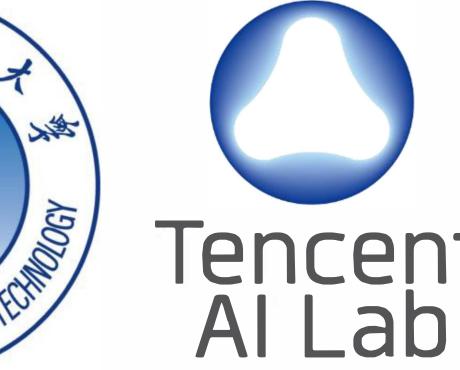


# NAT: Neural Architecture Transformer for Accurate and Compact Architectures

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





#### BACKGROUND AND MOTIVATION

Neural architectures can be divided into two categories, namely handcrafted and NAS based architectures.

#### Limitations of existing architecture design methods:

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

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

## CONTRIBUTIONS

- We propose a novel Neural Architecture Transformer (NAT) method to optimize any arbitrary architecture for better performance without extra computational cost.
- We cast the architecture optimization process 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 handcrafted and NAS-based architectures.

#### PROBLEM DEFINITION

Our goal: Transforming any architecture for better performance and less computational cost.

One solution: Replacing the redundant operations with the more efficient ones.

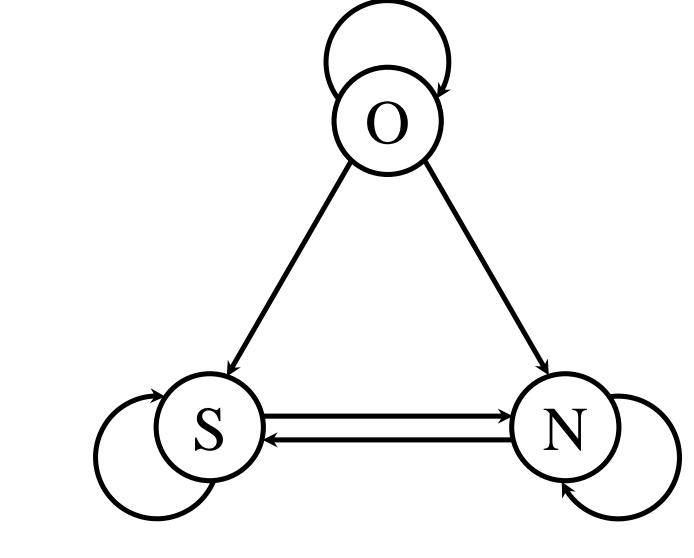


Figure 1: Operation transformation.

- We divide the operations into three categories  $\{S, N, O\}$ . S denotes skip connection, N denotes null connection, O denotes the operations other than skip or null connection.
- We have c(O) > c(S) > c(N), where  $c(\cdot)$  is a function to evaluate the computational cost.
- We constrain the possible transitions among S, N, and O in Figure 1 to reduce the computational cost.

#### MARKOV DECISION PROCESS FOR NAT

Let  $R(\alpha, w)$  be the performance measure for architecture  $\alpha$  with parameters w, e.g., the validation accuracy on validation data set. Then the architure optimization problem can be formulated as:

$$\max_{\theta} \mathbb{E}_{\beta \sim p(\cdot)} \left[ \mathbb{E}_{\alpha \sim \pi(\cdot|\beta;\theta)} R(\alpha|\beta) \right], \text{ s.t. } c(\alpha) \leq \kappa, \ \alpha \sim \pi(\cdot|\beta;\theta), \quad (1)$$

- $R(\alpha|\beta) = R(\alpha, w_{\alpha}) R(\beta, w_{\beta})$  denote the performance difference between the optimized architectures  $\alpha$  and the given architectures  $\beta$ .  $w_{\alpha}$  and  $w_{\beta}$  are the parameters of  $\alpha$  and  $\beta$ , respectively.
- $\mathbb{E}_{\beta \sim p(\cdot)} \left[ \mathbb{E}_{\alpha \sim \pi(\cdot | \beta; \theta)} R(\alpha | \beta) \right]$  indicates the expectation of  $R(\alpha | \beta)$ over the distribution of the given architectures  $\beta \sim p(\cdot)$  and the distribution of the optimized architectures  $\alpha \sim \pi(\cdot|\beta;\theta)$ .
- and  $\kappa$  is an upper bound of the cost.

To solve problem (1), we reformulate the constrained optimization problem into a Markov decision process (MDP), where

- An architecture is defined as a state.
- A transformation mapping  $\beta \to \alpha$  is defined as an action.
- The validation accuracy on validation set is regraded as reward.
- The policy  $\pi(\cdot|\beta;\theta)$  parameterized by  $\theta$  is a probability distribution of the action to transform  $\beta$  into the improved architecture  $\alpha$ .

# POLICY LEARNING BY GCN

To better exploit the adjacency information of architecture graph, we use a two-layer GCN and build the controller:

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \text{Softmax}\left(\mathbf{A}\sigma\left(\mathbf{A}\mathbf{X}\mathbf{W}^{(0)}\right)\mathbf{W}^{(1)}\mathbf{W}^{\text{FC}}\right).$$
 (2)

- A denotes the adjacency matrix of the architecture graph.
- X denotes the attributes of the nodes together with their two input edges in the graph.
- $W^{(0)}$  and  $W^{(1)}$  denote the weights of graph convolution layers.
- ullet W<sup>FC</sup> denotes the **weight** of the fully-connected layer,  $\sigma$  is a **non**linear activation function.
- $\bullet$  **Z** refers to 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 (NAT).

**Require:** The number of sampled input architectures in an iteration m, the number of sampled optimized architectures for each input architecture n, learning rate  $\eta$ , regularizer parameter  $\lambda$ , input architecture distribution  $p(\cdot)$ , shared model parameters w, transformer parameters  $\theta$ .

- : Initiate w and  $\theta$ . . while not convergent do
- for each iteration on training data do
- // Fix  $\theta$  and update w.
- Sample  $\beta_i \sim p(\cdot)$  to construct a batch  $\{\beta_i\}_{i=1}^m$ .
- Update the model parameters w by descending the gradient:  $w \leftarrow w \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_w \mathcal{L}(\beta_i, w)$ .
- for each iteration on validation data do // Fix w and update  $\theta$ .
- 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  $\theta$  by descending the gradient:  $\theta \leftarrow \theta \eta \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \left[ \nabla_{\theta} \log \pi(\alpha_{j} | \beta_{i}; \theta) \left( R(\alpha_{j}, w) R(\beta_{i}, w) \right) + \lambda \nabla_{\theta} H(\pi(\cdot | \beta_{i}; \theta)) \right].$
- end for
- 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

• Architecture optimization results on hand-crafted architectures

Table 1: Performance comparison of the optimized architectures obtained by different methods based on hand-crafted architectures. "/" denotes the original models that are not changed by architecture optimization methods.

		CIFAR-10					ImageNet			
Model	Method	#Params (M)	#MAdds (M)	Acc. (%)	Model	Method	#Params (M)	#MAdds (M)	Acc. (%)	
1120001									Top-1	Top-5
VGG16	/	15.2	313	93.56	VGG16	/	138.4	15620	71.6	90.4
	NAO[31]	19.5	548	95.72		NAO [31]	147.7	18896	72.9	91.3
	NAT	15.2	313	96.04		NAT	138.4	15620	<b>74.3</b>	92.0
ResNet20	/	0.3	41	91.37	ResNet18	/	11.7	1580	69.8	89.1
	NAO [31]	0.4	61	92.44		NAO [31]	17.9	2246	70.8	89.7
	NAT	0.3	41	92.95		NAT	11.7	1580	<b>71.1</b>	90.0
ResNet56	/	0.9	127	93.21	ResNet50	/	25.6	3530	76.2	92.9
	NAO [31]	1.3	199	95.27		NAO [31]	34.8	4505	77.4	93.2
	NAT	0.9	127	95.40		NAT	25.6	3530	77.7	93.5
MobileNetV2	/	2.3	91	94.47	MobileNetV2	/	3.4	300	72.0	90.3
	NAO [31]	2.9	131	94.75		NAO [31]	4.5	513	72.2	90.6
	NAT	2.3	91	95.17		NAT	3.4	300	72.5	91.0
					1					

#### • Architecture optimization results on NAS based architectures

Table 2: Comparison of the optimized architectures obtained by different methods based on NAS based architectures. "-" denotes that the results are not reported. "/" denotes the original models that are not changed by architecture optimization methods. † denotes the models trained with cutout.

		CIFAR-10					ImageNet			
Model	Method	#Params (M)	#MAdds (M)	Acc. (%)	Model	Method	#Params (M)	#MAdds (M)	Acc. Top-1	. (%) Top-5
AmoebaNet <sup>†</sup> [34]		3.2	-	96.73	AmoebaNet [34]		5.1	555	74.5	92.0
PNAS <sup>†</sup> [28]	1	3.2	-	96.67	PNAS [28]	/	5.1	588	74.2	91.9
SNAS <sup>†</sup> [48]	1	2.9	-	97.08	SNAS [48]	/	4.3	522	72.7	90.8
$GHN^{\dagger}$ [52]		5.7	-	97.22	GHN [52]		6.1	569	73.0	91.3
	/	4.6	804	97.11	ENAS [33]	/	5.6	679	73.8	91.7
ENAS <sup>†</sup> [33]	NAO [31] NAT	4.5	763	97.05		NAO [31]	5.5	656	73.7	91.7
		4.6	804	97.24		NAT	5.6	679	<b>73.9</b>	91.8
	/	3.3	533	97.06		/	5.9	595	73.1	91.0
DARTS <sup>†</sup> [29]	NAO [31]	3.5	577	97.09	<b>DARTS</b> [29]	NAO [31]	6.1	627	73.3	91.1
	NAT	3.0	483	97.28		NAT	3.9	515	<b>74.4</b>	92.2
	/	128	66016	97.89		/	11.35	1360	74.3	91.8
NAONet <sup>†</sup> [31]	NAO [31]	143	73705	97.91	NAONet [31]	NAO [31]	11.83	1417	74.5	92.0
	NAT	113	58326	98.01		NAT	8.36	1025	<b>74.8</b>	92.3

## VISUALIZATION OF ARCHITECTURES

• Architecture optimization results on hand-crafted architectures

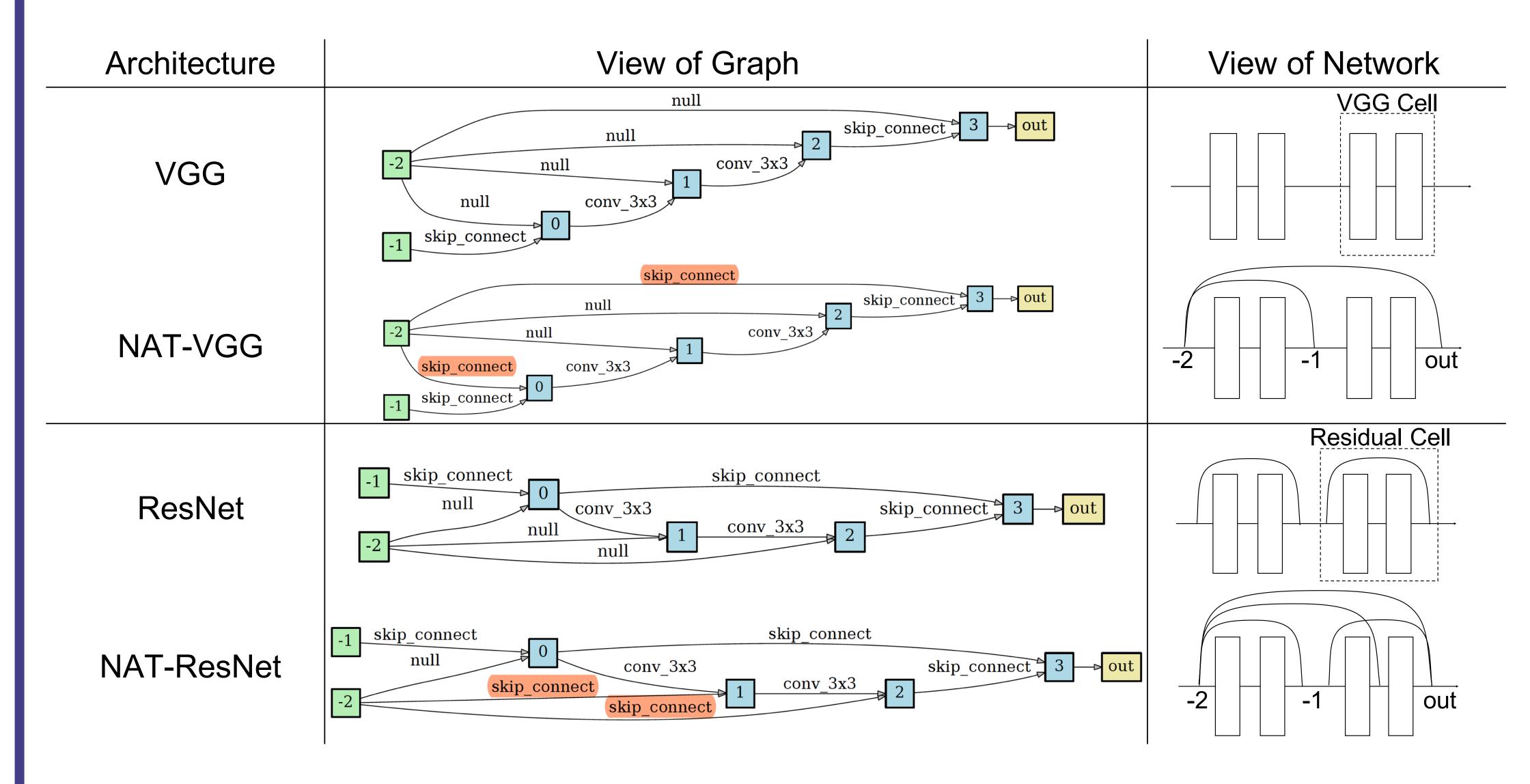


Figure 2: Visualization of some optimized hand-crafted architectures

• Architecture optimization results on NAS based architectures

Architecture	Normal cell	Reduction cell
DARTS	sep_conv_3x3 skip_connect sep_conv_3x3 sep_conv_3x3 sep_conv_3x3 sep_conv_3x3 sep_conv_3x3 sep_conv_3x3 sep_conv_3x3 sep_conv_3x3 sep_conv_3x3 skip_connect 3	max_pool_3x3  -2 max_pool_3x3  skip_connect 2  max_pool_3x3  out  skip_connect 3  max_pool_3x3  max_pool_3x3
NAT-DARTS	skip_connect  sep_conv_3x3 skip_connect  sep_conv_3x3 skip_connect  out  sep_conv_3x3 skip_connect  3	max_pool_3x3  -2 max_pool_3x3  skip_connect  max_pool_3x3  -1 max_pool_3x3  skip_connect  skip_connect
ENAS	sep_conv_3x3 avg_pool_3x3 -2 avg_pool_3x3 sep_conv_3x3 3 avg_pool_3x3 1 out sep_conv_5x5 4 -1 sep_conv_5x5 4 -1 sep_conv_5x5 4 -1 sep_conv_5x5 4	avg_pool_3x3  avg_pool_3x3  sep_conv_5x5  2  sep_conv_5x5  2  sep_conv_5x5  3  sep_conv_3x3  4  out  avg_pool_3x3  sep_conv_3x3  1
NAT-ENAS	sep_conv_3x3 -2 avg_pool_3x3 3 skip_connect sep_conv_3x3 2 -1 sep_conv_5x5 4 -1 sep_conv_5x5 4 sep_conv_3x3 1 -2 avg_pool_3x3 3 -2 avg_pool_3x3 1 -1 sep_conv_5x5 4 -1 sep_conv_5x5 4	avg_pool_3x3  avg_pool_3x3  sep_conv_5x5  out  avg_pool_3x3  sep_conv_5x5  out  avg_pool_3x3  sep_conv_3x3  1  sep_conv_3x3  1

Figure 3: Visualization of some optimized NAS-based architectures

# CONTACT INFORMATION AND CODE

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

