Neural Computing and Applications
https://doi.org/10.1007/s00521-021-06138-9

ORIGINAL ARTICLE

Machine learning for computationally efficient electrical loads estimation in consumer washing machines

Vittorio Casagrande[2] · Gianfranco Fenu[1] · Felice Andrea Pellegrino[1] · Gilberto Pin[3] · Erica Salvato[1] · Davide Zorzenon[4]

Received: 2 September 2020 / Accepted: 20 May 2021

Abstract
Estimating the wear of the single electrical parts of a home appliance without resorting to a large number of sensors is desirable for ensuring a proper level of maintenance by the manufacturers. Deep learning techniques can be effective tools for such estimation from relatively poor measurements, but their computational demands must be carefully considered, for the actual deployment. In this work, we employ one-dimensional Convolutional Neural Networks and Long Short-Term Memory networks to infer the status of some electrical components of different models of washing machines, from the electrical signals measured at the plug. These tools are trained and tested on a large dataset (502 washing cycles $\approx$ 1000 h) collected from four different washing machines and are carefully designed in order to comply with the memory constraints imposed by available hardware selected for a real implementation. The approach is end-to-end; i.e., it does not require any feature extraction, except the harmonic decomposition of the electrical signals, and thus it can be easily generalized to other appliances.

Keywords Long short term memory · One-dimensional convolutional neural network · Memory efficiency · Washing machine

1 Introduction

The reliability of a home appliance is an important component of its quality, thus tools for guaranteeing an appropriate level of maintenance are highly desirable for the manufactures. A possible, but expensive, way to ensure predictive maintenance is equipping the appliance with many sensors that report wear. Another approach (made possible by the availability of devices such as smart-plugs [8]), is based on the electrical signals drawn from the grid (physical quantities rather easy to measure), that can be used, employing appropriate machine learning tools, to infer the status of different components and raise maintenance intervention requests. Deep learning has proven to be effective when dealing with time series data (see, for instance, [15]) and is attractive for its ability to

This work has been partially supported by the Italian Ministry for Research under the initiative "Departments of Excellence" (Law 232/2016) and the framework of the 2017 Program for Research Projects of National Interest (PRIN), Grant No. 2017YKXYXJ.

✉ Erica Salvato
erica.salvato@phd.units.it

Vittorio Casagrande
vittorio.casagrande.19@ucl.ac.uk

Gianfranco Fenu
fenu@units.it

Felice Andrea Pellegrino
fapellegrino@units.it

Gilberto Pin
pingilbert@gmail.com

Davide Zorzenon
davide.zorzenon@tu-berlin.de

[1] Department of Engineering and Architecture, University of Trieste, Trieste, Italy

[2] Department of Electrical and Electronic Engineering, University College London, London, UK

[3] Department of Information Engineering, University of Padua, Padua, Italy

[4] Technische Universität Berlin, Control Systems Group, Berlin, Germany

Published online: 15 June 2021

automatically extract features from data, thus not requiring a thorough knowledge of the monitored appliances.

In a previous work by some of the authors [3], deep learning tools have been investigated as a means to infer the activity status of some internal components of washing-machines, from the electrical signals measured at the plug.

In particular, Convolutional Neural Networks (CNNs) and Long Short-Term Memories (LSTMs) have been employed: the former, only to classify the binary status of the electrovalves, the heater and the drain pump; the latter, for the same classification problems and, moreover, to provide an estimate of the drum speed (a regression problem). Results have shown that good performance can be achieved using LSTMs for the regression, and CNNs for the classification problem. Here, we expand the mentioned study, focusing only on the classification problem (Fig. 1), and we take into account hardware and memory constraints, as a premise for the actual deployment on an appropriate micro-controller. Such constraints turn out to have significant consequences, that allow for complementing the analysis carried out in [3]. Deep learning techniques have been already employed for similar tasks. In [22] and [19], deep learning tools have been used to detect the energy consumption of each appliance of a household, from aggregate measurements of voltage and/or current in the distribution system. The use of an evolutionary type of neural network, the Group Method Data Handling (GMDH), is proposed in [9] in order to predict the energy consumption of a smart home. In [24], a comparison is carried out of several deep learning tools (such as recurrent neural networks, LSTMs and auto-encoders) aimed at predicting energy consumption patterns in a building, detecting anomalies of usage that generate an increase in energy consumption and the device that causes them. In [25] a CNN-based multitasking learning method is used on the C-MAPSS dataset to highlight the correlation of the remaining useful life (RUL) estimation of a complex system with the detection of its health status. Abstracting from specific application under exam, embedding deep learning architectures into low-cost microcontrollers is an open challenge that engineers and data scientists are tackling from several directions [7]. Currently, the three main trends are (1) scaled-down AI with pruning and weight quantization [2, 14, 17, 29, 30]; (2) the use of simplified or approximated activation functions [4, 20, 31]; (3) the use of recursive NN structures (like e.g., LSTMs and Gated recurrent Units, GRU) [28].

While the main focus of these approaches is on image processing applications, in this work we address instead the problem of developing an embedded-AI solution for classification applied to time-series data.

The process of developing an effective embedded-AI solution, for target platforms with limited memory and hard computational constraints, is far from trivial and involves several steps, among which the most important are the choice of the network structure, the type of activation functions and the value of the training hyperparameters. Here, we face the former and the latter.

The remainder of the paper is organized as follows. The problem is stated in Sect. 2, along with a brief description of the machine learning tools that have been employed. In Sect. 3 the dataset and its pre-processing are described. Section 4 outlines the performance index for networks evaluation, and the hardware constraints. Sections 5 and 6 describe, respectively, the adopted architecture for the CNNs and LSTMs, and the hyperparameters settings complying with the memory restrictions. In Sect. 7, experimental results are reported and, finally, the conclusions are drawn in Sect. 8.

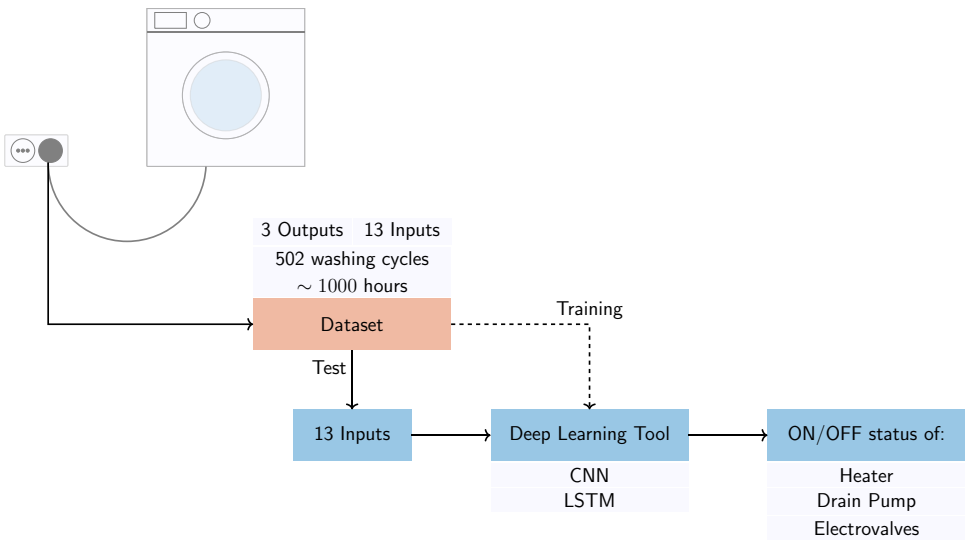

Fig. 1 General layout of the investigated application. A smart-plug [8] provides periodic measurements of specific electrical quantities during the operating cycles of an appliance. The resulting data are used first to train a model and, subsequently, to test it. The obtained model is able to provide a prediction of the activity status of some components, on the basis of 13 different input signals

2 Problem statement and tools

We face the problem of estimating the status of some components (referred to as "electrical loads," or simply "loads") of different washing machine models, based on electrical quantities measured at the plug. The collection of periodic measurements constitutes a time series. The estimation should be performed at any time, based on the time series collected to that time. Among the many tools that allow for making predictions from time series, we investigate the one-dimensional CNNs and the LSTMs. Both the tools are state-of-the-art approaches that have been proven successful in many challenging applications involving time series [10, 18, 21, 32–34].

More specifically, the problem can be stated as follows: *design a network that reaches a satisfactory performance in estimating the status of the electrical loads, from the electrical quantities measured at the plug, and is compliant with given memory constraints* (details are outlined in Sect. 4). This means that even if a network with better performance can be obtained, it is discarded whenever its memory demand exceeds the above mentioned limit. The promoted network will, therefore, be the best performing network attainable given the maximum available memory. The trade-off between memory demand and performance is a fundamental issue when trying to port a neural network to a micro-controller. That is the case, for example, of the deployment of neural networks into a smart-plug [8]: a device developed for connecting low-cost household appliances, with no connection capability to the Internet, to a home network system.

We report now some basic facts about CNNs and LSTMs, necessary to understand the design choices to be described next.

2.1 One-dimensional CNNs

Although historically developed for 2D data (images), CNNs can be used on one-dimensional data, such as the electrical signals drawn from the grid, recasting the networks architecture in a multi-channel one-dimensional CNN. The general layout of a one-dimensional CNN is shown in Fig. 2. The employed network architecture is described in detail in Sect. 5.1.

It consists of a first convolutional hidden layer, which operates over a one-dimensional sequence (i.e., the filters slide along a single dimension), directly followed by a pooling layer able to reduce data size (it combines the results of neuron clusters of one layer into a single neuron in the next layer). Additional pairs of convolutional and pooling layers are usually included in the network which then ends in a dense fully connected layer, able to interpret the features extracted by the convolutional part of the model. In order to be able to use more than one electrical time series as input, each signal has been treated as a different channel of the CNN *input layer*.

2.2 LSTMs

LSTMs are able to learn arbitrarily long term dependencies in time series thanks to a system of gating units that regulates the information flow through the network. A generic LSTM cell is outlined in Fig. 3.

The LSTMs cell core is the state c, that flows along the entire cell with only few linear interactions.

Gates are a way to remove or add information to this state. They consist of a gate activation function σ and a multiplier operator $\otimes$. The former outputs a number between 0 and 1 quantifying the transit permission of each component (i.e., 1 allows totally the transit, 0 precludes the transit).

Each LSTM cell has three gates. The first is called *forget gate* and selects the information that is allowed access to the cell. The second one, called *input gate*, decides which new information to store in the cell state, and with the help of a state activation function (σ_c) creates a vector of new candidate values that could be added to the state. The last one, the *output gate*, selects which part of the state to let out of the cell. The output of this gate activation function is multiplied by the state of the cell that has been previously passed into a σ_c layer.

3 Dataset description

We deal with the estimation problem in terms of supervised learning, i.e., we construct a training set composed of observations (values of the predictor variables, those that are actually measured and upon which the prediction depends) and corresponding target values (the values to be predicted). A learning machine is then trained, based on the training set, to predict the correct target values given the predictor values. The performance of the trained machine (referred to as a *model*) is evaluated on a set called test set, not employed for the training. The training and test sets are built based on the data collected by performing several washing cycles on four different washing machine models. The data collected within a washing cycle are referred to as a sequence. During data acquisition, four different types of washing cycles have been performed (cotton, delicate, synthetic, wool), each of which is characterized by a different maximum spinning speed and maximum temperature. In order to cover a wide range of realistic operating settings, tests have been performed by varying some operating parameters (Table 1), complying with the

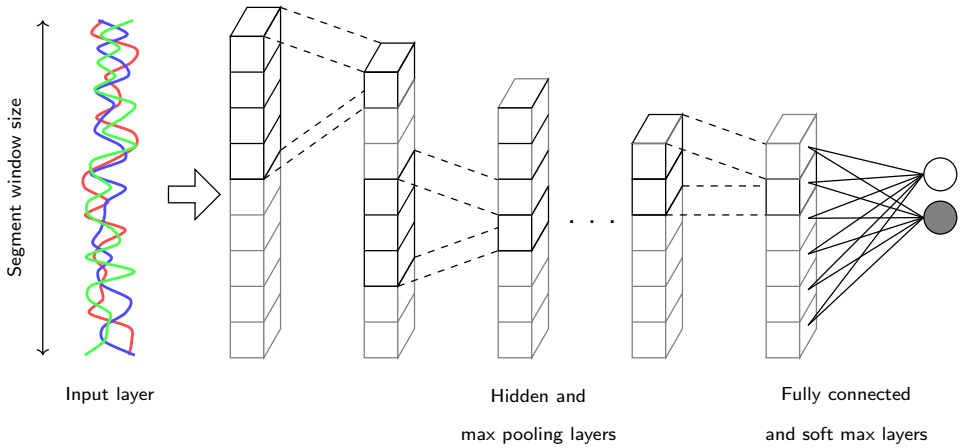

Fig. 2 General layout of a one-dimensional CNN. The exact description of the model used in the present application is provided in Sect. 5.1, while the tuned hyperparameters values are in Sect. 6.1

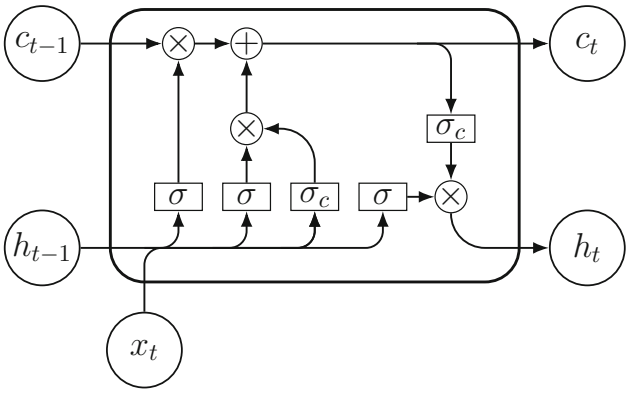

Fig. 3 General layout of a LSTM cell. The exact description of the model used in the present application is provided in Sect. 5.2, while the tuned hyperparameters values are in Sect. 6.2

constraints imposed by the selected cycle. Each recorded sequence contains data of a complete washing cycle performed with fixed operating parameters. The entire dataset is summarized in Table 2).

The recorded samples are acquired each 0.5s and consist of: the real and imaginary parts of current harmonics (1st, 3rd and 5th harmonics), the real and imaginary parts of voltage harmonics (1st, 3rd and 5th harmonics) and the cumulative energy drawn from the grid, which constitute the predictor variables; the heater status, the drain pump status, and the electrovalves status that, conversely, are the target variables. The choice of current and voltage harmonics as inputs for the networks is motivated by the observation that it is well known from power-electrical theory that each electrical device type possess a typical signature in the domain of current harmonics. Therefore, the harmonics can be regarded as data features for the problem under considerations, whose extraction from raw signals can be performed by conventional algorithms (e.g., FFT over sliding windows). The networks therefore are implicitly required to learn, from the training set, the harmonic signature of the electrical loads even in case of possible simultaneous activations.

Table 2 Number and total duration of the training and test sequences collected

	Training	Test	Total
Sequences	434	110	544
Duration (h)	1002	135	1137

The whole dataset is composed of 544 sequences, corresponding to 1137 hours of washing time. In Table 2, the number of sequences used for training and test are reported. The dataset splitting has been done manually once, by assigning each sequence to the training or testing set in order to ensure that the different operating conditions are represented roughly equally in both sets. The proportion between training and test data is approximately 80% and 20%.

The training process has been performed differently for CNNs and LSTMs. In the former, a fivefold cross-validation is performed on the training set in order to select the best model. The LSTMs tuning, conversely, relies on Bayesian optimization. For the latter, a random subset

Table 1 Operating parameters of washing machines during data collection

	Supply voltage	Load (laundry)	Spinning speed	Temperature
Ranges of values	230 V	Full	0	Room
	187 V	no load	maximum	90°C

consisting of 20% of the training data has been used for the validation. All the details concerning the hyperparameters tuning are reported in Sect. 6. The final performance highlighted in Sect. 7 are obtained feeding the resulting models with the test data.

Given the collected data, the preparation of the actual training and test sets is a non-trivial task that encompasses some design choices and must possibly address a class imbalance problem. We discuss the mentioned issues in the following.

3.1 Segmentation

CNNs require to be fed with fixed-size data. We have thus decided to segment each recorded sequence into subsequences (segments) and recast the problem in terms of *subsequence-to-label* classification. To each input subsequence, a single output (label) is assigned and the resulting model has to map input subsequences into output labels (Fig. 2). The subsequence length is a design parameter, affecting the architecture of the CNN, and will be discussed later on. The assignment criterion is a design choice as well: one could, for instance, take the target value corresponding to the last sample of the segment, or the median target of the whole segment; many different choices are possible. Based on an exploratory analysis, we decided to set the label of a segment as the target of the central sample of the considered segment; indeed, with such a choice we observed better performing classifiers. Although these results suggest that, for the considered problem, future samples improve the prediction of the current output, we did not investigate any further, and kept the labeling method fixed.

The LSTMs, conversely, do not need to be fed by observations of the same length, hence segmentation is not needed [16]. Notwithstanding, we did perform segmentation for the LSTMs as well, in combination with oversampling and/or undersampling, to alleviate the classes imbalance problem (see Sects. 3.2 and 6.2 for further details). Once the recorded sequences have been segmented into subsequences, two different strategies can be followed to train and employ LSTMs, i.e., the same subsequence-to-label approach employed for CNNs, and the *subsequence-to-subsequence* classification. The latter consists in mapping input subsequences into output subsequences of the same length and has a remarkable advantage over the former: it allows, at inference phase, to feed the network with *an entire unsegmented sequence*. Clearly, feeding the network with the actual acquired sequence, during the operation of the deployed device, has the advantage that there is no need to store segments, but it is sufficient to pass the measures sample-wise, as they are acquired; provided that the classification performance is acceptable, the memory requirements of such approach are independent of the length of the actual subsequences employed for training. For the sake of comparison, we have performed both subsequence-to-label and subsequence-to-subsequence classification when using LSTMs; in the latter, data segmentation was performed only for the training.

The segmentation process is defined by two parameters: the *window size* (i.e., the number of samples per segment) and the *stride* (number of samples between the first sample of two consecutive segments). For both CNNs and LSTMs, the stride was set to 1. Each segment is composed of:

$$n_s = n_u \times s, \tag{1}$$

values, where n_u is the number of inputs (13 in our case) and s is the window size (length of the segment). Since the number of inputs is fixed, the only parameter to tune is the window size. On the one hand, this value has to be large enough to contain sufficient information for a correct estimation, on the other, it should be as small as possible to avoid unnecessary memory consumption.

In the subsequence-to-subsequence approach of LSTMs, each target segment simply consists of the target values corresponding to the predictor segment and is entirely considered for the performance evaluation.

3.2 Class balancing

As shown in Table 3, the recorded sequences exhibit a high class imbalance, caused by the dominance of the OFF status. Class imbalance is a well-known problem in Machine Learning [6, 12] and must be properly addressed. On the one hand, one can attempt to establish a class balance acting on the training set; on the other, the performance metric that the learning algorithm seeks to maximize can be modified to cope with unbalance. In either case, the metric for evaluating the performance of the classifier must take the imbalance into account. Several methods have been proposed in literature ([5, 6]) to reduce the effect of class imbalance acting on the training set. Two well-known methods are:

– *oversampling*: duplicate randomly chosen segments of the less common class; such a random information-duplication procedure may lead, however, to a huge

Table 3 Percentage of ON and OFF samples for each load

	Heater	Drain pump	Electrovalves
OFF samples	86%	87%	98%
ON samples	14%	13%	2%

increase of dataset size and, in addition, to overfitting [5];
– *undersampling*: delete randomly chosen segments of the most common class, thus reducing the dataset size (possibly at the cost of deleting relevant information for the nets training process [6]).

We have applied both, and we report comparative results in Sect. 6.

As far as the evaluation metric is concerned, it is described in Sect. 4.

4 Performance indices and memory constraints

The trained models will be evaluated in terms of 1) the classification performance, i.e., the capability of inferring the correct value of the target variables, and 2) their memory footprint, i.e., the memory occupied by the deployed model in the target device.

4.1 Performance Indices

As already mentioned, each load can assume two values: ON or OFF, leading to three different binary classification problems (a different network is trained for each load).

Due to class imbalance, in the present scenario the *accuracy* (ratio of correct predictions to the total) is not enough for a proper evaluation of the performance. Hence, two other indices, *precision* and *recall* [11], are computed for each network. Their harmonic mean, the *F1-score*, is finally used as evaluation metric of the network performance:

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \quad (2)$$

In particular, two *F1-scores* are computed for each network: one for evaluating the capability of predicting the ON status, and the other for the OFF status. The goal of this work is training a classifier in which both these indices are as close to 100% as possible.

4.2 Memory Constraints

A 32-bit Cortex-M microcontroller [1], with 64 kB flash memory and 32 kB RAM, is available for the real-time implementation on the real appliances. Such a microcontroller, besides the computations required by the neural networks at inference time, has to perform simultaneously several other operations, such as the extraction of the harmonic information from voltage and current samples, and the management of input–output peripherals. Accordingly, an upper bound to the memory available for the networks has been set to 12 kB, which corresponds to 4 kB for each of the three networks (the three classifiers, one for each load). This is a rather tight constraint that, as shown in Sect. 6, has ultimately led us to discard CNNs and promote LSTMs.

5 Network architectures

In the following, we describe the architectures of the deployed networks: a one-dimensional CNN, and an LSTM. The reported architectural details are instrumental to the evaluation of the memory requirements of the trained models.

5.1 One-Dimensional Convolutional Neural Networks

The architecture of the convolutional neural networks used in this work consists of:

(i) input layer,
(ii) hidden layer 1,
(iii) max pooling layer 1,
(iv) hidden layer 2,
(v) max pooling layer 2,
(vi) ...,
(vii) hidden layer N_{hl},
(viii) fully connected layer,
(ix) softmax layer,
(x) classification layer;

where N_{hl} is the number of hidden layers, each of which is composed of:

(i) a convolutional layer,
(ii) a batch normalization layer,
(iii) a Rectified Linear Unit (ReLU) layer.

The input layer defines the dimension of the input observations; in our case, each observation has unitary height, width corresponding to the length of the segments, and number of channels equal to the number of input signals, i.e., 13. Each of the N_{hl} hidden layers has the same structure: first, the convolutional layer, consisting of a certain number of linear filters, whose coefficients constitute the trainable parameters of the layer; then, the batch normalization layer, which is customary to speed up the training of CNNs and, essentially, learns a rescaling and shift transform, whose parameters are trainable parameters as well; finally, the ReLU layer applies the rectifier activation function $\text{ReLU}(x) = \max(x, 0)$ to each element x of the output of the previous layer and does not contain trainable parameters. Architectural parameters such as

number of filters, filters length, stride and padding must be properly tuned in order to achieve good results in classification problems as will be discussed in the next section. The hidden layers are interspersed with max pooling layers, that perform dimensionality reduction by means of a maximum operation among nearby values and do not contain trainable parameters.

The last layers are the fully connected layer and the softmax layer. The former, learns an affine transform whose parameters (a weighting matrix and a bias vector) are trainable. Finally, the softmax layer (not containing trainable parameters) allows to obtain a probability distribution of K probabilities, by applying the softmax operation:

$$Y_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \tag{3}$$

where K is the total number of classes (2 in our problem), Y_i is the output probability of class i, and x_k is the k-th output of the fully connected layer. Accordingly, during training, a cross-entropy loss function is employed:

$$L = -\sum_{i=0}^{K-1} T_i \log(Y_i), \tag{4}$$

where T_i is the target output of each class and Y_i is the output of the network. Further details on the architecture can be found in [3].

5.2 Long Short-Term Memories

The architecture of the LSTMs employed in the present work is as follows:

(i) sequence input layer,
(ii) LSTM layer,
(iii) fully connected layer,
(iv) softmax layer.

The LSTM layer is the peculiar layer of this kind of network. Thanks to the system of gating units, represented in Fig. 3, the network can learn long-term dependencies. Specifically, the operation performed by each gate is:

$$y_t = \sigma(Wx_t + Rh_{t-1} + b), \tag{5}$$

where y_t is the output of an LSTM gate (e.g., the forget gate) at time t; x_t is the input to the LSTM layer at time t (the values at time t of the 13 input signals); h_t is the output of the LSTM layer at time t; W, R and b are, respectively, the input weights, the recurrent weights and the biases specific of an LSTM gate; σ is the gate activation function, i.e., the nonlinear function that controls the information flow through the gate. The weights and bias of the fully connected layer, along with W, R and b, are the trainable parameters of the LSTM. As far as the loss function is concerned, we employed the cross-entropy (4).

6 Hyperparameters Setting and Training

In the following, the hyperparameters setting procedures are explained for each network, along with more detailed information about their training conditions. The choices were done pursuing two main objectives: (1) maximize the network performance, and (2) minimize the memory requirements. Precisely, since the maximum memory that can be employed to store the network parameters is fixed (Sect. 4.2), our efforts have been devoted to achieving the best performance given the available memory.

6.1 CNNs

Given the structure discussed in Sect. 5.1, the hyperparameters that need to be tuned are: (1) the length of the segments, (2) the number of hidden layers, (3) the number of filters of each convolutional layer, (4) the width of the filters, (5) the stride and padding of each convolutional and max pooling layer.

In order to simplify the process of parameter tuning and reduce the degrees of freedom, we decided to fix the value of some of the hyperparameters. In particular: (1) padding was set to zero for each layer; (2) stride was set to one for convolutional layers and two for max pooling layers; (3) the window size was set to two for each max pooling layer, meaning that the dimension of the data is halved by each pooling operation; (4) inspired by [15], the number of filters $N_{\mathrm{f}}(i)$ of the i-th hidden layer ($i = 1, \ldots, N_{\mathrm{hl}}$) was set according to $N_{\mathrm{f}}(i) = 2^{i+\alpha}$, where $\alpha \in \mathbb{N}$ is a constant to be chosen.

Therefore, the hyperparameters to be tuned are the size of the segments s, the number of hidden layers N_{hl}, the width of the convolutional filters $w_1, \ldots, w_{Nhl}$, and α. It is worth noticing that the size of the network is monotonically increasing with respect to each of the remaining tunable hyperparameter.

Despite the many heuristics proposed in the literature for speeding up the tuning of the learner parameters (see, for instance, [23, 27]), we designed an ad hoc procedure, motivated by the need of keeping the size of the network as small as possible. It consists of the following steps:

1. set the hyperparameters to an initial configuration with low memory footprint;
2. train the CNN and validate it using a fivefold cross validation approach;

3. compare the average value of the F1-scores, resulting from the fivefold cross-validation, with thresholds expressing the minimum acceptable performance;
4. if the objective is reached, then stop, otherwise modify the hyperparameters into a more memory-demanding configuration[1], and go to step 2).

We applied several times such procedure, for each load. For example, for the electrovalves state classification, by fixing the minimum acceptable F1-score for both classes to 95%, and applying the above procedure, we obtained the following values:

$$s = 50,\ N_{\mathrm{hl}} = 3,\ w_1 = 5,\ w_2 = 4,\ w_3 = 3,\ \alpha = 2, \quad (6)$$

that result in a CNN having an F1-score of 97.65% for the ON class and of 99.82% for the OFF class. In particular, we report in Fig. 4a the average F1-scores obtained for some combinations of hyperparameters according to the above defined heuristic tuning procedure.

It is worth mentioning that the above procedure has been applied on both oversampled and undersampled training sets. Although undersampling has the advantage of leading to a smaller training set, and hence a faster training phase, oversampling has been finally preferred in terms of network performance. As an example, we report in Table 4 a comparison between the two methods, in terms of F1-score, referred to the electrovalves state classification. Apparently, undersampling caused the loss of relevant information that, in turn, determined a noticeable decrease of performance.

The memory footprint of the obtained CNN can be evaluated as follows.

The number of weights and biases to be stored for the i-th convolutional layer is

$$N_{\mathrm{cnv}}(i) = w_i \times N_{\mathrm{f}}(i) \times N_{\mathrm{f}}(i-1) + N_{\mathrm{f}}(i), \quad (7)$$

where $N_{\mathrm{f}}(i)$ and w_i are, respectively, the number and the width of the filters of the i-th convolutional layer, and $N_{\mathrm{f}}(0) = n_u$ is the number of inputs to the CNN. Similarly, the parameters to be stored for the i-th batch normalization layer are

$$N_{\mathrm{b}}(i) = 4 \times N_{\mathrm{f}}(i). \quad (8)$$

Indeed, each batch normalization layer requires the storage of the mean and the variance over the whole training set of the values coming from each filter of the previous layer. In addition, it requires the storage of an offset and a scale factor, resulting in four parameters per filter. Finally, the number of parameters of the fully connected layer is

[1] Each configuration is chosen empirically, taking into account reasonable constraints, such as the filter being shorter than the signal to be filtered.

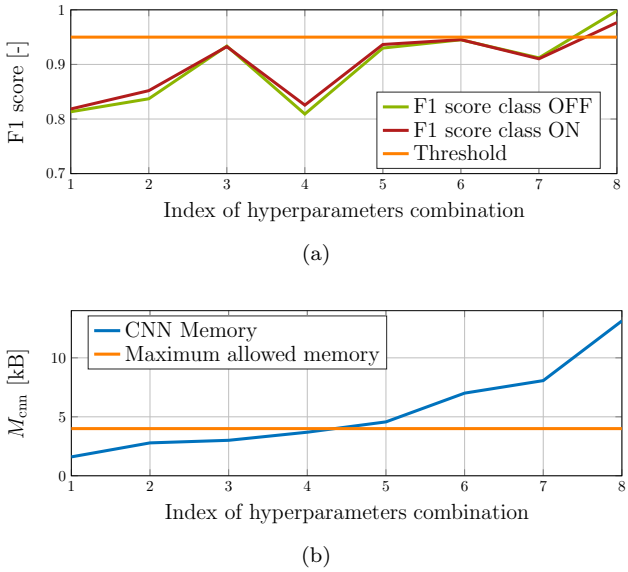

Fig. 4 F1-score (**a**) and related memory footprint (**b**) of the networks obtained during the heuristic procedure. Each index on the x-axis corresponds to a particular combination of hyperparameters values (for the sake of simplicity, eight combinations among the many that have been tested are reported). The only combination that exceeds the F1-score threshold is the 8-th, which corresponds to the values of the hyperparameters in Eq. (6). However, the CNN with these hyperparameters exceeds the available memory. We denote by $[s, \alpha, N_{hl}, (w_1, \ldots, w_{N_{hl}})]$ a hyperparameter combination. The index-hyperparameters association is the following: $1 = [20, 1, 2, (3, 2)]$, $2 = [20, 1, 3, (3, 2, 2)]$, $3 = [50, 1, 2, (5, 4)]$, $4 = [20, 2, 2, (3, 2)]$, $5 = [50, 1, 3, (5, 4, 3)]$, $6 = [50, 2, 2, (5, 4)]$, $7 = [20, 2, 3, (3, 2, 2)]$, $8 = [50, 2, 3, (5, 4, 3)]$ (colour figure online)

$$N_{\mathrm{fc}} = \left(\frac{s}{2^{Nhl-1}} + \sum_{i=1}^{Nhl} \frac{1 - w_i}{2^{Nhl-i}} \right) \times n_y \times N_{\mathrm{f}}(N_{\mathrm{hl}}) + n_y, \quad (9)$$

where s is the length of the input segments to the CNN, and n_y is the number of output classes and w_i is the width of the filters of the i-th convolutional layer. The latter affects the number of actual inputs to the fully connected layer, because, due to the absence of padding, each convolution results in a shorter signal than the original. All the remaining types of layers, namely max-pooling, softmax and classification, do not require to store any further parameter.

Therefore, assuming that the parameters are stored in single precision floating point arithmetic (32 bit), the memory size in kB required to store the CNN is:

$$M_{\mathrm{cnn}} = \frac{32}{8 \times 1024} \left[\left(\sum_{i=1}^{Nhl} N_{\mathrm{cnv}}(i) + N_{\mathrm{b}}(i) \right) + N_{\mathrm{fc}} \right]. \quad (10)$$

None of the well-performing CNNs obtained by applying several times the heuristic procedure described above led to the satisfaction of the memory constraints (see Fig. 4b). For example, in Fig. 5, the CNN memory footprint with

Table 4 CNNs performance with different classes balancing methods. The oversampling approach results to be the most effective solution

Method	F1 class ON	F1 class OFF
Undersampling	20.07%	83.72%
Oversampling	97.65%	99.82%

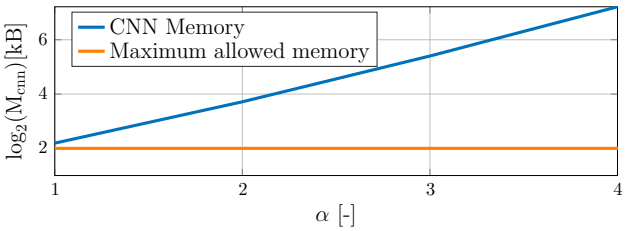

Fig. 5 Log-scale CNN memory footprint as a function of α (blue line). The orange line represents the available upper memory bound. The memory required by CNNs is always higher than the available one (colour figure online)

respect to α is shown (the remaining parameters are set as per Eq. (6)).

Thus, even if the classification of the load status can be solved by 1D-CNNs, the size of the obtained networks prevents their deployment on the available hardware.

6.2 LSTMs

The most important hyperparameter that has to be tuned for LSTM networks is the number of hidden units, that is, the state dimension of the network. To this aim, many methods, such as Bayesian optimization [26] and random search [13], can be found in literature. However, according to the problem statement reported in Sect. 2, it seems reasonable to compute the maximum state dimension allowed by the available memory. The memory required to store the weights and biases of an LSTM network is given by the following formula:

$$M_{\text{lstm}} = \frac{32}{8 \times 1024}[4\left(n_{\text{h}}n_{\text{u}} + n_{\text{h}}^2 + n_{\text{h}}\right) + n_{\text{h}}n_{\text{y}} + n_{\text{y}}] \tag{11}$$

where M_{lstm} is the memory footprint of the network (kB), 32 are the bits required to store a single precision number, n_{h} is the number of hidden units, n_{u} is the number of inputs and n_{y} is the number of outputs.

In Fig. 6, the memory required to store the network parameters in a logarithmic scale is shown ($n_{\text{y}} = 2$). The number of inputs is $n_u = 13$.

Due to memory constraints, the maximum number of hidden units that can be employed is 10. Hence, Bayesian

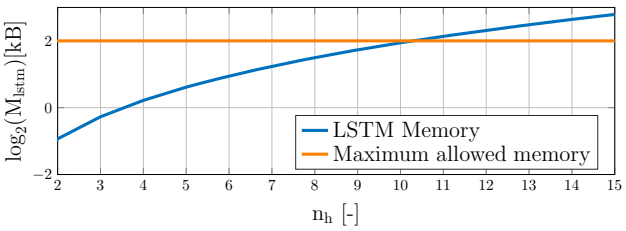

Fig. 6 Log-scale LSTM memory footprint as a function of the number of hidden units (blue line). The orange line represent the available upper memory bound. The memory required by LSTMs fulfills the constraints only for parameter values less than 10 (colour figure online)

optimization (as proposed in [26]) has been employed to find the optimal number of hidden units. The goal of the optimization problem is that of maximizing the F1-score. The optimization yielded the following result: $n_{\text{h}} = 10$. Being 10 the maximum allowable value, this result suggests that better performance could be achieved enlarging the state dimension. For the memory constraints, this is not possible; on the other hand, in the next section we show that the results are still satisfactory.

As already mentioned, to deal with the issue of class imbalance, the use of a class-balancing technique has been investigated, such as training-set segmentation paired with undersampling or oversampling (Sect. 3.2). Therefore, the training has been performed on segments using both the subsequence-to-subsequence and the subsequence-to-label approach. In subsequence-to-label, the training and the test sets have been segmented and the former undersampled (i.e., some random input–output pairs have been deleted from the training set in order to achieve a proportion of 50% of output labels). In subsequence-to-subsequence, undersampling is performed deleting random segments that contain only OFF samples. After this operation, the 50% of segments contain only OFF samples and the remaining 50% contain at least one ON. The test set does not require to be segmented since LSTMs can be used on sequences regardless their length. Hence, the test operation can be done on the full test sequences. Undersampling has been preferred, rather than oversampling, to avoid obtaining a too large training set.

7 Experimental Results

This section presents the experimental results obtained running the networks on the test dataset. The main hyperparameters of the networks have been set as explained in Sect. 6, while the parameters of the training algorithm, such as the initial learning rate (LR), its drop factor and its drop period (in epochs), are reported in Table 5.

Table 5 Other general parameters settings in the training algorithm

	CNN	LSTM	
		Subsequence-to-label	Subsequence-to-subsequence
Optimizer	Adam	Adam	Adam
n. epochs	20	20	120
Mini-batch size	4096	1024	8192
Initial LR	0.01	0.001	0.006
LR drop factor	–	–	0.8
LR drop period	–	–	20

Both CNNs and LSTMs have been trained (using the aforementioned hyperparameters and training algorithm settings) and tested.

We report the results, in terms of F1-scores, for both, although the CNN cannot be implemented in the target microcontroller due to memory size limitations. The results for the heater status detection, reported in Table 6, are very good for all the investigated networks. This comes at no surprise, since the heater status can be detected by only looking at the real part of the first current harmonic: when the heater is active it draws a large amount of active power from the grid, hence the first real current harmonic is particularly high.

Tables 7 and 8 also show the performance of the networks in charge of estimating the status of the drain pump and of the electrovalves, respectively. The results, in this case, show that:

1) the CNNs outperform the best performing LSTMs except in one case;
2) the subsequence-to-subsequence LSTM performs significantly better than the subsequence-to-label;

Table 6 F1-score performance on heater detection for CNN and LSTM models. The performance is comparable, but the former is discarded due to the violation of the memory constraint highlighted in Fig. 5

	CNN	LSTM	
		Subsequence-to-label	Subsequence-to-subsequence
F1 class ON	99.72%	98.95%	99.81%
F1 class OFF	99.95%	99.80%	99.66%

Table 7 F1-score performance on drain pump detection for CNN and LSTM. CNN and subsequence-to-subsequence LSTM have comparable performance. However, the former is discarded due to the violation of the memory constraint highlighted in Fig. 5. The resulting best model is the subsequence-to-subsequence LSTM

	CNN	LSTM	
		Subsequence-to-label	Subsequence-to-subsequence
F1 class ON	99.57%	85.39%	99.23%
F1 class OFF	99.75%	96.98%	99.87%

Table 8 F1-score performance on electrovalve detection for CNN and LSTM. CNN and subsequence-to-subsequence LSTM have comparable performance. However, the former is discarded due to the violation of the memory constraint highlighted in Fig. 5. The resulting best model is the subsequence-to-subsequence LSTM

	CNN	LSTM	
		Subsequence-to-label	Subsequence-to-subsequence
F1 class ON	98.39%	90.09%	98.05%
F1 class OFF	99.87%	99.19%	99.85%

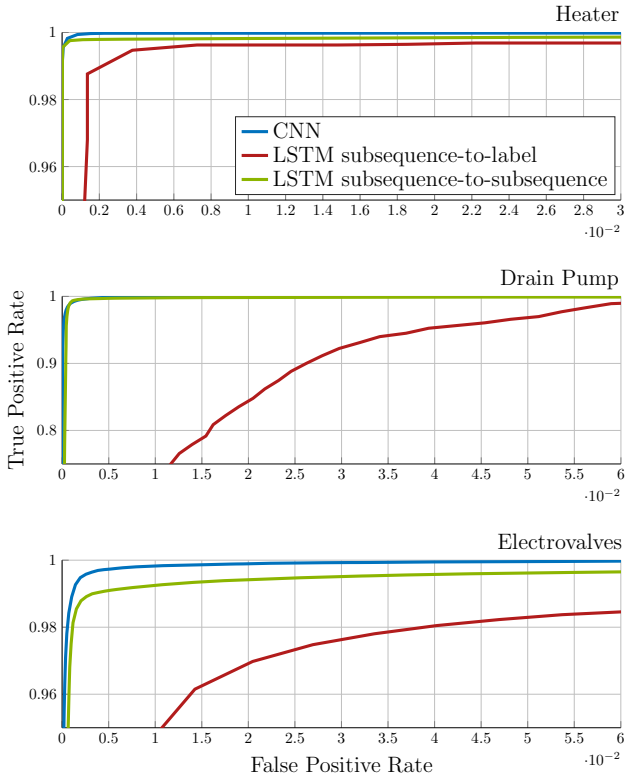

Fig. 7 Enlarged portions of ROC curves. Each subfigure compares the performance of the employed networks for a specific load. CNNs (blue) outperform both LSTMs in all the evaluated scenarios; while subsequence-to-subsequence LSTMs (green) perform better than subsequence-to-label LSTMs (red)

3) the performance of the best performing LSTMs are close to that of the CNNs, for the same task.

The Receiver Operating Characteristic (ROC) curves of each network, for the different load status estimation, are shown in Fig. 7.

The memory footprint of the CNN networks can be calculated using Eq. (10) with $N_{\text{cnv}}(1) = 528$, $N_{\text{cnv}}(2) = 528$, $N_{\text{cnv}}(3) = 1568$, $N_{\text{b}}(1) = 32$, $N_{\text{b}}(2) = 64$, $N_{\text{b}}(3) = 128$, $N_{\text{fc}} = 514$ and $N_{\text{hl}} = 3$, thus resulting in $M_{\text{cnn}} = 13.13\text{kB}$. Conversely, the memory footprint of the LSTM network can be calculated using Eq. (11) with $n_u = 13$, $n_h = 10$ and $n_y = 2$. The resulting required memory is $M_{\text{lstm}} = 3.84\text{kB}$. In summary, given the small difference in terms of performance, but the significantly smaller memory consumption, the LSTMs are preferable to the one-dimensional CNNs for the considered problem.

8 Conclusion

We have faced the problem of estimating the status of electrical loads of four models of washing machines from measurements of signals at the plug. To this aim, we have trained and tested two state-of-the-art machine learning tools, namely one-dimensional CNNs and LSTMs, on a dataset consisting of more than a thousand of hours of signals acquired during several hundreds of washing cycles. Besides classification performance, we took into account the tight constraints in terms of available memory on the target device. Our results show that, although both CNNs and LSTMs perform well on the test data, the former turns out to be too memory demanding given our hardware constraints, thus cannot be deployed. In addition, we have experimentally shown that oversampling and undersampling are effective in dealing with the high class imbalance that such datasets may exhibit.

Funding Open access funding provided by Università degli Studi di Trieste within the CRUI-CARE Agreement.

Declarations

Conflict of interest The authors declare that there is no conflict of interest.

References

1. https://www.arm.com/products/silicon-ip-cpu
2. Alvarez JM, Salzmann M (2017) Compression-aware training of deep networks. In: Advances in neural information processing systems, pp 856–867
3. Babichev A, Casagrande V, Della Schiava L, Fenu G, Fodor I, Marson E, Pellegrino FA, Pin G, Salvato E, Toppano M, Zorzenon D (2020) Loads estimation using deep learning techniques in consumer washing machines. In: Proceedings of the 9th international conference on pattern recognition applications and methods. La Valletta, pp 425–432
4. Basterretxea K, Tarela JM, Del Campo I (2004) Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons. IEE Proc Circuits Dev Syst 151(1):18–24
5. Batista GE, Prati RC, Monard MC (2004) A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explor Newslett 6(1):20–29
6. Buda M, Maki A, Mazurowski MA (2018) A systematic study of the class imbalance problem in convolutional neural networks. Neural Netw 106:249–259

7. Chakraborty I, Roy D, Ankit A, Roy K (2019) Efficient hybrid network architectures for extremely quantized neural networks enabling intelligence at the edge. arXiv preprint arxiv.org/abs/1902.00460
8. Della Schiava L, Marson E, Pin G, Posa P (2020) Smart plug and method for determining operating information of a household appliance by a smart plug, WO patent WO/2020/043737. https://patentscope.wipo.int/search/en/detail.jsf?docId=WO2020043737
9. Dey N. Fong S, Song W, Cho K (2017) Forecasting energy consumption from smart home sensor network by deep learning. In: International conference on smart trends for information technology and computer communications. Springer, pp 255–265
10. Gers FA, Eck D, Schmidhuber J (2002) Applying lstm to time series predictable through time-window approaches. In: Neural Nets WIRN Vietri-01. Springer, pp 193–200
11. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT press
12. Grangier D, Bottou L, Collobert R (2009) Deep convolutional networks for scene parsing. In: ICML 2009 deep learning workshop, vol 3. Citeseer, p 109
13. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J (2016) LSTM: a search space odyssey. IEEE Trans Neural Netw Learn Syst 28(10):2222–2232
14. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems, pp 1135–1143
15. Hannun AY, Rajpurkar P, Haghpanahi M, Tison GH, Bourn C, Turakhia MP, Ng AY (2019) Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nat Med 25(1):65
16. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
17. Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2017) Quantized neural networks: training neural networks with low precision weights and activations. J Mach Learn Res 18(1):6869–6898
18. Karim F, Majumdar S, Darabi H, Chen S (2017) Lstm fully convolutional networks for time series classification. IEEE Access 6:1662–1669
19. Kim J, Le TTH, Kim H (2017) Nonintrusive load monitoring based on advanced deep learning and novel signature. Comput Intell Neurosci
20. Kouretas I, Paliouras V (2019) Simplified hardware implementation of the softmax activation function. In: 2019 8th international conference on modern circuits and systems technologies (MOCAST), pp 1–4
21. Lee KB, Cheon S, Kim CO (2017) A convolutional neural network for fault classification and diagnosis in semiconductor manufacturing processes. IEEE Trans Semicond Manuf 30(2):135–142
22. Mocanu E, Nguyen PH, Gibescu M, Kling WL (2016) Deep learning for estimating building energy consumption. Sustain Energy Grids Netw 6:91–99
23. Ottoni AL, Nepomuceno EG, de Oliveira MS, de Oliveira DC (2020) Tuning of reinforcement learning parameters applied to sop using the Scott–Knott method. Soft Comput 24(6):4441–4453
24. Popa D, Pop F, Serbanescu C, Castiglione A (2019) Deep learning model for home automation and energy reduction in a smart home environment platform. Neural Comput Appl 31(5):1317–1337
25. San Kim T, Sohn SY (2020) Multitask learning for health condition identification and remaining useful life prediction: deep convolutional neural network approach. J Intell Manuf 1–11
26. Snoek J, Larochelle H, Adams RP (2012) Practical bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, pp 2951–2959
27. Solares JRA, Wei HL, Billings SA (2019) A novel logistic-narx model as a classifier for dynamic binary classification. Neural Comput Appl 31(1):11–25
28. Suda N, Loh D (2019) Machine learning on arm cortex-m microcontrollers. Arm Ltd., Cambridge
29. Ullrich K, Meeds E, Welling M (2017) Soft weight-sharing for neural network compression. arXiv preprint arxiv.org/abs/1702.04008
30. Weigend AS, Rumelhart DE, Huberman BA (1991) Generalization by weight-elimination with application to forecasting. In: Advances in neural information processing systems, pp 875–882
31. Zamanlooy B, Mirhassani M (2013) Efficient VLSI implementation of neural networks with hyperbolic tangent activation function. IEEE Trans Very Large Scale Integr (VLSI) Syst 22(1):39–48
32. Zębik M, Korytkowski M, Angryk R, Scherer R (2017) Convolutional neural networks for time series classification. In: Rutkowski L, Korytkowski M, Scherer R, Tadeusiewicz R, Zadeh LA, Zurada JM (eds) Artif Intell Soft Comput. Springer, Cham, pp 635–642
33. Zhao Z, Chen W, Wu X, Chen PC, Liu J (2017) Lstm network: a deep learning approach for short-term traffic forecast. IET Intel Transport Syst 11(2):68–75
34. Zheng Y, Liu Q, Chen E, Ge Y, Zhao JL (2014) Time series classification using multi-channels deep convolutional neural networks. In: Li F, Li G, Hwang S, Yao B, Zhang Z (eds) Web-age information management. Springer, Cham, pp 298–310