IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 10, 2019, accepted October 1, 2019, date of publication October 21, 2019, date of current version October 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2948577

Optimality Assessment of Memory-Bounded ConvNets Deployed on Resource-Constrained RISC Cores

MATTEO GRIMALDI, (Student Member, IEEE), VALENTINO PELUSO, (Student Member, IEEE), AND ANDREA CALIMERA, (Member, IEEE)
Politecnico di Torino, 10129 Turin, Italy
Corresponding author: Andrea Calimera (andrea.calimera@polito.it)

ABSTRACT A cost-effective implementation of Convolutional Neural Nets on the mobile edge of the Internet-of-Things (IoT) requires smart optimizations to fit large models into memory-constrained cores. Reduction methods that use a joint combination of *filter pruning* and *weight quantization* have proven efficient in searching the compression that ensures minimum model size without accuracy loss. However, there exist other optimal configurations that stem from the memory constraint. The objective of this work is to make an assessment of such memory-bounded implementations and to show that most of them are centred on specific parameter settings that are found difficult to be implemented on a low-power RISC. Hence, the focus is on quantifying the distance to optimality of the closest implementations that instead can be actually deployed on hardware. The analysis is powered by a two-stage framework that efficiently explores the memory-accuracy space using a lightweight, hardware-conscious heuristic optimization. Results are collected from three realistic IoT tasks (Image Classification on CIFAR-10, Keyword Spotting on the Speech Commands Dataset, Facial Expression Recognition on Fer2013) run on RISC cores (Cortex-M by ARM) with few hundreds KB of on-chip RAM.

INDEX TERMS Neural networks, Internet of Things, optimization methods, low power electronics.

I. INTRODUCTION AND MOTIVATIONS

Most IoT applications run Deep Convolutional Neural Networks (ConvNets hereafter) in the cloud, public or private depending on the context. However, there is wide consensus that the growth of a sustainable IoT ecosystem encompasses the deployment of ConvNets on the mobile edge [1]. The migration from cloud services is challenging and it first requires a proper understanding of the hardware characteristics of the hosting end-nodes. The focus of this work is on low-cost IoT applications (e.g. that described in [2]) where form-factor and energy budget are the main concern. In such cases, the software stack is developed over off-the-shelf embedded platforms powered by tiny RISC cores. With no lack of generality, we take as case study the microcontroller units (MCUs) of the Cortex-M family by ARM[1] shown in Fig. 1. Low-power MCUs have few KB of on-chip RAM (from 4 to 32 kB for the M0, from 256 to 512 kB for the M7 – depending on the chip-set). Off-chip memory supports are usually not integrated and when available they affect several metrics negatively, like latency, energy, integration cost, endurance and reliability. The instruction set architecture (ISA) comes with few integer options (16- and 8-bit) and often no floating-point. Lastly, they lack parallelism to accelerate vector operations. A small 2-lane Single Instruction Multiple Data (SIMD) unit is integrated into the M4 and M7 only.

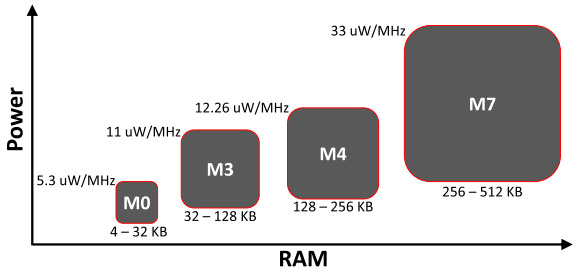


FIGURE 1. Cortex-M family: Active power and on-chip RAM size.

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

[1]https://os.mbed.com/platforms/

With these hardware figures there is very little room to deploy ConvNets of some practical use. In fact, even the simplest network is made of multidimensional arrays (tensors) of a size such as to prevent full on-chip storage and real-time processing. A practical way to address the resource bottleneck is to play at the algorithmic level using compression methods that reduce the cardinality of the inner tensors. Among the available solutions [3], *filter pruning* [4] and *weight quantization* [5] have become standard. Both of them are commonly applied at post-training with the aim of removing redundant information, namely, those parts of the ConvNet model which do not contribute, or that weakly contribute, to the overall accuracy. Pruning implements a selective dropping of weak parameters; quantization applies a bit-width scaling on the arithmetic representation of the parameters. The latest advances show that a joint combination of the two achieves state-of-the-art [6]. The returned ConvNets show fewer weights to store, hence fewer operations to run. These kinds of methods are usually accuracy-driven, namely they seek for the optimal model setting that guarantees the highest memory compression with the lowest accuracy loss, ideally zero. Model settings has a relative meaning here: it is the largest selection of weak parameters for pruning, the smallest bit-width of the arithmetic representation for quantization, or the optimal pruning-to-quantization ratio for joint methods.

The search for optimality gets challenging due to several reasons. Just like for training, the lack of a closed-form solution to describe the dynamics of the learning flow makes the optimization loop slow and uncertain. Furthermore, the constraints which rise from the hardware layer introduce an additional level of complexity overlooked by the most of the existing works. A problem formulation that does not consider these aspects might result too weak, or unsuited, for real life applications deployed on tiny RISC cores. Intuitively enough, accuracy cannot be the only dimension to explore. An optimization loop unconstrained in terms of memory footprint may return models that still do not fit the target core. Moreover, since quantization and pruning are lossy methods, they get controlled by means of a user-defined accuracy constraint. Since the cost function is unknown, the selection of such accuracy constraint is blind. There might be solutions close enough in accuracy, namely equivalent in terms of quality, but very far in memory occupation. Also, it is hard to prioritise pruning and quantization as they both affect the same source of information and finding the right balance is still an open issue. This suggests design space exploration is more reliable than multi-objective optimization. Obviously, the large number of hyper-parameters makes exhaustive approaches impractical and it calls for for smart heuristics instead. Not least, quantization below the 8-bit mark (e.g. from 7- to 2-bit [6]) remains a theoretic study as it asks for custom hardware components which are not available in low-power cores, e.g. variable bit-width integer MAC units and/or special memory architectures. Some low-power IoT cores offer 4-bit instructions, e.g. the GAP8 [7] powered by the PULP core [8], but up to now there are no ready-to-use IoT solutions for arbitrary bit-width scaling. Specialized neural cores, like the Imagination Series 2NX [9], offer multi-bit resolutions, yet with a power budget of few Watts. Other custom solutions, programmable [10] or hard-wired [11], are a too costly design option for the IoT domain. A patch for general cores is the use of specialized allocation strategies to store multiple weights within the same word and then feed the execution units in a proper manner. However, that may lead to huge performance overhead due to extra operations required to unpack/pack data from/to the memory banks [12]. Storing data with irregular bit-widths is another source of inefficiency as memory gets underutilized. In view of the above, it is questionable whether accuracy-driven, unconstrained compression methods can actually meet the needs of real-life applications.

This paper aims to benchmark theoretical against practical ConvNet implementations. The overall outcome of the assessment enables three main achievements. First, demonstrate that the implementation of practical ConvNets is governed by the actual memory constraint, and not just the model accuracy. Second, enumerate the optimal configurations in the memory-accuracy space when the optimization is conducted under very tight memory constraints. Third, quantify the distance between optimal (theoretical) configurations and the closest implementations that can be deployed on low-power MCUs.

The analysis is conducted using a novel two-stage pipeline driven by concurrent pruning and quantization: Prune-and-Quantize (**PaQ**). The optimization is hardware-aware, namely it integrates a smart selection of techniques tailored to meet the hardware specifications. As a key feature, the framework is built upon a lightweight memory-driven heuristic through which is possible to explore the memory-accuracy space efficiently. It also leverages a memory allocation model for bare metal environments together with an arithmetic emulator which do ensure accuracy and speed. Commercial or open-source frameworks do not have these features. The benchmarks are three realistic tasks that find application in the IoT domain: Image Classification (IC) on CIFAR-10 [13], Keyword Spotting (KWS) [14] and Facial Expression Recognition (FER) [15]. The hardware test-benches consist of two commercial boards powered by Cortex-M cores: NUCLEO-F412ZG (M4-256 kB), NUCLEO-F767ZI (M7-512 kB).

The remaining of the paper is organized as follows. Section II reviews state-of-the-art compression methods and describes the main motivations behind the optimization choices made in this work. Section III presents the hardware-aware compression framework adopted to collect the experimental results, with particular emphasis on the memory-driven **PaQ** heuristic. Section IV collects the main results and it drives the readers towards a proper understanding of memory-bounded ConvNets and their deployment, in particular: (*i*) the actual hardware requirements and how to judge the need of custom accelerators against general-purpose cores; (*ii*) the efficacy of the proposed

framework and its scalability. Finally, Section V concludes the paper.

II. BACKGROUND AND RELATED WORKS

In the early years of life, ConvNets were mainly optimized to improve accuracy. This brought to an exponential increase in size and complexity. With their massive use in cloud services, latency and energy consumption soon grew as key metrics to consider in order to guarantee scalability. Recently, the rise of edge computing pushed memory and storage capacity in the loop. During this fast evolution, several optimization methods have been introduced and tested on different architectures; a thorough overview is reported in [3]. However, the use of tiny MCUs restricts the optimization choices to a few of them only. This section gives a critical review of prior arts, motivating the choices implemented in this work.

A. PRUNING

Pruning techniques assume that ConvNets are over-parametrized, hence many parameters do not contribute to the predictive capacity of the model; rather, they introduce noise and therefore can be zeroed or removed. The pruning can be applied at different levels of spatial granularity, from *weight-level* to *filter-level*. The rule of thumb is that the finer the granularity, the better the accuracy-vs-compression trade-off. From a hardware perspective, a fine grain may result inefficient in general-purpose cores, while a coarse grain is more hardware-friendly, as it preserves the regularity of memory and resource allocation.

At the *weight-level* [16], the ConvNet is pruned in an unstructured manner, that is, every single weight can be removed, both from fully-connected layers and convolutional layers. To keep a regular shape of the matrix convolutions, weights are simply zeroed. This does not help to reduce the memory footprint directly, yet it increases the overall sparsity, i.e. the ratio between zero and non-zero parameters, which in turn enables compression methods based on sparse data representations [16]. To notice that the network may lose its regular structure and many optimizations for dense matrices can no longer be applied, e.g. matrix tiling [17]. Sparse representations get more efficient when supported by proper hardware. Examples for integrated accelerators are in the Texas Instruments TDAx processor [18] family or in the custom ASIC described in [19]. Unfortunately, low power budgets prevent the use of such components on MCUs. An equivalent software implementation based on compressed sparse row storage formats could be adopted, but the latency overhead due to extra operations is mitigated only when sparsity is pushed above a certain threshold [17].

At a middle level of granularity, weights are pruned as bunches of size such that the utilization (i.e. parallelism) of the SIMD unit is maximized [17]. Cores without a parallel SIMD unit do not benefit much from this approach incurring the same limitations of weight-level schemes.

At a *filter-level* [4], pruning works in a structured manner, namely following the topology of the ConvNet. Neurons (in the fully-connected layers) or convolutional filters (in the convolutional layers) are entirely dropped reducing both memory footprint and number of operations. Due to its regularity, it is the most suited strategy for RISC cores having an SRAM memory controlled with a standard indexing mechanism. For such specific reason, this work adopts filter pruning. One potential drawback is the risk of a higher accuracy drop because the information is drained out at a much faster pace (with respect to weight-pruning). However, short re-training stages allow recovering most of the information lost.

Different metrics have been proposed to drive the filter selection during the pruning process: (*i*) the ℓ_n-norm of the kernel weights, (*ii*) some statistics on the feature map's activation, like mean or standard deviation, (*iii*) the mutual information between activations and predictions, (*iv*) a combination of them. Yet, there is no consensus on the best option. The comparison study presented in [20] empirically demonstrates that the ℓ_1-norm proposed in [4] is a good compromise between accuracy and convergence time. Therefore, even if other metrics may improve the pruning confidence, this work relies on a revised version of the ℓ_1 scheme of [4] (Sec. III).

B. QUANTIZATION

While training is generally performed using single-precision floating-point (FP), a discrete integer representation is usually enough for inference. The aim of quantization is precisely that of alleviating the ConvNets complexity using a fixed-point (FX) representation for both weights and activations. Moving from FP to FX is not just an option, but it is mandatory for MCUs without a FP unit.

Pioneering works demonstrated that a FP ConvNet model can be squeezed to 16-bit and 8-bit FX [5] still preserving its expressive power. Later, extreme methods have been proposed to further decrease the model precision to ternary [21] or binary [22] weights, yet incurring large accuracy drops. The reduction of the bit-width allows a linear compression of the memory footprint, but mostly, it improves the memory bandwidth as multiple operands can be written/read within a single access. Obviously, this feature holds if quantization is run in compliance with the memory architecture. Moreover, since low-power cores provide few FX options, e.g. 8- and 16-bit for the Cortex-M cores, intermediate widths are not supported. For instance, a 32-bit SRAM line can host four 8-bit weights that can be easily fed to the execution units, while the use of 9-bit weights incurs in memory under-utilization and it requires specialized unpacking routines that affect latency [12]. ConvNet accelerators with arbitrary bit-width arithmetic, e.g. [9]–[11] (FPGA-based), are options. However, they dissipate more power than the MCUs targeted in this work ($\geq$ 300 mW vs tens of mW).

The literature presents plenty of schemes and techniques for accuracy-driven quantization. The following text gives a synthetic taxonomy which highlights the key aspects related to this work. Quantization is defined as *fixed* if equally

applied to all the layers, or *variable* when it uses different representations between layers [23].

The conversion scheme can be *linear* or *non-linear*. The *linear* scheme [5] makes use of a uniform distance between all the quantized weights. This is a trivial solution, yet the most suitable for generic hardware architectures due to its simplicity. The quantization range can be considered *symmetric*, if centred around zero, or *asymmetric* if shifted by a given offset. The choice is mainly driven by the shape of the weights distribution, but in general asymmetric quantization might result more accurate; on the other hand, it encompasses additional processing stages [24]. Furthermore, it is possible to quantize the weights using a *binary radix-point scaling*, or an *arbitrary linear scaling*. The former is implemented with simple bit-shift operations, the latter might result more accurate but it requires additional operations and hence more latency [24]. The *non-linear* approach makes use of custom conversion functions which map the full precision parameters onto irregularly interleaved ranges. It achieves higher accuracy than linear approaches as the irregular shape of the original distribution can be fitted with higher precision. The most popular examples are the *log-domain* [25] and *clustering* [16] approaches. These non-linear schemes are built using hash functions commonly implemented by custom hardware to improve performance [25] or, alternatively, by software routines that affect latency.

From this qualitative analysis it is thereby clear that, just as for pruning, there is a trade-off between flexibility and complexity: sophisticated schemes (i.e. linear with asymmetric/arbitrary scaling or non-linear) achieve higher accuracy as they can best fit to the original ConvNet shape, lightweight schemes (i.e. linear with binary/symmetric scaling) maximize the performance when small instruction set and limited hardware resources are available. The deployment of ConvNets on MCUs follows the simple rule *lighter is better*, hence the second class of methods is more suited. This motivates the choice made in this work: per-layer linear quantization scheme based on a power-of-two scaling with a fixed bit-width for all the layers (Sec. III).

C. PRUNING AND QUANTIZATION

Pruning and quantization work on distinct parts the ConvNet model: the number of parameters, the arithmetic precision of such parameters. They are orthogonal and can be jointly applied to achieve higher compression. There are multiple pairs of pruning rate and bit-width which ensure the same memory footprint, yet with a substantial difference in terms of accuracy. The challenge is to find the optimal solution among those available in the huge combinatorial search space. The authors of [6] showed that Bayesian optimization is an effective strategy. Even if they do not consider hardware constraints, it is fair to assume their method can be extended to a more hardware-friendly version. The contribution of our work differs, as the goal is not just to identify the most accurate compression, but rather to assess the distance to optimality of real-case hardware scenarios where memory becomes the primary concern. In that sense, the choice of the best optimizer practically fades, while stretching the coverage towards a more exhaustive search is important. Obviously, a brute-force search is unpractical due to the cardinality of the problem, hence smart heuristics are mandatory.

D. NEURAL ARCHITECTURE SEARCH

Neural Architecture Search (NAS) [26] is an emerging approach in which pruning and quantization get embedded into a global search where also the topological parameters of the ConvNet, e.g. number of layers, number of filters, connections between layers, etc., take part to the objective function,

Even though preliminary results show very promising, the exploration requires hundreds of GPUs and several days of training [26]. In this regard, the technology is in its infancy. The sentiment is that pruning and quantization are still valid options for local searches on a specific ConvNet topology (potentially obtained by NAS).

III. MEMORY-BOUNDED ConvNets

The memory footprint M_c of a ConvNet is function of the network parameters N_p (weights and biases) and the bit-width b used to represent the parameters. Given a target memory M_t, i.e. the actual on-chip memory that can be allocated[2], a memory-bounded ConvNet is a compressed version of the original floating-point ConvNet s.t. $M_c \leq M_t$ and the accuracy loss $\mathcal{L}$ is minimized. For different values of M_t there exists a set $\mathcal{P}$ of pairs $\{N_p, b\}$ that match M_t. Within $\mathcal{P}$, a pair $\{N_p^{\text{opt}}, b^{\text{opt}}\}$ is said *optimal* if it minimizes $\mathcal{L}$; a pair $\{N_p', b'\}$ is said *hardware-compliant* if can be ported on a physical device. b^{opt} can be of any integer value, while b' must be supported by a proper instruction set, i.e. $b' \in \{8, 16\}$ for our case study. An optimal pair is hardware-compliant if b^{opt} turns out to be 8- or 16-bit.

The assessment of the various solutions in the memory-accuracy space requires an evaluation framework to (*i*) implement a memory-constrained combination of pruning (to reduce N_p) and quantization (to reduce b), and (*ii*) emulate and deploy the compressed ConvNets.

A. FRAMEWORK OVERVIEW

The proposed framework is shown in Fig. 2. The toolchain is fed with a floating-point ConvNet (FP) trained within a standard development environment (e.g. PyTorch, TensorFlow) and it produces as output (*i*) the accuracy assessment of the compressed ConvNets that match the memory constraint (i.e. those with $\{N_p, b\} \in \mathcal{P}$) and (*ii*) the `.C` description of the compressed ConvNets that are hardware-compliant ($\{N_p', b'\} \in \mathcal{P}$). The `.C` is assembled using a neural-kernel library (the CMSIS-NN by ARM), then compiled and flashed on the target device. The assessment of those compressed ConvNets that do not fit the hardware (i.e. those with

[2] M_t can be lower than the physical memory available on-chip as other applications may run in background.

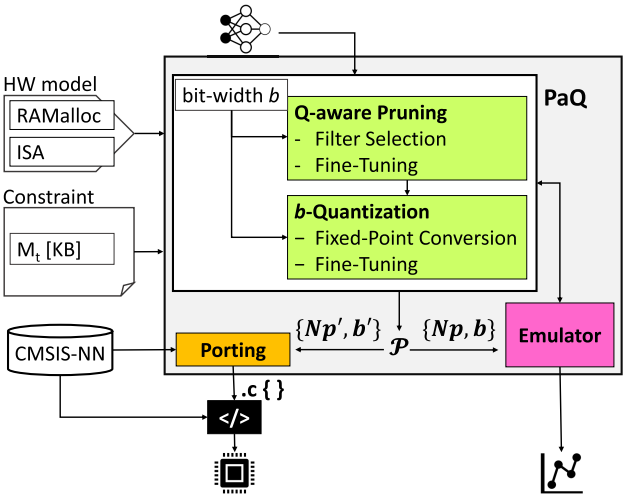


FIGURE 2. Framework overview.

$\{N_p, b\} \in \mathcal{P}$, $b \neq b'$) is run by means of an in-house fixed-point emulator; the same emulator is used to drive the fine-tuning stages (more details in Sec. III-D). The optimization kernel is called Prune and Quantize (**PaQ** hereafter) and it consists of two main stages: (*i*) quantization-aware (**Q-aware**) pruning; (*ii*) model quantization using a FX representation of b bits (**b-Quantization**). Both stages receive the parameter b as an internal constraint. As discussed in Sec. II, all the layers of a ConvNet share the same bit-width. This latter aspect has an impact on the compression speed, that is, quantization removes information at a faster pace. It is intuitive that removing a single filter on a layer is less intrusive than reducing the bit-width of all the weights of all the layers. A more interesting aspect is that there exists a circular dependence between pruning and quantization which frustrates the optimization. In fact, the value of b affects the number of filters that can be removed.

A model file containing the memory allocation strategy of the target architecture (*RAMalloc*) is used to estimate the physical RAM consumed during inference. As long as the HW-model and the neural-kernels library are available, the proposed framework works for any commercial MCU.

B. MEMORY MODEL

The amount of on-chip RAM allocated during the feed-forward pass depends on the implementation of the neural-kernels, which in turn is tightly coupled with the underlying hardware architecture. The following text describes the memory allocation policy deployed in the Cortex-M cores through the open-source CMSIS-NN library [24]. The same model can be extended to other architectures and/or libraries. The description takes as reference the convolution layers, which are the most expensive in terms of memory utilization,[3] but the model does also include fully-connected layers.

[3]State-of-the-art ConvNet designs rationed the number of fully connected layers in order to reduce the memory accesses.

Cortex-M MCUs are provided with a flash memory used to permanently store the ConvNet weights. At run-time, the same weights are block-loaded in a portion of the RAM referred to the *Weight Buffer* (WB). Most of the remaining RAM is taken by the *Activation Buffer* (AB), where partial results are temporarily stored. Partial results denote the input/output activations of each layer. Within a ConvNet, layers come with different topology, namely different number of filters, each of a different size. Therefore, each layer requires a different amount of memory, which is proportional to the dimension of its input and output tensors. Since ConvNets are executed layer-by-layer, AB is time-shared and its size is defined by the largest layer. Finally, a region of RAM is dedicated to temporary data structures internally used by the convolutional routines. To provide an accurate assessment of the memory requirements, the memory model includes their contribution. More specifically, the CMSIS-NN implements a tensor convolution as a matrix multiplication; this is done by converting the multidimensional input to a 2D array. The matrix, known as the Toeplitz matrix [27], is generated by the *im2col* routine which stores the result in a dedicated region of the RAM, the *im2col* buffer (I2CB). Similar to AB, the I2CB is time-shared among layers and its size is defined by the largest layer as well. In memory-constrained cores, a partial *im2col* routine is commonly adopted. It expands a selected portion of the input generating two columns of the Toeplitz matrix at a time. This allows reducing I2CB at the cost of some performance overhead.

The sum of the three buffers gives the overall RAM footprint, $M_c = WB + AB + I2CB$. Equation 1 gives the analytical model for a ConvNet of L layers represented with a physical bit-width b:

$$M_c = b \times \left[N_p + \max_{i \in L}(I_i + O_i) + \max_{i \in L}(im2col_i) \right] \quad (1)$$

The first term (N_p) refers to the WB buffer. It reflects the total number of parameters of the ConvNet. The number of weights is the product between the number of filters, the number of input channels and the size of kernels, while the number of biases equals the number of filters. For fully connected layers the number of weights is the product between the input dimension and the output dimension, while that of biases equals the dimension of the activation. The second term ($\max(I_i + O_i)$) refers to the AB buffer, with I_i and O_i the size of the activations (input and output respectively) of the layers. The last term ($\max(im2col_i)$) is for the I2CB buffer. The *im2col* processes one filter at a time, hence the size for a convolutional layer is the product of the three dimensions of a filter (Height, Width, and Depth), multiplied by 2 (two columns of the Toeplitz matrix); also in this case, the max operator takes the largest contribution among all the layers. As a side note, the contribution due to I2CB is usually pretty small, while WB is the dominant contribution. However, for ConvNets with a very compact topology, like those adopted into embedded applications, AB is not negligible (from 15% to 30% the overall RAM).

C. Q-AWARE PRUNING

In **PaQ**, filters are removed until the target memory M_t is met. Since M_c depends on the arithmetic precision, the pruning stage should be aware of b in order to optimize the number of pruned filters. The removal of a filter at the i-th layer simultaneously affects several parameters of eq. 1: the cardinality of the i-th and $i+1$-th convolutional layers (hence WB), the cardinality of the output activations of the i-th layer O_i (hence AB), the memory taken by the $im2col$ for the $(i+1)$-th layer (hence I2CB).

The iterative procedure of the Q-aware pruning is described in the pseudo-code of Algorithm 1. At each iteration, the least important filter from the least important layer (lines 3–5) is dropped. As previously introduced in Sec. II, the *importance* metric adopted to drive the filter selection is the ℓ_1-norm of the weights. The ℓ_1-norm is a good estimator to identify the components that affect less the prediction accuracy of the ConvNet [4], yet ensuring low complexity as it does not require the statistics on the activations of the hidden layers. Overall, it ensures a good trade-off between quality-of-results and complexity of the optimization loop. The framework adapts to other criteria however.

The loop iterates until the memory constraint M_t is met (line 2). The memory estimation is run using the memory model introduced in the previous section (embedded into the *RAMalloc* procedure). While the memory footprint is estimated on the base of the physical bit-width b, the model is not quantized yet at this stage. This gives to pruning a proper awareness of quantization.

After pruning, the network may experience a significant accuracy drop. The loss of information is however recovered (totally or partially depending on the actual constraint) by means of *fine-tuning* (line 7); the latter consists of a re-training stage (50 epochs in our experimental set-up) during which the weights are tuned using a standard error back-propagation scheme.

D. B-QUANTIZATION

After the Q-aware pruning, the model undergoes the actual quantization using a b-bit representation. As already motivated in Sec. II, the choice fell upon the most hardware-friendly quantization: (*i*) symmetric scheme, (*ii*) linear intervals [5], (*iii*) per-layer power-of-two scaling. Adopting a per-layer radix-point scheme brings higher accuracy without performance overhead.

The accuracy drop induced by quantization can be recovered (totally or partially depending on the actual constraints) through a customized fine-tuning stage that implements an incremental training procedure (iterated over 50 epochs in our experiments). The latter has the following main characteristics: the forward-propagation is run with fixed-point emulation; during back-propagation weights are kept in a floating-point format thus to allow small weight updates; weights are quantized at the end of every epoch using stochastic rounding.

Algorithm 1: Q-Aware Pruning Algorithm

Input: ConvNet [FP-32], Target Memory M_t, Bit-width b
Output: Compressed ConvNet

```
1 M_c = RAMalloc(ConvNet[FP-32], b)
2 while M_c > M_t do
3     Layer = Pick layer with lowest ℓ1-norm
4     Filter = Pick filter of Layer with lowest ℓ1-norm
5     Remove Filter
6     Update M_c
7 Fine-Tuning
8 return Compressed ConvNet
```

To emulate fixed-point arithmetic on GP-GPUs, an in-house emulator leverages the fake-quantization method introduced in [28]. It consists of a software wrapper that converts activations and weights (stored in fixed-point) to the 32-bit floating-point; after being processed, results are converted back to fixed-point.

E. PORTING AND EMULATION

Once compressed, the hardware-compliant ConvNets are translated in `C` code using the neural network kernels optimized for the target device. This work leverages the CMSIS-NN [24] library developed by ARM. It is a collection of optimized routines implementing the most common layers of deep neural networks and targeting the Cortex-M architecture. As already mentioned, the porting can be accomplished only for those bit-widths and memory budgets that meet the hardware constraints ($\{N_p', b'\}$). The framework provides emulation also for the other bit-width and memory constraints in order to estimate the distance between optimal and hardware-compliant solutions, which is one of the objectives of this work. The emulator is the same used within the **PaQ** flow.

IV. EXPERIMENTAL RESULTS

We used the proposed **PaQ**-based flow to explore the memory-accuracy space. The analysis aims to assess the optimality of hardware-compliant implementations and quantify their distance (in terms of accuracy) from theoretical solutions. This section is organized as follows. First, we introduce the ConvNets adopted as test-cases, together with the datasets used for the training stage and the evaluation. Second, we describe the hardware boards used as test-bench. Third, we present the collected results and discuss the key findings. Finally, we provide additional insights to validate **PaQ** and justify the selected optimization strategies. Table 1 summarizes the notations used throughout the text, together with their definition.

A. BENCHMARKS, DATASETS AND TRAINING

The experimental setup consists of three different tasks: Image Classification (IC), Keyword Spotting (KWS), Facial Expression Recognition (FER). All of them find application

TABLE 1. Table of abbreviations.

Notation	Description
M_c	Memory footprint of the ConvNet
M_t	Target memory
N_p	Number of network parameters (weights and biases)
b	Bit-width (ranging from $b_{\min} = 2$ to $b_{\max} = 16$, step one bit)
M_b	Memory footprint of the ConvNet quantized with b-bit and w/o pruning
$\mathcal{P}$	Set of pairs $\{N_p, b\}$ that matches M_t
$\mathcal{L}$	Top-1 accuracy loss
$\mathcal{L}_{\max}$	Top-1 accuracy loss boundary ($= 0.5\%$)
$\mathcal{T}$	Pleateau area collecting the $\{N_p, b\}$ pairs s.t. $\mathcal{L} \leq \mathcal{L}_{\max}$
P_x	Best-accuracy point
P_n	Pareto points in the memory-accuracy space ($n \in \mathbb{N}$)
PaQ-8	PaQ solutions with 8-bit
PaQ-16	PaQ solutions with 16-bit
Δ	Accuracy difference between optimal and hardware-compliant implementations

TABLE 2. Benchmark overview. Convolutional layer with shape (c_{out}, k_h, k_w), fully-connected layer with shape (c_{out}) and max-pooling layer with shape (k_h, k_w); k_h and k_w are the height and width of input planes in pixels, while c_{out} refers the number of output channels.

Application	IC	KWS	FER
Dataset	CIFAR-10 [13]	Speech Commands [14]	FER2013 [15]
Input	$3 \times 32 \times 32$	$1 \times 32 \times 40$	$1 \times 48 \times 48$
ConvNet Topology	Conv (32,5,5) MaxPool (3,3) Conv (32,5,5) MaxPool (3,3) Conv (64,5,5) MaxPool (3,3) FC (10)	Conv (64,20,8) MaxPool (1,3) Conv (64,10,4) MaxPool (1,1) FC (32) FC (128) FC (12)	Conv (32,3,3) Conv (32,3,3) Conv (32,3,3) MaxPool (2,2) Conv (64,3,3) Conv (64,3,3) Conv (64,3,3) MaxPool (2,2) Conv (128,3,3) Conv (128,3,3) Conv (128,3,3) MaxPool (2,2) FC (7)
Top-1 Acc.	82.80%	86.75%	66.48%

in several domains, like robotics, human-machine interface, and retail. Each task is powered by a different ConvNet model which has been carefully selected among those that can be realistically deployed on IoT devices. Tab. 2 reports the topology of the models together with the top-1 classification accuracy achieved using a floating-point representation (w/o any further optimization). Results are consistent with those available in the literature. Both training and testing are run in PyTorch, version 0.4.1. The training is iterated over 150-epochs using the Adam algorithm [29] with the following settings: learning rate 1e-3, linear decay 0.1 every 50-epochs, batch size of 128 samples randomly picked from the training set. Test set and training set are fully disjointed.

1) IMAGE CLASSIFICATION (IC)

It is the image recognition on the popular *CIFAR-10* dataset. There are 32×32 RGB images [13] evenly split in 10 classes, each class with 50000 and 10000 samples for the training set and test set respectively. Like in [24], the adopted ConvNet is taken from the Caffe framework [32]. It consists of three convolutional layers interleaved with max-pooling and one fully-connected layer.

2) KEYWORD SPOTTING (KWS)

A well-known application in the field of speech recognition, which is hard to deploy on low-power devices. However, when the problem is simplified to simple command detection (used as triggers), e.g. "Yes", the task achieves an affordable level of complexity.[4] The reference data set is the Speech Commands Dataset [14]; it counts of $65k$ $1s$-long audio samples collected during the repetition of 30 different words by thousands of different people. The goal is to recognize

[4]https://www.tensorflow.org/tutorials/sequences/audio_recognition

TABLE 3. Development boards adopted as test-bench for the assessment.

Board	Core	RAM	Flash	Frequency
NUCLEO-F412ZG [30]	Cortex-M4	256 KB	1 MB	100 MHz
NUCLEO-F767ZI [31]	Cortex-M7	512 KB	2 MB	216 MHz

10 specific keywords, i.e. "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", "Go", out of the 30 available words. Samples that do not belong to the 10 categories are labeled as "unknown". An additional "silence" class is made up of background noise samples (i.e. pink noise, white noise, and human-made sounds). The training set and test set collect 56196 and 7518 samples respectively. The adopted ConvNet model is the *cnn-trad-fpool3* described in [33]. It is made up of two convolutional layers, two max-pooling layers, and three fully-connected layers. The network is fed with the spectrogram of the recorded signal which is obtained through the pre-processing pipeline introduced in [33] (extraction of $time \times frequency = 32 \times 40$ inputs w/o any data augmentation).

3) FACIAL EXPRESSION RECOGNITION (FER)

It is about inferring the emotional state of people from their facial expression. Quite popular in the field of visual reasoning, this task is very challenging as many face images might convey multiple emotions. The reference data set is the *Fer2013* from the Kaggle competition [15]. It collects 32297 48×48 grayscale facial images split into 7 categories: "Angry", "Disgust", "Fear", "Happy", "Sad", "Surprise", "Neutral". The training set counts of 28708 samples, while the remaining 3589 are kept as the test set. The ConvNet model[5] shows nine convolutional layers evenly spaced by three max-pooling layers and one fully-connected layer.

B. HARDWARE SPECIFICATIONS AND TOOLS

As testbenchs we used two off-the-shelf boards powered with Cortex-M cores by ARM. As shown in Tab. 3, the selected devices cover different ranges of performance (Frequency) and memory space (RAM). The deployment on the target board is powered by the CMSIS-NN library v.5.4.0 provided by ARM. The `.C` source file is compiled using the GNU Arm Embedded tool-chain, version 6.3.1.

Within the **PaQ** flow, the classification accuracy is measured by means of the emulator mentioned in Sec. III-D. The tool is tuned for the ARM Cortex-M integer unit. Experiments were conducted on a GP-GPU workstation powered with a Titan GTX-1080 Ti by NVIDIA. Extensive testing revealed 100% match with results collected on the NUCLEO boards. The same emulator is used for the accuracy assessment of those compressed ConvNets that cannot be flashed into the ARM cores, whereas the hardware-compliant ConvNets are evaluated on-board (see the flow depicted in Fig. 2). The *RAMalloc* memory model is cross-validated with the results provided by the `gcc` compiler (all the variables are statically allocated) and those returned by tracking the memory usage at run-time (feature available with the mbed-os operating system,[6] version 5.11.0).

C. ACROSS THE MEMORY-ACCURACY SPACE

The exploration is run for a discrete set of memory constraints, i.e. $M_t \in [M_{b_{\min}}, M_{b_{\max}}]$, $b_{\min} = 2$, $b_{\max} = 16$, step one bit; M_b refers to the memory footprint of the ConvNet quantized with b bits w/o any pruning (e.g. M_2 is the memory after a 2-bit quantization). For intermediate memory constraints, i.e. $M_t \in (M_{b_i}, M_{b_{i+1}})$, the accuracy is interpolated (more details in Sec. IV-D). The collected results of the three applications are illustrated in Fig. 3(a), 3(b) and 3(c). The plots show the top-1 accuracy for every pair $\{N_p, b\} \in \mathcal{P}$. The yellow line highlights the implementations where only the b-Quantization (label Q) applies, i.e. no filters pruned. Since the Q-aware pruning skips the filter pruning as soon as the M_t is met, there might be memory-compliant solutions which belong to this line, e.g. in Fig 3(a) the 2-bit quantization alone meets the memory constraint of 33 kB. The region above the yellow line (light transparency) covers trivial implementations dominated by quantization, i.e. those for which $M_t > M_b$, while the exploration of the region below is more interesting as it brings the following considerations.

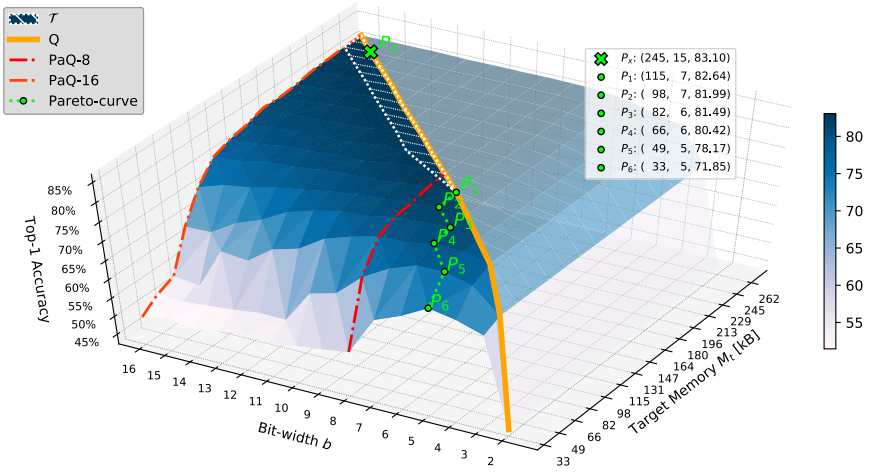
1) WEAKNESS OF ACCURACY-DRIVEN OPTIMIZATIONS

There is a plateau $\mathcal{T}$ (hatched area in the plot) where the accuracy gets very close to that of the original FP model, namely pruning and quantization impact accuracy marginally. Without loss of generality, we assume that a pair $\{N_p, b\}$ belongs to $\mathcal{T}$ if the accuracy drop with respect to the best-accuracy point (P_x, marked with the green cross in the plots and reported in the first row of Tab. 4) is less or equal than 0.5%. The existence of $\mathcal{T}$ is nothing new as ConvNets are often redundant due to over-parametrization [16]. The area of $\mathcal{T}$ may depend on the complexity of the task or the network topology.

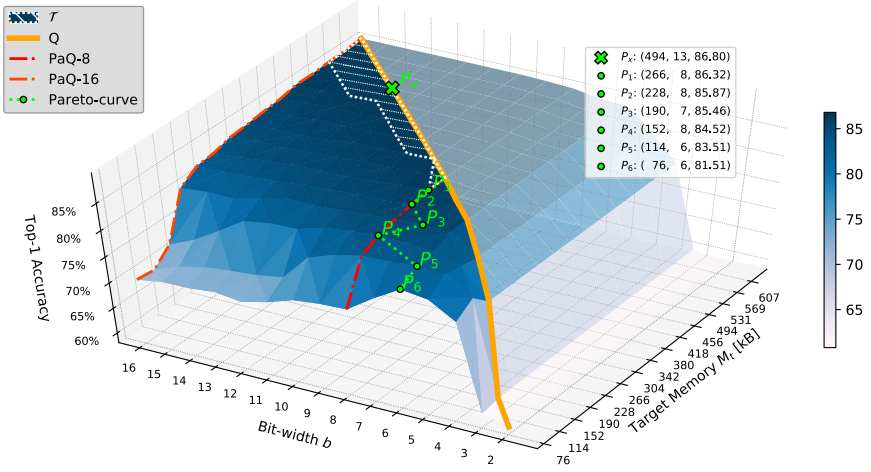
The accuracy-driven compression techniques proposed in the literature, e.g. [6], search for an unique combination of pruning and quantization which ensures the largest compression within a given accuracy loss $\mathcal{L}_{\max}$. Assuming a realistic constraint, e.g. $\mathcal{L}_{\max} = 0.5\%$, which is the same value used to define $\mathcal{T}$, the solution they return can be identified in

[5] Inspired by https://github.com/JostineHo/mememoji

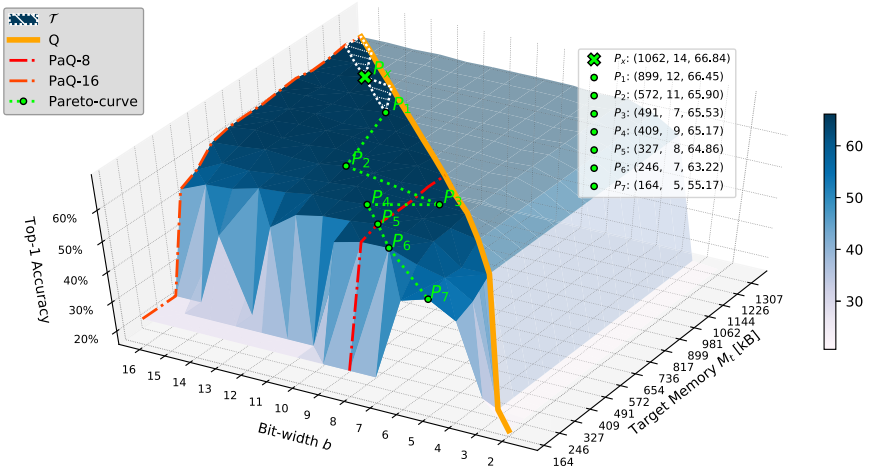
[6] https://os.mbed.com/blog/entry/Tracking-memory-usage-with-Mbed-OS/



(a) IC



(b) KWS



(c) FER

FIGURE 3. Solutions of **PaQ** in the memory-accuracy space for the three tasks under analysis (a) IC, (b) KWS, (c) FER. The solution with the best accuracy (P_x) is marked with the green cross. The hatched area (enclosed by the white dotted curve) defines the plateau ($\mathcal{T}$) collecting all solutions s.t. the accuracy loss $\mathcal{L} \leq 0.5$ w.r.t P_x. The yellow line (**Q**) highlights the solutions where only b-quantization was applied. The red dash-dotted lines (**PaQ-8** and **PaQ-16**) denote the solutions obtained with **PaQ** using 8- and 16-bit respectively, i.e. the implementations deployed on the physical device. The green dotted line connects the Pareto points (P) in the memory-accuracy space, i.e. all the solutions that have superior accuracy w.r.t. all the other points with the same target memory M_t. The right box collects the absolute coordinate of P_x and each Pareto point in the format (target memory, bit-width, top-1 accuracy).

TABLE 4. Optimal vs. hardware-compliant implementations under different memory constraints M_t: column *Pareto* reports the Pareto points P as for the plots of Fig. 3; columns PaQ-8 and PaQ-16 refer to solutions obtained with the proposed PaQ flow using 8- and 16-bit respectively which are the red lines in Fig. 3; column Δ collects the distance (the lower is better) between optimal and hardware-compliant implementations for both PaQ-8 and PaQ-16 in terms of accuracy. Cells corresponding to those implementations with a top-1 accuracy $\ll$50% have not been filled.

	M_t	*Pareto*			PaQ-8			PaQ-16		
		P	b	Top-1	b	Top-1	Δ	b	Top-1	Δ
IC	245	P_x	15	83.10	8	82.85	0.25	16	82.86	0.24
	115	P_1	7	82.64	8	82.44	0.20	16	77.31	5.33
	98	P_2	7	81.99	8	81.40	0.59	16	72.52	9.47
	82	P_3	6	81.49	8	80.79	0.70	16	65.21	16.28
	66	P_4	6	80.42	8	78.85	1.57	16	54.85	25.57
	49	P_5	5	78.17	8	71.64	6.53	16	53.00	25.17
	33	P_6	5	71.85	8	54.68	17.17	16	50.00	21.85
KWS	494	P_x	13	86.80	8	86.38	0.42	16	86.20	0.60
	266	P_1	8	86.32	8	86.32	0.00	16	83.80	2.52
	228	P_2	8	85.87	8	85.87	0.00	16	83.48	2.39
	190	P_3	7	85.46	8	85.28	0.18	16	81.60	3.86
	152	P_4	8	84.52	8	84.52	0.00	16	73.11	11.41
	114	P_5	6	83.51	8	83.00	0.51	16	70.42	13.09
	76	P_6	6	81.51	8	75.16	6.35	16	70.78	10.73
FER	1062	P_x	14	66.84	8	65.34	1.50	16	65.23	1.61
	899	P_1	12	66.45	8	65.34	1.11	16	65.59	0.86
	572	P_2	11	65.90	8	65.48	0.42	16	63.47	2.43
	491	P_3	7	65.53	8	64.75	0.78	16	58.43	7.10
	409	P_4	9	65.17	8	64.61	0.56	16	55.92	9.25
	327	P_5	8	64.86	8	64.86	0.00	16	-	-
	246	P_6	7	63.22	8	63.03	0.19	16	-	-
	164	P_7	5	55.17	8	-	-	16	-	-

our formulation as $\{N_p, b\} \in \mathcal{T}$ s.t. M_c is minimized. This solution represents the lower right corner of the plateau $\mathcal{T}$, denoted with P_1 (second row of each benchmark in Tab. 4).

An accuracy-driven, memory-unconstrained optimization of this kind might return ConvNets that do not fit into the physical memory. Let's consider FER for instance, the optimal implementation would take 899 kB of RAM using 12-bits, a configuration which is simply too large for our target devices. The focus of this work is on the region below such theoretic optimum, a region we refer as the deep memory space, and the proposed framework is specifically built to run this exploration. One may argue that other meta-heuristics, like Bayesian Optimization, can be guided towards this region of interest by integrating the memory footprint in the cost function. That is true in general, but those methods perform better in optimization rather than fine exploration. Moreover, the multi-objective function may result biased by the importance weights adopted.

2) MEMORY-ACCURACY PARETO CURVE

There is a Pareto curve in the deep memory space, the green dotted line in the plots. As already discussed, P_1 corresponds to the pair $\{N_p, b\}$ inside $\mathcal{T}$ having the smallest memory footprint. Instead, the remaining Pareto points lay outside $\mathcal{T}$ and represent those implementations that meet lower memory constraints at the cost of larger accuracy loss, $\mathcal{L} > 0.5\%$. The existence of these points is somehow intuitive, but a quantitative analysis may reveal interesting trends. The exact values of target memory M_t and bit-width b are reported in Tab. 4 together with the top-1 accuracy they achieve. As the numbers suggest, for many configurations the obtained accuracy gets very close to the best accuracy, yet ensuring substantial memory reduction. For instance: KWS shows a small accuracy drop of 1.34% (from 86.80% to 85.46%) with 62% of memory compression (from 494 kB to 190 kB); FER goes even better by showing 46% memory reduction (from 1062 kB to 572 kB) within an accuracy loss $< 1\%$ (from 66.84% to 65.90%). Similar conclusions can be inferred from the comparison among the other Pareto points.

3) OPTIMALITY OF HARDWARE-COMPLIANT SOLUTIONS

A more interesting analysis concerns the distance (measured as difference in accuracy) between the implementations on the Pareto curve and the implementations which are hardware-compliant, i.e. the pairs $\{N_p', b'\}$ with $b' \in [8, 16]$ highlighted with the red dash-dotted curves in the plots (labels PaQ-8 and PaQ-16 respectively). The top-1 accuracy for PaQ-8 and PaQ-16 are given in Tab. 4, together with the distance from the Pareto curve (column Δ). The results show that PaQ-8 outperforms PaQ-16 (smaller Δ). There are only two

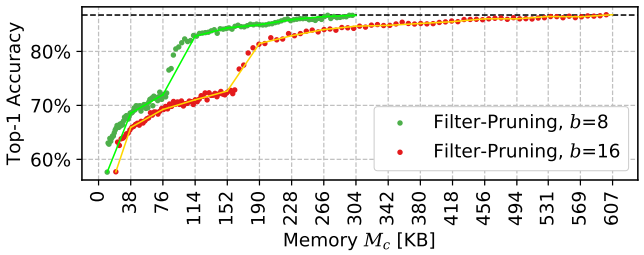


FIGURE 4. Top-1 accuracy vs. memory footprint for KWS.

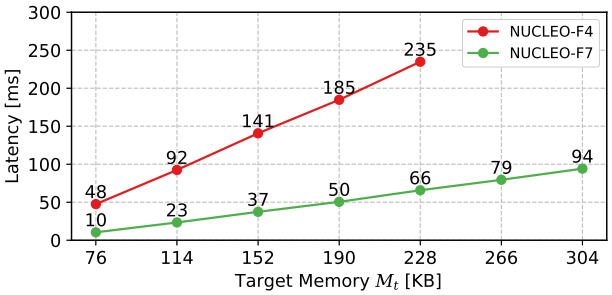


FIGURE 5. Average inference time per sample of PaQ-8 solutions on KWS.

exceptions, i.e. IC at $M_t = 245$ kB and FER at $M_t = 899$ kB, yet with a mere distance (0.25% in the worst case). The actual reason is that under the same memory budget, the 8-bit model has more remaining filters, hence the accuracy of 8-bit model is higher than the corresponding 16-bit one. In other words, the 8-bit models stop pruning earlier than 16-bit. This can also be proved by looking at numbers collected in Tab. 4, FER benchmark under a memory constraint $M_t = 327$ kB: the 16-bit model is so highly pruned that the accuracy falls down to impractical values, while the 8-bit model meets the memory constraint with less filters pruned and hence lower accuracy loss. The key insight is that a bit-width below the 8-bit mark is needed only for very tight constraints. For instance, KWS with memory constraint $M_t = 76$ kB, where the PaQ-8 implementation shows $\Delta \geq 1\%$, or FER with memory constraint $M_t = 164$ kB, where PaQ-8 cannot ensure reasonable accuracy. The conclusion is that arbitrary bit-widths are really needed in few specific cases and the adoption of specialized architectures needs to be assessed carefully.

D. VALIDATION OF PaQ

1) EFFICACY OF THE PROPOSED MEMORY-DRIVEN COMPRESSION

As described in Algorithm 1, the proposed version of filter pruning is memory-driven, i.e. the optimization loop ends as soon as the memory constraint is met. To motivate this stopping criteria, we provide the analysis of a pruning strategy where the constraint is given in a direct form, i.e. *number of filters to be pruned*; once pruned, the models undergo a quantization stage and then fine-tuning to recover accuracy. Using the number of filters as a control knob, it is therefore possible to span the entire memory range. Fig. 4 shows the results for KWS; the plot collects the top-1 accuracy achieved with 8- and 16-bit. The lines follow a pseudo-monotone trend: the lower the memory, the lower the classification accuracy. Negligible ripples are due to the noise introduced by fine-tuning. This fully justifies our choice: to stop the search as soon as the constraint M_t is met gives the highest accuracy for that specific M_t. The same trend holds for every bit-width used in our experiments (not shown in the figure for the sake of readability). Furthermore, the linear interpolation adopted to estimate the accuracy when $M_t \in (M_{b_i}, M_{b_{i+1}})$ is also validated. Indeed, the plot shows that accuracy is a piece-wise linear function of memory. The same considerations hold for the other ConvNet benchmarks.

2) ON THE SCALABILITY OF THE PROPOSED HARDWARE-DRIVEN OPTIMIZATION

The adopted **PaQ** scheme is hardware-friendly, namely, memory compression improves latency too. We refer to this kind of schemes as *latency proportional*. Fig. 5 shows the average latency for one feed-forward pass of the ConvNet used in KWS. The analysis is conducted under different memory constraints (the same reported in Tab. 4).

As PaQ-8 dominates PaQ-16 (please refer to the previous section), the reported results are for 8-bit only. The execution time is measured using the timer API provided by the mbed-os operating system and averaged over the entire test set. The experiments were run on the boards reported in Tab. 3, labeled as NUCLEO-F4 and NUCLEO-F7 for brevity. The NUCLEO-F4 board has a maximum RAM of 256 kB, therefore larger models cannot be deployed.

Adopting pruning and quantization schemes that preserve the regularity on the ConvNet topology is the key to achieve a direct proportionality between inference time and memory footprint. The choices implemented in the proposed framework go in this direction as they have been conceived to (*i*) reduce the number of memory accesses, (*ii*) alleviate the cost of the *im2col* procedure, and (*iii*) reduce the number of operations as the memory footprint gets smaller. The result is the linearity shown in the plot. The same trend holds for the other benchmarks.

3) EXECUTION TIME

The **PaQ** flow takes a few minutes for each fine-tuning stage (50 epochs each). The actual execution time may vary depending on the complexity of the ConvNet and the memory constraint. The worst case is the largest benchmark (FER): 25 minutes on average for each $\{M_t, b\}$ pair, 80% spent for the fine-tuning stages. A significant reduction can be achieved limiting the number of retraining epochs. Early stopping policies may be introduced to serve this purpose as it was done in other works to prevent over-fitting [34] or accelerate the training stage [35]. While the speed-up of the **PaQ** flow is out of the scope of this work, Table 5 supports our claim showing the number of fine-tuning epochs after which **PaQ** is already able to reach the highest top-1 accuracy.

Collected numbers refer to the average over all the pairs $\{M_t, b\}$ of the exploration space. For the three benchmarks, both pruning and quantization converge much earlier than the

TABLE 5. Average number of fine-tuning epochs to achieve the maximum top-1 accuracy on the test set.

	Q-aware Pruning	b-Quantization
IC	30.7	22.1
KWS	27.7	15.4
FER	18.7	13.5

50-epoch threshold we set for safety, revealing the potential margins.

V. CONCLUSION

This work introduced an accurate analysis of memory-bounded ConvNets. The study was conducted through an assessment framework that provides a hardware-conscious exploration of the memory-accuracy space. This represents a key differentiation factor with respect to existing optimization flows. Overall, our study proves that the design space exploration of compressed ConvNets enables to identify when custom hardware is needed. Understanding these cases is paramount to reduce the implementation costs, especially in the context of lightweight IoT applications. With the proposed framework, we demonstrated that optimal solutions are not that far from those that can be effectively ported into general-purpose hardware, even under the tight memory constraints posed by off-the-shelf MCUs. The results revealed that custom accelerators are not always needed: just for some applications and extreme memory compression they ensure savings. When deployed on commercial boards, hardware-compliant solutions have proven to be latency proportional, that is, performance linearly improves with the memory compression. This is an important feature which validates the efficiency and the scalability of the adopted prune-and-quantize strategy. As an additional insight, the analysis highlighted a natural bond between pruning and quantization that can be used in future works to drive and speed-up the optimization process.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[2] A. Gomez, F. Conti, and L. Benini, "Thermal image-based CNN's for ultra-low power people recognition," in *Proc. 15th ACM Int. Conf. Comput. Frontiers*, 2018, pp. 326–331.

[3] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[4] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2016, *arXiv:1608.08710*. [Online]. Available: https://arxiv.org/abs/1608.08710

[5] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, H. Yang, and Y. Wang, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2016, pp. 26–35.

[6] F. Tung and G. Mori, "CLIP-Q: Deep network compression learning by in-parallel pruning-quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7873–7882.

[7] *GAP8 Hardware Reference Manual Version 1.5.5*. Accessed: Oct. 20, 2019. [Online]. Available: https://gwt-website-files.s3.amazonaws.com/gap8_datasheet.pdf

[8] F. Conti, D. Rossi, A. Pullini, I. Loi, and L. Benini, "PULP: A ultra-low power parallel accelerator for energy-efficient and flexible embedded vision," *J. Signal Process. Syst.*, vol. 84, no. 3, pp. 339–354, 2016.

[9] *Why the PowerVR Series2NX NNA is the Future of Neural Net Acceleration*. Accessed: Oct. 20, 2019. [Online]. Available: https://www.imgtec.com/blog/why-the-powervr-2nx-nna-is-the-future-of-neural-net-acceleration

[10] Y. Umuroglu, L. Rasnayake, and M. Själander, "BISMO: A scalable bit-serial matrix multiplication overlay for reconfigurable computing," in *Proc. 28th Int. Conf. Field Program. Logic Appl.*, Aug. 2018, pp. 307–3077.

[11] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks," in *Proc. 45th Annu. Int. Symp. Comput. Archit.*, Jun. 2018, pp. 764–775.

[12] M. Rusci, "Quantized NNs as the definitive solution for inference on low-power ARM MCUs?: Work-in-progress," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2018, p. 12.

[13] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.222.9220

[14] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018, *arXiv:1804.03209*. [Online]. Available: https://arxiv.org/abs/1804.03209

[15] *Challenges in Representation Learning: Facial Expression Recognition Challenge*. Accessed: Oct. 20, 2019. [Online]. Available: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

[16] S. Han, H. Mao, W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: https://arxiv.org/abs/1510.00149

[17] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 548–560, 2017.

[18] M. Mathew, K. Desappan, P. K. Swami, and S. Nagori, "Sparse, quantized, full frame CNN for low power embedded devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 11–19.

[19] B. Moons and M. Verhelst, "An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, Apr. 2017.

[20] D. Mittal, S. Bhardwaj, M. M. Khapra, and B. Ravindran, "Recovering from random pruning: On the plasticity of deep convolutional neural networks," in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, Mar. 2018, pp. 848–857.

[21] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient AI applications," in *Proc. Int. Joint Conf. Neural Netw.*, May 2017, pp. 2547–2554.

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: ImageNet classification using binary convolutional neural networks," in *Computer Vision—ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 525–542.

[23] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, Mar. 2016, pp. 1–8.

[24] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm cortex-M CPUs," 2018, *arXiv:1801.06601*. [Online]. Available: https://arxiv.org/abs/1801.06601

[25] S. Vogel, M. Liang, A. Guntoro, W. Stechele, and G. Ascheid, "Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base," in *Proc. Int. Conf. Comput.-Aided Design*, 2018, p. 9.

[26] A.-C. Cheng, J.-D. Dong, C.-H. Hsu, S.-H. Chang, M. Sun, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Searching toward Pareto-optimal device-aware neural architectures," in *Proc. Int. Conf. Comput.-Aided Design*, 2018, p. 136.

[27] A. Vasudevan, A. Anderson, and D. Gregg, "Parallel multi channel convolution using general matrix multiplication," in *Proc. IEEE 28th Int. Conf. Appl.-Specific Syst., Archit. Processors*, Jul. 2017, pp. 19–24.

[28] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5784–5789, Nov. 2018.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980
[30] *NUCLEO-F412ZG*. Accessed: Oct. 20, 2019. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-f412zg.html
[31] *NUCLEO-F767ZI*. Accessed: Oct. 20, 2019. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-f767zi.html
[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, Nov. 2014, pp. 675–678.
[33] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. 16th Annu. Conf. Int. Speech Commun. Assoc.*, 2015, pp. 1–5.
[34] L. Prechelt, "Early stopping—But when?" in *Neural Networks: Tricks of the Trade*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds., 2nd ed. Berlin, Germany: Springer, 2012, pp. 53–67. doi: 10.1007/978-3-642-35289-8_5.
[35] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2017, *arXiv:1705.10823*. [Online]. Available: https://arxiv.org/abs/1705.10823

MATTEO GRIMALDI received the M.Sc. degree in biomedical engineering from the Politecnico di Torino, in 2017, where he is currently pursuing the Ph.D. degree in control and computer engineering. His research interests include efficient algorithms and design strategies for resource-driven compression and optimization of deep learning models.

VALENTINO PELUSO received the M.S. degree in electronic engineering from the Politecnico di Torino, in 2015, where he is currently pursuing the Ph.D. degree with the Electronic Design Automation Group, Department of Control and Computer Engineering. In February 2016, he joined the Department of Control and Computer Engineering, Politecnico di Torino, as a Research Assistant. His main research interest includes the optimization and compression of deep learning models for their deployment on low-power embedded systems.

ANDREA CALIMERA received the M.Sc. degree in electronic engineering and the Ph.D. degree in computer engineering from the Politecnico di Torino. He is currently an Associate Professor of computer engineering with the Politecnico di Torino. His research interests include design automation of digital circuits and embedded systems with emphasis on optimization techniques for low-power and reliable ICs, dynamic energy/quality management, logic synthesis, design flows, and methodologies for emerging computing paradigms and technologies.