

Digital Object Identifier

# Hardware-Aware Pruning for Efficient Inference on Embedded Devices

MARTIN LECHNER<sup>1,2,3</sup>, and AXEL JANTSCH<sup>1,2</sup> (Fellow, IEEE)

<sup>1</sup>Institute of Computer Technology, TU Wien, 1040 Vienna, Austria

<sup>2</sup>Christian Doppler Laboratory for Embedded Machine Learning, Institute of Computer Technology, TU Wien, 1040 Vienna, Austria

<sup>3</sup>Mission Embedded GmbH, 1100 Vienna, Austria

Corresponding author: Martin Lechner (e-mail: martin.lechner@tuwien.ac.at).

This work was supported in part by Austrian Federal Ministry for Digital and Economic Affairs; in part by the National Foundation for Research, Technology and Development; in part by the Christian Doppler Research Association, and in part by Technical University (TU) Wien Bibliothek through its Open Access Funding Programme

**ABSTRACT** With more powerful yet efficient embedded devices and accelerators being available for Deep Neural Networks (DNNs), machine learning is becoming an integral part of edge computing. In application scenarios with real-time requirements and multiple sensors connected to a single embedded device, such as autonomous driving, the optimization of DNNs plays a critical role during development and deployment to meet latency targets. However, optimization methods like pruning do not consider the underlying hardware platform. In this work, we propose a hardware-aware extension for existing pruning algorithms to optimally utilize the target hardware platform without requiring direct access to the device. We use estimation models to find Pareto-optimal points as pruning targets and apply three different optimization strategies to optimize latency (Stacking) or accuracy (Clipping), or to find a trade-off between both (Rounding). Our experimental results on embedded devices like a Nvidia Jetson Nano or a low-power accelerator like the Hailo-8 show a latency decrease of 21.44% on average when optimization for latency (Stacking), an accuracy increase of 2.46% when optimizing for accuracy (Clipping), and a latency decrease of 7.46% together with an increase in accuracy of 1.32% for Rounding across multiple DNNs, datasets, target sparsities, and embedded devices when compared to standard pruning.

**INDEX TERMS** Artificial neural networks, Pruning, Optimization, Estimation, Neural network hardware

## I. INTRODUCTION

DNNs emerged as a vital component in various use cases, such as computer vision and speech processing solutions. With the increasing demand for intelligent devices, the applications of DNNs are widespread, ranging from the automotive sector [1]–[3] and industrial applications [4] to medical imaging [5] and consumer-focused applications, such as virtual and augmented reality. However, executing computationally demanding DNNs on embedded devices with limited resources is often a challenge, making it necessary to run them on large data centers requiring a continuous and reliable network connection between the server and the device.

Embedded devices are becoming more capable with the addition of specialized hardware. This includes on-chip Graphics Processing Units (GPUs) (e.g., Nvidia Jetson family), integrated (e.g., NXP i.MX 8M Plus and i.MX 93) or external (e.g., Intel NCS2 or the Hailo-8 AI) Neural Processing Units (NPU), as well as design support for Convo-

lutional Neural Networks (CNNs) on FPGA platforms (e.g., Xilinx Vitis AI). These edge devices play an essential role in industrial applications, such as Advanced Driver Assistance Systems (ADAS), where real-time decisions must be made, or in rural areas where a stable network connection cannot be guaranteed. Incorporating embedded computing platforms can help developers achieve lower system latency, enhance system stability, and reduce complexity due to reduced communication needs.

However, the increasing complexity of State-of-the-Art CNNs still presents challenges in achieving desired latency goals. To address this issue, optimization techniques such as pruning [6]–[8], quantization [9], and shunt connections [10] have been devised and effectively implemented. On edge devices, the mentioned optimization techniques have their limitations. Specifically, pruning tools do not consider the underlying hardware, resulting in layer sizes that are not tuned for optimal utilization of the target platform.

As a motivational example, Figure 1 shows a single convolution layer with a fixed input size of  $64 \times 64$ , 64 input channels, and a varying number of output channels executed on embedded GPU platforms (Nvidia Jetson Nano and Jetson Orin Nano 8GB with TensorRT), an ARM processor with an integrated NPU (NXP i.MX 8M Plus), and an external accelerator (Hailo-8). Despite being different devices, the inference times follow a step function with different step widths and sizes. As such, the points immediately preceding a step (highlighted in green) stand out as they best utilize the underlying hardware architecture. Adding a single filter to the convolution requires an additional iteration during processing, resulting in a distinct jump in latency. We can consider these points Pareto-optimal since reducing the number of filters while staying on the same step has little to no advantage in terms of latency, but potentially decreases accuracy. Thus, pruning should ideally end up at the aforementioned points. This intuition is also covered in an extensive study in [11], where experiments show that pruning towards suboptimal points can even increase latency on certain hardware platforms.

Additionally, pruning is typically carried out on a machine-learning server with a specific compression rate or pruning factor as a target. However, this leads to a lack of knowledge about the model's real-world performance, particularly its latency, on an embedded platform, which can pose a significant challenge for developers seeking to optimize their models for deployment on resource-constrained devices.

To tackle the limitations discussed above, we propose a novel method to bridge the gap between pruning tools running on servers and the intended embedded platforms. We present a meta-method that wraps around an existing pruning algorithm to guide it toward Pareto-optimal points. We compare three variants: A simple clipping algorithm that cuts off any pruning beyond the Pareto-optimal points, a stacking algorithm that increases pruning towards Pareto-optimal points, and a combined approach that balances these two goals.

Specifically, this paper makes the following main contributions:

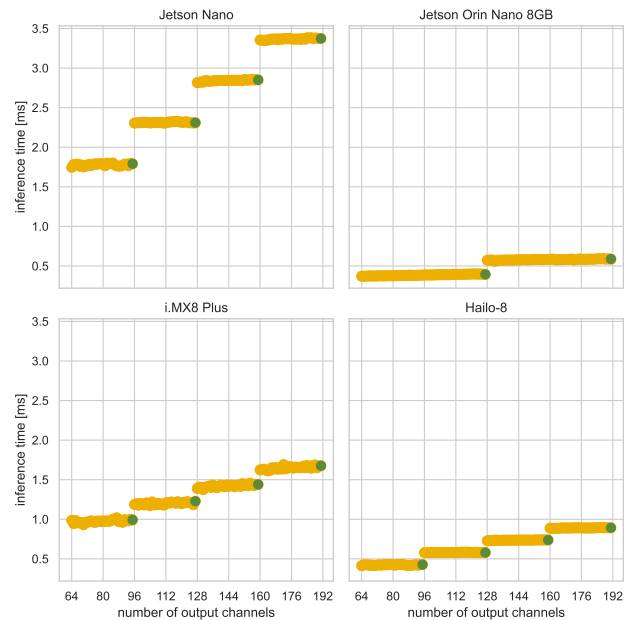
- A platform-aware pruning meta-method for optimized hardware utilization.
- A method to guide pruning towards Pareto-optimal layer sizes using platform models.
- A set of optimization strategies balancing accuracy and latency.

The rest of the paper is organized as follows. Section IV describes our approach to hardware-aware pruning. The results and comparisons to the state-of-the-art are shown in Section V with a conclusion drawn in Section VI.

## II. RELATED WORK

### A. NEURAL NETWORK PRUNING

Pruning is a well-established and effective method for compressing DNNs with minimal accuracy loss, dating back to a



**FIGURE 1.** A single convolution layer with a fixed input size of  $64 \times 64$ , 64 input channels, and a varying number of output channels executed on different hardware platforms and accelerators. The Pareto-optimal points are highlighted in green.

time long before the emergence of current DNNs. Optimal Brain Damage, introduced in 1989 [6], describes a technique for reducing the size of neural networks built out of dense layers by selectively deleting weights based on second-derivative information. It aims to improve generalization, reduce the number of training examples, and speed up learning by balancing training error against network complexity. As pointed out in [12], setting weights to zero results in sparse matrices in convolution layers, which dominate modern deep learning models. However, state-of-the-art hardware platforms have limited or no support for benefiting from sparse matrices. As such, they suggested removing entire feature maps based on the scale of the weights. This selection strategy is commonly referred to as magnitude pruning. Other methods differ in the way they select the removable weights. In [13], the authors utilize the scaling factors of batch normalization layers, [14] analyzes the sensitivity of each filter in a neural network and removes insensitive ones, while [15] identifies and removes redundant filters.

Based on how pruning operates on a neural network, it can be classified by different criteria: Weight pruning (targeting individual weights) or filter pruning (targeting entire filters), and structural pruning (removing weights or filters from a network) or non-structural pruning (setting weights or filters to zero). Following this scheme, the work in [6] uses non-structural weight pruning while [12] uses structural filter pruning. Analyzing weight and filter pruning, [16] found that non-structural weight pruning, while leading to the sparse matrix problem, achieves a better accuracy at the same compression rates as structural filter pruning. Thus,

they introduce stripe-wise pruning to prune filters at a finer granularity by targeting stripes within filters to maintain hardware compatibility while achieving high compression ratios. Modern CNNs introduce inter-layer dependencies like the residual connections in ResNet [17] and inverted residuals in MobileNetV2 [18], which pruning algorithms struggle to resolve. Group Fisher Pruning [19] presents a compression method to analyze such structures effectively. They introduce a layer grouping algorithm to identify coupled channels and employ Fisher information for importance evaluation. Torch-Pruning [8] is a general-purpose pruning toolkit using dependency graphs to resolve inter-layer dependencies. CLIP-Q [7] presents a combined approach for pruning and quantization, taking advantage of the complementary nature of both optimization techniques. CHEX [20] addresses the issue of pruning being a post-training optimization that still requires finetuning. They present a method for pruning during the initial training of a DNN, with the possibility of increasing the size of layers if necessary.

Although considerable work has been done on improving pruning algorithms to increase the compression rate while maintaining high accuracy, hardware-aware pruning for embedded hardware platforms is rarely addressed. As previously observed [21], [22], minimizing the number of floating-point operations (FLOPs) does not necessarily minimize the inference time. Observing the presence of step-like functions on GPU platforms, [21] presents an evolutionary Design Space Exploration (DSE) based approach to pruning towards optimal layer sizes. However, the experiments are limited to a desktop-grade GPU (Nvidia RTX 2080 Ti). In [23], the approach is extended using explainable AI metrics to rank the filters before pruning. In addition, the study conducted in [11] demonstrates the occurrence of step functions and shows that pruning towards arbitrary points between Pareto-optimal points can even increase inference latency on some devices. Partial Order Pruning [24] uses latency estimations based on look-up tables of layerwise measurements to find an optimal trade-off between latency and accuracy during pruning. Similarly, HALP [25] formulates structural pruning as a global resource allocation optimization problem using pre-measured latencies organized in a look-up table to optimize the latency-accuracy trade-off targeting server and desktop-grade GPUs. HALP uses a combined criterion for importance and latency to select filters for pruning, lacking flexibility to adapt to new developments. Torch-Pruning [8], [26] offers latency pruning based on measurements after each iteration, limiting the practical usage to training infrastructure. They replace the common sparsity target with a latency target and stop pruning when the latency target is reached without including platform-specific optimizations.

Our method uses prebuilt platform models to select pruning targets. This allows us to separate the target platform from the platform where the pruning algorithms are running, enabling us to optimize pruning for embedded devices. As our meta-method wraps around existing pruning tools, guiding the importance estimation towards optimal pruning targets,

we are flexible in swapping the importance estimation and pruning algorithm depending on the network architecture.

### B. TARGET ESTIMATION

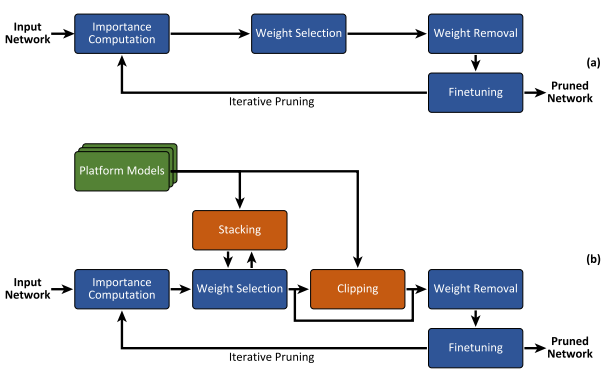
Optimizing the pruning targets towards the aforementioned Pareto-optimal points requires some knowledge of the underlying hardware architecture, which can be extracted from latency estimation models. Benchmark suites, such as Dawn-Bench [27], [28], MLPerf [29], and EMBench [30], provide databases containing latency measurements of DNNs for various platforms and software frameworks. However, extracting Pareto-optimal points is impossible as they only provide measurements for complete DNNs without layerwise information.

Dense, layerwise estimation models can provide sufficient information to obtain hardware-specific target points for pruning. Rouhani et al. [31] utilize micro-benchmarks to estimate the costs of individual operations and the number of neurons per layer to predict the latency for each layer. Paleo [32] develops an execution time model on a per-layer basis using an analytical approach that includes memory transfer and execution times. However, this model is based on heuristics implemented in CuDNN and is therefore restricted to Nvidia GPUs. NeuralPower [33] relies on a polynomial regression model to predict inference time, power, and energy. Their benchmarks are conducted via the Nvidia System Management Interface (SMI), only accessible on desktop and server-grade GPUs. PreVIOUS [34] introduces a slightly modified approach that utilizes linear regression models targeted at embedded CPU platforms, such as the Raspberry Pi 3. Blackthorn [35] uses layerwise analytical models based on linear and step functions to estimate the latency on embedded Nvidia GPUs, claiming to outperform other approaches by a large margin. Annette [36] proposes a methodology that extracts models from micro-kernel and multi-layer benchmarks, generating a combination of statistical and analytical models (mixed models) for network inference time estimation. Similarly, nn-Meter [37] is a kernel-based latency estimation framework for mobile CPUs, GPUs, and other edge devices.

Our pruning meta method is, in principle, agnostic to the latency estimation method. Analytical methods are advantageous because the mathematical formulas can be directly used, as we explain in section IV-A. However, as we also elaborate below, other latency estimators can be used as well by imposing an analytical estimation on top. Hence, our proposed pruning method can utilize any existing latency estimators.

## III. NEURAL NETWORK OPTIMIZATION FOR EMBEDDED SYSTEM

The primary goals in optimizing neural networks for embedded systems are to reduce their size and complexity, thereby requiring fewer computational resources for execution. Besides developing new, efficient architectures specifically designed for embedded devices, existing networks can be optimized in different ways. On the architectural level,



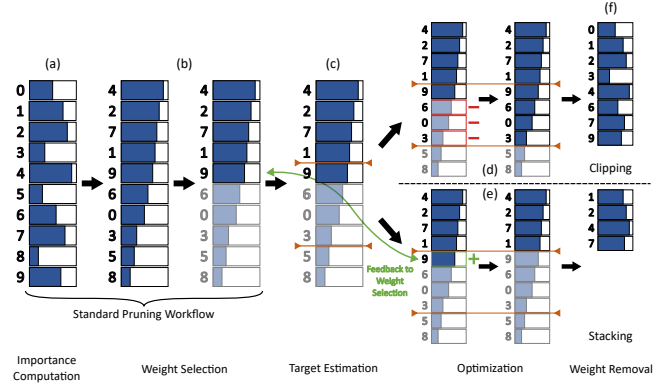
**FIGURE 2.** Overview of a standard pruning workflow (a) and integration of Clipping and Stacking (b). For Clipping and Stacking, only the corresponding block is active, while Rounding decides between Clipping and Stacking at runtime. The platform models guide Clipping and Stacking towards the Pareto-optimal points.

complex activation functions, such as the Exponential Linear Unit (ELU), which require computing exponential functions, can be replaced by simpler activations, like the Rectified Linear Unit (ReLU). At a network level, the size of a neural network can be reduced by removing filters or weights (pruning) or by reducing the complexity of the data type, e.g., from floating-point to integer (quantization). Additionally, the entire network can be simplified by merging layers, such as combining convolution layers with the activation function into a single layer. Together with partitioning the network onto the available resources in an embedded device, layer merging is typically performed by the hardware-specific toolchain, e.g., TensorRT in the case of Nvidia devices or Hailo-RT on the Hailo-8. A more exhaustive overview of neural network optimization methods can be found in [38], [39], with a detailed introduction to pruning in [40].

#### IV. HARDWARE-AWARE PRUNING

The process of pruning is an important step towards platform-optimized neural networks. It typically involves two iterative phases: pruning to reduce the network size and finetuning to regain lost accuracy. The pruning phase can be further divided into three distinct parts: importance computation, weight selection, and weight removal (Figure 2 (a)).

In the first step, weights or filters are rated by their impact on the accuracy of the complete network using an importance criterion (importance computation); the least important ones are tagged for removal (weight selection). Figure 3 (a)-(b) illustrates this process on a small example. In the second step, these tagged weights are eliminated from the network (weight removal), where the weights or entire filters can be set to zero or completely removed from the network structure, as shown in Figure 3 (f). The latter is commonly referred to as structural (filter) pruning. In the context of embedded systems, structural pruning has clear benefits in terms of performance [12]. Setting individual weights to zero (weight pruning) leads to sparse matrices. Since most embedded



**FIGURE 3.** Illustration of a typical pruning workflow. First, the importance of each filter (0-9) is computed (a). The length of the blue bar illustrates the importance of each filter, and the least important filters are selected for removal (b). Using our target estimation method, we find the Pareto-optimal pruning targets, indicated by the orange bars (c). Clipping (d) and Stacking (e) then modify the pruning masks so that the number of remaining filters matches the pruning targets. Finally, the selected filters are removed from the layer (f).

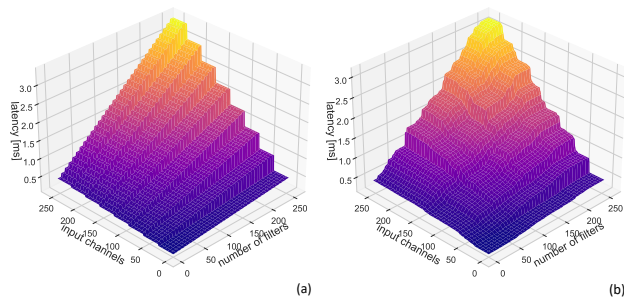
devices have limited or no support for sparse computing, no latency gains can be achieved, with a reduced memory footprint being the only remaining advantage. Therefore, this work focuses solely on structural filter pruning.

Our approach extends existing pruning methods to include hardware awareness, thus complementing them. Our meta-method is agnostic to the used importance criterion and the implementations of weight selection and weight removal since we only inject additional functionality into the weight selection, and between weight selection and weight removal as indicated by the orange blocks in Figure 2 (b).

#### A. PRUNING TARGET ESTIMATION

A critical step towards hardware-aware pruning is identifying the optimal pruning targets, i.e., analyzing the relation between latency and layer size to select the Pareto-optimal points in a step function, as outlined in the introduction. Therefore, platform-specific execution models are required for all covered devices. Compared to direct measurements, (analytical) models are not affected by measurement noise. In contrast to latency estimation models such as Blackthorn [35], Annette [36], and nn-Meter [37], we are interested in an accurate representation of the dependencies of the inference time on the shape and size of, for example, a convolution layer, rather than an accurate latency prediction. Thus, we need access to the shape of the estimated latency as a function of layer type and layer size to find possible target points. Analytical models, as implemented in Blackthorn, have the advantage of direct access to such data, while statistical or machine-learning-based models need to be evaluated or benchmarked first.

In Blackthorn, a combination of step functions and linear functions is used to model the latency of a single convolution layer. The models are built using multiple measurements through the device-specific execution provider, e.g., Ten-



**FIGURE 4.** Two-dimensional intersection of an analytical model (a) and a Random Forest Regressor fitted on the same data (b). Both have a fixed input size of  $w_{in} = h_{in} = 64$ . The statistical model is significantly smoother around the steps compared to the analytical model.

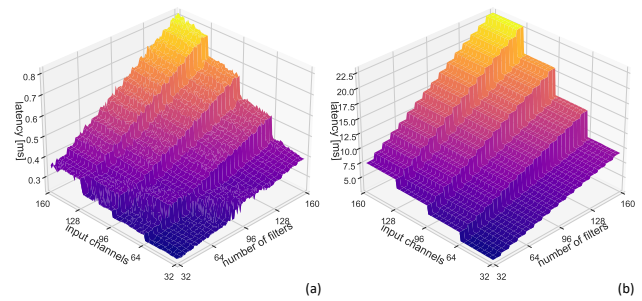
sorRT on Nvidia Jetson devices and Hailo-RT on the Hailo-8. The analytical functions are then fit to the measurements, excluding outliers, to achieve a platform model. The resulting platform model is always a combined, high-level model of the underlying hardware architecture and the inference framework. The step functions are represented using Equation (1), with the base latency  $d$  denoting the latency of a layer with a size of one, the step width  $w_{step}$ , and the step height  $h_{step}$  for a single dimension. Thus, we can directly use  $w_{step}$  to estimate pruning targets, and the Pareto-optimal points are then  $n \cdot w_{step}$  for  $n \in \mathbb{N}$ . Target estimation as part of our pruning workflow is illustrated in Figure 3 (c), indicated by the orange bars.

$$f_{step}(x) = d + \left\lfloor \frac{x-1}{w_{step}} \right\rfloor h_{step} \quad (1)$$

The models for the different types of layers consist of several stacked step or linear functions forming multi-dimensional planes. For a convolution layer with a given filter size, e.g.  $3 \times 3$  or  $5 \times 5$ , the dimensions of such a model are the number of input channels  $c_{in}$ , the number of filters or output channels  $k$ , and the width and height of the input ( $w_{in}$  and  $h_{in}$ ). The plane in Figure 4 (a) shows a two-dimensional intersection of such a model with a fixed input size of  $w_{in} = h_{in} = 64$  and a varying number of input channels and filters. Such a two-dimensional model can be represented using Equation (2) with the number of input channels  $c_{in}$  and the number of filters  $k$ . The constant terms  $d$  and  $h_{step}$  expand to step functions where  $f_{step,1}(k)$  describes the step function in the dimension of  $k$  at  $c_{in} = 1$  and  $f_{step,2}(k)$  the changes in the step height with an increasing number of filters  $k$ .

$$f_{step}(c_{in}, k) = \underbrace{f_{step,1}(k)}_d + \left\lfloor \frac{c_{in}-1}{w_{step}} \right\rfloor \underbrace{f_{step,2}(k)}_{h_{step}} \quad (2)$$

Figure 4 (b) shows the same intersection plane but modeled with a basic Random Forest Regressor as a representation for statistical or machine-learning-based models.



**FIGURE 5.** Two-dimensional intersection measured on a Jetson Nano with an input size of  $16 \times 16$  (a) and an input size of  $128 \times 128$  (b). Both have equal step sizes in the dimensions of the number of filters and the number of input channels. The noise visible on the surface in (a) is more pronounced due to the different scales.

These models tend to smooth the sharp steps and introduce deviations, although the effect is exaggerated in Figure 4 (b). This leads to a wrong selection of the pruning targets and eventually to a non-optimal CNN on the selected target device. For such statistical or machine-learning-based models, we can utilize Blackthorn to generate an abstract model on top to identify the step widths and, thus, the Pareto-optimal points we are looking for. As we are not interested in latency estimation at this stage, we can ignore the potential loss in accuracy.

Although the latency estimation models have multiple dimensions, we can ignore some to simplify the pruning target estimation. The input width and height are properties of the network architecture and the size of the input image, and thus are outside the scope of pruning. Additionally, the abstract models do not need to consider those dimensions for most hardware devices, as the step size is independent of the input width and height. In Figure 5, we show two 2-dimensional slices for an input size of  $16 \times 16$  and  $128 \times 128$  measured on an Nvidia Jetson Nano. Both have an equal step size of 8 for the number of filters and 32 for the number of input channels. Thus, we only have to take two dimensions ( $c_{in}$  and  $k$ ) into account for optimization.

In the following sections, we present different optimization strategies for pruning. Although all strategies add additional complexity to the pruning algorithms, the computational overhead and, thus, the potential increase in runtime, is negligible. Pruning is dominated by the finetuning step, which is done after each iteration of weight selection and weight removal. Even for a relatively small architecture like ResNet18, finetuning on CIFAR-10 takes around 2 hours, while weight selection and weight removal require only a couple of milliseconds.

### 1) Clipping

Clipping of pruned filters removes any additional pruning beyond the target points provided by the platform models and

is defined by

$$k_p = \left\lfloor \frac{\tilde{k}_p}{w_{step}} \right\rfloor w_{step} \quad (3)$$

with  $k_p$  being the number of filters after Clipping,  $\tilde{k}_p$  the number of filters after running the pruning algorithm, and  $w_{step}$  the platform-dependent step width.

This definition implies that Clipping is an optimizer for accuracy. The resulting neural network always has the same latency, or a slightly larger latency in case the steps are not entirely flat, as filters are solely added to a network, compared to regular pruning. Thus, the total amount of pruning is reduced. For the same reason, the memory footprint of the resulting network is always larger. On the other hand, increasing the number of available filters is beneficial for accuracy. Figure 6 (a) illustrates the effect of clipping on a flat step function.

Clipping directly acts on the output of weight selection (Figure 2 (b)). Since Clipping reduces the number of weights to be removed, no interaction with weight selection or importance computation is necessary, and thus, we do not have to modify existing algorithms. Figure 3 (d) illustrates how Clipping integrates into the pruning workflow.

## 2) Stacking

In contrast to Clipping, Stacking increases the pruning of each layer to a target point and is defined by

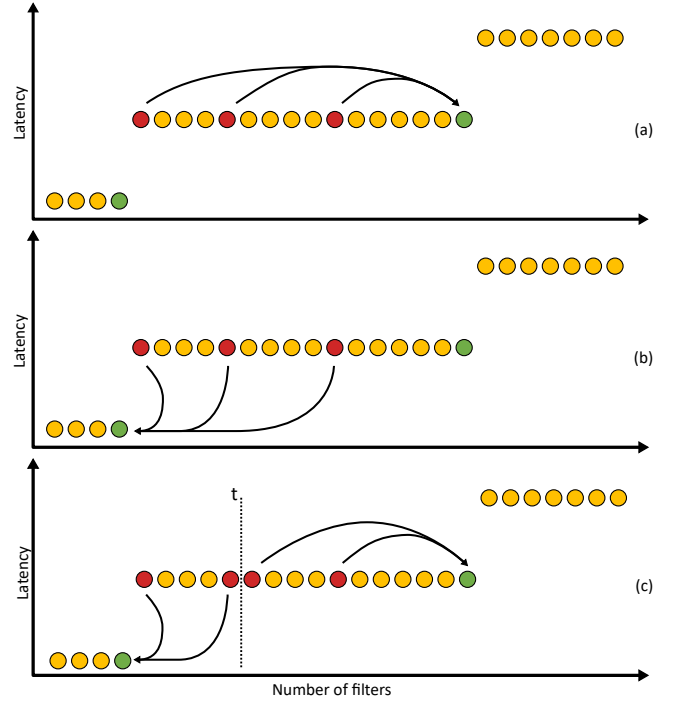
$$k_p = \left\lceil \frac{\tilde{k}_p}{w_{step}} \right\rceil w_{step} \quad (4)$$

As a result, Stacking is an optimizer for latency, and the opposite of Clipping, as filters are never removed from the pruning mask, i.e., the number of pruned filters always increases or stays the same compared to standard pruning. Thus, the resulting CNN is always faster and smaller compared to regular pruning, with the downside of sacrificing accuracy. Figure 6 (b) illustrates the effect of Stacking.

Compared to Clipping, Stacking has more impact on the original pruning algorithm as it also requires modifications to the weight selection or importance computation algorithms (Figure 2 (b)). After an initial weight selection, additional weights have to be tagged for removal. Figure 3 (e) shows Stacking as part of the pruning workflow with the required feedback to the weight selection stage. Additionally, we only apply Stacking when the target number of channels is above the first step and stick to the pruning target otherwise, e.g., for a step with of 32 all pruning targets  $32 < x < 63$  are reduced to 32 and pruning targets  $x < 32$  are kept. This is to avoid pruning layers down to a single channel, which has a significant impact on the accuracy.

## 3) Rounding

Rounding is a combination of Clipping and Stacking with a threshold  $t$  for balancing between optimizing for latency or accuracy (Equation 5). The threshold  $t$  is in the range



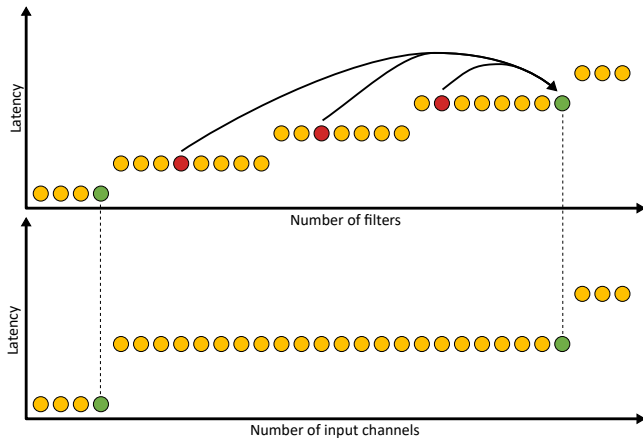
**FIGURE 6.** Overview of the presented optimization strategies: Clipping (a), Stacking (b), and Rounding (c). Layer sizes as a result of standard pruning (highlighted in red) are moved towards optimal points (green).

$[0, 1]$ , with  $t = 0$  being equal to Clipping and  $t = 1$  being equal to Stacking, and is a single hyperparameter for the entire network. Thus, to some layers of a CNN, Clipping will be applied, while others will be optimized using Stacking, combining the advantages of both strategies. Figure 6 (c) shows an example of Rounding with an arbitrary threshold.

$$k_p = \begin{cases} \left\lfloor \frac{\tilde{k}_p}{w_{step}} \right\rfloor w_{step} & \text{if } \frac{\tilde{k}_p \bmod w_{step}}{w_{step}} \geq t \\ \left\lceil \frac{\tilde{k}_p}{w_{step}} \right\rceil w_{step} & \text{otherwise} \end{cases} \quad (5)$$

## B. JOINT PRUNING TARGETS

As noted in Section IV-A, we have to consider the number of input channels and the number of filters for finding optimal pruning targets, requiring an additional constraint for Clipping and Stacking. Although pruning only reduces the number of filters  $k$ , the number of filters of one layer equals the number of input channels of the next layer ( $c_{in}^n = k^{n-1}$  for layer  $n$ ). For convenience, we use the term  $w_{step}(c_{in})$  as a shorthand for the step width in the dimension of the number of input channels and  $w_{step}(k)$  for the number of filters, respectively. Since the step widths usually do not match in both dimensions, the optimization process has to include  $w_{step}(c_{in})$  of the next layer to utilize the available hardware resources optimally. Thus,  $k_p$  has to additionally fulfil Equation (6) and  $w_{step}$  in Equations (3)-(5) becomes  $w_{step}(k)$ . This constraint is independent of the use of the *floor* or *ceil* function, as both cover all Pareto-optimal points.



**FIGURE 7.** Clipping with the additional constraint matching the number of input channels of the next layer with Equation (7) satisfied. The green dots show the Pareto-optimal points after joint optimization.

$$k_p = \left\lfloor \frac{\tilde{k}_p}{w_{step}(c_{in})} \right\rfloor w_{step}(c_{in}) \quad (6)$$

Depending on the relation between  $w_{step}(k)$  and  $w_{step}(c_{in})$ , we can distinguish two scenarios and assign most devices to one of the two resulting types.

#### 1) Scenario A

In the case where one step width is a divisor of the other, the larger step width can be used as a Pareto-optimal point. More generally, it requires that the Least Common Multiple (LCM) of a series of values  $x$  equals the maximum of that series (Equation (7)). The reasoning behind this is that every Pareto-optimal point in the dimension with the larger step width is always a Pareto-optimal point in all the other dimensions. Thus, we can simplify the selection of the pruning targets.

$$lcm(x) = \max(x) \implies \max(x) \text{ is Pareto-optimal} \quad (7)$$

As such, we can set  $w_{step} = lcm(w_{step}(k), w_{step}(c_{in}))$  to account for the additional constraint in Equation (6) and directly use the Equations provided for Clipping, Stacking, and Rounding. Figure 7 shows one example of Clipping for Scenario A with a step width of 24 for the number of filters and 8 for the number of input channels. In our results, we assign devices fulfilling Equation (7) to Type A.

#### 2) Scenario B

When Equation (7) does not hold, e.g., the step widths are 24 and 32 and  $lcm(24, 32) = 96$ , we cannot select a single point for each step plane as  $n \cdot lcm(w_{step}(k), w_{step}(c_{in}))$  would result in a highly coarse and, depending on the CNNs to be pruned, not feasible set of pruning targets. When the total number of filters in a layer is close to such a target or even lower, using the  $lcm(w_{step}(k), w_{step}(c_{in}))$  leaves only one pruning option or no pruning possibility, respectively. These devices will be categorized as Type B.

In this scenario, we still prioritize common multiples of  $w_{step}(k)$  and  $w_{step}(c_{in})$ , but fall back to omitting the additional constraint when not feasible. We then optimize only in the dimension with the larger step width, which usually also has a larger step height, to increase the effectiveness of our method.

## V. RESULTS

### A. EXPERIMENTAL SETUP

For our experiments, we extend the Torch-Pruning framework [8] with our hardware-aware optimizations. Our experiments cover different pruning strategies, target sparsities, network architectures, datasets, and embedded devices.

For the pruning strategies, we cover local pruning and global pruning. In local pruning, the sparsity target of each layer is the same, and the importance is calculated and ranked for each layer individually. Global pruning, on the other hand, ranks the importance of all filters across all layers, which typically leads to different compression ratios for each layer. In our local pruning experiments, we use the Magnitude Importance to rank and select the filters for removal. This does not limit the generality of the results, as the importance criteria only change how filters are chosen, not the number of filters to remove. Thus, the latency is not affected. The difference in accuracy between various importance criteria can be compensated during fine-tuning. For global pruning, we use the more sophisticated LAMP criterion [41]. Global pruning changes the sizes of the layers relative to each other. Thus, it is more sensitive to weight ranking. Additionally, we set the minimal layer size to 8 to avoid practically eliminating a layer (setting the number of channels to 1), which we found has a significant impact on accuracy. A possible accuracy impact is consistent across all experiments using the same importance criteria.

We ran experiments on ResNet18 and ResNet34, representing two sizes of a single network architecture, and on MobileNetV3-large, having a more complex architecture. We used three datasets of increasing difficulty — FashionMNIST, CIFAR-100, and Food101 — also representing different input shapes for the CNN. The images in FashionMNIST are  $28 \times 28$  grayscale images, CIFAR-100 uses  $32 \times 32$  RGB images, and Food101 consists of RGB images of different sizes, which are scaled to  $224 \times 224$  for training and inference.

Finally, we evaluate the latencies and accuracy on an Nvidia Jetson Nano, an Nvidia Orin Nano 8GB, and a Hailo-8 accelerator. The Nvidia devices represent embedded GPU platforms based on different microarchitectures, where the Jetson Nano (based on the Maxwell architecture) lacks the fast Tensor Cores that the Orin Nano has (Ampere architecture). The Hailo-8 is a low-power PCIe accelerator card connected to an Intel Core i7-10700K with 16GB RAM for our experiments. We used the provided tools to compile the networks for the hardware platforms, specifically TensorRT for Nvidia devices and Hailo-RT for the Hailo-8, primarily on the default settings, except for the Hailo-RT compilation,

Device	step width $c_{in}$	step width $k$	Type
Nvidia Jetson Nano	8	32	A
Nvidia Orin Nano 8GB	16	64	A
Nvidia Xavier	8	64	A
NXP i.MX 8 Plus	24	32	B
Hailo 8	16	32	A

**TABLE 1.** Step widths and device type for different embedded hardware platforms.

Standard Pruning	Clipping	Stacking	Rounding
51	64	32	32/64
22	32	22	22/32

**TABLE 2.** Comparison of the number of output channels after pruning. Increasing the pruning ratio to a point where the layer sizes get smaller than the step size, the layer sizes get more similar, leading to a smaller gap between the latency results of our optimization methods.

where we set the compiler optimization level to maximum. As the Hailo-8 operates on 8-bit integers by default, all results also include a potential loss due to quantization.

We ran each experiment on five independently pretrained networks, leading to a total of 560 experiments for each of our optimization methods (Clipping, Stacking, and Rounding).

### B. 2D PRUNING TARGETS ON EMBEDDED HARDWARE

As described in Section IV-B, hardware platforms differ in the step width in the dimensions of  $c_{in}$  and  $k$ . Depending on the relation between both step widths (Equation (7)), we can assign each hardware platform to one of the categories. Table 1 shows the step widths and corresponding categories for different embedded platforms. These values are then used as input to determine the pruning targets for Clipping, Stacking, and Rounding.

### C. PRUNING OPTIMIZATION FOR EMBEDDED DEVICES

Figure 8 (left) shows the overall results averaged across all experiments comparing our optimization methods with standard pruning for both hardware platforms. As the results for the Nvidia Jetson devices are similar (Figure 9), we combined them into a single column to increase the readability. As expected, Clipping leads to a comparable or slightly higher latency in most of our experiments with an increased accuracy. Stacking significantly improves the latency at the cost of a slight decrease in accuracy. Rounding with a threshold of  $t = 0.33$  favors Clipping over Stacking and generally slightly reduces the actual pruning ratio. However, the latency gains due to Stacking overcompensate the potential latency increase of Clipping, resulting in an improved latency while maintaining a better accuracy compared to standard pruning. Across all experiments, our optimization methods appear to favor the Jetson devices, as the possible improvements are more substantial and more consistent, as indicated by the smaller error bars. Comparing the results for local pruning

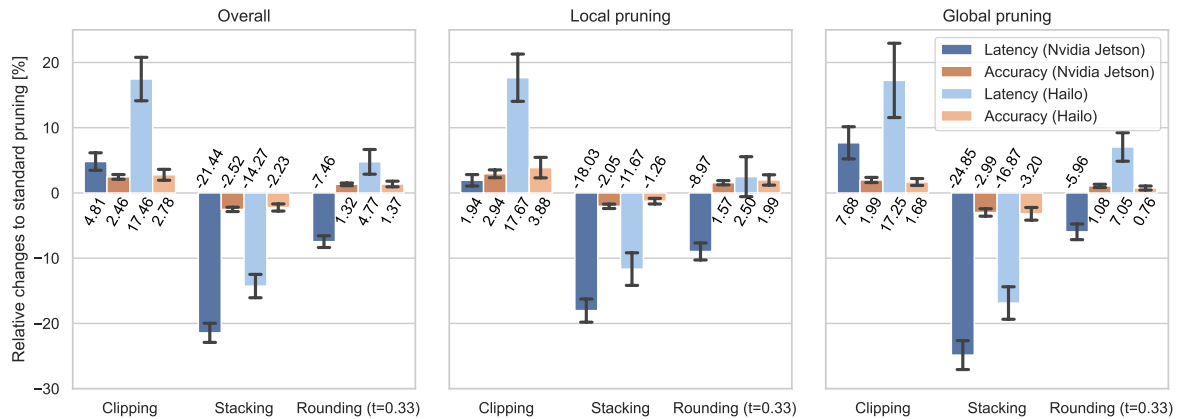
in Figure 8 (middle) and global pruning in Figure 8 (right), local pruning is more consistent and responds better to our optimizations, e.g., Clipping on the Jetson devices leads to a much smaller increase in latency using local pruning instead of global pruning. The main contributing factor is that when global pruning optimizes towards an extremely unbalanced pruning, the data transfer caused by the many unpruned layers adds more overhead than in the case of all layers pruned equally.

Table 3 and Table 4 contain more detailed results of our experiments. For all datasets, each network is trained five times independently, and the resulting networks are pruned and fine-tuned towards pruning targets of 0.2, 0.4, 0.6, and 0.8 for standard pruning as well as Clipping, Stacking, and Rounding. For rounding, we set the threshold  $t = 0.33$ . Both tables also include the latencies of the pruned networks evaluated on an Nvidia Jetson Nano, an Nvidia Orin Nano 8GB, and a Hailo-8 accelerator, together with the actual pruning ratio after applying our optimizations. Using local pruning, the pruning ratio for Clipping is increased by an average of 6.5 percentage points, Stacking reduces the pruning ratio by 4.6 percentage points, and Rounding increases the pruning ratio by only 2.1 percentage points. Global pruning stays much closer to standard pruning with differences of +4.2, -4.5, and +1.8 percentage points.

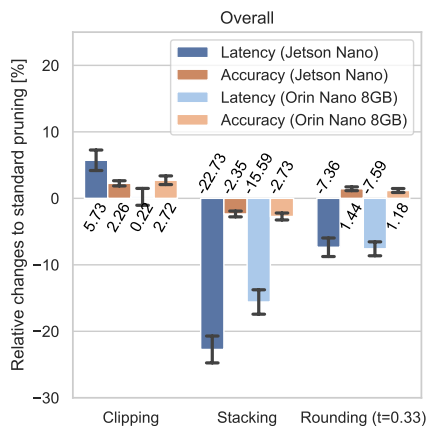
Figure 10 presents a more detailed evaluation of the influences of target pruning ratio, network architecture, and dataset complexity for local pruning. Thus, we can make a few observations. First, the latency gap between the different pruning methods gets smaller with an increasing pruning target (Figure 10 (a)). The main contributing factor is that the number of output channels of the individual layers becomes more similar as the pruning increases, reaching a point where the number of output channels is smaller than the step width. Table 2 shows a short example. The differences in accuracy, on the other hand, become more pronounced as each individual channel becomes more important when the total number decreases, thereby contributing more to the total accuracy of a neural network.

A second observation is that the differences in accuracy get bigger when increasing the complexity of the dataset (Figure 10 (b)). Our optimization methods have a minimal impact on the accuracy of the neural networks on the simple FashionMNIST dataset, whereas the Food101 dataset shows significant differences. This is expected since complex datasets require larger networks to reach a high accuracy. The latency differences are independent of the dataset. The latency mainly depends on the network size, which remains constant across different datasets, and the data that is moved through the DNN. Since small input images require adjustments in downsampling to maintain a feature map size that is sufficiently large, the data moved through the network is comparable despite the increased input size.

Comparing the results on the different network architectures shows that the latency differences on MobileNetV3 are smaller compared to both ResNet architectures (Figure 10



**FIGURE 8.** Comparison of the results of Clipping, Stacking, and Rounding (with a threshold of  $t = 0.33$ ) relative to standard pruning averaged across all our experiments (left), for local pruning (middle), and for global pruning (right). The error bars represent the standard deviation across the contributing experiments.



**FIGURE 9.** Comparison of the results of Clipping, Stacking, and Rounding (with a threshold of  $t = 0.33$ ) relative to standard pruning on the Nvidia Jetson devices. The error bars represent the standard deviation across the contributing experiments.

(c)), independent of the target device. This is because MobileNetV3 is, in general, more difficult to prune than any ResNet architecture due to its architectural differences. At 90% pruning, the latency of ResNet34 is approximately 20% of its initial latency, whereas MobileNetV3 still has a latency of 65%.

Across all experiments and subplots in Figure 10, it is again clearly visible that the two Nvidia devices respond better to our optimization methods, which is prominently visible in the large latency increase for Clipping.

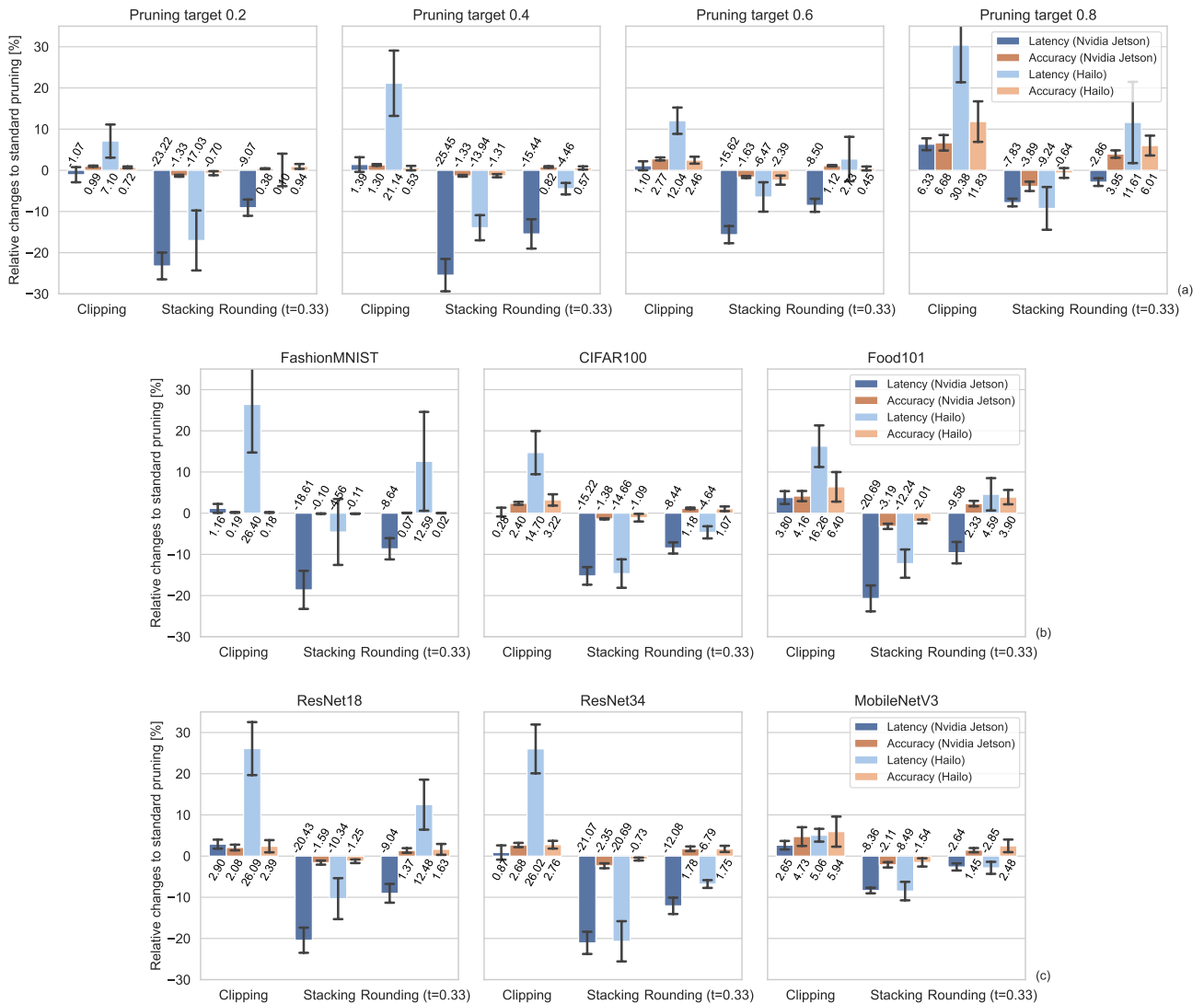
## VI. CONCLUSION AND FUTURE WORK

In this work, we presented optimization strategies for pruning and targeting embedded devices to maximize accuracy, to minimize latency, or to find a trade-off between the two. We used latency estimation models to extract the hardware-specific, Pareto-optimal pruning targets and

demonstrated our meta-method by extending an existing pruning framework, providing experiments for two embedded GPUs (Nvidia Jetson Nano and Orin Nano 8GB) and an external NPU platform (Hailo-8). Across all experiments, including pruning strategies, datasets, and network architectures, we achieved an average increase in accuracy of 2.46% at the cost of a latency increase of 4.81% for clipping on Nvidia Jetson devices compared to standard pruning. For Stacking, we achieved a latency gain of 18.03% while reducing accuracy by only 2.05%. Finally, rounding decreases the latency by 8.97% and also increases the accuracy by 1.57%. Our results show that significant performance improvements can be gained when optimizing pruning for a specific target hardware platform, without manually modifying the architecture of a CNN.

Although our meta-method is independent of the hardware platform, we can only achieve improvements compared to standard pruning when the relation between layer size and latency follows a step function to select Pareto-optimal points. This is usually the case for devices with parallel computation, such as GPUs, accelerators, and Field Programmable Gate Array (FPGA) implementations using systolic arrays. We demonstrated our method on two different GPU architectures and one NPU. Embedded CPU devices and microcontrollers typically lack vast parallel units and show a more linear relation, where our method cannot improve latency or accuracy. In addition, our optimization method relies on the estimation models to accurately represent the relation between the inference time of a layer and its dimensions, i.e., the step size. Still, we do not depend on the accuracy of latency estimation.

As a next step, we plan to extend our work to include estimation-based latency predictions, enabling a fully automatic pruning workflow for embedded devices towards a target latency without requiring access to the platform. In combination with our optimization methods, this can significantly simplify the optimization and improve the utilization of embedded devices without additional overhead.



**FIGURE 10.** Detailed comparison of the results of Clipping, Stacking, and Rounding illustrating the effect of different pruning ratios (a), different datasets/input sizes (b), and different network architectures (c) when using local pruning. The error bars represent the standard deviation across the contributing experiments.

**REFERENCES**

- [1] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, Feb. 2021.
- [2] Y. Cai, T. Luan, H. Gao, H. Wang, L. Chen, Y. Li, M. A. Sotelo, and Z. Li, "YOLOv4-5D: An effective and efficient object detector for autonomous driving," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–13, 2021.
- [3] L. Chen, S. Lin, X. Lu, D. Cao, H. Wu, C. Guo, C. Liu, and F.-Y. Wang, "Deep neural network based vehicle and pedestrian detection for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 6, pp. 3234–3246, Jun. 2021.
- [4] M. A. Hoffmann and R. Lasch, "Tackling industrial downtimes with artificial intelligence in data-driven maintenance," *ACM Comput. Surv.*, vol. 56, no. 4, pp. 1–33, oct 2023. [Online]. Available: <https://doi.org/10.1145/3623378>
- [5] H. Ding, K. Yan, Z. Tu, and P. Wang, "Exploring a universal training method for medical image classification," in *Proceedings of the 6th International Conference on Medical and Health Informatics, ser. ICMHI '22*. New York, NY, USA: Association for Computing Machinery, 10 2022, p. 1–8. [Online]. Available: <https://doi.org/10.1145/3545729.3545731>
- [6] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: [https://proceedings.neurips.cc/paper\\_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf)
- [7] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7873–7882.
- [8] G. Fang, X. Ma, M. Song, M. Bi Mi, and X. Wang, "Depgraph: Towards any structural pruning," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2023, pp. 16091–16101.
- [9] C. Gong, Y. Chen, Y. Lu, T. Li, C. Hao, and D. Chen, "Vecq: Minimal loss dnn model compression with vectorized weight quantization," *IEEE Transactions on Computers*, pp. 1–1, 2020.
- [10] B. Singh, D. Toshniwal, and S. K. Allur, "Shunt connection: An intelligent skipping of contiguous blocks for optimizing MobileNet-v2," *Neural Networks*, vol. 118, pp. 192–203, oct 2019.
- [11] V. Radu, K. Kaszyk, Y. Wen, J. Turner, J. Cano, E. J. Crowley, B. Franke, A. Storkey, and M. O’Boyle, "Performance aware convolutional neural

- network channel pruning for embedded gpus,” in 2019 IEEE International Symposium on Workload Characterization (IISWC), 2019, pp. 24–34.
- [12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *CoRR*, vol. abs/1608.08710, 2016.
- [13] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” *CoRR*, vol. abs/1708.06519, 2017. [Online]. Available: <http://arxiv.org/abs/1708.06519>
- [14] C. Yang and H. Liu, “Channel pruning based on convolutional neural network sensitivity,” *Neurocomputing*, vol. 507, pp. 97–106, Oct. 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231222009110>
- [15] B. Zhang, A. Davoodi, and Y. H. Hu, “Efficient inference of cnns via channel pruning,” *CoRR*, vol. abs/1908.03266, 2019. [Online]. Available: <http://arxiv.org/abs/1908.03266>
- [16] F. Meng, H. Cheng, K. Li, H. Luo, X. Guo, G. Lu, and X. Sun, “Pruning filter in filter,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 17 629–17 640. [Online]. Available: [https://proceedings.neurips.cc/paper\\_files/paper/2020/file/ccb1d45fb76f7c5a0bf619f979c6cf36-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/ccb1d45fb76f7c5a0bf619f979c6cf36-Paper.pdf)
- [17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, Jun. 2016, pp. 770–778.
- [18] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 4510–4520. [Online]. Available: [http://openaccess.thecvf.com/content\\_cvpr\\_2018/html/Sandler\\_MobileNetV2\\_Inverted\\_Residuals\\_CVPR\\_2018\\_paper.html](http://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html)
- [19] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J.-H. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, and W. Zhang, “Group fisher pruning for practical network compression,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 7021–7032.
- [20] Z. Hou, M. Qin, F. Sun, X. Ma, K. Yuan, Y. Xu, Y. Chen, R. Jin, Y. Xie, and S. Kung, “Chex: Channel exploration for cnn model compression,” in 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2022, pp. 12277–12288. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/CVPR52688.2022.01197>
- [21] C. Heidorn, N. Meyerhöfer, C. Schinabeck, F. Hannig, and J. Teich, “Hardware-aware evolutionary filter pruning,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, A. Orailoglu, M. Reichenbach, and M. Jung, Eds. Cham: Springer International Publishing, 2022, pp. 283–299.
- [22] M. Wess, D. Dallinger, D. Schnöll, M. Bittner, M. Götzinger, and A. Jantsch, “Energy profiling of DNN accelerators,” in *Proceedings of the 26th Euromicro Conference on Digital System Design (DSD)*, Durres, Albania, September 2023. [Online]. Available: <http://jantsch.se/AxelJantsch/papers/2023/MatthiasWess-DSD.pdf>
- [23] C. Heidorn, M. Sabih, N. Meyerhöfer, C. Schinabeck, J. Teich, and F. Hannig, “Hardware-aware evolutionary explainable filter pruning for convolutional neural networks,” *International Journal of Parallel Programming*, Feb. 2024. [Online]. Available: <https://link.springer.com/article/10.1007/s10766-024-00760-5#citeas>
- [24] X. Li, Y. Zhou, Z. Pan, and J. Feng, “Partial order pruning: For best speed/accuracy trade-off in neural architecture search,” in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 9137–9145.
- [25] M. Shen, H. Yin, P. Molchanov, L. Mao, J. Liu, and J. M. Álvarez, “HALP: hardware-aware latency pruning,” *CoRR*, vol. abs/2110.10811, 2021. [Online]. Available: <https://arxiv.org/abs/2110.10811>
- [26] “GitHub - VainF/Torch-Pruning: [CVPR 2023] Towards Any Structural Pruning; LLMs / SAM / Diffusion / Transformers / YOLOv8 / CNNs — github.com,” <https://github.com/VainF/Torch-Pruning>, [Accessed 25-06-2024].
- [27] C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, “Analysis of DAWNbench, a time-to-accuracy machine learning performance benchmark,” *ACM SIGOPS Operating Systems Review*, vol. 53, no. 1, pp. 14–25, 2019.
- [28] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, “DAWNbench: An end-to-end deep learning benchmark and competition,” *NIPS ML Systems Workshop*, 2017.
- [29] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou et al., “Mlperf inference benchmark,” *arXiv preprint arXiv:1911.02549*, 2019.
- [30] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, “Embench: Quantifying performance variations of deep neural networks across modern commodity devices,” in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, ser. EMDL ’19. ACM, 2019, pp. 1–6.
- [31] B. D. Rouhani, A. Mirhoseini, and F. Koushanfar, “Going deeper than deep learning for massive data analytics under physical constraints,” in 2016 International Conference on Hardware/Software Codesign and System Synthesis, 2016, pp. 1–3.
- [32] H. Qi, E. R. Sparks, and A. Talwalkar, “Paleo: A performance model for deep neural networks,” in *Proceedings of the International Conference on Learning Representations*, 2017.
- [33] E. Cai, D. Juan, D. Stamoulis, and D. Marculescu, “Neuralpower: Predict and deploy energy-efficient convolutional neural networks,” *CoRR*, vol. abs/1710.05420, 2017.
- [34] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez, “Previous: A methodology for prediction of visual inference performance on iot devices,” *arXiv e-prints*, p. arXiv:1912.06442, 2019.
- [35] M. Lechner and A. Jantsch, “Blackthorn: Latency estimation framework for CNNs on embedded nvidia platforms,” *IEEE Access*, vol. 9, pp. 110 074–110 084, 2021.
- [36] M. Wess, M. Ivanov, C. Unger, A. Nookala, A. Wendt, and A. Jantsch, “Annette: Accurate neural network execution time estimation with stacked models,” *IEEE Access*, vol. 9, pp. 3545–3556, 2021.
- [37] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, and Y. Liu, “nn-meter: Towards accurate latency prediction of dnn inference on diverse edge devices,” p. 19–23, 3 2022.
- [38] W. Roth, G. Schindler, B. Klein, R. Peharz, S. Tschitschek, H. Fröning, F. Pernkopf, and Z. Ghahramani, “Resource-efficient neural networks for embedded systems,” *J. Mach. Learn. Res.*, vol. 25, no. 1, Jan. 2024.
- [39] X. Wang and W. Jia, “Optimizing edge ai: A comprehensive survey on data, model, and system strategies,” 2025. [Online]. Available: <https://arxiv.org/abs/2501.03265>
- [40] K. Zhu, F. Hu, Y. Ding, W. Zhou, and R. Wang, “A comprehensive review of network pruning based on pruning granularity and pruning time perspectives,” *Neurocomputing*, vol. 626, p. 129382, 2025. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231225000542>
- [41] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin, “A deeper look at the layerwise sparsity of magnitude-based pruning,” *CoRR*, vol. abs/2010.07611, 2020. [Online]. Available: <https://arxiv.org/abs/2010.07611>

## APPENDIX A DETAILED RESULTS

Table 3 and Table 4 show the detailed results of all our experiments for local pruning (Table 3) and global pruning (Table 4).

	Pruning Target	Standard pruning		Clipping			Stacking			Rounding ( $t = 0.33$ )		
		Latency [ms]	Acc [%]	Actual Pruning	Latency [ms]	Acc [%]	Actual Pruning	Latency [ms]	Acc [%]	Actual Pruning	Latency [ms]	Acc [%]
<b>ResNet18 on Jetson Nano</b>												
FashionMNIST	0.2	5.43	95.58	0.12	5.39	95.65	0.27	4.20	95.46	0.17	4.89	95.59
FashionMNIST	0.4	3.53	95.51	0.33	3.55	95.57	0.47	2.59	95.35	0.37	3.14	95.54
FashionMNIST	0.6	1.92	95.21	0.52	1.93	95.37	0.64	1.54	95.04	0.57	1.68	95.29
FashionMNIST	0.8	0.93	94.82	0.73	0.97	95.24	0.83	0.88	94.88	0.77	0.92	94.98
CIFAR100	0.2	5.44	73.56	0.12	5.39	74.73	0.27	4.21	72.60	0.17	4.89	73.97
CIFAR100	0.4	3.53	72.25	0.33	3.55	73.15	0.47	2.60	71.28	0.37	3.15	73.14
CIFAR100	0.6	1.93	69.16	0.52	1.94	70.93	0.64	1.55	67.86	0.57	1.68	69.96
CIFAR100	0.8	0.94	62.91	0.73	1.05	65.36	0.83	0.90	62.09	0.77	0.93	64.81
Food101	0.2	14.94	77.18	0.13	15.35	78.19	0.27	9.79	75.57	0.20	14.05	77.87
Food101	0.4	10.29	74.53	0.33	11.12	75.93	0.47	6.42	72.76	0.40	7.28	75.08
Food101	0.6	5.34	70.92	0.53	5.56	73.15	0.64	4.38	69.41	0.60	5.19	72.09
Food101	0.8	2.98	64.34	0.73	3.08	69.86	0.83	2.75	60.33	0.78	2.95	68.77
<b>Resnet34 on Jetson Nano</b>												
CIFAR100	0.2	9.64	76.34	0.12	9.46	77.34	0.26	7.25	75.62	0.16	8.27	76.74
CIFAR100	0.4	6.36	75.43	0.34	6.09	76.16	0.47	4.33	74.30	0.36	5.38	75.90
CIFAR100	0.6	3.29	72.40	0.53	3.33	74.97	0.64	2.58	71.45	0.57	2.73	73.44
CIFAR100	0.8	1.48	67.28	0.74	1.52	69.46	0.84	1.33	66.46	0.76	1.36	68.70
Food101	0.2	26.61	81.41	0.13	28.86	82.15	0.26	18.07	79.74	0.21	23.97	81.48
Food101	0.4	17.47	79.14	0.34	19.14	80.10	0.47	10.86	77.86	0.40	12.89	79.69
Food101	0.6	8.94	74.95	0.53	9.53	77.14	0.64	6.80	73.62	0.60	8.35	75.99
Food101	0.8	4.31	68.01	0.74	4.92	73.00	0.83	3.75	62.29	0.78	4.29	73.01
<b>ResNet18 on Hailo-8</b>												
FashionMNIST	0.2	0.38	95.50	0.12	0.47	95.64	0.27	0.29	95.38	0.17	0.42	95.55
FashionMNIST	0.4	0.26	95.47	0.33	0.40	95.44	0.47	0.23	95.33	0.37	0.25	95.46
FashionMNIST	0.6	0.17	95.39	0.52	0.20	95.56	0.64	0.18	95.22	0.57	0.20	95.31
FashionMNIST	0.8	0.09	94.77	0.73	0.16	95.17	0.83	0.10	94.78	0.77	0.13	94.88
Food101	0.2	0.77	77.05	0.12	0.89	78.04	0.27	0.55	75.45	0.17	0.88	77.42
Food101	0.4	0.58	74.49	0.33	0.72	75.73	0.47	0.47	72.64	0.37	0.57	74.67
Food101	0.6	0.38	70.88	0.52	0.44	73.02	0.64	0.37	69.46	0.57	0.43	71.89
Food101	0.8	0.21	62.05	0.73	0.35	69.79	0.83	0.18	60.18	0.77	0.26	68.74
<b>Resnet34 on Hailo-8</b>												
CIFAR100	0.2	1.10	76.26	0.12	1.29	77.17	0.26	0.73	75.42	0.16	1.03	76.72
CIFAR100	0.4	0.66	74.95	0.34	0.83	75.82	0.47	0.51	74.14	0.36	0.63	75.70
CIFAR100	0.6	0.33	71.96	0.53	0.39	74.70	0.64	0.29	71.19	0.57	0.30	73.00
CIFAR100	0.8	0.14	65.96	0.74	0.20	69.17	0.84	0.12	66.19	0.76	0.13	68.56
<b>MobileNetV3_Large on Hailo-8</b>												
CIFAR100	0.2	2.30	79.04	0.18	2.30	79.57	0.22	2.29	78.21	0.19	2.27	79.26
CIFAR100	0.4	1.86	76.60	0.35	1.84	77.52	0.43	1.66	75.74	0.39	1.68	76.40
CIFAR100	0.6	1.15	72.72	0.56	1.19	73.31	0.65	1.01	67.93	0.60	1.14	71.93
CIFAR100	0.8	0.54	60.04	0.75	0.62	67.28	0.83	0.48	61.83	0.79	0.55	61.62
Food101	0.2	3.42	78.15	0.18	3.60	78.37	0.22	3.46	78.81	0.19	3.58	80.72
Food101	0.4	2.80	75.21	0.35	3.20	74.20	0.43	2.64	73.93	0.39	2.73	76.63
Food101	0.6	1.95	68.83	0.56	2.05	71.93	0.65	1.73	67.38	0.60	1.81	69.20
Food101	0.8	1.02	53.67	0.75	1.14	69.40	0.83	0.84	51.77	0.79	0.97	60.42
<b>Resnet34 on Orin Nano 8GB</b>												
CIFAR100	0.2	2.18	76.34	0.12	2.02	77.34	0.26	1.79	75.62	0.16	1.87	76.74
CIFAR100	0.4	1.79	75.43	0.34	1.72	76.16	0.47	1.48	74.30	0.36	1.62	75.90
CIFAR100	0.6	1.31	72.40	0.53	1.28	74.97	0.64	1.21	71.45	0.57	1.23	73.44
CIFAR100	0.8	1.01	67.28	0.74	1.05	69.46	0.84	0.94	66.46	0.76	0.97	68.70
Food101	0.2	4.77	81.41	0.13	4.26	82.15	0.26	3.09	79.74	0.21	3.97	81.48
Food101	0.4	3.69	79.14	0.34	3.49	80.10	0.47	2.39	77.86	0.40	2.57	79.69
Food101	0.6	2.09	74.95	0.53	2.01	77.14	0.64	1.83	73.62	0.60	1.90	75.99
Food101	0.8	1.32	68.01	0.74	1.41	73.00	0.83	1.20	62.29	0.78	1.25	73.01
<b>MobileNetV3_Large on Orin Nano 8GB</b>												
CIFAR100	0.2	2.04	80.23	0.18	2.04	80.86	0.22	1.89	79.63	0.19	2.03	80.27
CIFAR100	0.4	1.78	77.80	0.35	1.84	79.33	0.43	1.65	77.30	0.38	1.76	78.28
CIFAR100	0.6	1.55	74.25	0.56	1.54	75.97	0.65	1.41	72.04	0.60	1.47	74.87
CIFAR100	0.8	1.25	66.27	0.74	1.27	70.20	0.83	1.14	64.95	0.79	1.19	67.57
Food101	0.2	2.60	82.16	0.18	2.62	82.78	0.22	2.29	80.76	0.19	2.60	82.76
Food101	0.4	2.26	79.10	0.35	2.35	80.84	0.43	2.06	78.13	0.39	2.14	80.79
Food101	0.6	1.89	71.38	0.56	1.96	74.10	0.65	1.73	70.43	0.60	1.81	71.87
Food101	0.8	1.55	60.26	0.75	1.68	72.38	0.83	1.47	56.47	0.79	1.55	63.01

**TABLE 3.** Summary of the results of our experiments comparing standard pruning with our different optimization strategies using local pruning. The accuracies reported are always the median values of five independent pruning runs. As the Hailo-8 requires INT8 quantization, the reported accuracies are slightly lower compared to the other devices.

	Pruning Target	Standard pruning		Clipping			Stacking			Rounding ( $t = 0.33$ )		
		Latency [ms]	Acc [%]	Actual Pruning	Latency [ms]	Acc [%]	Actual Pruning	Latency [ms]	Acc [%]	Actual Pruning	Latency [ms]	Acc [%]
<b>ResNet18 on Jetson Nano</b>												
FashionMNIST	0.2	6.25	95.58	0.19	6.21	95.64	0.21	6.03	95.64	0.19	6.18	95.60
FashionMNIST	0.4	4.55	95.60	0.36	4.54	95.63	0.43	3.29	95.27	0.40	4.35	95.52
FashionMNIST	0.6	2.90	95.14	0.53	3.15	95.22	0.66	1.94	94.91	0.57	2.84	95.27
FashionMNIST	0.8	1.37	94.73	0.74	1.51	94.93	0.87	0.90	94.51	0.77	1.18	94.82
CIFAR100	0.2	6.22	73.40	0.18	6.26	74.48	0.24	5.62	73.09	0.19	6.21	73.64
CIFAR100	0.4	4.71	72.57	0.37	4.79	72.64	0.47	3.61	71.53	0.39	4.38	73.07
CIFAR100	0.6	3.03	70.14	0.54	3.27	71.75	0.66	1.88	68.95	0.58	2.88	71.00
CIFAR100	0.8	1.41	62.46	0.73	1.56	66.74	0.87	0.90	59.52	0.76	1.21	65.63
Food101	0.2	15.16	77.22	0.18	15.61	77.64	0.24	10.18	75.55	0.19	14.62	77.73
Food101	0.4	11.85	76.78	0.36	12.26	77.46	0.44	7.43	74.88	0.39	11.08	77.06
Food101	0.6	7.16	75.22	0.54	9.40	76.50	0.66	4.63	72.24	0.56	9.15	76.56
Food101	0.8	4.28	66.19	0.73	6.65	72.74	0.86	2.86	59.44	0.76	4.19	69.50
<b>Resnet34 on Jetson Nano</b>												
CIFAR100	0.2	11.56	76.41	0.17	11.59	76.52	0.25	9.69	75.75	0.18	11.41	76.66
CIFAR100	0.4	7.74	75.34	0.37	8.05	75.51	0.44	5.26	75.16	0.38	7.59	75.42
CIFAR100	0.6	4.95	71.85	0.53	5.47	74.48	0.67	2.60	70.91	0.57	4.56	73.21
CIFAR100	0.8	2.24	67.03	0.73	3.20	69.86	0.87	1.30	64.26	0.76	1.91	69.98
Food101	0.2	26.24	81.06	0.17	27.59	81.59	0.24	18.00	79.71	0.18	25.25	81.31
Food101	0.4	20.05	78.94	0.36	21.73	80.32	0.45	11.86	75.84	0.38	18.40	79.96
Food101	0.6	11.59	76.66	0.52	13.47	78.90	0.64	6.57	73.98	0.60	10.86	78.06
Food101	0.8	6.46	71.57	0.73	9.22	76.00	0.86	3.68	62.53	0.75	5.82	73.00
<b>ResNet18 on Hailo-8</b>												
FashionMNIST	0.2	0.40	95.57	0.19	0.40	95.59	0.21	0.39	95.60	0.19	0.40	95.59
FashionMNIST	0.4	0.38	95.58	0.36	0.39	95.58	0.43	0.34	95.24	0.40	0.37	95.48
FashionMNIST	0.6	0.30	95.07	0.53	0.36	95.19	0.66	0.24	94.91	0.57	0.31	95.16
FashionMNIST	0.8	0.18	94.74	0.73	0.23	94.80	0.87	0.14	94.53	0.74	0.20	94.81
Food101	0.2	0.71	75.75	0.18	0.88	76.42	0.24	0.61	73.35	0.19	0.88	76.62
Food101	0.4	0.67	74.42	0.36	0.89	75.81	0.44	0.44	72.55	0.39	0.78	75.30
Food101	0.6	0.57	73.03	0.54	0.82	75.43	0.66	0.32	69.68	0.56	0.69	74.94
Food101	0.8	0.28	65.69	0.73	0.59	71.99	0.87	0.18	54.61	0.76	0.36	66.73
<b>Resnet34 on Hailo-8</b>												
CIFAR100	0.2	1.07	76.34	0.17	1.13	76.23	0.25	0.89	75.14	0.18	1.12	76.43
CIFAR100	0.4	0.82	74.80	0.37	0.75	75.57	0.44	0.76	74.28	0.38	0.76	74.95
CIFAR100	0.6	0.48	71.68	0.53	0.57	74.12	0.67	0.38	69.92	0.57	0.49	72.89
CIFAR100	0.8	0.29	66.95	0.73	0.34	69.03	0.87	0.22	62.10	0.76	0.33	68.90
<b>MobileNetV3_Large on Hailo-8</b>												
CIFAR100	0.2	2.44	78.07	0.18	2.49	78.83	0.22	2.31	78.01	0.19	2.43	77.80
CIFAR100	0.4	1.84	75.63	0.38	1.95	76.19	0.42	1.67	75.54	0.39	1.90	76.33
CIFAR100	0.6	1.11	71.25	0.57	1.17	72.13	0.64	0.95	69.65	0.59	1.12	71.06
CIFAR100	0.8	0.54	63.74	0.77	0.62	66.45	0.82	0.43	58.60	0.78	0.60	63.79
Food101	0.2	3.89	78.03	0.19	3.98	77.70	0.22	3.64	78.01	0.20	3.85	76.26
Food101	0.4	3.06	79.73	0.38	3.10	78.83	0.42	2.91	76.11	0.39	3.09	79.19
Food101	0.6	2.43	76.06	0.58	2.54	77.96	0.64	2.08	76.35	0.59	2.40	77.68
Food101	0.8	1.30	71.34	0.77	1.48	72.93	0.84	1.17	64.61	0.78	1.46	73.01
<b>Resnet34 on Orin Nano 8GB</b>												
CIFAR100	0.2	2.37	76.41	0.17	2.21	76.52	0.25	2.00	75.75	0.18	2.19	76.66
CIFAR100	0.4	1.74	75.34	0.37	1.71	75.51	0.44	1.43	75.16	0.38	1.66	75.42
CIFAR100	0.6	1.53	71.85	0.53	1.41	74.48	0.67	1.11	70.91	0.57	1.36	73.21
CIFAR100	0.8	1.04	67.03	0.73	1.06	69.86	0.87	0.99	64.26	0.76	0.97	69.98
Food101	0.2	4.48	81.06	0.17	4.13	81.59	0.24	3.14	79.71	0.18	4.06	81.31
Food101	0.4	3.48	78.94	0.36	3.50	80.32	0.45	2.36	75.84	0.38	3.17	79.96
Food101	0.6	2.56	76.66	0.52	2.92	78.90	0.64	1.79	73.98	0.59	2.37	78.06
Food101	0.8	1.77	71.57	0.73	2.26	76.00	0.86	1.34	62.53	0.75	1.47	73.00
<b>MobileNetV3_Large on Orin Nano 8GB</b>												
CIFAR100	0.2	2.08	79.84	0.18	2.00	80.17	0.22	1.96	79.33	0.19	1.97	79.45
CIFAR100	0.4	1.85	77.16	0.38	1.79	77.88	0.42	1.71	77.52	0.38	1.76	77.31
CIFAR100	0.6	1.60	73.98	0.57	1.54	74.22	0.64	1.43	71.63	0.59	1.45	73.08
CIFAR100	0.8	1.34	66.33	0.77	1.36	68.91	0.84	1.15	61.27	0.78	1.22	66.12
Food101	0.2	2.75	83.07	0.19	2.68	83.16	0.22	2.54	82.60	0.20	2.61	83.04
Food101	0.4	2.49	82.01	0.38	2.48	82.30	0.42	2.23	80.82	0.39	2.34	82.01
Food101	0.6	2.25	79.94	0.58	2.27	80.53	0.64	1.97	78.77	0.59	2.19	79.88
Food101	0.8	1.99	75.20	0.77	1.99	76.17	0.84	1.52	68.52	0.78	1.76	76.46

**TABLE 4.** Summary of the results of our experiments comparing standard pruning with our different optimization strategies using global pruning. The accuracies reported are always the median values of five independent pruning runs. As the Hailo-8 requires INT8 quantization, the reported accuracies are slightly lower compared to the other devices.



MARTIN LECHNER received the B.Sc. and M.Sc. degrees from the Department of Electrical Engineering, TU Wien, Vienna, Austria, in 201 and 2018, respectively. He is currently pursuing the Ph.D. degree with the Institute for Computer Technology and is part of the Christian Doppler Laboratory for Embedded Machine Learning at TU Wien, Austria. His current research interests include hardware-accelerated machine learning and applications on embedded systems.



AXEL JANTSCH (Fellow, IEEE) received the Dipl.-Ing. and Ph.D. degrees in computer science from TU Wien, Vienna, Austria, in 1987 and 1992, respectively. From 1997 to 2002, he was an Associate Professor with the KTH Royal Institute of Technology, Stockholm. From 2002 to 2014, he was a Full Professor of electronic systems design with KTH. Since 2014, he has been a Professor of systems on chips with the Institute of Computer Technology, TU Wien. He has published five books as an editor and one as an author and over 300 peer-reviewed contributions in journals, books, and conference proceedings. His current research interests include systems on chips, self-aware cyber-physical systems, and embedded machine learning. He has given over 100 invited presentations at conferences, universities, and companies.

...