# MID-SEMESTER LAB EVALUATION

Conversational AI: Speech Processing and Synthesis

**(UCS749)**

## Recognize My Voice Commands


**Submitted by:**

Shatakshi Saxena (102117165)
BE Fourth Year, COPC (4CS6)

**Submitted to:**

Dr Raghav B. Venkataramaiyer

Associate Professor, CSED Department


**BE Fourth Year**



**Thapar Institute of Engineering and Technology, Patiala**

**September 2024**

## ABSTRACT

This research project develops an advanced audio recognition system using deep learning techniques. Initially, a sequential Convolutional Neural Network (CNN) model was employed on TensorFlow's Speech Commands dataset to establish a baseline for keyword recognition. Subsequently, the M5 CNN model, designed for 1D sequential audio data, was applied, demonstrating superior performance in recognizing speech commands. The M5 model was fine-tuned to enhance real-world applicability using a custom dataset of 35 audio command classes. Additionally, model quantization techniques, including Quantization-Aware Training (QAT), were implemented to optimize computational efficiency and model size while maintaining accuracy. This comprehensive approach, combining advanced neural architectures, custom dataset fine-tuning, and model optimization, represents a significant advancement in audio recognition systems with potential applications in voice-controlled interfaces, accessibility technologies, and audio-based security systems.


## INTRODUCTION

In recent years, advancements in deep learning have significantly enhanced the capabilities of audio recognition systems, particularly in the realm of keyword and command recognition. These systems are crucial in various applications, from virtual assistants and automated customer service to accessibility technologies and personal voice-controlled devices. The real-world applications of voice recognition technology offer substantial benefits to society, including assistance in healthcare, smart home automation, elderly care, customer service, and vehicle operation, thereby enhancing productivity, accessibility, and safety.

This project aims to develop an advanced audio recognition system using deep learning techniques, specifically focusing on recognizing user-defined voice commands. The research progresses through several key stages, each building upon the last to create a more robust and efficient system. A sequential Convolutional Neural Network (CNN) model is initially utilized and trained on TensorFlow's Speech Commands dataset. This dataset provides a robust foundation for handling general keyword recognition tasks, including a wide range of standard commands that serve as a useful benchmark for evaluating model performance. The sequential CNN architecture is particularly effective for processing audio data, as it can capture local patterns and features in the spectrogram representation of audio signals.

Building upon the insights gained from the sequential CNN, the project implements the M5 CNN model. The M5 architecture is designed explicitly for one-dimensional sequential audio data, offering several advantages over traditional CNN models, including efficient processing of raw audio waveforms, reduced computational complexity, and improved feature extraction from temporal audio patterns. The M5 model demonstrates superior performance in recognizing speech commands, making it an ideal choice for this audio recognition system. To address the specific needs of recognizing personalized voice commands, the project transitions to fine-tuning the pre-trained M5 model using a custom dataset. This dataset is

organized into 35 classes, each representing distinct voice commands, including those unique to users' vocal patterns. The fine-tuning process allows the model to adapt its learned features to the nuances of the custom commands, enhancing its recognition accuracy for specific use cases.

Given the substantial size of the dataset and the computational demands of deep learning models, model quantization becomes a critical consideration. The project employs Quantization-Aware Training (QAT), a sophisticated approach that simulates quantization effects during the training phase. This process reduces the precision of the weights and activations in the neural network, converting them from floating-point numbers to lower-bit integers. The integration of QAT offers several benefits, including decreased model computational footprint, reduced memory requirements, preserved accuracy and reliability, and enhanced model resilience to precision loss.

By combining model quantization with fine-tuning, the project efficiently handles extensive audio data while optimizing the model for real-world applications. This approach enhances the model's ability to recognize a broad spectrum of voice commands and ensures that it remains computationally feasible for deployment in resource-constrained environments. In conclusion, this research project bridges the gap between general audio recognition and personalized command recognition through advanced deep learning techniques and model optimization strategies. By leveraging sequential and M5 CNN architectures, fine-tuning with a custom dataset, and applying quantization techniques, the project demonstrates a comprehensive methodology for developing and deploying an effective voice command recognition system. The resulting system has the potential to significantly impact various fields, contributing to a more inclusive and efficient society through improved human-computer interaction.

## RELATED WORKS

*[Reference Paper summary in around 50 words]*

The paper discusses a speech commands dataset comprising 105,829 utterances recorded from 2,618 speakers optimized for training and evaluating keyword spotting models. It focuses on real-world audio, capturing single words and provides detailed evaluation metrics, including Top-One Error and Streaming Error Metrics. It aims to train speaker-independent models and foster collaboration. This dataset aims to meet the unique needs around building and testing on-device models, enabling model authors to demonstrate their architectures' accuracy using metrics comparable to other models.

[For more details, refer to the paper: Warden, P. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. arXiv:1804.03209v1 [cs.CL]. Google Brain, Mountain View, California. Available at: arXiv.]

# METHODOLOGY:

The methodology for this audio recognition system involves implementing and comparing two distinct neural network architectures: a Sequential Convolutional Neural Network (CNN) and the M5 model. Both models are designed to efficiently process and classify audio data, particularly for voice command recognition tasks.

## Models Used:

### Sequential CNN:

The Sequential CNN model is structured to handle spectrogram representations of audio signals. It accepts an input shape of (124, 129, 1) and begins with a resizing layer that normalizes the input to a fixed size of (32, 32, 1). This preprocessing step ensures consistent input dimensions and enhances processing efficiency. The architecture then employs two convolutional layers: the first with 32 filters and the second with 64 filters, both followed by Rectified Linear Unit (ReLU) activations. These layers are crucial for capturing essential features from the input spectrograms. Max-pooling and dropout layers are applied after each convolutional block to mitigate overfitting and downsample the data. The extracted features are then flattened and passed through two fully connected (dense) layers. The first dense layer contains 128 neurons, while the second serves as the output layer with 36 neurons, corresponding to the 35 command labels plus one background noise class. This compact model, containing approximately 1.6 million trainable parameters, balances computational efficiency and recognition accuracy, making it suitable for real-time tasks while handling large datasets effectively.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| resizing (Resizing) | (None, 32, 32, 1) | 0 |
| normalization (Normalization) | (None, 32, 32, 1) | 3 |
| conv2d (Conv2D) | (None, 30, 30, 32) | 320 |
| conv2d_1 (Conv2D) | (None, 28, 28, 64) | 18,496 |
| max_pooling2d (MaxPooling2D) | (None, 14, 14, 64) | 0 |
| dropout (Dropout) | (None, 14, 14, 64) | 0 |
| flatten (Flatten) | (None, 12544) | 0 |
| dense (Dense) | (None, 128) | 1,605,760 |
| dropout_1 (Dropout) | (None, 128) | 0 |
| dense_1 (Dense) | (None, 36) | 4,644 |

**Total params:** 1,629,223 (6.21 MB)
**Trainable params:** 1,629,220 (6.21 MB)
**Non-trainable params:** 3 (16.00 B)

**M5 CNN:**

The M5 model is specifically designed for processing raw audio waveforms. The model's architecture is defined in the M5 class, which inherits from nn.Module. It begins with an initial convolutional layer (conv1) that uses a large kernel size of 80 and a stride of 16, allowing it to capture low-level audio features effectively. This layer is followed by batch normalization (bn1) and max pooling (pool1) operations. The subsequent layers (conv2, conv3, and conv4) use smaller kernel sizes of 3, progressively increasing the number of channels from 32 to 64. Each convolutional layer is accompanied by batch normalization, ReLU activation, and max pooling, which help in feature extraction, normalization, and dimensionality reduction. The final part of the network consists of an average pooling operation followed by a fully connected layer (fc1) that maps the extracted features to the output classes. The forward method defines the data flow through these layers, culminating in a log-softmax operation for multi-class classification.

```
M5(
  (conv1): Conv1d(1, 32, kernel_size=(80,), stride=(16,))
  (bn1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv1d(32, 32, kernel_size=(3,), stride=(1,))
  (bn2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv1d(32, 64, kernel_size=(3,), stride=(1,))
  (bn3): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv1d(64, 64, kernel_size=(3,), stride=(1,))
  (bn4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool4): MaxPool1d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=64, out_features=35, bias=True)
)
Number of parameters: 26915
```

Both models incorporate techniques to enhance performance and prevent overfitting. The Sequential CNN uses dropout for regularization, while the M5 model relies on batch normalization. The M5 model's use of average pooling before the final fully connected layer reduces the number of parameters, potentially improving generalization. The number of parameters in the M5 model can be calculated using the count_parameters function, providing insight into the model's complexity. These architectures represent different approaches to audio recognition. The Sequential CNN works with spectrogram representations, leveraging 2D convolutions to capture frequency and time domain features. In contrast, the M5 model operates directly on the 1D audio waveform, using 1D convolutions to learn relevant features. This difference in input representation and processing allows for a comprehensive evaluation of audio recognition techniques, potentially catering to different use cases or computational constraints.

## Model Finetuning:

### Sequential CNN:

To fine-tune the pre-trained sequential CNN model for recognizing personalized voice commands, the custom dataset was extracted from a .tar file and loaded into training and validation subsets using TensorFlow's audio_dataset_from_directory. Quantization-aware training (QAT) was applied by wrapping the convolutional and dense layers with QuantizeWrapper, optimizing the model for deployment on resource-constrained devices. The model was then compiled with a reduced learning rate and fine-tuned on the custom dataset for 10 epochs, utilizing callbacks for model checkpointing and early stopping. After fine-tuning, the model was evaluated and saved in a quantized format to ensure efficient inference while maintaining performance.

The fine-tuning parameters for the model were configured to optimize performance on the custom dataset. The training was performed on 2 epochs, focusing on leveraging the custom dataset for improved accuracy. Key parameters included the use of tf.keras.callbacks.ModelCheckpoint will save the model's best version during training, ensuring the preservation of the highest-performing weights. Additionally, tf.keras.callbacks.EarlyStopping was employed to halt training early if no improvement in validation loss was observed for three consecutive epochs, thus preventing overfitting and optimizing training efficiency.

### M5 CNN:

Quantization in deep learning is converting a model from high-precision (e.g., float32) to lower-precision formats (e.g., int8). This process reduces the model's size and increases inference speed, making it more suitable for deployment on resource-constrained devices. The provided code applies dynamic quantization to the M5 CNN model, targeting torch.nn.Linear layers. Dynamic quantization specifically quantizes weights to int8 while keeping activations in the floating-point format during inference. This straightforward approach requires less computational overhead than static quantization, where both weights and activations are quantized. After applying dynamic quantization, the quantized model is saved and can be reloaded for inference, ensuring that its quantization parameters (such as scales and zero points) are intact and suitable for practical deployment scenarios.

Fine-tuning a quantized model involves converting it back to a full-precision format for further training. This step is crucial when additional training on a specific dataset is needed to improve the quantized model's performance. The code snippet first reverts the quantized model to full precision by preparing it with the torch for Quantization-Aware Training (QAT). quantization.prepare_qat method. This reversion allows the model to be retrained, adjusting its parameters to fit the new training data better. The fine-tuning process uses a modified training loop where a lower learning rate is typically employed to make fine-grained

adjustments. After fine-tuning, the model is converted back to its quantized form using a torch.quantization.convert. This final quantized model is saved ready for efficient inference with improved accuracy. This entire process ensures that the quantized model retains its performance and benefits from the refinements made during the fine-tuning phase.

## EXPERIMENTAL SETUP

### Dataset Description:

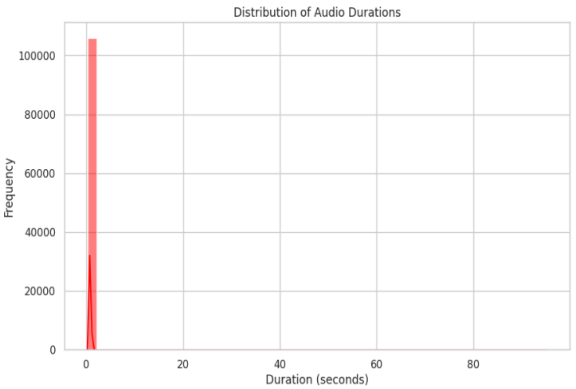
This project utilizes two distinct datasets for training and fine-tuning the model. The first dataset is the widely recognized "Speech Commands" dataset introduced by Pete Warden (2018) in his paper Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. This dataset consists of 105,829 utterances recorded by 2,618 speakers and 35 distinct voice commands. The dataset was designed to train and evaluate keyword spotting models and features a range of metrics such as Top-One Error and Streaming Error Metrics for assessing performance across various environments, including adversarial attacks and noise-tolerance improvements.

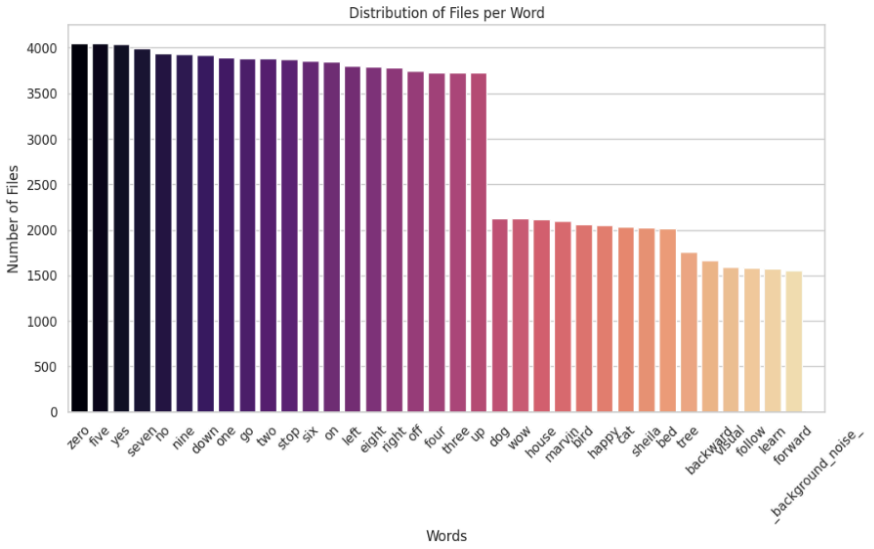
The second dataset is a custom-built collection, explicitly capturing the user's voice. It is created using personal voice recordings of the exact keywords as in the original dataset and recorded using a personal device to simulate real-world conditions, allowing the model to adapt to the specific voice characteristics of the user. This dataset was curated to enhance the model's performance in recognizing personalized voice commands. The custom dataset contains the same 35 command categories as the original dataset, including commands such as 'yes', 'no', 'up', 'down', 'left', 'right', 'stop', 'go', ‘ 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'backward', 'bed', 'bird', 'cat', 'dog', 'follow', 'forward', 'happy', 'house', 'learn', 'marvin', 'sheila', 'tree', 'visual', 'wow', and others, providing a comprehensive set of keywords for training. This dataset enables the fine-tuning of the model to accurately recognize the user's vocal patterns, thereby personalizing the audio recognition system. By combining the general-purpose dataset from Warden's research with the custom dataset tailored to the user's voice, the project ensures both broad keyword recognition capabilities and precise identification of personalized commands. The data is then split as per training, testing and validation set.

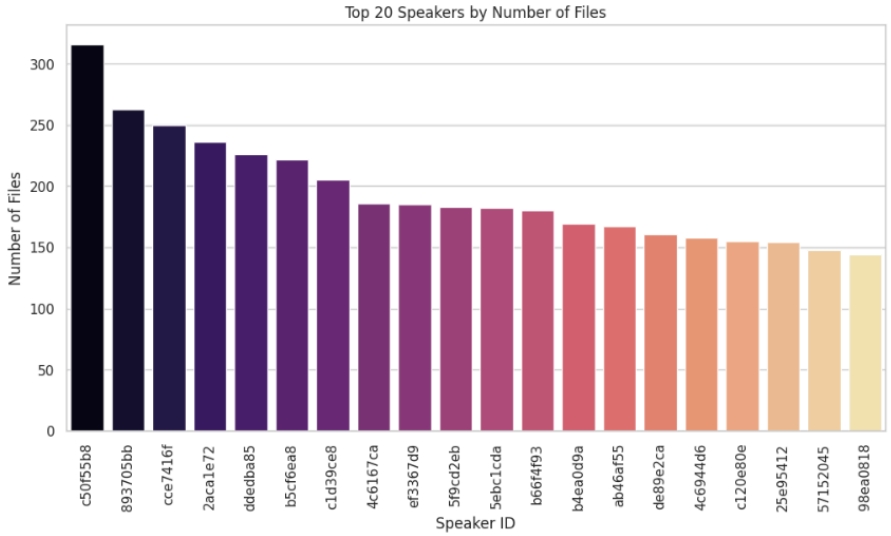
## Data Analytics and Visualization:

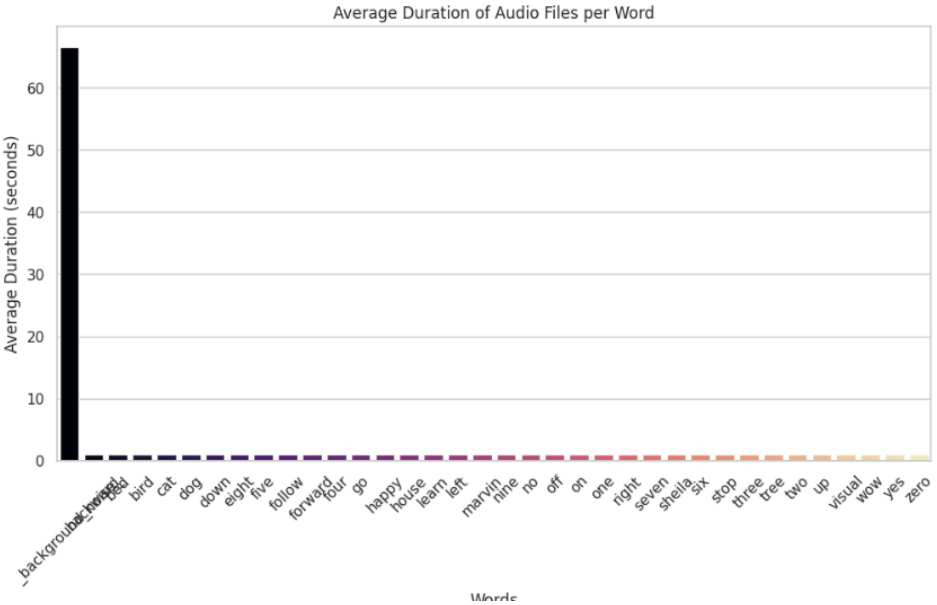
After collecting the data, a dataset summary is generated to provide an overview through the describe_data() function. This summary includes several key aspects: it reports the number of unique words (labels) spoken within the dataset and the number of distinct speakers. Additionally, it provides a breakdown of the count of audio files per word, offering insights into how many files correspond to each word. The dataset's audio duration statistics are also presented, including descriptive statistics about the lengths of the audio files. The distribution of sample rates and audio channels is analyzed to understand the variety of audio properties. Furthermore, it details the distribution of files for each word, illustrating the frequency of each word's appearance. A histogram shows the range of audio file durations, highlighting the spread of file lengths. The summary also features a bar chart that displays the number of files each speaker recorded, focusing on the top 20 speakers. Lastly, another bar chart illustrates the average duration of audio files for each word label, offering insights into whether some words tend to have consistently longer or shorter recordings.

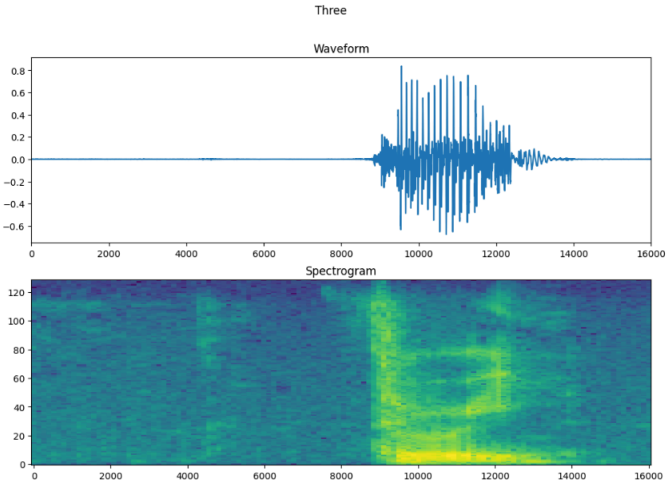
| | Duration (seconds) | Sampling Rate (Hz) |
|---|---|---|
| count | 501.000000 | 501.0 |
| mean | 0.983728 | 16000.0 |
| std | 0.073184 | 0.0 |
| min | 0.448000 | 16000.0 |
| 25% | 1.000000 | 16000.0 |
| 50% | 1.000000 | 16000.0 |
| 75% | 1.000000 | 16000.0 |
| max | 1.000000 | 16000.0 |

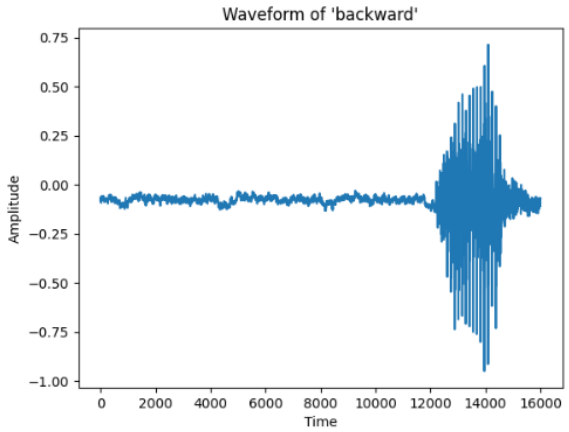


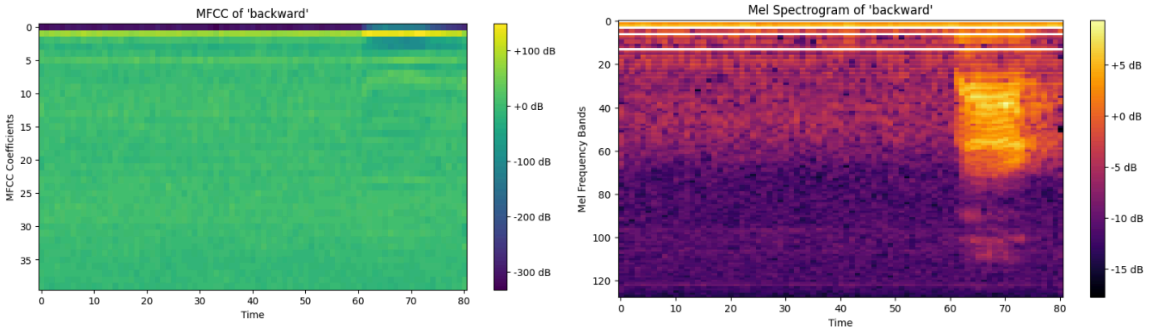

Distribution of Files per Word
Number of Files
0
500
1000
1500
2000
2500
3000
3500
4000
zero
five
yes
seven
no
nine
down
one
go
two
stop
six
on
left
eight
right
off
four
three
up
dog
wow
house
marvin
bird
happy
cat
sheila
bed
tree
backward
visual
follow
learn
forward
_background_noise_
Words

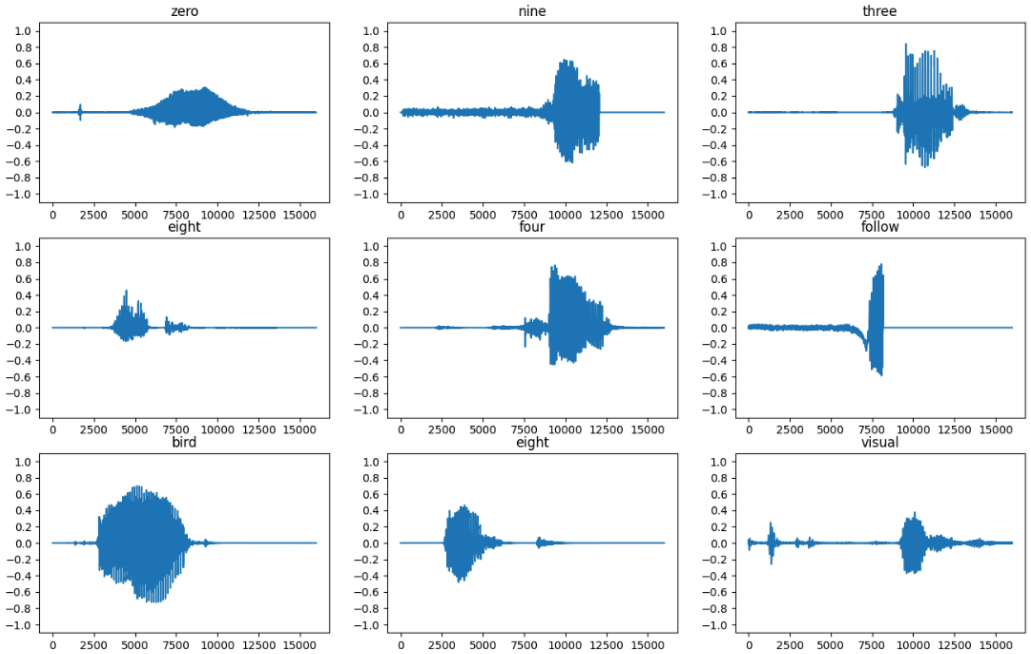

Top 20 Speakers by Number of Files
Number of Files
0
50
100
150
200
250
300
c50f55b8
893705bb
cce7416f
2aca1e72
ddedba85
b5cf6ea8
c1d39ce8
4c6167ca
ef3367d9
5f9cd2eb
5ebc1cda
b66f4f93
b4ea0d9a
ab46af55
de89e2ca
4c6944d6
c120e80e
25e95412
57152045
98ea0818
Speaker ID

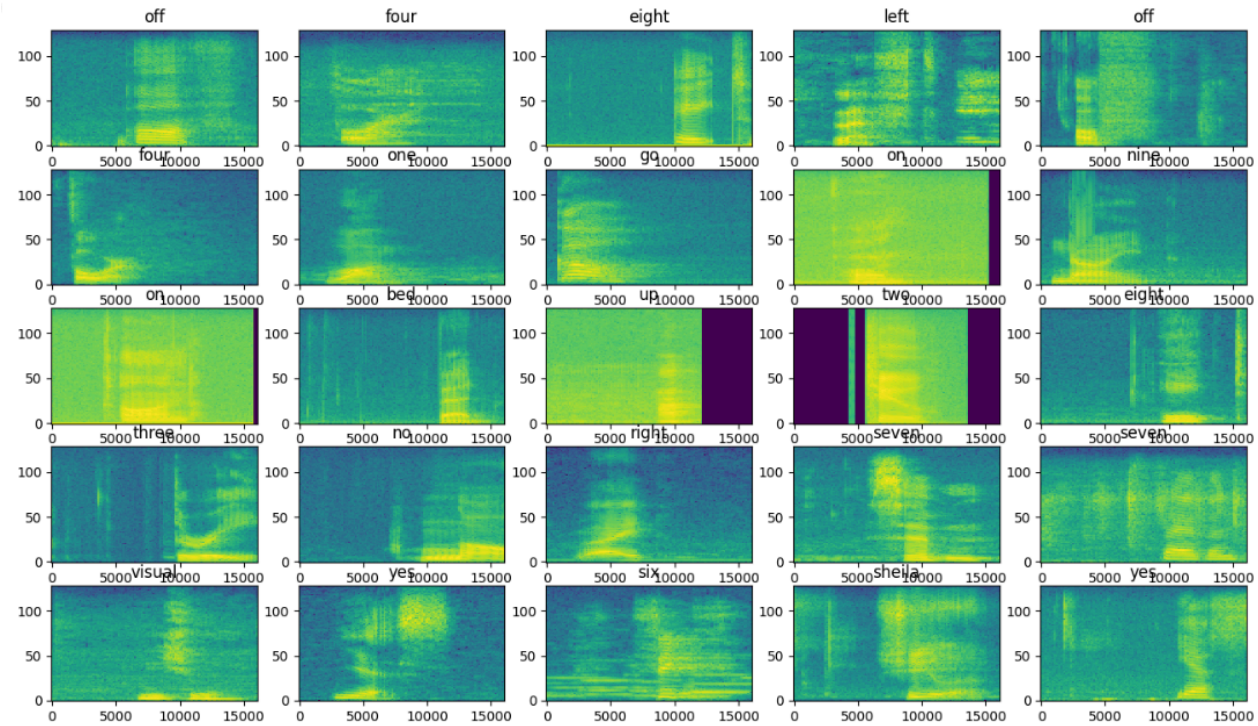

Average Duration of Audio Files per Word
Average Duration (seconds)
0
10
20
30
40
50
60
bird
cat
dog
down
eight
five
follow
forward
four
go
happy
house
learn
left
marvin
nine
no
off
on
one
right
seven
sheila
six
stop
three
tree
two
up
visual
wow
yes
zero
Words


Three
Waveform
Spectrogram


Waveform of 'backward'
Amplitude
Time


MFCC of 'backward'
MFCC Coefficients
Time
Mel Spectrogram of 'backward'
Mel Frequency Bands
Time



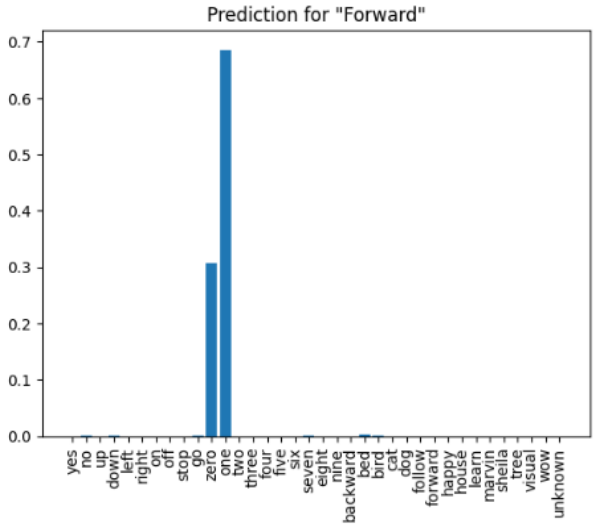

## Evaluation Metrics:

The model's performance was evaluated using several key metrics: accuracy, which measures the proportion of correctly classified instances among all predictions, and loss, which quantifies the difference between the predicted and actual values. The accuracy metric directly assesses the model's effectiveness in recognizing and classifying voice commands. In contrast, the loss metric helps monitor the model's learning progress and detect potential

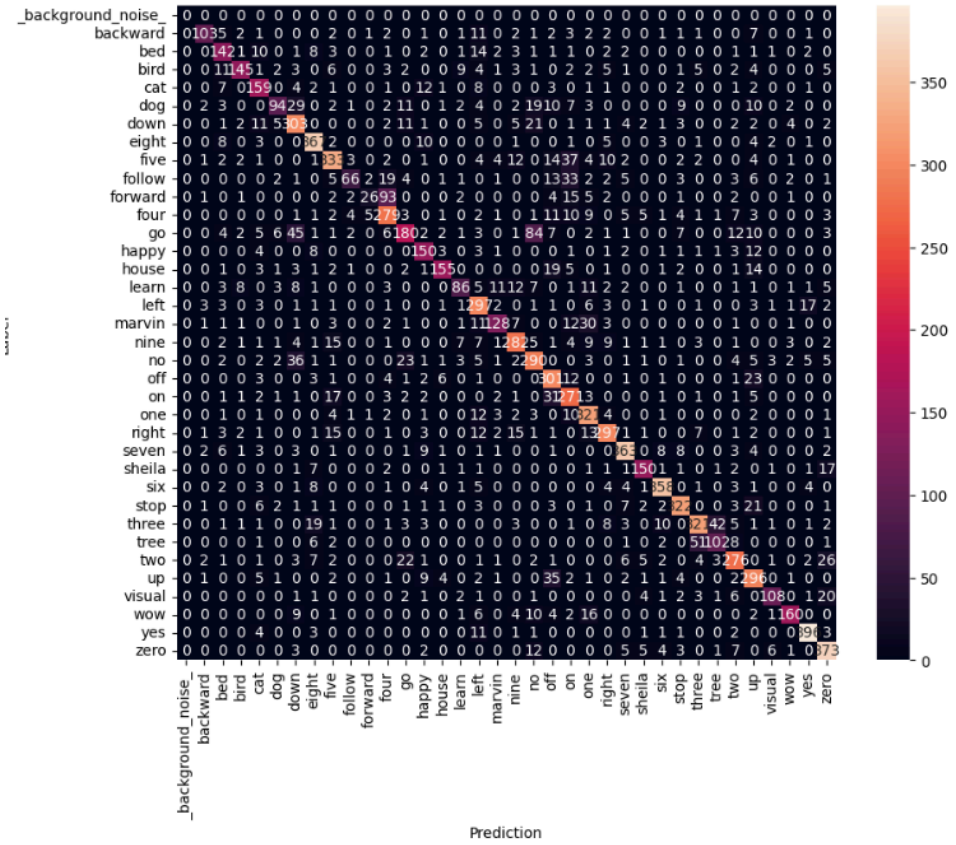
issues such as overfitting. These metrics collectively ensure a comprehensive evaluation of the model's performance on the training and validation datasets.

## Result Analysis:

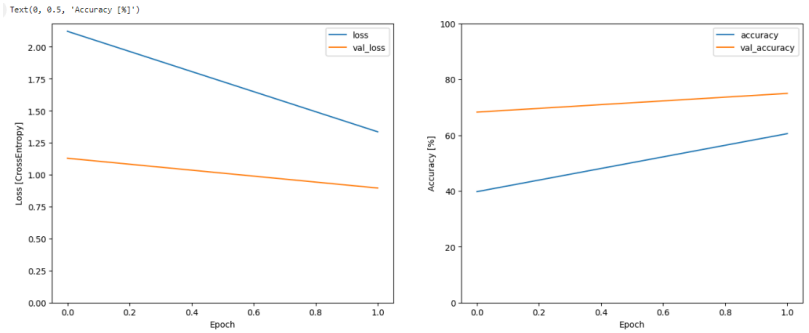
Following is the result analysis of the project:



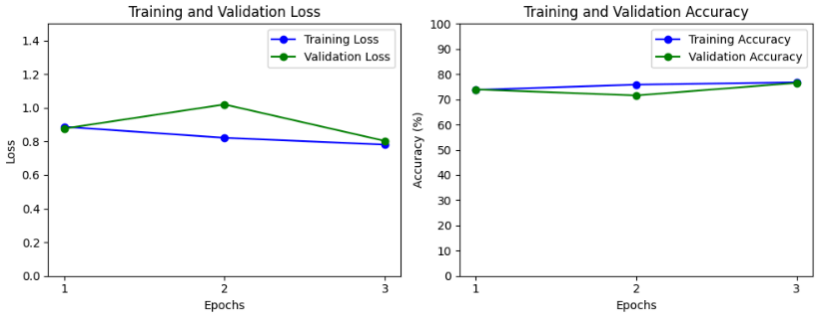

(Tested on a speech containing the term ‘forward’)



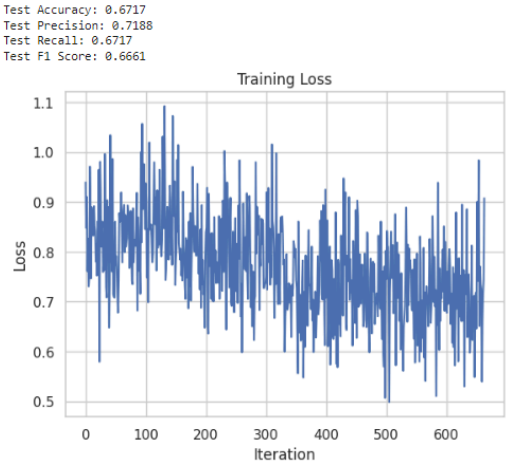
***Graphical Interpretation:***



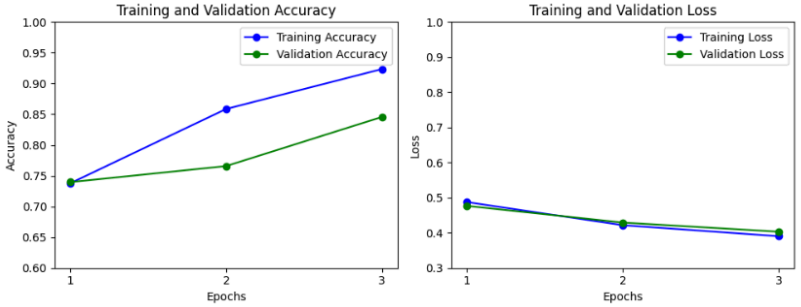

SCNN Model ( Pre-trained model on the original dataset)




SCNN Model ( Fine-Tuned model on the custom dataset)




M5 CNN Model ( Pre-trained model on the original dataset)




M5 CNN Model ( Fine-Tuuned Pre-trained model on the original dataset)

## Conclusion:

In conclusion, the project successfully adapted pre-trained sequential CNN and M5 CNN models for custom voice command recognition by fine-tuning it on a personalized dataset of voice recordings. The integration of quantization-aware training facilitated optimizing the model for deployment on resource-constrained devices, achieving a balance between performance and computational efficiency. The fine-tuning process, enhanced by strategic callbacks for model checkpointing and early stopping, ensured robust model training and prevented overfitting. The final quantized model demonstrated effective recognition capabilities on both the original and custom datasets, highlighting its practical applicability in real-time, resource-efficient voice command systems. Future work could explore expanding the system's use to various applications such as smart home automation, where voice commands could control devices like lighting and temperature, or in wearable technology for hands-free interaction with digital assistants. Additionally, integrating the system with natural language processing could enable more complex voice interactions and commands.