

Subnet-Aware Dynamic Supernet Training for Neural Architecture Search

Jeimin Jeon^{1,2}Youngmin Oh¹Junhyup Lee³Donghyeon Baek¹Dohyung Kim⁴Chanho Eom⁵Bumsub Ham^{1*}¹Yonsei University²Articron Inc.³Samsung Research⁴Samsung Advanced Institute of Technology⁵Chung-Ang University<https://cvlab.yonsei.ac.kr/projects/DYNAS/>

Abstract

N-shot neural architecture search (NAS) exploits a supernet containing all candidate subnets for a given search space. The subnets are typically trained with a static training strategy (e.g., using the same learning rate (LR) scheduler and optimizer for all subnets). This however does not consider that individual subnets have distinct characteristics, leading to two problems: (1) The supernet training is biased towards the low-complexity subnets (unfairness); (2) the momentum update in the supernet is noisy (noisy momentum). We present a dynamic supernet training technique to address these problems by adjusting the training strategy adaptive to the subnets. Specifically, we introduce a complexity-aware LR scheduler (CaLR) that controls the decay ratio of LR adaptive to the complexities of subnets, which alleviates the unfairness problem. We also present a momentum separation technique (MS). It groups the subnets with similar structural characteristics and uses a separate momentum for each group, avoiding the noisy momentum problem. Our approach can be applicable to various *N*-shot NAS methods with marginal cost, while improving the search performance drastically. We validate the effectiveness of our approach on various search spaces (e.g., NAS-Bench-201, Mobilenet spaces) and datasets (e.g., CIFAR-10/100, ImageNet).

1. Introduction

Designing network architectures customized for each hardware configuration has led to a significant attention over the last decade, with the increasing demand for applying them to various platforms. Neural architecture search (NAS) enables finding optimal network architectures automatically, providing better performance, compared to hand-crafted ones. Early works on NAS exploit reinforcement learn-

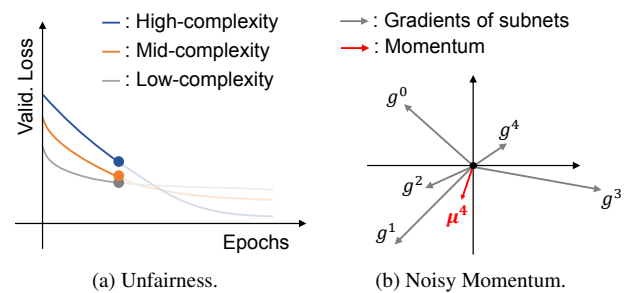


Figure 1. Illustrations of the challenges of *N*-shot NAS methods. (a) We visualize validation losses for the subnets having different complexities at training time. Existing methods do not consider the distinct optimization speed of subnets w.r.t. complexities. This causes an unfairness problem, where the high-complexity subnet is trained insufficiently, and the predicted performance falls behind the low-complexity one, even if it might be supposed to provide better performance. (b) We illustrate gradients g^t of subnets and the momentum μ^t at t -th iteration. We can see that the gradients vary according to the subnets, resulting in a noisy momentum and preventing a stable training process. (Best viewed in color.)

ing [32] or evolutionary algorithms [11] for searching network architectures, typically requiring extensive computational resources, e.g., in the order of thousands of GPU days [29, 39], which limits the practical applicability.

To reduce the search cost, *N*-shot NAS methods [3, 5–7, 12, 13, 15, 24, 28, 33, 38] have recently been introduced. They exploit either a single supernet for one-shot NAS [7, 12, 13, 28] or multiple sub-supernets partitioned from a supernet for few-shot NAS [15, 38]. The supernet is an over-parameterized network that contains all candidate architectures (*i.e.*, subnets) in a search space, serving as a performance predictor of subnets. For training the supernets, sampling-based methods [7, 12, 13, 28] are widely adopted nowadays due to their efficiency. These methods optimize the supernet by training one or multiple subnets sampled at each iteration. Existing methods, however, train the sampled subnets with a static training strategy, employing a fixed learning rate (LR) scheduler and the same op-

*Corresponding author.

imizer for all subnets within the supernet. These methods overlook the fact that the subnets have different characteristics (*e.g.*, complexities and structures) from each other, and thus they should be trained in different settings. The static training strategy mainly has two problems. First, it causes an unfairness problem, where the supernet training is biased towards the low-complexity subnets. The subnets with higher complexity typically need more training iterations than low-complexity counterparts, to reach sufficient performance levels for accurate ranking. Discarding this difference in the supernet training process places the rank of the high-complexity subnets behind the low-complexity ones, even when they might be supposed to provide better performance (Fig. 1(a)). Second, sharing a single optimizer for all subnets causes a noisy momentum problem. That is, the momentum storing the running mean of gradients becomes noisy, due to varying gradients across the subnets. Generally, exploiting a momentum in an optimizer stabilizes the training process, as it mitigates oscillations of network parameters by using the historical gradients. However, in the case of the supernet, the gradients come from diverse subnets within the search space, and they can vary significantly (Fig. 1(b)). Accordingly, the momentum characterized by the accumulation of the gradients fails to represent the gradients of individual subnets accurately, which might cause the gradients to steer toward the wrong direction.

In this paper, we introduce a novel dynamic supernet training framework that exploits distinct training strategies adaptive to individual subnets, effectively addressing the aforementioned problems for training supernets. To this end, we propose a complexity-aware LR scheduler (CaLR) that alleviates the unfairness problem for the subnets. Specifically, considering the number of parameters to tune for each subnet, we slowly decay LR of high-complexity subnets to encourage more parameter updates, while we accelerate the LR decay for low-complexity ones. This enables a more balanced training of each subnet according to its complexity, alleviating the ranking inconsistencies. Moreover, we introduce two metrics, Complexity Bias (CB) and Complexity-Convergence Correlation (C^3), that measure the unfairness problem in supernet training, and show that our CaLR can effectively reduce the unfairness problem. We also present a momentum separation technique (MS) that stabilizes the supernet training process. Motivated by the observation that the subnets with similar structures provide similar gradients [19, 28], we cluster the subnets based on the structural characteristics, and use a separate momentum buffer for each cluster, stabilizing the momentum updates. We show that our dynamic training strategy (CaLR + MS) can be applied to various N -shot NAS methods [7, 12, 38] efficiently, improving the search performance, and demonstrate the effectiveness of our method on various datasets, including CIFAR-

10/100 [20] and ImageNet [8]. The main contributions of our work are summarized as follows:

- We propose CaLR adjusting the LR based on the complexity of the subnets, addressing the unfairness problem. To measure the unfairness problem in supernet training, we introduce evaluation metrics, CB and C^3 .
- We propose MS exploiting a distinct momentum buffer for the subnets having similar structural characteristics, mitigating the noisy momentum problem and stabilizing a training process.
- Extensive experiments demonstrate that our approach enhances various N -shot NAS methods consistently, with a negligible additional search time.

2. Related Work

One-shot NAS. Early approaches to NAS adopt reinforcement learning [32] or evolutionary algorithms [11] to find an optimal network architecture, taking a huge amount of search costs [29, 39]. To tackle this problem, one-shot NAS methods [2, 3, 5–7, 12, 13, 16, 24, 27, 31, 33] exploit a supernet, that is, an over-parametrized network including all candidate subnets in a search space. Once training a single supernet, one-shot NAS methods can predict the performance of several subnets in the supernet without training each from scratch. Differentiable methods [3, 5, 6, 24, 33] formulate each layer as a weighted combination of candidate operations, where the weights are trained alternately with the parameters for the supernet. They should store gradients and activations for each operation at every layer, requiring substantial memory. To address this problem, sampling-based approaches [2, 7, 12, 13, 16, 27, 31] optimize the supernet by training a randomly sampled subnet at each step, storing the gradients and activations for the single subnet only. Despite the efficiency, they suffer from a poor ranking consistency mainly due to the weight sharing among the subnets [35]. That is, the weights of the subnets interfere with one another, which leads each subnet to converge insufficiently, and degrades the reliability of the supernet in terms of performance predictions. This problem can be alleviated by narrowing a search space [2, 16], or exploiting distillation [27, 31] or multi-subnet sampling strategies [7, 13]. Specifically, the works of [2, 16] shrink the search space by removing the redundant operations in order to lower the degree of the sharing. In [27, 31], a distillation technique [14] is adopted to boost the convergence of individual subnets. FairNAS [7] and SUMNAS [13] exploit multi-subnet sampling strategies for more balanced training across each subnet. Specifically, FairNAS [7] samples multiple subnets for each training step, and trains them under the same setting. SUMNAS [13] further improves FairNAS by adopting a meta-learning approach to alleviate the forgetting problem occurring during a supernet training [36]. Namely, the knowledge from subnets becomes forgotten

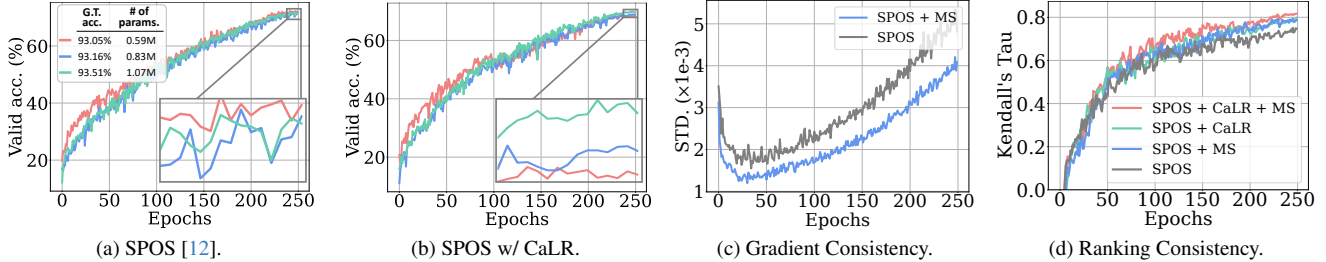


Figure 2. Empirical comparisons of SPOS [12] and SPOS with our dynamic training strategy. We train a supernet using the NAS-Bench-201 search space [9] on CIFAR-10 [20]. (a-b) Validation accuracies for three subnets sampled from the supernet using SPOS [12] without and with CaLR. Note that the sampled subnets have different complexities and ground-truth accuracies. (c) Plots of gradient consistency in terms of the standard deviation of gradients [25]. A smaller value indicates more consistent gradient direction over the training iterations. (d) Comparisons of various methods in terms of ranking consistency of the supernet, using the Kendall’s Tau [18].

gradually, as the training process goes on. These methods still exploit a static strategy to train a supernet, while not considering the unique characteristics of each subnet.

Recent approaches [17, 25, 34, 37] propose to use a non-uniform subnet sampling strategy, with an observation that sampling each subnet uniformly across the supernet with an equal probability is sub-optimal. GreedyNAS [17, 34] mainly samples potentially high-performing subnets, instead of weak ones. PA&DA [25] shows that reducing gradient variances in the supernet enhances the search performance in NAS, and proposes to sample subnets and images that lead to lower gradient variances. ShiftNAS [37] uses more training steps for the subnets that are not sufficiently trained compared to others. Our method is orthogonal to these approaches in that we focus on the optimization process of each subnet, rather than the sampling strategies. Ours is also efficient to implement, while estimating the distributions for sampling is computationally expensive.

Few-shot NAS. Few-shot NAS methods [15, 38] divide a supernet into multiple sub-supernets, and apply one-shot NAS approaches for each sub-supernet, to prevent coupling weights among subnets, providing better results, compared to one-shot approaches. Few-shot NAS methods focus on designing criteria for partitioning the supernet, and various strategies have been introduced. FSNAS [38] shows that randomly partitioning a supernet could be a simple solution to obtain decoupled weights among subnets. GMNAS [15] improves the splitting strategy using the gradient similarity between the subnets. Specifically, it assumes that subnets with different gradient directions are more likely to interfere with each other, and thus they should be separated. Similar to the one-shot NAS approaches, our method can also be applicable to the few-shot NAS methods, helping to better optimize each sub-supernet.

3. Method

In this section, we describe our approach within a framework of single-path one-shot NAS (SPOS) [12]. Extensions to other approaches [7, 38] other than SPOS are shown

in the supplement. We first review the search process of SPOS [12] (Sec. 3.1), and describe our dynamic supernet training method (Sec. 3.2).

3.1. Supernet training and architecture search

One-shot NAS methods aim to train weights \mathcal{W} of a supernet that contains all subnets α in a search space \mathcal{A} . They try to minimize an expected loss across all the subnets to estimate optimal weights \mathcal{W}^* as follows:

$$\mathcal{W}^* = \underset{\mathcal{W}}{\operatorname{argmin}} \mathbb{E}_{\alpha \sim \mathcal{A}} [\mathcal{L}(\alpha; \mathcal{W}(\alpha))], \quad (1)$$

where we denote by $\mathcal{W}(\alpha)$ active weights for the subnet α . $\mathcal{L}(\alpha; \mathcal{W}(\alpha))$ is a training loss for the subnet α with the weights of $\mathcal{W}(\alpha)$. At each training step, a subnet α is sampled from the search space \mathcal{A} uniformly, and the weights $\mathcal{W}(\alpha)$ are updated to minimize the training loss $\mathcal{L}(\alpha; \mathcal{W}(\alpha))$. Exploiting the trained supernet as a performance predictor for all possible subnets, we can search for the optimal architecture α^* under specific constraints, including FLOPs and latency, as follows:

$$\alpha^* = \underset{\alpha \in \mathcal{A}}{\operatorname{argmin}} L_{val}(\alpha; \mathcal{W}(\alpha)), \quad (2)$$

where L_{val} is a validation loss.

3.2. Subnet-aware dynamic training

Since the supernet is used to predict the performance of each subnet, improving a ranking consistency for the supernet is essential to achieve the better search performance. Existing methods assume that all subnets have the same convergence rate, if they are trained with the same setting. Accordingly, they typically use a static supernet training strategy, where the weights associated with each subnet (e.g., $\mathcal{W}(\alpha)$ and α , respectively, in Eq. (1)) are updated with the same LR scheduler and optimizer. The assumption does not hold in practice, since characteristics of the subnets are different from each other, leading to the unfairness and noisy momentum problems. To better understand

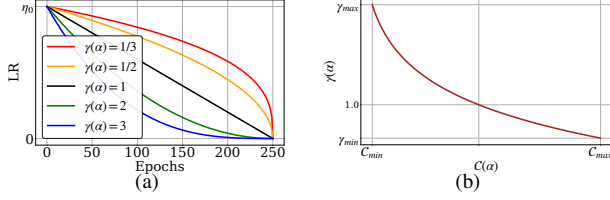


Figure 3. (a) Plots of LR by CaLR with varying the decay ratio of $\gamma(\alpha)$. CaLR sets a small decay ratio (*i.e.*, a large LR) for high-complexity networks, and vice versa. (b) Visualization of the decay ratio $\gamma(\alpha)$ based on the complexity score $C(\alpha)$.

these problems, we provide empirical analyses of the effect of the static supernet training strategy in Fig. 2. First, we visualize the validation accuracies of subnets with different complexities at training time (Fig. 2(a)). We can see that a static training approach may underestimate the performance of high-complexity subnets (green line) compared to low-complexity ones (red line), even though the former potentially has higher ground-truth accuracy. We also plot the gradient variance of the supernet at each training epoch (Fig. 2(c)). We can see that the variance is large over the training process, as the gradients vary significantly. This can cause a noisy momentum problem, making a supernet training process unstable. These problems lead to lower ranking consistency (Fig. 2(d)), which can result in sub-optimal search performance.

To address these problems, we introduce a novel supernet training method that adaptively adjusts training strategies for each subnet. Specifically, we propose a complexity-aware LR scheduler (CaLR), the LR scheduler adjusting LR values adaptively w.r.t. complexities of the subnets, to alleviate the unfairness problem. We also present a momentum separation technique (MS) that assigns different momentum buffers to the subnets, while considering their structural characteristics, stabilizing a training process. In the following, we provide detailed descriptions of CaLR and MS.

3.2.1. Complexity-aware LR Scheduler (CaLR)

We define the complexity score of a subnet as the number of parameters to tune¹. That is, the subnets of high complexities have more network parameters to learn, compared to those of low complexities, suggesting that the high-complexity subnets require more training steps for accurate ranking (Figs. 1(a) and 2(a)). Instead of directly training the subnets with more steps, which is computationally expensive, we propose to adjust a decay ratio of an LR scheduler adaptive to the complexity of subnets. Specifically, we decay LR values slowly for the subnets of high complexities, thereby anticipating more extensive training, while rapidly

¹Although we can exploit FLOPs as a complexity score, this requires more computations, compared to counting the number of parameters. In addition, we have empirically found that the number of parameters provides better results in terms of the Kendall’s Tau (see the supplement).

decaying them for the ones of low complexities (Fig. 3(a)).

Concretely, we formulate our CaLR as a polynomial function with a decay ratio $\gamma(\alpha)$ as follows:

$$\eta^t = \eta^0 \cdot \left(1 - \frac{t}{T}\right)^{\gamma(\alpha)}, \quad (3)$$

where η^t is a LR at the t -th step, and T is the total number of training steps. We lower the LR of high-complexity subnets slowly (*i.e.*, $\gamma_{\min} < \gamma(\alpha) < 1$), while decaying it faster for low-complexity ones (*i.e.*, $1 < \gamma(\alpha) < \gamma_{\max}$) (Fig. 3(a)), where γ_{\min} and γ_{\max} are the minimum and maximum decay ratio of CaLR, respectively, which we set as hyperparameters (see the supplement). To achieve this, we define the decay ratio $\gamma(\alpha)$, which is inversely proportional to the complexity of the subnet, as follows:

$$\gamma(\alpha) = \omega \log(C(\alpha)) + \tau, \quad (4)$$

where $C(\alpha)$ is a complexity score of a subnet α , defined as the number of parameters for the subnet, and ω and τ are coefficients for the affine transformation:

$$\omega = -\frac{\gamma_{\max} - \gamma_{\min}}{\log(C_{\max}) - \log(C_{\min})}, \quad (5)$$

$$\tau = \gamma_{\min} - \omega \log(C_{\max}). \quad (6)$$

We denote by C_{\min} and C_{\max} the minimum and maximum complexity scores of subnets in a search space, respectively. To compute the decay ratio $\gamma(\alpha)$ in Eq. (4), the complexity score is fed into the logarithmic function (Fig. 3(b)), which is a key to differentiating the LR decay for the subnets of different complexities. We then apply an affine transform with the coefficients of ω and τ to calibrate the score of $\log(C(\alpha))$ within the range of $\gamma(\alpha)$. Note that the logarithmic function enables the subnet with a complexity score close to the medium (around the center point between C_{\min} and C_{\max}) to be trained with a linearly decaying LR, *i.e.*, $\gamma(\alpha) = 1$ (Fig. 3(b)).

Adjusting LR based on the complexities of subnets by CaLR in Eq. (3) provides higher LR for the subnets of high complexities, and vice versa. Considering that high-complexity subnets have lots of parameters to tune, CaLR has an effect of exploring the parameter space more thoroughly for the high-complexity subnets, mitigating the unfairness problem (Fig. 2(b)).

3.2.2. Momentum Separation (MS)

Using a single optimizer for all subnets causes a noisy momentum problem. Since the sampled subnets vary at each training step, it is not enough to represent the gradients of individual subnets accurately by a single momentum buffer. It is highly likely that the gradients steer toward sub-optimal points (Figs. 1(b) and 2(c)). A straightforward solution is to use an individual momentum buffer for each subnet.

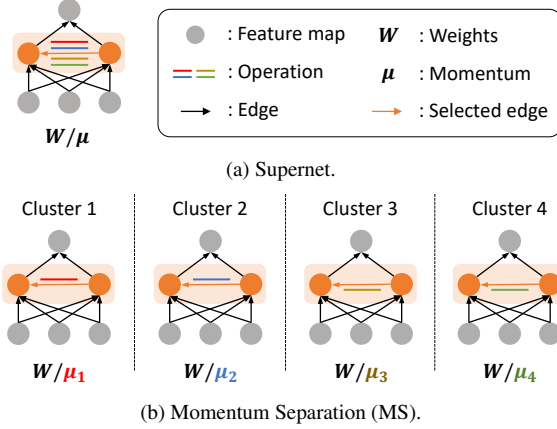


Figure 4. (a) The supernet shares weights and momentum for all subnets. (b) MS selects a single edge (or layer) from the supernet and clusters the subnets according to operations for the edge. It then assigns a distinct momentum buffer for each cluster, while the weights are shared for all clusters.

This, however, requires substantial memory, due to huge amounts of subnets in a search space (e.g., $\sim 7^{21}$ for MobileNet space [1]) and the size of each momentum buffer, which is the same as the number of parameters in the subnet. MS instead clusters the subnets using the structural characteristics, and assigns a distinct momentum buffer to each cluster. The reason behind this is that subnets with similar structures provide similar gradients during training [19, 28] (see the supplement for empirical verification). This suggests that the gradients of subnets from each cluster are likely to be consistent, alleviating the noisy momentum problem (Fig. 2(c)).

To this end, we consider that subnets are structurally similar, if they have the same operation at a specific edge (or layer) (Fig. 4)², and cluster the subnets as follows:

$$S_i = \{\alpha \in \mathcal{A} \mid \text{op}(\alpha, e) = o_i\}, \quad i \in [1, n], \quad (7)$$

where S_i is a cluster of subnets α with the same operation o_i at the chosen edge e , and $\text{op}(\alpha, e)$ represents an operation of the subnet α at edge e . Note that every edge in the supernet contains a set of candidate operations, $O = \{o_1, o_2, \dots, o_n\}$, where n is the number of candidate operations, and the subnets adopt a single operation from the set of O for every edge. For each cluster S_i , we assign a distinct momentum buffer μ_i . That is, if the sampled subnet α belongs to the cluster S_i , we update the momentum buffer μ_i at each training step, as follows:

$$\mu_i^t = \beta \cdot \mu_i^{t-1} + g^t, \quad (8)$$

where β is a coefficient for moving average, and g^t is the gradient w.r.t. a task loss at the t -th step. Note that MS

²In practice, we select the first layer for a macro search space (e.g., MobileNet space [1]), and a random edge for a micro search space (e.g., NAS-Bench-201 [9]). See the supplement for more details.

separates the momentum update process only, while using shared weights across all subnets, suggesting that the overall search time remains the same. Note also that the memory overheads for the momentum buffer are negligible, compared to those for forward and backward passes during supernet training [10]. MS is also complementary to CaLR, and thus exploiting both CaLR and MS can further boost the ranking consistency (Fig. 2(d)).

4. Experiments

In this section, we first describe our experimental settings (Sec. 4.1). We then demonstrate the effectiveness of our approach through extensive experiments, including quantitative results on various search spaces and datasets (Sec. 4.2), and in-depth analyses on our method (Sec. 4.3).

4.1. Experimental settings

We validate our approach on widely used search spaces, including the MobileNet space [1] and NAS-Bench-201 [9]. More details including the search spaces and hyperparameters are provided in the supplement.

MobileNet. The MobileNet space [1] consists of subnets with 21 mobile inverted bottleneck convolution (MBConv) layers [30]. Each MBConv layer has various parameters to determine, including the kernel sizes of (3, 5, 7) and expansion ratios of (3, 6). All layers except for the downsampling layers use a skip connection as an additional candidate operator, resulting in $\sim 7^{21}$ size of the search space.

We follow the standard settings [7, 12, 13, 25, 26] for evaluating the search performance, consisting of supernet training, evolutionary search, and retraining. We train the supernet on ImageNet [8] using the SGD optimizer with a momentum coefficient of 0.9 and a weight decay of $4e-5$. We set an initial LR as 0.045 and decay the LR with our CaLR. We apply our dynamic supernet training to three NAS approaches, including one-shot NAS methods (i.e., SPOS [12] and FairNAS [7]) and a few-shot NAS method (i.e., FSNAS [38]). We use an evolutionary search [11] to discover the best subnet. We retrain the retrieved subnet to obtain the final accuracy, using the same setting of [21–23].

NAS-Bench-201. NAS-Bench-201 [9] is a cell-based search space, which provides the stand-alone accuracies of each subnet on CIFAR-10, CIFAR-100 [20] and ImageNet-16-120 [4]. The cell structure of NAS-Bench-201 can be represented by a directed acyclic graph (DAG) with 4 nodes and 6 edges. Each edge in the DAG is characterized by one of the five candidate operations, including zeroize, skip-connect, 1×1 and 3×3 convolutions, and average pooling, leading to total 5^6 subnets in the search space.

Table 1. Quantitative comparison of the search performance on ImageNet [8] in the MobileNet space [1]. We report the top-1 validation accuracy, together with the number of parameters and FLOPs for the retrieved subnets. We also report the peak memory usage, and the GPU hours for training supernet, computed with 4 A5000 GPUs.

| Methods | Params. (M) | FLOPs (M) | Top-1 (%) | Peak Memory (MB) | GPU Hours (hours) |
|------------------|-------------|-----------|-------------|------------------|-------------------|
| SPOS-S [12] | 4.1 | 330 | 75.6 | | |
| SPOS-M | 4.3 | 387 | 76.2 | 54283 | 157 |
| SPOS-L | 4.5 | 471 | 76.6 | | |
| SPOS-S + Ours | 3.9 | 329 | 75.9 | | |
| SPOS-M + Ours | 4.5 | 396 | 76.7 | 54579 | 159 |
| SPOS-L + Ours | 4.7 | 459 | 77.1 | | |
| FairNAS-S [7] | 4.1 | 330 | 75.7 | | |
| FairNAS-M | 4.3 | 394 | 76.3 | 54400 | 364 |
| FairNAS-L | 4.7 | 472 | 76.7 | | |
| FairNAS-S + Ours | 4.1 | 326 | 75.8 | | |
| FairNAS-M + Ours | 4.5 | 399 | 76.6 | 54612 | 369 |
| FairNAS-L + Ours | 4.7 | 471 | 77.0 | | |
| FSNAS-S [38] | 4.0 | 330 | 75.8 | | |
| FSNAS-M | 4.3 | 399 | 76.4 | 54537 | 740 |
| FSNAS-L | 4.7 | 464 | 76.8 | | |
| FSNAS-S + Ours | 4.0 | 324 | 76.0 | | |
| FSNAS-M + Ours | 4.4 | 398 | 76.6 | 54724 | 744 |
| FSNAS-L + Ours | 4.5 | 467 | 77.2 | | |

We follow the supernet training scheme provided by the benchmark [9]. Specifically, we use the SGD optimizer with a momentum coefficient of 0.9 and a weight decay of $5e-4$. We use our CaLR as an LR scheduler with an initial LR of 0.025. We apply our method to SPOS [12], FairNAS [7] and FSNAS [38] as in the MobileNet space. Following [13], we evaluate all subnets within the supernet to find the best one, and report the search performance in terms of the top-1 test accuracy of the retrieved subnet. To measure the ranking consistency of a supernet, we compute the Kendall’s Tau [18] with a small set of subnets, where they are sampled following the strategy in [13], instead of using the whole subnets in the search space, which helps minimize noise in the ranking list [13]. Specifically, we evenly split entire subnets into 400 chunks based on stand-alone accuracies, and sample the best-performing one in each chunk. We then compute the Kendall’s Tau for the sampled subnets, using the rankings estimated by NAS methods and the stand-alone accuracies. For all experiments on NAS-Bench-201, we report the average and standard deviation over 3 runs.

4.2. Results

We evaluate our method on various search spaces, including the MobileNet space [1] and NAS-Bench-201 space [9]. More quantitative results, including comparisons to the state of the art and evaluations on additional search spaces, are provided in the supplement.

MobileNet. We compare in Table 1 the performance of subnets retrieved by our method and the baselines on the MobileNet search space, in terms of the top-1 validation accuracy on ImageNet, together with the number of parameters and FLOPs of the searched subnets. We also report the peak memory usage, and the GPU hours for training supernet. We apply our dynamic training technique to SPOS [12], FairNAS [7] and FSNAS [38], where we denote by SPOS + Ours, FairNAS + Ours and FSNAS + Ours, respectively. We search the subnets with various FLOPs constraints including 330M (small, S), 400M (medium, M), and 475M (large, L). We summarize the results of Table 1 in threefold: (1) Applying our dynamic training technique improves the performance of three baselines (SPOS, FairNAS, and FSNAS), regardless of FLOPs constraints. This demonstrates the effectiveness of our dynamic supernet training scheme, compared to the static one. Note that each baseline exploits distinct supernet training strategy, *e.g.*, sampling a single [12] or multiple [7] subnets at each training iteration, or using multiple sub-supernet [38]. This suggests that our method can be applied in a plug-and-play manner across diverse supernet training algorithms. (2) The performance gain of our method is generally more significant for the subnets with higher complexities. This is because, the subnets with higher complexities are more likely to suffer from the unfairness problem, resulting in poor ranking consistency, and our method can alleviate the problem by providing a more balanced training for each subnet. (3) Additional memory costs and time consumptions for our method are negligible, compared to those for the baselines. The increase in memory comes from the additional momentum buffers in MS. It is negligible, compared with the overall memory usage for forward and backward passes [10].

NAS-Bench-201. We show in Table 2 the search performance in the NAS-Bench-201 space, in terms of ranking consistencies and top-1 accuracies. We can see that the three baselines [7, 12, 38] coupled with our dynamic supernet training method provide better search performance consistently with negligible additional search cost. This confirms once again that our dynamic supernet training technique can be an effective alternative to the static one.

4.3. Discussion

We provide an ablation study and in-depth analyses on each component of our method, including CaLR and MS. For more discussions, please refer to the supplement.

Ablation study. We report in Table 3 the ablation study on each component of our method. We apply CaLR and MS to SPOS [12], FairNAS [7] and FSNAS [38], and measure the Kendall’s Tau and top-1 accuracy. We can see that applying either CaLR or MS improves the ranking consistency and search performance of the baselines. This implies that

Table 2. Quantitative comparison of different supernet training methods on CIFAR-10, CIFAR-100 [20], and ImageNet16-120 [8] datasets in NAS-Bench-201 [8]. We report the Kendall’s Tau, along with the top-1 accuracy (Top-1 Acc.) for each method. We also report the peak memory usage, and the GPU hours for training supernets on CIFAR-10, computed with a single RTX 2080Ti. The results include the average and standard deviations for 3 runs.

| Methods | CIFAR-10 | | CIFAR-100 | | ImageNet16-120 | | Peak Memory (MB) | GPU Hours (hours) |
|----------------|----------------------|---------------------|----------------------|---------------------|----------------------|---------------------|------------------|-------------------|
| | Kendall’s Tau | Top-1 Acc. | Kendall’s Tau | Top-1 Acc. | Kendall’s Tau | Top-1 Acc. | | |
| SPOS [12] | 0.751 ± 0.008 | 93.12 ± 0.03 | 0.787 ± 0.011 | 68.16 ± 0.35 | 0.718 ± 0.009 | 42.65 ± 0.33 | 1875 | 2.67 |
| SPOS + Ours | 0.814 ± 0.007 | 93.50 ± 0.33 | 0.81 ± 0.024 | 69.72 ± 0.41 | 0.743 ± 0.011 | 43.30 ± 0.70 | 1901 | 2.69 |
| FairNAS [7] | 0.766 ± 0.015 | 92.13 ± 0.18 | 0.79 ± 0.009 | 67.72 ± 0.40 | 0.699 ± 0.008 | 39.81 ± 0.42 | 1877 | 5.13 |
| FairNAS + Ours | 0.828 ± 0.020 | 93.52 ± 0.50 | 0.857 ± 0.015 | 70.91 ± 1.35 | 0.776 ± 0.016 | 44.63 ± 0.37 | 1903 | 5.15 |
| FSNAS [38] | 0.729 ± 0.019 | 93.43 ± 0.24 | 0.757 ± 0.013 | 69.49 ± 0.08 | 0.688 ± 0.018 | 42.97 ± 0.52 | 1901 | 16.04 |
| FSNAS + Ours | 0.767 ± 0.010 | 93.63 ± 0.21 | 0.774 ± 0.028 | 71.05 ± 0.17 | 0.728 ± 0.017 | 44.68 ± 0.06 | 1925 | 16.12 |
| Optimal | - | 94.37 | - | 73.51 | - | 47.31 | - | - |

Table 3. Quantitative comparison of the ranking consistency on CIFAR-10, CIFAR-100 [20], and ImageNet16-120 [8] datasets in NAS-Bench-201 [8]. We apply CaLR and MS to SPOS [12], FairNAS [7], and FSNAS [38], and report Kendall’s Tau / Accuracy.

| Baselines | CaLR | MS | CIFAR-10 | CIFAR-100 | ImageNet16-120 |
|-------------|------|----|--------------------|--------------------|--------------------|
| | - | - | 0.751/93.12 | 0.787/68.16 | 0.718/42.65 |
| SPOS [12] | ✓ | - | 0.805/93.3 | 0.803/68.94 | 0.733/42.83 |
| | - | ✓ | 0.772/93.45 | 0.807/69.11 | 0.724/43.04 |
| | ✓ | ✓ | 0.814/93.5 | 0.81/69.72 | 0.743/43.3 |
| FairNAS [7] | - | - | 0.766/92.13 | 0.790/67.72 | 0.699/39.81 |
| | ✓ | - | 0.819/93.21 | 0.832/69.29 | 0.74/43.13 |
| | - | ✓ | 0.784/92.89 | 0.815/68.93 | 0.725/42.89 |
| | ✓ | ✓ | 0.828/93.52 | 0.857/70.91 | 0.776/44.63 |
| FSNAS [35] | - | - | 0.729/93.43 | 0.757/69.49 | 0.688/42.97 |
| | ✓ | - | 0.749/93.57 | 0.764/70.52 | 0.71/43.32 |
| | - | ✓ | 0.750/93.52 | 0.769/69.97 | 0.705/43.51 |
| | ✓ | ✓ | 0.767/93.63 | 0.774/71.05 | 0.728/44.68 |

the static training scheme suffers from the unfairness and noisy momentum problems, regardless of the type of NAS methods, which can be alleviated by our dynamic training. We also observe that CaLR and MS can be complementary to each other, and applying both CaLR and MS further improves the ranking consistency.

Unfairness Problem. To see how the complexities of subnets affect a training process, we introduce two metrics, Complexity Bias (CB) and Complexity-Convergence Correlation (C^3). First, CB is designed to detect the bias caused by the complexities of subnets. Specifically, it computes the ratio of subnet pairs, where the subnet predicted in low rank has a higher complexity, but it is supposed to provide better performance:

$$CB = \frac{\sum_{(\alpha, \beta) \in \mathcal{S}} \mathbb{I}(\alpha, \beta) \mathbb{J}(\alpha, \beta)}{\sum_{(\alpha, \beta) \in \mathcal{S}} \mathbb{I}(\alpha, \beta)}, \quad (9)$$

where \mathcal{S} is a set of all possible pairs of subnets, and (α, β) is a pair of subnets within the set. The indicator function $\mathbb{I}(\alpha, \beta)$ returns 1, when the predicted ranking of the pair (α, β) misaligns with the ground truth, and 0 otherwise.

Table 4. Quantitative comparisons of CB and C^3 on CIFAR-10 [20] in the NAS-Bench-201 space [9]. Note that we sample the top 30% of subnets (4688 out of 5^6) based on the stand-alone accuracies to compute CB and C^3 , and we set k to $\mathcal{C}_{\min} + \mathcal{C}_{\max}$. See text for more details.

| Baseline | CaLR | $\gamma(\alpha) \propto$ | CB | C^3 |
|-----------|------|----------------------------------|-------------|--------------|
| | - | - | 0.76 | -0.19 |
| SPOS [12] | ✓ | $-\log(k - \mathcal{C}(\alpha))$ | 0.79 | -0.22 |
| | ✓ | $-\log(\mathcal{C}(\alpha))$ | 0.63 | -0.09 |

The function $\mathbb{J}(\alpha, \beta)$ is similarly defined but returns 1, when the subnet in the pair predicted in low rank has a higher complexity. Second, C^3 identifies the relationship between the complexity of subnets and the convergence ratio:

$$C^3 = \tau(\mathcal{C}(\mathcal{A}), CR(\mathcal{A})), \quad (10)$$

where $\tau(\cdot)$ outputs the Kendall’s Tau between inputs. $\mathcal{C}(\mathcal{A})$ and $CR(\mathcal{A})$ are the set of complexity scores and the convergence ratio of subnets in search space \mathcal{A} , respectively. The convergence ratio is defined as the accuracy estimated from the supernet over the stand-alone accuracy. Note that CB and C^3 approach to 0.5 and 0, respectively, as the ranking predictions of the supernet become less related to the subnet complexity.

We show in Table 4 the results of CB and C^3 on CIFAR-10 in NAS-Bench-201 [9]. We can see that the supernet trained using SPOS [12] provides CB of 0.76, which indicates that the supernet mis-ranks the subnets mainly due to the complexities. At the same time, the supernet shows a negative correlation between the convergence ratio and the subnet complexity (*i.e.*, C^3 of -0.19). This suggests that high-complexity subnets tend to be insufficiently trained compared to low-complexity ones, leading to the unfairness problem. From the first and third rows, we can see that applying CaLR to SPOS boosts the performance in terms of CB and C^3 . This shows that CaLR provides more adequate training for the subnets according to their complexities, which is a key factor in alleviating the unfairness problem.

Table 5. Quantitative results of the ranking consistency on CIFAR-10 [20] in the NAS-Bench-201 space [9] with different subnet clustering algorithms.

| Baseline | MS | Clustering | Kendall’s Tau |
|-----------|----|-----------------|--------------------|
| | - | - | 0.751±0.008 |
| SPOS [12] | ✓ | Random | 0.750±0.019 |
| | ✓ | Operation-based | 0.772±0.007 |

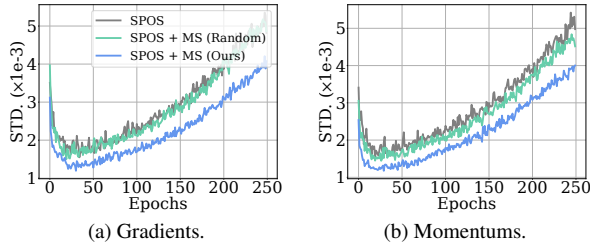


Figure 6. Plots of standard deviations for gradients and momentums of the supernet on CIFAR-10 [20] in NAS-Bench-201 [9].

Note that the LR decay ratio of CaLR is inversely proportional to the complexity score (*i.e.*, $\gamma(\alpha) \propto -\log(\mathcal{C}(\alpha))$). To further validate the effectiveness of CaLR, we apply it with a decaying strategy of an opposite effect, where the decay ratio is proportional to the complexity score of the subnet (*i.e.*, $\gamma(\alpha) \propto -\log(k - \mathcal{C}(\alpha))$, where $k = C_{\min} + C_{\max}$). We can see from the second row that this variant rather degrades the performance in terms of CB and C^3 , compared to the baseline, demonstrating that our LR strategy is effective in mitigating the unfairness problem.

Clustering algorithms. We show in Table 5 the effectiveness of MS in terms of the ranking consistency on CIFAR-10 in NAS-Bench-201. We apply MS to SPOS [12] with various clustering strategies. For the random clustering, subnets are randomly divided into 5 clusters, and assign a distinct momentum for each cluster, while our approach groups the subnet based on the shared operations at a chosen edge. We can see from the first and the third rows that our approach enhances the performance of SPOS [12]. On the other hand, the random clustering does not provide a performance gain, and the randomness causes the large variance of the Kendall’s Tau, suggesting that the MS is largely influenced by grouping techniques. We visualize in Fig. 6 gradient and momentum consistencies. Following [25], we compute the average standard deviations of gradients and momentums to measure the consistencies. We can see that applying our approach to SPOS reduces the standard deviations of gradients and momentums, demonstrating the effectiveness of our approach in handling the noisy momentum problem. Moreover, we observe that our clustering approach provides much lower standard deviations of gradients and momentums, compared to the random clustering.

Number of clusters. We show in Table 6 ablations on the number of clusters for MS on CIFAR-10 in NAS-Bench-

Table 6. Quantitative results of the ranking consistency on CIFAR-10 [20] in NAS-Bench-201 [9] with different numbers of clusters. We also report the peak memory usage in supernet training.

| Baseline | # of clusters | Kendall’s Tau | Peak Memory (MB) |
|-----------|---------------|--------------------|------------------|
| | 5 | 0.81±0.024 | 1901 |
| SPOS [12] | 25 | 0.813±0.017 | 2187 |
| | 125 | 0.781±0.022 | 2839 |
| | 625 | 0.752±0.027 | 6059 |

201. MS randomly selects a single edge from the supernet, and clusters the subnets according to operations for the edge. That is, we use five clusters, since each edge provides five candidate operations in the search space. In this experiment, to vary the number of clusters, we select 2 edges ($5^2 = 25$ clusters), 3 edges ($5^3 = 125$ clusters), and 4 edges ($5^4 = 625$ clusters) for clustering in MS. We can observe from the results that increasing the number of clusters from 5 to 25 yields a slight improvement in ranking consistency at the expense of more memory footprints, suggesting that 5 clusters (*i.e.*, selecting a single edge for clustering) is enough, in terms of the ranking consistency and memory, within our framework. We also see that choosing the number of clusters as 125 or 625 rather degrades the ranking consistency. This is because, with a substantial number of momentum clusters, each momentum buffer receives insufficient updates. The momentum buffer, storing the running mean of gradients to stabilize the training process, becomes less informative, which can negatively affect training supernet.

5. Limitations

Our framework can be applied to sampling-based NAS methods [2, 7, 12, 13, 16, 27, 31], but not to differentiable NAS methods [3, 5, 6, 24, 33]. This is because our framework considers the different characteristics of subnets sampled from the supernet, while differentiable NAS methods jointly optimize the supernet weights and operation importance coefficients, without explicitly considering the individual subnets. Despite this limitation, the growing interest in sampling-based methods over differentiable ones, due to their memory efficiency, highlights the importance of our work.

6. Conclusion

We have introduced a dynamic supernet training technique that considers the distinct characteristics of individual subnets for N -shot NAS. Specifically, we have proposed CaLR and MS to alleviate the unfairness and noisy momentum problems, respectively. We have demonstrated that our method can improve the search performance of various NAS methods, without bells and whistles. We believe that our method can be a strong baseline for future research on NAS.

Acknowledgements

This work was partly supported by IITP grant funded by the Korea government (MSIT) (No.RS-2022-00143524, Development of Fundamental Technology and Integrated Solution for Next-Generation Automatic Artificial Intelligence System, No.2022-0-00124, RS-2022-II220124, Development of Artificial Intelligence Technology for Self-Improving Competency-Aware Learning Capabilities) and the Yonsei Signature Research Cluster Program of 2024 (2024-22-0161).

References

- [1] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 5, 6
- [2] Minghao Chen, Jianlong Fu, and Haibin Ling. One-shot neural ensemble architecture search by diversity-guided search space shrinking. In *CVPR*, 2021. 2, 8
- [3] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019. 1, 2, 8
- [4] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of ImageNet as an alternative to the CIFAR datasets. *arXiv*, 2017. 5
- [5] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: Eliminating unfair advantages in differentiable architecture search. In *ECCV*, 2020. 1, 2, 8
- [6] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. DARTS-: Robustly stepping out of performance collapse without indicators. In *ICLR*, 2021. 2, 8
- [7] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search. In *ICCV*, 2021. 1, 2, 3, 5, 6, 7, 8
- [8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 2, 5, 6, 7
- [9] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020. 3, 5, 6, 7, 8
- [10] Yanjie Gao, Yu Liu, Hongyu Zhang, Zhengxian Li, Yonghao Zhu, Haoxiang Lin, and Mao Yang. Estimating gpu memory consumption of deep learning models. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020. 5, 6
- [11] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*. 1991. 1, 2, 5
- [12] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020. 1, 2, 3, 5, 6, 7, 8
- [13] Hyeonmin Ha, Ji-Hoon Kim, Semin Park, and Byung-Gon Chun. SUMNAS: Supernet with unbiased meta-features for neural architecture search. In *ICLR*, 2022. 1, 2, 5, 6, 8
- [14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NeurIPS Workshop*, 2014. 2
- [15] Shoukang Hu, Ruo Chen Wang, Lanqing Hong, Zhenguo Li, Cho-Jui Hsieh, and Jiashi Feng. Generalizing few-shot NAS with gradient matching. In *ICLR*, 2022. 1, 3
- [16] Yiming Hu, Yuding Liang, Zichao Guo, Ruosi Wan, Xiangyu Zhang, Yichen Wei, Qingyi Gu, and Jian Sun. Angle-based search space shrinking for neural architecture search. In *ECCV*, 2020. 2, 8
- [17] Tao Huang, Shan You, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. GreedyNASV2: Greedier search with a greedy path filter. In *CVPR*, 2022. 3
- [18] Maurice George Kendall. Rank correlation methods. 1948. 3, 6
- [19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019. 2, 5
- [20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2, 3, 5, 7, 8
- [21] Junghyup Lee and Bumsub Ham. AZ-NAS: Assembling zero-cost proxies for network architecture search. In *CVPR*, 2024. 5
- [22] Guihong Li, Yuedong Yang, Kartikeya Bhardwaj, and Radu Marculescu. ZICO: Zero-shot NAS via inverse coefficient of variation on gradients. In *ICLR*, 2023.
- [23] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-NAS: A zero-shot NAS for high-performance image recognition. In *ICCV*, 2021. 5
- [24] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019. 1, 2, 8
- [25] Shun Lu, Yu Hu, Longxing Yang, Zihao Sun, Jilin Mei, Jianchao Tan, and Chengru Song. PA&DA: Jointly sampling path and data for consistent NAS. In *CVPR*, 2023. 3, 5, 8
- [26] Youngmin Oh, Hyunju Lee, and Bumsub Ham. Efficient few-shot neural architecture search by counting the number of nonlinear functions. *AAAI*, 2025. 5
- [27] Houwen Peng, Hao Du, Hongyuan Yu, Qi Li, Jing Liao, and Jianlong Fu. Cream of the Crop: Distilling prioritized paths for one-shot neural architecture search. In *NeurIPS*, 2020. 2, 8
- [28] Jiefeng Peng, Jiqi Zhang, Changlin Li, Guangrun Wang, Xiaodan Liang, and Liang Lin. Pi-NAS: Improving neural architecture search by reducing supernet training consistency shift. In *ICCV*, 2021. 1, 2, 5
- [29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 1, 2
- [30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 5
- [31] Dilin Wang, Chengyue Gong, Meng Li, Qiang Liu, and Vikas Chandra. AlphaNet: Improved training of supernets with alpha-divergence. In *ICML*, 2021. 2, 8

- [32] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine learning*, 1992. [1](#), [2](#)
- [33] Peng Ye, Baopu Li, Yikang Li, Tao Chen, Jiayuan Fan, and Wanli Ouyang. β -DARTS: Beta-decay regularization for differentiable architecture search. In *CVPR*, 2022. [1](#), [2](#), [8](#)
- [34] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. GreedyNAS: Towards fast one-shot NAS with greedy supernet. In *CVPR*, 2020. [3](#)
- [35] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *ICLR*, 2020. [2](#)
- [36] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, and Steven Su. Overcoming multi-model forgetting in one-shot NAS with diversity maximization. In *CVPR*, 2020. [2](#)
- [37] Mingyang Zhang, Xinyi Yu, Haodong Zhao, and Linlin Ou. ShiftNAS: Improving one-shot NAS via probability shift. In *ICCV*, 2023. [3](#)
- [38] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *ICML*, 2021. [1](#), [2](#), [3](#), [5](#), [6](#), [7](#)
- [39] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. [1](#), [2](#)