PULP-NN: A computing library for quantized neural network inference at the edge on RISC-V based parallel ultra low power clusters

# PULP-NN: Accelerating Quantized Neural Networks on Parallel Ultra-Low-Power RISC-V Processors

Angelo Garofalo†, Manuele Rusci†, Francesco Conti†*, Davide Rossi† and Luca Benini†*
*DEI, University of Bologna, Italy*† *IIS lab, ETH Zurich, Switzerland**
{angelo.garofalo, manuele.rusci, davide.rossi}@unibo.it {fconti, lbenini}@iis.ee.ethz.ch

***Abstract*—We present PULP-NN, an optimized computing library for a parallel ultra-low-power tightly coupled cluster of RISC-V processors. The key innovation in PULP-NN is a set of kernels for Quantized Neural Network (QNN) inference, targeting byte and sub-byte data types, down to INT-1, tuned for the recent trend toward aggressive quantization in deep neural network inference. The proposed library exploits both the digital signal processing (DSP) extensions available in the PULP RISC-V processors and the cluster's parallelism, achieving up to $15.5$ MACs/cycle on INT-8 and improving performance by up to 63$\times$ with respect to a sequential implementation on a single RISC-V core implementing the baseline RV32IMC ISA. Using PULP-NN, a CIFAR-10 network on an octa-core cluster runs in $30\times$ and $19.6\times$ less clock cycles than the current state-of-the-art ARM CMSIS-NN library, running on STM32L4 and STM32H7 MCUs, respectively. The proposed library, when running on GAP-8 processor, outperforms by $36.8\times$ and by $7.45\times$ the execution on energy efficient MCUs such as STM32L4 and high-end MCUs such as STM32H7 respectively, when operating at the maximum frequency. The energy efficiency on GAP-8 is $14.1\times$ higher than STM32L4 and $39.5\times$ higher than STM32H7, at the maximum efficiency operating point.**

## I. Introduction

The Internet-of-Things has favored a rapid growth of the number of wireless-connected nodes for a large variety of applications, including agriculture [1], health monitoring [2], surveillance [3], structural monitoring [4]. Such a massive unconstrained increment poses severe challenges to the network infrastructure, due to the exponential increase of data flowing through the network. Capacity, security and reliability issues are exacerbated as the number of IoT nodes increases exponentially together with the ability to produce high-bandwidth data.

To address IoT scalability issues, data must be filtered at the edge of the network, on the sensor system itself [5], using compression and analytics algorithms. To this aim, Machine Learning (ML), including also state-of-the-art Deep Learning (DL), provides attractive solutions for edge processing. ML algorithms "squeeze" raw sensor data in a much more semantically dense format (i.e. classes or extracted high-level features/symbols), eventually packed into few bytes of information for wireless transmission.

To empower IoT nodes with smart capabilities [6], the design process of edge devices must trade-off the high computation and memory requirements of leading DL methods with the usual scarcity of resources of deeply embedded systems, powered by batteries or energy harvesters. Typically, deep network inference tasks run on GPUs or FPGAs devices, which however have a power envelope significantly higher than what can be sustained on extreme-edge devices, integrated with the sensors. On the other side of the spectrum, resource-constrained MCUs are flexible, due to their software programmability, low-cost, low-power and suitable for extreme-edge usage, but they present severe limitations in memory footprint and computation resources that may prevent meeting application-specific latency and accuracy requirements.

To reduce the computational cost and memory footprint of Neural Newtorks, so that they can fit the limited computing capability and storage capacity of MCU-class devices, recent progress in DL training methodologies has introduced novel quantization methods, aiming at compressing either network weights parameters or activations into 8-bit or smaller data types, while incurring into a reduced or even negligible accuracy loss [7]–[13]. Since Quantized Neural Networks (QNNs) feature much lower memory requirements than 32-bit floating point full precision models and low-bitwidth fixed-point execution units can operate efficiently at the core of the convolution routine, industry and academia are devoting a major effort to develop hardware and software platforms for efficient execution of QNNs on MCU-class devices.

In this work, we propose the first multicore computing library for QNN inference on fully programmable edge devices, which supports low bit-width (8-bit, 4-bit, 2-bit and 1-bit) operations. While efficient libraries for commercial MCUs have been proposed for edge QNN inference [14], [15], not many software solutions have been yet presented that efficiently exploit a parallel MCU architecture. We fill this void by building the back-end library upon the recent architectural template of parallel ultra-low-power RISC-V based platforms such as GAP8 [16], which improve energy efficiency and performance in IoT edge devices coupling parallelism with low voltage operation [17]. The main contributions of this paper are the following:

- *PULP-NN* [1], an open-source optimized library based on the CMSIS-NN [14], [15] dataflow including a full set of kernels and utilities to support the inference of Quantized

[1]https://github.com/pulp-platform/pulp-nn

Neural Networks (8,4,2 and 1-bit) on a DSP-optimized RISC-V based processor. By fully exploiting the DSP extensions available within the ISA, we can achieve a speedup of $9\times$ with respect to a plain*RV32IMC* ISA;
- We optimized the library for a Parallel Ultra-Low-Power (PULP) cluster of RISC-V processors, leading to near-linear speedup with respect to single core execution, increasing the throughput of each kernel by up to $7.5\times$ on eight cores;
- We optimized the convolution kernel, the most computing intensive task of CNN workloads, by improving data reuse, with a further 20% performance gain with respect to the original kernel of CMSIS-NN [14], with a $\sim$1.9$\times$ improvement with respect to the GAP-8 NN native library and an overall efficiency of $49\%$ in terms of MAC utilization, which implies just 1.01 LD/ST per MAC, and brings us to just a factor of 2 from the theoretical peak MAC utilization achievable using only register operands;
- We compare our solution with State-of-the-Art architectures and software, by running a CIFAR-10 quantized model on the GAP8 8-core cluster, outperforming by $19.5\times$ a high-end MCU (based on ARM CORTEX-M7) running the same network using the CMSIS-NN library. The inference with the proposed library also achieves $14.1\times$ better energy efficiency with respect to a highly energy efficient MCU (based on ARM CORTEX-M4).

These order-of-magnitude improvements with respect to State-of-the-Art MCUs demonstrate for the first time that extreme-edge inference of QNN models is indeed possible on today's parallel ultra-low power MCUs.

## II. Related Work

The success of Deep Learning (DL) has paved the way to many different DL deployments on embedded computing platforms of all kinds. In this section, we recap the state-of-the-art and give insights on its applicability to CNN inference at the extreme-edge, on IoT end-nodes.

***FPGA Based Approaches:*** Recent heterogeneous FPGAs such as Xilinx Zynq have enabled many solutions for CNN acceleration, embedding general purpose processors that manage the program flow, handle I/O and memory accesses, making them easier to program. As DSP-capable FPGAs have a power envelope in the order of Watts, numerical precision of the CNN operands plays a crucial role to achieve high performance and thus energy efficiency. While several architectures available in literature feature a precision of 16-bit (fixed-point) [18]–[21], more and more designs are moving towards lower precision. For example, Qiu et al. [22] proposed a CNN accelerator supporting 8 and 4-bit data, implemented on a Xilinx Zynq platform. On this trail, even extreme quantization approaches have been presented, exploiting ternary or binary networks [23], [24]. While most DSP-capable FPGAs currently do not offer a low enough power envelope to be used in IoT end-nodes, Lattice recently announced SenseAI class of FPGAs [25] providing a comprehensive hardware and software solutions for always-on artificial intelligence (AI) within a power budget between 1 mW and 1 W. However these ultra-low power FPGAs are currently too expensive for many applications where MCUs are traditionally chosen because of their low cost. Furthermore, they report [26] a measured performance of 8 fps with 64$\times$64 RGB input for a VGG8 like 16-bit CNN at a power consumption of 7 mW, which maps to $0.88\,mJ/frame$, and performance of 5 fps for a VGG network consisting of 6 convolution layers and 4 fully connected at a power consumption of 3.3 mW with an energy per inference of $0.66\,mJ/frame$. Both the results are significantly higher ($4.63\times$ and $3.48\times$, respectively) than the energy per frame that we report at the maximum efficiency point for our solution in sec. V.

***Application Specific Architectures:*** On the other side of the programmability spectrum, ASIC accelerators are known to achieve best in class performance and energy efficiency. Notable examples are Orlando [27] achieving energy efficiencies in the order of a few Top/s/W, and Origami [28] achieving a throughput of 274 Gop/s, with an efficiency of 803 Gop/s/W. Dropping the arithmetic precision of CNN operands has demonstrated to be a useful technique to reduce the memory footprint and the energy cost for computation [29]–[32]. UNPU [33] is an example of an accelerator targeting fully-variable weight bit-precision, achieving a peak energy efficiency of 50.6 Top/s/W at a throughput of 184 Gop/s. YodaNN [34] targets binary-weight networks and reaches energy efficiency up to 61 Top/s/W. Other accelerators exploit extreme quantization for the deployment of binary neural networks on silicon using in- or near-memory computing techniques (e.g., Brein [35], Conv-RAM [36]) with energy efficiencies in the range 20-55 Top/s/W. Such high energy efficiency and throughput achievable using ASIC accelerators are counterbalanced by limited flexibility, being application specific, which makes them unattractive to satisfy fully the flexibility demand of IoT edge nodes.

***Software Programmable Architectures:*** Software-programmable general-purpose processors provide the highest degree of flexibility in QNN inference at the edge. While CNNs are traditionally executed on programmable high-performance GPUs [40], [41] also with reduced precision support [42], these platforms are typically not designed to operate in the tight power envelope of IoT end-nodes, and their cost is off-spec too. Some architectures exploit the computing power of multi-core processors, such as Raspberry Pi 3+ [43], powered by a Quad-core ARM CORTEX-A53. Although these platforms are relatively inexpensive and flexible, their power consumption is too high as well.

To fit the power budget of IoT edge devices, many low power microcontrollers include ARM CORTEX-M cores. Among these solutions, STMicroelectronics proposed low-end (STM32L4 family based on ARM CORTEX M-4 cores and high-end (STM32H7 family featuring ARM CORTEX M-7 cores) microcontrollers supporting DL processing at the edge [37], [38]. To improve the computing capabilities of such tiny and cheap computing platforms, ARM recently announced the development of the ARMv8.1-M [44] architecture, featuring

**Summary of CNN Embedded Inference Computing Platform**

| | Performance | Energy Efficiency | Power Budget | Flexibility |
|---|---|---|---|---|
| **ASICs [27], [28], [33], [34]** | 1 - 10 Tops/s | 10 - 100 Tops/s/W | 1 mW - 1 W | Low |
| **FPGAs [18]–[22]** | 10 - 200 Gops/s | 1 - 10 Gops/s/W | 1 W - 10 W | Medium |
| **MCUs [37], [38]** | 100 - 300 Mops/s | 1 - 3 Gops/s/W | 1 mW - 1 W | High |
| **PULP SoCs [6], [16], [39]** | 1 - 2 Gops/s | 30 - 50 Gops/s/W | 1 mW - 100 mW | High |

TABLE I
THE TABLE SHOWS THE TRADE-OFFS AMONG THE CNN COMPUTING PLATFORMS DESCRIBED IN THE RELATED WORK SECTION.

Helium, an ISA extension tailored for DSP-oriented workloads, such as an inference task. However, such an extension is not supported yet by any device.

Other solutions move toward heterogeneous architectures, coupling microcontrollers with dedicated CNN accelerators, to deal with the extremely regular CNN workload. ARM proposed Trilium [45], a heterogeneous compute platform which provides flexible support for ML workloads. Conti et al. [46] proposed a convolution engine to be integrated in a microcontroller to speed up the convolutional kernels while Kendryte [47] is a dual-core RISC-V SoC outfitted with a CNN accelerator for AI applications. Flamand et al. proposed GAP8 [16], a multi-GOPS fully programmable RISC-V IoT-edge computing engine, featuring a cluster of 8 cores with dedicated DSP extensions and a CNN-specialized accelerator. These accelerators can give the MCU a 5 to 10× energy efficiency boost, but they are proprietary, closed, platform specific and currently not fully supported by the software design flows. Hence, their acceptance and penetration among application developers is still quite low.

Table I summarizes the trade-offs among the CNN computing platforms described so far. Next section will describe the State-of-the-Art of software solutions for MCU platforms, the main focus of this work.

***Optimized Software Libraries:*** On the MCU side, the limited computational and memory capabilities make aggressive software and algorithmic optimizations necessary to deploy DNN inference models on them. An efficient solution to reduce DNN memory footprint is to use fixed-point arithmetic and quantization of both weights and activations into 8-bit or smaller data types, at the cost of a minor drop in accuracy [7], [8], [13]. Relying on fixed-point quantized networks, ARM proposed the CMSIS-NN library [14], which maximizes the performance of the DL kernels on CORTEX-M series cores, supporting 16-bit and 8-bit fixed-point data. On the same trail, targeting a parallel MCU architecture such as GAP-8, the Greenwaves Technologies company released open-source a set of QNN kernels (16- and 8-bit data precisions) as part of a proprietary tiling solution [16]. The tiling procedure, exploiting the DMA controller available on GAP-8, hides the latency of fetching/storing activations and weights along the memory hierarchy introducing only a small overhead (a few %), thus enabling the processing of large networks whose single layers may not fit the MCU on-board memory. In this work we focus on the computational aspects of reduced precision quantized CNN inference. In this context, despite the demonstrated effectiveness of sub-byte aggressive quantization [11], only Rusci et al. [15] explored the impact of using low-precision (4-, 2- or 1-bit) convolution kernels on a Cortex-M7 microcontroller.

Our work aims at bridging this gap, leveraging the results of [11] and focusing on the computational side to enable efficient QNN inference at the edge on fully programmable devices. To this purpose, we propose an open-source QNN library targeting 8-bit as well as sub-byte quantized data types, down to 1-bit data, targeting parallel ultra-low-power (PULP) architectures. By exploiting the ISA extensions available on PULP architectures and tightly coupled cluster, our contributions outperform the CMSIS-NN based solutions by one order of magnitude in terms of performance and energy efficiency.

## III. BACKGROUND

### A. Quantized Neural Networks

A Deep convolution Neural Network (CNN) is made of several layers stacked one on top of the other. Each layer can be considered as a computation kernel, and the most computive-intensive ones are the convolution and the fully connected layers.

To favor the deployment of CNN models into resource-constrained devices, a set of constraints can be applied to the numeric domain of either network parameters or activation values, turning the original model into a Quantized Neural Network (QNN). One of the most effective approaches [7] to quantize a real-valued weight parameter $w$ to a $Q$-bit signed fixed-point number $q(w)$ is by using the following quantization function:

$$q(w) = clip_{[-1,1)}(2^{-(Q-1)} \cdot round(w \cdot 2^{(Q-1)}) \ , \quad (1)$$

where $clip_{[a,b)}(x) = max(a, min(x, b))$. We define then the integer $W = q(w) \cdot 2^{(Q-1)}$ as the corresponding INT-Q representation of $w$. According to [7], the quantization rule (1) applies also to any activation value. In this work, we explore the case of INT-8, INT-4, INT-2 and INT-1 data types as they are the most natural ones to fit in a 32-bit register of the targeted MCUs. If both weights and activations are INT-Q

values, the convolution becomes a sum of products operation in the integer domain:

$$\phi(w,x) = 2^{-2(Q-1)} \sum_{i \in C} W_i X_i \doteq 2^{-2(Q-1)} \cdot \Phi(W,X) \ . \quad (2)$$

where $C$ is the number of input channels and $\phi$ is the convolution operation. $\Phi(W,X)$ is the accumulator value with high precision, i.e. INT-32 for INT-8 operands and INT-16 for sub-byte (INT-4, INT-2, INT-1) operands. To produce an output activation value, the accumualtion is compressed back into Q bits, working as input for the next layer. For INT-8 data we adopt the compression approach proposed by Lai et al. [14], which relies on scaling and clamp operations, while for the 2 and 4 bit cases a thresholding-based [2] compression is considered, described by the staircase function that generalizes (1):

$$Y = q\big(\phi(x)\big) = \sum_{p=-2^{Q-1}}^{2^{Q-1}-1} \big(p \cdot \chi_{[\tau_p, t_{p+1})} \cdot \Phi(W,X)\big) \ , \quad (4)$$

where $\chi_s(\cdot)$ it the characteristic function of the interval $s$. In this equation also the threshold values feature high precision (INT-16), since they are meant to be compared with INT-16 accumulations. The staircase function is optimally implemented through a balanced binary tree where an INT-16 comparison takes place at every node. To produce a Q-bit output, $2^Q - 1$ threshold values per channel must be stored for any convolution layer. The INT-1 format, where activation and weight values are expressed by binary values, is a special case because the convolution can be reduced to a logical XNOR and a bit-count operation:

$$\Phi_{bin}(X) = \text{popcount}(W \,\text{xnor}\, X) \quad (5)$$

where $popcount(\cdot)$ is the bitcount operator. Also in this scenario, a thresholding procedure is applied for compression.

On the model accuracy side, it has been demonstrated that, through specific re-training techniques, the accuracy drop-off of quantized fixed-point networks can be significantly reduced [7], [10], [13]. Choi et al. [48], for example, have proved that a 4-bit quantization leads to an accuracy level close to single-precision floating point representation. The accuracy drop is limited to 3% when running ResNet50 on Imagenet with 2-bit weights and 4-bit activations and to 6.5% when downscaling the weights and activations to 2 bits. Furthermore, the authors of [11] investigated the trade-off between energy efficiency and accuracy of QNNs, highlighting the practical effectiveness of the sub-byte fixed-point networks. At the cost of specific retraining procedures, the accuracy drop of is kept very close to the single-precision floating point counterpart while the energy efficiency gain, at the iso-accuracy, is orders of magnitude higher. Moreover, for the investigated networks, trained on CIFAR-10 and MNIST datasets, the energy consumption achieved with 1- to 4-bit fixed-point networks, at iso-accuracy, outperforms the 8-bit counterpart by up to 10×.

[2]The $\tau_p$ thresholds absorb bias, batch normalization and the $2^{-2(Q-1)}$ factor. Specifically, considering the batch-normalized $y = \gamma/\sigma(b+\phi-\mu)+\beta$ (where $b$ is the bias, $\gamma$, $\sigma$, $\beta$, $\phi$ are the batch normalization parameters), the thresholds are

$$\tau_p = [2^{Q-1}(p \cdot \sigma/\gamma - 2^{Q-1} \cdot (b-\mu) + \beta \cdot \sigma/\gamma] \ . \quad (3)$$

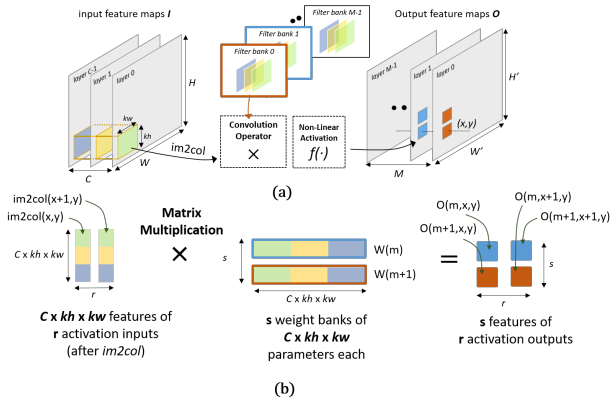



Fig. 1. (a) Dataflow of the spatial convolution kernel (b) Convolution inner loop computation as a matrix multiplication.

## B. Dataflow Schedule and Data Layout

In this subsection, we detail the dataflow schedule and data layout as implemented in the CMSIS-NN library [14], which is at the base of the proposed library. A convolution layer, standing as the basic building block for a CNN or a QNN model, produces an output feature map based on a set of weight filters and the output from the previous layer. An activation value of any output feature map is computed as the dot product between a weights filter bank and a region of the input feature map, i.e. the $C$ features values of every point under the area $kw$ x $kh$ of the filter. To efficiently implement this operation on an MCU-like device, the convolution is decomposed into two phases: an *im2col* step to load the input features of the current convolution into a contiguous memory array and a dot product. Besides the memory requirements of the activation maps and the model parameters, the *im2col* demands an extra memory footprint of $C$ x $kw$ x $kh$ values, on which the dot product operates. Fig 1(a) shows graphically this operation. Given this, the computation of one value of the output feature map, indicated as $O(m,x,y)$ becomes:

$$O(m,x,y) = \text{dot}\Big(W(m),\ im2col(x,y)\Big) \ , \quad (6)$$

where $W(m)$ is the $m$-th bank of weight filter, im2col is the unrolled input buffer of length $C$ x $kw$ x $kh$. The inner loop of the convolution dot product is realized through a matrix multiplication kernel, as depicted in Figure 1(b). In general, $s$ output features of $r$ activation outputs ($s$=2 and $r$=2 in the example in figure) can be computed at this low-level stage. As a specific case, CMSIS-NN implements a matrix multiplication kernel working on two spatially adjacent pixels of two consecutive channels inside the inner loop of the

convolution kernel; we identify this configuration as 2×2, as explained in detail in Section IV.IV-C.

Moreover, authors of [14] demonstrated the most convenient data layout to be Height-Width-Channel (HWC), as it introduces minor overhead when building the im2col buffer with respect to the Channel-Height-Width (CHW) layout. According to such a layout, the data along the channels is stored with a stride of 1, data along the width is stored with a stride equal to the number of channels $C$.

### C. Target Architectures

The target architecture of this work is based on a Parallel Ultra-Low-Power (PULP) cluster of RISC-V based processors. A commercial embodiment of this architectural template is GAP8 [16], on which we run our experiments. The GAP8 PULP cluster contains eight RISC-V cores, implementing a 4 stage in-order single-issue pipeline, supporting the RV32IMC instruction set [49], plus extensions targeting energy-efficient digital signal processing and machine learning (Xpulp) [50]. The cores are served by a 64kB L1 data memory, named Tightly-Coupled Data Memory (TCDM), enabling shared-memory parallel programming models such as OpenMP. The shared L1 can serve all memory requests accessing different banks in parallel with single cycle access latency. The 4 KB cluster program cache is also shared among the cores [51]. The cluster is also provided by an Event Unit which manages synchronization and thread dispatching, enabling low-overhead and fine-grained parallelism, thus high energy efficiency: each core waiting for a barrier is brought into a fully clock gated state. The cluster features also a DMA controller which manages the transfer between the L1 and the L2 memory (512kB in size), the latter residing outside-of-the-cluster of the GAP-8 architecture. The Xpulp extensions available in the ISA[3] include hardware loops, load/store with post-increment, Multiply and Accumulate as well as dedicated digital signal processing extensions inferred in the c code as built-in functions, presented below.

The SIMD vectorial instructions allow processing more sub-word data in parallel, most of them taking only one clock cycle. The vectorial data types to be used with such instructions are *v4s* and *v2s*: ***v4s*** allows to fill a 32bit register with four INT-8 data, ***v2s*** does the same by filling the register with two INT-16 integers, in one clock cycle. Sum of dot products SIMD instructions are provided to process either two 16 bit (*sdotp2*) or four 8 bit (*sdotp4*) integer operands in a single cycle. ***sdotp4*** takes two *v4s* data operands as input and computes the sum of dot products over the same accumulator, which is the INT-32 output of the built-in function. The ***max4*** instruction instead allows to compare two *v4s* operands by returning the element-wise maximum, in one cycle.

***bextract*** extracts, in one clock cycle, a specified number of bits ("size") from a register, starting at a specified position ("offset"). The extracted bits are then sign-extended and stored in the destination register. The natural counterpart is the ***bitinsert*** built-in function, specifying the number of bits to be inserted ("size") to the destination register, starting from the specified position ("offset"). ***pack4*** allows to pack four INT-8 variables in a SIMD *v4s* data type in two clock cycles. Finally, the ***popcnt*** built-in returns, in one cycle, the number of bits set to one in a word which is passed to the function as input.

[3]https://github.com/pulp-platform/riscv/tree/master/doc

## IV. PULP-NN Library

This section introduces the PULP-NN library and describes the optimization of the kernels with the presented RV32IMCXpulp extended ISA on a parallel cluster of eight processors and the optimization of the main computational kernel of the library: the matrix multiplication. We focus on the computational part since we are interested in exploring software solutions capable of achieving high computing performance and energy efficiency, on top of parallel edge architectures like PULP.

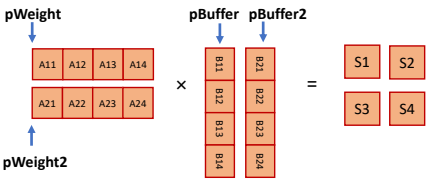
### A. Implementation and Optimization on RISC-V

We present implementation details of the most significant QNN kernels on the target RV32IMCXpulp ISA. The experiments are conducted assuming that all the data resides in L1 memory of the PULP cluster.

***INT-8 Kernels:*** The first layer for which we detail the implementation is the convolution one. We first consider the INT-8 kernel, as it also provides a basis for the implementation of INT-4 and INT-2. Starting from the implementation presented in section III.III-B, with a 2×2 matrix multiplication kernel, we optimize it to fully exploit the RV32IMCXpulp ISA. Since the matrix multiplication operation has to be looped over the size of each filter bank ($C$ x $kw$ x $kh$), we take advantage of the *hardware loops* to accelerate the *for* statement. In the inner loop, we also exploit the load and store with post-increment since the access pattern to the im2col and filter elements is extremely regular by construction. In the same way, we use the 8-bit SIMD instructions to increase the throughput of the computation. Figure 2 graphically schematizes the execution of the inner loop of the matrix multiplication kernel and reports the corresponding assembly code.

After filling two im2col buffers that are needed to compute two spatially adjacent output pixels, the matrix multiplication inner loop takes place as follows. At every iteration of the loop, four consecutive elements are loaded into the register file from each of the two im2col buffers (pointers *pBuffer1* and *pBuffer2* in the figure), and from two weight banks (pointers *pWeight* and *pWeight2*), after casting INT-8 pointers to *v4s*. The total number of load operations required is four. In this way we have sufficient elements to set four *sdotp4* built-in functions over four different accumulators. Hence, in a single run of the inner loop of the matrix multiplication kernel, we can compute four sdotp4 instructions, which correspond to 16 MAC operations, at the cost of four load instructions.

Since the fully connected kernel is a simple matrix by vector multiplication, the previous methodology naturally scales to it. Here there is no need to build the im2col buffer since the




| Pseudocode | Disassembled pseudocode | |
|---|---|---|
| For i=0 to size/4 | lp.setup | // hardware loop |
| V4s A1 = *((v4s*) pWeight); | p.lw w1, 4(a0!) | // load with post-increment |
| V4s A2 = *((v4s*) pWeight2); | p.lw w2, 4(a1!) | |
| V4s B1 = *((v4s*) pBuffer); | p.lw x1, 4(a2!) | |
| V4s B2 = *((v4s*) pBuffer2); | p.lw x2, 4(a3!) | |
| S1 = sdotp4(A1,B1); | pv.sdotsp.b s1, w1, x1 | // 4x8bit vectorial sdotp |
| S2 = sdotp4 (A1, B2); | pv.sdotsp.b s2, w1, x2 | |
| S3 = sdotp4(A2, B1); | pv.sdotsp.b s3, w2, x1 | |
| S4 = sdotp4(A2, B2); | pv.sdotsp.b s4, w2, x2 | |
| end | end | |

Fig. 2. 2×2 sized matrix multiplication kernel for INT-8 data operands.

spatial dimension of the filters is the same size as the spatial dimension of the input feature map. To reduce load instructions and exploit a data reuse mechanism, the fully connected kernel implements $2x1$ matrix multiplication kernel within the inner loop (see Section IV.IV-C and Figure 6). By loading two different subsets of weights, we can compute two consecutive output pixels along the channel dimension. By using the SIMD ISA extensions as before, with three loads we are able to set two *sdotp4* vector operations per loop cycle, which translates in 8 MACs.

Ancillary operations also take benefit of the DSP extensions. ReLU, which consists of a simple max looped over the input feature map, exploits *hardware loops*, load store with post-increment and the SIMD *max4* built-in instruction. The same is also used to optimize the max-pooling kernel, which is implemented in two steps: first along the width dimension, working destructively *in situ* on the input buffer; then along the height dimension.

***Sub-byte Extensions:*** The smallest data type well supported by the ISA with the SIMD extensions is INT-8. To exploit efficiently such vector operations, it is necessary to provide additional support functions to convert sub-byte data, i.e. INT-2 and INT-4, into INT-8. Having sub-byte operands compactly stored in memory, in the case of INT-4 data two consecutive elements are placed in a single byte. The casting operation, realized through the *pulp_nn_int4_to_int8* function, takes place either when building the im2col buffer as well as in the innermost loop of the matrix multiplication kernel to "unpack" weight elements. To reduce the overhead due to the unpacking operations, combined use of the *bextract* and *pack4* built-in functions allows to extract four INT-4 elements (weights or pixels) with few instructions, as shown in Figure 3. After loading eight INT-4 data with a single load, four elements are extracted by means of the *bitextract* built-in and packed into one single SIMD *v4s* variable, which feeds the matrix multiplication kernel.

The results of the matrix multiplication kernel (which is always performed with the INT-8 data type) are 16-bit long, as the accumulator features a precision higher than operands,

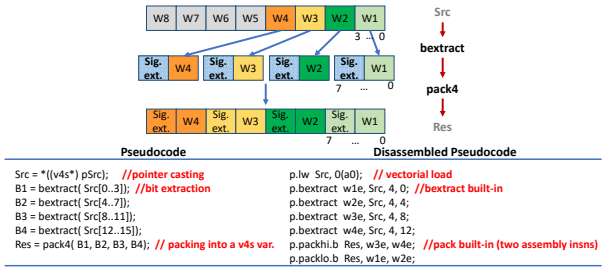



Fig. 3. INT-4 to INT-8 unpacking function.

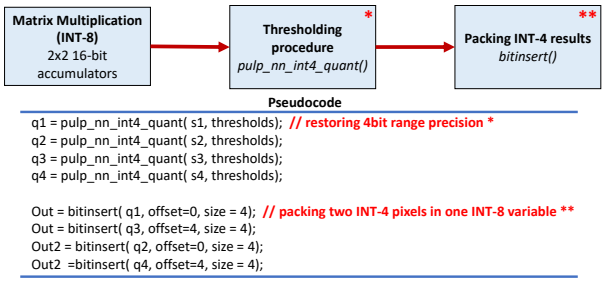



Fig. 4. The compression procedure for INT-4 data types.

as described in Section III.III-A. A compression procedure is thus needed to bring the result back to INT-4. Starting from the considerations in [15], the 16-bit accumulator is compared with the corresponding $2^4 - 1$ threshold values, using an optimal balanced binary tree function, named *pulp_nn_int4_quant*. Such a procedure is necessary to restore the precision of the results in a 4 bit range. To save memory footprint, two consecutive output INT-4 data are stored in a single-byte variable using the *bitinsert* built-in function. A graphical explanation of the compression mechanism is provided in Figure 4. A similar process is implemented for INT-2 convolutions, by featuring dedicated *packing* and *unpacking* functions.

***Binary Convolution Kernel:*** For the INT-1 data representation no casting/unpacking is needed because of the natural support provided by the ISA for binary operations. We exploit the bitwise instructions to implement the convolution kernel, which is based on bitwise XNOR operations between binary weights and binary inputs. The accumulator is filled by counting the number of ones occurring after the XNOR. To this purpose we use *popcnt* built-in. The 16-bit accumulator is compared with a single threshold and results either in a zero or one, stored back into memory by means of the *bitinsert* built-in function.

### B. Multicore Execution

As discussed above the *convolution kernel* execution consists of two phases: the im2col function and the matrix multiplication kernel. The proposed data-parallel multi-core optimization is motivated by the HWC format used to store pixels and weights and by the two phases of the dataflow. Because of the HWC format, it is convenient to split the workload along the spatial dimension of the output feature

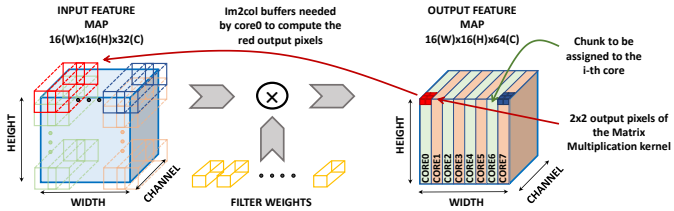
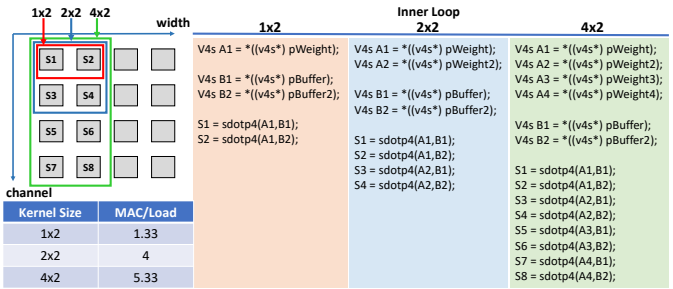



Fig. 5. The right side of the figure shows how the chunks are assigned to the 8 cores of the PULP cluster. To take advantage of the HWC data-layout each chunk is built along the spatial dimension of the output feature map. The left side gives a graphical intuition of the need each core has to create its private im2col buffer. Considering the 2×2 matrix multiplication kernel each core requires two private buffers of such type.




| Kernel Size | MAC/Load |
|---|---|
| 1x2 | 1.33 |
| 2x2 | 4 |
| 4x2 | 5.33 |

Fig. 6. Inner loop of the matrix multiplication considering different sizes of the kernel.

map, in a way that each core computes the full set of $M$ output features for a given output spatial coordinate, as shown in Figure 5. To implement this strategy, each core requires a private im2col buffer. More specifically, if we consider the 2×2 kernel, each core must allocate and load two im2col buffers before running the matrix multiplication kernel. Therefore, the parallelization boost comes at the cost of a small amount of additional memory footprint for the extra im2col buffers, which in the worst case (eight cores configuration) is about 9% of the total when considering 16×16×32 sized input feature map, 16×16×64 sized output feature map and 64×3×3×32 sized 3D convolution filter. The weights instead are shared among the cores.

Since the fully connected layer generates a set of neurons as output (i.e., the output feature map does not extend along any spatial dimension), the only dimension along which we can split the workload is the channel. We assign a balanced number of neurons to be computed to each core. The parallelization of the ReLu and the Max Pooling kernel is straight-forward: the chunk to be assigned to each core is a balanced group of pixels along the entire input feature map.

### C. Matrix Multiplication Kernel Size Exploration

To further increase the throughput of a memory intensive kernel such as matrix multiplication, it is important to reduce the cost of loading the operands into the registers as much as possible, by maximizing the *data reuse* at the register file level.

The direct implementation of the Equation (6) would be inefficient since, from a computation perspective, two loads are required (one to fetch an im2col element and one to fetch a weight parameter) to feed the MAC instruction. In this scenario, one load stall will be necessarily paid, degrading the IPC metric and reducing the throughput. To avoid the stall, multiple output data can be computed within the inner loop of the dot product routine, i.e., the inner loop of the matrix multiplication kernel.

When applying equation (6) to compute the output data at the spatial coordinate $(x+1, y)$, the formula becomes:

$$O(m, x+1, y) = \mathrm{dot}\Big(W(m),\ im2col(x+1, y)\Big)\ . \qquad (7)$$

We can notice that the same subset of weights is used in the computation of the output data at coordinates $(x, y)$ and $(x+1, y)$ . What changes is only the im2col buffer. When operating on these two point simultaneously, the inner loop consists of two dot product operations, which are performed over two different accumulators. By reusing the register that stores the elements of $W(m)$ along the spatial dimension we can set two *sdotp4* operations at the cost of one additional load (three in total), needed to fetch the elements of the second im2col buffer. So doing, we build the 1x2 sized kernel and increment the MAC to load ratio. If extending this strategy also to the feature dimension, the inner loop of the convolution can operate on a 2×2 sized kernel, i.e. computing four accumulations related to two features of two separate output pixels $(x, y)$ and $(x+1, y)$. Such a kernel size is the one used by ARM CMSIS-NN. In this case, an additional subset of weights, $W(m+1)$ is needed and, at the cost of four loads, we can perform four *sdotp4* operations in the inner loop. By means of this upgrading, the MAC to Load ratio grows up to 4.

Let us consider the 4x2 sized kernel, which means we want to compute two adjacent spatial pixels along four consecutive channels of the output feature map. Following what we said before, we need to build two im2col buffers, and we need four different subsets of weights. The elements loaded in the register file are reused similarly as presented before to maximize the MAC to Load ratio. Figure 6 explains the concept of register file data reuse. As a counterpart, we can explore the 2x4 sized kernel. In this case, the reasoning is reversed. The MAC to load ratio we can achieve in both cases is 5.33, as we compute 32 MACs at the cost of 6 load operations, in a single run of the inner loop. Thus we expect a better throughput with respect to the 2×2 sized area. It is interesting to notice that in the 2x4 case, the memory footprint is slightly higher than the 4x2 sized kernel because of the two additional im2col buffers. For the same performance, the former is thus to be preferred between the two.

It is important to notice that the upscaling of the kernel size is limited by the resources available in the register file to store operands and accumulators, thus limiting the *data reuse* design space at this level. We explore such a space to find the best register file data reuse condition which maximizes the throughput. The experimental results and further considerations are provided in Section V.V-C.

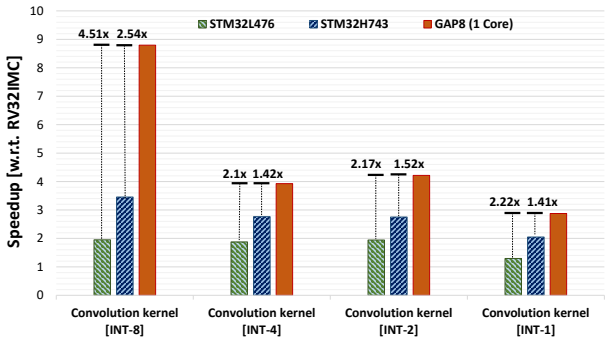



Fig. 7. Speed-up of PULP-NN conv kernels (single core execution on GAP-8) and CMSIS-NN conv kernels (on STM32H7 and STM32L4) with respect to RV32IMC ISA.

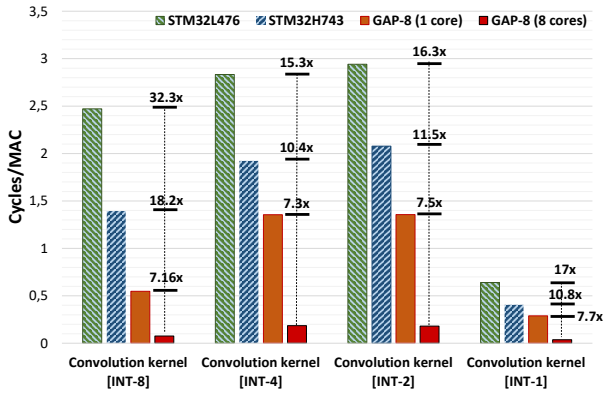



Fig. 8. Comparison in terms of cycles/MAC between the PULP-NN conv kernels on one/eight core(s) of GAP-8 cluster and CMSIS-NN conv kernels on STM32L4 and STM32H7.

## V. Experimental Results and Discussion

The solutions presented in this paper are evaluated on the off-the-shelf GAP8 [16] microcontroller, which is an embodiment of the target PULP architecture with eight cores. The same experiments can also be replicated on the open-source PULP platform[4] via RTL simulation.

### A. Comparison with RV32IMC ISA

To evaluate the proposed library, which exploits the DSP extensions available on the RI5CY processor [50], we first compare the optimized single core execution of the convolution kernels with respect to a corresponding *RV32IMC* ISA implementation, sweeping all the INT-Q datatypes supported. This evaluation is performed by benchmarking a convolution kernel operating on a 16x16x32 input tensor (HWC data-layout) with a filter size of 64x3x3x32 ($C$x$kw$x$kh$x$M$). We consider the convolution kernel as its workload is dominant when inferring an entire QNN (about 96 % on the CIFAR-10). As a second term of comparison, we run the kernels on off-the-shelf STM32H743 [38] and STM32L476 [37] commercial microcontrollers based on ARM CORTEX-M7 and CORTEX-M4 cores respectively, using the CMSIS-NN [14] library. To run the sub-byte quantized version of the convolution layer on such MCUs, we refer to [15]; the extension to the CMSIS-NN library is open access[5]. The results of the comparison are presented in terms of speedup with respect to the *RV32IMC* implementation and reported in Figure 7.

We achieve the best speedup on the INT-8 convolution kernel, mainly thanks to the 8-bit SIMD *sdotp* instructions. The ARM ISA features support for 16-bit instructions only, dividing by a factor of 2 the MAC throughput with respect to the RI5CY processor. Moreover additional rotate instructions are required on ARM architectures to pack 16-bit vector data to feed the MAC units [15]. Finally, hardware loops provide another factor of improvement with respect to ARM. Thanks to these extensions we outperform by $2.54\times$ and $4.51\times$ the STM32H7 and L4 MCUs respectively, despite the CORTEX-M7 processor available in the STM32H7 featuring a dual-issue pipeline.

When considering sub-byte data types, we notice a degradation of the speedup with respect to RV32IMC which passes from $8.8\times$ (INT-8) to $3.69\times$ and $4.22\times$ for INT-4 and INT-2 data respectively. Such degradation is due to the additional instructions to unpack and cast INT-2/4 operands to INT-8 ones. Although these operations are implemented with *bextract* and *pack4* instructions, they do not achieve the same speedup as the INT-8 convolution kernel, limiting the overall speedup for sub-byte kernels, still leading to a speedup of $1.42\times$ and $2.1\times$ with respect to STM32H7 and STM32L4 for INT-4 kernel, respectively, and a speedup of $1.52\times$ and $2.17\times$ with respect to H7 and L4 for INT-2 kernel, respectively. The ARM CORTEX-M7/M4 processors do not have ISA support for efficient bit manipulation instructions nor for popcount instruction which is helpful for the INT-1 case. However most of the computational load of this kernel is implemented with xnor instructions available in all considered ISAs. Hence, the proposed implementation, runs $1.41\times$ and $2.22\times$ faster than the extended CMSIS-NN solution on STM32H7 and STM32L4 respectively.

### B. Multicore Execution Results

In this section, we focus on the analysis of the multicore optimization of the kernels. Figure 8 shows a comparison of the convolution kernels running on the 8-core cluster of GAP-8 with respect to the equivalent CMSIS-NN implementation on STM32H7 and STM32L4. It is possible to notice that, due to the additional operations required to execute sub-byte kernels, their overall cycles/MAC are 0.186 for INT-4 and 0.181 for INT-2, both $2.4\times$ higher than the INT-8 case.However, we can notice how the software-efficient exploitation of the parallel processors cluster provides almost linear speedups ($7.16\times$ to $7.7\times$) with respect to the single core configuration, leading to a dramatic improvement of performance with respect to the equivalent execution on sequential RV32IMC (where the overall speedup passes from $8.8\times$ of the single-core execution to up $63\times$ when considering 8-cores) and on single-core

[4]https://github.com/pulp-platform.

[5]https://github.com/EEESlab/CMSIS_NN-INTQ

| Configuration | Nr. insns | I$ stall cycles | TCDM cont. cycles | Load stall cycles | Total exec. cycles | Speedup |
|---|---|---|---|---|---|---|
| **Convolution** | | | | | | |
| 1 CORE | $2546k$ | $1.3k$ (0.05%) | 0 | $18k$ (0.7%) | $2586k$ | $1\times$ |
| 2 CORES | $1286k$ | $4.5k$ (0.35%) | $1.4k$ (0.11%) | $11k$ (0.85%) | $1299k$ | $\mathbf{1.99}\times$ |
| 4 CORES | $636k$ | $5.7k$ (0.86%) | $3.8k$ (0.56%) | $5.5k$ (0.83%) | $660k$ | $\mathbf{3.92}\times$ |
| 8 CORES | $318k$ | $21.5k$ (5.96%) | $6.6k$ (1.83%) | $2.7k$ (0.75%) | $361k$ | $\mathbf{7.16}\times$ |
| **Fully connected** | | | | | | |
| 1 CORE | $20.7k$ | $0.03k$ (0.09%) | 0 | 0 | $33k$ | $1\times$ |
| 2 CORES | $10.4k$ | $1.1k$ (6.25%) | $1k$ (5.69%) | 0 | $17.6k$ | $\mathbf{1.89}\times$ |
| 4 CORES | $5.2k$ | $0.1k$ (1.19%) | $0.2k$ (2.38%) | 0 | $8.4k$ | $\mathbf{3.92}\times$ |
| 8 CORES | $2.6k$ | $0.1k$ (2.27%) | $0.3k$ (6.81%) | 0 | $4.4k$ | $\mathbf{7.52}\times$ |

TABLE II

THE TABLE SHOWS THE MULTICORE EXECUTION PROFILING OF THE KERNELS. THE MEASUREMENTS FOR MULTICORE CONFIGURATIONS ARE REPORTED AS AN AVERAGE OF THE MEASUREMENTS TAKEN ON EACH CORE. THE PERCENTAGE VALUE HIGHLIGHTS THE IMPACT OF EACH MEASURED CONTRIBUTION ON THE TOTAL EXECUTION CYCLES.

ARM architectures ($10\times$ to $32\times$). This huge performance gain enables the exploitation of the benefits of heavily quantized neural networks in terms of memory footprint, still performing one order of magnitude better than state-of-the-art ARM-based implementations.

To provide more insight on the multi-core optimizations, we present an exhaustive study of the performance achieved on the parallel cluster of GAP-8. First, we measure the amount of executed instructions per each core providing an indication of the Amdhal's limit of the kernels, i.e. the amount of cycles lost due to non-parallelizable code. As a second point, we measure the the number of cycles in which the cores are not waiting on a barrier (active cycle). Then we measure the architectural sources of overhead: number of cycles lost due to contention on the shared TCDM, cycles lost due to instruction cache stalls and cycle lost due to load stalls (read after write). The results for the convolution and fully-connected kernels are summarized in Table II.

Considering the convolution kernel, we achieve a Speedup of $7.16\times$ with eight cores. By analyzing the table we can notice that the Amdahl's limit of the kernels is around $8\times$ (thus, ideal), but we lose a small number of cycles due to architectural overheads: the 67% of this overhead is due to I$ non-idealities, 8% is due to load stalls and 20% is due to TCDM contention, which is reasonable as there are eight cores that access the same shared L1 memory. The number of I$ stalls increases with the number of cores due to the increasing contentions in the shared cache banks [51] (the banking factor of 8 can not completely remove the conflicts), on top of the I$ misses due to the large inner loop of the kernel. The parallel execution of the fully connected layer presents a speedup higher than the convolution kernel mainly thanks to the reduction of I$ stalls due to the smaller size of the kernel. The speedup is never lower than $7\times$ also when considering the max-pooling and ReLU kernels running on eight cores.

### C. Kernel Exploration

The exploration of the matrix multiplication kernel size design space is carried out for the INT-8 operands, considering sizes ranging from $1\times2$ to $4\times4$. The results are summarized in Figure 9.

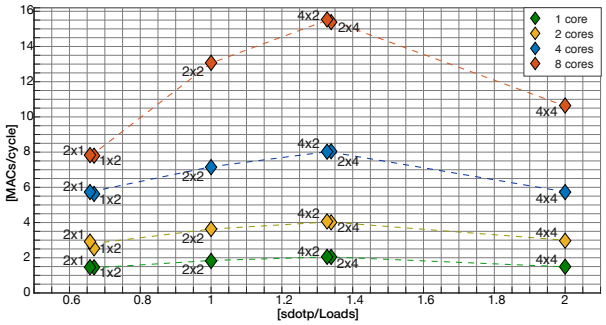



Fig. 9. Performance of the convolution layer considering different sized matrix multiplication kernels. On the x-axis we show the sdotp to load ratio to clarify how many *sdotp4* (equivalent to 4 MAC) we can set with one load. The label of each point of the graph, in the form of $a \times b$, specifies the kernel size considered. $a$ is the number of output features computed by the kernel, $b$ is the number of output activations.

A peak throughput of 15.5 MACs/cycle is reached when we consider a convolution kernel with a $4\times2$ sized matrix multiplication kernel running over eight cores of the cluster, achieving a result of just 1.01 LD/ST per MAC. This result translates in an overall efficiency of 49% in terms of MAC utilization, only a factor of 2 from the theoretical peak achievable (32 MACs/cycle) on a cluster of eight programmable cores with SIMD MAC units, i.e. considering the MAC units constantly fed. Nearly the same throughput is achieved with the $2\times4$ sized kernel, as the almost overlapping points in the graph suggests.

Then, the optimal sized kernel has been chosen taking into the account also the extra memory footprint needed to build the im2col buffers in the two configurations, which results to be lower for the $4\times2$ solution (see section 4.IV-C for more details). As regards the $1\times2$, $2\times1$ cases, they appear to be inefficient, as the amount of data reuse is meager and we pay the overhead due to the higher number of loads. For these configurations, the MAC to load ratio is slightly higher than 1. The $4\times4$ case instead would demonstrate to be the best, since

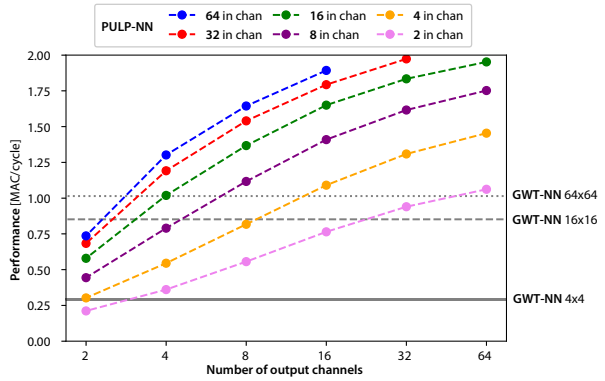



Fig. 10. Comparison between PULP-NN using a 4×2 kernel and the best result obtained by GWT-NN.

the first indication of ideal data reuse is equal to 8 (MAC/load). However, to set a 4×4 sized matrix multiplication kernel inner loop we should have at least 24 registers available (16 for the accumulators and 8 for the operands), while the target RISC-V, like most MCU-dedicated micro-architectures, has a register file with 32 general purpose registers. With only eight usable registers, the compiler has to spill variables to the stack to make room for the accumulators and operands, leading to significant performance degradation.

### D. Comparison with GAP8 Native Library

We compare our library with the optimized multi-core kernels that are openly distributed by GreenWaves Technologies as part of a proprietary tiling solution[6] and tailored for the GAP8 processor. We call this library GWT-NN. In this section, we compare the performance of PULP-NN on INT-8 data with that provided by GWT-NN. We focus on a 3×3 kernel in terms of filter size as a representative example constituting the bulk of most SoA DNNs.

Differently from PULP-NN, GWT-NN operates spatially on CHW-formatted data with explicit convolution filters working in a sliding window fashion, and accumulation over an appropriately sized INT-32 buffer. In the innermost loop, the GWT 3x3 kernel uses the register file to implement a sliding window and uses three *sdotp4* instructions to implement a total of 9 multiply-accumulate operations. [50] and [52] report further details with respect to this convolution kernel.

Figure 10 shows a comparison between the two libraries when running on a single core of the GAP-8 cluster, in terms of performance in MAC/cycle. For PULP-NN, the performance is swept by changing the number of input and output channels between 2 and 64 (only results from configurations fitting the L1 are shown). We chose the biggest input spatial size (24x24) for which configurations with 64 input or output channels fit L1. Conversely, for GWT-NN, performance is substantially independent of the number of in/out channels, but only on the spatial size of the input image; therefore, we fix their input/output channels at 4 and have them sweep their input size between 4, 16, and 64 pixels height/width.

As visible from Figure 10, PULP-NN outperforms GWT-NN for all small images, and in most cases of spatially bigger images by a significant margin. This is due to a combination of two effects: the 3x3 sliding window requires three loads and three *sdotp4* per output pixel, yielding a lower *sdotp4* per load ratio (1) with respect to the 4×2 PULP-NN kernel (1.4); moreover, only three MAC are used per each *sdotp4*, yielding a further loss of 25% in terms of efficiency. Consequently, the GWT-NN kernel is mostly competitive when the spatial size of the feature maps is much higher than the number of channels, e.g., in the first layer of a CNN. While, when the number of input/output channels is high, which typically represents the majority of the workload for state-of-the-art deep networks topologies [53], PULP-NN can achieve as much as a +89% speedup with respect to GWT-NN.

### E. Comparison with State-of-the-Art Architectures

To assess the library performance on an inference task, we run a full QNN, trained on CIFAR-10 dataset, on GAP-8, using PULP-NN back-end library. For comparison purposes, we run the same network also on State-of-the-Art edge of IoT ARM Cortex-M based microcontrollers (STM32H7 and STM32L4), using CMSIS-NN. STM32H7 and STM32L4 were chosen as representative of popular high-end and low-end MCU systems, showing a clear trade-off between performance and energy efficiency. The comparison with these two popular computing platforms allows to analyze where our results lay in terms of trade-off between computing performance and energy efficiency. The implemented network topology is composed by three convolution layers and one fully-connected layer, consisting of $26.7\,k$ parameters and $6.56$ MMACs in total[7]. The weights and the activations are quantized to INT-8 format. Such a topology is already used on IoT edge devices (MCUs) and also used by ARM to validate Neural Networks on low-power microcontrollers such as STM32L4 or STM32H7.

On GAP-8, the RGB image is initially stored in the L2 memory and brought in the L1 memory before the start of the inference task, through a DMA transfer. The activation values are then kept in the L1 memory to save on memory transfer overhead. Before the execution of each convolution or linear kernel the weights, initially residing on L2 memory, are brought in L1 through DMA as well. Also the im2col buffers are kept in L1 memory. On the STM32L4 microcontroller, the entire network is stored in the first level of memory, which consists of 128 kB SRAM. On STM32H7 the network is stored in SRAM as well and we enable also the harware data cache which is provided by the MCU architecture.

In the single core configuration, we are able to infer the entire network in 28.6 ms, when GAP-8 runs at 170 MHz. We achieve almost linear speedup when considering two and four cores, $1.99\times$ and $3.79\times$ respectively. With eight cores

[6]https://github.com/greenwaves-technologies/autotiler.

[7]The layer parameters can be found at: https://github.com/ARM-software/ML-examples/tree/master/cmsisnn-cifar10

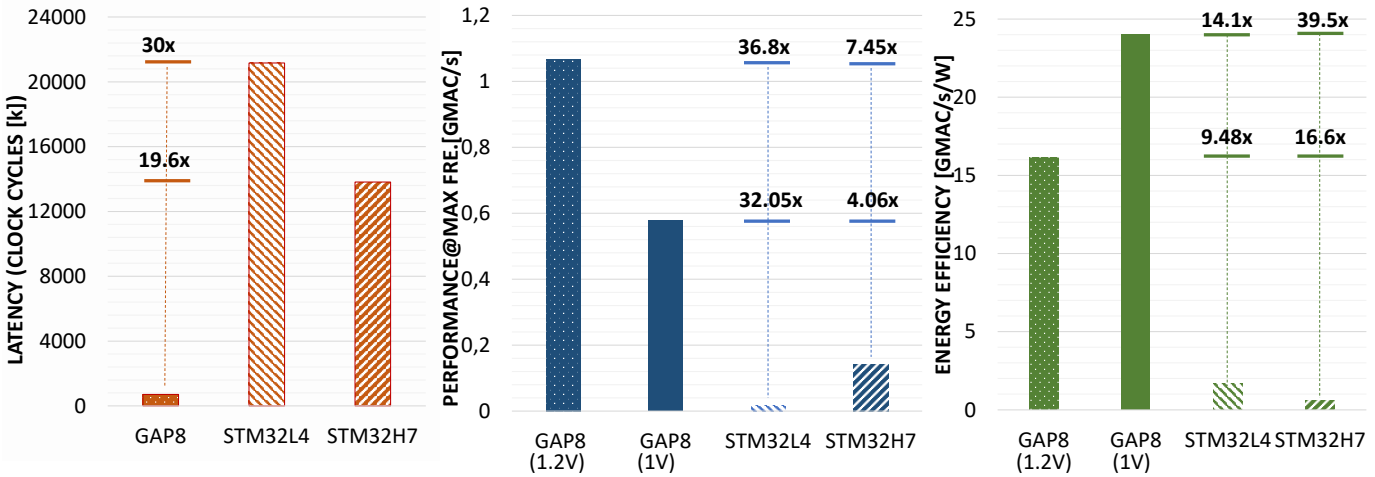



Fig. 11. This figure shows the execution cycles, the performance (at the maximum frequency) and energy efficiency (at the lowest consumption configuration) to infer the entire QNN on GAP8, STM32L4 and STM32H7 microcontrollers.

the speedup is slightly less than 7×. Figure 11 shows the comparison of PULP-NN implementation of the network on GAP-8 with respect to the CMSIS-NN implementation on STM32H743 and STM32L467 in terms of execution cycles, performance (i.e. also considering the maximum operating frequency of the devices), and energy efficiency.

Our PULP-NN CIFAR-10 achieves a peak performance of 1.07 GMAC/s at the frequency of 170 MHz and the supply voltage of 1.2 V on GAP-8, inferring 241 frame per second (fps) with an energy per inference of 0.27 mJ/frame. The performance is 7.45× better than the STM32H7 and 36.8× better than the STM32L4. The energy efficiency achieved at this operating point is 16.1 GMAC/s/W, 16.6× higher than the STM32H7 and 9.48× higher than STM32L4. At the same time, at the best energy point, at the supply voltage of 1V, PULP-NN achieves a performance of 577 MMAC/s on GAP-8, with energy efficiency of 24 GMAC/s/W, inferring 127 fps with 0.19 mJ/frame, and outperforming STM32H7 by 4.06× and STM32L4 by 32.05× in terms of performance and by 39.5× and 14.1× the same devices respectively, in terms of energy efficiency.

### F. Discussion

In this work we demonstrate that coupling optimized software libraries with a parallel ultra low power computing platform we achieve energy proportionality where, as opposed to commercial ARM-based solutions, we do not have to trade performance with energy efficiency, paving the way to fully software programmable CNN inference at the edge of the IoT. However, sub-byte kernels still suffer from drop-off in performance when compared to the INT-8 ones, despite their execution on GAP-8 performs more than one order of magnitude better with respect to MCU-based SoA solutions. The overhead, as highlighted in section V.V-A, is due to the hardware support of the target architecture only for 8-bit SIMD instructions, which makes necessary to introduce additional packing and unpacking functions. The sub-byte precision QNNs though, provide several advantages when deployed at the edge, since their memory footprint results much lower than the one of full precision NNs [7], making them more suitable to fit the limited memory capacity of MCU-like devices. Moreover, it has the potential to increase the energy efficiency, crucial for battery-powered devices [11]. Such enormous advantages might be counterbalanced by a drop-off in accuracy of the Network. Recent research though demonstrated that, by exploiting specific retraining techniques, the accuracy drop can be kept close to the floating point counterpart, leading to a cumulative loss which is acceptable for many IoT applications [13]. Based on that result, the research community is focusing more and more on the study and implementation of strongly quantized NNs. It is therefore important going further in the work presented in this paper to disclose fully the potential of heavily quantized networks on fully programmable edge devices. From the hardware perspective, providing the target ISA with sub-byte hardware SIMD operations will be a step forward to eliminate the software overhead and to double, at least, the performance and the energy efficiency with respect to the current optimal 8-bit solution.

## VI. Conclusion

We have presented PULP-NN: an optimized library to run QNNs at the edge, targeting INT-8, INT-4, INT-2, and INT-1 data operands. We showed that, by optimizing the library with the SIMD extensions and bit manipulation instructions of the targeted architecture, we heavily increase the performance of each kernel by up to 63x with respect to a corresponding RISC-V *IMC* implementation, in an eight core cluster configuration. Running an entire INT-8 QNN on GAP8 showed us that we can achieve a speedup (in terms of cycles) of 19.49× with respect to the inference of the network on an STM32H7 microcontroller, using CMSIS-NN library. Furthermore, the energy efficiency achieved on GAP8 results to be 24 $GMAC/s/W$, 14.1× higher with respect to the one

obtained with STM32L4 board. We conclude the same also for the performance: GAP8 achieves $1.066\ GMAC/s$, which is $7.45\times$ higher than the performance of STM32H7 board.

## ACKNOWLEDGEMENTS

This work was supported in part the OPRECOMP (Open trans-PREcision COMPuting) project founded from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 732631.

## REFERENCES

[1] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. N. Hindia, "An overview of internet of things (iot) and data analytics in agriculture: Benefits and challenges," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3758–3773, 2018.

[2] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, and S. Andreescu, "Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges," in *2015 IEEE International Conference on Services Computing*. IEEE, 2015, pp. 285–292.

[3] N. H. Motlagh, M. Bagaa, and T. Taleb, "Uav-based iot platform: A crowd surveillance use case," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 128–134, 2017.

[4] C. A. Tokognon, B. Gao, G. Y. Tian, and Y. Yan, "Structural health monitoring framework based on internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 3, pp. 619–635, 2017.

[5] 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.

[6] F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Gürkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou *et al.*, "An iot endpoint system-on-chip for secure and energy-efficient near-sensor analytics," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2481–2494, 2017.

[7] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[8] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[9] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization," *arXiv preprint arXiv:1811.08886*, 2018.

[10] 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.

[11] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2017, pp. 1921–1925.

[12] F. Conti, P. D. Schiavone, and L. Benini, "Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, 2018.

[13] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," *arXiv preprint arXiv:1905.13082*, 2019.

[14] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[15] M. Rusci, A. Capotondi, F. Conti, and L. Benini, "Work-in-progress: Quantized nns as the definitive solution for inference on low-power arm mcus?" in *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2018, pp. 1–2.

[16] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "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)*. IEEE, 2018, pp. 1–4.

[17] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gürkaynak, A. Teman, J. Constantin, A. Burg, I. Miro-Panades, E. Beignè *et al.*, "Energy-efficient near-threshold parallel computing: The pulpv2 cluster," *Ieee Micro*, vol. 37, no. 5, pp. 20–31, 2017.

[18] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: An efficient hardware accelerator for convolutional neural networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[19] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "An automatic rtl compiler for high-throughput fpga implementation of diverse deep convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[20] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for fpga-based convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[21] P. Meloni, A. Capotondi, G. Deriu, M. Brian, F. Conti, D. Rossi, L. Raffo, and L. Benini, "Neura ghe: Exploiting cpu-fpga synergies for efficient and flexible cnn inference acceleration on zynq socs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, p. 18, 2018.

[22] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *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.

[23] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on fpga," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–7.

[24] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[25] "Lattice. 2019. lattice sensai delivers 10x performance boost for low power, smart iot devices at the edge." https://www.latticesemi.com/About/Newsroom/PressReleases/2019/201911sensAI.

[26] "Lattice. 2018. accelerating implementation of low power artificial intelligence at the edge." http://www.latticesemi.com/view_document?document_id=52384.

[27] G. Desoli, N. Chawla, T. Boesch, S.-p. Singh, E. Guidetti, F. De Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh *et al.*, "14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 238–239.

[28] L. Cavigelli and L. Benini, "Origami: A 803-gop/s/w convolutional network accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2461–2475, 2017.

[29] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[30] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[31] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[32] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[33] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "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.

[34] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An architecture for ultralow power binary-weight cnn acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 48–60, 2018.

[35] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda *et al.*, "Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, 2018.

[36] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 488–490.

[37] "Stmicroelectronics. 2018. stm32l476 datasheet," https://www.st.com/resource/en/datasheet/stm32l476je.pdf.

[38] "Stmicroelectronics. 2018. stm32h743 datasheet," https://www.st.com/resource/en/datasheet/stm32h743bi.pdf.

[39] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr. wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing," *IEEE Journal of Solid-State Circuits*, 2019.

[40] "Nvidia. 2015. nvidia tegra x1," https://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf.

[41] "Nvidia. 2015. gpu-based deep learning inference: A performance and power analysis," https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf.

[42] "Nvidia. 2018, september. nvidia turing architecture in-depth," https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/.

[43] "Raspberry pi compute module 3+. 2019," https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM3plus_1p0.pdf.

[44] "Arm. 2019. armv8.1-m architecture," https://pages.arm.com/introduction-armv8.1m.html?_ga=2.237285124.508798244.1553788782-2017191492.1542023072.

[45] "Arm.project trillium machine learning platform," https://www.arm.com/products/silicon-ip-cpu/machine-learning/project-trillium.

[46] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 683–688.

[47] "2018. kendryte: K210 datasheet," https://s3.cn-north-1.amazonaws.com.cn/dl.kendryte.com/documents/kendryte_datasheet_20181011163248_en.pdf.

[48] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[49] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, "The risc-v instruction set manual, volume i: User-level isa, version 2.0, eecs department," *University of California, Berkeley*, 2014.

[50] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[51] I. Loi, A. Capotondi, D. Rossi, A. Marongiu, and L. Benini, "The quest for energy-efficient i $ design in ultra-low-power clustered many-cores," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 2, pp. 99–112, 2018.

[52] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64mw dnn-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, 2019.

[53] 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.