NAT: Neural Architecture Transformer for Accurate and Compact Architectures

Yong Guo*, Yin Zheng*, Mingkui Tan*†, Qi Chen, Jian Chen†, Peilin Zhao, Junzhou Huang

Neural Information Processing Systems Foundation

Background and Motivation

Limitations of existing architecture design methods:

- **Hand-crafted architecture design methods**: rely on **substantial human expertise** and **cannot fully explore the whole architecture design space**.
- **Neural architecture search (NAS) methods**: often produce **subopimal architectures** due to the extremely large search space.

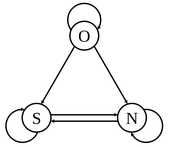
Both hand-crafted architectures and NAS based architectures may contain **non-significant** or **redundant operations**.

Contributions

- We propose a novel **Neural Architecture Transformer (NAT)** to optimize any arbitrary architecture for better performance without extra computational cost.
- We cast the architecture optimization problem into a **Markov decision process (MDP)** and employ **graph convolution network (GCN)** to learn the optimal policy on architecture optimization.
- Extensive experiments show the effectiveness of NAT on **both hand-crafted and NAS-based architectures**.

Problem Definition

- **Operations can be categorized into** $\{S, N, O\}$.
  1. S denotes skip connection
  2. N denotes null connection
  3. O denotes other connections



- **The computational cost follows:**

$$c(O) > c(S) > c(N)$$

- We only allow the transitions: $O \to S, O \to N, S \to N, S \to N$.

Goal: Transform any arbitrary architecture for **better performance and less computational cost**.

Solution: Replace redundant operations with **more efficient ones, such as S and N**.

Optimization for Single Architecture

Given a specific architecture $\widehat{\beta}$, we seek to find an optimal architecture α. The optimization problem can be formulated as:

$$\max_{\alpha} \; R\left(\alpha|\widehat{\beta}\right), \text{ s.t. } \; c(\alpha) \leq \kappa. \tag{1}$$

- $R(\alpha|\beta) = R(\alpha, w_\alpha) - R(\widehat{\beta}, w_{\widehat{\beta}})$ denotes the **performance difference** between the optimized architectures α and the given architectures $\widehat{\beta}$. w_α and $w_{\widehat{\beta}}$ are the parameters of α and $\widehat{\beta}$, respectively.
- $c(\cdot)$ **measures the computation cost of an architecture.**
- κ is an **upper bound of the cost**.

Optimization for Arbitrary Architecture

To optimize arbitrary architecture, we solve the following problem:

$$\max_{\theta} \; \mathbb{E}_{\beta\sim p(\cdot)}\left[\mathbb{E}_{\alpha\sim\pi(\cdot|\beta;\theta)}\, R\left(\alpha|\beta\right)\right], \text{ s.t. } \; c(\alpha) \leq \kappa, \; \alpha \sim \pi(\cdot|\beta;\theta). \tag{2}$$

We further cast the problem into an **Markov Decision Process**.

- An **architecture** is defined as a **state**.
- A **transformation mapping** $\beta \to \alpha$ is defined as an **action**.
- The **accuracy improvement** on validation set is regraded as **reward**.

Policy Learning by GCN

To better exploit the **adjacency information** of architecture graph, we use a two-layer **graph convolution network** to build the controller:

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \text{Softmax}\left(\mathbf{A}\sigma\left(\mathbf{A}\mathbf{X}\mathbf{W}^{(0)}\right)\mathbf{W}^{(1)}\mathbf{W}^{\text{FC}}\right). \tag{3}$$

- $\mathbf{A}$: **adjacency matrix** of the architecture graph.
- $\mathbf{X}$: **attributes of the nodes** in the graph.
- $\mathbf{W}^{(0)}$ and $\mathbf{W}^{(1)}$: **weights** of graph convolution layers.
- $\mathbf{W}^{\text{FC}}$: **weight** of the fully-connected layer.
- σ: **non-linear activation function**.
- $\mathbf{Z}$: probability distribution of different operations, *i.e.*, **the learned policy** $\pi(\cdot|\beta;\theta)$.

Training and Inference Method

- **Training method for NAT**

Algorithm 1 Training method for Neural Architecture Transformer.

Initiate w and θ.
while not convergent **do**
  for each iteration on training data **do**
    Sample $\beta_i \sim p(\cdot)$ to construct a batch $\{\beta_i\}_{i=1}^m$.
    Update the model parameters w by descending the gradient.
  end for
  for each iteration on validation data **do**
    Sample $\beta_i \sim p(\cdot)$ to construct a batch $\{\beta_i\}_{i=1}^m$.
    Obtain $\{\alpha_j\}_{j=1}^n$ according to the policy learned by GCN.
    Update the parameters θ by descending the gradient.
  end for
end while

- **Inferring the optimized architectures**
  1. Sample candidate architectures from the learned policy $\pi(\cdot|\beta;\theta)$.
  2. Select the architectures with the highest validation accuracy.

Results on Different Architectures

- **Results on hand-crafted architectures (comparisons on ImageNet)**

Model	Method	#Params (M)	#MAdds (M)	Acc. (%) Top-1	Acc. (%) Top-5
VGG16	/	138.4	15620	71.6	90.4
	NAO	147.7	18896	72.9	91.3
	NAT	138.4	15620	**74.3**	**92.0**
ResNet18	/	11.7	1580	69.8	89.1
	NAO	17.9	2246	70.8	89.7
	NAT	11.7	1580	**71.1**	**90.0**
ResNet50	/	25.6	3530	76.2	92.9
	NAO	34.8	4505	77.4	93.2
	NAT	25.6	3530	**77.7**	**93.5**
MobileNetV2	/	3.4	300	72.0	90.3
	NAO	4.5	513	72.2	90.6
	NAT	3.4	300	**72.5**	**91.0**

- **Results on NAS based architectures (comparisons on ImageNet)**

Model	Method	#Params (M)	#MAdds (M)	Acc. (%) Top-1	Acc. (%) Top-5
AmoebaNet [34]	/	5.1	555	74.5	92.0
PNAS [28]		5.1	588	74.2	91.9
SNAS [48]		4.3	522	72.7	90.8
GHN [52]		6.1	569	73.0	91.3
ENAS [33]	/	5.6	679	73.8	91.7
	NAO	5.5	656	73.7	91.7
	NAT	5.6	679	**73.9**	**91.8**
DARTS [29]	/	5.9	595	73.1	91.0
	NAO	6.1	627	73.3	91.1
	NAT	3.9	515	**74.4**	**92.2**
NAONet [31]	/	11.35	1360	74.3	91.8
	NAO	11.83	1417	74.5	92.0
	NAT	8.36	1025	**74.8**	**92.3**

Visualization of Architectures

- **Architecture optimization results on hand-crafted architectures**

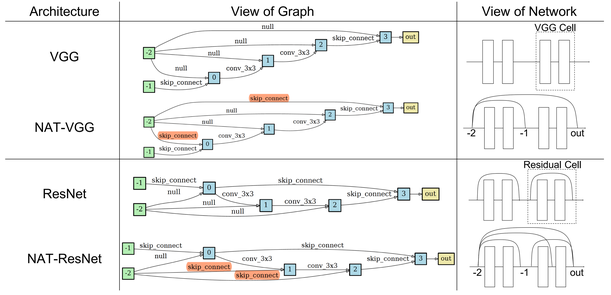


Figure 1: Visualization of some optimized hand-crafted architectures

- **Architecture optimization results on NAS based architectures**

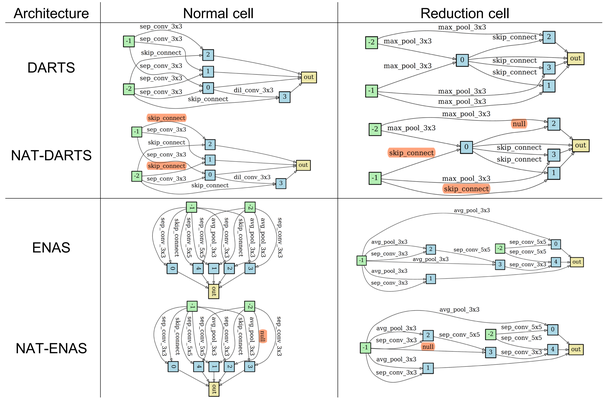


Figure 2: Visualization of some optimized NAS-based architectures

Contact Information and Code

- Email: mingkuitan@scut.edu.cn
- Code: https://github.com/guoyongcs/NAT