DDSAS: Dynamic and Differentiable Space-Architecture Search

Longxing Yang[1,2,3] YANGLONGXING20B@ICT.AC.CN
Yu Hu[1,2,3] * HUYU@ICT.AC.CN
Shun Lu[1,2,3] LUSHUN19S@ICT.AC.CN
Zihao Sun[1,2,3] SUNZIHAO18Z@ICT.AC.CN
Jilin Mei[1,2,3] MEIJILIN@ICT.AC.CN
Yiming Zeng[4] ZYMS5244@GMAIL.COM
Zhiping Shi[5] SHIZP@CNU.EDU.CN
Yinhe Han[1,2,3] YINHES@ICT.AC.CN
Xiaowei Li[2,3] LXW@ICT.AC.CN
[1]*Research Center for Intelligent Computing Systems, Institute of Computing Technology, CAS.*
[2]*State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS.*
[3]*School of Computer Science and Technology, University of Chinese Academy of Sciences.*
[4]*Tecent ADlab.*
[5]*Capital Normal University.*

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

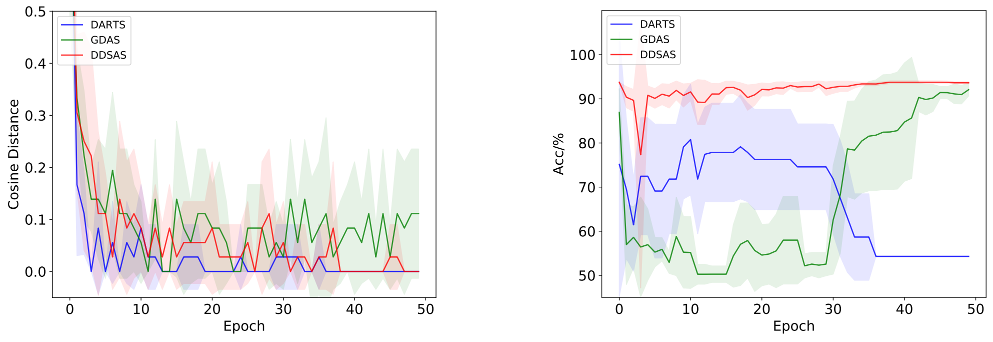
Neural Architecture Search (NAS) has made remarkable progress in automatically designing neural networks. However, existing differentiable NAS and stochastic NAS methods are either biased towards exploitation and thus may converge to a local minimum, or biased towards exploration and thus converge slowly. In this work, we propose a Dynamic and Differentiable Space-Architecture Search (DDSAS) method to address the exploration-exploitation dilemma. DDSAS dynamically samples space, searches architectures in the sampled subspace with gradient descent, and leverages the Upper Confidence Bound (UCB) to balance exploitation and exploration. The whole search space is elastic, offering flexibility to evolve and to consider resource constraints. Experiments on image classification datasets demonstrate that with only 4GB memory and 3 hours for searching, DDSAS achieves 2.39% test error on CIFAR10, 16.26% test error on CIFAR100, and 23.9% test error when transferring to ImageNet. When directly searching on ImageNet, DDSAS achieves comparable accuracy with more than 6.5 times speedup over state-of-the-art methods. The source codes are available at https://github.com/xingxing-123/DDSAS.

Keywords: Neural Architecture Search, Exploration-Exploitation Dilemma

1. Introduction

Neural architecture search (NAS) has demonstrated remarkable progress in automatically designing various neural networks. How to efficiently search an optimal architecture is the major theme throughout the development of NAS. Early works attempted to search an

. *Corresponding Author: huyu@ict.ac.cn



(a) Cosine distance between architectures (b) Search accuracy

Figure 1: (a) The cosine distance between architectures of adjacent epochs. A larger distance indicates higher diversity of searched architectures thus more exploration. The cosine distance of DDSAS is higher than that of DARTS. (b) The search accuracy along epochs. DDSAS converges near 30 epochs while GDAS does not converge well. The results come from NAS-Bench-201 and more details are provided in Section 4.5.1.

entire network Zoph and Le (2017) or search cells to build a network Real et al. (2019); Zoph et al. (2018). Due to the tremendous number of candidate architectures, these methods required thousands of GPU days to search and retrain architectures for a relatively simple image classification task on CIFAR10. Later on, Pham et al. (2018) proposed the weight sharing mechanism, reducing the search time to only a few hours.

While weight sharing reduces the search time, the large number of architecture evaluations becomes a new bottleneck. Liu et al. (2019) proposed a differentiable architecture search technique (DARTS) to address the issue. DARTS relaxes the discrete search space to be continuous so that the architecture can be optimized by gradient descent. However, gradient descent optimizer is more exploitative than explorative Jie et al. (2009), which means DARTS is biased towards exploitation and may fall into a local optimum Chen and Hsieh (2020); Xu et al. (2019). Stochastic differentiable methods Dong and Yang (2019b); Xie et al. (2019) alleviate the aforementioned problem by utilizing the Gumbel-Softmax technique Jang et al. (2017). On one hand, stepwise sampling ensures more exploration of the search space. On the other hand, the refined annealing process requires much more epochs, e.g. 150 epochs in SNAS Xie et al. (2019) and 240 epochs in GDAS Dong and Yang (2019b), resulting in slow convergence.

How to efficiently explore and exploit the search space to find an optimal architecture? In this work, we propose a Dynamic and Differentiable Space-Architecture Search (DDSAS) method to solve the problem. For encouraging exploration, we combine the differentiable search with dynamic space sampling to avoid being trapped in local optimum. For better exploitation, we utilize space-grained sampling to achieve faster convergence than architecture-grained sampling in the Gumbel-Softmax technique. As demonstrated in Figure 1, DDSAS has more exploration than DARTS and faster convergence than GDAS.

The flow of our method is shown in Figure 2. Our basic idea is to sequentially sample the search space, and then analyze the quality of the sampled subspaces to guide the next sampling. The contributions of this work are:

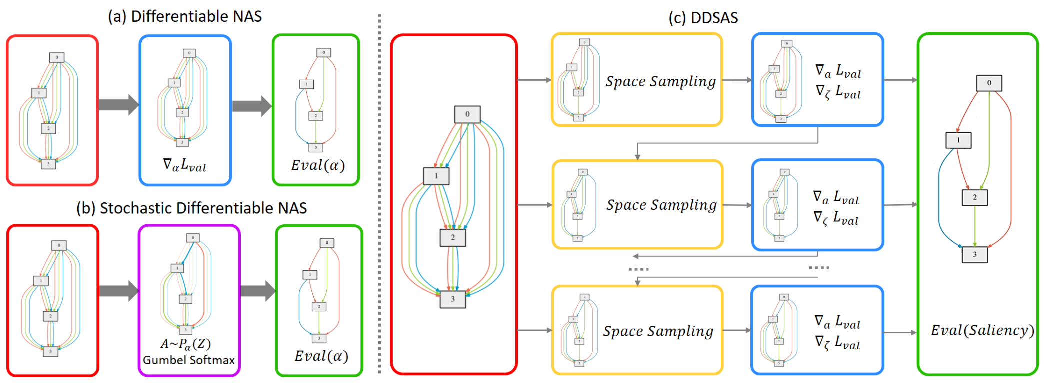


Figure 2: Comparison of differentiable NAS, stochastic differentiable NAS, and DDSAS. The red, blue, purple, orange, and green rectangles represent search space, differentiable search, stochastic differentiable search, space sampling, and performance evaluation, respectively. (a) Differentiable NAS method (e.g. DARTS) directly searches architectures in a whole search space. (b) Stochastic differentiable NAS method (e.g. SNAS) utilizes Gumbel-Softmax to sample architectures from the whole search space. (c) DDSAS samples subspaces from the whole search space, and then conducts a differentiable search in the sampled spaces. The succeeding space sampling is guided by the prior sample information.

1. We propose a DDSAS method that balances exploration-exploitation during the search by using Upper Confidence Bound (UCB) to guide the dynamic space sampling and the joint optimization of space and architecture. Due to the exploration-exploitation trade-off, DDSAS can efficiently find an optimal architecture. With only 4GB memory and 3 hours, DDSAS achieves 2.39% test error on CIFAR10, 16.26% test error on CIFAR100, and 23.9% test error when transferring to ImageNet.

2. The computation and memory cost can be flexibly controlled according to the resource limitations. Benefiting from space sampling, computation and memory resources are only related to subspace. When directly searching on ImageNet, DDSAS achieves comparable accuracy with more than 6.5 times speedup over state-of-the-art methods.

3. The whole search space can be elastic during the search. DDSAS can abandon useless operations or add new operations to realize shrinkage or expansion of the search space, which allows online adaptation to resource fluctuation. Experiments show that DDSAS with search space shrinkage and expansion can also stably obtain high-performance architecture.

2. Related Work

From the perspective of search space Lingxi et al. (2020), the related work can be broadly classified into four categories.

The NAS-RL Search Space. In the early stage, Zoph and Le (2017) proposed the NAS-RL search space to search hyper-parameters of the entire network. However, the search is time-consuming due to evaluating tremendous candidate architectures.

The NASNet Search Space. Inspired by the modular structure , Zoph et al. (2018) proposed the NASNet search space to search cells to build a network instead of entire

Z	a search space representation
A	an architecture representation
ζ	space parameters to calculate $f(\zeta)$
α	architecture parameters to calculate $g(Z, \alpha)$
T	the total number of sampling iterations
t	the sampled number of the operations
$f(\zeta)$	the probability of an operation sampled to the subspace.
$g(Z, \alpha)$	the probability of an operation selected for edges in one subspace.
L	a hyper-parameter for top-L selection, indicating the size of the subspace
M	a hyper-parameter, indicating the gradient descent step in the sampled subspace.
saliency	the values used to select the final architecture

Table 1: The definition of main notations utilized in the paper.

architectures. Lately, Liu et al. (2018) proposed PNAS to accelerate the search by using a sequential model-based optimization strategy. Real et al. (2019) proposed AmoebaNet based on evolutionary algorithms. To further reduce search time, Pham et al. (2018) proposed a weight-sharing method ENAS to avoid training every architecture from scratch and achieved searching in a few hours.

The MobileNet Search Space. This search space replaces cells in NASNet search space with efficient hand-crafted network blocks, e.g. MobileNet Andrew et al. (2017) blocks. The hyper-parameters of blocks are searched, e.g. kernel size and expansion ratio. Many works Cai et al. (2019); Tan et al. (2019); Wu et al. (2019) have shown that the search space is hardware-friendly and the lower-bound of accuracy is high Lingxi et al. (2020). Compared with searching well-designed blocks, searching cell topologies and operation types are more challenging, thus our experiments mainly focus on the DARTS search space.

The DARTS Search Space. DARTS Liu et al. (2019) simplified the NASNet search space and provided a search space with higher flexibility. DARTS and its variants Chen and Hsieh (2020); Chen et al. (2021); Li et al. (2020, 2021); Xu et al. (2019); Zela et al. (2020a) relax the discrete search space to be continuous, thus the architecture search and weight optimization is unified in gradient-based algorithm. Afterwards, Dong and Yang (2019b); Xie et al. (2019) leverage Gumbel-Softmax to enforce discrete architecture being searched in a gradient manner. P-DARTS Chen et al. (2019) progressively increases the depth of searched architectures with the shrinkage of the search space by reducing operations. CNAS Guo et al. (2020) gradually incorporates the learned knowledge to guide the search from a small space to a large space. Unlike P-DARTS which monotonically shrinks the search space and CNAS which expands it, DDSAS can either shrink or expand the search space, offering more flexibility to adapt to resource fluctuation. Wang et al. (2019) proposed LaNAS to iteratively learn action space and sample an architecture by Monte Carlo Tree Search (MCTS). Different from the discrete sampling in LaNAS, DDSAS jointly optimizes space and architecture parameters with gradient descent, thus achieving a much shorter search time (150 GPU days vs 0.13 GPU days, as shown in Table 2).

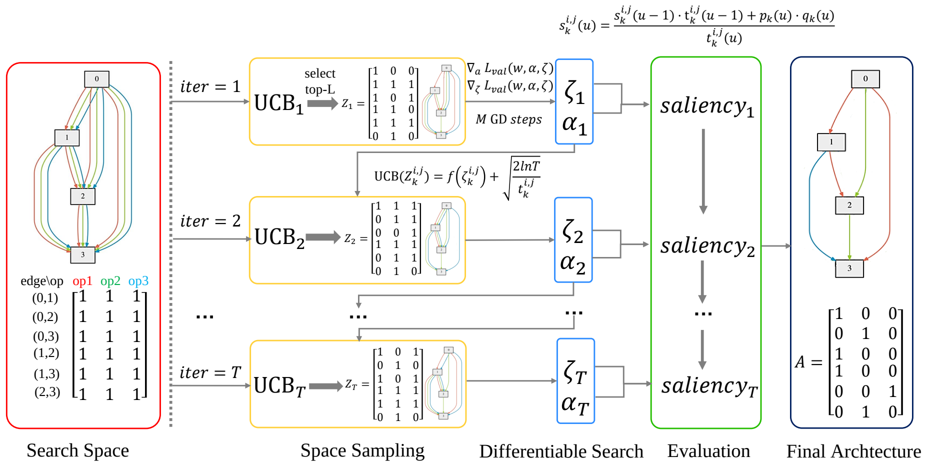


Figure 3: The detailed implementation of DDSAS. The colors have the same meaning as in Figure 2 (c), except the black means the final architecture. In each iteration, the top-L operations are selected as candidates to form a sampled subspace based on the UCB score (as shown in the orange rectangle). Differentiable search is used to optimize space and architecture parameters (as shown in the blue rectangle) for evaluation. The space parameters will be used in the UCB calculation of the next iteration to guide succeeding space sampling. The saliency will be updated in the evaluation stage (as shown in the green rectangle), and finally, the $salieny_T$ that fuses all the sampled subspace information is used for selecting an optimal architecture. *For the sake of easier understanding, we use the supernet diagram Liu et al. (2019) as an example.

3. Method

3.1. Problem Formalization

For cell-level NAS methods Liu et al. (2018), either a normal cell or a reduction cell can be represented as a directed acyclic graphs (DAG), i.e. a supernet. For conventional cell-level differentiable methods, the NAS problem can be formulated as:

$$\begin{aligned} &\min_{\alpha} \; L_{val}(\omega^*(\alpha), \alpha) \\ s.t. \quad &\omega^*(\alpha) = \arg\min_{\omega} L_{train}(\omega, \alpha) \end{aligned} \tag{1}$$

L_{train} and L_{val} are the training and validation loss respectively. ω and α are weight and architecture parameters of the supernet respectively.

In this work, we have the subspace hierarchy. Suppose there are T sampled subspaces $Z_{1\ldots T}$, then the NAS problem is formulated as:

$$\begin{aligned} &\min_{\zeta_i, \alpha_i} L_{val}(\omega^*(\zeta_i, \alpha_i), \zeta_i, \alpha_i; Z_i) \\ s.t. \quad &\omega^*(\zeta_i, \alpha_i) = \arg\min_{\omega} L_{train}(\omega, \zeta_i, \alpha_i; Z_i) \end{aligned} \tag{2}$$

where ζ are space parameters. Then we utilize *saliency* to select the final architecture. We calculate *saliency* based on ζ_i^*, α_i^*:

$$saliency = EvalAllSubspaces(\zeta_{1..T}^*, \alpha_{1..T}^*) \tag{3}$$

3.2. Search Space and Architecture Representation

Assuming the DAG has n nodes, e edges with m candidate operations applied to each edge, then the search space can be represented by $Z, Z \in R^{e \times m}, Z_k^{i,j} \in \{0, 1\}$, where the 0-element in Z means the k-th operation does not exist between edge (i, j), while 1-element means the operation exists. If $\forall Z_k^{i,j}, Z_k^{i,j} = 1$, then Z represents the whole search space as shown in the red rectangle in Figure 3. If $\exists Z_k^{i,j} = 0$, then Z is a subspace as shown in the orange rectangle in Figure 3. Besides, an architecture is represented by $A, A \in R^{e \times m}, A_k^{i,j} \in \{0, 1\}, ||A^{i,j}||_0 = 1$, which can be considered as a subspace only containing one architecture (e.g. the final architecture in the black rectangle of Figure 3).

3.3. Space Sampling

Space sampling is a key stage in DDSAS because it determines the trade-off between exploration and exploitation of the whole space. At the beginning of the search, the early iterations mainly focus on exploration, learning more knowledge about the landscape of the search space, while the later iterations tend to concentrate search on the subspaces in which better architectures more likely exist. We use the Upper Confidence Bound (UCB) Auer et al. (2002) to guide the balance of exploration and exploitation:

$$UCB(Z_k^{i,j}) = f(\zeta_k^{i,j}) + \sqrt{\frac{2logT}{t_k^{i,j}}} \tag{4}$$

where $\zeta_k^{i,j}$ is the space parameters, $f(\zeta_k^{i,j})$ is the estimated probability of $P(Z_k^{i,j} = 1)$, T is the total number of sampling iterations, and $t_k^{i,j}$ is the number of $Z_k^{i,j} = 1$. The $f(\zeta_k^{i,j})$ term corresponds to exploitation while the $\sqrt{\frac{2logT}{t_k^{i,j}}}$ term corresponds to exploration. We calculate $f(\zeta_k^{i,j})$ with the Sigmoid function:

$$f(\zeta_k^{i,j}) = Sigmoid(\zeta_k^{i,j}) \tag{5}$$

Afterwards, we select the operations with the highest top-L confidence to form the sampled subspace Z and use gradient descent to search in the sampled subspace with M steps. Ablation study on the two hyper-parameters L and M will be discussed in the supplementary material due to the limited space. Based on Eq. (4), the UCB sampling strategy tends to explore the operations with fewer sampled times and exploit the operations with higher estimation probabilities, thus balancing the exploration and exploitation of the search space. Besides, space sampling can decouple resource requirements from the whole search space settings by adjusting the two hyper-parameters L and M, making the proposed method resource-friendly.

3.4. Differentiable Search

To make the search space continuous, DARTS relaxes categorical choices by utilizing a softmax over all operations:

$$\bar{o}^{i,j}(x) = \sum_{k=1}^{m} \frac{exp(\alpha_k^{i,j})}{\sum exp(\alpha_{k'}^{i,j})} o_k(x) \tag{6}$$

where $o_k(x)$ is the k-th operation with a latent representation x, and $\bar{o}^{i,j}(x)$ is the mixed operation.

Because of having the subspace hierarchy, the quality of the sampled subspaces needs to be evalueted to guide next sampling. Therefore, we combine $f(\zeta)$ into differentiable search for optimization space parameters:

$$\bar{o}^{i,j}(x) = \sum_{k=1}^{m} \frac{f(\zeta_k^{i,j}) g(Z_k^{i,j}, \alpha^{i,j})}{\sum f(\zeta_{k'}^{i,j}) g(Z_{k'}^{i,j}, \alpha^{i,j})} o_k(x) \tag{7}$$

$$g(Z_k^{i,j}, \alpha^{i,j}) = \frac{I(Z_k^{i,j} = 1) exp(\alpha_k^{i,j})}{\sum_{k'}^{m} I(Z_{k'}^{i,j} = 1) exp(\alpha_{k'}^{i,j})} \tag{8}$$

where $g(Z_k^{i,j}, \alpha^{i,j})$ denotes the probability of the k-th operation selected for edge (i, j), i.e. $P(A_k^{i,j} = 1|Z^{i,j})$, and the value of $I(\cdot)$ is 1 when the condition is true, and 0 when false. If $Z_k^{i,j} = 0$, then $g(Z_k^{i,j}, \alpha^{i,j}) = 0$ and the corresponding operation will not participate in the forward propagation. Because the forward pass and the back-propagation pass is only conducted in the sampled subspace, the computation and memory cost is greatly reduced.

To better understand the Eq. (7), we reveal that the Eq. (7) is an approximation in terms of the edge-wise joint distribution of space and architecture, i.e. $P(Z^{i,j}, A^{i,j})$. For the sake of simplicity, we set $p_k = f(\zeta_k^{i,j}), q_k = g(Z_k^{i,j}, \alpha^{i,j}), \bar{p}_k = 1 - p_k, \bar{Z}_k^{i,j} = 1 - Z_k^{i,j}$.

Assume that operations in edge (i, j) are independently selected into the sampled subspace Z as candidates, and given $A_{k1}^{i,j} = 1$, then we get the edge-wise joint distribution:

$$\begin{aligned} P(Z^{i,j}, A^{i,j}) &= P(A^{i,j}|Z^{i,j}) \cdot P(Z^{i,j}) \\ &= (\prod_{k=1}^{m} (q_k)^{A_k^{i,j}}) \cdot (\prod_{k=1}^{m} (p_k)^{Z_k^{i,j}} (\bar{p}_k)^{\bar{Z}_k^{i,j}}) \\ &= q_{k1} \cdot (\prod_{k=1}^{m} (p_k)^{Z_k^{i,j}} (\bar{p}_k)^{\bar{Z}_k^{i,j}}) \\ &= q_{k1} p_{k1} \cdot (\prod_{k \neq k1} (p_k)^{Z_k^{i,j}} (\bar{p}_k)^{\bar{Z}_k^{i,j}}) \end{aligned} \tag{9}$$

Although it is straightforward to use Eq. (9) as the weight of the operation, an approximation can reduce computation cost. Ideally, if the first term $q_{k1}p_{k1}$ of Eq. (9) dominates the joint distribution, then we can use the normalized $p_k q_k$ as the weight of the mixed operation $\bar{o}^{i,j}(x)$ to make the search space continuous, i.e. Eq. (7).

Suppose $A1$ and $A2$ are two architectures yielded by subspace Z. We compare $k1$-th and $k2$-th operations under the $Z_k^{i,j}$. Let $A1_{k1}^{i,j} = 1, A2_{k2}^{i,j} = 1$, then:

$$\frac{P(Z^{i,j}, A1^{i,j})}{P(Z^{i,j}, A2^{i,j})} = (\frac{q_{k1}p_{k1}}{q_{k2}p_{k2}}) \cdot (\frac{\prod_{k \neq k1} (p_k)^{Z_k^{i,j}} (\bar{p}_k)^{\bar{Z}_k^{i,j}}}{\prod_{k' \neq k2} (p_{k'})^{Z_{k'}^{i,j}} (\bar{p}_{k'})^{\bar{Z}_{k'}^{i,j}}}) \tag{10}$$

So if the ratio of the second term of Eq. (10) is 1, then the first term is dominant. We will demonstrate the assumption is valid in ablation study (Section 4.5.2).

3.5. Evaluation

We select the optimal architecture based on the saliency of each operation. We use $s \in R^{e \times n}$ to evaluate the saliency of each operation on each edge. Suppose that $s_k^{i,j}(u)$ is the k-th operation between nodes i and j in the u-th iteration. If one operation is a candidate in the u-th iteration, its saliency will be updated, otherwise, no update. The update equation is as follows and please note that $t_k^{i,j}(u) = \sum_{v=1}^{u} Z_k^{i,j}(v)$:

$$s_k^{i,j}(u) = \frac{(s_k^{i,j}(u-1) \cdot t_k^{i,j}(u-1) + p_k(u) \cdot q_k(u))}{t_k^{i,j}(u)} \tag{11}$$

In other words, the saliency of an operation indicates the average performance of the operation in all sampled subspaces, which is equivalent to Eq. (11):

$$s_k^{i,j}(T) = \frac{\sum_{u=1}^{T} Z_k^{i,j}(u) \cdot p_k(u) \cdot q_k(u)}{t_k^{i,j}(T)} \tag{12}$$

The final architecture A is chosen based on saliency s. $argmax(s^{i,j}(T))$ is used for selecting an operation for edge (i, j). We edge-wisely select the non-zero operation with the highest saliency. And similar to DARTS, we let each node have two incoming edges.

The pseudocode of DDSAS is shown in Algorithm 1.

Algorithm 1 DDSAS

```
Input: subspace size $L$, the number of gradient descent steps $M$
Output: an architecture $A$
Initialize: network weights $\omega$, space parameters $\zeta$, architecture parameters $\alpha$, confidence
    $UCB$, operation saliency $s$, the total number of sampling iterations $T$, edge-wise sam-
    pling numbers of operations $t$, $steps$
 1: while not converged do
 2:   if step % M == 0 then
 3:     Update confidence $UCB$ by Eq. (4)
 4:     Sample the subspaces Z with the top-$L$ operations based on confidence in $UCB$
 5:     Edge-wisely accumulate the sampling numbers of operations: $t_k^{i,j} \leftarrow t_k^{i,j} + Z_k^{i,j}$
 6:     $T \leftarrow T + 1$
 7:   end if
 8:   Update space parameters $\zeta$ and architecture parameters $\alpha$ by $\nabla_\zeta L_{val}(\omega, \zeta, \alpha)$ and
      $\nabla_\alpha L_{val}(\omega, \zeta, \alpha)$ by Eq. (2)(7)
 9:   Update network weights $\omega$ by $\nabla_\omega L_{train}(\omega, \zeta, \alpha)$ by Eq. (2)
10:   $steps \leftarrow steps + 1$
11:   if step % M == 0 then
12:     Update saliency $s$ by Eq. (11)
13:   end if
14: end while
15: return an architecture $A$ with the highest operation saliency $s$.
```

4. Experiments

4.1. Datasets

We use DDSAS to search architectures on five datasets, including CIFAR10, CIFAR100, ImageNet, NAS-Bench-1Shot1, and NASBench201. CIFAR10 and CIFAR100 Krizhevsky and Hinton (2009) are two popular datasets containing 50K training images and 10K testing images, respectively. ILSVRC2012 Russakovsky et al. (2015) has 1.28M training and 50K validation images with 1000 object categories, which is used to test the transferability and scalability of DDSAS. Note that the search space on CIFAR10, CIFAR100, and ImageNet is the DARTS search space. NAS-Bench-1shot1 Zela et al. (2020b) is a benchmark framework with three one-shot search spaces from NAS-Bench-101 dataset Ying et al. (2019). The three spaces contain 6240, 29160, 363648 architectures respectively. NAS-Bench-201 Dong and Yang (2020) is an extension to NAS-Bench-101 with a different search space containing 15625 architectures. Also, we conducted ablation studies to further analyze DDSAS.

4.2. Results on CIFAR10 and CIFAR100

The predefined network in the search phase is the same as DARTS, which consists of eight layers of cells, including six normal cells and two reduction cells. Each cell has seven nodes with four intermediate nodes, two input nodes, and one output node. There are fourteen edges in one cell and eight candidates operations for one edge, which can build the whole search space of a normal cell or reduction cell. The eight candidate operations are *skip-connect*, *max-pool-3*$\times$*3*, *avg-pool-3*$\times$*3*, *sep-conv-3*$\times$*3*, *sep-conv-5*$\times$*5*, *dil-conv-3*$\times$*3*, *dil-conv-5*$\times$*5*, and *zero*. Besides, the size of subspace L is 28 and gradient descent steps M is 30. We also designed two experiments to investigate the adaptation of DDSAS to search space changes. One is shrinking the whole search space. Every 10 epochs, we delete the 14 operations that have the lowest saliency among all edges from the search space, i.e. subsequent space sampling will no longer consider the subspaces containing the deleted operations. DDSAS searches for a total of 60 epochs. The other one is expanding the whole search space. We first initialize the search space with three candidate operations for each edge, i.e. *skip-connect*, *zero*, *sep-conv-3*$\times$*3*. To reduce the bias on either nonparametric operations or parametric operations, we heuristically expand the search space by adding another operation per edge after every 10 epochs following the sequence of *max-pool-3*$\times$*3*, *dil-conv-3*$\times$*3*, *avg-pool-3*$\times$*3*, *sep-conv-5*$\times$*5*, *dil-conv-5*$\times$*5*. Likewise, a total of 60 epochs are searched. The final search space consists of eight operations. The hyper-parameter settings are placed in the supplementary material.

Table 2 shows the comparison between DDSAS and other methods on CIFAR10 and CIFAR100. We report the results by four independent runs with different random seeds. The average test error on CIFAR10 and CIFAR100 is 2.59% and 17.15% respectively, indicating that DDSAS can search for a good architecture under different seed initialization. Note that despite of similar performance and search cost on CIFAR10, DDSAS has significant performance gain on ImageNet than NASP/PC-DARTS in Table 3 (23.9 vs. 26.3/25.1). Compared with fixed search space, the average errors are reduced to 2.52% and 2.57% on CIFAR10, 16.74% and 16.77% on CIFAR100, indicating that shrinkage and expansion can improve search accuracy, especially on CIFAR100. This testifies that the whole search space

Method	CIFAR10		CIFAR100		GPU	Algorithm
	Test Err.(%)	Params(M)	Test Err.(%)	Params(M)	Days	
NASNet Zoph et al. (2018)	2.65	3.3	-	-	1800	RL
AmoebaNet Real et al. (2019)	2.55±0.05	2.8	-	-	3150	EA
PNAS Liu et al. (2018)	3.41±0.09	3.2	-	-	225	SMBO
ENAS Pham et al. (2018)	2.89	4.6	-	-	0.5	RL
LaNAS Wang et al. (2019)	2.53±0.05	3.2	-	-	150	MCTS
DARTS Liu et al. (2019)	3.00±0.14	3.3	20.58±0.44 ‡	-	1.5	gradient
SNAS Xie et al. (2019)	2.85±0.02	2.8	-	-	1.5	gradient
P-DARTS Chen et al. (2019)	2.5	3.4	**15.92**	3.6	0.3	gradient
RobustDARTS Zela et al. (2020a)	2.95±0.21	-	18.01±0.26	-	1.6	gradient
SDARTS Chen and Hsieh (2020)	2.61±0.02	3.3	-	-	1.3	gradient
GDAS Dong and Yang (2019b)	2.93	3.4	18.38	3.4	0.17	gradient
SGAS Li et al. (2020)	2.66±0.24	3.7	-	-	0.25	gradient
NASP Yao et al. (2020)	2.83±0.09	3.3	-	-	0.1	gradient
PC-DARTS Xu et al. (2019)	2.57±0.07	3.6	-	-	0.1	gradient
DrNAS Chen et al. (2021)	2.54±0.03	4.0	-	-	0.4	gradient
CNAS Guo et al. (2020)	2.60±0.06	3.7	-	-	0.3	gradient
DDSAS	2.59±0.17*	3.5	17.15±0.33*	3.4	0.13	gradient
DDSAS (shrinking)	2.52±0.06*	3.9	16.74±0.18*	3.7	0.13	gradient
DDSAS (expanding)	2.57±0.10*	3.7	16.77±0.38*	3.9	0.13	gradient
DDSAS (best)	**2.39**	3.6	16.26	4.2	0.13	gradient

Table 2: Comparison with state-of-the-art NAS methods on CIFAR10, CIFAR100. ‡ the data is reported by Zela et al. (2020a). * The mean and the standard deviation come from four independent search with different random seeds.

can be elastic during the DDSAS search process. The best architecture achieves test errors of 2.39% and 16.26%, coming from fixed search space and expanding search space respectively. This shows that DDSAS has excellent performance compared with other methods, while only requiring 4GB memory and 0.13 GPU days on one NVIDIA V100 GPU.

4.3. Results on ImageNet

We test the transferability of the best cells searched on CIFAR10 by evaluating them on ImageNet. As shown in Table 3, with only 0.13 GPU days, DDSAS can achieve 23.9% Top-1 error and 7.1% Top-5 error, outperforming all the other methods. To further verify the scalability of our method, we directly search on ImageNet. We follow PC-DARTS Xu et al. (2019) to randomly select 10% and 2.5% images from ImageNet to train weight parameters and space-architecture parameters respectively. A total of 60 epochs are searched with a batch size 512. We set L to 28 and M to 30, which is the same as CIFAR10. We only sample operations with trainable parameters in the first 40 epochs and sample all operations in the last 20 epochs. Our method achieves an error of 24.2/7.3% with 5.5M parameters and 0.58 GPU Days, which has comparable accuracy with more than 6.5 times speedup over state-of-the-art methods.

4.4. Results on NASBench

In NAS-Bench-1shot1, an architecture consists of three stacked blocks with a max-pooling operation in between. Each block has three cells. Selecting the final network architecture

Method	Top1 Err.(%)	Top5 Err.(%)	Params(M)	GPU Days	Algorithm
Transfer the optimal architecture to ImageNet					
NASNet Zoph et al. (2018)	26.0	8.4	5.3	1800	RL
AmoebaNet Real et al. (2019)	24.3	7.6	6.4	3150	EA
PNAS Liu et al. (2018)	25.8	8.1	5.1	225	SMBO
LaNAS Wang et al. (2019)	25.0	0.7	5.1	150	MCTS
DARTS Liu et al. (2019)	26.7	8.7	4.7	1.5	gradient
SNAS Xie et al. (2019)	27.3	9.2	4.3	1.5	gradient
P-DARTS Chen et al. (2019)	24.4	7.4	4.9	0.3	gradient
SDARTS Chen and Hsieh (2020)	25.2	7.8	-	1.3	gradient
GDAS Dong and Yang (2019b)	26.0	8.5	5.3	0.17	gradient
SGAS Li et al. (2020)	24.2	7.2	5.3	0.25	gradient
NASP Yao et al. (2020)	26.3	8.6	9.5	0.2	gradient
PC-DARTS Xu et al. (2019)	25.1	7.8	5.3	0.1	gradient
CNAS Guo et al. (2020)	24.6	7.4	5.3	0.3	gradient
DDSAS	**23.9**	**7.1**	5.3	0.13	gradient
Directly search on ImageNet in the DARTS search space					
DrNAS Chen et al. (2021)	24.2	7.3	5.2	3.9	gradient
PC-DARTS Xu et al. (2019)	24.2	7.3	5.3	3.8	gradient
DDSAS	24.2	7.3	5.5	**0.58**	gradient

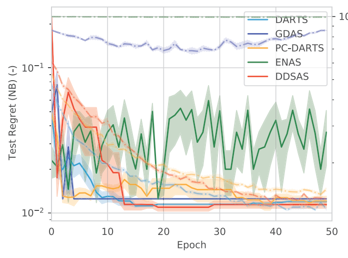
Table 3: Comparison with state-of-the-art NAS methods on ImageNet. The upper part lists the methods that search on CIFAR10 or CIFAR100 and then transfer the optimal architecture to ImageNet. The lower part lists the methods that directly search on ImageNet.

requires searching the connections of input and output nodes and the choice blocks. For the sake of simplicity, we only sample the space formed by the operation of choice blocks and directly use softmax to search for the connection of input and output nodes. As the search space changes, we modify the hyper-parameter L to 6, while M=30 remains unchanged. Other hyper-parameters are the same as the DARTS settings in NAS-Bench-1shot1. We searched for 50 epochs with six different random seeds, and showed the trend of errors of different methods in Figure 4.

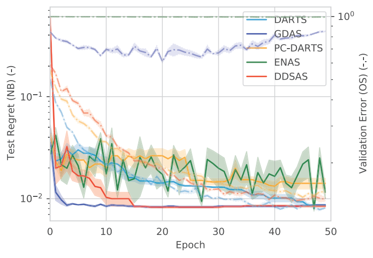
As shown in Figure 4, DDSAS has outstanding performance in the three search spaces of NAS-Bench-1shot1. Firstly, the solid line shows that DDSAS can reach the lowest test regret in different spaces. Secondly, DDSAS converges faster than DARTS, which shows the efficiency of DDSAS. Finally, the standard deviation of DDSAS after convergence is very small, showing the stability of DDSAS.

In NAS-Bench-201, an architecture is built by three stacked cells, which are connected by a residual block. Each stack has five normal cells and the residual block has stride 2. The DAG of a normal cell has four nodes and multiple edges. A mixture of five candidate operations is placed on each edge. We transfer DDSAS to this search space and set the hyper-parameter L=12 and M=30 because the number of candidate operations is less than that in CIFAR10. Following the settings in NAS-Bench-1shot1, we use random seeds in our search process and keep other hyper-parameters the same as in DARTS.

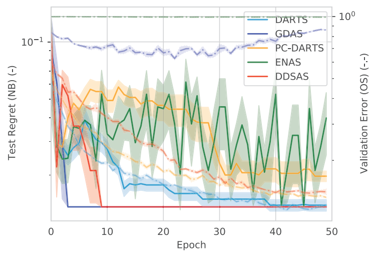
Table 4 compares DDSAS with other differentiable search methods on NAS-Bench-201. DDSAS achieves better performance than baseline methods on three datasets. Moreover,



(a) Space 1



(b) Space 2



(c) Space 3

Figure 4: Comparison of DDSAS with other one-shot NAS optimizers on the three different search spaces in NAS-Bench-1shot1. The solid lines show the anytime test regret (mean ± std), while the dashed blurred lines represent validation error. Colors indicate different methods and the red color represents DDSAS.

Method	Search (hours)	CIFAR10		CIFAR100		ImageNet-16-120	
		validation	test	validation	test	validation	test
RSPS Li and Talwalkar (2019)	2.2	80.42±3.58	84.07±3.61	52.12±5.55	52.31±5.77	27.22±3.24	26.28±3.09
SETN Dong and Yang (2019a)	3.2	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
ENAS Pham et al. (2018)	3.9	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
DARTS Liu et al. (2019)	3.2	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS Dong and Yang (2019b)	8.7	89.89±0.08	93.61±0.09	**71.34**±0.04	70.70±0.30	41.59±1.33	41.71±0.98
GDAS with 50 epochs	1.7	88.50±1.71	92.13±1.49	67.95±2.90	68.12±2.74	38.22±4.44	38.70±4.21
DDSAS (avg.)	**1.5**	**90.94**±0.44	**93.73**±0.32	71.23±0.95	**71.53**±1.02	**44.47**±0.89	**45.02**±1.20
DDSAS (best)	1.5	91.34	94.24	72.52	72.76	45.55	46.85
global optimal	-	91.61	94.37	73.49	73.51	46.77	47.31

Table 4: Comparison of DDSAS with other NAS methods on NAS-Bench-201. We report the average and the best performance of DDSAS.

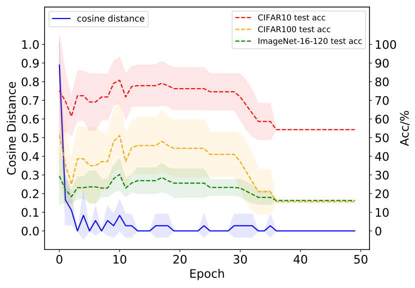
the best test accuracy 94.24% of DDSAS on CIFAR10 is only 0.13% less than the global optimal, indicating the effectiveness of DDSAS.

4.5. Ablation Study

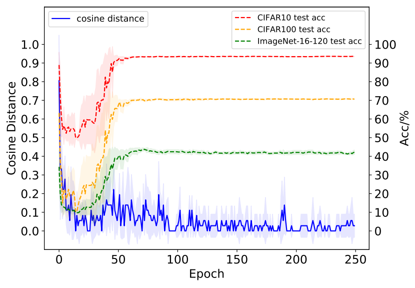
4.5.1. Exploration and Exploitation of Search Space

We use cosine distance and test accuracy to reflect the exploration and exploitation, respectively. The cosine distance is calculated by the one-hot encoding of architectures between adjacent epochs. Larger cosine distance indicates higher diversity of searched architectures thus more exploration, while higher accuracy indicates more effective exploitation.

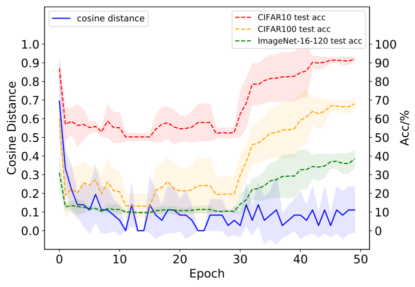
As shown in Figure 5, DARTS has low cosine distance, showing that the gradient descent algorithm used in DARTS is more exploitative than explorative Jie et al. (2009). Moreover, the test accuracy of DARTS drops quickly after 35 epochs, showing an occurrence of performance collapse. GDAS has good performance when searching with 250 epochs. However, it consumes 8.7 *hours* while DDSAS only needs 1.5 *hours*. When GDAS searches with 50 epochs at a faster annealing speed, both cosine distance and test accuracy have high standard deviation, suggesting more exploration yet less exploitation. DDSAS has higher cosine distance than DARTS but lower than GDAS, and achieves better accuracy, indicating balanced exploration and exploitation.



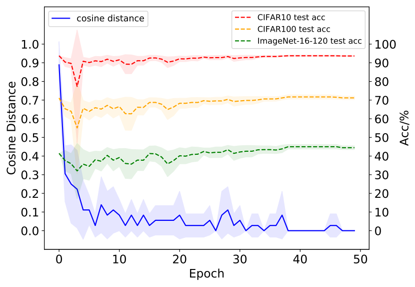
(a) DARTS, search with 50 epochs



(b) GDAS, search with 250 epochs



(c) GDAS, search with 50 epochs



(d) DDSAS, search with 50 epochs

Figure 5: The exploration-exploitation capability of DARTS, GDAS, and DDSAS on NAS-Bench-201. The blue solid line represents the cosine distance of the architectures between adjacent epochs. The red, orange and green regions represent standard deviation of the average test accuracy (mean $\pm$ std) on CIFAR10, CIFAR100, and ImageNet-16-120, respectively. (a) DARTS has low cosine distance and low test accuracy after 35 epochs. (b) GDAS has high standard deviation of test accuracy in the first 50 epochs. (c) Both cosine distance and test accuracy of GDAS have high standard deviation when searching with 50 epochs, especially for ImageNet-16-120. (d) DDSAS has higher cosine distance than DARTS but lower than GDAS, and achieves higher accuracy.

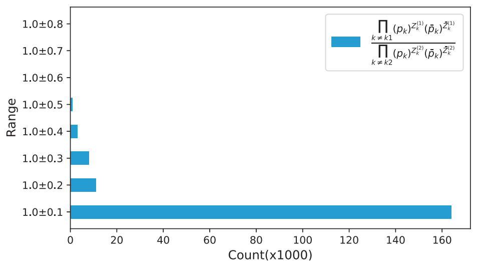
4.5.2. Approximation of Joint Probability

In Eq. (10), the difference between two architectures that are yielded by the same subspace depends on the two terms on the right-hand side of the equation. We empirically demonstrate that the second term is close to 1, so the joint distribution of space and architecture parameters is dominated by the first term in Eq. (9). All experimental settings are the same as those on CIFAR10. We run the experiment 4 times with different random seeds.

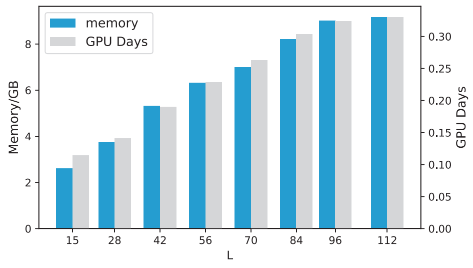
The number of combinations of $(A1_{k_1}^{i,j} = 1, A2_{k_2}^{i,j} = 1)$ for 4 nodes and 8 operation options is 191,276. Figure 6(a) shows 82.2% of the counts of combinations in which the second term of Eq. (10) is within the 1.0 ± 0.1 range, indicating that the second item is close to 1. Therefore, it is reasonable to use $p_k q_k$ as an approximation.

4.5.3. Resource Cost of DDSAS

Through space sampling, DDSAS alleviates the resource bottleneck incurred by searching in an entire space. The memory and time cost can be controlled by the hyper-parameter L.



(*a*) The counts of combinations



(*b*) Resource cost

Figure 6: (a) The counts of combinations in which the second term of Eq. (10) is within different ranges. (b) Resource cost of DDSAS. The histogram shows the usage of memory (blue) and GPU Days (grey) of different L.

For example, given the DARTS search space as the entire space, the DAG has 14 edges. Since at least one of eight operations is selected for each edge, L ranges in $[14, 112]$. When each edge contains only one operation, i.e. $L = 14$, the coefficient is equal to 1 according to Eq. (7). In this case, optimizing the space parameter and architecture parameter is meaningless. Therefore, the minimum of L is 15. In our experiments, we set L from $\{15, 28, 42, 56, 70, 84, 96, 112\}$ to measure memory and time cost with a batch size 64.

The resource cost is shown in Figure 6(b). The memory cost and search time linearly increase as L increases. Since L can take any integer within $[15, 112]$, DDSAS offers great flexibility for deployment on diverse devices.

5. Conclusion

In this work, we propose a dynamic and differentiable space-architecture search method to search for an optimal architecture. By balancing exploration and exploitation, DDSAS outperforms state-of-the-art NAS methods on multiple datasets. In the future, we will further explore the search space evolution.

Acknowledgments

This work is supported in part by the National Key R&D Program of China under Grant No. 2018AAA0102701, and in part by the National Natural Science Foundation of China under Grant No. 6217023268.

References

G. Howard Andrew, Zhu Menglong, Chen Bo, Kalenichenko Dmitry, Wang Weijun, Weyand Tobias, Andreetto Marco, and Adam Hartwig. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 2002.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *ICML*, 2020.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. In *ICLR*, 2021.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019.

Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *ICCV*, 2019a.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *CVPR*, 2019b.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020.

Yong Guo, Yaofo Chen, Yin Zheng, Peilin Zhao, Jian Chen, Junzhou Huang, and Mingkui Tan. Breaking the curse of space explosion: Towards efficient nas with curriculum search. In *ICML*, 2020.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Chen Jie, Xin Bin, Peng Zhihong, Dou LiHua, and Zhang Juan. Optimal contraction theorem for exploration-exploitation tradeoff in search and optimization. *IEEE Trans. Syst. Man Cybern. Part A*, 39(3):680–691, 2009. doi: 10.1109/TSMCA.2009.2012436.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

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

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *UAT*, 2019.

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *ICLR*, 2021.

Xie Lingxi, Chen Xin, Bi Kaifeng, Wei Longhui, Xu Yuhui, Chen Zhengsu, Wang Lanfei, Xiao An, Changa Jianlong, Zhang Xiaopeng, and Tian Qi. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *arXiv:2008.01475*, 2020.

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*, 2018.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning action space. *arXiv:1906.06832*, 2019.

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.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *ICLR*, 2019.

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*, 2019.

Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. Efficient neural architecture search via proximal iterations. In *AAAI*, 2020.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*, 2019.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020a.

Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *ICLR*, 2020b.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.