CENG 5030
Energy Efficient Computing

Mo06: Network Architecture Search

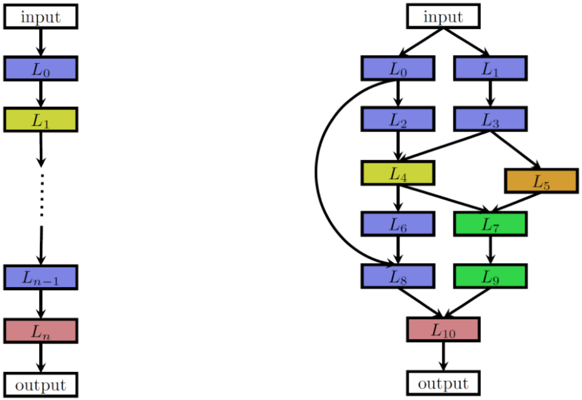
Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Latest update: September 2, 2023)

2023 Fall

Overview

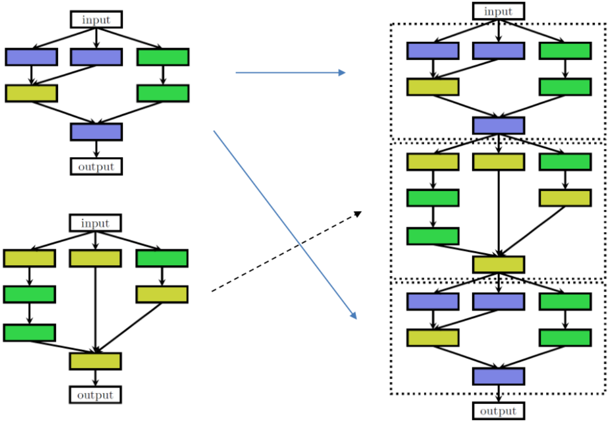
Basic architecture search



Each node in the graphs corresponds to a layer in a neural network [1]

[1] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. (2019). "Neural architecture search: A survey". In: *JMLR* 20.55, pp. 1–21

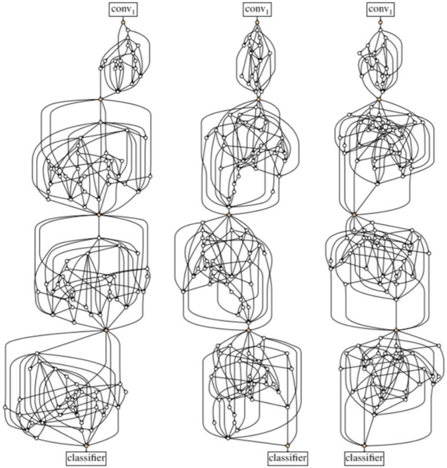
Cell-based search



Normal cell and reduction cell can be connected in different order[2]

[2] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. (2019). "Neural architecture search: A survey". In: *JMLR* 20.55, pp. 1–21

Graph-based search space



Randomly wired neural networks generated by the classical Watts-Strogatz model [3]

[3] Saining Xie et al. (2019). "Exploring randomly wired neural networks for image recognition". In: *Proc. ICCV*, pp. 1284–1293

Estimation strategy

Strategy

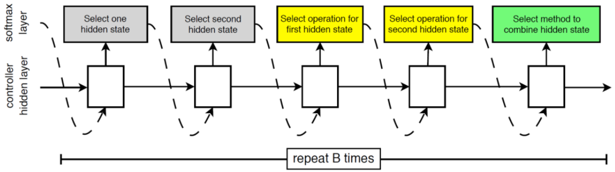
- Task specific
  - Classficiation tasks
    e.g., accuracy, error rate, etc.
  - Segmentation tasks
    e.g., pixel accuracy, MIoU
  - Generation tasks
    e.g., Inception Score, Frechet Inception Score, etc.
- Latency considered factors
  - #FLOPs
  - #Parameters

Tips

Different NAS methods can incorporate diverse factors into search consideration

Blackbox Optimization

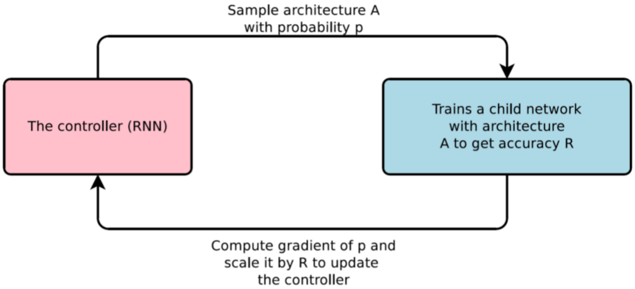
NAS as hyperparameter optimization



Controller architecture for recursively constructing one block of a convolutional cell [4]

- 5 categorical choices for N^{th} block
  - 2 categorical choices of hidden states, each with domain $0, 1, ..., N-1$
  - 2 categorical choices of operations
  - 1 categorical choices of combination method
  - Total number of hyperparameters for the cell: 5B (with B = 5 by default)
- Unstricted search space
  - Possible with conditional hyperparameters
    (but only up to a prespectified maximum number of layers)
  - Example: chain-structured search space
    - Top-level hyperparameter: number of layers L
    - Hyperparameters of layer K conditional on $L \geq k$

[4] Barret Zoph, Vijay Vasudevan, et al. (2018). "Learning Transferable Architectures for Scalable Image Recognition". In: *Proc. CVPR*



Overview of the reinforcement learning method with RNN [5]

Reinforcement learning with a RNN controller

- State-of-the-art results for CIFAR-10, Penn Treebank
- Large computation demands: **800 GPUs for 3-4 weeks, 12, 800 archtectures evaluated**

[5] Barret Zoph and Quoc Le (2017). "Neural Architecture Search with Reinforcement Learning". In: *Proc. ICLR*

Reinforcement learning with a RNN controller

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

where R is the reward (e.g., accuracy on the validation dataset)

Apply REINFORCEMENT rule

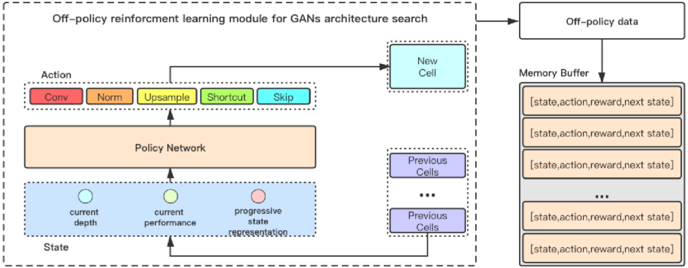
$$\bigtriangledown_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}[\bigtriangledown_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R]$$

Use Monte Carlo approximation with control variate methods, the graident can be approximated by

Approximation of gradients

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \bigtriangledown_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)(R_k - b)$$

Another example on GAN search[6]



Reward define

$$R_t(s,a) = IS(t) - IS(t-1) + \alpha(FID(t-1) - FID(t))$$

The objective loss function

$$J(\pi) = \sum_{t=0} \mathbb{E}_{(s_t,a_t)\ p(\pi)} R(s_t,a_t) = \mathbb{E}_{architecture\ p(\pi)} IS_{final} - \alpha FID_{final}$$

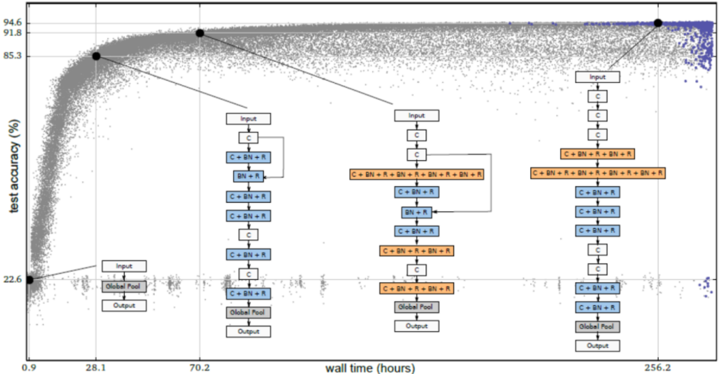
[6] Yuan Tian et al. (2020). "Off-policy reinforcement learning for efficient and effective GAN architecture search". In: *Proc. ECCV*.

Evolution

Evolution methods

Neuroevolution (already since the 1990s)

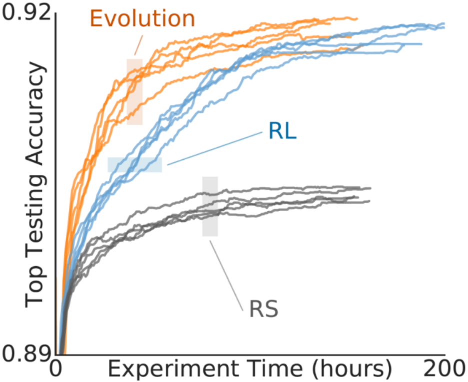
- Typically optimized both architecture and weights with evolutionary methods
  e.g., Angeline, Saunders, and Pollack 1994; Stanley and Miikkulainen 2002
- Mutation steps, such as adding, changing or removing a layer
  e.g., Real, Moore, et al. 2017; Miikkulainen et al. 2017



Regularized / Aging Evolution methods

- Standard evolutionary algorithm e.g. Real, Aggarwal, et al. 2019
  But oldest solutions are dropped from the population (even the best)
- State-of-the-art results (CIFAR-10, ImageNet)
  Fixed-length cell search space

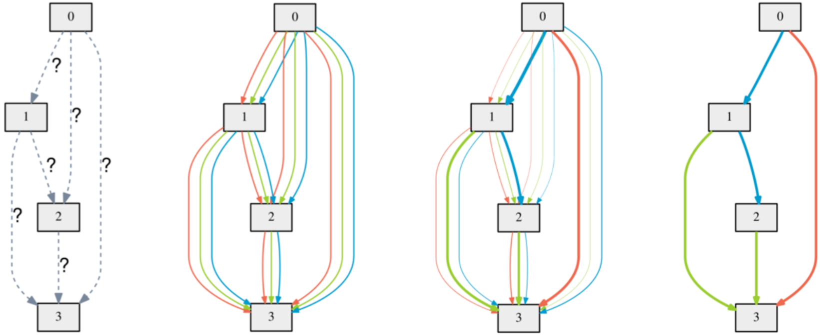
Comparison of evolution, RL and random search



Baysian optimzation methods

- Joint optimization of a vision architecture with 238 hyperparameters with TPE **bergstra2013making**
- Auto-Net
  - Joint architecture and hyperparameter search with SMAC
  - First Auto-DL system to win a competition dataset against human experts **mendoza2016towards**
- Kernels for GP-based NAS
  - Arc kernel
    Swersky, Snoek, and Adams 2013
  - NASBOT
    Kandasamy et al. 2018
- Sequential model-based optimization
  - PNAS
    C. Liu et al. 2018

Differentiable Search



Overview of SNAS [7]

Continous relaxation

$$\bar{O}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{exp(\alpha_O^{(i,j)})}{\sum_{o' \in \mathcal{O}} exp(\alpha_{o'}^{(i,j)})} o(x)$$

[7]Hanxiao Liu, Karen Simonyan, and Yiming Yang (2019). "DARTS: Differentiable architecture search". In: *Proc. ICLR*

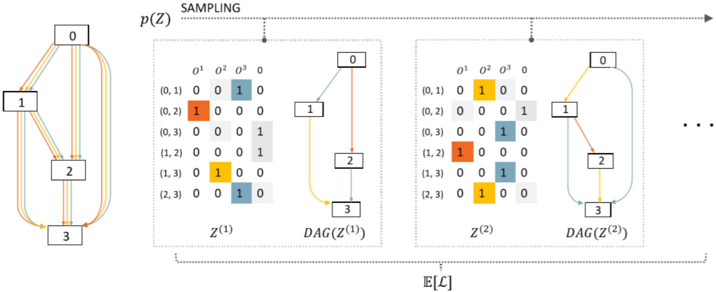
A bi-level optimization

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$s.t. \quad w^*(\alpha) = \underset{w}{argmin}\, \mathcal{L}_{train}(w, \alpha)$$

Algorithm DARTS algorithm

Require: Create a mixed operation $\hat{O}^{(i,j)}$ parameterized by $\alpha^{(i,j)}$ for each edge (i, j)
Ensure: The architecture characterized by α

1: **while** not converged **do**
2: Update architecture α by descending $\bigtriangledown_\alpha \mathcal{L}_{val}(w - \xi \bigtriangledown_w \mathcal{L}_{train}(w, \alpha), \alpha)$
3: ($\xi = 0$ if using first order approximation)
4: Update weights w by descending $\bigtriangledown_w \mathcal{L}_{train}(w, \alpha)$
5: **end while**
6: Derive the findal architecture based on the learned α



Overview of SNAS [8]

Stochastic NAS

$$\mathbb{E}_{Z\ p_\alpha(Z)}[R(Z)] = \mathbb{E}_{Z\ p_\alpha(Z)}[L_\theta(Z)]$$
$$x_j = \sum_{i<j} \tilde{O}_{i,j}(x_i) = \sum_{i<j} Z_{i,j}^T O_{i,j}(x_i)$$

where $\mathbb{E}_{Z\ p_\alpha(Z)}[R(Z)]$ is the objective loss, $Z_{i,j}$ is a one-hot random variable vector to each edge (i,j) in the neural network and x_j is the intermediate node

[8] Sirui Xie et al. (2019). "SNAS: stochastic neural architecture search". In: *Proc. ICLR*

Apply Gummbel-softmax trick to relax the $p_\alpha(Z)$

$$Z_{i,j}^k = f_{\alpha_{i,j}}(G_{i,j}^k) = \frac{exp(\frac{(\log \alpha_{i,j}^k + G_{i,j}^k)}{\lambda})}{\sum_{l=0}^n exp(\frac{\log \alpha_{i,j}^l + G_{i,j}^l}{\lambda})}$$

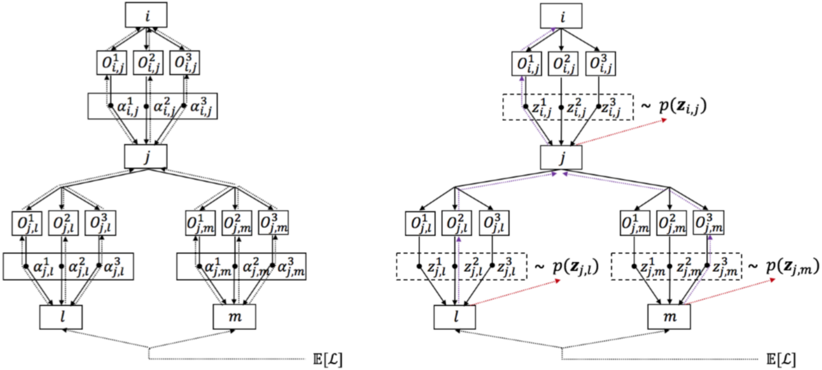
where $Z_{i,j}$ is the softened one-hot random variable, $\alpha_{i,j}$ is the architecture parameter, λ is the temperature of the Softmax function, and $G_{i,j}^k$ satisfies that

Gumbel distribution

$$G_{i,j}^k = -\log\left(-\log\left(U_{i,j}^k\right)\right)$$

where $U_{i,j}^k$ is a uniform random variable

Difference between DARTS and SNAS



A comparison between DARTS (i.e., the left) and SNAS (i.e., the right) [9]

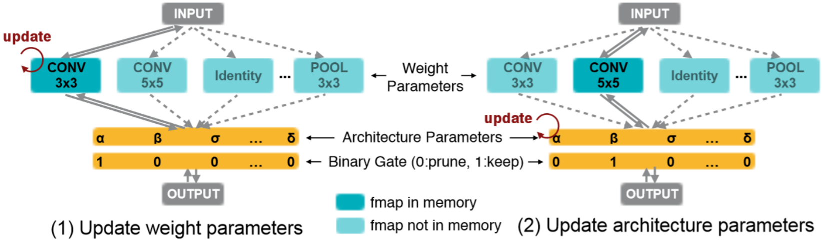
Summary

- Deterministic gradients in DARTS and Stochastic gradients in SNAS
- DARTS require that the derived neural network should be retrained while SNAS has no need

Discretize methods

Discretize the search space

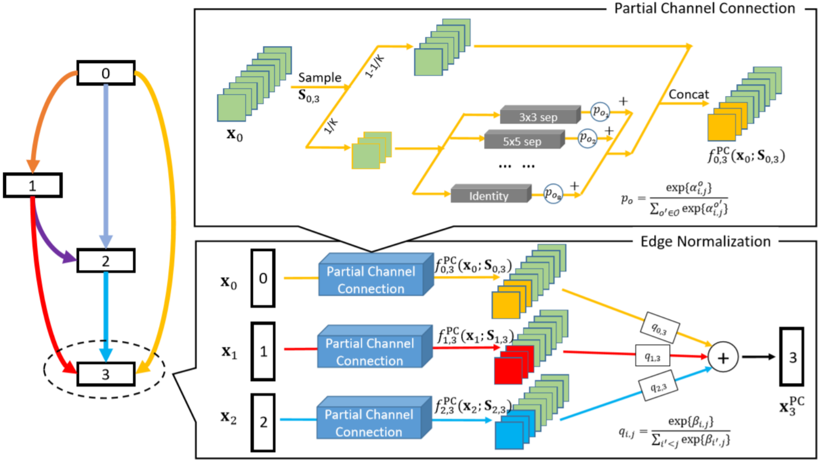
Discretize the search space (e.g., operators, path, channels etc.) to achieve efficient NAS algorithms



Learning both weight parameters and binarized architecture parameters [10]

[10] Han Cai, Ligeng Zhu, and Song Han (2019). "ProxylessNAS: Direct neural architecture search on target task and hardware". In: *Proc. ICLR*

Another example: PC-DARTS



Overview of PC-DARTS. [11]

[11] Yuhui Xu et al. (2020). "PC-DARTS: Partial channel connections for memory-efficient differentiable architecture search". In: *Proc. ICLR*

Discretize methods

Partial channel connection

$$f_{i,j}^{PC}(x_i; S_{i,j}) = \sum_{o\in\mathcal{O}} \frac{exp\alpha_{i,j}^{o}}{\sum_{o'\in\mathcal{O}} exp\alpha_{i,j}^{o'}} \cdot (S_{i,j} * x_i) + (1 - S_{i,j} * x_i)$$

where $S_{i,j}$ defines a channel sampling mask, which assigns 1 to selected channels and 0 to masked ones.

Edge normalization

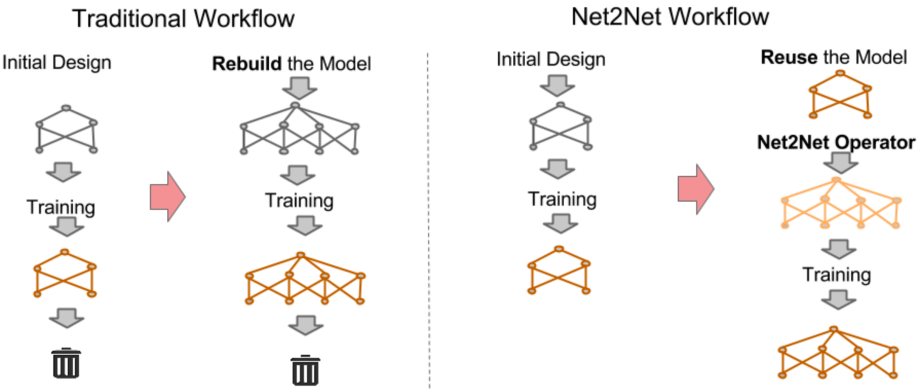
$$x_j^{PC} = \sum_{i<j} \frac{exp\beta_{i,j}}{\sum_{i'<j} exp\beta_{i',j}} \cdot f_{i,j}(x_i)$$

Edge normalization can mitigate the undesired fluctuation introduced by partial channel connection

Other Tips

Tip 1: Network morphisms[12]

- Change the network structure, but not the modelled function
  i.e., for every input the network yields the same output as before applying the network morphism
- Allow efficient moves in architecture space



[12] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens (2016). "Net2Net: Accelerating learning via knowledge transfer". In: *Proc. ICLR*.

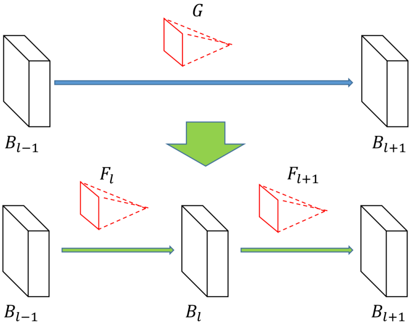
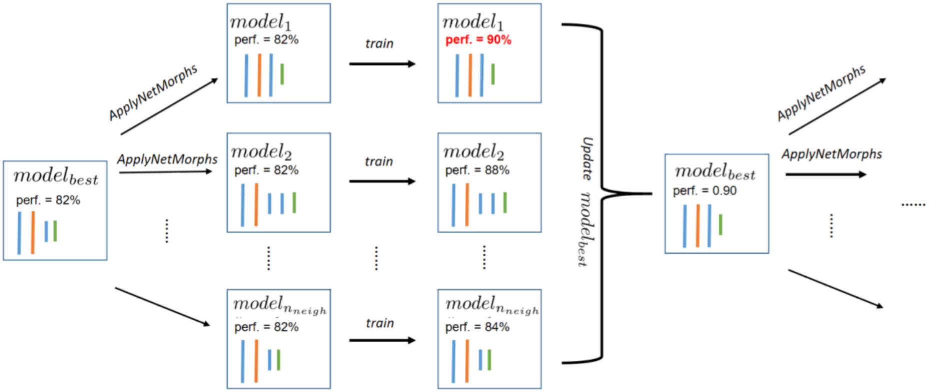


Figure 2: Network morphism linear. B_* represents blobs (hidden units), G and F_* are convolutional filters (weight matrices) for DCNNs (classic neural networks). G is morphed into F_l and F_{l+1}, satisfying Equation (6).

[13]Tao Wei et al. (2016). "Network morphism". In: *Proc. ICML*, pp. 564–572.

Tip 1: Network morphisms[14]



[14] Thomas Elsken, J Metzen, and Frank Hutter (2017). "Simple and efficient architecture search for CNNs". In: *Workshop on Meta-Learning at NIPS*.

Tip 2: Multi-fidelity optimization[15]

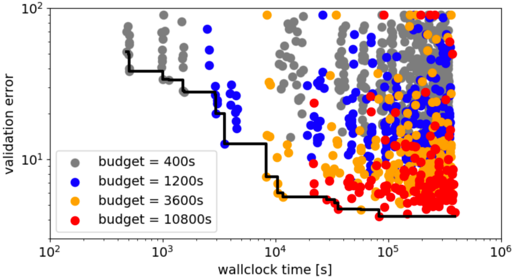


Figure 1: Validation error of all configurations evaluated on the different budgets during the whole optimization procedure. The best performing configuration (incumbent) as a function of time is visualized by the black line.

[15] Arber Zela et al. (2018). "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search". In: *arXiv preprint arXiv:1807.06906*.

NAS Benchmark

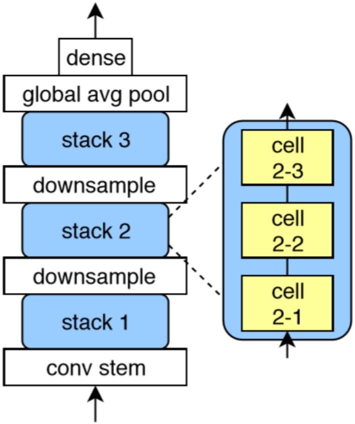
The motivation

NAS algorithms are hard to reproduce normally

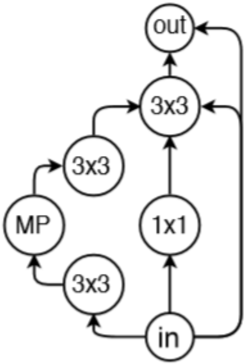
- Some NAS algorithms require months of compute time, making these methods inaccessible to most researchers
- Different proposed NAS algorithms are hard to compare since their different training procedures and different search spaces

Related works

- Chris Ying et al. (2019). "NAS-Bench-101: Towards reproducible neural architecture search". In: *Proc. ICML*, pp. 7105–7114
- Xuanyi Dong and Yi Yang (2020). "NAS-Bench-102: Extending the scope of reproducible neural architecture search". In: *Proc. ICLR*



The stem of the search space



Operation on node

The stem is composed of three cells, followed by a downsampling layer. The downsampling layer halves the height and width of the feature map via max-pooling and the channel count is doubled. The pattern are repeated three times, followed by global average pooling and a final dense softmax layer. The initial layer is a stem consisting of one 3×3 convolution with 128 output channels.

NAS-Bench-101

The space of cell architectures is a directed acyclic graph on V nodes and E edges, each node has one of L labels, representing the corresponding operation. The constraints on the search space

The search space

- $L = 3$
  - 3×3 convolution
  - 1×1 convolution
  - 3×3 max-pool
- $V \leq 7$
- $E \leq 9$
- input node and output node are pre-defined on two of V nodes

Encoding is implemented as a 7×7 upper-triangular binary matrix, by de-duplication and verification, there are **423, 000** neural network architectures

NAS-Bench-101

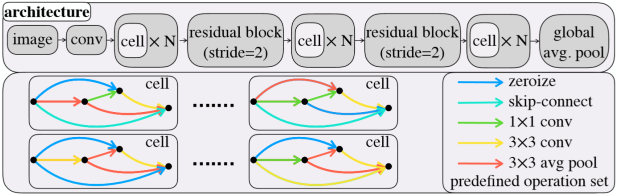
The dataset of NAS-Bench-101 is a mapping from the $(A, Epoch, trial\#)$ to

- Training accuracy
- Validation accuracy
- Testing accuracy
- Training time in seconds
- Number of trainable parameters

Applications

- Compare different NAS algorithms
- Research on generalization abilities of NAS algorithms

NAS-Bench-201



Top: the macro skeleton of each architecture candidate. **Bottom-left**: examples of neural cell with 4 nodes. Each cell is a directed acyclic graph, where each edge is associated with an operation selected from a predefined operation as shown in **Bottom-right**

Comparison between NAS-Bench-101 and NAS-Bench-201

NAS-Bench-101 uses Operation on node while NAS-Bench-201 uses Operation on edge as its search space

	#architectures	#datasets	$\|\mathcal{O}\|$	Search space constraint	Supported NAS alogrithms	Diagnostic information
NAS-Bench-101	510M	1	3	constrain #edges	partial	-
Nas-Bench-201	15.6K	3	5	no constraint	all	fine-grained info. (e.g., #params, FLOPs, latency)