Contrastive Neural Architecture Search with Neural Architecture Comparators

Yaofo Chen¹,²*, Yong Guo¹*, Qi Chen¹, Minli Li¹, Wei Zeng³, Yaowei Wang²†, Mingkui Tan¹,⁴†
¹South China University of Technology, ²Peng Cheng Laboratory, ³Peking University, ⁴Key Laboratory of Big Data and Intelligent Robot, Ministry of Education

{sechenyaofo, guo.yong, sechenqi, seminli.li}@mail.scut.edu.cn,
weizeng@pku.edu.cn, wangyw@pcl.ac.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 [11, 34, 37], face recognition [25, 40], and many other areas [5, 6, 48, 49]. One of the key factors behind the progress lies in the innovation of effective neural architectures, such as ResNet [17] and MobileNet [19]. However, designing effective architectures is often labor-intensive and relies heavily on human expertise. Besides designing architectures manually, Neural Architecture Search (NAS) seeks to design architectures automatically and outperforms the hand-crafted architectures in various tasks [1, 31].

Existing NAS methods seek to find the optimal architecture by maximizing the expectation of the performance of the sampled 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, 31] or a predictor [22, 27]. 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 [21, 24]. Thus, there would be a large performance deviation if we evaluate the architecture only with a single value w.r.t. ab-
solute 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 [21, 47] 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. Specifically, one can evaluate an architecture by feeding in the validation data on a specific task 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 [22, 27]. However, the training of predictor models still requires plenty of architectures with the ground-truth performance as the training architecture data, which are very expensive to obtain in practice. Thus, how to efficiently evaluate architectures with limited architectures with ground-truth performance becomes an important problem.

In this paper, we propose a Contrastive Neural Architecture Search (CTNAS) method that searches by architecture comparisons. To address the first limitation, we devise a Neural Architecture Comparator (NAC) to perform pairwise architecture comparisons. 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 more stable than the absolute performance (See analysis in Section 3.1). To constantly find better architectures, 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 task 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 for NAC.

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/design by state-of-the-art methods.

2. Related Work

Neural Architecture Search. NAS seeks to automatically design neural architectures in some search space. The pioneering work [50] exploits the paradigms of reinforcement learning (RL) to solve it. RL-based methods [1, 9, 31, 38, 50, 51, 13] seek to learn a controller with a policy \( \pi(\alpha; \theta) \) to generate architectures, where \( \alpha \) denotes a sampled architecture, \( \theta \) denotes the parameters of the policy. Specifically, they learn the controller by maximizing the expectation of some performance metric \( R(\alpha, w_\alpha) \),

\[
\max_\theta \mathbb{E}_{\alpha \sim \pi(\alpha; \theta)} R(\alpha, w_\alpha),
\]

where \( w_\alpha = \arg \min_w L(\alpha, w) \) and \( L(\alpha, w) \) is the training loss. Moreover, some studies [23, 32, 33] search for promising architectures using evolutionary algorithms. Different from the above methods, gradient based methods [7, 24, 42, 43] represent architectures by continuous relaxation and optimize by gradient descent. Unlike existing methods that obtain/predict the absolute performance, we seek to conduct comparisons between architectures to guide the search process. Based on the RL algorithm, 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. [16] propose a contrastive loss to solve the dimensionality reduction problem by performing pairwise comparison among different samples. Based on the contrastive loss, Sohn [36] 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 [44]. In this sense, it is possible to solve the ranking problem of NAS by conducting comparisons among architectures.

Comparisons with ReNAS [46]. ReNAS exploits a ranking loss to learn a predictor model that predicts the relative score of architectures. However, ReNAS is essentially different from our method and has several limitations. First, ReNAS searches with the predicted scores of architectures; while our CTNAS proposes a contrastive architecture search scheme that searches by comparing architectures. Second, 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 (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., $\alpha$) to denote architectures. Let $\Omega$ be the search space. For any architecture $\alpha \in \Omega$, let $w_\alpha$ be its optimal model parameters trained on some data set. Let $\mathbb{I}\{\cdot\}$ be an indicator function, where $\mathbb{I}\{A\} = 1$ if $A$ is true and $\mathbb{I}\{A\} = 0$ 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 $w_\alpha$ from a learned supernet $[3, 8, 15, 31]$. 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).

**Algorithm 1** The overall algorithm for CTNAS.

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

1: Randomly sample a set of architectures from $\Omega$ and obtain their accuracy $\{\alpha_i, \mathcal{R}(\alpha_i, w_{\alpha_i})\}_{i=1}^M$ by training a supernet.
2: Construct training data $\mathcal{A} = \{\mathbb{I}\{\alpha_i, \mathbb{I}\{\alpha'_i\} \mathbb{I}\{\mathcal{R}(\alpha_i, w_{\alpha_i}) \geq \mathcal{R}(\alpha'_i, w_{\alpha_i})\}\}_{i=1}^{M(M-1)/2}$ for NAC by traversing all pairwise combinations.
3: Initialize parameters $\theta$ for $\pi(\cdot; \theta)$ and $\varpi$ 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 $\mathcal{C} = \{(\alpha_i, \alpha'_i, y_i)\}_{i=1}^{\lvert \mathcal{C} \rvert}$.
8:  // Train the controller with NAC
9:  Sample $N$ architectures $\{\alpha_j\}_{j=1}^N$ by $\alpha \sim \pi(\cdot; \theta)$.
10:  Update $\theta$ using policy gradient:
11:      $\theta \leftarrow \theta + \eta \frac{\sum_{j=1}^N \nabla_{\theta} \log \pi(\alpha_j; \theta) \mathcal{N}(\alpha_j, \beta; \varpi)}{N}$.
12:  // Explore more data for training NAC
13:  Sample $N$ architectures $\mathcal{S} = \{(\alpha_j)_{j=1}^N \sim \pi(\cdot; \theta)$.
14:  Construct $\mathcal{D}$ with $\mathcal{S}$ by data exploration using Alg. 3.
15:  Let $\mathcal{C} = \mathcal{C} \cup \mathcal{D}$ and $\mathcal{B} = \mathcal{B} \cup \{\beta\}$.
16:  Update the baseline $\beta$ with $\mathcal{B}$ and $\mathcal{S}$ using Alg. 2.
17: end for
may reduce the influence of training fluctuations. Thus, to ensure the optimality, we only need to compute
\[
\Pr[\mathcal{R}(\alpha^*, w_{\alpha^*}) \geq \mathcal{R}({\alpha, w_{\alpha}})].
\] (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 \(\alpha\) being better than \(\alpha'\):
\[
p = \Pr[\mathcal{R}(\alpha, w_{\alpha}) \geq \mathcal{R}(\alpha', w_{\alpha'})] = \text{NAC}(\alpha, \alpha'; \varpi),
\] (3)

where \(\varpi\) is the parameter of NAC. The comparison probability \(p\) is more stable than the absolute performance since it may reduce the negative impact of accuracy deviation. For simplicity, we leave the details of NAC in the following.

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

where \(\theta\) denotes the parameters of the policy. To address the above optimization problem, following [31, 50], we train a controller with policy gradient [41]. Unlike existing reinforcement learning based NAS methods, we adopt the comparison probability \(p=\text{NAC}(\alpha, \beta; \varpi)\) as the reward. Given a specific \(\beta\), 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 \(\beta\). In other words, it may not find the optimal architecture. Moreover, in Problem (4), if \(\beta\) 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 scheme in the following.

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, 10]. Specifically, the difficulty of finding architectures than a gradually improved baseline architecture would increase. As we gradually improve the baseline, our CTNAS is able to constantly find better architectures during the search.

To improve the baseline architecture \(\beta\), we seek to select the best architecture from the 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 impact 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 \(\alpha_i\) with other architectures in \(\mathcal{H}\):
\[
s_i = \frac{1}{|\mathcal{H}| - 1} \sum_{1 \leq j \leq |\mathcal{H}|, i \neq j} \text{NAC}(\alpha_i, \alpha_j; \varpi).
\] (5)

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.
architectures. To guarantee that NAC can handle any architecture, we represent an architecture as a directed acyclic graph (DAG) 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 \( \Omega \), we can represent an architecture \( \alpha \in \Omega \) as a directed acyclic graph (DAG). 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 [20], we represent the graph \( \alpha \) using a data pair \((A_\alpha, X_\alpha)\), where \( A_\alpha \) denotes the adjacency matrix of the graph and \( X_\alpha \) denotes the learnable embeddings of nodes/operations in \( \alpha \).

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 \( \alpha \) and \( \alpha' \) as inputs, the proposed NAC predicts the probability of \( \alpha \) being better than \( \alpha' \) (See Figure 2 (b)). To calculate the comparison probability, we concatenate the features of \( \alpha \) and \( \alpha' \) 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:

\[
p = \text{NAC}(\alpha, \alpha'; \varpi) = \sigma \left( [Z_\alpha; Z_{\alpha'}] W^{FC} \right),
\]

where \( Z_\alpha \) denotes the features extracted from the architecture \( \alpha \), \( W^{FC} \) denotes the weight of the FC layer, \( [Z_\alpha; Z_{\alpha'}] \) refers to the concatenation of the features of \( \alpha \) and \( \alpha' \). Based on the graph data pair \((A_\alpha, X_\alpha)\), we use a two-layer GCN to extract the architecture features \( Z_\alpha \) following [14]:

\[
Z_\alpha = f(X_\alpha, A_\alpha) = A_\alpha \phi \left( A_\alpha X_\alpha W^{(0)} \right) W^{(1)},
\]

where \( W^{(0)} \) and \( W^{(1)} \) denote the weights of two graph convolutional layers, \( \phi \) is the a non-linear activation function (e.g., the Rectified Linear Unit (ReLU) [30]), and \( Z_\alpha \) 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 subnet or training a set of architectures from scratch. Based on the data set, we define the label for any pair \((\alpha, \alpha')\), \( \alpha \neq \alpha' \), sampled from \( \Omega \), by \( y_{i} = \mathbb{I}\{R(\alpha, w_\alpha) - R(\alpha', w_{\alpha'}) \geq 0\} \), and construct the training data \((\alpha, y_{i})\). 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 \( R(\alpha, w_\alpha) \) as \( R_\alpha \). Given a data set \( \{\alpha_i, R_\alpha\}_{i=1}^N \), we seek to find a ranking function \( f : \Omega \times \Omega \to \mathbb{R} \) by optimizing the ranking loss \( \ell_0(\alpha, \alpha', R_\alpha, R_{\alpha'}) = \mathbb{I}\{(R_\alpha - R_{\alpha'}) f(\alpha, \alpha') \leq 0\} \). However, directly optimizing \( \ell_0 \) is non-trivial since the indicator function is non-differential. To address this, we use a binary cross-entropy loss \( \mathcal{L}(f; \alpha, \alpha', y) = \mathbb{E}[\ell(f(\sigma(\alpha, \alpha'), y))] \) as a surrogate for \( \ell_0 \), 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 \( f : \Omega \times \Omega \to \mathbb{R} \) be some measurable function, architectures \( \alpha, \alpha' \in \Omega \). The surrogate loss \( \mathcal{L}(f; \alpha, \alpha', y) = \mathbb{E}[\ell(\sigma \circ f(\alpha, \alpha'), y)] \) is consistent with \( \mathcal{L}_0(f; \alpha, \alpha', R_\alpha, R_{\alpha'}) = \mathbb{E}[\mathbb{I}\{(R_\alpha - R_{\alpha'}) f(\alpha, \alpha') \leq 0\}] \).

Proposition 1 shows that learning NAC is equivalent to optimizing the ranking over architectures. We put the proof in the supplementary.
Table 1. Comparisons with existing methods in NAS-Bench-101 search space. “–” represents unavailable results. “Best Rank” denotes the percentile rank of the best searched architecture among all the architectures in the search space. “#Queries” denotes the number of pairs of architecture and its validation accuracy queried from the NAS-Bench-101 dataset. All methods are run 10 times.

<table>
<thead>
<tr>
<th>Method</th>
<th>KTau</th>
<th>Average Accuracy (%)</th>
<th>Best Accuracy (%)</th>
<th>Best Rank (%)</th>
<th>#Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>–</td>
<td>89.31 ± 3.92</td>
<td>93.46</td>
<td>1.29</td>
<td>423</td>
</tr>
<tr>
<td>DARTS [24]</td>
<td>–</td>
<td>92.21 ± 0.61</td>
<td>93.02</td>
<td>13.47</td>
<td>–</td>
</tr>
<tr>
<td>ENAS [31]</td>
<td>–</td>
<td>91.83 ± 0.42</td>
<td>92.54</td>
<td>22.88</td>
<td>–</td>
</tr>
<tr>
<td>FBNNet [42]</td>
<td>–</td>
<td>92.29 ± 1.25</td>
<td>93.98</td>
<td>0.05</td>
<td>–</td>
</tr>
<tr>
<td>SPOS [15]</td>
<td>0.195</td>
<td>89.85 ± 3.80</td>
<td>93.84</td>
<td>0.07</td>
<td>–</td>
</tr>
<tr>
<td>FairNAS [8]</td>
<td>-0.232</td>
<td>91.10 ± 1.84</td>
<td>93.55</td>
<td>0.77</td>
<td>–</td>
</tr>
<tr>
<td>ReNAS [46]</td>
<td>0.634</td>
<td>93.90 ± 0.21</td>
<td>94.11</td>
<td>0.04</td>
<td>423</td>
</tr>
<tr>
<td>RegressionNAS</td>
<td>0.430</td>
<td>89.51 ± 4.94</td>
<td>93.65</td>
<td>0.40</td>
<td>423</td>
</tr>
<tr>
<td>CTNAS (Ours)</td>
<td>0.751</td>
<td>93.92 ± 0.18</td>
<td>94.22</td>
<td>0.01</td>
<td>423</td>
</tr>
</tbody>
</table>

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 \( \binom{m}{1} = m(m-1)/2 \) training pairs to train NAC while the regression-based methods only have m training samples.

4.2. Data Exploration for Training Samples

Note that learning a good NAC requires a set of labeled data, \( \{ (\alpha_i, \alpha'_i, y_i) \}_{i=1}^{M} \). 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 NAC(·; \( \overline{x} \)), the predicted label of the previously unseen architecture pair can be computed by

\[
y' = 1 \{ \text{NAC}(\alpha, \alpha'; \overline{x}) \geq 0.5 \},
\]

where \( 1 \{ \cdot \} \) refers to an indicator function. Here, \( y' = 1 \) if \( \text{NAC}(\alpha, \alpha'; \overline{x}) \geq 0.5 \) 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:

\[
f_k = | \text{NAC}(\alpha_k, \alpha'_k; \overline{x}) - 0.5 |. \tag{9}
\]

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 [45], MobileNetV3 [18] and DARTS [24] search spaces. We put more details about the search space and the implementation in 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 pairs queried from the NAS-Bench-101 dataset to 423 for all the methods. To measure the rank correlation, we use another 100 architectures in the dataset to compute Kendall’s Tau (KTau) [35]. Note that a larger K Tau means evaluating architectures more accurately. Following the settings in [45], we obtain the test accuracy by averaging the accuracy of 3 different runs. Following [31], 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

\(^1\text{Due to page limit, we put the experimental results in DARTS search space into the supplementary.}\)

\(^2\text{The source code is available at https://github.com/chenyaofo/CTNAS.}\)
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.

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Architecture</th>
<th>Test Accuracy (%)</th>
<th>#MAdds (M)</th>
<th>#Queries (K)</th>
<th>Search Time (GPU days)</th>
<th>Total Time (GPU days)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Top-1</td>
<td>Top-5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MobileNetV2 (1.4×)</td>
<td>74.7</td>
<td>–</td>
<td>585</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>ShuffleNetV2 (2×)</td>
<td>73.7</td>
<td>–</td>
<td>524</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>NASNet</td>
<td>NASNet-A [51]</td>
<td>74.0</td>
<td>91.6</td>
<td>564</td>
<td>20</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>AmoebaNet-A [33]</td>
<td>74.5</td>
<td>92.0</td>
<td>555</td>
<td>20</td>
<td>–</td>
</tr>
<tr>
<td>DARTS</td>
<td>DARTS [24]</td>
<td>73.1</td>
<td>91.0</td>
<td>595</td>
<td>19.5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>P-DARTS [7]</td>
<td>75.6</td>
<td>92.6</td>
<td>577</td>
<td>11.7</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>PC-DARTS [43]</td>
<td>75.8</td>
<td>92.7</td>
<td>597</td>
<td>3.4</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td>CNAS [12]</td>
<td>76.0</td>
<td>92.6</td>
<td>576</td>
<td>100</td>
<td>0.3</td>
</tr>
<tr>
<td>MobileNetV3-like</td>
<td>MobileNetV3-Large</td>
<td>75.7</td>
<td>–</td>
<td>219</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>FBNet-C [42]</td>
<td>74.9</td>
<td>–</td>
<td>375</td>
<td>11.5</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>MnasNet-A3 [38]</td>
<td>76.7</td>
<td>93.3</td>
<td>403</td>
<td>8</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>ProxylessNAS [4]</td>
<td>75.1</td>
<td>92.3</td>
<td>465</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>OFA [3]</td>
<td>76.0</td>
<td>–</td>
<td>230</td>
<td>16</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>FBNetV2 [39]</td>
<td>76.3</td>
<td>92.9</td>
<td>321</td>
<td>11.5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>AtomNAS [29]</td>
<td>75.9</td>
<td>92.0</td>
<td>367</td>
<td>78</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Random Search</td>
<td>76.0</td>
<td>92.6</td>
<td>314</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Best Sampled Architectures</td>
<td>76.7</td>
<td>93.1</td>
<td>382</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>CTNAS (Ours)</td>
<td>77.3</td>
<td>93.4</td>
<td>482</td>
<td>1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

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

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 [18]. 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 [24] and restrict the number of multiply-adds (MAdds) to be less than 600M.

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 [45, 26]. 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 [46] (85.6 ms) and ENAS [31] (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>
<thead>
<tr>
<th>Method</th>
<th>CTNAS</th>
<th>ReNAS</th>
<th>ENAS</th>
<th>Training from Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Cost</td>
<td>4.1 ms</td>
<td>85.6 ms</td>
<td>2.7 s</td>
<td>5,430 h</td>
</tr>
</tbody>
</table>

6.2. Effect of Baseline Updating Scheme

To verify the effectiveness of the proposed baseline updating scheme 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. Randomly sampled variant samples sufficient good baselines in some cases, thus achieves higher performance than the fixed one.

Table 4. Comparisons of different baseline updating schemes.

<table>
<thead>
<tr>
<th>Method</th>
<th>CTNAS</th>
<th>Fixed Baseline</th>
<th>Random Updating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc. (%)</td>
<td>93.92±0.18</td>
<td>93.39±0.17</td>
<td>93.53±0.39</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>$r$</th>
<th>0.0</th>
<th>0.3</th>
<th>0.5</th>
<th>0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc. (%)</td>
<td>93.60±0.22</td>
<td>93.62±0.30</td>
<td>93.92±0.18</td>
<td>93.60±0.40</td>
</tr>
</tbody>
</table>

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}). From Figure 5, when the number of samples increases from 20 to 423, our CTNAS achieve better performance. However, our CTNAS yield similar accuracy when the number of 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) Our NAC has a low requirement for training samples since we can construct $m(m-1)/2$ 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 architectures and a baseline one. To constantly find better architectures, we propose a baseline updating scheme 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 (2019B010155002), CCF-Baidu Open Fund (CCF-BAIDU OF2020022), National Natural Science Foundation of China (NSFC) 61836003 (key project), Program for Guangdong Introducing Innovative and Enterpreneurial Teams 2017ZT07X183, Fundamental Research Funds for the Central Universities D2191240. Yong Guo is supported by National Natural Science Foundation of China (Grant No. 62072186), Guangdong Basic and Applied Basic Research Foundation (Grant No. 2019B1515130001).
References


