Few-shot Neural Architecture Search

Yiyang Zhao[1]* **Linnan Wang**[2]* **Yuandong Tian**[3] **Rodrigo Fonseca**[2] **Tian Guo**[1]
[1]Worcester Polytechnic Institute [2]Brown University [3]Facebook AI Research

Abstract

To improve the search efficiency for Neural Architecture Search (NAS), One-shot NAS proposes to train a single super-net to approximate the performance of proposal architectures during search via weight-sharing. While this greatly reduces the computation cost, due to approximation error, the performance prediction by a single super-net is less accurate than training each proposal architecture from scratch, leading to search inefficiency. In this work, we propose *few-shot NAS* that explores the choice of using multiple super-nets: each super-net is pre-trained to be in charge of a sub-region of the search space. This reduces the prediction error of each super-net. Moreover, training these super-nets can be done jointly via sequential fine-tuning. A natural choice of sub-region is to follow the splitting of search space in NAS. We empirically evaluate our approach on three different tasks in NAS-Bench-201. Extensive results have demonstrated that few-shot NAS, using only 5 super-nets, significantly improves performance of many search methods with slight increase of search time. The architectures found by DARTs and ENAS with few-shot models achieved 88.53% and 86.50% test accuracy on CIFAR-10 in NAS-Bench-201, significantly outperformed their one-shot counterparts (with 54.30% and 54.30% test accuracy). Moreover, on AUTOGAN [10] and DARTS [18], few-shot NAS also outperforms previously state-of-the-art models [10, 18].

1 Introduction

Neural Architecture Search (NAS) has attracted lots of interests over the past few years [2, 27, 38]. Using NAS, many deep learning tasks [10, 17, 28, 29, 36] improve their performance without human tuning. Standard NAS requires tremendous amount of computational costs (e.g., thousands of GPU hours) in order to find a superior neural architecture [2, 25, 38], most of which is due to evaluating new architecture proposals by training them from scratch. To reduce the cost, one-shot NAS [18, 24, 31] proposes to train a single super-net that represents all possible architectures in the search space. With super-net, individual architecture can be evaluated by inheriting corresponding weights from the super-net and performing simple feed-forwarding(shown in Fig.2(a)), reducing the search cost to just a few days (hours).

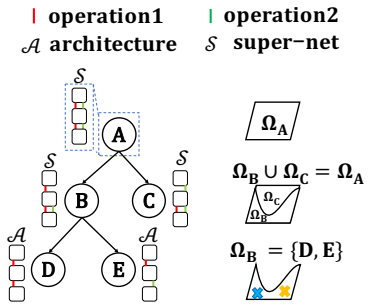


Figure 1: The connection between one-shot NAS and standard NAS. Ω denotes an architecute space.

On the other hand, one-shot NAS suffers from degraded search performance. Existing works [3, 8, 19, 20, 35] shows that there is a non-trivial performance gap between the architectures found by one-shot NAS and standard NAS. For example, the architectures found with super-net had 2.93% test error compared to 2.48% found with standard NAS on CIFAR10 [20]. The main reason is that, the

*Yiyang Zhao and Linnan Wang contributed equally

Preprint. Under review.

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
$\mathcal{S}$	super-net	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-super-net	$f(\mathcal{A})$	the evaluation of $\mathcal{A}$	$f(\mathcal{S}_{\mathcal{A}})$	the evaluation of $\mathcal{A}$ by super-net

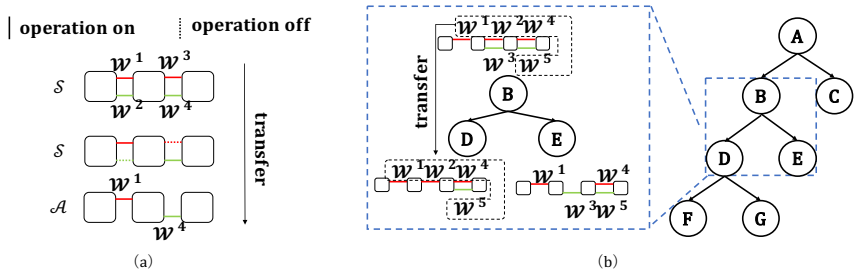


Figure 2: (a) An architecture A is evaluated by using weights inherited from the super-net. (b) Transfer learning is used to speedup sub-super-nets training.

performance predicted by the super-net has low correlation with the true performance. As an example, Section 2 shows that inaccurate performance prediction by super-net biases the search towards a wrong direction and hurt both the efficiency and the final results.

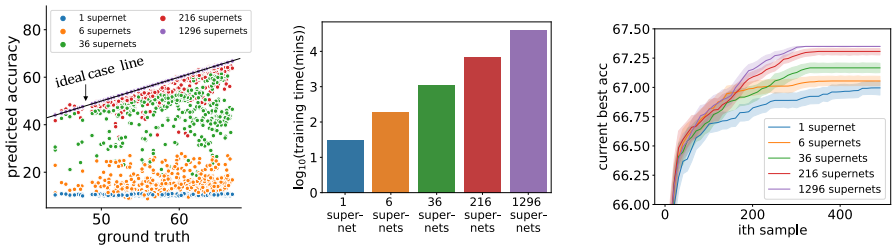
In this work, we propose *few-shot NAS* that uses multiple super-nets in architecture search. Instead of having one super-net covering the entire search space, which may be beyond its capacity, using multiple super-nets effectively addresses this issue by having each super-net modeling one part of the search space. To partition the search space, we follow the first few actions of the search tree in NAS, and place one *sub-super-net* on each of the search node in the first few layers of the tree. Figure 1 shows an exemplar search tree: the root represents the entire search space (i.e., all possible network operations), and leaves represent actual architectures in the space Ω. Moving down along one edge removes some network operations (while keeping others). During super-net pre-training, we restrict each sub-super-net to have the specific architecture defined by each search node. Moreover, these models can be trained efficiently by using a cascade of transfer learning: first the root sub-super-net 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 sub-super-nets, each of which is responsible for a region of the search space. Their prediction is more accurate in their own sub-trees.

Section 3 describes the detailed strategies for building few-shot NAS by splitting the super-net. With only 5 sub-super-nets, we show that our *few-shot NAS* greatly improved many existing NAS algorithms on NASBENCH-201 [8] dataset and several popular deep learning tasks in Section 4. For example, using DARTS with our *few-shot NAS* outperformed searching directly with one-shot NAS by 0.39 lower error in CIFAR-10. Moreover, on AUTOGAN [10] and DARTS [18], *few-shot NAS* also improved state-of-the-art models [10, 18], from 12.42 to 10.73 in FID score. An anonymized implementation of *few-shot NAS* can be found at https://drive.google.com/drive/folders/13MN_p6rdBSlpogLXTVDD8uWbxMwmqrHb?usp=sharing

2 A Motivating Example

In this section, we present a proof-of-concept investigation of a small architecture space to demonstrate the feasibility of *few-shot NAS*. In short, our empirical investigation demonstrates the feasibility of using sub-super-nets, which are split by super-net in a simple search space; and the need to balance the trade-offs between training time and accuracy prediction performance. In the next sections, we describe our *few-shot NAS* design targeting a generic search space and evaluate its performance with an open-source NAS benchmark NASBENCH-201 and popular application domains.

Investigation of *few-shot NAS* in a CNN Search Space. We designed a search space for sequential Convolutional Neural Network (CNN) architectures with 1296 architectures. We considered three



(a) True vs predicted accuracy (b) Super-net training time (c) Search performance of REA

Figure 3: Search performance comparison for a simple architecture space. We observe that deeper split leads to (a) better pairwise relationship between predicted and true accuracy, but (b) with substantial time increase. (c) A deeper split also leads to a more efficient search algorithm performance. The 1, 6, 36, 216, 1296 super-nets correspond to the 1, 2, 3, 4, 5 depth of the search tree in the Fig.1, respectively.

types of operations $\{conv1, conv3, maxpool3\}$ with five nodes. We fixed the number of edges to be four and built the super-net by connecting the edges with all possible operations.

We then split the super-net by the first edge into 6 sub-super-nets, where each sub-super-net keeps one or two operations in the first edge. Then, we recursively split the edges until the last and trained all resulting sub-super-nets. Specifically, there are 1, 6, 36, 216, 1296 sub-super-nets generated by splitting 0, 1, 2, 3, 4 edges. We trained the super-net and all sub-super-nets and used them to estimate the architecture performance in the search space.

Key Motivation. we observe that by using multiple super-nets in predicting an architecture's performance, the correlation between the true accuracy and the predicted accuracy has significantly improved. This leads to notable performance improvement in the search (Figure 3(c)). In Figure 3(a), each point represents the true accuracy (x-axis) and the predicted accuracy from super-net (y-axis). The ideal case is represented by a line that the predicted accuracy exactly matches with the true accuracy.

The accuracy distribution of using more super-nets clearly moves toward the idea case, therefore the corresponding search performance are subsequently improved. However, Figure 3(b) highlights the cost of training is also exponentially increasing with the number of super-nets. The training time of using more super-nets will eventually converge to standard NAS, which is not desirable computation-wise.

This motivates us to look into a trade-off point that improves the prediction quality with multiple super-nets at a reasonable training cost.

3 Our Proposed Solution: *Few-shot NAS*

In designing *few-shot NAS*, we answer the following three key questions: *(i)* how to divide the search space represented by the one-shot model to sub-super-nets (Section 3.1)?; *(ii)* how to reduce the training time of multiple sub-super-nets (Section 3.2)?; and *(ii)* how to choose the number of sub-super-nets given a search time budget (Section 3.3)? We also describe how to integrate *few-shot NAS* with existing NAS algorithms in Section 3.4 and Section 3.5.

3.1 Design of Progressive Split

Design Intuition. 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-super-net $\mathcal{S}^{\Omega^k}$ 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-super-net at the shallowest tree depth.

Designing a learning-based approach can be difficult as it can be very costly to obtain the training data in the form of $((\mathcal{A}, \mathcal{Q}), f(\mathcal{S}_{\mathcal{A}}^{\Omega^q}))$ where Ω^q is a sub-region created at depth $\mathcal{Q}$.

Definition of a Generic NAS Space. Before we describe our progressive split strategy that recursively split the architecture space at the certain tree depth, 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 its predecessor node.

Split Procedure Overview. We use the architecture space described in Figure 4 to illustrate how the progressive split strategy works. At the high level, we will go through the node in ascending order and for each node, we will generate one sub-super-net for each operation in all incoming edges. For example, the super-net $\mathcal{S}$ will first be split into m sub-super-nets at node N_1, given that N_1 only has one incoming edge, i.e., E_{01} with m operations, from the first node N_0. Then, we will split at N_2 (also corresponding to tree depth of 2) through its incoming edges E_{02} and E_{12}, and generate a total of $m^3 + 2m^2$ sub-super-nets. This is because we will split all m sub-super-nets generated at N_1 first by E_{02}, resulting in m^2 sub-super-nets that will be further split by E_{12}. We will also split each sub-super-net from N_1 by E_{12}, generating an additional m^2 sub-super-nets. We will repeat the same procedure until we reach the last node. Without loss of generality, we will generate $\prod_{a=1}^{i}((m+1)^a - 1)$ sub-super-nets after splitting at N_i.

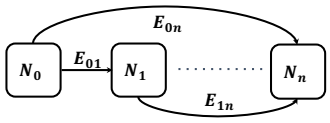


Figure 4: A generic architecture space.

3.2 Transfer Learning

Using the progressive split strategy, the number of sub-super-nets grows exponentially with the number of nodes n. Directly training all the resulting sub-super-nets can be computational 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-super-nets.

Similar to how an architecture candidate $\mathcal{A}$ inherits weights $\mathcal{W}_{\mathcal{A}}$ from the super-net weights $\mathcal{W}_{\mathcal{S}}$, we allow a sub-super-net $\mathcal{S}^{\Omega'}$ to inherit weights from its parent sub-super-net. For example, in Figure 2(b), the sub-super-net at node D can inherit its weight (w^1, w^2, w^4, w^5) from its parent at node B. Note w^3 is not passed down to D. By using transfer learning, each sub-super-net only need very small epochs to converge compared to training from scratch.

3.3 Super-net Training

The training setup for the super-net is the same as the architecture candidates.

In training procedure, we sum all the operations [18] between two nodes and concatenate all nodes together to the output.

Instead of directly training the entire super-net, we implemented a path-wise strategy [11] for training the super-net. We chose this stochastic training approach as it does not require hyper-parameter tuning and can improve the training efficiency of super-net. Specifically, in each iteration, we uniformly sample an architecture candidate from the architecture space that corresponds to a single path in the super-net. We train this architecture by zeroing out other edges and operations that do not belong to the activated path. In this way, we ensure that each architecture candidate has the equal probability to be trained.

We use this training strategy for both super-net and sub-super-nets, and pre-define a training time budget T. If the total training time of super-net and all currently trained sub-super-nets exceeds T, we would stop the progressive split to avoid training more sub-super-nets.

3.4 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 [18]. In other words, gradient-based algorithms are designed for and used with one-shot NAS. Gradient-based algorithms rely on the loss in each step to guide the search. Specifically, the weights of super-net is updated by training the super-net on the training set. Then, the architecture distribution parameters are optimized by training the super-net on the validation set. This procedure is repeated until both the weight and architecture distribution parameters are converged.

To use gradient-based algorithms with our *few-shot NAS*,

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

Compared to the one super-net in search space Ω, the loss of sub-super-net is more accurate in terms of their architecture candidates in the space. Therefore, in theory, the architecture found by *few-shot NAS* would be better than one-shot NAS.

3.5 Integration with Search-based Algorithms

Overview. Search-based algorithms can work with both one-shot and standard 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 standard NAS, or evaluating by a pre-trained super-net, in the case of one-shot NAS. For standard 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-super-nets by using progressive split and transfer learning, similar to what was described in Section 3.4. These converged sub-super-nets will be used as the basis to evaluate the performance of sampled architectures. For example, if a sampled architecture $\mathcal{A}$ falls into sub-super-net $\mathcal{S}^{\Omega^{'}}$, we will evaluate its performance $f(\mathcal{S}_{\mathcal{A}}^{\Omega^{'}})$ 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. Finally, we will select the final architecture based on test error.

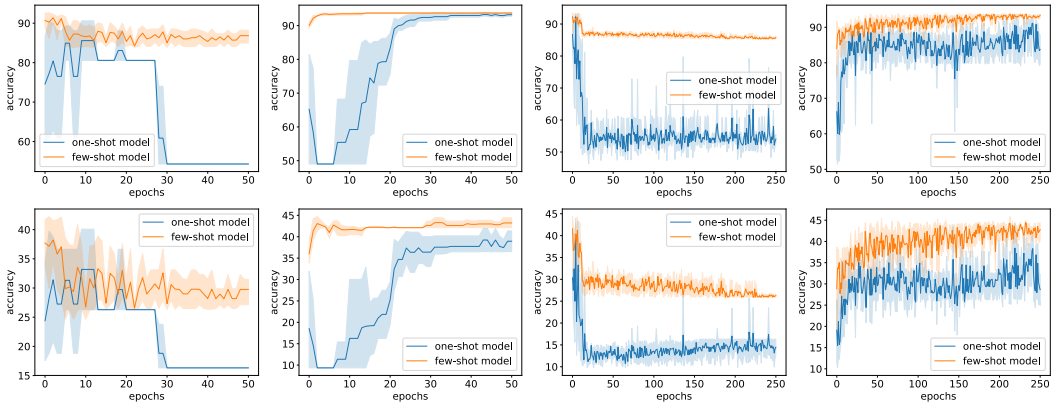
4 Experiments

To evaluate the performance of *few-shot NAS* in reducing the approximation error associated with super-net 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 different metrics to evaluate search efficiency of DARTS, PCDARTS, ENAS, SETN, REA, REINFORCE, HB, BOHB, SMAC, and TPE [4, 7, 9, 14, 16, 18, 24, 25, 31, 38] by one-shot/few-shot

Table 2: Test accuracy and search time comparison of state-of-the-art gradient-based algorithms on NASBENCH-201. We ran each algorithm on both one-shot and *few-shot NAS*, each for five times.

Algorithm	Method	#super-net	Time(hours)	CIFAR-10(%)	CIFAR-100(%)	ImageNet-16-120(%)
DARTS	Few-shot (Ours)	5	34.78	**88.53 ± 1.21**	**61.93 ± 2.94**	**33.10 ± 3.24**
	One-shot	1	7.10	54.30 ± 0.00	15.61 ± 0.00	16.32 ± 0.00
PCDARTS	Few-shot (Ours)	5	35.02	**93.78 ± 0.73**	**70.85 ± 1.42**	**43.42 ± 4.00**
	One-shot	1	7.43	93.23 ± 0.31	69.8 ± 0.99	36.60 ± 5.83
ENAS	Few-shot (Ours)	5	21.45	**85.60 ± 1.14**	**55.82 ± 1.42**	**26.89 ± 0.95**
	One-shot	1	4.56	54.30 ± 0.41	13.49 ± 2.47	15.14 ± 2.40
SETN	Few-shot (Ours)	5	35.67	**93.78 ± 0.11**	**71.42 ± 1.10**	**44.60 ± 0.72**
	One-shot	1	7.30	87.64 ± 0.39	58.87 ± 0.14	32.37 ± 0.04



(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*. Due to space limits, we only show the results for CIFAR-10 (1st row) and ImageNet-16-120 (2nd row). Shaded area represents the highest and lowest values based on five runs.

models on NASBENCH-201. 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, AutoGAN [10], and Penn Treebank [21].

4.1 Evaluations on NASBENCH-201

We use NASBENCH-201, a public architecture dataset, which provides a unified benchmark for up-to-date NAS algorithms [8]. 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 [33] does not provide all possible architectures in its 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 super-net corresponds to NASBENCH-201 has five nodes and five operations. Based on the progressive split method described in Section 3.1, we split the first edge in node N_1 and obtain five sub-super-nets. For this experiment, we did not train sub-super-nets 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. 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 super-net training and validation time.

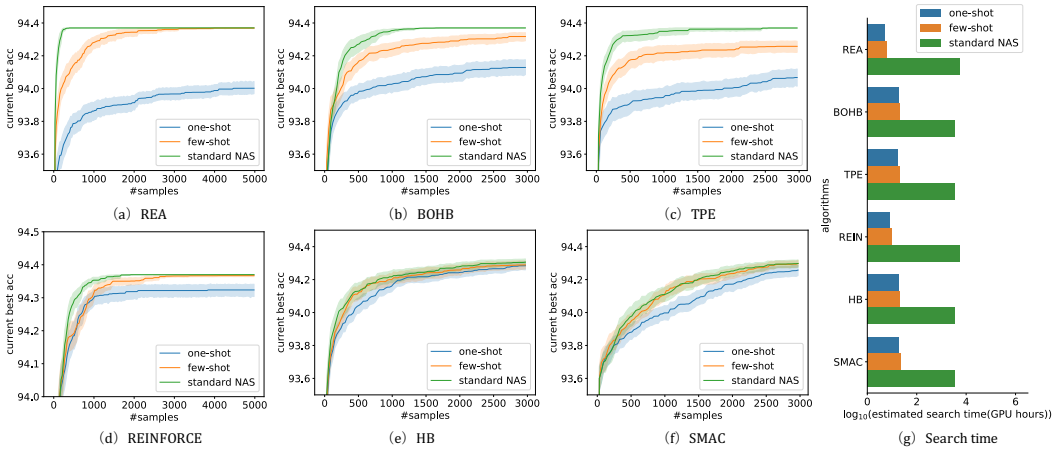


Figure 6: Current best accuracy and search time comparison of popular search-based algorithms. All algorithms were ran for 50 times for one-shot, few-shot and standard NAS, respectively.

Results Analysis. Table 2 compares the test accuracy and search time on three popular image datasets. We observe that all four gradient-based algorithms found architectures with higher accuracy (shown in bold) when using *few-shot NAS*, compared to using one-shot NAS. For example, the architecture found by using *few-shot NAS* on CIFAR-10 has an up to 33.02% higher accuracy (when using DARTS). The superior performance of *few-shot NAS* can be better understood by looking at Figure 5 which we report 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 [8]. The main reason caused by this is one-shot NAS would fall into a sub-optimal region due to inaccurate performance prediction by super-net. In contrast, *few-shot NAS* maintained a high quality of searched models since multiple super-nets 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 super-nets, 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 number of search epochs.

4.1.2 Search-based Algorithms

Methodology. As described in Section 3.3, we define search time budget of *few-shot NAS* to be less than twice as that of one-shot NAS. Therefore, we only split the super-net by first edge in the first node N_1 to five sub-super-nets. For this experiment, we used transfer learning described in Section 3.2 for training sub-super-nets. We ran each search-based algorithms for 50 times. We chose six different search-based algorithms including REA, REINFORCE, BOHB , HB, SMAC, and TPE [4, 9, 14, 16, 25, 38]. We evaluate the effectiveness of *few-shot NAS* by following the training procedure described in Section 3.5. 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 quantifying 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).

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 order of magnitude GPU hours when using 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.

4.2 Deep Learning Applications

CIFAR-10 in Practice. We chose two state-of-the-art NAS algorithms, a gradient-based algorithm DARTS and a search-based algorithm regularized evolution (REA) [18, 25], for evaluating the effectiveness of *few-shot NAS*. We used the same search space based on the original DARTS and REA papers. Table 3 compares the search performance. We see that using DARTS with our *few-shot NAS* outperformed searching directly with one-shot NAS by 0.39 lower error. Further, the architecture found with *few-shot NAS* also matches the state-of-the-art results on CIFAR-10 (first two rows in Table 3). Additionally, *few-shot NAS* only incurred a 25% search time increase. Similarly, *few-shot NAS* also improved the search efficiency of REA upon one-shot NAS, finding an architecture with 0.18 lower error with only 16.7% more search time. In short, *few-shot NAS* improved the efficiency of existing state-of-the-art search algorithms.

Comparison to AUTOGAN [10]. 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 and used CIFAR-10 without any data augmentation for training and a RNN controller for guiding the architecture search. We used Inception score (IS)(higher is better) and Frchet Inception Distance (FID) [26](lower is better) to evaluate the performance of GAN. Table 4 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 3: NASNet search results with our *few-shot NAS* vs. state-of-the-art results on CIFAR-10.

Method	#Params	Error	GPU days
NASNet-A+cutout[38]	3.3M	2.65	2000
AmoebaNet-B+cutout[25]	2.8M	2.50±0.05	3150
DARTS+cutout[18]	3.3M	2.76±0.09	4
DARTS(few-shot)+cutout	3.8M	**2.37±0.07**	5
REA(one-shot)+cutout	3.5M	2.70±0.03	3
REA(few-shot)+cutout	3.7M	**2.52±0.05**	3.5

Table 4: AUTOGAN vs. using *few-shot NAS*.

Method	Inception Score	FID Score
ProbGAN[12]	7.75±.14	24.60
SN-GAN[22]	8.22±.05	21.70±.01
MGAN[13]	8.33±.12	26.7
Improving MMD GAN[30]	8.29	16.21
AutoGAN-top1[10]	8.55±.10	12.42
AutoGAN-top2	8.42±.06	13.67
AutoGAN-top3	8.41±.12	13.87
AutoGAN(few-shot)-top1(ours)	8.60±.10	**10.73±.10**
AutoGAN(few-shot)-top2(ours)	**8.63±.09**	10.89±.20
AutoGAN(few-shot)-top3(ours)	8.52±.08	12.20

PENN TREEBANK in Practice [21]. 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 setting 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 super-net was first proposed as a way to reduce the computational cost of NAS [24]. Centering around super-net, a number of NAS algorithms including gradient-based [7, 18, 31] and search-based [3, 6, 11] were proposed. The search efficiency of these algorithms are dependent on the ability of super-net to approximate architecture performance.

To improve the super-net approximation accuracy, Bender et al. [3] proposed a path dropout strategy that randomly drops out weights of the super-net 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. [11] proposed a single-path one-shot training by only activating the weights from one randomly picked architecture in forward and backward propagation. This ensures that the weights of each architecture are updated without co-adaptation. Additionally, Yu et al. [34] found that training setup greatly impact super-net performance and identified useful parameters and hyper-parameters. Lastly, an angle-based approach [1, 5, 37] was proposed to improve the super-net approximation accuracy for individual architecture [32] and was shown to improve the architecture rank correlation. However, our few-shot models achieved better rank correlation than this angle-based approach, with a higher Kendall's Tau of 0.6242 compared to 0.5748 [15].

Our work focuses on reducing the super-net approximation error by dividing the architecture space to a few sub-super-nets and using the more effective one for architecture approximation. As such, our work is complementary and can be integrated to 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 standard NAS and can be integrated with both gradient-based and search-based algorithms. Our extensive evaluations on a recent NAS benchmark NASBENCH-201 and deep learning applications demonstrated that *few-shot NAS* significantly improved search performance of all popular one-shot methods with negligible search time increase. Furthermore, the final results from *few-shot NAS* have also outperformed previously published results by DARTs and AUTO-GAN.

7 Broader Impact

Our work connects standard NAS and one-shot NAS and was demonstrated to balance the trade-off between search time and the performance of found architectures. Specifically, our work leverages a few sub-super-nets to improve the approximation accuracy and transfer learning to shorten the sup-super-nets training time. The proposed *few-shot NAS* was empirically evaluated on an existing NAS benchmark as well as on various deep learning applications; we observe that *few-shot NAS* found architectures with higher accuracy compared to the state-of-the-art, with negligible search time increase. Moreover, our proposed *few-shot NAS* can be easily integrated with existing one-shot techniques, further increase the utility our work. In short, our work identifies a promising direction in NAS that might encourage future research efforts.

References

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

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

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

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

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

[6] X. Chu, B. Zhang, R. Xu, and J. Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *CoRR*, abs/1907.01845, 2019.

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

[8] X. Dong and Y. Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

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

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

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

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

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

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

[15] M. G. Kendall. A new measure of rank correlation. 1938.

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

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

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

[19] R. Luo, T. Qin, and E. Chen. Balanced one-shot neural architecture optimization. abs/1909.10815, 2020.

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

[21] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 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.

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

[23] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.

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

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

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

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

[28] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian. Sample-efficient neural architecture search by learning action space. *CoRR*, abs/1906.06832, 2019.

[29] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019.

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

[31] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong. {PC}-{darts}: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020.

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

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

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

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

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

[37] S. A. Zhiyuan Li. An exponential learning rate schedule for deep learning. 2020.

[38] B. Zoph, V. Vasudevan, J. Shlens, and Q. Le. 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)* $\mathcal{S}_i$ denotes a set of sub-super-nets that is split by i^{th} node. *(ii)* $\mathcal{S}_i^j$ denotes the j^{th} sub-super-net in $\mathcal{S}_i$.

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 progressive split the one-shot model into sub-super-nets in Algorithm 2, and the pseudocode for training (sub-)super-nets in Algorithm 3.

Algorithm 1 (Sub)-super-nets split and training

$\mathcal{S}_0 = \{\mathcal{S}\}$
define global $T \leftarrow TIME\_BUDGET$
Train($\mathcal{S}$, NONE)
for $j = 1 \rightarrow \#N$ **do**
  $\mathcal{S}_j \leftarrow$ ProgressiveSplit($\mathcal{S}_{j-1}, N_j$)
  if total time $> T$ **then**
    BREAK
  end if
  for $i = 1 \rightarrow sizeof(\mathcal{S}_j)$ **do**
    Train($\mathcal{S}_j^i$, $\mathcal{S}_{j-1}^{'}$)
  end for
end for

Algorithm 2 ProgressiveSplit($\mathcal{S}_{j-1}, N_j$)

$E_{set} \leftarrow$ Get_All_Possible_Edge_Comb(j)
$\mathcal{S}_j$ = { }
for E_{comb} in E_{set} **do**
  $\mathcal{S}_{new}$ = { }
  for E_{ij} in E_{comb} **do**
    if $\mathcal{S}_{new}$ is $Empty$ **then**
      for $q = 1 \rightarrow sizeof(\mathcal{S}_{j-1})$ **do**
        $\mathcal{S}_{new} = \mathcal{S}_{new} \cup$ Split_Edge($\mathcal{S}_{j-1}^q$, E_{ij})
      end for
    else
      $\mathcal{S}_{new}^{'}$ = { }
      for s in $\mathcal{S}_{new}$ **do**
        $\mathcal{S}_{new}^{'} = \mathcal{S}_{new}^{'} \cup$ Split_Edge(s, E_{ij})
      end for
      $\mathcal{S}_{new} = \mathcal{S}_{new} \cup \mathcal{S}_{new}^{'}$
    end if
  end for
  $\mathcal{S}_j$ = $\mathcal{S}_j \cup \mathcal{S}_{new}$
end for
return $\mathcal{S}_j$

function Split_Edge($\mathcal{S}_p^q$, E_{ij})
    $\mathcal{S}_{new} \leftarrow$ split $\mathcal{S}_p^q$ to m sub-super-nets given m operations
    return $\mathcal{S}_{new}$

function Get_All_Possible_Edge_Comb(j)
    return all combinations of [E_{0j}, E_{1j}, ..., $E_{(j-1)j}$]

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 super-net and sub-super-nets is consistent with architecture candidates. These experiments ran on 50 P100 GPUs.

D Experiment Setup for Section 4

(Sub-)super-net 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 super-net training setup is consistent with architecture candidates. For super-net 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-super-net 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 [8]. We used 6 P100 GPUs to train the super-net and 5 sub-super-nets.

Search Setup for DARTS on CIFAR-10. We used the same search space and training setup as described in the original DARTS paper [18]. 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-super-net 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 super-net and sub-super-nets.

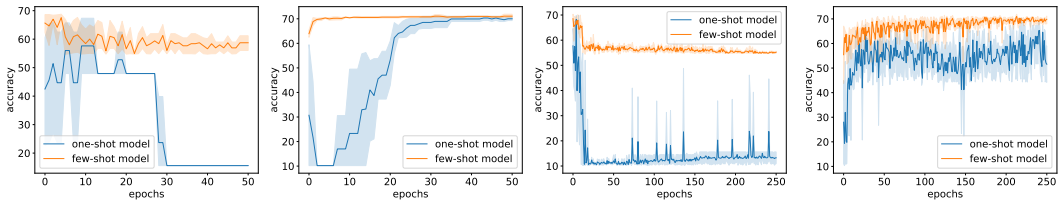
Search Setup for DARTs on PTB. The search space and the training setup of (sub)-supet-nets are identical to DARTS [18]. Concretely, both the embedding and the hidden sizes were set to 300. We used 6 P100 GPUs to train both the super-net and 5 (sub)-supet-nets. Each (sub-)super-net was trained for 50 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 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 AutoGAN [10]. Our search and training settings were identical to AutoGAN [10], which followed spectral normalization GAN [23] when training the (sub-)-super-nets. We split the super-net (shared GAN in [10]) to 3 sub-super-nets. 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 super-net. For each iteration, the shared GAN (super-net) was trained for 15 epochs, and the controller was trained for 30 steps. After the shared GAN (super-net) was trained, we transferred the weight to each sub-super-nets 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 7 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.



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

Figure 7: Anytime accuracy comparison of state-of-the-art gradient-based algorithms on *few-shot NAS* for CIFAR-100.