Neurocomputing 497 (2022) 227–238

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

STC-NAS: Fast neural architecture search with source-target consistency

Zihao Sun [a,c], Yu Hu [a,b,c], Longxing Yang [a,c], Shun Lu [a,c], Jilin Mei [a,c], Yinhe Han [a,b,c], Xiaowei Li [b,c]

[a] Research Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[b] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[c] University of Chinese Academy of Sciences, Beijing 100049, China

ARTICLE INFO

Article history:
Received 28 May 2021
Revised 22 August 2021
Accepted 21 November 2021
Available online 27 November 2021

Keywords:
Neural architecture search
Consistency
Automatic
Jensen-Shannon divergence

ABSTRACT

Neural architecture search (NAS) has shown very promising results for automatically designing network models. Most existing cell-based NAS approaches generate the target network model from a source super-network, which usually confront inconsistency issues. In this paper, we propose a new NAS method named STC-NAS, a fast neural architecture search with source-target consistency, so that not only the performance of the searched target model is improved but also the search process is boosted. Specifically, during the search phase, we sample the source super-network to let the samples be consistent with the target model. Moreover, we leverage the Jensen-Shannon divergence to ensure the samples are optimized in the direction of being more similar to the target model. Experimental results demonstrate that our method needs only 0.059 GPU-days to search on CIFAR-10. Benefited from its efficiency, STC-NAS can directly search the target super-network on the target task datasets, achieving 2.42% test error on CIFAR-10, 16.45% test error on CIFAR-100, and 24.2% test error on ImageNet datasets.

1. Introduction

Neural Architecture Search (NAS) is emerging as a new design paradigm to automatically design neural networks. After defining the search space that contains enormous neural networks, NAS explores the search space with specifically designed search strategies to find out the optimal network models. Recently, NAS has shown remarkable performance gain beyond manually designed networks on various tasks including classification [1,2], object detection [3,4] and semantic segmentation [5,6].

At the early stages, reinforcement learning (RL) [2,7,8] and evolutionary algorithms (EA) [9–11] are proposed to search for neural networks. However, these methods often need to sample many subnetworks and then train from scratch, which may consume thousands of GPU-days to find a relatively good architecture even on the small dataset CIFAR-10. To reduce the search cost, ENAS [12] proposed to leverage weight sharing among the same operations in different subnetworks, thus it only needs to optimize a super-network that contains all possible subnetworks and the target neural network can be evaluated by inheriting the parameters from the super-network directly. Afterwards, DARTS [13] proposed continuous relaxation of architecture parameters so that the architecture parameters and network weights can be trained alternately with gradient descent by solving a bi-level optimization problem.

Unfortunately, DARTS still suffers from serious inconsistency issues. This is because DARTS needs to optimize all candidate operations in a cell that deviates from the target cell, but only retaining the optimal operation on each edge by discretization at the end of the search that can dramatically downgrade the training accuracy of the super-network [14,15]. In addition, the softmax relaxation can cause unfair competition among the different mixed operations, and the optimization is also very time- and memory-consuming, so that most existing methods usually search in a proxy mode. The proxy can either be a network proxy or a dataset proxy. More specifically, the network proxy mode means the depth and the width of the super-network in the search phase are shallower and narrower than the target model for evaluation. However, as network depth and width have significant impacts on convergence [16,17], this network proxy mode may not lead to an optimal solution. Besides, DARTS and its variants need to search on a proxy dataset and then transfer the searched architecture to the target dataset for evaluation, which may produce the suboptimal solution as different datasets may have different distributions [18,19]. From the above mentioned issues, we can summarize four inconsistencies in the aspects of cell, discretization, depth/width, and task dataset. As shown in Fig. 1, the cell optimization process of DARTS is inconsistent, that is, it needs to optimize all candidate operations in an edge but maintain only one operation in the final cell. Moreover, the process of selecting operations discretely is also inconsis-

https://doi.org/10.1016/j.neucom.2021.11.082
0925-2312/

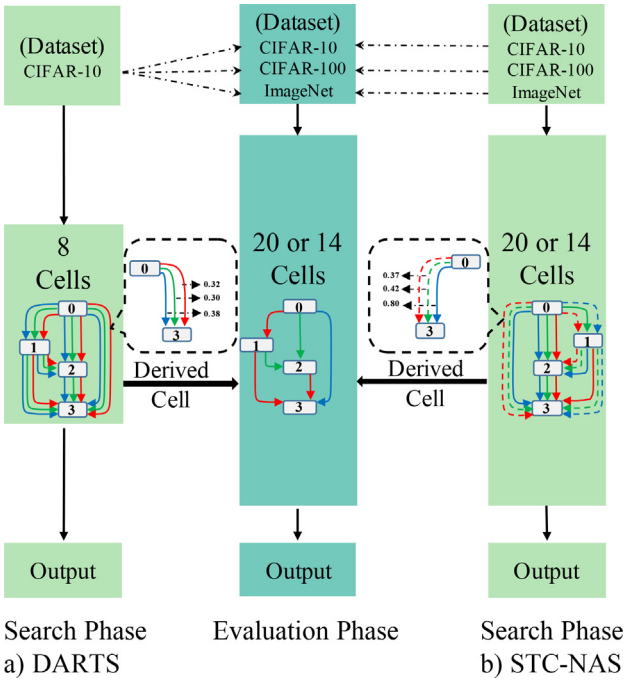


Fig. 1. The inconsistency issues of DARTS compared with our method STC-NAS. DARTS optimizes all the operations during the search phase, and the architecture parameter weights are very close, leading to the cell and discretization inconsistencies. Also, it searches a shallower super-network on a smaller dataset, incurring the depth and task dataset inconsistencies. Whereas our STC-NAS searches with the source-target consistency to avoid the above issues..

tent because the softmax function makes architecture weights among candidate operations be close. The search process is time- and memory-consuming due to the first two inconsistencies, so DARTS has to search in a proxy mode, i.e., searching on a shallower network and on a smaller dataset, leading to the depth and task dataset inconsistency.

In this paper, we propose STC-NAS, a fast sampling-based differentiable neural architecture search with source-target consistency. We maintain the cell topology sampled from the source super-network during the search process to be consistent with the final target evaluated architecture. To ensure that the optimization process of the sampled architecture is stable and consistent, we replace the softmax function with the ReLU function to calculate the weights of architecture parameters, thus preventing the competition among different candidate operations. Moreover, we leverage Jensen-Shannon divergence as an auxiliary loss for architecture parameters optimization. Finally, due to the consistent sampling optimization process, our method is time- and memory-efficient so that the network proxy and task dataset proxy are no longer needed. Our contribution can be summarized as follows:

- We develop a fast neural architecture search with source-target consistency, named STC-NAS, which means that we maintain the number of edges in the searched cell is the same as that in the target cell. Thus it greatly reduces the resource consumption and improves the search efficiency. As shown in Table 3, our method can improve the batch size to 256 with 5.5GB of GPU memory, but costs only 0.059 GPU-days to finish the search, which is much more efficient than DARTS and other methods.
- We use the ReLU function to independently update the architecture weights of each candidate operation without passive competition against each other under the softmax relaxation, and leverage Jensen-Shannon divergence as an auxiliary loss for architecture parameters optimization, which can strengthen the distinction among different candidate operations.
- Experimental results show that our method is very time- and memory-efficient and can directly search on the target super-network and task dataset without any proxy. STC-NAS achieves 2.42% test error on CIFAR-10, 16.45% test error on CIFAR-100, and 24.2% test error on ImageNet, outperforming the state-of-the-art works.

2. Related Works

Differentiable Architecture Search. Comparing to RL or EA based architecture search, differentiable NAS optimizes the network weights and architecture parameters alternatively using gradient descent. ENAS [12] proposed to share the same weights among different sub-networks if they have the same operation. Though ENAS reduces the search cost significantly, the search space is still discrete. To solve this problem, DARTS [13] relaxes architectures into continuous space, making the architecture search differentiable and hence more efficient. Unfortunately, the weight sharing algorithm suffers from the instability issues, which is named as discretization gap [14,15,20]. FairDARTS [14] forces the values of architecture parameters towards either zero or one so that the operations with very low weights can be easily removed. But as shown in experimental results of FairDARTS, the weights of many operations are close to one, which means the discretization gap still exists. Moreover, we leverage JS divergence loss as the regularization to strengthen the distinction among different candidate operations, which is also different from FairDARTS that introduces a zero-one loss to reduce the discretization gap. SGAS [21] and ASAP [22] gradually prune redundant operations during the search process, hence the search space gradually squeezes and approximates to the final target architecture. However, the greedy pruning method may remove important operations at the early stage before operations could be well trained. P-DARTS [17] proposed to progressively increase the depth of the super-network, but the depth of the super-network in the last stage of the search process is still shallower than the target super-network. Moreover, progressively increasing depth is not an end-to-end solution. PC-DARTS [23] can directly search on large datasets but at the price of utilizing only a subset of feature map channels.

Sampling-based Differentiable Architecture Search. SNAS [24] firstly leverages the Gumble-softmax technique to solve the inconsistency problem between the performance of the derived child network and the source super-network. However, the super-network contains the whole graph which is not only different from the derived architecture of the target network but also is computation- and memory-consuming. Afterwards, GDAS [25] and DATA [26] reduce the search cost by sampling discrete subnetwork. However, these methods need to compute the softmax of architecture parameters after each iteration, leading to competition among operations and preference of non-parameter operations. NASP [27] only propagates the proximal paths for each edge in search, but the active paths of super-network still deviate from the target-network as there is no connection for some edges in the target-network. ProxylessNAS [28] addresses the $O(nM)$ memory complexity issue by sampling two paths (i.e., operations) on each edge, so only two paths are involved in each update step of the architecture parameters, indicating $O(2M)$ memory complexity [29]. In fact, ProxylessNAS mitigates the inconsistency problem but not resolves it, as the target network only has one path on each edge.

Sampling-based Non-differentiable NAS. ENAS [12], AutoHAS [30], and TuNAS [31] all sample the sub-network to update the

super-network weights, so the memory and time complexity are the same as ours: O(M)/O(T). Their non-differentiable sampling process utilizes a reinforcement controller trained with policy gradient which is computed with the Monte Carlo estimation on the validation dataset. Whereas, our differentiable sampling process optimizes the loss function directly on the training dataset. The first disadvantage of sampling-based non-differentiable NAS methods is that the policy gradient computed with the Monte Carlo estimation has a higher variance than the standard SGD gradient [12], and the second disadvantage is that the reinforcement-learning based NAS method converges slowly and achieves noisy accuracy along with the search [22], inferior to differentiable NAS methods.

3. Methodology

3.1. Preliminary

Differentiable architecture search builds a super-network by stacking L cells. A cell is defined as a directed acyclic graph (DAG) with N nodes, where each node x^i represents the related feature map and each edge (i,j) is associated with mixed operation $o^{(i,j)}$. For each cell, the input nodes are represented by the outputs from the previous two cells, each intermediate node aggregates information flows from all of its predecessors, and the output node is defined as a concatenation of a fixed number of its predecessors. During the search stage, by constructing a source super-network consisting of L cells (e.g. $L = 8$ in DARTS), all the possible operations within the search space $\mathcal{O}$ on an edge (i,j) are parameterized as architecture parameters $\alpha^{(i,j)}$ by using softmax relaxation,

$$\bar{o}^{(i,j)}(x^j) = \sum_{o\in\mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o\prime\in\mathcal{O}} \exp(\alpha_{o\prime}^{(i,j)})} o(x^i). \tag{1}$$

Thus, the differentiable architecture search can be formulated as a bi-level optimization problem:

$$\begin{aligned} &\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha), \\ &s.t. \quad w^*(\alpha) = \arg\min_w \mathcal{L}_{train}(w, \alpha), \end{aligned} \tag{2}$$

where the network weights and architecture parameters are alternately optimized on the training and validation datasets respectively.

At the end of the search, the final architecture is derived by selecting the operation with the largest architecture weight for every mixture operation $o^{(i,j)} = argmax_{o\in\mathcal{O}} \alpha_o^{(i,j)}$, and then a deeper target super-network (e.g. $L = 20$ in DARTS) is built to evaluate the performance of the searched architecture.

3.2. The Inconsistency Collapse

As mentioned above, the previous differentiable architecture search optimizes all candidate operations in an edge by using softmax but only one operation will be selected, this may introduce the inconsistency collapse, and we categorize it as cell inconsistency and discretization inconsistency.

Cell Inconsistency. The number of edges in the source cell is inconsistent with the target cell, which means that the whole super-network needs to be loaded in the GPU, thus the search process is memory-consuming. As shown in Table 3, when using the same batch size of 64, DARTS requires 9.4GB of GPU memory which is extremely large when comparing with the 1.8GB of GPU memory required by STC-NAS. On the other hand, the architecture parameters are optimized by softmax relaxation, whereas it provides an exclusive competition since increasing one is at the cost of suppressing others. As a result, the non-parameter operation (e.g. skip-connection) becomes gradually dominant during optimization. From Fig. 2, we can see that half of the operations in the normal cell is skip-connection, and almost all operations in the reduction cell are also non-parametric. Because non-parameter operations are beneficial for the supernet training, but too many of them do not contribute to the final performance.

Discretization Inconsistency. At the end of the search, the final cell is derived by simply selecting the operation with the largest weight. However, the magnitudes of architecture parameters are so close that discretization can degrade the performance significantly. As can be seen from Fig. 3, the range of softmax α is too narrow to distinguish which operation is superior to others. For example, on the first edge of the normal cell, the operation of skip-connect and dil-conv-3x3 are the same weight and are also very close to the weight of sep-conv-3x3, so it is hard to distinguish the best operation by simply selecting the largest weight's operation. What's worse, Table 1 shows the accuracy has dropped by almost 79.2% by the discretization mentioned above.

The next, we propose our STC-NAS method to solve the inconsistency issues of differentiable neural architecture search. In order to express it more clearly, we distinguish the specific meaning of different α in Table 2, and the main framework of STC-NAS is shown in Fig. 4.

3.3. Sampling with Cell Consistency

To avoid unfair competition among different candidate operations, we use the Relu activation function[1] to compute the weights of the architecture parameters instead of softmax. In this way the importance of each operation is independently calculated without passively competing against each other, and the weights of each operation is guaranteed to be greater than zero. Otherwise, it will bring negative effects on the feature map if the weight of an operation is less than zero. Thus we replace Eq. (1) with

$$\bar{o}^{(i,j)}(x^j) = \sum_{o\in\mathcal{O}} Relu(\alpha_o^{(i,j)}) o(x^i). \tag{3}$$

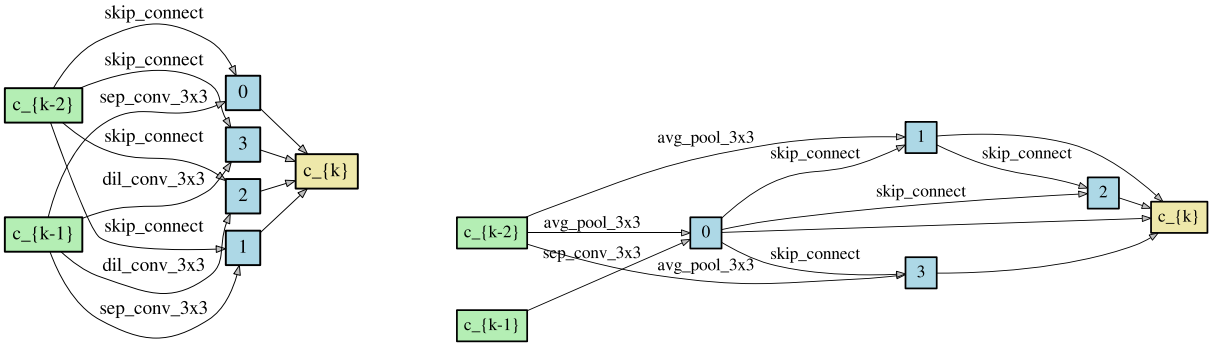
To implement the cell consistency in the search phase, we directly sample a target sub-cell from the DAG by applying the Kronecker delta function on all candidate operations on the edge (i,j), which guarantees that only the operation with the largest weight is activated for the edge (i,j). Thus, as shown in Fig. 4, the cell consistency is guaranteed because the number of edges in the searched cell is the same as that in the target cell. Specifically, we give the formulation of the Kronecker delta function in the discrete cases:

$$\delta_{mn} = \begin{cases} 1, m = n, \\ 0, m \neq n, \end{cases} \tag{4}$$

where the Kronecker delta δ_{mn} is a piecewise function of variables m and n. Then we use the *sifting* property of the Kronecker delt function to sample the most important operation. And the *sifting* property of the Kronecker delta function is: $\sum_m \delta_{mn} a_m = a_n$. When we set $n = argmax(\alpha)$, i.e., the index of the largest architecture weight on an edge. Then according to the "*sifting*" property, all m candidate operations will be applied to the Kronecker delta function δ_{mn}, and the weighted summation corresponds to the operation to get the final output. In other words, the output indicates the operation with the largest architecture weight.

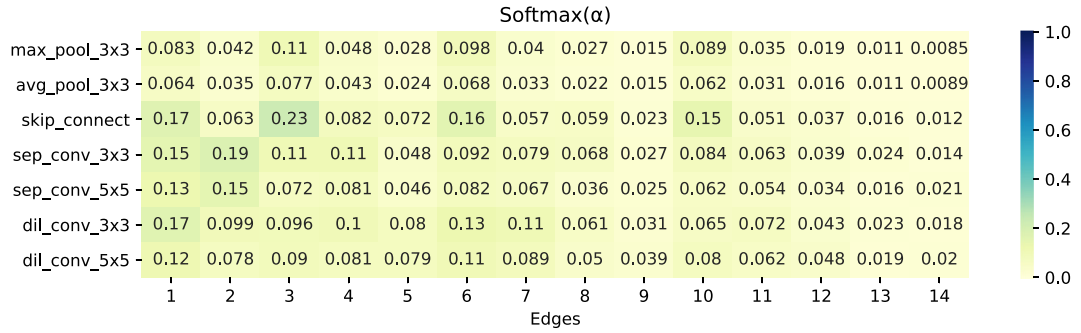
To escape from the local optima, we add extra uniformly-distributed noise on each operation during the sampling process. As the optimization progresses, we gradually anneal down the

[1] We compare different functions for architecture parameters, please refer to the ablation study.

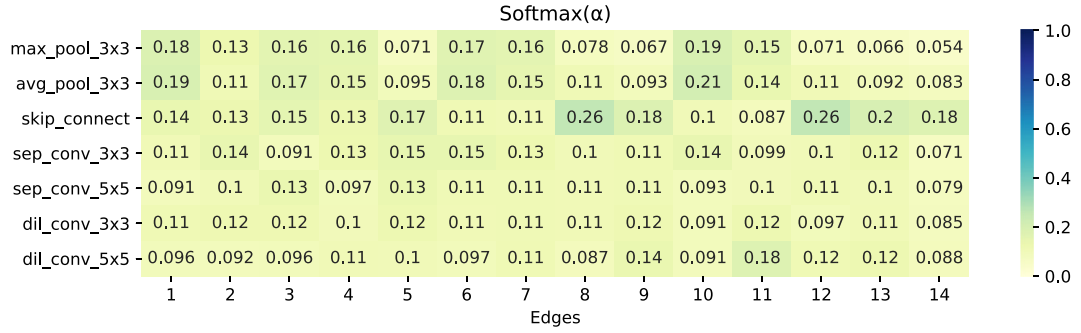


(a) Normal Cell (b) Reduction Cell

Fig. 2. DARTS searched cells on CIFAR-10.



(a) Heatmap of Normal Cell



(b) Heatmap of Reduction Cell

Fig. 3. The heatmap of softmax architecture parameters α for DARTS searched cells on CIFAR-10.

Table 1
The performance degradation of DARTS by simply discretization.

Architecture	Top-1 Valid. Acc. (%)
The whole super-network	88.1
Discretization of super-network	9.9(↓78.2)

range of noise so that the architecture parameters can play a decisive role. Therefore, Eq. (3) can be rewritten as :

$$\bar{o}^{(i,j)}(x^i) = \sum \delta_{mn} Relu(\alpha_o^{(i,j)}) o(x^i),$$
$$s.t. \quad n = argmax(\alpha_o^{(i,j)} + \epsilon_o^{(i,j)}), \tag{5}$$

Table 2
The mathematical form and specific meanings of different α.

Notation	Mathematical Form	Specific Meanings
α.	matrices	the whole architecture parameters of source super-cell
α_s	matrices	the architecture parameters of the sampled target sub-cell
$\alpha_o^{(i,j)}$	vector	the operation weight of a pair of node (i,j)
$\alpha_k^{(i,j)}$	scalar	the weight of the sampled operation k associated with the edge (i,j)

Table 3
Search computation complexity comparison between STC-NAS and others. "L" denotes the cell numbers of super-network. "BS" denotes the batch size. "MEM" denotes the GPU memory consumption. "M" and "T" means the memory and time complexity of target subnet. All are tested on a single NVIDIA V100 GPU.

L	Methods	BS	MEM	GPU-	Complexity	
			(GB)	Days	MEM	Time
-	Subnet	-	-	-	$O(M)$	$O(T)$
8	DARTS	64	9.4	0.75	$O(nM)$	$O(nT)$
	SNAS	64	9.4	1.45	$O(nM)$	$O(nT)$
	GDAS	256	6.8	0.18	$O(M)$	$O(T)$
	STC-NAS	64	**1.8**	0.142	$O(M)$	$O(T)$
	STC-NAS	128	3.1	0.083	$O(M)$	$O(T)$
	STC-NAS	256	5.5	**0.059**	$O(M)$	$O(T)$
20	DARTS	96	80	8.4	$O(nM)$	$O(nT)$
	SNAS	96	83	15.6	$O(nM)$	$O(nT)$
	GDAS	96	16	1.0	$O(M)$	$O(T)$
	STC-NAS	96	**12**	**0.22**	$O(M)$	$O(T)$

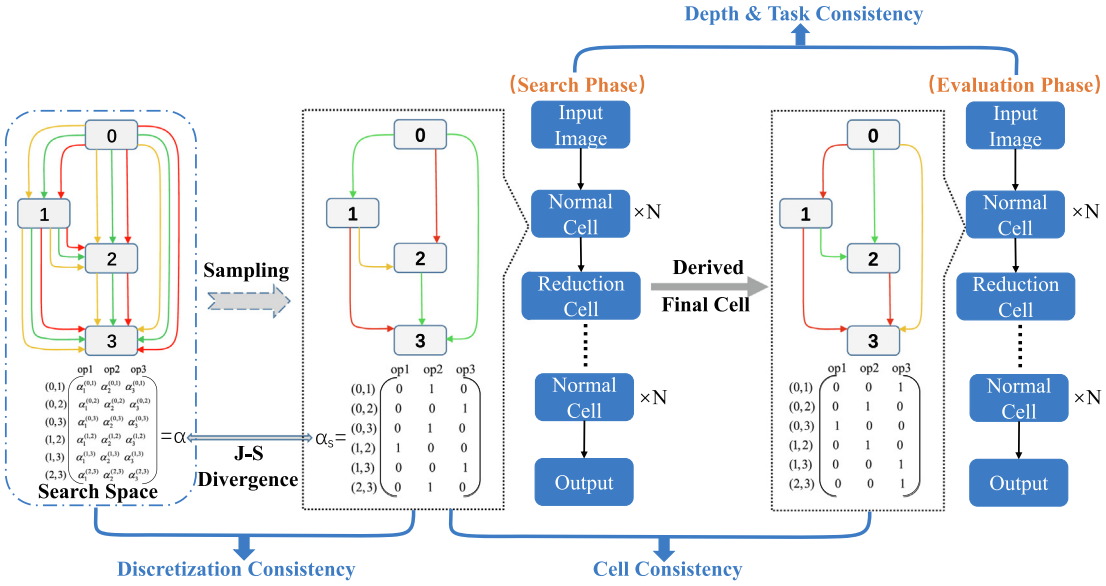


Fig. 4. The main framework of the proposed STC-NAS. We firstly sample a sub-cell whose number of edges is consistent with the target-derived cell. To strengthen the distinction among different candidate operations and make the sampling process stable, we leverage Jensen-Shannon divergence for architecture parameter optimization to guarantee the discretization consistency. Then the sampled sub-cell is stacked to build a super-network that is depth consistent with the target super-network and is directly searched on the target dataset.

where $\epsilon_o^{(i,j)}$ is the random noise whose range is annealed as the epoch increases, and then the Kronecker delta δ_{mn} is adopted to sample the most important operation that is associated with the edge.

3.4. Optimization with Discretization Consistency

To ensure that the architecture parameters are in the feasible area at each iteration, we adopt the Projected Gradient Descent (PGD) algorithm. Firstly, we define the constraint set of architecture parameters as follows:

$$\mathcal{Q} = \{||\alpha^{(i,j)}||_0 = 1, 0 \leqslant \alpha_k^{(i,j)} \leq 1\}, \tag{6}$$

which means that there is only one associated operation being activated for every edge, and the range of α_k is between zero and one. After the gradient descent update, if the point $\alpha_k - \eta\nabla f(\alpha_k)$ is out of the constraint set $\mathcal{Q}$, then we project it back to the feasible area with PGD:

$$\alpha_{k+1} = \mathcal{P}_{\mathcal{Q}}(\alpha_k - \eta\nabla f(\alpha_k)), \tag{7}$$

where η is the learning rate and $\mathcal{P}_{\mathcal{Q}}(\cdot)$ is the projection operator which is also an optimization problem:

$$\mathcal{P}_{\mathcal{Q}}(\alpha_k) = \arg\min_{\alpha_{k+1}\in\mathcal{Q}} \frac{1}{2}||\alpha_{k+1} - \alpha_k||_2^2, \tag{8}$$

i.e., given an initial point α_k, $\mathcal{P}_{\mathcal{Q}}$ tries to find a point $\alpha_{k+1} \in \mathcal{Q}$ which is the closest to α_k.

Moreover, in order to make each candidate operation distinguishable as much as possible, we leverage Jensen-Shannon divergence for architecture parameter optimization to ensure the discretization consistency. JS divergence measures the similarity of two probability distributions. Compared with the Kullback-Leibler (KL) divergence, JS divergence provides a symmetrical and normalized version of KL divergence. So we compute the JS loss using the sampled target sub-cell (i.e., α_s in Fig. 4) and and the source super-cell (i.e., α in Fig. 4), and in this way, the optimization is pushed towards the target sub-cell. The Jensen-Shannon divergence can be computed as:

$$JS(\alpha_s||\alpha) = \frac{1}{2}KL(\alpha_s||\frac{\alpha_s+\alpha}{2}) + \frac{1}{2}KL(\alpha||\frac{\alpha_s+\alpha}{2}), \tag{9}$$

Table 4
Search results on CIFAR-10 and CIFAR-100 and comparison with other state-of-the-art methods. STC-NAS-w/o DepthC means that we search on the shallow super-network without the depth consistency in the same way as other methods. We report the average results for three independent runs of searching with different initial random seeds.

Methods	CIFAR-10		CIFAR-100		Search Time	Search
	Test Err.(%)	Params(M)	Test Err.(%)	Params(M)	(GPU-days)	Algorithm
NASNet-A+cutout [2]	2.65	3.3	-	-	1800	RL
AmoebaNet-B+cutout [11]	2.55±0.05	2.8	-	-	3150	EA
PNAS [34]	3.41±0.09	3.2	-	-	225	SMBO
ENAS+cutout [12]	2.89	4.6	-	-	0.45	RL
DARTS(first order)+cutout [13]	3.00±0.14	3.3	17.76	3.3	4	Gradient
SNAS(moderate constraint) [24]	2.85±0.02	2.8	-	-	1.5	Gradient
GDAS+cutout [25]	2.93	3.4	18.38	3.4	0.21	Gradient
P-DARTS+cutout [17]	2.5	3.4	16.55	3.4	0.3	Gradient
PC-DARTS+cutout [23]	2.57±0.07	3.6	-	-	0.1	Gradient
DATA(M=7)+cutout [26]	2.59	3.4	-	-	1	Gradient
SGAS(Cri.1 avg.) [21]	2.66±0.24	3.7	-	-	0.25	Gradient
FairDARTS [14]	2.54±0.05	3.32±0.46	-	-	0.4	Gradient
SDARTS-ADV [35]	2.61±0.02	3.3	-	-	1.3	Gradient
NASP(7 operations)+cutout [27]	2.83±0.09	3.3	-	-	0.1	Gradient
STC-NAS-w/o DepthC(avg.)	2.48±0.03	3.89±0.11	17.01±0.29	3.82±0.03	**0.059**	Gradient
STC-NAS-w/o DepthC(best)	2.45	3.98	16.59	3.79	**0.059**	Gradient
STC-NAS(avg.)	2.46±0.04	3.88±0.31	16.71±0.19	3.85±0.24	0.22	Gradient
STC-NAS(best)	**2.42**	3.86	**16.45**	4.17	0.22	Gradient

where $KL(\cdot)$ is the Kullback-Leibler (KL) divergence that is computed as $KL(P||Q) = \sum(P\log P - P\log Q)$.

Thus the overall objective of STC-NAS is as follows:

$$\begin{aligned} &\min \quad \mathcal{L}_{val}(w^*(\alpha_s), \alpha_s) + \lambda JS(\alpha_s||\alpha), \\ &s.t. \quad w^*(\alpha_s) = \arg\min_w \mathcal{L}_{train}(w_s, \alpha_s). \end{aligned} \tag{10}$$

4. Experiments

4.1. Datasets

We conduct our method on four popular image classification datasets including CIFAR-10, CIFAR-100, ImageNet, and NAS-Bench-201 [32] or also called NATS-Bench [33]. Both of the CIFAR-10 and CIFAR-100 datasets have 50K training RGB images and 10K testing RGB images with a fixed spatial resolution of 32×32. The ILSVRC2012 ImageNet dataset contains 1.28M training and 50K validation images with 1000 object categories. We set the input image size as 224×224 and the number of multi-adds operations is strict to less than 600M. NAS-Bench-201 or NATS-Bench is a benchmark for almost up-to-date NAS algorithms, and it contains 15,625 neural architectures with detailed information including accuracy, loss, FLOPs, etc.

4.2. Search and Evaluation on CIFAR datasets

We conduct a bi-level optimization to alternately optimize the network weights on half of the training set and update the architecture parameters on the other half of the training set as valuation set, respectively. In particular, we use SGD optimizer to update network weights with initial learning rate 0.025, the minimum learning rate 0.001, momentum 0.9, and weight decay 3×10^{-4}. And the architecture parameters are optimized by Adam with initial learning rate 3×10^{-4}, momentum (0.5, 0.999), but we set the architecture weight decay to be zero to prevent the weight of not-sampled operations from dropping. In addition, we set the initial range of the uniformly-distributed noise as 0.3 which gradually decreases to zero[2], and the coefficient λ of JS loss increases from zero to two as the searching epoch progresses.[3]

STC-NAS significantly improves the search efficiency due to the cell consistency. As can be seen from Table 3, our method can improve the batch size to 256 while the memory consumption is still less than DARTS, and the search time is only 0.059 GPU-days which is about 12× speedup over DARTS. Besides, benefited from depth consistency, the overall search time of our method is within 0.22 GPU-days, while DARTS requires 8.4 GPU-days and 80GB GPU memory even on such a small dataset CIFAR-10.

In the evaluating phase, the searched normal cell and the reduction cell are stacked to build the target super-network, which consists of 20 layers with initial channel size 36. The network is trained from scratch for 600 epochs with batch size 96. We use an SGD optimizer with a weight decay of 3×10^{-4} and a momentum of 0.9. The initial learning rate starts from 0.025 and follows the cosine annealing strategy to a minimum of 0. The evaluation results on CIFAR-10 and CIFAR-100 are listed in Table 4.

The results show that our method is very efficient and can obtain a relatively better architecture. Comparing with other methods under the same experimental setting, i.e., searching a super-network of 8 cells, the average test error on CIFAR-10 and CIFAR-100 is 2.48% and 17.01% respectively, indicating that our method can search a stable architecture even with different seed initialization. Moreover, under the consistency of depth and task dataset, STC-NAS achieves 2.46% and 16.71% average test error on CIFAR-10 and CIFAR-100 and the best result is 2.42% and 16.45% respectively, which outperforms the state-of-the-art methods.

4.3. Search and Evaluation on ImageNet

We directly search on ImageNet with the target super-network which consists of 14 cells and 48 initial channels. Considering that ImageNet is extremely large, we randomly sample 10% images

[2] We compare the different range of initial sampling noise, please refer to the ablation study.

[3] We compare the performance of different coefficient λ, please refer to the ablation study.

Table 5
Search results on ImageNet and comparison with other state-of-the-art methods. STC-NAS-w/o DepthC&TC means that we search without the depth and task dataset consistency, i.e., the results are transferred from CIFAR-10 in the same way as other methods. *means the search space is layer-wise (i.e., based on blocks of MobileNetV2) which is different from the cell-based search space of DARTS.

Methods	Test Err. (%)		Params	Flops	Search Cost	Search
	Top-1	Top-5	(M)	(M)	(GPU-days)	Algorithm
NASNet-A [2]	26.0	8.4	5.3	564	1800	RL
AmoebaNet [11]	24.3	7.6	6.4	570	3150	EA
PNAS [34]	25.8	8.1	5.1	588	225	SMBO
DARTS [13]	26.7	8.7	4.7	574	4	Gradient
SNAS(mild constraint) [24]	27.3	9.2	4.3	522	1.5	Gradient
GDAS [25]	26.0	8.5	5.3	581	0.21	Gradient
P-DARTS [17]	24.4	7.4	4.9	577	0.3	Gradient
SGAS(Cri.1 best) [21]	24.2	**7.2**	5.3	585	0.25	Gradient
PC-DARTS [23]	25.1	7.8	5.3	586	0.1	Gradient
DATA [26]	24.9	8.0	5.0	588	1	Gradient
FairDARTS-B [14]	24.9	7.5	4.8	541	0.4	Gradient
ProxylessNAS [28]*	24.9	7.5	7.1	465	8.3	Gradient
DSNAS [29]*	25.7	8.1	-	324	17.5	Gradient
NASP(7 operations) [27]	27.2	9.1	4.6	-	0.1	Gradient
STC-NAS-w/o DepthC&TC	24.3	7.4	5.3	596	**0.059**	Gradient
STC-NAS	**24.2**	7.3	5.3	588	0.625	Gradient

Table 6
Search results on NAS-bench-201 and NATS-Bench. We report the average performance for three independent runs of searching. "Optimal" indicates the highest accuracy for each dataset on NAS-Bench-201 or also NATS-Bench.

	Methods	Search	CIFAR-10		CIFAR-100		ImageNet-16-120	
		(seconds)	validation	test	validation	test	validation	test
	Optimal	N/A	91.61	94.37	73.49	73.51	46.77	47.31
NAS- Bench- 201	RSPS [36]	8007.13	80.42±3.58	84.07±3.61	52.12±5.55	52.31±5.77	27.22±3.24	26.28±3.09
	DARTS [13]	11625.77	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 [25]	31609.80	89.89±0.08	93.61±0.09	**71.34±0.04**	**70.70±0.30**	41.59±1.33	41.71±0.98
	SETN [37]	34139.53	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 [12]	14058.80	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
NATS-Bench	RSPS [36]	8007.13	87.60±0.61	91.05±0.66	68.27±0.72	68.26±0.96	39.73±0.34	40.96±0.36
	DARTS(1st)	11625.77	49.27±13.44	59.84±7.84	61.08±4.37	61.26±4.43	38.07±2.90	37.88±2.91
	DARTS(2nd)	-	58.78±13.44	65.38±7.84	59.48±5.13	60.49±4.95	37.56±7.10	36.79±7.59
	GDAS [25]	31609.80	89.68±0.72	93.23±0.58	68.35±2.71	68.17±2.50	39.55±0.00	39.40±0.00
	SETN [37]	34139.53	90.00±0.97	92.72±0.73	69.19±1.42	69.36±1.72	39.77±0.33	39.51±0.33
	ENAS [12]	14058.80	**90.20±0.00**	**93.76±0.58**	70.21±0.71	70.67±0.62	40.78±0.00	41.44±0.00
	STC-NAS	**3600**	89.83±0.04	93.41±0.2	70.04±0.63	70.43±0.21	**44.45±0.16**	**45.03±0.36**

from each class as the training and validation dataset to optimize the network weights and architecture parameters respectively. We use SGD optimizer with an initial learning rate of 0.5, the minimum learning rate of 0.001, momentum 0.9, and weight decay 3×10^{-4}. And the Adam optimizer is used to update architecture parameters with an initial learning rate of 3×10^{-4} and momentum (0.5, 0.999). We search for 50 epochs in total on ImageNet to obtain the final architecture.

In the evaluation phase, we train the super-network of 14 cells and 48 initial channels for 250 epochs with a batch size of 1024 on ImageNet. We utilize the SGD optimizer with momentum 0.9 and initial learning rate 0.5 which is decayed by cosine strategy of 3×10^{-5}. From Table 5, we can see that by directly searching on ImageNet with the target super-network within 0.625 GPU-days, STC-NAS achieves state-of-the-art performance with 24.2% top-1 test error and 7.3% top-5 test error. In addition, we also transfer the architecture searched on CIFAR-10 to ImageNet to compare with other transfer-based methods. STC-NAS achieves 24.3% top-1 and 7.4% top-5 test error, indicating good transferability of the searched architecture.

4.4. Search and Evaluation on NAS-Bench-201

In NAS-Bench-201 [32] or also called NATS-Bench [33], the super-network is also stacked by cells but it only needs to search for normal cells and maintain the reduction cells as residual blocks with a stride of two. Each normal cell includes four nodes and five associated operations on every edge. We search for 50 epochs on CIFAR-10 and then index the accuracy of the searched architecture on the three datasets CIFAR-10, CIFAR-100, and Image-Net-16-120, respectively. We experiment three times independently with different initial random seeds, and the results are shown in Table 6. Comparing with other methods, STC-NAS achieves comparable performance to ENAS on CIFAR-10 and GDAS on CIFAR-100, and best results on ImageNet-16-120. Besides, STC-NAS only requires one hour to finish the search due to directly searching for the target architecture, which is much faster than the other methods.

5. Ablation Study

5.1. Analysis of Search Efficiency

During the search process, STC-NAS satisfies the source-target consistency. That is, we only need to search for the super-network consistent with the target model, which is extremely time- and resource-efficiency. From Table 3, we can see that the computation complexity of our method is the same as the target model, while DARTS and SNAS require n times resources because they optimize all of the candidate operations in the search phase. Though GDAS also optimizes the subnetwork, it leverages Gumble-softmax to compute the architecture parameters so that it needs more epochs to anneal down the temperature τ and the

fixed reduction cells may not optimal thus requires slightly more memory consumption.

5.2. The Ranking Correlation

The correlation between the performance of the search and evaluation phase is an important factor in the stability of the searching algorithm. The Kendall τ metric is a common measurement of the correlation between two rankings. We calculate the Kendall τ on our two experimental settings according to whether considering depth consistency. We repeat the experiments eight times with random seeds on CIFAR-10 to obtain the accuracy ranking of search and then retrain from scratch for each searched architecture to obtain the accuracy ranking of evaluation.

As can be seen from Fig. 5, the Kendall τ of searching with depth consistency is 0.64, which is better than searching on the shallower super-network. Besides, the average accuracy is also improved by depth consistency. So we can summarize that searching with source-target consistency is important for both correlation and accuracy.

5.3. The Effectiveness of the Relu Function

The weights of operations on an edge can be calculated by various functions, such as Softmax, Sigmoid and Relu. To demonstrate the effects of Softmax, Sigmoid and Relu on the performance of the searched architecture, we repeat the experiments for each function three times with different random seeds on CIFAR-10. From Fig. 6, we can see that the performances of Softmax and Sigmoid function are inferior to the Relu activation function. This is because the Softmax function introduces implicit competition among different operations thus preferring non-parameterized operations, while the Sigmoid function may cause the vanishing gradient problem when approaching the saturation zone. Using the Relu function can avoid the above mentioned problems and achieve better performance.

5.4. The Range of Sampling Noise

During the search process, we add extra uniformly-distributed noise to the architecture parameters when sampling a sub-cell from the whole DAG and the range of noise should be gradually annealed down to zero as the epoch increases so that better architectures are not disturbed. Here we diagnose what the initial range of noise is suitable. Fig. 7 demonstrates that the performance is extremely terrible without the noise because the searching is trapped in a local solution without exploration. Besides, the initial sampling noise of 0.3 performs better than the range of 0.1 or 0.5. It can be explained that the smaller range may have no enough exploration and in contrast, the larger range may not converge well to a better solution.

5.5. The Coefficient of JS-Loss

The Jensen-Shannon divergence loss is introduced to narrow the distribution gap between the sampled target sub-cell and the source super-cell. As the optimization progresses, the weight distribution of the super-cell gradually approximates the sampled target cell, improving performance by closing the discretization gap. Here we explore the influence of different coefficients of regulation. Static coefficient means that the coefficient λ is constant during the search process, while dynamic means that the coefficient λ increases as the search epoch increasing. From Table 7, we can see that when discarding the Jensen-Shannon divergence (λ=0), the performance is degraded, demonstrating the effectiveness of the Jensen-Shannon divergence. The suitable regulation weight is cap-

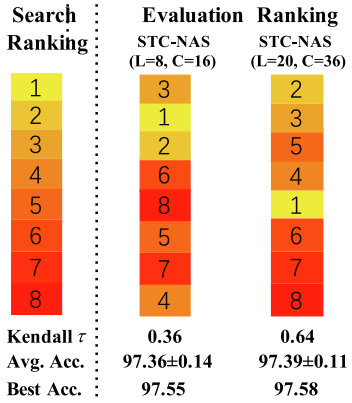


Fig. 5. The Kendall τ for architecture ranking between the search and evaluation phase. Architecture rankings are obtained from 8 independent runs of searching on CIFAR-10.

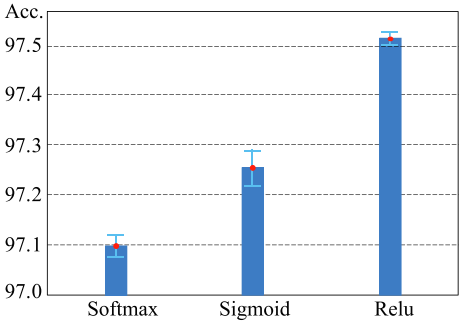


Fig. 6. Comparison of Softmax, Sigmoid and Relu for architecture parameters. The mean and the standard deviation are obtained from three independent runs of searching on CIFAR-10.

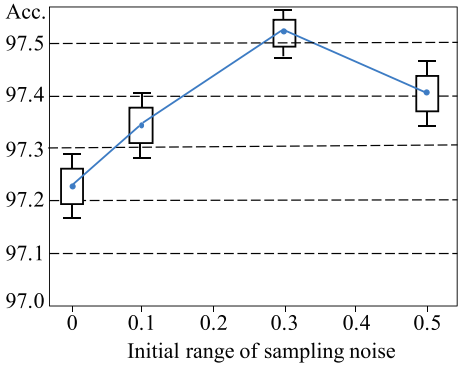


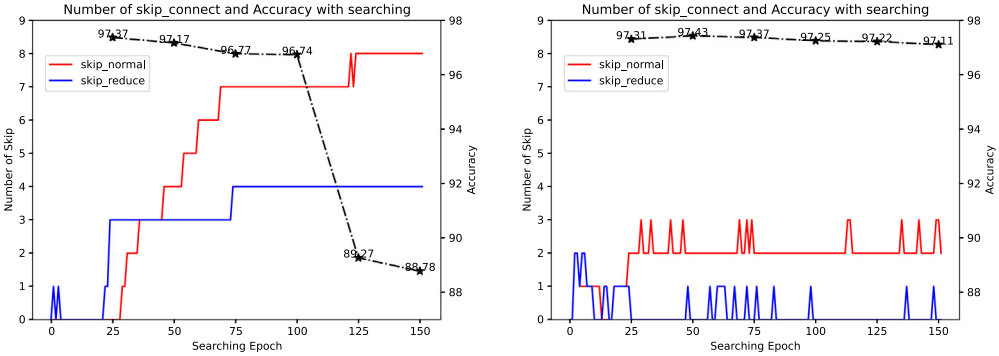
Fig. 7. Comparison of the different initial range of sampling noise. The mean and the standard deviation are obtained from three independent runs of searching on CIFAR-10.

Table 7
Comparison of different coefficients λ of JS-loss. The mean and the standard deviation are obtained from three independent runs of searching on CIFAR-10.

	Coefficient λ	Acc.
Static	0	97.19±0.12
	1	97.27±0.07
	2	97.31±0.11
Dynamic	0→1	97.16±0.26
	1→2	97.34±0.02
	0→2	**97.52±0.03**

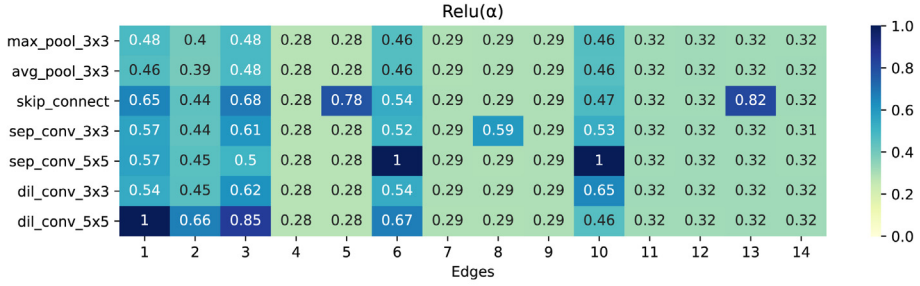
Table 8
Performance comparison for lacking any one of the four consistencies. The mean and the standard deviation are obtained from three independent runs of searching on CIFAR-100.

Cell	Discretization	Depth/Width	Task Dataset	Accuracy
✓	✓	✓	✓	**83.29±0.19**
×	✓	✓	✓	82.35±0.28(↓ 0.94)
✓	×	✓	✓	82.57±0.19(↓ 0.72)
✓	✓	×	✓	82.99±0.29(↓ 0.30)
✓	✓	✓	×	83.04±0.18(↓ 0.25)

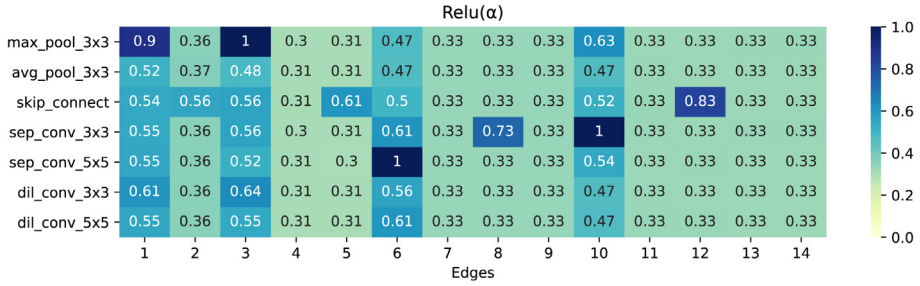


(a) DARTS (b) STC-NAS

Fig. 8. The number of skip-connect and accuracy with searching more epochs on CIFAR-10..



(a) Heatmap of Normal Cell



(b) Heatmap of Reduction Cell

Fig. 9. The heatmap of architecture parameters α for STC-NAS searched cells on CIFAR-10..

able of reducing the variance, and the coefficient λ of JS loss dynamically increasing from zero to two obtains the highest accuracy for our method. In particular, the optimization mainly focuses on architecture parameters at the early stage, and then the loss of distribution divergence gradually plays a more important role in narrowing the gap.

5.6. The Importance of Consistency

Our STC-NAS satisfies the consistency of cell, discretization, depth/width, and task dataset during the search process. Here we investigate that each of the four consistencies is indispensable for improving performance. We conduct the ablation study on

the CIFAR-100 dataset. As shown in Table 8, , STC-NAS achieves the best performance with the aforementioned four consistencies. Meanwhile, the accuracy is reduced for lacking any one of the four consistencies. The inconsistency of cell or discretization brings a more noticeable decrease than the task dataset inconsistency or the depth/width inconsistency, showing the effectiveness of the proposed method.

5.7. Analysis of Overfitting Collapse

Previous DARTS approaches rely on early-stop to mitigate skip-connect aggregation introduced by the softmax function. STC-NAS replaces the softmax with the ReLU function, so our method does not have such overfitting issue. We evaluate the search process with more than 50 epochs, i.e., 150 epochs, and obtain the number of skip-connect in the derived cell for each epoch. We train the searched architecture for every 25 epoch from scratch to obtain the final accuracy. As shown in Fig. 8, the solid red line and solid blue line represent the number of skip-connect for normal cell and reduction cell respectively, and the dotted black line represents the accuracy of the searched architecture every 25 epoch. From Fig. 8, we can see that for DARTS method, the number of skip-connect becomes more and more dominant as the search epochs increase, and finally the normal cell has only skip-connect in the last 25 epochs, which significantly downgrades the performance. However, our STC-NAS does not experience the phenomenon of skip-connect aggregation and the number of skip-connect is always less than three. The accuracy of STC-NAS always outperforms DARTS and does not drop dramatically even though searching with more epochs. The highest accuracy of STC-NAS occurs at the 50th epoch, suggesting that no early-stop is required to avoid overfitting during the search.

5.8. Visualization

Here we visualize the heatmap of architecture parameters for STC-NAS searched cells on CIFAR-10. From Fig. 9, we can see that the weights α of the selected edge are close to the one-hot distribution, i.e., the weights α of obtained operations are greater than those discarded operations, thus mitigating the discretization inconsistency. However, as can be seen from Fig. 3, the distribution of architecture parameters in DARTS is too close to select an operation that is significantly superior to others, thus bringing the discretization inconsistency.

We also visualize the best cells directly searched on CIFAR-10 in Fig. 10, the best cells directly searched on CIFAR-100 in Fig. 11 and the best cells directly searched on ImageNet in Fig. 12, respectively.

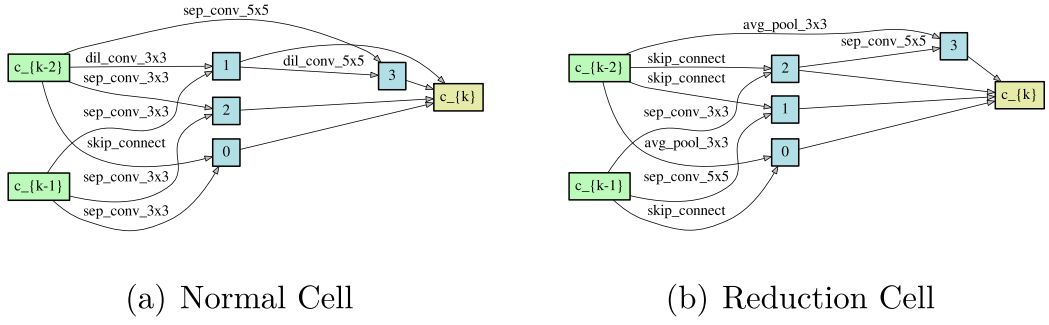


Fig. 10. Our best searched cells on CIFAR-10.

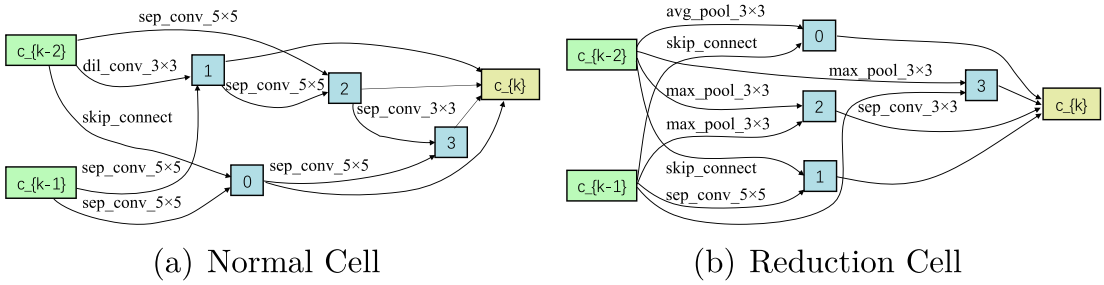


Fig. 11. Our best searched cells on CIFAR-100.

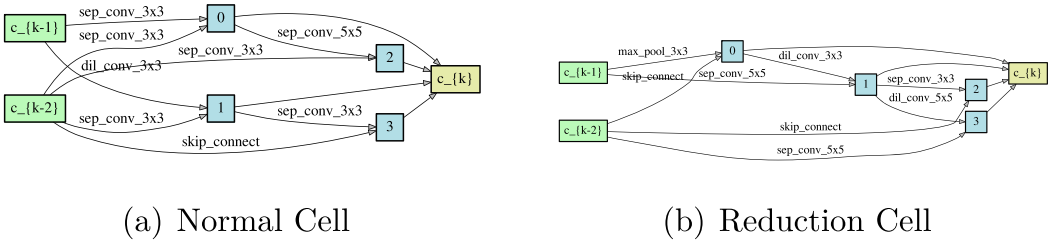


Fig. 12. Our best searched cells on ImageNet.

6. Conclusion

In this paper, we revisit the inconsistency issues in existing differentiable NAS methods and propose to implement fast differentiable architecture search with source-target consistency. The search process of STC-NAS is very fast and memory-efficient due to the guarantee of cell and discretization consistency, so that facilitates to directly search the target super-network on large datasets such as ImageNet. Experimental results demonstrate the searched networks achieve significantly better performance in comparison with state-of-the-art methods.

CRediT authorship contribution statement

Zihao Sun: Conceptualization, Methodology, Software, Validation, Formal-analysis, Investigation, Data-curation, Writing-original-draft, Writing-review-editing, Visualization. **Yu Hu:** Conceptualization, Methodology, Formal-analysis, Resources, Writing-original-draft, Writing-review-editing, Supervision, Funding-acquisition. **Longxing Yang:** Conceptualization, Methodology, Software, Writing-review-editing. **Shun Lu:** Conceptualization, Methodology, Software, Writing-review-editing. **Jilin Mei:** Conceptualization, Methodology, Writing-review-editing. **Yinhe Han:** Resources, Writing-review-editing, Funding-acquisition. **Xiaowei Li:** Writing-review-editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under grant No. 2018AAA0102701 and in part by the State Key Laboratory of Computer Architecture (ICT, CAS) under Grant No. CARCH5203.

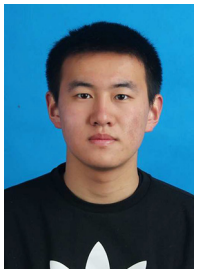
References

[1] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, in: ICLR, 2017..
[2] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: CVPR, 2018. .
[3] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, J. Sun, Detnas: Backbone search for object detection, in: NeurIPS, 2019..
[4] G. Ghiasi, T.-Y. Lin, Q.V. Le, Nas-fpn: Learning scalable feature pyramid architecture for object detection, in: CVPR, 2019. .
[5] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, J. Shlens, Searching for efficient multi-scale architectures for dense image prediction, in: NeurIPS, 2018..
[6] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A.L. Yuille, L. Fei-Fei, Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation, in: CVPR, 2019. .
[7] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, arXiv preprint arXiv:1611.02167 (2016)..
[8] I. Bello, B. Zoph, V. Vasudevan, Q.V. Le, Neural optimizer search with reinforcement learning, in: ICML, 2017..
[9] E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, J. Tan, Q.V. Le, A. Kurakin, Large-scale evolution of image classifiers, in: ICML, 2017..
[10] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search, in: ICLR, 2018. .
[11] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in: AAAI, 2019. .
[12] H. Pham, M. Guan, B. Zoph, Q. Le, J. Dean, Efficient neural architecture search via parameters sharing, in: ICML, 2018..
[13] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, in: ICLR, 2018..
[14] X. Chu, T. Zhou, B. Zhang, J. Li, Fair darts: Eliminating unfair advantages in differentiable architecture search, in: ECCV, 2020. .
[15] Y. Tian, C. Liu, L. Xie, J. Jiao, Q. Ye, Discretization-aware architecture search, arXiv preprint arXiv:2007.03154 (2020)..
[16] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: CVPR, 2016. .
[17] X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: Bridging the depth gap between search and evaluation, in: ICCV, 2019..
[18] B. Recht, R. Roelofs, L. Schmidt, V. Shankar, Do imagenet classifiers generalize to imagenet, in: ICML, 2019..
[19] Y. Li, Z. Yang, Y. Wang, C. Xu, Adapting neural architectures between domains, in: NeurIPS, 2020. .
[20] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, Z. Chen, L. Wang, A. Xiao, J. Chang, X. Zhang, et al., Weight-sharing neural architecture search: A battle to shrink the optimization gap, arXiv preprint arXiv:2008.01475 (2020)..
[21] G. Li, G. Qian, I.C. Delgadillo, M. Muller, A. Thabet, B. Ghanem, Sgas: Sequential greedy architecture search, in: CVPR, 2020. .
[22] A. Noy, N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes, L. Zelnik, Asap: Architecture search, anneal and prune, in: NeurIPS, 2020..
[23] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, H. Xiong, Pc-darts: Partial channel connections for memory-efficient architecture search, in: ICLR, 2019. .
[24] S. Xie, H. Zheng, C. Liu, L. Lin, Snas: stochastic neural architecture search, in: ICLR, 2018. .
[25] X. Dong, Y. Yang, Searching for a robust neural architecture in four gpu hours, in: CVPR, 2019. .
[26] J. Chang, X. Zhang, Y. Guo, G. Meng, S. Xiang, C. Pan, Data: Differentiable architecture approximation, in: NeurIPS, 2019. .
[27] Q. Yao, J. Xu, W.-W. Tu, Z. Zhu, Efficient neural architecture search via proximal iterations, in: AAAI, 2020. .
[28] H. Cai, L. Zhu, S. Han, Proxylessnas: Direct neural architecture search on target task and hardware, in: ICLR, 2018..
[29] S. Hu, S. Xie, H. Zheng, C. Liu, J. Shi, X. Liu, D. Lin, Dsnas: Direct neural architecture search without parameter retraining, in: CVPR, 2020. .
[30] X. Dong, M. Tan, A.W. Yu, D. Peng, B. Gabrys, Q.V. Le, Autohas: Efficient hyperparameter and architecture search, in: ICLR Workshop, 2021. .
[31] G. Bender, H. Liu, B. Chen, G. Chu, S. Cheng, P.-J. Kindermans, Q.V. Le, Can weight sharing outperform random architecture search? an investigation with tunas, in: CVPR, 2020. .
[32] X. Dong, Y. Yang, Nas-bench-201: Extending the scope of reproducible neural architecture search, in: ICLR, 2020. .
[33] X. Dong, L. Liu, K. Musial, B. Gabrys, Nats-bench: Benchmarking nas algorithms for architecture topology and size, in: IEEE TPAMI, 2021. .
[34] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, K. Murphy, Progressive neural architecture search, in: ECCV, 2018. .
[35] X. Chen, C.-J. Hsieh, Stabilizing differentiable architecture search via perturbation-based regularization, in: ICML, 2020..
[36] L. Li, A. Talwalkar, Random search and reproducibility for neural architecture search, in: UAI, 2020. .
[37] X. Dong, Y. Yang, One-shot neural architecture search via self-evaluated template network, in: ICCV, 2019..

Zihao Sun received the B.Eng. degree in School of Automation from University of Science & Technology Beijing, China, in 2018. He is currently pursuing the Ph. D. degree in computer science from the Research Center for Intelligent Computing Systems, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) and the University of Chinese Academy of Sciences (UCAS), Beijing. His current research interests include neural architecture search and deep learning.

Yu Hu received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 1997, 1999, and 2003, respectively. She is currently a Professor at the Research Center for Intelligent Computing Systems, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) and a Professor at the University of Chinese Academy of Sciences (UCAS). Her research interests generally include autonomous driving, deep learning and algorithm acceleration.

Longxing Yang received the B.Eng. degree in School of Software from Beijing Institute of Technology, China, in 2018. He is currently pursuing the Ph.D. degree in computer science from the Research Center for Intelligent Computing Systems, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) and the University of Chinese Academy of Sciences (UCAS), Beijing. His research interests include neural architecture search and neural network compression.



Shun Lu received the B.Eng. degree in School of Automation from University of Science & Technology Beijing, China, in 2019. He is currently pursuing the Ph. D. degree in computer science from the Research Center for Intelligent Computing Systems, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) and the University of Chinese Academy of Sciences (UCAS), Beijing. His research interests include neural architecture search.

Jilin Mei received the B.Eng. degree from University of Electronic Science and Technology of China, Chengdu, China, in 2014 and Ph.D. degree in School of Electronics Engineering and Computer Science from Peking University, Beijing, China, in 2020. He is currently working as a special research assistant (postdoctoral fellow) at the Research Center for Intelligent Computing Systems, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). His research interests include autonomous driving and semantic segmentation.

Yinhe Han received the B.Eng. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2001, and the Ph.D. degree from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2006. He is currently a Professor at the Research Center for Intelligent Computing Systems, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) and a Professor at the University of Chinese Academy of Sciences (UCAS). His current research interests include computer architecture and chip design for intelligent robot.

Xiaowei Li received the B.Eng. and M.Eng. degrees in computer science from the Hefei University of Technology, Hefei, China, in 1985 and 1988, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 1991. He is currently the Deputy (Executive) Director of the State Key Laboratory of Computer Architecture, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) and a Professor at the University of Chinese Academy of Sciences (UCAS). His current research interests include VLSI testing, design verification, and dependable computing.