Horn Detection: Keyword Spotter Model

BTP Presentation 2023

Ananya Aakriti: 2020CS10321
Mehul Kamboj: 2020CS10604
Yashwanth: 2020CS10406

What is a Keyword Spotting Algorithm?

A keyword spotting algorithm is a specialized technique used in audio processing and pattern recognition to identify specific keywords or phrases within a larger body of audio data.

Purpose: It is designed to detect predefined keywords or patterns of interest within audio recordings, making it an essential tool for various applications where audio recognition is needed.

Unlike traditional speech recognition systems that transcribe spoken words into text, keyword spotting focuses on recognizing specific keywords or sounds without the need for full transcription. These algorithms can operate in real-time, allowing for immediate keyword detection, or in offline mode for batch processing and analysis.

Applications of Keyword Spotting Algorithm?

01 **Security and Surveillance:** Detecting specific phrases or sounds in security camera audio.

02 **Automotive Safety:** Recognizing horn sounds, sirens, or other critical audio cues.

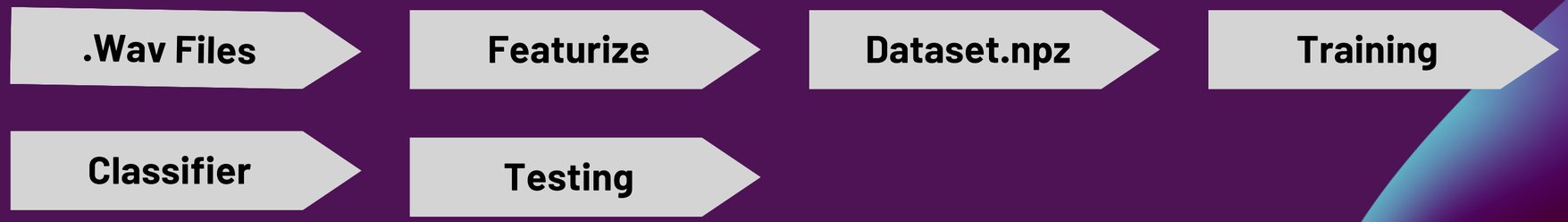
03 **Voice Assistants:** Triggering actions based on wake words (e.g., "Hey Siri" or "Alexa").

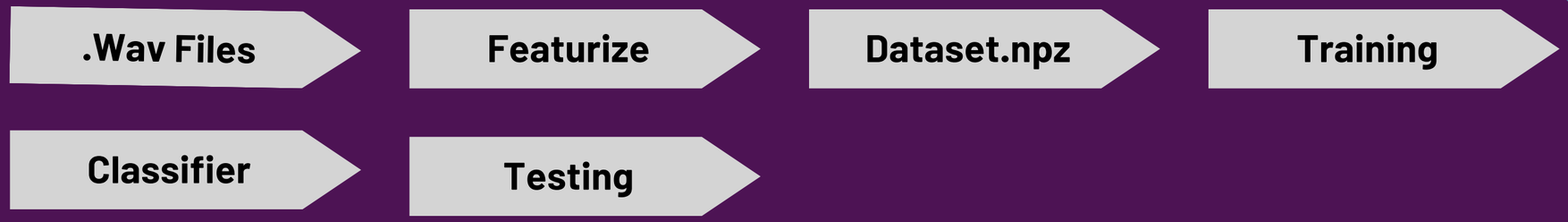
04 **Industrial Automation:** Identifying specific equipment malfunctions based on sound patterns.

05 **Noise Pollution Monitoring:** Detecting and categorizing environmental noise sources.

Explored Models

- https://microsoft.github.io/ELL/tutorials/Training-audio-keyword-spotter-with-pytorch/%E2%80%8B
- https://www.tensorflow.org/tutorials/audio/simple_audio%E2%80%8B
- After some consideration, we chose the first model which uses PyTorch and PyAudio.
- The picture below illustrates the processes followed by us and these are described further.





- wav files: These store audio in the Waveform Audio File Format, for recording and audio processing.
- Featurization: Key features are extracted from audio signals during featurization for analysis in audio processing.
- dataset.npz: After featurization, audio data undergoes conversion into a machine-learning-friendly format, often as a ".npz" NumPy archive, encompassing featurized data and labels.
- Training: The featurized dataset is utilized to train a keyword-spotting ML model.
- Classifier: This is the trained ML model that predicts keyword presence based on the extracted audio features.
- Testing: The trained classifier is evaluated on new audio clips to assess its performance and its ability to generalize to unseen data.

Brief Overview of Model Training Steps

- ELL Setup (Embedded Learning Library).
- ELL enables us to design and deploy intelligent machine-learned models onto single-board computers, like Raspberry Pi and Arduino.
- PyTorch And PyAudio Setup: PyTorch is used for deep learning and machine learning tasks, while PyAudio is used for audio input/output and audio processing applications.
- Setting up the Dataset to be sorted in Training, Validation & Testing dataset in the ratio of 7:2:1.
- Creation and setup of Featurizer Model was done to filter and modify dataset to start training.
- The next step involved training the Keyword Spotter to create a ONNX model and getting accuracies.

Data Collection

- The first step in developing a machine learning model is to collect a dataset .

- In this case, the dataset would need to contain audio recordings of honk sounds, as well as audio recordings of other types of sounds, such as traffic noise, pedestrian noise, and animal noises and many other sounds in the environment.

- We gathered a diverse collection of audio. However, the data we collected was not directly suitable for training a machine learning model.

- The audio recordings were too long, they contained noise, and they often had multiple honk sounds in a single recording.. Therefore, we needed to clean and trim the data before using it for training the model.

Why Trim Data Before Training?

There are several reasons why it is important to trim and clean audio data before training a machine learning model.

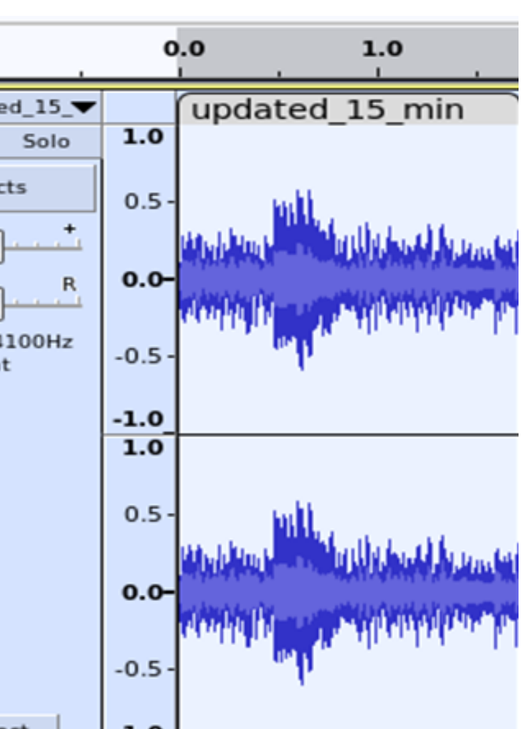
- Noise can interfere with the model's ability to learn the features of the target sound. For example, if the audio data contains a lot of noise, the model may learn to associate those noises with the target sound, which could lead to false positives.
- Trimming the recordings to individual honk sounds allows the model to focus on the target sound and ignore any other sounds that are present in the recording. This can help to improve the model's accuracy and precision.
- Trimming the recordings to a shorter length can make the training process faster and more efficient.

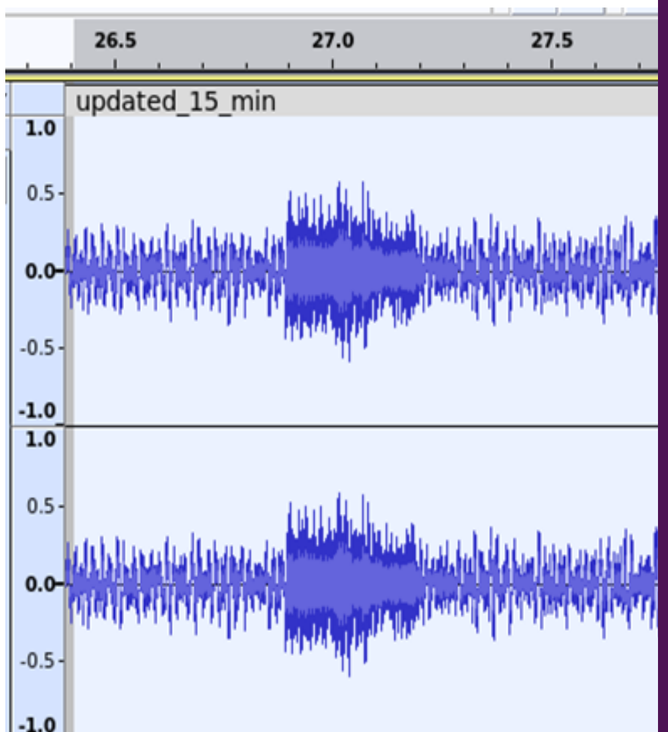
Data Preparation and Data Labelling

	HORN	NOHORN
Total number of audio files	229	872
Number of training files	160	610
Number of validation files	46	174
Number of testing files	23	88

- We used Audacity, a free and open-source audio editing software, to trim and denoise the recordings, and to split them into individual honk sounds.
- After preparing the data, we labeled it into two classes: Positive and Negative.
- Our positive data contains honk sounds made by different vehicles, such as cars, trucks, buses, and motorcycles.
- Our negative data contains a variety of negative sounds, such as engine sounds, traffic noise, pedestrian noise, animal noises, and many other sounds in the environment.

Identifying and labelling audio events in Audacity

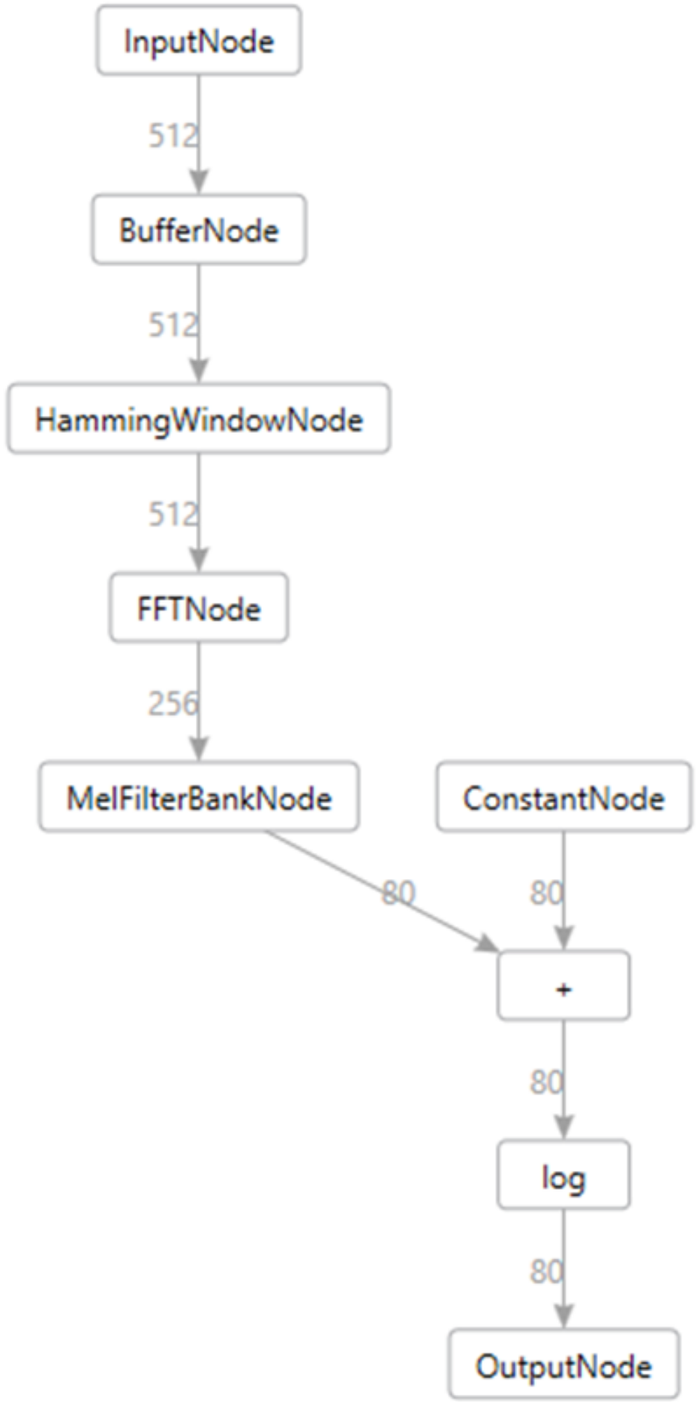




When we open an audio file in Audacity, we can see the waveform of the audio. The peaks of the waveform represent the loudest parts of the audio.

We can separate the parts of the waveform where there are peaks, because each peak represents a single audio event, such as a clap, a cough, or a gunshot.

Once we have separated the audio events, we can listen to them and label them as positive or negative, depending on the type of event.



Creating a Featurizer Model

- The featurizer model is a mel-frequency cepstrum (mfcc) audio transformer which preprocesses audio input, preparing it for use by the training process.
- This featurizer is created as an ELL model using the make featurizer command.
- We can preprocess all the audio files using this featurizer and create a compressed Numpy dataset with the result.
- Command: python make_dataset.py --list_file audio/training_list.txt --featurizer compiled_featurizer/mfcc --window_size 40 --shift 40

Training the Keyword Spotter

- Train the keyword spotter using the train_classifier.py script.
- PyTorch is used to train a GRU-based model.
- The trained model is exported as KeywordSpotter.onnx.
- Training involves epochs with loss and validation accuracy updates and the testing dataset contains files not seen during training.

```
Parameter Group 0
    alpha: 0
    centered: False
    eps: 1e-08
    lr: 0.001
    momentum: 0
    weight_decay: 1e-05
)
Epoch 0, Loss 0.518, Validation Accuracy 78.672, Learning Rate 0.001
Epoch 1, Loss 0.546, Validation Accuracy 80.391, Learning Rate 0.001
Epoch 2, Loss 0.453, Validation Accuracy 80.938, Learning Rate 0.001
Epoch 3, Loss 0.304, Validation Accuracy 80.859, Learning Rate 0.001
Epoch 4, Loss 0.332, Validation Accuracy 80.234, Learning Rate 0.001
Epoch 5, Loss 0.438, Validation Accuracy 80.625, Learning Rate 0.001
Epoch 6, Loss 0.403, Validation Accuracy 79.922, Learning Rate 0.001
Epoch 7, Loss 0.487, Validation Accuracy 80.469, Learning Rate 0.001
Epoch 8, Loss 0.416, Validation Accuracy 82.188, Learning Rate 0.001
Epoch 9, Loss 0.436, Validation Accuracy 81.016, Learning Rate 0.001
Epoch 10, Loss 0.458, Validation Accuracy 82.344, Learning Rate 0.001
Epoch 11, Loss 0.335, Validation Accuracy 82.266, Learning Rate 0.001
Epoch 12, Loss 0.354, Validation Accuracy 82.266, Learning Rate 0.001
Epoch 13, Loss 0.334, Validation Accuracy 82.031, Learning Rate 0.001
Epoch 14, Loss 0.389, Validation Accuracy 79.844, Learning Rate 0.001
Epoch 15, Loss 0.329, Validation Accuracy 81.719, Learning Rate 0.001
Epoch 16, Loss 0.400, Validation Accuracy 79.766, Learning Rate 0.001
Epoch 17, Loss 0.343, Validation Accuracy 82.578, Learning Rate 0.001
Epoch 18, Loss 0.337, Validation Accuracy 82.891, Learning Rate 0.001
Epoch 19, Loss 0.422, Validation Accuracy 82.578, Learning Rate 0.001
Epoch 20, Loss 0.311, Validation Accuracy 82.734, Learning Rate 0.001
Epoch 21, Loss 0.391, Validation Accuracy 81.953, Learning Rate 0.001
Epoch 22, Loss 0.499, Validation Accuracy 82.656, Learning Rate 0.001
Epoch 23, Loss 0.398, Validation Accuracy 82.031, Learning Rate 0.001
Epoch 24, Loss 0.353, Validation Accuracy 82.266, Learning Rate 0.001
Epoch 25, Loss 0.427, Validation Accuracy 82.891, Learning Rate 0.001
Epoch 26, Loss 0.352, Validation Accuracy 82.656, Learning Rate 0.001
Epoch 27, Loss 0.457, Validation Accuracy 82.500, Learning Rate 0.001
Epoch 28, Loss 0.671, Validation Accuracy 81.641, Learning Rate 0.001
Epoch 29, Loss 0.295, Validation Accuracy 82.578, Learning Rate 0.001
Trained in 83.45 seconds
Training accuracy = 84.297 %
Evaluating GRU keyword spotter using 675 rows of featurized test audio...
Saving evaluation results in './results.txt'
MEHUL PRINT :  640
Testing accuracy = 82.031 %
saving onnx file: GRU128KeywordSpotter.onnx
```

Hardware Specifications

Processor
AMD Ryzen™ 7 5700U Processor (8 Cores/ 16 Threads, 1.80 GHz up to 4.30 GHz)

Operating System
Ubuntu 22.04.3 LTS

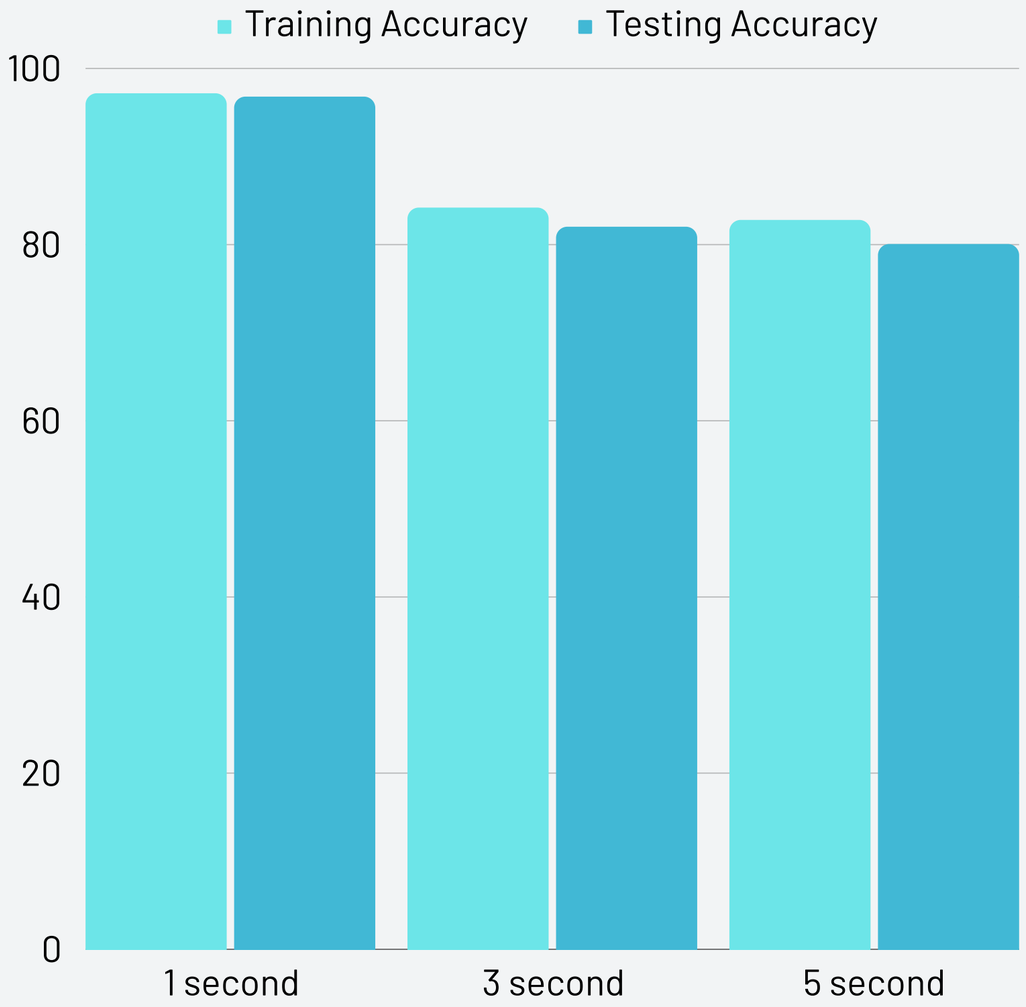
Memory
16 GB (8 GB Soldered DDR4 3200MHz + 8 GB SO-DIMM DDR4 3200MHz)

Hard Drive
512 GB SSD M.2 2242

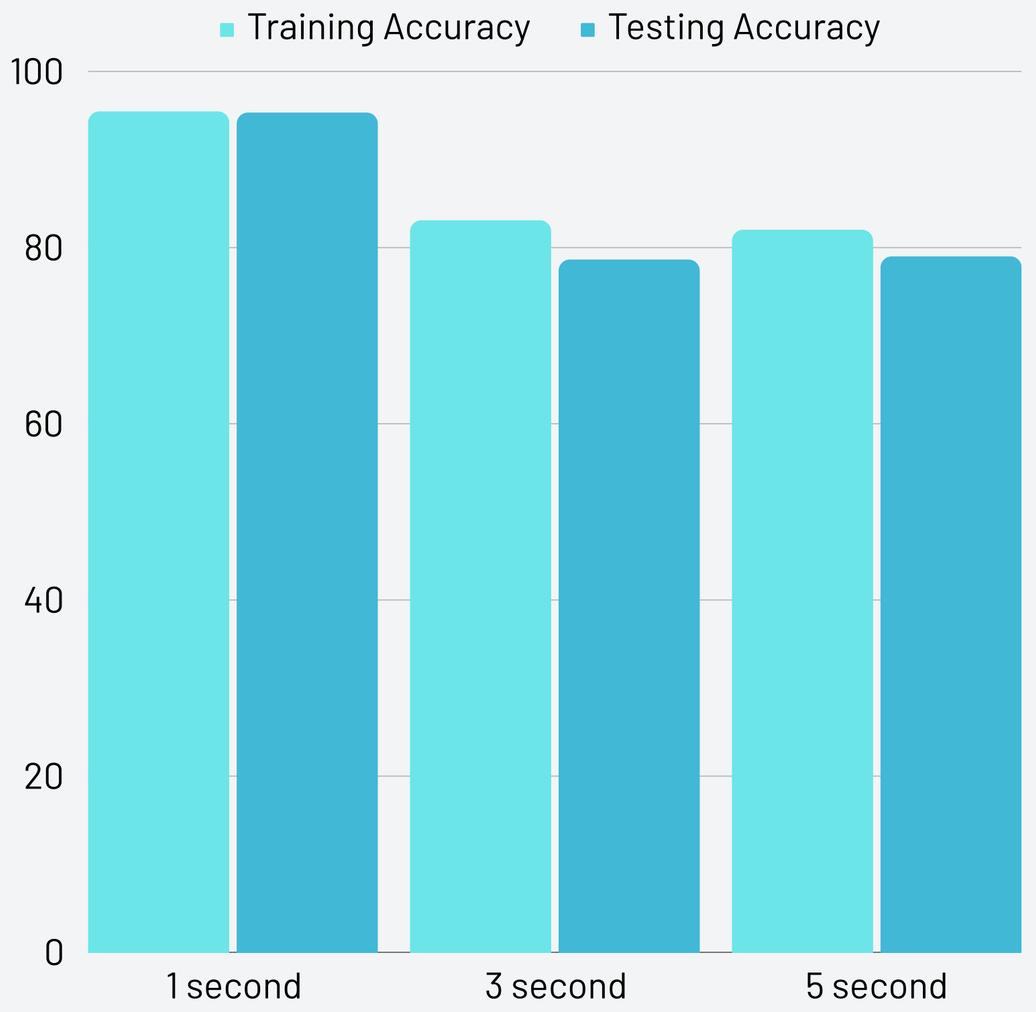
TRAINING & TESTING DIFFERENT HYPER PARAMETERS

We will be changing different hyper-parameters, namely Sampling Rate, input buffer size, input window size and have recorded the respective training and testing accuracies.

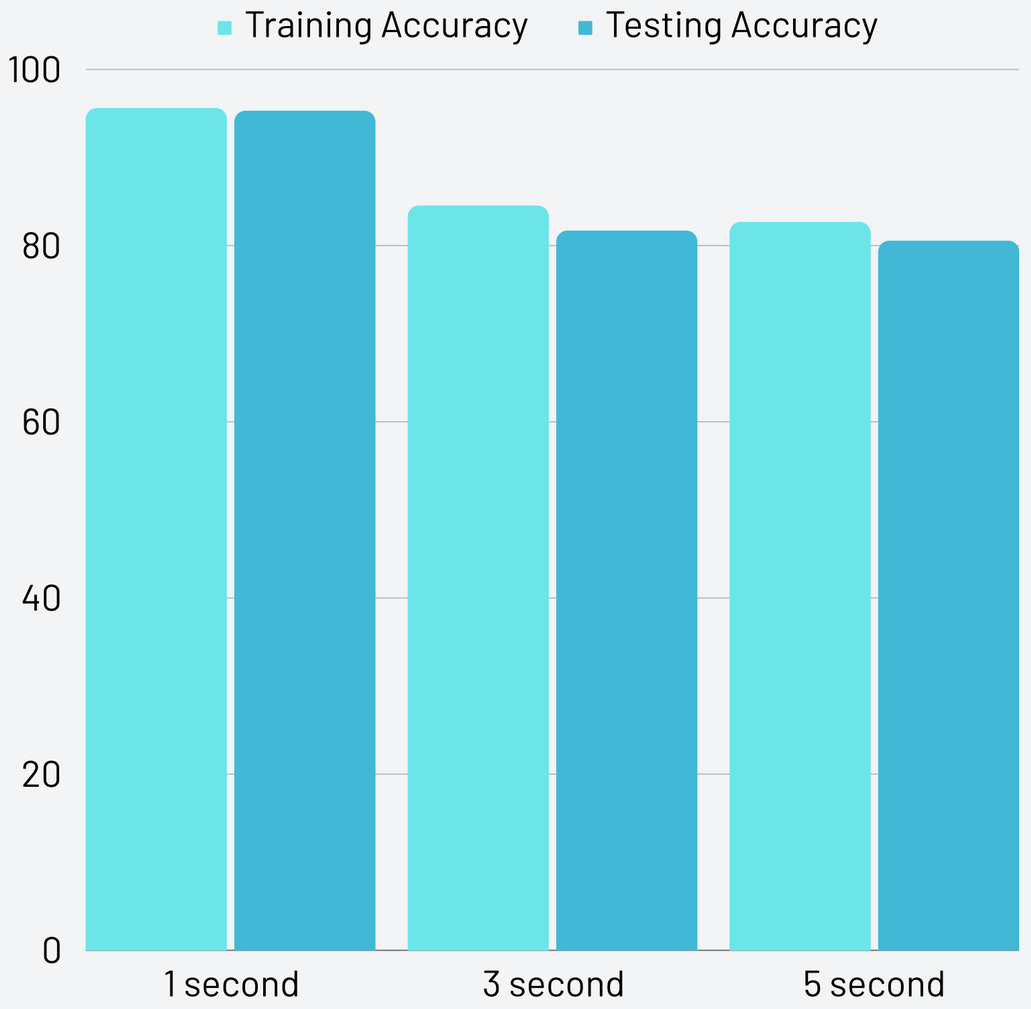
SAMPLING RATE 8000



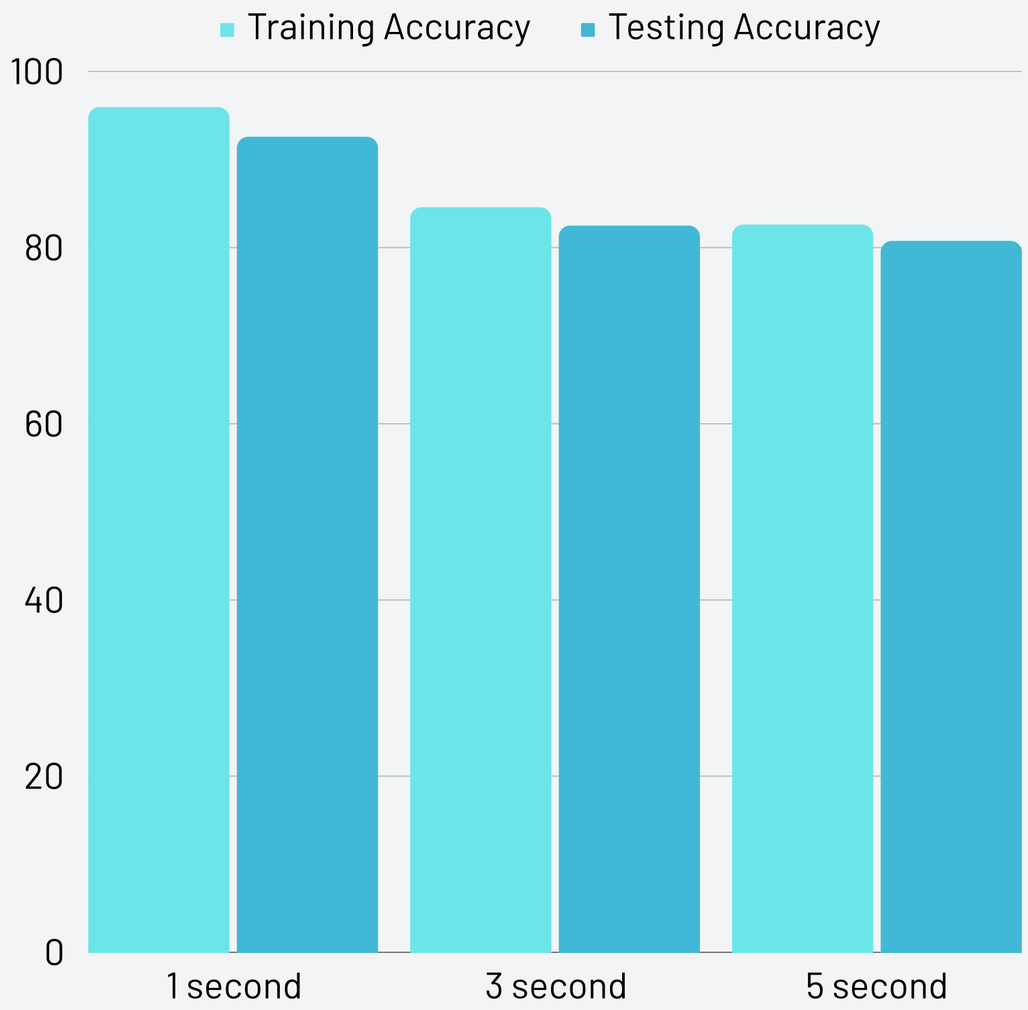
SAMPLING RATE 16000



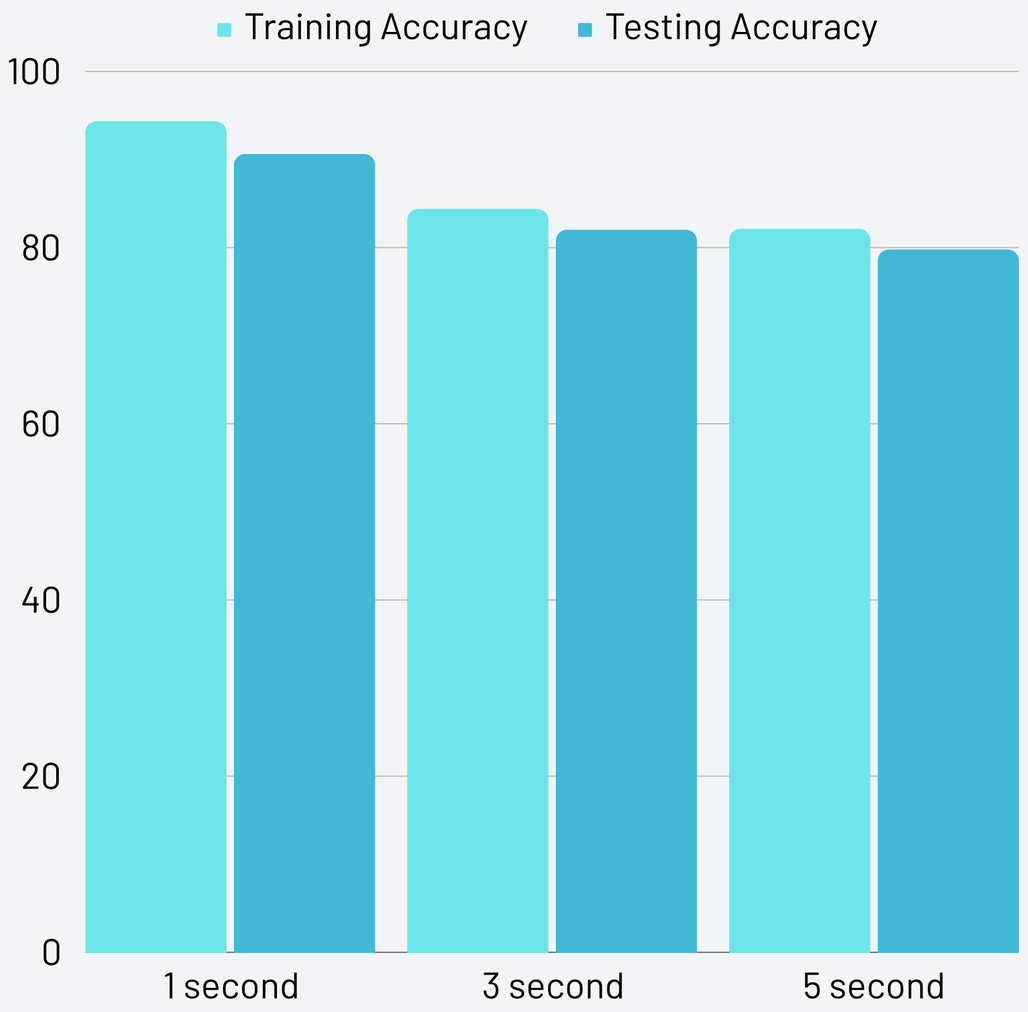
SAMPLING RATE 24000



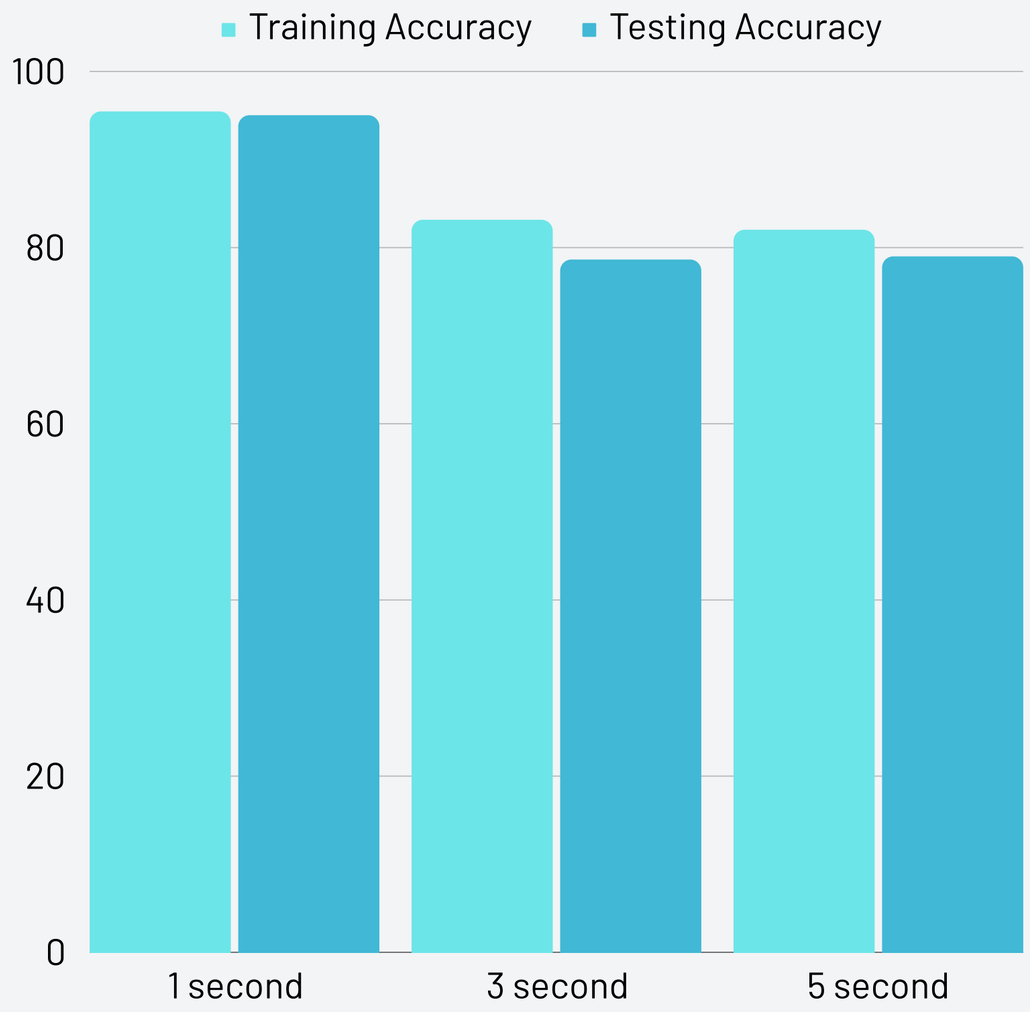
INPUT BUFFER SIZE
128



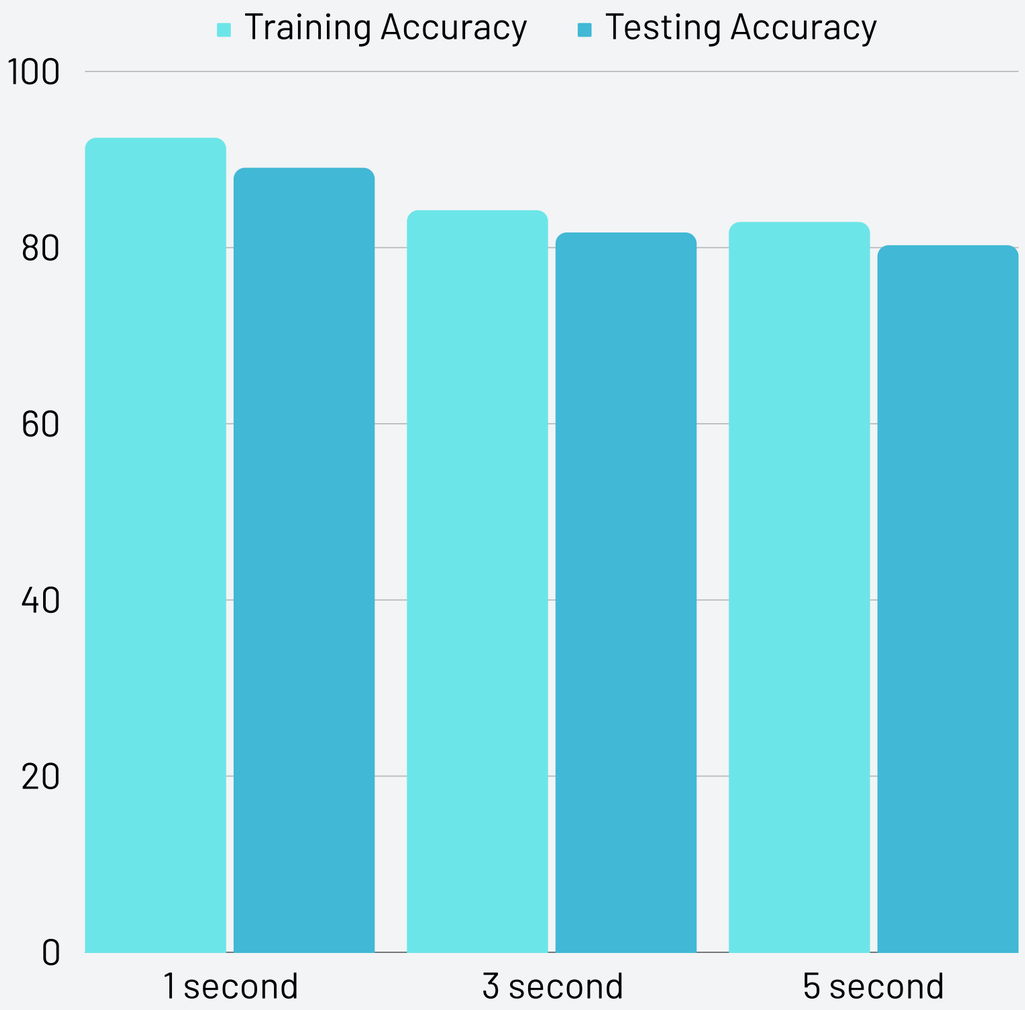
INPUT BUFFER SIZE 256



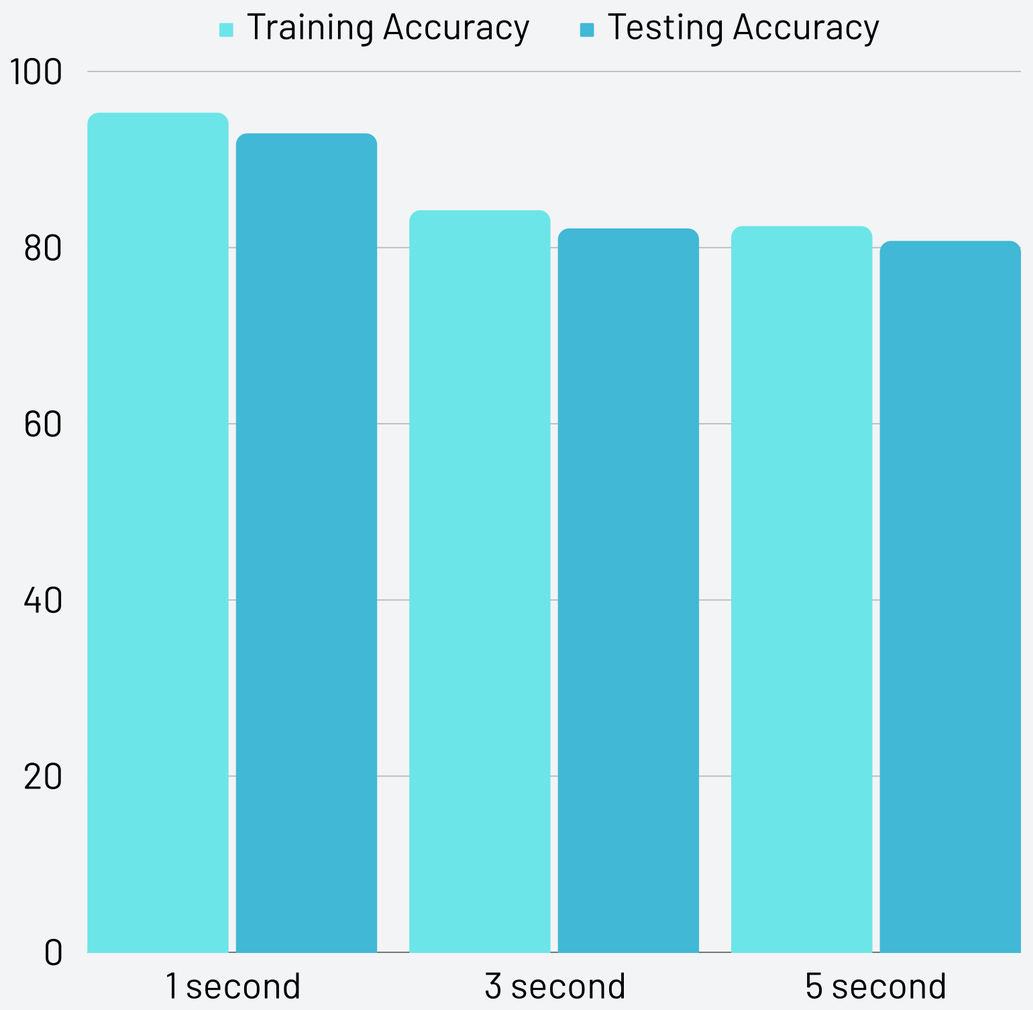
INPUT BUFFER SIZE 512



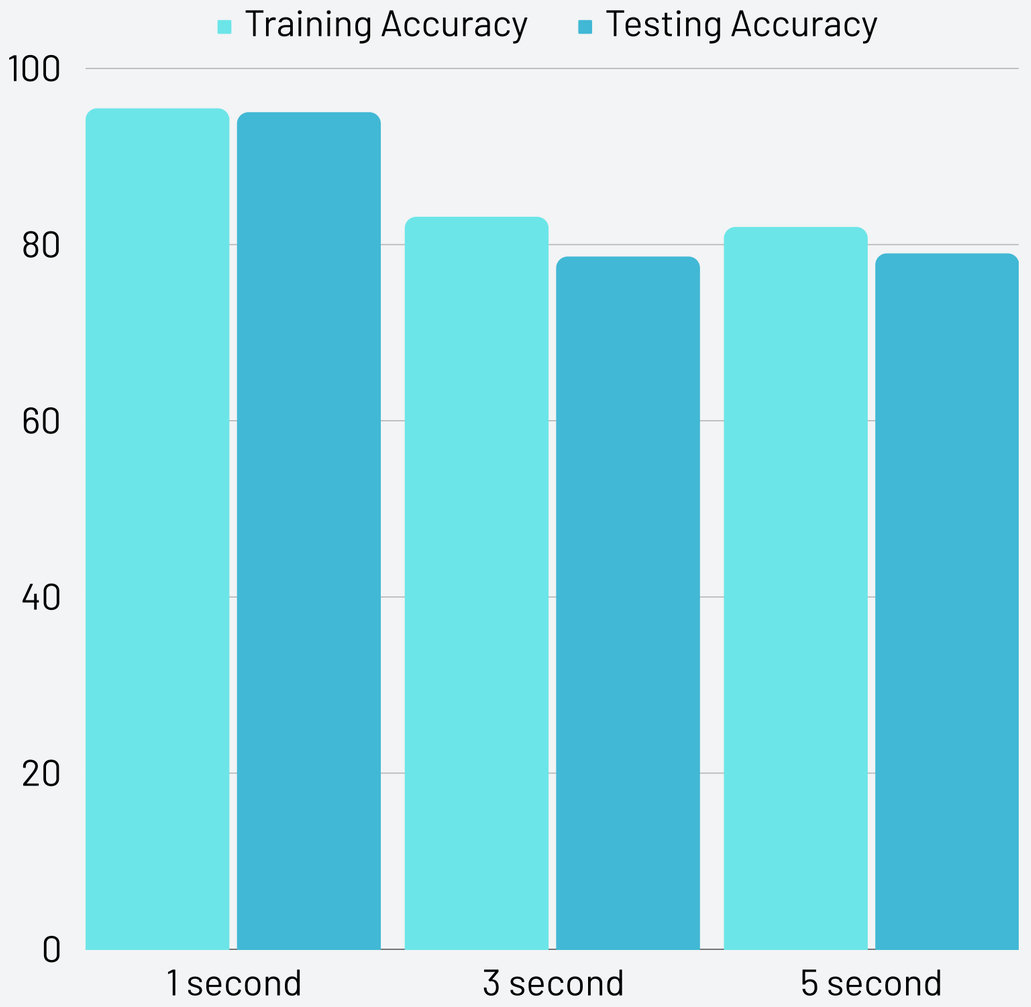
INPUT WINDOW SIZE 128



INPUT WINDOW SIZE 256



INPUT WINDOW SIZE 512



Observations And Results

Effects Of Sampling Rate:

- We have varied sampling rate from 8000 to 24000.
- We observed that the sampling rate of 8000 is better than others, because of reduced dimensionality and higher computational efficiency.

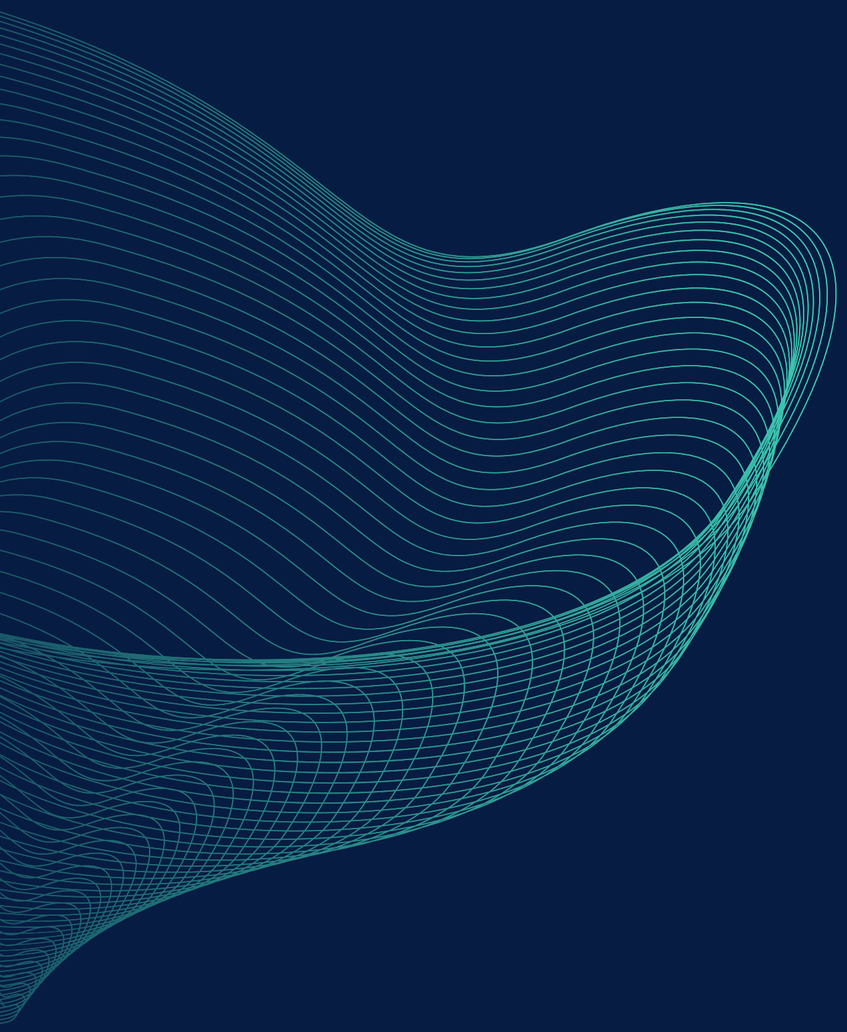
Effects Of Input Buffer Size

- We have varied input buffer size from 128 to 512.
- We observed smaller buffer size like 128 gave better results than higher rates and this could be because of lower memory consumption and higher temporal resolution.

Effects Of Input Window Size:

- We have varied input window size from 128 to 512.
- We observed that window size of 256 is giving better results and this could be attributed to model’s complexity and computational efficiency.

Milestones Achieved



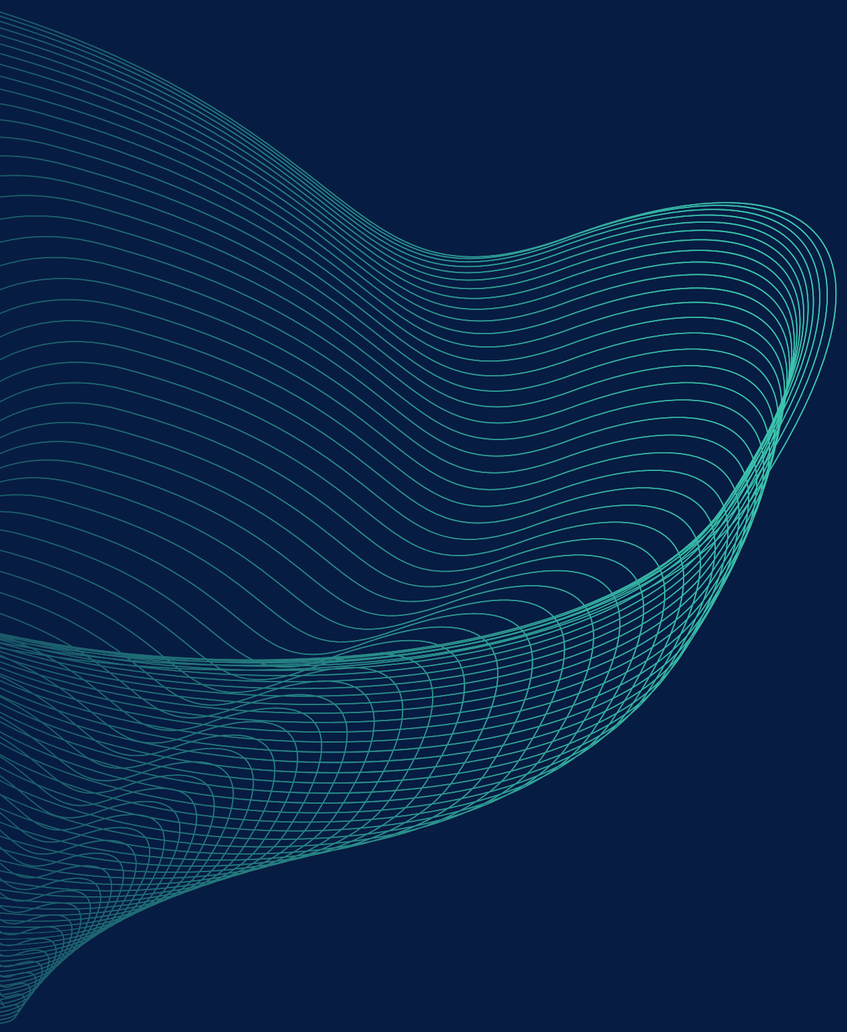
- Data Collection: We collected diverse audio recordings, including honk sounds and environmental noises, for a realistic and comprehensive dataset.

- Data Pre-processing: Data was preprocessed by cleaning, trimming, and segmenting audio recordings using Audacity, improving the model's focus on the target sound.

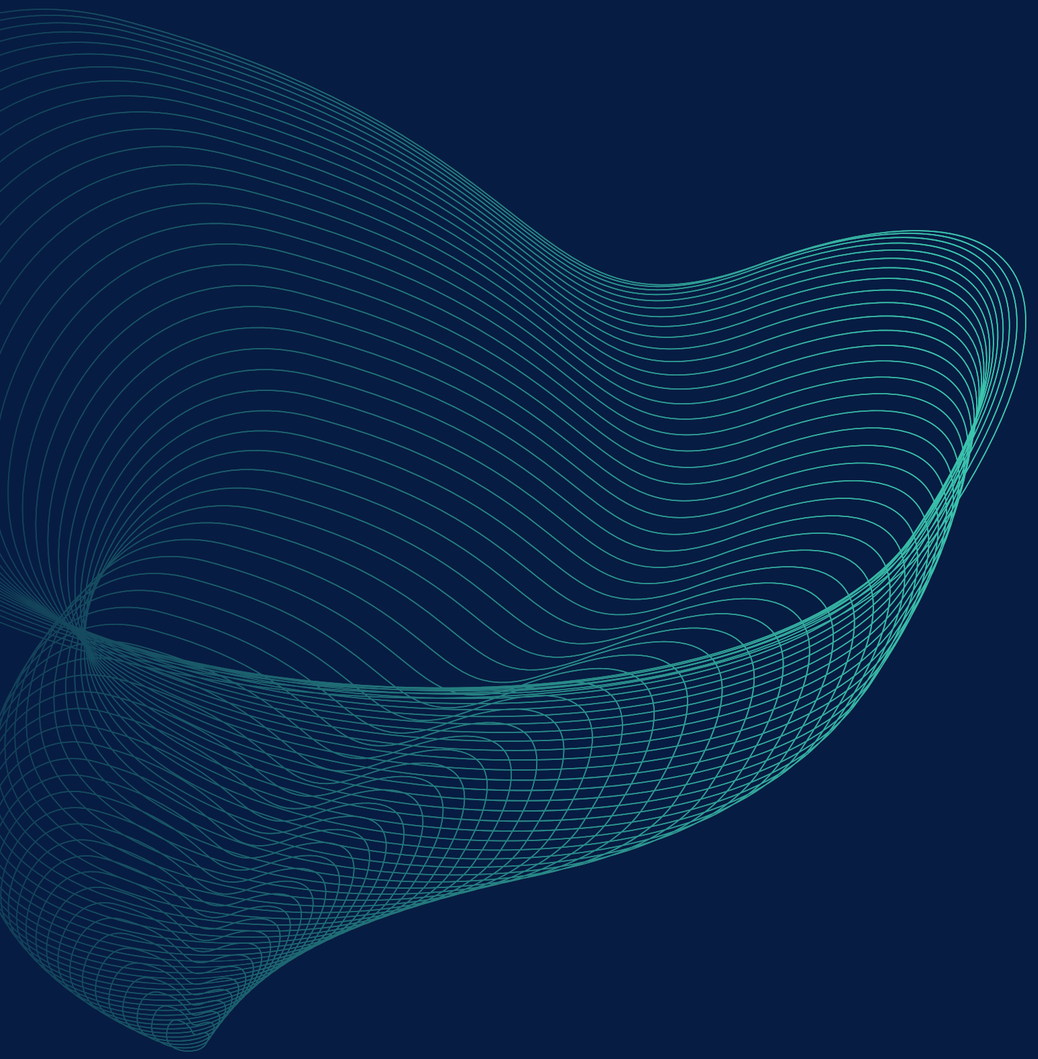
- Environment Setup: We set up ELL, along with configuring PyTorch for deep learning and PyAudio for audio-related tasks.

- Training And Testing the Model: Trained and tested a PyTorch-based GRU model on parameters like sampling rate and input buffer size that can be exported it as KeywordSpotter.onnx.

Future Proposals



- Presently we can create a larger dataset for horn sounds (+ve dataset). We wish to expand this dataset for better fine-tuning our model and better predicting the vehicle-honking.
- This Data can further help us understand more about the overall consequences faced by nature due to the vehicle-pollution.
- With the data on Noise Pollution Levels and air pollution levels of an area, we might be able to correlate and establish a relation between noise pollution and air pollution.



Thank You
Prof. Rijurekha Sen
Sir Sachin Kumar Chauhan

Contributors

- **Ananya Aakriti**
- **Mehul Kamboj**
- **Yashwant**