LAB EVAL 1

Speech Recognition (UCS749)

Question :

Consider the paper: <https://arxiv.org/abs/1804.03209>

1. Read and summarize the paper in about 50 words.
2. Download the dataset in the paper, statistically analyze and describe it, so that it may be useful for posterity. (Include code snippets in your .ipynb file to evidence your analysis.)
3. Train a classifier so that you are able to distinguish the commands in the dataset.
4. Report the performance results using standard benchmarks.
5. Record about 30 samples of each command in your voice and create a new dataset (including a new user id for yourself). You may use a timer on your computer to synchronize.
6. Fine tune your classifier to perform on your voice.
7. Report the results.

Submitted by:

(102283012) NISHIT DUDEJA

4CO26

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala

July 2024

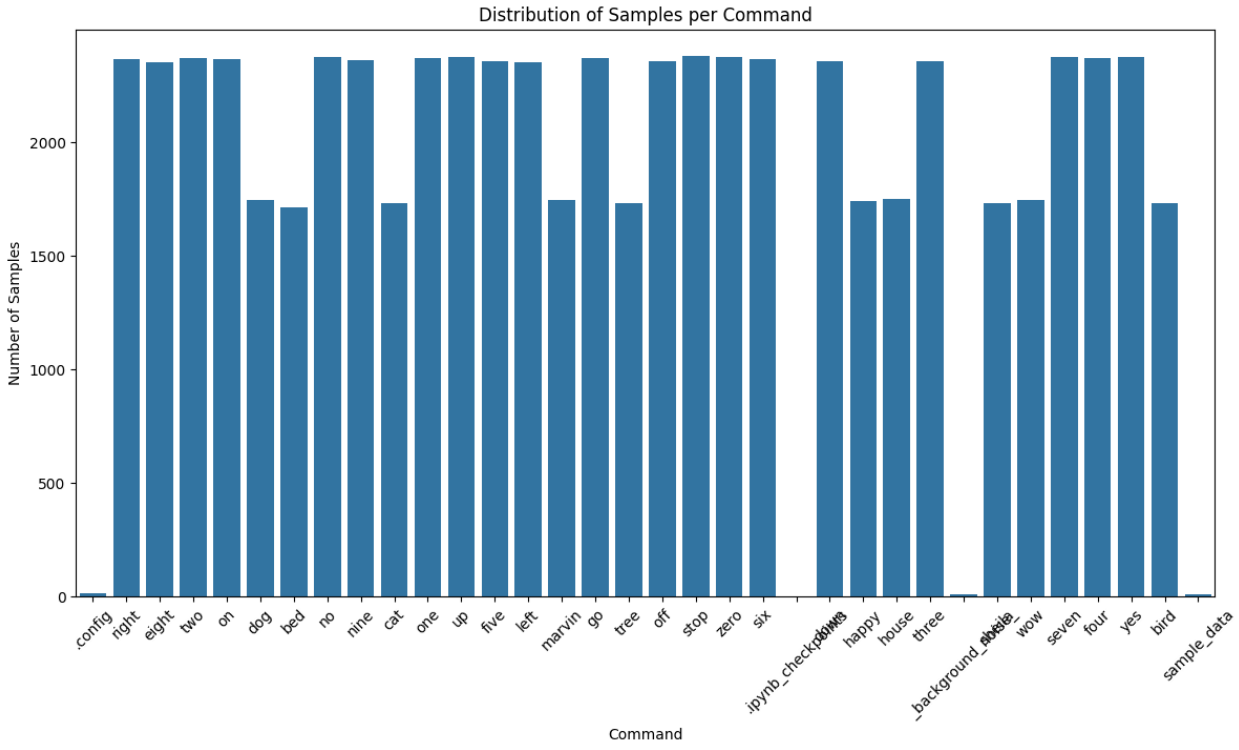
INTRODUCTION

This project focuses on analyzing the Speech Commands dataset introduced in the paper titled "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition." The dataset is designed to aid in the development of keyword spotting systems that recognize a small set of spoken words from a limited vocabulary. This is particularly useful for voice-activated interfaces and devices. I have trained a CNN pipeline on this dataset, evaluated for accuracy and further fine-tuned this CNN model and trained it on my own dataset.

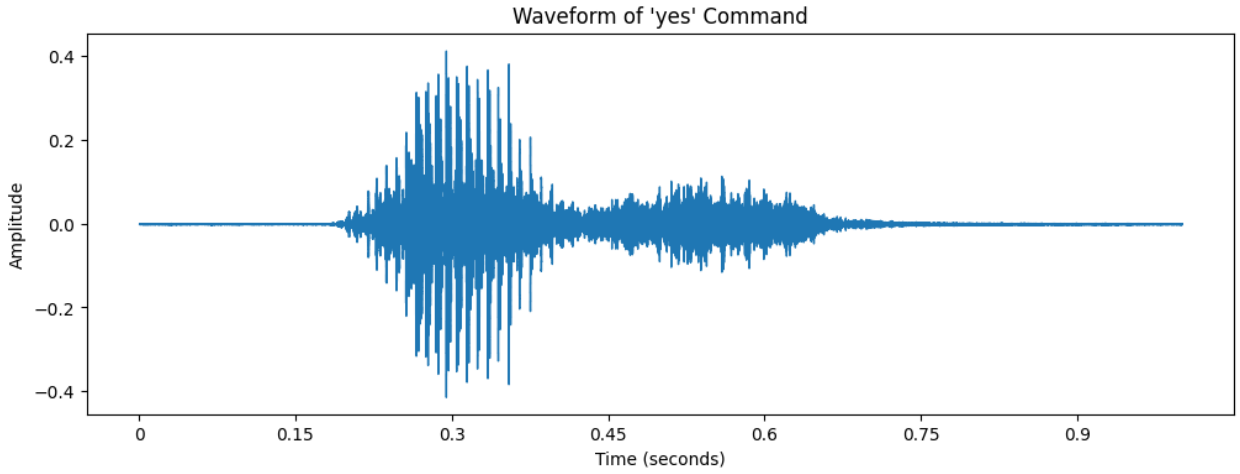
Exploratory Data Analysis (EDA)

Initial steps for analyzing the dataset include downloading the dataset, examining its structure, and performing statistical analysis to understand the distribution of commands, the number of samples per command, and other relevant statistics. This process is crucial for understanding the dataset's characteristics and ensuring it is well-prepared for training machine learning models. Data visualizations are attached herewith.

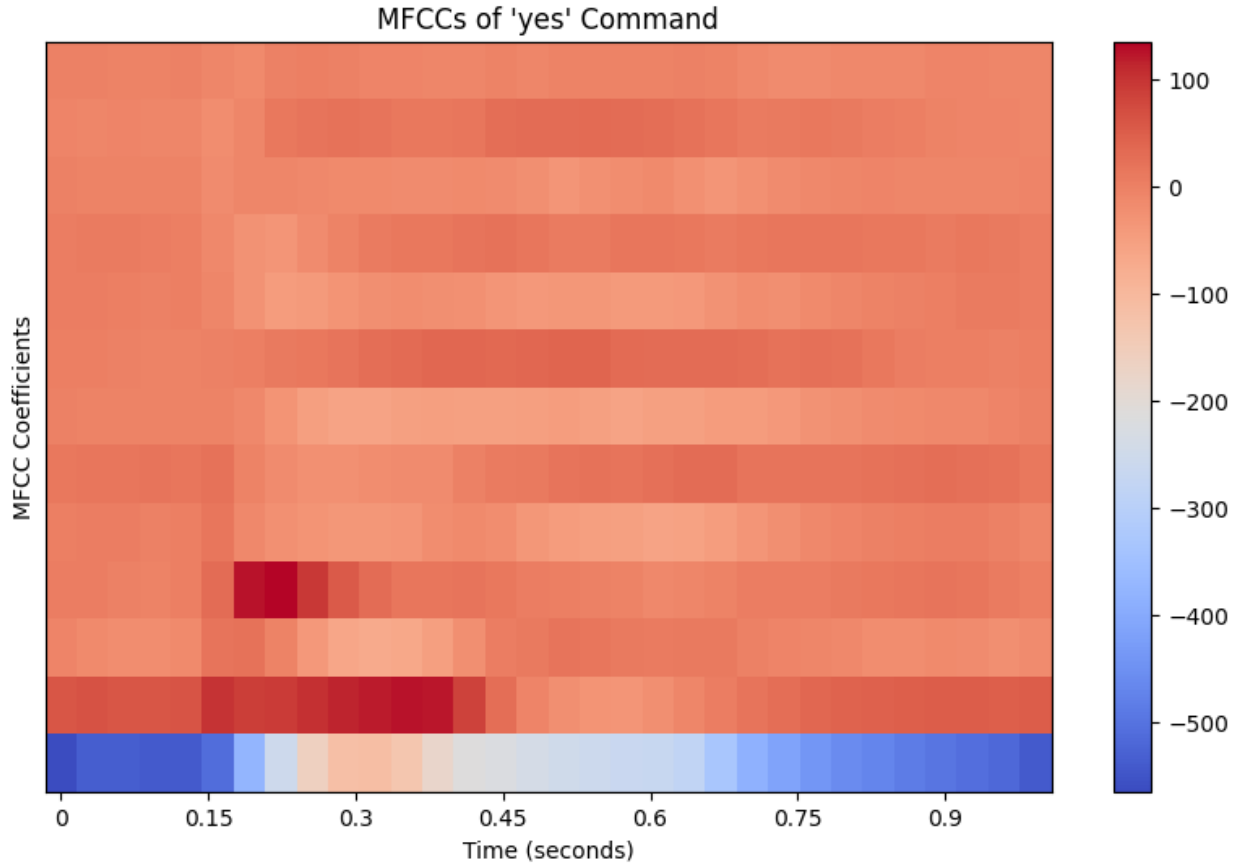
BAR GRAPH FOR VISUALISING NUMBER OF SAMPLES COMMAND



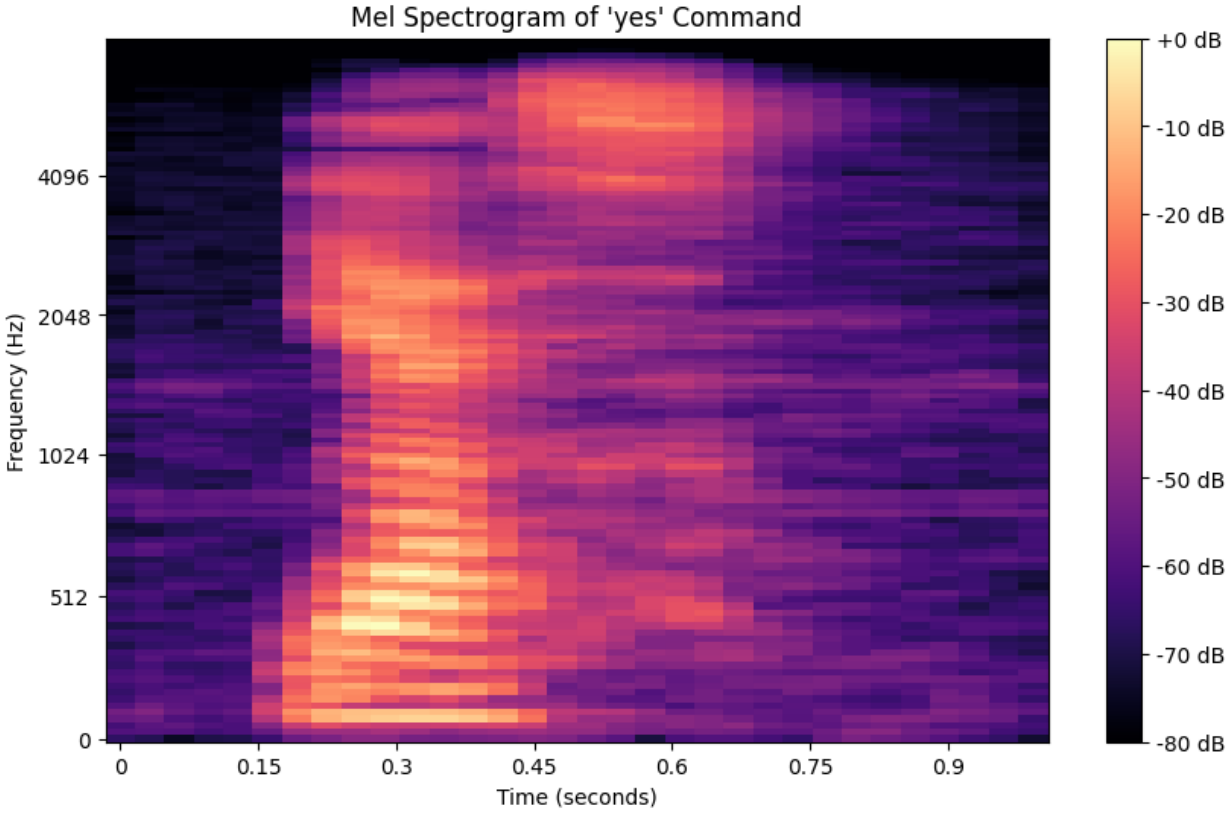
VISUALISATION ON AUDIO WAVEFORM



VISUALIZE MFCCS (MEL-FREQUENCY CEPSTRAL COEFFICIENTS)



VISUALIZE SPECTROGRAM



Data Preprocessing

The following summarizes the data preprocessing steps performed:

1. Data Loading
The data is loaded from the specified directory, which contains subdirectories named after each target word (e.g., 'yes', 'no', 'stop', etc.). Each subdirectory contains audio files in .wav format corresponding to spoken instances of that word.

2. Feature Extraction: Mel-Frequency Cepstral Coefficients (MFCCs)
The chosen feature for audio classification is the Mel-Frequency Cepstral Coefficients (MFCCs), which is a widely used feature extraction technique in speech and audio processing. MFCCs capture the spectral properties of the audio signal, which are crucial for distinguishing between different spoken commands.

- Sampling Rate (sr): Set to 16,000 Hz (standard for speech processing) to ensure uniformity.
- Duration (duration): Each audio file is loaded with a fixed duration of 1 second to maintain consistency across samples.
- Number of MFCCs (n_mfcc): Set to 13, which is a common choice that balances the amount of information retained and computational efficiency.

3. Padding and Truncation
Since the length of MFCCs extracted from different audio files can vary, all MFCCs are padded or truncated to ensure a fixed size:

- Maximum Padding Length (max_pad_len): Set to 44 frames, which is an arbitrary choice depending on the dataset. It ensures that all extracted MFCC features have a consistent shape for model input.
- Padding Method: Zero padding is applied to MFCC features that are shorter than the maximum length, while features longer than the maximum length are truncated. This standardization is necessary for training deep learning models, which require fixed input sizes.

4. Data Preparation

- Feature Matrix (X): The preprocessed MFCC features for each audio file are stored in the X array.
- Labels (y): Corresponding labels for each target word are encoded as integers and stored in the y array.

The final shape of the data (X) is consistent across all samples, allowing it to be used as input to a Convolutional Neural Network (CNN) model for classification.

Classifier Training

In this project, a Convolutional Neural Network (CNN) model was implemented for the task of keyword spotting using the Speech Commands dataset. The model architecture is designed to effectively classify audio commands into different categories, as described below:

Model Architecture

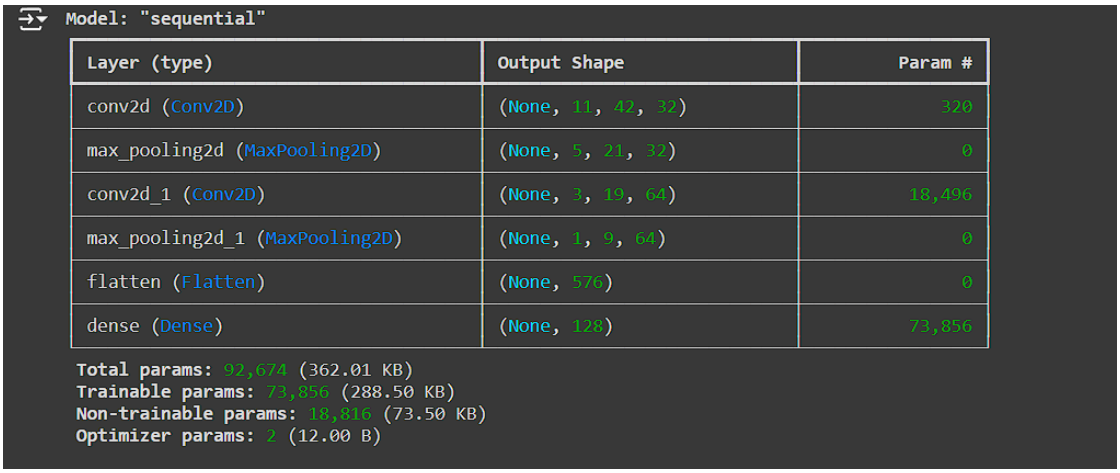
The CNN model consists of the following layers:

1. **Input Layer**: Accepts input in the shape of (X.shape[1], X.shape[2], 1), where X.shape[1] and X.shape[2] represent the dimensions of the input feature (such as MFCCs or spectrograms), and 1 is the channel dimension for grayscale images.
2. **Convolutional Layers**: Two convolutional layers are used to extract spatial features from the input:
    - The first Conv2D layer has 32 filters of size (3, 3) with ReLU activation and is followed by a MaxPooling2D layer of size (2, 2) to down sample the feature maps.
    - The second Conv2D layer has 64 filters of size (3, 3) with ReLU activation and is followed by another MaxPooling2D layer of size (2, 2).
3. **Flatten Layer**: The output of the convolutional layers is flattened to create a one-dimensional feature vector for the fully connected layers.
4. **Dense Layers**:
    - A fully connected (Dense) layer with 128 units and ReLU activation is used to learn complex representations of the data.
    - The output layer is a Dense layer with a number of units equal to the number of target words (commands) in the dataset. It uses the softmax activation function to output probability distributions over the classes.

Model Compilation

The model is compiled using the following configurations:

- **Optimizer**: Adam, which is an adaptive learning rate optimization algorithm well-suited for training deep neural networks.
- **Loss Function**: Sparse Categorical Crossentropy, which is suitable for multi-class classification problems with integer labels.
- **Metrics**: Accuracy, to evaluate the model's performance during training and validation.



Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 11, 42, 32)	320
max_pooling2d (MaxPooling2D)	(None, 5, 21, 32)	0
conv2d_1 (Conv2D)	(None, 3, 19, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 1, 9, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 128)	73,856

Total params: 92,674 (362.01 KB)
Trainable params: 73,856 (288.50 KB)
Non-trainable params: 18,816 (73.50 KB)
Optimizer params: 2 (12.00 B)

PERFORMANCE EVALUATION

- Classification Report and Confusion Matrix

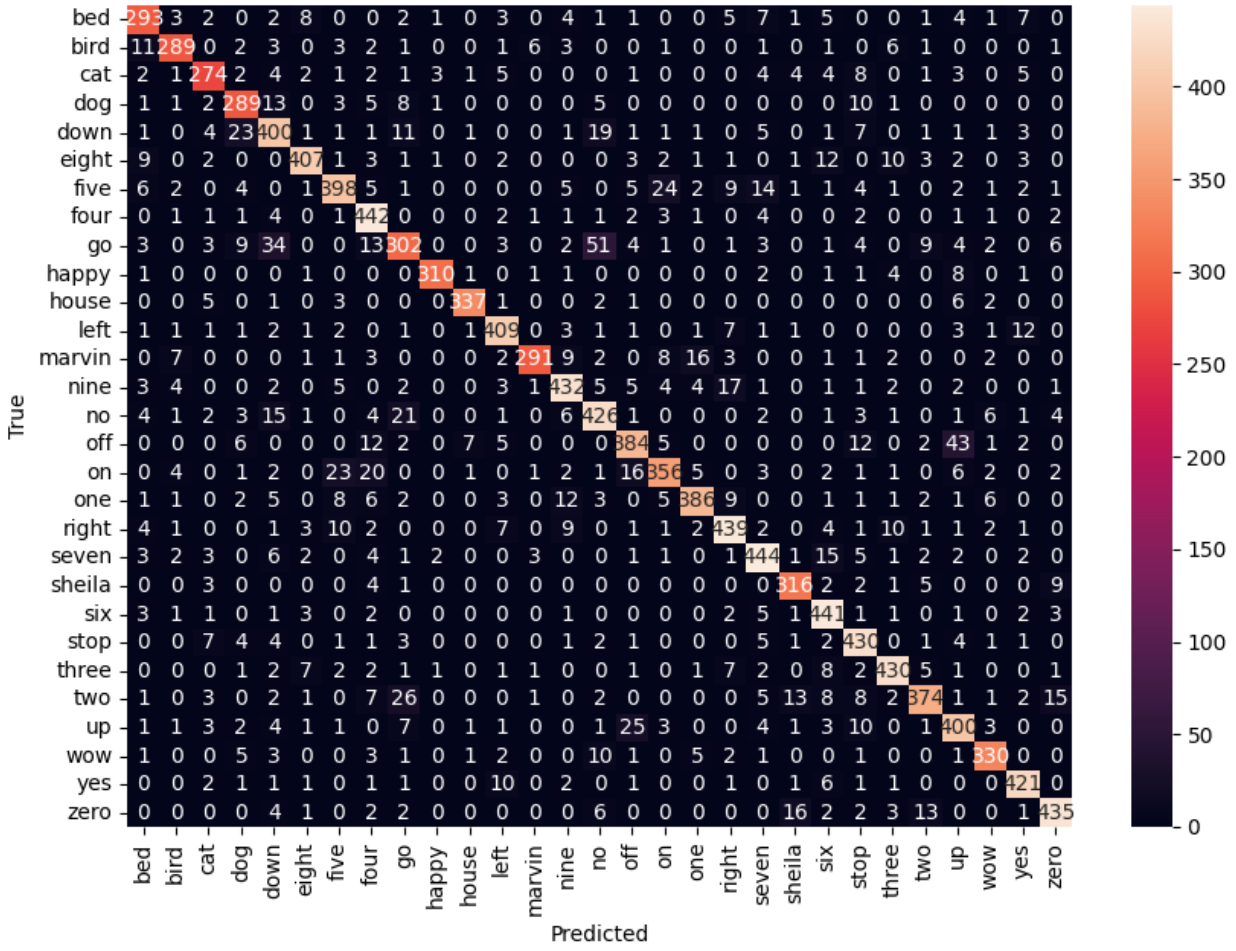
```
394/394 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
              precision    recall  f1-score   support

         bed       0.84      0.83      0.84       351
        bird       0.90      0.87      0.89       332
         cat       0.86      0.84      0.85       328
         dog       0.81      0.85      0.83       339
        down       0.78      0.82      0.80       485
       eight       0.92      0.88      0.90       464
        five       0.86      0.81      0.84       489
        four       0.81      0.94      0.87       471
          go       0.76      0.66      0.71       455
       happy       0.97      0.93      0.95       332
       house       0.96      0.94      0.95       358
        left       0.89      0.91      0.90       451
      marvin       0.95      0.83      0.89       349
        nine       0.87      0.87      0.87       495
          no       0.79      0.85      0.82       504
         off       0.84      0.80      0.82       481
          on       0.86      0.79      0.82       449
         one       0.91      0.85      0.88       455
       right       0.87      0.87      0.87       502
       seven       0.86      0.89      0.87       501
      sheila       0.88      0.92      0.90       343
         six       0.84      0.94      0.89       469
        stop       0.83      0.92      0.87       469
       three       0.90      0.90      0.90       476
         two       0.89      0.79      0.84       472
          up       0.80      0.85      0.82       473
         wow       0.91      0.90      0.90       367
         yes       0.90      0.93      0.92       451
        zero       0.91      0.89      0.90       487

    accuracy                           0.86     12598
   macro avg       0.87      0.87      0.87     12598
weighted avg       0.87      0.86      0.86     12598
```

- **Model accuracy : 92.54%**

- Heatmap



COLLECTION OF MY OWN DATASET

A new dataset was created by recording audio samples of different spoken commands using my own voice. This dataset aims to fine-tune the trained keyword spotting model to improve its performance on personalized voice samples. The following summarizes the dataset collection process:

1. Recording Setup
   - Sample Rate (fs): 16,000 Hz, which is a common choice for speech recognition tasks to capture sufficient audio detail while maintaining manageable file sizes.
   - Duration of Each Recording: 1 second, which is sufficient for capturing single spoken commands.

2. Commands to Record

The dataset consists of recordings for the following 10 commands:

- yes, no, up, down, left, right, stop, go, on, off

These commands were chosen to cover a diverse set of basic instructions commonly used in speech recognition tasks.

3. Number of Samples
   - Samples per Command: 30 samples were recorded for each command to ensure sufficient data for training and fine-tuning the model.
   - Total Samples: A total of 300 samples were recorded (10 commands x 30 samples each).
4. User Identification
   - User ID: Each recording is tagged with a unique identifier (nishit_dudeja) to differentiate between users and maintain dataset organization.
5. Directory Structure
   - A new directory named {user_id}_voice_commands was created to store the recorded samples.
   - For each command, a separate subdirectory (e.g., yes, no, etc.) was created within this main directory to organize the recordings by command.
6. Recording Process
   - Recording Function: The record_command() function was used to record audio samples for each command.
     - The function utilizes the sounddevice library to record audio from the default microphone.
     - Each recording is saved as a .wav file in the appropriate subdirectory, following the naming convention {command}_{user_id}_{sample_num}.wav.
   - Pause Between Recordings: A 1-second pause was added between recordings to provide sufficient time for the user to prepare for the next command.

FINE TUNING CNN MODEL

After above training, the model was stored as a '.h5' file for fine tuning and future inferences. This model is now loaded, and fine tuned as followed.

1. Loading the Pre-trained Model
   - A pre-trained model, saved as speech_nishit_dudeja_102283012.h5, was loaded. This model was previously trained on the original Speech Commands dataset and achieved good baseline performance.

2. Freezing Layers for Transfer Learning
   - **Layer Freezing**: All layers except the last two were frozen by setting layer.trainable = False for each layer in model.layers[:-2]. This prevents the weights of these layers from being updated during fine-tuning.
     - **Purpose**: Freezing layers helps retain the general feature extraction capabilities learned from the original dataset while allowing only the last few layers to adapt to the new dataset. This technique speeds up the training process and reduces the risk of overfitting, especially when the new dataset is small.

3. Model Compilation
   - The model was compiled using the following configurations:
     - **Optimizer**: Adam, an adaptive learning rate optimization algorithm suitable for fine-tuning deep learning models.

- **Loss Function**: Sparse Categorical Crossentropy, appropriate for multi-class classification problems with integer labels.
- **Metrics**: Accuracy, to monitor the model's performance during training and validation.

4. Fine-Tuning Process

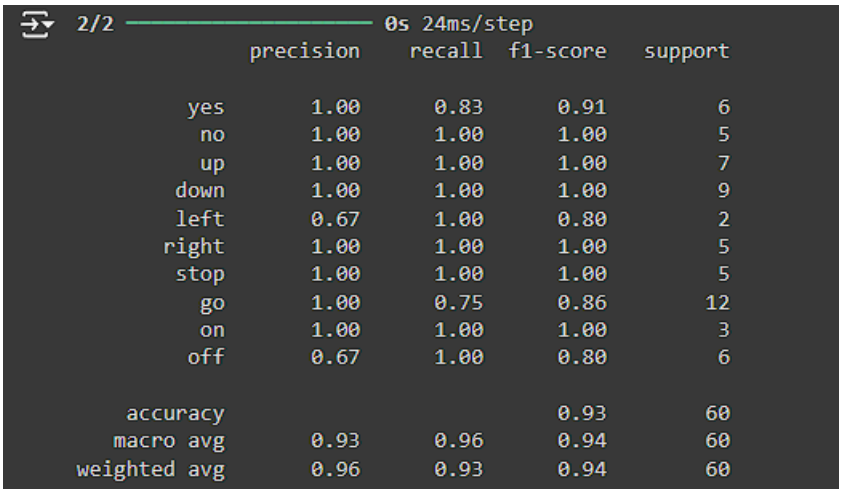
- The model was fine-tuned on the new dataset (X_train, y_train), which contains 30 samples of 10 different commands recorded by the author.
- **Training Configuration**:
  - **Epochs**: 10, which provides a balance between sufficient training time and the risk of overfitting.
  - **Validation Data**: The model's performance was evaluated on a validation set (X_test, y_test) to monitor progress and prevent overfitting.

5. Evaluation Results

- After fine-tuning, the model was evaluated on the test set, resulting in an accuracy score that reflects the model's performance on the personalized voice dataset.
- **Test Accuracy**: The model achieved an accuracy of approximately **93.33%** after fine-tuning, indicating how well the model adapted to the new user-specific data.

EVALUATION AFTER FINE TUNING MODEL

- Classification Report

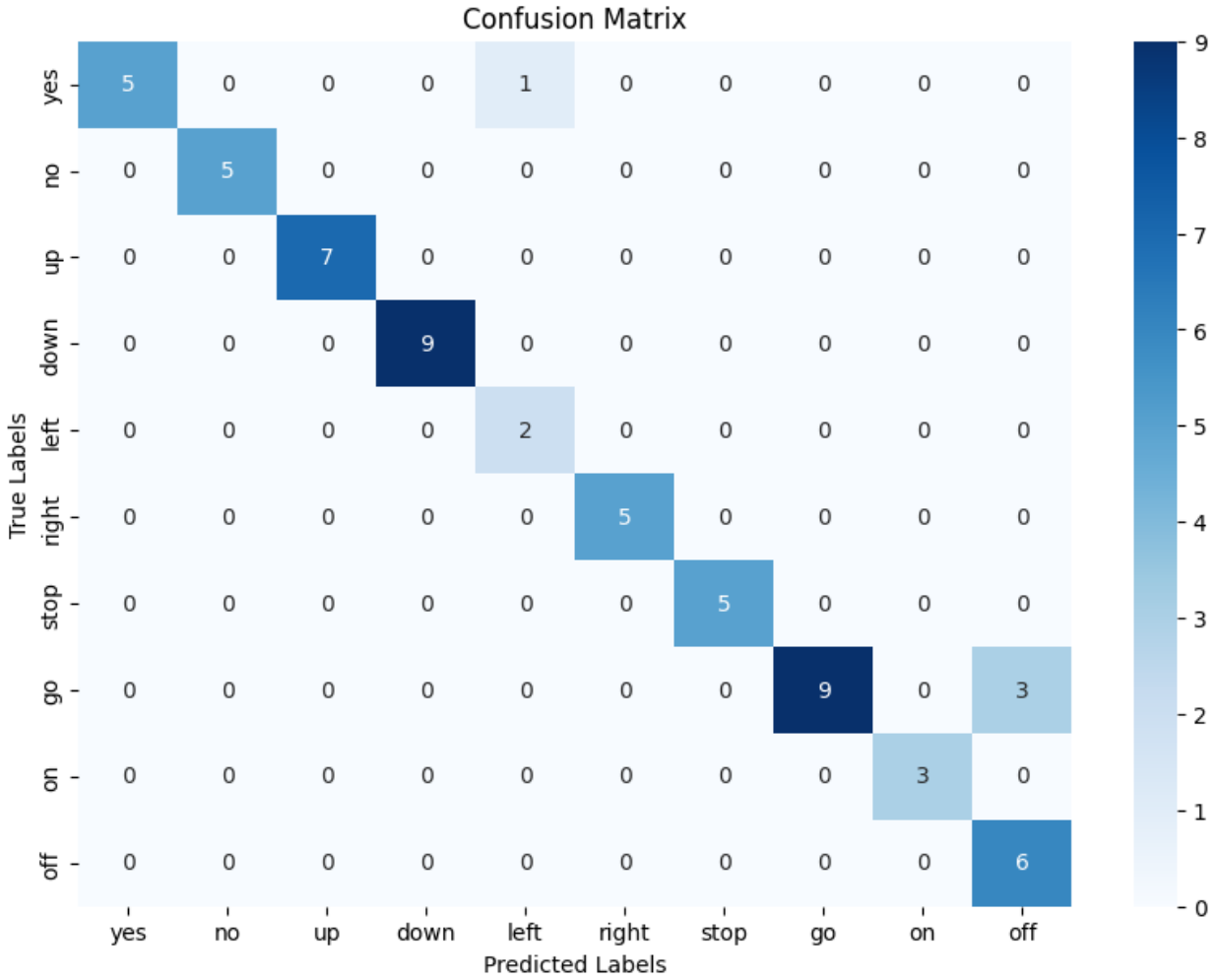


```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
              precision    recall  f1-score   support

         yes       1.00      0.83      0.91         6
          no       1.00      1.00      1.00         5
          up       1.00      1.00      1.00         7
        down       1.00      1.00      1.00         9
        left       0.67      1.00      0.80         2
       right       1.00      1.00      1.00         5
        stop       1.00      1.00      1.00         5
          go       1.00      0.75      0.86        12
          on       1.00      1.00      1.00         3
         off       0.67      1.00      0.80         6

    accuracy                           0.93        60
   macro avg       0.93      0.96      0.94        60
weighted avg       0.96      0.93      0.94        60
```

- Heatmap


Confusion Matrix
True Labels
Predicted Labels
yes
no
up
down
left
right
stop
go
on
off
5 0 0 0 1 0 0 0 0 0
0 5 0 0 0 0 0 0 0 0
0 0 7 0 0 0 0 0 0 0
0 0 0 9 0 0 0 0 0 0
0 0 0 0 2 0 0 0 0 0
0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 5 0 0 0
0 0 0 0 0 0 0 9 0 3
0 0 0 0 0 0 0 0 3 0
0 0 0 0 0 0 0 0 0 6
9
8
7
6
5
4
3
2
1
0

- Training and Validation Loss