Exercise 1: Keyword Spotting

Arceta, Althea Zyrie[1], Tan, Jose Tristan[2], and Tumalad, Shawne Michael[3]

[1]althea_arceta@dlsu.edu.ph
[2]tristan_tan@dlsu.edu.ph
[3]shawne_tumalad@dlsu.edu.ph

October 2, 2024

1 Introduction

Keyword spotting (KWS) is a useful technique for speech applications that allows users to activate devices by simply saying a keyword phrase. Recent advancements in deep learning have significantly improved the accuracy and efficiency of KWS models, making them suitable for integration into even small electronic devices and web browsers.

This exercise aims to explore the design, training, and deployment of a keyword-spotting neural network using TinyML hardware and Edge Impulse. The primary objective is to build a simple neural network model capable of recognizing specific keywords or phrases and deploying it on TinyML. This exercise serves to familiarize us with the fundamentals of TinyML, including dataset creation, model design, and on-device inference.

2 Background and Related Work

New technology keeps emerging every day. Slowly integrating and changing how we live, work, and communicate, making our lives significantly easier. However, as we grew more and more accustomed to technology, we became more and more dependent on technology. Hence, the demand for communication between humans and computers has never been greater. Speech recognition is one example of this technology, which allows computers to understand and analyze verbal messages. This technology is applied to applications across various domains, from virtual assistants to automated transcription services. However, while speech recognition offers high potential, its implementation often demands high computational resources and memory. This can be challenging for edge devices with limited capabilities (Selouani, 2011). Based on the following scenario, keyword spotting (KWS) can aid in addressing the resource limitations at the edge devices.

Keyword spotting (KWS) is a technique for speech applications that allows users to activate devices by simply saying a keyword phrase (Dhungana and Salehi, 2024). Examples of wake-up words include 'Alexa,'(Amazon, 2024) 'Hey Siri,'(*Siri - Apple* 2024) and 'Okay, Google' (Google, 2024). Additionally, advancements in artificial intelligence (AI) have significantly improved human-machine interaction, especially with technologies that convert speech into executable actions (Kheddar *et al.*, 2024).

However, as the demand for smarter, more responsive devices grows, more and more technologies are being developed to meet this demand. We have now reached a stage where AI applications can be deployed on compact, low-power devices without external servers. This marks the foundation of Tiny Machine Learning (TinyML), a breakthrough that enables efficient, real-time AI processing directly on edge devices.

TinyML, unlike conventional fashions that depend on powerful cloud-based virtual servers for processing, brings intelligence at once to facet gadgets, which include microcontrollers and Internet of Things (IoT) devices. This decentralized technique enables actual-time choice-making without consistent reliance on external servers (GeeksforGeeks, 2024). TinyML works by quantizing the AI model to reduce its overall size so that it can be deployed on edge devices. This, of course, decreases accuracy, so it is part of development to optimize the model for the given microcontroller's constraints and computational power. TinyML devices can be configured to adapt and enhance themselves as time passes, ensuring the model stays updated and powerful.

TinyML is scalable, energy efficient, private, and performs in real-time. As the devices used for TinyML are small and typically excluded from the cloud, such advantages are effective in many technological sectors. A well-known example would be smartwatches, which implement TinyML in their programs for tasks such as health monitoring. Similarly, the Arduino Nano 33 BLE, which is used in this exercise, brings these benefits to an accessible platform, allowing TinyML applications to run efficiently on small devices. Its low power consumption and real-time processing capabilities make it suitable for various applications, such as edge AI applications.

The edge AI platform used in this exercise is Edge Impulse. Edge Impulse is a suite of tools that helps you build, test, and deploy edge AI algorithms. It covers the entire workflow, from data collection to model optimization and production monitoring. Edge Impulse increases the velocity of edge AI development while reducing costs and associated risks (*Edge Impulse* 2024). It simplifies the process of bringing AI and machine learning to edge devices, allowing developers to create intelligent applications that can run on small, energy-efficient hardware.

TinyML is poised to revolutionize the way AI operates at the edge, enabling smarter, more responsive devices that can function independently of cloud servers. With its scalable, energy-efficient, and privacy-focused approach, TinyML brings intelligence to microcontrollers and IoT devices, empowering real-time decision-making in a variety of applications. As this technology continues to evolve, it will undoubtedly become a cornerstone of future innovations, shaping the landscape of AI-driven solutions for years to come.

3 Methodology

When creating a keyword spotting system, start by selecting a keyword to recognize. Consider how multi-syllable words (like "Hello world") outperform single-syllable words. For our group, the selected keywords are 'Talk to me,' 'Wake up,' and 'Hey Duino.'

1. Setting up: open the platform's website and log in to your account. If you haven't already done so, create a new account. Once logged in, navigate to the project creation page and establish a new project.

2. Data Collection: Record 10s to 60s audio samples using the selected keywords, ensuring pauses between every keyword. After, use the 'Split Sample' to split the audio into segments. Repeat the process till there are at least 10 minutes of samples per class. Additionally, it is crucial to ensure that the samples include a variety of speakers with different genders, ages, accents, and vocal characteristics. This diversity will help the model generalize better across different user profiles and improve overall performance in real-world conditions.

3. Noise and Unknown: In addition to your keyword, we'll also need audio that is not your keyword. Like background noise, the TV playing ('noise' class), and humans saying other words ('unknown' class). Add pre-built datasets for 'noise' and 'unknown' classes to our dataset.

4. Balancing the Dataset: Select Perform train/test split by going to Dashboard. This will automatically split your data between a training class (80%) and a testing class (20%).

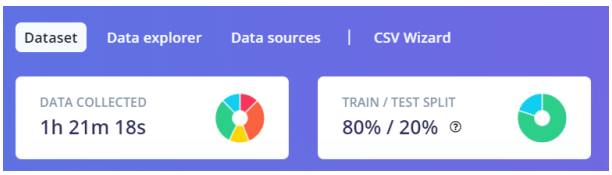


Figure 1: Balanced Dataset

5. Impulse Design: In the Create impulse tab, add Time series data, an Audio (MFCC), and a Classification (Keras) block. Leave the window size to the length of the keyword audio samples (in our case, 1000ms or 1 second) and click Save Impulse.

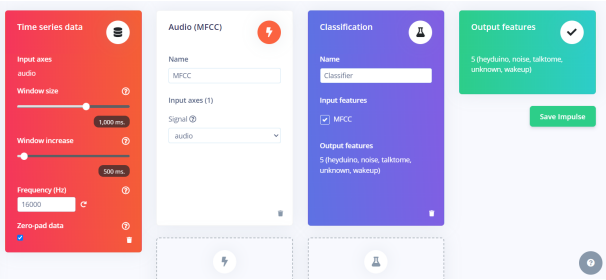


Figure 2: Impulse Design Parameters

6. Configuration of the MFCC block: After assembling the building blocks of our Impulse, click on the MFCC tab in the navigation menu on the left-hand side. In our case, we stuck to the default parameters.

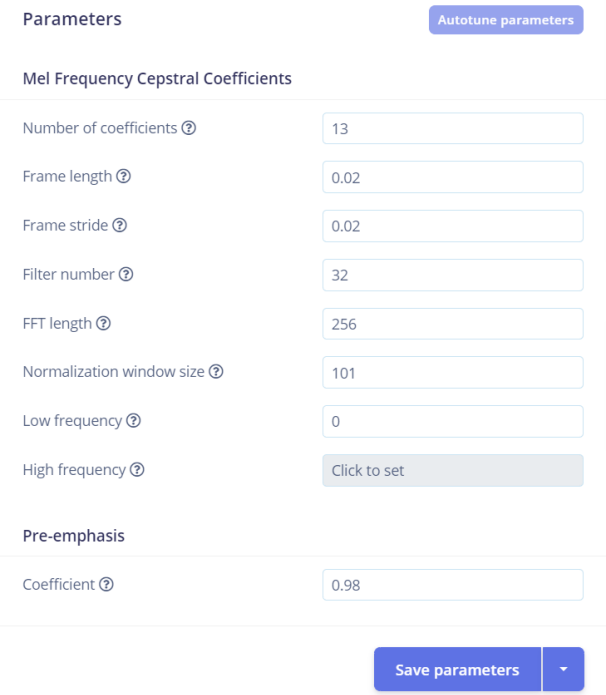


Figure 3: MFCC Block Configuration

7. Neural Network: With all data processed, it's time to start training a neural network. The network that we're training here will take the MFCC as an input. Configure the settings based on your choice on the NN Classifier tab.

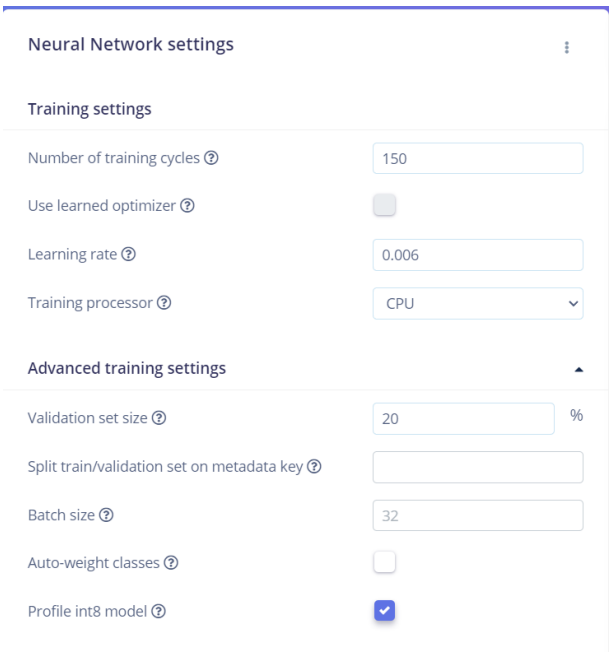


Figure 4: Neural Network Settings (1)

Figure 5: Neural Network Settings (2)

8. Testing and Validation: With everything in place, click Start training. Training will take a few minutes. When it's complete, the Last training performance panel appears at the bottom of the page. After, make sure to inspect and correct any misclassified samples or labels.

ACCURACY
92.50%

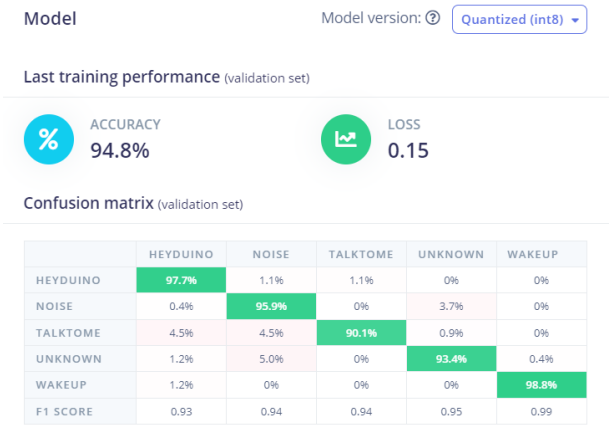
Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	0.95
Weighted average Recall	0.95
Weighted average F1 score	0.95

Confusion matrix

	HEYDUINO	NOISE	TALKTOME	UNKNOWN	WAKEUP	UNCERTAIN
HEYDUINO	96.4%	0%	0%	0.9%	0%	2.7%
NOISE	0.3%	91.6%	0%	3.6%	0%	4.5%
TALKTOME	0.9%	0%	94.0%	0%	0%	5.1%
UNKNOWN	0.3%	0.9%	0.6%	89.9%	0%	8.2%
WAKEUP	0%	0%	0%	0%	96.7%	3.3%
F1 SCORE	0.97	0.95	0.96	0.93	0.98	

Figure 6: Accuracy Result of the Test Dataset



Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY
94.8%

LOSS
0.15

Confusion matrix (validation set)

	HEYDUINO	NOISE	TALKTOME	UNKNOWN	WAKEUP
HEYDUINO	97.7%	1.1%	1.1%	0%	0%
NOISE	0.4%	95.9%	0%	3.7%	0%
TALKTOME	4.5%	4.5%	90.1%	0.9%	0%
UNKNOWN	1.2%	5.0%	0%	93.4%	0.4%
WAKEUP	1.2%	0%	0%	0%	98.8%
F1 SCORE	0.93	0.94	0.94	0.95	0.99

Figure 7: Accuracy Result of the Train Dataset

9. Deployment: After training the model, the model can be deployed to our TinyML Arduino Kit. In the Deployment tab, select 'Arduino Library' for Selected Development and configure the settings to your preferences. Click build to create a library that contains example code that can be deployed on an Arduino development board.

10. Verification: After deploying the sample code (") on the development board, the system is tested through console output to verify that the system works.

11. LED: After verifying the system, the sample code (shown below) is augmented to modify the board's LED output based on the model's readings.

```
static void keywordLED(ei_impulse_result_classification_t keyword[])
{
  int index = 0;
  // Find predicted keyword
  for (int i = 1; i < 5; i++) {
    if (keyword[i].value > keyword[index].value) {
      index = i;
    }
  }

  switch (index) {
    case 0: //heyduino
      blinkLED(3, 150, 255, 255, 0);  //Yellow
      break;
    case 1: //noise
      blinkLED(3, 150, 255, 255, 255);  //White
      break;
    case 2: //talktome
      blinkLED(3, 150, 0, 0, 255);  //Blue
      break;
    case 3: //unknown
      blinkLED(3, 150, 255, 0, 255);  //Purple
      break;
    case 4: //wakeup
      blinkLED(3, 150, 0, 255, 0);  //Green
      break;
  }

  return;
}
```

Figure 8: Code Snippet of keywordLED Function

This is done by assigning an LED color to each keyword and blinking the LED color three times to correspond to the maximum keyword value. Below is the list of keywords and their respective LED colors.

- 'Hey Duino' - Yellow
- 'Talk to me' - Blue
- 'Wake up' - Green
- 'Noise' - White
- 'Unknown' - Purple

4 Evaluation

The model will be assessed based on accuracy, precision, recall, and on-device performance. We conducted tests under various circumstances to determine performance. The following metrics were gathered:

- Accuracy: The percentage of correct predictions made by the model. To evaluate the model's accuracy, the researchers will test the model with a dataset that includes training and unseen test data.
- Precision: How well the model distinguishes between similar-sounding words. To assess the model's precision, researchers will present the model with a series of phonetically similar words or phrases (e.g., "Hey Duino" vs. "Arduino") to determine if it correctly identifies the target keyword without mistakenly triggering similar sounds.
- Adaptability: The model's ability to generalize across different voice recordings. To assess the model's adaptability, researchers will test the model under varied test conditions, including voices with different pitch ranges, genders, and ages and recordings made in different environments (e.g., quiet rooms vs. noisy streets).

5 Conclusion

Overall, this exercise provided valuable hands-on experience designing, training, and deploying a TinyML keyword-spotting neural network using Edge Impulse. We successfully implemented a foundational model incorporating a personalized dataset, achieving accurate keyword recognition on a mobile device.

Future work could delve deeper into model optimization. We may obtain even higher accuracy by adding more data or fine-tuning the neural net architecture. These enhancements would make our keyword-spotting approach more accurate in real-world circumstances.

References

Amazon (2024). *Alexa*, https://alexa.amazon.com/. (Accessed on 09/25/2024).

Dhungana, Prakash and Salehi, Sayed Ahmad (2024). "RTKWS: Real-Time Keyword Spotting Based on Integer Arithmetic for Edge Deployment", *2024 25th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–7. DOI: 10.1109/ISQED60706.2024.10528680.

Edge Impulse, (2024). https://edgeimpulse.com/faqs/. (Accessed on 10/2/2024).

GeeksforGeeks (2024). *What is TinyML? Tiny Machine Learning - GeeksforGeeks*, https://www.geeksforgeeks.org/what-is-tinyml-tiny-machine-learning/. (Accessed on 09/25/2024).

Google (2024). *Google Assistant, your own personal Google*, https://assistant.google.com/. (Accessed on 09/25/2024).

Kheddar, Hamza, Hemis, Mustapha, and Himeur, Yassine (2024). "Automatic speech recognition using advanced deep learning approaches: A survey", *Information Fusion*, Vol. 109, p. 102422. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2024.102422. **available at**: https://www.sciencedirect.com/science/article/pii/S1566253524002008.

Selouani, Sid-Ahmed (2011). *Speech Processing and Soft Computing — SpringerLink*, https://link-springer-com.dlsu.idm.oclc.org/book/10.1007/978-1-4419-9685-5. (Accessed on 09/29/2024).

Siri - Apple, (2024). https://www.apple.com/siri/. (Accessed on 09/25/2024).