Optimization Tools for ConvNets on the Edge


Doctoral Dissertation
Doctoral Program in Computer and Control Enginering (32.nd cycle)

Optimization Tools for ConvNets on the Edge

Valentino Peluso
* * * * * *

Supervisors
Prof. Enrico Macii, Supervisor
Prof. Andrea Calimera, Co-supervisor

Doctoral Examination Committee:
Prof. Rene van Leuken, Referee, Delft University of Technology
Prof. Francesco Regazzoni, Referee, Università della Svizzera italiana
Prof. Nicola Bombieri, Università degli Studi di Verona
Prof. Maurizio Martina, Politecnico di Torino
Prof. Fabrizio Lamberti, Politecnico di Torino

Politecnico di Torino
September 3, 2020

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Valentino Peluso
Turin, September 3, 2020

Summary

The advancement of low power technologies and design strategies for integrated circuits, together with the improvement of wireless communication systems and infrastructures, enabled a massive deployment of smart IoT sensors able to sense the physical world. Meanwhile, thanks to the recent breakthroughs in Artificial Intelligence (AI), Convolutional Neural Networks (ConvNets) in particular, computers took a further step towards the human intelligence, acquiring skills like autonomous learning and decision making. The integration of such AI technologies into the end-nodes of the IoT is the premise for a new paradigm-—-Artificial Intelligence of Things (AIoT)-—-where sensors will evolve from passive data collectors to active intelligent devices able to infer the meaning of data locally. This shift will thus enable the design of more efficient, scalable, and secure digital ecosystems.

The migration from the cloud to the edge devices poses several issues due to the complexity of modern ConvNet models. The quest for high accuracy has brought the design of ConvNets with billions of hidden parameters and millions of arithmetic operations, thus preventing their deployment on resource-constrained, low-power devices. To tackle this problem, there is a wide consensus that the design of portable ConvNets needs a proper understating of the hardware resources available from the early stage of the training process. Specifically, the optimization of ConvNets should encompass a multi-objective problem formulation, involving extra-functional metrics like memory, energy, and power, besides accuracy. Rather than improving accuracy, the goal is to identify the trade-off frontiers in the design space to pick the best solution meeting the resource constraints. In practice, this can be achieved with algorithmic transformation built upon compression methods that exploit the intrinsic resiliency of neural networks to identify and remove those parts of the model with less contribution to accuracy.

The search for optimality, however, 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. Moreover, the high number of dimensions to explore introduced an additional level of complexity overlooked by most of the existing works. A problem formulation neglecting these aspects might result too weak, or unsuited, for real-life applications. This

optimization problem recalls the design of digital integrated circuits (ICs), where multiple conflicting constraints should be addressed, like area, power, and performance. Which of these dimensions gets the highest priority depends on the use-cases, the cost requirements, and in general on the design specifications. For instance, some use-cases require fast processing, whereas those applications with relaxed timing constraints are often limited in area and power consumption. To serve this purpose, the EDA tools for the IC segment have been engineered in a modular way, providing a collection of computer-aided methods with specific goals and formulations. Designers are free to build their own pipeline, integrating the most suited tools depending on their needs.

Applying the same approach to the optimization of ConvNets seems a natural choice and it is exactly the main topic of this dissertation. Indeed, a one-size-fits-all solution does not exist due to the diversity of applications and the hardware back-end. Rather, dedicated solutions are needed for the analysis and optimization of memory, energy, and power, and their integration undergoes a vertical implementation, from software to hardware.

Moreover, compression methods originally applied at design-time can be operated at run-time, leading ConvNets to become dynamic algorithms that modulate the resource usage depending on external triggers raised at the application level (e.g. the battery level or the severity of the task). Once again, whereas the adaptive/dynamic control of resources is a well-known standard in hardware design (e.g. dynamic power management), it is a less explored field in the optimization of ConvNets. For this reason, special attention is devoted to this topic, demonstrating that algorithmic knobs for run-time reconfiguration introduce additional degrees of freedom in the optimization space.

This dissertation is organized into three main parts, each of them focusing on a specific design goal. In the first part, it focuses on aggressive memory optimization which is particularly suited for devices with extreme memory constraints ($<$1 MB). It first presents *Prune and Quantize*, a smart heuristic to explore the memory-accuracy space when neural network compression is pushed towards the deep memory region. Then, it introduces *Encoding-Aware Sparse Training*, a novel training technique for sparse ConvNets designed to maximize the compression rate of standard encoding algorithms. In the second part, the energy optimization problem is addressed elaborating the idea of *Adaptive ConvNets*, a solution that allows ConvNets to trade accuracy for energy at run-time. Two different implementations conceived for software-programmable neural accelerators with mixed-precision arithmetic are discussed and validated. In the third part, the focus shifts on power optimization, with emphasis on dynamic power and thermal management. A novel power distribution scheme named *FINE-VH* is presented, together with an automatic design methodology and the integration in a standard EDA

flow, which enables a more efficient Dynamic-Voltage-Frequency Scaling (DVFS) policy. At last, the efficacy of ConvNets under thermal and accuracy constraints is assessed using DVFS as the main control knob.

Overall, the technical contributions described in this dissertation offer a collection of design&optimization tools for ConvNets to (i) assess at the design-time both functional and extra-functional metrics, (ii) explore different dimensions of the design-space (iii) identify the optimal solution that can meet the hardware requirements.

Acknowledgements

I would like to thank Prof. Enrico Macii, for giving me the opportunity to join the EDA research group.

I am extremely grateful to prof. Andrea Calimera, for its valuable advice and the countless hours devoted to discuss my ideas and support my research.

I also acknowledge the project "SENSEI - Sensemaking for Scalable IoT Platforms with In-Situ Data-Analytics: A Software-to-Silicon Solution for Energy-Efficient Machine-Learning on Chip," funded by Compagnia di San Paolo, with Università di Bologna as academic partner and ST Microelectronics as non-academic partner.

It is not his possession of knowledge, of irrefutable truth, that makes the man of science, but his persistent and recklessly critical quest for truth.

Karl Raimund Popper
The Logic of Scientific Discovery

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context & Motivation

Deep Convolutional Neural Networks (ConvNets hereafter) are brain-inspired computational models that have brought breakthroughs in many fields, such as computer vision [1], speech recognition [2] and natural language processing [3]. In its more general embodiment, a ConvNet consists of a sequence of processing stages commonly called layers. At each layer, transformations learned during the training phase project raw data over a multi-dimensional space where standard classifiers can outperform the accuracy of humans [4]. From a computational viewpoint, ConvNets are nothing more than matrix multiplications between pre-trained parameters (the synaptic weights of the hidden neurons) and the input data.

The most common use-case for ConvNets is image classification where a multi-channel image (e.g. RGB) is processed producing as output the probability that the subject depicted in the picture belongs to a specific class of objects or concepts (e.g. car, dog, airplane, etc.). One can see this end-to-end inference process as a kind of data compression: high-volume raw-data (the pixels of the image) are compressed into a highly informative tag (the resulting class). In this regard, the adoption on the Internet-of-Things (IoT) is disruptive: distributed smart-objects with embedded ConvNets may implement data-analytics at the edge, near the source of data [5], with advantages in terms of predictability of the service response time, energy efficiency, privacy and, in general, scalability of the IoT infrastructure.

Starting from the astounding results obtained by Krizhevsky et al. [1] in 2012, ConvNets evolved quickly achieving impressive results. However, it is possible to recognize some basic characteristics common to many models. The organization of a ConvNet reflects the hierarchical structure of the primary sensory areas of

the visual cortex [6]. The rationale is the same implemented by the human brain, indeed: extract, evaluate, and combine features that have been learned to be common among the majority of samples belonging to the same class. The feature extraction is hierarchical, namely, low-level features are extracted first and then used to extract features at a higher level. Intuitively, edges may form shapes, which in turn may form objects. Features at the higher levels are then used to classify the content of the picture. This sequential procedure is built through a chain of computational layers that implement algebraic operations on matrices. Fig. 1.1 shows the topology of a generic ConvNet. Convolutional layers run matrix-matrix convolution between their local filters and the multi-dimensional map generated by previous layers; filters correspond to different features to extract. Activation layers (blue blocks) introduce non-linearity in the feature space applying specific functions, e.g. Rectified Linear Unit (ReLU), on the output map produced by the convolutional layers; non-linearity helps to amplify semantic differences. Finally, reduction layers apply a sub-sampling of the activation maps using functions like max-pooling or average pooling; sub-sampling helps in reducing the cardinality of the features, increase the level of abstraction and make classification less sensible to geometrical distortions. Once all the features have been extracted, the last stages of a ConvNets implement the actual classification. Fully connected layers serve this purpose using multi-layer perceptrons that apply geometric separation. At the very last stage, a softmax function is used to score the available labels; the one with the highest probability identifies the class.

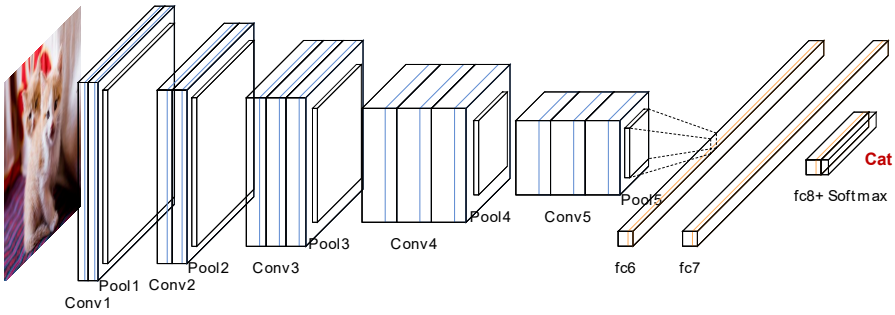


Figure 1.1: Schematic view of a ConvNet Architecture for Image Classification.

The design of ConvNets encompasses a training stage during which the synaptic weights of the hidden neurons are learned using a back-propagation algorithm (e.g. the Stochastic Gradient Descent [7]). The learning is supervised and accuracy-driven, namely, it adjusts the weights such that an accuracy loss function evaluated over a set of labeled samples is minimized. Although the basic theory is known since long time, only the advent of GPUs made ConvNets evolving from pure mathematical models to practical solutions. Indeed, the learning stage involves a

brute-force search that can be afforded only by massively parallel processing, just what GPUs offer.

Since a proper design and optimization methodology does not exist yet, today's ConvNets are typically oversized in terms of both computational and memory resources. Even small networks require billions of multiplications and millions of weights to be stored. This prevents their use on low-power embedded platforms which offer low storage capacity, low computational power, and limited energy budget. How to design ConvNets that fit the stringent resource constraints while preserving classification accuracy is an open issue.

Specifically, the processing of ConvNets under resource constraints is a technological challenge at the intersection between the theories of machine learning and those of computer engineering, where vertical strategies spanning from the design of highly efficient circuits and systems to algorithmic optimization and model training must be brought together.

The research community is attacking the problem from two opposite directions. At the software level, with the design of more compact network architectures and optimization techniques reducing the cardinality of ConvNets, yet preserving competitive prediction accuracy. At the hardware level, with the design of custom accelerators [8], or reconfigurable spatial architectures [9]. The achieved results are impressive: performance close to the TOPS with power consumption of few hundreds of μW [10].

In this chapter, we first review the relevant trends in this context (Section 1.2): Section 1.2.1 focuses on the software level, summarizing the main advancements in neural network design; Section 1.2.2 focuses on the hardware level, presenting a brief overview of the available hardware platforms for edge inference. Then, we introduce the goals and the main contributions of our research, together with the overall organization of the dissertation (Section 1.3).

1.2 Towards Edge Inference

1.2.1 Software Optimization: the Evolution of ConvNet Architectures

In the early years of life, ConvNets were mainly optimized to improve accuracy, without any concern on their computational complexity. Since higher accuracy can be afforded increasing the number of parameters and the operations, the evolution of ConvNets experienced an exponential increase in the resources needed for inference. This trend is clearly depicted in Fig. 1.2, which shows the accuracy-complexity trade-off achieved by the most common open-source ConvNets trained on the Imagenet dataset. The rise of accuracy from 50.6% (bottom-left MobileNet)

to 84.4% (top-right EfficientNet) comes at the cost of 132× more parameters.

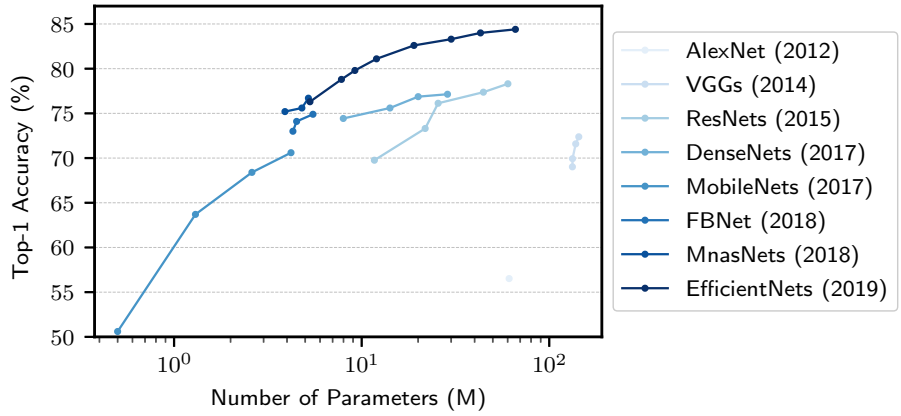


Figure 1.2: Top-1 Accuracy vs. Number of Parameters of open-source ConvNets on the Imagenet dataset.

The growing need of portability pushed designers to consider hardware related metrics besides accuracy. Rather than searching for the most accurate solution, designers soon focused on a wider exploration of the design space, aiming to offer a set of solutions in the accuracy-complexity space from which the user can pick the best configuration meeting the hardware constraints. To this purpose, modern ConvNets follow a common characteristic: they are built upon an architectural template that can be resized depending on memory and performance requirements. This can be observed in Fig. 1.2, which reports the different existing versions of each network. The different versions are generated through the implementation of different scaling knobs: depth, width, and resolution. Their joint application enables to span the design space. The depth indicates the number of processing layers; its adoption as scaling knob was introduced by VGG [11]. The width refers to the number of filters of each convolutional layer; one standard practice is to define a multiplicative factor—the width multiplier—that indicates the ratio of filters related to the baseline implementation. The resolution represents the spatial dimension of the input data, which also affects the cardinality of the inner tensors. Width and resolution scaling were pioneered by MobileNet [12] and are still adopted by more recent networks like EfficientNet [13]. The latter introduced the concept of *compound scaling*, which seeks the optimal balance between the three knobs. Regardless of the knob adopted, lower resources imply lower accuracy. The challenge is to identify Pareto fronts in the design space.

In this context, Neural Architecture Search (NAS) [14] has emerged as the most promising technology to discover novel topology configurations. The basic principle of NAS is to integrate the network hyper-parameters, like the number of layers, the filter shapes, and the kind of operations, in the training loop besides the

network weights. This enables to fully automate the search for new architectures, but at the cost of much longer training time, from few hours up to several days of GPU processing. FBnet, MnasNet, and EfficientNet (see Fig. 1.2) are some representative networks generated by NAS, achieving better trade-offs than previous hand-crafted designs.

Although the impressive advancements, there is still room for further optimization. In this regard, this dissertation investigates orthogonal strategies to lower the complexity of ConvNets, generated either by manual or automatic design. Our contribution is threefold. First, with compression methods that remove redundant information (i.e. pruning) and/or exploit the resiliency of ConvNets through low-precision arithmetic (i.e. quantization). Second, with the design of dynamic models that can modulate their resource requirements at run-time. Third, with a joint co-operation between software and hardware optimizations, especially when dealing with power minimization. A thorough overview of existing compression strategies related to our works is presented throughout the dissertation.

1.2.2 Hardware Optimization: Platforms for Edge Inference

Despite the research efforts towards the development of optimized and compressed models, the deployment of ConvNets remains an open problem. Besides complexity, one of the major obstacles is the diversity of the existing hardware architectures for edge inference. The main candidate solutions include (but are not limited to) (i) custom accelerators based on application-specific designs (ASICs) (ii) general-purpose embedded CPUs, and (iii) ultra low-power microcontroller units (MCUs). These options are not in competition, rather the optimal choice depends on the application, as each use case has different compute, area, and cost requirements. This heterogeneity prevents the development of a one-size-fits-all optimization pipeline. Moreover, the hardware can offer additional degrees of freedom in the optimization, providing specific features and/or knobs that enable to achieve a better quality of results with smaller and faster models. A comprehensive optimization of edge ConvNets should therefore include a set of different algorithmic solutions, each of them tailored to the underlying hardware. Whereas this aspect is often overlooked by current research, it receives particular attention in this dissertation. In this section, we summarize the pros and cons of the alternative hardware architectures, highlighting the opportunities and challenges of each case.

Application Specific Integrated Circuits

Custom ASICs for neural network inference are today integrated into many system-on-chips for the mobile segment. Their success lies in their performance stability and power efficiency, thanks to the availability of memory logic and functional cores dedicated to host only deep learning workloads. They consist of a processing unit, usually built as an array of processing elements (PEs), assisted by a distributed memory hierarchy that helps to improve the bandwidth and to avoid frequent accesses to large and energy-hungry off-chip memories (DRAM). The processing unit is an array of tightly coupled processing elements (PEs) that receive the incoming operands from the main inputs or the upstream nodes and produce partial results that are locally stored and/or passed downstream following a pre-defined data-flow. Thanks to the distributed communication among PEs, an array of N×N elements consumes N+N inputs and produces N×N outputs at each cycle, thereby enabling massive parallel computation. Systolic units are a particular class of such processing arrays that leverage the intrinsic locality of the convolution operation and enable efficient data reuse. Architectural templates of this kind can be re-scaled to fit the requirements of both cloud [15] and edge [16] systems.

Due to the success of low-precision ConvNets, custom ASICs often integrate multi-precision arithmetic units, which enable to exploit the resilience of neural networks to arithmetic errors in favor of higher compression ratio. Since the degree of resilience, hence the precision, depends on the network and the application, the trend is to develop flexible arithmetic units that support a fine-grain control on the bit-width [17].

The architecture presented in [18] is one of the most representative examples with hardware support for mixed-precision computing. The core is built using an array of multi-precision PEs that can deliver 1 × 16-bit MAC or 2 × 8-bit MAC per cycle. Running to low-precision, the PEs can operate at a lower voltage-frequency level, which means lower power consumption, keeping the same throughput of the full-precision. Hence, the joint application of voltage and precision scaling improves the overall energy efficiency. More recent architectures offer additional degrees of freedom with a finer selection of the bit-width: BitFusion [19] is an ASIC accelerator with support of 2-, 4-, 8-, and 16-bit operations; BISMO [20] is a time-scheduled design for FPGAs that implements a bit-serial matrix multiplication algorithm; UNPU [21] is an ASIC accelerator that provides an arithmetic instruction set with arbitrary bit-widths from 1- to 16-bit. Different solutions for variable precision are available also in commercial products, like the NVIDIA Turing GPU [22] (4- and 8-bit) and the PowerVR Series 2NX by Imagination [23] (arbitrary bit-width from 4 to 16-bit).

However, standard ConvNets do not exploit efficiently the reconfigurability of

the hardware. The bit-width is a fixed parameter, selected at design-time such that the prediction quality keeps the same level of the full-precision model. ConvNets are therefore designed and implemented as static models. Instead, the design of *Adaptive ConvNets* that can self-tune their arithmetic precision at run-time could enable dynamic trade-offs between accuracy and complexity. To this purpose, there is a strong need to develop novel algorithmic optimizations. In Chapter 3, we address this need, presenting two different strategies for the design and implementation of Adaptive ConvNets.

Embedded CPUs

Embedded CPUs represent the most attractive solutions to deploy ConvNets on mobile platforms like smartphones. Indeed, today most mobile devices integrate high-performance multi-core system-on-chips, which can become intelligent systems through a software update. Moreover, CPUs offer higher flexibility than custom hardware, as they can run also other tasks than inference, such as driving peripherals and sensors, and therefore host comprehensive sensing environments.

An example is the Cortex-A family by ARM. It comes with the NEON unit, an advanced single-instruction multiple-data (SIMD) architecture that supports float and integer vector operations. Proper use of such a module allows to maximize the parallelism, reduce the memory accesses, and hence achieve substantial performance boost [24].

The success of CPUs in the field of embedded ConvNets is demonstrated by the growing number of inference engines [25]. These tools provide an abstract user interface coupled with specialized deep learning libraries that collect handwritten kernels built with NEON intrinsics and/or assembly code. The most advanced and stable solutions include Arm NN by Arm, ncnn by Tencent, and TensorFlow Lite by Google.

Due to the integration of high-performance components in small form-factor, the primary bottleneck of CPU-based platforms is their limited thermal design power. Indeed, the execution of intensive workloads like ConvNets raises several concerns related to power dissipation. This problem is often neglected by standard optimization pipelines, which evaluate the inference performance in nominal operating conditions, a too optimistic choice not reflecting actual use-cases. To provide a more realistic analysis, Section 4.2 presents a thorough characterization of the achievable power-performance trade-offs in embedded CPUs.

Microcontreller Units

To sustain the scalability of the IoT, there is an increasing demand to deploy ConvNets on ultra low-power sensing systems powered by tiny MCUs. Coupled

with low-power sensors (e.g. MEMS), MCUs are extremely popular in several fields, spanning from battery-powered applications (e.g. wearable technologies) to high-end devices for navigation and positioning, predictive maintenance in industry, and augmented virtual reality components.

However, the migration towards such tiny devices introduces several challenges. ConvNets previously processed in the cloud with plenty of resources shall be processed in a mW power envelope using processor cores with tight resource budgets and low storage capacity. As an example, the RISC-based MCUs designed by ARM for the IoT segment (i.e. the Cortex-M family) integrate limited integer arithmetic options (16- and 8-bit) and very small on-chip memories (few hundreds of KB). Clearly, this limits the complexity of ConvNets that can be hosted.

As will be discussed in Chapter 2, MCUs poses additional constraints besides low performance and memory, originating from the limited instruction set. Due to the lack of proper hardware components, most of the existing compression strategies turn out to be inefficient when implemented on MCUs. Considering the actual constraints is therefore paramount when devising algorithmic optimizations. To answer this need, we present two novel compression strategies tailored to port ConvNets in memory-bounded MCUs.

1.3 Objectives & Contribution

Due to the growing diversity of applications and devices, a *one-size-fits-all* approach to optimize and deploy ConvNets might generate sub-optimal solutions. Instead, there is an urgent need for a comprehensive collection of dedicated tools, responding to different design goals and tailored to different use-cases. In this context, the objective of this dissertation is threefold:

- Develop cross-layer optimizations for software-to-silicon mapping of ConvNets, with vertical strategies aware of the opportunities and the limitations of the hosting systems in order to maximize the portability and the efficiency.
- Offer a collection of methods for the analysis and the compression of ConvNets, addressing different design goals: memory, energy, and power. Designers can adopt the most suited solutions depending on their needs.
- Devise dynamic knobs to extend the achievable accuracy-complexity trade-offs. Whereas ConvNets are built as static computational graphs, they can leverage software-based (e.g. arithmetic precision) and/or hardware-based (e.g. power management) knobs to adapt at run-time their computational effort depending on context variables, e.g. the available resource budget.

Figure 1.3 summarizes the organization of the thesis, which is split in three main chapters, one for each of the optimization metrics/objectives.

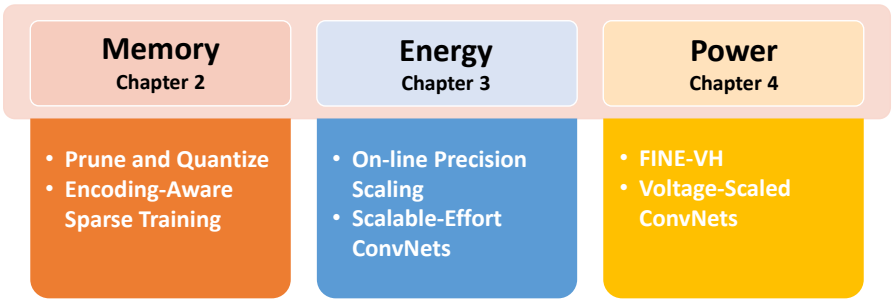


Figure 1.3: Schematic view of the the dissertation outline.

Chapter 2 focuses on memory-driven optimizations. We first review existing algorithmic compression techniques, with emphasis on their applicability on real use-cases. The conducted analysis reveals the limits of state-of-the-art techniques, which either cannot be ported on tiny IoT devices like micro-controllers or yield poor performance due to the lack of proper hardware support. We therefore present *Prune and Quantize*, an efficient heuristic to explore the memory-accuracy design space, assessing the figure of merits of theoretic vs. practical implementations. Second, we introduce a novel training procedure, referred to as *Encoding-Aware Sparse Training*, aiming at reducing the memory footprint of network weights. As demonstrated by the collected results, our proposal outperforms the accuracy-compression trade-offs of existing reduction strategies based on a sparse representation of data.

Chapter 3 deals with energy-driven optimizations, with emphasis on run-time scaling strategies for dynamic energy-accuracy trade-offs. The chapter first presents a taxonomy of *Adaptive ConvNets*, with a review and classification of related techniques. As key contributions, the chapter introduces two novel implementations of Adaptive ConvNets: *On-line Precision Scaling* and *Scalable-ConvNets*. The former is a design&optimization flow that makes use of per-layer precision scaling strategy to modulate energy and accuracy at run-time. The latter brings the dynamic nature of the human reasoning to static ConvNets, leveraging the concept of multilevel classification. Just like as humans, we show that ConvNets intrinsically learns to classify at different abstraction layers, spanning from coarse categories to fine details. Clearly, high-level classification is easier than low-level classification, creating the opportunity to either reduce the effort or improve the accuracy.

Chapter 4 addresses power-driven optimizations. The most common strategy to reduce power consumption is to design custom hardware components dedicated to run a specific task, inference in our case. Intuitively, architectures tailored to a single task can reach higher performance at lower power consumption than

general-purpose cores. Rather than focusing on architectural design, our contribution investigates an orthogonal direction, i.e. power management. First, we propose a novel power distribution scheme called *FINE-VH*, aiming to bring power management at an ultra-fine granularity, i.e. within the functional units. Through an optimization&emulation framework fully integrated in a standard EDA flow, we demonstrate that FINE-VH pushes power savings beyond the theoretical limits of standard DVFS strategies. Second, we explore the power-performance trade-offs of voltage-scaled ConvNets running on general-purpose systems, e.g. embedded CPUs, where limited cost budgets prevent the adoption of dedicated architectures or custom power distribution schemes. Our study reveals interesting trends, suggesting (i) new directions for thermal-aware training procedures and (ii) useful guidelines for a better co-operation between hardware and software knobs.

Finally, Chapter 5 concludes the dissertation, summarizing the main findings of our research.

Chapter 2

Memory-Driven Optimization

An efficient deployment of ConvNets on the network edge requires smart optimizations to fit large models into memory-limited cores. In this chapter, we focus on low-cost IoT applications [26, 27], where small form-factor and limited energy budget are the major bottlenecks. In these cases, the software stack runs on tiny RISC cores mounted on-board of off-the-shelf embedded platforms. Without loss of generality, we consider as a case study the microcontroller units (MCUs) of the Cortex-M family by ARM[1], reported in Table 2.1. Low-power MCUs integrate small on-chip RAM memories (from 4 to 32 KB for the M0, from 256 to 512 KB for the M7—depending on the chip-set). Off-chip memories are often not integrated as they degrade several metrics, like performance, energy, integration cost, endurance, and reliability. The instruction set architecture (ISA) supports only few 16- and 8-bit integers operations (16- and 8-bit) and often no floating-point. Furthermore, they lack parallel units to accelerate vector operations. A tiny 2-lane Single Instruction Multiple Data (SIMD) unit is available in the M4 and M7 cores only. These hardware figures prevent the deployment of ConvNets of some practical use. Even the most compact topology involves the processing of multidimensional tensors of a size that impedes full on-chip storage and real-time processing. A pragmatic solution is to play with algorithmic optimizations, e.g. through compression techniques that shrink the size of the inner tensors. Among the available options [28], pruning and quantization have emerged as the most adopted techniques. Both applied during or after training, they aimed to remove redundant information, namely, those components of the ConvNet with low impact on the prediction quality. Pruning removes less important parameters. Quantization reduces the bit-width of the arithmetic representation of the parameters. As demonstrated by the latest advancements, a joint combination of the two achieves

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

state-of-the-art [29]. Compressed ConvNets show fewer parameters to store, hence fewer operations to run.

Cortex-M	**Power** (μW/MHz)	**RAM** (KB)	**Floating** (32b)	**Integer** (16b,8b)	**SIMD Unit** (#lane)
M0	5.3	4-32	No	Yes	No
M3	11.0	32-128	No	Yes	No
M4	12.3	128-256	No/Optional	Yes	2
M7	33.0	256-512	No/Optional	Yes	2

Table 2.1: Hardware figures of the Cortex-M IoT MCUs by ARM.

In the first part of this chapter (Section 2.1), we summarize the existing strategies to alleviate the memory requirements of ConvNets. As will be discussed later, they are based on theoretic studies overlooking the constraints of real hardware. For this reason, they might returns solutions centered on specific configurations that result difficult to be ported on tiny general-purpose cores.

To address this limitation, we present two novel strategies to deploy ConvNets on memory-bounded cores, namely *Prune and Quantize* (Section 2.2) and *Encoding-Aware Sparse Training* (Section 2.3). The former is a two-stage framework that efficiently explores the memory-accuracy space using a lightweight, hardware-aware heuristic optimization. The latter is a novel memory-constrained training procedure that brings quantized ConvNets towards deep compression.

The content of this chapter is a revised version of our previous works found in [30, 31, 32].

2.1 Background on Neural Network Compression

With the advent of edge computing, memory and storage capacity have become the main design constraints to consider in order to guarantee the portability of ConvNets. For this reason, many compression methods have been conceived and validated on different platforms [28]. However, only a few of them can be implemented on tiny MCUs. This section reviews prior works in this topic, motivating the choices implemented in our proposals.

2.1.1 Pruning

Based on the assumption that ConvNets are over-parameterized, pruning techniques aim to identify and remove those parameters that less contribute to the

expressive power of the model. The pruning can operate at different degrees of spatial granularity. *Weight-pruning* is the finest level, filter-pruning the coarsest. As a general rule of thumb, a finer grain enables better trade-offs between accuracy and compression. Regarding the implementation on-board of general-purpose cores, a coarse grain achieves better performance, as it keeps regular memory and resource allocation.

Weight-pruning [33] follows an unstructured pattern, i.e. every single weight can be removed, both from fully-connected layers and convolutional layers. To preserve the regular structure of the multi-dimensional tensors, weights are simply zeroed. Unstructured sparsity does not reduce the memory footprint directly but generate long sequences of zero values that can be encoded with standard compression schemes [33]. As the regular structure may get lost, many techniques to accelerate the processing of dense matrices can no longer be applied, e.g. matrix tiling [34]. The efficient processing of sparse data requires dedicated hardware components supporting advanced indexing mechanisms. Examples of this kind can be found in the Texas Instruments TDAx processor [35] family or the custom ASIC described in [36]. Unfortunately, low power budgets impede the integration of such components on MCUs. Even though an equivalent software implementation based on compressed sparse row storage formats could be adopted, the latency overhead due to extra operations is mitigated only when sparsity overcomes a certain threshold [34].

Group-pruning removes bunches of adjacent weights. The group-size is a parameter tuned to maximize the utilization of parallel processing units [34]. Clearly, cores without a parallel unit do not benefit from this approach incurring the same limitations of weight-pruning. As will be discussed in Section 2.2, we present a novel training procedure based on group pruning. However, in our procedure, the group size is tailored to minimize memory rather than improve performance.

Filter-pruning [37] follows structured patterns: neurons (in the fully-connected layers) or entire convolutional filters (in the convolutional layers) are removed reducing both memory footprint and number of operations. Thanks to its regularity, its implementation is straightforward in RISC cores integrating an SRAM memory controlled with a standard indexing mechanism. One potential downside is the risk of a higher accuracy loss, because the information is drained out at a much faster pace (than weight-pruning). However, short few incremental training iterations help to recover the information lost.

Due to the lack of formal methods to identify the weakest filters, different proxies have been studied to determine the filter priority during the pruning process: (i) the ℓ_n-norm of the kernel weights, (ii) some statistics on the layers output, like mean or standard deviation, (iii) the mutual information between intermediate

outputs and final predictions, (*iv*) a combination of them. Yet, there is no consensus on which is the most efficient. The comparative evaluation conducted in [38] proves that the ℓ_1-norm [37] guarantees a good compromise between accuracy and convergence time.

2.1.2 Quantization

While training needs single-precision floating-point (FP) to achieve convergence, inference can reach the same prediction quality even with lower precision representation. The main benefit of quantization is the memory reduction of network weights and activations, which gets stored with fewer bits. In this regard, fixed-point (FX) arithmetic is the most adopted representation, thanks to its straightforward implementation on general-purpose cores. However, fixed-point quantization is not just an option, but it is mandatory for MCUs without support for floating-point arithmetic.

Seminal works like [39] showed 16-bit and 8-bit FX ConvNets reach the same prediction quality of the original FP representation. More recent methods explored extreme quantization down to ternary [40] or binary [41] representations, yet incurring a significant accuracy drop. The benefits of quantization go beyond memory reduction. The bit-width scaling alleviates the memory bandwidth utilization as multiple operands can be processed within single access. This benefit holds if the bit-width is compliant with the memory indexing scheme. In addition, since low-power cores support few FX instructions, e.g. 8- and 16-bit for the Cortex-M processor family, intermediate and lower widths are not viable. 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 underutilization and it requires additional unpacking routines that affect latency [42]. Custom arithmetic units supporting arbitrary bit-widths are integrated in ConvNet accelerators like [43, 19, 20]. However, they dissipate more power than the MCUs targeted in this chapter ($\geq 300\,\text{mW}$ vs. tens of mW).

The literature presents plenty of schemes and techniques for accuracy-driven quantization. The following text provides a broad classification which highlights the key aspects related to techniques presented in this chapter. Quantization is defined as *fixed* if the same bit-width is shared among the layers, or *variable* if the bit-width can vary across the layers [44].

Two conversion schemes do exist: *linear* or *non-linear*. The *linear* scheme [39] makes use of a uniform distance among all the quantized weights. This is a straightforward solution, yet the most adopted for generic hardware architectures thanks to its simplicity. The quantization range can be *symmetric*, if centered around zero, or *asymmetric* if shifted by a given offset. The choice is driven by the shape of the weights distribution, but in general asymmetric quantization guarantee higher

accuracy, at the cost of additional processing stages [45]. Moreover, it is possible to quantize the weights using a *binary radix-point scaling*, or an *arbitrary linear scaling*. The former can be implemented through simple bit-shift operations, the latter might result more accurate but it requires additional operations and hence more latency [45]. The *non-linear* scheme makes use of custom conversion functions which map the full precision parameters onto irregularly interleaved ranges. It achieves higher accuracy than linear schemes as it enables to fit distribution with irregular shapes. The most common examples are based on logarithmic representations [46] or *clustering* [33]. Overall, non-linear schemes are based on complex hash functions, which can be implemented by dedicated hardware units to accelerate their processing [46] or by equivalent software routines that affect latency.

Our qualitative analysis reveals a simple yet important consideration, i.e. there is a trade-off between flexibility and complexity: sophisticated schemes (e.g. linear with asymmetric/arbitrary scaling or non-linear) guarantee higher accuracy as they can best fit the original weights distribution, lightweight schemes (e.g. linear with binary/symmetric scaling) guarantee faster processing, especially when implemented on general-purpose cores with limited hardware resources. The deployment of ConvNets on MCUs follows the simple rule *lighter is better*, hence the second class of methods is preferred.

2.1.3 Pruning and Quantization

Pruning and quantization affect different sources of information: the trainable parameters and their arithmetic precision, respectively. As they are orthogonal strategies, their joint application can achieve higher compression. Intuitively, multiple combinations of pruning rate and bit-width reach the same memory compression, but with a substantial difference in terms of accuracy. Identify the most accurate settings in this huge search space is extremely challenging. The authors of [29] showed that Bayesian optimization could be a promising strategy. Even though they overlook the hardware constraints, it is fair to assume that their method can be extended to a more hardware-friendly version. The contribution of our proposals differ. Rather than identifying the most accurate configuration, we aim to run an extensive exploration of the memory-accuracy space, pushing our analysis towards the extreme memory region (<1 MB). In this regard, the development of the most efficient optimizer practically fades, whereas extending the coverage towards a more comprehensive exploration gets higher priority. Clearly, the size of the search space impede an exhaustive search, calling for the development of smart heuristics.

2.2 PaQ: Prune and Quantize

Most of the methods discussed in the previous section are accuracy-driven, namely, they try to identify the model setting that achieves the highest memory compression with minimum accuracy loss, ideally zero. Model settings have a relative meaning here: it is the largest selection of weak parameters for pruning, the smallest bit-width of the arithmetic representation for quantization, the optimal pruning-to-quantization ratio for joint methods.

Intuitively, accuracy cannot be the only metric that drives the optimization. An optimization loop unconstrained in terms of memory may return configurations still not meeting the requirements of the target hardware. Moreover, since quantization and pruning are lossy methods, they get controlled through a user-defined accuracy threshold. Since the cost function is unknown, the selection of such an accuracy threshold is blind. Solutions with similar accuracy, namely equivalent in terms of quality, might have very different memory footprint. Furthermore, finding the right balance between pruning and quantization still remains an open problem, suggesting that design space exploration is more reliable than multi-objective optimization. Obviously, the large number of possible model settings makes exhaustive exploration unpractical, which demands smart heuristics. Not least, quantization below the 8-bit mark (e.g. from 7- to 2-bit [29]) remains a theoretic study as it requires custom hardware components which are not an option in low-power cores, e.g. integer units with flexible bit-widths and/or dedicated memory architectures. Some low-power IoT cores support 4-bit instructions, e.g. the GAP8 [47] powered by the PULP core [48], but up to now there are no ready-to-use IoT solutions for arbitrary bit-width scaling. Specialized neural accelerators, like the Imagination Series 2NX [43], offer variable bit resolutions, yet with a power budget of few Watts. Other custom solutions, programmable [20] or hard-wired [19], are a too costly design option for the IoT domain. A patch for general cores is the use of software-based allocation strategies to pack multiple weights within the same word and then properly feed the execution units. However, the additional operations needed to manage the data might generate an unacceptable performance overhead [42]. As an additional source of inefficiency, storing data with irregular bit-widths lowers memory utilization. These observations raise a natural question, whether accuracy-driven, unconstrained compression methods can meet the needs of real-life use-cases.

In this section, we aim to benchmark theoretical against practical ConvNet implementations. The overall outcome of the assessment enables three main achievements. First, demonstrate that the practical implementation of 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. Lastly, assess the

accuracy difference between optimal (theoretical) configurations and the closest implementations that can be ported on low-power MCUs.

The analysis is conducted through a novel two-stage pipeline driven by concurrent pruning and quantization: Prune-and-Quantize (**PaQ**). The optimization is hardware-aware, namely, it involves 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 which enables efficient exploration of the memory-accuracy space. It also makes use of a memory allocation model for bare metal environments together with an arithmetic emulator which does ensure accurate and fast evaluation. Existing training frameworks do not offer these features. We validated PaQ on three realistic tasks which find application in the IoT domain: Image Classification (IC) on CIFAR-10 [49], Keyword Spotting (KWS) [50] and Facial Expression Recognition (FER) [51]. As hardware test-benches, we adopted two commercial boards integrating Cortex-M cores: NUCLEO-F412ZG (M4-256 KB), NUCLEO-F767ZI (M7-512 KB).

The remaining of the section is structured as follows. Section 2.2.1 presents the hardware-aware compression framework adopted to collect the experimental results, with particular emphasis on the memory-driven **PaQ** heuristic. Section 2.2.2 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 2.2.3 summarizes our findings.

2.2.1 Framework Overview

The memory size M_c of a ConvNet depends on the number of trainable parameters N_p (weights and biases) and the arithmetic precision adopted, i.e. the bit-width b. We define the target memory M_t as the available memory that can be used to store and run the ConvNet. Note that M_t can be lower than the total memory size of the hosting device, as part of the memory is dedicated to other routines, e.g. drivers to manage sensors. The original floating-point ConvNet undergoes a compression process in order to generate a memory-bounded ConvNet, i.e. a smaller version with $M_c \leq M_t$. A memory-bounded ConvNet can be defined through a pair $\{N_p, b\}$. Intuitively, for each M_t there exists a set $\mathcal{P}$ of pairs $\{N_p, b\}$ matching M_t. Within $\mathcal{P}$, a pair $\{N_p^{\text{opt}}, b^{\text{opt}}\}$ defines the *optimal* solution, i.e. the pair that minimizes $\mathcal{L}$. A pair $\{N_p', b'\}$ defines the *hardware-compliant* solutions, i.e. the pairs that can be ported on the target device. As mentioned, a pair is hardware-compliant if b' is supported by a proper instruction set, i.e. $b' \in \{8, 16\}$ for the Cortex-M processors adopted as case study. However, $b^{\text{opt}}\}$ might be of any integer value. Therefore, an optimal pair is hardware-compliant if b^{opt} turns

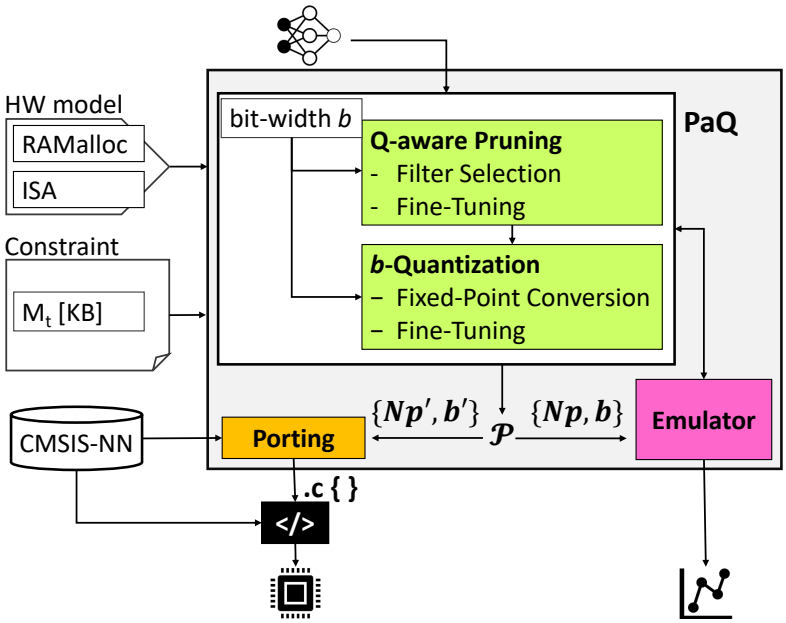


Figure 2.1: Framework overview.

out to be 8- or 16-bit.

The exploration of the memory-accuracy design 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*) measure the accuracy of the returned memory-bounded ConvNets (*iii*) and deploy memory-bounded ConvNets on the target device.

To answer these needs, we developed a novel framework, which is illustrated in Fig. 2.1. The framework takes as input a floating-point ConvNet (FP) trained with standard machine learning platforms (e.g. PyTorch, TensorFlow) and it produces as output (*i*) the accuracy evaluation of the compressed ConvNets that meet the memory constraint (i.e. those with $\{N_p, b\} \in \mathcal{P}$) and (*ii*) the `.C` code of the compressed ConvNets that are hardware-compliant ($\{N_p', b'\} \in \mathcal{P}$). The `.C` code is assembled using a neural network library (the CMSIS-NN by ARM) optimized for inference on MCUs, then compiled and flashed on the target device. The assessment of those compressed ConvNets not compliant with the hardware (i.e. those with $\{N_p, b\} \in \mathcal{P}$, $b \neq b'$) is conducted through an in-house fixed-point emulator. The same emulator drives the fine-tuning stages (more details will be presented later in the text). The core engine of the framework is called Prune and Quantize (**PaQ** hereafter) and it involves two main components: (*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. All the layers of a ConvNet share the same bit-width. This latter aspect affects the compression speed, that is, quantization removes information at a faster pace. Intuitively, removing a single filter on a layer is less intrusive than reducing the bit-width of all the weights of all the layers. Clearly, there exists a circular dependence between pruning and quantization which further complicates the optimization. Specifically, the value of b determines the number of filters that must be removed to match the memory constraint.

To estimate the physical RAM needed by inference, the framework integrates a model file containing the memory allocation strategy of the target architecture (*RAMalloc*). As long as both the HW-model and the neural network library are available, the framework can be extended to any hardware platform.

Memory Model

The amount of on-chip RAM allocated during inference depends on the software implementation of the neural network layers, which in turn is tightly coupled with the hosting hardware. The following text describes the memory allocation policy adopted in the Cortex-M cores through the open-source CMSIS-NN library [45]. The same model can be extended to other architectures and/or libraries. The description refers to the convolution layers, which are the most expensive in terms of memory utilization[2], but the model also includes fully-connected layers.

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 devoted to the *Activation Buffer* (AB), which stores partial results produced by intermediate layers. 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 can be time-shared, therefore 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 guarantee an accurate assessment of the memory requirements, the memory model includes their contribution. Specifically, the CMSIS-NN implements a tensor convolution as a matrix multiplication; this encompasses the conversion of multi-dimensional tensors to bi-dimensional arrays. The matrix, referred to as the Toeplitz matrix [52], is generated by the *im2col* routine, which

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

stores the result in a dedicated region of the RAM, the *im2col* buffer (I2CB). Similarly 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 thus generating two columns of the Toeplitz matrix at a time. This allows reducing the size of the 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 2.1 gives the analytical model for a ConvNet of L layers represented with a physical bit-width b:

$$M_c = b \times \left[N_p + \max_{i \in L}(I_i + O_i) + \max_{i \in L}\left(im2col_i\right) \right] \tag{2.1}$$

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 largest layer. The last term ($\max\left(im2col_i\right)$) 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 of I2CB is usually limited, while WB is dominant. However, for compact ConvNets like those adopted in IoT applications, AB is not negligible (ranging from 15% to 30% the overall RAM).

Q-Aware Pruning

In **PaQ**, filters are dropped until $M_c \leq M_t$. Since M_c depends on the arithmetic precision, the pruning stage should be aware of b to minimize the number of pruned filters. Removing a filter at the i-th layer simultaneously affects several parameters of eq. 2.1: the size of the i-th and i+1-th convolutional layers (hence WB), the size 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 removed. The *importance* metric adopted to determine the filter priority is the ℓ_1-norm of the weights. The ℓ_1-norm is a good proxy to identify those filters with marginal impact on the final output of the ConvNet [37], yet

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
```

ensuring efficient processing, as it does not require the statistics on the activations of the intermediate layers. Overall, it guarantees a good trade-off between quality-of-results and complexity of the optimization loop. However, the framework can work with any kind criteria.

The loop iterates until the memory constraint M_t is met (line 2). The memory estimation leverages the memory model introduced in the previous section (embedded into the *RAMalloc* procedure). Even though 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 the pruning stage proper awareness of quantization.

After pruning, the network may experience a substantial accuracy loss. This loss is however recovered (totally or partially depending on the actual constraint) through *fine-tuning* (line 7). The latter involves a re-training stage (50 epochs in our experiments) during which the weights are tuned using a standard error back-propagation scheme.

B-Quantization

After the Q-aware pruning, the model undergoes the quantization to a b-bit representation. As already motivated in Sec. 2.1, the choice fell upon the most hardware-friendly option: (*i*) symmetric scheme, (*ii*) linear intervals [39], (*iii*) per-layer power-of-two scaling. Adopting a per-layer radix-point scheme brings higher accuracy without performance overhead. The optimal radix-point is found through an iterative optimization, where different positions of the radix-point are tried.

The accuracy drop induced by quantization can be recovered (totally or partially depending on the actual constraints) through a dedicated 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 allowing small weight updates; weights are quantized at the end of every epoch using stochastic rounding.

To emulate fixed-point arithmetic on GP-GPUs, an in-house emulator leverages the fake-quantization method introduced in [53]. It consists of a software wrapper that converts activations and weights (stored in fixed-point) to the 32-bit floating-point; after processing, results are converted back to fixed-point.

Porting and Emulation

After compression, the hardware-compliant ConvNets are translated in `C` code using the neural network library optimized for the target device. We adopted the CMSIS-NN [45] library developed by ARM. The CMSIS-NN offers 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 is possible only for those bit-widths and memory budgets meeting the hardware constraints ($\{N'_p, b'\}$). The framework provides emulation for every bit-width and memory constraint thus 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.

2.2.2 Evaluation of PAQ

We used the proposed **PaQ** flow to run a design space exploration across the memory-accuracy space. This analysis aims to assess the optimality of hardware-compliant implementations and quantify their distance (i.e. the accuracy difference) from theoretical solutions. This section is organized as follows. First, we introduce the ConvNets selected as benchmarks, together with the datasets used for training and evaluation. Second, we describe the hardware boards employed to validate **PaQ**. Third, we report the collected results and discuss the main achievements. Finally, we present further insights to validate **PaQ** and justify the our optimization choices. For the convenience of the reader, in Table 2.2 we define the notations used throughout the text.

Benchmarks, Datasets and Training

For our experiments, we considered three different tasks: Image Classification (IC), Keyword Spotting (KWS), Facial Expression Recognition (FER). All of them find application in many practical use-cases, 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. Table 2.3 reports the topology of the models together with the top-1 classification

Table 2.2: List 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 solutions

accuracy achieved using a floating-point representation (w/o any further optimization). Results are consistent with those available in the recent literature. Both training and testing are run in PyTorch, version 0.4.1. The training is iterated over 150-epochs using the Adam algorithm [54] 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.

Image Classification (IC). For this task, we adopted the popular *CIFAR-10* dataset. It is composed by 32×32 RGB images [49] evenly split in 10 classes, each class with 50000 and 10000 samples for the training set and test set, respectively. Like in [45], the adopted ConvNet is taken from the Caffe framework [55]. It consists of three convolutional layers interleaved with max-pooling and one fully-connected layer.

Keyword Spotting (KWS). This task is a common 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), the task achieves an affordable level of complexity. The reference dataset is the Speech Commands Dataset [50]; 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 10 specific keywords, i.e. "Yes", "No", "Up", "Down","Left", "Right", "On", "Off", "Stop", "Go", out of the 30 available words. Samples that not belonging to the 10 categories are labeled as "unknown". An additional "silence" label is assigned to background noise samples (i.e. pink noise, white noise, and

Table 2.3: 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 [49]	Speech Commands [50]	FER2013 [51]
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%

human-made sounds). The training set and test set include 56196 and 7518 samples, respectively. For the classification, we picked the model called *cnn-trad-fpool3* described in [56]. It is made up of two convolutional layers, two max-pooling layers, and three fully-connected layers. The ConvNet takes as input the Mel-frequency cepstral coefficients of the spectrogram of the recorded signal in [56] (input shape is $time \times frequency = 32 \times 40$ inputs).

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 extremely challenging as many face images might convey multiple emotions. The reference dataset is the *Fer2013* from the Kaggle competition [51]. It collects 32297 48×48 grayscale facial images organized 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 shows nine convolutional layers evenly spaced by three max-pooling layers and one fully-connected layer.

Hardware Set-up and Toolchains

The presented framework (Fig. 2.1) is validated on two commercial boards integrating Cortex-M cores by ARM. As reported in Table 2.4, the selected boards host different chip-sets (M4 and M7), hence they differ in memory capacity (RAM and Flash) and performance (Frequency). The deployment on the target board is powered by the CMSIS-NN software library (v.5.4.0) developed by ARM. The `.C` description of the ConvNet is compiled using the GNU Arm Embedded tool-chain (version 6.3.1.).

Table 2.4: Hardware specification of the boards adopted as test-bench.

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

Within the **PaQ** flow, the prediction accuracy is evaluated through the emulator mentioned in Sec. 2.2.1. The emulator is tailored to replicate the behavior of 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% of accuracy between the outputs produced by the emulator and those collected on-board . The same emulator is adopted for the accuracy assessment of those compressed ConvNets that cannot be ported to the ARM cores, whereas the hardware-compliant ConvNets are evaluated on-board (see the flow depicted in Fig. 2.1). The *RAMalloc* memory model is cross-validated with the results produced by the gcc linker (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[3], version 5.11.0).

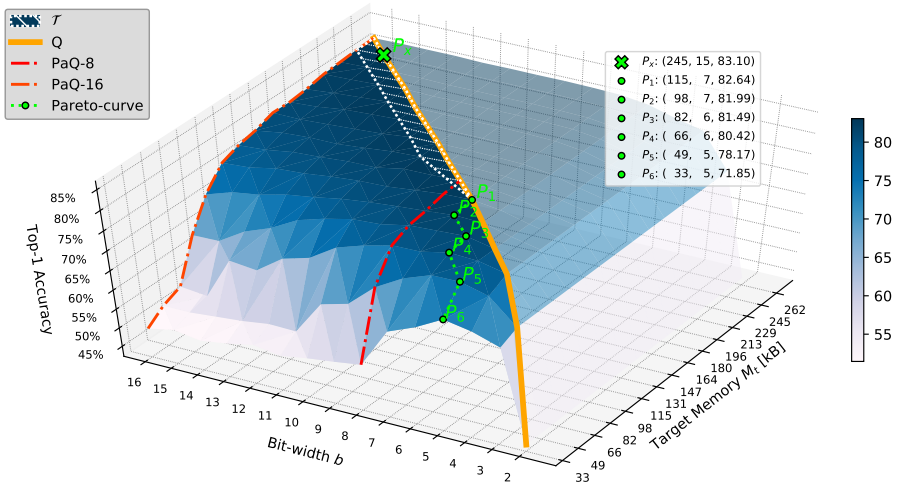
Exploration of the Memory-Accuracy Space

We conducted an extensive exploration 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 denotes 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 will be discussed later). The collected results of the three applications are illustrated in Fig. 2.2a, 2.2b and 2.2c. The plots show the top-1 accuracy for every pair belonging to the design space, i.e. $\{N_p, b\} \in \mathcal{P}$. The yellow line highlights the

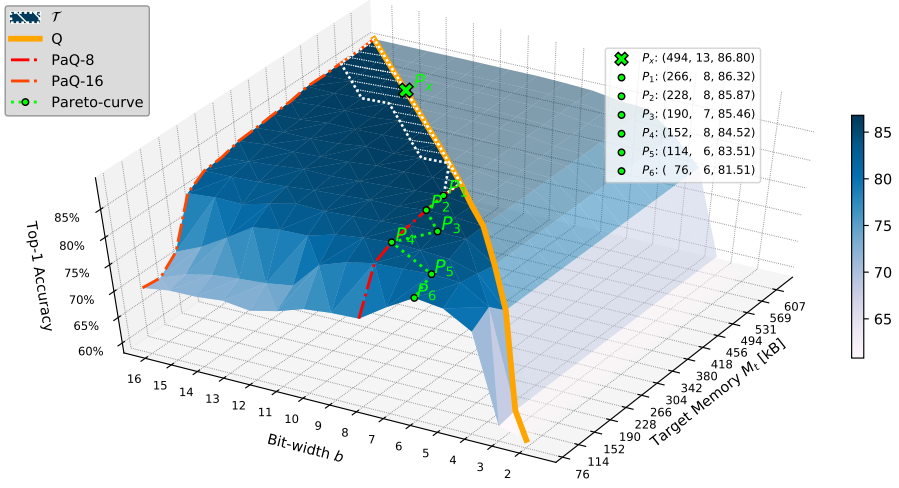
[3]https://os.mbed.com/blog/entry/Tracking-memory-usage-with-Mbed-OS/

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 2.2a 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 the focus of our analysis, which brings the following considerations.

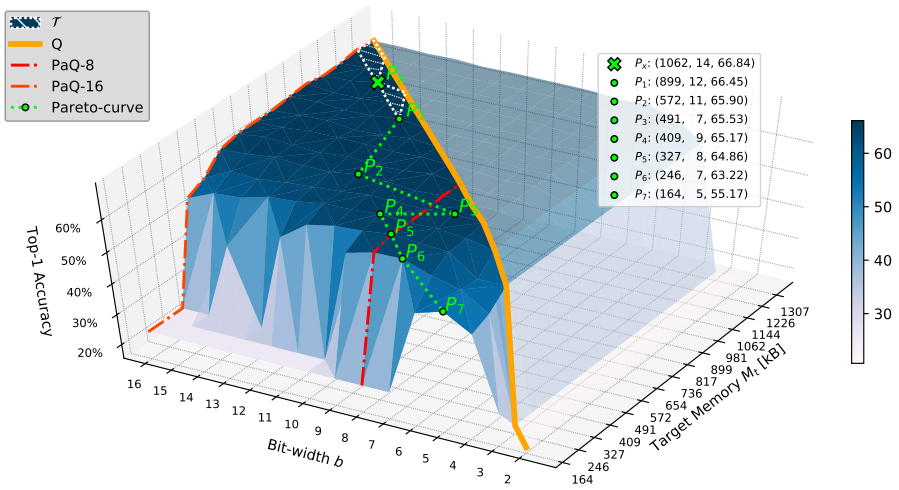
Weakness of accuracy-driven optimizations. We observed that many configurations reach an accuracy very close to that of the FP baseline, namely,



(a) IC



(b) KWS



(c) FER

Figure 2.2: 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) covers 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 applies. The red dash-dotted lines (**PaQ-8** and **PaQ-16**) highlights the solutions generated by **PaQ** using 8- and 16-bit respectively, i.e. the solutions compliant with the target hardware. The green dotted line indicates the Pareto points (P) in the memory-accuracy space, i.e. all the solutions more accurate than all the other points with the same target memory M_t. The right box reports the absolute coordinates of P_x and each Pareto point in the format (target memory, bit-width, top-1 accuracy).

pruning and quantization incur marginal accuracy degradation. These configurations shape a large flat area (hatched in white), referred to as the plateau $\mathcal{T}$. Without loss of generality, we assume that a pair $\{N_p, b\}$ belongs to $\mathcal{T}$ if the accuracy difference compared to the best-accuracy point (P_x, marked with the green cross in the plots and reported in the first row of Table 2.5) is less or equal than 0.5%. The existence of $\mathcal{T}$ is nothing new as significant parameter redundancy exists in ConvNets [33]. The area of $\mathcal{T}$ may depend on the complexity of the task or the network topology.

Most of the existing accuracy-driven compression pipelines, e.g. [29], seek an unique combination of pruning and quantization which maximizes 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 our formulation as $\{N_p, b\} \in \mathcal{T}$ s.t. M_c is minimized. This solution

Table 2.5: Optimal vs. hardware-compliant solutions under different memory constraints M_t: column *Pareto* lists the Pareto points P as for the plots of Fig. 2.2; columns **PaQ-8** and **PaQ-16** refer to solutions genereted by the **PaQ** flow using 8- and 16-bit respectively(red lines in Fig. 2.2); column Δ reports the distance (the lower is better) between optimal and hardware-compliant solutions for both **PaQ-8** and **PaQ-16** in terms of accuracy. Cells corresponding to those solutions 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	-	-

represents the bottom-right corner of the plateau $\mathcal{T}$, denoted with P_1 (second row of each benchmark in Table 2.5).

An accuracy-driven, memory-unconstrained optimization of this kind might return ConvNets not meeting the memory budget of the target device. In FER for instance, the optimal implementation needs 899 KB of RAM using 12-bits, a configuration which is larger than the available physical memory (512 KB in the best case) . Instead, we focus on the region below such theoretic optimum (referred

to as deep memory space). We built our framework specifically to explore this region. 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. Even though this is true, those methods perform better in optimization rather than extensive exploration. Moreover, the multi-objective cost function may result biased by the importance weights adopted.

Memory-accuracy Pareto curve. The green dotted line in the plots connects the configurations in the deep memory space belonging to the Pareto front. As already mentioned, P_1 corresponds to the pair $\{N_p, b\}$ inside $\mathcal{T}$ with minimum memory size. 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 $\mathcal{L} > 0.5\%$ (with respect to P_x). 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 of the Pareto solutions are reported in Table 2.5 together with the top-1 accuracy they achieve. As the numbers suggest, for many configurations the obtained accuracy is still 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.

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 reported in Table 2.5, 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 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 Table 2.5, 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. These findings are in line with previous works and provide additional evidence that 8-bit is more efficient than 16-bit. However, the key insight is that a bit-width below

the 8-bit mark is needed only for extreme constraints. For instance, KWS with memory constraint $M_t = 76\,\text{KB}$, where the PaQ-8 implementation shows $\Delta \geq 1\%$, or FER with memory constraint $M_t = 164\,\text{KB}$, where the PaQ-8 solution shows unacceptable accuracy degradation. Overall, arbitrary bit-widths are needed in few specific corner cases and the need of custom arithmetic units must be assessed carefully.

Validation of PaQ

Efficacy of the proposed memory-driven compression. As described in Algorithm 1, the filter pruning procedure integrated in PaQ is memory-constrained, i.e. the compression ends as soon as the memory constraint is met. To justify this stopping criteria, we performed a comparative analysis with a pruning procedure where the constraint is given in a direct form, i.e. *number of filters to be pruned*. After pruning, the ConvNets 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. 2.3 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. The observed trend motivates our choice: stopping the pruning procedure as soon as the constraint M_t is met ensures the highest accuracy for that specific M_t. The same trend holds for every bit-width used in our experiments (for the sake of readability other bit-widths are not shown in the figure). Furthermore, these findings also validate the linear interpolation adopted to estimate the accuracy when $M_t \in (M_{b_i}, M_{b_{i+1}})$. Indeed, the plot shows that accuracy is a piece-wise linear function of memory. The same considerations hold for the other benchmarks.

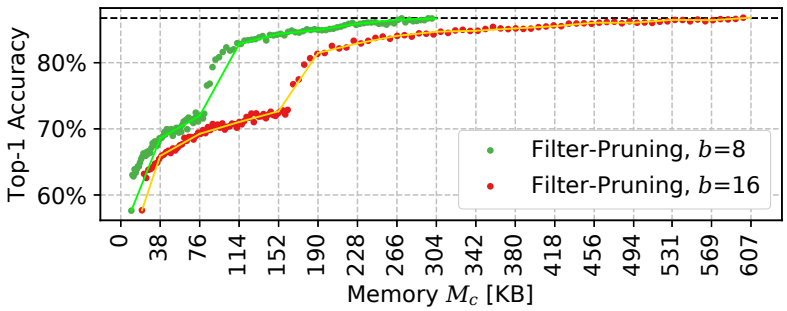


Figure 2.3: Top-1 accuracy vs. memory footprint for KWS.

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. 2.4

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 Table 2.5).

As PaQ-8 outperforms 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. Experiments were conducted on the boards reported in Table 2.4, indicated with the labels NUCLEO-F4 and NUCLEO-F7 for brevity. Since the total RAM of the NUCLEO-F4 board is 256 KB, larger models cannot be ported.

The use of pruning and quantization schemes that preserve the regularity of the ConvNet topology is paramount 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 favorable linearity shown in the plot. The same trend holds for the other benchmarks.

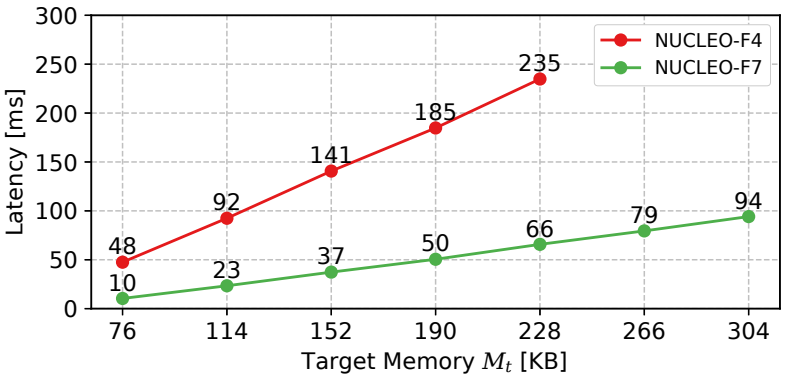


Figure 2.4: Average inference time per sample of PaQ-8 solutions on KWS.

Execution Time. The **PaQ** flow takes a few minutes for each fine-tuning stage (50 epochs each). 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. The execution time can be significantly reduced by limiting the number of retraining epochs without incurring any significant accuracy loss. Early stopping policies may be introduced to serve this purpose as it was done in other works to prevent over-fitting [57] or accelerate the training stage [58]. While the acceleration of the **PaQ** flow is left as future work, Table 2.6 supports our claim showing the number of fine-tuning epochs after which **PaQ** reaches the highest top-1 accuracy. Reported numbers refer to the average over all the pairs $\{M_t, b\}$ of the

Table 2.6: Average number of fine-tuning epochs to achieve the maximum top-1 accuracy on the test set.

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

exploration space. For the three benchmarks, both pruning and quantization converge much earlier than the 50-epoch threshold we set for safety, thereby revealing the optimization room of smart heuristics.

2.2.3 Discussion

We presented a thorough study of memory-bounded ConvNets. The study was conducted through an assessment framework that enables a hardware-aware exploration of the memory-accuracy design space. This represents a distinctive factor with respect to existing optimization flows. Thanks to the proposed framework, we demonstrated that optimal solutions could show similar accuracy than those that can be effectively implemented on general-purpose cores, even at the tight memory constraints posed by tiny MCUs. Collected results on real-life applications revealed that the benefits of custom accelerators are limited to few use-cases and/or extreme compression ratios. When tested on commercial boards, hardware-compliant solutions showed that memory reduction linearly improves performance, thereby demonstrating the efficiency and the scalability of the adopted optimization strategies.

Overall, our study proves that the design space exploration of memory-bounded ConvNets enables to understand when custom arithmetic units are needed. Understanding these cases is paramount to reduce the implementation costs, especially in the context of lightweight IoT applications. Furthermore, the analysis revealed a natural bond between pruning and quantization that can be used in future works to drive and accelerate the optimization process.

2.3 EAST: Encoding-Aware Sparse Training for Deep Memory Compression of ConvNets

As discussed in Section 2.1, several works explored aggressive optimization techniques for the memory compression of ConvNets. In the context of inference on MCUs, quantization via fixed-point arithmetic is a standard today. The use of

8-bit integers is a common choice for general-purpose cores [45] as it squeezes the memory footprint up to 4× related to 32-bit floating-point with no, or marginal, accuracy drop. However, quantization alone might be not enough to meet tight memory constraints. Sparse training via *weight pruning* [59, 60] is an additional strategy that can improve the compression if combined with some encoding scheme. The joint application of sparse training and quantization generates ConvNets prone to be compressed by lossless encoding schemes because the weight arrays contain long sequences of zeros or repeated values.

Intuitively, higher sparsity levels can achieve higher compression rates. However, under extreme constraints, i.e. a few tens of KBs, the level of sparsity which will let encoding schemes match the constraint is extremely high, much higher than what ConvNets may tolerate. Fig. 2.5 illustrates this important aspect for a 9-layer ResNet trained on CIFAR-10. Above 90% of sparsity, the value needed to achieve a memory footprint ≤40 KB, the accuracy curve suddenly drops.

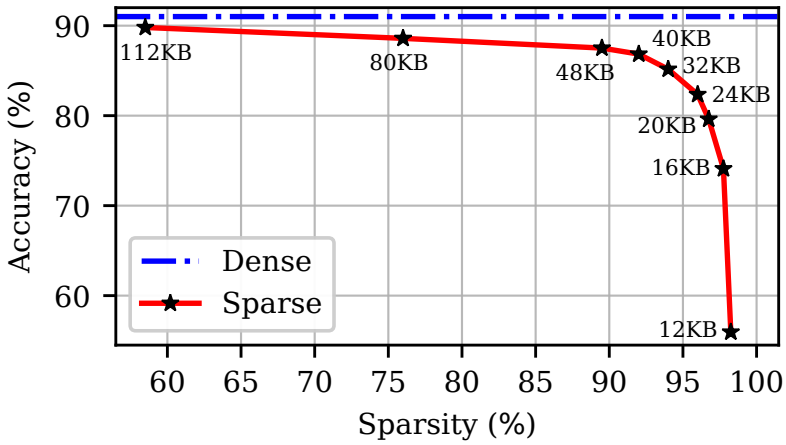


Figure 2.5: Sparsity vs. Accuracy trade-offs of a compressed 9-layer ResNet under different memory constraints (the labeled numbers). The net is trained on CIFAR-10, then compressed via weight pruning and encoding. The blue dash-dotted line marks the accuracy of the original dense version (140 KB).

This poses a novel challenge: *Is it possible to achieve higher compression rates with lower sparsity to preserve accuracy?* The technique presented in this section aims to answer this question, proposing a practical optimization strategy named *EAST* (Encoding-Aware Sparse Training). The underlying principle is simple, yet effective: find groups of adjacent weights to be pruned rather than pruning single connections. EAST involves a sparse training loop based on *adaptive group pruning* where the group size *adapts* to the target memory minimizing the level of sparsity needed. EAST adopts the *LZ4* [61] encoding scheme, which (*i*) can be implemented with an efficient routine of few bytes of memory and (*ii*) guarantees

fast decompression speed and negligible overhead during inference latency. In principle, EAST can work with any existing encoding scheme. The validation is conducted on a state-of-the-art 9-layer ResNet trained on the CIFAR-10 dataset and ported on an Arm Cortex-M4 MCU. A comparative analysis against the same network compressed with a standard weight pruning and encoded with the same LZ4 scheme demonstrates that EAST can reach higher accuracy (up to 8.73%) when the memory budget is very limited (12 KB of flash).

2.3.1 Motivation

Weight pruning is a common technique to generate sparse ConvNets. During training, less important connections are gradually removed until a target level of compression is reached. The weight magnitude is the most adopted proxy to determine the weights priority, under the assumption that collapsing low-value weights to zero has a low impact on the final output of the network. Pioneering works in this field [59] proved that most of the weights can be zeroed (up to 90%), still preserving the prediction quality of the dense baseline. Through a joint application of ultra low-bit quantization (down to 2 bits) and weight encoding, sparse ConvNets may achieve impressive compression ratios, up to 49× depending on the network topology [62]. Furthermore, more recent studies investigated the bond between weight pruning and quantization [29, 63], suggesting optimization strategies to reach the most efficient cooperation between pruning and quantization. However, this class of strategies demands the availability of arithmetic data-paths supporting flexible bit-widths, whereas general-purpose cores come with a limited instruction-set which limits the selection of the bit-width to few options, i.e., 8-bit for the Cortex-M core.

The empirical study conducted in [60] suggested that *large-sparse* models outperforms *small-dense* models. However, above a certain level of sparsity (>90%), the ConvNet may show prohibitive accuracy degradation. We tackle this problem with an encoding-aware pruning that matches the same target memory with fewer weights pruned.

An interesting work conducted in [64] revealed that a ConvNet contains an iso-accuracy sub-network that can be discovered via sparse training. This sub-network may represent the smallest achievable implementation that guarantees the highest accuracy, therefore it could be the best starting point for any encoding scheme. However, how its identification under extremely high levels of sparsity still remains an open problem. We propose a complementary approach, which maximizes the efficiency of the encoding scheme forcing a certain degree of proximity for zeroed weights.

Also, other works experimented pruning at higher-granularity, namely groups of adjacent weights. For instance, in [34] the group size is fixed to reflect the

parallelism of single-instruction multiple-data (SIMD) units. This enables to improve the inference performance. Rather than improving performance, we aim at decreasing the memory footprint. Specifically, we propose a novel version of group pruning aiming to accelerate the compression rate. As a distinctive feature, both the group size and the physical place of the pruned groups are tuned during training depending on the memory constraints. Moreover, our strategy can works also for cores without a SIMD unit (e.g. the low-power Cortex-M0 and M3 MCUs).

Sparse networks can be compressed through different encoding schemes. The most popular examples include *Huffman Coding* [62], *Compressed Sparse Column* [65], *Zero Run Length* [66]. We focus on the *LZ4* algorithm which proved faster than others during decompression (in the order of several GB/s on high-end CPUs). To improve the efficiency of LZ4, and in general of any compression algorithm, it is paramount to have big chunks of adjacent *zeros*. For this specific reason, an efficient compression strategy should take the sparsity distribution (and not just its absolute value) as a first-order variable.

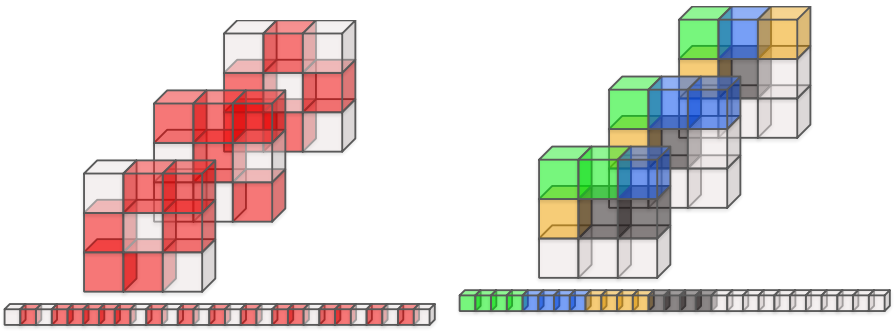
2.3.2 Flow overview

The design of sparse ConvNets involves three main steps: *(i) sparse training*, *(ii) quantization*, and *(iii) encoding*. The first, *Sparse training*, is to train the sparse network under a user-defined memory constraint. Then, *Quantization* reduces the arithmetic precision of the model to a given bit-width (8-bit in our case). Finally, *Encoding* compresses the model size leveraging favorable patterns generated by the sparse training. The EAST strategy implements the first stage.

EAST

Encoding-Aware Pruning. As already mentioned, accuracy-driven weight pruning algorithms return tensors with chains of zeros much longer than in the baseline dense model. Even though this helps to improve the efficiency of the encoding scheme, there is no direct control on the position of the zeros, which is driven by accuracy. The EAST technique relies on the assumption that a weight pruning which is aware of the encoding scheme could make better use of sparsity. This principle is illustrated in Fig. 2.6, which shows how multi-dimensional tensors are transformed into a 1-D array that can be processed by standard general-purpose cores. Fig. 2.6a refers to a standard weight pruning, while Fig. 2.6b is for a group pruning with group size (GS) of 4. In both cases, the picture refers to a *channel-last* layout organization (the same scheme used by the inference library adopted for deployment). The colored items mark the pruned weights. While for the standard method the pruned weights are often placed far away as the selection is just accuracy-driven (smaller weights pruned first), with a group pruning the

proximity of the pruned weights is forced by the size of the group itself. This brings to clear advantages. Indeed, even if both cases show the same sparsity (59%), group pruning gets 55% higher compression ratio (using LZ4). The savings get larger when considering tensors of higher dimensionality.



(a) Weight pruning.

(b) Group pruning ($GS = 4$)

Figure 2.6: Weight Pruning (a) vs Group Pruning (b). Colored weights denotes zero-values

Clearly, the group size serves as a control knob to achieve the best trade-off between accuracy, sparsity, and compression rate. When the available memory is extremely small, groups of larger size may help to reach higher compression with lower sparsity, hence preserving accuracy more. With a too-small group size, e.g. 1 as for standard weight pruning, the amount of sparsity needed by the encoding algorithm to meet the memory constraint would be too large, with negative effects on the accuracy. The EAST strategy implements a memory-driven adaptive group sizing during the sparse training procedure.

Sparse Training. In EAST, both sparsity and group size are gradually increased during the training loop until the memory constraint is met. In the beginning, the sparsity is low and the group size is set to one, hence EAST behaves like a standard weight pruning. If the memory constraint is not satisfied, sparsity and group size are updated following a pre-defined schedule. The sparse training iterates for a new bunch of epochs and if the memory constraint is still not met, sparsity and group size are newly updated. The larger the group size, the faster the memory reduction. Therefore, group pruning helps to converge faster attaining the target memory with a lower sparsity. The group selection is driven by the ℓ_2-norm: groups with lower norm are removed first. However, they can be restored later during the training steps that follow. Once the target memory is met, the sparsity and group size updates are stopped, the pruned weights are frozen, and the training iterates for the last set of epochs adjusting the remaining weights in order to maximize accuracy.

Hyperparamters. Group pruning is applied at the end of each epoch, namely after a complete iteration over the entire training set. The initial target sparsity is 30% with an increased step of 1% every epoch; the step is halved at epochs 20 and 50. The initial group size is set to one; starting from epoch 20, it increases with a step of 1 every 10 epochs.

Quantization & Encoding

After the sparse training, the 32-bit floating-point ConvNet is quantized to 8-bit. The effect of the quantization is (*i*) to reduce the memory footprint ensuring negligible accuracy loss, (*ii*) to increase the frequency of repeated weights, (*iii*) to accelerate the inference time. We adopted a binary-point quantization scheme which is fully compliant with the inference library used for on-board deployment (CMSIS-NN [45]), therefore tailored for the target MCU (the Cortex-M by ARM).

Finally, the quantized model is compressed. EAST is compliant with any encoding algorithm, but we observed that the LZ4 algorithm represents the most suitable choice for resource-constrained MCUs, thanks to its lightweight routine that guarantee a fast decoding process. On-board measurements validated this qualitative analysis. The implemented compression strategy is layer-wise, namely, layers are compressed as separate blocks. This solution allows more efficient management of the available SRAM as it avoids one-shot full model decoding. In fact, layers are processed in sequence during inference, therefore each layer block can be decoded independently and temporarily stored in the SRAM in time-shared buffers.

2.3.3 Experimental Results

Benchmarks, Datasets, and Training

We adopted as benchmark a 9-layer ResNet [67] (ResNet-9) for image classification on the popular CIFAR-10 dataset. ResNet-9 currently holds the first position in the DawnBench Competition [68]. In our implementation, we removed 75% of the filter from each convolutional layer. As it is already optimized for fast training and inference, this ConvNet represents a challenging test-case to assess the efficiency of different compression pipelines.

The dataset is split in training (45K images), validation (5K) and test (10K) set. The model with the highest accuracy on the validation set is selected for evaluation. We adopted a standard data augmentation pipeline, involving padding with random crop, random horizontal flip, and cutout. The same setting is used for both dense and sparse training. The training is driven by SGD for 200 epochs with a batch-size of 128. The learning rate follows a cosine annealing schedule

with an initial value of 0.1. All the experiments have been conducted in Pytorch 1.2.

Concerning quantization, the fixed-point position is determined by a heuristic that minimizes the mean squared error between the floating-point and the 8-bit values. For the intermediate activations, the statistics have been collected on a sub-set of the validation set (size 100 samples).

Table 2.7 reports the top-1 accuracy on the test set and the memory size of the network. The reported values refer to a standard training (i.e. EAST off). Results confirm the efficiency of quantization (column *Q8*) that gets 4× memory reduction with negligible accuracy losses (0.09%) w.r.t. the floating-point ConvNet (column *FP32*). Applying the LZ4 compression to the quantized model does not show significant savings: just a few bytes of memory reduction (column *Q8+LZ4*).

Table 2.7: Top-1 classification accuracy on CIFAR-10 and weight memory of the dense ResNet-9 after 32-bit floating-point training (FP32), after quantization (Q8), and after LZ4 compression (Q8+LZ4).

	FP32	**Q8**	**Q8+LZ4**
Top-1	91.10%	91.01%	91.01%
Memory	558 KB	140 KB	140 KB

Results

EAST opens the deep memory space. Table 2.8 reports the comparison between a standard sparse training via weight pruning (WP) and the proposed flow built upon EAST. The two are compared for different target memories (M_t). The WP is trained using the same sparsity schedule of EAST. For each M_t, the table collects the compression ratio (CR) achieved after quantization and encoding, the sparsity reached after training (columns S_WP and S_EAST), the top-1 accuracy measured on the test-set (A_WP and A_EAST) and the relative accuracy distance (ΔA) between EAST and WP. As demonstrated by previous works, when the memory constraint is met with low sparsity, weight pruning guarantees marginal accuracy losses. For instance, at $M_t = 112\,\mathrm{KB}$ the accuracy loss is only 1.21% lower than the dense 8-bit ConvNet (89.80% vs 91.01%). In this region of memory, EAST reaches similar accuracy levels than weight pruning, 0.34% lower in the worst case ($M_t = 112\,\mathrm{KB}$). However, in the deep memory space ($M_\mathrm{t} \leq 40\,\mathrm{KB}$) weight pruning starts experiencing dramatic accuracy degradation. The reason is that very high sparsity ($> 90\%$) is needed to reach the desired memory constraint, therefore the model loses its expressive power as only a few weights remain up. In this region EAST outperforms WP; the encoding-aware pruning enables better control of the

Table 2.8: Sparsity (S) and Top-1 Accuracy (A) of weight pruning (WP) and EAST on ResNet-9 at different memory constraint M_t (KB). CR is the compression ratio w.r.t. the floating-point ConvNet.

$M_\mathbf{t}$	**CR**	$S_\mathbf{WP}$	$S_\mathbf{EAST}$	$A_\mathbf{WP}$	$A_\mathbf{EAST}$	ΔA
112	5.0×	58.5%	49.5%	89.80%	89.46%	-0.34%
80	7.0×	76.0%	60.5%	88.67%	88.61%	-0.06%
48	11.6×	89.5%	74.8%	87.51%	87.44%	-0.07%
40	14.0×	92.0%	79.0%	86.80%	86.82%	**0.02**%
32	17.4×	94.0%	83.3%	85.30%	86.11%	**0.81**%
24	23.3×	96.0%	87.8%	82.33%	83.65%	**1.32**%
20	27.9×	96.8%	90.0%	79.63%	81.11%	**1.48**%
16	34.9×	97.5%	91.8%	74.16%	78.45%	**4.29**%
12	46.5×	98.3%	94.0%	55.59%	64.32%	**8.73**%

sparsity indeed ($S_\mathrm{EAST} < S_\mathrm{WP}$), preserving the same amount of information within the same amount of memory. On the extreme corner, $M_\mathrm{t} = 12\,\mathrm{KB}$, EAST is 8.73% more accurate than WP due to a lower sparsity (94% vs 98.3%). To emphasize the role of EAST, one can consider that with the same amount of sparsity (e.g. 94%) the model optimized with EAST is 2.7× smaller (row 12 KB vs 32 KB).

EAST accelerates the memory compression. Fig. 2.7 shows the evolution of the memory footprint during the training epochs for both WP (blue line) and EAST (red line) under the same memory constraint $M_\mathrm{t} = 32\,\mathrm{KB}$. During the first 20 epochs, when the group size is one (as set by the training schedule), EAST follows the same trend of WP. Every time the group size gets increased (events indicated with black dots), the memory compression accelerates quickly. As a result, EAST reaches the target memory (indicated with the dashed black line) 43 epochs sooner than WP. These findings suggest that the group size works as an effective knob to boost the compression rate without seeking additional sparsity.

Efficient deployment of sparse ConvNets. We validated the optimization flow on a STM32 NUCLEO-F412ZG [69] board powered with an Arm Cortex-M4 core running at 100 MHz, 1 MB of flash memory, and 512 KB of SRAM. As the inference engine, we adopted the CMSIS-NN library. The original dense ConvNet takes 28 KB of SRAM to store intermediate activations and classifies a single image in 492 ms. The sparse ConvNets needs 884 B of flash for the LZ4 routine, which thereby has a negligible impact on the compression rates achieved. Furthermore, an additional SRAM buffer of 36 KB is needed to store the decompressed weights. Since this buffer is time-shared among different layers, its size is given by the biggest layer. However, the buffer can be dynamically allocated just before the execution of the ConvNet.

The total execution time is function of the memory constraint M_t: the larger

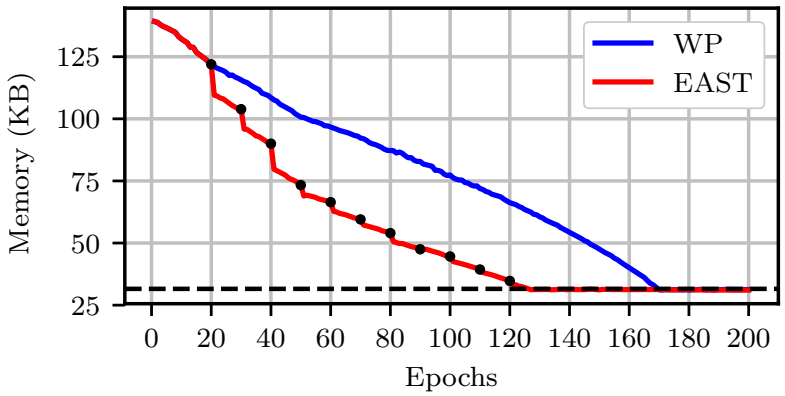


Figure 2.7: Epochs vs. Memory in weight pruning (blue line) and EAST (red line) for $M_t = 32\,\text{KB}$ (dashed line). The dots indicates when the group size increases.

the M_t, the longer the decompression stage. For ResNet-9 generated with EAST, the execution time ranges from 482 ms at $M_t = 12\,\text{KB}$ to 497 ms at $M_t = 112\,\text{KB}$. At the lowest memory, the decompression only accounts for 6 ms; in all cases, the network layers execute faster than the dense counterpart as the weights resides in the SRAM instead of flash.

Discussion

EAST opens new paths towards the optimization of ConvNets in memory-bounded cores. It is particularly suited for the deep memory space, where it outperforms state-of-the-art sparse training. Nevertheless, further investigation is needed to bridge the accuracy gap with dense nets at extreme constraints. First, future studies should explore other proxies than the ℓ_2-norm to drive the group selection. Second, group size and sparsity follow a relative straightforward scheduling during the training; to achieve better trade-offs among sparsity, group size, and the position of the pruned groups, a smarter approach based on automatic hyper-parameter tuning (e.g. Bayesian optimization or reinforcement learning) might help EAST to reach global optima in the design space.

Chapter 3

Energy-Driven Optimization

Energy consumption and execution speed are major sources of concern in embedded ConvNets. They jeopardize scalability and responsiveness both in cloud services, where millions of users generate large pools of concurrent requests with high energy demand, and in mobile applications, where compute and memory resources are limited by low power budgets impeding real-time performance. The latter is of particular interest in the emerging segment of distributed sensing systems where intelligence is decentralized on the mobile edge to preserve data privacy.

The research community is attacking the problem from opposite directions. From the bottom, with the design of custom ICs [70, 9] which show an energy consumption of few pico-Joules/operation. From the top, with new learning strategies [71]. Despite the impressive results achieved, modern ConvNets have an intrinsic limitation that prevents them to raise the bar of efficiency: they are conceived and implemented as static models. There is much more room for improvement instead, and biological systems that inspired the ConvNets may suggest additional dimensions to explore; one of these is *adaptivity*. The human brain operates as a dynamic machine adapting to the surrounding context. The effort that we, humans push to solve a task is not defined a-priory, it is tuned at run-time on the basis of the quality-of-results we are intended to achieve and/or depending on other non-functional metrics, like the criticality and relevance of the problem we are going to solve, our level of tiredness and even the amount of time available to accomplish the task. This happens in many of our daily activities when contextual constraints take precedence over the quality of the final answer. In other words, the brain achieves high efficiency as it pushes effort only when needed. ConvNets are on the opposite corner. Training is a single-objective optimization carried under the worst-case assumption that inference will be always operated at the finest level of detail, no matter the external environment and the actual requests. As our brain does, this conservative view can be relaxed for the sake of efficiency.

The computational effort of a ConvNet, and thus its level of classification accuracy, can be reduced during inference, depending on the context and in favor of less energy consumption or more speed. This useful when the application has to rescale its energy footprint (due to a peak of requests in the cloud or when the mobile system is running out of battery), but also when the classification task is not that critical to solve (in which case some accuracy loss is tolerable) or is not that difficult (in which case fewer resources are needed yet preserving accuracy). We refer to ConvNets with such ability as *Adaptive ConvNets*. Fig. 3.1 provides a generic and abstract view of their implementation. The template includes the hardware hosting the pre-trained ConvNet (a systolic neural accelerator in the example) and the control knob used to pull the system towards different points in the operating space (the 2D energy-accuracy space in the example); the shift depends on an external trigger and it is operated by a control knob during inference, i.e. at run-time, re-configuring the hardware and/or the ConvNet model.

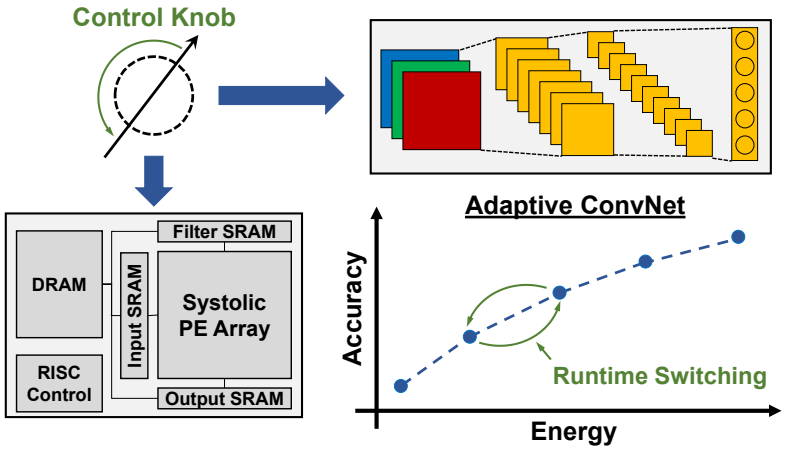


Figure 3.1: Abstract template for Adaptive ConvNets. At run-time, a control-knob changes the configuration of the processing elements (PEs) and of the ConvNet to switch among different operating point in the energy-accuracy space. In this work, the control-knob is precision scaling.

The design of an Adaptive ConvNet encompasses (*i*) the definition of the external trigger, that is an observable variable or a specific constraint forced by the environment, and (*ii*) the actual implementation of the control mechanism. Concerning the first point, previous works made several proposals. The most relevant examples involve the use of the complexity of the input data [72, 73, 74], or the energy budget available [75]. The strategies presented in this chapter deal with the latter class, i.e. energy-driven adaptive ConvNets. About the second point, different kinds of knobs are available, some of which defined at the algorithmic-level, i.e. over the ConvNet topology, like dynamic channel pruning [72, 76] and layer

skipping [73], while others, e.g. precision scaling [44], call for a proper hardware-software co-design. We adopted the precision scaling because it offers unique advantages as it will be discussed later in the text.

Specifically, in this chapter, we present two different strategies for the implementation of a hardware-aware, energy-driven Adaptive ConvNets, both based on a per-layer dynamic precision scaling. While a plethora of hardware solutions can implement energy-proportional multi-precision arithmetic, e.g. [18], our contribution is positioned on the higher levels of the design stack, namely, on the algorithmic implementation of dynamic precision scaling.

The first contribution introduces Adaptive ConvNets via on-line precision scaling. It involves the formulation of an on-line, training-free adaptive mechanism and its optimization methodology, the latter built over a multi-objective search algorithm based on a modified version of the Non-Dominated Sorting Genetic Algorithm II (NSGA-II). The solution returned with the proposed framework consists of a set of Pareto-optimal operating points in the energy-accuracy space. Each point identifies a different precision setting of the ConvNet layers that can be selected at run-time through software-programmable flags.

The second contribution introduces the concept of scalable-effort ConvNets, an effort-accuracy scalable model for classification of data at multilevel abstraction. Scalable-effort ConvNets can adapt at run-time to the complexity of the classification problem, i.e. the level of abstraction defined by the application (or context), and reach a given classification accuracy with minimal computational effort. The abstraction level represents the third dimension in the design-space besides accuracy and energy, thereby enabling more flexible trade-offs than standard ConvNets that always operate at the same level of abstraction.

The presented strategies have two distinctive features related to existing Adaptive ConvNets. First, they make use of a single weight-set rather than multiple models stored on-chip as separate configurations. Second, they can be applied to any pre-trained model, a key advantage compared to state-of-art implementations of Adaptive ConvNets that need dedicated re-training procedures. The remainder of this chapter is organized as follows. Section 3.1 introduces a taxonomy of existing strategies for Adaptive ConvNets. Section 3.2 discusses the background on quantization via fixed-point representations, together with a synthetic review of the most advanced implementations of mixed-precision ConvNets and their optimization. Section 3.3 presents our first implementation based on on-line precision scaling, together with the design, optimization, and implementation details. Section 3.4 deals with the second strategy, i.e. scalable-effort ConvNets, describing its working principle and a practical methodology for its implementation.

The content of this chapter is a revised and extended version of our previous research works conducted in [75, 77, 78].

Table 3.1: Schematic classification of different implementations of Adaptive ConvNets.

Method	**Control-Knob**	**Triggering Condition**	**Optimization**	**Search**	**Training Free**
Runtime Configurable DNNs [76]	Channel Skipping	Energy, Input Complexity	Single-Objective (S)	SGD	✗
Runtime Neural Pruning [79]	Channel Skipping	Input Complexity	Single-Objective (S)	SGD+RL	✗
ConvNet with AIGs [80]	Layer Skipping	Input Complexity	Single-Objective (S)	SGD	✗
SkipNet [73]	Layer Skipping	Input Complexity	Single-Objective (S)	SGD+RL	✗
Dynamic Bit-width Reconfiguration [74]	Precision Scaling	Input Complexity	—	—	✓
Feature Boosting and Suppression [72]	Channel Skipping	Input Complexity	Single-Objective (S)	SGD	✗
Slimmable Neural Networks [81]	Channel Skipping	Energy	Single-Objective (S)	SGD	✗
On-line Precision Scaling	Precision Scaling	Energy	Multi-Objective	NSGA-II	✓
Scalable-Effort ConvNets	Precision Scaling	Abstraction Level	Single-Objective (D)	Greedy	✗

3.1 Taxonomy of Adaptive ConvNets

This section provides the taxonomy of Adaptive ConvNets emphasizing the key aspects differentiating our strategy. A synthetic overview of the existing implementations is provided in Table 3.1, which reports a qualitative comparison based on five different criteria: (*i*) the control knob adopted, (*ii*) the external trigger that drives the adaptive mechanism, (*iii*) the optimization deployed at design-time to implement the control knob and (*iv*) its underlying search algorithm, (*v*) whether the method is training free, namely, it can be applied to pre-trained models w/o additional re-training stages.

3.1.1 Control Knob

The existing control knobs were mainly inspired by two techniques originally conceived as static optimizations: pruning and quantization. The two leverage the over-parametrization of ConvNets, yet acting on different parts of the model. The former, pruning, does reduce the cardinality of the ConvNet by altering the topology of the network, that is, removing those components of the graph that do not contribute (or that weakly contribute) to the classification accuracy. The latter, quantization, decreases the arithmetic precision of the operands (i.e. weights and activations) preserving the overall topology of the graph. Both can be applied to different spatial granularities, e.g. at a net-, layer-, channel-, filter- and weight-level, however, existing methods for Adaptive ConvNets work with coarser grains (i.e. net, layer, and channel) to preserve regularity of execution and scalability of the optimization method.

Pruning-based. Methods based on *channel-skipping* dynamically drop entire channels, i.e. entire 3D convolutional filters; the selection of filters is done at run-time, either driven by a pre-trained agent or scaling the number of filters across all the layers according to a predefined ratio – the width multiplier. Methods based on *layer-skipping* operate in a very similar way, yet at a coarser level bypassing entire convolutional layers.

Quantization-based. ConvNets show high resilience to arithmetic approximation. In fact, their inner tensors can be discretized using a less precise representation, e.g. fixed-point with fewer bits, and then re-trained for a few epochs to restore accuracy. Adaptive ConvNets require that such precision scaling could be dynamically applied at run-time, i.e. on-line during inference, thereby impeding any re-training stage. This aspect is paramount as it may introduce a substantial overhead. A naive solution is that of storing multiple models, each tuned and optimized for a specific precision, and then load one of them at a time depending on the external trigger; clearly a quite inefficient approach in terms of resource usage. Moreover, re-configurable hardware arithmetic units are needed in order to

ensure efficiency. Despite these design issues, dynamic precision scaling has a big plus when compared to pruning-based methods: it can be made training-free. The same is not for channel and layer skipping, which alter the network topology and therefore require to re-train the weights of the original ConvNet from scratch.

3.1.2 External Trigger

Another distinctive characteristic is the type of external variable or constraint that triggers the adaptive mechanism. The options are the input complexity, estimated before or during the inference; the available energy budget, which might enforce some accuracy degradation to achieve better efficiency; the abstraction level, which reflects the semantic organization of concepts in modern datasets.

Input Complexity. Most of the methods reported in Table 3.1 rely on the input complexity. The basic assumption is that "not all inputs are equal," that is, there are input patterns that are easy (hard) to classify because the important features are clearly exposed (masked) [82], hence, they apply an iterative process where the computational effort (managed through the control knob) is progressively increased till consensus is achieved, e.g. increasing the number of active channels or layers, or improving the arithmetic precision. Since most of the inputs show lower complexity, the overall effort is less on average. These methods are data-driven and there is not a direct control because the trigger is embedded into the input itself. As the main consequence, the savings are subject to high variability and in some cases overheads may also appear [74]. This is a key limiting factor for energy-bounded applications that call for real-time resource management.

Energy. A more reliable approach consists of using the actual energy budget as a direct constraint to drive the control knob. Under this class, there are those methods that define a set of ConvNets configurations in the energy-accuracy space. Such configurations, defined at design time with some reduction scheme based on pruning or quantization, are stored on-device and then deployed at runtime depending on the actual needs. The casual component proper of data-driven methods fades, leaving room to determinism and predictability. For such a specific reason, these methods are more suitable for low-power and real-time applications. However, to have multiple models on-board (e.g. one for each energy budget) is not that practical as it may require additional resources, memory in particular. The issue is addressed in this work.

Abstraction Level. ConvNets can be trained and optimized to recognize concepts at different levels of abstraction due to the hierarchical organization of concepts used in modern training sets. As will be demonstrated in Section 3.4, the desired abstraction level might thereby serve as the external trigger that pushes inference towards higher or lower effort. Intuitively, the computational effort is

inversely proportional to the abstraction level: to classify objects with macro-labels (e.g. animals and vehicles) is less complex than recognizing to which sub-label the objects belong (e.g. the kind of vehicles or animals). Leveraging this relationship, a given application is free to define the actual level of abstraction upon the contextual needs, while the control knob operates on the ConvNets set-up in order to resize the resource usage. Like for energy-driven methods, the control mechanism is deterministic but the need for multiple configurations at run-time is a primary source of overheads and should be carefully addressed.

3.1.3 Optimization and search engine

A variety of optimization procedures were proposed in previous works to build Adaptive ConvNets. The commonality between the existing methods is the single-objective nature of the formulation, implemented either by *scalarization* of the cost function, or *discretization* of the optimization space. Our implementation of Adaptive ConvNets via on-line precision scaling differs as we make use of a true multi-objective formulation built on an NSGA-II evolutionary engine.

Scalarization (S). The multi-objective problem is translated into a single-objective formulation. Specifically, the cost function is built as a combination of the original objectives and, if made differentiable, e.g. using linear weighted sums, classical back-propagation methods do apply. This offers the chance to integrate adaptive optimization and weights learning within the same training framework. As reported in the table, most of the prior arts make use of Stochastic Gradient Descent (SGD) and Reinforcement Learning (RL). However, the choice of the coefficients in the cost function represents a major source of sub-optimality. Indeed it is very hard to correctly weigh the objectives, especially for non-linear functions [83]. The authors of [76] introduced an incremental training strategy that starts from a compact network and then it incrementally increases the number of channels of each convolutional layer while keeping already trained weights constant. In [79], the authors proposed the use of RL to train an agent that drives the channel selection during inference. The method described in [80] implements a layer-skipping strategy where the decision of which layer to skip is approximated through a differentiable probability distribution and integrated into a standard SGD training. The proposal in [73] adopts a similar strategy but the selection is driven by RL. The authors of [72] resorted to a predictor block placed at the output of each layer that estimates, at run-time, which channel must be propagated to the next layer and which instead can be blocked; the predictor is trained via SGD. Elaborating on the same idea, the strategy presented in [81] introduces the design of a novel layer—the switchable batch normalization—in which the number of active filters is tuned dynamically by means of a width multiplier.

Discretization (D). The actual optimization is run along one single dimension of the design space representing the free variable to minimize, while the remaining dimensions are handled as constraints. For instance, assuming the 2D energy-accuracy space, the goal becomes to maximize accuracy under a given energy budget, or, to minimize energy within a certain accuracy threshold.

3.1.4 Training

An important aspect is the ability to implement adaptive ConvNets avoiding time demanding training stages. It is well known that to train a new ConvNet architecture is a burdensome task whose efficiency is dictated by the availability of enough computing resources and labeled samples. Since there exist a lot of pre-trained ConvNets which are the result of high optimization efforts, even manual or automatic (via NAS), the possibility of adapting ready-to-use networks is paramount to ensure low design costs and a short turnaround. As reported in the taxonomy table, most existing works do not have such a feature. The only exception can be found in [74], which however does not involve any specific optimization. In this regard, the methodology proposed in Section 3.3 is one of a kind.

3.2 Mixed-precision ConvNets

Our proposals for Adaptive ConvNets are built starting from the ground concept of quantization and mixed-precision. The following text thus provides a synthetic review of the existing static methods, with emphasis on software and hardware solutions.

3.2.1 Fixed-Point Quantization

While training is mostly performed on GPUs using 32-bit floating-point arithmetic, the deployment of trained ConvNets on energy-efficient systems generally encompasses quantization to fixed-point numbers. The two key advantages are the reduction of the model size and better utilization of the memory bandwidth. A good quantization procedure is the one that maximizes the compression – i.e. the bit-width used to store and process the data – with the minimum accuracy loss. Quantized ConvNets have been subject to extensive research in the last years and now can be considered a de-facto standard; a thorough overview of the available strategies is given in [84].

Preliminary works demonstrated that inference with 16- and 8-bit integer arithmetic can reach the same accuracy of 32-bit floating-point just after a short retraining stage [39]. Dedicated training procedures enabled more aggressive bit-width

scaling down to ternary [40] or binary [41] representations, yet with substantial accuracy loss.

A *uniform* fixed-point representation implies all the layers share the same bit-width, whereas in *mixed* representations each layer comes with its own bit-width [44]. The choice may depend on the available hardware and the application. Obviously, mixed approaches are more accurate as layers show different distributions and ranges, but also more complex, as moving across the layers requires extra operations to align the operands.

Regardless of the selected bit-width and its arrangement across layers, several quantization schemes can be adopted with different results. The first distinction is between *linear* and *non-linear* schemes. Within a *linear* scheme, fixed-point numbers are encoded using an affine mapping. Following a general formulation, an integer number Q can be converted into its floating-point approximation F using the following equation:

$$F = S(Q - O) \tag{3.1}$$

where S is the quantization slope and O the quantization offset. If O is unconstrained, i.e. can take any value, the quantization is *asymmetric*; if $O = 0$, then is *symmetric*, as the quantization range is centered around zero. Finally, when S is forced as a power of two, the quantization is *binary*, a well-suited approach for resource-constrained applications because the scaling of the radix-point can be implemented with a straightforward bit-shift. For what concerns *non-linear* schemes, the distance between two adjacent quantized intervals is free to change to better fit the original distribution. The encoding makes use of hash functions such as a *logarithmic* mapping [46] or *clustering* [33]. A non-linear scheme ensures a better approximation, but it requires dedicated look-up tables for an efficient implementation.

To summarize, the quantization process encompasses two stages: (*i*) the choice of the bit-width and (*ii*) the choice of the quantization scheme. A one-fits-all solution does not exist as it strongly depends on the kind of available hardware, the ConvNet model and its weight distribution. In order to implement the energy-accuracy control mechanism in Adaptive ConvNets, we make use of a per-layer mixed-precision linear quantization scheme with symmetric binary scaling.

3.2.2 Training Mixed-Precision ConvNets

Since the intermediate features of a ConvNet may cover uneven ranges and distributions, the optimal solution is to mix precisions across layers. This allows minimizing the average bit-width still preserving the accuracy of the full precision model. The first work proposing mixed-precision ConvNets is [44], where authors proposed a greedy heuristic that decreases the layers precision in topological order,

from input to output. However, when extra-functional metrics like energy are taken into consideration, substantial overheads may appear due to more complex hardware management. The first step toward hardware-conscious optimization was to include indirect proxy signals that can regulate the precision assignment. For instance, [39] describes the design of embedded ConvNets for Field Programmable Gate Arrays (FPGAs) and propose a mixed-precision strategy that is aware of the number of memory accesses. Unfortunately, proxy signals are weak predictors for both latency and energy consumption [85]. Motivated by this observation, more recent works introduced the actual energy consumption as a direct variable in the objective function. Energy can be directly measured [86] or estimated using simulation models [87]. This opened to a multi-objective problem, also mentioned in the taxonomy section, where two conflicting metrics, accuracy loss and energy in this case, should be minimized concurrently. A methodology based on "scalarization" was proposed [88] where the authors formulated the mixed-precision assignment as a differentiable problem solved within the training loop. The authors of [87] resorted to "discretization" with an RL-based agent used to find the optimal bit-width configuration subject to a given energy budget, while in [85] the authors proposed an accuracy-constrained formulation.

All the above methods are for static optimization only and they do not apply for Adaptive ConvNets. Indeed, weights are trained for a specific precision setting without any concern about dynamic scaling at run-time. Their adoption for Adaptive ConvNets would imply the storage of multiple weight sets, namely, one full model for each operating point, which we want to avoid.

3.3 Energy-Driven Adaptive ConvNets via On-line Precision Scaling

This part of the chapter presents our first proposal for Adaptive ConvNets, which is based on on-line precision scaling. The rest of this section is structured as follows. Section 3.3.1 presents the design flow, with emphasis on the proposed precision scaling strategy, the problem formulation, and its optimization via genetic search. Section 3.3.2 shows the actual implementation on a bit-width programmable processing element. Section 3.3.3 introduces the experimental setup and collects the main simulation results, demonstrating the efficiency of the proposed strategy and the effectiveness of the underlying design and optimization engine. As conclusions, Section 3.3.4 gives the final remarks.

3.3.1 Design and Optimization

Understanding Dynamic Precision Scaling

Dynamic precision scaling means to modulate the bit-width of weights and feature maps at inference time, from full precision n-bit ($n < 32$) to low precision m-bit ($m < n$) and backward upon request. Even though the choice of n and m is arbitrary, to consider hardware characteristics is paramount to achieve efficiency [31]. Operands with uneven bit-width (e.g. 12, 11, 10 or 7, 6, 5) may be a costly design option due to sub-optimal memory allocation and additional latency to feed the execution units in a regular manner. Moreover, custom PEs supporting arbitrary-precision arithmetic get larger in the area and hence more power-consuming. A common practice is to restrict n and m to a power-of-two on the basis of the hardware parallelism. In this work, we adopted an accelerator with 8-bit memory banks (both DRAM and SRAM) and a lightweight double-precision arithmetic unit supporting 16- and 8-bit MAC operations. Sub-multiples are still possible, e.g. n=8 and m=4, as long proper hardware is available.

In its simpler version, precision scaling is implemented by means of weight truncation, that is, only the m most significant bits of the weights are retrieved from memory and processed. When applied with a per-net granularity, i.e. all the layers scaled to m-bit, ConvNets experience dramatic accuracy loss. A quantitative comparison is reported in Table 3.2 which shows the top-1 accuracy for the five ConvNets we used as benchmark in this work; column **FP32** is for the original floating-point model, column **FX16** refers to full-precision fixed-point models (i.e. n-bit) quantized through a static method (without any re-training stage), column **FX8-T** is for the dynamic scaling to low-precision (i.e. m-bit) obtained truncating the weights of the **FX16** model. While the full precision quantization can retain accuracy (the worst-case degradation from **FP32** to **FX16** is 0.014%), precision lowering via truncation causes a sudden drop (>99% from **FP32** to **FX8-T**, which makes the models useless. An off-line re-training to fix the loss of **FX8-T** models has no meaning here as it would imply to have multiple fine-tuned models stored on-chip to make the system adaptive, one for each precision. The idea of on-line re-training could be feasible in principle, provided that there are enough compute resources and time to handle back-propagation and the availability of training data, which is unlikely to happen. Instead, the proposed dynamic approach is training-free and can be performed at run-time with limited overhead. Specifically, we propose to generate low precision weights using a half-even rounding of the full precision model. Intuitively, the rounding to the nearest neighbor reduces the arithmetic error minimizing the overall accuracy drop. The results collected in column **FX8-R** of Table 3.2 give experimental evidence: for the first four benchmarks (SqueezeNet and ResNets) the top-1 accuracy raises substantially, while for

the last one (MobileNet) the recovery is negligible and not enough to make the ConvNet of any practical use. This preliminary analysis suggests that the rounding scheme is a good candidate, but since the gap from full precision is still large ($\geq$10% even in the best case) it needs more elaboration.

Table 3.2: Top-1 classification accuracy of five state-of-the-art ConvNets on the ImageNet validation-set at different precision: 32-bit floating-point (FP32), 16-bit fixed point (FX16), 8-bit fixed-point with truncation (FX8-T), 8-bit fixed-point with rounding (FX-R).

Benchmark	**FP32**	**FX16**	**FX8-T**	**FX8-R**
SqueezeNet v1.1	56.360	56.348	0.872	43.796
ResNet18 v1	69.128	69.126	0.100	61.852
ResNet34 v1	72.688	72.688	0.104	63.826
ResNet50 v1	74.098	74.084	0.100	64.774
MobileNet v2	69.984	69.978	0.074	0.100

A finer per-layer precision scaling will help to reach higher quality. It is well known, in fact, that layers have a distribution of weights that changes with the depth of the layer itself and hence there are layers tolerating arithmetic errors less/more than others. This opens to an optimization problem: which precision to apply to which layer in order to reach minimum accuracy drop and high energy savings.

Framework Overview

As shown in Fig. 3.2, the design flow consists of two main stages, *quantization* and *Pareto search*. During quantization, a ConvNet model trained with 32-bit floating-point numbers is shifted to n-bit fixed-point representation ($n < 32$, 16 in this work) using a uniform quantization scheme w/o re-training. This serves as the full precision reference for dynamic precision scaling, and hence it is the model actually stored on-chip. During the Pareto search, the space of mixed-precision solutions is explored to find those configurations that maximize the energy savings ensuring minimum accuracy degradation. The exploration is driven by a multi-objective optimization algorithm that returns the Pareto-optimal points. Each Pareto-point has a precision setting for the L layers, full precision (i.e. n-bit) or low precision (i.e. m-bit), that makes the ConvNet working on that specific energy-accuracy balance. The number and values of the Pareto points are not defined *a-priori*, is the result of the searching phase that identifies the optimal configurations, and may therefore change depending on the ConvNet topology.

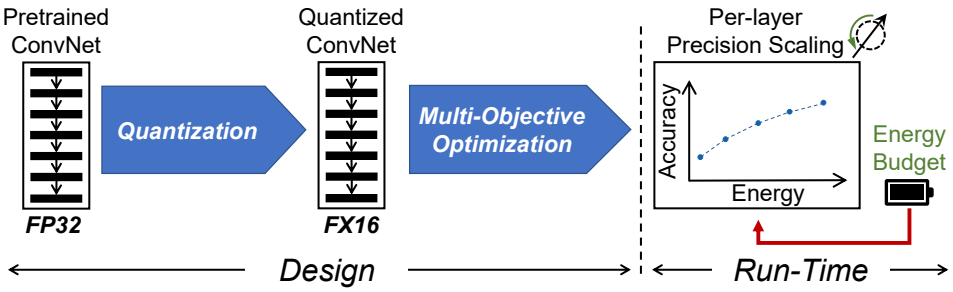


Figure 3.2: Design-Flow Overview

At run-time, a change of the energy budget triggers the adaptive mechanism which seeks the most accurate precision setting that satisfies the energy constraint and implements the per-layer precision scaling via on-line weight rounding.

Quantization

We adopted a linear quantization based on symmetric binary scaling. To match the different distributions across the network layers, we resorted to a variable fixed-point strategy where the position of the radix b is defined layer by layer. A greedy heuristic searches the fraction length that minimizes the mean squared error between the original 32-bit floating-point numbers X and the quantized n-bit numbers Q. The same procedure is applied to both intermediate activations and weights independently. For collecting the statistics of the activations, we used a subset of the training set referred to as *calibration set*. All the quantized values are computed using Equation 3.2:

$$Q = \text{clip}(\text{round}(X \cdot 2^b), -2^{n-1}, 2^{n-1} - 1) \tag{3.2}$$

where *clip* denotes the clipping functions to limit Q in the quantization range and *round* stands for half-even rounding.

For the assessment of accuracy, we built a custom version of the *fake-quantization* strategy introduced in [89], which enables the processing of quantized ConvNets on common GPUs. During inference, a software wrapper converts intermediate features and weights (stored in fixed-point) to the 32-bit floating-point; after being processed, results are converted back to fixed-point. The results previously reported in Table 3.2 validate our quantization strategy showing that the accuracy degradation from **FP32** to **FX16** is negligible.

Pareto Search

The procedure, graphically depicted in Fig. 3.3, aims at finding the set of points $\mathcal{F}$ that are Pareto. A point is Pareto if there is no other point with at

least one inferior objective and all other objectives inferior or equal. To search a Pareto point encompasses the concurrent optimization of conflicting objectives, classification loss and energy consumption in this case, within a region of interest which is limited below acceptable levels of accuracy loss (a few percents) in order to preserve the expressive power of the original ConvNet.

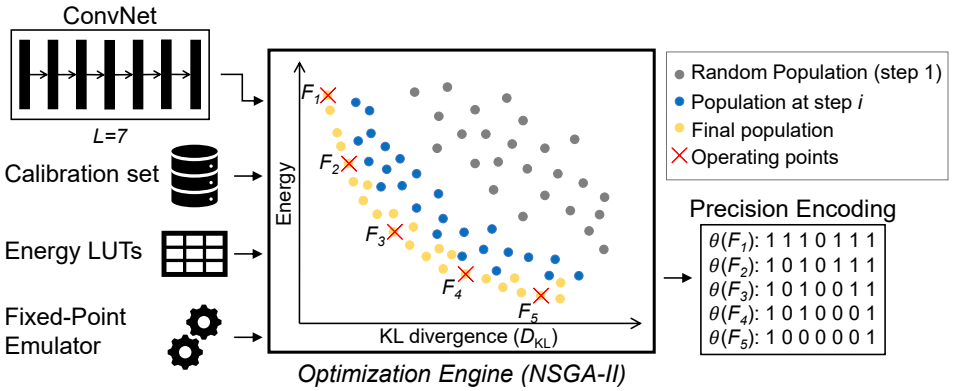


Figure 3.3: Schematic view of design-time optimization flow.

The optimization engine is fed with the full precision ConvNet obtained after quantization and it returns the optimal mixed-precision configurations. Other inputs are the calibration set for assessing the accuracy of the explored configurations (task supported by a fixed-point emulator working in background) and the energy look-up tables (LUTs) containing a pre-characterization of the energy consumption of the ConvNet. The example in the figure refers to a 7-layer ConvNets ($L = 7$) and produces five different configurations (red crosses) which can be stored into the device as a LUT. Our optimization makes use of a modified version of the NSGA-II algorithm [90], an evolutionary heuristic based on the notion of Pareto dominance. Starting from a random population (the grey dots in the figure), a set of genetic perturbations are applied to generate new individuals. Those who dominate the remainder of the population are picked to form the evolved population (blue dots). After a predefined number of iterations, the final population (yellow dots) is used as a pool from which we extract a set of Pareto optimal configurations $\mathcal{F}$ (red crosses) spread across the optimization space. As a key feature, the proposed NSGA-II* algorithm is optimized to point the search towards the high-accuracy region. Indeed, Pareto points with low accuracy are not of practical use, regardless of the achievable energy reduction.

In the following text, we details the multi-objective problem formulation and the NSGA-II* algorithm used as the solver.

1) Problem Formulation: Given a set $\mathcal{V}$ of layers (both convolutional and fully-connected are considered) with cardinality $|\mathcal{V}| = L$, find the finite set $\mathcal{F}$

of mixed-precision configurations that dominate the energy-accuracy space. Each item in $\mathcal{F}$ represents an integer labeling of the layers $\varphi : \mathcal{V} \rightarrow \{0,1\}^L$. The unknown θ is an array of L binary decision variables encoded as follows: $\theta_i = 1$ if the i-th layer is assigned to full precision (i.e. n-bit); $\theta_i = 0$ if the i-th layer is assigned to low-precision (i.e. m-bit). The cost functions of the multi-objective optimization are $\mathcal{L}_1$, as the accuracy loss, and $\mathcal{L}_2$, as the energy consumption, whose description is given in the next two paragraphs.

(a) Accuracy loss ($\mathcal{L}_1$): the accuracy loss due to a given mixed-precision configuration θ is expressed through an indirect metric, the Kullback-Leiber (KL) divergence (D_{KL}). It is a measure of distance between two probability distributions in the interval $[0, \mathrm{inf})$, where 0 means no distance, i.e. equality. Here the two probability distributions refer to the probability distribution sampled at the output of the ConvNet when it is set to full precision (pd_{full}) and when it is set with a mixed-precision configuration θ of interest (pd_{mixed}). The objective is therefore:

$$\min_{\theta} D_{\mathrm{KL}}(pd_{\mathrm{full}} \mid\mid pd^{\theta}_{\mathrm{mixed}}) \tag{3.3}$$

with pd_{full} and pd_{mixed} generated collecting the statistics of the output probability p_k for all the classes K over the calibration set, and p_k as the softmax of the logit z_k:

$$p_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)} \tag{3.4}$$

We experimentally proved the KL divergence is a good proxy for the assessment of accuracy loss introduced by the precision scaling: the lower it is, the lower the accuracy loss. Moreover, it does not need labeled data.

(b) Energy consumption ($\mathcal{L}_2$): the estimation of the energy consumption E during an inference run for a given mixed-precision configuration θ is based on a fast, yet accurate static model built upon off-line characterizations of the ConvNet layers. More specifically, E is given as the sum of the energy consumed by each layer V_i as follows:

$$E(\theta) = \sum_{i=1}^{L} E_i(\theta_i) \tag{3.5}$$

where the values for E_i are retrieved from the energy LUT (given as input to the optimization engine) collecting the energy consumption of each layer i under full precision, i.e. $E_i(\theta_i = 1)$, and low precision, i.e. $E_i(\theta_i = 0)$; the two are measured using a cycle accurate simulation engine (more details are provided in the experimental section). The objective is therefore given as:

$$\min_{\theta} E(\theta) \tag{3.6}$$

Algorithm 2: NSGA-II* pseudo-code

Input: ConvNet, N, maxeval, calset, P_{c}, P_{m}, KL_{th}
Output: Pareto Front F

1 Parent Solutions = Random generate N solutions
2 Evaluate D_{KL} and E on Parent
3 Assign penalty to Parent Solutions
4 **while** *Evaluations* < *maxeval* **do**
5 Binary Tournament Selection
6 Generate Offspring Solutions
7 Simplex Crossover with Probability P_{c}
8 Bitflip Mutation with Probability P_{m}
9 Evaluate D_{KL} and E on Offspring Solutions
10 Assign penalty on Offspring Solutions with $D_{\mathrm{KL}} > KL_{\mathrm{th}}$
11 F = Population Update
12 $F = \varepsilon$-nondominated sorting on F
13 **return** F

2) Search Engine The pseudo-code reported in Algorithm 2 describes the steps of the proposed NSGA-II*. The initial population is made up of N solutions θ (also called individuals) randomly generated (line 1), i.e. probability of $\theta_i = 1$ is 50%. For each solution θ the two cost functions are evaluated (line 2), namely D_{KL} (measured over the calibration set *calset*) and E. In order to push the search towards regions with the highest accuracy, we assign a penalty score to those solutions with high D_{KL} (line 3) such that individuals with a high penalty can be dropped regardless of their energy. Specifically, if a solution has $D_{\mathrm{KL}} > KL_{\mathrm{th}}$, its penalty score is D_{KL}, zero otherwise. A change of the threshold KL_{th} has the effect of limiting the maximum accuracy loss accepted. The new set of individuals (the offspring) is generated making parents evolve through standard genetic functions: (i) binary tournament selection using ranking and crowding distance (line 5) to select those individuals that will produce offspring from the remainder of the current population; simplex crossover with probability P_{c}; bitflip mutation with probability P_{m} (lines 6-8). As soon as the offspring is ready, the objective metrics and the penalty score are evaluated and annotated (lines 9-10), and the parent population is updated using non-dominated sorting and crowding distance selection as described in [90]. Among the new parent solutions, the dominant ones are picked and collected within $\mathcal{F}$. Domination is first assessed comparing the penalty scores, and then through the concept of Pareto dominance with respect to the two objective functions (line 11). The loop iterates for *maxeval* trials, after which the final population is ready and the subset of Pareto points is sampled using ε-nondominated sorting [91] (line 12). The key principle underlying

ε-dominance is that in a bounding-box of width ε_1 and height ε_2 only one Pareto point is admitted. This kind of sorting guarantees that solutions are spread over the energy-accuracy space. Finally, the set of selected operating points is returned (line 13).

In terms of complexity, the bottleneck is the evaluation of D_{KL}, as it requires a feed-forward execution of the network. The execution time depends on the complexity of the network and the size of the calibration set.

Even though our formulation is tuned for two precision options only (full:m and low:n), the extension to multiple bit-widths is straightforward and can serve as support for specialized processing units, e.g. [21]. The precision assignment can be defined as an integer problem indeed, where θ_i is an integer number in [0, $B-1$], where B enumerates all the bit-widths available in hardware. Concerning the algorithm, it is just needed to replace the bitflip mutation (line 8) with a polynomial mutation.

3.3.2 Runtime implementation

PE-reconfiguration

Fig. 3.4 shows an abstract view of a reconfigurable processing element (PE). The left (right) picture illustrates the execution flow at 16-bit (8-bit). When operating at a lower bit-width, the multi-precision MAC unit doubles the number of operations delivered each cycle and the timing critical path (red dashed arrow) is shortened. The available timing slack can be exploited by Dynamic Voltage Frequency Scaling to reduce power consumption while keeping the throughput constant. As shown in Fig. 3.4b, an 8-bit representation also improves memory utilization with fewer fetch operations and higher data reuse.

Weight-reconfiguration

The efficient execution of a low-precision layer requires that 8-bit weights reside in adjacent memory locations to enable parallel memory fetches. Based on this observation, the shift from full to low-precision requires a re-configuration process by which weights are properly organized into a new workspace to ensure affine access profiles during inference, as depicted in Fig. 3.5. The figure illustrates the two precision scaling options discussed in the previous section, truncation (3.5a) and rounding (3.5b), respectively. The process is managed by the RISC control unit. In the left case, the least significant byte (LSB) is discarded and only the most significant byte (MSB) is kept; truncated weights are then stored in adjacent locations of the DRAM. In the right case, half-even rounding is applied to reduce a 16-bit word to 8-bit. Our final implementation adopts the second option,

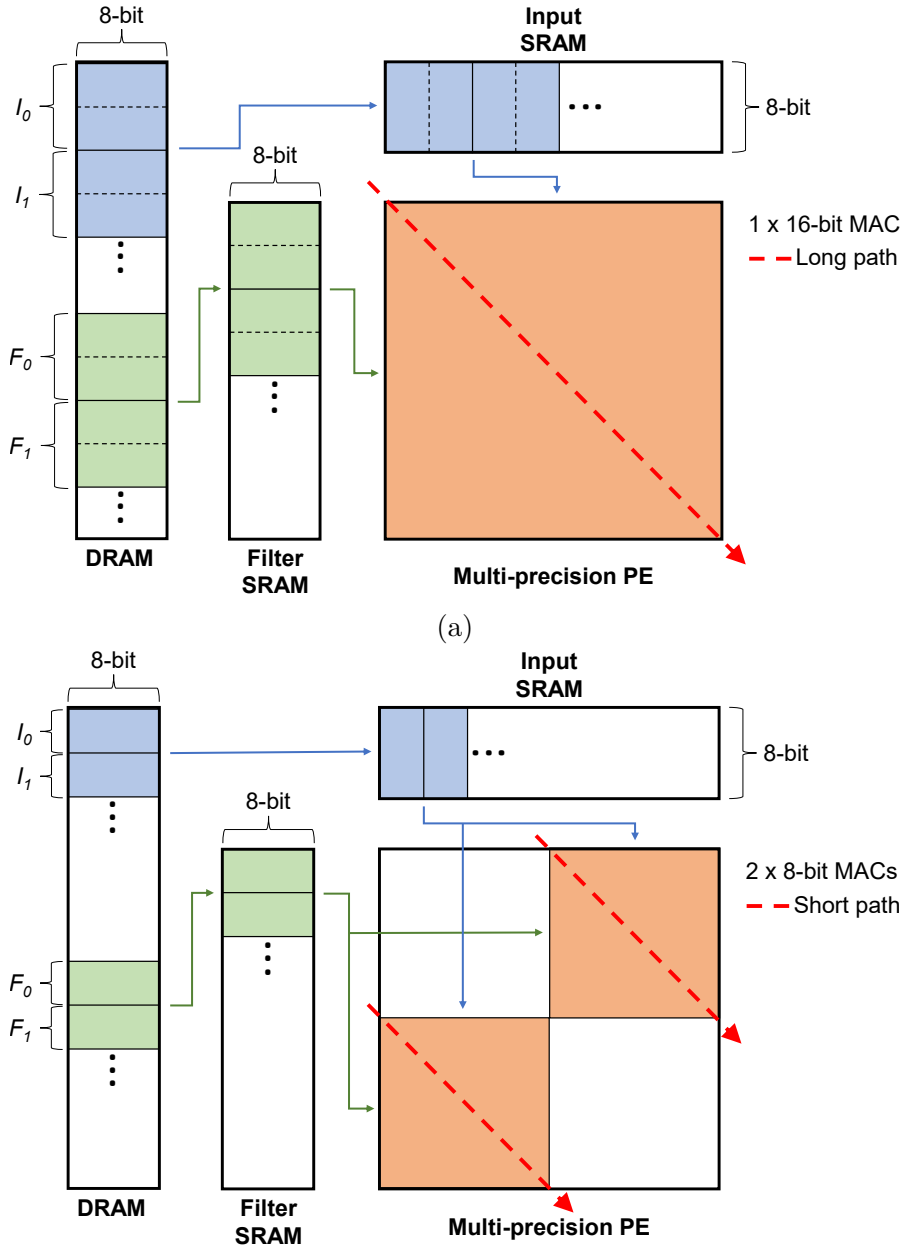


Figure 3.4: Abstract execution flow and PE configuration at 16-bit (3.4a) and 8-bit (3.4b).

i.e. half-even rounding, since it guarantees better energy-accuracy trade-offs, as demonstrated in the experimental section.

Extra memory space is needed to store the new scaled weights while keeping the

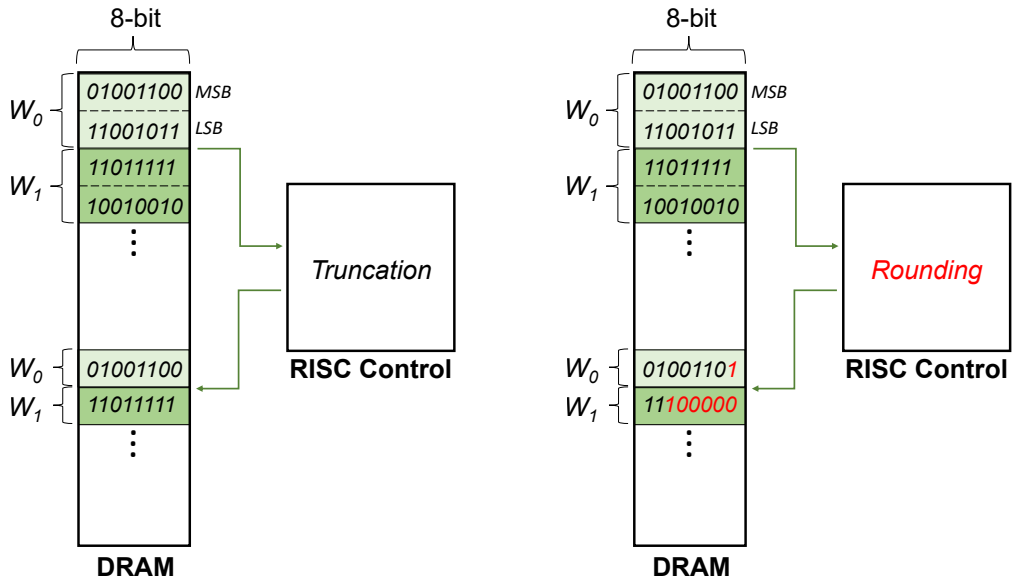


Figure 3.5: Weight (W_i) re-configuration with truncation (3.5a) and rounding (3.5b).

original full precision weights. However, the additional memory can be dynamically allocated and de-allocated when needed. As it will be detailed later, simulation results reveal the whole process introduces a limited overhead, with a negligible impact on the total energy consumption.

3.3.3 Experimental Results

The experimental section is organized as follows. We first describe the ConvNets adopted to benchmark the proposed solution. Then, we present the simulation model adopted to pre-characterize the energy LUT which are then used by the search algorithm for the assessment of Pareto optimal solutions. Next, we give additional details on the experimental setup. We therefore present the collected results with the aim to (*i*) demonstrate the efficiency of the proposed optimization, (*ii*) assess the solutions in the energy-accuracy space quantifying the savings brought by Adaptive ConvNets, (*iii*) compare the proposed single-weight solution against a naive multiple-weight adaptive strategy based on topology restructuring. Moreover, we validate the proposed algorithm, quantifying the contribution of its main components, i.e. weight re-configuration with rounding and the penalty score.

Benchmarks and Datasets

We applied and tested the proposed adaptive strategy on five state-of-the-art ConvNets[1] for image classification trained on the ImageNet (ILSVRC2012) dataset [93]. As reported in Table 3.3, the suite includes three different versions of the ResNet architecture [94] and two networks designed for embedded devices, namely SqueezeNet [95] and MobileNet [96]. The top-1 classification accuracy (*Top-1 Acc.* in Table 2.7) is measured on the ImageNet validation-set. The column *#Params* shows the number of weights for convolutional and fully-connected layers. *#Cycles* is the total number of cycles needed for inference, measured by a cycle-accurate simulator (see the next paragraph). *#Layers* (L) indicates the number of convolutional and fully-connected layers; it defines the size of the search space, which is equal to 2^L. As one can observe, the number of possible precision configurations ranges from 2.1×10^6 (for ResNet18) to 1.8×10^{16} (for ResNet50 and MobileNet), highlighting the need of an heuristic optimization.

Table 3.3: Benchmark ConvNets overview. Top-1 accuracy refers to the 32-bit floating-point model.

Benchmark	Top-1 Acc.	#Params	#Cycles	#Layers
ResNet18 v1	69.13%	11.68M	29.35M	21
ResNet34 v1	72.69%	21.78M	57.28M	37
ResNet50 v1	74.10%	25.50M	74.40M	54
SqueezeNet v1.1	56.36%	1.23M	6.45M	32
MobileNet v2	69.98%	3.47M	12.13M	54

Building the Energy Model

The Pareto search encompasses the assessment of the energy consumption under different precision settings. To accelerate the exploration, we adopted a static energy model where the total energy spent during a forward pass of the net is given as the sum of the contribution of each layer. The energy drained by each layer is characterized for the two precision options available (16- and 8-bit) and then stored in the energy look-up table which is fed as input to the search algorithm.

As a case study, we took a ConvNet accelerator based on a 2D systolic array as abstracted in Fig. 3.1. The system is organized as follows: the DRAM and SRAM memory arrays store the model parameters and the intermediate features; the

[1] Available on the ONNX model zoo [92]

Table 3.4: Hardware configuration of the adopted ConvNet accelerator.

#PE	SRAM size	DRAM size	Technology
8x8	3 x 128 KB	256 MB	40 nm CMOS

RISC core is in charge of managing the control flow; the PEs host the arithmetic workload. The main specifications are reported in Table 3.4. There is a global 256 MB DRAM and three 128 KB SRAM memories that offer support for the local storage of the input features map, weights, and output features map. The power characterization of the memory banks was performed with CACTI [97] for the DRAM and CACTI-P [98] for the SRAMs. Concerning the other components (the RISC core and the PE array), we borrowed the values reported in [18]. Each PE can deliver 1 $\times$ 16-bit MAC or 2 $\times$ 8-bit MACs, combining DVFS for power a higher energy efficiency. All the collected power metrics were integrated into SCALE-sim [99], an open-source, cycle-accurate ConvNet simulator for systolic accelerators. It takes as input the network topology and a high-level description of the hardware, and it returns the memory traces and the computation cycles needed for inference. In the experiments, we considered an output stationary flow. Knowing the power consumption of each component and its utilization, we computed and annotated the total energy spent by each layer.

Experimental Setup

Table 3.5 summarizes the hyper-parameters of our NSGA-II* implementation with their values used for the experiments. The calibration set includes one hundred samples randomly picked from the Imagenet training set. Quantization and multi-objective optimization share the same calibration set. As already mentioned, from the N solutions obtained by NSGA-II*, we extract a sparse set of operating-points using ε-nondominated sorting; the KL divergence and energy resolutions—ε_1 and ε_2 respectively—are set to 10% of the overall range of the corresponding metrics. The top-1 accuracy of the final solutions is evaluated on the ImageNet validation set, which is fully disjoint from the calibration set. The experiments were conducted on a GP-GPU workstation powered with a Titan GTX-1080 Ti by NVIDIA. As inference engine, we used PyTorch [100] version 1.0.

Results

Efficiency of the proposed algorithm. Fig. 3.6 shows the Pareto front obtained by Algorithm 2 for the selected benchmarks. The dot (•) denotes the full-precision model, where all the layers operate at 16-bit precision, while crosses

Table 3.5: Optimization parameters NSGA-II*

Notation	Definition	Value
$\boldsymbol{N}$	Population Size	100
maxeval	Maximum number of evaluations	2000
calset	Calibration set size	100
$\boldsymbol{P}_{\mathrm{c}}$	Crossover probability	1
$\boldsymbol{P}_{\mathrm{m}}$	Mutation probability	$1/L^*$
$\boldsymbol{KL}_{\mathrm{th}}$	KL divergence threshold	0.5
$\boldsymbol{\varepsilon_1}$	KL divergence resolution	10% ΔD_{KL}
$\boldsymbol{\varepsilon_2}$	Energy resolution	10% ΔE

* L: number of convolutional and fully-connected layers

$\times$ indicate the mixed-precision configurations returned by Algorithm 2. This illustration is insightful and allows us to infer three key observations. First, all the solutions are Pareto-optimal in the energy-accuracy space. This finding demonstrates that D_{KL} represents a good proxy for output quality. Second, the accuracy loss is constrained within reasonable values: <3% for SequezeNet; <4% for ResNet50; < 9% for MobileNet (the worst-case). The penalty score drives the exploration towards high accuracy regions, as demonstrated by further analysis reported later in Sec. 3.3.3. Third, the selected operating points are spread in the energy-accuracy space, enabling an effective trade-off. For instance, in ResNet18 the energy savings range from 18.0% to 35.2%. A slightly different trend holds for MobileNet where the Pareto points are less spread and the maximum savings get lower (21.77%) than in the other benchmarks. MobileNet presents a peculiar topology interleaving depthwise and pointwise convolutions to reduce the number of operations. As a drawback, the weights of the depthwise convolutions may show different ranges across the filters of the same layer [101] thereby preventing the adoption of a low-precision representation.

Table 3.6 summarizes the figures of merit of the Pareto fronts. For each benchmark, it shows the number of operating points (*# Points*), the average top-1 accuracy difference with respect to full-precision (Δ *Top-1*) and the average energy savings (*Savings*); the average is calculated over the Pareto points available. For all the ConvNets under analysis, we identified 5-6 operating points with average savings ranging from 19.2% to 29.8%. This allows modulating at run-time computational effort and accuracy, depending on the available energy budget or the application constraints. The last column (*Ex. Time*) reports the execution time taken by the Pareto search. It is limited to a few minutes (worst case is ResNet50 with about 25 minutes). As anticipated, the main bottleneck is the evaluation of

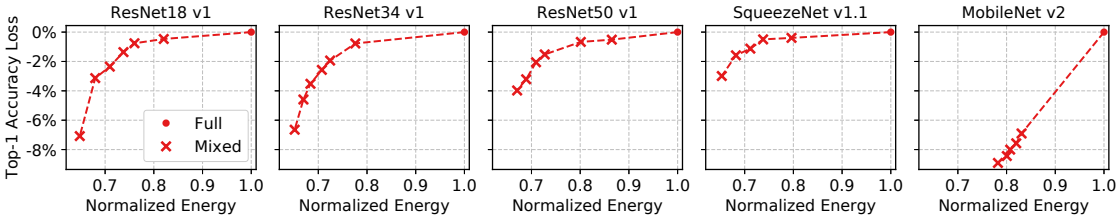


Figure 3.6: Pareto fronts of the selected benchmark. The x-axis is the normalized energy with respect to the full-precision ConvNet (all layers at FX16). The y-axis is the Top-1 Accuracy loss with respect to the full-precision ConvNet. The crosses (×) indicate the operating points returned by Algorithm 2. The dot (•) denotes the full-precision ConvNet.

the KL divergence, which requires a feed-forward execution of the net. A trade-off between quality of the Pareto front approximation and execution time might be enabled tuning the optimization hyper-parameters, i.e. the calibration set size (*calset*) and the number of evaluations (*maxeval*).

Table 3.6: Figure of merits of the Pareto fronts for the selected benchmarks.

Benchmark	**#Points**	**Δ Top-1**	**Savings**	**Ex. Time**
ResNet18 v1	6	2.5%	27.4%	8 min 32 s
ResNet34 v1	6	3.3%	29.8%	12 min 36 s
ResNet50 v1	6	2.0%	25.6%	25 min 17 s
SqueezeNet v1.1	5	1.3%	28.4%	6 min 19 s
MobileNet v2	5	8.0%	19.2%	14 min 33 s

Dynamic Precision vs. Static Topology Scaling. Fig. 3.7 gives a comparison between the proposed dynamic precision scaling vs static topology scaling. The plot shows the full-precision (dots) and the Pareto points (crosses) returned by our optimization for the three versions of ResNets adopted in this work. Energy values are normalized to the maximum energy configuration, i.e. the right-most dot in the plot (ResNet50). Topology scaling means selecting one of the dots depending on the energy budget, which implies the storing of three distinct models on-chip. First, precision scaling enables is a finer knob that offers more options (7 Points for each network vs 3 points of topology scaling). The analysis reveals adaptive precision scaling is a more flexible strategy. On the other hand, topology scaling may achieve better energy efficiency at the same accuracy level (ResNet18 vs ResNet34 mixed). Nonetheless, an interesting observation is that the two can be superimposed to extend the operating range. Obviously, this comes at the cost of more memory occupation (one weight-set for each loaded topology) and should

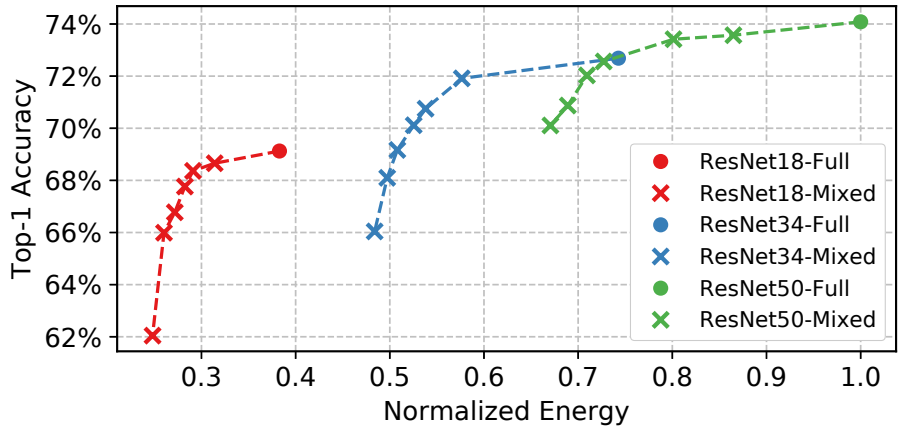


Figure 3.7: Energy vs Accuracy of topology and precision scaling. Energy is normalized with respect to ResNet50 at full-precision (green dot).

be carefully assessed considering other applications running in background may call for some memory space too.

Overhead of weight re-configuration. As already discussed, an efficient implementation of precision scaling requires a weight reconfiguration step to guarantee a memory-friendly layout of low-precision weights. To ensure the savings of precision scaling, the energy cost of this step should be limited. We thereby assessed the upper-bound of such energy cost. Specifically, we estimated the energy needed to load the original parameters from the DRAM and store the rounded parameters in the SRAM when all the layers switch to 8-bit (i.e. the worst-case scenario). Collected values are reported in Table 3.7, which reports the energy cost normalized to the energy of a single inference at full-precision for each benchmark. Overall, weight re-configuration has a limited impact on the inference total cost (less than 10% in the worst case). We observed a higher overhead for ResNets (from 8% to 9.6%), whereas smaller values for less complex architectures, i.e. SqueezeNet (4.4%) and MobileNet (6.3%). However, reported values refer to a pessimistic estimation. First, in actual cases only a subset of the layers is set to low-precision. Second, it is fair to assume that switching from one configuration to another is a less frequent event, that is, the network remains at the same operating point for multiple inferences mitigating the costs.

Ablation studies

This section aims to assess the proposed optimization strategy, in particular, the performance of precision scaling with rounding and the quality of the adopted penalty score.

Table 3.7: Upper bound of the energy cost for weight re-configuration. Values are normalized to the energy of a full-precision inference.

Benchmark	Energy Cost
ResNet18 v1	9.6%
ResNet34 v1	9.2%
ResNet50 v1	8.0%
SqueezeNet v1.1	4.4%
MobileNet v2	6.3%

Weight re-configuration with rounding. As a comparative analysis, Table 3.8 shows the results of the proposed algorithm when weight re-configuration is implemented by means of truncation. For all the networks, the algorithm fails to find solutions with $D_{\mathrm{KL}} < 0.5$, with negative effects also on accuracy. For MobileNet and ResNet50, all the returned solutions have accuracy $< 1\%$, with no operating points of practical use—indicated with a dash. For ResNet18 and ResNet34, the average accuracy degradation is high (31.0% and 42.7%, respectively) while energy savings are small (9.4% and 12.5%). Only for SqueezeNet there are solutions with D_{KL} greater than 0.5. Indeed, the average accuracy loss is within an acceptable range, but if compared with the rounding strategy, the loss is higher (3.3% vs 1.3%) with energy savings halved (14% vs 28%). Regarding the execution time, there is no significant difference. This is to be expected since during the optimization loop rounding and truncation needs similar processing time and their contribution is negligible with respect to the overall time (which is dominated by the evaluation of D_{KL}).

Table 3.8: Figure of merits of the Pareto fronts for the selected benchmarks using weight re-configuration with truncation.

Benchmark	#Points	Δ Top-1	Savings	Ex. Time
ResNet18 v1[1]	5	31.0%	9.4%	8 min 34 s
ResNet34 v1[1]	3	42.7%	12.5%	12 min 39 s
ResNet50 v1[1,2]	–	–	–	25 min 16 s
SqueezeNet v1.1	4	3.3%	14.0%	6 min 17 s
MobileNet v2[1,2]	–	–	–	14 min 32 s

[1] KL constraint not met.
[2] Top-1 Accuracy less than 1%.

Penalty score. To show the importance of a good penalty score and motivate

our custom version, we compared the performance of a standard NSGA-II implementation obtained by simply dropping line 10 of Algorithm 2). As a case study, we show the analysis conducted on MobileNet v2. Fig. 3.8 reveals that a standard implementation (NSGA-II, blue crosses) achieves higher energy savings, yet with a dramatic accuracy degradation: all the operating points have an accuracy below 30%, thus being impractical solutions. The introduction of the penalty score (NSGA-II*, red crosses) ensures that all the operating points have accuracy above 60%. To note that the red crosses are the same as the right-most plot in Fig. 3.6.

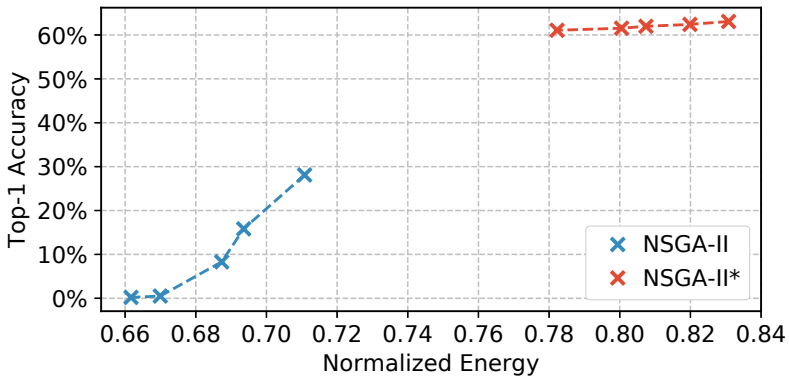
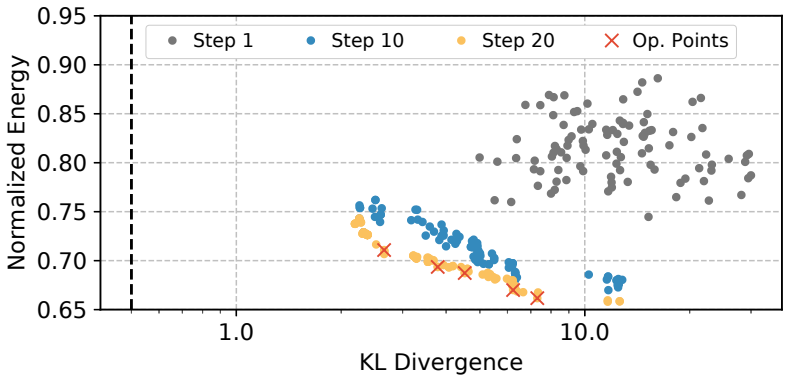


Figure 3.8: Pareto curve of MobileNet v2 with standard NSGA-II and our NSGA-II* implementation with penalty score.

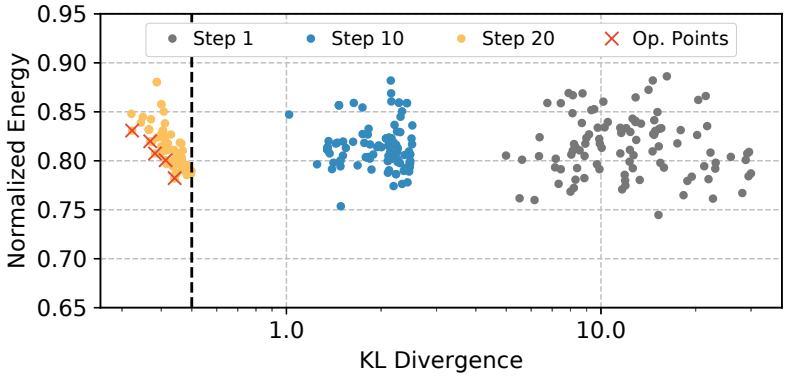
More insights can be drawn observing the evolution of the population across different iterations of Algorithm 2 (lines 5–11) in the two versions. Fig. 3.9 shows the evolution of the population across step 1, 10, and 20. Step 1 corresponds to the initial solution randomly generated. The dotted line indicates the KL threshold ($KL_{\text{th}} = 0.5$ in our experiments). The final solutions (red crosses) are selected from Step 20 using ϵ-nondominated sorting. NSGA-II (left plot) pushes the population towards the bottom of the plot, i.e. the low-energy region. However, D_{KL} keeps very close to that of a random solution, reflecting to final solutions with accuracy $< 30\%$. Thanks to the introduction of the penalty score, NSGA-II* (right plot) pushes the candidate solutions towards the left, i.e. the low D_{KL} region. Indeed, this procedure ensures to identify those mixed-precision configurations that keep the top-1 accuracy at reasonable levels, still guaranteeing significant energy savings ($> 15\%$).

3.3.4 Discussion

This work presented the design & optimization of energy-driven Adaptive ConvNets via per-layer precision scaling in software-programmable accelerators with mixed-precision arithmetic. In contrast to existing compression techniques that



(a)



(b)

Figure 3.9: Population evolution across different iterations for MobileNet v2 with NSGA-II (a) and NSGA-II* (b).

are conceived as static optimizations, the proposed design enables run-time energy-accuracy trade-off through dynamic resource management, leveraging the intrinsic resilience of neural networks to arithmetic errors. Based on a multi-objective search engine, the presented algorithm drives the optimization towards high-accuracy solutions, with short execution time even in very deep networks (> 50 layers). As a key distinctive feature, our optimization can be applied to any pre-trained models, without the need for additional training iterations, thereby ensures a fast deployment process. Experimental results proved the efficacy of our proposal, with trade-offs spanning over a large energy range and limited accuracy degradation. In state-of-the-art ConvNets for vision applications trained on ImageNet, we estimated energy savings up to 35.2% with an accuracy loss $< 8\%$ in the worst case.

3.4 Scalable-Effort ConvNets for Multilevel Classification

In this second part of the chapter, we present scalable-effort ConvNets. The remainder of the text is organized as follows. In Section 3.4.1, we first outline the motivation and our reasoning behind the presented strategy, which relies on the idea of multilevel classification. In Section 3.4.2, we formalize the multilevel classification problem for ConvNets. Then, we present a cross-layer strategy for single-model multi-precision multiply&accumulate (MAC) arithmetic (Section 3.4.3), together with its application for processing standard ConvNets (Section 3.4.4). In Section 3.4.5, we introduce an algorithm that implements *multi-precision, accuracy-driven per-layer precision assignment* through a greedy search in the abstraction-effort-accuracy optimization space. Finally, we present the experimental results collected on a representative set of ConvNets trained on the ImageNet dataset (Section 3.4.6) and discuss the key achievements of our proposal (Section 3.4.7).

3.4.1 Motivation

Modern ConvNets share a common characteristic: they are designed, trained, and deployed as static models. They are built as flat N-ways classifiers (N is the number of classes) that expend equal effort no matter the surrounding context and the level of the accuracy required by the application. Even though they are inspired by the working principles of the human brain, they lack one of the main characteristics of the human reasoning. We, as humans, can organize the knowledge by means of hierarchical semantic abstractions. This organization enables our reasoning to shift towards the proper level of abstraction depending on the context. We first perceive high-level properties of the surrounding environment, e.g. indoor or outdoor. Only if needed, we focus on low-level features, for instance to identify specific objects. Intuitively, low-level classifications require more effort. We might be interested in classifying not just the place we are, but also things around us, e.g. the type of vehicles in the street, car or truck, or even their brand and model. The abstraction process is typically driven by some external trigger.

This scalable mechanism makes our brain an efficient machine, capable of adapting to the context and reach the effort-accuracy trade-off that minimizes energy waste. Implementing this ability on IoT end-nodes represents a unique opportunity to improve the energy efficiency. For instance, a smart surveillance system could reduce the effort, hence energy, by running high-level classification for most of the time; in case of any suspicious event, more effort is pushed to "understand" and classify the triggering event. Examples of this kind are countless.

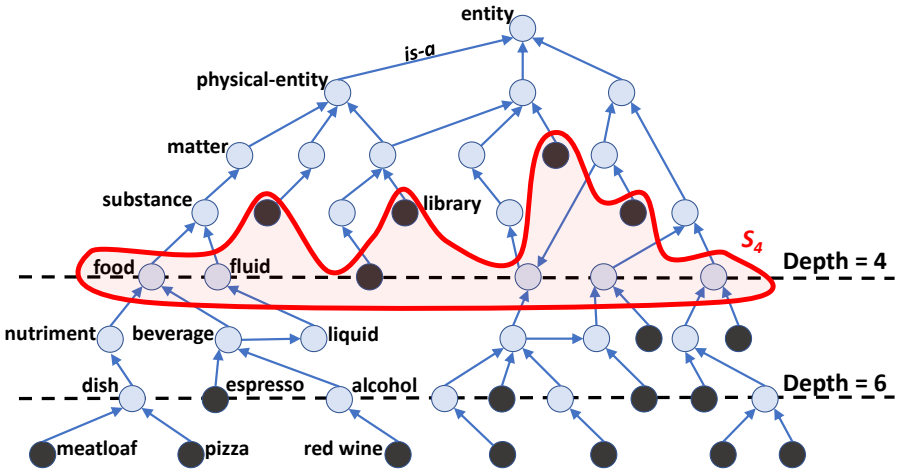


Figure 3.10: Schematic view of a WordNet-like graph.

This work aims to present a design and optimization strategy for the practical implementation of this brain-inspired paradigm with embedded ConvNets. Our proposal leverages (but is not limited to) arithmetic precision as a knob to modulate the computational effort at run-time: *depending on the required level of abstraction (defined by the application, i.e. the context), scalable-effort ConvNets reduce their precision to reach an accuracy target with minimal computational resources.* The working principle is based on an algorithmic relaxation of the multiply&accumulate (MAC) arithmetic, operating on a single set of weights to implement different precision options: *full* (8×8bit), *mixed* (8×4bit), *half* (4×4bit). This approach avoids the overhead of storing multiple weight-sets for each level of abstraction.

Note that scalable-effort ConvNets for multilevel classification differs from chained ConvNets for hierarchical classification [102] [103]. As will be detailed in the next section, multilevel classification encompasses semantic meaning [104] whereas hierarchical classification does not.

3.4.2 Multilevel Classification

Common datasets used for training ConvNets reflect the hierarchical organization of concepts built by the human brain during categorization. In this work, we adopt ImageNet [93], one of the most common dataset for large-scale image classification. ImageNet is composed by 1.2M images organized in 1000 classes. These classes were extracted from the WordNet database [105], a structured database of interconnected concepts.

In WordNet, concepts are organized in *synsets*, each of them groups word with

similar meanings. The WordNet structure describes the semantic relationships among synsets. A schematic view of a sub-set of WordNet is depicted in fig.. 3.10. Connections among synsets express the relation between more specific concepts (called *hyponym*) and more general ones (*hypernym*). This relation is commonly defined as the*is-a* relation. For instance, in figure *pizza* is-a *dish*. Each synset can be considered as a possible class in the multilevel classification problem, where a hyponym is a class of inferior abstraction than to its hypernym(s).

As can be inferred from the figure, the structure of WordNet can be represented as a direct acyclic graph (DAG), where each synset might have multiple hyponyms, e.g. both *pizza* and *meatloaf* are a *dish*. The set of nodes V represents the synsets (i.e. classes), the set of oriented edges E expresses the semantic relationships among synsets. An edge $e_{i,j}$ indicates that a node v_j (e.g. synset representing the class *beverage*) is a hypernym of node v_i (e.g. the subclass *alcohol*). At the top of the hierarchy, there is only one synset v_0 called *entity*; v_0 is the root of each semantic path. The abstraction level of node v_i is defined by its depth d_i, i.e., the length of the shortest path to v_0; d=6 for the node *dish*.

The multilevel classification problem can be formulated as follows: *Given a generic level of abstraction L, label the input image I with the class $s_i \in S_L$ that has the closest semantic meaning, i.e., s_i s.t. the class probability $p(s_i)$ is maximum.*

The level of abstraction L is represented as a cut C_L at depth L (dashed lines in Fig. 3.10). According to this formulation, L=0 refers to the most general level of abstraction.The set S_L is composed by the target classes s_i available at level L; S_L is the union between the set of nodes traversed by C_L and the set of 0-indegree nodes behind C_L. In Fig. 3.10, S_4={*food*, *fluid*, ...}∪{*library*, ...}. The $p(s_i)$ defines the probability that input I belongs to class s_i:

$$p(s_i) = \sum_{v_j \in FanIn(s_i)} p(v_j) \tag{3.7}$$

where v_j is a 0-indegree node, i.e. a synset w/o hyponyms (black nodes in Fig. 3.10) in the transitive fan-in of s_i. For instance, for s_i={*food* }, the set of v_j is {*meatloaf* }, {*pizza* }, {*espresso* }, and {*red wine* }. Note that nodes v_j represent the N classes of ImageNet (N=1000) commonly used to train ConvNets. Their distance from v_0 may differ as depicted in the DAG. Indeed, $p(v_j)$ represents the probability that the ConvNet labels the input I as v_j.

Since synsets might be hypernym of multiple hyponym, the target classes $s_i \in S$ may have intersecting fan-in. The node v_j={*red wine* } belongs to the transitive fan-in of both {*food* } and {*fluid* }, the same is for {*espresso* }. In these cases, we adopted a shortest-path policy: nodes v_j are assigned to the class s_i with minimum distance. Therefore, both {*red wine* } and {*espresso* } contribute to $p(s_j$={*food* }) and not to $p(s_j$={*fluid* }). As {*fluid* } has no remaining children

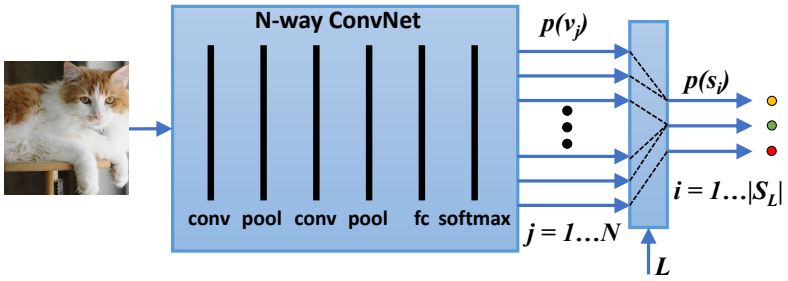


Figure 3.11: Multilevel classification with ConvNets.

in its fan-in, it is dropped from the set of available classes S_L. This approach guarantees $\sum_{s_i \in S_L} p(s_i)=1 \ \forall L$.

Standard ConvNets are built as flat N-way classifiers designed and trained to produce a probability over the N available classes, the black nodes in Fig. 3.10 (nodes v_j according to our formulation). To perform multilevel classification with ConvNets, an additional processing layer is needed. As shown in Fig. 3.11, this layer is fed with the level of abstraction L and computes Eq. (3.7). The processing involves N arithmetic sums arranged as per the set S_L (a LUT contains S_L for $\forall L$). Therefore, it does not require any dedicated hardware and the computational overhead is negligible if compared to the millions of arithmetic operations and weights of a standard ConvNet.

The presented formulation represents just one of the possible alternatives that define the multilevel classification problem. Our idea of scalable-effort ConvNets can be extended to other formulations, e.g. different approaches for extracting the high-level classes s_i.

Note that multilevel classification differs from hierarchical classification [102]. Within a multilevel classification problem, the number of target classes S_L depends on the level of abstraction L, the cardinality $|S|$ gets larger with L. Conversely, hierarchical classification is based on a static definition of the target classes. Specifically, it always operates on the same N classes, those defined at the bottom level of abstraction (black nodes in Fig. 3.10). Hierarchical classification works under the assumption that all inputs are not equal. For instance, in some cases the main features of the input might be masked, making the classification harder. Depending on the complexity of the input, the same N-way classification can be performed with less effort. Here comes the difference: whereas a multilevel strategy exploits the lower complexity brought by different abstractions, a hierarchical strategy exploits the lower complexity of the inputs.

Existing approaches for hierarchical classification encompass an iterative procedure where a chain of ConvNets with growing levels of complexity is processed

in sequence till consensus is achieved. Since most of the inputs show lower complexity, some links of the chain can be skipped, reducing the overall computational effort. For instance, the method introduced in [102] learns to distinguish "easy" classes from "difficult" classes during training. At inference time, "easy" classes are detected by a coarse category classifier, "difficult" classes are fed to a finer classifier. An extension of this strategy which involves longer chains of quantized micro-classifiers is described in [103].

3.4.3 Precision Scalable Arithmetic

Our implementation of scalable-effort ConvNets makes use of scalable-precision MAC operations. The computational effort is proportional to the number of arithmetic operations needed to process the ConvNet layers. We opted for a dynamic precision scaling strategy supported by the design of a variable-latency MAC hardware unit. This enable to tune the precision at run-time with minimal overhead.

Given two $M \times M$ matrices, i.e. the input map $\underline{\mathbf{I}}$ and the weights $\underline{\mathbf{W}}$, their convolution is the dot-product of the unrolled vectors I and W, each with length $M \times M$. Assuming each item I_i and W_i has a bit-width $N{=}2^n$ ($n{=}3$ in this work), their scalar product $I_i \times W_i$ can be implemented as a four-stage iterative procedure during which the most significant parts (I_i^{H}, W_i^{H}) and the least significant parts (I_i^{L}, W_i^{L}) are multiplied, shifted and accumulated [106], as schematically depicted in Fig. 3.12. Both I_i^{L} and W_i^{L} are $K = N/2$-bit unsigned integers, while I_i^{H} and W_i^{H} are $K = N/2$-bit signed integers. Three precision options can be achieved by stopping the procedure at different steps: *half* (K×K bit) 1-cycle, *mixed* (N×K bit) 2-cycles, *full* (N×N bit) 4-cycles.

Scalable Multiply-Accumulate Algorithm

The same three options can be extended to the dot-product, as described in Algorithm 3. At *half* precision, both the operands I_i and W_i are reduced to K bits. The first loop (lines 1-6) processes the most significant parts I_i^{H}, W_i^{H}. The result of the accumulation r is shifted by $2 \cdot K$ positions (line 8) and then returned (line 9). The bit shifting (line 8) is required to align the radix-point to that of the biases[2] ($2N$-bit operands). In order to reduce the error introduced by bit-width reduction, the weights W_i are rounded (line 3) using a *half-even rounding* scheme; the latter has been indicated as one of the most efficient when dealing with operand scaling [107]. At *mixed* precision, only one operand, the weight W_i, is reduced to K bits. Under this option, the second loop (lines 11-16) takes part

[2]The bias is the offset parameter of a ConvNet neuron

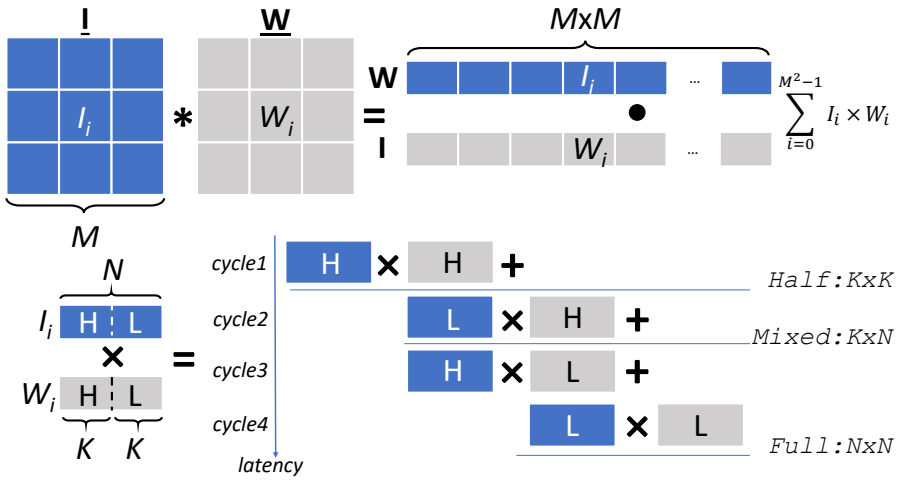


Figure 3.12: Iterative product procedure to compute the dot-product between two vectors.

to the computation; it iterates on W_i^{H} and I_i^{L}. The accumulation r is shifted by K positions (line 18) and then returned (line 19). Also in this case, W_i^{H} is rounded (lines 12-15) to mitigate the bit-width reduction. At *full* precision, both W_i and I_i are taken as N bit operands. In this case the last two loops (lines 20-24) come into play; they iterate on the least significant parts W_i^{L} and I_i (both H and L) thus to complete the remaining part of the multiplication. Rounding (lines 5 and 15) is not required under this option.

Processing Element

The structural components of the presented PE are illustrated in Fig. 3.13 for N=8. Operands I and W (and B, the bias) are stored into dedicated 8-bit (16-bit) registers. The multiplexers (MUXes) feed the arithmetic blocks with the proper portion of the operand (H or L), as explained in Algorithm 3. These MUXes are driven by an external control unit (not shown in the picture) through the signal sel_I, sel_W, sel_B.

The *round* unit implements the half-even rounding; rounded inputs are used for *half* and *mixed* precision (Algorithm 3). The half-even rounding takes W_H and W_L as input (4+4-bit) and returns the rounded value (4-bit).

The PE has a 5×5 multiplier, where the 5^{th} bit is used for the sign extension of the operands. The most significant parts (I_i^{H}, W_i^{H}) are signed, while the least significant parts (I_i^{L}, W_i^{L}) are unsigned; the MSB of I_i^{L} and W_i^{L} belongs to the module, while the MSB of I_i^{H} and W_i^{H} is the sign. The following mechanism is implemented: for I_i^{H} and W_i^{H}, the sign (MSB) is extended to the 5^{th} bit, while W_i^{L} is extended with a 0; an external control signal (not shown in the picture)

Algorithm 3: Scalable-precision MAC algorithm

Input: I, W, *precision*
Output: Dot-Product r

for $i = 0;\ i < M;\ i++$ **do**
  if *precision == half // mixed* **then**
    $W_i^{\mathrm{K}} = \mathrm{round}(W_i, K)$
  else
    $W_i^{\mathrm{K}} = W_i^{\mathrm{H}}$
  $r += I_i^{\mathrm{H}} \times W_i^{\mathrm{K}}$
if *precision == half* **then**
  $r = r \ll 2 \cdot K$; `// bias radix-point alignment`
  return r ; `// half precision: KxK`
$r = r \ll K$
for $i = 0;\ i < M;\ i = i + 1$ **do**
  if *precision == mixed* **then**
    $W_i^{\mathrm{K}} = \mathrm{round}(W_i, K)$
  else if *precision == full* **then**
    $W_i^{\mathrm{K}} = W_i^{\mathrm{H}}$
  $r += I_i^{\mathrm{L}} \times W_i^{\mathrm{K}}$
if *precision == mixed* **then**
  $r = r \ll K$; `// bias radix-point alignment`
  return r ; `// mixed precision: KxN`
for $i = 0;\ i < M;\ i = i + 1$ **do**
  $r += I_i^{\mathrm{H}} \times W_i^{\mathrm{L}}$
$r = r \ll K$
for $i = 0;\ i < M;\ i = i + 1$ **do**
  $r += I_i^{\mathrm{L}} \times W_i^{\mathrm{L}}$
return r ; `// full precision: NxN`

selects the proper option. Other sign extensions take place depending on the desired precision: at $half$ precision, I_i^{L} contains a 4-bit feature, therefore its sign is extended; at $full$ and *mixed* precision, I_i^{L} contains the least significant part of an 8-bit feature, and hence it is concatenated with a 0.

An additional MUX allows to accumulate partial products or to add the bias (sel_P). The biases are $2N$-bit operands; their radix-point is aligned to that of the accumulation result. Bias addition takes a fixed latency, i.e. two cycles, one for B_{L}, one for B_{H}. Since the output of the multiplier is 10-bit, B_{H} (signed) goes through a sign extension, B_{L} (unsigned) is zero-extended.

The 32-bit *accumulator* incorporates an embedded saturation logic that handles underflow/overflow conditions. A programmable *shifter* implements the shift of

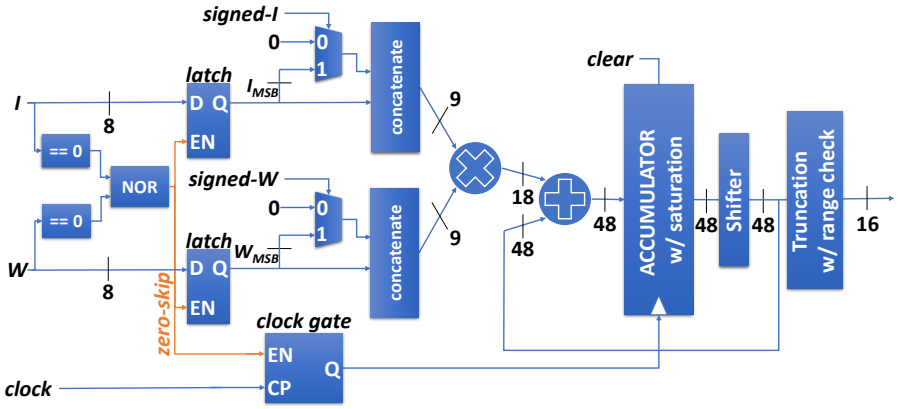


Figure 3.13: Structural view of the scalable-precision processing element.

partial results (see Algorithm 3).

The *truncation logic* is in charge of managing the storage of the final result into the output register O_8. The *range check* logic drives the saturation bit when the result does not fit the word-length (4 or 8). Depending on the bit-width requirement of the next layer[3], i.e. 4 or 8 bits, O_H and O_L are properly filled: for *half* precision, the result is stored either in O_H or O_L thus to enable interleaved memory scheme (O_8 contains two 4-bit activations); for *full* or *mixed* precision, O_H and O_L are filled in parallel with the most and least significant part of the accumulation.

The *zero-skipping* strategy [44] is implemented through latch-based operand isolation and clock-gating. If one of the operands equals to zero, then (*i*) latches at the input of the multiplier block the propagation of data minimizing the switching activity, (*ii*) a clock-gating cell disables the clock input of the accumulator thus reducing the equivalent load capacitance of the clock signal.

Table 3.9 shows the (normalized) computational effort as number of atomic operations, i.e. 4×4 MAC.

Table 3.9: Normalized computational effort (#Operations) at different precision settings (1 Operation = 1 MAC 4x4).

Precision	MAC	Bias-Add
Full	4	2
Mixed	2	2
Half	1	2

The proposed PE has been designed in Verilog and then implemented using the

[3]In scalable-effort ConvNets different layers may be assigned to different bit-widths.

Synopsys Galaxy Suite (versions L-2016.03). The design kit belongs to a 28 nm UTBB FDSOI industrial technology. The clock frequency is 1.0 ns for typical process corner, supply voltage 1.10 V, and room temperature. Power consumption is estimated using back-annotated switching activities extracted from a realistic ConvNet workload. The resulting core area is 1589 μm^2; the average power consumption is 1.23 mW. A comparison with a standard (i.e. single-precision) PE reveals the overhead is marginal. We designed a 4×4 PE w/o (*i*) the sign extension logic, (*ii*) the rounding logic; it only includes the interleaved read-in and write-out memory scheme (still useful for optimal buffering). The design is run using the same constraints and operating conditions used for our PE. The resulting core area is 1520 μm^2, while the power consumption 1.20 mW. Our scalable-precision PE is just 4.5% larger and consumes only 2.5% more power.

3.4.4 Fixed-point Quantization & Fine-Tuning

The design of *multi-precision* fixed-point ConvNets requires proper tools that emulate integer arithmetic units during the training and validation stages. To this purpose, we built a dedicated emulation framework, which can run on GP-GPUs to guarantee fast processing. The framework was integrated into the deep learning libraries available in PyTorch. Specifically, it leverages the *fake-quantization* approached presented in [108, 109]. During the feed-forward execution of the network, software wrappers placed across the layers convert intermediate activations and weights (stored in fixed-point) to the 32-bit floating-point; after processing, outputs are converted back to fixed-point.

Since activations and weights cover various ranges across different layers, we also adopted a dynamic fixed-point scheme [108] where the position of the radix-point is determined on a per-layer bias. For each layer, the statistics of activations are extracted over 5k images randomly picked from the training set; the integer-length (IL) is then computed as:

$$IL_l = \lfloor \log_2 \left[\max \left(|x_{\min}|, |x_{\max}| \right) \right] + 2 \rfloor \tag{3.8}$$

l identifies the layer, $x_{\min}$ and $x_{\max}$ are the minimum and maximum value for the activations of that layer l. The fraction-length FL_l is the difference between the word-length (given by precision) and the integer-length IL_l. The same method is adopted for the integer- and the fraction-length of weights.

The emulation also accounts for the bit-width of the accumulation register and its saturation/truncation scheme. The outputs of each layer are first scaled to 32-bit (see Fig. 3.13), then shifted and truncated as needed depending on the precision of the next layer.

In order to reduce the number of operations, batch-normalization layers are

fused with convolutional layers applying a re-scaling of weights and biases. Average pooling layers are implemented through *depth-wise* convolutional layers.

After quantization, the ConvNet may lose its prediction quality. However, the accuracy loss can be recovered (in some cases only partially) through additional re-training steps. This procedure, commonly called *fine-tuning* [108], works as follows: (*i*) the forward-propagation is run with fixed-point emulation; (*ii*) during back-propagation weights are kept in a floating-point representation thus to allow small weight updates; (*iii*) weights are quantized at every mini-batch using stochastic rounding. Our framework slightly differs as quantization (with stochastic rounding) is performed at the end of each epoch (rather than every mini-batch). We observed that this variant improves the convergence of the training process. The framework is embedded into PyTorch (version 0.3.1 for Python 3.6.0) and performed on a machine equipped with a Titan-XP GPU by NVIDIA (cuDNN 7.0 and CUDA 8.0).

3.4.5 Precision Assignment Heuristic

The working principle of scalable-effort ConvNets lies under the ability to exploit a higher abstraction, i.e. the classification of more "general" concepts/object, to reduce the computational effort, i.e. the arithmetic precision during inference.

As shown in previous works [44] and confirmed by our experiments (more details in Section 3.4.6), the spatial granularity to which precision scaling is applied plays an important role. The dynamic precision scaling adopted in this work is based on a *per-layer* assignment, namely, each computational layer is processed on its own independent arithmetic precision. This scheme enables a finer tuning of the effort-accuracy trade-off.

Due to the high cardinality of the problem, an exhaustive search is unpractical. Given N_l as the number of layers and N_p=3 the precision options (*full*, *mixed*, *low*), the available solutions are $N_p^{N_l}$; with N_l=55 (one of the benchmarks used in this work), 1.7×10^{26} solutions.

Algorithm 4 shows the pseudo-code of our greedy strategy for the multilevel, accuracy-driven, per-layer precision assignment problem. The procedure takes as input the fine-tuned ConvNet model at *full* precision, i.e. activations and weights of all the layer to 8-bit, and the level of abstraction L; it returns a near-optimal layer-by-layer precision mapping s.t. a target accuracy is met with the minimum number of arithmetic operations (equivalent number of 4×4 MAC). The target accuracy is given as input parameter as well.

The layers of the ConvNet model are sorted (line 1) following a descending complexity order. The complexity, defined as the total number of operations N_{ops}, is function of the layer topology (e.g. number and size of kernels) and its current

Algorithm 4: Heuristic optimization algorithm

Input: Model, L, CalibrationSet, Target Accuracy
Output: Precision-Scaled Model

```
1  SortedLayers ← sort(Layers, N_ops)
2  i = 0
3  while i < num(layers) do
4  |   Layer ← SortedLayers[i]
5  |   Decrease precision of Layer
6  |   Accuracy ← check(Model, CalibrationSet, L)
7  |   if Accuracy < TargetAccuracy then
8  |   |   Increase precision of Layer
9  |   |   i ← i + 1
10 |   |   continue
11 |   else
12 |   |   SortedLayers ← filter(Layers, half precision)
13 |   |   SortedLayers ← sort(Layers, N_ops)
14 |   |   i ← 0
15 return Model
```

precision; it is calculated as:

$$N_{\text{ops}} = \alpha_{\text{mac}} * N_{\text{mac}} + \alpha_{\text{bias}} * N_{\text{bias}} \tag{3.9}$$

where N_{mac} is the number of MAC operations and N_{bias} is the number of bias additions; the two factors α_{mac} and α_{bias} are derating factors that account for precision according to Table 3.9.

At each iteration, the first layer in the list *SortedLayers* is downgraded to lower precision: *full* $\mapsto$*mixed* or *mixed* $\mapsto$*low* (lines 4-5). The accuracy is then checked on the calibration set (line 6) for abstraction L: if the achieved accuracy is lower than the target accuracy, the precision of the layer is restored to its previous value (lines 7-8) and computation moves to the next layer (line 9); otherwise the new precision assignment is accepted (line 12), the ordered list is updated (lower precision implies lower complexity) (line 13), and the index i is reset thus to restart from the first layer (line 14). The optimization ends when the precision of any layer can be no longer reduced, that is, any further precision scaling of any layer induces accuracy violations.

The accuracy check (line 6) is run over a set of images randomly picked from the ImageNet validation set; we refer to this set as the *calibration set*. A separated subset (hereafter referred as the *test set*) of 40k images from the ImageNet validation set is used at a post-optimization stage to effectively quantify the accuracy of the precision-scaled ConvNet.

3.4.6 Experimental Results

In this section, we first validate the fine-tuning procedure (Section 3.4.4 adopted to generate accurate fixed-point ConvNets for flat N-way classification (i.e. withouth multilevel abstraction). These models represents the starting point over which scalable-effort ConvNets are built. Second, we assess the figure of merits of multilevel classification with singe-model *multi-precision* ConvNets (i.e. scalable-effort ConvNets).

The acronyms used in the text are as follows: **fp-32** for floating-point models; **Q-fx** for fixed-point quantized models w/o fine-tuning (8-bit for all the layers, both activations and weights, 16-bit for biases); **Fine-fx** for single-precision, fine-tuned models in fixed-point where all layers share the same precision; **scalable-effort**: our proposal, i.e. multi-precision models whose layer precision is assigned using Algorithm 4. For **Fine-fx** and **scalable-effort**, the available precision options are those introduced in Section 3.4.3: *full* (weights and activations 8-bit, biases 16-bit), *mixed* (weights 4-bit, activations 8-bit, biases 16-bit), *half* (weights and activations 4-bit, biases 16-bit).

Benchmarks

We adopted as benchmarks three state-of-the-art ConvNets for image classification: AlexNet, SqueezeNet (version 1.1), and MobileNet (version 2.0). Since we target embedded ConvNets, SqueezeNet and MobileNet represent the two most interesting benchmarks. They lack fully-connected layers, therefore, they are suited for low-power applications running on embedded platforms. We included AlexNet to demonstrate that our solution works for standard ConvNets too. The pre-trained fp-32 models trained on ImageNet 2012 [110] belong to open-source repositories.

Table 3.10: List of benchmarks.

ConvNet (fp-32)	#Param	#MAC	#Layers	Top-1 Acc.
AlexNet	61M	715M	8	56.544%
SqueezeNet v1.1	1.2M	352M	27	58.208%
MobileNet v2	3.5M	321M	55	71.774%

Fine-Tuning Validation

The results reported in this sub-section refer to a standard flat N-way classification, namely without any abstraction. Table 3.10 summarizes the characteristics of the three pre-trained **fp-32** ConvNets models. The number of parameters

Table 3.11: Top-1 accuracy difference (ΔAccuracy) with respect to fp-32 after quantization (Q-fx) and after fine-tuning at *full* precision (Fine-fx:*full*).

ConvNet	**Q-fx**	**Fine-fx**:*full*
AlexNet	-3.784%	-1.244%
SqueezeNet v1.1	-16.654%	-0.740%
MobileNet v2	-50.604%	-2.418%

(#Param) includes both weights and biases for all the layers; the total number of floating-point operations (#MAC) covers the number of bias additions. Both #Param and #MAC are calculated after fusing the batch-normalization layers with the convolutional layers. The column #Layers just shows the number of layers that takes part in the optimization process: convolutional, average pooling, fully-connected layers (when present). The top-1 classification accuracy (Top-1 Acc.) is measured using the ImageNet validation set (50k images). Collected results are consistent with those reported in [1, 95, 111].

Previous works have proven *full* precision **Fine-fx** models (**Fine-fx**:*full* hereafter) guarantee the same accuracy of **fp-32** [109]. Table 3.11 confirms this trend and validates the convergence of the proposed fine-tuning framework (Section 3.4.4). It shows accuracy difference (ΔAccuracy) with respect to the **fp-32** model obtained for the three benchmarks, w/o and w/ fine-tuning enabled, i.e. **Q-fx** and **Fine-fx**, respectively.

Our fine-tuning leverages the Adam optimizer [54] with learning rate 1.0×10^{-6} and batch size 128. The training process iterates over 25 epochs using 200k images randomly extracted from the ImageNet training set. As shown in table 3.11, accuracy loss due to quantization is almost fully recovered: **Fine-fx**:*full* models show a top-1 ΔAccuracy of 2.4% in the worst-case. Supported by these results, we used **Fine-fx**:*full* models as entry level for our optimization. Clearly, over-parametrized ConvNets like AlexNet shows higher resilience to quantization: ΔAccuracy is limited to −4% for **Q-fx**. Lightweight ConvNets for embedded applications show dramatic losses: from 58.208% to 41.554% for SqueezeNet, from 71.774% to 21.170% for MobileNet (Table 3.10 vs. Table 3.11).

Moreover, we studied the resilience of **Fine-fx** models at lower precision settings, i.e. *mixed* and *half*. The analysis is reported in Fig. 3.14, which shows the evolution of the top-1 test accuracy during the fine-tuning process. The plot refers to SqueezeNet (for MobileNet the same analysis holds). At epoch 0 the accuracy is the one measured after quantization (e.g. 41.554% for *full* precision). Fine-tuning enables substantial recovery for **Fine-fx**:*mixed*: from 0.224% (epoch 0) to 40.294% (epoch 25). Results get worse for **Fine-fx**:*low*: accuracy never goes above 0.10%.

These findings suggest precisions lower than 8-bit are too aggressive for ConvNets designed for embedded applications, like SqueezeNet; more computational effort is required indeed. Within the context of adaptive ConvNets, the optimization room given by single-precision models is limited, as 8-bit is by itself an effort lower-bound. By contrast, a finer spatial precision assignment (i.e. *per-layer* rather than *per-net*) opens to more flexible trade-offs.

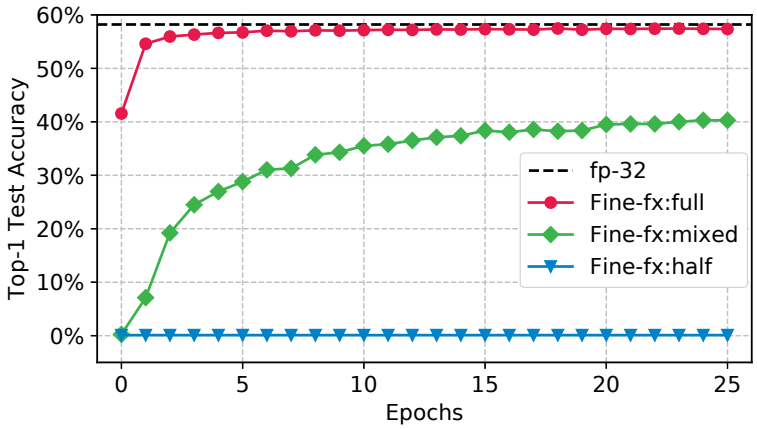


Figure 3.14: SqueezeNet Top-1 Accuracy during fine-tuning. Black line (dashed): **fp-32.**; Red line (○) **Fine-fx**:*full* (8x8); Green line (◇): **Fine-fx**:*mixed* (8x4); Blue line (▽): **Fine-fx**:*half* (4x4)

.

Scalable-Effort ConvNets

This section focuses on the gains brought by **scalable-effort** ConvNets. The analysis is conducted at four levels of abstraction L: 4, 6, 8 and 10; a smaller value of L refers to a higher abstraction (L=0 as the highest level). Results at the lowest level, i.e. L=16, are reported for a more fair comparison. To notice that L=16 is the same as a standard flat N-way classification (N=1000 for ImageNet).

Table 3.12 collects the number of classes (#Classes) for different values of L; the same table shows the top-1 accuracy achieved with **Fine-fx** at *full* precision. Reported numbers reveal a clear trend: the higher the abstraction level, the higher the classification accuracy. The three ConvNets achieves a remarkable accuracy above 93%. It is worth to emphasize that when L changes, the weights used to run the inference still remain the same; what differs is the way output probabilities are summed up (as described Fig. 3.11). This guarantees zero overhead in terms of memory footprint (single model storage) and negligible computational penalty (N=1000 sums over hundreds of millions of MAC).

Table 3.12: Top-1 Accuracy in multilevel classification at different levels (L) of abstraction. The test set is composed by 40k images.

		Top-1 Accuracy, **Fine-fx**:*full* model		
L	**#Classes**	**AlexNet**	**SqueezeNet**	**MobileNet**
4	20	93.655%	93.850%	96.052%
6	86	77.570%	78.205%	85.368%
8	365	69.293%	70.453%	79.750%
10	613	62.972%	64.705%	75.707%
16	1000	55.312%	57.545%	69.748%

The most interesting aspect concerns how **scalable-effort** ConvNets exploit the degree of freedom made available by a multilevel classification. The proposed single-model scalable-precision strategy is validated through an exploration of the 3-dimensional optimization space (abstraction-effort-accuracy). To this purpose, the per-layer precision assignment heuristic (Section 3.3.1) was applied on the three ConvNets under analysis. The heuristic makes use of a calibration set to evaluate the accuracy during the optimization loop. Such calibration set is composed by samples randomly picked from the validation set of ImageNet. Obviously, the size of the calibration set may sensibly affect the optimization process. For this reason, we present results for three different size options: 1k, 5k and 10k images. Concerning the test set, **scalable-effort** ConvNets are tested using 40k images randomly picked from the ImageNet validation set. The intersection between the calibration set and the test set is void.

All the results are reported in Tables 3.13, 3.14, 3.15 (reported at the end of the chapter). The column L contains the level of abstraction. The column #Operations refers to the overall number of operations for each **Fine-fx**:*full* ConvNet. The column Target Accuracy reports the accuracy constraints used for optimal precision assignment. Even though our algorithm accepts any value, our parametric analysis is run using as constraints the accuracy reached with a **Fine-fx**:*full* model at depth L, namely, the values reported in Table 3.12. The column ΔAcc. shows the distance between the accuracy achieved and the target accuracy. The column Op. Savings quantifies the equivalent number of 4×4 operations that can be dropped (thanks to precision scaling) w.r.t. the original model (column #Operations). Finally, the columns 8×4 and 4×4 shows the percentage of MAC at *mixed* and *half* precision respectively, that is how effort is distributed.

For all the three benchmarks, our scalable-effort strategy enables an intelligent use of the available resources. Indeed, depending on the abstraction level, **scalable-effort** ConvNets leave the freedom to choose over different optimal solutions, namely, different Pareto points of the effort-accuracy trade-off. The plot in Fig. 3.15 gives a more intelligible representation of this interesting aspect. It

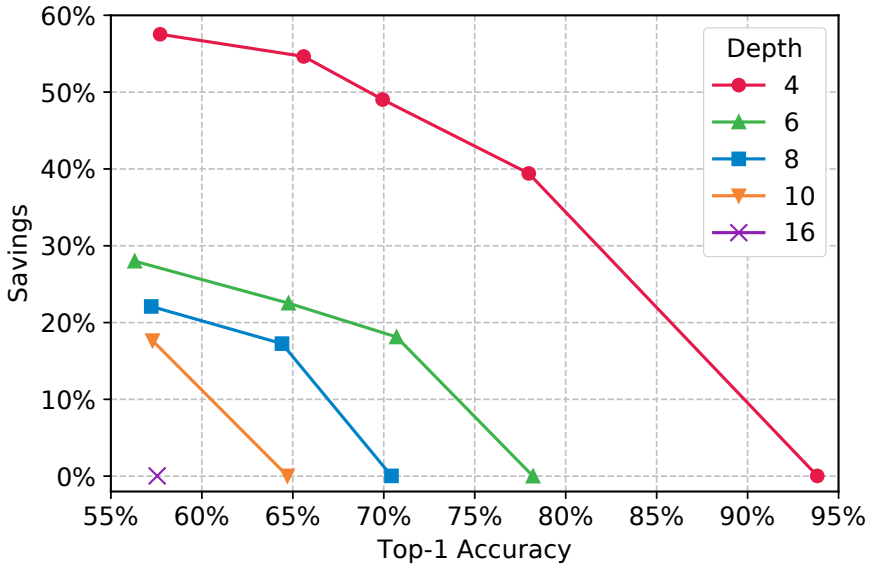


Figure 3.15: Operation Savings vs. Top-1 Accuracy trade-off for SqueezeNet with calibration set size equal to 10000.

shows the Pareto curves obtained by SqueezeNet (calibration set 10k images).

In terms of accuracy, the baseline is at the lowest abstraction, i.e. L=16, when resource savings is zero. The room for optimization released through higher abstractions can be consumed by choosing to (*i*) improve accuracy (best case 93.85% at L=4) or (*ii*) reduce the computational effort and thus save operations (best case 57.53%). In a real-life case, the choice might be done depending on the application, the remaining battery lifetime, the operating hours, the level of light, or any other context variable. Also, a **scalable-effort** ConvNet can move from one Pareto point (or curve) to others with minimal overhead (thanks to unique weight set).

Concerning effort distribution, the higher the abstraction, the larger the number of *mixed* and *low* precision operations. It should be noted that these two options were said to be no adequate for **Fine-fx** models (please refer to Fig. 3.14) due to dramatic accuracy drop.

Looking at the efficiency of the proposed heuristic, it may rarely happen that it fails to reach a Pareto point. An exception can be found in Table 3.15 (coordinates: L=4, target accuracy 79.750% and 85.368% - rows 3 and 4): operations reduces as target accuracy increases. This is due to the greedy nature of the algorithm.

As can be inferred from Tables 3.13, 3.14, 3.15, a too small calibration set may result in sub-optimality. First, it may happen that the accuracy gap from the target constraint (i.e. ΔAcc.) gets too large; this implies some optimization margin is weakly used. Second, the accuracy gap may turn negative; this implies

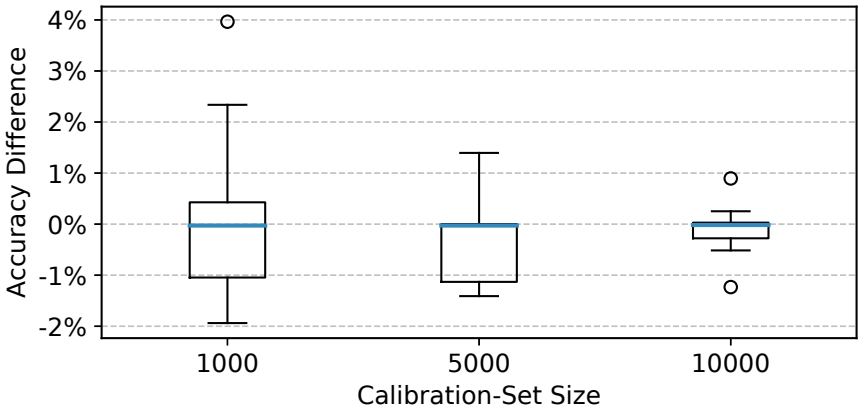


Figure 3.16: Variation of Aaccuracy Difference vs Calibration-Set Size for SqueezeNet.

accuracy violations. As graphically described in the Tukey boxplot of Fig. 3.16 (SqueezeNet), the larger the calibration set, the lower the distance from the target accuracy. A calibration set of 10k images guarantees a spread of [-0.28%, 0.03%] for 50% of the cases, [-1.23%, 0.90%] in the worst case.

3.4.7 Discussion

We demonstrated that standard ConvNets can be used to mimic the multilevel classification paradigm of the human brain. The proposed scalable-effort ConvNets leverage optimal effort distribution through a per-layer dynamic precision scaling strategy.

Our experiments proved that scalable-effort ConvNets can be tuned at run-time to achieve the best effort-accuracy trade-off depending on some external variables. We expect that the joint combination of scalable-effort ConvNets with multilevel training techniques and/or other effort knobs (e.g. pruning) will enable new optimization schemes.

Table 3.13: AlexNet results.

Fine-fx:*full*			Calibration-1k				Calibration-5k				Calibration-10k			
# Operations	L	Target Accuracy	ΔAcc.	Op. Savings	8 × 4	4 × 4	ΔAcc.	Op. Savings	8 × 4	4 × 4	ΔAcc.	Op. Savings	8 × 4	4 × 4
2858 M	4	55.312%	9.362%	40.26%	45%	24%	3.545%	42.00%	38%	31%	3.545%	42.00%	38%	31%
		62.972%	9.295%	38.09%	54%	15%	1.703%	40.26%	45%	24%	1.703%	40.26%	45%	24%
		69.293%	8.267%	37.36%	56%	12%	2.975%	38.09%	54%	15%	2.975%	38.09%	54%	15%
		77.570%	1.808%	34.31%	69%	0%	-0.010%	37.36%	56%	12%	0.228%	37.07%	56%	12%
		93.655%	0.000%	0.00%	0%	0%	-0.770%	21.26%	43%	0%	-0.100%	2.64%	5%	0%
	6	55.312%	5.175%	29.11%	58%	0%	-1.377%	29.98%	56%	2%	4.182%	29.39%	59%	0%
		62.972%	4.943%	26.18%	52%	0%	1.890%	28.92%	43%	10%	1.890%	28.92%	43%	10%
		69.293%	5.325%	21.54%	43%	0%	5.325%	21.54%	43%	0%	5.325%	21.54%	43%	0%
		77.570%	-0.310%	10.79%	22%	0%	-0.010%	2.64%	5%	0%	-0.032%	9.62%	19%	0%
	8	55.312%	3.292%	21.84%	40%	2%	-0.038%	26.46%	53%	0%	1.305%	26.18%	52%	0%
		62.972%	3.650%	21.26%	43%	0%	2.355%	21.54%	43%	0%	2.355%	21.54%	43%	0%
		69.293%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%
	10	55.312%	4.648%	21.26%	43%	0%	2.837%	21.54%	43%	0%	2.837%	21.54%	43%	0%
		62.972%	0.000%	0.00%	0%	0%	-0.137%	2.64%	5%	0%	0.000%	0.00%	0%	0%
	16	55.312%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%

Table 3.14: SqueezeNet results.

Fine-fx:*full*			Calibration-1k				Calibration-5k				Calibration-10k			
# Operations	L	Target Accuracy	ΔAcc.	Op. Savings	8 × 4	4 × 4	ΔAcc.	Op. Savings	8 × 4	4 × 4	ΔAcc.	Op. Savings	8 × 4	4 × 4
1402 M	4	57.545%	2.337%	56.55%	34%	53%	-1.282%	59.05%	48%	47%	0.163%	57.53%	54%	41%
		64.705%	3.965%	52.37%	48%	38%	-1.197%	54.87%	41%	46%	0.895%	54.64%	41%	46%
		70.453%	1.817%	47.73%	39%	38%	-0.078%	51.78%	47%	38%	-0.515%	49.04%	40%	39%
		78.205%	0.855%	39.75%	43%	25%	-1.388%	41.02%	44%	26%	-0.233%	39.42%	42%	25%
		93.850%	-0.647%	13.65%	27%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%
	6	57.545%	-1.688%	28.81%	46%	8%	-1.065%	28.87%	46%	8%	-1.233%	27.99%	44%	8%
		64.705%	-1.938%	26.49%	53%	0%	-1.410%	19.89%	28%	8%	0.058%	22.52%	45%	0%
		70.453%	-0.545%	21.61%	43%	0%	0.252%	18.12%	36%	0%	0.252%	18.12%	36%	0%
		78.205%	-1.445%	1.57%	3%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%
	8	57.545%	-0.078%	22.05%	44%	0%	0.160%	21.67%	43%	0%	-0.325%	22.10%	44%	0%
		64.705%	-1.645%	16.81%	34%	0%	-0.297%	17.26%	35%	0%	-0.297%	17.26%	35%	0%
		70.453%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%
	10	57.545%	-0.008%	18.06%	36%	0%	1.395%	17.21%	34%	0%	-0.258%	17.65%	35%	0%
		64.705%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%
	16	57.545%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%

Table 3.15: MobileNet results.

Fine-fx:*full*			Calibration-1k				Calibration-5k				Calibration-10k			
# Operations	L	Target Accuracy	ΔAcc.	Op. Savings	8×4	4×4	ΔAcc.	Op. Savings	8×4	4×4	ΔAcc.	Op. Savings	8×4	4×4
1269M	4	69.748%	-0.573%	38.60%	58%	13%	-2.353%	40.95%	63%	13%	-0.105%	41.70%	64%	13%
		75.707%	-1.165%	30.12%	60%	0%	-1.047%	24.25%	49%	0%	-0.245%	31.72%	64%	0%
		79.750%	-0.002%	30.04%	56%	3%	-0.562%	22.70%	36%	6%	0.375%	22.70%	36%	6%
		85.368%	1.325%	23.59%	47%	0%	-0.903%	26.57%	48%	3%	0.200%	23.42%	47%	0%
		96.052%	-0.635%	9.21%	19%	0%	-0.358%	5.66%	11%	0%	-0.177%	4.90%	10%	0%
	6	69.748%	-3.573%	21.34%	43%	0%	-1.240%	21.26%	40%	2%	-0.845%	19.64%	40%	0%
		75.707%	-2.790%	16.32%	32%	0%	-1.520%	14.48%	29%	0%	0.267%	16.17%	33%	0%
		79.750%	-1.445%	13.73%	28%	0%	-1.455%	13.38%	27%	0%	0.157%	14.28%	29%	0%
		85.368%	-1.660%	6.09%	12%	0%	-0.118%	3.37%	7%	0%	-0.008%	3.18%	6%	0%
	8	69.748%	-2.900%	15.93%	32%	0%	-0.555%	16.19%	33%	0%	-0.698%	13.73%	28%	0%
		75.707%	-1.655%	12.78%	26%	0%	-0.972%	11.06%	22%	0%	-0.252%	9.33%	19%	0%
		79.750%	-0.608%	4.87%	10%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%
	10	69.748%	-2.980%	14.28%	29%	0%	-1.625%	11.06%	22%	0%	-0.653%	11.80%	24%	0%
		75.707%	-0.917%	4.87%	10%	0%	-0.552%	3.92%	8%	0%	0.000%	0.00%	0%	0%
	16	69.748%	-1.168%	6.31%	13%	0%	0.000%	0.00%	0%	0%	0.000%	0.00%	0%	0%

Chapter 4

Power-Driven Optimization

Due to the increasing integration density and the growing performances of modern System-on-Chips, today power consumption is a major concern in the design of digital circuits. High power consumption raises several challenges, spanning from packaging design, system reliability, and battery lifetime [112]. The problem is well established in the literature of embedded systems, but it gets critical especially in deep learning applications. Indeed, deep learning models are highly parallel workloads that maximize the utilization of the available resources generating a rapid increase in the active power consumption.

Existing algorithmic solutions for the design of portable ConvNets fails to tackle power minimization efficiently [113]. Instead, the problem calls for novel solutions at the hardware-level. At design-time, with the development of dedicated architectures for deep learning acceleration. Intuitively, custom designs tailored to run specific workloads can achieve higher performance at lower power consumption than general-purpose cores at the cost of lower flexibility. At run-time, playing with hardware knobs for dynamic power management. Among them, Dynamic Voltage Frequency Scaling (DVFS) is a de-facto standard in modern digital systems. First, DVFS enables flexible power-performance trade-offs at run-time. Second, it is an effective knob to control the thermal stability of the system. Clearly, a joint cooperation between architectural optimization and power management further improve the overall efficiency. In this context, the analysis conducted in [114] demonstrated that considering DVFS at the early stages of the design of a deep learning accelerator enables to identify the most efficient architectural configurations.

To push the efficiency of DVFS beyond its theoretical limits, we present *FINE-VH*, a novel power distribution scheme for custom multi-processor System-on-Chips. The underlying principle of the proposed strategy is to bring the voltage control at an ultra-fine spatial granularity, i.e. within the functional units. In

the first part of this chapter, we report the implementation details and the collected simulation results. Specifically, we present a dedicated back-end flow that guarantees design convergence with minimum delay and area overhead.

However, in other use-cases, the high implementation costs prevent the adoption of custom designs. In these cases, general-purpose embedded CPUs are a valid alternative offering several advantages. Besides limited cost, embedded CPUs have a more flexible programmability. For instance, they can drive different peripherals such as sensors, actuators, and displays, i.e. they can collect data and make decisions autonomously. Being already integrated into many embedded platforms, they can become intelligent end-nodes through a simple software update. Despite the increasing number of software solutions to run ConvNets on embedded CPUs [25], most of the existing tools focus on performance optimization at nominal operating conditions, neglecting the strict power and thermal constraints of the hosting hardware. Meeting these constraints prevents the execution of intensive workloads (e.g. inference) at maximum performance for long runtime. This problem is addressed in the second part of this chapter, which presents a performance assessment of embedded ConvNets under thermal management. Our study covers the behavior of two control policies, namely reactive and proactive, implemented through the DVFS mechanism available on commercial embedded CPUs. The experimental results, collected on real hardware, guide neural network designers towards a proper understanding of the power constraints, suggesting novel directions for a thermal-aware optimization of ConvNets.

The content presented in this chapter is a revised version of our previous publications found in [115, 116, 117, 118].

4.1 Power optimization on ASICs: FINE-VH

With the twilight of the Moore's law era and the end of Dennard scaling, power consumption has become the main obstacle to the growth of digital System-on-Chips (SoCs) [119]. To tackle this challenge, the multi-core/many-core design paradigm emerged as a promising solution that guarantees high-performance under low power budgets. Indeed, the availability of multiple cores enables to modulate the performance and the number of active units depending on the timing constraints and/or the complexity of workload. Following this trend, modern deep learning accelerators are built upon highly parallel designs composed of arrays of interconnected processing elements [28]. In this context, Dynamic Voltage Frequency Scaling (DVFS) represents one of the most efficient strategy to minimize energy consumption. To achieve this goal, DVFS scales the supply voltage (Vdd) down to the minimum threshold meeting the frequency constraint (f_{clk}) imposed by the actual workload.

Originally, DVFS was conceived for "monolithic" SoCs, but the development of multi-processor SoCs (MP-SoCs) architectures [120] introduced an additional degree of freedom, i.e. the opportunity to set each processing unit at a different operating point [f_{clk}, Vdd]. This core-based, i.e. fine-grained, DVFS implementation allows to run parallel tasks asynchronously thereby reducing the minimum energy point of the entire SoC. One of the most representative application of fine-grain DVFS on a massively parallel architecture is reported in [121], in which 167 processors are orchestrated over a wide voltage-frequency range, achieving ultra-low power consumption, from 47.5 mW at 66 MHz-1.2 V to 608 µW at 66 MHz-0.675 V. Moreover, fine-grain DVFS represents a perfect knob to control and mitigate Process, Voltage, and Temperature (PVT) variations that might affect different cores after fabrication and during the lifetime of the circuit [122].

A practical implementation of DVFS on MP-SoCs makes use of programmable on-chip DC/DC converters that enable to change the supply voltage with a fine resolution step and fast swing (see Fig. 4.1-a). Unfortunately, the integration of DC/DC converters is not a practical solution. Indeed, DC/DC converters have an high implementation costs, since their basic components like capacitors and inductors have low integration density and therefore might occupy a large silicon area. These overheads are not sustainable in fine-grain DVFS, where a dedicated converter should be integrated within each single core.

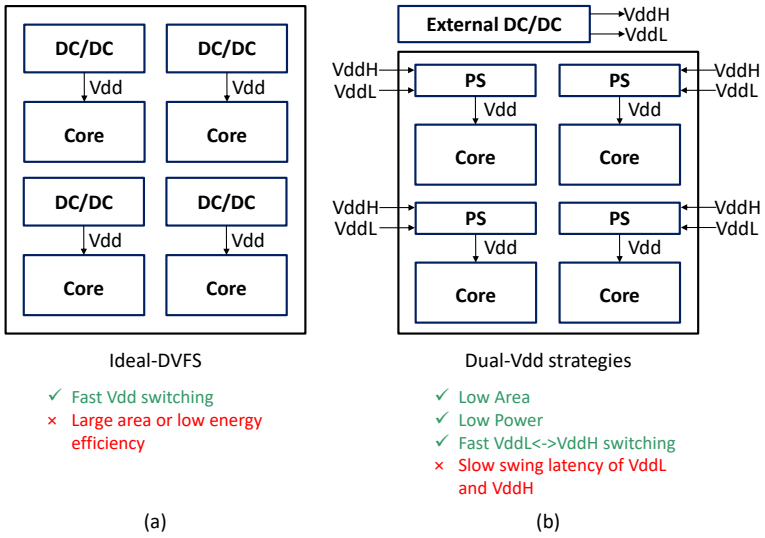


Figure 4.1: Schematic view ideal-DVFS and dual-Vdd power management.

For this reason, previous works tried to achieve, or at least get close to, the efficiency of high-resolution DVFS (ideal-DVFS hereafter), using a discrete set of supply voltages. In discrete DVFS strategies, two Vdd levels (VddL and VddH) are generated off-chip by an external DC/DC converter and evenly distributed

across the chip (see Fig. 4.1-b). The values of VddL and VddH are scaled up/down depending on the timing constraints, while the Vdd of each core is selected through dedicated power switches (PS). Although this design strategy minimizes the impact on area and power, it generates a speed penalty due to the slow voltage swing of external DC/DC converters [123]. However, this is an acceptable cost as in typical applications the voltage-frequency scaling process applies at a low rate.

The two most representative implementations of discrete DVFS are the Vdd-Hopping [124] and the Vdd-Dithering [125]. In Vdd-Hopping, the supply voltage range is split into a discrete set of values, two ore more depending on the external voltage regulator; at run-time the minimum Vdd meeting the frequency constraint is selected. In Vdd-Dithering, the Vdd switches from VddL to VddH implementing a Vdd time-sharing scheme in which the average frequency equals the frequency constraint. As described later, Vdd-Dithering enables to obtain a power-frequency curve that is a linear approximation of ideal-DVFS.

In contrast to existing techniques, which try to get close to the power efficiency of ideal-DVFS, we propose a novel solution that pushes power consumption beyond such theoretical limit. Recalling the strategies introduced in [126] and elaborated in [116], in this section we present *FINE-VH*, a power distribution scheme based on the application of the Vdd-Hopping scheme at a ultra-fine granularity, i.e. within-the-core. First, we conducted a comprehensive parametric analysis demonstrating that FINE-VH outperforms ideal-DVFS. Second, we applied the proposed scheme to a custom design for deep learning acceleration, showing the efficiency of FINE-VH in practical use-cases.

The implementation of FINE-VH poses several challenges, especially when the objective is to devise an automated design methodology and not a handcrafted design. Indeed, the generation of fine-grained voltage domains within the same functional unit raises several concerns during the place&route flow, e.g. area overhead and timing closure due to layout fragmentation and standard cell displacement. Furthermore, static power consumption increases due to leakage currents among logic gates belonging to voltage domains supplied at different Vdd. For instance, in a simple chain of two inverters, where the driven inverter is powered at nominal Vdd, its leakage power increases up to 5.2× if its driver is powered at 90%Vdd, 22.1× at 80%Vdd. Clearly, at this level of granularity the insertion of voltage-level shifters would imply huge design overheads and therefore is an impractical solution.

To address these needs, we describe a fully-integrated design flow involving incremental re-synthesis stages to guarantee fast timing/power convergence. Specifically, we leverage an optimal poly-bias assignment strategy that reduces intra-domain leakage currents without area/delay costs.

First, we validated FINE-VH on a simple RISC-V core, the *RI5CY*, embedded

in the ultra-low power multi-processor platform *PULP* [127]. On this benchmark, we conduct an accurate design space exploration which assesses different figures of merit, like power, area, and delay, across different degrees of granularity, using a layout partitioned into 9, 25, and 49 domains, and different Vdd pairs, i.e. multiple values of ΔVdd=VddH-VddL. Second, we tested FINE-VH on a larger design, the processing element integrated into the *LoC-1* deep learning accelerator [117], demonstrating the scalability of our strategy. For the synthesis, we adopted a cutting-edge industrial Fully-Depleted SOI (FDSOI) CMOS technology at 28 nm. Overall, the power-performance trade-offs achieved by FINE-VH outperform state-of-the-art DVFS solutions: ideal-DVFS, Vdd-Hopping, and Vdd-Dithering.

4.1.1 Background

Towards Ideal-DVFS

Implementing ideal-DVFS with the insertion of programmable on-chip DC/DC is not a viable option, as the huge area and power overhead would nullify the savings. Two valid alternatives that guarantee reasonable implementation costs are Vdd-Hopping [124] and Vdd-Dithering [125]. An abstract view of their working principle is depicted in Fig. 4.2. Whereas in ideal-DVFS (dashed line in Fig. 4.2) the supply voltage can be adjusted with a very high resolution, such techniques employs a discrete set of supply voltages.

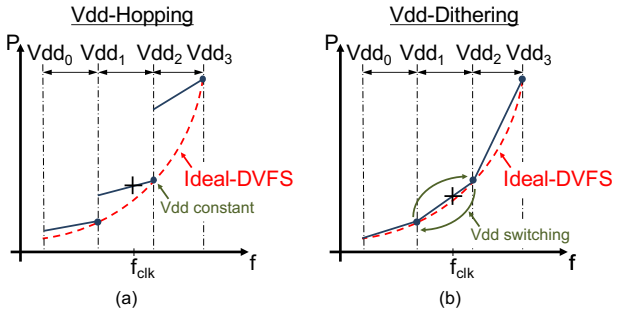


Figure 4.2: Performance-Power trade-off curves of existing DVFS schemes.

Vdd-Hopping: as shown in Fig. 4.2-a, in this scheme the supply voltage range is split into a specific set of intervals, three in the plots of the figure. Among the avaialble options, the proper voltage is selected at run-time in order to meet the frequency constraint (f_{clk}) imposed by the actual workload. Once f_{clk} is identified, the core is powered at the Vdd at the right edge of the interval including f_{clk} (Vdd_2 in Fig. 4.2-a). Within each interval, the Vdd is kept constant and the power consumption decreases linearly with f_{clk}. When f_{clk} crosses a new interval, power

scales accordingly with the new Vdd. Clearly, the power efficiency Vdd-Hopping drifts from ideal-DVFS as f_{clk} approaches the left edge of each interval.

Vdd-Dithering: differently from Vdd-Hopping, this method leverage a Vdd time-sharing scheme (Fig. 4.2-b). At the application level, the core works at low Vdd (Vdd_1 in Fig. 4.2-b) for the first portion of a task, then switches to high Vdd (Vdd_2) for the last portion. Given T_{low} as the time spent at low Vdd, i.e. low frequency f_{low}, and T_{high} as the time spent at high Vdd, i.e. high frequency f_{high}, the core operates at an average frequency (f_{avg}) proportional to the "switching ratio":

$$f_{avg} \propto \frac{(f_{low} \cdot T_{low}) + (f_{high} \cdot T_{high})}{T_{low} + T_{high}} \quad . \tag{4.1}$$

Modulating the ratio between T_{low} and T_{high} enables to center the target frequency f_{clk}. The physical implementation reported in [125] demonstrated that the overhead due to the non-ideality of power-switches are negligible, therefore the trend line of Fig. 4.2-b represents a realistic approximation of the power efficiency achieved by Vdd-Dithering.

Within-the-core power management

Following the natural scaling of power management experienced in the last years, FINE-VH brings the Vdd-Hopping concept at a finer level of granularity, i.e. within the core. In this section, we reviews this scaling trend through a classification of the possible granularity options. The taxonomy tree is shown Fig. 4.3.

Our objective is to bring the Vdd-Hopping concept at a finer level of granularity, i.e., within the core. The underlying idea is not new as it follows the natural scaling of other power-management experienced in the last years, e.g., Multi-Vdd, Body-Biasing, Power-Gating [128]. The taxonomy tree shown in Fig. 4.3 provides a brief classification of the possible granularity options.

In the beginning, low-power knobs like Multi-Vdd, Body-Biasing, or Power-Gating [128], were applied at the architectural level, where the grain was an entire *functional block* (Fig. 4.3-a). For instance, different pipeline stages can be dynamically fed by VddH and VddL, using an interpolation scheme similar to Vdd-Dithering [129]; together with a variable latency mechanism, this strategy enable to minimize the energy overhead due to process-variations. An extension of the scheme is presented in [130] to further improve the reliability of the system by means of dynamic local error detection&correction units for preventing timing violations.

Whereas operating at the architectural level is straightforward, the power efficiency is limited by the grain dimension. All the low-power knobs exploit an intrinsic feature of digital circuits, i.e. idleness. When a functional block is not

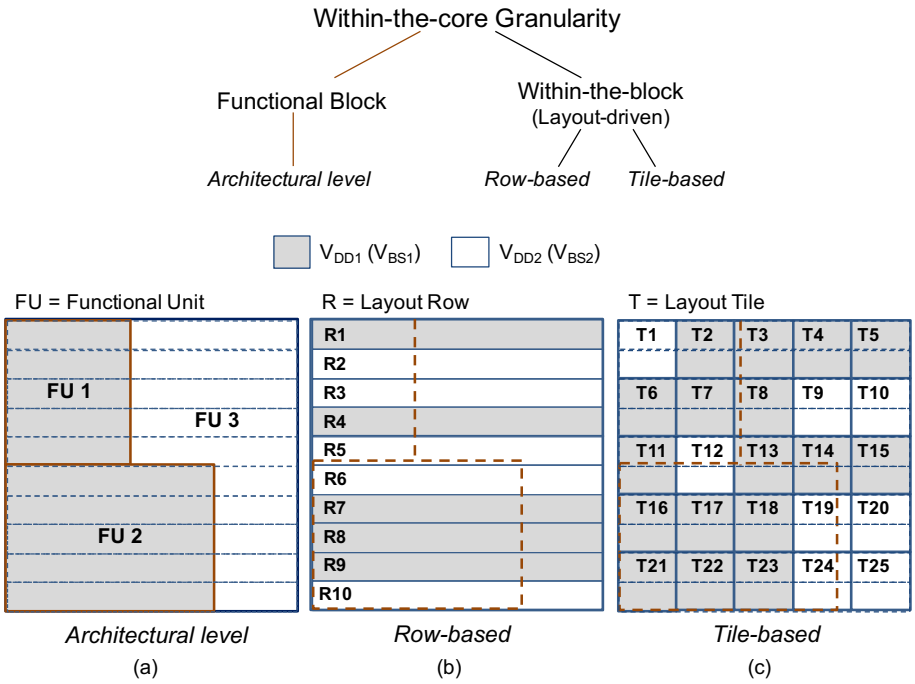


Figure 4.3: Classification of low-power knobs granularity. From coarse-grained architectural level (a), up to fine grained solutions like row-based (b) and tile-based (c) partitioning.

used or can operate at lower speed, its power can be reduced without affecting the performance of the overall system. Obviously, this implies the identification of idle functional blocks and active ones. However, there is still room for improvement, as large portions of each functional block might include non-critical components, e.g. standard cells belonging to short logic paths. Applying low-power knobs on those components can introduce additional optimization margins without any performance cost. Inspired by this intuition, power management underwent a development process towards the application of low-power knobs at a finer granularity, i.e. *within-the-block*.

Unfortunately, pushing low-power knobs at a fine granularity poses several concerns, because the row-based placement of a traditional physical design flow imposes strict geometrical constraints that limit the minimum size of the grain. This calls for a proper understanding of the physical constraints, forcing a regular partitioning of the layout. In this regard, the most common layout-driven solutions are *row-based* and *tile-based* partitioning.

In row-based partitioning (Fig. 4.3-b), the grain is the a single layout row. For instance, a row-based dual-Vdd assignment can be adopted for process variation

compensation [131]. Specifically, timing critical rows are grouped in high-Vdd clusters, whereas the non-critical ones are assigned to low Vdd. A dedicated timing-driven optimization places critical cells on adjacent rows to minimize leakage currents at the interfaces between rows. However, the Vdd assignment is statically done at design-time, preventing the implementation of adaptive power schemes. Similarly, [132] presents a row-based partitioning for ultra-fine grain body-biasing. In contrast to [131], each partition includes the same number of row, regardless the timing criticality of logic paths. The latter solution offers more flexibility, as it enables adaptive body-biasing tuning after fabrication to compensate process variations.

In tile-based partitioning, the grain is a regular portion of the layout. A pictorial example is reported in Fig. 4.3-c, which illustrate a layout arranged in 5x5 square mesh. The seminal work of [133] presented an adaptive dual-Vdd strategy applied on a cryptographic accelerator partitioned into 42 square tiles. Even though results were promising, the power savings are limited to 12% (with respect to a monolithic DVFS) due to static power overheads generated by intra-tile leakage currents. An additional example of tile-based partitioning is reported in [134], yet with a different objective; it supplies at high Vdd those tiles including standard cells whose electrical behavior needs a minimum operating voltage larger than most of the other cells. This guarantees fault-free operation, even though on average the circuit operates at Vdd$<$ Vdd_{min}.

Following the path of previous works, the FINE-VH strategy adopts a tile-based partitioning. However, the aim of FINE-VH differs, as the idea of bringing Vdd-Hopping at an ultra-fine granularity to push DVFS beyond its theoretical limits has not been explored yet.

4.1.2 Implementing FINE-VH

Design and Optimization

A practical implementation of FINE-VH requires a proper management of the design concerns generated by the layout fragmentation of the circuit. First, the cell displacement due to the partitioning could generates area and timing overheads. Second, static power consumption could increase due to leakage currents of standard cells powered at different Vdd. To address these issues, we devised a computer-aided design methodology integrated in a standard place&route flow.

In this section, we first details the layout partitioning procedure and the physical design steps needed to implement FINE-VH. Second, we introduce an optimal poly-bias assignment strategy aiming to mitigate intra-tile leakage at no timing penalty. Finally, we present a simulation/emulation procedure for optimal Vdd selection.

Physical Design

The FINE-VH strategy adopts a tile-based partitioning, schematically depicted in Fig. 4.4. The layout is partitioned in regular mesh of NxN square tiles (N=3 in the figure), each of them fed with dual-Vdd, i.e., low-Vdd (VddL) and high-Vdd (VddH), brought by the around-the-core power-rings. The two supply voltages are generated by external DC/DC converters and their values depend from the target frequency fixed at the application level (for instance, in Fig. 4.2, VddL=Vdd_1 and VddH=Vdd_2). Overall the power distribution network includes five power-grids running around the core through upper-metal horizontal/vertical stripes: VddH, VddL, Gnd, Vbn (n-bias), and Vbp (p-bias). Note that this scheme is compliant with adaptive/dynamic body-biasing (out of the scope of this work).

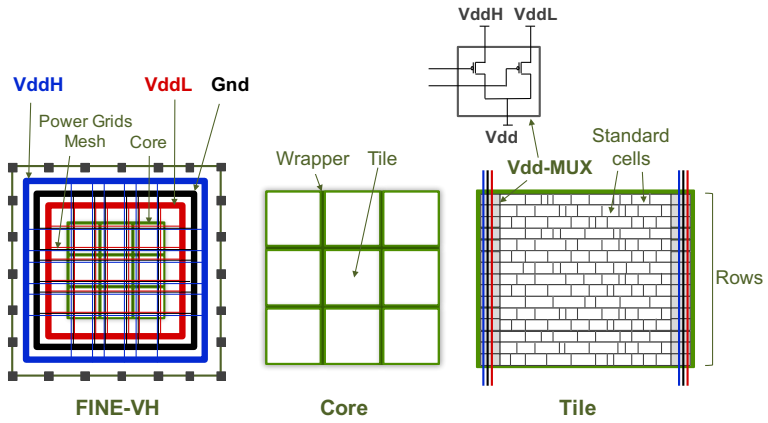


Figure 4.4: Tile-based partitioning and tile organization

The Vdd-selection is made possible by p-type header power switches that tie the layout rows to the power-grids. These switches are integrated into custom cells, called *Vdd-MUX*. In practice, the power-management unit is in charge of the Vdd-selection, loading the Vdd configuration bit-stream into dedicated chain of flip-flops that drives the Vdd-MUXes. The placement of the Vdd-MUXes follows a row-based insertion scheme [135], ensuring a uniform distribution across the tile. The power-grids run over the Vdd-MUX columns, therefore simple vertical vias connect VddL and VddH to the Vdd-MUX cells.

Adjacent tiles are isolated by a void-space wrapper that break the continuity among the lower-metal power rails. The wrapper is needed since adjacent tiles might have a different Vdd. The wrapper width should guarantee the minimum metal-to-metal distance, which is a design constraints defined by the technology adopted.

The key feature of FINE-VH is that the tile-based partitioning follows a "no-look" style. Once the grain size is fixed through the design parameter N, the layout is partitioned at the floorplanning stage, before the placement of the functional

blocks of the core. One might wonder if this choice could generate overhead as functional blocks are split across multiple tiles. However, this represents the only viable solution that guarantees (i) a regular power distribution netowork and (ii) the convergence of a commercial placer to a fast timing closure.

From a practical viewpoint, the FINE-VH flow encompasses six different stages fully integrated into a commercial design platform by *Synopsys*® by mean of dedicated TCL procedures:

1. **Synthesis**: logic synthesis using technology libraries characterized at the maximum Vdd, e.g., 1.0V for our 28 nm technology.

2. **Floorplanning**: estimation of the core area and creation of an empty layout; the latter is then automatically partitioned into NxN regular tiles using placement blockages.

3. **PG-Synthesis**: power-grids are synthesized following a regular mesh over the partitioned layout.

4. **Placement**: the Vdd-MUXes are placed at the boundaries of the tiles while standard cells are placed within the tiles so that timing constraints are satisfied.

5. **Post-Placement leakage optimization**: a re-synthesis stage performing optimal poly-bias assignment for those cells at the interface of the tiles (additional details in the next subsection).

6. **Routing**: a standard timing-driven routing for logic signals.

Intra-tile leakage power reduction via Poly-Bias optimization

In a FINE-VH design, static power might increase due to larger leakage currents between the *interface-cells*, i.e. cells driven by signals coming from other tiles. When an interface-cell receives an input signal with a voltage lower than its Vdd, its internal pull-up network remains partially turned-ON, thus increasing leakage currents (see Fig. 4.5-a).

To mitigate this overhead, it is possible to increase to p-MOS threshold voltage (V_{th}) of the interface-cells. Two practical solutions to achieve this goal are gate length modulation [136] and high-V_{th} transistors [137]. The FDSOI CMOS technology, adopted in this work, provides multi-V_{th} libraries obtained with former technique, which is also referred to as *poly-biasing* (PB) and shown in Fig. 4.5-b. Specifically, each logic gate is available in four different versions: PB0 (the standard V_{th}), PB4, PB10 and PB16 (the highest V_{th}).

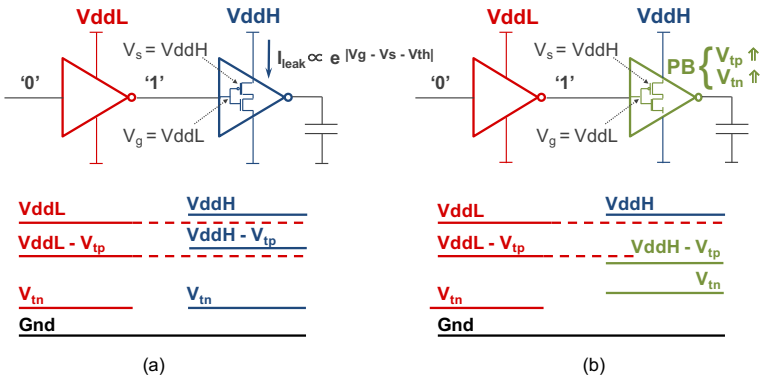


Figure 4.5: Intra-tile leakage current (a) and its mitigation via poly-biasing (b)

As the Vdd selection is done at run-time, identifying the interface-cells affected by intra-tile leakage is not feasible. Needless to say, a conservative strategy in which all the interface-cells are mapped to high-V_{th} would generate excessive delay overhead. For this reason, we present a timing-driven post-placement poly-bias assignment which works as illustrated in Fig. 4.6. Starting from a placed netlist of standard V_{th} cells, i.e., PB0, the interface-cells are first identified (a) and then virtually isolated in a separated netlist with back-annotated delay information (b). Using the optimization engine embedded into the physical synthesizer, a timing-driven multi-PB assignment is run (c). The netlist returned by the multi-PB assignment step has the minimum leakage configuration, i.e., the largest set of high-V_{th} cells, that satisfies the delay constraints. Finally, the resulting PB assignment is annotated into the main netlist (d).

The actual leakage optimization takes place in step (c). To avoid delay penalties, the optimization involves only those interface cells belonging to logic paths with a timing slack grater than certain threshold. Furthermore, to maximize the number of cells with highest V_{th} (PB16), the procedure encompasses three incremental PB-assignment stages. At each stage, only one class of PB cells is considered: at the first stage PB16, at the second stage PB10, at the third stage PB4. In this procedure, PB16 cells (which guarantee the highest protection from intra-tile currents) gets higher priority than the PB10 and PB4 cells (for which intra-tile leakage compensation is minimal)

The slack thresholds adopted for the selection of the interface cells involved in the poly-bias assignment depends on the actual stage of the optimization. For example, at the first stage (PB16 assignment), the slack threshold is larger, as PB16 cells introduces a larger delay overhead than PB10 and PB4 cells. At the last stage (PB4), the slack threshold is relaxed. To determine the exact values of these thresholds, we extracted the *slow-down factor* α_{sd}, i.e. the average delay overhead, induced by each class of PB cells and defined the slack threshold for each stage trough equation (4.2):

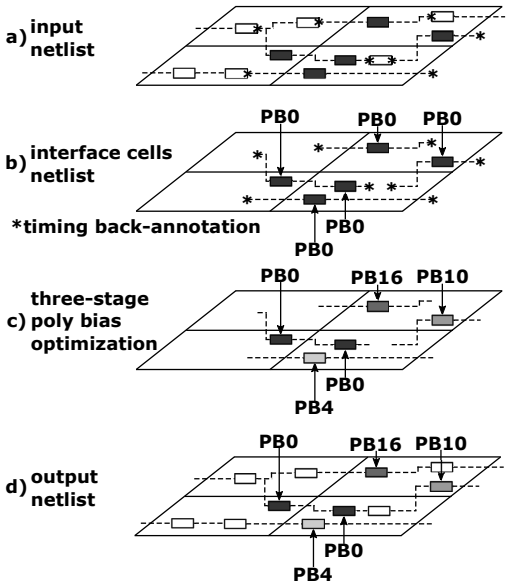


Figure 4.6: Optimal poly-bias assignment through local re-synthesis

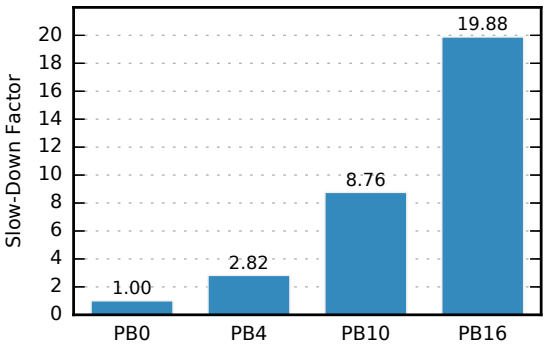


Figure 4.7: Average slow-down factors for the different poly-bias options.

$$S_{\text{th}}(PB) = d_{\text{avg}} \cdot \alpha_{\text{sd}} \ , \quad PB = [4, 10, 16] \ . \tag{4.2}$$

where d_{avg} denotes the average propagation delay of a cell belonging to the critical path. Figure 4.7 reports the collected slow-down factors: 2.82x for PB4, 8.72x for PB10, 19.88x for PB16.

As a final remark, note that the presented multi-stage re-synthesis is suited not only for the 28 nm FDSOI technology but it can be extended for every technology offering multi-V_{th} libraries.

4.1.3 Simulation and Emulation

The assessment of the figure-of-merits of a FINE-VH design requires the development of a dedicated static-analysis engine able to process level-shifter free multi-Vdd designs. For this purpose, we opted for a static estimation of intra-tile leakage currents based on an off-line characterizations. In addition, as FINE-VH exploits a dynamic (i.e. at run time) Vdd-selection, an accurate assessment should include the emulation of power-management policies. Regarding this point, we implemented a simple, yet effective timing-driven Vdd-selection.

Intra-Tile Leakage Power Estimation

In this section, we present a static analysis engine to estimate intra-tile like power avoiding heavy SPICE simulations of the entire core for multiple Vdd-selections. More specifically, we built a look-up table annotating the leakage power derating fcator for all possible input patterns (logic 0/logic 1) and all possible VddL/VddH configurations. Similarly to standard timing libraries, the look-up tables are generated for different voltage corners. The collected look-up tables enable to compute the leakage power employing the same model integrated into commercial tools:

$$P_{\text{leak}} = \sum_{i=1}^{2^n} P_i \cdot L_i \cdot k_i \tag{4.3}$$

where n denotes the number of input pins (the number of pins, for sequential cells), P_i the input pattern probability, L_i the nominal static power extracted from standard timing libraries, and k_i the leakage power derating factor picked from the LUT. Clearly, k_i=1 if the driver cell is placed in a tile sharing the same Vdd of the logic cell under analysis.

Vdd-selection

To emulate power-management at design-time we devised a timing-driven Vdd-selection (pseudo-code in in Algorithm 5). The algorithm estimates the timing criticality of each tile counting the number of cells with negative slack. Tiles with the largest number of critical cells are assigned to VddH first. The procedure is iterated until all the cells show a positive slack.

First, the algorithm assigns all the tiles to VddL (line 1). For each tile a *criticality score* is initialized to zero (line 3). Once the most critical path is extracted, all the cells belonging to that path are stored in a dedicated list called *Cell_List* (line 4). For each cell in *Cell_List*, the propagation delay is extracted and added to the criticality score of the tile hosting the current cell (lines 5 to 8). Once all

Algorithm 5: Vdd-selection Procedure

```
   Input: VddL, VddH, f_clk
   Output: Vdd assignment
1  set_VddL([All_tiles]);
2  while Worst_Slack < 0 do
3  |  zero(Tile_Score[All_tiles]);
4  |  Cell_List ← Cells ∈ Critical Path
5  |  foreach Cell ∈ Cell_List do
6  |  |  Cell_Delay ← propagation delay of Cell;
7  |  |  Tile_ID ← tile hosting Cell;
8  |  |  Tile_Score[Tile_ID] += Cell_Delay;
9  |  end
10 |  Tile_Score ← Tile_Score[Tiles@VddL];
11 |  Tiles_Score ← sort(Tile_Score, decreasing);
12 |  Critical_Tile ← Tile_Score[0];
13 |  set_VddH(Critical_Tile);
14 end
```

the cells in *Cell_List* have been processed, those tiles still supplied at VddL are sorted according to their score (line 10 and 11). The tile with the highest score (line 12) is assigned to VddH (line 13). The procedure ends when the core presents a positive timing slack, i.e. when the target f_{clk} is met (line 2).

4.1.4 Evaluating FINE-VH

In this section, we report experimental results collected on two different benchmarks. The first one is a small design consisting of a tiny general purpose cores used to demonstrate the feasibility of the presented strategy. As a second benchmark, we adopted a processing element integrated in a deep learning accelerator. Compared to the former benchmark, the processing element is a larger design that enables to validate the scalability of FINE-VH.

Validation Benchmarks

We first tested the FINE-VH flow on the RI5CY core, an open-source RISC-V instruction set architecture used in the low-power parallel-processing platform PULP [127]. The core includes the following functional blocks: a prefetch buffer, an instruction decoder, a 31x32 bit register file, an integer ALU, a single-cycle 32x32 integer multiplier, a control status register, hardware loop unit, debug unit, and a load&store unit. Figure 4.8 illustrates the layout of the die after tile partitioning for N=49.

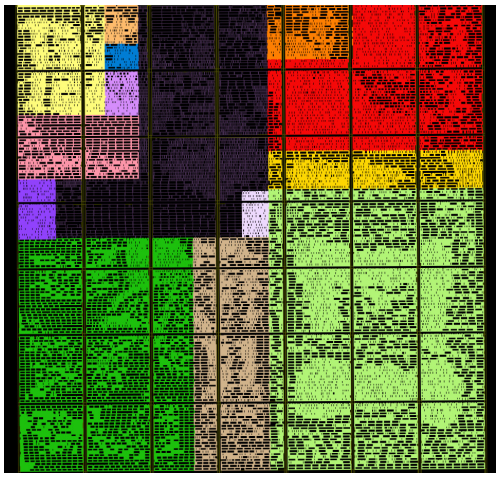


Figure 4.8: Layout partitioning of A RI5CY core after standard-cell placement (49 tiles).

Experimnetal Set-up

For static timing and power analysis, we resorted to the engine integrated in the STA tool by Synopsys (PrimeTime). We used technology libraries provided by the silicon vendor characterized for Vdd ranging from 0.60 V to 1.00 V (step of 50 mV) and worst-case corner (slow-slow and 125 °C).

We performed a comparative analysis between existing DVFS schemes and FINE-VH. The experimental settings is reported as follows.

- **Ideal-DVFS:** for each Vdd, the maximum frequency is extracted and set as the working frequency. The Vdd ranges from 0.60 V up to 1.00 V with a step of 25 mV, resulting in eight (f, Vdd) operating points; for those Vdd not available in the library set, we used a cross-library scaling feature embedded into the STA tool.

- **Vdd-Hopping:** the Vdd range [0.60 V-1.00 V] is split into a finite set of intervals having a fixed width: ΔVdd=200 mV and ΔVdd=100 mV; the resulting Vdd values are 0.60 V, 0.80 V, 1.00 V and 0.60 V, 0.70 V, 0.80 V, 0.90 V, 1.00 V respectively. The Vdd is chosen depending on the target frequency (please refer to Fig. 4.2).

- **Vdd-Dithering:** same Vdd ranges/intervals of the Vdd-Hopping scheme, but power consumption is a liner interpolation of points obtained through ideal-DVFS (please refer to Fig. 4.2).

- **FINE-VH:** the technique proposed in this work. As for the previous schemes, the Vdd ranges is [0.60 V-1.00 V]; two ΔVdd options are explored, i.e., 200 mV

(Vdd range split into two intervals) and 100 mV (Vdd range split into four intervals).

For all the above design strategies the average power is extracted considering realistic switching activities of the primary inputs, i.e., annotating static probabilities and toggle rates extracted from functional simulations.

Results

Table 4.1 reports the figures of merit of the RI5CY core after the FINE-VH flow is applied at different levels of granularity (#Tiles=1 denotes a standard design without FINE-VH). Both the core area and the number of layout rows increase with granularity due to wrapper insertion around the tiles. Such a void space is devoted for de-cap cells insertion and intensive routing. The area overhead ranges from 2.42% for 9 tiles to 5.95% for 49 tiles. The most interesting note is that the active cell area and the nominal delay@1.00 V (i.e., critical path delay when all the tiles are supplied at 1.00 V) keep almost constant; this proves the convergence of the design flow, even at 49 tiles. As one can observe, the percentage of interface cells increases with the number of tiles, together with the intra-tile interconnections. However, as will be shown later in the text, the proposed poly-biasing optimization helps to control the intra-tile leakage overhead.

Table 4.1: Figures of merit of the RI5CY after FINE-VH

# Tiles	**1**	**9**	**25**	**49**
Core Area μm^2	36229	37105	37626	38386
# Rows	158	160	161	163
Cell Area μm^2	25550	25397	25302	25698
Delay@1.00V ns	3.00	2.95	2.99	3.00
Interface Cells	0.0%	38.44%	56.89%	62.65%

Concerning the V_{th} distribution, Fig. 4.9 shows that the PB optimization procedure makes extensive use of cells at the highest V_{th} (PB16), above 57% in all the three configurations. This highlights how the leakage optimization engine embedded into commercial tools well fits FINE-VH purposes when properly instructed.

Figure 4.10 shows the power vs. frequency trade-off curves for a 49-tile FINE-VH configuration and the three state-of-the-art DVFS schemes. Numbers are normalized with respect to an ideal-DVFS implementation (dashed line in the plot) supplied at minimum Vdd. As expected, Vdd-Hopping and Vdd-Dithering do approximate the behavior of ideal-DVFS, even though in a different manner. Within each Vdd interval the Vdd-Hopping gets worse at lower frequencies, while

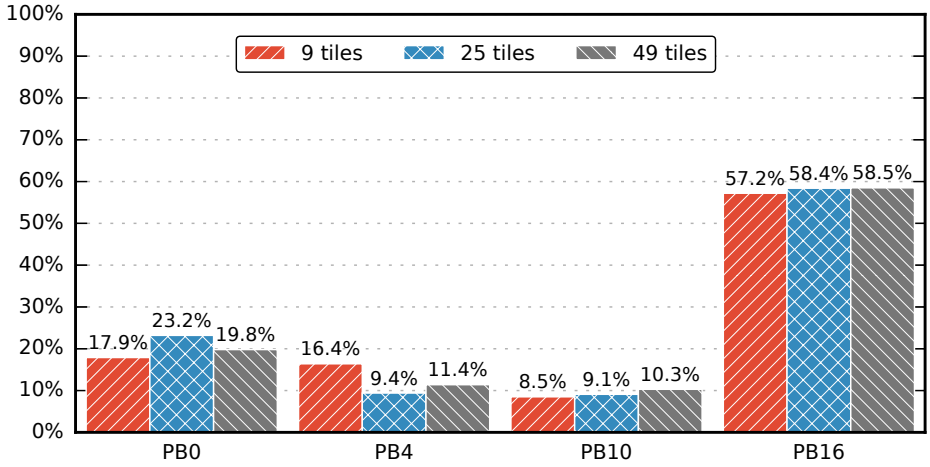


Figure 4.9: Poly-bias distribution across the interface-cells

Vdd-Dithering always runs close to ideal-DVFS. FINE-VH outperforms the competitors for all the operating points, both for ΔVdd=200 mV and ΔVdd=100 mV. When ΔVdd = 200 mV, average power reductions of 43.5% and 27.5% are obtained with respect to Vdd-Hopping and Vdd-Dithering respectively. Moreover, FINE-VH outperforms ideal-DVFS achieving a 23.4% of power savings. The benefits of the proposed technique are even more empathized at ΔVdd=100 mV, where the average power consumption is reduced to 35.3% with respect to Vdd-Dithering and to 34.6% with respect to ideal-DVFS.

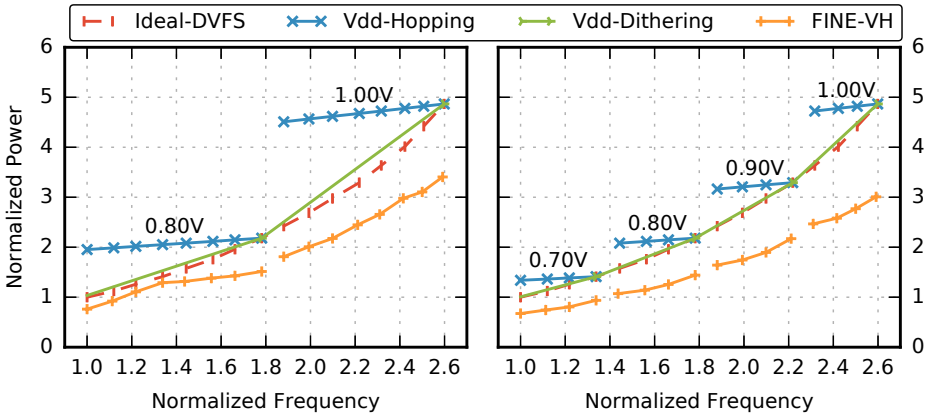


Figure 4.10: Comparative analysis among four DVFS schemes: i) ideal-DVFS, ii) Vdd-Hopping, iii) Vdd-Dithering, iv) FINE-VH (49 tiles); ΔVdd=200 mV (left), ΔVdd=100 mV (right).

The power savings for each operating point are detailed through Fig. 4.11

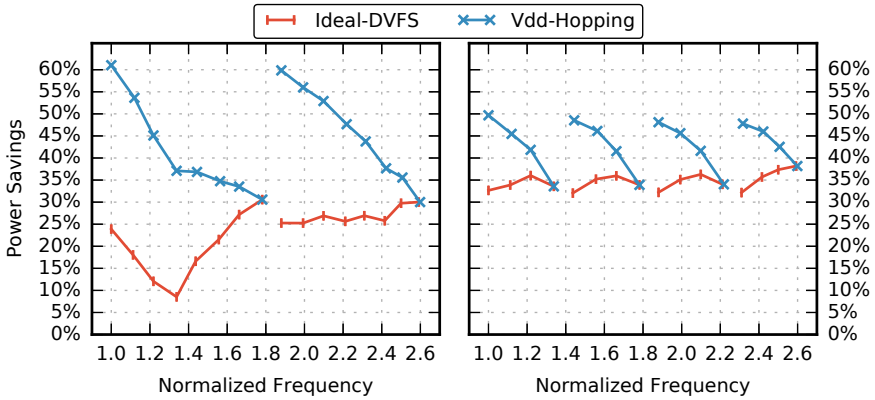


Figure 4.11: Power savings of the proposed FINE-VH (49 tiles) with respect to ideal-DVFS and Vdd-Hopping; ΔVdd=200 mV (left), ΔVdd=100 mV (right).

(the plot does not show savings with respect to Vdd-Dithering as they are close to ideal-DVFS). For ΔVdd=200 mV savings range from 8.5% to 30.6% with respect to ideal-DVFS and from 30.0% to 61.0% with respect to Vdd-Hopping. For ΔVdd= 100 mV power savings range from 32.0% to 38.2% with respect to ideal-DVFS and from 33.6% to 49.7% with respect to Vdd-Hopping. When considering a direct comparison to Vdd-Hopping, larger savings are achieved at lower operating frequencies (left side of each Vdd interval), where a finer granularity allows to supply more portions of the layout at the low voltage. Vdd-Hopping, instead, forces the core running at a Vdd that is quite far from the optimal one.

Working with ΔVdd=100 mV brings larger average savings (with respect to ideal-DVFS) for two main reasons: *(i)* a finer voltage granularity allows a better selection of tiles that can be set at VddL, hence, it helps to get closer the global optimal; *(ii)* a smaller ΔVdd guarantees better noise margins while mitigating the effects of intra-tile leakage. This would suggest that a smaller ΔVdd is a better design option. That's true as long as power/delay overheads of external voltage regulators are neglected. Indeed, when the target frequency imposed at the application level is shifted outside the reference interval, VddL and VddH need to be set to different values (please refer to Fig. 4.2); this implies some extra power overheads and additional latencies due to off-chip DC/DC converters. Intuitively, a lower number of intervals, i.e., a larger ΔVdd, may allow to cover a larger set of target frequencies with the same pair of values for VddL and VddH. In other words, a larger ΔVdd reduces the probability of an external voltage shift. The choice of an optimal ΔVdd is a trade-off imposed by design specifications.

Figure 4.12 shows the percentage of the cell area supplied at low Vdd when FINE-VH is applied at different granularity, i.e., 9, 25 and 49 tiles. The plot clearly shows that 49 tiles give the best savings. For both the two ΔVdd options, working at higher frequencies decreases the amount of silicon area powered at low Vdd.

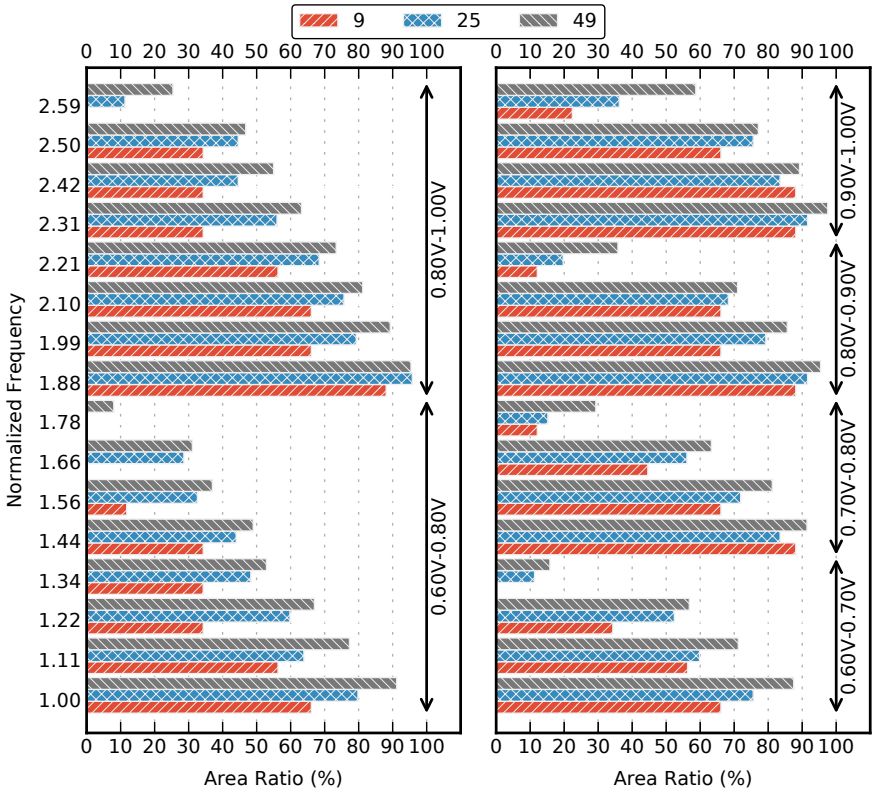


Figure 4.12: Percentage of standard cell area @VddL for different number of tiles. For instance, at the maximum frequency, $f_{\text{clk}} = 2.59$, with 9 tiles, the percentage of area at low Vdd drastically reduces to zero for ΔVdd=200 mV, and to 2.4% for ΔVdd=100 mV. A lower granularity implies larger tiles that, most probably, will contain at least one timing critical logic path that forces the selection of a high Vdd. This issue is progressively mitigated as the granularity gets finer. As one can observe, at the maximum frequency $f_{\text{clk}} = 2.59$, with ΔVdd=200 mV, the percentage of area powered at low Vdd increases to 11.3% at 25 tiles and 25.4% at 49 tiles. Savings are even larger when ΔVdd=100 mV, 36.2% at 25 tiles and 58.7% for 49 tiles. The advantage of a finer granularity can be better appreciated considering the average over the whole frequency spectrum: for ΔVdd=200 mV, the average percentage increases to 38.5% (9 tiles), 52.0% (25 tiles), 58.9% (49 tiles); for ΔVdd=100 mV, the average percentage increases to 54.0% (9 tile), 60.7% (25 tiles), 69.2% (49 tiles). This confirms once again the rule of thumb: "the finer, the better".

Figure 4.13 shows the power savings (with respect to ideal-DVFS) achieved with the proposed PB optimization. For ΔVdd=200 mV, do not using poly-biasing nullifies all the savings brought by FINE-VH, i.e., negative savings. On the contrary, the PB assignment helps recovering the overheads due to a level-shifter free

strategy as multi-V_{th} cells substantially reduce the intra-tile leakage and are intrinsically less leaky. Our PB strategy achieves average savings of 23.4%. For ΔVdd=100 mV, though the overhead imposed by intra-tile leakage currents is smaller, our PB optimization allows larger savings, 34.6% (PB) against 10.8% (no PB), without any performance penalty.

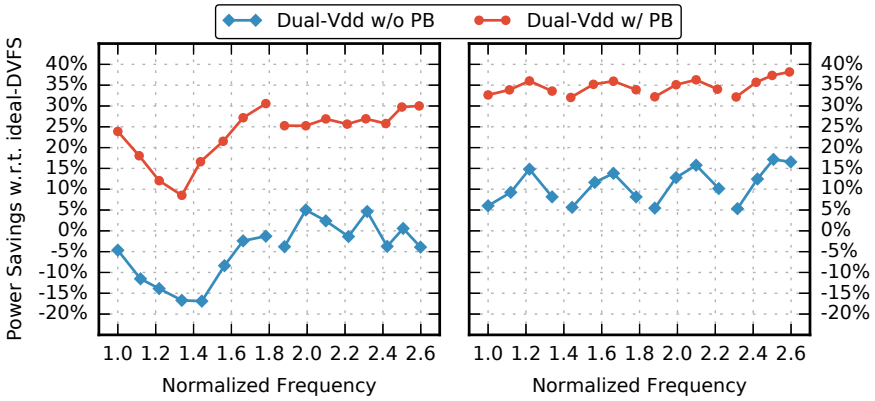


Figure 4.13: Power savings with respect to ideal-DVFS before and after PB optimization for ΔVdd=100 mV and ΔVdd=200 mV (49 tiles)

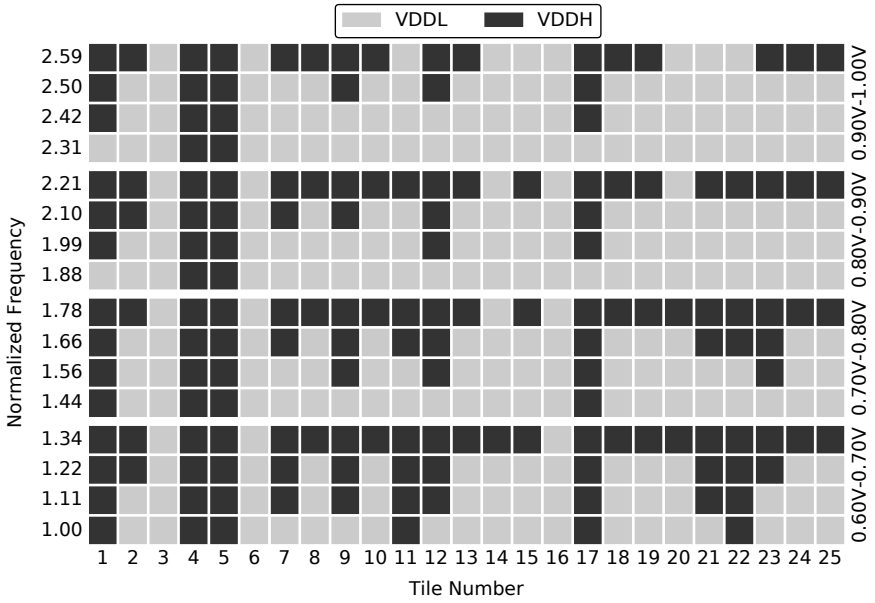


Figure 4.14: Voltage Assignment (25 tiles)

Finally, Fig. 4.14 reports the Vdd selection obtained through Algorithm 5 for

the case ΔVdd=100 mV. The plot refers to 25-tile configuration. For each operating frequency, it shows the Vdd assigned to each tile. When the frequency increases, the number of tiles supplied at low Vdd decreases. However, some tiles, like the #4 and the #5, are more critical than others as they are constantly fed by VddH; other tiles are less critical, like tile #3, #6, #16, and, even at higher frequencies, they still keep running at low Vdd. The most critical functional blocks are the arithmetic units, for which at least one tile is always supplied at high Vdd. The proof that the Vdd-assignment algorithm detects the most timing-critical tiles can be inferred by observing the regularity of the Vdd distribution within each individual voltage interval: once a critical tile is assigned to VddH it never swaps to VddL. This holds for tile #7, #8, #9, #10, #12.

Case Study: FINE-VH on deep learning accelerators

To assess the benefits of FINE-VH on a deep learning accelerator, we applied our design flow to synthesize the processing element (PE) of the *LoC-1* architecture [117] The *LoC-1* is a low-power micro-programmable architecture hosting a 2D array of identical processing elements. The PE is the basic block of the accelerator. It consists of a SIMD-like (Single Instruction Multiple Data) unit integrating four parallel lanes. The lanes have been conceived for flexibility and high performance. Indeed, their programmable architecture supports all the main operations (convolution, pooling, non-linearity) found in DL algorithms and can be dynamically resized to match different input feature-map and kernel size. The flow of operations is managed by a local control unit. Each lane has local memory and an execution unit. The local memory is divided in two banks, a private and a shared bank; the former is only locally accessible whereas the latter can be accessed also by other lanes. The execution unit contains a multiply-and-accumulate unit whose adder can also be used independently. The PE also hosts a network-on-chip router that dispatches data and instructions from/to other PEs. We refer the interested reader to [117] for a detailed description at the architectural level of the PE and the *LoC-1* accelerator.

The PE was partitioned with 400 tiles and was synthesized with f_{clk}=1.3 GHz @ 1.0 V using a CMOS 28 nm FDSOI technology. After the place&route stage, we measured a core area of 101 362 µm^2. The design was fully characterized in terms of power and performance. Fig. 4.15 shows a comparison between a core-level Vdd-Hopping and FINE-VH. The plot reports the normalized power consumption for the two schemes with 16 frequencies and 4 voltage intervals. FINE-VH outperforms Vdd-Hopping, resulting up to 20% power savings. These findings confirm the efficacy of the proposed strategy.

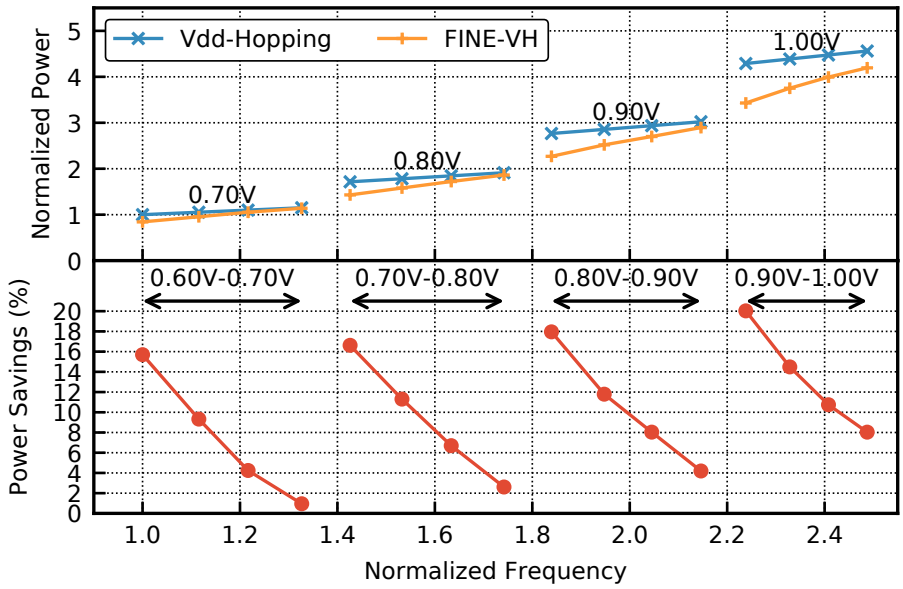


Figure 4.15: Power comparison between Vdd-Hopping and FINE-VH for a PE partitioned into 400 tiles (after place&route).

4.1.5 Discussion

Ultra-Fine Grain Vdd-Hopping (FINE-VH) improves the efficiency of state-of-the-art DVFS schemes. We implemented a fully automated layout-assisted, level-shifter free flow which enables tile-based Vdd-Hopping at ultra-fine granularity with minimum design overheads. The proposed technique includes a timing-driven incremental re-synthesis stage for optimal poly-biasing assignment addressing the intra-tile leakage power reduction without incurring in delay degradation. FINE-VH was first validated on a RISC-V core for MP-SoC applications and then experimented on a processing element for deep learning acceleration. Both designs were mapped onto a commercial 28 nm FDSOI technology. To measure the benefits of the proposed technique, we devised an experimental framework capable of emulating the Vdd-selection at different target clock frequencies and estimating the intra-tile leakage power.

Experimental results shows that FINE-VH outperforms existing power-management schemes, like Vdd-Hopping and Vdd-Dithering, and, most importantly, goes beyond the theoretical limit imposed by ideal-DVFS. An accurate parametric analysis clearly stated that finer layout granularity enhances FINE-VH power efficiency. Average power savings at low Vdd, when ΔVdd is fixed to 200 mV, increases from 38.5% for 9 tiles, up to 58.9% for 49 tiles. Once again the rule of thumb "the finer, the better" is confirmed. An important design variable is the difference between VddL and VddH, ΔVdd, as a proper choice may depend on the characteristics of

the off-chip voltage regulators and the actual design specifications. That said, our quantitative analysis reveals that FINE-VH can work for a wide range of ΔVdd values, from 100 mV (10%Vdd) up to 200 mV (20%Vdd).

4.2 Power optimization on CPUs: Voltage-Scaled ConvNets

Power consumption is a primary source of concern for the execution of deep learning workloads, especially for real-time applications that run inference over a continuous stream of data. Commonly referred to as continuous inference, some real-world applications include context sensing [138], health monitoring [139], and object tracking [140], for which ceaseless uploading of data in the cloud would be impractical due to high communication costs and unpredictable response time.

Today, general-purpose CPUs represent the most adopted design choice for embedded platforms. Millions of chip-sets integrating Reduced Instruction Set Computers (RISCs) are already in the field (e.g., in smartphones) and can be empowered with ConvNets through a simple software update [141]. To deploy compute-intensive workloads like ConvNets, a common practice is to adopt heterogeneous designs powered by high-end multi-core CPUs with parallel arithmetic units. For instance, the ARM Cortex-A15 of the Samsung Exynos 5422 system-on-chip (SoC) [142] embeds a four-core CPU that can operates with a maximum frequency of 2.0 GHz. Each core is equipped with the NEON unit [143], a Single-Instruction Multiple-Data (SIMD) data-path for parallel arithmetic. Furthermore, dedicated low-level routines optimized for linear algebra (e.g. convolutions between multi-dimensional matrices) are used at compile time to build highly efficient executable code [144].

However, even the most compact ConvNets are super-dense workloads that flood the hardware, saturating both memory and CPU utilization. While this may seem positive in terms of efficiency, it raises a serious concern over long execution intervals, when thermal issues arise affecting the reliability. This aspect is neglected in modern deep learning optimization frameworks. Since high-end cores are integrated into embedded devices with a small form factor, high utilization rates come at the cost of much higher power density, which in turn generates more heat than cooling systems can dissipate. The on-chip temperature increases quickly, reaching critical values (e.g. 90 °C) even in short time windows. High temperatures activate several degradation mechanisms that affect the lifetime of the device [112] and the user experience. This problem calls for a control mechanism to avoid thermal runaway.

The working principle of thermal control strategies is to switch active cores

into a low power state as soon as their temperature reaches a critical threshold. Thanks to lower power dissipation, temperatures cool down, allowing to restore the thermal stability of the system. As a downside, thermal control strategies limit the CPUs speed, bringing a significant mismatch between nominal (i.e. at maximum speed) and actual performance (i.e. under thermal management). As will be shown later, state-of-the-art ConvNets, both accuracy optimized ConvNets (e.g. Inception [145]) and performance optimized ConvNets (e.g. MobileNets [12]), exceeds the safety-critical temperature just after 1–3 s of continuous inference, making the average performance gap quite large. Neglecting this aspect may have dramatic impacts on the dependability of the system, leading to functional failures in the worst case. Instead, a thermal-conscious deployment of ConvNets would help to improve several figures of merit. While thermal issues and thermal-aware hardware/software co-design are well established topics in the literature, the intersection with ConvNets is a less explored field. An in-depth analysis may reveal interesting trends with new insights for the optimization of ConvNets. This is the aim of this section, which presents a thorough assessment of the performance of ConvNets under thermal management, implemented into low power CPUs for mobile applications.

Among the existing alternatives, supply-voltage lowering is a common knob to match Thermal Design Power (TDP) constraints. If combined with frequency scaling, it enables a cubic reduction of dynamic power consumption. Moreover, lower voltages reduce static power. Embedded CPUs integrate Dynamic Voltage and Frequency Scaling (DVFS) mechanisms offering an extended range of operating points in the power-temperature-performance space. DVFS is managed by software routines, called thermal governors, which implement temperature-driven DVFS to maximize performance within the available TDP budget. Identifying the best control policy is an interesting problem that has been extensively addressed in the literature. While sophisticated schemes based on workload prediction and/or temperature speculation are currently available [146], ConvNets are static graphs with data-independent workloads. This offers a unique opportunity to profile the thermal-vs.-performance behavior at design-time. We thereby developed an automatic framework that supports multiple ConvNet models allowing a parametric analysis over different use cases. The experiments, conducted over four state-of-the-art ConvNets for computer vision tasks ported on an Odroid-XU4 board [147] powered by the ARM Cortex-A15 core, enables the following key achievements:

- Quantify the thermal headroom of ConvNets in the context of continuous inference. Our analysis identifies applications that can be critical for power-constrained devices.
- Assess the performance of ConvNets under thermal-aware DVFS. The experiments cover two control policies, namely reactive and proactive.

- Identify the optimal operating points of voltage scaled ConvNets. The analysis suggests useful insights to support the development of smarter control policies specialized for ConvNets.
- Prove that the thermal profile of ConvNets depends on the network architecture. The collected results reveal the need for new optimization techniques for training thermal-aware ConvNets.

Our experiments demonstrate that designers should carefully assess the thermal and power constraints of the hosting hardware to avoid mismatches between expected performance (at design-time) and run-time execution. The conducted analysis serves as a baseline for future thermal-aware ConvNet optimization. For instance, standard optimization methods like neural network compression [86] and neural architecture search [148] might exploit the power-thermal characterization of the target hardware.

The remainder of this section is organized as follows. We first summarize the standard thermal management mechanisms implemented in off-the-shelf embedded systems. Second, we describe the proposed characterization framework. Finally, we report the experimental results and discuss the main findings of our analysis.

4.2.1 Thermal-Aware Power Management in embedded CPUs: Reactive vs. Proactive DVFS

Thermal Management Strategies

Thermal management strategies change the operating point of the system to reduce the power consumption and control the on-chip temperature. Among the available options, DVFS represents one of the most effective knobs since the active power consumption shows a quadratic dependence on voltage and a linear dependence on frequency.

Commercial CPUs offer a wide set of voltage and frequency (VF) levels (19 in the Cortex-A15), which enable a fine grained control on power and performance. As will be discussed in Section 4.2.3, we observed that other knobs are less efficient for controlling temperature during continuous inference. Each VF level identifies a specific operating point in the power-performance space. The maximum performance can be achieved using the highest voltage and the maximum frequency available, which we refer to as VF_{max}; within the Cortex-A15, $VF_{\text{max}} = 1.3625\,\text{V}$ @ $2\,\text{GHz}$. Changing the operating point at run-time enables managing the power-performance trade-off, which means controlling the temperature gradient at the cost of performance penalty.

An efficient management policy aims to guarantee thermal stability with minimum speed degradation. Off-the-shelf SoCs implements a reactive thermal management mechanism. To meet high computational demands, the active cores operate at $VF_{\max}$ and invoke a safety mechanism, thermal throttling, that reduces the VF level when the temperature reaches a critical threshold, thus preventing the processor, and the whole device, from overheating. For instance, the Cortex-A15 CPU down-scales the voltage-frequency level from $VF_{\max}$ to $VF_{\text{low}} = 0.8875\,\text{V}$ @ 900 MHz when the temperature exceeds $T_{\max} = 90\,°\text{C}$. A qualitative analysis of this strategy is depicted in Figure 4.16.

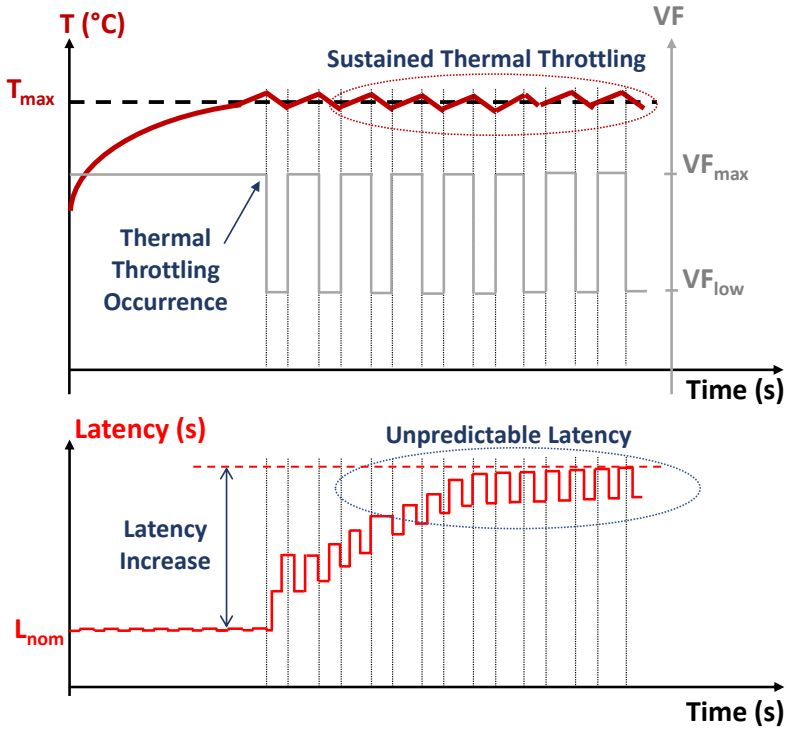


Figure 4.16: Qualitative trends of temperature (above) and inference latency (below) over time under reactive thermal management.

Under intensive workloads, like those of ConvNets, this mechanism may lead to significant performance degradation, especially when continuous inference is held for long time intervals. As shown in the top plot of Figure 4.16, running the cores at maximum performance pushes the temperature towards the critical threshold $T_{\max}$ and forces the SoC to throttle the performance of the cores switching from the high performance state $VF_{\max}$ to the low power state VF_{low}. As soon as the temperature falls below $T_{\max}$, the SoC switches back to $VF_{\max}$, forcing another invocation of thermal throttling in a very short time; the sequence repeats ceaselessly till

the task ends. This working mode is called sustained thermal throttling: the temperature fluctuates around the safety threshold over a sustained period, and so does the voltage-frequency operating point, which moves up and down between VF_{max} and VF_{low}. As shown in the bottom plot of Figure 4.16, this has a negative impact on latency: (i) working at VF_{low} introduces an overhead with respect to the nominal latency L_{nom}; (ii) the cyclic swapping from high performance (VF_{max}) to low power (VF_{low}) modes makes the latency less predictable. For these reasons, reactive strategies turn out to be quite inefficient. In the specific case of continuous inference, we measured a latency overhead ranging from 30% to 43% depending on the ConvNet, together with an increase of variability up to 70× (see Section 4.2.3 for more details).

Proactive thermal management represents a more efficient alternative. It works ahead of time as it enacts a more stable voltage lowering before the temperature reaches critical limits. More precisely, the CPU is made to work at an intermediate operating point VF_{opt} between VF_{max} and VF_{low} from the beginning. The benefits are qualitatively shown in Figure 4.17, which provides a comparison against the reactive strategy. While the proactive approach introduces some performance overhead on the very first short term ($L_{\text{opt}} > L_{\text{nom}}$), it ensures substantial gains in the long term because it prevents the occurrence of the throttling events. Overall, the average latency improves, while the predictability is guaranteed for much longer. Under highly demanding workloads of a very long duration, the temperature might reach critical values even with a proactive control, and hence, thermal throttling (from VF_{opt} to VF_{low}) may still occur. However, its occurrence is less frequent.

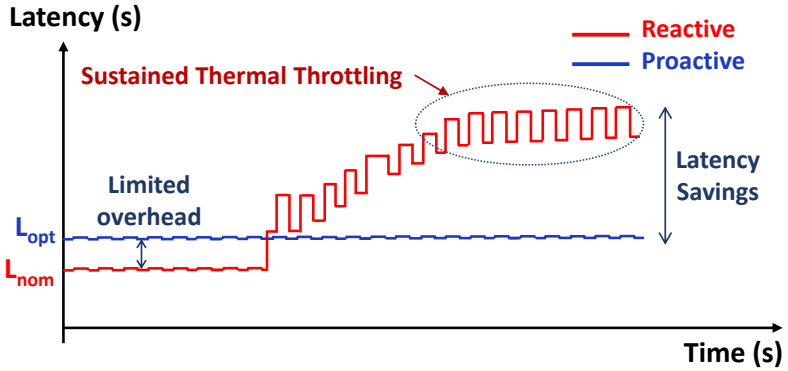


Figure 4.17: Inference latency in reactive (red) and proactive (blue) thermal management.

Optimal Power-Performance Trade-off

For a given maximum temperature, there exists an optimal trade-off between power (i.e., the VF level) and performance. Figure 4.18 gives a graphical representation of such an optimality problem. The plot shows the average latency (L_{avg}) for different VF levels considering a pre-defined sequence of N inference runs. On the right side of the minimum latency point, achieved working at VF_{opt}, it happens that frequent throttling events induce performance penalty. On the left side of VF_{opt}, thermal throttling does not occur often, but latency increases due to a too conservative voltage-frequency scaling. The precise position of VF_{opt} depends on the total execution time, i.e., the number of inference runs N, and the topology of ConvNet (size, number of operations, and memory allocation).

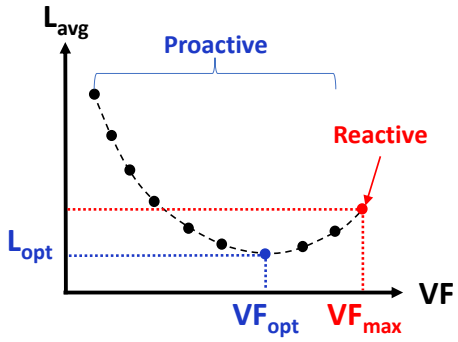


Figure 4.18: Average latency (L_{avg}) under reactive and proactive thermal management strategies.

Proactive DVFS Policies

Several works investigated the implementation of proactive thermal management on embedded systems. They presented control policies that aimed to identify the optimal operating point of the system depending on the current workload. An exhaustive taxonomy of the existing techniques can be found in [146] This subsection summarizes the most important approaches.

The seminal work of [149] presented a closed-loop controller that adjusts the voltage and frequency level to reduce the error between the expected and measured performance. More advanced controllers incorporate regression models to predict the future temperature and identify the operating point that achieves maximum performance, yet avoiding thermal violation. The model can be trained off-line on a set of representative benchmarks [150] or it can be continuously updated at run-time [151, 152].

Motivated by the observation that ConvNets are static graphs that always execute the same flow of operations, we propose a characterization framework that

enables extracting the thermal profile of a given ConvNet at design time. Rather than proposing a novel controller, this work aims to quantify the performance of ConvNets in a power/thermal constrained environment and to identify the best operating points for thermal management during continuous inference.

4.2.2 Thermal-Aware Performance Optimization and Characterization Framework

The problem of finding the optimal operating point of continuous-inference applications under proactive thermal management can be formulated as follows: given a pre-trained ConvNet, deployed on a given embedded CPU, and made to run for a fixed number of inferences N, find the voltage-frequency operating point VF_{opt} that minimizes the average latency. Considering the relatively low cardinality of the solution space, we opted for an exhaustive exploration conducted through the characterization framework shown in Figure 4.19.

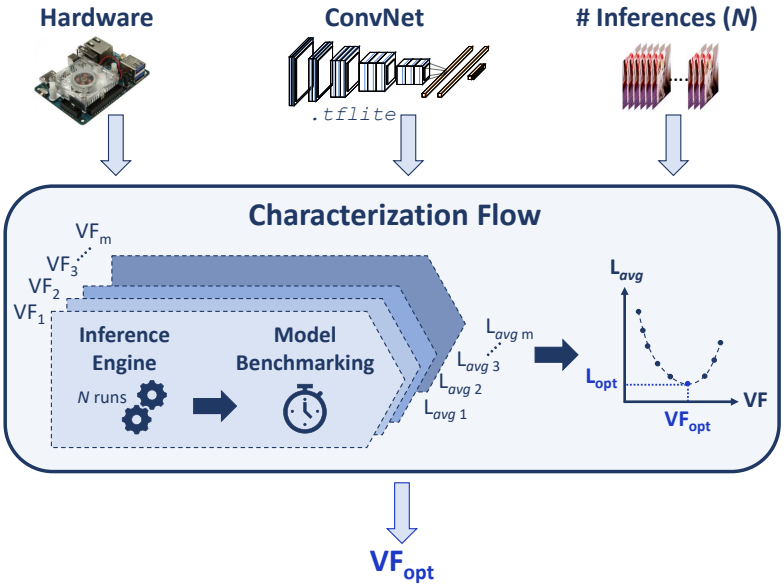


Figure 4.19: Schematic view of the proposed characterization flow.

The framework consists of two main components: (i) an inference engine that runs the ConvNet workload and (ii) a benchmarking tool that is in charge of assessing the performance, i.e., the inference latency. The inference engine is based on TensorFlow Lite (TFL) by Google, i.e., a collection of software routines for deep learning highly optimized to run tensor graphs on multi-core processors integrating an SIMD data-path. Furthermore, TFL integrates a benchmarking utility, called the TensorFlow Lite Model Benchmark, that allows the measurement

of the inference time on the target device by randomly assembling inputs from the dataset. The tool collects several statistics recorded on-board, in particular the average latency and the standard deviation over multiple runs.

The frameworks are fed with three main inputs: (i) a ConvNet architecture in `tflite` format; (ii) the number of continuous inferences N; (iii) the specifications of the device that hosts the ConvNet (i.e., the available VF levels). The framework is compiled and executed on the specified hardware to collect the average latency over N continuous inferences run for all available VF points. The main outcome is the minimum latency value L_{opt} and the corresponding optimal operating point VF_{opt}.

4.2.3 Experimental Setup and Results

The objective of the analysis reported in this section is threefold: (i) understand when continuous inference generates temperature violations; (ii) quantify the actual performance of ConvNets under reactive/proactive thermal management and different network architectures; (iii) identify the optimal operating points for voltage scaled ConvNets to guide the development of smart control policies oriented toward neural tasks. The contents are organized as follows. First, we report the hardware specifications of the board adopted in the experiments, together with the software environment used for the deployment. Second, we describe the ConvNets taken as benchmarks. Finally, we report the collected results and discuss the key insights.

Hardware Platform and Software Configurations

The hardware test bench was the Odroid-XU4 platform equipped with the Samsung Exynos 5422 SoC. The platform hosts a quad-core CPU (ARM Cortex-A15), that can operate up to $VF_{\text{max}} = 1.3625\,\text{V}$ @ $2\,\text{GHz}$ in nominal conditions. The operating system running on-board is Ubuntu Mate 16.04, kernel Version 3.10.106-154, released by Hardkernel. The standard thermal governor implements a reactive policy, which scales the VF level of the CPU cores down to $VF_{\text{low}} = 0.8875\,\text{V}$ @ $900\,\text{MHz}$ as soon as the temperature exceeds the safety threshold $T_{\text{max}} = 90\,^{\circ}\text{C}$. The operating-system kernel offers 19 voltage and frequency levels with a step of $100\,\text{MHz}$ (the minimum frequency is $200\,\text{MHz}$). For the sake of brevity, we denote the VF operating points just using the frequency value (in GHz). The board was cooled with an active fan controlled by pulse-width modulation (PWM); all the experiments were run at a constant fan speed of 36%. Unless explicitly specified, collected measurements refer to four thread execution.

The inference engine was TensorFlow Lite 1.14. It offers a set of low-level routines tailored to run neural operators on the ARM Cortex-A architecture.

Specifically, the convolutional operators make use of SIMD instructions to leverage the parallel arithmetic data-paths of the NEON unit [89]. In our setup, TensorFlow Lite was cross-compiled using the GNU ARM Embedded Toolchain (Version 6.5) [153].

ConvNet Benchmarks

The adopted ConvNets were picked from the TensorFlow Hosted Models [154] repository. In particular, we used two representative types of models: MobileNet and Inception. For each model, we investigated two different versions for a total of four ConvNets; their features are summarized in Table 4.2. All the ConvNets make use of an 8-bit fixed-point representation, a common choice for edge inference as it guarantees lower memory footprint and improved performance with marignal accuracy loss related to the floating-point. The column Memory reports the size of the `tflite`, which contains the data structures needed to execute the model on-chip, i.e., the network weights and the topology description. The column Top-1 refers to the top-1 classification accuracy measured on the ImageNet validation set. The column L_{nom} reports the nominal latency, i.e. the one measured under maximum performance (VF_{max}). The reported numbers refer to the average over 100 inference runs, each of them interleaved by a two-second pause to avoid temperature variations of the chip. The column σ_{nom} reports the standard deviation of the nominal latency measured over the same 100 runs.

Table 4.2: Memory, accuracy, and nominal latency of the selected benchmarks.

ConvNet	Memory (MB)	Top-1 (%)	L_{nom} (ms)	σ_{nom} (ms)
MobileNet v1	4.3	70.0	31.99	0.06
MobileNet v2	3.4	70.8	30.24	0.06
Inception v1	6.4	70.1	87.84	0.13
Inception v4	41.0	79.5	658.06	0.57

MobileNets are lightweight models optimized for high performance on embedded applications. Inception models are designed to maximize accuracy; therefore, they have a more complex architecture that requires more memory and computational resources. Inception v4 achieves 8.7% higher accuracy than MobileNet v2 at the cost of 12× more memory and 22× higher latency.

Results

Thermal headroom in continuous inference. Table 4.3 reports the number of continuous inferences N_{safe} and the execution time t_{safe} at $VF_{\text{max}} = 2.0\,\text{GHz}$

before the temperature exceeds the critical threshold $T_{\max} = 90\,°\text{C}$. For all the ConvNets, the critical threshold was reached after a low number of inferences, e.g., only four in Inception v4. This motivated the need for thermal management.

It is possible to observe different trends across the selected benchmarks. For instance, Inception v4 presented the first thermal throttling event 2.3× later than MobileNets v2 (2.93 s vs. 1.27 s). This finding suggests that the thermal gradient strictly depends on the topology of ConvNets, which is quite intuitive as different models come with a different number of layers of different sizes and cardinality. Obviously, the smaller the net, the larger the number of inferences run within t_{safe}.

Table 4.3: Thermal headroom of different ConvNets in continuous inference. N_{safe} and t_{safe} are the maximum number of consecutive inferences and the execution time at safe temperature values (i.e., $T < T_{\max}$).

ConvNet	$\boldsymbol{N_{\text{safe}}}$	$\boldsymbol{t_{\text{safe}}}$ **(s)**
MobileNet v1	39	1.26
MobileNet v2	42	1.27
Inception v1	25	2.21
Inception v4	4	2.93

Performance under thermal management A more interesting analysis concerns the performance gap between reactive thermal management (i.e., working at $VF_{\max}$) and proactive thermal management (i.e., working at VF_{opt}). For proactive, the optimal level VF_{opt} is extracted from the proposed characterization framework (see Section 4.2.2). The framework runs a continuous number of inferences N, with N ranging from 50 to 1000 with a step of 50 inferences. Figure 4.21 (see the end of the section) reports the collected results for all the benchmarks.

The plots in Figure 4.21a show the average latency as a function of N; the red "×" marker refers to reactive. whereas the blue "+" marker is for proactive. The black dashed line quantifies the nominal latency L_{nom} (the exact value is also reported in the label). In both cases, thermal management produces a performance overhead with respect to L_{nom}. As expected, proactive management outperforms reactive management as it mitigates the occurrence of thermal throttling. For longer execution time, i.e., larger N, the level of VF_{opt} scales down (value reported in the blue boxes, in GHz), as larger thermal headroom is needed to ensure safety. Again, the performance analysis reveals different trends depending on the ConvNet topology. First, MobileNets showed a higher performance overhead with respect to L_{nom} than Inception nets when running in continuous inference. In the worst case, the overhead was 43% and 30% for MobileNet v2 and Inception v4, respectively. Second, in proactive management, the value of VF_{opt} varied with N, but also with

the kind of network. For example, in MobileNets, VF_{opt} scaled down to 1.7 GHz, whereas in Inception nets, the minimum value was 1.8 GHz.

Figure 4.21b gives a more detailed analysis of the performance gains achieved with proactive management. Concerning the MobileNets, the savings against reactive increased up to 15.3% and 16.8% for v1 and v2, respectively; for the Inception nets, the performance savings were greater than 10% for $N > 100$, with peaks of 15.1% and 16.2% for v1 and v4, respectively.

Proactive management also guaranteed lower latency variability. This is shown in Figure 4.21c. The plots report the standard deviation σ measured at different N. For all the benchmarks, a proactive strategy kept σ close to the variability measured at nominal conditions (depicted by the black dashed line). These findings demonstrate that proactive management enables a more efficient and reliable neural task scheduling.

Topology impact on temperature. To better analyze the impact of different topologies on the temperature gradient, we collected the chip temperature with a sampling rate of 10 ms over an interval time of 100 s. Figure 4.22a (reported at the end of the section) reveals that the Inception nets had a slower temperature gradient than that of the MobileNets, both in reactive and proactive management. This suggests there is room for power- and thermal-aware design of ConvNet operators that might help to achieve higher thermal stability. Current research trends in deep learning do not consider this aspect as the design of ConvNets is mainly driven by performance and energy optimization in nominal conditions, which is misleading indeed. Furthermore, we measured the latency of each inference over the same activity time in order to highlight the variations resulting from thermal management. The analysis is reported in Figure 4.22b. In reactive management (red line), the latency quickly increased, both in terms of absolute values and variability, as soon as the SoC entered the sustained thermal throttling regime. This condition held for all the benchmarks. On a long term period, proactive management (blue line) outperformed the reactive approach by far, ensuring better performance on average with a higher degree of stability. As expected, the latency under pro-active management became worse just in a very short term window at the beginning, when the cores were cold.

The bar plot in Figure 4.20 quantifies the percentage of time during which the cores were pushed to work in the low power mode (VF_{low}) due to the occurrence of thermal throttling. The percentage was much lower for proactive management indeed. Interestingly, Inception nets were less prone to thermal protection, even if their size is larger than MobileNets. Under proactive management, Inception v1 spent 277× less time in throttling than MobileNet v1, even if working at a higher VF level (1.8 GHz vs. 1.7 GHz, as shown in Figure 4.22b). Overall, proactive management can be appreciated as an effective strategy to reduce the throttling

time (2.77% as the worst case).

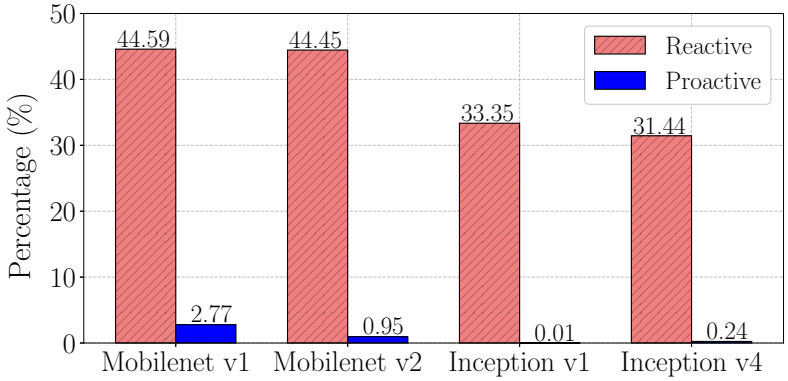


Figure 4.20: Percentage of execution time at $VF_{\text{low}} = 900\,\text{MHz}$ over a runtime of 100 s.

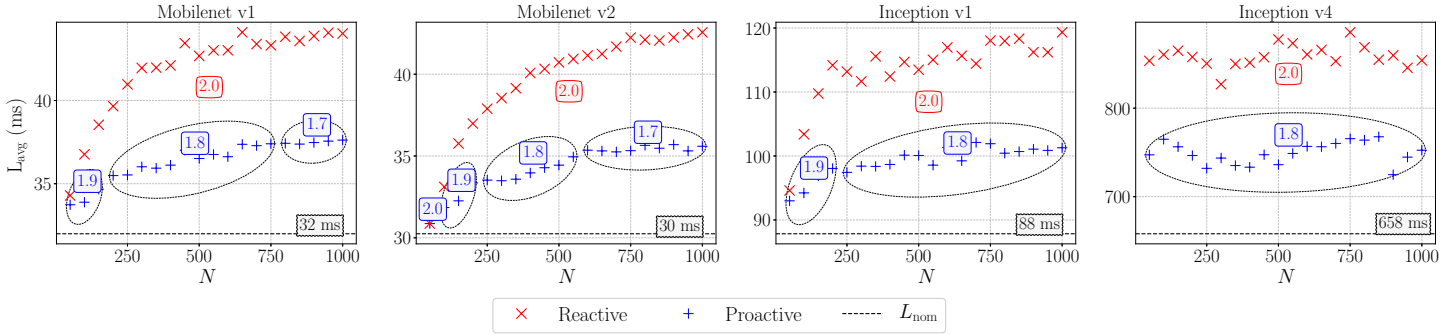
Efficiency of DVFS. Multi-core CPUs offer another power knob to control the thermal state of the chip: Dynamic Power Management (DPM) via core shutdown. This strategy adjusts the number of active cores to limit peak power consumption. Given the parallel nature of ConvNets, reducing the number of processing elements could have a severe impact on performance. Instead, supply voltage reduction enables a finer control on the power-performance trade-off. Table 4.4 provides empirical evidence of this observation. For each benchmark, the table reports the nominal latency of three thread execution (column $L_{\text{nom-3}}$) and the worst case latency measured under DVFS based proactive management at four thread execution (column $L_{\text{opt-wc}}$). Even in nominal conditions, i.e., at VF_{max} w/o thermal throttling, DPM had lower performance than thermal-aware DVFS ($L_{\text{nom-3}} > L_{\text{opt-wc}}$ in all cases). The collected results demonstrate that operating at low voltage is the most effective solution for the thermal management of ConvNets.

Table 4.4: Nominal inference latency at 3 thread execution vs. worst case latency under DVFS based proactive management at 4 thread execution.

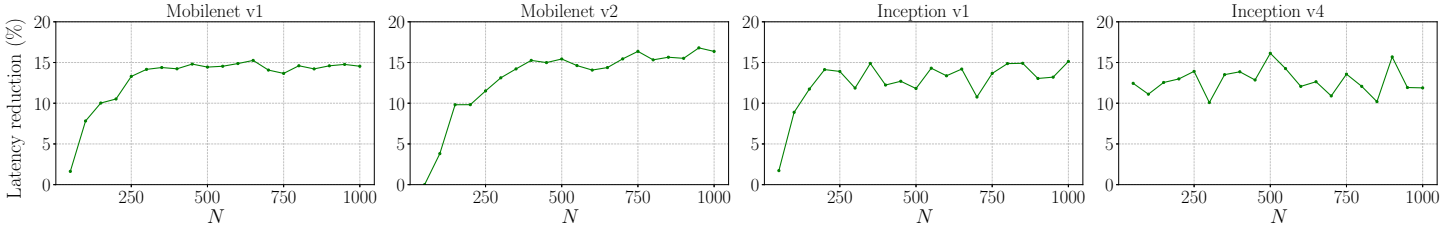
ConvNet	$\boldsymbol{L_{\text{nom-3}}}$ **(ms)**	$\boldsymbol{L_{\text{opt-wc}}}$ **(ms)**
MobileNet v1	41	38
MobileNet v2	38	36
Inception v1	103	102
Inception v4	770	768

4.2.4 Discussion

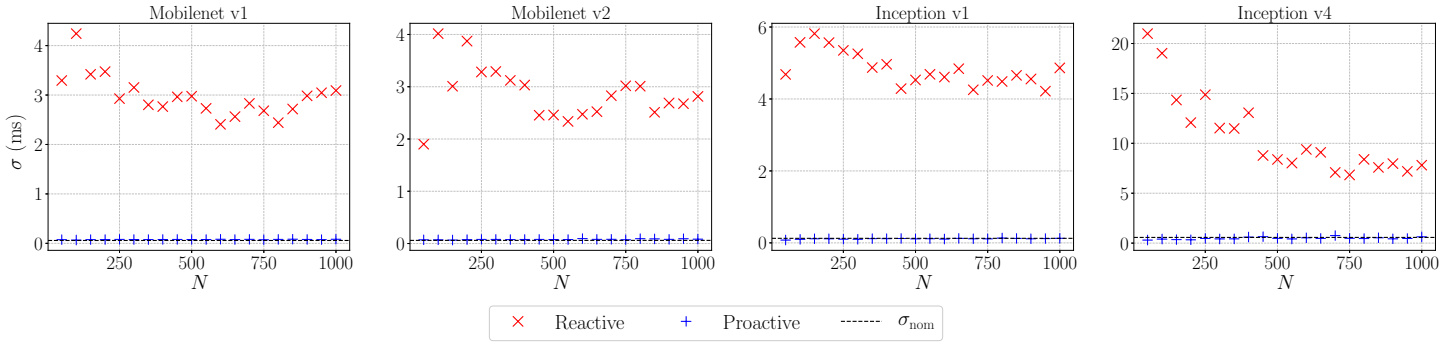
We presented a parametric characterization of the performance of embedded ConvNets under thermal management. The analysis, conducted on an off-the-shelf CPU for mobile applications, involved standard DVFS-based thermal protection schemes, i.e. both reactive and proactive control policies. The collected results serve as useful guidelines for hardware-conscious optimization of ConvNets, as they highlight a substantial mismatch between nominal and real performance in power-constrained systems that need thermal management to meet the TDP constraints. A thorough assessment proved that proactive control policies could help to limit the performance penalty and ensure low performance variability. Finally, the findings of our study suggest that future research efforts should investigate the design of thermal-aware neural network operators, as they provided empirical evidence that the thermal profile of ConvNets depends on the shape of the internal layers. There might exist networks with the same memory footprint and latency, but different thermal profiles. Identifying those configurations could help to improve the efficiency of continuous inference tasks.



(a) Average latency vs. inference number (N).



(b) Average latency reduction (in %) of proactive management with respect to reactive management.



(c) Latency standard deviation (σ) over N inferences.

Figure 4.21: Results of the characterization flow.

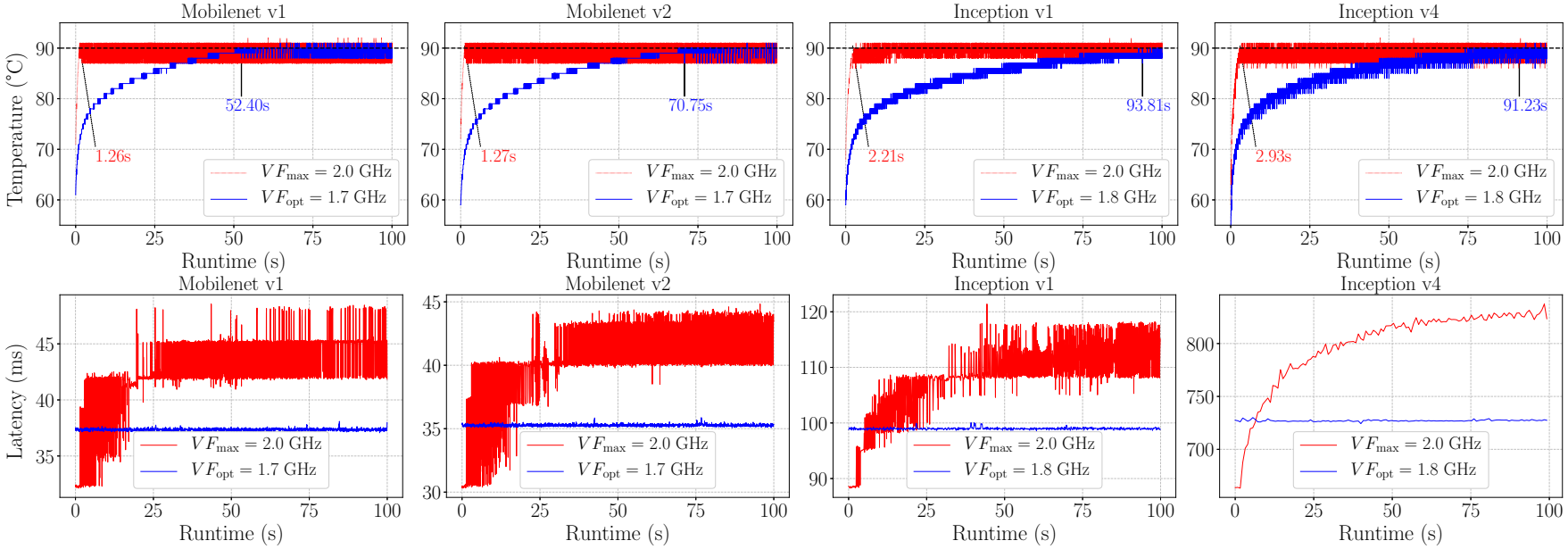


Figure 4.22: Temperature gradient (top) and inference latency (down) of continuous inference for 100 s. The annotations reports the occurrence of the first thermal throttling event.

Chapter 5

Conclusions

Thanks to the rapid advancements in deep learning theories, ConvNets have become state-of-the-art for data-analytics. The growing interest to bring data-analytics near the source of data stimulated the development of a wide variety of hardware platforms for efficient processing of ConvNets under tight resource budgets. In the beginning, software and hardware optimizations evolved as parallel layers. However, today there is a growing consensus that only a stronger interaction between the two layers can enable the shift of ConvNets from high-performance cloud servers to low-power IoT end-nodes. In this context, design automation plays a crucial role, with the development of cross-layer strategies for optimal software-to-silicon mapping. That's the target addressed by this dissertation.

In this thesis, we showed that the cloud-to-edge shift raises several challenges besides the computational complexity of ConvNets. The primary source of concern is the heterogeneity of application requirements and hardware solutions. Indeed, the deployment of ConvNets encompasses a multi-objective optimization involving different extra-functional metrics like memory, energy, and power. Furthermore, the increasing number of alternative hardware platforms requires a proper awareness of the limitations and opportunities for each case.

To answer these needs, we presented a comprehensive set of design&optimization strategies to explore the existing trade-offs in the accuracy-complexity space. Each of the proposed strategies targets a specific design goal. In this way, designers can select the most suited solutions to meet the design specifications. All the presented strategies share two common principles: (i) consider the characteristics of the underlying hardware from the early stages of the optimization process to guarantee portability; (ii) exploit additional knobs provided by the hardware through a joint co-operation with algorithmic optimization to maximize efficiency. Moreover, our proposals include both static (i.e. applied at design-time) and dynamic (i.e. operating at run-time) optimization strategies. The latter are of paramount importance as they enable to extend the optimization room.

Overall, our contribution can be summarized as follows.

In the context of memory optimization, we addressed the deployment of ConvNets on memory-bounded general-purpose MCUs. We first presented PaQ, a two-stage framework that efficiently explores the memory-accuracy space using a lightweight, hardware-conscious heuristic optimization. The proposed tool enabled the assessment of memory-bounded ConvNets, showing that most of them are centered on specific parameter settings that are found difficult to be implemented on a low-power RISC. To reduce the memory footprint of network weights, we presented EAST, a novel memory-constrained training procedure that leads quantized ConvNets towards deep compression. EAST implements an adaptive group pruning designed to maximize the compression rate of the weight encoding scheme.

Regarding energy optimization, we introduced the concept of Adaptive ConvNets, i.e. models that can tune their behavior at run-time to meet different energy budgets and accuracy constraints. We proposed two implementations of Adaptive ConvNets, both of them based on a per-layer precision scaling strategy. In On-line Precision Scaling the optimization is built upon a multi-objective problem formulation solved through a modified version of the NSGA-II genetic algorithm that ensures a fast design space exploration. In scalable-effort ConvNets, we presented a design methodology to perform classification of data at multilevel abstraction and reach a given classification accuracy with minimal computational effort.

Concerning power optimization, we focused on power management strategies specialized for ConvNets. With FINE-VH, we proposed a novel power distribution scheme for multi-processing engines (e.g. deep learning accelerators) to improve the efficiency of standard DVFS. For those cases where custom solutions are not viable, we studied the achievable power-performance trade-offs of ConvNets under thermal-aware DVFS, revealing the urgent need to tackle power optimization in the design of neural network architectures.

List of publications

This appendix lists all the publications produced during the PhD years. The list also reports some works not discussed in this dissertation. Most of them, however, are still related to the design and the optimization of embedded ConvNets.

Overall, the list of publications co-authored by Valentino Peluso includes:

- 3 accepted papers in international journals
- 3 book chapters
- 13 accepted papers in international conferences

International Journals

- Matteo Grimaldi, **Valentino Peluso**, Andrea Calimera: Optimality Assessment of Memory-Bounded ConvNets Deployed on Resource-Constrained RISC Cores. IEEE Access 7: 152599-152611 (2019).
- **Valentino Peluso**, Roberto Giorgio Rizzo, Andrea Calimera: Performance Profiling of Embedded ConvNets under Thermal-Aware DVFS. Electronics 8 (12), 1423.
- **Valentino Peluso**, Roberto Giorgio Rizzo, Andrea Calimera: Efficacy of Topology Scaling for Temperature and Latency Constrained Embedded ConvNets. Journal of Low Power Electronics and Applications 10 (1), 10.

Book Chapters

- **Valentino Peluso**, Roberto Giorgio Rizzo, Andrea Calimera, Enrico Macii, Massimo Alioto: Beyond Ideal DVFS Through Ultra-Fine Grain Vdd-Hopping. VLSI-SoC (Selected Papers) 2016: 152-172.
- Roberto Giorgio Rizzo, **Valentino Peluso**, Andrea Calimera, Jun Zhou: On the Efficiency of Early Bird Sampling (EBS) an Error Detection-Correction Scheme for Data-Driven Voltage Over-Scaling. VLSI-SoC (Selected Papers) 2017: 153-177.

- **Valentino Peluso**, Andrea Calimera: Energy-Accuracy Scalable Deep Convolutional Neural Networks: A Pareto Analysis. VLSI-SoC (Selected Papers) 2018: 107-127.

Proceedings of International Conferences

- **Valentino Peluso**, Andrea Calimera, Enrico Macii, Massimo Alioto: Ultra-Fine Grain Vdd-Hopping for energy-efficient Multi-Processor SoCs. VLSI-SoC 2016: 1-6.
- Roberto Giorgio Rizzo, **Valentino Peluso**, Andrea Calimera, Jun Zhou, Xin Liu: Early bird sampling: A short-paths free error detection-correction strategy for data-driven VOS. VLSI-SoC 2017: 1-6.
- **Valentino Peluso**, Andrea Calimera: Energy-Driven Precision Scaling for Fixed-Point ConvNets. VLSI-SoC 2018: 113-118.
- Giulia Santoro, Mario R. Casu, **Valentino Peluso**, Andrea Calimera, Massimo Alioto: Design-Space Exploration of Pareto-Optimal Architectures for Deep Learning with DVFS. ISCAS 2018: 1-5.
- **Valentino Peluso**, Andrea Calimera: Weak-MAC: Arithmetic Relaxation for Dynamic Energy-Accuracy Scaling in ConvNets. ISCAS 2018: 1-5.
- **Valentino Peluso**, Andrea Calimera: Scalable-effort ConvNets for multi-level classification. ICCAD 2018.
- Giulia Santoro, Mario R. Casu, **Valentino Peluso**, Andrea Calimera, Massimo Alioto: Energy-performance design exploration of a low-power micro-programmed deep-learning accelerator. DATE 2018: 1151-1154.
- Daniele Jahier Pagliari, **Valentino Peluso**, Yukai Chen, Andrea Calimera, Enrico Macii, Massimo Poncino: All-digital embedded meters for on-line power estimation. DATE 2018: 737-742.
- **Valentino Peluso**, Matteo Grimaldi, Andrea Calimera: Arbitrary-Precision Convolutional Neural Networks on Low-Power IoT Processors. VLSI-SoC 2019: 142-147.
- **Valentino Peluso**, Roberto Giorgio Rizzo, Antonio Cipolletta, Andrea Calimera: Inference on the Edge: Performance Analysis of an Image Classification Task Using Off-The-Shelf CPUs and Open-Source ConvNets. SNAMS 2019: 454-459.

- **Valentino Peluso**, Antonio Cipolletta, Francesco Vaiana, Andrea Calimera: Integer ConvNets on Embedded CPUs: Tools and Performance Assessment on the Cortex-A Cores. ICECS 2019: 598-601.
- **Valentino Peluso**, Antonio Cipolletta, Andrea Calimera, Matteo Poggi, Fabio Tosi, Stefano Mattoccia: Enabling Energy-Efficient Unsupervised Monocular Depth Estimation on ARMv7-Based Platforms. DATE 2019: 1703-1708.
- Matteo Grimaldi, **Valentino Peluso**, Andrea Calimera: EAST: Encoding-Aware Sparse Training for Deep Memory Compression of ConvNets. AICAS 2020: 233-237.

Bibliography

[1] Alex Krizhevsky et al. «Imagenet classification with deep convolutional neural networks». In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[2] Richard Socher et al. «Parsing natural scenes and natural language with recursive neural networks». In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 129–136.

[3] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. «Speech recognition with deep recurrent neural networks». In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. «Deep learning». In: *nature* 521.7553 (2015), pp. 436–444.

[5] Xiaowei Xu et al. «Scaling for edge inference of deep neural networks». In: *Nature Electronics* 1.4 (2018), p. 216.

[6] Yoshua Bengio et al. «Learning deep architectures for AI». In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[7] Léon Bottou. «Large-scale machine learning with stochastic gradient descent». In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[8] Renzo Andri et al. «YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights». In: *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE. 2016, pp. 236–241.

[9] Yu-Hsin Chen et al. «Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks». In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138.

[10] Lukas Cavigelli et al. «Origami: A 803-GOp/s/W Convolutional Network Accelerator». In: *IEEE Transactions on Circuits and Systems for Video Technology* 27.11 (Nov. 2017), pp. 2461–2475.

[11] Karen Simonyan and Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition». In: *arXiv preprint arXiv:1409.1556* (2014).

[12] Andrew G Howard et al. «Mobilenets: Efficient convolutional neural networks for mobile vision applications». In: *arXiv preprint arXiv:1704.04861* (2017).

[13] Mingxing Tan and Quoc Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». In: *International Conference on Machine Learning*. 2019, pp. 6105–6114.

[14] An-Chieh Cheng et al. «Searching toward pareto-optimal device-aware neural architectures». In: *Proceedings of the International Conference on Computer-Aided Design*. ACM. 2018, p. 136.

[15] Norman P. Jouppi et al. «In-Datacenter Performance Analysis of a Tensor Processing Unit». In: *SIGARCH Comput. Archit. News* 45.2 (June 2017), pp. 1–12.

[16] Stephen Cass. «Taking ai to the edge: Google's tpu now comes in a maker-friendly package». In: *IEEE Spectrum* 56.5 (2019), pp. 16–17.

[17] Valentino Peluso et al. «Weak-MAC: Arithmetic Relaxation for Dynamic Energy-Accuracy Scaling in ConvNets». In: *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE. 2018, pp. 1–5.

[18] Bert Moons et al. «DVAFS: Trading computational accuracy for energy through dynamic-voltage-accuracy-frequency-scaling». In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE. 2017, pp. 488–493.

[19] Hardik Sharma et al. «Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks». In: *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press. 2018, pp. 764–775.

[20] Yaman Umuroglu et al. «Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing». In: *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2018, pp. 307–3077.

[21] Jinmook Lee et al. «UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision». In: *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE. 2018, pp. 218–220.

[22] *NVIDIA TENSOR CORES*. 2019. URL: https://www.nvidia.com/en-us/data-center/tensorcore (visited on 05/25/2019).

[23] *Why the PowerVR Series2NX NNA is the future of neural net acceleration*. 2019. URL: https://www.imgtec.com/blog/why-the-powervr-2nx-nna-is-the-future-of-neural-net-acceleration (visited on 05/25/2019).

[24] Valentino Peluso et al. «Inference on the Edge: Performance Analysis of an Image Classification Task Using Off-The-Shelf CPUs and Open-Source ConvNets». In: *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. IEEE. 2019, pp. 454–459.

[25] Valentino Peluso et al. «Integer ConvNets on Embedded CPUs: Tools and Performance Assessment on the Cortex-A Cores». In: *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE. 2019, pp. 598–601.

[26] Andres Gomez, Francesco Conti, and Luca Benini. «Thermal image-based CNN's for ultra-low power people recognition». In: *Proceedings of the 15th ACM International Conference on Computing Frontiers*. ACM. 2018, pp. 326–331.

[27] Michele Magno et al. «DeepEmote: Towards multi-layer neural networks in a low power wearable multi-sensors bracelet». In: *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*. IEEE. 2017, pp. 32–37.

[28] Vivienne Sze et al. «Efficient processing of deep neural networks: A tutorial and survey». In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[29] Frederick Tung et al. «CLIP-Q: Deep network compression learning by in-parallel pruning-quantization». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7873–7882.

[30] Valentino Peluso, Matteo Grimaldi, and Andrea Calimera. «Arbitrary-Precision Convolutional Neural Networks on Low-Power IoT Processors». In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE. 2019, pp. 142–147.

[31] Matteo Grimaldi, Valentino Peluso, and Andrea Calimera. «Optimality assessment of memory-bounded convnets deployed on resource-constrained risc cores». In: *IEEE Access* 7 (2019), pp. 152599–152611.

[32] Matteo Grimaldi, Valentino Peluso, and Andrea Calimera. «EAST: Encoding-Aware Sparse Training for Deep Memory Compression of ConvNets». In: *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE. 2020, pp. 233–237.

[33] Song Han et al. «Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding». In: *arXiv preprint arXiv:1510.00149* (2015).

[34] Jiecao Yu et al. «Scalpel: Customizing dnn pruning to the underlying hardware parallelism». In: *ACM SIGARCH Computer Architecture News*. Vol. 45. 2. ACM. 2017, pp. 548–560.

[35] Manu Mathew et al. «Sparse, quantized, full frame cnn for low power embedded devices». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 11–19.

[36] Bert Moons et al. «An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS». In: *IEEE Journal of solid-state Circuits* 52.4 (2017), pp. 903–914.

[37] Hao Li et al. «Pruning filters for efficient convnets». In: *arXiv preprint arXiv:1608.08710* (2016).

[38] Deepak Mittal et al. «Recovering from random pruning: On the plasticity of deep convolutional neural networks». In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 848–857.

[39] Jiantao Qiu et al. «Going deeper with embedded fpga platform for convolutional neural network». In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM. 2016, pp. 26–35.

[40] Hande Alemdar et al. «Ternary neural networks for resource-efficient AI applications». In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2547–2554.

[41] Mohammad Rastegari et al. «Xnor-net: Imagenet classification using binary convolutional neural networks». In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[42] Manuele Rusci et al. «Quantized NNs as the definitive solution for inference on low-power ARM MCUs?: work-in-progress». In: *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press. 2018, p. 12.

[43] *Why the PowerVR Series2NX NNA is the future of neural net acceleration*. URL: https://www.imgtec.com/blog/why-the-powervr-2nx-nna-is-the-future-of-neural-net-acceleration/e (visited on 04/08/2019).

[44] Bert Moons et al. «Energy-efficient convnets through approximate computing». In: *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*. IEEE. 2016, pp. 1–8.

[45] Liangzhen Lai and Naveen Suda. «Enabling Deep Learning at the loT Edge». In: (2018), pp. 1–6.

[46] Sebastian Vogel et al. «Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base». In: *Proceedings of the International Conference on Computer-Aided Design*. ACM. 2018, p. 9.

[47] *GAP8 Hardware Reference Manual*. URL: https://gwt-website-files.s3.amazonaws.com/gap8_datasheet.pdf (visited on 08/08/2019).

[48] Francesco Conti et al. «PULP: A ultra-low power parallel accelerator for energy-efficient and flexible embedded vision». In: *Journal of Signal Processing Systems* 84.3 (2016), pp. 339–354.

[49] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[50] Pete Warden. «Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition». In: *arXiv preprint arXiv:1804.03209* (2018).

[51] *Challenges in representation learning: Facial expression recognition challenge*. URL: http://www.kaggle.com/ (visited on 05/08/2019).

[52] Aravind Vasudevan et al. «Parallel multi channel convolution using general matrix multiplication». In: *Application-specific Systems, Architectures and Processors (ASAP), 2017 IEEE 28th International Conference on*. IEEE. 2017, pp. 19–24.

[53] Philipp Gysel et al. «Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks». In: *IEEE Transactions on Neural Networks and Learning Systems* 29.11 (2018), pp. 5784–5789.

[54] Diederik P Kingma et al. «Adam: A method for stochastic optimization». In: *arXiv preprint arXiv:1412.6980* (2014).

[55] Yangqing Jia et al. «Caffe: Convolutional architecture for fast feature embedding». In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 675–678.

[56] Tara N Sainath et al. «Convolutional neural networks for small-footprint keyword spotting». In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

[57] Lutz Prechelt. «Early stopping-but when?» In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[58] Bowen Baker et al. «Accelerating neural architecture search using performance prediction». In: *arXiv preprint arXiv:1705.10823* (2017).

[59] Song Han et al. «Learning both weights and connections for efficient neural network». In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[60] Michael Zhu and Suyog Gupta. «To prune, or not to prune: exploring the efficacy of pruning for model compression». In: *arXiv preprint arXiv:1710.01878* (2017).

[61] *LZ4*. URL: https://lz4.github.io/lz4/.

[62] Song Han, Huizi Mao, and William J. Dally. «Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding». In: *4th International Conference on Learning Representations, ICLR 2016*. 2016.

[63] Matteo Grimaldi, Valerio Tenace, and Andrea Calimera. «Layer-Wise Compressive Training for Convolutional Neural Networks». In: *Future Internet* 11.1 (2019), p. 7.

[64] Jonathan Frankle et al. «The Lottery Ticket Hypothesis at Scale». In: *arXiv preprint arXiv:1903.01611* (2019).

[65] Song Han et al. «EIE: efficient inference engine on compressed deep neural network». In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2016, pp. 243–254.

[66] Angshuman Parashar et al. «Scnn: An accelerator for compressed-sparse convolutional neural networks». In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2017, pp. 27–40.

[67] *TorchSkeleton*. URL: https://github.com/wbaek/torchskeleton (visited on 05/08/2019).

[68] Cody Coleman et al. «Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark». In: *ACM SIGOPS Operating Systems Review* 53.1 (2019), pp. 14–25.

[69] *NUCLEO-F412ZG*. URL: https://www.st.com/en/evaluation-tools/nucleo-f412zg.html (visited on 05/08/2019).

[70] Renzo Andri et al. «YodaNN: An architecture for ultralow power binary-weight CNN acceleration». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.1 (2017), pp. 48–60.

[71] Jiuxiang Gu et al. «Recent advances in convolutional neural networks». In: *Pattern Recognition* 77 (2018), pp. 354–377.

[72] Xitong Gao et al. «Dynamic Channel Pruning: Feature Boosting and Suppression». In: *International Conference on Learning Representations*. 2019.

[73] Xin Wang et al. «Skipnet: Learning dynamic routing in convolutional networks». In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 409–424.

[74] Daniele Jahier Pagliari, Enrico Macii, and Massimo Poncino. «Dynamic Bit-width Reconfiguration for Energy-Efficient Deep Learning Hardware». In: *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM. 2018, p. 47.

[75] Valentino Peluso and Andrea Calimera. «Energy-Driven Precision Scaling for Fixed-Point ConvNets». In: *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE. 2018, pp. 113–118.

[76] Hokchhay Tann et al. «Runtime configurable deep neural networks for energy-accuracy trade-off». In: *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM. 2016, p. 34.

[77] Valentino Peluso and Andrea Calimera. «Energy-Accuracy Scalable Deep Convolutional Neural Networks: A Pareto Analysis». In: *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*. Springer. 2018, pp. 107–127.

[78] Valentino Peluso and Andrea Calimera. «Scalable-effort convnets for multilevel classification». In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2018, pp. 1–8.

[79] Ji Lin et al. «Runtime neural pruning». In: *Advances in Neural Information Processing Systems*. 2017, pp. 2181–2191.

[80] Andreas Veit and Serge Belongie. «Convolutional networks with adaptive inference graphs». In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 3–18.

[81] Jiahui Yu et al. «Slimmable Neural Networks». In: *International Conference on Learning Representations*. 2019.

[82] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. «Conditional Deep Learning for Energy-Efficient and Enhanced Pattern Recognition». In: *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. DATE '16. Dresden, Germany: EDA Consortium, 2016, pp. 475–480. ISBN: 9783981537062.

[83] Michael Lones. *Sean Luke: essentials of metaheuristics*. 2011.

[84] Yu Cheng et al. «A survey of model compression and acceleration for deep neural networks». In: *arXiv preprint arXiv:1710.09282* (2017).

[85] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. «Designing energy-efficient convolutional neural networks using energy-aware pruning». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5687–5695.

[86] Tien-Ju Yang et al. «Netadapt: Platform-aware neural network adaptation for mobile applications». In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 285–300.

[87] Kuan Wang et al. «Haq: Hardware-aware automated quantization with mixed precision». In: (2019), pp. 8612–8620.

[88] Bichen Wu et al. «Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search». In: *arXiv preprint arXiv:1812.00090* (2018).

[89] Benoit Jacob et al. «Quantization and training of neural networks for efficient integer-arithmetic-only inference». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2704–2713.

[90] Kalyanmoy Deb et al. «A fast and elitist multiobjective genetic algorithm: NSGA-II». In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[91] Marco Laumanns et al. «Combining convergence and diversity in evolutionary multiobjective optimization». In: *Evolutionary computation* 10.3 (2002), pp. 263–282.

[92] *Open Neural Network eXchange (ONNX) Model Zoo*. 2019. URL: https://github.com/onnx/models (visited on 05/25/2019).

[93] Jia Deng et al. «Imagenet: A large-scale hierarchical image database». In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[94] Kaiming He et al. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[95] Forrest N Iandola et al. «SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size». In: *arXiv preprint arXiv:1602.07360* (2016).

[96] Mark Sandler et al. «Mobilenetv2: Inverted residuals and linear bottlenecks». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520.

[97] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. «Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0». In: *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society. 2007, pp. 3–14.

[98] Sheng Li et al. «CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques». In: *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press. 2011, pp. 694–701.

[99] Ananda Samajdar et al. «Scale-sim: Systolic cnn accelerator». In: *arXiv preprint arXiv:1811.02883* (2018).

[100] Adam Paszke et al. «PyTorch: An imperative style, high-performance deep learning library». In: *Advances in Neural Information Processing Systems*. 2019, pp. 8024–8035.

[101] Tao Sheng et al. «A quantization-friendly separable convolution for mobilenets». In: *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE. 2018, pp. 14–18.

[102] Zhicheng Yan et al. «HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition». In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2740–2748.

[103] Bert Moons et al. «14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi». In: *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE. 2017, pp. 246–247.

[104] Alsallakh Bilal et al. «Do convolutional neural networks learn class hierarchy?» In: *IEEE transactions on visualization and computer graphics* 24.1 (2018), pp. 152–162.

[105] George A Miller. «WordNet: a lexical database for English». In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[106] Israel Koren. *Computer arithmetic algorithms*. Universities Press, 2002.

[107] Clive Maxfield et al. *The definitive guide to how computers do math: featuring the virtual DIY calculator*. Vol. 1. John Wiley & Sons, 2005.

[108] Philipp Gysel et al. «Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks». In: *IEEE transactions on neural networks and learning systems* 29.11 (2018), pp. 5784–5789.

[109] *Fixed Point Quantization*. URL: https://www.tensorflow.org/performance/quantization.

[110] Olga Russakovsky et al. «ImageNet Large Scale Visual Recognition Challenge». In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[111] Mark Sandler et al. «Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation». In: *arXiv preprint arXiv:1801.04381* (2018).

[112] David Brooks et al. «Power, thermal, and reliability modeling in nanometer-scale microprocessors». In: *Ieee Micro* 27.3 (2007), pp. 49–62.

[113] Valentino Peluso, Roberto Giorgio Rizzo, and Andrea Calimera. «Efficacy of Topology Scaling for Temperature and Latency Constrained Embedded ConvNets». In: *Journal of Low Power Electronics and Applications* 10.1 (2020), p. 10.

[114] Giulia Santoro et al. «Design-space exploration of pareto-optimal architectures for deep learning with dvfs». In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2018, pp. 1–5.

[115] Valentino Peluso et al. «Ultra-fine grain Vdd-hopping for energy-efficient multi-processor SoCs». In: *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE. 2016, pp. 1–6.

[116] Valentino Peluso et al. «Beyond ideal DVFS through ultra-fine grain vdd-hopping». In: *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*. Springer. 2016, pp. 152–172.

[117] Giulia Santoro et al. «Energy-performance design exploration of a low-power microprogrammed deep-learning accelerator». In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 1151–1154.

[118] Valentino Peluso, Roberto Giorgio Rizzo, and Andrea Calimera. «Performance Profiling of Embedded ConvNets under Thermal-Aware DVFS». In: *Electronics* 8.12 (2019), p. 1423.

[119] Vasanth Venkatachalam and Michael Franz. «Power reduction techniques for microprocessor systems». In: *ACM Computing Surveys (CSUR)* 37.3 (Sept. 2005), pp. 195–237.

[120] Tejaswini Kolpe, Antonia Zhai, and Sachin S. Sapatnekar. «Enabling improved power management in multicore processors through clustered DVFS». In: *DATE'11: Design, Automation & Test in Europe Conference & Exhibition*. IEEE. Mar. 2011, pp. 1–6.

[121] Dean N. Truong et al. «A 167-processor computational platform in 65 nm CMOS». In: *IEEE Journal of Solid-State Circuits* 44.4 (Apr. 2009), pp. 1130–1144.

[122] Saurabh Dighe et al. «Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor». In: *IEEE Journal of Solid-State Circuits* 46.1 (Nov. 2010), pp. 184–193.

[123] J. Park et al. «Accurate modeling and calculation of delay and energy overheads of dynamic voltage scaling in modern high-performance microprocessors». In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. Aug. 2010, pp. 419–424.

[124] Sylvain Miermont, Pascal Vivet, and Marc Renaudin. «A power supply selector for energy-and area-efficient local dynamic voltage scaling». In: *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*. Vol. 4644. Springer, 2007, pp. 556–565.

[125] Edith Beigné et al. «An asynchronous power aware and adaptive NoC based circuit». In: *IEEE Journal of Solid-State Circuits* 44.4 (Apr. 2009), pp. 1167–1177.

[126] Valentino Peluso et al. «Ultra-Fine Grain Vdd-Hopping for energy-efficient Multi-Processor SoCs». In: *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE. Sept. 2016, pp. 1–6.

[127] Davide Rossi et al. «A 60 GOPS/W, -1.8 V to 0.9 V body bias ULP cluster in 28 nm UTBB FD-SOI technology». In: *Solid-State Electronics* 117 (Mar. 2016), pp. 170–184.

[128] L. Bolzani et al. «Enabling concurrent clock and power gating in an industrial design flow». In: *2009 Design, Automation Test in Europe Conference Exhibition*. Apr. 2009, pp. 334–339. DOI: 10.1109/DATE.2009.5090684.

[129] X. Liang, G. Y. Wei, and D. Brooks. «Revival: A Variation-Tolerant Architecture Using Voltage Interpolation and Variable Latency». In: *IEEE Micro* 29.1 (Jan. 2009), pp. 127–138.

[130] M. S. Gupta et al. «Tribeca: Design for PVT variations with local recovery and fine-grained adaptation». In: *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Dec. 2009, pp. 435–446.

[131] Mohammad Reza Kakoee and Luca Benini. «Fine-Grained Power and Body-Bias Control for Near-Threshold Deep Sub-Micron CMOS Circuits». In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 1.2 (June 2011), pp. 131–140.

[132] Yasumi Nakamura et al. «1/5 Power reduction by global optimization based on fine-grained body biasing». In: *Proceedings of the Custom Integrated Circuits Conference*. IEEE. Sept. 2008, pp. 547–550.

[133] Atsushi Muramatsu et al. «12% Power reduction by within-functional-block fine-grained adaptive dual supply voltage control in logic circuits with 42 voltage domains». In: *ESSCIRC'11: Proceedings of the 37th European Solid-State Circuits Conference*. IEEE. Sept. 2011, pp. 191–194.

[134] Tadashi Yasufuku et al. «24% Power reduction by post-fabrication dual supply voltage control of 64 voltage domains in VDDmin limited ultra low voltage logic circuits». In: *ISQED'12: Thirteenth International Symposium on Quality Electronic Design*. IEEE. Mar. 2012, pp. 586–591.

[135] Pietro Babighian et al. «Post-layout leakage power minimization based on distributed sleep transistor insertion». In: *ISLPED'04: International Symposium on Low Power Electronics and Design*. IEEE. Aug. 2004, pp. 138–143.

[136] Dipankar Saha et al. «Row-Based Dual Vdd Assignment, for a Level Converter Free CSA Design and Its Near-Threshold Operation». In: *Advances in Electrical Engineering* 2014 (July 2014), pp. 1–6.

[137] A. U. Diril et al. «Level-shifter free design of low power dual supply voltage CMOS circuits using dual threshold voltages». In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13.9 (Sept. 2005), pp. 1103–1107.

[138] Lorenzo Seidenari et al. «Deep artwork detection and retrieval for automatic context-aware audio guides». In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 13.3s (2017), p. 35.

[139] Aiguo Wang et al. «A comparative study on human activity recognition using inertial sensors in a smartphone». In: *IEEE Sensors Journal* 16.11 (2016), pp. 4566–4578.

[140] Shuochao Yao et al. «Deepsense: A unified deep learning framework for time-series mobile sensing data processing». In: *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2017, pp. 351–360.

[141] Carole-Jean Wu et al. «Machine learning at facebook: Understanding inference at the edge». In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2019, pp. 331–344.

[142] *Exynos 5 Octa 5422 Processor: Specs, Features.* URL: https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5422/ (visited on 08/11/2019).

[143] Minwoo Jang, Kukhyun Kim, and Kanghee Kim. «The performance analysis of ARM NEON technology for mobile platforms». In: *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*. ACM. 2011, pp. 104–106.

[144] Valentino Peluso et al. «Enabling energy-efficient unsupervised monocular depth estimation on armv7-based platforms». In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 1703–1708.

[145] Christian Szegedy et al. «Going deeper with convolutions». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[146] Young Geun Kim, Joonho Kong, and Sung Woo Chung. «A survey on recent OS-level energy management techniques for mobile processing units». In: *IEEE Transactions on Parallel and Distributed Systems* 29.10 (2018), pp. 2388–2401.

[147] *Odroid-XU4 User Manual*. Hardkernel. URL: https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf.

[148] Han Cai, Ligeng Zhu, and Song Han. «ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware». In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HylVB3AqYm.

[149] Onur Sahin, Paul Thomas Varghese, and Ayse K Coskun. «Just enough is more: Achieving sustainable performance in mobile devices under thermal limitations». In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press. 2015, pp. 839–846.

[150] Samuel Isuwa et al. «TEEM: Online Thermal-and Energy-Efficiency Management on CPU-GPU MPSoCs». In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 438–443.

[151] Ganapati Bhat et al. «Algorithmic optimization of thermal and power management for heterogeneous mobile platforms». In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.3 (2017), pp. 544–557.

[152] Somdip Dey et al. «Edgecoolingmode: An agent based thermal management mechanism for dvfs enabled heterogeneous mpsocs». In: *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. IEEE. 2019, pp. 19–24.

[153] *Linaro Toolchain*. URL: https://www.linaro.org/downloads/ (visited on 08/11/2019).

[154] *TensorFlow Lite Hosted Models*. URL: https://www.tensorflow.org/lite/guide/hosted_models (visited on 08/11/2019).

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.