Contrastive Neural Architecture Search with Neural Architecture Comparators

Yaofo Chen[1*], Yong Guo[1*], Qi Chen[1], Minli Li[1], Yaowei Wang[2], Wei Zeng[3], Mingkui Tan[1,4†]
[1]South China University of Technology, [2]Peng Cheng Laboratory, [3]Peking University,
[4]Key Laboratory of Big Data and Intelligent Robot, Ministry of Education
{sechenyaofo, guo.yong, sechenqi, seminli_li}@mail.scut.edu.cn,
wangyw@pcl.ac.cn, weizeng@pku.edu.cn, mingkuitan@scut.edu.cn

Abstract

One of the key steps in Neural Architecture Search (NAS) is to estimate the performance of candidate architectures. Existing methods either directly use the validation performance or learn a predictor to estimate the performance. However, these methods can be either computationally expensive or very inaccurate, which may severely affect the search efficiency and performance. Moreover, as it is very difficult to annotate architectures with accurate performance on specific tasks, learning a promising performance predictor is often non-trivial due to the lack of labeled data. In this paper, we argue that it may not be necessary to estimate the absolute performance for NAS. On the contrary, we may need only to understand whether an architecture is better than a baseline one. However, how to exploit this comparison information as the reward and how to well use the limited labeled data remains two great challenges. In this paper, we propose a novel Contrastive Neural Architecture Search (CTNAS) method which performs architecture search by taking the comparison results between architectures as the reward. Specifically, we design and learn a Neural Architecture Comparator (NAC) to compute the probability of candidate architectures being better than a baseline one. Moreover, we present a baseline updating scheme to improve the baseline iteratively in a curriculum learning manner. More critically, we theoretically show that learning NAC is equivalent to optimizing the ranking over architectures. Extensive experiments in three search spaces demonstrate the superiority of our CTNAS over existing methods.

1. Introduction

Deep neural networks (DNNs) have made significant progress in various challenging tasks, including image classification [40, 43, 13], face recognition [31, 46], and many other areas [5, 55, 56, 6]. One of the key factors behind the progress lies in the innovation of effective neural architectures, such as ResNet [20] and MobileNet [22]. However, designing effective architectures is often labor-intensive and relies heavily on human expertise. Besides designing architectures manually, one can also use Neural Architecture Search (NAS) methods [58] to design architectures automatically. In practice, the automatically searched architectures often outperform the hand-crafted ones in various tasks [1, 37].

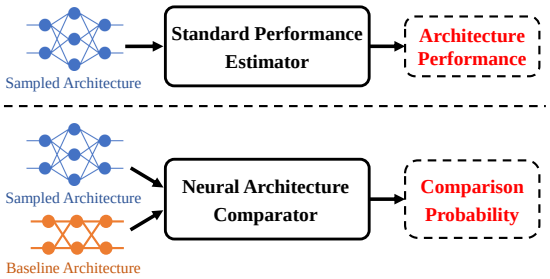


Figure 1. Comparison between the standard performance estimator (top) and our NAC (bottom). Unlike the standard estimator that predicts the absolute performance, our NAC takes two architectures as inputs and outputs the comparison probability of the sampled architectures being better than the baseline architecture.

Existing NAS methods seek to find the optimal architecture in a predefined search space by maximizing the expectation of the performance of the searched architectures. Thus, how to estimate the performance of architectures is a key step in NAS. In practice, the searched architectures can be evaluated by the absolute performance provided by a supernet [3, 8, 37] or a predictor [27, 33]. However, using the absolute performance as the training signal may suffer from two limitations.

First, it is non-trivial to obtain stable and accurate absolute performance for all the candidate architectures. In practice, the performance of architectures may fluctuate a lot under the training with different random seeds [26, 29].

*Authors contributed equally.
†Corresponding author.

Thus, there would be a large performance deviation if we evaluate the architecture only with a single value w.r.t. absolute performance. As a result, using the absolute performance as the training signals may greatly hamper the search performance. Based on such signals, a randomly searched architecture may even outperform the architectures obtained by existing NAS methods [26, 54] in practice. Thus, how to obtain stable and accurate training signals to guide the search is an important problem.

Second, it is time-consuming to obtain the absolute performance from the supernet during the search process. Specifically, one can evaluate an architecture by feeding in a set of validation data to obtain the accuracy. However, given a large number of validation data, obtaining the validation accuracy for candidate architectures via forward propagation can be computationally expensive. To address this issue, one can learn a regression model to predict the performance of architectures [27, 33]. However, the training of predictor models still requires plenty of architectures with the ground-truth performance as the training data, which are very expensive to obtain in practice. Thus, how to efficiently evaluate architectures with limited architectures with ground-truth performance becomes an urgent and important problem.

In this paper, we propose a Contrastive Neural Architecture Search (CTNAS) method that finds promising architectures by comparing different architectures. To address the first limitation, we devise a Neural Architecture Comparator (NAC) to perform pairwise architecture comparison. Unlike existing methods that rely on the absolute performance, we use the comparison results between the searched architectures and a baseline one as the reward (See Figure 1). In practice, the pairwise comparison results are easier to obtain and also more stable than the absolute performance (See analysis in Section 3.1). To constantly find better architectures during the search, we propose to improve the baseline gradually via a curriculum learning manner. To address the second limitation, the proposed NAC evaluates architectures via pairwise comparisons and avoid performing forward propagation on validation data. Thus, the evaluation can be much more efficient and greatly accelerate the search process (See Table 5.2). Moreover, we also propose a data exploration method that exploits the architectures without ground-truth performance to improve the generalization ability of NAC to unseen architectures. In this way, we are able to effectively reduce the requirement of the training data.

Our contributions are summarized as follows.

- We propose a Contrastive Neural Architecture Search (CTNAS) method that searches for promising architectures by taking the comparison results between architectures as the reward.
- To guarantee that CTNAS can constantly find better architectures, we propose a curriculum updating scheme to gradually improve the baseline architecture. In this way, CTNAS has a more stable search process and thus greatly improves the search performance.
- Extensive experiments on three search spaces demonstrate that the searched architectures of our CTNAS outperform the architectures searched/designed by state-of-the-art methods.

2. Related Work

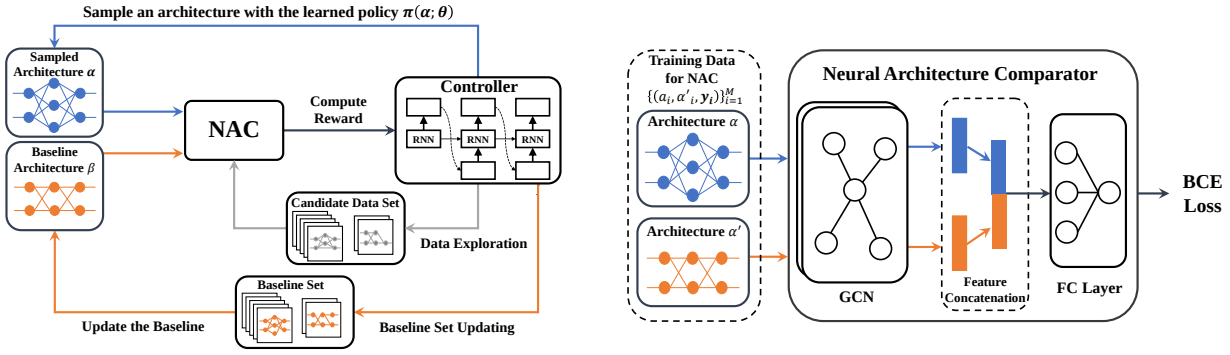
Neural Architecture Search. NAS seeks to automatically design neural architectures by finding an optimal one in some search space. The pioneering work [58] exploits the paradigms of reinforcement learning (RL) to solve it. RL-based methods [1, 11, 37, 44, 58, 59, 15] seek to learn a controller with a policy $\pi(\alpha;\theta)$ to generate architectures, *i.e.*, $\alpha \sim \pi(\alpha;\theta)$, where α denotes a sampled architectue, θ denotes the parameters of the policy. Specifically, they learn the controller by maximizing the expectation of some performance metric $\mathcal{R}(\alpha, w_\alpha)$ (*e.g.*, validation accuracy),

$$\max_{\theta} \ \mathbb{E}_{\alpha\sim\pi(\alpha;\theta)}\mathcal{R}\left(\alpha, w_\alpha\right), \tag{1}$$

where $w_\alpha = \arg\min_w \mathcal{L}\left(\alpha, w\right)$ and $\mathcal{L}\left(\alpha, w\right)$ is the training loss. Moreover, some studies [28, 38, 39] search for promising architectures using evolutionary algorithms. Different from these methods searched within a discrete space, gradient based methods [7, 29, 48, 50] represent architectures by continuous relaxation and optimize by gradient descent. Unlike existing methods that obtain/predict the absolute performance, we seek to conduct architecture comparison to guide the search process. Based on RL, our method maximizes the expectation of the comparison probability of the sampled architectures being better than a baseline one instead of the absolute performance.

Contrastive Learning. Contrastive learning aims to learn the similarity/dissimilarity over the samples by performing comparisons among them. Specifically, Hadsell *et al.* [18] propose a contrastive loss to solve the dimensionality reduction problem by performing pairwise comparison among different samples. Based on the contrastive loss, Sohn [42] proposes a Multi-class N-pair loss to allow joint comparison among multiple negative data pairs. As for NAS, finding the optimal architecture can be considered as a ranking problem over a set of candidate architectures [51]. In this sense, it is possible to solve the ranking problem of NAS by conducting comparisons among architectures.

Comparisons with ReNAS [53]. ReNAS exploits a ranking loss to learn a predictor model that predicts the relative performance of architectures. However, ReNAS is essentially different from our method and has several limitations. **First**, they have different search methods. ReNAS searches for promising architectures with the predicted scores of architectures. Unlike ReNAS, our CTNAS proposes a contrastive



(a) Overall scheme of Contrastive Neural Architecture Search

(b) Neural Architecture Comparator

Figure 2. The overview of CTNAS and NAC. (a) The proposed NAC first takes the sampled architecture and the baseline one as inputs, and outputs the comparison probability of them. Then, our CTNAS adopts the probability as the reward to train the controller. During the training, we update the baseline sampled from the controller. Besides, we perform data exploration on the sampled architectures to construct a candidate data set. (b) We optimize NAC with the binary cross-entropy (BCE) loss computed by the comparison probability and the label indicated which one is better between two input architectures.

architecture search scheme that searches by comparing the sampled architectures with a gradually improved baseline. **Second**, they have different generalization abilities to unseen architectures. ReNAS heavily relies on the training data and thus may be hard to generalize to unseen architectures. In contrast, the proposed CTNAS introduces a data exploration method to improve the generalization, leading to a benefit to find promising architectures (See results in Table 1).

3. Proposed Method

Notation. Throughout the paper, we use the following notations. We use calligraphic letters (*e.g.*, $\mathcal{A}$) to denote a set. We use the lower case of Greek characters (*e.g.*, α) to denote architectures. Let Ω be the search space. For any architecture $\alpha \in \Omega$, let w_α be its well-learned model parameters trained on some data set. Let $\mathbb{1}\{\cdot\}$ be an indicator function, where $\mathbb{1}\{A\} = 1$ if A is true and $\mathbb{1}\{A\} = 0$ if A is false. Let $\Pr[\cdot]$ be the probability for some event.

In this paper, we propose a Contrastive Neural Architecture Search (CTNAS) that conducts architecture search by taking the comparisons between architectures as the reward. To ensure that CTNAS is able to constantly find better architectures, we seek to gradually improve/update the baseline via a curriculum learning manner. We show the training method of CTNAS in Algorithm 1.

3.1. Motivation

In neural architecture search (NAS), one of the key steps is to evaluate the candidate architectures. Most existing methods evaluate architectures through the absolute performance $\mathcal{R}(\alpha, w_\alpha)$ with w_α from a learned supernet [3, 8, 17, 37]. However, finding promising architectures with the absolute performance comes with two challenges. First, there would be deviation on the absolute performance with different training random seeds. Searching with this fluctuated performance may incur training difficulties for the NAS model. Second, obtaining absolute performance with a learned supernet is computationally expensive since it needs to compute the accuracy on plenty of validation data. Given a large number of candidate architectures in the search process, it would be very time-consuming (*e.g.*, several GPU days).

In this paper, we seek to improve the search performance by designing a more effective evaluation method. To begin

Algorithm 1 The overall algorithm for CTNAS.

Require: Learning rate η for policy gradient, parameters M, N and K ($K \ll N$).

1: Randomly sample a set of architectures from Ω and obtain their accuracy $\{\alpha_i, \mathcal{R}(\alpha_i, w_{\alpha_i})\}_{i=1}^{M}$ by training a supernet.
2: Construct training data $\mathcal{A}=\{(\alpha_i, \alpha'_i, y_i)\}_{i=1}^{M(M-1)/2}$ for NAC by traversing all pairwise combinations.
3: Initialize parameters θ for $\pi(\cdot;\theta)$ and ϖ for NAC.
4: Initialize the baseline architecture $\beta \sim \pi(\cdot;\theta)$.
5: Let $\mathcal{C}=\mathcal{A}$, $\mathcal{D}=\emptyset$, $\mathcal{B}=\emptyset$.
6: **for** $t = 1, \ldots, T$ **do**
7: Train NAC with data $\mathcal{C}=\{(\alpha_i, \alpha'_i, y_i)\}_{i=1}^{|\mathcal{C}|}$.
8: *// Train the controller with NAC*
9: Sample N architectures $\{\alpha_j\}_{j=1}^{N}$ by $\alpha \sim \pi(\cdot;\theta)$.
10: Update θ using policy gradient:
$\theta \leftarrow \theta + \eta \frac{1}{N} \sum_{j=1}^{N} [\nabla_\theta \log \pi(\alpha_j;\theta) \mathrm{NAC}(\alpha_j, \beta; \varpi)]$.
11: *// Explore more data for training NAC*
12: Sample N architectures $\mathcal{S}=\{\alpha_i\}_{i=1}^{N} \sim \pi(\cdot;\theta)$.
13: Construct $\mathcal{D}$ with $\mathcal{S}$ by data exploration using Alg. 3.
14: Let $\mathcal{C} = \mathcal{C} \cup \mathcal{D}$ and $\mathcal{B} = \mathcal{B} \cup \{\beta\}$.
15: Update the baseline β with $\mathcal{B}$ and $\mathcal{S}$ using Alg. 2.
16: **end for**

with, let us revisit the definition of optimal architecture. Suppose that α^* is the optimal architecture in the search space Ω, we would have $\mathcal{R}(\alpha^*, w_{\alpha^*}) \geq \mathcal{R}(\alpha, w_\alpha), \forall\, \alpha \in \Omega$. However, since $\mathcal{R}(\alpha, w_\alpha)$ may not be very accurate, it is more reasonable and rigorous to require that $\mathcal{R}(\alpha^*, w_{\alpha^*}) \geq \mathcal{R}(\alpha, w_\alpha)$ holds in high probability. For example, it is often much easier to recognize which one is better among two architectures compared to estimating the absolute performance of them. In this way, the comparison results become more stable and may reduce the influence of training fluctuations. Thus, to ensure the optimality, we only need to compute

$$\Pr[\mathcal{R}(\alpha^*, w_{\alpha^*}) \geq \mathcal{R}(\alpha, w_\alpha)]. \tag{2}$$

The above probability implies that we may not need to obtain the absolute performance to solve the NAS problem.

3.2. Contrastive Neural Architecture Search

In this paper, we propose a Contrastive Neural Architecture Search (CTNAS) method that finds promising architectures via architecture comparisons. Unlike existing methods that rely on the absolute performance, we seek to obtain the ranking of candidate architectures using a series of pairwise comparisons. Specifically, we can learn a comparison mapping, called Neural Architecture Comparator (NAC), to compare any two architectures $\alpha, \alpha' \in \Omega$ and output the probability of α being better than α':

$$p = \Pr[\mathcal{R}(\alpha, w_\alpha) \geq \mathcal{R}(\alpha', w_{\alpha'})] = \text{NAC}(\alpha, \alpha'; \varpi), \tag{3}$$

where ϖ is the parameter of NAC. The comparison probability predicted by NAC is more stable than the absolute performance since it may reduce negative impact of accuracy deviation. For simplicity, we leave the details of NAC in the following section.

From Eqn. (3), it is possible to use the comparison probability predicted by NAC as the reward signal to train the NAS model. Formally, given a baseline architecture $\beta \in \Omega$, we hope to learn a policy $\pi(\alpha; \theta)$ by solving the following optimization problem:

$$\max_{\theta}\ \mathbb{E}_{\alpha \sim \pi(\alpha;\theta)} \Pr[\mathcal{R}(\alpha, w_\alpha) \geq \mathcal{R}(\beta, w_\beta)], \tag{4}$$

where θ denotes the parameters of the policy. To address the above optimization problem, following [37, 58], we train a controller with policy gradient [47]. Unlike existing reinforcement learning based NAS methods, we adopt the comparison probability p=NAC$(\alpha, \beta; \varpi)$ as the reward. Given a specific β, the controller seeks to conduct a comparison with it to search for better architectures.

However, solving Problem (4) can only enable the model to find architectures that are better than the baseline β. In other words, it may not find the optimal architecture. Moreover, in Problem (4), if β is too weak or strong, the above optimization problem becomes meaningless (*i.e.*, the optimal objective value will be trivially 1 or 0, respectively). To address this issue, we propose a baseline updating scheme to improve/update the baseline gradually. In this way, our CTNAS is able to find better architectures iteratively. We will detail the baseline updating method in the following.

Algorithm 2 Baseline updating via curriculum learning.

Require: Existing baseline architectures $\mathcal{B}$, sampled architectures $\mathcal{S}$, and learned architecture comparator NAC$(\cdot, \cdot; \varpi)$.

1: Initialize comparison score $\hat{s} = 0$.
2: Construct a candidate baseline set $\mathcal{H} = \mathcal{B} \cup \mathcal{S} = \{\alpha_i\}_{i=1}^{|\mathcal{H}|}$.
3: **for** $i = 1, \ldots, |\mathcal{H}|$ **do**
4: Compute score for architecture $\alpha_i \in \mathcal{H}$ by

$$s_i = \frac{1}{|\mathcal{H}| - 1} \sum_{1 \leq j \leq |\mathcal{H}|, i \neq j} \text{NAC}(\alpha_i, \alpha_j; \varpi).$$

5: **if** $s_i \geq \hat{s}$ **then** $\hat{s} = s_i$ and $\beta = \alpha_i$. **end if**
6: **end for**
7: Return β.

3.3. Baseline Updating via Curriculum Learning

Since CTNAS takes the comparison result with the baseline architecture as the reward, the search performance heavily relies on the baseline architecture. Given a fixed baseline architecture, the controller is only able to find better architectures than the baseline. If the baseline is not good enough, the searched architecture cannot be guaranteed to be a promising one. Thus, it becomes necessary to gradually improve/update the baseline architecture during the search.

To this end, we propose a curriculum updating scheme to improve the baseline during the search process (See Algorithm 2). The key idea follows the basic concept of curriculum learning that humans and animals can learn much better when they gradually learn new knowledge [2, 12]. Specifically, we break the whole NAS problem into a series of simpler sub-problems, *i.e.*, solving Problem (4) with a gradually improved baseline architecture. Since we gradually improve the baseline, our CTNAS is able to constantly find better architectures during the search.

To improve the baseline architecture β, we seek to select the best architecture from the set of previously searched architectures (See Algorithm 2). Specifically, we build a candidate baseline set $\mathcal{H}$ and dynamically incorporate sampled architectures into it. To avoid the negative influence from the possible error of NAC in a single comparison, for any architecture $\alpha_i \in \mathcal{H}$, we compute the average comparison probability s_i by comparing α_i with all the other architectures in $\mathcal{H}$:

$$s_i = \frac{1}{|\mathcal{H}| - 1} \sum_{1 \leq j \leq |\mathcal{H}|, i \neq j} \text{NAC}(\alpha_i, \alpha_j; \varpi). \tag{5}$$

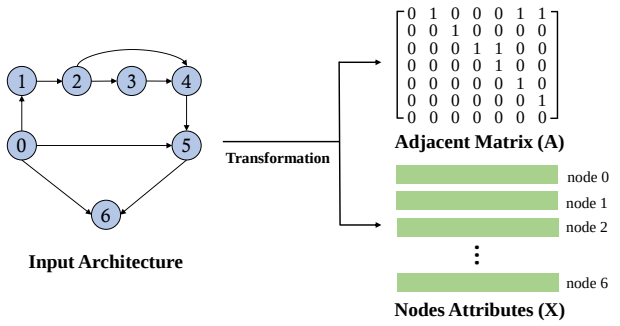


Figure 3. Architecture representation method of the proposed CTNAS. The nodes of the architecture DAG indicate the operations. The edges represent the flow of information.

Based on the best architecture in the past as the baseline, our CTNAS is able to further improve the search performance by finding better architectures than the baseline.

Compared with existing NAS methods, our CTNAS is more stable and reliable since the contrastive search scheme searches for better architectures than the best one of the previously searched architectures. Thus, our CTNAS consistently finds better architectures than existing methods on different search spaces (See results in Tables 1 and 2).

4. Neural Architecture Comparator

To provide a valid reward for CTNAS, we propose a neural architecture comparator (NAC) that compares any two architectures. To guarantee that NAC can handle any architecture, we represent an architecture as a directed acyclic graph (DAG) [37] and build NAC using a graph convolutional network (GCN) to calculate the comparison probability. Moreover, we develop a data exploration method that trains NAC in a semi-supervised way to reduce the requirement of the training data.

4.1. Architecture Comparison by GCN

Given an architecture space Ω, we can represent an architecture $\alpha \in \Omega$ as a directed acyclic graph (DAG) [37]. As shown in Figure 3, each node represents a computational operation (*e.g.*, convolution or max pooling) and each edge represents the flow of information. Following [24], we represent the graph α using a data pair $(\mathbf{A}_\alpha, \mathbf{X}_\alpha)$, where $\mathbf{A}_\alpha$ denotes the adjacency matrix of the graph and $\mathbf{X}_\alpha$ contains the attributes of the nodes/operations.

The proposed NAC compares any two architectures and predicts the comparison probability as the reward. To exploit the connectivity information inside the architecture graph, we build the NAC model with a graph convolutional network. Specifically, given two architectures α and α' as inputs, the proposed NAC predicts the probability of α being better than α' (See Figure 2 (b)). To calculate the comparison probability, we concatenate the features of α and α' and send them to a fully-connected (FC) layer. Then, the sigmoid function $\sigma(\cdot)$ takes the output of the FC layer as input and outputs the comparison probability:

$$p = \mathrm{NAC}(\alpha, \alpha'; \varpi) = \sigma\left([\mathbf{Z}_\alpha; \mathbf{Z}_{\alpha'}]\mathbf{W}^{FC}\right), \quad (6)$$

where Z_α denotes the features extract from the architecture α, $\mathbf{W}^{FC}$ denotes the weight of the FC layer, $[\mathbf{Z}_\alpha; \mathbf{Z}_{\alpha'}]$ refers to the concatenation of the features of α and α'. Based on the graph data pair $(\mathbf{A}_\alpha, \mathbf{X}_\alpha)$, we use a two-layer GCN to extract the architecture features $\mathbf{Z}_\alpha$ following [16]:

$$\mathbf{Z}_\alpha = f(\mathbf{X}_\alpha, \mathbf{A}_\alpha) = \mathbf{A}_\alpha \phi\left(\mathbf{A}_\alpha \mathbf{X}_\alpha \mathbf{W}^{(0)}\right)\mathbf{W}^{(1)}, \quad (7)$$

where $\mathbf{W}^{(0)}$ and $\mathbf{W}^{(1)}$ denote the weights of two graph convolutional layers, ϕ is the a non-linear activation function (*e.g.*, the Rectified Linear Unit (ReLU) [36]), and $\mathbf{Z}_\alpha$ refers to the extracted features.

To train the proposed NAC, we need a data set that consists of architectures and their performance evaluated on some data set (*e.g.*, CIFAR-10). The data pairs of architecture and its performance can be obtained by training a supernet or training a set of architectures from scratch. Based on the data set, we define the label for any pair (α_i, α_i'), $\alpha_i \neq \alpha_i'$, sampled from Ω, by $y_i = \mathbb{1}\{\mathcal{R}(\alpha_i, w_{\alpha_i}) - \mathcal{R}(\alpha_i', w_{\alpha_i'}) \geq 0\}$, and construct the training data $(\alpha_i, \alpha_i', y_i)$. Thus, the training of NAC can be considered a binary classification problem (*i.e.*, the label $y \in \{0, 1\}$). We solve the problem by optimizing the binary cross-entropy loss between the probability p predicted by NAC and the ground truth label y.

In the following, we will discuss the relationship between architecture comparison and ranking problems. For convenience, we rewrite $\mathcal{R}(\alpha, w_\alpha)$ as $\mathcal{R}_\alpha$. Given a data set $\{\alpha_i, \mathcal{R}_{\alpha_i}\}_{i=1}^M$, we seek to find a ranking function $f : \Omega \times \Omega \to \mathbb{R}$ by optimizing the ranking loss $\ell_0(\alpha, \alpha', \mathcal{R}_\alpha, \mathcal{R}_{\alpha'}) = \mathbb{1}\{(\mathcal{R}_\alpha - \mathcal{R}_{\alpha'}) f(\alpha, \alpha') \leq 0\}$. However, directly optimizing ℓ_0 is non-trivial since the indicator

Algorithm 3 Data exploration for NAC.

Require: Sampled architecture set $\mathcal{S}=\{\alpha_i\}_{i=1}^N$, learned architecture comparator $\mathrm{NAC}(\cdot,\cdot;\varpi)$, and parameter K.

Initialize the set of predicted label data $\mathcal{E}=\emptyset$ and the confidence score set $\mathcal{F}=\emptyset$.
Build $\mathcal{G}=\{(\alpha_k, \alpha_k')\}, \alpha_k, \alpha_k' \in \mathcal{S}, \alpha_k \neq \alpha_k'$.
for $k = 1, \ldots, |\mathcal{G}|$ **do**
  p_k=NAC$(\alpha_k, \alpha_k'; \varpi)$. *// Compute comparison probability*
  y_k'=$\mathbb{1}\{p_k \geq 0.5\}$.*// Compute label according to probability*
  Let $\mathcal{E} = \mathcal{E} \cup \{(\alpha_k, \alpha_k', y_k')\}$.
  f_k=$|p_k - 0.5|$. *// Compute confidence score*
  Let $\mathcal{F} = \mathcal{F} \cup \{f_k\}$.
end for
Select top-K architecture pairs according to confidence score $\mathcal{D}$ = top-$K(\mathcal{E}, \mathcal{F}, K)$.
Return $\mathcal{D}$.

Table 1. Comparisons with existing methods in NAS-Bench-101 search space. "–" represents unavailable results. "#Queries" denotes the number of pairs of architecture and its validation accuracy queried from NAS-Bench-101 dataset. All methods are run for 10 times.

Method	KTau	Average Accuracy (%)	Best Accuracy (%)	Best Rank (%)	#Queries
Random	–	89.31 ± 3.92	93.46	1.29	423
DARTS [29]	–	92.21 ± 0.61	93.02	13.47	–
ENAS [37]	–	91.83 ± 0.42	92.54	22.88	–
FBNet [48]	–	92.29 ± 1.25	93.98	0.05	–
SPOS [17]	0.195	89.85 ± 3.80	93.84	0.07	–
FairNAS [8]	-0.232	91.10 ± 1.84	93.55	0.77	–
ReNAS [53]	0.634	93.90 ± 0.21	94.11	0.04	423
RegressionNAS	0.430	89.51 ± 4.94	93.65	0.40	423
CTNAS (Ours)	**0.751**	**93.92 ± 0.18**	**94.22**	**0.01**	423

function is non-differential. To address this, we use a binary cross-entropy loss $\mathcal{L}(f;\alpha,\alpha',y) = \mathbb{E}\left[\ell(\sigma \circ f(\alpha,\alpha'),y)\right]$ as a surrogate for ℓ_0, where NAC uses the sigmoid activation function $\sigma(\cdot)$ for the output layer, and thus can be denoted as $\sigma \circ f$. We have the following proposition for NAC.

Proposition 1 *Let $f : \Omega \times \Omega \rightarrow \mathbb{R}$ be some measurable function, architectures $\alpha, \alpha' \in \Omega$. The surrogate loss $\mathcal{L}(f;\alpha,\alpha',y) = \mathbb{E}\left[\ell(\sigma \circ f(\alpha,\alpha'),y)\right]$ is consistent with $\mathcal{L}_0(f;\alpha,\alpha',\mathcal{R}_\alpha,\mathcal{R}'_\alpha) = \mathbb{E}\left[\mathbb{1}\{(\mathcal{R}_\alpha - \mathcal{R}'_\alpha)f(\alpha,\alpha') \leq 0\}\right]$.*

Proposition 1 shows that learning NAC is equivalent to optimizing the ranking over architectures. We put the proof in the supplementary.

Advantages of NAC over existing evaluation methods. 1) More stable evaluation results: our NAC is able to provide more stable and accurate reward signals by directly comparing architectures instead of estimating the absolute performance, leading to high rank correlation (See results in Sec. 5.1). **2) Lower evaluation time cost:** unlike existing methods evaluate architectures by computing accuracy on the validation data, our NAC achieves this only by comparing two architecture graphs. Thus, our method is able to greatly reduce the search cost and accelerate the search process (See more discussions in Sec. 6.1). **3) Lower requirement for training samples:** given m architectures with the corresponding performance, we can construct $\binom{m}{2} = m(m-1)/2$ training pairs to train NAC while the regression-based methods only have m training samples.

4.2. Data Exploration for Training NAC

Note that learning a good NAC requires a set of labeled data, *i.e.*, $\{(\alpha_i,\alpha'_i,y_i)\}_{i=1}^{M}$. However, we can only obtain a limited number of labeled data due to the limitation of computational cost in practice. Given a limited training data set, the performance of NAC model may deteriorate, leading to training difficulties for architecture search. Thus, how to efficiently evaluate architectures using NAC with limited data preparations is an important problem.

To address this issue, we propose a data exploration method that adopts the sampled architectures during the search as the unlabeled data. For these unlabeled data, we propose to take the class with maximum probability predicted by NAC as its label. As shown in Algorithm 3, given the latest NAC model $\text{NAC}(\cdot,\cdot;\varpi)$, the predicted label of the previously unseen architecture pair can be computed by

$$y' = \mathbb{1}\{\text{NAC}(\alpha,\alpha';\varpi) \geq 0.5\}, \tag{8}$$

where $\mathbb{1}\{\cdot\}$ refers to an indicator function. Here, $y' = 1$ if $\text{NAC}(\alpha,\alpha';\varpi) \geq 0.5$ and $y' = 0$ otherwise. However, the predicted label can be noisy since the NAC model may produce wrong predictions. To address this, for the k-th architecture pair, we evaluate the prediction quality by computing the confidence score:

$$f_k = \left|\text{NAC}(\alpha_k,\alpha'_k;\varpi) - 0.5\right|. \tag{9}$$

In practice, the higher the confidence score is, the more reliable the predicted label will be. We select the data with predicted labels with top-K confidence scores and combine them with the labeled data to train NAC. To balance these two kinds of data, we set the proportion of the predicted label data to 0.5 (See discussions in Sec. 6.3). With the increase of unlabelled data during training, our NAC is able to improve the generalization ability to unseen architectures.

5. Experiments

We apply our CTNAS to three different search spaces, namely NAS-Bench-101 [52], MobileNetV3 [21] and DARTS [29][1] search spaces. We put more details about the search space and the implementation in the supplementary. All implementations are based on PyTorch[2].

5.1. Experiments on NAS-Bench-101

Implementation Details. For a fair comparison, we set the number of architecture-accuracy pair queried from the NAS-Bench-101 dataset to 423 for all the methods. To measure the rank correlation, we use another 100 architectures

[1]Due to page limit, we put the experimental results in DARTS search space into the supplementary.

[2]The source code is available at https://github.com/chenyaofo/CTNAS.

Table 2. Comparisons of the architectures searched/designed by different methods on ImageNet. "–" means unavailable results. † denotes we test the accuracy from the pretrained model in the official repository. "Total Time" includes the time cost of training the supernet/child networks and the search process. "#Queries" denotes the number of architectures queried from the supernet for the validation accuracy.

Search Space	Architecture	Test Accuracy (%)		#MAdds (M)	#Queries (K)	Search Time (GPU days)	Total Time (GPU days)
		Top-1	Top-5				
	MobileNetV2 (1.4×) [40]	74.7	–	585	–	–	–
	ShuffleNetV2 (2×) [34]	73.7	–	524	–	–	–
NASNet	NASNet-A [59]	74.0	91.6	564	20	–	1800
	AmoebaNet-A [39]	74.5	92.0	555	20	–	3150
DARTS	DARTS [29]	73.1	91.0	595	19.5	4	4
	P-DARTS [7]	75.6	92.6	577	11.7	0.3	0.3
	PC-DARTS [50]	75.8	92.7	597	3.4	3.8	3.8
	CNAS [14]	75.4	92.6	576	100	0.3	0.3
MobileNetV3-like	MobileNetV3-Large [21]	75.2	–	219	–	–	–
	FBNet-C [48]	74.9	–	375	11.5	1.8	9
	MnasNet-A3 [44]	76.7	93.3	403	8	–	–
	ProxylessNAS [4]	75.1	92.3	465	–	–	8.3
	OFA [3]	76.4	–	397	16	1.7	51.7
	FBNetV2 [45]†	76.3	92.9	321	11.5	5	25
	AtomNAS [35]	75.9	92.0	367	78	–	–
	Random Search	76.0	92.6	314	–	–	50
	Best Sampled Architectures	76.7	93.1	382	–	–	50
	CTNAS (Ours)	**77.3**	**93.4**	482	1	**0.1**	50.1

in the dataset to compute Kendall's Tau (KTau) [41]. Note that a larger KTau means evaluating architectures more accurately. Following the settings in [52], we obtain the test accuracy by averaging the accuracy of 3 different runs. Following [37], we train the CTNAS model for 10k iterations with a batch size of 1 in the training.

Comparisons with State-of-the-art Methods. We compare our proposed CTNAS with state-of-the-art methods in NAS-Bench-101 search space. From Table 1, the proposed CTNAS achieves higher KTau value (0.751) than other NAS methods. The results show CTNAS evaluates architectures accurately, which is beneficial to the search process. Besides, CTNAS achieves the highest average testing accuracy (93.92%), and the best architecture searched by CTNAS is the top 0.01% in the search space with the testing accuracy of 94.22%. We also compare CTNAS with its variant (*i.e.*, RegressionNAS) that trains a regression-based predictor with L2 loss to predict the absolute performance. CTNAS has a much higher average testing accuracy and a lower variance of the accuracy than RegressionNAS, which demonstrates the stability of our method.

5.2. Experiments on ImageNet

Implementation Details. We apply our CTNAS to a MobileNetV3-like search space [21]. We first train a supernet with the progressive shrinking strategy [3] (cost 50 GPU days). Then, we sample 1000 architectures in the search space as the training data for NAC. Following [3], we compute the validation accuracy on 10k images sampled from the training set of ImageNet. For a fair comparison, we following the mobile setting [29] and restrict the number of multiply-adds (MAdds) to be less than 600M.

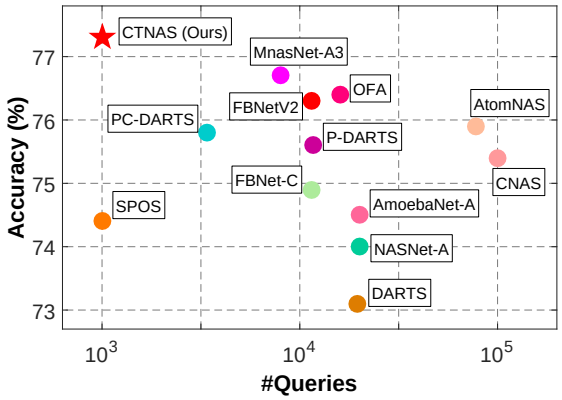


Figure 4. The accuracy *vs.* the number of queries among different methods on ImageNet.

Comparisons with State-of-the-art Methods. We compare the performance of CTNAS with other NAS methods on ImageNet in Table 2. CTNAS achieves 77.3% top-1 accuracy and 93.4% top-5 accuracy, which consistently outperforms existing human-designed architectures and state-of-the-art NAS models searched in different search spaces. Besides, we compare the searched architecture with the best one of 1000 sampled architectures. The searched architecture achieves higher accuracy (77.3% *vs.* 76.7%), which demonstrates the effectiveness of CTNAS. We report the number of queries from supernet for validation accuracy of different methods in Figure 4. Note that querying from supernet takes up most of the search time cost. Thus, the number of queries becomes an important metric to measure search cost [52, 32]. CTNAS has fewer queries (1k) than other

methods but achieves the highest searched performance. The results demonstrate that NAC is more efficient and greatly accelerates the search process (*i.e.*, only 0.1 GPU days).

6. Further Experiments

6.1. Comparisons of Architecture Evaluation Cost

In this experiment, we compare our NAC with other NAS methods in terms of the time cost to rank 100 neural architectures. From Table 3, our NAC has much lower time cost (4.1 ms) than existing NAS methods, such as ReNAS [53] (85.6 ms) and ENAS [37] (2.7 s). The reasons have two aspects: 1) The inputs of NAC have lower dimensions. NAC only takes architecture graphs as inputs while weight sharing methods (*e.g.*, ENAS) need to compute the accuracy on the validation data. Besides, ReNAS uses manually-deigned features of architectures as inputs which have higher dimensions than ours. 2) Our NAC only consists of 3 layers while the models in considered methods often have tens of layers.

Table 3. Time cost of evaluating 100 architectures.

Method	CTNAS	ReNAS	ENAS	Training from Scratch
Time Cost	**4.1 ms**	85.6 ms	2.7 s	5,430 h

6.2. Effect of Baseline Updating Method

To verify the effectiveness of the proposed baseline updating method via curriculum learning, we compare our CTNAS with two variants, namely *Fixed Baseline* and *Randomly Updating*. Fixed Baseline variant finds promising architectures with a fixed baseline architecture in the whole searching process. Random Updating variant updates the baseline with a randomly sampled architecture. From Table 4, our CTNAS outperforms these two variants by a large margin. The reason is that the fixed or randomly sampled baseline may be too weak. In this case, it is hard for the controller to find promising architectures. The experimental results demonstrate the effectiveness of our proposed baseline updating method.

Table 4. Comparisons of different baseline updating methods.

Method	CTNAS	Fixed Baseline	Random Updating
Acc. (%)	**93.92±0.18**	93.39±0.17	93.53±0.39

6.3. Effect of Data Exploration Method

To investigate the effect of the proposed data exploration method, we conduct more experiments in NAS-Bench-101 search space. Let $\mathcal{M}$ and $\mathcal{U}$ be the set of labeled data and data with labels predicted by NAC in a batch, respectively. The proportion of data with predicted labels is $r = |\mathcal{U}|/(|\mathcal{U}| + |\mathcal{M}|)$. As shown in Table 5, compared with that without data exploration ($r = 0$), CTNAS achieves better performance when the proportion r is set to 0.5. Besides, either a low or high proportion would hamper the training of CTNAS, leading to a poor searched performance. Thus, we set the proportion r to 0.5 in the experiments.

Table 5. Comparisons of different proportions (r) of data with predicted labels.

r	0.0	0.3	0.5	0.8
Acc. (%)	93.60±0.22	93.62±0.30	**93.92±0.18**	93.60±0.40

6.4. Effect of the Number of Training Samples

To investigate the effect of the number of training architectures/samples, we perform more experiments with different numbers of training samples (*i.e.*, {20, 50, 100, 423, 2115, 4230}). The results are shown in Figure 5. When the number of samples increases from 20 to 423, the searched architectures achieve better performance. However, the searched architectures yield similar accuracy when the number of training samples is larger than 423. The results show that a small number of training samples are sufficient for our NAC. The reasons are two folds: 1) The proposed data exploration method provides more training samples. 2)Oour NAC has a low requirement for training samples since we can construct $m(m-1)/2$ training pairs from m labeled samples.

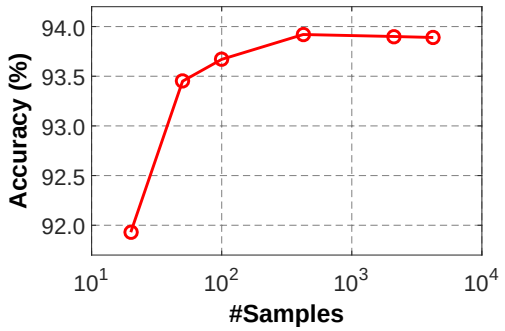


Figure 5. Comparisons of different number of training samples.

7. Conclusion

In this paper, we have proposed a novel Contrastive Neural Architecture Search (CTNAS) method that performs search based on the comparison results of the sampled architecture and a baseline one. To constantly find better architectures, we propose a baseline updating method via curriculum learning to improve the baseline gradually. To provide the comparison results, we devise a Neural Architecture Comparator (NAC) that takes two architectures as inputs and outputs the probability that one is better than the other. Moreover, we propose a data exploration method for NAC to reduce the requirement of training data and improve the generalization ability of NAC to evaluate unseen architectures. The experimental results in three search spaces demonstrate that our CTNAS outperforms the state-of-the-art methods.

Acknowledgements. This work was partially supported by Key-Area Research and Development Program of Guangdong Province (2018B010107001, 2019B010155002, 2019B010155001), National Natural Science Foundation of China (NSFC) 61836003 (key project), National Natural Science Foundation of China (NSFC) 62072190, 2017ZT07X183, Tencent AI Lab Rhino-Bird Focused Research Program (No.JR201902), Fundamental Research Funds for the Central Universities D2191240. Yong Guo is supported by the National Natural Science Foundation of China (Grant No.62072186), the Guangdong Basic and Applied Basic Research Foundation (Grant No.2019B1515130001).

References

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017. 1, 2

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning*, pages 41–48, 2009. 4

[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. 1, 3, 7, 13

[4] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 7

[5] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *The European Conference on Computer Vision*, pages 833–851, 2018. 1

[6] Peihao Chen, Deng Huang, Dongliang He, Xiang Long, Runhao Zeng, Shilei Wen, Mingkui Tan, and Chuang Gan. Rspnet: Relative speed perception for unsupervised video representation learning. In *AAAI Conference on Artificial Intelligence*, 2021. 1

[7] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *The IEEE International Conference on Computer Vision*, pages 1294–1303, 2019. 2, 7

[8] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019. 1, 3, 6, 14

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 13

[10] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 13, 15

[11] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 7036–7045, 2019. 2

[12] Çağlar Gülçehre and Yoshua Bengio. Knowledge matters: Importance of prior information for optimization. *The Journal of Machine Learning Research*, 17(1):226–257, 2016. 4

[13] Yong Guo, Jian Chen, Qing Du, Anton Van Den Hengel, Qinfeng Shi, and Mingkui Tan. Multi-way backpropagation for training compact deep neural networks. *Neural Networks*, 2020. 1

[14] Yong Guo, Yaofo Chen, Yin Zheng, Peilin Zhao, Jian Chen, Junzhou Huang, and Mingkui Tan. Breaking the curse of space explosion: Towards efficient nas with curriculum search. In *International Conference on Machine Learning*, 2020. 7

[15] Yong Guo, Yongsheng Luo, Zhenhao He, Jin Huang, and Jian Chen. Hierarchical neural architecture search for single image super-resolution. *IEEE Signal Processing Letters*, 27:1255–1259, 2020. 2

[16] Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, and Junzhou Huang. NAT: Neural architecture transformer for accurate and compact architectures. In *Advances in Neural Information Processing Systems*, pages 735–747, 2019. 5

[17] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *The European Conference on Computer Vision*, 2020. 3, 6, 13, 14

[18] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 1735–1742, 2006. 2

[19] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 6307–6315, 2017. 15

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1, 13

[21] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *The IEEE International Conference on Computer Vision*, pages 1314–1324, 2019. 6, 7, 13, 14

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[23] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 2261–2269, 2017. 15

[24] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016. 5, 13

[25] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. FractalNet: Ultra-deep neural networks without residuals. In

International Conference on Learning Representations, 2017. 14

[26] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of Uncertainty in Artificial Intelligence*, page 129, 2019. 1, 2

[27] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *The European Conference on Computer Vision*, pages 19–35, 2018. 1, 2, 15

[28] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018. 2

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 1, 2, 6, 7, 13, 14, 15

[30] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. 12

[31] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. SphereFace: Deep hypersphere embedding for face recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 6738–6746, 2017. 1

[32] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *arXiv preprint arXiv:2002.10389*, 2020. 7

[33] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, pages 7827–7838, 2018. 1, 2, 15

[34] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In *The European Conference on Computer Vision*, pages 116–131, 2018. 7

[35] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *International Conference on Learning Representations*, 2020. 7

[36] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pages 807–814, 2010. 5

[37] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pages 4092–4101, 2018. 1, 2, 3, 4, 5, 6, 7, 8, 15

[38] A. J. Piergiovanni, Anelia Angelova, and Michael S. Ryoo. Tiny video networks. *arXiv preprint arXiv:1910.06961*, 2019. 2

[39] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019. 2, 7, 15

[40] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 1, 7

[41] Pranab Kumar Sen. Estimates of the regression coefficient based on Kendall's Tau. *Journal of the American Statistical Association*, 63(324):1379–1389, 1968. 7, 14

[42] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems*, pages 1849–1857, 2016. 2

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 1, 14

[44] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. MnasNet: Platform-aware neural architecture search for mobile. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 2, 7

[45] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 12962–12971, 2020. 7

[46] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. CosFace: Large margin cosine loss for deep face recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 5265–5274, 2018. 1

[47] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. 4

[48] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 2, 6, 7

[49] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. 15

[50] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. 2, 7

[51] Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2019. 2

[52] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114, 2019. 6, 7, 13, 14

[53] Yixing, Yunhe Wang, Kai Han, Shangling Jui, Chunjing Xu, Qi Tian, and Chang Xu. ReNAS: Relativistic evaluation of

neural architecture search. *arXiv preprint arXiv:1910.01523*, 2019. 2, 6, 8, 14

[54] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020. 2

[55] Runhao Zeng, Wenbing Huang, Mingkui Tan, Yu Rong, Peilin Zhao, Junzhou Huang, and Chuang Gan. Graph convolutional networks for temporal action localization. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019. 1

[56] Runhao Zeng, Haoming Xu, Wenbing Huang, Peihao Chen, Mingkui Tan, and Chuang Gan. Dense regression network for video grounding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1

[57] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2018. 15

[58] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 1, 2, 4

[59] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018. 2, 7, 15

Supplementary for "Contrastive Neural Architecture Search with Neural Architecture Comparators"

In the supplementary, we provide a detailed proof of Proposition 1, more experimental results and more details on the CTNAS method. We organize our supplementary as follows.

- In Section A, we provide the detailed proof for Proposition 1.
- In Section B, we describe how to construct the training data for CTNAS.
- In Section C, we show more training details of CTNAS.
- In Section D, we detail the evaluation method for the searched architectures.
- In Section E, we provide more details on the three considered search spaces.
- In Section F, we give more details on the evaluation metric Kendall's Tau.
- In Section G, we give more experimental results on DARTS search space.
- In Section H, we show the visualization results of the searched architectures.

A. Proof of Proposition 1

In this section, we provide the detailed proof of Proposition 1. We first introduce a useful theorem.

Let $\mathcal{X}$ be a set of samples to be ranked. Given $x, x' \in \mathcal{X}$ with the labels $\mathcal{R}_x$ and $\mathcal{R}_{x'}$, a score function $f : \mathcal{X} \times \mathcal{X} \to \mathcal{R}$, the pairwise ranking loss is defined as $\ell_0(x, x', \mathcal{R}_x, \mathcal{R}_{x'}) = \mathbb{1}\{(\mathcal{R}_x - \mathcal{R}_{x'}) f(x, x') \leq 0\}$. The expected true risk is defined as $R_0(f) = \mathbb{E}[\ell_0]$. Since the indicator function is non-differential, directly optimize ℓ_0 is non-trivial. In practice, the pairwise ranking problem aims to learn a ranking function by optimizing a surrogate loss function. Let $\Phi(\cdot)$ be some measurable function. A surrogate loss can be defined as $\ell_\Phi(x, x', \mathcal{R}_x, \mathcal{R}_{x'}) = \Phi(-\text{sgn}(\mathcal{R}_x - \mathcal{R}_{x'}) \cdot f(x, x'))$. The expected surrogate risk is defined as $R_\Phi(f) = \mathbb{E}[\ell_\Phi]$. The following theorem [30] analyzes the statistical consistency of the surrogate loss ℓ_Φ with respect to the true loss ℓ_0.

Theorem 1 *Let $R_0^* = \inf_f R_0(f)$ and $R_\Phi^* = \inf_f R_\Phi(f)$. Then for all functions f, as long as Φ is convex, we have*

$$R_0(f) - R_0^* \leq \Psi^{-1}(R_\Phi(f) - R_\Phi^*), \tag{10}$$

where

$$\Psi(x) = H^-(\frac{1+x}{2}) - H^-(\frac{1-x}{2}),$$
$$H^-(\rho) = \inf_{\alpha : \alpha(2\rho - 1) \leq 0} (\rho\Phi(-\alpha) + (1-\rho)\Phi(\alpha)).$$

The above theorem states that if an algorithm can effectively minimize the right-hand side of the inequality (10) to approach zero, then the left-hand side will also approaches zero. In other words, if $\Phi(\cdot)$ is a convex function, the surrogate loss ℓ_Φ is statistically consistent with the true loss ℓ_0. Based on Theorem 1, we then prove Proposition 1.

Proposition 1 *Let $f : \Omega \times \Omega \to \mathcal{R}$ be some measurable function, $\sigma(\cdot)$ be the sigmoid function, and $\alpha, \alpha' \in \Omega$. The surrogate loss $\mathcal{L}(f; \alpha, \alpha', y) = \mathbb{E}[\ell(\sigma \circ f(\alpha, \alpha'), y)]$ is consistent with $\mathcal{L}_0(f; \alpha, \alpha', \mathcal{R}_\alpha, \mathcal{R}_{\alpha'}) = \mathbb{E}[\mathbb{1}\{(\mathcal{R}_\alpha - \mathcal{R}_{\alpha'}) f(\alpha, \alpha') \leq 0\}]$.*

Proof Given $y = \mathbb{1}\{\mathcal{R}_\alpha - \mathcal{R}_{\alpha'} \geq 0\}$ and $\sigma(x) = \frac{1}{1+e^{-x}}$, we have the following equation,

$$\begin{aligned}
\mathcal{L}(f; \alpha, \alpha', y) =& \mathbb{E}[-y \log \sigma(f(\alpha, \alpha')) - (1-y) \log(1 - \sigma(f(\alpha, \alpha'))] \\
=& \mathbb{E}[-y \log \sigma(f(\alpha, \alpha')) + (1-y)(f(\alpha, \alpha') - \log \sigma(f(\alpha, \alpha')))] \\
=& \mathbb{E}[(1-y) f(\alpha, \alpha') + \log(1 + \exp(-f(\alpha, \alpha')))] \\
=& \mathbb{E}[\log(1 + \exp(-\text{sgn}(\mathcal{R}_\alpha - \mathcal{R}_{\alpha'}) \cdot f(\alpha, \alpha')))] \\
=& \mathbb{E}[\Phi(-\text{sgn}(\mathcal{R}_\alpha - \mathcal{R}_{\alpha'}) \cdot f(\alpha, \alpha'))],
\end{aligned} \tag{11}$$

where $\Phi(x) = \log(1 + \exp(x))$. Since $\Phi(x)$ is a convex function, according to Theorem 1, the surrogate loss $L(f; \alpha, \alpha', y)$ is consistent with $L_0(f; \alpha, \alpha', \mathcal{R}_\alpha, \mathcal{R}_{\alpha'})$.

B. More Details on Data Construction

As mentioned in Section 4, the training of NAC relies on a set of training data. Here, we provide more details on the training data of NAS-Bench-101 search space [52], MobileNetV3-like search space [21] and DARTS search space [29].

Data Construction in NAS-Bench-101 Search Space. To train the proposed CTNAS model, we use the architectures and the corresponding performance as training data in the NAS-Bench-101 search space. Google has released 423k architectures with corresponding performance on CIFAR-10 in the NAS-Bench-101 search space. The performance includes the training accuracy, validation accuracy, test accuracy, training time and the number of trainable parameters. We randomly sample 423 architectures (0.1% of the whole architectures) as the training set and 100 architectures as the validation set. For training the proposed CTNAS, we construct the training architecture pair (α, α') by randomly sampling from the training architecture set. The label of the architecture pair (α, α') is computed from the validation accuracy of architecture α and α'.

Data Construction in MobileNetV3-like Search Space. To obtain the training data pairs for CTNAS in the MobileNetV3-like search space, we train a supernet with a progressive shrinking strategy following the setting in [3]. We then randomly sample 1000 architectures in the search space and apply the weights from the supernet ($1.0\times$) to these architectures. Then we evaluate the architectures on 10000 validation images sampled from the training set of ImageNet to obtain the validation accuracy. The evaluation takes less than 0.1 GPU days. Finally, we compute the label of the training pairs for NAC with the validation accuracy of the architectures.

Data Construction in DARTS Search Space. Training CTNAS requires some architectures with the corresponding performance. To this end, we first train a supernet for 120 epochs with a uniform sampled strategy [17]. Following DARTS [29], we train the supernet on 50% of the official training set of CIFAR-10 and evaluate them for the validation accuracy on the remain 50% using standard data augmentation techniques [20]. The batch size and initial learning rate are set to 96 and 0.01, respectively. Then, we randomly sample 1000 architectures in the search space. The validation accuracy of each sampled architecture is computed by inheriting weights from the supernet. The label of the training pairs for NAC is computed from the average validation accuracy of the last 10 epochs in the supernet training.

C. More Training Details of CTNAS

In this section, we show more training details of CTNAS while searching in the NAS-Bench-101 [52] and MobileNetV3-like search space [21]. We follow the settings in [24] to build our NAC model. Specifically, we first use two GCN layers to extract features of two input architecture. Then, we concatenate the features and send them to a fully-connected (FC) layer to produce the probability of the first architecture being better than the other one. Since the number of nodes of the architecture graph may be less than 7 in NAS-Bench-101 search space, we pad the size of the adjacency matrix to 7 with zero paddings.

In the training, we train the CTNAS model for 10k iterations. We use Adam with a learning rate of 2×10^{-4} and a weight decay of 5×10^{-4} as the optimizer. The training batch size of NAC is set to 256. To explore more training data, we randomly sample 512 architecture pairs every iteration and add the top 256 architecture pairs with the predicted labels according to comparison probability into the training data batch (See Algorithm 3). We update the baseline architecture every 1000 iteration. We add the controller's sample entropy to the reward, which is weighted by 5×10^{-4}.

D. More Details on Architecture Evaluation

In this section, we elucidate the details of evaluation method for the searched architectures in NAS-Bench-101 [52], MobileNetV3-like [21] and DARTS [29] search space.

More Evaluation Details in NAS-Bench-101 search space. NAS-Bench-101 has released the performance of all the architectures in its search space, including the training accuracy, the validation accuracy and the test accuracy. Following the settings in NAS-Bench-101, we report the average test accuracy of the searched architecture over 3 different runs.

More Evaluation Details in MobileNetV3-like search space. We evaluate the architectures searched in MobileNetV3-like search space on ImageNet. Following the setting in [29], the number of multiply-adds (MAdds) in the searched architectures is restricted to be less than 600M. Following the setting in [3], we directly apply the weights form the supernet ($1.0\times$) to the searched architectures, and then evaluate them on the validation set of ImageNet [9] without finetune.

More Evaluation Details in DARTS search space. To evaluate the searched architectures, we train them from scratch on CIFAR-10. We build the final convolution network with 18 learned cells, including 16 normal cells and 2 reduction cells. The two reduction cells are put at the $1/3$ and $2/3$ depth of the network, respectively. The initial number of the channels is set to 44. Following the setting in [29], we train the model for 600 epochs using the batch size of 96. The training images are padded 4 pixels on each side. Then the padded images or their horizontal flips are randomly cropped to the size of 32×32. We use cutout [10] with a length of 16 in the data augmentation. 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 with a minimum of 0.001. Additional enhancements include path dropout [25] of probability of 0.2 and auxiliary towers [43] with a weight of 0.4.

E. More Details on Search Space

In this paper, we consider three search spaces, namely the NAS-Bench-101 search space [52], MobileNetV3-like search space [21] and DARTS search space [29]. We show the details of these search spaces below.

NAS-Bench-101 Search Space. NAS-Bench-101 defines a search space that is restricted to small topology structures, usually called cells. Each cell contains 7 nodes at most, including IN and OUT nodes, which represent the input and output tensors of the cell. The remaining nodes can be selected from 3×3 convolution, 1×1 convolution and 3×3 max pooling. The number of connections in a cell is limited to no more than 9. The whole convolution network is stacked with 3 blocks followed by a global average pooling and a fully-connected layer. Each block is stacked with 3 cells followed by a downsampling layer, where the image height and width are halved via max-pooling and the channel count is doubled. The initial layer of the model is a stem consisting of a 3×3 convolution with 128 output channels.

MobileNetV3-like Search Space. We consider the search space proposed by MobileNetV3. The model is divided into 5 units with gradually reduced feature map size and increased channel number. Each unit consists of 4 layers at most where only the first layer has stride 2 if the feature map size decrease. All the other layers in the units have stride 1. In our experiments, we search for the number of layers in each unit (chosen from $\{2, 3, 4\}$), the kernel size in each layer (chosen from $\{3, 5, 7\}$) and the width expansion ratio in each layer (chosen from $\{3, 4, 6\}$).

DARTS Search Space. We consider a cell-based search space proposed by DARTS [29], which includes two cell types, namely the normal cell and reduction cell. The normal cell keeps the spatial resolution of feature maps. The reduction cell reduces the height and width of feature maps by half and doubles the number of channels. Each cell contains 7 nodes, including 2 input nodes, 4 intermediate nodes and 1 output node. The output node is the concatenation of the 4 intermediate nodes. There are 8 candidate operations between two nodes, including 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.

F. More Details on Kendall's Tau

To evaluate the performance estimators, we use Kendall's Tau (KTau) [41] to compute the ranking correlation between the predicted performance and the performance obtained by training from scratch over a set of architectures. Given n architectures $\{\alpha_i\}_{i=1}^{n}$, there are $\binom{n}{2} = \frac{n(n-1)}{2}$ architectures pairs. The KTau τ can be computed by

$$\tau = \binom{n}{2}^{-1} \sum_{i<j} \big[\text{sgn}(f(\alpha_i) - f(\alpha_j)) \cdot \text{sgn}(\mathcal{R}(\alpha_i, w_{\alpha_i}) - \mathcal{R}(\alpha_j, w_{\alpha_j}))\big], \tag{12}$$

where $\text{sgn}(\cdot)$ is a sign function, $f(\cdot)$ is a performance predictor, $\mathcal{R}(\cdot, w)$ is the performance obtained by training from scratch. The KTau τ is in the range of $[-1, 1]$. The value of τ represents the correlation between the ranking of the predicted performance and the performance obtained by training from scratch. A higher KTau means that the predicted performance ranking is more accurate.

We compare the KTau of different NAS methods in the NAS-Bench-101 search space. The results are shown in the Table A. Our proposed CTNAS achieves higher KTau (0.751) than the considered methods, including ReNAS [53], SPOS [17] and FairNAS [8]. These results mean that the performance ranking predicted by our method is very close to that of training from scratch. In the search process, the higher KTau means a more accurate reward, often resulting in better search performance.

Table A. Comparisons of the KTau in the NAS-Bench-101 search space for different NAS methods.

Method	ReNAS [53]	SPOS [17]	FairNAS [8]	CTNAS (Ours)
KTau	0.634	0.195	-0.232	**0.751**

G. More Results on DARTS Search Space

Implementation Details. We train a supernet for 120 epochs with a uniform sampled strategy [17]. We randomly sample 1000 architectures in the DARTS search space [29], and obtain their validation accuracy by inheriting weights from the

Table B. Comparisons with state-of-the-art models on CIFAR-10. "cutout" indicates evaluating the architecture using cutout [10] data argumentation. "–" means unavailable results.

Architecture	Test Accuracy (%)	#Params. (M)	Search Cost (GPU days)
DenseNet-BC [23]	96.54	25.6	–
PyramidNet-BC [19]	96.69	26.0	–
Random search baseline	96.71 $\pm$ 0.15	3.2	–
NASNet-A + cutout [59]	97.35	3.3	1,800
NASNet-B [59]	96.27	2.6	1,800
NASNet-C [59]	96.41	3.1	1,800
AmoebaNet-A + cutout [39]	96.66 $\pm$ 0.06	3.2	3,150
AmoebaNet-B + cutout [39]	96.63 $\pm$ 0.04	2.8	3,150
SNAS [49]	97.02	2.9	1.5
ENAS + cutout [37]	97.11	4.6	0.5
NAONet [33]	97.02	28.6	200
GHN [57]	97.16 $\pm$ 0.07	5.7	0.8
PNAS + cutout [27]	97.17 $\pm$ 0.07	3.2	–
DARTS + cutout [29]	97.24 $\pm$ 0.09	3.4	4
Best in Sampled Architectures + cutout	97.33 $\pm$ 0.07	3.7	–
CTNAS + cutout (Ours)	**97.41** $\pm$ **0.04**	3.6	**0.3**

supernet to construct the data for NAC training. We train the controller for 10k iteration with a batch size of 1 following the settings in [37]. We add the controller's sample entropy to the reward, which is weighted by 5×10^{-4}.

Comparisons with State-of-the-art Methods. We compare the searched performance of our proposed CTNAS with the state-of-the-art methods in DARTS [29] search space. From Table B, the architecture searched by CTNAS outperforms both manually designed and automatically searched architectures. We also compare the searched architecture with the best one in 1000 sampled architectures. The searched architecture achieves higher average accuracy (97.41% *vs.* 97.33%), which demonstrates the effectiveness of the proposed CTNAS. Moreover, our CTNAS takes 0.2 GPU days to train the supernet and 0.1 GPU day for the search phrase, which is much faster than the considered NAS methods. The reason is that our NAC has a much lower time cost while evaluating architectures (See results in Sec. 6.1 in the paper).

H. Visualization Results of Searched Architectures

We show the visualization results of searched architectures of three consider search space in Figure A, B and C, respectively. In Figure A, we show the best architecture searched by CTNAS in NAS-Bench-101 search space, which is the third best architecture in the whole search space. In Figure B, we show the resulting architecture searched in MobileNetV3-like search space, which achieves 77.3% top-1 accuracy and 93.4% top-5 accuracy on ImageNet. These architectures searched by CTNAS outperform existing human-designed architectures and automatically searched architectures. The visualization results of the normal cell and reduction cell searched by CTNAS in DARTS search space are shown in Figure C, which achieves the average accuracy of 97.41% on CIFAR-10.

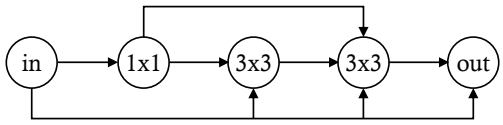


Figure A. The best architecture searched by CTNAS in NAS-Bench-101 search space.

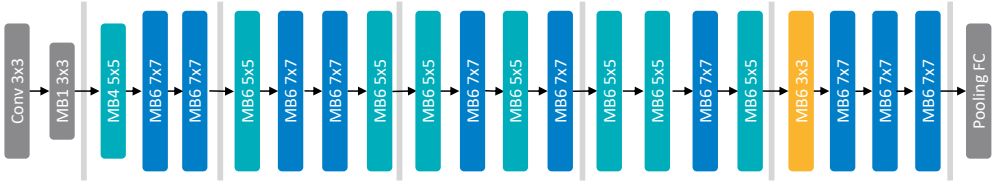
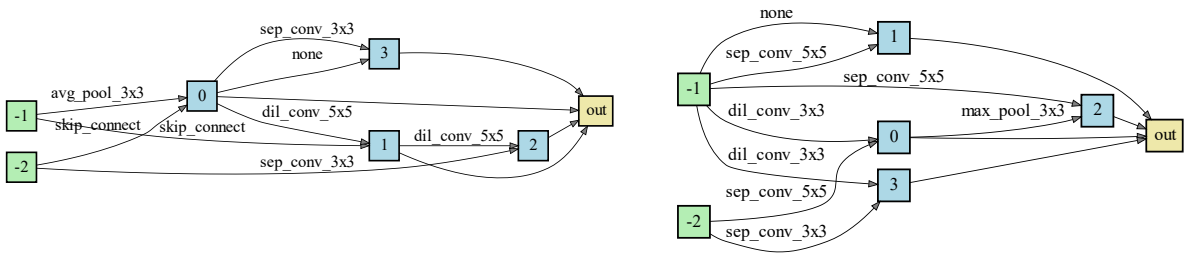


Figure B. The architecture searched by CTNAS in MobileNetV3-like search space.



(a) Normal cell. (b) Reduction cell.

Figure C. The architecture of the convolutional cells found by CTNAS in DARTS search space.