CASA0018 Deep Learning for Sensor Networks

Coursework Report

Home Device Detector

Word count: 1780

GitHub URL: https://github.com/Youngwer/TML_HomeDeviceDetector

1.Introduction

1.1 Project Background and Motivation

The rise of smart homes, powered by Internet of Things (IoT) technologies, has dramatically transformed domestic environments. Especially, the integration of Tiny Machine Learning (TinyML) into home appliances has opened new avenues for enhancing automation and user interaction by enabling efficient, on-device intelligence for resource-constrained IoT devices, (Mishra, Malche and Hirawat, 2024). However, Deaf and hard of hearing (DHH) individuals face challenges in benefiting from these technologies, as they cannot hear critical auditory cues from household devices, such as microwave, or oven timers, which limits their ability to fully use the safety and convenience of smart home appliances. (Jain *et al.*, 2020). Therefore, there exists a need to develop inclusive smart home designs for the disabled to address these accessibility barriers.

1.2 Project Overview and Purpose

This project addresses these challenges by developing an embedded sound recognition system named *HomeDeviceDetector*, which leverages a TinyML model trained on Edge Impulse and deployed and on an Arduino Nano BLE Sense to detect and classify home sound events, such as smoke alarms and doorbells, delivering real-time visual alerts via LEDs (within 2 seconds) for DHH users.

1.3 Research Question

How can the device accurately detect and classifies critical household sound events on resource-constrained Arduino devices?

1.4 Prior Work and Technical Background

Research has demonstrated TinyML's potential in smart homes through applications like keyword spotting for voice-based automation, achieving high accuracy in resource-constrained environments(Mishra, Malche and Hirawat, 2024). Additionally, studies have explored sound awareness systems called HomeSound for DHH users, using visual and haptic feedback to convey information of household sounds(Jain *et al.*, 2020). Furthermore, TinyML has been applied in wearable IoT devices for sign language recognition, translating gestures into verbal language for hearing-impaired users(Sharma, Gupta and Kumar, 2024).
Furthermore, Edge Impulse enables for embedded machine learning (ML) development, offering an open-source machine learning and signal processing platform in a low-power embedded environment with excellent data security and privacy protection by processing voice locally(Koo and Seo, 2023).

2. Project Configuration and Methodology

2.1 Hardware Configuration

Hardware Components

Arduino Nano 33 BLE Sense:
As the core processing unit, which is an ideal for running TinyML models while maintaining low power consumption.
Visualisation part:
it consists of **three LEDs:**

- **Green** for **normal** environment;
- **Red** for **smoke alarm** detection;
- **Yellow** for **doorbell** alerts.

The circuit diagram is shown in Figure 1 below.

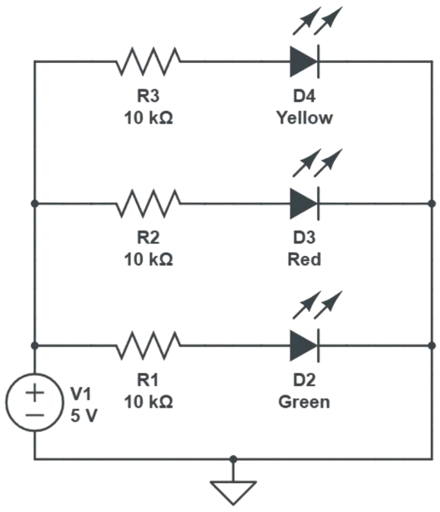


Figure 1. Circuit Diagram

Enclosure

The enclosure was designed to be compact using Fusion 360 for 3D modeling and PrusaSlicer for preparing the print, which also includes precise openings for the USB connection and optimal positioning for the onboard microphone to capture audio effectively. Below shows the simple enclosure in Figure 2.

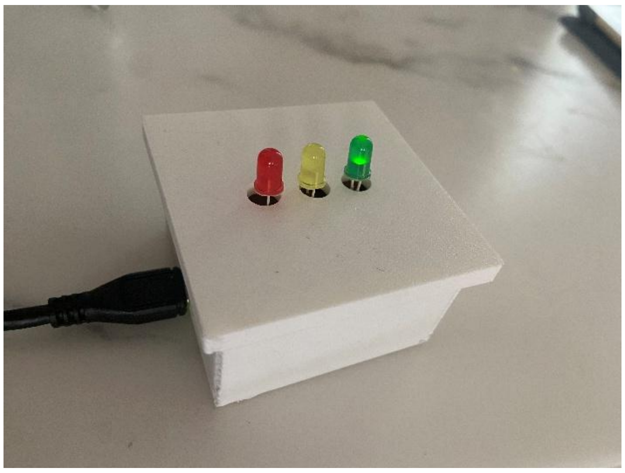


Figure 2. Simple Enclosure

2.2 Data Collection and Processing

Data Collection Strategy

Due to the impracticality of collecting real-time smoke alarm sounds (which would be potentially dangerous in a home environment), audio samples for both doorbell and smoke alarm categories were sourced from online repositories(Pixabay; Uppbeat) in WAV format. To better simulate real-world conditions with environmental noise, some doorbell samples were recorded using my mobile phone, which intentionally introduced background noise to improve model robustness.

For the "normal" category, smartphone recordings captured typical household acoustic environments, including conversation, mouse clicks and other ambient noises, which help the model distinguish between everyday sounds and the target events.

Methodology

The audio processing workflow adopts a structured approach, beginning with the collection of raw audio samples, followed by feature extraction through the conversion of raw audio into Mel-Frequency Energy (MFE) spectrograms to capture frequency characteristics. Subsequently, a convolutional neural network architecture is implemented for model training to enable audio classification, and finally, the trained model undergoes quantization and optimization for deployment on an Arduino platform.

Figure 3 shows the overall workflow of the Home Device Detector.

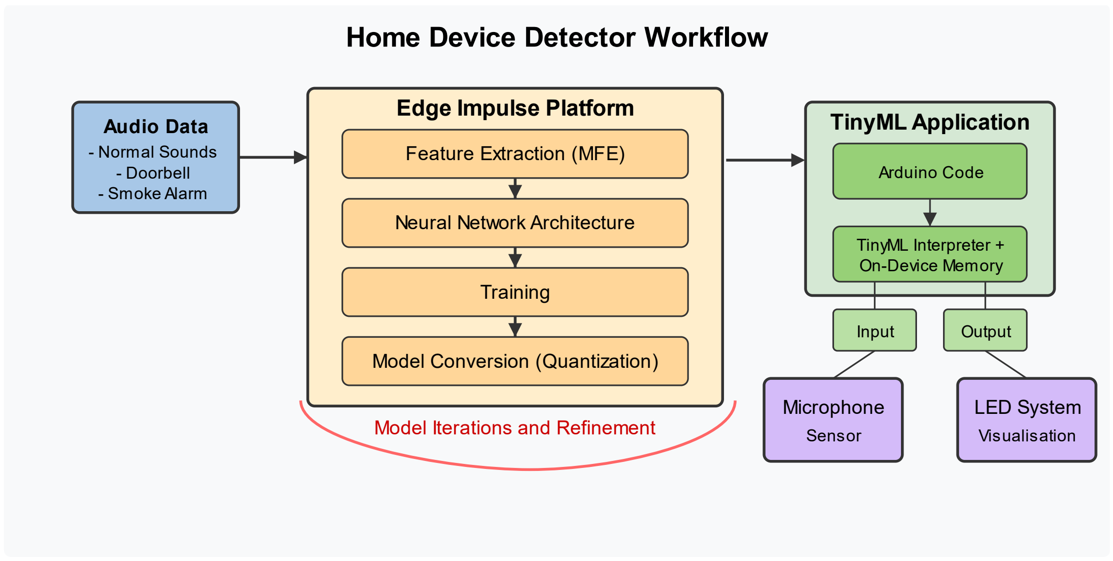


Figure 3. Workflow of Home Device Detector

3 Edge Impulse Development

3.1 Data Acquisition and Balancing

Building on the data collection strategy outlined previously, the practical implementation on Edge Impulse began with uploading and organizing the audio samples for model training. Most doorbell sounds were trimmed to approximately 1 second duration, while normal environmental sounds and smoke alarm samples were initially kept longer at 3-5 seconds.

Challenge: Data Imbalance

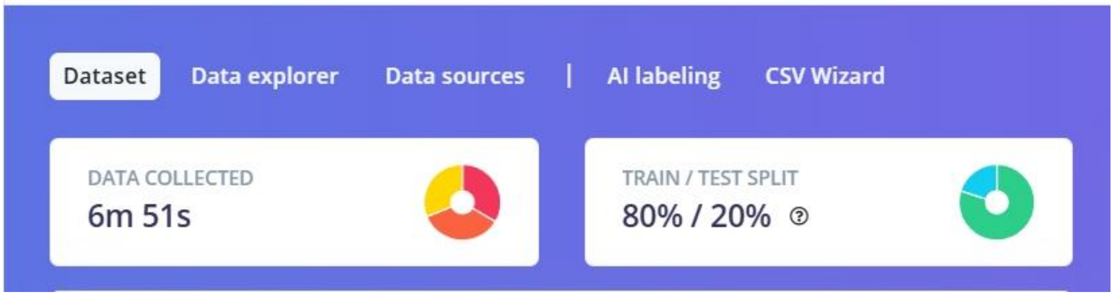
During early testing, a significant challenge emerged that severely impacted model performance. The initial dataset contained highly unbalanced class distributions, the audio length of smoke alarm and Doorbell sounds is 1min 56s and 1min 18s respectively.
This imbalance resulted in poor model performance with testing accuracy only reaching approximately 70%.

Solution: Balancing Class Distribution

I carefully adjusted the dataset to achieve better balance between classes. The final training dataset contained more equalized durations:

- Smoke alarm audio: 1 minute 42 seconds
- Doorbell sounds: 1 minute 50 seconds
- Normal environmental audio: 1 minute 56 seconds

This balanced dataset was then split into training and testing sets with an 80:20 ratio, ensuring that each set contained a representative distribution of all three sound categories, which is shown below in Figure 4.



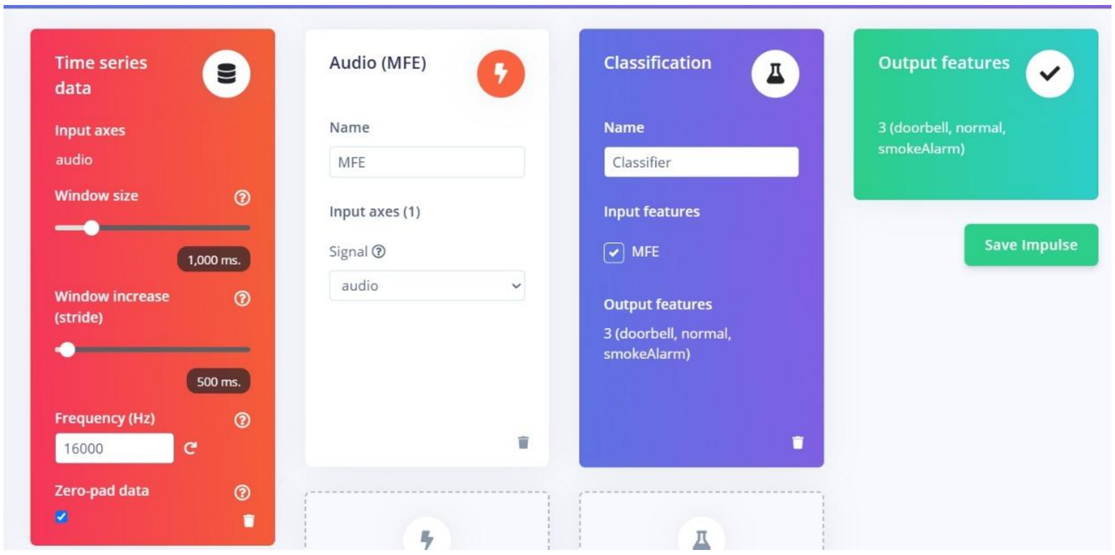
DATA COLLECTED
6m 51s

TRAIN / TEST SPLIT
80% / 20%

Figure 4. Dataset of Audio

3.2 Creating the Impulse

The next step is to creating an "impulse" - the sequence of processing blocks that transform raw sensor data into classification outputs. The figure 5 below shows the overview of the impulse.



Time series data
Input axes: audio
Window size: 1,000 ms.
Window increase (stride): 500 ms.
Frequency (Hz): 16000
Zero-pad data: ✓

Audio (MFE)
Name: MFE
Input axes (1)
Signal: audio

Classification
Name: Classifier
Input features: ✓ MFE
Output features: 3 (doorbell, normal, smokeAlarm)

Output features
3 (doorbell, normal, smokeAlarm)

Figure 5. "Create Impulse" Overview

Time Series Data Configuration

The first step was configuring the time series data parameters:

- **Window size**: 1000ms
- **Window increase**: 500ms
- **Frequency**: 16kHz sampling rate (avoid to be too high)

This configuration resulted in 16,000 sample points per window, which was determined to be an optimal balance between capturing sufficient audio information and staying within the memory limitations of the target device.

Challenge: Memory Allocation Failure

After completing initial model training and attempting to deploy to the Arduino device, I encountered a technical obstacle. The **serial monitor** displayed the following error messages:

```
Failed to start PDM! ERR: Could not allocate audio buffer (size 48000), this could be due to the window length of your model
```

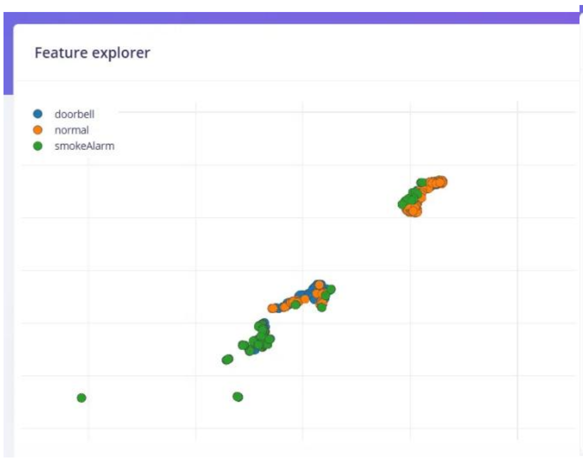
This error indicated a memory allocation failure on the device. The Arduino Nano 33 BLE Sense with only 256KB of SRAM could not allocate a buffer for 48,000 sample points (3 seconds at 16kHz).

Solution: Parameter Optimization

To resolve this deployment issue, I reconfigured the impulse with smaller parameters that would fit within the device's memory constraints, such as decrease the window size to 1 seconds:

Processing Block Selection and Configuration

I selected the **Mel-frequency Energy (MFE)** spectral features extraction method due to its effectiveness in capturing relevant frequency characteristics while requiring less computational resources; For the learning block, I selected the **Keras Classification**.
I optimized the MFE parameters for better feature separation, the most important parameter adjustment is the increase of **filter number** (from 24 to 48) and **FFT length** (from 1024 to 2048) which helped achieve better separation between the features of different sound classes, which is shown in figures 6 and 7.



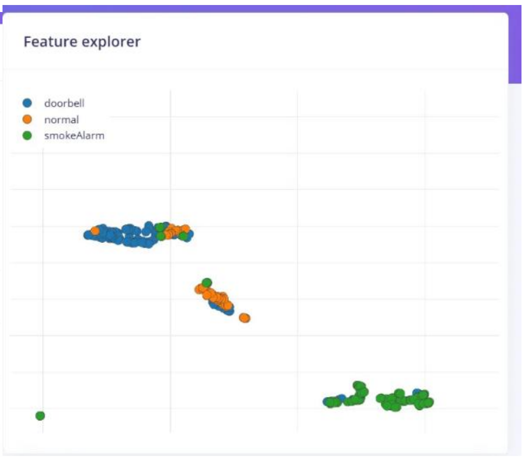


Figure 6. Audio Data Diagram with Default Parameters

Figure 7. Audio Data Diagram with Refined Parameters

3.3 Neural Network Architecture and Training

After configuring the time series data parameters and MFE processing block, the next crucial step was designing and training the neural network classifier - the core of the sound recognition system.

Neural Network Architecture

For classification, I implemented a Convolutional Neural Network (CNN) architecture specifically optimized for audio pattern recognition, which is shown below in Figure 8.

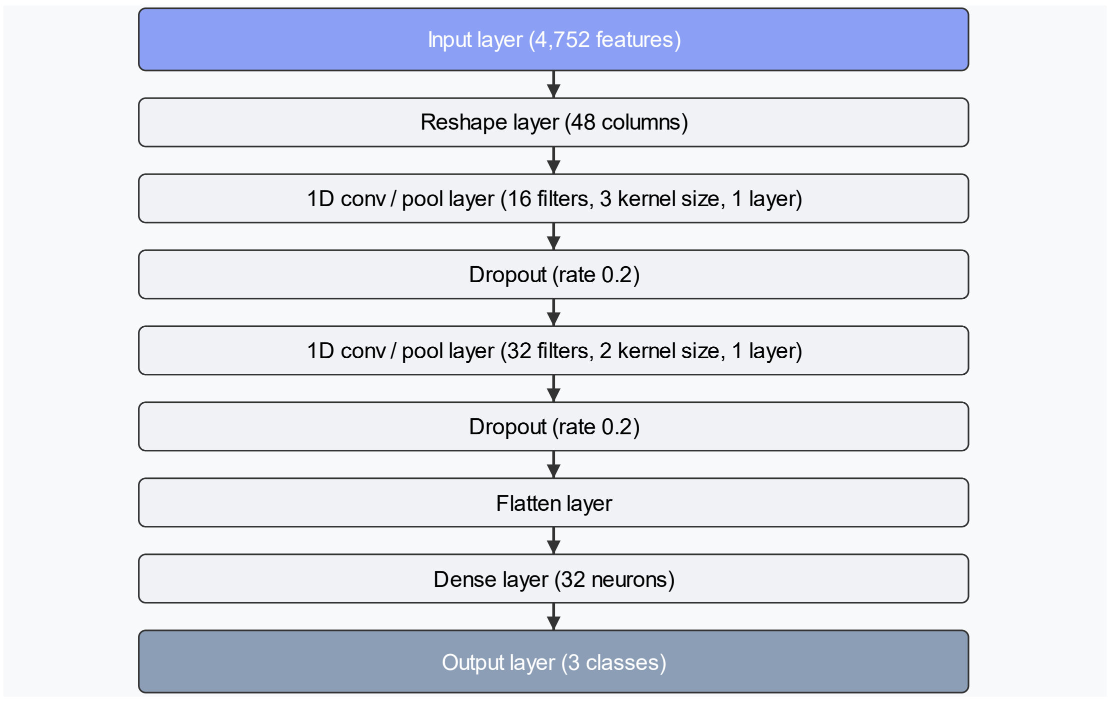

Input layer (4,752 features)
Reshape layer (48 columns)
1D conv / pool layer (16 filters, 3 kernel size, 1 layer)
Dropout (rate 0.2)
1D conv / pool layer (32 filters, 2 kernel size, 1 layer)
Dropout (rate 0.2)
Flatten layer
Dense layer (32 neurons)
Output layer (3 classes)

Figure 8. Neural Network Architecture of Home Sound Detector

Iterative Model Training Process and Test

Initial Configuration

- Training epochs: 50
- Learning rate: 0.001
- Data augmentation: Enabled with low noise addition
- **Initial results**: 86.9% accuracy with 0.45 loss

Training Duration Optimization

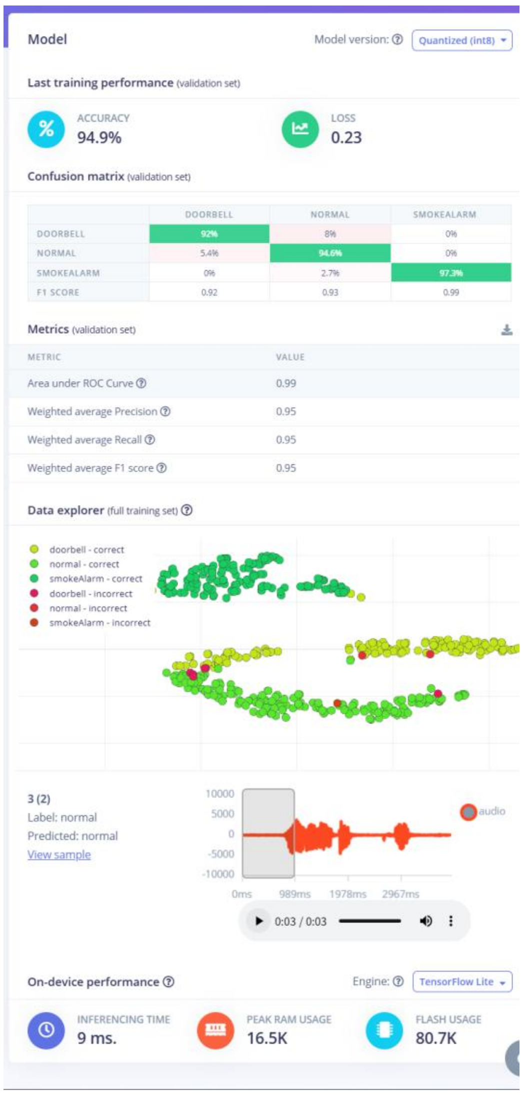
I Increased training epochs from 50 to 100 and the results improve to 88.9% accuracy with 0.38 loss. It demonstrated that the model benefited from additional training iterations.

Regularization Fine-tuning

I adjust both dropout layers to 0.2 (from default values), which reduce over-regularization issues. Through experiment with dropout rates ±0.05, I confirming 0.2 as optimal.

Architecture Enhancement

Later, dense layer was added after the flatten layer, and filter configuration was optimized((16 filters in first layer, 32 filters in the second)), which significantly improved accuracy to 94.9% with 0.25 loss, which is shown in Figure 9 below


Model — Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 94.9%

LOSS 0.23

Confusion matrix (validation set)

	DOORBELL	NORMAL	SMOKEALARM
DOORBELL	92%	8%	0%
NORMAL	5.4%	94.6%	0%
SMOKEALARM	0%	2.7%	97.3%
F1 SCORE	0.92	0.93	0.99

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	0.99
Weighted average Precision	0.95
Weighted average Recall	0.95
Weighted average F1 score	0.95

Data explorer (full training set)

3 (2)
Label: normal
Predicted: normal
View sample

On-device performance — Engine: TensorFlow Lite

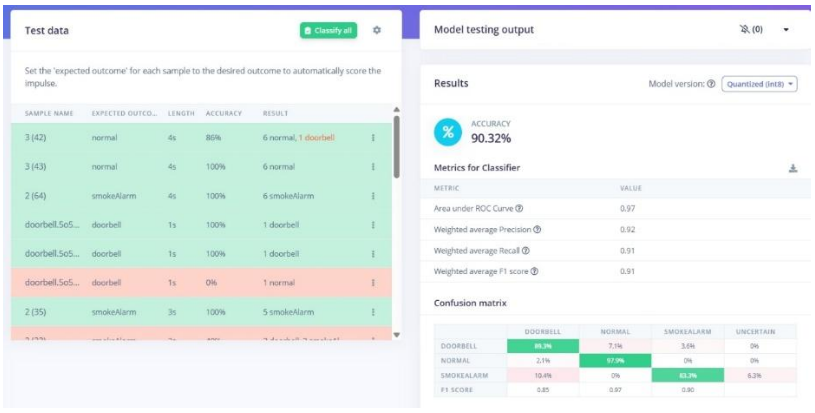
INFERENCING TIME	PEAK RAM USAGE	FLASH USAGE
9 ms.	16.5K	80.7K

Figure 9 Training Set Result

Addressing Overfitting and Refine in Test Set

Despite the high training accuracy, I observed suboptimal performance on the test set, indicating potential **overfitting** and **limited generalization ability**.

To address this issue, I Adjust kernel sizes in the 1D convolutional layers (3 for first layer, 2 for second), which can improve the model's ability to capture relevant audio patterns without memorizing training data. Therefore, test set performance is enhanced over 90% accuracy, which is shown below in Figure 10.


Test data

Set the 'expected outcome' for each sample to the desired outcome to automatically score the impulse.

SAMPLE NAME	EXPECTED OUTCO...	LENGTH	ACCURACY	RESULT
3 (42)	normal	4s	86%	6 normal, 1 doorbell
3 (43)	normal	4s	100%	6 normal
2 (64)	smokeAlarm	4s	100%	6 smokeAlarm
doorbell.5o5...	doorbell	1s	100%	1 doorbell
doorbell.5o5...	doorbell	1s	100%	1 doorbell
doorbell.5o5...	doorbell	1s	0%	1 normal
2 (35)	smokeAlarm	3s	100%	5 smokeAlarm

Model testing output

Results

Model version: Quantized (int8)

ACCURACY
90.32%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	0.97
Weighted average Precision	0.92
Weighted average Recall	0.91
Weighted average F1 score	0.91

Confusion matrix

	DOORBELL	NORMAL	SMOKEALARM	UNCERTAIN
DOORBELL	89.3%	7.1%	3.6%	0%
NORMAL	2.1%	97.9%	0%	0%
SMOKEALARM	10.4%	0%	83.3%	6.3%
F1 SCORE	0.85	0.97	0.90	

Figure 10. Testing Set Results

The final model achieved over 94% accuracy on the training set and demonstrated good generalization capability on the test set, meeting the project's target threshold for a safety-critical application.

4. Arduino Deployment and Implementation

The next phase was deploying it to the Arduino Nano 33 BLE Sense device and implementing the code to handle audio processing, inference, and visual feedback.

4.1 Model Export and Deployment

The trained model was exported from Edge Impulse as an Arduino library package (HomeDeviceDetector_inferencing.h), which included the quantized TensorFlow Lite model, pre-processing code, and an inference runner optimized for the ARM Cortex-M4 processor.

4.2 Arduino Code Implementation

The implementation focuses on continuous audio monitoring with specific optimizations for reliable detection:

Audio Capture and Buffering

```
typedef struct {
    signed short *buffers[2];
    unsigned char buf_select;
    unsigned char buf_ready;
    unsigned int buf_count;
    unsigned int n_samples;
} inference_t;
```

This double-buffering approach allows continuous audio capture while processing the previously recorded segment, ensuring no sound events are missed.

LED Visual Feedback System

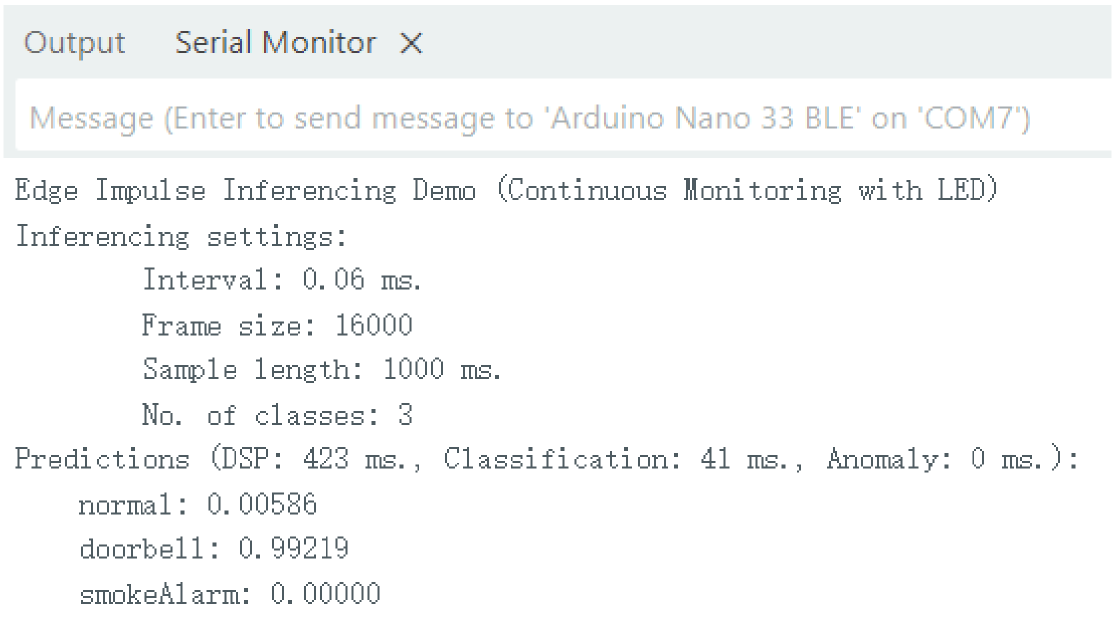
The visual feedback is implemented with distinctive blinking patterns for each sound type:

```
void update_leds(float normal_prob, float doorbell_prob, float smokealarm_prob) {
    if (normal_prob >= CONFIDENCE_THRESHOLD) {
        digitalWrite(LED_NORMAL, HIGH);
    } else {
        digitalWrite(LED_NORMAL, LOW);
    }
    if (doorbell_prob >= CONFIDENCE_THRESHOLD) {
        digitalWrite(LED_DOORBELL, HIGH);
        for (int i = 0; i < 3; i++) {
            digitalWrite(LED_DOORBELL, LOW);
            delay(100);
            digitalWrite(LED_DOORBELL, HIGH);
            delay(100);
        }
    } else {
        digitalWrite(LED_DOORBELL, LOW);
    }
    if (smokealarm_prob >= CONFIDENCE_THRESHOLD) {
        digitalWrite(LED_SMOKEALARM, HIGH);
        for (int i = 0; i < 5; i++) {
            digitalWrite(LED_SMOKEALARM, LOW);
            delay(50);
            digitalWrite(LED_SMOKEALARM, HIGH);
            delay(50);
        }
    } else {
        digitalWrite(LED_SMOKEALARM, LOW);
    }
}
```

This creates unique visual patterns:

- Normal sounds: Steady green light
- Doorbell: Yellow light
- Smoke alarm: Red light

When the device detects doorbell, the serial monitor reacts like:


Output
Serial Monitor
Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM7')

```
Edge Impulse Inferencing Demo (Continuous Monitoring with LED)
Inferencing settings:
        Interval: 0.06 ms.
        Frame size: 16000
        Sample length: 1000 ms.
        No. of classes: 3
Predictions (DSP: 423 ms., Classification: 41 ms., Anomaly: 0 ms.):
    normal: 0.00586
    doorbell: 0.99219
    smokeAlarm: 0.00000
```

Figure 11. Serial Monitor Output

4.3 Implementation Challenges

Key challenges in the code implementation included: memory management within the Arduino's limited resources, and finding the optimal confidence threshold (set at 80%).

The final implementation successfully achieved reliable sound event detection with visual feedback within the targeted response time, making it suitable for assisting DHH individuals in home environments.

5. Results and Evaluation

5.1 Results

After multiple iterations of development, the final model achieved 94.9% training accuracy and 90.32% testing accuracy with a loss of 0.23. The confusion matrix showed strong classification capabilities, with most misclassifications occurring between doorbell and smoke alarm classes due to acoustic similarities.

5.2 Critical Reflection and Experience Learnt

There are several valuable experiences I gained through this coursework.
First, data balance proved crucial - an early iteration with imbalanced classes achieved only 70% accuracy, while the balanced dataset dramatically improved performance; Second, overfitting is easy to happen and preventing overfitting is important and necessary to increase generalization ability of the model.

The system still has several limitations. In real-world testing, the device occasionally enters an "uncertain" state where no LEDs illuminate. Certain sounds in household music sometimes trigger false doorbell detections. The acoustic similarity between some doorbell and smoke alarm sounds leads to occasional misclassifications.

Future improvements could include expanding the dataset with more diverse samples to adding other important household sound categories, and solved the mis-triggering problems with more sophisticated test work and logic design.

6. Conclusion

The HomeDeviceDetector successfully demonstrates how TinyML can create practical assistive technology for DHH individuals. The system addresses an important accessibility gap while showcasing the potential of embedded ML to deliver specialized solutions where traditional cloud-based approaches might be impractical.

Bibliography

Jain, D. *et al.* (2020) 'HomeSound: An Iterative Field Deployment of an In-Home Sound Awareness System for Deaf or Hard of Hearing Users', in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. *CHI '20: CHI Conference on Human Factors in Computing Systems*, Honolulu HI USA: ACM, pp. 1–12. Available at: https://doi.org/10.1145/3313831.3376758.

Koo, G.-S. and Seo, Y.G. (2023) 'Performance Analysis of Speech Processing Techniques Using Edge Impulse', *Journal of Digital Contents Society*, 24(6), pp. 1327–1338. Available at: https://doi.org/10.9728/dcs.2023.24.6.1327.

Mishra, J., Malche, T. and Hirawat, A. (2024) 'Embedded Intelligence for Smart Home Using TinyML Approach to Keyword Spotting', in *ECSA-11*. *ECSA-11*, MDPI, p. 30. Available at: https://doi.org/10.3390/ecsa-11-20522.

Pixabay 'Door Bell Sound'. Available at: https://pixabay.com/sound-effects/search/door-bell/ (Accessed: 12 April 2025).

Sharma, S., Gupta, R. and Kumar, A. (2024) 'A TinyML solution for an IoT-based communication device for hearing impaired', *Expert Systems with Applications*, 246, p. 123147. Available at: https://doi.org/10.1016/j.eswa.2024.123147.

Uppbeat 'Smoke Alarm Sound'. Available at: https://uppbeat.io/sfx/category/alarm/smoke-alarm (Accessed: 12 April 2025).

AI Use Acknowledgment

I acknowledge the use of Claude (3.7 Sonnet, https://claude.ai/new) as an assistive tool in accordance with UCL's Category 2 guidelines. Specifically:
1. To receive structural suggestions for the report.
Prompts used: My report is about a device to detect home sound designed for DDH people in smart homes base on Edge Impulse, give me the outline of the report.
2. To ensure grammar and style consistency.
Prompts used: Check the grammar of the report and correct the wrong one and some sentence with unclear expression.
All content, analysis, and ideas presented are my own, and AI was used solely as a supportive resource, not to generate original ideas or replace my own work.