Incremental On-Device Tiny Machine Learning

Simone Disabato
simone.disabato@polimi.it
Dipartimento di Elettronica, Informazione e Bioingegneria,
Politecnico di Milano
Milano, Italy

Manuel Roveri
manuel.roveri@polimi.it
Dipartimento di Elettronica, Informazione e Bioingegneria,
Politecnico di Milano
Milano, Italy

ABSTRACT

Tiny Machine Learning (TML) is a novel research area aiming at designing and developing Machine Learning (ML) techniques meant to be executed on Embedded Systems and Internet-of-Things (IoT) units. Such techniques, which take into account the constraints on computation, memory, and energy characterizing the hardware platform they operate on, exploit approximation and pruning mechanisms to reduce the computational load and the memory demand of Machine and Deep Learning (DL) algorithms.

Despite the advancement of the research, TML solutions present in the literature assume that Embedded Systems and IoT units support only the *inference* of ML and DL algorithms, whereas their *training* is confined to more-powerful computing units (due to larger computational load and memory demand). This also prevents such pervasive devices from being able to learn in an incremental way directly from the field to improve the accuracy over time or to adapt to new working conditions.

The aim of this paper is to address such an open challenge by introducing an incremental algorithm based on transfer learning and k-nearest neighbor to support the on-device learning (and not only the inference) of ML and DL solutions on embedded systems and IoT units. Moreover, the proposed solution is general and can be applied to different application scenarios. Experimental results on image/audio benchmarks and two off-the-shelf hardware platforms show the feasibility and effectiveness of the proposed solution.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; **Sensor networks**; • **Computing methodologies** → **Learning paradigms**.

KEYWORDS

Tiny Machine Learning, Incremental Learning, Deep Learning, Embedded Systems, Internet-of-Things.

ACM Reference Format:
Simone Disabato and Manuel Roveri. 2020. Incremental On-Device Tiny Machine Learning. In *The 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3417313.3429378

AIChallengeIoT '20, November 16–19, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-8134-5/20/11…$15.00
https://doi.org/10.1145/3417313.3429378

1 INTRODUCTION

Technological progress in Internet-of-Things (IoT) and embedded systems has opened the way to a pervasive presence of distributed embedded applications in our everyday lives as well as in diversified segments of the market. However, to meet user expectations and satisfy application constraints (e.g., prolong the system lifetime, guarantee Quality-of-Service and increase application credibility) intelligent functionalities provided through the machine and deep learning solutions should be considered. This is a crucial ability to make decisions directly on the pervasive devices, hence reducing the "data-production-to-decision" latency and not requiring a permanent connection with the Cloud. Unfortunately, the high demand in computation and memory required by traditional machine and deep learning techniques do not match with the constraints on computation, memory, and energy characterizing embedded systems and IoT units [1, 4, 30].

Filling this gap is the goal of Tiny Machine Learning (TML), which is a novel and challenging research area aiming at designing and developing "tiny" machine learning (ML) and deep learning (DL) techniques able to be executed on embedded systems and IoT units. To achieve this goal, such techniques rely on approximation and pruning mechanisms to reduce the computational load and memory demand of ML/DL solutions. Relatively-large literature exists in the field (mainly based on neural networks, decision trees, and support vector machine suitably approximated and pruned) paving the way for embedded systems and IoT units able to pervasively execute ML algorithms in the environment they operate in. An analysis of the related literature is available in Section 2.

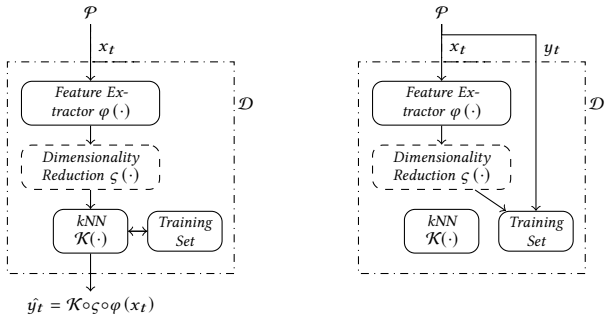
Despite the advances in the field, the *ability of embedded systems and IoT units to locally train their machine/deep learning models directly on-the-device* remains an *open research challenge*. In fact, all the solutions present in the literature (see Section 2) assume that devices only support the *inference* of ML/DL models, while the *training* (requiring a larger computational load and memory demand) is carried out on "external" more-powerful units (e.g., Cloud or Edge Computing systems). This also implies that embedded systems and IoT units are not even able to incrementally learn on the field as soon as new data arrive to improve their accuracy over time or to adapt to new working conditions.

This paper aims at addressing the aforementioned challenge by introducing a TML algorithm able to support the incremental learning of ML/DL models directly on embedded systems and IoT units. This goal is achieved by introducing an incremental mechanism based on transfer deep learning and nearest-neighbor that is able to support the local training and the evolution over time of intelligent

embedded systems and IoT units initially trained with only a small amount of data. This results in embedded systems or IoT units able to increase their accuracy over time and, if needed, to adapt to new working conditions. The proposed algorithm, which is here tailored to two application scenarios (i.e., image classification and speech command recognition), is general and can be applied to different application scenarios (e.g., image detection, sound classification, and event detection).

Experimental results on state-of-the-art benchmarks and two off-the-shelf hardware platforms show the feasibility and effectiveness of the proposed solution.

The remaining of the paper is organized as follow: Section 2 revises the related literature, whereas Section 3 details the proposed solution. In Section 4, the experimental results are presented and discussed and the conclusions are finally drawn in Section 5.



(a) Testing. **(b) Incremental Training.**

Figure 1: The proposed solution for incremental on-device learning on embedded systems and IoT units.

2 RELATED LITERATURE

The research in the field of ML/DL for embedded systems and IoT units is highly fragmented and mainly addressed from two different perspectives: custom hardware and approximated ML/DL.

Custom hardware refers to the design and development of embedded hardware platforms specifically meant for ML/DL. These custom platforms guarantee higher performance and lower power-consumption than general-purpose hardware. Unfortunately, these advantages come at the expense of increased complexity of the design phase together with reduced flexibility [8, 16, 37]. In the scenario of custom hardware, GPUs [10], TPUs [33], or neural hardware would allow to significantly reduce training and inference time but, unfortunately, they cannot be considered viable technological solutions in embedded systems and IoT units.

The design of *approximated machine/deep learning solutions* able to take into account technological constraints on memory, computation, and energy (characterizing embedded systems and IoT units) is a relevant and steadily-growing research field. Techniques belonging to this research field are diversified and fragmented but generally fall under the umbrella of Tiny Machine Learning (TML) [4, 29]. Interestingly, in TML, most of the related literature revolves around approximated Convolutional Neural Networks. For example, [17] introduces a methodology to explore sparse (and pruned) CNN architectures able to be executed on microcontroller units, whereas [23] proposed a decision tree-based technique, named Bonsai, to efficiently replicate the CNNs inference on Arduino boards. [19, 22] proposed the pruning of CNN layers or channels, along with Huffman coding on the weights, to reduce the memory footprint of CNN parameters, whereas [25] proposed decomposition techniques exploiting the redundancy of the convolutional kernels. [12] introduced linear approximations to these kernels, while [28] designed convolutional layers that are a linear combination of a few separable filters. Similarly, [24] proposed approximation techniques on normalization and pooling layers.

A different approach to approximate CNNs aims at reducing the memory required to store the CNN weights through quantization by using limited precision data types [7, 18] or binary weights [26, 27]. In such a direction, [1] combined both task dropping and precision scaling techniques to design approximated CNNs able to be executed in off-the-shelf embedded systems.

Other solutions focus on reducing the mean inference time of DL neural networks. In this direction, Adaptive Early Exit CNNs [5], Branchynet CNNs [31], and Gate-Classification CNNs [13] can provide the final classification output also at intermediate layers according to the input content, thus not requiring the execution of the whole CNN pipeline. However, the inference time is reduced at the expense of an increment of the memory footprint.

Summing up, all the TML solutions currently available assume that only the inference is performed on the device, whereas the training is carried out elsewhere.

3 THE PROPOSED SOLUTION

Let $\mathcal{P}$ a data-stream generating process that, at each time t, provides a pair (x_t, y_t) sampled from an unknown probability distribution $p_t(x, y)$, where x is the input of the proposed DL-solution $\mathcal{D}$ (e.g., an image or an audio clip) and y its classification label. Let Λ be the set of possible labels y and λ the cardinality of Λ, i.e., the number of classes in the considered classification problem.

Following the *test-then-train* approach [15], at each time instant t the proposed incremental TML solution $\mathcal{D}$ provides a classification output $\hat{y}_t = \mathcal{D}(x_t)$ for the input x_t and, after that, the supervised information, i.e., the true label y_t, is provided to $\mathcal{D}$. Without any loss of generality, this approach can be easily extended to the scenario where $\mathcal{D}$ classifies the input at each time instant while supervised information is provided more rarely.

More specifically, the supervised information (x_t, y_t) is provided to $\mathcal{D}$ to allow the incremental learning of the ML/DL model over time. Hence, thanks to this ability, embedded systems and IoT units endowed with the proposed solution $\mathcal{D}$ can be initially trained directly on-the-field with only a limited amount of data and, then, incrementally evolve due to the availability of new supervised samples (x_t, y_t)s to increase the accuracy.

Without any loss of generality, the process $\mathcal{P}$ is assumed to be stationary. However, the proposed architecture $\mathcal{D}$ can deal also with changes in $\mathcal{P}$, if a change-detector is provided, following an active adaptive approach [14, 15].

In this paper the proposed solution $\mathcal{D}$ is derived from [1], where a pre-trained Convolutional Neural Network (CNN) is approximated through Task-Dropping and Precision Scaling to match the memory and computation requirements characterizing embedded

systems or IoT units. In particular, Task-Dropping is justified by the transfer learning paradigm [36], which allows us to select a subset of the CNN processing chain as a feature extractor, leaving the classification task to an ad-hoc and lighter classifier, such as the SVMs or DTs. However, the limited computational and memory ability characterizing the micro-controller units (MCU) present in embedded systems and IoT units prevent the training of such classifiers directly on the device, hence requiring different approaches for the incremental on-device tiny learning. Moreover, the approach of [1] is not-incremental, meaning that once the approximated CNN is deployed on the embedded system, no further update of the classifier is possible even if new supervised samples are available.

More specifically, the proposed solution $\mathcal{D}$ relies on Task-Dropping to design a DL-based Feature-Extractor $\varrho(\cdot)$ from pre-trained models, but the classification is provided by a k-Nearest Neighbour (kNN) based classifier [3]. This family of non-parametric classifiers is theoretically-grounded and has the advantage of not requiring any training procedure (a particularly relevant feature for our scenario). For this reason, kNN classifiers can be incrementally updated when needed by simply including the additional supervised samples in their knowledge base. As an example, an IoT unit meant to recognize voice commands (e.g., to switch on and off home lights or open doors) could be initially trained with a limited subset of samples, and then the command recognition can be fine-tuned with the samples coming from a specific user (or users) directly and incrementally on-the-field. Remarkably, the proposed incremental TML solution $\mathcal{D}$ for embedded systems and IoT units is general and can be applied to different application scenarios. In the experimental Section 4, $\mathcal{D}$ will be tailored to image classification and speech recognition.

More formally, the proposed solution $\mathcal{D}$ receives an input x_t (e.g., an audio clip or an image according to the specific scenario) and provides as output its classification $\hat{y}_t \in \Lambda$. As shown in Figure 1, $\mathcal{D}$ comprises the following three steps:

(1) *a DL-based Feature-Extractor* $\varrho(\cdot)$ extracting the features from x_t. The Feature-Extractor is defined as in [1] from a pre-trained CNN (or any other DL model), satisfying the constraints on computation, memory, and energy characterizing the embedded systems and IoT units. Please refer to the paper for details about Task-Dropping (e.g., pruning of layers) and Precision Scaling (e.g., weights precision reduction). It is worth noting that, once the Feature-Extractor ϱ has been defined, it is not modified by the incremental updates.
(2) a *Dimensionality-Reduction operator* $\varsigma(\cdot)$ aiming at reducing the dimensionality of features generated by the feature extractor ϱ. To achieve this goal two different solutions have been considered:
   - *Filter-Selection without supervised information.* Given that the final convolutional layer of ϱ is characterized by F filters and following the transfer learning approach, this first dimensionality reduction technique selects the $f < F$ filters that are characterized by the highest mean activation on publicly-available benchmarks or datasets. This step is carried out at design-time (hence before loading the code on the device) and can be seen as part of the Task-Dropping step. The discarded filters are indeed not loaded on the embedded systems and IoT units, hence reducing the memory occupation and computational load;
   - *Filter-Selection with supervised information.* Differently from the previous approach, here the selection of the $f < F$ filters is accomplished directly on the device and aims at maximizing a specific figure of merit on the given classification problem. In this work, the considered figure of merit is the distance among classes in Λ, i.e., the f selected filters are those whose activations have the largest distance among the λ classes on an initially acquired training set. Even in this case, the feature-extractor ϱ is modified after the filter selection by discarding (and not executing) the $F - f$ filters that have not been selected. Being fulfilled on the device, this filter selection phase could, in principle, be re-executed when needed. However, in this case, all the F filters should be kept on the device to allow the selection (hence not reducing the memory occupation of the filters).
(3) a kNN *classifier* $\mathcal{K}(\cdot)$ processing the input features and incrementally learning over time. The kNN is a statistical classifier based on majority voting, i.e., the class of the input is defined as the majority class of the k nearest neighbors of the input sample (being k a parameter of the classifier). In our TML solution, the input of the classifier is either the output of the dimensionality-reduction operator $\varsigma \circ \varrho$ or the output of the feature extractor ϱ (when no dimensionality reduction is considered). The number k of neighbors used for the classification is defined as the ceiling of the square root of the available samples (further details about the selection of k can be found in [2]). This classifier is particularly interesting for our scenario since, being non-parametric, it can operate even with few training samples and is meant to operate incrementally (not requiring a proper training phase). In the proposed solution $\mathcal{D}$ this incremental step is done by simply including the new supervised information (x_t, y_t) in the training set of the classifier. These advantages come, in principle, at the expense of the following two drawbacks: the need to keep all the training samples in the knowledge base of the classifier and the computational complexity associated with the exploration of the whole training set for the inference. Both issues could be addressed by considering condensing [20, 32] and/or editing [35] techniques to reduce both the memory and the complexity required by the kNN classifier. In particular, the condensing techniques aim at finding the minimal subset of samples that is able to correctly classify all the available samples (hence discarding the remaining ones), whereas the editing techniques remove the samples that do not agree with their neighborhood.

4 EXPERIMENTAL RESULTS

The proposed solution has been validated on two different application scenarios, i.e., speech command identification in short audio clips and image classification, and on two different hardware platforms, i.e., a Raspberry Pi 3B+ and an STM32F7 board. Remarkably, the Raspberry Pi 3B+ belongs to the category of embedded PCs being equipped with a 1.4GHz 64-bit quad-core processor, 1GB of

RAM, and hosting a Linux Operating System, whereas the STM32F7 board is a technological solution for embedded systems and IoT units characterized by a 216 MHz-Cortex M7, 512 KB of RAM, 2 MB of Flash, and no OS. This difference is particularly relevant since the STM32F7 board (as well as the other hardware platforms based on MCU) has never been considered a viable technological solution for the on-device incremental learning of ML and DL solutions.

The remaining of the Section is organized as follow. Section 4.1 describes the application scenarios along with the corresponding considered datasets, whereas Section 4.2 details the employed feature extractor ϱ and its dimensionality reduction operator ς. Sections 4.3 and 4.4 present the comparison with a state-of-the-art solution on the Raspberry Pi 3B+ and the implementation of the proposed solution on the STM32F7, respectively.

4.1 Application Scenarios and Datasets

In the experiments, the following two application scenarios have been considered:

- In the *speech command identification* scenario the goal is to correctly recognize a speech command given by the user within a one-second long audio clip. For this purpose, the *Synthetic Speech Commands Dataset* [6, 34] has been considered. This dataset is composed of 30 classes, each of which corresponding to one single command, e.g., "*up*", "*left*", "*yes*", "*go*", or a number from "*zero*" to "*nine*". There is a huge variety in the voices within the one-second long WAV files composing the dataset and there are two variants of each class, i.e., noisy or noise-free.
- In the *image classification* scenario the goal is to classify an image containing exactly one object. The employed dataset is the well-known ImageNet [11], comprising 1000 classes.

In both scenarios, $\lambda = 2$, i.e., a binary classification problem is addressed.

4.2 Feature Extractor

In this experimental analysis, the considered feature extractor ϱ is the first layer of the well-known ResNet-18 CNN [21]. Such a layer comprises a convolutional layer with 64 7x7 filters with stride 2, a batch-normalization layer, a ReLU non-linearity, and a 3x3 max-pooling layer with stride 2.

In the *speech command identification* scenario the audio waveform is converted into a spectrogram through a Short-Time Fourier Transform with a window of size 512 (then "colored" by relying on a colormap), whereas in the *image classification* the images are resized to the size of 224x224x3 before being passed as input to $\mathcal{D}$.

In our experimental analysis, both types of Filter-Selection described in Section 3 have been considered. In the case of Filter-Selection without supervised information, the feature extractor ϱ is followed by the dimensionality reduction operator ς, where the number of selected filter f is 5 and the selected filters are those characterized by the largest mean activation on a group of randomly selected classes within the ImageNet dataset. In the case of Filter-Selection with supervised information, the dimensionality reduction operator refers to $f = 1$ filter selected (as described in Section 3) among the 5 filters mentioned above.

4.3 Experimental Results on Raspberry Pi 3B+

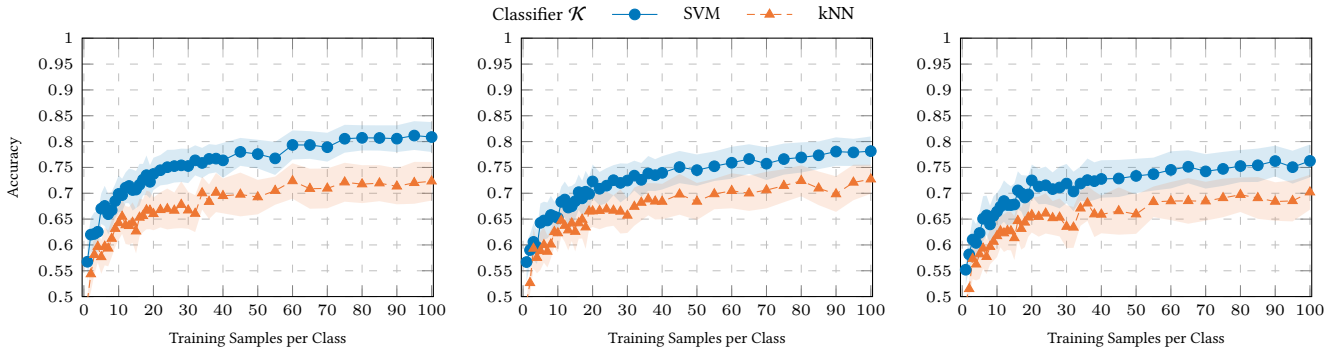
The proposed solution has been implemented in Python on the Raspberry Pi 3B+. For both application scenarios, the available audio clips and images are divided into a test set of size 600 samples and a training set of increasing size (from 1 sample per class to 100 samples per class). The first 20 seeds (from 1 to 20) have been considered for comparison by computing mean and standard deviation of the following two figures of merit:

- the classification accuracy on the test set;
- the computing times required by each step of the proposed solution $\mathcal{D}$ on the whole test set. In detail, the measured times are those of the features extraction (comprising the data loading and the processing of ϱ and ς, if used), the training phase of the classifier $\mathcal{K}$, and the inference of $\mathcal{K}$. The computation times associated with the feature extraction and the inference refers to the execution on the whole test set. Hence, the feature extraction and inference time on a single image can be computed by dividing by 600 the times shown in the results. The training time does not refer to the test set and depends only on the number of samples incrementally introduced in the classifier training set.

In particular, the kNN classifier $\mathcal{K}$ within $\mathcal{D}$ is compared with a Support Vector Machine (SVM) [9] operating on the same extracted features. It is crucial to point out that the SVM classifier used as a comparison is not incremental and its training phase is unfeasible on MCUs of embedded systems and IoT units (as commented in Section 4.4). Finally, at each time instant t, the proposed solution $\mathcal{D}$ receives a couple of supervised samples belonging to the two different classes in Λ, after having computed the classification error on the test set.

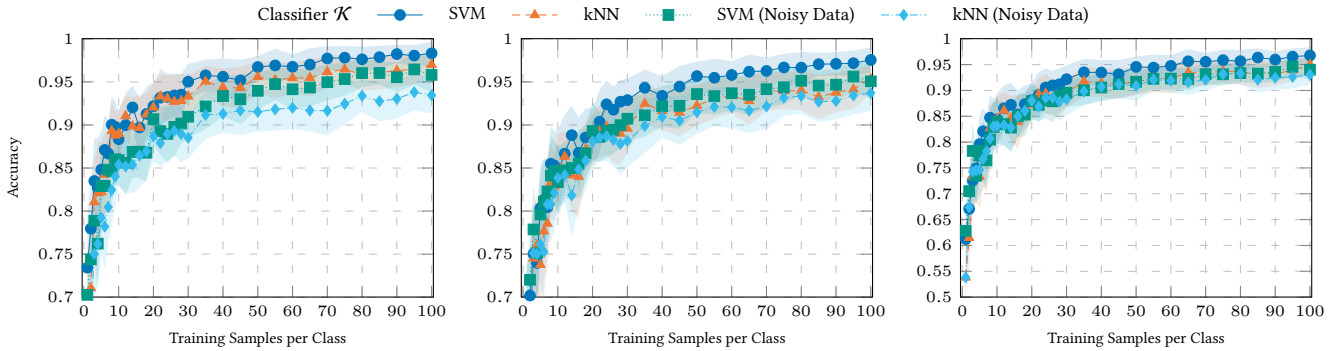
Image Classification Scenario Results. The classification accuracy and the computing times on the *image classification* scenario are shown in Figures 2 and 4, respectively. As expected, the SVM provides larger accuracies than the kNN, with a gap in the order 0.05 - 0.10 (see Figure 2). This advantage in accuracy comes at the expense of a more complex training phase whose effect is evident in the training times of both classifiers (see Figure 4). Moreover, the complexity of the SVM training phase prevents its use on the MCU of the STM32F7 board (as described in Section 4.4). Interestingly, starting from 10-20 samples per class, it is possible to observe that there are negligible increments of classification accuracy when new samples are incrementally inserted into the classifiers. This result, which is more evident on the kNN, is crucial to show that, in principle, it is possible to store only a smaller number of samples with negligible losses in classification accuracy. As expected, Figure 4 shows that the computing time required by the kNN for the classification is significantly influenced by the number of training samples (the more the samples the higher the time) and by the number of considered neighbors (as shown by the steps in the plot). Hence, condensing and editing techniques advised in Section 3 would allow to significantly reduce the computation time and the memory demand with a negligible impact on the classification accuracy.

It is worth noting that the time required by the whole architecture $\mathcal{D}$ is dominated by the extraction of the features, which could even be a hundred times larger than the classification task.



(a) No dimensionality reduction. (b) Filter-Selection without supervised information. (c) Filter-Selection with supervised information.

Figure 2: The accuracy in the *image classification* scenario, with a different number of samples per class (x-axis). The results represent the mean ± standard deviation over 20 experiments on pairs of classes.



(a) No dimensionality reduction. (b) Filter-Selection without supervised information. (c) Filter-Selection with supervised information.

Figure 3: The accuracy in the *speech command identification* scenario, with a different number of samples per class (x-axis). The results represent the mean ± standard deviation over 20 experiments on pairs of classes.

Interestingly, the adoption of a dimensionality reduction operator ς has huge advantages in the computation time (refer to Figures 4b and 4c) since the feature extraction time is halved and the classification one is reduced up to twenty times. Unfortunately, as shown in Figures 2b and 2c the reduction of the computation time results in an accuracy drop.

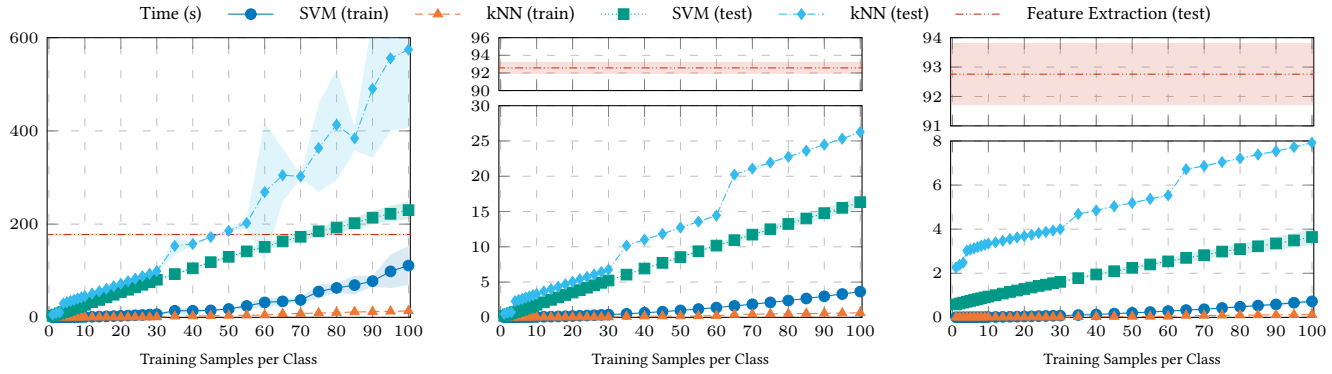
Speech Command Identification Scenario Results. The classification accuracy and the computation times on the *speech command identification* scenario are shown in Figures 3 and 5, respectively.

In this scenario the gap between the SVM and the kNN is significantly reduced, being always smaller than 0.05. Remarkably, the use of a kNN provides classification accuracies that are slightly worse than the ones of the SVM but with the significant advantage of not requiring a training phase. Even in this scenario, the classification accuracy of both classifiers reaches a plateau, here 35 samples per class in noisy commands and 20-25 samples on noise-free commands. Similarly to the *image classification* scenario, the computation time required by $\mathcal{D}$ is dominated by the feature extractor. However, despite the spectrograms generated from the one-second-long audio clips are smaller than the image size of 224x224x3, the measured times are larger, suggesting that the most time-consuming operation is the generation of the spectrogram.

The effects of applying a dimensionality reduction operator ς are still evident in the reduction of the computation time (see Figures 5b and 5c) since the feature extractor time is reduced by 20s and the classification time up to ten times. Differently, when considering the accuracy (Figures 3b and 3c), there is a different behavior on noisy or not audio clips. The accuracy of the former is higher, whereas it is reduced on the latter. However, in both cases, the effect on the classification accuracy is very small (smaller than 0.05).

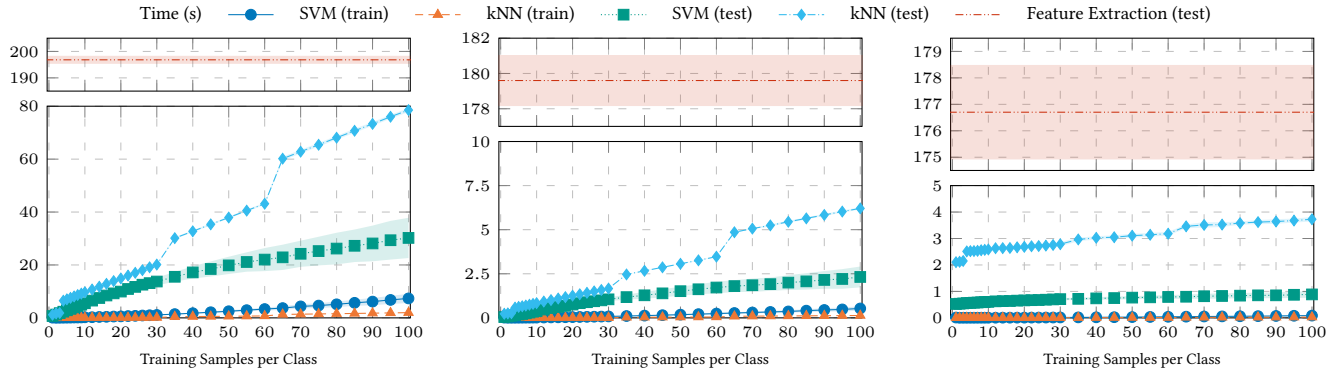
4.4 Experimental Results on STM32F7

The architecture $\mathcal{D}$ is moved, with a C implementation to the STM32F7 in the *image classification* scenario where the Filter-Selection without supervised information dimensionality reduction operator detailed in Section 4.2 is considered. Here, the considered figure of merit is the computation times since the accuracy is not affected by the technological implementation (hence classification accuracies shown for the Raspberry Pi 3B+ still hold). The computation times about the feature extractor and the kNN classifier are shown in Table 1 for two configurations of the training set (i.e., 10 training samples and 20 training samples), whereas the training time is not considered since no training occurs. Several



(a) No dimensionality reduction. (b) Filter-Selection without supervised information. (c) Filter-Selection with supervised information.

Figure 4: The times measured in seconds on a Raspberry Pi 3B+ in the *image classification* scenario, with a different number of samples per class (x-axis). The results represent the mean ± standard deviation over 20 experiments on pairs of classes and are divided into the various steps of the proposed solution $\mathcal{D}$ to provide a detailed view.



(a) No dimensionality reduction. (b) Filter-Selection without supervised information. (c) Filter-Selection with supervised information.

Figure 5: The times measured in seconds on a Raspberry Pi 3B+ in the *speech command identification* scenario, with a different number of samples per class (x-axis). The results represent the mean ± standard deviation over 20 experiments on pairs of classes and are divided into the various steps of the proposed solution $\mathcal{D}$ to provide a detailed view.

Table 1: The architecture $\mathcal{D}$ results in the *image classification* scenario and on the STM32F76ZI unit compared with the Raspberry Pi 3B+. The results are split into the feature extractor with 5 filters Filter-Selection without model information $\varsigma \circ \varrho$ and the classifier $\mathcal{K}$, where two configurations have been considered.

	Memory (KB)	Inference Time 1 Image (s)	
		Raspberry Pi 3B+	STM32F76ZI
$\varsigma \circ \varrho$	2.87	0.154	2.099
$\mathcal{K}$: kNN (NN=10, k=4)	612.50	0.004	0.009
$\mathcal{K}$: kNN (NN=20, k=5)	1 225.00	0.006	0.019

comments arise. First, these results show the feasibility of the proposed incremental solution on an MCU of an embedded system or IoT unit. Second, as shown in Section 4.3, the computation time associated with the feature extractor is larger than the one of the classifier. Finally, there is a 13x factor between the computation times of the STM32F7 and the ones of the Raspberry Pi 3B+, which corresponds to the ratio between the number of cores (1 to 4) and clock frequency (216 MHz to 1.4GHz).

5 CONCLUSIONS

The on-device learning of ML/DL solutions in embedded systems and IoT units is still an open and relevant research challenge. This paper aimed to introduce a novel incremental on-device solution based on deep transfer-learning and kNN classification allowing the incremental learning directly on embedded systems and IoT units to improve the accuracy over time and adapt (if needed) to new working conditions. The feasibility and effectiveness of the proposed solution have been tested on two off-the-shelf technological platforms and two application scenarios (i.e., speech command recognition and image classification).

Future work will encompass the introduction of incremental condensing and editing mechanisms for the kNN to reduce the execution time and the memory demand and the definition of adaptation mechanisms based on active classifiers to deal with time-varying data generation processes.

REFERENCES

[1] Cesare Alippi, Simone Disabato, and Manuel Roveri. 2018. Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, Porto, 212–223.

[2] Cesare Alippi and Manuel Roveri. 2008. Just-in-time adaptive classifiers—Part II: Designing the classifier. *IEEE Transactions on Neural Networks* 19, 12 (2008), 2053–2064.

[3] Naomi S Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician* 46, 3 (1992), 175–185.

[4] Colby R Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, et al. 2020. Benchmarking TinyML Systems: Challenges and Direction. *arXiv preprint arXiv:2003.04821* (2020).

[5] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 527–536.

[6] Johannes Buchner. 2017. Synthetic Speech Commands: A public dataset for single-word speech recognition. *Dataset available from https://www.kaggle.com/jbuchner/synthetic-speech-commands-dataset/* (2017).

[7] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5918–5926.

[8] Lukas Cavigelli and Luca Benini. 2016. Origami: A 803-gop/s/w convolutional network accelerator. *IEEE Transactions on Circuits and Systems for Video Technology* 27, 11 (2016), 2461–2475.

[9] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[10] Henggang Cui, Hao Zhang, Gregory R Ganger, Phillip B Gibbons, and Eric P Xing. 2016. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the Eleventh European Conference on Computer Systems*. 1–16.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[12] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*. 1269–1277.

[13] Simone Disabato and Manuel Roveri. 2018. Reducing the Computation Load of Convolutional Neural Networks through Gate Classification. In *2018 International Joint Conference on Neural Networks (IJCNN)*, Vol. 2018-July. IEEE, 208–215.

[14] Simone Disabato and Manuel Roveri. 2019. Learning Convolutional Neural Networks in presence of Concept Drift. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[15] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. 2015. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine* 10, 4 (2015), 12–25.

[16] Aysegul Dundar, Jonghoon Jin, Berin Martini, and Eugenio Culurciello. 2016. Embedded streaming deep neural networks accelerator with applications. *IEEE transactions on neural networks and learning systems* 28, 7 (2016), 1572–1583.

[17] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. 2019. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. In *Advances in Neural Information Processing Systems*. 4977–4989.

[18] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*. 1737–1746.

[19] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[20] Peter Hart. 1968. The condensed nearest neighbor rule (Corresp.). *IEEE transactions on information theory* 14, 3 (1968), 515–516.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[22] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *The IEEE International Conference on Computer Vision (ICCV)*.

[23] Ashish Kumar, Saurabh Goyal, and Manik Varma. 2017. Resource-efficient machine learning in 2 KB RAM for the internet of things. In *International Conference on Machine Learning*. 1935–1944.

[24] Dawei Li, Xiaolong Wang, and Deguang Kong. 2018. Deeprebirth: Accelerating deep neural network execution on mobile devices. In *Thirty-second AAAI conference on artificial intelligence*.

[25] Shaohui Lin, Rongrong Ji, Chao Chen, Dacheng Tao, and Jiebo Luo. 2018. Holistic cnn compression via low-rank decomposition with knowledge transfer. *IEEE transactions on pattern analysis and machine intelligence* 41, 12 (2018), 2889–2905.

[26] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*. 345–353.

[27] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*. Springer, 525–542.

[28] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. 2013. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2754–2761.

[29] Ramon Sanchez-Iborra and Antonio F Skarmeta. 2020. TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. *IEEE Circuits and Systems Magazine* 20, 3 (2020), 4–18.

[30] Jie Tang, Dawei Sun, Shaoshan Liu, and Jean-Luc Gaudiot. 2017. Enabling deep learning on IoT devices. *Computer* 50, 10 (2017), 92–96.

[31] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2464–2469.

[32] Ivan Tomek. 1976. Two Modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6, 11 (1976), 769–772.

[33] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. 2019. Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv preprint arXiv:1907.10701* (2019).

[34] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[35] Dennis L Wilson. 1972. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics* 3 (1972), 408–421.

[36] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*. 3320–3328.

[37] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 161–170.