

By

Abhishek Sebastian	20BEC1118
Harish D	20BEC1034
Gabriel K	20BEC1320

A project report submitted to

Dr. A Annis Fathima

School of Electronics Engineering

In partial fulfilment of the requirements for the course of

ECE- Digital Signal Processing

In

B.Tech. Electronics and communication engineering



VIT[®]

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Vellore Institute of Technology, Chennai

Vandalur-Kelambakkam road, Chennai – 600127

November 2022

Bonafide certificate

Certified that this project report entitled “Detecting Langur Alert Calls using MFE, Spectrogram and NN Classifier” is a bonafide work of Abhishek Sebastian (20BEC1118) , Harish D (20BEC1034) and Gabriel K (20BEC1320) carried out the project work under my supervision and guidance.

Dr. Annis Fathima A

Associate professor

School of electronics engineering (sense),

Vit university, Chennai

Chennai-600127.

Abstract

The International Union for Conservation of Nature maintains a list of threatened species called the "Red List" (IUCN). The Red List describes the severity and specific root causes of an endangered species' endangerment. The seven conservation levels included on the Red List are less of a concern, near threatened, vulnerable, endangered, critically endangered, extinct in the wild, and extinct. Each category represents a different degree of threat. Any ecosystem's long-term resilience and vitality greatly depend on species recovery. And other significant species that are essential to their ecosystems are making a comeback. Recent trends observed globally indicate that the number of endangered species making a comeback has increased. Langur Alert calls detecting is a frontier system proposed to detect animal species density in forests using their characteristics sounds. The system proposes the use of DSP algorithms like MFE, Spectrograms and NN. The trained model has acquired an accuracy of 97.10% in predicting langur alert calls.

Acknowledgement

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Annis Fathima**, professor, school of electronics engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Susan Elias**, dean of the school of electronics engineering, VIT Chennai, for extending the facilities of the school towards our project and for her unstinting support.

We express our thanks to our programme chair **Dr. Mohanaprasad K** for their support throughout the course of this project.

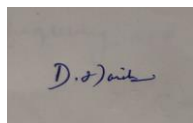
We also take this opportunity to thank all the faculty of the school for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

Team members



Abhishek Sebastian



Harish D



Gabriel K

Table of contents

Serial no.		Name	Page no.
		Abstract	3
		Acknowledgement	4
1		Introduction	6
	1.1	Objectives and goals	6
2		Theory	7
3		Result and analysis	13
4		Conclusion	14
5		References	14
6		Appendix- code	15

1. Introduction

1.1 Objectives and Goals

The main objective of the project is to develop a system , that is capable of detecting langur alert calls from the forest ambience. By doing so with enough number of replicas of the system in the forest ,we will be able to map species density among the forests. This information can be used to improve endangered species rehabilitation , tiger sightings during wildlife safari and lot more.

Goals:

- To detect langur alert calls from the forest ambience.
- To detect non-langur alert calls (birds chirping, cricket noises and other forest ambience)
- To provide model results in form of predictions and LED warning.

Recent Work Done:

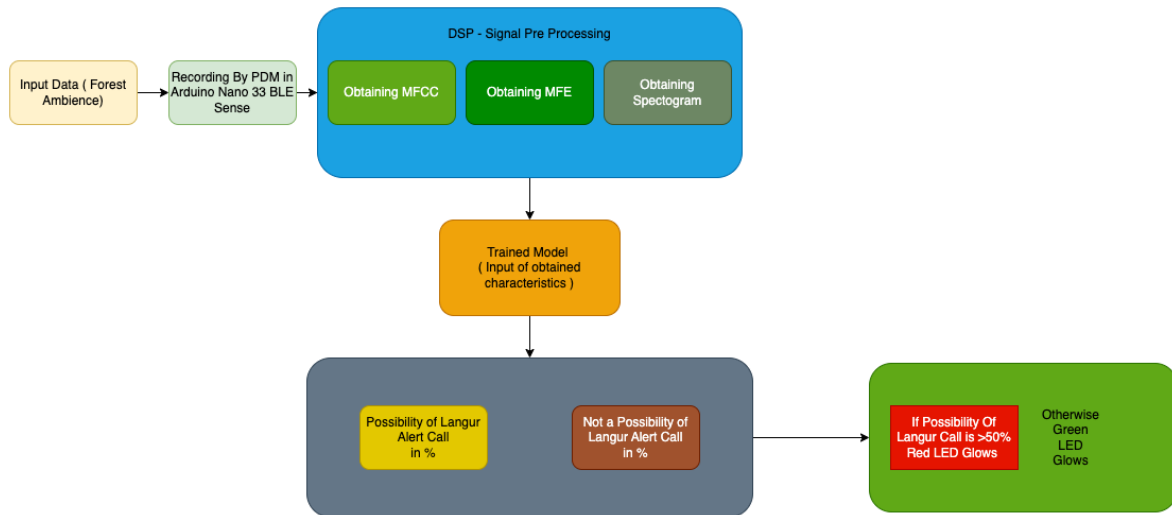
Researchers are working hard to integrate TinyML and low powered microcontrollers, this section discusses the important work that has been done in the area of TinyML and Tensor flow lite,

Using Arduino Nano 33 BLE Sense, Alati et al[1] proposed machine learning algorithms for greenhouse gas temperature Forecasting, The Paper focuses the importance of TinyML applications and Arduino Nano 33 BLE Sense's Low Power Capabilities. Waqar et al [2] proposed design of a speech anger recognition system using an Arduino Nano 33 BLE Sense's embedded microphone and a custom trained ML model using TinyML and SER Approach. Ali et al [3] prepared a research study on the Arduino Nano 33 BLE Sense and its machine learning and other processing capabilities, the following work also emphasises Arduino Nano 33 BLE Sense's application in wireless sensor networks. Kristian et al [4] in their paper, have discussed about Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework, The Former's Study Provides information on the computational capability of Arduino Nano 33 BLE Sense. Agus

Kurniawan in his Book [5] has explained in depth applications of Arduino Nano 33 BLE Sense through sample hands-on projects. Warden et al in their book [6] Has put forth valuable information on TinyML, Machine Learning with Tensor Flow Lite on Arduino compatible boards and other ultra-low power Microcontrollers. The Former's Book also discusses on Optimising energy usage, Optimising Model and porting model from tensor flow to tensor flow lite. Sharan et al [7] Prepared an overview of the current developments in sound recognition technology.

2. Theory

Block diagram



Algorithm

The hardware module consists of Arduino Nano BLE 33 Sense.

The embedded microphone , in the board records the ambient sound for 2 seconds, processes it and predicts if there is a a langur call heard or not.

Red light in the board → langur call is detected

Green light in the board → no langur calls detected

Alert calls of langurs have a very distinctive pattern from other alert call's that of deer's and other forest ambience noises. With a number of audio dataset of langur's alert calls and other forest ambience noises we can predict predator presences in

various parts of the forest. By doing so , the forest officials can dynamically change their safari routes to improvise the probability of predator sightings in the forests.

Hardware development:

Arduino nano 33 BLE sense was used for machine learning and PDM microphone integration. This is an original Arduino board that was released in 2019. The board is powered by an NRF52840 processor (cortex m4f), which can clock up to 64mhz and has a 1mb flash memory and 256kb SRAM. It also has a variety of interfaces such as i2c, SPI, USB, I2S, and UART[5].

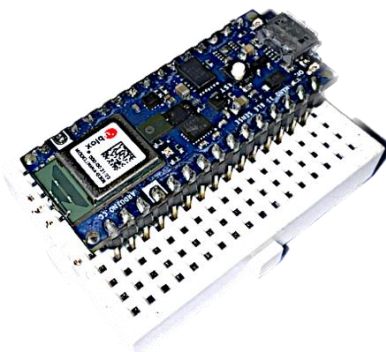


Figure2. Arduino Nano 33 BLE Sense

PDM-embedded microphone:

The MP34DT05-a is a digital mems microphone that is extremely small, low-power, omnidirectional, and designed with a capacitive sensing element and an IC interface. The sensing component, which can detect acoustic waves, is created utilizing a unique silicon micromachining procedure intended for the creation of audio sensors. Using a CMOS manufacturing method, the ic interface can be designed with a specialized circuit that can deliver a digital signal externally in PDM format.

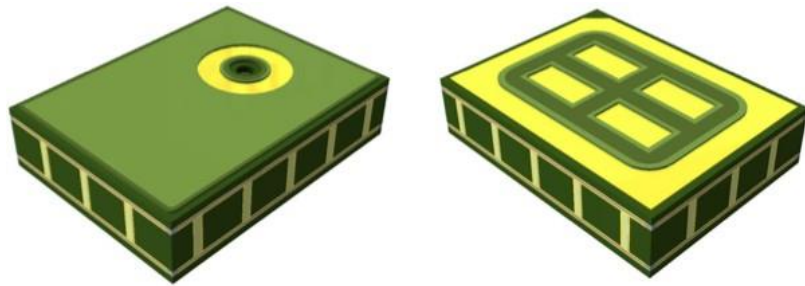


Figure2. PDM microphone - [mp34dt05](#)

Software development:

TensorFlow lite for microcontrollers (TFLM) was used in the earthquake probe. TensorFlow is a machine learning algorithm execution interface and implementation that was introduced in 2015.

TensorFlow lite for microcontrollers is an experimental adaptation of TensorFlow lite aimed at microcontrollers with only a few kb of memory. Existing tensor flow environments may be utilized for development, training, and testing, which is one of the key benefits of TFLM [9]. If a microcontroller is the ultimate deployment device, a model must be translated to TensorFlow lite and subsequently to TFLM data structures. The first step is to get some keras or equivalent framework training. When the findings are satisfactory, the model can be stored. A model must be converted to TF lite format after it has been saved.

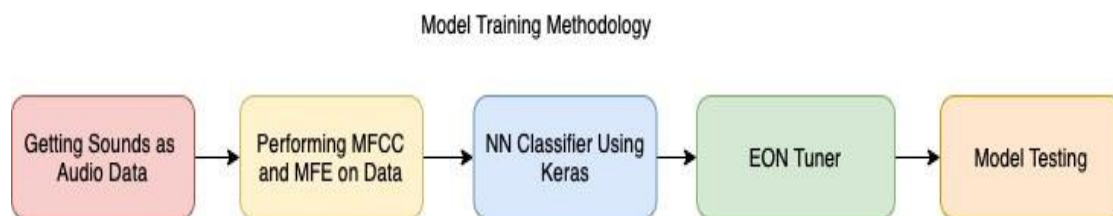


Figure2. Model training methodology

The above image explains the working flow of custom model training, to detect sounds of interest using edge impulse.

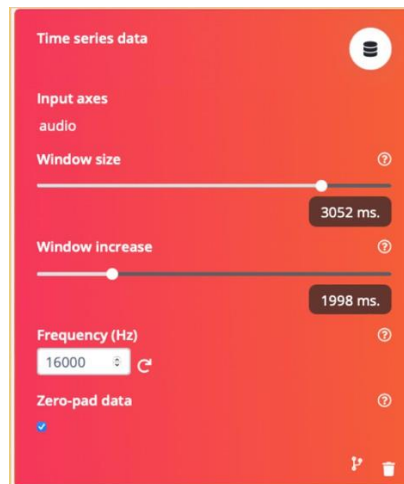
Training the model primarily consists of 5 major steps, starting with preparing the dataset for all possible scenarios ,performing suitable digital signal processing procedures like MFCC(Mel Filter Crystal Coefficients), MFE (MEL Filter bank Energy) and get features for the audio files. With processed features and raw data , a neural network audio classifier is trained based on TensorFlow and the same model

is tuned using an open- source tool for efficiency and higher computational ability. The tuned model is then tested to verify accuracy. The following section briefs the above steps in detail

Data set :

Langur alert calls' files : <https://www.kaggle.com/datasets/abhisheksebastian/langur-alert-calls>

forest ambience' files : <https://www.kaggle.com/datasets/kenjee/z-by-hp-unlocked-challenge-3-signal-processing>



The following data set is clipped to 3000ms and 16000 Hz for generalized training. The langur alert calls were obtained from Thailand forests by environmentalists. The latter was pre-processed using a professional audio editing service called “audacity”, additional processing like noise , frequency masking was performed for all of the data-set.

DSP Data pre-processing :

A signal's temporal and frequency properties are extracted using the audio MFE processing block. In the frequency domain, however, it employs a non-linear scale known as mel-scale. It works effectively with audio data, primarily for non-voice recognition applications where the sounds to be categorized are easily distinguishable by the human ear.

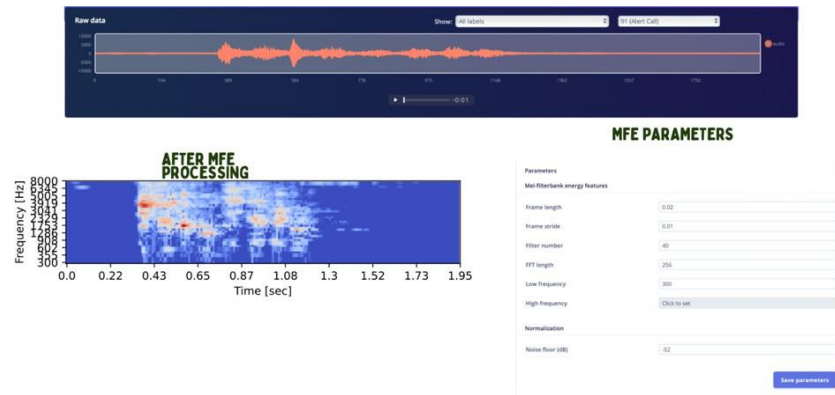


Figure. MFE Pre Processing



Figure. Spectrogram pre-processing

Machine learning :

Neural network classifier used here is **keras**, which is suitable for training audio signals, the ratio of training set and validation set is 0.8:0.2, with a learning rate of 0.0005 and 100 epochs.

The model trained is based on **sequential** from TensorFlow and **Adam** is used as the model optimizer.

```

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer, Dropout, Conv1D, Conv2D, Flatten, Reshape,
MaxPooling1D, MaxPooling2D, BatchNormalization, TimeDistributed, ReLU, Softmax
from tensorflow.keras.optimizers import Adam
EPOCHS = args.epochs or 500
LEARNING_RATE = args.learning_rate or 0.0001
# this controls the batch size, or you can manipulate the tf.data.Dataset objects yourself
BATCH_SIZE = 32
train_dataset = train_dataset.batch(BATCH_SIZE, drop_remainder=False)
validation_dataset = validation_dataset.batch(BATCH_SIZE, drop_remainder=False)

# model architecture
model = Sequential()
model.add(Dense(30, activation='relu',
                activity_regularizer=tf.keras.regularizers.l1(0.0001)))
model.add(Dense(15, activation='relu',
                activity_regularizer=tf.keras.regularizers.l1(0.0001)))
model.add(Dense(classes, name='y_pred', activation='softmax'))

# this controls the learning rate
opt = Adam(learning_rate=LEARNING_RATE, beta_1=0.9, beta_2=0.999)
callbacks.append(BatchLoggerCallback(BATCH_SIZE, train_sample_count, epochs=EPOCHS))

# train the neural network
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
model.fit(train_dataset, epochs=EPOCHS, validation_data=validation_dataset, verbose=2, callbacks=
callbacks, class_weight=ei_tensorflow.training.get_class_weights(Y_train))

# Use this flag to disable per-channel quantization for a model.
# This can reduce RAM usage for convolutional models, but may have
# an impact on accuracy.
disable_per_channel_quantization = False

```

Fig: ML Training Code Snippet



Fig: ML Training Results with respect to Validation set.

EON tuner :

The EON tuner is an open-source development tool from **Edge Impulse** that allows developers to choose the optimum trade-offs for their applications. This tunes the neural network to Arduino Nano BLE 33 Sense considering its restrictions and capabilities.

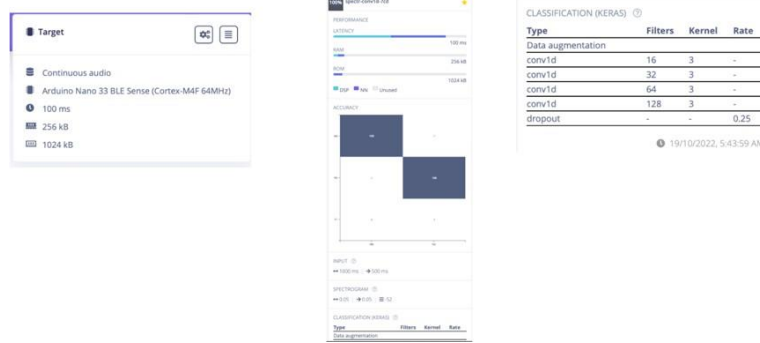


Fig. EON Tuner Excerpt

3. Result and analysis

Result:



Fig: Working of the system

Red light in the board → langur call is detected

Green light in the board → no langur calls detected

Model testing :

Model testing results

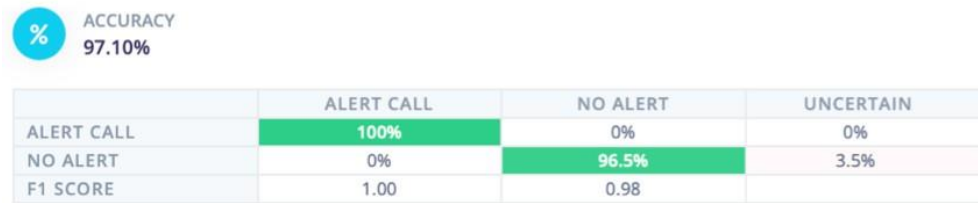


Fig. Model testing results with respect to training set.

It can be inferred from the above figure that , the model is able to predict alert calls with 100% accuracy and 96.5% for no alerts.

This discrepancies in no alerts is because of the varied data set, having different sounds like (bird chirping, cricket noises, insects calls etc.).This Uncertainty in predictions can be reduced by proper training.

4. Conclusion

System that is capable of detecting langur calls from forest ambience is designed successfully with an accuracy of 100% for alert calls and 96.5% for forest ambience.

Future scope:

The system can detect only langur calls for now, but in the future it can be done for many more species. The readings from the systems can be made more meaningful by combining with GPS to get accurate geolocation data and Wi-Fi connectivity for cloud data logging.

5. References

1. Alati MF, Fortino G, Morales J, Cecilia JM, Manzoni P. Time series analysis for temperature forecasting using TinyML. In 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC) 2022 Jan 8

- (pp. 691-694). IEEE.
2. Waqar DM, Gunawan TS, Morshidi MA, Kartiwi M. Design of a speech anger recognition system on Arduino nano 33 BLE sense. In 2021 IEEE 7th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA) 2021 Aug 23 (pp. 64-69). IEEE.
 3. Al-Mimi H, Al-Dahoud A, Fezari M, Daoud MS. A Study on New Arduino NANO Board for WSN and IoT Applications. International Journal of Advanced Science and Technology. 2020;29(4):10223-30.
 4. Dokic K, Martinovic M, Mandusic D. Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework. In 2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM) 2020 Sep 25 (pp. 1-6). IEEE.
 5. Kurniawan A. IoT Projects with Arduino Nano 33 BLE Sense. Berkeley: Apress. 2021;129
 6. Warden P, Situnayake D. Tinymml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers. O'Reilly Media; 2019 Dec 16.
 7. Sharan RV, Moir TJ. An overview of applications and advancements in automatic sound recognition. Neurocomputing. 2016 Aug 5;200:22-34.

Appendix

Code For Neural Network:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer, Dropout,
Conv1D, Conv2D, Flatten, Reshape, MaxPooling1D, MaxPooling2D,
BatchNormalization, TimeDistributed, ReLU, Softmax
from tensorflow.keras.optimizers import Adam
EPOCHS = args.epochs or 100
LEARNING_RATE = args.learning_rate or 0.005
# this controls the batch size, or you can manipulate the
tf.data.Dataset objects yourself
BATCH_SIZE = 32
train_dataset = train_dataset.batch(BATCH_SIZE,
drop_remainder=False)
validation_dataset = validation_dataset.batch(BATCH_SIZE,
drop_remainder=False)
```

```

# model architecture
model = Sequential()
model.add(Reshape((int(input_length / 32), 32),
input_shape=(input_length, )))
model.add(Conv1D(32, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Conv1D(64, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Conv1D(128, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(classes, name='y_pred', activation='softmax'))

# this controls the learning rate
opt = Adam(learning_rate=LEARNING_RATE, beta_1=0.9, beta_2=0.999)
callbacks.append(BatchLoggerCallback(BATCH_SIZE,
train_sample_count, epochs=EPOCHS))

# train the neural network
model.compile(loss='categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])
model.fit(train_dataset, epochs=EPOCHS,
validation_data=validation_dataset, verbose=2,
callbacks=callbacks)

# Use this flag to disable per-channel quantization for a model.
# This can reduce RAM usage for convolutional models, but may have
# an impact on accuracy.
disable_per_channel_quantization = FalseCode

```