Jurnal
Teknologi

Full Paper

KEYWORD SPOTTING SYSTEM WITH NANO 33 BLE SENSE USING EMBEDDED MACHINE LEARNING APPROACH

Nurul Atikah Abbas, Mohd Ridzuan Ahmad*

Department of Control and Mechatronics Engineering, Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia

Article history
Received
15 June 2022
Received in revised form
28 January 2023
Accepted
29 January 2023
Published Online
19 April 2023

*Corresponding author
mdridzuan@utm.my

Abstract

Due to the obvious advancement of artificial intelligence, keyword spotting has become a fast-growing technology that was first launched a few years ago by hidden Markov models. Keyword spotting is the technique of finding terms that have been pre-programmed into a machine learning model. However, because the keyword spotting system model will be installed on a small and resource-constrained device, it must be minimal in size. It is difficult to maintain accuracy and performance when minimizing the model size. We suggested in this paper to develop a TinyML model that responds to voice commands by detecting words that are utilized in a cascade architecture to start or control a program. The keyword detection machine learning model was built, trained, and tested using the edge impulse development platform. The technique follows the model-building workflow, which includes data collection, preprocessing, training, testing, and deployment. 'On,' 'Off,' noise, and unknown databases were obtained from the Google speech command database V1 and applied for training and testing. The MFCC was used to extract features and CNN was used to generate the model, which was then optimized and deployed on the microcontroller. The model's evaluation represents an accuracy of 84.51% based on the datasets. Finally, the KWS was successfully implemented and assessed on Arduino Nano 33 BLE Sense for two studies in terms of accuracy at three different times and by six different persons.

Keywords: Edge impulse, Keyword spotting, TinyML, MFCC, CNN

Abstrak

Disebabkan kemajuan yang jelas dalam kecerdasan buatan, pengesanan kata kunci telah menjadi teknologi yang berkembang pesat yang mula-mula dilancarkan beberapa tahun lalu oleh model Markov tersembunyi. Pengesanan kata kunci ialah teknik mencari istilah yang telah dipraprogramkan ke dalam model pembelajaran mesin. Walau bagaimanapun, kerana model sistem pengesanan kata kunci akan dipasang pada peranti yang kecil dan terhad sumber, ia mestilah bersaiz minimum. Sukar untuk mengekalkan ketepatan dan prestasi apabila meminimumkan saiz model. Kami mencadangkan dalam kertas ini untuk membangunkan model TinyML yang bertindak balas kepada arahan suara dengan mengesan perkataan yang digunakan dalam seni bina lata untuk memulakan atau mengawal program. Model pembelajaran mesin pengesanan kata kunci telah dibina, dilatih dan diuji menggunakan platform pembangunan dorongan tepi. Teknik ini mengikut aliran kerja pembinaan model, yang merangkumi pengumpulan data, prapemprosesan, latihan, ujian dan penggunaan. 'Hidup,' 'Mati,' hingar dan pangkalan data yang tidak diketahui diperoleh daripada pangkalan data arahan pertuturan Google V1 dan digunakan untuk latihan dan ujian. MFCC digunakan untuk mengekstrak ciri dan CNN digunakan untuk menjana model, yang kemudiannya dioptimumkan dan digunakan pada mikropengawal. Penilaian model mewakili ketepatan 84.51% berdasarkan set data. Akhirnya, KWS telah berjaya dilaksanakan dan dinilai pada Arduino Nano 33 BLE Sense untuk dua kajian dari segi ketepatan pada tiga masa berbeza dan oleh enam orang berbeza.

Kata kunci: Edge impulse, Kata kunci, Pembelajaran mesin kecil, MFCC, CNN

1.0 INTRODUCTION

Machine learning is an intriguing branch of Artificial Intelligence (AI) that employs a variety of techniques to intelligently manage large and complex amounts of data. It is based on institutions from a variety of fields, including statistics, knowledge creation, power, database, causal inference, computer systems, machine vision, and natural language processing. Machine learning may also be trained to accomplish activities that would normally need human intellect.

The advancement of AI techniques such as machine learning has accelerated the creation of intelligent systems [1]. Machine learning inference is commonly offloaded to the cloud, where computer resources are more abundant. Offloading, on the other side, raises the cost of latency, energy, and privacy. It also requires on-going communication connectivity, such as Wi-Fi.

The concept of AI is continually expanding, and it has entered our sights as a new choice in life. One of the most prominent advancements in machine learning has been audio or voice recognition systems. However, it is a cloud-based solution for work completion. This technique is impractical to deploy because of the demand for continual internet connectivity and privacy concerns.

Tiny machine learning was proposed for the improvement of machine learning on a small and resource-constrained device by including extremely resource-constrained hardware, software, machine learning algorithms, compilers, and tools to squeeze a machine learning model into a few kilobytes of memory [2]. TinyML is a relatively young area that combines machine learning with embedded devices. An embedded system is a tiny computing device that consumes relatively little electricity. TinyML is the greatest answer since it is a self-contained system that does machine learning directly on the IoT device. TinyML may solve the shortcoming of dependent machine learning models since it is not dependent on the internet or high-end computational resources.

TinyML analyses data and does inference on its own instead of transmitting it to the cloud. As a result, the data will be saved on the device, lowering the danger of data privacy. It would also save storage and infrastructure expenses by not having to transfer data to the cloud on a continuous basis; only relevant data would be kept after inference [3]. The model's delay or latency issue will be removed because the entire operation is completed on the device. TinyML is commonly used for keyword recognition, visual wake words, and anomaly detection [4]. The practice of finding terms that are utilized in a cascade architecture to start or run a system, such as a mobile phone that reacts to voice instructions, is known as keyword spotting or hot word detection [5],[6].

Alexa and Siri are two examples of voice-recognition-based products that are now being marketed. As the activation or wake word for a system, this gadget will detect a keyword. People are becoming increasingly interested in adopting smart technology as a result of the increasing need for technology, and there are several approaches to make the gadget smarter. These technologies have helped to feed the "Internet of Things" notion of intelligent settings such as smart buildings and smart homes [7]. Smart switches, plugs [8], light bulbs, fans, heating systems, and even blinds [9] might all be utilized to integrate smart home systems. Smart gadgets, which are usually paired with a personal assistant, enable users to do a wide range of tasks with little effort, generally from a distance via spoken commands such as Alexa turning on the lights or even through rudimentary automation systems that do not require user input [9]. However these products have limitations in terms of privacy issues, internet dependent, cloud-based dependent, big memory size and latency.

The keyword spotting system (KWS) is a system that accepts an input signal and generates a particular action after detecting a term. KWS's conventional method is based on Hidden Markov Models (HMM). However, in order to perform the machine learning, this model demands a huge memory, a vast vocabulary, and a high computation. Deep learning algorithms have been shown in recent years to give an efficient solution because of their minimal memory footprint and efficient performance. As described in [10], the Deep Neural Network (DNN) model is built on a microcontroller using optimization approaches aimed at addressing power, memory, and real-time restrictions at the edge. In this technique, the 16-bit integers model reduces inferencing time and memory footprint while maintaining accuracy. Furthermore, the inferencing time and memory footprint produce an additional improvement while contributing to a minor loss in accuracy.

Recently, more sophisticated neural network models were developed to improve the current KWS model. Convolutional neural networks (CNN) have risen in popularity in recent years as image recognition algorithms have improved, and they are widely used for KWS in embedded systems [11]. Mel-Frequency Cepstrum Coefficients (MFCC) are used as a preprocessing step before feeding the neural network to remove unwanted noise [12]. The benefit of this data structure is that it can extract features that reflect both short-term and long-term relationships using 1D convolutional operators.

In 2018, an attention-based neural network for a small-footprint keyword spotting was proposed using a convolutional recurrent neural network (CRNN). The CRNN-based attention model achieves 1.02% false rejection rate (FRR) at 1.0 false alarm (FA) per hour with only 84K parameters [13]. A low-power accelerator called MAX78000 was utilized to deploy the KWS system with the suggested CNN architecture. This model modifies according to the restrictions of the hardware and uses a neural architecture search (NAS) technique [12]. However,

a system with MFCC consumes high energy more than three times with the proposed KWS system and caused high latency [12].

To provide an economical alternative and increase performance [6], the unique depthwise separable convolutional neural network (DS-CNN) [6],[11],[14],[15] was introduced for the embedded KWS. CNN is altered on each layer of convolution, beginning with the second layer, which includes depthwise and pointwise convolutions, as well as a batch normalization layer with ReLU activation. Transform a rectangle with a variety of grid shapes to allow the model to focus more on regions with more information. Deformable Convolution Network (DCN) is the name given to this model.

The Recurrent Neural Network (RNN) is another common way of developing a smart audio processing system. RNN and MFCC are employed in the construction of the voice recognition system [16]. Because they outperform older methods such as hidden Markov models (HMMs), convolutional neural networks (CNNs), deep neural networks (DNNs), recurrent neural networks (RNNs), deformable convolutional networks (DCNs), and depthwise separable convolutional neural networks (DS-CNNs) have been used to replace them [17].

The Google Speech Commands Dataset is a standard dataset used for keyword spotting systems in audio recognition. Google Speech Command Dataset v1 was made up of 65000 single-speaker, single-word recordings of 30 distinct terms, with a total of 1881 speakers contributing to the dataset, guaranteeing a high level of speaker variety [18]. It has 35 words contributed by thousands of different individuals for v2. Based on recent research for KWS applications [6],[18],[19],[20],[21], the dataset is relabeled to form 12 classes: "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", "Go" as well as "Silence" which contains no speech utterances and "Unknown" which contains data from the remaining 20 and 25 keywords in the original dataset based on v1 and v2 [18]. The dataset is divided into three sets: training, validation, and test, with a ratio of 80:10:10 [12].

To strengthen the system's resilience, an augmentation procedure is used where this is the technique of introducing real-world background noise to be trained using audio recordings in order to improve performance [11],[12].

In this study, we focus on constructing a keyword detection system for a minimal memory-footprint model utilizing convolutional neural networks, or CNN. This project's scope is to develop a keyword spotting machine learning model for a microcontroller. It also implements optimization methods to optimize the model to a smaller memory size.

Edge impulse is employed in the development of an algorithm that decreases the energy consumption of the system in order to generate a system that suits the resource-constrained device. Edge Impulse is a development platform used to create a machine learning model for the targeted edge devices [22]. There are a few main parts: data collection, design model, training, testing, and deployment phase. The development of the model starts with data collection and design. The machine learning models provides a high accuracy if the system has a lot of training data and testing data. The model's input is an MFCC feature extracted from the Google Speech Command Dataset v1 sets. During the model's construction, the training data was supplemented with noise on each audio file to improve performance in the presence of noise. Then, training is done on the cloud and the trained model can be deployed on the edge device. The impulse can be directly run on a mobile phone or computer or it can be exported to a library or built firmware. Finally, the KWS system revealed that our suggested technique is effective independent of time or individuals preferences.

2.0 METHODOLOGY

2.1 Keyword Spotting System

Building a KWS using an Edge Impulse platform requires a few crucial steps. This system workflow started with data collection by building a dataset of voice recordings of different genders. Then, the data were labeled with the correct classes before being trained. A feature extraction, MFCC, and NN Keras Classifier were implemented in the model training and testing involving uploading files to Edge Impulse. Finally, the trained model was deployed on Arduino Nano 33 BLE Sense. The general flow involving software and hardware is summarized in Figure 1.

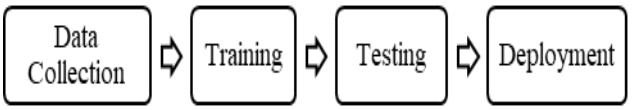

Figure 1 Project workflow

2.2 Data Collection

The data collection aimed at developing audio datasets of certain keywords. A Google Speech Commands v1 was the ideal dataset to be used in this system. 'On', 'Off', noise, and unknown are the keywords used for the dataset. 1858 audio files were uploaded and labeled 'On' in the Edge Impulse. Moreover, the 'Off' keyword consists of 1860 files with audio of various people. As many as 1123 u2nknown words were added in the dataset containing other words than 'On' and 'Off'. 1131 noise including noise from dish wash, cat meowing, exercise bike, air conditioner, airport announcement, babble, copy machine, munching, shutting the door, typing, vacuum cleaner, bus, cafe, car, field, hallway, kitchen, living room, metro, park, restaurant, chair,

tap, station, and traffic were successfully uploaded and labeled as noise.

Each audio file was ensured to be at least 1 second in length before proceeding to the training process. The dataset was automatically divided into training and testing sets with a ratio of 80:20 respectively. Therefore, a total of 1858,1860,1131 and 1123 audio files labeled 'On', 'Off', noise and unknown for training and 454 'On', 469 'Off', 263 noise and 267 unknown files for testing purposes.

2.3 Training

Training and building models are based on Edge Impulse. 80% of the data was used for training to produce 4 output features, which are On, Off, Noise, and Unknown. All audios were set to a 1 second window size with a 16 kHz before being fed to the feature extraction part. Then, it uses signal processing to extract features and a learning block to classify new data.

Due to many unrelated disturbances, the raw data signal for KWS and speech or voice recognition will not be used as input for the neural network. As a result, before feeding the data to the neural network, the signal's characteristics must be removed. For signal processing, the Mel-Frequency Cepstral Coefficient was deployed. Mel-frequency cepstral coefficients were the most common feature extraction method (MFCCs). The raw data was processed by MFCC to extract features, where a discrete cosine transform is applied to each filter bank. The data becomes easier to analyze with the classifier.

The MFCC features are the input of the learning block called as NN Keras Classifier. During this process, the data were classified based on the percentage of the identified outputs. Two hyperparameters were fixed from the training settings which are the training cycles of 100, a learning rate of 0.005 and 20 validation set size. Therefore, there are 100 epochs to train the neural network and the learning rate shows how fast the network learns. If overfitting happens, the learning rate needs to be lowered.

Figure 2 shows the neural network architecture used in the development of this model. Neural network architecture consists of an input layer and 3 1D convolutional layers with 8, 16, and 32 neurons or filters respectively with 3 kernel sizes and 1 layer. 3 layer was fixed to be used in the model because using a lower layer and a higher layer caused the degradation of the classification accuracy. Use a dropout rate of 20%–50% of neurons on average, with 20% serving as a suitable starting point. A probability that is too low has little impact, whereas a probability that is too high prevents the network from learning enough. Each convolutional layer or conv layer is followed by a dropout layer with a rate of 0.25 as it gives high performance. Dropout can reduce the risk of a model overfitting the dataset by randomly cutting a fraction of network connections during training. The last layer is the flatten layer. It flattens the multi-dimensional data into a single dimension and provides an output for the classification.

The training process is done with data augmentation. The function of this is to increase accuracy by randomly transforming data during training. Therefore, it allows the model to run more cycles without overfitting. Adding a low random noise to each spectrogram is one of the methods. The presence of noise on the spectrogram was used to evaluate the network performance.

The model trained has 4 output classes:

a. On
b. Off
c. Noise
d. Unknown

The tiny machine learning model training is to verify whether it can distinguish the keywords and classify them according to the expected results.

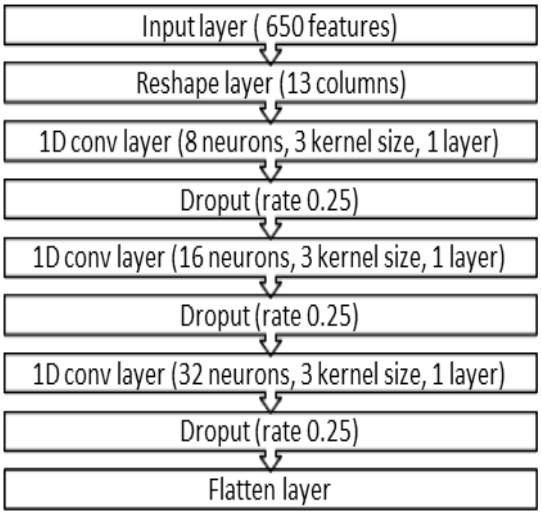

Figure 2 Neural network architecture

2.4 Testing

The testing process followed the same steps in training but with a different audio file. For each class, a model was trained and tested before deployment. The main purpose of model testing is the same as training which is to classify the specific word detected by comparing it to the existing dataset and training model. 20% of the datasets are tested to see the accuracy and performance of the model for all keywords in the dataset.

2.5 Deployment

A low-end embedded device which is Arduino Nano 33 BLE Sense was used for the inferencing of the machine learning model. This selection is due to the fact that this device has a built-in microphone for the

purpose of collecting data. Arduino Nano 33 BLE Sense meets the characteristic of tiny- embedded device with a 1MB on-board flash memory and 256KB SRAM. For deployment on Arduino Nano 33 BLE Sense, the keyword spotting model is converted to Arduino library to make the model can run without an internet connection, low latency, and low power consumption. During the conversion, the model undergoes optimization methods using quantization. The EON compiler was turned on to quantize the model to an 8-bit network for increasing the on-device performance. After quantization, the accuracy may increase a little compared to unoptimized models. However, the optimization may reduce the accuracy of the training model.

Arduino IDE was used to upload the library. Some changes have been made to the code for analyzation of the model. The built-in microphone and LED on Arduino Nano 33 BLE Sense were utilized in completing the system. The microphone was used to collect the input data which is the audio that may consist of keywords. For built-in LED, Arduino has three colors: Red, Green, and Blue. Red is used to indicate the 'Off 'keyword while green turn on when the system detects the 'On' keyword. The evaluation was made based on the detection of the 'On' and the 'Off' keywords. The system was tested on the microcontroller based on three different times and six different people. The system performance was tested at three different times is to prove the system able to response and operate well at any time. Six different people will contribute different voice tones of the input, which is the keyword of the system, and we will analyze the response of the system toward the voice. A system with high performance should be able to detect the keywords at any time and voice tone.

3.0 RESULTS AND DISCUSSION

The keyword spotting system runs the training and testing based on the dataset. The main focus of the study is to classify the voice as belonging to 'On' or 'Off'. The amount of time and number of samples used for model training and testing for all the classes are listed in Table 1. Each sample data has a sampling rate of 16 Hz.

This tiny machine learning model's training data was specifically for keyword categorization. 'On' and 'Off' keywords were learned using 80% of the total data obtained, including noise and unknown terms. Table 2 depicts the training model performance outcomes for both keywords. The results show the classification accuracy and loss during training. The training models had an accuracy of 88% and a loss of 0.36. The picture also depicts the confusion matrix with F1 Score for each class. This result indicates that the model did well in the training dataset.

According to the results, noise, 'Off,' and 'On' had greater accuracy than unknown. Noise accuracy was 98.2 percent, 93%, and 93.9%, respectively. Unknown receives just 60% accuracy due to a higher confusion categorization than other classes. 12.4% of unknown terms were identified as 'Off.' This is because the unknown words sound close to 'Off,' and the model misidentified the unknown. 21.8% were incorrectly categorized as 'On' rather than unknown. Due to the close term to 'On,' such as 'One', the model identifies the categorization of 'On' higher than unknown. As a result, the accuracy for unknown is lower than for others.

Table 1 Training and testing datasets

	On	Off	Noise	Unknown	Total
Training data	30m22s 1858 samples	30m35s 1860 samples	18m51s 1131 samples	18m43s 1123 samples	1h38m31s 5972 samples
Test data	7m27s 454 samples	7m42s 469 samples	4m23s 263 samples	4m27s 267 samples	24m 1453 samples
Total	37m49s 2312 samples	38m17s 2329 samples	23m14s 1394 samples	23m10s 1390 samples	2h26m31s 7425 samples

Table 2 Training result

Accuracy	Loss
88.0%	0.36

Confusion Matrix (Validation Set)

	Noise	Off	On	Unknown
Noise	98.2%	0.53%	0%	1.4%
Off	2.7%	93.0%	3.2%	1.1%
On	1.6%	2.4%	93.9%	2.1%
Unknown	5.8%	12.4%	21.8%	60%
F1 Score	0.93	0.92	0.89	0.72

The testing models went through the same process as the training model. The goal of the testing is to categorize the terms 'On' and 'Off.' 20% of the data collected was used for testing.

Table 3 depicts the model's performance against the test set data. The model is 91.6% accurate for noise, 90.6% accurate for 'Off,' 90.3% accurate for 'On,' and 55.4% accurate for unknown. The degradation of each class's accuracy is due to incorrect detection and uncertainty. When the model detects two classes at the same time, the situation becomes uncertain. For example, the actual word is 'House,' but the model detects it as unknown and 'Off' with nearly identical values, so the model outputs uncertain rather than a class with higher values. The accuracy of the testing model is 82.4%, which is slightly lower than that of the training model.

The model was then deployed on an Arduino Nano 33 BLE sense using an optimization model.

Table 4 depicts the quantized model running on the microcontroller. When compared to the unoptimized model, the optimized model with 8 bits has a higher accuracy. The model, which had an accuracy of 84.51 percent and a latency of 26ms, was converted to the Arduino library for deployment. Following that, some coding changes were made to ensure that the model includes led lights to indicate classification.

The coding was compiled and run on the microcontroller using the Arduino IDE, where the model is modified. Following the completion of the uploading process, the model was tested on the microcontroller itself.

Two experiments are carried out to evaluate the model's functionality and performance on the microcontroller. The first is to test the machine learning model's performance three times. The model should perform the same regardless of time. The model will detect the keywords without issue in the morning, day, and night. This is because each time has a different external noise from the environment. The model's performance is then tested using different people's voice commands. Different people have different voice tones that can cause the model to misclassify the keywords. Any voice command containing the keywords was expected to trigger a response from the model.

Table 3 Testing result

Accuracy
84.24%

	Noise	Off	On	Unknown	Uncertain
Noise	91.6%	1.1%	0.8%	0.4%	6.1%
Off	0.9%	90.6%	1.5%	0.9%	6.2%
On	0%	1.8%	90.3%	1.5%	6.4%
Unknown	6.4%	6.4%	9.4%	55.4%	22.5%
F1 Score	0.92	0.92	0.91	0.69	

Then, the results were calculated based on the student's t-test method to know the confidence interval of the data. In the student's t-test, the t-distribution table was used, which shows the critical values of the t-distribution. In this case, a one-tailed test was chosen to find the average range of the data with the upper level and lower level. The confidence interval was calculated using a 95% confidence level. A 95% confidence level shows the estimated data would match the results from the population if it was repeated and it can almost be positive that the results are the same as the samples. Therefore, without repeating the same experiments we can know the estimated mean for the result. A confidence interval for the mean is a method of estimating the true population mean with a margin of error by using this equation

$$\bar{\bar{x}} \pm t\frac{\sigma}{\sqrt{n}} \quad (1)$$

According to the results, noise, 'Off,' and 'On' had greater accuracy than unknown. Noise accuracy was 98.2 percent, 93%, and 93.9%, respectively. Unknown receives just 60% accuracy due to a higher confusion categorization than other classes. 12.4% of unknown terms were identified as 'Off.' This is because the unknown words sound close to 'Off,' and the model misidentified the unknown. 21.8% were incorrectly categorized as 'On' rather than unknown. Due to the close term to 'On,' such as one, the model identifies the categorization of 'On' higher than unknown. As a result, the accuracy for unknown is lower than for others.

3.1 Different Times

The model was tested at various times of day and night to see how it performed at different times. 'On' and 'Off' were tested at 9 a.m., 3 p.m., and 9 p.m. An accurate result from a total of 200 'On' and 'Off' cycles achieved by a single person is illustrated in Table 5.

Table 4 Optimization model

Available optimization for NN Classifier

Quantized	Confusion Matrix						
(Int 8)	RAM Usage	Latency	91.6%	1.1%	0.8%	0.4%	6.1%
	6.5K	26ms	0.9%	90.6%	1.5%	0.9%	6.2%
	Flash Usage	Accuracy	0%	1.8%	90.3%	1.5%	6.4%
	37.6K	84.51%	6.4%	6.4%	9.4%	55.4%	22.5%
Unoptimized (Float 32)	Confusion Matrix						
	RAM Usage	Latency	91.6%	1.1%	0.8%	0.4%	6.1%
	9.6K	25ms	0.9%	90.6%	1.5%	0.9%	6.2%
	Flash Usage	Accuracy	0%	1.8%	90.3%	1.5%	6.4%
	45.3K	84.24%	6.4%	6.4%	9.4%	55.4%	22.5%

During the morning, 200 times repeated keywords for each class were tested, generating positive results of 174 and 176 for 'On' and 'Off,' respectively. This achieves an accuracy of 87% for 'On' and 88% for 'Off.'

We tested again, but this time around 3 p.m., to ensure that the model fits every time. Out of 200 times, only 23 were misclassified for 'On' and 14 were misclassified for 'Off'. For 'On' and 'Off,' the accuracy was 88.5% and 93%, respectively. The accuracy has a small increasing value. This is due to the experiment at this time being done with less external noise in the environment.

Next, the performance on the night was positive, with 173 correct out of 200 for 'On' classifications and only 25 'Off' classifications misclassified. In summary, the classification accuracy for the night test is 86.5% for 'On' and 87.5% for 'Off'. Based on the data, at 3 pm the accuracy is the highest but there is not much significant difference with other times. Therefore, we can conclude that the systems are able to operate and react correctly to the keywords regardless of the

time. Figure 3 depicts the accuracy of the experiment at various times.

According to the results of the experiments, the mean of the samples for 'On' is 174.67 with a standard deviation of 2.082, while the mean for 'Off' is 179 with a standard deviation of 6.083. This yields a mean accuracy of 87.33% for 'On' and 89.5% for 'Off'. The samples were subjected to a student's t-test with a 95% confidence level. Using the one-tail t-distribution table, a t value of 4.303 was obtained with two degrees of freedom. The value of the results ranges from 174.67±5.172 with 179.84 as the upper level and 169.50 as a lower level values for 'On'. By the range from 179±15.11, the upper level for 'Off' is 184.11 and lower level is 173.88.

Table 5 Accurate inferencing data for different times

	On	Offf
9am	174	176
3pm	177	186
9pm	173	175

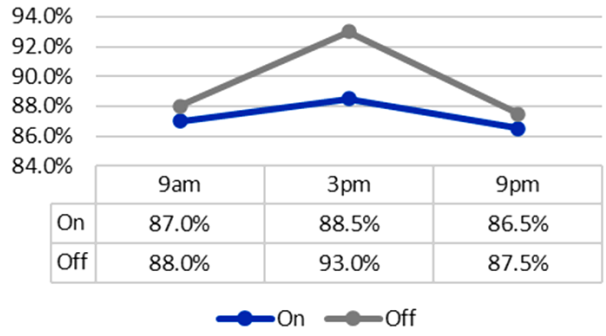

Figure 3 Accuracy for 3 different times

3.2 Different Persons

The model was evaluated using voice commands from people of various genders and ages. It was tested with six different people during the day. Two females and four males ranging in age from 16 to 52 years. Each person repeated the 'On' and 'Off' keywords 50 times. The model was tested because it should have high accuracy and good performance for any keyword spoken by anyone. The model was expected to detect the keyword regardless of the person's gender or age.

As shown in Table 6, the outcome of the experiment does not differ significantly between individuals. Persons 2 and 4 have the highest accuracy for 'On' classification with 94 %, while Persons 1 and 3 have the highest accuracy for 'Off' classification with 94 %. The system's lowest accuracy was 88 % for Person for 'Off' classification due to an uncertain class, as in Figure 4. The mean for 'On' and 'Off' is calculated to be 46.33 and 45.67, respectively.

On this data, a student's t-test was also performed after first calculating the standard deviation. The standard deviations for each class, 'On' and 'Off,' are 0.5164 and 1.2111, respectively. Using the one-tail t-distribution table, a t value of 2.571 was obtained with a 95 % confidence level and 6 samples. The value of the results ranges from 46.33±0.542 with 46.87 upper level and 45.79 lower level values for 'On'. By the range from 45.67±1.271, the upper level is 46.94 and lower level is 44.4 for 'Off'.

Table 6 Accurate inferencing data for different persons

	On	Off
Person 1	46	47
Person 2	47	45
Person 3	46	47
Person 4	47	46
Person 5	46	45
Person 6	46	44

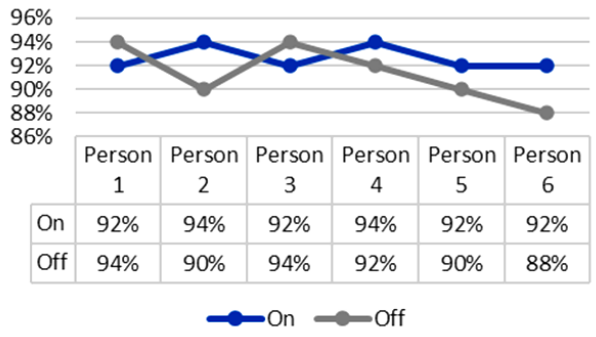

Figure 4 Result of different person's accuracy

4.0 CONCLUSION

Finally, the development of the Keyword Spotting System was completed successfully, and the system's accuracy was validated using Edge Impulse and the Arduino Nano 33 BLE Sense board. The MFCC and CNN were implemented on the model system that utilizes augmented data on Edge Impulse for the training and testing phases. After being quantized to an 8-bit integer for live validation accuracy, the model was deployed on the Arduino Nano 33 BLE Sense. According to the data, training, and testing resulted in 88% and 84.24 % accuracy, respectively. The model can acknowledge the keywords and perform successfully at any time and with every voice command according to the training and testing of the dataset using the Edge Impulse. The mean accuracy for the 'On' and 'Off' classes is 87.33 % and 89.5 % for various times, while it is 92.67 % and 91.33 % for different people respectively. This technique highlights how important data collecting was before building the model. Because of the use of an existing dataset, the model's performance is quite accurate.

Conflicts of Interest

The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper.

Acknowledgement

This work has been supported by Universiti Teknologi Malaysia and the Ministry of Higher Education through Hi-Tech (F4) Q.J130000.4623.00Q15 and FRGS/1/2020/TK0/UTM/02/43 grants, respectively.

References

[1] B. Qolomany *et al.* 2019. Leveraging Machine Learning and Big Data for Smart Buildings: A Comprehensive Survey. *IEEE Access*. 7: 90316-90356. Doi: 10.1109/ACCESS.2019.2926642.

[2] C. Banbury *et al.* 2020. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. [Online]. Available: http://arxiv.org/abs/2010.11267.

[3] A. D. I. A. Kadir, A. Al-Haiqi, and N. M. Din. 2021. A Dataset and TinyML Model for Coarse Age Classification Based on Voice Commands. *15th IEEE Malaysia Int. Conf. Commun. Emerg. Technol. IoT 5G, MICC 2021 - Proc*. 75-80. Doi: 10.1109/MICC53484.2021.9642091.

[4] V. Janapa Reddi *et al.* 2022. Widening Access to Applied Machine Learning with TinyML. *Harvard Data Sci. Rev*. 1-20. Doi: 10.1162/99608f92.762d171a.

[5] S. Choi *et al.* 2019. Temporal Convolution for Real-time Keyword Spotting on Mobile Devices.*Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*. 3372-3376. Doi: 10.21437/Interspeech.2019-1363.

[6] Y. Zhang, N. Suda, L. Lai, and V. Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. 1-14. [Online]. Available: http://arxiv.org/abs/1711.07128.

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutorials*. 17(4): 2347-2376. Doi: 10.1109/COMST.2015.2444095.

[8] F. García-Vázquez, H. A. Guerrero-Osuna, G. Ornelas-Vargas, R. Carrasco-Navarro, L. F. Luque-Vega, and E. Lopez-Neri. 2021. Design and Implementation of the e-switch for a Smart Home. *Sensors*. 21(11): 1-17. Doi: 10.3390/s21113811.

[9] L. Filipe, R. S. Peres, and R. M. Tavares. 2021. Voice-Activated Smart Home Controller Using Machine Learning. *IEEE Access*. 9(May): 66852-66863. Doi: 10.1109/ACCESS.2021.3076750.

[10] P. E. Novac, G. B. Hacene, A. Pegatoquet, B. Miramond, and V. Gripon. 2021. Quantization and Deployment of Deep Neural Networks On Microcontrollers. *Sensors*. 21(9): 1-32. Doi: 10.3390/s21092984.

[11] P. M. Sørensen, B. Epp, and T. May. 2020. A Depthwise Separable Convolutional Neural Network for Keyword Spotting on an Embedded System. *Eurasip J. Audio, Speech, Music Process*. 2020(1). Doi: 10.1186/s13636-020-00176-2.

[12] M. G. Ulkar and O. E. Okman. 2021. Ultra-Low Power Keyword Spotting at the Edge. [Online]. Available: http://arxiv.org/abs/2111.04988.

[13] C. Shan, J. Zhang, Y. Wang, and L. Xie. 2018. Attention-based End-to-end Models for Small-footprint Keyword Spotting.*Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*. 2018: 2037-2041. Doi: 10.21437/Interspeech.2018-1777.

[14] C. De la Parra, A. Guntoro, and A. Kumar. 2020. Improving Approximate Neural Networks for perception Tasks Through Specialized Optimization. *Futur. Gener. Comput. Syst.* 113: 597-606. Doi: 10.1016/j.future.2020.07.031.

[15] Y. Wei, Z. Gong, S. Yang, K. Ye, and Y. Wen. 2022. EdgeCRNN: An Edge-computing Oriented Model of Acoustic Feature Enhancement for Keyword Spotting. *J. Ambient Intell. Humaniz. Comput*. 13(3): 1525-1535. Doi: 10.1007/s12652-021-03022-1.

[16] S. Yang, Z. Gong, K. Ye, Y. Wei, Z. Huang, and Z. Huang. 2020. EdgeRNN: A Compact Speech Recognition Network with Spatio-Temporal Features for Edge Computing. *IEEE Access*. 8: 81468-81478. Doi: 10.1109/ACCESS.2020.2990974.

[17] J. S. P. Giraldo, V. Jain, and M. Verhelst. 2021. Efficient Execution of Temporal Convolutional Networks for Embedded Keyword Spotting. *IEEE Trans. Very Large Scale Integr. Syst.* 29(12): 2220-2228. Doi: 10.1109/TVLSI.2021.3120189.

[18] P. Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. [Online]. Available: http://arxiv.org/abs/1804.03209.

[19] R. Tang and J. Lin. 2018. Deep Residual Learning for Small-Footprint Keyword Spotting Raphael Tang David R. Cheriton School of Computer Science University of Waterloo. *2018 IEEE Int. Conf. Acoust. Speech Signal Process*. 5484-5488,

[20] R. Tang and J. Lin. 2017. Honk: A PyTorch Reimplementation of Convolutional Neural Networks for Keyword Spotting. 1-3. [Online]. Available: http://arxiv.org/abs/1710.06554.

[21] T. Mo, Y. Yu, M. Salameh, D. Niu, and S. Jui. 2020. Neural Architecture Search for Keyword Spotting. *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*. 1982-1986. Doi: 10.21437/Interspeech.2020-3132.

[22] P. P. Ray. 2022. A Review on TinyML: State-of-the-art and prospects. *J. King Saud Univ. - Comput. Inf. Sci.* 34(4): 1595-1623. Doi: 10.1016/j.jksuci.2021.11.019.