Language Identifier App

A MINI PROJECT REPORT

Submitted by

AMRUTHA RAMANAN(720818104006)
ASWIN S(720818104009)
Tamil Selvan V P(720818104058)
VIGNESH Kumar k(720818104061)

In partial fulfillment for the award of the degree
Of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

HINDUSTHAN INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2021

BONAFIDE CERTIFICATE

Certified that this project report **"Language Identifier"** is the bonafide work of **"AMRUTHA RAMANAN(720818104006), ASWIN(720818104009), TAMIL SELVAN V P (720818104058), VIGNESH KUMAR K (720818104061)"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
DR.A.JAMEER BASHA, M.TECH., Ph.D.	DR.A.JAMEER BASHA, M.TECH.,Ph.D.
HEAD OF DEPARTMENT	**SUPERVISOR**
Professor	Professor
Computer Science and Engineering,	Computer Science and Engineering,
Hindusthan Institute of Technology,	Hindusthan Institute of Technology,
Coimbatore – 641 032.	Coimbatore – 641 032.

Submitted for the University Project Viva Voce examination conducted on

………………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

Language Identification is the process of determining in which natural language the contents of the text is written. Language identification is always been an important research area which has been carried out from early 1970's. Still it is a fascinating field to be studied due to increased demand of natural language processing applications. In many applications, it works as a primary step of some larger process. In this paper, a number of applications are outlined where language identification is working successfully. Language Identification can be done using two types of techniques: computational techniques and non-computational techniques. Computational techniques are based on statistical methods and requires large set of training data for each of the language while non-computational techniques require that researcher must have extensive knowledge about the language to-be-identified.

TABLE OF CONTENTS

LIST OF FIGURES

APPENDIX II - SCREENSHOTS

CHAPTER 1
INTRODUCTION

- In recent years automatic language processing technologies have seen large improvements in terms of performance, use and acceptance. Speech recognition and speech-to-speech translation systems manifest themselves in a large variety of applications.
- In a globalizing world and growing multi-cultural societies one of the most important requirements to spoken language technology is the ability to cope with multiple languages in a robust and natural fashion. In order to achieve these objectives, technologies are required that enable the easy use of language without being impeded by language barriers.
- Today's smart systems are capable of multi-lingual speech processing, but usually it is the user's turn to decide which language pairs or pools should be worked on. Even the most sophisticated speech-to-speech and speech-to-text translation systems generally precede, besides the step of choosing one or more target languages, the explicit declaration of the source language.
- In our opinion, a highly desirable feature for intuitive and self-explanatory speech processing applications is the automation of this additional step, as it is a natural process in human-to-human interaction to identify the language, that was chosen to enter into a dialogue with, without asking. In the past decades a vivid interest grew in automatizing language identification with help of well-established speech processing technologies. The associated field of research is referred to as automatic language recognition or, more commonly, as automatic language identification (LID). The main idea is to especially utilize and exploit the sources of information, which are essential in the human language identification process.

1.1 OVERVIEW

LANGUAGE IDENTIFY USING SPEECH RHYTHM:

The conventional speech recognition systems can handle the input speech of a specific single language. To realize multi-lingual speech recognition, a language should be firstly identified from input speech where Language Identification (LID) approach for the multi-lingual system. The standard LID tasks depend on common acoustic features used in speech recognition. However, the features may convey insufficient language-specific information, as they aim to discriminate the general tendency of phonemic information. It characterises language-specific properties, considering computation complexity. We focus on speech rhythm features providing the prosodic characteristics of speech signals. The rhythm features represent the tendency of consonants and vowels of languages, and therefore, classifying them from speech signals is necessary. For the rapid classification, Gaussian Mixture Model (GMM)-based learning in which two GMMs corresponding to consonants and vowels are firstly trained and used for classifying them. By using the classification results, we estimate the tendency of two phonemic groups such as the duration of consonantal and vocalic intervals and calculate rhythm metrics called R-vector.

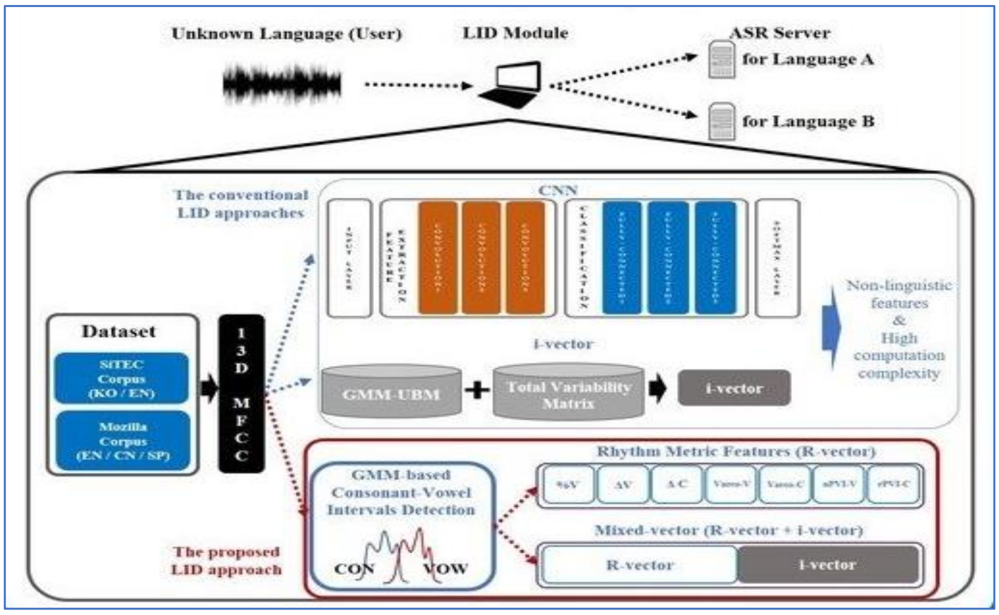

Unknown Language (User)
LID Module
ASR Server
for Language A
for Language B
The conventional LID approaches
CNN
Non-linguistic features & High computation complexity
Dataset
SiTEC Corpus (KO / EN)
Mozilla Corpus (EN / CN / SP)
13D MFCC
i-vector
GMM-UBM
Total Variability Matrix
i-vector
Rhythm Metric Features (R-vector)
%V
ΔV
ΔC
Varco-V
Varco-C
nPVI-V
rPVI-C
GMM-based Consonant-Vowel Intervals Detection
CON
VOW
Mixed-vector (R-vector + i-vector)
R-vector
i-vector
The proposed LID approach

FIG 1: SYSTEM IDEA

CHAPTER 2
LITERATURE SURVEY

An N-Gram-and-Wikipedia Joint Approach to Natural Language Identification,

by Xi Yang, Wenxin Liang

The main contributions of their work are summarized as follows:

- Effective identification of multiple languages presents in multilingual document based on their proposed approach.
- Their method can detect up to 250 languages greatly exceeding the milestone set by current practices in this area.
- Their method employs only one single language corpus. Thereby completely avoiding the need to collect multiple corpora for different languages
- They were the first researchers to bring in Wikipedia as a huge corpus in field of Natural Language Identification.

The core of the proposed N-Gram-and-Wikipedia joint approach can be divided into two distinct steps:

(1) Dividing the text into separate units based on the language they are written in using the segmentation algorithm which is based on concept of N-Gram

(2) Language of each unit is to be identified using the multiple language versions of Wikipedia articles.

The text is divided into separate units by calculating similarity of each unit with the local English corpus based on N-gram frequency statistics, and then segregating each unit accordingly.

They particularly used local English corpus as a base for the calculation of distance between every to-be-identified paragraph. And, thereby the degree of language difference between two adjacent paragraphs is calculated by finding their absolute subtraction of distance.

Larger the difference indicated the higher probability that the two are in different languages. In short, the English corpus itself was not employed to identify languages, but was thought of as a benchmark to measure and quantify the language properties of paragraphs, making mathematical comparison and the subsequent language identification between adjacent paragraphs possible.

After it, from each unit a segment of words are obtained to be used as a search keyword for domain Wikipedia.org using Google. Regularity in Wikipedia URLs made it possible to know the language used in that specific page. Hence, by studying the URL obtained language of each keyword is determined leading to determination of original text.

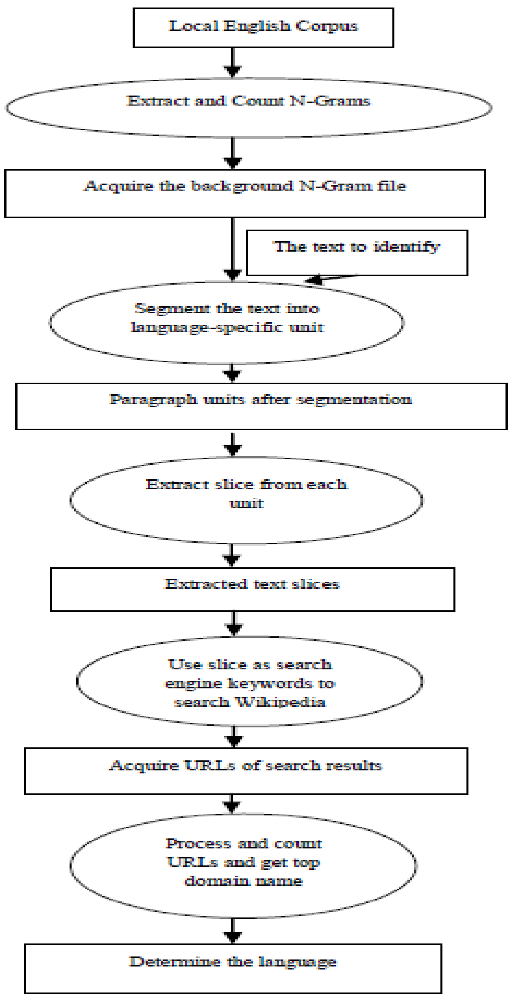

Local English Corpus
Extract and Count N-Grams
Acquire the background N-Gram file
The text to identify
Segment the text into language-specific unit
Paragraph units after segmentation
Extract slice from each unit
Extracted text slices
Use slice as search engine keywords to search Wikipedia
Acquire URLs of search results
Process and count URLs and get top domain name
Determine the language

FIG 2.1: N-gram System

CHAPTER 3
LANGUAGE IDENTIFIER

Language identifying is the first step toward achieving a variety of tasks like detecting the source language for machine translation, improving the search relevancy by personalizing the search results according to the query language providing uniform search box for a multilingual dictionary tagging data stream from Twitter with appropriate language etc.

3.1 LANGUAGE CLASSIFIER

While classifying languages belonging to disjoint groups is not hard, disambiguation of languages originating from the same source and dialects still pose a considerable challenge in the area of natural language processing. Regular classifiers based on word frequency only are inadequate in making a correct prediction for such similar languages and

utilization of state of the art machine learning tools to capture the structure of the language has become necessary to boost the classifier performance. In this work we took advantage of recent advancement of deep neural network based models showing stellar performance in many natural language processing tasks to build a state of the art language classifier. Language identification is used in many NLP tasks and in some of them simple rules2are often good enough. But for many other applications, such as web crawling, question answering or multilingual documents processing, more sophisticated approaches need to be used Language identification is used in many NLP tasks and in some of them simple rules2 are often good enough. But for many other applications, such as web crawling, question answering or multilingual documents processing, more sophisticated approaches need to be used QR code has become a focus of advertising strategy, since it provides a way to access a brand's website more quickly than by manually entering a URL.

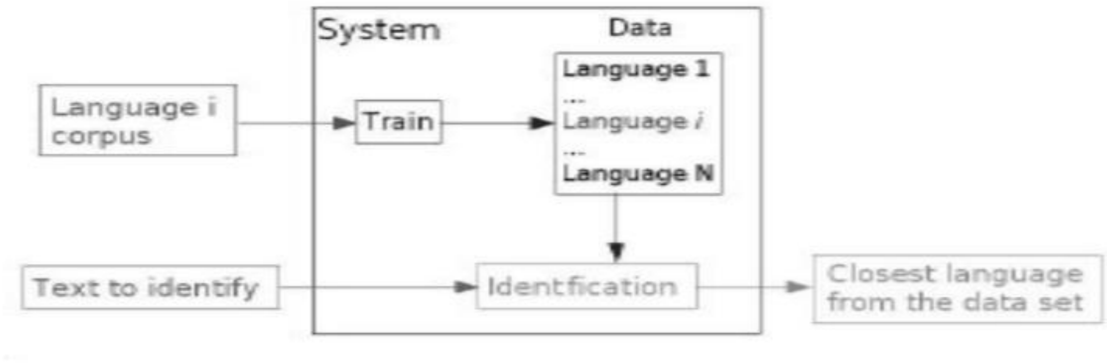


FIG 3.1: LANGUAGE ARCHITECTURE

3.2 LANGUAGE SYSTEM DESCRIPTION

The system is divided broadly into dataset formation, pre-processing, feature extraction and classification. The entire system is programmed using MATLAB R2014a. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)[4], which consisted of both male and female speech audio samples was used to test for 3 emotions- anger, happiness and sadness. Firstly, the audio database is divided into training and testing sets. Each signal from both the sets is pre-processed to make it suitable for data gathering and analysis. In succession, the features are extracted from the pre-processed signal. The feature vectors are the input to the multiclass support vector machine (SVM) classifier which forms a model corresponding to every emotion. The test signal is tested with every model in order to classify and detect its emotion.

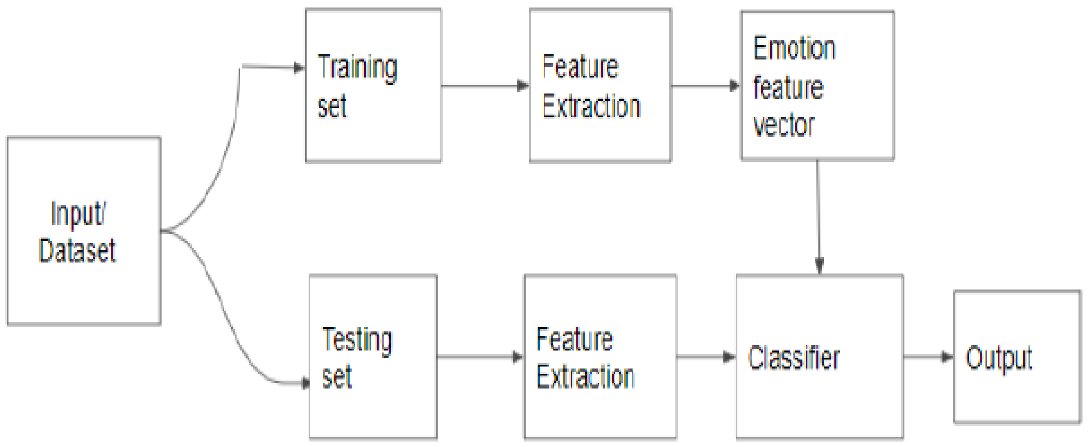


FIG 3.2: SYSTEM BLOCK DIAGRAM

CHAPTER 4

AUDIO DATASET SYSTEM

Audio signals often contain regions of absolute silence occurring at the beginning or at the end and sometimes in between higher frequencies. It is required to remove this unwanted part from the signal and hence desilencing is performed. Silence removal is performed by applying a particular threshold to the signal. We get a signal in which the unvoiced parts which do not contain any relevant data are removed and the voiced parts are retained.

4.1 DESCRIPTION OF AUDIO DATASET

In this section, we analyze main features of audio data, and the hierarchical structure of audio data .

As is shown in , definitions of audio structural units with different time granularity is proposed, and the hierarchical structure of audio data contains 1) Audio high level semantic unit, 2) Audio shot, 3) Audio clip and 4) Audio frame.

- Audio frame: Audio is a non-stationary random process, and its characteristics change with time varying. However, its change is very slow. Therefore, the audio signal can be divided into a number of short periods of processing. These short segments are generally 20-30ms (called as the audio frame) and audio frame is the smallest units in the audio processing.
- Audio clip: As the audio frame time granularity is too small and it is difficult to extract meaningful semantic contents, we should define a new audio unit with larger time granularity (named as Audio clip). Audio clip is made up of a number of frames, of which the length of time is fixed.

The characteristics of the audio segment are calculated on the basis of audio frame.

- Audio shot: This concept is generated from the video footage. As audio segment are too short, it is not suitable to be used for semantic content analysis. An audio structural unit containing the same audio class is defined as an audio shot. The audio shot is composed of several audio segments within a same class
- Audio high level semantic unit: It refers to an audio structural unit with rich semantic contents which are formed by the different combinations of the audio shots.

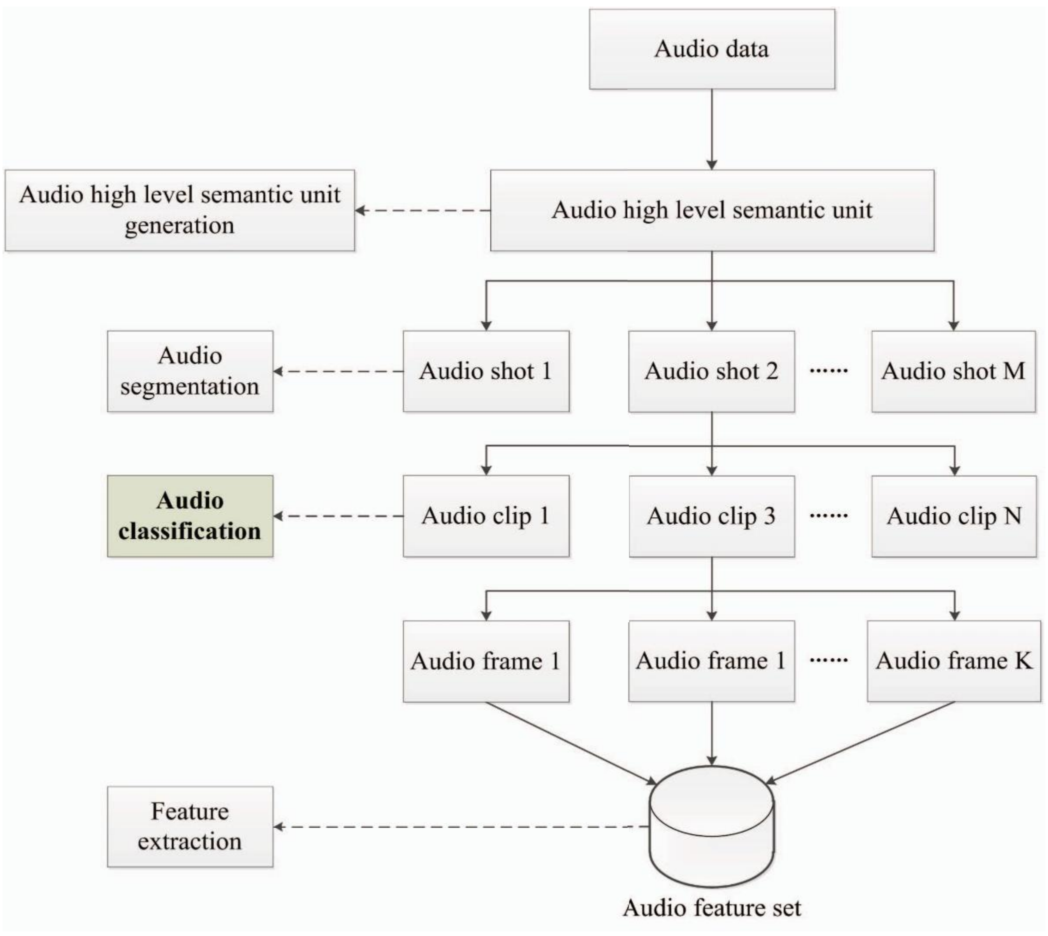


FIG 4.1: Hierarchical structure of audio data

4.2 AUDIO TRAINING DATASET

As for most deep learning problems, we will follow these steps:

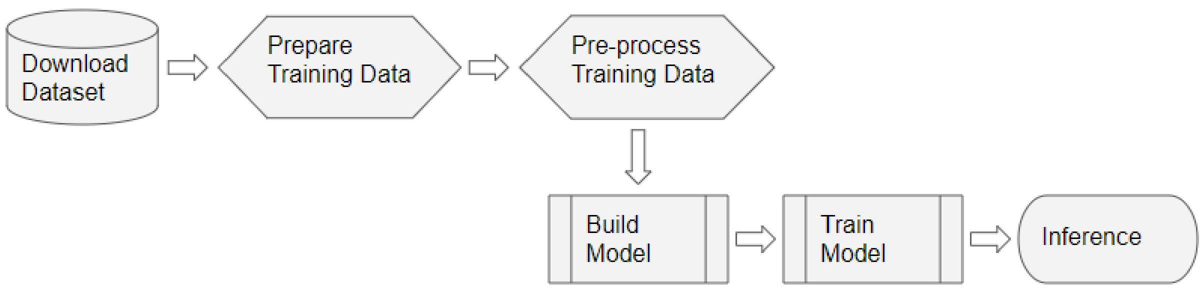
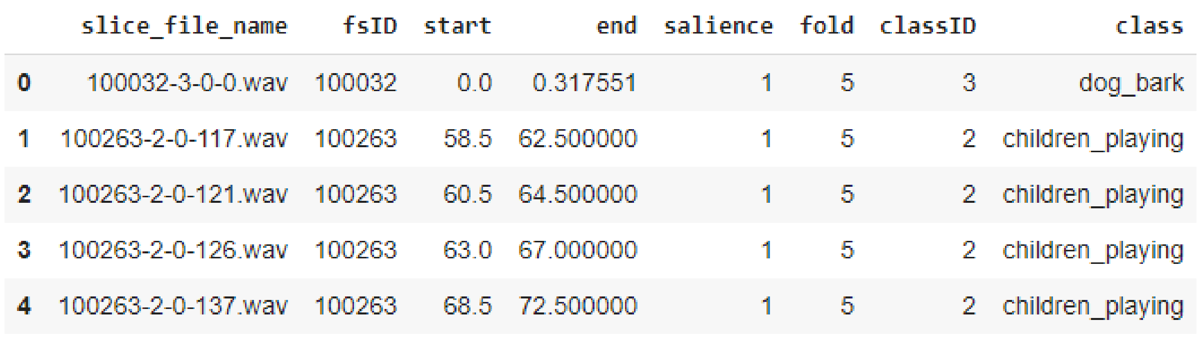


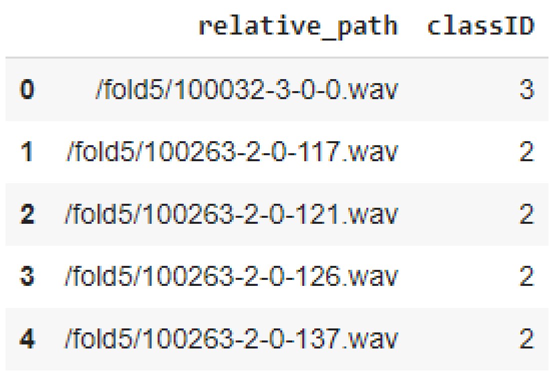
FIG 4.2: BLOCK DIAGRAM OF DATASET

Deep Learning Workflow (Image by Author)The training data for this problem will be fairly simple: The features (X) are the audio file paths The target labels (y) are the class names Since the dataset has a metadata file that contains this information already, we can use that directly. The metadata contains information about each audio file.



	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

Since it is a CSV file, we can use Pandas to read it. We can prepare the feature and label data from the meta-data. This gives us the information we need for our



	relative_path	classID
0	/fold5/100032-3-0-0.wav	3
1	/fold5/100263-2-0-117.wav	2
2	/fold5/100263-2-0-121.wav	2
3	/fold5/100263-2-0-126.wav	2
4	/fold5/100263-2-0-137.wav	2

Training data with audio file paths and class IDs

- Scan the audio file directory when metadata isn't available
- Having the metadata file made things easy for us. How would we prepare our data for datasets that do not contain a metadata file?
- Many datasets consist of only audio files arranged in a folder structure from which class labels can be derived. To prepare our training data in this format, we would do the following:

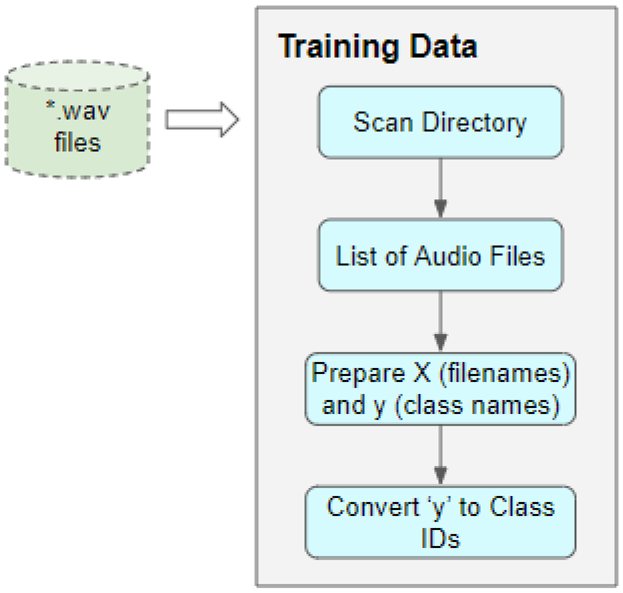


FIG 4.3: DATASET FORMAT

- Preparing Training Data when metadata isn't available (Image by Author)
- Scan the directory and prepare a list of all the audio file paths.
- Extract the class label from each file name, or from the name of the parent sub-folder
- Map each class name from text to a numeric class ID
- With or without metadata, the result would be the same — features consisting of a list of audio file names and target labels consisting of class IDs.

- *Alternative IR Model* **:**It is the enhancement of classical IR model making use of some specific techniques from some other fields. Cluster model, fuzzy model and latent semantic indexing (LSI) models are the example of alternative IR model.

CHAPTER 5

METHODOLOGY

5.1 EXTRACTION STAGES OF AUDIO

The implementation of the feature extraction stages are

A. Pre-processing

1) Sampling: Sampling is the first and important step of signal processing. Signals which we used normally, are all analogue signals i.e. continuous time signals. Therefore, for processing purpose in computer, discrete signals are better. In order to convert these continuous time signals to discrete time

signals, sampling is used.

2) Pre-emphasis: The input signal often has certain low frequency components which will result in samples similar to their adjacent samples. These parts represent a slow variation with time and hence are not of any importance while extracting signal information. Therefore, we are performing

pre-emphasizing by applying a high pass filter on the signal in order to emphasize the high frequency components which represent the rapidly changing signal. This will provide vital information. Use where H(z) denotes pre-emphasized signal.

3) De-silencing: Audio signals often contain regions of absolute silence occurring at the beginning or at the end and sometimes in between higher frequencies. It is required to remove this unwanted part from the signal and hence desilencing is performed. Silence removal is performed by applying a particular threshold to the signal. We get a signal in which the unvoiced parts which do not contain any relevant data are removed and the voiced parts are retained.

4) Framing: For the purpose of analysis, observable stationary signal is preferable. If we observe the speech signal on a short time basis, we are able to get a stationary signal. We divide the input signal into small constituent frames of a specific time interval. Generally, for speech processing, it was

observed that frame duration of 20-30 ms is implemented. This ensures two things- firstly, that considerable stationary signal value is obtained for small time duration and secondly, the signal does not undergo too much changes in the interval. We have utilized a frame duration of 25 ms.

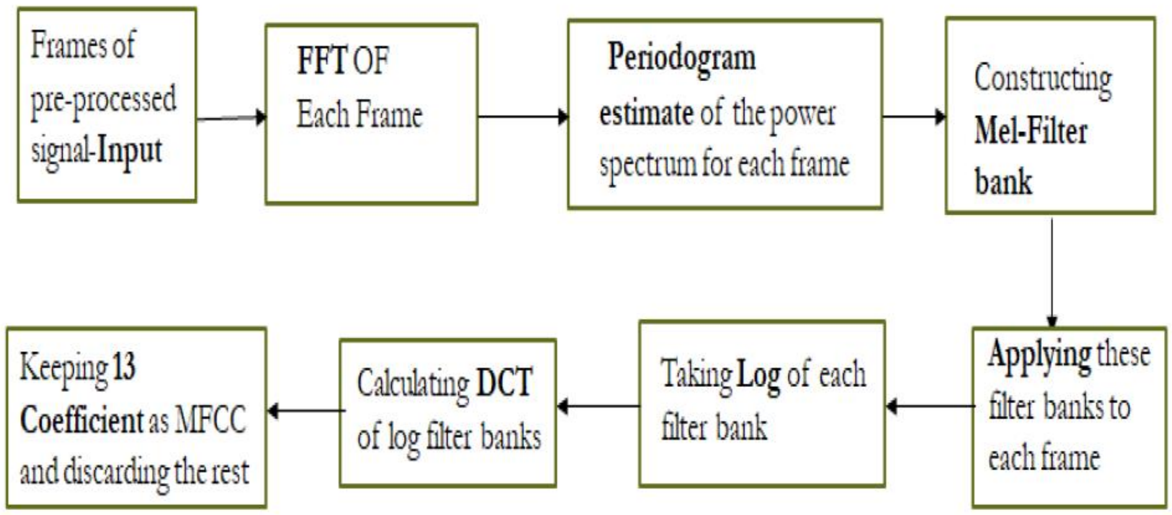
5) Windowing: Most of the digital signals are large and infinite that they cannot be analysed entirely at the same time. For better statistical calculations and processing, values of signal at each point should be available. In order to convert large digital signal into a small set for processing and analysing, and for smoothing ends of signal, windowing is performed. Different windowing techniques are available namely Rectangular, Hamming, Blackman etc. We have applied Hamming window on the framed signal. The Hamming window is represented by (3), where w(n) is windowed signal, M is the window.

5.2 FEATURE EXTRACTION

1) Energy Feature Extraction for each Frame: For the speech signal, we need to work with stationary signals therefore we calculate the energies of each frame. The energy of the signal is related to the sample value s(m) as denoted Through.

Here, s(m) represents the samples within each frame and ET represents energy of signal. A single energy value for each frame is obtained and then plotted simultaneously to obtain the short-term energy plot of the signal wherein the larger peaks represent high frequency frames.

2) MFCC feature vector extraction for each frame: MFCCs represent the short-term power spectrum envelope. This envelope in turn is representative of the shape of the human vocal tract which determines the sound characteristics. Therefore, MFCC is a vital feature for speech analysis. The block diagram of the steps to extract MFCC feature vector.



Frames of pre-processed signal-Input
FFT OF Each Frame
Periodogram estimate of the power spectrum for each frame
Constructing Mel-Filter bank
Applying these filter banks to each frame
Taking Log of each filter bank
Calculating DCT of log filter banks
Keeping 13 Coefficient as MFCC and discarding the rest

FIG 5.1: MFCC BLOCK DIAGRAM

CHAPTER 6
MACHINE LEARNING TECHNIQUES

Building machine learning models to classify, describe, or generate audio typically concerns modelling tasks where the input data are audio samples.

6.1 LANGUAGE IDENTIFICATION USING I-VECTORS

An i-Vector based LID approach was first introduced in [5, 7].In the i-Vector paradigm, a language-specific GMM mean supervector M can be represented in terms of the language and channel independent supervector m, a low rank total variability matrix T, a vector w, and a residual noise term as,

$M = m + T w \rightarrow (1)$

In (1), wis a random vector with a standard normal distribution N(0, I), and □is a residual noise term N(0,Σ). The T matrix is learned using large amounts of training data. In the LID framework, utterance-labels correspond to the language of the corresponding utterances. A comprehensive treatment of the I -Vector extraction procedure is presented .Once the i-Vectors corresponding to the training and test-sets are extracted, several approaches such as a Support Vector Machine (SVM) back-end, Gaussian back-end, or Gaussianized Cosine Distance Scoring (GCDS) can be used to determine the language of the test utterances

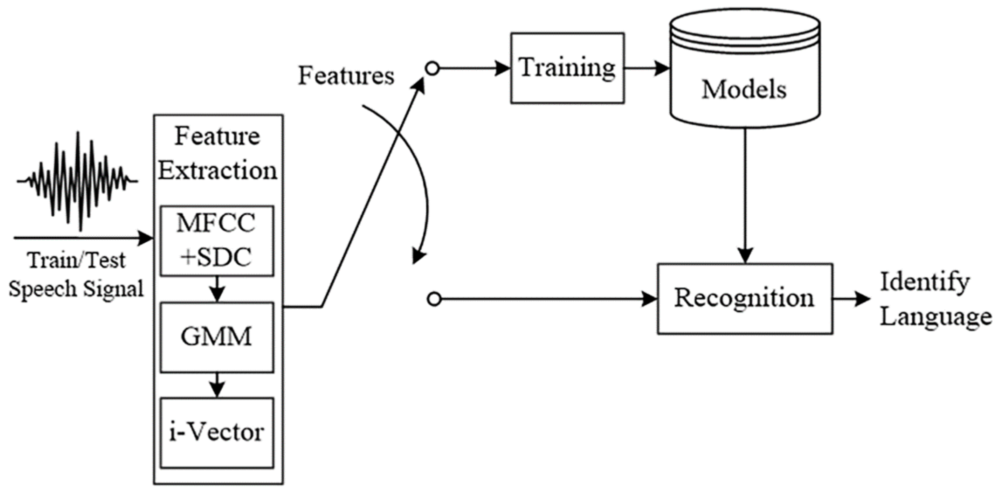


FIG 6.1: LANGUAGE IDENTIFYING SYSTEM

6.2.DEEP NEURAL NETWORK (DNN) USING I-VECTORS

Large amounts of labelled training data were used to initially train an ASR system, which was then used to generate the senone alignments to train a audio. In this framework, the CNN/DNN replaced a GMM-UBM to compute the posteriors needed for i-Vector extraction, and subsequent steps of i-Vectors based Language Identifier essentially remained unchanged. A more direct approach to Language Identifier using DNNs was outlined where the output layer nodes correspond to the in-set languages along with a single node for out-of-set languages. This approach also utilized large amounts of labelled training data and employed PLP features. Our study proposes to train a DNN Language Identifier using i-Vectors unlike existing CNN/DNN based Language techniques. To account for the out-of-set data present in the test-set, a novel 2-step DNN training strategy, where the initial DNN is trained using only in-set labelled training data. The initial DNN is then used to estimate out-of-set labels from the development data. Next, we train a second DNN for Language with both in-set and estimated out-of-set labels. Moreover, since the amount of training data is very limited, we also investigate and comment on the efficacy of some popular techniques to overcome the issue of limited training data. We have used the DNN toolkit for the experiments reported.

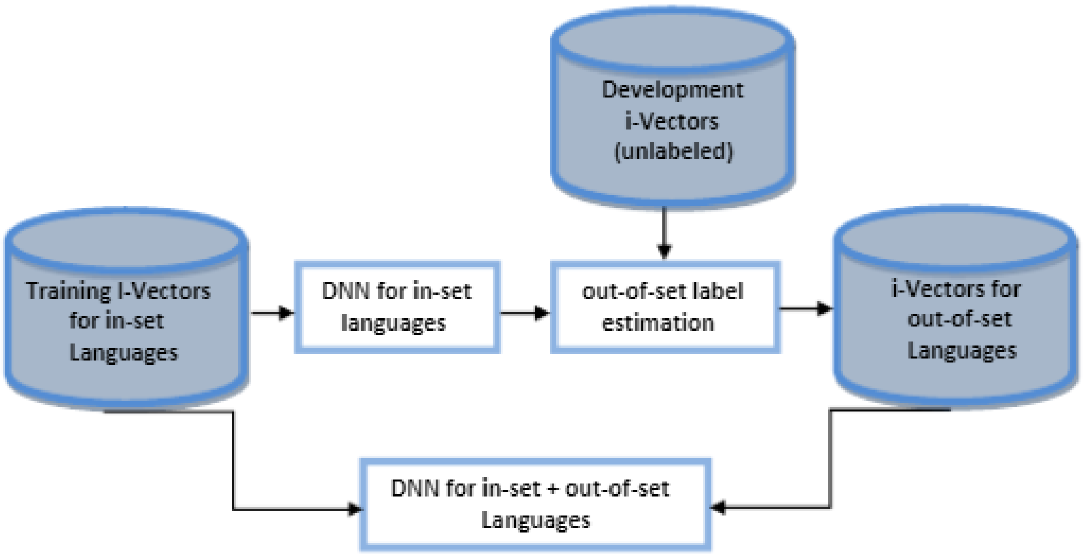

Development i-Vectors (unlabeled)
Training I-Vectors for in-set Languages
DNN for in-set languages
out-of-set label estimation
i-Vectors for out-of-set Languages
DNN for in-set + out-of-set Languages

FIG 6.2: DNN TRAINING USING I-VECTORS

6.3.TERMINOLOGY

6.3.1 SAMPLING AND SAMPLING FREQUENCY

In signal processing, sampling is the reduction of a continuous signal into a series of discrete values. The sampling frequency or rate is the number of samples taken over some fixed amount of time. A high sampling frequency results in less information loss but higher computational expense, and low sampling frequencies have higher information loss but are fast and cheap to compute.

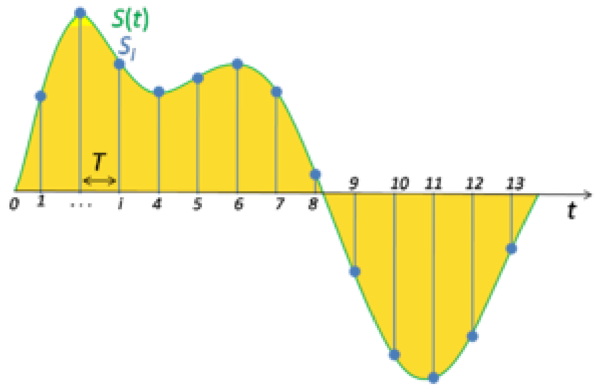


FIG 6.3.1. SAMPLING AND SAMPLING FREQUENCY

6.3.2. AMPLITUDE

The **amplitude** of a sound wave is a measure of its change over a period (usually of time). Another common definition of amplitude is a function of the magnitude of the difference between a variable's extreme values.

6.3.3. FOURIER TRANSFORM

$$\mathbf{STFT}\{x(t)\}(\tau,\omega) \equiv X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-j\omega t}\,dt$$

The Fourier Transform decomposes a function of time (signal) into constituent frequencies. In the same way a musical chord can be expressed by the volumes and frequencies of its constituent notes, a Fourier Transform of a function displays the amplitude (amount) of each frequency present in the underlying function (signal).

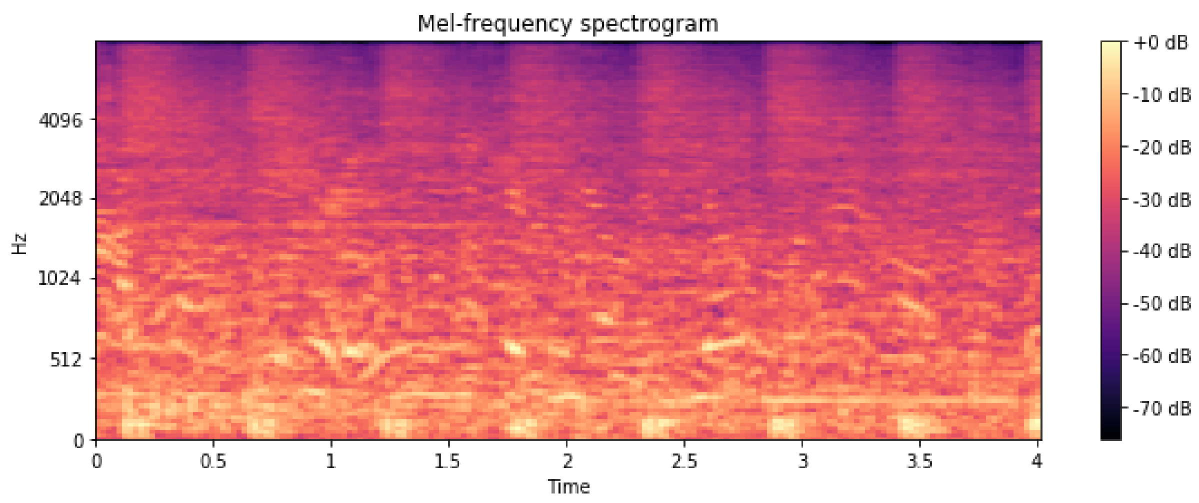


FIG 6.3.3. THE FOURIER TRANSFORM OF THE SIGNAL

6.3.4. STECTROGRAM

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. A nice way to think about spectrograms is as a stacked view of periodograms across some time-interval digital signal.

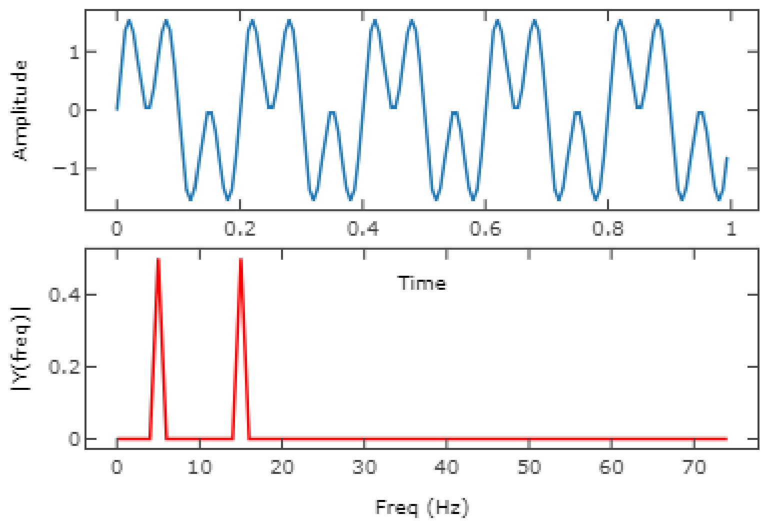


FIG 6.3.4. MEL-FREQUENCY SPECTROGRAM

CHAPTER 7

MODULES

7.1 TRAINING MODULE

- Each human voice and speech pattern is unique. They differ in intonation, pace, pronunciation and dialect. These factors complicate the development of automated speech recognition systems.
- A reliable speech recognition system must be trained using a high volume of high-quality speech recordings and developed by a diverse group of individuals to cover the range of human language nuances and, as such, be capable of performing the correct actions.
- Building machine learning models to classify, describe, or generate audio typically concerns modeling tasks where the input data are audio samples. These audio samples are usually represented as time series, where the y-axis measurement is the amplitude of the waveform.
- Original signal goes through the microphone, which can be just simplified or approximated as a bandpass filter. The recorded signal, at this moment, will broke into three different procedures simultaneously. All the details of the acoustics will be tricky, but there are a few keywords that will be used when you want to mimic the real world so that you can prepare the right dataset. Really common case people think of is mostly home or

office, in those indoors there are still lots of different noises, there could be kitchen noise, you might be vacuuming, there's outside noise from the street.

- A typical audio processing process involves the extraction of acoustics features relevant to the task at hand, followed by decision-making schemes that involve detection, classification, and knowledge fusion.

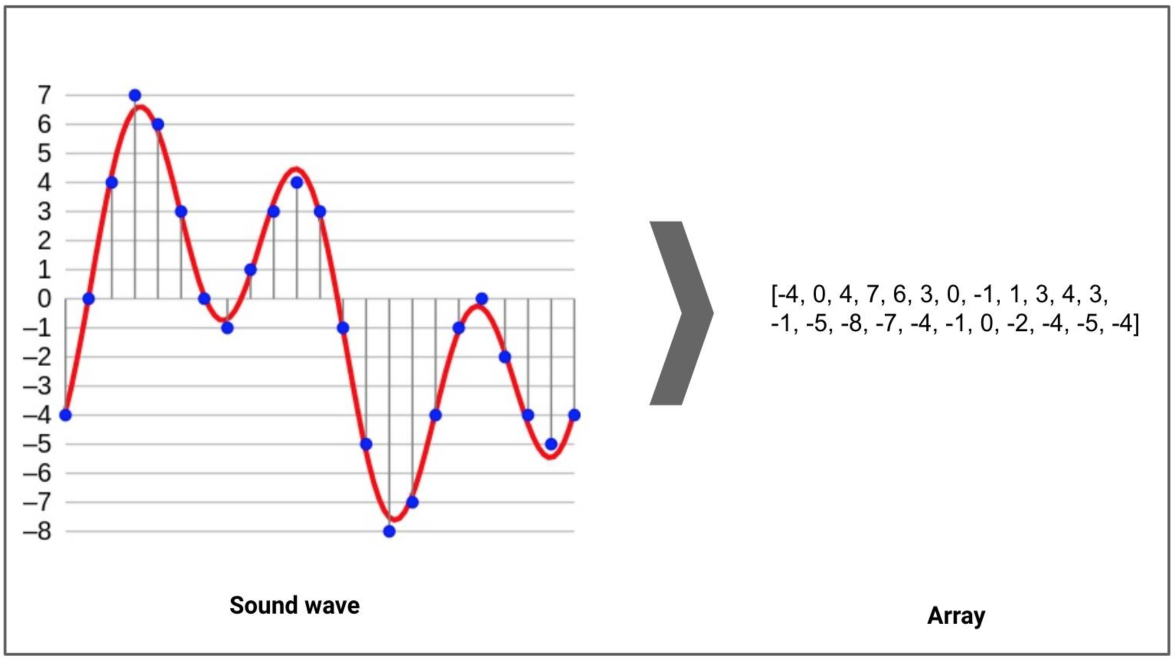


FIG 7.1. SOUND WAVE TO DIGITAL CONVERSION USING ARRAY

7.2 PACKAGES

7.2.1.**TENSORFLOW**

- TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

- TensorFlow is a symbolic math library based on dataflow and differentiable programming. It is used for both research and production at Google.

- TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 in 2015.

- It enables you to build dataflow graphs and structures to define how data moves through a graph by taking inputs as a multi-dimensional array called Tensor. It allows you to construct a flowchart of operations that can be performed on these inputs, which goes at one end and comes at the other end as output.

- It takes input as a multi-dimensional array, also known as tensors, to construct a sort of flowchart of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

- The audio file will initially be read as a binary file, which you'll want to convert into a numerical tensor.

- To load an audio file, you will use tf.audio.decode wav, which returns the WAV-encoded audio as a Tensor and the sample rate.

- A WAV file contains time series data with a set number of samples per second. Each sample represents the amplitude of the audio signal at that specific time. In a 16-bit system, like the files in mini speech commands, the values range from -32768 to 32767. The sample rate for this dataset is 16kHz.

7.2.1.1.KERAS IN TENSORFLOW

- TensorFlow 2 is an end-to-end, open-source machine learning platform. You can think of it as an infrastructure layer for differentiable programming. It combines four key abilities.

- Efficiently executing low-level tensor operations on CPU, GPU, or TPU.

- Computing the gradient of arbitrary differentiable expressions.

- Scaling computation to many devices (e.g. the Summit supercomputer at Oak Ridge National Lab, which spans 27,000 GPUs).

- Exporting programs ("graphs") to external runtimes such as servers, browsers, mobile and embedded devices.
  .

- Keras is the high-level API of TensorFlow 2: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity

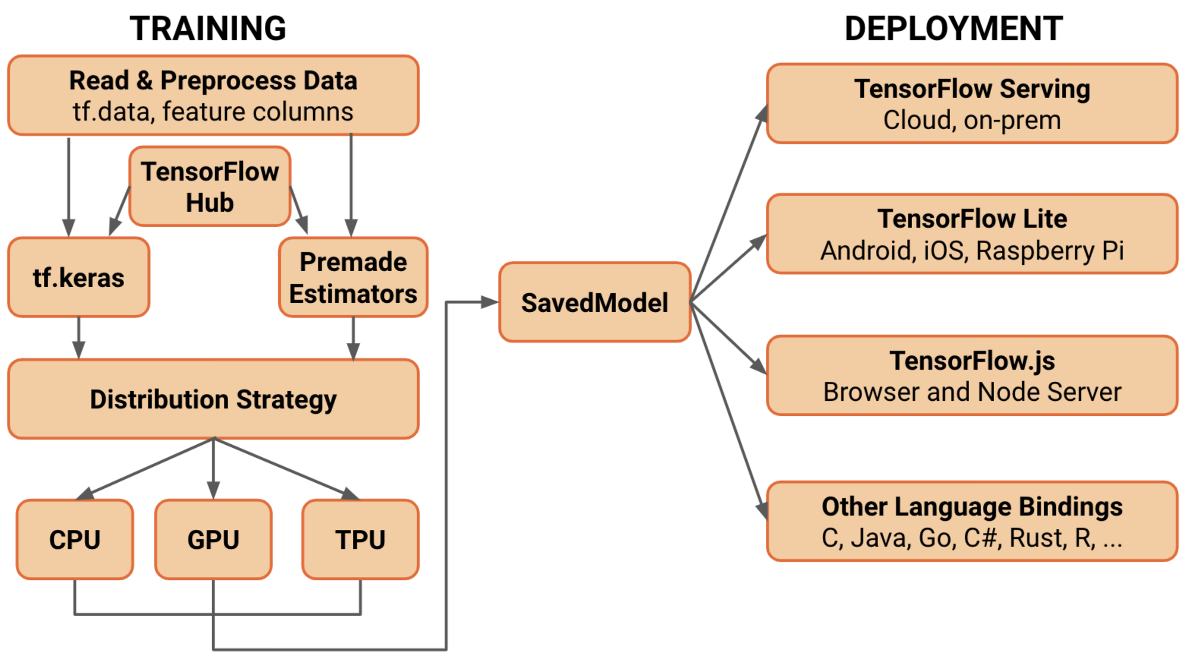


FIG 7.2.1.1. FLOW CHART OF TENSORFLOW WITH KERAS

7.2.2. PYAUDIO

PyAudio provides Python bindings for PortAudio, the cross-platform audio I/O library. With PyAudio, you can easily use Python to play and record audio on a variety of platforms.

- To use PyAudio, first instantiate PyAudio using pyaudio.PyAudio() which sets up the portaudio system.

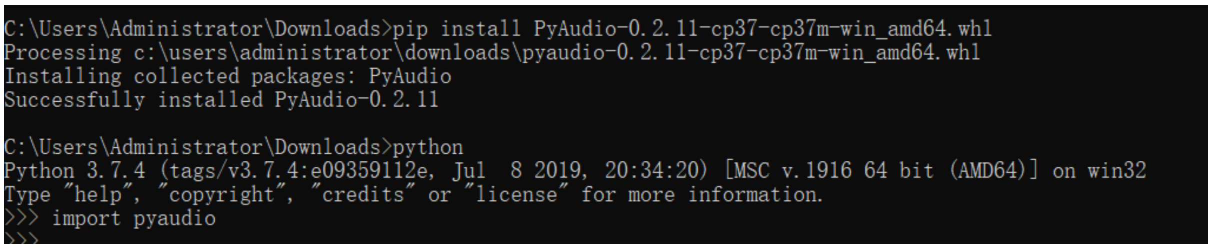
- To record or play audio, open a stream on the desired device with the desired audio parameters using pyaudio.PyAudio.open(). This sets up a pyaudio.Stream to play or record audio.

- Play audio by writing audio data to the stream using pyaudio.Stream.write(), or read audio data from the stream using pyaudio.Stream.read().

- Note that in “blocking mode”, each pyaudio.Stream.write() or pyaudio.Stream.read() blocks until all the given/requested frames have been played/recorded. Alternatively, to generate audio data on the fly or immediately process recorded audio data, use the “callback mode” outlined below.

- Use pyaudio.Stream.stop_stream() to pause playing/recording, and pyaudio.Stream.close() to terminate the stream.

- Finally, terminate the portaudio session using pyaudio.PyAudio.terminate().



```
C:\Users\Administrator\Downloads>pip install PyAudio-0.2.11-cp37-cp37m-win_amd64.whl
Processing c:\users\administrator\downloads\pyaudio-0.2.11-cp37-cp37m-win_amd64.whl
Installing collected packages: PyAudio
Successfully installed PyAudio-0.2.11

C:\Users\Administrator\Downloads>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyaudio
>>>
```

7.2.2. PYAUDIO INSTALLATION IN CMD

CHAPTER 8

ANDROID STUDIO

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA

8.1 ANDROID APP MODULES

Android Studio displays your project files in the Android project view, as shown in figure 8.1. This view is organized by modules to provide quick access to our project LANGUAGE IDENTIFIER with key source files.

All the build files are visible at the top level under Gradle Scripts and each app module contains the following folders:

- *Manifests*: Contains the AndroidManifest.xml file with user permissions.
- *Kotlin*: Contains the Kotlin source code files, including KUnit test code.
- *Res*: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation.

To see the actual file structure of the project, select Project from the Project dropdown

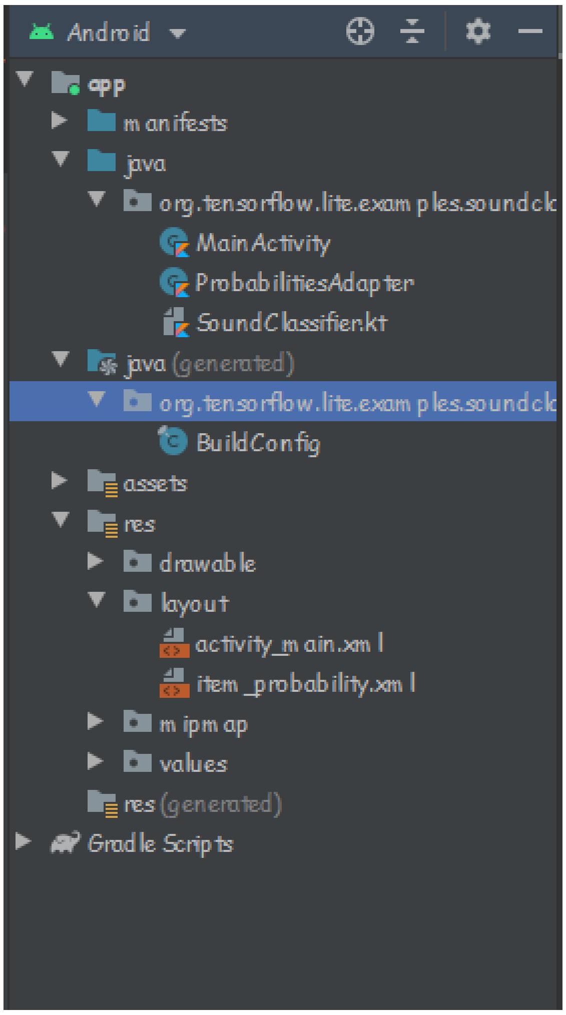


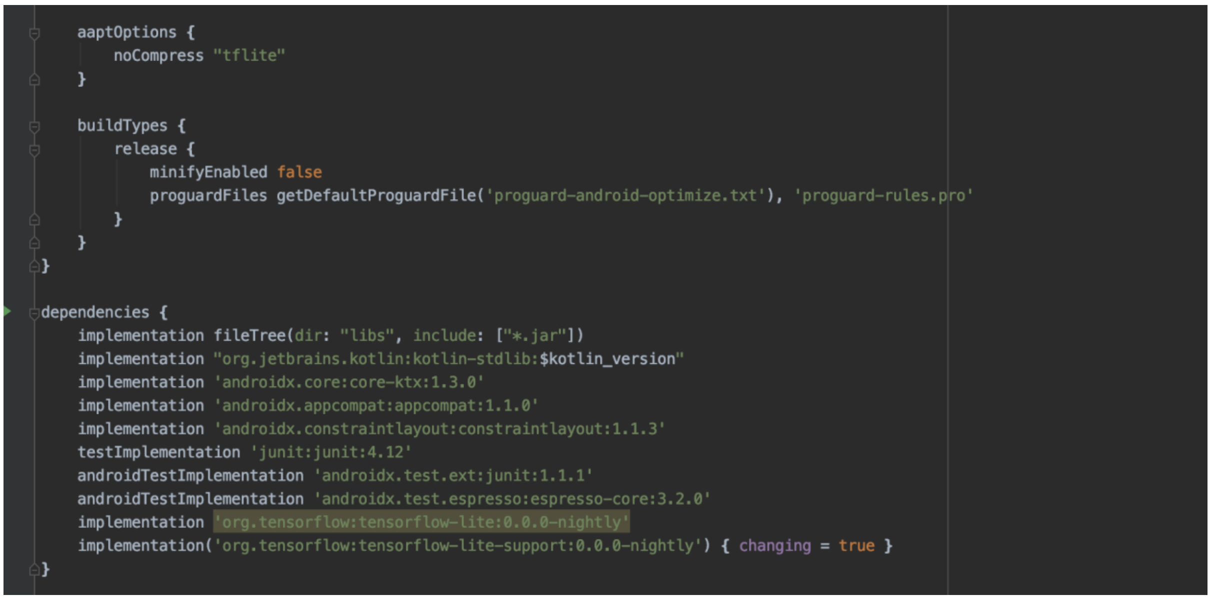
FIG 8.1. THE PROJECT FILE IN ANDROID VIEW

8.2. KOTLIN FOR AUDIO CLASSIFICATION

8.2.1. Build.gradle

Once the app is created, open the build.gradle file and mention the TensorFlow dependencies that are required to be bundled as part of the app. There are two specific TensorFlow libraries: tensorflow-lite and tensorflow-lite-support.

Also, we need to enable the setting that does not compress the tflite model during app creation through aaptOptions— as shown below:

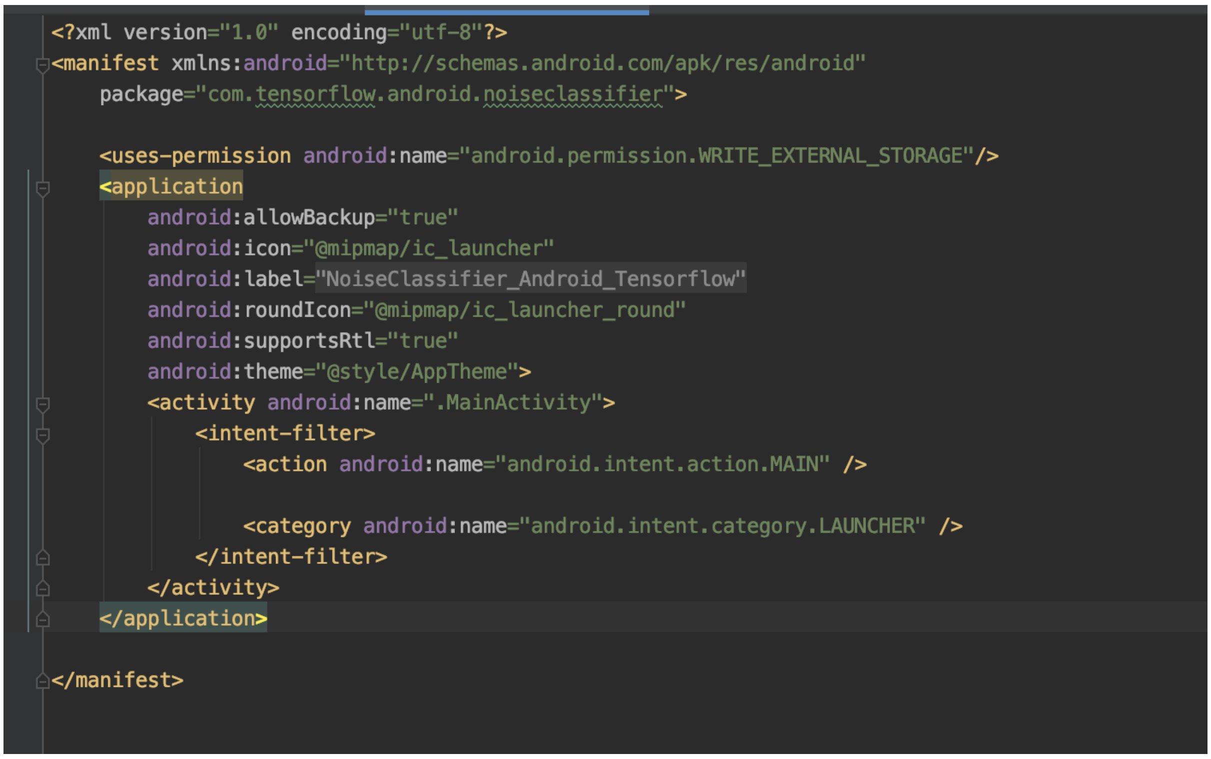


```
    aaptOptions {
        noCompress "tflite"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.3.0'
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'org.tensorflow:tensorflow-lite:0.0.0-nightly'
    implementation('org.tensorflow:tensorflow-lite-support:0.0.0-nightly') { changing = true }
}
```

Fig 8.2.1 Build.grade file

8.2.2. AndroidManifest.xml

Make sure to mention the write permission on the app with uses-permission in the AndroidManifest.xml file:



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tensorflow.android.noiseclassifier">

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="NoiseClassifier_Android_Tensorflow"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

FIG 8.2.2. AndroidManifest.xml

Assets folder: Create an assets folder in your app by right clicking on your project in Android Studio and selecting ‘New → Folder → Assets Folder’. Once this assets folder is created, copy the tflite model created in the Python section along with a text file listing the labels in the Urban Sound classification dataset in .txt file format.

8.2.3. SAMPLE AUDIO FILES:

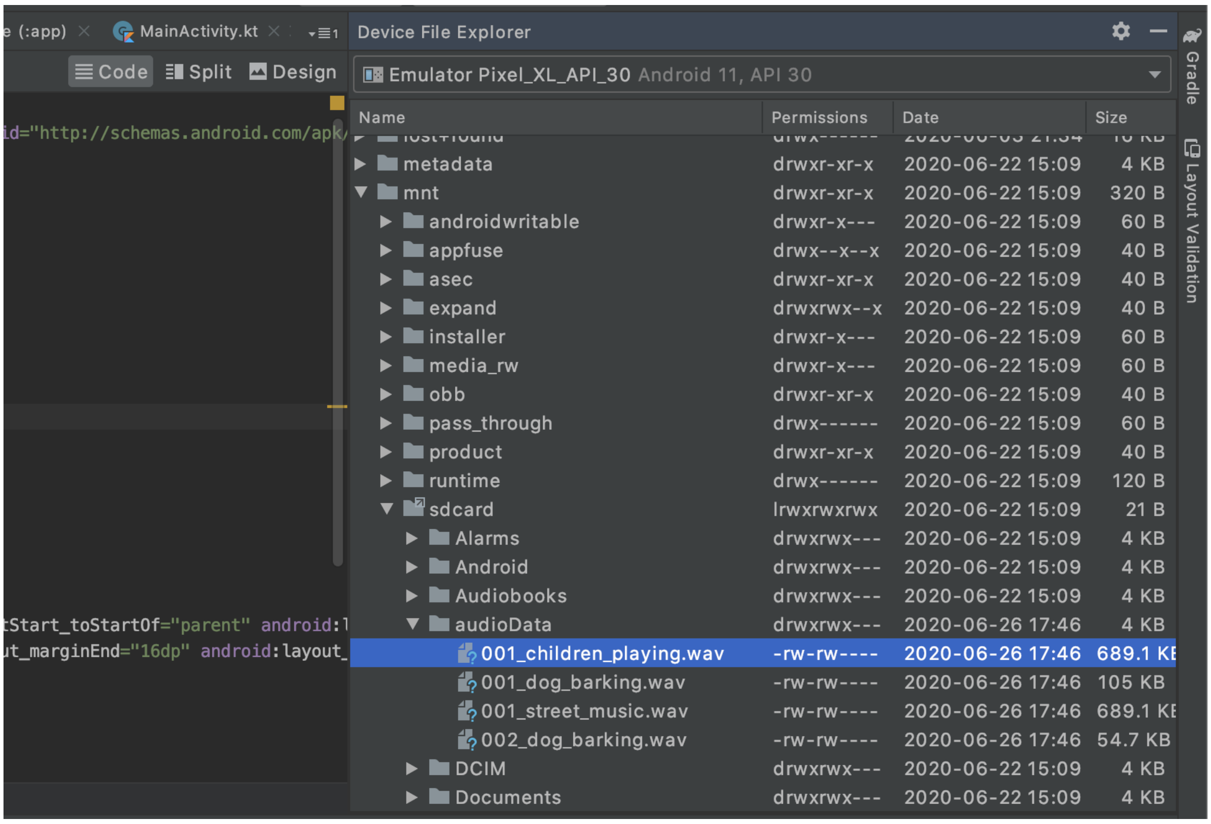
Since this demo app is about audio classification using the UrbanSound dataset, we need to copy some of the sample audio files present under the Sample Audio directory into the external storage directory of our emulator with the below steps:

i Launch the emulator.

ii Open ‘Device File Explorer’, present at the bottom right corner of Android Studio.

iii Navigate to the mnt/sdcard directory and create a directory named audioData.

iv Right click and upload the sample audio files under the audioData directory.



```
Emulator Pixel_XL_API_30 Android 11, API 30

id="http://schemas.android.com/apk/
tStart_toStartOf="parent" android:l
ut_marginEnd="16dp" android:layout_

Name                          Permissions  Date              Size
▶ metadata                    drwxr-xr-x   2020-06-22 15:09  4 KB
▼ mnt                         drwxr-xr-x   2020-06-22 15:09  320 B
  ▶ androidwritable           drwxr-x---   2020-06-22 15:09  60 B
  ▶ appfuse                   drwx--x--x   2020-06-22 15:09  40 B
  ▶ asec                      drwxr-xr-x   2020-06-22 15:09  40 B
  ▶ expand                    drwxrwx--x   2020-06-22 15:09  40 B
  ▶ installer                 drwxr-x---   2020-06-22 15:09  60 B
  ▶ media_rw                  drwxr-x---   2020-06-22 15:09  60 B
  ▶ obb                       drwxr-xr-x   2020-06-22 15:09  40 B
  ▶ pass_through              drwx------   2020-06-22 15:09  60 B
  ▶ product                   drwxr-xr-x   2020-06-22 15:09  40 B
  ▶ runtime                   drwx------   2020-06-22 15:09  120 B
  ▼ sdcard                    lrwxrwxrwx   2020-06-22 15:09  21 B
    ▶ Alarms                  drwxrwx---   2020-06-22 15:09  4 KB
    ▶ Android                 drwxrwx---   2020-06-22 15:09  4 KB
    ▶ Audiobooks              drwxrwx---   2020-06-22 15:09  4 KB
    ▼ audioData               drwxrwx---   2020-06-26 17:46  4 KB
        001_children_playing.wav  -rw-rw----  2020-06-26 17:46  689.1 KB
        001_dog_barking.wav       -rw-rw----  2020-06-26 17:46  105 KB
        001_street_music.wav      -rw-rw----  2020-06-26 17:46  689.1 KB
        002_dog_barking.wav       -rw-rw----  2020-06-26 17:46  54.7 KB
    ▶ DCIM                    drwxrwx---   2020-06-22 15:09  4 KB
    ▶ Documents               drwxrwx---   2020-06-22 15:09  4 KB
```

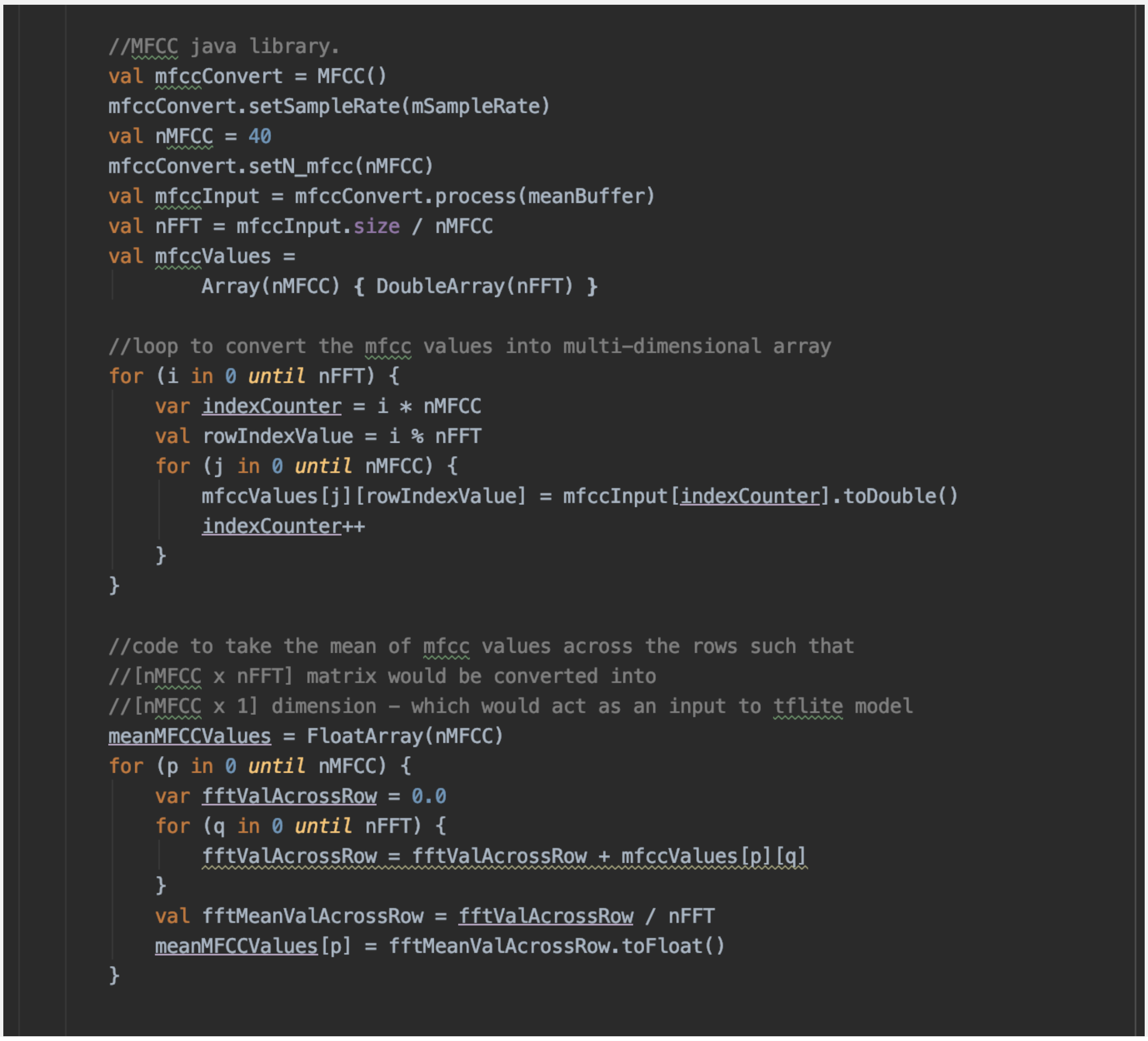
FIG 8.2.3. SAMPLE AUDIO FILES

Activity_main.xml: In the activity_main.xml file, under the res/layout directory, mention the UI controls required for the app. Here, we’ll create some very simple UI controls, with one spinner listing ‘wav’ files in the audioData directory, along with a button to classify the sounds in it.

8.2.4. MFCC GENERATION FROM AUDIO FILES

As mentioned at the beginning of this post, one of the major challenges with audio classification apps in Android is with regard to feature generation from audio files. We don’t have any readily-available library to perform this task.

Also, another critical requirement— MFCC values generated by the Java library should exactly match the corresponding python values, as only then will our model perform properly.



```
//MFCC java library.
val mfccConvert = MFCC()
mfccConvert.setSampleRate(mSampleRate)
val nMFCC = 40
mfccConvert.setN_mfcc(nMFCC)
val mfccInput = mfccConvert.process(meanBuffer)
val nFFT = mfccInput.size / nMFCC
val mfccValues =
        Array(nMFCC) { DoubleArray(nFFT) }

//loop to convert the mfcc values into multi-dimensional array
for (i in 0 until nFFT) {
    var indexCounter = i * nMFCC
    val rowIndexValue = i % nFFT
    for (j in 0 until nMFCC) {
        mfccValues[j][rowIndexValue] = mfccInput[indexCounter].toDouble()
        indexCounter++
    }
}

//code to take the mean of mfcc values across the rows such that
//[nMFCC x nFFT] matrix would be converted into
//[nMFCC x 1] dimension - which would act as an input to tflite model
meanMFCCValues = FloatArray(nMFCC)
for (p in 0 until nMFCC) {
    var fftValAcrossRow = 0.0
    for (q in 0 until nFFT) {
        fftValAcrossRow = fftValAcrossRow + mfccValues[p][q]
    }
    val fftMeanValAcrossRow = fftValAcrossRow / nFFT
    meanMFCCValues[p] = fftMeanValAcrossRow.toFloat()
}
```

FIG 8.2.4. MFCC GENERATION

TFLITE INTERPRETER:

- Ok…we’ve created an Android app, made necessary imports and configurations, processed the audio files, and generated the mean MFCC values. similar to the Python library — next, let’s feed the values to our TensorFlow Lite model and generate the predictions.
- This is where it gets interesting. Our TFLite model expects the input in the form of ‘1x40x1x1’ tensor — whereas whatever MFCC values we’ve generated are in 1D float array format. So we need to create a tensor in Java and have it fed to the model.
- TensorBuffer is the feature to use here, and you can refer to the code, where I’ve provided the detailed comments on each of the processing steps.
- Once the predictions are made, we need to process the probability values to retrieve the top predictions and to display the value.

CHAPTER 9

SYSTEM REQUIRMENTS

9.1 HARDWARE REQUIRMENTS

PROCESSOR	: Intel i7 8Gen / Ryzen 7
OPERATING SYSTEM	: Windows/Mac/Linux
RAM	: 32 GB DDR4 3200Mhz
GPU	: 8GB GDDR6
MEMORY	: 256GB SSD/1TB Hard Disk

9.2 SOFTWARE REQUIREMENTS

OPERATING SYSTEM	: Android
CODING LANGUAGE	: Kotlin & Python
IDE	: Android Studio
FRONT END	: Android SDK
BACK END	: Kotlin programming, python

9.3. SMART PHONE REQUIREMENTS:

PROCESSOR	: Snapdragon 625
RAM	: 4 GB RAM Minimum
ANDROID VERSION	: Android 6.0 Minimum
DISK SPACE	: 64GB

CHAPTER 10
CONCLUSION AND FUTURE ENHANCEMENT

10.1 CONCLUSION

This project is created for the use of company’s customer care support and it is also used for automatic speech recognition. Using this application, we can detect the Indian languages in Multilingual translation systems in smart phone. It accurately identifies the language from the given speech sample. We introduce the software that is able to detect all the Indian languages in real-world data. The app also consists of overlap factor which amplifies the audio signal. Using the application, Other Major Languages like Nepali, Manipuri, Urudhu can be identified. This application shows the probability of the language spoken by the user at the run time.

10.2 FUTURE ENHANCEMENT

In future, we can also add more amount of data set to get more accurate result of the languages and we can increase more number of languages to identify in real world data .

APPENDIX I – SOURCE CODE

1.LANGUAGE IDENTIFIER APP SOURCE CODE

MainActivity.kt

```
package org.tensorflow.lite.examples.soundclassifier

import android.Manifest
import android.content.pm.PackageManager
import android.os.Build
import android.os.Bundle
import android.util.Log
import android.view.WindowManager
import androidx.annotation.RequiresApi
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat
import
org.tensorflow.lite.examples.soundclassifier.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
  private val probabilitiesAdapter by lazy { ProbabilitiesAdapter() }

  private lateinit var soundClassifier: SoundClassifier

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
```

```
val binding = ActivityMainBinding.inflate(layoutInflater)
setContentView(binding.root)

soundClassifier = SoundClassifier(this, SoundClassifier.Options()).also {
  it.lifecycleOwner = this
}

with(binding) {
  recyclerView.apply {
    setHasFixedSize(true)
    adapter = probabilitiesAdapter.apply {
      labelList = soundClassifier.labelList
    }
  }

  keepScreenOn(inputSwitch.isChecked)
  inputSwitch.setOnCheckedChangeListener { _, isChecked ->
    soundClassifier.isPaused = !isChecked
    keepScreenOn(isChecked)
  }

  overlapFactorSlider.value = soundClassifier.overlapFactor
  overlapFactorSlider.addOnChangeListener { _, value, _ ->
    soundClassifier.overlapFactor = value
  }
}
```

```
    soundClassifier.probabilities.observe(this) { resultMap ->
      if (resultMap.isEmpty() || resultMap.size > soundClassifier.labelList.size) {
        Log.w(TAG, "Invalid size of probability output! (size: ${resultMap.size})")
        return@observe
      }
      probabilitiesAdapter.probabilityMap = resultMap
      probabilitiesAdapter.notifyDataSetChanged()
    }

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
      requestMicrophonePermission()
    } else {
      soundClassifier.start()
    }
  }

  override fun onTopResumedActivityChanged(isTopResumedActivity: Boolean) {
    // Handles "top" resumed event on multi-window environment
    if (isTopResumedActivity) {
      soundClassifier.start()
    } else {
      soundClassifier.stop()
    }
  }

  override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
  ) {
    if (requestCode == REQUEST_RECORD_AUDIO) {
      if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
        Log.i(TAG, "Audio permission granted :)")
        soundClassifier.start()
      } else {
        Log.e(TAG, "Audio permission not granted :(")
      }
    }
  }

  @RequiresApi(Build.VERSION_CODES.M)
  private fun requestMicrophonePermission() {
    if (ContextCompat.checkSelfPermission(
        this,
        Manifest.permission.RECORD_AUDIO
      ) == PackageManager.PERMISSION_GRANTED
    ) {
      soundClassifier.start()
    } else {
      requestPermissions(arrayOf(Manifest.permission.RECORD_AUDIO),
REQUEST_RECORD_AUDIO)
    }
  }
  private fun keepScreenOn(enable: Boolean) =
    if (enable) {

window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_O
N)
    } else {

window.clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_
ON)
    }


  companion object {
    const val REQUEST_RECORD_AUDIO = 1337
    private const val TAG = "AudioDemo"
  }
}
```

activity_main.xml

```
          <?xml version="1.0" encoding="utf-8"?>

<androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="@android:color/white"
        android:elevation="4dp"
        app:layout_constraintTop_toTopOf="parent">
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:contentDescription="@string/tensorflow_lite_logo"
            android:src="@drawable/tfl2_logo_dark" />
    </androidx.appcompat.widget.Toolbar>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingTop="?attr/actionBarSize">

        <com.google.android.material.switchmaterial.SwitchMaterial
            android:id="@+id/input_switch"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:checked="true"
            android:text="@string/label_input"
            android:textColor="@color/dark_blue_gray800"
            android:textSize="16sp"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:switchPadding="8dp"
            app:useMaterialThemeColors="true" />

        <TextView
            android:id="@+id/overlap_factor_text_view"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="12dp"
            android:text="@string/overlap_factor_label"
            android:textColor="@color/dark_blue_gray800"
            android:textSize="16sp"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="@id/input_switch"
```

```
            app:layout_constraintTop_toBottomOf="@id/input_switch" />

        <com.google.android.material.slider.Slider
            android:id="@+id/overlap_factor_slider"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginEnd="16dp"
            android:valueFrom="0.0"
            android:valueTo="1.0"

app:layout_constraintBottom_toBottomOf="@+id/overlap_factor_text_view"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toEndOf="@id/overlap_factor_text_view"
            app:layout_constraintTop_toTopOf="@id/overlap_factor_text_view"
            tools:value="0.8" />

        <View
            android:id="@+id/divider_view"
            android:layout_width="match_parent"
            android:layout_height="1dp"
            android:layout_marginStart="16dp"
            android:layout_marginTop="32dp"
            android:layout_marginEnd="16dp"
            android:background="@color/gray400"

app:layout_constraintTop_toBottomOf="@id/overlap_factor_text_view" />

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recycler_view"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_marginTop="24dp"
            android:scrollbars="vertical"
            app:layoutManager="LinearLayoutManager"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="@id/divider_view"
            tools:itemCount="3"
```

```
          tools:listitem="@layout/item_probability" />

    </androidx.constraintlayout.widget.ConstraintLayout>

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

2. LANGUAGE PROBABILITY CODE

ProbabilitiesAdapter.kt

```
package org.tensorflow.lite.examples.soundclassifier

import android.animation.ObjectAnimator
import android.content.res.ColorStateList
import android.view.LayoutInflater
import android.view.ViewGroup
import android.view.animation.AccelerateDecelerateInterpolator
import androidx.recyclerview.widget.RecyclerView
import
org.tensorflow.lite.examples.soundclassifier.databinding.ItemProbabilityBindin
g

internal class ProbabilitiesAdapter :
RecyclerView.Adapter<ProbabilitiesAdapter.ViewHolder>() {
  var labelList = emptyList<String>()
  var probabilityMap = mapOf<String, Float>()

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
    val binding =
      ItemProbabilityBinding.inflate(LayoutInflater.from(parent.context), parent,
false)
    return ViewHolder(binding)
  }

  override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val label = labelList[position]
    val probability = probabilityMap[label] ?: 0f
    holder.bind(position, label, probability)
  }

  override fun getItemCount() = labelList.size

  class ViewHolder(private val binding: ItemProbabilityBinding) :
    RecyclerView.ViewHolder(binding.root) {
    fun bind(position: Int, label: String, probability: Float) {
      with(binding) {
        labelTextView.text = label
        progressBar.progressBackgroundTintList = progressColorPairList[position
% 3].first
        progressBar.progressTintList = progressColorPairList[position %
3].second

        val newValue = (probability * 100).toInt()
        // If you don't want to animate, you can write like `progressBar.progress =
newValue`.
        val animation =
          ObjectAnimator.ofInt(progressBar, "progress", progressBar.progress,
newValue)
        animation.duration = 100
        animation.interpolator = AccelerateDecelerateInterpolator()
        animation.start()
      }
    }

    companion object {
      /** List of pairs of background tint and progress tint */
      private val progressColorPairList = listOf(
        ColorStateList.valueOf(0xfff9e7e4.toInt()) to
ColorStateList.valueOf(0xffd97c2e.toInt()),
        ColorStateList.valueOf(0xfff7e3e8.toInt()) to
ColorStateList.valueOf(0xffc95670.toInt()),
        ColorStateList.valueOf(0xffecf0f9.toInt()) to
ColorStateList.valueOf(0xff714Fe7.toInt()),
      )
    }
  }
}
```

3. SOUND CLASSIFIER CODE

SoundClassifier.kt

```
package org.tensorflow.lite.examples.soundclassifier

import android.content.Context
import android.media.AudioFormat
import android.media.AudioRecord
import android.media.MediaRecorder
import android.os.SystemClock
import android.util.Log
import androidx.annotation.MainThread
import androidx.lifecycle.DefaultLifecycleObserver
import androidx.lifecycle.LifecycleOwner
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import java.io.BufferedReader
import java.io.IOException
import java.io.InputStreamReader
import java.nio.FloatBuffer
import java.util.Locale
import java.util.concurrent.TimeUnit
import java.util.concurrent.locks.ReentrantLock
import kotlin.concurrent.withLock
import kotlin.math.ceil
import kotlin.math.sin
import org.tensorflow.lite.Interpreter
import org.tensorflow.lite.support.common.FileUtil
class SoundClassifier(context: Context, private val options: Options = Options())
:
  DefaultLifecycleObserver {
  class Options constructor(


    val metadataPath: String = "labels.txt",


    val modelPath: String = "sound_classifier.tflite",
    val sampleRate: Int = 44_100,
    val audioPullPeriod: Long = 50L,
    val warmupRuns: Int = 3,

    val pointsInAverage: Int = 10,
    var overlapFactor: Float = 0.8f,

    var probabilityThreshold: Float = 0.3f,
  )

  val isRecording: Boolean
    get() = recordingThread?.isAlive == true

  val probabilities: LiveData<Map<String, Float>>
    get() = _probabilities
  private val _probabilities = MutableLiveData<Map<String, Float>>()

  private val recordingBufferLock: ReentrantLock = ReentrantLock()

  var isClosed: Boolean = true
    private set
  var lifecycleOwner: LifecycleOwner? = null
    @MainThread
    set(value) {
      if (field === value) return
      field?.lifecycle?.removeObserver(this)
      field = value?.also {
        it.lifecycle.addObserver(this)
      }
    }

  var overlapFactor: Float
    get() = options.overlapFactor
    set(value) {
      options.overlapFactor = value.also {
        recognitionPeriod = (1000L * (1 - value)).toLong()
      }
    }

  var probabilityThreshold: Float
    get() = options.probabilityThreshold
    set(value) {
      options.probabilityThreshold = value
    }

  var isPaused: Boolean = false
    set(value) {
      field = value
      if (value) stop() else start()
    }

  lateinit var labelList: List<String>
    private set
```

}

```
private var recognitionPeriod = (1000L * (1 - overlapFactor)).toLong()

private lateinit var interpreter: Interpreter

private var modelInputLength = 0

private var modelNumClasses = 0

private lateinit var predictionProbs: FloatArray

private var latestPredictionLatencyMs = 0f

private var recordingThread: Thread? = null
private var recognitionThread: Thread? = null

private var recordingOffset = 0
private lateinit var recordingBuffer: ShortArray

private lateinit var inputBuffer: FloatBuffer

init {
  loadLabels(context)
  setupInterpreter(context)
  warmUpModel()
}

override fun onResume(owner: LifecycleOwner) = start()
override fun onPause(owner: LifecycleOwner) = stop()

fun start() {
  if (!isPaused) {
    startAudioRecord()
  }
}
fun stop() {
  if (isClosed || !isRecording) return
  recordingThread?.interrupt()
  recognitionThread?.interrupt()

  _probabilities.postValue(labelList.associateWith { 0f })
}

fun close() {
  stop()

  if (isClosed) return
  interpreter.close()

  isClosed = true
}

private fun loadLabels(context: Context) {
  try {
      val                                                   reader                                                    =
BufferedReader(InputStreamReader(context.assets.open(options.metadataPath))
)
      val wordList = mutableListOf<String>()
      reader.useLines { lines ->
        lines.forEach {
          wordList.add(it.split(" ").last())
        }
      }
      labelList = wordList.map { it.toTitleCase() }
    } catch (e: IOException) {
      Log.e(TAG, "Failed to read model ${options.metadataPath}: ${e.message}")
    }
  }

  private fun setupInterpreter(context: Context) {
    interpreter = try {
      val tfliteBuffer = FileUtil.loadMappedFile(context, options.modelPath)
      Log.i(TAG, "Done creating TFLite buffer from ${options.modelPath}")
      Interpreter(tfliteBuffer, Interpreter.Options())
    } catch (e: IOException) {
      Log.e(TAG, "Failed to load TFLite model - ${e.message}")
      return
    }

    // Inspect input and output specs.
    val inputShape = interpreter.getInputTensor(0).shape()
    Log.i(TAG, "TFLite model input shape: ${inputShape.contentToString()}")
  modelInputLength = inputShape[1]

  val outputShape = interpreter.getOutputTensor(0).shape()
  Log.i(TAG, "TFLite output shape: ${outputShape.contentToString()}")
  modelNumClasses = outputShape[1]
  if (modelNumClasses != labelList.size) {
    Log.e(
      TAG,
      "Mismatch between metadata number of classes (${labelList.size})" +
        " and model output length ($modelNumClasses)"
    )
  }
  // Fill the array with NaNs initially.
  predictionProbs = FloatArray(modelNumClasses) { Float.NaN }

  inputBuffer = FloatBuffer.allocate(modelInputLength)
}

private fun warmUpModel() {
  generateDummyAudioInput(inputBuffer)
  for (n in 0 until options.warmupRuns) {
    val t0 = SystemClock.elapsedRealtimeNanos()

    // Create input and output buffers.
    val outputBuffer = FloatBuffer.allocate(modelNumClasses)
    inputBuffer.rewind()
    outputBuffer.rewind()
    interpreter.run(inputBuffer, outputBuffer)

    Log.i(
      TAG,
      "Switches: Done calling interpreter.run(): %s (%.6f ms)".format(
        outputBuffer.array().contentToString(),
        (SystemClock.elapsedRealtimeNanos() - t0) / NANOS_IN_MILLIS
      )
    )
  }
}

private fun generateDummyAudioInput(inputBuffer: FloatBuffer) {
  val twoPiTimesFreq = 2 * Math.PI.toFloat() * 1000f
  for (i in 0 until modelInputLength) {
    val x = i.toFloat() / (modelInputLength - 1)
    inputBuffer.put(i, sin(twoPiTimesFreq * x.toDouble()).toFloat())
  }
}

@Synchronized
private fun startAudioRecord() {
  if (isRecording) return
  recordingThread = AudioRecordingThread().apply {
    start()
  }
  isClosed = false
  }

  private fun startRecognition() {
    recognitionThread = RecognitionThread().apply {
      start()
    }
  }

  private inner class AudioRecordingThread : Thread() {
    override fun run() {
      var bufferSize = AudioRecord.getMinBufferSize(
        options.sampleRate,
        AudioFormat.CHANNEL_IN_MONO,
        AudioFormat.ENCODING_PCM_16BIT
      )
      if (bufferSize == AudioRecord.ERROR || bufferSize == AudioRecord.ERROR_BAD_VALUE) {
        bufferSize = options.sampleRate * 2
        Log.w(TAG, "bufferSize has error or bad value")
      }
      Log.i(TAG, "bufferSize = $bufferSize")
      val record = AudioRecord(
        // including MIC, UNPROCESSED, and CAMCORDER.
        MediaRecorder.AudioSource.VOICE_RECOGNITION,
        options.sampleRate,
        AudioFormat.CHANNEL_IN_MONO,
        AudioFormat.ENCODING_PCM_16BIT,
        bufferSize
```

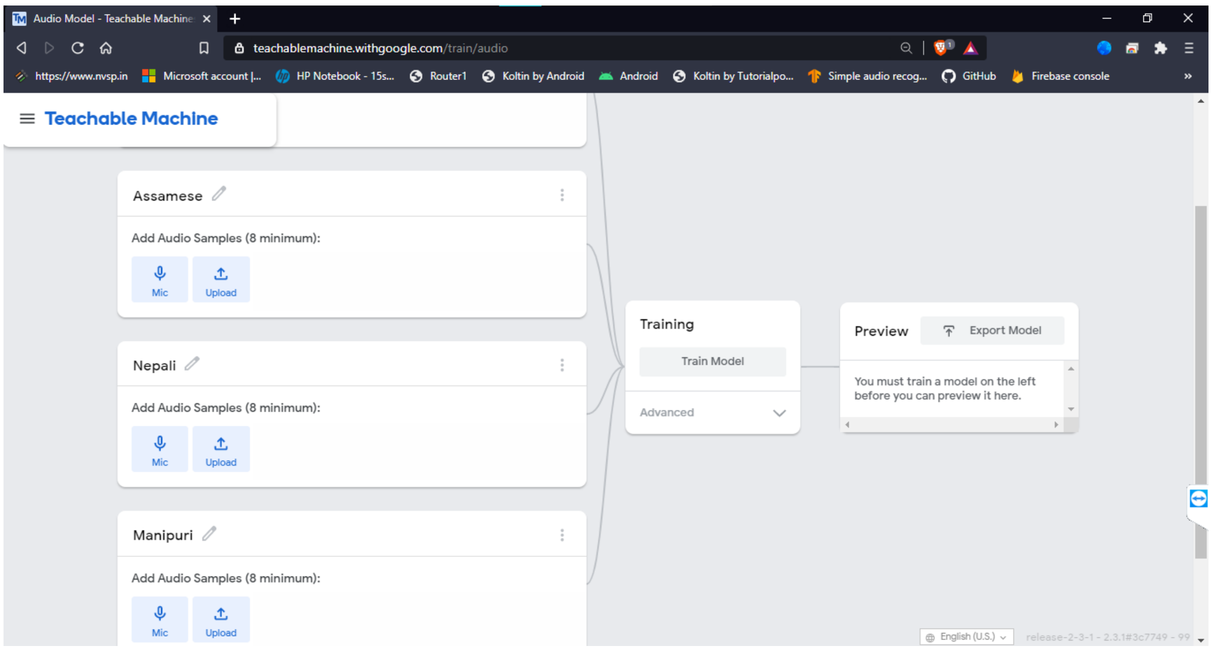
```
)
if (record.state != AudioRecord.STATE_INITIALIZED) {
  Log.e(TAG, "AudioRecord failed to initialize")
  return
}
Log.i(TAG, "Successfully initialized AudioRecord")
val bufferSamples = bufferSize / 2
val audioBuffer = ShortArray(bufferSamples)
val recordingBufferSamples =
  ceil(modelInputLength.toFloat() / bufferSamples.toDouble())
    .toInt() * bufferSamples
Log.i(TAG, "recordingBufferSamples = $recordingBufferSamples")
recordingOffset = 0
recordingBuffer = ShortArray(recordingBufferSamples)
record.startRecording()
Log.i(TAG, "Successfully started AudioRecord recording")

// Start recognition (model inference) thread.
startRecognition()

while (!isInterrupted) {
  try {
    TimeUnit.MILLISECONDS.sleep(options.audioPullPeriod)
  } catch (e: InterruptedException) {
    Log.w(TAG, "Sleep interrupted in audio recording thread.")
    break
  }
```

```
    when (record.read(audioBuffer, 0, audioBuffer.size)) {
     AudioRecord.ERROR_INVALID_OPERATION -> {
      Log.w(TAG, "AudioRecord.ERROR_INVALID_OPERATION")
     }
     AudioRecord.ERROR_BAD_VALUE -> {
      Log.w(TAG, "AudioRecord.ERROR_BAD_VALUE")
     }
     AudioRecord.ERROR_DEAD_OBJECT -> {
      Log.w(TAG, "AudioRecord.ERROR_DEAD_OBJECT")
     }
     AudioRecord.ERROR -> {
      Log.w(TAG, "AudioRecord.ERROR")
     }
     bufferSamples -> {
      // We apply locks here to avoid two separate threads (the recording and
      // recognition threads) reading and writing from the recordingBuffer at the
same
      // time, which can cause the recognition thread to read garbled audio
snippets.
      recordingBufferLock.withLock {
       audioBuffer.copyInto(
        recordingBuffer,
        recordingOffset,
        0,
        bufferSamples
       )
       recordingOffset = (recordingOffset + bufferSamples) %
recordingBufferSamples
```

```
          }
         }
        }
       }
      }
     }

  private inner class RecognitionThread : Thread() {
    override fun run() {
      if (modelInputLength <= 0 || modelNumClasses <= 0) {
        Log.e(TAG, "Switches: Cannot start recognition because model is unavailable.")
        return
      }
      val outputBuffer = FloatBuffer.allocate(modelNumClasses)
      while (!isInterrupted) {
        try {
          TimeUnit.MILLISECONDS.sleep(recognitionPeriod)
        } catch (e: InterruptedException) {
          Log.w(TAG, "Sleep interrupted in recognition thread.")
          break
        }
        var samplesAreAllZero = true

        recordingBufferLock.withLock {
          var j = (recordingOffset - modelInputLength) % modelInputLength
          if (j < 0) {
            j += modelInputLength
      }

      for (i in 0 until modelInputLength) {
        val s = if (i >= options.pointsInAverage && j >= options.pointsInAverage) {
          ((j - options.pointsInAverage + 1)..j).map { recordingBuffer[it % modelInputLength] }
            .average()
        } else {
          recordingBuffer[j % modelInputLength]
        }
        j += 1

        if (samplesAreAllZero && s.toInt() != 0) {
          samplesAreAllZero = false
        }
        inputBuffer.put(i, s.toFloat())
      }
    }
    if (samplesAreAllZero) {
      Log.w(TAG, "No audio input: All audio samples are zero!")
      continue
    }
    val t0 = SystemClock.elapsedRealtimeNanos()
    inputBuffer.rewind()
    outputBuffer.rewind()
    interpreter.run(inputBuffer, outputBuffer)
    outputBuffer.rewind()
```

```
        outputBuffer.get(predictionProbs) // Copy data to predictionProbs.

        val probList = predictionProbs.map {
          if (it > probabilityThreshold) it else 0f
        }
        _probabilities.postValue(labelList.zip(probList).toMap())

        latestPredictionLatencyMs = ((SystemClock.elapsedRealtimeNanos() - t0) /
1e6).toFloat()
      }
    }
  }

  companion object {
    private const val TAG = "SoundClassifier"

    private const val NANOS_IN_MILLIS = 1_000_000.toDouble()
  }
}

private fun String.toTitleCase() =
  splitToSequence("_")
    .map { it.capitalize(Locale.ROOT) }
    .joinToString(" ")
    .trim()
```

APPENDIX II – SCREENSHOTS



Audio Model - Teachable Machine
teachablemachine.withgoogle.com/train/audio
Teachable Machine
Assamese
Add Audio Samples (8 minimum):
Mic
Upload
Nepali
Add Audio Samples (8 minimum):
Manipuri
Add Audio Samples (8 minimum):
Training
Train Model
Advanced
Preview
Export Model
You must train a model on the left before you can preview it here.

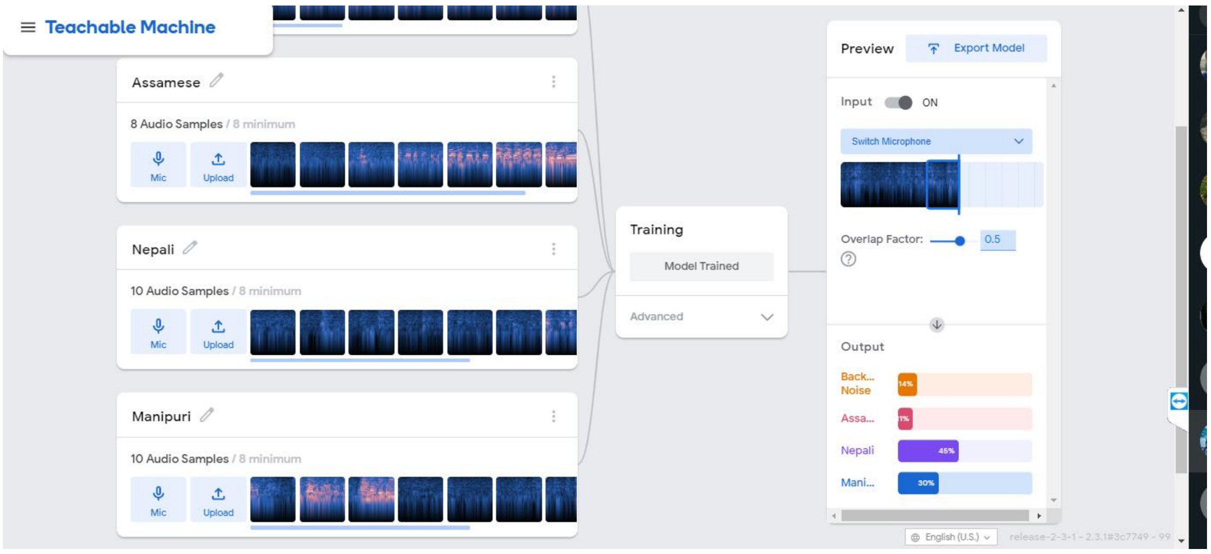


FIG A.1: TEACHABLE MACHINE

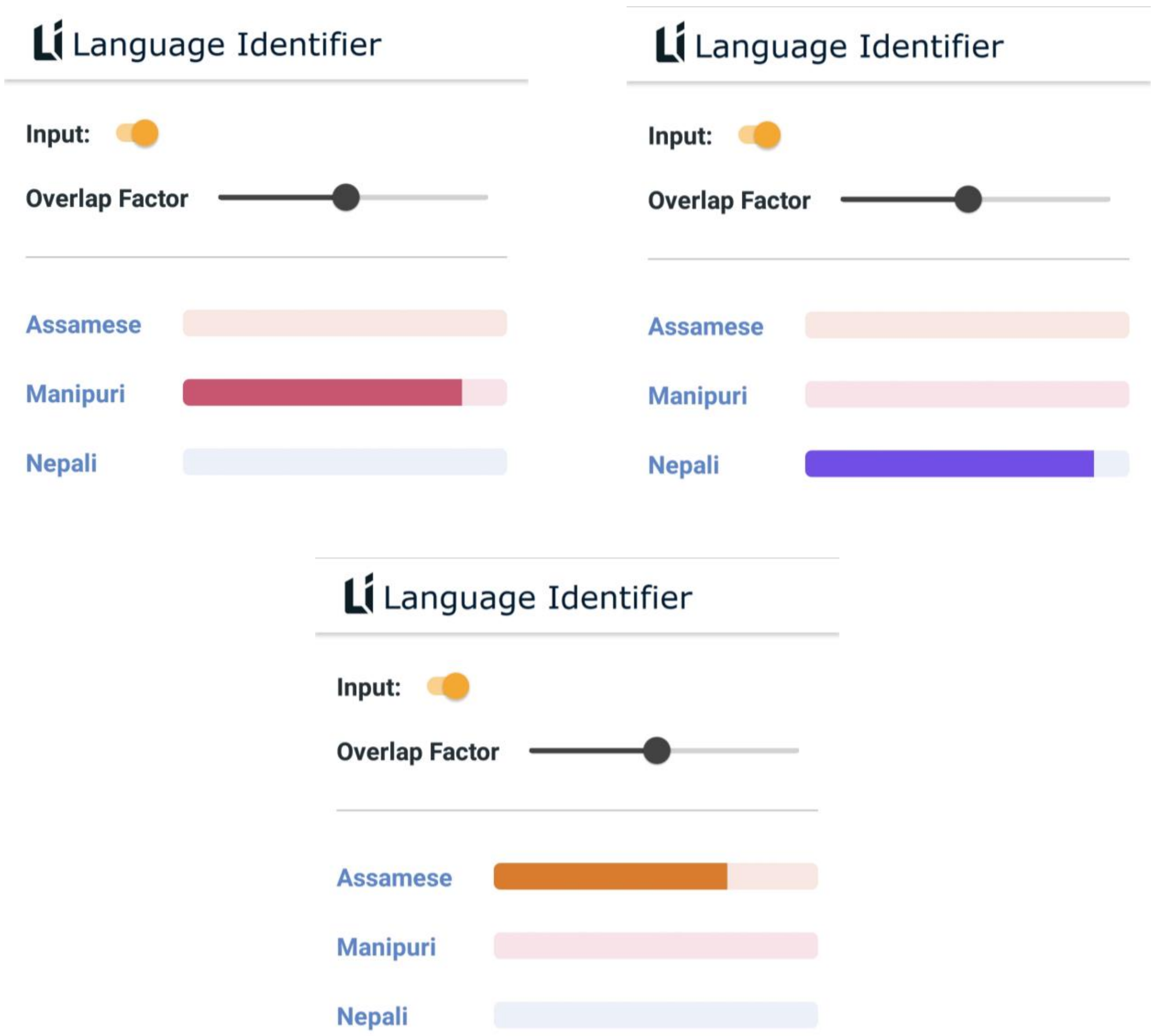


FIG A.2: LANGUAGE IDENTIFIER APPLICATION

REFERENCES

- “Machine Learning For Absolute Beginners: A Plain English Introduction (Second Edition)” by Oliver Theobald
- edureka! – YouTube channel Machine learning course “https://www.youtube.com/watch?v=GwIo3gDZCVQ”
- Audio classification with Machine Learning by Krishna “https://www.youtube.com/channel/UCNU_lfiiWBdtULKOw6X0Dig”
- A Survey of Language Identification Techniques and Applications, Author: Archana Garg,Vishal Gupta, Manish Jindal, Punjab University, Chandigarh, India
- Adaptive Noise Cancelling for Audio Signals Using Least Mean Square Algorithm ,Author: Amav Mendiratta, Dr. Devendra Jha, Scientific Analysis Group (SAG) , New Delhi, India.