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 supernet, 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-adaptation 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 accuracy at 600 MB FLOPS and 77.5% top-1 accuracy at 238 MFLOPS; on CIFAR10, it reaches 98.72% top-1 accuracy without using extra data or transfer learning. In Auto-GAN, few-shot NAS outperforms the previously 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 interest over the past few years (Zoph et al., 2018; Tan et al., 2019; Baker et al., 2017). Using NAS, many deep learning tasks (Chen et al., 2019b; 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 a specific architecture can be approximately evaluated by picking the corresponding weights from the supernet (and masking out other missing edges, see Fig. 2(a)) without training, reducing the evaluation (and thus search) cost to just a few days (hours).

However, one-shot NAS suffers from degraded search performance due to inaccurate predictions from the supernet. Other works also have explicitly shown that using supernet degrades the final performance due to 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 with the analysis in (Yu et al., 2019b), the main reason for this 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. While one supernet may not be able to model the entire search space due to its limited capacity and co-adaptation of operations (Bender et al., 2018), using multiple supernets effectively addresses these issues by having each supernet modeling one part of the search space. The supernet allows parallel operations, i.e., compound edges, shown in Fig. 2(b) as multiple lines of different colors. In this case, we separate the entire search space $\Omega$ (i.e., all possible network operations) into disjoint partitions by picking one edge from the compound edge and assign a sub-supernet to model each leaf partition respectively. This procedure can be done in a recursive manner to yield a hierarchical partition. Although few-shot NAS increases the number of supernets, they can be trained efficiently by using a cascade of transfer learning: first, the root supernet is trained, then the child sub-supernet starts with the weights of the root and gets fine-tuned, etc. 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.

An immediate question is whether a lot of sub-supernets are needed to outperform one supernet. It turns out that a few sub-supernets already lead to strong performance, as demonstrated in Fig. 3. Empirically, 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 accuracy at 600 MB FLOPS and 77.5% top-1 accuracy at 238 MFLOPS on ImageNet, and 98.72% top-1 accuracy on CIFAR-10 without using extra data or transferring weights from a network pre-trained on ImageNet. Moreover, by reusing 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-adaptation 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 the 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 $\Omega$, and by masking, supernet can transform to any architecture in $\Omega$. This also implies we can enumerate all the architectures in $\Omega$ 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 $\Omega$; 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, 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 $\Omega_B$ and $\Omega_C$, respectively. So, the predictions from the 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 that contains 1296 architectures and trained each architecture toward 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, we empirically have multiple supernets to train, and we propose a recursive fine-tuning work in sec. 3 that can reduce the supernet training time.

3. Methodology

In designing few-shot NAS, we answer the following 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 the integration of few-shot NAS with existing NAS algorithms in Section 3.3 and Section 3.4.

3.1. Design of Split Strategy

Section 2 demonstrates the key insight of our method that the evaluation $f(S_A^{\Omega_k})$ of an architecture $A$ using a sub-supernet $S_A^{\Omega_k}$ is more accurate as $\Omega_k$ gets smaller. However, the time to split the initial architecture space $\Omega$ grows exponentially with tree depth. Therefore, the ideal split is to find the best trade-off of split supernet numbers and the total search time.

**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 to introduce some necessary concepts that will be used throughout the paper.

The whole architecture space $\Omega$ 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 compound 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

![Figure 4. A generic architecture space.](image-url)
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 has at least one edge from its predecessor node.

**Split Procedure Analysis.** Given a search space, e.g., one that was depicted in Figure 4, the full DAG contains \( \frac{n(n-1)}{2} \) compound edges. Each compound edge has \( m \) choices from the operations, resulting in a total of \( mn(n-1)/2 \) architectures. Training all \( mn(n-1)/2 \) architectures, as done by the vanilla NAS, can provide accurate rank information but is time-consuming. To evaluate the impact of compound edge splitting on ranking architectures, we calculate the Kendall’s Tau (rank correlation) for different split strategies on NasBench-201 (more details of this dataset in Section 4.1).

Specifically, we reuse the supernet from NasBench-201 with 5 operation types. The supernet is a 4 nodes full DAG containing 6 independent compound edges, and each compound edge consists of the 5 parallel pre-defined operations. Hence, there are \( C_6^1 = 6 \), \( C_6^2 = 15 \), and \( C_6^3 = 20 \) choices to split 1, 2, and 3 edges. For example, there are 6 choices to split 1 compound edge, and each choice would generate 5\(^1\) sub-supernets because of 5 operations in a compound edge. We trained all 6, 15, and 20 edge choices with corresponding 1, 2, and 3 compound edges split to investigate the impact of split edge choices on rank correlation.

Table 2 shows the rank correlation when split with different numbers of compound edges. First, similar to what we have observed in Section 2, increasing the number of split compound edges leads to a higher rank correlation. Second, given the same number of compound edges to split, the exact choice of which compound edge to split has a negligible impact on the rank correlation as indicated by the low standard deviation. Therefore, we can randomly choose which compound edge(s) to split and focus on how many compound 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. Generally, \( T \) is set to be twice of one-shot supernet training time.

### 3.2. Transfer Learning

The number of sub-supernets grows exponentially with the number of split compound edges. Directly training all the resulting sub-supernets can be computationally intractable and also lose the benefit of one-shot NAS. In this section, we integrate the transfer learning technique to accelerate the training procedure of sub-supernets.

Similar to how an architecture candidate \( A \) inherits weights \( \mathcal{W}_A \) from the supernet weights \( \mathcal{W}_S \), we allow a sub-supernet \( S' \) to inherit weights from its parent sub-supernet. For example, in Figure 2(b), after training the supernet of \( \Omega_A \), the supernet of \( \Omega_B \) and \( \Omega_C \) can inherit the weights from shared operations in supernet of \( \Omega_A \) as initialization and then start training. Compared to training from scratch, each sub-supernet converges in only a few epochs with transfer learning.

### 3.3. Integration with Gradient-based Algorithms

**Gradient-based NAS Overview.** Gradient-based algorithms work on a continuous search space, which can be converted from the DAG. Gradient-based algorithms treat 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 the 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 \( S \) 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 choose the sub-supernet \( S' \) with the lowest validation loss from all sub-supernets. Lastly, we pick the best architecture \( A' \) from the \( S' \) based on the architecture distribution parameters.

### 3.4. Integration with Search-based Algorithms

**Search-based NAS Overview.** For search-based algorithms, a value function of candidate architecture is needed to guide the search. The value function can be non-differentiable and is often provided by either one-shot or vanilla 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. By starting with a few initial architectures, search-based algorithms sample the next architecture \( A \) from the search space based on previous sampled architectures and search algorithms until an architecture with satisfactory performance is found.

To use search-based algorithms with our few-shot NAS, we

<table>
<thead>
<tr>
<th>#edges to split</th>
<th>#split choices</th>
<th>Mean</th>
<th>Std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0.653</td>
<td>0.012</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>0.696</td>
<td>0.016</td>
</tr>
<tr>
<td>3</td>
<td>125</td>
<td>0.752</td>
<td>0.018</td>
</tr>
</tbody>
</table>

Table 2. Rank correlation analysis using Kendall’s Tau (Kendall., 1938) for different split strategies.
will first train a number of sub-supernets. 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 $A$ falls into sub-supernet $S^{(i)}$, we will evaluate its performance $f(S^{(i)}_A)$ by inheriting the weights $W_{S^{(i)}}$. 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 architectures based on their performance.

4. Experiments

To evaluate the performance of few-shot NAS in reducing the approximation error associated with supernet and improving search efficiency of search algorithms, we conducted two types of evaluations. The first bases on an existing NAS dataset, and the second type compares 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, PCDARTS, 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., 2020). 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 the information of 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 information 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 such information on each architecture without time-consuming training.

4.1.1. Gradient-based Algorithms

Methodology. The supernet corresponding to NasBench-201 has four nodes and five operations. Based on the split method described in Section 3.1, we split one compound edge (i.e., parallel operations) in search space and obtain five sub-supernets. For this experiment, we train sub-supernets by skipping the transfer learning described in Section 3.2. Doing so allows us to compare the anytime performance between one-shot and few-shot NAS, by keeping the same training epochs. Due to the limit of the computing resource, we only split one compound edge in the search space since training more sub-supernets exponentially increases the time cost without transfer learning.

We chose a number of recently proposed gradient-based search algorithms, including DARTS, ENAS, PCDARTS, and SETN (Liu et al., 2019b; Pham et al., 2018; Xu et al., 2020; Dong & Yang, 2019) for evaluating the search performance of few-shot NAS. We used two metrics to compare the search performance between one-shot and few-shot NAS: (i) test accuracy which is obtained by evaluating the final architecture found by a NAS algorithm; and (ii) search time that includes the 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, when using one-shot NAS, both the architectures found by DARTS and ENAS can easily trap in a bad performance region with the search progress, an observation that is consistent with the original paper (Dong & Yang, 2020). The potential reason is one-shot NAS has an inaccurate performance prediction. In contrast, few-shot NAS could obtain higher-quality searched models than one-shot NAS. The good performance of few-shot NAS is likely due to using multiple supernets, allowing us to have a more accurate performance estimation 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. In short, we show that by using few-shot NAS, gradient-based algorithms can have more effective and efficient search 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 twice as that of one-shot NAS. Therefore, we only split the supernet by the one compound edge between the first node $N_0$ and second node $N_1$ (i.e. $E_0$ in Figure 4) to five sub-supernets. Note that, in this experiment, we trained all sub-supernets with transfer learning described in Section 3.2. We chose six different search-based algorithms including REA, REIN-
FORCE, BOHB, HB, SMAC, and TPE (Real et al., 2019; Zoph et al., 2018; Li et al., 2018; Falkner et al., 2018; Hutter et al., 2011; Bergstra et al., 2012) and ran each search algorithm 50 times. We evaluated the effectiveness of few-shot NAS by following the procedure described in Section 3.4. We used two metrics to evaluate the performance of search-based algorithms. The first metric is \(i^{\text{th}}\) best accuracy which denotes 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 includes the training time of supernet (for one-shot NAS) and sub-supernets (for few-shot NAS) and the time for finding the satisfiable architecture(s).

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

Figure 6(g) compares the total search time. All search-based algorithms took three to four orders of magnitude GPU hours when using vanilla NAS, compared to both one-shot NAS and our few-shot NAS. Further, few-shot NAS took similar search time compared to one-shot NAS, both completing the search within 24 hours. We also observe that, by using transfer learning, few-shot NAS saved a significant amount of search time compared to few-shot NAS without transfer learning. Such time saving with transfer learning was achieved with negligible search performance variation in terms of current best accuracy. Together, these results suggest that transfer learning is a valuable addition to our few-shot NAS design.

Finally, we compare the rank correlation (the higher, the better) in Table 3. We show that few-shot NAS, with the different number of sub-supernets, achieved better rank correl-
Few-shot Neural Architecture Search

Search time
(a) REA (b) BOHB (c) TPE
(d) REINFORCE (e) HB (f) SMAC

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 with/without transfer learning and vanilla NAS, respectively.

relation than prior work. The good correlation achieved by few-shot NAS indicates its effectiveness in finding higher accuracy architecture.

4.2. Evaluation on NasBench1-shot-1

We evaluated our few-shot NAS on NasBench1-shot-1 (Zela et al., 2020), which is a public neural architecture dataset similar to NasBench-201. Instead of creating a new search space like NasBench-201, NasBench1-shot-1 supports one-shot NAS algorithms by directly leveraging NasBench-101 (Ying et al., 2019). In this experiment, we used three sub-supernet in our few-shot NAS and kept all other settings the same as Section 4.1.

Result Analysis. Figure 7 shows search results on NasBench1-Shot-1. Our few-shot NAS improved the search efficiency both in terms of the test accuracy of found architectures and search time. Specifically, with DARTS, PCDARTS, and ENAS, our few-shot NAS can quickly find good architectures, i.e., finding architectures with test error less than 0.07 in about 10 epochs; while one-shot PCDARTS requires near 30 epochs to find an architecture with similar performance. The final searched architectures by few-shot NAS also outperform the counterparts by one-shot NAS.

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 used the same search space based on the original DARTS, REA, and LaNas.

Table 4 compares the search performance achieved by few-shot NAS to recently proposed NAS algorithms. We observe that few-shot NAS substantially improves the test accuracy across DARTS, REA, and LaNas. Specifically, we see that few-shot NAS with DARTS outperformed searching directly with one-shot (original) DARTS by 0.43 lower error. Further, the architecture found with few-shot NAS also outperforms 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
few-shot NAS architectures consistently outperform existing models that have similar parameters.

**Table 4.** Applying few-shot NAS on existing NAS methods on CIFAR-10 using the cutout (Deviere & Taylor, 2017) and autoaugmentation (Cubuk et al., 2018). Without auto-augmentation, few-shot DARTS-Small still consistently outperforms existing models with the same training setup. In short, all three architectures found by few-shot NAS prove from 8.55 to 8.63, and the FID was reduced from 8.30 to 8.78.

<table>
<thead>
<tr>
<th>Method</th>
<th>Data Augmentation</th>
<th>#Params</th>
<th>Err</th>
<th>GPU days</th>
</tr>
</thead>
<tbody>
<tr>
<td>NASNet-A</td>
<td>cutout</td>
<td>3.3M</td>
<td>2.65</td>
<td>2000</td>
</tr>
<tr>
<td>AmoebaNet-B-Small</td>
<td>cutout</td>
<td>2.8M</td>
<td>2.05</td>
<td>3150</td>
</tr>
<tr>
<td>AmoebaNet-B-Large (Real et al., 2019)</td>
<td>cutout</td>
<td>14.0M</td>
<td>2.13</td>
<td>N/A</td>
</tr>
<tr>
<td>AlphaNet</td>
<td>cutout</td>
<td>2.4SM</td>
<td>2.54</td>
<td>1000</td>
</tr>
<tr>
<td>NAS</td>
<td>cutout</td>
<td>3.2M</td>
<td>3.14</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Space</th>
<th>#Params</th>
<th>Top 1 Acc. (%)</th>
<th>FID Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoGAN (He et al., 2019)</td>
<td>Mobile</td>
<td>3.9M</td>
<td>74.48</td>
<td>N/A</td>
</tr>
<tr>
<td>MobileNet-V3-Large (Howard et al., 2019)</td>
<td>Mobile</td>
<td>5.4M</td>
<td>74.7</td>
<td>N/A</td>
</tr>
<tr>
<td>MnasNet-A2 (Tan et al., 2019)</td>
<td>Mobile</td>
<td>4.8M</td>
<td>75.6</td>
<td>N/A</td>
</tr>
<tr>
<td>FBNetV2-L (Wan et al., 2020)</td>
<td>Mobile</td>
<td>6.2M</td>
<td>77.2</td>
<td>600</td>
</tr>
<tr>
<td>EfficientNetD0 (Tan &amp; Le, 2019)</td>
<td>Mobile</td>
<td>3.5M</td>
<td>78.1</td>
<td>N/A</td>
</tr>
<tr>
<td>AutoNas (Mao et al., 2020)</td>
<td>Mobile</td>
<td>3.5M</td>
<td>77.6</td>
<td>N/A</td>
</tr>
<tr>
<td>few-shot OFA-Net-Small</td>
<td>Mobile</td>
<td>3.3M</td>
<td>77.0</td>
<td>68</td>
</tr>
<tr>
<td>MobileNet-V3-Large (Howard et al., 2019)</td>
<td>Mobile</td>
<td>4.8M</td>
<td>74.48</td>
<td>N/A</td>
</tr>
<tr>
<td>MobileNet-V3-Large (Howard et al., 2019)</td>
<td>Mobile</td>
<td>5.4M</td>
<td>74.7</td>
<td>N/A</td>
</tr>
<tr>
<td>MobileNet-V3-Large (Howard et al., 2019)</td>
<td>Mobile</td>
<td>6.2M</td>
<td>77.2</td>
<td>600</td>
</tr>
<tr>
<td>EfficientNetD0 (Tan &amp; Le, 2019)</td>
<td>Mobile</td>
<td>3.5M</td>
<td>78.1</td>
<td>N/A</td>
</tr>
<tr>
<td>AutoNas (Mao et al., 2020)</td>
<td>Mobile</td>
<td>3.5M</td>
<td>77.6</td>
<td>N/A</td>
</tr>
<tr>
<td>few-shot OFA-Net-Small</td>
<td>Mobile</td>
<td>3.3M</td>
<td>77.0</td>
<td>68</td>
</tr>
</tbody>
</table>

**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 OPA 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 OPA-Large achieves Sota 80.5% top1 accuracy at 600M FLOPS.

<table>
<thead>
<tr>
<th>Method</th>
<th>Space</th>
<th>#Params</th>
<th>Top 1 Acc. (%)</th>
<th>FID Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoGAN (He et al., 2019)</td>
<td>Mobile</td>
<td>3.9M</td>
<td>74.48</td>
<td>N/A</td>
</tr>
<tr>
<td>MobileNet-V3-Large (Howard et al., 2019)</td>
<td>Mobile</td>
<td>5.4M</td>
<td>74.7</td>
<td>N/A</td>
</tr>
<tr>
<td>MnasNet-A2 (Tan et al., 2019)</td>
<td>Mobile</td>
<td>4.8M</td>
<td>75.6</td>
<td>N/A</td>
</tr>
<tr>
<td>FBNetV2-L (Wan et al., 2020)</td>
<td>Mobile</td>
<td>6.2M</td>
<td>77.2</td>
<td>600</td>
</tr>
<tr>
<td>EfficientNetD0 (Tan &amp; Le, 2019)</td>
<td>Mobile</td>
<td>3.5M</td>
<td>78.1</td>
<td>N/A</td>
</tr>
<tr>
<td>AutoNas (Mao et al., 2020)</td>
<td>Mobile</td>
<td>3.5M</td>
<td>77.6</td>
<td>N/A</td>
</tr>
<tr>
<td>few-shot OFA-Net-Small</td>
<td>Mobile</td>
<td>3.3M</td>
<td>77.0</td>
<td>68</td>
</tr>
</tbody>
</table>

**Table 6.** Applying 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Inception Score</th>
<th>FID Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProGAN (He et al., 2019)</td>
<td>7.75±.14</td>
<td>24.60</td>
</tr>
<tr>
<td>SN-GAN (Myato et al., 2018)</td>
<td>8.22±.05</td>
<td>217.0±.01</td>
</tr>
<tr>
<td>MGAN (Hoang et al., 2018)</td>
<td>8.33±.12</td>
<td>26.7</td>
</tr>
<tr>
<td>Improving MMD GAN (Wang et al., 2019b)</td>
<td>8.29</td>
<td>16.21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Inception Score</th>
<th>FID Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoGAN-top1</td>
<td>8.55±.10</td>
<td>12.42</td>
</tr>
<tr>
<td>AutoGAN-top2</td>
<td>8.42±.06</td>
<td>13.67</td>
</tr>
<tr>
<td>AutoGAN-top3</td>
<td>8.41±.12</td>
<td>13.87</td>
</tr>
<tr>
<td>few-shot AutoGAN-top1</td>
<td>8.63±.09</td>
<td>10.73±.10</td>
</tr>
<tr>
<td>few-shot AutoGAN-top2</td>
<td>8.63±.09</td>
<td>10.73±.10</td>
</tr>
<tr>
<td>few-shot AutoGAN-top3</td>
<td>8.52±.08</td>
<td>12.20</td>
</tr>
</tbody>
</table>

**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 OPA 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 OPA-Large achieves Sota 80.5% top1 accuracy at 600M FLOPS.

**Table 6.** Applying 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.

**Penult 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 used 1 GPU day to find an architecture with worse performance (55.7 test Perplexity).

**Neural Architecture Search on ImageNet.** We selected two state-of-the-art NAS algorithms that were designed for the ImageNet, including ProxylessNAS and Once-for-All NAS (OFA) (Cai et al., 2019; 2020). We used the same training setup for few-shot NAS as ProxylessNAS and OFA.

Table 5 compares the search performance. First, we can see that our few-shot NAS significantly improved the accuracy and kept similar FLOPs numbers on both two NAS algorithms with the one-shot NAS. Second, our few-shot OPA achieves the best top-1 accuracy (with the same scale of FLOPs numbers) compared to architectures found with all other search methods.

**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 Frechet 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 using our few-shot NAS. We observe that by 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.
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 the 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 (Hu et al., 2020) and was shown to improve the architecture rank correlation. However, our few-shot NAS 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 into a few sub-supernets to eliminate the co-adaptation 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 approach, few-shot NAS, for fast evaluation of candidate network architectures in Neural Architecture Search. Few-shot NAS uses a few sub-supernets to efficiently and accurately evaluate the performance of candidate architectures, balancing the fast but low-quality evaluation in one-hot NAS with the expensive yet accurate evaluation in vanilla NAS. As a general evaluator, Few-shot NAS can be integrated with both gradient-based and search-based optimization of neural architecture. Our extensive evaluations on a recent NAS benchmark NasBench-201, NasBench1-shot-1 and deep learning applications demonstrated that few-shot NAS significantly improved search performance of all popular one-shot methods with negligible search time overhead. Furthermore, the final results from few-shot NAS also outperform previously published results by DART’s, ProxylessNas, Once-for-all NAS, and AUTO-GAN.

Acknowledgements

This work was supported in part by National Science Foundation Grants #1755659 and #1815619.

References


Few-shot Neural Architecture Search


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


Few-shot Neural Architecture Search


