CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices

Alessandro Capotondi, *Member, IEEE*, Manuele Rusci, Marco Fariselli, and Luca Benini, *Fellow, IEEE*

***Abstract*—Low-precision integer arithmetic is a necessary ingredient for enabling Deep Learning inference on tiny and resource-constrained IoT edge devices. This brief presents CMix-NN, a flexible open-source[1] mixed low-precision (independent tensors quantization of weight and activations at 8, 4, 2 bits) inference library for low bitwidth Quantized Networks. CMix-NN efficiently supports both Per-Layer and Per-Channel quantization strategies of weights and activations. Thanks to CMix-NN, we deploy on an STM32H7 microcontroller a set of Mobilenet family networks with the largest input resolutions (224x224) and higher accuracies (up to 68% Top1) when compressed with a mixed low precision technique, achieving up to +8% accuracy improvement concerning any other published solution for MCU devices.**

***Index Terms*—Artificial neural networks, inference mechanisms, microcontrollers, TinyML, edge computing.**

I. Introduction

RUNNING inference tasks at the edge of the sensing infrastructure minimizes the user's network bandwidth and improves the response time [1]. When envisioning battery-powered IoT devices with smart capabilities, the limited energy budget puts constraints on the design of computing platforms for edge computing. Microcontroller Units (MCUs), which dominate the low-power spectrum of the computing platforms, typically feature limited memory resources (up to few MB of FLASH and below 1 MB of on-chip RAM) and lack floating-point hardware support. Hence, the deployment of high computational and memory requirements of deep learning workloads [2] on edge devices results exceptionally challenging.

Quantization methods aim at compressing deep network parameters and temporary values to either fit the memory constraints and lower the execution latency by exploiting SIMD vector instructions. It is well understood that 8 bits quantization can be achieved at almost-zero accuracy degradation, even without a retraining process [3], [4]. However, such kind of quantization level results not sufficient when addressing complex problems requiring deep inference models. As an example, the memory requirement of the most accurate 8-bit MobilenetV1 (70.1% on Imagenet) is higher than the memory footprint of an STM32H7 microcontroller. To increase the compression rate while paying a limited accuracy drop, several works exploit sub-byte, i.e., less than 8 bits, quantization based on retraining flows [5]–[9]. This class of work focuses on homogeneous quantization, in which the same number of bits is used for both weights and activations, and does not consider implementation aspects on resource-constrained devices. To close the gap between model compression and deployment, some works proposed a lossless threshold-based compressor, to convert the output of the integer convolution into the quantized input of the next convolutional layer [10]–[12]. In contrast, [13] proposed a compact integer-only 8-bit quantization methodology, requiring less memory footprint than threshold-based approaches by performing the folding of batch normalization parameters into convolutional weights before applying Per-Layer quantization. The approach described in [14] leveraged heterogeneous mixed-precision to deploy deep inference networks on tiny MCUs. The proposed technique aims at cutting the number of bits of individual weight or activation tensors below 8 bits up to fit the memory constraints and, at the same time, paying a limited accuracy drop if compared to the full-precision network. However, despite the numerous works addressing quantization on the server-side, no solution is provided for the deployment phase, especially if considering the mixed-precision sub-byte scenario.

To tackle this problem, we present CMix-NN, an open-source mixed-precision library for quantized neural networks deployment on microcontroller targets. Differently, from state-of-the-art deployment solutions available for MCUs, our library supports convolutional kernels with any bit precision in the set of 8, 4 and 2 bits, for any of the convolution operands. Thanks to our library, this brief describes, for the very first time, the deployment of a MobilenetV1 capable to achieve up to 68% Top-1 accuracy for the Imagenet classification task on an STM32 MCU device. The result is 8% better than the highest accuracy reported so far in the literature.

This brief: i) Describe a methodology to turn a Mixed Low-Precision network, quantized according to the state-of-the-art approaches, into an integer-only deployment network; ii) Expose the CMix-NN library for Mixed Low-Precision

Manuscript received February 3, 2020; accepted March 7, 2020. Date of publication March 27, 2020; date of current version May 6, 2020. This work was supported by the EU ECSEL-H2020 AI4DI Project under Grant 826060. This brief was recommended by Associate Editors Yajun Ha and Edoardo Bonizzoni. *(Corresponding author: Alessandro Capotondi.)*

Alessandro Capotondi is with the Department of Physics, Mathematics and Informatics, University of Modena and Reggio Emilia, 41125 Modena, Italy (e-mail: alessandro.capotondi@unimore.it).

Manuele Rusci and Marco Fariselli are with the DEI, University of Bologna, 40126 Bologna, Italy (e-mail: manuele.rusci@unibo.it; marco.fariselli2@studio.unibo.it).

Luca Benini is with the DEI, University of Bologna, 40126 Bologna, Italy, and also with the IIS, ETH Zürich, 8092 Zürich, Switzerland (e-mail: luca.benini@unibo.it).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSII.2020.2983648

[1]CMix-NN is available at https://github.com/EEESlab/CMix-NN.

convolutional, discussing our optimized software implementation and giving insights on the proposed solution; iii) Provide a quantitative analysis of the impact of different quantization strategies on different MobilenetV1 models running on ARM Cortex-M7.

II. Related Work

Concerning the deployment of deep inference networks, several works introduced frameworks, software stacks, and hardware solutions optimized in terms of latency and energy consumption [2], [15]–[19]. When targeting mid-high processors, such as ARM Cortex-A cores, optimized software backends rely on a DSP-oriented implementation to accelerate the basic computational kernel for deep learning workloads: the matrix multiplication [20]–[22]. However, these libraries are tailored for cores featuring an energy consumption exceeding the requirement of battery-operated systems and do not support sub-bytes data-type compressed operands, hence lacking computation support for ultra low-precision quantized models. To deploy these networks on ARM Cortex-A cores, some works rely on bit-serial implementations [10], [23], [24]. These software stacks, decompose low-bitwidth convolutions into bitwise operations. If looking at processing units strongly optimized in terms of power consumption, i.e., the MCU world, current software stacks still lacks in terms of software support for DL model deployments, mainly due to the severe computational and memory limitations. To fill this gap, STMicroelectronics released X-CUBE-AI, an expansion pack for automating the deployment of 32-bit floating point and 8-bit quantized DL high-level models into low-end STM32 microcontrollers [25]. CMSIS-NN [26], which is the current state-of-the-art software stack for inference on ARM Cortex-M devices, operates convolution operators on 8, 16 bits data. Unfortunately, the usage of the library has been demonstrated only for tiny models, limited to less than ten classes classification, hence not requiring aggressive quantization fine-tuning [27], [28]. CMix-NN enables easy and effective deployment on tiny MCU devices of arbitrarily quantized CNN models, whose topologies would never meet the typical memory requirements if utilizing state-of-the-art inference libraries. To speed up the inference phase on parallel RISC-V architectures, PULP-NN kernels [29] exploit 4x8 bits SIMD MAC instructions and bit-wise extension to gain benefits in case of bit-precision lower than 8 bits. However, this library provides support only for homogeneous quantization, hence not addressing mixed-precision, and is optimized for data residing in L1 memory, which prevents the application on large and complex models, such as Imagenet MobilenetV1.

III. CMix-NN: Mixed-Precision Inference Library

CMix-NN is a flexible inference framework targeting arbitrarily mixed-precision inference. The library is optimized for the ARM instruction set with vector arithmetic extensions tailored for ARMv7-M ISA (ARM Cortex-M4 and Cortex-M7). The core of the library is composed of a complete set of convolutional kernels featuring a mixed low-bitwidth for the weights, input and output activations. Any combination of 8, 4, 2 bitwidth is supported, and different quantization strategies.

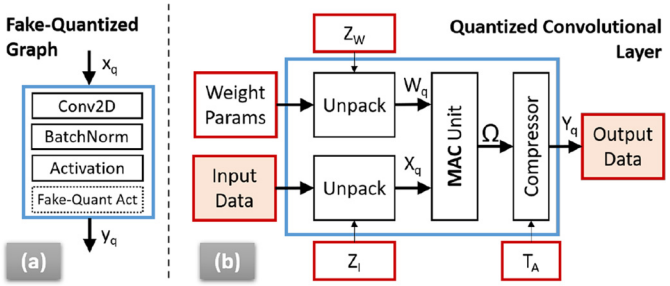

Fig. 1. (a) Fake-Quantized Graph. (b) Quantized Convolutional Layer. Dataflow model for generic forward derived from a fake-quantized sub-graph.

A. Integer-Only Quantized Networks

The quantization of weights and activations values maps every real value parameter into one of the 2^Q discrete levels, where Q is the target number of bits. In case of uniform quantization, every tensor $\mathbf{t}$, either representing weights or activations or only a subset of them, can be quantized across the numeric range $[T_L, T_H]$ with a given number of bits Q [13], such as:

$$t_q = \text{quant}(\mathbf{t}) = \text{round}(\frac{\text{clamp}(\mathbf{t}, T_L, T_H)}{S_t}) \cdot S_t = \mathbf{T_q} \cdot S_t \quad (1)$$

where $\mathbf{T_q}$ is an integer tensor and $S_t = \frac{T_H - T_L}{2^Q - 1}$ is a real scaling factor. Moreover, Equation (1) can be applied layer-wise or feature-wise on the output-feature dimension, depending if (T_L, T_H) parameters are computed Per-Layer (*PL* in the text) or Per-Channel (*PC*). The additional parameters of the Per-Channel quantization are justified by a higher precision of the quantized inference network [8], [14]. To turn a typical subgraph including a convolutional layer followed by a batch normalization layer (Figure 1a) into an integer-only representation, the parameters of batch normalization can be folded into the convolutional weights before the quantization step [8], [13]. We indicate this solution as *FB* in text. However, due to the high accuracy drop caused by the folding process in case of low-bitwidth weights, the Integer-Channel Normalization (ICN) technique aims at folding the extra non-convolutional parameters into the activation function itself [14].

Thanks to these strategies, every fake-quantized sub-graph of a network can be mapped into a basic building block, namely the Quantized Convolutional Layer (QCL). Figure 1b graphically illustrates the QCL internal components. Red boxes represent the memory requirements while the blue box describes the computational dataflow that implements the transfer function of a low-precision convolutional layer. Concerning this latter block, the low-precision *MAC unit* is the key module, aiming at accumulating the convolution result over a temporary high precision variable Ω, INT-32 in our case. To lower the latency, the ISA of MCU architectures typically provides vectorized MAC operations (e.g., ARMv7-M includes 2x 16 bit SIMD MAC instructions). The *Unpack* block uncompress and cast, or simply load, the operands of the convolution. Moreover, in case of asymmetric quantization of weights, an additive offset $-Z_w$ applies to the loaded parameter values to transpose them into the custom asymmetric domain. Once the integer convolution is executed,

TABLE I
QUANTIZATION COMPRESSION FACTORS

Quantization	Compressor		
PL+FB [13]	$M = \frac{S_x \tilde{S_w}}{S_y} \in \mathbb{R}^+$	$M_0 \in [0.5, 1)$	$N_0 \in \mathbb{Z}$
PL+ICN [14]	$M = \frac{S_x \tilde{S_w}}{S_y} \frac{\gamma}{\sigma} \in \mathbb{R}^n$	$M_0 \in [-1, 1)$	$N_0 \in \mathbb{Z}^n$
PC+ICN [14]	$M = \frac{S_x \tilde{S_w}}{S_y} \frac{\gamma}{\sigma} \in \mathbb{R}^n$	$M_0 \in [-1, 1)$	$N_0 \in \mathbb{Z}^n$

Note that n equals to the number of output features.

TABLE II
MEMORY REQUIREMENTS OF A QUANTIZED CONVOLUTIONAL LAYER

Quant	Z_x/Z_y	Weights	Z_w	B_q	M_0	N_0
Bits	8	Q_w	8/16	32	32	8
PL+FB	1	$OF \cdot k_w \cdot k_h \cdot IF$	1	OF	1	1
PL+ICN	1	$OF \cdot k_w \cdot k_h \cdot IF$	1	OF	OF	OF
PC+ICN	1	$OF \cdot k_w \cdot k_h \cdot IF$	OF	OF	OF	OF

the *Compressor* unit operates the final compression on the high-precision accumulation Ω through a set of parameters T_A, whose nature varies depending on the applied quantization flavor. In general, the compressor can be expressed as:

$$Y_q = Z_y + \text{clamp}\left(\left\lfloor M_0 \cdot 2^{N_0}(\Omega + B_q) \right\rfloor, Y_L, Y_H\right) \quad (2)$$

where $\Omega = \sum (X - Z_x) \cdot (W - Z_w)$ is the integer high-precision accumulation and B_q is the integer bias vector, with high-precision format. Based on the chosen quantization flavor and integer transformation, some extra-parameters M can be derived as detailed in Table I. M_0 and N_0 parameters in Equation (2) express the factors M as $M_0 \cdot 2^{N_0}$, with $\text{abs}(M_0) \in [0.5, 1)$. If batch normalization is folded into convolutional weights within the scenario of Per-Layer quantization ($PL+FB$), M_0 and N_0 are scalars. Instead, when relying on ICN, M_0 and N_0 are vectors of cardinality equal to the channel-feature dimension and M_0 values can be either positive or negative because of the folding of the batch normalization γ parameter.

Table II reports the memory requirements (in terms of the amount of data and bit precision) of permanent (fixed) parameters for every QCL function, depending on the used quantization flavor.

B. CMix-NN Convolutional Layer Execution Model

Figure 2 shows the pseudocode of the generic Quantized Convolutional Layer (QCL). A mixed-precision QCL workload, like CMSIS-NN [26], splits the convolution between a *im2col* phase and a *MatMul* loop, which returns two features of two consecutive output pixels for any loop iteration. The *im2col* function, realized through the function *im2colQ*, loads Q bits input data from *tensorIn* and copy them to temporary buffers *ptrBuff0* and *ptrBuff1* after casting to 2x16 bits vectors. Due to the output-stationary nature of the kernel, the majority of the computation time is spent on the inner loop of this *MatMul* phase, named as *AccLoop*. Inside this loop, the *UnpackQ* function uncompresses 32 bits packed weight data, *tensorW*, from original Q-bits format to 2x16-bit vectors $w[i]$, as requested by the MAC unit (in the ARMv7-M case, vectorized SIMD 2x16 MAC instructions are used). Depending on the weight bit precision Q_w, 4 or 8 or 16 elements

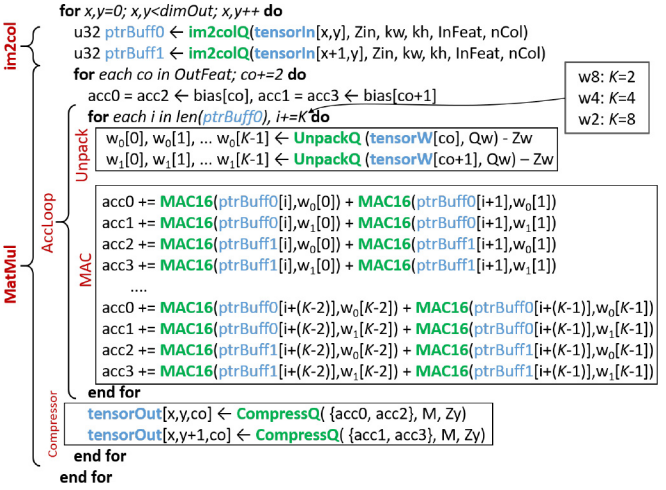

Fig. 2. Generic CMix-NN Quantized Convolutional Layer Pseudocode.

```
int32 inA = *((int32 *) tensorW);
w[0]= __UXTB16(      inA      & 0x000F000F);
w[1]= __UXTB16(__ROR(inA, 4)  & 0x000F000F);
w[2]= __UXTB16(__ROR(inA, 8)  & 0x000F000F);
w[3]= __UXTB16(__ROR(inA, 12) & 0x000F000F);
```

Fig. 3. ***UnpackQ* pseudocode for upacking a 32-bit vector of 4-bit tensors to 2x16-bit vector.** In case of 4-bit datatype, each 32-bit contains eight tensors. We extract each couple of them into four 2x16-bit vectors. The extraction of each couple of 4-bit tensors require three instructions: *__ROR*, a register right rotate; *AND*, a logical AND for the bitmask; *__UXTB16*, extend two 8-bit values to two 16-bit values.

(parametrized by K in the pseudocode) are uncompressed from a single 32 bits word loaded from the packed *tensorW* if the number of bits is, respectively, 8 or 4 or 2 bits. Once the operations of the inner loop complete, accumulation value are compressed back to Q_y bits and stored back to memory into the packed array *tensorOut*. The *CompressQ* function can implement all the compression techniques detailed in Table I and generalized by Equation (2).

In the case of 4-bit data type, each 32-bit contains eight elements. The *UnpackQ* extracts 2-by-2 elements into four 2x16-bit vectors (see Figure 3). The operation requires three instructions: a right rotate; a logical AND for bit masking; and a 2x8-bit to 2x16-bit extension. Using unsigned datatypes, the unpack requires 1.5 instructions per element, instead of 2 as proposed by [12].

C. CMix-NN Quantization Flavors

For any combination of bitwidth between input, output, and weights, CMix-NN supports Per-Layer (PL) and Per-Channel (PC) with ICN or FB (PL only) compression rules. In contrast to CMSIS-NN, CMix-NN targets asymmetric quantization for convolutional kernels. Considering the output-stationary dataflow of the QCL, it is convenient to apply additive offset on the input data within the *im2col* phase. Weights offsets are handled differently according to the type of quantization. If using Per-Layer quantization (one single Z_w offset per layer), a cumulative offset value is computed at runtime and added to the convolution bias, according to:

$$\Omega = \sum X_q \cdot (W - Z_w) = \sum X_q \cdot W - \sum X_q \cdot Z_w \quad (3)$$

TABLE III
COMPARISON WITH STATE-OF-THE ART IMAGENET MOBILENET ON A STM32H7 FITTING 2MB ROM AND 512KB R/W MEMORY

	X-CUBE-AI	CMSIS-NN	CMix-NN (our)						
Network	Mob-V1 128_0.25	Mob-V1 192_0.5	Mob-V1 192_0.5	Mob-V1 224_0.75	Mob-V1 224_0.75	Mob-V1 224_0.75	Mob-V1 192_0.5	Mob-V1 160_0.50	Mob-V1 128_0.25	Mob-V2 224_1.0
Quantization	—	PL	PL	PC+ICN	PL+ICN	PL+ICN	PC+ICN	PC+ICN	PC+ICN	PC+ICN
Tensor Type	FP32	INT8[1]	INT8[2]	*w4a4*	*mixed*		*w8a8*	*w8a8*	*w8a8*	*mixed*
Avg Act. Bits	32	8	8	4	6.72	6.72	8	8	8	5.35
Avg Wgt. Bits	32	8	8	4	5.99	5.99	8	8	8	4.61
Model Size (MB)	1.88	1.37	1.37	0.98	1.97	1.97	1.37	1.37	0.49	1.95
Latency (s)	0.206	0.437	0.510	2.290	**1.86**	**1.419**	**0.677**	**0.460**	0.110	2.013
MAC/Cycle	0.14	0.52	0.45	0.30	0.36	0.48	0.33	0.35	0.26	0.30
Accuracy Top-1	45.0%	59.5%	*59.5%*[3]	60.5%	**68.2%**	**67.0%**	**62.9%**	**61.25%**	44.6%	54.0%
Energy (μJ)	54.40	115.40	134.64	604.69	491.15	374.70	178.77	121.46	29.05	531.55

[1] Tensorflow Quantization-Aware training to INT8 datatypes for weights and activations.
[2] Post-Training Quantization to INT8 datatypes for weights and activations.
[3] Literature does not provide MobilenetV1 implementations. We optimistically report the score from Tensorflow Lite.

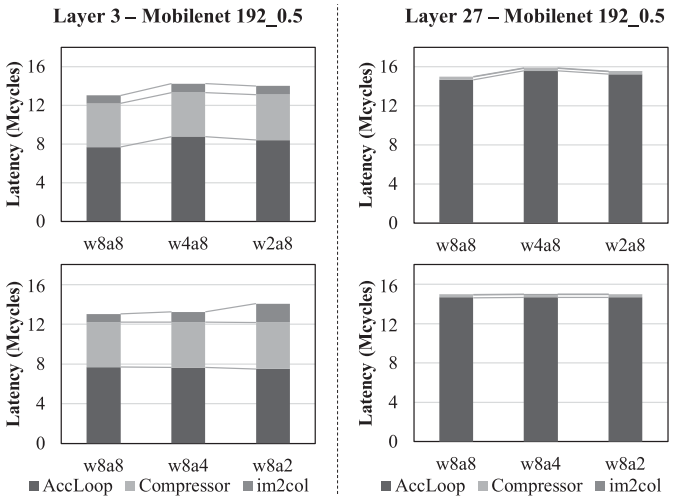

Fig. 4. Latency breakdown of MobilenetV1 Pointwise Convolutional layers using different bitwidths. Left, the measured latencies, expressed in millions on CPU Cycles (MCycles) for *AccLoop*, *Compressor*, and *Im2col* on the MobilenetV1 Layer #3 for different weight (*w*) and activation (*a*) bitwidths. Right, the same analysis for Layer #27.

This expedient lightens the inner loop in which the Z_w subtractions are no more needed. In total, we hence have only one subtraction for each element of *ptrBuff* compared to one subtraction for each MAC operation required by the general algorithm (see Figure 2). In the MobilenetV1 specific case, this optimization leads to an overall latency reduction of 5.4% in the 160_0.25 model and 8.8% in the 192_0.25. When Per-Channel quantization is applied, however, due to the larger number of Z_w (one for each output feature) and the memory limitation of MCU devices, this optimization cannot be applied, and Z_w must be applied in the *MatMul* inner loop.

IV. EXPERIMENTAL RESULTS

The experimental results aim to quantify latency-accuracy tradeoff of different quantization low-bitwidth mixed-precision strategies supported by CMix-NN focusing on the state-of-the-art DL models of the MobilenetV1 family. All the different configuration of the Mobilenet are quantized using a training-aware quantization scheme and converted into a "only-integer" forward model as described in [14]. All the experiments execute on an STM32H743 SoC, which is an ARM Cortex-M7 based MCU, running at 480 MHz, equipped with 2 MB of FLASH, 512KB of SRAM.

A. CMix-NN Low-Bitwith Characterization

We profile the first (MobilenetV1 Layer #3) and the last (MobilenetV1 Layer #27) pointwise layers of a MobilenetV1 192_0.5. The first case is representative of a layer with a low number of channels and a significant spatial dimension, while the opposite scenario is analyzed within the second case. The four charts of Figure 4 show the latency breakdown for the *im2col* extraction, the *AccLoop* and *Compression* respectively, on different bitwidths. The two histograms at the left show the time spent in millions of cycles in the case of a 32 elements *im2col* (MobilenetV1 Layer #3). The other histograms at the right refer to the case of 1K elements *im2col* size (MobilenetV1 Layer #27). We can observe that *AccLoop* computation is sensitive only to the weight quantization (upper figures). These overheads (up to 14% in case of fixed 8-bit activations) are due to the extraction operations for *UnpackQ* when low-bitwidth types are used for weights quantization. On the contrary, reducing the number of activation bits impact the *im2col* execution time (lower figures). When *im2col* is small, the unpacking of the activation tensor takes ≈10% of the whole layer execution time. Passing from *a8* to *a2* can double the time for the tensor extraction. These overheads can increase the execution time of the whole convolutional layer up to 8%. Instead, when the *im2col* size increases, the impact dramatically goes below 1%, as shown in the chart at the bottom right. In this case, the quantization of the activation becomes free of overheads.

B. CMix-NN Network-Level Comparison

We use the mixed-precision configurations of the MobilenetV1 family networks provided by Rusci *et al.* [14], setting the memory constraints $M_{RO} = 2MB$ and $M_{RW} = 512kB$, corresponding with the memory characteristics of the STM32H7 device, used in this brief. Table III compares latency, accuracy, MAC/Cycles, and Energy consumption on a set of different MobilenetV1 networks

implemented through CMix-NN to the other state-of-the-art deployments when 2MB memory constraints are considered. CMix-NN enables the deployment of a much bigger models (224_α) compared to homogeneous state-of-the-art inference libraries (192_0.5 X-CUBE-AI, 192_0.5 CMSIS-NN) achieving up to up to 68% (PC+ICN), and 67% (PL+ICN) Top-1 accuracy on the Imagenet problem. These scores surpass by 23% the score achieved by X-CUBE-AI in FP32, and by 8% the score obtainable by the 8-bit uniform quantization supported by CMSIS-NN or X-CUBE-AI. Focusing on the latency and efficiency, CMix-NN provides up to 2× MAC/Cycle compared to floating-point backend exploiting 2x16-bit SIMD operations. PC+ICN efficiency peak, measured in 224_0.75, is 0.36 MAC/Cycle, which is ≈24% less than the maximum MAC/Cycle measured in X-CUBE-AI (0.52) and PL+ICN (0.48). The mixed precision results up to 1.6x faster than homogeneous 4-bit implementation (*w4a4*), also showing much lower Top-1 accuracy. Also, the 160_0.5 PC+ICN model features nearly the same latency as the 192_0.5 with CMSIS-NN and X-CUBE-AI, but we measure a +1.75% higher accuracy. Latency and Accuracy highlighted results on the Table III belong to the Pareto front when the accuracy is higher than 60% Top-1. All of those points are obtained by the CMix-NN inference library. Finally, applying the mixed-precision quantization method presented by [14] to the recent MobilenetV2 224_1.0 we archived only 54% Top-1 accuracy, probably due to the aggressive quantization required on some critical layers (i.e., tensors of the first inverted residual blocks feature 2 bit of precision). Meanwhile, this is not the main contribution of this brief, we can conclude that the selection of the best topologies for edge devices is very critical when quantization is applied [30].

V. Conclusion

In this brief, we presented CMix-NN, an open-source, flexible inference library for MCU-based edge devices. CMix-NN provides an optimized backend to deploy state-of-the-art mixed low-precision deep neural networks on ARM Cortex-M processors. CMix-NN enables end-to-end deployment large MobilenetV1 topologies achieving up to 68% Top1 accuracy on the Imagenet problem fitting 2 MB memory. The max accuracy achieved is respectively 8%, and 23% more accurate than state-of-the-art 8-bit, and FP32 models fitting 2 MB memory. Low-bitwidth mixed-precision latency effects are dominated by the lack of sub-byte and mixed-precision SIMD operations at MCU-class ISA. Near-to-future ISA extensions will fill this gap, enabling memory and latency efficient deployment of mixed low-bitwidth deep neural networks.

References

[1] C.-J. Wu *et al.*, "Machine learning at Facebook: Understanding inference at the edge," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2019, pp. 331–344.

[2] A. Canziani, E. Culurciello, and A. Paszke, "Evaluation of neural network architectures for embedded systems," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2017, pp. 1–4.

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

[4] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," 2019. [Online]. Available: arXiv:1906.04721.

[5] Y. Choukroun, E. Kravchik, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," 2019. [Online]. Available: arXiv:1902.06822.

[6] S. R. Jain, A. Gural, M. Wu, and C. Dick, "Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware," 2019. [Online]. Available: arXiv:1903.08066.

[7] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," 2019. [Online]. Available: arXiv:1902.08153.

[8] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018. [Online]. Available: arXiv:1806.08342.

[9] Z.-G. Liu and M. Mattina, "Learning low-precision neural networks without straight-through estimator (STE)," 2019. [Online]. Available: arXiv:1903.01061.

[10] Y. Umuroglu and M. Jahre, "Streamlined deployment for quantized neural networks," 2017. [Online]. Available: arXiv:1709.04060.

[11] H. Gao, W. Tao, D. Wen, T.-W. Chen, K. Osa, and M. Kato, "IFQ-Net: Integrated fixed-point quantization networks for embedded vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2018, pp. 607–615.

[12] 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 *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ ISSS)*, 2018, pp. 1–2.

[13] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.

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

[15] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 253–256.

[16] M. Blott *et al.*, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfig. Technol. Syst. (TRETS)*, vol. 11, no. 3, p. 16, 2018.

[17] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: An efficient hardware accelerator for convolutional neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2017, pp. 1–4.

[18] L. Bai, Y. Zhao, and X. Huang, "A CNN accelerator on FPGA using depthwise separable convolution," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 10, pp. 1415–1419, Oct. 2018.

[19] J. Lee, J. K. Eshraghian, K. Cho, and K. Eshraghian, "Adaptive precision cnn accelerator using radix-x parallel connected memristor crossbars," 2019. [Online]. Available: arXiv:1906.09395.

[20] M. Dukhan, Y. Wu, and H. Lu, *Qnnpack: Open Source Library for Optimized Mobile Deep Learning*. Accessed: Dec. 9, 2019. [Online]. Available: https://engineering.fb.com/ml-applications/qnnpack/

[21] E. Wang *et al.*, "Intel math kernel library," in *High-Performance Computing on the Intel® Xeon Phi™*. Cham, Switzerland: Springer, 2014, pp. 167–188.

[22] B. Jacob *et al.*, "Gemmlowp: A small self-contained low-precision GEMM library, Google Inc.," Accessed: Mar. 31, 2020. [Online]. Available: https://github.com/google/gemmlowp.

[23] M. Cowan, T. Moreau, T. Chen, and L. Ceze, "Automating generation of low precision deep learning operators," 2018. [Online]. Available: arXiv:1810.11066.

[24] A. Tulloch and Y. Jia, "High performance ultra-low-precision convolutions on mobile devices," 2017. [Online]. Available: arXiv:1712.02427.

[25] *X-Cube-AI: AI Expansion Pack for STM32CubeMX*, STMicroelectronics, Geneva, Switzerland, Accessed: Dec. 9, 2019. [Online]. Available: https://www.st.com/en/embedded-software/x-cube-ai.html

[26] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm cortex-M CPUs," 2018. [Online]. Available: arXiv:1801.06601.

[27] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," 2017. [Online]. Available: arXiv:1711.07128.

[28] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," 2019. [Online]. Available: arXiv:1906.05721.

[29] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," 2019. [Online]. Available: arxiv:1908.11263.

[30] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization," 2018. [Online]. Available: arXiv:1811.08886.