Channel-wise Mixed-precision Assignment for DNN Inference on Constrained Edge Nodes

Channel-wise Mixed-precision Assignment for DNN Inference on Constrained Edge Nodes

Matteo Risso†, Alessio Burrello*, Luca Benini*, Enrico Macii‡, Massimo Poncino†, Daniele Jahier Pagliari†
†Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy
*Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy
‡Inter-university Department of Regional and Urban Studies and Planning, Politecnico di Torino, Turin, Italy
Corresponding Email: matteo.risso@polito.it

***Abstract*—Quantization is widely employed in both cloud and edge systems to reduce the memory occupation, latency, and energy consumption of deep neural networks. In particular, *mixed-precision* quantization, i.e., the use of different bit-widths for different portions of the network, has been shown to provide excellent efficiency gains with limited accuracy drops, especially with optimized bit-width assignments determined by automated Neural Architecture Search (NAS) tools. State-of-the-art mixed-precision works *layer-wise*, i.e., it uses different bit-widths for the weights and activations tensors of each network layer. In this work, we widen the search space, proposing a novel NAS that selects the bit-width of each weight tensor *channel* independently. This gives the tool the additional flexibility of assigning a higher precision only to the weights associated with the most informative features. Testing on the MLPerf Tiny benchmark suite, we obtain a rich collection of Pareto-optimal models in the accuracy vs model size and accuracy vs energy spaces. When deployed on the MPIC RISC-V edge processor, our networks reduce the memory and energy for inference by up to 63% and 27% respectively compared to a layer-wise approach, for the same accuracy.**

***Index Terms*—Deep Learning, NAS, Quantization, TinyML**

I. Introduction

Deep Learning (DL) models can enhance the "smartness' of many edge systems, such as drones [1], wearables [2], and smart assistants [3]. However, the traditional paradigm based on offloading Deep Neural Networks (DNNs) execution to the cloud has several limitations in terms of latency, energy efficiency, and privacy, as it relies on continuous data exchange over the network, often through an unpredictable, unreliable and energy hungry wireless channel [4]. Furthermore, the huge carbon footprint of data center processing makes the sustainability of this approach questionable [5].

These limitations have spurred an increasing academic and industrial interest for Tiny Machine Learning (*TinyML*), i.e., the study of efficiency-driven optimizations of ML and DL models aimed at making their energy and memory requirements compatible with the tight constraints of edge devices. *Quantization* is a key step of most TinyML pipelines, which takes advantage of the inherent resilience of DNNs to replace the floating point data representations typically used during training with integers on a limited number of bits [6]. This is extremely beneficial for edge deployment, as it reduces the memory footprint of the model and avoids the use of slow, power-hungry floating point operations [6].

Conventional quantization strategies use the same precision for the entire DNN (so-called *fixed-precision*) [7], [8], but several recent works have shown that a *mixed-precision* scheme, using different bit-width for different network portions, can further reduce the DNN complexity without accuracy drops [2], [9]. However, the additional flexibility of mixed-precision comes with the new challenging problem of optimizing the *bit-width assignment* to different parts of the network. Partially to limit the scope of such optimization, state-of-the-art mixed-precision solutions currently assign bit-widths *layer-wise*. That is, the data representation is optimized independently for each network layer, possibly differently for weights and activations [9]. Even in this setting, the search space is huge; for example, considering a small MobileNet V1 [10] and 2/4/8-bit as available bit-widths, the number of possible solutions is 10^{26}. Accordingly, a proper optimization cannot be performed by hand, but requires automated Neural Architecture Search (NAS) tools. Recent works such as [9], [11] have therefore proposed different optimization methods to find precision assignments offering good trade-offs between the task accuracy and the total memory occupation of the network, or the latency on the final deployment target.

In this work, we assess for the first time the potential of an even finer-grain *channel-wise mixed-precision assignment*. That is, we explore the space of all possible bit-width assignments to each channel of each weight tensor in a Convolutional Neural Network (CNN), while maintaining layer-wise quantization granularity for activations. We achieve this goal thanks to a light-weight gradient-based search method, which belongs to the family of *Differentiable* NAS (DNAS). Our channel-wise scheme is fully compatible with existing hardware and software libraries, since the resulting multi-weights-precision convolutions can be easily reconverted into multiple smaller convolutions working in parallel. With experiments on the four benchmarks of the MLPerf Tiny suite [12] we show that our tool is able to find a rich front of Pareto-optimal solutions. When deployed on the MPIC [13] RISC-V edge processor, the networks found by our DNAS are able to reduce memory footprint/energy up to **63%/27%** at iso-accuracy with respect to a per-layer mixed-precision search. Our code is open-sourced at: https://github.com/EmbeddedML-EDAGroup/multi-prec-nas.

This work has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007321. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Belgium, Czech Republic, Germany, Italy, Sweden, Switzerland, Turkey.

Accepted as a conference paper at the 2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC).

II. Background and Related Works

Quantization. Quantization is used ubiquitously in TinyML to improve DNN compression and energy-efficiency [6]. Quantization-aware training (QAT) [7] is a de-facto standard technique that simulates the effect of quantization during training, through so-called *fake-quantization* operations, in order to improve the final accuracy of the quantized model. Post-training quantization [14] is a viable alternative when re-training is not possible.

This work targets the popular *affine quantization* scheme [6], that maps float tensors $\mathbf{t}$ to n-bit integers as:

$$\mathbf{t}_n = \underset{0:2^n-1}{\text{clamp}}\left(\text{round}\left(\frac{\mathbf{t}-\alpha_\mathbf{t}}{\varepsilon_\mathbf{t}}\right)\right) \tag{1}$$

where clamp simply restricts the values to $[0 : 2^n - 1]$, $[\alpha_\mathbf{t}, \beta_\mathbf{t})$ is the range of values that can be mapped without saturation, and $\varepsilon_\mathbf{t} = (\beta_\mathbf{t} - \alpha_\mathbf{t})/(2^n - 1)$ is the quantization step. Notable works that explore this quantization scheme are LQ-Net [15] and PACT [7], which learn the optimal step and value range during training. Note that our mixed-precision NAS can be be easily extended to non-linear quantization schemes [16], but this paper focuses on affine quantization due to its hardware friendliness, which results in a lightweight implementation on most deployment targets [6].

Neural Architecture Search. NAS tools are usually designed to explore the space of possible network *topologies* (e.g., selecting among alternative layers or tuning their geometric hyper-parameters). In that, they constitute a more effective alternative to the manual rules of thumb derived from experience that drove the design of early efficient DNNs [10].

In particular, NAS tools oriented at TinyML co-optimize the network performance (e.g., classification accuracy) and some cost metric (e.g., memory occupation, latency or energy consumption). Seminal approaches leverage an iterative procedure consisting of: i) sampling one ore more architectures, ii) training them to estimate task accuracy and iii) profiling their cost either through direct deployment or through some proxy model [17], [18]. Sampling is driven by black-box optimizers such as Evolutionary Algorithms (EA) [17] or Reinforcement Learning (RL) [18]. While powerful, these algorithms require thousands of iterations (hence GPU hours) per search, impairing their sustainability.

To cope with these limitations, Differentiable NAS (DNAS) techniques use gradient descent to simultaneously train the network and optimize its architecture, relaxing the topology optimization problem to make it differentiable [19]. Accuracy/cost co-optimization is achieved adding to the standard training loss a regularization term $\mathcal{L}_\mathcal{R}$, modeling the target cost metric (size, energy, etc) in a differentiable way. Thus, the loss function becomes:

$$\mathcal{L}(W;\theta) = \mathcal{L}_\mathcal{T}(W;\theta) + \lambda\mathcal{L}_\mathcal{R}(\theta) \tag{2}$$

where $\mathcal{L}_\mathcal{T}$ is the standard task loss, W is the set of network weights, and θ is the set of NAS parameters that determine the network topology. The scalar regularization strength λ can be tuned to guide the research towards more accurate or more lightweight networks. The encoding of alternative topologies is obtained from the values θ either: i) using them to perform a soft-selection among alternative versions of each layer, typically through a softmax (so-called *super-net* approaches) [19], [20], or ii) using them as masks to prune away parts of a large seed model, e.g., some channels in a Conv. layer (so-called *mask-based* approaches) [21], [22].

Mixed-Precision Assignment. Recently, NAS techniques have also been employed to address the bit-width assignment problem in mixed-precision DNNs. Similarly to topology-NAS, the goal is to simultaneously maximize accuracy and minimize inference costs. Some methods, such as RELEQ [23] and HAQ [11] use RL agents. While the black-box optimizer allows them to guide the precision search with actual latency or energy measurements from the target hardware, these methods suffer from the same efficiency downsides described previously for other RL-based NAS. The authors of EdMIPS [9], instead, proposed the first DNAS for mixed-precision assignment. To learn the optimal precision, EdMIPS *simulates* the effect of quantization at different bit-widths for the input activations and weights of each layer, similarly to fake-quantization in QAT [6]. The fake-quantized tensors are combined through "softmax-ed" NAS parameters, optimized during training. Using the formulation of (2), gradient-descent is encouraged to increase the parameters associated with lower bit-widths, if doing so does not harm accuracy. Since our method is inspired by [9] more details are provided in Sec. III.

III. Proposed Method

Mixed-Precision NAS methods like HAQ [11] and EdMIPS [9] assign an independent precision to the weights and activations of each network *layer*. Their rationale is that different layers exhibit different degrees of redundancy and feature extraction power, thus using the same precision for the entire network would be sub-optimal.

In this work we push this idea forward, proposing a lightweight DNAS method able to learn an independent precision assignment for each weight tensor *channel* in convolutional (Conv) or fully-connected (FC) layers, i.e., the two layer types that mostly influence the inference complexity of CNNs. In other words, we assign an independent precision to the weights of *each filter* in Conv layers[1], and to the weights associated with *each output neuron* in FC layers. We maintain the activation bit-width assignment layer-wise, for implementation reasons clarified below. In the following, we discuss the details of our method for Conv layers only, since the extension to FC is straight-forward.

Our channel-based precision assignment explores a larger and finer-grain solution space. For instance, considering the same MobileNetV1 mentioned in Sec I and a width multiplier of 0.25, the number of solutions grows from 10^{26} in a layer-wise approach to 10^{74} in ours. However, this allows our method to exploit the *different relative importance of extracted features* within a single layer to further optimize the model.

[1]By filter, we mean a slice of weights that processes all channels of an input activation patch, and produces a single output channel.

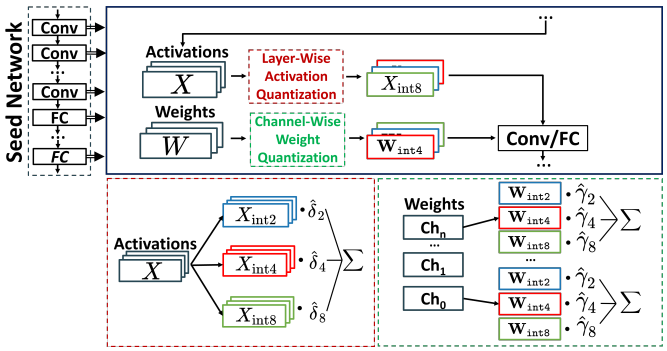

Fig. 1. Overview of the proposed approach.

A. Precision Assignment Optimization Method

Fig. 1 summarizes the flow of our approach. For each optimized layer of the target DNN, we first fake-quantize the activation tensor X at all supported bit-widths $p_x \in P_x$, e.g., $P_x = \{2, 4, 8\}$ bit. We then combine the fake-quantized tensors through a vector of NAS parameters $\delta^{(n)} \in \mathbb{R}^{|P_x|}$, where the n superscript refers to the n-th layer, in a way similar to [9]. Specifically, we first compute $\hat{\delta}^{(n)} = \mathrm{SM}(\delta^{(n)}; \tau)$ where $\mathrm{SM}(x; \tau)$ is the softmax with temperature:

$$SM(x;\tau) = \frac{e^{x_i}/\tau}{\sum_i e^{x_i}/\tau}, \ \forall i \tag{3}$$

so that the elements of $\hat{\delta}^{(n)}$ sum to 1. As detailed in Sec. III-B, the temperature τ is progressively annealed during training, driving the softmax to increasingly resemble a non-differentiable *argmax*, which is the function used to select the final precision at the end of the optimization.

Then, the *effective* activation tensor is obtained as:

$$\hat{X}^{(n)} = \sum_{p_x \in P_X} \hat{\delta}^{(n)}_{p_x} \cdot X_{p_x} \tag{4}$$

where X_{p_x} is the p_x-bit fake-quantized version of the original float tensor X, and we loosely use p_x to indicate both a precision and an index in the $\hat{\delta}$ array. Similarly to [9], the rationale of (4) is that the larger $\hat{\delta}^{(n)}_{p_x}$, the more $\hat{X}^{(n)}$ becomes similar to the result of a p_x-bit quantization.

To explore the channel-wise assignments of bit-widths $p_w \in P_W$ for weights, which is the main novelty of our work, we further associate each layer with a 2D matrix $\gamma^{(n)} \in \mathbb{R}^{C^{(n)}_{out} \times |P_W|}$, where $C^{(n)}_{out}$ is the number of output channels/filters in the layer. The i-th row of the matrix $\gamma^{(n)}_i$ contains the NAS parameters that will determine the bit-width assignment for the i-th channel. Accordingly, each row is applied an independent softmax to obtain $\hat{\gamma}^{(n)}_i = SM(\gamma^{(n)}_i; \tau)$.

Next, the i-th effective weight tensor slice $\hat{W}^{(n)}_i$, i.e., the i-th effective filter, is obtained similarly to (4), as:

$$\hat{W}^{(n)}_i = \sum_{p_w \in P_W} \hat{\gamma}^{(n)}_{i,p_w} \cdot W^{(n)}_{i,p_w} \tag{5}$$

Lastly, the stack of these slices along the C_{out} dimension is used, together with $\hat{X}^{(n)}$ to produce the layer output:

$$Y^{(n)} = \mathrm{Conv}(\hat{X}^{(n)}, \underset{i}{\mathrm{stack}}(\hat{W}^{(n)}_i)) \tag{6}$$

Our method uses *weight sharing*, that is, all fake-quantized weights and activations are obtained from a *single* float tensor, and are generated just once per-layer on the fly, i.e., we have $|P_W|$ and $|P_X|$ temporary copies of X and W during forward DNN passes. Thus, the memory overhead of our method during training is almost identical to [9], except for the new $\gamma^{(n)}$ matrix, whose impact is negligible. A key difference between our method and [9] is instead in the quantization scheme, i.e., the function mapping $X \rightarrow X_{p_x}$ and $W \rightarrow W_{p_w}$. Namely, we replace the original Gaussian quantizer used in [9] with the PaCT method described in [7]. The main reason is that PaCT layers are fully compatible with our deployment target [13]. Further, we found that it also yields superior results.

We optimize the precision assignment while training the DNN weights by minimizing the loss formulation of (2), where $\theta = \{\delta^{(1)}, ..., \delta^{(n)}, \gamma^{(1)}, ..., \gamma^{(n)}\}$ is the set of NAS trainable parameters for all layers.

We use two different expressions for the complexity regularizer $\mathcal{L}_\mathcal{R}$ depending on the optimization objective. Specifically, when the goal is to minimize the *model size*, i.e., the total amount of non-volatile memory required to store the network, the NAS should be guided to select smaller bit-widths for weights tensors channels where possible, whereas activations bit-widths have no impact. A simple regularizer that achieves this objective for a single Conv layer is:

$$\mathcal{L}_\mathcal{R}{}^{(n)} = C^{(n)}_{in} K^{(n)}_x K^{(n)}_y \sum_{i=1}^{C^{(n)}_{out}} \sum_{p_w \in P_W} \hat{\gamma}^{(n)}_{i,p_w} \cdot p_w \tag{7}$$

where $C^{(n)}_{in}$ $K^{(n)}_x$ and $K^{(n)}_y$ are number of input channels and the horizontal and vertical convolution kernel sizes. In practice, this regularizer computes the *effective number of weight bits* in the layer, multiplying each softmax coefficient $\hat{\gamma}^{(n)}_{i,p_w}$ times the corresponding bit-width p_w. Thus, for instance, it assigns a double cost to the coefficient associated with 4bit precision with respect to the 2bit one. Note that when optimizing the model size, we disable the search over activation bit-widths and fix all $\hat{X}^{(n)}$ tensors to 8bit.

We also consider the optimization of the *energy consumption* of the network, which depends both on weights and activations precision. The corresponding regularizer is:

$$\mathcal{L}_\mathcal{R}{}^{(n)} = \Omega^{(n)} \cdot \sum_{p_x \in P_X} \hat{\delta}^{(n)}_{p_x} \sum_{i=1}^{C^{(n)}_{out}} \sum_{p_w \in P_W} \hat{\gamma}^{(n)}_{i,p_w} \mathrm{C}(p_x, p_w) \tag{8}$$

where $\Omega^{(n)}$ is the total number of operations required to produce the n-th layer output, which is independent from the precision assignment and can be computed from well-known formulas for Conv or FC layers. The rest of (8) computes the *expected average energy per operation* of the layer. Specifically, $\mathrm{C}(p_x, p_w)$ is a Look-Up Table (LUT) returning an estimate of the energy/OP of a p_x-bit $\times p_w$-bit convolution. The LUT is populated profiling the target hardware, and is necessary because for most hardware platforms, the energy cost of arithmetic operations at sub-byte precision is not linearly proportional to the bit-width. The regularizer simply weighs each

Algorithm 1

for $i \leftarrow 1, \ldots, \text{Epochs}_{\text{wu}}$ **do** # warmup loop
  Update W based on $\nabla_W \mathcal{L}_{\mathcal{T}}(W_{p_{max}})$
while not converged **do** # search loop
  if #samples $< 20\%$ current epoch **then**
    Update θ based on $\nabla_\theta(\mathcal{L}_{\mathcal{T}}(W;\theta) + \lambda\mathcal{L}_{\mathcal{R}}(\theta))$
  else
    Update W based on $\nabla_W(\mathcal{L}_{\mathcal{T}}(W;\theta))$
  Anneal temperature τ
for $i \leftarrow 1, \ldots, \text{Epochs}_{\text{ft}}$ **do** # fine-tuning loop
  Freeze θ
  Update W based on $\nabla_W \mathcal{L}_{\mathcal{T}}(W)$

combination of NAS parameters for activations and weights with the cost of the corresponding mixed-precision operation.

The overall regularization loss for the NAS is obtained summing either (7) or (8) over all layers.

B. Training Procedure

Alg. 1 shows the training scheme of our method, which is made of three distinct phases. In the initial *warmup* phase, the network is trained normally with QAT at the maximum supported precision p_{max}, while keeping NAS parameters frozen. Thus, only the task-specific loss function $\mathcal{L}_{\mathcal{T}}$ and the normal W weights are optimized. In all our experiments, we consider $p_{max} = 8$bit. Warmup needs to be performed only once, reusing the result for multiple searches.

The second phase represents the core of the optimization. In it, both weights W and NAS parameters θ are trained to minimize (2). Following the scheme proposed in [21] we train NAS parameters and normal weights in an alternated fashion within each epoch. First, only the NAS parameters are trained on a random 20% split of the training samples. Then, only the network weights are trained on the remaining 80%. At the end of each epoch we anneal the softmax temperature τ to make the choice among alternative bit-widths more "decisive". In all our experiments, τ is initially set to 5 and progressively annealed by $e^{-0.0045}$ as in [21]. Note that in our closest prior work [9], the alternate W and θ training and the softmax temperature were not present. However, we found experimentally that both techniques improve the training stability and final result quality, not only for our approach but also for [9].

Lastly, in the *fine-tuning* phase, the θ parameters are frozen to the final learned values, and the softmax is replaced with an argmax to select a single precision for each weight channel or activation layer. Then, only the weights W are fine-tuned based on $\mathcal{L}_{\mathcal{T}}$. In all our experiments, the warmup and fine-tuning epochs $\text{Epochs}_{\text{wu}}$ and $\text{Epochs}_{\text{ft}}$ are set equal to the training epochs reported in the papers describing the reference DNNs. Conversely, the search phase termination is controlled with an early-stop mechanism.

C. Implementation Details

In this section, we show that besides having the potential to improve the *theoretical* compression and efficiency of DNNs compared to standard mixed-precision, our method is also *fully compatible* with existing hardware and software libraries for mixed-precision inference, with minimal overheads. The top-left part of Fig. 2 shows the DNAS output for a generic Conv layer, with all filters of assigned 2,4 or 8-bit, independently from their ordering in memory (e.g., the 3rd and last filter are 8-bit, while the 4th filter is 2bit, etc). To deploy such a layer, we first *reorder* the filters grouping them by bit-width (top-right of the figure). Importantly, this also affects the order of the output activations channels. Accordingly, to preserve the functionality of the *following* layer, its weight tensor has to be re-organized along the C_{in} axis, so that each weight is still multiplied with the correct input activation. In the figure, we show this associating a color pattern to the two filters that change position due to reordering, and using the same pattern to highlight the change in activations and the corresponding reorganization of the next layer weights.

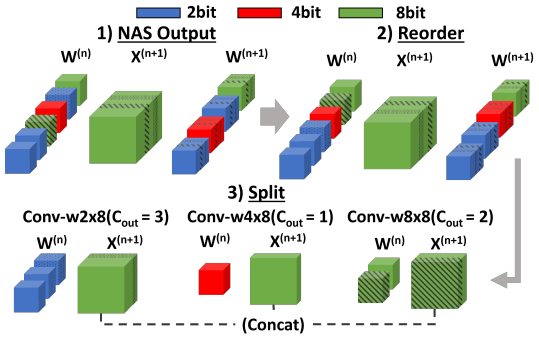

Fig. 2. Layer re-organization to support channel-wise precision assignment.

After this re-organization, which is performed offline and does not have run-time overheads, the layer can be *split* into $|P_W|$ separate convolutions working in parallel, each with a fraction of the original output channels. These sub-layers have a single bit-width for W, hence they are fully compatible with existing mixed-precision hardware and libraries [13], [14]. Moreover, since the activation bit-width is assigned layer-wise, all outputs have the *same* precision, and can be simply concatenated (i.e., stored in adjacent memory locations) to be readily usable as inputs for the next layer. Allowing this simple concatenation is the reason why we do not perform channel-wise precision assignment for activations too. In fact, in that case the next layer's filters would have to process an interleaved mix of different precision inputs, which is not currently supported by inference libraries. Nevertheless, the study of fine-grain activation precisions will be subject of our future work. In the current version, instead, the only overhead of our method compared to standard mixed-precision is the control flow to schedule the three sub-layers. However, this cost is negligible compared to the benefits of the proposed fine-grain assignment.

IV. Experimental Results

A. Setup

We evaluate our channel-wise mixed-precision DNAS on the four benchmarks of the MLPerf Tiny suite [12]. Below is a summary of each task, while more details can be found in [12]. *Image Classification* (IC) targets the CIFAR-10 dataset

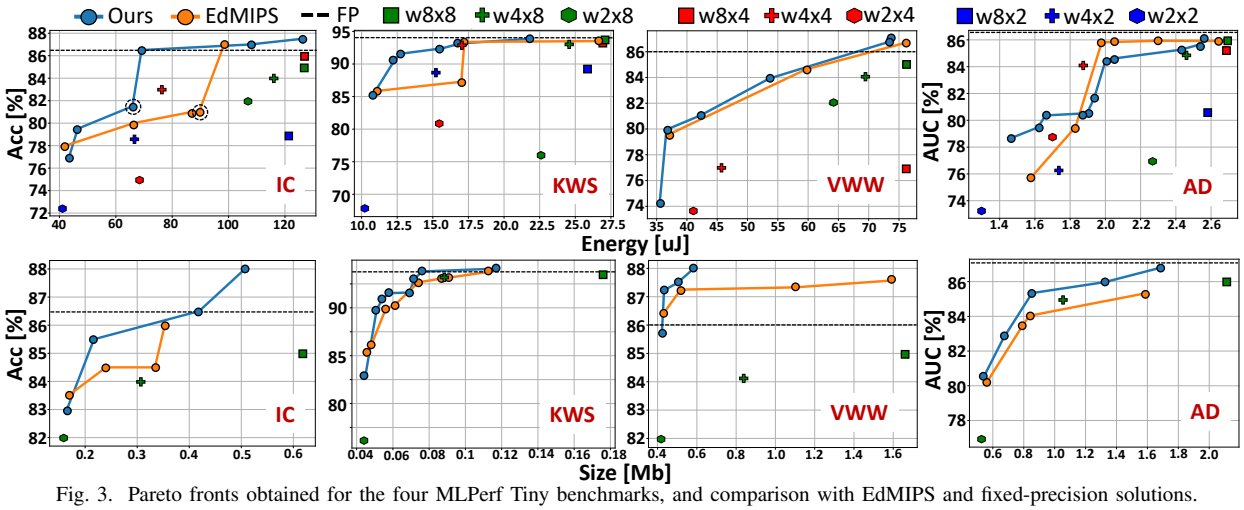

Fig. 3. Pareto fronts obtained for the four MLPerf Tiny benchmarks, and comparison with EdMIPS and fixed-precision solutions.

using a custom ResNet-like CNN with a backbone of 8 convolutional layers. *Keyword Spotting* (KWS) targets the Google Speech Commands (GSC) v2 dataset with a small Depthwise Separable CNN (DS-CNN) originally proposed in [3]. *Visual Wake Word* (VWW) uses the MSCOCO 2014 dataset for a presence detection task, solved with a MobileNetV1 [10] with a width-multiplier of 0.25. Lastly, *Anomaly Detection* (AD) targets the Toy-car subset of the DCASE2020 dataset, with a simple Dense Autoencoder.

The proposed DNAS is implemented in Python 3.9 using PyTorch v1.10.2. Our deployment target is MPIC [13], a RISC-V core including optimized hardware units for the execution of MAC operations with inputs independently quantized to $p_{w/x} \in \{2, 4, 8\}$ bit. The LUT implementing the cost function of (8) is built using the energy/OP values profiled from the MPIC core running at 250MHz, for all combinations of activations and weights precisions within the $p_{w/x}$ set.

B. Search-Space Exploration

Fig. 3 shows the results of applying our mixed-precision assignment method to the four MLPerf Tiny benchmarks. The vertical axes of all graphs report the task scores, i.e. the accuracy for IC, KWS and VWW and the Area Under the ROC Curve (AUC) for AD. The horizontal axes of the first row of graphs report the energy consumption of the models in μJ, whereas those in the second row report the model sizes in Mb.

Each plot compares the mixed-precision solutions found with our tool (blue dots) with the results of EdMIPS [9] (orange dots). To have a fair comparison, in which our method's advantages are solely due to the channel-wise precision assignment, we run our NAS and EdMIPS with identical training protocols, including the 20/80% alternate θ/W training and the τ annealing. Moreover, we replace the original quantization algorithm of [9] with PaCT [7], since the former would not be compatible with deployment on MPIC.

We also compare against all relevant fixed-precision quantization baselines, denoted as "wNxM", where N and M are the W and X bit-widths taken from the $\{2, 4, 8\}$ bit set. In memory plots, we only report wNx8 baselines, since the activations bit-width is not relevant for model size. Lastly, the accuracy of a floating point version of each model is shown as a horizontal dashed line. We do not report the float model energy and size since MPIC does not have a Floating Point Unit (FPU).

Each of the Pareto-optimal points reported in the graphs for our method and for [9] refers to a DNN with a different precision assignment. Multiple points are obtained changing the regularization strength λ in (2). We use the regularizer expressions of (7) and (8) for memory and energy results respectively, both for our tool and for [9]. Noteworthy, for all the benchmarks neither the float models nor the full 8-bit one outperform the best mixed-precision assignment, due to the well-known overfitting reduction effect of low bitwidth quantization [6].

The left-most part of Fig. 3 shows the results obtained on the IC task. Our fine-grain mixed-precision approach outperforms all fixed-precision baselines and EdMIPS, both in terms of energy and model size. Noteworthy, it saves up to 26.4% energy and 35% memory with respect to EdMIPS at iso-accuracy, while also obtaining a higher maximum accuracy, $+0.5\%$/$+1\%$ depending on the regularizer used. Similar results are obtained also for KWS (middle-left part of Fig. 3). Again, our method Pareto-dominates all comparison baselines, finding solutions that save up to 27.2% energy and 15.6% memory for the same accuracy, and improve the best score by $+4.3\%$ and $+0.7\%$ respectively.

The middle-right part of Fig. 3 reports the results on VWW. On this benchmark, we Pareto-dominate fixed-precision networks both in terms of memory and energy. Conversely, compared to EdMIPS, our method is particularly beneficial to reduce the memory footprint of the network, with up to 63.4% saving at equal accuracy, and a maximum accuracy improvement is $+0.4\%$. Instead, the benefit in terms of energy is limited, although we are still able to find a *larger number of Pareto-optimal points* with respect to EdMIPS, which translates to more flexibility for designers. Note that, for this benchmark, the fixed-precision networks with 2bit activations are not shown because their training does not converge.

Lastly, the right-most part of Fig. 3 shows the results on AD. As for all other benchmarks, our method obtains superior

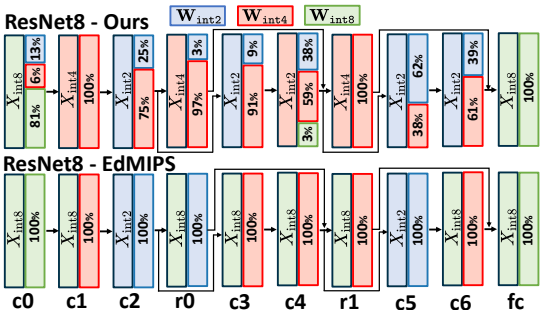

Fig. 4. Example of found architectures for the IC benchmark.

results in terms of AUC versus model size. The memory saving is at most 46.1% for the same AUC with respect to EdMIPS. In this case however, [9] outperforms our method in terms of AUC versus energy in the high score regime (up to 21.8% saving), while our method is superior for lower AUC values (up to 11.6%). We attribute this result to the more difficult optimization problem solved by our tool. In fact, the AD Autoencoder is composed solely of FC layers with 128 channels (i.e., neurons), except for the bottleneck. With so many channels, the difference between the search space explored by our method and by [9] explodes. Hence, the gradient-based DNAS optimization likely ends up in a local minimum. Nonetheless, we think that this issue could be solved by tuning the training hyper-parameters specifically for this task, whereas in our experiments we keep all settings identical across benchmarks for fairness and reproducibility.

C. Results Analysis

Fig. 4 shows an example of the precision assignments determined by our tool and by [9]. The two architectures are ResNet-8 for the IC task, obtained with the energy regularizer of (8), and correspond to the two circled Pareto points of Fig. 3, i.e., those for which our method obtains the largest energy saving with no accuracy drop. Specifically, our channel-wise model saves 26.4% energy with an accuracy improvement of $+0.5\%$. Each rectangle represents a Conv (cn or rn, where the latter are those in residual branches) or FC layer, with the activation bit-width reported on the left, and the fraction of weight channels associated to each precision on the right.

This example shows some interesting insights about the effectiveness of our approach. For instance, it can be noticed that EdMIPS quantizes most of the activations with 8bit, whereas our method exploits its additional flexibility to *reduce* the activation precision, compensating it with an increase in the bit-width assigned to an often small subset of the weights channels (e.g., only 3% in c4), in order to obtain the same final accuracy. Eventually, only the first and last layer activations, which notoriously often require higher precision [24] remain at 8bit. Although we report a single example for sake of space, similar considerations apply to the results on the other 3 benchmarks.

As a last remark, Fig. 3 results show that our method saves more memory than energy on MPIC. However, these memory savings could translate into further energy (and latency) reductions when considering more complex hardware, with multiple memory hierarchy levels, as larger portions of the DNN can be kept in faster and more efficient L1 memories [25].

V. Conclusions

In this work, we have proposed a new fine-grain mixed-precision search algorithm, able to efficiently select the optimal precision for each channel of each convolutional layer in CNNs. When compared with a state-of-the-art mixed-precision DNAS, and on four edge-relevant use-cases, our tool finds richer and better trade-offs in the accuracy vs. latency/energy-consumption space, showing up to **27%** energy consumption reduction and up to **63%** memory reduction with no accuracy penalty.

References

[1] D. Palossi *et al.*, "Fully onboard ai-powered human-drone pose estimation on ultralow-power autonomous flying nano-uavs," *IEEE IoT Journal*, vol. 9, no. 3, pp. 1913–1929, 2022.

[2] A. Burrello *et al.*, "Q-ppg: Energy-efficient ppg-based heart rate monitoring on wearable devices," *IEEE Trans. Biomed. Circuits Syst.*, 2021.

[3] Y. Zhang *et al.*, "Hello edge: Keyword spotting on microcontrollers," *arXiv:1711.07128*, 2017.

[4] F. Daghero *et al.*, "Energy-efficient deep learning inference on edge devices," in *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, Eds. Elsevier, 2021, vol. 122, pp. 247–301.

[5] B. Ramprasad *et al.*, "Sustainable computing on the edge: A system dynamics perspective," in *ACM Proc. of HotMobile '21*, 2021, p. 64–70.

[6] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *IEEE CVPR*, 2018.

[7] J. Choi *et al.*, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv:1805.06085*, 2018.

[8] Y. Choukroun *et al.*, "Low-bit quantization of neural networks for efficient inference," in *2019 IEEE/CVF ICCVW*, 2019, pp. 3009–3018.

[9] Z. Cai *et al.*, "Rethinking differentiable search for mixed-precision neural networks," in *Proc. IEEE/CVF CVPR*, 2020, pp. 2349–2358.

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

[11] K. Wang *et al.*, "Haq: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE/CVF CVPR*, 2019, pp. 8612–8620.

[12] C. Banbury *et al.*, "Mlperf tiny benchmark," *arXiv:2106.07597*, 2021.

[13] G. Ottavi *et al.*, "A mixed-precision risc-v processor for extreme-edge dnn inference," in *2020 IEEE ISVLSI*, 2020, pp. 512–517.

[14] A. Capotondi *et al.*, "CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices," *IEEE Tran. Circuits Syst. II: Express Briefs*, 2020.

[15] D. Zhang *et al.*, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. ECCV*, 2018, pp. 365–382.

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

[17] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. ICML*. PMLR, 2017, pp. 2902–2911.

[18] M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proc. IEEE CVPR*, 2019, pp. 2820–2828.

[19] H. Liu *et al.*, "Darts: Differentiable architecture search," *arXiv:1806.09055*, 2018.

[20] H. Cai *et al.*, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv:1812.00332*, 2018.

[21] A. Wan *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *CVPR*, 2020, pp. 12 965–12 974.

[22] M. Risso *et al.*, "Lightweight neural architecture search for temporal convolutional networks at the edge," *IEEE Trans. Comp.*, pp. 1–1, 2022.

[23] A. Elthakeb *et al.*, "Releq: an automatic reinforcement learning approach for deep quantization of neural networks," in *NeurIPS, 2018*, 2019.

[24] F. Daghero *et al.*, "Ultra-compact binary neural networks for human activity recognition on risc-v processors," in *Proc. 18th ACM CF*, 2021.

[25] A. Burrello *et al.*, "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus," *IEEE Trans. Comput.*, 2021.