mathematics

MDPI

Article

AMED: Automatic Mixed-Precision Quantization for Edge Devices

Moshe Kimhi *, Tal Rozen, Avi Mendelson and Chaim Baskin

Computer Science Department, Technion IIT, Haifa 3200003, Israel; tal.rozen@technion.ac.il (T.R.); mendlson@technion.ac.il (A.M.); chaimbaskin@technion.ac.il (C.B.)
* Correspondence: moshekimhi@campus.technion.ac.il

Abstract: Quantized neural networks are well known for reducing the latency, power consumption, and model size without significant harm to the performance. This makes them highly appropriate for systems with limited resources and low power capacity. Mixed-precision quantization offers better utilization of customized hardware that supports arithmetic operations at different bitwidths. Quantization methods either aim to minimize the compression loss given a desired reduction or optimize a dependent variable for a specified property of the model (such as FLOPs or model size); both make the performance inefficient when deployed on specific hardware, but more importantly, quantization methods assume that the loss manifold holds a global minimum for a quantized model that copes with the global minimum of the full precision counterpart. Challenging this assumption, we argue that the optimal minimum changes as the precision changes, and thus, it is better to look at quantization as a random process, placing the foundation for a different approach to quantize neural networks, which, during the training procedure, quantizes the model to a different precision, looks at the bit allocation as a Markov Decision Process, and then, finds an optimal bitwidth allocation for measuring specified behaviors on a specific device via direct signals from the particular hardware architecture. By doing so, we avoid the basic assumption that the loss behaves the same way for a quantized model. Automatic Mixed-Precision Quantization for Edge Devices (dubbed AMED) demonstrates its superiority over current state-of-the-art schemes in terms of the trade-off between neural network accuracy and hardware efficiency, backed by a comprehensive evaluation.

Keywords: deep learning; quantization; CNN; latency optimization

MSC: 68T07

Citation: Kimhi, M.; Rozen, T.; Mendelson, A.; Baskin, C. AMED: Automatic Mixed-Precision Quantization for Edge Devices. *Mathematics* **2024**, *12*, 1810. https://doi.org/10.3390/math12121810

Academic Editor: Xiang Li

Received: 7 April 2024
Revised: 28 May 2024
Accepted: 4 June 2024
Published: 11 June 2024

1. Introduction

Deep neural networks have established themselves as the primary algorithmic solution for a wide array of real-world applications. However, the computational and memory requirements of DNNs are considerable, leading to notable latency and power consumption during both the training and inference processes.

To meet the rapidly increasing demand for algorithmic capabilities in embedded systems, such as autonomous vehicles, drones, and medical devices, prior research has suggested various techniques for reducing these devices' power and energy consumption. Some of the techniques already deployed in different systems are matrix compression [1–4], pruning [5,6], hardware-aware neural architecture search [7–9], and quantization [10–12].

Quantization is a promising and straightforward technique to accelerate neural network architectures because it reduces the model's memory footprint and the computation complexity by a large factor.

The standard metric to evaluate a quantized model's computational complexity is the number of multiply–accumulate (MAC) operations or bit operations (BOPs) it demands. Neither consider other factors such as communication complexity [13], memory

utilization, and inter-layer relations. Using ultra-low bit quantization does not, therefore, consistently improve the chip's overall performance due to communication and memory boundaries [14].

To this end, recent developments in industry [15,16] and academia [17,18] have introduced support for various precision matrix multiplications, leading to a new line of work focusing on mixed-precision (MP) quantization [19–21], that is assigning a different bandwidth to each matrix (i.e., weights and activations).

MP quantization methods focus on either finding the optimal solution by the estimation of the loss manifold of a full-precision model and assess what layers would cause the least steepest change to the local minimum the model reached at one quantization step [19,20]. In this work, we argue that, when quantizing a model *during* the training process, the minimum on this loss manifold could be very different. This realization led us to the conclusion that the quantization should be looked at as a process, and the optimal step depends on the intermediate state of the model, not on a final form. This way, an intermediate quantized model would be easier to train and the quantization loss would be lower, so the trajectory to the final quantized model is smoother.

Hardware-aware methodologies, sometimes coupled with Neural Architectural Search (NAS), have shown promising progress in the field of hardware-aware models. Current work [8,9] directly measures signals from the hardware simulator. Nevertheless, to incorporate hardware constraints into the loss, they neglect the dependencies between layers, that is the latency per layer per precision is a fixed number. This assumption is inaccurate in cases where the memory utilization is high because the communication boundary agrees with the computational one. NAS can provide a solution that is better tailored to the task. Even so, the search space is commonly huge, and the cost of training [22] and the carbon footprint [23] are high.

Our solution, Automatic Mixed-Precision Quantization for Edge Devices (dubbed AMED), is an algorithmic framework that chooses mixed-precision bit allocation per layer by looking at reduced precision as a Markov Decision Process (MDP), using signals directly from the hardware. The procedure is simple to use, not bounded to a specific HW design, and easily fine-tuned by the user to find a good balance between performance and resource consumption, as shown in Figure 1.

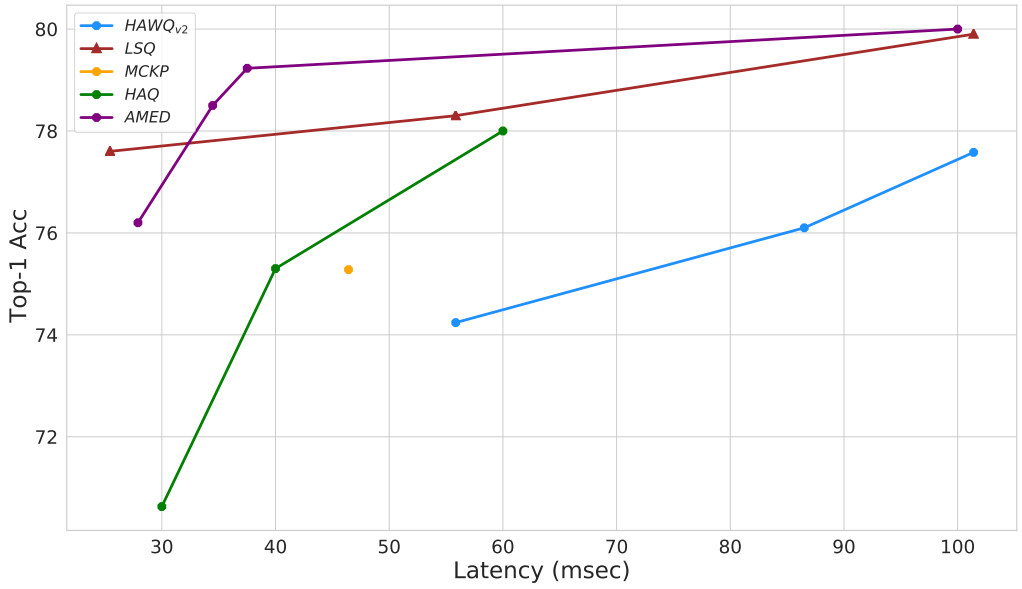

Figure 1. ResNet-50 quantized models on a latency–accuracy plane. Circles are mixed-precision quantization, and triangles are uniform quantization. Our models achieve a better Pareto curve of dominant solutions in the two-dimensional plane for ultra-low precision.

This paper makes the following contributions:

- A novel framework for mixed-precision quantization for DNNs that look at reduced precision as a Markov process.
- A quality score that represents the accuracy–latency trade-off with respect to the hardware constraint. This allows us to create custom-fit solutions for a range of device-specific hardware constraints via direct hardware signals in the training procedure.
- Extensive experiments conducted on different hardware setups with different models on standard image classification benchmarks (CIFAR100, ImageNet). These outperform previous methods in terms of the accuracy–latency trade-off.
- A proposed modular framework, i.e., the sampling method, hardware properties, accelerator simulator, and neural network architecture are all independent modules, making it applicable to any given case.

Our source code, experimental settings, and quantized models are available at https://github.com/RamorayDrake/AMED/, accessed on 28 January 2023.

2. Materials

Multi-Objective Optimization

In a simple learning paradigm, the optimization process attempts to minimize an objective function; the example in this work is a classification model, minimizing the cross-entropy loss, denoted by $\mathcal{L}_{CE}$.

Multi-objective problems can have both positively correlated objectives, as well as negatively correlated objectives (also known as conflicting objectives or adversary objectives). The problem we aim to solve in this paper is the accuracy–latency trade-off of the quantized model, in which one objective is minimizing $\mathcal{L}_{CE}$, while the other is minimizing the latency, denoted by $\mathcal{L}_{lat}$. This idea can be extended to other hardware objectives (such as model size or power consumption); however, these typically correlate with $\mathcal{L}_{lat}$ and are beyond the scope of this work.

Both objectives are conflicting because quantizing our model to a lower representation aims to reduce the latency while increasing the cost of quantization error. Due to these conflicting objectives, multi-objective optimization problems are known to lack optimal solutions with respect to all of the objectives [24]. Typically, there are many optimal (or suboptimal) solutions because one is not comparable with the other. Solution **A** dominates solution **B** ($\mathbf{A} \prec \mathbf{B}$) if all objective values of **A** are better than or equal to the respective objective value of **B**. Dominant solutions define a Pareto-optimal curve in the objective's plane. To solve multi-objective optimization, it is very common to use single-objective approaches or a Pareto approach.

A single-objective approach aggregates the objectives into one term, where the basic aggregation method is the weighted sum vector [25], denoted by α, which encodes the user's priorities over the different objectives. This method is very common when training a DNN, but has the following drawbacks:

- The weighted sum vector α must be determined beforehand and requires a grid search or meta-learning, both of which can be costly and have difficulty converging.
- Not all objectives can be optimized via the same optimization scheme. For example, a gradient-based optimizer cannot be used when only some of the objectives are differentiable.

A Pareto approach is based on sampling methods and finding the Pareto curve by adopting only dominant solutions. The main drawback of this approach in the perspective of a quantization scheme is the computational cost. Each sample needs to first quantize a neural network and train it, before it can evaluate the objectives.

3. Literature Review

Quantized Neural Networks

Quantization is one of the most efficient and commonly used techniques for the acceleration and compression of models. Generally, quantization procedures can be categorized into Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT):

- **PTQ** uses a small calibration set to obtain the optimal quantization parameters without the need or capability to use the entire dataset.
- **QAT** conducts the quantization during the training process and, thus, uses the entire training corpus.

QAT is more appealing when quantizing a model into ultra-low precision because the network is trained to withhold quantization noise [26]. Because we are focusing on ultra-low precision, we use the QAT technique for this work.

Homogeneous quantization quantizes both weights and activations using the same bitwidth throughout the network, scaling from 32-bit to as low as binary [27,28]. Ref. [29] proposes a learnable quantizer with a basis vector that is adaptable to the weights and activations. Refs. [12,30] presents a clipping method that automatically optimizes the quantization scales during model training. In [31], a non-uniform step size was learned as a model parameter so that it became more sensitive to the quantization transition points. Ref. [32] takes this method a step further and proposes a weight regularization algorithm that encourages a sharp distribution for each quantization bin and encourages it to be as close to the target quantized value as possible. Ref. [33] employed a series of hyperbolic tangent functions to approach the staircase function for low-bit quantization gradually.

Recent research, however, has shown that different layers in a DNN model contribute to the performance differently and contain different redundancies. Therefore, the notion of a mixed-precision quantization scheme that assigns a different precision bitwidth to each layer of the DNN can offer better accuracy while compressing the model even further. Nevertheless, determining each layer's precision is quite challenging because of the large search space. In [34], NAS was used for allocating the bitwidth by employing BOPs in the cost function. HAQ [19] leverages reinforcement learning to determine the quantization policy layerwise. Additionally, it takes the hardware accelerator's feedback into account in the architecture design, attaining an optimized solution for deterministic constraints. It fails, however, to find a solution that stochastically reduces latency and power consumption. FBNET [8] combines all possible design choices into a stochastic super-net and approximates the optimal scheme via sampling. These search methods can require a large quantity of computational resources, which scale up quickly with the number of layers.

Therefore, other works attempt to allocate the bitwidth using different methods. Ref. [35] used the first-order Taylor expansion to evaluate the loss sensitivity due to the quantization of each channel, and then adjusted the bitwidth channelwise. Ref. [36] considered each layer's bitwidth as an independent trainable parameter. Ref. [37] allocated differentiable bitwidths to layers, which consequently offers smooth transitions between neighboring quantization bit levels, all while meeting a target computational constraint. HAWQ [20,38] applies mixed-precision quantization using the Hessian information. Power iteration is adopted to compute the top Hessian eigenvalue, which is used to determine which layers are more prone to quantization. It relies on a proxy signal that assumes that all devices benefit from a higher compression rate and a lower number of operations, and does not consider the specific hardware design. Ref. [39] also used the Hessian trace and formulated the mixed-precision quantization as a discrete constrained optimization problem solved by a greedy search algorithm. Ref. [40] differentially learned the quantization parameters for each layer, including the bitwidth, quantization level, and dynamic range. FILM-QNN [21] investigates mixed-precision quantization in the layer scope. This means that each layer's parameters have a different precision. While this has the benefit of being even more efficient, the metadata of which parameters are sent to which multiply and accumulate (MAC) unit are not feasible in most current hardware architectures.

The quantization can also be performed in two different ways. Works such as [19,31,33,36,38] perform uniform quantization in which the width of the quantization bins is a single parameter. A more complex non-uniform quantization allows different bin widths, which reduces the quantization error [29,35].

To perform a standard MAC operation in a fixed-point representation on a given hardware, all of the data (weights and activations) must be equally scaled. This means that, when implementing non-uniform quantization, a lookup table to a higher representation is required. While the communication costs are reduced, the computational and area costs of non-uniform quantization are very high and inefficient.

4. Method

This section introduces our proposed technique for mixed-precision quantization. It leverages the strengths of both approaches described in Section 2. We achieve this by first combining the desired properties of the model into a single, unified objective function. Then, we employ the Pareto optimization approach to efficiently sample solutions that achieve a desirable trade-off between these properties. We define the problem (Section 4.1) and describe our perspective of the quantization process of neural networks (Section 4.2), following which, we describe our MP quantizer (Section 4.3) and our MP simulator (Section 4.4). Finally, we introduce the technique of AMED (Section 4.5).

4.1. Problem Definition

The DNN architecture parameters grouped into $\mathbf{N}$ layers are denoted by θ (Figure 2). The bit-allocation vector is denoted by $\mathbf{A} \in \mathbb{M}^{\mathbf{N}}$, where $\mathbb{M}$ is the number of possible bit allocations. Additionally, a quantized set of model parameters is denoted by θ_A. $\mathbf{A}$ is a temporal vector, denoted by $\mathbf{A}_t$ at time t, and the l-th layer bitwidth at time t is denoted by $\mathbf{A}_t^l$.

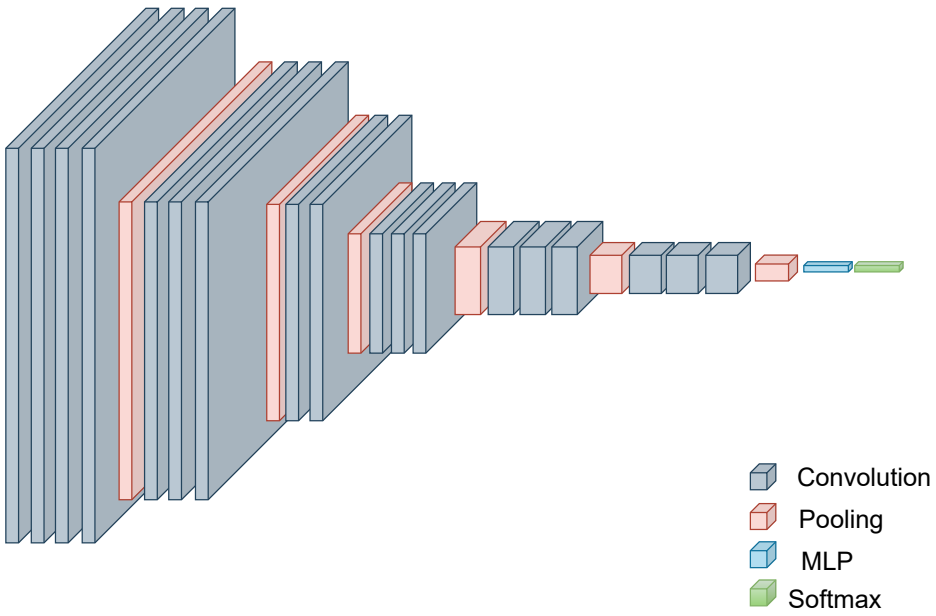

Figure 2. A diagram of DNN architecture activation maps.

Due to the dependence on specific accelerator properties, a differentiable form of the hardware objective (e.g., latency) cannot be directly measured in advance. This renders traditional optimization techniques relying on gradient descent inapplicable. Furthermore, the non-i.i.d. nature of latency across layers adds another layer of complexity. Quantization choices for one layer (e.g., $\mathbf{A}^{l-1}$) can influence the cache behavior of subsequent layers (e.g., l), leading to unpredictable changes in the overall latency. This makes individual layer optimization ineffective.

Motivated by multi-objective optimization approaches that can handle trade-offs between competing goals, we propose encoding user-specified properties of the DNN model (e.g., accuracy, memory footprint, and latency) into a single, non-differentiable objective function. By doing so, we aim to avoid introducing relaxations that might not accurately capture the true behavior of the hardware platform, potentially leading to suboptimal solutions.

We define:

$$\mathbf{Z} = \mathcal{L}_{CE} + \beta \mathcal{L}_{Lat} \tag{1}$$

Since the objective function $\mathbf{Z}$ is non-differentiable with respect to the model parameters, we employ a nested optimization approach. This approach involves optimizing the inner loop (minimizing $\mathcal{L}_{CE}$) before tackling the outer loop (minimizing $\mathbf{Z}$).

To identify a set of solutions that best balance the trade-off between model performance and hardware constraints, we introduce two modifications to our objective function:

- We introduce a penalty term that becomes increasingly negative as the quantized model's accuracy falls below a user-specified threshold. This ensures that solutions prioritize models meeting the desired accuracy level.
- We incorporate a hard constraint on the model size. This constraint acts as a filter, preventing the algorithm from exploring solutions that exceed a user-defined maximum size for the quantized model.

With respect to the above and by using a maximization problem, our objective is defined as follows:

$$\textbf{max } \{(\mathbf{Z}_{ref} - \mathbf{Z}) \odot M\} \tag{2}$$

where:

$$M = \begin{cases} \mathbf{1}, & \text{size} < \text{Memory size} \\ 0, & else \end{cases}$$

and $\mathbf{Z}_{ref}$ would be the baseline performance. We used a uniform 8-bit quantized network performance as $\mathbf{Z}_{ref}$.

4.2. Multivariate Markov Chain

Quantization error contributes to the overall approximation error. These errors can be conceptualized as non-orthogonal signals within the error space. Crucially, the error vectors may not always point in the same direction.

Simply quantizing the model's weight matrices and adding the resulting error to predictions is insufficient. Furthermore, quantized models without a subsequent fine-tuning step often underperform.

Nahshan et al. [41] demonstrates that the quantization error signals of models with similar bitwidths exhibit smaller angles in the error space, indicating higher similarity. This suggests that, even for models destined for very low precision, an intermediate quantization step (e.g., 8 bits) can be beneficial.

To the best of our knowledge, the optimal approach for progressively quantizing deep learning models, especially those with mixed precision and complex architectures, remains an open question. The high dimensionality of such a process makes traditional ablation studies challenging.

While it is well known that optimization of DNNs is not an MDP nor canonical for each precision [42,43], Nahshan observations motivate our exploration of temporal precision in the quantization process. We propose modeling the network's state as a function of its previous mixed-precision configuration, akin to a multivariate Markov chain.

Formally, we define the bit allocation as a multivariate Markov chain, as defined in [44].

Let ${A^{(i)}}_t$ be the bit allocation of the i-th layer at time t, defined as:

$$ {A^{(i)}}_{t+1} = \lambda_{ii}\mathbf{P}^{(ii)}{A^{(i)}}_t + \sum_{j=1,j\neq i}^{N} \lambda_{ij}{A^{(j)}}_t \text{ for } i = 1,2,\ldots,N. \tag{3} $$

where:

$$ \lambda_{ij} \geq 0, \sum_{j=1}^{N} \lambda_{ij} = 1 \text{ for } i = 1,2,\ldots,N. \tag{4} $$

and $\mathbf{P}^{(ii)}$ is a one-step transition probability matrix for the i-th layer precision as a Markov chain.

In the context of this work, every layer of the neural network i has a precision at time t, and the transition for it to a new precision at $t+1$ is given by Equation (3).

We denote the multivariate transition matrix of all layers as $\mathcal{Q}$:

$$ \mathcal{Q} = \begin{pmatrix} \lambda_{11}\mathbf{P}^{(11)} & \lambda_{12}I & \ldots & \lambda_{1N}I \\ \lambda_{21}I & \lambda_{22}\mathbf{P}^{(22)} & \ldots & \lambda_{2N}I \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{N1}I & \lambda_{N2}I & \ldots & \lambda_{NN}\mathbf{P}^{(NN)} \end{pmatrix} \tag{5} $$

and thus, the bit-allocation multivariate Markov chain update rule is:

$$ \mathbf{A}_{t+1} = \mathcal{Q}\mathbf{A}_t \tag{6} $$

Modeling $\mathcal{Q}$ explicitly is challenging, since we would like to model it so close minima of the loss will have the highest probability. As an alternative to explicit modeling, we suggest using sampling techniques over $\hat{\mathcal{Q}}$ where $\mathcal{Q} \propto \hat{\mathcal{Q}}$. In this study, we use random walk Metropolis–Hastings [45], a Markov chain Monte Carlo (MCMC) method. For $\hat{\mathcal{Q}}$, we construct a distribution from Objective 2.

We use an exponential moving average (EMA), denoted by γ, to avoid cases of diverged samples of quantized models:

$$ \hat{\mathcal{Q}}_{t+1} = \gamma\hat{\mathcal{Q}}_t + \mathbf{q} \tag{7} $$

where $\gamma = 0.01$ empirically reduces the number of required samples.

The update of new samples $\mathbf{q}$ is by applying a logarithmic scale over Equation (2):

$$ \mathbf{q} = \log\left(\frac{\mathbf{Z}_{ref}}{\mathbf{Z}}\right) \odot \mathbf{M} = \log\left(\frac{\mathcal{L}_{CE}^{ref} + \beta\mathcal{L}_{Lat}^{ref}}{\mathcal{L}_{CE} + \beta\mathcal{L}_{Lat}}\right) \odot \mathbf{M} \tag{8} $$

where $\mathbf{q}$ is the update step of the transition matrix $\hat{\mathcal{Q}}$, which is simply updating the probability of transitioning to a different precision based on the scaled loss. $\mathbf{Z}$ is defined in Equation (1) by the weighted sum of losses. Note that the only difference here from Equation (2) is the monotonic logarithm function, as well as taking the mask out of the log. since the mask operated elementwise (Hadamard product) with the objective.

We employ the random walk Metropolis–Hastings algorithm (Algorithm 1) to determine whether to accept a new bit-allocation vector $\mathbf{A}$ or retain the current one.

Algorithm 1 Random walk Metropolis–Hastings step.

Input: $\hat{\mathcal{Q}}$, $\mathbf{A}_i$
$bbA_* = \arg\max \hat{\mathcal{Q}}$ ▷ Axis 2
$\alpha = \frac{\hat{\mathcal{Q}}[\mathbf{A}_*]}{\hat{\mathcal{Q}}[\mathbf{A}]}$ ▷ α is the layerwise acceptance ratio;
if $0 \leq \alpha \leq 1$ **then** ▷ Element-wise
  $\mathbf{B} \sim \text{Bern}(\alpha)$
  $\mathbf{A}_{i+1} = \mathbf{A}_* \mathbf{B} + \mathbf{A}_i(\mathbf{1} - \mathbf{B})$
else
  $\mathbf{A}_{i+1} = \mathbf{A}_*$
end if

The algorithm proceeds in the following steps:

1. **Candidate Generation:** A candidate vector, denoted by A_*, is proposed for the next allocation.
2. **Layerwise acceptance ratio:** A layerwise acceptance ratio, α, is calculated.
3. **Bernoulli-Based Acceptance:**
   - If $\alpha \leq 1$, a Bernoulli distribution with probability α is used for acceptance:
     - If the Bernoulli trial succeeds, A_* is accepted.
     - Otherwise, the current allocation is retained.
   - If $\alpha \geq 1$, A_* is automatically accepted.

This acceptance scheme adaptively balances exploration and exploitation:
Exploration Phase: Accepts a higher proportion of new candidates, encouraging broad space exploration. **Exploitation Phase:** Preferentially accepts candidates that leverage knowledge from previously discovered samples.

4.3. Quantizer

Our approach is agnostic to the specific quantization technique employed. As long as the chosen quantizer is compatible with the user's target accelerator, it can be seamlessly integrated. The experiments presented in this paper leverage a quantization-aware training (QAT) quantizer along with a learnable quantization scale, as advocated in [31].

During the optimization of the quantized network, we employ a technique called fake quantization. This involves maintaining a full-precision (FP32) copy of the weights, while using a simple straight-through estimator for back-propagation [46]. This estimator approximates the gradients of the quantized weights during training.

To quantize a matrix M with b bits, we follow a two-step process:

1. *Scaling and rounding:*
$$M_{int} = \text{round}\left(\frac{M}{S}\right)$$
Here, M_{int} represents the integer version of M, obtained by scaling M by a factor S (often referred to as the scaling factor) and then rounding the result.
2. *Clamping:*
$$\bar{M} = \text{clamp}(M_{int}, \text{min}, \text{max})$$
The scaled and rounded integer M_{int} is then clamped to the valid range representable by b bits. This ensures the quantized values stay within the intended range.
   - For symmetric quantization, $\text{min} = -2^{(b-1)} + 1$ and $\text{max} = 2^{(b-1)} - 1$.
   - For asymmetric quantization, $\text{min} = 0$ and $\text{max} = 2^{(b-1)}$.

The clamping values depend on the chosen quantization scheme: symmetric or asymmetric. A figure illustrating low-bit multiplication in a layer is provided in Figure 3.

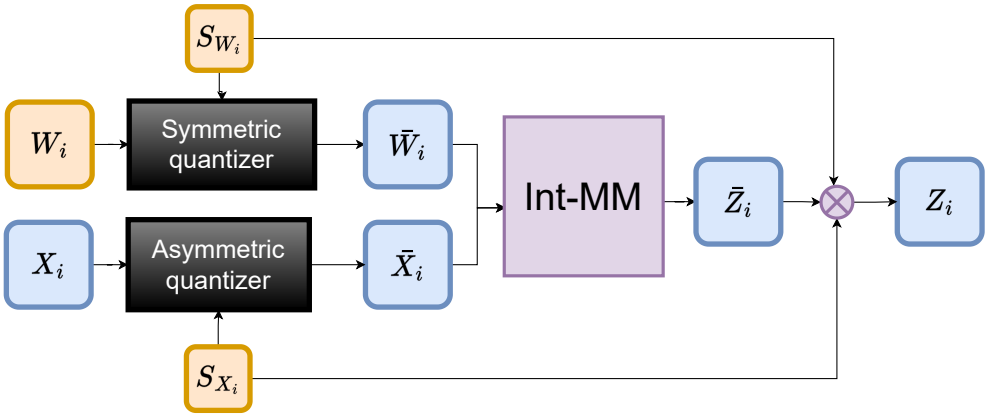

Figure 3. Quantization of the i-th layer of the network. W_i indicates the weights, X_i the input activations, and $\bar{W}_i, \bar{X}_i$ the quantized versions, respectively. S_{W_i}, S_{X_i} are the learnable parameters of the quantization. (scale). In Blue are dynamically changed tensors, while in orange the parameters. Weights and activations are quantized with respect to the scaling factor (with rounding and clamping as described in Section 4.3. The quantized versions are multiplied in an integer matrix multiplication accelerator and produce a quantized vector $\bar{Z}_i$. With respect to the scaling factors, we can dequantize them into Z_i, which can yield a prediction in FP, or quantize them again in a different precision in the next layer.

4.4. Simulator

To enable direct sampling of signals from emulated hardware, we developed a hardware accelerator simulator. This simulator allows us to model various aspects of an accelerator, including the inference time for different architectures, and generate signals usable within our algorithm. These signals can extend beyond latency to encompass memory utilization, energy consumption, or other relevant metrics.

4.4.1. Underlying Architecture Modeled

Our simulator draws heavily on SCALE-Sim v2 [17], a Systolic Accelerator Simulator (SAS) capable of cycle-accurate timing analysis. A SAS is ideal for DNN computations due to its efficient operand movement and high compute density. This setup minimizes global data movement, largely keeping data transfer local (neighbor to neighbor within the array), which improves both energy efficiency and speed. Our SAS can additionally provide power/energy consumption, memory bandwidth usage, and trace results, all tailored to a specific accelerator configuration and neural network architecture. We extended its capabilities by incorporating support for diverse bitwidths and convolutional neural network (CNN) architectures not originally supported, such as Inverted Residuals (described in [47]).

4.4.2. Simulator Approximations

1. `Optimal Data Flow Assumptions`: The simulator models specific types of dataflows—Output Stationary (OS), Weight Stationary (WS), or Input Stationary (IS)—and assumes an ideal scenario where outputs can be transferred out of the compute array without stalling the compute operations. In real-world implementations, such smooth operations might not always be feasible, potentially leading to a higher actual runtime.

2. `Memory Interaction`: This simplistically models the memory hierarchy, assuming a double-buffered setup to hide memory access latencies. This model may not fully capture the complex interactions and potential bottlenecks of real memory systems.

The original *ScaleSim* simulator was validated against real hardware setups using a detailed in-house RTL model [17].

4.4.3. Using the Simulator

The simulator operates on two key inputs:

1. Network Architecture Topology File: This file specifies the structure of the neural network, including the arrangement of layers and their connections.

2. Accelerator Properties Descriptor: This descriptor defines the characteristics of the target hardware accelerator, such as its memory configuration and processing capabilities.

We evaluated the simulator using various network architectures: ResNet-18, ResNet-50 [48], and MobileNetV2 [47]. While MobileNetV3 could potentially be explored in future work, it is not included in the current set of experiments. The specific accelerator properties used are detailed in Table 1.

Table 1. The accelerator setup for a compact accelerator based on SCALE-Sim [17], the SCALE-Sim micro-controller with low memory, and Eyeriss [49]. The proprieties we used in the simulator for each setup are listed in the table. Data flow indicates the stationarity (weights—"ws"; activations—"as"; output—"os"), i.e., what data should remain in the SRAM for the next computed layer. "os" writes the output directly to the input feature map SRAM.

Name	SCALE-Sim	SCALE-Sim Low Mem	Eyeriss v1
PE array height	32	32	12
PE array width	32	32	14
Input feature map SRAM (KB)	64	4	108
Filter SRAM (KB)	64	4	108
Output feature map SRAM (KB)	64	4	108
Data flow	os	os	ws
Bandwidth (w/c)	10	10	10
Memory banks	1	1	1
Speed (GHz)	0.2	0.1	0.2

SRAM Utilization Estimation:

The current simulator estimates SRAM utilization based on bandwidth limitations. Incorporating a more accurate calculation of SRAM utilization within the simulator is a potential area for future improvement. This would provide a more precise signal for the algorithm.

Simulator Output:

The simulator generates a report for each layer, containing various metrics such as the following:

- Compute cycles;
- Average bandwidths for DRAM accesses (input feature map, filters, output feature map);
- Stall cycles;
- Memory utilization (potentially improved in future work);
- Other details specified in Appendix C.

Extracting Latency Metrics:

From the reported compute cycles and clock speed (f), we calculate the computation latency as $\frac{C}{f}$. Similarly, the memory latency for each SRAM is estimated using:

$$\frac{b}{M - BW \times \text{word size}} \times f$$

where

- C denotes the compute cycles;
- f denotes the clock speed;
- b denotes the number of bits required for the specific SRAM;
- $M - BW$ denotes the memory bandwidth;
- Word size is assumed to be 16 bits.

The total latency of the quantized model is determined by the maximum latency value obtained from these calculations (computation and memory latencies for each layer).

4.5. Training and Quantizing with AMED

AMED employs a Metropolis–Hastings algorithm (Algorithm 1) to sample precision vectors **A**. These precision vectors guide the quantization process, resulting in a mixed-precision model θ_A. Details on the quantization procedure, including activation quantization matching weight precision, can be found in Section 4.3.

Following quantization, we perform a two-epoch optimization step using stochastic gradient descent (SGD) to fine-tune both the quantized model parameters θ_A and the scaling factors S_{W_i} and S_{X_i} associated with the weights and activations, respectively.

The quantized model's performance is evaluated on the validation set using the cross-entropy loss $\mathcal{L}^l_{CE}$ and on a simulated inference scenario detailed in Section 4.4, using the latency loss $\mathcal{L}^l_{lat}$. Both loss values contribute to updating the estimated expected utility $\hat{\mathcal{Q}}$ (Equation (7)) and guide the sampling of a new precision vector **A**.

Figure 4 illustrates the quantization process, while Algorithm 2 details the complete workflow.

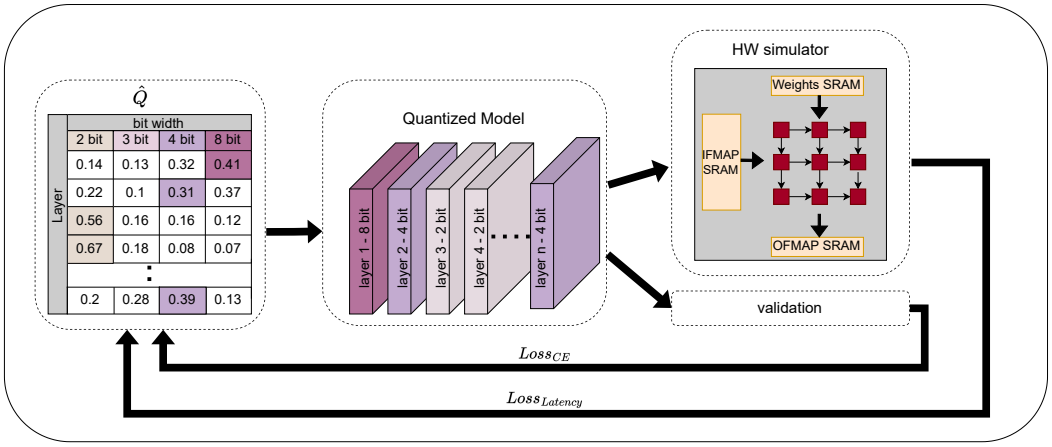

Figure 4. An illustration of Algorithm 2. $\hat{\mathcal{Q}}$ Table represents bit-allocation vector **A**

Algorithm 2 Training procedure of AMED.

Input: dataset: $\mathbf{D}(x, y)$, model: θ, params: β, γ, simulator **S**
$\mathbf{A}^i_0 = 8; \forall i \in L$
$\theta_0 = \theta_{\mathbf{A}_0}$
$\hat{\mathcal{Q}} \sim U(2, 8)$ ▷ between all bit representations
Fit θ_0
Compute reference $\hat{\mathcal{L}_{CE}}(\theta_0, \mathbf{D})$,$\hat{\mathcal{L}_{Lat}}(\mathbf{S}, \theta_0)$
for i in epoch **do**
  Evaluate $\mathcal{L}_{CE}(\theta_i, \mathbf{D})$,$\mathcal{L}_{Lat}(\mathbf{S}, \theta_i)$
  Update $\hat{\mathcal{Q}}$ ▷ from (7)
  Update $\mathbf{A}_i$ ▷ by Algorithm 1
  Quantize the model $\theta_{\mathbf{A}_i}$
  Fit $\theta_{\mathbf{A}_i}$
end for

The algorithm described in Algorithm 2 is agnostic to the quantizer and to the hardware specification and simulation. This means that one can use any quantization technique that relies on the statistics of the weights and activations in a single layer and any hardware or hardware simulator and use AMED to choose the best mixed-precision bit allocation for the hardware.

5. Results

In this section, we present our quantization experimental results on the ImageNet dataset [50] for different architectures.

We applied our AMED algorithm to determine the bitwidth of each layer. We used an SGD optimizer with a momentum of 0.9 and a weight decay of 10^{-3} for ResNet and a weight decay of 10^{-5} for MobileNet. We ran each network for 80–90 epochs with a starting learning rate of 10^{-2} for ResNet and 10^{-3} for MobileNet. The learning rate dropped by a factor of 10 every 30 epochs. The batch size was 256, and we used common data augmentations of random horizontal flip and random crop. The values of β as listed in Table 2 are $\beta_1 = 1$ and $\beta_2 = 10$. We also followed the common practice of not quantizing the classifier (FC layer), which is less than 3% of the latency of the smallest network we trained. All experiments used a pretrained model quantized uniformly to 8 bits by the regime of [31]. We applied a uniform quantization technique and learnable scale for both weights and activations, as described in Figure 3. The scale factor S^W for the weights was initialized with the statistics from the INT8 quantized model, and S^X for activations was initialized with the statistics from one batch (of size 256) in the following form:

$$S_0^x = \frac{max(\|X\|)^2}{2^{(b-1)} - 1} \tag{9}$$

where X is the first batch of images and b is the number of bits that represent the quantized layer. All experiments using the SCALE-Sim accelerator hardware setup for the ImageNet dataset are listed in Table 2. Other hardware performance parameters are given in Table A1.

For comparison, we tested the quantization performance of our method, as well as other methods found in the literature: HAWQ [38], MCKP [39], LSQ [31], DDQ [40], and LIMPQ [51] for ResNet-18, HAQ [19], HAWQ [38], MCKP [39], LSQ [31], and LIMPQ [51] for ResNet-50, and MCKP [39], DDQ [40], HAQ [19], PROFIT [52], and LSQ + BR [32] for MobileNet-V2.

Table 2. Performance comparison with state-of-the-art methods on ImageNet, noticeable good results in bold. N_{MP} indicates mixed precision using N as the minimum allowed bitwidth. ψ is our re-implementation, with pretrained FP32 weights from [53]. We only present our implementation when we achieved better results than the original paper. If the original paper we are comparing to did not publish the model's bit allocation, we could not run the simulator and find the latency or calculate the model size.

Network	Method	Bitwidth	Acc (%)	Latency (ms)	Size (MB)
ResNet-18	FP32	32/32	71.1	92.34	43.97
	LSQ_ψ	8/8	**71.0**	34.97	14.68
	LSQ_ψ	4/4	68.73	24.4	7.34
	HAWQv2	$4_{MP}/4_{MP}$	70.22	32.83	8.52
	MCKP	$3_{MP}/4_{MP}$	69.66	28.05	7.66
	DDQ	$4_{MP}/4_{MP}$	**71.2**	29.03	7.83
	LIMPQ	$3_{MP}/3_{MP}$	69.7	20.03	7.55
	LIMPQ	$4_{MP}/4_{MP}$	**70.8**	33.05	8.52
	AMED (β_1)	$2_{MP}/2_{MP}$	**70.87**	**9.3**	7.87
	AMED (β_2)	$2_{MP}/2_{MP}$	67.77	**5.07**	7.06

Table 2. *Cont.*

Network	Method	Bitwidth	Acc (%)	Latency (ms)	Size (MB)
ResNet-50	FP32	32/32	80.1	263.64	102.06
	LSQ_ψ	8/8	**79.9**	101.4	20.74
	LSQ_ψ	4/4	78.3	55.84	10.37
	LSQ_ψ	3/3	77.6	**25.44**	7.79
	MCKP	$2_{MP}/4_{MP}$	75.28	46.42	7.96
	HAQ	$3_{MP}/3_{MP}$	75.3	—	—
	HAWQv2	$2_{MP}/4_{MP}$	76.1	86.51	10.13
	LIMPQ	$3_{MP}/4_{MP}$	76.9	**32.51**	8.11
	AMED (β_1)	$2_{MP}/2_{MP}$	**79.23**	**37.52**	11.89
	AMED (β_2)	$2_{MP}/2_{MP}$	78.5	**34.47**	7.75
MobileNetV2	FP32	32/32	71.80	104.34	17.86
	LSQ_ψ	8/8	71.6	39.52	12.54
	MCKP	$2_{MP}/8$	71.2	22.42	9.82
	HAQ	$3_{MP}/3_{MP}$	66.99	—	—
	HAQ	$4_{MP}/4_{MP}$	71.47	15.89	10.47
	DDQ	$4_{MP}/4_{MP}$	**71.8**	29.25	10.217
	PROFIT	$4_{MP}/4_{MP}$	71.5	—	—
	LSQ + BR	3/3	67.4	**11.96**	11.429
	AMED (β_1)	$2_{MP}/2_{MP}$	**71.29**	15.01	6.34
	AMED (β_2)	$2_{MP}/2_{MP}$	71.2	**11.85**	10.12

We tested our model on various hardware setups. Figure 5 shows that our method produced a different bit allocation for MobilenetV2 for each hardware constraint. The effect of different values for β for the same model and the same hardware simulator is shown in Figure 6, and other outcomes for different hardware constraints for ResNet-50 can be seen in Figure A1.

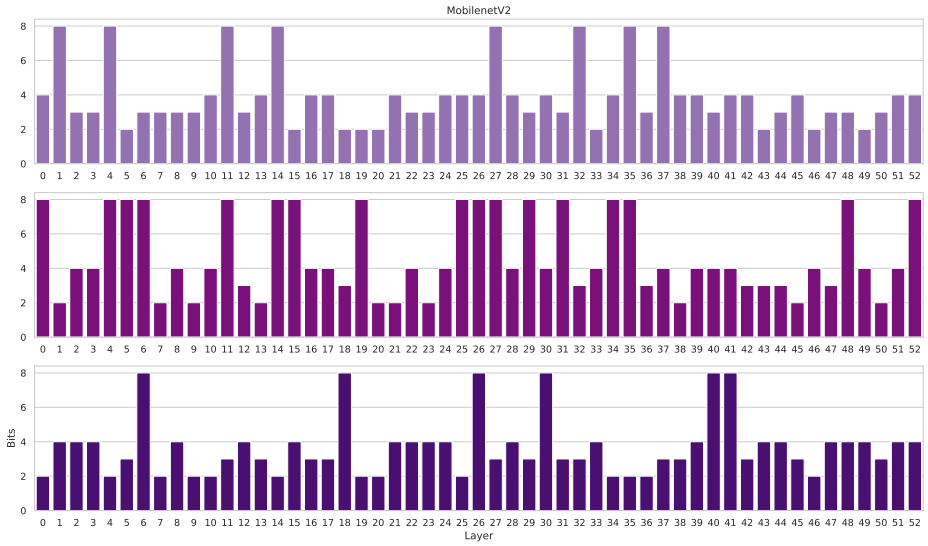

Figure 5. Quantization bit allocation of MobileNetV2 following our method using the simulator. The top figure is the SCALE-Sim setup; the middle is the Eyeriss setup; the bottom is SCALE-Sim with low memory. Depthwise convolutions have a higher feature map and, thus, higher memory footprint, and we can see that Algorithm 2 allocates fewer bits when the system memory is low, i.e., the model is memory-bounded. Models with higher memory allocate the bits differently due to the locality of the boundary (memory or computational) by the layer. This figure does not include the first and last layers, which we quantize to 8 bits.

Figure 6. Quantization bit allocation of ResNet-18 following our method using the simulator. Both are for the SCALE-Sim setup. The top figure is for $\beta = 1$, and the bottom is for $\beta = 10$. When choosing a higher value for β, the algorithm chooses lower precision for the model for the same hardware constraint.

We report our quantized results in Table 2. Because AMED can efficiently provide a trade-off between latency and accuracy, we can control the accuracy degradation and latency requirements easily by a simple adjustment of the hyperparameters. For each architecture, we report multiple results that demonstrate this trade-off. As seen in Table 2, for ResNet-18, we achieved more than a x2.6 latency improvement with only a 0.23% drop in accuracy compared to the latest state-of-the-art quantization methods: LSQ [31] and DDQ [40]. For ResNet-50, compared to the 4-bit models, we outperformed in accuracy by 0.2–0.93% while still improving latency. For MobileNetV2, we can also see more than a 1 ms improvement in latency compared to the HAQ [19] 4-bit mixed-precision model with a 0.5% degradation in accuracy. We emphasize that all results are prone to the β setting and a degradation in accuracy can easily be compensated for by a higher latency. As demonstrated in Figures 1, 7 and 8, AMED achieves a better Pareto curve for the accuracy–latency trade-off, which can dominant other quantized models.

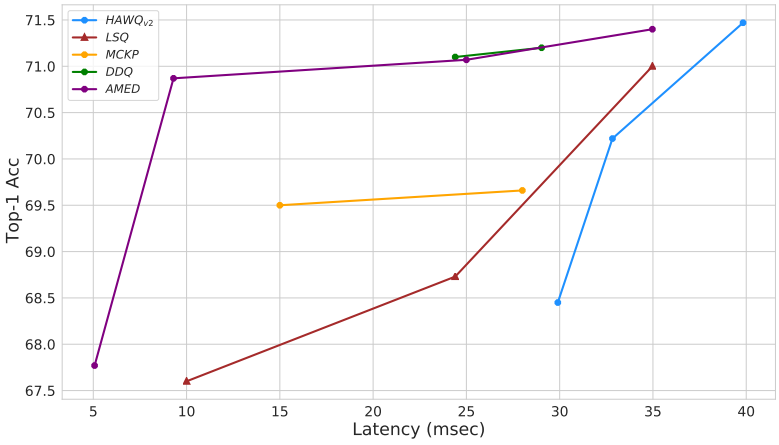

Figure 7. ResNet-18 quantized models on a latency–accuracy plane. Circles are mixed precision, and triangles are uniform quantization. Our models achieved a better Pareto curve of the dominant solution in the two-dimensional plane for ultra-low precision.

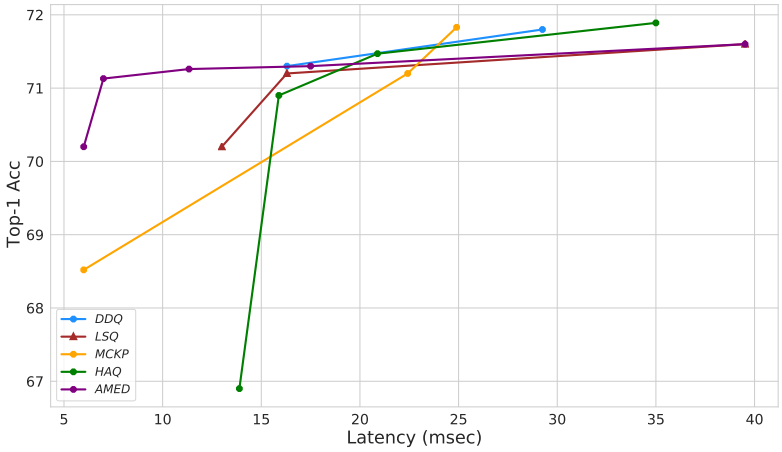

Figure 8. MobileNetV2 quantized models on a latency–accuracy plane. Circles are mixed precision, and triangles are uniform quantization. Our models achieved a better Pareto curve of the dominant solution in the two-dimensional plane for ultra-low precision.

Ablation Study

In this section, we describe our tests of AMED on CIFAR100 [54] with a ResNet-18 architecture with different hyperparameters, as listed in Table 3. The method chooses the bitwidths, resulting in lower latency when β is higher, as expected, because as shown in Equation (1), the higher β increases the objective's latency component. The use of EMA helps when we fix a small β value. This result emphasizes the importance of averaging the score of each bitwidth with older samples.

Table 3. Performance comparison of different hyperparameters of ResNet-18 on CIFAR100. We used the SCALE-Sim simulator described in Table 1, and the latency is normalized to one image inference.

β	EMA	Top-1 (%)	Top-5 (%)	Latency (ms)
1	0.9	75.59	96.15	10.14
1	0.5	73.08	92.19	14.68
1	0.2	77.86	95.09	14.68
1	0.1	77.58	92.97	12.47
1	0.01	78.21	94.97	8.99
100	0.01	77.29	91.44	6.15
10	0.01	78.29	93.97	8.15
1	0.01	78.52	94.53	8.98
0.1	0.01	78.72	96.23	12.61

6. Discussion

In this paper, we introduced a novel mixed-precision quantization method called AMED. The proposed method relies on a novel meta-bit allocation strategy that finds an optimal bitwidth among different neural network layers by measuring direct signals from a hardware accelerator simulator. Our method significantly reduces the required exploration space compared to previous mixed-precision methods, due to the simplicity of the perspective of the low-degree objective. The extensive evaluations we performed demonstrated the superiority of our method over standard image classification benchmarks in terms of the accuracy–latency trade-off compared to the prior state of the art. The ability to obtain higher accuracy in a shorter training time results in lower time-to-market solutions.

Future work to examine this method for efficient NAS could yield a computationally efficient search, which will reduce the carbon footprint of the procedure and reveal better models in terms of inference time.

Another intriguing future topic is the exploration of different loss terms, solving dense prediction tasks like semantic segmentation [55,56].

Improvements in the simulator, such as enabling dynamic workflow based on the computational graph, or support for special hardware solutions such as fast sparse matrix multiplication [57], combined with pruning, could result in very promising outcomes.

Future Directions

Computationally efficient NAS: Integrating AMED with efficient Neural Architecture Search (NAS) techniques has the potential to yield a significantly more computationally efficient search process. This would not only reduce the carbon footprint associated with the search procedure, but also potentially uncover models with superior inference times.

Exploration of diverse loss terms: The ability to directly measure various empirical values within the deployment system through the hardware simulator opens doors for exploring a much broader range of loss terms. Unlike previous approaches, differentiability is no longer a prerequisite for loss terms, allowing us to consider factors like power consumption, memory usage, and bandwidth within the existing deployment environment.

Enhanced Simulator Capabilities: Further improvements to the simulator, such as enabling dynamic workflows based on the computational graph or incorporating support for specialized hardware solutions like Fast Sparse Matrix Multiplication [57] alongside pruning techniques, hold immense promise for achieving groundbreaking results.

Author Contributions: Conceptualization and methodology, M.K., C.B. and A.M.; code, M.K.; formal analysis, writing and visualization, M.K., T.R., C.B. and A.M.; resources, C.B. and A.M.; supervision and Funding, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Israel Innovation Authority, Nofar grant.

Data Availability Statement: Accessed data for the experiments for CIFAR-10 and ImageNet can be found at https://www.cs.toronto.edu/~kriz/cifar.html and https://www.image-net.org/ (accessed on 28 January 2023), respectively.

Acknowledgments: The authors would like to thank Tal Kopetz and Olya Sirkin from CEVA LTD for the helpful discussions and brainstorming during this project.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FLOP	floating-point operations
MAC	multiply–accumulate
DNNs	deep neural networks
MP	mixed-precision
FP32	floating-point with 32 bits
NAS	Neural Architectural Search
CE	cross-entropy
PTQ	Post-Training Quantization
QAT	Quantization-Aware Training
PE	processing element
SRAM	static random-access memory

Appendix A. Additional Experiments

This appendix includes the results for the Eyeriss hardware setup, as listed in Table 1. The results are presented in Table A1 in this appendix and show that our method finds a different precision per layer for each model. We also found that, for a low memory boundary, our model chooses strategic layers to maintain high precision regardless of the size of the feature map, and tries to compensate for the latency over other layers. More results and more experiments can be reproduced using the GitHub repository mentioned in this paper.

Table A1. The same comparison as in Table 2 and the same notations, but with the latency from the simulator of the Eyeriss setup from Table 1. Note that our model yields a different bitwidth for each layer because the signal from the hardware is different in this setup.

Network	Method	Bitwidth	Acc (%)	Latency (ms)	Size (MB)
ResNet-18	FP32	32/32	71.1	—	43.97
	LSQ_ψ	8/8	70.0	3.42	14.68
	LSQ_ψ	4/4	68.73	0.85	7.34
	HAWQv2	$4_{MP}/4_{MP}$	70.22	2.94	8.52
	MCKP	$3_{MP}/4_{MP}$	69.66	2.13	7.66
	DDQ	$4_{MP}/4_{MP}$	71.2	2.22	7.83
	AMED	$2_{MP}/2_{MP}$	70.84	0.32	6.16
	AMED	$2_{MP}/2_{MP}$	71.0	0.55	6.6
ResNet-50	FP32	32/32	80.1	—	102.06
	LSQ_ψ	8/8	79.9	9.65	20.74
	LSQ_ψ	4/4	78.3	2.42	10.37
	LSQ_ψ	3/3	77.6	0.8	7.79
	MCKP	$2_{MP}/4_{MP}$	75.28	3.22	7.96
	HAWQv2	$2_{MP}/4_{MP}$	76.1	7.82	10.13
	AMED	$2_{MP}/2_{MP}$	79.34	3.26	9.74
	AMED	$2_{MP}/2_{MP}$	79.43	3.45	9.75
MobileNetV2	FP32	32/32	71.80	—	17.86
	LSQ_ψ	8/8	71.6	7.8	12.54
	MCKP	$2_{MP}/8$	71.2	4.44	9.82
	HAQ	$3_{MP}/3_{MP}$	70.9	—	—
	HAQ	$4_{MP}/4_{MP}$	71.47	2.03	10.47
	DDQ	$4_{MP}/4_{MP}$	71.8	5.87	10.217
	PROFIT	$4_{MP}/4_{MP}$	71.5	—	—
	LSQ + BR	3/3	67.4	3.9	11.429
	AMED	$2_{MP}/2_{MP}$	71.24	1.56	12.1
	AMED	$2_{MP}/2_{MP}$	70.97	1.21	9.53

Appendix B. Bit Allocations

In Figure A1, we add the bit allocation for ResNet-50 with a different hardware setup, and in Figure 6, we use different values for β for ResNet-18.

One can see that both affect the bit allocation significantly towards building a more efficient network when the memory boundary is closer or when we increase the hardware constraints on the objective.

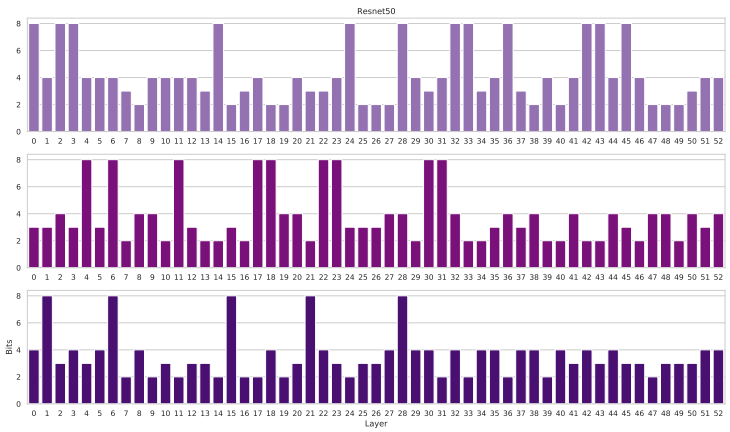

Figure A1. Quantization bit allocation of ResNet-50 following our method using the simulator. The top figure is the SCALE-Sim setup; the middle is the Eyeriss setup; the bottom is SCALE-Sim with low memory.

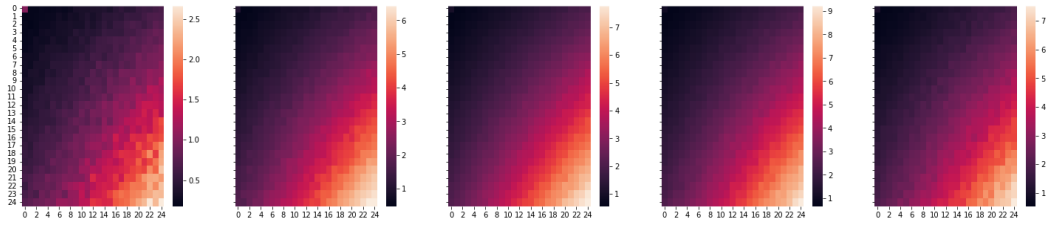

Figure A2. Visualization of the loss surface of two subsequent layers of ResNet-18. At a higher bitwidth (left), the interactions between layers are relatively small, making layerwise optimization possible. On the other hand, with a bitwidth decrease (right), with an increase in the quantization loss, the interactions become tangible and the loss is higher. A per-layer optimization depends on the initial point and is potentially sub-optimal.

Appendix C. Reports

The simulator provides the following detailed reports about the performance during the inference of a CNN:

- Computation report: Provides layerwise details about Total Cycles, Stall Cycles, Overall Utilization, Mapping Efficiency, and Computation Utilization. An example is shown in Table A2.
- Bandwidth report: Provides layerwise details about Average IFMAP SRAM Bandwidth, Average FILTER SRAM Bandwidth, Average OFMAP SRAM Bandwidth, Average IFMAP DRAM Bandwidth, Average FILTER DRAM Bandwidth, and Average OFMAP DRAM Bandwidth.
- Detailed access report: Provides layerwise details about the number of reads and writes, and the start and stop cycles, for both of the above-mentioned reports.

Table A2. An example of the SCALE-Sim simulator computation report similar to [17] for MobileNetV2 uniformly quantized to 2 bits (not including the FC layer).

LayerID	Total Cycles	Stall	Overall Util %	Mapping Efficiency %	Compute Util %
0	137,148	0	16.10753928	53.09483493	16.10742183
1	132,649	0	43.61369102	53.00234993	43.61336223
2	36,847	0	9.574727929	28.125	9.574468085
3	61,151	0	15.70538503	76.5625	15.70512821
4	701,907	0	71.27146118	76.38573961	71.27135964
5	15,483	0	24.68513854	40.625	24.6835443
6	25,283	0	21.22176957	76.04166667	21.22093023
7	374,807	0	71.87994421	75.31844429	71.87975243
8	20,187	0	28.399465	40.625	28.39805825
9	25,283	0	21.22176957	76.04166667	21.22093023
10	374,807	0	71.87994421	75.31844429	71.87975243
11	5149	0	36.4002719	52.0625	36.39320388
12	9399	0	25.28460475	74.265625	25.28191489
13	157,519	0	70.24724002	72.76722301	70.24679406
14	6349	0	39.36052922	52.0625	39.35433071
15	9399	0	25.28460475	74.265625	25.28191489
16	157,519	0	70.24724002	72.76722301	70.24679406
17	6349	0	39.36052922	52.0625	39.35433071
18	9399	0	25.28460475	74.265625	25.28191489

Table A2. *Cont.*

LayerID	Total Cycles	Stall	Overall Util %	Mapping Efficiency %	Compute Util %
19	157,519	0	70.24724002	72.76722301	70.24679406
20	3555	0	34.11392405	45.1171875	34.10433071
21	6173	0	38.2998542	75.390625	38.29365079
22	123,129	0	76.17864191	77.54464286	76.17802323
23	6243	0	38.8515137	45.1171875	38.84529148
24	6173	0	38.2998542	75.390625	38.29365079
25	123,129	0	76.17864191	77.54464286	76.17802323
26	6243	0	38.8515137	45.1171875	38.84529148
27	6173	0	38.2998542	75.390625	38.29365079
28	123,129	0	76.17864191	77.54464286	76.17802323
29	6243	0	38.8515137	45.1171875	38.84529148
30	6173	0	38.2998542	75.390625	38.29365079
31	123,129	0	76.17864191	77.54464286	76.17802323
32	6243	0	57.68861124	66.9921875	57.6793722
33	11,059	0	48.01858215	79.0234375	48.01424051
34	262,299	0	80.32093146	81.28125	80.32062524
35	8931	0	60.48874706	66.9921875	60.48197492
36	11,059	0	48.01858215	79.0234375	48.01424051
37	262,299	0	80.32093146	81.28125	80.32062524
38	8931	0	60.48874706	66.9921875	60.48197492
39	11,059	0	48.01858215	79.0234375	48.01424051
40	262,299	0	80.32093146	81.28125	80.32062524
41	3827	0	58.33714398	64.59960938	58.32190439
42	7103	0	51.84649092	71.92687988	51.83919271
43	139,231	0	72.87237576	73.39477539	72.87185238
44	6131	0	60.69054803	64.59960938	60.68065068
45	7103	0	51.84649092	71.92687988	51.83919271
46	139,231	0	72.87237576	73.39477539	72.87185238
47	6131	0	60.69054803	64.59960938	60.68065068
48	7103	0	51.84649092	71.92687988	51.83919271
49	139,231	0	72.87237576	73.39477539	72.87185238
50	12,263	0	60.31099649	64.20084635	60.30607877
51	16,043	0	61.18127844	73.03059896	61.1774651
52	21,471	0	2.916724885	3.057861328	2.916589046

References

1. Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; Lempitsky, V.S. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. *arXiv* **2015**, arXiv:1412.6553.
2. Ullrich, K.; Meeds, E.; Welling, M. Soft Weight-Sharing for Neural Network Compression. *arXiv* **2017**, arXiv:1702.04008.
3. Chmiel, B.; Baskin, C.; Zheltonozhskii, E.; Banner, R.; Yermolin, Y.; Karbachevsky, A.; Bronstein, A.M.; Mendelson, A. Feature Map Transform Coding for Energy-Efficient CNN Inference. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; pp. 1–9. [CrossRef]
4. Baskin, C.; Chmiel, B.; Zheltonozhskii, E.; Banner, R.; Bronstein, A.M.; Mendelson, A. CAT: Compression-Aware Training for bandwidth reduction. *J. Mach. Learn. Res.* **2021**, *22*, 1–20.
5. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both Weights and Connections for Efficient Neural Network. *arXiv* **2015**, arXiv:1506.02626.
6. Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; Li, H.H. Learning Structured Sparsity in Deep Neural Networks. In Proceedings of the NIPS, Barcelona, Spain, 9 December 2016.
7. Liu, H.; Simonyan, K.; Yang, Y. DARTS: Differentiable Architecture Search. *arXiv* **2019**, arXiv:1806.09055.
8. Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; Keutzer, K. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 10726–10734.
9. Cai, H.; Zhu, L.; Han, S. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *arXiv* **2019**, arXiv:1812.00332.
10. Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; Zou, Y. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv* **2016**, arXiv:1606.06160.
11. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *J. Mach. Learn. Res.* **2018**, *18*, 1–30.

12. Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P.I.J.; Srinivasan, V.; Gopalakrishnan, K. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *arXiv* **2018**, arXiv:1805.06085.
13. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful. *IEEE Solid-State Circuits Mag.* **2020**, *12*, 28–41. [CrossRef]
14. Karbachevsky, A.; Baskin, C.; Zheltonozhskii, E.; Yermolin, Y.; Gabbay, F.; Bronstein, A.M.; Mendelson, A. Early-Stage Neural Network Hardware Performance Analysis. *Sustainability* **2021**, *13*, 717. [CrossRef]
15. Apple. *Apple Describes 7 nm A12 Bionic Chips*; EENews: Washington, DC, USA, 2018.
16. Nvidia. *Nvidia Docs Hub: Train With Mixed Precision*. 2023. Available online: https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html (accessed on 28 January 2023).
17. Samajdar, A.; Joseph, J.M.; Zhu, Y.; Whatmough, P.; Mattina, M.; Krishna, T. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim. In Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Boston, MA, USA, 23–25 August 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 58–68.
18. Sharma, H.; Park, J.; Suda, N.; Lai, L.; Chau, B.; Kim, J.K.; Chandra, V.; Esmaeilzadeh, H. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 1–6 June 2018; pp. 764–775.
19. Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; Han, S. HAQ: Hardware-Aware Automated Quantization. *arXiv* **2018**, arXiv:1811.08886.
20. Dong, Z.; Yao, Z.; Gholami, A.; Mahoney, M.W.; Keutzer, K. HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 293–302.
21. Sun, M.; Li, Z.; Lu, A.; Li, Y.; Chang, S.E.; Ma, X.; Lin, X.; Fang, Z. FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization. In Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Virtual, 27 February–1 March 2022.
22. Sun, J.; Li, G. An End-to-End Learning-based Cost Estimator. *arXiv* **2019**, arXiv:1906.02560.
23. Strubell, E.; Ganesh, A.; McCallum, A. Energy and Policy Considerations for Deep Learning in NLP. *arXiv* **2019**, arXiv:1906.02243.
24. Srinivas, N.; Deb, K. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evol. Comput.* **1994**, *2*, 221–248. [CrossRef]
25. Deb, K. *Multi-Objective Optimization Using Evolutionary Algorithms*; John Wiley & Sons: Hoboken, NJ, USA, 2001.
26. Li, H.; De, S.; Xu, Z.; Studer, C.; Samet, H.; Goldstein, T. Training Quantized Nets: A Deeper Understanding. In Proceedings of the NIPS, Long Beach, CA, USA, 4–9 December 2017.
27. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized Neural Networks. *arXiv* **2016**, arXiv:1602.02505.
28. Rozen, T.; Kimhi, M.; Chmiel, B.; Mendelson, A.; Baskin, C. Bimodal Distributed Binarized Neural Networks. *arXiv* **2022**, arXiv:2204.02004.
29. Zhang, D.; Yang, J.; Ye, D.; Hua, G. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. *arXiv* **2018**, arXiv:1807.10029.
30. Baskin, C.; Liss, N.; Chai, Y.; Zheltonozhskii, E.; Schwartz, E.; Giryes, R.; Mendelson, A.; Bronstein, A.M. NICE: Noise Injection and Clamping Estimation for Neural Network Quantization. *arXiv* **2021**, arXiv:1810.00162.
31. Esser, S.K.; McKinstry, J.L.; Bablani, D.; Appuswamy, R.; Modha, D.S. Learned Step Size Quantization. *arXiv* **2020**, arXiv:1902.08153.
32. Han, T.; Li, D.; Liu, J.; Tian, L.; Shan, Y. Improving Low-Precision Network Quantization via Bin Regularization. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 10–17 October 2021; pp. 5241–5250.
33. Gong, R.; Liu, X.; Jiang, S.; Li, T.H.; Hu, P.; Lin, J.; Yu, F.; Yan, J. Differentiable Soft Quantization: Bridging Full-Precision and Low-Bit Neural Networks. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 4851–4860.
34. Zur, Y.; Baskin, C.; Zheltonozhskii, E.; Chmiel, B.; Evron, I.; Bronstein, A.M.; Mendelson, A. Towards Learning of Filter-Level Heterogeneous Compression of Convolutional Neural Networks. *arXiv* **2019**, arXiv:1904.09872.
35. Zhao, S.; Yue, T.; Hu, X. Distribution-aware Adaptive Multi-bit Quantization. In Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021; pp. 9277–9286.
36. Yang, H.; Duan, L.; Chen, Y.; Li, H. BSQ: Exploring Bit-Level Sparsity for Mixed-Precision Neural Network Quantization. *arXiv* **2021**, arXiv:2102.10462.
37. Yang, L.; Jin, Q. FracBits: Mixed Precision Quantization via Fractional Bit-Widths. In Proceedings of the AAAI, Palo Alto, CA, USA, 22–24 March 2021.
38. Dong, Z.; Yao, Z.; Cai, Y.; Arfeen, D.; Gholami, A.; Mahoney, M.W.; Keutzer, K. HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks. *arXiv* **2020**, arXiv:1911.03852.
39. Chen, W.; Wang, P.; Cheng, J. Towards Mixed-Precision Quantization of Neural Networks via Constrained Optimization. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 10–17 October 2021; pp. 5330–5339.
40. Zhang, Z.; Shao, W.; Gu, J.; Wang, X.; Ping, L. Differentiable Dynamic Quantization with Mixed Precision and Adaptive Resolution. *arXiv* **2021**, arXiv:2106.02295.

41. Nahshan, Y.; Chmiel, B.; Baskin, C.; Zheltonozhskii, E.; Banner, R.; Bronstein, A.M.; Mendelson, A. Loss Aware Post-training Quantization. *Mach. Learn.* **2021**, *110*, 3245–3262. [CrossRef]
42. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2159.
43. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980v9. [CrossRef].
44. Ching, W.K.; Zhang, S.; Ng, M.K. On Multi-dimensional Markov Chain Models. *Pac. J. Optim.* **2007**, *3*, 235–243.
45. Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **1953**, *21*, 1087–1092. [CrossRef]
46. Bengio, Y.; Léonard, N.; Courville, A.C. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv* **2013**, arXiv:1308.3432.
47. Sandler, M.; Howard, A.G.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
48. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
49. Chen, Y.h.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE J. Solid-State Circuits* **2017**, *52*, 127–138. [CrossRef]
50. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 248–255.
51. Tang, C.; Ouyang, K.; Wang, Z.; Zhu, Y.; Wang, Y.; Ji, W.; Zhu, W. Mixed-Precision Neural Network Quantization via Learned Layer-wise Importance. *arXiv* **2022**, arXiv:2203.08368. [CrossRef].
52. Park, E.; Yoo, S. PROFIT: A Novel Training Method for sub-4-bit MobileNet Models. *arXiv* **2020**, arXiv:2008.04693.
53. Wightman, R. PyTorch Image Models. 2019. Available online: https://github.com/rwightman/pytorch-image-models (accessed on 21 April 2022). [CrossRef]
54. Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. Master's Thesis, University of Toronto, Toronto, ON, Canada, 2009. Available online: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (accessed on 19 September 2021).
55. Chen, L.C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. *arXiv* **2018**, arXiv:1802.02611. [CrossRef].
56. Kimhi, M.; Kimhi, S.; Zheltonozhskii, E.; Litany, O.; Baskin, C. Semi-Supervised Semantic Segmentation via Marginal Contextual Information. *arXiv* **2023**, arXiv:2308.13900. [CrossRef].
57. Srivastava, N.; Jin, H.; Liu, J.; Albonesi, D.H.; Zhang, Z. MatRaptor: A Sparse-Sparse Matrix Multiplication Accelerator Based on Row-Wise Product. In Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Athens, Greece, 17–21 October 2020; pp. 766–780.