Abstract

Weight sharing has become a de facto standard in neural architecture search because it enables the search to be done on commodity hardware. However, recent works have empirically shown a ranking disorder between the performance of stand-alone architectures and that of the corresponding shared-weight networks. This violates the main assumption of weight-sharing NAS algorithms, thus limiting their effectiveness. We tackle this issue by proposing a regularization term that aims to maximize the correlation between the performance rankings of the shared-weight network and that of the standalone architectures using a small set of landmark architectures. We incorporate our regularization term into three different NAS algorithms and show that it consistently improves performance across algorithms, search-spaces, and tasks.

1. Introduction

Modern algorithms for neural architecture search (NAS) can now find architectures that outperform the human-designed ones for many computer vision tasks [21, 42, 7, 34]. A driving factor behind this progress was the introduction of parameter sharing [27], which reduces the search time from thousands of GPU hours to just a few and has thus become the backbone of most state-of-the-art NAS frameworks [3, 49, 5, 23]. At the heart of all these methods is a shared network, a.k.a. super-net, that encompasses all architectures within the search space.

To train the super-net, NAS algorithms essentially sample individual architectures from the super-net and train them for one or a few steps. The sampling can be done explicitly, with strategies such as reinforcement learning [27, 6], evolutionary algorithms [15, 39], or random sampling [52, 18], or implicitly, by relying on a differentiable parameterization of the architecture space [19, 22, 5, 42, 53]. Whether explicit or implicit, the underlying assumption of these methods is that the relative performance of the individual architectures in the super-net is highly correlated with the performance of the same architectures when they are trained in a stand-alone fashion. If this were the case, one could then safely choose the best individual architecture from the super-net after the search and use it for evaluation. However, this assumption was disproved in [52, 54], who showed a correlation close to zero between the two rankings on complex search spaces [47, 22]. The major reason behind this is fairly intuitive: To be optimal, different individual architectures should have different parameter values, which they cannot because the parameters are shared. Super-net training will thus not produce the same results as stand-alone training. More importantly, there is no guarantee that even the relative ranking of the architectures will be maintained. While for simple, linear search spaces the ranking can be improved by using a carefully crafted sampling strategy [8, 9], addressing the ranking disorder for more realistic, complex search spaces remains an open problem [50].

In this paper, we propose to explicitly encourage archi-
tectures represented by the super-net to have a similar ranking to their counterparts trained in a stand-alone fashion. As illustrated by Figure 1, we leverage a set of landmark architectures, that is, architectures with known stand-alone performance, to define a regularization term that guides super-net training towards this goal. We show that a small set of landmark architectures suffices to significantly improve the global ranking correlation, so that the overall search procedure, including the independent training of the landmark architectures, remains tractable.

Our regularization term is general and does not make assumptions about the specific sampling algorithm used for super-net training. As such, it can easily be combined with many popular weight-sharing NAS algorithms. We demonstrate this by integrating it into three different algorithms [15, 24, 11] that are representative of three different categories of weight-sharing NAS algorithms: i) Algorithms that sample architectures from the super-net in an unbiased manner throughout the super-net training [18, 52, 2, 15, 8]; ii) approaches that employ learning-based samplers, which are updated during the training based on the performance of the partially-trained super-net [27, 19, 24, 40, 55]; and iii) algorithms that rely on differentiable architecture search [22, 6, 44, 25, 45].

Our extensive experiments on CIFAR-10 and ImageNet show that landmark regularization significantly reduces the ranking disorder that occurs in these algorithms and that they are consequently able to consistently find better-performing architectures. To further showcase the effectiveness and generality of our approach, we study its use in the context of architecture search for monocular depth estimation. To the best of our knowledge, this is the first attempt at performing NAS for this task. We, therefore, construct a dedicated search space and show that a landmark-regularized NAS algorithm can find novel architectures that improve upon the state of the art in this field.

2. Related work

Different from manual designing convolutional neural networks, which have been shown successful in many computer vision tasks [16, 14, 41, 51], neural architecture search (NAS) methods automate the design process and can be categorized into conventional approaches, that obtain architecture performance via stand-alone training [56, 57, 36, 40, 31, 32, 39], and weight sharing NAS, where the performance is obtained from one or a few super-nets that encompass all architectures within the search space [27, 24, 6, 55, 26, 48]. Motivated by the success of early NAS works, the literature has now branched into several research directions, such as using multi-objective optimization to discover architectures under resource constraints for mobile devices [37, 36, 42, 6, 15, 3], applying NAS to other computer vision tasks than image recognition [21, 7, 20, 34], and using knowledge distillation to eliminate the performance gap between super-net and stand-alone training for linear search spaces based on MobileNet [5, 49].

In contrast to the diversity of these research directions, super-net training in weight-sharing NAS has remained virtually unchanged since its first appearance in [18, 15, 2]. At its core, it consists of sampling one or few architectures at each training step, and updating the parameters encompassed by these architectures with a small batch of data. This approach has been challenged in many ways [4, 18, 52, 46], particular thanks to the introduction of the NASBench series [28, 47, 12, 35, 54, 10] of NAS benchmarks, which provide stand-alone performance of a substantial number or architectures and thus facilitate the analysis of the behavior of NAS algorithms. A critical issue that has been identified recently is the inability of most modern NAS algorithms to surpass simple random search under a fair comparison. In [52], this was traced back to the low ranking correlation between stand-alone performance and the corresponding super-net estimates. While recent works [15, 8] have shown that the ranking correlation is high on a MobileNet-based search space, where one only searches for the convolutional operations and the number of channels, others [50, 54] have revealed that the correlation remains low on cell-based NASNet-like search spaces [22, 24, 56], even when carefully tuning the design of the super-net.

In this paper, we introduce a simple, differentiable regularization term to improve the ranking correlation in weight-sharing NAS algorithms. We show that the regularization term can be used in a variety of weight-sharing NAS algorithms, and that it leads to a consistent improvement in terms of ranking correlation and final search performance.

Our regularization leverages the stand-alone performance of a few architectures. While some contemporary works also use ground-truth architecture performance, our approach differs fundamentally from theirs. Specifically, these methods aim to train a performance predictor, based on an auto-encoder in [23] or on a graphical neural network in [38], and are thus only applicable to weight-sharing NAS strategies that exploit such a performance predictor. By contrast, we add a regularizer to the super-net training loss, which allows our method to be applied to most weight-sharing NAS search strategies. Furthermore, our approach requires an order of magnitude fewer architectures with associated stand-alone performance; in our experiments, we use 30 instead of 300 in [23] and 1000 in [38].

3. Preliminaries

We first revisit the basics of super-net training and highlight the ranking disorder problem.

Let \( \Omega \) be a search space, defined as a set of \( N \) neural network architectures \( a_i, i \in [1, N] \). stand-alone training
optimizes the parameters $\theta_{a_j}$ of architecture $a_j$ independently from the other architectures by minimizing a loss function $\mathcal{L}(x, \theta_{a_j})$, thus yielding the optimal parameters $\theta^*_{a_j}$ for the given training data $x_{\text{train}}$. Without weight sharing, NAS then aims to train a search algorithm $S$ to sample architectures $S(\Omega) = \{a_k\}$ whose stand-alone performance outperforms that of other architectures, that is, $\mathcal{L}(x, \theta^*_{a_k}) < \mathcal{L}(x, \theta^*_{a_j}), \forall j \notin S(\Omega)$. In its simplest form, i.e., random search, there is no search algorithm to train per se, and one just samples a set of architectures, trains them in a stand-alone fashion, and ranks them to choose the ones with lowest loss.

With a proper search algorithm, however, NAS without weight sharing requires training and evaluating an impractically large amount of stand-alone architectures [56, 57, 39, 36, 37]. To circumvent this, weight-sharing NAS strategies construct a super-net $\theta^*$ that encompasses all architectures in the search space. The relative performance of individual architectures sampled from the super-net then acts as an estimate of their relative stand-alone performance. Training is typically formulated as minimizing a joint loss over all architectures that are represented by the super-net:

$$\mathcal{L}_s(\theta^*) = \sum_{i=1}^{N} \mathcal{L}(x, \theta_{a_i}^*),$$

where the optimization is made tractable by randomly sampling terms from this sum for each update.

In contrast to stand-alone training, parameters overlap between different architectures and in general we have that $\theta_{a_i}^* \cap \theta_{a_j}^* \neq \emptyset$. Since the parameters shared by $a_i$ and $a_j$ would typically not have the same optimal values in the stand-alone training [4], the optimal solution of super-net training is not the same as that of stand-alone training, and neither is the ranking of the architectures.

4. Landmark regularization

We address the issue of low ranking correlation by introducing a simple yet effective approach to regularize super-net training with prior knowledge about the relative performance of individual architectures. To this end, we sample $M << N$ architectures to form a set of landmark architectures, $\Omega_L = \{a_i\}_{i=1}^M$, and obtain their stand-alone performance $\mathcal{L}_{\text{valid}}(x_{\text{valid}}, \theta^*_{a_i})$ on validation data, where $\theta_{a_i}^* = \arg\min \mathcal{L}(x_{\text{train}}, \theta_{a_i})$. We shorten this to $\mathcal{L}_A(x, \theta_{a_i}^*)$ to simplify the notation. To ensure that the trained super-net is predictive of the performance of the stand-alone architectures, we aim to preserve the relative performance of these landmark architectures in the super-net. Formally, we want to enforce that

$$\mathcal{L}_A(x, \theta_{a_i}^*) \leq \mathcal{L}_A(x, \theta_{a_j}^*) \Rightarrow \mathcal{L}(x, \theta_{a_i}^*) \leq \mathcal{L}(x, \theta_{a_j}^*),$$

for all pairs of architectures $i, j \in [1, M]$.

To achieve this, we propose a differentiable regularization term that can readily be integrated into standard super-net training procedures. We first sort the landmark architectures in ascending order based on their ground-truth loss:

$$\forall i, j \in [1, M], \quad i < j \iff \mathcal{L}_A(x, \theta_{a_i}^*) \leq \mathcal{L}_A(x, \theta_{a_j}^*),$$

and use this ordering to define a regularization term

$$\mathcal{R}(\theta^*) = \sum_{i=1}^{M} \sum_{j=i+1}^{M} \max(0, \mathcal{L}(x, \theta_{a_i}^*) - \mathcal{L}(x, \theta_{a_j}^*)),$$

which penalizes deviations from the ordering induced by (2). Since all operations involved in the proposed regularizer are differentiable, it is straightforward to implement in existing deep learning frameworks. Note that the several alternative formulations of this loss are possible. We discuss them in the supplementary material.

The landmark-regularized training loss is then given by

$$\mathcal{L}(\theta^*) = \mathcal{L}_s(\theta^*) + \lambda \mathcal{R}(\theta^*),$$

\[ \text{Figure 2: Overview of our approach. Left: During the search phase, we first sample a set of landmark architectures and obtain their stand-alone performance. We train the super-net with our regularization term such that the landmark ranking is preserved. Right: After a round of training, we sample the best architectures given the current super-net performance and evaluate their stand-alone performance. We add these architectures to the set of landmarks and repeat the process for a few iterations.} \]
where $\lambda > 0$ is a hyper-parameter. To avoid leakage of validation data into the super-net training, we follow [22, 24, 6] and split the training set into two parts. We use one part to evaluate the super-net loss $L_s$ and the other part to evaluate the regularization $R$ during training.

**Computational cost.** The complexity of evaluating the regularizer discussed above is $O(M^2)$. This factor can have a significant impact on the training time, as the regularizer has to be evaluated at every training iteration. To reduce this computational burden while still encouraging the super-net to encode a correct architecture ranking, we propose to randomly sample $m$ pairs of landmark architectures $i, j$ at each iteration and evaluate their ranking:

$$R(\theta^a) = \sum_{i,j} \max(0, L(x, \theta^a_i) - L(x, \theta^a_j)).$$

(6)

This reduces the time complexity from $O(M^2)$ to $O(m)$. We will show empirically that even evaluation with a single pair introduces virtually no degradation of the resulting architecture ranking.

**Landmark selection.** The choice of the landmark architectures has an impact on the effectiveness of our regularizer. In particular, we would like to use landmarks that cover the complete search space. To promote this, we introduce the diverse landmark sampling strategy described by Algorithm 1. We start by randomly sampling a root architecture from the search space. We then generate $M - 1$ diverse architectures by mutating the root architecture such that the Hamming distance is larger than a threshold $\tau$. For example, in the DARTS search space [22], one architecture is encoded as a sequence, where each element represents selecting either a previous node or an operation. Mutating an architecture is then achieved by randomly altering one element, and the Hamming distance between two architectures is computed as the number of unequal elements.

**Regularization schedule.** For all practical applications, the number of landmark architectures will be several orders of magnitude smaller than the total number of architectures in the search space. The regularization term thus needs to have high weight to be effective and have a noticeable effect on the training. However, too much regularization can negatively impact the training dynamics, especially in the early stages. To alleviate this issue, we propose to enable regularization after a warm-up phase and to gradually increase its influence using a cosine schedule. Specifically, we set the regularization weight at epoch $t$ to

$$\lambda_t = 1_{t > t_w} \cdot \frac{1}{2} \left(1 + \cos \frac{\pi (t - t_w)}{t_{total}}\right) \lambda_{max},$$

(7)

where $t_w$ denotes the number of warm-up epochs, $t_{total}$ denotes the total number of epochs, and $\lambda_{max}$ denotes the final value for the regularization parameter.

---

**Algorithm 1: Landmark-regularized training.**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Search space $\Omega$, NAS algorithm $S$, super-net and stand-alone losses $L_s, L_A$, distance threshold $\tau$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Model $a_t \leftarrow \arg\min_{a \in \Omega_t} L_{GT}(x, \theta_a)$</td>
</tr>
</tbody>
</table>

**Application to existing NAS methods.** The proposed regularization term is independent of the search algorithm, and thus widely applicable to many different weight-sharing NAS algorithms. We discuss its use in three different classes of NAS strategies, specific instances of which will act as baselines in our experiments.

We categorize weight sharing NAS algorithms into three broad categories according to their interaction with the super-net: i) unbiased architecture sampling algorithms [18, 52, 2, 15, 8] that sample one or a few paths uniformly at random, ii) learning based sampling that favors the most promising architectures given the performance of the current, partially trained super-net [27, 19, 24, 40, 55], and iii) differentiable architecture search that parameterizes the architecture sampling probability as part of the super-net [22, 6, 44, 25, 45].

For the first two categories, our method can be directly incorporated into the super-net training to improve its quality, and hence to improve the final search results. For algorithms in the last category, the algorithm is usually composed of two distinct phases that are executed alternatingly. In the first phase the parameters that define the architecture are fixed and only the weights are updated, whereas in the second phase the weights are fixed and the architecture parameters are updated. Our regularization term can directly be integrated into the first phase when using discrete architectures in the forward pass as in works that employ the Gumbel-Softmax [11, 42] or binary gates [6, 44]. When
using continuous architecture specifications in the forward pass [22, 45], we do not have a discrete sub-path to sample and evaluate the regularization term, so additional care has to be taken to incorporate it. We discuss various ways to do this in supplementary material.

Multi-iteration pipeline. Figure 2 depicts the complete training pipeline. We first sample landmark architectures using our landmark selection strategy and obtain their stand-alone performance. We then train the NAS algorithm with landmark regularization. After a round of training, we sample the top $M$ architectures using the trained NAS algorithm, obtain their stand-alone performances, and add these architectures to the set of landmarks. We proceed training of the super-net with the expanded set of landmarks and iterate this process.

In our experiments on different tasks and algorithms, we observed a stable improvement after sampling 3 sets of $M = 10$ architectures for a total of 30 landmarks, which is computationally feasible. Additionally, our algorithm can leverage previously trained models to improve the search by simply adding them to the landmark set. Considering that search spaces usually encompass billions of architectures, the number of landmarks is negligible.

5. Experiments

To validate the landmark regularization we incorporate it into three popular weight-sharing algorithms and evaluate them on the task of image classification using the CIFAR-10 [17] and ImageNet [33] datasets. We then discuss a new search space for monocular depth estimation architectures and show that our approach also applies to this new task. Finally, we ablate the key components and hyperparameter choices of our landmark regularization.

We used PyTorch for all our experiments and follow the evaluation framework defined in [52] to ensure a fair comparison with the baseline methods. Following [28, 29], we shorten the training time from 600 to 100 epochs on CIFAR-10 and from 250 to 50 on ImageNet, which still yields a good prediction quality. We release our code at https://github.com/kcyu2014/nas-landmarkreg.

Baselines. We select single-path one-shot (SPOS) as a representative unbiased architecture sampling algorithm. We use SPOS to train the super-net, followed by an evolutionary search to select the best models based on the super-net performance [15]. Among the learning-based architecture sampling methods, we select neural architecture optimization (NAO) [24], which trains an explicit auto-encoder-based performance predictor. Finally, for differentiable architecture search, we select the gradient-based search using a differentiable architecture sampler (GDAS) [11], which has been widely used in other works [44, 42, 6, 19]. See the supplementary material for more details and hyperparameter settings of the baseline algorithms.

Hyperparameters. We sample $M = 10$ landmark architectures at each iteration, and perform $T = 3$ iterations. We sample $m = 1$ pairs for each training step and set $\lambda_{max} = 10$ in all of our experiments unless otherwise specified. The Hamming distance threshold $\tau$ is set according to the configuration of each search space. We train the baselines for the same total number of epochs, to ensure that any performance improvement cannot be attributed to our approach sampling more architectures.

Metrics. We follow [15, 50] and report the ranking correlation in terms of the sparse Kendall-Tau (S-KdT). We sample 200 architectures randomly to compute this metric for the CIFAR-10 experiments, 90 for the ImageNet experiments, and 20 for the monocular depth estimation experiments. Note that we exclude the landmark architectures from this set to avoid reporting overly optimistic numbers for our approach. Furthermore, following [52, 12], we report the mean and best stand-alone performance of the best architectures found over three independent runs.

5.1. Image classification on CIFAR-10

Since the inception of NAS, CIFAR-10 has acted as one of the main datasets to benchmark NAS performance [57, 27, 24, 15, 45, 48]. We utilize two search spaces, NASBench-101 and NASBench-201, for which the stand-alone performance of many architectures is known.

<table>
<thead>
<tr>
<th>Model</th>
<th>S-KdT</th>
<th>Mean Acc.</th>
<th>Best Rank</th>
<th>Best Acc.</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPOS</td>
<td>0.267 ± 0.02</td>
<td>91.02 ± 0.52</td>
<td>38953</td>
<td>92.82</td>
<td>11.89</td>
</tr>
<tr>
<td>SPOS+Ours</td>
<td>0.347 ± 0.03</td>
<td>92.48 ± 0.51</td>
<td>27697</td>
<td>92.99</td>
<td>14.92</td>
</tr>
<tr>
<td>NAO</td>
<td>0.329 ± 0.11</td>
<td>90.56 ± 0.88</td>
<td>131969</td>
<td>91.60</td>
<td>15.46</td>
</tr>
<tr>
<td>NAO+Ours</td>
<td>0.457 ± 0.03</td>
<td>92.32 ± 1.32</td>
<td>9513</td>
<td>93.34</td>
<td>21.04</td>
</tr>
</tbody>
</table>

Table 1: Results on NASBench-101 and NASBench-201. We report the S-KdT at the end of training, the mean stand-alone accuracy of the searched architectures, the best rank, and the best accuracy. Each method was run 3 times. We also report the search cost in hours on Tesla-V100 (32Gb).
matrix-based configuration used in NASBench-101 is not amenable to differentiable approaches \cite{54,47}. We set the Hamming distance threshold to $\tau = 5$.

As shown in Table 1 (top), landmark regularization improves the ranking correlation (S-KdT) from 0.267 to 0.347 for SPOS, and from 0.329 to 0.457 for NAO. This translates to a 1-2% improvement in terms of mean stand-alone accuracy over three runs. The best architecture discovered on this search space, thanks to our regularizer, ranks 9313-th which corresponds to the top 2% of architectures. Note that NAO without landmark regularization consistently got trapped in local minima, leading to its best architecture being only in the top 30%.

**NASBench-201.** We report results on the NASBench-201 cell search space \cite{12}, where each cell is a fully connected graph with 4 nodes. Each edge contains 5 searchable operations. This yields a total of 15,625 architectures. We set $\tau = 5$ and benchmark all three baseline algorithms on this space. As shown in Table 1 (bottom), landmark regularization consistently improves the ranking correlation (S-KdT) across all three methods. We also observe an improvement in mean accuracy of more than 4% with SPOS. The best architecture is obtained by GDAS with landmark regularization and ranks 109-th. This corresponds to the top 0.7% architectures across the search space. In Figure 3 (top), we plot the mean S-KdT as super-net training progresses. We can see that the regularization improves the ranking correlation by a significant margin, especially towards the end of training. Note that we extend our experiments to CIFAR-100 and ImageNet16-120 of NASBench-201 in supplementary material.

**DARTS search space.** The NASBench search spaces are relatively small. To evaluate our approach in a more realistic scenario, we make use of the DARTS search space \cite{22,44,45,25} which spans $3.3 \times 10^{13}$ architectures and is commonly used to evaluate real-world NAS performance. In contrast to NASBench, for which we could query the existing stand-alone performances, here, we need to train the discovered architecture from scratch. To compute the ranking correlation, we rely on the 5,000 pre-trained models of \cite{28} from which we randomly sample 200 as before.

We report our results in Table 2. We observe a clear improvement in terms of both S-KdT and mean accuracy over three independent searches across all three algorithms. Interestingly, the best model obtained using our ranking loss can surpass the baseline models by almost 1% with only around 50% of the parameters. The best model from GDAS with our regularizer surpasses the state-of-the-art model on this space \cite{45} by 0.58% with 30% fewer parameters.

**5.2. Image classification on ImageNet**

To further evidence the effectiveness of our method, we move to ImageNet classification. For evaluation, we pick the best model of three independent runs predicted by each NAS algorithm and train them from scratch on the entire ImageNet training set for 50 epochs. We follow the setup of \cite{45} and use stochastic gradient descent with a linear learning rate scheduler, which linearly increases from 0.1 to 0.5 in the first five epochs, and decreases to 0 over the remaining 45 epochs. We use a weight decay of $3e-4$ and a label smoothing coefficient of 0.1 for all models. For this dataset, we use the popular DARTS search space, which provides 120 architectures trained for 50 epochs \cite{28}. We split these architectures into 90 to report the sparse Kendall-Tau evaluation metric, and 30 for landmark sampling. We

![Figure 3: Evolution of the S-KdT of three NAS algorithms on two search spaces. Landmark regularization significantly improves the ranking correlation of the super-net in all cases.](image-url)

### Table 2: Results on the DARTS search space on CIFAR-10. Our best model (GDAS+Ours) surpasses the state-of-the-art model of \cite{45} (94.02% accuracy with 3.62M parameters) with 30% fewer parameters.

<table>
<thead>
<tr>
<th>Model</th>
<th>S-KdT</th>
<th>Mean Acc.</th>
<th>Params</th>
<th>Best Acc.</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPOS</td>
<td>0.058 ± 0.010</td>
<td>92.80 ± 0.03</td>
<td>5.082M</td>
<td>92.88</td>
<td>12.53</td>
</tr>
<tr>
<td>SPOS+Ours</td>
<td>0.206 ± 0.018</td>
<td>93.41 ± 0.43</td>
<td>2.181M</td>
<td>93.84</td>
<td>15.96</td>
</tr>
<tr>
<td>GDAS</td>
<td>0.176 ± 0.014</td>
<td>90.48 ± 2.95</td>
<td>3.418M</td>
<td>93.43</td>
<td>9.27</td>
</tr>
<tr>
<td>GDAS+Ours</td>
<td>0.209 ± 0.001</td>
<td>94.32 ± 0.28</td>
<td>2.540M</td>
<td>94.60</td>
<td>13.24</td>
</tr>
<tr>
<td>NAO</td>
<td>0.102 ± 0.018</td>
<td>92.93 ± 0.87</td>
<td>5.080M</td>
<td>93.03</td>
<td>19.72</td>
</tr>
<tr>
<td>NAO+Ours</td>
<td>0.231 ± 0.012</td>
<td>93.53 ± 0.43</td>
<td>2.184M</td>
<td>93.78</td>
<td>28.18</td>
</tr>
</tbody>
</table>

### Table 3: Results on ImageNet. We report mean top-1 accuracy over 3 runs after 50 epochs and best top-1 accuracy after 50 and 250 epochs, respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>S-KdT</th>
<th>Mean Top-1</th>
<th>Params</th>
<th>Best Top-1 (50/250)</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPOS</td>
<td>0.210 ± 0.010</td>
<td>64.57 ± 3.30</td>
<td>4.579M</td>
<td>67.88 / 73.69</td>
<td>7.29</td>
</tr>
<tr>
<td>SPOS+Ours</td>
<td>0.267 ± 0.018</td>
<td>67.38 ± 0.92</td>
<td>4.766M</td>
<td>68.61 / 74.58</td>
<td>9.39</td>
</tr>
<tr>
<td>GDAS</td>
<td>0.247 ± 0.012</td>
<td>67.50 ± 0.26</td>
<td>5.076M</td>
<td>67.26 / 74.03</td>
<td>9.13</td>
</tr>
<tr>
<td>GDAS+Ours</td>
<td>0.272 ± 0.023</td>
<td>68.82 ± 0.24</td>
<td>5.073M</td>
<td>68.36 / 74.82</td>
<td>10.39</td>
</tr>
<tr>
<td>NAO</td>
<td>0.253 ± 0.006</td>
<td>67.70 ± 0.43</td>
<td>4.675M</td>
<td>68.21 / 73.71</td>
<td>8.39</td>
</tr>
<tr>
<td>NAO+Ours</td>
<td>0.279 ± 0.003</td>
<td>68.89 ± 0.58</td>
<td>4.488M</td>
<td>69.58 / 74.92</td>
<td>11.21</td>
</tr>
</tbody>
</table>
Figure 4: Monocular depth estimation search space. (a) We modify MiDaS [30] to construct the search space. We keep the backbone unchanged and search for fusion blocks in the decoder branches. (b) To define a fusion block, we model the input from the backbone and from the preceding fusion block as two nodes. We add two feature nodes, which sum up all previous inputs. The nodes are connected by edges, which represent the searchable operations. The final output node $Y$ takes the output of the last feature node and applies a potentially searchable upsampling operation. (c) Each edge, except the output edge, represents an Edge Op (blue) that contains five operations to choose from, while an upsampling edge (purple) contains four. (d) We propose four sub-space configurations: ‘V1’ indicates that we only search edge operations and fix the upsampling operator to bilinear upsampling. ‘V2’ includes a search over the upsampling operators. ‘Sync’ indicates that all fusion blocks share the same configuration, while ‘Non-sync’ allows them to differ.

5.3. Monocular depth estimation

To showcase the generality of our approach, we apply our landmark-regularized NAS to the task of monocular depth estimation. Monocular depth estimation aims to predict pixel-wise depth from a single RGB image. Different paradigms have emerged on how to train single-image depth predictors, ranging from fully supervised training [1, 43, 30], to self-supervised approaches [13]. We follow the supervised paradigm and use the loss function proposed by Ranftl et al. [30] to search for an architecture on the RedWeb [43] dataset.

Search space. Figure 4 gives a detailed overview of the structure of our search space. Figure 4 (a) shows the structure of a traditional monocular depth estimation network. It is composed of a backbone network that acts as a feature extractor, typically pre-trained on ImageNet, followed by decoder fusion blocks that aggregate multi-scale information into a final prediction. Since using a pre-trained high-capacity network has been shown to be of high importance for final performance [1, 30], its architecture is fixed. We thus propose to search for the fusion blocks that define the decoder.

As depicted in Figure 4 (b), each fusion block is a search cell and takes the output of its preceding fusion block and the features from the backbone network as input. As the first fusion block does not have a predecessor, we simply duplicate the features from the encoder. Figure 4 (c) shows the possible searchable operations, whereas (d) illustrates four possible configurations of our search space. We report the total number of architectures [5] for each search space in Table 4. While the search space is relatively simple, it is large enough to exhibit the problem of ranking disorder.

Results. We use the single-path one-shot algorithm to benchmark the influence of our regularization term on this task. We run $T = 2$ iterations, and sample 10 architectures in the first iteration. After the first round of training, we pick the three top models, obtain their stand-alone performance, and add them to the set of landmarks before we perform training for a second iteration. In addition to S-KdT, we report the scale- and shift-invariant loss [30] on the validation set as the performance metric.

Our results in Table 4 indicate that our method consistently yields an improvement in terms of S-KdT for all four configurations of the search space. Our best model
Table 5: Influence of the regularization parameter $\lambda$. Left: Different schedules (c.f. top plots) to modify the strength of regularization throughout training. Right: Influence of $\lambda_{\text{max}}$ with the increasing cosine schedule.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>$\lambda_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Const</td>
<td>Cos $\downarrow$</td>
</tr>
<tr>
<td>S-KdT</td>
<td>0.753</td>
</tr>
<tr>
<td>Mean Acc</td>
<td>91.13</td>
</tr>
</tbody>
</table>

Table 6: Influence of the number of iterations $T$ on NASBench-201.

<table>
<thead>
<tr>
<th>m Pair(s)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-KdT</td>
<td>0.771</td>
<td>0.803</td>
<td>0.801</td>
<td>0.805</td>
<td>0.812</td>
<td>0.807</td>
<td>0.791</td>
</tr>
<tr>
<td>Mean Acc</td>
<td>87.66</td>
<td>92.08</td>
<td>92.12</td>
<td>92.13</td>
<td>92.10</td>
<td>92.11</td>
<td>92.00</td>
</tr>
</tbody>
</table>

Table 7: Influence of the stochastic approximation of Eq. 4. We randomly sample $m$ pairs from 30 landmarks during each training step.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>T=1</th>
<th>T=3</th>
<th>T=10</th>
<th>T=1</th>
<th>T=3</th>
<th>T=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-KdT</td>
<td>0.763</td>
<td>0.771</td>
<td>0.758</td>
<td>0.760</td>
<td>0.802</td>
<td>0.811</td>
</tr>
<tr>
<td>Mean Acc</td>
<td>91.13</td>
<td>91.48</td>
<td>91.78</td>
<td>91.29</td>
<td>92.08</td>
<td>92.17</td>
</tr>
</tbody>
</table>

We have presented a simple yet effective approach to leverage a few landmark architectures to guide the super-net training of weight-sharing NAS algorithms towards a better correlation with stand-alone performance. Our strategy is applicable to most NAS algorithms and our experiments have shown that it consistently improves both the ranking correlation between the super-net and stand-alone performance as well as the final performance across three different search algorithms and three different tasks. Additionally, our approach can leverage the information from previously trained stand-alone models to improve NAS performance. In the future, we will focus on developing a more advanced landmark sampling strategy.
References


[8] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. *arXiv:*, 2019. 1, 2, 4


[18] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *UAI*, 2019. 1, 2, 4

[19] Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot NAS by suppressing the posterior fading. *CVPR*, 2020. 1, 2, 4, 5


[27] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018. 1, 2, 4, 5


Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Neural architecture search by learning action space for monte carlo tree search. AAAI, 2020.


Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. ICLR, 2019.

