Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A study on the application of TensorFlow compression techniques to Human Activity Recognition

CHIARA CONTOLI[1], EMANUELE LATTANZI[1]
[1]Department of Pure and Applied Sciences, University of Urbino, Piazza della Repubblica 13, 61029 Urbino, IT (e-mail: chiara.contoli@uniurb.it, emanuele.lattanzi@uniurb.it)

Corresponding author: Chiara Contoli (e-mail: chiara.contoli@uniurb.it).

Chiara Contoli is a researcher co-funded by the European Union - PON Research and Innovation 2014-2020.

ABSTRACT In the human activity recognition (HAR) application domain, the use of deep learning (DL) algorithms for feature extractions and training purposes delivers significant performance improvements with respect to the use of traditional machine learning (ML) algorithms. However, this comes at the expense of more complex and demanding models, making harder their deployment on constrained devices traditionally involved in the HAR process. The efficiency of DL deployment is thus yet to be explored. We thoroughly investigated the application of TensorFlow Lite simple conversion, dynamic, and full integer quantization compression techniques. We applied those techniques not only to convolutional neural networks (CNNs), but also to long short-term memory (LSTM) networks, and a combined version of CNN and LSTM. We also considered two use case scenarios, namely cascading compression and stand-alone compression mode. This paper reports the feasibility of deploying deep networks onto an ESP32 device, and how TensorFlow compression techniques impact classification accuracy, energy consumption, and inference latency. Results show that in the cascading case, it is not possible to carry out the performance characterization. Whereas in the stand-alone case, dynamic quantization is recommended because yields a negligible loss of accuracy. In terms of power efficiency, both dynamic and full integer quantization provide high energy saving with respect to the uncompressed models: between 31% and 37% for CNN networks, and up to 45% for LSTM networks. In terms of inference latency, dynamic and full integer quantization provide comparable performance.

INDEX TERMS Compression techniques, Deep learning, Dynamic quantization, ESP32, Full integer quantization, Human Activity Recognition, Microcontrollers, Pruning

I. INTRODUCTION

IN the last years, the Internet of Things (IoT) paradigm has been widely adopted to develop wearable IoT systems thanks to the ever-increasing interest in many fields of applications, such as health care monitoring, fitness tracking, gaming, and virtual and augmented reality, to name a few. According to Grand View Research [1], smart wearable technology is driving industry growth, with a global wearable technology market valued at USD 61.30 billion in 2022 and is expected to expand at a compound annual growth rate of 14.6% from 2023 to 2030.

Human activity recognition (HAR) is a hot research topic since the beginning of the 2000s when first works on the detection of posture and motion via accelerometry were published [2]. Indeed, the ability to acquire and identify a wide range of activities allows for an increase in people's life quality depending on the application area. Human activities are typically distinguished into three categories according to the most recent literature by Arshad *et al.* [3]: daily living, real-time and user activities. Daily living activities (e.g., eating, brushing teeth, sleeping) are further distinguished into static (e.g, sitting) and dynamic (e.g., running). Real-time are defined by Arshad *et al.* as those related to healthcare and surveillance, whereas user activities as those related to individual or group activities. Regardless of the category, HAR has always been considered a classification problem.

The success gained by machine learning (ML) for classification problems such as, for example, speech recognition

and face detection [4, 5] encouraged the research community to adopt this approach also to the HAR use case with successful results in terms of recognition accuracy [6]. The subsequent adoption of deep learning (DL) further increased performance and brought the benefit of automatic features extraction, a key stage of HAR because of the correlation between activity recognition performances and the extraction of relevant features. However, increased performances are paid in terms of more complex deep models which demand higher computational requirements and increased memory, thus making the deployment of such deep models on embedded-constrained devices harder. This is because embedded devices are typically characterized by very hard constraints in terms of memory, computation, and energy.

The deployment of deep models on embedded devices is challenging. A recent attempt to achieve this goal is to leverage tiny machine learning (TML), an approach tailored to design and develop tiny ML and DL models on resource-constrained devices. Two strategies carried out to implement TML are hardware acceleration and approximation techniques. Previous work in the area of hardware acceleration leveraged central processing unit (CPU), graphic processing unit (GPU), application-specific integrated circuit (ASIC), and field programmable array (FPGA) to design and implement embedded ML. In [7], low-power, ultra low-power, and powerful embedded devices for ML at the edge are listed and grouped by their GPU, CPU, acceleration capacity, ML usage, and application examples. A comprehensive review of the state-of-the-art tools and techniques for efficient edge inference is provided by Shuvo *et al.* in[8]. In their paper, the authors highlight mainly four research directions for efficient DL inference on edge devices: i) novel DL architecture and algorithm design; ii) optimization of existing DL methods; iii) development of algorithms hardware codesign; and iv) efficient accelerator design for DL deployment.

The alternative strategy, approximation techniques, leverages techniques that have been well investigated in the literature, and consist of applying so-called compression techniques such as pruning and quantization to reduce network complexity. To target a microcontroller (MCU) environment, Federov *et al.* combine neural architecture search with pruning to automatically design convolutional neural networks (CNNs) that meet MCU constraints [9]. Authors in [10] make use of the TensorFlow Lite (TFL) converter to deploy CNNs on an Arduino Nano 33 BLE. A combined adoption of the two strategies is also used: in [11], authors propose a hardware-software framework to accelerate ML inference on edge devices using a modified TFL for Microcontroller models running on an MCU and a dedicated Neural Processing Unit (NPU) custom hardware accelerator. It exploits pruning to reduce computational complexity. It is worth mentioning that most of the literature explores the approximation of CNN only.

Despite the blooming research activity around DL deployment on MCU, the investigation of HAR on low-power MCU is yet to be explored. In this paper, we use TensforFlow compression techniques to investigate the efficiency of deep learning networks on a very resource-constrained wearable device in the context of human activity recognition. We consider not only the CNN network but also the long short-term memory (LSTM) network and a combined version of the CNN with the LSTM. The goal is to characterize the networks in terms of classification accuracy, inference latency, and energy consumption.

To provide a thorough investigation, we consider the simple lite conversion, pruning, dynamic quantization, and full integer quantization as TensorFlow compression strategies, which we apply in two different use case-scenario: i) cascading compression, ii) stand-alone compression. The goal of this work is twofold: on the one hand, we want to perform an in-depth analysis that compares the deployment of relevant DNNs on a target MCU; on the other one, we want to provide insights on: i) the feasibility of the deployment of those networks on the target MCU; ii) a comparison in terms of classification performance, inference time, and power-efficiency; iii) our experience in the adoption of Keras and TensorFlow in the design and deployment process of deep learning networks for HAR on MCU.

We test our approach on a real low-constrained device, the ESP32, on top of which we developed a sensor-based HAR application leveraging Espressif ESP32-wroom-32 DevKit [12]. Our results show that the cascading compression scenario is not feasible because of errors either at design time or at execution time on the ESP32, thus preventing carrying out the performance investigation. Instead, in the stand-alone scenario, the dynamic quantization proves to be the best solution because of the negligible loss of accuracy, with respect to other techniques. In terms of power efficiency, both dynamic and full integer quantization provide a higher energy saving with respect to the simple lite conversion: between 31% and 37% for CNN networks, and up to 45% for LSTM networks. In terms of inference latency, dynamic and full integer quantization provide comparable performance. Our work shows that multiple network types can be deployed onto an ESP32 device, each showing interesting performance parameters.

In the rest of the paper, we first provide some background about deep learning for HAR, compression techniques, and Keras and TensorFlow, and review the literature about the state-of-the-art of HAR on MCU in Section II. Section III details the design of our presented methodology and all considered network types. Later, in Section IV we describe the experimental setup. We then detail experimental results and provide a complete discussion in Section V and VI, respectively. Finally, we conclude the paper in Section VII with some final considerations.

II. BACKGROUND AND RELATED WORK

HAR gained a lot of attention in recent years because of the broad range of real-world applications. Early diagnosis, rehabilitation, and patient assistance can be provided in medical decision processes for healthcare monitoring purposes;

industrial applications, gaming, and sport/fitness tracking are of great interest as well. Two main approaches are leveraged for HAR: camera-based and sensor-based recognition. Camera and inertial sensors allow to detect a set of daily human activities via computer vision techniques and acceleration/location sensors, respectively.

A. DEEP LEARNING FOR HUMAN ACTIVITY RECOGNITION

HAR consists of identifying several types of physical movements carried out daily, such as fitness activities (running, jogging, walking), or behavioral activities (eating, brushing teeth) to cite a few. Two main approaches are leveraged for HAR: camera-based and sensor-based recognition. Camera and inertial sensors allow to detect a set of daily human activities via computer vision techniques and acceleration/location sensors, respectively.

The very first approach to HAR comes from the learning theory, which leverages classification algorithms (such as support vector machines) to identify patterns. Since HAR is considered a multi-class classification problem, such an approach showed promising results. However, a limitation of this type of method is that the phase of feature selection, which is carried out prior to the classification algorithm employment, must be performed manually, typically by a domain expert.

Machine learning algorithms have been adopted for a long time to handle HAR problems; however, despite the successful results obtained with the application of machine learning [13], given the complexity of the HAR task, the research community shifted from a shallow learning approach to a deep learning approach. Deep learning allows automatic features selection and offers higher performances. Given that sensor-based HAR is employed in the majority of the studies, recent literature is rich in work proving the superiority of the combined adoption of convolutional neural networks (CNNs) with long short-term memory (LSTM) networks [14, 15, 16, 17], compared to their stand-alone adoption.

The convolutional neural network was born as a deep network specialized for image recognition[18]. The typical architecture envisages one or more convolutional layers followed by one or more pooling layers. Convolutional and pooling layers are responsible for the automatic features extraction during the training process. Convolutional layers contain kernels and filters; the former is responsible for the input conversion in the convolution operation, whereas the latter is responsible for setting the number of output filters during the convolution. The classification network is typically composed of one or more fully connected layer (that follows the pooling one) that produces the output.

The long short-term memory network (LSTM) is a type of recurrent neural applied in several application domains such as time series prediction, natural language processing, text recognition, and computer vision, to cite a few[19]. Despite several versions of LSTM architectures exist, the so-called vanilla LSTM turns out to be the most popular one. In such an architecture, a LSTM unit consists of a cell in charge of keeping values over time, and of three gates. The input, the forget, and the output gate are in charge of adding, removing, and using as output the information in the cell state.

A combined adoption of CNN and LSTM allows for capturing spatial and temporal features, respectively, thus providing higher accuracy. Most of these works are tailored to propose new deep neural network models with the aim of boosting performances, typically increasing accuracy and reducing inference latency, memory footprint, and energy/resource consumption. Also, in the majority of these studies those models are assumed to run either at the edge of the network, such as mobile phones or gateways [20], or on power-full servers possibly in cloud [21].

Deep networks deployment in wearable resource-constrained devices is hard because of the models' complexity and incurs higher memory occupation and execution time. As well surveyed in [22], there are mainly three approaches that allow bringing deep learning (DL) models to mobile devices, also known as optimization methods: i) model compression techniques, ii) software acceleration, and iii) hardware acceleration. As mentioned in the survey, DL in (human) activity recognition has been carried out and well-investigated on smartphones, and partially on embedded devices such as Raspberry Pi. However, the adoption of model compression techniques on more resource-constrained devices, such as microcontrollers, is still poorly investigated.

B. COMPRESSION TECHNIQUES

Two well-known compression techniques are pruning a quantization, which Liang *et al.* recently surveyed in terms of methods of compression and mathematical formulation [23]. Pruning consists of removing unnecessary parameters or neurons, and connections because do not provide a significant contribution to resulting accuracy. As of today, pruning can be distinguished depending on various aspects that are considered during the operation. In particular, three categories exist: i) whether the pruned network is symmetric or not, it is classified as structured or unstructured pruning; ii) based on the pruned element type, it is classified as neurons or connections pruning; iii) based on when pruning steps are carried out, i.e., after training but before inference, or during the inference process, it is classified as static pruning or dynamic pruning, respectively. In this work, we consider the static pruning technique only.

Quantization consists of reducing weight representation by reducing bit width numbers, typically from floating point values to integer values. The most widely used quantization techniques are: i) post-training quantization, which envisages the model training followed by weight quantization, and as a last step a model (re)optimization to generate the quantized model; ii) quantization-aware training, which envisages the weight quantization during training, and then the network is re-trained to fine-tune the model precision to compensate the accuracy degradation occurred during the quantization process.

A further classification of quantization techniques can be done by identifying where quantization takes place, i.e., layer-wise or channel-wise. In this work, we consider post-training quantization techniques such as dynamic quantization and full integer 8-bit quantization.

C. TENSORFLOW AND KERAS

TensorFlow (TF) is an end-to-end machine learning platform powered by Google [24]. TF provides tools and libraries to process and load data, build custom models or leverage existing ones, and run and deploy models on several environments, including production systems. A detailed description of TF's core components and design decision is provided by Bo *et al.* in [25]; basically, a TF program consists of two main parts: the construction part, which allows building the machine (or deep) learning network model, and the execution part, which allows to train and evaluate the network model. To achieve both parts TF provides an Application Programming Interface (API); the most widely used API (because of its completeness and stability) is provided in the Python language. However, other supported languages are Java, C++, Go, and Swift.

TF provides also TensorFlow Lite (TFL)[26] a library for network models deployment on mobile devices, microcontrollers, and edge devices. In particular, TFL allows converting a base TF model into a compressed version via the so-called TFLite converter. Keras [27] is a deep learning framework, built on top of TF version 2, which provides a Python API to allow developers to simplify network model creation and experimentation. In this work, we used Keras, TF, and TFL to carry out our investigation.

D. HAR ON MICROCONTROLLERS

The investigation of HAR on low-power microcontroller units (MCUs) blossomed in the last few years: back in 2020, Novac *et al.* evaluated the implementation of multi-layer perceptrons and convolutional neural networks for HAR on an ARM-Cortex-M4F-based MCU [28]. In particular, they compared the supervised learning methods to the unsupervised and online learning ones, by proving the higher benefits of the latter. Authors refer to online learning as the ability of a neural network to adapt itself to a new set of data, even though the initial learning phase is over. The ultimate goal was to study the trade-off between classification performance and embedded implementation constraints while taking advantage of unsupervised and online learning. Compared to this work, we both used TensorFlow Lite for microcontrollers and the UCI-HAR dataset for validation purposes. However, they did not use any optimization techniques, i.e., they did not explore the adoption of any compression techniques but the simple lite conversion. They explored the deployment on a SparkFunEdge board, whereas we explored the deployment on another target MCU.

Subsequently, Novac *et al.* further explored the deployment of HAR on MCU by proposing a new quantization method, together with a new framework that allows training, quantizing, and deploying deep neural networks [29]. Their MicroAI framework provides neural network training based on Keras or PyTorch, and a conversion tool that produces a portable C code starting from a trained model. They only consider convolutional neural networks (specifically, the ResNetv1 model architecture); int8 quantization-aware training, int16 post-training quantization, and float32 were considered as quantization techniques, and SparkFun Edge and Nucleo-L452RE-P were considered as MCU platforms (both belonging to the Cortex-M4F core family). The ultimate goal was to provide an alternative to existing frameworks, such as the open-source TensorFlow Lite for microcontrollers and the proprietary STM32Cube, and to provide an improvement to existing optimization techniques. Compared to this work, we adopted other compression techniques compared to their one, and we investigated multiple types of deep neural networks compared to them, which only evaluated the ResNet architecture.

Daghero *et al.* applied Binary Neural Networks (BNNs) to HAR to decrease network complexity via an extreme form of quantization [30]; indeed, by using BNNs the precision of data format, both weights and layers input/output, is reduced to 1-bit precision. Specifically, the authors propose a BNN inference library that targets RISC-V processors; such an approach allows to target of very compact networks in terms of channels, i.e., future maps, in each layer of the network. Subsequently, authors extended their work[31, 32] by proposing a set of efficient one-dimensional convolutional neural networks (1D CNNs) and testing optimization techniques such as sub-type and mixed-precision quantization. The aim was to find a good trade-off between accuracy and memory occupation. As a target platform, they leveraged again the RISC-V MCU. Compared to these works, we leveraged other compression techniques and tested deployment on a different target MCU.

Similarly, Ghibellini *et al.* proposed a CNN model for falling and running detection within an industrial environment for safety purposes. This work shows very preliminary results in terms of accuracy and model size. dynamic range quantization was applied to reduce the size of the model which is then deployed on the firmware of an Arduino BLE 33 Sense [33].

To sum up the comparison with the existing literature, besides the one-dimensional convolutional neural network, which is the only one considered in all the others' work, we explore also the deployment, on a target MCU, of other deep neural networks (DNNs) relevant to the context of HAR. Indeed, we consider the vanilla long short-term memory (LSTM) network and its combination with the one-dimensional convolutional neural network (1D CNN). In terms of findings with respect to these works, our work confirms that the deployment of a deep network on a target MCU, and related performance, are tightly coupled with the software framework that provides the compression techniques, the target hardware, the compression techniques itself, and of course the network complexity.

The goal of this work is twofold: on the one hand, we want to perform an in-depth analysis that compares the deployment of relevant DNNs on a target MCU; on the other one, we want to provide insights on: i) the feasibility of the deployment of those networks on the target MCU; ii) a comparison in terms of classification performance, inference time, and power-efficiency; iii) our experience in the adoption of Keras and TensorFlow in the design and deployment process of deep learning networks for HAR on MCU.

III. METHODOLOGY

Here we describe the methodology we proposed to investigate the efficiency of deep neural networks by applying compression techniques, and characterizing the network in terms of classification accuracy, inference latency, and energy consumption on a very resource-constrained wearable device while maintaining an appreciable level of accuracy. As referenced in section II, CNN and LSTM networks show the best results in terms of accuracy for activity recognition; therefore, we chose these networks and focused on tuning those parameters that impact the complexity, and the effectiveness of the network.

Examples of such parameters are the number of layers, the number of neurons per layer, and the connectivity among layers. By properly dimensioning these parameters, and subsequently applying compression techniques, we investigate if we are able to run deep networks on very resource-constrained devices, and how parameters impact network performances. To reach our goal, we used the Espressif ESP32-wroom-32 DevKit, on top of which we moved and tested several network complexities. Table 1 lists the network types and the network structures adopted in our investigation. The network ID is used as a shortcut to refer to the network type and its relative structure.

A. PROPOSED NETWORKS

In this work, we used two layers of a one-dimensional convolutional neural network and one layer of a one-dimensional convolutional neural network (2L_1D_CNN and 1L_1D_CNN, respectively). 1D refers to the signal dimension being processed by the input layer, compared to 2D signal processing such as images. The two layers scenario envisages two convolutional 1D layers, followed by a pooling layer and two fully connected dense layers, in which the last one provides the output. The one-layer scenario is structured in the same way, except for the fact that there is only one convolutional 1D layer. It is worth mentioning that, for both network types: i) a dropout layer is placed after the 1D convolutional layer, and a flatten layer is placed after the pooling layer; ii) the softmax function is used in the last layer to infer the activity label.

We used vanilla LSTM both in a stand-alone fashion and combined with the one-layer 1D CNN network. The stand-alone version scenario envisages one LSTM layer, followed by a dropout layer and two fully connected dense layers, in which the last one provides the output. In such a scenario the network terminates with the computation of the softmax function, as well. In the combined adoption with 1L_1D_CNN network, the structure is as follows: one convolutional 1D layer followed by a batch normalization and by a ReLU layer, which is then followed by the vanilla LSTM network.

B. PROPOSED NETWORK DIMENSIONING

The first step of our study consists of network dimension planning; the size of each network must be chosen carefully because otherwise, the network model would not fit inside the ESP32 device memory. After a brief manual tuning of the structure parameters, we end up in the network structures listed in Table 1. For each network type, we decide to start from what we expect to be the biggest size capable of fitting inside the device, and from that size, we then decrease one, or two, parameters in order to lower the network complexity.

For the two layers one-dimensional convolutional neural network (2L_1D_CNN), the biggest network size we chose is 64 filters per layer (64F) and 3 kernels (3K). The choice of 3 kernels is because we have seen it is a value typically used in literature; instead, the choice of the filter value is related to the fine-tuning phase: for values higher than 64, even by applying whatever compression technique, the resulting network model is not able to fit inside the memory of the ESP32. To gradually decrease the network complexity we decide to reduce the number of filters in both two layers while keeping the same number of kernels, planning a total of 9 network models. As network ID we chose the acronym *C2L_n* (which stands for complexity 2 layer and where n represents a sequential number).

For the one-layer one-dimensional convolutional neural network (1L_1D_CNN) we made the same reasoning about network structure parameters and network ID as per 2L_1D_CNN, and we choose the same values both for filters and kernels. The same applies to the vanilla long short-term memory network (Vanilla LSTM), this time choosing as the biggest size 256 hidden neurons (256H) and 256 dense neurons (256D). In such a scenario, we decide to gradually reduce the number of hidden neurons while leaving unchanged the number of hidden dense neurons.

For the combination of the one-layer one-dimensional convolutional neural network and the vanilla long short-term memory (1L_1D_CNN Vanilla) we planned only 4 network models in order to limit the combination of parameters that can be changed. Indeed, we decided to keep unchanged the configuration of the vanilla LSTM structure while reducing the number of filters in the 1L_1D_CNN.

C. PROPOSED WORKFLOW

Our proposed workflow consists of three phases that are represented in Figure 1. In the first phase, the raw data are extracted from a dataset and are fed into the base network model to perform train and subsequent evaluation. The 3-axial accelerometer and gyroscope data are extracted in the form of six signals from the dataset and have been divided

TABLE 1. Network types and structure.

Net Type	Net ID	Net Structure	Net Type	Net ID	Net Structure
2L_1D_CNN	C2L_9	64F 64F 3K 100D	1L_1D_CNN	C1L_9	64F 3K 100D
	C2L_8	32F 32F 3K 100D		C1L_8	32F 3K 100D
	C2L_7	20F 20F 3K 100D		C1L_7	20F 3K 100D
	C2L_6	16F 16F 3K 100D		C1L_6	16F 3K 100D
	C2L_5	10F 10F 3K 100D		C1L_5	10F 3K 100D
	C2L_4	8F 8F 3K 100D		C1L_4	8F 3K 100D
	C2L_3	4F 4F 3K 100D		C1L_3	4F 3K 100D
	C2L_2	2F 2F 3K 100D		C1L_2	2F 3K 100D
	C2L_1	1F 1F 3K 100D		C1L_1	1F 3K 100D
Vanilla	CV_11	256H 256D	1L_1D_CNN Vanilla	–	–
	CV_10	200H 256D		–	–
	CV_9	164H 256D		–	–
	CV_8	148H 256D		–	–
	CV_7	128H 256D		–	–
	CV_6	64H 256D		–	–
	CV_5	32H 256D		–	–
	CV_4	16H 256D		C1LV_4	16F 3K 256H 256D
	CV_3	8H 256D		C1LV_3	8F 3K 256H 256D
	CV_2	4H 256D		C1LV_2	4F 3K 256H 256D
	CV_1	2H 256D		C1LV_1	2F 3K 256H 256D

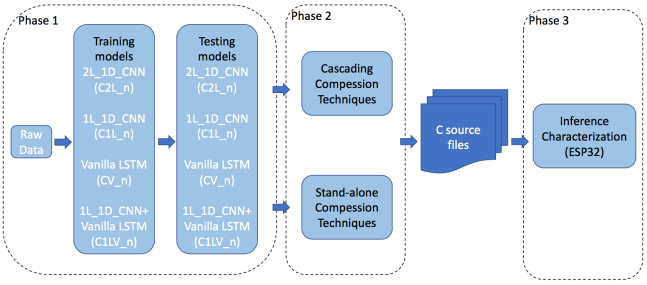


FIGURE 1. General workflow.

into time windows to generate samples for training and testing purposes. In the second phase, the base network model is subject to the compression process. We planned two different compression processes:

1) cascading compression process;
2) stand-alone compression process.

Cascading compression process consists of applying different compression techniques combined together, one after the other. In this scenario, we envisage three combinations that start from the base model:

- we apply the pruning technique and subsequently the conversion with TensorFlow Lite;
- we apply the pruning technique combined with the post-training full integer quantization;
- we apply the pruning technique combined with the post-training dynamic quantization.

Note that the pruning procedure is carried out by varying the model sparsity as shown in Figure 2(a).

Stand-alone compression process consists instead of applying each compression technique only to the base network model, thus avoiding any kind of combination. Indeed, in this scenario, we first apply directly to the base network the lite conversion, the post-training dynamic quantization, and the full integer quantization. This workflow is graphically represented in Figure 2(b).

The application of each compression technique via the TensorFlow lite converter allows to subsequently generate a C source file containing the char array version of the lite network model. To generate such a file, which allows running the model on a microcontroller, it is necessary to use the Unix command *xxd*[1]. In the third phase, each C source file is moved onto the ESP32 to test if the model fit on the platform and, in case it does, to test its performance in term of inference latency. It is worth mentioning that the proposed workflow has been applied to each network listed in Table1.

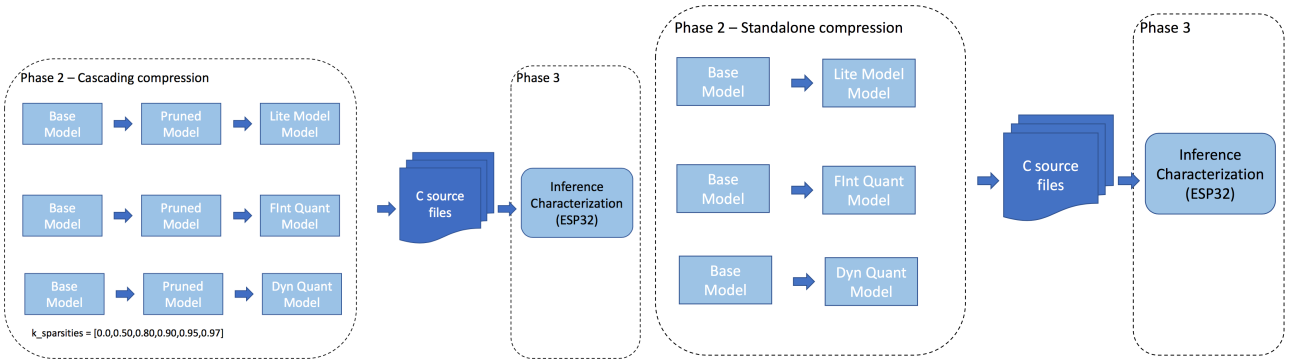
IV. EXPERIMENTAL SETUP

We developed a Keras-TensorFlow application in a Google Colab environment. We used TensorFlow nightly developer version 2.10 to build, train and evaluate network models. To apply compression techniques we used TensorFlowLite which allows performing several compression techniques, including those we choose for our investigation, i.e.: conversion to lite model, post-training dynamic quantization, and full integer 8-bit quantization.

A. HAR DATASET

To carry our investigation out, in this work we leveraged the UCI-HAR dataset [34]. Here, information is collected via a smartphone, where accelerometer and gyroscope signals are sampled at 50 Hz. The monitored activities are: *walking, walking upstairs, walking downstairs, sitting, standing, and lying down*. All the activities are gathered from 30 subjects aged between 18 and 48 years, and of varying races, heights, and weights.

[1] http://www.tensorflow.org/lite/microcontrollers/build_convert?hl=en



(a) Cascading compression process

(b) Stand-alone compression process

FIGURE 2. Compression techniques processes

The raw dataset is split into 75% for training and 25% for testing. The training set has been used to train each network model listed in Table 1. Then, the testing set has been used to assess the performance of each network model.

B. REAL APPLICATION SCENARIO

We developed a sensor-based HAR application on top of an Espressif ESP32-wroom-32 DevKit connected to an MPU6050 integrated 6-axis motion tracking device that combines a 3-axis gyroscope, and 3-axis accelerometer [35]. The application was entirely developed in C++ using the ESP-IDF platform (Espressif IoT Development Framework) with the TensorFlow Lite Micro libraries installed.

The ESP32 has two CPU cores that can be individually controlled running at a clock frequency adjustable from 80 MHz to 240 MHz. The chip also has a low-power mode that can be used to save power while performing peripheral I/O tasks that do not require much computing effort. Our version of the ESP32 has 4 MB of flash memory for saving the firmware and 400 KB of RAM memory.

Because a TensorFlow model is represented through a static set of weights and instructions, it can be stored directly inside the flash memory together with the firmware components so that the maximum size of the installable model is slightly less than the 4 MB.

The HAR application designed for this work consists of two tasks that respectively perform the data collection from the motion tracking device (MPU6050) and the ML inference. The two tasks behave according to the producer-consumer paradigm and are executed independently on one of the two available cores.

C. MEASUREMENT SETUP

To estimate the energy consumption of the ESP32 device, we measured the voltage drop across a sensing resistor (9.8Ω) placed in series with the device's power supply. The device was powered at 3.3V through an NGMO2 Rohde & Schwarz dual-channel power supply [36], and we sampled the signals to be monitored during the experiments by means of a National Instruments NI-DAQmx PCI-6251 16-channel data acquisition board connected to a BNC-2120 shielded connector block [37, 38].

V. EXPERIMENTAL RESULTS

In this section, we report the results obtained by applying compression techniques in cascading and in stand-alone modes. We first investigated the cascading mode by following the workflow previously described and shown in Figure 2(a), and then we investigated the stand-alone mode shown in Figure 2(b).

A. CASCADING COMPRESSION TECHNIQUES CASE

Despite in principle compression techniques might be combined together to enhance performance, we experienced issues both at design and execution time. Note that, when we talk about design time we mean "when writing and running the Keras-TensorFlow application on the Colab environment"; whereas when we talk about execution time, we mean "when deploying the trained network model on the ESP32 device".

At design time, when combining pruning with all the other quantization techniques, we experienced errors raised by the TensorFlow Lite *interpreter* runtime environment. We were able to solve those errors by enabling specific TensorFlow Lite and TensorFlow operators (OPS). This step is necessary because the built-in operators supported by the TensorFlow Lite library are limited to a subset of numbers of the TensorFlow operators. Therefore, the conversion is successful only if TensorFlow OPS are enabled in the TensorFlow Lite model.

However, the C source file obtained after the conversion raised an error when the network model is deployed on the ESP32 device. For this reason, we were not able to perform phase 3, i.e., the inference characterization on the ESP32. At the time of such experimentation, we also engaged the TensorFlow Lite developers community to solve this issue, but without finding a proper solution. We speculate that the issue might be caused by several sources, among which: i)

the adoption of the pruning technique: we experienced that, by avoiding the pruning procedure in the process, and by avoiding enabling TensorFlow OPS, this acts as a sort of "workaround" to make the compression process successful both at design and execution time; ii) either at TensorFlow Lite side or at ESP-IDF platform side, there is something which is still unsupported, thus leading to the impossibility of testing the combined adoption of compression techniques.

Results obtained for such a scenario contributed to motivating us to investigate the stand-alone scenario, whose results are discussed in the next sections.

B. STAND-ALONE COMPRESSION TECHNIQUES CASE

In such a scenario, we were able to carry out a thorough characterization by providing a comparison in terms of classification performance, power efficiency, and inference time. The ESP32 is able to work in two different modes: power management enabled (PME), i.e., the CPU is able to go down to save power, and power management disabled (PMD). We performed our analysis leveraging both modes to evaluate the power efficiency and the inference time of the HAR application on the ESP32.

Figures shown in the next sections not only consider each network type but also each compression technique, as well. Compression techniques are represented by colors: green for Lite conversion, cyan for Dynamic quantization, and red for Full integer quantization.

1) Accuracy

The base network model (i.e., a network model not subject to any compression technique) achieves a maximum accuracy of around:

- 89% for the 2L_1D_CNN network;
- 88% for the 1L_1D_CNN network;
- 92% for the vanilla LSTM network;
- 92% for the combined adoption of 1L_1D_CNN and vanilla LSTM network.

Figure 3 shows the highest classification accuracy of each network type, calculated over the UCI-HAR dataset, considering the application of each compression technique, besides the maximum accuracy reached by the corresponding base network model. It is interesting to note that by simply lite converting the 1L_1D_CNN+Vanilla we never obtain a fitting network model. It is also worth mentioning that, to evaluate the maximum accuracy, we did not consider the accuracy reached by the networks that, even compressed, are not able to fit inside the ESP32 device because of their memory occupation size. Table 2 reports with an ✗all the network models (with respect to the compression technique: L-Lite, D-Dynamic, F-Full integer) not fitting inside the ESP32.

Results in Figure 3 show that by applying the lite conversion the accuracy drop is: 2% for the 2L_1D_CNN network, 1% for the 1L_1D_CNN network, and 4% for the vanilla LSTM network. By applying dynamic quantization there is no accuracy drop, whereas by applying full integer quantization there is always an accuracy drop of 2%.

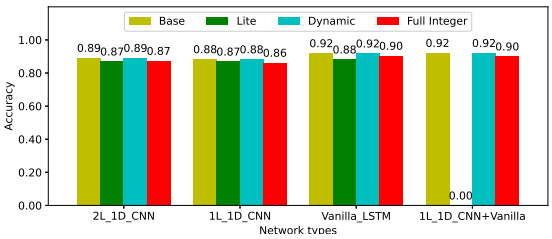


FIGURE 3. Network classification accuracy.

TABLE 2. Network models fitting on the ESP32.

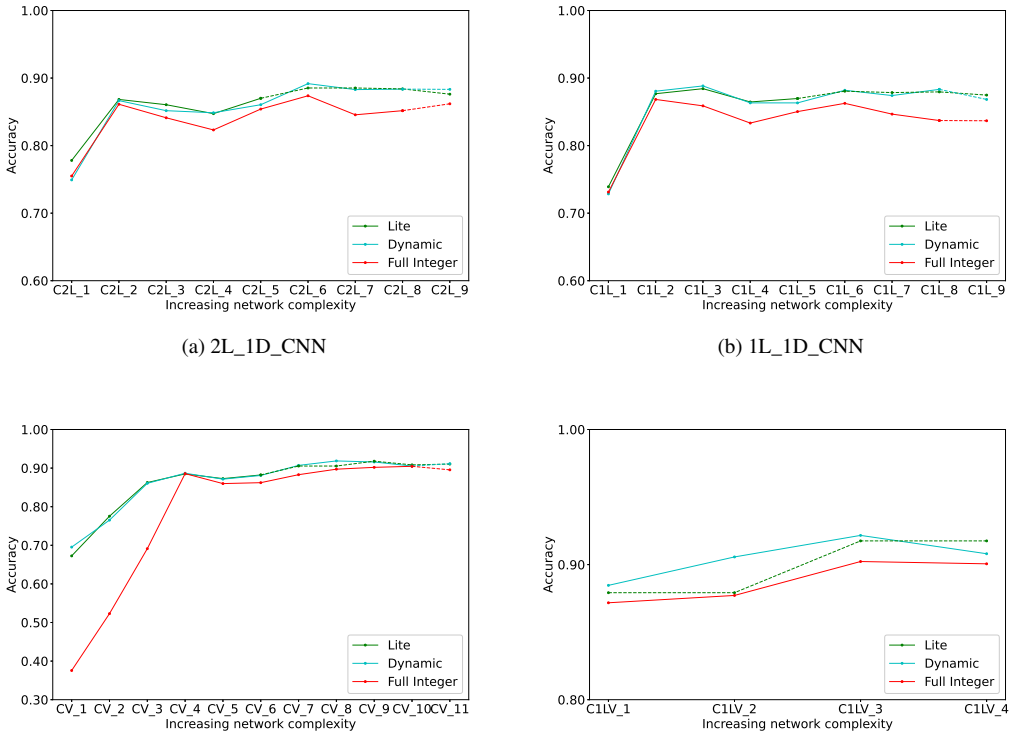
Net Type	Net ID	L	D	F	Net Type	Net ID	L	D	F
2L_1D_CNN	C2L_9	✗	✗	✗	1L_1D_CNN	C1L_9	✗	✗	✗
	C2L_8	✗	✓	✓		C1L_8	✗	✓	✓
	C2L_7	✗	✓	✓		C1L_7	✗	✓	✓
	C2L_6	✗	✓	✓		C1L_6	✗	✓	✓
	C2L_5	✓	✓	✓		C1L_5	✓	✓	✓
	C2L_4	✓	✓	✓		C1L_4	✓	✓	✓
	C2L_3	✓	✓	✓		C1L_3	✓	✓	✓
	C2L_2	✓	✓	✓		C1L_2	✓	✓	✓
	C2L_1	✓	✓	✓		C1L_1	✓	✓	✓
Vanilla	CV_11	✗	✗	✗	1L_1D_CNN Vanilla	–	–	–	–
	CV_10	✗	✓	✓		–	–	–	–
	CV_9	✗	✓	✓		–	–	–	–
	CV_8	✗	✓	✓		–	–	–	–
	CV_7	✗	✓	✓		–	–	–	–
	CV_6	✓	✓	✓		–	–	–	–
	CV_5	✓	✓	✓		–	–	–	–
	CV_4	✓	✓	✓		C1LV_4	✗	✓	✓
	CV_3	✓	✓	✓		C1LV_3	✗	✓	✓
	CV_2	✓	✓	✓		C1LV_2	✗	✓	✓
	CV_1	✓	✓	✓		C1LV_1	✗	✓	✓

Figure 4 shows the accuracy provided by each compression technique applied to each network type when the network complexity is increased. Each dashed colored line represents the accuracy reached by those network models that are not able to fit inside the ESP32. For the sake of readability, the Figure does not show the curve related to the accuracy of the base network model, i.e., the model not subject to any of the three compression techniques. The accuracy of the base network model of each network type ranges between:

- 77% and 89% for the 2L_1D_CNN network;
- 73% and 88% for the 1L_1D_CNN network;
- 67% and 92% for the vanilla LSTM network;
- 87% and 92% for the combined adoption of 1L_1D_CNN and vanilla LSTM network.

The minimum accuracy reached by the base network model is the one provided by the less complex network model of each network type.

Very simple base networks can indeed provide a low level of accuracy (e.g., 67% for the most simple vanilla LSTM base network, CV_1). Therefore, the application of compression techniques may result in a higher loss of accuracy in less complex networks compared to more complex ones. This is for example the case of CV_1 vanilla LSTM network when subject to the full integer quantization, which reaches a level of accuracy of around 37% as shown in Figure 4(c). For



(a) 2L_1D_CNN

(b) 1L_1D_CNN

(c) Vanilla

(d) 1L_1D_CNN Vanilla

FIGURE 4. Classification accuracy of each network type per increasing network complexity, obtained by applying lite conversion, dynamic quantization, and full integer quantization.

vanilla LSTM networks, whose network complexity is higher (or equal) to CV_4, the three techniques provide comparable performance. For all the other network types, the full integer quantization incurs in a higher loss of accuracy compared to other compression techniques. However, this is a known effect since the full integer quantization converts the math model weights and input from a float format into an int format.

2) Power-efficiency

The procedure carried out to evaluate the power efficiency, described in the following, was evaluated both for the PMD and the PME case.

The sampling and the inference power were measured during the execution of the HAR application on the ESP32, and the generated waveform is depicted in Figure 5. The plot reported on the top shows the power consumption of the HAR application in the PMD case. The power peaks corresponding to the single readings of the sensors, carried out by the core #0 of the CPU, and the large increase in power corresponding to the inference phase, are clearly recognizable. The plot on the bottom, on the other hand, reports the power consumption in the PME case. It is worth noting the energy saving during the sampling period in this case due to core #0 shutting down between receiving two consecutive samples. Starting from this characterization, we are therefore able to evaluate the total energy consumed by each network compressed model considering the whole cycle, i.e., the energy spent during the sampling phase summed to the energy spent during the inference phase.

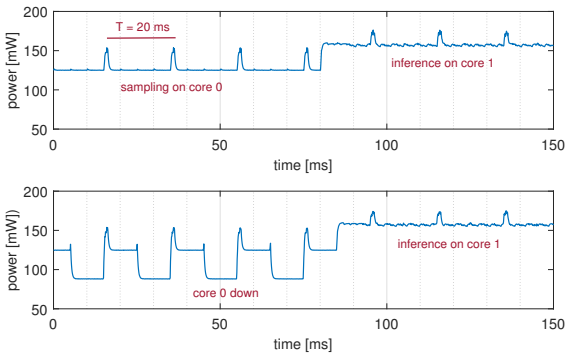


FIGURE 5. Power consumption waveforms collected during the execution of the HAR application on the ESP32 device (top) and on the ESP32 with power management enabled (bottom).

For each network type, all the respective fitting network models reported in Table 2 were deployed one by one on the ESP32 device. Given a fitting network model, the HAR application was run to sample real-time data directly from the gyroscope and the accelerometer and to perform real-

time inference measurements. We collected 10 inference measurements for each network compressed model in order to get an average of the inference time. Table 3 reports the power consumption measured during sampling and inference phases in PMD and PME cases respectively. A window size of 2 seconds is a reasonable choice given the characteristics of the UCI-HAR dataset in terms of activities to be classified and sampling ratio.

TABLE 3. Characterization of the power consumption of the device with power management disabled (PMD) and power management enabled (PME) during sampling and inference phases.

	PMD	PME
Sampling Power [Watt]	0,1267	0,1055
Inference Power [Watt]	0,1581	0,1582

Figure 6 shows the results in terms of energy consumption of each network type in the PMD. In particular, the figures show the Pareto curves of the accuracy loss plotted versus the energy consumption, where points represent the Pareto optimal network type architectures. It is worth mentioning that similar considerations can be done for the PME case.

Considering the 2L_1D_CNN network in Figure 6(a), the best trade-off is provided by the C2L_2 network configuration: applying one of the three compression techniques yields an energy consumption of around 253.6mJ, with an accuracy of 86%. However, the real distinction is the inference time, which can be decreased to a minimum of around 5ms for the C2L_2 network, compared to the maximum of around 61ms for the C2L_7 network. In such a scenario is therefore advantageous the adoption of a low complex 2L_1D_CNN network, i.e., the C2L_2 network configuration.

In the 1L_1D_CNN network case in Figure 6(b), the application of the three compression techniques to two of the lowest complex 1L_1D_CNN network (i.e., C1_L3 and C1_L2) provides almost the same high level of accuracy, around 86-88%, and the same energy consumption, around 253mJ. In terms of inference time, the range goes from a maximum of around 12ms given by the lite conversion of the C1_L3 network, to a minimum of around 5ms given by the full integer compression of the C1_L2 network. In such a scenario is therefore advantageous the adoption of a low complex 1L_1D_CNN network, i.e., the C1_L2 network.

In the vanilla LSTM network case, Figure 6(c), the application of either dynamic quantization or lite conversion to the CV_4 network provides the same accuracy, around 88%, and the same energy consumption, around 255mJ. In this case, it is advantageous the adoption of the dynamic conversion because results in a lower inference time, around 60ms compared to around 72ms provided by the lite conversion. It is worth noting that, to reach an accuracy of around 91%, it is necessary to pick a more complex vanilla LSTM network (i.e., CV_8), resulting however in a high energy consumption, around 324mJ, and a very high inference latency, over 2s, which might be unacceptable in certain applications.

In the combined adoption of 1L_1D_CNN and vanilla LSTM network case, Figure 6(d), as mentioned in the previous section, the application of the lite conversion does not produce any fitting model suitable for the ESP32 capacity. The application of quantization to C1LV_1, C1LV_2 and C2LV_3 networks provides the same energy consumption, around 255mJ, and an accuracy ranging from an 88% of the C1LV_1 network to a 92% of the C1LV_3 network. In this case, in terms of inference latency, the difference ranges from around 56ms of the C1LV_1 network to around 66ms of the C1LV_3 network. In this case, it is advantageous the adoption of dynamic quantization applied to C1LV_1 network.

VI. DISCUSSION

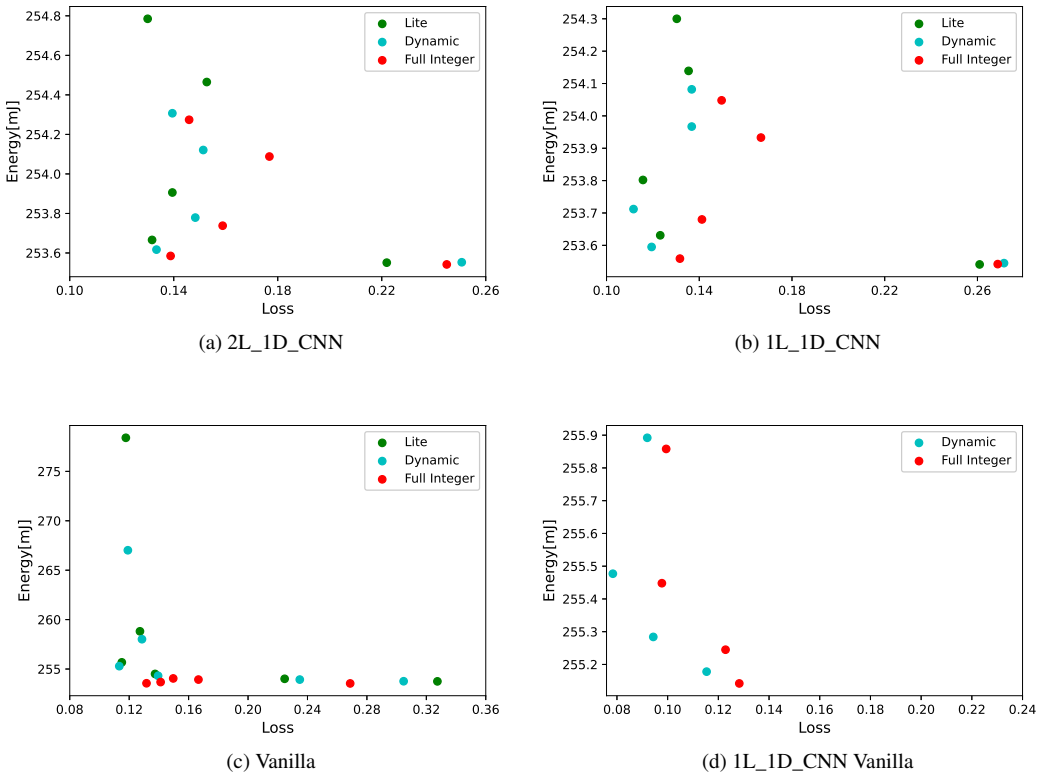
The results described in the previous section allow us to provide insights into our experience with the Keras-TensorFlow framework and the deployment onto ESP32. Here the discussion is limited to the stand-alone compression techniques scenario because we were able to perform a complete characterization.

We further analyzed the power efficiency of dynamic and full integer quantization by estimating their energy saving with respect to the lite conversion. Here, we only consider the inference energy saving because the sampling energy is independent of the network model. Figure 7 shows the inference energy saving, in the PMD case, induced by the two quantization techniques with respect to the base value of the lite model. Notice that, in the case of the CNN and LSTM combined network no configuration of lite models was able to run on the ESP32 device; therefore, we were not able to estimate the energy saving in such a case. For each network type, the estimation was evaluated for all the network configurations for which we successfully obtained three executable models.

In the case of models 2L_1D_CNN and 1L_1D_CNN, Figure 7(a) and (b), respectively, it is interesting to note that: for the simplest model (C2L_1 and C1L_1) both techniques do not affect the energy consumption which is always very low. Starting from C2L_2 to C2L_5, and from C1L_2 to C1L_5 configurations, the energy saved by the full integer quantization always overcomes that obtained with the dynamic technique. However, it seems that as the complexity of the models increases, the differences between the two techniques decrease, both reaching around an energy-saving close to 40% and 30%, respectively.

In the case of vanilla models, Figure 7(c), the trend is similar to the other two network types, except for the CV_6 configuration, which shows a significantly much higher saving, compared to the lower network complexity configurations. This is due to the high difference in the inference energy consumption between CV_6 and CV_5.

Table 4 shows the total energy consumption, in both PMD and PME cases, which account for sampling energy and inference energy. It is worth noting that, in the PME case, given the same network type and network configurations, the total energy consumed is way lower compared to the



(a) 2L_1D_CNN

(b) 1L_1D_CNN

(c) Vanilla

(d) 1L_1D_CNN Vanilla

FIGURE 6. Pareto curve reporting the accuracy loss Vs the expected energy consumption in case power management is disabled (PMD).

PMD case. This is because, in the PME case, during the sampling phase the ESP32 shuts core #0 of the CPU down, thus allowing a higher energy saving, compared to the PMD case.

Considering that, from the classification accuracy point of view, the dynamic quantization technique does not involve a priceable loss, we would like to advise in any case the use of this technique even if the energy savings produced by the full integer quantization could be slightly greater. Furthermore, we remind you that the application of one of the two quantization techniques is even indispensable for models of considerable size which otherwise would not execute on some extremely low-power platforms such as the ESP32.

The four network types adopted in our investigation show interesting performance either in terms of accuracy, power efficiency, and inference time. Therefore, by properly dimensioning the base network model type, each network type can become a good candidate to be deployed on the ESP32. As far as the combined 1D_1L_CNN+vanilla is concerned, by deciding to keep the number of hidden and dense neurons unchanged, we only got 4 base networks, compared to the other 9/11 base models. Despite not being reported in Table 1, for the 1D_1L_CNN+vanilla network we tried also to explore the case of decreasing hidden neurons while leaving unchanged the number of filters, kernels, and (high) dense neurons. In particular, we considered two additional networks with the following network structure: 16F 3K 8H 256D, let's name it C1LV_E3, and 16F 3K 4H 256D, let's name it C1LV_E2.

Considering the PMD case, the outcome was the following: in terms of the accuracy of the base models, we got the same range highlighted in section V-B2; in terms of power efficiency, we observed a slight reduction of the energy consumption, compared to the other 4 networks formerly investigated. However, we observed a significant inference time reduction compared to C1LV_3 and C1LV_2, which we consider "comparable networks". In particular, considering the dynamic quantization technique, between C1LV_3 and C1LV_E3 we observed an inference time reduction of a 34%; whereas between C1LV_2 and C1LV_E2 we observed a reduction of around 49%. Similar values of inference time reduction were observed for the full integer quantization technique, as well. Analog consideration can be done for the PME case. We speculate that, by decreasing also the number of dense layers, there would be an additional gain in terms of inference latency.

VII. CONCLUSION AND FUTURE WORK

This paper investigates the feasibility, and the impact, of deep learning model deployment on a low-power ESP32 device, applied to a HAR case study. We considered four network types, namely, a two layers convolutional neural

TABLE 4. Total energy consumption for fitting network models on the ESP32.

		Energy[mJ]					
		PMD			PME		
Net Type	Net ID	L	D	F	L	D	F
2L_1D_CNN	C2L_5	254.785	254.307	254.274	213.321	212.521	212.469
	C2L_4	254.465	254.121	254.088	212.783	212.213	212.167
	C2L_3	253.906	253.779	253.738	211.842	211.641	211.582
	C2L_2	253.666	253.617	253.585	211.455	211.374	211.320
	C2L_1	253.551	253.553	253.542	211.255	211.265	211.236
1L_1D_CNN	C1L_5	254.300	254.082	254.048	212.518	212.145	212.098
	C1L_4	254.139	253.967	253.933	212.242	211.948	211.890
	C1L_3	253.802	253.712	253.680	211.680	211.524	211.469
	C1L_2	253.631	253.595	253.559	211.393	211.328	211.273
	C1L_1	253.541	243.545	243.542	211.237	211.248	211.236
Vanilla	CV_6	278.398	267.018	267.042	253.025	233.724	233.623
	CV_5	258.816	258.017	257.982	220.119	218.738	218.698
	CV_4	255.676	255.290	255.254	214.825	214.158	214.101
	CV_3	254.503	254.313	254.280	212.859	212.534	212.481
	CV_2	254.010	253.932	253.899	212.020	211.890	211.839
	CV_1	253.751	253.763	253.731	211.604	211.617	211.561

network, a one layer convolution neural network, a vanilla long short-term memory network, and a combination of those two latter. Those networks were fine-tuned to find potential fitting models. We carried out a thorough analysis of the feasibility, and the impact, of the application of the most well-known TensorFlow compression techniques, i.e., pruning, lite conversion, dynamic quantization, and full integer quantization. Specifically, we considered the application of those techniques in a cascading mode and in a stand-alone mode. The impact was evaluated in terms of accuracy and energy consumption, with consequences also on the inference latency. Two working modes were employed for the ESP32: power management disabled (PMD), and power management enabled (PME).

In the cascading case it is not possible to carry out the performance characterization because of experienced issues both at design and at execution time. We, therefore, recommend avoiding concatenating the pruning technique with the other ones. In the stand-alone case, dynamic quantization is recommended because yields a negligible loss of accuracy. In terms of power efficiency, both dynamic and full integer quantization provide high energy saving with respect to the uncompressed models: between 31% and 37% for CNN networks, and up to 45% for LSTM networks. In terms of inference latency, dynamic and full integer quantization provide comparable performance. As a final remark, we would like to recommend the adoption of dynamic quantization because of its negligible accuracy drop and its comparable performance with the full integer technique in terms of power efficiency and inference latency.

Our work shows that multiple network types can be deployed onto an ESP32 device, each showing interesting performance parameters. We hope that our experience with Keras/TensorFlow on the ESP32 will motivate the reader to further explore such a broad landscape in the design and deployment process of deep learning networks on MCU for HAR. In future work, we would like to solve the issue we faced with the pruning technique to be able to complete our investigation. We would also like to carry out such a thorough investigation on other target hardware embedded platforms to evaluate and compare different outcomes in terms of inference latency and power efficiency. Another research opportunity could be to explore other model compression techniques, such as for example knowledge distillation.

REFERENCES

[1] G. V. Research, "Wearable technology mrket size," 2023, last accessed 2023-03-16. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/wearable-technology-market

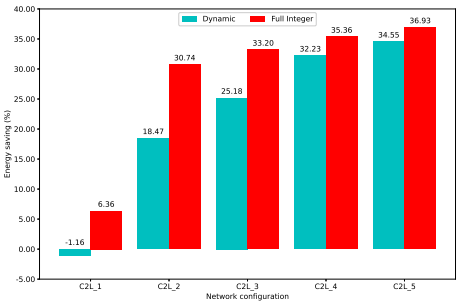
[2] F. Foerster, M. Smeja, and J. Fahrenberg, "Detection of posture and motion by accelerometry: a validation study in ambulatory monitoring," *Computers in human behavior*, vol. 15, no. 5, pp. 571–583, 1999.

[3] M. H. Arshad, M. Bilal, and A. Gani, "Human activity recognition: Review, taxonomy and open challenges," *Sensors*, vol. 22, no. 17, p. 6463, 2022.

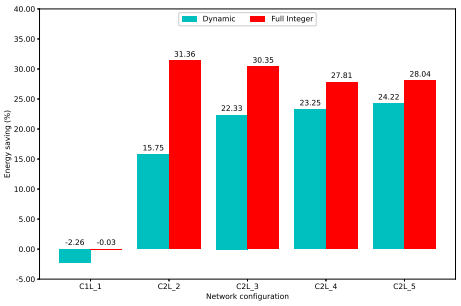
[4] L. Deng and X. Li, "Machine learning paradigms for speech recognition: An overview," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 5, pp. 1060–1089, 2013.

[5] F. Navabifar, M. Emadi, R. Yusof, and M. Khalid, "A short review paper on face detection using machine learning," in *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*. Citeseer, 2011, p. 1.

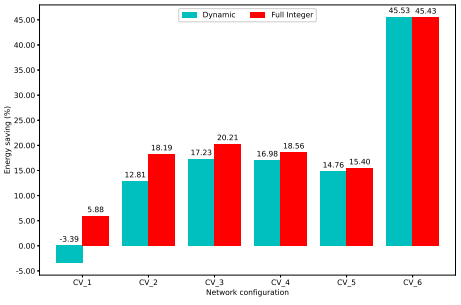
[6] L. Cheng, Y. Guan, K. Zhu, and Y. Li, "Recognition of human activities using machine learning methods with wearable sensors," in *2017 IEEE 7th annual computing and communication workshop and conference*



(a) 2L_1D_CNN



(b) 1L_1D_CNN



(c) Vanilla

FIGURE 7. Inference energy saving obtained by applying dynamic and full integer quantization to all network types when varying network configuration complexity.

(CCWC). IEEE, 2017, pp. 1–7.

[7] L. Martin Wisniewski, J.-M. Bec, G. Boguszewski, and A. Gamatié, "Hardware solutions for low-power smart edge computing," *Journal of Low Power Electronics and Applications*, vol. 12, no. 4, p. 61, 2022.

[8] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, "Efficient acceleration of deep learning inference on resource-constrained edge devices: A review," *Proceedings of the IEEE*, 2022.

[9] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[10] S. Qin, J. Zhang, H. Shen, and Y. Wang, "Arm movements recognition by implementing cnn on microcontrollers," in *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)*. IEEE, 2021, pp. 171–176.

[11] E. Manor and S. Greenberg, "Custom hardware inference accelerator for tensorflow lite for microcontrollers," *IEEE Access*, vol. 10, pp. 73 484–73 493, 2022.

[12] Espressif, "Esp32-c3-wroom-02 datasheet," 2022, last accessed 2023-02-07. [Online]. Available: https://www.espressif.com/en/support/documents/technical-documents

[13] S. Ramasamy Ramamurthy and N. Roy, "Recent trends in machine learning for human activity recognitionâĂŤa survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1254, 2018.

[14] R. Mutegeki and D. S. Han, "A cnn-lstm approach to human activity recognition," in *2020 international conference on artificial intelligence in information and communication (ICAIIC)*. IEEE, 2020, pp. 362–366.

[15] S. Mekruksavanich, P. Jantawong, N. Hnoohom, and A. Jitpattanakul, "Deep learning models for daily living activity recognition based on wearable inertial sensors," in *2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, 2022, pp. 1–5.

[16] S. Mekruksavanich and A. Jitpattanakul, "Smartwatch-based human activity recognition using hybrid lstm network," in *2020 IEEE SENSORS*. IEEE, 2020, pp. 1–4.

[17] M. Ronald, A. Poulose, and D. S. Han, "isplinception: An inception-resnet deep learning architecture for human activity recognition," *IEEE Access*, vol. 9, pp. 68 985–69 001, 2021.

[18] P. Kim and P. Kim, "Convolutional neural network," *MATLAB deep learning: with machine learning, neural networks and artificial intelligence*, pp. 121–147, 2017.

[19] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, 2020.

[20] W. Qi, H. Su, and A. Aliverti, "A smartphone-based adaptive recognition and real-time monitoring system for human activities," *IEEE Transactions on Human-Machine Systems*, vol. 50, no. 5, pp. 414–423, 2020.

[21] V. Bianchi, M. Bassoli, G. Lombardo, P. Fornacciari, M. Mordonini, and I. De Munari, "Iot wearable sensor and deep learning: An integrated approach for personalized human activity recognition in a smart home environment," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8553–8562, 2019.

[22] T. Zhao, Y. Xie, Y. Wang, J. Cheng, X. Guo, B. Hu, and Y. Chen, "A survey of deep learning on mobile devices: Applications, optimizations, challenges, and research opportunities," *Proceedings of the IEEE*, vol. 110, no. 3, pp. 334–354, 2022.

[23] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.
[24] "TensorFlow official website," http://www.tensorflow.org/?hl=en, accessed: 2023-03-08.
[25] B. Pang, E. Nijkamp, and Y. N. Wu, "Deep learning with tensorflow: A review," *Journal of Educational and Behavioral Statistics*, vol. 45, no. 2, pp. 227–248, 2020.
[26] "TensorFlow Lite for microcontrollers," http://www.tensorflow.org/lite/microcontrollers?hl=en, accessed: 2023-03-08.
[27] "Keras official website," http://keras.io/, accessed: 2023-03-08.
[28] P.-E. Novac, A. Castagnetti, A. Russo, B. Miramond, A. Pegatoquet, and F. Verdier, "Toward unsupervised human activity recognition on microcontroller units," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2020, pp. 542–550.
[29] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and deployment of deep neural networks on microcontrollers," *Sensors*, vol. 21, no. 9, p. 2984, 2021.
[30] F. Daghero, C. Xie, D. J. Pagliari, A. Burrello, M. Castellano, L. Gandolfi, A. Calimera, E. Macii, and M. Poncino, "Ultra-compact binary neural networks for human activity recognition on risc-v processors," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, 2021, pp. 3–11.
[31] F. Daghero, A. Burrello, C. Xie, M. Castellano, L. Gandolfi, A. Calimera, E. Macii, M. Poncino, and D. J. Pagliari, "Human activity recognition on microcontrollers with quantized and adaptive deep neural networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 4, pp. 1–28, 2022.
[32] F. Daghero, D. J. Pagliari, and M. Poncino, "Two-stage human activity recognition on microcontrollers with decision trees and cnns," in *2022 17th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*. IEEE, 2022, pp. 173–176.
[33] A. Ghibellini, L. Bononi, and M. Di Felice, "Intelligence at the iot edge: activity recognition with low-power microcontrollers and convolutional neural networks," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2022, pp. 707–710.
[34] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra, and D. Anguita, "Transition-aware human activity recognition using smartphones," *Neurocomputing*, vol. 171, pp. 754–767, 2016.
[35] InvenSense Inc., "Mpu-6050 product specification," 2023, last accessed 2023-02-07. [Online]. Available: https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/
[36] Rohde&Schwarz, "Ngmo2 datasheet," 2020, last accessed 2023-02-07. [Online]. Available: https://www.rohde-schwarz.com/it/brochure-scheda-tecnica/ngmo2/
[37] National.Instruments, "Pc-6251 datasheet," 2020, last accessed 2023-02-07. [Online]. Available: http://www.ni.com/pdf/manuals/375213c.pdf
[38] ——, "Installation guide bnc-2120," 2020, last accessed 2023-02-07. [Online]. Available: http://www.ni.com/pdf/manuals/372123d.pdf

CHIARA CONTOLI graduated in Computer Engineering from the University of Bologna, Italy in 2013 and graduated with her Ph.D. from the University of Bologna in Electronics, Telecommunications and Information Technologies Engineering in 2017. For two years, she worked as a software developer in the industry. Since 2022 she is a fixed-term junior assistant Professor in Computer Engineering at the Department of Pure and Applied Sciences (DiSPeA) of the University of Urbino, Italy. Her research interests are in network management and network softwarization, network architectures and protocols, the Internet of Things, machine learning, and power management. She is also interested in programming languages for networks and enjoys working on interdisciplinary problems.

EMANUELE LATTANZI received the Laurea degree in 2001 and the Ph.D. degree from the University of Urbino, Italy, in 2005. Since 2001, he has been with the Information Science and Technology Institute, University of Urbino. In 2003, he was with the Department of Computer Science and Engineering, Pennsylvania State University, as a Visiting Scholar with Prof. V. Narayanan. From 2008 until 2020, he has been Assistant Professor in Computer Engineering at the Department of Pure and Applied Sciences (DiSPeA) of the University of Urbino, Italy, where he is currently an Associate Professor in Computer Engineering. His research interests include wireless sensor networks, energy-aware routing algorithms, machine learning, and Internet of Things.