A Flexible In-Memory Computing Architecture for Heterogeneously Quantized CNNs

Flavio Ponzina, Marco Rios, Giovanni Ansaloni, Alexandre Levisse and David Atienza
Embedded Systems Laboratory (ESL), EPFL, Switzerland
{flavio.ponzina, marco.rios, giovanni.ansaloni, alexandre.levisse, david.atienza}@epfl.ch

***Abstract*—Inferences using Convolutional Neural Networks (CNNs) are resource and energy intensive. Therefore, their execution on highly constrained edge devices demands the careful co-optimization of algorithms and hardware. Addressing this challenge, in this paper we present a flexible In-Memory Computing (IMC) architecture and circuit, able to scale data representations to varying bitwidths at run-time, while ensuring high level of parallelism and requiring low area. Moreover, we introduce a novel optimization heuristic, which tailors the quantization level in each CNN layer according to workloads and robustness considerations. We investigate the performance, accuracy and energy requirements of our co-design approach on CNNs of varying sizes, obtaining up to 76.2% increases in efficiency and up to 75.6% reductions in run-time with respect to fixed-bitwidth alternatives, for negligible accuracy degradation.**

***Index Terms*—In-Memory Computing, CNN, Quantization, Edge computing**

I. Introduction

Edge Artificial Intelligence (edgeAI) is fostering a revolution in the computing landscape [1]. It offers real-time and personalized solutions in a plethora of scenarios, ranging from health monitoring to augmented reality. EdgeAI devices acquire inputs (video feeds or bio-signals among others) from their environment and interpret them locally. However, computationally demanding algorithms such as Convolutional Neural Network (CNN) models are necessary in order to infer highly abstract features (e.g.: objects, acute health conditions).

CNN inference in edgeAI must be performed within tight performance and energy constraints, as edge devices usually operates for long periods while relying on small energy sources (battery and/or harvesters). Efficiency for edgeAI is therefore key and, as summarized in Section II, a very active research topic. One avenue towards energy minimization centers on software-level optimizations, such as the adoption of low-bitwidth data representations (quantization) and the elision of computing paths with a low influence on final results (pruning) [2]. Other efforts focus instead on hardware specialization, tailoring computing resources to efficiently support CNN workloads. In this field, In-Memory Computing (IMC) is emerging as a very promising approach [3]. By enabling arithmetic operations inside the memory hierarchy, IMC induces a dramatic increase in the computation-to-communication ratio. Additionally, IMC enables a high degree of parallelism when executing the many multiply-accumulate (MAC) operations at the heart of CNN inferences. We observe that the above-mentioned hardware and software approaches, which at present have mainly be studied in isolation, are instead inter-dependent. On one side, IMC is particularly amenable towards quantization, as decreases in bitwidths lead to more efficient in-memory MAC operations. On the other, effective solutions at the circuit level are required to maximise the benefits deriving from pruning and quantization.

This work has been partially supported by the EC H2020 WiPLASH project (GA No. 863337), the ERC Consolidator grant COMPUSAPIEN (GA No. 725657), and by the Swiss NSF ML-Edge Project (GA No. 182009).

Motivated by this observation, we herein present a flexible IMC architecture able to accommodate different bitwidths. The variable bitwidth is used to represent the weights encoding a CNN model, as well as the activations propagating through its layers. The architecture supports a flexible degree of word-level parallelism for multiplicands and a fine-grained tuning of the bitwidth of multiplier values. Moreover, we present a heuristic approach for the hardware-aware pruning and quantization of CNN models, hence showcasing its benefits in terms of run-time and energy efficiency. The heuristic approach favourably compares with respect to unscalable exhaustive explorations, which are unpractical for large CNNs.

All in all, our contributions are summarized as follows:

- We present a novel IMC approach able to effectively execute parallel MAC operations among signed operands of selectable bitwidth, both in the multiplicand and the multiplier, detailing the devised circuit-level solutions.
- We describe how convolutional and fully connected layers characterizing CNN models are effectively mapped on the IMC array architecture, maximising operations parallelism and data reuse.
- We introduce a hardware-aware optimization methodology for reducing the workload and memory footprint of CNN models, which iteratively quantizes weights and activations on a per-layer and per-filter basis.
- We showcase that only a modest hardware overhead (6.4%) is induced by supporting variable bitwidths. On the other hand, important energy gains are achieved by tailored CNN models (76.19%, 62.80%, 69.72% for AlexNet [4], VGG16 [5] and Mobilnet [6], respectively) when processing the CIFAR-100 dataset, for a maximum accuracy degradation of 1%,

The paper proceeds as follows: Section II provides the state of the art of IMC architectures and CNNs. Section III presents our flexible IMC architecture, detailing the circuit-level solu-

tions adopted to support different quantization levels for the weights and activations. Section IV presents our resource-aware CNN optimization method. Section V experimentally evaluates the efficiency and performance benefits deriving from our strategy. Finally, Section VI concludes the paper.

II. State of the Art

A. CNN Optimization for Edge Computing

CNNs are a class of neural networks having layers implemented as convolutions between input activations and filters, which store trained parameters. A pioneering implementation is LeNet-5 [7], which presents three convolutional (CONV) layers, followed by a single fully connected (FC) one. More recent topologies feature a higher number of layers: as examples, AlexNet [4] is composed of five CONV layers and three FC ones, while Resnet variants can have tens of CONV layers [8]. Such evolution allows state of the art CNN to tackle very complex problems, e.g., performing object recognition on datasets with hundreds of classes [9] [10]. On the other side of the coin, it also brings an explosion in the number of required parameters and arithmetic operations, with ResNet-50 having 25 million weights and almost 4 billion multiply-accumulate (MAC) operations. More lightweight alternatives, such as MobileNet [6], still require 4 million parameters and 800 million MACs.

Supporting such high workloads in edge devices with limited computational capabilities and energy budgets is a major challenge. Addressing it, several works encode weights and activations in efficient fixed-point formats. Crucially, when performed judiciously, the entailed *quantization* induces little or no loss in accuracy, even when employing few bits [11]. Moreover, entire computational paths (e.g., convolution filters) having little contribution to results can be entirely removed, an approach termed *pruning* in literature [12].

B. In-Memory Computing Architectures for Edge CNNs

A different avenue to increase the efficiency of CNN inference is to employ specialized hardware [13]. IMC solutions are especially promising in this context [3]. They cope with the high memory and computational intensity of CNNs, leveraging the regular computing pattern and the data reuse opportunities of inference. Moreover, IMC solutions minimize data transfers and parallelize executions.

IMC accelerators can be broadly divided in two categories. Crosspoint architectures employ meshes of variable resistances to encode weights and arrays of analog-to-digital converters to read-back results [14]. They can efficiently perform matrix-vector multiplications, but are prone to noise due to manufacturing variability, temperature and voltage fluctuations.

A second strategy, that we also consider in this paper, is that of bit-line computing [15]. This approach can be readily integrated in existing memory systems and hierarchies by adding the capability to concurrently activate multiple memory words, and employing sense amplifiers to perform bit-wise operations. Bit-line computing on SRAM array dictates that by activating two, or more, wordlines at the same cycle clock,

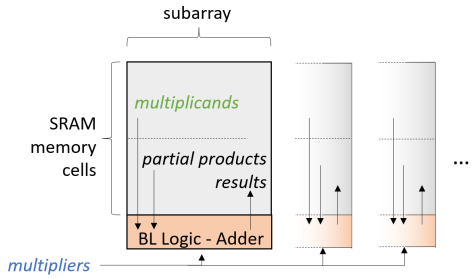


Fig. 1. High-level block scheme of the IMC architectures. Memory cells store multiplicand inputs, partial products and MAC outputs, computed by the IMC

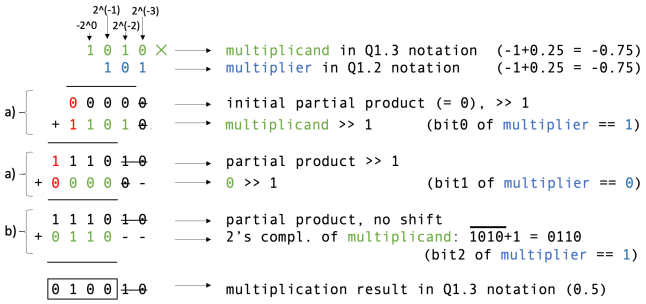


Fig. 2. Multiplication between 2's complement numbers in fixed point format. Barred bits are dropped. All right-shifts are intended as arithmetic shifts.

the discharge pattern on the bitlines behaves as the logic operations AND and NOR. Recent works [16] [17] have shown that such operations can be reliably performed at high clock frequencies if the activated wordlines belong to different Local Groups (LGs) inside the SRAM array. Inputs are read in the LG Periphery (LGP) circuit, which features sense amplifiers and read/write ports. Addition operations are then derived from bit-wise ones using in-memory adders embedded in the IMC array. Finally, multiplications and MACs are performed as sequences of addition and shift-add operations.

The effectiveness of bitline computing is highly influenced by quantization and pruning, as they both determine the number of performed shift-add operations. Nonetheless, current works do not explore this interdependence in detail. As an example, [16] adopts a fixed 16-bits representation for activations and an 8-bit one for weights. We instead show that important efficiency gains can be harnessed when employing more aggressive stances. In particular, we highlight that heterogeneous per-layer schemes can be supported with little hardware overhead.

While integer representations have been considered as a basis for IMC operations [18], these are very prone to overflows, which can drastically impact accuracy. Our architecture instead supports fixed-point, 2's complement signed operand with a single integer bit, implementing MACs using in-memory arithmetic right-shifts.

III. IMC Architecture

Our architecture is structured as a SRAM memory array, organised in several subarrays, each comprising Local Groups (LGs). Such arrangement allows to perform in parallel multiple

operations among operands in the same subarrays and different LGs, computing convolutions (for CONV layers) and linear combinations (for FC layers) as a sequence of MAC operations. Importantly, only one bit of the multiplier is required to compute each MAC partial product. As illustrated in Fig. 1, we leverage this characteristic by storing only one of the operands (activation or weights) and the results in memory subarrays, while the proper bit of the other operand value is streamed as an input. This arrangement allows to a) effectively parallelize execution in multiple IMC subarrays, performing multiple multiplications concurrently and b) support arbitrary multiplier bitwidths.

At the circuit level, local groups in subarrays have their own set of local bitlines (LBL and LBLb). The LBLs are then connected to Global BitLines (GBL) through a read/write port, depending on the operation. To read a word or perform an IMC operation, both LBLs and GBLs are pre-charged. Then, a voltage-based sense amplifier inside the LG asserts a logic value once the wordline is activated. This signal controls whether the GBL is discharged or not. Write operations are performed by setting up the data on the global bitlines and activating the wordline and the write port at the LG Periphery (LGP).

A. In-Memory Operations

The proposed IMC architecture supports MACs in the form ($\sum_{i,j} a_i \times b_j$), in fixed-point format and between values in the range $[-1, 1)$, encoded in 2's complement. Values are represented with 1 bit for the integer part, and a varying number n of bits for the fractional part (a format commonly indicated as $Q1.n$). Fig. 2 provides a step-by-step illustration of the operations performed to multiply the multiplicand A and multiplier B. The multiplicand and the result have the same bitwidth. Such procedure avoids overflows, even if it potentially induces a rounding error[1]. For each multiplication, operation (a) is performed for every bit on the multiplier but the MSb, where operation (b) is used:

(a) Addition of two operands, arithmetically right-shifted by one bit. One operand is the partial product, while the second is either 0 or the multiplicand, depending on the multiplier bit.

(b) Addition between two operands. The partial product and either 0 or the 2's complement of the multiplicand. The 2's complement operations consists in negating all the bits and setting the carry-in signal to 1.

The 2's complement multiplication requires extra operations compared to the unsigned multiplication. Three memory operations are required to calculate a single partial product (two right-shifts and one addition). To avoid slowing down the multiplication, the IMC architecture needs to support two new operations: in-situ right-shift and in-situ negation. These two operations are used to speed up the calculation of partial products as they are performed simultaneously with the addition.

[1]The exact multiplication result in the illustrated case would be 0.5625, which cannot be represented in Q1.3 format.

We implemented an embedded shift structure as in [18], and depicted in Fig. 3(a). The read ports inside the LGP are extended so the output from the adjacent sense amplifier controls the discharge of the GBL (red arrows). Fig. 2 represents the shifted bit in red. It can be noticed that the right-shifted bit is not always zero. The MSb value is right-shifted and also kept in the MSb when the shifted value is represented in 2's complement.

The 2's complement of multiplicands is computed by negating their value inside LGs. To do so, we employ two multiplexers to invert the LBL and LBLb signals, as illustrated in Fig. 3(a) with blue arrows. Additionally, the carry in of the adder in the subarray periphery is set to 1.

In-situ 2's complements and shifts are chained to additions, without requiring additional clock cycles. Therefore, all in-memory operations require two clock periods, one to read the operands and perform the in-memory operation computation, and one to write back the result.

B. In-Memory Word Level Parallelism

Our architecture allows to store, in each memory location, either a single 16-bit word or two 8-bit ones. The latter configuration is adopted in layers more resilient to quantization errors. In this case, the workload is effectively halved, as two multiplications are deployed inside the subarray simultaneously. The read and write operations are executed equally regardless of the mode. Conversely, the in-memory right-shift and addition require circuit-level solutions to enable word level parallelism.

Fig. 3(b) presents the circuit that enables the selection of one of these configurations. We exemplify it here with one 8-bits (bit<n+7:n>) word. In order to achieve 16-bits operations, two such blocks are assembled. Two multiplexers are employed in each block to control the mode selection. Considering a subarray that stores 16 bits words (bit<15:0>), the table presented in Fig. 3(c) shows the outputs of the multiplexers M1 and M2 for each of the two-byte arrays (bit<7:0> and bit<15:8>). The multiplexer M1 controls the shift-right input on the MSb. Therefore, the bit shifted in the bit<15> is always s<15>. On the other hand, the shifted bit in the bit<7> is either s<7> for the 8-bits mode, or s<8> for the 16-bits mode. The carry in on the LSb is the CIN signal, which is 0 for a conventional addition or 1 for an addition with the 2's complement. Therefore, the carry in of the bit<8> is either the C<7> for the 16 bits mode or CIN for the 8-bits mode.

TABLE I
ASSIGNMENT OF OPERANDS FOR FULLY CONNECTED AND CONVOLUTIONAL LAYERS

Operand \ Layer type	**CONV**	**FC**
Multiplicand (stored in IMC subarrays)	Activations	Weights
Multiplier (broadcasted to IMC subarrays)	Weights	Activations

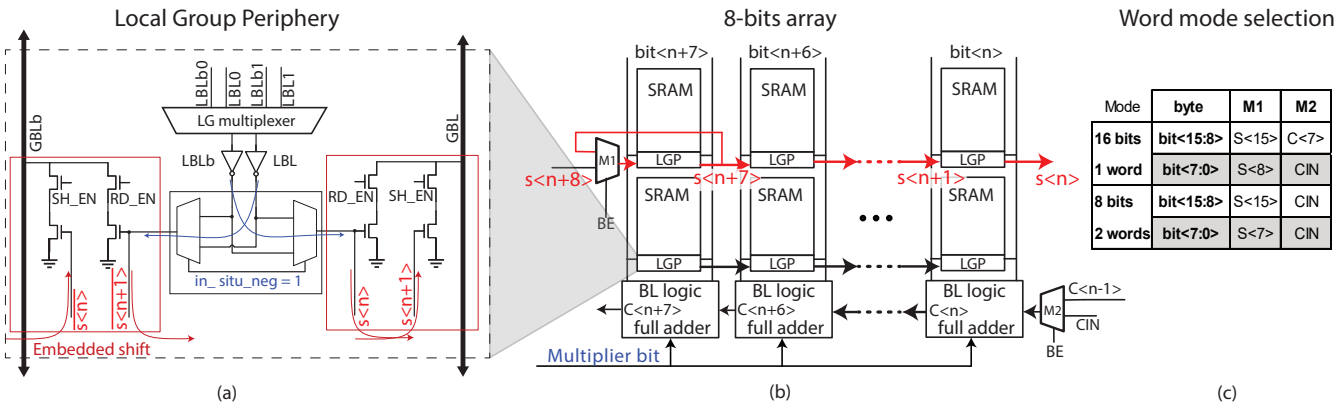


Fig. 3. (a) The Local Group Periphery (LGP) block schematic, blue arrows represent the data path for the in-situ negation operation. Red arrows show the functionality of the in-situ right-shift. (b) 8-bits array of our IMC architecture. (c) Multiplexers M1 and M2 output for the two word modes: 1 word of 16 bits or 2 words of 8 bits.

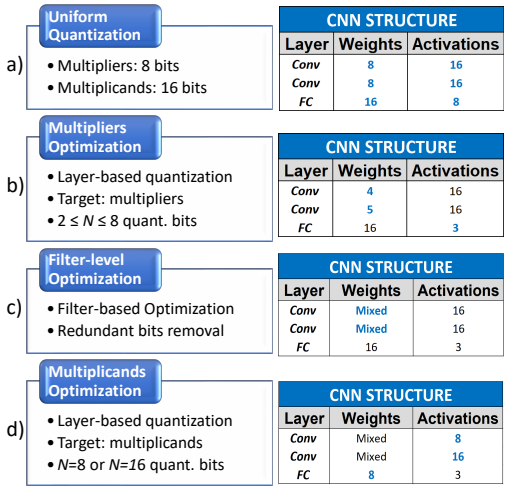


Fig. 4. Workload-aware quantization and pruning methodology (left). Running example (right).

IV. Resource-aware Quantization and Pruning

We exploit the flexibility of the IMC architecture introduced in the previous section to optimize the run-time efficiency of both CONV and FC layers. As indicated in Table I, in the case of CONV layers, we consider input activations as multiplicands (stored in IMC subarrays) and weights as multipliers (broadcasted to perform parallel MAC operations). Conversely, for FC layers we employ weights as multiplicands (since each weight is used once), while activations are broadcasted as multipliers. We then optimize the bitwidth of both multiplicands and multipliers following the scheme in Fig. 4. Without loss of generality, in the figure we consider a running example of a CNN with two CONV and one FC layers.

The starting point of our flow are homogeneously quantized CNNs having 16-bits multiplicands and 8-bits multipliers (Fig. 4-a). As shown in [15] and [16], they have indistinguishable accuracies with respect to floating point implementations and act as baselines for our comparisons. In the first optimization step (Fig. 4-b), the multiplier bitwidth is independently tailored for each layer. To this end, we attempt to reduce the multiplier bitwidths starting from the layer having the highest number of MAC operations (and therefore to higher potential for savings). We then retrain the network for a small number of epochs and check the obtained accuracy of the new configuration. If the accuracy degradation exceeds a user-defined threshold we backtrack (we considered 1% and 5% maximum degradations for the experiments in Section V) . In a similar fashion, we then iteratively target the layers having the second, third etc. most numerous MAC operations. Once all layers have been processed once, we further try to reduce the multiplier bitwidth of the highest-workload layer from which we haven't previously backtracked. The iteration continues until no further bitwidth reductions are possible.

Focusing on CONV layers, we observe that a large amount of CNN filters do not use the entire value range when representing weights, especially after the above-mentioned aggressive quantization. As illustrated in Fig. 4-c, we therefore drop, without loss of accuracy, the most significant bits (MSbs) on a per-filter bases if allowed by weight ranges, correspondingly scaling convolution outputs. As an example, if the value range of a filter is $\subset [-0.25, 0.25)$, 2 MSbs can be dropped, and outputs should be divided by $2^{Dropped\_bits} = 4$. In this stage, the convolution of filters having all weights equal to zero can be avoided. The last step of the optimization flow (Fig. 4-d) centers on the reduction of the bitwidth of multipliers. It leverages the word-level parallelism available in the memory arrays of the IMC architecture. We tailor them per-layer, with an approach similar to the one adopted for multiplicands. However, in this case, we only admit 16-bit and 8-bits widths, which correspond to one and two values per memory word, respectively[2].

V. Results

A. Experimental Setup

As a test vehicle for our experiments, we considered an IMC architecture composed by subarrays of 640 bytes each and interfaced to an external read/write port using an H

[2]While in principle our approach could be extended to 4 values of 4 bits and 8 values of 2 bits per word, such settings would incur in unacceptably large accuracy degradations.

tree interconnect. With a methodology analogous to [17], we implemented the IMC architecture as a full custom design and performed hspice energy and timing characterization. Targeting a 28nm CMOS technology from TSMC, the architecture can operate at a frequency of 2.2 GHz. The subarray requires 376pJ for reading a 16-bit word, and 414pJ energy for writing it. An in-memory shift-add operation requires 381pJ. The subarray has an area of $1240\mu m^2$, of which 26.5% is used for the in-memory computing logic and local group periphery circuit. 6.4% of the total area is used to implement the in-situ negation, embedded shift and wordline parallelism. Finally, 67% of the area is filled up by the SRAM cells.

Similarly to [19], we employed a cycle-accurate simulator[3] to emulate the execution of convolutional and fully connected layers. The simulator tiles the input of each layer and distributes the tiles to different subarrays, employing multiple rounds if the number of tiles exceeds the number of subarrays. We assume a data transfer bandwidth of one word per cycle for writing inputs and reading back results. IMC operations require two cycles: one to perform the IMC operation and one to write the result. These are executed in parallel on each subarray.

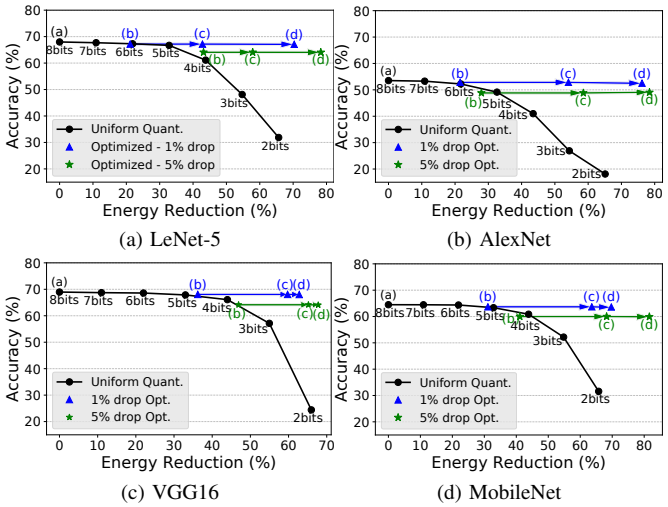
We evaluated our architecture and optimization framework on several benchmarks: we tested LeNet-5 [7] on CIFAR-10 [9] and AlexNet [4], VGG16 [5] and MobileNet [6] on the CIFAR-100 dataset [9]. Accuracy values for various CNNs and optimization levels were retrieved using PyTorch [20] and the quantization functions described in [21]. CNNs are first trained using floating point precision for 200 epochs, obtaining accuracies in line with the state of the art. Similarly to [16], models are then homogeneously quantized to 16-bit multiplicands and 8-bit multipliers and refined for 20 additional training epochs, resulting in no accuracy loss. To establish a baseline, we iteratively repeated the same procedure to further homogeneously reduce the multipliers bitwidth N down to 2 bits, retraining the models for five fine-tuning epochs at each step. Five epochs are also run after each optimization step in the following phases (Fig. 4-b and Fig. 4-d).

B. Experimental Results

The tradeoff between energy improvements and the corresponding accuracy degradations of different quantized solutions is summarized in Fig. 5 for the cases of 32 and 128 subarrays. The black lines report the achieved accuracy for baseline CNNs having a homogeneous bitwidth of the multiplier values. In this case, a bitwidth of $N = 5$ bits results in energy savings of 35%, with an accuracy degradation of 1.3% on average, which rapidly increases for further bitwidths reductions. On the other hand, blue and green lines report the accuracy/energy at each step of our proposed methodology (as illustrated in Fig. 4) for maximum accuracy drops of 1% and 5%, respectively.

The obtained results highlight that the three optimizations phases tangibly reduce energy requirements (up to 81.6% for

[3]https://www.epfl.ch/labs/esl/research/open-source-tools/cnn2blade/



(a) LeNet-5 (b) AlexNet

(c) VGG16 (d) MobileNet

Fig. 5. Accuracy at different energy reductions in homogeneously quantized CNNs (black ●) and our optimized CNNs for a 1 % (blue ▲) and a 5 % (green ★) accuracy drops, in the four evaluated bencharks running in our architecture using 32 subarrays (LeNet-5) and 128 subarrays (other CNNs).

TABLE II
INFERENCE ENERGY AND INFERENCES PER SECOND (IPS) WHEN RUNNING THE PROPOSED BENCHMARKS IN OUR ARCHITECTURE HAVING 4, 32 OR 128 SUBARRAYS.

	Energy (μJ)			**IPS**		
Subarrays	**4**	**32**	**128**	**4**	**32**	**128**
LeNet-5	0.69	0.64	1.63	3304	8544	8560
AlexNet	1070	364	323	2	15	43
VGG16	470	151	126	5	36	110
MobileNet	255	84	70	9	65	195

a 5% accuracy drop threshold). Crucially, steps (a) and (b) are inter-dependent, because per-layer quantization increases the ratio of prunable filters. Indeed, our optimized models incur in very low degradations, achieving at the same time efficiencies higher than those obtained with a homogeneous 2-bit quantization. Such favorable energy/accuracy trade-off is enabled by the flexibility supported by our IMC architecture, which support the computation of highly heterogeneous CNN models. Latency trends closely follow energy ones, since inference time is reduced as a result of lower operands bitwidths, which lowers both the amount of data transfers and the number of MAC operations.

Table II summarizes the inference energy and the corresponding Inferences Per Second (IPS) of our optimized CNNs running in our proposed IMC architecture with a 5% accuracy drop constraint, for different amounts of subarrays. We observe that the higher the number of subarrays, the higher the energy efficiency and IPS. In fact, having more subarrays allows to execute more MAC operations in parallel, reducing computational time and thus leakage. LeNet-5 is an exception, because its small size and minimal computational requirement prevent larger degrees of parallelization.

A more detailed description of the structure of the CNN models resulting from our flow is depicted in Fig. 6, which shows the average layer bitwidth of our optimized benchmarks.

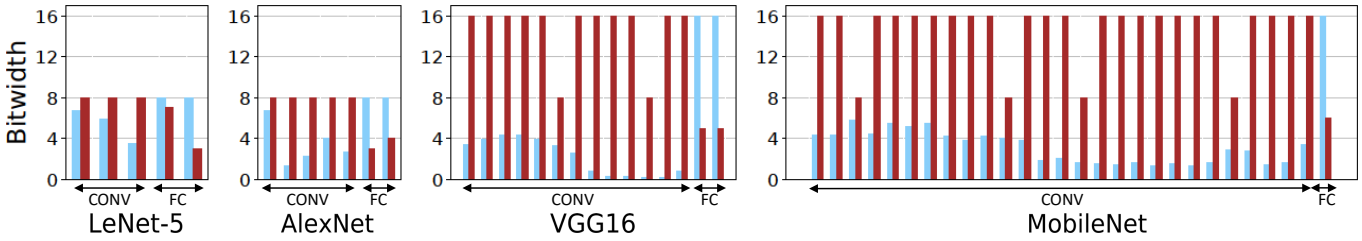
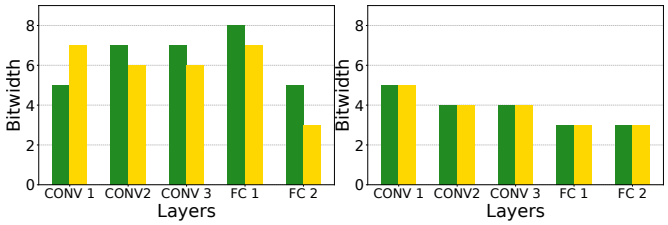


Fig. 6. Average weights (left blue bars) and activations (right red bars) bitwidth in convolutional and fully connected layers of our tested benchmarks.



(a) Drop=1% (b) Drop=5%

Fig. 7. Optimal multipliers bitwidth (left green bars) compared to the layer-based optimization solution (right yellow bars) as described in Section IV.

Blue bars represent the bitwidth of the weights, while red bars correspond to the bitwidth of the activations. Averages are impacted by the pruning of filters containing all zeroes after quantization (27% of filters in LeNet-5, 70% in VGG16), and the optimization of the ones that don't require the entire data range. The likelihood of finding prunable filters is higher in larger models like MobileNet and VGG16. More importantly, such ratio is dramatically increased by aggressive quantization, as small values are casted to 0. Indeed, on average only 3% of filters only contain zeros in floating-point models. We also observe that intermediate layers are more amenable to high bitwidth reductions, up to an average 0.13 bit per weight in the 11th CONV layer of VGG16. Conversely, initial layers exhibit lower robustness towards quantization, suggesting that they may have a crucial role in the input feature extraction.

In Fig. 7, we further evaluate the performance of the optimization flow described in Section IV. Therein, we compare the structure of an optimized LeNet-5 model derived using the workload-aware heuristic with one obtained using an exhaustive exploration of all possible multiplier bitwidths. Such comparison is only feasible on the simple case of LeNet-5, because a) the number of possible quantization settings grows exponentially with the number of layers and b) the evaluation of all combinations is, in the case of exhaustive explorations, done after re-training. Conversely, our heuristic orders the layers only based on their required MACs, computed a priori. Fig. 7(a) shows that, in the case of a 1% accuracy degradation threshold, the configuration resulting from the heuristic is very close to the optimal one, with a maximum deviation of two bits in the first CONV layer. This difference accounts for only a 7% difference in energy between the optimal solution and the one retrieved by our heuristic. For a 5% accuracy drop, heuristic and exhaustive results are instead identical.

VI. Conclusion

In this paper, we have introduced a novel IMC architecture designed to efficiently run convolutional neural network inferences, by enabling the parallel execution of MAC instructions in different subarrays. MAC operands of convolutional and fully connected layers are mapped to subarrays maximising both data reuse and parallelism of IMC operations. We have also presented a CNN optimization methodology to heterogeneously reduce the operands bitwidths. The obtained models have been evaluated in the proposed IMC architecture and compared with uniformly quantized CNNs. Our results showed energy and latency improvements up to 76.2%, with just 1% accuracy degradation.

References

[1] Z. Zhou *et al.*, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, 2019.

[2] T. Liang *et al.*, "Pruning and quantization for deep neural network acceleration: A survey," *arXiv:2101.09671*, 2021.

[3] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, 2020.

[4] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, 2017.

[5] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[6] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[7] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[8] K. He *et al.*, "Deep residual learning for image recognition," *CVPR*, 2016.

[9] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[10] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, 2015.

[11] A. Zhou *et al.*, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv:1702.03044*, 2017.

[12] Y. He *et al.*, "Channel pruning for accelerating very deep neural networks," in *IEEE ICCV*, 2017.

[13] A. Reuther *et al.*, "Survey and benchmarking of machine learning accelerators," in *IEEE HPEC*, 2019.

[14] D. Ielmini *et al.*, "Device and circuit architectures for in-memory computing," *Advanced Intelligent Systems*, 2020.

[15] K. C. Akyel *et al.*, "DRC2: Dynamically reconfigurable computing circuit based on memory architecture," *IEEE ICRC*, 2016.

[16] A. W. Simon *et al.*, "Blade: Bitline accelerator for devices of the edge," *ACM GLSVLSI*, 2019.

[17] W. A. Simon *et al.*, "Blade: An in-cache computing architecture for edge devices," *IEEE Transactions on Computers*, 2020.

[18] M. Rios *et al.*, "An associativity-agnostic in-cache computing architecture optimized for multiplication," *IEEE VLSI-SoC*, 2019.

[19] M. Rios *et al.*, "Running efficiently cnns on the edge thanks to hybrid sram-rram in-memory computing," in *IEEE/ACM DATE*, 2021.

[20] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv:1912.01703*, 2019.

[21] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE ICCV*, 2018.