Few-shot Neural Architecture Search

Yiyang Zhao [1] **Linnan Wang** [2] **Yuandong Tian** [3] **Rodrigo Fonseca** [2] **Tian Guo** [1]

Abstract

Efficient evaluation of a network architecture drawn from a large search space remains a key challenge in Neural Architecture Search (NAS). Vanilla NAS evaluates each architecture by training from scratch, which gives the true performance but is extremely time-consuming. Recently, one-shot NAS substantially reduces the computation cost by training only one supernetwork, a.k.a. *supernet*, to approximate the performance of every architecture in the search space via weight-sharing. However, the performance estimation can be very inaccurate due to the co-adaption among operations (Bender et al., 2018). In this paper, we propose *few-shot NAS* that uses multiple supernetworks, called *sub-supernet*, each covering different regions of the search space to alleviate the undesired co-adaption. Compared to one-shot NAS, few-shot NAS improves the accuracy of architecture evaluation with a small increase of evaluation cost. With only up to 7 sub-supernets, few-shot NAS establishes new SoTAs: on ImageNet, it finds models that reach 80.5 top-1 at 600 MB FLOPS and 77.5 top-1 at 238 MFLOPS; on CIFAR10, it reaches 98.72 top-1 without using extra data or transfer learning. In Auto-GAN, few-shot NAS outperforms the previous published results by up to 20%. Extensive experiments show that few-shot NAS significantly improves various one-shot methods, including 4 gradient-based and 6 search-based methods on 3 different tasks in NASBENCH-201 and NasBench1-shot-1.

1. Introduction

Neural Architecture Search (NAS) has attracted lots of interests over the past few years (Zoph et al., 2018; Tan et al., 2019; Baker et al., 2017). Using NAS, many deep learning tasks (Yukang Chen, 2019; Gong et al., 2019; Liu et al., 2019a; Wang et al., 2019b;a) improve their performance without human tuning. vanilla NAS requires a tremendous amount of computational costs (e.g., thousands of GPU hours) in order to find a superior neural architecture (Zoph et al., 2018; Baker et al., 2017; Real et al., 2019), most of which is due to evaluating new architecture proposals by training them from scratch. To reduce the cost, one-shot NAS (Pham et al., 2018; Liu et al., 2019b) proposes to train a single supernet that represents all possible architectures in the search space. With supernet, the performance of architecture can be approximately evaluated by inheriting the corresponding weights from the supernet without training, reducing the search cost to just a few days (hours).

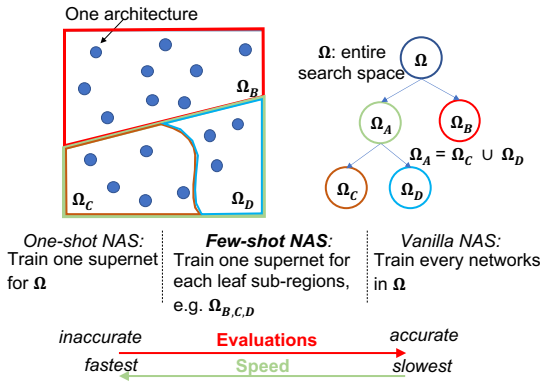
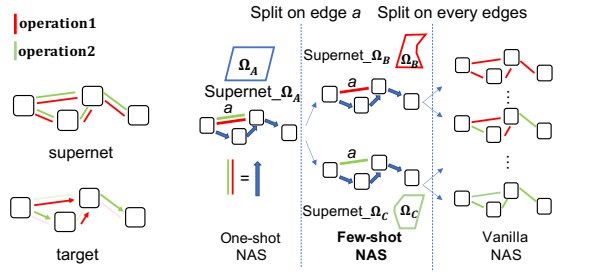


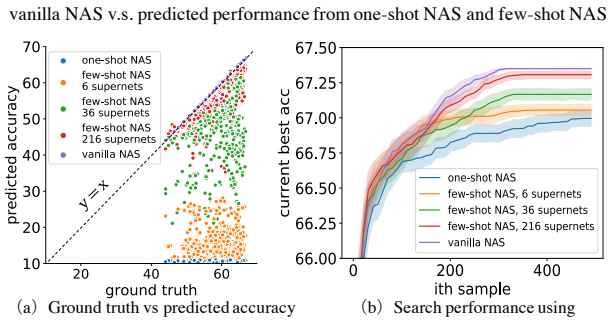
Figure 1. Few-shot NAS is a tradeoff between the vanilla NAS and one-shot NAS that intends to maintain accurate evaluations in vanilla NAS and the speed advantages of one-shot NAS.

However, one-shot NAS suffers from degraded search performance due to the inaccurate predictions from the supernet. On NASBench-201, the best reported Kendall's Tau (Kendall., 1938) (a measurement of rank correlation) between the performance predicted from a supernet and the true performance is only 0.5748 (Yiming Hu, 2020). Other works also have explicitly shown using supernet degrades the final performance due to the inaccurate performance predictions. For example, (Yu et al., 2019b) observes that, without using the supernet, the average performance of NAS algorithms such as ENAS and NAO is 1% higher than using it on NASBench-101, and they also conclude that the supernet never produces the true ranking. Besides, many works (Bender et al., 2018; Yu et al., 2020; Luo et al., 2018; Dong & Yang, 2020; Luo et al., 2020) also show that there is a non-trivial performance gap between the architectures found by one-shot NAS and vanilla NAS. Being consistent



(a) Masking supernet (b) Enumerating a supernet into individual architectures in the search space by splitting every compounded edges

Figure 2. (a) masking supernet to a specific architecture. (b) the motivation of using few-shot NAS to alleviate the co-adaption impact. After splitting on edge a, supernet_Ω_B exclusively predicts architectures in Ω_B, so does supernet_Ω_C.



(a) Ground truth vs predicted accuracy (b) Search performance using regularized evolution

#Supernets	1	6	36	216	1296
Kendall Tau	0.013	0.12	0.26	0.63	1.0

(c) Rank correlations(Kendall Tau) for different numbers of supernets

Figure 3. (a) Using multi-supernets clearly improves the correlation and (c) provides the correlation score (Kendall Tau) at different numbers of supernets in (a); (b) shows the improved performance predictions result in better performance on NAS.

with the analysis in (Yu et al., 2019b), the main reason is that the performance predicted by the supernet has a low correlation with the true performance. As an example, Section 2 shows that inaccurate performance prediction by supernet biases the search towards a wrong direction and hurts both the efficiency and the final results.

In this work, we propose *few-shot NAS* that uses multiple supernets in the architecture search. Instead of having one supernet covering the entire search space, which may be beyond its capacity and suffer from the co-adaption effect from the compound edges, using multiple supernets effectively addresses these issues by having each supernets modeling one part of the search space and by reducing the number of compound edges. We did a proof-of-concept in Fig. 3 to verify the idea.

To partition the search space, we recursively split the compound edges on the supernet. Figure 2 shows an exemplar search tree: the root represents the entire search space Ω (i.e., all possible network operations), and leaves represent actual architectures in Ω. Moving down along one edge dissect one compound edge into several supernets covering different parts of Ω. Although few-shot NAS increases the number of supernets, supernets can be trained efficiently by using a cascade of transfer learning: first, the root supernet is trained, then the weights of the root are inherited to its children as initialization and fine-tuned, and so on. In this manner, we construct a collection of supernets, each of which is responsible for a region of the search space. Please refer to section 3 for the methodology details.

With only 5 sub-supernets, we show that our *few-shot NAS* greatly improved many existing NAS algorithms on NASBENCH-201 (Dong & Yang, 2020) and several popular deep learning tasks in Section 4. Particularly, with our *few-shot NAS*, we found SOTA efficient models that demonstrate 80.5 top-1 at 600 MB FLOPS and 77.5 top-1 at 238 MFLOPS on ImageNet, and 98.72 top-1 on CIFAR-10 without using extra data or transferring weights from a network pre-trained on ImageNet. Moreover, by re-using the same search code from AUTOGAN (Gong et al., 2019), *few-shot NAS* also improved the results in (Gong et al., 2019), from 12.42 to 10.73 in FID score.

2. Background and Motivation

The negative impact of co-adaption of operations from a compound edge was first identified by Bender et al (Bender et al., 2018); and they show that the compound operations on an edge of the supernet can degrade the correlation between the estimated performance from a supernet and the true performance from training-from-scratch. While Bender et al primarily focused on using drop path or dropout to ensure a robust supernet for performance prediction, our method was motivated by the following observation on one-shot NAS and vanilla NAS.

One-shot NAS uses a supernet to predict the performance of a specific architecture by deactivating the extra edges w.r.t a target architecture on the supernet via masking (Fig. 2(a)), then perform evaluations using the masked supernet. Therefore, we can view supernet as a representation of search space Ω, and by masking, supernet can transform to any architectures in Ω. This also implies we can enumerate all the architectures in Ω by recursively splitting every compound edge in a supernet. Fig. 2(b) illustrates the splitting process, the root is the supernet and leaves are individual architectures in the search space Ω; the figure illustrates the case of splitting the compound edge a, and the recursively split follows similar procedures on all compound edges. In Fig. 2(b), one-shot NAS is the fastest but the most inaccurate in evaluations, while vanilla NAS is the most accurate in evaluations but the slowest. However, the middle ground,

Table 1. The definition of notations used through the paper.

Ω	the whole architecture space	$\mathcal{A}$	an architecture in the architecture space	m	number of operations in the architecture space
$\mathcal{S}$	supernet	N_i	the ith node in the architecture space	n	number of nodes in the architecture space
$\Omega^{'}$	a sub-region of the whole architecture space	E_{ij}	the mixture operations between node i and j	$\mathcal{W}$	weights of neural network
$\mathcal{S}^{\Omega^{'}}$	a sub-supernet	$f(\mathcal{A})$	the evaluation of $\mathcal{A}$	$f(\mathcal{S}_{\mathcal{A}})$	the evaluation of $\mathcal{A}$ by supernet

i.e. using multiple supernets, between one-shot NAS and vanilla NAS remains unexplored.

In a supernet, the effect of co-adaption results from combined operations on edges; therefore the evaluation of vanilla NAS is the most accurate. Based on this logic, it seems using several sub-supernets is a reasonable approach to alleviate the co-adaption effect by dissecting a compound edge into several separate sub-supernets that take charge of different sub-regions of the search space. For example, Fig. 2(b) shows few-shot NAS eliminates one compound edge a after splitting, resulting in two supernets for Ω_B and Ω_C, respectively. So, the predictions from resulting sub-supernet are free from the co-adaption effects from the split compound edge a.

We designed a controlled experiment to verify the assumption that *using multi-supernets improves the performance prediction*. First, we designed a search space having 1296 architectures, and trained each architecture toward the convergence to collect the final evaluation accuracy as the ground truth. Then we split the one-shot version of supernet into 6, 36, 216 sub-supernets following the procedures in Fig. 2(b). Finally, we trained each supernet with the same training pipeline in (Bender et al., 2018), and compared the predicted 1296 architecture performance to the ground truth using 1 (one-shot NAS), 6, 36, 216 supernets. Fig. 3 visualizes the results, and it indicates using multi-supernets significantly improves the correlation between predicted performance and the ground truth. Specifically, in Fig. 3(c) the Kendall's Tau (Kendall., 1938) ranking correlation of using 1 supernet (one-shot NAS), 6 supernets, 36 supernets, 216 supernets are 0.013, 0.12, 0.26, 0.63, respectively. As a result, the search algorithm takes fewer samples to find better networks due to more accurate performance predicted from supernets (Fig. 3(b)).

In sec 4, we conducted extensive experiments on various applications to ensure the proposed idea will generalize to other domains, including image recognition, language modeling, and image generation using Generative Adversarial Network (GAN). While all the experiments suggest few-shot NAS is an effective approach, the proposed splitting process indicates the number of supernets exponentially increases with the number of splits, rendering new computation challenges. In the methodology section, we introduce a cascade of transfer learning to speed up the supernet training. Please refer to sec. 3 for details.

3. Methodology

In designing *few-shot NAS*, we answer the following several key questions: *(i)* how to divide the search space represented by the one-shot model to sub-supernets and how to choose the number of sub-supernets given a search time budget (Section 3.1)? *(ii)* how to reduce the training time of multiple sub-supernets (Section 3.2)?; We also describe how to integrate *few-shot NAS* with existing NAS algorithms in Section 3.3 and Section 3.4.

3.1. Design of Split Strategy

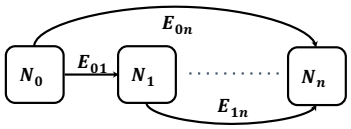


Figure 4. A generic architecture space.

Our empirical observation from Section 2 can be summarized as following: the evaluation $f(\mathcal{S}_{\mathcal{A}}^{\Omega^k})$ of an architecture A using a sub-supernet $\mathcal{S}^{\Omega^k}$ is closer to the true accuracy $f(\mathcal{A})$ as Ω^k gets smaller, i.e., deeper in the tree. However, the prediction improvement for A diminishes with any sub-region Ω^p smaller than sub-region Ω^q where $A \in \Omega^q$. Furthermore, the time to split the initial architecture space Ω grows exponentially with tree depth. In short, the ideal split would be determined based on individual architecture and find the sub-supernet at the shallowest tree depth.

Definition of a Generic NAS Space. Before we describe our split strategy, we first define a generic NAS space that is compatible with one-shot NAS. We use this architecture space for introducing some necessary concepts that will be used throughout the paper. The whole architecture space Ω is represented by a directed acyclic graph (DAG) shown in Figure 4. Each node denotes a latent state, e.g., feature maps in CNNs, and each edge represents a mixture of operations. We consider an architecture space with n nodes and m operations. Each node i is denoted as N_i where $i \in [1, n]$; E_{ij} represents a set of m edges that connects node N_i and N_j, where m denotes the number of operations. Any architecture candidate that can be found in the space has only one edge in E_{ij}. In other words, there is exactly one operation from N_i to N_j in any architecture candidate. In addition, an available architecture at least has one edge from

Table 2. Rank correlation analysis using Kendall's Tau (Kendall., 1938) for different split strategies.

#split_edges	#Supernet	Mean	Std.
1	5	0.653	0.012
2	25	0.696	0.016
3	125	0.752	0.018

its predecessor node.

Split Procedure Analysis. Given a search space, e.g., one that was depicted in Figure 4, the supernet with m operations and n nodes can be split into a total of $m^{n(n-1)/2}$ architectures by all $\frac{n(n-1)}{2}$ edges. Training all $m^{n(n-1)/2}$ architectures, as done by the vanilla NAS, is time-consuming but can provide accurate rank information. To evaluate the impact of edge splitting on ranking architectures, we calculate the Kendall's Tau for different splitting strategies by leveraging the NASBENCH-201 (more details of this dataset in Section 4.1).

Specifically, we take the same search space used by NASBENCH-201 with 5 operation types and 4 nodes and split it in a total of 6, 15, and 20 ways by splitting 1, 2, and 3 edges, respectively. For each split, we train a total of 5^k sub-supernets where $k \in [1, 2, 3]$. In all, we train a total of 2905 sub-supernets to convergence for calculating the Kendall Tau for each split.

Table 2 shows the rank correlation when splitting with different numbers of edges. First, similar to what we have observed in Section 2, increasing the number of split edges leads to a higher rank correlation. Second, given the same number of edges to split, the exact choice of which edge to split has negligible impact on the rank correlation as indicated by the low standard deviation. Therefore, we can randomly choose which edge(s) to split and focus on how many edge(s) to split. In this work, we pre-define a training time budget T. If the total training time of supernet and all currently trained sub-supernets exceeds T, we will stop the split to avoid training more sub-supernets.

3.2. Transfer Learning

Using the progressive split strategy, the number of sub-supernets grows exponentially with the number of nodes n. Directly training all the resulting sub-supernets can be computationally intractable and also goes against the insight of one-shot NAS. In this section, we describe how we use a transfer learning technique to accelerate the training procedure of sub-supernets.

Similar to how an architecture candidate $\mathcal{A}$ inherits weights $\mathcal{W}_{\mathcal{A}}$ from the supernet weights $\mathcal{W}_{\mathcal{S}}$, we allow a sub-supernet $\mathcal{S}^{\Omega^{'}}$ to inherit weights from its parent sub-supernet. For example, in Figure 2(b), after training the supernet of Ω_A, the supernet of Ω_B and Ω_C can inherit the weights from shared operations in supernet of Ω_A as initialization and then start training. By using transfer learning, each sub-supernet only needs very small epochs to converge compared to training from scratch.

3.3. Integration with Gradient-based Algorithms

Overview with Gradient-based NAS. Gradient-based algorithms work on a continuous search space, which can be converted from the DAG. Gradient-based algorithms treat the NAS as a joint optimization problem where both the weight and architecture distribution parameters are optimized *simultaneously* by training (Liu et al., 2019b). In other words, gradient-based algorithms are designed for and used with one-shot NAS.

To use gradient-based algorithms with our *few-shot NAS*, we *first* train the supernet until it converges. Then, we split the supernet $\mathcal{S}$ to several sub-supernets as described in Section 3.1 and initialize these sub-supernets with weights and architecture distribution parameters transferred from their parents. Next, we train these sub-supernets to converge and repeat the above steps if the predefined search time budget has not been depleted. Lastly, we choose the sub-supernet $\mathcal{S}^{\Omega^{'}}$ with the lowest validation loss from all the converged sub-supernets, and pick the best architecture $\mathcal{A}^*$ from the $\mathcal{S}^{\Omega^{'}}$ based on the architecture distribution parameters.

3.4. Integration with Search-based Algorithms

Overview. Search-based algorithms can work with both one-shot and vanilla NAS. To start, search-based algorithms often need to pick the first few architectures. Then search-based algorithms evaluate the performance of these architectures either through training, in the case of vanilla NAS, or evaluating by a pre-trained supernet, in the case of one-shot NAS. For vanilla NAS, it is not strictly necessary to train these architectures to converge, and one can use early stopping to obtain an intermediate result. After warm-up, search-based algorithms will sample the next architecture $\mathcal{A}$ from the search space based on its previous architecture, until an architecture with satisfiable performance, e.g., test accuracy, is found.

To use search-based algorithms with our *few-shot NAS*, we will first train a number of sub-supernets by using progressive split and transfer learning, similar to what was described in Section 3.3. These converged sub-supernets will be used as the basis to evaluate the performance of sampled architectures. For example, if a sampled architecture $\mathcal{A}$ falls into sub-supernet $\mathcal{S}^{\Omega^{'}}$, we will evaluate its performance $f(\mathcal{S}^{\Omega^{'}}_{\mathcal{A}})$

by inheriting the weights $\mathcal{W}_{\mathcal{S}^{\Omega'}}$. Once the search algorithms complete, we will pick the top K architectures with the best performance empirically and train these architectures to converge and select the final architecture based on test error.

4. Experiments

To evaluate the performance of *few-shot NAS* in reducing the approximation error associated with supernet and in improving search efficiency of search algorithms, we conducted two types of evaluations. The first is based on an existing NAS dataset and the second type is comparing the architectures found by using *few-shot NAS* to state-of-the-art results in popular application domains.

We first evaluate the search performance of *few-shot NAS* in different NAS algorithms. We use two metrics(search cost and accuracy) to evaluate search efficiency of DARTS, PC-DARTS, ENAS, SETN, REA, REINFORCE, HB, BOHB, SMAC, and TPE (Liu et al., 2019b; Xu et al., 2020; Pham et al., 2018; Dong & Yang, 2019; Real et al., 2019; Zoph et al., 2018; Li et al., 2018; Falkner et al., 2018; Hutter et al., 2011; Bergstra et al., 2012) by one-shot/few-shot models on NASBENCH-201. We also evaluate the search performance of *few-shot NAS* with DARTS, PCDARTS, and ENAS on NasBench1-shot-1 (Zela et al., 2020b). Then we extend *few-shot NAS* to different open domain search spaces and show that the found architectures significantly outperform the ones obtained by one-shot NAS. Our found architectures also reach state-of-the-arts results in CIFAR10, ImageNet, AutoGAN (Gong et al., 2019), and Penn Treebank.

4.1. Evaluation on NASBENCH-201

We use NASBENCH-201, a public architecture dataset, which provides a unified benchmark for up-to-date NAS algorithms (Dong & Yang, 2020). NASBENCH-201 contains *all* 15625 architectures in the search space, making it possible to evaluate the efficiency of gradient-based search algorithms. In contrast, prior datasets such as NASBENCH-101 (Ying et al., 2019) do not provide all possible architectures in their search space. For each architecture, NASBENCH-201 contains information such as size, training and test time, and accuracy on CIFAR-10, CIFAR-100, and ImageNet-16-120. Consequently, NAS algorithms can leverage information on each architecture without time-consuming training.

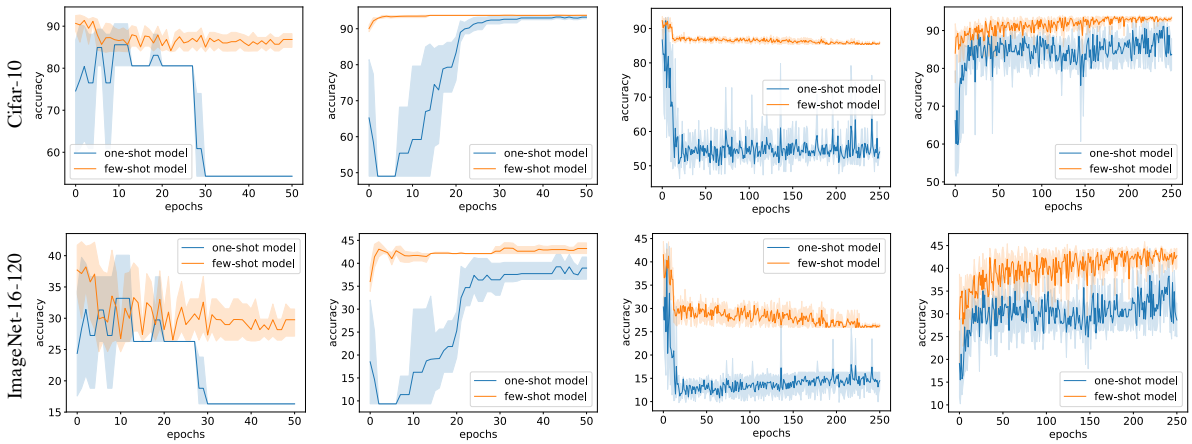
4.1.1. GRADIENT-BASED ALGORITHMS

Methodology. The supernet corresponds to NASBENCH-201 has five nodes and five operations. Based on the split method described in Section 3.1, we split one edge in search space and obtain five sub-supernets. For this experiment, we did not train sub-supernets with transfer learning described in Section 3.2. This is to compare anytime performance between one-shot and *few-shot NAS*, by keeping the same training epochs. Due to limit of the computing resource, we only split one edge in the search space since without transfer learning, training more sub-supernets exponentially increases the time cost. We chose a number of recently proposed gradient-based search algorithms including DARTS and ENAS for evaluating *few-shot NAS*. We used two metrics: *(i) test accuracy* is obtained by evaluating the final architecture found by a NAS algorithm; and *(ii) search time* describes the total time of search including supernet training and validation time.

Result Analysis. Figure 5 shows the anytime test accuracy of searched models. In the case of training on CIFAR-10 (first row), when using one-shot NAS, both DARTS and ENAS were trap in a bad performance region, which is exactly consistent with the original paper (Dong & Yang, 2020). The main reason caused by this is one-shot NAS would fall into a sub-optimal region due to inaccurate performance prediction. In contrast, *few-shot NAS* maintained a high quality of searched models since multiple supernets have more accurate performance to guide the search. Additionally, in the case of PCDARTS and SETN, even though one-shot NAS was able to eventually find a good architecture, i.e., with more than 90% accuracy, it took 10X more search epochs than our *few-shot NAS*. As *few-shot NAS* took on average 4.8X of that of one-shot NAS (without transfer learning) in training additional supernets, this translates to more than twice search time savings. In short, we show that by using *few-shot NAS*, gradient-based algorithms can have more efficient search, both in terms of found architectures and the number of search epochs.

4.1.2. SEARCH-BASED ALGORITHMS

Methodology. As described in Section 3.1, we define the search time budget of *few-shot NAS* to be less than twice as that of one-shot NAS. Therefore, we only split the supernet by the first edge in the first node N_1 to five sub-supernets. For this experiment, we used transfer learning described in Section 3.2 for training sub-supernets. We ran each search-based algorithms for 50 times. We chose six different search-based algorithms including REA, REINFORCE, BOHB , HB, SMAC, and TPE. We evaluate the effectiveness of *few-shot NAS* by following the training procedure described in Section 3.4. We used two metrics to evaluate the performance of search-based algorithms. We first use i^{th} *best accuracy* to denote the best test accuracy after searching all i architectures. This metric helps to quantify the search efficiency, as a good search algorithm is expected to find an architecture with higher test accuracy with fewer samples. The second metric is *total search time* which defines the time for a search algorithm to find the satisfiable architecture(s).



(a) DARTS (b) PCDARTS (c) ENAS (d) SETN

Figure 5. Anytime accuracy comparison of state-of-the-art gradient-based algorithms on *few-shot NAS*. Shaded area represents the highest and lowest values based on five runs.

Table 3. Rank correlation analysis using Kendall's Tau on NASBENCH-201 with different methods.

Method	Kendall's Tau	Cost(Hours)
Random	0.0022	0
EN^{2}AS (Zhang et al., 2020)	0.378	N/A
One-shot	0.5436	6.8
Angle (Yiming Hu, 2020)	0.5748	N/A
Few-shot(5-supernets)	**0.653**	10.1
Few-shot(25-supernets)	**0.696**	18.6
Few-shot(125-supernets)	**0.752**	31.8

Result Analysis. Figure 6 compares the best accuracy after searching a certain number of architectures. We *first* observe that *few-shot NAS* was able to find the global optimal architectures in around 3500 samples when using REA, and 3000 samples when using REINFORCE. Second, we see that with REA, BOHB, and TPE, *few-shot NAS* significantly improved the search efficiency over one-shot NAS. Lastly, with REINFORCE, HB, and SMAC, all three NAS algorithms achieved slightly better search efficiency with *few-shot NAS* compared to using one-shot NAS.

Figure 6(g) compares the search time. All search-based algorithms took three to four orders of magnitude GPU hours when using vanilla(standard) NAS, compared to both one-shot NAS and our *few-shot NAS*. Specifically, *few-shot NAS* only took slightly more search time, about 10 hours, compared to one-shot NAS. Both one-shot and *few-shot NAS* finished the search within 24 hours. Finally, we compare the rank correlation between our *few-shot NAS* and other approaches in Table 3. The good correlation achieved by *few-shot NAS* indicates its effectiveness in finding higher accuracy architecture.

4.2. Evaluation on NasBench1-shot-1

We evaluate our few-shot NAS on NasBench1-shot-1 (Zela et al., 2020b), which is a public neural architecture dataset similar to NASBENCH-201. Instead of creating a new search space like NASBENCH-201, NasBench1-shot-1 enables one-shot search algorithms implementing on NASBENCH-101 and its search space is consistent with NASBENCH-101. We split three sup-supernets as our few-shot models and keep the same setting with NASBENCH-201.

Result Analysis. Figure 7 shows search results on NasBench1-Shot-1. DARTS, PCDARTS, and ENAS by few-shot model can quickly find good architectures, i.e. test error less than 0.07 while one-shot PCDARTS require near 30 epochs to find an architecture with similar performance. For all search algorithms except random search by one-shot models, they took about more than 5X more search epochs than our few-shot model to find an architecture with similar accuracy. Therefore, our few-shot NAS improved the search efficiency both in found architectures and search time.

4.3. Deep Learning Applications

CIFAR-10 in Practice. We chose three state-of-the-art NAS algorithms, one gradient-based algorithm(DARTs) and two search-based algorithms including regularized evolution (REA) and LaNas (Liu et al., 2019b; Real et al., 2019; Wang et al., 2019a), for evaluating the effectiveness of *few-shot NAS*. We also compared our results with the most recent NAS algorithms listing in table 4. We used the same search space based on the original DARTS, REA and LaNas papers.

Table 4 compares the search performance. We see that using DARTS with our *few-shot NAS* outperformed searching

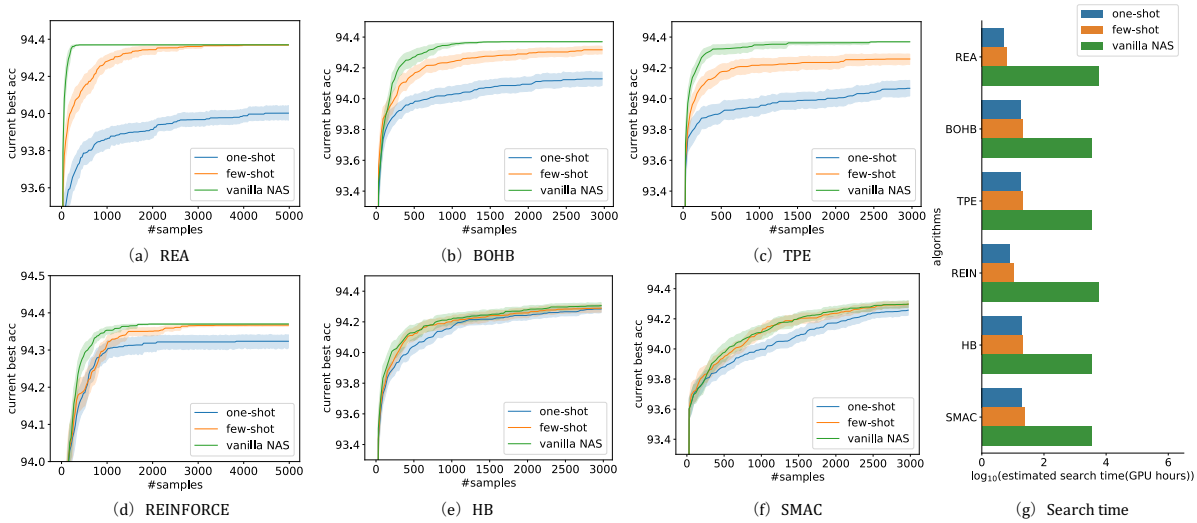


Figure 6. Current best accuracy and search time comparison of popular search-based algorithms. All algorithms were ran for 50 times for one-shot, few-shot and vanilla NAS, respectively.

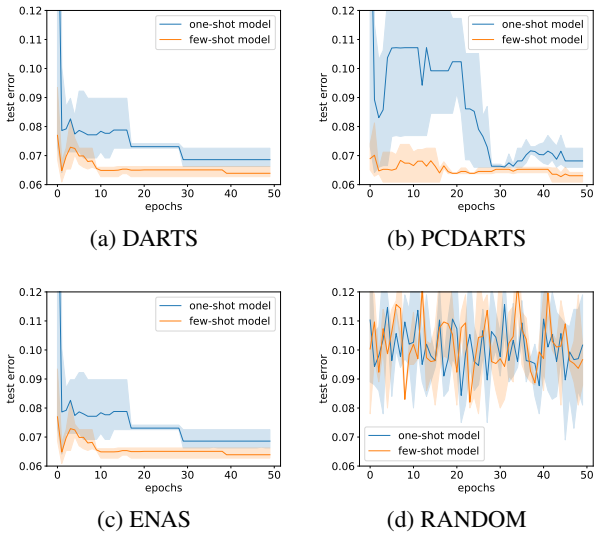


Figure 7. Search results on Nasbench1-shot-1. Each search algorithm ran 3 times.

directly with one-shot NAS by 0.43 lower error. Further, the architecture found with *few-shot NAS* also matches the state-of-the-art results on CIFAR-10. Additionally, *few-shot NAS* only incurred a 35% search time increase. Similarly, *few-shot NAS* also improved the search efficiency of REA upon one-shot NAS, finding an architecture with 0.21 lower error with only 16.7% more search time. For LaNas, few-shot decrease the test error from 1.68 to 1.58 with only 26.7% extra search time. All of our few-shot searched results outperform results of other methods in the table with the same training setup. In short, *few-shot NAS* improved the efficiency of existing state-of-the-art search algorithms.

Neural Architecture Search on ImageNet. We selected two state-of-the-art NAS algorithms working on ImageNet, including ProxylessNAS and Once-for-All NAS(OFA) (Cai et al., 2019; 2020). Our few-shot NAS keeps the same training setup with ProxylessNAS and OFA.

Table 5 compares the search performance. We can see that our few-shot NAS significantly improved the accuracy and kept similar FLOPs numbers on both two NAS algorithms with one-shot model. Our few-shot OFA Net also achieves the best top1 accuracy compared to the models with different search methods on the same scales of Flop numbers.

Comparison to AUTOGAN (Gong et al., 2019). AUTOGAN was proposed to search for a special architecture called GAN, which consists of two competing networks. The networks, a generator and a discriminator, play a min-max two-player game against each other. We followed the same setup described in the AUTOGAN paper. We used Inception score (IS)(higher is better) and Frchet Inception Distance (FID) (Salimans et al., 2016)(lower is better) to evaluate the performance of GAN. Table 6 compares the top three performing GANs found by both original AUTOGAN and with using our *few-shot NAS*. We observe that using *few-shot NAS*, the inception score of the best architecture was improved from 8.55 to 8.63 and the FID was reduced from 12.42 to 10.73. Additionally, the top two architectures found using *few-shot NAS* had very close performance, one with the lowest inception score and the other with the lowest FID. In short, all three architectures found by *few-shot NAS* had better inception score and FID than state-of-the-art results.

Table 4. Applying *few-shot NAS* on existing NAS methods on CIFAR-10 using the NASNet search space. Our results demonstrate that 1) *few-shot NAS* consistently improves the final accuracy of various one-shot based NAS methods under the same setup. Please note we only extend one-shot based DARTS, REA, and LaNAS by replacing the single supernet with 7 supernets in their public release; 2) after integrating with multiple supernets, few-shot DARTS achieves SOTA 98.72% top-1 accuracy on CIFAR-10 using the cutout (Devries & Taylor, 2017) and auto-augmentation (Cubuk et al., 2018). Without auto-augmentation, few-shot DARTS-Small still consistently outperforms existing models that have similar parameters.

Method	Data Augmentation	#Params	Err	GPU days
NASNet-A (Zoph et al., 2018)	cutout	3.3M	2.65	2000
AmoebaNet-B-small (Real et al., 2019)	cutout	2.8M	2.50±0.05	3150
AmoebaNet-B-large (Real et al., 2019)	cutout	34.9M	2.13±0.04	3150
AlphaX (Wang et al., 2019b)	cutout	2.83M	2.54±0.06	1000
NAO (Luo et al., 2018)	cutout	3.2M	3.14±0.09	225
DARTS (Liu et al., 2019b)	cutout	3.3M	2.76±0.09	1
P-DARTS (Chen et al., 2019)	cutout	3.4M	2.5	0.3
PC-DARTS (Xu et al., 2020)	cutout	3.6M	2.57±0.07	0.3
Fair-DARTS (Chu et al., 2019b)	cutout	3.32M	2.54±0.05	3
BayeNAS (Zhou et al., 2019)	cutout	3.4M	2.81±0.04	0.2
CNAS (Lim et al., 2020)	cutout	3.7M	2.60±0.06	0.3
MergeNAS (Wang et al., 2020)	cutout	2.9M	2.68±0.01	0.6
ASNG-NAS (Akimoto et al., 2019)	cutout	3.32M	2.54±0.05	0.11
XNAS (Nayman et al., 2019)	cutout + autoaug	3.7M	1.81	0.3
one-shot REA	cutout + autoaug	3.5M	2.02±0.03	0.75
one-shot LaNas (Wang et al., 2019a)	cutout + autoaug	3.6M	1.68±0.06	3
few-shot DARTS-Small	cutout	3.8M	**2.31±0.08**	1.35
few-shot DARTS-Large	cutout	45.5M	**1.92±0.08**	1.35
few-shot DARTS-Small	cutout + autoaug	3.8M	**1.70±0.08**	1.35
few-shot DARTS-Large	cutout + autoaug	45.5M	**1.28±0.08**	1.35
few-shot REA	cutout + autoaug	3.7M	**1.81±0.05**	0.87
few-shot LaNas	cutout + autoaug	3.2M	**1.58±0.04**	3.8

Table 5. Applying *few-shot NAS* on existing NAS methods on ImageNet using the EfficientNet search space. Being consistent with the results on CIFAR-10 in Table. 4, the final accuracy from few-shot OFA and ProxylessNAS also outperforms their original one-shot version under the same setting, except for replacing the single supernet with 5 supernets. Particularly, Few-shot OFA-Large achieves SoTA 80.5% top1 accuracy at 600M FLOPS.

Method	Space	#Params	#FLOPs	Top 1 Acc(%)	GPU hours
AutoSlim (Yu & Huang, 2019)	Mobile	5.7M	305M	74.2	N/A
MobileNetV3-Large (Howard et al., 2019)	Mobile	5.4M	219M	74.7	N/A
MnasNet-A2 (Tan et al., 2019)	Mobile	4.8M	340M	75.6	N/A
FBNetV2-L1 (Wan et al., 2020)	Mobile	N/A	325M	77.2	600
EfficientNetB0 (Tan & Le, 2019)	Mobile	5.3M	390M	77.3	N/A
AtomNAS (Mei et al., 2020)	Mobile	5.9M	363M	77.6	N/A
few-shot OFA_Net-Small	Mobile	5.6M	238M	**77.50**	68
MobileNetV2 (Sandler et al., 2018)	Mobile	6.9M	585M	74.7	N/A
ShuffleNet-V2 (Ma et al., 2018)	Mobile	N/A	590M	74.9	N/A
ProxylessNAS (Cai et al., 2019)	Mobile	7.12M	465M	75.1	200
ChamNet (Dai et al., 2019)	Mobile	N/A	553M	75.4	N/A
RegNet (Radosavovic et al., 2020)	Mobile	6.1M	600M	75.5	N/A
OFA_Net (Cai et al., 2020)	Mobile	9.1M	595M	80.0	40
few-shot ProxylessNAS	Mobile	4.87M	521M	**75.91**	280
few-shot OFA_Net-Large	Mobile	9.2M	600M	**80.50**	68

Table 6. Apply *few-shot NAS* to AutoGAN by only replacing the supernet with 3 supernets in their public release. Few-shot AutoGAN demonstrates up to 20% better performance than the original one-shot AutoGAN.

Method	Inception Score	FID Score
ProbGAN(He et al., 2019)	7.75±.14	24.60
SN-GAN(Miyato et al., 2018b)	8.22±.05	21.70±.01
MGAN(Hoang et al., 2018)	8.33±.12	26.7
Improving MMD GAN(Wang et al., 2019c)	8.29	16.21
AutoGAN-top1(Gong et al., 2019)	8.55±.10	12.42
AutoGAN-top2	8.42±.06	13.67
AutoGAN-top3	8.41±.12	13.87
few-shot AutoGAN-top1	8.60±.10	**10.73±.10**
few-shot AutoGAN-top2	**8.63±.09**	10.89±.20
few-shot AutoGAN top3	8.52±.08	12.20

PENN TREEBANK in Practice (Marcus et al., 1994). Lastly, we evaluate *few-shot NAS* on Penn Treebank (PTB), a widely-studied benchmark for language models. We used the same search space and training setup as the original DARTS to search RNN on PTB. By using *few-shot NAS*, we achieved the state-of-the-art test Perplexity of **54.89** with an overall cost of 1.56 GPU days. In comparison, the original DARTS found an architecture with worse performance (55.7 test Perplexity) with 1 GPU day.

5. Related Works

Weight-sharing supernet was first proposed as a way to reduce the computational cost of NAS (Pham et al., 2018). Centering around supernet, a number of NAS algorithms including gradient-based (Liu et al., 2019b; Xu et al., 2020; Dong & Yang, 2019) and search-based (Bender et al., 2018; Chu et al., 2019a; Guo et al., 2019) were proposed. The search efficiency of these algorithms is dependent on the ability of supernet to approximate architecture performance.

To improve the supernet approximation accuracy, Bender et al. (Bender et al., 2018) proposed a path dropout strategy that randomly drops out weights of the supernet during training. This approach improves the correlation between one-shot NAS and individual architecture accuracy by reducing weight co-adaptation. In a similar vein, Guo et al. (Guo et al., 2019) proposed a single-path one-shot training by only activating the weights from one randomly picked architecture in forward and backward propagation. Additionally, Yu et al. (Yu et al., 2019a) found that training setup greatly impacts supernet performance and identified useful parameters and hyper-parameters. Lastly, an angle-based approach (Zhiyuan Li, 2020; Arora et al., 2019; Carbonnelle & Vleeschouwer, 2018) was proposed to improve the supernet approximation accuracy for individual architecture (Yiming Hu, 2020) and was shown to improve the architecture rank correlation. However, our few-shot models achieved better rank correlation than this angle-based approach(see table 3). Our work focuses on reducing the supernet approximation error by dividing the supernet to a few sub-supernets to eliminate the co-adaption among supernet operations. As such, our work is complementary and can be integrated into the aforementioned work.

6. Conclusion

In this work, we proposed a novel way, *few-shot NAS*, to balance the search time and the performance of found architecture. *Few-shot NAS* leverages the search space between one-shot NAS and vanilla NAS. Our experiments show that few-shot NAS outperformed both in NAS benchmark dataset and different practical application domains.

References

Akimoto, Y., Shirakawa, S., Yoshinari, N., Uchida, K., Saito, S., and Nishida, K. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 171–180, 2019.

Arora, S., Li, Z., and Lyu, K. Theoretical analysis of auto rate-tuning by batch normalization. In *International Conference on Learning Representations*, 2019.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. *International Conference on Learning Representations*, 2017.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning*, 10–15 Jul 2018.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. *Nips*, 2012.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/pdf/1812.00332.pdf.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

Carbonnelle, S. and Vleeschouwer, C. D. On layer-level control of DNN training and its impact on generalization. *CoRR*, abs/1806.01603, 2018.

Chen, X., Xie, L., Wu, J., and Tian, Q. Progressive DARTS: bridging the optimization gap for NAS in the wild. *CoRR*, abs/1912.10952, 2019. URL http://arxiv.org/abs/1912.10952.

Chu, X., Zhang, B., Xu, R., and Li, J. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *CoRR*, abs/1907.01845, 2019a. URL http://arxiv.org/abs/1907.01845.

Chu, X., Zhou, T., Zhang, B., and Li, J. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *CoRR*, abs/1911.12126, 2019b. URL http://arxiv.org/abs/1911.12126.

Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018. URL http://arxiv.org/abs/1805.09501.

Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., Vajda, P., Uyttendaele, M., and Jha, N. K. Chamnet: Towards efficient network design through platform-aware model adaptation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11390–11399, 2019. doi: 10.1109/CVPR.2019.01166.

Devries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. URL http://arxiv.org/abs/1708.04552.

Dong, X. and Yang, Y. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

Falkner, S., Klein, A., and Hutter, F. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Gong, X., Chang, S., Jiang, Y., and Wang, Z. Autogan: Neural architecture search for generative adversarial networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. *CoRR*, abs/1904.00420, 2019. URL http://arxiv.org/abs/1904.00420.

He, H., Wang, H., Lee, G.-H., and Tian, Y. Probgan: Towards probabilistic gan with theoretical guarantees. In *International Conference on Learning Representations(ICLR)*, 2019.

Hoang, Q., Nguyen, T. D., Le, T., and Phung, D. MGAN: Training generative adversarial nets with multiple generators. In *International Conference on Learning Representations*, 2018.

Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019. URL http://arxiv.org/abs/1905.02244.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the conference on Learning and Intelligent OptimizatioN (LION 5)*, 2011.

Kendall., M. G. A new measure of rank correlation. 1938.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 2018.

Lim, H., Kim, M.-S., and Xiong, J. {CNAS}: Channel-level neural architecture search, 2020. URL https://openreview.net/forum?id=rklfIeSFwS.

Liu, C., Chen, L.-C., Schroff, F., Adam, H., Hua, W., Yuille, A., and Fei-Fei, L. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, 2019a.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations(ICLR)*, 2019b.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in Neural Information Processing Systems 31*. 2018.

Luo, R., Qin, T., and Chen, E. Balanced one-shot neural architecture optimization. abs/1909.10815, 2020. URL http://arxiv.org/abs/1909.10815.

Ma, N., Zhang, X., Zheng, H., and Sun, J. Shufflenet V2: practical guidelines for efficient CNN architecture design. *CoRR*, abs/1807.11164, 2018. URL http://arxiv.org/abs/1807.11164.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The Penn Treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

Mei, J., Li, Y., Lian, X., Jin, X., Yang, L., Yuille, A., and Yang, J. Atomnas: Fine-grained end-to-end neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BylQSxHFwr.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018a. URL http://arxiv.org/abs/1802.05957.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018b.

Nayman, N., Noy, A., Ridnik, T., Friedman, I., Jin, R., and Zelnik, L. Xnas: Neural architecture search with expert advice. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 1977–1987. Curran Associates, Inc., 2019.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning(ICML)*, 2018.

Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., and Dollar, P. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Association for the Advancement of Artificial Intelligence(AAAI)*, 2019.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. URL http://arxiv.org/abs/1905.11946.

Tan, M., Chen, B., Pang, R., Vasudevan, V., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Conference on Computer Vision and Pattern Recognition(CVPR)*, 2019.

Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., Vajda, P., and Gonzalez, J. E. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Wang, L., Xie, S., Li, T., Fonseca, R., and Tian, Y. Sample-efficient neural architecture search by learning action space. *CoRR*, abs/1906.06832, 2019a. URL http://arxiv.org/abs/1906.06832.

Wang, L., Zhao, Y., Jinnai, Y., Tian, Y., and Fonseca, R. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *CoRR*, abs/1903.11059, 2019b. URL http://arxiv.org/abs/1903.11059.

Wang, W., Sun, Y., and Halgamuge, S. Improving MMD-GAN training with repulsive loss function. In *International Conference on Learning Representations*, 2019c.

Wang, X., Xue, C., Yan, J., Yang, X., Hu, Y., and Sun, K. Mergenas: Merge operations into one for differentiable architecture search. In Bessiere, C. (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pp. 3065–3072. ijcai.org, 2020. doi: 10.24963/ijcai.2020/424. URL https://doi.org/10.24963/ijcai.2020/424.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. {PC}-{darts}: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJlS634tPr.

Yiming Hu, Yuding Liang, Z. G. R. W. X. Z. Y. W. . Q. G. J. S. Angle-based search space shrinking for neural architecture search. 2020.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. NAS-bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Yu, J. and Huang, T. S. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *CoRR*, abs/1903.11728, 2019. URL http://arxiv.org/abs/1903.11728.

Yu, K., Ranftl, R., and Salzmann, M. How to train your super-net: An analysis of training heuristics in weight-sharing nas. 2019a.

Yu, K., Sciuto, C., Jaggi, M., Musat, C., and Salzmann, M. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019b.

Yu, K., Sciuto, C., Jaggi, M., Musat, C., and Salzmann, M. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020.

Yukang Chen, Tong Yang, X. Z. G. M. X. X. J. S. Detnas: Backbone search for object detection, 2019.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=H1gDNyrKDS.

Zela, A., Siems, J., and Hutter, F. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=SJx9ngStPH.

Zhang, M., Li, H., Pan, S., Liu, T., and Su, S. One-shot neural architecture search via novelty driven sampling. In Bessiere, C. (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 3188–3194. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

Zhiyuan Li, S. A. An exponential learning rate schedule for deep learning. 2020.

Zhou, H., Yang, M., Wang, J., and Pan, W. BayesNAS: A Bayesian approach for neural architecture search. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7603–7613. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/zhou19e.html.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. Learning transferable architectures for scalable image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

A. Additional Notations

We use two additional notations for pseudocode description: *(i)* $\mathcal{S}_{id}$ denotes a set of sub-supernets that is split by id numbers of edges. *(ii)* $\mathcal{S}_{id}^{n}$ denotes the n^{th} sub-supernet in $\mathcal{S}_{id}$.

B. End-to-end Pipeline Pseudocode

Below we list the pseudocode for the end-to-end split and training pipeline in Algorithm 1, the pseudocode for random split the one-shot model into sub-supernets in Algorithm 2, and the pseudocode for training (sub-)supernets in Algorithm 3.

Algorithm 1 (Sub)-supernets split and training

$\mathcal{S}_0 = \{\mathcal{S}\}$
define global $T \leftarrow TIME\_BUDGET$
Train($\mathcal{S}$, NONE)
$\mathcal{S}_0 \leftarrow \mathcal{S}$
$id \leftarrow 0$
while total time $< T$ **do**
  $j \leftarrow random(0, \#N)$
  $i \leftarrow random(0, j)$
  $\mathcal{S}_{id+1} \leftarrow$ RandomSplit($\mathcal{S}_{id}, E_{ij}$)
  for $n = 1 \rightarrow sizeof(\mathcal{S}_{id+1})$ **do**
    Train($\mathcal{S}_{id+1}^{n}, \mathcal{S}_{id}^{'}$)
  end for
  $id \leftarrow id + 1$
end while

Algorithm 2 RandomSplit($\mathcal{S}_{id}, E_{ij}$)

$\mathcal{S}_{new} \leftarrow$ split $\mathcal{S}_{id}$ to m sub-supernets given m operations
return $\mathcal{S}_{new}$

Algorithm 3 Train(s, $parent$)

if $parent$ IS NOT NONE **then**
  $W_s \leftarrow W_{parent}$
end if
While s NOT CONVERGE **do**
  forward(s)
  backward(s)
end While

C. Experiment Setup for Section 2

Each architecture was trained for 150 epochs with batch size of 128. The initial channel is 16. We used the SGD optimizer with an initial learning rate of 0.025, followed by a cosine learning rate schedule through the training. We set the momentum rate to 0.9 and a weight decay of 3×10^{-4}. The training setup of supernet and sub-supernets is consistent with architecture candidates. These experiments ran on 50 P100 GPUs.

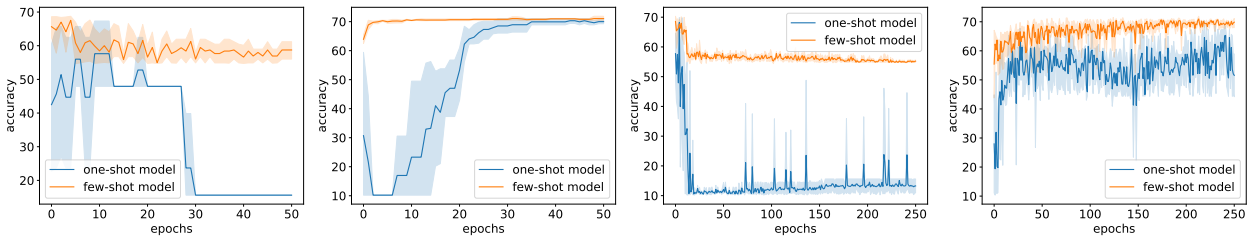
D. Experiment Setup for Section 4

(Sub-)supernet Training Setup for NASBENCH-201. Each architecture was trained for 200 epochs with 256 batch size. The initial channel is 16. We used the SGD optimizer with an initial learning rate of 0.1, followed by a cosine learning rate schedule through the training. The momentum rate was set to 0.9. We used a weight decay of 5×10^{-4} and a norm gradient clipped at 5. Cutout technique was not used in the training. The supernet training setup is consistent with architecture candidates. For supernet training, we changed the initial learning rate to 0.025 and total epochs to 300. The batch size is 128 and the weight decay was set to 1×10^{-4}. Each sub-supernet approximately took 40-50 epochs to converge after transfer learning. For each NAS algorithm, we used the same setup as described in the NASBENCH-201 (Dong & Yang, 2020). We used 6 P100 GPUs to train the supernet and 5 sub-supernets.

Search Setup for DARTS on CIFAR-10. We used the same search space and training setup as described in the original DARTS paper (Liu et al., 2019b). Specifically, the available operations in the search space include 3 x 3 and 5 x 5 separable convolutions, 3 x 3 and 5 x 5 dilated separable convolutions, 3 x 3 max pooling, 3 x 3 average pooling, identity, and zero. We trained 8 cells using DARTS for 50 epochs, with batch size 64 (for both the training and validation sets). The initial number of channels was set to 16. Each sub-supernet took 5-20 epochs to be converge. We used the momentum SGD optimizer with an initial learning rate of 0.025, followed by a cosine learning rate schedule through the training. We used a momentum rate of 0.9 and a weight decay of 3×10^{-4}. This experiment ran on 10 P100 GPUs for training both supernet and sub-supernets.

We trained the network for 1500 epochs using a batch size of 128 and use a momentum SGD optimizer with an initial learning rate of 0.025, followed by a cosine learning rate schedule through the training. We use weight decay as the regularization.

Search Setup for DARTs on PTB. The search space and the training setup of (sub)-supet-nets are identical to DARTS (Liu et al., 2019b). Concretely, both the embedding and the hidden sizes were set to 300. We used 6 P100 GPUs to train both the supernet and 5 (sub)-supet-nets. Each (sub-)supernet was trained for 50 epochs using SGD without momentum, with a learning rate of 20. The batch size was set to 256 and the weight decay was set to 3×5^{-7}. We



(a) DARTS (b) PCDARTS (c) ENAS (d) SETN

Figure 8. Anytime accuracy comparison of state-of-the-art gradient-based algorithms on *few-shot NAS* for CIFAR-100.

applied a variational dropout of 0.2 to word embeddings, 0.75 to the cell input, and 0.25 to all the hidden nodes. We also applied a dropout rate of 0.75 to the output layer.

Search Setup for ImageNet (Gong et al., 2019). For proxylessNas, we exactly keep the same search pipeline with original paper (Cai et al., 2019), We randomly sample 50,000 images from the training set as a validation set during the architecture search. For our few-shot NAS, we split 3 sub-supernets. The (sub)supernets parameters are updated using the Adam optimizer with an initial learning rate of 0.001. The (sub)supernets is trained on the remaining training images with batch size 256. For once for all NAS, the search setup is also consistent with original OFA (Cai et al., 2020). In specific, we use the same architecture space as MobileNetV3 (Howard et al., 2019), for supernet training, we use the standard SGD optimizer with Nesterov momentum 0.9 and weight decay is set to 3×10^{-5}. The initial learning rate is 2.6, and we use the cosine schedule for learning rate decay. We split 5 sub-supernets. The (sub)supernets are trained for 180 epochs with batch size 2048 on 64 32G V100 GPUs.

Search Setup for AutoGAN (Gong et al., 2019). Our search and training settings were identical to AutoGAN (Gong et al., 2019), which followed spectral normalization GAN (Miyato et al., 2018a) when training the (sub-)-supernets. We split the supernet (shared GAN in (Gong et al., 2019)) to 3 sub-supernets. The learning rate of both generator and discriminator were set to $2e^{-4}$. We used the hinge loss and an Adam optimizer. The batch size of discriminator was 64 and the generator was 128. The initial learning rate was set to $3.5e^{-4}$. The AutoGAN searched for 90 iterations for one supernet. For each iteration, the shared GAN (supernet) was trained for 15 epochs, and the controller was trained for 30 steps. After the shared GAN (supernet) was trained, we transferred the weight to each sub-supernets and trained them for 12 epochs. We trained the controller with 30 steps. The discovered architectures were trained for 50,000 generator iterations. We used 4 P100 GPUs in this experiment.

E. Evaluation of Gradient-based Algorithms on CIFAR-100

Figure 8 shows the anytime accuracy of running state-of-the-art gradient-based algorithms on *few-shot NAS*. We observe similar trend as shown for CIFAR-10 in Figure 5.

Consistent with fig.5, the anytime CIFAR-100 test accuracy is shown in fig.8. Based on fig.8, The curve of each algorithms keep a very similar trend with in CIFAR-10. Therefore, we are able to get a same results with sec.4.1.1. The few-shot NAS gradient-based algorithms can have more efficient search, both in terms of found architecture and number of search epochs.

F. One-shot NAS v.s. Few-shot NAS by Robust DARTS (Zela et al., 2020a)

Table 7. Few-shot Robust DARTS vs. One-shot Robust DARTS over 4 Search Space

Method	Space	Top 1 Acc(%)
one-shot	s1	96.49
few-shot	s1	**96.81**
one-shot	s2	96.22
few-shot	s2	**96.55**
one-shot	s3	97.19
few-shot	s3	**97.28**
one-shot	s4	95.60
few-shot	s4	**96.30**

We use our few-shot NAS with Robust DARTS searching architectures over 4 different search spaces defined by original paper (Zela et al., 2020a). For table 7, we can see that the accuracy of architectures searched by our few-shot are significantly better than one-shot over all 4 search spaces. Our training setup is strictly same with its original paper.