NAT: Neural Architecture Transformer for Accurate and Compact Architectures

Yong Guo[1*], Yin Zheng[2*], Mingkui Tan[1*†], Qi Chen[1],
Jian Chen[1†], Peilin Zhao[3], Junzhou Huang[3,4]

[1]South China University of Technology
[2]Weixin Group, Tencent [3]Tencent AI Lab
[4]University of Texas at Arlington

Published in NeurIPS 2019

Contents

Contents

Background

Deep neural networks have achieved great success in many computer vision tasks, such as image classification, face recognition, object detection, etc.

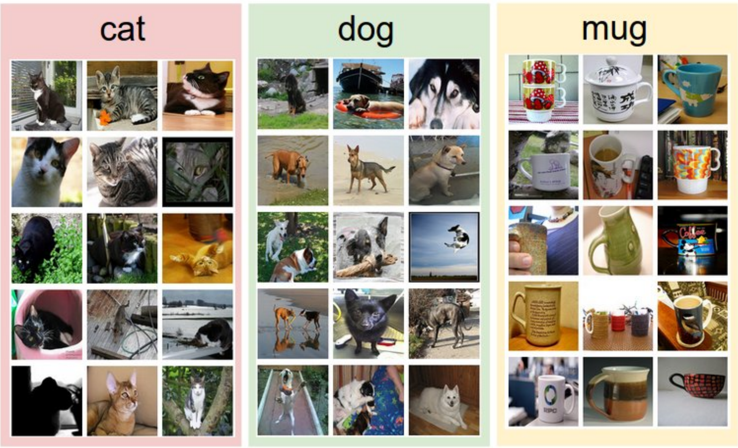
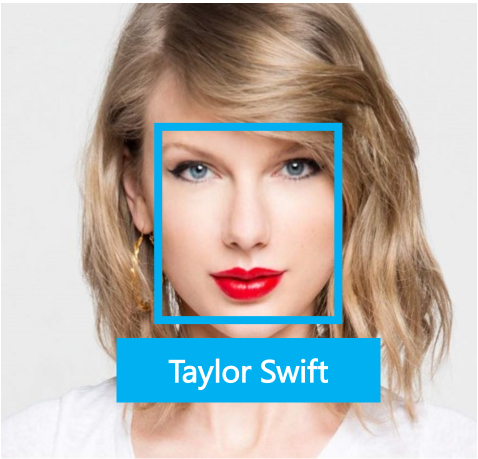
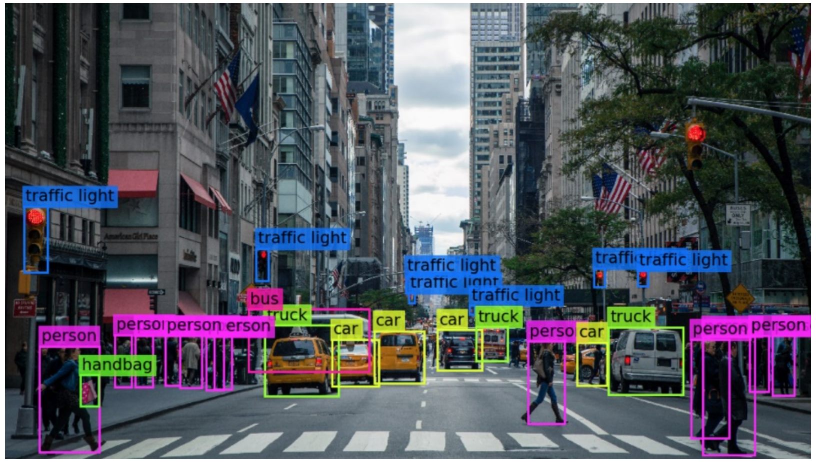


Image Classification



Face Recognition



Object Detection

Figure: Applications of deep neural networks.

Neural Architecture Design

- Neural architecture design is one of the key factors behind the success of deep neural networks.

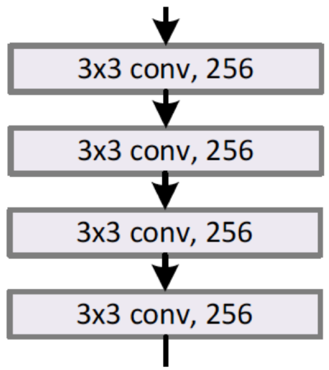
- Existing architectures can be divided into two categories:

  1. Hand-crafted architectures

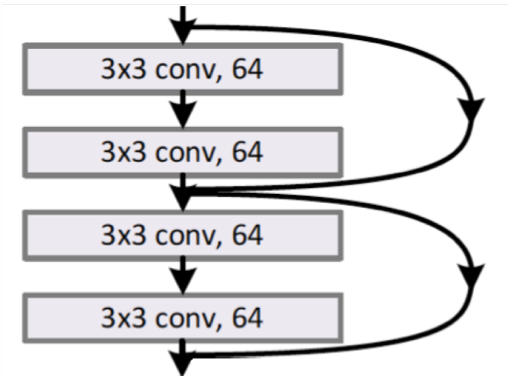
  2. Automatically searched architectures

Hand-crafted Architectures

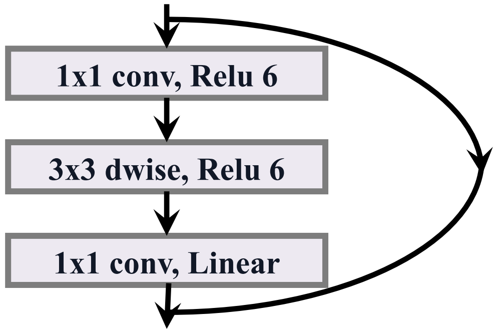
Several widely used hand-crafted architectures:



VGG



ResNet



MobileNetV2

Limitations of hand-crafted architecture design process

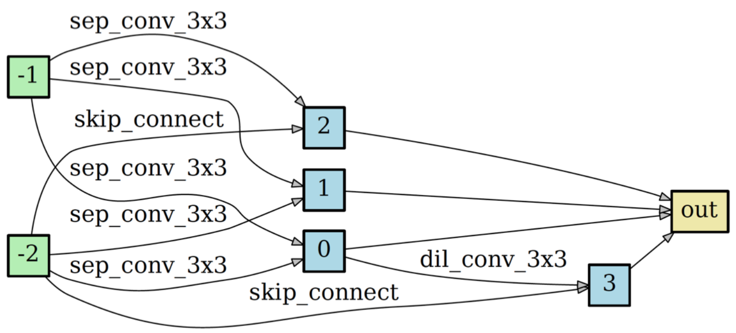
- Hand-crafted methods rely on substantial human expertise.
- Hand-crafted methods cannot fully explore the whole architecture space.

Automatically Searched Architectures

- There is a growing interest to replace the manual process of architecture design by Neural Architecture Search (NAS).

Graph Representation of Architectures: an architecture can be represented by a directed acyclic graph (DAG).

- Node: feature maps of a specific layer
- Edge: a computational operation, e.g., convolution



DARTS normal cell

Limitations of NAS methods

- Search space is extremely large, *e.g.*, billions of candidate architectures.
- NAS methods may find suboptimal architectures with limited performance.

Architecture Optimization

Since both the hand-crafted and NAS based architectures are not optimal, can we optimize architectures to obtain the better ones?

- One can design architecture optimization methods to optimize existing architectures for better performance.

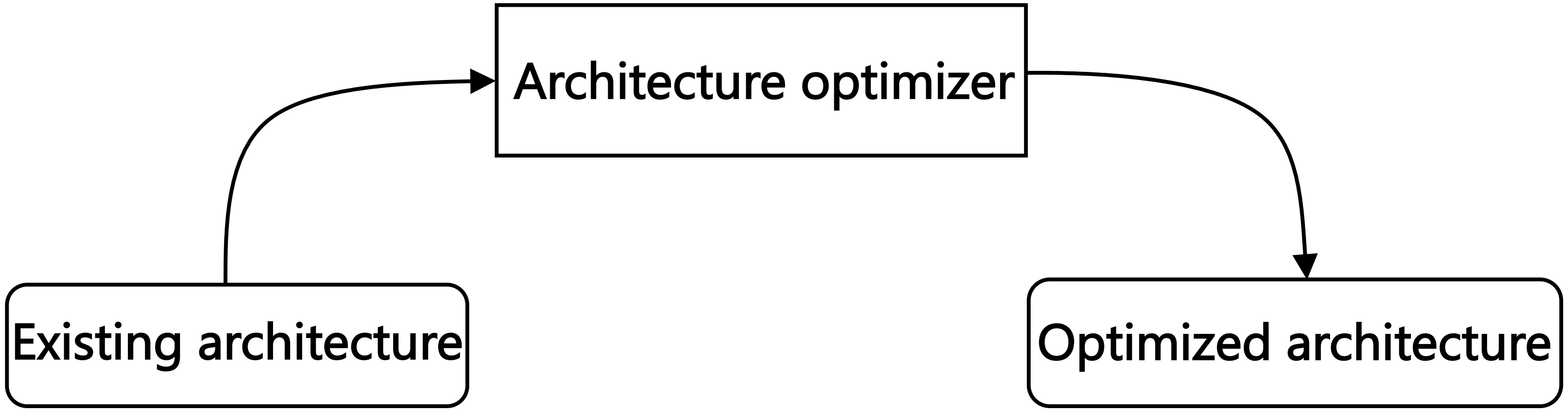
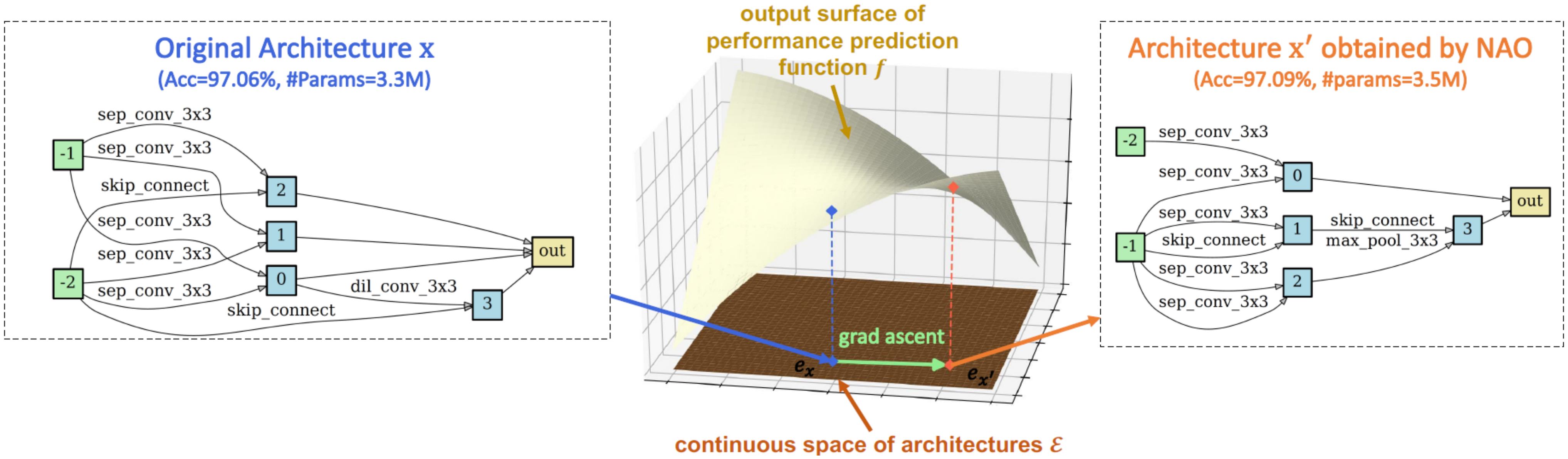


Figure: Architecture optimization scheme.

Existing Architecture Optimization Methods

- Neural Architecture Optimization (NAO)



Limitations of NAO

- NAO may introduce extra parameters or additional computational cost.
- NAO has a NAS search space that is unnecessarily huge and expensive to train.

Contents

Motivation

- Both hand-crafted architectures and NAS based architectures may contain non-significant or redundant operations.
- Existing architecture optimization methods may introduce extra parameters or additional computational cost into the architectures.

How to transform the redundant operations in **any arbitrary architecture** to improve the performance without introducing extra computational cost?

Problem Definition

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

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

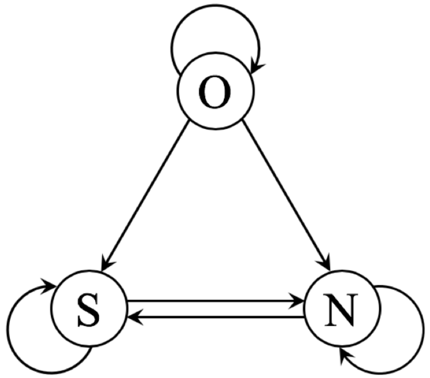


Figure: Operation transformation scheme.

- We divide the operations into three categories $\{S, N, O\}$. S denotes skip connection, N denotes null connection, O denotes the other operations.
- We have $c(O)>c(S)>c(N)$, where $c\,(\cdot)$ evaluates the computational cost.
- To reduce the computational cost, we allow the transitions: $O\rightarrow S$, $O\rightarrow N$, $S\rightarrow N$.
- Since skip connection has negligible cost but often can significantly improve the performance, we also allow $N\rightarrow S$.

Optimization for Arbitrary Architecture

Given any arbitrary architecture $\beta \sim p(\cdot)$, we seek to find the corresponding optimal architecture α . Then, the optimization problem can be formulated as

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

- $R(\alpha|\beta) = R(\alpha, w_\alpha) - R(\beta, w_\beta)$ denotes the performance improvement between the optimized architectures α and the given architectures β. w_α and w_β are the parameters of α and β .
- $c(\cdot)$ is a function to measure the computation cost of architectures.
- κ is an upper bound of the computational cost.

Optimization for Arbitrary Architecture

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

- It is non-trivial to directly obtain the optimal α.
- We instead sample α from the well learned policy, denoted by $\pi(\cdot \mid \beta; \theta)$, *i.e.*, $\alpha \sim \pi(\cdot \mid \beta; \theta)$.

To learn the policy, we solve the following optimization problem:

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

where $\mathbb{E}_{\beta \sim p(\cdot)} [\mathbb{E}_{\alpha \sim \pi(\cdot \mid \beta; \theta)} R(\alpha \mid \beta)]$ denotes the expectation of $R(\alpha \mid \beta)$ over the distribution of $\beta \sim p(\cdot)$ and the distribution of $\alpha \sim \pi(\cdot \mid \beta; \theta)$.

Optimization for Arbitrary Architecture

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

Several challenges regarding the optimization problem

- It is hard to find a comprehensive measure to accurately evaluate the cost.
- The upper bound of computational cost κ is hard to determine.

Markov Decision Process for Learning NAT

Our solution

- We cast the optimization problem into an architecture transformation problem and reformulate it as a Markov decision process (MDP).
- We seek to optimize architectures by making a series of decisions to replace redundant operations with the more computationally efficient operations.

Benefits: We do not have to evaluate the cost $c(\alpha)$ or determine the upper bound κ to obtain an architecture with less computational cost.

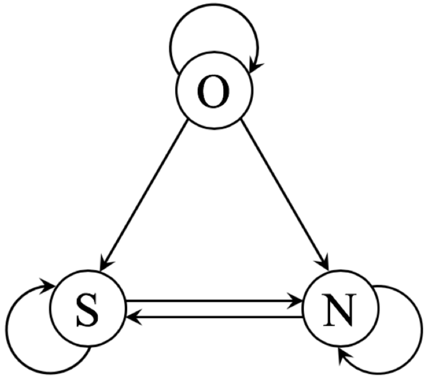


Figure: Operation transformation scheme.

Markov Decision Process for Learning NAT

Details of MDP

- 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**.
- The **policy** $\pi(\cdot \mid \beta; \theta)$ parameterized by θ is the **probability distribution of the action**.

Based on MDP, how to build a model to learn the optimal policy π ?

Policy Learning by Graph Convolution Networks

To better exploit the adjacency information of the operations in an architecture, we use a two-layer graph convolutional network (GCN) to build the controller:

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

Notations

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

Training Method

We train the transformer parameters θ and the model parameter w in an alternative way.

- Training the model parameters w :

$$w \leftarrow w - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_w \mathcal{L}(\beta_i, w)$$

where $\mathcal{L}(\cdot)$ is the cross-entropy loss, η is the learning rate.

- Training the transformer parameters θ :

To encourage exploration, we introduce an entropy regularization term:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\beta \sim p(\cdot)} \left[\mathbb{E}_{\alpha \sim \pi(\cdot|\beta;\theta)} \left[R(\alpha, w) - R(\beta, w) \right] + \underline{\lambda H\big(\pi(\cdot|\beta;\theta)\big)} \right] \\ &= \sum_{\beta} p(\beta) \left[\sum_{\alpha} \pi(\alpha|\beta;\theta) \big(R(\alpha, w) - R(\beta, w) \big) + \underline{\lambda H\big(\pi(\cdot|\beta;\theta)\big)} \right] \end{aligned}$$

where $H(\cdot)$ evaluates the entropy of the policy, and λ controls the strength of the entropy regularization term.

Training Method

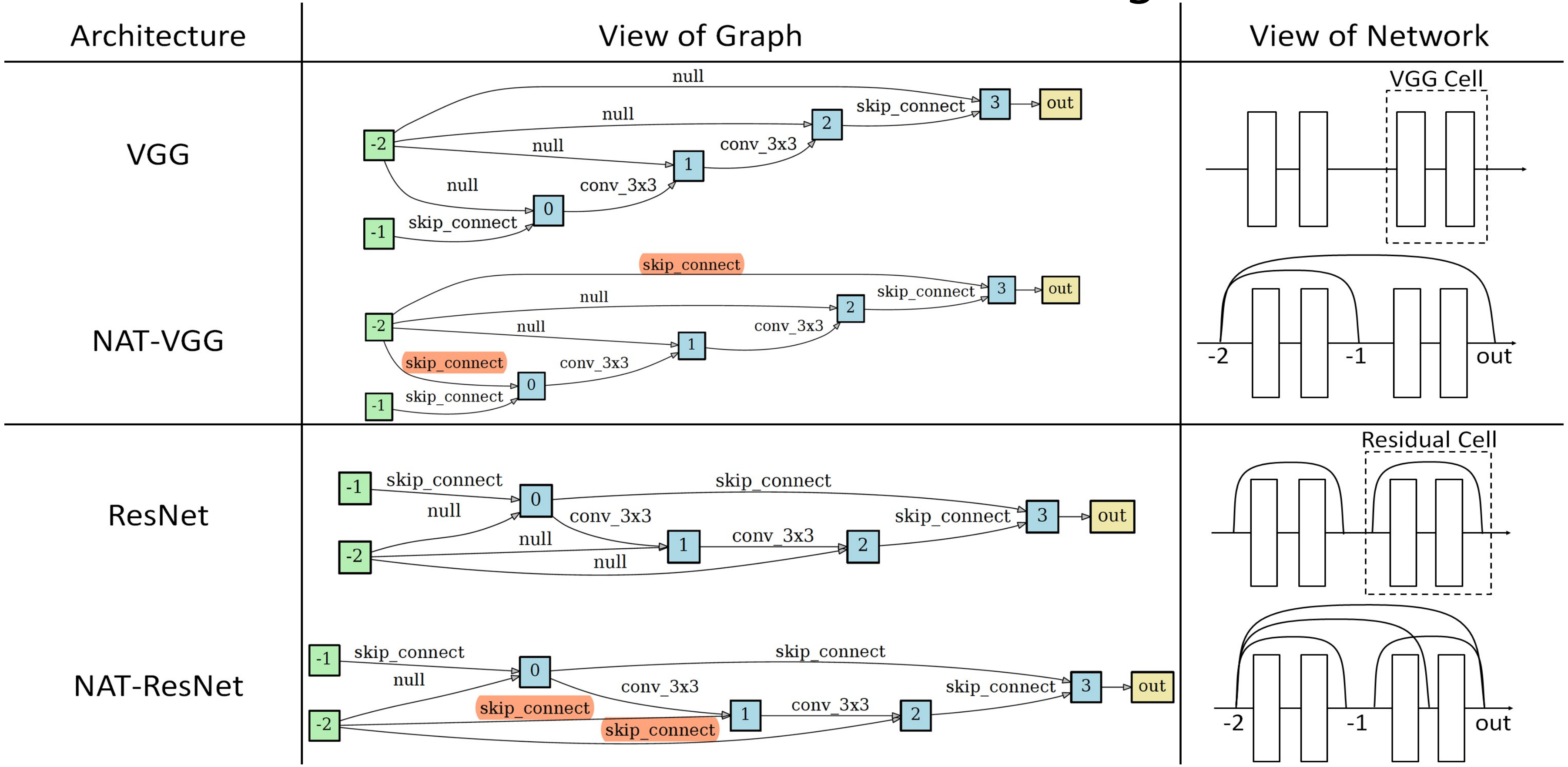
Algorithm 1 Training method for Neural Architecture Transformer (NAT).

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 ascending the gradient.
  end for
end while

Contents

Visual Results of Hand-crafted Architectures

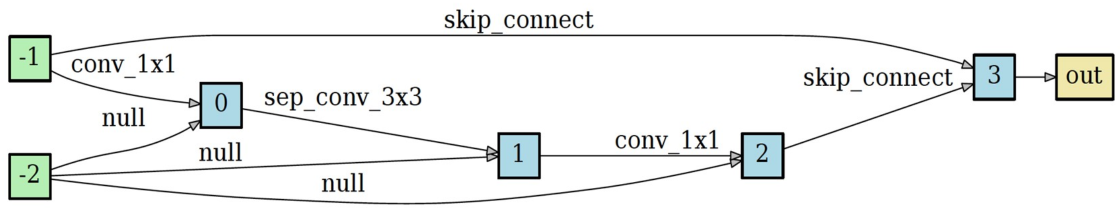
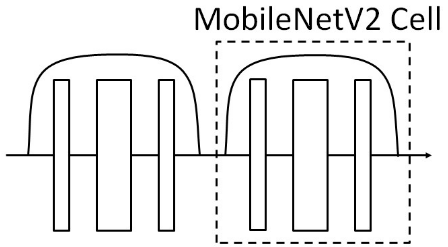
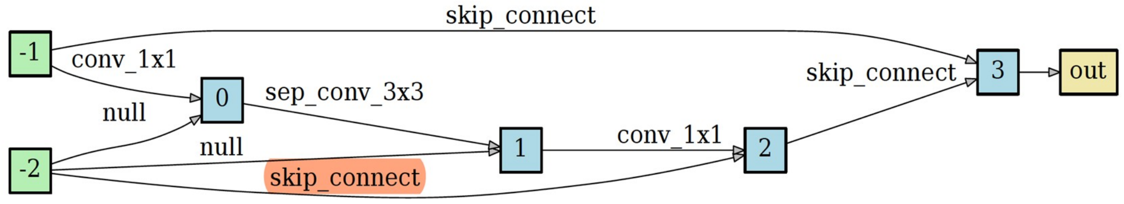
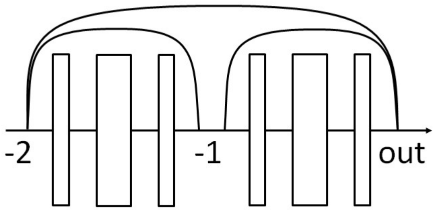
- Results on several hand-crafted architectures, including VGG, ResNet, and MobileNet.



➢ NAT introduces additional skip connections to improve the performance.

Visual Results of Hand-crafted Architectures

- Results on several hand-crafted architectures, including VGG, ResNet, and MobileNet.

Architecture	View of Graph	View of Network
MobileNetV2		
NAT-MobileNetV2		

➢ NAT introduces additional skip connections to improve the performance.

Comparison on Hand-crafted Architectures

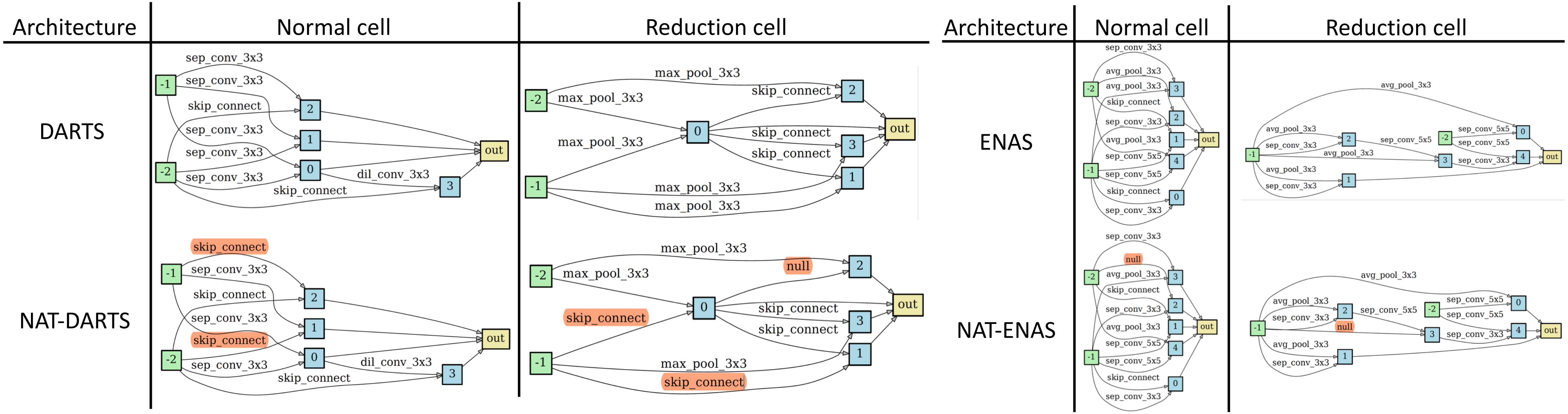
- Results on several hand-crafted architectures, including VGG, ResNet, and MobileNet.

CIFAR-10					ImageNet					
Model	Method	#Params (M)	#MAdds (M)	Acc. (%)	Model	Method	#Params (M)	#MAdds (M)	Acc. (%) Top-1	Acc. (%) Top-5
VGG16	/	15.2	313	93.56	VGG16	/	138.4	15620	71.6	90.4
	NAO[32]	19.5	548	95.72		NAO [32]	147.7	18896	72.9	91.3
	NAT	15.2	315	**96.04**		NAT	138.4	15693	**74.3**	**92.0**
ResNet20	/	0.3	41	91.37	ResNet18	/	11.7	1580	69.8	89.1
	NAO [32]	0.4	61	92.44		NAO [32]	17.9	2246	70.8	89.7
	NAT	0.3	42	**92.95**		NAT	11.7	1588	**71.1**	**90.0**
ResNet56	/	0.9	127	93.21	ResNet50	/	25.6	3530	76.2	92.9
	NAO [32]	1.3	199	95.27		NAO [32]	34.8	4505	77.4	93.2
	NAT	0.9	129	**95.40**		NAT	25.6	3547	**77.7**	**93.5**
MobileNetV2	/	2.3	91	94.47	MobileNetV2	/	3.4	300	72.0	90.3
	NAO [32]	2.9	131	94.75		NAO [32]	4.5	513	72.2	90.6
	NAT	2.3	92	**95.17**		NAT	3.4	302	**72.5**	**91.0**

- NAT based models yield significantly better performance with approximately the same computational cost as the baseline models.

Visual Results on NAS based Architectures

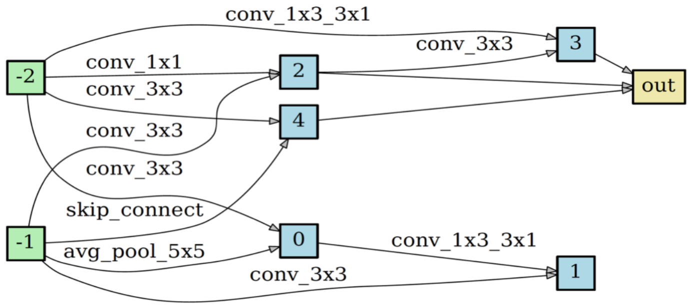
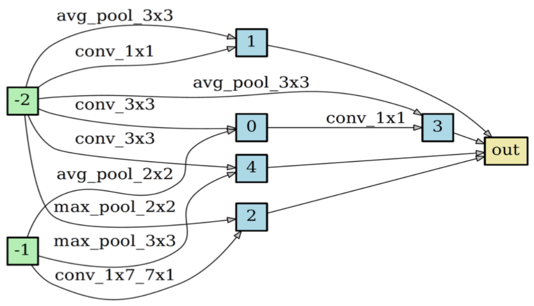
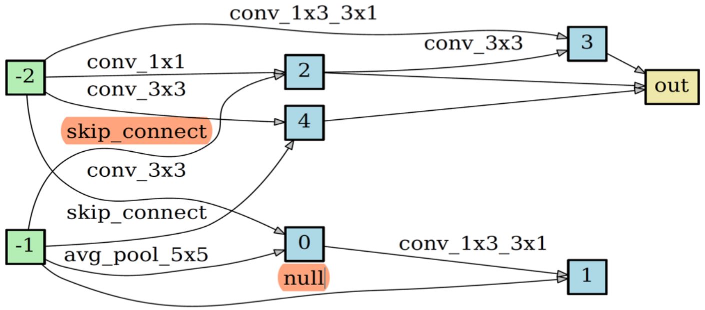
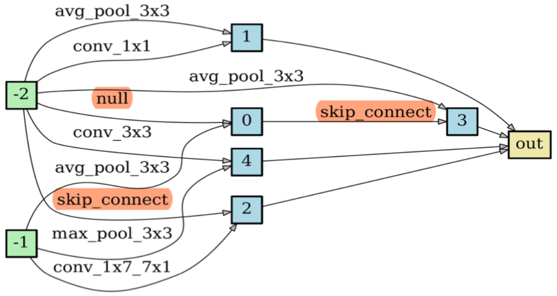
- Results on several NAS based architectures, including ENAS, DARTS, and NAONet.



➢ NAT replaces several redundant operations with the skip connections or directly removes the connections to reduce computation cost.

Visual Results on NAS based Architectures

- Results on several NAS based architectures, including ENAS, DARTS, and NAONet.

Architecture	Normal cell	Reduction cell
NAONet		
NAT-NAONet		

➢ NAT replaces several redundant operations with the skip connections or directly removes the connections to reduce computation cost.

Comparison on NAS based Architectures

- Results on several NAS based architectures, including ENAS, DARTS, and NAONet.

CIFAR-10					ImageNet					
Model	Method	#Params (M)	#MAdds (M)	Acc. (%)	Model	Method	#Params (M)	#MAdds (M)	Acc. (%) Top-1	Acc. (%) Top-5
AmoebaNet† [37]	/	3.2	-	96.73	AmoebaNet [37]	/	5.1	555	74.5	92.0
PNAS† [29]	/	3.2	-	96.67	PNAS [29]	/	5.1	588	74.2	91.9
SNAS† [50]	/	2.9	-	97.08	SNAS [50]	/	4.3	522	72.7	90.8
GHN† [54]	/	5.7	-	97.22	GHN [54]	/	6.1	569	73.0	91.3
ENAS† [36]	/	4.6	804	97.11	ENAS [36]	/	5.6	679	73.8	91.7
	NAO [32]	4.5	763	97.05		NAO [32]	5.5	656	73.7	91.7
	NAT	4.6	804	**97.24**		NAT	5.6	679	**73.9**	**91.8**
DARTS† [30]	/	3.3	533	97.06	DARTS [30]	/	5.9	595	73.1	91.0
	NAO [32]	3.5	577	97.09		NAO [32]	6.1	627	73.3	91.1
	NAT	3.0	483	**97.28**		NAT	3.9	515	**74.4**	**92.2**
NAONet† [32]	/	128	66016	97.89	NAONet [32]	/	11.35	1360	74.3	91.8
	NAO [32]	143	73705	97.91		NAO [32]	11.83	1417	74.5	92.0
	NAT	113	58326	**98.01**		NAT	8.36	1025	**74.8**	**92.3**

➢ NAT based models yield significantly better performance with less or comparable computational cost as the baseline models.

Comparison of Different Policy Learners

- We compare several policy learners, including Random Search, LSTM, and two GCN based methods.

Method	VGG16	ResNet20	MobileNetV2	ENAS†	DARTS†	NAONet†
/	93.56	91.37	94.47	97.11	97.06	97.89
Random Search	93.17	91.56	94.38	96.58	95.17	96.31
LSTM	94.45	92.19	95.01	97.05	97.05	97.93
Maximum-GCN	94.37	92.57	94.87	96.92	97.00	97.90
Sampling-GCN (Ours)	**95.93**	**92.97**	**95.13**	**97.21**	**97.26**	**97.99**

- Our Sampling-GCN method significantly outperforms the other methods.

Contents

Conclusion

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

Thanks!

Q & A