COURSE DETECTION FOR NODE DEVICE

Project Echo

Abstract

Using what I have previously found in this trimester, I will develop a lightweight but effective algorithm that can filter audio. Then, I will evaluate the current performance by using time measurement and accuracy.

Nhat Minh Dang

222172836

Contents

A) Introduction:

Project Echo was founded in Trimester 3 of 2022 by Stephan Kokkas, Andrew Kudilczak and Daniel Gladman. The project aims to support global conservationists by developing advanced audio classification systems that can facilitate the non-intrusive monitoring, discovery and tracking endangered species and their predators within their natural habitats. The system of IoT will collect data and transmit to a center server via an API, where an AI-driven model classifies species and record vital data. Then, the Human Machine Interface (provides real time visualization of simulated animals and vocalization events on an interactive map.

As the project develops, more and more audio data are collected, increasing the current capacity. These audio files are varied in frequency, quality, duration, etc. In this task, I will digest deeper into optimizing the frequency and save some space for new audio datasets. I will create a simple set of algorithms that is lightweight enough to put into the node device with optimized frequency and most effective quality lower method found in previous researches.

B) Definitions and Concepts:

1) Frequency in Audio:

Frequency refers to the number of sound wave cycles that occur per second, measured in Hertz (Hz). It originates from the physical vibrations of an object, such as vocal cords, instruments, or environmental sources, which cause air molecules to oscillate and produce sound waves. These vibrations vary in speed; slower vibrations result in lower frequencies (e.g., deep growls), while faster vibrations create higher frequencies (e.g., bird chirps).

Frequency is a critical feature in audio analysis because it captures the tonal and spectral characteristics unique to different sound sources, including animal vocalizations. In animal audio classification, frequency patterns play a vital role in distinguishing species or behaviours, as many animals produce sounds within specific frequency ranges that reflect their physical structure or communication needs. Therefore, optimizing classification models with specific frequency bands helps isolate meaningful signal components and reduces noise or irrelevant data. For instance, removing irrelevant high frequencies in low-pitched animal calls can improve model focus and accuracy, while retaining key bands ensures that the classifier captures the essential acoustic signatures unique to each animal. This optimization is essential in creating robust and efficient audio classification systems.

Below is the picture of Audio Spectrum (**Figure 1**), which is divided into specific ranges. Each field is designed with different purpose:

- Sub bass: Spanning from 20 Hz to 60 Hz. The sound is deep and resonant, which is often felt rather than heard. Kick drums and bass synthesizers are examples for this.
- Lower Midrange: Covering 250 Hz to 500 Hz, the bass segment provides warmth and substance to sound. This range is less perceptible to the human ear, requiring strong amplification for precise reproduction. This includes instruments like bass guitars and lower piano registers, which anchor the overall sound structure.
- Midrange: From 500 Hz to 2 kHz, the midrange is popular for vocal articulation and the distinct separation of instruments within a max.

- Upper midrange: Spanning from 2 kHz to 4 kHz, the upper midrange contains the harmonics of lower pitched instruments, significantly affecting clarity and timbre and requires heightened sensitivity in human hearing.
- Presence: Ranging between 4 to 6 kHz, it is a demonstration of high pitch sounds from instruments like violins and guitars.
- Brilliance: From 6 kHz to 20 kHz, the brilliance range captures high frequency details that bring sparkle and life to audio, followed by adding ethereal quality to music.

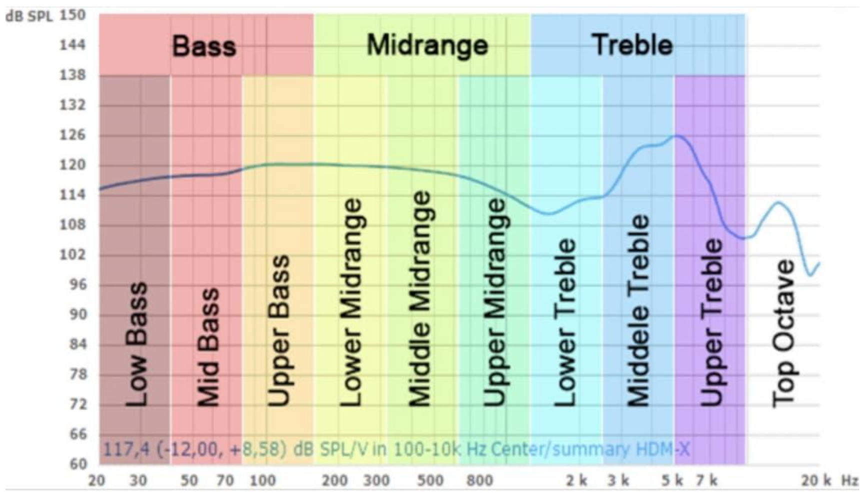


Figure 1: Audio Spectrum.

2) Fast Fourier Transform (FFT):

The Fast Fourier Transform (FFT) is a mathematical algorithm used to transform a signal from the time domain into the frequency domain. In the context of audio processing, this transformation is essential for analysing the frequency components of an audio signal. The audio signal, represented as a time-series of amplitude values (y), contains raw sound wave data that is difficult to interpret directly when trying to understand its frequency content. By applying FFT, the signal is decomposed into its constituent sinusoidal components, revealing the different frequencies and their respective amplitudes (or energies).

FFT calculates the Discrete Fourier Transform (DFT) of a signal, which is computationally expensive using direct computation. The FFT optimizes this process, reducing the complexity from $O(n^2)$ to O(n log n) where n is the number of samples in the signal. The result of the FFT is an array of complex numbers, where each element corresponds to a specific frequency and its associated amplitude. These amplitudes indicate the strength (energy) of the signal at that frequency.

3) Trimming Silence:

The **Trimming Silence method** implemented in this code utilizes the librosa.effects.trim function to preprocess audio signals by removing silent sections from the beginning and end of each recording. Silence is identified based on an amplitude threshold, which is set to a maximum decibel level of 20 dB relative to the signal's peak. This ensures that any low-amplitude noise or pauses, often unrelated to the primary sound content, are excluded from further processing. By trimming silence, the method focuses on the meaningful parts of the audio, reducing irrelevant data and improving the efficiency and accuracy of subsequent steps like feature extraction and classification. The trimmed audio is then passed through the Fast Fourier Transform (FFT) for energy analysis, enabling a cleaner and more precise representation of sound features within the relevant frequency range. This approach is particularly beneficial for lightweight edge-device applications, where computational efficiency and effective filtering of irrelevant data are critical.

4) Libraries:

- Os: Used to interact with the file system. In my code, the os module lists .wav files in the specified directory, enabling batch processing of audio files.
- Librosa: A Python library for audio analysis. It is used to load audio files, extract their duration, and preprocess them for further analysis.
- Numpy: A library for numerical computation. Here, it is used for operations like computing the Fast Fourier Transform (FFT), filtering frequency ranges, and calculating energy proportions.
- Time: Used to measure execution time. The time module calculates the total time taken for the program to process all audio files.

C) Implementation:

1) Global configuration:

- AUDIO_PATH: Specifies the directory path where .wav files are stored for processing.
- MIN_FREQ and MAX_FREQ: Define the frequency range (in Hz) of sounds typically produced by animals.
- MIN_DURATION: Sets the minimum length (in seconds) for an audio file to be considered valid for analysis.
- ANIMAL_THRESHOLD: A threshold representing the minimum proportion of energy in the target frequency range required to classify an audio file as containing animal sounds.

2) How does the code work:

The code processes a directory of .wav audio files to detect which ones contain animal sounds. It is lightweight, relying on spectral analysis and heuristic thresholds rather than machine learning models, making it suitable for edge devices. The process is divided into each steps:

- File Listings: scan the directory specified in AUDIO_PATH and identifies all .wav files.
- Audio Preprocessing: Each audio file is loaded using librosa. Files shorter than MIN_DURATION are discarded to avoid processing incomplete or noisy data.
- Feature Extraction: FFT is applied to transform the audio signal into its frequency domain. The resulting frequency bins and their corresponding amplitudes are used for further analysis.

- Classification: The proportion of energy within the specified frequency range (MIN_FREQ to MAX_FREQ) is calculated. If this proportion exceeds the ANIMAL_THRESHOLD, the file is classified as containing animal sounds.
  Note: Using the result from the task Effect of Frequency Band Removal, I will test in the optimised range 300 – 2000 Hz. Then, in the preprocessing steps, I use trimming silence at optimal db = 20 to reduce the file size as a result of the task Impact of Quality Audio.
- Output and Evaluation: The program outputs the list of audio files classified as containing animal sounds and calculates the accuracy of the classification as a proportion of the processed files.
- Execution Time measurement: The total time taken to process all files is calculated and displayed.
- The code will be executed multiple time for average time calculation.

3) Evaluation Method:

a) Detection Accuracy:

Accuracy is calculated as the ratio of files classified as containing animal sounds to the total number of processed files. This provides a quantitative measure of the algorithm's performance.

b) Execution Time:

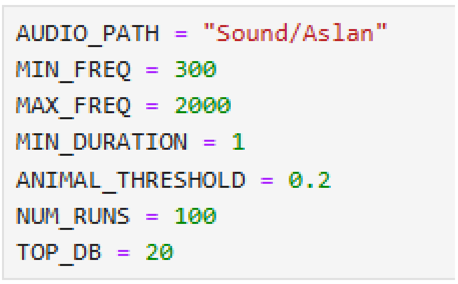
The time taken for the entire process is measured, giving an insight into the efficiency of the algorithm. This is particularly important for deployment on edge devices.

c) Output verification:

The list of files classified as containing animal sounds can be manually verified to assess the heuristic rules' effectiveness.

4) Result Example:

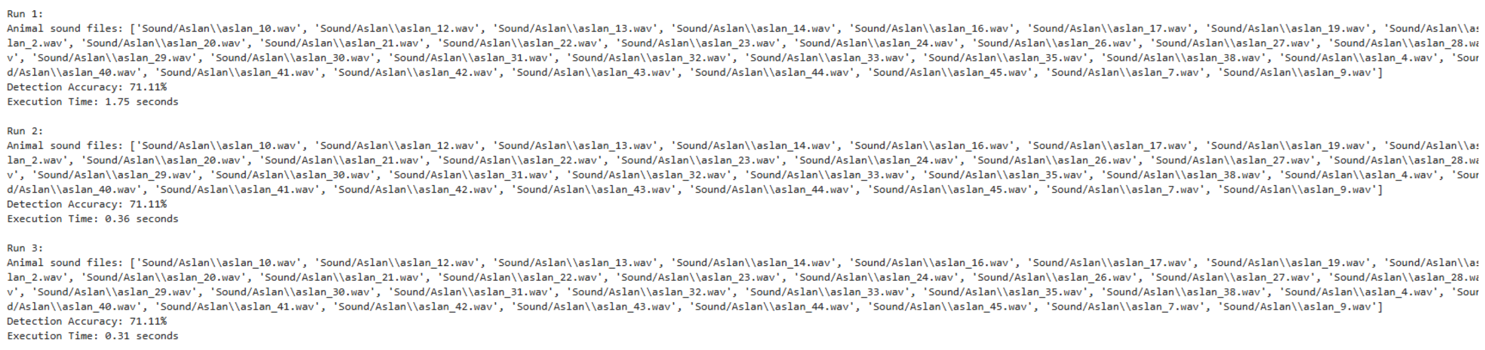
For data testing, I will focus mainly on the previous datasets that I used for the previous task, including lion, cat, dog, sheep and frog. In this report, I will include my example of configuration and result processing Aslan. Below is my configuration:



```
AUDIO_PATH = "Sound/Aslan"
MIN_FREQ = 300
MAX_FREQ = 2000
MIN_DURATION = 1
ANIMAL_THRESHOLD = 0.2
NUM_RUNS = 100
TOP_DB = 20
```

Figure 2: My configuration.

After executing the code, this is my example result:



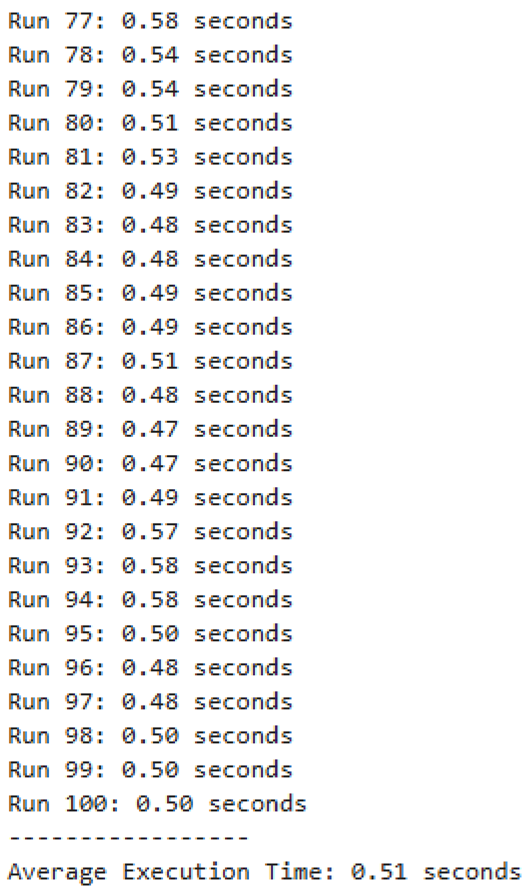
```
Run 1:
Animal sound files: ['Sound/Aslan\\aslan_10.wav', 'Sound/Aslan\\aslan_12.wav', 'Sound/Aslan\\aslan_13.wav', 'Sound/Aslan\\aslan_14.wav', 'Sound/Aslan\\aslan_16.wav', 'Sound/Aslan\\aslan_17.wav', 'Sound/Aslan\\aslan_19.wav', 'Sound/Aslan\\as
lan_2.wav', 'Sound/Aslan\\aslan_20.wav', 'Sound/Aslan\\aslan_21.wav', 'Sound/Aslan\\aslan_22.wav', 'Sound/Aslan\\aslan_23.wav', 'Sound/Aslan\\aslan_24.wav', 'Sound/Aslan\\aslan_26.wav', 'Sound/Aslan\\aslan_27.wav', 'Sound/Aslan\\aslan_28.wa
v', 'Sound/Aslan\\aslan_29.wav', 'Sound/Aslan\\aslan_30.wav', 'Sound/Aslan\\aslan_31.wav', 'Sound/Aslan\\aslan_32.wav', 'Sound/Aslan\\aslan_33.wav', 'Sound/Aslan\\aslan_35.wav', 'Sound/Aslan\\aslan_38.wav', 'Sound/Aslan\\aslan_4.wav', 'Sour
d/Aslan\\aslan_40.wav', 'Sound/Aslan\\aslan_41.wav', 'Sound/Aslan\\aslan_42.wav', 'Sound/Aslan\\aslan_43.wav', 'Sound/Aslan\\aslan_44.wav', 'Sound/Aslan\\aslan_45.wav', 'Sound/Aslan\\aslan_7.wav', 'Sound/Aslan\\aslan_9.wav']
Detection Accuracy: 71.11%
Execution Time: 1.75 seconds

Run 2:
Animal sound files: ['Sound/Aslan\\aslan_10.wav', 'Sound/Aslan\\aslan_12.wav', 'Sound/Aslan\\aslan_13.wav', 'Sound/Aslan\\aslan_14.wav', 'Sound/Aslan\\aslan_16.wav', 'Sound/Aslan\\aslan_17.wav', 'Sound/Aslan\\aslan_19.wav', 'Sound/Aslan\\as
lan_2.wav', 'Sound/Aslan\\aslan_20.wav', 'Sound/Aslan\\aslan_21.wav', 'Sound/Aslan\\aslan_22.wav', 'Sound/Aslan\\aslan_23.wav', 'Sound/Aslan\\aslan_24.wav', 'Sound/Aslan\\aslan_26.wav', 'Sound/Aslan\\aslan_27.wav', 'Sound/Aslan\\aslan_28.wa
v', 'Sound/Aslan\\aslan_29.wav', 'Sound/Aslan\\aslan_30.wav', 'Sound/Aslan\\aslan_31.wav', 'Sound/Aslan\\aslan_32.wav', 'Sound/Aslan\\aslan_33.wav', 'Sound/Aslan\\aslan_35.wav', 'Sound/Aslan\\aslan_38.wav', 'Sound/Aslan\\aslan_4.wav', 'Sour
d/Aslan\\aslan_40.wav', 'Sound/Aslan\\aslan_41.wav', 'Sound/Aslan\\aslan_42.wav', 'Sound/Aslan\\aslan_43.wav', 'Sound/Aslan\\aslan_44.wav', 'Sound/Aslan\\aslan_45.wav', 'Sound/Aslan\\aslan_7.wav', 'Sound/Aslan\\aslan_9.wav']
Detection Accuracy: 71.11%
Execution Time: 0.36 seconds

Run 3:
Animal sound files: ['Sound/Aslan\\aslan_10.wav', 'Sound/Aslan\\aslan_12.wav', 'Sound/Aslan\\aslan_13.wav', 'Sound/Aslan\\aslan_14.wav', 'Sound/Aslan\\aslan_16.wav', 'Sound/Aslan\\aslan_17.wav', 'Sound/Aslan\\aslan_19.wav', 'Sound/Aslan\\as
lan_2.wav', 'Sound/Aslan\\aslan_20.wav', 'Sound/Aslan\\aslan_21.wav', 'Sound/Aslan\\aslan_22.wav', 'Sound/Aslan\\aslan_23.wav', 'Sound/Aslan\\aslan_24.wav', 'Sound/Aslan\\aslan_26.wav', 'Sound/Aslan\\aslan_27.wav', 'Sound/Aslan\\aslan_28.wa
v', 'Sound/Aslan\\aslan_29.wav', 'Sound/Aslan\\aslan_30.wav', 'Sound/Aslan\\aslan_31.wav', 'Sound/Aslan\\aslan_32.wav', 'Sound/Aslan\\aslan_33.wav', 'Sound/Aslan\\aslan_35.wav', 'Sound/Aslan\\aslan_38.wav', 'Sound/Aslan\\aslan_4.wav', 'Sour
d/Aslan\\aslan_40.wav', 'Sound/Aslan\\aslan_41.wav', 'Sound/Aslan\\aslan_42.wav', 'Sound/Aslan\\aslan_43.wav', 'Sound/Aslan\\aslan_44.wav', 'Sound/Aslan\\aslan_45.wav', 'Sound/Aslan\\aslan_7.wav', 'Sound/Aslan\\aslan_9.wav']
Detection Accuracy: 71.11%
Execution Time: 0.31 seconds
```

Figure 3: Example outputs.

As can be seen from the result, the program returns the list of qualified audio files, followed by calculating accuracy and returns the execution time. Furthermore, only the first execution took the largest time consumption. This is because the system has already learned the feature in the first iteration.



```
Run 77: 0.58 seconds
Run 78: 0.54 seconds
Run 79: 0.54 seconds
Run 80: 0.51 seconds
Run 81: 0.53 seconds
Run 82: 0.49 seconds
Run 83: 0.48 seconds
Run 84: 0.48 seconds
Run 85: 0.49 seconds
Run 86: 0.49 seconds
Run 87: 0.51 seconds
Run 88: 0.48 seconds
Run 89: 0.47 seconds
Run 90: 0.47 seconds
Run 91: 0.49 seconds
Run 92: 0.57 seconds
Run 93: 0.58 seconds
Run 94: 0.58 seconds
Run 95: 0.50 seconds
Run 96: 0.48 seconds
Run 97: 0.48 seconds
Run 98: 0.50 seconds
Run 99: 0.50 seconds
Run 100: 0.50 seconds
-----------------
Average Execution Time: 0.51 seconds
```

Figure 4: Average Time

With the output of time execution, the longer the number of iterations, the shorter the execution time. In this animal, the time execution remains stable at 0.5 seconds. Although the program does not use any specific learning model, the features are extracted and can be used for later preference.

5) Result Summary:

This is my result after testing all 5 species of animals (without using Trimming Silence):

Animal	Accuracy	Time (Average – Max)
Lion	71.11%	0.5 – 1.75 sec
Cat	58.79%	1.48 - 3.06 sec
Dog	83.77%	1.82 - 3.55 sec
Sheep	65%	0.19 – 1.95 sec
Frog	25.71%	0.18 - 1.81 sec

This is my result after testing all 5 species of animals (using Trimming Silence):

Animal	Accuracy	Time (Average – Max)
Lion	68.69%	0.33 – 2.33 sec
Cat	54.77%	2.42 – 4.1 sec
Dog	79.06%	1.34 – 2.94 sec
Sheep	57.5%	0.18 – 1.8 sec
Frog	25.71%	0.15 - 1.8 sec

As can be seen in the result, most of the animals have high accuracy and the execution time is reduced by half after the first execution. However, with distinction Frequency Range, Frog might require further configurations to investigate.

When comparing with the result using trimming silence, although the accuracy is reduced slightly, the execution time is reduced significantly. For the frog, the accuracy and time execution remain unchanged. This indicates that the Trimming silence also have an effective approach for course detection.

D) Conclusions:

With this algorithm, I have reduced the complexity the algorithm but maintain reasonable accuracy, followed by execution time. However, due to the limited in testing time, there are more testing cases required to develop the functions. For example, because these data are specific animal species, the data can be mixed to make the system harder to detect. Moreover, a variety of Quality Lowering methods can be applied.

Finally, the comparison between the performance for Trimming Silence provides a significant insight. While the method increases the execution time of the first iteration, the later loops are optimised significantly.