Supplementary Material: Hardware-Aware Mixed-Precision Neural Networks using In-Train Quantization

Manoj Rohit Vemparala*[1]
manoj-rohit.vemparala@bmw.de
Nael Fasfous*[2]
nael.fasfous@tum.de
Lukas Frickenstein*[1]
lukas.frickenstein@bmw.de
Alexander Frickenstein[1]
alexander.frickenstein@bmw.de
Anmol Singh[1]
anmol.singh@bmw.de
Driton Salihu[2]
driton.salihu@tum.de
Christian Unger[1]
christian.unger@bmw.de
Naveen Shankar Nagaraja[1]
naveen-shankar.nagaraja@bmw.de
Walter Stechele[2]
walter.stechele@tum.de

[1] BMW AG
Munich, Germany

[2] Chair of Integrated Systems
Technical University of Munich
Munich, Germany

S.1 Understanding Training Behaviour

In Fig. 1, we demonstrate the training behaviour of the proposed in-train quantization approach. We highlight the training curves for the ablation study performed in Tab. 1 of the main paper, to understand the influence of the scaling factor v on the final prediction accuracy and BOPs reduction. We consider ResNet20 and ResNet56 trained on the CIFAR-10 and CIFAR-100 datasets to observe the improvement in accuracy and reduction in BOPs. For the 4-bit uniform quantization (indicated in blue), we observe that the BOPs remain constant across the training steps. We constrain our mixed-precision models to a achieve 2× BOPs reduction compared to the uniform 4-bit model (see Tab. 1 of the main paper for final quantitative improvements). We train uniformly quantized and constrained mixed-precision models for 300 epochs with a step learning decay policy. For uniform quantization, we decay the learning rates by 0.1 at 80, 160 and 240 epochs. For the constrained mixed-precision models, we decay the learning

* indicates that the authors have contributed equally.

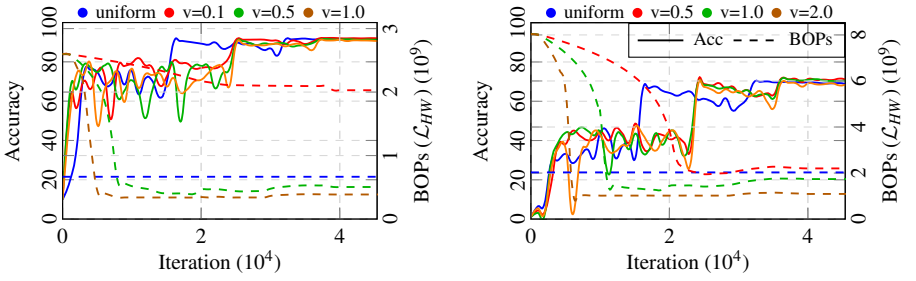
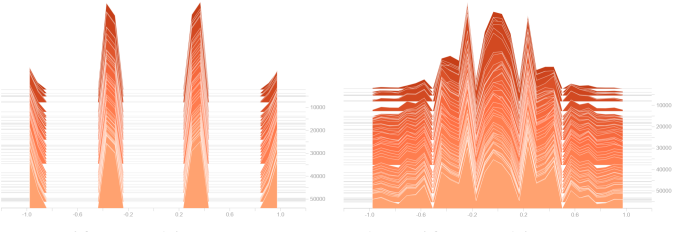


Figure 1: Comparison of in-train quantization behaviour for ResNet20 (left) and ResNet56 (right) with uniform quantization and different scaling factors.

rate at 120, 180 and 240 epochs. We ensure convergence in the quantization strategy before the learning rate decay. For our constrained mixed-precision models (indicated by red, green, orange), we observe reduction in BOPs across the training steps. We increase the scaling factor v to achieve the desired BOPs constraint. For ResNet56, we observe 1.2×, 1.6× reduction in BOPs compared to the uniform 4-bit configuration for scaling factor v= 1.0, 2.0, respectively, with no degradation in accuracy. We demonstrate the weight distributions indicating the effectiveness of progressive quantization in the supplementary material Sec. S.2.

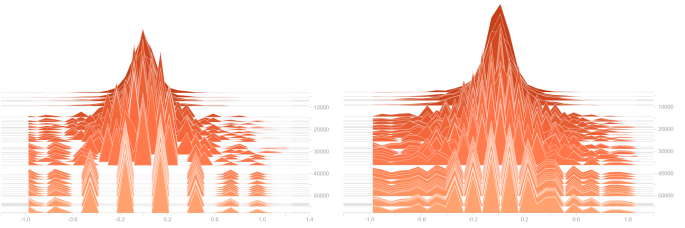
In Fig. 2 and Fig 3, we compare the distribution of W_q for our approach against the distribution under uniform quantization. We observe that uniform quantization training [2] does not change the number of unique values (peaks) across the training steps, allocating a *fixed* number of values from the start of the training. In our approach, we observe *progressive* quantization, starting from a smooth normal distribution and slowly converging to discrete peaks of quantized values. This allows larger gradient flow during the initial stages of the training and improves the trade-off between prediction accuracy and HW complexity for the resulting mixed-precision neural network.



(a) uniform 2-bit `conv2_9` (b) uniform 4-bit `conv2_9`

Figure 2: Distribution of quantized weights W_q for uniform PACT [2]

In Fig. 4, we compare the resulting quantization strategy for the BOPs optimized (top) and latency optimized (bottom) mixed-precision models. We observe lower bit-widths for the latency optimized mixed-precision strategy for the shallower layers. In the deeper layers, we observe higher assignment of bit-width for the activations. This is due to deeper layers being less computationally intensive, allowing more bit-widths to be allocated to maintain prediction accuracy, without adding too much latency. Particularly, activation bit-widths are increased, which have smaller volumes in the deeper layers of the CNN.



(a) 8bit→3-bit `conv2_7` (b) 8bit→4-bit `conv2_9`

Figure 3: Distribution of quantized weights W_q for our proposed approach

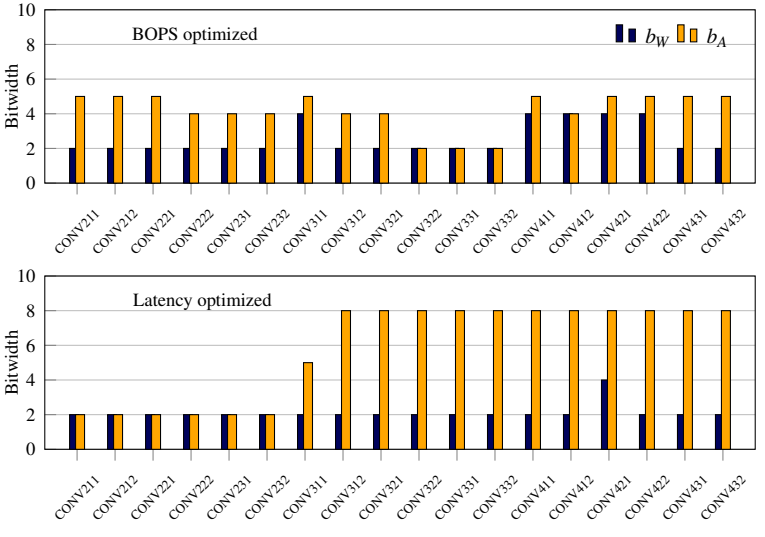


Figure 4: Comparsion of layerwise bit-width strategy for BOPs (top) and latency (bottom) optimized in-train quantization.

S.2 Mixed Precision DeepLabv3 for Semantic Segmentation

The semantic segmentation task is critical to applications in robotics and autonomous driving. High-quality segmentation is computationally complex by several orders of magnitude, when compared to classification tasks. This complexity is due to the typically larger input image resolution and the additional layers needed for semantic segmentation (bottleneck, ASPP block and decoder layers). For the DeepLab-based CNN architecture, we use ResNet18 as the backbone network and the last two residual blocks use dilation rate of 2. The Atrous Spatial Pyramid Pooling (ASPP) block contains dilation rates of {1, 8, 12, 18}. Our approach produces 0.7 pp better mean average over union (mIOU) with 15% lower BOPs and similar training cost.

Table 1: Comparison of our approach on DeepLabV3 based semantic segmentation with uniformly quantized models.

Model/ Dataset	Quant.	BOPs (G)	mIOU (%)	Training Cost (GPU hours)**
DeepLabv3 CityScapes	8bit [5]	5365	66.6	26hrs 1min
	4bit [5]	1469	65.4	
	Ours	**1254**	**66.1**	28hrs 23min

** Training cost is measured on a NVIDIA V100 GPU

S.3 Execution Schedule Awareness

To demonstrate the sensitivity of the optimal quantization strategy with respect to the target hardware, we can formulate GP regressors which capture different scheduling schemes on the inference hardware. For our target BISMO bit-serial accelerator, convolutional and dense workloads are transfromed into GEMM workloads.

The convolution workloads can be lowered into a general matrix multiplication (GEMM), by representing the W^l and A^{l-1} tensors as 2-D matrices $\texttt{Mat}_W$ and $\texttt{Mat}_A$, according to Eq. (1). The dimensions m and n represent the rows and columns of each matrix.

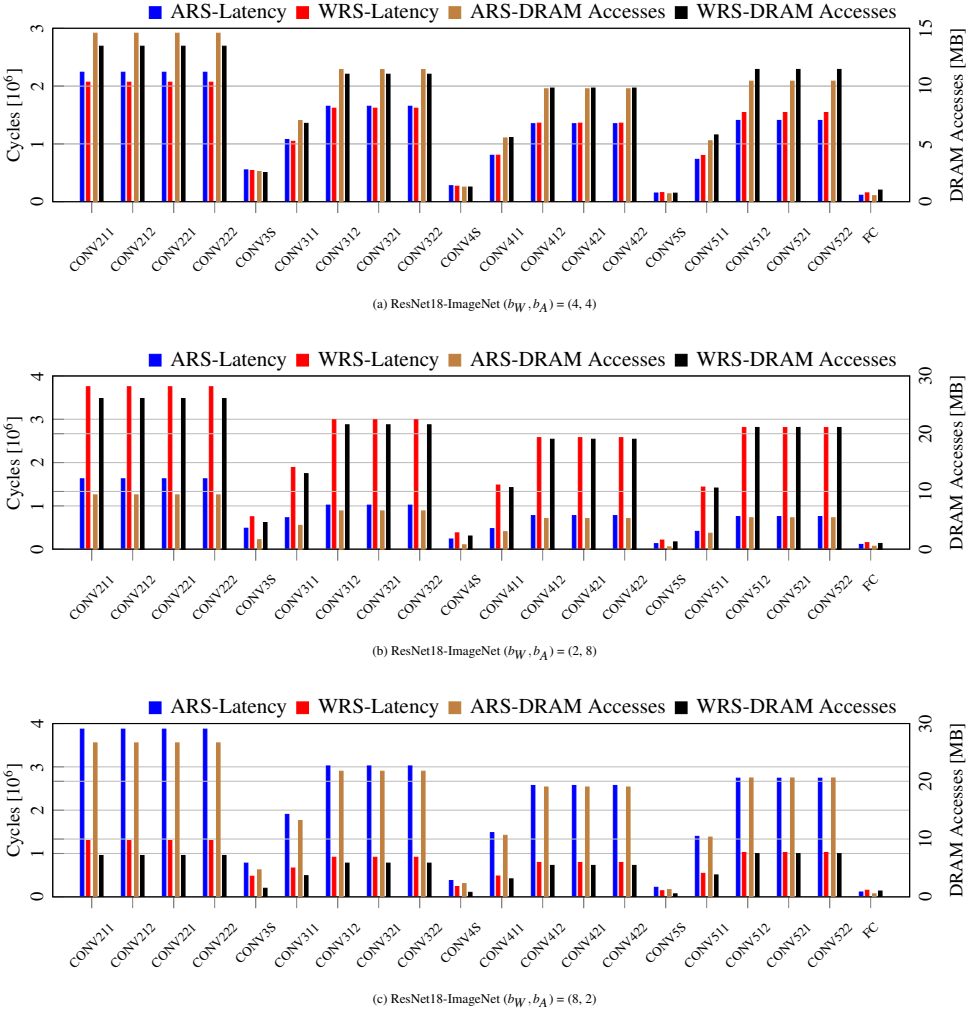
$$\begin{aligned} \texttt{Mat}_W \in \mathbb{R}^{m_W \times n_W},\ \texttt{Mat}_A \in \mathbb{R}^{m_A \times n_A} \\ m_A = n_W = C_i \times K_x \times K_y, \\ m_W = C_o,\ \ n_A = X_o \times Y_o \end{aligned} \tag{1}$$

$$A^l = \texttt{Conv}(W^l, A^{l-1}) = \texttt{Mat}_W \times \texttt{Mat}_A \tag{2}$$

Note that in Eq. (2), transposing both matrices and switching their order would also produce the convolution result. Hardware accelerators typically exploit data reuse to minimize the number of costly off-chip DRAM calls they need to perform during execution. For example, if one column from the right-hand side (RHS) matrix is to be computed against every row from the left-hand side (LHS) matrix, the accelerator can load the RHS column once and stream through all the LHS rows, until the column has been used exhaustively for the GEMM computation. The BISMO scheduler executes GEMM workloads in this manner, although it is agnostic to the workload being provided. We could then consider forcing the weights $W^l \ \forall\ l \in L$ to remain in the RHS matrix throughout the execution, which would result in a weight-reuse schedule (WRS). Conversely, maintaining activations A^{l-1} in the RHS, would result in an activation-reuse schedule (ARS).

To verify the HW-awareness of our method, we consider the contrasting ARS and WRS schemes, which favor the reuse of either activations or weights, respectively. We construct GP regressors which can capture the differences between these schedules and analyse the effect of these subtle HW-specific details on our in-train quantization method. With this, we verify the degree of HW-awareness possible through our differentiable GP regressors and in-train quantization technique.

In Fig. 5-a, we execute ResNet18 for ImageNet on an 6×6×256 BISMO array [5]. The weight and activation bits are set to 4-bits for all layers (uniform). We see that reusing the smaller volume of weights at the start of the CNN leads to WRS performing better, while ARS improves the execution for the latter half of the network, where reusing the activations is more beneficial. This is due to the fact that the total redundant reads are reduced when reusing the smaller volume datatype, as a larger portion of it can be stored on-chip and used exhaustively, leading to fewer total processing passes.



(a) ResNet18-ImageNet $(b_W, b_A) = (4, 4)$

(b) ResNet18-ImageNet $(b_W, b_A) = (2, 8)$

(c) ResNet18-ImageNet $(b_W, b_A) = (8, 2)$

Figure 5: ARS and WRS execution of ResNet-18 for ImageNet with uniform quantization. The results motivate the in-train quantization method to make schedule-aware decisions on layer-wise, datatype bit-widths, as shown in the paper.

In Fig. 5-b and -c, we execute ResNet18-ImageNet again, once with $(b_W, b_A) = (2, 8)$ and then with $(b_W, b_A) = (8, 2)$, for all layers. It can clearly be observed that with the expensive cost for reading the 8-bit wide datatype, the corresponding schedule which reuses that datatype has a significantly improved execution compared to the other. For example, ARS performs better than WRS for all layers, when the activations are 8-bit wide (Fig. 5-b). Conversely, WRS beats ARS when $b_W = 8$ (Fig. 5-c). Therefore, reusing the more *costly* datatype brings an advantage to the execution schedule. It is important to note, that all runs presented in Fig. 5 are on the exact same HW, but with a different schedule. This indicates that simply knowing the theoretical peak operations per second (OPS) of a hardware accelerator is not sufficient to have *real* HW-awareness. Subtleties, such as the schedule and datatype reuse highly influence the execution.

In the paper, we showed that our in-train quantization technique is aware of the accelerator *as well as* the execution schedule. The in-train quantization method benefits from the differentiable HW-metrics provided by the GP regressor, allowing it to consider such subtleties of the HW and make more HW-friendly decisions, which result in benefits on real synthesized HW.

S.4 Quantization-Aware Adversarial Training

To realize adversarially robust mixed-precision neural networks, we target two objectives: 1) compressing a model to reduce the computational complexity of a neural network, and 2) increasing the robustness against an adversary manipulating input data. Both can be effectively achieved by formulating a joint optimization problem. We adopt the concept of fast adversarial training [6], but learn the number of unique values $|U| \in \{1, 256\}$ required to represent weights and activations for each layer with our proposed method. We derive the quantized value $\tilde{x_q}$ based on the proposed progressive quantization approach in Sec. 3.2 from the main paper. We detail the quantization for weights and activations as W_q and A_q in Eq. (3) and Eq. (4), respectively. We clip the activations $\mathtt{A}$ between the range $[0, +c]$ due to the non-linear activation function (ReLU) and approximate them as $\mathtt{A_q} \in \{0,\ 1,\ 2, ...,\ (2^b - 1)\}$, similar to [2].

$$\mathtt{A_q} = Round(\mathtt{Clip}(A, 0, c) \cdot \frac{(2^b - 1)}{c}) \cdot \frac{c}{(2^b - 1)} \tag{3}$$

We clip the weight values using a $tanh()$ function and limit the range between [-1, 1], similar to the work in DoReFa-Net [2]. We approximate the continuous domain of weights $\mathtt{W}$ into the discrete values $\mathtt{W_q} \in \{\ -(2^{b-1} - 1), ..., -2,\ -1, 0,\ +1,\ +2, ...,\ (2^{b-1} - 1)\}$.

$$\mathtt{W_q} = 2\big(Round(\mathtt{Clip_{tanh}}(W) \cdot (2^{b-1} - 1)) \cdot \frac{1}{(2^{b-1} - 1)}\big) - 1 \tag{4}$$

$$\mathtt{Clip_{tanh}}(\mathtt{x}) = \frac{tanh(x)}{2max(tanh(x))} + \frac{1}{2} \tag{5}$$

Attacks against a neural network are described as finding a minimal perturbation δ of an image I (forming the the adversarial example $I_{\text{adv}} = I + \delta$) with label Y that changes the outcome of a given model represented by the prediction function $f(\cdot)$ [3]. For adversarial

training, we make use of this generation principle, while maximizing the loss $\mathcal{L}$ for a given perturbation budget ε:

$$\min_{W,M} \mathop{\mathbb{E}}_{(I,Y)\sim\mathcal{D}} \left[\max_{|\delta|\leq\varepsilon} \mathcal{L}\left(f(I+\delta, W_q), Y\right) \right]. \tag{6}$$

Our in-train quantization approach aims to achieve a balanced trade-off between natural accuracy Acc_{nat} (calculated from the ground-truth labels Y for the corresponding images I), adversarial robustness Acc_{adv}, and model complexity φ during the training process, rather than introducing separate (post-training) phases for finding the quantization strategy with iterative fine-tuning. In principle, one may use different methods for generating adversarial examples for training, such as FGSM [3], PGD [2] and CW [1]. Wong et al. [6], however, show that using FGSM in combination with random initialization is particularly effective. With this, the cost of training, measured in GPU hours, with one iteration of FGSM, is significantly lower than with other variants like PGD-based adversarial training [2]. We integrate the in-train update operations of the unique values U for quantized weights W_q and activations A_q in the FastAT procedure as shown in Alg.1.

Algorithm 1: Joint learning of quantization strategy and adversarial training.

Require : Training samples $\mathcal{D}$, perturbation strength ε, step size α

1 Initialize θ, $|U| \leftarrow 256$
2 **for** $Epoch = 1, \ldots N_{epochs}$ **do**
3 **for** *Batch* $B \subset \mathcal{D}$ **do**
4 Initialize perturbation $\delta \leftarrow random\_uniform(-\varepsilon, +\varepsilon)$
5 Sample a batch of K examples $\{(I^{(1)}, Y^{(1)}), \cdots, (I^{(K)}, Y^{(K)})\}$ from data distribution.
6 Use FGSM attack to generate perturbations on batch K to update δ
7 $\delta \leftarrow \delta + \alpha \cdot sign(\nabla_\delta \mathcal{L}(f(I+\delta, Qunat(W)), Y)$
8 $\delta \leftarrow \max(\min(\delta, \varepsilon), -\varepsilon)$
9 $I_{\text{adv}} \leftarrow I + \delta$
10 Update weights W and unique values $|U|$ using SGD for adversarial images:
11 $W \leftarrow W - \eta \cdot \nabla_W \mathcal{L}(f(I_{\text{adv}}, Qunat(W)), Y)$
12 **if** $E_{Prune,\ Start} \leq Epoch \leq E_{Prune,\ End}$ **then**
13 $|U| \leftarrow |U| - \eta_{|U|} \cdot \nabla_{|U|} \mathcal{L}(f(I_{\text{adv}}, Qunat(W)), Y)$
14 **end**
15 **end**
16 **end**

During each training step, we generate a uniform random initialization for the adversarial perturbation as shown in line 4, followed by performing a step into the ascent gradient direction (line 7) scaled by the step size α. We update the weights and unique values $|U|$ of the neural network jointly in line 11 for clean and adversarial images, with learning rate η. During these update steps, the importance scores for trainable unique values for each layer $|U|$ get accumulated. Line 13 reduces the number of unique values based on a hardware loss $\mathcal{L}_{\text{HW}}$ and cross entropy loss $\mathcal{L}_{\text{ce}}$ (see Sec. 3.2 in the main paper). As shown in line 12, we start and freeze the quantization strategy at the epoch corresponding to $E_{\text{Quant, Start}}$ and $E_{\text{Quant, End}}$ respectively.

Hyperparameters for Adversarial Training. For the robustness aware mixed-precision adversarial training (Sec. 4.3), we use random FGSM with strength $\varepsilon = 8/255$ and step size $\alpha = 10/255$ to generate adversarial perturbations during the training process. Similar to Wong *et al*. [6], we use a cyclic learning rate schedule, which linearly increases the learning rate from zero up to the maximum learning rate ($\eta = 0.2$), and decreases back to zero. A period of 10 epochs is used, where the maximum is reached at 5 epochs. For evaluating robustness of the quantized models, the PGD attack is performed with $\varepsilon = 8/255$ and $\alpha = 2/255$ for 20 iterations.

References

[1] Nicholas Carlini and David A. Wagner. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017. doi: 10.1109/SP.2017.49.

[2] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I.-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *ArXiv*, abs/1805.06085, 2018. URL http://arxiv.org/abs/1805.06085.

[3] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[4] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

[5] Yaman Umuroglu, Lahiru Rasnayake, and Magnus Sjalander. Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *Field Programmable Logic and Applications (FPL)*, 2018.

[6] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=BJx040EFvH.

[7] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, 2016. URL http://arxiv.org/abs/1606.06160.