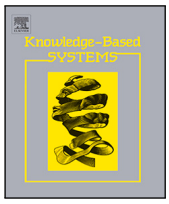
Knowledge-Based Systems 305 (2024) 112620

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



Accelerating one-shot neural architecture search via constructing a sparse search space

Hongtao Huang [a,*], Xiaojun Chang [b], Lina Yao [a,c]

[a] School of Computer Science and Engineering, University of New South Wales, Kensington, Sydney, 2052, NSW, Australia
[b] Faculty of Engineering & Information Technology, University of Technology Sydney, Ultimo, Sydney, 2007, NSW, Australia
[c] CSIRO's Data61, Eveleigh, Sydney, 2015, NSW, Australia

ARTICLE INFO

Keywords:
Automated machine learning (AutoML)
Neural architecture search (NAS)
Multi-platform network deployment

ABSTRACT

Neural Architecture Search (NAS) has garnered significant attention for its ability to automatically design high-quality deep neural networks (DNNs) tailored to various hardware platforms. The major challenge for NAS is the time-consuming network estimation process required to select optimal networks from a large pool of candidates. Rather than training each candidate from scratch, recent one-shot NAS methods accelerate the estimation process by only training a supernet and sampling sub-networks from it, inheriting partial network architectures and weights. Despite significant acceleration, the supernet training with a large search space (i.e., the number of candidate sub-networks) still requires thousands of GPU hours to support high-quality sub-network sampling. In this work, we propose SparseNAS, an approach for one-shot NAS acceleration by reducing the redundancy of the search space. We observe that many sub-networks in the space are underperforming, with significant performance disparity to high-performance sub-networks. Crucially, this disparity can be observed early in the beginning of the supernet training. Therefore, we train an *early predictor* to learn this disparity and filter out high-quality networks in advance. Then, the supernet training will be conducted in this space sub-space. Compared to the state-of-the-art one-shot NAS, our SparseNAS reports a 3.1× training speedup with comparable network performance on the ImageNet dataset. Compared to the state-of-the-art acceleration method, SparseNAS reports a maximum of 1.5% higher Top-1 accuracy and 28% training cost reduction with a 7× bigger search space. Extensive experiment results demonstrated that SparseNAS achieves better trade-offs between efficiency and performance than state-of-the-art one-shot NAS.

1. Introduction

In recent years, the fast development of deep learning and deep neural networks (DNNs) has been widely applied to extensive tasks including but not limited to image processing [1], robotics [2], recommendation systems [3]. DNNs also empower many hardware devices to become "smart", such as real-time image analytics [4,5], natural language recognition [6,7], health monitoring [8], etc.

Different hardware devices, ranging from powerful servers to lightweight edge devices, have different hardware resource constraints, such as processing power, memory limitation and latency requirements. The Cloud-Edge [9,10] DNN deployment scheme has become popular for mobile intelligence domains, which utilize the powerful computing ability of cloud servers to train large DNN models and then fine-tune these models for edge devices. However, manually specializing well-trained DNNs for various edge devices is labour-consuming and resource-intensive. Recent popular smart devices, such as smartphones, tablets, and the Internet of Things (IoTs), require tailored lightweight DNNs to meet their hardware limitations [11]. Models for different platforms need to be adapted accordingly through architectural adjustments and model retraining. Therefore, there is a growing need for efficient model deployment in mobile intelligence domains.

Neural architecture search (NAS) has emerged as a powerful automated tool for accelerating the process of network lightweight [12–15]. By leveraging advanced search algorithms, NAS can explore different network architectures for resource-limited devices with less human labour. However, the repeated train–evaluate process for each candidate network necessitates huge computing resources. Recent one-shot NAS methods reduce the computational overhead by treating all architecture as different sub-networks of a supernet and sharing network parameters among all sub-networks [16–20]. Thereby, one-shot NAS only train the supernet, and each sub-network inherits parameter weights from the supernet without retraining, leading to significant efficiency gains.

* Corresponding author.
E-mail addresses: hongtao.huang@unsw.edu.au (H. Huang), XiaoJun.Chang@uts.edu.au (X. Chang), lina.yao@data61.csiro.au (L. Yao).

https://doi.org/10.1016/j.knosys.2024.112620
Received 18 June 2024; Received in revised form 21 September 2024; Accepted 9 October 2024
Available online 23 October 2024

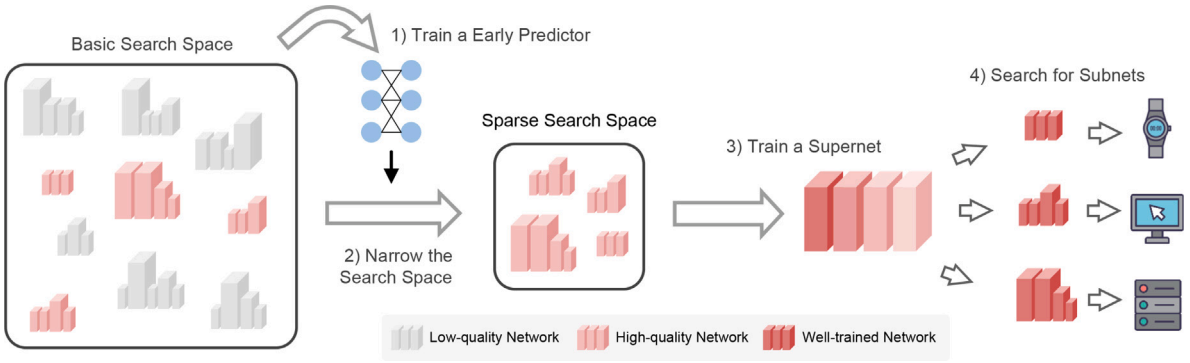


Fig. 1. A diagram of the procedure of SparseNAS. Given a basic search space, we conduct four steps to train our SparseNAS as follows: **(1)** we first train an *early predictor* based on the architecture knowledge in the basic space; **(2)** We then leverage the predictor to filter out high-quality architectures (in red) and restructure a sparse search space; **(3)** We optimize a supernet based on the sub-space; **(4)** After training, we search sub-networks from the supernet for hardware deployments.

Despite demonstrating promising results, one-shot NAS confronts the challenge of training a high-performing supernet to support the subsequent sampling process. The supernet can be regarded as a set of various sub-networks or a huge network architecture space. Optimizing a supernet is equivalent to simultaneously optimizing all sub-networks in the search space, which can be cast as a challenging multi-objective problem. Recent one-shot NAS tends to define a large search space to ensure diversity and explore potential high-quality sub-networks. However, the more sub-networks there are, the more complex the supernet will be to optimize and converge, which often requires substantial computational resources (e.g., over 1200 GPU hours) for the supernet training process.

To further reduce the cost of the NAS process and accelerate DNN deployment for mobile intelligence, we propose an efficient one-shot NAS, SparseNAS, which builds a sparse search space containing only high-quality networks for network searching. Specifically, we train an *early predictor* to extract high-quality sub-networks at the beginning and then rebuild a sparse search space for supernet training. After training, SparseNAS can directly sample sub-networks from the sparse space without employing an evolutionary search [21–23], which also reduces the search cost compared to the previous method [13,15]. Fig. 1 illustrates the workflow of our SparseNAS. Our overall contributions are as follows:

1. In this paper, we fully analyze the limitations of training a supernet in a larger search space from three aspects: the lack of high-quality sub-networks, the multi-model forgetting problem, and the disparity of sub-network performance.
2. To alleviate those limitations, we propose SparseNAS, a NAS approach aimed at constructing a sparse space with potential high-quality models at the beginning of the NAS process. Specifically, we develop an *early predictor* to extract a set of high-quality networks as a new space based on the performance disparity.
3. Compared to the state-of-the-art (SOTA) one-shot NAS, SparseNAS achieve a **3.1**× training speedup with comparable Top-1 accuracy on the ImageNet dataset. Compared to the SOTA acceleration method, SparseNAS reports a training cost reduction up to **28**% within a **7**× bigger network search space. Sub-networks in our sparse space achieve an average of **1.1**% and a maximum of **1.5**% higher Top-1 accuracy than their counterparts.
4. Since most of the sub-networks in SparseNAS are high-quality, the sub-network search process is about **40**× faster than the process in the SOTA without additional evolutionary search.
5. We further deploy searched sub-networks to several smart edge devices, including Samsung S22 Ultra, Samsung Tab S8+, Huawei Mate50 Pro+ and Huawei Watch GT3 for on-device latency test, where SparseNAS also reports a better latency and accuracy.

2. Related work

As the demand for designing complex DNN models rapidly grows, NAS methods are becoming increasingly prevalent in architecture design and network deployment, gradually replacing partial hand-crafted processes. Early NAS [21,24–28] automated the network designing process leveraging reinforcement learning or evolutionary search. However, early NAS methods [21,27,28] suffer from substantial computational overhead for network performance estimation, often requiring thousands of GPU days to train and evaluate a large number of candidate models from scratch. This extensive computational burden hinders the practical applicability of early NAS approaches and makes them less feasible for real-world scenarios with limited computational resources, such as mobile intelligence.

Recent one-shot NAS methods alleviate the huge computing overhead by optimizing a supernet to represent a wide range of candidate sub-networks. They significantly reduce the computational cost by sharing parameter weights across all the sub-networks without extra training. Different one-shot NAS methods differ in how to construct the supernet. Cell-based methods regard the supernet as a directed acyclic graph (DAG) and use a continuously differentiable relaxation to parameterize the search space [16–18,20,29–32]. For example, DARTS [20] relax the operation choices to be differentiable and optimize all weights with a continuous relaxation of in the search space. SWD-NAS [31] proposed a dual-attention mechanism to alleviate the performance collapse in DARTS. GENAS [31] introduced an evolutionary framework to relieve the coupling problem in DARTS. However, cell-based methods suffer from coupled weight in supernet [33] and complicated optimization with sensitive hyper-parameters [33,34].

To address the coupled problem, architecture-based one-shot NAS [13,15,33,35,36], decouple the one-shot NAS process in two separated stages: (1) the supernet optimization stage and (2) the architecture (sub-network) searching stage. Different from cell-based methods, the supernet in architecture-based methods takes a well-designed network (e.g., ResNet or MobileNet-v3) as a prototype with extensive elastic network hyperparameters, such as the number of blocks, the number of layers, the number of channels in the convolution kernel and so on. Random network sub-structures will be involved in each training iteration during the supernet training stage. This training strategy successfully supports the sub-network sampling in the second stage.

Different architecture-based one-shot NAS methods explore varying strategies to sample sub-networks and optimize proper weights for the supernet. SPOS [33] firstly introduces a random sampling method. OFA [13] proposes a phased training strategy, *progressive shrinking*, which starts training from sampling large networks to smaller ones by gradually providing smaller architectural configuration choices. The following work, CompOFA [15], accelerates OFA's strategy by a dimension-coupling constraint but results in performance degradation.

Despite the empirical success of two-stage one-shot NAS, the multi-model forgetting problem [37,38] remains a significant challenge. We further discuss this phenomenon in 3.3.

As for the search stage, recent methods [13,15,34–36,39] conduct an evolutionary search [22,23,25] to explore specialized sub-networks under given resource constraints. However, it is time-consuming to evaluate each candidate for comparison within a vast search space. An alternative is to train a performance predictor [13,15] to estimate network performance. During the evolutionary search, the predictor provides approximated network performance for network comparison and ranking.

3. Motivation

3.1. Problem formalization

As mentioned in Section 2, the architecture-based one-shot NAS has two stages. The first stage defines a search space $\mathcal{A}$ containing different architectures (i.e., sub-networks α_i). Due parameters sharing, sub-networks' weights W_{α_i} are a sub-set of the supernet weight $W_{\mathcal{A}}$ (i.e., $W_{\alpha_i} \subseteq W_{\mathcal{A}}$). The goal of this stage is to minimize the loss of every sub-network by updating the supernet weight, which can be cast as a multi-objective problem as follows:

$$\min_{W_{\mathcal{A}}} \sum_{i=1}^{N} \mathcal{L}(W_{\alpha_i};\ X_{trn}) \tag{1}$$

where X_{trn} represents the training dataset, and $\mathcal{L}(\cdot)$ represents the loss function. N is the total number of candidate architecture in $\mathcal{A}$. Since weight-sharing, the bigger the N, the harder $W_{\mathcal{A}}$ is to optimize. In practice, since N is usually extremely large (e.g., 10^{19}), Eq. (1) is often approximated by optimizing a set of sampled candidates (n_{sub} sub-networks) for each mini-batch input as follow:

$$\min_{W_{\mathcal{A}}} \sum_{i=1}^{n_{sub}} \mathcal{L}(W_{\alpha_i};\ X_{batch}) \tag{2}$$

where X_{batch} represents a mini-batch of input data. As the total training iterations can span thousands or even millions, a large scale of sub-networks will be trained and aggregated their gradients for updating the supernet. A larger n_{sub} allows the supernet to obtain gradients from different sub-networks for the same number of training iterations but increases the total duration of training.

After supernet training, the second stage aims to extract the optimal sub-network α^* under given constraints (e.g., model size limitation or latency requirements):

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} \mathrm{ACC}_{val}(W_{\alpha}, R;\ X_{val}) \tag{3}$$

where ACC_{val} refers to the network accuracy on validation dataset. R denotes the resource constraints, and X_{val} denotes the validation dataset.

Typical one-shot NAS approaches create a vast search space with diverse candidate networks for exploring potential high-performing architectures. The diversity within the search space allows for a more comprehensive exploration of possible architectures, enabling the identification of innovative and high-performing network designs that might not be present in a more constrained search space. However, approaches require thousands of GPU hours to train a high-performance supernet due to the large search space. For example, OFA [13] spend over 1200 GPU hours to optimize a vast search space with 10^{19} candidates. In the following part of this section, we revisit the one-shot NAS optimization challenge within a large search space and specifically answer the question: *Do we really need a huge search space?*

Table 1
A statistical analysis of the search space.

N_{models}	ACC_{mean}	ACC_{std}	$N_{\text{high-qlt}}$	*Ratio*
500	74.2%	0.49%	83	16.6%
1000	74.0%	0.46%	168	16.8%
5000	74.1%	0.47%	815	16.3%
10000	74.0%	0.47%	1624	16.2%

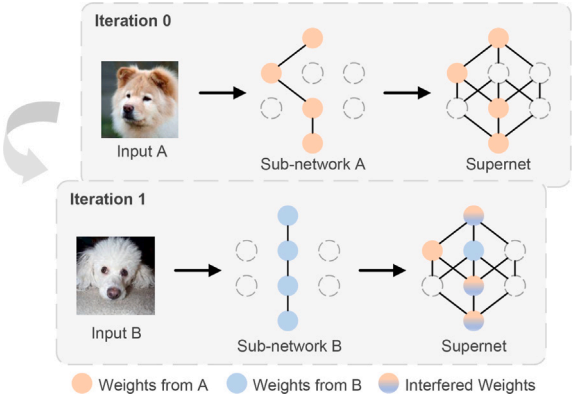


Fig. 2. An illustration of the multi-forgetting problem in one-shot NAS. Weights from each sub-network might interfere with each other and damage the optimization of the supernet.

3.2. The lack of high-quality sub-networks

In addition to the size of the search space, we are concerned about the interference during supernet training caused by inferior networks. The target of NAS is to search for high-quality networks. Thus, we try to figure out how many candidates in a vast space can be regarded as 'high-quality'. Here, we roughly identify networks with validation accuracy higher than the sum of mean accuracy and corresponding standard deviation as high-quality networks.

Table 1 reports a statistical analysis of the search space OFA-MobileNetV3 [13]. We measure the validation accuracy of networks on several random-sampled sub-spaces to roughly analyze the vast search space. Here, N_{models} denotes the number of randomly sampled networks. ACC_{mean} and ACC_{std} represent the average validation accuracy and the standard deviation accuracy in each sub-space. $N_{\text{high-qlt}}$ and *Ratio* represent the number of high-quality networks and their percentage of the total. Here, a network will be considered a promising model if it has better accuracy than $ACC_{\text{mean}} + ACC_{\text{std}}$. About 16% network can be considered high-quality while other 84% models are inferior. If we set the threshold to the sum of the mean accuracy and twice the standard deviation, only 1% models are high-quality networks.

The presence of low-quality networks can lead to a serious forgetting problem, which has several detrimental effects on the optimization process. Much of the training budget has been wasted on these useless networks during training. In addition, they also increase the searching burden in the *architecture searching stage* for extracting high-performing sub-networks for deployment.

3.3. The multi-model forgetting problem

The multi-model forgetting problem refers to the performance decline among previous-trained sub-networks in the search space, first observed in early one-shot NAS [37,38]. Specifically, the training of subsequent sub-networks in one-shot NAS can lead to the overwriting of shared weights in the previous sub-networks, resulting in a performance decrease in the previous sub-networks and negatively impacting the overall optimization process. As illustrated in Fig. 2, the current

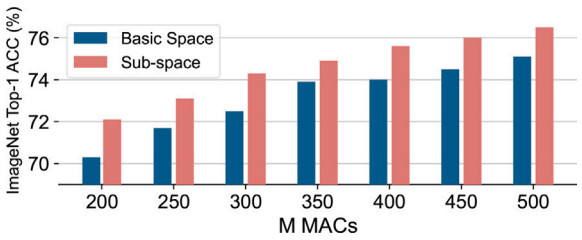


Fig. 3. Network performance comparison in a basic space and in a sampled sub-space. The basic search space is OFA-MobileNetV3 [13], and the sub-space comprises 2000 sub-networks randomly sampled from the basic search space. There are seven different sizes of networks, ranging from 200M MACs to 500M MACs. MACs stands for the multiply–accumulate operation, a common step that computes the product of two numbers and adds that product to an accumulator. We conduct *uniform sampling* [33] for supernet training.

sampled sub-network interferes with the previous networks, and the supernet may 'forget' valuable knowledge learned from the previous one. This phenomenon significantly hinders the optimization of the supernet, resulting in a longer training period and heavy training overhead.

This problem also exists in one-shot NAS with two stages. During the supernet training stage, a random set of sub-networks are involved in forward-and-backwards propagation for a given data batch as Eq. (2). Since weight-sharing, the supernet is updated by aggregated gradients from these sub-networks. Although this optimization process supports the sub-network sampling with inherited weights in the following stage, the gradients from high-quality sub-networks are affected by those from low-quality sub-networks as the number of training iterations increases. The forgetting problem complicates the optimization of the supernet, resulting in longer training periods.

To further explore the relationship between the multi-model forgetting problem and the search space size, we randomly sample a sub-space with 2000 networks from OFA-MobileNetV3 search space [13] and train two supernets within the sub-space and the original space for the same epoch. In Fig. 3, we illustrated the validation accuracy of several sub-networks in each of these two spaces. We observe that sub-networks in the sub-space show higher performance than those in the basic space with a maximum accuracy improvement of 1.81% at 300 MACs, which indicates that networks in a smaller space are less interfered with by other networks. Directly reducing the number of objectives in a multi-objective task can simplify optimization, thereby benefiting supernet training. Candidates in a smaller search space are more likely to be randomly selected and have their corresponding network weights updated during supernet training, leading to more thorough training. Since the high-quality candidates account for only a small proportion (see Section 3.2), we prefer to optimize a sparse, selective space rather than a random sub-space.

3.4. The disparity of sub-network performance

Compared to training a standard DNN, a major difference in supernet training is that multiple sub-networks are sampled and trained simultaneously at each iteration. Prior research [40] reported that standard DNNs with high-quality network structures perform well even at the early training epoch. We hypothesize that this phenomenon also occurs among varying sub-networks during supernet training.

To verify this hypothesis, we train a supernet for 300 epochs and evaluate 1000 random sub-networks on ImageNet in Fig. 4. As shown in Fig. 4 (Top), high-quality sub-networks (marked in red) keep their leading performance both at the beginning and at the convergence. Fig. 4 (Bottom) further shows validation records of two sub-networks; one is high-performing while the other has an inferior performance. The high-performing one shows superior accuracy during training, indicating that the performance gap is persistent. These observations

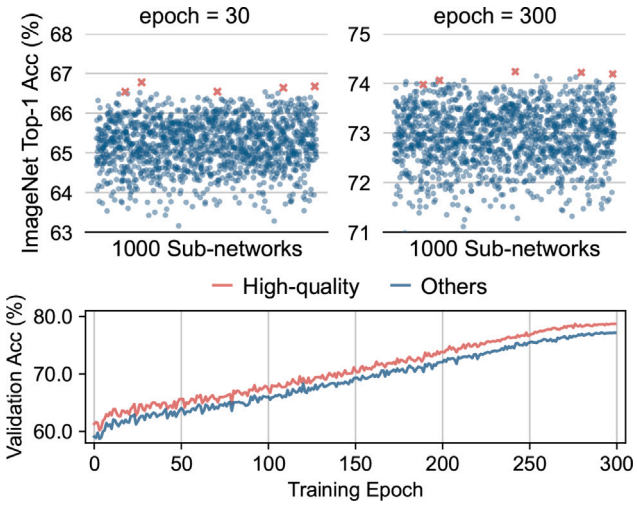


Fig. 4. Performance evaluation for sub-networks randomly sampled from the same supernet. The supernet is trained with *uniform sampling* [33] for 300 epochs. **Top:** A validation accuracy comparison of sub-networks in 30 and 300 epochs on ImageNet [41]. High-quality networks are marked in red. **Bottom:** A validation accuracy record of a high-quality sub-networks and a low-quality one throughout the supernet training process.

validate our hypothesis. Due to weight sharing, all candidate networks are optimized simultaneously, leading to a constant performance disparity. Sub-networks that demonstrate superior performance and lead the optimization process during the early epochs are highly likely to be high-quality architectures.

4. Methodology

Motivated by the aforementioned observations, we intend to rebuild a smaller search space, which maintains the diversity and quality of sub-networks, alleviating the phenomenon of multi-model forgetting. After that, the supernet will be optimized in this sparse space.

Here, we give a definition of the sparser space. Considering the basic search space $\mathcal{A}$ with parameter weight W, we define a sub-space $\mathcal{A}^* \subset \mathcal{A}$ where networks in this space have a higher network accuracy than those not in under the same recourse constraint in the basic space as follow:

$$\begin{aligned} &\forall\ \alpha^* \in \mathcal{A}^*,\ \alpha \in \mathcal{A},\ \alpha \notin \mathcal{A}^* \\ &\text{if } \mathrm{R}(\alpha^*) = \mathrm{R}(\alpha), \\ &\text{then } \mathrm{ACC}_{val}(W_{\alpha^*}) \geq \mathrm{ACC}_{val}(W_{\alpha}). \end{aligned} \tag{4}$$

where α^* and α represent sub-networks from $\mathcal{A}^*$ and $\mathcal{A}$, respectively. ACC_{val} is the validation accuracy. We further rewrite the supernet optimization (i.e., Eq. (1)) with a sparse space (i.e., Eq. (4)) as below:

$$\begin{aligned} &\min_{W^*_{\mathcal{A}}} \sum_{i=1}^{N^*} \mathcal{L}(W^*_{\alpha_i};\ X_{trn}), \\ &\text{s.t.}\ \ \forall\ \alpha^* \in \mathcal{A}^*,\ \alpha \in \mathcal{A},\ \alpha \notin \mathcal{A}^* \\ &\qquad \text{if } \mathrm{R}(\alpha^*) = \mathrm{R}(\alpha), \\ &\qquad \text{then } \mathrm{ACC}_{val}(W_{\alpha^*}) \geq \mathrm{ACC}_{val}(W_{\alpha}). \end{aligned} \tag{5}$$

However, it is difficult to guarantee the actual accuracy constraint $\mathrm{ACC}_{val}(W_{\alpha_i^*}) \geq \mathrm{ACC}_{val}(W_{\alpha_i})$ for two issues. Firstly, evaluate all candidates within a vase space $\mathcal{A}$ (e.g., 10^{19} candidates) is extremely time-consuming. Secondly, since the supernet is un-trained, we cannot evaluate sub-networks to get their actual accuracy.

To address the first issue, we approximate the actual accuracy constraint to the estimated accuracy constraint. In Section 2, we mentioned that the performance predictor is widely used for architecture search. Previous works train a predictor for sub-network searching during

Algorithm 1: Training supernet in a sparse space

Input: Search space $\mathcal{A}$; Early predictor $\mathcal{P}_e(\cdot)$; Maximum space size N^*; Number of sampled sub-networks n_{sub}

Create a empty space $\mathcal{A}^*$
for *i in* 1, 2, ..., N^* **do**
  Random sample a set of networks $\{\alpha^*_{i,0}, \cdots, \alpha^*_{i,j}\}$
  Estimate networks $\{\mathcal{P}_e(\alpha^*_i), \cdots, \mathcal{P}_e(\alpha^*_{i,0})\}\}$
  Evolutionary search the high-quality network α^*_i
  Append α^*_i into $\mathcal{A}^*$
end
Initialize the supernet weights W
while *not convergence* **do**
  Draw a mini-batch of data
  Randomly sample a small set n_{sub} of sub-networks $\{\alpha^*_1, \alpha^*_2, \cdots, \alpha^*_n\}$ from $\mathcal{A}^*$ for propagation
  Aggregate gradients of $\{\alpha^*_1, \alpha^*_2, \cdots, \alpha^*_n\}$
  Update the parameter weights W.
end

supernet training [34] or after supernet training [13,15], while we intend to leverage accuracy predictor to extract high-quality network before supernet training. We intend to replace the actual accuracy comparison with the predicted accuracy comparison to avoid network evaluation. However, the second issue still remains since the training of a traditional predictor also requires actual network accuracy as ground truth.

Our solution is to train the *early predictor* as an alternative choice. In Section 3.4, we observe that sub-networks with higher accuracy at the beginning are more likely to show superior accuracy than others at the convergence. Based on this observation, we train the *early predictor* with network accuracy at the early stage to approximate the traditional predictor. The performance gap between the two kinds of predictor is small. More practical details of *early predictor* are illustrated in Section 5.2.

By this means, we can leverage the *early predictor* to provide estimated accuracy for given candidate networks. Let $\mathcal{P}_e(\cdot)$ denote the *early predictor*, and we can update Eq. (5) into:

$$\begin{aligned} &\min_{W^*_{\mathcal{A}}} \sum_{i=1}^{N^*} \mathcal{L}(W^*_{\alpha_i};\ X_{trn}), \\ &\text{s.t.}\ \ \forall\ \alpha^* \in \mathcal{A}^*,\ \alpha \in \mathcal{A},\ \alpha \notin \mathcal{A}^* \\ &\quad \text{if } \mathrm{R}(\alpha^*) = \mathrm{R}(\alpha), \\ &\quad \text{then } \mathcal{P}_e(\alpha^*) \geq \mathcal{P}_e(\alpha). \end{aligned} \tag{6}$$

We also conduct an evolutionary search [21] to better sample high-quality networks. Instead of evaluating candidate network performance after training in traditional evolutionary-based NAS [21], our *early predictor* efficiently provides network estimated performance for the search.

Algorithm 1 illustrates a workflow of training a supernet within a sparse search space. A similar idea of building a smaller space has already been proposed by CompOFA [15]. The difference is that we carefully consider the balance between the complexity and the diversity of the new space $\mathcal{A}^*$ by setting $\mathcal{N}^*$ and applying $\mathcal{P}_e(\cdot)$, rather than a fixed search space produced by a heuristic strategy in CompOFA. We provide a detailed comparison of two methods in Section 5.

5. Experiments and results

5.1. Experiment settings

5.1.1. Search space settings

Following OFA [13], we use the OFA-MobileNetV3 as the basic search space. Table 2 report the network configurations of this search space. There are five dynamic stages named DMBConv, which refer to inverted dynamic inverted residual block [13]. *Depth* represents the number of dynamic convolution blocks (or layers) in the dynamic stage. *Width* and *Expand* denote the output channel width of each block and the width's corresponding expand ratio. The maximum channel width is calculated by *Width* × *Expand*. *Kernel* is the kernel size of each block. The dynamic network configurations, which include variations in network depth $D = \{2, 3, 4\}$, channel width expansion $W = \{3, 4, 6\}$, and kernel size $K = \{3, 5, 7\}$, result in $((3\times3)^2+(3\times3)^3+(3\times3)^4)^5 \approx 2\times10^{19}$ unique candidate architectures. Besides, we also generate SparseNAS to another search space in Section 5.10 for an ablation study.

Table 2
OFA-MobileNetV3 with dynamic network configurations.

Stage	*Depth*	*Width*	*Expand*	*Kernel*
Conv	1	16	–	3
MBConv	1	16	1	3
DMBConv1	{2, 3, 4}	24	{3, 4, 6}	{3, 5, 7}
DMBConv2	{2, 3, 4}	40	{3, 4, 6}	{3, 5, 7}
DMBConv3	{2, 3, 4}	80	{3, 4, 6}	{3, 5, 7}
DMBConv4	{2, 3, 4}	112	{3, 4, 6}	{3, 5, 7}
DMBConv5	{2, 3, 4}	160	{3, 4, 6}	{3, 5, 7}
Conv	1	960	–	1
Conv	1	1280	–	1

5.1.2. Datasets settings

All the experiment results are measured on ImageNet12 [41], a large-scale image classification dataset with over one million images in 1000 classes. All images are preprocessed to 256 × 256 resolutions. There are about 120,000,000 images for training and about 150,000 images for testing.

5.1.3. Baseline settings

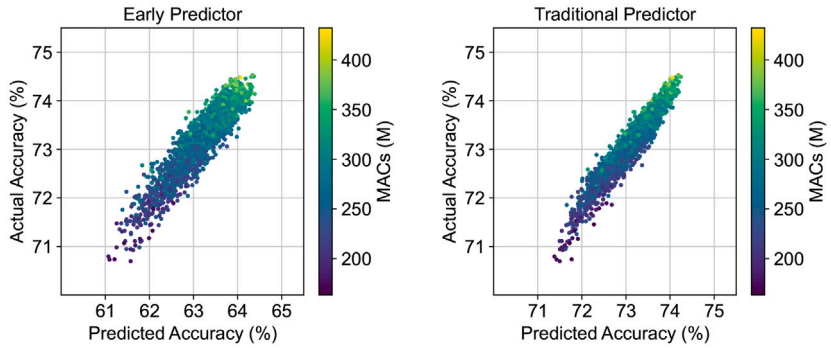
We choose CompOFA [15] as the main baseline. CompOFA is the state-of-the-art efficient one NAS which conducts a heuristic network sampling strategy by coupling the block depth and layer width choices $\{D, W\}$ to $(2, 3)$, $(3, 4)$ and $(4, 6)$ and fixes the kernel size. This heuristic strategy led to a narrower search space with 243 candidate networks in the network family. CompOFA reports empirical experiment results in the OFA-MobileNetV3 search space. However, this strategy lacks flexibility and diversity since the coupling rules determine its candidate networks. Those candidates have not been verified by performance evaluation or estimation yet, which damages the performance of individual subnets and requires much computation to search for an optimal model.

5.1.4. Implementation details of SparseNAS

Two main stages require DNN training in SpareNAS: (1) training a proxy supernet for the *early predictor* and (2) training the supernet on the sparse search space. The first stage takes 30 epochs with a learning rate of 0.1, while the second stage takes 150 epochs with an initial learning rate of 0.008 for the first 5 epochs and 0.08 for the remaining 145 epochs. We trained the model on five NVIDIA RTX 3090 GPUs with a batch size 128. Other hyperparameters are following CompOFA.

5.2. Implementation details of the early predictor

The *early predictor* is a vital component for SparseNAS. It predicts the corresponding performance based on the input network architecture. As mentioned in Section 4, we leverage the *early predictor* $\mathcal{P}_e(\cdot)$ to build a sparse search space $\mathcal{A}^*$ based on the disparity between high-quality and low-quality sub-networks. In this section, we first provide detailed information about the training procedures. After that, we verify its feasibility by comparing it with a traditional predictor.



Predictors	RMSE (%)	Kendell's τ
30-ep	0.266	0.712
100-ep	0.252	0.733
150-ep	0.247	0.756
200-ep	0.235	0.771
300-ep	0.219	0.775

(c)

Fig. 5. Predictor performance testing. **(a)** and **(b)**: Predicted results for 1000 random networks from the *early predictor* and a traditional predictor. The predictor generates the predicted accuracy, and the corresponding actual accuracy is evaluated on the ImageNet test dataset. **(c)**: RMSE and Kendell's τ of different predictors trained by training materials (architecture-accuracy pairs) from different supernets. "X-ep" denotes this predictor is trained with the training materials provided by a supernet trained for X epochs.

Table 3
RMSE comparisons of different predictors.

Predictor	Time Cost (s)	RMSE (%)
FFN	12.9	0.53
LSTM	17.5	0.41
RF	1.05	0.29
GBDT	0.71	0.27
XGBoost	0.25	0.30

5.2.1. The training procedures for the early predictor

The function of a predictor is to give a prediction of the corresponding performance based on the input network architecture. In other words, the input of the predictor is the network architecture, and the output is the predicted performance (e.g., the accuracy in the classification task).

The training steps of the *early predictor* are: (1) Train a proxy supernet on the original search space for 30 epochs; (2) Randomly sample and evaluate 1000 sub-networks and obtain architecture-accuracy pairs; (3) Train the predictor based on the evaluation pairs.

The proxy supernet is trained with *uniform sampling* [33] on ImageNet, which randomly samples two sub-networks for optimization in one data batch with a learning rate of 0.1 and the batch size is 128. The training cost of the proxy supernet is about 30 GPU hours on a single RTX 3090 GPU.

As for the predictor, we experimented with different types of models to build the predictor, including a three-layer feedforward neural network (FNN) in OFA [13] and CompOFA [15], the LSTM filter in GreedyNASv2 [42], the random forest regressor (RF) in AttentiveNAS [34], and two machine-learning-based regressors GBDT [43] and XGBoost [44]. Table 3 reports the root-mean-square error (RMSE) between the predicting accuracy and the actual accuracy. The RF, GBDT and XGBoost regressor contain 100 estimators trained with a learning rate of 0.01. Other models use the original settings. The time cost is measured on an Intel Xeon Platinum 8255C CPU. In this case, ML-based approaches, RF, GBDT and XGBoost, report a lower RMSE and less time cost than other approaches, and GBDT reports the lowest RMSE. Therefore, we choose GBDT as our *early predictor*.

5.2.2. The feasibility analysis for the early predictor

To verify the feasibility of our *early predictor*, we measure its predictive accuracy by comparing it with a traditional predictor generated from a 300-epoch supernet for comparison. All other hyper-parameters are the same except for the proxy supernet. The experiment results are reported in Fig. 5(a). The horizontal coordinate indicates predicted accuracy, and the vertical coordinate indicates actual accuracy evaluated on the test dataset. The traditional predictor is thoroughly the better one in predicting accuracy for the test set, while the *early predictor* also performs well with competitive performance.

Moreover, we further calculate the root-mean-square error (RMSE) and Kendell's τ ranking correlation between the predicted and actual accuracy for different predictors, as shown in Fig. 5(b). For RMSE, we shift the data domain of predicted accuracy to the domain of actual accuracy by calculating the difference in the mean of predicted accuracy and actual accuracy. RMSE and Kendell's τ for the 30-ep predictor (i.e., the *early predictor*) are 0.266% and 0.712, while 0.219% and 0.775 for the 300-ep predictor (i.e., the traditional predictor). The experiment results indicate that the *early predictor* is as reliable as a traditional predictor with similar predictive accuracy.

5.3. Implementation details of the sparse space

To enhance the generalization for varying deployment scenarios, we conduct a uniform MACs interval sampling, which divides the basic search space into groups of equal MACs and then samples the same number of high-quality networks from each group. In practice, we sample five networks from each 1M MACs from 150M to 500M MACs by evolutionary search [21]. The population size is 50, and the number of iterations is 100 for the evolutionary search. We use multi-processing with Intel Xeon Platinum 8255C CPU to accelerate the search, and the cost is less than four hours. There are a total of 1750 candidate networks in the new space. We note that candidate networks can be flexibly sampled according to demand.

5.4. Complexity analysis of NAS

In general, NAS processes are very time-consuming because their evaluation and ranking require training all networks in the search space from scratch. The training cost is the most critical factor in determining the time cost of the NAS process. Considering there are N candidate networks, we analyze the time complexity with the notation $\mathcal{O}(\cdot)$ of SparseNAS against other NAS methods in Table 5. Similar to recent one-shot NAS methods, such as SSRNAS [15] and CompOFA [15], SparseNAS trains a supernet and shares network parameter weights with all sub-networks in the search space, rather than training each candidate networks from scratch as MnasNet [12]. Although our method trains a proxy supernet for the early predictor, the complexity of SparseNAS is $\mathcal{O}(1)$, which is irrelated with N. For more details on the supernet training, please refer to Section 5.5.

Table 4
Training stage and duration comparison for three methods.

Model	Stage	n_{sub}	Epochs	GPU Hours
Teacher	–	–	180	153.0
OFA	Kernel Training	1	125	97.9
	Depth Warm-up	2	25	31.7
	Depth Training	2	125	135.4
	Width Warm-Up	4	25	42.9
	Width Training	4	125	200.0
	Total Hours: **507.9**			
CompOFA	Compound Warm-Up	4	25	37.1
	Compound Training	4	125	185.4
	Total Hours: **222.5**			
SparseNAS	Early Predictor	2	30	25.5
	Sparse Training	3	150	175.3
	Total Hours: **200.8**			
SparseNAS†	Early Predictor	2	30	25.5
	Sparse Training	2	150	135.0
	Total Hours: **160.5**			

Table 5
Complexity analysis of different NAS methods.

Method	MnasNet	SSRNAS	CompOFA	SparseNAS
Complexity	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

5.5. Training stages and costs

In this section, we compare the streamlining and speedup of the supernet training processes. Following CompOFA and OFA, SparseNAS also adopt knowledge distillation [45] to assist supernet optimization by training the biggest sub-network for 180 epochs as a teacher network. The teacher network provides network parameters for the supernet initialization and soft labels in each training iteration. In Section 2, we have mentioned that in the supernet training stage, one-shot NAS optimize n_{sub} sub-networks to update network parameters in each data batch (see Eq. (2)). SparseNAS contains a bigger ratio of high-quality networks, allowing us to speed up the training using a smaller $n_{sub} = 3$. Additionally, SparseNAS† reports a further acceleration by setting $n_{sub} = 2$, but sub-network performance decreases slightly.

Table 4 reports the comparison of training schedules and time costs. The training cost is measured by an NVIDIA RTX 3090. There is a complex five-step training order in OFA known as *progressive shrinking*, which is complex and time-consuming. CompOFA and SparseNAS reduce the supernet training to two steps and one step, respectively. For a fair comparison, we also take the training cost of the *early predictor* into account. Within a high-quality search space, SparseNAS no longer need the 25-epoch warmup stage. Compared to OFA, our method reaches a maximum 3.1× speedup. Compared to CompOFA, our SparseNAS reports a 28% training cost reduction. In Section 5.7, we show that, even with less time cost, the sub-network in our search space does not encounter an obvious performance degradation as CompOFA.

5.6. Searching stages and costs

As for the search stage, SparseNAS reports a faster search procedure for two reasons. First, our search space is sparse and easily traversable. Second, candidate networks in our space have already been verified as high-quality networks before supernet training. Thus, given the constraints of different MACs, SparseNAS can quickly select target sub-networks without an evolutionary search. Table 6 reports a search cost comparison, where *#evo*, *#evo** and *#unevo* denote evolutionary search, evolutionary search with fewer iterations, and not using evolutionary search, respectively. 200~ denotes the MACs constraints from 200M to 300M MACs, 300~ and 400~ are similar. SparseNAS reduce the search time to 13 ms by **40**× faster than CompOFA in different MACs intervals. Meanwhile, SparseNAS does not need to further train a performance predictor for estimation [13] since we create the *early predictor* at the beginning.

Table 6
An average time cost of the sub-network searching process.

Model	Search Method	Avg Cost (100M MACs interval)		
		200~(M)	300~(M)	400~(M)
OFA	*#evo*	129.4 (s)	125.9 (s)	137.1 (s)
CompOFA	*#evo**	582.5 (ms)	573.1 (ms)	608.0 (ms)
SparseNAS	*#unevo*	12.9 (ms)	13.1 (ms)	13.3 (ms)

Table 7
A Top-1 accuracy comparison of models on ImageNet.

Model	MACs	Top-1 ACC(%)
MobileNetV2 [46]	300M	72.0
MnasNet [12]	317M	75.3
ProxylessNAS [14]	320M	74.6
SPOS [33]	328M	74.7
FairNAS [47]	321M	74.7
FBNet [48]	375M	74.9
MobileNetV3 [4]	356M	75.2
SWD-NAS [31]	–	75.5
GreedyNAS [36]	284M	76.2
CompOFA [15]	300M	76.3
EfficientNet-B0 [49]	390M	76.3
GENAS [32]	–	76.1
OFA [13]	300M	76.7
SparseNAS†	295M	76.5
SparseNAS	295M	76.7

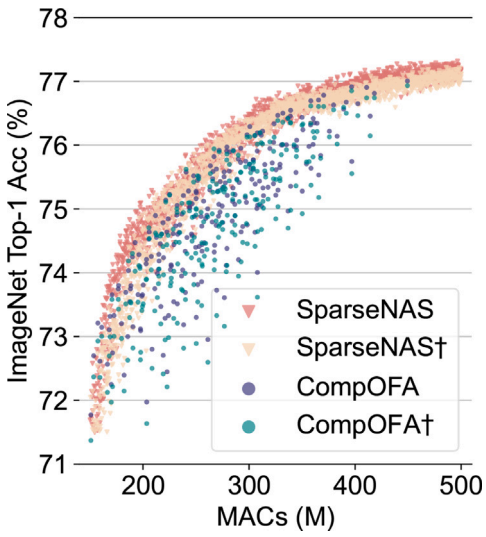


Fig. 6. A validation accuracy comparison of the network family from SparseNAS and CompOFA.

5.7. Sub-network performance

We evaluate the accuracy of sub-networks by measuring their ImageNet Top-1 accuracy. In Table 7, we illustrated the highest sub-network performance of different NAS methods with similar constraints of MACs. Here, † represents $n_{sub} = 2$, reducing the supernet cost with a slight decrease in subnetwork performance. Our SparseNAS and SparseNAS† report 76.7% and 76.5% Top-1 accuracy, better than other architecture-based one-shot NAS methods except OFA. According to Sections 5.5 and 5.6, SparseNAS show a better balance between

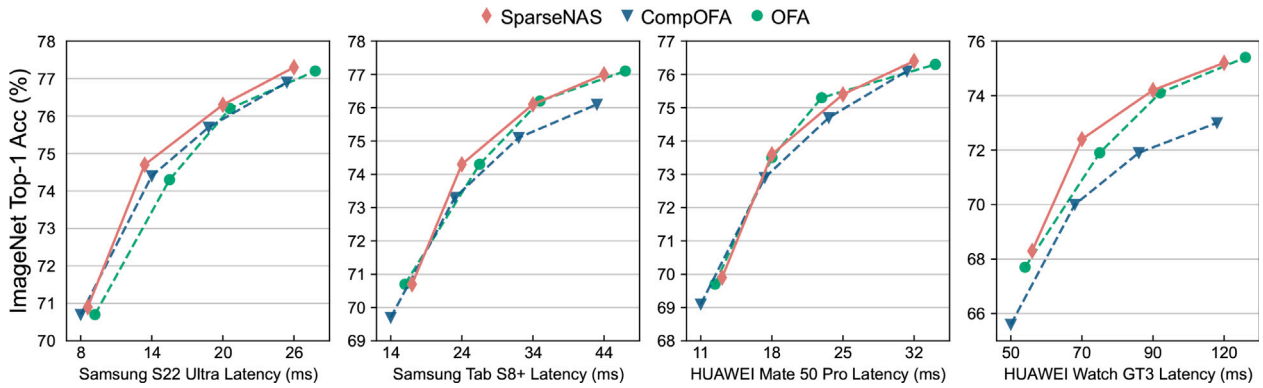


Fig. 7. Comparisons of latency-accuracy trade-off on four edge devices. Sub-networks sampled from SparseNAS, CompOFA and OFA have the same MACs and batch size = 1. The model accuracy is evaluated on the ImageNet validation dataset.

network performance and cost than OFA. Compared to the latest cell-based methods, SparseNAS reports a 1.2% and 0.6% higher accuracy than SWD-NAS and GENAS, respectively.

5.8. Search space comparison

Fig. 6 demonstrates the performance of all sub-networks within the search space of SparseNAS and CompOFA. SparseNAS provide a larger network family containing 1750 networks, which is 7× bigger than CompOFA. Over **70**% sub-networks in SparseNAS achieve higher network performance than CompOFA's with an average of **1.1**% and a maximum **1.5**% accuracy gap under similar MACs constraints. Sub-networks of SparseNAS† also show higher model performance than CompOFA and CompOFA†. We further compare the model performance of three approaches that share the same basic search space.

5.9. Sub-network performance on smart devices

Considering the application in mobile intelligence domains, SparseNAS can provide specialized DNN models for diverse mobile devices to meet hardware resource constraints. We extensively verified the effectiveness of SparseNAS on the latest smart devices, including Samsung Galaxy S22 Ultra, Samsung Galaxy Tab S8+, HUAWEI Mate 50 Pro and HUAWEI Watch GT3. Fig. 7 shows a detailed accuracy-latency comparison. Sub-networks from different supernets have similar MACs. We measure different latencies by resizing the image resolution of the model inputs. SparseNAS demonstrates a similar model accuracy to OFA and outperforms CompOFA with approximate MACs.

5.10. Generalize to other search space

We generalize SparseNAS to another search space, ProxylessNAS [50], which also contains a large number of different networks ranging from 256M to 968M MACs. We compare SparseNAS with CompOFA in this search space. The training stages of the two methods are identical to the stage in OFA's search space. The number of sub-networks in CompOFA is 243 due to CompOFA's coupling strategy. To build a sparse space, we sample 10 sub-networks for every 10M MAC from 300M to 900M, and the total number of candidates in this new space is 1200.

Table 8 reports the training schedule and training duration of two methods in the space in ProxylessNAS. Both methods first train the biggest network in the search space as a teacher network for knowledge distillation. Our SparseNAS-p takes 12% less training time than CompOFA-p. Fig. 8 depicts the model performance of all sub-networks generated by two methods. Despite the change in basic search space, the sub-space built by SparseNAS also shows higher efficiency and accuracy than CompOFA's.

Table 8
Training duration comparison of CompOFA and SparseNAS applied in ProxylessNAS's space.

Model	Stage	n_{sub}	Epochs	GPU Hours
Teacher	–	–	180	163.0
CompOFA-p	Compound Training-1	4	25	38.8
	Compound Training-2	4	125	193.8
	Total Hours: **232.6**			
SparseNAS-p	Early Predictor	2	30	37.0
	Sparse Training	2	150	157.5
	Total Hours: **194.5**			

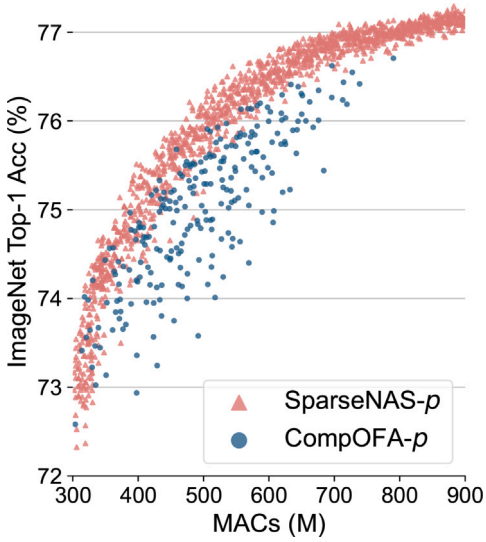


Fig. 8. A network performance comparison between SparseNAS-p and CompOFA-p.

5.11. Ablation study for space size

In this section, we further explore the influence of setting different space sizes as shown in Fig. 9. There are three supernets with different space sizes, including $N = 1750$, $N = 7000$, and $N = 24,500$. $N = 1$ represents a standard DNN. We use the same *early predictor* to sample sub-networks and standard DNNs. All candidates have the same search space settings, network hyper-parameters and the number of training epochs. After training, we randomly sampled a set of different sub-networks from each space for evaluation and comparison.

As the search space becomes bigger, the accuracy of sub-networks shows a slight decrease. This observation strongly verifies our hypothesis in 3.3 that the multi-model forgetting problem has a negative impact

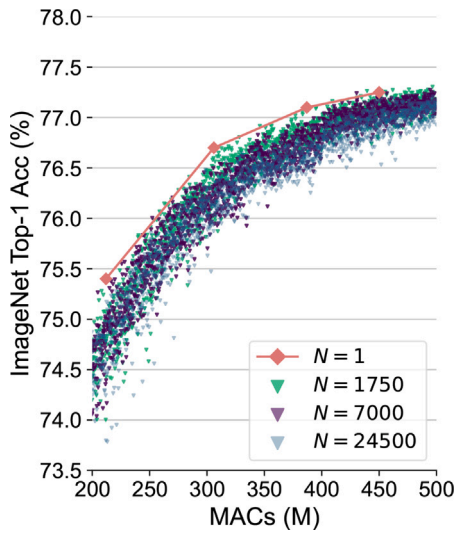


Fig. 9. Network accuracy of sampled sub-networks in different sizes of space. N is the number of candidate sub-networks in the search space.

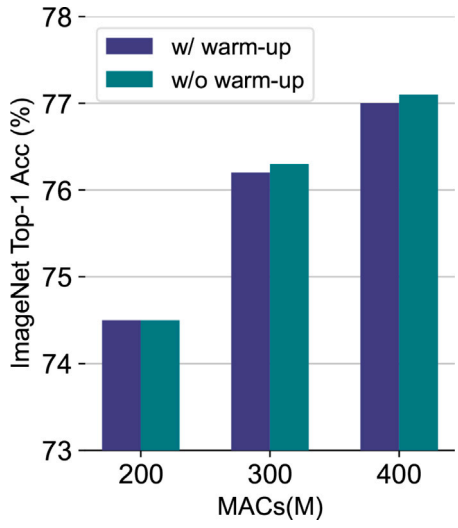


Fig. 10. Performance comparison of supernet training with warm-up epoch and without warm-up in SparseNAS.

on supernet optimization and can be alleviated by narrowing the search space.

5.12. Ablation study for warm-up epoch

In Section 5.5, we mentioned that OFA [13] and CompOFA [15] conduct a warmup training for 25 epochs with a lower learning rate. However, we observe that SparseNAS using warmup epochs results in a slightly lower model accuracy (about 0.1% on average) than without warmup epochs, as shown in Fig. 10. Therefore, we abandoned the 25-epoch warmup in our SparseNAS.

5.13. Ablation study for supernet initialization

As mentioned in Section 4, we create a sparser search space via an early predictor that is generated by a 30-epoch proxy supernet (see Section 5.2). After that, we train a supernet based on the sparse space.

One intuitive idea for initializing the new supernet is inheriting the network weights from the 30-epoch supernet since the new supernet

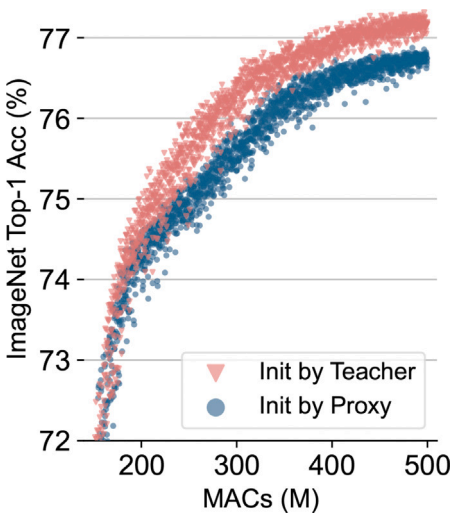


Fig. 11. Performance comparison of two different initialization methods in SparseNAS.

can be regarded as a subset of the 30-epoch one. However, our experiments(see Fig. 11) report that initialized by the proxy supernet leads to a significant performance degradation compared to initialized by the teacher network (i.e., the biggest network for knowledge distillation).

6. Conclusion and future works

6.1. Conclusion

In this work, we introduce SparseNAS, a novel approach for accelerating NAS training by sparsifying the search space. SpraseNAS reduces up to **28**% training cost and speeds up network search by a factor of **45**× with higher network performance compared to the SOTA space simplification approach. Sub-networks provided by SparseNAS achieve an average of **1.1**% and a maximum of **1.5**% higher Top-1 accuracy than their counterparts. These networks also demonstrate better accuracy-latency balance on smart mobile devices. We also generalize SparseNAS to another search space and prove its efficiency and effectiveness. We hope our approach will inspire more NAS researchers toward a deeper understanding of supernet optimization.

6.2. Limitations and future work

There are limitations to this research that point to promising areas for future investigation. Firstly, our work mainly focuses on convolution-based network architecture since we are interested in lightweight network deployment. Other types of neural architecture frameworks (e.g., transformer-based architecture) should be undertaken for further research. In SparseNAS, we build the convolution-based search space based on three dynamic hyperparameters: depth, width expansion and kernel size, as discussed in Section 5.1.1. Each hyperparameter has three different search options. As for transformer-based search space, which excludes the hyperparameter of kernel size, we plan to expand the number of search options of the other two hyperparameters to explore the diversity of the network architectures.

Secondly, the setting of the *early predictor* is based on empirical experiment results on specific search space. Further analysis should be undertaken to study more general predictor settings. Thirdly, our on-device evaluation ranges from server-based hardware to smart mobile devices, but IoT devices or on-chip systems are not included. More work should be done to explore these lighter hardware platforms.

CRediT authorship contribution statement

Hongtao Huang: Writing – original draft, Software, Methodology, Conceptualization. **Xiaojun Chang:** Writing – review & editing, Validation, Supervision. **Lina Yao:** Writing – review & editing, Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[2] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274.

[3] S. Zhang, L. Yao, A. Sun, Y. Tay, Deep learning based recommender system: A survey and new perspectives, ACM Comput. Surv. (CSUR) 52 (1) (2019) 1–38.

[4] A.G. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q.V. Le, H. Adam, Searching for mobilenetv3, in: 2019 IEEE/CVF International Conference on Computer Vision, ICCV, 2019, pp. 1314–1324.

[5] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017, ArXiv abs/1704.04861.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, vol. 30, 2017.

[7] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2019, ArXiv abs/1810.04805, https://api.semanticscholar.org/CorpusID:52967399.

[8] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, R.X. Gao, Deep learning and its applications to machine health monitoring, Mech. Syst. Signal Process. (2019) https://api.semanticscholar.org/CorpusID:125608550.

[9] C. Thapa, P.C.M. Arachchige, S. Camtepe, L. Sun, Splitfed: When federated learning meets split learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 8485–8493.

[10] D. Yao, L. Xiang, Z. Wang, J. Xu, C. Li, X. Wang, Context-aware compilation of dnn training pipelines across edge and cloud, in: Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, vol. 5, (4) 2021, pp. 1–27.

[11] M. Almeida, S. Laskaridis, A. Mehrotra, L. Dudziak, I. Leontiadis, N.D. Lane, Smart at what cost?: characterising mobile deep neural networks in the wild, in: Proceedings of the 21st ACM Internet Measurement Conference, 2021, https://api.semanticscholar.org/CorpusID:238215587.

[12] M. Tan, B. Chen, R. Pang, V. Vasudevan, Q.V. Le, Mnasnet: Platform-aware neural architecture search for mobile, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2019, pp. 2815–2823.

[13] H. Cai, C. Gan, S. Han, Once for all: Train one network and specialize it for efficient deployment, 2020, ArXiv abs/1908.09791.

[14] H. Cai, L. Zhu, S. Han, Proxylessnas: Direct neural architecture search on target task and hardware, 2019, ArXiv abs/1812.00332.

[15] M. Sahni, S. Varshini, A. Khare, A. Tumanov, Compofa: Compound once-for-all networks for faster multi-platform deployment, 2021, ArXiv abs/2104.12642.

[16] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, 2019, ArXiv abs/1806.09055.

[17] X. Dong, Y. Yang, Searching for a robust neural architecture in four gpu hours, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2019, pp. 1761–1770, https://api.semanticscholar.org/CorpusID:198903996.

[18] T. Véniat, L. Denoyer, Learning time/memory-efficient deep architectures with budgeted super networks, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2017, pp. 3492–3500, https://api.semanticscholar.org/CorpusID:3383456.

[19] X. Zhang, Z. Huang, N. Wang, S. Xiang, C. Pan, You only search once: Single shot neural architecture search via direct sparse optimization, IEEE Trans. Pattern Anal. Mach. Intell. 43 (2018) 2891–2904, https://api.semanticscholar.org/CorpusID:53221329.

[20] Q. Yao, J. Xu, W.-W. Tu, Z. Zhu, Differentiable neural architecture search via proximal iterations, 2019, ArXiv abs/1905.13577, https://api.semanticscholar.org/CorpusID:173188550.

[21] E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, J. Tan, Q.V. Le, A. Kurakin, Large-scale evolution of image classifiers, 2017, ArXiv abs/1703.01041.

[22] N. Li, L. Ma, G. Yu, B. Xue, M. Zhang, Y. Jin, Survey on evolutionary deep learning: Principles, algorithms, applications, and open issues, ACM Comput. Surv. 56 (2) (2023) 1–34.

[23] Y. Sun, B. Xue, M. Zhang, G.G. Yen, Evolving deep convolutional neural networks for image classification, IEEE Trans. Evol. Comput. 24 (2) (2019) 394–407.

[24] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, 2016, arXiv preprint arXiv:1611.02167.

[25] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in: AAAI, 2019.

[26] L. Xie, A.L. Yuille, Genetic cnn, in: 2017 IEEE International Conference on Computer Vision, ICCV, 2017, pp. 1388–1397.

[27] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, 2016, arXiv preprint arXiv:1611.01578.

[28] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710.

[29] Y. Liu, K. Zhu, Z. Liu, Ssrnas: Search space reduced one-shot nas by a recursive attention-based predictor with cell tensor-flow diagram, in: 2021 International Joint Conference on Neural Networks, IJCNN, IEEE, 2021, pp. 1–8.

[30] Y. Liu, Z. Yu, Z. Liu, W. Tian, Efficient neural architecture design via capturing architecture-performance joint distribution, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2024, pp. 1738–1746.

[31] Y. Xue, X. Han, Z. Wang, Self-adaptive weight based on dual-attention for differentiable neural architecture search, IEEE Trans. Ind. Inform. (2024).

[32] Y. Xue, X. Han, F. Neri, J. Qin, D. Pelusi, A gradient-guided evolutionary neural architecture search, IEEE Trans. Neural Netw. Learn. Syst. (2024).

[33] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, J. Sun, Single path one-shot neural architecture search with uniform sampling, in: ECCV, 2020.

[34] D. Wang, M. Li, C. Gong, V. Chandra, Attentivenas: Improving neural architecture search via attentive sampling, in: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2021, pp. 6414–6423.

[35] J. Yu, P. Jin, H. Liu, G. Bender, P.-J. Kindermans, M. Tan, T. Huang, X. Song, Q.V. Le, Bignas: Scaling up neural architecture search with big single-stage models, in: ECCV, 2020.

[36] S. You, T. Huang, M. Yang, F. Wang, C. Qian, C. Zhang, Greedynas: Towards fast one-shot nas with greedy supernet, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2020, pp. 1996–2005.

[37] Y. Benyahia, K. Yu, K.B. Smires, M. Jaggi, A.C. Davison, M. Salzmann, C. Musat, Overcoming multi-model forgetting, in: International Conference on Machine Learning, PMLR, 2019, pp. 594–603.

[38] M. Zhang, H. Li, S. Pan, X. Chang, S.W. Su, Overcoming multi-model forgetting in one-shot nas with diversity maximization, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2020, pp. 7806–7815.

[39] B. Guo, L. Xu, T. Chen, P. Ye, S. He, H. Liu, J. Chen, Latency-aware neural architecture performance predictor with query-to-tier technique, IEEE Trans. Circuits Syst. Video Technol. (2023) http://dx.doi.org/10.1109/TCSVT.2023.3287684, 1–1.

[40] J. You, J. Leskovec, K. He, S. Xie, Graph structure of neural networks, in: International Conference on Machine Learning, 2020.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: CVPR, 2009.

[42] T. Huang, S. You, F. Wang, C. Qian, C. Zhang, X. Wang, C. Xu, Greedynasv2: Greedier search with a greedy path filter, in: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2021, pp. 11892–11901.

[43] J.H. Friedman, Greedy function approximation: A gradient boosting machine., Ann. Statist. 29 (2001) 1189–1232.

[44] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[45] G.E. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015, ArXiv abs/1503.02531.

[46] M. Sandler, A.G. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[47] X. Chu, B. Zhang, R. Xu, J. Li, Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search, in: 2021 IEEE/CVF International Conference on Computer Vision, ICCV, 2021, pp. 12219–12228.

[48] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, K. Keutzer, Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2019, pp. 10726–10734.

[49] M. Tan, Q.V. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, 2019, ArXiv abs/1905.11946.

[50] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A.L. Yuille, J. Huang, K.P. Murphy, Progressive neural architecture search, in: ECCV, 2018.