This is the final peer-reviewed accepted manuscript of:

Nazareno Bruschi, Angelo Garofalo, Francesco Conti, Giuseppe Tagliavini, and Davide Rossi. 2020. Enabling mixed-precision quantized neural networks in extreme-edge devices. In *17th ACM International Conference on Computing Frontiers 2020* **(CF'20), May 11–13, 2020, Catania, Italy. ACM, NewYork, NY, USA, pp. 217-220. ISBN:978-1-4503-7956-4**

The final published version is available online at:

http://doi.org/10.1145/3387902.3394038

Enabling Mixed-Precision Quantized Neural Networks in Extreme-Edge Devices

Nazareno Bruschi
University of Bologna, Italy

Angelo Garofalo
University of Bologna, Italy

Francesco Conti*
University of Bologna, Italy

Giuseppe Tagliavini
University of Bologna, Italy

Davide Rossi
University of Bologna, Italy

ABSTRACT

The deployment of Quantized Neural Networks (QNN) on advanced microcontrollers requires optimized software to exploit digital signal processing (DSP) extensions of modern instruction set architectures (ISA). As such, recent research proposed optimized libraries for QNNs (from 8-bit to 2-bit) such as CMSIS-NN and PULP-NN. This work presents an extension to the PULP-NN library targeting the acceleration of mixed-precision Deep Neural Networks, an emerging paradigm able to significantly shrink the memory footprint of deep neural networks with negligible accuracy loss. The library, composed of 27 kernels, one for each permutation of input feature maps, weights, and output feature maps precision (considering 8-bit, 4-bit and 2-bit), enables efficient inference of QNN on parallel ultra-low-power (PULP) clusters of RISC-V based processors, featuring the RV32IMCXpulpV2 ISA. The proposed solution, benchmarked on an 8-cores GAP-8 PULP cluster, reaches peak performance of 16 MACs/cycle on 8 cores, performing 21× to 25× faster than an STM32H7 (powered by an ARM Cortex M7 processor) with 15× to 21× better energy efficiency.

CCS CONCEPTS

• **Computer systems organization** → **Parallel architectures**;
• **Computing methodologies** → **Machine learning**.

KEYWORDS

Embedded Systems, Quantized Neural Network, Low power Architectures

1 INTRODUCTION

An increasing amount of Internet-of-Things (IoT) applications acquire data from low-power sensors and transmit it wirelessly after some forms of compression. Machine Learning (ML) algorithms, and in particular Convolutional Neural Networks (CNNs), provide an effective solution for these applications thanks to their capability to squeeze raw sensor data in a much more dense format (e.g., classes or extracted high-level features). As such, a recent trend lies into deploying deep learning functionality on embedded

microcontrollers (MCU), which are the de-facto standard compute platform for IoT end-nodes thanks to their flexibility, low-power, and low-cost.

On the other hand, the computing power and memory footprint of MCUs is often not suitable for implementing state-of-the-art models. A recent trend in embedded CNNs to reduce both computational cost and memory footprint of CNNs is quantization [5][3]. This approach, representing the network weights and features with 8-bit or even smaller data types, such as 4-bit or 2-bit, has demonstrated the capability to reduce the memory footprint of state-of-the-art networks [9], with negligible accuracy loss. Optimized software libraries for Quantized Neural Networks (QNNs) have been proposed by the industry by means of CMSIS-NN library [7], targeting 16-bit and 8-bit QNNs on Cortex-M microcontrollers; as well as by the research community, such as PULP-NN, an open-source library targeting RISC-V processors, and supporting heavily quantized CNNs working on 8-bit, 4-bit, 2-bit, or 1-bit data [2]. To further reduce the memory footprint, recent works show how mixed-precision quantization can achieve better performance compared to symmetrical quantization of input feature maps (*ifmaps*), weights, and output feature maps (*ofmaps*). For example, applying this approach on a MobileNetV1 CNN achieves 7× memory footprint reduction, while incurring an accuracy loss of only 4% [1] with respect to the 32-bit integer representation.

With this aim, we propose an extension to the PULP-NN open-source library, which includes 27 convolution kernels, one for every permutation of *ifmaps*, weights, and *ofmaps* quantization level, for 8-bit, 4-bit and 2-bit, overtaking the limitations of the current open-source library supporting symmetrical quantization only. Our solution can reach 16 MACs per cycle on octa-core execution on the GreenWaves Technologies GAP-8 [4] processor, up to 25× and 46× faster with up to 45× and 21× less energy than the execution on a STM32H7 and STM32L4, which are commercial MCUs based on an ARM Cortex M7 and M4 core respectively.

2 BACKGROUND

2.1 QNNs and Mixed-Precision QNNs

A QNN is defined by mapping all real-valued tensors involved in a DNN layer (weights $\mathbf{w}$, *ifmaps* $\mathbf{x}$, *ofmaps* $\mathbf{y}$) to integers. In this work

---

we focus on layer-wise linear quantization, where each real-valued tensor $\mathbf{t}$ in the range $[\alpha_{\mathbf{t}}, \beta_{\mathbf{t}})$ is built such that:

$$\mathbf{t} = \alpha_{\mathbf{t}} + \varepsilon_{\mathbf{t}} \cdot INT(\mathbf{t}) \tag{1}$$

where $INT(\mathbf{t})$ is an N-bit integer-valued tensor with the same dimensionality of $\mathbf{t}$, and $\varepsilon_{\mathbf{t}} = (\beta_{\mathbf{t}} - \alpha_{\mathbf{t}})/2^N$. We further constrain $\alpha_{\mathbf{x}} = \alpha_{\mathbf{y}} = 0$ for *ifmaps* and *ofmaps*. A QNN of this kind can be trained efficiently by means of linear quantization-aware training [6], which produces a QNN using real-valued tensors of the form of Eq. 1. The application of *linear* layers (e.g., convolutional and dense), normalization (e.g., batch-norm) and activation (e.g., ReLU) in a QNN can then be mapped to a linear operation combined with a pointwise normalization/activation, working directly on the integer-valued tensors:

$$INT(\mathbf{y}) = quant\Big(linear\big(INT(\mathbf{w}), INT(\mathbf{x})\big)\Big) \tag{2}$$

Notice that the accumulator $\varphi \doteq linear\big(INT(\mathbf{w}), INT(\mathbf{x})\big)$ is still integer-valued, but requires in general more bits than its inputs (i.e., ε_{φ} will be smaller than $\varepsilon_{\mathbf{x}}$ and $\varepsilon_{\mathbf{w}}$). *quant* normalizes φ with an affine transformation of parameters κ and λ, then collapses its values, "converting" it to a representation with less bits (i.e., with bigger $\varepsilon_{\mathbf{y}}$) [1]:

$$INT(\mathbf{y}) = quant(\varphi) = \text{clip}_{[0,\beta_{\mathbf{y}})}\left(\left\lfloor \big(\kappa \cdot INT(\varphi) + \lambda\big) \cdot \varepsilon_{\varphi}/\varepsilon_{\mathbf{y}} \right\rfloor\right) \tag{3}$$

In mixed-precision QNNs, the number of bits used for $\mathbf{w}$, $\mathbf{x}$ and $\mathbf{y}$ is not constrained to be the same. This class of QNNs have been shown [1] to better fit embedded constraints while incurring in a less severe accuracy hit than non-mixed-precision QNNs; massive memory gains can be exploited on tensors and layers that are less sensitive to strong quantization while still keeping more sensitive ones at a higher precision. Here, we focus on 2-, 4- and 8-bit quantization for $\mathbf{w}$ (signed), $\mathbf{x}$ and $\mathbf{y}$ (unsigned), while we always consider 32 bits for the accumulator φ (signed).

2.2 PULP-NN

The software solution we propose is built upon an open-source Parallel Ultra-Low-Power (PULP) cluster of eight RISC-V based processors [2]. The cores feature a 4-stage in-order pipeline and the RV32IMC ISA, extended with efficient digital signal processing custom instructions, namely XpulpV2. A detailed description of these extensions can be found in [8]. The key elements of a PULP cluster are a low-latency multi-banked Tightly Coupled Data Memory (TCDM), enabling shared-memory parallel programming models such as OpenMP or OpenCL, and an Event Unit which manages synchronization and thread dispatching, enabling low-overhead and fine-grained parallelism, guaranteeing high efficiency for parallel workloads. In this work, we leverage a commercial embodiment of the PULP architecture fabricated in CMOS 55nm called GAP-8 [4].

The PULP-NN library, extended in this work to support mixed-precision QNNs, relies on the Height-Width-Channel (HWC) data layout and on an execution flow optimized to target resource-constrained MCUs. A layer is run as a combination of three phases:
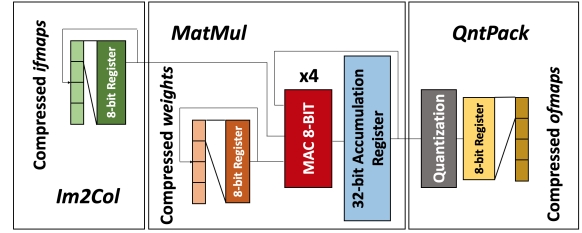
---

[1]The κ and λ parameters can be integrated directly in the ladder function, resulting in a *quant* function that produces $INT(\mathbf{y})$ by comparing φ with a set of 2^N *thresholds* [9].
[2]https://github.com/pulp-platform/pulp

Figure 1: Concept scheme of Mixed-Precision Approach.

```
void pulp_nn_int4_to_int8(int8_t *Src, int8_t *Out)
{
    int8_t bext1 = bext(Src, 4, 0 );
    int8_t bext2 = bext(Src, 4, 4 );
    int8_t bext3 = bext(Src, 4, 8 );
    int8_t bext4 = bext(Src, 4, 12);
    *((v4s*)Out) = pack(bext1, bext2, bext3, bext4);
    Out++;
    bext1 = bext(Src, 4, 16);
    bext2 = bext(Src, 4, 20);
    bext3 = bext(Src, 4, 24);
    bext4 = bext(Src, 4, 28);
    *((v4s*)Out) = pack(bext1, bext2, bext3, bext4);
}
```

Figure 2: Example of code of efficient bit extraction using XpulpV2 extension.

the *im2col* step re-arranges the 3D input features of the current layer into a 1D vector, the *linear* part of the layer is implemented through a Matrix Multiplication (*MatMul*) kernel, while a final stage of quantization, named *QntPack*, implements the *quant* function of Eq. 2, compresses the *MatMul* result into the desired precision and then stores back the *ofmap*.

The *MatMul* loads the quantized weights from four different filter banks and the input features from two different *im2col* buffers into the register file from the TCDM. Exploiting the data locality of the loaded features and weights within the register file enables computing two spatially adjacent output features of four consecutive output channels in each run of *MatMul* inner loop, optimizing the execution of the kernel. Further details can be found in [2]. Since the results of *MatMul* need to be accumulated into higher precision variables, they need to be compressed back to the desired output precision. While for 8-bit output features scaling and clamp operations can be used [7], an effective solution for sub-byte outputs consists of a thresholding-based procedure [1, 5]. This operation compares an input with a set of thresholds (see Section 2.1). Every *Conv* kernel presented in this work is parallelized on the H-spatial dimension of *ofmaps* [2], resulting into an almost ideal speed-up on an 8-cores cluster (7.5×).

3 MIXED-PRECISION KERNELS

In this section, we describe the proposed mixed-precision software kernels, highlighting the optimizations made to boost the kernels on the PULP cluster. In the context of a mixed-precision convolution kernel (*Conv*), the precision of the *ifmaps* determines the specific *im2col* function to be used, the precision of the *weights* determines the specific *MatMul* kernel, while the *ofmap* determines the specific *QntPack* kernel.

As the underlying hardware offers support only for 8-bit SIMD instructions, when sub-byte input features are considered, additional unpacking functions must be added to the *im2col* procedure

```
void pulp_nn_insert_int4(int8_t Src1, int8_t Src2, int8_t *Out)
{
    int8_t mask = 0xf0;
    int8_t n_mask = not(mask);
    int8_t off = 0x04;
    *Out = bins(Src1, n_mask, Src2, mask, off);
}
```

Figure 3: Example of code of efficient bit compression using XpulpV2 extension.

to extract and sign-extend the sub-byte operands into INT-8, natively supported by the sum-of-dot product units. Fig. 1 highlights a general scheme of mixed-precision *Conv* structure. Depending on the *ifmap* and weights precision, different specific casting functions are built, to reduce the number of load instructions needed to fetch the compressed features, hence minimizing the memory traffic and improving the performance. The casting operation takes place also in the innermost loop of the *MatMul* kernel to 'unpack' the weight elements. To reduce unpacking operation overhead, we exploit the bit manipulation instructions provided by the XpulpV2 ISA.

As depicted in Fig. 2 for the 4-bit case without loss of generality, we exploit the XpulpV2 *bit extraction* operation, inferred in the C code as a built-in function (*bext*), which extracts a specified number of bits from a 32-bit register in one clock cycle, also extending the sign bit. With one 32-bit load instruction that fetches eight lower-precision operands, we can obtain eight 8-bit operands (with sign extension), packed in two 32-bit vector registers, ready to be handled in a SIMD fashion. For the 2-bit case, the cost of the load operation is further amortized, since with one 32-bit load we obtain 16 8-bit operands, achieving 0.0625 loads per operand, half than in the 4-bit case.

Since unpacking is a critical operation for *MatMul* kernels, several solutions have been explored, leading to the following optimal kernels structure:

- 8-bit weights: 6 32-bit loads and 8 SIMD MACs for a total of 14 cycles per iteration.
- 4-bit weights: 8 32-bit loads, 32 extractions, 16 pack and 16 SIMD MAC, for a total of 72 cycles per iteration.
- 2-bit weights: like the previous one but, due to the different level of unrolling in the extraction function, there is a total of 140 cycles per iteration.

Each inner loop is iterated over the *im2col* size. The number of iterations depends on the overall number of MACs for each iteration.

Since the *MatMul* works on 32-bit accumulators as specified in Section 2.1, the *QntPack* function quantizes and pack it to the desired *ofmap* precision. While simple shifts and clamps are used to restore the output range in 8-bit and store it in an 8-bit variable, to compress back sub-byte results in an 8-bit one, additional *packing* functions have to be added after the thresholding-based quantization. This is implemented efficiently, exploiting the *bit insert* function that acts as a natural counterpart of the *bit extract*, which compresses the data and packs them into 8-bit variables (see Fig. 3).

4 EXPERIMENTAL RESULTS

We ran our kernels on GAP-8 as a commercial product, an edge low-power and octa-core PULP device optimized to perform DNN algorithms [4] and then we compared the execution performance and the energy consumption of an STM32H7 and STM32L4 MCU,
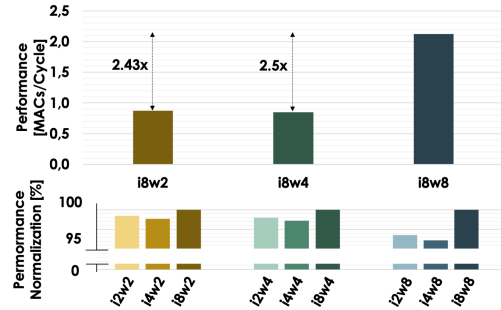

Figure 4: MACs/cycle in a single-core *linear* execution in different *weights* precision and its fluctuations by varying *ifmaps* precision.

Table 1: Average overhead cost in cycles per output pixel and its variance by varying the *ofmaps* precision.

ofmaps precision	cycles/output pixel	variance
8-bit	2.01	+/- 0.57
4-bit	16.64	+/- 4.47
2-bit	8.02	+/- 1.15

which run the same layer and the same kernels. Although the proposed library is fully flexible, we present the results of a reference layer featuring 32x16x16 *ifmaps* size, 64x16x16 *ofmaps* size and 3x3 filters. This layer has a 288 im2col buffer size, referred to in the following as *Reference Layer*, and it is among the ones featuring the best performance on the targeted architectures. We considered the MACs per cycle and cycles per output pixel as the key metrics to define the performance of the library.

4.1 Single- and Multi-Core Execution Results

As seen in Section 3, the inner loop has a different number of iterations depending on the weights precision level. Therefore, we should expect a decrease in terms of performance of 2.57× and 2.5× with respect to the 8-bit *MatMul*, for 4-bit and 2-bit *MatMul*, respectively, due to the unpacking overhead.

To isolate the contributions of the *linear* kernel execution, in Fig. 4 we consider *im2col* and *MatMul* in isolation, removing the per-output-pixel overhead of the *QntPack* function. The plot shows how much weight unpacking impacts the MACs per cycle performance metric compared to the 8-bit case. In line with expectations, performance drops by 2.43× and 2.5× in 2-bit and 4-bit scenarios, respectively. The solution proposed for 2-bit weights is more efficient than 4-bit because it reduces the number of load instructions per MAC, despite introducing more unpacking and packing instructions in the inner loop. Under the bars, Fig. 4 shows how much the *ifmaps* precision impacts performance at a fixed *weights* precision. We observe how the pattern is similar to the one for weights (the solution for 8-bit is the best one, while 2-bit is better than 4-bit); however, it is important to note that the variation is much smaller than that observed when changing weight precision.

In Tab. 1, we investigate the overhead introduced by the *QntPack* function to the overall layer computation. Due to deep compiler optimization and instruction cache effects, these results have a high
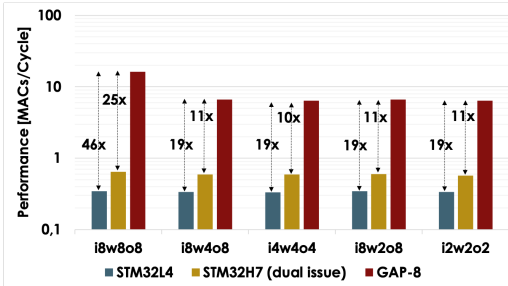
Figure 5: Speed-up of PULP-NN Mixed-Precision on GAP-8 (8 cores) over STM32H7 and STM32L4 on the *Reference Layer*.

Figure 6: GAP-8, STM32H7 and STM32L4 energy consumption when they run the *Reference Layer*.

variance, which we explicitly represent in the table. In particular, depending on the size and structure of the innermost loop, code integrating the *linear* and *QntPack* functions is optimized differently in each case, resulting in a different number of inner instructions but also in different binary code sizes, triggering more instruction cache misses in some cases with respect to others. Despite this significant variability, we can observe clear trends in the overhead *QntPack* introduces. When using thresholds as activation functions, the average results in Tab. 1 is as expected, because they are realized with if-else nested statements that perform the binary search in the range in which the output value is found. 4-bit quantization requires twice the number of threshold comparisons than 2-bit quantization, therefore we expect double of cycles per output pixel. Most non-idealities come from this operation, which is expensive in terms of branches and pipeline stalls. Furthermore, when we have sub-byte quantization of output, bit compression instructions also enter into the game to pack two or four pixels into an *ofmap* byte, and 8-bit *ofmaps* perform better than 4-bit and 2-bit operations.

4.2 Comparison with state-of-the-art

To compare our work with the state-of-the-art, we used an STM32H7 and an STM32L4 running the *Reference Layer*, which are a dual and single issue processors respectively. In Fig. 5 we can see the cycle/-cycle speed-up that an octa-core GAP-8 can achieve respect to these devices. In terms of MACs per cycle, we achieve up to 25× and 46× in a *Conv* kernel with 8-bit *ifmaps*/*ofmaps* and weights. The contributions of this improvement are certainly to attribute not only to octa-cores execution but also to the XpulpV2 ISA that, compared to ARM Cortex-M7 and -4, features extensions to perform SIMD 8-bit MACs in one cycle with respect to 16-bit SIMD MACs of ARM Cortex-M-based ones. On the other hand, also when unpacking is necessary, we still perform up to 11× and 19× respectively.

Finally, we compared the energy consumption of the *Reference Layer* on the benchmarked platforms. We used the two different operating modes for GAP-8: low-power and high-performance (see Fig. 6). Despite the less scaled technology node used for the implementation of GAP-8 with respect to STM32H7 (i.e., 55 vs. 40 nm CMOS) and the higher frequency respect to STM32L4 (i.e., 90 MHz vs 80 MHz), GAP-8 performs with 45× and 21× less energy consumption in the low-power mode and 31× and 15× in the high-performance one, with 8-bit precision operands. When the unpacking is necessary, the energy consumption stills up to 20× and 9× in low-power mode and 14× and 6× in high-performance
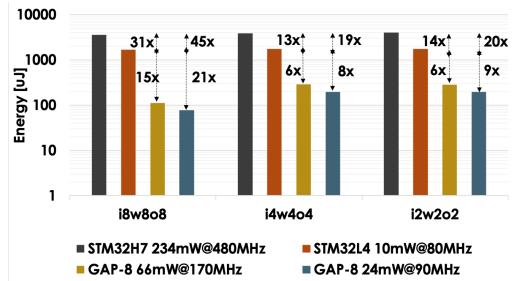
one respect to STM32H7 and STM32L4 execution respectively, as depicted in Fig. 6. This demonstrates the potential of the parallel execution on an optimized cluster with PULP-specific instruction set extensions, coupled with an optimized software abstraction layer able to efficiently exploit the underlying hardware.

5 CONCLUSION

We have presented an open-source software library for mixed-precision inference on parallel ultra-low-power clusters at the edge of the IoT. The proposed library supports 8-bit, 4-bit and 2-bit QNN kernels and for all variants of input feature maps, weights, and output feature maps. Exploiting the DSP capabilities of the XpulpV2 extensions, coupled with the performance gain of parallel execution, our solution can reach up to 16 MACs per cycle on quantized convolutional kernels on the 8-core PULP cluster of the GAP-8 SoC. These results outperform by 25× the execution of the same kernels on an STM32H7 microcontroller, with 21× better energy efficiency compared to STM32L4 microcontroller.

ACKNOWLEDGMENTS

REFERENCES

[1] Alessandro Capotondi et. al. 2020. CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices. *IEEE Transactions on Circuits and Systems II: Express Briefs* (2020), 1–1.

[2] Angelo Garofalo et. al. 2019. PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378, 2164 (Dec 2019), 20190155. https://doi.org/10.1098/rsta.2019.0155

[3] Bert Moons et. al. 2017. Minimum Energy Quantized Neural Networks. arXiv:cs.NE/1711.00215

[4] Eric Flamand et. al. 2018. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 1–4.

[5] Itay Hubara et. al. 2016. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. arXiv:cs.NE/1609.07061

[6] Jungwook Choi et. al. 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks. arXiv:cs.CV/1805.06085

[7] Liangzhen Lai et. al. 2018. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. arXiv:cs.NE/1801.06601

[8] Michael Gautschi et. al. 2016. A near-threshold RISC-V core with DSP extensions for scalable IoT Endpoint Devices. arXiv:cs.AR/1608.08376

[9] Manuele Rusci et. al. 2018. 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* (Turin, Italy) *(CODES '18)*. IEEE Press, Article Article 12, 2 pages.