ROME: Robustifying Memory-Efficient NAS via Topology Disentanglement and Gradient Accumulation

Xiaoxing Wang*[1], Xiangxiang Chu*[2], Yuda Fan[1], Zhexi Zhang[1], Bo Zhang[2], Xiaokang Yang[1], and Junchi Yan†[1]

[1]Dep. of Computer Science and Engineering & MoE Key Lab of AI, Shanghai Jiao Tong University
[2]Meituan

Abstract

Albeit being a prevalent architecture searching approach, differentiable architecture search (DARTS) is largely hindered by its substantial memory cost since the entire supernet resides in the memory. This is where the single-path DARTS comes in, which only chooses a single-path submodel at each step. While being memory-friendly, it also comes with low computational costs. Nonetheless, we discover a critical issue of single-path DARTS that has ***not*** *been primarily noticed. Namely, it also suffers from severe* performance collapse *since too many parameter-free operations like skip connections are derived, just like DARTS does. In this paper, we propose a new algorithm called RObustifying Memory-Efficient NAS (ROME) to give a cure. First, we disentangle the topology search from the operation search to make searching and evaluation consistent. We then adopt Gumbel-Top2 reparameterization and gradient accumulation to robustify the unwieldy bi-level optimization. We verify ROME extensively across 15 benchmarks to demonstrate its effectiveness and robustness.*

1. Introduction

Despite the fast development of neural architecture search (NAS) [52] to aid network design in vision tasks like classification [32, 6, 41, 42], object detection [11, 36], and segmentation [22], there has been an urging demand for faster searching algorithms. Early methods based on the evaluation of a huge number of candidate models [52, 31, 16] require unaffordable costs (typically 2k GPU days). In the light of weight-sharing mechanism introduced in SMASH [2], a variety of low-cost approaches have emerged [1, 27, 24]. DARTS [24] has taken the dominance with a myriad of follow-up works [38, 3, 39, 10, 5, 47]. In this paper, we investigate a single-path based variation of DARTS, typically GDAS [10], for its fast speed and low GPU memory.

Unlike many DARTS variants that have to perform all candidate operations, single-path methods [30, 10, 39], also termed as memory-efficient NAS[1], are developed to sample and activate only a subset of operations in the supernet. For differentiable search, Gumbel-Softmax reparameterization tricks [17, 25] are generally employed [38, 10, 39].

In this paper, we show that single-path methods also suffer from severe *performance collapse* where many parameterless operations accumulate, akin to that of DARTS as broadly discussed in [48, 4, 39, 8, 20]. We propose a robust algorithm called ROME to resolve this issue.

We observe that single-path methods usually intertwine topology search with operation search, which creates inconsistency between searching and evaluation. We first disentangle the two from each other. Specifically, in addition to the existing architectural parameters (α) that represent the significance of each operation, we involve topology parameters (β) to denote the relative importance of edges. A single-path architecture can then be induced by both α and β. Moreover, to further robustify the searching process, we propose a *gradient accumulation* strategy during the bi-level optimization. We sketch the framework of ROME in Fig. 1. In a nutshell, our contributions are,

1) Revealing performance collapse in single-path differentiable NAS. Similar to the performance collapse problem in DARTS, the architectures searched by single-path based methods can also be dominated by parameterless operations, especially skip connections. However, this issue hasn't been deeply explored, which motivates us to propose a new robust, lower memory cost and search cost method.

2) Consistent search and evaluation by disentangling topology search from operation selection. We introduce

*Equal contribution, † Correspondence author.

[1]We interchangeably use the term 'single-path' and 'memory-efficient'.

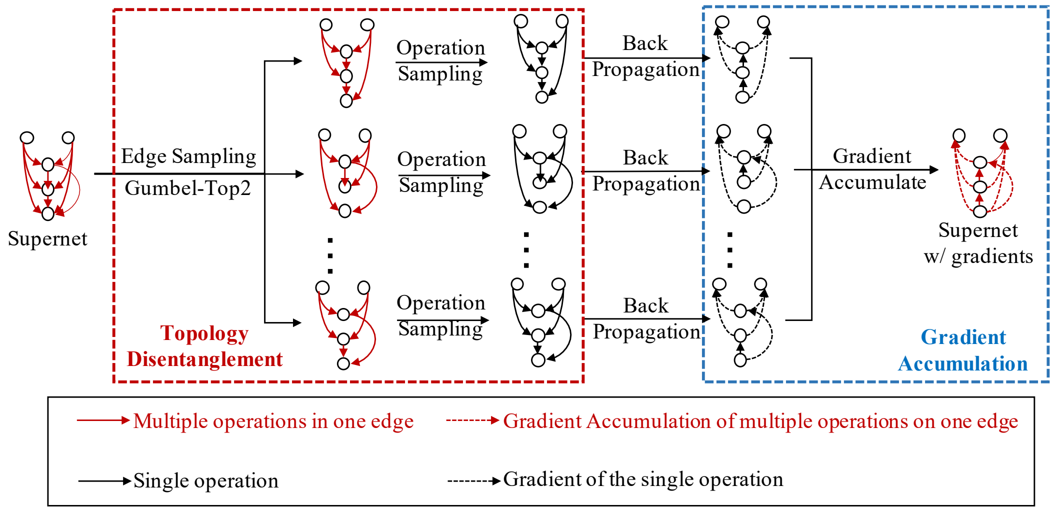


Figure 1: ROME (v2): Gumbel-Top2 is devised to sample edges to satisfy the restriction that each node has in-degree 2. Subnetworks apply forward and backward independently. Gradients are accumulated to update the supernet weights at once.

independent topology parameters to unwind topology from operations, which avoids structure inconsistency between the search and evaluation stage. We further devise Gumbel-Top2 re-parameterization to make our method differentiable and provide its theoretical validity. To our best knowledge, this is the first work to achieve consistent search and evaluation for single-path differentiable NAS.

3) Robustifying bi-level optimization via gradient accumulation. We devise two gradient accumulation techniques to address the aforementioned issue. One helps fair training for each candidate operations. The other instead reduces the estimation variance on architectural weights.

4) Strong performance while maintaining low memory cost. Tested on the popular settings[2] for NAS [48, 21, 6], our approach achieves state-of-the-art on various search spaces and datasets across **15** benchmarks. Our approach is fast, robust, and memory efficient. Compared with GDAS and PC-DARTS, our approach costs **26%** and **38%** lower memory in the standard search space of DARTS respectively. The source code will be made publicly available.

2. Related Work

Differentiable Neural Architecture Search. Similar to [53, 27] that uses a directed acyclic graph to represent a cell, DARTS [24] constructs a cell-based supernet and further introduces architectural weights to represent the importance of each operation. DARTS proposes two types (first-order and second-order) of approximation that alternatively update operation parameters and architectural weights with stochastic gradient descent. However, since the supernet subsumes all connections and operations within the search space, DARTS risks exhausting GPU memory. A possible attempt to resolve this issue is done by progressively pruning operations [5], which is still an indirect approach and requires strong regularization tricks. Apart from that, recent works [48, 4] also point out an instability phenomenon of DARTS. These issues significantly restrict its application.

Memory-efficient NAS. To reduce GPU memory cost, several prior works have revised the forward procedure of the supernet. PC-DARTS [40] makes use of partial connections instead of the full-fledged supernet. Some works [37, 36, 35] proposes to merge all parametric operations into one convolution, a similar super-kernel strategy is also used in single-path NAS [30]. ProxylessNAS [3] samples two operations on each edge during the search process, which enables proxyless searching on large datasets. Single-path supernets like SPOS [13] and FairNAS [7] sample only a single path at each iteration, however, both require an additional searching stage to choose the final models. Single-path differentiable methods like GDAS [10] sample a subgraph of the DAG at each iteration, which is by far the most efficient. However, we observe that a severe instability issue occurs, which has been previously neglected.

Performance collapse of DARTS. The collapse issue is one of the most critical problems in differentiable architecture search [20, 40, 8, 4, 48]. It has been shown that DARTS [24] prefers to choose parameterless operations, leading to its performance collapse [20, 8]. Recent works [48, 4] utilize the eigenvalue of the Hessian matrix as an indicator of collapse and design various techniques to regularize high eigenvalues. Instead, other works [20, 5] directly constrain the number of skip connections as 2 to avoid collapse. Nonetheless, the previous methods are specifically designed for the full-path training scheme. Whereas the collapse problem in single-path has been rarely studied.

[2]Independently search under different random seeds instead of multiple training of a single best-searched model.

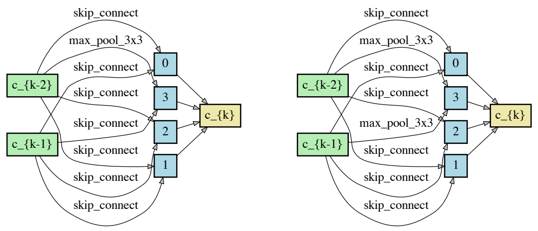


Figure 2: Two failure examples of GDAS [10] in the DARTS search space in our experiment by running the authors' code, where normal cells are full of parameterless operations. The average accuracy is 96.52% on CIFAR-10, while models searched by our method can achieve 97.42%.

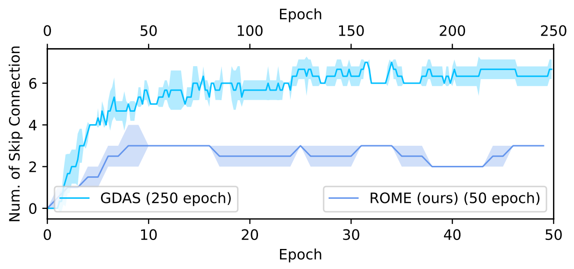


Figure 3: Evolution of the number of skip connections in a normal cell of GDAS vs. ROME.

3. Methodology

3.1. Collapse in Single-Path Differentiable NAS

DARTS [24] optimizes a supernet stacked by normal cells and reduction cells. A cell owns N nodes $\{x_i\}_{i=1}^{N}$ denoting latent representation. Edge $e_{i,j}$ from node x_i to x_j integrates all candidate operations $\mathcal{O}$ whose importance is represented by architecture parameter $\alpha_{i,j}^{o}$. Since the weights of all operations are involved in the forward and backward process, DARTS is very memory-consuming.

To reduce memory cost, GDAS [10] proposes to sample a sub-set of operations at each iteration. For edge $e_{i,j}$, a one-hot random vector $z_{i,j} \in \{0,1\}^{|\mathcal{O}|}$ is sampled, indicating only one candidate operation is selected during the forward pass and back-propagation.

However, we observe that the normal cell learned by GDAS has 4 skip connections, and GDAS (FRC) even contains 5 skip connections, implying performance collapse issue also exists in single-path based methods.

We rerun the released code of GDAS [10] for several times and observe that the normal cells are dominated by skip connections and max pooling, shown in Fig. 2. We also draw the evolution of skip connections of GDAS in Fig. 3.

3.2. Possible Reasons of Performance Collapse

We conjecture that the following two factors contribute most to the collapse issue, which motivates us to provide a remedy in each regard.

Inconsistency between the searching and evaluation stage. Structural inconsistency between the supernet and the final network mainly appears at the operation level and the topology level. Operation-level inconsistency, *i.e.*, searching with many operations but evaluating only with the most significant one, has been alleviated in recent single-path methods [10, 39] by sampling one operation on each edge at each iteration in the search phase. However, topology-level inconsistency has long been neglected. Specifically, the nodes in the supernet connect with all its predecessors, while the nodes in the final network must only have in-degree 2. In this paper, we eliminate such inconsistency by disentangling topology and operation search.

Insufficient sampling for candidate operations. The instability issue for single-bath methods can be largely attributed to its stochastic nature that involves sub-sampling. Specifically, at each iteration, only one operation is sampled for each edge, resulting in an insufficient training of weights $\boldsymbol{\theta}$. It also causes high variance for the gradients of $\boldsymbol{\alpha}$, hence influencing the searching convergence. To this end, we propose *multiple sampling* and *gradient accumulation* to train the supernet and reduce the gradient variance of architectural weights.

Based on the above reasoning, we propose topology disentanglement (Sec 3.3) to resolve inconsistency and gradient accumulation (Sec 3.6) to rectify the instability caused by insufficient sampling. Fig. 3 illustrate the number of skip connections in a normal cell searched by GDAS and our method (ROME), showing that our method can effectively alleviate the collapse issue.

3.3. Topology Disentanglement

To disentangle the search for topology and operations on each edge, we use an indicator $\boldsymbol{B}_{i,j} \in \{0,1\}$ to denote whether edge $e_{i,j}$ is selected, and $\boldsymbol{A}_{i,j}^{o} \in \{0,1\}$ for whether operation o on edge $e_{i,j}$ is selected. Sampling architecture z with M connections can be decomposed into two parts: sample M edges first, and their operations second.

Sampling for edges. Topology inconsistency exists in single-path based methods [10, 39], as all 14 edges in a cell are selected in the search stage but the final architecture only has 8 edges. To address this issue, we propose to sample the same number of edges in search.

Each intermediate node should connect with exact two predecessors, satisfying DARTS's constraints. Formally, we use $\boldsymbol{B}_{i,j}$ to indicate whether the edge $e_{i,j}$ between node

x_i and x_j is sampled, and we enforce,

$$\sum_{i<j} \boldsymbol{B}_{i,j} = 2, \quad \forall j. \tag{1}$$

We give two techniques of edge sampling in Sec 3.4.

Sampling for operations. We use $\boldsymbol{A}^o_{i,j}$ to denote whether the operator o is sampled on the edge $e_{i,j}$, and we adopt Gumbel-Max technique to sample operations, where $\boldsymbol{g}^o_{i,j}$ is sampled from Gumbel$(0,1)$ distribution[3], and $\tilde{\boldsymbol{\alpha}}^o_{i,j} = \frac{\exp(\boldsymbol{\alpha}^o_{i,j})}{\sum_{o' \in \mathcal{O}} \exp(\boldsymbol{\alpha}^{o'}_{i,j})}$ is the normalized architectural weights:

$$\boldsymbol{A}_{i,j} = \text{one\_hot}\left[\arg\max_{o \in \mathcal{O}} (\log \tilde{\boldsymbol{\alpha}}^o_{i,j} + \boldsymbol{g}^o_{i,j})\right] \in \mathbb{R}^{|\mathcal{O}|}, \tag{2}$$

To make the objective function differentiable to architectural weights $\boldsymbol{\alpha}$, we relax the discrete distribution to a continuous one by Gumbel-Softmax:

$$\tilde{\boldsymbol{A}}^o_{i,j} = \frac{\exp\left[(\log \tilde{\boldsymbol{\alpha}}^o_{i,j} + \boldsymbol{g}^o_{i,j})/\tau\right]}{\sum_{o'=1}^{|\mathcal{O}|} \exp\left[(\log \tilde{\boldsymbol{\alpha}}^{o'}_{i,j} + \boldsymbol{g}^{o'}_{i,j})/\tau\right]},$$

$$\boldsymbol{A}_{i,j} = \text{one\_hot}\left[\arg\max_{o \in \mathcal{O}} \tilde{\boldsymbol{A}}^o_{i,j}\right], \tag{3}$$

where the temperature τ gradually decreases in search.

3.4. From Gumbel-Max to Gumbel-Top2 Reparameterization

We propose two variations of edge sampling techniques, *i.e.* Gumbel-Max and Gumbel-Top2, based on which we derive two versions of ROME (v1 and v2).

3.4.1 Gumbel-Max (ROME-v1).

Suppose node x_j has j possible predecessors, there are $\frac{j \times (j-1)}{2}$ types of edge choices. We use $\boldsymbol{I}^{(i,k)}_j (i < k < j)$ to indicate whether node x_j is connected both to x_i and x_k, i.e. when $\boldsymbol{I}^{(i,k)}_j = 1$, we have $\boldsymbol{B}_{i,j} = \boldsymbol{B}_{k,j} = 1$ and $\boldsymbol{B}_{m,j} = 0 (\forall m < j, m \neq i, j)$.

We then set a trainable variable $\boldsymbol{\beta}^{(i,k)}_j$ to denote the importance of each edge choice for node x_j, such that

$$p\left(\boldsymbol{I}^{(i,k)}_j = 1\right) = \frac{\exp(\boldsymbol{\beta}^{(i,k)}_j)}{\sum_{i'<k'<j} \exp(\boldsymbol{\beta}^{(i',k')}_j)} \triangleq \tilde{\boldsymbol{\beta}}^{(i,k)}_j. \tag{4}$$

We use Gumbel-Max technique where $\boldsymbol{g}^{(i,k)}_j$ obeys Gumbel (0,1) distribution,

$$\boldsymbol{I}_j = \text{one\_hot}\left[\arg\max_{i<k<j} (\log \tilde{\boldsymbol{\beta}}^{(i,k)}_j + \boldsymbol{g}^{(i,k)}_j)\right] \in \mathbb{R}^{\frac{j \times (j-1)}{2}}. \tag{5}$$

[3] $\boldsymbol{g}^o_{i,j} = -\log(-\log(\boldsymbol{\epsilon}^o_{i,j}))$, $\boldsymbol{\epsilon}^o_{i,j}$ obeys uniform distribution

Take a cell as a whole, if edge $e_{i,j}$ is sampled, there must be another chosen $e_{k,j}$. Thus $\mathbf{B}_{i,j}$ can be formulated by $\mathbf{I}_j$:

$$\boldsymbol{B}_{i,j} = \sum_{k<i} \boldsymbol{I}^{(k,i)}_j + \sum_{k>i} \boldsymbol{I}^{(i,k)}_j. \tag{6}$$

Gumbel-Softmax reparameterization is used to retain gradient information,

$$\tilde{\boldsymbol{I}}^{(i,k)}_j = \frac{\exp\left\{\left[\log \tilde{\boldsymbol{\beta}}^{(i,k)}_j + \boldsymbol{g}^{(i,k)}_j\right]/\tau\right\}}{\sum_{s<t<j} \exp\left\{\left[\log \tilde{\boldsymbol{\beta}}^{(s,t)}_j + \boldsymbol{g}^{(s,t)}_j\right]/\tau\right\}},$$

$$\boldsymbol{I}_j = \text{one\_hot}\left[\arg\max_{i<k<j} \tilde{\boldsymbol{I}}^{(i,k)}_j\right].$$

3.4.2 Gumbel-Top2 (ROME-v2).

Enumerating all possible edge combinations as in ROME-v1 is straightforward but superfluous, hence in ROME-v2 we directly sample two edges per node. We define the probability of each edge $e_{i,j}$ as $p(e_{i,j})$. Given edge importance is denoted by $\boldsymbol{\beta}$, the sampling probability $p(e_{i,j}) = \frac{\exp(\boldsymbol{\beta}_{i,j})}{\sum_{k<j} \exp(\boldsymbol{\beta}_{k,j})} \triangleq \tilde{\boldsymbol{\beta}}_{i,j}$.

To satisfy the constraints on the cell topology (Eq. 1), ROME-v2 extends Gumbel-Max to Gumbel-Top2:

$$\tilde{\boldsymbol{B}}_{i,j} = \frac{\exp\left((\log \tilde{\boldsymbol{\beta}}_{i,j} + \boldsymbol{g}_{i,j})/\tau\right)}{\sum_{i'<j} \exp\left((\log \tilde{\boldsymbol{\beta}}_{i',j} + \boldsymbol{g}_{i',j})/\tau\right)}, \tag{7}$$

$$\boldsymbol{B}_{i,j} = \begin{cases} 1, & i \in \arg\operatorname{top2}_{i'<j}(\tilde{\boldsymbol{B}}_{i',j}) \\ 0, & otherwise \end{cases} \tag{8}$$

We also demonstrate that the Gumbel-Top2 technique is equivalent to sampling two different edges *without replacement* with probability simplex p_i, so that Gumbel-Top2 sampling can be made differentiable.

3.5. Theoretical Proof on Gumbel-Top2

We show that our Gumbel-Top2 technique is equivalent to sampling two different edges *without replacement* with probability simplex p_i, so that Gumbel-Top2 sampling can be made differentiable. We sketch our proof as follows.

Let p_i be the probability of sampling e_i by a single choice among n predecessors. Without loss of generality, we compute the probability of sampling e_1 in two schemes, and show they are in fact equivalent.

1. **Sampling two edges without replacement**: the probability of e_1 being sampled is $p_1 + \sum_{i=2}^n p_i \frac{p_1}{1-p_i}$.
2. **Sampling with Gumbel-Top2**: Gumbel random variable g_i is obtained by sampling a uniform random variable ϵ_i from $[0,1]$, i.e. $g_i = -\log(-\log \epsilon_i)$. Then the

edges are ranked by $(\log p_i + g_i)$ among which the top two are chosen. There are two cases when e_1 is sampled,

(a) e_1 ranks first, and the probability of e_1 being sampled is p_1 due to Gumbel-Max scheme.

(b) e_1 ranks second, and e_i ranks first. We can directly get its probability,

$$\int_0^1 \epsilon_1^{p_2/p_1} \epsilon_1^{p_3/p_1} \cdots (1 - \epsilon_1^{p_i/p_1}) \cdots \epsilon_1^{p_n/p_1} d\epsilon_1$$
$$= \int_0^1 (1 - \epsilon_1^{p_i/p_1}) \epsilon_1^{\frac{1-p_i}{p_1} - 1} d\epsilon_1 = p_i \frac{p_1}{1 - p_i}.$$

The probability of edge e_1 being sampled is thus $p_1 + \sum_{i=2}^{n} p_i \frac{p_1}{1-p_i}$.

3.6. Gradient Accumulation

Lastly, we tackle the issues caused by insufficient sampling. Suppose architectural weights $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be the parameters of a distribution for architectures. A candidate architecture z is obtained by independently sampling edges and operations. Suppose z owns M edges $\{e_1, ..., e_M\}$ and the corresponding operations $\{o_1, ..., o_M\}$, then the probability of z being selected is given by

$$p(z; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_{i=1}^{M} p(e_i; \boldsymbol{\beta}) \times p(o_i | e_i; \boldsymbol{\alpha}). \qquad (9)$$

The search process can be thus modeled as finding optimal $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to minimize the expectation of validation loss of the architectures as Eq. 10, where $\boldsymbol{\theta}_z^*$ denotes the optimal operation parameters for the sampled architecture z.

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\beta}} \quad & \mathbb{E}_{z \sim p(z; \boldsymbol{\alpha}, \boldsymbol{\beta})} \left[L_{val}(\boldsymbol{\theta}_z^*, z) \right], \\ \text{s.t.} \quad & \boldsymbol{\theta}_z^* = \arg\min_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta}, z) \end{aligned} \qquad (10)$$

However, architecture z changes at each iteration while the corresponding operation weights $\boldsymbol{\omega}$ are updated only once. Apparently, single-path approaches suffer from two problems: *unfair* and *biased training* for candidate operations, and creating *large variance* for architectural weights.

We propose two effective techniques based on *gradient accumulation*. **First**, to boost fair training for operations, we sample K sub-models from the supernet and accumulate gradients for each sub-model within one iteration. Weights $\boldsymbol{\omega}$ are updated by the accumulation of gradients from K sub-models. **Second**, to reduce the variance for architectural weights, we sample another K sub-models and average the gradients of architectural weights. Suppose that the gradient of α be a random variable whose variance is σ^2, then averaging among K samples reduces the variance to σ^2/K. Specifically, we alternately update operation parameters $\boldsymbol{\theta}$ and architectural parameters $\boldsymbol{\alpha}$ (similar for $\boldsymbol{\beta}$) as:

Algorithm 1 ROME (with two versions v1 and v2).

Input: iteration count T; number of sampling K; initialized operation parameters $\boldsymbol{\theta}$; and architectural weights $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$;
Output: optimal architecture z^*;

```
1:  for $t = 1 \to T$ do
2:      Sample two batches of data samples $D_s$ and $D_t$ from
        two disjoint datasets;
3:      for $k = 1 \to K$ do
4:          Topology sampling by Eq. 6 (v1) or Eq. 7 (v2);
            Operation sampling by Eq. 3;
5:          Get sampled architecture $z_k$;
6:      end for
7:      Gradient accumulation and update $\boldsymbol{\alpha}, \boldsymbol{\beta}$ by Eq. 11 on
        $D_s$, where $L_{val}$ is cross entropy;
8:      for $k = 1 \to K$ do
9:          Topology sampling by Eq. 6 (v1) or Eq. 7 (v2);
            Operation sampling by Eq. 3;
10:         Get sampled architecture $z'_k$;
11:     end for
12:     Gradient accumulation and update $\boldsymbol{\theta}$ by Eq. 12 on
        $D_t$, where $L_{train}$ is cross entropy;
13: end for
14: return: $z^* = \arg\max_z p(z; \boldsymbol{\alpha}, \boldsymbol{\beta})$
```

$$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \frac{1}{K} \sum_{k=1}^{K} \nabla_{\boldsymbol{\alpha}} L_{val}(\boldsymbol{\theta}, z_k), \qquad (11)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \sum_{k=1}^{K} \nabla_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta}, z'_k), \qquad (12)$$

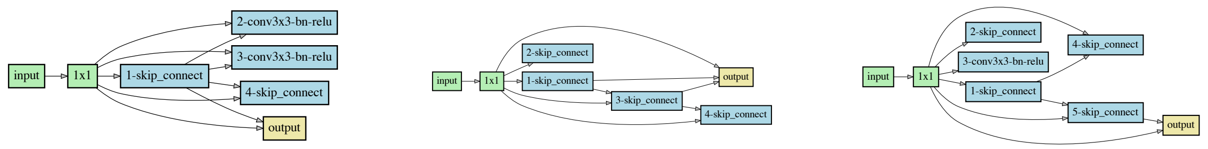
where z_k, z'_k denote the sampled architectures. We can now summarize our ROME method wholely in Alg. 1.

4. Experiments

4.1. Protocols

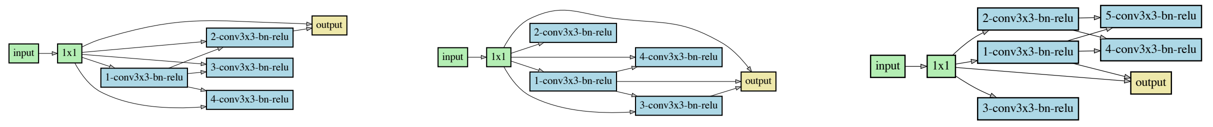
Search spaces. In this paper, we denote DARTS's search space as S0, which comprises a stack of duplicate normal cells and reduction cells. Each cell is represented by a DAG with 4 intermediate nodes. Candidate operations between each two nodes are {maxpool, avgpool, skip_connect, sep_conv 3×3 and 5×5, dil_conv 3×3 and 5×5}. We exclude {none} operation from the default DARTS search space to satisfy the topology constraint in Eq. 1. Under S0 space, we search and evaluate on CIFAR-10 [18] and ImageNet [9] respectively.

We also conduct experiments on four reduced search spaces, S1-S4, introduced by R-DARTS [48] to evaluate the



(a) S1 (b) S2 (c) S3

Figure 4: GDAS fails on NAS-Bench-1Shot1 [49] when searching on CIFAR-10 in all three search spaces when skip connection are added into choices. In each MixedOp, we have three choices: {maxpool3x3, conv3x3-bn-relu, skip-connect}.



(a) S1 (b) S2 (c) S3

Figure 5: ROME-V2 resolves the aggregation of skip connections on NAS-Bench-1Shot1 [49]. Notice intermediate nodes concatenate their outputs as the input for the output node, while some have loose ends and don't feed to the output node.

stability of our method. S1 is a pre-optimized search space, where each edge in the supernet has a predefined set of candidate operations. In the other 3 search spaces, candidate operations on each edge are the same (see the details in the supplementary). Following [48], we search and evaluate in these 4 search spaces on CIFAR-10, CIFAR-100 [18], and SVHN [26] (12 benchmarks in total).

Search settings. Similar to DARTS, the supernet consists of 8 cells with 16 initial channels. We search for 50 epochs and set the sampling number $K = 7$. Unless explicitly written, ROME-v2 is used by default throughout the paper since it is more efficient and robust. For operation parameters, we use the SGD optimizer with a momentum of 0.9 and an initial lr of 0.05; For architectural weights, we adopt the Adam optimizer with an initial lr of 3×10^{-4}.

Evaluation settings. We use standard evaluation settings as DARTS [24] by training the inferred model for 600 epochs using SGD with a batch size of 96. For search space S0, inferred models are constructed by stacking 20 cells with 36 initial channels, and trained under the same settings following [5, 24]. For S1-S4, we strictly follow the settings in R-DARTS [48] for a fair comparison.

4.2. Robustness Evaluation

For comprehensive evaluation, we follow the recommended best practices for NAS by [21, 48, 4, 6, 8] to report the mean and variance across several times of parallel *searching with various random seeds*, through which the robustness of a method can be measured. We appeal to the community for avoiding a common mistake that trains a single best-searched model several times, which only tests the convergence stability of a single model.

Discussion on collapse behavior across popular NAS benchmarks. We argue that excluding an important operation for search space can cause illusive conclusions. Specifically, NAS-Bench-1Shot1 [49] suggests that Gumbel-based NAS is quite robust. However, this observation is laying on the basis that popular skip connections are not included in the search space [44]. After adding skip connection into the choices, we perform GDAS using their released code, whose best model found is full of skip connections, which again supports our discovery of collapse issue in single-path based NAS, as shown in Fig. 4. In contrast, ROME does not suffer the same issue in these search spaces, as shown in Fig. 5.

Robustness evaluation on 12 hard benchmarks. We follow R-DARTS [48] to evaluate the performance and across 3 datasets in S1-S4 search spaces, see Table 1. Our methods robustly outperform other methods with a clear margin across all these benchmarks.

Additionally, we observe that parameterless operations in GDAS dominate the normal cell in both S2 and S3, while our method effectively handles their numbers and thus stabilizes the searching stage.

4.3. Performance Comparison

Performance in S0 on CIFAR. We follow DARTS [24] and search on the CIFAR-10. Table 2 shows that ROME achieves state-of-the-art performance with only 0.3 GPU-days[4]. ROME has an average of 2.58±0.07% error rate,

[4]GDAS searches 250 epochs and costs 0.2 GPU-days. ROME searches 50 epochs with $K = 7$, equivalent to searching 350 epochs by GDAS.

Benchmark		DARTS†	ES†	ADA†	GDAS		ROME	
		Error (%)	Error (%)	Error (%)	Error (%)	Num	Error(%)	Num
C10	S1	4.66±0.71	3.05±0.07	3.03±0.08	2.89±0.09	3.8±0.4	**2.66±0.06**	1.3±0.4
	S2	4.42±0.40	3.41±0.14	3.59±0.31	3.89±0.17	6.0±0.7	**3.14±0.14**	2.0±0.0
	S3	4.12±0.85	3.71±1.14	2.99±0.34	3.04±0.10	6.5±0.5	**2.61±0.00**	2.0±0.0
	S4	6.95±0.18	4.17±0.21	3.89±0.67	3.34±0.10	0.0±0.0	**3.21±0.12**	0.0±0.0
C100	S1	29.93±0.41	28.90±0.81	24.94±0.81	24.49±0.08	4.0±0.0	**22.71±0.71**	2.3±0.4
	S2	28.75±0.92	24.68±1.43	26.88±1.11	24.57±0.47	6.3±0.4	**22.91±0.75**	3.5±0.5
	S3	29.01±0.24	26.99±1.79	24.55±0.63	22.86±0.17	3.0±0.7	**22.43±0.36**	2.5±0.5
	S4	24.77±1.51	23.90±2.01	23.66±0.90	24.14±0.89	2.3±1.1	**20.95±0.45**	0.0±0.0
SVHN	S1	9.88±5.50	2.80±0.09	2.59±0.07	2.48±0.04	2.8±0.4	**2.34±0.06**	0.8±0.4
	S2	3.69±0.12	2.68±0.18	2.79±0.22	3.05±0.02	7.8±0.4	**2.41±0.07**	1.0±0.0
	S3	4.00±1.01	2.78±0.29	2.58±0.07	3.62±0.36	7.5±0.5	**2.56±0.03**	1.5±0.5
	S4	2.90±0.02	2.55±0.15	2.52±0.06	2.51±0.06	1.5±1.5	**2.34±0.00**	0.0±0.0

Table 1: Comparison in 4 search spaces and 3 datasets. For each algorithm, we independently search for 3 times under the settings in R-DARTS [48] and report the averaged performance. 'EA' and 'ADA' are two methods proposed by R-DARTS. 'Num': To reveal the collapse issue, we also report the average number of parameterless operations in the discovered normal cell for GDAS and ROME †: Results are obtained from R-DARTS.

Models	Params (M)	FLOPs (M)	Error (%)	Cost GPU Days	SP
DARTS-V1 [45]	-	-	3.38±0.23	0.4	✗
P-DARTS [5]‡	3.3±0.2	540±34	2.81±0.14	0.3	✗
PC-DARTS [40]‡	3.6±0.5	592±90	2.89±0.22	0.1	✗
PR-DARTS [51]‡	3.4	-	2.68±0.10	0.2	✗
ISTA-NAS [43] ‡	3.3	-	2.71±0.10	0.05	✗
R-DARTS [48]	-	-	2.95±0.21	1.6	✗
SDARTS-ADV [4]	3.3	-	2.61±0.02	1.3	✗
DARTS-PT [33]	3.0	-	2.61±0.08	0.8	✗
NASI-FIX [29]	3.9	-	2.79±0.07	0.24	✗
ZARTS [34]	3.7	-	2.54±0.07	1.0	✗
Few-shot NAS [50]	3.8	-	2.31±0.08	1.35	✗
GDAS [10]	3.4	-	2.93	0.2	✓
SNAS [39]	2.8	-	2.85±0.02	1.5	✓
ROME-v1 (best)	4.5	683	2.53	0.3	✓
ROME-v1 (avg.)	4.0±0.6	670±21	2.63±0.09	0.3	✓
ROME-v2 (best)	3.6	591	2.48	0.3	✓
ROME-v2 (avg.)	3.7±0.2	595±28	2.58±0.07	0.3	✓

Table 2: Averaged performance among 4 independent runs of search on CIFAR-10. ‡: reproduced result using their released code since they didn't report the average performance. †: FLOPs are calculated by their released architecture. SP: single-path based method

Models	Params (M)	Error (%)	Cost GPU Days
ResNet [14]	1.7	22.10$^\diamond$	-
AmoebaNet [28]	3.1	18.93$^\diamond$	3150
PNAS [23]	3.2	19.53$^\diamond$	150
ENAS [27]	4.6	19.43$^\diamond$	0.45
DARTS [24]	-	20.58±0.44*	0.4
GDAS [10]	3.4	18.38	0.2
P-DARTS [5]	3.6	17.49‡	0.3
R-DARTS [48]	-	18.01±0.26	1.6
NASI-FIX [29]	4.0	16.12±0.38	0.24
ZARTS [34]	4.1	16.29±0.53	1.0
ROME-V1 (avg.)	4.4±0.2	17.41±0.12	0.3
ROME-V1 (best)	4.4	17.33	0.3
ROME-V2 (avg.)	3.4±0.3	17.71±0.11	0.3
ROME-V2 (best)	3.3	17.57	0.3

Table 3: Comparison of searched models on CIFAR-100. $^\diamond$: Reported by [10], *: Reported by [48], ‡:Rerun their code.

which is slightly higher than up-to-date SOTAs such as SDARTS-ADV [4]. However, ROME is more than 4× faster. Compared with R-DARTS [48], ROME robustly outperforms it with 5× fewer search costs. Our best model achieves 97.52% accuracy with 3.6M parameters.

We also search on CIFAR-100 and show the results in Table 3. ROME surpasses all the methods and achieves state-of-the-art with only 0.3 GPU-days search cost.

Performance in S0 on ImageNet. First, we transfer the architecture searched on CIFAR-10 to ImageNet following the common practice [24, 5, 19, 8]. Same as [19, 8], we train models for 250 epochs with a batch size of 1024 by SGD optimizer with a momentum of 0.9 and an initial lr of 0.5 base learning rate. We also utilize an auxiliary classifier strategy. The results are shown in Table 4, where ROME

Models	**FLOPs** (M)	**Params** (M)	**Top-1** (%)	**Cost** (GPU days)
AmoebaNet-A [28]	555	5.1	74.5	3150
NASNet-A [53]	564	5.3	74.0	2000
PNAS [23]	588	5.1	74.2	225
DARTS [24]	574	4.7	73.3	0.4
P-DARTS [5]	577	5.1	75.3	0.3
FairDARTS-B [8]	541	4.8	75.1	0.4
SNAS [39]	522	4.3	72.7	1.5
PC-DARTS [40]	586	5.3	74.9	0.1
GDAS [10]	581	5.3	74.0	0.2
ROME (ours)	576	5.2	75.3	0.3
DARTS, P-DARTS	OOM when batch-size ≥ 32			
FairNAS-C [7]	321	4.4	74.7	12
ProxylessNAS [3]	465	7.1	75.1	8.3
FBNet-C [38]	375	5.5	74.9	9
PC-DARTS [40]‡	597	5.3	75.4	3.8
GDAS [10]‡	405	3.6	72.5	0.8
ROME (ours)	556	5.1	75.5	0.5

Table 4: Performance on ImageNet. The first block indicates the models *transferred* from CIFAR-10; The second block indicates that the models are *directly searched* on ImageNet. ‡: reproduced using their released code.

achieves 75.3% top-1 accuracy.

Second, as ROME features low memory cost and great robustness, we directly search on ImageNet as well. We randomly sample 10% images to train operation parameters and another 10% to train architectural weights. A supernet is constructed by stacking 8 cells with 16 initial channels. We search for 30 epochs with $K = 3$. The batch size is set as 256. Our search cost is reduced to 0.4 GPU days on a single Tesla V100. We fully train the discovered model for 250 epochs with the same evaluation settings as above. Results are illustrated in Tabel 4, showing that ROME achieves 75.5% top-1 accuracy. To make fair comparisons, we reproduce GDAS [10] under the same settings (90 epochs). However, the network is dominated by skip connections and only achieves 72.5% top-1 accuracy.

5. Ablation Study

Sensitivity to the Sampling Number K. K is a hyper-parameter in our gradient accumulation strategy, which is designed to reduce the variance of noise on $\nabla_{\boldsymbol{\alpha}} L_{val}$ and stabilizes the search as analyzed in Sec. 3.6.

Table 5 compares the performance by setting K as $1, 4, 7, 10$ in ROME. We search and evaluate on CIFAR-10. Three parallel tests on each setting are conducted. Note we adjust the number of search epochs to have same number of iterations per test. We observe that the performance increases monotonically with K which verifies our analysis that biased training shall be alleviated. The result demonstrates the effectiveness of our gradient accumulation strategy. Also, as the accuracy saturates at K=7, we set K=7.

K	**Acc** (%)	**# Params**	**# Epochs**
1	97.12±0.06	3.34M	350
4	97.28±0.07	3.57M	87
7	97.42±0.07	3.73M	50
10	97.46±0.12	4.06M	35

Table 5: Sensitivity study of sampling number K. For each setting, we adjust the number of search epochs according to K for fair comparison. We do three parallel tests on each setting and report the mean and standard deviation.

TD	GA		Acc (%)	TD	GA		Acc (%)
	$\boldsymbol{\theta}$	$\boldsymbol{\alpha}$			$\boldsymbol{\theta}$	$\boldsymbol{\alpha}$	
×	×	×	96.52±0.07	✓	×	×	97.12±0.06
×	×	✓	96.85±0.31	✓	×	✓	97.22±0.07
×	✓	×	96.98±0.05	✓	✓	×	97.34±0.07
×	✓	✓	97.14±0.05	✓	✓	✓	**97.42±0.07**

Table 6: Component study of ROME on CIFAR-10.

Component analysis for instability issue. There are two major components that contribute to the cure for instability in ROME: topology disentanglement (TD) and gradient accumulation (GA) for $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$. To show their efficacy, we conduct ablation studies in S0 space on CIFAR-10.

Results are shown in Table 6, where 'GA for $\boldsymbol{\theta}$' indicates that gradient accumulation is applied over $K = 7$ sampled architectures to train operation parameters; 'GA for $\boldsymbol{\alpha}$' indicates that gradients for architectural parameters over $K = 7$ sampled architectures is accumulated and averaged. The setting without TD and GA (first line in Table 6) degenerates to GDAS [10]. Our ROME adopts both TD and GA, which is the last line in Table 6.

We observe that TD alone can significantly improve the searching performance, showing that the *inconsistency issue is the principal reason for performance collapse*. This is as expected. On the one hand, inconsistent topology between search and evaluation results in an inconsistent searching objective; On the other hand, training the weights of 14 operations (w/o TD) is much more difficult than training 8 operations (w/ TD), which degrades the convergence.

Moreover, we observe that gradient accumulation on $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$ can further improve the search performance, which confirms our analysis that performance collapse issue also comes from insufficient sampling.

Memory Analysis. Table 7 compares GPU memory cost in S0 search space on CIFAR-10. ROME has the lowest

Method	DARTS	GDAS	PC-DARTS		ROME
			M=4	M=2	
Memory (G)	9.4	3.1	3.7	5.7	**2.3**

Table 7: GPU Memory cost comparison. We measured the cost based on a batch size of 64, where the supernet has 16 initial channels, and 8 layers.

memory cost thanks to our disentanglement of the search for topology. Unlike GDAS that preserves multiple edges for each node, we strictly sample 2 edges for each node leading to 26% memory reduction compared to GDAS.

PC-DARTS [40] uses partial channels during the search stage to reduce GPU memory cost, in which the partial ratio is controlled by a hyperparameter M. But M requires careful calibration for different tasks. In contrast, ROME doesn't require calibrating such an extra hyperparameter and is more memory-efficient.

6. Comparison with Prior Works

Here we highlight the difference of ROME from the existing works. **1) ROME vs. DOTS.** DOTS [12] explores an edge importance representation for one-shot NAS in a multi-stage fashion but the operations are divided into two groups (parameter-free and parameter-bearing, following DropNAS [15]) beforehand, which is a very strong prior. The length of each stage has to be tuned carefully from dataset to dataset. In contrast, no prior or extra hyper-parameters are used in ROME; **2) ROME vs. DDW.** Unlike DDW [46] that belongs to dynamic networks with a changeable topology dependent on inputs, ROME is a NAS method designed to search for a static architecture. **3) ROME vs. SNAS.** SNAS [3] adopts Gumbel-softmax via masking, whose supernet still resides in the memory and thus not memory-efficient. In contrast, ROME is a truly single-path NAS method and inherits the property of low memory cost. SNAS didn't deal with the collapse issue either.

7. Conclusion

In this paper, we highlight the performance collapse issue of single-path differentiable NAS, and attribute the cause to topology inconsistency between searching and evaluation, as well as the stochastic nature of sampling for candidate operations. To address the above issues, we propose ROME that features topology disentanglement and gradient accumulation strategy to stabilize the searching process. ROME achieves state-of-the-art results across 15 recent popular benchmarks, which manifests its strong performance, low memory cost and robustness.

Acknowledgments

Xiaoxing Wang and Yuda Fan partly contributed to this work when they were intern at Meituan and now Yuda Fan is with ETHZ. This work was partly supported by National Key R&D Program of China (No. 2022ZD0118700), NSFC (62222607), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

References

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and Simplifying One-Shot Architecture Search. In *ICML*, pages 549–558, 2018. 1

[2] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. SMASH: One-Shot Model Architecture. Search Through HyperNetworks. In *ICLR*, 2018. 1

[3] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*, 2019. 1, 2, 8, 9

[4] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *ICML*, 2020. 1, 2, 6, 7

[5] 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, 6, 7, 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. 1, 2, 6

[7] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *ICCV*, 2021. 2, 8

[8] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. *ECCV*, 2020. 1, 2, 6, 7, 8

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, pages 248–255. IEEE, 2009. 5

[10] Xuanyi Dong and Yi Yang. Searching for a Robust Neural Architecture in Four GPU Hours. In *CVPR*, pages 1761–1770, 2019. 1, 2, 3, 7, 8

[11] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, pages 7036–7045, 2019. 1

[12] Yuchao Gu, Lijuan Wang, Yun Liu, Yi Yang, Yu-Huan Wu, Shao-Ping Lu, and Ming-Ming Cheng. DOTS: decoupling operation and topology in differentiable architecture search. In *CVPR*, 2021. 9

[13] 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. 2

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016. 7

[15] Weijun Hong, Guilin Li, Weinan Zhang, Ruiming Tang, Yunhe Wang, Zhenguo Li, and Yong Yu. Dropnas: Grouped operation dropout for differentiable architecture search. In *IJCAI*, 2022. 9

[16] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *ICCV*, 2019. 1

[17] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2016. 1

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. Technical report, Citeseer, 2009. 5, 6

[19] Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Müller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *CVPR*, 2020. 7

[20] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019. 1, 2

[21] Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. *JMLR*, 21(243):1–18, 2020. 2, 6

[22] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, pages 82–92, 2019. 1

[23] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive Neural Architecture Search. In *ECCV*, pages 19–34, 2018. 7, 8

[24] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2019. 1, 2, 3, 6, 7, 8

[25] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*. OpenReview.net, 2017. 1

[26] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPSW*, 2011. 6

[27] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*, 2018. 1, 2, 7

[28] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, volume 33, pages 4780–4789, 2019. 7, 8

[29] Yao Shu, Shaofeng Cai, Zhongxiang Dai, Beng Chin Ooi, and Bryan Kian Hsiang Low. NASI: label- and data-agnostic neural architecture search at initialization. In *ICLR*, 2022. 7

[30] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours. *ECML PKDD*, 2019. 1, 2

[31] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*, 2019. 1

[32] Mingxing Tan and Quoc V Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*, 2019. 1

[33] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable NAS. In *ICLR*, 2021. 7

[34] Xiaoxing Wang, Wenxuan Guo, Jianlin Su, Xiaokang Yang, and Junchi Yan. ZARTS: on zero-order optimization for neural architecture search. In *NeurIPS*, 2022. 7

[35] Xiaoxing Wang, Zhirui Lian, Jiale Lin, Chao Xue, and Junchi Yan. Diy your easynas for vision: Convolution operation merging, map channel reducing, and search space to supernet conversion tooling. In *PAMI*, 2023. 2

[36] Xiaoxing Wang, Jiale Lin, Juanping Zhao, Xiaokang Yang, and Junchi Yan. Eautodet: Efficient architecture search for object detection. In *ECCV*, volume 13680, pages 668–684, 2022. 1, 2

[37] Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. Mergenas: Merge operations into one for differentiable architecture search. In *IJCAI*, 2020. 2

[38] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *CVPR*, 2019. 1, 8

[39] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic Neural Architecture Search. In *ICLR*, 2019. 1, 3, 7, 8

[40] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *ICLR*, 2020. 2, 7, 8, 9

[41] Chao Xue, Xiaoxing Wang, Junchi Yan, Yonggang Hu, Xiaokang Yang, and Kewei Sun. Rethinking bi-level optimization in neural architecture search: A gibbs sampling perspective. In *AAAI*, pages 10551–10559, 2021. 1

[42] Chao Xue, Xiaoxing Wang, Junchi Yan, and Chun-Guang Li. A max-flow based approach for neural architecture search. In *ECCV*, volume 13680, pages 685–701, 2022. 1

[43] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *NeurIPS*, 33, 2020. 7

[44] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114, Long Beach, California, USA, 09–15 Jun 2019. PMLR. 6

[45] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *ICLR*, 2020. 7

[46] Kun Yuan, Quanquan Li, Shaopeng Guo, Dapeng Chen, Aojun Zhou, Fengwei Yu, and Ziwei Liu. Differentiable dynamic wirings for neural networks. In *ICCV*, pages 327–336, 2021. 9

[47] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and Robustifying Differentiable Architecture Search. In *ICLR*, 2020. 1

[48] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020. 1, 2, 5, 6, 7

[49] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *ICLR*. OpenReview.net, 2020. 6

[50] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 12707–12718, 2021. 7

[51] Pan Zhou, Caiming Xiong, Richard Socher, and Steven Hoi. Theory-inspired path-regularized differential network architecture search. In *NeurIPS*, 2020. 7

[52] Barret Zoph and Quoc V Le. Neural Architecture Search with Reinforcement Learning. In *ICLR*, 2017. 1

[53] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning Transferable Architectures for Scalable Image Recognition. In *CVPR*, volume 2, 2018. 2, 8