Powering One-shot Topological NAS with Stabilized Share-parameter Proxy

Ronghao Guo[1*], Chen Lin[2*], Chuming Li[2], Keyu Tian[1],
Ming Sun[2], Lu Sheng[1⋆], and Junjie Yan[2]

[1] College of Software, Beihang University
{16211042,17375491,lsheng}@buaa.edu.cn
[2] SenseTime Research
{linchen,lichuming,sunming1,yanjunjie}@sensetime.com

Abstract. One-shot NAS method has attracted much interest from the research community due to its remarkable training efficiency and capacity to discover high performance models. However, the search spaces of previous one-shot based works usually relied on hand-craft design and were short for flexibility on the network topology. In this work, we try to enhance the one-shot NAS by exploring high-performing network architectures in our large-scale Topology Augmented Search Space (*i.e*, over 3.4×10^{10} different topological structures). Specifically, the difficulties for architecture searching in such a complex space has been eliminated by the proposed stabilized share-parameter proxy, which employs Stochastic Gradient Langevin Dynamics to enable fast shared parameter sampling, so as to achieve stabilized measurement of architecture performance even in search space with complex topological structures. The proposed method, namely Stablized Topological Neural Architecture Search (ST-NAS), achieves state-of-the-art performance under Multiply-Adds (MAdds) constraint on ImageNet. Our lite model ST-NAS-A achieves 76.4% top-1 accuracy with only 326M MAdds. Our moderate model ST-NAS-B achieves 77.9% top-1 accuracy just required 503M MAdds. Both of our models offer superior performances in comparison to other concurrent works on one-shot NAS.

Keywords: Stablized One-shot NAS, Network Topology

1 Introduction

Significant progress made by convolution neural networks (CNN) in challenging computer vision tasks has raised the demand to design powerful neural networks. Instead of manually design, Neural architecture search (NAS) has demonstrated great potentials in recent years. Early works of NAS by Real *et al*[29, 28] and Elsken *et al*[11] achieved promising results but can only be applied to small datasets due to their large computation expenses. To this end, one-shot based

* First two authors contributed equally.
⋆ Corresponding author: Lu Sheng

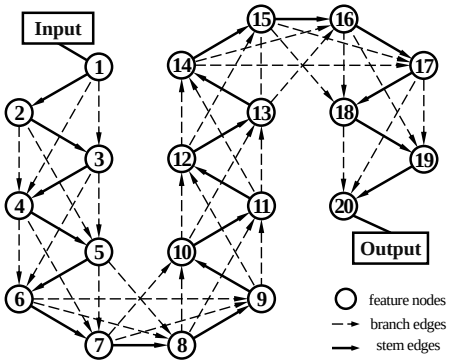
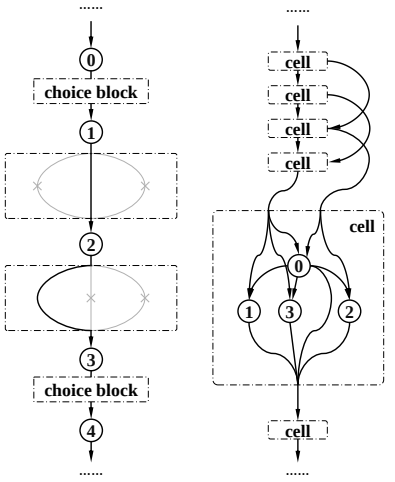


Fig. 1. An instance of our topology augmented search space. It contains over 3.4×10^{10} different network topologies which enables us to explore complex network typologies. Solid line denotes the chain-structured stem edges, and dotted line represents branch edges which connects feature maps with depth difference 2 or 3.



(a) SPOS space (b) DARTS space

Fig. 2. Two typical search spaces in previous work. Fig. (a) shows a chain-structured space while Fig. (b) shows a cell-based search space.

methods have drawn much interest thanks to its promising training efficiency and remarkable ability to discover high-performing models. One-shot method usually utilizes a hyper-network, which subsumes all architectures in the search space, and use shared weights to evaluate different architectures.

However, the search space of previous works (*e.g*, shown in Fig. 2) were usually carefully designed and did not enjoy too much flexibility on the network topology. For example, as one of mostly applied search spaces in the one-shot literature, the chain-structured search space[14] has sequentially connected intermediate feature maps, between which the edges are chosen from a set of computation operations. Networks with better operations can be discovered on this search space, but the network topology remains trivial. However, previous works [10, 17] on network architecture design proved that complex topology will tremendously enhance the performance of deep learning models. We argue that complex topological structures added in search space will improve the performance of the searched network architectures as shown in Table 1.

In this work, we are interested in exploring complex network typologies with one-shot method. We propose a novel network architecture search space shown in Fig. 1 which contains over 3.4×10^{10} different network topologies, enabling the discovery of complex topology networks. The search space is obtained by introducing numerous computation modules as edges between nodes. A topology based architecture sampler is also introduced to sample architectures during one-shot training stage from the hyper-network. However, the great diversity introduced by topologies brings difficulties to the one-shot approach. Specifi-

cally, we observe high variance of performance estimation through the one-shot shared parameters in two cases: estimation through shared parameters at different epochs of a single run and estimation through shared parameters obtain in different runs. Zhang *et al*[43] explore the reason behind the variance of ranking under weight sharing strategy. Thus the ranking ability of shared parameters is compromised.

To eliminate the interference of complex topologies, we estimate the expectation of architecture performance in additional training epochs of hyper-network via multiple samples of shared parameters. An fast weights sampling methods based on Stochastic Gradient Langevin Dynamics is developed to sample shared parameters efficiently.

The resulted Stabilized Topological Neural Architecture Search (ST-NAS) achieves compatible performance with the state-of-the-art NAS method. The resulted architecture ST-NAS-A obtains 76.4% top-1 accuracy with only 326M MAdds. A larger architecture ST-NAS-B obtains 77.9% top-1 accuracy with around 503M MAdds.

To summarize, our main contributions are as follows:

1. We introduce a topology augmented neural architecture search space that enables the discovery of efficient architectures with complex topology.
2. To relieve the complex topology's interference on model ranking, we modified model evaluation based on the expectation of the sharing parameters' performance.
3. We empirically demonstrate improvements on ImageNet classification under the same MAdds constraints compared with previous work, and show that the searched architectures transfer well to COCO object detection.

2 Related Work

Recently, auto machine learning methods have received a lot of attention due to its ability to design augmentation [9, 22], loss function [19] and network architectures [45, 28, 4, 44, 25, 14, 13, 21, 20].

Early neural architecture search (NAS) works normally involves reinforcement learning [1, 45, 44, 46, 13, 36] or evolution algorithm [29, 26] to search for high-performing network architectures. However, these methods are usually computationally expensive which limits its uses in real scenarios.

Recent attentions have been focused on alleviating the computation cost via weight sharing method. This method usually contains a single training process of an over parameterized hyper-network which subsumes all candidate models, *i.e*, weights of the same operators are shared across different sub-models. Notably, Liu *et al*[25] proposes continuous relaxations which enables optimizing network architectures with gradient decent, Cai *et al*[5] proposes a proxy-less method to search on target datasets directly and Bender *et al*[2] introduces one-shot method to decouple training and searching stages. Our NAS work take the use of the weight sharing hyper-network but relieve the variance during model training.

Early hand-craft neural networks [15, 35, 33] tend to stack repeated motifs. Works in [34, 15, 17, 16] introduce different manual designed network topologies and result in performance gain.

Motivated by manual designed architectures [15, 35, 33], a widely used search space in works [45, 25, 44, 26, 13] are proposed to search for such motifs, dubbed cells or blocks, rather than all possible architectures. This search space is called cell-based space. Another widely used search space adopted in [5, 14, 36, 42] is called chain-structured space. This space sequentially stacks several operation layers where each layer serves its output as next layer's input. NAS methods are adopted to search for operation layers in different position of this space. Work in [41] explores random wiring networks with less human prior and achieves comparable performance with manual designed networks.

3 Approach

Methods for NAS usually consist of three basic components: search space, performance estimation and search strategy. In this section, we first introduce our novel Topology Augmented Search Space and a new sampling strategy for hyper-network training in this particular space. Secondly, we provide new model performance estimation approach to relieve the variance of model ranks during the training of hyper-network. Finally the evolution algorithm for network search is described.

3.1 Topology Augmented Search Space

Motivation To demonstrate the improvement of complex topology against a sequential structure, we take ResNet-18 as a baseline and shows a subtle change on the topology obtains obvious performance boost. We randomly add 4 residual blocks to connect the feature maps of blocks in ResNet-18's [15] chain structure with 3 random seeds, and rescale the width to keep the same FLOPs, the results are in Table 1. The 3 complex structures imply the great potential of topology-based structure search.

Search Space A neural network is denoted as a directed acyclic graph (DAG) defined by E, V, where the node $v_i \in V$ indicates the feature connected by edge

Architecture	Res-18	Rand0	Rand1	Rand2
Accuracy (%)	70.2	71.5	72.0	69.6

Table 1. The accuracy of ResNet-18 and three networks with 4 random skip residual blocks added on the baseline. The three networks are scaled to keep the FLOPs same with ResNet18. Obvious boosting is obtained via exploration in a more complex topology space.

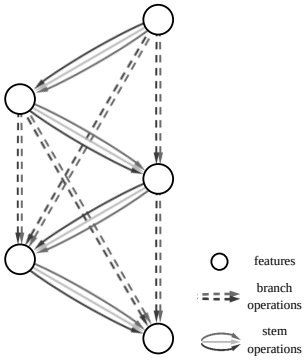


Fig. 3. Illustration of candidates in stem edges and branch edges.

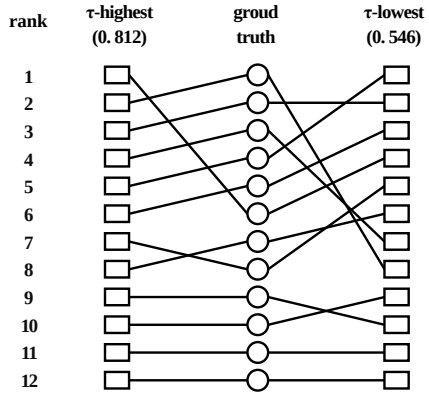


Fig. 4. Detailed ranking of the best and the worst among 20 runs. The ground truth ranking is provided in the middle. This figure shows that the quality of ranking exceedingly differs from s.

$e_i \in E$ and edges represent CNN operators. The nodes $v_1, v_2, ...$ are indexed by the order of computation of their corresponding feature maps.

In our formulation, each e_i is a minimum search unit, also referred as a choice block, which contains a set of candidates computation blocks. A hyper-network is the network which subsumes all the sub-architectures in the search space. Following the previous works, we divide our search space into several sub DAGs (stages), each of which downsamples the input by a factor 2.

To enable the discovery of complex topology architectures, a novel topology augmented search space is proposed. In our search space, edges are divided into two categories, stem edges and branch edges, detailed in Fig. 3.

Stem Edges are non-removable edges which always appear in candidate architectures. The stem edges exist between all node pairs (v_i, v_j), where $|i-j| = 1$. Stem edges are chain-structured, which sequentially connect all consecutive nodes in each stages. We use the 9 kinds of linear bottlenecks (LB) [31] with SE module [16] as the candidate choices of stem edges. Further, on stem edges between feature maps with the same resolution, identity operation is added as an extra candidate to enhance topological diversity and depth flexibility. Therefore, there are 10 choices in the sequential structures.

Branch Edges are optional to contribute to topology diversity in the search space. The branch edges exist between all node pairs (v_i, v_j), where $|i-j| \in \{2, 3\}$. The candidate choices of branch edges are the same to stem edges. Differently, the branch edges could be abandoned flexibly.

When v_i and v_j has different resolution, the stride of convolution operation in the edge is automatically adapted to align the feature maps. The number of nodes in a single stage is required to define the search space. Based on previous method, we set the number of nodes in each stage as 2, 2, 4, 8, 4.

The search space we proposed ensures network topology complexity. Network topology in this work is defined as the DAG formed by nodes and edges. For nodes, the total number of topology is $2^{2(n-3)+1}$. The search space we used in the experiment contains 20 nodes in total, which is around 3.4×10^{10} topologies. For comparison, the topologies contained in cell-based search space is around 7.2×10^9.

3.2 Training the One-shot Hyper-network

One-shot method uses the hyper-network to estimate the performance of architectures. Since huge amount of architectures exists in the hyper-network concurrently, training the hyper-network in whole will make the parameters of different architectures correlated with each other. To reduce the correlation, one-shot method samples a new network architectures m_k at each gradient step and update the only the activated part of the shared parameters.

$$\begin{aligned} \theta_T = \theta_{T-1} + \alpha \cdot \nabla L(\mathcal{F}(X, m_k, \theta_{T-1}), Y), \\ m_k \sim P_t(m_k). \end{aligned} \tag{1}$$

$\mathcal{F}$ makes prediction of input X utilizing sampled model m_k. Thus the gradient of parameters unused by m_k remains zero. The architecture sampling distribution P_t is usually set to trivial uniform sampling [14] across the choice for each single edge.

Suppose there are I_{stem} choices for stem edges and I_{branch} for branch edges other than *none*, a simple uniform sampling strategy in our search space can be described as:

$$p_{stem}(o_i) = \frac{1}{I_{stem}}, \tag{2}$$

$$p_{branch}(o_i) = \frac{1}{I_{branch} + 1}. \tag{3}$$

However, the network sampled under this strategy in our space tends to sample architectures with high computational cost, because each of the large amount of branch edges has a low probability to be *none*. Consequently, the architecture with low computational cost in the hyper-network will under-fit, which would cause a bias in evaluation stage. Thus, the sampling strategy needs further consideration. The whole training process of hyper-network can be finded in Algo. 1. Suppose that C_{target} is our target MAdds and C_{m_k} is the MAdds of architecture m_k, the sampling strategy should meet:

$$E_{m_k \sim P_t(m_k)}[C_{m_k}] = C_{target}. \tag{4}$$

To meet the constraints on expected computation, the sampling probability of *none* choice in branch edges, p_{drop}, is defined to adjust the expected computation cost of sampled networks:

$$p_{branch}(o_i) = \begin{cases} \frac{1-p_{drop}}{I_{branch}}, \ o_i \neq none, \\ p_{drop}, \ o_i = none. \end{cases} \tag{5}$$

Algorithm 1 Hyper-network Training

Inputs: D_{train}, T, B
$G(E, V)$ = InitializeHyperNetwork()
for $t = 1 : T$ **do**
  $p_{stem} = \frac{1}{I_{stem}}$
  $p_{branch} = \frac{1-p_{drop}}{I_{branch}}$ if o_i $\neq$ $none$ $else$ p_{drop}
  $E^{'}_{stem}$ = Sample(E_{stem}, p_{stem})
  $E^{'}_{branch}$ = Sample(E_{branch}, p_{branch})
  $m = G(E^{'}_{stem} \cup E^{'}_{branch}, V)$
  D_{batch} = Sample(D_{train}, $uniform$, B)
  TrainForOneStep(m, D_{batch})
end for
Outputs: $G(E, V)$

3.3 Stabilizing Performance Estimation

In search stage, evaluating an architecture through the shared parameters is essential for exploring promising results. Previous work on one-shot method usually measure the network performance with fully trained hyper-network weights directly. In this section, we first demonstrate our observation on random shuffling of candidates architectures ranking in our search space. Then we introduce our approach to improve the ranking stability.

Instability of One-shot NAS Since the hyper-network is trained T iterations, the shared parameter obtained after training is denoted as θ_T. We define a accuracy function $Acc(m_k, \theta)$ which maps the model architecture m_k and hyper-network weights θ to the validation set accuracy. The value of Acc function can be estimated by simply loading the weight used by m_k and testing the model performance on validation set. The score function, denoted as $S(m_k)$, of previous approach is simply

$$S(m_k) = Acc(m_k, \theta_T). \tag{6}$$

However, the true score function should be the actual performance of the model m_k on validation set: $Acc(m_k, \theta_{m_k})$, where θ_{m_k} denotes the weight obtained by sampling and training m_k only. One-shot approach takes a approximation to reuse the shared parameters for different architectures. Although this is empirically useful, we observe high variance of the model ranking in two cases: rankings at different epochs and rankings by different runs.

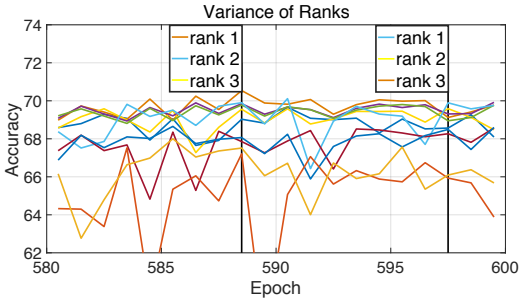


Fig. 5. The ranks of 10 random architectures during the last 20 training epochs, where drastic fluctuation can be observed. Ranks of three architectures at two time steps are shown in the figure, and each of them has different ranks at the two steps.

We randomly sample a set of architecture and obtain their independent weight. We rank their performance under shared parameters on validation set by different checkpoints at the last 20 epochs training of hyper-network. As shown in Fig. 5 , the rank of a single checkpoint fluctuates a lot during hyper-network training process and hardly distinguishes the performance of architectures. If we repeat the hyper-network training with different random seeds for 20 times and obtain shared parameters $\theta_T^i, i \in [1, 20]$. We quantify the correlation between rank of each θ_T^i and ground truth rank by Kendall's τ coefficient [18]. Here, we show the ranking performance of the best and worst runs in Fig. 4.

These two observations imply the necessity of a stabilized evaluation strategy. To present our strategy, formulation of the instability need to be introduced. In this paper, we model the performance estimation randomness as an unbiased noise. Since the shared parameters is fundamentally different from the parameters trained independently, we use a function ϕ to model the affect of weight sharing. General consensus has been reached: empirical $Acc(m_k, \theta_T)$ provides inaccurate but useful ranking, which demonstrates the desired rank preserving effect of ϕ. In summary, our model to describe the quantity relationship is:

$$Acc(m_k, \theta_T) = \phi(Acc(m_k, \theta_{m_k})) + \upsilon, \tag{7}$$

$$E[\upsilon] = 0. \tag{8}$$

It is obvious that the existence of the noise term υ would hurt the model ranking. The most trivial approach to alleviate the negative effects of υ is to train multiple hyper-networks, and eliminate the noise by taking expectation. However, this approach requires several times more computation resources for hyper-network training.

SG-MCMC Sampling The sampling process is described in Algo. 2. In order to obtain high-quality low correlation samples of optimized shared parameters efficiently, we investigate the rich literature of Markov Chain Monte Carlo

Algorithm 2 Shared Parameter Sampling by SGLD

Inputs: D_{train}, T_{SGLD}, T_{epoch}, B, $G(E,V)$, α
$\mathcal{G}_{sample} = \emptyset$
for $t = 1 : T_{SGLD}$ **do**
  $p_{stem} = \frac{1}{I_{stem}}$
  $p_{branch} = \frac{1-p_{drop}}{I_{branch}}$ *if* $o_i \neq none$ *else* p_{drop}
  $E^{'}_{stem} = \text{Sample}(E_{stem}, p_{stem})$
  $E^{'}_{branch} = \text{Sample}(E_{branch}, p_{branch})$
  $m = G(E^{'}_{stem} \cup E^{'}_{branch}, V)$
  $D_{batch} = \text{Sample}(D_{train}, uniform, B)$
  $\theta(m) = \theta(m) + \alpha \nabla L(\mathcal{F}(m, D_{batch})) + \sqrt{2\alpha}\ \epsilon$
  if $t \equiv 0\ mod\ T_{epoch}$ **then**
    $\mathcal{G}_{sample}$.Append$(G(E,V))$
  end if
end for
Outputs: $\mathcal{G}_{sample}$

(MCMC) sampling methods [3]. Recently, a few works demonstrate that constant learning rate stochastic gradient decent could be modified to Stochastic Gradient Langevin Dynamics (SGLD) to realize a Stochastic Gradient MCMC method under mild assumption[6, 39]. Here, we apply SGLD [39, 38] to approximate iid samples of share parameters posterior. The update rule we use, is simply

$$\begin{aligned} \Delta\theta_T = \alpha\nabla(\frac{1}{\mathcal{B}}\sum_{i}^{\mathcal{B}} L(\mathcal{F}(x_i, m_t, \theta_T), y_i)) + \sqrt{2\alpha}\ \epsilon, \\ m_t \sim P_t(m) \text{ and } \epsilon \sim \mathcal{N}(\mathbf{0}, I). \end{aligned} \tag{9}$$

Here $\mathcal{B}$ is the number of data used to compute gradients (batch size). The step size α is set to the final learning rate of sub-net training. To ensure the independence, we generate each sample after SGLD update iterates for a data epoch.

To generate the iid samples of shared weights, we load the weights θ_T of the hyper-network after its training finishes, and set $\theta_T^0 = \theta_T$ as the initial sample. Then, for each θ_T^i, we apply SGLD to obtain the next sample of parameter posterior θ_T^{i+1} with the rule in Eq. (9). Thus we can obtain multiple samples of hyper-network parameters.

Average Accuracy and Parameter Once we have K samples of $\{\theta_T^1, \theta_T^2, ..., \theta_T^K\}$ which approximates the parameters obtained by different run. To eliminate the effect of random noise υ and stabilize the performance estimation, we propose two approaches: score expectation and parameter expectation.

Expectation over scores approach is to define the score $S(m)$ of each model m as the expectation of validation accuracy over K sampled shared weights.

$$S_s(m) = \frac{1}{K}\sum_{i}^{K} Acc(m, \theta_i). \tag{10}$$

Expectation over parameters approach is to take the average of sampled shared parameters and use average parameters to evaluate the performance of each model.

$$S_p(m) = Acc(m, \frac{1}{K}\sum_{i}^{K}\theta_i). \tag{11}$$

Independent fine-tuning When evaluating the single architecture performance, loads the weights from the hyper-network and resuming training the architecture independently should be able to get more architecture-relevant weights. Thus we test this approach in our experiment.

3.4 Evolution Algorithm

Inspired by recent work [26, 36], we apply evolution algorithm NSGA-II as the search agent. In this section, we first introduce some basic concept of NSGA-II. Next we discuss how we apply NSGA-II to our search space.

NSGA-II We seek to obtain the model architecture with excellent performance under the constraint of computational expense. NSGA-II is the most popular choice among multi-objective evolutionary method. The core component of NSGA-II, is the Non Dominated Sorting which benefits the trade off between conflicting objectives. Since our optimization target is to minimize MAdds and maximize performance of architecture under different computational constraints.

Initialization To reduce manual bias and explore the search space better, we use random initialization for all individuals of the first generation. More specifically, each architecture randomly select basic operators for each block in the search space.

Crossover and Mutation Single-point crossover on random position is adopted in our evolution algorithm. For two certain individuals $m_1 = (x_1, x_2, ..., x_n)$ and $m_2 = (y_1, y_2, ..., y_n)$, a single-point crossover strategy on position p will result in a new individual $m_3 = (x_1, x_2, ..., x_p, y_{p+1}, y_{p+2}, ..., y_n)$.

We use random choice mutation to enhance generation diversity. When a mutation happens to an individual, a selected operation block in it is changed to another available choice randomly.

4 Experiments and Results

We verify the effectiveness of our method on a large classification benchmark, ImageNet [30]. In this section, we firstly describe our implementation details. Secondly, we present the performance of searching results on ImageNet as well as comparison with state of the art methods., Finally, we demonstrate the advantage of our designs via ablation study.

4.1 Experiments Settings

Datasets We conduct experiments on the ImageNet, a standard benchmark for classification task. It has $1.28M$ training images and $50K$ validation images.
Train Details of Hyper-network For the training of hyper-network, we adopt cosine learning rate scheduler with learning rate initialized as 0.1 and decaying to 2.5e-4 during 600 epochs. A L2 regularization is used and its weight is set to 1e-4. The optimizer is mini-batch stochastic gradient decent (SGD) with batch size 512 and we set momentum as 0 to decouple the gradients of architectures sampled in different batches. Hyper-parameter p_{drop} is set to 0.6. The hyper-network is trained on 32 GTX-1080Ti GPUs. We implement the stabilized evaluation of our method by saving 20 checkpoints at $600, 601, ..., 619$ epochs described in Sec. 3.3. The fine-tune strategy mentioned above is conducted with learning rate 2.5e-4.
Search Details The evolution agent randomly generates 45 individuals for initialization. Then it repeats the exploitation and exploration loop where it generates 45 individuals via single-point crossover and random mutation. It conducts 22 loops and evaluates 990 models. At last, we choose the top ranked 2 models under different MAdds constraints.
Training Details of Resulted Architecture For the independent training of resulted architectures, we use cosine learning rate scheduler with initial learning rate 0.8. We train the model for 300 epochs with batch size 2048 and adopt SGD optimizer with nesterov and momentum value 0.9. To prevent overfitting, we use L2 regularization with weight 1e-4 and standard augmentations including random crop and colorjitter.

4.2 Main Results

ST-NAS looks for models with objectives of low MAdds and high accuracy. We adopt expectation over parameters to select two resulted models separately under small and large MAdds constraints, namely, ST-NAS-A and ST-NAS-B.

Model	Search space	Params (M)	MAdds (M)	Top-1 acc (%)	Top-5 acc (%)
DARTS [25]	Cell-based	4.7	574	73.3	91.3
Proxyless-R [5]	Chain-structured	-	320	74.6	92.2
Single-path NAS [32]	Chain-structured	4.3	365	75.0	92.2
FairNAS-A [8]	Chain-structured	4.6	388	75.3	92.4
FBNet-C [40]	Chain-structured	5.5	375	74.9	-
SPOS [14]	Chain-structured	-	328	74.7	-
BetaNet-A [12]	Chain-structured	7.2	333	75.9	92.8
ST-NAS-A(ours)	**Topology augmented**	**5.2**	**326**	**76.4**	**93.1**

Table 2. Performance comparison between ST-NAS and efficient NAS methods on ImageNet. Our model ST-NAS-A archieve best top 1 accuracy with the least MAdds.

Model	Search space	Params (M)	MAdds (M)	Top-1 acc (%)	Top-5 acc (%)
*MNASNet-A1 [36]	Chain-structured	3.9	312	75.2	92.5
*MNASNet-A2 [36]	Chain-structured	4.8	340	75.6	92.7
*RCNet-B [42]	Chain-structured	4.7	471	74.7	92.0
*NASNet-B [46]	Cell-based	5.3	488	72.8	91.3
*EfficientNet-B0 [37])	Chain-structured	5.3	390	76.3	93.2
ST-NAS-A (ours)	**Topology augmented**	**5.2**	**326**	**76.4**	**93.1**
1.4-MobileNetV2 [31]	Chain-structured	6.9	585	74.7	92.5
2.0-ShuffleNetV2 [27]	Chain-structured	7.4	591	74.9	-
*NASNet-C [46]	Cell-based	4.9	558	72.5	91.0
*NASNet-A [46]	Cell-based	5.3	564	74.0	91.6
*1.4-MNASNet-A1 [36]	Chain-structured	-	600	77.2	93.7
*RENASNet [7]	Cell-based	5.4	580	75.7	92.6
*PNASNet [24]	Cell-based	5.1	588	74.2	91.9
ST-NAS-B (ours)	**Topology augmented**	**7.8**	**503**	**77.9**	**93.8**

Table 3. Performance comparison among ST-NAS, manual designed networks and sample based NAS methods on ImageNet. Notably sample based methods with mark $*$ takes much more computation resources. We show that the architecture discovered by ST-NAS perform better than both sample based NAS and manually designed architectures while maintaining least MAdds.

Architectures and performance of them compared with state-of-the-art methods are discussed in this subsection.

Performance on ImageNet. We compare ST-NAS method with efficient NAS methods, including DARTS, ProxyLessNAS and FBNet, in Table 2. Our model ST-NAS-A outperforms all of them while with the least MAdds and comparable parameter number.

For architectures resulted from high cost, *i.e*, manually designed networks and networks obtained by sample-based methods, we compare ST-NAS with them in two groups divided by MAdds, as shown in Table 3. At a much less search cost, our ST-NAS outperform all the methods in both MAdds groups.

Performance on COCO. Our implementation is based on feature pyramid network (FPN)[23]. Different models pretrained on ImageNet is utilized as feature extractor. All the models are trained for 13 epochs, known as $1\times$ schedule. The results are shown in Table 4. Our ST-NAS-A backbone outperforms MobileNetV2. The ST-NAS-B performs comparably with ResNet50 with much less MAdds.

4.3 Ablation Studies

Improvement on topology We adopt our search method on chain structured space. The result on ImageNet is 77.0% with 500M MAdds. Compared

Model	MAdds (backbone) (G)	mAP
MobileNetV2	0.33	31.7
ST-NAS-A	0.33	33.2
ResNet18	1.81	32.2
ST-NAS-B	0.50	35.3
ResNet50	4.09	36.9
ST-NAS-B*	1.03	37.7

Table 4. Performance on COCO dataset. The channel number of ST-NAS-B is scaled to get ST-NAS-B*. ST-NAS-A outperforms MobileNetV2 by 1.5% COCO AP while maintaining same MAdds. ST-NAS-B* achieves 0.5% higher than ResNet50 but needs only a quarter MAdds.

with ST-NAS-B, searching for a topology obtain 0.9% improvement with similar computation cost.

Rank Fluctuation To explain the importance of our stabilization mechanism, we randomly sample a set of architecture and rank their performance under shared parameters on validation set at the last 10 epochs training of hyper-network. As shown in Fig. 5, the rank of a single checkpoint fluctuates a lot during hyper-network training process and hardly distinguishes the performance of architectures, implying the necessity of a stabilized evaluation strategy.

Ranking Verification We further verify this reduction by quantifying the ranking ability of different evaluation strategies by correlation coefficient between ranks in hyper-network and the ground truth ranks. Kendall's tau coefficient is adopted as the metric in our verification. We randomly sample 12 networks and train them form scratch to obtain the ground truth rank. To compare with single checkpoint, we make use of checkpoints of 10 epochs at the

Estimation approach	τ
Single checkpoint	0.71
Fine-tune	0.64
SGLD-param	0.84
SGLD-acc	0.81

Table 5. τ of different rank stabilization approach we proposed. The original baseline is single checkpoint achieves 0.71 of Kendall τ which is 0.07 higher than the fine-tune approach. The parameter expectation and accuracy expectation method is tied and outperform the baseline with a margin.

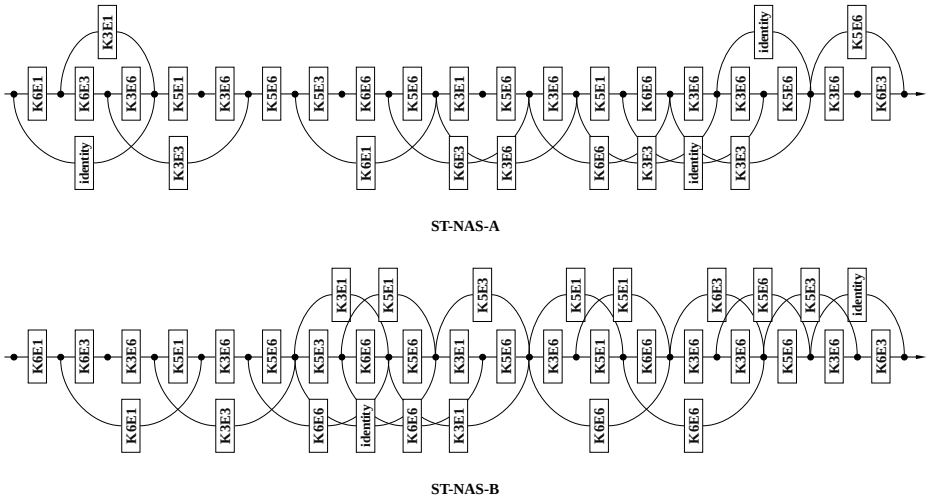


Fig. 6. Resulted architectures. Linear bottle-neck contains a $K \times K$ group-wise convolution layer between two 1×1 point-wise convolution layers. Expand ratio is defined as the ratio between group-wise convolution channels and point-wise convolution channels. We describe a linear bottle-neck with its expand ratio, *i.e*, the number after "E", and its group-wise convolution kernel size, *i.e*, the number after "K".

591-th, 592-th, ..., 600-th epoch to generate 10 ranks and get 10 correlation coefficients with the ground truth rank. The median of the 10 correlation coefficients is adopted to compare with other strategies. It is observed in Table 5 that SGLD consistently achieves higher correlation coefficients than fine-tune and single checkpoint, which verifies the effectiveness of SGLD in the reduction of parameter variance.

5 Conclusion

We proposed a topology-diverse search space and a novel search method, ST-NAS. In ST-NAS, we improve both the sampling strategy during hyper-network training and the architecture evaluation approach by rigorous theoretical analysis. Sound experiments demonstrate the effectiveness of our designs and achieve consistent improvements under different computation cost constraints.

Acknowledgement This work was supported by the National Key Research and Development Project of China (No. 2018AAA0101900).

References

1. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167 (2016)

2. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning. pp. 549–558 (2018)
3. Bishop, C.M.: Pattern recognition and machine learning. springer (2006)
4. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Smash: one-shot model architecture search through hypernetworks. arXiv preprint arXiv:1708.05344 (2017)
5. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. arXiv preprint arXiv:1812.00332 (2018)
6. Chen, C., Carlson, D., Gan, Z., Li, C., Carin, L.: Bridging the gap between stochastic gradient mcmc and stochastic optimization. In: Artificial Intelligence and Statistics. pp. 1051–1060 (2016)
7. Chen, Y., Meng, G., Zhang, Q., Xiang, S., Huang, C., Mu, L., Wang, X.: Reinforced evolutionary neural architecture search. arXiv preprint arXiv:1808.00193 (2018)
8. Chu, X., Zhang, B., Xu, R., Li, J.: Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. arXiv preprint arXiv:1907.01845 (2019)
9. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation policies from data. arXiv preprint arXiv:1805.09501 (2018)
10. Du, X., Lin, T.Y., Jin, P., Ghiasi, G., Tan, M., Cui, Y., Le, Q.V., Song, X.: Spinenet: Learning scale-permuted backbone for recognition and localization. arXiv preprint arXiv:1912.05027 (2019)
11. Elsken, T., Metzen, J.H., Hutter, F.: Simple and efficient architecture search for convolutional neural networks. arXiv preprint arXiv:1711.04528 (2017)
12. Fang, M., Wang, Q., Zhong, Z.: Betanas: Balanced training and selective drop for neural architecture search. arXiv preprint arXiv:1912.11191 (2019)
13. Guo, M., Zhong, Z., Wu, W., Lin, D., Yan, J.: Irlas: Inverse reinforcement learning for architecture search. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9021–9029 (2019)
14. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. arXiv preprint arXiv:1904.00420 (2019)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
16. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018)
17. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
18. Kendall, M.G.: A new measure of rank correlation. Biometrika **30**(1/2), 81–93 (1938)
19. Li, C., Yuan, X., Lin, C., Guo, M., Wu, W., Yan, J., Ouyang, W.: Am-lfs: Automl for loss function search. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 8410–8419 (2019)
20. Li, X., Lin, C., Li, C., Sun, M., Wu, W., Yan, J., Ouyang, W.: Improving one-shot nas by suppressing the posterior fading. arXiv preprint arXiv:1910.02543 (2019)
21. Liang, F., Lin, C., Guo, R., Sun, M., Wu, W., Yan, J., Ouyang, W.: Computation reallocation for object detection. arXiv preprint arXiv:1912.11234 (2019)
22. Lin, C., Guo, M., Li, C., Yuan, X., Wu, W., Yan, J., Lin, D., Ouyang, W.: Online hyper-parameter learning for auto-augmentation strategy. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 6579–6588 (2019)

23. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2117–2125 (2017)
24. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018)
25. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
26. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: Nsga-net: a multi-objective genetic algorithm for neural architecture search. arXiv preprint arXiv:1810.03522 (2018)
27. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 116–131 (2018)
28. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 4780–4789 (2019)
29. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2902–2911. JMLR. org (2017)
30. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of computer vision **115**(3), 211–252 (2015)
31. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018)
32. Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., Marculescu, D.: Single-path nas: Designing hardware-efficient convnets in less than 4 hours. arXiv preprint arXiv:1904.02877 (2019)
33. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
34. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
35. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
36. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)
37. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946 (2019)
38. Teh, Y.W., Thiery, A.H., Vollmer, S.J.: Consistency and fluctuations for stochastic gradient langevin dynamics. The Journal of Machine Learning Research **17**(1), 193–225 (2016)
39. Welling, M., Teh, Y.W.: Bayesian learning via stochastic gradient langevin dynamics. In: Proceedings of the 28th international conference on machine learning (ICML-11). pp. 681–688 (2011)

40. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10734–10742 (2019)
41. Xie, S., Kirillov, A., Girshick, R., He, K.: Exploring randomly wired neural networks for image recognition. arXiv preprint arXiv:1904.01569 (2019)
42. Xiong, Y., Mehta, R., Singh, V.: Resource constrained neural network architecture search: Will a submodularity assumption help? In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1901–1910 (2019)
43. Zhang, Y., Lin, Z., Jiang, J., Zhang, Q., Wang, Y., Xue, H., Zhang, C., Yang, Y.: Deeper insights into weight sharing in neural architecture search. arXiv preprint arXiv:2001.01431 (2020)
44. Zhong, Z., Yang, Z., Deng, B., Yan, J., Wu, W., Shao, J., Liu, C.L.: Blockqnn: Efficient block-wise neural network architecture generation. arXiv preprint arXiv:1808.05584 (2018)
45. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
46. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)