataset and made it open source to help researchers and enthusiasts. Researchers' primary purpose was to publish a high-quality deepfake videos dataset that can help researchers and enthusiasts evaluate and train their deepfake algorithms to produce a well-performing system. The authors vividly mention that over the past five years (From 2015 till 2020), the quality of deepfake videos generated from the deepfake software has been producing a high-quality video. The deepfake generation software's are being updated based on the flaws found by researchers over the years, which means that the detection systems are becoming obsolescent.

The Celeb-DF dataset consists of 590 original videos, in which 59 different celebrities from diverse genders, ages and ethnic groups are selected carefully. These videos of celebrities are originated from various clips on YouTube, which the researcher collects.

The original videos are used to generate 5639 deepfake videos, which are synthesised further to make them impossible to detect by any humans. The original videos are

converted using the popular open-source deepfake generation algorithm. However, the facial features in each video are swapped among other videos of different celebrities. Furthermore, Yuezun et al. [48] have used a synthesised algorithm to make the visual quality of the deepfake and original videos high quality and more challenging to be detected. The research states that the algorithm was created based on other researchers' flaws in older versions of deepfake videos.

Analysing the original and deepfake videos, the main features of the videos are as follows:

- The average length of the video is 13 seconds, with a standard frame rate of 30 frames per second.
- The resolution of the videos is range from 64x64 pixels to 256x256 pixels.
- The file size of the videos ranges from 2 MB to 15 MB

The features of the celeb-df dataset videos make it perfectly suitable for our detection system as it falls within our requirements. Another advantage of using this dataset for training is that the deepfake detection system would effortlessly detect deepfake videos of celebrities. However, the only disadvantage of using this dataset for training is that it consists of lower resolution deepfake videos.

4.2.2 FaceForensics Dataset

Andreas et al. [49] have generated deepfake videos after examining the realism of state-of-the-art deepfake videos. The researchers primarily group deepfake videos into facial re-enactment, facial replacement, and Neural textures before evaluating the deepfake videos with various open-source deepfake software. However, the authors have collaborated with Google to create realistic videos that are based on specific scenarios. Over the past years, Google has worked with paid and consenting actors to create 363 original videos. The intention of creating such a dataset is to generate deepfake videos with higher resolution and realistic videos. A total of 3068 deepfake videos are generated using the two most popular open-source deepfakes [49]. Additionally, the video quality in the dataset is further improvised, producing different ranges from low to high resolution. The features of the videos in the dataset are as follows:

- The resolutions of the videos are 480 pixels, 720 pixels and 1080 pixels. The high-quality video from the dataset is made available to download using H.264 codec video coder. The coder can enhance the videos to 8k resolution, which most social media platforms use.
- The average length of the video is 1 minute.
- The file size ranges from 30 MB to 200 MB

Since the videos consist of higher resolution videos and follow our requirements, this dataset is suited for training the deepfake detection system. Moreover, using this dataset for training the system would be advantageous as it improves our system from detecting any higher resolution deepfake videos on social media platforms.

4.2.3 Deepfake Detection Challenge Dataset

The deepfake detection challenge [50] was a public event that was posted on the Kaggle website in 2019. Multinational companies such as Amazon, Facebook and Microsoft collaborated to build this challenge for enthusiasts and researchers worldwide. The main purpose of this challenge was to find innovative ideas to detect deepfake videos or voice manipulation.

However, the dataset published by the hosts consists of 77 original videos of various paid actors. The original video is created by different paid actors based on scenarios such as a single person in one frame and two or more people in one frame. The original videos were created to produce a range of high to low-quality deepfake videos.

The host created the Deepfake detection challenge dataset for a competition. Therefore we are made unaware of how the deepfake videos are generated. However, the feature of the videos in the dataset are as follows:

- The resolution of the videos range from 64x64 pixel to 1080 pixels
- The average length of the video is 20 seconds, with a standard frame rate of 35 frames per second.
- The size of the video ranges from 1MB to 6MB.

The dataset has a wide range of features, but has significantly fewer videos compared to the other datasets. Therefore, it would be most suitable to evaluate our deepfake detection system rather than train our system with the dataset.

Overall, the public datasets collected satisfy the dataset requirements mentioned before. Although celeb-df has a lower visual quality feature than the FaceForensic++ dataset, it would be better to use the dataset to train the deepfake detection system in 2 different phases. In addition, training the deepfake detection system for low and high resolution would increase the performance of the detection system in predicting deepfake videos on social media platforms.

Since the deepfake detection challenge dataset is a small size dataset and consist of a wide range of features, it is best suited for evaluating the deepfake detection system. Apart from the deepfake detection challenge dataset for evaluation, we will be collecting videos (deepfake and non-deepfake videos) from various social media platforms (such as Facebook, Instagram, TikTok, YouTube and Twitter) to evaluate our detection system further.

4.3 Data Pre-Processing

4.3.1 Converting Video to Frames

Video is a collection of images that are arranged in a specific order. These images of the video can be referred to as frames. Deep neural network algorithms can be trained as either video or image datasets. Most researchers have trained their deepfake detection systems by converting the videos into multiple frames or images. However, only a few researchers prefer to train their deepfake detection system with videos as input. As mentioned previously, using an image dataset to train our detection system is preferable to using video datasets due to reasons such as:

- Using an image or video dataset to train our detection system has similar steps, and both the dataset helps the system classify a deepfake video.
- However, using a video-based dataset is an extra step that deep neural network algorithms must do before training. The extra step is that the algorithm converts videos to frames automatically.

- Training DNN models with video-based datasets would require more processing time to train than using an image-based dataset. As our training dataset size is enormous, the detection system would take more space and time for training. Therefore, to save time and computational resources, using an image-based dataset is the most preferable.

- When a video-based dataset is used to train DNN they automatically convert the video into frames, which might cause problems. The automatically converted frames that might not contain any facial region would be invalid to classify using the deepfake detection system. In a deepfake detection system, the primary focus is to classify the facial areas in the video. Therefore, the video classification would produce a faulty performance compared to image classification.

Additionally, another advantage of using image-based dataset is that our system can be trained more efficiently by using more time and computational resources. Understating that the deepfake detection system should mainly focus on the facial regions in the video, it would be a better approach to train the system with frames of videos that consists of only the facial areas. Thus, in this research project, we would be training our detection system with only frames that consist of facial regions. Before converting the videos into frames, we have used a face detector to detect faces in each video. Figure 4.3 gives an overview of how the videos will be converted into frames.

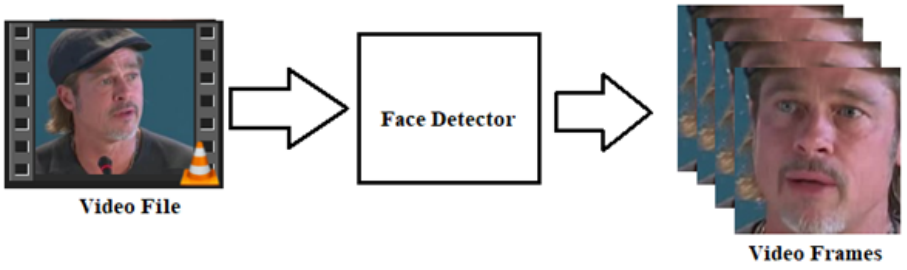

FIGURE 4.3: Converting Videos into Frames

Building a face detector from scratch would consume a lot of time. Therefore, we have used a Python library to detect faces in each video for this research project. The python library used is called dlib, an open-source toolkit that uses machine learning algorithms to locate the facial points on a person's face in a frame. Using dlib toolkit has various advantages than disadvantages. A few significant benefits are:

- This toolkit can detect faces at almost all angles of the frames of each video.

- Since dlib is built using C++ programming language, it is compatible with most of the currently used hardware.
- Dlib uses significantly less memory space and time for detecting faces, and the average time taken to detect a face in an image is 0.2 seconds.

Figure 4.4 is a code snippet that shows how dlib library is implemented in the project. A person's face consists of several features, such as eyes, mouth, nose, and eyebrows. The get_frontal_face_detector function of the dib library maps the points surrounding each face feature and returns the facial information to a variable.

```
import dlib
detector = dlib.get_frontal_face_detector()
```

FIGURE 4.4: Dlib Python Library

Figure 4.5 is the main code snippet, showing how each frame from the video is extracted after detecting the facial regions and then storing the data accordingly. The video's frames are extracted before the dlib libraby detects if its facial values are present in it. If a facial value is detected, then the face region from the frame is cropped and stored.

```
cap = cv2.VideoCapture(os.path.join(real_vid, vid))
frameRate = cap.get(3)
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]
```

FIGURE 4.5: Main Code Snippet to convert video to frames

The celeb_DF and FaceForensic datasets' videos are converted into frames and stored in two different groups, original and deepfake. Only the frames that consist of facial regions are stored in two different folders, original and deepfake folders, respectively. Generally, the face detector helps the detection system increase its performance by eliminating the invalid video frames from the training data.

As mentioned before that the celeb_DF, FaceForensic and deepfake detection challenge dataset has a frame rate of 30 frames per second. Table 4.2 demonstrates the total number of frames extracted from all the original and deepfake videos, respectively. These

Dataset	Total Original video Frames	Total Deepfake video Frames	Used For
Celeb_DF	584	5627	Training
FaceForensic++	234	2818	Training
Deepfake detection Challenge	75	300	Testing

TABLE 4.2: Total number of frames converted from the public dataset.

extracted frames of the dataset will be further pre-processed and used for training our deepfake detection system in this project.

4.3.2 Data Augmentation

In machine learning, data augmentation is a technique used to alter features of an image or a video in a dataset, generating different versions of datasets to overcome overfitting in machine learning models and solve various other problems. However, as mentioned in the problem statement, most social media platforms have built-in filters or data augmentation techniques that help users manipulate images or videos. Most of the previous researchers have failed to consider the effects of these data augmentation techniques (also known as social media filters).

Nevertheless, L. Bondi et al. [51] have suggested that different data augmentation techniques affect the convolutional neural network (CNN) based deepfake detectors performance. The researcher's experiment was conducted on detectors trained on the public dataset, including Celeb-df and FaceForencis datasets. This experiment was focused on analysing data augmentation techniques such as:

- HF: Horizontal flip of the video frames.
- BC: Brightness and contrast changes.
- HSV: Hue, Saturation, and Value changes.
- ISO: Addition of ISO noise.
- GAUS: Addition of Gaussian noise.
- DS: Downscaling with a factor between 0.7 and 0.9.
- JPEG: JPEG compression with a random quality factor between 50 and 99.

Overall, as mentioned in [51], these techniques significantly affect the CNN-based deepfake detectors. Few of the data augmentation such as DS and JPEG does not affect the performance of public datasets like FaceForensic. Few techniques do not affect specific public datasets because of the paid actors' scenarios to create the original videos. In addition, the research was concluded by stating that the data augmentation is valid only when it is selected carefully considering the dataset used for training and its features.

However, based on the experiments conducted, few data augmentation have huge effects on celeb_df, and FaceForensic datasets. They are as follows:

- HF: Horizontal flip of the video frames.
- BC: Brightness and contrast changes
- HSV: Hue, Saturation, and Value changes.
- ISO: Addition of ISO noise.
- GAUS: Addition of Gaussian noise.

These data augmentation techniques are also considered the essential filters used in popular social media platforms such as Facebook, Instagram, TikTok and Twitter. Therefore, we will generate different versions of Celeb-DF and FaceForensics datasets using these data augmentation techniques in this project.

Most deep neural network algorithms have built-in functions to augment the frames. However, these built-in function does not include all the selected data augmentation techniques. Therefore, in this project, we have augmented each video frame of the training data set using a python package called Albumentations.

Figure 4.6 is a code snippet demonstrating the implementation of frame augmentation in the project. Using the Albumentations package, we have created a pipeline of data augmentation techniques. We have given a probability of 50% for each augmentation technique, which means that each technique will only be applied to 50% of the frames /images in the datasets.

```
# import albumentations package
import albumentations as A
from albumentations.pytorch.transforms import img_to_tensor
# A complex augmentation pipeline
augmentation_pipeline = A.Compose(
    [
        A.HorizontalFlip(p = 0.5), # horizontal flip to 50% of images
        A.OneOf(
            [
                # apply one of transforms to 50% of images
                A.RandomGamma(),
                A.RandomBrightnessContrast(),
            ],
            p = 0.5
        ),
        A.HueSaturationValue(p=0.5),
        A.ISONoise(p=0.5),
        A.GaussNoise(p=0.5),
    ],
    p = 1
)
```

FIGURE 4.6: Main Code Snippet to perform image augmentation.

Figure 4.7 shows the output of the above code using different data augmentation techniques on an image. Since there are a lot of filter combinations on multiple social media platforms, we have used random values for HSV, ISONoise, GAUS, and BC. Figure 13 vividly shows the randomized features of the images and all the possibilities of the data augmentation pipeline.

FIGURE 4.7: The result of the augmentation code.

Using the data augmentation techniques for the video frames has two main advantages, which are as follows:

- As shown in table 4.3, the total number of video frames for both groups are imbalanced. Therefore, we have balanced the total number of frames for original and deepfake videos using the data augmentation method. A balanced number of frames would help us train our model effectively.
- By training our model with augmented images, we have a high possibility of detecting any deepfake videos on multiple social media platforms. Users uploading deepfake videos on social media augment the videos by using filters or changing the brightness. The pre-processing augmentation method would make it easier for our system to detect augmented deepfake videos.

After applying the data augmentation techniques to the training dataset, table 6 shows the total number of original and deepfake video frames for both the training dataset. This dataset will be further used in training our model.

Dataset	Total Original video Frames / Images	Total Deepfake video Frames / Images
Celeb_DF	11,253	11,253
FaceForensic++	6084	6084

TABLE 4.3: Total Number of Frames in our training dataset after augmentation method.

Before building our deepfake detection system, we have one last step of pre-processing the images. Since machine learning or deep learning models cannot read images directly, we need to convert the images into an array of numbers. In this project, we have converted all the original videos frames and deepfake video frames into an array of numbers, as shown in the image Figure 4.8.

The code snippet shown in Figure 4.8 splits the dataset in training and validation, which is 80% of the data for training and 20% for validation. As mentioned before, Celeb_DF and FaceForensics datasets have vast features amongst each other. Therefore, we will be training our model with each dataset separately. Hence, we must split the dataset into training and validation. Furthermore, understanding that the dataset is split into X and Y values, where X consists of the array of images in float values and Y is the label that states if the values are deepfake or original [0, 1]. The X and Y values for both datasets will be converted separately before feeding them to our models for training.

```
batch_size = 32
img_height = 224
img_width = 224
datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=False,vertical_flip=False, validation_split=0.2)

def train_dp(data_dir):
    train_gen = datagen.flow_from_directory(directory=data_dir, target_size=(224,224),
                                            class_mode='binary',batch_size=32,subset='training')
    return train_gen

def val_dp(data_dir):
    val_gen = datagen.flow_from_directory(directory=data_dir, target_size=(224,224),
                                          class_mode='binary',batch_size=32,subset='validation')
    return val_gen
```

FIGURE 4.8: A code snippet: Helps to cover images to an array of values and split the dataset into training and validation.

4.4 Deep Learning Models

4.4.1 Model Selection

Model selection plays a vital role in the experiment, as we need to consider a model which would be suitable for detecting deepfake videos on social media platforms. However,

a survey conducted by a few researchers [7, 9, 14] states that a convolution neural network(CNN) is the most efficient deep neural network model used in a detection system.

A convolutional neural network (CNN) is a neural network model that extracts information from an image and represents it in numerical format. The CNN model extract features of an image automatically using neural network layers such as hidden layers. Then the images are classified by the model based on its features. Researchers have found similarities between how a human brain works, and both works to distinguish the essential features of an image and then classify it. For instance, when humans look at a cat picture, we tend to classify it immediately by looking at its features. However, a CNN model also similarly classifies the cat image by understanding its features mathematically. The mathematical representation of an image includes its filters as a dot product pixel value. These values are scrutinized before developing a visual context of the images to classify.

Figure 4.9 shows an architecture of a fully connected CNN. It consists of various layers, such as convolution and pooling. The convolution layers extract the features from an image based on its filters such as brightness, HSV (Hue, Saturation, Values), IOS noise and other features. Whereas the pooling layer replaces the output with a maximum value summary of the data, allowing the final layer to classify the image easier. Since CNN can automatically extract features from an image and classify it, this model makes it suitable for classifying deepfake videos.

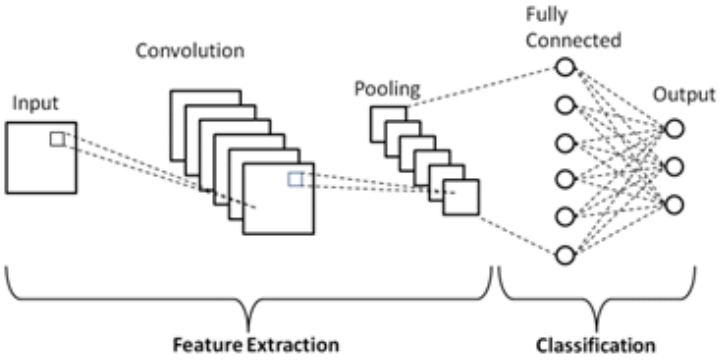

FIGURE 4.9: CNN Architecture

The CNN is considered a basic neural network algorithm for image classification. However, over the years, CNN has been further improved by researchers and other experts. These improvements in CNN have allowed the researchers to create various versions of CNN based models such as Le-Net (Year-1998), Alex Net (Year-2012), VGGNet (Year-2014), Inception module Google Net (Year-2014), and ResNet (Year-2015). Researchers and other experts select the CNN-based models based on the problems they will be used to solve. For instance, Alex Net and Le-Net is a CNN-based model primarily used for mobile applications such as the Financial Banking application.

Previous researchers have developed deepfake detectors considering these variations in the CNN model. An article published [33] gives a vivid idea of how these CNN based algorithms impact the detection of deepfake. Based on this survey conducted, we have made a few generalized analyses of CNN based models:

- Few surveys conducted compared all the CNN based algorithms performance when used for deepfake detection systems. The survey results state that the three of the CNN-based algorithms outperformed most of the deepfake detection systems. The deep neural network models are VGG-16, InceptionNet (GoogleNet) and EfficientNet models. When the deepfake detection system was trained with pre-trained versions of these models, they achieved an accuracy ranging from 98% to 99%.

- The article published also researched the algorithms used in deepfake detection challenges hosted by multinational companies. The research suggests that the EfficientNet model, a CNN-based neural network algorithm, is suitable for deepfake detection as it outperformed all the deep neural network algorithms. The author claims that the participants who won the challenges have mainly used the EfficientNet algorithm to classify deepfake videos. One of the challenge winners achieved an accuracy of 99% using the EfficientNet model.

- The research also mentioned that CNN based models like VGG-16 and InceptionNet consist of layers whose primary focus is to extract the facial features from an image. VGG-16 and InceptionNet are deep neural network models which are

mainly used for face recognition and face detection applications. However, researchers have further developed the InceptionNet model, creating different versions of the model over the years. InceptionNet version 3 has managed to outperform all the CNN based algorithms in multiple applications. This achievement is due to its capability to extract essential features from an image.

In this experiment, we will combine various CNN-based deep neural network models forming a detection system, also known as an ensemble model. Inspired by the state-of-the-art, we have selected the most efficient CNN-based models. They are as follows:

1. VGG-16
2. InceptionNet-v3
3. EffcientNetB0

These models being an advanced version of CNN architecture, we believe that the ensemble of these methods would produce an effective way to detect high-quality deepfake videos on social media platforms. However, this research will evaluate our ensemble method and other existing ensemble detection systems, giving proof of concept.

To summarize the model selection, figure 4.10 process of training for our proposed system. As seen in figure 4.10, the CNN-based model's training will be performed in two different phases. In phase 2 of training, the models from phase 1 will be further trained with a different dataset to improve its overall performance. To conclude, the models will be trained two times with different datasets each time, preventing the model from underfitting. The algorithms created a stronger relationship between the video's frames of the dataset (input layer) and predictions made (output layer).

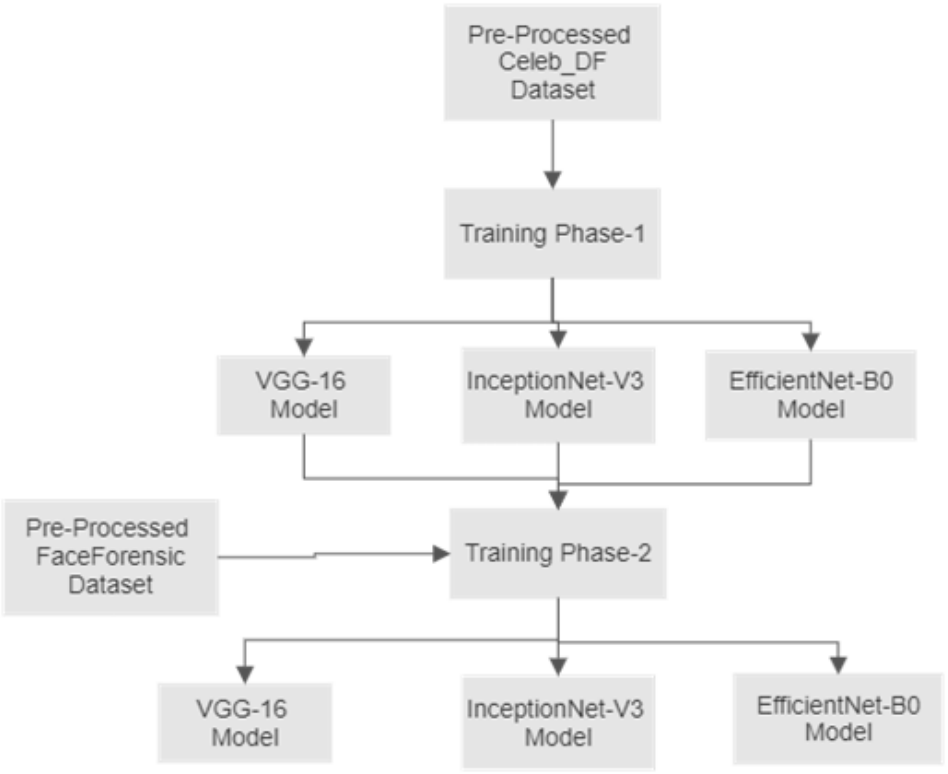

FIGURE 4.10: Training process of the proposed system.

4.4.2 VGG-16

VGG-16 is a convolutional neural network model which was proposed particularly for large scale image recognition. Initially, the model was trained with 14 million images belonging to 1000 different classes, and the model achieved an accuracy of 93%, which outperformed all the other deep convolutional networks during the year 2011. Figure 4.11 shows the architecture of VGG 16, which is a combination of convolutional neural networks (CNN). VGG-16 model consists of convolutional layers, pooling layers, and fully connected layers to predict or group an image accordingly.

As the model's name suggests, VGG-16 has 16 layers of weights, a huge network with about 138 million parameters (also known as image features). The input of VGG-16 has a fixed input size of 224 X 224 RGB image. Hence, the model works at its best when the input image from the dataset has a size of 224 X 224 (height X width) and is an RGB image. The input images are further sent to the convolutional layers, which filter the features of each image before sending it to the max-pooling layer. The max-pooling layer is used to calculate the maximum or most significant values in each batch of the

convolutional layers. The Max-pooling layer in VGG-16 maps the important images' features, outperforming other pooling methods such as the average pooling used in the image classification method. Further in the architecture, the fully connected layer is the activation function such as SoftMax or Sigmoid, which helps the model transform the input weights into an output layer used in prediction. Overall, VGG-16 is a neural network that can work for image classification tasks and provide good performance.

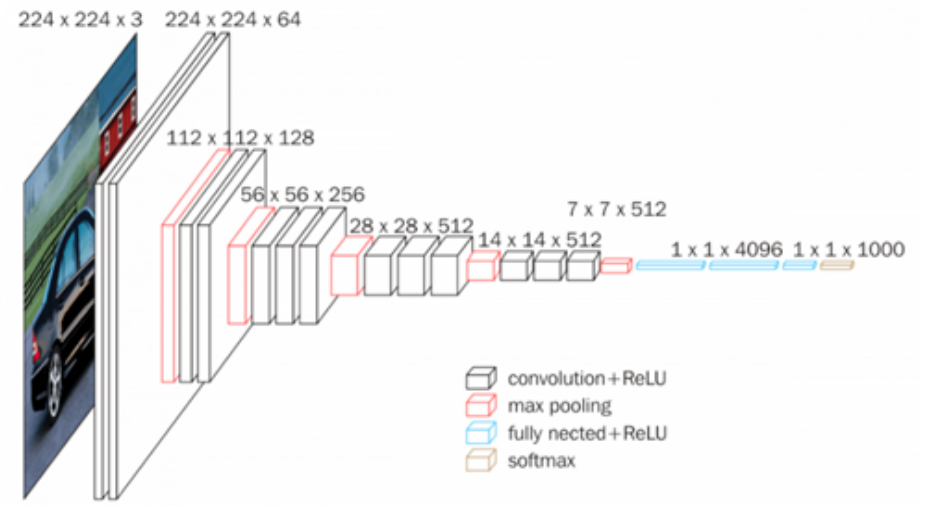

FIGURE 4.11: VGG-16 model architecture.

Most of the existing deepfake detection systems created by previous researchers and experts have used a pre-trained VGG 16 model and transfer learning method to predict deepfake videos. A pre-trained model is a previously trained model with a large benchmark dataset to solve an initial problem of classifying cats and dog images. These pre-trained models have weights stored from previous training, and they can be used in the form of transfer learning techniques to detect deepfake videos. Using this method of building a detection system can give high accuracy, but various limitations can cause a negative transfer. The negative transfer in context detecting deepfake videos means that the high-quality deepfake videos generated by deepfake software can be harder to detect over the years. Therefore, in the research project, we will build a VGG-16 model and the other models from scratch and train in such a way that it will detect any deepfake videos ranging from high to low resolutions.

Since we are building the VGG-16 from scratch, we have modified the model accordingly. Figure 4.12 shows the VGG-16 architecture, which we will be using in training and detecting deepfake videos. The VGG-16 architecture used in our project is similar to

that of the VGG-16 models shown in figure 4.11. However, LeakyReLU, Flatten, and Dropout are the extra layers added to our model. The reason for adding LeakyReLU, Flatten, and Dropout layers to the model are to prevent the model from overfitting. These layers remove any noise or random fluctuations in the training data which can occur.

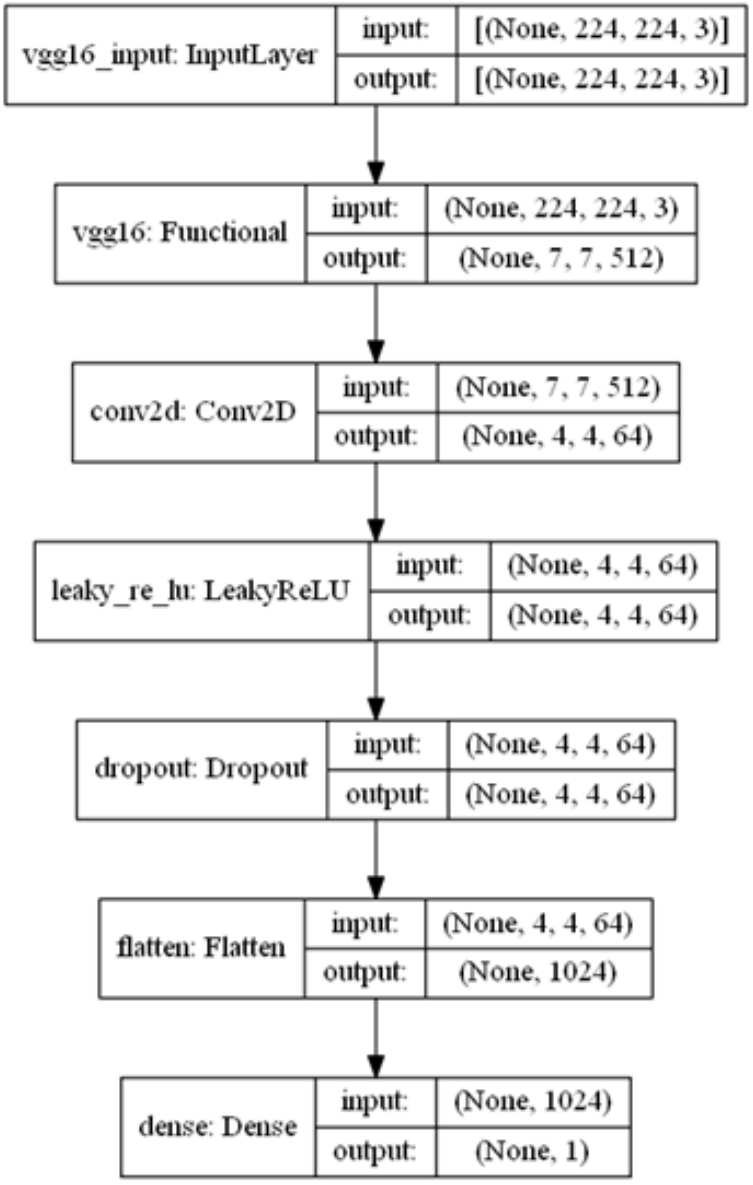

FIGURE 4.12: Modified VGG-16 model architecture.

The dense layer is the final layer of the VGG-16 model, as shown in figure 4.12. This layer is modified from the standard VGG-16 architecture (Figure 4.11). As we know that the deepfake detection is a binary classification problem, which means that a video is fake if its value is 1 and it is real if it is 0. This final layer helps us predict the deepfake videos by transforming a single image in the dataset to a single value, either 0 or 1. The

best activation function for a binary classification problem is sigmoid, which maps any input to an output ranging from 0 to 1. For small values (<-5), sigmoid returns 0, and for larger values (>5), the result of the function gets close to 1.

The code snippet in figure 4.13 is a compilation of the VGG model with a loss function. We have used the binary cross-entropy function, which is a loss function used to compute the cross-entropy loss between the true label and the predicted label. As we have only two labels (0 or 1), binary cross entropy would help us evaluate the VGG-16 model with the validation dataset by comparing the loss values computed in predicted and actual values.

The second most important thing in compiling the model is the optimizer, as shown in figure 4.13. Optimisers are methods used generally to change the attributes of a neural network, such as weights and learning rates, to reduce the losses. For our VGG-16 model, we have used SDG (Stochastic Gradient Descent) optimizer with a learning rate of 0.001. The learning rate controls changes such as the estimated error change in the model each time the weights are updated. Sometimes choosing the learning rate is very challenging; therefore, we selected 0.001 as the learning rate after testing the model several times with a dummy training dataset and validation set. However, we found that our VGG-16 model is most compatible with a learning rate of 0.001.

```
sgd = optimizers.SGD(learning_rate=0.0001, momentum=0.5, decay = 0.0001)
```

```
vgg_model.compile(loss='binary_crossentropy',
                  optimizer=sgd,
                  metrics=['accuracy'])
```

FIGURE 4.13: A Code snippet that optimizes and compiles the VGG-16 model.

Since we have built our VGG-16 model, we will be training this model with two different datasets consecutively, as shown in figure 4.10.

4.4.2.1 Training Phase-1

In phase-1 of our training the VGG-16 model, we have used the pre-processed Celeb-df dataset to train. The pre-processing of the celeb-df dataset, as shown in figure 4.8, provides us with a multi-dimensional array of values that we will be using to train and validate with a ratio of 8:2.

Figure 4.14 is the code snippet used in training the model with the celeb_df dataset. As shown in the code snippet fit (), functions train the model with an epoch of 20, which means the model will be trained 20 times consecutively. For each epoch, the steps are calculated according to the length of the dataset. Finally, we have used model checkpoint to save changes made to the model and prevent any losses happening to the model. However, we have also used the early stopping mechanism to hint that the model will come to an early stop if the validation accuracy is not proportional to training accuracy.

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint("VGG_s_models/VGG16_1.h5", monitor='val_accuracy',
                             verbose=1, save_best_only=True,
                             save_weights_only=False, mode='auto', save_freq=1)
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20, verbose=1, mode='auto')
history = vgg_model.fit(x=train_ds, steps_per_epoch=len(train_ds),validation_data=val_ds,
                        validation_steps=len(val_ds), epochs=20,callbacks=[checkpoint, earlystop])
```

FIGURE 4.14: A code snippet to Train VGG-16 model (Phase1)

The result from training the model is represented in a graphical form. Figure 4.15 shows the training and test (validation) accuracy. We can see that the validation (test) accuracy fluctuates proportionally to the training accuracy. This fluctuation states that there is a bit of noise in the training dataset, causing the model to overfit.

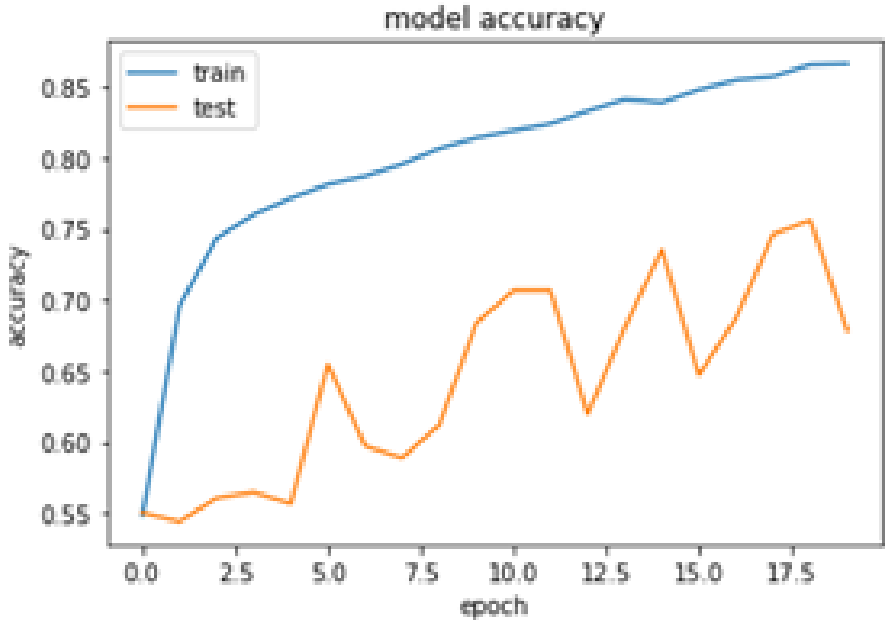

FIGURE 4.15: VGG-16 Model Accuracy Graph (Phase-1)

Apart from the training accuracy and validation accuracy, observing the losses of the model is also important to evaluate. Figure 4.16 shows the loss of the training and validation (test). As the accuracy is steadily increasing, the loss is gradually decreasing. However, the validation loss fluctuates and falls slowly, suggesting that the model is overfitting.

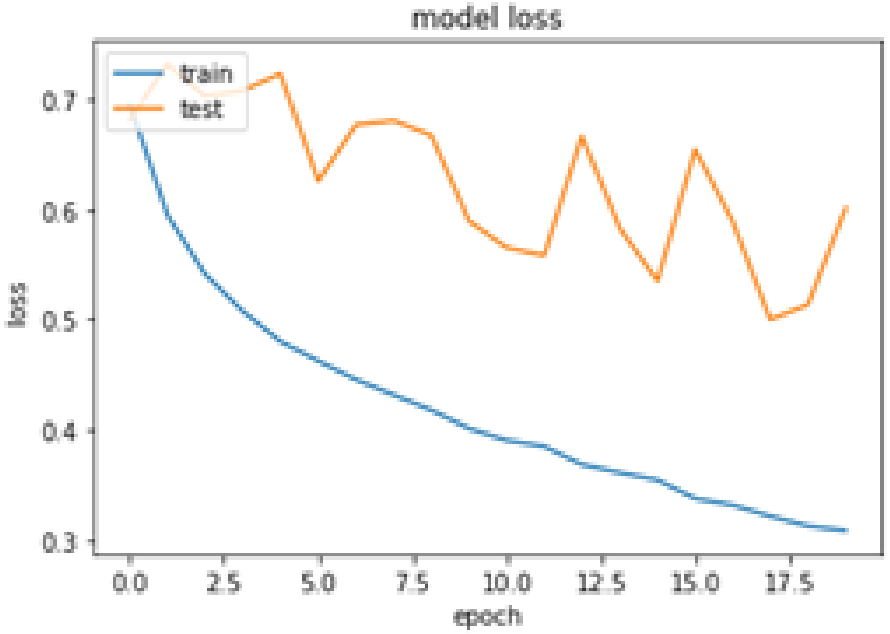

FIGURE 4.16: VGG-16 Model Loss (Phase-1)

Even though the validation accuracy and loss fluctuate, it gradually increases and decreases respectively, which shows that the model can improve its performance. Since we will re-train the model using a different dataset, the model has a chance to improve its overall performance.

To make the model available to re-train, we have saved the model weights in a folder that we will use for re-training.

4.4.2.2 Training Phase-2

In phase-2 of training, we will be improving the VGG-16 model from phase-1 training. Re-training the model would only increase the performance and prevent the model from overfitting further.

In this phase of training the VGG-16, we have used the pre-processed FaceForensic dataset consisting of high-resolution face images to train the model. Figure 4.17 shows the code snippet which is used to re-train the model.

We have used the exact training mechanism of phase 1 training, where we have used 20 epochs to re-train the model.

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint('VGG_s_models/VGG16_2.h5', monitor='val_acc', verbose=1, save_best_only=True, mode='max')
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='auto')

history = re_model.fit(x=train_ds_2, steps_per_epoch=len(train_ds_2),validation_data=val_ds_2,
                       validation_steps=len(val_ds_2), epochs=20, callbacks=[checkpoint,early])
```

FIGURE 4.17: A Code Snippet to Train VGG-16 model (Phase-2)

The results of phase 2 of training are represented graphically in figure 4.18 and figure 4.19. Comparing the model accuracy of phase-1 and phase-2, we can see that the training accuracy increased exponentially from phase 1 of training. The model accuracy in phase-2 is gradually growing with each epoch.

However, the validation (test) accuracy has remained the same with fluctuations, but it has a slight increment in the overall accuracy. The fluctuation in the accuracy still shows that there might be a possibility of overfitting.

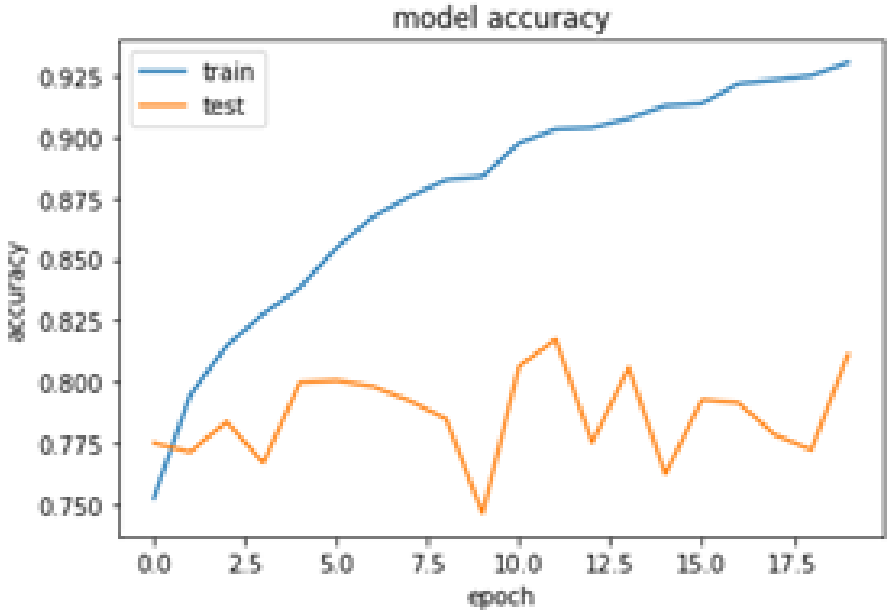

FIGURE 4.18: VGG-16 Model Accuracy Graph (Phase-2)

Now comparing the model loss of phase 1 and phase 2, the training loss is similar to phase 1, which gradually decreases for each epoch. However, the validation loss slowly drops and fluctuates vigorously for each epoch compared to the phase 1 validation loss. Figure 4.19 show the phase-2 model loss.

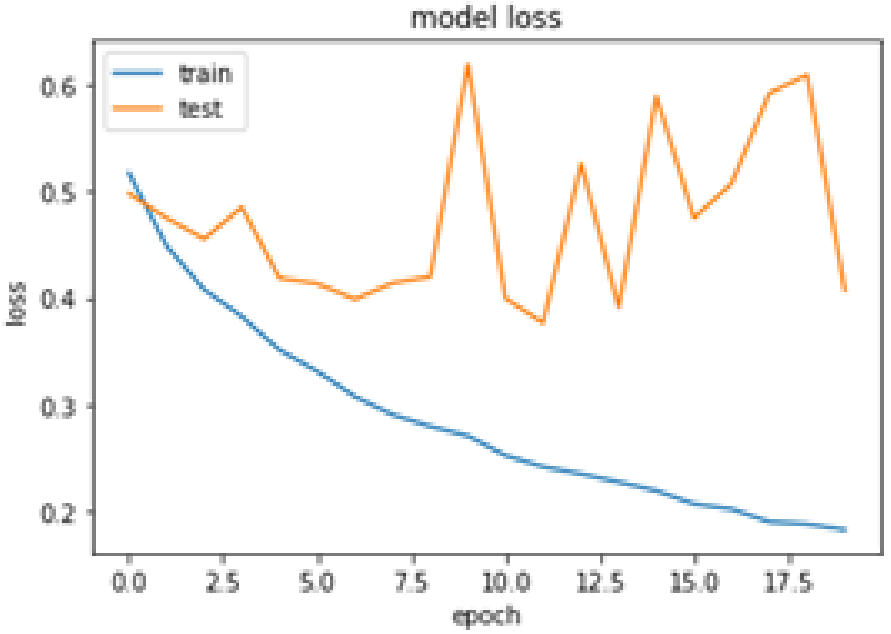

FIGURE 4.19: VGG-16 Model Loss (Phase-2)

In both the phase, the validation accuracy and loss are fluctuating with gradually increasing and decreasing respectively—the graphs of the validation accuracy state that

there is a possibility that the model is overfitting. However, we will clearly understand the VGG_16 model performance only after evaluation with our test dataset. Even though the model is overfitting, we will be able to take measures after observing the overall performance of the VGG-16 model.

Nevertheless, the overall validation accuracy from phase 1 to phase 2 drastically increased from 65% to 83%. The improvement in the validation accuracy suggests that the VGG-16 model is still able to predict deepfake videos with high accuracy.

4.4.3 InceptionNet V3

Christian S et al. published a research paper in 2015, which proposed a new deep convolutional neural network architecture named Inception. The main hallmark of this new version of CNN based architecture was to improve the computing resources utilization inside the network. IncpetionNet, also known as GoogLeNet, was created to improvise the performance of the CNN algorithm by increasing the size of the architecture. However, the researchers state that increasing the model's size would lead to overfitting and computation resources is required to run the algorithm. Therefore, the InceptionNet architecture was created with a sparsely connected architecture which was intended to solve the issues related to increasing the model size.

Figure 4.20 is an Inception model architecture with a dimension reduce, which limits input channels. The module shows that the author designed the architecture to increase with width rather than depth. By increasing the architecture by width, it would prevent deep neural network models from overfitting. In addition, it would also reduce the computational cost of the model by adding an extra layer of 1 X 1 convolutions, as shown in figure 4.20. This extra layer limits the input channels and ensures the model has an optimal solution in utilizing resourcing to compute large dataset tasks. The module was made open-source, calling it the GoogLeNet (InceptionNet Version 1) model, which was available for other researchers and experts to use for a wide range of applications, including big data and mobile applications.

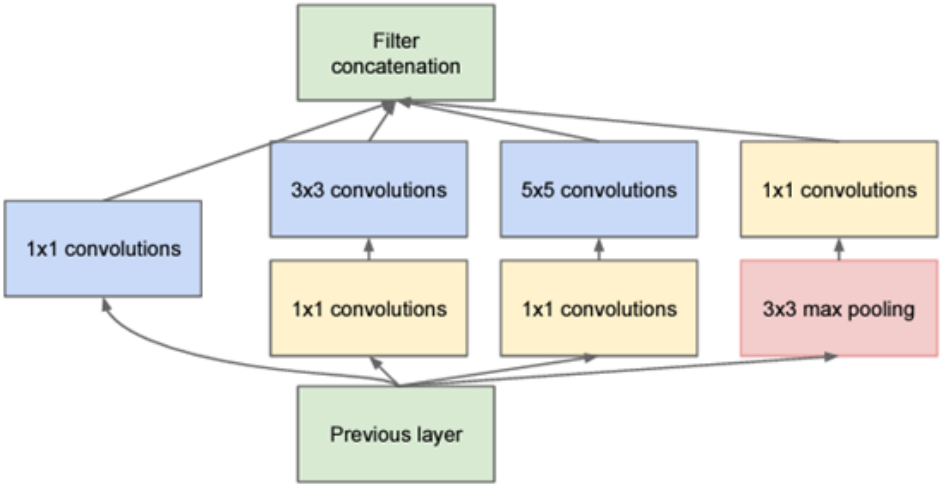

FIGURE 4.20: Standard InceptionNet Architecture

Over the years, the GoogLeNet (InceptionNet-V1) was further improved to produce various versions of InceptionNet architecture, increasing accuracy and reducing computational complexity. Zbigniew et al.[52] have published research proposing the versions of GoogLeNet (InceptionNet-V1). Researchers state that even though InceptionNet version 1 increased the model size and reduced the computational cost, there is still a gap for improvements that can produce high accuracy and better performance for various applications. Therefore, the researchers proposed InceptionNet-v2 (Version 2) and InceptionNet-v3 (Version 3), which had higher accuracy with less utilization of resources, and the models also reduced the complexity of IncpetionNet version 1 model.

For InceptionNet version 2, researchers used smart factorization methods and efficient convolutions layers to reduce the complexity of the models. In addition to these changes of the InceptionNet version 2 model, authors also noted that adding extra layers such as the average pooling method and Dropout methods to the architecture can produce a higher accuracy. By adding these layers to the model, it was able to detect the higher resolution images up to 1080 X 720 and small objects with lower resolutions of about 79 x 79. This improvement in the model architecture increased the accuracy when used for big data applications and others. However, the dropout layer was used to factorize the convolution layers and maintain a relatively low computation cost. The researchers made these improvements in the previous version InceptionNet and proposed a new version of InceptionNet, naming InceptionNet-v3 (Version 3). To better understand the InceptionNet-v3 model, figure 4.21 represents the model we have built.

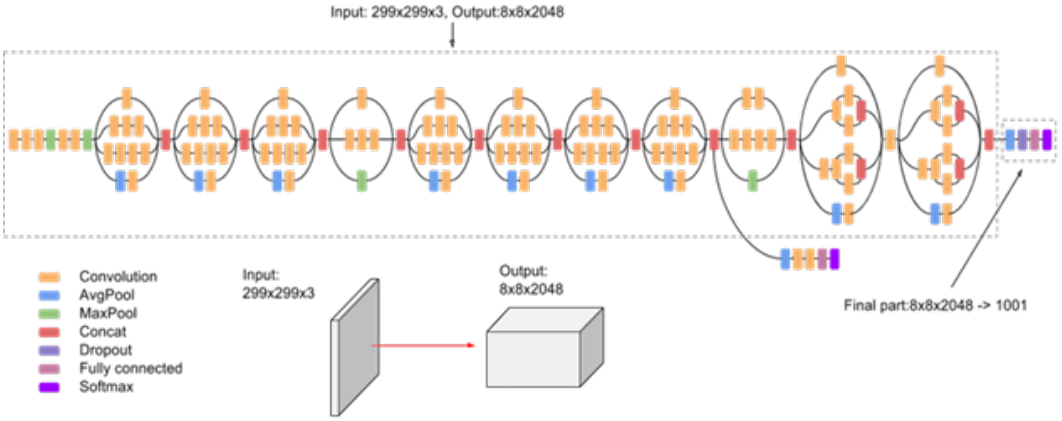

FIGURE 4.21: Our InceptionNet-V3 model's Arcitecture

InceptionNet-v3 would be a perfect model to detect deepfake videos, as it extracts features from images ranging from low to high resolutions. Since our training dataset consists of a wide range of resoltions of deepfake and orginal face images, inceptionNet-v3 would be able to incease the perfromance of our detection system and reduce the computational cost.

We have created the same replication of the InceptionNet-v3 model for our detection system, as shown in figure 4.21. However, since the deepfake detection system is a binary classficaiton problem, we modificed the last dense layer of the model to help the model transform the input layers to the output layer, producing a range of values between 0 to 1. This dense layer is similar to that of the VGG-16 model used in the above subsection. Identical to the VGG-16 model, we have also used the sigmoid activation function to transform the values within the range 0 to 1. Figure 4.22 is a code snippet on the implementation of the last layer, which is modified.

Figure 4.22 also shows the model compilation, similar to the VGG-16 model, as we used binary cross-entropy as our loss function. The loss function would help us evaluate our model, verifying the predicted labels using the actual labels of the validation dataset.

However, unlike in the VGG-16 model, we have used an RMSprop optimizer to help the InceptionNet-v3 model learn in mini-batches of data in a stochastic technique. The RMSprop uses an adaptive learning rate technique that changes over time, which prevents the model from overfitting and losing any dataset during the training. This optimizer suits well for the InceptionNet-v3 model because of the more extended width of layers

used in the model. RMSprop optimizer would help the models from losing the dataset during training.

```
# Add a final sigmoid layer with 1 node for classification output
x = layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.models.Model(base_model.input, x)

model.compile(optimizer = RMSprop(learning_rate=0.0001), loss = 'binary_crossentropy', metrics = ['acc'])
```

FIGURE 4.22: A code snippet to modifying the denser layer of InceptionNet-v3 and model compilation using RMSprop optimizer

Similar to the training of the VGG-16 model for this model, we will also be training it in two different phases with different datasets in each phase.

4.4.3.1 Training Phase-1

In phase-1 of training our InceptionNet-v3 model, we have used the pre-processed Celeb-df dataset to train. As we did in VGG-16 phase 1 training, we have used the multi-dimensional array values generated after pre-processing the celeb-df dataset, as shown in figure 4.8. The dataset values are further split into training dataset and validation dataset with a ratio of 8:2.

Figure 4.23 is the code snippet used in training the model with celeb_df dataset. Unlike in VGG-16 model phase 1 training, for the InceptionNet-v3 phase-1 training, we have used 30 epochs, as shown in figure 4.23. For each epoch, the steps are calculated according to the length of the dataset. After testing the InceptionNet-v3 model with a dummy dataset, we found that 30 epoch was a stable amount to train. Finally, we have used model checkpoint and early stopping mechanism, same as we did in VGG-16 model training. The early mechanism will help us check if the validation accuracy is proportional to training accuracy. If it is not propositional, the training will come to an early stop.

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint('Inception_v3_models/Inception_v3.h5', monitor='val_acc', verbose=1, save_best_only=True, mode='max'
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='auto')

history = model.fit(x=train_ds, steps_per_epoch=len(train_ds),validation_data=val_ds,
                    validation_steps=len(val_ds), epochs=30, callbacks=[checkpoint,early])
```

FIGURE 4.23: A code snippet to Train IncpetionNet-v3 model (Phase1)

The result from training the model is represented in a graphical form. Figure 4.24 shows the training and test (validation) accuracy. We can see that the validation (test) accuracy fluctuates, and it is proportional to the training accuracy. Like in the VGG-16

model training, this model also has a fluctuation that states that there is a bit of noise in the training dataset, which might be a possibility of overfitting.

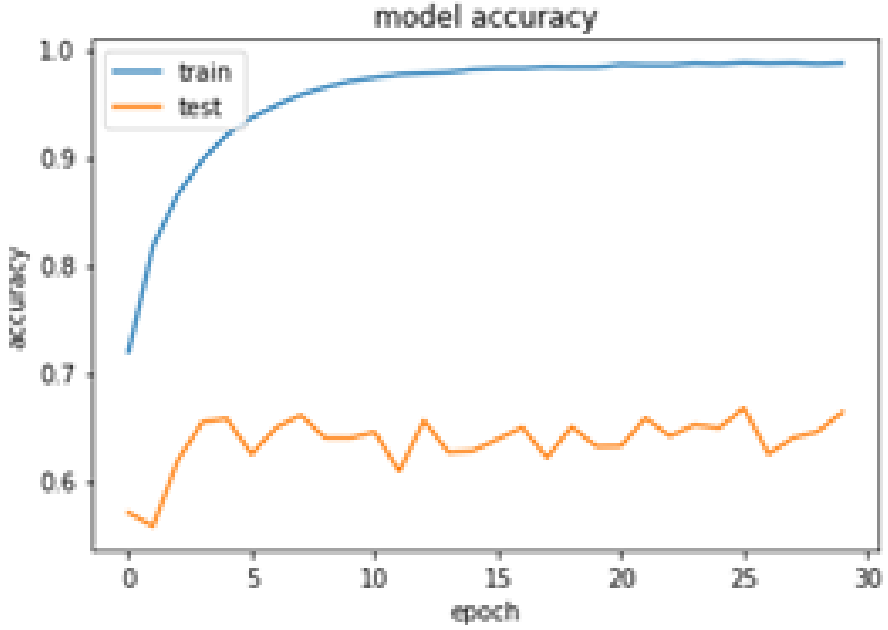

FIGURE 4.24: InceptionNet-v3 Model Accuracy Graph (Phase-1)

Apart from the training accuracy and validation accuracy, observing the losses of the model is also important to evaluate. Figure 4.25 shows the loss of the training and validation (test). As the accuracy is steadily increasing, the loss is gradually decreasing. However, the validation loss is fluctuating and rising. Which clearly shows the signs of overfitting.

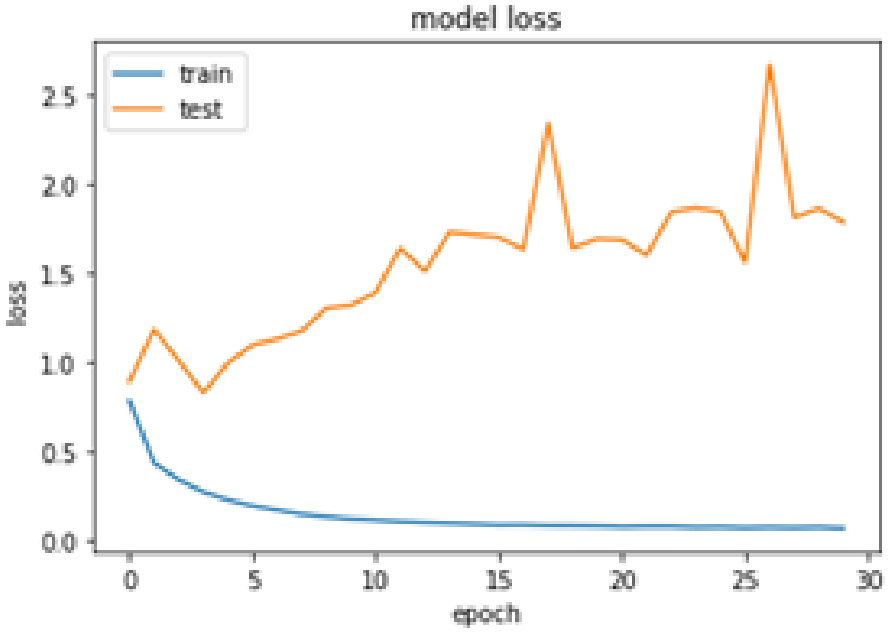

FIGURE 4.25: InceptionNet-v3 Model Loss (Phase-1)

We can observe that validation accuracy is fluctuating and gradually increasing. We also observed that the validation loss is rising progressively. The model loss graph suggests that the model is overfitting ultimately. Since we will re-train the model using a different dataset, the model has a chance to improve its accuracy and its overall performance. Therefore, we are not going to make any changes to the model in this phase.

To make the model available to re-train, we have saved the model weights in a folder that we will use for re-training.

4.4.3.2 Training Phase-2

After phase-1 of training, we are now aware that the training of InceptionNet-v3 caused overfitting. Therefore, as shown in figure 4.35, we have lowered the learning rate of the model. By reducing the learning rate, the optimizer would control the changes done to the model and avoid overfitting.

```
from keras import backend as K
# To get learning rate
print(K.get_value(re_model.optimizer.learning_rate))
# To set learning rate
K.set_value(re_model.optimizer.learning_rate, 0.00001)
```

FIGURE 4.26: A code snippet to modify the learning rate of the RMSprop Optimizer

After setting a new value to the RMSprop optimizer, we have used all the improvements made to the InceptionNet-v3 model from phase 1 to re-train it in this phase 2 of training.

We have used the pre-processed FaceForensic dataset for this training phase, which consists of high-resolution face images to train the model. Figure 4.27 shows the code snippet which is used to re-train the model. We have also reduced the epochs from 30 to 15 in this training phase to avoid overfitting and increase the performance. The rest of the parameters in the fit() function is similar to phase-1 of training.

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint('Inception_v3_models/Inception_v3_2.h5', monitor='val_acc', verbose=1, save_best_only=True, mode='m
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='auto')

history = re_model.fit(x=train_ds_2, steps_per_epoch=len(train_ds_2),validation_data=val_ds_2,
                       validation_steps=len(val_ds_2), epochs=15, callbacks=[checkpoint,early])
```

FIGURE 4.27: A code snippet to Train InceptionNet-v3 model (Phase-2)

The results of phase 2 of training are represented graphically in figure 4.28 and figure 4.29. Comparing the model accuracy of phase-1 and phase-2, we can see that the training accuracy increased exponentially from phase 1 of training. The model accuracy in phase-2 is gradually growing with each epoch.

However, the validation (test) accuracy has remained the same with fluctuations, but it slightly incremented overall accuracy. The change in the accuracy still shows that there might be a possibility of overfitting.

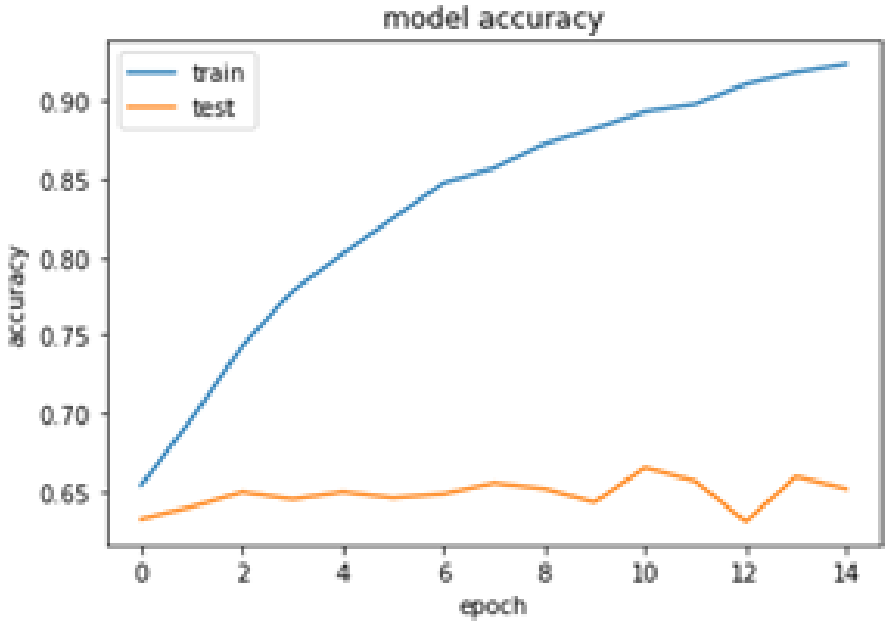

FIGURE 4.28: IncpetionNet-v3 Model Accuracy Graph (Phase-2)

Now comparing the model loss of phase 1 and phase 2, the training loss is similar to phase 1, which gradually decreases for each epoch. However, the validation loss slowly drops and fluctuates vigorously for each epoch compared to the phase 1 validation loss. Figure 4.29 show the phase-2 model loss. However, the overall validation loss has decreased drastically from phase 1. The overall loss suggests that the model still will be able to detect high-quality deepfake videos.

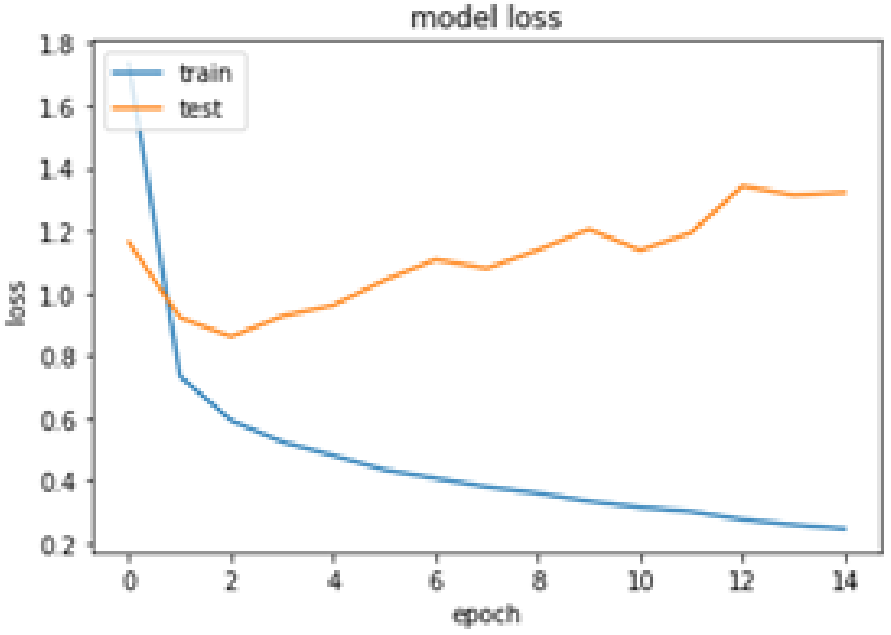

FIGURE 4.29: InceptionNet-v3 Model Loss (Phase-2)

In both the phase, the validation accuracy and loss are fluctuating with gradually increasing and decreasing respectively—the graphs of the validation accuracy state that there is a possibility that the model is overfitting. However, we will clearly understand the InceptionNet-v3 model performance only after evaluation with our test dataset. Even though the model is overfitting, we will be able to take measures after observing the overall performance of the InceptionNet-v3 model.

Nevertheless, the overall validation accuracy is increased slightly from 65% to 68%, and the overall validation loss is decreased from 2.0% to 1.3%. These results suggest that the InceptionNet-v3 model is still able to predict deepfake videos with reasonable accuracy.

4.4.4 EfficientNet-B0

Mingxing T et al. [53] have proposed a new scaling method that can be used to scale the model's height, width, and resolution using an effective compound coefficient. EfficientNet consists of 8 different models proposed by the researchers that have outperformed all the Convolutional neural network-based algorithms. CNN based algorithms like InceptionNet have an arbitrary scaling method using factorization. This means that the algorithms width or height, or resolution is increased based on the computation resources. However, the arbitrary scaling method has various issues such as model overfitting and low performance for a few applications that use high computational resources. The researchers have proposed EfficientNet algorithms to overcome these issues by introducing a uniform scaling method, where the networks width, depth and resolution have a fixed set of coefficients. For instance, if we want to increase the computational resources to 2N, then we can improve the network depth by αN, width by βN and the resolution of the images by γN, where α, β, γ are constant coefficients. These constant coefficients are determined by the model and the compound coefficient ϕ, which is scaled uniformly in a principled way to form various EfficientNet algorithms.

The EfficientNet models achieve both higher accuracy and efficiency than that of the existing CNN based algorithms. Figure 4.30 shows a graph that compares all the EfficientNet models from B0 to B7 with all the CNN-based models (Black dotted line). The number of paraments shown in the graph depends on the computation resources. A greater number of paraments means that the computational resource is greater. EfficientNet-B7

is explicitly built for high computational resources. Comparing it to the CNN algorithms, it achieves an accuracy of 97% outperforming all the other CNN algorithms (plotted in black dotted lines on the graph) that use high computational resources.

Overall, figure 4.30 gives an idea of comparing the EfficientNet models to other models and suggests that different EfficientNet models work best based on their number of paraments or computational resources.

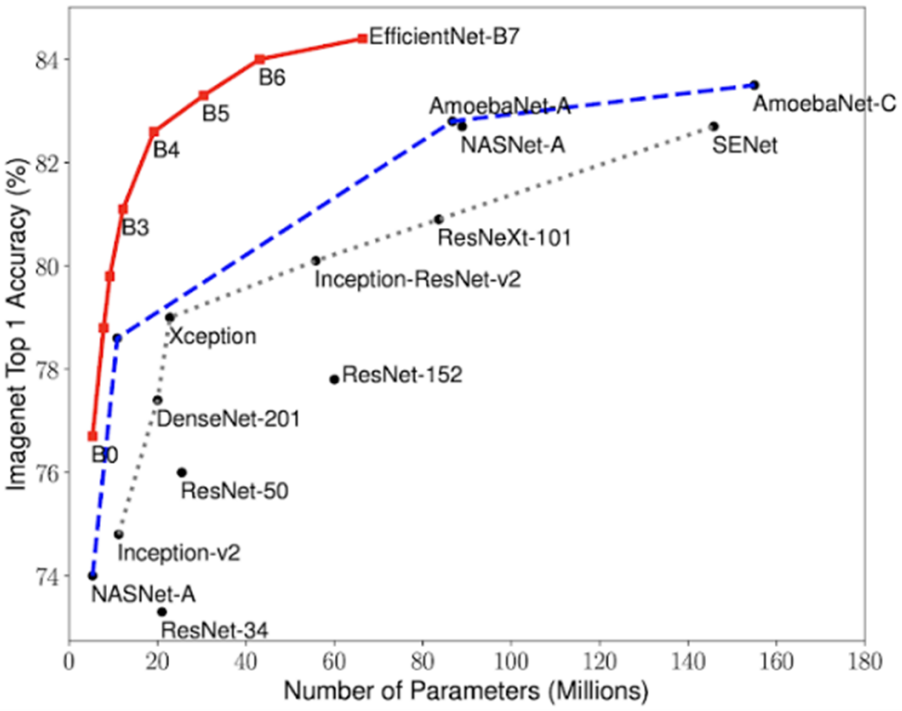

FIGURE 4.30: A graphical representation comparing all the CNN models (dotted black line) with EfficientNet models (Red Line).

For building our deepfake detection system, we are using EfficientNet-B0 because we use a hardware system with less computational resources to build the detection system. An advantage of using EfficientNet-B0 is that it has overperformed InceptionNet-v3 and VGG-16 models, achieving higher accuracy in all the different computation resources. Another major benefit of using EfficientNet-B0 is that it is a lightweight algorithm that can run faster with the training and detection of our deepfake videos. Additionally, EffcientNet-B0 also detects high-resolution deepfake videos, including resolutions supported by most existing social media applications.

In our project, we have built EffcientNet-B0 from scratch. Figure 4.31 shows the architecture of the EfficientNet-b0 model which we have used. The model's input layer

accepts only images with a size of 244 X 244 (height X width). Since the EfficientNet-B0 model has a very depth architecture, we have mentioned efficientnet-b0: Functional in figure4.31 to make the architecture less complex. However, the model extracts all the important features such as brightness, pixel rate, and other image features from high-resolution images in the dataset.

In this model, we have also used a global average pooling method that calculates the average of all the extracted features and stores as a single value. As EfficientNet-B0 has a greater depth and width than other models, the global average pooling method fits the model the best.

Finally, we have used a dropout layer similar to that of the VGG-16 model in this model. This layer helps us to prevent the model from overfitting the training dataset. The dense layer is modified for all the CNN models in our project, as the layer helps us convert the image values to a binary value (Input layer to output).

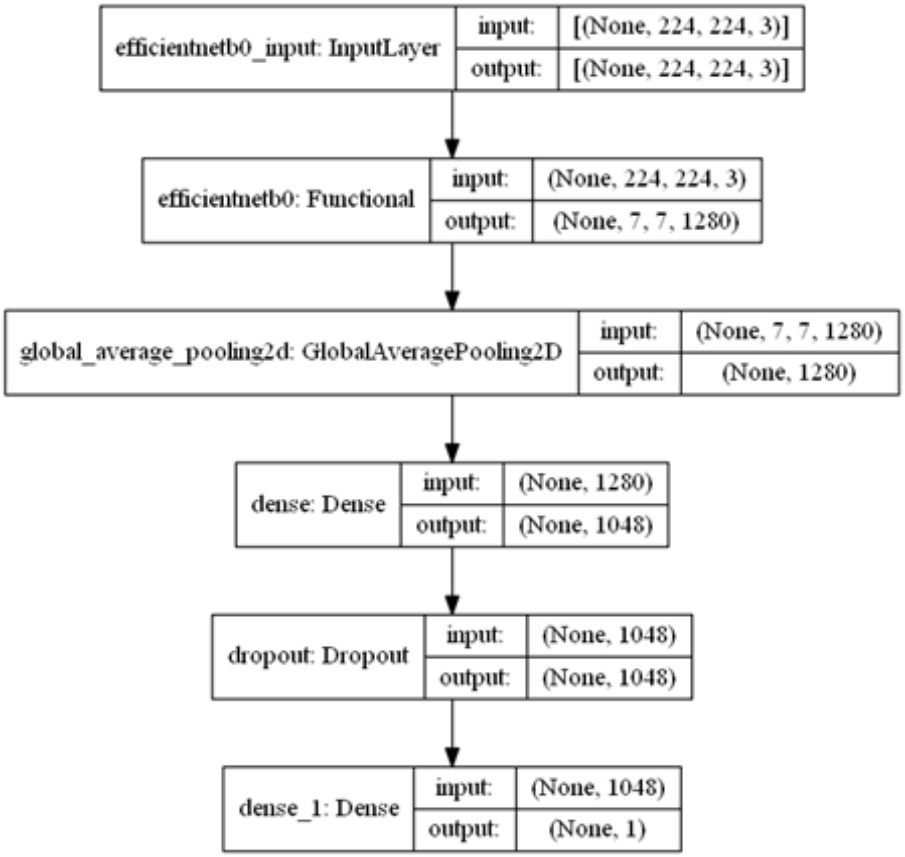

FIGURE 4.31: EfficientNet-B0 model Architecture.

Figure 4.32 shows the code snippet to compile the model, similar to the other two models in the project. However, the optimizer we have used is RMSprop similar to that we used

in InceptionNet-v3, which will help the model be evaluated with the predicted labels and the actual labels of the validation dataset.

```
opt = RMSprop(learning_rate=0.00001)
model_rmsprop.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
model_rmsprop.summary()
```

FIGURE 4.32: A code snippet to modify the denser layer of the EfficientNet-B0 model and its compilation using RMSprop optimizer.

This efficientnet-b0 model will be trained in two phases with two different datasets accordingly, the same way we have done for the other models during training.

4.4.4.1 Training Phase-1

In phase-1 of training our EfficientNet-B0 model, we have used the pre-processed Celeb-df dataset to train. This dataset is further split into training and validation datasets with a ratio of 8:2.

We have implemented the EfficientNet-B0 using the fit () function. In the function, we have used only epochs of 15. We chose fewer epochs for training this model than other models because after running the models multiple times, we found that 15 epoch is most suitable for training our EfficientNet-B0 model. However, during this training phase, we have not implemented model checkpoints or early stopping as it requires a lot of resources.

Observing the training results, we have represented them in a graphical form (as shown in figures 4.33 and figure 4.25). From figure 4.33, we understand that the validation (test) accuracy is fluctuating, gradually increasing until the last epoch. However, the training accuracy increases for each epoch. The fluctuation in the validation accuracy might be a possibility of overfitting, which is similar to the other two models.

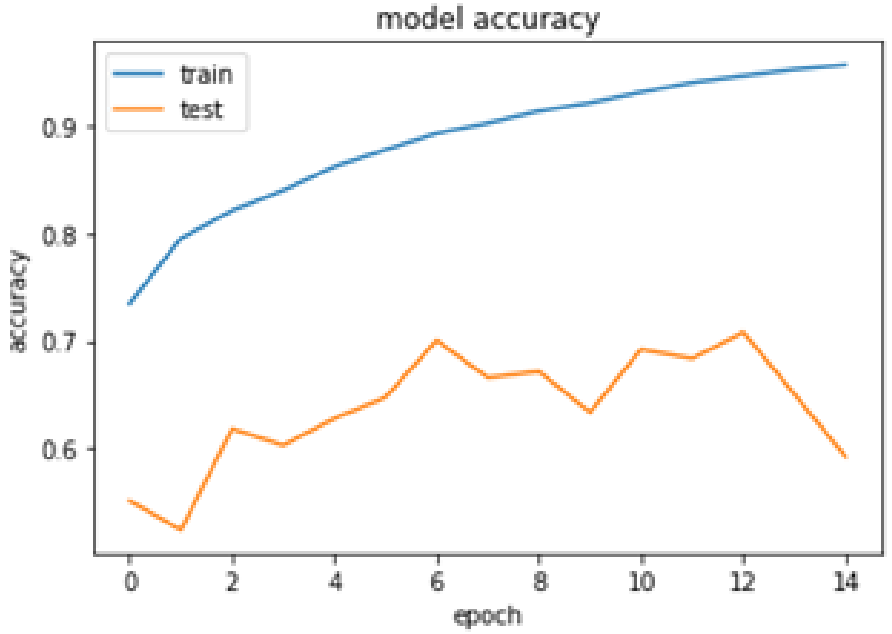

FIGURE 4.33: EfficientNet-B0Model Accuracy Graph (Phase-1).

Observing the model's loss in phase 1 of training EfficientNet-B0, we see that validation loss is slightly fluctuating, but the training loss is gradually increasing. We can clearly state that the validation loss is fluctuating with a very slight difference.

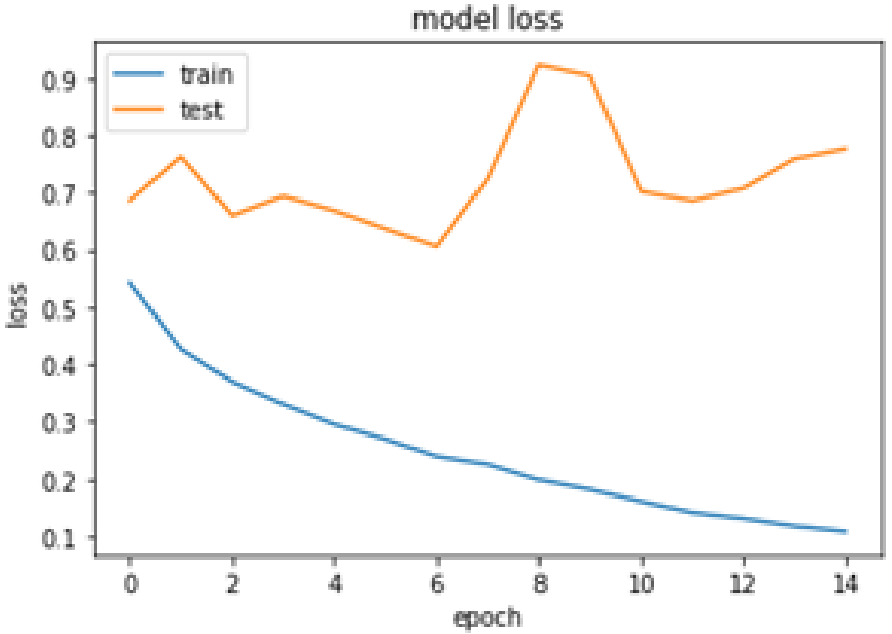

FIGURE 4.34: EfficientNet-B0Model Loss Graph (Phase-1).

The main reasons for the validation accuracy and loss to fluctuate in this phase might be because we have not used the model checkpoints and early stopping mechanism

or because of the high learning rate. However, we can still state that the model is performing well because the fluctuation in validation loss and accuracy is very low. These observations are further correct in phase 2 of training.

Finally, to make the model available to re-train, we have saved the model weights in a folder that we will use for re-training.

4.4.4.2 Training Phase-2

Noting the observation of the model's performance in phase-1, we have made various changes in this phase 2 of training the EfficientNet-B0 model. As shown in figure 4.26, a code snippet, we have lowered the learning rate of the model. By reducing the learning rate, the optimizer would control the changes done to the model and avoid overfitting.

```
from keras import backend as K
# To get learning rate
print(K.get_value(re_model.optimizer.learning_rate))
# To set learning rate
K.set_value(re_model.optimizer.learning_rate, 0.000001)
```

FIGURE 4.35: A code snippet to modify the learning rate of the RMSprop Optimizer.

After setting a new value to the RMSprop optimizer, we have used all the improvements made to the EfficientNet-B0 model's phase 1 training in phase 2 to re-train it with a FaceForensic dataset.

In phase 2 of training the EfficientNet-B0 model, we have used the pre-processed dataset that consists of high-resolution face images to train the model. The implementation of phase 2 of training is shown in figure 4.36. The code snippet shows that we used model checkpoint and early stopping to save the model and verify the validation accuracy accordingly for this training phase.

As we observed in phase 1 of training, the validation loss and accuracy fluctuated vigorously in the last few epochs. Therefore, we have increased the total epoch from 15 to 20. The implementation is shown in figure 4.36.

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint('EffiecientNetB0_models/EffcientNetB0_model_2.h5', monitor='val_accuracy', verbose=1, save_best_only
early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20, verbose=1, mode='auto')

history = re_model.fit(x=train_ds_2, steps_per_epoch=len(train_ds_2),validation_data=val_ds_2,
                       validation_steps=len(val_ds_2), epochs=20, callbacks=[checkpoint,early])
```

FIGURE 4.36: A code snippet to Train EfficientNet-B0 model (Phase-2).

After re-training the model for 20 epochs, we represented the results in graphical form as shown in Figures 4.37 and 4.38.

In Figure 4.37, The model's training accuracy and model's validation accuracy are propositional. As we can observe, there is no fluctuation in validation accuracy, and it gradually increases with each epoch. However, the validation (test) accuracy in the last two epochs remains constant. The graph in figure 4.37 states that the model is well trained without any problems of overfitting or underfitting.

The overall model's validation accuracy has drastically increased from 58% to 75% in phase-2.

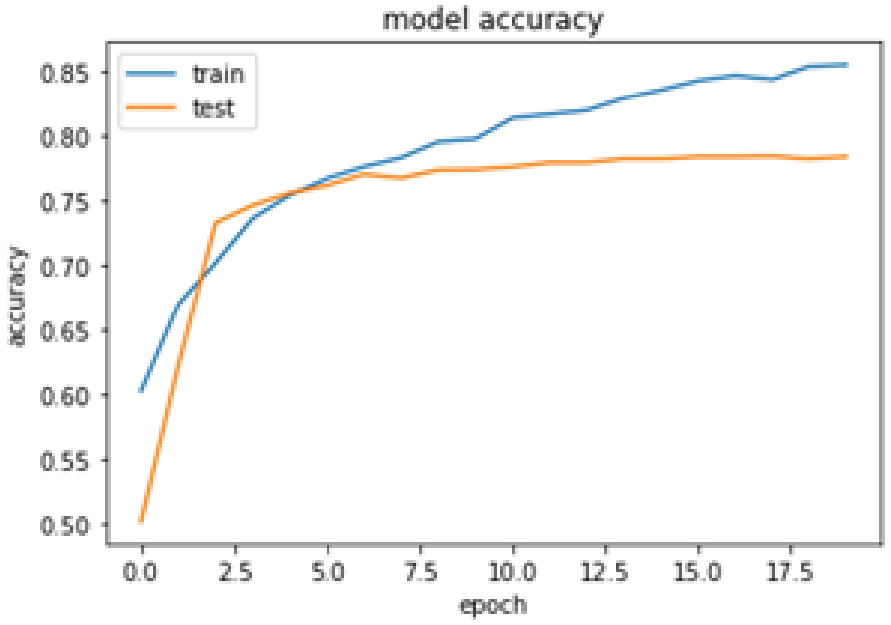

FIGURE 4.37: EfficientNet-B0 Model Accuracy Graph (Phase-2).

As we observed no overfitting in the model's accuracy graph, we should be expecting a similar result in the model's loss. Figure 44 states that the model's training loss remains constant, which is a straight line. The model's validation loss shows a sudden drop, and then it comes to a constant after the second epoch. The loss values are minute in

which we are not able to observe in figure 4.34. We can now confirm that the model is perfectly working, and it can predict values without any overfitting.

The validation's loss has no fluctuation when compared to the model's loss of phase 1 and phase 2. The validation's loss and training loss has been drastically decreased from phase 1.

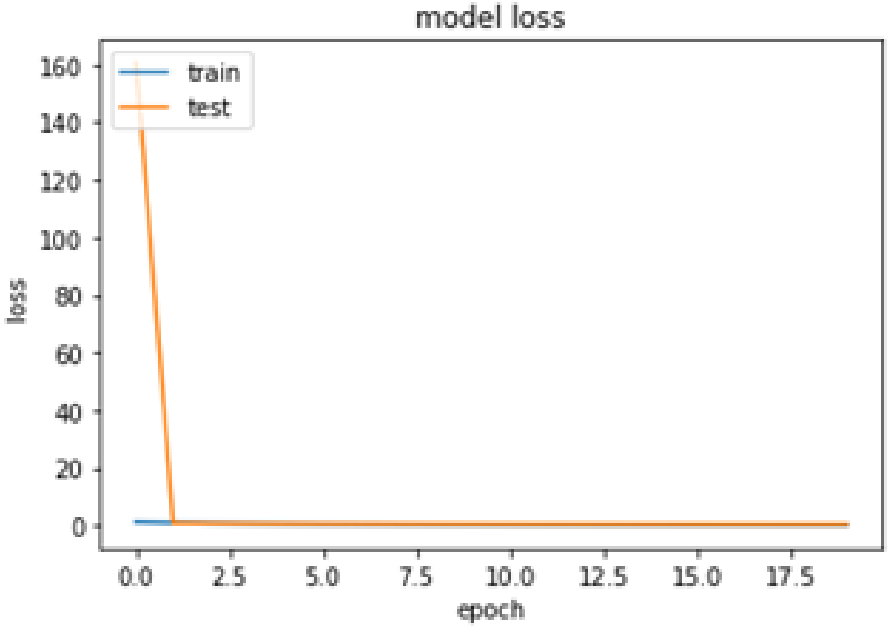

FIGURE 4.38: EfficientNet-B0 Model Loss Graph (Phase-2).

Now comparing the model's performance in phase-2, we were able to avoid the model overfitting successfully. The overall validation accuracy of the model has been drastically increased from achieved 58% to 78% in phase 2 of the training. The overall validation accuracy suggests that the EfficientNet-B0 model can successfully predict deepfake videos with reasonable accuracy. Now that we have trained all three models, the VGG-16 has achieved the highest validation accuracy of 83%. However, EfficentNet-B0 and InceptionNet-v3 have achieved an accuracy of 78% and 68%, respectively.

Nevertheless, the VGG-16 and IncepetionNet-v3 still show some signs of overfitting in the model. But the values of validation's accuracy and loss has a minimal fluctuating rate. These might be due to various reasons, which we will be discussing in detail in the discussion chapter of the research. However, at the end of the project, we will be using a model stacking ensemble method to predict deepfake videos. The ensemble model would help us eliminate any minor fluctuations in the models' accuracy and loss.

Chapter 5

Evaluation

In this section, we will be evaluating our trained deepfake models in two differed evaluation methods. The evaluation methods are as follows:

1. Models' Evaluation: We will evaluate all three models and create a stacking model (Ensemble method).
2. Evaluating Existing system and our proposed system: We will assess our stacking model with two other existing systems with a set of selected videos.

5.1 Models's Evaluation

As mentioned previously, for testing our deepfake models, we will be using deepfake detection challenge dataset. We have pre-processed this dataset by converting it into frames and then data augmentation pre-processing methods, which we did previously for the other two public datasets (FaceForensic and Celeb_DF datasets). Figure 5.1 shows a few images as an example from the test dataset after pre-processing.

FIGURE 5.1: Sample data frames from the test dataset.

After the test data pre-processing, there are 1200 original video frames and 4800 deepfake video frames. These frames are further converted into a multidimensional array using the code snippet in figure 5.2. Since we trained our models with an image size of 224 X 224 (height X width), we have considered the same image size for testing. The code snippet in figure 5.2 converts the values into X and Y, where X is the extracted multidimensional array with float values and Y is a binary value (0 or 1).

```
test_dir = 'final_dataset/ddff/'
datagen = ImageDataGenerator(rescale=1./255)

test_gen = datagen.flow_from_directory(directory=test_dir, target_size=(224,224),
                                       class_mode='binary',batch_size=10, shuffle=False)
```

FIGURE 5.2: A code snippet to convert frames to an array of values.

We will be using the test_gen variable, which has the X and Y values of the test dataset, to evaluate all three models individually and then compare their performance. For our evaluation, we will use various computing metrics for each model. Firstly, we are going to calculate the accuracy of each of the models using,

$$Accuracy = \frac{NumberofCorrectPredictions}{TotalNumberofPredictionsmade}$$

However, calculating the accuracy is not enough to analyse the overall performance of the model. Since we are solving a binary classification problem, there are values such as true positives (correctly predicted deepfake videos), false negatives (correctly predicted original videos), false positives (falsely predicted deepfake videos), and true negative (falsely predicted original videos) that must be calculated during the models' evaluation. Therefore, we will be computing three other metrics to understand the model's performance better. The three metrics are as follows:

- Precision: This is an excellent measure to determine if the deepfake videos in the test dataset are misclassified by the deep neural network model. To calculate this metric, we are using:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

- Recall: This metric calculates how many deepfake videos were detected correctly by the deep neural network model. To calculate recall, we are using the formula:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

- F1 Score: It measures test accuracy, as it is calculated using precision and recall. By calculating the F1 score, we determine the balance between precision and recall and find the uneven class distribution (Checking if the original videos are classified correctly). To calculate the F1 score, we are using:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

We have used a python library called classification_report to help compute these metrics for each model.

5.1.1 VGG-16 Model

We will be testing phase-2 of the trained VGG-16 model, as this phase is the updated version of the VGG-16 model. After phase-2 of training the model, we stored the model weights in a folder. Now for testing the model, we will be loading it, as shown in figure 5.3. We have used the load_model () function from the TensorFlow library in the code snippet to load the stored model. The vgg16 variable will be used to predict values from the test dataset, shown in figure 5.3.

```
#VGG16 Final Model
vgg16 = tf.keras.models.load_model('VGG_s_models/model_2/')
```

FIGURE 5.3: Code snippet to store VGG-16 wights in a variable.

After loading the VGG-16 model' weights, we now have used the predict () function as shown in Figure 5.4 to predict our test dataset. The predict functions return binary values (0 or 1), which we consider as predicted labels.

```
vgg16_predictions = vgg16.predict(test_gen, steps=len(test_gen), verbose = 1)
```

FIGURE 5.4: Code snippet to predict test dataset using VGG-16.

We now have our final predicted values, and we are also aware of the real values of the test dataset, which is the actual values. Figure 5.5 is code snippets that show the accuracy of the VGG-16 model. Comparing the predicted and actual labels of the test dataset, VGG-16 has achieved an accuracy of 71% (approx.).

```
base_score_test = metrics.accuracy_score(test_gen.classes, DT(vgg16_predictions))
print('VGG Model Test Score ',base_score_test)
```

```
VGG Model Test Score  0.7095
```

FIGURE 5.5: VGG_16 model's accuracy.

For a more detailed analysis of the VGG_16 model performance, we used the classification_report python library to compute Precision, recall and F1 score metrics. Figure 5.6 shows the code snippet and the result simultaneously.

From the report, we can observe that the average precision of the model is 72%, which means that 72% of the original and deepfake videos in the test dataset were correctly predicted by the model. However, the average recall of the model is 71%, which means only 71% of the deepfake video's frames in the dataset were correctly predicted by the model. The overall F1 score is 71%, which means the model could achieve a balanced score of 71%.

```
base_report_VGG = classification_report(test_gen.classes,DT(vgg16_predictions))
print(base_report_VGG)
```

```
              precision    recall  f1-score   support

           0       0.82      0.81      0.82      4800
           1       0.29      0.30      0.29      1200

    accuracy                           0.71      6000
   macro avg       0.55      0.56      0.56      6000
weighted avg       0.72      0.71      0.71      6000
```

FIGURE 5.6: VGG-16's classification report.

To better understand how many frames in the dataset were predicted correctly and how many were mispredicted. Figure 5.7 is a confusion matrix, which plots the dataset's actual labels against the model's predicted label. There are 6000 video frames in the deepfake detection challenge dataset. We can observe that only 3791 deepfake videos frames and 402 original video frames were correctly predicted by our VGG-16 model.

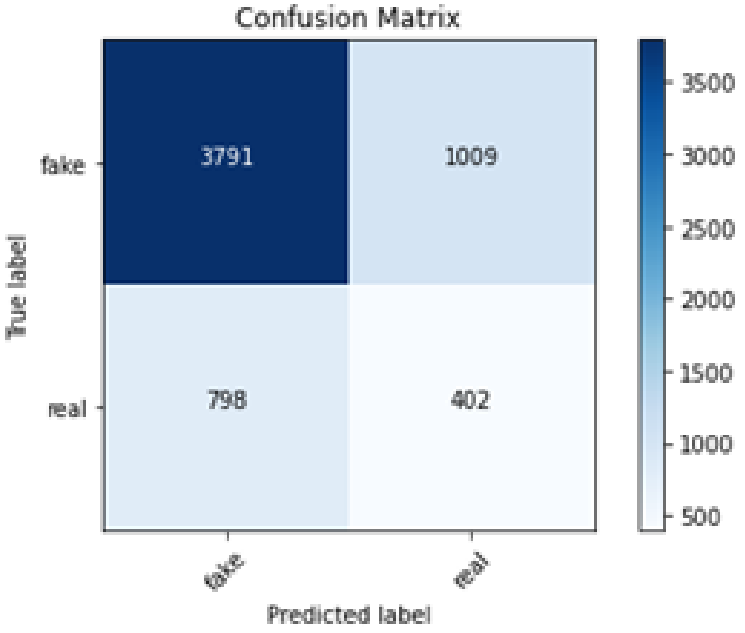

FIGURE 5.7: VGG-16's confusion matrix.

5.1.2 InceptionNet-v3 Model

The phased-2 trained IncepetionNet-v3 model will be tested, as it was the most updated version of the InceptionNet-v3 model. After we trained the IncepetionNet-v3 model in both phases, we stored the model's weights in a folder. It has been loaded and stored in a variable for testing, similar to figure 5.3. The code snippet that we have used the load_model () function from the TensorFlow library to load the stored model.

After loading the IncepetionNet-v3 model's weight, we have now used the predict () function to predict 6000 video frames in the test dataset. The predict functions return a binary value (0 or 1) for each frame in the dataset, also known as predicted labels.

After predicting each frame in the test dataset, we verify the predicted labels with actual labels and compute the model's overall accuracy. Figure 5.8 is code snippets that show the accuracy of the IncepetionNet-v3 model. Our model achieved an overall accuracy of 74%, which is higher than the VGG-16 model's accuracy.

```
base_score_test_IN = metrics.accuracy_score(test_gen.classes,DT(InceptionNet_predictions)
print('Inception Model Test Score ',base_score_test_IN)
```

```
Inception Model Test Score  0.7438333333333333
```

FIGURE 5.8: InceptionNet-v3 model's accuracy.

For a more detailed analysis of the IncepetionNet-v3 model performance, we used the classification_report python library to compute Precision, recall and F1 score metrics. Figure 5.9 shows the code snippet, and the result computed using the library.

From the report, we can observe that the average precision of the model is 73%, which means that 73% of the original and deepfake videos in the test dataset were correctly predicted by the model. However, the average recall of the model is 74%, which means only 74% of the deepfake video's frames in the dataset were correctly predicted by the model. The overall F1 score is 74%, which means the model could achieve a balanced score of 74%.

```
base_report_IN = classification_report(test_gen.classes,DT(InceptionNet_predictions))
print(base_report_IN)
```

```
              precision    recall  f1-score   support

           0       0.83      0.86      0.84      4800
           1       0.33      0.28      0.30      1200

    accuracy                           0.74      6000
   macro avg       0.58      0.57      0.57      6000
weighted avg       0.73      0.74      0.74      6000
```

FIGURE 5.9: IncepetionNet-v3's classification report.

To better understand how many frames in the dataset were predicted correctly and how many were mispredicted. Figure 5.10 is a confusion matrix, which plots the dataset's actual labels against the model's predicted label. The graph shows that 4036 out of 4800 deepfake videos frames have been predicted correctly. We can also observe that our InceptionNet-v3 model correctly predicted only 367 out of 1200 original video frames.

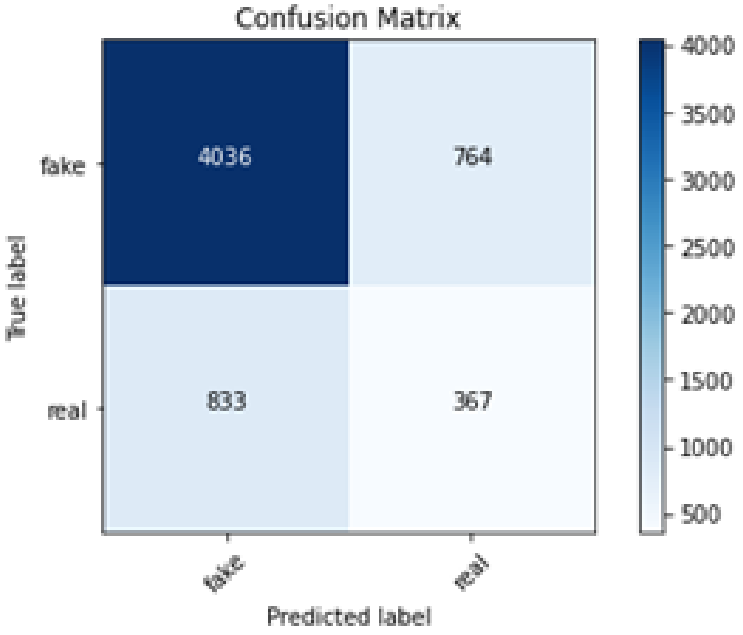

FIGURE 5.10: InceptionNet-v3's confusion matrix.

5.1.3 EfficientNet-B0 Model

Similar to the other model's testing, for the EfficientNet-B0 model, we have used the saved the phase 2 trained model. We loaded the EfficientNet-B0 model's weights to a variable from a folder. We have also used the same TensorFlow function to load the weights and save them in a variable.

After loading the EfficientNet-B0 model's weight, we have used the predict () function to predict 6000 video frames in the test dataset. The predict functions return a binary value (0 or 1) for each frame in the dataset, also known as predicted labels.

After predicting each frame in the test dataset, we verify the predicted labels with actual labels and compute the model's overall accuracy. Figure 5.11 is code snippets that show the accuracy of the EfficientNet-B0 model. Our model achieved an overall accuracy of 73%, greater than the VGG-16 model and almost equal to the IncepetionNet-v3 model.

```
base_score_test_EN = metrics.accuracy_score(test_gen.classes,DT(EffcientNet_predictions))
print('EffcientNetB0 Model Test Score ',base_score_test_EN)
```

```
Inception Model Test Score  0.7256666666666667
```

FIGURE 5.11: EfficientNet-B0 model's accuracy.

To get a more detailed analysis of the EfficientNet-B0 model's performance, we used the classification_report python library to compute Precision, recall and F1 score metrics. Figure 5.12 shows the code snippet, and the result computed.

From the report, we can observe that average precision and recall metrics are 73% which means that 73% of the original and deepfake videos in the test dataset were correctly predicted by the model, and 73% of the deepfake video's frames in the dataset were correctly predicted. In addition, the overall F1 score is 74%, which means that the precision and recall of the model are balanced equally.

```
base_report_EN = classification_report(test_gen.classes,DT(EffcientNet_predictions))
print(base_report_EN)
```

	precision	recall	f1-score	support
0	0.83	0.83	0.83	4800
1	0.31	0.31	0.31	1200
accuracy			0.73	6000
macro avg	0.57	0.57	0.57	6000
weighted avg	0.73	0.73	0.73	6000

FIGURE 5.12: EfficientNet-B0 model's classification report.

To better understand how many frames in the dataset were predicted correctly and how many were predicted wrongly. Figure 5.13 is a confusion matrix, which plots the dataset's actual labels against the model's predicted label. The graph shows that only 3886 out of 4800 deepfake videos frames have been predicted correctly. We can also observe that our EfficientNet-B0 model correctly predicted only 407 out of 1200 original video frames.

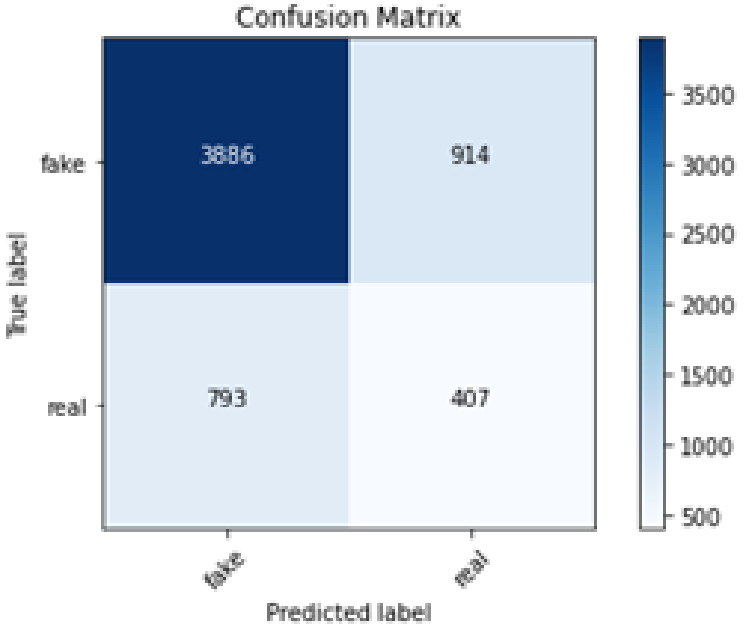

FIGURE 5.13: EfficientNet-B0 model's confusion matrix.

5.1.4 Ensemble Model

Ensemble learning in machine learning is an approach that combines the predictions of multiple models to improve the overall performance. Using an ensemble learning method for our deep learning models, we can achieve a better generalisation performance. For our detection system, using an ensemble method would have various advantages, few of them are as follows:

- Using an ensemble model will continuously increase our performance of detecting videos compared to the models individually. Combining all the model's predictions for detecting videos on social media platforms with unique features would be more efficient and effective.

- As we saw those individual models like VGG 16 and inception V3 have various noises in the data during prediction, we can create a more stable prediction using an ensemble. The stability and robustness when using an ensemble method will reduce all the data noise and provide us with an aggregated prediction.

- An ensemble model will also help us better understand our detection system by capturing the model's linear and nonlinear relationship with the data.

The primary purpose of using an ensemble method is to improve the overall performance of our system. As there are various ensemble learning strategies, one such strategy is model stacking. In this project, we will use stacking as our ensemble strategy to combine all the three deep learning models we have built.

Stacked generalisation or stacking is an ensemble machine learning algorithm that combines the predictions of multiple deep learning models and learns to provide the best prediction over a test dataset. The main reason for using stacking is that it can utilise various well-performing models' capabilities on a classification task and make predictions that have better performance than any single model in the ensemble.

The architecture of a stacking model consists of base models and a meta-model. The base models are the deep learning models which we previously built and evaluated. However, a meta-model is a model that we will be using to learn how to combine the base models' predictions best. The model diagram shown in figure 5.14 will give a better idea about the stacking model used in this project.

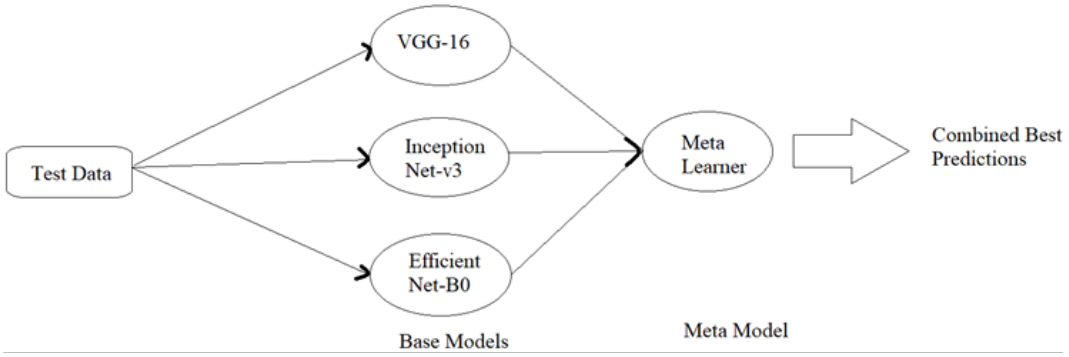

FIGURE 5.14: Stacking model architecture.

We have implemented the stacking model in a step-by-step process of coding. The steps are as follows:

Step 1: Before building our stacking model, we have used Deepfake Detection Challenge Dataset for testing. The dataset is pre-processed and converted into a multidimensional array of X (Images converted in float values) and Y (Labels consisting of 0 or 1 values) values

Step 2: We created a list called members, which store all three model's weights. We loaded the models in different variables, and we created a list of these models. Figure 5.15 shows the code snippet of the list of models stored.

```
members = [vgg16, InceptionNet, EffcientNet]
```

FIGURE 5.15: List of all three models' weight variables.

Step 3: We created a function called stacked_data(), which uses the members variable and test dataset images to make all three models predict values for the test dataset's frames. As shown in figure 5.16, the models' predictions are then stacked together in an order using a dstack() function. Finally, the function returns a list of list values with all the predictions made from the three models individually in a specific order.

```
# create stacked model input dataset as outputs from the ensemble
def stacked_data(members, inputX):
    stackX = None
    for model in members:
        # make prediction
        yhat_1 = model.predict(inputX, verbose=0)
        yhat = DT(yhat_1)
        # stack predictions into [rows, members, probabilities]
        if stackX is None:
            stackX = yhat
        else:
            stackX = dstack((stackX, yhat))
    # flatten predictions to [rows, members x probabilities]
    stackX = stackX.reshape((stackX.shape[0], stackX.shape[1]*stackX.shape[2]))
    return stackX
```

FIGURE 5.16: stacked_data() function returns a list of values consisting of all the prediction values from the three models.

Step 4: We now have all the predictions made by all the models for the test dataset. Now we must train our meta-model with the predictions made by our base models, as shown in figure 5.14. The implementation of the meta-learning model is shown in figure 5.17. We have used SVM (Support Vector Machine) as our meta learner. SVM is a linear model for classification or regression problems that creates a hyperplane line to separate the data into groups. We have used a python library to implement the SVM algorithm and learn predicted values from all three models in our function. In figure 5.18 shows the training of the stacking model stores the model's weights in the model variable. We will be using this SVM model to further provide us with the best prediction made for our test dataset.

```
# fit a model based on the outputs from the ensemble members
from sklearn.svm import SVC
def fit_stacked_model(members, inputX, inputy):
    # create dataset using ensemble
    stackedX = stacked_dataset(members, inputX)
    # fit standalone model
    model = SVC(kernel='sigmoid')#meta learner
    model.fit(stackedX, inputy)
    return model
```

FIGURE 5.17: Code snippet to train our meta-learner(SVM classifier).

```
model = fit_stacked_model(members, test_gen,test_gen.classes)
```

FIGURE 5.18: Code snippet to call the meta-learner training function.

Step 5: In this final step, use the stacked model to predict the best values for our test dataset. In figure 5.19, the code snippet shows the stacked prediction model being called with the members variable (list of three model weights), X values of the test dataset and Y values of the test dataset. We have stored the prediction values of the stacked model in the yhat variable, and we will be further evaluating the results.

```
yhat = stacked_prediction(members, model, test_gen)
```

FIGURE 5.19: A code snippet stores the predict values from the stacking model.

After predicting the values for our test dataset with our stacked models, we achieved an accuracy of 77%. The code snippet and the results are shown in figure 5.20.

```
base_score_test = metrics.accuracy_score(test_gen.classes,yhat )
print('Stacking Model Test Score ',np.round(base_score_test,3)*100)
```

```
Stacking Model Test Score  76.7
```

FIGURE 5.20: Stacking Model's accuracy.

To get a more detailed evaluation, we generated the classification report. As shown in Figure 5.21 we achieved an average precision of 72%, an average recall of 77%, and an average f1-score of 74%.

```
base_report_EN = classification_report(test_gen.classes,yhat)
print(base_report_EN)
```

	precision	recall	f1-score	support
0	0.82	0.91	0.86	4800
1	0.34	0.18	0.24	1200
accuracy			0.77	6000
macro avg	0.58	0.55	0.55	6000
weighted avg	0.72	0.77	0.74	6000

FIGURE 5.21: Stacking model's classification report.

Figure 5.22 shows a confusion matrix, showing the correctly predicted data frames from the test dataset. The stacked model detected 4385 out of 4800 deepfake video frames and 218 out of 1200 original video frames.

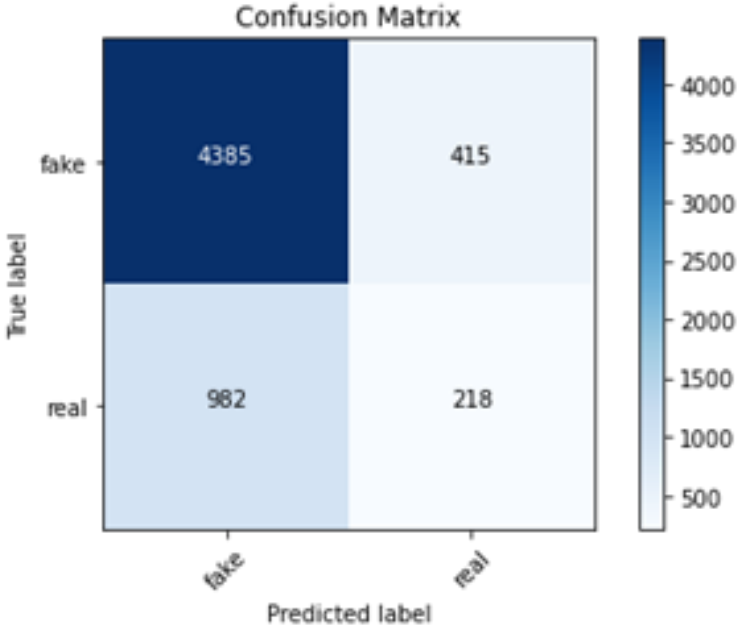

FIGURE 5.22: Stacking model's confusion matrix.

We have evaluated all three models and our stacked models, and we can compare our base models with the meta-model (stacking model). Table 5.1 shows all the evaluation metrics values of the three models and the meta-model. We can observe that our meta-model has achieved the highest accuracy amongst all the models. Apart from the accuracy, the meta-model has also achieved the highest recall, which states that this model has detected a greater number of deepfake video frames than other deep neural network

models. Therefore, these values clearly state that our implementation of a stacking model is a well-performing model that can surpass the base models.

Models	Precision	Recall	F-1 Score	Accuracy
VGG-16	72%	71%	71%	71%
InceptionNet-v3	73%	74%	74%	74%
EfficientNet-B0	73%	73%	73%	73%
Meta Model (Stacking Model)	72%	77%	74%	77%

Table 5.1: All model's evaluation results.

5.2 Evaluating Existing Systems with the Proposed System

We have now built and evaluated our deepfake detection system, but we need to find if our proposed approach is suitable for detecting deepfake videos on social media platforms. In this evaluation method, we have selected 32 videos from various social media platforms such as TikTok, Instagram, Twitter, and Facebook. In these 32 selected videos, 16 are deepfake videos that random users posted on these platforms for entertainment purposes. The other 16 videos are also from social media platforms, but they are non-deepfakes videos that have been augmented with built-in filters of the platforms. We specifically selected 16 deepfake videos from social media platforms considering certain requirements, which are as follows:

- We selected deepfake videos that were hard to detect or find any flaws. This requirement would make it more challenging for the deepfake detection systems to detect videos.
- We selected deepfake videos that were augmented by modifying the videos' brightness, contrast, or add-on filters from the platforms.
- We also focused on selecting deepfake videos which were partially deepfake. For instance, if the video has only a few frames consisting of deepfake faces, the rest of the frames are non-deepfake.

These requirements challenge the deepfake detection systems by testing their performance with different scenarios. Based on these three requirements, we selected deepfake videos over social media platforms and downloaded them after consulting the users who posted them.

The 32 videos we selected were pre-processed by converting them into multiple frames and storing them in folders of two groups (real and deepfake). Figure 5.23 shows a few frames that were pre-processed from the videos.

Now that we have selected our dataset, we will be further evaluating our ensemble model. We have also selected two open-sourced deepfake detection systems created by [44] and [46], which we will be evaluated in this section. Both the existing deepfake detection

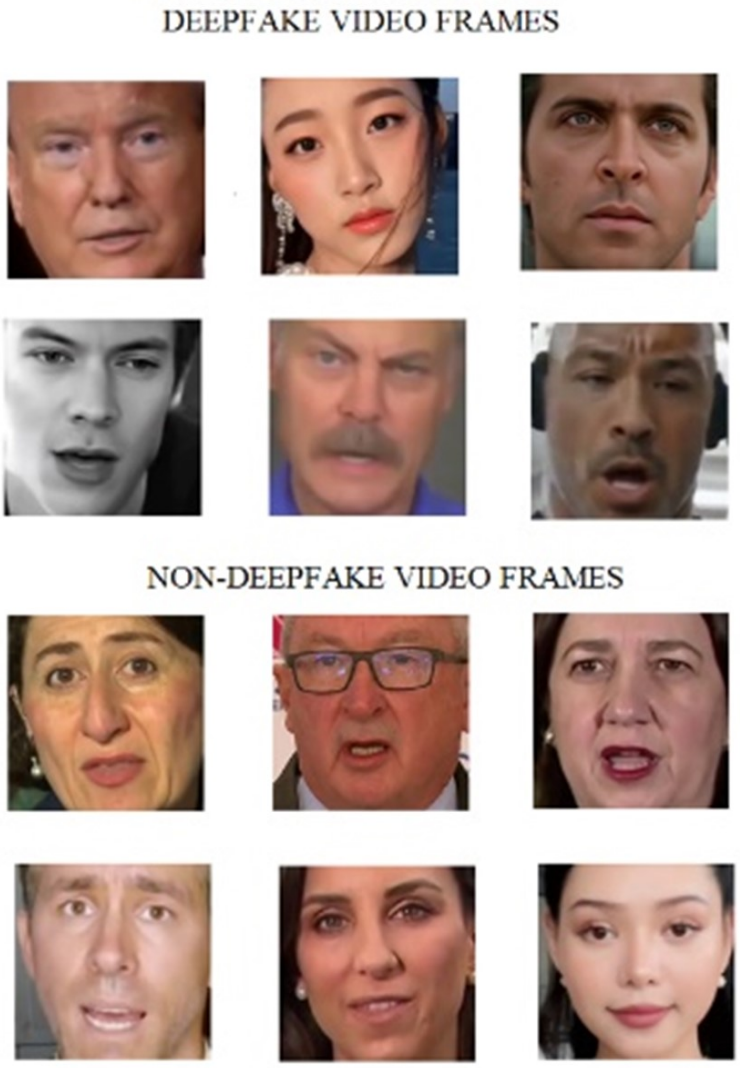

FIGURE 5.23: Sample video frames from the test dataset.

system uses an ensemble method to detect deepfake videos. This section will evaluate the two existing detection systems and our proposed system with the 32 videos which we selected previously. After conducting a detailed evaluation, we will further compare the performance of our detection system with both the existing systems.

5.2.1 Proposed System

Before evaluating the existing deepfake detection systems, we evaluated our model. To evaluate our model, we converted the test data video's frames into a multidimensional array. The array consists of X and Y values, where the X values are float array values of the frames, and the Y values are labels that consist of binary values (0 or 1). Figure

5.24 is the code snippet used to convert the frames to an array of values. There are 923 frames, of which 489 are deepfake video frames, and 434 are non-deepfake video frames.

```
test_dir = 'custom_dataset/custom_dataset_frames/'
datagen = ImageDataGenerator(rescale=1./255)
test_gen = datagen.flow_from_directory(directory=test_dir, target_size=(224,224),
                                       class_mode='binary',batch_size=10, shuffle=False)
```

```
Found 923 images belonging to 2 classes.
```

FIGURE 5.24: A code snippet to convert frames to an array of values.

As we have already created our ensemble model, as mentioned in subsection 6.4, we will be using this model to predict the video frames. Figure 5.25 is a code snippet that is used to generate predictions for the video frames values.

```
yhat = stacked_prediction(members, model, test_gen)
```

FIGURE 5.25: A code snippet stores the predict values from the stacking model.

After predicting the values for our test dataset with our ensemble model, we achieved an accuracy of 76.4%. The code snippet and the results are shown in figure 5.26.

```
base_score_test = metrics.accuracy_score(test_gen.classes,yhat )
print('Stacking Model Test Score ',np.round(base_score_test,3)*100)
```

```
Stacking Model Test Score  76.4
```

FIGURE 5.26: Stacking Model's accuracy.

To get a more detailed evaluation, we generated the classification report. As shown in Figure 5.27, we achieved an average precision of 78%, an average recall of 76% and finally, an average f1-score of 76%.

```
base_report_EN = classification_report(test_gen.classes,yhat)
print(base_report_EN)
```

	precision	recall	f1-score	support
0	0.85	0.67	0.75	489
1	0.70	0.87	0.78	434
accuracy			0.76	923
macro avg	0.78	0.77	0.76	923
weighted avg	0.78	0.76	0.76	923

FIGURE 5.27: Stacking Model's classification report.

Figure 5.28 shows a confusion matrix that displays the correctly predicted data frames from the test dataset. The stacked model detected 328 out of 489 deepfake video frames and 377 out of 434 original video frames.

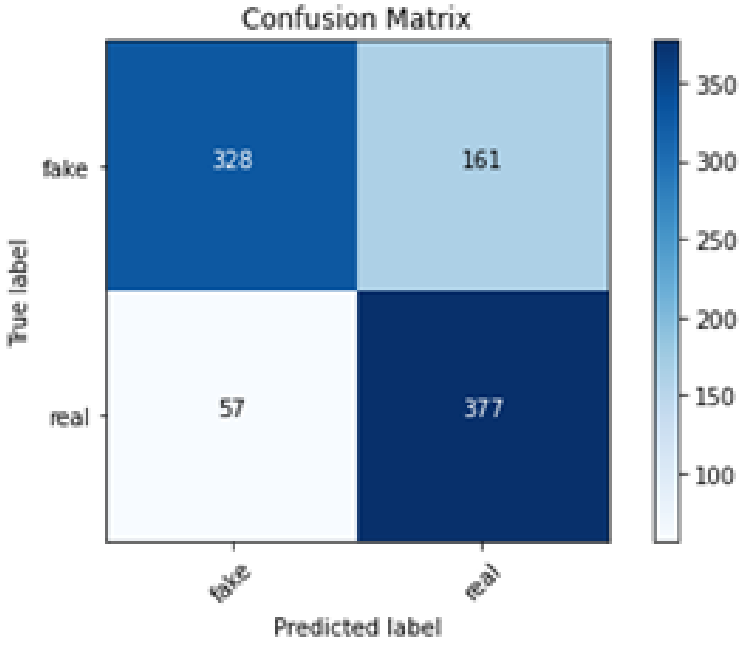

FIGURE 5.28: Stacking Model's confusion matrix.

5.2.2 Existing System-1

Over the years, various researchers have published their deepfake detection system as open to motivate other researchers to develop an improvised version of their detection system. One such researcher has published their deepfake detection system on the GitHub platform. This project was published on GitHub in the year 2020, and the

main purpose of the project was to detect deepfake videos and control the spreading of disinformation on social media [44].

However, the researcher has mentioned that the project aims to develop an ensemble CNN architecture-based model using the transfer learning method. As mentioned before, transfer learning is a method to use an already built and trained model to detect deepfake videos. The researcher has trained VGG-16 and ResNet-50 deep learning models to detect deepfake videos, and then further, a classifier is used to ensemble the models [54]. To get a better idea of the project, Figure 5.29 shows the architecture of the detection system implementation.

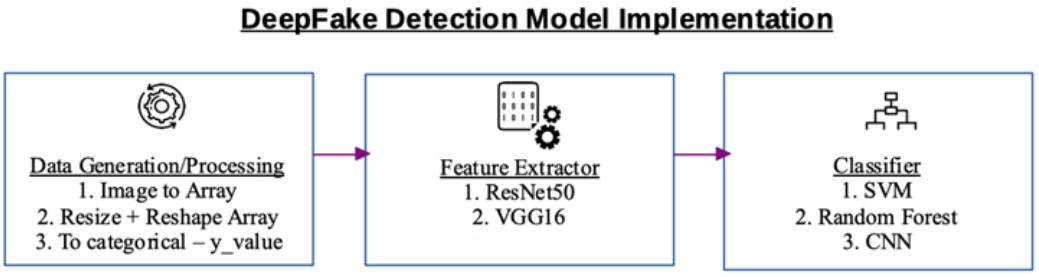

FIGURE 5.29: Existing System-1's model architecture [44].

Figure 5.29 suggests that the researchers have used three different methods of the classifier to create an ensemble. Observing the evaluation code of the project, the SVM classifier-based ensemble model has achieved the highest accuracy. The researcher also mentions that the SVM, Random Forest, and CNN classifiers achieved an accuracy of 82%, 76% and 60%, respectively.

There are various similarities and differences when comparing our deepfake detection system with this existing deepfake detection model. The similarities are as follows:

- The existing detection system is an ensemble model, which consists of VGG-16 and ResNet-50. Similarly, our model is also an ensemble model which consists of VGG-16. IncepetionNet-v3 and EfficientNet-B0.
- The existing system has used an SVM classifier to build its ensemble model, and it has achieved the highest accuracy amongst the other classifiers used. However, in our proposed model, we have also used an SVM classifier to build an ensemble model, and it has achieved an accuracy of 77%.

- Both the detection systems have trained CNN based architecture models to detect deepfake videos.

Now the difference between the existing system [44,46] and our proposed system is as follows:

- In the pre-processing stage, our proposed system has used a method to augment the video frames. However, the existing system has not used any augmentation method, but it has extracted the videos into a video frame similar to our proposed system.
- The existing system has used the transfer learning method to train its CNN based models. However, our proposed system has built deep learning models from scratch.

Considering these differences and similarities, we evaluate the existing system with the videos we selected from social media. As per the instructions given by the researcher on the GitHub page, we recreated the existing system. Since the researcher mentioned SVM classifier achieved the highest accuracy, we specifically re-created the system using an SVM classifier ensemble method. After re-creating the system, we evaluated the model. As shown in Figure 5.30, the system achieved an overall accuracy of 40.6

```
base_score_test = metrics.accuracy_score(base_y, base_y_pred)
print('Base Model Test Score ',base_score_test)
```

```
Base Model Test Score  0.40625
```

FIGURE 5.30: Existing System-1 model's accuracy.

We also evaluated the existing system with precision, recall and f1 score evaluation metrics. In figure 5.31, the code snippet and the results of the evaluation are shown. The system achieved an average precision of 55%, an average recall of 63% and an average f1 score of 58%.

```
base_report = classification_report(base_y,base_y_pred)
print(base_report)
```

	precision	recall	f1-score	support
0	0.66	0.70	0.68	335
1	0.33	0.50	0.40	177
micro avg	0.52	0.63	0.57	512
macro avg	0.49	0.60	0.54	512
weighted avg	0.55	0.63	0.58	512
samples avg	0.52	0.63	0.56	512

FIGURE 5.31: Existing System-1 model's classification report.

5.2.3 Existing System -2

Nicol'o B et al.[46] have proposed an ensemble deepfake detection model to tackle the problem of modern face manipulation techniques, also known as deepfake. In the proposed solution, the author combined the EfficientNet and XceptionNet deep neural network algorithm to detect deepfake videos. The authors have used two publicly available datasets to train their models: Celeb-DF [35] and Deepfake Detection Challenge [50] datasets.

The detection system is built using a siamese neural network architecture to detect deepfakes. The Siamese network is a neural network architecture that combines multiple models with identical architectures and trains them with very less data to produce a higher prediction rate. In the project, the siamese network contains EfficientNet-B4 and XceptionNet neural network algorithms. As mentioned previously, EfficentNet-B4 is a variant of an Efficient algorithm, which can detect videos using high computation resources. Additionally, the author has added XceptionNet neural network algorithm in the siamese network, a CNN-based algorithm that can be modified to detect high-resolution deepfake videos. These two algorithms are modified accordingly to form a Siamese network with identical parameters, weight, and configuration. Figure 5.32 give a better understanding of the architecture used in this project.

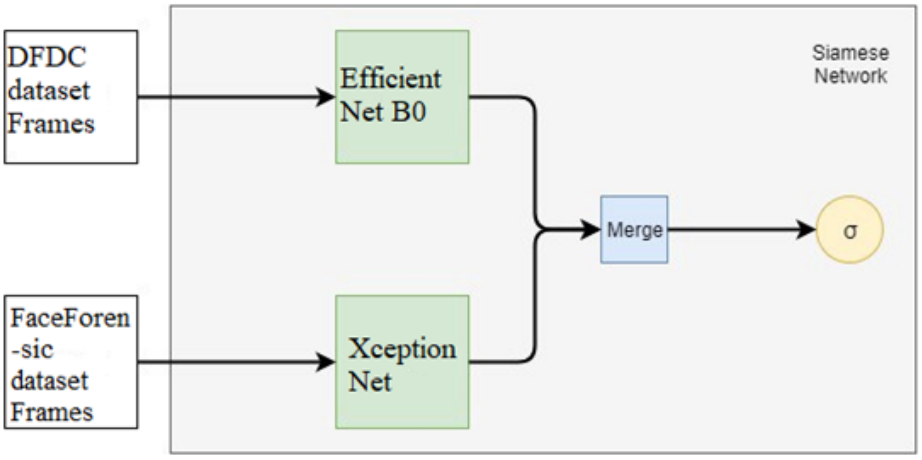

FIGURE 5.32: Existing System-2 model's architecture [1].

As shown in Figure 5.32, the training of the models in the networks uses individual dataset frames, and then the weights of the trained models are merged. The merged training weights are used as a single model to detect deepfake videos. The author concludes the research by stating that the detection system achieved an accuracy of 94%. And individually, the EfficientNet-B4 algorithm in the architecture achieved higher accuracy than the XceptionNet algorithm. However, the author also states there are a few disadvantages of using the siamese training method to detect deepfake videos, which are as follows:

- Comparing the time taken to train the model in a siamese architecture is longer than training normal CNN models. This drawback is because the Siamese architecture uses a pairwise training method, and the learning rate is low.
- The siamese model prediction values are different compared to other neural network models, as it provides the distance between the classes (real or fake). This disadvantage might be a problem when evaluating the model's predictions, and it has a high chance of providing a wrong prediction for a deepfake video.

There are still a few similarities and differences when our proposed system is compared to this existing system. The similarities are as follows:

- Our proposed system and the existing system have used a similar publicly available dataset to train the detection system.

- A similar pre-processing method of converting videos to frames and then converting them into values of arrays has been used by both the detection system. The author [46] also states that this pre-processing method is the most suited for the CNN-based algorithm.

Now the difference between the existing system [46] and our proposed system is as follows:

- The existing system has used a new Siamese neural network architecture approach to combine the neural network algorithm. Whereas in our proposed system, we have used the ensemble learning method.
- Both the detection systems have been trained in a different approach. The existing system uses a par-wise training method, and our proposed system has used a normal neural network training method.
- The prediction values in both the system are different, and they are also interpreted differently. The prediction values for the existing system are distance values to the classes (Real or Fake). In contrast, in the proposed system, the prediction values are a probability (ranging from 0 to 1).

Considering the similarities and differences in both systems, we further evaluated the existing system using the model's weights, which the author made open source. We downloaded the model's weights from the author's official GitHub repository and evaluated it with our social media videos which we have selected. Figure 5.33 shows the code snippet calculating the system's accuracy, and it achieved an overall accuracy of 50%.

```
res_score_test = metrics.accuracy_score(res_y, res_y_pred)
print('Base Model Test Score ',res_score_test)
```

```
Base Model Test Score  0.498046875
```

FIGURE 5.33: Existing System-2 model's accuracy.

We also produced a classification report which evaluated the system with precision, recall and f-1 score metrics. Figure 5.31 shows the evaluation results, and the system achieved an average precision of 57%, recall of 50% and the f1 score of 51%.

```
res_report = classification_report(res_y,res_y_pred)
print(res_report)
```

```
              precision    recall  f1-score   support

           0       0.36      0.61      0.46       177
           1       0.68      0.44      0.53       335

   micro avg       0.50      0.50      0.50       512
   macro avg       0.52      0.52      0.50       512
weighted avg       0.57      0.50      0.51       512
 samples avg       0.50      0.50      0.50       512
```

FIGURE 5.34: Existing System-2 model's classification report.

Now that we have evaluated our proposed solution and two other existing detection systems with our selected videos, we can compare the results accordingly. Table 5.2 shows all the evaluation metrics results of the existing system and our proposed system.

Observing the evaluation results in table 5.2, our proposed system has outperformed the other two models by achieving an accuracy of 77%. Our model has achieved the highest average precision, which states that our model could detect a greater number of deepfake videos frames than the other existing systems. Apart from the average precision and recall, our proposed model achieved the highest f1 score, which is a measure of the test dataset accuracy. Our model was able to outperform both the existing systems because it was trained to detect videos that are mostly uploaded on social media platforms. Another reason is that our model was trained to detect videos ranging from low resolution to high resolution, whereas both the existing systems could only detect high-resolution deepfake videos.

Systems	Precision	Recall	F-1 Score	Accuracy
Proposed System	78%	76%	76%	77%
Existing System-1 [44]	55%	63%	58%	40.6%
Existing System-2 [46]	57%	50%	51%	49.89%

TABLE 5.2: Evaluation results of the deepfake detection systems.

Chapter 6

Analysis and Discussion

In this chapter we will analyse the evaluation results thoroughly and discuss the proposed system, comparing it with the existing detection systems.

6.1 Analysis of the Results

In this section we will be analysing the two evaluation methods we conducted on our proposed detection system. The evaluation methods that we have performed are model's evaluation and evaluation of the existing system and proposed system with a social media-based dataset. We will be analysing and discussing the results of these evaluations in more detail.

6.1.1 Analysing the Models's Evaluation Results

We selected a public dataset built for a deepfake detection challenge to test all three models and our ensemble method's performance. A primary reason for selecting this public dataset for evaluation was because it consisted of videos ranging from low resolution to high resolution. One of our research objects is to detect a wide range of deepfake videos, and this dataset helped us evaluate our detection system suitably.

During our model evaluation, we evaluated VGG-16, InceptionNet-v3 and EfficeintNet-B0 using precision, recall, f1 score and accuracy as our metrics for evaluation. Table 5.1 shows the results of all three models' performance and includes our ensemble model's

performance. From the evaluation results, we made a few observations, which are as follows:

- Out of the three model's performance, we observed that InceptionNet-v3 and EfficientNet-B0 achieved the highest average precision. This achievement of the InceptionNet-v3 and EfficientNet-B0 models state that they detected the highest number of deepfake video frames out of all the deepfake video frames in the test dataset.
- We observed that the IncepetionNet-v3 models had achieved the highest average of f1 sore and recall compared to the other models. This achievement of the model means that it has outperformed the other models and correctly grouped the most number of video frames in the test dataset.

Nevertheless, there might have been few aspects that has affected the performance of all the three models:

- The test dataset is imbalanced as there were total of 1200 original video frames and 4800 deepfake video frames.
- As mentioned previously, VGG-16 and InceptionNet-v3 models have shown over-fitting that can affect the performance during evaluation.

However, to solve these issues with the model, we created an ensemble model that combined all the three model's predictions of the test dataset and then used a meta learner (Support Vector Machine Classifier) to provide us with the best results. From Table 5.1, we can observe that the meta learner has outperformed all three models by achieving an accuracy of 77% and an average f1 score of 74%. Grouping most of the video frames in the test dataset, the meta learner proved to produce the best results combing all three models.

6.1.2 Analysing the Existing System and Proposed System's Evaluation Results.

The primary purpose of this research project is to build a deepfake detection to detect deepfake video on social media platforms. Therefore, we selected a few videos over different social media platforms to test them against our detection system and two other existing systems in this evaluation. The videos were selected based on specific criteria, which is mentioned in section 6.2. Apart from testing these videos over our proposed system, we have also evaluated two existing systems [44] and [46]. This evaluation method gave us an idea about how our proposed system differs from the existing systems in detecting deepfake videos.

Table 5.2 shows the evaluation results conducted on our proposed system and both the existing systems. Even though both the existing system [44] and [46] have been built with different deep neural network models and pre-processing steps, they have achieved 40.6% and 49.8%, respectively. Nevertheless, our proposed system has outperformed the existing systems by achieving an accuracy of 77%. Our system was also able to detect most of the video frames in the dataset.

The evaluation result clearly states that both the existing systems are not suitable to detect deepfake videos on social media platforms. The main reason for this setback for the existing systems is that most of the deepfake videos on social media platforms are augmented by changing brightness, contrasts, HSV and adding built-in filters and noises to the video before uploading it. These changes to the video have affect the performance of most of the existing systems. Therefore, we have augmented our video frames of the training dataset before training the deep learning models to achieve our research objective.

To summarise all the findings from both the evaluation methods:

1. We noticed from the model's evaluation that the InceptionNet-v3 model outperformed VGG-16 and EfficientNet-B0 models by achieving an overall accuracy of 74%. Another finding from this evaluation is that an ensemble model amplifies the performance of all the models. The ensemble model has achieved higher accuracy than all the CNN models in this research project.

2. The second evaluation method determined that detecting augmented deepfake videos is challenging for the existing deepfake detection system. In contrast, our proposed system detects augmented or non-augmented deepfake videos on social media exceptionally well. From this evaluation, we also found the augmented videos impact the performance of the deepfake detection system.

The evaluation methods' findings show that we have designed a well-performing model to detect deepfake videos on social media. The proposed system answers our research question, as it helps the social media applications to malicious deepfake videos.

6.2 Discussion on Proposed System

To summarise the working of our proposed detection system, we have built the system using two main stages, pre-processing data and training. In the pre-processing stage, we followed the following steps:

- We converted the public video datasets into frames using dlib python library.
- Then we augmented the video frames for each dataset using a python package called Albumentations.

In our training stage, we trained three different CNN based neural network algorithms (VGG-16, InceptionNet-v3 and EfficientNet-B0) from scratch in two training phases, as shown in figure .

Our detection system was able to achieve a reasonable accuracy and outperform a few existing systems. However, there are various drawbacks and benefits of detecting deepfake videos for social media platforms using our detection systems. In this subsection, we will analyse the two main stages of our detection system and discuss its benefits and drawbacks.

6.2.1 Pre-Processing Stage

As mentioned previously, we have used two main steps to pre-process our data. In the first step, we used dlib library to perform face recognition for each frame in a video, and

then we stored only the frames which had faces. Researcher in their existing detection systems has done a similar method to convert videos to frames. However, only two of the researchers have used dlib library to perform face recognition on video frames.[41] states that the dlib library is a CNN based detector that has outperformed most of the existing face detectors by achieving an accuracy of 99%. However, the author also states that the dlib library has only one drawback:

- It is not very efficient when detecting faces on a real-time video. Since we have not used any real-time videos for our project, we can ignore this drawback.

However, for our deepfake detection system, the dlib library has more benefits than drawbacks. A few of the significant benefits are as follows:

- We observed that Dlib face detector was able to detect faces on video frames in all different angles. Figure 6.1 shows few of the faces detected by the dlib face detector which are in odd angles.

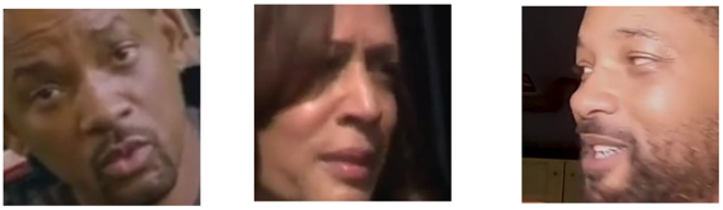

FIGURE 6.1: Sample deepfake video frames in different angle.

- Another observation we made was that the dlib face detector could detect faces and store the frame for one video in less than a second. The execution time for dlib on our CPU was very fast.

Apart from the benefits of using dlib we also observed one major drawback in our pre-processing stage. Since we are using social media videos, there are possibilities that the video can have emojis. The dlib face detector recognises the emojis as a face, as shown in figure 6.2, an example from our test dataset. Figure 6.2 shows an actual video frame with a face and emojis on top and the video frame captured by the dlib face detector. The dlib face detector has stored the cropped facial region and the emojis in a separate frame. This drawback might cause a problem for our deepfake detection system as it can wrongly group the frames that have emojis.

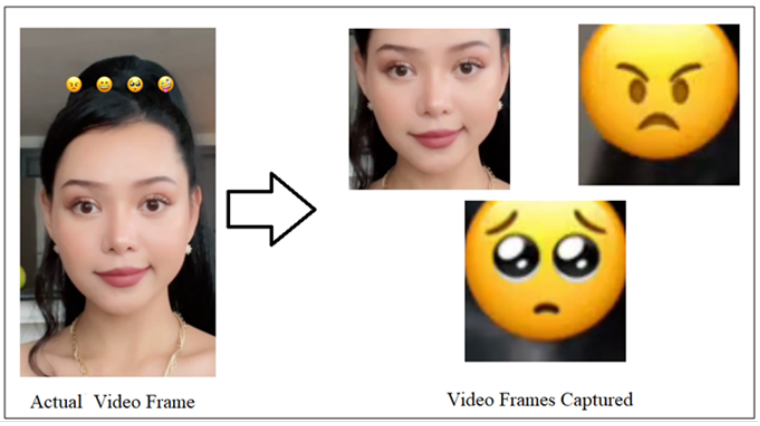

FIGURE 6.2: Dlib video frame cropping process.

Now, observing the second step in our pre-processing stage, we augment the video frames by changing their brightness, contrast, HUE, and saturation. We also added various noise to the video frames. As mentioned before, none of the researchers has used this method of pre-processing in their detection systems. However, there were no drawbacks we observed using this method in our data pre-processing. But one significant advantage of training our deep neural network models with augmenting video frames is that it would allow us to detect deepfake videos which are augmented on social media platforms. We also ensured that only 50% of the video frames in our training dataset were augmented so that our detection system would also learn to detect non-augmented deepfake video frames.

Overall, in the pre-processing stage, we only observed one drawback from the dlib face detector, which affected our performance of the proposed deepfake detection system.

6.2.2 Models Training Stage

In our training stage of the detection system, we have trained three different CNN-based deep neural network algorithms (VGG-16, InceptionNet-v3 and EfficientNet-B0). Unlike the existing deepfake detection systems, we have built the models from scratch before training them. As mentioned previously, all the well-performing deep neural network-based detection systems shown in table 5.1 have used the transfer learning method to detect deepfake videos. One major disadvantage of using transfer learning is that it can have a negative transfer. A negative transfer in machine learning happens when a trained model with an initial problem is used to train a problem unrelated to the

initial problem. Most CNN-based deep neural network algorithms are trained on a common initial problem: grouping cat and dog images. Using these models to re-train and detect deepfakes would be a major drawback as it is unrelated to the initial problem. This drawback of using the transfers learning method has made the existing deepfake detection systems outdated. These systems would find it hard to detect high resolution or flawless deepfake videos in the current time. Therefore, to avoid this drawback, we have built all three deep neural networks from scratch.

Training a deep learning model to detect deepfake videos from scratch have various benefits and drawbacks. The benefits are as follows:

- Building a model from scratch has benefited us from modifying the architecture of the models. We modified all three models in a way it can learn to detect deepfake videos with any resolutions. As mentioned previously, we also added an extra layer to prevent the model from overfitting.
- Our deep learning models can be modified anytime to detect higher quality deepfake videos. In all the three models built, we can add or remove layers and re-train it with different sets of datasets to detect high-quality deepfake videos in the future.

Apart from the benefits of building models from scratch, there are a few drawbacks, and they are as follows:

- As mentioned previously, deep learning models such as VGG-16, InceptionNet-v3 and EfficientNet-B0 have a heavy architecture with more than 50,000 parameters and multiple layers. We observed that training these models consumed a lot of time and computational resources.
- Another major drawback in building models from scratch is that it requires more data to learn and predict deepfake. Due to this drawback, there might be the occurrence of model underfitting.

These drawbacks are why our model could not outperform most existing systems that have used the transfer learning method to detect deepfake. Nevertheless, our models have performed well by achieving an average accuracy of 72% while testing it with a public dataset. Apart from the accuracy, we have also observed a few developments made by our models during the two training phases. The developments are as follows:

- All three models have dramatically increased their performance from phase 1 to phase 2 of training. As mentioned in section(training), we can observe that the validation accuracy in phase 2 training of the models has drastically increased compared to phase 1.
- In phase 2, the efficientnet-b0 model eliminated the overfitting problem that was observed during phase 1 of the training. This development was achieved by adding early stopping and model checkpoint parameters during phase 2 of training the model.

These developments while training the models have benefited us to detect high-quality deepfake videos. However, we also observed a few problems that occurred while training the models. The problems that occurred are as follows:

- During the training of VGG-16 and InceptionNet-v3 models, they showed signs of overfitting. This problem occurred due to two main reasons.
  - As there is an imbalance in the training and validation dataset, the ratio of augmented video frames are not evenly distributed.
  - The models have a complex architecture with more layers than a standard CNN model, making the models overfit.
- Another problem that we observed was that both the models have a heavy architecture. It takes more time and computational resources to train these models.

Even though the models have these two significant issues, we have moved forward to use VGG-16 and InceptionNet-v3 to detect deepfake for one primary reason. We have used an ensemble method that combines the prediction results of all three models, which has eliminated the overfitting problem in VGG-16 and InceptionNet-v3.

The ensemble method has not only eliminated overfitting in the model it has also improved the overall performance of the detection system by achieving an accuracy of 77%. As mentioned previously, the stacking model has outperformed all three model's accuracies, and it also has correctly grouped most of the video frames from the test dataset.

To summarise the limitations of our proposed system, we found two significant drawbacks that have affected our system performance. The drawbacks are as follows:

- The dlib face detector used in our pre-processing data stage has some limitations for our systems. The invalid data frame affects the models trained and also affects the evaluation cause damage to the overall performance of our system. For instance, the system will wrongly classify the data frames that do not have any facial elements, affecting the true positive values in our evaluation.

- The VGG-16 and InceptionNet-v3 are overfitting models which have affected the overall performance of the detection system. According to [FaceAugmentation], overfitting occurs in a CNN-based deepfake detection system due to the public dataset videos used to train the model. The author states that datasets such as Celeb-DF and FaceForensic have specific invalid features that cannot be removed or modified. The invalid features have caused overfitting in most of the existing CNN based deepfake detection systems.

6.2.3 Comparing Performance of Our Proposed System with the Existing Systems

Our literature review discussed the three main methods researchers have used to build a deepfake detection system, using machine learning classifier, forensic tools, and then deep learning method. Observing the performance of the systems, we can state that the deep learning-based detection systems have outperformed all the other methods. However, our review also mentioned various surveys that say deep learning deepfake detection systems are unfit to detect deepfake videos on social media.

Our proposed system takes motivation from the issues of that existing systems, as mentioned in our problem statement. Our evaluation found that our system has outperformed two of the existing systems built using the ensemble method of CNN-based algorithms. Apart from these findings, we can compare the performance of the existing systems that have used augmented data frames to train CNN algorithms.

Das et al. [44] have used augmented video frames to train the CNN model. The research suggests that the augmentation has affected the model's performance by decreasing the accuracy from 92% to 79% when evaluated with a public dataset. However, the research concludes by stating the model had overcome various issues with the existing deepfake system. The study was able to detect high-quality deepfake videos that were generated

from paid versions of deepfake software. However, the model built by the researcher was not focused on being created for social media platforms. Since the system was not open-sourced, we could not evaluate it with a standard test dataset. Nevertheless, the findings from this research have almost similar results to our proposed system.

Another similar research was published by [51] , where the researchers trained CNN models with different augmentation techniques on video frames. The study used horizontal flip, brightness and contrast changes, HSV(Hue, Saturation and Value) changes, and ISO changes augmentation techniques to train multiple CNN models to compare their performance. The augmentation techniques are similar to our proposed system augmentation technique. However, the research suggests that all the augmentation has affected the performance by drastically decreasing accuracy compared with other CNN models. These findings are similar to our proposed systems, as existing systems [44] and [46] showed 98% and 97% accuracy when evaluated with a non-augmented dataset. Our research shows that the existing systems [44] and [46] achieved 45% and 49% accuracy with the augmented dataset. These results clearly show how augmented deepfake video has affected the performance of the existing system.

The findings of research [44] and [51] resemble our proposed systems finding, suggesting that the augmentation effect deepfake detection performance. Another similarity in all our systems is that we have performed augmentation to our training dataset to make our detection system detect deepfake videos. However, in our research, we evaluated our system with real deepfake videos found on social media. We found that our system was able to detect all the deepfake videos successfully. Apart from similarities, the difference between [44], [51] and our proposed system is that our system is built with multiple deep learning models and have used an ensemble method. Our research has provided evidence the ensemble method amplifies the overall performance. There has been a lack of research on using augmented video frames to train CNN models and detect deepfake videos. However, according to [40], using augmented frames to train models is an apt solution to detect real-life deepfake videos on social media.

Chapter 7

Conclusion and Future Directions

In this research project, our goals were to solve all the issues with the existing Deep Neural Network (DNN) based deepfake detection systems and build a detection system suitable for social media platforms. After conducting a detailed investigation on the existing systems, we found that these systems have been outdated and performing low with the latest versions of deepfake videos. We also observed that all the existing detection systems lack various features, making them unsuitable for detecting deepfake videos on social media. To solve these issues in the existing systems and build a model more suitable for social media platforms, we proposed a new approach of building a dynamic deepfake detection system that can be re-trained.

We have used a new approach to pre-process our dataset to train DNN models in our proposed system. We converted the videos to frames using a face detector, which helps the models train and perform well. Another pre-processing method that we used for our dataset was augmentation. Based on previous studies, we state that augmented videos affect the performance of deepfake detection systems. The study also states that most social media videos are augmented using various filters, making it challenging for the existing systems to detect them. In our proposed approach, we overcame this issue by training our DNN models with augmented video frames. Finally, to make our system more dynamic, we built the DNN models from scratch, allowing us to modify or re-train the models with a different dataset. A dynamic deepfake detection system will enable us to improve the system by re-training the models with the latest deepfake videos, enhancing the system's performance.

We evaluated our proposed system with the deepfake detection challenge dataset [50], and our model achieved an accuracy of 77%. However, we also evaluated our proposed system with social media videos that are augmented. Along with our proposed system, we also evaluated two other existing systems [44, 46]and with the same set of social media videos, which allowed us to compare their performance with our proposed system. After a thorough evaluation, we can state that our proposed system has outperformed both the existing systems. Our proposed system also performed well with social media videos because we trained our models with augmented video frames. However, based on the evaluation, we can state that our proposed model has succeeded in detecting all kinds of deepfake videos on social media platforms. Finally, this research provides an adequate answer to our research question mentioned in chapter 3.

Nevertheless, we have a lot of opportunities to improve our models further to perform better. Our future direction is to solve the two significant flaws, as mentioned previously. The first issue is the overfitting CNN-based models in our system. Another issue is that we have difficulties eliminating invalid video frames from our dataset to train our model. To solve these issues in our future system, we can take two main steps. The steps are as follows:

- Re-training our VGG-16 and InceptionNet-v3 model with a new dataset using the bagging ensemble method. Using the bagging ensemble method, we can eliminate the overfitting in both the models and enhance the overall system's performance.
- To eliminate the invalid data frame in the training dataset, we can implement a new face detection algorithm that detects only faces and emanate emojis or any other noise found in the frames.

Implementing these two steps would undoubtedly enhance the system's performance and provide the best deepfake detection system for social media platforms. Apart from these changes, we can further improve our proposed approach by implementing a voice deepfake detection system. As mentioned in [42] all the detection systems focus on detecting facial replacement and facial re-enactment deepfake technologies. However, the survey states that training the models to classify if the voice in deepfake videos is manipulated or not would produce a well-performing detection system. Therefore, we believe that adding this extra function of detecting the voice feature of the deepfake video would enhance the performance of our detection system.

Appendix A

Appendix

The Project has been uploaded on GitHub

- https://github.com/sanathkumar5671/Deepfake_research_project

All the code and results of this research project have been uploaded on the this public space.

Bibliography

[1] N. Bonettini, E. D. Cannas, S. Mandelli, L. Bondi, P. Bestagini, and S. Tubaro, "Video face manipulation detection through ensemble of cnns," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, Conference Proceedings, pp. 5012–5019.

[2] Y. Liu and Y.-F. B. Wu, "Fned: a deep network for fake news early detection on social media," *ACM Transactions on Information Systems (TOIS)*, vol. 38, no. 3, pp. 1–33, 2020.

[3] D. G. Johnson and N. Diakopoulos, "What to do about deepfakes," *Commun. ACM*, vol. 64, no. 3, p. 33–35, 2021. [Online]. Available: https://doi.org/10.1145/3447255

[4] S. Greengard, "Will deepfakes do deep damage?" *Communications of the ACM*, vol. 63, no. 1, pp. 17–19, 2019.

[5] B. Zhu, H. Fang, Y. Sui, and L. Li, "Deepfakes for medical video de-identification: Privacy protection and diagnostic information preservation," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, Conference Proceedings, pp. 414–420.

[6] K. Shu, A. Bhattacharjee, F. Alatawi, T. H. Nazer, K. Ding, M. Karami, and H. Liu, "Combating disinformation in a social media age," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 10, no. 6, p. e1385, 2020.

[7] S. Lyu, "Deepfake detection: Current challenges and next steps," in *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. IEEE, Conference Proceedings, pp. 1–6.

[8] C. Whyte, "Deepfake news: Ai-enabled disinformation as a multi-level public policy challenge," *Journal of Cyber Policy*, vol. 5, no. 2, pp. 199–217, 2020.

[9] B. U. Mahmud and A. Sharmin, "Deep insights of deepfake technology: A review."

[10] B. Zi, M. Chang, J. Chen, X. Ma, and Y.-G. Jiang, "Wilddeepfake: A challenging real-world dataset for deepfake detection," in *Proceedings of the 28th ACM International Conference on Multimedia*, Conference Proceedings, pp. 2382–2390.

[11] F. D. Medeiros and R. B. Braga, "Fake news detection in social media: A systematic review," in *XVI Brazilian Symposium on Information Systems*, Conference Proceedings, pp. 1–8.

[12] B. Guo, Y. Ding, L. Yao, Y. Liang, and Z. Yu, "The future of false information detection on social media: New perspectives and trends," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–36, 2020.

[13] Y. Liu and S. Chawla, "Social media anomaly detection: Challenges and solutions," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Conference Proceedings, pp. 2317–2318.

[14] Y. Mirsky and W. Lee, "The creation and detection of deepfakes: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–41, 2021.

[15] deepfake, "Fakeapp," 2020. [Online]. Available: https://www.malavida.com/en/soft/fakeapp/

[16] dfaker, "Dfaker github," 2020. [Online]. Available: https://github.com/dfaker/df

[17] shaoanlu, "faceswap-gan," 2019. [Online]. Available: https://github.com/shaoanlu/faceswap-GAN

[18] deepfakes, "faceswap," 2021. [Online]. Available: https://github.com/deepfakes/faceswap

[19] iperov, "Deepfacelap," 2021. [Online]. Available: https://github.com/iperov/DeepFaceLab

[20] Jinfagang, "Faceswap-pytorch," 2020. [Online]. Available: https://github.com/jinfagang/faceswap-pytorch

[21] M. Koopman, A. M. Rodriguez, and Z. Geradts, "Detection of deepfake video manipulation," in *The 20th Irish machine vision and image processing conference (IMVIP)*, Conference Proceedings, pp. 133–136.

[22] A. Agarwal, R. Singh, M. Vatsa, and A. Noore, "Swapped! digital face presentation attack detection via weighted local magnitude pattern," in *2017 IEEE International Joint Conference on Biometrics (IJCB)*. IEEE, Conference Proceedings, pp. 659–665.

[23] X. Yang, Y. Li, and S. Lyu, "Exposing deep fakes using inconsistent head poses," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Conference Proceedings, pp. 8261–8265.

[24] P. Korshunov and S. Marcel, "Deepfakes: a new threat to face recognition," *Assessment and detection*, 2018.

[25] R. Durall, M. Keuper, F.-J. Pfreundt, and J. Keuper, "Unmasking deepfakes with simple features," *arXiv preprint arXiv:1911.00686*, 2019.

[26] Z. Akhtar and D. Dasgupta, "A comparative evaluation of local feature descriptors for deepfakes detection," in *2019 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE, Conference Proceedings, pp. 1–5.

[27] S. Agarwal, H. Farid, Y. Gu, M. He, K. Nagano, and H. Li, "Protecting world leaders against deep fakes," in *CVPR Workshops*, Conference Proceedings, pp. 38–45.

[28] Y. Zhang, L. Zheng, and V. L. Thing, "Automated face swapping and its detection," in *2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP)*. IEEE, Conference Proceedings, pp. 15–19.

[29] J. Straub, "Using subject face brightness assessment to detect 'deep fakes'(conference presentation)," in *Real-Time Image Processing and Deep Learning 2019*, vol. 10996. International Society for Optics and Photonics, Conference Proceedings, p. 109960H.

[30] F. Marra, D. Gragnaniello, L. Verdoliva, and G. Poggi, "Do gans leave artificial fingerprints?" in *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE, Conference Proceedings, pp. 506–511.

[31] H. Mo, B. Chen, and W. Luo, "Fake faces identification via convolutional neural network," in *Proceedings of the 6th ACM Workshop on Information Hiding and Multimedia Security*, Conference Proceedings, pp. 43–47.

[32] H. H. Nguyen, J. Yamagishi, and I. Echizen, "Capsule-forensics: Using capsule networks to detect forged images and videos," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Conference Proceedings, pp. 2307–2311.

[33] M. Du, S. Pentyala, Y. Li, and X. Hu, "Towards generalizable forgery detection with locality-aware autoencoder," *arXiv preprint arXiv:1909.05999*, 2019.

[34] T. Fernando, C. Fookes, S. Denman, and S. Sridharan, "Exploiting human social cognition for the detection of fake and fraudulent faces via memory networks," *arXiv preprint arXiv:1911.07844*, 2019.

[35] J. Li, T. Shen, W. Zhang, H. Ren, D. Zeng, and T. Mei, "Zooming into face forensics: A pixel-level analysis," *arXiv preprint arXiv:1912.05790*, 2019.

[36] N. Yu, L. S. Davis, and M. Fritz, "Attributing fake images to gans: Learning and analyzing gan fingerprints," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, Conference Proceedings, pp. 7556–7566.

[37] M. S. Rana and A. H. Sung, "Deepfakestack: A deep ensemble-based learning technique for deepfake detection," in *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, Conference Proceedings, pp. 70–75.

[38] A. Mitra, S. P. Mohanty, P. Corcoran, and E. Kougianos, "A machine learning based approach for deepfake detection in social media through key video frame extraction," *SN Computer Science*, vol. 2, no. 2, pp. 1–18, 2021.

[39] M. Masood, M. Nawaz, K. M. Malik, A. Javed, and A. Irtaza, "Deepfakes generation and detection: State-of-the-art, open challenges, countermeasures, and way forward," *arXiv preprint arXiv:2103.00484*, 2021.

[40] D. Gamage, J. Chen, and K. Sasahara, "The emergence of deepfakes and its societal implications: A systematic review."

[41] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake news detection on social media: A data mining perspective," *ACM SIGKDD explorations newsletter*, vol. 19, no. 1, pp. 22–36, 2017.

[42] B. U. Mahmud and A. Sharmin, "Deep insights of deepfake technology: A review," *arXiv preprint arXiv:2105.00192*, 2021.

[43] S. Lyu, "Deepfake detection: Current challenges and next steps," in *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. IEEE, Conference Proceedings, pp. 1–6.

[44] S. Das, S. Seferbekov, A. Datta, M. Islam, and M. Amin, "Towards solving the deepfake problem: An analysis on improving deepfake detection using dynamic face augmentation," *arXiv preprint arXiv:2102.09603*, 2021.

[45] Y. Li, C. Zhang, P. Sun, H. Qi, and S. Lyu, "Deepfake-o-meter: An open platform for deepfake detection," *arXiv preprint arXiv:2103.02018*, 2021.

[46] C. Leibowicz, S. McGregor, and A. Ovadya, "The deepfake detection dilemma: A multistakeholder exploration of adversarial dynamics in synthetic media," *arXiv preprint arXiv:2102.06109*, 2021.

[47] I. Marketing, "Social media video specs for 2021 — always up-to-date guide," 2021. [Online]. Available: https://influencermarketinghub.com/social-media-video-specs/

[48] Y. Li, "celeb-deepfakeforensics," 2020. [Online]. Available: https://github.com/yuezunli/celeb-deepfakeforensics

[49] Google, "Contributing data to deepfake detection research," 2019. [Online]. Available: https://ai.googleblog.com/2019/09/contributing-data-to-deepfake-detection.html

[50] Kaggle, "Deepfake detection challenge," 2020. [Online]. Available: https://www.kaggle.com/c/deepfake-detection-challenge/data

[51] L. Bondi, E. D. Cannas, P. Bestagini, and S. Tubaro, "Training strategies and data augmentations in cnn-based deepfake video detection," *arXiv preprint arXiv:2011.07792*, 2020.

[52] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Conference Proceedings, pp. 2818–2826.

[53] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, Conference Proceedings, pp. 6105–6114.

[54] Z. Mehta, "Deepfake detection using deep learning methods for synthesized videos," 2020, Web Page. [Online]. Available: https://github.com/zeel97/DeepFakeDetection