Received June 16, 2020, accepted June 27, 2020, date of publication July 3, 2020, date of current version July 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3007046

Li-Ion Batteries Parameter Estimation With Tiny Neural Networks Embedded on Intelligent IoT Microcontrollers

GIULIA CROCIONI[1], DANILO PAU[2], (Fellow, IEEE), JEAN-MICHEL DELORME[3], AND GIAMBATTISTA GRUOSSO[1], (Senior Member, IEEE)
[1]DEIB, Politecnico di Milano, 20133 Milano, Italy
[2]STMictroelectronics, 20864 Agrate Brianza, Italy
[3]STMictroelectronics, Cedex 38019 Grenoble, France

Corresponding author: Giambattista Gruosso (giambattista.gruosso@polimi.it)

ABSTRACT Lithium-ion (Li-Ion) batteries are rechargeable batteries which can maximize battery lifespan thanks to their chemical abilities, at the same time increasing power energy density. For these reasons, Li-Ion batteries have earned considerable popularity, and they are widely used both in mobile computing devices (e.g. smartphones and smartwatches) and automotive systems (e.g. hybrid and electric vehicles). A fundamental parameter for battery health monitoring is the State of Health (SoH), which is computed from the maximum releasable capacity, and which represents battery functionality in energy storage and delivery. Among the most used data-driven approaches are Machine Learning (ML) algorithms, such as Support Vector Machines (SVMs), Random Forest (RF) regressions, and Artificial Neural Networks (ANNs). This article presents a comparison of different ML algorithms for estimating maximum releasable capacity of Li-Ion batteries, with a special focus on the implementation of both Forward and Recurrent ANNs (FNNs and RNNs, respectively), using prognostic Li-Ion battery data sets provided by the National Aeronautics and Space Administration (NASA). After an evaluation of models performances in terms of RMSE and MAE, STM32Cube.AI tool was used to convert pre-trained ANNs to optimized ANSI C code for STM32 microcontrollers (MCUs), and to profile their complexity automatically. Finally, in order to decrease models size with minimal accuracy loss, the implemented ANNs were quantized via STM32Cube.AI, converting weights and activations from 32-bit floating-point to 8-bit integer precision. TensorFlow Lite for Microcontrollers (TFLM) was used as benchmark in the analysis and validation of both non-quantized and quantized models, and the performances obtained via STM32Cube.AI and TFLM were compared.

INDEX TERMS Battery modeling, neural networks, simulation, forecasting, micro-controller, support vector machine, capacity battery modeling, estimation, data-driven.

I. INTRODUCTION

Lithium-Ion (Li-Ion) batteries are increasingly used to power many systems, including portable devices, such as smartphones, and automotive systems, such as Hybrid Electric Vehicles (HEV) and Electric Vehicles (EV) [1]. These rechargeable batteries present several advantages in terms of high specific energy and power [2], outlasting the lifespan of traditional batteries, such as Nickel Metal Hydride (NiMH) [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Jonghoon Kim.

Evaluation and monitoring of battery health, safety, and reliability are performed through the Battery Management System (BMS) [4], in which the State of Health (SoH) is a fundamental parameter. In fact, SoH represents a measure of the battery remaining capacity, compared to the nominal one [5]:

$$SOH = \frac{C_{max}}{C_{rated}} 100\% \tag{1}$$

In the equation 1, C_{max} represents the maximal releasable capacity (in Ampere Hours) and measures the maximum amount of charge stored by the battery, while C_{rated} represents its nominal capacity, provided by the manufacturer.

The maximal releasable capacity can be computed by integrating the discharge current of the fully charged battery over time [6]:

$$C_{max} = \int_{0}^{t_{cutoff}} I_d \, dt \tag{2}$$

During a battery lifespan, its capacity decays even if it is not used, due to internal aging processes. Due to this phenomenon, performances decrease and the battery is typically no longer considered reliable after a 20% fading in its nominal value [7], [8]. Therefore, SoH and maximal releasable capacity estimations are essential to reduce safety risks and to prevent critical failures, allowing for appropriate battery replacement [9].

Several data-driven methods based on Machine Learning (ML) techniques have been developed for estimating SoH [10], [11]. Measurable battery parameters such as terminal voltage, current and temperature, can be extracted via the Controller Area Network (CAN) bus [12], and can be used for capacity fading and SoH estimation.

Weng *et al.* [13] developed a model based on Support Vector Regression (SVR), using partially charging data of a single Li-Ion cell to estimate capacity fading of other 7 cells. However, due to the same charge profile used for all data generation, the complexity of the estimation was reduced. Wei *et al.* [14] proposed a SVR-based model for batteries capacity fading prediction, based on one of the Li-Ion battery datasets made available by National Aeronautics and Space Administration (NASA) [15]. Also in this work, the data used for the forecast were only those generated by the same load profile. Additionally, although in real-world conditions the charge/discharge may not be complete, the entire charge/discharge cycles were used as model input. In the study of Li *et al.* [16] random forest regression was used to estimate online the capacity of Li-Ion batteries, again using complete charge/discharge cycles as model input. A random decision forest was proposed by Xu *et al.* [17] for battery health estimation, using one of the Li-Ion battery prognostic datasets made available by NASA [15]. The same batteries used for both training and test sets considerably reduced the complexity of capacity estimation.

Despite the high performances obtained through traditional algorithms, Deep Learning (DL) methods have proven to be more flexible and effective, due to the high non-linearity and non-direct observability of SoH. For example, a Convolutional Neural Network (CNN) was used to estimate SoH from raw battery measurements, obtaining excellent results. Reference [18] However, the same load profile used for data generation and the entire cycles of charge given as input limited the algorithm practicality. Then partial charge profiles were used for State of Charge (SoC) generation, which was given as input to the model together with voltage, current and temperature. However, since SoC is not directly observable and estimable, further errors are introduced a priori in the model [19]. Choi *et al.* [7] developed a Feedforward Neural Network (FNN), a CNN and a Long Short-Term Memory (LSTM) network to estimate the capacity of Li-Ion batteries from their voltage, current and temperature charge profiles. The dataset used is one of the Li-Ion battery datasets made available by NASA [15], and as in other studies the complexity of the forecast decreased due to the same load profile used for data generation. Another research work [20] proposes an Independently Recurrent Neural Network (IndRNN) for SOH estimation which uses randomized battery usage data collected from Li-Ion batteries adopting dynamic load profiles [15], obtaining a Root Mean Squared Error (RMSE) of 0.0133 and a Mean Absolute Error (MAE) of 0.0114. In this case the 18 input features are derived from voltage, current and temperature measured during a so-called Random Walk (random charge/discharge load profile). However, in a real use-case, all data of a Random Walk before being input to the network should be stored in memory, and then should be processed (e.g. computation of mean voltage), greatly limiting the possibility to use resource constrained hardware, such as microcontrollers (MCUs).

Despite the great success of the neural networks inference, these models typically contain several millions of parameters [21], making their storage and evaluation a considerable hardware bottleneck [22]. In fact, the reduction in neural networks memory requirements has recently aroused great interests from the ML community [23], [24]. In particular, running ML inference on MCUs allows to use Artificial Intelligence (AI) with several hardware devices reducing energy consumption, latency, processing power, memory and storage, without data living the device and thus preserving privacy. TensorFlow Lite for Microcontrollers is an example of tool found in the state of the art which makes possible to run ML models on MCUs with only KiloBytes (KB) of memory. TensorFlow Lite for Microcontrollers (TFLM) is an experimental port of TensorFlow Lite, a ML framework for embedded devices develped by Google, and it has been tested widely with the Arm Cortex-M Series architectures. [25]

In this article, we present different tiny neural networks architectures to predict the maximum releasable capacity of Li-Ion batteries, using the datasets made available by NASA [15], and comparing the models in terms of accuracy. Then, the pre-trained NNs were embedded into a STM32 microcontroller for low power and low memory embedded applications. The rest of the article is organized as follows. In Section II the architectures of the proposed tiny neural networks are described. Section III describes dataset preparation, predictive models settings and the boards and tools used for the complexity analysis. Section IV presents results and their evaluation. Finally, conclusions are summarized in Section V.

II. DESCRIPTION OF THE TINY ARCHITECTURES AND THEIR MAIN ADVANTAGES

The architectures used for estimating maximum releaseable capacity were CNN, LSTM, GRU (Gated Recurrent Unit), CNN LSTM, and CNN GRU. In fact, studies have shown that a CNN applied to time series can extract significant patterns by reducing noise [26]. Moreover, leveraging temporal and spatial structure of a CNN allows to learn complex input features [27]. Among Recurrent Neural Networks (RNNs), the LSTM architecture has been very successful with time series, and in particular with their long-term dependencies [28], [29]. Moreover, LSTM networks overcome the vanishing gradient problem, experienced by standard RNNs [30]. Another type of recurring units proposed more recently are the GRU units [31], which like the LSTM ones allow the RNNs employing them to perform well in tasks where long-term dependencies are important, such as time series [32]. Finally, given the complementarity of architectures that are well suited at temporal modelling (e.g. LSTM and GRU) and convolutive layers that reduce frequency variations [33], the aforementioned RNNs were combined with the CNN architecture.

A. CONVOLUTIONAL NEURAL NETWORK

The implemented CNN consists of a 1D convolutional layer, two subsequent hidden dense layers, and a last dense output layer. The input data are provided in the format that CNN expects, i.e. the three-dimensional one: number of samples, time steps (20), and features (4). The filters with which the convolutional layer is initialized are 32, of size 4 × 4. After this first layer, for internal covariate shift problems reduction and for regularization, Batch Normalization [34] is applied. Then, after the Rectified Linear unit (ReLu) activation function, the flattened output goes through two fully connected layers, the first one with 32 neurons and the second one with 16. Both layers use the ReLu activation function. Then, 20% of the dense layers input units were randomly dropped for overfitting reduction [35]. The last layer consists of only one node, and gives the network output. The proposed CNN architecture is shown in Figure 1.

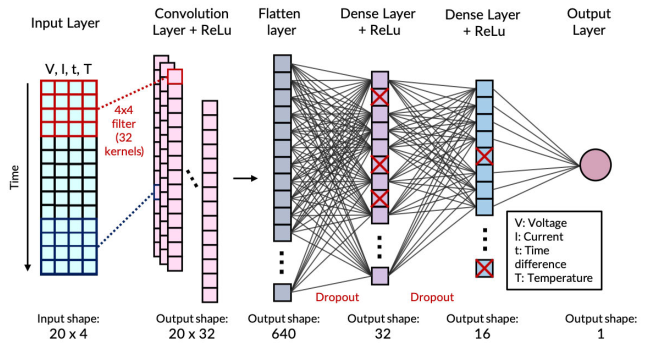


FIGURE 1. The implemented CNN Architecture consists of a 1D convolutional layer, a flatten layer, two subsequent hidden dense layers, and a last dense output layer. Input data are provided in three-dimensional format: number of samples, time steps (20), and features (4).

B. LONG SHORT-TERM MEMORY NETWORK

The proposed LSTM architecture consists of a LSTM layer and a dense output layer. The input data are provided in the format that LSTM expects, i.e. the three-dimensional one: number of samples, time steps (20), and features (4). The first layer consists of 50 LSTM output units [36], and uses the Hyperbolic Tangent as activation function. The network output is given by the dense output layer, which consists of a single node. The proposed LSTM architecture is shown in Figure 2.

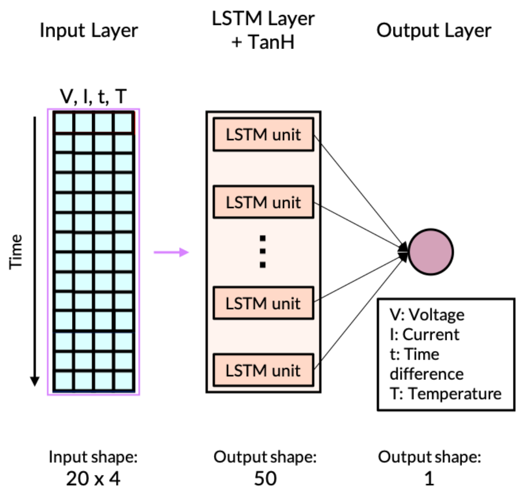


FIGURE 2. The implemented LSTM Architecture consists of a LSTM layer and a dense output layer. Input data are provided in three-dimensional format: number of samples, time steps (20), and features (4).

C. CONVOLUTIONAL AND LONG SHORT-TERM MEMORY NETWORK

In the implemented CNN LSTM, the convolutional layer aims at input features extraction, while the LSTM layer supports sequence prediction. The input data are provided in the format that CNN expects, i.e. the three-dimensional one: number of samples, time steps (20), and features (4). The filters with which the convolutional layer is initialized are 32, of size 4 × 4, and after output normalization it uses the ReLu activation function. The obtained feature maps may be sensitive to the location of the features in the input [38]. Thus, a max pooling layer is used for the selection of the maximum of each pair of values, halving the feature maps and summarizing the features in the input. Then, a LSTM layer with 32 output units [36] and with TanH activation function is used. The network output is given by the dense output layer, which consists of a single node. The proposed CNN LSTM architecture is shown in Figure 3.

D. GATED RECURRENT UNITS NETWORK

The proposed GRU architecture consists of a GRU layer and a dense output layer. The input data are provided in the format that GRU expects, i.e. the three-dimensional one: number of samples, time steps (20), and features (4). The first layer consists of 16 GRU output units, and uses the Hyperbolic Tangent as activation function. The network output is given by the dense output layer, which consists of a single node. The proposed GRU architecture is shown in Figure 4.

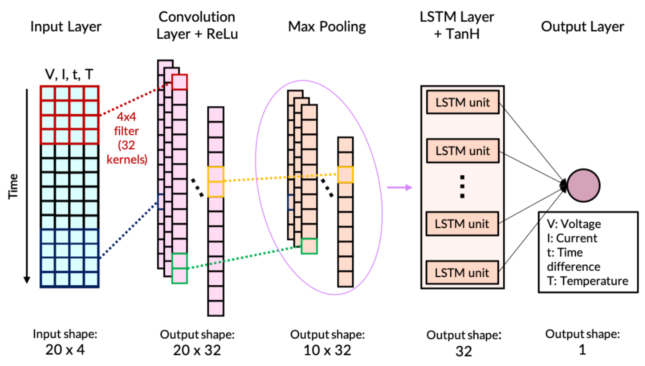


FIGURE 3. The implemented CNN LSTM Architecture consists of a 1D convolutional layer, a max pooling layer, a LSTM layer and a dense output layer. Input data are provided in three-dimensional format: number of samples, time steps (20), and features (4).

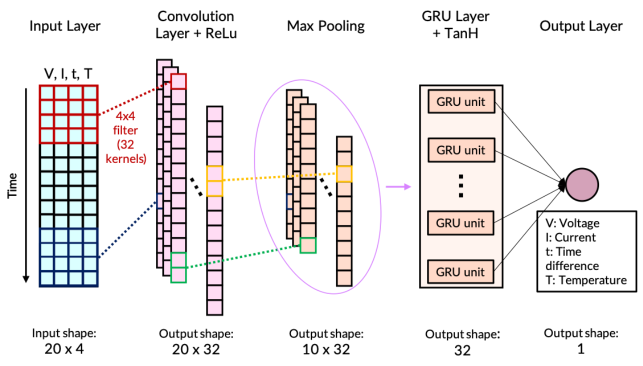


FIGURE 5. The implemented CNN GRU Architecture consists of a 1D convolutional layer, a max pooling layer, a GRU layer and a dense output layer. Input data are provided in three-dimensional format: number of samples, time steps (20), and features (4).

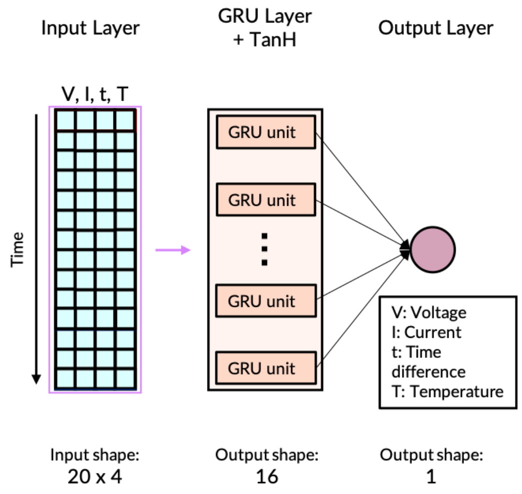


FIGURE 4. The implemented GRU Architecture consists of a GRU layer and a dense output layer. Input data are provided in three-dimensional format: number of samples, time steps (20), and features (4).

E. CONVOLUTIONAL AND GATED RECURRENT UNITS NETWORK

The implemented CNN GRU consists of a convolutional layer followed by a max pooling layer, a GRU layer and a dense output layer. The input data are provided in the format that CNN expects, i.e. the three-dimensional one: number of samples, time steps (20), and features (4). The filters with which the convolutional layer is initialized are 32, of size 4×4, and after output normalization it uses the ReLu activation function. The GRU layer has 32 output units and uses TanH activation function. The network output is given by the dense output layer, which consists of a single node. The proposed CNN GRU architecture is shown in Figure 5.

III. PROPOSED METHODOLOGY

A. PROBLEM DESCRIPTION AND DATA PREPROCESSING

The dataset chosen for the analysis application is one of those provided by NASA Ames Prognostics Center of Excellence (PCoE) [15], and it contains measurements of voltage, current, temperature, capacity and impedance of Li-Ion batteries, generated through charge, discharge and impedance operational profiles. Charge and discharge tests ended when the batteries reached the 30% fade in rated capacity (from 2 A h to 1.4 A h). The data generation mode used fixed reference charge and discharge profiles, and the resulting dataset is particularly suited to the estimation of the remaining capacity, as evidenced by many studies we have based on, which used this dataset to estimate the remaining capacity [7], [14], [17], [37]. Due to the frequent detection of incorrectly entered or measured values in some sets during qualitative data observation, battery sets number 1, 2, 3, 4 and 6 were selected for further analysis. For example, those batteries whose capacity values assume zero values after the first cycles, but then return to almost the initial nominal value, were not included in the case study.

In data generation tests, the charge profile was applied by keeping the current constant at 1.5 A until a voltage of 4.2 V was reached, then keeping the latter constant until 20 µA was reached (as reference see Figure 6). Instead, the discharge profiles are different for each battery set, and Table 1 shows their main differences in terms of discharge current [A], current shape, and temperature [°C]. Figure 7 shows discharge cycles of three different batteries as functions of voltage [V] and SoC [p.u.].

TABLE 1. Main discharge conditions for each selected dataset.

Dataset	Discharge current [A]	Current shape	Temperature [C°]
#1	2	Constant current level	24
#2	4	Square wave, 50% duty cycle at 0.05 Hz	24
#3	4	Constant current level	43
#4	1	Constant current level	4
#6	2	Constant current level	4

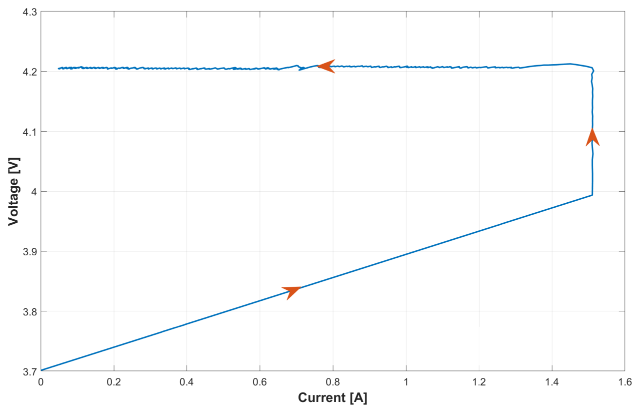


FIGURE 6. **Charging cycle used for data generation tests for all the batteries of the dataset. The charge profile was applied by keeping the current constant at 1.5 A until a voltage of 4.2 V was reached, then keeping the latter constant until 20μA was reached.**

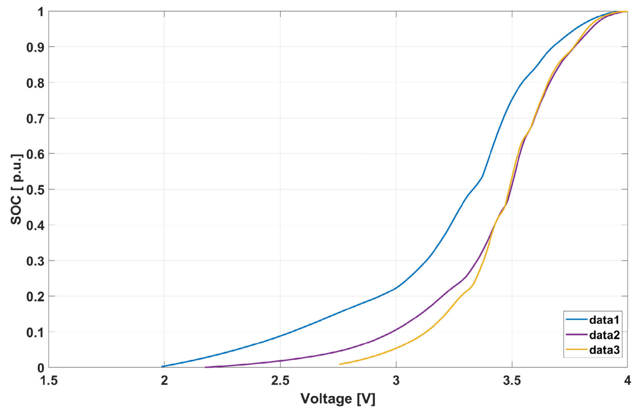


FIGURE 7. **Discharging cycles of three different batteries [15]. The cycles are shown as functions of voltage [V] and SoC [p.u.].**

Because of the greater variability of experimental conditions during the discharge cycles, which makes them closer to real-use cases and increases the complexity of the modelling, only these cycles were used for further analysis. In order to compute battery releasable capacity, the discharge current is integrated over time starting from a fully charged battery [6]. However, a full discharge may not be possible in real cases. Therefore, only some samples for each discharge cycle were selected, and form the dataset for the presented analysis. In particular, battery output current, terminal voltage, time difference between samples and temperature are the four features selected for each discharge cycle. The target of the estimation is the capacity of the considered cycle.

Considering all the selected batteries, the total number of observations (discharge cycles and capacity values) was 1462. In order to create an independent test set for model performance evaluation, one battery per set, namely battery number 05 (set 1), 27 (set 2), 30 (set 3), and 46 (set 4) was inserted in the test set (22 % of the total data), while the rest was used as training set. In addition to it, after data shuffling 20% of the training set was randomly sampled to form the validation set, used for model loss evaluation and parameters tuning [39].

B. PREDICTIVE MODELS

Before the NNs embedding in the MCU, the presented models were trained and tested using the sets described in Subsection III-A. The Adaptive Moment Estimation (Adam) optimization algorithm was used to train to networks due to its efficiency in terms of adaptive learning rate method and its suitability to non-stationary problems [40]. The Mean Squared Error (MSE) is the default loss function for regression problems, and it was chosen as the function to minimize. Since the datasets used in the state of the art were different and had several limitations (as discussed in more detail in Section IV-A) which influenced the prediction accuracy, a direct comparison of the results obtained with those of the state of the art was not possible.

Thus, a Random Forest (RF) model and an SVR model were evaluated using the aforementioned dataset, and then were used as a baseline for the developed architectures. RF model makes predictions using multiple decision trees that find the partition to which the new input belongs, choosing between subspaces with small variations [41]. The number of model trees has been set to 4 to make it comparable in size to the developed neural networks. SVR applies Support Vector Machine (SVM) theory to regression tasks. In SVMs, kernel functions map input data to a higher dimensional feature space, and a separating hyperplane maximizes the margin [42]. The kernel chosen for the model was the linear one, since its results were better than ones obtained with the Polynomial and Radial Basis Function (RBF) kernels. Both in neural networks and baseline models, MinMaxScaler was used to scale the features and to preserve the original data distribution shape [43].

C. COMPLEXITY PROFILING

The trained neural networks were embedded into ARM Cortex-M4 MCU, integrated on a SensorTile Wireless Industrial Node (STWIN). It was chosen since it is very suitable for industrial applications which need condition monitoring, being provided with several industrial IoT sensors, an ultra-low-power MCU (STM32L4R9ZIJ6), modular architecture, and a 480 mA h Li-Po battery. In addition to this, BLE wireless and WiFi connectivity on-board are supported. The memory embedded in the core system MCU consists in 2 Mbytes of Flash and 640 Kbytes of SRAM. [44] The tool used for the pre-trained architectures conversion was STM32Cube.AI software tool (version 5.0.0), which automatically converts the models into optimized ANSI C code for embedded MCUs, and allows their performances analysis, running the validation both on PC and on the chosen MCU.

Then, the tool was used to generate 8-bit integer quantized models which reduce run-time models size, improving both hardware latency and power consumption. In particular, each NN weights and associated activations are converted from 32-bit floating-point to 8-bit integer precision, with little

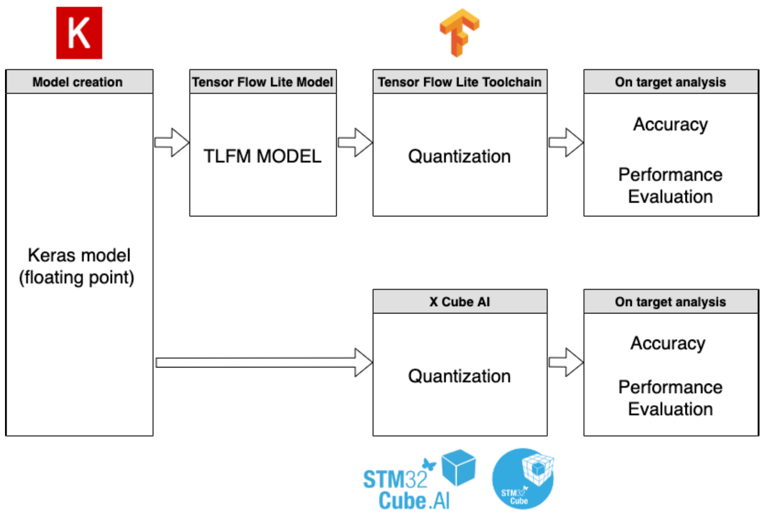


FIGURE 8. Functional Diagram of the main steps of the quantization process in TFLM and STM32Cube.AI.

degradation in model accuracy. The algorithm used to generate the quantized tensors is "Minmax", which bases the quantization process on min and max values of all weights and activations tensors. The arithmetic used is the integer arithmetic, which allows a more efficient implementation than floating point inference, and which is based on a representative convention used by Google for quantized models [45]. For the integer arithmetic, the real numbers r are mapped to 8-bit integer numbers q according to the following equation, where the scale factor is an arbitrary positive real number:

$$q = \frac{r}{scale} + zero\_point \tag{3}$$

The quantized values q are linearly distributed around the zero point, and precision depends on the scale factor. Weights format can be asymmetric (i.e. zero point not fixed) or symmetric (zero point equal to 0). In the latter case, it is possible to limit cost of operations by using kernel optimization implementations. On the other hand, symmetric format for activations is not supported due to their asymmetric nature. Post-quantized TensorFlow Lite models use symmetric format and signed integer type for the weights, and asymmetric format and signed integer type for the activations. Thus, the same scheme (ss/sa) was chosen for the quantization process. Quantization of recursive layers (i.e. LSTM and GRU layers) is not supported, and therefore their execution remains in floating-point. Reference [46] Also quantized models were analysed, running the validation both on PC and on the chosen MCU, and compared with non-quantized models performances.

Finally, in order to benchmark TFLM (r2.2), the main state-of-the-art tool comparable with STM32Cube.AI, CNN model was analysed and validated using both STM32Cube.AI and TFLM tools. Benchmarking was performed using only the CNN model, as current versions of STM32Cube.AI and TFLM do not support recurrent layers. In fact, quantization of recursive layers resulted in a drastic loss of overall accuracy, and for both STM32Cube.AI and TFLM the support for the 8-bit quantization is still evolving [46], [47]. Both non-quantized and quantized models were embedded into three boards (i.e. Nucleo-H743ZI @480MHz, Nucleo-F411 @100MHz, and Nucleo-L4r5ZI @120MHz), and the performances obtained via STM32Cube.AI and TFLM were compared. In order to perform the analysis with TFLM, CNN model was converted from *.h5* format to TensorFlow Lite format (*.tflite*), before being quantized with TensorFlow post training quantization. In both processes, the quantization scheme used was ss/sa. The main steps of STM32Cube.AI and TFLM quantization processes are shown in Figure 8.

IV. RESULTS AND EVALUATION

A. DATA PREPROCESSING

The choices made for the dataset aim to overcome the limits found in the datasets used in the state of the art. Among them, Wei *et al.* [14], Xu *et al.* [17], Li *et al.* [16] used entire charge/discharge cycles. In these cases the estimation of capacity is simplified and does not reproduce real world conditions, in which a complete cycle may not be available. In the dataset used for this work, only some samples were inserted for each signal. Weng *et al.* [13], Wei *et al.* [14], Xu *al.* [17], Choi *et al.* [7] used datasets whose data were generated using the same load profile. Instead, the dataset used in this work has only the discharge cycles, due to the greater variability of their experimental conditions. Additionally, Xu *et al.* [17] reduced the complexity of capacity prediction by using the same batteries in both the test and training phases. In contrast, in the dataset used in this article the batteries inserted in the test set and in the training set are different from each other, increasing the prediction complexity and the similarity to a real use case.

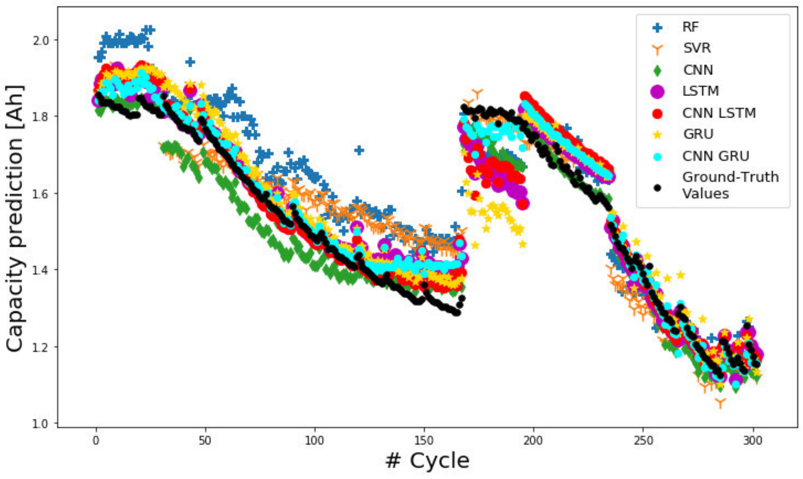


FIGURE 9. Developed models capacity predictions [Ah], using the test set described in the Subsection III-A. The predicted values are superimposed on the ground-truth values in black, which are the ones extracted from the NASA dataset [15].

Finally, the satisfactory results obtained by Prakash and Vigneswaran [20] cannot be compared to the ones presented in this work, due to the different methodologies used for network input features extraction. In fact, while Prakash *et al.* collected data during a random charge/discharge load profile and then processed them to extract input features, implying large data storage and off-line data processing, in this article only some samples for each cycle were selected, and they were directly used as input features for the networks, making cheap real-time SOH estimation possible.

B. PREDICTIVE MODELS

The prediction accuracy of the developed architectures was evaluated using RMSE and MAE metrics. Since ML algorithms usually use randomness during the training procedure, such as in initialization of NNs weights, and in the input features partitioning for splitting points in RF, they can produce different results at each run. Thus, the models were trained and tested 10 times each, using a different value for the pseudo-random number generator. The metrics obtained for each model are shown in Table 2 in terms of mean value and maximum variation obtained from it. NNs gave better results than the two models chosen as baseline, RF and SVM, and in particular CNN GRU outperformed other NNs architectures, with maximum RMSE and MAE of 0.0488 and 0.0414, respectively, producing errors bellow the acceptable SOH error range of ± 0.05 for EVs [20]. The total number of parameters for each of the proposed architectures is shown in Table 3, and it can be noticed that CNN GRU was almost the model with the lowest number of parameters, exceeded only by GRU network.

TABLE 2. Capacity estimation errors for the proposed neural networks architectures.

Model	RMSE	MAE
RF	0.1308±0.0152	0.1120±0.0109
SVR	0.0861±0.0023	0.0716±0.0002
CNN	0.0720±0.0153	0.0543±0.0106
LSTM	0.0708±0.0056	0.0482±0.0052
CNN LSTM	0.0633±0.0112	0.0434±0.0001
GRU	0.1394±0.0021	0.1101±0.0120
CNN GRU	**0.0486±0.0002**	**0.0404±0.0010**

TABLE 3. Number of parameters of the proposed neural networks architectures.

Model	Total number of parameters
CNN	21729
LSTM	11051
CNN LSTM	9025
CNN GRU	6945
GRU	1025

Capacity predictions obtained during test phase, before MCU embedding, are shown in Figure 9, superimposed on the ground-truth values. Figure 10 shows prediction residuals $r = y - \hat{y}$ for each developed model, where y represents the true capacity values, and $\hat{y}$ the predicted ones. It can be noticed that the models prediction error of cycles from 167 to 194 is larger than in the rest of the test set, meaning that in this interval the models are not able to capture relationships well enough. In fact, these predictions refer to test set battery number 27 (set 2, see Subsection III-A), whose set is the only one with a squared discharge current wave, and has the least data in the training set, representing only 7% of the total.

C. COMPLEXITY PROFILING

The results of NNs complexity analysis obtained via STM32Cube.AI are shown in Table 4, in terms of Random Access Memory (RAM), Flash, and Multiplier ACCumulator (MACC) operations in 32-bit floating points. Then the

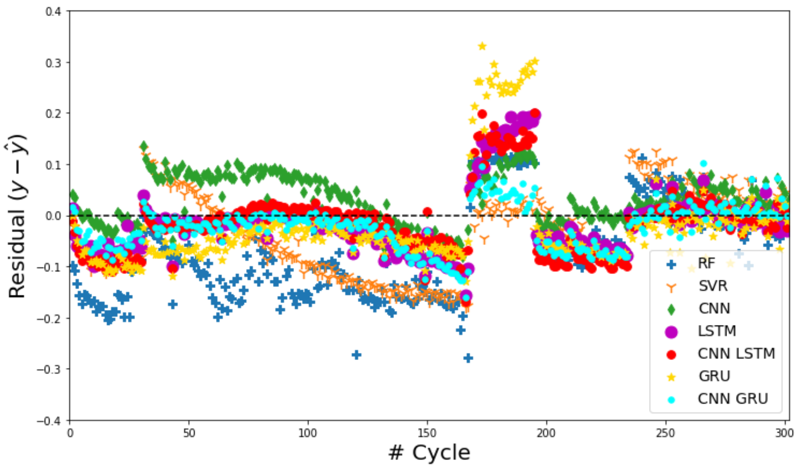


FIGURE 10. Capacity predictions residuals [Ah] of the developed models, using the test set described in the Subsection III-A.

TABLE 4. Results of Neural Networks complexity analysis.

Model	RAM [kB]	Flash [kB]	MACC [fp32]
CNN	2.690	86.40	31968
LSTM	0.200	44.80	221050
CNN LSTM	1.540	35.97	95104
GRU	0.064	4.10	19856
CNN GRU	1.540	27.27	73664
Total	6.034	198.54	441642

TABLE 5. Neural Networks on-board evaluation, using test set data.

	Evaluation status	RMSE	MAE
CNN	STM32 C-model	0.0720	0.0543
	Original model	0.0720	0.0543
	Cross-accuracy (Reference vs C-model)	0.0000	0.0000
LSTM	STM32 C-model	0.0708	0.0482
	Original model	0.0708	0.0482
	Cross-accuracy (Reference vs C-model)	0.0000	0.0000
CNN LSTM	STM32 C-model	0.0633	0.0434
	Original model	0.0633	0.0434
	Cross-accuracy (Reference vs C-model)	0.0000	0.0000
GRU	STM32 C-model	0.1394	0.1101
	Original model	0.1394	0.1101
	Cross-accuracy (Reference vs C-model)	0.0000	0.0000
CNN GRU	STM32 C-model	0.0486	0.0404
	Original model	0.0486	0.0404
	Cross-accuracy (Reference vs C-model)	0.0000	0.0000

pre-trained models were loaded to the chosen MCU, and the test set mentioned in Subsection III-A was used to validate them. The NNs obtained evaluations are shown in Table 5. RMSE and MAE for STM32 C-model and for the original one are shown for each model, together with cross-accuracy, defined considering the original model outputs as ground truth values for the C-model outputs. All original models are correctly converted to ANSI C code, keeping their performances unchanged.

Afterwards, the aforementioned models were quantized and the complexity profiling obtained through the validation on the chosen MCU was compared with non-quantized models one. In particular, Figure 11 shows a comparison of the weights size [KB] of non-quantized models (floating-point models) with those of quantized ones. Other comparisons are shown in Figure 12, in terms of RAM size [KB], and in Figure 13, in terms of mean CPU cycles needed for a single inference divided for the number of MACC operations.

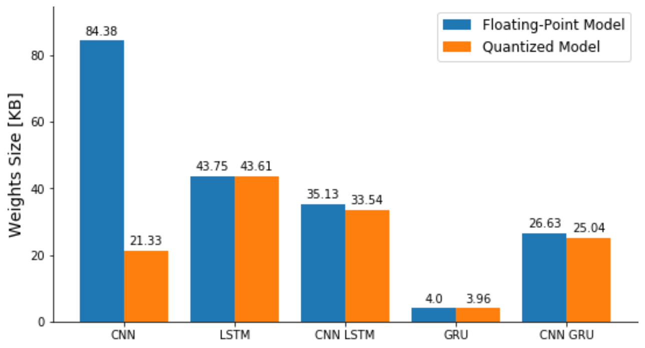


FIGURE 11. Weights size [KB] for floating-point model in blue and for quantized model in orange. The results were obtained from the validation of the models on the MCU described in the Subsection III-C.

In all comparisons, it can be noted that the quantized CNN model is characterized by a drastic drop in weights and RAM sizes, and in ratio between CPU cycles and MACC operations. In particular, in CNN quantized model weights and RAM size are reduced by more than half compared to the non-quantized one. Cycles/MACC also decrease considerably, being about halved. Also for the quantized CNN LSTM and CNN GRU there is an improvement in performances, reducing both memory usage and single inference

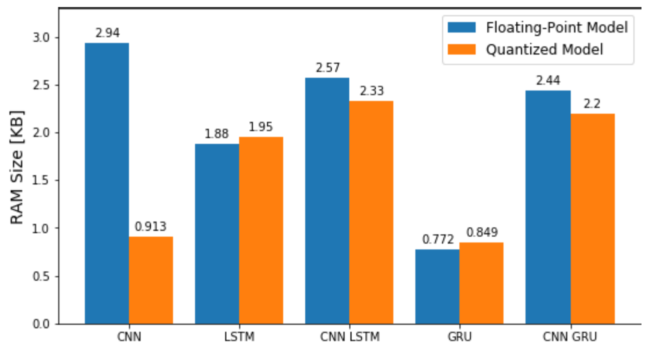


FIGURE 12. Total RAM size [KB] for floating-point model in blue and for quantized model in orange. The results were obtained from the validation of the models on the MCU described in the Subsection III-C.

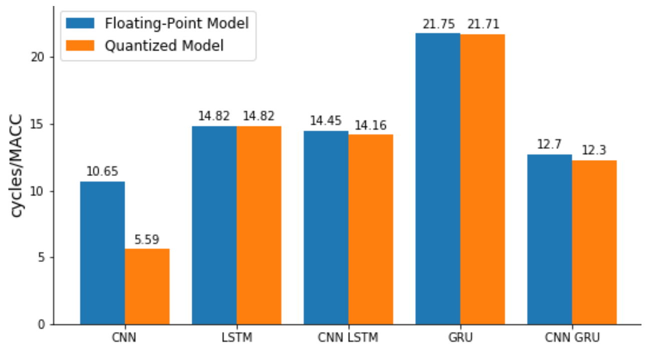


FIGURE 13. Single inference CPU cycles / MACC for floating-point model in blue and for quantized model in orange. The results were obtained from the validation of the models on the MCU described in the Subsection III-C.

mean cycles/MACC required, but in these cases the drop is less drastic than in CNN model. This behavior is due to the fact that CNN LSTM and CNN GRU models are composed of recurrent layers, which as already explained in the Subsection III-C are not quantized, and therefore their execution remains in floating-point. Finally, LSTM and GRU quantized models performances are slightly worse than the non-quantized ones. In fact, both LSTM and GRU models are composed of two layers in series: while the first layer is recurrent, and produces activations in floating-point precision since its quantization is not supported, the second layer is dense, and accepts activations in 8-bit integer precision since it is quantized. Between these two layers, a quantization layer which converts the activations from floating-point to 8-bit integer precision is needed, decreasing the quantized models performances. Table 6 shows that the improvement in terms of models sizes, CPU cycles and MACC operations required is at the expense of a slight degradation in the quantized models accuracy. In particular, the accuracy loss in the quantized CNN model is the greatest (i.e. 0.0334 additional RMSE and 0.0331 additional MAE), since the absence of recurrent layers makes the quantization complete, thus increasing accuracy loss. In all other networks, the accuracy of the quantized models decreases by less than 1 percentage point.

TABLE 6. Degradation in quantized models accuracy.

Quantized Model	Additional RMSE	Additional MAE
CNN	0.0334	0.0331
LSTM	0.0033	0.0030
CNN LSTM	0.0022	0.0017
GRU	0.0030	0.0029
CNN GRU	0.0052	0.0041

TABLE 7. Inference energy consumption estimation E_I for both non-quantized and quantized CNN models, computed according to [49] modeling and energy cost data.

	32-bit Energy Cost [nJ]	8-bit Energy Cost [nJ]
E_{MACC}	0.00460	0.00023
E_M	0.00920	0.00046
E_I	399.07941	18.65420

TABLE 8. Inference time ([ms], average of 16 iterations) of CNN model, using TFLM and STM32Cube.AI.

Board	Tool	Average inference time [ms]
Nucleo H743ZI		Non-quantized model
	TFLM	0.714
	X-CUBE-AI	0.423
		Quantized model
	TFLM	0.242
	X-CUBE-AI	0.203
Nucleo F411		Non-quantized model
	TFLM	6.161
	X-CUBE-AI	2.821
		Quantized model
	TFLM	2.084
	X-CUBE-AI	1.573
Nucleo L4r5ZI		Non-quantized model
	TFLM	5.546
	X-CUBE-AI	2.534
		Quantized model
	TFLM	1.949
	X-CUBE-AI	1.443

NNs energy consumption is crucial in battery constrained embedded applications [48], and the use of a quantized model can be very convenient in reducing energy costs. In order to compare the energy cost of the non-quantized CNN model with the quantized one, the inference energy consumption was modeled according to Horowitz [49], considering a typical processing hardware platform for CNNs [48], [50]. For the sake of simplicity, we provide a reference numerical comparison sourcing the energy cost data of a 45 nm processor technology from [49]. The energy consumption of the local SRAM read/write operation is modeled as the energy E_{MACC} of a single MACC operation, while it models the access to the main SRAM as $E_M = 2 \times E_{MACC}$. The cost of bias additions, batch normalization and similar operations is considerded to be E_{MACC}. Reference [48] Table 7 shows the inference energy consumption estimation E_I for both non-quantized and quantized CNN models. The inference cost of the quantized model resulted more than one order of magnitude lower than the non-quantized one.

Finally, Tables 8 and 9 show the performances results obtained from STM32Cube.AI v5.0.0 and TFLM r2.2 validation for both non-quantized and quantized CNN model, for each of the three MCUs used. In particular, Table 8 shows the average inference time ([ms], average of 16 iterations), and for all three boards the models processed with STM32Cube.AI perform slighlty better. Also Table 9 shows that the results obtained via STM32Cube.AI outperform TFLM ones, in this case in terms of CNN model size (RAM [KB] including inputs buffer, and Flash [KB]). The summary Table 10 shows the gain percentage in STM32Cube.AI complexity profiling results, compared to those obtained in TFLM.

TABLE 9. CNN model size in terms of RAM [KB] including inputs buffer, and Flash [KB], using TFLM and STM32Cube.AI.

Tool	Memory	Non-quantized model size [KB]	Quantized model size [KB]
TFLM	RAM	5.120	1.280
	Flash	88.692	25.504
X-CUBE-AI	RAM	2.880	0.912
	Flash	86.404	21.844

TABLE 10. Savings [%] in STM32Cube.AI complexity profiling results compared to TFLM ones.

	X-Cube-AI vs TFLM savings [%]
Inference time [ms]	Nucleo H743ZI: 16.12
	Nucleo F411: 24.52
	Nucleo L4r5ZI: 25.96
RAM size [KB]	28.75
Flash size [KB]	14.35

V. SUMMARY AND CONCLUSION

In this article, we showed different NNs architectures (i.e. CNN, LSTM, CNN LSTM, GRU, and CNN GRU) that predict the maximum releasable capacity of Li-Ion batteries, from which SoH can be easily computed. Li-Ion batteries datasets made available by NASA [15] were used for both the training and the testing phase. The final dataset was created with the aim of overcoming the limits found in the state of the art (e.g. the application of the same load profile for data generation), as widely motivated in the Subsection IV-A.

According to the results of the predictions, CNN GRU exceeded all other NNs, with maximum RMSE and MAE of 0.0488 and 0.0414, respectively, and all NNs gave better results than the baseline models (i.e. RF and SVM). Then, via STM32Cube.AI tool, the pre-trained models were incorporated into ARM Cortex-M4 MCU, and integrated on a STWIN, converting them into optimized ANSI C code. The NNs were validated both on PC and MCU, resulting in unchanged performances. The NNs models were also quantized, converting weights and associated activations from 32-bit floating-point to 8-bit integer precision, with a slight degradation in the models accuracy. The results obtained in the complexity profiling showed that weights size [KB], RAM size [KB], and Cycles/MACC decreased significantly in the quantized CNN model, and less drastically in the quantized CNN LSTM and CNN GRU models, due to their recurrent layer which is not supported for quantization. Finally, TFLM tool was benchmarked analysing and validating both non-quantized and quantized CNN models via STM32Cube.AI, whose results outperformed TFLM ones in terms of average inference time [ms], RAM size including inputs buffer [KB], and Flash size [KB].

REFERENCES

[1] Y. Nishi, "Lithium ion secondary batteries; past 10 years and the future," *J. Power Sources*, vol. 100, nos. 1–2, pp. 101–106, Nov. 2001.

[2] V. Marano, S. Onori, Y. Guezennec, G. Rizzoni, and N. Madella, "Lithium-ion batteries life estimation for plug-in hybrid electric vehicles," in *Proc. IEEE Vehicle Power Propuls. Conf.*, Dearborn, MI, USA, Sep. 2009, pp. 536–543.

[3] Q. Miao, L. Xie, H. Cui, W. Liang, and M. Pecht, "Remaining useful life prediction of lithium-ion battery with unscented particle filter technique," *Microelectron. Rel.*, vol. 53, no. 6, pp. 805–810, Jun. 2013.

[4] H. Rahimi-Eichi, U. Ojha, F. Baronti, and M.-Y. Chow, "Battery management system: An overview of its application in the smart grid and electric vehicles," *IEEE Ind. Electron. Mag.*, vol. 7, no. 2, pp. 4–16, Jun. 2013.

[5] M. Murnane and A. Ghazel, "A closer look at state of charge (SOC) and state of health (SOH) estimation techniques for batteries," *Analog Devices*, vol. 2, pp. 426–436, May 2017.

[6] A. Kirchev, "Battery management and battery diagnostics," in *Electrochemical Energy Storage for Renewable Sources and Grid Balancing*. Amsterdam, The Netherlands: Elsevier, 2015, pp. 411–435, ch. 20.

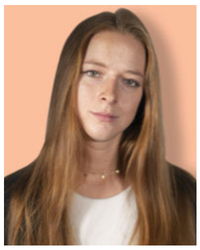
[7] Y. Choi, S. Ryu, K. Park, and H. Kim, "Machine learning-based lithium-ion battery capacity estimation exploiting multi-channel charging profiles," *IEEE Access*, vol. 7, pp. 75143–75152, 2019.

[8] *Lithium-ion Battery Datasheet, Battery Model, LIR18650 2600mAh*, EEMB Co., Ltd., Moscow, Russia, 2010.

[9] A. Farmann, W. Waag, A. Marongiu, and D. U. Sauer, "Critical review of on-board capacity estimation techniques for lithium-ion batteries in electric and hybrid electric vehicles," *J. Power Sources*, vol. 281, pp. 114–130, May 2015.

[10] X. Hu, J. Jiang, D. Cao, and B. Egardt, "Battery health prognosis for electric vehicles using sample entropy and sparse Bayesian predictive modeling," *IEEE Trans. Ind. Electron.*, vol. 63, no. 4, pp. 2645–2656, Apr. 2016.

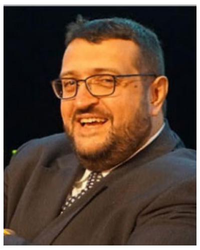
[11] L. Lu, X. Han, J. Li, J. Hua, and M. Ouyang, "A review on the key issues for lithium-ion battery management in electric vehicles," *J. Power Sources*, vol. 226, pp. 272–288, Mar. 2013.
[12] M. Zheng, B. Qi, and H. Wu, "A li-ion battery management system based on CAN-bus for electric vehicle," in *Proc. 3rd IEEE Conf. Ind. Electron. Appl.*, Jun. 2008, pp. 1180–1184.
[13] C. Weng, Y. Cui, J. Sun, and H. Peng, "On-board state of health monitoring of lithium-ion batteries using incremental capacity analysis with support vector regression," *J. Power Sources*, vol. 235, pp. 36–44, Aug. 2013.
[14] J. Wei, G. Dong, and Z. Chen, "Remaining useful life prediction and state of health diagnosis for lithium-ion batteries using particle filter and support vector regression," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5634–5643, Jul. 2018.
[15] B. Saha and K. Goebel. *Battery Data Set, NASA Ames Prognostics Data Repository*. Accessed: Oct. 10, 2019. [Online]. Available: http://ti.arc.nasa.gov/project/prognostic-datarepository
[16] Y. Li, C. Zou, M. Berecibar, E. Nanini-Maury, J. C.-W. Chan, P. van den Bossche, J. Van Mierlo, and N. Omar, "Random forest regression for online capacity estimation of lithium-ion batteries," *Appl. Energy*, vol. 232, pp. 197–210, Dec. 2018.
[17] H. Xu, Y. Peng, and L. Su, "Health state estimation method of lithium ion battery based on NASA experimental data set," *IOP Conf., Mater. Sci. Eng.*, vol. 452, Dec. 2018, Art. no. 032067.
[18] E. Chemali, "Intelligent state-of-charge and state-of-health estimation framework for Li-ion batteries in electrified vehicles using deep learning techniques," Ph.D. dissertation, Dept. Elect. Comput. Eng., McMaster Univ., Hamilton, ON, Canada, 2018.
[19] R. Zhang, B. Xia, B. Li, L. Cao, Y. Lai, W. Zheng, H. Wang, and W. Wang, "State of the art of lithium-ion battery SOC estimation for electrical vehicles," *Energies*, vol. 11, no. 7, p. 1820, Jul. 2018.
[20] P. Venugopal and V. T., "State-of-Health estimation of li-ion batteries in electric vehicle using IndRNN under variable load condition," *Energies*, vol. 12, no. 22, p. 4338, Nov. 2019.
[21] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. Xu Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "GPipe: Efficient training of giant neural networks using pipeline parallelism," 2018, *arXiv:1811.06965*. [Online]. Available: http://arxiv.org/abs/1811.06965
[22] S. Nimit Sohoni, R. Christopher Aberger, M. Leszczynski, J. Zhang, and C. Ré, "Low-memory neural network training: A technical report," Stanford Univ., Stanford, CA, USA, Tech. Rep., 2019.
[23] 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," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.
[24] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*. [Online]. Available: http://arxiv.org/abs/1710.01878
[25] (Mar. 31, 2020). *TensorFlow Lite for Microcontrollers*. Accessed: Apr. 22, 2020. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers
[26] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," 2017, *arXiv:1703.04691*. [Online]. Available: http://arxiv.org/abs/1703.04691
[27] M. Ibrahim, M. Torki, and M. Elnainay, "CNN based indoor localization using RSS time-series," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 01044–01049.
[28] I. Sutskever, "Training recurrent neural networks," Ph.D. dissertation, Univ. Toronto, Toronto, ON, Canada, 2013.
[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural-computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[30] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. 23rd Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn.*, 2015.
[31] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*. [Online]. Available: http://arxiv.org/abs/1406.1078
[32] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: http://arxiv.org/abs/1412.3555
[33] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 4580–4584.
[34] R. Ilango. (2019). *Batch Normalization—Speed Up Neural Network Training*. Accessed: Oct. 24, 2019. [Online]. Available: https://medium.com/@ilango100/batch-normalization-speed-up-neural-network-training-245e39a62f85
[35] J. Xiong, K. Zhang, and H. Zhang, "A vibrating mechanism to prevent neural networks from overfitting," in *Proc. 15th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2019, p. 1929.
[36] S. Verma. (2019). *Understanding Input and Output shapes in LSTM—Keras*. Accessed: Mar. 10, 2020. [Online]. Available: https://medium.com/@shivajbd/understanding-input-and-output-shape-in-lstm-keras-c501ee95c65e
[37] B. Saha and K. Goebel, "Modeling Li-ion battery capacity depletion in a particle filtering framework," in *Proc. Annu. Conf. Prognostics Health Manage. Soc. (PHM)*, Jan. 2009, pp. 1–10.
[38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[39] C. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford Univ. Press, 1995, p. 372.
[40] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–15.
[41] L. Breiman, "Random forests," in *Machine Learning*. Norwell, MA, USA: Kluwer, 2001, pp. 5–32.
[42] C.-H. Wu, J.-M. Ho, and D. T. Lee, "Travel-time prediction with support vector regression," *IEEE Trans. Intell. Transp. Syst.*, vol. 5, no. 4, pp. 276–281, Dec. 2004.
[43] J. Hale. (2019). *Scale, Standardize, or Normalize With Scikit-Learn*. Accessed: Oct. 31, 2019. [Online]. Available: http://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02
[44] *STWIN SensorTile Wireless Industrial Node Development Kit and Reference Design for Industrial IoT Applications, Data brief, Rev. 1*, STMicroelectronics, Geneva, Switzerland, 2019.
[45] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017, *arXiv:1712.05877*. [Online]. Available: http://arxiv.org/abs/1712.05877
[46] *Online Documentation—Command Line Interface X-CUBE-AI Expansion Package, R1.2 - AI PLATFORM R5.0.0 (Embedded Inference Client API 1.1.0)—Command Line Interface R1.2.0, 2019, Available in X-CUBE-AI Expansion Package*. [Online]. Available: http://Documentation/command_line_interface.html
[47] E. Torti, A. Fontanella, M. Musci, N. Blago, D. Pau, F. Leporati, and M. Piastra, "Embedded real-time fall detection with deep learning on wearable devices," in *Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2018, pp. 405–412.
[48] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," 2017, *arXiv:1711.00215*. [Online]. Available: http://arxiv.org/abs/1711.00215
[49] M. Horowitz, "Energy table for 45 nm process," in *Stanford VLSI Wiki*. 2014. [Online]. Available: http://vlsiweb.stanford.edu
[50] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–247.



GIULIA CROCIONI was born in Castiglione del Lago, Italy, in 1995. She received the B.S. and M.S. degrees in biomedical engineering from the Politecnico di Milano, Italy, in 2019, and the M.S. degree in bioengineering from the University of Illinois at Chicago (UIC), IL, USA, in 2019. She was a Teaching Assistant in neural engineering with the Neural Engineering Laboratory, UIC. She was also an Intern with the Research and Development Artificial Intelligence Tools and Application Team, STMicroelectronics. Since 2020, she has been a Data Scientist with the Brain Team - Big Data and Analytics, Allianz SpA and specializing with politecnico di milano as Data scientist. Her main research interests include advanced data analysis, machine learning, and the IoT.

DANILO PAU (Fellow, IEEE) received the B.S. and M.S. degrees in electronic engineering from the Politecnico di Milano, Italy, in 1992. In 1991, he joined the System Research and Applications Department, STMicroelectronics. He worked on Research and Development subjects, such as memory reduced HDMAC and MPEG video decoding, video coding, video transcoding, embedded 3D and 2D graphics, computer vision and deep learning, and transforming them into company products. He founded and served the First Chairman of the STMicroelectronics Technical Staff Italian Community. Since 2019, he has been an Industry Ambassador Member with the Action for Industry, the IEEE Region 8 South Europe. He is currently a Technical Director and a Fellow Member of Technical Staff position. His scientific production consists of 83 articles. He holds over 78 granted patents. He has 23 invited talks/seminars at various universities and conferences. He has contributed with 113 documents to the development of Compact Descriptors for Visual Search (CDVS), the group that successfully developed ISO-IEC 15938-13 MPEG Standard. He was the Funding Chair of MPEG Ad Hoc Group on Compact Descriptor for Video Analysis (CDVA), formerly Compact Descriptors for Video Search (CDViS). He is the Vice-Chair with the Task Force on Intelligent Cyber-Physical Systems, the IEEE Computational Intelligence Society.

JEAN-MICHEL DELORME was born in Champigny-sur-Marne, France, in 1967. He received the Engineering degree in electronic engineering and the master's degree in vision for robotic from the University of Clermont-Ferrand (Clermont II), France, in 1993, and the Ph.D. degree in implementation of neural network on massively parallel machine for speaker recognition application from the Research Laboratory, University of Clermont-Ferrand. In 1996, he was a Software Application Engineer with the Hard Disk Drive Control Domain, STMicroelectronics. From 2001 to 2012, he was a Senior Software Designer and an Architect in multi-media framework domain for smartphone application with STMicroelectronics and ST-Ericsson. In 2013, he was a Security Software Designer to develop an advanced open-source trusted execution environment. Since 2016, he has been a Senior Software Designer with the Research and Development Micro-Controller Department, STMicroelectronics, to support the embedded applications in sensor, machine learning, and the IoT domains.



GIAMBATTISTA GRUOSSO (Senior Member, IEEE) was born in 1973. He received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in electrical engineering from the Politecnico di Torino, Italy, in 1999 and 2003, respectively. From 2002 to 2011, he was an Assistant Professor with the Department of Electronics and Informatics, Politecnico di Milano, where he has been an Associate Professor, since 2011. He is the author of more than 80 papers on Journals and conferences on the topics. He does research in electrical engineering, electronic engineering, and industrial engineering. His main research interests include electrical vehicles transportation electrification, electrical power systems optimization, simulation of electrical systems, digital twins for smart mobility, factory and city, and how they can be obtained from data.