Inference On The Edge: Performance Analysis of an Image Classification Task Using Off-the-shelf CPUs and Open-Source ConvNets

Valentino Peluso, Roberto G. Rizzo, Antonio Cipolletta, Andrea Calimera
Department of Control and Computer Engineering,
Politecnico di Torino, 10129 Turin, Italy
Mail: andrea.calimera@polito.it

Abstract—**The portability of Convolutional Neural Networks (ConvNets) on the mobile edge of the Internet has proven extremely challenging. Embedded CPUs commonly adopted on portable devices were designed and optimized for different kinds of applications, hence they suffer high latency when dealing with the parallel workload of ConvNets. Reduction techniques playing at the algorithmic level are viable options to improve performance, e.g. topology optimization using alternative forms of convolution and arithmetic relaxation via fixed-point quantization. However, their efficacy is hardware sensitive. This paper provides an overview of these issues using as a case study an image classification task implemented through open-source resources, namely different architectures of MobileNet (v1), scaled, trained and quantized for the ImageNet dataset. In this work, we quantify the accuracy-performance trade-off on a commercial board hosting an ARM Cortex-A big.LITTLE system-on-chip. Experimental results reveal mismatches which arise from the hardware.**

I. Introduction

The creation of a smart society goes through the ability to communicate information efficiently. The role of the ICT is to provide the technology stack by which highly informative data can flow from/to humans/machines autonomously. In this regard, Internet-of-Things (IoT) and Machine-Learning (ML) are twin pillars. IoT technologies enable ubiquitous devices to sample and transmit data over the Internet. An efficient implementation of IoT encompasses fast connectivity across safe mobile networks, as well as accurate sensors and actuators integrated into mobile platforms. ML is the software infrastructure that allows machines to evolve from simple controllers to agents that learn. It includes algorithms, statistical models, and training strategies used to infer abstract information from raw data and to discover relationships among unknown conditions. Together, IoT and ML contribute to the implementation of intelligent services capable of predicting upcoming events or trends and that can take decisions accordingly.

IoT and ML work as vertical layers bonded by a mutual need of deploying the *inference* stage on the *edge*, i.e. on the end-nodes of the Internet, where data are generated. For the IoT, "edge inference" is a means to achieve (i) real-time service response, (ii) less energy waste due to data movement from/to the cloud, (iii) more privacy as data stay local [1]. For ML, it means to improve the training stage as (i) data are processed earlier, before being stacked on huge data-bases from which is hard to distill new knowledge, (ii) distributed sensors become active nodes that contribute with their computational power alleviating the workload of data-centers [2].

Among the existing classes of ML methods, *Deep Learning* (DL) is the one that can benefit most from the edge computing paradigm. Indeed, DL is a supervised learning strategy that requires a huge amount of heterogeneous data and high computational power to work properly. It reached large popularity due to its ability to train feed-forward Deep Neural Networks (DNNs) which have proven very efficient for end-to-end classification of unstructured data, like images, audio, and text. Convolutional Neural Networks (ConvNets), in particular, achieved breakthroughs in several domains. Computer vision, speech recognition, and natural language processing are practical examples where ConvNets exceeded human-level accuracy. Within such domains, to enable inference on the edge means to deploy ConvNets on mobile devices.

The shift towards the edge is not a free lunch, however. ConvNets are computationally expensive: even the simplest topology has millions of parameters to store and billions of multiply-and-accumulate operations to run for a single forward pass. Mobile devices, like smartphones, drones, vehicles, might not have enough resources to host this kind of workload. Hence, the challenge is to accelerate the processing in order to achieve real-time response with a limited budget of energy. There exist several strategies to accomplish this task that span the whole design hierarchy [3], from hardware [4], designing specialized accelerators, e.g. low-power spatial architectures with tightly coupled execution units, to software [5], leveraging algorithmic optimization to reduce the cardinality of the ConvNet still preserving accuracy. High efficiency can be achieved with holistic approaches where hardware and software are vertically co-designed. Unfortunately, this is not always possible as many applications are built upon off-the-shelf components to ensure low design costs, fast prototyping, and short turnaround. The degrees of freedom are therefore limited and the efficiency of algorithmic optimization might be substantially affected.

In this paper, we report on the performance of a practical case study: an image classification task implemented with a popular ConvNet, i.e. MobileNet (v1) by Google [6], deployed on a commercial chipset, i.e. the Samsung Exynos 5422 SoC [7],

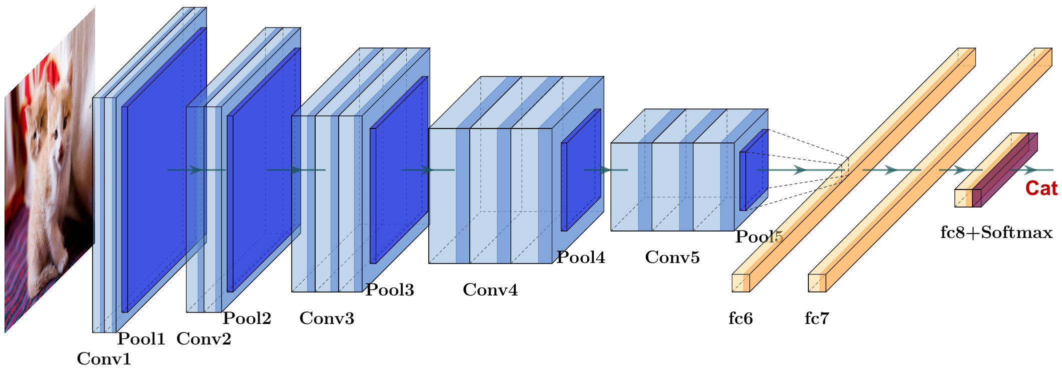


Figure 1: Generic ConvNet Architecture for Image Classification.

through an open-source inference engine, i.e. TensorFlow Lite by Google [8]. The analysis aims at quantifying the impact of optimization at the algorithmic-level, i.e. topology reduction and arithmetic relaxation via fixed-point quantization, on the two commercial mobile CPUs embedded in the adopted SoC, i.e. the Cortex-A7 and the Cortex-A15 cores provided by ARM through the big.LITTLE ARMv7 architecture [9]. Results plotted in the accuracy-latency space reveal different trends depending on the hosting CPU.

II. Image Classification with MobileNet

A. Overview on Deep Convolutional Neural Networks

Image classification is to recognize the content of an image and to classify it with the labels that were available at training time. Several computer vision applications leverage this task to extract semantic information. A practical example comes from social media platforms which run image classification to generate descriptions, captions, tags and, in general, for copywriting. Other emerging applications are automatic indexing of biomedical images, fault detection on manufactured goods or predictive maintenance for tools and machinery in a fab-line, segmentation and object recognition for augmented reality in education and retail.

Starting from the astounding results obtained by Krizhevsky et al. [10] 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 [11]. 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 shows the topology of a generic ConvNet. Convolutional layers (light blue blocks) 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 (mid-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 (dark blue blocks) 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 (yellow blocks) 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.

B. MobileNet: a ConvNet for Mobile Applications

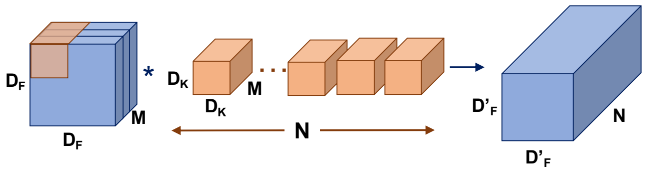
In the beginning, ConvNets were mainly optimized to improve accuracy. This brought to deeper and more complex models with increased size [12] [13] [14]. The growing demand for portable applications pushed designers to focus on more compact and fast ConvNets: MobileNet is an example. It is based on the concept of depth-wise separable convolution originally introduced in [15]. As depicted in Fig. 2, a standard convolution is factorized as two consecutive stages: **(1)** *depthwise convolution*, where each input channel is convolved with each single filter; **(2)** *pointwise convolution*, where 1×1 convolutions combine the outputs generated by the depthwise convolution. The result is an approximation of the standard convolution with fewer arithmetic operations.

For a standard convolution, assuming stride one and padding, the number of multiplications P is given as follows:

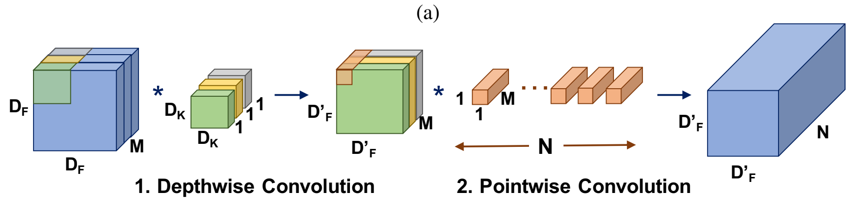
$$P_{std} = D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{1}$$

with M the number of input channels, N the number of output channels, $D_K \times D_K$ the kernel size, and $D_F \times D_F$ the size of the feature map. For a depthwise separable convolution the contributions are two, that of the depthwise stage:

$$P_{dw} = D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{2}$$



(a)



(b)

Figure 2: Standard Convolution (a) vs. Depthwise Separable Convolution (b).

and that of pointwise stage:

$$P_{pw} = M \cdot N \cdot D_F \cdot D_F \tag{3}$$

The total number is the sum $P_{dws} = P_{dw} + P_{pw}$. Therefore, the resulting reduction factor w.r.t. standard convolution is:

$$\sigma = \frac{P_{dws}}{P_{std}} = \frac{1}{N} + \frac{1}{D_K^2} \tag{4}$$

As reported in [6], MobileNets with 3×3 depthwise convolutions reach compression in the range $8\times$-$9\times$.
In this work, we make use of the pre-trained models available in the TensorFlow Hosted Models [16] repository for an input resolution of 224×224.

C. ImageNet

The ImageNet dataset [17] collects more than 15 millions high-resolution images classified with roughly 22,000 classes. Since 2010, images have been collected from the web and labeled by humans using *Amazon Mechanical Turk* crowd-sourcing tool. As part of the *Pascal Visual Object Challenge*, an annual competition called *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC) adopts the ImageNet as test-bench. There exist different versions of ImageNet, each of them of a different size. The one used to train MobileNet (v1) is the ILSVRC12. It consists of 1.2 million samples distributed across 1000 different classes; the validation set contains 50,000 images, while the test set is made of 100,000 images. For the training stage of MobileNet, images are resized to 224×224 by cropping the center of the original versions.

III. Towards the Edge: MobileNet Optimization

Even if MobileNet is already a lightweight ConvNet, mobile applications may call for additional optimization to achieve higher performance and/or meet stringent energy budgets. The two orthogonal techniques analyzed in this work play at different levels of granularity: *(i)* topology-level, *(ii)* weight-level. The former applies a layer re-sizing, the latter plays with the arithmetic precision of the inner parameters leveraging a fixed-point quantization.

Topology-level optimization MobileNet gives the possibility to tune a hyper-parameter $\alpha \in (0, 1]$ which defines the layers width multiplier, being the baseline model defined with α=1. Such parameter defines the ratio of input and output channels dropped within each depthwise convolution layer. For instance, given a layer with M input channels and N output channels, the scaled model counts $\alpha \cdot M$ and $\alpha \cdot N$ channels respectively. The overall number of multiplications within a depthwise separalable convolutional is as follows:

$$P_{dws} = D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F \tag{5}$$

Pre-trained models are available for $\alpha = \{0.25, 0.5, 0.75, 1.0\}$. Width multiplier has the effect of reducing the number of multiplications and parameters by roughly α^2. To notice that all depthwise convolutions share the same α.

Weight-level optimization Quantization via fixed-point representation is commonly used to shrink the model size. The pre-trained weights of a floating-point ConvNet can be mapped into a discrete space using integer representations. This arithmetic shift enables to: *(i)* reduce the memory footprint by a factor proportional to the number of scaled bit; *(ii)* accelerate inference thanks to lower memory bandwidth. The bit-width may range from 16- to 2-bit [18]. However, there is a wide consensus that 8-bit is enough to prevent accuracy loss.

Several quantization methods and scaling schemes do exist, each with different performance and accuracy. The one adopted on the pre-trained MobileNets publicly available in [16] is *asymmetric* [19]. Such scheme maps real numbers r to integers q, as depicted in Fig. 3. A more formal formulation is given in the following equation:

$$q = \lfloor S \cdot r + Z \rceil \tag{6}$$

where the $\lfloor \cdot \rceil$ operator refers to nearest integer rounding. The min./max. values of the floating-point distribution are mapped to the min./max. values of the integer range. This is obtained by means of two parameters: S and Z. The former is the *scale factor* (or slope), the latter is *quantization offset* (or zero-point). The scale factor S is a positive real number defined as:

$$S = \frac{q_{\max} - q_{\min}}{r_{\max} - r_{\min}} \tag{7}$$

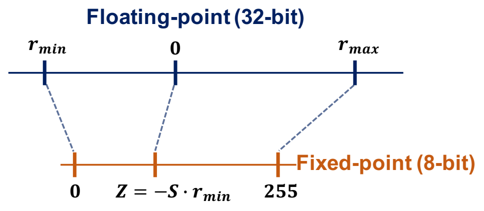


Figure 3: Asymmetric Quantization scheme.

with $q_{\min} = 0$, $q_{\max} = 2^n - 1$, and n the bit-width of the integer representation. The offset Z is the quantized value q corresponding to the real value 0. That implies that the real value $r = 0.0$ is mapped onto an exact fixed-point number. This is an important aspect as ConvNets may show high sparsity.
To notice that all the layers share the same bit-width n=8, while the quantization scheme is applied channel-wise, namely, each filter within each convolutional layer has its own quantization scale factor S and offset Z. This is paramount as filters show uneven weights distributions [20].

IV. The Embedded Platform

A. Hardware Specifications

The commercial board adopted as the hardware test-bench is the ODROID-XU4 developed by Hardkernel. It integrates the Samsung Exynos 5422 System-on-Chip (SoC) powered with a 32-bit ARM big.LITTLE architecture widely used in high-end embedded platforms. As reported in Table I, the SoC is an octa-core split into two quad-core clusters: `big` and `LITTE`. The former is the *high-performance* cluster consisting of four Cortex-A15 cores running at a maximum frequency of 2 GHz; the latter is the *low-power* cluster with four Cortex-A7 cores running at a maximum frequency of 1.4 GHz. A private 32 kB L1 cache is available for each single core, while the L2 cache is shared by cores belonging to the same cluster: 2 MB for `big`, 512 kB for `LITTLE`. Lastly, 2 GB of on-chip RAM is shared between the two clusters.
It is worth mentioning that the two clusters are used separately in this work. Indeed, the goal is to assess the effect of algorithmic-level optimization on different architectures and not to measure the maximum performance of the whole SoC.

B. Arm NEON Technology

Single-Instruction Multiple Data (SIMD) refers to the class of architectures which rely on multiple processing elements to accelerate highly parallel workloads, like ConvNets. With one single instruction and multiple pieces of data loaded up, the same operation is executed over all the data simultaneously obtaining performance boost.
Both Cortex-A15 and Cortex-A7 come with the NEON technology, a 128-bit SIMD data-path designed to give support to multimedia applications. This NEON unit is programmable and can support vector operations over different types of data: signed/unsigned integers with 8, 16, or 32 bit-width, and 32-bit single-precision floating-point. Obviously, the maximum parallelism reduces with the increase of the bit-width. A 32×64-bit register file works as the local memory which can

Table I: Hardware specifications of the Samsung Exynos 5422 SoC integrated into the ODROID-XU4 board.

Cluster	**CPUs**	**Freq.**	**L1**	**L2**	**RAM**
big	4×A15	2 GHz	4×32 kB	2 MB	2 GB
LITTLE	4×A7	1.4 GHz	4×32 kB	512 kB	

host up to 256 (8-bit) integers or, alternatively, 64 (32-bit) floats. This flexibility allows an efficient handling of vectors and better utilization of the bandwidth.

V. Experimental Results

A joint combination of the optimization methods introduced in Section III gives developers a unique opportunity to balance memory footprint, performance and prediction accuracy, on the base of system requirements and/or the available hardware resources. The results collected in this section quantify such metrics through a trade-off analysis. The contents are organized as follows. First, we introduce the on-board environment used to collect the performance statistics. Second, we provide a memory-accuracy analysis of the pre-trained MobileNet models; this analysis is hardware-independent, namely it holds the same whatever the hosting CPU is. Third, we show the performance-accuracy analysis for the target architectures introduced in Section IV. Though is intuitive that optimized, hence less complex MobileNets are faster and less accurate, our analysis aims to identify solutions that are best in either accuracy or performance, that is, identify the optimal settings which can be deployed to cover different operating constraints.

A. Inference Engine And The On-board Environment

TensorFlow Lite (TFL) by Google is an inference engine, i.e. a collection of software routines for deep learning, highly optimized to run tensor-graphs, like MobileNet, on the Cortex-A architecture. More specifically, the convolution operators leverage the ARM NEON instruction-set, for both the floating-point and the integer implementations. TFL provides users with an abstract interface that loads pre-trained models in the *tflite* format. In our experiments, we used TensorFlow Lite version 1.14, cross-compiled using the GNU ARM Embedded Toolchain (version 6.5) [21].
TFL integrates a benchmarking utility, called *TensorFlow Lite Model Benchmark*, to facilitate the measurement of the inference time on the target device. Specifically, it assembles random inputs and collects the latency statistics. The returned value is the average execution time recorded on-board. In our setup, we iterated over 100 runs, interleaved by a 2-second pause to avoid thermal throttling. Since the first execution needs more time to properly configure the model and allocate memory, a warm-up run is executed before starting the actual measurements. All the experiments are run in single-thread mode.

B. Memory-Accuracy Analysis

As already mentioned in the previous sections, different versions of MobileNet are available in the TensorFlow Hosted Models [16] repository. Those used in this work are sized

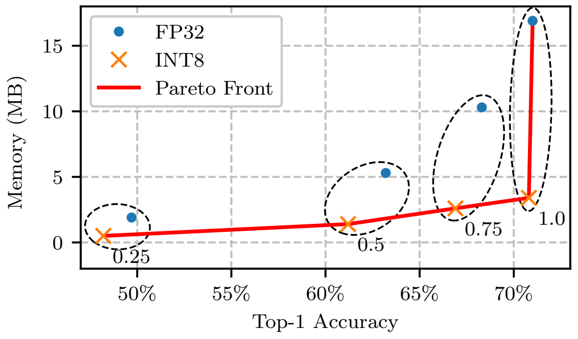


Figure 4: Accuracy Memory trade-off for different α and arithmetic precision for MobileNet (v1).

to handle 224×224 input images. We consider four different topologies, namely $\alpha = \{0.25, 0.5, 0.75, 1.0\}$, and two arithmetic representations, the original floating-point (FP32) and the 8-bit fixed-point (INT8). Overall, there are 8 different ConvNets ported onto two CPUs for a total of 16 cases.

The plot drawn in Fig. 4 maps each configuration in the memory-accuracy space. Accuracy refers to the top-1 classification accuracy measured on the ImageNet validation set. Memory refers to the size of the *tflite* file containing the data structures needed to port the model on-chip, including the network weights and the topology description. Within the plot, blue dots are for FP32, while orange crosses are for INT8; configurations with the same topology, i.e. same α, are grouped within an ellipse (dashed black line). The Pareto curve (solid red line) connects the optimal configurations.

Several considerations can be drawn. Memory and accuracy are monotone w.r.t. both α and arithmetic precision, yet with a different slope. Quantization is very efficient. For any given α, the memory reduction from FP32 to INT8 is $\times 3.7$ with limited accuracy loss (best-case 0.2% for α=1.0, worst-case 2.0% for α=0.5). An ideal FP32-to-INT8 scaling should return $\times 4$ memory reduction, which is not the case in practice as the model flashed on-board also contains auxiliary data structures, such as the network topology and other layers' parameters, which do not scale with the arithmetic precision.

Even more interesting, quantization dominates over topology scaling. For instance, the configuration α=1.0-INT8 has 2.5% higher accuracy and 6.9 MB less memory than α=0.75-FP32. FP32 and INT8 accuracy get very close for larger values of α. As a result, it is unlikely to use the configuration α=1.0-FP32 as it gains 0.2% in accuracy w.r.t. α=1.0-INT8 at the cost of 13.5 MB of memory space. By contrast, the accuracy distance between FP32 and INT8 increases as α gets smaller.

For the most lightweight model, i.e. α=0.25, the accuracy reaches very low values: a mere 49.7% for FP32. However, such low accuracy configurations might represent a baseline for simpler applications, e.g. classification tasks with fewer classes, run on less powerful CPUs.

As a final remark, it is important to highlight that this analysis holds for the ImageNet dataset while different trends might be observed for other kinds of applications. The same consideration does hold for results reported in the next subsection.

Table II: Frames per second for different MobileNet configurations on the ODROID-XU4 board.

	A15 `big`		**A7** `LITTLE`	
α	**FP32**	**INT8**	**FP32**	**INT8**
0.25	43.86	56.11	8.87	14.24
0.50	14.52	24.19	2.92	5.97
0.75	6.81	13.63	1.45	3.24
1.00	4.16	8.96	0.87	2.10

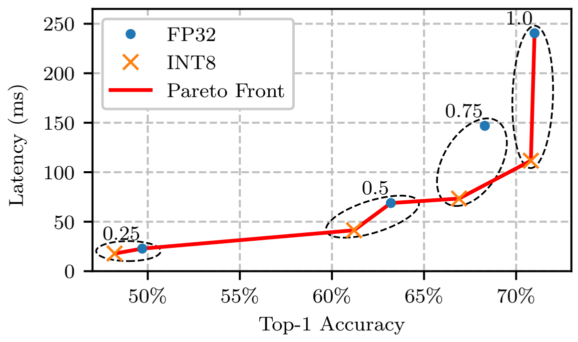
C. Performance-Accuracy Analysis

This section first considers the performance gain brought by algorithmic optimization on different hardware architectures, then it focuses on a study of optimality across the latency-accuracy space.

a) Performance Assessment: Table II collects the average frames per second (FPS) measured for the 16 configurations under analysis, namely $\alpha = \{0.25, 0.5, 0.75, 1.0\}$ using FP32 and INT8, deployed on the two target architecture, A15 and A7 cores. Within the A15 core, topology scaling from α=1.0 to α=0.25 gets impressive speed-up: $\times 10.55$ for FP32, $\times 6.26$ for INT8. α=0.25 ensures real-time performance (>30 FPS) for both FP32 and INT8. Obviously, the A7 core runs at a lower frequency, hence it cannot achieve that throughput. When considering the average over the four values of α, for FP32 (INT8) the A15 core is $4.8\times$ ($4.1\times$) faster. Despite this absolute differences, the speed-up trend is quite similar for both the cores.

The use of quantized weights does not just reduce memory (as shown in Fig. 4), but it also improves latency. However, the performance gain of INT8 w.r.t. FP32 is strictly related to the value of α and the underlying hardware architecture. For the A15 core, the savings achieved by quantization range from $1.27\times$ (row α=0.25) to $2.21\times$ (row α=1.0); for the A7 core, they go from $1.60\times$ (row α=0.25) to $2.41\times$ (row α=1.0). As per these results, quantization looks more efficient on larger networks deployed on small cores. Larger networks have more convolutional filters, therefore the percentage of those parameters that benefit from precision scaling is larger. Concerning the dependence from the core size, is the cache size to play the key role. It is known that the larger the cache, the lower the number of accesses to the main memory. To be considered that single access to the main memory has a latency several times longer than that of an arithmetic operation. The use of a scaled bit-width increases the amount of data that resides in cache. This helps to improve performance. Obviously, reduced cache size has a larger impact on performance. That is why an inference run on the smaller A7 core is more sensitive to quantization.

b) Pareto Analysis: The plots in Fig. 5 maps each configuration in the latency-accuracy space. Labels and markers have the same meaning of those used in Fig. 4. The most evident result is that the optimal configurations (Pareto curve - solid red line) do change depending on the underlying hardware architecture. The configuration α=0.5-FP32 is a representative case: it is Pareto for the A15 core, but not for the A7. That suggests algorithmic optimization unevenly improve latency



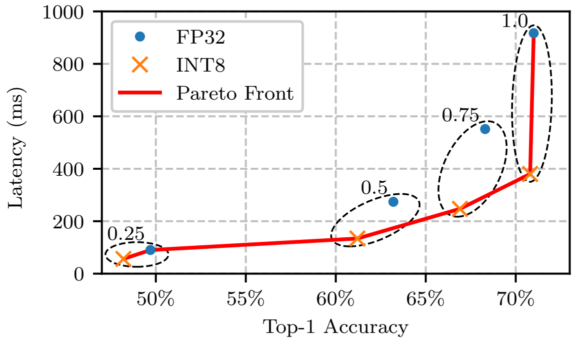


Figure 5: Accuracy Latency trade-off for different α and arithmetic precision for MobileNet (v1): A15 (left), A7 (right)

depending on the hardware characteristics. Unlike for memory, where INT8 configurations dominate (please refer to Fig. 4), FP32 configurations may turn to be optimal when performance comes into play. An example is given by the configuration α=0.5-FP32 which is a Pareto point for the A15 core.
This analysis suggests that moving across the latency-accuracy space is not trivial and it asks for an accurate optimality assessment at first. Let us consider a practical example. To improve the accuracy of the configuration α=0.75-INT8, it is more convenient to scale up the network, namely to move on α=1.0-INT8, rather than increasing the arithmetic precision as one would intuitively think. In fact, α=1.0-INT8 shows higher accuracy and lower latency for both A15 and A7. Overall, optimal configurations cannot be defined *a-priori* and a proper balance between topology organization and precision scaling is needed. This is in line with existing works which propose concurrent design exploration via Neural Architecture Search.

VI. Conclusions

There exist multiple design choices and optimization strategies that can help to enable inference on the edge. Though all them attack the same problem, namely to make ConvNets smaller and faster, they play with different parameters and metrics. Therefore, the effect of a joint combination should be carefully weighed and balanced also depending on the characteristics of the hosting hardware. The issue becomes quite relevant when resources are severely limited, as for in the mobile segment. This work assessed this aspect by proposing a parametric study conducted for off-the-shelf components and ready-to-use tools, i.e. different versions of MobileNet (v1), trained on Imagenet, optimized with TensorFlow Lite, and deployed on two commercial ARM CPUs. The collected results show the relevance of the above problem providing a first reference to developers who approach the problem.

Acknowledgment

Co-funded by Compagnia di San Paolo.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.
[3] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
[4] G. Santoro, M. R. Casu, V. Peluso, A. Calimera, and M. Alioto, "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.
[5] V. Peluso and A. Calimera, "Scalable-effort convnets for multilevel classification," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.
[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
[7] Exynos 5 octa 5422 processor: Specs, features. [Online]. Available: https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5422
[8] Tensorflow lite. [Online]. Available: https://www.tensorflow.org/lite
[9] big.little technology: The future of mobile. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[11] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[14] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.
[16] Tensorflow lite hosted models. [Online]. Available: https://www.tensorflow.org/lite/guide/hosted_models
[17] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.
[18] Y. G. L. X. Y. C. Aojun Zhou, Anbang Yao, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *International Conference on Learning Representations,ICLR2017*, 2017.
[19] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.
[20] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.
[21] Linaro toolchain. [Online]. Available: https://www.linaro.org/downloads/