Fast and Accurate Inference on Microcontrollers with Boosted Cooperative Convolutional Neural Networks (BC-Net)

Fast and Accurate Inference on Microcontrollers With Boosted Cooperative Convolutional Neural Networks (BC-Net)

Luca Mocerino, *Student Member, IEEE* and Andrea Calimera, *Member, IEEE*

***Abstract*—Arithmetic precision scaling is mandatory to deploy Convolutional Neural Networks (CNNs) on resource-constrained devices such as microcontrollers (MCUs), and quantization via fixed-point or binarization are the most adopted techniques today. Despite being born by the same concept of *bit-width lowering*, these two strategies differ substantially each other, and hence are often conceived and implemented separately. However, their joint integration is feasible and, if properly implemented, can bring to large savings and high processing efficiency. This work elaborates on this aspect introducing a boosted collaborative mechanism that pushes CNNs towards higher performance and more predictive capability. Referred as *BC-Net*, the proposed solution consists of a self-adaptive conditional scheme where a lightweight binary net and an 8-bit quantized net are trained to cooperate dynamically. Experiments conducted on four different CNN benchmarks deployed on off-the-shelf boards powered with the MCUs of the Cortex-M family by ARM show that BC-Nets outperform classical quantization and binarization when applied as separate techniques (up to 81.49% speed-up and up to 3.8% of accuracy improvement). The comparative analysis with a previously proposed cooperative method also demonstrates BC-Nets achieve substantial savings in terms of both performance (+19%) and accuracy (+3.45%).**

***Index Terms*—Edge AI, Deep Learning, Convolutional Neural Networks, Low-power, Microcontroller, Binary.**

I. Introduction

WITH the record-breaking results achieved in the last decade, Deep Neural Networks (DNNs) won the challenge against other machine learning tools in tasks of human-level complexity. Convolutional Neural Networks (CNNs), in particular, reached the highest accuracy in several visual computing applications, such as object classification [1], and also proved to be effective and attainable for other data-analytics tasks [2]. Motivated by those findings, a wide class of Internet-of-Things (IoT) services began to use CNNs as tool for making sense of the raw data sampled with in-field sensors. In many real-life applications today, the data-cycle is cloud-centric, namely, data are ceaselessly transmitted to servers hosting CNNs trained to infer decisions that are then sent back to the end-nodes. Despite the ease of implementation, this IT solution shows several weaknesses [3]. First, a centralized infrastructure represents a single-point-of-failure. Second, the communication chain is inefficient due to latency uncertainty and energy consumption. Third, the lack of data locality feeds user's concerns about privacy. A smarter, yet challenging alternative is offered by the emerging edge computing paradigm: CNNs can be pushed closer to the source of data, onto the end-nodes, breaking the dependence from external resources. Unfortunately, the processing cores deployed on the edge are designed for tight power budgets, therefore too small and too slow for complex CNNs with millions of parameters and heavy workloads. With no lack of generality, in this work we tackle the case of edge devices with off-the-shelf microcontroller units (MCUs) running on a few hundred-mW power envelope, like those belonging to the Cortex-M family designed by ARM for the IoT segment[1]. Such cores count on small on-chip RAM ($1 < MB$), an instruction set with few integer options (16- or 8-bit), and limited vector extensions for parallel processing (a 2-lane Single Instruction Multiple Data unit — SIMD — is available for some high-end core of the family). The only option here is to shrink down the size of the CNN model thus to meet the resource constraints. A large body of literature reported several reduction techniques based on algorithmic optimizations that leverage the statistical nature of deep learning. The most of them are static, namely operated at design time through time-invariant knobs. For instance, a common practice is to use compact data representations, like mini-float [4], [5] or fixed-point integers [6], reducing the bit-width for both weights and activations. Several works demonstrated that the quantization of a CNN, from 32-bit floating-point to 8-bit fixed-point, has a negligible effects on accuracy [7], [8], but enables a better usage of the memory bandwidth since multiple data can be packed and retrieved with a single instruction. The choice of the bitwidth is strictly related to the parallelism of the underlying Instruction Set Architecture (ISA). For the MCUs targeted in this work, 8-bit is the standard [9], while uneven bit-widths not supported by native instructions, e.g. 7-, 6-, 5-bit, may introduce an overhead due to extra operations for packing/unpacking data in a regular manner [8].

An extreme case of quantization is the so called binarization, where operands, weights and/or activations, are reduced to a single bit, leading to Binary Neural Networks (BNNs) [10], [11]. This setting reaches the smallest memory footprint and enables the replacement of integer multiply-and-accumulate (MACs) with simple Boolean operators, i.e. bit-wise XNOR and pop-count, ensuring more parallelism and high energy efficiency. Despite the benefits on these extra-functional metrics, BNNs are known to suffer large accuracy drop

Luca Mocerino and Andrea Calimera are with the Department of Control and Computer Engineering, Politecnico di Torino, Italy, 10129 (e-mail: andrea.calimera@polito.it)

[1]https://os.mbed.com/platforms

compared to 8-bit fixed-point counterparts, from 2%, up to 10% and 20%, and even more depending on the CNN topology and the training dataset [10], [11], [12].

Due to their different training methodology, quantization to 8-bit and binarization have been often applied separately. However, they have complementary features that can be combined for the sake of efficiency, for instance, by means of dynamic mechanisms that exploit the functional redundancy brought by a time-scheduled cooperation of multiple predictors. Seminal works in this area, e.g. [13], introduced the concept of *conditional* flows, where a simpler/weaker net (e.g. trained with less parameters and less filters) is aided by a more complex/expressive net only when its prediction is considered unreliable, namely below a certain *confidence* threshold. A more recent proposal described in [14] and referred as CoopNet enabled for the first time a conditional fixed-point/binary flow for CNNs on general-purpose cores. However, CoopNet, just like its predecessors, shows an intrinsic limitation: the strong and weak nets deployed for inference still work as separate entities, unaware of each other. This aspect is a major source of overhead this work aims to address.

Elaborating on the CoopNet idea, we hereby introduce *Boosted Cooperative Convolutional Neural Network* (BC-Net), a novel software-tunable CNN template based on a boosted reuse mechanism that enables highly efficient conditional inference with 8-bit and binary CNNs on low-power MCUs. As main technical contributions, the paper introduces:

- a two-step training procedure for the inter-operability of 8-bit and binary models within the same architecture;
- a framework for fast deployment on MCUs that integrates an off-line tool for centering the BC-Nets on a desired accuracy/latency operating point.

Pre-trained CNNs mapped onto the BC-Net template reach better performance, both in terms of accuracy and latency. The experiments conducted on two commercial boards provided by STMicroelectronics and powered with the ARM Cortex-M7 reveal that for three applications, i.e. image classification (IC) on two data sets (CIFAR-10 and CIFAR-100), keyword spotting (KW) and facial emotion recognition (FER), BC-Nets achieve a speed-up of 81.49% and a gain of accuracy up to 3.8% w.r.t the 8-bit model. Moreover, BC-Net improves over the earlier CoopNet concept with a best-case speed-up of 19% and 3.45% more accuracy.

II. Background

A. Convolutional Neural Networks (CNNs)

CNNs are a class of deep learning inference models suited for multi-dimensional inputs, like images and spectrograms. The input pattern is processed by a chain of layers designed to extract and classify the hierarchical features learned from annotated samples during the training stages. Fig. 1-a graphically depicts the common structure of a feed-forward CNN. The first cell in the chain is in charge of the *Features Extraction*, then the *Classification* cell separates the features previously extracted delivering the actual prediction in the form of output probability over the available classes C. Within the first cell, there is a number of sequenced layers, the most of which

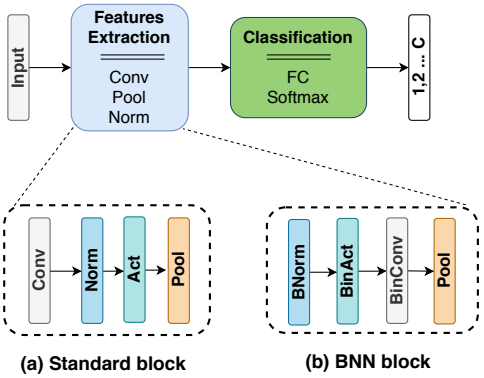

Figure 1: Architecture of a CNN and a typical organization of the hidden layers, full-precision (a) and binarized (b)

are convolutional layers (Conv). A Conv layer implements the multi-dimensional convolution between the feature maps generated by the previous layer and its own inner filters. The Conv layers are often interleaved by other kinds of operators, whose number and order may differ. Among the most used, there are Normalization layers (Norm), which normalize the intermediate features with mean and standard deviation enabling layer-independent training, or dropout layers (Drop), which temporally remove some units in order to avoid overfitting during training. The activation layers (Act) introduce non-linearity by means of point-wise functions, like *Rectified Linear Unit* (ReLU) or *hyperbolic tangent* (tanh), whereas the pooling layers (Pool) operate the re-scaling of the intermediate map via max or average down-sampling. The classification cell is typically made by fully connected layers (FC), often one or two, that implement the linear separation in the feature space and produce the logits. The latter are finally processed through a *Softmax* operator that calculates the probability that the given input does belong to a certain class.

B. Quantization

While the training stage of a CNN model is usually run with single-precision floating-point arithmetic (FP32), several works [6], [15] demonstrated that an inference stage using integer representations with a reduced bit-width (FX) can retain the same accuracy of FP32.

As stated in the introduction, fixed-point quantization is a *must do* stage to deploy CNNs into low-power MCUs. Squeezing the parameters to a lower parallelism ensures a theoretic reduction of the memory footprint proportional to the number of dropped bits. Even more important, it enables an efficient use of parallel instructions for reading/writing and processing the convolutional layers. Considering general-purpose cores, a reduction below the 8-bit mark, e.g. in the range $[2, 7]$, not only introduces additional quality loss [7], but also complicates the resource management due to irregular memory layouts. An efficient management of sub-types claims for custom operators and memory architecture indeed, which are not available on low-end cores due to area/power constraints. For such reason, 8-bit is taken as standard.

Many previous works presented several quantization schemes. Even if a detailed review is out of the scope of this work, we provide a compact taxonomy to frame our solution. A *uniform* FP32-to-FX conversion makes use of a fixed quantization step for all the parameters [16]. The quantization range is said to be *symmetric* or *asymmetric* if centered or shifted with respect to zero. On the other hand, *non-uniform* schemes split the range into uneven quantization intervals using a non-linear function (e.g logarithm) [17] or a hash function [18]. As a rule of thumb, non-uniform and uniform/asymmetric schemes achieve higher accuracy as they ensure a better fitting of the original data, however they call for heavy run-time software routines or custom hardware [17]. Instead, uniform/symmetric schemes are simpler as they can be implemented using add-shift integer operators; for this reason, they are more suited for resource-bounded cores with a limited instruction set. To notice that all these quantization schemes can be applied at different granularity, e.g. on the whole network, layer-wise, or filter-by-filter; the finer the granularity, the higher the quality of result, yet with performance overhead.

In this work we resorted to the q-bit ($q = 8$) fixed-point quantization proposed in [6], which is a uniform/symmetric scheme with a power-of-two per-layer radix-point scaling. This choice is compliant with the optimized kernels included in CMSIS-NN [9] library developed by ARM and used for code porting. From an arithmetic viewpoint, the quantization function $\sigma(p)$, with p as the floating-point value to be quantized, is formally defined as follows:

$$\sigma(p) = \text{clip}_{[-1,1)}\left(\frac{\text{round}(p \cdot 2^{q-1})}{2^{q-1}}\right) \tag{1}$$

where $clip_{[x,y)}(p) = max(x, min(y, p))$.

The resulting convolution between the input feature map $\mathbf{x} \in \mathbb{R}^{ch \times w_{in} \times h_{in}}$ and the local weights $\mathbf{w} \in \mathbb{R}^{ch \times kw \times kh}$ is as follows:

$$\mathbf{x} * \mathbf{w} = 2^{-2(q-1)} \sum_{i \in Ch} X_i \cdot W_i \tag{2}$$

with Ch as the number of channels, W and X the quantized version of the full precision weights and activations.

C. Binarized Neural Networks

Several works proposed extreme quantization to 1-bit for weights and/or feature maps. Courbariaux et. al presented a CNN with binary weights -1 and 1, without altering the precision of the feature maps which are kept to full precision [19]. Later, the same authors extended the binarization to feature maps adopting the *sign* as activation function [10]; this is the first example of Binarized Neural Network (BNNs) where all the floating-point operators are replaced by Boolean XNOR and pop-count (i.e. 1's counting). Such aggressive approximation comes at the cost of severe accuracy loss ($\sim 29\%$ on ImageNet [20]). Nonetheless, the request for high processing efficiency and compact memory footprint motivated the research of new training algorithms exploiting mixed representations. Among the most effective approaches, there are those with extended feature maps representation (≥ 2 bit) and/or full-precision scaling factors for weights and/or feature maps. Zhou et al. [21] proposed a generalization enabling multi-bits representation for weights, feature maps and gradient propagation. Finally, the XNOR-Net shown in [11] achieved the state-of-the-art introducing a full-precision scaling factor for weights and activations. In the same paper, the authors describes a normalization layer to regularize the activations and stabilize the training. These features contributed to enhancing the representational capability of the earliest binary nets: 16.3% and 21.4% more accuracy w.r.t [19] and [10] on the ImageNet dataset.

D. BNN processing

In this work we borrowed from [11] the binary XNOR-Net template (BNN hereafter). Fig. 1-b depicts its basic structure, where the suffix *Bin* denotes the binarized layers. A comparison to classical fixed-point CNNs (Fig 1-a) highlights the layer re-ordering adopted in BNN. The feature maps normalization (BNorm) done before *binarization* (BinAct) and convolution (BinConv) has proven to have beneficial effects for the accuracy. As a further optimization, the normalization and binarization layers can be fused for computational efficiency.

From a mathematical viewpoint, a generic feature map $\mathbf{x} \in \mathbb{R}^{ch \times w_{in} \times h_{in}}$ is binarized to $\mathbf{X}$ as follows:

$$\mathbf{X} = BinAct_{0,1}(x) = \begin{cases} 1 \, x \geq c \\ 0 \, x < c \end{cases} \tag{3}$$

where the parameter $c = \mu - \beta/\gamma\sqrt{\sigma^2 + \epsilon}$ includes all batch normalization parameters: variance σ^2, mean μ, scale γ, shift factor β, and ϵ for numerical stability. Moreover, c is constant at inference time and is represented with 16-bit in our implementation. The weights $\mathbf{w} \in \mathbb{R}^{ch \times kw \times kh}$ are binarized off-line with the *sign* function ($\mathbf{W}$ = sign(x)). Given the input feature $\mathbf{X}$ and the weights $\mathbf{W}$ in binary form, their convolution is approximated as follows:

$$\mathbf{x} * \mathbf{w} \approx \text{popcount}\,(\mathbf{X} \text{ xnor } \mathbf{W}) \cdot \alpha \tag{4}$$

where $\alpha = \frac{1}{n}\|\mathbf{w}\|_{l1}$ is the scaling factor for weights represented with 16-bit. The original paper [11] introduced an additional re-scaling factor K for the feature maps which is computed on-line as the average value for each feature map channel, namely, adding a special *average pooling* layer. However, we empirically found that such layer can be dropped achieving substantial speed-up during inference and training at the cost of minimal accuracy loss ($< 0.5\%$).

From a hardware perspective, the processing of BNNs may take advantage from custom accelerators, e.g. [22], [23], that implement special functions, such as parallel XNOR, pop-count and fast conversions for storage and mixed-precision arithmetic. The authors of [24] demonstrated that the performance gap between hardware accelerated and pure software implementations is huge. A custom FPGA-based inference engine gets $8.5\times$ faster and $20\times$ more energy efficient than a general-purpose Intel core. However, these accelerators are currently not available for off-the-shelf, low-power general-purpose cores, which, instead, can only rely on specialized software macros. For instance, the pop-count can

be implemented with a sequence of mask-&-shift instructions. It is expected that the next generations of cores will come with an extended ISA to serve this purpose. Unlike CNNs with arbitrary-precision fixed-point, BNNs have the advantage of a regular memory layout, where binary weights can be easily grouped in single word. Therefore, even if their processing on general purpose cores gets slower than it could be on custom accelerators, BNNs get faster than CNNs quantized to 8-bit and are good candidates to implement conditional networks.

III. Related Works

A. Conditional CNNs

Inputs are not created equal. Recognizing a certain object from a clear and pale background is easier than distinguishing it in a crowded scenario. Despite that, classical CNNs are designed to spend the same maximal effort on all inputs with a large resource waste on average. To mitigate this effect, pioneering works like [13] proposed the use of two predictors of increasing complexity that are activated in sequence depending on a *confidence score*, or *score margin*. More in details, a weaker inference model is used for *easy* inputs, and a more complex model for *hard* inputs; the latter is activated when the confidence of the weak model is below a given safety threshold. The main result is that, on average, the strong model works sporadically, with average savings in terms of latency. Similarly, the authors of [25] implemented the same principle using multiple fixed-point nets of increasing arithmetic precision. Handling dynamic bit-width at run-time requires custom hardware, such as scalable-precision MAC units and custom memory operations. On the same line, Amiri et al. in [26] leveraged the heterogeneity of a custom architecture to host models with different arithmetic: a single-precision floating-point model running on a high-performance ARM core and a binary model implemented by means of a custom FPGA accelerator. Besides needing custom hardware units, the technique requires an on-line training phase that introduces a substantial overhead. Among other solutions thought for general-purpose cores, we found a conditional execution mechanism in [27], where a number of inference models packed as an inference ensemble are executed according to a confidence metric. The target here are cloud platforms with countless resources, namely, large memory pools to store the ensemble and lots of computational resources to execute many full-precision models (more than eight as the best case reported in [27]). The shortage of memory space and computing power on the edge devices makes those approaches less appealing. A more recent solution introduced in [14] is tailored to these needs instead. Named CoopNet, from cooperative nets, it promotes a simplified, yet efficient dynamic scheduling between a fixed-point and a binary CNN made possible thanks to a new design template. Such lightweight conditional mechanism, together with a stable training procedure and the low hardware requirements, led CoopNet to outperform prior arts. However, there are still optimization margins for improvement, and the BC-Net proposed in this work is proof.

B. Hybrid Precision

A very recent trend in CNNs quantization is the adoption of *hybrid* solutions where higher precision (from 2- to 8-bit fixed-point or 32-bit floating-point) and binary arithmetic operate within the same network with the aim of reducing the accuracy loss introduced by binarization. Pioneering works like [28] proposed to replace some of the binary layers with single-precision floating-point FP32. More precisely, the most error-resilient layers are taken in a binary format, whereas the remaining ones are kept to full-precision. A finer hybrid quantization mechanism is also described in [29], where authors introduced the intra-layer, filter-level hybrid precision, enabling the coexistence of binary and high precision weights with the same layer. The idea was extended to more sophisticated CNNs (e.g. ResNet) and also combined with other compression strategies, like the width-multiplier expansion. Even further, [30] elaborated on the optimal per-layer precision assignment problem, introducing a more sophisticated dynamic activation functions. To notice that all these methods are conceived as static, time-invariant techniques not able to adapt to the complexity of data. Moreover, they offer a practical solution to compress the model size, but can reach speed-up only when ported to custom hardware units. In fact, for the same reasons discussed in the previous sections, arbitrary bitwidth scaling get slower than 8-bit [31], [32]. It is not a coincidence that most of the those prior works make use of other proxies (e.g. FLOPs) to assess performance [28], [29].

IV. BC-Net

A. Concept and Architecture

We introduce BC-Net using as reference the cooperative template proposed in [14]. The latter is built starting from a 8-bit (INT8 for short) and a binary model (BNN) obtained from the original floating-point (FP32) via fixed-point quantization and binarization respectively. INT8 is more accurate but large and slow, BNN is faster and compact but less accurate. The input is processed by the BNN at first, then, if the result satisfies a certain *confidence score* it is forwarded to the output, otherwise, the prediction is discarded and the INT8 model is called in order to improve the quality of the result. In the worst case, the same input is processed twice, which means the resources consumed by BNN are wasted without making any contribution. In other words, the BNN inference represents a computational overhead as the valid prediction is taken from INT8.

Elaborating on this key aspect, BC-Net implements a more efficient management of the BNN model promoting an information recycling mechanism that gives a *boost* to the final accuracy. Fig. 2 provides an abstract view of the BC-Net architecture. It consists of the BNN and the INT8 nets controlled by the confidence score and interfaced with a few glue layers. There are two operating modes: the *Light Mode* (LM) when the BNN prediction is evaluated as reliable and therefore used as main output; the *Full Mode* (FM) when the INT8 is activated and the resulting prediction is smartly obtained by taking into account the result of BNN previously computed. The integration is obtained concatenating the logits of BNN and INT8 into a single tensor which is then fed as input to a fully-connected

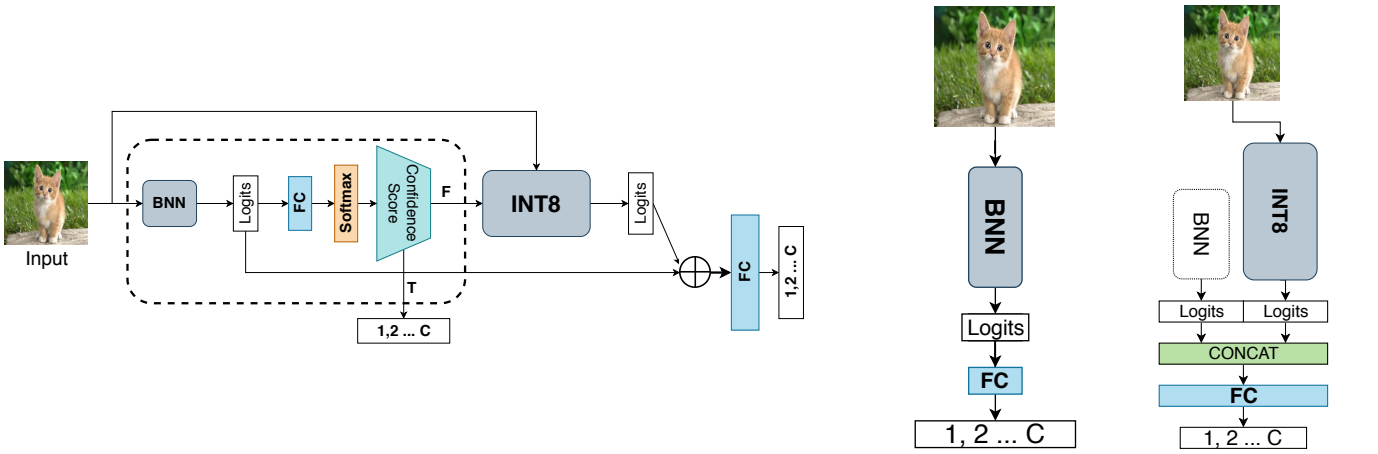

Figure 2: BC-Net Architecture (left). On the right two processing cases: Light Mode (**LM**) *(left)*, Full Mode (**FM**) *(right)*

layer in charge of the final classification over the C classes. Intuitively, the information extracted by BNN does not get lost as in CoopNet, but served to raise the accuracy of INT8.

The criterion adopted to balance the computing effort is accuracy driven, namely, based on the uncertainty of the BNN prediction. The recent literature counts a plenty of methods to evaluate such metric, the most of which rely on Monte Carlo sampling [33] and Bayesian methods [34] that however require multiple inference executions, drastic topological changes and/or unconventional training mechanisms [35]. These characteristics do contrast with the requirements of low-power applications. This motivates our choice of a simpler, yet reliable proxy signal, which is the *Confidence Score* (CS) defined as follows:

$$CS = P_{BNN}(y_i|x) - P_{BNN}(y_j|x) \tag{5}$$

$P_{BNN}(y_c|x)$ is the conditional probability estimated by the *softmax* layer of the BNN that a given input x belongs to a given class $c \in \{1, 2 \dots C\}$; i and j refer to the indexes of the first and second highest scored classes. A high *CS* means the BNN was able to classify the given input with enough confidence, whereas a low *CS* means that the two topmost scored classes get very close to each other, which reveals high uncertainty. In this latter case, the BC-Net switches from LM to FM mode.

A key variable to consider for a safe control policy is the *Confidence Threshold (CT)*, that is the value below which the FM mode is activated. To notice that the CT can be used as a dynamic knob to reach different accuracy-latency trade-off.

B. Training

Training a BC-Net model requires a two-step procedure: (i) the training of the BNN model, (ii) the training of the INT8 model in FM mode, namely, considering the BNN results as part of the input domain. During the first step, the BNN is trained stand-alone using a standard binarization method. In this work we opted for that presented in [11] as introduced in Section II-C. To be noted that the first and last layers of the BNN model are kept full-precision, i.e. 8-bit, as also suggested by references in the literature [19], [10], [11]. Concerning the second step, the BNN weights previously learned at step one are frozen, while the remaining parts of the BC-Net architecture, namely the INT8 model and the glue layers, are first trained using a standard back-propagation algorithm with floating-point arithmetic, and then projected in the integer domain (8-bit) via quantization; for such purpose we used the method proposed in [6] as introduced in Section II-B.

C. Confidence Threshold Estimation

Once BC-Net is trained, a key aspect is the setting of the threshold CT as it affects the accuracy-latency trade-off: the higher the CT, the higher the activation frequency of the FM mode, which in turn leads to higher accuracy but slower inference. Providing users with a simulation-free model for both accuracy and latency as function of the CT is of paramount importance to accelerate the deployment of BC-Nets, and thus to reach the desired trade-off according to the design specifications. To this end, we developed a characterization framework, named *Static Threshold Estimation* (STE) tool and shown in Fig. 3. Before proceeding with the description of the tool, it is important to specify the formalism we adopted to distinguish the dataset used for the training of the BC-Net and that used to train/test the two regressors: the prefix *Model (M)* is for BC-Net model, while *Regression (R)* is for the regressors.

The STE pipeline takes as input the BC-Net and the model test set (MTS) which is made up of samples not used during the training stages of BC-Net. This choice guarantees an unbiased evaluation of BC-Net, hence a better accuracy of the regressors[2]. As main outcomes, the STE produces two predictors function of the parameter CT: (i) the accuracy of BC-Net, (ii) the activation frequency of the FM model (N_{FM}), as a good proxy for latency. From such two regressors, it is possible to get a hardware-agnostic estimation of both functional and extra-functional metrics and hence to calibrate off-line the BC-Net.

Internally, the pipeline is organized as follows. The MTS is split into a model calibration set (MCS) (10%) and a model validation set (MVS) (90%) with samples equally distributed over the available classes. Then, the BC-Net is run over the MCS

[2]An alternative approach would consist of collecting and labeling samples directly from the final application and then using them to train off-line the two regressors.

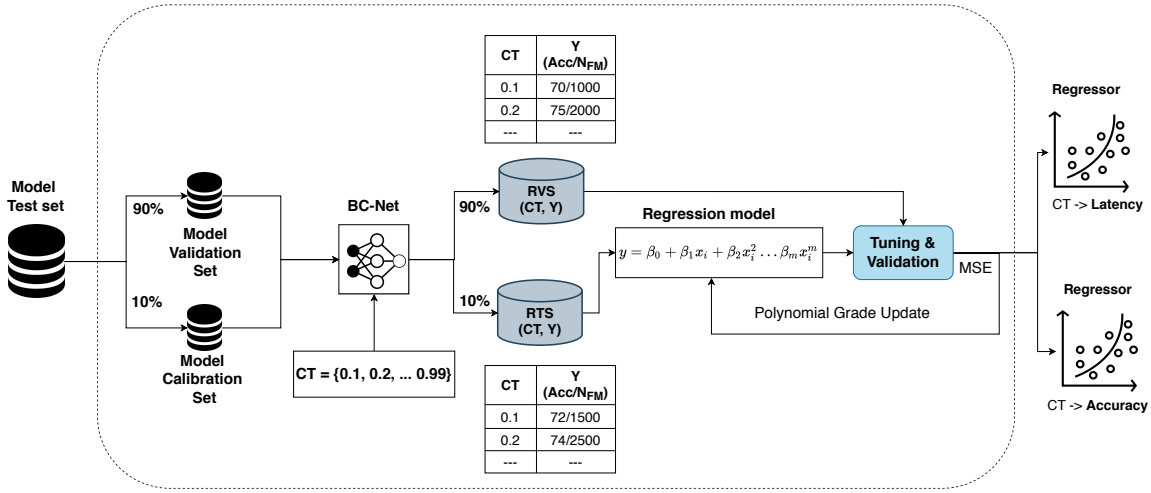

Figure 3: Static Threshold Estimation (STE) Pipeline

for different value of $CT \in [0, 1[$; we empirically found that 100 different CT values guarantee a favorable prediction accuracy avoiding overfitting. The resulting performance metrics (i.e. accuracy and number of times FM is activated) are collected and stored in the vector Y. The union between CT and Y results into the data-set to fit and validate the regression models, in particular, the regression training set (RTS) (10%) and the regression validation set (RVS) (90%). We opted for a *polynomial regressor* trained using as objective function the squared error E as reported in the following equation:

$$E = \sum_{i=0}^{D} |p(CT_i) - y_i|^2 \tag{6}$$

where $p(CT_i)$ is the i-th prediction and y_i the true label (Accuray/N_{FM}). The grade of the polynomials is increased progressively, until the mean square error (MSE) gets below a user defined threshold ϵ. The two predictors are made available as main output, providing an effective tool for assessing the control policy of the BC-Net.

D. Modeling of Extra-Functional Metrics

This section introduces the models for the assessment of the extra-functional properties of BC-Net, latency and memory footprint. Although our experiments have been collected on board, a description of the different contributions that impact the figures-of-merit can help the overall understanding, also providing further analysis tools for the end-users.

1) **Inference Latency***:* Given a generic BC-Net, its latency is modeled through the following equation:

$$L_{BCNet}(CT) = \begin{cases} L_{LM} + L_{CS} & CS \geq CT \\ L_{FM} + L_{CS} & CS < CT \end{cases} \tag{7}$$

$L_{LM} = L_{BNN}$ is the latency in LM mode, under which only the BNN is run; $L_{FM} = L_{BNN} + L_{INT8} + L_{FC}$ is the latency in LF mode, with L_{INT8} the latency spent by INT8 and L_{FC} the latency of the glue layers (i.e. the CONCAT+FC as can be inferred by Fig.2); L_{CS} is the contribution due to the evaluation of the confidence score CS and the comparison with CT. To notice that L_{CS} is the time consumed for a single integer subtraction and comparison, hence negligible. The two terms L_{LM} and L_{FM} can be retrieved from on-board characterization as they are affected by the type of the processing core. For batch inference, Equation 7 can be generalized as follows:

$$L_{BCNet}(BS, CT) = \sum_{i=1}^{BS} L_{BCNet_i}(CT) \tag{8}$$

where *BS* is the cardinality of the batch and $L_i(CT)$ is the latency of the i-th batch sample.

2) **On-chip RAM Footprint***:* The processing cores targeted by this work are those of the Cortex-M family by ARM. Therefore, the memory footprint can be computed according to the memory layout implemented with the neural kernels of the CMSIS-NN library [9]. More in details, the overall RAM occupied by a feed-forward CNN is given by the sum of three main contributions: (i) buffer to store the model parameters, i.e. the weights of all the layers, (ii) buffer for intermediate input and output activations, (iii) buffer required for the *im2col* routine. For BC-Net, the total memory space is the sum of the RAM taken by the BNN model, that for the INT8 model, and that of the fully-connected layer. The CT parameter is one Byte, hence negligible. It is worth noticing that both binary and 8-bit fixed-point arithmetic are hosted on a single board with static memory allocation. During inference they are block-loaded in the RAM and made ready to use, therefore there is no run-time overhead for task/context switching.

V. Experimental Results

A. Benchmarks and Datasets

We tested and evaluated the performance of BC-Nets on three classification tasks; Table I provides a complete overview, with details about the topology of the CNNs deployed for each data-set.

Task	IC				KS		FER	
Dataset	CIFAR-10/100 [36]				GSC [37]		FER2013 [38]	
Model	CaffeNet		VGG-6		GscNet		FerNet	
Input	$3 \times 32 \times 32$				$1 \times 32 \times 32$		$1 \times 44 \times 44$	
Model Topology	Conv	$32 \times 5 \times 5$	Conv	$32 \times 3 \times 3$	Conv	$32 \times 5 \times 5$	Conv	$32 \times 3 \times 3$
	MaxPool	$(3, 3, 2)$	Conv	$32 \times 3 \times 3$	MaxPool	$(3, 3, 2)$	Conv	$32 \times 3 \times 3$
	Conv	$32 \times 5 \times 5$	MaxPool	$(2, 2, 2)$	Conv	$32 \times 5 \times 5$	Conv	$32 \times 3 \times 3$
	MaxPool	$(3, 3, 2)$	Conv	$64 \times 3 \times 3$	MaxPool	$(3, 3, 2)$	MaxPool	$(2, 2, 2)$
	Conv	$64 \times 5 \times 5$	Conv	$64 \times 3 \times 3$	Conv	$64 \times 5 \times 5$	Conv	$64 \times 3 \times 3$
	MaxPool	$(3, 3, 2)$	MaxPool	$(2, 2, 2)$	Conv	$64 \times 5 \times 5$	Conv	$64 \times 3 \times 3$
	FC	1024×10	Conv	$128 \times 3 \times 3$	MaxPool	$(3, 3, 2)$	Conv	$64 \times 3 \times 3$
			Conv	$128 \times 3 \times 3$	FC	1024×31	MaxPool	$(2, 2, 2)$
			MaxPool	$(2, 2, 2)$			Conv	$128 \times 3 \times 3$
			FC	512×100			Conv	$128 \times 3 \times 3$
							Conv	$128 \times 3 \times 3$
							MaxPool	$(2, 2, 2)$
							FC	128×7

Table I: Benchmark overview. Convolutional layers with shape $(ch_o \times kh \times kw)$, fully-connected layers with shape $(ch_i \times ch_o)$ and pooling layers with shape (kh, kw, s); kh and kw are the kernel height and width and s the stride, while ch_i and ch_o refer respectively to the number of input and output channels.

1) Image Classification (IC): the classical image recognition task on the popular CIFAR dataset [36]. We evaluated the two versions available, i.e. ten classes (CIFAR-10) and hundred classes (CIFAR-100). In both cases, there are overall 60k 32×32 RBG images with labels representing animals (e.g. cat, dog) and vehicles (car, truck, etc.). The CNN benchmark adopted for the 10-class problem (IC-10) is that belonging to the open-source framework Caffe [39], whereas the CNN for the 100-class problem (IC-100) is a modified version of the popular VGG net (VGG-6) [40].

2) Keyword Spotting (KS): a speech recognition task aimed at detecting keywords or commands. The dataset, named Google Speech Commands (GSC), comes for Google research [3]. It consists of 65k one-second-long audio samples collected during the repetition of 30 different words by thousands of different people. The goal is to recognize 30 specific keywords, i.e. "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", "Go", out of the 30 available words. Samples that do not belong to the 30 categories are labeled as "unknown". The training set and the test set collect 56196 and 7518 items respectively, which are stored in the form of audio spectrograms of the recorded speech. As the CNN benchmark (GSCNet) we deployed a modified version of the neural net presented in [37].

3) Facial Expression Recognition (FER): another visual reasoning task where the purpose is to recognize different emotions from facial images. The Fer2013 dataset we used for training and testing was first introduced during a Kaggle competition [38]. It contains 48×48 samples: 28709 for training, 7178 for testing. The CNN benchmark (FerNet) is inspired by an open-source implementation [4].

The selection of the above benchmark suite was done considering the amount of resources available on the targeted MCUs. Although BC-Net can even work on more complex scenarios (e.g. CNNs to classify high definition images over thousands of classes like ImageNet), those are out of reach for the mobile segment considered in this work and perhaps not even useful, or of interest, when considering edge devices with tiny cameras and small on-chip memory.

B. Experimental/Hardware set-up

The objective is to provide a comparison between BC-Net and (i) standard fixed-point quantization and binarization, and (ii) a state-of-the-art cooperative mechanism. All the CNN benchmarks have been quantized and binarized with the methods presented in Section II-B and II-C respectively, while for BC-Net we used the training flow introduced in Section IV-B. For what concerns BC-Net, the additional layers implementing the cooperative mechanism were trained through the Adam optimizer [41] with an adaptive learning rate (lr) scheduler set as follows: lr updated with step 0.1 every 15 consecutive epochs in which the validation loss does not change. No form of data augmentation was applied on the original dataset.

The trained models are parsed and translated to `.C` by means of header files `.h` containing weights in a format compliant with the CMSIS-NN kernel library by ARM [9]. More precisely, the inference engine deployed on board is built with an extended version of the CMSIS-NN that gives support for binary convolutions [31].

As evaluation platforms we adopted two off-the-shelf boards provided by STMicroelectronics and powered with the Cortex-M7 core by ARM: the NUCLEO-F767ZI[5], with 512kB of RAM, 2MB of Flash memory and 216MHz of clock frequency; the STM32H743[6], with 1MB of RAM, 2MB of Flash memory, and 480MHz of clock frequency. For a fair performance assessment, the frequency of this second board is rescaled to 216MHz.

C. Static Analysis: Accuracy, Memory and Latency

Table II summarizes the functional and non-functional properties collected on-board: Top-1 accuracy (%), RAM footprint (kB), and inference latency (ms). For each benchmark, we thus provide a comparison among the available implementations: the

[3]https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html

[4]https://github.com/JostineHo/mememoji

[5]https://www.st.com/en/evaluation-tools/nucleo-f767zi.html

[6]https://www.st.com/en/microcontrollers-microprocessors/stm32h743-753.html

Benchmark	Precision		Top-1 Acc. (%)	Mem. Size (kB)	Latency (ms)
IC-10	FP32		80.25	523	-
	INT8		80.20	131	98.96
	BNN		76.52	94	65.29
	BC-Net	LM	76.52	94	65.29
		FM	84.00	235	164.26
IC-100	FP32		65.13	1583	-
	INT8		64.68	396	147.39
	BNN		54.06	123	54.46
	BC-Net	LM	54.06	123	54.46
		FM	66.48	569	202,35
KS	FP32		90.30	1060	-
	INT8		89.50	265	66.30
	BNN		87.60	90	32.56
	BC-Net	LM	87.60	90	32.56
		FM	92.55	386	99.06
FER	FP32		65.16	2345	-
	INT8		64.70	587	353.93
	BNN		62.86	118	174.33
	BC-Net	LM	62.86	118	174.33
		FM	66.48	705	528.27

Table II: Main Functional and Extra-Functional proprieties for each benchmark using different implementation options (BC-Net in static mode, i.e. thresholding off).

original 32-bit floating-point (FP32), the 8-bit fixed point model (INT8), the binary weight model (BNN), and our proposal (BC-Net). For BC-Net, the two operating modes, i.e. light-mode (LM) and full-mode (FM), are taken separately, namely, they are compiled disabling the dynamic control mechanism, but still leaving the glue layers for overhead assessment. The latency for FP32 is not reported due to the lack of a support for that format in the adopted neural library CMSIS-NN. One should also consider that such floating-point models are too large to fit the available memory, especially if other applications are running in background.

A first consideration concerns the effect of quantization and binarization as stand-alone techniques. In line with previous works, it emerged that while the accuracy drop from the FP32 to INT8 model is almost negligible, the binarization process introduces a substantial accuracy degradation, which is quite evident in a complex task as the IC-100 (-11.07% w.r.t FP32). Looking at BC-Net, the LM mode shows the same quality of BNN, which is intuitive, given the architecture of our template. More interestingly, the FM mode gives a substantial accuracy boost. It outperforms INT8 (+3.80% for IC-10, +1.8% for IC-100, +3.05% for KS, +1.78% for FER), and, even more surprisingly, it gets more accurate than the original FP32 model (+3.75% for IC-10, 1.35% for IC-100, +2.25% for KS, +1.32% for FER). Such results are even more impressive if compared to binary net. For instance, looking at the IC-100 task, BC-Net improves on BNN by +12.42%. This is mainly due to the joint training procedure adopted for the FM mode, by which the INT8 is enriched with the prediction outcome of the BNN. As a result, the learned features are more rich and hence able to bring more expressive power and better classification to the resulting networks. In other words, while a post-training quantization to 8-bit cannot go beyond the original model used as seed, BC-Net leverages a larger feature space reaching better generalization.

For the other extra-functional metrics, INT8 achieves a linear memory reduction w.r.t FP32 ($4\times$ as expected), while BNN goes much more far with an impressive compression ratio (from

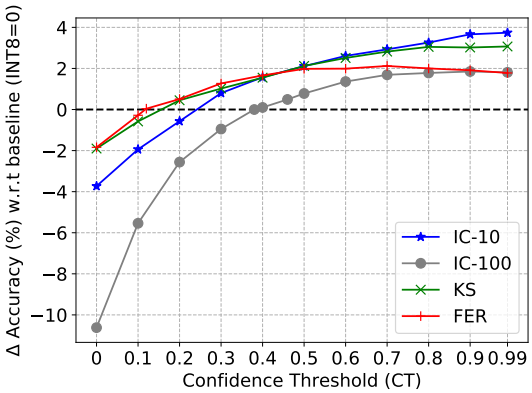

Figure 4: Δ Accuracy (%) w.r.t baseline model (INT8) vs Confidence Threshold

$5.56\times$ for IC-10 to $19.87\times$ for FER) which in turn ensures a substantial speed-up w.r.t. INT8 (from $1.52\times$ for IC-10 to $2.7\times$ for IC-100). As a side note, the latency reduction reached by BNN is bounded by the presence of the first and last layers that are kept to 8-bit. BC-Net inherits the same static behavior of its baseline models. Under the LM mode, it is just like BNN, both in terms of memory footprint and latency. In FM mode, the total memory is the sum of INT8, BNN, and the glue layers, the latter with an impact of less than 10% (4.4% for IC-10, 9.6% for IC-100, 8.7% for KW, 0.13% for FER). Similarly, the overall latency is the time taken by BNN and INT8 to run in sequence, plus the contribution due to the processing of the glue layers, which is negligible (from 0.05 ms for FER to 0.5 ms for IC-100). Memory and latency overhead becomes even more marginal when considering the savings brought by the dynamic control mechanism discussed in the next sub-section.

D. Dynamic Analysis: Accuracy-Latency Trade-off

The objective here is to assess the control mechanism governing the dynamics of BC-Nets. To this aim, we present a set of parametric analyses using the confidence threshold CT as the main control variable.

The first plot in Fig. 4 shows the accuracy gain reached by the BC-Nets using the INT8 model as ground. As a general rule, the BC-Net gets more accurate ($\Delta > 0$) as CT increases. In fact, to ask for a higher confidence score implies a more frequent activation of the FM mode, which has been proven to be more accurate as reported in the static analysis. Looking at the peak accuracy ($CT = 99\%$), BC-Net improves INT8 by far: 3.80% for IC-10, 2.00% for IC-100, 3.16% for KS and 2.12% for FER. The break-even point CT_{be} (i.e. CT s.t.$\Delta = 0$) may shift depending on the complexity of the dataset and the predictive ability of the CNN adopted. For our CNN benchmarks we found the following: $CT_{be} = 0.24$ for IC-10, $CT_{be} = 0.38$ for IC-100, $CT_{be} = 0.16$ for KS, $CT_{be} = 0.12$ for FER.

A more interesting study focuses on the existing relationship between accuracy and latency. The line plots in Fig. 6 show the boost of accuracy brought by BC-Net as function of the speed-up measured on-board. As in the previous analysis, INT8 is taken as the reference baseline. To notice that the speed-up is an indirect, yet more meaningful metric controlled by the CT

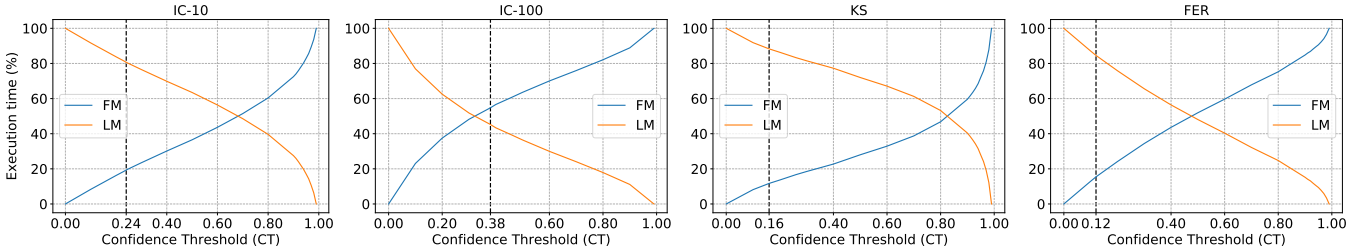

Figure 5: The fraction of Execution Time (%) spent in light mode (LM) and full mode (FM) during the inference stage, varying the CT, for all four benchmarks (IC-10/100, KS, FER). The dot line represents $CT = CT_{be}$.

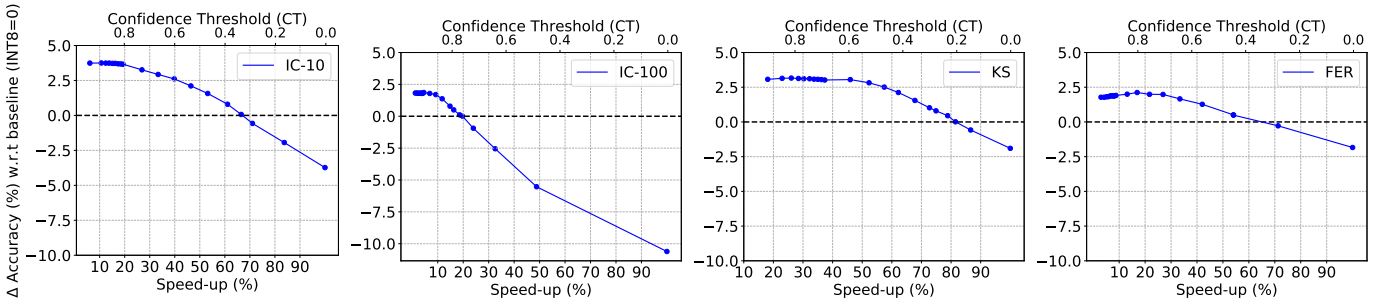

Figure 6: Δ accuracy (%) w.r.t. baseline model vs runtime average speed-up (%) for the four benchmarks (IC-10/100, KS, FER).

knob (that is shown as the second x-axis on the top). A low CT (right side of the plots) accelerates the inference, letting the LM mode be considered highly reliable even if the prediction of the binary net might be far from truth. This emerges from Fig. 5 which shows the impact of CT on the execution time spent in LM and FM. Under this condition, BC-Net can be made run at a much higher speed compared to classical INT8 nets, still keeping reasonable accuracy levels; only IC-100 suffers a sudden drop. On the opposite corner (left side of the plots), with a high CT, the speed-up decreases and accuracy gets back to higher scores. Interestingly, at the break-even ($\Delta = 0$) BC-Net ensures the same predictive quality of INT8 with a remarkable speed-up: 67.71% for IC-10, 19.70% for IC-100, 81.49% for KS, 66.95% for FER. It is worth noticing that the dynamic mechanism at the basis of BC-Net allows to reach the same accuracy of the INT8 reference, yet saving the majority of calls to the slower networks. For instance, at the break-even, for IC-10 FM is activated just 19.36% of the time, 54.75% for IC-100, 11.7% for KS and 15.43% on FER. This is the key feature that makes the proposed BC-Net suitable for resource constrained applications: same quality in less time, hence higher energy efficiency.

E. BC-Net vs. CoopNet

In this section, we reported a comparative analysis between BC-Net and CoopNet [14]. The analysis does focus on the dynamic behavior of the two nets in terms of accuracy-latency trade-off.
Table III shows the speed-up achieved by CoopNet and BC-Net when they are set to work under the same level of accuracy (column *Accuracy Level*). As test case we considered the accuracy achieved by the full precision model FP32 and that of the 8-bit fixed-point model INT8. The percentage of speed-up (BC-Net vs. CoopNet) is reported in brackets. For instance, on the IC-10 task, BC-Net is 15.33% and 16.13% faster than CoopNet. The speed-up gets even higher for FER, 17.99% and 19.05%, while for IC-100 it is worth noticing that CoopNet was not able to reach the FP32 accuracy (marked with a dash).
A further and more detailed analysis reported in Fig. 7 reveals once again the intrinsic efficiency of BC-Net. Indeed, the available optimization slack of BC-Net can be consumed to improve the accuracy prediction rather than to speed up the inference, and that can be achieved by just setting the CT to the proper value. In particular, the barcharts show BC-Net (blue bars) gets more accurate than CoopNet (gray bars) under the same speed constraint. At the best case, i.e. speed-up 10%, BC-Net outperforms CoopNet by far: +3.45% for IC-10, +1.6% for IC-100, +1.4% for FER, +3.04% for KS; for points where CoopNet performs best, BC-Net brings good savings: +3.1% for IC-10 and +1.15% for FER (at 20% speed-up), +1.6% for IC-100 (at 10% speed-up), +1.5% for KS (at 50% speed-up). On IC-100 we observed some negative trend when the speed-up gets greater than 30% (more negative Δ), which is however negligible (0.47% as the worst case). We can thereby state BC-Net is a win-win solution that designers might exploit to meet different constraints of accuracy and/or performance.

F. Validation of the STE Tool

A proper setting for BC-Net encompasses the choice of the confidence threshold CT that guarantees a certain level of accuracy or speed-up. In order to enable this at design-time, we introduced in Section IV-C the *static threshold estimation* (STE) framework, a pipeline to build the predictors for both accuracy and latency depending on CT, simply using the available data. Fig. 8 shows how the models generated by the STE (dashed line) fit some random test (dots) obtained by feeding the BC-Net with unseen data. In the top row of the plots, we reported the accuracy, again in the form of distance (Δ) from INT8, while on the bottom row of the figure there is the activation frequency

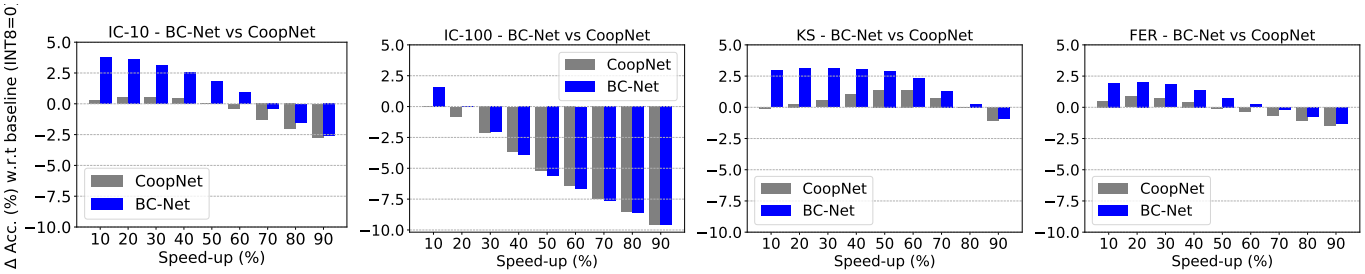

Figure 7: Comparative Analysis. Δ accuracy (%) w.r.t. baseline model vs runtime average speed-up (%) for the three benchmarks (IC, KS, FER). Blue bars refer to BC-Net while grey to CoopNet results.

Application	Model	Accuracy Level	Speed-up (%)
IC-10	CoopNet [14]	FP32	51.20
		INT8	51.58
	BC-Net	FP32	66.53 (**+15.33**)
		INT8	67.71 (**+16.13**)
IC-100	CoopNet [14]	FP32	-
		INT8	11.49
	BC-Net	FP32	16.27
		INT8	19.70 (**+8.21**)
KS	CoopNet [14]	FP32	69.53
		INT8	80.16
	BC-Net	FP32	74.92 (**+5.39**)
		INT8	81.49 (**+1.33**)
FER	CoopNet [14]	FP32	36.20
		INT8	47.90
	BC-Net	FP32	54.09 (**+17.99**)
		INT8	66.95 (**+19.05**)

Table III: Comparative Analysis. Speed-up (%) for CoopNet [14] and BC-Net (our) for two accuracy levels.

of the FM mode (N_{FM}). Over each plot we also annotated the mean square error (MSE) and the maximum error (ME) of the models. The relationship between CT and accuracy is quadratic, with a MSE almost negligible for the four benchmarks, and a maximum error ranging from 0.23 (KS) to 0.43 (IC-100). On the other hand, one can notice a cubic trend for the N_{FM}, with the largest errors for KS: MSE$=$ 0.0009, ME$=$ 0.11. Those findings prove the predictors are highly precise, and hence a valuable tool for setting BC-Nets up. We do not propose any mechanism to find an *optimal* CT because the operating-point in the accuracy-latency space must be defined by the final user, according to application and domain-specific constraints.

VI. Conclusions

Enabling fast and efficient processing of DNN models on ultra-low-power platforms is challenging. BC-Net shows how a dynamic processing mechanism represents a suitable solution when the hardware support is scarce and the resources bounded. In this complex scenario, BC-Net outperforms existing methods in terms of run-time latency and accuracy. The achieved results revealed: (i) up to 81.49% of speed-up and up to 3.8% of accuracy improvement w.r.t. standard fixed-point quantization, (ii) up to 19% of speed-up and up to 3.45% of accuracy improvement w.r.t CoopNet, a state-of-the-art binary-integer cooperative mechanism.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] Z. Jianqiang *et al.*, "Deep convolution neural networks for twitter sentiment analysis," *IEEE Access*, vol. 6, pp. 23 253–23 260, 2018.

[3] W. Shi *et al.*, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[4] X. Sun *et al.*, "Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2019, pp. 4901–4910.

[5] L. Mocerino *et al.*, "Energy-efficient convolutional neural networks via recurrent data reuse," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 848–853.

[6] I. Hubara *et al.*, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, pp. 6869–6898, 2017.

[7] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7873–7882.

[8] M. Grimaldi *et al.*, "Optimality assessment of memory-bounded convnets deployed on resource-constrained risc cores," *IEEE Access*, vol. 7, pp. 152 599–152 611, 2019.

[9] L. Lai *et al.*, "CMSIS-NN: efficient neural network kernels for arm cortex-m cpus," *CoRR*, vol. abs/1801.06601, 2018.

[10] M. Courbariaux *et al.*, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[11] M. a. Rastegari, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proceedings of the European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[12] L. Mocerino and A. Calimera, "Tentaclenet: A pseudo-ensemble template for accurate binary convolutional neural networks," in *Proceedings of the IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2020, pp. 261–265.

[13] E. Park *et al.*, "Big/little deep neural network for ultra low power inference," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct 2015, pp. 124–132.

[14] L. Mocerino and A. Calimera, "Coopnet: Cooperative convolutional neural network for low-power mcus," in *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Nov 2019, pp. 414–417.

[15] D. Lin *et al.*, "Fixed point quantization of deep convolutional networks," in *Proceedings of the International Conference on Machine Learning*, 2016, pp. 2849–2858.

[16] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[17] S. Vogel *et al.*, "Efficient hardware acceleration of cnns using logarithmic data representation with arbitrary log-base," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[18] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[19] M. Courbariaux *et al.*, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proceedings of the Advances in neural information processing systems*, 2015, pp. 3123–3131.

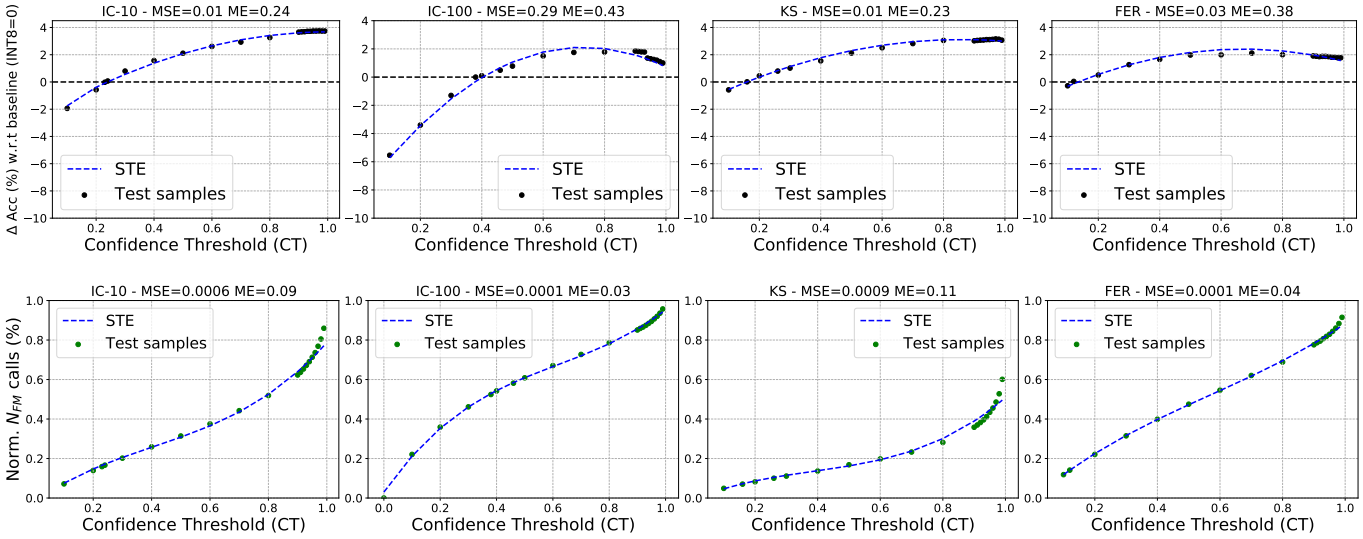

Figure 8: (Top) Δ Accuracy (%) w.r.t baseline predicted via STE vs CT. (Below) The predicted N_{FM} calls normalize w.r.t validation set size varying CT. The dots represent the real measurement on whole validation set (accuracy and the N_{FM} calls)

[20] J. Deng *et al.*, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[21] S. Zhou *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[22] A. Al Bahou *et al.*, "Xnorbin: A 95 top/s/w hardware accelerator for binary convolutional neural networks," in *Proceedings of the IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE, 2018, pp. 1–3.

[23] A. Agrawal *et al.*, "Xcel-ram: Accelerating binary neural networks in high-throughput sram compute arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 8, pp. 3064–3076, 2019.

[24] D. J. Moss *et al.*, "High performance binary neural networks on the xeon+ fpga™ platform," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–4.

[25] D. J. Pagliari *et al.*, "Dynamic bit-width reconfiguration for energy-efficient deep learning hardware," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ser. ISLPED '18. New York, NY, USA: ACM, 2018, pp. 47:1–47:6.

[26] S. Amiri *et al.*, "Multi-precision convolutional neural networks on heterogeneous hardware," in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 419–424.

[27] H. Tann *et al.*, "Flexible deep neural network processing," *arXiv preprint arXiv:1801.07353*, 2018.

[28] A. Prabhu *et al.*, "Hybrid binary networks: optimizing for accuracy, efficiency and memory," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 821–829.

[29] I. Chakraborty *et al.*, "Efficient hybrid network architectures for extremely quantized neural networks enabling intelligence at the edge," *arXiv preprint arXiv:1902.00460*, 2019.

[30] K. Huang *et al.*, "Efficient quantization for neural networks with binary weights and low bitwidth activations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3854–3861.

[31] M. Rusci *et al.*, "Work-in-progress: Quantized nns as the definitive solution for inference on low-power arm mcus?" in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2018, pp. 1–2.

[32] G. Ottavi *et al.*, "A mixed-precision risc-v processor for extreme-edge dnn inference," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 512–517.

[33] B. Lakshminarayanan *et al.*, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Proceedings of the Advances in neural information processing systems*, 2017, pp. 6402–6413.

[34] J. Gast and S. Roth, "Lightweight probabilistic deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3369–3378.

[35] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *Proceedings of the International Conference on Machine Learning*, 2015, pp. 1861–1869.

[36] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[37] T. N. Sainath *et al.*, "Convolutional neural networks for small-footprint keyword spotting," in *Interspeech*, 2015.

[38] I. Goodfellow *et al.*, "Challenges in representation learning: A report on three machine learning contests," 2013.

[39] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678.

[40] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[41] D. P. Kingma *et al.*, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Luca Mocerino received M.Sc. in Computer Science at Politecnico in Turin in 2017. There, since January 2018, he has been a Research Assistant and Ph.D. Candidate in Computer and Control Engineering with the EDA group. His research interests focus mainly on Machine/Deep learning, smart embedded systems and low-power/energy design techniques for digital ICs.

Andrea Calimera took the M.Sc. degree in Electronic Engineering and the Ph.D. degree in Computer Engineering both from Politecnico di Torino. He is currently an Associate Professor of Computer Engineering at Politecnico di Torino. His research interests cover the areas of design automation of digital circuits and embedded systems with emphasis on optimization techniques for low-power and reliability, energy/quality management, logic synthesis, design flows for emerging computing paradigms.