On-Device Customization of Tiny Deep Learning Models for Keyword Spotting With Few Examples

Manuele Rusci and Tinne Tuytelaars, *KU Leuven, 3000, Leuven, Belgium*

Designing a customized keyword spotting (KWS) deep neural network (DNN) for tiny sensors is a time-consuming process, demanding training a new model on a remote server with a dataset of collected keywords. This article investigates the effectiveness of a DNN-based KWS classifier that can be initialized on-device simply by recording a few examples of the target commands. At runtime, the classifier computes the distance between the DNN output and the prototypes of the recorded keywords. By experimenting with multiple tiny machine learning models on the Google Speech Command dataset, we report an accuracy of up to 80% using only 10 examples of utterances not seen during training. When deployed on a multicore microcontroller with a power envelope of 25 mW, the most accurate ResNet15 model takes 9.7 ms to process a 1-s speech frame, demonstrating the feasibility of on-device KWS customization for tiny devices without requiring any backpropagation-based transfer learning.

A voice command sensor placed on everyday objects can recognize a set of target keywords to enable speech-controlled functionalities. The keyword classification algorithm, commonly denoted as keyword spotting (KWS) in the literature,[1] runs locally on-device to process the audio data recorded by the microphone. These smart sensors are typically battery powered, and microcontroller units (MCUs) are used as data processing engines to meet the stringent energy requirements. MCUs feature a power consumption of up to a few tens of milliwatts but, on the other side, present limited computation power and on-chip memory capacity, making the porting of robust speech processing pipelines on-device highly challenging.

Recently, deep neural networks (DNNs) for KWS have been efficiently implemented on low-power MCUs.[2] These DNN solutions feature up to only a few hundred thousand parameters to fit the memory constraints of tiny MCU devices, typically lower than a few megabytes. Small-sized solutions, which belong to the recently born tiny machine learning (TinyML) domain (which deals with machine learning approaches for tiny and resource-constrained devices), are also applied for wake word detection—e.g., "Hey Snips!"[3]—where the multiclass keyword classification turns into a simpler binary problem. At present, however, the state-of-the-art TinyML KWS design approaches demand a time-consuming data collection and labeling procedure to train application-specific DNN models using high-performance servers, which represents a bottleneck for rapid prototyping in new environments and target scenarios.

To address this limitation, this article explores whether DNN-based KWS classifiers tailored for TinyML devices can achieve simple and fast on-device customization of the target keywords using only few (shot) recorded examples. The overview of the adopted approach is depicted in Figure 1. First, a DNN-based feature extractor is trained on a large labeled dataset, which is disjoined from the target environment, using the triplet loss that has been recently demonstrated to be effective for learning a robust representation for KWS systems.[4] The DNN model is fed by temporal–frequency (TF) maps extracted from a speech segment of fixed length and outputs an embedding vector in a low-dimensional space, e.g., 64–256 feature values. The learning process aims at reducing the distance between the embedding vectors belonging to the

Digital Object Identifier 10.1109/MM.2023.3311826
Date of publication 6 September 2023; date of current version 2 November 2023.

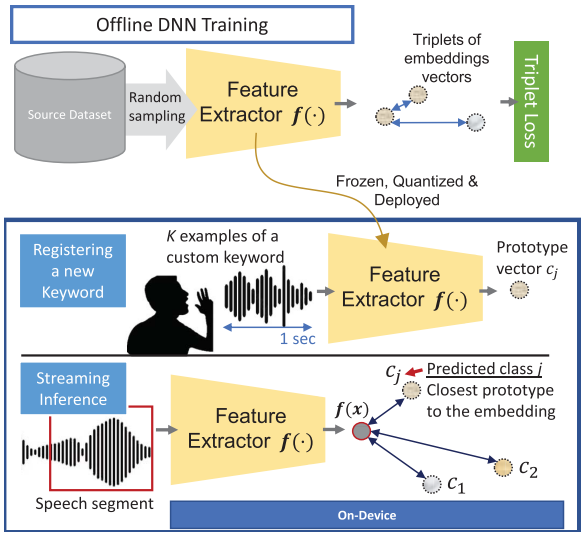


FIGURE 1. Keyword spotting (KWS) customization using few-shot utterances. After training a DNN feature extractor, the class prototypes are computed on-device and used for classification.

same class; conversely, embeddings from different categories are kept distant in the feature space. After quantizing and deploying the KWS feature extractor on an MCU-based sensor system, a classifier of voice commands is initialized simply by recording one or more examples for every user-defined speech command. At inference time, KWS classification is based on the distance between the output feature vector of the current sample and the class prototypes.

This article makes the following contributions:

- We analyze the effectiveness of DNN feature extractors trained with the triplet loss for the customization of a set of KWS models and discuss the insights of the learned representation.
- We describe a design methodology to realize an on-device, customized KWS sensor device powered by a tiny MCU engine.
- We analyze and compare the accuracy scores of multiple KWS models and their latency, memory, and energy costs on a state-of-the-art MCU chip.

Our approach is tested on the public Google Speech Command (GSC) dataset considering multiple tiny DNN backbones as feature extractors. We run experiments using few-shot enrollments from the target scenario and assess the quality of detecting custom keywords while being robust to false positives. When using only 10 examples, we show that a top accuracy of up to 80% at a false acceptance rate (FAR) of 5% can be achieved with a ResNet15 model, +3% and +11% higher than, respectively, large and small depthwise-separable convolutional neural network (DS-CNN) models. We observe these scores to be lossless with respect to 8-bit quantization. On the best-in-class GAP9 MCU, which features a compute cluster with nine RISC-V cores and a hardware accelerator for convolution operations running in a power envelope of 25 mW,[7] the most accurate ResNet15 model takes only 9.7 ms to process 1 s of streaming audio. Overall, our solution is the first to provide on-device customization of the target keyword in the TinyML domain without requiring any backpropagation-based learning of the classifier using data from the target scenario.

OUR APPROACH IS TESTED ON THE PUBLIC GOOGLE SPEECH COMMAND (GSC) DATASET CONSIDERING MULTIPLE TINY DNN BACKBONES AS FEATURE EXTRACTORS.

DNN FEATURE EXTRACTOR DESIGN

Let $f(\cdot)$ be a DNN feature extractor that operates on speech utterances of 1 s with a sampling frequency of 16 kHz. The audio signal is preprocessed by computing the first 10 mel frequency cepstral coefficients (MFCC) features. We use a frame size of 40 ms and a stride of 20 ms; Hamming windowing is applied. As a result, a TF map of size 49×10 and denoted as x^S is fed to the DNN model.

The feature extractor is trained on a source dataset disjoined from the target scenario. We employ the recent Multilingual Spoken Words Corpus (MSWC) dataset[6], which includes up to 39,000 unique utterances in the English train partition. The average number of occurrences per class is 180; the class distribution is highly unbalanced. In our experiments, we only consider speech samples from the 500 categories with the highest number of occurrences, for a total amount of 2.7 million samples (~5470 utterances per class on average). We adopt the triplet loss L_{TR} as the optimization function to learn the parameters of the feature extractor, as proposed in a recent work.[4] The training process is iterated for multiple epochs structured in episodes. At every training episode, the data loader samples a batch of Q random samples from M randomly chosen classes (from the 500 available) for a total of $Q \times M$ utterances. After the MFCC

TABLE 1. Deep neural network feature extractor for tiny machine learning keyword spotting.

Model	Parameters	MMAC	Maximum Activation Size	Emb Size
TCResNet8	61,240	3.0	23,184	48
DS-CNN-S	21,824	2.6	24,991	64
DS-CNN-M	132,956	9.4	52,116	172
DS-CNN-L	407,376	28.0	112,332	276
ResNet15	482,368	235.1	537,344	64

DS-CNN: depthwise-separable convolutional neural network; Emb: embedding vectors; MMAC: millions of multiply-and-accumulate operations; TC: temporal convolution.

preprocessing, the batch of TF input data, denoted as $\{x^s_{i,c}\}, i = 1,..Q, c = 1,...M$, is fed to the model f to compute the embedding vectors $z^s_{i,c} = f(x^s_{i,c})$. For every batch, a set of N_{tr} triplets is drawn by associating to every couple of samples from the same category a random point from a different category (random negative mining[4]). Given a triplet ($z^s_{i,c}$, $z^s_{j\neq i,c}$, z^s_{neg}), where z^s_{neg} is a sample of the current batch not belonging to the class c, the triplet loss is computed as

$$L_{\text{TR}} = \sum_{N_{\text{tr}}} \max(d(z^s_{i,c}, z_{j\neq i,c}) - d(z^s_{i,c}, z^s_{\text{neg}}) + m, 0). \quad (1)$$

In more detail, $d(\cdot)$ is the squared Euclidean distance, and m is the margin.

Table 1 lists the feature extractor models considered in our study. We report the number of parameters and the millions of multiply-and-accumulate (MAC) operations per inference. Also, we indicate the size of the largest activation map of the model, which impacts the requirement of read–write memory, i.e., RAM memory of the MCU. On the contrary, model parameters are fixed and can be stored in a read-only memory (e.g., on-chip flash memory). Among the others, we run experiments with DS-CNN models of different sizes.[2] These convolutional architectures replace common 2-D convolution layers, e.g., those used in ResNet15, with a combination of depthwise and pointwise layers to reduce the number of parameters and operations. On the contrary, the temporal convolution (TC) ResNet8 makes use of 1-D convolutions after reshaping the input TF data and treating the MFCC features as input channels.

Every model computes the embedding vectors from the output of the convolution-based backbone. We refer to this operation as *embedding compute* in the "Experimental Results" section. Note that ResNet15 and TCResNet8 feature a BatchNorm layer after the last convolution layer as proposed in the original work using triplet loss.[4] On the DS-CNN backbones, we place instead a layer normalization layer, i.e., LayerNorm, which was observed to lead to a good representation.[12] After the normalization, the embedding vectors are computed using a spatial average pooling layer and applying L2 normalization. The size of the final embedding vectors is reported in Table 1.

ON-DEVICE FEW-SHOT KWS

After training, the feature extractor is frozen for deployment on the MCU platform. We apply 8-bit posttraining quantization to calibrate the quantization ranges. A *prototype-based classifier* is initialized on top of the feature extractor with few examples of the target speech commands. The speech samples can be collected on-device by recording multiple audio segments of 1 s including the command. More formally, the KWS classifier setup takes K examples for the N custom classes $\{x^T_{i,j}\}, i = 1,..K, j = 1..N$, i.e., the speech keywords. For simplicity, this study considers the same number of examples for every target class even if the number of samples per class can be different.

After the audio recording, the TF map of every audio keyword is fed to the feature extractor to compute the embedding vector $z^T_{i,j}$. For every class j, a prototype vector c_j is calculated as

$$c_j = \frac{1}{K}\sum_{i=1}^{K} z^T_{i,j}, \quad j = 1...N. \quad (2)$$

In addition to the target classes, we add a prototype for the *silence* class and one for the *unknown* class. These vectors are computed by feeding, respectively, K samples of background noise and K random words to the feature extractor. We assume that none of the unknown examples include a target keyword. These extra prototypes can be computed before deployment and directly stored on-device.

At inference time, the TF map x of the current speech segment is classified by predicting the class label y_{pred} as

$$y_{\text{pred}} = \begin{cases} \text{argmin}_j\, d(f(x), c_j), & \text{if} \quad d(f(x), c_j) < \gamma \\ y_{\text{unk}}, & \text{otherwise} \end{cases} \quad (3)$$

where y_{unk} is the index of the *unknown* class, and γ is the score threshold. This γ parameter can be tuned to

TABLE 2. Accuracy for 10-keyword classification with a varying number of examples at false acceptance rates of 5% and 1%.

Model	1 Shot				3 Shots				5 Shots				10 Shots			
softmax	✓	–	✓	–	✓	–	✓	–	✓	–	✓	–	✓	–	✓	–
FAR	5%		1%		5%		1%		5%		1%		5%		1%	
TCResNet8	0.33	0.38	0.17	0.23	0.47	0.56	0.28	0.39	0.52	0.63	0.35	0.46	0.58	0.67	0.39	0.51
DS-CNN-S	0.27	0.37	0.15	0.22	0.45	0.57	0.24	0.37	0.49	0.65	0.29	0.45	0.54	0.69	0.34	0.50
DS-CNN-M	0.40	0.45	0.26	0.28	0.59	0.64	0.40	0.48	0.64	0.71	0.48	0.54	0.70	0.74	0.53	0.59
DS-CNN-L	0.46	0.47	0.32	0.33	0.66	0.70	0.47	0.51	0.70	0.75	0.55	0.58	0.74	0.77	0.59	0.64
RESNET15	0.49	**0.55**	0.37	**0.36**	0.70	**0.74**	0.57	**0.58**	0.74	**0.78**	0.62	**0.64**	0.78	**0.80**	0.68	**0.70**

The bold values refer to the best scores. FAR: false acceptance rate.

adjust the false acceptance rate (FAR) of the classifier, i.e., the ratio of unknown samples classified as a target keyword. A lower value of γ forces a more conservative behavior by predicting more samples as unknowns. Note that the optimal tuning of γ to achieve a target value of FAR can only be derived a posteriori. In this work, we simply propose setting the value of γ as the margin m during the test. We motivate this choice by noting that the loss function (1) drives the training process to guarantee a class separation distance larger than m, on average. In our study, we also investigate the impact of applying a *softmax* normalization to the distance vector $d(f(x), c_j)$ in (3), which is commonly used in other few-shot learning approaches, e.g., ProtoNet.[9]

EXPERIMENTAL RESULTS

We train the feature extraction models of Table 1 on the MSWC dataset for 40 epochs of 400 episodes using Adam as the optimizer with a learning rate of 0.001, which is divided by 10 after 20 epochs. Every episodic batch loads 80 samples from 20 classes, for a total of 1600 data per batch, from which triplets are sampled. The margin m is 0.5. To account for different initialization seeds, every training process is repeated three times, and every trained model is individually tested. The final results are averaged over the multiple runs.

For the testing phase, we use GSC, a collection of speech utterances of 1 s belonging to 35 categories. We split the dataset into a positive partition GSC+ and a negative partition GSC−. GSC+ is composed of 10 target keywords: *on*, *off*, *left*, *right*, *up*, *down*, *go*, *stop*, *yes*, and *no*. We use five categories (*backward*, *forward*, *visual*, *follow*, and *learn*) to compute the prototype of the *unknown* class. The GSC− partition includes the samples from the remaining 20 classes. We keep the original train/test split of GSC, and we use the train split to fetch the (few) samples to compute the prototypes of (2). The *silence* prototype is obtained by feeding the DNN model with background noise waveforms also belonging to the GSC dataset.

To assess the quality of a trained feature extractor model on the target GSC dataset, we measure the classification accuracy (ACC) as the ratio of correctly classified utterances of the GSC+ test set. On the other hand, we use GSC− to assess the FAR as the percentage of negative samples misclassified as positives. An ideal classifier scores ACC $=$ 100% and FAR $=$ 0%. However, in a real system, the ACC score tends to decrease when the FAR decreases, e.g., by reducing the threshold (3).

Table 2 reports the ACC scores for the considered models at FARs of 5% and 1%, obtained by opportunely tuning the threshold a posteriori. We run experiments using one, three, five, and 10 shots, optionally applying the *softmax* on the distance vector (3). Because the results are impacted by the selection of the few-shot samples to compute the prototype, we repeat every test 10 times and average the obtained statistics. Using the L2 Euclidean distance for classification without softmax systematically leads to the highest accuracy, which increases with the model capacity and the number of examples used to compute the prototype. Among the tested models, ResNet15 achieves the highest scores of 80% and 70% at FARs of, respectively, 5% and 1%. In particular, the accuracy level at FAR=5% is 3% and 11% higher than the ACC of DS-CNN-L and DS-CNN-S. Conversely, a maximum accuracy of up to 74% is reached with a low number of examples, i.e., three. To get more insights on the tradeoff between accuracy and FAR, Figure 2 visualizes the receiver operating characteristic curve of the one instance of the models initialized with 10 examples.

Figure 3 plots ACC versus FAR scores with a fixed threshold $\gamma = m = 0.5$, as proposed for the deployment.

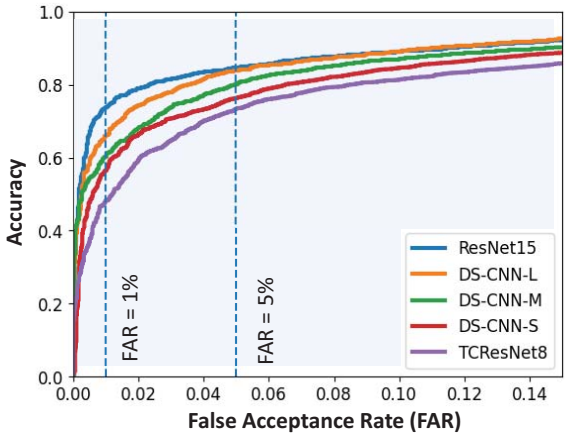


FIGURE 2. Receiver operating characteristic curves of the KWS models after 10-shot initialization. DS-CNN: depthwise-separable convolutional neural network; TC: temporal convolution.

The Pareto front (upper left corner) is populated by instances of the ResNet15 model. With a low fixed threshold, the FAR achieves a value as low as 0.1%–0.2%, and the accuracy increases up to 54% using 10 examples. Under the same settings, DS-CNN-L reaches an accuracy of 58% at an FAR of 0.6%, meaning the classifier will trigger three times more false positive alarms per hour than ResNet15. In the plot, we observe a trend where models with a lower capacity present a higher accuracy and FAR than models with a higher capacity. This is explained by the relaxed compactness of the target class clusters achieved by the big models at the cost of a superior interclass distance.

Figure 4 gives insights of the representation learned by one of the trained ResNet15 models. The test

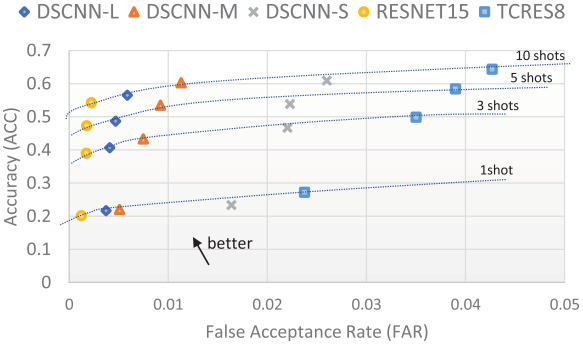
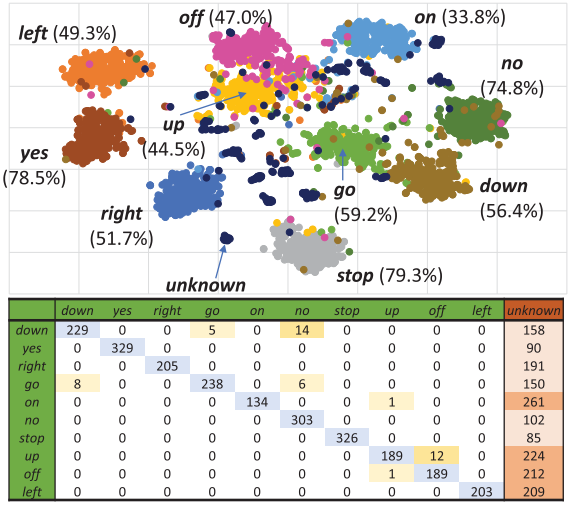


FIGURE 3. Tiny machine learning model scores in a few-shot setting with a fixed threshold $\gamma = m$.



	down	yes	right	go	on	no	stop	up	off	left	unknown
down	229	0	0	5	0	14	0	0	0	0	158
yes	0	329	0	0	0	0	0	0	0	0	90
right	0	0	205	0	0	0	0	0	0	0	191
go	8	0	0	238	0	6	0	0	0	0	150
on	0	0	0	0	134	0	0	1	0	0	261
no	0	0	0	0	0	303	0	0	0	0	102
stop	0	0	0	0	0	0	326	0	0	0	85
up	0	0	0	0	0	0	0	189	12	0	224
off	0	0	0	0	0	0	0	1	189	0	212
left	0	0	0	0	0	0	0	0	0	203	209

FIGURE 4. (Top) The t-distributed stochastic neighbor embedding representation of the test sample embeddings obtained from the ResNet15 model. The ACC score is reported close to every class cluster. (Bottom) Confusion matrix obtained when $\gamma = m$.

embeddings obtained from the GSC+ utterances are plotted on a 2-D space using t-SNE (upper part of the figure), together with a subset of unknown samples (the black points not clustered). Some classes, e.g., *stop*, *yes*, and *no*, feature a high accuracy because of the compact representation and the nonoverlap with adjacent classes. Conversely, other keywords, such as *up*, *off* , and *on*, suffer from a scattered representation, leading to a per-class accuracy of <50% when γ = m. As can be seen from the confusion matrix in the lower part of the figure, the model predicts a large part of the positive samples as unknowns, while critical errors, i.e., the wrong target class, are a minimal part.

MCU DEPLOYMENT

We assess the deployment costs of our solution on the GAP9 MCU, which includes a compute cluster with nine RISC-V cores, supporting vectorized half-precision floating point (FP16) and 8-bit integer MAC instructions, and a hardware (HW) convolution accelerator. To quantize and deploy the DNN models, we adopt the *GAPflow* toolset, which is provided by the chip manufacturer. Every model is quantized to 8 bit with asymmetric per-channel quantization ranges, as supported by the HW accelerator. We rely on post-training quantization by feeding four random training samples to the frozen DNN model to estimate the quantization ranges.

TABLE 3. Accuracy of the quantized models and deployment costs on the GAP9 MCU at 240 MHz and powered at 0.65 V.

Model	Performance 10 shots on 4 + 2 keywords		DNN inf Time (ms)	Efficiency (MAC/cyc)	Emb Compute (ms)	Energy per inf (μJ)
	Full Precision	Quantized				
TCResNet8	ACC: 0.62 FAR: 0.0083	ACC: 0.63 FAR: 0.0079	1.11	9.1	0	28.3
DS-CNN S	ACC: 0.69 FAR: 0.0088	ACC: 0.69 FAR: 0.0085	0.61	18.3	0.8	35.7
DS-CNN M	ACC: 0.66 FAR: 0.0064	ACC: 0.66 FAR: 0.0061	1.7	22.3	2.02	94.8
DS-CNN L	ACC: 0.70 FAR: 0.0049	ACC: 0.69 FAR: 0.0051	4.08	27.7	3.23	186.2
ResNet15	ACC: 0.73 FAR: 0.0013	ACC: 0.73 FAR: 0.0013	9.21	66.8	0.48	246.9

ACC: classification accuracy; DNN: deep neural network; inf: inference; MAC/cyc: multiply-and-accumulate operations per clock cycle.

Table 3 reports the accuracy before and after the quantization of a 10-shot test with four classes (*yes*, *stop*, *go*, and *no*) in addition to the *unknown* and silence prototypes. We reduced the number of keywords with respect to the previous setting to attain to a real use case scenario with an acceptable accuracy ($>$70%). Overall, the post-training quantization works in a lossless manner on our few-shot problem. On the ResNet15, batch normalization layers are quantized individually because the current version of the tool cannot fold them into the subsequent convolution layers (other models feature a reverse order of these layers). In this configuration, 8-bit quantization initially caused an accuracy drop with respect to the original model that is fully recovered by casting the BatchNorm layer quantization to FP16.

Table 3 also reports the latency measured on-chip when the clock frequency is set at 240 MHz (at a supply voltage of 0.65 V). In particular, we notice the DNN inference speed of ResNet15 is 2.25$\times$ faster than DS-CNN-L irrespective of the 8.4$\times$ higher number of MAC operations. This is due to the variable execution efficiency of different layer types on the GAP9 HW convolution accelerator. While a 3$\times$3 2-D convolution layer inside ResNet15 can reach a peak efficiency of up to 100 MAC operations per clock cycle (MAC/cyc), the pointwise layers of the DS-CNN are limited to 40 MAC/cyc. In the latter model, the computation of the embedding vector with a LayerNorm operator is also impacting the total time, constituting 44% of the workload on the cluster cores (*Emb Compute* in the table). This workload includes the computation of the mean and standard deviation of the feature map values before the average pooling, which may be reduced by further optimization. On the contrary, ResNet15 avoids such a cost by using BatchNormalization before the pooling layer. Considering a power consumption of 25.5 mW,[7] the solution using the largest model is 8.7$\times$ more energy hungry than running TC-ResNet8 but $+$10% more accurate and with a 0.66% lower FAR on the considered benchmark.

COMPARISON WITH OTHER WORKS

Zhang et al.[2] showed an MCU implementation of lightweight DNN models for KWS, including the DS-CNN backbones. When trained end to end on the GSC dataset, these models report an accuracy of $>$94% on the 12 categories. Vygon and Mikhaylovskiy[4] proposed training a ResNet15 using the triplet loss on the same GSC dataset, achieving a top performance of $>$98% accuracy when using a k-nearest neighbors (KNN) as a classifier. A similar approach is adopted by Huh et al.[5] but proposing a centroid-based classifier or a support vector machine (SVM) trained with all of the positive samples of the train set. In this work, we also adopt the triplet loss to train our tiny DNN feature extractors but on the MSWC dataset, which is disjoined from the test set. Additionally, our prototype-based classifier is initialized with only a few examples and features a lower memory cost than the KNN and the SVM, which have to store all of the reference examples or the support vectors.

TABLE 4. Comparison with other methods.

Approach	Number of shots	ACC	FAR	ACC	Target Train
		$\gamma = 0$		FAR = 5%	
DS-CNN-L					
Triplet + prototypes (ours)	3	0.84	0.37	**0.70**	—
	5	0.86	0.44	**0.75**	
	10	0.88	0.51	**0.77**	
Transfer learning classifier	3	0.77	0.79	0.30	Only class
	5	0.82	0.70	0.44	
	10	0.87	0.60	0.56	
End to end[2]	All data	**0.95**	—	—	Class + feature
ResNet15					
Triplet + prototypes (ours)	3	0.85	0.35	**0.74**	—
	5	0.87	0.39	**0.78**	
	10	0.88	0.55	**0.80**	
Triplet + SVM[5]	All data	0.93	0.35	—	Feat extractor
Triplet + KNN[4]	All data	**0.98**	—	—	Feat extractor

Bold refers to the best scores. KNN: k-nearest neighbors; SVM: support vector machine.

In the context of few-shot learning for KWS, many works relied on the prototypical network (ProtoNet) approach.[9] Similarly to our proposed approach, ProtoNet uses episodic training and prototype-based classification, but cross entropy is adopted as the optimization criterion. Unfortunately, several precedent studies fetched train and test data from the same domain (typically GSC) or did not analyze the FAR of the classifier on negative data, reporting only the accuracy over a test set of target keywords. Additionally, no work in this class focused on TinyML models and their implementation.

Another set of works, denoted as "query by example," leverages the embeddings extracted using a recurrent neural network (RNN) to individuate a wake word. As an example, the work by Chen et al.[8] trained a two-layer long short-term memory and proposed using a similarity score to compare the last embedding state with a reference vector of the target keyword. RNN methods are, however, expensive at inference time because of the memory-bound workload, which is up to 28 times less efficient than running convolutional models on our target HW.[10] A design of this class has been prototyped instead on an Hi3516EV200 development board with an ARM Cortex-A7 processor running at 900 MHz and 32 MB of memory under a power envelope of 100 mW, which is much higher than the capacity of MCU devices.[11]

Recently, Mazumder et al.[6] used transfer learning to train a DNN-based classifier with few shots. This approach places a linear classifier on top of a backbone model trained on the multilingual MSWC dataset. The authors employed an EfficientNet model with 11 million parameters as the backbone and trained the classifier to distinguish a single target keyword from the *unknown* and background (silence) classes.

Table 4 compares the accuracy scores at FAR = 5% with three, five, and 10 shots and 10 classes with the results of other methods. In Table 4, we also report the ACC and FAR when $\gamma = 0$. For comparison purposes, we applied a transfer learning approach to the DS-CNN-L model in a few-shot regime. In this case, we train the DS-CNN-L backbone on the MSWC dataset featuring a linear classifier for classifying the 500 keyword classes. At test time, a new classifier is trained on the frozen backbone for eight epochs. Our experimental results show that our proposed approach is both more accurate than a transfer-learning-based solution and does not require any training operation on the target dataset, in opposition to preceding methods.[2,4,5] In particular, we notice that the classifier trained using transfer learning is more prone to false positives in an open-set setting: the accuracy drop when tuning the threshold γ to achieve the target FAR of 5% .

CONCLUSION

This work showed the effectiveness of TinyML models for KWS classification with few-shot initialization without any backpropagation-based learning. As a top result, we showed a ResNet15 model able to score 80% at an FAR of 5% using only 10 samples per class not seen during training. This model runs efficiently on an MCU device in fewer than 10 ms under a power envelope of 25 mW. We believe this result motivates an increasing research effort toward the development of fast customization techniques for TinyML devices. Future work will also address adaptation strategies to bridge the accuracy gap with respect to models trained end to end on the target dataset.

AS A TOP RESULT, WE SHOWED A RESNET15 MODEL ABLE TO SCORE 80% AT AN FAR OF 5% USING ONLY 10 SAMPLES PER CLASS NOT SEEN DURING TRAINING.

ACKNOWLEDGMENTS

This work was partly supported by the European Horizon Europe program under Grant Agreement 101067475. We thank Marco Fariselli for the technical support in developing the system solution.

REFERENCES

1. I. Lopez-Espejo, Z. H. Tan, J. H. Hansen, and J. Jensen, "Deep spoken keyword spotting: An overview," *IEEE Access*, vol. 10, pp. 4169–4199, 2022, doi: 10.1109/ACCESS.2021.3139508.
2. Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," 2017, *arXiv:1711.07128*.
3. A. Coucke, M. Chlieh, T. Gisselbrecht, T. Leroy, M. Poumeyrol, and T. Lavril, "Efficient keyword spotting using dilated convolutions and gating," in *Proc. Int. Conf. IEEE Acoust., Speech Signal Process. (ICASSP)*, 2019, pp. 6351–6355, doi: 10.1109/ICASSP.2019.8683474.
4. R. Vygon and N. Mikhaylovskiy, "Learning efficient representations for keyword spotting with triplet loss," in *Proc. Speech Comput., 23rd Int. Conf.*, 2021, pp. 773–785, doi: 10.1007/978-3-030-87802-3_69.
5. J. Huh, M. Lee, H. Heo, S. Mun, and J. S. Chung, "Metric learning for keyword spotting," in *Proc. IEEE Spoken Lang. Technol. Workshop*, 2021, pp. 133–140, doi: 10.1109/SLT48900.2021.9383571.
6. M. Mazumder, C. Banbury, J. Meyer, P. Warden, and V. J. Reddi, "Few-shot keyword spotting in any language," in *Proc. Interspeech*, 2021, pp. 4214–4218, doi: 10.21437/Interspeech.2021-1966.
7. "ML commons tiny inference benchmark results v1.0." MLCommons. Accessed: Sep. 19, 2023. [Online] Available: https://mlcommons.org/en/inference-tiny-10/
8. G. Chen, C. Parada, and T. Sainath, "Query-by-example keyword spotting using long short-term memory networks," in *Proc. Int. Conf. IEEE Acoust., Speech Signal Process. (ICASSP)*, 2015, pp. 5236–5240, doi: 10.1109/ICASSP.2015.7178970.
9. B. Kim, S. Yang, I. Chung, and S. Chang, "Dummy prototypical networks for few-shot open-set keyword spotting," in *Proc. Interspeech*, 2022, pp. 4621–4625.
10. M. Rusci, M. Fariselli, M. Croome, F. Paci, and E. Flamand, "Accelerating RNN-based speech enhancement on a multi-core MCU with mixed FP16-INT8 post-training quantization," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases (ECML PKDD)*, 2022, pp. 606–617, doi: 10.1007/978-3-031-23618-1_41.
11. M. Li, "A lightweight architecture for query-by-example keyword spotting on low-power IoT devices," *IEEE Trans. Consum. Electron.*, vol. 69, no. 1, pp. 65–75, Feb. 2023, doi: 10.1109/TCE.2022.3213075.
12. A. Zhai and H. Wu, "Classification is a strong baseline for deep metric learning," in *Proc. Brit. Mach. Vision Conf. (BMVC)*, 2019, p. 91.

MANUELE RUSCI holds a Marie Skłodowska-Curie Actions postdoctoral fellowship at KU Leuven, 3000, Leuven, Belgium. His research interests include embedded machine learning and low-power smart sensors. Rusci received his Ph.D. degree in electrical engineering from the University of Bologna. Contact him at manuele.rusci@esat.kuleuven.be.

TINNE TUYTELAARS is a full professor at KU Leuven, 3000, Leuven, Belgium. Her main interests are computer vision and, in particular, topics related to image representations, continual learning, and dynamic architectures. Tuytelaars received her Ph.D, degree 2000 from KU Leuven. She is a Member of IEEE. Contact her at tinne.tuytelaars@esat.kuleuven.be.