A Novel Learning Strategy for the Trade-off between Accuracy and Computational Cost: a Touch Modalities Classification Case Study

Christian Gianoglio, Edoardo Ragusa, Paolo Gastaldo, and Maurizio Valle, *Senior Member, IEEE*

***Abstract*—Wearable systems require resource-constrained embedded devices for the elaboration of the sensed data. These devices have to host energy-efficient artificial intelligence (AI) algorithms to output information to a human user. In this regard, the single-layer feed-forward neural networks (SLFNNs) proved to be very effective for deployment on very low-end devices. SLFNNs showed promising results when the goal was to balance the trade-off between accuracy and predictor complexity. Nevertheless, the elaboration system, which consists of data pre-processing and features extraction stages, affects the resource occupancy, computational cost, and also eventual generalization performance. In this paper, we propose a learning strategy based on the out-of-samples technique that leads to finding the configurations of the elaboration unit and the parameters of the predictor that best balance the generalization accuracy and the computational cost of the whole system. An intensive study over the touch modalities classification problem shows that the best solution in terms of the only accuracy is achieved by the SLFNNs, outperforming existing algorithms. Moreover, the learning procedure selects the configuration of the processing and features extraction stages, and the parameters of the predictor that present the best trade-off between generalization performance and computational cost.**

***Index Terms*—Tactile sensing systems, artificial intelligence for embedded devices, loss function.**

I. Introduction

With the growth of technology, many applications employ wearable electronics to detect and analyze signals from the body and the environment [1]. These systems use sensors and digital embedded units for data elaboration [2]. The embedded units process data received from the sensors and perform requested inference operations.

Machine learning algorithms set the basis for this advanced processing because the relation between sensors data and the inferred information cannot be modeled explicitly [3]. In the mean time, small-batteries supply wearable devices, hence the algorithms need to be computationally light. On the one hand, deep-learning paradigms offer high performance in terms of prediction accuracy; on the other hand, the hardware implementation of the associate decision functions does not meet the requirements set by resource limitation. Among AI algorithms, the single-layer feedforward neural networks (SLFNNs) proved appealing for the deployment on resources-constrained devices [4], [5].

Besides, the whole elaboration system consists of many stages: signals pre-processing, features extraction, and inference. Each stage affects resources occupancy, power consumption and accuracy of the prediction. In fact, changing these modules means changing the data space from which the predictor is learned. Hence, during the design of the elaboration system, it is crucial to keep in mind the hardware constraints for all the involved stages.

To optimize the trade-off between the classification accuracy and the computational cost of the elaboration process, we propose a data-driven learning procedure that seeks for the best balance. Usually, because the elaboration system must be deployed on an embedded device with hardware constraints, the choices of the signal processing and the features extraction modules are carried out by the designer of the system, and so they are a-priori fixed or however chosen between few candidates. Starting from an approach based on the out-of-sample technique [6], where a loss function is evaluated into the validation phase to find the best configuration of predictor parameters achieving the best generalization performance, we propose a learning strategy that through the tuning of a free parameter balances the trade-off between the generalization accuracy and the computational cost of the whole system. Indeed, the strategy can learn from data the predictor and the processing techniques keeping count of different processing strategies, predictor architectures, configurations of the predictor (i.e. the hyperparameters), and computational cost.

The learning procedure is quite general and leads to a simple, yet flexible method that can be easily applied to almost all the ML techniques in combination with different processing stages, when the out-of-sample technique is employed. The loss function includes one free parameter that accounts for the overall hardware constraints.

We applied the learning strategy to the design of the

Christian Gianoglio, Edoardo Ragusa, Paolo Gastaldo, and Maurizio Valle are with the Department of Electrical, Electronic, Telecommunications Engineering, and Naval Architecture (DITEN), University of Genoa, 16145 Genova, Italy (e-mail: {christian.gianoglio, edoardo.ragusa}@edu.unige.it; {paolo.gastaldo, maurizio.valle}@unige.it).

The authors acknowledge partial financial support from TACTIle feedback enriched virtual interaction through virtual realITY and beyond (TACTILITY) project: EU H2020, Topic ICT-25-2018-2020, RIA, Proposal ID 856718.

processing involved in touch modalities classification [3]. In this regard, we proposed and characterized a large set of pre-processing techniques, various feature extraction methods, and a set of classifiers. For each stage we discussed and tested the most appealing solutions existing in the literature, we proposed novel suitable solutions, and retrieved the computational cost as the number of floating operation (FLOPs) using the approximation proposed in [7].

An extensive experimental campaign on a real world dataset [3] confirms the potentialities of the proposed procedure. When only the accuracy is considered in the learning problem, single-layer feed-forward neural networks combined with a hand crafted features set outperform the state-of-the-art results. When the hardware constraints become preponderant, the learning procedure selects the solution that presents the best trade-off between generalization performance and computational cost.

Overall, the contribution of the paper can be summarized as follow:

- we propose a data-driven approach based on a novel loss function that, through the out-of-sample approach, finds the best configuration between pre-processing techniques, features extraction methods, and model architectures, balancing accuracy and the computational cost in terms of FLOPS;
- we validate our proposal on real world dataset, by tackling the touch modalities classification problem achieving the best accuracy with respect to state-of-the-art solutions by employing the random-based single-layer feed-forward neural networks with hand-crafted features;
- through the evaluation of the loss function, we also show the random-based networks present the best balance between generalization performance and computational cost.

The remainder of the paper is organized as follows: Section II presents the related works, Section III depicts the learning strategy, Section IV describes the touch modalities classification problem and the elaboration units, Section V reports the results and discussion, eventually Section VI concludes the paper.

II. Related Works

A. Machine Learning for the Edge

The literature abounds of works that address the challenge of deploying machine learning models on edge devices, applying learning strategies to trade-off the computational cost of the predictors and the inference accuracy. In [8], the authors reduced the number of weights of deep networks through low-rank approximation, with negligible loss in accuracy. In [9], the authors employed an iterative learning strategy to reduce the number of parameters in neural networks through a three-step procedure, where connections were learned via normal network training, then low-weight connections were pruned, and eventually the remaining connections were fine-tuned. In [10], the authors adopted different classifiers of increasing complexity that were chosen run-time based on the richness of details of the input sample. They saved energy, keeping the accuracy comparable with baseline solutions. Similarly, in [11], the authors proposed a strategy to design classifiers of increasing complexity using pre-trained deep networks. The accuracy and energy consumption depended on a set of thresholds chosen at run-time, based on the desired trade-off. Other works [12], [13] employed the teacher-student learning paradigm for compression. Instead of reducing a large model into a smaller one, these methods start with the smaller model increasing step-by-step its complexity until a proper generalization accuracy is achieved, as the student who tries to learn from the large model as his teacher. Moreover, in these works the student learned from both the teacher and data. In [14], the authors reduced the computational load of deep networks by constraining the weights and activations to be digital. Some other works can be found in the literature proposing other strategies to reduce the computational cost with a low drop in accuracy [15]–[18].

The majority of the mentioned works did not focus on the optimization of a loss function that keeps count of the computational cost during training, but they sought to reduce the complexity of bigger models with pruning, fixed-representation, thresholds, and iterative techniques. However, we present a learning strategy based on the out-of-sample technique that can be employed with any model, also keeping count the pre-processing stages that affect both accuracy and computational cost. Similarly to our work, [19] proposed an optimal policy based on a cost function to find the best trade-off between performance and power consumption on a combination of different classification algorithms employed on state-of-the-art datasets. The main difference with our proposal is that [19] designed a loss function to find the best accuracy combining several classifiers, and penalizing the solutions that contain more models and with the higher power consumptions. In our case, the number of elaboration stages is fixed containing only one algorithm per stage, so we don't penalize the number of algorithms but we reward the solutions that are energy efficient.

B. Machine Learning for Tactile Sensing

In the last years, tactile sensing has become widely adopted in robotics and prosthetics both to restore the human's lost sense of touch and for the robot objects manipulation. In [3], the authors employed tensor-SVM and tensor-RLS algorithms to classify three touch modalities collected from piezoelectric sensors as tensor data. In [20], the authors adopted deep networks and transfer learning on the same dataset, transforming the tensor data into RGB images. Recently, in [21] the recurrent neural networks (RNNs) have been exploited to address the same problem showing an improvement in terms of accuracy and a reduction of the predictors complexity. The RNNs were able to capt the dependencies between time samples and easily deal with 3D tensor data. In [22], the authors classified 4 touch pattern categories, collected on 3x3 sensors array and integrated with accelerometer information, with a decision tree predictor. In [23], the LogitBoost [24] algorithm was used to classify six touch modalities applied to an artificial sensitive skin based on electrical impedance tomography. In [25], nine touch modalities collected on a humanoid covered with an artificial skin were classified through the SVM classifier. Moreover, the authors over-performed the [23] results with a properly choice of the features set. Between these works, only one [21] explicitly tackled the hardware implementation problem proposing a solution based on deep learning. Some other works employed the single-layer feed-forward neural networks (SLFNNs) to process tactile data, setting random the parameters of the hidden neurons to lead a faster training procedure. These SLFNNs are known as random-based networks. These algorithms presented a valuable trade-off between computational cost and generalization performance [26]–[28]. Moreover, random-based networks solutions optimized for the resource-constrained devices have been proposed [4], [5], [29], also adopting strategies to reduce the computational cost, increasing the generalization capabilities [30]–[36]. In [37], the authors compared the extreme learning machine (ELM), the regularized least square, and the SVM algorithms in classifying object materials from raw tactile data. In [38], the authors developed an extreme kernel sparse learning method to classify objects based on 21 features extracted from tactile datasets. The methodology was based on extreme learning machine paradigm and kernel sparse learning. In [39], the authors designed a spiking network based on the ELM to classify different textures. They also addressed the trade-off between classification accuracy and energy consumption.

III. Methodology

In general, let consider a classification problem with an input space $\mathcal{X} \in \mathbb{R}^d$ and an output space $\mathcal{Y} \in \{1, ..., N\}$ where N represents the number of the classes. The dataset $D_n = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n),\ \boldsymbol{x} \in \mathcal{X},\ y \in \mathcal{Y}\}$ collects n independent and identically distributed (i.i.d.) data from an unknown probability distribution P. Our goal is to build a classifier or, in other words, to construct a function $f : \mathcal{X} \longrightarrow \mathcal{Y}$, which predicts $\mathcal{Y}$ from $\mathcal{X}$. To choose f, we measure the expected error, i.e. the risk $L(f)$, performed by the function on the entire data population $(\mathcal{X}, \mathcal{Y})$, according to the statistical learning theory [40]:

$$L(f) = E_{(\mathcal{X},\mathcal{Y})}\left[l(f(\boldsymbol{x}), y)\right] \tag{1}$$

where E is the expected value computed on the population $(\mathcal{X}, \mathcal{Y})$, and $l(f(\boldsymbol{x}), y)$ is a suitable loss function that measures the discrepancy between the prediction of f and the true $\mathcal{Y}$, according to some user-defined criteria. Let us consider a class of functions $\mathcal{F}$, thus the optimal classifier $f^* \in \mathcal{F}$ is

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}}\, L(f). \tag{2}$$

Since P is unknown, we cannot directly evaluate the risk, nor find f^*. The only available option is to depict a method for selecting $\mathcal{F}$, and consequently f^*, based on the available data and, eventually, some a priori knowledge.

The empirical risk minimization approach suggests the estimation of the true risk $L(f)$ by its empirical version computed over D_n:

$$\hat{L}_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(\boldsymbol{x}_i), y) \tag{3}$$

so that

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}}\, \hat{L}_n(f). \tag{4}$$

Unfortunately, $\hat{L}_n(f)$ typically underestimates $L(f)$ and can lead to severe overfitting because, if the class of functions $\mathcal{F}$ is sufficiently large, it is always possible to find a function that perfectly fits the data but shows poor generalization capability. For this reason, it is necessary to add a regularizer over f that avoid overfitting increasing the generalization performance of the classifier, resulting in:

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}}\, \hat{L}_n(f) + \lambda R(f) \tag{5}$$

where λ is an hyperparameter and R the regularizer (e.g., lasso, ridge, elastic net).

Now, if the goal is to solve (5), the problem is defining the $\mathcal{F}$ space. In general, one should take into account the role of hyperparameters, i.e., the parameters that determine the model structure (e.g., the number of neurons in a neural network); they affect, directly or indirectly, the function class $\mathcal{F}$ where the learning algorithm searches for the, possibly optimal, function [40]. Thus, the $\mathcal{F}$ space actually involves several function classes $\mathcal{F}_1, \mathcal{F}_2, ...$, which differ for the different setting of the hyperparameters. In the statistical learning theory, the out-of-sample approach [6] suggests the use of an independent dataset (validation set), sampled from the same data distribution that generated the training set, that leads to the evaluation of generalization performance by varying the hyperparameters of the classifier. Given additional m samples $D_m = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\}$ (i.e. the validation set) and given a classifier f_n, which has been trained on D_n (i.e. the training set), considering the several function classes $\mathcal{F}_i$ indexed by different values of the hyperparameters, then the optimal classifier $f^*_n = f^*_{n,i^*}$ is the result of the following minimization process:

$$\begin{aligned} & i^* = \underset{i}{\operatorname{argmin}}\, \hat{L}_m(f^*_{n,i}) \\ & \text{where} \\ & f^*_{n,i} = \underset{f \in \mathcal{F}_i}{\operatorname{argmin}}\, \hat{L}_n(f) + \lambda R(f) \end{aligned} \tag{6}$$

where the $\mathcal{F}_i$ space is user-defined, based on a priori knowledge. If we are interested in estimating the generalization error of f^*_n, we need to compute again the cost (3), but using some data (i.e., the test set) that has not been involved in this procedure.

As presented, the out-of-sample technique leads to find the best predictor between a possible pool of candidates $\mathcal{F}_i$, with several configuration of the hyperparameters. In practice, the validation set gives the possibility to compare different predictor configurations, looking for the one that presents the lowest generalization error. In addition, the validation set could be also used to find the best processing modules that generate different P spaces from which the data are sampled, as mentioned at the beginning of the section. In fact, our proposal is to include the processing stages in the learning problem (5) as hyperparameters, solving (6) to find best configuration of the predictor architecture and the processing units. Because we want to deploy the elaboration system on an embedded device, it is crucial to take into consideration the computational cost of the elaboration system during the learning phase, looking for a solution that balances the generalization performance and the computational cost. As stated in the previous paragraph, it is possible to include the processing stages in the learning phase as hyperparameters because they affect the generalization performance of the predictor. Moreover, one can keep count of the computational cost by adding a regularization term in (5) that penalizes burdensome solutions from the hardware point of view. The new problem could be formalized as:

$$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{L}_n(f) + \lambda R(f) + \theta R_H(f) \tag{7}$$

where θ is a hyperparameter that weights the hardware constraint R_H. As λ balances the regularization term that prevents the overfitting, θ trade-offs the accuracy of the predictor and its computational cost represented by the R_H function. R_H is user-defined and could represent the latency from the data acquisition until the prediction, the elaboration system resources occupancy, the number of FLOPs required to get a prediction, etc. Adding the new regularizer, we complicate the learning problem, increasing the learning cost: each time that a new R_H is evaluated, it is necessary to repeat the whole learning procedure. Moreover, (7) presents three terms that are conflicting during the learning, making the problem approachable by a Pareto-based multi-objective learning [41], where the objective function is a vector. Nevertheless, fixing the parameter that weights the hardware constraints on the designer-needs leads to simplify the learning procedure, through the scalarization of the objective function [41]. Thus, similarly to the rationale that leads to solve (5) through the out-of-sample technique, we propose to employ the validation set to find the best pool of the hyperparameters, evaluating the generalization performance as in (6) adding the computational cost R_H, fixing θ as the user-needs. In this way, we can find the best configuration of the hyperparameters as the sum of two terms: the first that estimates the generalization capability and the second that measures the computational cost. This resulting in:

$$i^* = \operatorname*{argmin}_{i} \hat{L}_m(f^*_{n,i}) + \theta R_H(f^*_{n,i}) \tag{8}$$

where

$$f^*_{n,i} = \operatorname*{argmin}_{f \in \mathcal{F}_i} \hat{L}_n(f) + \lambda R(f). \tag{9}$$

Procedure 1 proposes our data-driven learning method that allows to train the predictor f, keeping count the hardware constraints.

Procedure 1 Learning Procedure

Input
- Training set D_n
- Validation set D_m
- Hyper-parameters pool $\mathcal{HP}$
- User-defined θ

1. **Methodology**

```
1: for each $i$ in $\mathcal{HP}$ do
2:     Solve (9) with $i$ and $D_n$
3: end for
4: Use $D_m$ to solve (8)
```

2. **Output** Return f^*_{n,i^*}

IV. Case Study

To validate the proposed learning strategy introduced in the previous section, we addressed the touch modalities classification problem [3], where tactile sensors acquire signals that will be processed and classified by a resources constrained device. The figure in the graphical abstract shows the elaboration stages. Three stages contribute to the performance of the elaboration system both in terms of generalization accuracy and computational cost: signal pre-processing, feature extraction, and prediction. In this case study, R_H assesses the computational cost expressed in terms of FLOPs. Procedure 1 is employed taking into account different combinations of pre-processing approaches, features extraction methods, and predictor setup. In the following, we describe in details the single stages of the elaboration system and their computational cost in terms of FLOPs.

A. Computational Cost as FLOPs

The computational cost of each stage has been assessed under the hypothesis that the classification system is deployed on a microprocessor with floating point unit (FPU). Table I reports the cost in term of FLOPs of the most frequently used operations and functions.

TABLE I
FLOPs Count for most frequently used Operations [7]

Operation	FLOP Count
Add, Subtract, Multiplication Isolated Assignment, Type Conversion, Comparison	1
Division, Square Root	4
Sine, Exponential	8

According to [7], the number of floating point operations executed is often architecture-independent. Thus, the table assigns a unitary value ($FLOPCount = 1$) to a set of basic floating operations. The division and the square root require four time the unitary value, while the sine and the exponential require eight time the unitary value. Obviously, one may also refer to a different setup for the FLOPs assigned to the different operations, in particular when targeting specific computer architectures.

B. Dataset

The dataset, publicly available at http://www.sealab.diten.unige.it, has been collected on three touch modalities [3], i.e. slide a finger, brush a paintbrush, and rolling a washer. The dataset is made of 840 tensor data, 280 per class. Seventy subjects performed each touch action two times on a 4x4 piezoelectric sensors matrix in two directions, i.e. horizontal and vertical starting from a random position. Each touch lasted 10 seconds and signals have been sampled at $3kHz$. Each datum is a 3-mode tensor where the first two modes represent the 4x4 sensors geometry (i.e. a total of 16 channels) and the third dimension contains the time samples ($10s \times 3KSamples/s = 30000$ $samples$ per each channel). The dataset is formalized as: $\mathcal{D}_{Raw} = \{(\mathcal{X}, y)_i; \mathcal{X}_i \in \mathbb{R}^{4\times4\times30000}; y \in \{Slide, Brush, Roll\}; i = 1, ..., 840\}$. Figure 1 sketches the three touch modalities and an example of an acquired signal.

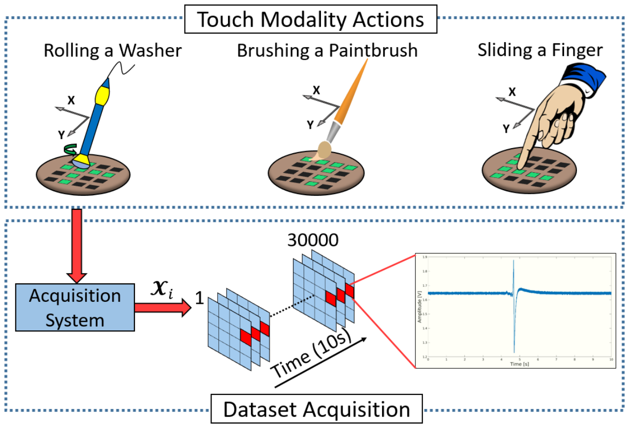

Fig. 1. Three modality actions and an example of acquired data. In the top part, the three actions performed on the matrix of sensors are shown; in the bottom, the acquisition system acquires a tensor datum where each taxel represents a sensor and the third dimension the signals samples (red squares).

C. Signal pre-processing

We adopted two strategies to pre-process the acquired signals. The first one was presented in [42] and consists in the reduction of the number of signal samples from $n = 30000$ to $\hat{n} = 6144$ in each tensor channel, checking at which sample each signal exceeds a predefined threshold. Then, the signals were averaged with 50% of overlapped samples to downsample the tensor from $4 \times 4 \times 6144$ to $4 \times 4 \times Ns$, where Ns was chosen as $Ns = \{8, 16, 32, 64\}$ representing the number of slices in the time dimension, i.e. the number of two-dimensional fragment obtained by fixing all indices but two (in this case, we are referring to the slices having dimensions 4×4, with the time dimension index fixed) . In this way, from the dataset $\mathcal{D}_{Raw}$, we created 4 new datasets that we called $\mathcal{D}_8$, $\mathcal{D}_{16}$, $\mathcal{D}_{32}$, and $\mathcal{D}_{64}$, based on the number of slices Ns (in the following we will refer to these datasets with the subscript Ns).

The second strategy concerns different processing techniques based on state-of-the-art filtering methods [43]. Because the acquired signals had a DC component different from 0V, we normalized the data in $\mathcal{D}_{Raw}$ to have 0 mean value. Let be $\mathcal{D}$ the normalized dataset. Furthermore, as each touch lasted 10s but the informative part is around the peak as shown in Figure 1, we reduced through an automatic method the number of samples of each signal by taking the relevant part (RP), i.e.

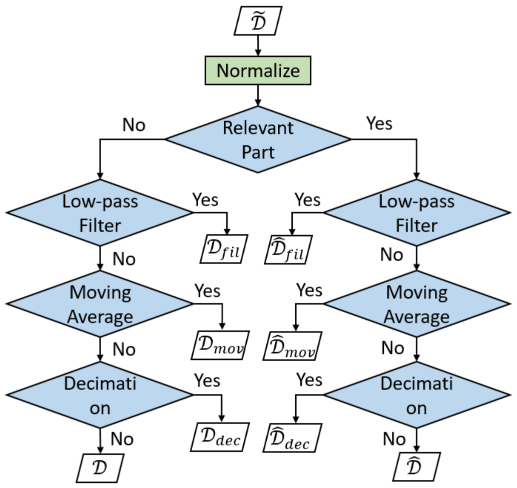

Fig. 2. Flow chart showing the datasets generated after the processing and noise filtering techniques.

the part that presents the greatest energy content. The resulting signals were stored in a new dataset $\hat{\mathcal{D}}$. Procedure 2 depicts the steps to extract the signals relevant part.

Procedure 2 Extract the Signals Relevant Part

```
Input
  • Data set $\mathcal{D}$
  • Number of bins $nbins$
  • Number of samples $nSam$ of the resulting signals
1. Methodology
1: for all signals in $\mathcal{D}$ do
2:     Divide the signal in $nbins$ parts
3:     Compute the energy in each bin as $\sum_{i=0}^{nSbin} x_i^2$, where $nSbin$ is the number of samples in each bin
4:     Select the bin containing the maximum energy
5:     Take $nSam/2$ samples before and after the first sample of the chosen bin
6:     Insert the resulting signal in $\hat{\mathcal{D}}$
7: end for
2. Output Return $\hat{\mathcal{D}}$
```

We employed three processing techniques to filter out noise on both datasets $\mathcal{D}$ and $\hat{\mathcal{D}}$: 1) the first approach was based on a low-pass Chebychev FIR filter of order $N_{FIR} = 50$, with a cutoff frequency of 10Hz; 2) the second method employed the moving average, smoothing data with a gaussian function over a window of $L = 100$ samples at time; 3) the third technique adopted the decimation process, reducing the sample rate of the signals by a factor (in our case the factor was $F = 30$). Before decimating, a low-pass Chebychev IIR filter of order $N_{IIR} = 8$, with a cut frequency proportional to F, smoothed the data. Figure 2 shows the generated datasets with the second strategy based on filtering methods.

Pre-processing FLOPs

Table II reports the pre-processing FLOPs count for each signal with respect to the number of samples n. The first column represents the pre-processing technique, the second one the number of FLOPs for each method. The rows are divided in two main parts according to the strategies presented in the previous section: the first one (i.e., **First Strategy**) refers to the techniques presented in [42], while the second (i.e., **Second Strategy**) to the filtering based techniques [43].

Actually, the normalization (i.e. $\boldsymbol{x} - \overline{x}$) requires a division to compute the signal mean value but, because the number of samples is equal for each signal, it is possible to compute the term $1/n$ only once, store it in the device memory, and perform the division like a multiplication. The rationale holds also for the computation of the moving average where the length L of the window is a-priori known.

TABLE II
PRE-PROCESSING TECHNIQUES CONSUMPTION IN TERMS OF FLOPS

Pre-Processing Technique	FLOPs Count
First Strategy	
Thresholding	$n/2 + 3$
Average 50%	$(Ns - 2)(2n/Ns + 1) + 2(n/Ns + 1)$
Second Strategy	
Normalization	$2n + 1$
Relevant part	$2n + nbins + 1$
Low-pass filter	$2n * (N_{FIR} + 1)$
Moving Average	$n * (2L + 1)$
Decimation	$4n * (N_{IIR} + 1)$

D. Features Extraction

To find a good representation of data that feed the classifiers in the next stage, we extracted features from the processed signals. In [44], the authors described a large amount of useful techniques to extract features from signals in time. Because the feature extraction stage is hosted by a low power and wearable device it is important to find significant features that have a low impact on the overall power computation. Among the huge pool of possible candidates, we considered those features that require a limited number of FLOPs to be computed. Thus, we obtained three sets of features related to the statistical moments (SM) (i.e. median, variance, skewness, and kurtosis), stationary points and energy (SPE) (i.e. maximum, minimum, maximum-minimum distance, and energy), and inter-samples differences (ISD) (i.e. mean absolute deviation, median absolute deviation, mean of differences between adjacent samples, and mean of absolute differences between adjacent samples). Eventually, we created new datasets over all the possible combinations of the features sets, computed from the datasets generated by two pre-processing strategies described in Section IV-C. From the first pre-processing strategy, we obtained 28 datasets: 7 combinations of the features sets over the 4 datasets $\mathcal{D}_{Ns},\ Ns = \{8, 16, 32, 64\}$. For the second strategy, we obtained 56 datasets: 7 combinations of the features sets over the 8 datasets $\mathcal{D}_{fil}, \mathcal{D}_{mov}, \mathcal{D}_{dec}, \mathcal{D}, \hat{\mathcal{D}}_{fil}, \hat{\mathcal{D}}_{mov}, \hat{\mathcal{D}}_{dec}, \hat{\mathcal{D}}$. Table III reports all the generated datasets from the features extraction stage. The first column is the starting dataset $\mathcal{D}_{Raw}$. The second column represents the processing techniques [21], while the third and the fourth columns show the second strategy, i.e., if the relevant part (Rel) has been computed and the applied filtering methods (Filt). According to Section IV-C and Figure 2 in particular, the filtering methods are the low-pass Chebychev FIR filter (LPF), the moving average (Mov), the decimation (Dec), and no-processing (No Pr). The 5-th column summarizes the dataset generated after the processing stage. The 6-th column asserts if some features were extracted. The last column shows the datasets names. The datasets names followed this rationale: we used $\mathcal{F}$ to indicate that the features have been extracted, we kept the subscripts that refer to the processing techniques, and we added the names of the adopted features sets as superscript. As examples, $\hat{\mathcal{F}}^{SM,SPE}_{Dec}$ means that we extracted the SM and SME features from the dataset $\hat{\mathcal{D}}_{Dec}$, and $\mathcal{F}^{All}_{16}$ means that we extracted all the features from the dataset D_{16}. For space reasons, we do not show all the possible combinations of features, i.e. $Feat = \{SM,\ ISD,\ SPE,\ SM{+}ISD,\ SM + SPE,\ ISD + SPE,\ All\}$, neither the possible number of slices $Ns = \{8,\ 16,\ 32,\ 64\}$.

TABLE III
GENERATED DATASET

Input	Processing Techniques			Proc. Dat.	Feat. Ext.	Output
	[21]	Rel	Filt			
$\mathcal{D}_{Raw}$	✓			$\mathcal{D}_{Ns}$		$\mathcal{D}_{Ns}$
	✓			$\mathcal{D}_{Ns}$	✓	$\mathcal{F}^{Feat}_{Ns}$
			No Pr	$\mathcal{D}$	✓	$\mathcal{F}^{Feat}_{NoP}$
			LPF	$\mathcal{D}_{fil}$	✓	$\mathcal{F}^{Feat}_{LPF}$
			Mov	$\mathcal{D}_{mov}$	✓	$\mathcal{F}^{Feat}_{Mov}$
			Dec	$\mathcal{D}_{dec}$	✓	$\mathcal{F}^{Feat}_{Dec}$
		✓	No Pr	$\hat{\mathcal{D}}$	✓	$\hat{\mathcal{F}}^{Feat}_{NoP}$
		✓	LPF	$\hat{\mathcal{D}}_{fil}$	✓	$\hat{\mathcal{F}}^{Feat}_{LPF}$
		✓	Mov	$\hat{\mathcal{D}}_{mov}$	✓	$\hat{\mathcal{F}}^{Feat}_{Mov}$
		✓	Dec	$\hat{\mathcal{D}}_{dec}$	✓	$\hat{\mathcal{F}}^{Feat}_{Dec}$

As mentioned above, the datasets contains 3-mode tensor data. Thus, each touch modality ($\mathcal{X}_i$) actually contains 16 signals. Applying feature extraction techniques to each signal, we obtained 3-mode tensor having dimensions $4 \times 4 \times nfeat$, where $nfeat$ is the number of extracted features. Therefore, to accomplish the neural networks input layer structure, it was necessary reshape the tensor data. Indeed, we unrolled the tensors along the sensors taxels obtaining a vector for each tensor datum, i.e. $\mathcal{F} \longrightarrow \mathcal{F}' = \{(\boldsymbol{x}, y)_i, \boldsymbol{x} \in \mathbb{R}^{16*nfeat}, y \in \{Slide, Brush, Roll\}; i = 1, ..., 840\}$, where 16 are the number of sensors in the array. Figure 3 sketches an example of data reshaping after features extraction.

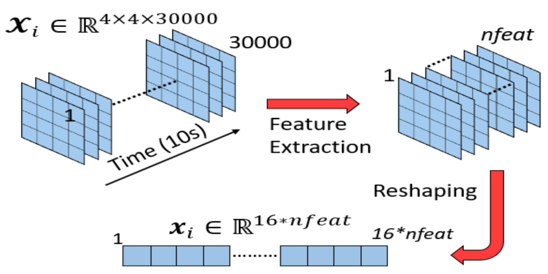
$\mathcal{X}_i \in \mathbb{R}^{4\times4\times30000}$
30000
1
Time (10s)
Feature Extraction
nfeat
1
Reshaping
$\boldsymbol{x}_i \in \mathbb{R}^{16*nfeat}$
1
16*nfeat

Fig. 3. Example of data reshaping.

Features Extraction FLOPs

Table IV summarizes the extracted features, and the computational cost in terms of FLOPs. To clarify, n refers to the current number of signals samples, e.g. after the relevant part detection and the decimation technique the number of samples is lower than in $\tilde{\mathcal{D}}$ as in the case of the thresholding processing. The median was computed with the median of medians algorithm requiring only $4n$ FLOPs. Nonetheless, because of the normalization, the computational cost of the mean was not considered. Moreover, sometimes it was possible to employ partial results of a feature to compute others features saving FLOPs, e.g. in the case of SM features set we reused some partial results (e.g. variance $tmp_{var} = x_i * x_i$) to compute the other features (i.e. skewness $tmp_{sk} = tmp_{var} * x_i$, kurtosis $tmp_{ku} = tmp_{sk} * x_i$), reducing the number of FLOPs. In addition, as for the implementation of the pre-processing techniques, some features require a division for the number of signal samples (e.g. variance, skewness, and kurtosis) that could be realized by a multiplication if the number of signals samples n was previously stored in the memory as a constant.

TABLE IV
EXTRACTED FEATURES PER FEATURE SET AND FLOPS COUNT

Features Sets	Extracted Features	FLOPs Count
SM	Median	$4n$
	Variance	$2n+1$
	Skewness	$2n+10$
	Kurtosis	$2n+6$
SPE	Energy	$2n$
	Maximum	n
	Minimum	n
	Max-Min Dist.	1
ISD	Mean Abs. Dev.	$2n+1$
	Med. Abs. Dev.	$10n$
	Mean of Diff.	$2n+1$
	Mean Abs. Diff.	$2n+1$

E. Predictors

We trained four different AI algorithms with the datasets generated by the features extraction stage. Two of them are based on deep learning, i.e. long short term memory (LSTM) and gated recurrent unit (GRU) neural networks, and are well described in [21]. The others present one hidden layer to keep low their computational cost and are known as single-layer feed-forward neural networks (SLFNNs).

In this paper, among all the random-based SLFNNs, we adopted the ones that exploit the extreme learning machine (ELM) paradigm. In the Appendix, we briefly describe the ELM predictor. The first network is the standard ELM paradigm (S-ELM) based on the sigmoid activation function (i.e., $\phi = sigmoid$ in (14)). The second SLFNN is based on the ELM but with the hard-limit activation function (H-ELM) and the weights represented as power of two (i.e., $\phi = sign$ and $w_{dn} = s_{dn} * 2^{-k_{dn}}$ in (14), where s_{dn} is the sign and k_{dn} is an integer number). This simplification proved to be very effective on low-cost and low-power devices [4], [5].

Predictors FLOPs

In this section, we introduce the cost of the predictors in terms of FLOPs. The cost depends on the classifiers architecture as the number of hidden neurons; we expressed the FLOPs costs through the architectures parameters. The FLOPs cost of the S-ELM is:

$$FLOPs_{S-ELM} = N * (2D + 14) + 2c * N \qquad (10)$$

where N is the number of neurons, c is the number of the classes (in case of the bi-class problem $c = 1$), and $D = 16 * nfeat$ is the number of features as reported in Figure 3. Considering the predictor (14), with $\phi = sigmoid$ (that takes 13 FLOPS in Table I), the cost of each neuron $\phi(\boldsymbol{x} \cdot \boldsymbol{\omega_n} + b_n)$ is $2D + 14$. The FLOPs cost of the H-ELM is:

$$FLOPs_{H-ELM} = N * (D + 1) + c * N. \qquad (11)$$

With $\phi = sign$, the cost of each neuron is $D + 1$. The difference with the neuron cost of the S-ELM is due: a) the activation functions, in fact the $sign$ function leads to a simple change of sign, with respect to the sigmoid that requires a division and an exponential calculation; b) in [4], [5] the authors demonstrated that representing the data as signed integer and weights as power of two it is possible to replace multipliers with shifts (with a negligible cost), with a negligible loss in the eventual accuracy with respect to the floating point representation. It is worth stressing that the H-ELM has been designed for implementation on FPGAs. Here, to propose a fair comparison with the other algorithms, (11) assesses the computational cost of H-ELM in terms of FLOPs, i.e., when considering the deployment of the network on a microprocessor that hosts an FPU. The FLOPs cost of the LSTM is:

$$\begin{aligned} FLOPs_{LSTM} = {} & Ns * N(8N + 8D + 12+ \\ & + 3sigm + 2tanh) + 2N * c \end{aligned} \qquad (12)$$

where Ns are the time slices, D is the number of input features ($D = 16$ as the number of sensors), and N is the number of hidden units. The sigmoid $sigm$ and the hyperbolic tangent $tanh$ require 13 and 16 FLOPs respectively, according to Table I. The FLOPs cost of the GRU is:

$$\begin{aligned} FLOPs_{GRU} = {} & Ns * N(7N + 6D + 19+ \\ & + 2sigm + tanh) + 2N * c. \end{aligned} \qquad (13)$$

A brief description of the units of the two deep neural networks is reported in the Appendix.

Eventually, Table V summarizes the cost in term of FLOPs of each predictor for the 3-class problem.

TABLE V
FLOPS COST OF 3-CLASS PREDICTORS

Classif. Algor.	FLOPs Cost
H-ELM	$N * (D + 4)$
S-ELM	$N * (2D + 20)$
LSTM	$Ns * N(8N + 8D + 12 + 3sigm + 2tanh) + 2N * c$
GRU	$Ns * N(7N + 6D + 19 + 2sigm + tanh) + 2N * c$

Predictors Training Setup

The classifiers were implemented to deal with a 3-class classification problem, with three output neurons, i.e. one for each class, and a softmax function to assign the label (i.e. $Slide$, $Brush$, $Roll$) to the neuron having the highest probability. For the learning procedure, we defined the hyper-parameters of the AI algorithms.

For the H-ELM and S-ELM neural networks:

- number of maximum hidden layer neurons $N = \{50, 100, 200, 500, 1000\}$;
- random weights for H-ELM $w_{dn} = s_{dn} * 2^{-k_{dn}}$ with $k \in [1, 8]$;
- random weights for S-ELM in $(-1, +1)$
- random biases for H-ELM as [4];
- random biases for S-ELM in $(-1, +1)$;
- regularization term (15): $\lambda = \{2^i, i = -10, -9, ..., 10\}$.

For the LSTM and GRU neural networks:

- number of hidden neurons $N = \{10, 25, 50, 100\}$;
- batch size $bs = \{48, 96\}$ as in [21];
- number of epochs $ep = \{48, 96\}$ as in [21]

The SLFNNs and the recurrent networks (GRU and LSTM) have different configurations of N because of the different structure of the eventual predictor; in fact, recurrent networks may face the risk of overfitting when $N > 100$. According to Procedure 1, we also defined a pool of candidates for the θ parameter as $\theta = \{0, 0.5, 1, 2, 5, 10\}$. For all the classifiers, each experiment involved 10 runs, i.e. 10 random training/validation/test sets; 70% of patterns were used as a training set, 15% as validation set for the model selection (i.e. to choose the best N and λ), and 15% for the test set. The generalization accuracy of each run was assessed by averaging over 10 extractions of the random parameters for H-ELM and S-ELM classifiers, and 10 starting values of the weights for LSTM and GRU. The H-ELM and S-ELM predictors were trained over 88 datasets (i.e. all the datasets presented in Table III), while the LSTM and GRU over 4 datasets (i.e. the $\mathcal{D}_{Ns}$ in Table III).

V. Results and Discussion

In this section, we first report the results about the generalization performance of the four predictors computed over the combinations of the processing strategies and features extraction techniques, without involving the proposed learning procedure. Secondly, we investigate the trade-off between accuracy and computational cost evaluating the loss function (8) over eight elaboration units, varying the θ parameter that weights the relevance of the hardware constraints. Table VI summarizes the number and name of the datasets employed to train predictors in each elaboration unit. The first column presents the predictors, the second and the third columns report the processing strategies, the 4-th column states if features have been extracted, the 5-th and 6-th columns indicate the number and the names of datasets generated by the previous stages that fed the predictors, respectively (we abbreviated the names as in Table III, using also the subscript $Proc$ to indicate that we employed one of the second strategy processing).

TABLE VI
Datasets per Elaboration Unit

Predictors	**Process. Strat.**		**Feat. Extr.**	**Num of Datasets**	**Datasets**
	First Str.	Sec. Str.			
H-ELM S-ELM		✓	✓	56	$\mathcal{F}^{Feat}_{Proc}$ $\hat{\mathcal{F}}^{Feat}_{Proc}$
	✓			4	$\mathcal{D}_{Ns}$
	✓		✓	28	$\mathcal{F}^{Feat}_{Ns}$
LSTM GRU	✓			4	$\mathcal{D}_{Ns}$

A. *Generalization Results*

We evaluated the generalization performance on test data not used before for training and validation, setting the parameter $\theta = 0$ in (8) (i.e., without considering the computational cost). Table VII shows the outcomes in terms of accuracy of H-ELM. We report two configurations of the maximum number of hidden neurons N_{max} where the best configuration of N is chosen in the set $N \leq N_{max}$ with $N_{max} = \{100, 1000\}$ (1 and 2 in the table respectively), showing the range of the accuracies (for the others configurations, i.e. 200 and 500, we obtained intermediate results).

The first column reports the datasets accordingly to Table III, the second column shows the two configurations of N_{max}, from the third to the last columns there are all the accuracies averaged on the 10 runs obtained from the different combinations of the feature sets listed in Table IV including no features (i.e. the $\mathcal{D}_{Ns}$ datasets). Concerning the no processing technique (NoP) we show the results on the relevant part of the signals (Section IV-C) as it achieved the best performance. In all the other cases, the best performance was obtained taking into account the whole signals. We highlight in green and in yellow the first two best accuracies obtained over each processing technique and the two configurations of the hidden neurons. The table shows that the best generalization performance (79.0%) is achieved by $N_{max} = 1000$, considering the relevant part, without filtering or decimating the data, and extracting the SM features, and without computing the relevant part, but filtering the data with the low-pass filter, and extracting the SM features. The corresponding datasets are $\hat{\mathcal{F}}^{SM}_{NoP}$ and $\mathcal{F}^{SM}_{Fil}$.

Table VIII shows the accuracies of the S-ELM and has the same structure of Table VII. The overall best accuracy (80.2%) is achieved with 1000 neurons, without considering the relevant part, using decimation, and extracting the ISD+SM features. The corresponding dataset is $\mathcal{F}^{ISD+SM}_{Dec}$.

Table IX reports the accuracies for LSTM and GRU predictors. The best accuracies are obtained by computing 32 slices in both the algorithms, i.e. the $\mathcal{D}_{32}$ dataset.

The tables show how H-ELM and S-ELM with the second strategy processing and features extraction techniques perform better than the other solutions in terms of generalization accuracy, not taking into account the computational costs of the elaboration. In the next section, we analyze the results keeping count of both accuracy and computational cost, by using the Procedure 1 described in Section III, showing which

TABLE VII
GENERALIZATION ACCURACIES OF H-ELM

Dat.	N_{max}	Features $(Feat)$							
		SPE	ISD	SM	SPE ISD	ISD SM	SPE SM	All	No Fea
$\hat{\mathcal{F}}^{Feat}_{NoP}$	1	.697	.599	.722	.642	.679	.720	.684	-
	2	.741	.619	.790	.684	.747	.779	.750	-
$\mathcal{F}^{Feat}_{LPF}$	1	.695	.623	.734	.653	.705	.725	.694	-
	2	.745	.667	.790	.704	.772	.789	.767	-
$\mathcal{F}^{Feat}_{Mov}$	1	.695	.632	.718	.654	.696	.715	.693	-
	2	.747	.677	.782	.710	.770	.776	.769	-
$\mathcal{F}^{Feat}_{Dec}$	1	.695	.628	.719	.658	.692	.714	.693	-
	2	.747	.680	.781	.708	.768	.781	.766	-
$\mathcal{F}^{Feat}_{8}$ $\mathcal{D}_8$	1	.647	.647	.587	.652	.617	.613	.622	.655
	2	.672	.684	.628	.690	.660	.650	.665	.694
$\mathcal{F}^{Feat}_{16}$ $\mathcal{D}_{16}$	1	.655	.649	.622	.655	.627	.627	.636	.635
	2	.697	.702	.655	.707	.678	.674	.690	.695
$\mathcal{F}^{Feat}_{32}$ $\mathcal{D}_{32}$	1	.650	.636	.631	.657	.638	.645	.642	.628
	2	.701	.689	.680	.710	.695	.700	.700	.682
$\mathcal{F}^{Feat}_{64}$ $\mathcal{D}_{64}$	1	.660	.637	.648	.659	.652	.662	.653	.613
	2	.708	.686	.700	.709	.710	.718	.715	.677

TABLE VIII
GENERALIZATION ACCURACIES OF S-ELM

Dat.	N_{max}	Features $(Feat)$							
		SPE	ISD	SM	SPE ISD	ISD SM	SPE SM	All	No Fea
$\hat{\mathcal{F}}^{Feat}_{NoP}$	1	.709	.643	.763	.694	.732	.750	.728	-
	2	.744	.647	.780	.721	.766	.782	.777	-
$\mathcal{F}^{Feat}_{LPF}$	1	.714	.693	.775	.714	.768	.756	.751	-
	2	.745	.732	.796	.763	.798	.794	.799	-
$\mathcal{F}^{Feat}_{Mov}$	1	.713	.690	.766	.722	.769	.752	.748	-
	2	.743	.716	.787	.762	.795	.793	.798	-
$\mathcal{F}^{Feat}_{Dec}$	1	.714	.693	.770	.717	.769	.749	.748	-
	2	.747	.715	.794	.761	.802	.794	.800	-
$\mathcal{F}^{Feat}_{8}$ $\mathcal{D}_8$	1	.639	.662	.596	.664	.629	.609	.621	.658
	2	.664	.674	.617	.683	.668	.655	.672	.690
$\mathcal{F}^{Feat}_{16}$ $\mathcal{D}_{16}$	1	.659	.664	.640	.671	.656	.641	.660	.649
	2	.692	.713	.654	.721	.693	.676	.707	.700
$\mathcal{F}^{Feat}_{32}$ $\mathcal{D}_{32}$	1	.671	.654	.658	.664	.676	.668	.673	.623
	2	.704	.690	.697	.705	.714	.715	.711	.680
$\mathcal{F}^{Feat}_{64}$ $\mathcal{D}_{64}$	1	.686	.652	.680	.677	.688	.686	.690	.612
	2	.717	.687	.729	.717	.736	.734	.738	.680

is the best solution when the hardware constraints become relevant.

B. Novel Loss Function Results

In this experimental campaign, we evaluated both the generalization performance and the computational cost of eight elaboration units. According to Table VI, the eight elaboration units consist of: H-ELM and S-ELM predictors with the second strategy processing $(Proc2)$ and feature extraction (FE) stages, H-ELM and S-ELM predictors with the first strategy processing stage $(Proc1)$, H-ELM and S-ELM predictors with $Proc1$ and FE, LSTM and GRU predictors with $Proc1$.

According to Section III, all the eight units were trained by following the steps of Procedure 1, varying the θ parameter as $\theta = \{0,\ 0.5,\ 1,\ 2,\ 5,\ 10\}$ to differently balance the trade-off between accuracy and the computational cost. The computational costs of all the possible configurations of preprocessing, features extraction, and predictor models were computed a priori and employed in Procedure 1 to find the optimal hyper-parameters.

TABLE IX
GENERALIZATION ACCURACIES OF LSTM AND GRU

Datasets	LSTM	GRU
$\mathcal{D}_8$	.720	.715
$\mathcal{D}_{16}$	.724	.706
$\mathcal{D}_{32}$	.727	.719
$\mathcal{D}_{64}$	.721	.702

First, we employed Procedure 1 to find the best configuration of processing, features, and predictor architecture for each elaboration unit, as θ varies. Eventually, we computed the loss function (8) on each unit by using the best configuration and the test data, comparing the results of the loss function values for each θ value. Figure 4 reports the results. For each θ value, we distinguish 8 bars that correspond to the eight elaboration units: the first two bars refer to H-ELM and S-ELM predictors with $Proc2$ and FE the second two bars concerns H-ELM and S-ELM predictors with $Proc1$, the third two bars represent H-ELM and S-ELM predictors with $Proc1$ and FE, the last two bars identify LSTM and GRU predictors $Proc1$. For $\theta \leq 2$, the figure shows that the best loss function values are achieved by the H-ELM and the S-ELM with $Proc2$ and FE. When θ is bigger, i.e. the computational cost weights more than accuracy in the loss function, the best values are achieved by H-ELM and S-ELM predictors with $Proc1$ and FE, and by LSTM and GRU predictors. Although the computational cost of the H-ELM predictor is lower than the S-ELM given the number of neurons N, experimental outcomes showed that sometimes the latter classifier achieved better results in terms of the balance between the computational cost and generalization accuracy. This is not surprising, as the H-ELM network is based on the threshold activation function, which somewhat affects its generalization performance. Hence, given a classification problem, H-ELM usually requires a larger number of neurons than S-ELM to achieve the same generalization performance.

Table X shows the processing techniques and the extracted features for each elaboration unit and for each θ value, chosen by the evaluation loss function (8) during the learning procedure (Procedure 1). In particular, the first column reports the elaboration unit, the second column the θ values, the third column the processing techniques, the 4-th column the extracted features, the 5-th column the average number of neuron of the predictor, the 6-th column the average computational cost in terms of kFLOPs, and the 7-th column the average generalization accuracy. The averages have been computed on the test data on the different numbers of runs and random starting values of the weights, as explained in Section IV-E. The overall trend is that bigger the θ lower the number of FLOPs of the elaboration units, because the learning procedure rewards the solutions with a lower computational cost. According to Figure 4, we highlighted in green the best elaboration units for each value of θ. All the best units involve ELM-based predictor, showing the best trade-off between generalization performance and computational cost.

VI. CONCLUSIONS

The work presented a novel learning technique based on data-driven approach, which led to trade-off the accuracy

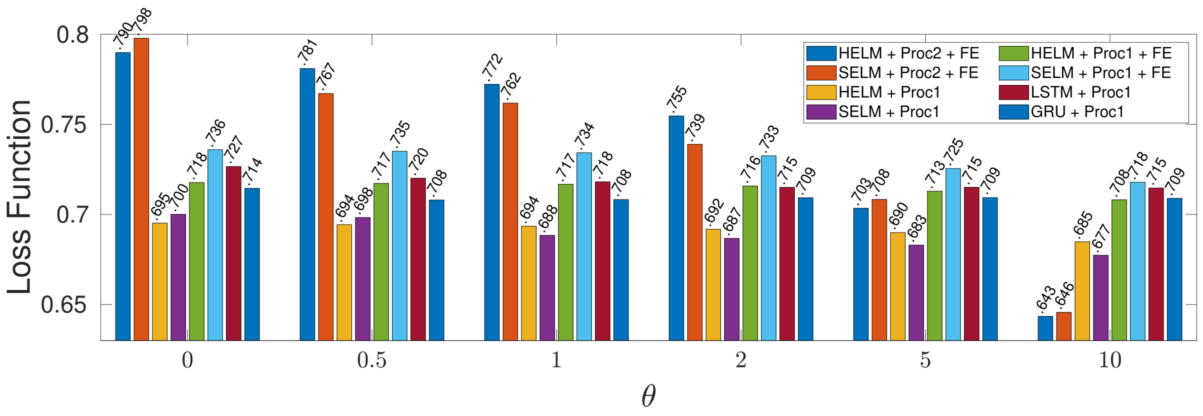

Fig. 4. Loss function values on test data of the eight elaboration units, varying the θ weight.

and computational cost by taking into consideration also the processing units that elaborated signals before classification. We applied the learning procedure to real-world dataset made of three touch modalities, considering the computational cost as the number of FLOPs. The results showed that, when accuracy alone mattered, the elaboration unit that achieved the best generalization performance, 79.8% in average, was based on the random-based SLFNNs (in particular, the S-ELM predictor) with the proposed processing strategy (LPF filtering) and involving the features extraction stage (SM+ISD features). This solution outperformed the state-of-the-art result of 74.5% achieved by a LSTM network. Furthermore, by increasing the weight of the computational cost, the learning strategy allowed one to always choose the best compromise between accuracy and computational cost. In fact, new elaboration units have been selected as best solutions with respect to the one that achieved the best generalization performance when only the accuracy has been taken into consideration. Actually, all the best solutions relied on the SLFNNs, but with different choices in the processing and features extraction stages. In general, the proposed learning strategy can be always employed to deploy an ML-based elaboration system on a given computer system. In any case, the user has to define the suitable cost function R_H related to the underlying target. For example, when targeting a deployment on FGPA, R_H could measure the resource usage. For a micro-controller R_H could measure the memory usage of the classification model or the latency of prediction; in the former case one privileges smaller models in terms of parameters at the expense of the generalization accuracy.

References

[1] Y. Gu, T. Zhang, H. Chen, F. Wang, Y. Pu, C. Gao, and S. Li, “Mini review on flexible and wearable electronics for monitoring human health information,” *Nanoscale research letters*, vol. 14, no. 1, pp. 1–15, 2019.

[2] M. Franceschi, L. Seminara, S. Dosen, M. Strbac, M. Valle, and D. Farina, “A system for electrotactile feedback using electronic skin and flexible matrix electrodes: experimental evaluation,” *IEEE transactions on haptics*, vol. 10, no. 2, pp. 162–172, 2016.

[3] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, “Computational intelligence techniques for tactile sensing systems,” *Sensors*, vol. 14, no. 6, pp. 10 952–10 976, 2014.

[4] E. Ragusa, C. Gianoglio, P. Gastaldo, and R. Zunino, “A digital implementation of extreme learning machines for resource-constrained devices,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 8, pp. 1104–1108, 2018.

[5] E. Ragusa, C. Gianoglio, R. Zunino, and P. Gastaldo, “A design strategy for the efficient implementation of random basis neural networks on resource-constrained devices,” *Neural Processing Letters*, pp. 1–19, 2019.

[6] D. Anguita, A. Ghio, L. Oneto, and S. Ridella, “In-sample and out-of-sample model selection and error estimation for support vector machines,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 9, pp. 1390–1406, 2012.

[7] H. A. Thant, K. M. San, K. M. L. Tun, T. T. Naing, and N. Thein, “Mobile agents based load balancing method for parallel applications,” in *6th Asia-Pacific Symposium on Information and Telecommunication Technologies*. IEEE, 2005, pp. 77–82.

[8] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” *arXiv preprint arXiv:1404.0736*, 2014.

[9] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” *arXiv preprint arXiv:1506.02626*, 2015.

[10] N. K. Jayakodi, A. Chatterjee, W. Choi, J. R. Doppa, and P. P. Pande, “Trading-off accuracy and energy of deep inference on embedded systems: A co-design approach,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2881–2893, 2018.

[11] N. K. Jayakodi, S. Belakaria, A. Deshwal, and J. R. Doppa, “Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, no. 1, pp. 1–24, 2020.

[12] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.

[13] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, “Improved knowledge distillation via teacher assistant,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5191–5198.

[14] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1,” *arXiv preprint arXiv:1602.02830*, 2016.

[15] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, “Cnvlutin: Ineffectual-neuron-free deep neural network computing,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.

[16] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, “Eie: Efficient inference engine on compressed deep neural network,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[17] K. Neshatpour, F. Behnia, H. Homayoun, and A. Sasan, “Icnn: An iterative implementation of convolutional neural networks to enable energy and computational complexity aware dynamic approximation,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 551–556.

[18] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, “Incremental network

TABLE X
ELABORATION UNITS CONFIGURATIONS, AVERAGE NEURONS, AVERAGE FLOPS, AND AVERAGE GENERALIZATION ACCURACY

Elab. Units	Theta	Processing	Features	Av. Neurons	Av. kFLOPs	Av. Accuracy ± STD
HELM + *Proc2* + *FE*	0	LPF	SM	726	50689.4	78.98 ± 3.05
	0.5	Rel + NoP	SM	755	2331.5	78.98 ± 2.47
	1	Rel + NoP	SM	755	2331.5	78.98 ± 2.47
	2	Rel + NoP	SM	755	2331.5	78.98 ± 2.47
	5	NoP	SM	688	1726.8	76.35 ± 3.79
	10	NoP	SM	688	1726.8	76.35 ± 3.79
SELM + *Proc2* + *FE*	0	LPF	SM+ISD	655	60661.1	79.78 ± 3.05
	0.5	Rel + NoP	All	724	2572.7	77.70 ± 3.66
	1	Rel + NoP	SM	583	2366.5	77.97 ± 3.44
	2	NoP	All	658	1945.7	76.69 ± 4.97
	5	NoP	SM	558	1762.6	77.00 ± 3.08
	10	NoP	SM	466	1749.0	76.77 ± 3.01
HELM + *Proc1*	0	16 Slices	-	686	602.9	69.52 ± 3.72
	0.5	16 Slices	-	686	602.9	69.52 ± 3.72
	1	16 Slices	-	686	602.9	69.52 ± 3.72
	2	16 Slices	-	686	602.9	69.52 ± 3.72
	5	8 Slices	-	673	501.0	69.37 ± 3.75
	10	8 Slices	-	631	495.4	69.37 ± 3.66
SELM + *Proc1*	0	16 Slices	-	714	804.5	70.01 ± 3.84
	0.5	16 Slices	-	714	804.5	70.01 ± 3.84
	1	8 Slices	-	687	601.7	69.01 ± 3.32
	2	8 Slices	-	687	601.7	69.01 ± 3.32
	5	8 Slices	-	622	583.7	69.06 ± 3.17
	10	8 Slices	-	531	558.6	69.02 ± 3.17
HELM + *Proc1* + *FE*	0	64 Slices	SM+SPE	831	523.7	71.77 ± 3.09
	0.5	64 Slices	SM+SPE	831	523.7	71.77 ± 3.09
	1	64 Slices	SM+SPE	831	523.7	71.77 ± 3.09
	2	64 Slices	SM+SPE	831	523.7	71.77 ± 3.09
	5	64 Slices	SM+SPE	831	523.7	71.77 ± 3.09
	10	64 Slices	SM+SPE	773	516.0	71.70 ± 2.92
SELM + *Proc1* + *FE*	0	64 Slices	SM+ISD	695	607.2	73.60 ± 3.01
	0.5	64 Slices	SM+ISD	695	607.2	73.60 ± 3.01
	1	64 Slices	SM+ISD	695	607.2	73.60 ± 3.01
	2	64 Slices	SM+ISD	695	607.2	73.60 ± 3.01
	5	64 Slices	SM+ISD	651	594.9	73.35 ± 3.02
	10	64 Slices	SM+ISD	562	570.3	73.17 ± 3.11
LSTM + *Proc1*	0	32 Slices	-	75	1827.7	72.66 ± 2.87
	0.5	32 Slices	-	75	1827.7	72.66 ± 2.87
	1	16 Slices	-	54	1095.6	72.43 ± 3.18
	2	8 Slices	-	39	702.6	72.02 ± 4.03
	5	8 Slices	-	29	531.1	72.02 ± 4.03
	10	8 Slices	-	24	480.1	72.02 ± 4.03
GRU + *Proc1*	0	8 Slices	-	60	1089.8	71.45 ± 4.21
	0.5	8 Slices	-	60	1089.8	71.45 ± 4.21
	1	8 Slices	-	49	895.9	71.45 ± 4.21
	2	8 Slices	-	35	658.7	71.45 ± 4.21
	5	8 Slices	-	27	526.7	71.45 ± 4.21
	10	8 Slices	-	22	463.8	71.45 ± 4.21

quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[19] S. I. Mirzadeh and H. Ghasemzadeh, "Optimal policy for deployment of machine learning models on energy-bounded systems," in *29th International Joint Conference on Artificial Intelligence (IJCAI)*, 2020, pp. 3422–3429.

[20] M. Alameh, A. Ibrahim, M. Valle, and G. Moser, "Dcnn for tactile sensory data classification based on transfer learning," in *2019 15th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*. IEEE, 2019, pp. 237–240.

[21] M. Alameh, Y. Abbass, A. Ibrahim, G. Moser, and M. Valle, "Touch modality classification using recurrent neural networks," *IEEE Sensors Journal*, 2021.

[22] S.-y. Koo, J. G. Lim, and D.-s. Kwon, "Online touch behavior recognition of hard-cover robot using temporal decision tree classifier," in *RO-MAN 2008-The 17th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE, 2008, pp. 425–429.

[23] D. Silvera Tawil, D. Rye, and M. Velonaki, "Interpretation of the modality of touch on an artificial arm covered with an eit-based sensitive skin," *The International Journal of Robotics Research*, vol. 31, no. 13, pp. 1627–1641, 2012.

[24] J. Friedman, T. Hastie, R. Tibshirani *et al.*, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *Annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[25] M. Kaboli, A. Long, and G. Cheng, "Humanoids learn touch modalities identification via multi-modal robotic skin and robust tactile descriptors," *Advanced Robotics*, vol. 29, no. 21, pp. 1411–1425, 2015.

[26] W. Zong, G.-B. Huang, and Y. Chen, "Weighted extreme learning machine for imbalance learning," *Neurocomputing*, vol. 101, pp. 229–242, 2013.

[27] J. Tang, C. Deng, and G.-B. Huang, “Extreme learning machine for multilayer perceptron,” *IEEE transactions on neural networks and learning systems*, vol. 27, no. 4, pp. 809–821, 2015.
[28] P. Gastaldo, F. Bisio, C. Gianoglio, E. Ragusa, and R. Zunino, “Learning with similarity functions: a novel design for the extreme learning machine,” *Neurocomputing*, vol. 261, pp. 37–49, 2017.
[29] S. Decherchi, P. Gastaldo, A. Leoncini, and R. Zunino, “Efficient digital implementation of extreme learning machines for classification,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 8, pp. 496–500, 2012.
[30] T. Guo, L. Zhang, and X. Tan, “Neuron pruning-based discriminative extreme learning machine for pattern classification,” *Cognitive Computation*, vol. 9, no. 4, pp. 581–595, 2017.
[31] E. Ragusa, C. Gianoglio, R. Zunino, and P. Gastaldo, “A computationally light pruning strategy for single layer neural networks based on threshold function,” in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2019, pp. 89–92.
[32] C. Lian, Z. Zeng, W. Yao, and H. Tang, “Ensemble of extreme learning machine for landslide displacement prediction based on time series analysis,” *Neural Computing and Applications*, vol. 24, no. 1, pp. 99–107, 2014.
[33] A. Iosifidis, A. Tefas, and I. Pitas, “Dropelm: Fast neural network regularization with dropout and dropconnect,” *Neurocomputing*, vol. 162, pp. 57–66, 2015.
[34] J. Zhai, L. Zang, and Z. Zhou, “Ensemble dropout extreme learning machine via fuzzy integral for data classification,” *Neurocomputing*, vol. 275, pp. 1043–1052, 2018.
[35] E. Ragusa, C. Gianoglio, P. Gastaldo, and R. Zunino, “Improving the robustness of threshold-based single hidden layer neural networks via regularization,” in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2020, pp. 276–280.
[36] E. Ragusa, C. Gianoglio, R. Zunino, and P. Gastaldo, “Random-based networks with dropout for embedded systems,” *Neural Computing and Applications*, vol. 33, no. 12, pp. 6511–6526, 2021.
[37] S. Decherchi, P. Gastaldo, R. S. Dahiya, M. Valle, and R. Zunino, “Tactile-data classification of contact materials using computational intelligence,” *IEEE Transactions on Robotics*, vol. 27, no. 3, pp. 635–639, 2011.
[38] H. Liu, J. Qin, F. Sun, and D. Guo, “Extreme kernel sparse learning for tactile object recognition,” *IEEE transactions on cybernetics*, vol. 47, no. 12, pp. 4509–4520, 2016.
[39] M. Rasouli, Y. Chen, A. Basu, S. L. Kukreja, and N. V. Thakor, “An extreme learning machine-based neuromorphic tactile sensing system for texture recognition,” *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 2, pp. 313–325, 2018.
[40] V. N. Vapnik, “An overview of statistical learning theory,” *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.
[41] Y. Jin and B. Sendhoff, “Pareto-based multiobjective machine learning: An overview and case studies,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 3, pp. 397–415, 2008.
[42] M. Alameh, Y. Abbass, A. Ibrahim, and M. Valle, “Smart tactile sensing systems based on embedded cnn implementations,” *Micromachines*, vol. 11, no. 1, p. 103, 2020.
[43] S. W. Smith *et al.*, *The scientist and engineer’s guide to digital signal processing*. California Technical Pub. San Diego, 1997, vol. 14.
[44] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, “Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package),” *Neurocomputing*, vol. 307, pp. 72–77, 2018.
[45] G.-B. Huang, L. Chen, C. K. Siew *et al.*, “Universal approximation using incremental constructive feedforward networks with random hidden nodes,” *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

APPENDIX

ELM predictor: In the following, we briefly describe the random-based SLFNNs predictors for a bi-class problem (all the considerations hold also for the multi-class classification). Let $\mathcal{T} = \{(\boldsymbol{x}, y)_i; \boldsymbol{x} \in \mathbb{R}^D; y \in \{-1, 1\}; i = 1, ..., Z\}$ be a labeled bi-class training set and $\phi(\boldsymbol{x} \cdot \boldsymbol{\omega} + b)$ an activation function, where $\omega \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are free parameters. An ELM is a SLFNN having N neurons in the hidden layer that remaps input data into a $\mathbb{R}^N$ feature space and relies on the following decision function:

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} \beta_n \phi(\boldsymbol{x} \cdot \boldsymbol{\omega}_n + b_n) \tag{14}$$

where the parameters $\{\boldsymbol{\omega}, b\}_n$ are set randomly and ϕ is a nonlinear, piecewise continuous function that satisfies the universal approximation capability theorems for ELMs [45] (e.g. sigmoid, rbf, tanh). Training an ELM requires to solve a Regularized Least Square (RLS) problem in the feature space $\mathbb{R}^N$. Let consider $\boldsymbol{H}$ as a $Z \times N$ matrix, where $h_{in} = \phi(\boldsymbol{x}_i^t \boldsymbol{\omega}_n + b_n)$, thus the learning problem is:

$$\min_{\boldsymbol{\beta}} \{\|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|^2\} \tag{15}$$

where λ is a regularization parameter, and $\boldsymbol{\beta}$ is the weight vector that connects the hidden layer to the output. The solution is:

$$\boldsymbol{\beta} = \boldsymbol{H}^T (\lambda \boldsymbol{I} + \boldsymbol{H}\boldsymbol{H}^T)^{-1} \boldsymbol{y} \tag{16}$$

Deep Networks predictors: A LSTM unit equations with a forget unit are:

$$\begin{aligned}
\boldsymbol{f}_t &= sigm(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f)\\
\boldsymbol{i}_t &= sigm(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i)\\
\boldsymbol{o}_t &= sigm(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o)\\
\tilde{\boldsymbol{c}}_t &= tanh(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c)\\
\boldsymbol{c}_t &= \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tilde{\boldsymbol{c}}_t\\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ tanh(\boldsymbol{c}_t)
\end{aligned} \tag{17}$$

while a GRU unit equations are:

$$\begin{aligned}
\boldsymbol{z}_t &= sigm(\boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{U}_z \boldsymbol{h}_{t-1} + \boldsymbol{b}_z)\\
\boldsymbol{r}_t &= sigm(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{U}_r \boldsymbol{h}_{t-1} + \boldsymbol{b}_r)\\
\hat{\boldsymbol{h}}_t &= tanh(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{U}_h (\boldsymbol{r}_t \circ \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h)\\
\boldsymbol{h}_t &= (\boldsymbol{1} - \boldsymbol{z}_t) \circ \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \circ \hat{\boldsymbol{h}}_t
\end{aligned} \tag{18}$$

where $\circ$ denotes the element-wise product, the subscript t indexes the time step, and:

$$\begin{aligned}
&\boldsymbol{x}_t \in \mathbb{R}^d : \text{input vector}\\
&\boldsymbol{h}_t \in \mathbb{R}^N : \text{output vector}\\
&\hat{\boldsymbol{h}}_t \in \mathbb{R}^N : \text{candidate activation vector}\\
&\boldsymbol{f}_t \in \mathbb{R}^N : \text{forget gate's activation vector}\\
&\boldsymbol{i}_t \in \mathbb{R}^N : \text{input/update gate's activation vector}\\
&\boldsymbol{o}_t \in \mathbb{R}^N : \text{output gate's activation vector}\\
&\boldsymbol{z}_t \in \mathbb{R}^N : \text{update gate vector}\\
&\boldsymbol{r}_t \in \mathbb{R}^N : \text{reset gate vector}\\
&\tilde{\boldsymbol{c}}_t \in \mathbb{R}^N : \text{cell input activation vector}\\
&\boldsymbol{c}_t \in \mathbb{R}^N : \text{cell state vector}\\
&\boldsymbol{W} \in \mathbb{R}^{N \times d}, \boldsymbol{U} \in \mathbb{R}^{N \times N}, \boldsymbol{b} \in \mathbb{R}^N : \text{weight matrices}\\
&\text{and bias vector parameters which need to be learned}\\
&\text{during training}
\end{aligned} \tag{19}$$

with the superscript d refers to the number of input features, and N to the number of hidden units.