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

Xudong Wang*1,2†, Li Lyna Zhang*2‡, Jiahang Xu2, Quanlu Zhang2, Yuqing Wang3, Yujing Wang2, Ningxin Zheng2, Ting Cao2, Mao Yang2

1Shanghai Jiao Tong University 2Microsoft Research 3Microsoft

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)

†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-dedicated quantization-friendly search space.

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 simplified 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 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, 5]. 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:

**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 hyperparameter 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 factors and zero points [20]. If the operator has low data-reuse-rate, such as the activation functions (Hswish),
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 Conv 1×1 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× speedup on Pixel 4, but it is a quantization-unfriendly operator on CPU with a 0.8× 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 \(T_1, \ldots, n\), we treat every constraint equally important, and define Q-T score as the sum of each constraint: \(Q(\mathcal{A}, T_1, \ldots, n) = Q(\mathcal{A}, T_1) + Q(\mathcal{A}, T_2) + \ldots + Q(\mathcal{A}, T_n)\). \(Q(\mathcal{A}, T_i)\) is defined as:

\[
Q(A, T_i) = E_{\alpha \in \mathcal{A}}.LAT(\alpha) \leq T_i [Acc_{INT8}(\alpha)]
\]

(1)

where \(\alpha\) 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.
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 search structure in a search space as elastic stage E_{b,c}. Suppose a search space A has N stages, it can be modularized as:

\[ A = STEM \circ E_{b,c}^1 \circ ... \circ E_{b,c}^N \circ HEAD \]  

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 population.

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 NAS, which can easily introduce high complexity. Fortunately, we can easily construct a large hyperspace through search space modularization in Eq. (2).

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, DWCConv 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 \( \{w_{\text{min}}, \ldots, w_{\text{max}}\} \), which provides better channel width choices for final INT8 model search.
Specifically, as described in Fig. 4(b), we define a wide range of $[w_{\text{min}}, w_{\text{max}}]$ by enlarging the channel widths in existing spaces, and allow each elastic stage to choose a subset of $cout$ from $[w_{\text{min}}, w_{\text{max}}]$. 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 enforces 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 $3 \times 3$ 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 $N$ search spaces, where each sampled space is encoded as $(E_1^{b,c} \circ E_2^{b,c} \circ \ldots \circ E_N^{b,c})$. 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 $E_{b,c}^i$ is selected for mutation, we first randomly modify its block type and produce $E_{b',c}^i$ for child 1, then we mutate the widths and produce $E_{b,c'}^i$ 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.

**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.
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 \textit{SpaceEvo@CPU} and \textit{SpaceEvo@Pixel4}, the searched model families are \textit{SEQnet@cpu} and \textit{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. \textit{SpaceEvo@CPU} and \textit{SpaceEvo@Pixel4} consistently deliver superior quantized models than state-of-the-art search spaces. Under the same latency, the best quantized models from \textit{SpaceEvo@cpu} significantly surpass the existing state-of-the-art search 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.

\textbf{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.

\textbf{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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc%</th>
<th>INTR</th>
<th>INTR speedup*</th>
<th>FP32 FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV3Small</td>
<td>66.3</td>
<td>4.4 ms</td>
<td>1.1 x</td>
<td>67.4</td>
</tr>
<tr>
<td>SEQNet@cpu-A0</td>
<td>74.7</td>
<td>4.4 ms</td>
<td>2.0 x</td>
<td>74.8</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>71.4</td>
<td>7.3 ms</td>
<td>2.2 x</td>
<td>72.0</td>
</tr>
<tr>
<td>ProxylessNAS-R</td>
<td>74.6</td>
<td>8.8 ms</td>
<td>1.8 x</td>
<td>74.6</td>
</tr>
<tr>
<td>OQAT-8bit</td>
<td>74.8</td>
<td>9.8 ms</td>
<td>1.8 x</td>
<td>75.2</td>
</tr>
<tr>
<td>MobileNetV3Large</td>
<td>74.5</td>
<td>10.3 ms</td>
<td>1.5 x</td>
<td>75.2</td>
</tr>
<tr>
<td>OFA (#25)</td>
<td>75.6</td>
<td>11.2 ms</td>
<td>1.5 x</td>
<td>76.4</td>
</tr>
<tr>
<td>SEQNet@cpu-A1</td>
<td>77.4</td>
<td>8.8 ms</td>
<td>2.4 x</td>
<td>77.5</td>
</tr>
<tr>
<td>APQ-8bit</td>
<td>73.6</td>
<td>15.0 ms</td>
<td>1.5 x</td>
<td>73.6</td>
</tr>
<tr>
<td>AttentiveNAS-A0</td>
<td>76.1</td>
<td>15.1 ms</td>
<td>1.4 x</td>
<td>77.3</td>
</tr>
<tr>
<td>OQAT-8bit</td>
<td>76.3</td>
<td>14.9 ms</td>
<td>1.7 x</td>
<td>76.7</td>
</tr>
<tr>
<td>EfficientNet-B0</td>
<td>76.7</td>
<td>18.1 ms</td>
<td>1.6 x</td>
<td>77.3</td>
</tr>
<tr>
<td>SEQNet@cpu-A2</td>
<td>78.5</td>
<td>14.1 ms</td>
<td>2.4 x</td>
<td>78.8</td>
</tr>
<tr>
<td>APQ-8bit</td>
<td>74.9</td>
<td>20.0 ms</td>
<td>1.5 x</td>
<td>75.0</td>
</tr>
<tr>
<td>AttentiveNAS-A1</td>
<td>76.9</td>
<td>19.5 ms</td>
<td>1.6 x</td>
<td>77.3</td>
</tr>
<tr>
<td>OQAT-8bit</td>
<td>77.2</td>
<td>22.2 ms</td>
<td>1.4 x</td>
<td>78.4</td>
</tr>
<tr>
<td>AttentiveNAS-A2</td>
<td>77.5</td>
<td>22.5 ms</td>
<td>1.3 x</td>
<td>78.8</td>
</tr>
<tr>
<td>SEQNet@cpu-A3</td>
<td>79.5</td>
<td>18.9 ms</td>
<td>2.6 x</td>
<td>79.6</td>
</tr>
<tr>
<td>FBNetV2-L1</td>
<td>75.8</td>
<td>25.0 ms</td>
<td>1.2 x</td>
<td>77.2</td>
</tr>
<tr>
<td>FBNetV3-A</td>
<td>78.2</td>
<td>27.7 ms</td>
<td>1.3 x</td>
<td>79.1</td>
</tr>
<tr>
<td>SEQNet@cpu-A4</td>
<td>80.0</td>
<td>24.4 ms</td>
<td>2.4 x</td>
<td>80.1</td>
</tr>
</tbody>
</table>

(b) Results on the Google Pixel 4 with TFLite

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc%</th>
<th>Pixel4 Latency</th>
<th>INTR speedup*</th>
<th>FP32 FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV3Small</td>
<td>66.3</td>
<td>6.4 ms</td>
<td>1.3 x</td>
<td>67.4</td>
</tr>
<tr>
<td>SEQNet@pixel4-A0</td>
<td>73.6</td>
<td>5.9 ms</td>
<td>2.1 x</td>
<td>73.7</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>71.4</td>
<td>16.5 ms</td>
<td>1.9 x</td>
<td>72.0</td>
</tr>
<tr>
<td>ProxylessNAS-R</td>
<td>74.6</td>
<td>18.4 ms</td>
<td>1.8 x</td>
<td>74.6</td>
</tr>
<tr>
<td>MobileNetV3Large</td>
<td>74.5</td>
<td>15.7 ms</td>
<td>1.5 x</td>
<td>75.2</td>
</tr>
<tr>
<td>APQ-8bit</td>
<td>74.6</td>
<td>14.9 ms</td>
<td>2.0 x</td>
<td>74.4</td>
</tr>
<tr>
<td>OFA (#25)</td>
<td>75.6</td>
<td>14.8 ms</td>
<td>1.7 x</td>
<td>76.4</td>
</tr>
<tr>
<td>OQAT-8bit</td>
<td>75.8</td>
<td>15.2 ms</td>
<td>1.9 x</td>
<td>76.2</td>
</tr>
<tr>
<td>AttentiveNAS-A0</td>
<td>76.1</td>
<td>15.2 ms</td>
<td>2.0 x</td>
<td>77.3</td>
</tr>
<tr>
<td>SEQNet@pixel4-A1</td>
<td>77.6</td>
<td>14.7 ms</td>
<td>2.2 x</td>
<td>77.7</td>
</tr>
<tr>
<td>APQ-8bit</td>
<td>75.1</td>
<td>20.0 ms</td>
<td>1.9 x</td>
<td>75.1</td>
</tr>
<tr>
<td>OQAT-8bit</td>
<td>76.5</td>
<td>20.4 ms</td>
<td>1.8 x</td>
<td>76.8</td>
</tr>
<tr>
<td>AttentiveNAS-A1</td>
<td>77.2</td>
<td>21.1 ms</td>
<td>2.0 x</td>
<td>78.4</td>
</tr>
<tr>
<td>AttentiveNAS-A2</td>
<td>77.5</td>
<td>22.7 ms</td>
<td>2.0 x</td>
<td>78.4</td>
</tr>
<tr>
<td>SEQNet@pixel4-A2</td>
<td>78.3</td>
<td>19.4 ms</td>
<td>2.3 x</td>
<td>78.4</td>
</tr>
<tr>
<td>FBNetV2-L1</td>
<td>75.8</td>
<td>26.7 ms</td>
<td>1.5 x</td>
<td>77.2</td>
</tr>
<tr>
<td>OQAT-8bit</td>
<td>77.0</td>
<td>29.9 ms</td>
<td>1.7 x</td>
<td>77.2</td>
</tr>
<tr>
<td>FBNetV3-A</td>
<td>78.2</td>
<td>30.5 ms</td>
<td>1.5 x</td>
<td>79.1</td>
</tr>
<tr>
<td>SEQNet@pixel4-A3</td>
<td>79.5</td>
<td>30.8 ms</td>
<td>2.1 x</td>
<td>79.5</td>
</tr>
<tr>
<td>EfficientNet-B0</td>
<td>76.7</td>
<td>36.4 ms</td>
<td>1.7 x</td>
<td>77.3</td>
</tr>
<tr>
<td>SEQNet@pixel4-A4</td>
<td>79.9</td>
<td>35.5 ms</td>
<td>2.2 x</td>
<td>80.0</td>
</tr>
</tbody>
</table>

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 fine-tuning, 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.

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 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, Hardswish, 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


