TinyRCE: Forward Learning Under Tiny Constraints

Danilo Pau, FIEEE
System Research and Development
STMicroelectronics
Agrate Brianza, Italy
danilo.pau@st.com

Prem Kumar Ambrose
System Research and Development
STMicroelectronics
Agrate Brianza, Italy
premkumar.ambrose@st.com

Andrea Pisani
System Research and Development
STMicroelectronics
Agrate Brianza, Italy
aptrp99@gmail.com

Fabrizio M. Aymone
System Research and Development
STMicroelectronics
Agrate Brianza, Italy
fabrizio.aymone@gmail.com

Abstract—**The challenge posed by on-tiny-devices learning targeting ultra-low power devices has recently attracted several machine learning researchers. A typical on-device model learning session processes real time streams of data produced by heterogeneous sensors. In such a context, this paper proposes TinyRCE, a forward-only learning approach based on a hyper-spherical classifier aiming to be deployed on microcontrollers and, potentially, on sensors. The learning process is fed by labeled data streams to be classified by the proposed method. The classical RCE algorithm has been modified adding a forget mechanism to discard useless neurons from the classifier's hidden layer, since they could become redundant over time. TinyRCE is fed with compact features extracted by a convolutional neural network which could be an extreme learning machine. In such case, the weights of the topology were randomly initialized instead of trained offline with backpropagation. Its weights are stored in a tiny read-only memory of 76.45KiB. The classifier required up to 40.26KiB of RAM to perform a complete on-device learning workload in 0.216s, running on an MCU clocked at 480MHz. TinyRCE has been evaluated with a new interleaved learning and testing protocol to mimic an on-tiny-device forward learning workload. It has been tested with openly available datasets representing human activity monitoring (PAMAP2, SHL) and ball-bearing anomaly detection (CWRU) case studies. Experiments have shown that TinyRCE performed competitively against a supervised convolutional topology followed by a SoftMax classifier trained with backpropagation on all these datasets.**

Keywords— On-Tiny-Device Learning, TinyML, Hyper-Spherical Classifier, Extreme Learning Machines, Feature Extraction, Human Activity Recognition.

I. Introduction

Supervised Artificial Intelligence (AI) and Tiny Machine Learning (TinyML) are nowadays widely adopted approaches. They are deployed in many products in multiple applications offering high performance. [1] benchmarked multiple Deep Neural Networks (DNNs) reporting accuracy, memory usage, complexity, and latency, showing that as the complexity of DNNs grows to face more challenging problems, DNNs demand more memory and computational capability to support learning and K-fold validation, thus their model size increases as well. The deployment of complex DNNs on tiny devices and their use to process sensor-generated data streams can also be affected by a confidence drop between the accuracy achieved in training and the inference phase [2], due to various possible causes. As a consequence, continuous updates are required over time, thus the DNNs need to be re-trained. The de-facto standard training approach, adopted by many deep learning frameworks [1], is based on backpropagation and stochastic gradient descent (SGD). Indeed, backpropagation requires many iterations of forward and backward workloads applied to different partitions of the training data (K-folds) which, in turn, require large storage. Retraining with such an approach requires powerful computational and storage assets. Unfortunately, for tiny devices like micro-controller units (MCU), performing Continuous Learning (CL) becomes too costly. In the popular cloud-based support scenario, low-complexity tiny devices stream sensor data to the cloud, while the latter performs all the highly resource-demanding AI workloads. Although cloud-based solutions [3], with virtually unlimited computational and storage resources, seem to have clear advantages over the capability of tiny devices, they still account for many disadvantages, including risks for privacy and security of user data, higher latency, and the difficulty to achieve nW to µW power consumption targets. Backpropagation can also be heavily affected by catastrophic forgetting (CF) [4], which is the main threat to achieve CL. All of that represents an opportunity to reconsider supervised learning procedures and extend the state of the art beyond backpropagation, by conceiving new on-tiny-device learning algorithms. This paper proposes an approach to on-tiny-device CL without using backpropagation. Section II introduces the problem statement and requirements to be fulfilled by the solution; section III reviews existing related works and reports their limitations; section IV summarizes the datasets used to shape the case studies for experimenting the learning and testing workloads; section V explains the proposed solution, highlights differences with respect to the previous works and describes how the On-Device Learning (ODL) field was approached; section VI reports the experimental results; section VII analyzes the complexity of the proposed solution with respect to its deployment on MCUs, followed by the conclusions and future works in section VIII.

II. Problem Statement and Requirements

The question posed at the beginning of the work was: can incremental learning of multiple categories happen, totally on-line without catastrophic forgetting or back-propagation and be deployable on memory-constrained devices? Therefore, the requirements were set as reported in TABLE I. The proposed solution shall store much less sensor data than K-fold backpropagation learning procedures. The learning shall be determined by the few data made available at a certain time by the sensor. On-device learning of the DNN model shall happen within the MCU embedded memory constraints. The DNN model should take less time to complete the learning than to acquire the data. Most importantly, the DNN shall learn to classify multiple sequentially presented categories of data. The process shall maximize accuracy and be compared against backpropagation-based learning.

TABLE I. REQUIREMENTS TO BE FULFILLED BY TINY ODL

No.	Requirements
1.	Real time forward learning.
2.	No backpropagation.
3.	Deployable in MCU, optionally in the sensor within its embedded memory capacity.
4.	Perform classification.
5.	Capable of interleaved learning and testing workloads
6.	Minimal storage of sensor data.
7.	Limited latency to run learning workloads.

III. RELATED WORKS

A. On-device learning

CF [4] resulted from DNNs trying to learn from new data presented sequentially. The models discarded previous knowledge, leading to poor performances and misclassifications. CL extended the knowledge of DNNs to different tasks without retraining them from scratch nor forgetting previous knowledge. [5] reviewed different CL methods aimed to accumulate knowledge over different tasks. [6] reported CL methods for edge devices based on Latent Replay (LR). It was targeted for a specific 10-core Parallel Ultra Low Power (PULP) processor called VEGA. CL was supported with less than 64MB of memory. LR [7] consists in storing for later replay intermediate feature maps of few old data samples, instead of large data inputs, reducing up to 48x the memory usage.

A.1 Edge devices. [8] introduced an approach to train the DNN on hardware without backpropagation. HLS4ML, a Python package for ML inference in FPGAs, was used. The approach has not been tested on MCUs, though. [9] introduced an anomaly detector for rotating machines, which could be trained on-line with a low-power wireless Raspberry Pi Pico node. On-device learning increased the model's detection performance. TinyTL [10] proposes to update only the biases, freezing the weights, since doing so allows to not store intermediate activations. It also introduces a memory-efficient bias module to account for the resulting loss in capacity. It has been capable of achieving the same accuracy of fine-tuning full models while providing 7.3-12.9x memory savings. It required more than 37MB to run image classification and perform training on a GPU.

A.2 ODL on MCU. [11], [12] reported a state-of-the-art review for both ODL and ML on tiny devices and summarized the importance of the challenges faced. TinyOL [13] introduced an additional custom layer to the end of a static DNN. This layer's weights were fine-tuned on-device using real time sensor data. New classes were learnt with a customizable layer. Incremental learning removed large storage of data. This work did not support 8-bit MCUs, and the static model had to be trained offline. TML-CD [14] was a K-Nearest Neighbor (KNN), capable of updating the training set and discarding outdated knowledge after a change was detected in the data due to concept drift (CD), which is a serious problem faced by tiny devices when deployed in real applications. [15] proposed an algorithm/system co-design framework which was able to perform on-device learning requiring only 256KiB of memory. Quantization-Aware Scaling, a tiny training engine which supported sparse update of the DNN, parameters were key contributions.

Other ODL approaches targeted for MCUs were based on Reservoir Computing (RC). [16] was a deeply quantized anomaly detector of oil leaks in wind turbines feasible on 32bit MCU. It was tested on a specific dataset [17]: inference required 129.2KiB RAM; learning required 2.12MB of RAM. [18] introduced an ODL-capable anomaly detector for water distribution systems (WDS). [19] detected anomalies in ECG signals, achieving an accuracy of 95.4% with a memory usage of 60KiB; [20] was an Extreme Learning Machine (ELM) method mixed with RC, which ran with less than 192KiB of RAM and was used for field-oriented motor control.

B. Extreme Learning Machines

[21] introduced the concept of ELM on edge devices. ELM was introduced by [22] for regression and multiclass classification. Their training converged several times faster than Convolutional NN (CNN) with backpropagation. [23] clarified more the concepts behind ELM. Moreover, homogeneous architecture-based ELM DNNs have been used for regression tasks and to discriminate binary and multiple classes. [24] proposed a survey of on-line ELM to let the parameters evolve in a sequential manner, avoiding retraining using past samples that the model already saw. [25] ELM with Local Connections (ELM-LC) clustered inputs and hidden neurons into groups. Compared to traditional ELM approaches, it performed better.

C. Introduction to RCE and Hyperspherical Classifiers

RCE classifiers [26] are a branch of hyper-spherical classifiers [27]; like K-NN classifiers they can be trained without backpropagation. In the simplest form, they consist of three layers: an input, hidden, and output layer. The hidden and output layers are initialized by storing the multidimensional features computed by the input layer, which is usually a feature extractor. Hidden neurons are instantiated by a selected copy of the input features and defined by a center point and a radius. They compose the classifier's feature space. The radius determines the space of influence its hidden neuron forms in the feature space. Each hidden neuron is fully connected to the input layer, while only being connected to one neuron of the output layer. The latter is thus sparsely connected to the hidden layer. During the learning process, a new hidden neuron is created storing a copy of an input feature whenever this is not found inside any existing hidden neuron's region of influence. To verify if an input feature is inside the region of any already instantiated hidden neurons, a similarity measure is computed iteratively through all of them. The measure is calculated between the input data and each region of influence's center point, e.g., the Hamming distance is used if features are binarized; alternatively, the Euclidean distance is used. If such a distance is lower than the hypersphere's radius of only one hidden neuron, then the association to it is unambiguous. Vice versa, it would be ambiguous. To create a new hidden neuron, the copy of the uncharted input feature is associated with the default radius value. Should a neuron feature an infinitely small radius, it can be considered redundant and as such be removed. Unfortunately, in a basic implementation, that would not happen. [26] explained that this category of classifiers can discriminate patterns from a class that has not yet been learnt such as an unknown class. This type of approach has been proved to be applicable successfully to anomaly detection problems.

D. Related works on Hyperspherical classifiers

[28] was a DNN to segment hands appearing into images. The skin-colored pixels were classified and segmented using the RCE. An appropriate color space to analyze the skin was strategically chosen employing a reduced number of calculations. A new approach to reduce the neuron's radius has been introduced to lower the processing.

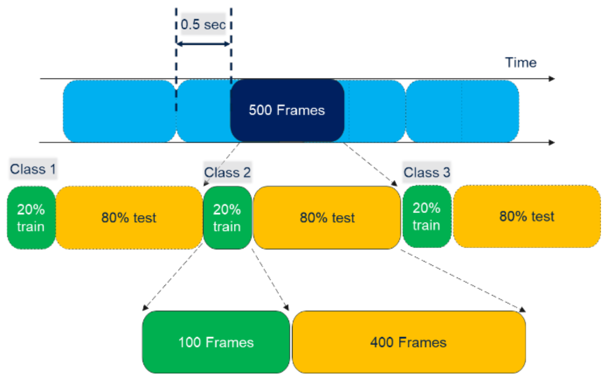


Fig. 1. Data arrangement to support both learning and testing workloads for HAR and BBAC. 6 refers to 3-axis accelerometer and 3-axis gyroscope integrated inside the chip's package of inertial sensors. Note the samples are presented in streaming order as they are temporally produced by the sensing element at its output.

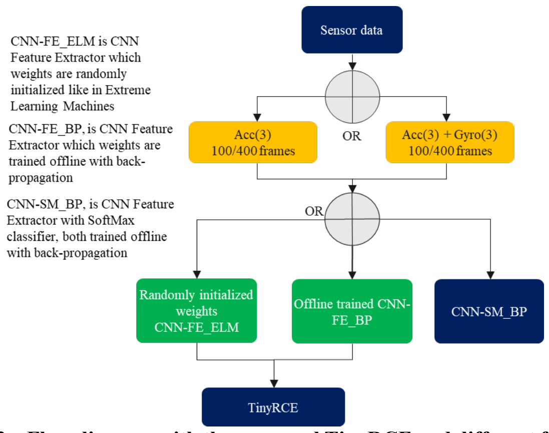
However, it did not approach removing redundant neurons. [29] proposed a RCE inductive classifier, which was more efficient than other similar approaches. All the above approaches were not meant for MCU deployment, adopted traditional offline training methods, and did not perform any ODL processing on MCU.

IV. Datasets

This work's case studies and simulations focused on Human Activity Recognition (HAR) and Ball Bearing Anomaly Classification (BBAC) datasets.

[30] PAMAP2 is a human physical activity monitoring dataset. Data were captured involving 9 subjects wearing 3 Inertial Measurement Units (IMU) and a heart rate monitor device. For the experimentation of the proposed solution, both the 3D-acceleration and 3D-gyroscope data with a G-range equal to 16 in different carry positions (hand, chest, and ankle) were used. A total of 24 classes is provided, but the maximum available classes per user are 12. [31] University of Sussex-Huawei Locomotion (SHL) is the user's locomotion and transport dataset. It provides data acquired using smartphones worn in different body positions: over the hand, chest, hip, and inside a backpack. It provides data from 3 users captured for 3 days using 4 smartphones and accumulating 59 hours of annotated recordings each, a total of 227 hours of data divided in 8 different classes. Case Western Reserve University Bearing Dataset (CWRU[1]) was used to represent the BBAC use case. It provides drive-end and fan-end vibration and rotation speed data from a 2hp reliance electric motor. Gyroscope data were collected close to the drive end and were sampled at 12 and 48KHz. Accelerometer data collected at fan end was sampled at 12KHz, with 10 health conditions.

This work arranged the raw sensor data, sampled at 100Hz, in frames composed of 100 samples, 6 degrees of freedom (DOF) values each, shifted with a step size equal to 0.5s. 500 of these frames were grouped together and used by the learning and testing workloads. 20% (100) of the 500 frames, i.e. 100s of acquisitions, were used by the learning workload. New classes are introduced sequentially over time and associated to the frames as shown in Fig. 1. The remaining 80% frames, capturing 400s of data acquisitions, were used by the testing workload. Should not further learning workload be needed (e.g. due to scarcity of available data in the datasets or no new classes being presented over time), then the testing could run longer than 400 seconds. Detection of concept drift was not part of the testing scenario of this work and will be experimented in future work. Differently from the K-fold learning and validation procedure, where data may be sampled in randomly ordered batches, the sequential, temporal order of the data produced by the sensor was not altered during the simulations of the proposed classifier. An interleaved forward fashion for both the learning and testing workloads is thus shown in Fig. 2.



CNN-FE_ELM is CNN Feature Extractor which weights are randomly initialized like in Extreme Learning Machines
CNN-FE_BP, is CNN Feature Extractor which weights are trained offline with back-propagation
CNN-SM_BP, is CNN Feature Extractor with SoftMax classifier, both trained offline with back-propagation
Sensor data
Acc(3) 100/400 frames
OR
Acc(3) + Gyro(3) 100/400 frames
OR
Randomly initialized weights CNN-FE_ELM
Offline trained CNN-FE_BP
CNN-SM_BP
TinyRCE

Fig. 2. Flow diagram with the proposed TinyRCE and different feature extraction approaches also using the SoftMax layer.

V. Proposed ODL Algorithm

Tiny Restricted Coulomb Energy (TinyRCE) NN is a hyper-spherical classifier variant proposed by this work. Fig. 2 proposes the overall processing flow to compare the proposed solution to a traditional backpropagation-based approach. The raw sensor data is input for the CNN feature extractor (FE). This was both randomly initialized (CNN-FE_ELM) and offline trained using backpropagation (CNN-FE_BP). Then their output features fed the TinyRCE classifier. The CNN connected to the SoftMax (CNN-SM_BP) was trained with backpropagation for comparison purposes.

A. Feature Extractor (CNN, CNN-FE)

The input features of TinyRCE were computed by a 2D CNN-FE to fulfill the requirements set in Section II. The FE converted the raw time series into 2D features, like [32], [33] approaches. The topology of FE was hand-crafted with trial-and-error attempts to design a computational graph which could output 24 values and, at the same time, be deployable on MCU. The value 24 was chosen as a trade-off between accuracy and computational complexity required by the learning workload. TABLE II. reports the topology of the CNN-FE and CNN-SM NNs. Both required FLASH memory of 76.45KiB assuming 32-bit floating point precision weights. CNN-FE_BP was trained offline with backpropagation. Then by removing its last dense SoftMax layer, it was connected to the TinyRCE classifier to run the learning and testing workloads. However, under a challenging ODL requirement, which excludes the use of backpropagation, the CNN-FE model shall not be trained offline. Therefore CNN-FE(_ELM)'s weights were initialized with a random normal distribution; the topology performed adequately, with a limited drop of about 1 to 3% with respect to the CNN-

[1] https://engineering.case.edu/bearingdatacenter/download-data-file

FE_BP's accuracy. Thus, it removed the need for offline training.

TABLE II. CNN_FE, CNN-SM TOPOLOGY FOR HAR AND BBAC USE CASES

Layers	Shape	Parameters
Convolution 2D	10,10,64	9664
MaxPooling 2D	5,5,64	0
Convolution 2D	5,5,6	9606
MaxPooling 2D	2,2,6	0
Flatten	Number of features	0
Dense	Number of classes	200÷300
	Total Parameters	19,570

The CNN-FE compacted the features up to 25 times with respect to the original 1s of raw data (i.e., from 6x100 to 24 features) to feed the classifier. Thus, the associated MCU RAM storage cost was reduced from 117KiB to 4.8KiB.

TABLE III. TINYRCE (FED BY CNN-FE_ELM) VS CNN-SM_BP PERFORMANCES ON USER 2 FOR USE CASES HAR AND BBAC

B. Proposed classifier optimized for ODL

TinyRCE was designed to address all the requirements in TABLE I. The whole pipeline of the proposed solution adopted the ELM variant of CNN-FE. Only the hidden neurons of TinyRCE were optimized by the learning

DNN	Training and Validation Method	**SHL** User2, hip carry position	**PAMAP2** User2, hand carry position	**CWRU** Drive end sensor position
CNN-SM_BP (30 epochs)	K-fold	84.19%	77.45%	80.52%
	ODL	**99.68%**	**92.54%**	**97.09%**
CNN-FE_ELM with TinyRCE	K-Fold	80%	60.3%	53.4%
	ODL (2 to 6 epochs)	**99.61%**	**91.84%**	**95.21%**

procedure meant to run on the MCU. This workload happened in two phases. During the acquisition of raw data, the user was performing and annotating the corresponding HAR activity, followed by feature extraction. The storage of the features in the MCU memory happened during this phase. In the next phase, TinyRCE optimized the hidden and output layers by processing these stored features. Only the features and associated labels resulting from the HAR user annotation process were used to start the TinyRCE learning workload. Due to the fixed size of the MCU's embedded memory, incremental learning, and the dynamic addition of new neurons into TinyRCE could quickly saturate it if handled in an uncontrolled way. Therefore, a neuron-culling mechanism was introduced to prevent that behavior. To guide the pruning decisions, each hidden neuron is assigned an age value. At time t_0, the age was set to *0* for each instantiated hidden neuron; at each iteration, the age of involved neurons was either increased or decreased. It was increased if the involved neuron brought to a correct prediction through its activation and decreased otherwise. Every input feature was iterated across the list of hidden neurons during the learning step, and the amount of increment or decrement value was the ratio (eq. 1) between the distance from the feature to the center of the hidden neuron and its radius.

$$age^{(new)} = age^{(old)} \pm \frac{dist(h_j, x_t)}{R_j} \qquad \text{(eq. 1)}$$

TABLE IV. CNN-SM_BP VS CNN-FE_ELM + TINYRCE ON PAMAP2 (HAR USE CASE): AVG. OF 8 USERS WITH 8 COMMON CLASSES

This age value enabled pruning of redundant neurons, during the learning. The pruning mechanism was triggered if a user-defined threshold had been exceeded. The maximum number of hidden neurons TinyRCE could store on the MCU was an implementation-selectable threshold, which was meant

Sensor Carry Position	TinyRCE Accuracy	CNN-SM_BP Accuracy	TinyRCE Variance	CNN-SM_BP Variance
Hand	87.62	**91.13**	4.95	**4.54**
Chest	89.67	**91.71**	3.08	**2.97**
Ankle	87.37	**89.37**	3.94	**3.06**

to set an upper limit on the memory footprint allowed. This limit depended on the specific MCU's onto which to deploy TinyRCE. To prevent it from consuming all the on-chip memory, the redundant neurons of the class with the lowest or negative age values were culled. Therefore, the MCU processing cost was reduced; and as a positive side effect, it marginally improved the classification accuracy, by less than 1%. To prevent CF, a fixed budget of hidden neurons was defined for each class. The pruning mechanism selected the neurons to be pruned also considering said budget of available neurons per class, in order to maintain a balanced pool of hyperspheres for every class.

Furthermore, TinyRCE was meant to detect unknown input patterns, thus reporting to the user about the need to provide a corresponding label. As soon as the user provided the label, it was associated to the corresponding features and stored in temporary memory to serve the learning workload (otherwise discarded if no label was associated). A total of between 1 and 100 samples of the training data, each of 100B requiring up to 9.76KiB RAM, could be stored by the algorithm as an implementation example for HAR case study.

It is well known that hyper-spherical classifiers produce feature spaces that overlap with each other's classes. This implies to activate simultaneously several hidden neurons associated to different classes. To avoid this issue, only the output of the hidden neuron for which the input pattern was closest to its center was considered the most reliable.

VI. EXPERIMENTAL RESULTS

TinyRCE with CNN-FE_ELM was tested with the proposed evaluation method, depicted in Fig. 2. It used batches of time series raw data in which 20% of them were used by the learning workload, and the remaining 80% were used by the testing workload. This approach was used to quantitatively measure TinyRCE's performance. It mimicked the incremental forward learning step interleaved with testing one. It has been tested with SHL, PAMAP2 (for HAR), and CWRU (for BBAC) datasets which provided, respectively, 8, 12, and 10 classes per user (for SHL and PAMAP2) or per motor type (for CWRU). All three datasets were decomposed into frames of 100 samples per second grouped into sliding windows shifted with a hop size of 50 samples. These frames were then composed into streams featuring classes presented incrementally one after the other, as shown in Fig. 1. The proposed split was the opposite of the traditional 80/20% split used by the backpropagation with K-fold validation.

CNN-SM_BP was the benchmark for performance comparison against TinyRCE. Since the interleaved workflow that was applied represents another contribution of this work, it was not possible to structure a comparison with the results of existing benchmarks. For K(=5)-fold validation, the traditional 80/20% split was used for both TinyRCE and CNN-SM_BP. For comparison, the results of the learning step applied to a single user are shown in TABLE III. These results were achieved on User 2, in the scenario where the learning is personalized per user. Said user was selected because it featured the highest data quantity available. TinyRCE marginally underperformed CNN-SM_BP by 0.7% achieving an accuracy of 91.84% for the PAMAP2 dataset (12 classes). Whereas on the SHL (8 classes) and CWRU (10 classes) datasets, TinyRCE performed against CNN-SM_BP by a marginal difference of 0.07% and non-marginal 1.88% drop, respectively. Furthermore, on the PAMAP2 dataset, an extensive evaluation was performed involving eight users with the same available eight classes across them. TABLE IV. reports the average accuracy and variance on them and per carry position. On average, TinyRCE underperformed by 2.52% with respect to CNN-SM_BP.

VII. Complexity analysis for tiny devices

Input features at time t required 48B. The hidden and output layer of TinyRCE required 30.46KiB to store the features. The training data required a buffer of 9.76KiB (24 features plus the corresponding labels times 4B). The classifier required buffering 2s of inputs for the inference, whose footprint was 2.34KiB (assuming 100s of data per 6 axes sampled, 2B each, at 100 Hz). The detailed layer-wise estimated memory usage of TinyRCE is shown in TABLE VI.

TABLE V. Complexity profiling without accounting the K(=5)-fold procedure. (*) means random weights

Metrics	CNN-SM_BP (single inference) FP32	CNN-SM_BP (learning) FP32 (K fold=1)	CNN-FE_ELM + TinyRCE (Single inference) FP32	CNN-FE_ELM + TinyRCE (learning) FP32
MACC	1.213 M	**8 G**	1.251 M	**23.4 M**
FLASH (KiB)	76.45	-	76.45 (*)	
RAM (KiB)	13.59	**954.57**	40.2	
Latency STM32L4 (ms) @80MHz	123.4	**813.25 sec**	127.2	**2.38 sec**
Latency STM32H7 (ms) @480 MHz	11.21	**73.86 sec**	11.55	**216**

The complexity was expressed as Multiply and ACCumulate operations (MACC) for TinyRCE inference and estimated accordingly to equation (eq. 2), where h is the number of hidden neurons, and n is the number of features input to the classifier, which for HAR and BBAC is equal to 24 features.

$$MACC_{inference} = h * [(n * 5) + 10] \qquad \textit{(eq. 2)}$$

The latency, memory footprint, MACC for CNN-SM_BP and TinyRCE both inference and with K(=1)-Fold learning and validation procedure are reported in TABLE VI. The CNN-SM_BP models were automatically analyzed with the X-CUBE-AI ver. 7.2.0 tool. The complexity was 1.21 million MACC per inference, while according to (eq. 2), TinyRCE inference was estimated to be 39K MACC; this, added with the CNN-FE_ELM's complexity, resulted in a total of 1.25 million MACC. Thus, the inference time was estimated to be 11.55ms on the STM32H7 MCU at 480MHz and 127.2ms on the STM32L4 MCU at 80MHz.

TABLE VI. Estimated memory footprint for TinyRCE. Parameters are coded with 32-bit floating point precision

Use	Parameters	RAM(Bytes) Inference	RAM(Bytes) Learning
Input layer	24	48	0
Hidden layer	(200 ÷ 300) *26	31,200	31,200
Output layer	8-12	12	12
Inference Data buffer	2*600	2400	0
Data stored for learning	(50 ÷ 100) *25	0	10,000
Other additional	5	0	20
	Total	**32.87KiB**	**40.26KiB**

CNN-FE_ELM occupied a total of 76.45KiB FLASH to store the weights, which were never required to be updated by the learning workload. In a very optimized implementation, they could potentially be generated on-line by a random software procedure, reducing furthermore the storage needed. TinyRCE's hidden layer required 30.47KiB, its output layer 12B, and the training data 9.76KiB. The total RAM was 40.26KiB to 41.26KiB.

Even though this analysis shows that CNN-SM_BP requires lower memory and inference time, it does not consider the very memory-demanding K-fold learning process, which requires to store the activations of every layer in order for backpropagation to compute the weights' derivatives.

TinyRCE learning's first phase was needed to acquire raw sensor data, convert it into features, and store the features into temporary memory (10KiB); CNN-FE_ELM required 1.213M $MACC_{inference}$ to process 1s of data. TinyRCE second phase was accountable for 23.4M $MACC_{learning}$. This number was computed by using Eq. (3):

$$MACC_{learning} = (h * [(n * 5) + 10]) * (N * E) \qquad \textit{(eq. 3)}$$

In (eq. 3), h is the number of hidden neurons (300), n is the number of features per input (24), N is the number of training frames (100), and E is the number of epochs. For the estimations, the maximum number of epochs encountered while conducting the study (6) was used.

The total training time was estimated for STM32L4 and STM32H7 and was computed by using 8.13 and 4.43 cycles per each MACC (which were measured by running NN workloads on the MCU). Results were 2.38s and 0.216s, respectively. The learning of CNN-SM_BP with respect to CNN-FE_ELM on the considered MCUs proved to be much more complex (342x). The RAM footprint of the former was close to 1MB, due to the required storage of activations for weight updates, while the latter required just 40.2 KiB.

VIII. CONCLUSIONS AND FUTURE WORKS

This paper introduced TinyRCE, to perform on MCU learning workloads of multiple classes presented sequentially in a forward fashion. No backpropagation method was required. A specific protocol for both learning and testing workloads was introduced, mimicking the streaming acquisition of raw sensor data. TinyRCE has been compared against CNN-SM_BP trained with backpropagation. It has also been evaluated by the K-fold protocol. TinyRCE achieved an accuracy of 99.61% on SHL, 91.84% on PAMAP2, and 95.21% on CWRU requiring only 40.26KiB of RAM. 1KiB more is required for input data extra buffering. The classifier's inference time was estimated to 11.55ms on STM32H7 MCU clocked at 480MHz and 127.2ms on STM32F4 MCU at 80MHz. TinyRCE proved competitive against CNN-SM_BP's traditional topology, while reducing the complexity and memory needed to perform learning. As future work, TinyRCE will be coded with low bit count fractional arithmetic. Moreover, 8 bits quantization of the CNN-FE will be experimented to lower the memory usage to store the weights as low as 19.35 KiB. Footprint reduction could enable integration into the sensor digital logic. Furthermore, to develop concept drift detection to trigger learning workloads. This will improve the configuration and dynamic adaptation of the hidden layer, as well as to investigate different model feature extractors experimented over new datasets and case studies.

IX. REFERENCES

[1] S. Bianco, R. Cadène, L. Celona, and P. Napoletano, "Benchmark Analysis of Representative Deep Neural Network Architectures," *CoRR*, vol. abs/1810.00736, 2018, [Online]. Available: http://arxiv.org/abs/1810.00736

[2] D. Vela, A. Sharp, R. Zhang, T. Nguyen, A. Hoang, and O. S. Pianykh, "Temporal quality degradation in AI models," *Sci Rep*, vol. 12, no. 1, p. 11654, 2022, doi: 10.1038/s41598-022-15245-z.

[3] F. Samie, L. Bauer, and J. Henkel, "From Cloud Down to Things: An Overview of Machine Learning in Internet of Things," *IEEE Internet Things J*, vol. 6, no. 3, pp. 4921–4934, 2019, doi: 10.1109/JIOT.2019.2893866.

[4] P. Kaushik, A. Gain, A. Kortylewski, and A. L. Yuille, "Understanding Catastrophic Forgetting and Remembering in Continual Learning with Optimal Relevance Mapping," *CoRR*, vol. abs/2102.11343, 2021, [Online]. Available: https://arxiv.org/abs/2102.11343

[5] M. de Lange *et al.*, "A Continual Learning Survey: Defying Forgetting in Classification Tasks," *IEEE Trans Pattern Anal Mach Intell*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022, doi: 10.1109/TPAMI.2021.3057446.

[6] L. Ravaglia, M. Rusci, D. Nadalini, A. Capotondi, F. Conti, and L. Benini, "A TinyML Platform for On-Device Continual Learning with Quantized Latent Replays," Oct. 2021, doi: 10.1109/JETCAS.2021.3121554.

[7] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, Jun. 2020, doi: 10.1016/j.inffus.2019.12.004.

[8] D. AbdulQader, S. Krishnan, and C. N. Coelho, "Enabling Incremental Training with Forward Pass for Edge Devices," Mar. 2021, [Online]. Available: http://arxiv.org/abs/2103.14007

[9] H. Matsutani, M. Tsukada, and M. Kondo, "On-Device Learning: A Neural Network Based Field-Trainable Edge AI," Mar. 2022, [Online]. Available: http://arxiv.org/abs/2203.01077

[10] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han, "TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning," 2020.

[11] D. Pau and P. K. Ambrose, "A quantitative review of automated neural search and on-device learning for tiny devices," Sep. 2022, doi: 10.36227/techrxiv.18724562.v1.

[12] D. Pau and P. Kumar Ambrose, "Automated Neural and On-Device Learning for Micro Controllers."

[13] H. Ren, D. Anicic, and T. Runkler, "TinyOL: TinyML with Online-Learning on Microcontrollers," Mar. 2021, [Online]. Available: http://arxiv.org/abs/2103.08295

[14] S. Disabato and M. Roveri, "Tiny Machine Learning for Concept Drift," Jul. 2021, [Online]. Available: http://arxiv.org/abs/2107.14759

[15] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-Device Training Under 256KB Memory," Jun. 2022.

[16] M. Cardoni, D. pietro Pau, L. Falaschetti, C. Turchetti, and M. Lattuada, "Online Learning of Oil Leaks Anomalies in Wind Turbines with Block Based Binary Reservoir," 2021. [Online]. Available: https://doi.org/

[17] M. Cardoni, "Oil leak dataset." Mendeley, 2021. doi: 10.17632/NBXZXN3FFK.1.

[18] D. Pau, A. Khiari, and D. Denaro, "Online learning on tiny micro-controllers for anomaly detection in water distribution systems."

[19] N. Abdennadher, D. Pau, and A. Bruna, "Fixed complexity tiny reservoir heterogeneous network for on-line ECG learning of anomalies," in *2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*, 2021, pp. 233–237. doi: 10.1109/GCCE53005.2021.9622022.

[20] N. O. Federici, D. Pau, N. Adami, and S. Benini, "Tiny Reservoir Computing for Extreme Learning of Motor Control."

[21] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks."

[22] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme Learning Machine for Regression and Multiclass Classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 513–529, 2012, doi: 10.1109/TSMCB.2011.2168604.

[23] G.-B. Huang, "What are Extreme Learning Machines? Filling the Gap Between Frank Rosenblatt's Dream and John von Neumann's Puzzle," *Cognit Comput*, vol. 7, no. 3, pp. 263–278, 2015, doi: 10.1007/s12559-015-9333-0.

[24] S. Zhang, W. Tan, and Y. Li, "A Survey of Online Sequential Extreme Learning Machine," in *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2018, pp. 45–50. doi: 10.1109/CoDIT.2018.8394791.

[25] F. Li, S. Yang, H. Huang, and W. Wu, "Extreme Learning Machine with Local Connections," Jan. 2018, [Online]. Available: http://arxiv.org/abs/1801.06975

[26] M. J. Hudak, "RCE Networks: An Experimental Investigation," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, Seattle, WA, USA: IEEE, 1991, pp. 849–854. doi: 10.1109/IJCNN.1991.155290.

[27] M. H. Hassoun, "Fundamentals of Artificial Neural Networks," *Proceedings of the IEEE*, vol. 84, no. 6, 2005, doi: 10.1109/jproc.1996.503146.

[28] C. Sui, N. M. Kwok, and T. Ren, "A Restricted Coulomb Energy (RCE) neural network system for hand image segmentation," in *Proceedings - 2011 Canadian Conference on Computer and Robot Vision, CRV 2011*, 2011, pp. 270–277. doi: 10.1109/CRV.2011.43.

[29] C. and E. F. Fanizzi Nicola and d'Amato, "ReduCE: A Reduced Coulomb Energy Network Method for Approximate Classification," in *The Semantic Web: Research and Applications*, P. and C. F. and C. P. and H. T. and H. E. and M. R. and O. E. and S. M. and S. E. Aroyo Lora and Traverso, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 323–337.

[30] A. Reiss and D. Stricker, "Introducing a New Benchmarked Dataset for Activity Monitoring," Sep. 2012. doi: 10.1109/ISWC.2012.13.

[31] H. Gjoreski *et al.*, "The University of Sussex-Huawei Locomotion and Transportation Dataset for Multimodal Analytics With Mobile Devices," *IEEE Access*, vol. 6, pp. 42592–42604, 2018, doi: 10.1109/ACCESS.2018.2858933.

[32] W. Chen and K. Shi, "A deep learning framework for time series classification using Relative Position Matrix and Convolutional Neural Network," *Neurocomputing*, vol. 359, pp. 384–394, 2019, doi: https://doi.org/10.1016/j.neucom.2019.06.032.

[33] T. Li, Y. Zhang, and T. Wang, "SRPM–CNN: a combined model based on slide relative position matrix and CNN for time series classification," *Complex & Intelligent Systems*, vol. 7, no. 3, pp. 1619–1631, 2021, doi: 10.1007/s40747-021-00296-y.