Tackling Time-Variability in sEMG-based Gesture Recognition with On-Device Incremental Learning and Temporal Convolutional Networks

Tackling Time-Variability in sEMG-based Gesture Recognition with On-Device Incremental Learning and Temporal Convolutional Networks

Alessio Burrello[1], Marcello Zanghieri[1], Cristian Sarti[1], Leonardo Ravaglia[1], Simone Benatti[1,3], Luca Benini[1,2]

***Abstract*— Human-machine interaction is showing promising results for robotic prosthesis control and rehabilitation. In these fields, hand movement recognition via surface electromyographic (sEMG) signals is one of the most promising approaches. However, it still suffers from the issue of sEMG signal's variability over time, which negatively impacts classification robustness. In particular, the non-stationarity of input signals and the surface electrodes' shift can cause up to 30% degradation in gesture recognition accuracy. This work addresses the temporal variability of the sEMG-based gesture recognition by proposing to train a Temporal Convolutional Network (TCN) incrementally over multiple gesture training sessions. Using incremental learning, we re-train our model on stored *latent* data spanning multiple sessions. We validate our approach on the UniBo-20-Session dataset, which includes 8 hand gestures from 3 subjects. Our incremental learning framework obtains 18.9% higher accuracy compared to a baseline with a standard single training session. Deploying our TCN on a Parallel, Ultra-Low Power (PULP) microcontroller unit (MCU), GAP8, we achieve an inference latency and energy of 12.9 ms and 0.66 mJ, respectively, with a weight memory footprint of 427 kB and a data memory footprint of 0.5-32 MB.**

I. Introduction

Human-Machine Interfaces (HMI) are constantly evolving in many industrial, commercial, and clinical applications [1]. HMI based on decoding surface electromyographic (sEMG) signals, to recognize hand gestures, is a well-established paradigm [2] and recently proposed machine learning methods improve the classification accuracy above 90% [3], [4], [5].

However, sEMG is strongly affected by fatigue, user adaptation, perspiration, sensor shifts, and other factors [6], which cause a strong variability of the signal over time, and impair the long-term robustness of recognition algorithms, causing up to 30% accuracy drop [7]. Deep Learning (DL), thanks to larger sEMG datasets and feature learning, mitigated the issue, leveraging training on multiple acquisition sessions [5], reducing the inter-session accuracy degradation to less than 5%. On the other hand, multi-session training presents 2 major drawbacks: it requires a large amount of data, impossible to fit into embedded devices with strict memory constraints, and the computational resources needed for a complete online forward/backward propagation exceed the capabilities of edge processors. Hence it is only possible in a laboratory setting, where data from new sessions can be uploaded on desktop machines or servers for re-training the DL model.

Incremental learning [8] tackles the problem of continually improving a learning model over new training examples. To perform it at the extreme edge, we can re-train only a portion of the network by exploiting stored feature maps in place of the full training dataset. This is often sufficient to allow the network to learn new data distributions [9] incrementally. A similar problem has recently been addressed in [10] for sEMG signals, where new gestures are learned over time, reaching an accuracy of $\sim 85\%$, using Hyperdimensional Computing (HDC). Despite the high efficiency of the HDC, its accuracy reduces progressively as new classes are added, posing limits to the number of learned gestures.

In this work, we target on-device multi-session training, and we tackle the time-variability of sEMG-based hand gesture recognition, proposing to incrementally train a Temporal Convolutional Network (TCN), a DL model for time series suitable for resource-constrained devices and real-time applications. We developed an incremental learning approach that allows training the network directly on the embedded device, as shown in [9], after gathering new data over time. The main contributions of the paper are:

- We show an incremental learning setup improving a TCN's accuracy from a single-session training by 18.9%, from 74.6% to 93.5%. The accuracy degrades only by 1.3% if compared to simultaneous training on all training sessions[1], thus showing that a continuous re-training of the network is possible by storing only latent data from previous training sessions, using only a small incremental dataset with a footprint of only 8MB (as opposed to the full dataset of 144 MB).
- We compare our TCN with SoA gesture recognition classifiers SVM [5] and HDC with the incremental learning policy, reaching 5.60% higher accuracy than the best competitor, an RBF-SVM using RMS input feature.

*This work was supported in part by the European H2020 FET Project OPRECOMP under Grant 732631.

[1]A. Burrello, M. Zanghieri, C. Sarti, L. Ravaglia, S. Benatti and L. Benini are with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy. name.surname@unibo.it

[2]L. Benini is also with the Department of Information Technology and Electrical Engineering at ETH Zurich, 8092 Zurich, Switzerland. lbenini@iis.ee.ethz.ch

[3]S.Benatti is also with the Dipartimento di Scienze e Metodi dell'Ingegneria, University of Modena e Reggio Emilia, Italy . simone.benatti@unimore.it

[1]Complete network training is not feasible neither on-chip and in a real-scenario where data are acquired over time

- We analyze the memory footprint and MACs of our solution. We also deploy our TCN on the commercial GAP8 edge processor [11], achieving 12.9 ms latency and an energy envelope of only 0.66 mJ per inference.

II. Materials & Methods

A. Surface Electromyographic Signal

EMG signal [12], [13] is a valuable index of muscles' activity since it is the biopotential that originates from the current generated by the ions flowing through the membrane of the muscular fibres in response to electrical stimuli from the central nervous system. Typical EMG amplitudes range from $10\,\mu\mathrm{V}$ to $1\,\mathrm{mV}$, with bandwidth up to $\sim 2\,\mathrm{kHz}$. The major noise sources which increase EMG variability and reduce signal quality are generated by motion artefacts, floating ground noise, crosstalk, power line interference [14] as well as variable skin impedance, electrode repositioning, and user adaptation [15]. For this reason, a pattern recognition algorithm requires periodic re-training or calibration to enable a reliable interface.

B. UniBo-20-Session Dataset

The dataset analyzed in this work comprises 20 acquisition sessions spread over 10 days (morning + afternoon) [5]. The dataset involves 3 subjects (all male, age 29 ± 3 years). Data were acquired using a custom 8-electrodes platform sampling at $4\,\mathrm{kHz}$. [5]

The average duration of each session is $7.5\,\mathrm{min}$, and each includes the rest position and 8 hand gestures repeated 6 times, with both contraction time and a rest interval of $3\,\mathrm{s}$. With 16-bit data resolution and subsampling to $2\,\mathrm{kHz}$, every single session has a memory footprint of $24.3\,\mathrm{MB} \pm 2.8\,\mathrm{MB}$. Therefore, storing even a single session could be critical for edge nodes with a small FLASH memory.

C. Temporal Convolutional Networks

Temporal Convolutional Networks (TCNs) are a promising sub-class of CNNs that have obtained outstanding results on benchmark time-series tasks [16], [17]. In addition to higher accuracy, TCNs have lower complexity than Recurrent Neural Networks (RNNs) [3]. Hence they represent a promising solution for edge deployment and time-series processing. A TCN contain many 1D convolutional layers, presenting two key characteristics:
(1) *causality*: each output y_{t_n} of the layer is computed by combining only elements x_{t_i} s.t. $t_i \leq t_n$, to not include future information in the intermediate time series through the network;
(2) *dilation*: to increase the Receptive Field (RF) without increasing the number of parameters, a fixed step d is applied between kernel weights; for instance, a filter with kernel size $k = 3$ and dilation $d = 4$ leads to RF $= (k-1)d + 1 = 9$. Thus, a 1D convolution layer produces an output as:

$$\mathbf{y}_n^o = \mathrm{Conv}(\mathbf{x}) = \sum_{l=1}^{L} \sum_{i=0}^{k-1} \mathbf{W}_i^{l,m} \mathbf{x}_{n-d,i}^l$$

with $\mathbf{x}$ input feature map and $\mathbf{y}$ output feature map, n the time index, $\mathbf{W}$ the filter weights, L the number of input channels, m the output channel, d dilation, and k the kernel size.

For this work, we exploit a similar topology to the one proposed in [5], as depicted in Figure 1. The network is composed of three sections:
- the *Features Extractor*, composed of 6 1D convolutional layers and 3 pooling layers; the structure is identical to the one shown in [5];
- the *Features Aggregator* includes 3 1D convolutions and 2 fully-connected layers, which mix the features extracted into a 256-dimension latent space;
- the final *Classifier*, two fully-connected layers, one 256×128 and one 128×9, which takes latent space as input and returns a 9-dimension score. The highest number is the assigned gesture. All layers have ReLU non-linearity as an activation function and are followed by Batch-Normalization (BN).

D. Incremental Learning applied to Gesture Recognition

Algorithm 1 Incremental Re-training protocol

Input: $\mathrm{TCN}_{\mathtt{int8}}$ weights [0:12], $\mathrm{TCN}_{\mathtt{fp32}}$ weights [6:12], stored latent data $\mathcal{D}_l$ from all seen sessions.
for $epochs \leftarrow 1, \cdots, 22$ **do**
  for $batch \leftarrow 1, \cdots, 244$ **do**
    forward: loss $\leftarrow$ $\mathrm{TCN}_{\mathtt{int8}}$[6:12][$\mathcal{D}_l[batch]$]
    backward: $\mathrm{TCN}_{\mathtt{fp32}}$[6:12] weights $\leftarrow$ loss
  end for
end for
$\mathrm{TCN}_{\mathtt{int8}}$[6:12] $\leftarrow$ linear_quant.($\mathrm{TCN}_{\mathtt{fp32}}$[6:12])
Output: $\mathrm{TCN}_{\mathtt{int8}}$[6:12]

Incremental Learning (or Continual Learning) has been introduced to address the challenge of learning new data distributions or patterns, updating a pre-existing model incrementally. The simplest way to tackle the problem is to periodically re-train a model on a dataset enriched with novel data. However, this approach poses a key challenge for deployment on constrained edge devices since training an entire network, running forward and backward propagation, is computationally demanding and requires maintaining a large amount of data stored on the device.

Therefore, literature proposes several novel approaches, such as [18], where the regularization is used to avoid forgetting the previously learned information. Another solution consists of storing in a non-volatile memory only a small amount of the new-gathered inputs (e.g., new images from a camera) to perform few-epochs of re-training of the network on all the saved data (Rehearsal Strategies) [19]. However, these methods either achieve low performance or necessitate large storage memories. For instance, [8] shows that, on the CORe50 benchmark, iCaRL [19] has 375 MB memory overhead, with an accuracy drop of up to 70%.

A promising approach for memory-constrained devices is based on the re-training of a network with *latent data* [8],

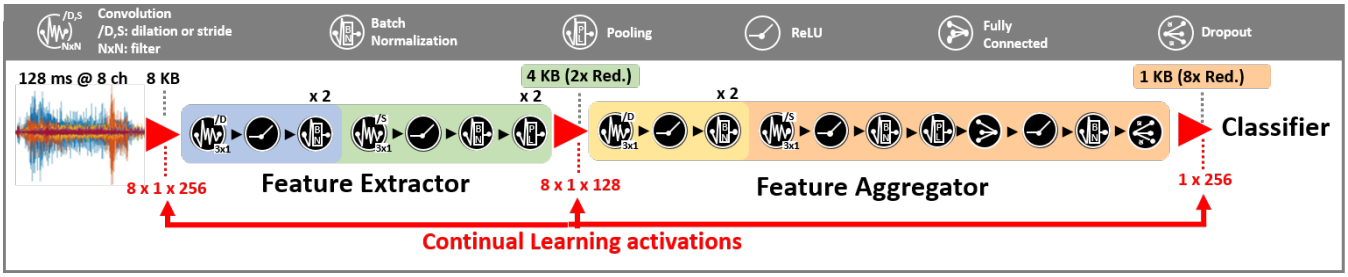

Fig. 1. Temporal Convolutional Network architecture. Three different configurations for feeding input / latent data for re-training over different sessions are highlighted with red arrows. Each sample has a memory footprint of 4 kB, 2 kB and 256 B, respectively, in the three different settings.

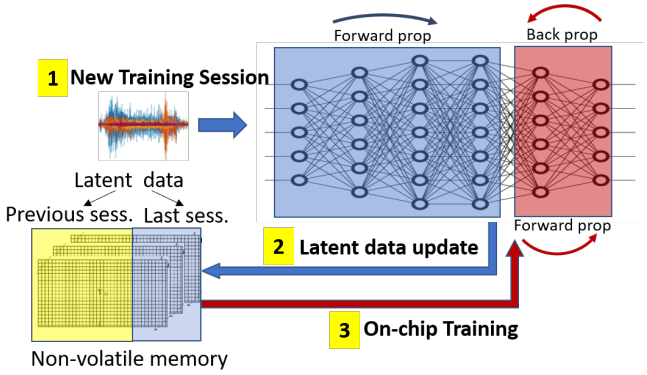

Fig. 2. Incremental Learning scenario with latent data stored on external non-volatile memory. At each new data acquisition session (1), a new re-training is performed: first, input data are fed-forward through the first network stage, which is frozen, and the outputs of this stage are stored as latent data (2); then, these latent data are used to re-train the second part of the network to adapt it to the new signal pattern (3).

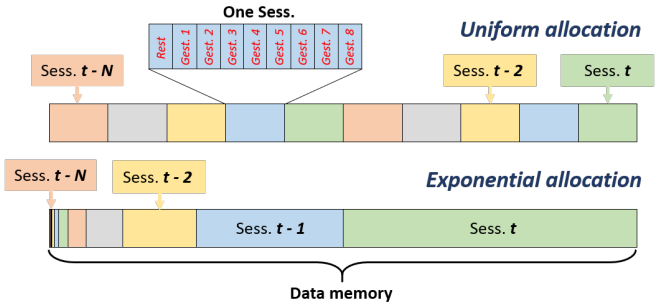

Fig. 3. Data storing for continual learning. Two approaches are analyzed, uniform and exponential decaying data distribution. In each single session, each class is equally represented by downsampling the rest class.

which are partial outputs coming from the forward propagation of initial layers of the network. As depicted in Fig. 2, latent data are intermediate activations produced a subset of the layers of a network. During a new training session, several *latent data* (labelled) are produced by calculating the output (forward prop) of the first N layers. Part of these *latent data* are stored in the local memory, partially replacing existing latent data stored from previous sessions.

Stored latent data are used to re-train the last layers of the network (i.e., from layer $N+1$), resulting in partial re-training of the neural network, updating only the last layers with the conventional forward/backward propagation, while freezing the first layers. Besides the advantage in terms of computational resources, continual learning also benefits reduced memory occupation of the *latent data* compared to raw signals. For instance, Fig. 1 shows that storing latent data produced by the last layer of the Feature Extractor reduces the data memory footprint by $2\times$, or equally allows storing $2\times$ more training data in the same memory footprint (if compared to raw inputs storage). Note that this is possible since the dimension of intermediate activations progressively shrinks through the network. While we have a network input dimension of 2048 elements, the input data dimension of the Features Aggregator is 1024 elements, and the one of the Classifier is just 256 elements.

In our experiments, we use an incremental learning strategy with latent data [8] to re-train our TCN on new recording sessions over time directly on the edge microcontroller to tackle the inter-session signal variability due to user experience and sensor repositioning. If not considered, EMG variability leads to a catastrophic performance loss of up to 30% [7]. This technique allows training on sessions acquired over time directly on a wearable device, avoiding having a static model whose performance degrades over time and streaming data to a cluster for re-training [9]. In Algorithm 1, we report the pseudocode of the TCN re-training procedure based on [8]. We re-train our network for 22 epochs, each with 244 mini-batches. Note that while we use the int8 format to reduce inference time, we store an fp32 copy of the model for re-training. At the end of this step, the new fp32 trained model is quantized, and both the fp32 (for future re-training) and the int8 (for inference) models are saved. In this example, we show the re-training of layers 6 to 12, i.e., the Feature Aggregator and the Classifier, using the latent data $\mathcal{D}_l$ (the intermediate activation produced by layer 5, obtained from all training sessions) stored in the data memory (64 MB flash memory of the board we target).

To the best of our knowledge, we are the first to analyze how re-training a neural network on the edge impacts gesture recognition accuracy over time.

E. Incremental Learning Policies

We show that our method can improve up to 20% a conventional 1-session training, and it reaches accuracy comparable with a conventional 10-session training strategy, tested on the UniBo-20-Sessions dataset. [5]. We test either the storage of raw data or the intermediate (latent) activations of the network. In particular, as shown in Figure 1, we explore the re-training of either *i*) the whole TCN, *ii*) the Feature Aggregator + Classifier, or *iii*) only the Classifier.

TABLE I

ACCURACY OF OUR TCN AND OF STATE-OF-THE-ART METHODS TRAINED WITH SINGLE SESSION (1^{ST}, 10^{TH}), ALL TRAINING SESSIONS SIMULTANEOUSLY (1^{ST}-TO-10^{TH}), AND OVER TIME (CONTINUAL).

	Recognition accuracy			
Train session(s):	**1st**	**10th**	**1st-to-10th**	**Incremental**
SVM [5]	**78.3%**	**88.1%**	91.4%	87.9%
HDC [10]	70.6%	81.0%	82.9%	82.9%
TCN (Our Work)	74.6%	81.2%	**94.8%**	**93.5%**

Note that to train only part of the network (i.e., only Features Aggregator+Classifier or only Classifier), all previous layers are frozen, thus performing back-propagation only on deepest blocks, as explained in [9]. Furthermore, we show two ways of storing data in a constrained memory: a *uniform allocation* and an *exponential allocation*, shown in Figure 3. With *uniform allocation*, an equal memory footprint is assigned to each training session, giving the same weight to all sessions. For instance, when a new session i is gathered, a portion $(i-1)/i$ of the allocated memory is preserved, while $1/i$ of the memory is assigned to the new session. In contrast, the *exponential allocation* increases the importance of most recent sessions: each newly added session receives $1/2$ of the data memory while discarding $1/2$ of the previously stored data. Both *uniform* and *exponential* allocation policies follow the same rule in discarding old samples: discarding step preserves the current ratio between stored sessions, i.e. each old session discards the same fraction of its own data. The uniform or exponential distribution is thus preserved at each new step.

Fig. 2 demonstrates our on-chip training procedure, inspired by [9]. On a new sEMG training session (1), we produce and store the new latent data in the data memory using one of the above-described policies, i.e. *exponential* or *uniform* allocation (2). After that, we perform the re-training of the second part of the network on the whole data stored in the data memory (3).

III. EXPERIMENTAL RESULTS

This section analyzes our TCN network's performance when trained over many sessions with different memory constraints. We then compare our TCN with SoA algorithms. Finally, we report the power and energy consumption of the network when deployed on a commercial edge-device, GAP8 [11].

A. Incremental Learning improvement

The last row of Table I shows our TCN accuracy under four training conditions. Note that all the models and the configurations are always tested on the last 10 data sessions, i.e., sessions 11-20. First, we show a single-session training. In the first column, we train only with 1^{st} gathered session, while in the second column with the 10^{th} session, i.e., the last gathered session before the testing ones. Then, we show a complete offline training with all the sessions from 1^{st} to 10^{th}, which represents the ideal case where all

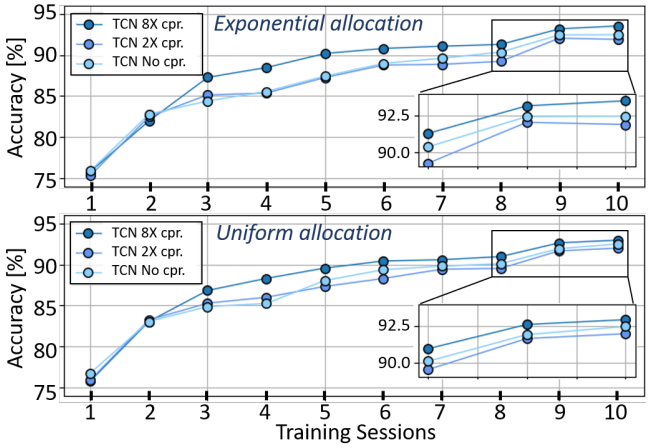

Fig. 4. Incremental training with sessions 1 to 10. We used 8 MB of data for re-training the algorithm. No cpr. (compression) refers to the re-training of the entire network, 2× cpr. to re-training of last two blocks, Features Aggregator and Classifier, and 8× cpr. to the re-training of solely the Classifier.

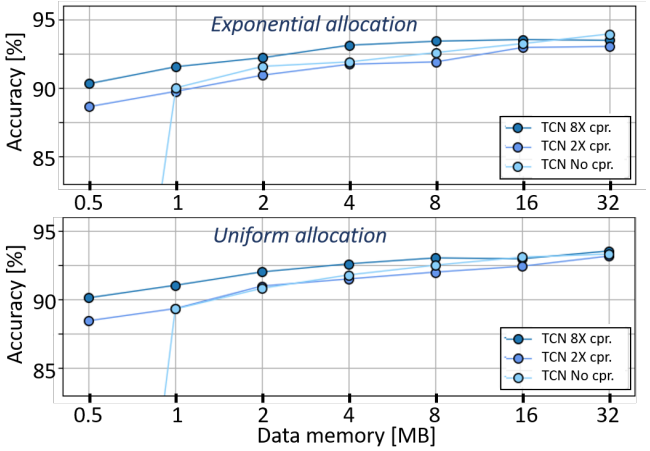

Fig. 5. Sweeping of data memory for TCN re-training. The data memory stores raw input/latent data from the first ten sessions of the dataset.

newly acquired data can be stored and used for network re-training. Finally, we present the re-training over sessions of our network. In particular, in Table I, we report the case of maximum data stored compression[2], with only the final *Classifier* re-trained. Compared to single-session training, the incremental learning approach improves the classification accuracy of 12.3%-18.9%. Remarkably, the higher is the time gap between training sessions and testing ones, the bigger is the accuracy loss. Further, compared to using all the 10 sessions simultaneously to train our network (unfeasible on an edge device, since storing all data requires $\sim$ 144 MB), the incremental approach causes a loss of only 1.3% accuracy (93.5% vs. 94.8%).

Fig. 4 shows the incremental training trend of our TCN when new sessions are added (from session 1 to 10). We report both exponential and uniform memory allocation. We also report the 3 scenarios of re-training previously introduced, i.e., re-training the whole network, re-training the Features Aggregator and the Classifier, or re-training only

[2]A memory of 8 MB was considered as a constraint for storing latent data.

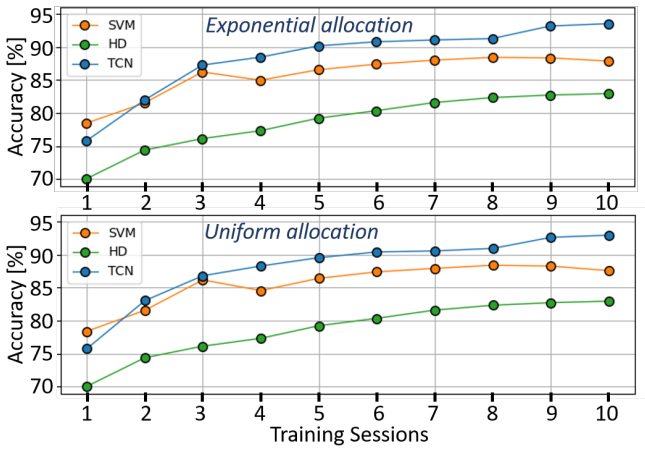

Fig. 6. Comparison of the best configuration of our TCN with SVM and HD classifiers.

the Classifier. For these three cases, for each single stored example, we either store the raw input data of dimension $8 \times 1 \times 256$, the Features Extractor output ($8 \times 1 \times 128$ latent data, thus with a $2\times$ compression compared to input), or the Features Aggregator output (1×256 latent data, $8\times$ compression), respectively, as shown in Fig. 1. Since our problem can be classified as a *new instances* problem [8] (i.e., new examples from the same classes are shown, changing the data distribution over time), giving more relevance to most recent sessions using exponential allocation results in slightly better performance, namely 93.5% vs. 92.6%. Moreover, the graphs show that re-training only the Classifier and thus storing latent data of dimension 1×256 (with compression of $8\times$ compared to input data) is always convenient.

Finally, Fig. 5 demonstrates our technique applied with different memory footprint constraints. Using just 2 MB, the exponential allocation, and re-training only the Classifier, we can already reach an accuracy of 93.2%. Notably, reducing the data memory leads to the re-training of only the Classifier to outperform the other two scenarios, with a higher performance gap of 2.4%-4.9%. Overall, re-training only the Classifier achieves the highest accuracy while reducing the data memory footprint by a factor of $8\times$ (less than $\sim 1\%$ loss using 1 MB memory instead of 8MB).

We compare our algorithm with two state-of-the-art classifiers for sEMG gesture recognition, RBF-SVM [6], and HDC [10] fed with the RMS feature, using the same four training configurations previously exposed. While HDC naturally learns new classes and new instances over time, for SVM, we consider the same memory allocation of our TCN, re-training the Classifier with the RMS data stored over sessions. We consider a data memory of 8 MB. The Tab.I shows that while using a single session for training leads our TCN to underperform compared to SoA, when we apply our incremental learning approach, we outperform SoA by 5.6% after training with 10 sessions. Further, Fig.6 shows that after having seen at least 3 training sessions, our TCN outperforms both SoA methods.

TABLE II

DEPLOYMENT METRICS OF OUR CONTINUAL LEARNING SETUP. THE ADDITIONAL 136 kB DURING TRAINING REPRESENT THE STORAGE OF FLOATING-POINT WEIGHTS.

	Data memory	**Net memory**	**Latency**	**Energy**
Training	$0.5 \div 8$ MB	$427 + 136$ kB	54.4 s	2.77 J
Inference	0 MB	427 kB	12.9 ms	0.66 mJ

On GAP8 [11] @ 100 MHz, 1 V (51 mW).

B. Edge deployment

We have deployed our solution on the GAP8 multicore platform [11], a microcontroller focused on AI edge-applications, to demonstrate its suitability for edge execution. GAP8 couples a fabric controller composed of one CV32E40P core for control tasks and 512 kB L2 memory extendable via a HyperBus interface with an 8-cores cluster accelerator for highly demanding workloads and 64 kB single-cycle latency L1 data memory. The clock frequency reaches up to 250 MHz, with an average power consumption of 51.0 mW at 100 MHz. After a post-training linear quantization, performed with the NEMO tool [20], to deploy our int8 quantized network, we use the DORY tool presented in [21] with an extension to the backend to support the dilation parameter.

Table II reports the memory footprint, the latency, and the energy per classification during inference. Our TCN only requires 427 kB for inference and an additional 136 kB to store fp32 weights of re-training layers, while the data memory occupies from 0.5 to 8 MB, depending on the chosen trade-off between data-memory and accuracy. The data are stored in the Flash, and the memory constraint depends on the used chip (for our tests, we test the algorithms on the GAPuino board with 64 MB flash memory). Running the TCN framework on GAP8 at 100 MHz, 1 V, the power envelope is SI51.0mW (most energy-efficient configuration), and we obtain 12.9 ms latency per inference, with an energy consumption of just 0.66 mJ.

Furthermore, based on the work of [9], considering 22 epochs for a full re-training with a minibatch size of 244 and Stochastic Gradient Descent (SGD), we estimate a total latency of 54.4 s with an energy consumption of 2.77 J, for a complete update of the network[3].

IV. CONCLUSIONS

We present a novel technique to tackle the time-variability in sEMG based gesture recognition. We combine Incremental Learning with a Temporal Convolutional Network to demonstrate an almost full recovering of performance compared to the "offline" training and an improvement of 18.9% compared to single-session training. We explore different memory allocation methodologies and different retraining network sections, achieving better performance than the state-of-the-art. Finally, we deploy our solution

[3]Both input data and intermediate activations are saved on 32 bits and the training is done in fp32.

on a commercial edge node, demonstrating its suitability for onboard gesture recognition. Future work will focus on retraining by using smaller formats than 32 bit fp.

REFERENCES

[1] R. Meattini *et al.*, "An semg-based human–robot interface for robotic hands using machine learning and synergies," *IEEE Trans. on CPMT*, 2018.

[2] T. Scott Saponas *et al.*, "Making muscle-computer interfaces more practical," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 851–854.

[3] J. L. Betthauser *et al.*, "Stable electromyographic sequence prediction during movement transitions using temporal convolutional networks," in *Int. Conf. NER*. IEEE, 2019.

[4] P. Tsinganos *et al.*, "Improved gesture recognition based on semg signals and tcn," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 1169–1173.

[5] M. Zanghieri *et al.*, "Robust real-time embedded emg recognition framework using temporal convolutional networks on a multicore iot processor," *IEEE transactions on biomedical circuits and systems*, vol. 14, no. 2, pp. 244–256, 2019.

[6] B. Milosevic *et al.*, "Exploring Arm Posture and Temporal Variability in Myoelectric Hand Gesture Recognition," *Int. Conf. on BioRob*, 2018.

[7] S. Benatti *et al.*, "Analysis of robust implementation of an emg pattern recognition based control," in *Int. Conf. BioSTEC*, 2014.

[8] L. Pellegrini *et al.*, "Latent replay for real-time continual learning (2019)," *arXiv preprint arXiv:1912.01100*, 2019.

[9] L. Ravaglia *et al.*, "Memory-latency-accuracy trade-offs for continual learning on a risc-v extreme-edge node," in *2020 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2020, pp. 1–6.

[10] S. Benatti, F. Montagna, V. Kartsch, A. Rahimi, D. Rossi, and L. Benini, "Online learning and classification of emg-based gestures on a parallel ultra-low power platform using hyperdimensional computing," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 3, pp. 516–528, 2019.

[11] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–4.

[12] C. J. De Luca *et al.*, "The use of surface electromyography in biomechanics," *Journal of applied biomechanics*, 1997.

[13] R. M. Rangayyan, *Biomed. signal analysis*. John Wiley & Sons, 2015.

[14] M. Tomasini *et al.*, "Power line interference removal for high-quality continuous biosignal monitoring with low-power wearable devices," *IEEE Sensors Journal*, vol. 16, no. 10, pp. 3887–3895, 2016.

[15] M. Zanghieri *et al.*, "Temporal variability analysis in semg hand grasp recognition using temporal convolutional networks," in *IEEE Int. Conf.AICAS*. IEEE, 2020.

[16] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018.

[17] C. Lea *et al.*, "Temporal convolutional networks for action segmentation and detection," in *IEEE Conf. CVPR*, 2017.

[18] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Reply to huszár: The elastic weight consolidation penalty is empirically valid," *Proceedings of the National Academy of Sciences*, vol. 115, no. 11, pp. E2498–E2498, 2018.

[19] S. Rebuffi *et al.*, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[20] F. Conti, "Technical Report: NEMO DNN Quantization for Deployment Model," University of Bologna, Tech. Rep., 2020.

[21] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus," 2020.