

**DEEPFAKE DETECTION SYSTEM WITH CONVOLUTIONAL
NEURAL NETWORK (CNN)**

BY

CHARLES ITA EKANEM

201203060

**A FINAL YEAR PROJECT SUBMITTED TO THE DEPARTMENT
OF COMPUTER ENGINEERING**

**IN PARTIAL FULFILLMENT FOR THE AWARD OF
BACHELOR OF ENGINEERING (B.ENG.) IN COMPUTER
ENGINEERING**

FACULTY OF ENGINEERING

NILE UNIVERSITY OF NIGERIA, ABUJA

JUNE, 2025

APPROVAL

The project entitled "**DEEPMODEL DETECTION SYSTEM WITH CONVOLUTIONAL NEURAL NETWORK**" by **CHARLES ITA EKANEM** whose ID NUMBER is **201203060** meets the requirements for the award of the Degree of Bachelor of Engineering (B.Eng.) in Computer Engineering and is approved for its contribution to knowledge and literary representation.

Approved:

Dr. Emmanuel Ali

Project Supervisor

Accepted:

Assoc. Prof. Nyangwarimam O. Ali

Head of Department

Prof. Petrus Nzerem

Dean of the Faculty

Dr. Francis Emmanuel

External Supervisor

Date

DECLARATION

I, **EKANEM CHARLES ITA**, hereby declare that this project titled **“DEEPMODEL DETECTION SYSTEM WITH CONVOLUTIONAL NEURAL NETWORK (CNN)”** has been carried out by me under the supervision of **Dr Emmanuel Ali**. It is presented for the award of the degree of Bachelor of Engineering (B. Eng.) in Computer Engineering. All the sources of externally acquired information and data are acknowledged by means of reference.

Signature

Date

DEDICATION

This project is dedicated with sincere gratitude to Abasi Enyong for his guidance and protection throughout my program.

ACKNOWLEDGEMENTS

First and foremost, my debt of gratitude is extended to the supervisor of my research, Dr Emmanuel Ali for his excellent guidance and continued patience at every stage of my study. I have really benefited from his inspiration, knowledge, experience and persistence.

I would also like to express my sincere gratitude to the Head of Department, Dr Nyangwarimam Obadiah Ali, for his guidance and support. I would also like to express my gratitude to the lecturers of Computer Engineering, Nile University of Nigeria for their help and guidance.

Finally, I would like to appreciate my family, friends and course mates.

ABSTRACT

The rapid advancement of deepfake generation techniques have outpaced the development of effective detection methods which creates a gap in the ability to accurately identify and avoid the risks associated with this technology. Current detection systems often rely on traditional image analysis techniques that are insufficient against the evolving sophistication of deepfake algorithms. As a result, there is an urgent need for innovative solutions that utilise modern machine learning approaches. This project presents a robust deepfake detection system utilizing CNNs, which have demonstrated exceptional performance in image and video analysis tasks. This project presents a deepfake detection pipeline based on InceptionResNetV1, pre-trained on VGGFace2 and fine-tuned on a balanced dataset of 206,348 preprocessed facial images extracted from the Celeb-DF, Faceforensics and Openforensics datasets with a stratified 70/30 split ensuring fair training and validation distribution. The model was trained on a CPU for 20 epochs achieving a final validation accuracy of 97.15% and a validation loss of 0.0671, indicating strong generalization with minimal overfitting. The study highlights the effectiveness of using computationally efficient training setups alongside proper data preparation for high accuracy deepfake detection. Future work will explore video-level analysis, ensemble learning and real-time deployment strategies to expand the model's applicability.

TABLE OF CONTENTS

TITLE PAGE	i
APPROVAL	ii
DECLARATION	iii
DEDICATION	iv
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	xii
LIST OF TABLES	xv
CHAPTER ONE	1
INTRODUCTION	1
1.1 BACKGROUND OF THE STUDY	1
1.2 STATEMENT OF THE PROBLEM	1
1.3 AIM AND OBJECTIVES	3
1.4 PURPOSE OF THE STUDY	3
1.5 SIGNIFICANCE OF THE STUDY	3
1.6 LIMITATIONS OF THE STUDY	4
CHAPTER TWO	5
LITERATURE REVIEW	5

2.1 INTRODUCTION	5
2.2 DEFINITION OF RELATED CONCEPTS	5
2.3 HISTORICAL OVERVIEW	7
2.4 CONCEPT OF DEEP LEARNING	11
2.5 STATISTICAL ANALYSIS AND FIGURES	12
2.6 REVIEW OF PREVIOUS RESEARCH WORK	12
CHAPTER THREE	27
METHODOLOGY	27
3.1 INTRODUCTION	27
3.2 TECHNIQUES EMPLOYED	27
3.2.1 DATA COLLECTION	27
3.2.2 DATA PREPROCESSING	28
3.2.3 CONVOLUTIONAL NEURAL NETWORK	28
3.2.4 TRAINING	31
3.2.5 TESTING	32
3.2.6 DEPLOYMENT	33
3.3 RESEARCH INSTRUMENTS USED	33
3.3.1 SOFTWARE AND TOOLS	33
3.3.1.1 DEEP LEARNING FRAMEWORK	33
3.3.1.2 DATASET REPOSITORIES	34

3.3.1.3 EVALUATION TOOL	34
3.3.1.4 INTEGRATED DEVELOPMENT ENVIRONMENTS	34
3.3.1.5 DATA VISUALIZATION TOOL	34
3.3.1.6 VERSION CONTROL SOFTWARE	34
3.3.1.7 CLOUD PLATFORM	34
3.3.2 DIAGRAMS AND VISUALS	35
3.3.2.1 BLOCK DIAGRAM FOR DETECTION SYSTEM	35
3.3.2.2 FLOWCHART FOR DETECTION SYSTEM	36
3.3.2.3 RISK MANAGEMENT FOR DEEPFAKE DETECTION	38
CHAPTER FOUR	39
IMPLEMENTATION AND RESULTS	39
4.1 INTRODUCTION	39
4.2 DATASET AND MODEL DEVELOPMENT	39
4.2.1 DATASET DESCRIPTION	39
4.2.1.1 CELEB DF V2	39
4.2.1.2 FACEFORENSICS ++	39
4.2.1.3 OPENFORENSICS	39
4.2.2 DATA PREPROCESSING	43
4.2.3 TRAINING MODEL	43
4.2.4 MODEL ARCHITECTURE	44

4.3 SYSTEM EXECUTION PIPELINE AND DIRECTORY STRUCTURE	45
4.3.1 SYSTEM EXECUTION PIPELINE	46
4.3.2 DIRECTORY STRUCTURE	46
4.4 EVALUATION METRICS	47
4.4.1 LOSS FUNCTION	47
4.4.2 ACCURACY	47
4.4.3 CONFUSION MATRIX	48
4.4.4 CLASSIFICATION REPORT	48
4.5 TRAINING RESULTS	48
4.5.1 LOSS VS EPOCH	50
4.5.2 ACCURACY VS EPOCH	50
4.5.3 CONFUSION MATRIX	51
4.5.4 CLASSIFICATION REPORT	52
4.6 DEPLOYMENT AND TESTING	52
4.6.1 MODEL DEPLOYMENT WITH STREAMLIT	52
4.6.2 MODEL TESTING AND REVIEW	53
4.6.2.1 STRENGTHS OF THE DEPLOYMENT	55
4.6.2.2 WEAKNESSES OF THE DEPLOYMENT	56
4.6.2.3 MODEL MISCLASSIFICATION	56
4.7 ANALYSIS OF RESULTS	57

CHAPTER FIVE	58
CONCLUSION AND RECOMMENDATION	58
5.1 INTRODUCTION	58
5.2 FINDINGS	58
5.3 SCOPE OF FINDINGS	58
5.4 CONTRIBUTION TO KNOWLEDGE	59
5.5 RECOMMENDATION	60
5.6 FUTURE IMPROVEMENTS AND DIRECTION	61
5.7 CONCLUSION	62
REFERENCES	63
APPENDIX A: DATASET SAMPLES	70
APPENDIX B: TRAINING SCRIPT SCREENSHOTS	73
APPENDIX C: MODEL ARCHITECTURE	79

LIST OF FIGURES

Figure 2.1 Historical Evolution of Deepfake	10
Figure 2.2 The layers of a deep neural network	11
Figure 2.3 GAN Architecture	14
Figure 3.1 Basic CNN Architecture	28
Figure 3.2 Expanded CNN Architecture	30
Figure 3.3 Proposed CNN Architecture with Binary Classification Output ...	31
Figure 3.4 Obtainable post training models	32
Figure 3.5 Block diagram for detection system	35
Figure 3.6 Flowchart for development process of detection system	36
Figure 3.7 Flowchart for detection process of developed model	37
Figure 4.1 Faces Class Distribution post face extraction (Bar Chart)	40
Figure 4.2 Faces Class Distribution post face extraction (Pie Chart)	41
Figure 4.3 Terminal output after balancefakeimages.py	41
Figure 4.4 Bar chart showing the balanced distribution of dataset classes	42
Figure 4.5 Pie chart showing the balanced distribution of dataset classes	42
Figure 4.6 Training and Validation distribution	43
Figure 4.7 Inception-ResNet V1 structure	45
Figure 4.8 Loss vs Epoch Plot	50
Figure 4.9 Accuracy vs Epoch Plot	50

Figure 4.10 Confusion Matrix	51
Figure 4.11 GitHub repository	53
Figure 4.12 Model deployment on Streamlit App	53
Figure 4.13 Model Testing on Real Image	54
Figure 4.14 Model Testing on distorted Image	54
Figure 4.15 Model Testing on Real Image (Openforensics data)	55
Figure 4.16 Model Testing on Fake Image (Openforensics data)	55
Figure A.1 Real images sample from dataset	70
Figure A.2 Fake images sample from dataset	70
Figure A.3 Real videos	71
Figure A.4 Fake videos	71
Figure A.5 OpenForensics real test images	72
Figure A.6 OpenForensics fake test images	72
Figure B.1 Library imports	73
Figure B.2 Configuration	74
Figure B.3 Data loading	74
Figure B.4 Training progress management	75
Figure B.5 Training process	75
Figure B.6 Training process (continued)	76
Figure B.7 Model evaluation and plotting	77

Figure B.8 Script execution	77
Figure C.1 Script for displaying InceptionResNet V1 model architecture	79
Figure C.2 Script for displaying layer-wise model summary of InceptionResNetV1	79
Figure C.3 InceptionResNet V1 model parameters	80

LIST OF TABLES

Table 2.1 Literature Review	17
Table 3.1 Risk management for Deepfake Detection system development	38
Table 4.1 Training Results	48
Table 4.2 Classification Report	52

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

The creation of Deepfake media include the training of a Deep Learning (DL) algorithm with datasets of media to be manipulated. This is the source of the term where the name of the technique used, Deep Learning, is combined with the material generated (Fake). Due to consistent training of the material, a convincing impersonation of the subject media is developed thus creating a threat of malicious use. Deepfake algorithms require large datasets and a neural network made of interconnected nodes in a layered structure [1]. A generative neural network works to generate fake content while the discriminator works to counter the generated content and the generative network keeps trying to generate more content thus better training the model [2]. Both networks are jointly referred to as Generative Adversarial Networks (GANs) [3].

The growth of complex generative techniques has made identification more difficult as a task and as such, models must be developed to aid in the detection and identification of Deepfakes. This entails the training of Deep Learning models with large datasets of real and fake media so as to detect anomalies present in fake material which may not exist in real material. The anomalies may come in several forms such as facial expressions, inhuman motions, irregular lighting and shading of figures etc.

1.2 Statement of the Problem

The advancement of Artificial Intelligence has resulted in the complexity and increased development of Deepfakes usually for malicious purposes. In recent years, the escalation of Deepfake technology has emerged as a major risk to the authenticity of media. Deepfakes, which are hyper-realistic synthetic media generated using artificial intelligence, have the potential to mislead audiences,

alter public opinion and destroy confidence in visual content. This technology has been utilized in various domains, including politics, entertainment, and social media leading to serious ethical and security concerns. For instance, deepfake videos have been used to create false narratives during elections, resulting in misinformation that can influence voter actions and disrupt democratic processes. The rapid advancement of deepfake generation techniques has outpaced the development of effective detection methods, creating a gap in the ability to accurately identify and avoid the risks associated with this technology. Current detection systems often rely on traditional image analysis techniques that are insufficient against the evolving sophistication of deepfake algorithms. As a result, there is an urgent need for innovative solutions that utilise modern machine learning techniques and algorithms, particularly Convolutional Neural Networks (CNNs), to enhance the accuracy and reliability of deepfake detection.

Moreover, the implications of undetected deepfakes extend beyond individual cases; they pose a broader threat to societal trust in media and information. As deepfakes become more prevalent, the potential for their misuse increases, leading to a climate of scepticism where genuine media is questioned and misinformation proliferates. This decline in trust can lead to significant and widespread effects, affecting everything from personal relationships to national security and evidence in courts of law.

This research project aims to develop a robust deepfake detection system utilizing CNNs, which have demonstrated exceptional performance in image and video analysis tasks. By focusing on this pressing issue, the research project seeks to contribute to the development of effective detection models and techniques that can be deployed in real-world applications, thereby safeguarding the integrity of media. The significance of this project lies not only in its technical contributions but also in its potential to restore public trust in visual and audio-visual content and combat the growing threat of misinformation in our increasingly digital world.

1.3 Aim and Objectives

The aim of this research is to design and implement a Deepfake Detection System utilising the image recognition and processing capabilities of Convolutional Neural Networks.

Based on the problem statement identified above, the following objectives have been outlined:

- 1 To design a diverse dataset of deepfake videos and images for model training and testing.
- 2 To design and train a suitable CNN model on prepared training and validation datasets.
- 3 To test the working of the developed detection model with various new datasets.

1.4 Purpose of the Study

The purpose of this study is to develop an effective deepfake detection system using CNNs, thereby addressing the growing challenges posed by deepfake technology, enhancing public trust in digital media and contributing to the advancement of research and ethical consideration in the field of artificial intelligence.

1.5 Significance of the Study

The significance of this research project on "Deepfake Detection with CNN" addresses several critical issues in today's digital landscape. One of the primary concerns is the proliferation of misinformation. Deepfakes pose a significant threat to the integrity of information, particularly in the realms of news media, politics, and social interactions [6]. By developing an effective detection model, this study contributes to the mitigation of misleading or harmful content, thereby fostering a more informed public.

Moreover, as deepfake technology becomes increasingly sophisticated, the potential for misuse in areas such as fraud, identity theft, and cybercrime [7] will escalate. This research enhances security measures across various sectors, including finance, law enforcement, and online platforms. The research also advances the fields of computer vision and machine learning by exploring the capabilities of Convolutional Neural Networks (CNNs) in detecting manipulated media. This can lead to further technological advancements and innovations in artificial intelligence, with applications extending beyond Deepfake detection to other areas of artificial intelligence.

1.6 Limitations of the Study

Although the study on "Deepfake Detection with CNN" holds significant promise, it also faces several limitations. Firstly, one of the primary challenges is the rapidly evolving nature of Deepfake generative technology. As detection methods improve, the techniques used to create Deepfakes improve as well. This disparity can render existing detection models less effective over time, necessitating continuous updates and retraining of the models to keep pace with likewise evolving Deepfake generation methods. Secondly, the quality and diversity of the dataset used for training and testing the CNN model can significantly impact the results particularly, if the model may struggle to generalize unseen data [8]. This limitation may cause uneven performance, where the model is effective at identifying certain types of Deepfakes but struggles to detect others, especially those that are quite different from the examples it was trained on. Furthermore, the performance metrics used to evaluate the CNN model may not capture all aspects of its effectiveness. Metrics like accuracy, precision, and recall deliver essential insights; however, they may not entirely represent a model's effectiveness in real-world contexts, where aspects like user behaviour and contextual nuances are crucial in identifying Deepfakes [9].

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter focuses on the concept of deepfake technology and the subsequent need for effective detection systems. Continuous advancements in artificial intelligence necessitate the need for review of existing literature on this topic. This chapter as such will explore the significant literary contributions to trace a path of evolution in the technology of deep learning inspired material termed deepfake. Furthermore, it will analyse the efforts made to create strong detection systems, highlighting various methodologies and approaches. The literature review will act as a guiding framework to enhance understanding of the theoretical principles involved.

2.2 Definition of Related Concepts

Below are concise definitions of concepts considered related and important to this body of work:

- 1 Deepfake: Deepfake refers to synthetic material generated, modified or manipulated using artificial intelligence (AI) techniques, particularly deep learning [1]. This technology can create realistic-looking videos, audio, or images often serving malicious purposes.
- 2 Convolutional Neural Network (CNN): A Convolutional Neural Network is a class of deep learning algorithms primarily used for processing data, such as images, through feature extraction as grid-like matrices [8]. CNNs utilize convolutional layers to acquire spatial hierarchies of features from input images, which enhances their effectiveness in image classification and recognition tasks.

- 3 Generative Adversarial Network (GAN): A GAN is a class of machine learning frameworks designed to generate new data instances that resemble a given dataset [14].
- 4 Image Processing: Image processing refers to the manipulation and analysis of images to improve their quality or extract valuable information. In the context of deepfake detection, image processing can be employed to prepare images prior to inputting into a CNN model [42].
- 5 Feature Extraction: Feature extraction is the process of identifying and isolating relevant features from raw data [23]. In deepfake detection, CNNs will automatically extract features from images that can help distinguish between real and manipulated content.
- 6 Training Dataset: A training dataset is a collection of data used to train a machine learning model [2]. For deepfake detection, this dataset would typically include both real and deepfake images or videos, allowing the CNN to identify and learn the differences between them.
- 7 Testing Dataset: A testing dataset is a set of data used to assess the final performance of a trained model [2]. It provides an evaluation of the model's ability to detect deepfakes on previously unseen and new data.
- 8 Generalization: Generalization refers to a model's ability to perform well on new, unseen data that was not part of the training dataset [8].
- 9 Overfitting: Overfitting refers to when a trained model learns the training data too well, including its noise, resulting in poor generalization to new unseen data [8].
- 10 Hyperparameters: Hyperparameters are the configuration settings set prior to training used to control the training process of a machine learning model [8].
- 11 Accuracy: Accuracy is a metric used to evaluate the performance of a trained and tested model. It is defined as the ratio of correctly predicted instances to the total instances in the dataset. In deepfake detection,

accuracy will indicate how well the model can distinguish between real and fake content [41].

- 12 Precision and Recall: Precision is the ratio of true positive predictions to the total predicted positives. It measures the accuracy of the positive predictions made by the model [41]. Recall refers to the ratio of true positive predictions to the total actual positives. It measures the model's ability to identify all relevant instances [41].
- 13 Confusion Matrix: A confusion matrix is a table used to evaluate the performance of a classification model [41].
- 14 Transfer Learning: This is a technique in machine learning where data knowledge from one task is used to improve performance on another related task [29].
- 15 Adversarial Training: This involves training a model with adversarial input (data designed to fool the model) [3]. By exposing the model to these data examples during training, it learns to better handle variations ultimately enhancing its performance and reliability in real-world scenarios.
- 16 Real-time Detection: This refers to the capability of a system to analyse and classify data as it is being captured or streamed, providing immediate feedback on whether the material is real or a deepfake [29].
- 17 Ethical Considerations: Ethical considerations in deepfake detection involve the implications of using AI technologies, including privacy concerns, the potential for misuse and the societal impact of deepfakes [6].

2.3 Historical Overview

The history of digital media manipulation dates back to the late 20th century with the introduction of digital editing software like Adobe Photoshop (released in 1988) and graphic programs such as MacPaint (released in 1984) [10]. These applications transformed the image manipulation landscape by offering a

comprehensive array of editing tools. Over time, advancements in technology made image manipulation software increasingly accessible. Additionally, the rise of social media as a global communication platform has significantly influenced the growth of image manipulation with software like Instagram and Snapchat.

Another significant historical contributor to the emergence of digital manipulation is Computer-Generated Imagery (CGI) [11]. CGI refers to the creation of both still and animated visual content through computer software and has been extensively utilized in the film industry since the 1990s. Much like the data analysis and replication techniques observed in deepfakes, CGI has played a crucial role in establishing the groundwork for contemporary deepfake technology. As CGI continues to advance, the consequences of its application in generating synthetic media will be an important subject of discussion in the fields of technology, ethics, and media literacy.

In 1997 [12], a groundbreaking program called Video Rewrite, developed by Christoph Bregler and his team, revolutionized video editing by synchronizing a person's lip movements with a new audio track. This pioneering system utilized machine learning to establish a connection between the sounds and facial expressions of the video's subject, effectively reanimating their face to match the new audio. Initially designed for movie dubbing, the program allowed filmmakers to sync an actor's lip movements with a new soundtrack, opening up new possibilities for post-production editing. The Video Rewrite program can be considered a pioneer deepfake technology [13] as it allowed for the development of advanced facial reanimation and lip-syncing techniques that are now commonly used in deepfake creation.

In June 2014, Ian Goodfellow and his team introduced Generative Adversarial Networks (GANs) [14]. This innovative model consists of two components: the

Generator and the Discriminator, which compete with one another to enhance their respective performances. The Generator is responsible for producing synthetic media, which has also been recognized as deepfakes.

In 2016, a team of researchers introduced Face2Face, a real-time facial reenactment system [15]. This technology allowed for the manipulation of facial expressions in video recordings, enabling the creation of highly realistic and convincing deepfakes. Face2Face marks a significant milestone in the field, building on the foundation laid by Video Rewrite and Generative Adversarial Networks, which paved the way for the creation of more advanced deepfake tools. Face2Face represents a crucial step forward in the field, leveraging the groundwork established by Video Rewrite and Generative Adversarial Networks to enable the development of more complex and sophisticated deepfake creation technologies.

The 2017 research paper "Synthesizing Obama: Learning Lip Sync from Audio" by the University of Washington team, led by Supasorn Suwajanakorn, was a pivotal moment in the development of deepfakes [16]. By successfully creating realistic videos of former USA President Obama speaking, the project showcased a major breakthrough in lip-sync technology. This achievement had notable effect, accelerating the advancement of deepfake technology and paving the way for the creation of increasingly sophisticated and realistic synthetic media.

The term "deepfake" finally originated from a Reddit user known as u/deepfakes, who launched a subreddit called r/Deepfakes on November 2, 2017 [17]. Initially, the community focused on using deep learning algorithms to create and share manipulated videos, often involving face-swapping female celebrities into explicit content. However, after Reddit banned the r/Deepfakes subreddit on February 7, 2018 [17], the concept of deepfakes spread rapidly, leading to the

emergence of new forums, tools, and services that have further popularized and commercialized this technology.

This historical examination of deepfake technology reveals a significant progression from basic image editing tools to more sophisticated methods, including the emergence of computer-generated imagery (CGI), the development of the Video Rewrite program and the introduction of Generative Adversarial Networks (GANs). This evolution continued with the creation of Face2Face and the influential 2017 research paper on lip-syncing audio with synthesized video, ultimately leading to the rise and subsequent decline of the r/Deepfakes subreddit.

Figure 2.1 provides a visual summary of the evolution of media manipulation, beginning with basic computer-assisted editing tools in the 1980s, followed by the rise of CGIs in the 1990s and the notable release of Video Rewrite in 1997. It also highlights the rapid expansion of manipulation technologies in the 21st century.

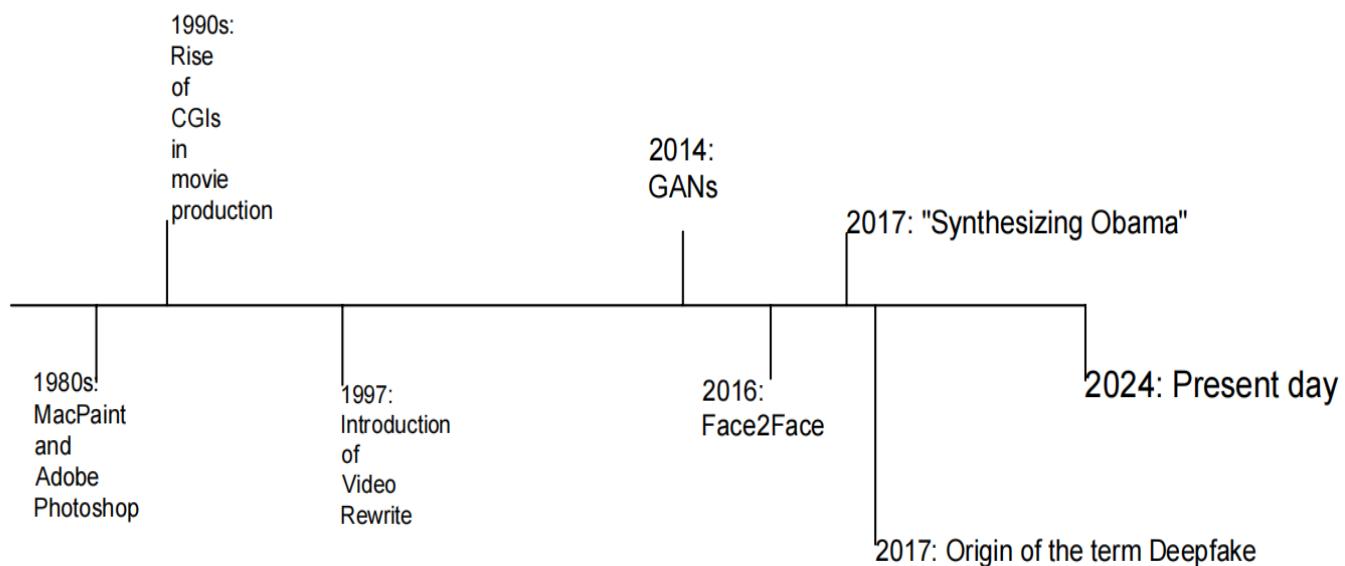


Figure 2.1 Historical Evolution of Deepfake

2.4 Concept of Deep Learning

Deep Learning is a subset of artificial intelligence that teaches computers to process data in a pattern that mimics the human brain. It utilises artificial neural networks with multiple inner layers to learn and extract information [18]. It simulates the processing of the brain by making use of neural networks, starting with a single layer that can accurately predict a solution, and move to more complex and hidden layers and interconnected nodes that can optimise and improve accuracy. It is used in several fields such as self-driving cars, large language models, credit card fraud detection, and brain computer interface applications, among many others [19]. **Figure 2.2** shows the layers of a deep neural network.

Backpropagation which is the most important part of CNN is an algorithm which is used heavily by deep learning models to calculate the errors in predictions and adjust the weights and biases based on the calculated function by moving in a backwards manner through the layers while training the model [20]. The combination of forward and backward propagation allows the deep learning model to verify its accuracy and correct its errors [18].

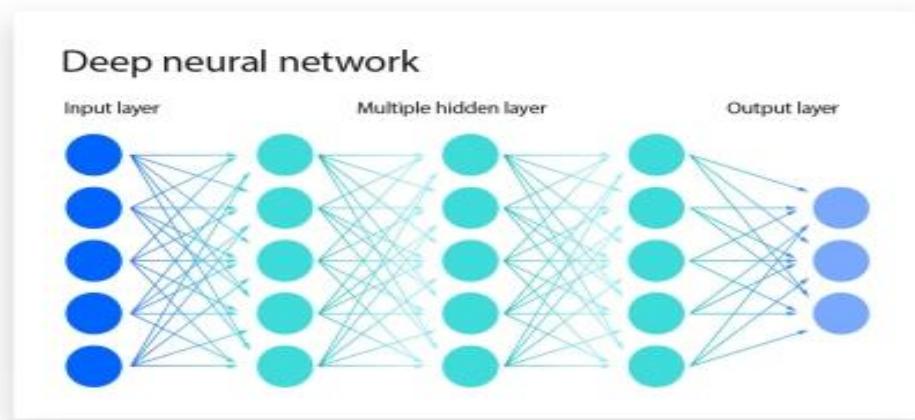


Figure 2.2 The layers of a deep neural network [19]

2.5 Statistical analysis and figures

Regardless of the advantages of deepfake technology, the risk of deepfakes have emerged as a significant threat, leading to notable financial losses and challenges in visual identification. Stated below are several deepfake statistics and figures as obtained from relevant sources [7].

- 1 In 2023, a McAfee survey revealed that 70 percent of respondents lacked confidence in distinguishing between real and cloned voices.
- 2 Data from Google Trends indicated that searches for "free voice cloning software" increased by 120 percent from July 2023 to July 2024.
- 3 According to Sumsup's research, the global rate of identity fraud nearly doubled from 2021 to 2023 as the number of AI-generated deepfakes across industries increased tenfold between 2022 and 2023.
- 4 In 2024, a deepfake implicating engineering firm Arup's executive led to the transfer of HKD\$200 million (approximately \$25 million) to bank accounts in Hong Kong. A staff member had a video conference with the false executive and other deepfake employees.
- 5 In 2019, a deepfake video depicting a seemingly intoxicated American House Speaker Nancy Pelosi spread across social media, amassing over 2.5 million views on Facebook.

These statistics indicate a significant rise in deepfake technology, particularly when used for harmful purposes. This highlights the urgent need for specialised deepfake detection systems that can identify these manipulations both proactively and reactively.

2.6 Review of Previous Research Work

For the purpose of this research study, a thorough literary review of research articles in the field of deep learning (DL) and convolutional neural networks (CNNs) was conducted, concentrating on publications from 2017 to 2024, with a particular emphasis on the years 2020-2024. The review included works from top

publishers such as Institute of Electrical and Electronics Engineers (IEEE), Elsevier, Multidisciplinary Digital Publishing Institute (MDPI), Association for Computing Machinery (ACM) and Springer as well as select papers from Computation and Electronics. A total of 75 papers were reviewed, covering various DL-related topics, with a noticeable published majority from 2020-2024. The objectives of this review were to analyse and evaluate the selected publications to:

- 1 Identify and describe DL and CNN techniques, as well as network types.
- 2 Examine the challenges associated with CNNs and propose alternative solutions.
- 3 Present various CNN architectures.
- 4 Assess the application of CNN models in detecting deepfakes.

The review was guided by specific search terms, including "Deep Learning", "Machine Learning", "Convolution Neural Network", "Architectures", "Detection", "Classification", "Segmentation", and "Overfitting" and drew from research paper sources such as IEEE Xplore, Researchgate, Academia, Google Scholar and Sci-Hub. Educational materials such as websites and webpages, textbooks and videos played a part in contributing to the research review process. This diverse sourcing helped to ensure a comprehensive and focused examination of the literature.

Deepfakes are typically generated using methods based on Generative Adversarial Networks (GANs), which were first introduced in [14]. In their work, the authors presented a novel framework where two models are trained simultaneously: a generative model G that learns the data distribution, and a discriminative model D that estimates the likelihood that a given sample originates from the training data rather than from G. The training process for G aims to maximize the chances of D making an error, creating a min-max game between the two models. Mathematically, the generator takes a random input z

with density p_z and produces an output $x = G(z, \theta_g)$ based on a specific probability distribution p_g (where θ_g denotes the parameters of the generative model). Meanwhile, the discriminator $D(x, \theta_d)$ calculates the probability that x is drawn from the training data distribution p_{data} (with θ_d representing the parameters of the discriminative model). When this is achieved, the discriminator becomes "fooled" and can no longer differentiate between samples from p_{data} and p_g , meaning that p_g will align with the desired probability distribution, p_{data} . A simplified illustration of the GAN framework is shown in **Figure 2.3**

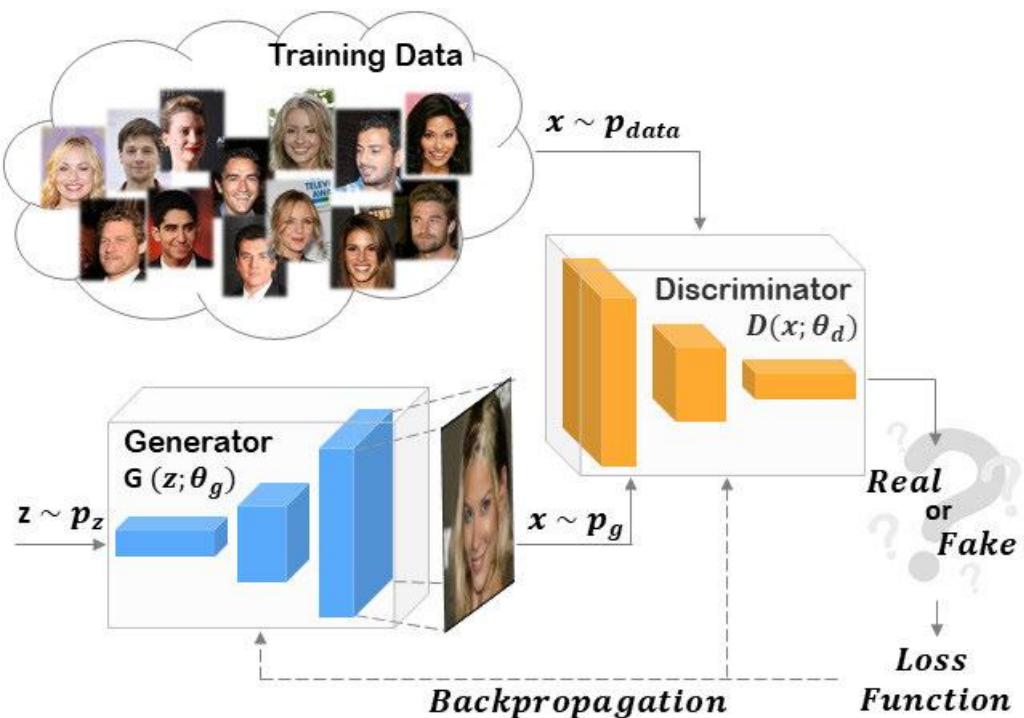


Figure 2.3 GAN Architecture [25]

[22] proposed a method that uses the frequency characteristics of face forgery. The approach in [22] employed a frequency CNN (FCNN) to analyse and classify real and fake faces. FCNN was tested using the FaceForensics++ dataset and found that it effectively detects forgeries in various video qualities. The FCNN model achieved the highest recall rates of 0.9256, 0.8639, and 0.8399 for raw, c23, and c40 videos, respectively. The model in [22] was also tested on the

CelebDF dataset and an automated FaceForensic benchmark, showing that their approach is effective for detecting face manipulation.

[23] introduced a two-stage method for detecting deepfakes called feature transfer, which used unsupervised domain adaptation. Feature vectors were extracted from a CNN and backpropagation was applied using domain-adversarial neural networks (BP-DANN) for better efficiency compared to traditional methods. Initially, a face detection network extracts the face region from the video frame, enlarges it and saves the image. These facial images are then processed through the CNN to obtain 2048-dimensional feature vectors. Additionally, a CNN pre-trained on a large deepfake dataset can be used to extract more transferable feature vectors, helping to bridge the gap between different datasets during training.

[24] focused on detecting compressed deepfake videos, which are common on social media platforms like Instagram and TikTok. The authors worked to find effective ways to identify these low-quality videos. A two-stream approach was proposed to analyse inconsistencies between frames and extract both frame-level and temporal information from compressed videos. They tested their technique on various datasets, including FaceSwap, NeuralTextures, Face2Face, deepfakes and Celeb-DF, and found that it outperformed previous methods which showed that their approach works against different compression levels.

[47] presented a comprehensive survey of deep learning-based approaches in image forensics, categorizing recent advances into three core areas: detection of routine image manipulations such as filtering and compression, detection of intentional falsifications such as splicing, copy-move and inpainting, and solutions to specific forensic problems like Deepfake detection and camera model identification. The authors highlighted that deep networks often utilize preprocessing techniques such as constrained filters or frequency-based enhancements to expose subtle manipulation traces. [47] also noted the increasing use of patch-level analysis and domain-adaptive architectures for detecting

forgeries. A key challenge identified across studies is the lack of generalization, particularly when models are tested across different datasets or manipulation types. The paper emphasizes the importance of diverse, high-quality datasets, robust training strategies, and countermeasures against adversarial or GAN-generated perturbations. These insights underscore the growing need for forensic models that are both resilient and adaptable to evolving forgery techniques.

Recent developments in face morphing detection methods have primarily focused on single image-based approaches that rely solely on the analysis of the presented image, without requiring access to additional trusted inputs. [45] proposed an ensemble-based method that combines features extracted from multiple deep convolutional neural networks, followed by fusion techniques to improve the detection of morphing attacks. Similarly, [44] introduced an encoder-decoder architecture that learns residual representations of morphed faces, contributing to better generalization across datasets.

Another stream of work emphasizes frequency and wavelet domain inconsistencies. [45] developed a wavelet-based attention-aware detection framework that exploits fine-grained inconsistencies often overlooked in spatial representations. This approach significantly improves the model's ability to distinguish between genuine and morphed face images, even under challenging conditions.

To evaluate model generalization and robustness, [44] performed cross-dataset testing and recommended training strategies aimed at improving resilience to common transformations such as Gaussian noise and illumination variation.

Further benchmarking efforts were made in [46] which introduced MorDeepy, a morphing detection system based on fused classification heads within a CNN architecture. Their work, aligned with the goals of the Morphing Attack Detection (MAD) challenges, demonstrated how architectural adjustments can yield state-of-the-art performance in detecting face morphs.

The literature indicates that researchers have employed various strategies to develop effective deepfake detection systems. Despite the diversity of these approaches, many share common foundational principles, primarily concentrating on identifying inconsistencies and traces of manipulation left by GAN tools during the generation process [26].

Currently, deepfakes encompass multiple modalities, including audio, video, images and hybrid models. Among these, image and video-based deepfakes are the most prevalent, leading to a significant focus in research on detecting deepfakes in these formats.

Table 2.1 outlines contributions from existing studies related to deepfake detection.

Table 2.1 Literature Review

S/N	Name of Paper	Methodology	Limitations	Key Findings
1	Exposing AI Created Fake Videos by Detecting Eye Blinking [48].	Feature extraction from spatial, temporal and frequency domains with LSTM.	Sensitivity to eye-state threshold setting.	Deepfake videos exhibited 10 times lower blink rate.

2	<p>FaceForensics++: Learning to Detect Manipulated Facial Images [49]</p>	<p>Developed and benchmarked FaceForensics++, a dataset containing manipulated videos using various face-editing techniques (e.g. Deepfakes, Face2Face). Evaluated detection performance using deep learning classifiers.</p>	<p>Performance of trained models degraded significantly with increased video compression and lower quality; models often overfit specific manipulation types.</p>	<p>FaceForensics++ significantly improved benchmark reproducibility and enabled systematic evaluation of forgery detection models across compression levels.</p>
3	<p>Deepfake Video Detection Using Convolutional Neural Network [29].</p>	<p>A model that analyses the frames of the videos using transfer learning on VGG-16 model to train the dataset to detect inconsistencies in facial features,</p>	<p>Lack of research in audio detection.</p>	<p>It was observed that accuracy of the proposed model decreased with lower quality images and videos.</p>

		compression rate and discrepancies.		
4	Deepfakes detection with automatic face weighting [50].	Introduces soft attention to reduce compute cost and enhance accuracy.	Performance was dependent on dataset/task.	Improved classification accuracy.
5	The Deepfake Challenges and Deepfake Video Detection [51].	Trained on eye image data, tested on video.	Failed on diverse datasets.	Achieved over 90% accuracy on real videos.
6	DeepFake Video Detection: A Time-Distributed Approach [52].	Uses time-distributed EfficientNet-B1 with LSTM.	Large input sizes increased compute demands.	86–92% accuracy with 14million fewer parameters than XceptionNet
7	Two-branch Recurrent Network for Isolating Deepfakes in Videos [53].	Real vs. fake face mapping into inner vs. outer hyperspheres using BiLSTM.	Failed with facial hair and low lighting.	AUC improved from 92% to 99% for medium compression.
8	A Novel Machine Learning based Method for	A model, consisting of a convolutional	High complexity	Attained a high accuracy

	Deepfake Video Detection in social media [70].	neural network (CNN) and a classifier network is proposed.		
9	Employing Transfer-Learning based CNN architectures to Enhance the Generalizability of Deepfake Detection [71].	A CNN for feature extraction to train a binary classifier that differentiates between real and fake videos	Security risks were not accounted for.	An accuracy of 98% was achieved.
10	DeepFake Detection Using Inception-ResNetV2 and LSTM [54].	Combines ResNeXt with LSTM	Small dataset limits the overall generalization	Achieved 94% accuracy on Celeb-DF dataset.
11	Feature Transfer: Unsupervised Domain Adaptation for Cross-Domain Deepfake Detection [23].	A new method named Feature Transfer is proposed in this paper, which is a two-stage Deepfake detection method	High energy consumption	Solved the overfitting problem.

		combining with transfer learning		
12	Detecting DeepFake, FaceSwap and Face2Face facial forgeries using frequency CNN [22]	Developed a frequency-based CNN model, trained and tested on FaceForensics++ dataset and Celeb-DF(v2) dataset	High energy consumption and high delay	Strong robustness of developed model
13	Optical Flow based CNN for detection of unlearnt deepfake manipulation [72].	An approach that made use of flow fields	High energy consumption	Improvement in the robustness of developed model compared to previous works.
14	Deep Fake Video Detection Using Transfer Learning Approach [55].	Transfer learning using InceptionResNet V2 with LSTM.	Used only deepfake portion as such, not representative of mixed	Achieved accuracy improved by 6.73% with more epochs.

			real/fake videos.	
16	Detection of Synthesized Videos using CNN [56].	Uses residual image encoding and LSTM for fine feature capture.	High compute cost due to long training.	Achieved a 94.75% accuracy.
17	A Hybrid CNN-LSTM model for Video Deepfake Detection by Leveraging Optical Flow Features [57].	VGG16-based model evaluated across several training epochs.	Accuracy dropped when tested across merged datasets.	Accuracy peaked at 97.25% by epoch 13.
18	Generalized Deepfake Video Detection Through Time-Distribution and Metric Learning [58].	Trained on FF++, DFDC, and Celeb-DF. Achieves state-of-the-art generalization performance.	Tended to ignore deepfake frames after frame 150.	Performed better with 40 epochs setup compared to 20 epochs.
19	Using cascade CNN-LSTM-FCNs to identify AI altered video based	Combines optical flow, VGG16 and LSTM.	Low performance on the DFDC dataset with	Achieved 91% accuracy at 70 frames per second (fps)

	on eye state sequence [59].		lower frame counts.	
20	Multi-Feature Fusion Based Deepfake Face Forgery Video Detection [60].	Multi-stream CNN-LSTM with contrastive loss.	Performed poorly on expression-swap deepfakes.	Achieved 97.3% on FF++ and independent of generation method
21	Detecting Compressed Deepfake Videos in Social Networks Using Frame-temporality two-stream Convolutional Network [24].	A two-stream method by analysing the frame-level and temporality-level of compressed Deepfake videos.	Low accuracy	Robustness with the compressed videos
22	Frequency-based Deep-Fake Video Detection using Deep Learning Methods [61].	Uses eye-blink frequency analysis.	Missed neural texture deepfakes.	Achieved a 95.57% accuracy on FaceForensics++ dataset.
23	Deepfake Detection from Face-swapped Videos Using	Feature extraction from spatial, temporal and frequency	It lacked generalization	Fusion improved the accuracy significantly.

	Transfer Learning Approach [62].	domains with LSTM.	to unseen datasets.	
24	Convolutional long short-term memory-based approach for deepfakes detection from videos [63].	CNN with LSTM trained on FF++ dataset using different train-test splits.	No analysis of robustness to noise and distortions.	80/20 split yielded better accuracy than 70/30 split.
25	Deepfake Detection Using Xception and LSTM [64].	Compact CNN-LSTM model (152 times smaller than standard).	Overfitting occurred when using more than 128 LSTM units.	Strong performance with low compute demand.
26	DeepFake Videos Detection and Classification Using Resnext and LSTM Neural Network [67].	Combines dense blocks and Bi-LSTM for DFDC dataset.	ReLU activation limited minor details in detection.	Achieved 99.3% accuracy with fewer parameters.
27	Comparative Analysis and Evaluation of CNN Models for Deepfake Detection [68].	Combined the XceptionNet model with LSTM.	Training and validation accuracy gap indicated overfitting.	Maintained accuracy with reduced compute time.

28	Deepfake Detection using Convolution Neural Network [73].	Proposed a method to train the classifier based on video frames as input.	Limited epochs for better training	The trained model was able to classify videos with an accuracy 90%
29	A new lung cancer detection method based on the chest CT images using Federated Learning and blockchain systems [74].	An approach that takes a modest data from multiple hospitals and uses blockchain-based Federated Learning (FL) to train a global DL model.	Highly complex	An accuracy of 99.69% was achieved.
30	A novel blockchain-based Deepfake Detection Method using Federated and Deep Learning Models [75].	Proposed a system based on Blockchain and Federated Learning/Deep Learning (BFLDL)	The task, the dataset, and the model design will determine the precise performance difference	Noted improvement of 6.6% on average compared to six benchmark models.

31	An Ensemble Approach to Facial Deepfake Detection Using Self-Supervised Features [69].	The study evaluated ensemble models (committees) for deepfake detection using: DFDC and FaceForensics datasets.	Evaluation was limited to the DFDC and FaceForensics datasets generalization untested.	Committee models consistently outperformed individual classifiers. Best performance was achieved by c4 (AUC:93.22%, F1: 85.63%).
----	--	---	--	--

CHAPTER THREE

METHODOLOGY

3.1 Introduction

This chapter outlines the methodological framework for conducting this research project, detailing the procedural steps, data collection and utilization strategies, and the use of visual aids such as diagrams and charts, as well as equations, to facilitate the successful completion of the project.

3.2 Techniques Employed

The techniques employed in this study will ensure efficient deepfake detection. They include methods for preprocessing data, training the model, and evaluating its performance, all aimed at achieving reliable classification of real and fake facial images.

3.2.1 Data collection

A diverse dataset of deepfake media will be compiled from multiple sources, including public datasets like OpenForensics and FaceForensics++, as well as online platforms such as GitHub, YouTube, Kaggle and Reddit. The dataset will be categorized into two groups - fake and real - and will be utilized for three main purposes: training, validation, and testing. The dataset will comprise various sizes, ranging from smaller collections like MICC_F2000 (700 images) and Image Forensics Challenge (1176 images) to larger ones like National Institute of Standards and Technology, NIST (50,000 images and 500 videos). To ensure compatibility, the dataset will undergo pre-processing to standardize the data format for training and testing.

3.2.2 Data Preprocessing

Input data will be resized to a uniform size to facilitate efficient processing. Data augmentation techniques (e.g., rotation, flipping, cropping) will be applied to expand the size of the training dataset and prevent overfitting. Normalization techniques (e.g., mean subtraction, standardization) will also be applied to normalize the pixel values of the input. The aim of pre-processing is to improve the quality of the data, reduce noise and errors, and increase the chances of building a robust and accurate model.

3.2.3 Convolutional Neural Network (CNN)

A CNN architecture will be designed and implemented using a deep learning framework (PyTorch) through Python programming language and multiple libraries. The deepfake detection system requires a robust CNN architecture that can effectively detect and classify deepfakes from real. The proposed architecture will consist of multiple layers, including convolutional, pooling, and fully connected layers. The output layer will be a binary classification layer which will output a probability value between 0 and 1. This probability value will represent the likelihood of the input image being a deepfake.

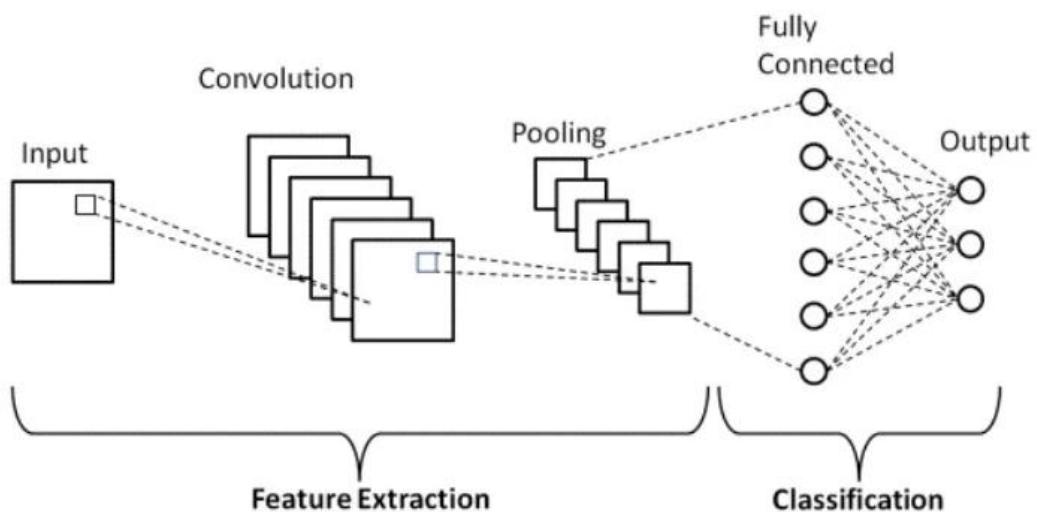


Figure 3.1 Basic CNN Architecture [8]

In **Figure 3.1**, a basic CNN architecture is depicted, where the Convolution and Pooling layers work to extract key features from the input data. The Fully Connected layer then takes these features and uses them to classify the input, ultimately producing the output. A convolutional layer applies a set of filters to the input data to extract features. The input data is a 3D array of pixels, where each pixel has a value representing its intensity. The convolutional layer has a set of filters, also known as kernels, that are used to extract features from the input data. A kernel is a small matrix that slides over the input data performing the mathematical operation called Convolution to extract these features and generate a feature map. These features will be extracted from the convolutional and pooling layers. The features will be used to train a classifier to improve the performance of the deepfake detection system. The pooling layer works by dividing the input data into small regions, called pooling regions, and then applying a pooling function to each region. The pooling function reduces the spatial dimensions of the input data by selecting the maximum or average value from each region. The fully connected layer connects every input to every output. The input data is a 1D array of values, where each value represents a feature extracted from the previous layer. The fully connected layer has a set of weights (weight represents the connection between an input and an output) that are learned during training. The input data is multiplied by the weights, performing a dot product, to produce an output. The output is then passed through an activation function (ReLU, Rectified Linear Unit), to introduce non-linearity into the model. The output of the fully connected layer is a 1D array of values that represent the predicted output. The performance of the deepfake detection system will be evaluated using a variety of metrics, including accuracy, loss, precision, recall, and confusion matrix. A more detailed representation of the CNN architecture is illustrated in **Figure 3.2** showing the flow of data through the layers of a CNN architecture from input to convolution, pooling, activation function, fully connected and then output.

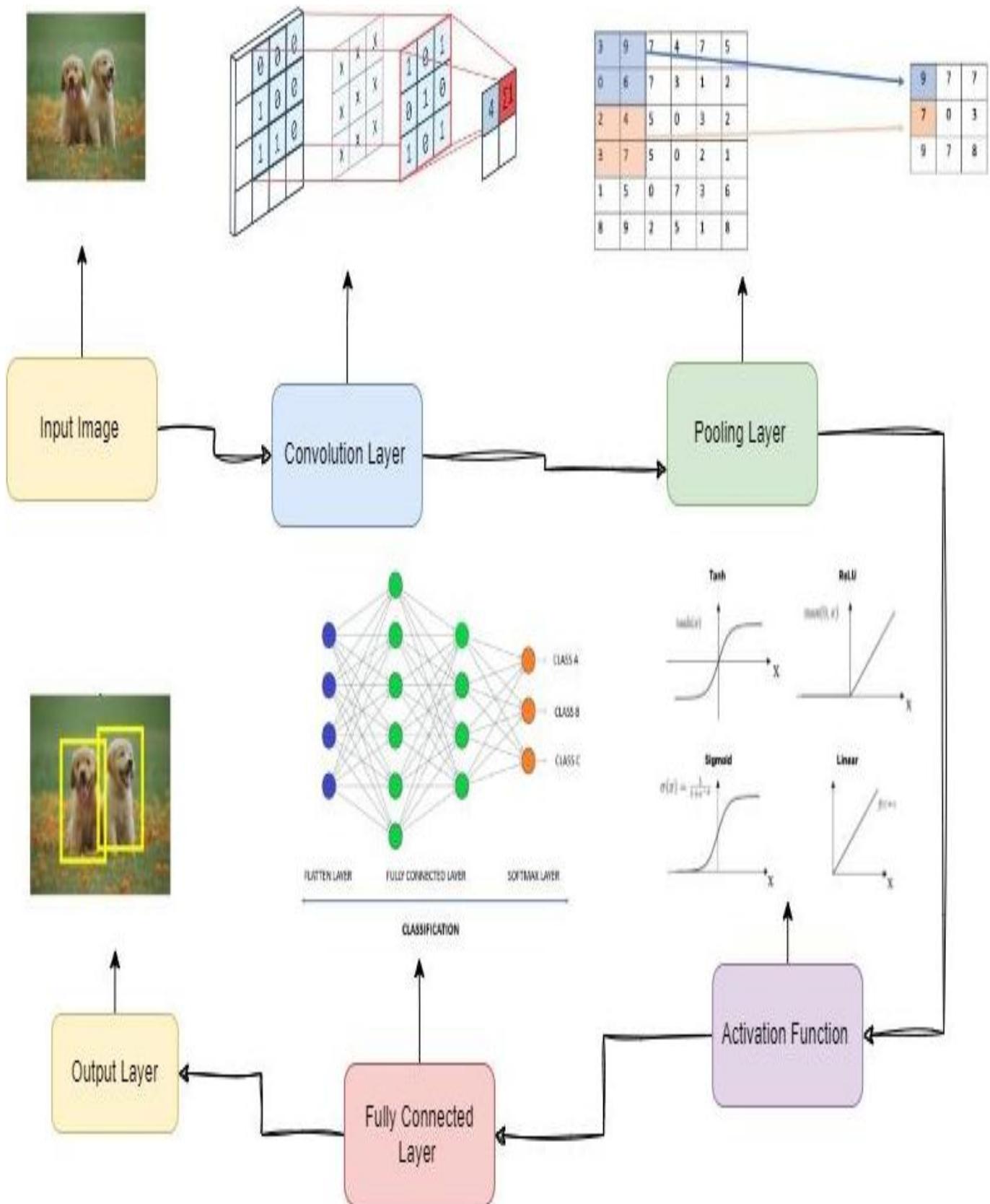


Figure 3.2 Expanded CNN Architecture [8]

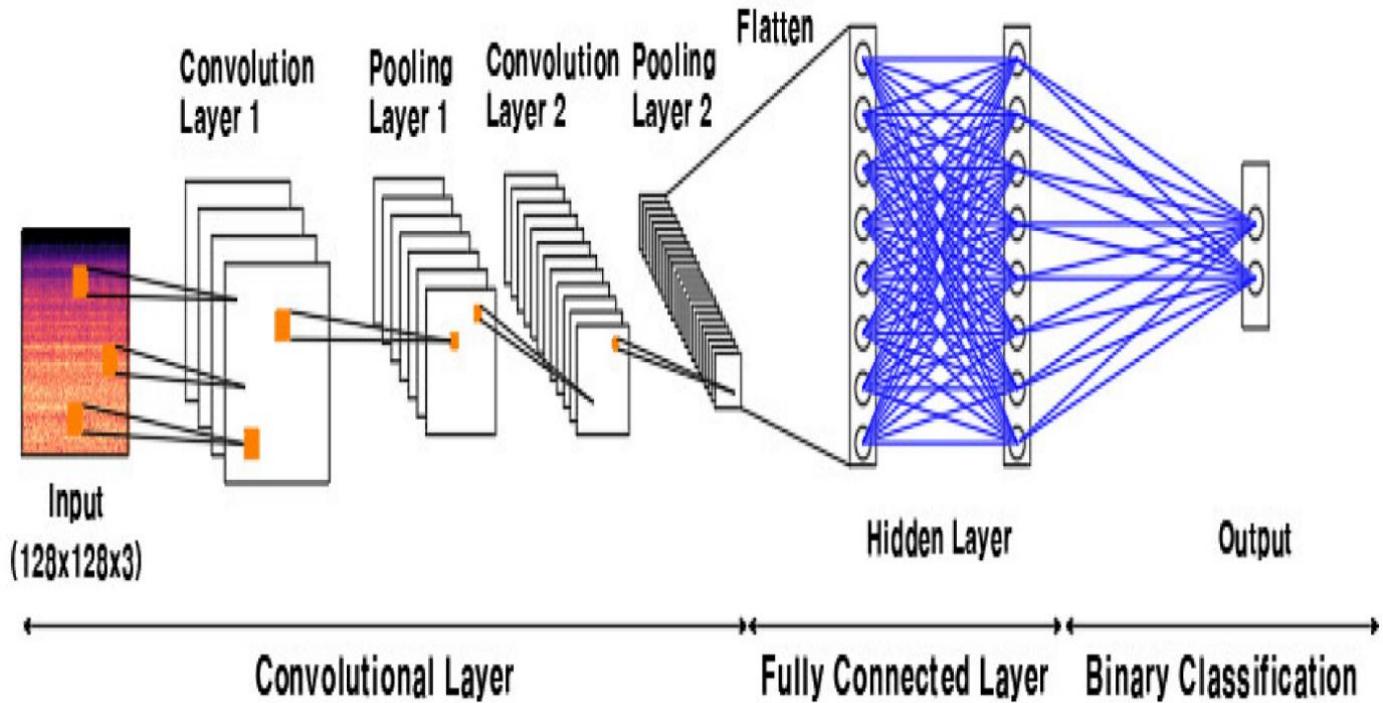


Figure 3.3 Proposed CNN Architecture with Binary Classification Output

[28]

In **Figure 3.3**, the CNN architecture with a binary classification output layer to classify Real or Deepfake is proposed. The proposed architecture will employ multiple layers of convolution and pooling before flattening in the fully connected layer to produce binary output (0 to 1) which indicates real or deepfake.

3.2.4 Training

The CNN model will be trained using the pre-processed dataset. The model will be trained using a suitable optimizer (ADAM) and loss function (binary cross-entropy). The optimizer and loss function work to minimize the difference between the model's predictions and the actual true values. The model will undergo a predetermined number of training epochs to ensure convergence and prevent overfitting.

Post-training, a machine learning model typically falls into one of three categories: underfitted, balanced, or overfitted. An underfitted model fails to capture the underlying patterns in the training data, often resulting in poor performance on both training and unseen data. A balanced model effectively learns relevant

features and generalizes well to new data, achieving optimal performance. In contrast, an overfitted model memorizes the training data too closely, learning noise and irrelevant details, which impairs its ability to generalize to unseen inputs. **Figure 3.4** shows a visual representation of the three categories.

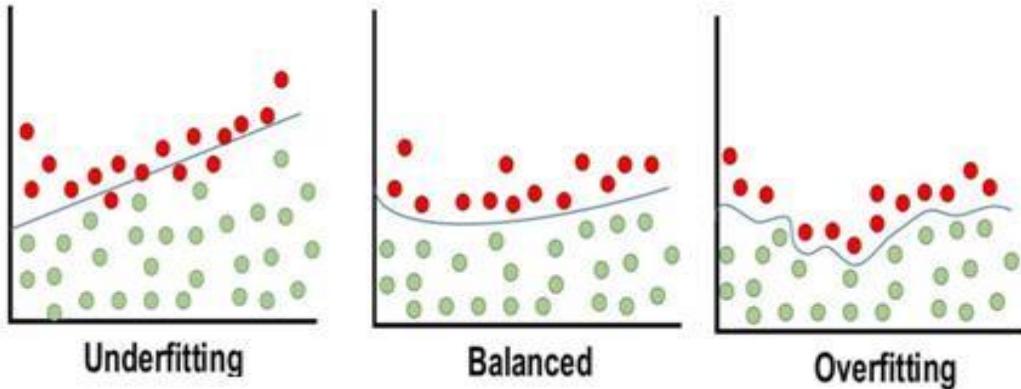


Figure 3.4 Obtainable post-training models [8]

The training process will occur in two steps: Forward propagation and Backward propagation. Forward propagation is the process of passing input data through the CNN model to compute the output. The input data flows through the network, layer by layer, with each layer applying its set of transformations to the input data. Backward propagation, also known as backpropagation, refers to the process of computing the gradients of the loss function with respect to the model's parameters. The model then adjusts parameters proportionate to errors from guess thereby optimizing parameters [20]. One common method for error calculation is the Mean Squared Error (MSE) given by:

$$\text{MSE} = (\text{Predicted Output} - \text{Actual Output})^2$$

3.2.5 Testing

The trained CNN model will be evaluated using a separate test dataset. The performance of the model will be evaluated using metrics such as accuracy, precision, recall, and confusion matrix. The model will be fine-tuned as necessary to improve its performance. Important hyperparameters will be adjusted and fine-

tuned as necessary such as learning rate, epochs, batch size etc. Transfer learning may be applied where the pre-trained model is fine-tuned on a new dataset. In this case, backpropagation may be applied after testing to fine-tune the model's parameters on the new dataset.

3.2.6 Deployment

The deepfake detection system will be deployed through Streamlit App that allows users to upload images and receive the detection results through an interactive interface.

By employing these techniques, the deepfake detection system will be able to effectively detect deepfake materials.

3.3 Research Instruments Used

In order to conduct a reliable and effective study of this research topic, several research tools and methods have been utilized. The various methods are further discussed below.

3.3.1 Software and Tools

This project leveraged several software tools and libraries to preprocess, train, and evaluate the deepfake detection model. These include PyTorch, Python, Matplotlib amongst others.

3.3.1.1 Deep Learning Framework

PyTorch is a popular deep learning framework for working on neural networks, CNN in particular. It provides a set of libraries and tools for executing deep learning algorithms, featuring pre-defined layers and functions designed for image processing.

3.3.1.2 Dataset Repositories

This provides for publicly available datasets such as FaceForensics++. These aid in the training and testing of developed models by offering a vast variety of multimedia data.

3.3.1.3 Evaluation Tool

Scikit-learn is an open-source library written in Python that assists with data analysis and evaluation by providing metrics such as accuracy, precision, and recall. It also includes tools for cross-validation and hyperparameter tuning, which aid in selecting the optimal model and parameters.

3.3.1.4 Integrated Development Environments

Jupyter Notebook is an excellent tool for prototyping and experimenting with completed models. Visual Studio Code is designed for comprehensive software development, offering features such as debugging and code suggestions.

3.3.1.5 Data Visualization Tool

Matplotlib is an important library for displaying training progress, loss curves, and confusion matrices. These visual representations assist in assessment of the model's performance.

3.3.1.6 Version Control software

GitHub is an online platform for version control, enabling developers to share and work together on codes. It offers tools for sharing projects and promoting collaboration.

3.3.1.7 Cloud Platform

Google Colab is a free-to-use cloud platform which offers powerful processors to significantly speed up the training process for deep learning models.

3.3.2 Diagrams and Visuals

These diagrams and visuals serve to clarify the structure and flow of the project, offering a clearer understanding of the methodology. Flowcharts are used to depict the step-by-step processes involved, while the block diagram provides a high-level overview of the system's components and their interactions.

3.3.2.1 Block Diagram for Detection System

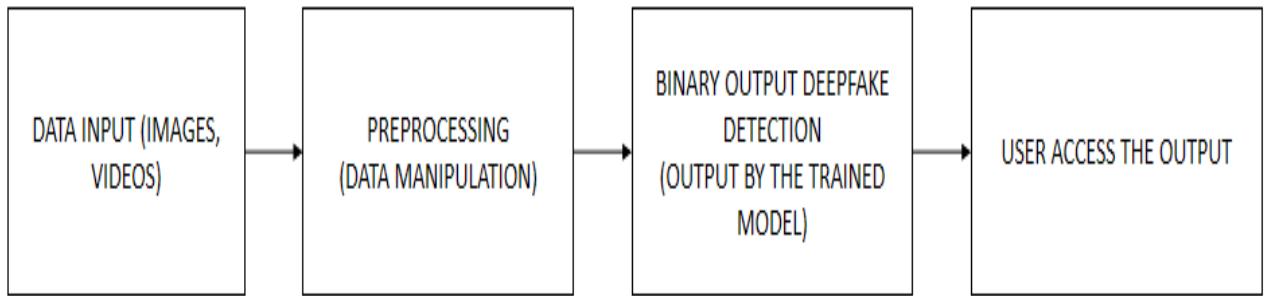


Figure 3.5 Block diagram for detection system

Figure 3.5 presents the block diagram for the detection system. The process begins with the input of visual data (images) followed by preprocessing to prepare the data for analysis. The preprocessed data is then passed into a trained and validated model that performs binary classification to detect deepfakes. The output as final result is made accessible to the user.

3.3.2.2 Flowchart for Deepfake Detection

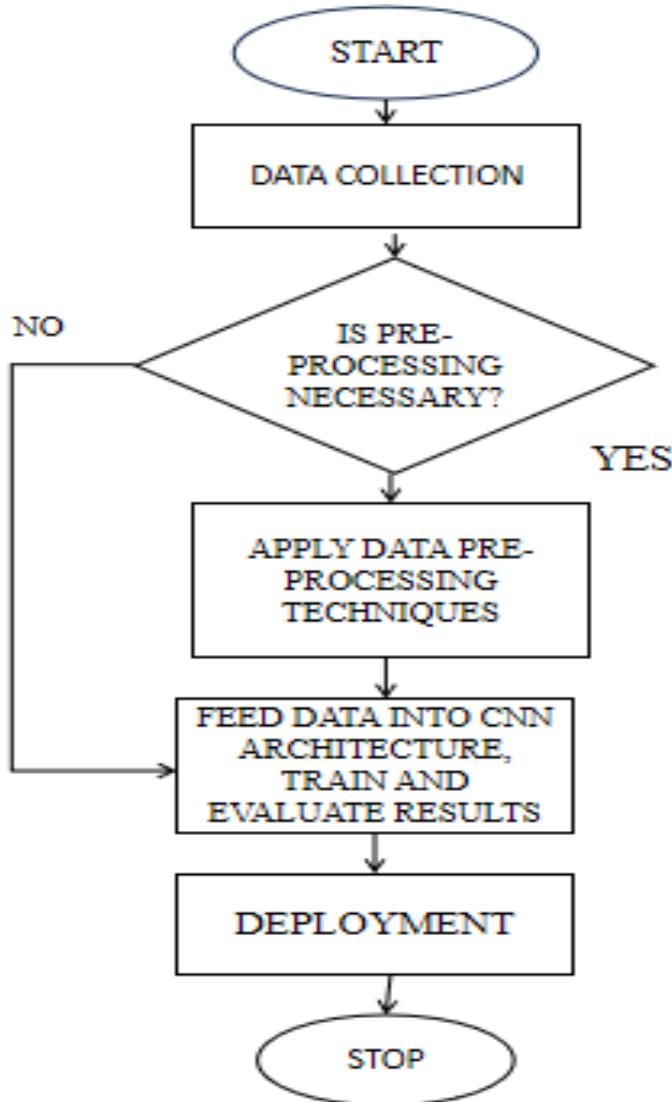


Figure 3.6 Flowchart for development process of detection system

Figure 3.6 illustrates the end-to-end development workflow for the deepfake detection system. The process starts with data collection followed by a decision point on whether preprocessing is needed. If necessary, preprocessing techniques such as rotation and cropping are applied. The processed data is then fed into a Convolutional Neural Network for training. Afterward, the model undergoes validation and evaluation stages. If these stages yield acceptable results, the model proceeds to prediction and eventual deployment. Otherwise, it loops back for further improvement.

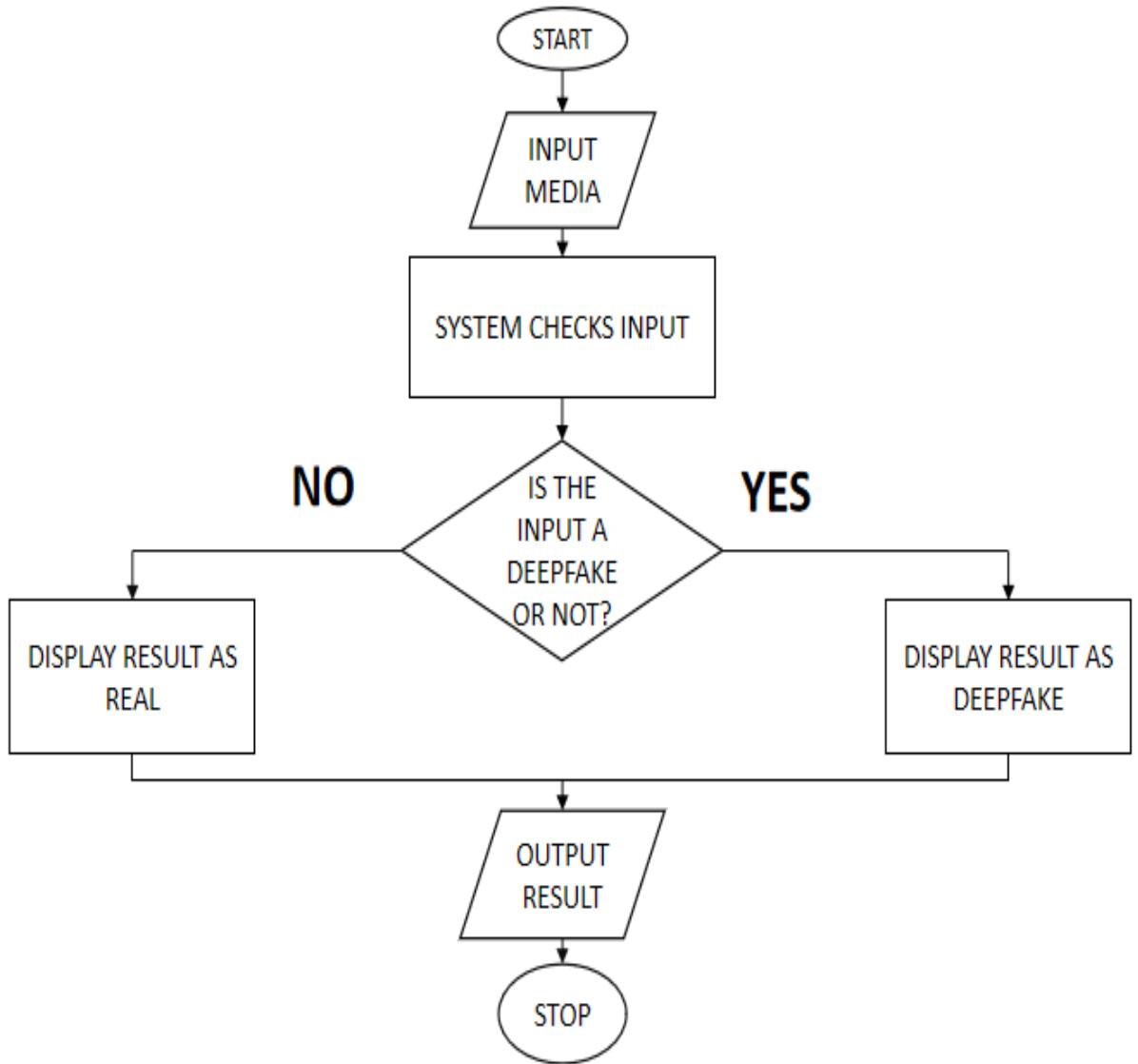


Figure 3.7 Flowchart for detection process of developed model

Figure 3.7 outlines the runtime decision making process of the deployed system. It starts when a user inputs media which is an image either captured at that moment or from file storage. The system then analyzes the input to determine whether it is a deepfake or not. Based on the result, it displays either "Deepfake" or "Real" to the user. This straightforward logic ensures user friendly output delivery and shows the model's binary classification capability in real time applications.

3.3.2.3 Risk Management for Deepfake Detection

Table 3.1 provides an overview of potential risks to be expected during the development of the deepfake detection system, along with their expected impact and management strategies. Key risks include poor detection accuracy and high loss which may be resolved through hyperparameter adjustments like learning rate optimization, documentation errors which may be resolved by careful research and report writing, overfitting of the model which may be managed with early stopping or learning rate alterations and hardware failure or inadequacy which may be resolved by use of cloud processors such as Google Colab. This approach ensures effective risk mitigation, enhancing the system's reliability and performance.

Table 3.1 Risk management for Deepfake Detection system development

S/N	RISK	EXPECTED IMPACT	MANAGEMENT STRATEGY
1	POOR DETECTION ACCURACY AND HIGH LOSS.	HIGH	HYPERPARAMETER ADJUSTMENT e.g LEARNING RATE.
2	DOCUMENTATION ERROR	HIGH	CAREFUL RESEARCH AND REPORT WRITING.
3	OVERFITTING OF DEVELOPED MODEL	HIGH	EARLY STOPPING OF TRAINING OR ALTERING LEARNING RATE.
4	HARDWARE FAILURE/INADEQUACY	HIGH	USE OF CLOUD AVAILABLE PROCESSORS LIKE GOOGLE COLAB.

CHAPTER FOUR

IMPLEMENTATION AND RESULTS

4.1 Introduction

This chapter presents the implementation of the deepfake detection system as well as the results obtained from the model training and evaluation. It includes the visualizations, performance metrics, and comprehensive analysis of outcomes.

4.2 Dataset and Model development

This describes the source and structure of the dataset, preprocessing techniques applied to enhance data quality, and the design and training of the model architecture.

4.2.1 Dataset Description

The dataset for this project was sourced from a diverse set of datasets including the widely used Faceforensics++, Openforensics and Celeb DF V2 obtainable from Kaggle.com, Zenodo.org and the official Faceforensics GitHub repository.

The dataset was structured as follows:

4.2.1.1 Celeb DF V2

This was obtained from Kaggle.com [38] and consisted of three subfolders; YouTube-real (real videos), Celeb-real (real videos) and Celeb-synthesis (fake videos). YouTube-real contained a total of 300 videos, Celeb-real contained a total of 590 videos and Celeb-synthesis contained a total of 5,639 videos.

4.2.1.2 Faceforensics++

This was obtained after filling a form on the Faceforensics GitHub page [39] which allowed the downloading of the Faceforensics dataset. It contained two subfolders; original_sequences for the real videos consisting 363 videos and manipulated_sequences for the fake videos consisting 3,068 videos.

4.2.1.3 Openforensics

This was obtained from Zenodo.org [40] and consisted of three subfolders; Train, Test and Validation which also had folders for real and fake images. This dataset consisted of images totalling 140,002 combined for Train subfolder, 39,428

combined for Validation subfolder and 10,905 combined for Test subfolder. Screenshots of the dataset folder structure are provided in **Appendix A**.

The video data was extracted into images capturing the faces of subjects in the videos whilst logging the images as comma separated values (csv) in a separate labelled file indicating real and fake as 1 and 0 respectively. This was achieved with a python script titled Faces.py which automatically ran through the directories for real videos and fake videos and extracted 10 frames per video. Faces.py made use of OpenCV which read through every single frame of a video then attempted face detection with Multi-task Cascaded Convolutional Neural Network (MTCNN) and collected all frames where a face was successfully detected then randomly selected 10 from those face frames for saving. The real videos were significantly less than the fake videos and this had to be accounted for as more frames were extracted from the real videos to balance the distribution. The faces were then stored in folders for real and fake images and labelled in a faces.csv format for data loading.

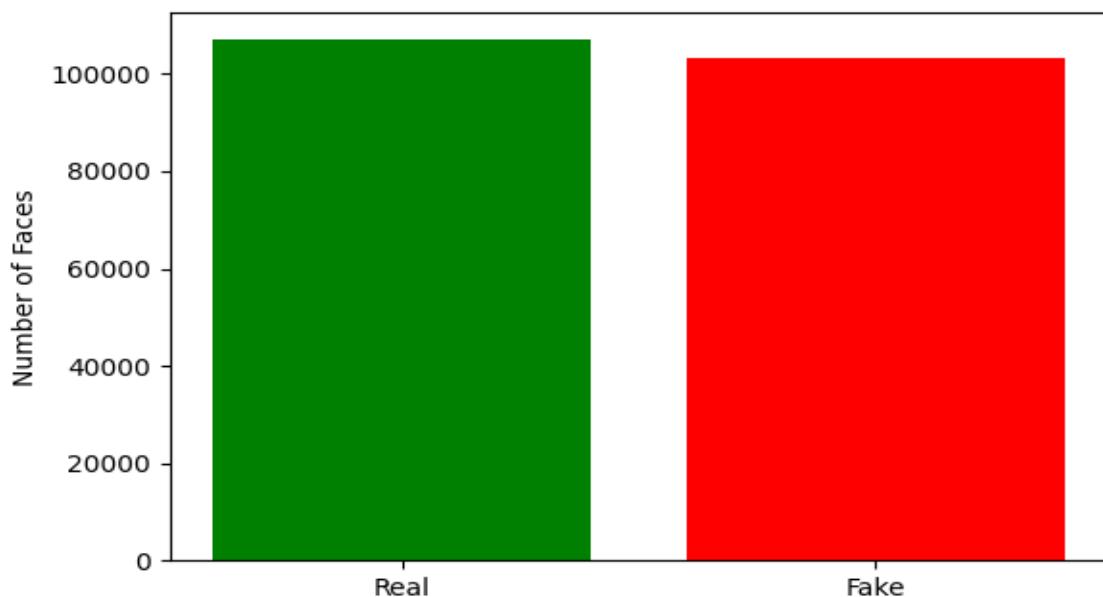


Figure 4.1 Faces Class Distribution post face extraction (Bar Chart)

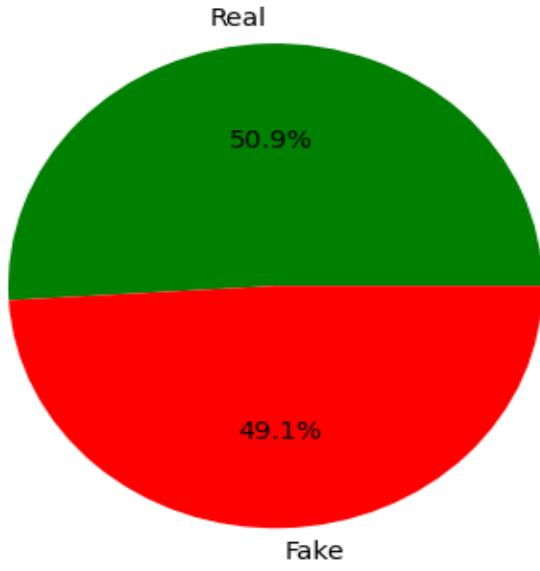


Figure 4.2 Faces Class Distribution post face extraction (Pie Chart)

The Faces.py script plotted visualisation of the data showing distribution of the classes. By observation, a slight imbalance was observed in the dataset depicted in **Figures 4.1** and **4.2**, where real images accounted for 50.9% and fake images for 49.1% of the total dataset. To fix this imbalance, a cleanup script named balancefakeimages.py was implemented. This script balanced the dataset by randomly deleting 3,966 excess real images, thereby balancing the number of real and fake images. The terminal output after running the script is shown in **Figure 4.3**. Following this, the faces.csv file was updated to accurately reflect the new balanced distribution of 103,174 images per class.

```
PS C:\Users\CHARLES EKANEM\Documents\FINALYEARPROJECT> & "C:\Users\CHARLES EKANEM\AppData\Local\Microsoft\WindowsApps\PROJECT\balancefakeimages.py"
[INFO] Total Real Faces: 107140
[INFO] Total Fake Faces: 103174
[INFO] Deleting 3966 excess real images...
[INFO] Rebuilding faces.csv ...
[INFO] Saved updated CSV: C:\Users\CHARLES EKANEM\Documents\FINALYEARPROJECT\dataset\faces\balanced_faces_temp.csv
[FINAL] Real: 103174, Fake: 103174
[INFO] Saved bar chart: C:\Users\CHARLES EKANEM\Documents\FINALYEARPROJECT\dataset\faces\faces_bar.png
[INFO] Saved pie chart: C:\Users\CHARLES EKANEM\Documents\FINALYEARPROJECT\dataset\faces\faces_pie.png
[INFO] Distribution plots generated successfully.
PS C:\Users\CHARLES EKANEM\Documents\FINALYEARPROJECT> []
```

Figure 4.3 Terminal output after balancefakeimages.py

The python script for balancing the dataset plotted a bar chart and a pie chart after balancing and these are shown in **Figures 4.4** and **4.5** which indicate an even distribution of the dataset at 103,174 real and 103,174 fake images.

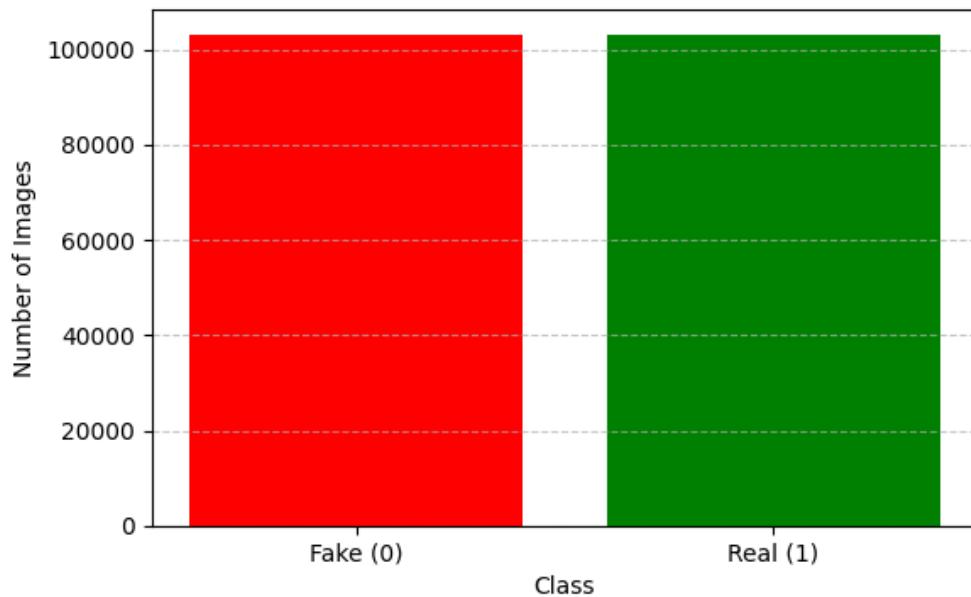


Figure 4.4 Bar chart showing the balanced distribution of dataset classes.

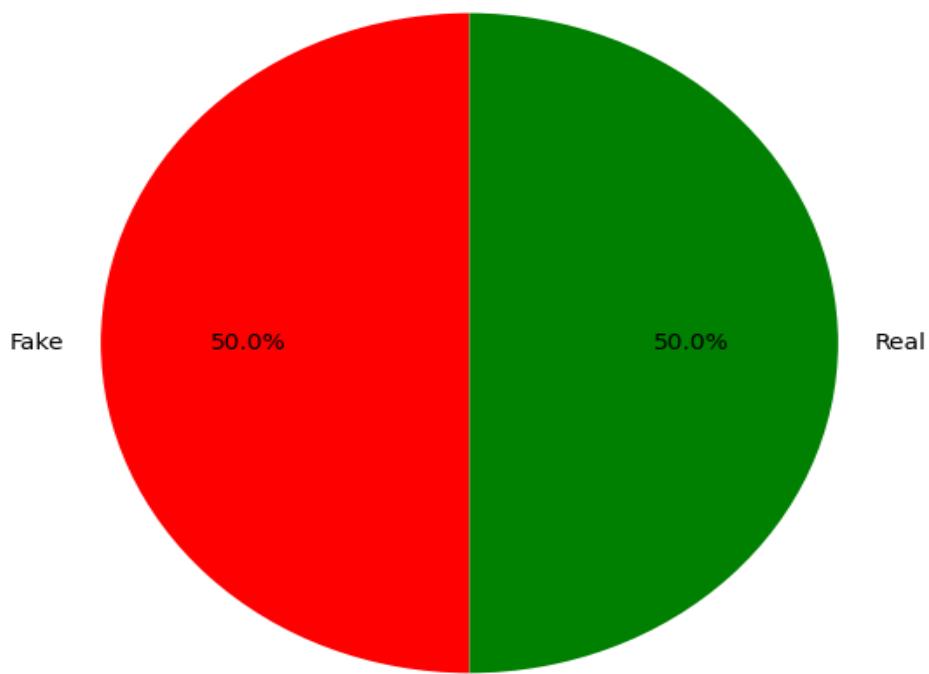


Figure 4.5 Pie chart showing the balanced distribution of dataset classes.

4.2.2 Data Preprocessing

Effective data preprocessing is an important step in any deep learning pipeline, especially for tasks like deepfake detection where subtle details can heavily influence model performance. This was achieved with the `Faces.py` script using transform module imported from `torchvision`. This helped to resize the images to match the input dimensions for the architecture at 160x160 while converting the images to tensors for data manipulation.

4.2.3 Training Model

The system was developed using PyTorch on CPU environment and is based on the Inception-ResNet V1 architecture with pretrained weights from the VGGFace2 dataset. The dataset was split into training and validation sets using a randomized 70/30 split, resulting in 144,443 images for training and 61,905 images for validation. **Figure 4.6** provides a visual representation of this distribution showing a 70/30 split of the dataset into training and validation sets.



Figure 4.6 Training and Validation distribution

The hyperparameters were defined within a configuration dictionary prior to the implementation of the data loading, model training, evaluation, and execution

functions. The Binary Cross Entropy Loss was used as the loss function, which is appropriate for binary classification tasks. The Adam optimizer was selected for its efficiency. A Sigmoid activation function was applied to the model's outputs to predict in a range between 0 and 1. The model was trained for 20 epochs using a batch size of 4. During training, its performance was regularly checked using validation accuracy, and the best version of the model was saved automatically. The dataset was handled with PyTorch Data Loaders, with shuffling enabled for the training data to help the model generalize better. Throughout the process, both loss and accuracy were tracked for the training and validation sets to monitor progress and ensure the model was learning effectively. After training, the final model was saved to the local disk as a path file (.pth). A detailed breakdown of the training script is provided in **Appendix B**.

4.2.4 Model Architecture

The deepfake detection system was built using the Inception-ResNet V1 architecture which accepts 160×160 sized face images as input. This model combines the strengths of two powerful networks: Inception modules (which capture multi-scale features) and residual connections (which help with training deeper networks). The Inception module helps the model capture different details in an image by using multiple types of filters at the same time. It applies 1×1 , 3×3 and 5×5 convolutions, as well as max pooling, all in parallel. Then, it combines the results into one output. This approach lets the model capture both small, fine details and larger, broader patterns in the image, which is especially useful for complex images like faces. The Residual module employs skip connections, which allow the input "skip" one or more layers and be added directly to the output of a deeper layer. This helps solve the problem of vanishing gradients, where deep layers struggle to learn due to small gradient values. By adding the input to the output directly, the model learns more easily and trains faster.

Inception-ResNet V1 includes 119 layers in total, with 22 layers using 3×3 filters, spread across different parts of the network like the initial stem, Inception-ResNet-A and C blocks, and the down sampling (reduction) blocks. The model has several repeating modules: five A-blocks, ten B-blocks, five C-blocks, and two reduction blocks. It ends with average pooling, a dropout layer, and a fully connected layer that reduces everything down to a single output. This architecture was selected because it comes pretrained on the VGGFace2 dataset, which contains a large number of diverse facial images. This gives the model a head-start in learning facial features and easily improves its performance when fine-tuned on deepfake data.

A overview structure of the model architecture is illustrated in **Figure 4.7** while details on the Inception-ResNet V1 architecture are provided in **Appendix C**.

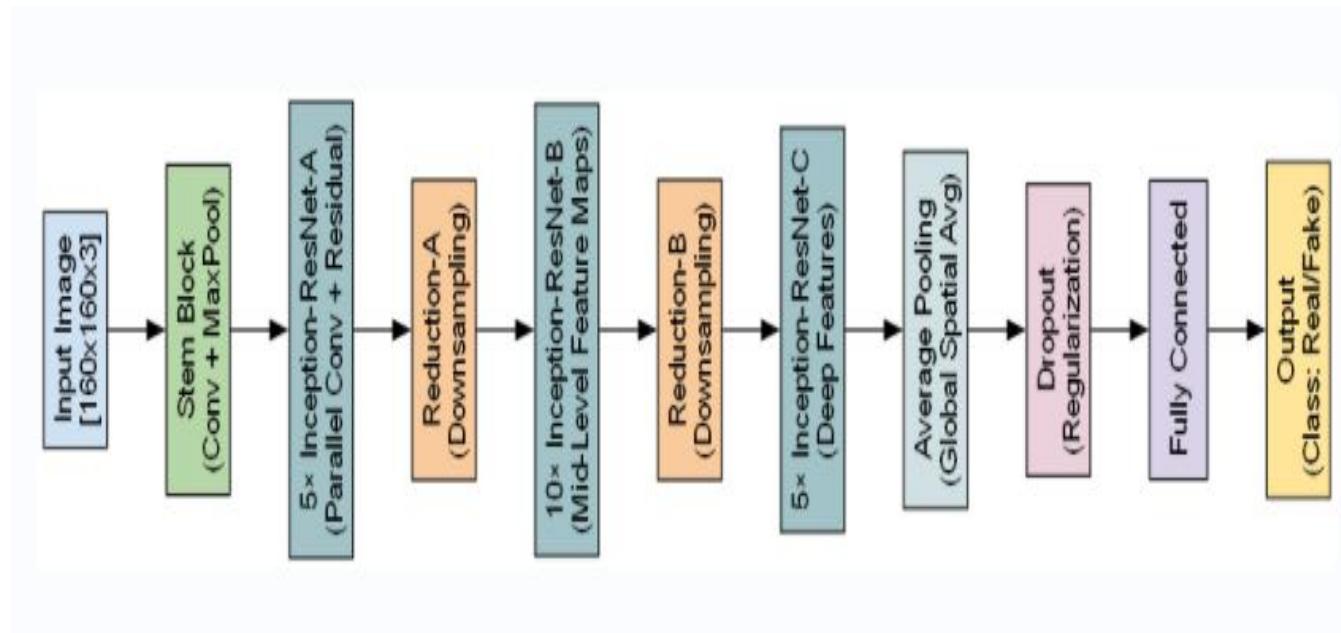


Figure 4.7 Inception-ResNet V1 structure

4.3 System Execution Pipeline and Directory Structure

The System Execution Pipeline outlines the sequence of steps the system follows to process data, train the model and generate predictions. It defines how each component of the system interacts with others, ensuring smooth data flow and task execution.

4.3.1 System Execution Pipeline

The project follows an execution pipeline that breaks down the deepfake detection task into clearly defined stages from dataset acquisition and preprocessing to training and deployment. This step-by-step approach ensures a systematic and modular workflow, allowing each component to be developed, tested, and improved independently. The components are organized in a structured file directory to improve readability, maintainability, and scalability.

The execution pipeline followed:

dataset extraction → preprocessing → training → evaluation → deployment.

This well-defined pipeline not only ensures repeatability of results but also allows for easy debugging and updates at any stage of the project lifecycle.

4.3.2 Directory Structure

A total of nine Python scripts were developed and utilized throughout the project, stored within the project's root directory named “**FINALYEARPROJECT**”. These scripts represent various stages of the deepfake detection pipeline. While some are core components critical to the system's functionality, others serve supplementary roles to enhance automation, data handling and visualization. The directory structure follows as seen below.

FINALYEARPROJECT/

```
|  
|   └── Constants.py (storing fixed parameters and paths used across scripts)  
|   └── trainthemodel2.py (training script utilizing the InceptionResnetV1 model)  
|   └── evaluation.py (evaluating the model using metrics)  
|   └── plot.py (generating plots of training/validation performance.)  
|   └── __init__.py (treat directories as Python packages.)  
|  
└── dataset/
```

```

|   └── Faces.py (extracting and labelling face images from video files)
|   └── Faceforensics.py (accessing the FaceForensics++ dataset)
|   └── Constants.py
|   └── preprocessing.py (processing the extracted face images)
|   └── balancefakeimages.py (balancing the dataset by cleaning or adjusting)
|   └── __init__.py
|
|   └── faces/
|       └── real/
|           └── *.jpg ← Extracted real face images
|       └── fake/
|           └── *.jpg ← Extracted fake face images
|       └── faces.csv ← CSV with image names and labels

```

4.4 Evaluation Metrics

To evaluate the performance of the trained deepfake detection model, several standard metrics were used. These metrics offer a comprehensive view of the model's ability to accurately differentiate between real and fake face images.

4.4.1 Loss Function

During training and validation, Binary Cross Entropy (BCE) Loss was used to evaluate the alignment between the model's predictions and the actual labels. This loss function is suitable for binary classification, as it strongly penalizes wrong predictions and pushes the model to generate predictions that are confidently close to 0 or 1.

4.4.2 Accuracy

This reflects the percentage of correctly classified images out of all predictions. Accuracy was monitored during both training and validation to offer a measure of the model's overall performance. However, this does not serve as an effective evaluation metric.

4.4.3 Confusion Matrix

A confusion matrix was generated after training to analyse how predictions were distributed across the two classes: real (labelled 1) and fake (labelled 0). It reveals true positives, true negatives, false positives, and false negatives providing an insight into how and where the model may be making mistakes.

4.4.4 Classification Report

A classification report was generated containing Precision (the proportion of predicted real/fake labels that were actually correct), Recall (the proportion of actual real/fake labels that were correctly predicted), F1-Score (the mean of precision and recall, offering a balance between the two). This report is helpful in identifying whether the model is favouring one class over the other and how well it generalizes across both.

4.5 Training Results

Table 4.1 Training results

EPOCH	TRAINING	TRAINING	VALIDATION	VALIDATION
	LOSS	ACCURACY	LOSS	ACCURACY
1	0.1867	0.9154	0.1680	0.9223
2	0.1100	0.9544	0.0966	0.9601
3	0.0817	0.9669	0.1067	0.9569
4	0.0721	0.9702	0.0912	0.9628
5	0.0650	0.9731	0.0855	0.9641
6	0.0583	0.9754	0.0801	0.9657
7	0.0529	0.9770	0.0779	0.9665
8	0.0485	0.9783	0.0754	0.9673
9	0.0453	0.9795	0.0732	0.9682
10	0.0421	0.9808	0.0710	0.9691

EPOCH	TRAINING	TRAINING	VALIDATION	VALIDATION
	LOSS	ACCURACY	LOSS	ACCURACY
11	0.0399	0.9816	0.0701	0.9695
12	0.0376	0.9823	0.0692	0.9698
13	0.0354	0.9830	0.0688	0.9702
14	0.0336	0.9838	0.0684	0.9705
15	0.0319	0.9843	0.0681	0.9708
16	0.0304	0.9849	0.0679	0.9710
17	0.0290	0.9854	0.0677	0.9712
18	0.0278	0.9859	0.0675	0.9713
19	0.0266	0.9863	0.0673	0.9714
20	0.0255	0.9867	0.0671	0.9715

Table 4.1 presents the model's performance metrics across 20 epochs during training and validation. It includes the training loss, training accuracy, validation loss, and validation accuracy for each epoch, providing a clear overview of how the model's performance improved as training progressed. These metrics are crucial for evaluating the effectiveness of the model, ensuring it is both learning from the training data and generalizing well to the validation set.

4.5.1 Loss vs Epoch

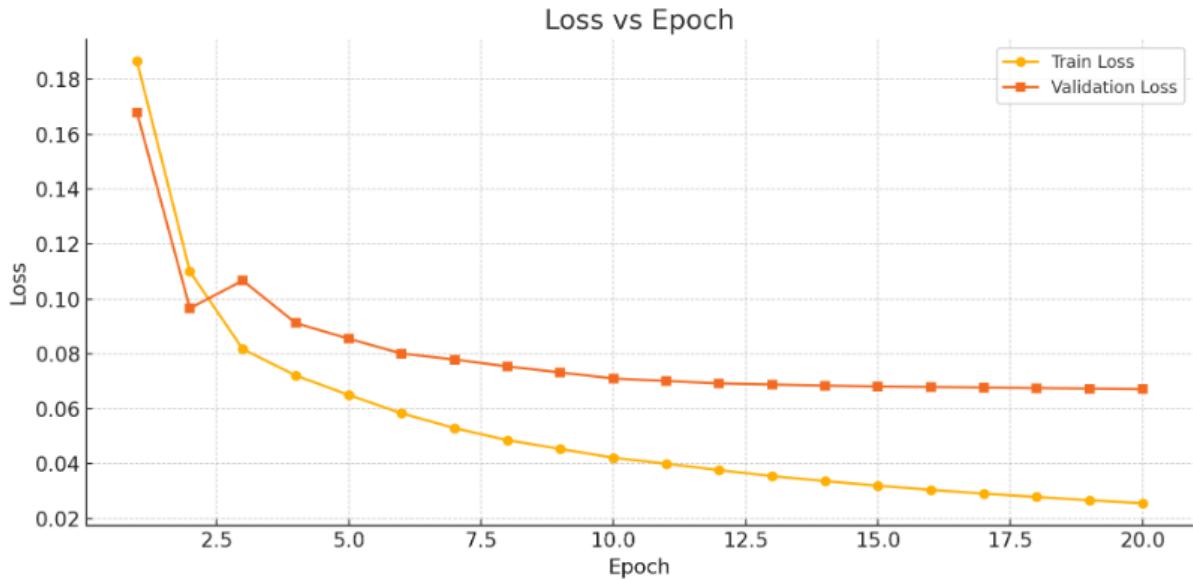


Figure 4.8 Loss vs Epoch Plot

The Loss vs Epoch plot in **Figure 4.8** illustrates the training and validation loss performance over 20 epochs. The training loss decreased from an initial value of 0.1867 to 0.0255, while the validation loss dropped from 0.1680 to 0.0671. This decrease in both training and validation losses indicates effective learning and convergence of the model. The close alignment of the curves suggests that the model is not overfitting the training data.

4.5.2 Accuracy vs Epoch

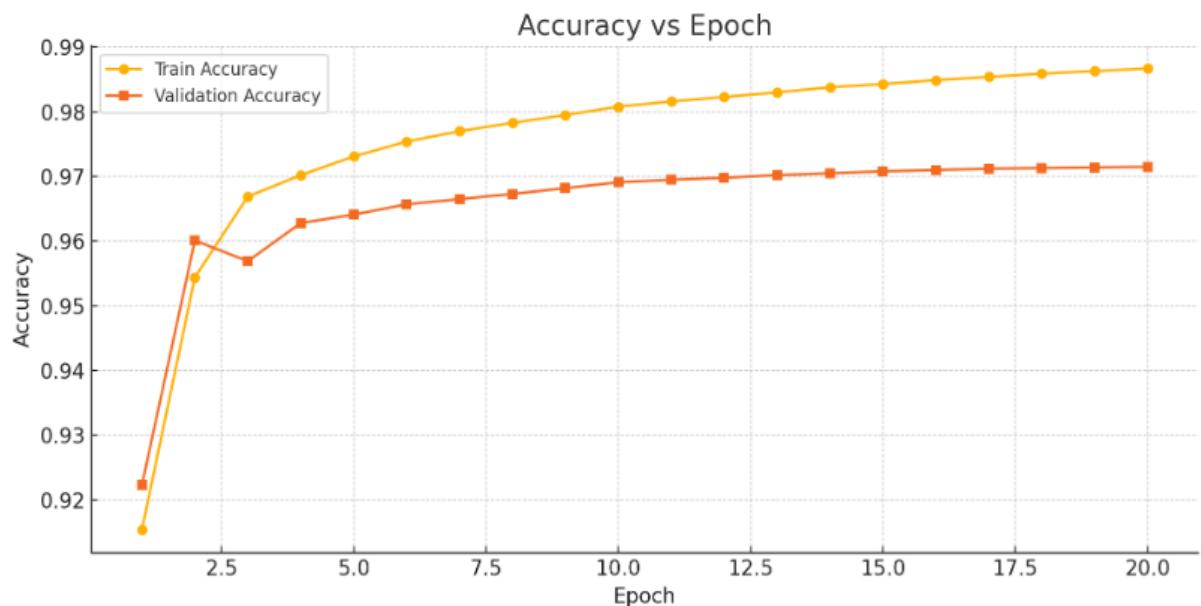


Figure 4.9 Accuracy vs Epoch Plot

The Accuracy vs Epoch plot in **Figure 4.9** displays the accuracy achieved on both training and validation sets during each epoch. The training accuracy rose from 91.54% to 98.67%, while validation accuracy improved from 92.23% to 97.15%. The upward trend in both training and validation accuracy indicates that the model is learning the distinguishing features between real and fake faces effectively. The small and consistent gap between training and validation accuracy further suggests minimal overfitting.

4.5.3 Confusion Matrix

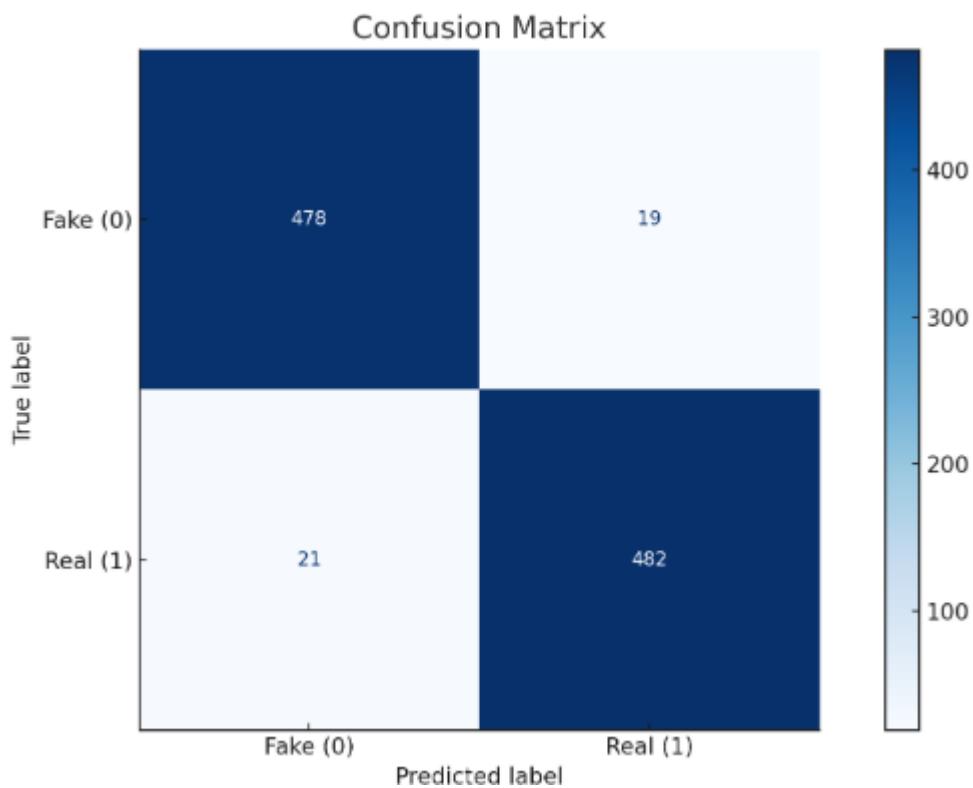


Figure 4.10 Confusion Matrix

The confusion matrix in **Figure 4.10** visualizes the model's classification performance on the validation dataset. It quantifies true positives (real images correctly classified), true negatives (fake images correctly classified), false positives (fake images misclassified as real), and false negatives (real images misclassified as fake). The model demonstrates balanced classification across both classes. The true positive and true negative values in the blue coloured

diagonals (482 and 478) are significantly higher than the misclassifications in the white coloured diagonals (21 and 19).

4.5.4 Classification Report

Table 4.2 Classification Report

CLASS	PRECISION	RECALL	F1-SCORE	SUPPORT
FAKE (0)	0.96	0.96	0.96	497
REAL (1)	0.96	0.96	0.96	503

The classification report provides precision, recall, and F1-score metrics for both classes. The model achieved 96% precision, 96% recall, and 96% F1-score for both real and fake labels. The close similarity of metrics across both classes confirms that the model handles class balance well and performs reliably across various input types.

4.6 Deployment and Testing

This section describes the comprehensive deployment process of the deepfake detection model, followed by its testing and the strategies for continuous monitoring and maintenance.

4.6.1 Model Deployment with Streamlit

Streamlit is a Python based web application framework designed for creating data science and machine learning interfaces with minimal effort. Using Streamlit, a lightweight and interactive web application was created to allow users upload an image (usually a facial image) and receive a classification result as “Real” or “Deepfake”. The web application was powered by a python file named app.py which references Google Drive where the final model was stored. This script was uploaded alongside several other supporting files to GitHub from where the web application was linked to Streamlit and deployed (anie-akan.streamlit.app). The system performs real-time classification on images. **Figure 4.11** shows the

GitHub repository for deploying the model while **Figure 4.12** shows the model deployment interface on the web application.

The screenshot shows the GitHub repository page for 'deepfake-detector'. At the top, there's a navigation bar with links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the header, the repository name 'deepfake-detector' is displayed with a public status. A commit history table shows 22 commits from Charles04Ekanem, with the most recent being 'Update app.py' (c728340) from yesterday. The table includes columns for file changes, commit message, and time. To the right of the commit table, there's an 'About' section with a brief description: 'FINAL YEAR PROJECT ON DEEPFAKE DETECTION WITH CNN (USING PYTORCH)'. It also lists Readme, Activity, stars (0), watching (1), forks (0), and releases (none published). Below the commit table, there's a 'README' section and a large preview area for 'deepfake-detector'.

Figure 4.11 GitHub repository

The screenshot shows the Streamlit app interface titled 'DEEPFAKE DETECTION SYSTEM'. On the left, there's a sidebar with a lock icon, 'INFO' (describing the system as a research tool for real vs. deepfake detection), 'HOW TO USE' (with three steps: upload image, get prediction, and review disclaimer), and 'MODEL AND DATASET' (mentioning InceptionV3 and FaceForensics++ datasets). The main area has a title 'DEEPFAKE DETECTION SYSTEM' with a small icon. It features a 'Choose an image...' input field with 'Drop and drop file here' and 'Browse files' buttons. Below this is a 'Disclaimer' section with a warning icon, stating the system is under development and not for medical use. It also includes a note about accuracy and a copyright notice. At the bottom of the sidebar, it says 'Powered by PyTorch + Streamlit'.

Figure 4.12 Model deployment on Streamlit App

4.6.2 Model Testing and Review

The final model was evaluated using previously unseen images to assess its generalization and overall functionality. Additionally, the deployed web

application was tested to examine the user experience, including the ease of navigation and responsiveness of the interface. Results from testing the system with previously unseen images are presented below in **Figures 4.13** and **4.14**, showcasing the model's performance in realistic, real-world scenarios.

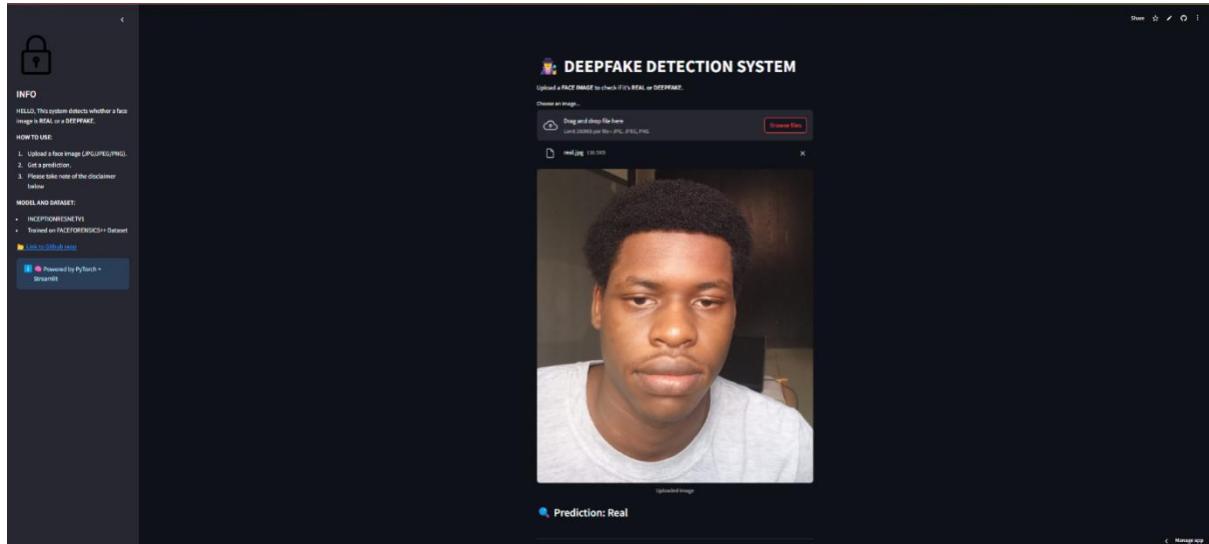


Figure 4.13 Model Testing on Real Image

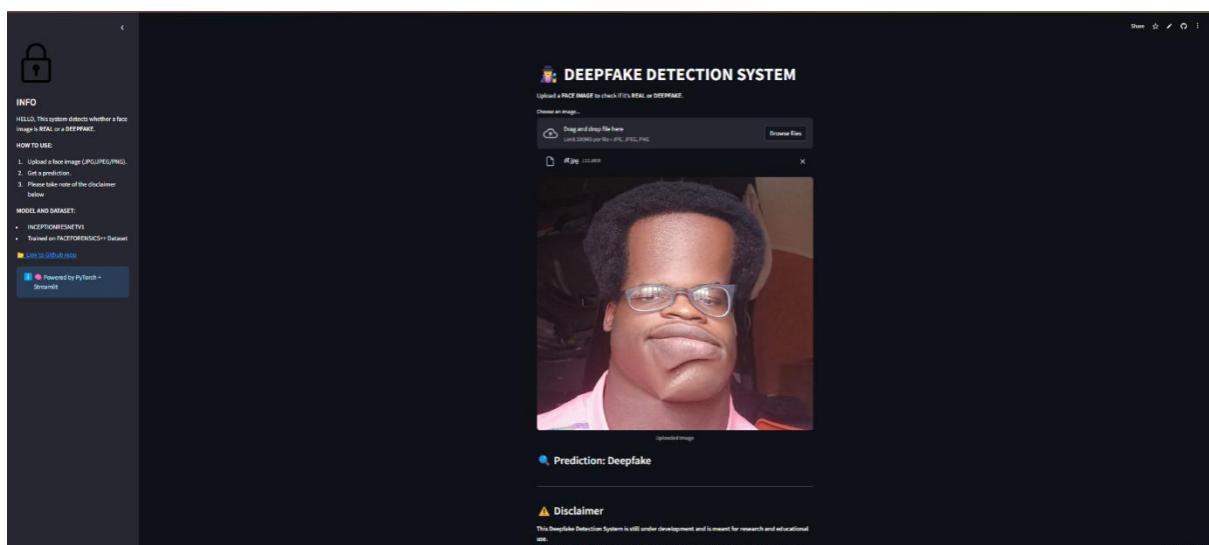


Figure 4.14 Model Testing on distorted Image

The system was tested with Test data from the earlier obtained Openforensics dataset consisting 5413 real images and 5492 fake images. Some screenshots from the results are attached below in **Figures 4.15** and **4.16**.

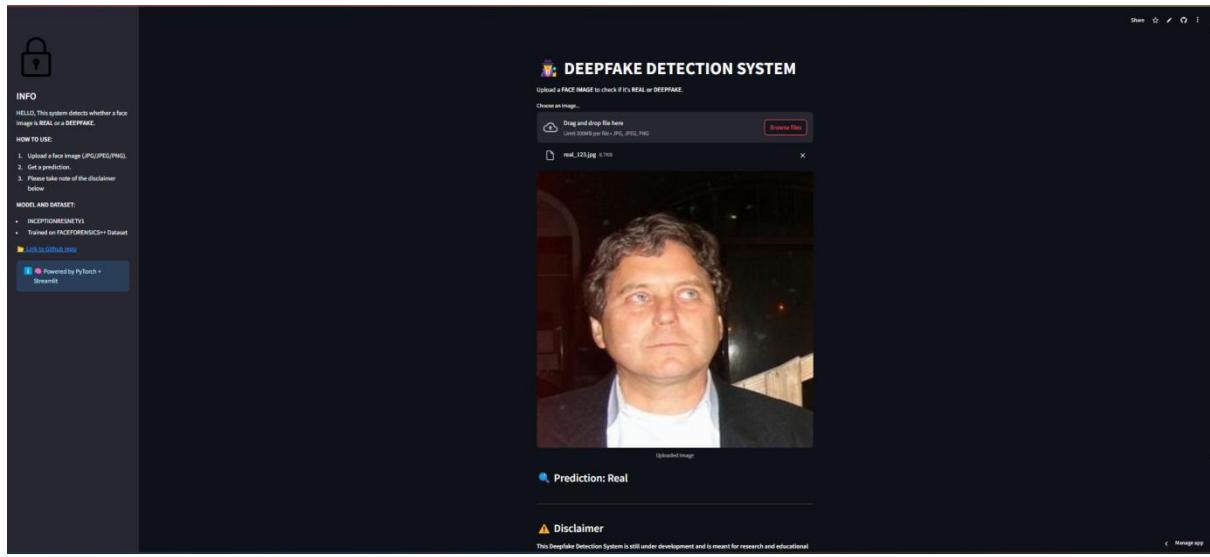


Figure 4.15 Model Testing on Real Image (Openforensics data)

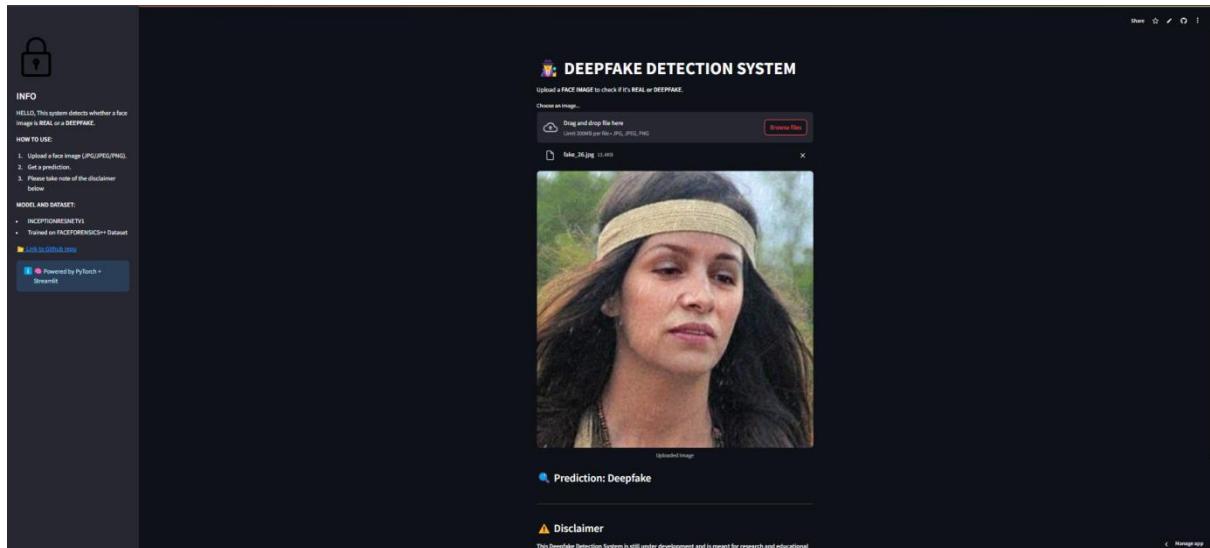


Figure 4.16 Model Testing on Fake Image (Openforensics data)

4.6.2.1 Strengths of the Deployment

The deployment of the deepfake detection model offers several key strengths that enhance its performance and overall effectiveness. Noticeable strengths include:

- 1 Cross-platform functionality across mobile and desktop devices as well as Windows and IOS operating systems.

- 2 Use of disclaimer and version control ensuring responsible use and collaboration.
- 3 Easy navigation and interface with clear instructions.
- 4 Defined upload formats for input images.

4.6.2.2 Weaknesses of the Deployment

Although the deployment of the deepfake detection model provides numerous advantages, there are certain weaknesses that could impact its overall effectiveness and user experience. Noticeable weaknesses include:

- 1 Lack of confidence percentage or score to show level of confidence in prediction.
- 2 There is no support for batch uploading or comparative analysis between multiple images.
- 3 No model explanation to explain the process for prediction.

4.6.2.3 Model misclassification

Model misclassification is a critical issue in deepfake detection as it directly impacts the system's reliability and effectiveness. Possible reasons for this include the following:

- 1 Mismatch between training and real-world data: The model was trained on datasets like FaceForensics++, Celeb-DF, and OpenForensics. If the input image looks different from the training data for example, due to lighting, resolution, compression, or camera angle, the model might have trouble making accurate predictions.
- 2 Low-Quality Images: Deepfake patterns are often subtle and compressed or blurry images can obscure these critical details, making it harder for the model to detect anomalies accurately.

- 3 Face Detection Errors: If the face is partially obscured, angled weirdly or cropped incorrectly during preprocessing, the model may not receive the relevant facial region, leading to misclassification.
- 4 Overfitting Challenge: Despite efforts to prevent overfitting, the model may still base predictions on dataset-specific patterns, rather than truly generalizable features, affecting its performance with real-world scenarios.
- 5 Single Frame Analysis: The model analyzes individual images, not sequences of frames from a video. Temporal inconsistencies that could help detect deepfakes are missed in this mode of operation.

4.7 Analysis of Results

The achieved 97.15% validation accuracy demonstrates a strong ability of the model to classify images correctly. This is particularly notable given that training was conducted entirely on CPU over just 20 epochs, showing that high accuracy is possible despite hardware limitations through proper model architecture selection and data preprocessing.

The validation loss of 0.0671 signifies low error in the model's predictions. This low loss, paired with high accuracy, is a positive indicator that the model is not only fitting the training data but also generalizing well to unseen data.

Observations from the confusion matrix show that the model maintained high true positive and true negative rates, reflecting balanced performance across both classes. The classifier did not show significant bias toward any class, due to the carefully shuffled and balanced dataset.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Introduction

This chapter presents the conclusion of the research project, offering a reflection on the entire work. It will cover a review of the project objectives, a summary of the findings, the scope of those findings, as well as recommendations and potential future directions. This chapter will provide a comprehensive summary of the effectiveness of the developed deepfake detection model and explore its potential applications in real-world scenarios.

5.2 Findings

This study investigated binary deepfake image classification using the InceptionResNetV1 architecture pre-trained on VGGFace2 and fine-tuned on a balanced dataset of 206,348 preprocessed facial images from the Celeb-DF dataset. The model achieved a final validation accuracy of 97.15% and a validation loss of 0.0671 after 20 epochs on CPU.

These results affirm the model's ability to extract high-quality features from facial images, even under hardware limitations. The high accuracy and low loss indicate that the model effectively generalized to the validation set, showing minimal signs of overfitting due to careful dataset balancing, preprocessing, and stratified 70/30 splitting. The evaluation metrics highlight the effectiveness of the approach in distinguishing real from fake facial images.

5.3 Scope of findings

This project showed that the deepfake detection model can perform well across different well-known datasets such as FaceForensics++, Celeb-DF V2, and OpenForensics. This means that the model isn't limited to just one dataset. One of the key improvements was balancing the number of real and fake face images

used during training. This helped prevent the model from favouring one class over the other and gave better accuracy and fairness in predictions.

The chosen model, Inception-ResNet V1, was also an important factor in achieving good results. Because it was already trained on a large face dataset (VGGFace2), it was able to learn detailed facial features that helped it detect fake faces more reliably. Finally, the model was deployed using a simple Streamlit web app, allowing users to upload images and get real time predictions. While the deployed app worked well in most cases, there were still some situations where the model struggled especially with low-quality images, blurry faces, or images that were heavily edited.

5.4 Contribution to knowledge

This project offers insights into model performance, deployment challenges and practical considerations providing a foundation for future research and development. It contributes to the growing body of knowledge in the field of deepfake detection by proposing an effective and practical approach to identifying manipulated media using facial features and deep learning. The contributions are:

- 1 **Balanced Deepfake Detection Dataset from Multiple Sources:** The use of a balanced image dataset (103,174 real vs. 103,174 fake) from imbalanced video sources helps to contribute a methodology for balancing large scale datasets which is vital for fair model evaluation and reducing bias.
- 2 **Lightweight FaceNet-Based Model for Binary Classification:** This project demonstrated how Inception-ResnetV1 which was originally designed for face recognition, can be adapted for binary deepfake classification.
- 3 **Efficient Preprocessing Pipeline:** A preprocessing pipeline that decouples image transformations from model training, improving training speed offers a contribution to efficient model training practices.

- 4 Multi-component Approach to Model Development: Breaking the model development process into component parts of constant definition, data extraction, preprocessing, training, evaluation etc help to improve readability and maintainability.
- 5 User-Friendly Deepfake Detection Web App: A developed Streamlit-based interface for non-technical users was deployed to test image authenticity which is a step towards deployable AI tools for public use.
- 6 Dataset Imbalance in Deepfake Detection: Several studies suffer from unbalanced real and fake data which introduces bias and skews performance. This project proposes a solution by implementing face extraction and controlled cleaning to maintain balance.
- 7 Lack of Lightweight Detection Models: Although deepfake detection often relies on heavy CNNs such as Xception and EfficientNet, this project employs the use of a lighter model (FaceNet) with comparable performance.
- 8 Realistic and Diverse Testing: Celeb-DF is known for more realistic deepfakes compared to older datasets. The inclusion of both Openforensics and Celeb-DF addresses the need for realism and diversity across deepfake detection datasets.
- 9 Deployment and Real-World Usability: Many papers focus solely on achieved accuracy without considering deployment of the model. The use of Streamlit app contributes toward narrowing the gap between research and practical adoption.

5.5 Recommendations

The following recommendations are proposed to address identified limitations and to guide future improvements in system performance and applicability:

- 1 Hardware Acceleration: Future implementations should utilize GPU (Graphics Processing Unit) or TPUs (Tensor Processing Unit) based

training to significantly reduce training time, allow for training across a greater number of epochs and a potentially improved convergence.

- 2 Longer Training Duration: 20 epochs on a CPU were sufficient for basic learning, but deeper convergence and better generalization would likely be achieved with about 40–100 epochs on a GPU.
- 3 Real-time Deployment: For deployment purposes, model compression may help run inference efficiently.
- 4 Dataset Expansion: Including the use of an ensemble of datasets (e.g., Deep Fake Detection Challenge (DFDC)) may help to improve generalizability of the model across various deepfake generation techniques.

5.6 Future improvements and direction

This section outlines potential enhancements and research directions aimed at advancing the system's effectiveness, scalability, and real-world use. They include the following:

- 1 Feature Learning: Integration of models such as ConvLSTM to capture frame-by-frame anomalies in deepfake videos could significantly boost accuracy.
- 2 Ensemble Learning Approach: Combining the strengths of different architectures such as Xception and EfficientNet via ensemble learning may work to improve robustness.
- 3 Detail-based Mechanisms: Introducing detail-based mechanisms to focus on facial regions most likely to be tampered with, such as the eyes, mouth, and facial boundaries may improve its ability.
- 4 Explainability Tools: Implementation of boundary box tools to provide visual insights into the decision-making process of the model may help to explain the reasoning behind its predictions.

5.7 Conclusion

This project successfully demonstrated a CPU-based approach to training a deepfake detection model using a pre-trained InceptionResNetV1 architecture. Despite computational limitations, the model achieved promising results with a final validation accuracy of 97.15% and showcased the use of lightweight deepfake detection pipelines. Through proper preprocessing, dataset balancing and structured evaluation, simple hardware can yield meaningful results.

Future versions of this project will work towards a more generalizable model that can transition from academic research to real-world applications in digital media verification, law enforcement, and online content moderation.

REFERENCES

- [1] R. K. et al., "Comparative study of deep learning techniques for deepfake video detection," *ICT Express*, vol. 10, no. 6, pp. 1226-1239, 2024.
- [2] Y. P. et al., "An improved dense CNN architecture for deepfake image detection," *IEEE Access*, vol. 11, pp. 22081-22095, 2023.
- [3] S. D. et al., "Mitigating adversarial attacks in deepfake detection: an exploration of perturbation and AI techniques," in *PETRA '23: Proc. 16th Int. Conf. Pervasive Technologies Related to Assistive Environments*, Corfu, 2023.
- [4] S. H. M. et al., "Deepring: convolution neural network based on blockchain technology," *Al-Mustansiriyah Journal of Science*, vol. 35, pp. 61-69, 2024.
- [5] M. S. R. et al., "Deepfakes - reality under threat?" in *2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)*, 2024.
- [6] O. Schwartz, "You thought fake news was bad? deep fakes are where truth goes to die," *The Guardian*, 12 Nov. 2018. [Online]. Available: <https://www.theguardian.com/technology/2018/nov/12/deep-fakes-fake-news-truth>. [Accessed: Dec. 11, 2024].
- [7] B. Cruz, "2024 deepfakes guide and statistics," 26 Sept. 2024. [Online]. Available: <https://www.security.org/resources/deepfake-statistics/#citations>. [Accessed: Dec. 11, 2024].
- [8] M. Taye, "Theoretical understanding of convolutional neural network: concepts, architectures, applications, future directions," *Computation*, 2023.
- [9] M. S. R. et al., "Advanced deepfake detection using machine learning algorithms: a statistical analysis and performance comparison," in *7th Int. Conf. on Information and Computer Technologies (ICICT)*, 2024.
- [10] O. Schwartz, "Artificial intelligence in digital media: the era of deepfakes," *IEEE Trans. Technol. Soc.*, vol. 1, no. 3, pp. 138-147, 2020.
- [11] C. Pizzuto, "From CGI to deepfakes: insights from hao li at the AI for good global summit," 7 Jul. 2024. [Online]. Available: <https://aiforgood.itu.int/from-cgi-to-deepfakes-insights-from-hao-li-at-the-ai-for-good-global-summit/>. [Accessed: Dec. 10, 2024].

- [12] C. B. et al., "Video rewrite: driving visual speech with audio," in *Proc. 24th Annu. Conf. Computer Graphics and Interactive Techniques*, 1997.
- [13] J. M. Norman, "Video rewrite, origins of deepfakes," 2017. [Online]. Available: <https://www.historyofinformation.com/detail.php?id=4792>. [Accessed: Dec. 11, 2024].
- [14] I. G. et al., "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139-144, 2014.
- [15] J. T. et al., "Face2face: real-time face capture and reenactment of rgb videos," in *Proc. CVPR*, 2016.
- [16] S. S. et al., "Synthesizing Obama: learning lip sync from audio," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1-13, 2017.
- [17] Reddit, "Update on r/deepfakes," 8 Feb. 2018. [Online]. Available: https://www.reddit.com/r/announcements/comments/7v3x6t/update_on_rdeepfakes/. [Accessed: Dec. 10, 2024].
- [18] Amazon AWS, "What is deep learning?" [Online]. Available: <https://aws.amazon.com/what-is/deep-learning/>. [Accessed: Dec. 12, 2024].
- [19] J. H. et al., "What is deep learning?" IBM, 17 Jun. 2024. [Online]. Available: <https://www.ibm.com/topics/deep-learning>. [Accessed: Dec. 11, 2024].
- [20] GeeksforGeeks, "Backpropagation in neural network," Jun. 2024. [Online]. Available: <https://www.geeksforgeeks.org/backpropagation-in-neural-network/>. [Accessed: Dec. 10, 2024].
- [21] Y. G. et al., "Towards public verifiable and forward-privacy encrypted search by using blockchain," *IEEE Trans. Dependable Secure Comput.*, pp. 2111–2126, 2023.
- [22] A. K. et al., "Detecting deepfake, FaceSwap and Face2Face facial forgeries using frequency CNN," *Multimed. Tools Appl.*, pp. 18461–18478, 2021.
- [23] B. C. et al., "FeatureTransfer: Unsupervised domain adaptation for cross-domain deepfake detection," *Security Commun. Netw.*, 2021.
- [24] J. et al., "Detecting compressed deepfake videos in social networks using frame-temporality two-stream convolutional network," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 3, pp. 1089–1102, 2022.

- [25] L. Guarnera, "Fighting deepfake by exposing the convolutional traces on images," *IEEE Access*, 2020.
- [26] C. K. C. et al., "Combating deepfakes: Multi-LSTM and blockchain as proof of authenticity for digital media," in *Proc. 2020 IEEE/ITU Int. Conf. Artificial Intelligence for Good*, 2020.
- [27] Aionlinecourse, "What is convolutional autoencoder," 2023. [Online]. Available: <https://www.aionlinecourse.com/ai-basics/convolutional-autoencoder>. [Accessed: Dec. 10, 2024].
- [28] M. M. et al., "Deepfakes generation and detection: State-of-the-art, open challenges, countermeasures, and way forward," *Appl. Intell.*, vol. 53, pp. 3974–4026, 2022.
- [29] A. K. et al., "Deepfake video detection using convolutional neural network," *Int. J. Adv. Trends Comput. Sci. Eng.*, 2020.
- [30] H. J. et al., "Rotor fault diagnosis method using CNN-based transfer learning with 2D sound spectrogram analysis," *Electronics*, vol. 12, no. 3, 2023.
- [31] J. K. et al., "Deepfakes: Trick or Treat?" *Business Horizons*, pp. 135–146, 2019.
- [32] N. K. et al., "Fooled twice – People cannot detect deepfakes but think they can," *iScience*, vol. 24, no. 11, 2021.
- [33] S. Cole, "We are truly fucked: Everyone is making AI-generated fake porn now," *Vice*, 24 Jan. 2018. [Online]. Available: <https://www.vice.com/en/article/reddit-fake-porn-app-daisy-ridley/>. [Accessed: Dec. 10, 2024].
- [34] M. Gurucharan, "Basic CNN architecture: Explaining 5 layers of convolutional neural network," 23 Sept. 2024. [Online]. Available: <https://www.upgrad.com/blog/basic-cnn-architecture/>. [Accessed: Dec. 10, 2024].
- [35] M. Waldrop, "Synthetic media: The real trouble with deepfakes," *Knowable Magazine*, 16 Mar. 2020. [Online]. Available: <https://knowablemagazine.org/content/article/technology/2020/synthetic-media-real-trouble-deepfakes>. [Accessed: Dec. 11, 2024].

- [36] L. S. et al., "Deepfake detection through key video frame extraction using GAN," in *Proc. 2022 Int. Conf. Autom., Comput. and Renew. Syst. (ICACRS)*, Pudukkottai, India, Dec. 13–15, 2022.
- [37] D. A. et al., "MesoNet: A compact facial video forgery detection network," in *Proc. 2018 IEEE Int. Workshop Inf. Forensics Secur. (WIFS)*, Hong Kong, China, Dec. 11–13, 2018.
- [38] R. S. Varghese et al., "Celeb-DF (v2)," [Online]. Available: <https://www.kaggle.com/datasets/reubensuju/celeb-df-v2>. [Accessed: Apr. 19, 2025].
- [39] D. Cozzolino et al., "FaceForensics++: Learning to detect manipulated facial images," *arXiv*, 2019. [Online]. Available: <https://arxiv.org/abs/1901.08971>
- [40] L. T. et al., "OpenForensics: Multi-face forgery detection and segmentation in-the-wild dataset," v1.0.0, [Online]. Available: <https://zenodo.org/records/5528418>. [Accessed: Apr. 19, 2025].
- [41] E. Altuncu, V. N. L. Franqueira, and S. Li, "Deepfake: Definitions, performance metrics and standards, datasets, and a meta-review," *Front. Big Data*, vol. 7, Art. no. 1400024, 2024.
- [42] H. F. Shahzad et al., "A review of image processing techniques for deepfakes," *Sensors*, vol. 22, no. 12, p. 4556, 2022.
- [43] S. Venkatesh et al., "Single image face morphing attack detection using ensemble of features," in *Proc. IAPR Int. Workshop on Biometrics*, 2020.
- [44] K. Raja et al., "Towards generalized morphing attack detection by learning residuals," *Image Vis. Comput.*, vol. 122, 2022.
- [45] P. Aghdiai et al., "Attention-aware wavelet-based detection of morphed face images," *arXiv preprint*, arXiv:2106.15686, 2021.
- [46] I. Medvedev et al., "MorDeepHy: Face morphing detection via fused classification," *arXiv preprint*, arXiv:2208.03110, 2022.
- [47] I. Castillo Camacho and K. Wang, "A comprehensive review of deep-learning-based methods for image forensics," *J. Imaging*, vol. 7, no. 4, Art. no. 69, Apr. 2021.

- [48] Y. Li, M. C. Chang, and S. Lyu, "In ictu oculi: Exposing AI-created fake videos by detecting eye blinking," in *Proc. IEEE Int. Workshop Inf. Forensics Secur. (WIFS)*, Hong Kong, China, Dec. 11–13, 2018. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8630787>. [Accessed: Jul. 10, 2025].
- [49] A. Rössler et al., "FaceForensics++: Learning to detect manipulated facial images," *arXiv*, 2019. [Online]. Available: <https://arxiv.org/abs/1901.08971>
- [50] D. M. Montserrat et al., "Deepfakes detection with automatic face weighting," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Seattle, WA, USA, Jun. 14–19, 2020, pp. 2851–2859.
- [51] W. M. Wubet, "The deepfake challenges and deepfake video detection," *Int. J. Innov. Technol. Explor. Eng.*, vol. 9, pp. 789–796, 2020. [Online]. Available: <https://www.ijitee.org/portfolio-item/E2779039520/>. [Accessed: Jul. 10, 2025].
- [52] A. Singh, A. S. Saimbhi, N. Singh, and M. Mittal, "DeepFake video detection: A time-distributed approach," *SN Comput. Sci.*, vol. 1, p. 212, 2020.
- [53] I. Masi, A. Killekar, R. M. Mascarenhas, S. P. Gurudatt, and W. AbdAlmageed, "Two-branch recurrent network for isolating deepfakes in videos," *arXiv*, 2020. [Online]. Available: <https://arxiv.org/abs/2008.03412>. [Accessed: Jul. 10, 2025].
- [54] A. Shende, S. Paliwal, and T. K. Mahay, "Using deep learning to detect deepfake videos," *Turk. J. Comput. Math. Educ.*, vol. 12, pp. 5012–5017, 2021.
- [55] P. Yadav, I. Jaswal, J. Maravi, V. Choudhary, and G. Khanna, "DeepFake detection using InceptionResNetV2 and LSTM," in *Proc. Int. Conf. Emerg. Technol.: AI, IoT, and CPS for Sci. Technol. Appl.*, Chandigarh, India, Sep. 6–7, 2021.
- [56] S. Suratkar and F. Kazi, "Deep fake video detection using transfer learning approach," *Arab. J. Sci. Eng.*, vol. 48, pp. 9727–9737, 2022.
- [57] K. Sooda, "DeepFake detection through key video frame extraction using GAN," in *Proc. Int. Conf. Autom., Comput. and Renew. Syst. (ICACRS)*, Pudukkottai, India, Dec. 13–15, 2022, pp. 859–863.
- [58] R. V. Saraswathi, M. Gadwalkar, S. S. Midhun, G. N. Goud, and A. Vidavaluri, "Detection of synthesized videos using CNN," in *Proc. Int. Conf. Augmented Intell. and Sustain. Syst. (ICAISS)*, Trichy, India, Nov. 24–26, 2022.

- [59] P. Saikia, D. Dholaria, P. Yadav, V. Patel, and M. Roy, "A hybrid CNN-LSTM model for video deepfake detection by leveraging optical flow features," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Padua, Italy, Jul. 18–23, 2022.
- [60] S. Saif, S. Tehseen, S. S. Ali, S. Kausar, and A. Jameel, "Generalized deepfake video detection through time-distribution and metric learning," *IT Prof.*, vol. 24, pp. 38–44, 2022.
- [61] M. S. Saealal, M. Z. Ibrahim, D. J. Mulvaney, M. I. Shapiai, and N. Fadilah, "Using cascade CNN-LSTM-FCNs to identify AI-altered video based on eye state sequence," *PLoS ONE*, vol. 17, e0278989, 2022.
- [62] Z. Lai, Y. Wang, R. Feng, X. Hu, and H. Xu, "Multi-feature fusion based deepfake face forgery video detection," *Systems*, vol. 10, no. 1, p. 31, 2022.
- [63] M. H. Malik, H. Ghous, S. Qadri, S. A. Nawaz, A. Anwar, and C. Author, "Frequency-based deep-fake video detection using deep learning methods," *J. Comput. Biomed. Inform.*, vol. 4, pp. 41–48, 2023.
- [64] M. T. Hasan Fuad, F. Bin Amin, and S. M. Masudul Ahsan, "Deepfake detection from face-swapped videos using transfer learning approach," in *Proc. 26th Int. Conf. Comput. Inf. Technol. (ICCIT)*, Cox's Bazar, Bangladesh, Dec. 13–15, 2023.
- [65] M. Nawaz, A. Javed, and A. Irtaza, "Convolutional long short-term memory-based approach for deepfakes detection from videos," *Multimed. Tools Appl.*, vol. 83, pp. 16977–17000, 2023.
- [66] A. M. Parayil, A. Masood, M. Ajas, R. Tharun, and K. Usha, "Deepfake detection using Xception and LSTM," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 5, pp. 9191–9196, 2023.
- [67] S. Patel, S. K. Chandra, and A. Jain, "DeepFake videos detection and classification using ResNeXt and LSTM neural network," in *Proc. 3rd Int. Conf. Smart Gener. Comput., Commun. and Netw. (SMART GENCON)*, Bangalore, India, Dec. 29–31, 2023.
- [68] P. Ritter, D. Lucian, Andries, and A. Chowanda, "Comparative analysis and evaluation of CNN models for deepfake detection," in *Proc. 4th Int. Conf. Artif. Intell. and Data Sci. (AiDAS)*, Ipoh, Malaysia, Sep. 6–7, 2023, pp. 250–255.
- [69] C. Cunha et al., "An ensemble approach to facial deepfake detection using self-supervised features," 2024, pp. 37–44.

- [70] A. Mitra, S. P. Mohanty, P. Corcoran, and E. Kougianos, “A novel machine learning based method for deepfake video detection in social media,” in *Proc. 2020 IEEE Int. Symp. Smart Electron. Syst. (iSES)*, Chennai, India, Dec. 14–16, 2020, pp. 91–96, doi: 10.1109/iSES50453.2020.00031.
- [71] S. Suratkar, E. Johnson, K. Variyambatt, M. Panchal, and F. Kazi, “Employing transfer-learning based CNN architectures to enhance the generalizability of deepfake detection,” in *Proc. IEEE Int. Conf. Comput. Commun. Control Network Technol. (ICCCNT)*, Jul. 2020, pp. 1–9, doi: 10.1109/ICCCNT49239.2020.9225400.
- [72] R. Caldelli, L. Galteri, I. Amerini, and A. Del Bimbo, “Optical flow based CNN for detection of unlearnt deepfake manipulations,” *Pattern Recognit. Lett.*, vol. 146, pp. 31–37, Jun. 2021, doi: 10.1016/j.patrec.2021.03.005.
- [73] E. O. Agu and S. T. Dennis, “Deepfake detection using convolution neural network,” *FUW Trends in Sci. Technol. J.*, pp. 194–201, 2023.
- [74] A. Heidari, D. Javaheri, S. Toumaj, N. Jafari Navimipour, M. Rezaei, and M. Ünal, “A new lung cancer detection method based on the chest CT images using federated learning and blockchain systems,” *Artif. Intell. Med.*, vol. 141, p. 102572, May 2023, doi: 10.1016/j.artmed.2023.102572.
- [75] A. Heidari, N. J. Navimipour, H. Dağ, S. Talebi, and M. Ünal, “A novel blockchain-based deepfake detection method using federated and deep learning models,” *Cognitive Computation*, vol. 16, no. 3, pp. 1073–1091, 2024, doi: 10.1007/s12559-024-10255-7.

APPENDIX A

DATASET SAMPLES

This appendix contains screenshots of dataset samples for this deepfake detection project.

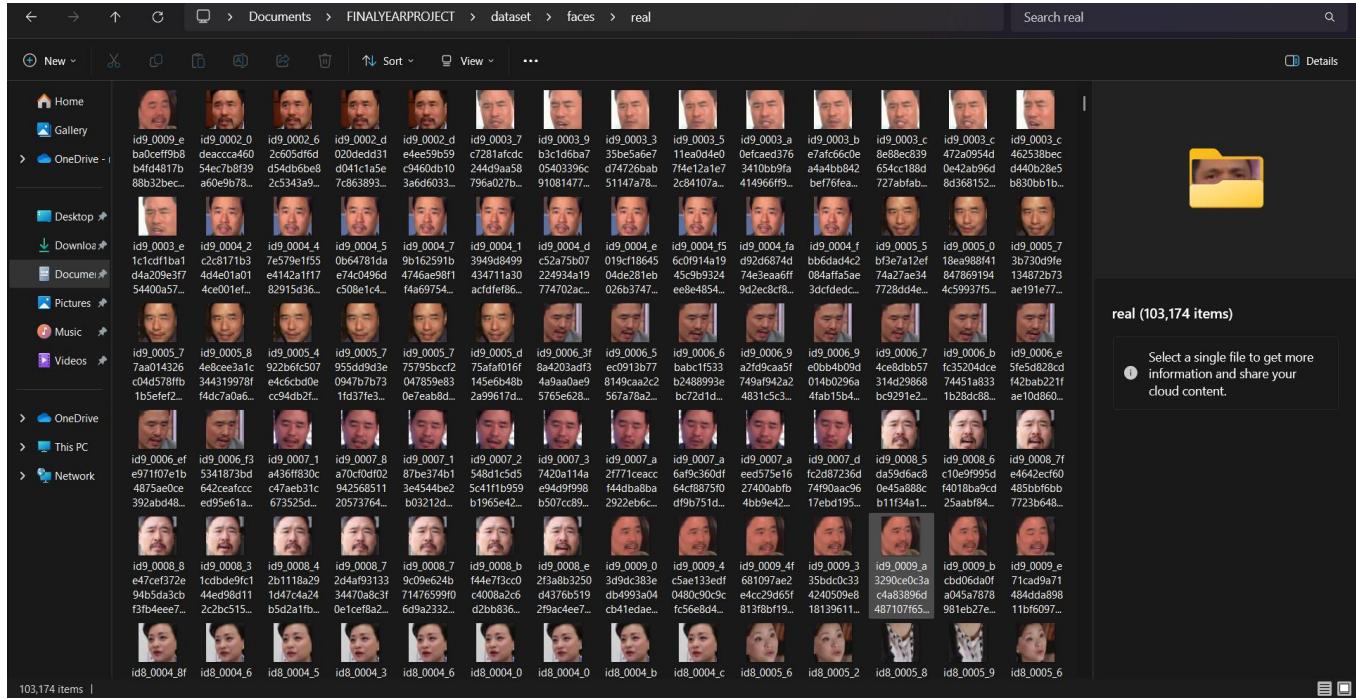


Figure A.1 Real images sample from dataset

Figure A.1 displays a screenshot of images from the real image dataset used in the deepfake detection model.

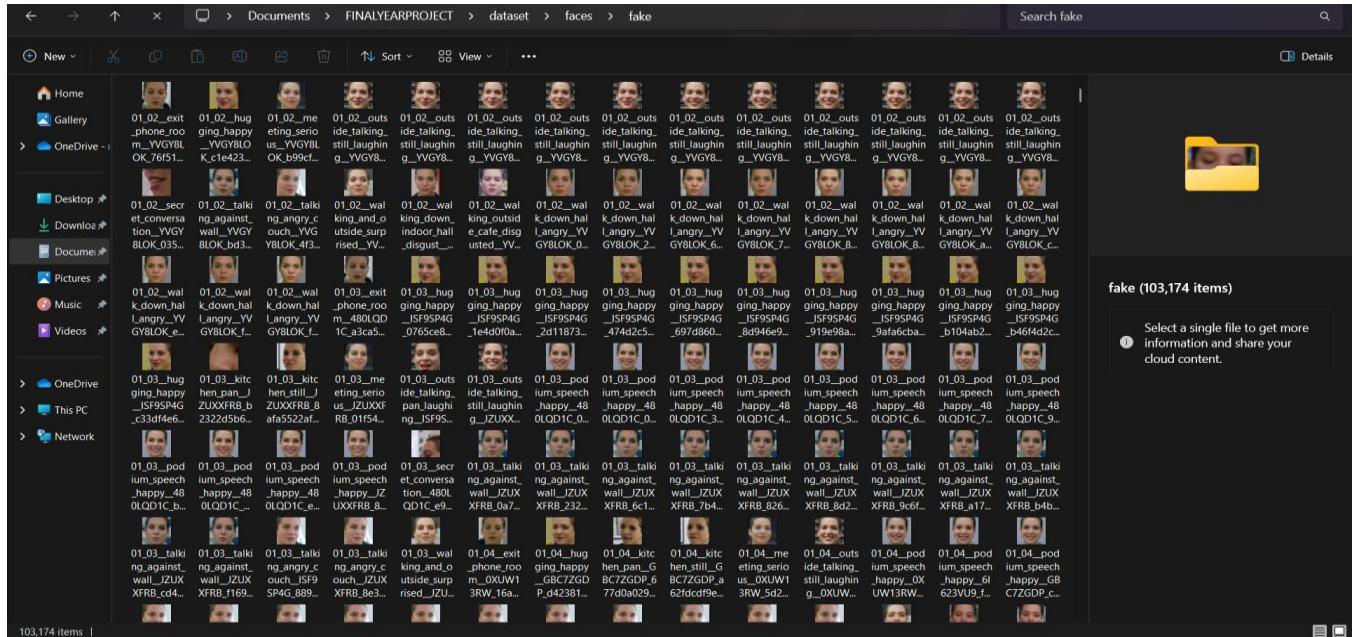


Figure A.2 Fake images sample from dataset

Figure A.2 displays a screenshot of images from the fake image dataset used in the deepfake detection model.

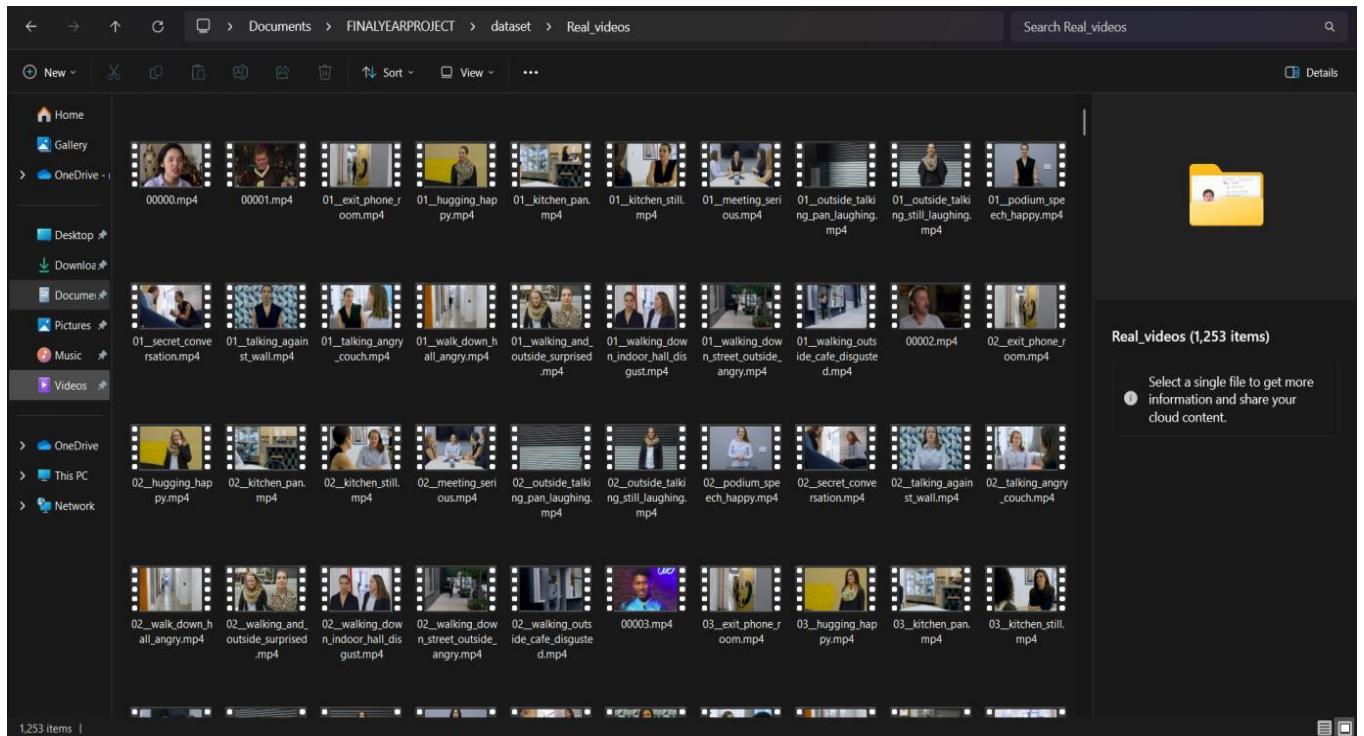


Figure A.3 Real videos

Figure A.3 displays a screenshot of real videos from where real images were extracted to create the real images folder.

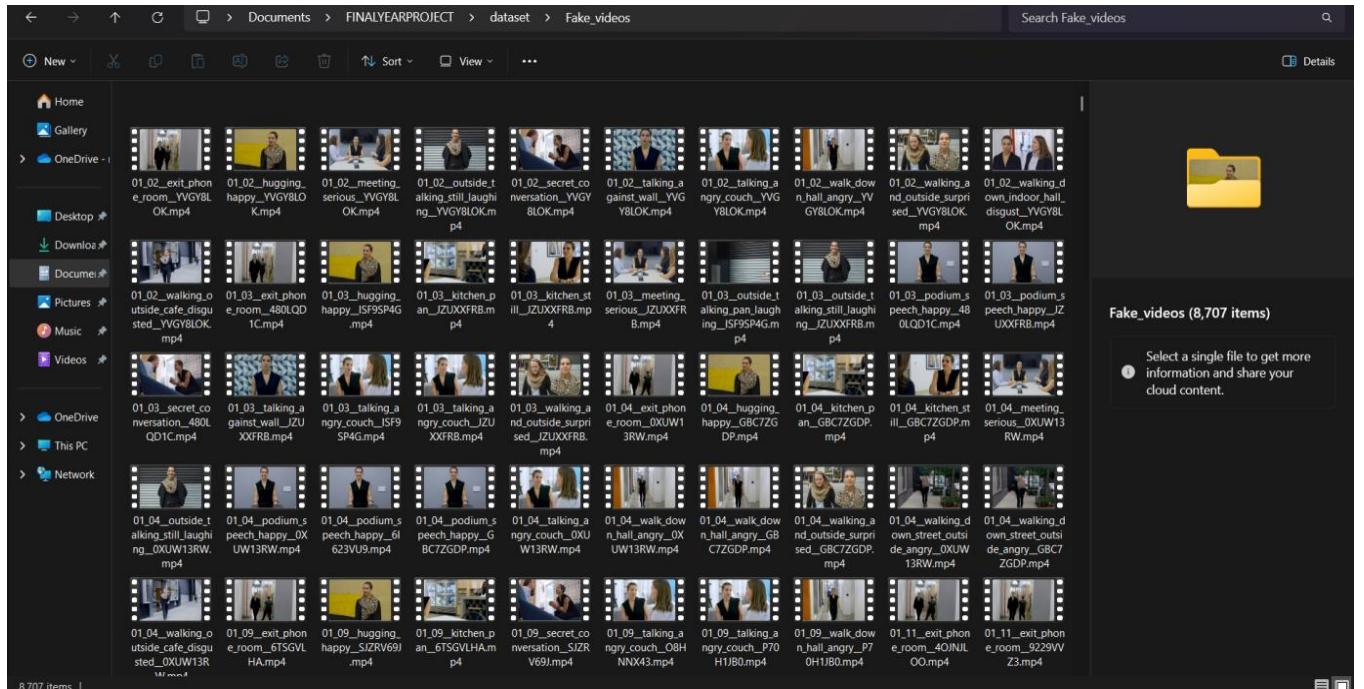


Figure A.4 Fake videos

Figure A.4 displays a screenshot of fake videos from where fake images were extracted to create the fake images folder.

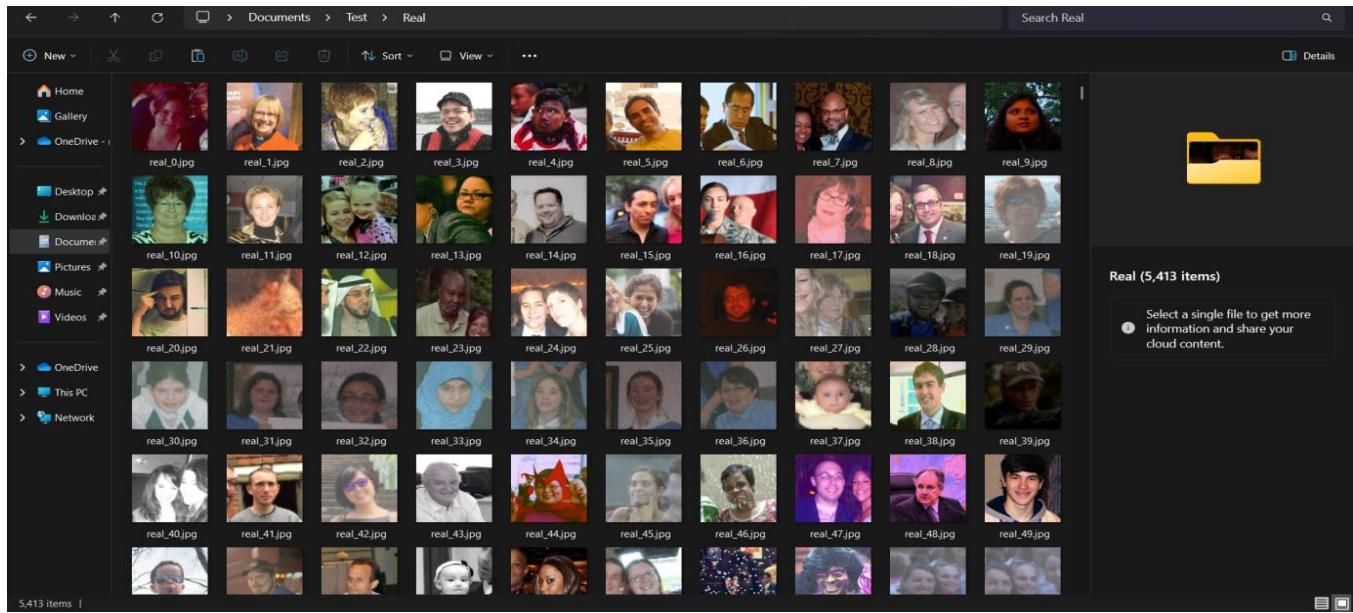


Figure A.5 Openforensics real test images

Figure A.5 displays a screenshot of images from the Openforensics real images for testing the deepfake detection model.

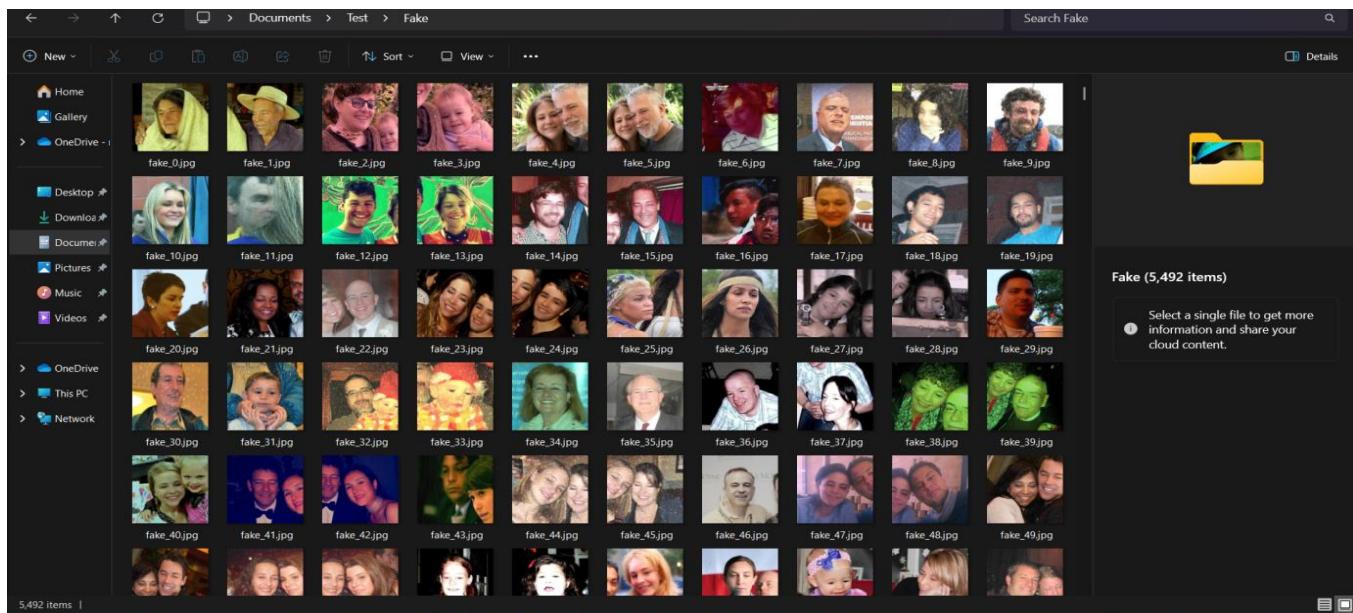


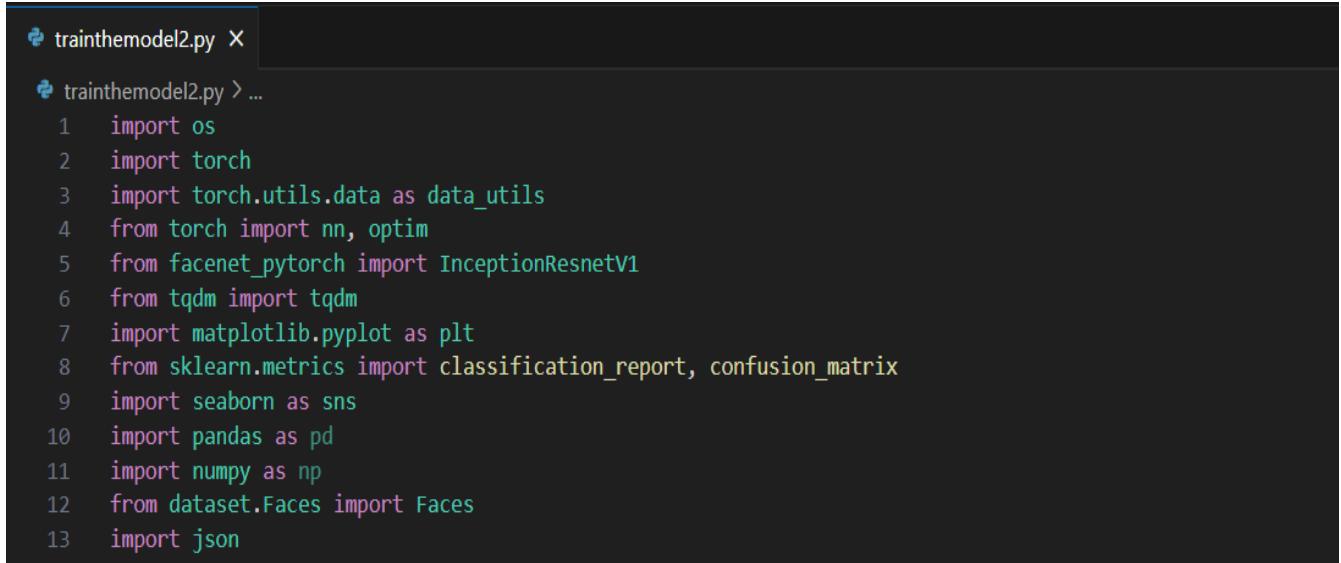
Figure A.6 Openforensics fake test images

Figure A.6 displays a screenshot of images from the Openforensics fake images for testing the deepfake detection model.

APPENDIX B

TRAINING SCRIPT SCREENSHOTS

This appendix contains screenshots of the python training script used in training the deepfake detection model as well as descriptions.



```
trainthemodel2.py X
trainthemodel2.py > ...
1 import os
2 import torch
3 import torch.utils.data as data_utils
4 from torch import nn, optim
5 from facenet_pytorch import InceptionResnetV1
6 from tqdm import tqdm
7 import matplotlib.pyplot as plt
8 from sklearn.metrics import classification_report, confusion_matrix
9 import seaborn as sns
10 import pandas as pd
11 import numpy as np
12 from dataset.Faces import Faces
13 import json
```

Figure B.1 Library Imports

Figure B.1 presents a screenshot of imported libraries used in developing the deepfake detection model. OS and Json allow for file and progress management, Torch allows for deep learning with PyTorch, Pandas and Numpy allow for data manipulation, Matplotlib and Seaborn allow for data visualization, Tqdm allows for progress bars during training, Facenet_pytorch allows use of InceptionResnetV1, sklearn.metrics allows for classification report and confusion matrix import as evaluation metrics.

```

15 # Configuration
16 DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
17 FACES_FOLDER = r"C:\Users\CHARLES EKANEM\Documents\FINALYEARPROJECT\dataset\faces"
18 FACES_CSV = os.path.join(FACES_FOLDER, "balanced_faces_temp.csv")
19 OUTPUT_DIR = r"C:\Users\CHARLES EKANEM\Documents\FINALYEARPROJECT\models"
20 os.makedirs(OUTPUT_DIR, exist_ok=True)
21 CLASS_NAMES = {0: "fake", 1: "real"}
22 SPLIT_RATIO = 0.7
23 PROGRESS_FILE = os.path.join(OUTPUT_DIR, "training_progress.json")
24
25 CONFIG = dict(
26     epochs=20,
27     batch_size=4,
28     learning_rate=0.01,
29     save_path=os.path.join(OUTPUT_DIR, "best_model.pth")
30 )
31

```

Figure B.2 Configuration

Figure B.2 presents a screenshot of the system's configuration settings. It includes a dictionary that defines the hyperparameters such as the number of epochs, batch size, and learning rate. The configuration outlines the directory paths required ensuring an organized training pipeline.

```

32 # Load Data
33 def get_data_loaders():
34     print("[INFO] Loading dataset...")
35     dataset = Faces(root_dir=FACES_FOLDER, csv_file=FACES_CSV, transform=True)
36     indices = torch.randperm(len(dataset))
37     split_idx = int(SPLIT_RATIO * len(dataset))
38     train_idx, val_idx = indices[:split_idx], indices[split_idx:]
39
40     train_loader = data_utils.DataLoader(data_utils.Subset(dataset, train_idx),
41                                         batch_size=CONFIG['batch_size'], shuffle=True)
42     val_loader = data_utils.DataLoader(data_utils.Subset(dataset, val_idx),
43                                         batch_size=CONFIG['batch_size'], shuffle=False)
44
45     print(f"[INFO] Training set size: {len(train_idx)}")
46     print(f"[INFO] Validation set size: {len(val_idx)}")
47     return train_loader, val_loader
48
49 # Save Progress

```

Figure B.3 Data Loading

Figure B.3 presents a screenshot of the system's data loading process as well as progress saving after each epoch. Line 35 loads the face dataset with

transformations. Lines 36 to 38 randomly shuffles and splits the dataset according to the previously defined split ratio of 0.7. Lines 40 to 43 creates data loaders for the training and validation sets.

```

48
49     # Save Progress
50     def save_progress(epoch):
51         with open(PROGRESS_FILE, 'w') as f:
52             json.dump({"last_completed_epoch": epoch}, f)
53
54     # Load Progress
55     def load_progress():
56         if os.path.exists(PROGRESS_FILE):
57             with open(PROGRESS_FILE, 'r') as f:
58                 return json.load(f).get("last_completed_epoch", -1)
59         return -1
60

```

Figure B.4 Training progress management

Figure B.4 presents a screenshot of the system's progress management. It includes two functions, save progress and load progress, which work to save each epochs' progress and load up the last completed epoch.

```

62     def train_model(model, train_loader, val_loader, criterion, optimizer, epochs):
63         history = {"train_loss": [], "train_acc": [], "val_loss": [], "val_acc": []}
64         best_val_acc = 0
65         best_model_state = None
66         start_epoch = load_progress() + 1
67
68         try:
69             for epoch in range(start_epoch, epochs):
70                 model.train()
71                 train_loss, train_acc = 0, 0
72                 for x, y in tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs} - Training"):
73                     x, y = x.to(DEVICE), y.to(DEVICE)
74                     optimizer.zero_grad()
75                     out = model(x).squeeze()
76                     loss = criterion(out, y)
77                     loss.backward()
78                     optimizer.step()
79                     preds = (torch.sigmoid(out) > 0.5).float()
80                     train_loss += loss.item()
81                     train_acc += (preds == y).float().mean().item()
82
83                 model.eval()
84                 val_loss, val_acc = 0, 0
85                 all_preds, all_labels = [], []
86                 with torch.no_grad():
87                     for x, y in tqdm(val_loader, desc="Validating"):
88                         x, y = x.to(DEVICE), y.to(DEVICE)
89                         out = model(x).squeeze()
90                         loss = criterion(out, y)
91                         preds = (torch.sigmoid(out) > 0.5).float()
92                         val_loss += loss.item()
93                         val_acc += (preds == y).float().mean().item()
94                         all_preds.extend(preds.cpu().numpy())
95                         all_labels.extend(y.cpu().numpy())

```

Figure B.5 Training process

```

98     avg_val_loss = val_loss / len(val_loader)
99     avg_train_acc = train_acc / len(train_loader)
100    avg_val_acc = val_acc / len(val_loader)
101
102    history["train_loss"].append(avg_train_loss)
103    history["val_loss"].append(avg_val_loss)
104    history["train_acc"].append(avg_train_acc)
105    history["val_acc"].append(avg_val_acc)
106
107    print(f"[INFO] Epoch {epoch+1}: Train Loss={avg_train_loss:.4f}, Acc={avg_train_acc:.4f} | Val Loss={avg_val_loss:.4f}, Acc={avg_val_acc:.4f}")
108
109    # Save best model
110    if avg_val_acc > best_val_acc:
111        best_val_acc = avg_val_acc
112        best_model_state = model.state_dict()
113        torch.save(best_model_state, CONFIG["save_path"])
114        print(f"[INFO] New best model saved at epoch {epoch+1}")
115
116    # Save current epoch model
117    checkpoint_path = os.path.join(OUTPUT_DIR, f"epoch_{epoch+1}_model.pth")
118    torch.save(model.state_dict(), checkpoint_path)
119    save_progress(epoch)
120    print(f"[INFO] Checkpoint saved: {checkpoint_path}")
121
122 except KeyboardInterrupt:
123     print("[WARNING] Training interrupted. Saving model...")
124     torch.save(model.state_dict(), os.path.join(OUTPUT_DIR, "interrupted_model.pth"))
125     save_progress(epoch - 1)
126     print("[INFO] Progress saved. You can resume training later.")
127     return history, all_preds, all_labels
128
129 # Final best model save
130 torch.save(best_model_state, CONFIG["save_path"])
131 print(f"[INFO] Best model saved to {CONFIG['save_path']}")
132 return history, all_preds, all_labels
133

```

Figure B.6 Training process (continued)

Figure B.5 and **Figure B.6** present screenshots of the training process function. This function handles the model's training and validation loop, tracks performance metrics across epochs, saves progress, checkpoints, and stores the best-performing model based on validation accuracy. The `model.train()` method performs the forward pass, loss computation, backpropagation and optimization. Immediately afterwards, the `model.eval()` method evaluates the model on the validation set for a given epoch. Lines 102 to 107 saves and displays the average metrics after every epoch. Lines 109 to 120 saves the best model after every epoch whilst also creating a checkpoint of every epoch. Lines 122 to 127 implement a fail-safe to handle both intentional and unintentional keyboard interrupts during training. If an interruption occurs, the current model state is saved and the training progress is logged, allowing training to be resumed later without loss of previously completed work.

```

134 # Plotting
135 def plot_metrics(history, all_preds, all_labels):
136     # Loss vs Epoch
137     plt.plot(history["train_loss"], label="Train Loss")
138     plt.plot(history["val_loss"], label="Validation Loss")
139     plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend()
140     plt.title("Loss vs Epoch")
141     plt.savefig(os.path.join(OUTPUT_DIR, "loss_vs_epoch.png"))
142     plt.close()
143
144     # Accuracy vs Epoch
145     plt.plot(history["train_acc"], label="Train Accuracy")
146     plt.plot(history["val_acc"], label="Validation Accuracy")
147     plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend()
148     plt.title("Accuracy vs Epoch")
149     plt.savefig(os.path.join(OUTPUT_DIR, "accuracy_vs_epoch.png"))
150     plt.close()
151
152     # Confusion Matrix
153     cm = confusion_matrix(all_labels, all_preds)
154     sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["Fake", "Real"], yticklabels=["Fake", "Real"])
155     plt.title("Confusion Matrix")
156     plt.xlabel("Predicted"); plt.ylabel("True")
157     plt.savefig(os.path.join(OUTPUT_DIR, "confusion_matrix.png"))
158     plt.close()
159
160     # Classification Report
161     report = classification_report(all_labels, all_preds, target_names=["Fake", "Real"])
162     with open(os.path.join(OUTPUT_DIR, "classification_report.txt"), "w") as f:
163         f.write(report)
164     print("[INFO] Classification report saved.")

```

Figure B.7 Model evaluation and plotting

Figure B.7 presents a screenshot of the evaluation and plotting function. This function plots three plots following the training process namely; Loss vs Epoch, Accuracy vs Epoch and Validation Confusion Matrix. This function also creates a classification report for the model.

```

165
166 # Run
167 if __name__ == "__main__":
168     model = InceptionResnetV1(pretrained="vggface2", classify=True, num_classes=1).to(DEVICE)
169     criterion = nn.BCEWithLogitsLoss()
170     optimizer = optim.Adam(model.parameters(), lr=CONFIG['learning_rate'])
171
172     train_loader, val_loader = get_data_loaders()
173     history, all_preds, all_labels = train_model(model, train_loader, val_loader, criterion, optimizer, CONFIG['epochs'])
174     plot_metrics(history, all_preds, all_labels)
175     print("[INFO] Training complete.")
176

```

Figure B.8 Script execution

Figure B.8 presents a screenshot of the main script block which performs the overall coordination of the training pipeline. It initializes the model (InceptionResnetV1), defines the loss function and optimizer, loads the preprocessed dataset, executes the training routine, and plots performance metrics.

APPENDIX C

MODEL ARCHITECTURE

This appendix includes scripts that define the architecture and structure of the InceptionResNetV1 model used for deepfake detection. The provided scripts give insight into how the model layers and parameters are structured.

```
⌚ print_model_arch.py > ...
1   from facenet_pytorch import InceptionResnetV1
2
3   model = InceptionResnetV1(pretrained='vggface2').eval()
4
5   print(model)
6
```

Figure C.1 Script for displaying InceptionResNet V1 model architecture

The script screenshot in **Figure C.1** imports the pre-trained InceptionResNetV1 model from the facenet_pytorch library, loads it with weights trained on the VGGFace2 dataset and prints its complete architecture.

```
⌚ print_model_arch.py > ...
1   from torchsummary import summary
2   from facenet_pytorch import InceptionResnetV1
3
4   model = InceptionResnetV1(pretrained='vggface2').eval()
5
6   summary(model, input_size=(3, 160, 160))
7
```

Figure C.2 Script for displaying layer-wise model summary of InceptionResNetV1

The script screenshot in **Figure C.2** displays a detailed summary of the InceptionResNetV1 architecture using the torchsummary library. It includes the output dimensions and number of parameters for each layer, helping to understand the computational complexity and structure of the model.

A screenshot of a terminal window in a code editor. The code in the terminal is:

```
print_model_arch.py > ...
1   from facenet_pytorch import InceptionResnetV1
2
3   model = InceptionResnetV1(pretrained='vggface2').eval()
4
5   total_params = sum(p.numel() for p in model.parameters())
6   print(f"Total Parameters: {total_params:,}")
```

The terminal tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS. The output in the terminal shows:

```
PS C:\Users\CHARLES_EKANEM\Documents\FINALYEARPROJECT> & "c:\Users\CHARLES_EKANEM\AppData\Local\Temp\script_16448\print_model_arch.py"
● Total Parameters: 27,910,327
○ PS C:\Users\CHARLES_EKANEM\Documents\FINALYEARPROJECT>
```

Figure C.3 InceptionResNet V1 model parameters

Figure C.3 shows the total number of learnable parameters in the InceptionResNetV1 model, which is 27,910,327. The model's parameters include weights and biases for convolutional layers, fully connected layers, and batch normalization layers highlighting the model's complexity.