DEEPFAKE VOICE DETECTION USING DEEP LEARNING TECHNIQUE

A PROJECT REPORT

Submitted by

RAJEE P (2022503052)

MAHASWIN G (2022503506)

VIJAY ROSHAN S (2022503576)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

MADRAS INSTITUTE OF TECHNOLOGY, CHROMEPET

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2025

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**Deepfake voice detection using Deep Learning Technique**" is the bonafide work of **Ms.Rajee P (2022503052), Mr. Mahaswin G (2022503506), Mr. Vijay Roshan S (2022503576)** who carried out the project under my supervision.

SIGNATURE

Dr. P. Jayashree
HEAD OF THE DEPARTMENT

Department of Computer Technology
Anna University, MIT Campus
Chromepet – 600 044

SIGNATURE

Dr. P. Pabitha
SUPERVISOR
Associate Professor
Department of Computer Technology
Anna University, MIT Campus
Chromepet – 600 044

ABSTRACT

Deepfake audio, generated using advanced Text-to-Speech (TTS) and Voice Conversion (VC) techniques, has become a serious threat to security, privacy, and trust in digital communications. This technology allows malicious actors to convincingly mimic genuine voices, enabling impersonation and misinformation on a large scale. As deepfake audio continues to evolve, the potential for misuse grows, affecting not only individuals but also organizations and governments.

It poses a significant risk in financial fraud, where attackers manipulate victims into transferring money or revealing sensitive information. Political manipulation is another concern, as deepfake audio can be used to create fake statements attributed to public figures, influencing public opinion.

This project presents a novel approach for deepfake audio detection by training a Convolutional Neural Network model with variations in the layers, supporting the features extracted. Multi-channel feature extraction captures comprehensive audio features, followed by a series of data transformation which helps in standardizing the huge amount of data, enabling robust detection by leveraging learned features from mel-spectrogram, and log-power spectra, addressing limitations of other recent methods.

This approach enhances detection accuracy by integrating both temporal and spectral audio characteristics, improving the model's ability to distinguish subtle differences between real and fake voices. The proposed method aims to contribute to the growing field of audio forensics, offering a scalable and effective solution to combat the rise of synthetic audio threats.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

S. No	ABBREVIATION	DEFINITION
1	Conv1D	1-Dimensional Convolutional Layer
2	ReLU	Rectified Linear Unit

CHAPTER 1

INTRODUCTION

1.1 DEEP LEARNING

Deep Learning is a subfield of machine learning that focuses on algorithms inspired by the structure and function of the human brain, known as artificial neural networks. Unlike traditional machine learning, which often requires manual feature extraction, deep learning models automatically learn to extract useful features from raw data, making them highly effective for complex tasks such as image recognition, natural language processing, and speech recognition.

At the heart of deep learning are deep neural networks (DNNs), which consist of multiple layers of interconnected nodes (or neurons). These layers include input, hidden, and output layers. The hidden layers are where the actual learning happens, as data is transformed through a series of nonlinear operations. The term "deep" refers to the use of many such layers, allowing the network to model intricate patterns and hierarchical relationships in the data.

One of the key innovations in deep learning is the use of backpropagation, a training technique that adjusts the weights of the network by minimizing the difference between the predicted and actual outputs using gradient descent. With the advent of powerful computational resources, such as GPUs, and the availability of large-scale datasets, deep learning has become feasible and widely adopted.

Popular architectures include Convolutional Neural Networks (CNNs) for image and video data, Recurrent Neural Networks (RNNs) and Transformers for sequential data like speech and text, and Generative Adversarial Networks (GANs) for generating synthetic data. These models have achieved state-of-the-art performance in various applications ranging from medical diagnostics to autonomous driving.

1.2 DEEPFAKE VOICE

Deepfake voice refers to synthetic audio that mimics a real person's voice using advanced artificial intelligence techniques such as Text-to-Speech (TTS) and Voice Conversion (VC). These technologies, powered by deep learning, can generate highly realistic speech that is nearly indistinguishable from genuine human voices. While they have beneficial applications in entertainment, accessibility, and virtual assistants, deepfake voices pose significant threats to security, privacy, and public trust.

Malicious actors can exploit deepfake audio to impersonate individuals, commit financial fraud, or spread misinformation. For instance, attackers may use fake voices to trick people into transferring money or disclosing sensitive information, or fabricate public statements to manipulate opinions. As the technology continues to evolve, the ability to detect deepfake voices becomes increasingly critical for protecting individuals and organizations from deception and harm.

1.2.1 HOW DEEPFAKE VOICE IS GENERATED?

Deepfake voices are created using advanced artificial intelligence (AI) and deep learning techniques, primarily involving Text-to-Speech (TTS) and Voice Conversion (VC) models. These technologies are designed to generate human-like speech or transform one person’s voice into another's while preserving naturalness and emotional tone. The process involves multiple steps and powerful neural network models that learn the unique characteristics of a person’s voice.

1. Voice Data Collection

To generate a deepfake voice, the first step is to collect audio recordings of the target speaker. These recordings are often several minutes to hours long and contain various speech patterns, tones, and emotions. The more diverse and high-quality the dataset, the more realistic the generated voice will be.

2. Feature Extraction

Next, the raw audio is transformed into acoustic features like mel-spectrograms, pitch, and intonation patterns. These features capture essential elements of speech, such as tone, rhythm, and frequency, which are then used to train the AI model.

3. Model Training

Using deep learning architectures such as Recurrent Neural Networks (RNNs), Transformer models, or Variational Autoencoders (VAEs), the system learns to map text or source speech features to the target speaker’s voice characteristics. TTS systems generate speech from text, while VC systems convert one speaker’s voice into another's using a learned mapping.

4. Voice Synthesis

Once trained, the model can generate new speech by inputting any desired text (TTS) or source audio (VC). The system then synthesizes this input into speech that closely mimics the target person's voice, often with alarming accuracy.

5. Post-processing

The generated voice may undergo enhancements to improve naturalness and reduce noise. Techniques such as waveform generation using models like WaveNet or HiFi-GAN are often applied for high-quality audio output.

1.2.1 CLASSIFICATION OF REAL VOICES AND FAKE VOICES

Features Used for Classification

a. Spectral Features

Mel-Spectrogram: Visual representation of the audio's frequency content over time, capturing pitch and timbre.

Log Power Spectrogram: Provides information on energy distribution across frequency bands.

MFCC (Mel-Frequency Cepstral Coefficients): Captures short-term power spectrum of sound, useful in speech recognition.

Chroma Features: Represent pitch class content, useful in musical or tonal audio analysis.

b. Prosodic Features

Pitch (F0 contour): Variations can be unnatural in deepfake voices.

Energy/Amplitude: Deepfakes often fail to replicate natural volume fluctuations.

Speech Rate and Duration: May differ subtly in synthetic voices.

c. Voice Quality Features

Jitter and Shimmer: Measures of frequency and amplitude variations—typically less variable in deepfakes.

Harmonics-to-Noise Ratio (HNR): Indicates breathiness or roughness in voice.

d. Latent Features

Features learned by autoencoders, transformers, or CNNs during training, used as compressed representations for classification.

Model Structures for Classification

a. Convolutional Neural Networks (CNNs)

- Effective for learning local patterns in spectrograms.
- Commonly used in image-based audio representations.

b. Recurrent Neural Networks (RNNs) / LSTM / GRU

- Capture temporal dependencies in speech sequences.
- Useful when using sequential features like MFCCs or pitch.

c. Transformers

- Self-attention mechanisms enable capturing long-range dependencies across audio frames.
- Examples: Wav2Vec, AST (Audio Spectrogram Transformer).

d. Autoencoder-Based Models

- Train autoencoders on real audio only; use reconstruction error as a feature for classification.
- Often combined with classifiers like MLP, CNN or ResNet.

e. Ensemble Models

I. Combine multiple classifiers or feature sets for improved accuracy and robustness.

1.3 DEEPFAKE VOICE DETECTION

Deepfake voice detection typically involves analyzing acoustic features and identifying subtle artifacts or inconsistencies in synthetic speech. Real human voices exhibit natural variations in pitch, tone, energy, and timing, while fake voices often lack such variability or display unnatural patterns. By examining these cues, machine learning and deep learning models can distinguish between genuine and fake audio samples.

As deepfake generation methods improve, detection becomes increasingly complex. Some deepfakes can replicate breathing, emotional tone, and ambient noise, making them harder to detect. To combat this, researchers are developing multi-channel detection systems that combine handcrafted features with deep-learned representations and utilize ensemble models for better robustness.

Deepfake voice detection plays a vital role in safeguarding digital integrity. With the growing accessibility of voice cloning tools, it's essential to stay ahead with innovative detection strategies that can adapt to evolving threats. The future

of secure communication depends on our ability to reliably distinguish real voices from synthetic ones.

1.4 OBJECTIVES

- To develop a robust deep learning-based system capable of accurately distinguishing between real and deepfake audio using spectral and prosodic features.
- To explore and evaluate the effectiveness of various feature extraction methods (e.g., mel-spectrograms, MFCCs, log power spectra) in enhancing the detection of synthetic voice artifacts.
- To improve the detection accuracy and generalization of the model by integrating handcrafted features with learned representations through multi-channel or ensemble architectures.

CHAPTER 2

LITERATURE SURVEY

[1] uses Frequency-time domain knowledge distillation which uses a teacher-student framework for deepfake audio detection. The teacher learns complex frequency and time features from high-quality audio, while the lightweight student learns from compressed audio. It needs paired High and low quality audio

The Deep4SNet classification model in [2] and the designed adversarial attacks leverage a generative adversarial network architecture and reduce the detector's accuracy to nearly 0%

[3] utilized spectral, temporal, chroma, and frequency-domain features as inputs to simplified ML architectures, distinguishing between genuine and spoofed audio.

[4] proposes a dual-branch network-based model for manipulative speech detection, utilizing fused Mel features. The overall structure of the network can be divided into three parts: speech signal processing, time–frequency feature extraction, and feature fusion with decision-making.

[5] uses a dual-stream architecture for AI-synthesized speech detection, combining feature decomposition and synthesizer feature augmentation. The synthesizer stream captures specific artifacts, while the content stream learns independent features using pseudo-labeling and adversarial learning.

[6] uses the Segment-based Anti-Spoofing Detection (SASD) method which detects spoofed speech by segmenting audio into words and silence, extracting WCQCC and AZCR features. A Biased Decision Strategy (BDS) classifies speech

as bona fide or spoofed, enabling faster detection by focusing on anti-spoofing features.

CHAPTER 3

PROPOSED WORK

3.1 INTRODUCTION

Deepfake audio, generated using advanced Text-to-Speech (TTS) and Voice Conversion (VC) techniques, has become a serious threat to security, privacy, and trust in digital communications. This technology allows malicious actors to convincingly mimic genuine voices, enabling impersonation and misinformation on a large scale. It poses a significant risk in financial fraud, where attackers manipulate victims into transferring money or revealing sensitive information.

Deepfake audio detection faces significant challenges when applied to low-quality, compressed speech used in real-world applications such as telephone conversations, social media voice messages. This research aims to develop detection techniques capable of identifying deepfake audio in compressed and noisy environments.

The Proposed Architecture diagram shown in Figure 3.1 includes:

I. Data Collection:

The first step in the pipeline is to collect large amount of real and fake audio samples to train the model.

II. Data Preprocessing:

The next step is to clean the audio samples by removing the background noises and trimming the leading and trailing silence which reduces the sample size without compromising the features in it.

III. Feature Extraction:

After cleaning the audio samples, the next step is to extract the required features from the samples. In our work, we mainly focus on Mel spectrogram and Log power Spectrum.

IV. Training the model:

After extracting the required features, the features are processed and trained in our proposed model, which classifies the label (Real or Fake) of the audio sample.

3.2 DATA COLLECTION

The ASVspoof 2021 DF dataset is designed for deepfake speech detection and was part of the Automatic Speaker Verification Spoofing and Countermeasures (ASVspoof) challenge 2021. The dataset includes real and spoofed audio from different synthetic speech generation techniques.

It contains both real speech and spoofed speech (generated through various deepfake and synthetic speech methods).

The spoofed audio is generated using methods such as Voice Conversion (VC), Text-to-Speech (TTS), and Neural Vocoder (NV) techniques and compression algorithms such as MP3, OGG, AAC, Opus.

The dataset includes around 100,000 utterances from multiple speakers (including both males and females), across different languages and environments.

Audio samples are typically provided in FLAC format at a 16 kHz sampling rate.

For our model, we have taken 1,000 real and 1,000 fake audio samples to train the Conv1D neural network. Random splitting is used to test the model.

3.3 DATA PREPROCESSING

3.3.1 NEED FOR DATA PROCESSING

Data preprocessing is essential to improve the performance and reliability of deepfake voice detection models. Noise reduction helps eliminate background disturbances that can mask subtle audio cues needed for accurate classification. Silence trimming at the start and end ensures that the model focuses on meaningful speech content rather than irrelevant gaps. Together, these steps enhance feature consistency and model generalization across diverse audio samples.

3.3.2 NOISE REDUCTION

Uses the noisereduce library to reduce background noise in the audio signal. The noisereduce Python library is based on spectral noise gating using Wiener filtering and power spectral density (PSD) estimation. The core formula used for noise reduction is derived from Wiener filtering.

3.3.3 SILENCE TRIMMING

Uses pydub to detect and remove leading and trailing silence based on a defined silence threshold. A common threshold value for detecting silence in pydub is -40 dBFS. This helps in standardizing audio lengths and focusing the model on

actual speech content rather than silent segments. Removing silence also reduces unnecessary computation and improves training efficiency.

3.4 FEATURE EXTRACTION

Feature Extraction is the process of extracting the required features (here, Mel spectrogram and Log power spectrum) from the given audio sample dataset. It uses the Python library Librosa to perform feature extraction. Librosa is a powerful and versatile Python library for analyzing and processing audio and music data. It provides tools for feature extraction, audio manipulation, and visualization. Using this, the model extracts mel-spectrogram (using STFT and Mel filter bank) and Log-power spectrum (power spectrogram and Logarithmic Transformation) from the audio input.

3.5 AUDIO FEATURE PREPARATION

The extracted features had to go through a series of steps in order to get standardized and normalized so that the model can be trained smoothly. To do that we incorporate the following steps:

i) Grouping features
ii) Normalizing the data
iii) Concatenating features
iv) Padding and Masking the data

3.5.1 GROUPING FEATURES

The extracted features grouped into real and fake features. The audios' ID is taken from the metadata and concatenated with the feature and saved as .npy file.

These files will be stored in the name of audio id followed by feature.

3.5.2 NORMALIZING FEATURES

The stored features are then normalized in the scale of [0,1] using MinMax Normalization. This ensures that all features contribute equally during model training, preventing dominance by features with larger values. Normalization also accelerates convergence and improves the stability of deep learning models.

3.5.3 CONCATENATING FEATURES

Standardizes the time dimension of all feature arrays by trimming them to the minimum length among all features. This ensures consistent shape for downstream processing. Combines multiple trimmed features into a single multi-channel array by stacking them along a new axis. This is useful for feeding into models expecting multi-dimensional input.

3.5.4 MASKING AND PADDING

The extracted features are then padded to a uniform shape using zero-padding. This ensures that all audio samples have the same time and frequency dimensions, which is required for batch processing in deep learning models. Padding helps in handling variable-length inputs and allows the model to learn consistently across the dataset.

3.6 CLASSIFICATION

The final stage of the model involves classifying each audio sample as either real or fake. The preprocessed and padded features are passed through Conv1D + residual model trained to detect subtle differences between genuine and synthetic speech. The model learns discriminative patterns using both spectral and temporal cues. Based on the learned features, the model outputs a probability or binary label indicating the authenticity of the voice.

The classification is done by the Conv1D + residual model. The CNN architecture is designed to effectively extract and learn features from the audio samples.

i. Convolutional Layer

The Conv1D layers are used to extract important temporal and spectral patterns from the audio features. They scan across the input with filters to detect local feature combinations such as frequency transitions. These layers are essential for learning hierarchical representations from raw or transformed audio. In this model, they serve as the foundation for deeper feature extraction before classification.

ii. Pooling layer

MaxPooling1D is used to downsample the feature maps, reducing dimensionality and computation. It keeps the most prominent features while discarding less relevant information. Pooling adds translation invariance, allowing the model to detect patterns regardless of position. This step helps improve generalization and accelerates training.

iii. Residual Layer

The residual blocks allow the model to learn complex patterns without suffering from vanishing gradients. They add the input (shortcut) back to the output of convolution layers, enabling better gradient flow. This structure helps in training deeper networks more efficiently by preserving information across layers. Residual connections enhance performance and stability during learning.

iv. ReLU

ReLU (Rectified Linear Unit) introduces non-linearity into the model by zeroing out negative values. It helps the network learn complex functions by enabling activation of only useful neurons. ReLU is computationally efficient and prevents issues like saturation that occur in sigmoid/tanh. In this model, it activates features after convolutions and residual blocks, aiding deep learning.

CHAPTER 4

IMPLEMENTATION

4.1 PLATFORMS USED

Google Colab

Google Colab is a cloud-based Jupyter Notebook environment that allows users to write and execute Python code interactively. It provides free access to GPU and TPU resources, making it suitable for training deep learning models. Colab is well-suited for machine learning, data science, and education tasks. It requires no setup to use and integrates with Google Drive, making it easy to access and share notebooks.

4.2 TOOLS USED

i. NumPy:

Used for efficient numerical operations and handling large multi-dimensional arrays, essential for feature processing. Helps in reshaping, stacking, and mathematical computations on audio data.

ii. Pandas:

Used for handling structured data like labels and metadata in tabular format. Makes it easy to read, filter, and merge data from sources like CSV or

Excel.

iii. Matplotlib:

A plotting library used to visualize audio signals, spectrograms, and training metrics. Essential for debugging and understanding model behavior over time.

iv. Seaborn:

Built on top of Matplotlib, it provides cleaner and more attractive statistical visualizations. Often used to plot confusion matrices and distribution plots.

v. Tensorflow / Keras:

TensorFlow is a deep learning framework, and Keras is its high-level API for building neural networks. Used here to define, compile, and train the deepfake voice detection model.

vi. Scikit-learn (sklearn)

Offers tools for data splitting, normalization, class balancing, and evaluation metrics. Used for preprocessing and generating reports like confusion matrix and classification score.

vii. Google Colab Drive

Allows integration of Google Drive with Colab notebooks. Used to load/save datasets and model checkpoints directly from Drive.

viii. Librosa

A Python library for audio analysis and feature extraction (e.g., mel-spectrograms, MFCCs). Also provides audio visualization and playback support.

ix. Soundfile

Used for reading and writing high-quality audio files (like WAV, FLAC). Preferred for lossless file handling in preprocessing pipelines.

x. Noisereduce

Applies noise reduction algorithms to clean the audio before feature extraction. Improves the quality of input data by removing background disturbances.

xi. Pydub

Simplifies audio operations like trimming, converting, or detecting silence. Commonly used for preprocessing tasks like silence removal at audio boundaries.

4.3 PSEUDO CODE FOR DATA CLEANING

Input: Both compressed Real and fake audio (16kHz).

Output: Cleaned Real and fake audio (16kHz)

Steps:

1. DEFINE FUNCTION trim_silence(audio, silence_thresh, min_silence_len)

2. COMPUTE start_trim using detect_leading_silence(audio, silence_thresh)

3. COMPUTE end_trim using detect_leading_silence(audio.reverse(), silence_thresh)

4. IF start_trim is less than total duration - end_trim:

5. RETURN trimmed audio (audio[start_trim : duration - end_trim])

6. ELSE:

7. RETURN original audio (prevent over-trimming)

8. DEFINE FUNCTION detect_leading_silence(sound, silence_threshold)

9. INITIALIZE trim_ms = 0

10. WHILE trim_ms < total length of sound AND sound volume is below silence_threshold:

11. INCREMENT trim_ms

12. RETURN trim_ms (amount of silence to remove)

13. APPLY trim_silence() on the loaded audio

14. SAVE cleaned audio in FLAC format to output_path

15. ITERATE through all files in input_folder:

16. IF file extension is ".flac":

17. SET input_path = full path to input file

18. SET output_path = full path to output file

4.4 PESUDO CODE FOR FEATURE EXTRACTION

Input: Cleaned Real and fake audio (16kHz)

Output: X_{mel} (mel spectrogram), X_{log} (log power spectrum)

Steps:

1.DEFINE FUNCTION extract_features(file_path, max_length)

2. IF file_path is not a valid audio:

3. RETURN None, None, None

4. LOAD audio signal 'y' and sample rate 'sr' from file_path with sr = 22050

5. PAD or TRIM waveform 'y' to fixed size (22050 samples)

6. COMPUTE mel_spec using mel-spectrogram on 'y' with 128 mel bands

7. CONVERT mel_spec to log scale using power_to_db

8. COMPUTE log_power_spec using STFT of 'y'

9. CONVERT amplitude to dB scale

10. PAD or TRIM mel_spec to 'max_length' along time axis

11. PAD or TRIM log_power_spec to 'max_length' along time axis

12. RETURN waveform, mel_spec, log_power_spec

4.5 PSEUDO CODE FOR FEATURE PREPROCESSING

Input: X_{mel} (mel spectrogram), X_{log} (log power spectrum).

Output: $X_{features}$

Steps:

1.FUNCTION load_and_group_features(folder_path):

2. INIT audio_features = {}

3. FOR EACH file IN folder_path:

4. IF file ENDS WITH '.npy':

5. SET file_path = JOIN(folder_path, file)

6. EXTRACT audio_id FROM file

7. IF audio_id NOT IN audio_features:

8. INIT audio_features[audio_id] = {}

9. IF 'mel_spec' IN file:

10. LOAD mel_spectrogram FROM file_path

11. STORE IN audio_features[audio_id]['mel_spectrogram']

12. ELSE IF 'log_power_spec' IN file:

13. LOAD log_spectrogram FROM file_path

14. STORE IN audio_features[audio_id]['log_spectrogram']

15. PRINT "Loaded {len(audio_features)} audio files."

16. RETURN audio_features

17.FUNCTION normalize_feature(feature):

18. INIT scaler = MinMaxScaler()

19. IF feature IS 1D:

20. RESHAPE feature TO 2D

21. APPLY scaler TO feature

22. RESHAPE BACK TO 1D

23. ELSE:

24. TRANSPOSE feature

25. APPLY scaler TO feature

26. TRANSPOSE BACK

27. RETURN normalized_feature

28.FUNCTION match_sample_size(features):

29. SET target_samples = MIN(

30. LENGTH(features['mel_spectrogram']),

31. LENGTH(features['log_spectrogram'])

32.)

33. TRUNCATE mel_spectrogram AND log_spectrogram TO target_samples

34. RETURN features

35.FUNCTION combine_features(features):

36. FLATTEN mel_spectrogram TO 2D

37. FLATTEN log_spectrogram TO 2D

38. CONCATENATE mel_spectrogram AND log_spectrogram ALONG COLUMNS

39. RETURN combined_features

40.FUNCTION pad_and_mask(features, max_feature_dim):

41. INIT padded_features = []

42. INIT masks = []

43. FOR EACH feature IN features:

44. IF feature COLUMNS < max_feature_dim:

45. PAD feature WITH ZEROS TO max_feature_dim

46. ELSE:

47. TRUNCATE feature TO max_feature_dim

48. CREATE mask FOR feature (1s FOR VALID, 0s FOR PADDED)

49. APPEND padded_feature TO padded_features

50. APPEND mask TO masks

51. RETURN padded_features, masks

4.6 PSEUDO CODE FOR CONVOLUTIONAL NEURAL NETWORK

Input: $X_{features}$

Output: Real/Fake (Classification)

Steps:

1.DEFINE FUNCTION residual_block(x, filters, kernel_size=3, stride=1)

2. SET shortcut = x

3. APPLY Conv1D with given filters, kernel_size, stride, same padding → x

4. APPLY BatchNormalization → x

5. APPLY ReLU activation → x

6. APPLY another Conv1D with same filters and kernel_size, same padding → x

7. APPLY BatchNormalization → x

8. IF stride ≠ 1 OR number of channels in shortcut ≠ filters:

9. APPLY Conv1D with filters and stride 1 → shortcut

10. APPLY BatchNormalization → shortcut

11. ADD x and shortcut (skip connection)

12. APPLY ReLU activation → x

13. RETURN x

14.DEFINE FUNCTION build_improved_model(input_shape)

15. DEFINE input layer with shape = input_shape → inputs

16. RESHAPE input to merge last two dimensions → x

17. APPLY Conv1D (128 filters, kernel size 7, same padding) → x

18. APPLY BatchNormalization → x

19. APPLY ReLU activation → x

20. APPLY MaxPooling1D (pool size 2) → x

21. APPLY residual_block with 128 filters → x

22. APPLY Dropout (rate 0.3) → x
23. APPLY residual_block with 256 filters and stride 2 → x
24. APPLY Dropout (rate 0.3) → x
25. APPLY residual_block with 512 filters and stride 2 → x
26. APPLY Conv1D (1 filter, kernel size 1, sigmoid activation) → attention
27. MULTIPLY x and attention (element-wise) → x
28. APPLY GlobalAveragePooling1D → x
29. APPLY Dense layer (256 units, ReLU activation) → x
30. APPLY Dropout (rate 0.4) → x
31. APPLY Dense layer (1 unit, sigmoid activation) → outputs
32. DEFINE model from inputs to outputs
33. RETURN model

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 128, 258, 2)	0	-
reshape (Reshape)	(None, 128, 516)	0	input_layer[0][0]
conv1d (Conv1D)	(None, 128, 128)	462,464	reshape[0][0]
batch_normalization (BatchNormalizatio…	(None, 128, 128)	512	conv1d[0][0]
re_lu (ReLU)	(None, 128, 128)	0	batch_normalizat…
max_pooling1d (MaxPooling1D)	(None, 64, 128)	0	re_lu[0][0]
conv1d_1 (Conv1D)	(None, 64, 128)	49,280	max_pooling1d[0]…
batch_normalizatio…	(None, 64, 128)	512	conv1d_1[0][0]

(BatchNormalizatio…			
re_lu_1 (ReLU)	(None, 64, 128)	0	batch_normalizat…
conv1d_2 (Conv1D)	(None, 64, 128)	49,280	re_lu_1[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 64, 128)	512	conv1d_2[0][0]
add (Add)	(None, 64, 128)	0	batch_normalizat… max_pooling1d[0]…
re_lu_2 (ReLU)	(None, 64, 128)	0	add[0][0]
dropout (Dropout)	(None, 64, 128)	0	re_lu_2[0][0]
conv1d_3 (Conv1D)	(None, 32, 256)	98,560	dropout[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 32, 256)	1,024	conv1d_3[0][0]
re_lu_3 (ReLU)	(None, 32, 256)	0	batch_normalizat…
conv1d_4 (Conv1D)	(None, 32, 256)	196,864	re_lu_3[0][0]
conv1d_5 (Conv1D)	(None, 32, 256)	33,024	dropout[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 32, 256)	1,024	conv1d_4[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 32, 256)	1,024	conv1d_5[0][0]
add_1 (Add)	(None, 32, 256)	0	batch_normalizat… batch_normalizat…
re_lu_4 (ReLU)	(None, 32, 256)	0	add_1[0][0]
dropout_1 (Dropout)	(None, 32, 256)	0	re_lu_4[0][0]
conv1d_6 (Conv1D)	(None, 16, 512)	393,728	dropout_1[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 16, 512)	2,048	conv1d_6[0][0]
re_lu_5 (ReLU)	(None, 16, 512)	0	batch_normalizat…
conv1d_7 (Conv1D)	(None, 16, 512)	786,944	re_lu_5[0][0]
conv1d_8 (Conv1D)	(None, 16, 512)	131,584	dropout_1[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 16, 512)	2,048	conv1d_7[0][0]
batch_normalizatio…	(None, 16, 512)	2,048	conv1d_8[0][0]

(BatchNormalizatio…			
add_2 (Add)	(None, 16, 512)	0	batch_normalizat… batch_normalizat…
re_lu_6 (ReLU)	(None, 16, 512)	0	add_2[0][0]
conv1d_9 (Conv1D)	(None, 16, 1)	513	re_lu_6[0][0]
multiply (Multiply)	(None, 16, 512)	0	re_lu_6[0][0], conv1d_9[0][0]
global_average_poo… (GlobalAveragePool…	(None, 512)	0	multiply[0][0]
dense (Dense)	(None, 256)	131,328	global_average_p…
dropout_2 (Dropout)	(None, 256)	0	dense[0][0]
dense_1 (Dense)	(None, 1)	257	dropout_2[0][0]

CHAPTER 5

RESULTS AND ANALSIS

5.1 INTERMEDIATE RESULT OF PADDED VALUES

The audio features are padded to achieve the same shape to give as input to the model. For that, the highest length of feature set is taken and the remaining audio samples are padded with zeros to achieve same shape for all the audio samples.

```
Padded values (first 2 timesteps):
 [[[1.         1.        ]
  [0.9304687  0.99033844]
  [0.23190543 0.80410755]
  ...
  [0.         0.        ]
  [0.         0.        ]
  [0.         0.        ]]

 [[0.4215091  0.8899913 ]
  [0.633569   0.94188404]
  [0.31962806 0.8547609 ]
  ...
  [0.         0.        ]
  [0.         0.        ]
  [0.         0.        ]]]
```

5.2 PERFORMANCE METRICS OF THE TEST DATASET

Accuracy is a metric that measures how often a machine learning model

correctly predicts the output. It is calculated as:

$$\text{Accuracy} = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ Predictions} * 100$$

It tells you the percentage of times the model got the answer right. Our model has got an accuracy rate of 92%.

```
Test Accuracy: 0.9200
```

5.3 CONFUSION MATRIX OF THE TEST DATASET

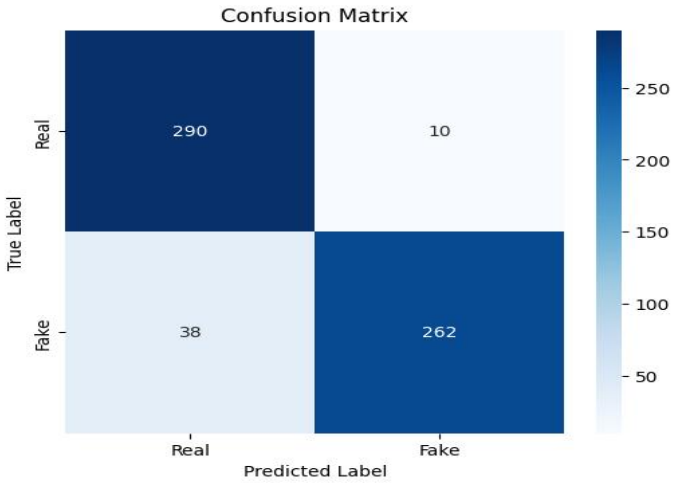
A confusion matrix is a performance evaluation tool for classification models. It shows how well the model's predicted classes match the actual classes by organizing results into four categories:

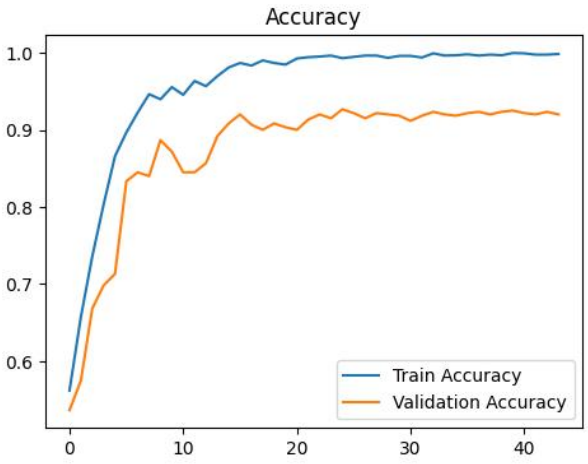
True Positive (TP): Correctly predicted positive class

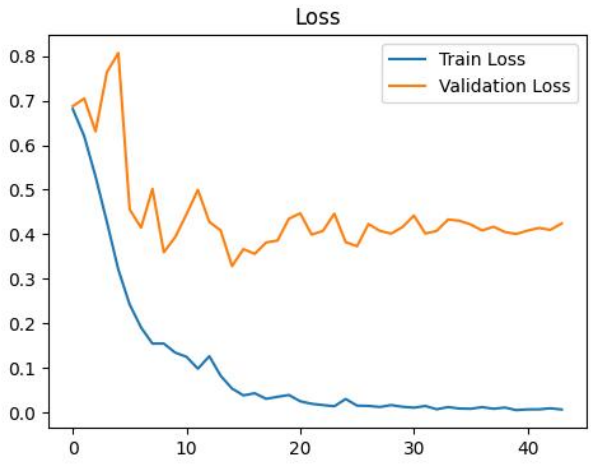
True Negative (TN): Correctly predicted negative class

False Positive (FP): Incorrectly predicted positive class (actually negative)

False Negative (FN): Incorrectly predicted negative class (actually positive)

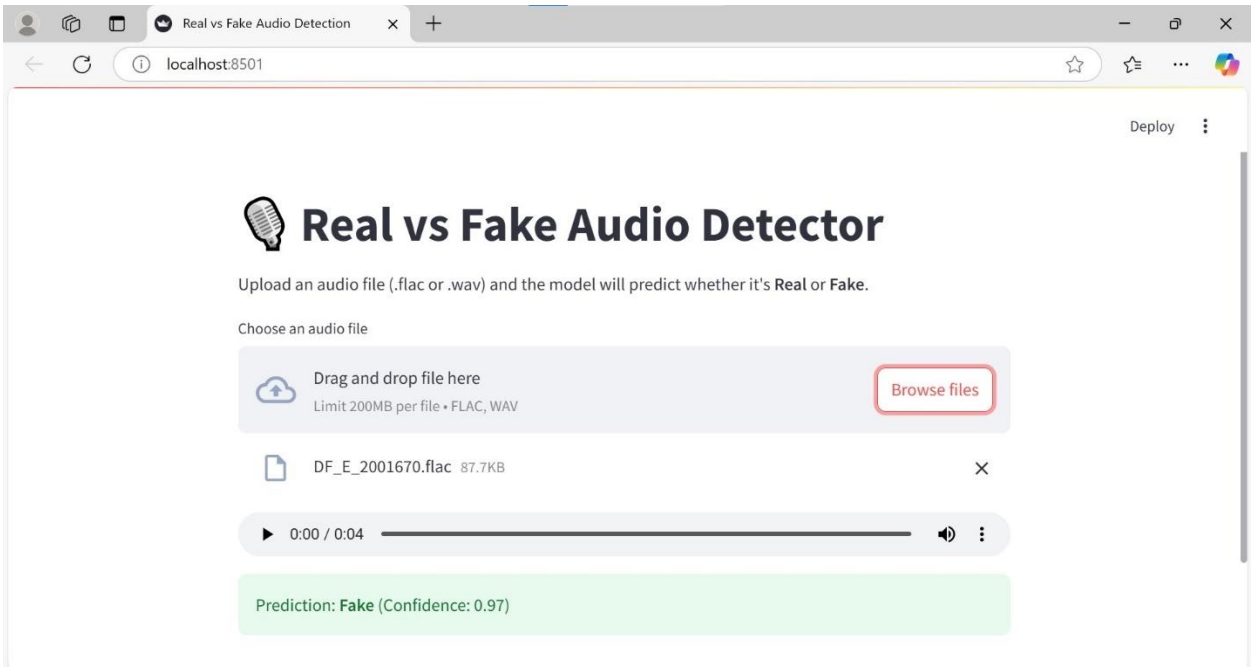


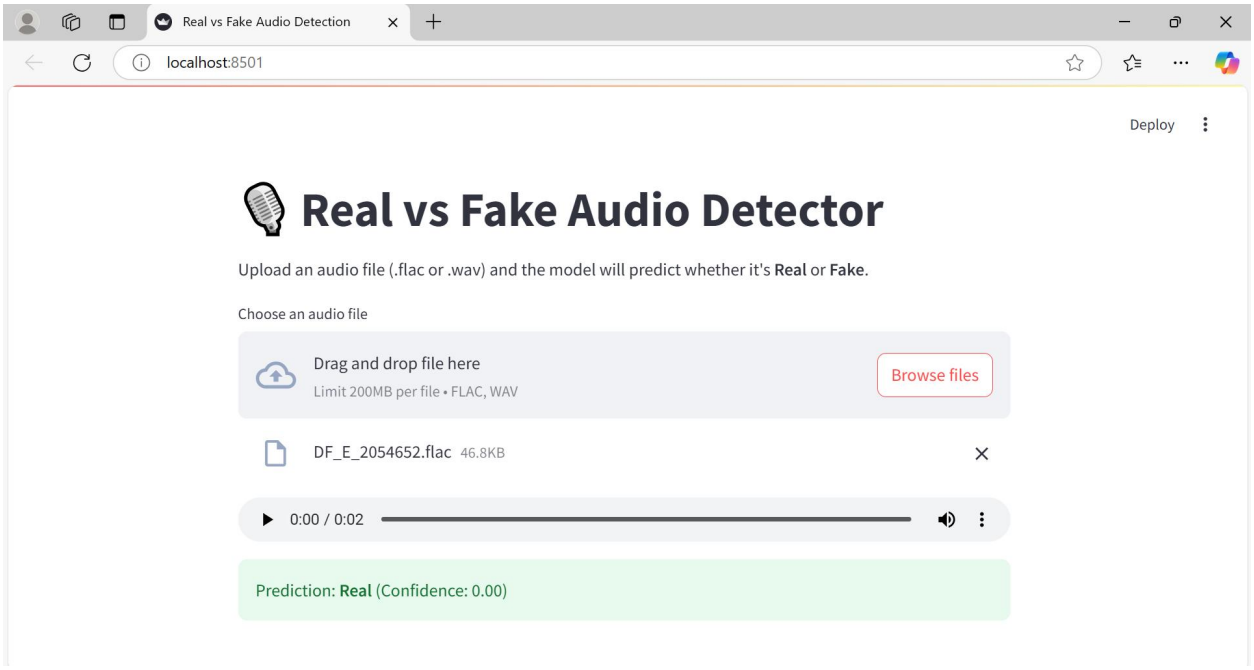




5.4 CLASSIFICATION OF THE EXTERNAL INPUT AUDIO GIVEN

After training the model, we tested the model with some external audio input samples. Their results are given below:





CHAPTER 6

CONCLUSION AND FUTURE WORKS

In conclusion, the proposed deep learning model demonstrates strong potential in detecting deepfake voices with high accuracy and reliability. By leveraging multi-channel audio features such as mel-spectrograms and log-power spectra, the model effectively captures both temporal and spectral patterns of audio signals. The integration of convolutional layers, residual blocks, and attention mechanisms enhances the model's ability to focus on key distinguishing features between real and fake voices. Data preprocessing techniques like noise reduction, silence trimming, and feature normalization contribute significantly to the model's robustness.

The training process benefits from proper balancing and regularization techniques to avoid overfitting. Through comprehensive evaluation metrics, including accuracy, AUC, and the confusion matrix, the model's performance has been validated. The architecture proves efficient for real-world applications, especially in scenarios involving security and authentication. This work highlights the growing need for reliable deepfake detection systems. Future work may explore real-time detection and broader feature sets. Overall, the model lays a strong foundation for combating voice-based misinformation.

FUTURE WORK

- For future work, the model can be extended to support real-time deepfake voice detection for live audio streams and communication platforms.

- Incorporating more diverse datasets, including various languages, accents,

and audio qualities, will help improve generalization across different environments.

- Additionally, integrating advanced audio representations such as MFCCs, pitch contours, and phase information could further enhance detection accuracy.

CHAPTER 7

REFERENCES

[1] B. Wang, Y. Tang, F. Wei, Z. Ba and K. Ren, "FTDKD: Frequency-Time Domain Knowledge Distillation for Low-Quality Compressed Audio Deepfake Detection," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 32, pp. 4905-4918, 2024, doi: 10.1109/TASLP.2024.3492796.

[2] M. Usama Tanveer Gujjar, K. Munir, M. Amjad, A. U. Rehman and A. Bermak, "Unmasking the Fake: Machine Learning Approach for Deepfake Voice Detection," in IEEE Access, vol. 12, pp. 197442-197453, 2024, doi: 10.1109/ACCESS.2024.3521026. Z. Xu, K. Wei, X. Yang and C. Deng, “Point-Supervised Video Temporal Grounding,” in IEEE Transactions on Multimedia, vol. 25, pp. 6121-6131, 2023, doi: 10.1109/TMM.2022.3205404.

[3] G. Ali, J. Rashid, M. Rameez Ul Hussnain, M. Usman Tariq, A. Ghani and D. Kwak, "Beyond the Illusion: Ensemble Learning for Effective Voice Deepfake Detection," in IEEE Access, vol. 12, pp. 149940-149959, 2024, doi: 10.1109/ACCESS.2024.3457866.

[4] D. Song, N. Lee, J. Kim and E. Choi, "Anomaly Detection of Deepfake Audio Based on Real Audio Using Generative Adversarial Network Model," in IEEE Access, vol. 12, pp. 184311-184326, 2024, doi: 10.1109/ACCESS.2024.3506973.

[5] R. K. Bhukya, A. Raj and D. N. Raja, "Audio Deepfakes: Feature Extraction and Model Evaluation for Detection," 2024 5th International Conference for Emerging Technology (INCET), Belgaum, India, 2024, pp. 1-6, doi: 10.1109/INCET61516.2024.10593405.

[6] K. Zaman, I. J. A. M. Samiul, M. Sah, C. Direkoglu, S. Okada and M. Unoki, "Hybrid Transformer Architectures With Diverse Audio Features for Deepfake Speech Classification," in IEEE Access, vol. 12, pp. 149221-149237, 2024, doi: 10.1109/ACCESS.2024.3478731.

[7] J. Xue, C. Fan, J. Yi, J. Zhou and Z. Lv, "Dynamic Ensemble Teacher-Student Distillation Framework for Light-Weight Fake Audio Detection," in IEEE Signal Processing Letters, vol. 31, pp. 2305-2309, 2024, doi: 10.1109/LSP.2024.3431936.

[8] D. K, G. Chhabra, Pallavi, S. A. Tiwaskar, S. Hemelatha and V. Saraswat, "Application of Machine Learning for the Detection of Audio Deep Fake," 2024 Global Conference on Communications and Information Technologies (GCCIT), BANGALORE, India, 2024, pp. 1-6, doi: 10.1109/GCCIT63234.2024.10862652.

[9] J. M. Lapates, B. D. Gerardo and R. P. Medina, "Performance Evaluation of Enhanced DCGAN s for Detecting Deepfake Audio Across Selected FoR Datasets," 2024 15th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, Republic of, 2024, pp. 54-59, doi: 10.1109/ICTC62082.2024.10827547.

[10] E. Karamatlı, A. T. Cemgil and S. Kırbız, "Audio Source Separation Using Variational Autoencoders and Weak Class Supervision," in IEEE Signal Processing Letters, vol. 26, no. 9, pp. 1349-1353, Sept. 2019, doi: 10.1109/LSP.2019.2929440.

[11] Y. Wang, B. Dai, G. Hua, J. Aston and D. Wipf, "Recurrent Variational Autoencoders for Learning Nonlinear Generative Models in the Presence of

Outliers," in IEEE Journal of Selected Topics in Signal Processing, vol. 12, no. 6, pp. 1615-1627, Dec. 2018, doi: 10.1109/JSTSP.2018.2876995.

[12] X. Liu et al., "ASVspoof 2021: Towards Spoofed and Deepfake Speech Detection in the Wild," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 31, pp. 2507-2522, 2023, doi: 10.1109/TASLP.2023.3285283.