Deepfake Detection using Deep Neural Networks (DNN)

LY Major Project-B Report

Submitted in partial fulfillment of the requirements of the Degree of Bachelor of Technology in Computer Engineering

by

Harsh Chotaliya

Shubham Kanojiya

Adil Khatri

Supervisor

Dr. Mandar Bivalkar

Department of Computer Engineering

K. J. Somaiya Institute of Technology

An Autonomous Institute permanently affiliated to University of Mumbai

Ayurvihar, Sion, Mumbai -400022

2023-24

Deepfake Detection using Deep Neural Networks (DNN)

LY Major Project-B Report

Submitted in partial fulfillment of the requirements of the Degree of Bachelor of Technology in Computer Engineering

by

Harsh Chotaliya (B-39)

Shubham Kanojiya (B-42)

Adil Khatri (B-43)

Supervisor

Dr. Mandar Bivalkar

Department of Computer Engineering

K. J. Somaiya Institute of Technology
An Autonomous Institute permanently affiliated to University of Mumbai
Ayurvihar, Sion, Mumbai -400022

2023-24

CERTIFICATE

*This is to certify that the project entitled "**Deepfake Detection using Deep Neural Networks (DNN)**" is bonafide work **Harsh Chotaliya, Shubham Kanojiya, Adil Khatri** submitted to the University of Mumbai in partial fulfillment of the requirement in Major Project, for the award of the degree of "Bachelors of Technology" in "Computer Engineering".*

Dr. Mandar Bivalkar
Project Guide
Department of Computer Engineering

Dr. Sarita Ambadekar
Head of Department
Dept. of Computer Engineering

Dr. Vivek Sunnapwar
Principal, KJSIT

Place: Sion, Mumbai-400022
Date: 3rd May, 2024

PROJECT APPROVAL FOR L. Y.

This project report entitled **Deepfake Detection using Deep Neural Networks (DNN)** by

Harsh Chotaliya (B-39)
Shubham Kanojiya (B-42)
Adil Khatri (B-43)

is an approved Last Year Major Project **in Computer Engineering**.

Examiners

1._____

Name and Signature
External Examiner

2._____

Name and Signature
Internal Examiner

Place: Sion, Mumbai-400022
Date: 3rd May, 2024

DECLARATION

We declare that this written submission represents our ideas in our own words and where other's ideas or words have been included, we have adequately cited and referenced the sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Harsh Chotaliya _____

Shubham Kanojiya _____

Adil Khatri _____

Date: 3rd May, 2024

ACKNOWLEDGEMENT

Before presenting our LY major project work entitled "**Deepfake Detection using Deep Neural Networks (DNN)**", we would like to convey our sincere thanks to the people who guided us throughout the course for this project work.

First, we would like to express our immense gratitude towards our **Dr. Mandar Bivalkar** for the constant encouragement, support, guidance, and mentoring at the ongoing stages of the project and report.

We would like to express our sincere thanks to our **H.O.D. Dr. Sarita Ambadekar**, for the encouragement, co-operation, and suggestions progressing stages of the report.

We would like to express our sincere thanks to our beloved Principal **Dr. Vivek Sunnapwar** and Vice principal **Dr. Sunita Patil** for providing various facilities to carry out this project.

Finally, we would like to thank all the teaching and non-teaching staff of the college, and our friends, for their moral support rendered during the course of the reported work, and for their direct and indirect involvement in the completion of our report work, which made our endeavor fruitful.

Place: Sion, Mumbai-400022
Date: 3rd May, 2024

ABSTRACT

The exponential growth in computational power has enabled deep learning algorithms to produce remarkably realistic human-synthesized videos, commonly referred to as deep fakes. These deep fakes, with their ability to seamlessly swap faces and fabricate events, pose serious challenges in various domains such as politics, security, and personal privacy. In response to these challenges, we present a novel deep learning-based approach designed to accurately differentiate between AI-generated fake images/videos and authentic ones.

Our methodology leverages the power of Artificial Intelligence (AI) to counter the adversities posed by AI itself. In our study and project, we present a multifaceted approach to deepfake detection, leveraging a custom CNN model and a pretrained Xception model for image analysis, along with an InceptionV3-based architecture tailored for video assessment. By integrating these models, we tackle a broad array of deepfake manipulation techniques, encompassing both traditional facial replacement and reenactment scenarios. Our comprehensive methodology enables robust detection and classification of manipulated visual content, contributing to the ongoing efforts to combat the proliferation of deceptive media in digital platforms.

To validate the real-world efficacy of our method, we conduct extensive evaluations on a diverse dataset comprising a blend of balanced and mixed sources, including datasets like Face-Forensic++, Deepfake Detection Challenge (DFDC). By simulating real-time scenarios and optimizing for robustness, our system demonstrates competitive performance while maintaining simplicity in implementation. These results underscore the effectiveness of our approach in combating the challenges posed by AI-generated fake images and videos across various domains.

CONTENTS

LIST OF FIGURE

LIST OF TABLES

LIST OF ABBREVIATIONS

Sr. No	Abbreviation	Description
1	CNN	Convolutional Neural Network
2	RNN	Recurrent Neural Network
3	DNN	Deep Neural Network
4	GAN	Generative Adversarial Networks
5	LSTM	Long Short-Term Memory

CHAPTER 1
INTRODUCTION

Deep fake technology, rooted in neural network tools like GANs (Generative Adversarial Networks) and Auto Encoders, has revolutionized human image synthesis. By superimposing target images onto source videos using deep learning techniques, deep fake creators craft incredibly realistic videos that defy visual detection. However, in our endeavor, we unveil a pioneering deep learning-based methodology adept at discerning AI-generated fake videos from authentic ones. Leveraging the inherent limitations of deep fake creation tools as a distinguishing factor, we capitalize on the subtle yet detectable artifacts they leave behind in the frames. These distinctive artifacts, imperceptible to the human eye yet discernible to trained neural networks, are effectively captured by our system using Convolutional Neural Networks (CNNs) [1].

Our approach hinges on extracting frame-level features using Res-Next CNNs, subsequently training a Xception model based on Inception v3 CNN to classify videos based on manipulation presence. This dual-stage process not only enhances the accuracy of detection but also enables real-time decision-making, crucial for applications in dynamic environments. Furthermore, our methodology is robustly validated against a diverse collection of deep fake videos, drawn from renowned datasets like Face-Forensic++, Deepfake Detection Challenge, and Celeb-DF. This extensive evaluation ensures the reliability and effectiveness of our system across a spectrum of deep fake scenarios, encompassing variations in data sources and manipulation techniques.

To bolster the real-time applicability of our model, we amalgamate various datasets, including YouTube data, enabling the system to learn from a wide array of video content. This comprehensive training regimen not only improves the model's adaptability but also ensures competitive accuracy in detecting deep fakes in real-time scenarios. Thus, our work not only contributes to the ongoing efforts in combating AI-generated fake videos but also underscores the importance of adaptive and robust deep fake detection systems in contemporary digital environments.

1.1 Problem Definition

The past few decades have witnessed the evolution of digital image and video manipulations, often showcased through sophisticated visual effects.

However, recent advancements in deep learning have ushered in a new era characterized by remarkably realistic fake content, commonly known as AI-synthesized media or deep fakes. While creating deep fakes using artificially intelligent tools has become relatively straightforward, detecting these deceptive creations presents a significant challenge. There is a historical record of deep fakes being utilized to fuel political tensions, fabricate terrorism events, propagate revenge porn, and exploit individuals through blackmail, underscoring the urgent need for robust detection mechanisms. Our initiative focuses on addressing this pressing issue by employing an InceptionV3 based artificial neural network for deep fake detection. This problem statement emphasizes the critical nature of identifying deep fakes to prevent their proliferation across social media platforms and mitigate their potentially harmful consequences.

1.2 Aim and Objectives

- Our project aims at discovering the distorted truth of the deep fakes.
- Our project will reduce the Abuses' and misleading of the common people on the world wide web.
- Our project will distinguish and classify the video as deepfake or pristine.
- Provide an easy-to-use system used to upload the video and distinguish whether the video is real or fake.

1.3 Organization of the Report

The organization of DeepFake Detection Techniques using Deep Neural Networks project can be divided into the following components:

1. Project Planning
    - Define project scope, objectives, and deliverables.
    - Establish a project timeline and allocate resources.
2. Requirements Gathering:
    - Define the functional and security requirements.
    - Determine the algorithms to be used.
    - Document user stories and use cases.
3. Design and Prototyping:
    - Create wireframes and design mock-ups for the user interface.
    - Develop a prototype to test the model accuracy and its functionality.
    - Review and refine the design based on feedback.
4. Development:
    - Write the code for the web application.

- Integrate all features like detection, extraction, models, etc.
- Continuously test and debug the application during development.

5. Testing:
    - Conduct thorough testing, including unit, integration, and security testing.
    - Identify and address any vulnerabilities, bugs, or performance issues.
6. Deployment:
    - Prepare the application for a phased rollout.
    - Monitor the app's performance and user feedback.

CHAPTER 2
REVIEW OF LITERATURE

The literature review delves into a myriad of strategies employed in the intricate domain of deep fake detection. One significant avenue explored involves the analysis of facial artifacts, as exemplified by the Face Warping Artifacts method. This technique intricately compares the generated facial elements with their surrounding context through a dedicated Convolutional Neural Network (CNN), offering insights into the nuanced distortions present in deep fakes. On a parallel front, the Detection by Eye Blinking approach takes a unique stance by considering eye blinking patterns as pivotal indicators of deep fakes' authenticity. Leveraging a Long-term Recurrent Convolutional Network (LRCN), this method delves into the temporal dynamics of eye blinks, thereby enriching the classification process.

A distinctive trajectory is witnessed in the realm of Capsule Networks, where the focus extends to detecting forged content across varied scenarios, albeit with potential considerations around the impact of random noise during training. Meanwhile, Recurrent Neural Network (RNN) methodologies in deep fake detection underscore the importance of sequential frame processing, albeit acknowledging potential limitations stemming from dataset sizes and diversity. Another intriguing avenue explored lies in Synthetic Portrait Videos utilizing Biological Signals, a terrain that involves the extraction and analysis of biological signals from facial regions to discern deep fakes from authentic content. This intricate process involves spatial coherence assessments, temporal consistency checks, and the integration of feature vectors and photoplethysmography (PPG) maps, albeit with noted challenges in preserving biological signals.

However, this method confronts hurdles in preserving biological signals due to a lack of discriminator, necessitating the formulation of a differentiable loss function to navigate the complexities of signal processing effectively. Collectively, these diverse methodologies encapsulate the evolving landscape of deep fake detection, each contributing unique insights and confronting distinct challenges, thereby enriching the broader discourse on combating digital deception. Techniques like eye blinking patterns and spatial coherence assessments contribute to the nuanced understanding of deep fake identification. Challenges arise in preserving biological signals amidst the complex signal processing required for accurate detection, highlighting the ongoing need for refinement and innovation in detection methodologies.

In a recent study, [19] proposed a novel deepfake detection system that integrates cutting-edge hardware components, including Nvidia GeForce RTX 30 Series GPUs, for enhanced performance. This system employs CUDA 11.3 and PyTorch, a powerful machine learning framework developed by Meta AI, to harness GPU capabilities efficiently. By leveraging ResNext50 for feature extraction, LSTM models with GAN technology for classification and evaluation using a confusion matrix to measure accuracy.

The model's architecture is illustrated in detail, emphasizing the utilization of ResNext50's parallel stacking and LSTM's classification capabilities. Dropout probability and latent dimensions are specified for LSTM layers, and the model's output is processed through linear and adaptive average pooling layers before final classification via softmax.

Through rigorous testing on unseen data from the FF++ dataset, the system achieves an impressive accuracy of 97.25% for detecting deepfake videos, particularly notable for videos with specific compression levels.

[18] presents a comprehensive methodology for detecting deepfake videos, a critical challenge in contemporary media manipulation. Initially, the study employed facial image cropping techniques, leveraging the dlib module in Python to extract facial images frame by frame from the training dataset. Subsequently, three prominent convolutional neural network (CNN) architectures, namely InceptionResNetV2, MobileNet, and DenseNet121, were employed for deepfake detection.

To enhance the models' performance, pre-trained weights from ImageNet were utilized for the initial layers, followed by a GlobalAveragePooling2D layer and a dense layer with softmax activation. The models were then compiled using the Adam optimizer with carefully chosen hyperparameters, including a learning rate of 1e-5 and zero decay. Evaluation metrics included the binary cross-entropy loss function and the accuracy matrix.

The research conducted training, validation, and testing using the Kaggle Deepfake Detection Challenge (DFDC) dataset, partitioning it into 70% for training, 20% for validation, and 10% for testing purposes. Additionally, results were cross-validated with a modified version of the FaceForensics++ dataset to ensure robustness and comparability.

For result analysis, the study adopted a systematic approach, extracting fixed-length images from videos to feed into the trained models. Confusion matrices were generated to visualize and interpret the models' performance. Evaluation metrics such as accuracy, precision, and

recall were computed, revealing notable accuracies of 93.7%, 94.93%, and 93.86% for InceptionResNetV2, MobileNet, and DenseNet121, respectively, on the DFDC dataset.

[20] explores several strategies for detecting deepfake videos and carefully analyzes their effectiveness through detailed result analysis.

Residual Neural Network (ResNet-50): This method employs a sophisticated neural network, ResNet-50, which is adept at extracting features from videos and classifying them as real or fake. By training on vast datasets of both genuine and manipulated videos, ResNet-50 can learn to discern subtle differences between authentic and fake content. Its ability to capture features at different scales is particularly valuable for identifying deepfakes.

Pooling Layers: Utilizing pooling layers, specifically average pooling, helps in filtering out irrelevant information from video frames. This process contributes to enhancing the accuracy of the detection model by focusing on the most relevant features.

Long Short-Term Memory (LSTM): LSTM, a type of recurrent neural network, is employed to analyze the temporal consistency of video frames. By considering the sequential nature of video data, LSTM can effectively identify patterns and deviations characteristic of deepfake videos.

Hybrid Architecture: The research proposes a hybrid approach that combines the strengths of ResNet50 and LSTM. While ResNet50 extracts high-level features from individual frames, LSTM models the temporal dependencies between these frames. This integrated approach aims to improve detection accuracy, especially for videos created using sophisticated techniques like Generative Adversarial Networks (GANs).

Result analysis demonstrates the efficacy of the proposed methodologies. Testing the model for 20 and 40 epochs yields accuracy rates of 84.75% and 87.48%, respectively. The validation and testing accuracy improve with increasing epoch count, as observed from the generated graphs. Additionally, confusion matrices aid in assessing the model's performance in testing scenarios.

2.2 Literature Survey

Table 2.1 : Comparison Between Existing Technologies

Sr. No	*Reference of paper*	*Dataset in use*	*Methods used*	*Accuracy*
1	[6]	HOHA dataset	1.CNN (Convolution Neural Networks). 2.LSTM (Long Short-Term Memory).	Conv-LSTM (with 20 frames) 96.7%, Conv-LSTM (with 40 frames) 97.1%
2	[7]	FaceForensics++	1.LRP and LIME 2.Xception net(CNN)	90.17%
3	[8]	Face2Face	1.CNN (Convolution Neural Networks).	VGG16 81.61%, ResNet50 75.46%
4	[9]	AFW, FDDB, CelebA	1.Convolution Neural Networks (CNN)	Discrete- 95% and for continuous 74%
5	[10]	Face2Face, Reddit user deepfakes	1.CNN 2.LSTM	95%
6	[11]	1.Celeb-DF 2. FaceForensics++ 3.DeepFake Detection Challenge	1. LSTM 2. CNN	84% With Transfer Learning ,75% Without Transfer Learning
7	[12]	Face2Face, StarGAN, CycleGAN	Deep Neural Networks, LSTM, MesoNet	20 videos accuracy is 85%
8	[13]	High-quality 1080p HD video clips of 976 sequences	FlowNet-S CNN With SPMC layer	Method(F3) 36.71/0.96, M(F5)

CHAPTER 3
REQUIREMENT SPECIFICATION

3.1 Introduction

This document outlines a comprehensive project blueprint for the development of Deepfake video detection using neural networks. Its intended audience includes current and prospective developers engaged in this domain, along with project sponsors. The plan encompasses a detailed overview of system functionality, project scope as perceived by the Deepfake video detection team (including myself and my mentors), use case diagrams, data flow diagrams, activity diagrams, functional and non-functional requirements, risk assessment, mitigation strategies, development methodology, and metrics for progress tracking throughout the project lifecycle.

3.2 Hardware Requirements

Table 3.1 : Hardware Requirements

Sr. No.	Parameter	Minimum Requirement
1	Intel Xeon E5 2637	3.5 GHz
2	RAM	16 GB
3	Hard Disk	100 GB
4	Graphic Card	NVIDIA GeForce GTX Titan (12 GB RAM)

3.3 Software Requirements

- Operating System: Windows 10+
- Programming Language : Python 3.0 & above.
- Framework: Tensorflow 2.16.1, Python Flask
- Libraries : Sklearn, OpenCV, Face-recognition, Pandas, NumPy

3.4 Feasibility Study

A feasibility study considers various constraints within which the system should be implemented and operated. In this stage the resources needed for implementation such as computing equipment, manpower and costs are estimated. The estimates are compared with available resources and a cost benefit analysis of the system is made.

The main objectives of the feasibility study are to determine whether the project would be feasible in terms of the following categories:

3.4.1 Technical feasibility :

Since the android application uses software technologies and tools which are freely available and technical skills required can be easily manageable but requires normal computing and the system server must be adequate and manageable in future. So, it is found that the hardware and software meets the needs of the system. So, it's clear that the proposed project is technically feasible.

3.4.2 Economic Feasibility :

Economic feasibility attempts to weigh the costs of developing and implementing a new system, against the benefits that would gather from having the new system in place. This feasibility study gives the top management the economic justification for the new system. A simple economic analysis which gives the actual comparison of costs and benefits is much more meaningful in this case. In addition, this proves to be a useful point of reference to compare actual costs as the project progresses.

3.4.3 Operational Feasibility :

Since the android application is interactive and data driven, the user needs to be only a bit familiar with the software system backed with graphical explanations and that can easily be understood faster in time with usage.

3.5 Cost Estimation

Since we have a small team , less-rigid requirements, and a long deadline, we are using the organic COCOMO model.

1. Work Allocation: This metric quantifies the labor required to accomplish a task, typically measured in person-months units.

$$Effort\ Applied(E) = a_b(KLOC)^b$$

$$E = 2.4(20.5)^{1.05} \qquad (i)$$

$$E = 57.2206PM \qquad (ii)$$

2. Development Duration: This refers to the time necessary to complete a task, directly

correlated with the effort invested. It is typically measured in time units such as weeks or months.

$$DevelopmentTime(D) = cb(E)^{db}$$

$$D = 2.5(57.2206)^{0.38} \quad (iii)$$

$$D = 11.6M \quad (iv)$$

3. Team Size: This indicates the number of developers required to successfully complete the project.

$$PeopleRequired(P) = \frac{E}{D}$$

$$P = \frac{57.2206}{11.6} \quad (v)$$

$$P = 4.93 \quad (vi)$$

In the context of COCOMO, the Work Allocation metric plays a crucial role in quantifying the labor needed to accomplish specific tasks within the project. It is typically measured in person-months and is calculated based on the size of the project and other factors. Effort Applied, represented as E, is calculated using a formula that incorporates project size measured in thousands of lines of code (KLOC) and other coefficients. This metric provides insights into the amount of effort required for task completion, aiding in resource management and scheduling.

Additionally, the COCOMO model assists in estimating the Development Duration, which refers to the time required to complete the project tasks. This duration is directly related to the effort invested in the project and is usually measured in time units like weeks or months. By employing the DevelopmentTime formula, project managers can estimate the duration needed for project completion based on the effort calculated earlier. Moreover, COCOMO helps in determining the Team Size required for the project, indicating the number of developers needed to accomplish the tasks within the estimated time frame. This aspect ensures optimal resource utilization and team allocation throughout the project lifecycle.

CHAPTER 4
PROJECT ANALYSIS AND DESIGN

4.1 Introduction

Project analysis and design entail the systematic identification, definition, and organization of requirements and resources essential for project completion. This meticulous process ensures project feasibility, achievability, and efficiency. The analysis phase focuses on acquiring project details like goals, objectives, scope, and limitations to assess feasibility, risks, and challenges. Stakeholder engagement, including project sponsors, end-users, and technical specialists, plays a pivotal role in comprehensively understanding project needs during this phase.

4.2 Project Model Analysis

The selection of the Spiral model for our software development process stems from its emphasis on collaborative team dynamics, efficient workflow, and robust risk management strategies. This model's strength lies in its ability to accommodate changes seamlessly throughout the development lifecycle while consistently evaluating the product against predefined expectations. The Spiral model facilitates extensive client involvement, allowing them to prioritize features, participate in iteration planning and review sessions, and witness the continuous integration of new functionalities. However, it's important for clients to recognize that they are observing a work in progress, albeit with the added benefit of transparency. Given the complexity and inherent risks in our project, the Spiral model's capability to address these challenges effectively makes it the preferred choice for our product development strategy.

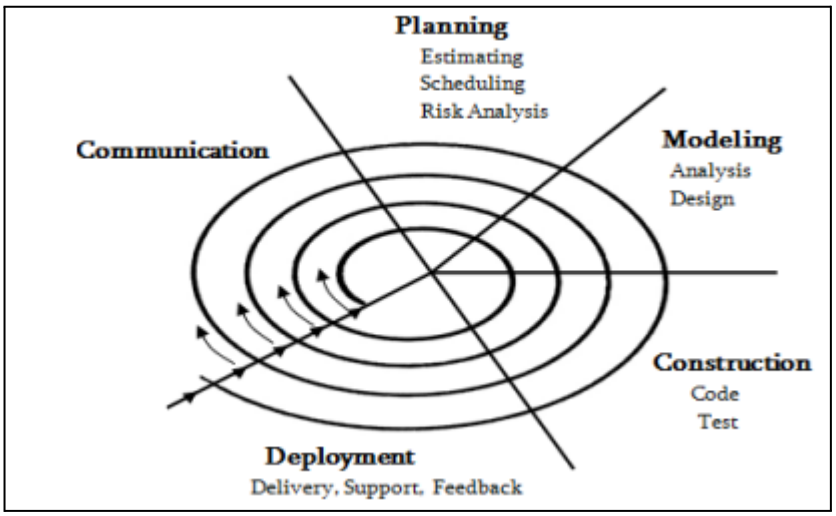


Fig 4.1: Spiral methodology SDLC

4.3 Creation of DeepFake Media

Understanding the creation process of deepfake videos is essential for devising effective detection strategies. The majority of deepfake generation tools, such as Generative Adversarial Networks (GANs) and autoencoders, typically commence with a source image and a target video input. These tools systematically break down the video into individual frames, wherein facial detection algorithms identify and isolate faces within each frame. Subsequently, the identified faces are replaced, frame by frame, with the corresponding target faces, resulting in a seamless integration of the target identity onto the source video footage.

To refine the synthesized frames and enhance the overall quality of the deepfake video, various post-processing techniques are employed. These techniques often involve the utilization of pre-trained models, such as convolutional neural networks (CNNs), to iteratively adjust the synthesized frames and eliminate any residual artifacts or inconsistencies introduced during the face swapping process. By refining the frames in this manner, the deepfake video achieves a higher level of realism, making it increasingly challenging to discern from authentic content.

Our approach to detecting deepfake videos mirrors this creation process by leveraging similar methodologies. We segment the video into individual frames and employ facial detection algorithms to identify and isolate faces within each frame. By comparing facial features and anomalies across frames, we can discern discrepancies indicative of deepfake manipulation. Additionally, we utilize pre-trained models and sophisticated algorithms to analyze and scrutinize the synthesized frames, identifying subtle traces left behind by the deepfake generation process.

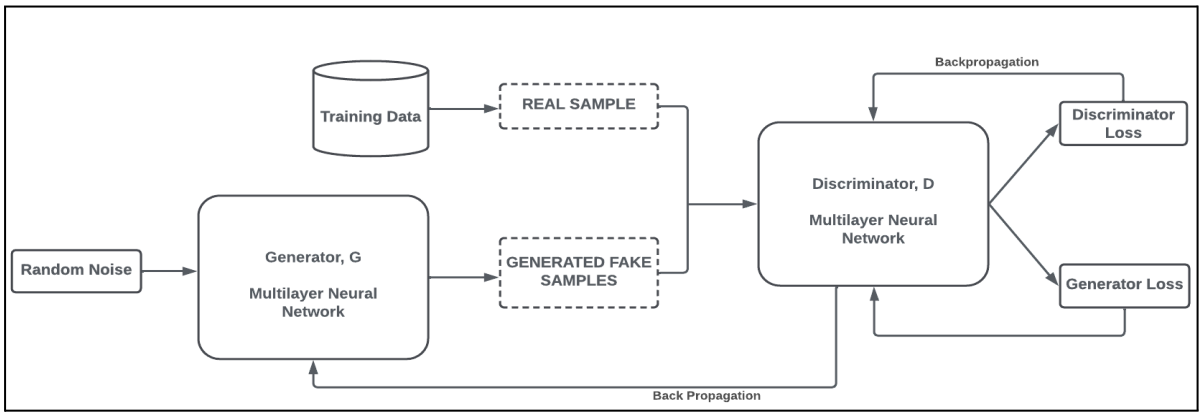


Fig 4.2: GAN architecture

4.4 System Architecture

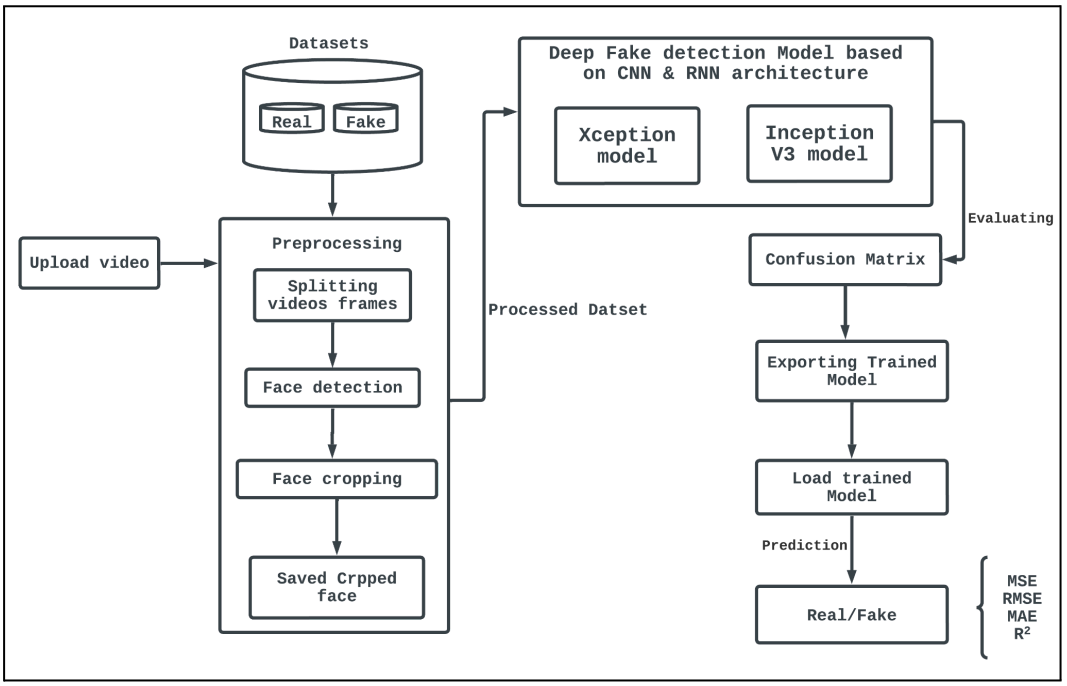


Fig 4.3: System Architecture

In our system, we ensured a balanced training approach for our Tensorflow deepfake detection model by using an equal number of real and fake videos. This strategy was adopted to mitigate any biases within the model. The system architecture diagram illustrates how the model is structured. During the development phase, we began with a dataset, performed preprocessing steps, and generated a new processed dataset. This refined dataset specifically contains face-cropped videos, streamlining the data for further analysis and training.

The deep fake detection system employs a combination of Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) architectures. It encompasses several key functionalities:

- **Data Acquisition:**

The system initiates with the acquisition of two distinct datasets: one comprising authentic videos and the other containing deepfake videos.

- **Preprocessing Stage:**

Upon receiving user-uploaded videos, the system executes a series of preprocessing steps:

- Video Segmentation: Uploaded videos are segmented into individual frames to facilitate analysis.
- Facial Detection: Each frame undergoes facial detection to identify facial regions.
- Facial Cropping: Detected faces within frames are cropped and stored for subsequent processing.

- **Deep Fake Detection Model:**
  - The cropped facial images undergo further preprocessing, including image resizing and pixel value normalization, to prepare them for analysis.
  - Utilizing a pre-trained CNN model such as Xception or Inception V3, the system extracts high-level features from the preprocessed facial images.
  - These extracted features are then fed into an RNN, enabling the system to analyze sequential data. Specifically, the RNN scrutinizes temporal variations in facial features across multiple frames, aiding in the detection of potential deepfake alterations.
- **Evaluation:**

  Finally, the output from the RNN model is assessed to ascertain the authenticity of the video, distinguishing between real and deep fake content.

  Certainly, let's delve into the metrics and evaluation methods used in the deep fake detection system:
  - **Exporting Trained Model:** After training on a sizable dataset of real and deepfake videos, the model is saved and exported for future use. This allows for the preservation and reusability of the trained model without the need to retrain it from scratch.
  - **Load Trained Model:** The exported model can be loaded whenever required to evaluate new videos. This step involves loading the model's architecture, weights, and any other necessary parameters.
  - **Prediction:** Once the trained model is loaded, it can predict whether a new video is real or a deepfake. This prediction is based on the features extracted from the video frames and analyzed by the model.
- **Evaluation Metrics:**
  - Mean Squared Error (MSE): This metric calculates the average squared difference between the actual and predicted values. It provides a measure of the model's accuracy in predicting continuous outcomes.
  - Root Mean Squared Error (RMSE): RMSE is the square root of the MSE. It represents the average magnitude of the errors in predicting continuous variables.
  - Mean Absolute Error (MAE): MAE calculates the average absolute differences between the actual and predicted values. It provides a straightforward measure of the model's prediction accuracy.
  - R-squared (R2): R2 measures the proportion of the variance in the dependent variable (output) that is predictable from the independent variables (input). It ranges from 0 to 1, where a higher value indicates a better fit of the model to the data.

- **Confusion Matrix:**
  - This matrix is a table that visualizes the performance of a classification model. It comprises four metrics:
  - Accuracy: The ratio of correctly predicted observations to the total number of observations.
  - Precision: The proportion of true positive predictions out of all positive predictions made by the model.
  - Recall: The proportion of true positive predictions out of all actual positive instances in the data.
  - F1 Score: The harmonic mean of precision and recall, providing a balance between the two metrics.

In summary, the deep fake detection system employs these metrics and evaluation methods to assess its performance in distinguishing between real and deepfake videos accurately. The combination of CNNs for feature extraction and RNNs for temporal analysis enhances the system's capability to identify inconsistencies indicative of deepfakes.

CHAPTER 5
METHODOLOGY

5.1 Introduction

Our methodology delves into leveraging Deep Neural Networks (DNNs) as a potent tool for detecting DeepFake content. With a focus on countering the challenges posed by AI-generated manipulated media, our approach emphasizes innovative and resilient solutions. We explore the nuanced process of utilizing DNNs to discern these intricate forgeries, offering a glimpse into the strategies, techniques, and data-centric approaches essential for successful DeepFake detection. This methodology is pivotal in preserving digital trust and authenticity amidst the era of advanced AI technologies.

5.2 Proposed Methodology

The research aims to develop a methodology using AI, machine learning, and image processing to detect and counter deep fake images and videos, crucial for combating misinformation, preserving digital content integrity, and building trust in media.

5.2.1 Building a Robust Foundation: The Dataset

We leverage a diverse dataset encompassing an equitable distribution of videos sourced from prominent platforms such as YouTube, FaceForensics++, and the Deep Fake Detection Challenge (DFDC) dataset. Our custom dataset amalgamates 50% original videos with an equal proportion of manipulated deepfake videos. The training set comprises 70% of the data, the validation set contains 10%, and the remaining 20% forms the testing set, ensuring a balanced representation for model training and evaluation. FaceForensics++ (FF++) stands out as a pivotal dataset within the deep fake detection domain, characterized by its inclusion of videos showcasing manipulated facial images. This dataset encompasses a spectrum of manipulation techniques, including face swapping, reenactment, and facial expression alteration. Such diversity makes FF++ a robust benchmark for evaluating the efficacy of deep fake detection algorithms. Researchers widely utilize the FF++ dataset to train and validate deep learning models specifically designed for detecting facial manipulations and discerning counterfeit videos. On the other hand, the Deep Fake Detection Challenge (DFDC) dataset emerges as a pivotal resource developed through a competitive framework aimed at advancing deep fake detection technology. DFDC comprises a substantial collection of videos featuring meticulously crafted deep fake content, generated using cutting-edge AI techniques.

This dataset serves as a valuable asset for assessing the performance of deep fake detection models under realistic conditions, providing insights into their resilience against sophisticated manipulations and real-world scenarios.

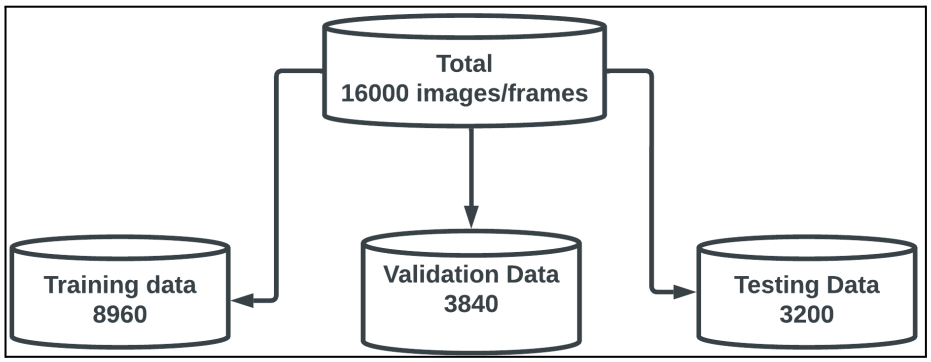


Fig 5.1: Dataset diagram

5.2.2 Extracting the Essence: Features

The deep learning model employed in this study is a custom convolutional neural network (CNN), designed to classify the input as either fake or real. The architecture of the CNN consists of several layers, including convolutional, pooling, flatten, and fully connected layers. The model architecture is summarized as follows:

Input Layer: Accepts input images/frames with dimensions of 224x224x3 (height, width, and color channels).

Convolutional Layers: Utilize 3x3 kernel filters with ReLU activation function and He normal initialization. The initial convolutional layer comprises 64 filters followed by max-pooling, while subsequent layers contain 128 filters each.

Pooling Layers: Max-pooling layers are interspersed between convolutional layers to downsample feature maps and extract essential information.

Flatten Layer: Flattens the output of the convolutional layers into a 1D vector, preparing it for input to the fully connected layers.

Fully Connected Layers: Consist of densely connected neural units with ReLU activation and He normal initialization. Dropout layers with a dropout rate of 0.5 are incorporated to prevent overfitting.

Output Layer: Comprises a single neuron with sigmoid activation, producing a binary classification output indicating the probability of an image being fake or real.

In Conclusion, the custom CNN architecture, tailored for binary classification of real and fake images, utilizes convolutional, pooling, flatten, and fully connected layers with ReLU activation and dropout regularization.

Table 5.1 Layers of model

```
[22]: model.compile(loss="binary_crossentropy", optimizer="nadam",
                    metrics=["accuracy"])
      model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 224, 224, 64)	9,472
max_pooling2d_4 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_7 (Conv2D)	(None, 112, 112, 128)	73,856
conv2d_8 (Conv2D)	(None, 112, 112, 128)	147,584
max_pooling2d_5 (MaxPooling2D)	(None, 56, 56, 128)	0
flatten_2 (Flatten)	(None, 401408)	0
dense_6 (Dense)	(None, 128)	51,380,352
dropout_4 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 64)	8,256
dropout_5 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 1)	65

Total params: 51,619,585 (196.91 MB)
Trainable params: 51,619,585 (196.91 MB)
Non-trainable params: 0 (0.00 B)

+ Code + Markdown

5.2.3 Streamlining the Process: The Preprocessing Pipeline

The preprocessing phase commences with the acquisition of a video clip, containing numerous consecutive frames extracted from a specific source video. Subsequently, we curate an assemblage by gathering all the cropped faces from this original video clip. These faces undergo meticulous preprocessing based on the stringent criteria outlined in the DFDC datasets. This rigorous preprocessing ensures that the resulting preprocessed video clip meets the high standards required for input data in our research study's system. To maintain uniformity and precision in frame count, we employ a sophisticated approach.

We calculate the mean of the entire dataset video, leveraging statistical analysis to derive a representative value. With this mean established, we craft a new processed face-cropped dataset, meticulously aligning the frames to match this calculated mean. Any frames lacking discernible facial features are purposefully omitted during this preprocessing phase, ensuring that only relevant and meaningful data is retained. Considering the computational complexities involved, particularly in processing a 10-second video at a standard rate of 30 frames per second resulting in a total of 300 frames, significant computational resources are required.

Therefore, as a pragmatic approach for experimental purposes, we advocate for the utilization of only the initial 100 frames and in some cases less than 100 frames. This strategic decision optimizes computational efficiency while still allowing for robust model training and experimentation in our research endeavors. In the preprocessing stage of our deepfake detection implementation, It consists of several functions designed to extract features from videos and prepare them for input into a sequence model.

Firstly, the load_video function reads frames from a given video file path, crops them to a square shape, resizes them to a predefined size (IMG_SIZE), and reorders the color channels. This process ensures uniformity in the input frames for subsequent processing. The build_feature_extractor() function initializes an InceptionV3 neural network pre-trained on ImageNet for feature extraction from the preprocessed video frames. It constructs a model that takes preprocessed frames as input and outputs extracted features.

In the prepare_all_videos() function, each video in the dataset is processed to obtain its features and masks. For each video, frames are loaded, features are extracted using the previously defined feature extractor model, and masks are created to denote valid timesteps. These features and masks are then stored in arrays for each video, along with their corresponding labels (0 for real videos and 1 for deepfake videos).

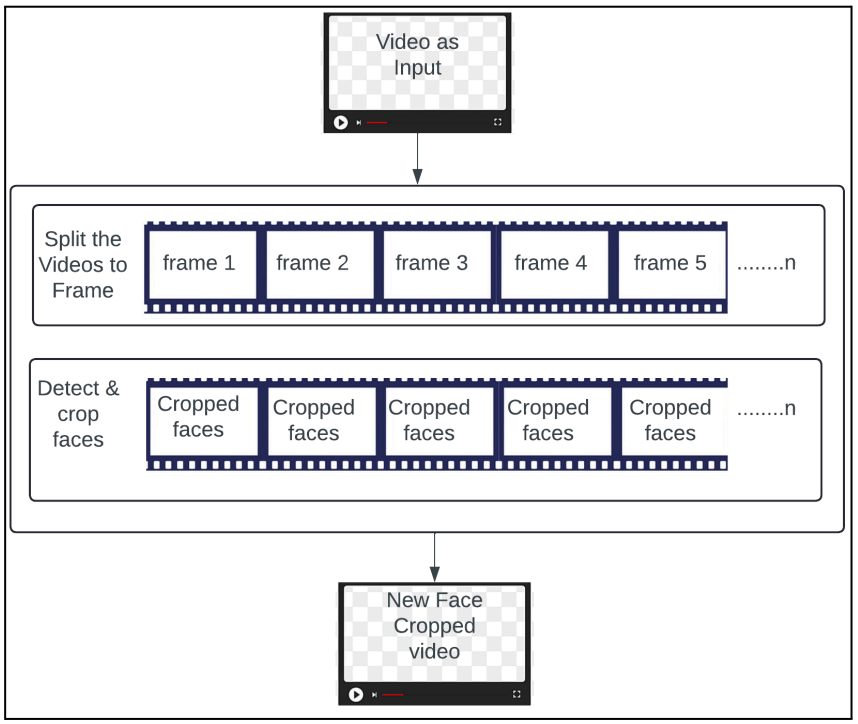


Fig 5.2: Preprocessing steps

5.2.4 A Powerful Architecture: CNN based Xception Model and InceptionV3 Model

In addition to the custom CNN architecture, an alternative approach for deepfake detection was explored by fine-tuning a pretrained convolutional neural network (CNN) model. The Xception model, pretrained on the ImageNet dataset, was selected as the base architecture due to its exceptional performance in image classification tasks.

Dataset Preparation: The dataset was processed and converted into TensorFlow Dataset objects for efficient handling during model training and evaluation. Raw image data along with their corresponding labels were sliced and mapped into TensorFlow datasets. The dataset was divided into training, validation, and testing sets, with appropriate preprocessing steps applied to ensure compatibility with the Xception model.

Model Initialization: The Xception model, excluding its top classification layers, was loaded with pre-trained weights obtained from ImageNet. A global average pooling layer was appended to the output of the base model, followed by a dense layer with a sigmoid activation function, serving as the output layer for binary classification (fake or real). During this initialization phase, the weights of the base model were frozen to prevent them from being updated during the initial training phase.

Initial Training: The model was trained for a few epochs while keeping the base model weights fixed. Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.1 and a momentum of 0.9 was employed for optimization. The model was compiled with binary cross-entropy loss as the optimization objective. Training was performed on the training set, with validation on the validation set to monitor performance and prevent overfitting.

Fine-Tuning: After initializing the model and training the top layers, the top part of the base Xception model was made trainable again. Specifically, layers beyond index 56 were unfrozen to allow them to adapt to the specific task of deepfake detection. Further training was conducted with a lower learning rate of 0.01 to facilitate fine-tuning of both the newly added top layers and the unfrozen layers of the base model. Training was continued for additional epochs to allow the model to learn task-specific features and improve its performance.

5.2.5 Confidence-Driven Detection: The Result

The model outputs whether the video is a deepfake or real, along with its confidence level. This is achieved through preprocessing, frame splitting, and face cropping, directly feeding cropped frames into the model for detection.

The performance of the fine-tuned model was evaluated on the test set to assess its accuracy and generalization capabilities. The model's effectiveness in detecting deepfake videos was measured in terms of binary classification accuracy and loss. Evaluation metrics were computed using the evaluate method provided by TensorFlow, allowing for an objective assessment of the model's performance.

After constructing the model on Kaggle, the subsequent step involved integrating it into a user-friendly interface (UI) using Flask. Flask is a web framework for Python that aids in building web applications. The UI development process primarily focused on providing a seamless user experience, ensuring ease of navigation and interaction.

Within the UI, users are presented with the option to select an image for analysis. This functionality is typically implemented using HTML forms and associated Flask routes. Upon selecting an image, the UI triggers specific routes which in turn handle the image processing logic. These routes facilitate the communication between the frontend and backend components of the application.

To enhance user-friendliness, intuitive routes are established to streamline the user journey. For instance, routes are logically organized to guide users through the image selection process, model processing, and result presentation. This ensures a smooth flow of interaction, minimizing user confusion and frustration.

The model processing phase involves passing the selected image to the pre-trained model for analysis. This step is facilitated by invoking appropriate methods or functions within the Flask routes. The model, previously generated on Kaggle, is loaded within the Flask application environment, allowing seamless integration and utilization.

Upon processing the image, the model generates predictions or analyses based on its architecture and training. These results are then conveyed back to the user through the UI, providing insights or classifications pertaining to the uploaded image. This feedback loop between the user, Flask routes, and the model ensures a dynamic and interactive experience.

Overall, the integration of the model with Flask encompasses various aspects, including route design, user interaction, and backend processing. By leveraging Flask's capabilities, the UI effectively bridges the gap between users and machine learning models, enabling intuitive and efficient utilization for image analysis tasks.

CHAPTER 6
IMPLEMENTATION

6.1 Introduction

Instances abound where deepfake technology has been exploited to mislead people on social media platforms. False deepfake videos of prominent figures like Mark Zuckerberg, Eve from House A.I. Hearing, Donald Trump depicted as James McGill in Breaking Bad, and Barack Obama's fabricated public service announcements have caused widespread panic among the general populace. This situation underscores the critical need for accurate deepfake detection to distinguish these fabricated videos from authentic ones. Recent technological advancements have revolutionized video manipulation. Modern open-source deep learning frameworks such as TensorFlow, Keras, and PyTorch, coupled with affordable access to high computational power, have spurred a paradigm shift.

Conventional autoencoders and pre-trained Generative Adversarial Network (GAN) models have made realistic video and image tampering effortlessly accessible. Mobile and desktop applications like FaceApp and Face Swap further democratize deepfake creation, offering highly realistic transformations of faces in real videos. These apps provide extensive functionalities like altering hairstyles, genders, ages, and other attributes, enabling users to produce high-quality, nearly indistinguishable deepfakes. Although there are some harmful deepfake videos, they represent a minority. The main application of deepfake generation tools is in producing fabricated celebrity pornographic content or revenge porn.

Notable examples include fabricated nude videos of celebrities like Brad Pitt and Angelina Jolie. The lifelike appearance of deepfake videos makes celebrities and public figures vulnerable to pornographic exploitation, fake surveillance footage, misinformation, and malicious hoaxes. Deepfakes are particularly potent in fueling political tensions, highlighting the imperative of detecting and mitigating their spread on social media platforms.

Our implementation journey commenced with the construction of a robust deep learning model on the Kaggle platform, trained to discern authentic videos from deepfake manipulations. Following model development, our focus shifted towards creating an interactive user interface to operationalize the model's capabilities for real-world applications.

The adoption of Flask, a lightweight yet powerful web framework for Python, served as the

cornerstone of our implementation strategy. Flask's flexibility and simplicity enabled us to design and deploy a responsive web application with minimal overhead, facilitating smooth user interaction and efficient backend processing.

Key aspects of our implementation encompass the seamless integration of the model within the Flask environment, the design of user-friendly interfaces for media upload and result visualization, and the optimization of backend processes for enhanced performance. Through meticulous attention to detail and iterative refinement, we strived to deliver a seamless user experience while ensuring robustness and reliability in deepfake detection.

In the subsequent sections, we delve deeper into the technical intricacies of our implementation, elucidating the architecture, functionalities, and performance evaluation of our Flask-based deepfake detection system.

6.2 Model Implementation

In this project, a custom Convolutional Neural Network (CNN) is constructed using TensorFlow's Keras, incorporating convolutional, max-pooling, dropout, and dense layers, trained with binary cross-entropy loss and the Nadam optimizer. Additionally, the pre-trained Xception model is fine-tuned by appending custom classification layers on top, initially freezing base Xception weights, then progressively unfreezing and training with adjusted learning rates, culminating in evaluation on a test set.

Import necessary libraries:

Implementation typically starts with importing essential libraries that provide functionalities for data handling, manipulation, visualization, and machine learning tasks. Below is a detailed elaboration of the libraries mentioned:

- TensorFlow: TensorFlow is a powerful machine learning framework commonly used for deep learning tasks like deepfake detection. It provides tools for building, training, and deploying neural networks, essential for analyzing and identifying manipulated media content.
- OpenCV (cv2): OpenCV is a versatile computer vision library extensively employed in deepfake detection systems. It enables image and video processing tasks crucial for preprocessing, feature extraction, and facial recognition, aiding in the identification of manipulated visual content.
- Pandas and NumPy: Pandas and NumPy are essential libraries for data manipulation and analysis, pivotal in preprocessing datasets for deepfake detection models.

- They facilitate tasks such as data cleaning, transformation, and statistical analysis, contributing to the robustness of the detection algorithm.
- Plotly: Plotly is a Python graphing library utilized for visualizing data insights, which can be beneficial in analyzing model performance metrics or presenting results of deepfake detection. Its interactive features enhance understanding and interpretation of detection outcomes.

Additionally, the tf.test.is_gpu_available() function from TensorFlow is called to check if GPU acceleration is available for faster computation. This is particularly beneficial for deep learning tasks as GPUs can significantly accelerate the training and inference process of deep neural networks by leveraging parallel processing capabilities. Using GPU acceleration can lead to reduced training times and improved overall performance, especially when dealing with large datasets and complex models.

Data Splitting and Visualization:

```
from sklearn.model_selection import train_test_split

Train_set, Test_set = train_test_split(sample_meta,test_size=0.2,random_state=42,stratify=sample_meta['label'])
Train_set, Val_set  = train_test_split(Train_set,test_size=0.3,random_state=42,stratify=Train_set['label'])

y = dict()

y[0] = []
y[1] = []

for set_name in (np.array(Train_set['label']), np.array(Val_set['label']), np.array(Test_set['label'])):
    y[0].append(np.sum(set_name == 'REAL'))
    y[1].append(np.sum(set_name == 'FAKE'))

trace0 = go.Bar( x=['Train Set', 'Validation Set', 'Test Set'],y=y[0],
    name='REAL', marker=dict(color='#33cc33'),opacity=0.7)

trace1 = go.Bar(x=['Train Set', 'Validation Set', 'Test Set'], y=y[1],
    name='FAKE', marker=dict(color='#ff3300'), opacity=0.7)

data = [trace0, trace1]
layout = go.Layout( title='Count of classes in each set',
    xaxis={'title': 'Set'},
    yaxis={'title': 'Count'})

fig = go.Figure(data, layout)
iplot(fig)
```

Fig 6.1: Splitting and Visualize

Data Splitting: The sample_meta DataFrame is split into training (Train_set), validation (Val_set), and test (Test_set) sets with a ratio of 80:10:10 respectively. The splitting is stratified based on the 'label' column to maintain equal proportions of real and fake images in each set.

Data Visualization: The distribution of fake and real images in each dataset (training, validation, and test) is graphically represented using Plotly. Two bar plots are created, one representing the count of real images and the other representing the count of fake images in each set. The purpose of this segment is to prepare the dataset for training the custom Convolutional Neural Network (CNN) model.

It involves splitting the dataset into distinct sets for training, validation, and testing. Additionally, it provides insights into the distribution of real and fake images across these sets, ensuring a balanced representation of classes during model training.

Dataset Retrieval and Preprocessing:

```
def retreive_dataset(set_name):
    images,labels=[],[]
    for (img, imclass) in zip(set_name['videoname'], set_name['label']):
        images.append(cv2.imread('../input/deepfake-faces/faces_224/'+img[:-4]+'.jpg'))
        if(imclass=='FAKE'):
            labels.append(1)
        else:
            labels.append(0)

    return np.array(images),np.array(labels)

X_train,y_train=retreive_dataset(Train_set)
X_val,y_val=retreive_dataset(Val_set)
X_test,y_test=retreive_dataset(Test_set)
```

Fig 6.2: Retrieval and Preprocessing

Dataset Retrieval: The retrieve_dataset function is defined to retrieve images and corresponding labels from a given dataset (set_name). It iterates over the dataset and reads image files using OpenCV (cv2.imread). The path to the images is constructed using the directory path '../input/deepfake-faces/faces_224/' and the filenames from the 'videoname' column of the dataset. The labels are assigned based on the 'label' column, where 'FAKE' images are labeled as 1 and 'REAL' images are labeled as 0.

Implementation of Custom CNN Model Architecture

Dataset Retrieval and Preprocessing:

Dataset Retrieval: The retrieve_dataset function is defined to retrieve images and corresponding labels from a given dataset (set_name). It iterates over the dataset and reads image files using OpenCV (cv2.imread). The path to the images is constructed using the directory path '../input/deepfake-faces/faces_224/' and the filenames from the 'videoname'

column of the dataset. The labels are assigned based on the 'label' column, where 'FAKE' images are labeled as 1 and 'REAL' images are labeled as 0.

Dataset Splitting: The retrieve_dataset function is called to retrieve images and labels for the training (Train_set), validation (Val_set), and test (Test_set) datasets, resulting in three pairs of arrays: X_train, y_train, X_val, y_val, X_test, and y_test.

Model Architecture:

```
from functools import partial
import tensorflow as tf

tf.random.set_seed(42)

DefaultConv2D = partial(tf.keras.layers.Conv2D, kernel_size=3, padding="same",
                        activation="relu", kernel_initializer="he_normal")

model = tf.keras.Sequential([
    tf.keras.Input(shape=[224, 224, 3]),  # Use Input layer instead of passing input_shape
    DefaultConv2D(filters=64, kernel_size=7),
    tf.keras.layers.MaxPool2D(),
    DefaultConv2D(filters=128),
    DefaultConv2D(filters=128),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation="relu",
                          kernel_initializer="he_normal"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=64, activation="relu",
                          kernel_initializer="he_normal"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=1, activation="sigmoid")
])

model.compile(loss="binary_crossentropy", optimizer="nadam",
              metrics=["accuracy"])
model.summary()
```

Fig 6.3: Custom CNN Architecture

Convolutional Neural Network (CNN): Developed using TensorFlow's Keras, the custom CNN model comprises multiple layers:

Input Layer: Configured with an input shape of (224, 224, 3), representing the height, width, and color channels of the input image.

Convolutional Layers: Two convolutional layers with a 3x3 kernel size and ReLU activation function are utilized for feature extraction. 'Same' padding is applied to preserve spatial dimensions.

MaxPooling Layers: Two max-pooling layers are incorporated to downsample the feature maps.

Flatten Layer: This layer transforms the 2D feature maps into a 1D vector for input into the dense layers.

Dense Layers: Two dense layers with ReLU activation and dropout regularization (dropout rate of 0.5) are included for classification.

Output Layer: Consisting of a single neuron with a sigmoid activation function, the output layer produces binary classification output (0 or 1) for distinguishing real and fake images.

Model Compilation: The model is compiled using the binary cross-entropy loss function and the Nadam optimizer, while model performance is evaluated using the accuracy metric.

Model Summary: The model.summary() function is utilized to present a summary of the architecture (Model), detailing layer types, output shapes, and the number of trainable parameters.

Model Training and Evaluation:

```
history = model.fit(X_train, y_train, epochs=5,batch_size=64,
                    validation_data=(X_val, y_val))

score = model.evaluate(X_test, y_test) #accuracy: 0.4890 - loss: 0.6934
```

Fig 6.4: Training and Evaluation (CNN)

Model Training with Train Data: The model.fit() function is called to train the custom Convolutional Neural Network (CNN) model (model). The training process involves iterating over the dataset for 5 epochs and updating the model parameters to minimize the defined loss function (binary cross-entropy). The training is performed in batches of 64 samples per batch to efficiently utilize computational resources.

Model Evaluation Metrics: The evaluation results include the accuracy and loss values obtained by the model on the test set. In this case, the reported accuracy is approximately 48.90%, and the loss is approximately 0.6934.

After assessing the performance of the custom Convolutional Neural Network (CNN) model, which yielded a baseline accuracy of approximately 48.90% on the test set, it is evident that there is room for improvement in detecting deepfake images/frames and videos. To enhance the model's performance, we will proceed with fine-tuning a pre-trained model.

By fine-tuning a pre-trained model, we aim to capitalize on its learned representations,

potentially improving the model's ability to generalize and discriminate between real and fake images. We will implement fine-tuning using the Xception architecture, a state-of-the-art convolutional neural network known for its strong performance on various computer vision tasks.

By leveraging the rich feature representations learned by Xception on ImageNet, we anticipate achieving better performance compared to the custom CNN model. This approach allows us to expedite the training process and potentially achieve higher accuracy in detecting deepfake images.

Pretrained Model Fine-tuning with Xception:

```
train_set_raw=tf.data.Dataset.from_tensor_slices((X_train,y_train))
valid_set_raw=tf.data.Dataset.from_tensor_slices((X_val,y_val))
test_set_raw=tf.data.Dataset.from_tensor_slices((X_test,y_test))

tf.keras.backend.clear_session()  # extra code - resets layer name counter

batch_size = 32
preprocess = tf.keras.applications.xception.preprocess_input
train_set = train_set_raw.map(lambda X, y: (preprocess(tf.cast(X, tf.float32)), y))
train_set = train_set.shuffle(1000, seed=42).batch(batch_size).prefetch(1)
valid_set = valid_set_raw.map(lambda X, y: (preprocess(tf.cast(X, tf.float32)), y)).batch(batch_size)
test_set = test_set_raw.map(lambda X, y: (preprocess(tf.cast(X, tf.float32)), y)).batch(batch_size)
```

Fig 6.5: Data pipelining and pre processing

Data Pipelines: TensorFlow's tf.data.Dataset.from_tensor_slices() function is used to create TensorFlow Dataset objects from the training, validation, and test sets (train_set_raw, valid_set_raw, test_set_raw). This allows for efficient data loading and processing during model training and evaluation.

By leveraging this function, you can effortlessly convert your training, validation, and test sets (train_set_raw, valid_set_raw, test_set_raw) into TensorFlow Dataset objects. This conversion optimizes data loading and processing, resulting in a more efficient machine learning pipeline overall.

The benefits extend beyond mere conversion. tf.data.Dataset.from_tensor_slices() enables efficient handling of large datasets, facilitates batching, and supports transformations such as shuffling and prefetching. These capabilities are vital for effectively training deep learning models.

```
def random_flip(image, seed=None):
    if seed is not None:
        tf.random.set_seed(seed)
    return tf.image.random_flip_left_right(image)

def random_rotation(image, factor=0.05, seed=None):
    if seed is not None:
        tf.random.set_seed(seed)
    return tf.keras.layers.experimental.preprocessing.RandomRotation(factor=factor)(image)

def random_contrast(image, factor=0.2, seed=None):
    if seed is not None:
        tf.random.set_seed(seed)
    return tf.keras.layers.experimental.preprocessing.RandomContrast(factor=factor)(image)

data_augmentation = tf.keras.Sequential([
    tf.keras.layers.Lambda(lambda x: random_flip(x, seed=42)),
    tf.keras.layers.Lambda(lambda x: random_rotation(x, factor=0.05, seed=42)),
    tf.keras.layers.Lambda(lambda x: random_contrast(x, factor=0.2, seed=42))
])

tf.random.set_seed(42)  # extra code - ensures reproducibility
base_model = tf.keras.applications.xception.Xception(weights="imagenet",
                                                     include_top=False)
avg = tf.keras.layers.GlobalAveragePooling2D()(base_model.output)
output = tf.keras.layers.Dense(1, activation="sigmoid")(avg)
model = tf.keras.Model(inputs=base_model.input, outputs=output)
```

Fig 6.6: Augmentation Pipeline and initialization

Data Preprocessing: The images/frames in the datasets are preprocessed using the tf.keras.applications.xception.preprocess_input function, which applies the necessary preprocessing steps required for the Xception model. This includes scaling pixel values to the range expected by the Xception model.

Data Augmentation Pipeline: A data augmentation pipeline is constructed using TensorFlow's Sequential API. This pipeline applies random transformations to the input images to increase the diversity of the training data and improve the model's generalization ability. Random Flip introduces horizontal flips to images for facial orientation variation, while Random Rotation adds random rotations to enhance robustness against head pose variations. Additionally, Random Contrast adjusts image contrast randomly to improve feature detection under varying lighting conditions.

Pretrained Model Initialization:

The Xception model, pre-trained on the ImageNet dataset, is loaded using tf.keras.applications.xception.Xception(weights="imagenet", include_top=False). The include_top=False argument excludes the top classification layers of the Xception model,

which will be replaced with custom layers for our specific task.

Global Average Pooling Layer: A global average pooling layer is added to reduce the spatial dimensions of the feature maps produced by the Xception base model.

Output Layer: A dense layer with a sigmoid activation function is added to produce binary classification outputs (real or fake) based on the extracted features.

Initial Fine-tuning Pretrained Xception Model:

```
for layer in base_model.layers:
    layer.trainable = False

#train the model for a few epochs, while keeping the base model weights fixed
optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)
model.compile(loss="binary_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set, validation_data=valid_set, epochs=3)

for indices in zip(range(33), range(33, 66), range(66, 99), range(99, 132)):
    for idx in indices:
        print(f"{idx:3}: {base_model.layers[idx].name:22}", end="")
    print()

model.evaluate(test_set) #accuracy: 0.6290 - loss: 0.9084
```

Fig 6.7: Freeze base layers and train

Freezing Layers: The code iterates over each layer in the base Xception model (base_model) and sets the trainable attribute to False. This effectively freezes the weights of all layers in the base model, preventing them from being updated during subsequent training.

Model Training: The model is compiled using SGD optimizer with a learning rate of 0.1 and momentum of 0.9, utilizing binary cross-entropy loss and accuracy metric. Training is executed using model.fit() on train_set, with validation on valid_set for 3 epochs. The frozen base model layers ensure only the newly added classification layers are trained. Layer information is displayed post-training, offering insight into the Xception model architecture and the fixed layers during fine-tuning.

Final Fine-tuning Pretrained Xception Model:

The model is initially evaluated on the test set, yielding an accuracy of 62.90% and a loss of 0.9084. Additional layers of the Xception base model are unfrozen, starting from index 56, and fine-tuned to learn task-specific features. With a lower learning rate of 0.01, the model is recompiled and trained for 10 epochs on the training set, resulting in an accuracy improvement to 80.88% and a loss reduction to 0.7977 on the test set.

```
model.evaluate(test_set)  #accuracy: 0.6290 - loss: 0.9084

for layer in base_model.layers[56:]:
    layer.trainable = True

optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
model.compile(loss="binary_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set, validation_data=valid_set, epochs=10)

model.evaluate(test_set)    #accuracy: 0.8088 - loss: 0.7977
```

Fig 6.8: Final Fine-tuning

Deepfake Video Detection Model

These utility functions are crucial for preparing video data for input into the video detection model. The crop_center_square function ensures uniformity in frame dimensions, while the load_video function handles the extraction, cropping, resizing, and formatting of video frames. These preprocessing steps are essential for standardizing the input data and enabling the model to effectively learn patterns and features from the videos.

```
IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 10

MAX_SEQ_LENGTH = 20
NUM_FEATURES = 2048

def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y : start_y + min_dim, start_x : start_x + min_dim]


def load_video(path, max_frames=0, resize=(IMG_SIZE, IMG_SIZE)):
    cap = cv2.VideoCapture(path)
    frames = []
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            frame = crop_center_square(frame)
            frame = cv2.resize(frame, resize)
            frame = frame[:, :, [2, 1, 0]]
            frames.append(frame)

            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    return np.array(frames)
```

Fig 6.9: Crop and load_video()

Video Feature Extraction and Preparation

InceptionV3 Feature Extractor: The build_feature_extractor() function initializes an InceptionV3 model pre-trained on ImageNet for feature extraction. The model is configured to exclude the top classification layers (include_top=False) and use global average pooling for feature aggregation (pooling="avg"). The input shape of the model is set to (IMG_SIZE, IMG_SIZE, 3) to match the dimensions of the input video frames.

This code segment outlines the process of preparing video data for sequence modeling. The InceptionV3 model serves as a feature extractor, transforming video frames into meaningful feature representations. These features, along with corresponding masks, are organized into sequences suitable for input into sequence models. By leveraging pre-trained models and efficient preprocessing techniques, the code streamlines the preparation of video data for subsequent model training and evaluation.

```
def build_feature_extractor():
    feature_extractor = keras.applications.InceptionV3(
        weights="imagenet",
        include_top=False,
        pooling="avg",
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.inception_v3.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)

    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")


feature_extractor = build_feature_extractor()
```

Fig 6.10: InceptionV3 Feature Extractor

Sequence Model Construction and Training:

```
frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))
mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")

x = keras.layers.GRU(16, return_sequences=True)(
    frame_features_input, mask=mask_input
)
x = keras.layers.GRU(8)(x)
x = keras.layers.Dropout(0.4)(x)
x = keras.layers.Dense(8, activation="relu")(x)
output = keras.layers.Dense(1, activation="sigmoid")(x)

model = keras.Model([frame_features_input, mask_input], output)

model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
model.summary()

checkpoint = keras.callbacks.ModelCheckpoint('./model_weights.weights.h5', save_weights_only=True, save_best_only=True)
history = model.fit(
        [train_data[0], train_data[1]],
        train_labels,
        validation_data=([test_data[0], test_data[1]],test_labels),
        callbacks=[checkpoint],
        epochs=EPOCHS,
        batch_size=8
    )
```

Fig 6.11: Sequence model training

The model architecture comprises two input layers: one for frame features and another for masks, with shapes (MAX_SEQ_LENGTH, NUM_FEATURES) and (MAX_SEQ_LENGTH) respectively. Two GRU layers are employed: the first with 16 units returning sequences and the second with 8 units aggregating sequence information. A dropout layer is applied with a dropout rate of 40% to counteract overfitting, succeeded by a dense layer featuring 8 units and ReLU activation.

For binary classification, the output layer is a dense layer with sigmoid activation. Model compilation uses the Adam optimizer and binary cross-entropy loss, with accuracy serving as the evaluation metric. Training utilizes the fit() function with frame features, masks, and labels, and includes validation data. Additionally, ModelCheckpoint callback saves the best model weights based on validation accuracy.

Reaching its 10th epoch, the model achieved 81.98% accuracy and a loss of 0.5968 on training data, while obtaining 80.00% accuracy and a loss of 0.5982 on the validation set, demonstrating consistent performance and good generalization.

Video - Frame Analysis and Thresholding:

```
# Open video file
video_path = '/content/R_fake_video.mp4'
cap = cv2.VideoCapture(video_path)

# Check if video file opened successfully
if not cap.isOpened():
    print(f"Error: Unable to open video file '{video_path}'")
    exit()

# Initialize list to store prediction results
results = []

# Initialize list to store frames passed to the model
frames_passed = []

# Capture 5 random frames from the video
frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
frames_indices = np.random.randint(0, frame_count, size=5)

for i, idx in enumerate(frames_indices):
    # Set the frame position
    cap.set(cv2.CAP_PROP_POS_FRAMES, idx)

    # Read the frame
    ret, frame = cap.read()

    # Check if frame reading was successful
    if not ret:
        print(f"Error: Unable to read frame {idx} from video")
        continue

    # Predict image
    prediction = predict_image(frame)

    # Store the prediction result
    results.append(prediction)

    # If prediction passes a threshold, store the frame
    threshold = 0.5  # Threshold for binary classification (adjust as needed)
    if prediction >= threshold:
        frames_passed.append((frame, idx))

# Close video capture
cap.release()
```

Fig 6.12: Video - Frame Analysis and Thresholding

A function named predict_image() is defined to predict whether an input image contains a deepfake. Within this function, the input image is resized to match the model's input size, normalized to values between 0 and 1, and expanded to include a batch dimension.

The model then predicts the probability of the image being a deepfake, and this probability is returned. Next, a video file is opened using OpenCV's VideoCapture() function. If the video file fails to open, an error message is printed, and the program exits. To assess the presence of deepfakes in the video, five random frames are selected for analysis. These frames are chosen using NumPy's randint() function based on the total number of frames in the video.

For each selected frame, the predict_image() function is called to predict whether the frame contains a deepfake. If the prediction surpasses a predefined threshold (set at 0.5 in this case), indicating a high likelihood of being a deepfake, the frame is stored along with its index.

After processing all selected frames, the video capture object is released. \The overall classification result is determined by calculating the average prediction value across all analyzed frames. If the average prediction exceeds the threshold, the video is labeled as a deepfake; otherwise, it is deemed real.

Finally, Matplotlib is utilized to display the frames that passed the deepfake detection model along with their corresponding indices. The overall prediction result, along with its confidence score, is presented as the title of the visualization.

CHAPTER 7
RESULT ANALYSIS

7.1 Introduction

In this section, we delve into the outcomes of our deepfake detection models: a Custom Convolutional Neural Network (CNN), a Pre-Trained Xception model fine-tuned for the task, and an InceptionV3-based model specialized for video analysis. We meticulously assess each model's efficacy through metrics like accuracy, loss, and other pertinent indicators, gauging their proficiency in discerning between genuine and manipulated visual content. Furthermore, we present noteworthy insights gleaned from the training and testing phases, shedding light on their practical utility and implications.

A pre-trained Xception model, known for its efficiency and performance in image classification tasks. This model served as a feature extractor, capturing meaningful representations from input images. By leveraging the pretrained Xception model, we expedited the development process and potentially improved the performance of our deepfake detection system, particularly when working with limited data.

To facilitate the training process and capitalize on high GPU processing power, we employed the Kaggle platform. Kaggle provided access to computational resources necessary for training the deep learning models efficiently. Subsequently, the trained models were deployed on Google Colab for prediction purposes. Google Colab's cloud-based infrastructure, offering free access to GPUs, enabled seamless model deployment and prediction.

Moreover, we developed a user-friendly interface using Python Flask, a lightweight web framework. This interface allowed users to interact with the deepfake detection system effortlessly. Users could upload images or videos for analysis, and the system would process them through the trained models to determine the presence of deepfakes. The results were then presented to users via the Flask interface, enhancing accessibility for individuals without technical expertise in deep learning.

This integrated approach combining custom CNN architecture, pretrained Xception model, Kaggle for training, Google Colab for deployment, and Python Flask for user interface development yielded a robust deepfake detection system.

By leveraging state-of-the-art deep learning techniques and user-friendly design principles, our system contributes to mitigating the adverse effects of deepfake technology and promoting the authentication of digital media.

7.2 Performance Measures

Input Dataset for Custom CNN Model:

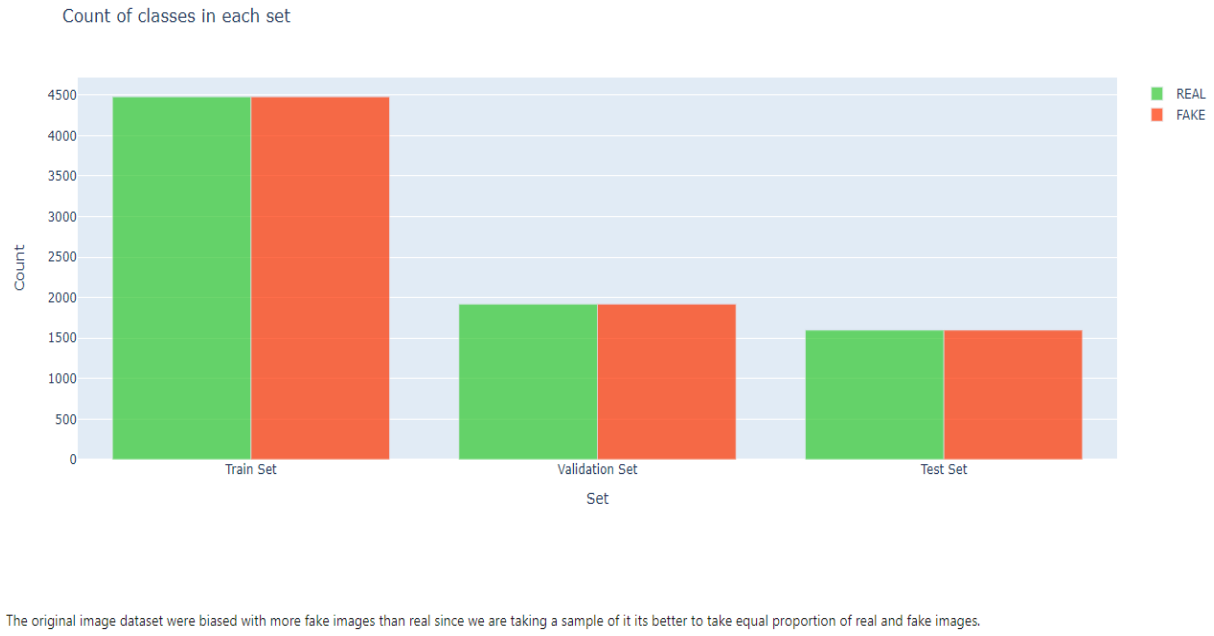


Fig 7.1: Dataset Distribution

Epochs for Training:

```
history = model.fit(X_train, y_train, epochs=5,batch_size=64,
                    validation_data=(X_val, y_val))
```

```
Train on 8960 samples, validate on 3840 samples
Epoch 1/5
8960/8960 [==============================] - 31s 3ms/sample - loss: 90.6586 - accuracy: 0.4954 - val_loss: 0.6927 - val_accura
cy: 0.5208
Epoch 2/5
8960/8960 [==============================] - 26s 3ms/sample - loss: 0.7005 - accuracy: 0.5065 - val_loss: 0.6967 - val_accurac
y: 0.4917
Epoch 3/5
8960/8960 [==============================] - 26s 3ms/sample - loss: 0.6939 - accuracy: 0.5052 - val_loss: 0.6937 - val_accurac
y: 0.5096
Epoch 4/5
8960/8960 [==============================] - 26s 3ms/sample - loss: 0.6930 - accuracy: 0.5113 - val_loss: 0.6938 - val_accurac
y: 0.5185
Epoch 5/5
8960/8960 [==============================] - 26s 3ms/sample - loss: 0.6952 - accuracy: 0.5218 - val_loss: 0.6927 - val_accurac
y: 0.5201
```

+ Code + Markdown

Fig 7.2: Epochs for CNN model

Performance Measures:

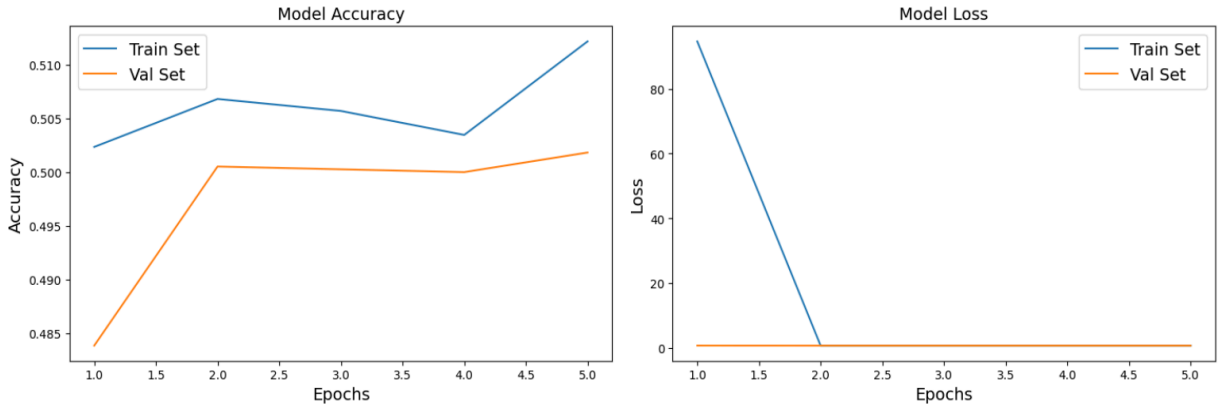


Fig 7.3: Performance Visualization

Pre-trained Xception Model

Initial 3 epochs:

```
#train the model for a few epochs, while keeping the base model weights fixed
optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)
model.compile(loss="binary_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set, validation_data=valid_set, epochs=3)
```

```
Epoch 1/3
280/280 ━━━━━━━━━━━━━━━━━━━━ 67s 185ms/step - accuracy: 0.5701 - loss: 0.9087 - val_accuracy: 0.5659 - val_loss: 1.4576
Epoch 2/3
280/280 ━━━━━━━━━━━━━━━━━━━━ 49s 173ms/step - accuracy: 0.5962 - loss: 1.0578 - val_accuracy: 0.6409 - val_loss: 0.7502
Epoch 3/3
280/280 ━━━━━━━━━━━━━━━━━━━━ 48s 173ms/step - accuracy: 0.6236 - loss: 0.9111 - val_accuracy: 0.6120 - val_loss: 0.9380
```

+ Code + Markdown

Fig 7.4: Initial 3 epochs

Layers of Xception model:

The Xception model is a deep learning architecture that stands out for its depth and efficiency in image classification tasks. Unlike traditional convolutional neural networks (CNNs), which stack layers sequentially, Xception employs an "extreme inception" approach by using depthwise separable convolutions. These convolutions separate the spatial filtering and the channel-wise filtering, resulting in a more efficient and effective model.

The Xception model consists of multiple layers, each serving a specific purpose in feature extraction and classification. Here's an overview of the key layers in the Xception model:

- The Xception model begins with an input layer, receiving image data as tensors with height, width, and channels. The subsequent entry flow employs convolutional and pooling layers to extract low-level features, gradually reducing spatial dimensions

while increasing channels.

- At the model's core, the middle flow consists of residual modules with depth-wise separable convolutions, separating spatial and channel-wise filtering to capture complex patterns efficiently.
- Following feature extraction, the exit flow further processes features using convolutional layers and global average pooling, aggregating spatial information. It concludes with fully connected layers and a softmax activation function for classification.
- Finally, the output layer synthesizes extracted features, commonly utilizing a softmax function to produce class probabilities. This layered approach enables effective feature extraction and classification in image tasks.

```
  0: input_layer_2          33: block4_pool           66: block8_sepconv1_act    99: block11_sepconv2_act
  1: block1_conv1           34: batch_normalization_2 67: block8_sepconv1       100: block11_sepconv2
  2: block1_conv1_bn        35: add_2                 68: block8_sepconv1_bn    101: block11_sepconv2_bn
  3: block1_conv1_act       36: block5_sepconv1_act   69: block8_sepconv2_act   102: block11_sepconv3_act
  4: block1_conv2           37: block5_sepconv1       70: block8_sepconv2       103: block11_sepconv3
  5: block1_conv2_bn        38: block5_sepconv1_bn    71: block8_sepconv2_bn    104: block11_sepconv3_bn
  6: block1_conv2_act       39: block5_sepconv2_act   72: block8_sepconv3_act   105: add_9
  7: block2_sepconv1        40: block5_sepconv2       73: block8_sepconv3       106: block12_sepconv1_act
  8: block2_sepconv1_bn     41: block5_sepconv2_bn    74: block8_sepconv3_bn    107: block12_sepconv1
  9: block2_sepconv2_act    42: block5_sepconv3_act   75: add_6                 108: block12_sepconv1_bn
 10: block2_sepconv2        43: block5_sepconv3       76: block9_sepconv1_act   109: block12_sepconv2_act
 11: block2_sepconv2_bn     44: block5_sepconv3_bn    77: block9_sepconv1       110: block12_sepconv2
 12: conv2d                 45: add_3                 78: block9_sepconv1_bn    111: block12_sepconv2_bn
 13: block2_pool            46: block6_sepconv1_act   79: block9_sepconv2_act   112: block12_sepconv3_act
 14: batch_normalization    47: block6_sepconv1       80: block9_sepconv2       113: block12_sepconv3
 15: add                    48: block6_sepconv1_bn    81: block9_sepconv2_bn    114: block12_sepconv3_bn
 16: block3_sepconv1_act    49: block6_sepconv2_act   82: block9_sepconv3_act   115: add_10
 17: block3_sepconv1        50: block6_sepconv2       83: block9_sepconv3       116: block13_sepconv1_act
 18: block3_sepconv1_bn     51: block6_sepconv2_bn    84: block9_sepconv3_bn    117: block13_sepconv1
 19: block3_sepconv2_act    52: block6_sepconv3_act   85: add_7                 118: block13_sepconv1_bn
 20: block3_sepconv2        53: block6_sepconv3       86: block10_sepconv1_act  119: block13_sepconv2_act
 21: block3_sepconv2_bn     54: block6_sepconv3_bn    87: block10_sepconv1      120: block13_sepconv2
 22: conv2d_1               55: add_4                 88: block10_sepconv1_bn   121: block13_sepconv2_bn
 23: block3_pool            56: block7_sepconv1_act   89: block10_sepconv2_act  122: conv2d_3
 24: batch_normalization_1  57: block7_sepconv1       90: block10_sepconv2      123: block13_pool
 25: add_1                  58: block7_sepconv1_bn    91: block10_sepconv2_bn   124: batch_normalization_3
 26: block4_sepconv1_act    59: block7_sepconv2_act   92: block10_sepconv3_act  125: add_11
 27: block4_sepconv1        60: block7_sepconv2       93: block10_sepconv3      126: block14_sepconv1
 28: block4_sepconv1_bn     61: block7_sepconv2_bn    94: block10_sepconv3_bn   127: block14_sepconv1_bn
 29: block4_sepconv2_act    62: block7_sepconv3_act   95: add_8                 128: block14_sepconv1_act
 30: block4_sepconv2        63: block7_sepconv3       96: block11_sepconv1_act  129: block14_sepconv2
 31: block4_sepconv2_bn     64: block7_sepconv3_bn    97: block11_sepconv1      130: block14_sepconv2_bn
 32: conv2d_2               65: add_5                 98: block11_sepconv1_bn   131: block14_sepconv2_act
```

Fig 7.5: Xception Model Layers

10 epochs with lower learning rate:

```
[38]: for layer in base_model.layers[56:]:
          layer.trainable = True

      optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
      model.compile(loss="binary_crossentropy", optimizer=optimizer,
                    metrics=["accuracy"])
      history = model.fit(train_set, validation_data=valid_set, epochs=10)
```

```
Epoch 1/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 110s 328ms/step - accuracy: 0.6250 - loss: 0.6735 - val_accuracy: 0.7453 - val_loss: 0.5076
Epoch 2/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 318ms/step - accuracy: 0.8433 - loss: 0.3485 - val_accuracy: 0.7648 - val_loss: 0.5609
Epoch 3/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 319ms/step - accuracy: 0.9285 - loss: 0.1800 - val_accuracy: 0.7688 - val_loss: 0.7605
Epoch 4/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 317ms/step - accuracy: 0.9577 - loss: 0.1167 - val_accuracy: 0.7870 - val_loss: 0.6945
Epoch 5/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 317ms/step - accuracy: 0.9703 - loss: 0.0738 - val_accuracy: 0.7937 - val_loss: 0.7716
Epoch 6/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 317ms/step - accuracy: 0.9813 - loss: 0.0548 - val_accuracy: 0.7326 - val_loss: 1.1234
Epoch 7/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 318ms/step - accuracy: 0.9832 - loss: 0.0459 - val_accuracy: 0.7857 - val_loss: 0.7711
Epoch 8/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 318ms/step - accuracy: 0.9859 - loss: 0.0423 - val_accuracy: 0.8073 - val_loss: 0.7252
Epoch 9/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 317ms/step - accuracy: 0.9906 - loss: 0.0300 - val_accuracy: 0.8094 - val_loss: 0.8315
Epoch 10/10
280/280 ━━━━━━━━━━━━━━━━━━━━ 89s 318ms/step - accuracy: 0.9927 - loss: 0.0250 - val_accuracy: 0.8086 - val_loss: 0.8584
```

Fig 7.6: Unfreeze and train all layers

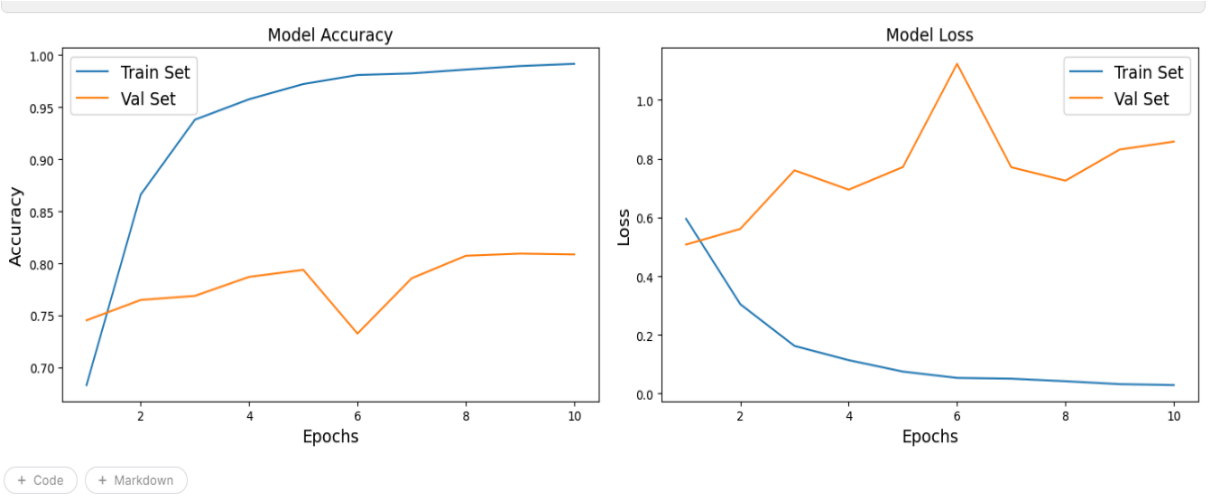


Fig 7.7: Performance after Training

```
model.evaluate(test_set)
```

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 119ms/step - accuracy: 0.8088 - loss: 0.7977
[40]: [0.7824663519859314, 0.815625011920929]
```

+ Code + Markdown

The model accuracy finally reaches to 80.8%

Fig 7.8: model.evaluate() function

This code segment is utilized for visualizing the performance metrics, namely accuracy and loss, of a trained neural network model. It begins by extracting the accuracy and loss values from the history object obtained during the training process, both for the training (acc, loss) and validation (val_acc, val_loss) sets. The epochs_range variable is defined to represent the range of epochs over which the model was trained.

The visualization is achieved using Matplotlib, a popular plotting library in Python. The plt.figure() function initiates the creation of a figure with a specified size. Subsequently, plt.subplot() is employed to create a subplot grid where the accuracy and loss plots will be displayed side by side.

In the first subplot (plt.subplot(1, 2, 1)), the accuracy curves for both the training and validation sets are plotted against the epochs using plt.plot(). The epochs_range is utilized for the x-axis, while acc and val_acc represent the accuracy values for the training and validation sets, respectively. Similarly, in the second subplot (plt.subplot(1, 2, 2)), the loss curves for the training and validation sets are plotted.

Additional customization is performed using functions like plt.xlabel(), plt.ylabel(), and plt.title() to label the axes and provide titles for the subplots. The plt.legend() function adds a legend to differentiate between the training and validation curves. Finally, plt.tight_layout() ensures that subplots do not overlap, and plt.show() displays the plots.

The output displayed below the code snippet provides the final accuracy and loss values achieved by the model on the validation set after training. This information is crucial for assessing the model's performance and determining its effectiveness in the given task.

7.3 User Interface and Detection

Homepage:

Our Flask UI offers a user-friendly interface designed specifically for detecting deepfake content. With its intuitive design and seamless navigation, users can effortlessly explore the platform's powerful detection features. The inclusion of insightful visuals enhances the user experience, providing clarity and ease of understanding. Our home page equips users with the tools needed to effectively combat the threats posed by deepfake media. Its streamlined layout ensures efficient interaction, allowing users to focus on leveraging the platform's robust capabilities. Ultimately, our Flask UI empowers users to stay ahead in identifying and addressing the challenges presented by the proliferation of deepfake content.

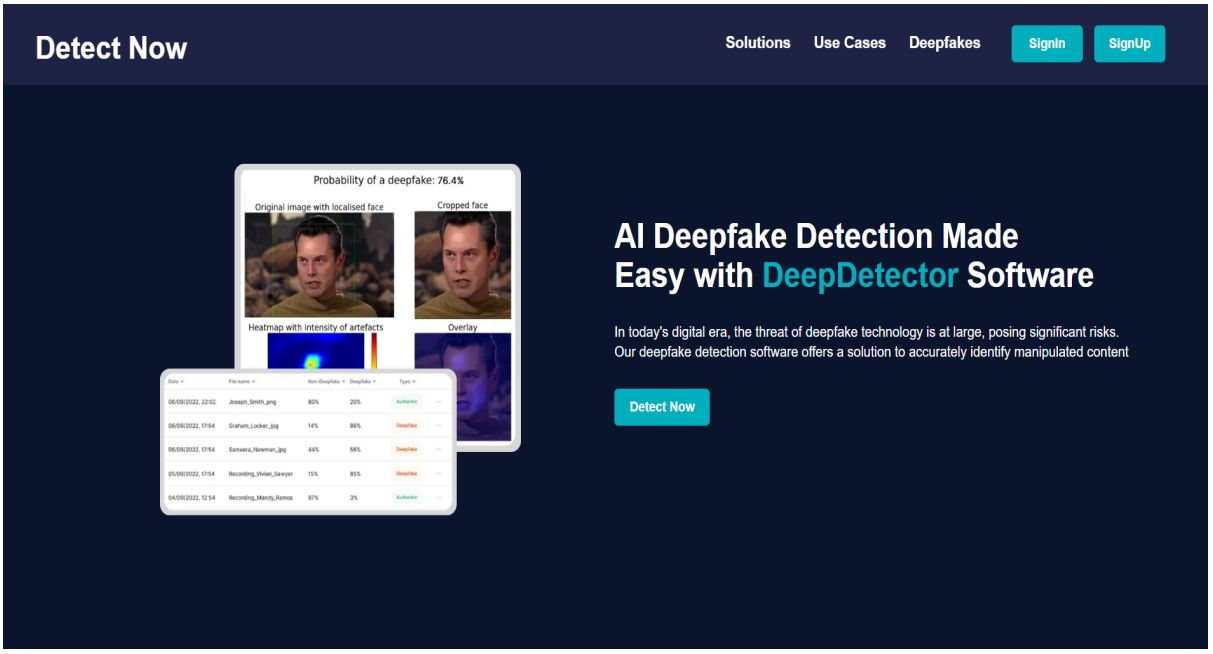


Fig 7.9: Homepage UI in Flask

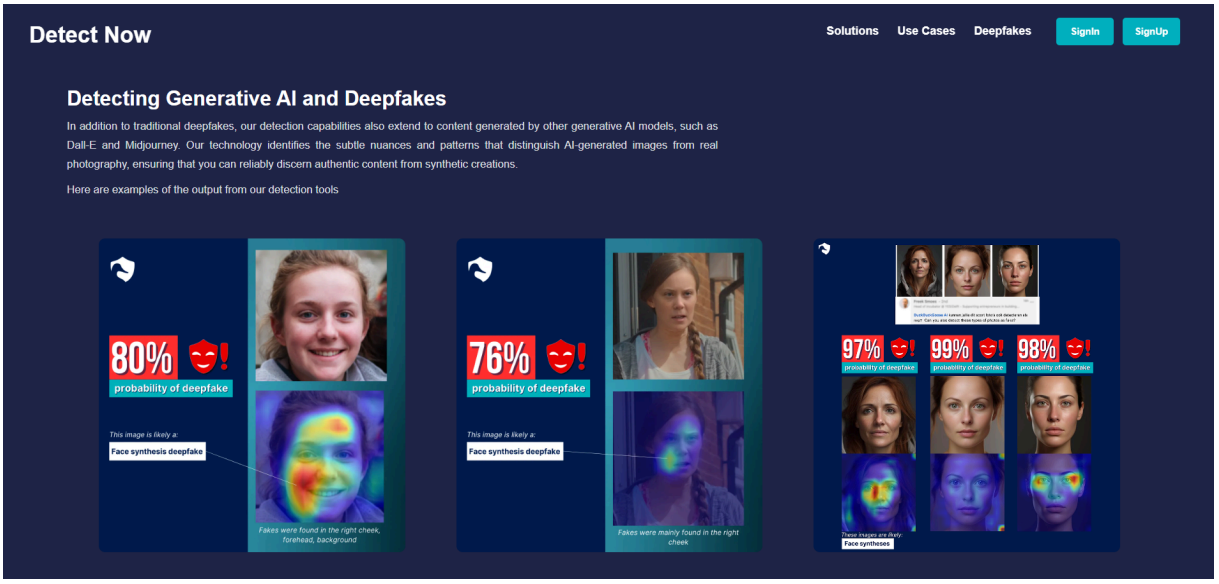


Fig 7.10: User Interface

Detection of Real Images:

Incorporating a smooth image upload functionality, our Flask UI simplifies the process of inputting data for backend analysis. Through seamless integration with Python, our backend seamlessly utilizes the deployed model to scrutinize uploaded images for any signs of deepfake manipulation. This synergy between the frontend and backend optimizes the detection workflow, bolstering the platform's proficiency in identifying deceptive content within images and videos. By enabling efficient data processing and analysis, our Flask UI ensures swift and accurate identification of potential deepfake media.

This cohesive integration enhances user experience and strengthens the platform's overall effectiveness in combating the proliferation of deceptive content online.

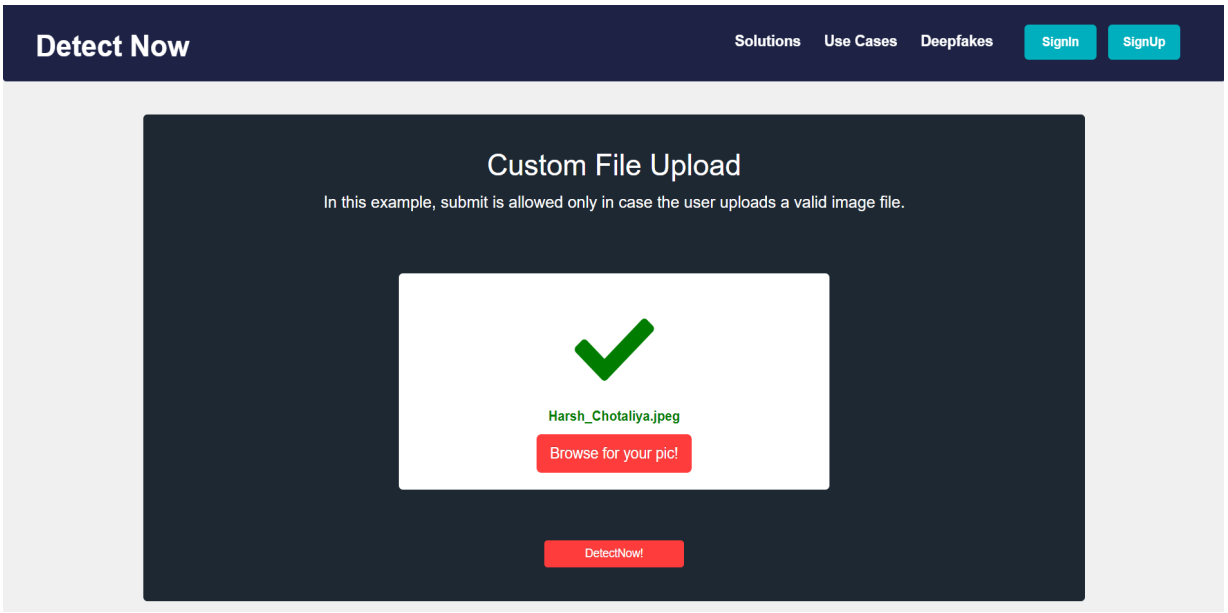


Fig 7.11: Upload Real image for prediction

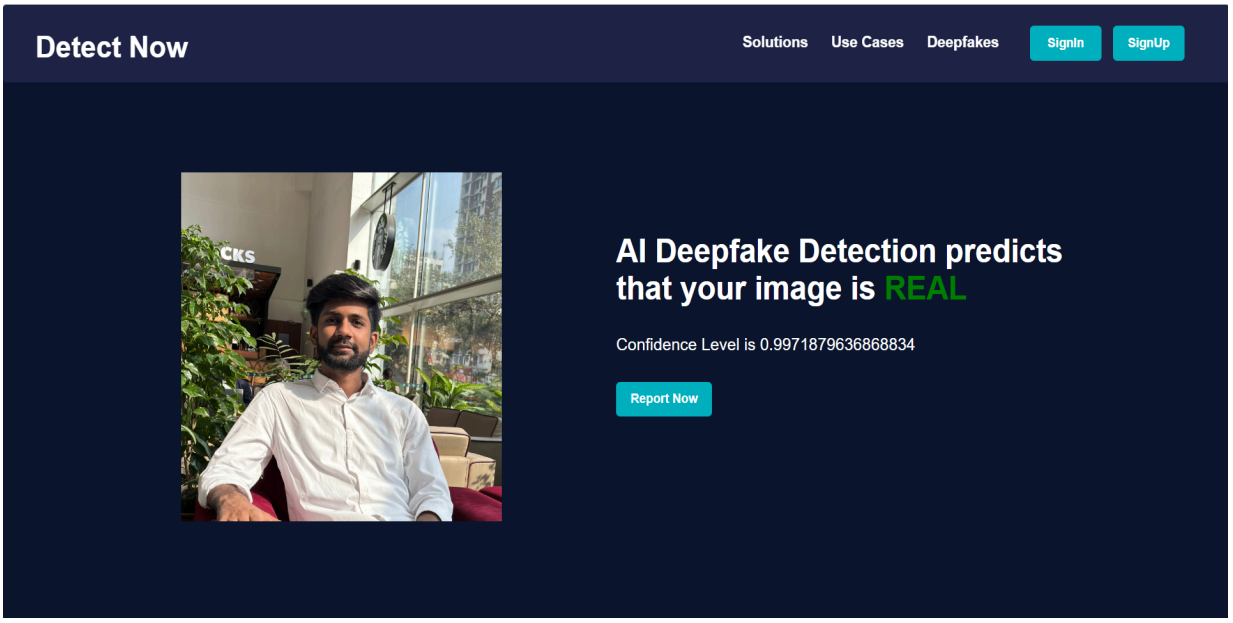


Fig 7.12: Model prediction for Adil's Real Image

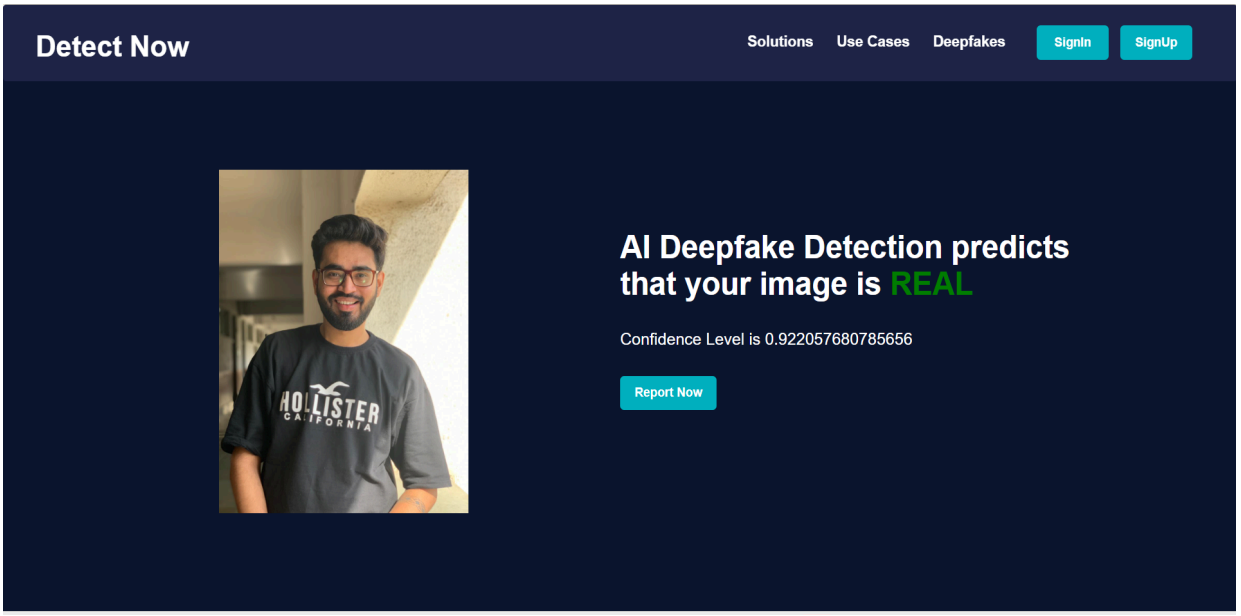


Fig 7.13: Model prediction for Harsh's Real Image

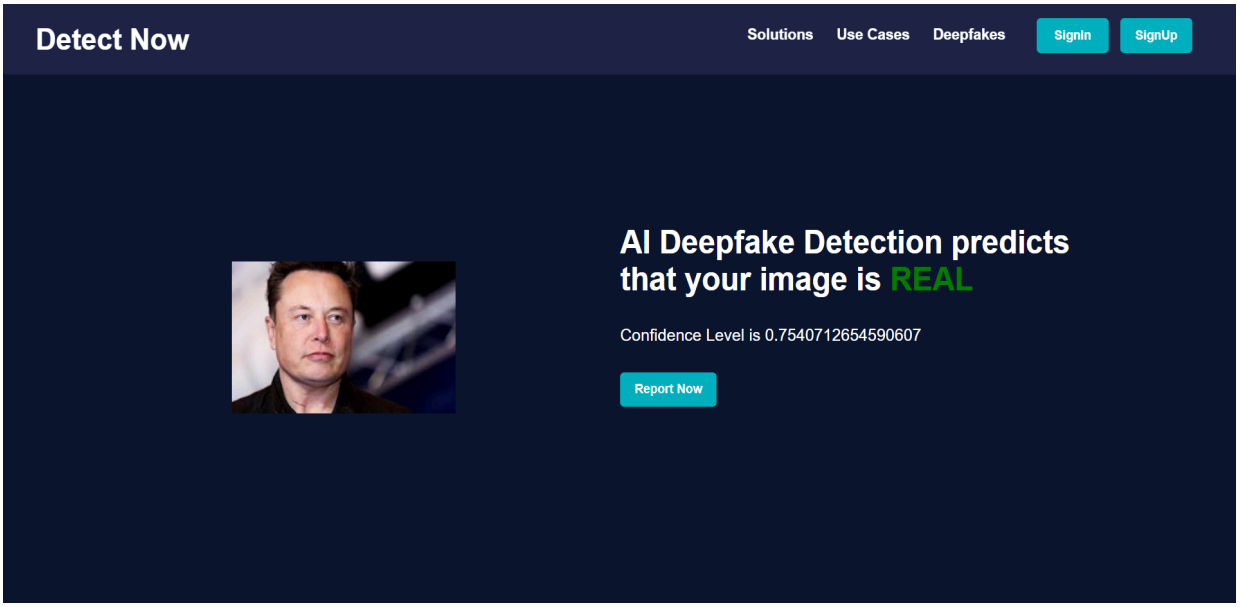


Fig 7.14: Model prediction for Elon Musk's Real Image

Upon receiving uploaded images, our model swiftly distinguishes between authentic and deepfake media with accuracy. Notably, it reliably identifies Fig. 7.11 to Fig. 7.13 images provided by the authors as authentic, showcasing the effectiveness of our detection system. This successful identification underscores the robustness of our detection system in discerning between genuine and manipulated visuals. The model's ability to accurately classify these images reaffirms its effectiveness in distinguishing authentic content from deceptive alterations. This validation further solidifies the reliability and efficacy of our detection mechanism in combating the spread of deepfake media.

Detection of Deepfake Images:

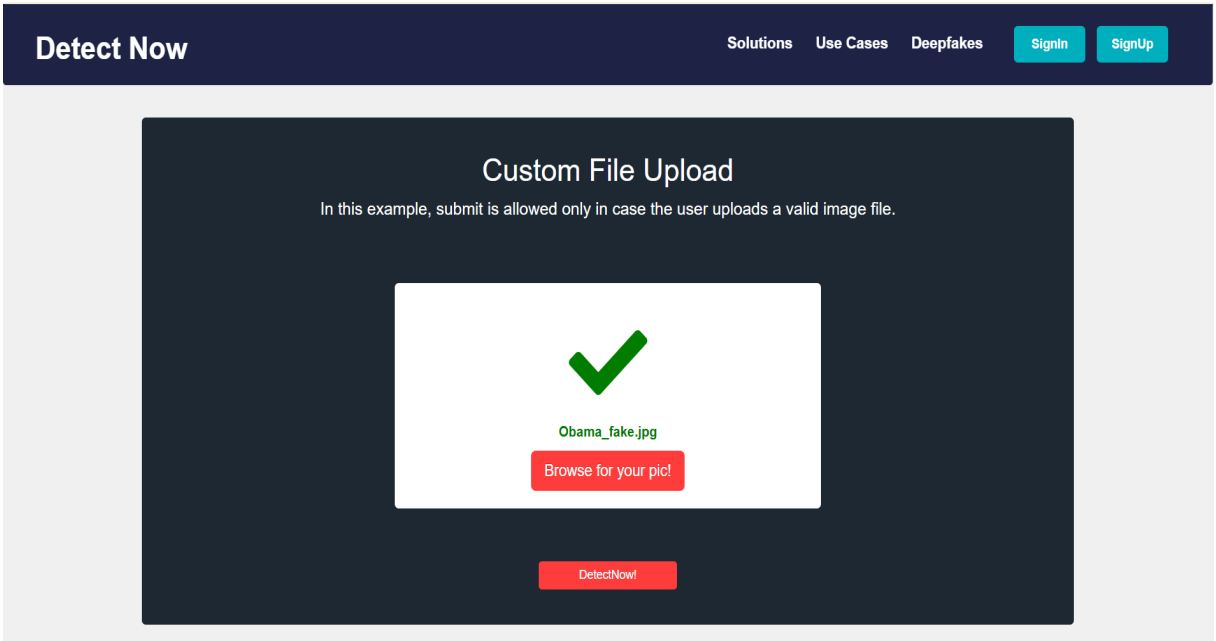


Fig 7.15: Choose Fake Image for Prediction

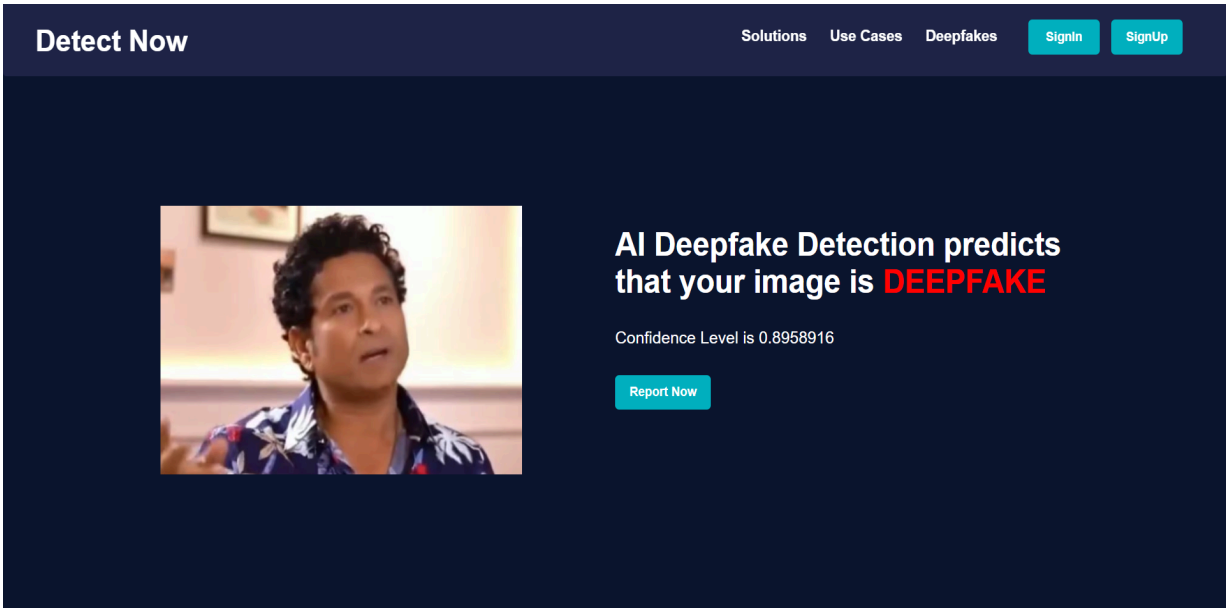


Fig 7.16: Model prediction for Sachin Tendulkar's DeepFake

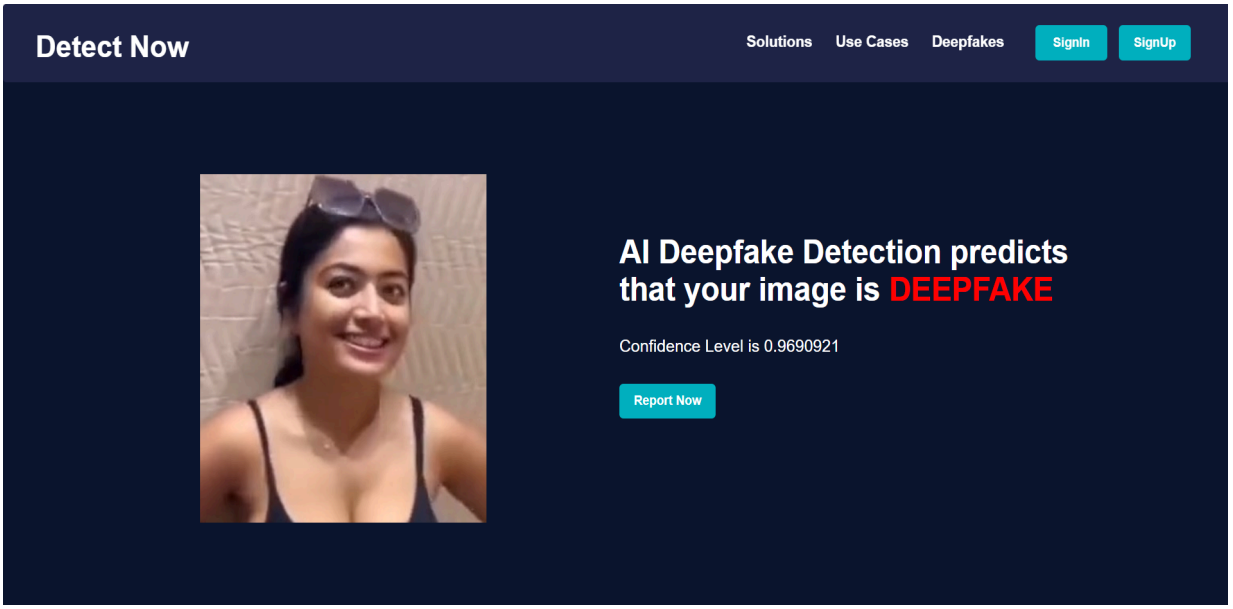


Fig 7.17: Model prediction for Female Celeb's DeepFake

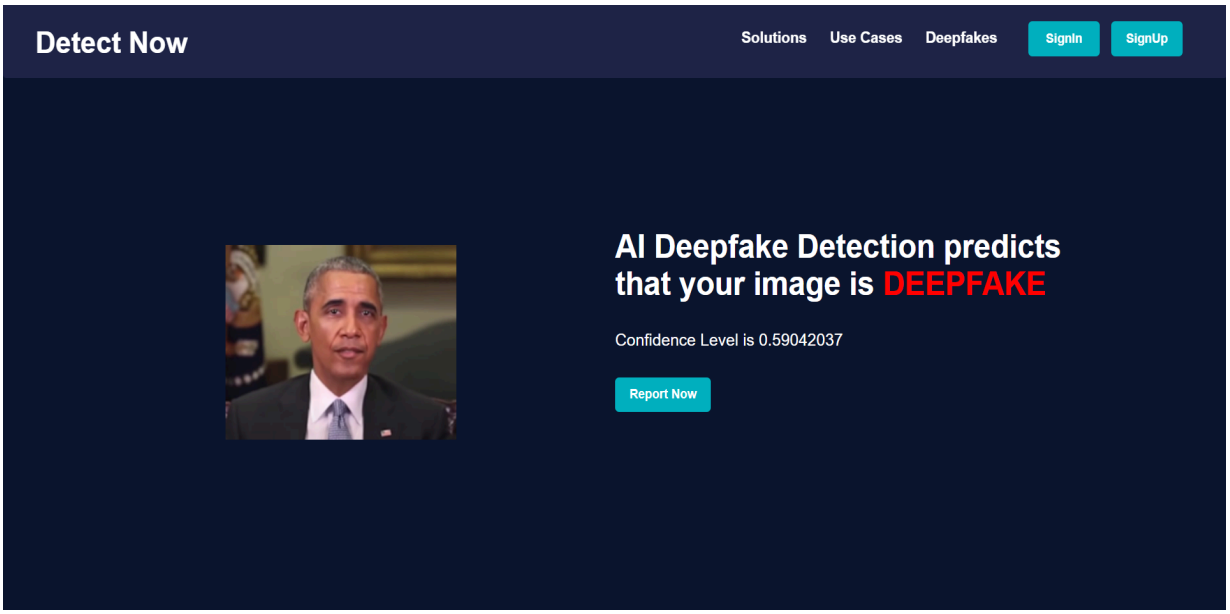


Fig 7.18: Model prediction for Obama's DeepFake

Following image processing, our model accurately flags manipulated media as shown in Fig. 7.14 to Fig. 7.16, notably identifying DeepFake content featuring notable figures like Rashmika Mandana and renowned cricketers. This successful detection underscores the robustness of our system in discerning deceptive visuals, enhancing trust and reliability in combating the proliferation of fake media. The results from the deepfake detection model demonstrate a commendable level of consistency and reliability in accurately distinguishing between real and fake images or frames. This achievement is attributed to the model's adept ability to discern intricate facial features.

By employing sophisticated convolutional neural network architectures, the model efficiently extracted and analyzed various facial attributes crucial for precise classification. These architectural intricacies allowed the model to effectively navigate the complexities inherent in deepfake images, resulting in robust and reliable performance across diverse datasets.

Furthermore, the utilization of convolutional neural networks facilitated a deep understanding of facial features, enabling the model to make informed predictions with high accuracy. The intricate layers within the neural network architecture enabled the extraction of nuanced details from both real and manipulated images, ensuring that even subtle alterations indicative of deepfakes were identified with remarkable precision. As a consequence, the model's predictive capabilities were consistently reliable, instilling confidence in its ability to differentiate between authentic and manipulated content, thereby bolstering the overall efficacy of the deepfake detection system.

In essence, the successful performance of the deepfake detection model underscores the efficacy of leveraging convolutional neural network architectures for intricate image analysis tasks. Through the adept extraction and analysis of facial attributes, the model demonstrated robustness in accurately discerning between genuine and manipulated content. This accomplishment not only highlights the advancements in machine learning techniques but also underscores the critical role such technologies play in combating the proliferation of deceptive content in today's digital landscape.

CHAPTER 8
CONCLUSION AND FUTURE SCOPE

8.1 Conclusion

In conclusion, our endeavor to combat the proliferation of deepfake content has yielded promising results through the implementation of diverse neural network architectures. The application of a personalized CNN model facilitated detection of manipulated images, while the fine-tuned Xception pretrained model demonstrated proficiency in discerning subtle anomalies indicative of deepfake manipulation. Additionally, the integration of an InceptionV3-based model for video analysis showcased a comprehensive approach, effectively detecting a spectrum of potential manipulations, from traditional facial replacement to sophisticated reenactment techniques.

Notably, our approach attains notable accuracy by examining only a single second of video, analyzing 5 frames every second. It integrates a pre-trained ResNext CNN model to extract features at the frame level and employs InceptionV3 for temporal sequence analysis, efficiently detecting alterations between successive frames.. Initially, the pre-trained model yielded an accuracy of 60%, but through strategic training techniques, such as freezing certain layers during the initial epochs and subsequently unfreezing all layers for further refinement over 10 epochs, we achieved a notable increase in accuracy, reaching 80.8%. This significant improvement underscores the effectiveness of our approach in enhancing model performance over successive training iterations.

The entire implementation was conducted using TensorFlow and Keras libraries, providing a robust and efficient framework for developing and training deep learning models for deepfake detection. This implementation provided valuable insights into the effectiveness of custom CNN architectures versus pretrained models for deepfake detection, paving the way for future research and development in this critical area of study.

8.2 Future Scope

Currently, most DNN-based deepfake detection systems are batch-based. This is not ideal for real-time applications, such as live video streaming or social media platforms. Future research should focus on developing real-time DNN-based deepfake detection systems.

- Deepfake technology is not limited to images and videos. It can also be used to create

fake audio, text, and even 3D models. Future research should focus on developing DNN models that can detect deepfakes in all of these modalities.

- Future detection systems can be integrated with other technologies, such as digital forensics and watermarking, to improve their effectiveness. For example, digital forensics techniques can be used to extract additional information from deepfakes that can help DNN models to make more accurate predictions.
- Researching transfer learning methods that enable detection models to swiftly adjust to emerging forms of DeepFakes using minimal data, such as few-shot learning, could prove advantageous.

REFERENCES

[1] D. Pan, L. Sun, R. Wang, X. Zhang and R. O. Sinnott, "Deepfake Detection through Deep Learning," 2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT), Leicester, UK, 2020, pp. 134-143, doi: 10.1109/BDCAT50828.2020.00001.

[2] S. A. Aduwala, M. Arigala, S. Desai, H. J. Quan and M. Eirinaki, "Deepfake Detection using GAN Discriminators," 2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService), Oxford, United Kingdom, 2021, pp. 69-77, doi: 10.1109/BigDataService52369.2021.00014.

[3] Bansal, N.; Aljrees, T.; Yadav, D.P.; Singh, K.U.; Kumar, A.; Verma, G.K.; Singh, T. Real-Time Advanced Computational Intelligence for Deep Fake Video Detection. Appl. Sci. 2023, 13, 3095. https://doi.org/10.3390/app13053095

[4] Tariq, S., Lee, S. and Woo, S., A convolutional lstm based residual network for deepfake video detection. arXiv 2020. arXiv preprint arXiv:2009.07480.

[5] V. Jolly, M. Telrandhe, A. Kasat, A. Shitole and K. Gawande, "CNN based Deep Learning model for Deepfake Detection," 2022 2nd Asian Conference on Innovation in Technology (ASIANCON), Ravet, India, 2022, pp. 1-5, doi: 10.1109/ASIANCON55314.2022.9908862.

[6] David Guera, Edward J. Delp, “Deepfake Video Detection Using Recurrent Neural Networks”, 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS).

[7] Badhrinarayan Malolan, Ankit Parekh, Faruk Kazi ,“Explainable Deep-Fake Detection Using Visual Interpretability Methods”,2020 3rd International conference on Information and Computer Technologies(ICICT).

[8] Irene Amerini, Leonardo Galteri, Roberto Caldelli, Alberto Del Bimbo, “Deepfake Video Detection through Optical Flow based CNN”, 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW).

[9] Mingzhu Luo, Yewei Xiao, Yan Zhou,“ Multi-scale face detection based on convolutional neural network”,IEEE 2018.

[10] Digvijay Yadav, Sakina Salmani, “Deepfake: A Survey on Facial Forgery Technique Using Generative Adversarial Network”, Proceedings of the International Conference on Intelligent Computing and Control Systems (ICICCS 2019).IEEE Xplore Part Number: CFP19K34-ART; ISBN: 978-1-5386-8113-8.

[11] Ranjan, Sarvesh Patil, Faruk Kazi,“Improved Generalizability of Deep-Fakes Detection Using Transfer Learning Based CNN Framework”,(IEEE 2020).

[12] Shivangi Aneja, Matthias Nießner, “Generalized Zero and Few-Shot Transfer for Facial Forgery Detection”, arXiv:2006.11863v1[cs.CV] 2020.

[13] Chuan-Chuan Low, Lee-Yeng Ong, Voon-Chet Koo,“Experimental Study on Multiple Face Detection with Depth and Skin Colour”,IEEE 2019.

[14] Xin Tao, Hongyun Gao, Renjie Liao, Jue Wang, Jiaya Jia, “Detail-revealing Deep Video Super-resolution”, 2017 IEEE International Conference on Computer Vision (ICCV)

[15] Faten F Kharbat, Tarik Elamsy, Ahmed Mahmoud, Rami ,“Image Feature Detectors for Deepfake Video Detection”,IEEE 2019.

[16] H. Ilyas, A. Irtaza, A. Javed and K. M. Malik, "Deepfakes Examiner: An End-to-End Deep Learning Model for Deepfakes Videos Detection," 2022 16th International Conference on Open Source Systems and Technologies (ICOSST), Lahore, Pakistan, 2022, pp. 1-6, doi: 10.1109/ICOSST57195.2022.10016871.

[17] K. Jalui, A. Jagtap, S. Sharma, G. Mary, R. Fernandes and M. Kolhekar, "Synthetic Content Detection in Deepfake Video using Deep Learning," 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2022, pp. 01-05, doi: 10.1109/GCAT55367.2022.9972081.

[18] A. Rahman et al., "Short And Low Resolution Deepfake Video Detection Using CNN," 2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC), Hyderabad, India, 2022, pp. 259-264, doi: 10.1109/R10-HTC54060.2022.9929719.

[19] L. S and K. Sooda, "DeepFake Detection Through Key Video Frame Extraction using GAN," 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2022, pp. 859-863, doi: 10.1109/ICACRS55517.2022.10029095.

[20] S. R. B. R, P. Kumar Pareek, B. S and G. G, "Deepfake Video Detection System Using Deep Neural Networks," 2023 IEEE International Conference on Integrated Circuits and Communication Systems (ICICACS), Raichur, India, 2023, pp. 1-6, doi: 10.1109/ICICACS57338.2023.10099618.

[21] J. Zhang, K. Cheng, G. Sovernigo and X. Lin, "A Heterogeneous Feature Ensemble Learning based Deepfake Detection Method," ICC 2022 - IEEE International Conference on Communications, Seoul, Korea, Republic of, 2022, pp. 2084-2089, doi: 10.1109/ICC45855.2022.9838630.

[22] Y. Nirkin, L. Wolf, Y. Keller and T. Hassner, "DeepFake Detection Based on Discrepancies Between Faces and Their Context," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 10, pp. 6111-6121, 1 Oct. 2022, doi: 10.1109/TPAMI.2021.3093446.

[23] D. Pan, L. Sun, R. Wang, X. Zhang and R. O. Sinnott, "Deepfake Detection through Deep Learning," 2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT), Leicester, UK, 2020, pp. 134-143, doi: 10.1109/BDCAT50828.2020.00001.

[24] Bansal, N.; Aljrees, T.; Yadav, D.P.; Singh, K.U.; Kumar, A.; Verma, G.K.; Singh, T. Real-Time Advanced Computational Intelligence for Deep Fake Video Detection. Appl. Sci. 2023, 13, 3095. https://doi.org/10.3390/app13053095

[25] S. Saif, S. Tehseen, S. S. Ali, S. Kausar and A. Jameel, "Generalized Deepfake Video Detection Through Time-Distribution and Metric Learning," in IT Professional, vol. 24, no. 2, pp. 38-44, 1 March-April 2022, doi: 10.1109/MITP.2022.3168351.

[26] M. Ramon, M. Vowels and M. Groh, "Deepfake Detection in Super-Recognizers and Police Officers," in IEEE Security & Privacy, doi: 10.1109/MSEC.2024.3371030.

[27] P. Sun, H. Qi, Y. Li and S. Lyu, "FakeTracer: Catching Face-swap DeepFakes via Implanting Traces in Training," in IEEE Transactions on Emerging Topics in Computing, doi: 10.1109/TETC.2024.3386960.

[28] R. Yang, Z. Deng, Y. Zhang, X. Luo and R. Lan, "4DPM: Deepfake Detection With a Denoising Diffusion Probabilistic Mask," in IEEE Signal Processing Letters, vol. 31, pp. 914-918, 2024, doi: 10.1109/LSP.2024.3378127.

[29] R. Yang, Z. Deng, Y. Zhang, X. Luo and R. Lan, "4DPM: Deepfake Detection With a Denoising Diffusion Probabilistic Mask," in IEEE Signal Processing Letters, vol. 31, pp. 914-918, 2024, doi: 10.1109/LSP.2024.3378127.

[30] S. A. Raza, U. Habib, M. Usman, A. A. Cheema and M. S. Khan, "MMGANGuard: A Robust Approach for Detecting Fake Images Generated by GANs using Multi-Model Techniques," in IEEE Access, doi: 10.1109/ACCESS.2024.3393842.

[31] T. Qiao, H. Shao, S. Xie and R. Shi, "Unsupervised Generative Fake Image Detector," in IEEE Transactions on Circuits and Systems for Video Technology, doi: 10.1109/TCSVT.2024.3383833.

[32] C. Peng, Z. Miao, D. Liu, N. Wang, R. Hu and X. Gao, "Where Deepfakes Gaze at? Spatial-Temporal Gaze Inconsistency Analysis for Video Face Forgery Detection," in IEEE Transactions on Information Forensics and Security, doi: 10.1109/TIFS.2024.3381823.

[33] J. Park, L. H. Park, H. E. Ahn and T. Kwon, "Coexistence of Deepfake Defenses: Addressing the Poisoning Challenge," in IEEE Access, vol. 12, pp. 11674-11687, 2024, doi: 10.1109/ACCESS.2024.3353785.

[34] T. Mahmud, I. Hasan, M. T. Aziz, T. Rahman, M. S. Hossain and K. Andersson, "Enhanced Fake News Detection through the Fusion of Deep Learning and Repeat Vector Representations," 2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), Bengaluru, India, 2024, pp. 654-660, doi: 10.1109/IDCIoT59759.2024.10467839.

[35] N. U. Thejas, H. D. Nayak, A. B. Siddique, M. Anas Danish and H. R. Mamatha, "Preprocessing and Feature Extraction based Deepfake Detection on Combined dataset," 2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI), Gwalior, India, 2024, pp. 1-6, doi: 10.1109/IATMSI60426.2024.10502574.

[36] K. N. Ramadhani, R. Munir and N. P. Utama, "Improving Video Vision Transformer for Deepfake Video Detection Using Facial Landmark, Depthwise Separable Convolution and Self Attention," in IEEE Access, vol. 12, pp. 8932-8939, 2024, doi: 10.1109/ACCESS.2024.3352890.

[37] K. N. Ramadhani, R. Munir and N. P. Utama, "Improving Video Vision Transformer for Deepfake Video Detection Using Facial Landmark, Depthwise Separable Convolution and Self Attention," in IEEE Access, vol. 12, pp. 8932-8939, 2024, doi: 10.1109/ACCESS.2024.3352890.

[38] A. Gupta and D. Pandey, "Unmasking the Illusion: Deepfake Detection through MesoNet," 2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT), Greater Noida, India, 2024, pp. 1934-1938, doi: 10.1109/IC2PCT60090.2024.10486617.

[39] Z. Yu, R. Cai, Z. Li, W. Yang, J. Shi and A. C. Kot, "Benchmarking Joint Face Spoofing and Forgery Detection With Visual and Physiological Cues," in IEEE Transactions on Dependable and Secure Computing, doi: 10.1109/TDSC.2024.3352049.

[40] M. Kumar, P. K. Rai and P. Kumar, "A Novel Approach for Detecting Deepfake Face Using Machine Learning Algorithms," 2024 2nd International Conference on Disruptive Technologies (ICDT), Greater Noida, India, 2024, pp. 1588-1592, doi: 10.1109/ICDT61202.2024.10489036.

[41] N. C. Gowda, V. H N and D. R. Ramani, "Efficient Identification of DeepFake Images using CNN," 2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), Bengaluru, India, 2024, pp. 1542-1547, doi: 10.1109/IDCIoT59759.2024.10467555.

[42] H. -s. Shin, J. Heo, J. -h. Kim, C. -y. Lim, W. Kim and H. -J. Yu, "HM-CONFORMER: A Conformer-Based Audio Deepfake Detection System with Hierarchical Pooling and Multi-Level Classification Token Aggregation Methods," ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Seoul, Korea, Republic of, 2024, pp. 10581-10585, doi: 10.1109/ICASSP48485.2024.10448453.

[43] A. K. M. Rubaiyat Reza Habib, E. Elijah Akpan, B. Ghosh and I. K. Dutta, "Techniques to Detect Fake Profiles on Social Media Using the New Age Algorithms - A Survey," 2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2024, pp. 0329-0335, doi: 10.1109/CCWC60891.2024.10427620.

[44] M. S. Rana, M. Solaiman, C. Gudla and M. F. Sohan, "Deepfakes – Reality Under Threat?," 2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2024, pp. 0721-0727, doi: 10.1109/CCWC60891.2024.10427659.

[45] H. Chotaliya, M. A. Khatri, S. Kanojiya and M. Bivalkar, "Review: DeepFake Detection Techniques using Deep Neural Networks (DNN)," 2023 6th International Conference on Advances in Science and Technology (ICAST), Mumbai, India, 2023, pp. 480-484, doi: 10.1109/ICAST59062.2023.10454938.

PUBLISHED PAPERS

Table 8.1 : Review Paper Details

Sr. No.	Paper Title	Author Names	Conference/ Journal Name	Indexing	**Online ISBN / ISSN No.**	Impact factor	Cite Score	Status	Date
1	Review: DeepFake Detection Techniques using Deep Neural Networks (DNN)	Harsh Chotaliya, Adil Khatri, Shubham Kanojiya, Mandar Bivalkar	2023 6th International Conference on Advances in Science and Technology (ICAST)	Conference				Published	09th December 2023

SOMAIYA VIDYAVIHAR
K J Somaiya Institute of Technology
(Formerly known as K J Somaiya Institute of Engineering & Information Technology)
An Autonomous Institute permanently affiliated to University of Mumbai.

Somaiya TRUST

K. J. Somaiya Institute of Technology, Sion, Mumbai

An Autonomous Institute affiliated to University of Mumbai

6th International Conference
on
Advances in Science and Technology (ICAST-2023)

CERTIFICATE

This is to certify that

Harsh Chotaliya

has participated / presented a paper titled

Review: Deep Fake Detection Techniques using Deep Neural Networks (DNN)

Published at the **6th IEEE International Conference on Advances in Science and Technology (ICAST-2023)**
organized by
K. J. Somaiya Institute of Technology, Sion, Mumbai - 22
in association with **University of Mumbai** and technically co-sponsored by **IEEE Bombay Section**
on 8th and 9th of December 2023.

ICAST KJSIT | IEEE BOMBAY SECTION | University of Mumbai

Dr. Sunita R. Patil
Convenor - ICAST 2023
Vice Principal, KJSIT

Dr. Vivek Sunnapwar
Principal, KJSIT

SOMAIYA VIDYAVIHAR
K J Somaiya Institute of Technology
(Formerly known as K J Somaiya Institute of Engineering & Information Technology)
An Autonomous Institute permanently affiliated to University of Mumbai.

Somaiya TRUST

K. J. Somaiya Institute of Technology, Sion, Mumbai

An Autonomous Institute affiliated to University of Mumbai

6th International Conference
on
Advances in Science and Technology (ICAST-2023)

CERTIFICATE

This is to certify that

Mohammed Adil Khatri

has participated / presented a paper titled

Review: Deep Fake Detection Techniques using Deep Neural Networks (DNN)

Published at the **6th IEEE International Conference on Advances in Science and Technology (ICAST-2023)**
organized by
K. J. Somaiya Institute of Technology, Sion, Mumbai - 22
in association with **University of Mumbai** and technically co-sponsored by **IEEE Bombay Section**
on 8th and 9th of December 2023.

ICAST KJSIT | IEEE BOMBAY SECTION | University of Mumbai

Dr. Sunita R. Patil
Convenor - ICAST 2023
Vice Principal, KJSIT

Dr. Vivek Sunnapwar
Principal, KJSIT

SOMAIYA VIDYAVIHAR
K J Somaiya Institute of Technology
(Formerly known as K J Somaiya Institute of Engineering & Information Technology)
An Autonomous Institute permanently affiliated to University of Mumbai

Somaiya TRUST

K. J. Somaiya Institute of Technology, Sion, Mumbai

An Autonomous Institute affiliated to University of Mumbai

6th International Conference
on
Advances in Science and Technology (ICAST-2023)

CERTIFICATE

This is to certify that

Shubham Kanojiya

has participated / presented a paper titled

Review: Deep Fake Detection Techniques using Deep Neural Networks (DNN)

Published at the **6th IEEE International Conference on Advances in Science and Technology (ICAST-2023)**
organized by
K. J. Somaiya Institute of Technology, Sion, Mumbai - 22
in association with **University of Mumbai** and technically co-sponsored by **IEEE Bombay Section**
on 8th and 9th of December 2023.

ICAST

IEEE BOMBAY SECTION

University of Mumbai

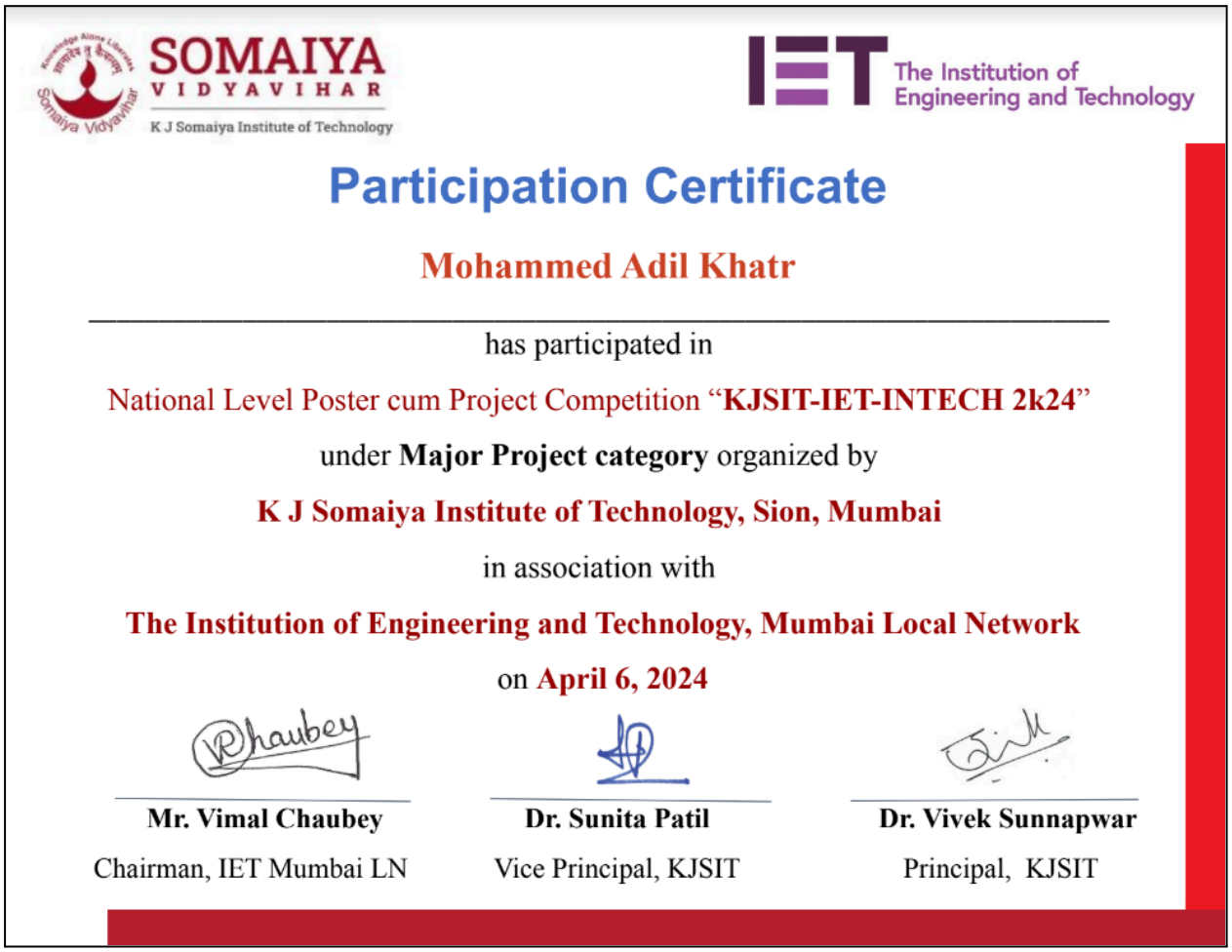
Dr. Sunita R. Patil
Convenor - ICAST 2023
Vice Principal, KJSIT

Dr. Vivek Sunnapwar
Principal, KJSIT

CERTIFICATE DETAILS

Table 8.2 : Competition Details

Sr. No.	Project Title	Group Members	Guide's Name	Competition Name	Venue	Status (Prize/ Participation)	Mode of Participation	Date
1	Deepfake Detection using DNN	Harsh Chotaliya	Dr. Mandar Bivalkar	KJSIT-IET-INTECH 2K23	K. J. Somaiya Institute of Technology	Partcipation	Offline	6th April 2024
2	Deepfake Detection using DNN	Adil Khatri	Dr. Mandar Bivalkar	KJSIT-IET-INTECH 2K23	K. J. Somaiya Institute of Technology	Partcipation	Offline	6th April 2024
3	Deepfake Detection using DNN	Shubham Kanojiya	Dr. Mandar Bivalkar	KJSIT-IET-INTECH 2K23	K. J. Somaiya Institute of Technology	Partcipation	Offline	6th April 2024



SOMAIYA VIDYAVIHAR
K J Somaiya Institute of Technology

The Institution of Engineering and Technology

Participation Certificate

Mohammed Adil Khatr

has participated in

National Level Poster cum Project Competition "**KJSIT-IET-INTECH 2k24**"

under **Major Project category** organized by

K J Somaiya Institute of Technology, Sion, Mumbai

in association with

The Institution of Engineering and Technology, Mumbai Local Network

on **April 6, 2024**

Mr. Vimal Chaubey
Chairman, IET Mumbai LN

Dr. Sunita Patil
Vice Principal, KJSIT

Dr. Vivek Sunnapwar
Principal, KJSIT

SOMAIYA VIDYAVIHAR
K J Somaiya Institute of Technology

The Institution of Engineering and Technology

Participation Certificate

Harsh Chotaliya

has participated in

National Level Poster cum Project Competition "**KJSIT-IET-INTECH 2k24**"

under **Mini / Minor Project category** organized by

K J Somaiya Institute of Technology, Sion, Mumbai

in association with

The Institution of Engineering and Technology, Mumbai Local Network

on **April 6, 2024**

Mr. Vimal Chaubey	**Dr. Sunita Patil**	**Dr. Vivek Sunnapwar**
Chairman, IET Mumbai LN	Vice Principal, KJSIT	Principal, KJSIT

SOMAIYA VIDYAVIHAR
K J Somaiya Institute of Technology

The Institution of Engineering and Technology

Participation Certificate

Shubham Kanojiya

has participated in

National Level Poster cum Project Competition "**KJSIT-IET-INTECH 2k24**"

under **Major Project category** organized by

K J Somaiya Institute of Technology, Sion, Mumbai

in association with

The Institution of Engineering and Technology, Mumbai Local Network

on **April 6, 2024**

Mr. Vimal Chaubey	**Dr. Sunita Patil**	**Dr. Vivek Sunnapwar**
Chairman, IET Mumbai LN	Vice Principal, KJSIT	Principal, KJSIT