Contrastive Neural Architecture Search with Neural Architecture Comparators

Yaofo Chen*, Yong Guo*, Qi Chen, Minli Li, Wei Zeng, Yaowei Wang†, Mingkui Tan†

CVPR
VIRTUAL JUNE 19-25

Background and Motivation

Limitations of existing neural architecture search (NAS) methods:

- It is non-trivial to obtain **stable and accurate absolute performance** for all the candidate architectures.
- It is **time-consuming to obtain the absolute performance** from the supernet during the search process.

Contributions

- 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**.

Contrastive Neural Architecture Search

Unlike existing methods that rely on the absolute performance, we seek to obtain the ranking of candidate architectures with a comparison mapping, called **Neural Architecture Comparator (NAC)**:

$$p = \Pr[\mathcal{R}(\alpha, w_\alpha) \geq \mathcal{R}(\alpha', w_{\alpha'})] = \mathrm{NAC}(\alpha, \alpha'; \varpi), \tag{1}$$

where ϖ is the parameter of NAC. Our NAC compares two architectures α and α' and output **the probability of α being better than α'**.

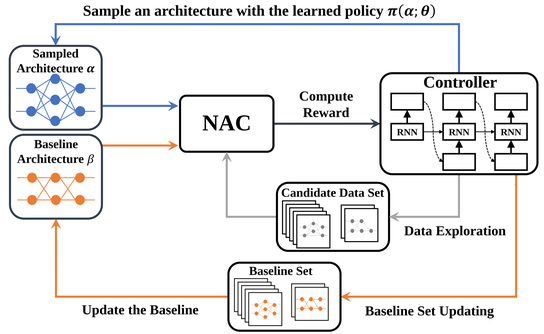
Based on Eqn. (1), we propose a **Contrastive Neural Architecture Search (CTNAS)** method that exploits **the comparison probability** predicted by NAC as the **reward signal**. Formally, given a baseline architecture $\beta \in \Omega$, we learn a policy $\pi(\alpha; \theta)$ by solving the following optimization problem:

$$\max_{\theta} \ \mathbb{E}_{\alpha \sim \pi(\alpha;\theta)} \Pr[\mathcal{R}(\alpha, w_\alpha) \geq \mathcal{R}(\beta, w_\beta)], \tag{2}$$

where θ denotes the parameters of the policy. To address the above optimization problem, we train a controller with **policy gradient** method.

Overview of CTNAS

Our NAC first takes the sampled architecture and the baseline one as inputs, and outputs **the comparison probability of them**. Then, the controller adopts **the probability as the reward**. During the training, we **improve the baseline** with the architectures sampled from the controller.



Training Method

Algorithm 1: The overall algorithm for CTNAS.

Require: Learning rate η, 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 pairwaise 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 the data exploration method.
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**

Baseline Updating via Curriculum Learning

In Problem (2), if β is **too weak or strong**, the 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**.

Algorithm 2: Baseline updating via curriculum learning.

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

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

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

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

Neural Architecture Comparator

Given two architectures α and α' as inputs, our NAC predicts the **probability of α being better than α'**. To this end, we concatenate the features of α and α' and send them to a fully-connected (FC) layer:

$$p = \mathrm{NAC}(\alpha, \alpha'; \varpi) = \sigma\left([\mathbf{Z}_\alpha; \mathbf{Z}_{\alpha'}]\mathbf{W}^{FC}\right), \tag{3}$$

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

$$\mathbf{Z}_\alpha = f(\mathbf{X}_\alpha, \mathbf{A}_\alpha) = \mathbf{A}_\alpha \phi\left(\mathbf{A}_\alpha \mathbf{X}_\alpha \mathbf{W}^{(0)}\right) \mathbf{W}^{(1)}, \tag{4}$$

where $\mathbf{W}^{(0)}$ and $\mathbf{W}^{(1)}$ denote the weights of GCN, ϕ is the a non-linear activation function (*e.g.*, ReLU), and $\mathbf{Z}_\alpha$ refers to the extracted features.

Data Exploration for Training NAC

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 **limited labeled data** in practice.

To address this issue, we propose a **data exploration** method that adopts unlabeled architecture pairs to assist the training. Specifically, we take **the class with maximum probability predicted by NAC** as the label.

Comparisons with SoTA Methods

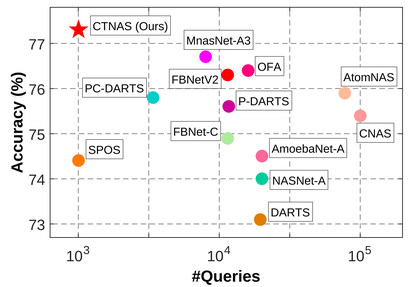
- **Comparisons with state-of-the-art methods on NAS-Bench-101**

Method	KTau	Average Accuracy (%)	Best Accuracy (%)	Best Rank (%)	#Queries
Random	–	89.31 ± 3.92	93.46	1.29	423
DARTS	–	92.21 ± 0.61	93.02	13.47	–
ENAS	–	91.83 ± 0.42	92.54	22.88	–
FBNet	–	92.29 ± 1.25	93.98	0.05	–
SPOS	0.195	89.85 ± 3.80	93.84	0.07	–
FairNAS	-0.232	91.10 ± 1.84	93.55	0.77	–
ReNAS	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

- **Comparisons with state-of-the-art methods on ImageNet**

Search Space	Architecture	Test Accuracy (%) Top-1	Top-5	#MAdds (M)	#Queries (K)	Search Time (GPU days)	Total Time (GPU days)
	MobileNetV2 (1.4×)	74.7	–	585	–	–	–
	ShuffleNetV2 (2×)	73.7	–	524	–	–	–
NASNet	NASNet-A	74.0	91.6	564	20	–	1800
	AmoebaNet-A	74.5	92.0	555	20	–	3150
DARTS	DARTS	73.1	91.0	595	19.5	4	4
	P-DARTS	75.6	92.6	577	11.7	0.3	0.3
	PC-DARTS	75.8	92.7	597	3.4	3.8	3.8
	CNAS	75.4	92.6	576	100	0.3	0.3
MobileNet	MobileNetV3-Large	75.2	–	219	–	–	–
	FBNet-C	74.9	–	375	11.5	1.8	9
	MnasNet-A	76.7	93.3	403	8	–	–
	ProxylessNAS	75.1	92.3	465	–	–	8.3
	OFA [?]	76.4	–	397	16	1.7	51.7
	FBNetV2	76.3	92.9	321	11.5	5	25
	AtomNAS	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

- **The accuracy *vs.* #queries among different methods on ImageNet**



Contact Information and Code

- Email: mingkuitan@scut.edu.cn
- Code: https://github.com/chenyaofo/CTNAS