Deep Fake Audio Detection

Project Team

Hafiz Usman	18P-0100
Umair Azad	19P-0030
Faiq Aslam	19P-0082

Session 2019-2023

Supervised by

Ms.Sara Rehmat

Department of Computer Science

National University of Computer and Emerging Sciences
Peshawar, Pakistan

December, 2023

Student's Declaration

We declare that this project titled "*Deep Fake Audio Detection*", submitted as requirement for the award of degree of Bachelors in Computer Science, does not contain any material previously submitted for a degree in any university; and that to the best of our knowledge, it does not contain any materials previously published or written by another person except where due reference is made in the text.

We understand that the management of Department of Computer Science, National University of Computer and Emerging Sciences, has a zero tolerance policy towards plagiarism. Therefore, We, as authors of the above-mentioned thesis, solemnly declare that no portion of our thesis has been plagiarized and any material used in the thesis from other sources is properly referenced.

We further understand that if we are found guilty of any form of plagiarism in the thesis work even after graduation, the University reserves the right to revoke our BS degree.

Hafiz Usman	Signature: ________________
Umair Azad	Signature: ________________
Faiq Aslam	Signature: ________________

____

Verified by Plagiarism Cell Officer
Dated:

Certificate of Approval

The Department of Computer Science, National University of Computer and Emerging Sciences, accepts this thesis titled *Deep Fake Audio Detection*, submitted by Hafiz Usman (18P-0100), Umair Azad (19P-0030), and Faiq Aslam (19P-0082), in its current form, and it is satisfying the dissertation requirements for the award of Bachelors Degree in Computer Science.

Supervisor

Ms.Sara Rehmat Signature: ________________

____

Zeshan Khan

FYP Coordinator

National University of Computer and Emerging Sciences, Peshawar

____

Dr. Nouman Azam

HoD of Department of Computer Science

National University of Computer and Emerging Sciences

Acknowledgements

I would like to start by mentioning Allah (SWT), the Most Gracious and the Most Merciful. Allah (SWT) granted us the strength to finish this project, and we are truly grateful for all the opportunities, challenges, and blessings that came our way. Many people, directly or indirectly, played a role in this, especially our family members who motivated and supported us. A big thanks to each one of them for sacrificing their time and resources.I would like to express our sincere appreciation to Ms. Sara Rehmat for supervising our project. She has been a constant support, helping us through tough times and giving us her valuable time whenever we needed it. In conclusion, we want to thank everyone mentioned, and we hope to make you proud in the future, Insha'Allah.

Hafiz Usman

Umair Azad

Faiq Aslam

Abstract

Deepfake audios are those audios that are created by AI technologies. There are different types of fake audios, and we're specifically focusing on clipped audio. Right now, methods for detecting fake audios only work for certain languages. Since there's no work done in Urdu, which is widely spoken in Asia and South Asia, we decided to tackle that. Our project is all about research. We want to know whether a deep learning model is good at spotting clipped audios. We created Urdu real audio dataset by extracting audio from urdu documented videos on YouTube and shuffle the bits within them to make fake ones. After that, we did some preprocessing work by applying noise cancelation and used Mel Frequency Cepstrum Coefficient feature extraction for extracting features of an audio. We then finetune a Resnet-50. It's a fancy deep learning model that's already been trained before.Our model is doing really well.We have successfully demonstrated that only clipping can also be used to detect whether an audio is "Real" or "Fake". We have got an overall accuracy of 0.8309, recall of 0.8309, precision of 0.83, and F1 score of 0.8309. This means our deep learning model is great at catching clipped audios. Particularly for public figures like politicians and celebrities, it's incredibly helpful. They might utilize it to prevent any misconceptions brought up by fake audios.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Deepfake audios refers to those audios that are generated by Artificial Intelligence(AI) and deep learning techniques. There are different types of fake audios. Some of them are Speech Synthesis, Voice Cloning, Spliced(clipped) audios.

1. **Speech Synthesis:** These are synthetic voices that are created by Text to speech (TTS) systems, and can be used to create fake audios. These systems convert written text into spoken words and some advance TTS models can mimic the nuances of natural speech.

2. **Voice Cloning:** This involves deep learning algorithms to replicate someone's voice. By training a model on a person's voice recordings, it can generate new audio that sounds like the person speaking.

3. **Spliced(clipped) audios:** Spliced audios refers to a type of manipulated audio in which different segments or pieces of audio recordings are combined to create a new, often misleading, composition. This can involve taking fragments of audio from different sources of the same person and seamlessly meaning them to form a contextually altered audio clip. Spliced audios can be created using various editing techniques.

Let’s assume two audio clips of a person:

> **Audio 1:** “Climate change is a serious issue and it is affecting our environment.”
>
> **Audio 2:** “I’m happy that John went to Germany for higher studies.”
>
> **Spliced Audio(Generated):** “I’m happy that climate change is affecting our environment”.

Here different segments from two audios are merged together to create a new audio, Notably, the voice remains unaltered, but the context is manipulated, highlighting the potential danger of such fake audios.

In our project we aimed to address these issues by utilizing a pretrained deep learning model ResNet-50 on an audio dataset containing both real audios and spliced(clipped) audios. Detecting the impact of these manipulated audios is crucial, especially in worst-case scenarios where misinformation could have severe consequences.

1.2 Motivation

It has been observed that the number of audio leaks in Pakistan is a worrying trend, as it points to a growing trend of using technology to manipulate and spread disinformation. These audio clips are frequently directed towards public figures or politicians or other powerful people in an attempt to discredit or harm their reputation. The application of deepfake audio technology can make it very difficult to distinguish between real and fake audio recordings, as the manipulated clips can sound very convincing. This can lead to a breakdown in trust and credibility, as people may start to doubt the authenticity of all audio recordings, including those that are real.It is important for people to be aware of the potential for deepfake audio manipulation, and for authorities to take steps to prevent the spread of such disinformation. This can include developing and deploying effective detection technologies, educating people on how to identify manipulated audio, and enforcing laws and regulations that prevent the spread of fake or misleading information.

1.3 Existing System

Deepfake audio datasets in many languages have been used to train different deep learning algorithms for detection. There's been a ton of research on synthetic audios, but surprisingly, none of it focuses on Urdu. For this reason, we are delving into the realm of Urdu language in our project.

1.4 Proposed System

The first step in our new system is to create a dataset. For real audios we extract Urdu audios from YouTube videos and shuffle the segments between to create fake audios. After creation of raw dataset we then preprocessed the dataset and then we trained a deep learning model to make predictions whether an audio is real or fake.

Chapter 2

Review of Literature

2.1 Deep Fake Audio Detection

In the field of audio forensics, audio authentication is a preliminary endeavour when an audio clip is utilised as proof. These days, changing audio is simple thanks to technological advancements. audio manipulation through splicing, deletion, and copy-move operations [12] ,[14]. Splicing is the act of inserting audio at the beginning, middle, or end of another audio in a forgery [12] ,[4] ,[5]. Such tampering can occur in audio manipulation, so it is important for audio forensics to detect it when it occurs.Prior research on audio splicing detection has extracted features like the Decay Rate Parameter (reverberation parameters), Mel Frequency Cepstral Coefficient (MFCC), Linear Prediction Coefficient (LPC), and Electric Network Frequency (ENF). A sign of tampering is fickleness in these features. The computed characteristics are subsequently utilised by classifiers, such as the Support Vector Machine (SVM), Hidden Markov Model (HMM), and Gaussian Mixture Model (GMM), to identify instances of manipulation [12], [13].Numerous research papers have limitations, such as a lack of audio data sets and inadequate results based on one, two, or three second audio samples.In 2017, a user going by the handle "deepfakes" posted a fake video on Reddit featuring the face of a different actor. This was the first instance of Deepfake appearing online. It is inevitable that, being a new technology, it comes with a host of legal challenges that violate private rights such as reputation, copyright, and portraiture and cause financial and reputational damage to companies [10], [9]. In addition, a fake video featuring a politician or government official being unveiled is likely to trigger a crisis in the media, social unrest, and national instability [11],[1],[2].The process of using a speaker's voiceprint to confirm their identification is known as automatic speaker verification, or ASV. One of the most practical methods of biometric identification is ASV. Unfortunately, spoofing poses a serious risk to the security provided by ASV systems, just like it does for any biometric. Since the produced images, videos, and audios are only growing more realistic, it is getting difficult to detect faked data due to the development of new technologies like deepfakes [6]. The requirement for technologies that can automatically assess the integrity of those material is growing along with the quality of generative algorithms.ASVspoof2021 [3], an extension of ASVspoof2019 [7], aims to develop countermeasures to detect spoofed audio involving text-to-speech (TTS), voice conversion (VC), and replayed attacks. No training or development data has been released that matches the telephony encoding and transmission artefacts encountered during evaluation. This is to address real-world spoofing scenarios.

Chapter 3

Project Vision

3.1 Scope

- Developing and implementing a deep learning model for detecting deep fake audio recordings in a specific language (i.e. Urdu).
- Collecting a dataset of real and fake audio recordings for training and testing the deep learning model.
- Preprocessing and feature extraction of audio data using machine learning algorithms.
- Evaluating the accuracy and effectiveness of the deep-fake audio detection model through rigorous testing and validation.
- Developing a user interface or API for accessing the deepfake audio detection software.

3.2 Problem Statement

The lack of effective deep fake audio detection techniques for Urdu is a cause for concern, as it is a widely spoken language in regions that are often affected by social and political instability.. The spread of false audio content in Urdu can exacerbate these issues, making it imperative to develop language-specific deep fake detection methods. More research and development are needed in this area to address this growing concern.

3.3 Proposed System

For our project, we are utilising clipping to determine whether the audio is authentic or fraudulent. We download five thousand 10-second YouTube videos in order to create clipping-based audios. Next, we use a Python script to shuffle the audio' segments.

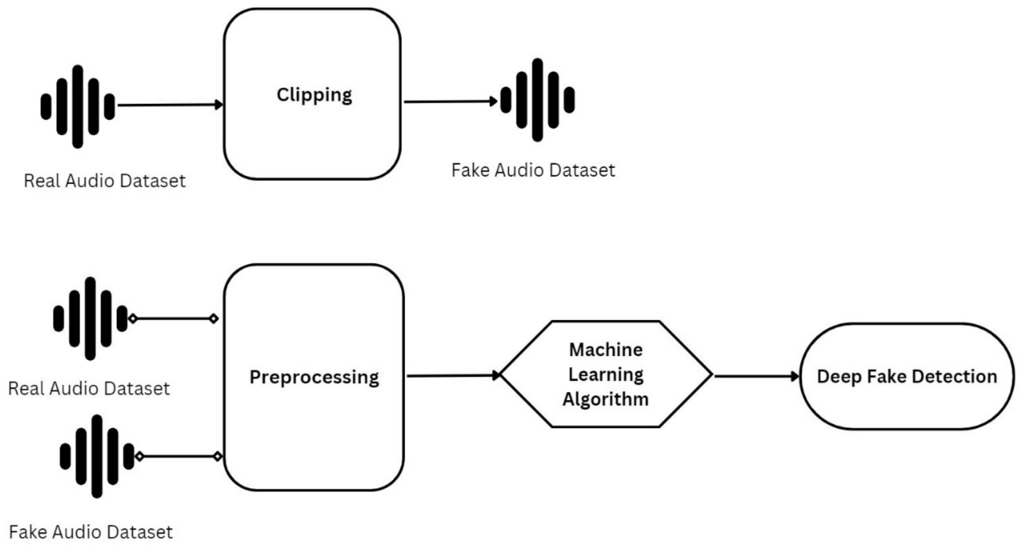


Figure 3.1: Proposed System

3.4 Objectives

The objective is to develop a deep learning model specifically designed for the detection of deepfake audio recordings in the Urdu language. This involves collecting and curating a large data set of real and manipulated Urdu audio recordings, preprocessing the data, and training machine learning algorithms such as convolutional neural networks or recurrent neural networks to detect and classify deep fake audio recordings accurately.

3.5 Business Opportunities

- Selling software to individuals, businesses, or governments who are concerned about the spread of disinformation and want to protect themselves and their audiences.
- Offering software as a service, where clients can pay for access to your software through a subscription model.
- Partnering with other companies or organizations to integrate your software into their existing platforms or services.
- Offering training and education programs to teach others how to use your software and detect deepfake audio.

Chapter 4

Software Requirements Specifications

This chapter will have the functional and non functional requirements of the project.

4.1 User Requirements

- The user can sign up and log in when they visit the page.
- Once the user logs in or signs up, it receives audio input from them and determines whether or not it is fake.
- Then the user has the option to download the audios fake or original status.

4.2 Non Functional Requirements

The following are the non-functional requirements of our software:

- When a user enters audio, our system retrieve the results and indicates whether the audio is real or not.
- Our system only processes one audio at a time, so it should operate flawlessly.
- The system should be reliable.

4.3 Use Case Diagram

Consider a use case diagram as a graphic representation of how users interact with a system.The user is the only primary actor in this scenario. The user can visit the homepage, register, log in, and upload an audio file. After that, the system activates, performs some pre-processing magic, and informs the user of the authenticity of the audio. It functions much like a step-by-step manual.

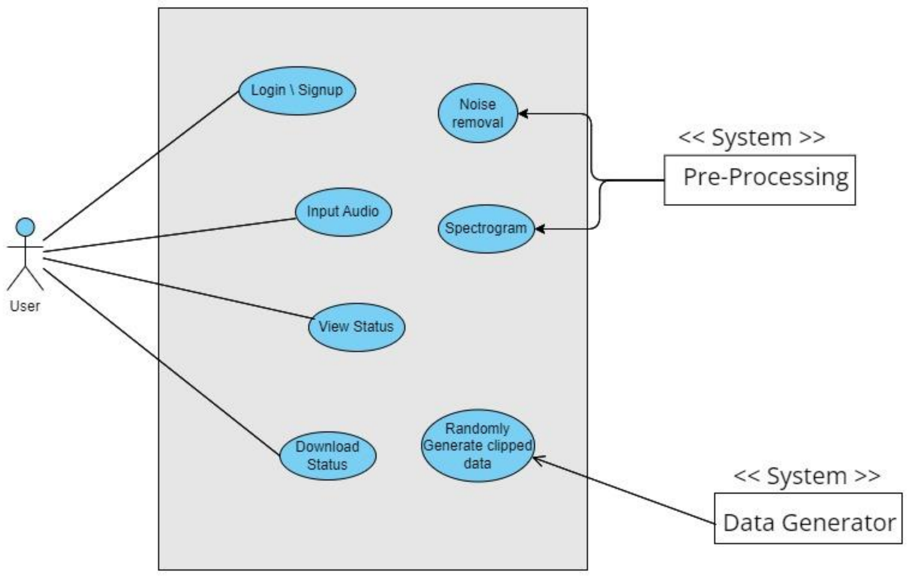
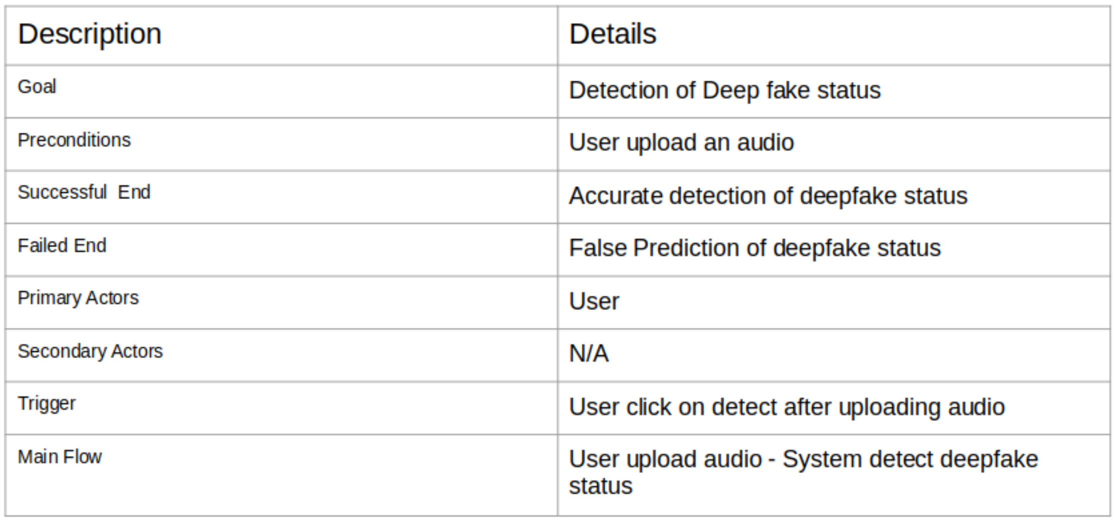


Figure 4.1: Use Case Diagram

4.3.1 Use Case Description



Description	Details
Goal	Detection of Deep fake status
Preconditions	User upload an audio
Successful End	Accurate detection of deepfake status
Failed End	False Prediction of deepfake status
Primary Actors	User
Secondary Actors	N/A
Trigger	User click on detect after uploading audio
Main Flow	User upload audio - System detect deepfake status

Figure 4.2: Use Case Description

4.4 Activity Diagram

Think of an activity diagram as a map of the system. In our scenario, a user inputs an audio file, which our web application checks to see if it is valid or not. After validation, we perform certain pre-processing tasks such as noise reduction, convert to spectrograms, and then extract features. Subsequently, our classifier will determine whether the audio is authentic or not and present the results to the user.

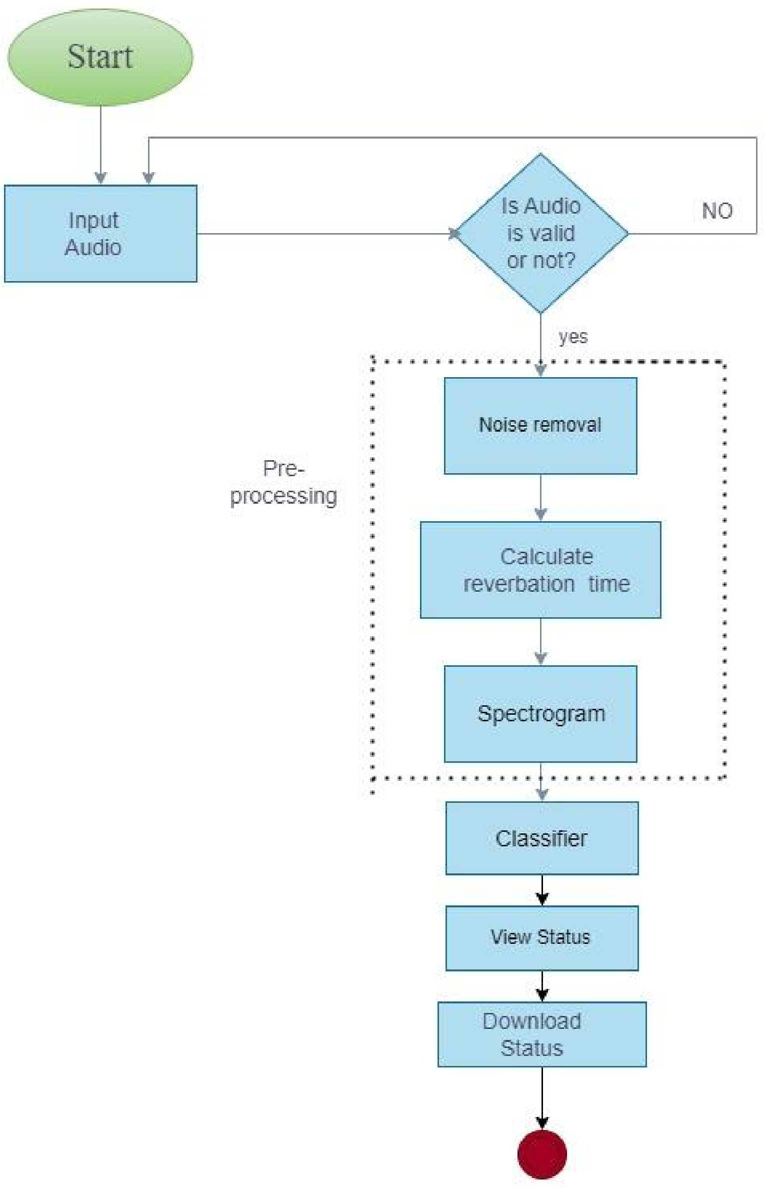


Figure 4.3: Activity Diagram

Chapter 5

Iteration 1

5.1 Fyp-1 Iteration Plan

In first iteration plan we undertook the task of gathering a dataset comprising various audio samples. Once the audio data was assembled, we proceeded to generate artificial or synthetic audio files through a conversion process. Following this, we engaged in a comprehensive pre-processing stage, which involved various techniques to enhance and prepare the data for further analysis. Subsequently, we focused on training a specialized model tailored for handling replay-based audio scenarios. This multi-step approach allowed us to effectively evaluate and address the challenges associated with this specific type of audio data.

5.2 Swimlane Diagram

Consider a Swimlane diagram, which is a visual representation of how several process components collaborate to complete a task. According to our diagram, a user must first register, log in, then upload an audio file to verify whether it is real or fake. After that, our system performs some preprocessing and decides using a deep learning classifier. Ultimately, the results are displayed to the user directly on the interface.

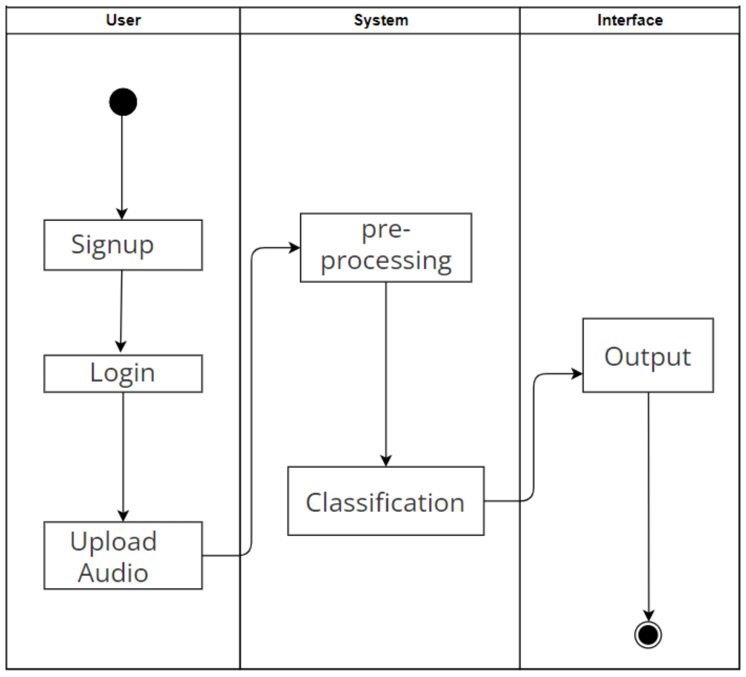


Figure 5.1: Swimlane Diagram

5.3 System FLow Diagram

Consider a system flow diagram as a straightforward map that illustrates how a system operates. We have two different types of sounds in this instance: real and fake. After converting these audio files into spectrograms, we extract specific features. We then run these features via a classifier after that. The classifier finally provides us with a result.

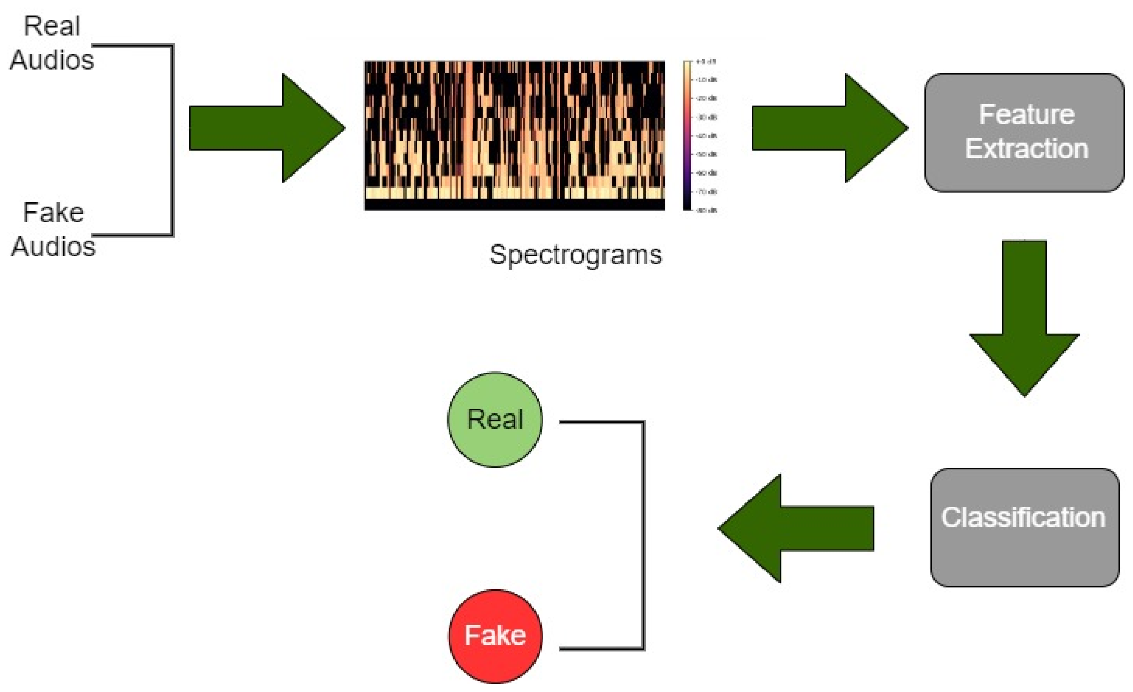


Figure 5.2: System Flow Diagram

5.4 Block Diagram

Consider a block diagram as a type of map that shows how a system or model works. It deconstructs the system into its component parts and explains how each one functions to keep the whole thing together. Every block has a specific function or goal to do, and when combined, they allow the system to function as a whole. It's similar to dividing a large work into smaller, more doable tasks.

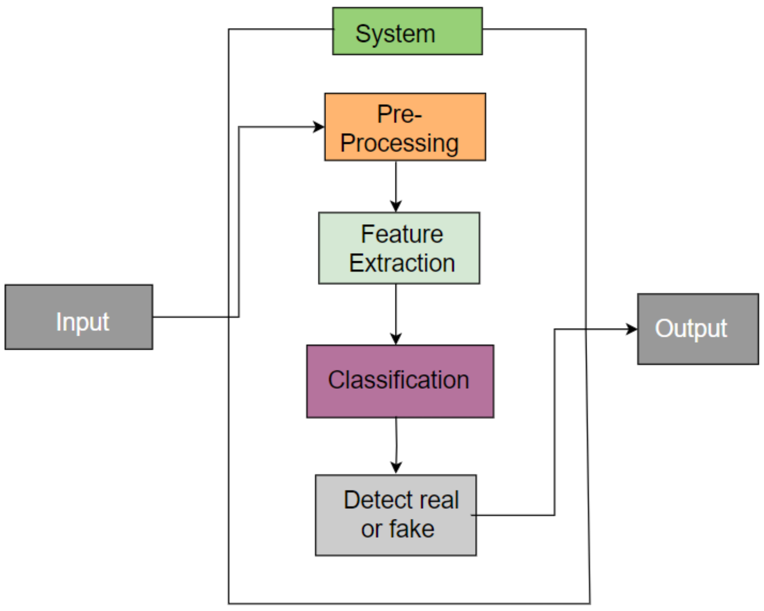


Figure 5.3: Block Diagram

Chapter 6

Iteration 2

6.1 FYP-2

In the second iteration of our plan, we set out to train model on the dataset we created. We created spectrograms of the audio dataset and labeled it. We saved Real audio spectrogram in one folder named as ‘Real’ and fake audio spectrogram on different folder named as ‘Fake’. After that we splitted the data into training testing and validation and trained the model. Spectrograms are basically images of the audios we are converting audios into images and then train the model on the images we created. As this project is experimentation so we tested different base models and pretrained models. We have experimented the training on CNN-base model which is not trained on any data before along with different pretrained models like Mobilenet v2, Resnet 18, Resnet 50.

6.2 Timeline1

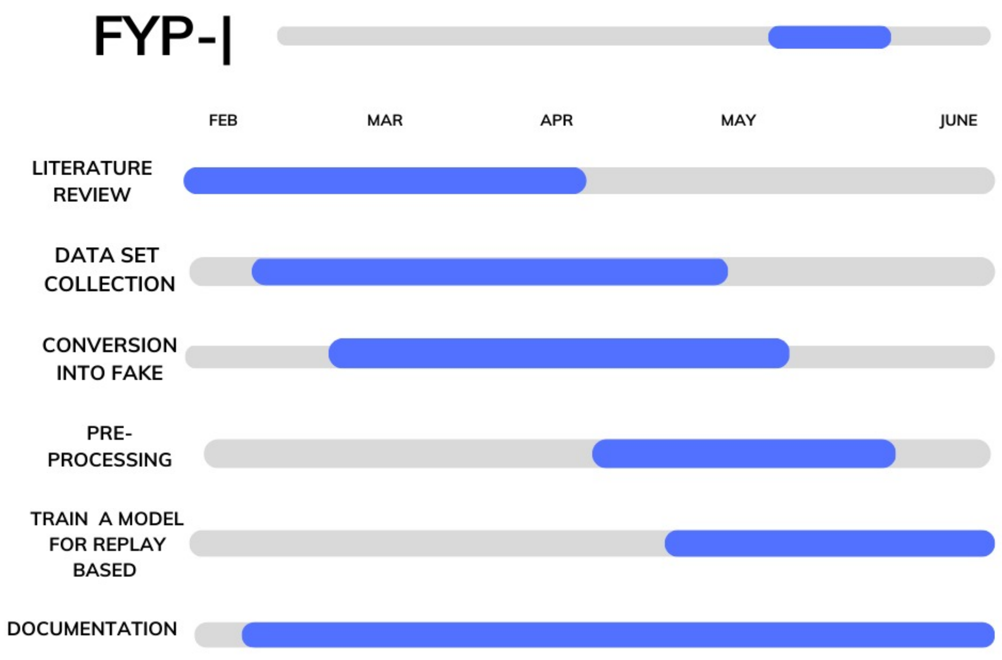


Figure 6.1: Timeline 1

6.3 Timeline2

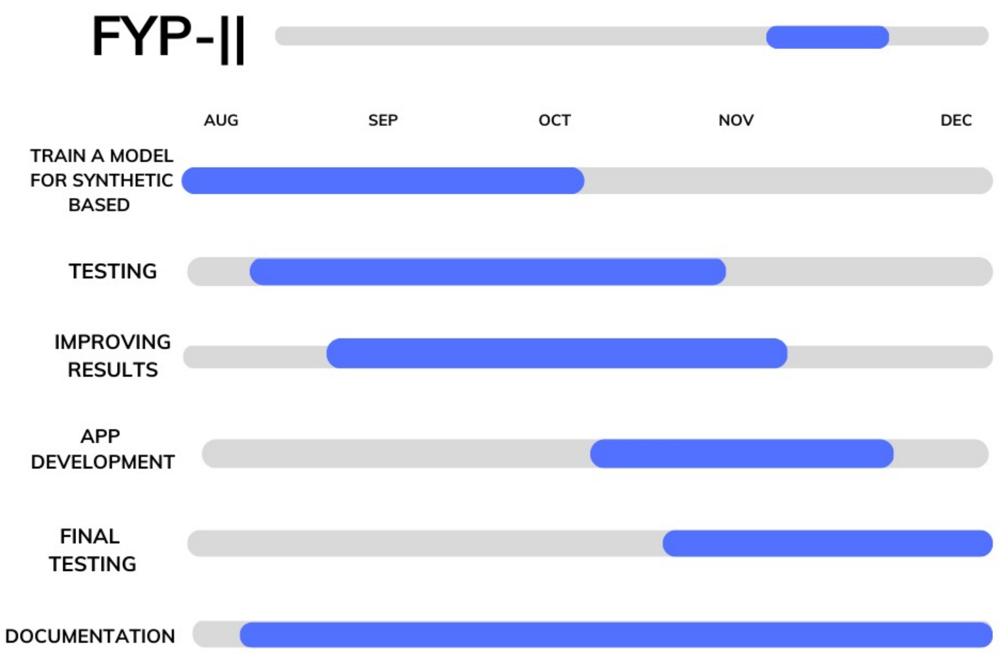


Figure 6.2: Timeline 2

Chapter 7

Implementation

7.1 Data Generation

It was rather difficult to create our dataset because there isn't a pre-made Urdu audio collection. So, here's what we did: we grabbed videos from YouTube, took out the audio, and then chopped it up into small pieces to make real audio clips for our dataset. Once we had those, we got crafty and made fake audios by messing with the real ones. We did this thing called "clipping," where we broke the real audio into chunks and mixed them up. That is how our dataset was assembled.

7.2 Pre-processing

We prepared our dataset after obtaining it. Initially, we eliminated any irritating background noise from the audio recordings. After that, we converted them into spectrograms. The next phase included utilizing MFCC to extract features from these spectrograms. Lastly, we ran them through a number of deep learning models.

$$\text{MFFCC}(x) = \sum_{i \in N(x)} \frac{\mu_A(i) \cdot f(i) \cdot C_i}{deg(i)} \tag{7.1}$$

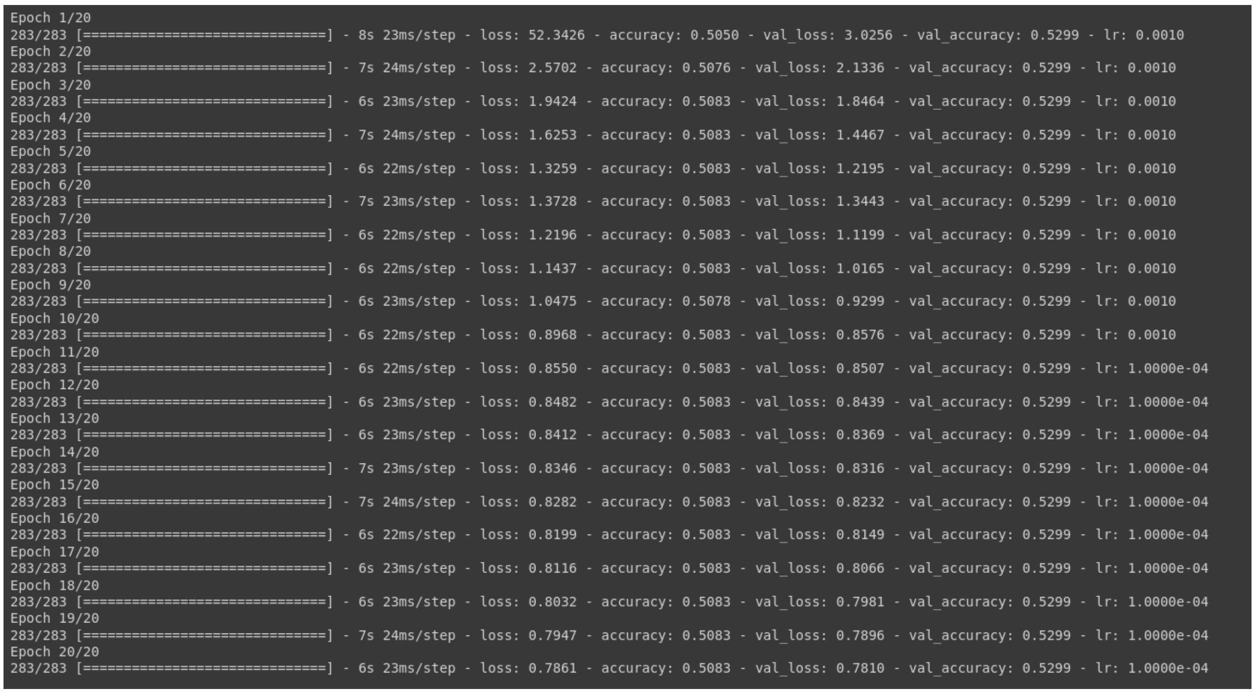
In this formula:

- $N(x)$ represents the neighborhood of node x.
- $\mu_A(i)$ is the fuzzy membership value of node i.
- $f(i)$ is the frequency of node i.
- C_i is the local clustering coefficient of node i.
- $deg(i)$ is the degree of node i.

7.3 Models

7.3.1 CNN-Base Model

We have used 5 layers which includes a Convolutional layer ('Conv2D') with 64 filters each of size (3, 3) along with ReLU activation, One MaxPooling Layer ('MaxPooling2D') with a pool size of (2,2), One Fatten layer ('Flatten') to convert previous layer output into a 1-Dimension vector.We have used two dense layers, one with 128 neurons with activation function ReLU and second one with 1 neuron and activation function sigmoid.But we don't got exceptional results after the training. Our model is not even learning the features properly.Because we have total of 5000 audios which means 5000 images(spectrograms) and which is very less to train a base model. So we planned to shift towards pretrained models.



```
Epoch 1/20
283/283 [==============================] - 8s 23ms/step - loss: 52.3426 - accuracy: 0.5050 - val_loss: 3.0256 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 2/20
283/283 [==============================] - 7s 24ms/step - loss: 2.5702 - accuracy: 0.5076 - val_loss: 2.1336 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 3/20
283/283 [==============================] - 6s 23ms/step - loss: 1.9424 - accuracy: 0.5083 - val_loss: 1.8464 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 4/20
283/283 [==============================] - 7s 24ms/step - loss: 1.6253 - accuracy: 0.5083 - val_loss: 1.4467 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 5/20
283/283 [==============================] - 6s 22ms/step - loss: 1.3259 - accuracy: 0.5083 - val_loss: 1.2195 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 6/20
283/283 [==============================] - 7s 23ms/step - loss: 1.3728 - accuracy: 0.5083 - val_loss: 1.3443 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 7/20
283/283 [==============================] - 6s 22ms/step - loss: 1.2196 - accuracy: 0.5083 - val_loss: 1.1199 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 8/20
283/283 [==============================] - 6s 22ms/step - loss: 1.1437 - accuracy: 0.5083 - val_loss: 1.0165 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 9/20
283/283 [==============================] - 6s 23ms/step - loss: 1.0475 - accuracy: 0.5078 - val_loss: 0.9299 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 10/20
283/283 [==============================] - 6s 22ms/step - loss: 0.8968 - accuracy: 0.5083 - val_loss: 0.8576 - val_accuracy: 0.5299 - lr: 0.0010
Epoch 11/20
283/283 [==============================] - 6s 22ms/step - loss: 0.8550 - accuracy: 0.5083 - val_loss: 0.8507 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 12/20
283/283 [==============================] - 6s 23ms/step - loss: 0.8482 - accuracy: 0.5083 - val_loss: 0.8439 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 13/20
283/283 [==============================] - 6s 23ms/step - loss: 0.8412 - accuracy: 0.5083 - val_loss: 0.8369 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 14/20
283/283 [==============================] - 7s 23ms/step - loss: 0.8346 - accuracy: 0.5083 - val_loss: 0.8316 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 15/20
283/283 [==============================] - 7s 24ms/step - loss: 0.8282 - accuracy: 0.5083 - val_loss: 0.8232 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 16/20
283/283 [==============================] - 6s 22ms/step - loss: 0.8199 - accuracy: 0.5083 - val_loss: 0.8149 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 17/20
283/283 [==============================] - 6s 23ms/step - loss: 0.8116 - accuracy: 0.5083 - val_loss: 0.8066 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 18/20
283/283 [==============================] - 6s 23ms/step - loss: 0.8032 - accuracy: 0.5083 - val_loss: 0.7981 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 19/20
283/283 [==============================] - 7s 24ms/step - loss: 0.7947 - accuracy: 0.5083 - val_loss: 0.7896 - val_accuracy: 0.5299 - lr: 1.0000e-04
Epoch 20/20
283/283 [==============================] - 6s 23ms/step - loss: 0.7861 - accuracy: 0.5083 - val_loss: 0.7810 - val_accuracy: 0.5299 - lr: 1.0000e-04
```

Figure 7.1: Training Accuracy of CNN

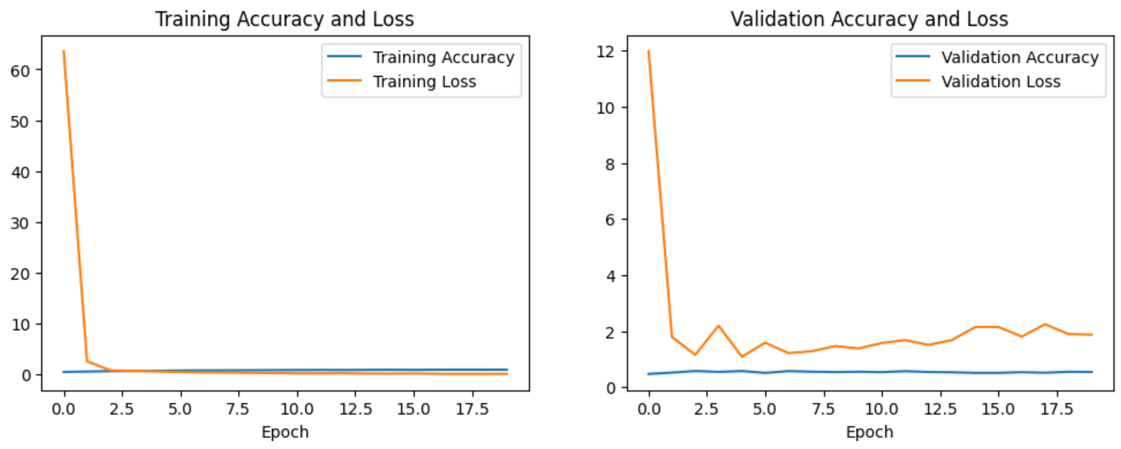


Figure 7.2: CNN Training Validation Graph

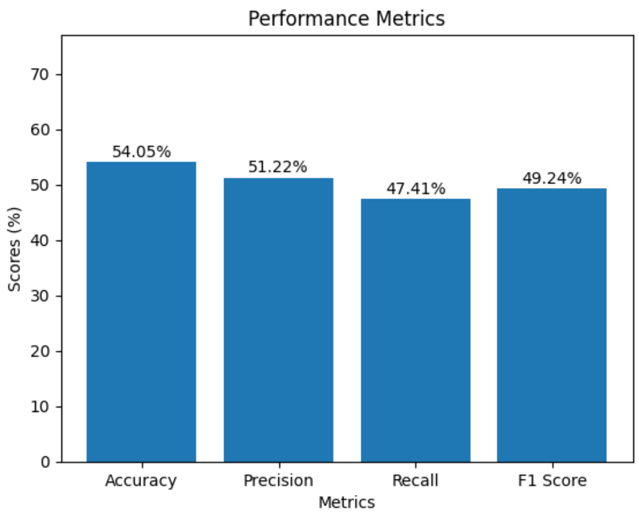
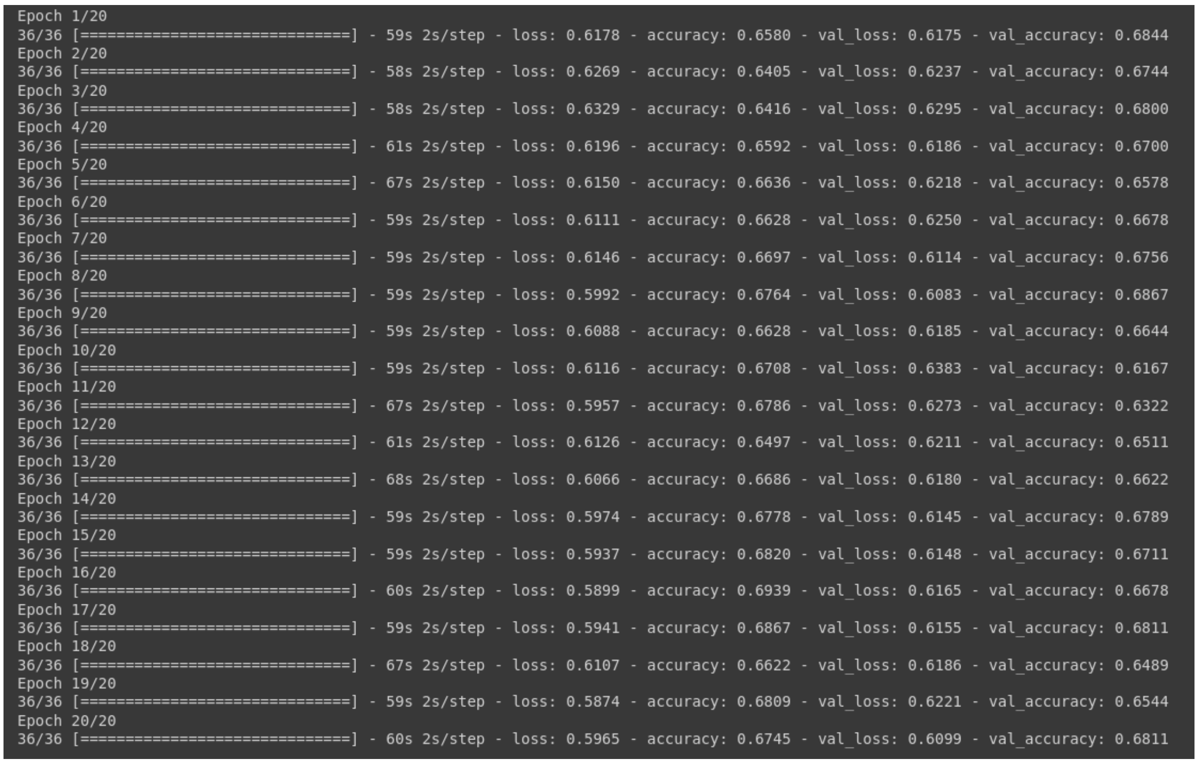


Figure 7.3: Performance Matrix of CNN

7.3.2 MobileNet V2

MobileNet-v2 is a type of deep learning model, specifically a convolutional neural network, that consists of 53 layers. You can use a pre-trained version of this network, which has been trained on over a million images from the ImageNet database. This pre-trained model is capable of categorizing images into 1000 different object classes. This model require input size of 224-by-224 so we compressed our spectrogram images to the size of (224,224). We froze all the base model layers so that the weights will not get updated while training. We have have used two layers dense layer with 256 number of neurons along with activation function ReLU and dropout layer for regularization.

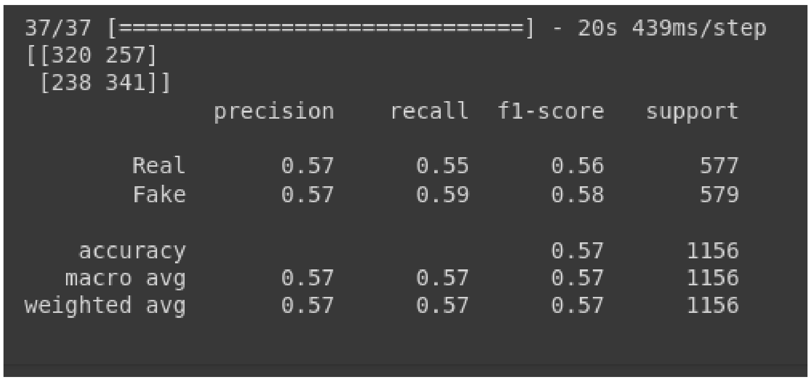
This is the result.



```
Epoch 1/20
36/36 [==============================] - 59s 2s/step - loss: 0.6178 - accuracy: 0.6580 - val_loss: 0.6175 - val_accuracy: 0.6844
Epoch 2/20
36/36 [==============================] - 58s 2s/step - loss: 0.6269 - accuracy: 0.6405 - val_loss: 0.6237 - val_accuracy: 0.6744
Epoch 3/20
36/36 [==============================] - 58s 2s/step - loss: 0.6329 - accuracy: 0.6416 - val_loss: 0.6295 - val_accuracy: 0.6800
Epoch 4/20
36/36 [==============================] - 61s 2s/step - loss: 0.6196 - accuracy: 0.6592 - val_loss: 0.6186 - val_accuracy: 0.6700
Epoch 5/20
36/36 [==============================] - 67s 2s/step - loss: 0.6150 - accuracy: 0.6636 - val_loss: 0.6218 - val_accuracy: 0.6578
Epoch 6/20
36/36 [==============================] - 59s 2s/step - loss: 0.6111 - accuracy: 0.6628 - val_loss: 0.6250 - val_accuracy: 0.6678
Epoch 7/20
36/36 [==============================] - 59s 2s/step - loss: 0.6146 - accuracy: 0.6697 - val_loss: 0.6114 - val_accuracy: 0.6756
Epoch 8/20
36/36 [==============================] - 59s 2s/step - loss: 0.5992 - accuracy: 0.6764 - val_loss: 0.6083 - val_accuracy: 0.6867
Epoch 9/20
36/36 [==============================] - 59s 2s/step - loss: 0.6088 - accuracy: 0.6628 - val_loss: 0.6185 - val_accuracy: 0.6644
Epoch 10/20
36/36 [==============================] - 59s 2s/step - loss: 0.6116 - accuracy: 0.6708 - val_loss: 0.6383 - val_accuracy: 0.6167
Epoch 11/20
36/36 [==============================] - 67s 2s/step - loss: 0.5957 - accuracy: 0.6786 - val_loss: 0.6273 - val_accuracy: 0.6322
Epoch 12/20
36/36 [==============================] - 61s 2s/step - loss: 0.6126 - accuracy: 0.6497 - val_loss: 0.6211 - val_accuracy: 0.6511
Epoch 13/20
36/36 [==============================] - 68s 2s/step - loss: 0.6066 - accuracy: 0.6686 - val_loss: 0.6180 - val_accuracy: 0.6622
Epoch 14/20
36/36 [==============================] - 59s 2s/step - loss: 0.5974 - accuracy: 0.6778 - val_loss: 0.6145 - val_accuracy: 0.6789
Epoch 15/20
36/36 [==============================] - 59s 2s/step - loss: 0.5937 - accuracy: 0.6820 - val_loss: 0.6148 - val_accuracy: 0.6711
Epoch 16/20
36/36 [==============================] - 60s 2s/step - loss: 0.5899 - accuracy: 0.6939 - val_loss: 0.6165 - val_accuracy: 0.6678
Epoch 17/20
36/36 [==============================] - 59s 2s/step - loss: 0.5941 - accuracy: 0.6867 - val_loss: 0.6155 - val_accuracy: 0.6811
Epoch 18/20
36/36 [==============================] - 67s 2s/step - loss: 0.6107 - accuracy: 0.6622 - val_loss: 0.6186 - val_accuracy: 0.6489
Epoch 19/20
36/36 [==============================] - 59s 2s/step - loss: 0.5874 - accuracy: 0.6809 - val_loss: 0.6221 - val_accuracy: 0.6544
Epoch 20/20
36/36 [==============================] - 60s 2s/step - loss: 0.5965 - accuracy: 0.6745 - val_loss: 0.6099 - val_accuracy: 0.6811
```

Figure 7.4: MobileNet V2

We have got 57% testing accuracy. 68% validation accuracy.



```
37/37 [==============================] - 20s 439ms/step
[[320 257]
 [238 341]]
              precision    recall  f1-score   support

        Real       0.57      0.55      0.56       577
        Fake       0.57      0.59      0.58       579

    accuracy                           0.57      1156
   macro avg       0.57      0.57      0.57      1156
weighted avg       0.57      0.57      0.57      1156
```

Figure 7.5: MobileNet V2

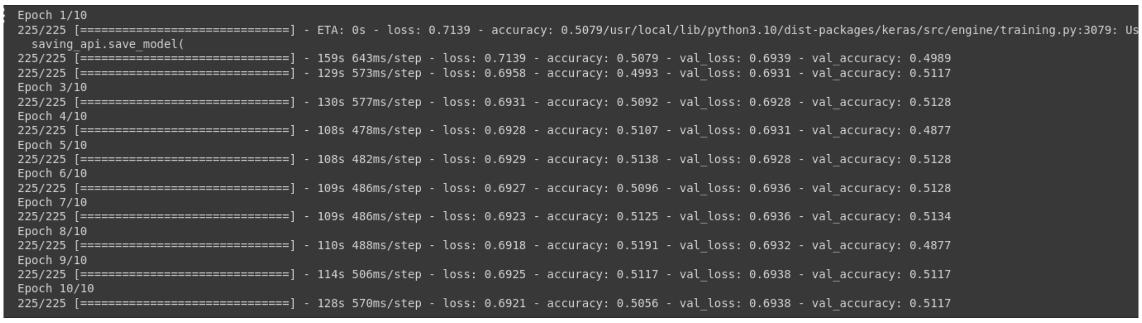
Which is just like a fluke.

7.3.3 Resnet-18

ResNet-18 is a particular type of convolutional neural network (CNN) that falls under the ResNet (Residual Network) family. ResNet-18, much like other ResNet models, is created to overcome the challenge of training extremely deep neural networks. Specifically, ResNet-18 is made up of 18 layers, and that's why it's called ResNet-18. It includes various types of layers, such as convolutional layers, batch normalization, activation functions (usually Rectified Linear Units or ReLUs), and max-pooling layers. ResNet-18 is commonly used for tasks like image classification and has shown remarkable performance on widely recognized datasets like ImageNet. We finetuned Resnet-18 on our spectrograms and we sill donot get exceptional results. These are the results of resnet 18.

7.3.4 Resnet-50

ResNet-50 is another version of the ResNet (Residual Network) design, and it's more extensive than ResNet-18. The "50" in ResNet-50 indicates the total number of layers in the network. Unlike earlier models, ResNet-50 is made up of 50 layers, and its architecture is more intricate. It includes deeper and wider layers to capture more detailed features in the data. Training our spectrograms images in Resnet-50 also didn't give exceptional results.



```
Epoch 1/10
225/225 [==============================] - ETA: 0s - loss: 0.7139 - accuracy: 0.5079/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: Us
  saving_api.save_model(
225/225 [==============================] - 159s 643ms/step - loss: 0.7139 - accuracy: 0.5079 - val_loss: 0.6939 - val_accuracy: 0.4989
225/225 [==============================] - 129s 573ms/step - loss: 0.6958 - accuracy: 0.4993 - val_loss: 0.6931 - val_accuracy: 0.5117
Epoch 3/10
225/225 [==============================] - 130s 577ms/step - loss: 0.6931 - accuracy: 0.5092 - val_loss: 0.6928 - val_accuracy: 0.5128
Epoch 4/10
225/225 [==============================] - 108s 478ms/step - loss: 0.6928 - accuracy: 0.5107 - val_loss: 0.6931 - val_accuracy: 0.4877
Epoch 5/10
225/225 [==============================] - 108s 482ms/step - loss: 0.6929 - accuracy: 0.5138 - val_loss: 0.6928 - val_accuracy: 0.5128
Epoch 6/10
225/225 [==============================] - 109s 486ms/step - loss: 0.6927 - accuracy: 0.5096 - val_loss: 0.6936 - val_accuracy: 0.5128
Epoch 7/10
225/225 [==============================] - 109s 486ms/step - loss: 0.6923 - accuracy: 0.5125 - val_loss: 0.6936 - val_accuracy: 0.5134
Epoch 8/10
225/225 [==============================] - 110s 488ms/step - loss: 0.6918 - accuracy: 0.5191 - val_loss: 0.6932 - val_accuracy: 0.4877
Epoch 9/10
225/225 [==============================] - 114s 506ms/step - loss: 0.6925 - accuracy: 0.5117 - val_loss: 0.6938 - val_accuracy: 0.5117
Epoch 10/10
225/225 [==============================] - 128s 570ms/step - loss: 0.6921 - accuracy: 0.5056 - val_loss: 0.6938 - val_accuracy: 0.5117
```

Figure 7.6: Resnet 50 Training Accuracy on spectrograms

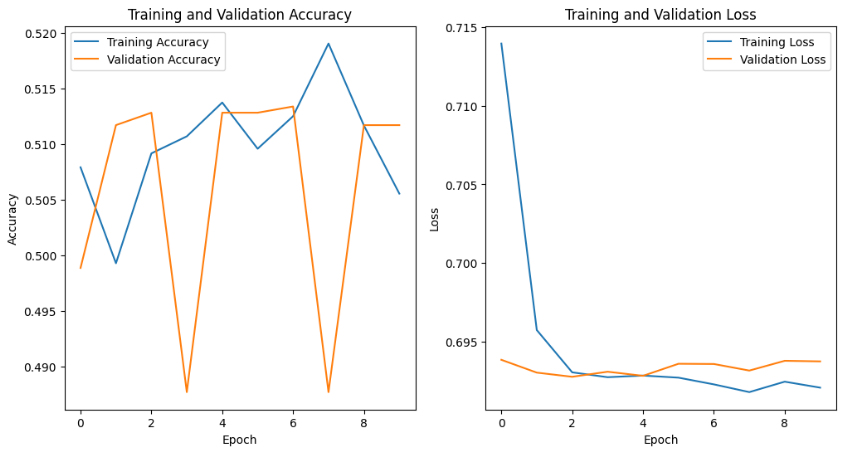


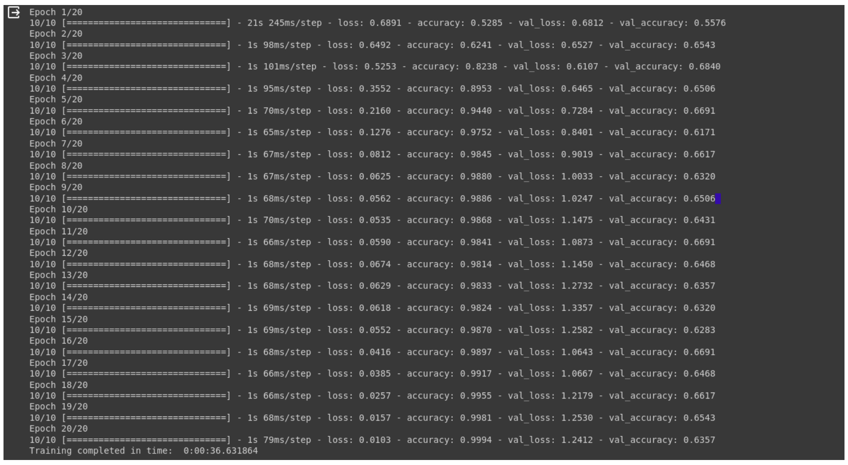
Figure 7.7: Resnet 50 Training Validation Graph

The reason could be the models are not trained on such types of images dataset. So we decided to change our preprocessing.Instead of creating spectrograms for the audio, we thought of a better approach: extracting Mel-frequency cepstral coefficients (MFCCs) as features. MFCCs are widely used in speech and audio processing to represent the short-term power spectrum of a sound signal. We first calculate the number of frames in each audio clips of 10 seconds then we calculate the number of Mel filters to be used in a MFCC calculation. We ensured that the number of filters are at most 40. We than extracts the MFCCs from the audios separately and ensuring the fixed number of frames we used padding. and stored it in a csv file considering their original shape doesn't change. After that we splitted our dataset into training, validation and testing with the ratio of 80-20. With 80% testing, and 20% is divided into training and testing.

We first tried training a base-CNN model in order to check whether the model is learning the features or not. While training models in spectrogram we found out that model is not even learning the features properly. Model is not being able to properly distinguish between Real and Fake audios spectrograms. So in order to test whether model is learning the features or not we tested on a base CNN model.

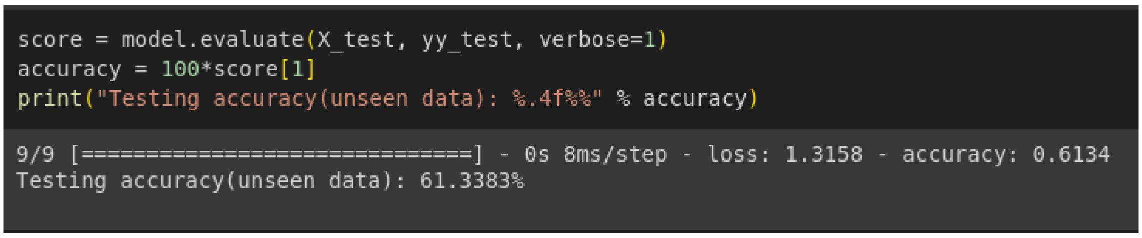
7.3.5 CNN-base Model

Model is learning exceptionally well. The training accuracy of the model was 98% but we got 61% testing accuracy. Because of less number of dataset.



```
Epoch 1/20
10/10 [==============================] - 21s 245ms/step - loss: 0.6891 - accuracy: 0.5285 - val_loss: 0.6812 - val_accuracy: 0.5576
Epoch 2/20
10/10 [==============================] - 1s 98ms/step - loss: 0.6492 - accuracy: 0.6241 - val_loss: 0.6527 - val_accuracy: 0.6543
Epoch 3/20
10/10 [==============================] - 1s 101ms/step - loss: 0.5253 - accuracy: 0.8238 - val_loss: 0.6107 - val_accuracy: 0.6840
Epoch 4/20
10/10 [==============================] - 1s 95ms/step - loss: 0.3552 - accuracy: 0.8953 - val_loss: 0.6465 - val_accuracy: 0.6506
Epoch 5/20
10/10 [==============================] - 1s 70ms/step - loss: 0.2160 - accuracy: 0.9440 - val_loss: 0.7284 - val_accuracy: 0.6691
Epoch 6/20
10/10 [==============================] - 1s 65ms/step - loss: 0.1276 - accuracy: 0.9752 - val_loss: 0.8401 - val_accuracy: 0.6171
Epoch 7/20
10/10 [==============================] - 1s 67ms/step - loss: 0.0812 - accuracy: 0.9845 - val_loss: 0.9019 - val_accuracy: 0.6617
Epoch 8/20
10/10 [==============================] - 1s 67ms/step - loss: 0.0625 - accuracy: 0.9880 - val_loss: 1.0033 - val_accuracy: 0.6320
Epoch 9/20
10/10 [==============================] - 1s 68ms/step - loss: 0.0562 - accuracy: 0.9886 - val_loss: 1.0247 - val_accuracy: 0.6506
Epoch 10/20
10/10 [==============================] - 1s 70ms/step - loss: 0.0535 - accuracy: 0.9868 - val_loss: 1.1475 - val_accuracy: 0.6431
Epoch 11/20
10/10 [==============================] - 1s 66ms/step - loss: 0.0590 - accuracy: 0.9841 - val_loss: 1.0873 - val_accuracy: 0.6691
Epoch 12/20
10/10 [==============================] - 1s 68ms/step - loss: 0.0674 - accuracy: 0.9814 - val_loss: 1.1450 - val_accuracy: 0.6468
Epoch 13/20
10/10 [==============================] - 1s 68ms/step - loss: 0.0629 - accuracy: 0.9833 - val_loss: 1.2732 - val_accuracy: 0.6357
Epoch 14/20
10/10 [==============================] - 1s 69ms/step - loss: 0.0618 - accuracy: 0.9824 - val_loss: 1.3357 - val_accuracy: 0.6320
Epoch 15/20
10/10 [==============================] - 1s 69ms/step - loss: 0.0552 - accuracy: 0.9870 - val_loss: 1.2582 - val_accuracy: 0.6283
Epoch 16/20
10/10 [==============================] - 1s 68ms/step - loss: 0.0416 - accuracy: 0.9897 - val_loss: 1.0643 - val_accuracy: 0.6691
Epoch 17/20
10/10 [==============================] - 1s 66ms/step - loss: 0.0385 - accuracy: 0.9917 - val_loss: 1.0667 - val_accuracy: 0.6468
Epoch 18/20
10/10 [==============================] - 1s 66ms/step - loss: 0.0257 - accuracy: 0.9955 - val_loss: 1.2179 - val_accuracy: 0.6617
Epoch 19/20
10/10 [==============================] - 1s 68ms/step - loss: 0.0157 - accuracy: 0.9981 - val_loss: 1.2530 - val_accuracy: 0.6543
Epoch 20/20
10/10 [==============================] - 1s 79ms/step - loss: 0.0103 - accuracy: 0.9994 - val_loss: 1.2412 - val_accuracy: 0.6357
Training completed in time:  0:00:36.631864
```

Figure 7.8: CNN Training Accuracy on MFCCs Features



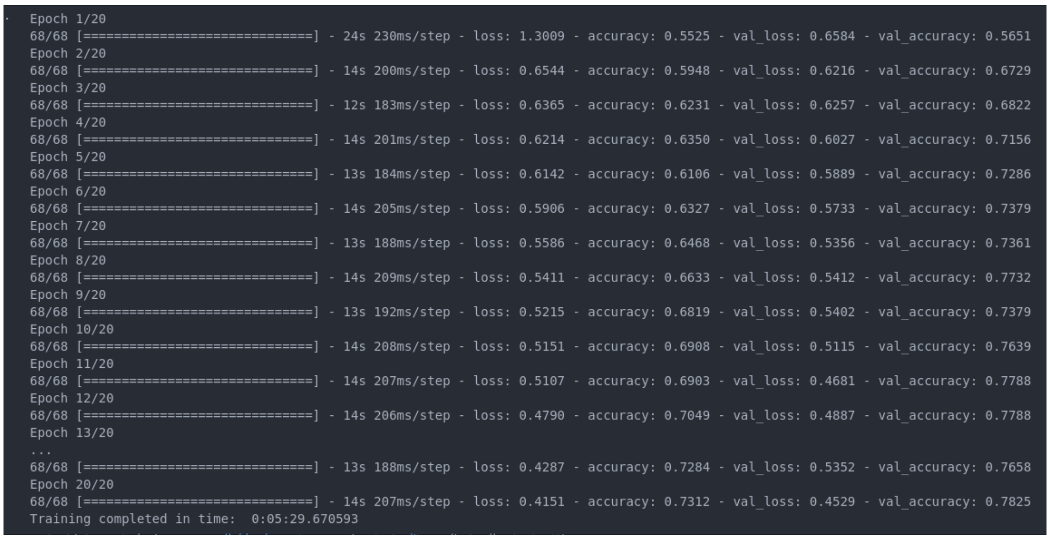
```
score = model.evaluate(X_test, yy_test, verbose=1)
accuracy = 100*score[1]
print("Testing accuracy(unseen data): %.4f%%" % accuracy)
```

```
9/9 [==============================] - 0s 8ms/step - loss: 1.3158 - accuracy: 0.6134
Testing accuracy(unseen data): 61.3383%
```

Figure 7.9: CNN Testing Accuracy on MFCCs Features

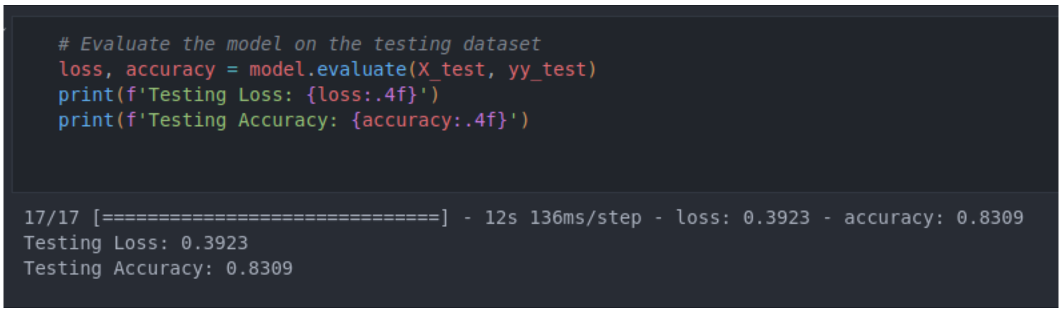
7.3.6 Resnet-50

We then used Resnet-50 to increase the accuracy of our model. We used 3 layers Flatten, Dense layer with 256 neurons along with activation function ReLU and Dropout layer with rate of 0.5. We first run the training for 10 epochs with the batch size of 32 and we got 72% accuracy. After that we increase the number of epochs and batch size to 64. We got accuracy training accuracy of 73%, validation accuracy of 78% and testing accuracy of 83%.



```
Epoch 1/20
68/68 [==============================] - 24s 230ms/step - loss: 1.3009 - accuracy: 0.5525 - val_loss: 0.6584 - val_accuracy: 0.5651
Epoch 2/20
68/68 [==============================] - 14s 200ms/step - loss: 0.6544 - accuracy: 0.5948 - val_loss: 0.6216 - val_accuracy: 0.6729
Epoch 3/20
68/68 [==============================] - 12s 183ms/step - loss: 0.6365 - accuracy: 0.6231 - val_loss: 0.6257 - val_accuracy: 0.6822
Epoch 4/20
68/68 [==============================] - 14s 201ms/step - loss: 0.6214 - accuracy: 0.6350 - val_loss: 0.6027 - val_accuracy: 0.7156
Epoch 5/20
68/68 [==============================] - 13s 184ms/step - loss: 0.6142 - accuracy: 0.6106 - val_loss: 0.5889 - val_accuracy: 0.7286
Epoch 6/20
68/68 [==============================] - 14s 205ms/step - loss: 0.5906 - accuracy: 0.6327 - val_loss: 0.5733 - val_accuracy: 0.7379
Epoch 7/20
68/68 [==============================] - 13s 188ms/step - loss: 0.5586 - accuracy: 0.6468 - val_loss: 0.5356 - val_accuracy: 0.7361
Epoch 8/20
68/68 [==============================] - 14s 209ms/step - loss: 0.5411 - accuracy: 0.6633 - val_loss: 0.5412 - val_accuracy: 0.7732
Epoch 9/20
68/68 [==============================] - 13s 192ms/step - loss: 0.5215 - accuracy: 0.6819 - val_loss: 0.5402 - val_accuracy: 0.7379
Epoch 10/20
68/68 [==============================] - 14s 208ms/step - loss: 0.5151 - accuracy: 0.6908 - val_loss: 0.5115 - val_accuracy: 0.7639
Epoch 11/20
68/68 [==============================] - 14s 207ms/step - loss: 0.5107 - accuracy: 0.6903 - val_loss: 0.4681 - val_accuracy: 0.7788
Epoch 12/20
68/68 [==============================] - 14s 206ms/step - loss: 0.4790 - accuracy: 0.7049 - val_loss: 0.4887 - val_accuracy: 0.7788
Epoch 13/20
...
68/68 [==============================] - 13s 188ms/step - loss: 0.4287 - accuracy: 0.7284 - val_loss: 0.5352 - val_accuracy: 0.7658
Epoch 20/20
68/68 [==============================] - 14s 207ms/step - loss: 0.4151 - accuracy: 0.7312 - val_loss: 0.4529 - val_accuracy: 0.7825
Training completed in time:  0:05:29.670593
```

Figure 7.10: Renet 50 Training Accuracy on MFCCs Features



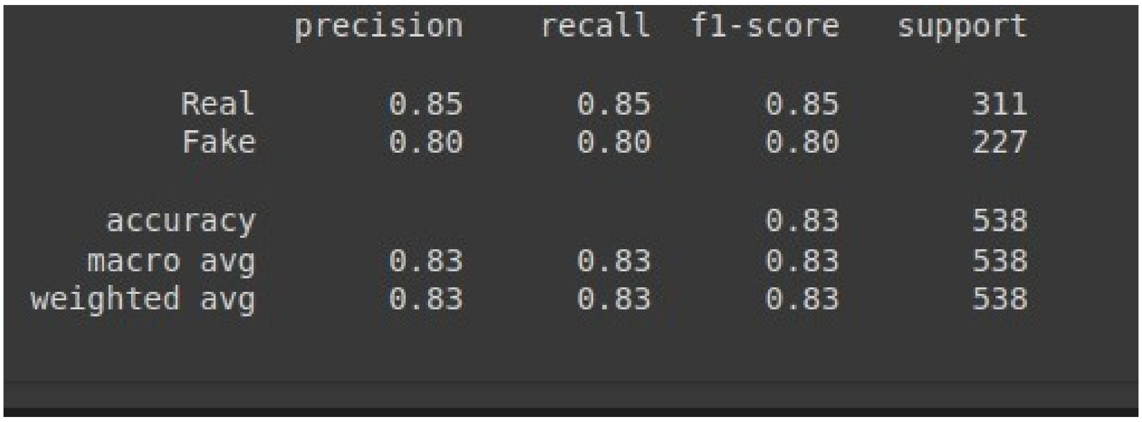
```
# Evaluate the model on the testing dataset
loss, accuracy = model.evaluate(X_test, yy_test)
print(f'Testing Loss: {loss:.4f}')
print(f'Testing Accuracy: {accuracy:.4f}')
```

```
17/17 [==============================] - 12s 136ms/step - loss: 0.3923 - accuracy: 0.8309
Testing Loss: 0.3923
Testing Accuracy: 0.8309
```

Figure 7.11: Renet 50 Testing Accuracy on MFCCs Features

7.4 Results

7.4.1 Evaluation Metrics



```
              precision    recall  f1-score   support

        Real       0.85      0.85      0.85       311
        Fake       0.80      0.80      0.80       227

    accuracy                           0.83       538
   macro avg       0.83      0.83      0.83       538
weighted avg       0.83      0.83      0.83       538
```

Figure 7.12: Evaluation Metrics Resnet50

7.4.2 Confusion Matrix

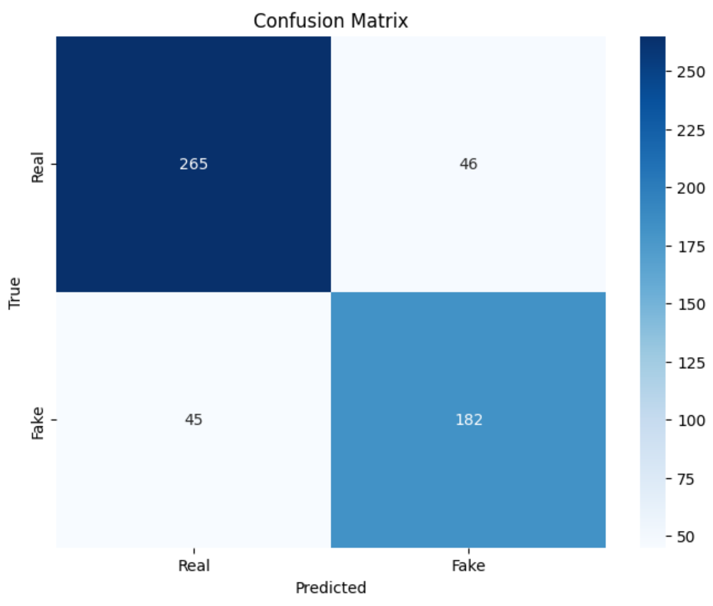


Figure 7.13: Confusion Matrix Resnet 50

Chapter 8

Conclusion and future work

8.1 Conclusion

This incredible technology is currently the coolest thing in security: it can detect bogus audio, especially clips. For people like politicians, celebrities, or influencers, this is incredibly helpful. It assists them in avoiding any misconceptions or incorrect information that the public may be spreading about them. They can utilize this to stop misinformation about them from spreading and causing misconceptions. To be honest, we came up with this notion since everyone is afraid of having untrue things published about them. As we previously stated, research is the main focus of our initiative. We can now confidently state that our deep learning model does a pretty excellent job of identifying those false audio clips. It is incredibly accurate, with 0.8309 percent. So, yeah, it's a game-changer in security!

8.2 Future Work

Urdu is widely spoken, yet there is no work available for detection of deepfake audios in this language. Research is the main focus of our project, and here's the fun part: you can really run with it! We can further improve the accuracy by training or fine-tuning on diverse amount of data. According to our experiment if we have large amount of data to train, we can simply train a base deep learning model instead of fine-tuning a model. Our experiments shows that we got exceptional training accuracy on base-CNN model. We will get extra ordinary results on training a model as well if we have more dataset. Considering our approach in this project we can also detect Synthetic audios (TTS) and voice cloning as well.

Bibliography

[1] A. Abbasi, A. R. Javed, F. Iqbal, Z. Jalil, T. R. Gadekallu, and N. Kryvinska. nine authorship identification using ensemble learning. *Scientific Reports*, 12(1):1–16, Jun 2022.

[2] S. Anwar, M. O. Beg, K. Saleem, Z. Ahmed, A. R. Javed, and U. Tariq. Social relationship analysis using state-of-the-art embeddings. *ACM Transactions on Asian and Low-Resource Language Information Processing*, Jun 2022.

[3] H. Delgado et al. Automatic speaker verification spoofing and countermeasures challenge evaluation plan. *arXiv preprint*, 2021.

[4] Hong Zhao et al. three anti-forensics of environmental-signature-based audio splicing detection and its countermeasure via rich-features classification. *IEEE Transactions on Information Forensics and Security, vol. 11, no. 7, pp*, 1603-1617, 2016.

[5] Hong Zhao et al. four audio splicing detection and localization using environmental signature. *Multimedia Tools and Applications*, 76(12):13897–13927, 2017.

[6] T. T. Nguyen et al. Deep learning for deepfakes creation and detection: A survey. *2020*, 2020.

[7] X. Wang et al. A large-scale public database of synthesized converted and replayed speech. *2019*, 2019.

[8] He K. Zhang X. Ren S. Sun J. Deep residual learning for image recognition. *IEEE conference on computer vision and pattern recognition*, 2016.

[9] A. R. Javed, Z. Jalil, W. Zehra, T. R. Gadekallu, D. Y. Suh, and M. J. Piran. seven a comprehensive survey on digital video forensics: Taxonomy challenges and future directions. *Engineering Applications of Artificial Intelligence*, 106, Nov 2021.

[10] A. R. Javed, Z. Jalil, W. Zehra, T. R. Gadekallu, D. Y. Suh, and M. J. Piran. six a comprehensive survey on digital video forensics: Taxonomy challenges and future directions. *Engineering Applications of Artificial Intelligence*, 106, Nov 2021.

[11] Da Luo, Haojun Wu, and Jiwu Huang. eight audio recapture detection using deep learning. pages 478–482, 2015.

[12] M. Imran one Z. Ali and M. Alsulaiman. An automatic digital audio authentication/forensics system. *IEEE Access, vol. 5*, pp. 2994-3007, 2017, ISSN 2169-3536.

[13] Rashmika Patole, Gunda Kore, and Priti Rege. five re-verberation based tampering detection in audio recordings. 2017.

[14] two M. Imran et al. Blind detection of copy-move forgery in digital audio forensics. *IEEE Access, vol. 5, pp. 12843-12855*, 2017, ISSN 2169-3536.