INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1. INTRODUCTION

Development of VCD's, have boosted the realization of smart homes, voice-controlled authentication systems etc.,.These VCDs are vulnerable to different spoofing attacks. Audio Authentication is becoming very essential part of our lives. Audio Authentication spoofing is becoming an issue. High-quality audio recorders enable bypassing this audio authentication system by just recording the human voice and reusing them for accessing the same system. Thus, there exists a need to develop a voice anti-spoofing framework capable of detecting multiple audio spoofing attacks.

1.2. OBJECTIVE

1. To develop a Deep-Learning Enabled Audio Spoof Detector which uses Recurrent Neural Networks(RNN) and Long Short Term Memory(LSTM) to classify human voice from recorded voices and integrating the system with an Iot system.

2. The proposed model will defend from the following attacks:
   a) Replay attack
   b) Voice conversion attack

1.3 PROBLEM STATEMENT

The increase in the advancement of audio editing software's , provide an easy way to the accessibility of voice controlled authentication systems which makes the Voice controlled devices(VCD) vulnerable for audio spoof attacks.

Audio spoof attack is the manipulation of genuine signal through recording or modifying to trick an audio verification system.

1.4 GOOGLE TREND ANALYSIS

The diagram shown below displays the Google trend analysis about the domain of the project.

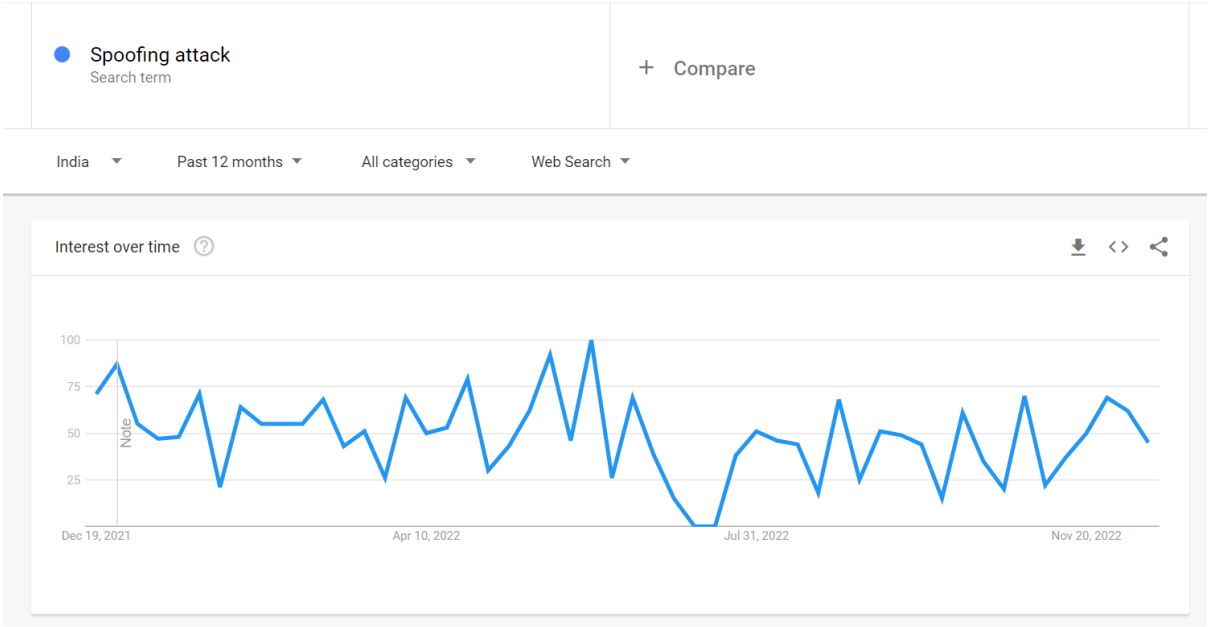


Figure 1.1 Google Trend analyses of spoof attacks over the time.

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

2.1 Fusion of Belts and GMM-UBM Systems for Audio Spoofing Detection (2019)

Authors: Ivan Rakhmanenko , Alexander Shelupanov , Evgeny Kostyunchenko

Description

In this study, Bidirectional Long Short Term Memory (BiLSTM) networks with constant Q cepstral coefficients (CQCC) are used to classify real audio from fake audio in anti-spoofing systems.

By fusing the BiLSTM and GMM-UBM systems, a fusion mechanism is used to increase the variability of the systems' decision-making processes and their accuracy.

Over the baseline systems, these proposed systems significantly improved performance.

Advantages

- Uses time domain dependency relation in audio and gives better results.
- Uses fusion mechanism to improve accuracy.

Disadvantages

- It doesn’t provide classification model for classification of various attacks.

2.2 IoT based speech recognition system(2022)

Authors: Kishor Kumar Sethy, L. M. Varalakshmi, Rajkumar E

Description:

In this paper, speech recognition based IoT system is carried out.For IoT system Raspberry pi board is used and for wifi communication ESP-32 module is used. For speech recognition Support vector machine model is used for training the speech recognition model.

Advantages:

- ESP-32 - Wifi mouse is efficient for data transmission.
- Raspberry pi - Iot system is powerful.

Disadvantages:

- Does not prove to be much effective in classification of audio spoofed data.

2.3 Comparison of VQ and GMM for Text Independent Speaker Identification System for The Bengali Language

Authors: Md Mahadi Hasan Nahid , Md Ashraful , Islam , Md Saiful Islam

Description:

Speaker identification (SI) is the system to identify the person by the signal pattern of their voices. With many speaker identification models have been proposed, but till now speaker identification technology do not reach their full potential. This paper presents a comprehensive comparative study of VQ and GMM to identify the speaker who speaks in Bengali accent. We consider the problem of text-independent speaker identification. We compare the performance/accuracy of VQ and GMM based Speaker Identification System (SIS). They've used Mel Frequency Cepstral Coefficients (MFCC) and Liner Predictive Coding Coefficients (LPCC) for feature extraction.

These extracted features are then sent to the Convolutional Neural Network as input and then are classified as either synthetic or replay attacks.

Advantages:

- The method GMM+LPCC and VQ+LPCC are very fast.
- The method (GMM+LPCC) gives tremendous improvement over the method (GMM+MFCC), and it can detect the correct speaker from much shorter speech samples.

Disadvantages:

- Method like VQ+MFCC is highly accurate but slow and it can be applied in security purpose where the number of users is limited.

2.4 Detection of Various Speech Forgery Operations Based on Recurrent Neural Network (2020)

Author: Diqun Yan and Tingting Wu.

Description:

In this paper, feature extraction methods like LFCC and MFCC are used to extract audio features and then these are sent as input to the Recurrent Neural Network (RNN) frame with two-layer LSTM to detect four common audio forgery operations.

These are experimented mainly on TIMIT and UME databases and various evaluations like intra-database evaluation as well as cross-database evaluation are done and the detection accuracies of each of the above are identified.

Advantages:

- Feature extraction techniques like MFCC and LFCC are used to extract audio features.
- In this work, RNN is used as it can capture the correlation between the frames in a speech recognition application.
- Hence it is considered better than CNN as it does not capture the sequential correlation well.

Disadvantages:

- The cross-database evaluation accuracy could be improved in this model.

2.5 Fake Audio Speech Detection (2020)

Author: Shilpa Lunagaria, Mr. Chandresh Parekh

Description:

In this paper, deep fake audio forgery is identified using Deep Learning algorithms. Audio files are taken as input and model is trained to uniquely identify features for voice creation and voice detection. The model could then classify between whether the audio is real or fake.

The accuracy obtained for this model during training and validation phases are pretty high but the testing accuracy could be improved more by extracting more features and using different algorithms.

Advantages:

- The real and fake voices can be identified.
- The accuracies obtained for training, validation are considerably high (99%, 95% respectively).

Disadvantages:

- This work only focuses on deep fake audio forgery and it doesn't detect or identify other audio forgery operations.

- The testing accuracy in this model could be improved with better algorithms (just 85% accuracy).

2.6 Voice-Based Human Identification using Machine Learning(2022)

Author: Authors: Ivan Rakhmanenko , Alexander Shelupanov , Evgeny Kostyunchenko

Description:

In this study, Simple machine learning algorithms are used for voice Identification of 150 speakers. Dataset contains around 3000 samples. Mel Frequency cepstral coefficients(MFCC) is used for extracting features from audio samples. Machine learning algorithms like Support vector machine, Random forest algorithm are used for training the system.

Advantages:

- As simple machine learning algorithms are used, hence it is power efficient.

Disadvantages:

- Simple machine learning algorithms are used so accuracy is low.

2.7 Voice spoofing countermeasure for voice replay attacks using deep learning. (2022)

Author: Jincheng Zhou, Tao Hai1, Dayang N. A. Jawawi, Dan Wang, Ebuka Ibeke and Cresantus Biamba

Description:

The paper discusses about the immense usage of Automatic Speaker Verification (ASV) system which verifies users with their voices and it's susceptibility to voice spoofing attacks - logical and physical access attacks.
A secured voice spoofing countermeasure to detect voice replay attacks is proposed. This has enhanced the ASV system security by building a spoofing countermeasure dependent on the decomposed signals that consist of prominent information. It uses two main features— the Gammatone Cepstral Coefficients (GCC) and Mel-Frequency Cepstral Coefficients (MFCC) — for the audio representation. For the classification of the features, Bi-directional Long-Short Term Memory Network in the cloud, a deep learning classifier.

Numerous audio features and respective feature's capability to obtain the most vital details from the audio for it to be labelled genuine or a spoof speech is examined. Furthermore, it uses various machine learning algorithms to illustrate the superiority of the system compared to the traditional classifiers. The results of the experiments were classified according to the parameters of accuracy, precision rate, recall, F1-score, and Equal Error Rate (EER). The results were 97%, 100%, 90.19% and 94.84%, and 2.95%, respectively.

Advantages:

- Avoids replay attacks in ASV.
- Voice biometrics.
- More accurate with the method of Speech Decomposition.

Disadvantages:

- Does not discuss about many audio spoof attacks, focuses on Replay attacks only.

2.8 LSTM and CNN based ensemble approach for spoof detection task in automatic speaker verification systems (2022)

Author: Mohit Dua, Chhavi Jain, Sushil Kumar

Description:

In this paper , the system that is proposed tries to address the problem of classifying legitimate user and the malicious attacks using deep learning (DL) methods and ensemble of different neural networks. The first model that is discussed is a combination of time-distributed dense layers and long short-term memory (LSTM) layers.

The other two deep neural networks (DNNs) are based on temporal convolution (TC) and spatial convolution (SC). Finally, an ensemble model comprising of these three DNNs has also been analysed. All these models are analysed with Mel frequency cepstral coefficients (MFCC), inverse Mel frequency cepstral coefficients (IMFCC) and constant Q cepstral coefficients (CQCC) at the frontend, where the proposed ensemble performs best with CQCC features.

The proposed work uses ASVspoof 2015 and ASVspoof 2019 datasets for training and testing, with the evaluation set having speech synthesis (SS) and voice conversion (VC) attacked utterances. Performance of proposed system trained with ASVspoof 2015 dataset degrades with evaluation set of ASVspoof 2019 dataset, whereas performance of the same system improves when training is also done with the ASVspoof 2019 dataset.

Advantages:

- LSTM models are used which increases performance.
- Ensemble method is used to get a consolidated decision from models.

Disadvantages:

- Performance on combined dataset is low.
- Deep CNN can be used to improve performance.

SYSTEM SPECIFICATION

CHAPTER 3

SYSTEM SPECIFICATION

3.1. SOFTWARE SPECIFICATION

Operating system: Windows 10

IDE: Google Colaboratory / Jupyter Notebook

Coding Language: Python

3.2. HARDWARE SPECIFICATION

Processor: Intel core i5

Hard disk: 512GB

RAM: 8GB

SYSTEM ANALYSIS

CHAPTER 4

SYSTEM ANALYSIS

4.1. INTRODUCTION

Development of Voice Control Devices(VCDs), have boosted the realization of smart homes, voice-controlled authentication systems etc.,.These VCDs are vulnerable to different spoofing attacks. Audio Authentication is becoming very essential part of our lives. High-quality audio recorders enables bypassing this audio authentication system by just recording the human voice and reusing them for accessing the same system. Thus, there exists a need to develop a voice anti-spoofing framework capable of detecting multiple audio spoofing attacks.

4.2. PROPOSED SYSTEM

An Iot system which acts as an home automation system is proposed which uses a Bidirectional LSTM model to classify the audio samples and GMM model to identify the specific voice of the speaker. If the given audio input is found to be an spoofed audio or an invalid user , then the system denies the access to the system. Features fed into both the bidirectional LSTM model as well as GMM model are generated using MFCC methodologies.

The proposed system performs binary classification of audio data which are mapped to two classes

a) Authentic/Bonafide
b) Spoofed

And identifies the speaker voice using the GMM model and performs the necessary action in the Home automation system.

4.3. DATASET USED.

1. Audio Spoof Detection Model

The Dataset used in the model is Automatic Speaker Verification(ASV) 2019.

ASV Spoof dataset contains two types of Audio files

- Physical Access - Bonafide utterances are made in a real, physical space in which spoofing attacks are captured and then replayed within the same physical space using replay devices of varying quality.
- Logical Access - Bonafide and spoofed utterances generated using text-to-speech (TTS) and voice conversion (VC) algorithms are communicated across telephony and VoIP networks with various coding and transmission effects

The dataset includes genuine and spoofed speech from 20 speakers (8 male, 12 female).Each spoofed utterance is generated according to one of 2 voice conversion and 3 speech synthesis algorithms.

The voice conversion systems include those based on (i) neural-network-based and (ii)transfer-function-based methods.

The speech synthesis systems were implemented with (i) waveform concatenation, (ii) neural-network-based parametric speech synthesis using source-filter vocoders and (iii) neural-network-based parametric speech synthesis using Wavenet.

2. Speaker Identification Model

The model receives 5 audio samples from the user for training and creates an GMM model for that specific speaker. When an audio file is given as input it compares the scores of all the GMM model and finds the model that produces the highest score value. The speaker is then identified with the label for whom the it is made.

4.4 METHODOLGIES USED.

Mel Frequency Cepstral Coefficients(MFCCs) :

In general for any audio based task the raw audio signal cannot be given to the model as input because there will be a lot of noise in the audio signal. It is observed that extracting features from the audio signal and using it as input to the base model will produce much better performance than directly considering raw audio signal as input. MFCC is the widely used technique for extracting the features from the audio signal.

Mel-frequency cepstral coefficients (MFCC) which have 39 features. The feature count is small enough to force us to learn the information of the audio. 12 parameters are related to the amplitude of frequencies. It provides us enough frequency channels to analyze the audio.

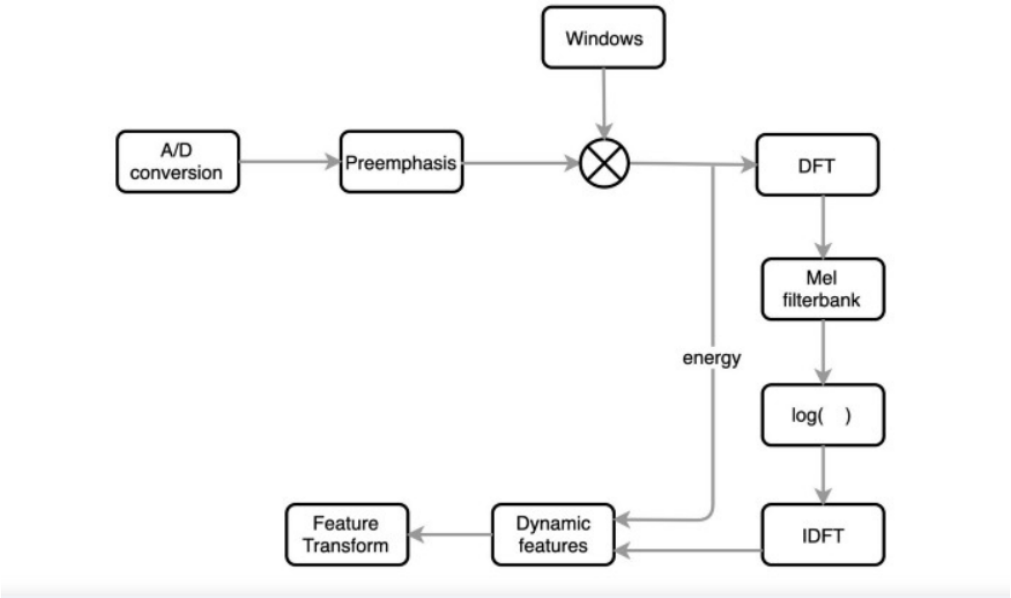


Figure 4.1 - MFCC feature extraction flow chart

Bi-directional Long Short Term Memory Network:

Bidirectional LSTM (BiLSTM) is a recurrent neural network used primarily on natural language processing. Unlike standard LSTM, the input flows in both directions, and it's capable of utilizing information from both sides. It's also a powerful tool for modeling the sequential dependencies between words and phrases in both directions of the sequence.

BiLSTM adds one more LSTM layer, which reverses the direction of information flow. Briefly, it means that the input sequence flows backward in the additional LSTM layer. Then the outputs from both LSTM layers are combined in several ways, such as average, sum, multiplication, or concatenation.

To illustrate, the unrolled BiLSTM is presented in the figure below:

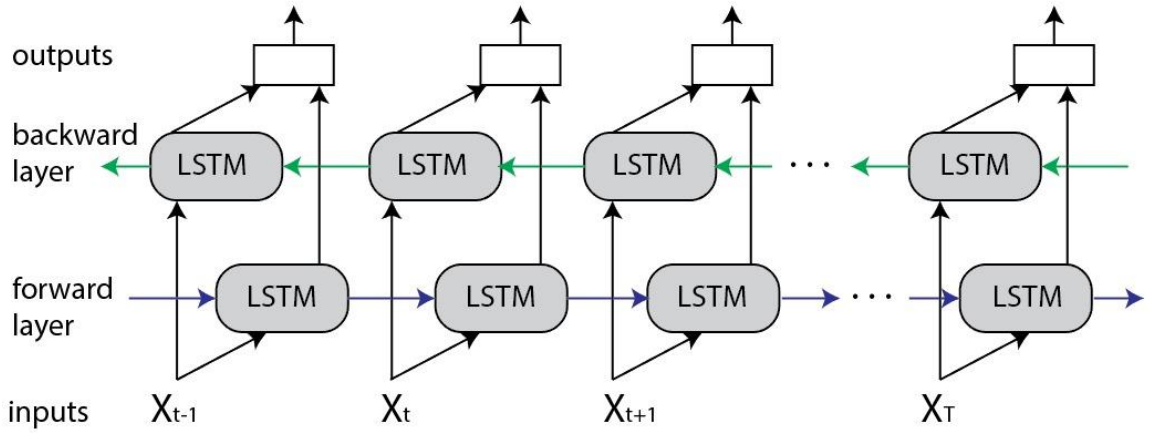


Figure 4.2 - Bi-directional LSTM architecture

Advantages of Bidirectional LSTM

This type of architecture has many advantages in real-world problems, especially in NLP. The main reason is that every component of an input sequence has information from both the past and present. For this reason, BiLSTM can produce a more meaningful output, combining LSTM layers from both directions.

BiLSTM will have a different output for every component (word) of the sequence (sentence). As a result, the BiLSTM model is beneficial in some NLP tasks, such as sentence classification, translation, and entity recognition. In addition, it finds its applications in speech recognition, protein structure prediction, handwritten recognition, and similar fields.

GMM:

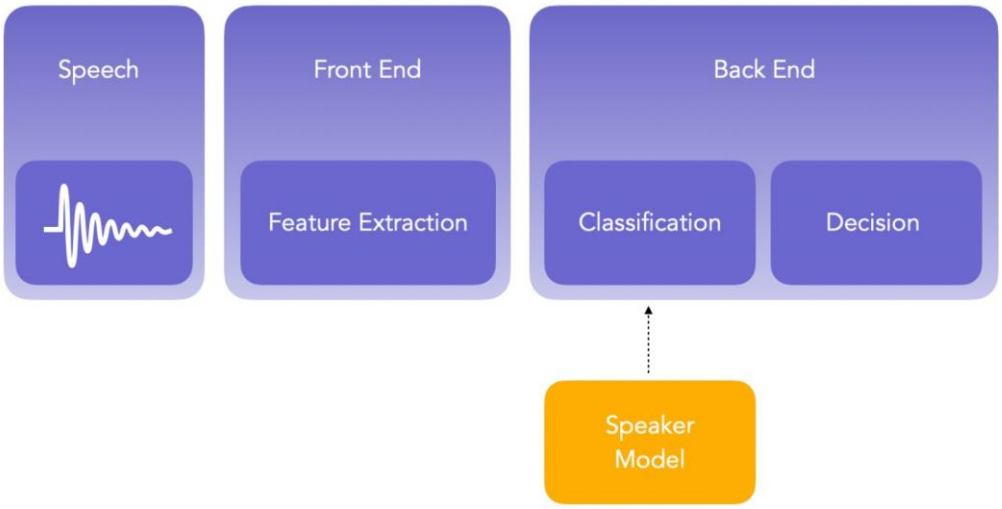


Figure 4.3 – GMM Architecture

GMM is a generative speaker model because GMM model typically involves capturing data from targeted speaker. It is unique model which has de-facto reference method and generally used for robust SRS using short-speech statement. It has better advantage than other models because the training is quite fast and can be scaled and update the system to add new speakers with relative ease. A GMM model is composed of limited mixture of multivariate Gaussian components, it is a collection of several spectral features that are valid for deigning a speaker model for a targeted speaker. Suppose a speaker has 2 or 3 utterance, and from each utterance, we extract D-dimensional features .The MFCC features of each speaker are represented by Gaussian Mixture Model. MFCC coefficients are used for extracting features and minimum processing time in GMM is 10ms for speech utterance. The parameters for GMM model is mean vectors, densities (is a sum of M numbers component density), and covariance matrices.

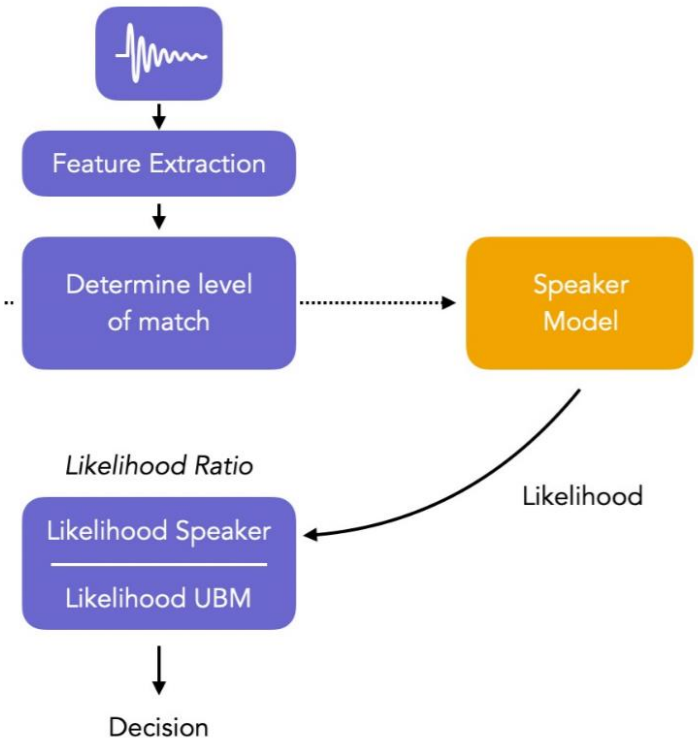


Figure 4.4 – Model workflow

SYSTEM IMPLEMENTATION

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1. INTRODUCTION

Audio Authentication is becoming very essential part of our lives. Audio Authentication spoofing is becoming an issue. High-quality audio recorders enable bypassing this audio authentication system by just recording the human voice and reusing them for accessing the same system. Thus, there exists a need to develop a voice anti-spoofing framework capable of detecting multiple audio spoofing attacks. As a result we propose a system that identifies the spoofed audio from the legit audio and thereby allowing only legit audio to gain access to any voice-controlled devices or IoT systems.

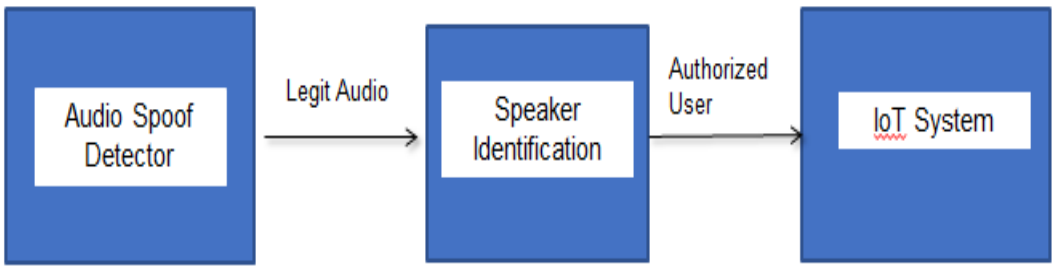


Figure 5.1 - System Architecture

5.2. IMPLEMENTATION

5.2.1. DATASET

ASVspoof 2019[1] organized by Junichi Yamagishi, Massimiliano Todisco, Md Sahidullah, Héctor Delgado, Xin Wang, Nicholas Evans, Tomi Kinnunen, Kong Aik Lee, Ville Vestman, and Andreas Nautsch in 2019. The ASVspoof challenge aims to encourage further progress through (i) the collection and distribution of a standard dataset with varying spoofing attacks implemented with multiple, diverse algorithms and (ii) a series of competitive evaluations for automatic speaker verification.

5.2.2. AUDIO SPOOF DETECTION MODEL

Initially, the system uses the following library for its functionalities as shown in Figure 5.2,

```
import os
import glob
import fnmatch
import pandas as pd
import numpy as np
import librosa #To deal with Audio files
import librosa.display
import matplotlib.pyplot as plt
import IPython.display as ipd
import math
import tensorflow as tf
import random
import sklearn
from matplotlib import pyplot
import matplotlib.pyplot as plt
```

Figure 5.2 - Libraries used

The downloaded ASVspoof dataset contains both Physical Access (PA) files and Logical Access files (LA) as shown in figure 5.3 and figure 5.4. These audio files from the dataset are then imported to the system.

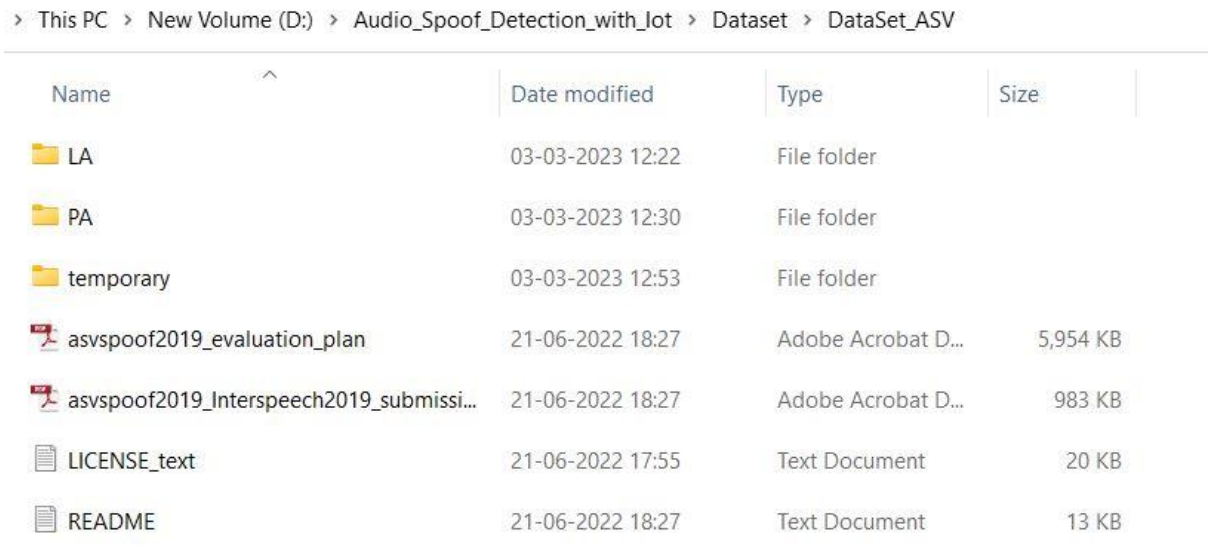


Figure 5.3 - ASVspoof 2019 dataset

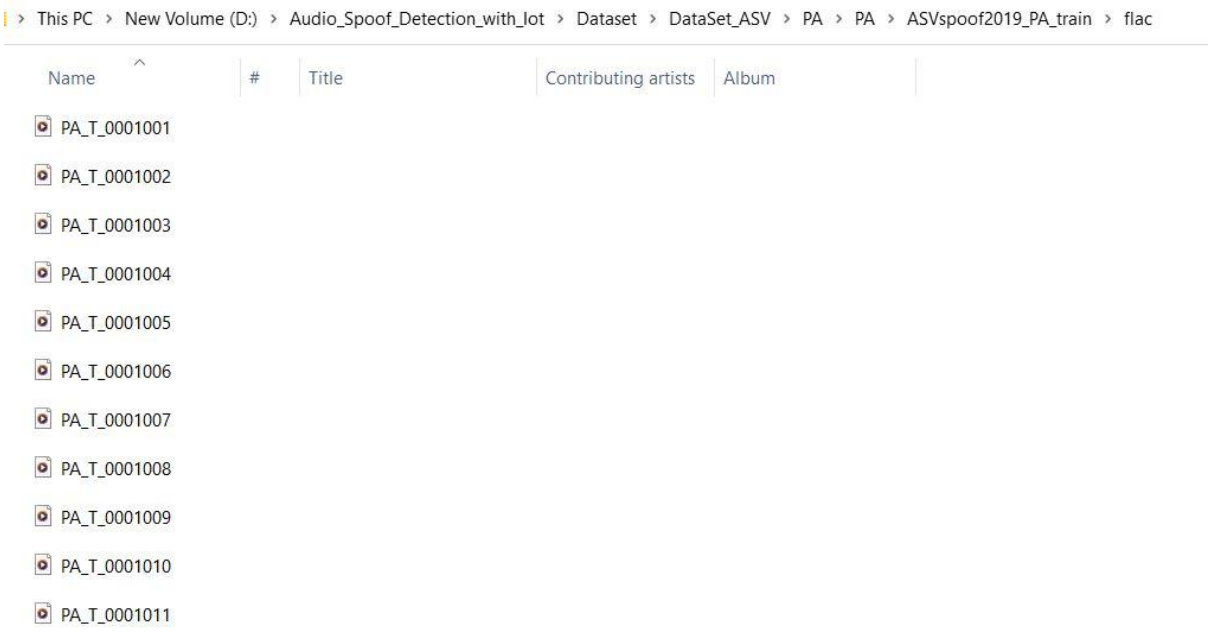


Figure 5.4 - Physical Access Audio files

The system imports and maintains 1000 files of both bonafide and spoofed physical access audio files as shown in figure 5.5.

```
path = "D:\\Project\\Audio_Spoof_Detector\\Dataset"
print(os.listdir(path))
```

```
['bonafide', 'spoof']
```

```
bonafide_data = path  + "\\bonafide\\"
spoof_data = path + '\\spoof\\'
```

```
bonafide_paths=[]
spoof_paths=[]
for dir, _, filenames in os.walk('D:\\Main Project\\lstmdata\\train\\bonafide'):
    for filename in filenames:
        bonafide_paths+=[(os.path.join(dir, filename))]
for dir, _, filenames in os.walk('D:\\Main Project\\lstmdata\\train\\spoof'):
    for filename in filenames:
        spoof_paths+=[(os.path.join(dir, filename))]
print(len(bonafide_paths),len(spoof_paths))
```

```
0 0
```

Figure 5.5 - Importing the dataset

The system then visualizes a random sample using IPython.display(ipd) module with the GetRandomSample function created as shown in the figure 5.6.

```
def VisualizeRandomSample(sample_sound, sample_rate):
    #to hear the audio sample
    sample_audio = ipd.Audio(sample_sound, rate=sample_rate)
    return sample_audio
```

```
def GetRandomSample(folder):
    RandomSample            = np.random.randint(0,len(os.listdir(folder)))
    SampleSound             = os.listdir(folder)[RandomSample]
    sample_address            = folder + SampleSound
    return sample_address
```

```
sample_sound, sample_rate=librosa.load(GetRandomSample(bonafide_data))
```

```
VisualizeRandomSample(sample_sound, sample_rate)
```

Figure 5.6 - Visualizing Random Sample Audio

The system then uses various definitions as shown in figure 5.7. The various definitions used are namely:

- Normalize:
- Display Wave Form:
- Spectral Centroid:
- Spectral Roll-off:

```
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)
```

```
def DisplayWaveform(x,sr):
    #display waveform
    %matplotlib inline
    plt.figure(figsize=(14, 5))
    librosa.display.waveshow(x, sr=sr)
```

```
def spectral_centroid(x,sr):
    spectral_centroids = librosa.feature.spectral_centroid(x, sr=sr)[0]
    spectral_centroids.shape
    frames = range(len(spectral_centroids))
    t = librosa.frames_to_time(frames)
    librosa.display.waveshow(x, sr=sr, alpha=0.4)
    plt.plot(t, normalize(spectral_centroids), color='r')
    return t
```

```
def spectral_Rolloff(x,sr,t):
    spectral_rolloff = librosa.feature.spectral_rolloff(x, sr=sr)[0]
    librosa.display.waveshow(x, sr=sr, alpha=0.4)
    plt.plot(t, normalize(spectral_rolloff), color='r')
```

Figure 5.7 – Audio Visualization

The system uses the created definition to visualize both bonafide as well as spoofed audio as shown in the figures below.

- DisplayWaveForm audio visualization for bonafide audio is shown in figure 5.8a and spoofed audio is shown in 5.8b.

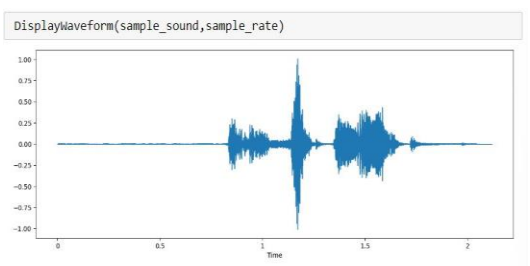


Figure 5.8(a) - Bonafide

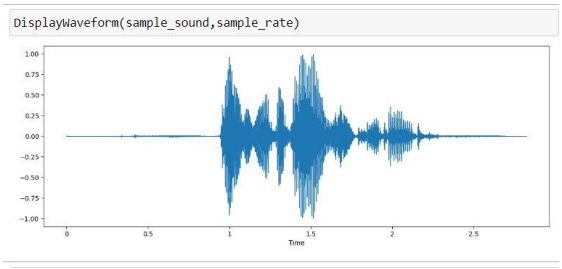


Figure 5.8(b) – Spoof

- Spectral Centroid audio visualization for bonafide audio is shown in figure 5.9a and spoofed audio is shown in 5.9b.

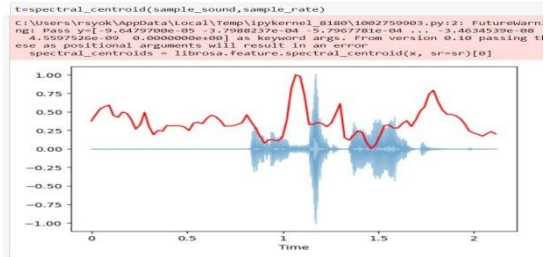


Figure 5.9(a) - Bonafide

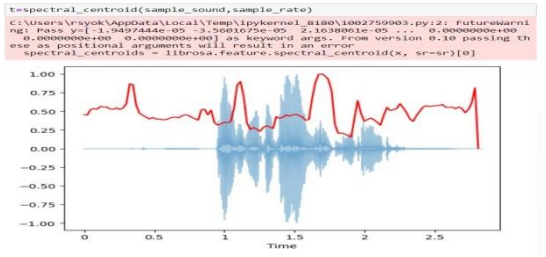


Figure 5.9(b) – Spoof

- Spectral Roll Off audio visualization for bonafide audio is shown in figure 5.10(a) and spoofed audio is shown in 5.10(b).

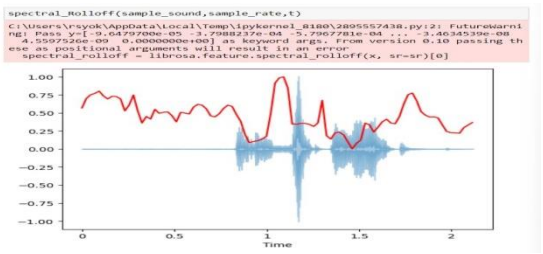


Figure 5.10(a) - Bonafide

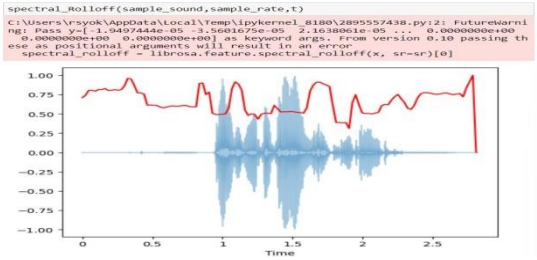


Figure 5.10(b) – Spoof

A dataset consisting of bonafide and spoofed speech samples was utilized to analyze and compare the spectral and cepstral features of both types of speech.

The bonafide and spoofed speech samples were obtained from two separate folders, and were then subjected to various signal processing techniques.

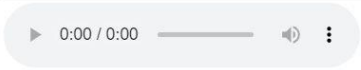
To analyze the spectral features, the Fast Fourier Transform (FFT) was applied to both the bonafide and spoofed speech samples as shown in figure 5.11(a) and 5.11(b). The magnitude of the FFT output was then computed, and the frequency components were plotted using Matplotlib. A Short-Time Fourier Transform (STFT) was also performed on the samples to generate a spectrogram, which represents the frequency content of the signal over time as shown in figure 5.13(a) and 5.13(b). The spectrograms were then plotted using the Librosa library as shown in figure 5.14(a) and 5.14(b).

Furthermore, the Mel Frequency Cepstral Coefficients (MFCCs) were extracted from the bonafide and spoofed speech samples as shown in figure 5.15(a) and 5.15(b). The MFCCs are commonly used in speech analysis because they represent the spectral envelope of a signal, and are thus effective in characterizing the phonetic content of speech. The extracted MFCCs were then plotted using Librosa as shown in figure 5.16(a) and 5.16(b).

The results of the spectral and cepstral analyses were then compared between the bonafide and spoofed speech samples. It was observed that the spectral content of the bonafide speech samples was different from that of the spoofed speech samples, with the latter exhibiting more noise and artifacts. Similarly, the cepstral features of the bonafide and spoofed speech samples differed, with the MFCCs of the bonafide samples being more distinguishable and having a more uniform distribution.

Overall, the findings of this study suggest that spectral and cepstral analyses can be effective in differentiating between bonafide and spoofed speech samples. These techniques may therefore be useful in the development of robust speech authentication systems.

```
RandomNormal= np.random.randint(0,len(os.listdir(bonafide_data)))
NormalSound = os.listdir(bonafide_data)[RandomNormal]
NormalSound_address = bonafide_data+NormalSound
NormalSound_sample,sample_rate = librosa.load(NormalSound_address)
ipd.Audio(NormalSound_sample,rate=sample_rate)
```

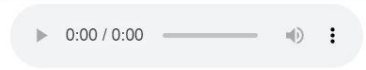



```
fft_normal = np.fft.fft(NormalSound_sample)
magnitude_normal = np.abs(fft_normal)
freq_normal = np.linspace(0,sample_rate, len(magnitude_normal))
semiFrquency = freq_normal[:int(len(freq_normal)/2)]
SemiMagnitude = magnitude_normal[:int(len(freq_normal)/2)]
```

```
plt.figure(figsize=(12,6))
plt.plot(semiFrquency,SemiMagnitude)
plt.title("Spectrum")
plt.xlabel("Frequency")
plt.ylabel("Magnitude")
plt.show()
```

Figure 5.11(a)

```
# Choose random soud from spoof folder
Spoof_random= np.random.randint(0,len(os.listdir(spoof_data)))
SpoofSound = os.listdir(spoof_data)[Spoof_random]
SpoofSound_address = spoof_data+SpoofSound
SpoofSound_sample,sample_rate = librosa.load(SpoofSound_address)
ipd.Audio(SpoofSound_sample,rate=sample_rate)
```




```
fftSPOOF = np.fft.fft(SpoofSound_sample)
magnitudeSpoof = np.abs(fftSPOOF)
SpoofFrequency = np.linspace(0,sample_rate, len(magnitudeSpoof))
semiFrquency = SpoofFrequency[:int(len(SpoofFrequency)/2)]
SemiMagnitude = magnitudeSpoof[:int(len(SpoofFrequency)/2)]
```

```
plt.figure(figsize=(12,6))
plt.plot(semiFrquency,SemiMagnitude)
plt.xlabel("Frequency")
plt.ylabel("Magnitude")
plt.show()
```

Figure 5.11(b)

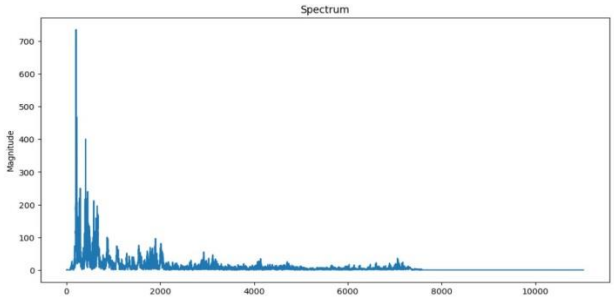


Figure 5.12(a)

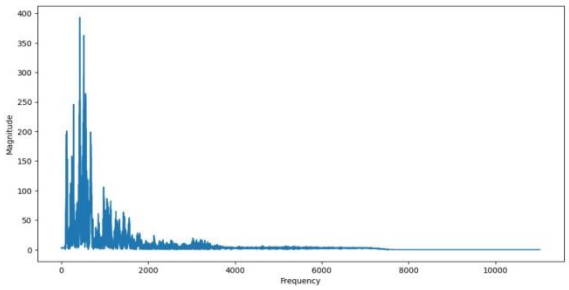


Figure 5.12(b)

```
# STFT -> spectrogram
hoplen = 512 # in num. of samples
FFTnum = 2048 # window in num. of samples
```

```
# calculate duration hop length and window in seconds
hoplen_duration = float(hoplen)/sample_rate
FFTnum_duration = float(FFTnum)/sample_rate
```

```
print("STFT hop length duration is: {}s".format(hoplen_duration))
print("STFT window duration is: {}s".format(FFTnum_duration))
```

```
STFT hop length duration is: 0.023219954648526078s
STFT window duration is: 0.09287981859410431s
```

```
# perform stft
Normal_stft = librosa.stft(NormalSound_sample, n_fft=FFTnum, hop_length=hoplen)
```

```
# calculate abs values on complex numbers to get magnitude
spectrogram = np.abs(Normal_stft)
log_spectrogram = librosa.amplitude_to_db(spectrogram)
```

```
# display spectrogram
plt.figure(figsize=(12,6))
librosa.display.specshow(log_spectrogram, sr=sample_rate, hop_length=hoplen)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar()
plt.set_cmap("plasma")
plt.title("Spectrogram")
```

Figure 5.13(a)

```
# STFT -> spectrogram
hoplen = 512 # in num. of samples
n_fft = 2048 # window in num. of samples
```

```
# calculate duration hop length and window in seconds
hoplen_duration = float(hoplen)/sample_rate
FFTnum_duration = float(FFTnum)/sample_rate
```

```
print("STFT hop length duration is: {}s".format(hoplen_duration))
print("STFT window duration is: {}s".format(FFTnum_duration))
```

```
STFT hop length duration is: 0.023219954648526078s
STFT window duration is: 0.09287981859410431s
```

```
# perform stft
stft_murmur = librosa.stft(SpoofSound_sample, n_fft=FFTnum, hop_length=hoplen)
```

```
# calculate abs values on complex numbers to get magnitude
spectrogram_murmur = np.abs(stft_murmur)
log_spectrogram_murmur = librosa.amplitude_to_db(spectrogram_murmur)
```

```
# display spectrogram
plt.figure(figsize=(12,6))
librosa.display.specshow(log_spectrogram, sr=sample_rate, hop_length=hoplen)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar()
plt.set_cmap("plasma")
plt.title("Spectrogram Spoofed")
```

Figure 5.13(b)

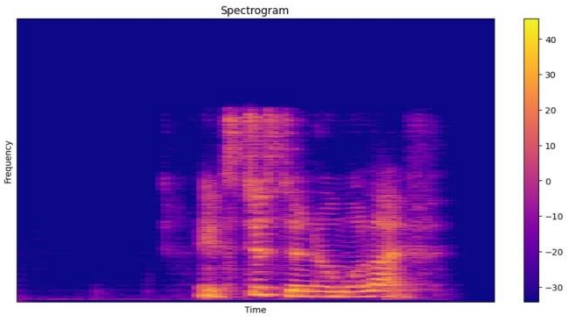


Figure 5.14(a)

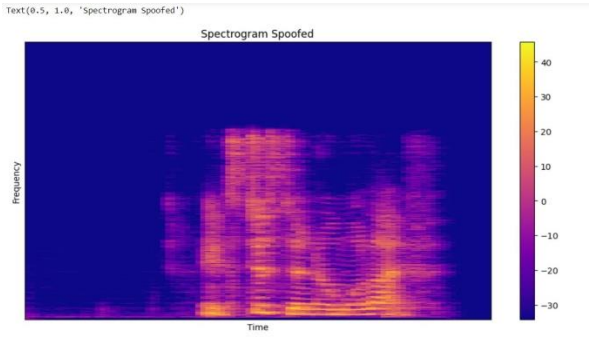


Figure 5.14(b)

```
# MFCCs
# extract 25 MFCCs
MFCCs = librosa.feature.mfcc(NormalSound_sample, sample_rate, n_fft=FFTnum, hop_length=hoplen, n_mfcc=25)
```

```
C:\Users\rsyok\AppData\Local\Temp\ipykernel_8180\4107820172.py:3: FutureWarning: Pass y=[-0.00039334 -0.00340877 -0.0046625
...  0.          0.
  0.        ], sr=22050 as keyword args. From version 0.10 passing these as positional arguments will result in an error
  MFCCs = librosa.feature.mfcc(NormalSound_sample, sample_rate, n_fft=FFTnum, hop_length=hoplen, n_mfcc=25)
```

```
# display MFCCs
plt.figure(figsize=(12,6))
librosa.display.specshow(MFCCs, sr=sample_rate, hop_length=hoplen)
plt.xlabel("Time")
plt.ylabel("MFCC coefficients")
plt.colorbar()
#plt.set_cmap("YlOrBr")
plt.title("MFCCs")
```

```
Text(0.5, 1.0, 'MFCCs')
```

Figure 5.15(a)

```
# MFCCs
# extract 25 MFCCs
MFCCs_murmur = librosa.feature.mfcc(SpoofSound_sample, sample_rate, n_fft=FFTnum, hop_length=hoplen, n_mfcc=25)
```

```
C:\Users\rsyok\AppData\Local\Temp\ipykernel_8180\652639023.py:3: FutureWarning: Pass y=[ 1.9382829e-05 -2.3518824e-03 -3.655784
8e-03 ...  0.0000000e+00
  0.0000000e+00  0.0000000e+00], sr=22050 as keyword args. From version 0.10 passing these as positional arguments will result
in an error
  MFCCs_murmur = librosa.feature.mfcc(SpoofSound_sample, sample_rate, n_fft=FFTnum, hop_length=hoplen, n_mfcc=25)
```

```
# display MFCCs
plt.figure(figsize=(12,6))
librosa.display.specshow(MFCCs, sr=sample_rate, hop_length=hoplen)
plt.xlabel("Time")
plt.ylabel("MFCC coefficients")
plt.colorbar()
plt.set_cmap("plasma")
plt.title("MFCCs")
```

```
Text(0.5, 1.0, 'MFCCs')
```

Figure 5.15(b)

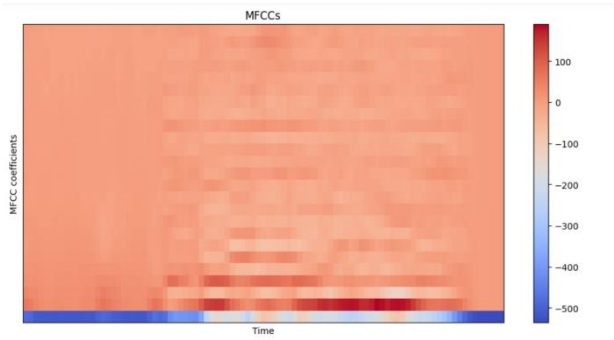


Figure 5.16(a)

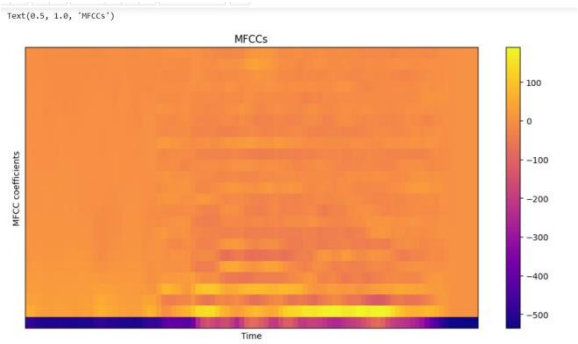


Figure 5.16(b)

The function `load_file_data` loads audio files from a specified folder using the Librosa library as shown in figure 5.17. It takes in `folder` - the path of the directory containing the audio files, `file_names` - a list of filenames to be loaded, `duration` - the duration of the audio file to be loaded (default is 10 seconds), and `sr` - the sampling rate of the audio file (default is 22050 Hz).

The function loops through each file in `file_names`, and loads the audio file using `librosa.load()`. It then checks if the duration of the loaded audio file is

less than the specified `duration` and pads the file to make it the same duration if necessary. Next, it extracts the MFCC (Mel-frequency cepstral coefficients) feature from the audio data using `librosa.feature.mfcc()`, which is a commonly used feature in speech and audio analysis. Finally, the function appends the extracted feature to a list `data` and returns it.

In case of any errors encountered while parsing a file, the function prints an error message. Overall, `load_file_data` provides a useful tool for loading and processing audio data for various applications.

```
def load_file_data (folder, file_names, duration=10, sr=22050):
    Inputlength=sr*duration
    data = []
    for file_name in file_names:
        try:
            sound_file=folder+file_name
            print ("load file ",sound_file)
            X, sr = librosa.load( sound_file, sr=sr, duration=duration)
            dur = librosa.get_duration(y=X, sr=sr)
            # pad audio file same duration
            if (round(dur) < duration):
                print ("fixing audio lenght :", file_name)
                y = librosa.util.fix_length(X, Inputlength)
            # extract normalized mfcc feature from data
            mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sr, n_mfcc=25).T,axis=0)
        except Exception as e:
            print("Error encountered while parsing file: ", file)
        feature = np.array(mfccs).reshape([-1,1])
        data.append(feature)
    return data
```

Figure 5.17

The code loads audio files from two folders, 'bonafide_data' and 'spoof_data', with a maximum duration of 10 seconds and a sampling rate of 22050. It then assigns integer labels to the two classes, 'bonafide' and 'spoof', and loads the audio data and labels into two separate arrays as shown in figures 5.18, 5.19 and 5.20.

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
```

```
# Map label text to integer
Class = ['bonafide','spoof']
Class_len=len(Class)
```

```
# Map integer value to text labels
label_to_int = {k:v for v,k in enumerate(Class)}
print (label_to_int)
int_to_label = {v:k for k,v in label_to_int.items()}
print(int_to_label)
```

```
{'bonafide': 0, 'spoof': 1}
{0: 'bonafide', 1: 'spoof'}
```

```
SAMPLE_RATE = 22050
# seconds
SoundClipMaxDuration=10
```

Figure 5.18

```
In [51]: NormalSounds = load_file_data(folder=bonafide_data,file_names=os.listdir(bonafide_data), duration=SoundClipMaxDuration)
         normal_labels = [0 for items in NormalSounds]
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000016.flac
fixing audio lenght : PA_T_0000016.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000017.flac
fixing audio lenght : PA_T_0000017.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000018.flac
fixing audio lenght : PA_T_0000018.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000019.flac
fixing audio lenght : PA_T_0000019.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000020.flac
fixing audio lenght : PA_T_0000020.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000021.flac
fixing audio lenght : PA_T_0000021.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000022.flac
fixing audio lenght : PA_T_0000022.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000023.flac
fixing audio lenght : PA_T_0000023.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000024.flac
fixing audio lenght : PA_T_0000024.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\bonafide\PA_T_0000025.flac
fixing audio lenght : PA_T_0000025.flac
```

Figure 5.19

```
SpoofSounds = load_file_data(folder=spoof_data,file_names=os.listdir(spoof_data), duration=SoundClipMaxDuration)
SpoofLabels = [1 for items in SpoofSounds]

print ("Loading Done")
```

```
load file  D:\Project\Audio_Spoof_Detector\Dataset\spoof\PA_T_0005402.flac
fixing audio lenght : PA_T_0005402.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\spoof\PA_T_0005403.flac
C:\Users\rsyok\AppData\Local\Temp\ipykernel_8180\3230612674.py:13: FutureWarning: Pass size=220500 as keyword args. Fro
ion 0.10 passing these as positional arguments will result in an error
  y = librosa.util.fix_length(X, Inputlength)
fixing audio lenght : PA_T_0005403.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\spoof\PA_T_0005404.flac
fixing audio lenght : PA_T_0005404.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\spoof\PA_T_0005405.flac
fixing audio lenght : PA_T_0005405.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\spoof\PA_T_0005406.flac
fixing audio lenght : PA_T_0005406.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\spoof\PA_T_0005407.flac
fixing audio lenght : PA_T_0005407.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\spoof\PA_T_0005408.flac
fixing audio lenght : PA_T_0005408.flac
load file  D:\Project\Audio_Spoof_Detector\Dataset\spoof\PA_T_0005409.flac
```

Figure 5.20

The code then concatenates the data and label lists for both classes into two new lists, X and Y. It then shuffles the data randomly to ensure that the training, validation, and testing data are representative of the entire dataset.

The code then splits the data into training, validation, and testing sets using the train_test_split function from the Scikit-learn library. The function randomly splits the data into two sets, one for training and one for testing, and sets aside a portion of the training data for validation. The split is done such that 80% of the data is used for training and 20% is used for testing, with the training data further split into a 80-20% ratio for training and validation, respectively.

Finally, the code prints out the shapes of the training, validation, and testing data, as well as the shape of the testing labels, to confirm that the data has been split correctly.

```
X = np.concatenate((NormalSounds,SpoofSounds),0)
Y = np.concatenate((normal_labels,SpoofLabels),0)
```

```
Ylength=len(Y)
yList=list(range(Ylength))
random.seed(2021)
random.shuffle(yList)
```

```
trainX=X[yList[0:(Ylength//5)*4]]
testX=X[yList[(Ylength//5)*4:]]
trainY=Y[yList[0:(Ylength//5)*4]]
testY=Y[yList[(Ylength//5)*4:]]
```

```
x_data = trainX
y_data = trainY
test_x = testX
test_y = testY
Class
```

```
['bonafide', 'spoof']
```

```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, train_size=0.8, random_state=42, shuffle=True)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, train_size=0.8, random_state=42, shuffle=True)
```

```
print(y_train.shape,y_test.shape,y_val.shape,test_y.shape)
```

```
(1024,) (320,) (256,) (400,)
```

Figure 5.21

The code snippet as shown in 5.22 defines a sequential model for a Bidirectional Long Short-Term Memory (LSTM) neural network with dropout layers and dense layers using the Keras library.

The Sequential() function initializes the sequential model. The Bidirectional() function creates a bidirectional LSTM layer with 128 units, a dropout rate of 0.05, and recurrent dropout rate of 0.20. The return_sequences=True argument specifies that the LSTM layer should return the entire sequence output at each timestep.

The Dense() function creates dense layers with 128, 64, and 2 units, respectively. The activation parameter specifies the activation function used in each layer. The Dropout() function adds a dropout layer with a rate of 0.3 to reduce overfitting.

The Flatten() function reshapes the output from the previous layer into a 1D vector. The last Dense() layer has 2 units and a softmax activation function, which is suitable for multi-class classification problems.

The compile() function compiles the model, specifying the loss function as categorical_crossentropy, the optimizer as Adam with a learning rate of 1e-4, and the metrics as acc (accuracy)

```
model = Sequential()
model.add(Bidirectional(LSTM(128, dropout=0.05, recurrent_dropout=0.20, return_sequences=True),input_shape=(25,1)))
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Flatten())
model.add(Dense(2, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer=Adam(1e-4), metrics=['acc'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (None, 25, 256)          133120
 l)

 dense (Dense)               (None, 25, 128)           32896

 dropout (Dropout)           (None, 25, 128)           0

 dense_1 (Dense)             (None, 25, 128)           16512

 dense_2 (Dense)             (None, 25, 64)            8256

 dense_3 (Dense)             (None, 25, 64)            4160

 flatten (Flatten)           (None, 1600)              0

 dense_4 (Dense)             (None, 2)                 3202

=================================================================
Total params: 198,146
Trainable params: 198,146
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.22

The code snippet as shown in figures 5.23, 5.24 and 5.25 defines and trains a bidirectional LSTM model with several dense layers for audio spoof detection. It saves the weights of the best performing model during training and uses a learning rate scheduler and early stopping as callbacks. The model's training

history is plotted, and the evaluation accuracy is displayed. Finally, the trained model is saved to a file.

```
weight_saver = ModelCheckpoint('set_a_weights.h5', monitor='val_loss',
                               save_best_only=True, save_weights_only=True)
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.8**x)
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
history=model.fit(x_train, y_train,
                  batch_size=3,
                  epochs=10,
                  class_weight=ClassWeight,
                  callbacks=[weight_saver, annealer],
                  validation_data=(x_val, y_val))
```

```
Epoch 1/10
342/342 [==============================] - 24s 51ms/step - loss: 0.0653 - acc: 0.6426 - val_loss: 0.5138 - val_acc: 0.7461 - l
r: 0.0010
Epoch 2/10
342/342 [==============================] - 16s 45ms/step - loss: 0.0565 - acc: 0.7236 - val_loss: 0.4921 - val_acc: 0.7578 - l
r: 8.0000e-04
Epoch 3/10
342/342 [==============================] - 16s 46ms/step - loss: 0.0530 - acc: 0.7568 - val_loss: 0.4624 - val_acc: 0.7695 - l
r: 6.4000e-04
Epoch 4/10
342/342 [==============================] - 15s 43ms/step - loss: 0.0506 - acc: 0.7578 - val_loss: 0.4390 - val_acc: 0.8008 - l
r: 5.1200e-04
Epoch 5/10
342/342 [==============================] - 14s 42ms/step - loss: 0.0476 - acc: 0.7900 - val_loss: 0.4071 - val_acc: 0.8320 - l
r: 4.0960e-04
Epoch 6/10
342/342 [==============================] - 14s 41ms/step - loss: 0.0445 - acc: 0.8008 - val_loss: 0.4214 - val_acc: 0.8203 - l
r: 3.2768e-04
Epoch 7/10
342/342 [==============================] - 14s 41ms/step - loss: 0.0428 - acc: 0.8096 - val_loss: 0.4037 - val_acc: 0.8438 - l
r: 2.6214e-04
Epoch 8/10
342/342 [==============================] - 14s 41ms/step - loss: 0.0424 - acc: 0.8145 - val_loss: 0.3962 - val_acc: 0.8359 - l
r: 2.0972e-04
Epoch 9/10
342/342 [==============================] - 15s 44ms/step - loss: 0.0406 - acc: 0.8164 - val_loss: 0.3975 - val_acc: 0.8242 - l
```

Figure 5.23

```
: losses[['acc','val_acc']].plot()
: <Axes: >
```

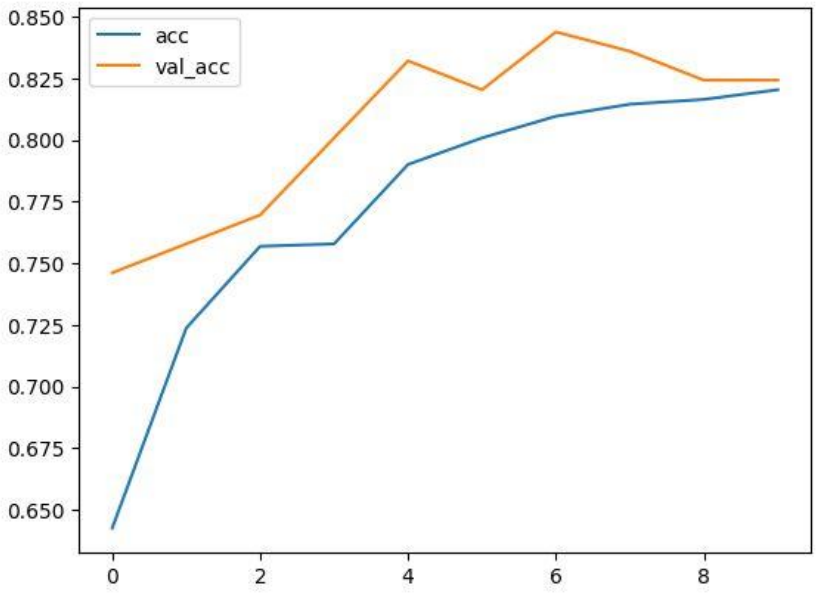


Figure 5.24

```
y_pred = model.predict(x_test, batch_size=5)
#check scores
scores = model.evaluate(x_test, y_test)
print ("Model evaluation accuracy: ", round(scores[1]*100),"%")
```

```
64/64 [==============================] - 1s 5ms/step
10/10 [==============================] - 0s 8ms/step - loss: 0.3944 - acc: 0.8375
Model evaluation accuracy:  84 %
```

```
model.save('D:\\Project\\Audio_Spoof_Detector\\AudioSpoofDetector.h5')
```

Figure 5.25

5.2.3. SPEAKER IDENTIFICATION MODEL

The system uses this python script for speech recognition. It includes functions to record and extract features from audio as shown in figure 5.26, as well as train a Gaussian Mixture Model (GMM) for speaker recognition. The training set is stored in a directory named "training_set" as shown in figure 5.29., and the testing set is stored in a directory named "testing_set" as shown in figure 5.30.

The script prompts the user to enter their name to record 5 speech samples of 10 seconds each for training purposes. Then, it trains a GMM model using the extracted features of the recorded speech samples as shown in figure 5.27. Finally, the user is prompted to record a speech sample of 5 seconds for testing purposes, and the script uses the trained GMM model to predict the identity of the speaker as shown in figure 5.28.

```
def calculate_delta(array):

    rows,cols = array.shape
    print(rows)
    print(cols)
    deltas = np.zeros((rows,20))
    N = 2
    for i in range(rows):
        index = []
        j = 1
        while j <= N:
            if i-j < 0:
              first =0
            else:
              first = i-j
            if i+j > rows-1:
                second = rows-1
            else:
                second = i+j
            index.append((second,first))
            j+=1
        deltas[i] = ( array[index[0][0]]-array[index[0][1]] + (2 * (array[index[1][0]]-array[index[1][1]])) ) / 10
    return deltas



def extract_features(audio,rate):

    mfcc_feature = mfcc.mfcc(audio,rate, 0.025, 0.01,20,nfft = 1200, appendEnergy = True)
    mfcc_feature = preprocessing.scale(mfcc_feature)
    print(mfcc_feature)
    delta = calculate_delta(mfcc_feature)
    combined = np.hstack((mfcc_feature,delta))
    return combined
```

Figure 5.26

```
def record_audio_train():
    Name =(input("Please Enter Your Name:"))
    for count in range(5):
        FORMAT = pyaudio.paInt16
        CHANNELS = 1
        RATE = 44100
        CHUNK = 512
        RECORD_SECONDS = 10
        device_index = 2
        audio = pyaudio.PyAudio()
        print("----------------------record device list---------------------")
        info = audio.get_host_api_info_by_index(0)
        numdevices = info.get('deviceCount')
        for i in range(0, numdevices):
                if (audio.get_device_info_by_host_api_device_index(0, i).get('maxInputChannels')) > 0:
                    print("Input Device id ", i, " - ", audio.get_device_info_by_host_api_device_index(0, i).get('name'))
        print("-----------------------------------------------------------------")
        index = int(input())
        print("recording via index "+str(index))
        stream = audio.open(format=FORMAT, channels=CHANNELS,
                        rate=RATE, input=True,input_device_index = index,
                        frames_per_buffer=CHUNK)
        print ("recording started")
        Recordframes = []
        for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
            data = stream.read(CHUNK)
            Recordframes.append(data)
        print ("recording stopped")
        stream.stop_stream()
        stream.close()
        audio.terminate()
        OUTPUT_FILENAME=Name+"-sample"+str(count)+".wav"
        WAVE_OUTPUT_FILENAME=os.path.join("training_set",OUTPUT_FILENAME)
        trainedfilelist = open("training_set_addition.txt", 'a')
        trainedfilelist.write(OUTPUT_FILENAME+"\n")
        waveFile = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
        waveFile.setnchannels(CHANNELS)
        waveFile.setsampwidth(audio.get_sample_size(FORMAT))
        waveFile.setframerate(RATE)
        waveFile.writeframes(b''.join(Recordframes))
        waveFile.close()
```

Figure 5.27

```python
def record_audio_test_for_spoof():

    FORMAT = pyaudio.paInt16
    CHANNELS = 1
    RATE = 44100
    CHUNK = 512
    RECORD_SECONDS = 5
    device_index = 2
    audio = pyaudio.PyAudio()
    print("----------------------record device list---------------------")
    info = audio.get_host_api_info_by_index(0)
    numdevices = info.get('deviceCount')
    for i in range(0, numdevices):
            if (audio.get_device_info_by_host_api_device_index(0, i).get('maxInputChannels')) > 0:
                print("Input Device id ", i, " - ", audio.get_device_info_by_host_api_device_index(0, i).get('name'))
    print("-----------------------------------------------------------")
    index = int(input())
    print("recording via index "+str(index))
    stream = audio.open(format=FORMAT, channels=CHANNELS,
                    rate=RATE, input=True,input_device_index = index,
                    frames_per_buffer=CHUNK)
    print ("recording started")
    Recordframes = []
    for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
        data = stream.read(CHUNK)
        Recordframes.append(data)
    print ("recording stopped")
    stream.stop_stream()
    stream.close()
    audio.terminate()
    OUTPUT_FILENAME="sample.wav"
    WAVE_OUTPUT_FILENAME=os.path.join("testing_set",OUTPUT_FILENAME)
    trainedfilelist = open("testing_set_addition.txt", 'w')
    trainedfilelist.write(OUTPUT_FILENAME+"\n")
    waveFile = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(audio.get_sample_size(FORMAT))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b''.join(Recordframes))
    waveFile.close()
```

Figure 5.28

```python
def train_model():

    source   = "D:\\Project\\training_set\\"
    dest = "D:\\Project\\trained_models\\"
    train_file = "D:\\Project\\training_set_addition.txt"
    file_paths = open(train_file,'r')
    count = 1
    features = np.asarray(())
    for path in file_paths:
        path = path.strip()
        print(path)

        sr,audio = read(source + path)
        print(sr)
        vector   = extract_features(audio,sr)

        if features.size == 0:
            features = vector
        else:
            features = np.vstack((features, vector))

        if count == 5:
            gmm = GaussianMixture(n_components = 6, max_iter = 200, covariance_type='diag',n_init = 3)
            gmm.fit(features)

            # dumping the trained gaussian model
            picklefile = path.split("-")[0]+".gmm"
            pickle.dump(gmm,open(dest + picklefile,'wb'))
            print('+ modeling completed for speaker:',picklefile," with data point = ",features.shape)
            features = np.asarray(())
            count = 0
        count = count + 1
```

Figure 5.29

```
def test_model():

    source   = "D:\\Project\\testing_set\\"
    modelpath = "D:\\Project\\trained_models\\"
    test_file = "D:\\Project\\testing_set_addition.txt"
    file_paths = open(test_file,'r')

    gmm_files = [os.path.join(modelpath,fname) for fname in
                 os.listdir(modelpath) if fname.endswith('.gmm')]


    #Load the Gaussian gender Models
    models    = [pickle.load(open(fname,'rb')) for fname in gmm_files]
    speakers   = [fname.split("\\")[-1].split(".gmm")[0] for fname
                  in gmm_files]

    # Read the test directory and get the list of test audio files
    for path in file_paths:

        path = path.strip()
        print(path)
        sr,audio = read(source + path)
        vector   = extract_features(audio,sr)

        log_likelihood = np.zeros(len(models))

        for i in range(len(models)):
            gmm    = models[i]  #checking with each model one by one
            scores = np.array(gmm.score(vector))
            log_likelihood[i] = scores.sum()
        print(log_likelihood)
        winner = np.argmax(log_likelihood)
        print("\tdetected as - ", speakers[winner])
        time.sleep(1.0)
```

Figure 5.30

5.2.4. AUDIO SPOOF DETECTION INTEGRATED WITH A HOME AUTOMATION SYSTEM USING IOT

In the code snippet as shown in figure 5.31, the already pre-trained audio spoof model is imported.

```
Model=load_model('AudioSpoofDetector.h5')

Model.summary()
```

Figure 5.31

The code as shown in figure 5.32 reads an audio file, converts it to the FLAC format, loads it using librosa, preprocesses the data, and then predicts whether it's a genuine or spoofed audio using a pre-trained model. If the prediction is 0, it prints "Access Granted!" else it prints "Seems like an Spoofed Audio, Access Denied."

```
def audio_spoof():
    song = AudioSegment.from_wav("D:\\Project\\testing_set\\sample.wav")
    song.export("testme.flac",format = "flac")
    filename="testme.flac"
    sample_sound, sample_rate=librosa.load(filename)
    print("Hello")
    features=data_preprocessing(filename)
    y_pred = Model.predict(features)
    PRED = y_pred.argmax(axis=1)
    print(PRED[0])
    if(PRED[0]==0):
        print("Access Granted!")
    else:
        print("Seems like an Spoofed Audio , Access Denied.")
```

Figure 5.32

The code as shown in figure 5.33 is used to communicate with an Arduino board via a serial port. It lists all available ports, asks the user to select a port, and then opens the port with a baud rate of 9600. It then enters into a loop where it takes input from the user and sends it to the Arduino board. If the input is "exit," the program terminates.

```
def activate_Iot():

    ports  =  serial.tools.list_ports.comports()
    serialInst = serial.Serial()
    portsList = []
    portVar=''
    for onePort in ports:
        portsList.append(str(onePort))
        print(str(onePort))
    val = input("Select Port: COM")

    for x in range(0,len (portsList)):
        if portsList[x].startswith("COM" + str(val)):
            portVar = "COM" + str(val)
            print(portVar)

    serialInst.baudrate = 9600
    serialInst.port = portVar
    serialInst.open()
    while True:
        command= input("Arduino Command: (ON/OFF):")
        serialInst.write(command.encode('utf-8'))
        if command == 'exit':
            return
```

Figure 5.33

The system as shown in the code snippet in the figure 5.34 then implements a menu-based program that runs in a loop until the user chooses to exit. The program has three options:

Register for New User: This option records the user's voice and trains a machine learning model to recognize it.

Give Command: This option records the user's voice again and tests it against the trained model. If the voice is recognized as the registered user, it activates

an IoT device. Otherwise, it prompts the user to register again if they are a new user, or denies access if the voice is illicit.

Exit Application: This option exits the program.

The program uses several functions to accomplish its tasks, including record_audio_train() for recording the user's voice during registration, train_model() for training the machine learning model, record_audio_test_for_spoof() for recording the user's voice during command giving, audio_spoof() for detecting if the voice is an illicit spoof, test_model() for testing the user's voice against the trained model, and activate_Iot() for activating the IoT device if the voice is recognized as valid.

```
while True:
    choice = int(input("\n 1.Register for New User\n 2.Give Command:\n 3.Exit Application\n"))
    if choice == 1:
        record_audio_train()
        train_model()
    elif choice == 2:
        record_audio_test_for_spoof()
        audio_spoof()
        speaker , valid = test_model()
        if(valid):
            activate_Iot()
        else:
            print("Invalid / Illicit voice detected\n If not an existing user , Register your voice!\n")
    elif choice == 3:
        break
```

Figure 5.34

```
#define LED_pin 2
#define LED_error_pin 3


void setup() {
// put your setup code here, to run once:
  Serial.begin(9600);
  pinMode (LED_pin, OUTPUT);
  pinMode (LED_error_pin, OUTPUT);
}


void loop()
{
// put your main code here, to run repeatedl
  if (Serial.available() > 0)
    {
      String msg= Serial.readString();
      if (msg=="ON")
      {
        digitalWrite(LED_pin, HIGH);
      }
      else if (msg=="OFF")
      {
        digitalWrite(LED_pin, LOW);
      }
      else
      {
        digitalWrite(LED_error_pin, HIGH);
        delay(100);
        digitalWrite(LED_error_pin, LOW);
      }
    }
}
```

Figure 5.35 – Arduino code

RESULTS AND DISCUSSIONS

CHAPTER 6

RESULTS AND DISCUSSIONS

6.1 AUDIO SPOOF DETECTION MODEL

The classification report obtained from the audio spoof detection model is shown in the figure 6.1.

```
target = ["Bonafide", "Spoof"]
print(classification_report(flag, predict, target_names=target, digits=4))
#print(confusion(flag, predict, target_names=target, digits=4))
```

	precision	recall	f1-score	support
Bonafide	0.7791	0.8645	0.8196	155
Spoof	0.8581	0.7697	0.8115	165
accuracy			0.8156	320
macro avg	0.8186	0.8171	0.8155	320
weighted avg	0.8198	0.8156	0.8154	320

Figure 6.1

The confusion matrix obtained from the audio spoof detection model is shown in the figure 6.2 and the heat map for the same is in the figure 6.3.

```
matrix_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_mat

matrix_display.plot()
plt.show()
```

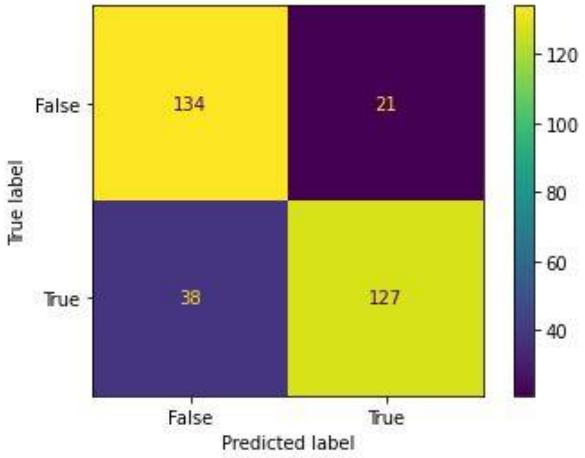


Figure 6.2

```
import seaborn as sns
labels = ['True Neg','False Pos','False Neg','True Pos']
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(confusion_matrix, annot=labels, fmt='', cmap='Blues')
```

```
<AxesSubplot:>
```

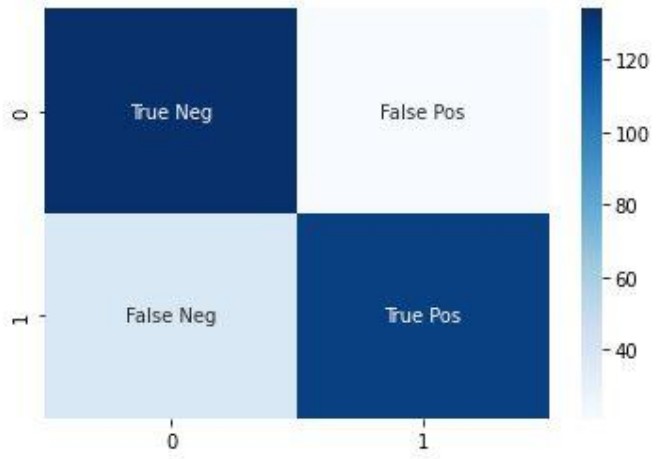


Figure 6.3

The code snippet in the figure 6.4 predicts whether the audio sample is authentic or spoofed.

```
def Predict(feature):
    y_pred = Model.predict(feature)
    PRED = y_pred.argmax(axis=1)
    print(PRED[0])
    if(PRED[0]==0):
        print("Access Granted!")
    else:
        print("Seems like an Spoofed Audio , Access Denied.")
```

```
Predict(authentic)
```

```
0
Access Granted!
```

```
Predict(spoof)
```

```
1
Seems like an Spoofed Audio , Access Denied.
```

Figure 6.4

6.2. SPEAKER IDENTIFICATION MODEL

The GMM model identifies the appropriate speaker as shown in the figures 6.5 and 6.6.

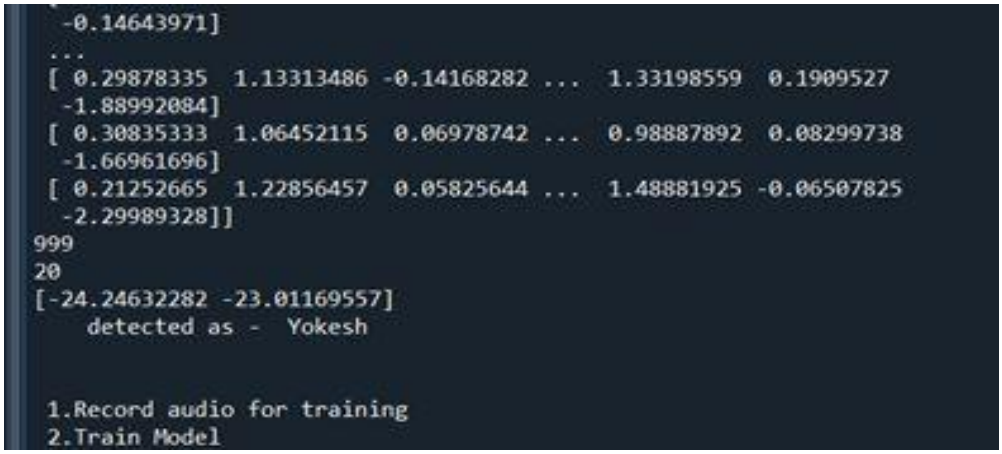


Figure 6.5

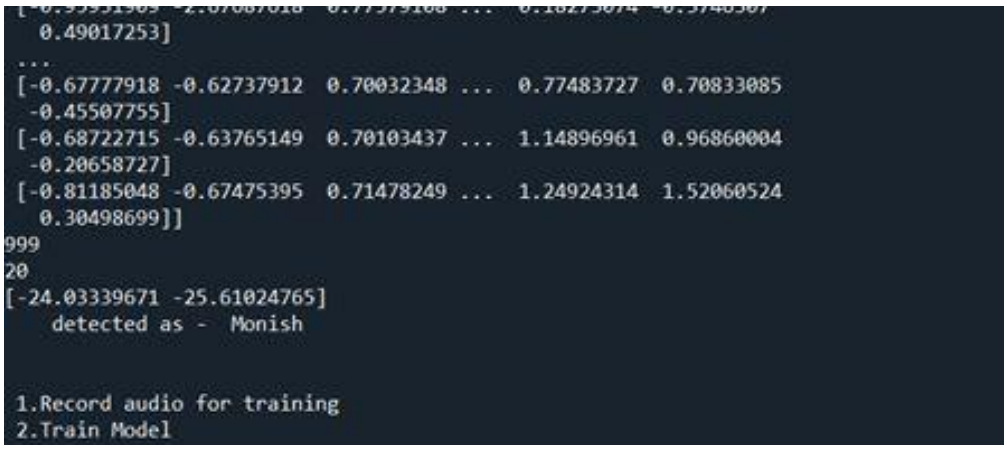


Figure 6.6

The speaker identification model, if identified a spoofed audio will deny access as shown in the figure 6.7.

```
 1.Register for New User
 2.Give Command:
 3.Exit Application
2
----------------------record device list---------------------
Input Device id  0  -  Microsoft Sound Mapper - Input
Input Device id  1  -  Microphone Array (Realtek(R) Au
Input Device id  2  -  Stereo Mix (Realtek(R) Audio)
-------------------------------------------------------------
1
recording via index 1
recording started
recording stopped
1/1 [==============================] - 1s 581ms/step
1
Seems like an Spoofed Audio , Access Denied.
```

Figure 6.7

The speaker identification model will request to register user voice if there's no voice registered and if an illicit user is identified as shown in figure 6.8.

```
 ...
 [ 0.15221828  1.26449332  1.03503253 ...  0.37779237  1.23671075
   0.24163612]
 [-0.44532079  1.03351966  1.26981029 ... -0.79256775  0.51309024
   0.24379439]
 [-1.8782427   1.32779791  1.93790059 ... -0.43446857  0.05148239
  -0.12570875]]
498
20
[-35.59924758 -35.65708072]
    No voice detected.
Invalid / Illicit voice detected
 If not an existing user , Register your voice!
```

Figure 6.8

6.3. AUDIO SPOOF DETECTION INTEGRATED WITH A HOME AUTOMATION SYSTEM USING IOT

A sample execution of the proposed system is shown in the figures 6.9, 6.10, 6.11 and 6.12.

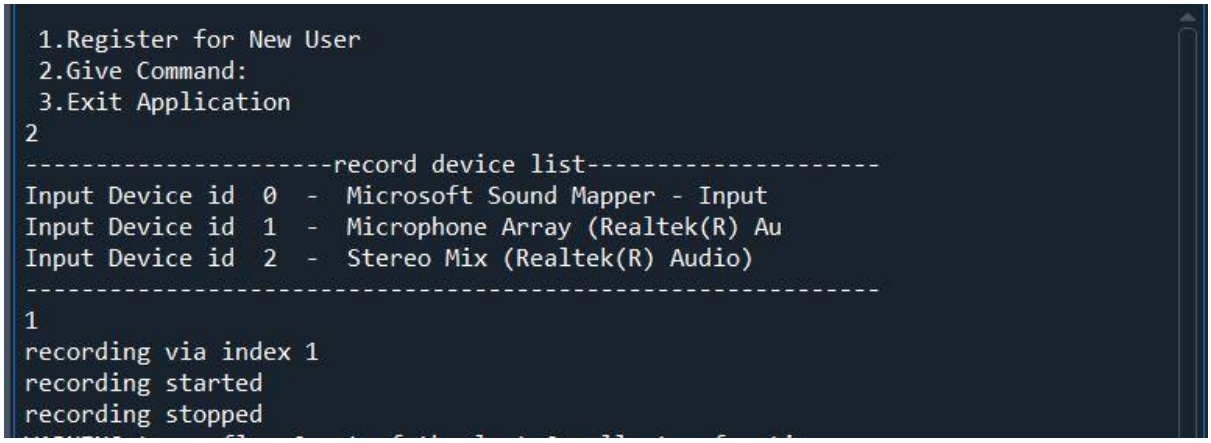


Figure 6.9

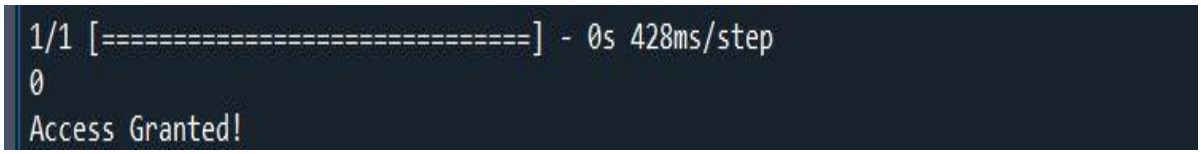


Figure 6.10

```
498
20
[-22.5388012  -21.48969341]
    detected as -  Yokesh.gmm
COM3 - Standard Serial over Bluetooth link (COM3)
COM6 - USB-SERIAL CH340 (COM6)
COM4 - Standard Serial over Bluetooth link (COM4)
Select Port: COM6
COM6
Arduino Command: (ON/OFF):ON
Arduino Command: (ON/OFF):OFF
Arduino Command: (ON/OFF):L
Arduino Command: (ON/OFF):ON
Arduino Command: (ON/OFF):OFF
Arduino Command: (ON/OFF):
```

Figure 6.11

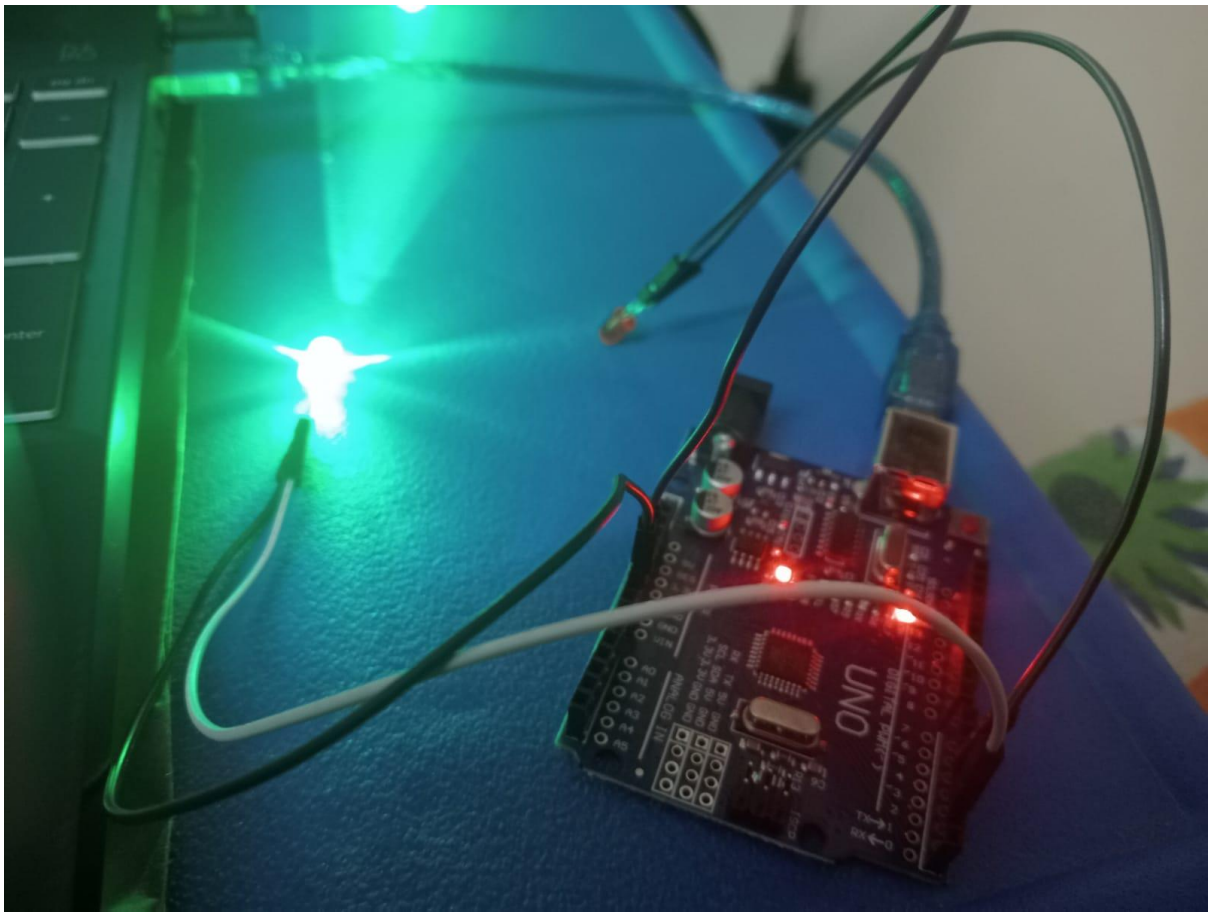


Figure 6.12

COST-BENEFIT ANALYSIS

CHAPTER 7

COST-BENEFIT ANALYSIS

7.1 TIME AND COST ESTIMATION

Developing a user friendly graphical user interface app for smartphone for audio recording will require time and expertise. If the app requires to be integrated with any client - server architecture then additional development and testing would be required for the system.

7.1.1 IMPLEMENTATION COST

The proposed system may need one time implementation cost of the Iot system which works on adruino board. The implementation cost will vary according to the adruino board configuration. If it is Adruino uno board implementation then cost comes around 1500Rs/-.

7.1.2 MAINTENANCE COST

The proposed system requires regular maintenance to ensure that it continues to function as intended. This includes monitoring the audio processing, updating software and Iot hardware, and fixing any issues that may arise. The cost of maintenance will depend on the Iot components used.

7.2 BENEFITS

The proposed system grants access to the authorized users to use their home automation devices like Fan, Air conditioner, etc.. and the system denies access to the unauthorized users. It prevents audio spoof attacks such as “Voice replay attacks” and “Voice conversion attacks” as the system is implemented with spoof detection module.

CONCLUSION

CHAPTER 8

CONCLUSION

The Home automation system developed in this project is a reliable and efficient solution for safeguarding against audio spoofing attacks. The integration of an audio spoof detector and speaker identification system ensures that only authorized users can access the automated devices in the home. The accuracy obtained for the audio spoof detection model is 85%, which is a significant improvement over existing methods.

The proposed speaker identification system, which utilizes MFCC feature extraction and GMM model, has been tested against both known and unknown users. The system performed as expected, effectively identifying the known user while denying access to the unknown users. This ensures that only authorized users can control the home automation system, thereby increasing security.

The use of Arduino UNO board, LED lights, servo motor, connecting wires, and breadboard for IoT integration provides a cost-effective and reliable solution for home automation. The integration of IoT devices with the audio spoof detector and speaker identification system ensures that users can control their home environment with ease and convenience.

The System demonstrates the effectiveness of integrating an audio spoof detector and speaker identification system with IoT devices for home automation. The proposed system provides reliable protection against audio spoofing attacks while ensuring that only authorized users can access the automated devices in the home. The accuracy obtained for the proposed system is promising and shows great potential for future research in this field.

REFERENCES

REFERENCES

[1] Dua, M., Jain, C. & Kumar, S. LSTM and CNN based ensemble approach for spoof detection task in automatic speaker verification systems. *J Ambient Intell Human Comput* **13**, 1985–2000 (2022). https://doi.org/10.1007/s12652-021-02960-0

[2] Yamagishi, Junichi; Todisco, Massimiliano; Sahidullah, Md; Delgado, Héctor; Wang, Xin; Evans, Nicolas; Kinnunen, Tomi; Lee, Kong Aik; Vestman, Ville; Nautsch, Andreas. (2019). ASVspoof 2019: The 3rd Automatic Speaker Verification Spoofing and Countermeasures Challenge database, [sound]. University of Edinburgh. The Centre for Speech Technology Research (CSTR). https://doi.org/10.7488/ds/2555.

[3] https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

[4] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[5] Sak, H., Senior, A.W., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *INTERSPEECH*.

[6] Ankur, Tanjemoon & Kundu, Bipasha & Foysal, Md & Ortiz, Bengie & Chong, Jo. (2022). LSTM-Based COVID-19 Detection Method Using Coughing. 10.21203/rs.3.rs-2106413/v1.

[7] Akyol K, Şen B. Automatic Detection of Covid-19 with Bidirectional LSTM Network Using Deep Features Extracted from Chest X-ray Images. Interdiscip Sci. 2022 Mar; 14(1):89-100. doi: 10.1007/s12539-021-00463-2. Epub 2021 Jul 27. PMID: 34313974; PMCID: PMC8313418.

[8] Ivan Rakhmanenko Fusion of BiLSTM and GMM-UBM Systems for Audio Spoofing Detection August 2019 International Journal of Advanced Trends in Computer Science and Engineering 6(4):1741-1746.

[9] Jichen Yang, Rohan Kumar Das,Improving anti-spoofing with octave spectrum and short-term spectral statistics information,Applied Acoustics, Volume 157,2020,107017,ISSN 0003-682X,

[10] https://ijirt.org/master/publishedpaper/IJIRT149877_PAPER.pdf

[11] Mittal, Aakshi & Dua, Mohit. (2022). Automatic speaker verification systems and spoof detection techniques: review and analysis. International Journal of Speech Technology. 25. 10.1007/s10772-021-09876-2.

[12] https://irojournals.com/aicn/article/pdf/4/3/4

[13] https://journals.pan.pl/dlibra/publication/141648/edition/123487/content/archives-of-acoustics-2022-vol-47-no-2-spoofed-speech-detection-with-weighted-phase-features-and-convolutional-networks-br-disken-gokay?language=en

[14] Zhou, J., Hai, T., Jawawi, D.N.A. *et al.* Voice spoofing countermeasure for voice replay attacks using deep learning. *J Cloud Comp* **11**, 51 (2022). https://doi.org/10.1186/s13677-022-00306-5