# MATAR: Multi-Quantization-Aware Training for Accurate and Fast Hardware Retargeting

Pierpaolo Mori[1,2], Moritz Thoma[2], Lukas Frickenstein[2], Shambhavi Balamuthu Sampath[2],
Nael Fasfous[2], Manoj Rohit Vemparala[2], Alexander Frickenstein[2],
Walter Stechele[3], Daniel Mueller-Gritschneder[3], Claudio Passerone[1]
[1]Politecnico Di Torino, Turin, Italy; [2]BMW Group, Munich, Germany; [3]Technical University of Munich, Munich, Germany
{<firstname>.<lastname>}[1]@polito.it,[2]@bmw.de,[3]@tum.de

***Abstract*—Quantization of deep neural networks (DNNs) reduces their memory footprint and simplifies their hardware arithmetic logic, enabling efficient inference on edge devices. Different hardware targets can support different forms of quantization, e.g. full 8-bit, or 8/4/2-bit mixed-precision combinations, or fully-flexible bit-serial solutions. This makes standard quantization-aware training (QAT) of a DNN for different targets challenging, as there needs to be careful consideration of the supported quantization-levels of each target at training time. In this paper, we propose a generalized QAT solution that results in a DNN which can be retargeted to different hardware, without any retraining or prior knowledge of the hardware's supported quantization policy. First, we present the novel training scheme which makes the model aware of multiple quantization strategies. Then we demonstrate the retargeting capabilities of the resulting DNN by using a genetic algorithm to search for layer-wise, mixed-precision solutions that maximize performance and/or accuracy on the hardware target, without the need of fine-tuning. By making the DNN agnostic of the final hardware target, our method allows DNNs to be distributed to many users on different hardware platforms, without the need for sharing the training loop or dataset of the DNN developers, nor detailing the hardware capabilities ahead of time by the end-users of the efficient quantized solution. Models trained with our approach can generalize on multiple quantization policies with minimal accuracy degradation compared to target-specific quantization counterparts.**

## I. Introduction

Convolutional Neural Networks (CNNs) are widely used to solve various computer vision tasks. The remarkable accuracy achieved by state-of-the-art models usually comes at the cost of a large number of parameters and multiply-accumulate (MAC) operations that make deploying such models on resource-constraint edge hardware challenging. Resources on embedded platforms are often insufficient for such tasks, particularly when considering latency constraints in edge application scenarios such as autonomous driving. Therefore, in order to reduce their computational and memory footprint, quantization became a standard technique to compress the models before the deployment on edge hardware. Quantizing a CNN implies reducing the bit-width adopted to represent the weights and activations, at the cost of introducing quantization error that can lead to prediction quality degradation. Quantization-aware training (QAT) techniques are typically used to make the model aware of the quantization error at training-time to minimize such degradation [1], [2]. Standard QAT provides quantized models with high prediction quality. However, such models have predetermined quantization bit-widths, and their accuracy is guaranteed only for the quantization strategy adopted at training time. Modern CNN hardware accelerators have different architectures that can take advantage of different quantization strategies (e.g. low-bit uniform quantization, mixed-precision, or bit-serial). Therefore, each architecture requires a dedicated QAT solution in order to retrieve a model that can maximize the resource utilization and, thereby, the efficiency of the hardware platform adopted for inference. Thus, training a single model to be easily deployed on various hardware platforms, which benefit from different quantization strategies, with minimal prediction quality degradation, would enable the distribution of CNN models to many users without the need to share the training loop or dataset. In existing works ( [3]–[5]), the trained models work only for the quantization bit-widths adopted at training time and cannot generalize on unseen quantization policy or on mixed-precision solutions.

In this work, we present multi-quantization-aware training for accurate and fast hardware retargeting (MATAR), a new unified training scheme that can be used to train models considering multiple quantization bit-widths at training time, enabling the model to be quantized after training also on quantization bit-widths that were not considered during the training. MATAR provides a possible solution to train CNN models that can be deployed efficiently on different devices with different architectures and different quantization bit-widths without the need for retraining or fine-tuning.

The main contributions of this work can be summarized as follows:

- We present a novel training scheme that considers multiple quantization bit-widths at training time (MATAR). We show how sharing clipping factors and batch-norm parameters enables models trained with MATAR to generalize on quantization bit-widths not seen during the training.
- We explore the quantization retargeting capabilities of the models trained with MATAR on uniform and mixed-precision use-cases. Furthermore, we implement a genetic algorithm (GA) to search for mixed-precision configurations of MATAR-trained networks that optimize the hardware execution metrics, while minimizing the prediction quality degradation.
- MATAR enables the GA to evaluate solutions without the need for fine-tuning, making the GA fast and accurate in its navigation of the search space. This generates

979-8-3503-4859-0/DATE24/© 2024 EDAA

ready-for-deployment Pareto-optimal solutions for a given hardware target, without the need for fine-tuning.

## II. Related Work

### A. Quantization-Aware Training

QAT has been widely adopted to model the quantization error during training to reap the benefits of quantization without degrading the task accuracy. DoReFa-Net [1] was an early QAT work, adopting the straight-through estimator (STE) to approximate the gradient of discrete, quantized weights and activations that are clipped between (0,1). The work in PACT [2] improved the training procedure by introducing a layer-wise learnable clipping factor ($c$) to saturate the distribution of activations between $[-c, +c]$. Furthermore, AdaBits [5] first focused on adaptive deployment capabilities. The authors employed bit-width-specific clipping factors in a joint QAT framework to obtain models trained for different quantization bit-widths. Although the aforementioned works produced accurate quantized models, such quantized networks cannot generalize on quantization bit-widths different from those considered at training time.

In our work, we present a training scheme that allows the retrieval of accurate models quantized on unseen bit-widths. Shared clipping factors and batch normalization parameters allow to regularize activation distributions, paving the way to fine-tuning-free mixed-precision search and retargeting to different hardware platforms.

### B. Supernets

In [3], the authors proposed Once-for-All networks, an approach that trains a very large supernet that can be dissected later to provide lighter models for different hardware deployment targets. The work in [4] presented once-quantization-aware training (OQAT) as an extension of the supernet concept to quantization. In OQAT models, multiple quantization bit-widths are considered at training time, resulting in a massive network, where each quantized layer can update its latent weights according to the assigned bit-width. Once trained, such supermodels are dissected to get the quantized model that best fits the target hardware platform. Despite the remarkable prediction quality of such approaches and their generalizability on different devices, the time and resources required to train supernets is often prohibitive, limiting their democratization to most developers without access to extensive compute clusters. In this work, we embrace the single-training multi-quantization cause by introducing MATAR. However, different from the supernet approaches, we consider multiple quantized versions of the same model during training and limit the training time by sharing weights, clipping factors, and batch normalization parameters across the quantized layers.

### C. Hardware-CNN Co-design

Works such as [6], [7] consider hardware metrics to guide the search for efficient models. In HAQ [6], the authors resort to reinforcement-learning exploration to determine the optimal layer-wise quantization bit-widths. The RL-agent automates the search, evaluating the the quantized solutions on real hardware. In [7], the authors introduce APQ, an approach to jointly perform neural architecture search, pruning, and quantization. A quantization-aware accuracy predictor and a lookup table containing the hardware metrics for each layer are used to estimate accuracy and latency, respectively. The work in [8] leverages the multi-objective non-dominated sorting genetic algorithm (NSGA-II) to determine suitable layer-wise pruning ratios and evaluates the compressed networks on a model of the target hardware, speeding up the search. In AnaConGA [9], the authors propose a nested GA that jointly maximizes Pareto-dominant solutions for quantization strategies while exploring different sizes of the hardware accelerator.

Inspired by these works, in MATAR, we rely on a meta-heuristic approach, namely NSGA-II, to search for layer-wise quantization solutions that optimize the hardware metrics (compute cycles, DRAM accesses), and minimize the prediction quality degradation. Moreover, different from other works [6]–[9], the mixed-precision quantized solutions provided by our approach do not need fine-tuning, speeding-up the design space exploration. Table I summarizes a categorical comparison against the mentioned related works.

TABLE I: Comparison to existing works for adaptive deployment scenarios.

| Work | Search Cost | HW Retarget | Mixed-Precision |
|---|---|---|---|
| HAQ [6] | High | ✗ | ✓ |
| APQ [7] | High | ✗ | ✓ |
| OQAT [4] | High | ✓ | ✗ |
| AdaBits [5] | Low | ✓ | ✗ |
| **MATAR** | **Low** | ✓ | ✓ |

## III. Methodology

### A. Fundamentals of Quantization-Aware Training (QAT)

Without loss of generality, consider a convolutional layer $l \in [1, ..., L]$ in an $L$-layer deep CNN with weights $\mathcal{W}^l \in \mathbb{R}^{k_x \times k_y \times C_i \times C_o}$, represented by $\mathcal{W}_{bits}$ bits, computed against an input feature map $\mathcal{A}^{l-1} \in \mathbb{R}^{H_i \times W_i \times C_i}$ represented by $\mathcal{A}_{bits}$ bits. Here, $H_i$, $W_i$ and $C_i$ represent the feature map's spatial and channel dimensions. $k_x$, $k_y$ and $C_o$ are the kernel window and output channel dimensions. The convolution of $\mathcal{W}^l$ and $\mathcal{A}^{l-1}$ with stride $s$ produces $\mathcal{A}^l \in \mathbb{R}^{H_o \times W_o \times C_o}$, where $H_o$, $W_o$, and $C_o$ are output spatial and channel dimensions.

The complexity of a convolutional layer can be expressed as the number of bit-wise binary operations $BOPs$ that must be performed (Eq. 1).

$$\text{BOPs} = H_o \times W_o \times C_o \times k_x \times k_y \times C_i \times \mathcal{W}_{bits} \times \mathcal{A}_{bits} \quad (1)$$

Eq. 1 captures how reducing the number of bits allotted for weights and activations representation can relax the arithmetic complexity of the convolution operation. Quantization of the operands allows then to optimize memory movement and achieve high-throughput parallel computation on simpler integer arithmetic hardware. However, reducing the precision

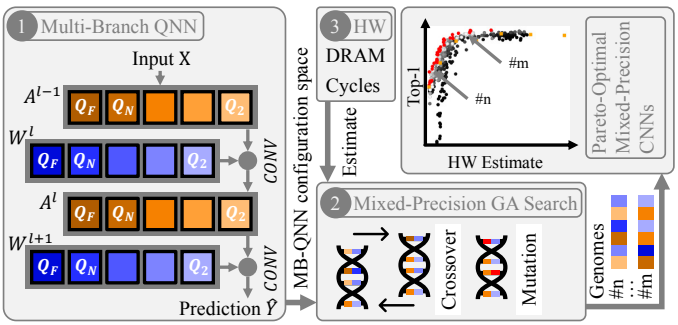



Fig. 1: Overview of MATAR. 1: the multi-branch model is trained first. 2: The GA looks for solutions to maximize accuracy and HW metrics. 3: Pareto-optimal solutions are evaluated on HW.

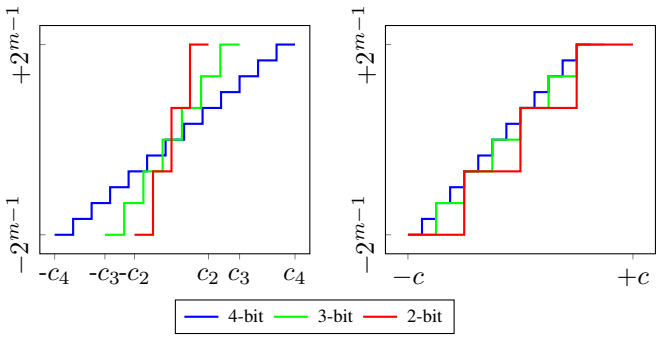



Fig. 2: On the left, other clipping approaches use different clipping factors to optimize the bit utilization. On the right, MATAR shares the clipping factor across multiple quantization bit-widths pushing the weights to better utilize the clipped range for different quantization-levels. Low-precision quantized values are determined by simply downsampling the higher-precision values.

of the operands involved in the convolution operation causes a numerical error (quantization error) that results in a prediction quality degradation, especially for low-bit quantization policies (4-bit and below). In such scenarios, it is necessary to consider quantization effects at training time, making the model aware of the quantization error. The quantization function $Q_n(x)$ applied to an arbitrary element $x$ (i.e. weight or activation pixel) to reduce its bit-width to $n$-bits is shown in Eq. 2. $x$ is clipped between $[-c, +c]$, where the clipping threshold $c$ is a trainable variable for every layer and is determined by the task-specific loss function of the CNN model's training [2]. Based on the determined $c$ for a given datatype, a scaling factor $v = c/(2^n - 1)$ is defined. For activations, we clip the values between the range of $[0, c]$ instead of $[-c, c]$, due to the ReLU activation function.

$$Q_n(x) = \texttt{Round}(\texttt{Clip}(x, -c, +c)/v) \times v \quad (2)$$

In order to deal with the discreteness of Eq. 2 which blocks the gradient flow during training, the straight-through estimator (STE) [10] is applied. This enables updating floating-point (latent) weights during backpropagation and using the quantized values during inference.

### B. Multi-branch Quantization-Aware Training

Our goal is to train a model such that it is aware of different quantization bit-widths, in order to easily retarget it and quantize it post-training to different quantization policies supported by different hardware platforms at deployment.

*1) Multi-branch Convolution:* We define a new 2D convolutional layer, characterized by $(M+1)$ inputs, the floating point one and $M$ quantized branches, where $M$ equals the number of quantization levels we want to make the model aware of. Each quantized branch is responsible for quantizing the weight and the input distributions according to Eq. 3, where $m$ defines the branch-specific quantization policy with $m \in M = [N, ..., 2]$, and $N$ defining the widest quantization level (e.g. 8-bit).

$$Q_m(x) = \texttt{Round}(\texttt{Clip}(x, -c, +c)/v_m) \times v; \quad \forall\, m \in M \quad (3)$$

$$v_m = c/(2^m - 1) \quad (4)$$

$v_m$ represents the scaling factor computed to map the discrete float range to the $m^{th}$ integer range ([-$2^{m-1}$, $2^{m-1}$-1]). For weights, we adopted [1], resulting in a clipping factor $c$ equals to 1, where the weights used in each quantized branch are the quantized version of the float weights of the floating branch on $m$ bits. Thus, we train ***only one weight tensor*** per layer. For activations, different from other works like [5], the clipping factor $c$ is a trainable parameter, and it is ***shared*** across all the branches. This strategy brings two main benefits: **(1)** Sharing the clipping factor allows the retrieval of the lower-bit quantization levels by simply downsampling the higher bit-width branches, without the need for re-quantizing the floating-point values. In Fig. 2, a visual comparison is made between other approaches that use quantization-specific clipping factors ( [2], [5]) and our approach that forces different bit-widths to share the same clipping factor. Low bit-width values can be easily computed by simply right-shifting higher bit-width data. **(2)** Moreover, sharing the clipping factor $c$ helps regularize the activation distributions across all the quantized branches, resulting in similar distributions. Consequently, layer-wise batch normalization across the branches can share the trainable parameters, and simply need to tune mean and variance according to the branch statistics.

In summary, each layer in the CNN model is replicated in $M + 1$ branches. The float weight tensor (the only trainable one), is quantized on $M$ different bit-widths, one for each branch. Then, each branch performs the convolution among the $m$-bit quantized input tensor and the $m$-bit weight tensor. The resulting output is finally quantized according to the bit-width adopted in the branch. This is depicted in block 1 of Fig. 1.

*2) Backpropagation Loss computation:* The $M+1$ branches make $M + 1$ corresponding predictions, resulting in $M + 1$ losses that are then accumulated, providing the overall model loss. We explore the opportunity to weight this loss sum (Eq. 5), by scaling the individual losses by $\alpha_m$, in order to increase the importance of the losses related to more challenging branches

(e.g. lower bit quantization).

$$\mathcal{L}_{total} = \sum_{m=0}^{M+1} \alpha_m \mathcal{L}_m \tag{5}$$

During backpropagation, the only weights that are updated are the ones belonging to the float branch, taking into account the contributions of the quantized branches. The gradient of the float weights of each layer $l$, can then be expressed as a weighted sum of the gradients flowing from the $m$ quantized branches (Eq. 6). Thus, the float weights are updated to reduce the loss from all the quantized branches, making the float weights aware of multiple quantization bit-widths.

$$\frac{\delta \mathcal{L}_{total}}{\delta w_l} = \sum_{m=0}^{M+1} \alpha_m \frac{\delta \mathcal{L}_m}{\delta w_{l_m}} \tag{6}$$

Similarly, the overall gradient for the layer-specific clipping factor $c_l$ for layer $l$ is computed by weighting the gradients of the $m$ quantized activations ($y_m$), as shown in Eq. 7.

$$\frac{\delta y}{\delta c_l} = \sum_{m=0}^{M+1} \alpha_m \frac{\delta y_m}{\delta c_l} \tag{7}$$

### C. Fast and Accurate Quantization for Hardware Retargeting

Once the MATAR model is trained as described in the previous sections, it is ready for deployment on several hardware platforms. According to the hardware's capabilities we can determine two classes of solutions: *uniform* and *mixed-precision*.

*1) Uniform Solutions:* Some hardware architectures are characterized by a processing element (PE) array optimized for a specific quantization bit-width (e.g. uniform 2,4, or 8-bit). In such scenario, our multi-branch model can be easily quantized uniformly without the need for fine-tuning. The shared trainable clipping factor ($c$) and batch normalization parameters across the branches allow the retrieval of quantized solutions that the model was not specifically trained for (e.g. in case of $M = [2, 4, 8]$, solutions for 3, 5, 6, and 7 bits can be determined without retraining).

*2) Mixed-Precision Solutions:* Other PEs can take advantage of the variable bit-widths (e.g. 4/8 bit: NVIDIA Orin, Qualcomm Snapdragon 8 Gen2, bit-serial: BISMO [11]) to speed-up the computation (lower bit-widths) or to reduce the quantization error (higher bit-widths). In this case, weights and/or activations of each layer $l$ can be quantized differently according to the number of supported quantization strategies ($S$). Therefore, it is crucial for such devices to properly select the optimal layer-wise quantization bit-widths according to the target constraints. The discrete, non-differentiable space of the possible quantized solutions for a CNN model composed of $\mathcal{L}$ layers has a massive size of $S^{2L}$.

*3) Genetic Algorithm-based Mixed-Precision Quantization:* We formulate the quantization of weights and activations as a search problem, relying on a genetic algorithm (GA) to efficiently explore the massive search space of mixed-precision solutions. A single genome represents a potential quantization strategy with as many genes as layers in the CNN. Each gene embeds a quantization strategy for the layer, setting the bit-widths for the weights ($\mathcal{W}_{bits}$) and the activations ($\mathcal{A}_{bits}$). We use a multi-objective Pareto-optimal selection approach (NSGA-II) that considers multiple HW/SW criteria for individual selection (e.g. task accuracy, number of execution cycles, DRAM accesses, etc.). Finally, single-point crossover and mutation operators are used to create more diverse offspring from fit parent genomes in the population. The GA search starts with an initial population size $|\mathcal{P}|$. Each individual in the population $\mathcal{P}$ is a CNN with randomly selected bit-widths for weights and activations ($\mathcal{W}_{bits}$, $\mathcal{A}_{bits}$, with $\mathcal{W}_{bits} \not\equiv \mathcal{A}_{bits}$). Then, these CNN models are evaluated and selected according to prediction quality and hardware metrics. The selected individuals go through single-point crossover and mutation to better explore the search space. This process is repeated for $g$ generations. The multi-branch quantized training of MATAR makes the pareto-optimal models selected by the GA ready-for-deployment without the need for retraining, thereby increasing the speed and fidelity of the search. This loop can be seen in box 2 and 3 of Fig. 1.

**Algorithm 1:** Genetic Algorithm for Mixed Precision Search.

**Initialize** : Genome encoding, Population size $|\mathcal{P}|$, crossover probability $p_c$, mutation probability $p_m$, random set of weights ($\mathcal{W}_{bits}$) and activations ($\mathcal{A}_{bits}$) bit-widths ($\mathcal{A}_{bits} \not\equiv \mathcal{W}_{bits}$), and maximum generation number $g$.
**Input** : Random initial population $\mathcal{P}$, CNN pretrained with MATAR.
**Output** : Pareto-optimal mixed-precision quantized solutions $\mathcal{P}_{opt}$, balancing accuracy and hardware metrics.
**for** *gen in range(1, g)* **do**
  **Evaluation:** Evaluate the fitness of each individual in $\mathcal{P}$ according to Accuracy and HW metrics.
  **Selection** : Non-dominated sorting and crowding distance selection.
  **Crossover** : Select two individuals and perform single-point crossover with probability $p_c$.
  **Mutation** : Perform replace mutation on each offspring with probability $p_m$. Update population $\mathcal{P}$.
**end**

## IV. Experiments

We evaluate MATAR with ResNet-20, ResNet-56, and ResNet-18 on CIFAR-10, CIFAR-100 and ImageNet datasets. We train the models using a NVIDIA A100 GPU and, if not otherwise mentioned, all the training hyperparameters are adopted from the base implementation. We use the Xilinx Z7020 SoC on the PYNQ-Z1 board to evaluate the quantized solutions found by the genetic algorithm.

### A. Multi-branch Quantization Aware Training

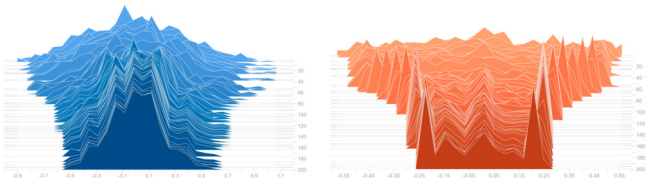
We first run an ablation study to analyze different configurations in terms of number of quantized branches (M), the loss scaling ($\alpha$) for each quantized branch, and training time (Table II). For each set of branches and loss scaling, we evaluate the accuracy of the float model and of all the uniform quantized solutions (from 2 to 8 bit). Considering all the possible branches ($|M|$=7) makes the training more challenging, leading to high accuracy solutions for higher bit-widths but degrading the

prediction quality for the 2-bit branch. On the other side, training the model with only the upper and lower-bound bit-widths (8 and 2), results in the shortest training time but causes an accuracy degradation for intermediate quantization levels. Moreover, as introduced in section III-B, we weight the loss so that the more challenging branches (lower bit-width) have a higher impact on the overall loss. In this way, during the training, the optimizer updates the float weights giving more importance to the effect of the low bit-widths. In the end, we identified the setup with $M = [8, 4, 2]$ branches and a weighted loss $\alpha = [0.4, 0.1, 0.2, 0.3]$ (where the first element in the set represents the loss scaling for the float branch), as the best trade-off in terms of accuracy among the different branches and training time.

TABLE II: Ablation study ResNet20 on CIFAR-10 Dataset.

| Branches (M) | Loss Weight ($\alpha$) | Accuracy Evaluation(%) Float | Q8 | Q7 | Q6 | Q5 | Q4 | Q3 | Q2 | Training Time (h) |
|---|---|---|---|---|---|---|---|---|---|---|
| [8, 7, 6, 5, 4, 3, 2] | [1, 1, 1, 1, 1, 1, 1] | 90.59 | **90.88** | **90.89** | **90.81** | 90.52 | 90.11 | **88.2** | 82.68 | 5.70 |
| [8, 2] | [1, 1, 1] | 90.51 | 90.14 | 90.14 | 90.18 | 78.68 | 79.68 | 82.57 | 84.03 | **3.60** |
| [8, 4, 2] | [1, 1, 1, 1] | 91.00 | 90.80 | 90.78 | 90.58 | 90.13 | 90.47 | 86.85 | 86.56 | 4.10 |
| [8, 4, 2] | [0.4, 0.1, 0.2, 0.3] | **91.39** | 90.78 | 90.77 | 90.47 | **90.64** | **90.49** | 87.800 | **86.99** | 4.10 |
| [8, 6, 4, 2] | [0.3, 0.1, 0.1, 0.2, 0.3] | 91.28 | 90.62 | 90.62 | 90.54 | 90.60 | 90.25 | 85.95 | 84.21 | 4.88 |

Following the depiction earlier in Fig. 2, we demonstrate in Fig. 3 the impact of multi-quantization awareness on the latent floating-point weight distribution of the MATAR-trained CNN. The histograms show the distribution of weights for the last convolutional layer of the considered ResNet20-CIFAR-10 example from Table II after 200 epochs of training, as well as an 8-bit QAT trained version for reference. The peaks emerging in the MATAR-trained distribution indicate that it can be sampled for different bit-widths without retraining, where the weight values that can be sampled for all quantization levels (the two peaks on the edges of the distribution) have the most frequently occurring values, and the middle peaks reflect other areas of the distribution which may be sampled for different quantization strategies.



(a) 8-bit weight distribution (b) MATAR weight distribution

Fig. 3: MATAR influences the weight distribution to accommodate future unseen quantization levels. The above example shows the latent floating-point weights of the last convolution layer of ResNet20-CIFAR10. The peaks in distribution emerging in the MATAR-trained network after 200 epochs differentiate it from simple 8-bit QAT.

### B. Uniform Quantization and Hardware Retargeting

The ablation study in Table II showed how 3 branches (M=[8,4,2]) are able to preserve the accuracy for quantization bit-widths that have never been seen during the training (i.e. 7, 6, 5, 3-bits), enabling to model to easily be retargeted to different quantization requirements of unseen hardware. In Table III, we further investigate this aspect on ResNet56 and ResNet18 models on CIFAR-100 and ImageNet datasets.

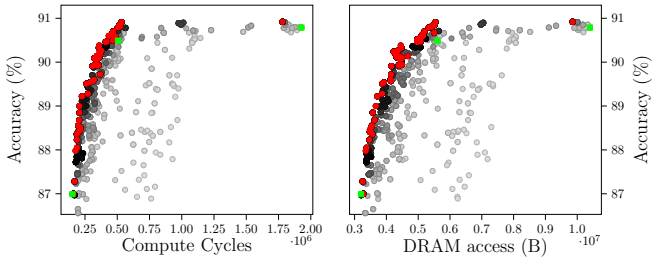
Additionally, to show that this property is only achievable with MATAR training, we also train each model using standard QAT [2] for a specific bit-width (8, 4, and 2-bits) along with one version using our MATAR approach. Then, we evaluate each version of the model (QAT 8-bit, 4-bit, 2-bit, and MATAR) ***retargeted*** on different uniform quantization bit-widths (8 down to 2-bit). In all the examples proposed, MATAR exhibits an interpolation capability to flexibly retarget to different quantization policies, without the need for retraining, while standard QAT models perform best on the bit-widths they were trained for and suffer when retargeted to other bit-widths (highlight in red for results degrading below MATAR). This characteristic is enabled by the decision to adopt the same trainable layer-wise scalar clipping factor ($c$) for all the MATAR branches, regularizing the activation distributions of the multiple branches. This feature allows for training the model only once, and deploying it on multiple hardware platforms, without the need of fine-tuning, sharing the dataset, or knowing the target hardware quantization policy ahead of time.

TABLE III: Uniform interpolation and hardware retargeting capabilities of models trained with MATAR compared to standard quantization specific solutions [2].

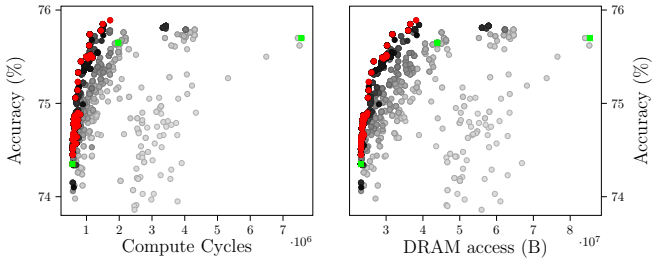
| Model/ Dataset | Approach | Training Q8 | Q4 | Q2 | Float Acc | Retargeted Accuracy (%) Q8 | Q7 | Q6 | Q5 | Q4 | Q3 | Q2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ResNet20 CIFAR-10 | Quant8 | ✓ | | | 92.08 | 92.08 | 92.03 | 91.36 | 88.05 | 64.53 | 20.29 | 9.93 |
| | Quant4 | | ✓ | | 91.86 | 91.85 | 91.76 | 91.82 | 91.77 | 92.12 | 87.56 | 16.82 |
| | Quant2 | | | ✓ | 87.68 | 67.68 | 67.64 | 68.42 | 69.45 | 68.08 | 71.79 | 87.7 |
| | **MATAR** | ✓ | ✓ | ✓ | **91.39** | **90.78** | **90.77** | **90.47** | **90.64** | **90.49** | **87.80** | **86.99** |
| ResNet18 CIFAR-100 | Quant8 | ✓ | | | 76.92 | 76.87 | 76.62 | 74.99 | 68.48 | 11.93 | 1.06 | 1.13 |
| | Quant4 | | ✓ | | 76.85 | 76.80 | 76.80 | 76.84 | 76.79 | 76.99 | 74.76 | 35.98 |
| | Quant2 | | | ✓ | 74.40 | 62.01 | 62.23 | 62.05 | 62.41 | 63.4 | 63.95 | 74.55 |
| | **MATAR** | ✓ | ✓ | ✓ | **75.95** | **75.70** | **75.73** | **75.76** | **75.50** | **75.48** | **74.65** | **73.74** |
| ResNet56 CIFAR-100 | Quant8 | ✓ | | | 69.95 | 69.86 | 69.68 | 68.08 | 61.47 | 29.50 | 3.32 | 1.04 |
| | Quant4 | | ✓ | | 69.70 | 69.22 | 69.20 | 69.22 | 68.78 | 70.41 | 60.83 | 14.52 |
| | Quant2 | | | ✓ | 62.00 | 30.75 | 30.53 | 31.96 | 33.66 | 34.83 | 61.89 | 62.00 |
| | **MATAR** | ✓ | ✓ | ✓ | **69.85** | **68.39** | **68.47** | **68.41** | **67.45** | **67.57** | **60.57** | **53.81** |
| ResNet18 ImageNet | Quant8 | ✓ | | | 67.48 | 67.42 | 67.21 | 65.72 | 54.67 | 16.89 | 0.14 | 0.11 |
| | Quant4 | | ✓ | | 66.87 | 65.08 | 65.20 | 64.85 | 64.77 | 66.61 | 51.92 | 0.17 |
| | Quant2 | | | ✓ | 64.35 | 31.24 | 31.24 | 31.66 | 31.2 | 35.96 | 38.46 | 63.12 |
| | **MATAR** | ✓ | ✓ | ✓ | **66.80** | **65.74** | **65.84** | **65.78** | **65.42** | **65.74** | **61.05** | **62.77** |

### C. Mixed-Precision Solution Search

We push the interpolation and retargeting capabilities of MATAR further by exploring mixed-precision quantization. The shared clipping factor $c$ and the shared batch normalization parameters regularize the activation distributions across the branches, enabling the possibility to derive new CNN models by fetching layers from different branches. Given a hardware accelerator that can take advantage of mixed-precision quantization (e.g. BISMO [11]), we look for weights and activation bit-widths that can maximize the hardware efficiency (compute cycles, DRAM accesses), without sacrificing the prediction quality of the pre-trained model. For such a massive space, we rely on the genetic algorithm search introduced in section III-C. For all the experiments, we run the GA with a population size $|\mathcal{P}| = 50$ and a number of generations $g = 50$. We set mutation ($p_m$) and crossover ($p_c$) probabilities to 0.9. We first synthesized the BISMO accelerator defined as HW3 in [6], [9], with BISMO hardware parameters $D_m$=$D_n$=8 and $D_k$=256. We adopt the equations defined by [9] to model the number of cycles and DRAM accesses. We then start the GA search,




(a) ResNet20-CIFAR-10




(b) ResNet18-CIFAR-100

Fig. 4: 2D facets of a 3D Pareto-front for optimal mixed-precision quantized solutions with respect to accuracy, compute cycles, and DRAM accesses. Red dots identify the Pareto-optimal solutions; green squares represent the accuracy-HW-metrics for uniform 8,4,2 bit-widths models.

looking for solutions that minimize the number of compute cycles and the number of DRAM accesses, while preserving the prediction quality of the models trained with our approach. We evaluated the final solutions on real hardware synthesized on the PYNQ-Z1. In Fig. 4, the Pareto-optimal solutions (red dots) found by the GA are presented for ResNet20 (Fig. 4a) and ResNet18 (Fig. 4b) models on CIFAR-10 and CIFAR-100 datasets, respectively. Green squares (■) indicate the baseline accuracy and hardware metrics for uniform 8, 4, 2 bit-widths models. In both cases, the GA provides a range of possible mixed-precision solutions ***retargeted*** with minimal accuracy degradation compared to 8-bit solution and lower compute cycles and DRAM accesses, that can be selected according to the deployment constraints. We further evaluate the proposed approach on ResNet18, ResNet20, and ResNet56 models trained with MATAR on CIFAR-10, CIFAR-100, and ImageNet datasets in Table IV. For each model, we run the genetic algorithm mixed-precision search and select solutions from the Pareto-front that maximize the accuracy (accuracy leader), the hardware metrics (HW leader), and the dominated hypervolume (HV leader). Moreover, we evaluated the solutions on real hardware and compared the estimated and measured cycles and DRAM accesses (Table IV). Finally, we reported the search time needed by the GA to finalize the search. Our approach requires 72 hours of training plus 21.4 hours for the mixed-precision search for ImageNet training, resulting in 12.84× speed-up compared to [4] and 25.8× compared to [7].

TABLE IV: Evaluation of HW, accuracy and hypervolume leader solutions on synthesized HW.

| Dataset | Model | Search Time (h) | Selection Strategy | Accuracy | HW Estimates | | HW Measured | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Cycles (k) | DRAM (MB) | Cycles (k) | DRAM (MB) |
| CIFAR-10 | ResNet20 | 0.89 | Accuracy Lead | 90.92 | 1.78e3 | 9.8 | 1.81e3 | 9.8 |
| | | | HW Lead | 86.99 | 153 | 3.2 | 156 | 3.3 |
| | | | HV Lead | 90.13 | 345 | 4.4 | 345 | 4.5 |
| CIFAR-100 | ResNet56 | 4.10 | Accuracy Lead | 68.49 | 2.08e3 | 17.6 | 2.08e3 | 18.1 |
| | | | HW Lead | 54.86 | 405 | 8.55 | 406 | 8.52 |
| | | | HV Lead | 66.3 | 960 | 12.6 | 960 | 12.8 |
| | ResNet18 | 4.25 | Accuracy Lead | 75.89 | 1.73e3 | 38.2 | 1.75e3 | 38.8 |
| | | | HW Lead | 74.34 | 572 | 23.2 | 581 | 23.6 |
| | | | HV Lead | 75.51 | 1.09e3 | 30.15 | 1.09e3 | 31.2 |
| ImageNet | ResNet18 | 21.42 | Accuracy Lead | 65.99 | 12.9e3 | 204 | 13.3e3 | 206 |
| | | | HW Lead | 63.02 | 2.28e3 | 80.7 | 2.30e3 | 81 |
| | | | HV Lead | 65.18 | 10.3e3 | 187 | 10.5e3 | 188 |

## V. Conclusion

In this paper, we presented MATAR, a new training scheme that makes the floating-point model aware of multiple quantization bit-widths, enabling fast and accurate post-train retargeting for quantized hardware execution. Unlike standard QAT, we show how models trained with MATAR can be retargeted on uniform quantization bit-widths not seen during the training. Furthermore, we adopted a genetic algorithm to explore the large solution space of mixed-precision models that can improve the efficiency of the hardware adopted. MATAR heavily reduces the search time since the mixed-precision solutions provided do not need retraining and are ready to be deployed on the desired hardware. MATAR represents a novel training scheme which sits between inflexible standard QAT approaches and quantized supernet training schemes in terms of search time and prediction quality.

## References


[1] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2018.

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

[3] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[4] M. Shen, F. Liang, R. Gong, Y. Li, C. Li, C. Lin, F. Yu, J. Yan, and W. Ouyang, "Once quantization-aware training: High performance extremely low-bit architecture search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5340–5349, 2021.

[5] Q. Jin, L. Yang, and Z. Liao, "Adabits: Neural network quantization with adaptive bit-widths," 2020.

[6] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8612–8620, 2019.

[7] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2078–2087, 2020.

[8] P. Morì, M.-R. Vemparala, N. Fasfous, S. Mitra, S. Sarkar, A. Frickenstein, L. Frickenstein, D. Helms, N. S. Nagaraja, W. Stechele, *et al.*, "Accelerating and pruning cnns for semantic segmentation on fpga," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 145–150, 2022.

[9] N. Fasfous, M. R. Vemparala, Frickenstein, *et al.*, "Anaconga: Analytical hw-cnn co-design using nested genetic algorithms," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 238–243, 2022.

[10] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013.

[11] Y. Umuroglu, L. Rasnayake, and M. Sjalander, "Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing," 2018.