SpaceEvo: Hardware-Friendly Search Space Design for Efficient INT8 Inference

Xudong Wang*[1,2§] Li Lyna Zhang*[2‡] Jiahang Xu[2] Quanlu Zhang[2] Yujing Wang[3]
Yuqing Yang[2] Ningxin Zheng[2] Ting Cao[2] Mao Yang[2]

[1]Shanghai Jiao Tong University [2]Microsoft Research [3]Microsoft
xudongwang176@gmail.com; {lzhani, jiahangxu, quzha, yujwang, yuqyang, nizhen, ticao, maoyang}@microsoft.com
https://github.com/microsoft/Moonlit/tree/main/SpaceEvo

Abstract

The combination of Neural Architecture Search (NAS) and quantization has proven successful in automatically designing low-FLOPs INT8 quantized neural networks (QNN). However, directly applying NAS to design accurate QNN models that achieve low latency on real-world devices leads to inferior performance. In this work, we identify that the poor INT8 latency is due to the quantization-unfriendly issue: the operator and configuration (e.g., channel width) choices in prior art search spaces lead to diverse quantization efficiency and can slow down the INT8 inference speed. To address this challenge, we propose SpaceEvo, an automatic method for designing a dedicated, quantization-friendly search space for each target hardware. The key idea of SpaceEvo is to automatically search hardware-preferred operators and configurations to construct the search space, guided by a metric called Q-T score to quantify how quantization-friendly a candidate search space is. We further train a quantized-for-all supernet over our discovered search space, enabling the searched models to be directly deployed without extra retraining or quantization. Our discovered models, ***SEQnet****, establish new SOTA INT8 quantized accuracy under various latency constraints, achieving up to 10.1% accuracy improvement on ImageNet than prior art CNNs under the same latency. Extensive experiments on real devices show that SpaceEvo consistently outperforms manually-designed search spaces with up to 2.5× faster speed while achieving the same accuracy.*

1. Introduction

INT8 Quantization[27, 19, 10, 3] is a widely used technique for deploying DNNs on edge devices by reducing 4× in model size and memory cost for full-precision (FP32) models. However, prior art DNN models achieve only marginal speedup from INT8 quantization (in Fig. 2(a)), the still high latency after quantization making them difficult to deploy in latency-critical scenarios. Designing models that achieve high accuracy and low latency after quantization becomes the important but challenging problem.

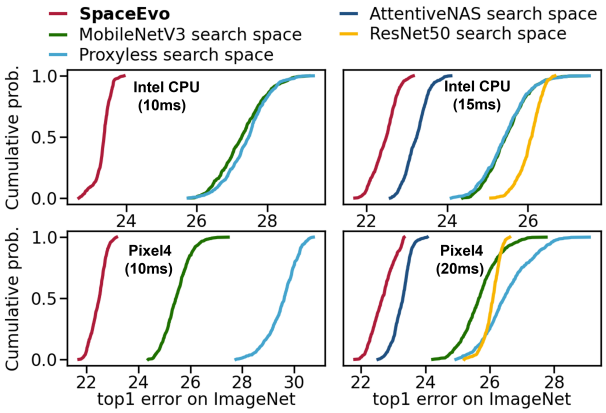


Figure 1. INT8 quantized model error distributions (EDF) of different NAS search spaces. We propose to *search* quantization-friendly search space for each hardware, which yields significantly better INT8 model populations than SOTA search spaces.

Neural Architecture Search (NAS) is a powerful tool for automating efficient quantized model design [36, 39, 12, 37, 6]. Recently, OQAT [32] and BatchQuant [1] achieve remarkable search efficiency and accuracy by adopting a two-stage paradigm. The first stage trains a weight-shared quantized supernet assembling all candidate architectures in the search space. This allows all the sub-networks (subnets) to simultaneously reach comparable quantized accuracy as when trained from scratch individually. The second stage uses typical search algorithms to find subnets with best quantized accuracy under different FLOPs constraints. This approach avoids the need to retrain each subnet for accuracy evaluation, greatly improving the search efficiency.

*Equal contribution
§Work was done during the internship at Microsoft Research
‡Corresponding author (lzhani@microsoft.com)

Though promising in optimizing model FLOPs, we identify a significant challenge when directly applying two-stage NAS to low quantized latency scenarios. This is due to the *quantization-unfriendly search space* issue, where prior art search spaces can unexpectedly *impede INT8 latency*. Consequently, since INT8 quantization yields only a marginal speedup, we are forced to search for small-sized models to fulfill latency criteria, which can unfortunately restrict NAS to find better quantized models for edge devices. Then, a question naturally arise: *Can we design a quantization-friendly search space, allowing NAS to discover larger and superior models that meet the low INT8 latency requirements?*

We start by conducting an in-depth study to understand the factors that determine INT8 quantized latency and how they affect search space design. Our study shows: *(1) both operator type and configurations (e.g., channel width) greatly impact the INT8 latency*; Improper selections can slow down the INT8 latency. For instance, Squeeze-and-Excitation (SE) [16] and Hardswish [14] are widely-used operators in current search spaces as it improves accuracy with little latency introduced. However, their INT8 inference speeds are *slower* than FP32 inference on Intel CPU (Fig. 3(a)), because the extra costs (*e.g.*, data transformation between INT32 and INT8) introduced by quantization outweigh the latency reduction by INT8 computation. *(2) The quantization efficiency varies across different devices, and the preferred operator types can be contradictory.*

The above study motivates us to design specialized quantization-friendly search spaces for each hardware, rather than relying on a single, large search space as seen in SPOS [12] for all hardware, which provides different operator options per layer. This is because two-stage NAS requires the search space to adhere to a specific condition for training the supernet, wherein each layer must utilize a fixed operator. Our study indicates significant variations in optimal operators across different hardware. Thus, customizing the search space for each hardware is crucial for optimal results. However, designing such specialized quantization-friendly search spaces for various edge devices presents a significant challenge, requiring expertise in both AI and hardware domains, as well as many trial and error attempts to optimize accuracy and INT8 latency for each hardware.

In this paper, we propose SpaceEvo, a novel method for automatically designing specialized quantization-friendly search space for each hardware. The search space is comprised of hardware-preferred operators and configurations, enabling the search of larger and better models with low INT8 latency. With the discovered search space, we leverage two-stage quantization NAS to train a quantized-for-all supernet, and utilize evolution search [4] to find best quantized models under various INT8 latency constraints. Our approach addresses three key challenges: (1) What is the definition of a quantization-friendly search space in terms of both quantized accuracy and latency? (2) How to automatically design a search space without human expertise? (3) How to handle with the prohibitive cost caused by quality evaluation of a candidate search space?

To address the first challenge, we propose a latency-aware *Q-T* score to quantify the effectiveness of a candidate search space, which measures the INT8 accuracy-latency quality of **top-tier** subnets in a search space. The behind intuition is that the goal of NAS is to search top subnets with better accuracy-latency tradeoffs.

Then, we introduce an evolutionary-based search algorithm that can effectively search a quantization-friendly search space with highest *Q-T* score. Searching a search space involves discovering *a collection of model population* that contains billions of models, which is challenging and easily introduce complexity. To address this challenge, we propose to factorize and encode a search space into a sequence of elastic stages, which have flexible operator types and configurations. Through this design, the task of search space design is then simplied to find a search space with the optimal elastic stages, so that existing search algorithms can be easily applied. Specifically, we design a stage-wise hyperspace to include many candidate search spaces and leverage aging evolution [30] to perform random mutations of elastic stages for search space evolution. The evolution is guided by maximizing the *Q-T* score.

Finally, estimating the quality score (Q-T score) of a search space involves a costly training process for evaluating the accuracy of sub-networks, which presents a significant obstacle for our evolutionary algorithm. Naively adopting a two-stage NAS approach, training a supernet for each candidate search space [8, 7], is prohibitively expensive, taking of thousands GPU hours. To address this issue, we draw inspiration from block-wise knowledge distillation [26, 22] and propose a *block-wise search space quantization scheme*. This scheme trains each elastic stage separately and rapidly estimates a model's quantized accuracy by summing block-level loss with a quantized accuracy lookup table, as shown in Fig. 5. This significantly reduces the training and evaluation costs, while providing effective accuracy rankings among search spaces. We summarize our contributions as follows:

- We study the INT8 quantization efficiency on real-world edge devices and find that the choices of operator types and configurations in a quantized model can significantly impact the INT8 latency, leaving a huge room for design optimization of quantized models.

- We propose SpaceEvo to automatically design a hardware-dedicated quantization-friendly search space and leverage two-stage quantization NAS to produce superior INT8 models under various latency constraints.

- We present three innovative techniques that enable the first-ever efficient and cost-effective evolution search to explore a search space comprising billions of models.
- Extensive experiments on two real-world edge devices and ImageNet demonstrate that our automatically designed search spaces significantly surpass manually-designed search spaces. Our discovered models, SEQnet, establish the new state-of-the-art INT8 quantized accuracy-latency tradeoffs. For instance, SEQnet@cpu-A4, achieves 80.0% accuracy on ImageNet, which is 3.3ms faster with 1.8% higher accuracy than FBNetV3-A. Moreover, SpaceEvo delivers superior tiny models, achieving up to 10.1% accuracy improvement over ShuffleNetV2x0.5 (41M FLOPs, 4.3ms).

2. Related Works

Quantization has been widely used for efficiency in deployment. Extensive efforts can be classified into post-training quantization (PTQ) [27, 2] and quantization-aware training (QAT) [19, 24, 10, 3]. QAT generally outperforms PTQ in quantizing compact DNNs to 8bit and very low-bit (2, 3, 4bit) by finetuning the quantized weights. Despite their success, traditional quantization methods focus on minimizing accuracy loss for a given pre-trained model, but ignore the real-world inference efficiency.

NAS for Quantization. Early works [36, 39, 12, 37, 6] formulates mixed-precision problem into NAS to search layer bit-width with a given architecture. Recently, [32, 1] train a quantized-for-all supernet to search both architecture and bit-width. The searched models can be directly evaluated with comparable accuracy to train-from-scratch. However, little attention is paid on optimizing quantized latency on real-world devices. Through searching quantization-friendly search space, our discovered quantized models can achieve both high accuracy and low latency.

Search Space Design. Starting from [5], the manually-designed MBConv-based space becomes the dominant in most NAS works [5, 4, 41, 35]. RegNet [29] is the first to present standard guidelines to optimize a search space by each dimension. Recently, [17, 28, 23, 40, 8, 7] propose to shrink to a better compact search space by either pruning unimportant operators or configurations. However, these works focus on optimizing the accuracy and little attention is paid on quantization-friendly search space design. Our work is the first lightweight solution towards this direction.

3. On-device Quantization Efficiency Analysis

To understand what factors lead to quantization-unfriendly issue, we conduct a comprehensive study on two widely-used edge devices equipped with high-performance inference engine: an Intel CPU device supported with VNNI instructions and onnxruntime [25] (Intel CPU) and a Pixel 4 phone CPU with TFLite 2.7 [11] (Pixel 4). Note that we follow existing practices [42, 34, 38] to measure the latency. We reveal key observations as follows:

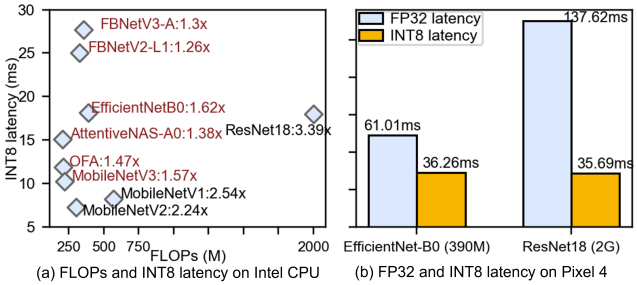


Figure 2. INT8 latency and speedups (annotated) for SOTA models. FLOPs and FP32 latency are not good indicators of INT8 latency; Compact models have very limited INT8 speedup ($\sim$1.5$\times$).

Observation 1: *FP32 latency and FLOPs are not good indicators of INT8 latency.*

To deploy on edge devices, a common belief is that a compact model with low FLOPs or FP32 latency is preferred than a larger model. However, Fig. 2 shows that neither of them is a good indicator of INT8 latency. In Fig. 2(b), a very large model (ResNet18) can be even faster than a compact model (EfficientNet-B0) after quantization. Moreover, the recent SOTA compact models searched by OFA [4] and AttentiveNAS [35] all have marginal INT8 speedups, suggesting that optimizing FLOPs and FP32 latency can not lead to lower INT8 latency.

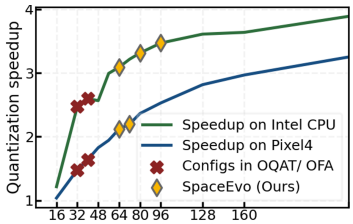
Observation 2: *The choices of operators' types and configurations greatly impact the INT8 latency.*

The prior art search spaces adopted in recent two-stage NAS works are MobileNetV2 or MobileNetV3 based chain-structures, where each search space comprises a sequence of blocks (stages). The block type is *fixed* to the MBConv and is allowed to search from a *handcraft range of hyper-parameter configurations* including kernel size, expansion ratio, channel width and depth, which are designed with human wisdom. For instance, many works [5, 35] observe that edge-regime CNNs prefer deeper depths and narrower channels, and manually set small channel numbers but large depths in the search space.

However, we find that many block type and configuration choices in current search spaces unexpectedly slow down the INT8 latency. We first study the operator type impact in Fig. 3(a). SE and Hardswish are lightweight operators in edge-regime search spaces, but their INT8 inference becomes slower on Intel CPU. Compared to Conv, DWConv can greatly reduce the FLOPs, but it benefits less from INT8 quantization. The root cause is that quantization introduces extra cost, such as (1) data transformation between INT32 and INT8 [34], and (2) additional computation caused by scaling fators and zero points [20]. If the operator has low data-reuse-rate, such as the activation functions (Hswish),

Operator	Intel CPU	Pixel 4
Conv	2.6x	2.5x
DWConv	1.2x	2.0x
SE	0.7x	1.4x
Hardswish	0.7x	0.5x
Swish	0.8x	2.1x

(a) Avg. INT8 latency speedup (relative to FP32) of major operators in MobileNetV3.



(b) Conv1x1 speedups

Figure 3. (a) The choice of operator type leads to significantly different quantized speedup. (b) Conv1x1 speedups with various channel numbers. Config: HW=28, C_{out}=4xC_{in}(expand=4).

the extra cost may outweigh the latency reduction by the low-bit computation. For high data-reuse operators (Conv), this cost is amortized and thus achieve large speedup [34].

Besides the operator type, the configuration choices also determine the quantization efficiency. Fig. 3(b) shows the speedups of Conv1$\times$1 under various channel widths. Results show that small channel widths in OFA search space cannot benefit well from quantization. This is because the additional quantization cost has a large impact when the channel width is small, limiting the latency acceleration. In contrast, SpaceEvo can automatically design a search space with larger channel widths for better efficiency.

Observation 3: *Quantization-friendly settings are diverse and contradictory across devices.*

In Fig. 3, we also observe that the quantization-friendly operators are different and can be contradictory on diverse devices. For instance, Swish achieves a 2$\times$ speedup on Pixel 4, but it is a quantization-unfriendly operator on CPU with a 0.8$\times$ slowdown. The reason is that quantization speedups are highly dependent on the inference engines and hardware [34, 21]. Intel VNNI supports the VPDPBWSD hardware instruction [18], which fuses three instructions into one to speedup INT8 computation. Without VNNI, INT8 hardly gains speedup on Intel CPUs. Moreover, the implementations in inference engines have to fully utilize hardware instructions for latency reduction. For example, onnxruntime does not implement a quantization kernel for Hardswish. Even on a VNNI-supported CPU, the use of Hardswish in a quantized model slows down the latency.

The need for hardware specialized search space. The above observations suggest that there is no single structure (block types in a model) that is optimal for quantization on all hardware. This poses a challenge for the two-stage NAS paradigm, as the supernet training requires all models in the search space to share an isomorphic structure. To address this issue, our work proposes to design a specialized quantization-friendly search space for each hardware. Each search space is tailored to the unique characteristics of the hardware and includes an optimal structure with elastic depths, widths, and kernel sizes.

4. Methodology

4.1. The Core Design Concept

In this section, we present our methodology for automatically designing a specialized quantization-friendly search space for any target hardware. Different from architecture search, where the goal is to find the single best model from the space, we aim to discover a *model population* that contains billions of accurate and INT8 latency-friendly architectures. We draw inspiration from the neural architecture search process and propose to use an evolutionary search algorithm to explore such a quantization-friendly model population. To achieve this, we introduce SpaceEvo, which is built on the following three techniques.

First, we need a metric that quantifies how quantization-friendly a candidate search space is. We define a Q-T score that is efficiently measured by top-tier subnets' INT8 accuracy-latency (Sec. 4.2).

Second, existing evolutionary search algorithms are designed for searching a single model architecture rather than a large search space encompassing billions of architectures. we propose a novel approach that we call the "elastic stage." By factorizing the search space into a sequence of elastic stages ((Sec. 4.3), we enable traditional aging evolution methods, such as the aging evolution [30], to be directly applied to search the space (Sec. 4.4).

Third, searching a search space with a maximum Q-T score can be prohibitively costly since the corresponding supernet must be trained from scratch for accuracy evaluation. We propose a block-wise search space quantization scheme to significantly reduce the training cost(Sec. 4.5).

4.2. Search Space Quality Score

Latency-aware space quality of Q-T score. Before space search, we need a score to quantify how quantization-friendly a search space is, which serves as the search objective. Since our ultimate goal is to search the best quantized models from the searched space, we treat a space with good quality if its **top-tier** subnets achieve optimal quantized accuracy under latency constraints T. Due to the fact that real-world applications usually have different deployment requirements, we use **multiple INT8 latency constraints** to measure a space's quality. For search space $\mathcal{A}$ and a set of quantized latency constraints $T_{1,...,n}$, we treat every constraint equally important, and define Q-T score as the sum of each constraint: $\mathcal{Q}(\mathcal{A}, T_{1,...,n})$= $\mathcal{Q}(\mathcal{A}, T_1)$+$\mathcal{Q}(\mathcal{A}, T_2)$+..., $\mathcal{Q}(\mathcal{A}, T_n)$, $\mathcal{Q}(\mathcal{A}, T_i)$ is defined as:

$$\mathcal{Q}(\mathcal{A}, T_i) = \mathbb{E}_{\alpha \in \mathcal{A}, LAT(\alpha) \leq T_i}[Acc_{int8}(\alpha)] \quad (1)$$

where α denotes a top-tier (best searched) subnet in $\mathcal{A}$ and $Acc_{int8}(\alpha)$ is its top-1 quantized accuracy evaluated on ImageNet validation set, $LAT(\alpha)$ predicts the quantized latency on target device.

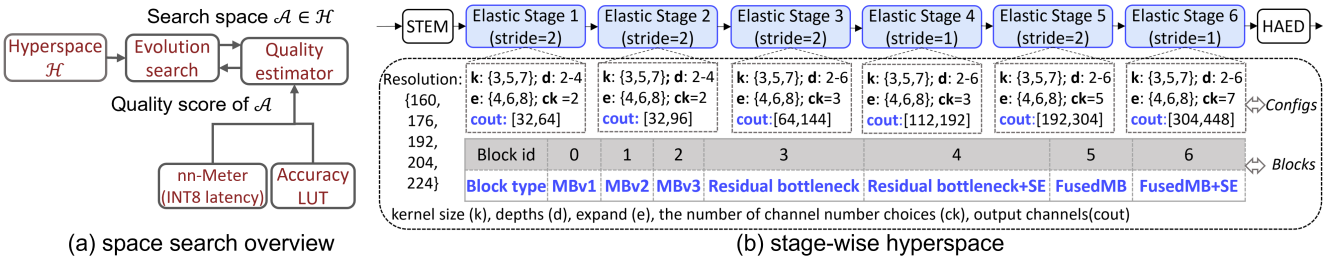


Figure 4. (a) We simplify space search into model search process; (b) Illustration of our hyperspace. A sampled search space is encoded by a sequential elastic stages. Contents in blue are searched: an elastic stage can search its block type and channel number list.

However, it's non-trivial to obtain the top-tier subnets from a candidate search, as if often involves an expensive full architecture search process. We adopt a *zero-cost* policy. Specifically, we randomly sample 5k subnets and select top 20 that under the latency constraints as the top-tier models to approximate the expectation term. The top 20 subnets are rapidly identified through the use of an accuracy look-up-table and a quantized latency predictor (Sec. 4.5).

4.3. Elastic Stage and Problem Formulation

We observe that existing two-stage NAS adopt a chain-structured search space [4, 41, 35], which can be factorized as a sequence of STEM, HEAD and N searchable stages. Each stage defines a range of configurations c (e.g., kernel size, channel width, depth) for a specific block type b, and allows NAS to find the optimal architecture settings.

Elastic stage. Without loss of generality, we define a stage structure in a search space as elastic stage $E_{b,c}$, which has elastic configurations c for a fixed block b. Suppose a search space $\mathcal{A}$ has N stages, it can be modularized as:

$$\mathcal{A} = STEM \circ E^1_{b,c}, ... \circ E^N_{b,c} \circ HEAD \tag{2}$$

For instance, the search space in OQAT [32] and BatchQuant [1] can be factorized as 6 elastic stages and STEM (first Conv) and a classification head. Each elastic stage represents a set of configuration choices for the MBv3 block. Through the definition of elastic stage, we can simply use Eq. (2) to denote the contents of a model popuation.

Problem definition. Operator type b and configuration c are two crucial objectives when searching quantization-friendly search space. Through the definition of elastic stage, the task of space search can be simplified to find a search space with optimal elastic stages, which has a similar goal with neural architecture search. We formulate our problem as:

$$\mathcal{A}(E^1_{b,c} \circ E^2_{b,c} \circ ... \circ E^N_{b,c})^* = \underset{E^i_{b,c} \in \mathcal{H}^i}{\arg\max}\, \mathcal{Q}(\mathcal{A}(E^1_{b,c} \circ E^2_{b,c} \circ ... \circ E^N_{b,c}), T) \tag{3}$$

where $\mathcal{A}(\cdot)$ denotes the search space, and $E^i_{b,c}$ is the i^{th} elastic stage of $\mathcal{A}(\cdot)$. $\mathcal{H}$ denotes the hyperspace which covers all possible search spaces. $\mathcal{Q}$ is the Q-T score for measuring a search space's quality.

As illustrated in Fig. 4(a), given the constraints T (i.e., a set of targeted quantized latency), SpaceEvo aims to find the optimal elastic stages $(E^1_{b,c} \circ E^2_{b,c} \circ ... \circ E^N_{b,c})^*$ from the 1^{st} to N^{th} stage for $\mathcal{A}^*$ that has the maximum Q-T score: the top-tier quantized models can achieve best accuracy under the constraint T. The constraint T can be the quantized latency under any bits. In this work, we focus on the latency of INT8 quantized models, because INT8 quantization is widely supported by real-world edge devices and can achieve real speedup. Other lower bits typically require specific hardware (e.g., FPGA) to get the latency.

4.4. Searching the Search Space

We now describe our evolutionary search algorithm that solves the problem in Eq. (3).

Hyperspace design. Analogous to NAS, hyperspace $\mathcal{H}$ defines which search space a search algorithm might discover. Defining a hyperspace to cover many candidate search spaces for space search is a second-order problem for NAS, which can easily introduce high complexity. Fortunately, we can easily construct a large hyperspace through search space modularization in Eq. (2).

We construct a large hyperspace in Fig. 4, in which a search space can be encoded by N=6 sequential elastic stages along with STEM and HEAD. We search the following two dimensions for an elastic stage:

- Block (operator) type b: MBv1 [15], MBv2 [31], MBv3 [14], residual bottleneck [13], residual bottleneck with SE, FusedMB [33] and FusedMB with SE. Conv is the major operator in residual bottleneck and FusedMB, thus they are quantization-friendly blocks; the efficiency of MB blocks relies on the device. For example, DWConv and SE are less quantization-efficient on Intel CPU.

- Output channel width list *cout*. In Sec. 3, we observe that quantized models can better utilize hardware under a larger channel number setting. However, directly increasing the channel numbers will also lead to longer latency. Therefore, we search the optimal stage-wise channel width list *cout*: $\{w^*_{min}, ..., w^*_{max}\}$, which provides better channel width choices for final INT8 model search.

Specifically, as described in Fig. 4(b), we define a wide range of $[w_{min}, w_{max}]$ by enlarging the channel widths in existing search spaces, and allow each elastic stage to choose a subset of *cout* from $[w_{min}, w_{max}]$.

Besides channel widths, other configuration dimensions (e.g., kernel size) also impact a model's quantized latency. However, searching all dimensions leads to a large amount of choices in one stage, which exponentially enlarges the hyperspace size. Fortunately, other dimensions usually have a small space (e.g., kernel size selects from {3,5,7}). It's easy to find the optimal value for a model in the final NAS process. Therefore, we follow existing practices to configure the choices of kernel size, depth, and expand ratios. Our final searched INT8 model architectures SEQnet suggest that the optimal quantization-friendly kernel sizes and expand ratios are chosen. For example, kernel size of 3×3 brings more INT8 latency speedups for DWConv, and it is the dominate kernel size choice in DWConv related blocks.

Suppose that a stage has m choices of channel widths, there would be 7 (operator types) $\times m$ candidates for each stage. In total, for a typical search space with $N = 6$ stages, the hyperspace has $\sim 10^9$ candidate search spaces, which is extremely large and poses challenges for efficient search.

Evolutionary space search. The structure of hyperspace is similar to a typical model search space in NAS [12], so we can easily apply existing NAS search algorithms. Taking this advantage, we leverage aging evolution [30] to search the large hyperspace. We first randomly initialize a population of P search spaces, where each sampled space is encoded as $(E^1_{b,c} \circ E^2_{b,c} \circ ... \circ E^N_{b,c})$. Each individual is rapidly evaluated with Q-T score. After this, evolution improves the initial population in mutation iterations. At each iteration, we sample S random candidates from the population and select the one with highest score as the *parent*. Then we alternately mutate the *parent* for block type and widths to generate two *children* search spaces. For instance, suppose the i^{th} stage $E^i_{b,c}$ is selected for mutation, we first randomly modify its block type and produce $E^i_{b^*,c}$ for *child 1*, then we mutate the widths and produce E^i_{b,c^*} for *child 2*. We evaluate their Q-T scores and add them to current population. The oldest two are removed for next iteration. After all iterations finish, we collect all the sampled space and select the one with best score as the final search space.

4.5. Efficient Search Space Quality Evaluation

We now address the efficiency challenge caused by Q-T score evaluation. The most accurate evaluation is to get accuracy by training a supernet (search space) from scratch and measure latency on target device. However, it's impractical to conduct large-scale search due to the prohibitive cost. For example, it costs more than 10 days to train a supernet on 8 V100 GPUs [41]. To reduce the cost, we build an accurate INT8 latency predictor by nn-Meter [42], then we introduce block-wise quantization scheme.

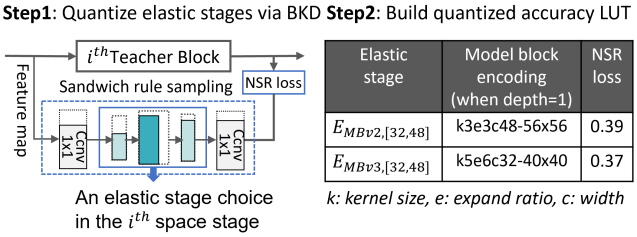


Figure 5. We adopt block-wise knowledge distillation to reduce search space quality evaluation cost. We add two linear transformation layers (Conv 1x1) to match teacher's feature map widths.

Block-wise knowledge distillation (BKD) is firstly proposed in DNA [22] and then further improved in DONNA [26]. It originally uses block-wise representation of existing models (teacher) to supervise a corresponding student model block. This technique can provide a relative accuracy ranking of all possible models without requiring them to be trained from scratch. In our work, we extend BKD to supervise the training of all elastic stages (each contains a large amount of blocks).

Fig. 5 illustrates the BKD process. In the first step, we use EfficientNet-B5 as the teacher, and separately train each elastic stage to mimic the behavior of corresponding teacher block by minimizing the NSR loss [26] between their output feature maps. Specifically, the i^{th} stage receives the output of $(i-1)^{th}$ teacher block as the input and is optimized to predict the output of i^{th} teacher block with NSR loss. Since an elastic stage contains many blocks with different channel widths, we add two learnable linear transformation layers at the input and output for each elastic stage to match teacher's feature map shape. Moreover, we adopt sandwich rule [41] to sample four paths to improve the training efficiency. Each elastic stage is firstly trained for 5 epochs and then performed 1 epoch LSQ+ [3] for INT8 quantization.

Accuracy lookup table. In the second step, we construct a INT8 accuracy lookup table to reduce the evaluation cost. Specifically, we evaluate all possible blocks in each elastic stage and record their NSR losses on the validation set in the lookup table. The quantized loss of a model is estimated by summing up the NSR loss of all its blocks by rapidly looking up each elastic stages from the table. We inverse the measured loss to approximate the quantized accuracy for Q-T score evaluation.

In our work, the BKD and lookup table construction can be sped up in a parallel way and finished in 1 day, which amounts a one-time cost before aging evolution search.

5. Evaluation

Setup. We evaluate our method on ImageNet-1k dataset [9] and two popular edge devices. The INT8 latency constraints are {8, 10, 15, 20, 25} *ms* for Intel CPU, and {15, 20, 25, 30, 35} *ms* for Pixel4. For each device, we search 5k search

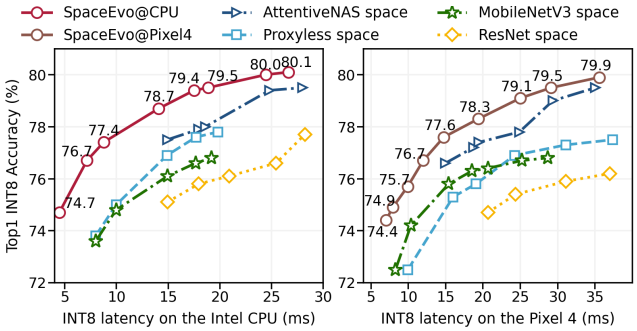


Figure 6. Best searched INT8 models with comparison to state-of-the-art NAS search spaces. Our searched spaces are proven to be the most quantization-friendly for the target device.

spaces in total and return the one with highest Q-T score. The population size P is 500 and sample size S is 125.

Once SpaceEvo discovers a quantization-friendly search space for the target device, we train a *quantized-for-all* supernet. We start by pretraining a full-precision supernet without quantizers on ImageNet for 360 epochs. We adopt the sandwich rule and inplace distillation in BigNAS [41]. Then, we perform quantization-aware training (QAT) on the trained supernet for 50 epochs, which follows the same training protocol (i.e., sandwich rule and inplace distillation). We use LSQ+ as the QAT algorithm for better quantized accuracy. To derive INT8 model for deployment, we use the evolutionary search in OFA [4] to search 5k models for various given INT8 latency constraints. We list out detailed training settings in supplementary materials. In the following, we refer to the two searched spaces as *SpaceEvo@CPU* and *SpaceEvo@Pixel4*, the searched model families are *SEQnet@cpu* and *SEQnet@pixel4*.

5.1. The Effectiveness of SpaceEvo

Comparison with SOTA search spaces. To demonstrate the high-performance of our searched spaces, we compare with prior art manually-designed search spaces including: (1) MobileNetV3 search space that is adopted in two-stage quantization NAS OQAT [32] and BatchQuant [1]; (2) ProxylessNAS and AttentiveNAS search spaces that achieve superior performance on mobile devices; and (3) ResNet50 search space proposed by OFA that is a handcraft quantization-friendly space on our two devices. For fair comparison, we use one supernet training and QAT receipt for all search spaces. We conduct evolutionary search to compare the best-searched models from each search space. We use the random seed of 0. For all experiments, search space is the only difference.

Fig. 6 compares the best searched INT8 models from different search spaces. SpaceEvo@CPU and SpaceEvo@Pixel4 consistently deliver superior quantized models than state-of-the-art search spaces. Under the same latency, the best quantized models from SpaceEvo@cpu significantly surpass the existing state-of-the-art search

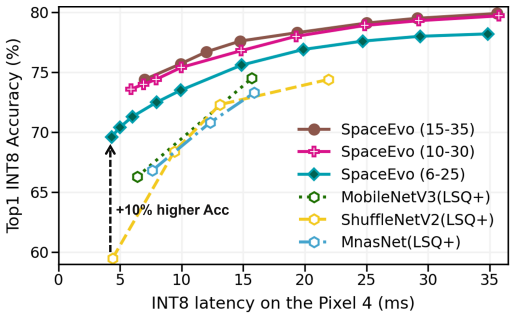


Figure 7. Search space design under diverse INT8 latency constraints. SpaceEvo (6-25 ms) delivers superior tiny INT8 models.

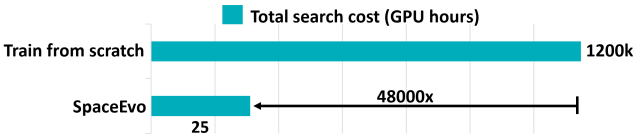


Figure 8. Search cost measured on 8 Nvidia V100 GPUs.

spaces with +0.7% to +3.8% (+0.4% to +3.2% on Pixel4) higher accuracy. Moreover, our search space is the only one that is able to deliver superior quantized models under both extremely low (only∼5ms) and large latency constraints.

SpaceEvo under diverse latency constraints. We extensively study the effectiveness of SpaceEvo under different latency constraints. Specifically, we perform space search under two tight constraints of {10, 15, 20, 25, 30}ms and {6, 10, 15, 20, 25}ms on Pixel4. The results are shown in Fig. 7. Our proposed method can handle the diverse latency requirements and produce high-quality spaces. As expected, the searched spaces under 10-30ms and 6-25ms have much more low-latency quantized models.

To further verify the effectiveness of these low-latency models, we compare with existing SOTA tiny models. Significantly, even under the extremely low latency constraints of 6-25 ms, our searched space delivers very competitive tiny quantized models. Compared to the smallest model ShuffleNetV2x0.5, we can achieve +10.1% higher accuracy under the same latency of 4.3 ms.

Search cost. As depicted in Fig. 8, our algorithm, SpaceEvo, is designed to be lightweight and suitable for real-world usage, requiring only 25 GPU hours to search a space of 5000 iterations. This remarkable speed is mainly due to our block-wise search space quantization scheme, which significantly reduces the cost of search space quality evaluation. In comparison, Fig. 8 demonstrates that training each search space from scratch without this scheme would consume an impractical 1200k GPU hours.

5.2. The Effectiveness of Discovered INT8 Models

In this section, we demonstrate that our searched spaces deliver state-of-the-art quantized models. We compare with two strong baselines: (1) *prior art manually-designed and NAS-searched models*; and (2) *quantization-aware NAS*. For

Table 1. ImageNet results compared with SOTA quantized models on two devices. *: latency compared to FP32 inference.

(a) Results on the Intel VNNI CPU with onnxruntime

Model	Acc% INT8	CPU Latency INT8	CPU Latency speedup*	Acc% FP32	FLOPs
MobileNetV3Small	66.3	4.4 ms	1.1×	67.4	56M
SEQnet@cpu-A0	**74.7**	**4.4 ms**	**2.0**×	74.8	163M
MobileNetV2	71.4	7.3 ms	2.2×	72.0	300M
ProxylessNAS-R	74.6	8.8 ms	1.8×	74.6	320M
OQAT-8bit	74.8	9.8 ms	1.8×	75.2	214M
MobileNetV3Large	74.5	10.3 ms	1.5×	75.2	219M
OFA (#25)	75.6	11.2 ms	1.5×	76.4	230M
SEQnet@cpu-A1	**77.4**	**8.8 ms**	**2.4**×	77.5	358M
APQ-8bit	73.6	15.0 ms	1.5×	73.6	297M
AttentiveNAS-A0	76.1	15.1 ms	1.4×	77.3	203M
OQAT-8bit	76.3	14.9 ms	1.7×	76.7	316M
EfficientNet-B0	76.7	18.1 ms	1.6×	77.3	390M
SEQnet@cpu-A2	**78.5**	**14.1 ms**	**2.4**×	78.8	638M
APQ-8bit	74.9	20.0 ms	1.5×	75.0	393M
OQAT-8bit	76.9	19.5 ms	1.6×	77.3	405M
AttentiveNAS-A1	77.2	22.4 ms	1.4×	78.4	279M
AttentiveNAS-A2	77.5	22.5 ms	1.3×	78.8	317M
SEQnet@cpu-A3	**79.5**	**18.9 ms**	**2.6**×	79.6	981M
FBNetV2-L1	75.8	25.0 ms	1.2×	77.2	325M
FBNetV3-A	78.2	27.7 ms	1.3×	79.1	357M
SEQnet@cpu-A4	**80.0**	**24.4 ms**	**2.4**×	80.1	1267M

(b) Results on the Google Pixel 4 with TFLite

Model	Acc% INT8	Pixel4 Latency INT8	Pixel4 Latency speedup*	Acc% FP32	FLOPs
MobileNetV3Small	66.3	6.4 ms	1.3×	67.4	56M
SEQnet@pixel4-A0	**73.6**	**5.9 ms**	**2.1**×	73.7	107M
MobileNetV2	71.4	16.5 ms	1.9×	72.0	300M
ProxylessNAS-R	74.6	18.4 ms	1.8×	74.6	320M
MobileNetV3Large	74.5	15.7 ms	1.5×	75.2	219M
APQ-8bit	74.6	14.9 ms	2.0×	74.4	340M
OFA (#25)	75.6	14.8 ms	1.7×	76.4	230M
OQAT-8bit	75.8	15.2 ms	1.9×	76.2	287M
AttentiveNAS-A0	76.1	15.2 ms	2.0×	77.3	203M
SEQnet@pixel4-A1	**77.6**	**14.7 ms**	**2.2**×	77.7	274M
APQ-8bit	75.1	20.0 ms	1.9×	75.1	398M
OQAT-8bit	76.5	20.4 ms	1.8×	76.8	347M
AttentiveNAS-A1	77.2	21.1 ms	2.0×	78.4	279M
AttentiveNAS-A2	77.5	22.7 ms	2.0×	78.8	317M
SEQnet@pixel4-A2	**78.3**	**19.4 ms**	**2.3**×	78.4	402M
FBNetV2-L1	75.8	26.7 ms	1.5×	77.2	325M
OQAT-8bit	77.0	29.9 ms	1.7×	77.2	443M
FBNetV3-A	78.2	30.5 ms	1.5×	79.1	357M
SEQnet@pixel4-A3	**79.5**	**30.8 ms**	**2.1**×	79.5	591M
EfficientNet-B0	76.7	36.4 ms	1.7×	77.3	390M
SEQnet@pixel4-A4	**79.9**	**35.5 ms**	**2.2**×	80.0	738M

baseline (1), we collect official pre-trained FP32 checkpoints and conduct LSQ+ QAT to get the quantized accuracy. The hyperparameter settings follow the original LSQ+ paper, except that we set a larger epoch of 10 to achieve better accuracy. The latency numbers are measured on our devices. For (2), we compare with strong baselines including APQ [37] and OQAT [32]. Specifically, we limit APQ to search for the fixed 8bit (INT8) models. Since OQAT has no 8bit supernet checkpoint, we follow the official source code and conduct supernet QAT for 50 epochs. The final INT8 models are searched under the same INT8 latency predictors for fair comparison.

Results. Table 1 summarizes comparison results. Remarkably, our searched model family, SEQnet significantly outperform SOTA efficient models and quantization-aware NAS searched models, with higher INT8 quantized accuracy, lower INT8 latency and better speedups. Without finetuning, our tiny models SEQnet@cpu-A0 and SEQnet@pixel4-A0 achieve 74.7% and 73.6% top1 accuracy on ImageNet, which is 8.4% and 7.3% higher than MobileNetV3-Small (56M FLOPs) while maintaining the same level quantized latency. For larger models, SEQnet@cpu-A4 (80.0%) outperforms FBNetV3-A with 1.8% higher accuracy while runs 3.3ms faster. In particular, to achieve the same level accuracy (i.e., around 77.2%), AttentiveNAS-A1 has 22.4ms latency while SEQnet@cpu-A1 (77.4%) only needs 8.8 ms (2.6 × faster). More importantly, our searched models can better utilize the INT8 hardware optimizations: the latency speedups compared to full-precision inference are all larger than 2×, and this leaves room to search large-size models with higher accuracy.

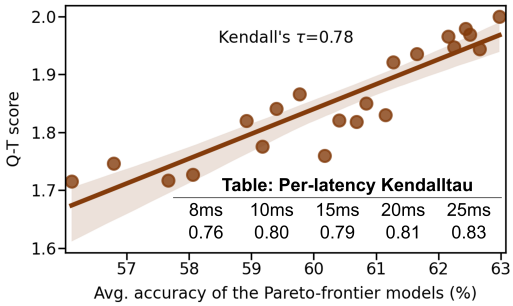


Figure 9. Q-T score effectiveness (Kendall's τ) on ranking search space quality. We achieve a high space ranking correlation.

Table 2. Different space search methods and their best resulting quantized models on Pixel4. Baseline is a SOTA mobile-friendly AttentiveNAS space. *: the search dimension use the same settings in AttentiveNAS.

Method	Op	Width	Best quantized models 10ms	15ms	20ms	30ms	36ms
Baseline	-	-	-	76.6	77.2	79.0	79.5
SpaceEvo-op	search	fix*	75.0	76.6	77.8	78.6	78.8
SpaceEvo-width	fix*	search	75.4	77.4	78.0	79.1	79.5
SpaceEvo	search	search	**75.7**	**77.6**	**78.3**	**79.5**	**79.9**

5.3. Ablation Study

Q-T score effectiveness. Q-T score is crucial as it guides the space evolution process. To evaluate its effectiveness, we randomly sample 30 search spaces, and measure the rank correlation (Kendall's τ) between their Q-T score and their actual Pareto-frontier models' accuracies. Specifically, we use Intel CPU as the test device and set a same latency constraints of {8, 10, 15, 20, 25}ms. For each sampled space, we train it from scratch for 50 epochs, and conduct evolutionary search to get the Pareto-frontier models' accuracies. As shown in Fig. 9, the Kendall's τ between the Q-T score and the actual Pareto-frontier models' accuracies is 0.8, which indicates a very high rank correlation.

Ablation study on two search dimensions. In Sec. 3, we conclude that operator type and configuration are two key factors impacting INT8 latency, which serves as the two search objectives of SpaceEvo. To verify the effectiveness, we create two strong baselines based on the SOTA edge-regime AttentiveNAS search space: (1) SpaceEvo-op: we fix each elastic stage's width to AttentiveNAS space, then allow each elastic stage to search for the optimal operator; and (2) SpaceEvo-width: we fix all elastic stages' block types to AttentiveNAS space, then search for the optimal width. Table 2 reports the space comparison between different search methods on the Pixel4. By searching both operator type and width, SpaceEvo finds the optimal search space where its best searched quantized models achieve the highest accuracy under all latency constraints. Moreover, even searching for one dimension, SpaceEvo-op and SpaceEvo-width outperform the manually-designed AttentiveNAS space under small latency constraints.

Search space design implications. We now summarize our learned experience and implications for designing quantization-friendly search spaces. We notice that the searched spaces show different preferences when targeting different devices: (i) All stages should use much wider channel widths compared to existing manually-designed spaces on the cpu, while only early stages prefer wider channels on Pixel 4. (ii) Since SE and Swish are INT8 latency-friendly on mobile phones, so our auto-generated search spaces for Pixel4 have many MBv3 stages. On Intel CPU, INT8 quantization slows down SE, Hardwish, and Swish, making FusedMB and MBv2 the priority for search spaces, with only the last two stages using MBv3. The details are provided in supplementary.

6. Conclusion

In this paper, we introduced SpaceEvo, the first to automatically design a quantization-friendly space for target device, which delivers superior INT8 quantized models with SOTA efficiency on real-world edge devices. Extensive experiments on two popular devices demonstrate its effectiveness compared to prior art manual-designed search spaces. We plan to apply SpaceEvo to other hardware efficiency such as energy-efficient search space design in the future.

References

[1] Haoping Bai, Meng Cao, Ping Huang, and Jiulong Shan. Batchquant: Quantized-for-all architecture search with robust quantizer. In *NeurIPS*, 2021. 1, 3, 5, 7

[2] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, 2019. 3

[3] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *CVPRW*, 2020. 1, 3, 6

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020. 2, 3, 5, 7

[5] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 3

[6] Zhaowei Cai and Nuno Vasconcelos. Rethinking differentiable search for mixed-precision neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2346–2355. Computer Vision Foundation / IEEE, 2020. 1, 3

[7] Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, and Haibin Ling. Searching the search space of vision transformers. In *NeurIPS*, 2021. 2, 3

[8] Yuanzheng Ci, Chen Lin, Ming Sun, Boyu Chen, Hongwen Zhang, and Wanli Ouyang. Evolving search space for neural architecture search. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6639–6649, 2021. 2, 3

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009. 6

[10] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *ICLR*, 2020. 1, 3

[11] Google. Tensorflow lite, 2022. 3

[12] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020. 1, 2, 3, 6

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 5

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *ICCV*, 2019. 2, 5

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 5

[16] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 2

[17] Yiming Hu, Yuding Liang, Zichao Guo, Ruosi Wan, Xiangyu Zhang, Yichen Wei, Qingyi Gu, and Jian Sun. Angle-based search space shrinking for neural architecture search. In *ECCV*. 3

[18] Intel. Intel 64 and ia-32 architectures software developer's manual, 2022. 4

[19] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 1, 3

[20] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018. 3

[21] Sumin Kim, Gunju Park, and Youngmin Yi. Performance evaluation of int8 quantized inference on mobile gpus. *IEEE Access*, 9:164245–164255, 2021. 4

[22] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020. 2, 6

[23] Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot nas by suppressing the posterior fading. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 3

[24] Christos Louizos, Mathias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *ICLR*, 2019. 3

[25] Microsoft. onnxruntime, 2022. 3

[26] Bert Moons, Parham Noorzad, Andrii Skliar, Giovanni Mariani, Dushyant Mehta, Chris Lott, and Tijmen Blankevoort. Distilling optimal neural networks: Rapid search in diverse spaces. In *ICCV*, 2021. 2, 6

[27] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *ICCV*, 2019. 1, 3

[28] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. Asap: Architecture search, anneal and prune. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pages 493–503, 2020. 3

[29] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020. 3

[30] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 2, 4, 6

[31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 5

[32] Mingzhu Shen, Feng Liang, Ruihao Gong, Yuhang Li, Chuming Li, Chen Lin, Fengwei Yu, Junjie Yan, and Wanlin Ouyang. Once quantization-aware training: High performance extremely low-bit architecture search. In *ICCV*, 2021. 1, 3, 5, 7, 8

[33] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning*, pages 10096–10106. PMLR, 2021. 5

[34] Xiaohu Tang, Shihao Han, Li Lyna Zhang, Ting Cao, and Yunxin Liu. To bridge neural network design and real-world performance: A behaviour study for neural networks. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 21–37, 2021. 3, 4

[35] Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *Conference on Computer Vision and Pattern Recognition*, 2021. 3, 5

[36] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *CVPR*, 2019. 1, 3

[37] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 1, 3, 8

[38] Xudong Wang, Li Lyna Zhang, Yang Wang, and Mao Yang. Towards efficient vision transformer inference: A first study of transformers on mobile devices. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, HotMobile '22, page 1–7. Association for Computing Machinery, 2022. 3

[39] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. 2019. 1, 3

[40] Xin Xia, Xuefeng Xiao, Xing Wang, and Min Zheng. Progressive automatic design of search space for one-shot neural architecture search. In *WACV*, 2022. 3

[41] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *ECCV*, 2020. 3, 5, 6, 7

[42] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, page 81–93, New York, NY, USA, 2021. ACM. 3, 6