EAI Endorsed Transactions
on AI and Robotics

Research Article EAI.EU

Lightweight Keyword Spotting with Inter-Domain Interaction and Attention for Real-Time Voice-Controlled Robotics

Hien Vu Pham[1], Thuy Phuong Vu[1], Huong Thi Nguyen[1], Minhhuy Le[1,2*]

[1]Intelligent Communication System Laboratory (ICSLab), Phenikaa University, Hanoi, Vietnam
[2]Faculty of Electrical Electronic Engineering, Phenikaa University, Hanoi, Vietnam

Abstract

Voice-controlled systems are increasingly essential in robotics, enabling intuitive, hands-free interaction. However, deploying Keyword Spotting (KWS) models on resource-constrained microcontrollers remains challenging, as existing approaches often sacrifice either efficiency or accuracy. This study proposes a lightweight KWS model optimized for embedded systems, balancing computational efficiency and recognition performance. The model leverages inter-domain interaction to extract spectral (MFCC) and temporal features, with an attention mechanism enhancing keyword detection. It achieves 93.70% accuracy on Google Command v2-12, with 0.359s inference time, 34.9KB peak RAM, and 98.7KB flash usage, outperforming DS-CNN-S in speed and memory efficiency. These results demonstrate the feasibility of real-time, voice-controlled robotics on low-power microcontrollers, paving the way for efficient, embedded speech recognition systems.

Received on 19 November 2024; accepted on 11 March 2025; published on 18 March 2025

Keywords: TinyML, Speech Commands, Channel Attention, Keyword Spotting

doi:10.4108/airo.7877

1. Introduction

Hands-free control offers significant potential in robotics, enabling intuitive interaction with robots and end-effectors through voice commands. By integrating Keyword Spotting (KWS) technology, robotic systems can efficiently respond to predefined voice commands, eliminating the need for physical interfaces or controllers. This approach is particularly advantageous in scenarios where manual operation is impractical or hazardous, such as industrial automation, healthcare, or search and rescue missions.

The deployment of KWS on resource-constrained microcontrollers is especially promising for lightweight robotic platforms with limited computational and power resources. Unlike traditional continuous speech recognition systems, which are resource-intensive, KWS provides an efficient solution suitable for real-time operation on low-power devices. This capability allows the development of portable, low-cost robotic systems equipped with voice command functionality, enabling seamless operation in diverse environments. For example, end-effectors can execute tasks like picking, placing, or assembling components in response to simple spoken commands.

To showcase its versatility, we apply Keyword Spotting (KWS) within a non-destructive testing (NDT) system to control sensor movement, thereby demonstrating its effectiveness in a controlled environment. The low latency, minimal memory footprint, and high accuracy of KWS models underscore their suitability for such applications, laying a strong foundation for future advancements in voice-activated robotic systems on resource-limited hardware platforms.

In this study, we present a lightweight deep-learning framework for KWS that is possible to run in real-time on a low-resource microcontroller. The proposed model leverages inter-domain interactions, enabling it to effectively assimilate information from both the Mel-frequency cepstral coefficients (MFCCs) domain [2] and the temporal domain. Moreover, an attention

*Corresponding author. Email: huy.leminh@phenikaa-uni.edu.vn

mechanism is incorporated to facilitate a comprehensive exploration of the significance associated with each channel. Additionally, our investigation includes empirical analyses aimed at assessing the impact of two distinct pooling methodologies, namely Average Pooling and Max Pooling, on the weighting of channel attention. We tested the proposed framework on the Google Command v2-12 commands dataset [1], which has ten predefined keywords, as well as categories for silence and unknown words. The proposed model achieves 93.70% accuracy, which is 6.1% surpassing the benchmark's model (i.e., DS-CNN [3]). Implementing on microcontrollers (e.g., Arduino Nano 33 BLE Sense), the model requires only 34.9KB peak RAM and 98.7KB flash memories and takes 0.359 seconds in the inference time, which is 3 times faster than the benchmarks' model.

The structure of this paper is organized as follows: Section 2 provides a comprehensive review of related work, offering the necessary context and background to situate the proposed approach within the existing body of research. Section 3 presents a detailed explanation of the features utilized in the system. The proposed methodology is elaborated upon in Section 4. Section 5 describes the experimental setup and reports the results obtained from real-world recordings, emphasizing the performance and effectiveness of the proposed approach. Finally, Section 6 concludes the paper with a summary of findings and a discussion on potential avenues for future research.

2. Related works

Keyword spotting (KWS) has seen significant advancements with deep learning models outperforming traditional methods in both accuracy and efficiency. Researchers have explored various architectures, each with trade-offs between performance and computational cost. Below, we review the main approaches and their relevance to real-time, microcontroller-based KWS systems.

Convolutional neural networks (CNNs) have long been the backbone of KWS due to their ability to extract robust features from audio spectrograms. Among these, DS-CNN [1] introduced depthwise separable convolutions, significantly reducing parameter count and computational overhead while maintaining strong performance on the Google Speech Commands dataset. Building on this idea, BC-ResNet [2] employed broadcasted residual connections to further enhance efficiency. While these models achieve competitive accuracy, they remain relatively expensive in terms of inference speed and memory, making them challenging to deploy on low-power microcontrollers.

Attention mechanisms have gained popularity for improving feature selection in neural networks. De Andrade et al. [3] introduced attention-based architectures to enhance keyword classification. Similarly, Berg et al. [4] developed Keyword Transformer (KWT), a model leveraging full self-attention, outperforming CNN-based architectures on benchmark datasets. Seo et al. [5] proposed Wav2KWS, which applies transfer learning from a Wav2Vec 2.0 encoder to enhance speech command recognition. While these transformer-based approaches achieve state-of-the-art accuracy, they are often computationally demanding, making them less practical for embedded systems.

Beyond CNNs and transformers, researchers have explored alternative architectures to optimize KWS models. ConvMixer [6] integrates token mixing to improve feature interactions while maintaining compact model size. Vygon and Mikhaylovskiy [7] introduced triplet loss-based embeddings, combining metric learning with kNN classifiers to refine keyword classification. These methods offer innovative ways to enhance recognition accuracy, but their computational overhead still limits deployment on microcontrollers.

The rise of TinyML has driven efforts to adapt deep learning models for low-power microcontrollers. Banbury et al. [8] introduced the TinyMLPerf benchmark, evaluating ultra-low-power machine learning models on accuracy, latency, and energy consumption. Rusci et al. [9] demonstrated that a quantized ResNet15 model could achieve 80% accuracy with minimal training data, requiring only 25 mW of power. Bushur et al. [10] explored various neural network architectures for KWS, optimizing them for FPGA and edge devices. Jeoung et al. [11] implemented a CNN-based KWS model on a Raspberry Pi 3, achieving 96.7% accuracy by incorporating locally recorded utterances. Abbas et al. [12] designed a TinyML-based KWS system for the Arduino Nano 33 BLE Sense, achieving 84.5% accuracy after quantization. These studies demonstrate the feasibility of running deep learning models on microcontrollers, but most still involve trade-offs between accuracy, memory consumption, and inference speed.

Several studies have benchmarked KWS models specifically for edge devices. Zhang et al. [1] optimized CNN architectures for microcontrollers, focusing on parameter efficiency. Chollet [13] introduced depthwise separable convolutions, now widely used in lightweight neural networks, including KWS models. Howard et al. [14] further refined efficient architectures through the development of MobileNetV3, emphasizing trade-offs between accuracy and efficiency. These studies highlight the ongoing need for compact, high-performance models tailored to real-time applications.

While previous research has made significant progress in optimizing KWS models, there remains a gap in achieving both high accuracy and real-time performance on embedded systems. Our work bridges this gap by integrating inter-domain

interaction and channel attention, allowing the model to extract key spectral and temporal features without excessive computational cost. Unlike transformer-based approaches, which require high memory bandwidth, our model is optimized for low-latency inference and efficient memory usage, making it well-suited for real-world applications such as voice-controlled robotics.

3. Data Preparation

The data used to train is Google Commands version 2 [15] which is a collection of spoken word recordings created to aid in the training and assessment of keyword detection systems. The main aim is to facilitate the development and evaluation of compact models capable of accurately identifying specific words from a predefined set of ten while minimizing erroneous detection caused by background noise or irrelevant speech.

The specified target labels include "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", and "Go" (10 labels). In alignment with the Tensorflow speech command dataset, two supplementary labels, "Unknown" and "Silence", are introduced. The "Unknown" category encompasses words such as "Bed", "Bird", "Cat", "Dog", "Happy", "House", "Marvin", "Sheila", "Tree", and "Wow", which exhibit phonetic resemblance to the target words and serve as stringent tests for the model's discriminatory capabilities. The "Silence" class also comprises samples extracted from background noise files, each comprising 16,000 features. Subsequently, the raw data transforms into the Mel-frequency cepstral coefficients (MFCCs) signal as Fig. 2 which are coefficients that collectively make up an MFC[16].

The time-series audio data undergo preprocessing to derive Mel-Frequency Cepstral Coefficients (MFCCs), renowned as pivotal features for subsequent analytical endeavors such as speech recognition or audio classification. The initial phase entails the computation of the Short Time Fourier Transform (STFT) to effectuate the conversion of the audio signal into its frequency domain manifestation as Eq. 1 where $x(t)$ represents the input time-series audio signal and $X(t, f)$ represents the Short Time Fourier Transform at time t and frequency f. It divides the signal into short frames, applies a window function (in this case, the Hann window), and then computes the Fourier transform for each frame.

$$X(t, f) = \text{STFT}(x(t)) \tag{1}$$

After this transformation, Mel-scaled spectrograms are derived by applying a linear-to-Mel frequency conversion with the Eq. 2 where $S(t, f)$ denotes the spectrogram derived from the magnitude of the STFT. Following this, values are stabilized by applying the natural logarithm to the Mel-scaled spectrograms denoted as $M(t, m)$. Here, $W(f, m)$ represents the linear-to-Mel weight matrix, thus resulting in log-magnitude Mel-scale spectrograms $L(t, m)$, where ϵ is a small constant added to avoid taking the logarithm of zero as Eq. 4.

$$S(t, f) = |X(t, f)| \tag{2}$$

$$M(t, m) = \sum_{f=0}^{F} S(t, f) \cdot W(f, m) \tag{3}$$

$$L(t, m) = \log(M(t, m) + \varepsilon) \tag{4}$$

Subsequently, MFCCs are computed through a succession of mathematical operations utilizing these spectrograms, thereby encapsulating crucial spectral attributes of the audio signal. The resultant MFCCs are then reshaped to adhere to the desired input format for subsequent analyses. This step is illustrated in Eq. 5,

$$\text{MFCCs}(t, c) = \text{MFCCs}(L(t, m))[: C] \tag{5}$$

where c denotes the index of an individual MFCC coefficient at time frame t, and C represents the total number of coefficients retained from the extraction process.

Adhering to defined parameters for benchmark results [8], the value of spectrogram length is set as 49, window strides is 20, and the number of DCT coefficients C with the values of 10. Subsequently, the dataset is partitioned into training and testing sets at an 80-20% ratio. Within the training process, 20% of the training set is allocated for model validation. It is evident that the number of instances labeled as "Unknown" significantly surpasses the others. In contrast, the count of "Silence" files is notably lower compared to the other categories, displaying an inverse trend.

4. Proposed methods

4.1. Model Architecture

Drawing inspiration from the ConvMixer block introduced in [6] and the concept of channel attention outlined in [14], the procedure is divided into two distinct sub-modules: one targeting the coefficient domain and the other focusing on the temporal domain. The coefficients capture various aspects of the spectral characteristics of the signal, such as the distribution of energy across different frequency bands. Typically, the first coefficient represents the overall signal energy, while the subsequent coefficients capture more detailed spectral information. The coefficient axis thus represents the extracted features or spectral attributes of the signal. Therefore, we apply *channel attention* to explore its meaningful feature along the channel axis. The time axis corresponds to different frames of the audio signal. These frames are typically overlapping segments of the

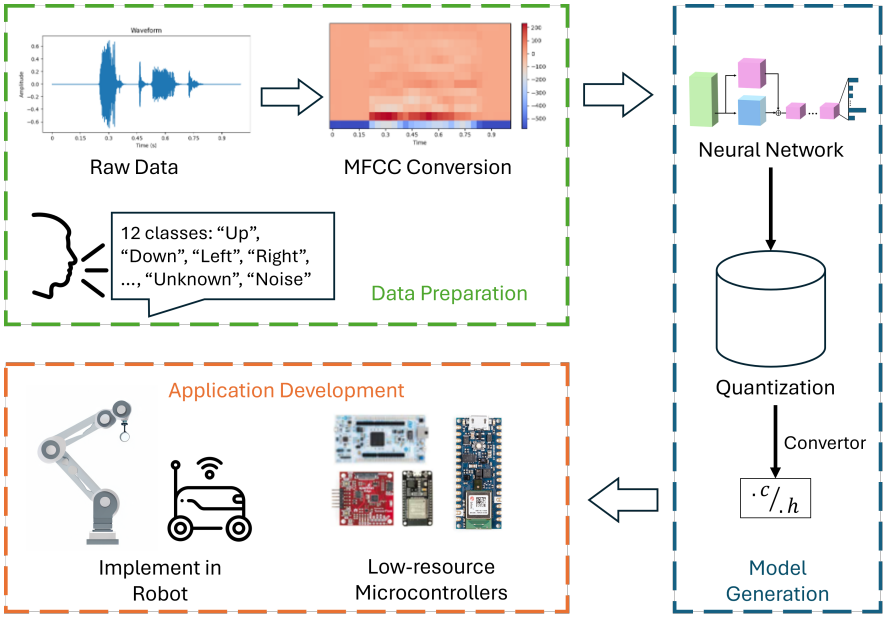

Figure 1. End-to-End Workflow for Keyword Spotting for Robotics Application.

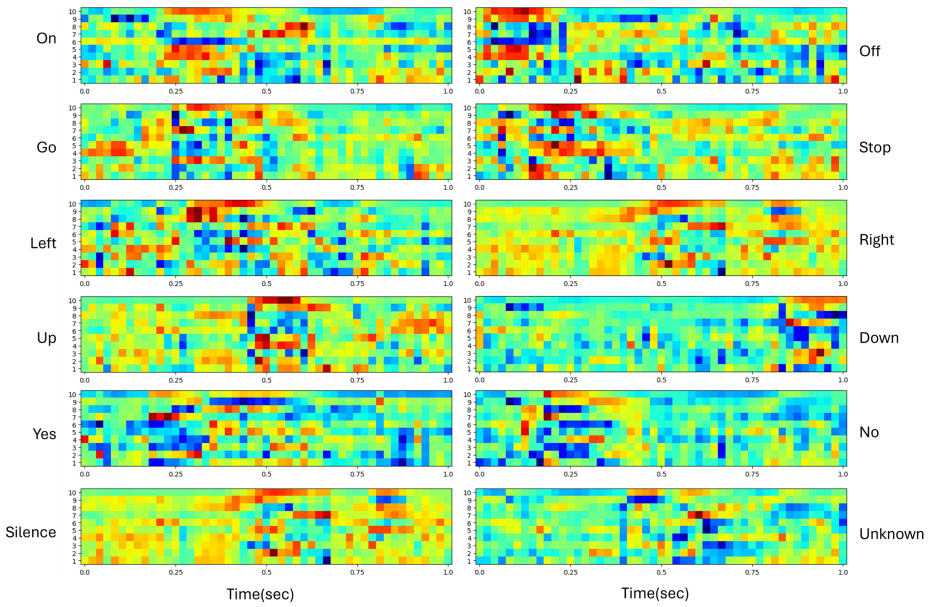

Figure 2. MFCC representations of sample audio inputs for each keyword class in the Google Speech Commands v2-12 dataset.

input signal, each capturing a short-duration snippet of the audio waveform. The frame axis thus represents the temporal evolution of the signal, allowing analysis over time. The skip connections [17] are used along with inter-domain blocks to address the vanishing gradient problem and facilitate the training of deep architectures. Additionally, residual connections encourage feature reuse and enable the network to learn residual functions, simplifying the learning task and potentially improving generalization performance. In our block, there are many sublayers inside, by providing shortcut paths for gradient flow, they alleviate the issue of gradients becoming too small as they propagate through numerous layers. This allows for more efficient training of deep networks by enabling easier optimization and faster convergence.

Depthwise Separable Convolution. Depthwise separable convolutions [13] aims at increasing demand for efficient neural network architectures, particularly in

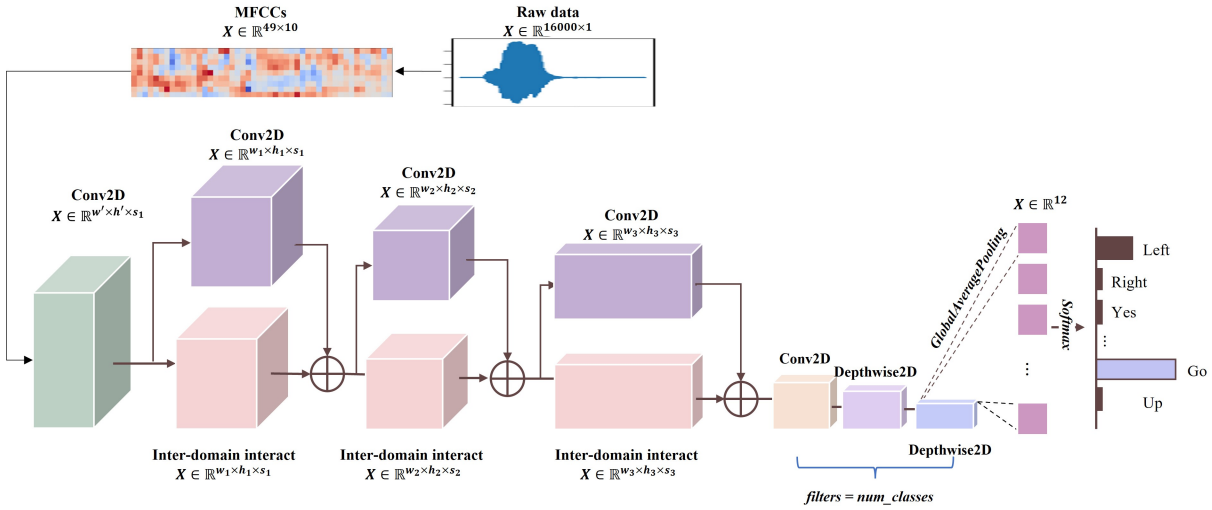

Figure 3. The proposed KWS model architecture, incorporating three inter-domain interaction layers with skip connections. Each block integrates spectral and temporal feature extraction through depthwise convolutions and attention mechanisms to enhance keyword recognition.

mobile and embedded vision applications where computational resources are limited. By decoupling traditional convolutions' spatial and channel-wise operations into separate depthwise and pointwise convolution layers, depthwise separable convolutions significantly reduce the number of parameters and computational costs while maintaining model performance.

The depthwise convolution applies a single filter per input channel, performing separate convolutions for each input channel. This operation reduces the number of parameters and computations compared to standard convolutions. For instance, with a normal convolutional neural network, to get output with shape (H', W', C) from input (H, W, C) (keeping the same dimension as the input), C kernels of size $k \times k \times C$ must be applied on the convolution process including $C \times k \times k \times C$ number of parameters. However, the depthwise convolution only requires $k \times k \times C$ parameters.

Each input filter is convolved with each kernel of the depthwise convolution as in Eq. 6.

$$\text{Out}_{k,l,m} = \sum_{i,j} \text{Kernel}_{i,k,m} \cdot \text{Filter}_{k+i-1,l+j-1,m} \qquad (6)$$

where k and l denote the spatial positions of the output feature map, m represents the channel index, i and j are indices iterating over the kernel dimensions, $\text{Kernel}_{i,k,m}$ is the weight of the convolution kernel at position (i, k, m), and $\text{Filter}_{k+i-1,\,l+j-1,\,m}$ is the corresponding value from the input feature map.

The output is then linearly combined with a 1×1 convolution to adjust the number of output filters.

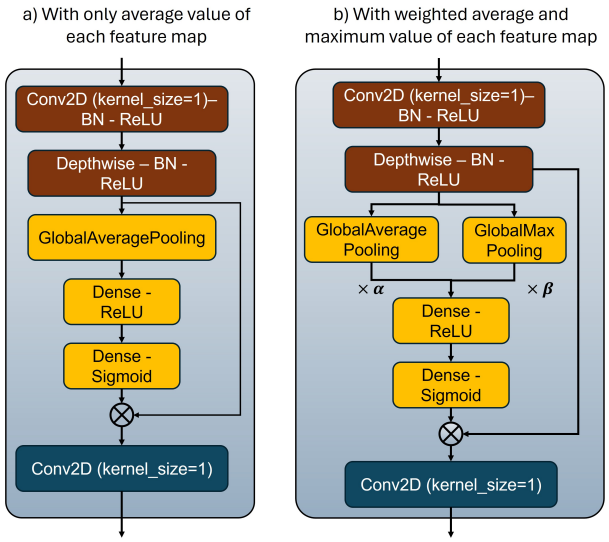

Figure 4. Visualization of the channel attention mechanism for extracting meaningful spectral features. (a) Attention weights based on average feature values. (b) Attention weights using a combination of average and maximum values for improved feature selection.

Channel attention. In this study, we employ channel attention rather than self-attention or full spatial attention due to its computational efficiency, making it well-suited for microcontroller deployment. Self-attention mechanisms, such as those used in Transformer-based models, require global dependencies and involve significant matrix multiplications, leading to increased computational cost and memory usage. In contrast, channel attention focuses on selecting the most informative

Table 1. Detailed configuration of an inter-domain interact block

Module name	Layers	Input shape	Filters	Kernel size	Strides	Output shape
Shortcut	Conv2D BatchNorm ReLU	$w \times h \times c$	s	3×3	-	$w' \times h' \times s$
Coef	Conv2D BatchNorm ReLU	$w \times h \times c$	$c/4$	1×1	1	$w \times h \times c/4$
	Depthwise2D BatchNorm ReLU	$w \times h \times c/4$	-	3×3	-	$w' \times h' \times c/4$
	Global AveragePooling	$w \times h \times c/4$	-	-	-	$c/4$
	Dense ReLU	$c/4$	-	-	-	$c/16$
	Dense ReLU	$c/16$	-	-	-	$c/4$
	Multiply	$w' \times h' \times c/4$ $c/4$	-	-	-	$w' \times h' \times 1$
Temp	Depthwise2D BatchNorm ReLU	$w' \times h' \times 1$	-	$h \times 3$	1	$w' \times h' \times 1$
	Conv2D BatchNorm ReLU	$w \times h \times 1$	s	1×1	-	$w' \times h' \times s$
Interact	Add (coef, time)	$w' \times h' \times s$ $w' \times h' \times s$	-	-	-	$w' \times h' \times s$
Residual	Add (interact, shortcut)	$w' \times h' \times s$ $w' \times h' \times s$	-	-	-	$w' \times h' \times s$

frequency bands, refining feature maps without introducing excessive overhead. This makes it particularly effective for lightweight keyword spotting (KWS) tasks, where maintaining low latency is critical for real-time applications.

The input, denoted as $\mathbf{X} \in \mathbb{R}^{w \times h \times 1}$, traverses through the channel attention module aimed at emphasizing meaningful saliency maps. This process involves extraction via conventional convolutional layers and a depthwise convolutional layer in Eq. 7, which endeavors to strike a delicate balance between accuracy and parameter count. Each channel's attention is guided by assigning corresponding weights, intended to spotlight important features. Initially, representative features from each channel are isolated, typically achieved through computations such as maximum or average values; in this study, the average value of each feature map is utilized. These extracted values subsequently undergo interaction via two fully connected layers. the quantity of hidden nodes in the initial dense layer constitutes one-quarter of those present in the subsequent layer as the proposal of [14]. The resultant output $\mathbf{A}_f \in \mathbb{R}^{1 \times 1 \times f}$, scaled within the range of 0 to 1 to represent desired weights as Eq. 8, is then multiplicatively applied to the respective input channels as shown in Eq. 9 to get $\mathbf{F} \in \mathbb{R}^{w' \times h' \times f}$ where $w' < w$ and $h' < h$, f is the number of channel in the next layer, and σ is the *Sigmoid* activation function. The last convolutional layer is used to make the output shape fit with the shape in the next stage.

$$\mathbf{E}_f = \text{ReLU}(\text{depthwise}(\text{ReLU}(\text{conv}(\mathbf{X})))) \tag{7}$$

$$\mathbf{A}_f = \sigma(\text{FC}(\text{FC}(\text{GAP}(\mathbf{E})))) \tag{8}$$

$$\mathbf{F} = \mathbf{E}_f \cdot \mathbf{A}_f \tag{9}$$

Global Average Pooling (GAP) and Global Max Pooling (GMP) Nevertheless, in Eq. 8, it is evident that the weights of the channels are solely determined by the average value, potentially resulting in an equalized impact of all discriminative features. Consequently, rather than solely focusing on the average value, we advocate for considering the maximum value as well to survey the impact of both pooling methods, and the result is given in the Table. 2. In certain scenarios, the maximum value represents the most discriminative feature. Both the average and maximum values are accompanied by corresponding coefficients α and β as Fig. 4 and Eq. 11.

$$\mathbf{A}_f = \sigma(\mathrm{FC}(\mathrm{FC}(\boldsymbol{\alpha}\mathrm{GAP}(\mathbf{E}) + \boldsymbol{\beta}\mathrm{GMP}(\mathbf{E})))) \tag{10}$$

Inter-domain Interaction. To enhance the quality of the extracted feature maps, inter-domain interaction is employed to integrate spectral (MFCC) and temporal features, yielding a more robust and holistic representation.

Once spectral feature maps have been refined using the previously described channel attention mechanism, they are further processed by two sequential 1D convolution layers: a conventional convolutional layer and a depthwise convolutional layer. This step is designed to capture intricate temporal patterns, as formulated in Eq. 11.

$$\mathbf{T} = \mathrm{ReLU}(\mathrm{conv}(\mathrm{ReLU}(\mathrm{depthwise}(\mathbf{F})))) \tag{11}$$

Following this, the extracted temporal features from Eq. 11 are seamlessly fused with the spectral features from Eq. 9, forming a unified representation, as demonstrated in Eq. 12.

$$\mathbf{M} = \mathbf{F} + \mathbf{T} \tag{12}$$

Additionally, a skip connection consisting solely of a Conv2D layer is incorporated to mitigate vanishing gradients and facilitate the efficient learning of intricate relationships between the two domains.

The integration of complementary features not only enriches the overall representation of the signal but also strengthens robustness against environmental noise and speaker variability, thereby enhancing the model's ability to accurately distinguish between similar sounds.

4.2. Implementation Details

We trained the model shown in Fig. 3 with the values of *s1*, *s2*, and *s3* set to 16, 32, and 64 for each block, respectively, as detailed in Table 1. The training was conducted for 50 epochs with a batch size of 128 samples per iteration, using the Adam optimizer initialized with a learning rate of 0.001, which was reduced 10 times when validation loss stopped improving.

Furthermore, in our study, we have also incorporated several model architectures proposed by [1], which have been optimized to accommodate the input shape of 40×10, as outlined in the benchmark paper by [8]. To ensure a fair and meaningful comparison, we retrained these models from scratch on our dataset, rather than directly using the results reported in previous studies. This avoids any potential misunderstanding that the reported results are taken from other papers. Additionally, we implemented another neural network to highlight the performance of our proposed block when compared with other commonly used architectures.

The DenseNet network comprises three dense blocks, each consisting of batch normalization, ReLU activation, and 3×3 convolutional layers. Within each dense block, feature maps from preceding layers are concatenated with new feature maps generated by the composite functions, enabling feature reuse and improved feature propagation. Transition layers between dense blocks reduce the number of feature maps and control model complexity. The network concludes with fully connected layers followed by a Softmax activation function to generate class probabilities.

All models were trained on a High-Performance Computer with $2 \times A100$ GPU and 128G RAM. For deployment on the Arduino Nano 33 BLE Sense, we optimized the model using post-training quantization, reducing memory consumption while maintaining accuracy. The microcontroller's Cortex-M4F processor operates at 64 MHz, with 256 KB RAM and 1 MB Flash, limiting model size. To ensure efficient inference, we incorporated lightweight activation functions, including ReLU6 and Hard Swish, and minimized the number of MAC operations per forward pass by utilizing depthwise convolutions.

In summary, our methodology utlines a comprehensive approach combining inter-domain feature extraction with efficient attention mechanisms tailored for resource-constrained environments. With the model architecture and training protocols rigorously defined, we now turn to evaluating its performance. The following section presents a detailed analysis of experimental results, including accuracy, inference speed, memory efficiency on the Google Command v2-12 dataset, and some ablation studies conducted to observe the performance of the model with different types of architectures.

5. Results and Discussion

The performance metrics of the model are illustrated in Fig. 7 (right). Within the confusion matrix, the accuracy

of each class is depicted through comparisons of actual versus predicted labels. Most classes achieve accuracies exceeding 90%, except for the Left and Stop classes, which attain accuracies of 87% and 89%, respectively. Notably, all classes tend to be misclassified as class Unknown. This outcome is expected, considering the content discussed in Section 3, where the unknown class encompasses numerous words that bear close phonetic resemblance to the target labels.

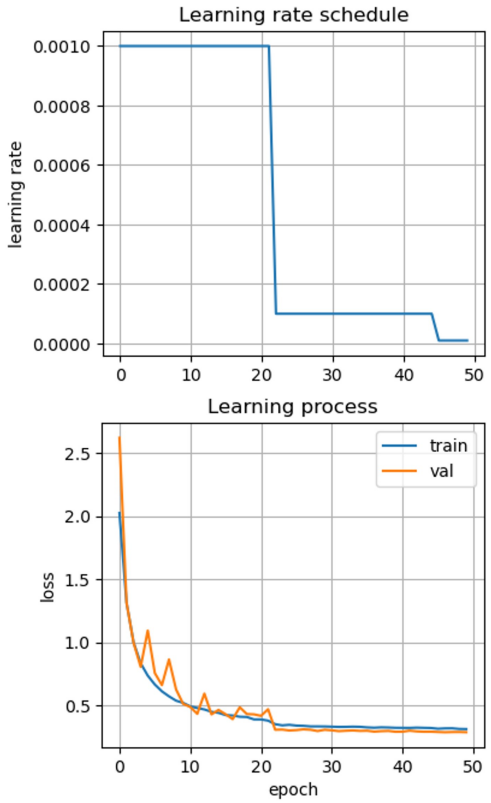

Figure 5. Learning process of the best model with GAP only which converges quickly after 10 epochs

Table 2. Comparative analysis of the effects of Global Max Pooling on model performance where α and β are the coefficients of GAP and GMP, respectively

α	β	Accuracy (valid)	Accuracy (test)	F1 (%)	Recall (%)
0.5	0.5	92.90	93.61	93.60	93.61
0.8	0.2	91.45	92.06	91.75	92.06
0.2	0.8	93.09	**93.70**	**93.67**	**93.70**
1.0	0.0	**93.52**	93.68	93.67	93.68
0.0	1.0	90.36	91.07	91.04	91.07

Fig. 5 illustrates the learning process of the trained model. The model demonstrates proficient learning with each successive epoch. Notably, both the training and validation processes exhibit pronounced convergence following ten epochs, subsequently maintaining stability beyond the twentieth epoch. Despite a decrease in the learning rate, significant improvement in results is unattainable. Remarkably, there is an absence of overfitting throughout the training process.

Table. 2 presents the performance metrics of the model employing Global Average Pooling (GAP), Global Max Pooling (GMP), and a combination of both pooling methods. Either GAP or GMP yields results that are largely comparable to each other. The disparities between the two methodologies are not deemed significant. Table. 3 shows the performance of the models in [1], which is implemented to make them fit input shape of 40×10 as in benchmark paper.[8].

The discrepancy between the raw data and their corresponding extracted features at the final layer of the proposed model, employing Principal Component Analysis (PCA) to project all data points into a lower-dimensional space, is depicted in Fig. 6. The data processed by the model exhibit well-defined clustering and demonstrate a noticeable trend in classification.

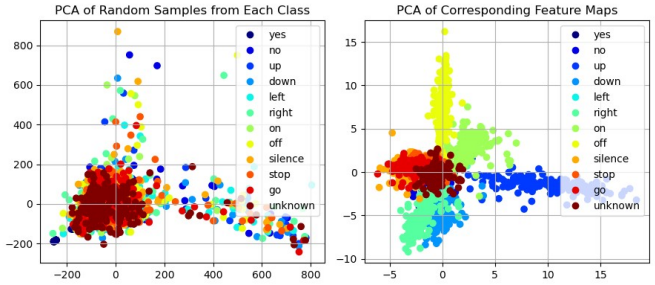

Figure 6. Visualization of raw data samples (left) and feature embeddings from the final model layer (right) using PCA. The improved clustering in the right plot demonstrates the model's ability to learn discriminative features for keyword classification.

The inclusion of channel attention significantly enhances keyword detection by selectively amplifying important spectral features. Without attention, the model treats all frequency components equally, potentially diluting the relevance of critical information. Our results show that integrating channel attention leads to a higher classification accuracy (93.70%) while maintaining a compact model size. This suggests that attention mechanisms, even in a lightweight form, can improve feature discrimination and robustness in KWS tasks. While more complex attention mechanisms, such as self-attention, could further enhance performance, they come at the cost of higher computational requirements, making them less practical for microcontroller-based deployments.

The results in Table 4 and Table 5 highlight the efficiency of the proposed model across different microcontroller platforms. On the Arduino Nano 33 BLE Sense,

Table 3. Comparison of our model with others on the dataset

Model	Params (K)	FLOPS (M)	Acc (%)	F1 (%)	Recall (%)
DS-CNN [8]	38.6	24.9	87.60	86.83	85.92
DenseNet [8]	**12.8**	**0.5**	85.1	82.5	82.83
DS-CNN-S[1]	23.6	5.4	87.60	86.83	85.92
DS-CNN-M[1]	137.8	19.8	86.40	86.0	84.5
DS-CNN-L[1]	415.9	56.9	86.70	84.4	84.6
Ours	35.1	5.5	**93.70**	**93.67**	**93.70**

Table 4. Model Performance on Inference on Arduino Nano 33 BLE Sense

Model	Inference time(s)	Peak RAM (KB)	Flash usage(KB)
DS-CNN-S[1]	1.135	65.2	114.0
DS-CNN-M[1]	2.061	81.0	199.2
DS-CNN-L[1]	2.855	137.6	496.4
Ours	**0.359**	**34.9**	**98.7**

Table 5. Model Performance on Inference on STM32 MCUs

Model	Inference time(ms)	Peak RAM (KB)	Flash usage(KB)
DS-CNN-S[1]	13.63	**34.34**	**98.2**
DS-CNN-M[1]	23.66	69.15	195.92
DS-CNN-L[1]	59.78	115.05	484.2
Ours	**11.37**	39.46	120.97

our model achieves an inference time of 0.359 seconds, which is three times faster than DS-CNN-S (1.135s) and eight times faster than DS-CNN-L (2.855s), while also reducing peak RAM usage by 47% and flash memory by 80% compared to DS-CNN-L. Similarly, on the STM32 Nucleo-H743ZI2 (Cortex-M7, 480 MHz), our model achieves an inference time of 11.37 ms, outperforming DS-CNN-S (13.63 ms) and DS-CNN-L (59.78 ms) by 16.5% and over five times, respectively. Although our model requires slightly more RAM (39.46 KB vs. 34.34 KB for DS-CNN-S on STM32), it remains significantly more efficient in terms of both inference speed and flash memory usage. These results confirm that our model is well-suited for real-time, resource-constrained applications, demonstrating strong adaptability across different microcontroller architectures while maintaining a balance between speed, memory efficiency, and computational complexity.

The proposed model in Sec. 4 was tried with some different architectures with 1, 2, and 3 interact layers, and the results are shown in Fig. 7 and the detailed performance in Table. 6.

Furthermore, we conducted experiments to evaluate the model's performance under different configurations, specifically by selectively retaining

Table 6. Comparison of our model performance applied with different number of interact layers

Model	Params (K)	Acc (%)	F1 (%)	Recall (%)
1	5	85.38	85.38	85.03
2	33	92.22	92.22	92.21
3	**35**	**93.68**	**93.68**	**93.68**

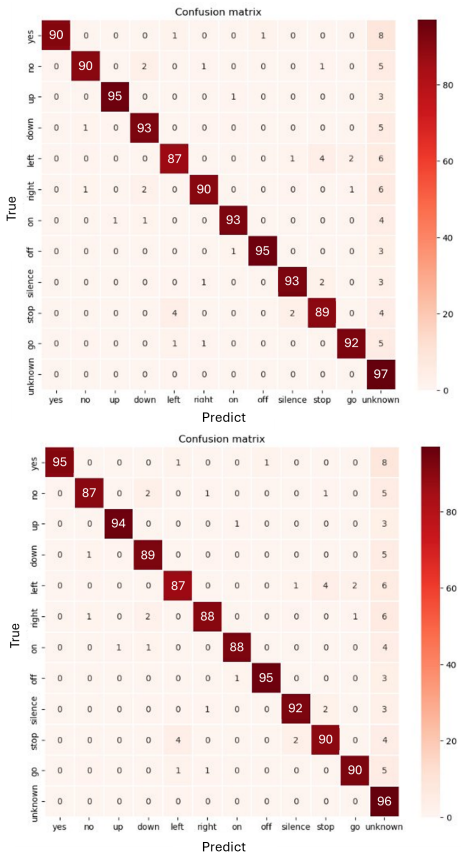

Figure 7. Confusion matrix of the best model without GAP including 2 (top) and 3 (bottom) inter-domain interaction layers

certain CNN layers while preserving skip connections and removing inter-domain interaction modules. This adapted model achieved an accuracy of 88.13% with a corresponding loss of 0.42. In contrast, our proposed model achieved a significantly lower loss of approximately 0.22, nearly half that of the model

utilizing only CNN layers. Additionally, we explored alternative configurations in the final layer, replacing Global Average Pooling with Global Max Pooling and integrating a Fully Connected Layer. The outcomes of these variations are detailed in Table. 7.

6. Conclusion

This paper introduced a novel lightweight deep learning framework for keyword spotting (KWS), combining input-domain knowledge with attention mechanisms to enhance feature extraction and model focus. The exploration of distinct pooling methods further refined channel weighting, bolstering the framework's capacity for effective classification. Empirical evaluations demonstrated the model's efficacy, achieving a classification accuracy of 93.70% on a 12-label dataset—outperforming the tinyMLPerf benchmark model by 6.1% and other models like DS-CNN-S, M, and L by 6-8%. The proposed framework also exhibited significant improvements in efficiency, with an inference time of 0.359 seconds (three times faster than DS-CNN-S) and peak RAM and flash usage of 34.9KB and 98.7KB, respectively, representing a 50% reduction compared to the smallest DS-CNN model. These results highlight the framework's suitability for resource-constrained environments, particularly in robotics, enabling intuitive voice-controlled operations such as movement and task execution. Future work will focus on optimizing the model's efficiency and performance through techniques like quantization and pruning. Additionally, we aim to develop an end-to-end framework that eliminates reliance on preprocessing or hand-crafted features, broadening its applicability beyond KWS to other domains while maintaining compatibility with resource-limited devices.

References

[1] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[2] B. Kim, S. Chang, J. Lee, and D. Sung, *Broadcasted residual learning for efficient keyword spotting*, 2023. arXiv: 2106.04140 [cs.SD]. [Online]. Available: https://arxiv.org/abs/2106.04140.

[3] D. C. De Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," *arXiv preprint arXiv:1808.08929*, 2018.

[4] A. Berg, M. O'Connor, and M. T. Cruz, "Keyword transformer: A self-attention model for keyword spotting," *arXiv preprint arXiv:2104.00769*, 2021.

[5] D. Seo, H.-S. Oh, and Y. Jung, "Wav2kws: Transfer learning from speech representations for keyword spotting," *IEEE Access*, vol. 9, pp. 80 682–80 691, 2021.

[6] D. Ng, Y. Chen, B. Tian, Q. Fu, and E. S. Chng, "Convmixer: Feature interactive convolution with curriculum learning for small footprint and noisy far-field keyword spotting," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2022, pp. 3603–3607.

[7] R. Vygon and N. Mikhaylovskiy, "Learning efficient representations for keyword spotting with triplet loss," in *Speech and Computer: 23rd International Conference, SPECOM 2021, St. Petersburg, Russia, September 27–30, 2021, Proceedings 23*, Springer, 2021, pp. 773–785.

[8] C. Banbury, V. J. Reddi, P. Torelli, *et al.*, "Mlperf tiny benchmark," *arXiv preprint arXiv:2106.07597*, 2021.

[9] M. Rusci and T. Tuytelaars, "On-device customization of tiny deep learning models for keyword spotting with few examples," *Ieee Micro*, 2023.

[10] J. Bushur and C. Chen, "Neural network exploration for keyword spotting on edge devices," *Future Internet*, vol. 15, no. 6, p. 219, 2023.

[11] J. E. Jeoung, Y. K. Yeow, and M. Ahemad, "Keyword spotting on embedded system with deep learning," in *Proceedings of 2019 electrical engineering symposium*, vol. 3, 2019, pp. 87–91.

[12] N. A. Abbas and M. R. Ahmad, "Keyword spotting system with nano 33 ble sense using embedded machine learning approach," *Jurnal Teknologi (Sciences & Engineering)*, vol. 85, no. 3, pp. 175–182, 2023.

[13] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[14] A. Howard, M. Sandler, G. Chu, *et al.*, *Searching for mobilenetv3*, 2019. arXiv: 1905.02244 [cs.CV].

[15] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[16] M. Xu, L.-Y. Duan, J. Cai, L.-T. Chia, C. Xu, and Q. Tian, "Hmm-based audio keyword generation," in *Pacific-Rim Conference on Multimedia*, Springer, 2004, pp. 566–574.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[18] S. Majumdar and B. Ginsburg, "Matchboxnet: 1d time-channel separable convolutional neural network architecture for speech commands recognition," *arXiv preprint arXiv:2004.08531*, 2020.

[19] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted residual learning for efficient keyword spotting," *arXiv preprint arXiv:2106.04140*, 2021.

[20] D. Seo, H.-S. Oh, and Y. Jung, "Wav2kws: Transfer learning from speech representations for keyword spotting," *IEEE Access*, vol. PP, pp. 1–1, May 2021. DOI: 10.1109/ACCESS.2021.3078715.

Table 7. Comparison of our model performance applied with some modification in the last layer with the replacement of Global Average Pooling with Global Max Pooling or Dense layer

Model	Params (K)	Loss (smaller is better)	Accuracy (larger is better)
Without interaction + GAP	35.09	0.42	88.13
Interaction + GAP	35.09	**0.22**	**93.68**
Interaction + GMP	35.09	0.47	86.79
Interaction + Dense	35.82	0.28	91.02

[21] M. N. Miah and G. Wang, "Keyword spotting with deep neural network on edge devices," in *2022 IEEE 12th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, IEEE, 2022, pp. 98–102.

[22] M. Bezoui, A. Elmoutaouakkil, and A. Beni-hssane, "Feature extraction of some quranic recitation using mel-frequency cepstral coeficients (mfcc)," in *2016 5th international conference on multimedia computing and systems (ICMCS)*, IEEE, 2016, pp. 127–131.

[23] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques," *arXiv preprint arXiv:1003.4083*, 2010.

[24] *Edge Impulse*, https://edgeimpulse.com/, Accessed: 2024.