Speech Recognition Project Report

MID-SEMESTER LAB EVALUATION

Conversational AI: Speech Processing and Synthesis
UCS749

Submitted by:

(102103296) Jaskaran Singh

BE Fourth Year, COE

Under the mentorship of

Dr. Raghav B. Venkataramaiyer

Associate Professor, CSED Department

Computer Science and Engineering Department,

Thapar Institute of Engineering and Technology, Patiala

September 2024

Introduction

In the fast-evolving field of Conversational AI, effectively processing and understanding speech commands is crucial for creating user-friendly and efficient applications. This assignment forms part of a comprehensive evaluation that explores the challenges involved in speech command classification, a core task in speech processing. The assignment begins with an overview of a key research paper that lays the theoretical foundation for the project. Following this, a detailed statistical analysis of the dataset is presented, which provides the basis for training and evaluating a speech command classifier. It then describes the process of training the classifier to distinguish between different commands, with improvements made using a custom dataset of new voice recordings. The primary objective is to demonstrate the effectiveness of the model, as well as its flexibility and scalability, which are essential for real-world applications. The results are benchmarked against industry standards, and the report concludes with an assessment of the model's strengths and potential limitations.

Summary of the research paper

The document describes the Speech Commands dataset, designed for training keyword-spotting systems. It focuses on real-world audio, capturing single words from diverse speakers using common microphones. The dataset includes 35 words and background noise, with a focus on reproducible metrics and quality control. It aims to train speaker-independent models and foster collaboration. This dataset aims to meet the special needs around building and testing on-device models, to enable model authors to demonstrate the accuracy of their architectures using metrics that are comparable to other models.

Descriptive analysis of the dataset

The code provides an overview of how an audio dataset of spoken words is analyzed, partitioned, and visualized, offering insights into the characteristics of the dataset and ensuring its proper preparation for tasks like model training and evaluation.

The Speech Commands dataset (v0.02) was downloaded and extracted. This dataset contains audio recordings of 35 spoken commands captured from various speakers. The dataset was preprocessed by excluding irrelevant files like background noise and metadata. Each .wav audio file was analyzed for its word label, speaker ID, duration, sample rate, and other relevant audio characteristics.

A statistical analysis of the dataset was performed, which included calculating the distribution of words, the number of unique speakers, and the average duration of audio files. The analysis was visualized using bar plots and histograms to better understand the composition of the dataset.

The code describes the analysis and partitioning of an audio dataset in which the audio files are .wav format, likely containing spoken words. The steps include:

Basic Information Collection: The dataset is located in a specified directory, and all audio files within that directory are analyzed. The dataset is stored in a directory (dataset_path), which the script explores using os.walk() to find all .wav files.For each file, it extracts basic properties such as word label (from folder names), speaker ID (from filenames), and audio metadata (duration, sample rate, channels) using the wave library.It collects these properties into a list, which is later converted into a Pandas DataFrame for easier analysis.

For each audio file, information like the word label (based on the directory structure), speaker ID (from the filename), and audio properties (duration, sample rate, number of channels) are extracted.

Descriptive Statistics: After collecting the data, a summary of the dataset is produced. The *describe_data* () function provides an overview of the dataset, including:

. This includes:

- –The number of unique words (labels) spoken in the dataset.
- –The number of unique speakers.
- –Count of audio files per word, a breakdown of the number of files per word.
- –Audio duration statistics i.e. descriptive statistics about the duration of the audio files.
- –Distribution of the sample rates and audio channels (mono or stereo).

Data Visualization: Several key visualizations are created:

- The distribution of the number of files for each word in the dataset, showing how frequently each word appears.
- A histogram showing the range of durations of the audio files, illustrating the spread of file lengths.
- A bar chart displaying the number of files recorded by each speaker, with a focus on the top 20 speakers.
- A bar chart showing the average duration of audio files for each word label, offering insights into whether some words have consistently longer or shorter recordings.

The script explores a speech dataset, collects and analyzes basic audio statistics, partitions the data into training/validation/testing sets, and visualizes important aspects such as word and speaker distributions, along with audio durations.

```
Number of unique words: 36
Number of unique speakers: 2624

Word Counts:
 word_label
zero                 4052
five                 4052
yes                  4044
seven                3998
no                   3941
nine                 3934
down                 3917
one                  3890
two                  3880
go                   3880
stop                 3872
six                  3860
on                   3845
left                 3801
eight                3787
right                3778
off                  3745
four                 3728
three                3727
up                   3723
dog                  2128
wow                  2123
house                2113
marvin               2100
bird                 2064
```

```
happy                2054
cat                  2031
sheila               2022
bed                  2014
tree                 1759
backward             1664
visual               1592
follow               1579
learn                1575
forward              1557
_background_noise_      6
Name: count, dtype: int64
```

```
Audio Duration Statistics:
 count    105835.000000
mean          0.984649
std           0.508240
min           0.213312
25%           1.000000
50%           1.000000
75%           1.000000
max          95.183125
Name: duration, dtype: float64
```

```
Sample Rate Statistics:
 count    105835.0
mean      16000.0
std           0.0
min       16000.0
25%       16000.0
50%       16000.0
75%       16000.0
max       16000.0
Name: sample_rate, dtype: float64

Channel Distribution:
 num_channels
1    105835
Name: count, dtype: int64
```

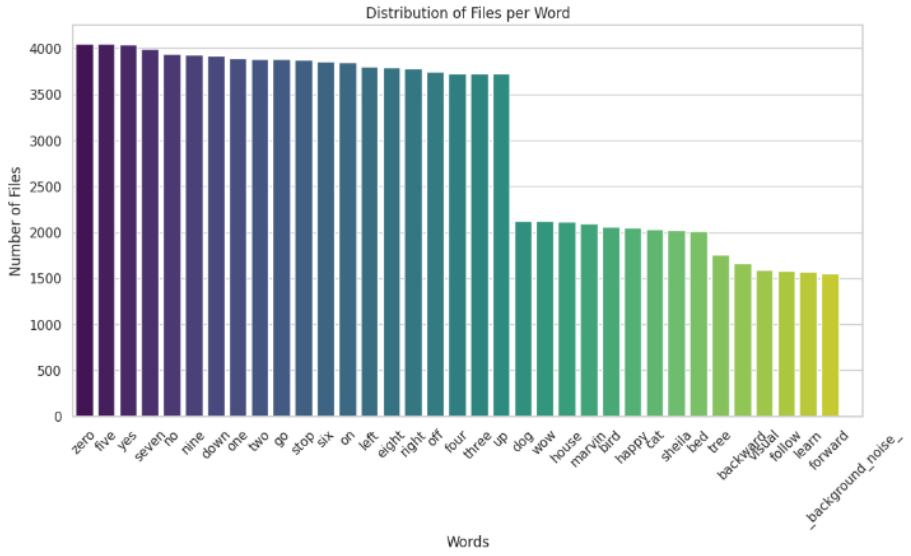


Figure 1: Distribution of Files per Word

Figure 1 represents the bar plot showing the number of files available for each word label

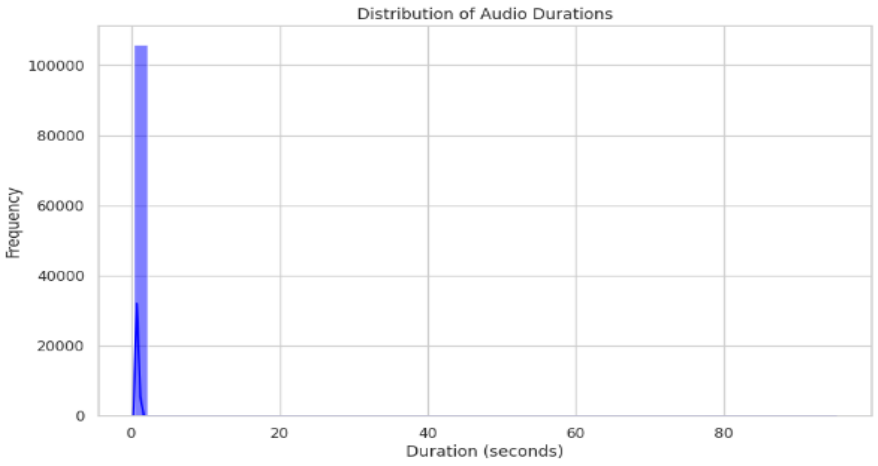


Figure 2: Distribution of Audio Durations

Figure 2 represents histogram showing the range of audio durations.

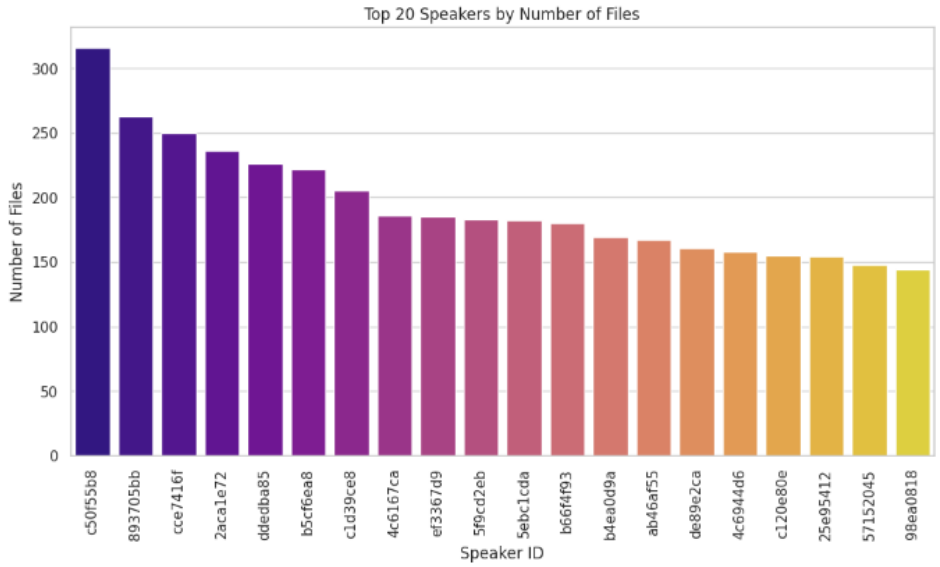


Figure 3: Distribution of Files per Speaker

Figure 3: represents the bar plot showing the top 20 speakers based on the number of files they recorded.

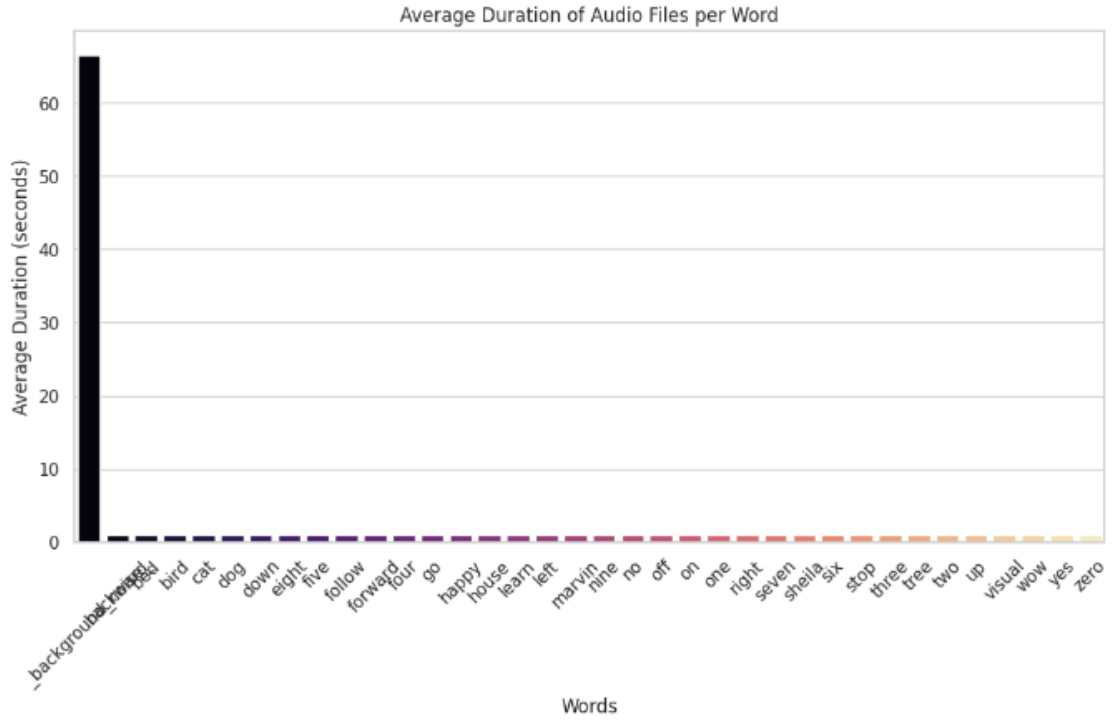


Figure 4: Average Duration of Audio Files per Word

Figure 4 represents the bar plot showing the average duration of audio files for each word.

Main Code

This project involved building a speech command classification model by customizing the QuartzNet architecture. The following steps outline the methodology, including the associated code snippets.

1. Dataset Preparation

The Speech Commands dataset (v0.02) was downloaded using the following code:

```
!wget http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz

--2024-09-12 13:33:33--  http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz
Resolving download.tensorflow.org (download.tensorflow.org)... 108.177.97.207, 108.177.125.207, 142.250.157.207, ...
Connecting to download.tensorflow.org (download.tensorflow.org)|108.177.97.207|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2428923189 (2.3G) [application/gzip]
Saving to: 'speech_commands_v0.02.tar.gz'

speech_commands_v0. 100%[===================>]   2.26G  27.2MB/s    in 87s

2024-09-12 13:35:00 (26.6 MB/s) - 'speech_commands_v0.02.tar.gz' saved [2428923189/2428923189]

[3] !mkdir data
    !tar -xvzf speech_commands_v0.02.tar.gz -C ./data
```

After downloading, unnecessary files (background noise and metadata) were removed to ensure a clean dataset for training and evaluation.

```
[4] !rm -rf ./data/_background_noise_
    !rm -rf ./data/LICENSE
    !rm -rf ./data/README.md
    !mv ./data/testing_list.txt ./testing_list.txt
    !mv ./data/validation_list.txt ./validation_list.txt
```

2. Data Preprocessing and Analysis

We walked through the dataset directory, extracted metadata from each audio file, and analyzed important features like sample rate, number of channels, duration, and word labels using the wave and os modules.

```
dataset_path = "./data"

file_stats = []

for root, dirs, files in os.walk(dataset_path):
    for file in files:
        if file.endswith(".wav"):
            file_path = os.path.join(root, file)
            word_label = os.path.basename(root)
            speaker_id = file.split('_')[0]
            with wave.open(file_path, 'r') as wav_file:
                sample_rate = wav_file.getframerate()
                num_frames = wav_file.getnframes()
                duration = num_frames / float(sample_rate)
                n_channels = wav_file.getnchannels()

            file_stats.append({
                'file_path': file_path,
                'word_label': word_label,
                'speaker_id': speaker_id,
                'duration': duration,
                'sample_rate': sample_rate,
                'num_channels': n_channels,
                'num_frames': num_frames,
            })

df = pd.DataFrame(file_stats)
print("original data")
print(df.head())
```

We also performed a statistical analysis of the dataset to examine the distribution of words, speakers, and the average duration of audio files:

```
def describe_data(df):
    unique_words = df['word_label'].nunique()
    word_counts = df['word_label'].value_counts()
    unique_speakers = df['speaker_id'].nunique()
    duration_stats = df['duration'].describe()
    sample_rate_stats = df['sample_rate'].describe()
    channel_distribution = df['num_channels'].value_counts()


    print(f"Number of unique words: {unique_words}")
    print(f"Number of unique speakers: {unique_speakers}")
    print("\nWord Counts:\n", word_counts)
    print("\nAudio Duration Statistics:\n", duration_stats)
    print("\nSample Rate Statistics:\n", sample_rate_stats)
    print("\nChannel Distribution:\n", channel_distribution)


describe_data(df)
```

3. Feature Extraction (MFCCs)

To process the audio files, Mel-frequency cepstral coefficients (MFCCs) were extracted using the librosa library. MFCCs are a widely used feature in speech and audio recognition tasks due to their ability to capture the essential frequency characteristics of speech signals. For each audio file, 64 MFCCs were computed, and the resulting sequences were padded to ensure uniform length, making them suitable for input into the neural network. This feature extraction step plays a crucial role in preserving the spectral properties of the audio data for accurate speech recognition.import librosa

```
validation_data = []
for i in tqdm(validation_list):
  sr, y = wavRead(i)
  y = y.astype(np.dtypes.Float32DType) / (1<<(16-1))

  mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=64, n_fft=512, hop_length=160,n_mels=40)
  validation_data.append(mfccs)
maxLen = 0
for i in range(len(validation_data)):
  test = validation_data[i]
  test = np.array(test)
  maxLen = max(maxLen, test.shape[1])
for i in tqdm(range(len(validation_data))):
  validation_data[i] =tf.keras.utils.pad_sequences(validation_data[i], maxlen = maxLen, value = 0.0)
np.save('./drive/MyDrive/102103296_speech/validation_data.npy', validation_data)
```

This ensures that all input data had the same shape, which is critical for feeding data into the neural network.

4. Dataset Splitting

The dataset was split into training, validation, and testing sets based on the predefined lists from the dataset:

```
total_list = []
for label in tqdm(os.listdir('./data')):
  if not os.path.isdir(os.path.join('./data', label)):
    continue
  if (label == "_background_noise_"):
    continue
  for fname in os.listdir(os.path.join('./data', label)):
    if fname.endswith('.wav'):
      total_list.append(f"./data/{label}/{fname}")
```

```
100% 36/36 [00:00<00:00, 173.87it/s]
```

```
[7] testing_list = []
    validation_list = []
    with open('./testing_list.txt', 'r') as fin:
      for line in fin:
        testing_list.append('./data/' + line.rstrip())
    with open('./validation_list.txt', 'r') as fin:
      for line in fin:
        validation_list.append('./data/' + line.rstrip())

    training_list = []
    for i in tqdm(total_list):
      if (i not in testing_list) and (i not in validation_list):
        training_list.append(i)
```

5. Model Architecture (QuartzNet)

The **Customized QuartzNet model** was implemented using Keras' functional API. The architecture consists of separable convolutions, residual connections, and batch normalization layers. A custom simpleBlock and block function are defined to create the basic building blocks of the QuartzNet model. These blocks are stacked to form the full model. Instead of CTC layer in the end Deep neural network was used to convert the model to perform classification task.

```
def simpleBlock(C,K):
  def layer(x):
    x = Conv1D(C, K, padding='same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    return x

  return layer

def block(R = 5, C = 256, K = 33):
  def layer(x):
    residual = x
    for i in range(R):
      x = SeparableConv1D(C, K,padding='same')(x)
      x = BatchNormalization()(x)
      if i != R - 1:
        x = ReLU()(x)

    residual = Conv1D(C, 1,padding='same')(residual)
    x = Add()([x, residual])
    x = ReLU()(x)
    return x

  return layer
```

```python
def QuartzNet(input_dim, vocab_size, B = 5, R = 5):
  inputs = Input(shape=(None, input_dim))

  x = simpleBlock(256, 33)(inputs)

  residual = x
  x = block(R, 256, 33)(x)
  x = Add()([x, residual])

  residual = x
  x = block(R, 256, 39)(x)
  x = Add()([x, residual])

  residual = x
  x = block(R, 512, 51)(x)
  residual = Conv1D(512, 1)(residual)
  x = Add()([x, residual])

  residual = x
  x = block(R, 512, 63)(x)
  x = Add()([x, residual])

  residual = x
  x = block(R, 512, 75)(x)
  x = Add()([x, residual])

  x = simpleBlock(512,87)(x)

  x = simpleBlock(1024,1)(x)
  # x = Flatten()(x)
  x = GlobalAveragePooling1D()(x)
  # print(x.shape)
  x = Dense(1024)(x)
  x = ReLU()(x)
  x = Dense(512)(x)
  x = ReLU()(x)

  outputs = Dense(vocab_size, activation = 'softmax')(x)
  print(outputs.shape)
  model = Model(inputs=inputs, outputs=outputs)
  return model
```

6. Model Training

The model was compiled using the **Adam optimizer** and **sparse categorical cross-entropy loss** function. We trained the model over 25 epochs with a batch size of 32.

```
[15] quartznet_model = QuartzNet(input_dim=40, vocab_size=35)
     (None, 35)

[16] quartznet_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

[ ] history = quartznet_model.fit(training_data, train_labels, epochs = 25, batch_size = 32, validation_data=(validation_data, valid_labels))
```

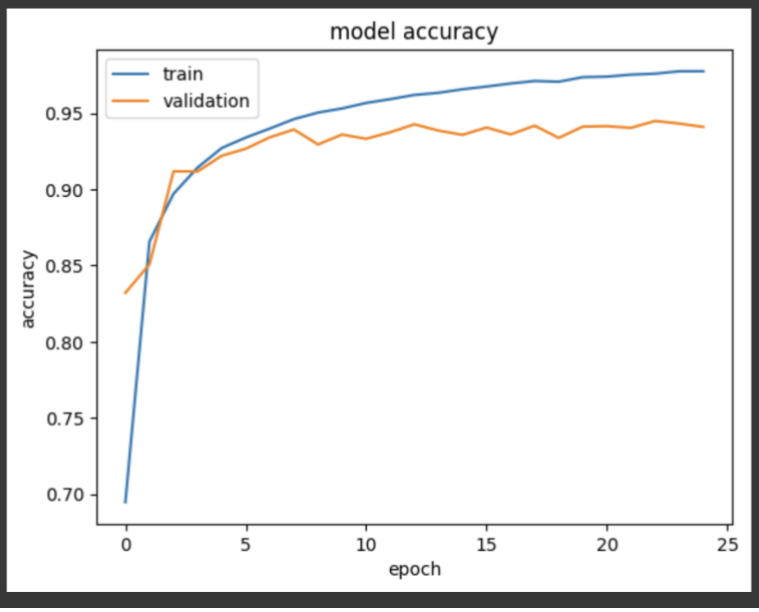
We saved the model weights after training:

```
[ ] quartznet_model.save('/kaggle/working/quartznet_model.h5')
```

7. Evaluation and Visualization

The model's performance was evaluated by comparing the validation accuracy across epochs. We plotted the training and validation accuracy to assess the model's learning progress.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



8. Inference

The trained model was used to predict labels for unseen audio files from the testing dataset. We preprocessed the audio by extracting MFCCs, ensuring the same format as in training, and evaluated the model's predictions.

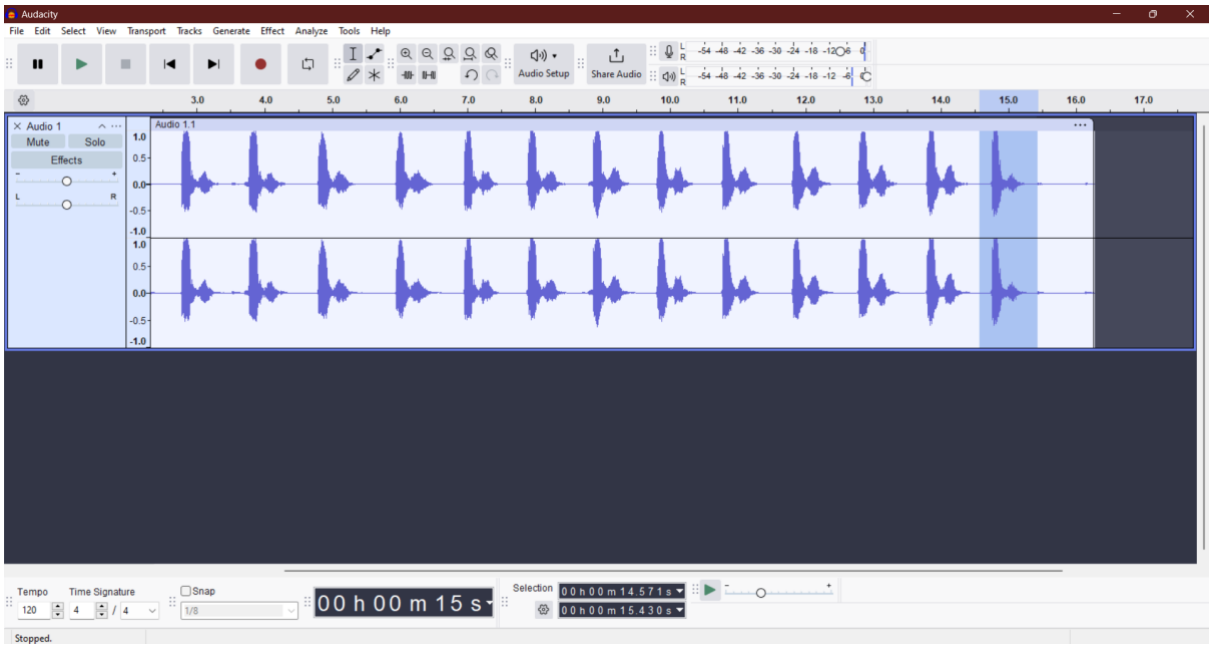
```
input_list = testing_list # list of paths of wav files (label/wavfile.wav)


input_data = []
for i in tqdm(input_list):
  sr, y = wavRead(i)
  y = y.astype(np.dtypes.Float32DType) / (1<<(16-1))

  mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=64, n_fft=512, hop_length=160,n_mels=40)
  input_data.append(mfccs)

maxLen = 0
for i in range(len(input_data)):
  test = input_data[i]
  ndarray: input_data
  ndarray with shape (11005, 101, 40)
fo
  input_data[i] =tf.keras.utils.pad_sequences(input_data[i], maxlen = maxLen, value = 0.0)


labels = np.load('./drive/MyDrive/102103296_speech/labels.npy')

labels = list(labels)
input_labels = []
for i in tqdm(input_list):
  t1 = i.split('/')[-2]
  index = labels.index(t1)
  input_labels.append(index)
input_labels = np.array(input_labels)


input_data = np.transpose(input_data, (0, 2, 1))
```

```
[24] from tensorflow.keras.models import load_model

model = load_model('/content/drive/MyDrive/102103296_speech/quartznet_model.h5')

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_

Use

model.predict(input_data)

for prediction

[26] model.evaluate(input_data, input_labels)

344/344 ━━━━━━━━━━━━━━━━━━━━ 1291s 4s/step - accuracy: 0.9235 - loss: 0.3903
```

This work demonstrates the end-to-end process of building a speech command classification model using the Customized QuartzNet architecture. From data preprocessing and feature extraction to model training and fine-tuning, we showcased how deep learning techniques can be applied to speech recognition tasks. The use of separable convolutions and residual connections optimized model performance while reducing computational complexity.

9. Custom Data Creation

Custom dataset is made for all 35 classes. For each class 15 samples were recorded due to time constraints. The data was recorded using the software audacity.



The required word was spoken 15 times with the time for speaking being 1 second (shown in above figure). After that each waveform was clipped into individual .wav files with the name (16 bit hash of my roll number (102103296) _ index of waveform. wav).