Breaking the Curse of Space Explosion: Towards Efficient NAS with Curriculum Search

Yong Guo [* 1] **Yaofo Chen** [* 1 2] **Yin Zheng** [* 3] **Peilin Zhao** [* 4] **Jian Chen** [1 5] **Junzhou Huang** [4] **Mingkui Tan** [1]

Abstract

Neural architecture search (NAS) has become an important approach to automatically find effective architectures. To cover all possible good architectures, we need to search in an extremely large search space with billions of candidate architectures. More critically, given a large search space, we may face a very challenging issue of space explosion. However, due to the limitation of computational resources, we can only sample a very small proportion of the architectures, which provides insufficient information for the training. As a result, existing methods may often produce sub-optimal architectures. To alleviate this issue, we propose a curriculum search method that starts from a small search space and gradually incorporates the learned knowledge to guide the search in a large space. With the proposed search strategy, our Curriculum Neural Architecture Search (CNAS) method significantly improves the search efficiency and finds better architectures than existing NAS methods. Extensive experiments on CIFAR-10 and ImageNet demonstrate the effectiveness of the proposed method.

1. Introduction

Deep neural networks (DNNs) have been producing state-of-the-art results in many challenging tasks including image classification (Krizhevsky et al., 2012; Liu et al., 2020; Zheng et al., 2015; Guo et al., 2020a), semantic segmentation (Shelhamer et al., 2017; Chen et al., 2018; Huang et al., 2019), and many other areas (Zheng et al., 2016b;a; Jiang

---

[*]Equal contribution . This work is done when Yong Guo works as an intern in Tencent AI Lab. [1]School of Software Engineering, South China University of Technology [2]Pazhou Laboratory [3]Weixin Group, Tencent [4]Tencent AI Lab, Tencent [5]Guangdong Key Laboratory of Big Data Analysis and Processing. Correspondence to: Mingkui Tan <mingkuitan@scut.edu.cn>, Jian Chen <ellachen@scut.edu.cn>.
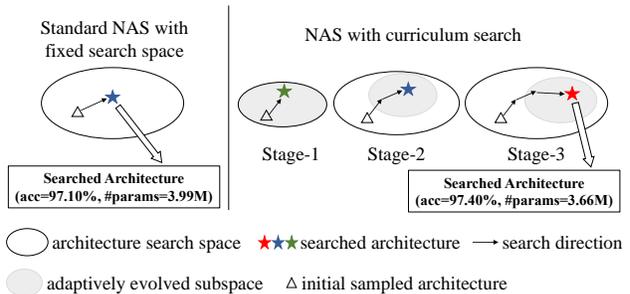
Figure 1. Comparisons of the search process between standard NAS methods and our proposed curriculum NAS (CNAS) method. By gradually refining the candidate subspace, CNAS performs much accurate architecture sampling, thus can significantly improve the search efficiency and find better architectures than standard NAS methods.

et al., 2017; Chen et al., 2019; Zeng et al., 2019; Hosseini et al., 2020; Guo et al., 2020b; Cao et al., 2018). Besides designing DNNs manually, there is a growing interest in automatically designing effective architectures by neural architecture search (NAS) methods (Zoph & Le, 2017; Pham et al., 2018). Existing studies show that the automatically searched architectures often outperform the manually designed ones in both image classification tasks and language modeling tasks (Zoph et al., 2018; Lian et al., 2020).

NAS methods seek to search for an optimal architecture in a predefined search space. To this end, existing methods have to explore the whole search space by sampling sufficiently many architectures. However, the search space is often extremely large (*e.g.*, billions of candidate architectures (Pham et al., 2018)), resulting in the space explosion issue. To be specific, due to the limitation of computational resources, we can only sample a very small proportion of the architectures from the search space. Thus, we can only receive very limited information to guide the search process. As a result, NAS models become hard to train and often find sub-optimal architectures.

To alleviate the space explosion issue, we seek to improve the search by conducting more accurate sampling, *i.e.*, exploring the subspace that contains potentially better architectures. In this way, given limited resources to conduct

sampling, we can find good architectures with relatively high probability and thus improve the search performance. However, how to train NAS models to perform more accurate sampling is still unknown.

To address the above issues, we propose a novel curriculum search method to improve the search performance. Specifically, we first conduct the search in a relatively small space where sufficient exploration is possible. Then, we gradually enlarge the search space and use the previously learned knowledge to make the sampling more accurate in the larger search space. The key idea follows the basic concept of curriculum learning that humans and animals can learn much better when they gradually learn new knowledge (Bengio et al., 2009; Gülçehre & Bengio, 2016).

Equipped with the curriculum search, we propose a Curriculum Neural Architecture Search (CNAS) method. Our CNAS enlarges the search space by gradually increasing the number of candidate operations and exploits the previously learned knowledge to achieve accurate sampling. As shown in Figure 1, once we have found some good architectures in a small search space and gradually enlarge the space, it becomes more likely to find a candidate subspace (grey circle) that shares some common knowledge with the previous one but contains potentially better architectures. When we consider multiple stages to perform architecture search, the subspace would adaptively evolve along the search process. Based on the adaptively evolved subspace, CNAS is able to conduct more accurate sampling to find better architectures in a large search space. Furthermore, to improve the training stability when we introduce a previously unseen operation, we propose an operation warmup strategy to make all the operations relatively fair when sampling architectures. Extensive experiments demonstrate the superiority of our CNAS over existing methods.

Our contributions are summarized as follows:

- We propose a novel Curriculum Neural Architecture Search (CNAS) method to alleviate the training difficulties of the NAS problem incurred by the extremely large search space. To this end, we break the original NAS problem into a series of simpler problems and train the controller model in a progressive manner.

- We propose a curriculum search method that gradually incorporates the knowledge learned from a small search space. To this end, we start from a search space with one operation and gradually add new operations. Thus, the previously learned knowledge about how to use a specific operation can be effectively preserved.

- Extensive experiments on several benchmark data sets show that the architectures found by our CNAS significantly outperform the architectures obtained by state-of-the-art NAS methods.

2. Related Work

Neural architecture search. In the past few years, neural architecture search (NAS) has attracted considerable attention to automatically design effective architectures. Zoph & Le (2017) first propose to learn a controller for an optimal configuration of each convolution layer. However, it performs search for an entire network, leading to extremely large search space and very high search cost. To reduce the search space, NASNet (Zoph et al., 2018; Pham et al., 2018) proposes to search for the optimal neural cell rather than the whole network. Related to our method, Liu et al. (2018a) propose a PNAS method that gradually enlarges the search space and performs architecture search in a progressive manner. Specifically, PNAS picks the top K architectures in each stage and gradually adds nodes to progressively enlarge the search space. However, there are several limitations with PNAS. First, the unselected architectures as well as all the architectures that are obtained by adding additional nodes would be ignored. As a result, it greatly limit the possible search space to find good architectures in the next stage. Second, PNAS has to train a large number of architectures until convergence to learn a performance predictor, resulting in extremely high search cost (*i.e.*, 255 GPU days).

Curriculum learning. Bengio et al. (2009) propose a new learning strategy called curriculum learning, which improves the generalization ability of model and accelerates the convergence of the training process. Recently, many efforts have been made to design effective curriculum learning methods (Wang et al., 2019; Zhang et al., 2019). Kumar et al. (2010) propose a self-paced learning algorithm that selects training samples in a meaningful order. Khan et al. (2011) provide evidence about the consistency between curriculum learning and the learning principles of human. Bengio et al. (2013) further present insightful explorations for the rationality of curriculum learning. Matiisen et al. (2019) build a framework that produces Teacher-Student curriculum learning. Moreover, curriculum learning has been applied to NAS (Cheng et al., 2018; Zoph & Le, 2017). Cheng et al. (2018) propose an InstaNAS method that uses a dynamic reward function to gradually increase task difficulty, and Zoph & Le (2017) propose to gradually increase the number of layers. Unlike these methods, we construct simpler problems from a new perspective of search space in the context of the NAS problem.

3. Preliminary

For convenience, we revisit the Neural Architecture Search (NAS) problem. Reinforcement Learning (RL) based NAS methods seek to learn a controller to produce candidate architectures. Let θ be the trainable parameters and Ω be the search space. A controller can produce a candidate

architecture α as follows:

$$\alpha \sim \pi(\alpha; \theta, \Omega), \tag{1}$$

where π is a policy learned by the controller, *i.e.*, a distribution of candidate architectures. In practice, the policy learned by the controller can be either the discrete form (*e.g.*, NAS (Zoph & Le, 2017) and ENAS (Pham et al., 2018)), or the differentiable form like DARTS (Liu et al., 2019).

To measure the performance of an architecture, we have to train a model built with the architecture until convergence. Let $\mathcal{L}$ be the loss function on training data. Given an architecture α, the optimal parameters $w^*(\alpha)$ can be obtained by $w^*(\alpha) = \arg\min_w \mathcal{L}(\alpha, w)$. Then, one can measure the performance of α by some metric $R(\alpha, w^*(\alpha))$, *e.g.*, the accuracy on validation data. Following the settings of NAS (Zoph & Le, 2017) and ENAS (Pham et al., 2018), we use Reinforcement Learning (RL) and take the performance $R(\cdot)$ as the "reward" to train the controller model.

The goal of RL-based NAS is to find an optimal policy by maximizing the expectation of the reward $R(\alpha, w^*(\alpha))$. Thus, the NAS problem can be formulated as a bilevel optimization problem:

$$\begin{aligned} \max_\theta \ &\mathbb{E}_{\alpha \sim \pi(\alpha;\theta,\Omega)} \mathcal{R}(\alpha, w^*(\alpha)), \\ \text{s.t. } &w^*(\alpha) = \arg\min_w \mathcal{L}(\alpha, w). \end{aligned} \tag{2}$$

To solve this problem, one can update θ and w in an alternative manner (Zoph & Le, 2017; Pham et al., 2018).

4. Curriculum Neural Architecture Search

For NAS methods, the huge search space is often the main bottleneck to the architecture search performance. In general, a small space often leads to sub-optimal architectures with inferior performance, but a large search space would incur severe space explosion issue and may even make the learning task infeasible. Specifically, since we can only explore a very small proportion of architectures to train the controller, it is very difficult to learn a NAS model to find good architectures in a large search space.

In this paper, we seek to improve NAS by alleviating the space explosion issue. We first analyze the size of the search space. Then, we propose a curriculum search strategy to break the original NAS problem into a series of simpler problems and then we solve each problem in a progressive manner. For convenience, we call our method Curriculum Neural Architecture Search (CNAS). An illustrative comparisons between the standard NAS methods and the proposed CNAS can be found in Figure 1.
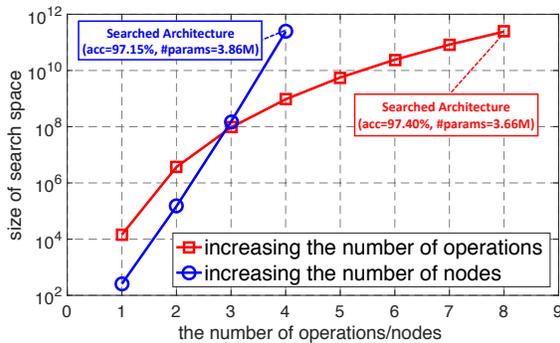
Figure 2. Comparisons of different search spaces on one cell architecture. Following (Liu et al., 2019), we consider a cell architecture with 4 intermediate nodes and 8 candidate operations.

4.1. Search Space Size Analysis

We consider learning a generic computational cell because searching a good architecture for an entire network is very computationally expensive (Zoph et al., 2018). In NAS methods (Zoph & Le, 2017; Pham et al., 2018; Liu et al., 2019), a cell-based architecture α can be represented by a directed acyclic graph (DAG), *i.e.*, $\alpha = (\mathcal{V}, \mathcal{E})$. Here, $\mathcal{V}$ is the set of nodes that represent the feature maps in a neural network. $\mathcal{E}$ is the set of the edges that represent some computational operations (*e.g.*, convolution or pooling). For convenience, we denote the numbers of nodes by B.

Following (Liu et al., 2019), a DAG contains two input nodes, $B-3$ intermediate nodes, and one output node. The input nodes denote the outputs of the nearest two cells in front of the current one. The output node concatenates the outputs of all the intermediates to produce a final output of the cell. In the DAG, each intermediate node also takes two previous nodes in this cell as inputs. In this sense, there are $2 \times (B-3)$ edges in the DAG and we will determine which operation should be applied to each of them.

Given B nodes and K candidate operations, the size of the search space Ω can be computed by[1]

$$|\Omega| = K^{2(B-3)}\big((B-2)!\big)^2. \tag{3}$$

From Eqn. (3), the search space can be extremely large when we have a large B or K. For example, ENAS (Pham et al., 2018) has a search space of $|\Omega| \approx 5 \times 10^{12}$ with $B=8$ and $K=5$, and DARTS (Liu et al., 2019) has a search space of $|\Omega| \approx 2 \times 10^{11}$ with $B=7$ and $K=8$. In the extremely large search space, we can only sample a very limited number of architectures. As a result, the feedback/reward from the sampled architectures is insufficient, making it hard to train a good controller. As a result, the search process may severely suffer from the space explosion issue.

---

[1] More analysis of search space size is put in supplementary.
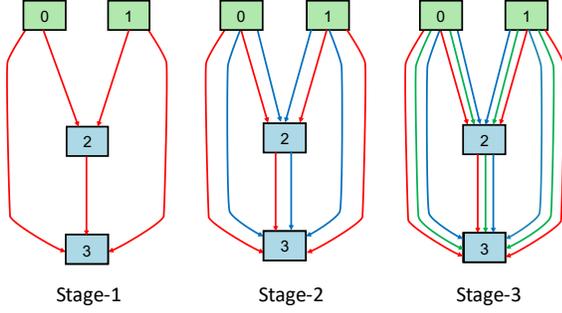
Figure 3. An overview of the search space used by CNAS. We show the candidate operations of the super network in different stages. The edges with different colors denote different operations. For simplicity, we omit the output node in this figure.

4.2. NAS with Curriculum Search

To alleviate the space explosion issue, we seek to improve the search process by providing more powerful information to improve the quality of architecture sampling. To this end, we exploit the idea of curriculum learning that humans often learn much better when they gradually learn new knowledge/concepts. Specifically, we propose to break the NAS problem into a series of simpler problems. Since the size of the search space is an indicator of the difficulty level of the NAS problem, we may change the size of search space to construct the problems with different difficulty levels. In this sense, we can cast the training of NAS models into a multi-stage process to gradually incorporate previously learned knowledge to find better architectures.

As mentioned in Section 4.1, the size of search space $|\Omega|$ depends on the number of nodes B and the number of candidate operations K. In this sense, we can adjust either B or K to obtain the search spaces with different sizes. To find a better choice between these two methods, we compare the search space w.r.t. different B and K. Following (Pham et al., 2018; Liu et al., 2019), we adopt a widely used setting with $B=7$ nodes (*i.e.*, 4 intermediate nodes) and $K=8$ candidate operations. In this case, we investigate the effect of increasing the number of nodes and operations on the size of the search space in Figure 2. From Figure 2 and Eqn. (3), increasing node would make the size of search space grow much faster than increasing operation. As a result, increasing node would introduce a non-negligible gap between adjacent stages. Thus, the training difficulty incurred by the extremely increased search space is still severe. On the contrary, increasing operation from 1 to K provides a more slow growth of search space, making progressive training possible (See the detailed comparisons in Section 6.1).

Thus, we seek to enlarge the search space by gradually increasing the number of candidate operations. Specifically, we start from the search space with a single operation and

---

Algorithm 1 Training method for CNAS.

Require: The operation sequence O, learning rate η, the number of the iterations for operation warmup M, the uniform distribution of architectures $p(\cdot)$, the controller's policy $\pi(\cdot)$, super network parameters w, controller parameters θ.
1: Initialize w and θ, $\Omega_0 = \varnothing$.
2: **for** $i=1$ to $|O|$ **do**
3:     Enlarge Ω_i by adding O_i to the set of candidate operations;
4:     // *Operation warmup*
5:     **for** $j=1$ to M **do**
6:         Sample $\alpha \sim p(\alpha; \Omega_i)$;
7:         $w \leftarrow w - \eta \nabla_w \mathcal{L}(\alpha, w)$;
8:     **end for**
9:     **while** not convergent **do**
10:         // *Update θ by maximizing the reward*
11:         **for** each iteration on validation data **do**
12:             Sample $\alpha \sim \pi(\alpha; \theta, \Omega_i)$;
13:             Update the controller by ascending its gradient:
14:             $\mathcal{R}(\alpha, w)\nabla_\theta \log \pi(\alpha; \theta, \Omega_i) + \lambda H(\pi(\cdot; \theta, \Omega_i))$;
15:         **end for**
16:         // *Update w by minimizing the training loss*
17:         **for** each iteration on training data **do**
18:             Sample $\alpha \sim \pi(\alpha; \theta, \Omega_i)$;
19:             $w \leftarrow w - \eta \nabla_w \mathcal{L}(\alpha, w)$.
20:         **end for**
21:     **end while**
22: **end for**

---

then add a new operation to the set of candidate operations in each stage. To accelerate the search process, we adopt the parameter sharing (Pham et al., 2018) technique that makes all the child networks share their parameters in a super network. For clarity, we show the super network with the progressively growing search space in Figure 3. Without loss of generality, we add the operations in a random order (See discussions about the order in Section 7.2).

Curriculum training scheme. Based on the curriculum search strategy, we can obtain a series of problems with different difficulty levels. However, how to effectively solve these problems to improve the training of the controller model still remains a question. To address this issue, we propose a curriculum training algorithm and show the details in Algorithm 1. Specifically, we progressively train the controller to solve the problems with different search spaces. During the training, we gradually increase the number of operations from 1 to K. Thus, the whole training process can be divided into K stages. To encourage the diversity when sampling architectures, we introduce an entropy term into the objective. Let Ω_i be the search space of the i-th stage. The training objective in the i-th stage becomes

$$\max_{\theta} \ \mathbb{E}_{\alpha \sim \pi(\cdot; \theta, \Omega_i)} \left[\mathcal{R}\left(\alpha, w^*(\alpha)\right) \right] + \lambda H\left(\pi\left(\cdot; \theta, \Omega_i\right)\right),$$

$$\text{s.t. } w^*(\alpha) = \arg\min_{w} \mathcal{L}\left(\alpha, w\right),$$

$$\tag{4}$$

where $\pi(\cdot; \theta, \Omega_i)$ denotes the learned policy *w.r.t.* Ω_i, $H(\cdot)$ evaluates the entropy of the policy, and λ controls the

strength of the entropy regularization term. Note that $\pi(\alpha; \theta, \Omega_i)$ denotes the probability to sample some architecture α from the policy/distribution $\pi(\cdot; \theta, \Omega_i)$. This entropy term enables CNAS to explore the unseen areas of previous search stages and thus escape from local optima.

Inferring architectures. Once we obtain a good controller model, we can use it to infer good architectures. Given K candidate operations, we take the learned policy $\pi(\cdot; \theta, \Omega_K)$ obtained in the final stage, *i.e.*, the one with the largest search space, as the final policy to sample architectures. Following (Zoph & Le, 2017; Pham et al., 2018), we first sample 10 architectures and then select the architecture with the highest validation accuracy.

4.3. Operation Warmup

In CNAS, we gradually add new operations into the set of candidate operations. However, the corresponding parameters of the new operation are randomly initialized while the old operations have been extensively trained, leading to severe unfairness issues among operations (Chu et al., 2019). As a result, the architectures with the new operation often yield very low rewards, making the controller tend not to choose it in the following training process.

To address this issue, we propose an effective operation warmup method. Specifically, when we add a new operation, we fix the controller model and only train the parameters of the super network. To improve the fairness of operations, we uniformly sample candidate architectures to train each operation with equal probability (Chu et al., 2019). In this way, the candidate architectures with the newly added operation achieve comparable performance with the existing ones. With the operation warmup method, we make the search process more stable and obtain significantly better search performance (See results in Section 7.1).

5. More Discussions on CNAS

In this section, we conduct further analysis of the proposed CNAS method. We first investigate our advantages over the existing NAS methods. Then, we discuss the differences between CNAS and a related work PNAS.

5.1. Advantages of CNAS over the standard NAS

The major advantage lies in the proposed curriculum search strategy. Specifically, CNAS trains the controller in a small search space in the early stage. Compared with the large search space, we can easily obtain a good controller since we can sufficiently explore the small search space (*e.g.*, $|\Omega| = 120$ when $K = 1$). In this case, we do not need to consider which operation should be chosen but learn an optimal cell topology (*i.e.*, node connection method). When we gradually increase K, CNAS only needs to learn the

new concept (*i.e.*, the new operation) to fit the larger search space. More critically, we can take the previously learned knowledge about which cell topology is good and explore the subspace that shares similar topology in the larger space. As a result, it is more likely to find better architectures compared with the standard NAS method searched in a fixed search space (See results in Section 6.1).

5.2. Differences from PNAS

A related work PNAS (Liu et al., 2018a) also conducts architecture search in a progressive manner. However, there exist several major differences between our method and PNAS. First, PNAS gradually increases the number of nodes to conduct a progressive search. However, we analyze the size of the search space and propose to gradually enlarge the search space by introducing additional operations. Second, PNAS exploits a heuristic search method that periodically removes a large number of possible architectures from the search space and thus limits the exploration ability. However, CNAS performs architecture search in the original search space specified by each stage, making it possible to find potentially better architectures. Third, PNAS has to train a large number of architectures until convergence to learn a performance predictor, resulting in extremely high search cost (*e.g.*, 255 GPU days). However, CNAS exploits the weight sharing technique (Pham et al., 2018) and yields significantly lower search cost (See Table 1).

6. Experiments

We apply the proposed CNAS to train the controller model on CIFAR-10 (Krizhevsky & Hinton, 2009). Then, we evaluate the searched architectures on CIFAR-10 and ImageNet (Deng et al., 2009). All the implementations are based on PyTorch.[2] We organize the experiments as follows. First, to demonstrate the effectiveness of our proposed CNAS, we compare the performance of the proposed CNAS with two related variants. Second, we compare the performance of the architectures searched by CNAS with state-of-the-art image classification methods on CIFAR-10 and ImageNet.

Compared methods. To investigate the effect of the proposed curriculum search strategy, we investigate and compare the following methods: 1) **Fixed-NAS**: For each stage of CNAS, we keep the search space fixed and train a controller from scratch. Following the settings in (Liu et al., 2019), we set the number of the nodes B to 7 (*i.e.*, 4 intermediate nodes) and the number of candidate operations K to 8. 2) **CNAS**: We train the controller in a growing search space by gradually adding new operations while keeping B unchanged. 3) **CNAS-Node**: By fixing K, we gradually increase B from 1 to 4. We also compare the proposed CNAS

---

[2]The code is available at https://github.com/guoyongcs/CNAS.
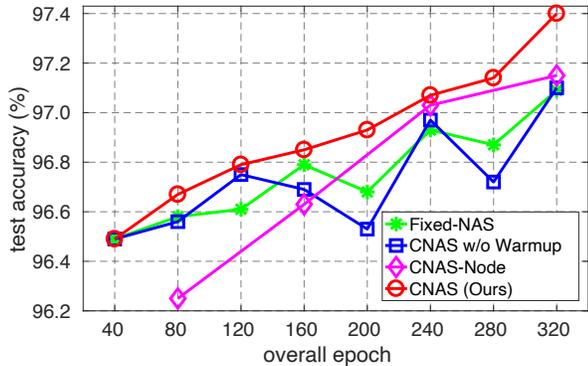
Figure 4. Performance comparisons of the architectures obtained by different methods during the search process. All the models are evaluated on the test set of CIFAR-10. Each point indicates the average performance of the architectures searched over 5 independent experiments in different stages.

with several state-of-the-art image classification methods, such as NASNet (Zoph et al., 2018), AmoebaNet (Real et al., 2019), PNAS (Liu et al., 2018a), ENAS (Pham et al., 2018), DARTS (Liu et al., 2019), *etc*.

6.1. Demonstration of CNAS

To investigate our CNAS, we compare the performance of the architectures searched in each stage with Fixed-NAS and CNAS-Node. For a fair comparison, we train the controller on CIFAR-10 using these three methods for the same epochs, *i.e.*, 320 epochs in total. We use the same operation order for both CNAS and Fixed-NAS (*i.e.*, CNAS and Fixed-NAS have the same search space in each stage). We sample architectures at the end of each stage and train them to convergence. All the architectures are limited to 3.8M parameters in the evaluation. We show the comparisons of different methods in different stages in Figure 4.

From Figure 4, both CNAS and Fixed-NAS architectures obtain better performance as the search space increases. However, our CNAS architectures consistently outperform Fixed-NAS ones for all stages. This implies that directly searching in a large search space (*i.e.*, Fixed-NAS) is more difficult than searching in a progressively growing search space (*i.e.*, CNAS). Since our CNAS learns the controller in a progressive manner, the knowledge learned in the smaller search space will be transferred to the next training stage. With the help of knowledge inherited from the previous learning, CNAS finds better architectures than Fixed-NAS.

Compared with CNAS-Node, the architectures found by CNAS achieve better performance at the same epoch. As for the largest search space, the searched architecture of CNAS also yields significantly better performance than CNAS-Node one (97.40% vs. 97.15%). Moreover, the

improvement of performance between the last two stages in CNAS-Node becomes smaller. The reason is that the search space is increasing more quickly with the addition of nodes (See Figure 2), which introduces a large gap between the last two stages. In contrast, the growth of the search space is more smooth with the addition of the operation, the gap between two adjacent stages of our proposed CNAS is smaller than CNAS-Node. As a result, CNAS finds better architectures than CNAS-Node.

6.2. Evaluation on CIFAR-10

We first search for the convolution cells with our proposed CNAS on CIFAR-10 data set. Then, we build the final convolution networks by stacking the learned cells and evaluate them on CIFAR-10 data set.

Training details. Following the setting in (Liu et al., 2019), convolution cells have two types, namely the normal cell and the reduction cell. Each cell contains 7 nodes, including 2 input nodes, 4 intermediate nodes, and 1 output node. The available operations between two nodes include 3×3 depthwise separable convolution, 5×5 depthwise separable convolution, 3×3 max pooling, 3×3 average pooling, 3×3 dilated convolution, 5×5 dilated convolution, identity and none. We force the first added operation to have parameters (*e.g.*, convolution) for the reason that the sampled network without parameters cannot be trained. We divide the official training set of CIFAR-10 into two parts, 40% for training the super network parameters and 60% for training the controller parameters. We train the controller for 320 epochs in total, with 40 epochs for each stage. Before adding operations at each stage, we perform the operation warmup for 20 epochs. More details can be found in the supplementary.

Evaluation details. The final convolution network is stacked with 20 learned cells: 18 normal cells and 2 reduction cells. We set the initial number of the channels to 36. Following (Liu et al., 2019), we train the network for 600 epochs using the batch size of 96. We use an SGD optimizer with a weight decay of 3×10^{-4} and a momentum of 0.9. The learning rate starts from 0.025 and follows the cosine annealing strategy to a minimum of 0.001. We use cutout (DeVries & Taylor, 2017) with a length of 16 for data augmentation. We report the mean and standard deviation of 10 independent experiments for our final convolution network. More details can be found in the supplementary.

Comparisons with state-of-the-art methods. We compare our CNAS with state-of-the-art methods in Table 1 and show the learned normal and reduction cells in Figure 5. The architecture found by CNAS achieves the average test accuracy of 97.40%, which outperforms all the considered methods. By searching in the progressively growing search space, our CNAS makes use of the knowledge inherited rather than train from scratch. In this way, the architecture

Table 1. Comparisons with state-of-the-art models on CIFAR-10. We report the mean and standard deviation of the test accuracy over 10 independent experiments for different models.

Architecture	Test Accuracy (%)	Params (M)	Search Cost (GPU days)
DenseNet-BC (Huang et al., 2017)	96.54	25.6	–
PyramidNet-BC (Han et al., 2017)	96.69	26.0	–
Random search baseline	96.71 ± 0.15	3.2	–
NASNet-A + cutout (Zoph et al., 2018)	97.35	3.3	1800
NASNet-B (Zoph et al., 2018)	96.27	2.6	1800
NASNet-C (Zoph et al., 2018)	96.41	3.1	1800
AmoebaNet-A + cutout (Real et al., 2019)	96.66 ± 0.06	3.2	3150
AmoebaNet-B + cutout (Real et al., 2019)	96.63 ± 0.04	2.8	3150
DSO-NAS (Zhang et al., 2018b)	97.05	3.0	1
Hierarchical Evo (Liu et al., 2018b)	96.25 ± 0.12	15.7	300
SNAS (Xie et al., 2019)	97.02	2.9	1.5
ENAS + cutout (Pham et al., 2018)	97.11	4.6	0.5
NAONet (Luo et al., 2018)	97.02	28.6	200
NAONet-WS (Luo et al., 2018)	96.47	2.5	0.3
GHN (Zhang et al., 2018a)	97.16 ± 0.07	5.7	0.8
PNAS + cutout (Liu et al., 2018a)	97.17 ± 0.07	3.2	225
DARTS + cutout (Liu et al., 2019)	97.24 ± 0.09	3.4	4
CARS + cutout (Yang et al., 2019)	97.38	3.6	0.4
CNAS + cutout	**97.40 ± 0.06**	3.7	0.3

(a) Normal cell.

(b) Reduction cell.
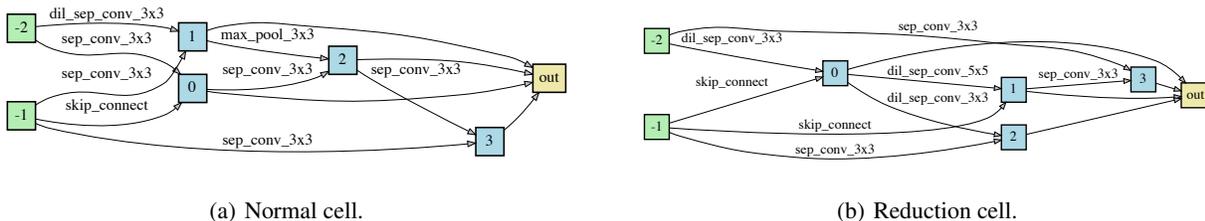
Figure 5. The architecture of the convolutional cells found by CNAS. We conduct architecture search on CIFAR-10 and evaluate the architecture on both CIFAR-10 and ImageNet datasets.

search problem becomes simpler. As a result, CNAS finds better architectures than other methods.

6.3. Evaluation on ImageNet

To verify the transferability of the learned cells on CIFAR-10, we evaluate them on a large-scale image classification data set ImageNet, which contains 1,000 classes with 128k training images and 50k testing images.

Evaluation details. We stack 14 cells searched on CIFAR-10 to build the final convolution network, with 12 normal cells and 2 reduction cells. The initial number of the channels is set to 48. Following the settings in (Liu et al., 2019), the network is trained for 250 epochs with a batch size of 256. We use an SGD optimizer with a weight decay of 3×10^{-5}. The momentum term is set to 0.9. The learning rate is initialized to 0.1 and we gradually decrease it

to zero. Following the setting in (Pham et al., 2018; Liu et al., 2018a; 2019), we consider the mobile setting where multiply-adds (Madds) is restricted to be less than 600M. More details can be found in the supplementary.

Comparisons with state-of-the-art methods. We compare the performance of the architecture found by CNAS with several state-of-the-art models and report the results in Table 2. Under the mobile setting, the architecture found by CNAS achieves 75.4% top-1 accuracy and 92.6% top-5 accuracy, outperforming the human-designed architectures and NAS based architectures. Moreover, compared with NASNet-A, AmoebaNet-A, and PNAS, our CNAS architecture also achieves competitive performance even with two or three orders of magnitude fewer computation resources. Compared with other heavyweight model, *e.g.*, ResNet-18 and Inception-v1, our model yields better performance with significantly less computation cost.

Table 2. Comparisons with state-of-the-art image classifiers on ImageNet. "-" denotes the results that are not reported.

Architecture	Test Accuracy (%)		#Params (M)	#MAdds (M)	Search Cost (GPU days)
	Top-1	Top-5			
ResNet-18 (He et al., 2016)	69.8	89.1	11.7	1814	–
Inception-v1 (Szegedy et al., 2015)	69.8	89.9	6.6	1448	–
MobileNet (Howard et al., 2017)	70.6	89.5	4.2	569	–
NASNet-A (Zoph et al., 2018)	74.0	91.6	5.3	564	1800
NASNet-B (Zoph et al., 2018)	72.8	91.3	5.3	488	1800
NASNet-C (Zoph et al., 2018)	72.5	91.0	4.9	558	1800
AmoebaNet-A (Real et al., 2019)	74.5	92.0	5.1	555	3150
AmoebaNet-B (Real et al., 2019)	74.0	92.4	5.3	555	3150
GHN (Zhang et al., 2018a)	73.0	91.3	6.1	569	0.8
SNAS (Xie et al., 2019)	72.7	90.8	4.3	522	1.5
DARTS (Liu et al., 2019)	73.1	91.0	4.9	595	4
NAT-DARTS (Guo et al., 2019)	73.7	91.4	4.0	441	-
PNAS (Liu et al., 2018a)	73.5	91.4	5.1	588	255
MnasNet-92 (Tan et al., 2019)	74.8	92.0	4.4	-	-
ProxylessNAS (Cai et al., 2019)	75.1	92.5	7.1	-	8.3
CARS (Yang et al., 2019)	75.2	92.5	5.1	591	0.4
CNAS	**75.4**	**92.6**	5.3	576	**0.3**

7. Further Experiments

We conduct two further experiments to investigate the effect of operation warmup and different operation orders.

7.1. Effect of Operation Warmup

We investigate the effect of operation warmup on the search performance of CNAS. For a fair comparison, we train different controllers with the same number of epochs. From Figure 4, without operation warmup, the controller tends to find sub-optimal architectures and the search performance is also very unstable during the training phase. When equipped with the proposed operation warmup, the resultant controller consistently outperforms that without operation warmup in all training stages. These results demonstrate the necessity and effectiveness of the proposed operation warmup.

7.2. Effect of Different Operation Orders

We compare the performance of the architectures searched by CNAS with different operation orders. Since the search space is gradually enlarged by adding operations, different operation orders may correspond to different search spaces, leading to different searched architectures. We repeat the search experiment 5 times with the same settings except for the orders of adding operations on CIFAR-10. We report the mean accuracy of these architectures found by CNAS over 5 runs in Figure 4. CNAS achieves better mean accuracy than Fixed-NAS with different operation orders. The experimental results indicate the proposed CNAS is not sensitive to the orders of the operations.

8. Conclusion

In this paper, we have proposed a Curriculum Neural Architecture Search (CNAS) method to alleviate the training difficulty incurred by the space explosion issue. To this end, we propose a curriculum search strategy that breaks the NAS problem into a series of simpler problems and solves them in a progressive manner. Specifically, we solve the NAS problems by gradually enlarging the search spaces and incorporating the learned knowledge to guide the search. To construct these problems, we gradually introduce new operations into the search space. By inheriting the knowledge learned from the smaller search spaces, CNAS can greatly improve the search performance in the largest space. Extensive experiments on CIFAR-10 and ImageNet demonstrate the superiority of CNAS over existing methods.

Acknowledgements

References

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *International Conference on Machine Learning*, pp. 41–48, 2009.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35 (8):1798–1828, 2013.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Cao, J., Guo, Y., Wu, Q., Shen, C., Huang, J., and Tan, M. Adversarial learning with local coordinate coding. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 707–715, 2018.

Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.

Chen, X., Zheng, Y., Zhao, P., Jiang, Z., Ma, W., and Huang, J. A generalized locally linear factorization machine with supervised variational encoding. *IEEE Transactions on Knowledge and Data Engineering*, 32(6):1036–1049, 2019.

Cheng, A.-C., Lin, C. H., Juan, D.-C., Wei, W., and Sun, M. Instanas: Instance-aware neural architecture search. *arXiv preprint arXiv:1811.10201*, 2018.

Chu, X., Zhang, B., Xu, R., and Li, J. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Gülçehre, Ç. and Bengio, Y. Knowledge matters: Importance of prior information for optimization. *The Journal of Machine Learning Research*, 17(1):226–257, 2016.

Guo, Y., Zheng, Y., Tan, M., Chen, Q., Chen, J., Zhao, P., and Huang, J. Nat: Neural architecture transformer for accurate and compact architectures. In *Advances in Neural Information Processing Systems*, pp. 737–748, 2019.

Guo, Y., Chen, J., Du, Q., Van Den Hengel, A., Shi, Q., and Tan, M. Multi-way backpropagation for training compact deep neural networks. *Neural networks*, 2020a.

Guo, Y., Chen, J., Wang, J., Chen, Q., Cao, J., Deng, Z., Xu, Y., and Tan, M. Closed-loop matters: Dual regression networks for single image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5407–5416, 2020b.

Han, D., Kim, J., and Kim, J. Deep pyramidal residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hosseini, B., Montagne, R., and Hammer, B. Deep-aligned convolutional neural network for skeleton-based action recognition and segmentation. *Data Science and Engineering*, pp. 126–139, 2020.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2261–2269, 2017.

Huang, Z., Wang, X., Huang, L., Huang, C., Wei, Y., and Liu, W. Ccnet: Criss-cross attention for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 603–612, 2019.

Jiang, Z., Zheng, Y., Tan, H., Tang, B., and Zhou, H. Variational deep embedding: An unsupervised and generative approach to clustering. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pp. 1965–1972, 2017.

Khan, F., Mutlu, B., and Zhu, J. How do humans teach: On curriculum learning and teaching dimension. In *Advances in Neural Information Processing Systems*, pp. 1449–1457, 2011.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Kumar, M. P., Packer, B., and Koller, D. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pp. 1189–1197, 2010.

Lian, D., Zheng, Y., Xu, Y., Lu, Y., Lin, L., Zhao, P., Huang, J., and Gao, S. Towards fast adaptation of neural architectures with meta learning. In *International Conference on Learning Representations*, 2020.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision*, 2018a.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018b.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*, 2019.

Liu, J., Zhuang, B., Zhuang, Z., Guo, Y., Huang, J., Zhu, J., and Tan, M. Discrimination-aware network pruning for deep model compression. *arXiv preprint arXiv:2001.01050*, 2020.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, pp. 7827–7838, 2018.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher-student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pp. 4092–4101, 2018.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019.

Shelhamer, E., Long, J., and Darrell, T. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and 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.

Wang, Y., Gan, W., Yang, J., Wu, W., and Yan, J. Dynamic curriculum learning for imbalanced data classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5017–5026, 2019.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., and Xu, C. Cars: Continuous evolution for efficient neural architecture search. *arXiv preprint arXiv:1909.04977*, 2019.

Zeng, R., Huang, W., Tan, M., Rong, Y., Zhao, P., Huang, J., and Gan, C. Graph convolutional networks for temporal action localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7094–7103, 2019.

Zhang, C., Ren, M., and Urtasun, R. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2018a.

Zhang, D., Han, J., Zhao, L., and Meng, D. Leveraging prior-knowledge for weakly supervised object detection under a collaborative self-paced curriculum learning framework. *International Journal of Computer Vision*, 127(4):363–380, 2019.

Zhang, X., Huang, Z., and Wang, N. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567*, 2018b.

Zheng, Y., Zemel, R. S., Zhang, Y.-J., and Larochelle, H. A neural autoregressive approach to attention-based recognition. *International Journal of Computer Vision*, 2015.

Zheng, Y., Tang, B., Ding, W., and Zhou, H. A neural autoregressive approach to collaborative filtering. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 2016a.

Zheng, Y., Zhang, Y.-J., and Larochelle, H. A deep and autoregressive approach for topic modeling of multimodal data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):1056–1069, 2016b.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and 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.