International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V73I3P117

Volume 73 Issue 3, 230-236, March 2025
© 2025 Seventh Sense Research Group®

Original Article

Embedded Device Keyword Spotting Model with Quantized Convolutional Neural Network

Salma A. Alhashimi[1], Ali Aliedani[2]

[1]*Department of Control and Computer Engineering, Almaaqal University, Basrah, Iraq.*
[2]*Department of Computer Engineering, University of Basrah, Basrah, Iraq.*

[1]*Corresponding Author : salma.ali.alhashimi@almaaqal.edu.iq*

Received: 24 October 2024 Revised: 15 February 2025 Accepted: 20 February 2025 Published: 28 March 2025

Abstract - *This research investigates the challenges of implementing machine learning models in keyword-spotting applications on embedded devices. A convolutional neural network for the Arabic keyword-spotting model is utilized in embedded devices represented by microcontrollers. The assessments are conducted on both the trained and pre-trained models. The QCoNet is the proposed model that considers the resource constraints of embedded devices by employing the quantization technique to decrease the necessary computational and memory capacity. Furthermore, MobileNet v4, a pre-trained model, is assessed under the same conditions. MobileNet v4 achieves a better accuracy of 99.2% compared to QCoNet, which achieves an accuracy of 96.7%. Nevertheless, it incurs additional expenses for processing and storage. The Arduino Nano 33 BLE is utilized for data collection and model deployment. With different noise level circumstances, the Arabic words were recorded. Furthermore, data augmentation techniques are employed to enhance the likelihood of the system's responsiveness to various environmental situations.*

Keywords - *Keyword spotting, Convolution Neural Network, Quantization, MobileNet v4, QCoNet.*

1. Introduction

Keyword spotting identifies particular sentences or words in an audio stream, typically intending to initiate a specific response. In recent years, it has been used in a spread of applications, such as “OK, Google” and “Hey, Siri,” that are applied to computing-constrained devices. It has the capability of recognizing speech in circumstances with a variety of levels of noise, such as a room filled with people or outdoor settings. It is utilized for command recognition, enabling users to manipulate a device or application through predefined spoken commands. They are also applicable in wake-word applications. The device is triggered to execute more voice commands and/or perform complex tasks upon detecting the wake word.

Various methodologies are employed in the design of keyword-spotting models. The early strategies included implementing Large-Vocabulary Continuous Speech Recognition (LVCSR) systems. These methodologies were employed to process the speech signal and locate particular phrases within the generated lattice structures. The lattices depict distinct sequences of phonetic units that are highly likely based on the speech data. A significant drawback of LVCSR-based Keyword Spotting (KWS) systems is their intensive computational requirements, which demand considerable processing power and result in latency issues [1].

Hidden Markov Models (HMM) have been used as an alternative method in the design of the KWS model. It relies on the probabilistic associations among observable sequences of events. A great deal of studies have been conducted in the field of speech recognition [2]. In keyword/fill HMM-based KWS, Gaussian Mixture Models (GMMs) were employed to represent the acoustic properties, leading to the generation of state emission likelihoods [3].

Deep neural networks are widely utilized in Keyword Spotting (KWS) applications and are the predominant technology in this field. Nevertheless, this comes with the drawback of requiring substantial processing resources. One attractive feature of Convolutional Neural Networks (CNNs) [4] is their capacity to restrict the number of model multiplications to comply with computational limitations. This may be achieved by altering several hyperparameters, such as filter stride, kernel size, and pooling size [5]. The current prevalent approach in machine learning for embedded devices involves training the proposed model on high-performance devices like cloud computing and then adapting the trained model to limited devices using machine learning compression techniques. The training process is conducted separately from embedded devices due to the significant computational resources it demands, which can be spared during the execution phase.

 This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

Response time, storage capacity, and power consumption are indicators of the characteristics of the required resources in machine learning models [6]. The response time can be measured by determining the throughput and latency. It can be evaluated regarding the number of Floating-point Operations per Second (FLOPS) or Multiplier-Accumulate (MAC), representing the expense of matrix computation handling in machine learning. Memory capacity plays a vital role in machine learning in providing the storage resources for storing the model and sensor data that can be used in training or inference processes. Furthermore, the number of read-and-write operations in memory has cast a shadow on processing time and energy consumption [7]. Finally, the use of energy, especially in embedded devices, has a significant effect on deploying machine learning paradigms. It can be evaluated at the expense of run time and memory operations.

The project's contributions can be succinctly summarized as follows:

1. Create an Arabic dataset appropriate for the specific scenario in which an augmentation technique will be used to increase the amount of data by introducing a noise signal into the sensor data.
2. Develop two convolutional neural networks specifically designed for audio categorization, incorporating quantization techniques to decrease the model's size.

The remaining part of this research is organized sequentially. Initially, a microcontroller is defined and explained. Subsequently, an examination of pertinent research in the domain will be presented. Currently, we will examine the sequence of machine learning models in detail. Afterwards, a thorough examination and evaluation of the execution of the suggested system will be conducted. The last segment is the concluding part.

2. Microcontroller

A microcomputer is a compact electronic device with a central processing unit, memory, and various peripheral devices in a single Integrated Circuit (IC). Microcontrollers have limited resources, Especially Regarding Memory (SRAM) and storage (Flash). The on-chip memory is considerably smaller than that of mobile devices, with a size that is three orders of magnitude smaller. It is even smaller than cloud GPUs, with a size of five to six orders of magnitude smaller [8]. The primary function of microcontrollers is to execute a designated task. These devices are frequently employed in scenarios with constrained resources, including wildlife monitoring, ocean health assessment, rescue missions, fitness tracking, troubleshooting industrial equipment, vehicle navigation systems, and unmanned aerial vehicles [9, 10]. The proliferation of Internet of Things (IoT) devices equipped with constantly operating microcontrollers is undergoing substantial and swift expansion, approaching a total of 250 billion.

3. Related Work

In general, there is extensive work adapting neural network models on embedded KWS devices to investigate three factors: performance, computation, and storage cost. Convolutional Neural Networks (CNNs) have the ability to achieve better performance in deep Keyword Spotting (KWS) tasks compared to fully connected Feedforward Neural Networks (FFNNs) while using fewer parameters. [11] utilized a conventional multilayer perceptron to attain substantial enhancements compared to prior HMM-based methods. Sainath and Parada expanded upon the previous research and attained superior outcomes by employing Convolutional Neural Networks (CNNs). The work in [12] handles the performance of the KWS system by shifting the required processing to the server side without considering the threat to user privacy and the end device draining power resources due to the connection to the server side, even in true negative and false positive conditions. In [13] introduced, the KWS model was introduced using a time-delay neural network. The work reduced the computation cost by skipping the framing aspect. [14] used a deep learning model represented by 3 convolution layers for implementing keyword spotting in Raspberry pi3. The work investigated the accuracy-related to phonetic variation by training the proposed system on global and local datasets. ResNet, which stands for Residual Neural Network, has been used in [15] to improve the accuracy of the keyword spotting model. [16] developed a multilingual keyword spotting system that identified the emergency keyword "help" in four languages: English, Arabic, Kurdish, and Malay. Arduino Nicla Vision was adopted in [17] for 14 English keywords that can be applied in home automation applications. In [18], a keyword-detection system based on a deep convolutional spiking neural network was proposed.

In embedded systems, quantization methods are essential, particularly for Convolutional Neural Networks (CNNs) and color processing. Fixed-point quantization can substantially decrease storage needs, computational requirements, and energy usage in embedded devices [19]; regarding color quantization, methods involving image decomposition and self-organized neural network classifiers have been suggested, offering efficient implementation for embedded systems [20]. These approaches divide images into smaller segments or windows, minimizing on-chip memory requirements and allowing for rapid processing with reduced power consumption. For CNNs, researchers have developed various quantization techniques to create stable fixed-point networks that maintain their performance levels [19]. A methodical approach to developing energy-efficient embedded systems involves initial algorithm creation using high-level programming languages, followed by quantization analysis and implementation in an assembly language specific to the target chip [21]. This strategy is particularly important for mobile and personal medical devices where battery longevity is a critical factor.

4. Tiny Machine Learning Life Cycle

It refers to the stages of developing and deploying the process of machine learning. It involves the following stages.

4.1. Feasibility Study

It examines the extent to which machine learning may be applied, particularly in the context of tiny machine learning in embedded devices, considering the constraints imposed by hardware and software. Moreover, it assesses the characteristics of the observed environment, including requirements for data, such as the types of data sources and the level of data accuracy.

4.2. Data Preparation

It covers nominating relevant data that simulates the applied data in the applied domain. In addition, it requires gathering a considerable amount of data used in training, validation, and test models. The complexity of the involved features and the included noisy data scaled the amount of data. The transformation of the raw data to the applied format, labeling data, and cleaning data processes are required to build suitable machine learning models. Furthermore, it can be appended with data processing techniques such as normalization and augmentation.

Concerning audio applications, the preparation step within the training and inference phases involves feature extraction and representation of the received signals in image format. Mel Frequency Cepstrum Coefficient (MFCC) is commonly used. Prior research has demonstrated that the human voice is discernible at significantly lower frequencies [15]. As a result, raw spectrograms lack the full extent of their potential meaning. An effective approach to improving this situation is to assess the frequencies of the spectrogram in decibels rather than the raw amplitudes. Due to the logarithmic nature of decibels, spectrograms effectively represent the extensive range of frequencies on a more easily comprehensible scale.

The MFCC filter involves dividing the audio stream into overlapping parts. The Fast Fourier Transform (FFT) calculates the power spectrum. The Mel-filterbank is implemented using the following formula to catch variations in low frequencies. The discrete cosine transform is employed. The purpose of this is to remove the relationship between the energies of the filter bank, which were computed using overlapping filters.

$$M(f) = 1125 \ln (1 + \frac{f}{700})$$

4.3. Design Model

The process involves determining the quantity of neurons, structuring them into multiple layers, and establishing connections between these layers. A convolutional neural network (CNN or ConvNet), a type of deep learning architecture, learns directly from input data. CNNs excel at identifying patterns within images for object recognition, classification, and categorization. They are also effective in classifying audio, time series, and signal data. During training, filters were applied to each image at various resolutions, with the resulting convolved image serving as input for the subsequent layer.

Furthermore, activation functions (including sigmoid, tanh, and Rectified Linear Units (RELU)) were employed to capture intricate data patterns. After feeding the input data to the input layer, the CNN comprises the following parts: The next layer does the job of extracting features from the input images by applying filters called convolutional layers. To reduce spatial dimensions while retaining important features, pooling layers are used to compute the average value in a region or take the maximum value in a region. To enhance the generalization and reduce overfitting, batch normalization and dropout approaches are applied. Finally, the image category was predicted at the classification layer. This can be achieved using sigmoid activation and binary classification or softmax activation to obtain the probability distribution of various classification aspects.

The criteria can be categorized according to a systematic experiment that examines the relationship between the model's performance and the scale of its architecture. Furthermore, pruning processes can be employed during the learning phase to eliminate superfluous neurons. An alternative approach to specifying the model (transfer learning) involves starting with a pre-trained neural network that does similar tasks and using it as an initial prototype that can be fine-tuned depending on the patterns observed in the input data. A major challenge in deploying machine learning models on embedded devices is the absence of a uniform operating system and system architecture across different devices. Embedded device designers typically prioritize cost and size reduction, as evidenced by the diverse selection of device frameworks available. This project employs Tensorflow, a popular machine-learning framework developed by Google. The platform provides a wide range of tools and libraries that may be used to build machine-learning models. It is compatible with a diverse array of hardware.

4.4. Training Model

Tensorflow Lite is a lightweight edition of Tensorflow that applies to devices with constrained resources. It supports a subset of operations required to run the inference on the edge device. It provides an optimization approach, such as quantizations, to shrink the trained models with less influence on the system's performance. In addition, it converts the trained models to flat buffer form, an array of bytes that facilitates the load and storage on microcontrollers, especially since most microcontrollers have file systems. Further, the Tensorflow Lite interpreter is adopted to run the model on the embedded device.

4.5. Model Evaluation

During the training phase, the model that has been trained is assessed as part of the validation process. Neural network validation, often referred to as validation during training or evaluation of the validation set, is an essential and pivotal stage in the training process of a neural network. It involves regularly assessing the model's performance on a distinct subset of the training dataset not utilized for training purposes. This enables the practitioner to evaluate the model's performance on data that has not been previously observed and determine its capacity to generalize. The validation process aids in identifying overfitting, optimizing hyperparameters, and selecting the most effective model. The gathered dataset is divided into two separate segments. One application of the training is seen in the training and validation datasets. The remaining portion of the dataset is utilized for model testing.

4.6. Model Deployment

Once the trained model has been tested, the acceptable test metrics, such as accuracy, loss rate, and inference time, have been fulfilled. Before calling the models and identifying the phrases, it is necessary to perform initialization actions. Figure 1 depicts the primary components of the executed code on the embedded device. The process starts with the startup phases, including tasks such as declaring variables, initializing the interpreter, allocating memory for intermediate results, and specifying the sensor data sources. Moreover, the primary component of the loop involves a series of actions to record the audio stream from the microphone, extract the feature by converting the signal to the frequency domain and calculating MFCC, and subsequently provide the input to the model. Ultimately, acknowledge and act upon the given directive.

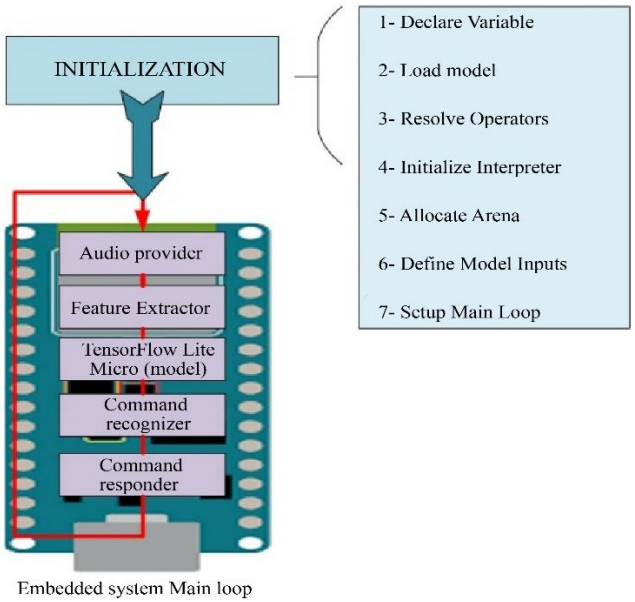

Fig. 1 Main function parts implemented while running the model on the embedded systems. These parts include initialization and the main loop parts, which are executed periodically

5. Implementation

This study involves the assessment of an Arabic keyword-spotting model utilizing two convolutional neural networks. One specific convolutional neural network is created and evaluated, while the other utilizes a pre-existing convolutional network (MobileNet v4).

Based on references [21, 24], it is estimated that around 86.8% of the world's population lacks any level of competency in the English language. Since the introduction of computers, the majority of the computer ecosystem has been created using the English language.

Currently, speech recognition models for languages other than English are inaccurate due to the widespread use of English as the predominant language for audio data [22]. In order to make this ecosystem accessible to end users across various products, services, and utilities, it is imperative to translate it into several languages. Initially, we organized our dataset to standardize the gathered sensor data in the operational area.

An inherent obstacle in implementing neural networks is the disparity that exists between the datasets used for training and those used for actual execution. That can arise from the presence of extra noise in the work environment and fluctuations in the accent. The dataset has been tagged with four keywords (go, back, left, and right), which can be used to guide the robot. Furthermore, a comprehensive data augmentation technique is employed to encompass a diverse array of potential sensor data. The model under consideration is trained via edge impulse cloud computing.

The keyword spot process pipeline consists of four main stages. It starts by capturing the audio signal from the microphone as a waveform signal. Arduino Nano 33 BLE Sense Rev2 is used in this project. It is based on a 64 MHz Arm Cortex-M4F processor (with FPU), 1 MB of flash, and 256 KB of RAM.

After that, the spectrogram of the received raw signal is computed with 13 cepstral coefficients after transferring the signal to the frequency domain by FFT with 650 features, as demonstrated in Figure 2. The third stage represents the inferring process of the convolutional neural network. Firstly, the proposed model, named QConNet, as shown in Figure 3, is deployed.

In this model, two sequential convolutional neural layers are adopted with adjectives to capture the complex features. This convolutional layer configuration describes a layer with 8 filters, each using a 3x3 kernel, constrained by a maximum norm of 1, applying 'same' padding to the input, and using the ReLU activation function. In addition, it operates with a dropout layer with a rate of 0.5.

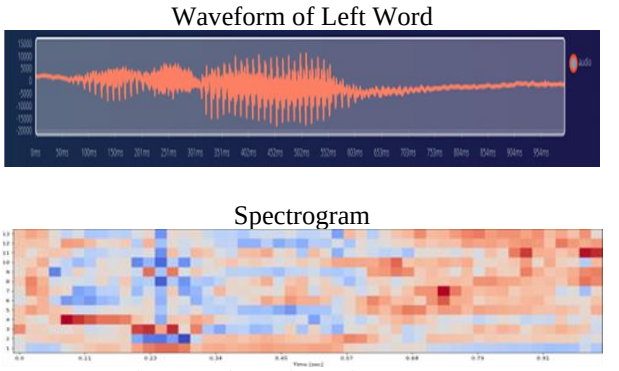

Fig. 2 Example of the waveform of the left word and the corresponding spectrogram

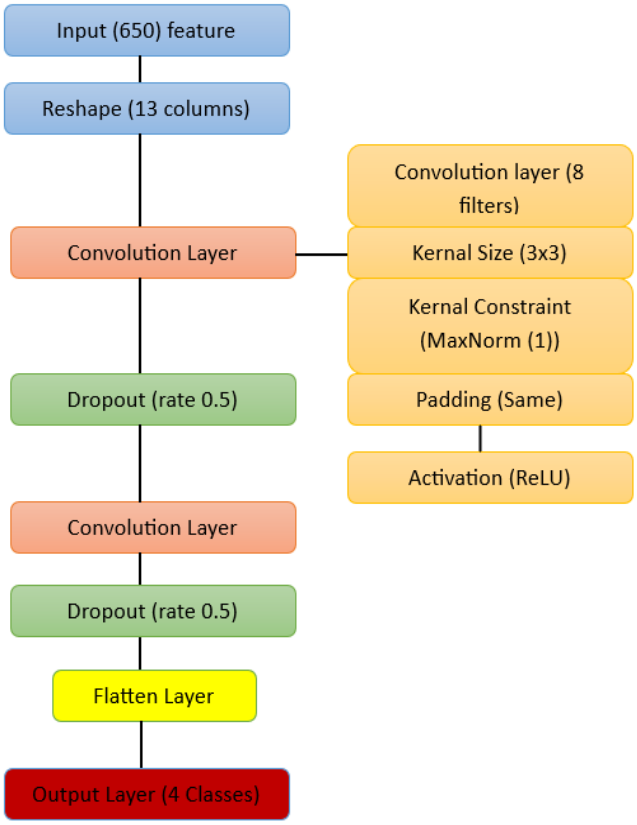

Fig. 3 Architecture overview of the QConNet model

Dropout layers are a form of regularization used in neural networks to reduce the possibility of overfitting. During training, units and their connections are randomly removed from the neural network. This technique reduces dependence on individual features and compels the network to acquire more resilient features.

The test model has an accuracy of 96.7%. Figure 3 illustrates the model's precision in terms of the F1 score. The F1 score is a statistic utilized to evaluate the effectiveness of a classification model. The computation is derived from the harmonic mean of the recall and precision metrics of the test model. Recall is the proportion of correctly classified positive samples (true positives) out of all the actual positive samples (true positives and false negatives), while precision is the proportion of correctly classified positive samples (true positives) out of all positive classification efforts (true positives and false positives).

$$recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F1 = \frac{2}{recall^{-1} + precision^{-1}}$$

	BACK	GO	LEFT	RIGHT
BACK	100%	0%	0%	0%
GO	0%	100%	0%	0%
LEFT	5.6%	5.6%	88.9%	0%
RIGHT	0%	0%	0%	100%
F1 SCORE	0.96%	0.96%	0.94%	1.00%

Fig. 4 The training confusion matrix

The Second approach, MobileNet v4 [23], is used on the same dataset. It achieves better accuracy than QConNet at the expense of higher computation costs.

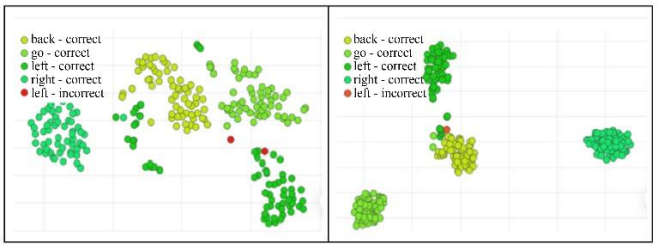

Fig. 5 Data exploration of the same train dataset with respect to QConNet (left side) and MobileNet v4 (right side)

Table 1. The usage of RAM and FLASH and the processing time required for both QConNet and MobileNet V4

	Inference Time	**RAM Usage**	**Flash Usage**
QConNet	33 ms	12K	54.3K
MobileNet V4	532 ms	205K	677.6K

As shown in the tables above, MobileNet v4 outperforms QCoNet in terms of accuracy, achieving 99.2% compared to QCoNet's 96.7%. However, this higher accuracy comes at the cost of increased RAM and Flash memory usage. MobileNet v4 requires 205 KB of RAM and 677.6 KB of Flash, whereas QCoNet utilizes only 12 KB of RAM and 543 KB of Flash. Additionally, MobileNet v4 has a longer inference time, exceeding QCoNet's by approximately 500 ms. The quantization process reduces QCoNet's model size by 50%

while maintaining an accuracy of 96.7%. Although MobileNet v4 experiences a slight accuracy drop after quantization, it remains computationally expensive.

MobileNet v4 is a pre-trained convolutional neural network created by Google specifically designed to achieve high accuracy while working within the limitations of edge devices. The model reduces the number of parameters and computation cost by employing depthwise separable convolutions. This is achieved by splitting the regular convolutional process into separate depthwise and pointwise convolutional layers. During a depthwise convolutional layer, each input channel of the feature map is convolved with a separate filter. This technique maintains the depth of the feature map, which refers to the number of channels, while simultaneously capturing spatial information within each channel. The pointwise convolution is performed by convolving the intermediate feature maps derived from the depthwise convolution with a collection of 1x1 filters. It has the ability to adjust the number of output channels, allowing for the manipulation of dimensionality by either decreasing or increasing it as needed.

To improve the robustness of the application by reducing the rate of false positives. The continuous filtering of detecting keywords was deployed on the edge device. The inference process includes storing temporary instances of the detecting keyword in a buffer to reduce the chance that the detecting words are part of other words that can give different meanings. For example, in the English language, the go word can become "goodbye," "goal," or "gold." In technical terms, the audio is sampled simultaneously with the inferencing and output processes. While the inference is being executed, audio sampling concurrently proceeds as a background activity. Twofold buffering technology is employed. A single buffer is used for the audio sampling process, which is filled with recent sample data. The second buffer is used for the inference process. It retrieves sample data from the buffer, extracts the features, and performs the inference.

6. Conclusion

The integration of machine learning techniques and embedded systems has become prevalent in a wide array of applications. The limited resources of embedded systems have hindered the use of machine learning methods that demand substantial processing and storage capabilities. Embedded systems necessitate compression approaches that decrease the size of the implementation model and simplify processing complexity in order to locate the machine learning model. Quantization plays a crucial role in decreasing the size of model parameters, hence reducing the computational workload. However, this refers to measuring model performance, which may be quantified by accuracy and inference time. The impact of a decrease on system performance can be assessed by considering the practical uses of the model. For instance, the medical application necessitates a high level of system accuracy, which poses a potential risk to the patient's life. Conversely, the application for home automation can function effectively with minimal compromise in model accuracy. This study involved the evaluation of two convolutional neural networks for a microcontroller system. They provide varying levels of accuracy and storage capacity needs.

References

[1] Ming Sun et al., “Compressed Time Delay Neural Network for Small-Footprint Keyword Spotting,” *Amazon Science*, pp. 1-5, 2017. [Google Scholar] [Publisher Link]

[2] B.H. Juang, and L.R. Rabiner, “Hidden Markov Models for Speech Recognition,” *Technometrics*, vol. 33, no. 3, pp. 251-272, 1991. [CrossRef] [Google Scholar] [Publisher Link]

[3] J.G. Wilpon, L.G. Miller, and P. Modi, “Improvements and Applications for Key Word Recognition Using Hidden Markov Modeling Techniques,” *International Conference on Acoustics, Speech, and Signal Processing*, Canada, vol. 1, pp. 309-312 1991. [CrossRef] [Google Scholar] [Publisher Link]

[4] Tara N. Sainath, and Carolina Parada, “Convolutional Neural Networks for Small Footprint Keyword Spotting,” *Interspeech*, pp. 1478-1482, 2015. [CrossRef] [Publisher Link]

[5] Iván López-Espejo et al., “Deep Spoken Keyword Spotting: An Overview,” *IEEE Access*, vol. 10, pp. 4169-4199, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[6] Sauptik Dhar et al., “A Survey of On-Device Machine Learning: An Algorithms and Learning Theory Perspective,” *ACM Transactions on Internet of Things*, vol. 2, no. 3, pp. 1-49, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Vivienne Sze et al., “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[8] Ji Lin et al., “Mcunet: Tiny Deep Learning on IoT Devices,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 11711-11722, 2020. [Google Scholar] [Publisher Link]

[9] Swapnil Sayan Saha et al., “Tinyodom: Hardware-Aware Efficient Neural Inertial Navigation,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 2, pp. 1-32, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[10] Swapnil Sayan Saha et al., "Auritus: An Open-Source Optimization Toolkit for Training and Development of Human Movement Models and Filters Using Earables," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 2, pp. 1-3, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[11] Guoguo Chen, Carolina Parada, and Georg Heigold, "Small-Footprint Keyword Spotting Using Deep Neural Networks," *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, pp. 4087-4091, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[12] Assaf Hurwitz Michaely et al., "Keyword Spotting for Google Assistant Using Contextual Speech Recognition," *IEEE Automatic Speech Recognition and Understanding Workshop*, Okinawa, Japan, pp. 272-278, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[13] Samuel Myer, and Vikrant Singh Tomar, "Efficient Keyword Spotting Using Time Delay Neural Networks," *arXiv*, pp. 1-5, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[14] Yonatan Alon, "Real-Time Low-Resource Phoneme Recognition on Edge Devices," *arXiv*, pp. 1-20, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[15] Jong Ee Jeoung, You Kok Yeow, and Muhammad Mun`im Ahmad Zabidi, "Keyword Spotting on Embedded System with Deep Learning," *Proceedings of 2019 Electrical Engineering Symposium*, vol. 3, pp. 87-91, 2019. [Google Scholar]

[16] Danyar Nabaz, Noraimi Shafie, and Azizul Azizan, "Design of Emergency Keyword Recognition Using Arduino Nano BLE Sense 33 And EdgeImpulse," *Open International Journal of Informatics*, vol. 11, no. 2, pp. 46-57, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[17] Jyoti Mishra, Tomithy Malche, and Amit Hirawat, "Embedded Intelligence for Smart Home Using TinyML Approach to Keyword Spotting," *Engineering Proceedings*, vol. 82, no, 1, pp. 1-9, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[18] Emre Yılmaz et al., Deep Convolutional Spiking Neural Networks for Keyword Spotting," *Proceedings of the Interspeech*, China, pp. 2557-2561, 2020. [Google Scholar] [Publisher Link]

[19] Motaz Al-Hami et al., "Towards a Stable Quantized Convolutional Neural Networks: An Embedded Perspective," *10th International Conference on Agents and Artificial Intelligence*, vol. 2, pp. 573-580, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[20] A. Atsalakis et al., "Colour Quantisation Technique Based on Image Decomposition and its Embedded System Implementation," *IEE Proceedings - Vision, Image and Signal Processing*, vol. 151, no. 6, pp. 511-524, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[21] Neeraj Magotra et al., "Real-Time Energy Efficient Embedded System Development Methodology," *2013 IEEE Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE)*, Napa, CA, USA, pp. 284-289, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[22] Yundong Zhang et al., "Hello Edge: Keyword Spotting on Microcontrollers," *arXiv*, pp. 1-14, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[23] Raphael Tang, and Jimmy Lin, "Deep Residual Learning for Small-Footprint Keyword Spotting," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, AB, Canada, pp. 5484-5488, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[24] Andrew G. Howard et al., "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv*, pp. 1-9, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[25] David Crystal, *English as a Global Language*, 2nd ed., Cambridge University Press, pp. 1-229, 2003. [Google Scholar] [Publisher Link]

[26] Pete Warden, and Daniel Situnayake, *Tinyml: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers*, O'Reilly Media, pp. 1-149, 2020. [Google Scholar] [Publisher Link]