Neural Architecture Search

EECE695D: Efficient ML Systems

Spring 2025

Recap

- Suppose that we have:
  - a neural network architecture, denoted by arch
  - an optimization algorithm, denoted by opt

- **KD.** Given a small NN architecture, optimize it well?

$$\min_{\mathrm{opt}} \quad \mathrm{loss}(\mathrm{arch}, \mathrm{opt})$$

($\mathrm{loss}(\,\cdot\,,\,\cdot\,)$ denotes the loss after training)

Overview

- Consider optimizing another variable:
  - Find a NN architecture that can be trained well

$$\min_{\text{arch}} \quad \text{loss}(\text{arch}, \text{opt})$$

    - Put a constraint / regularizer on the $\text{size}(\text{arch})$
      - FLOPs
      - Memory constraint

Motivation

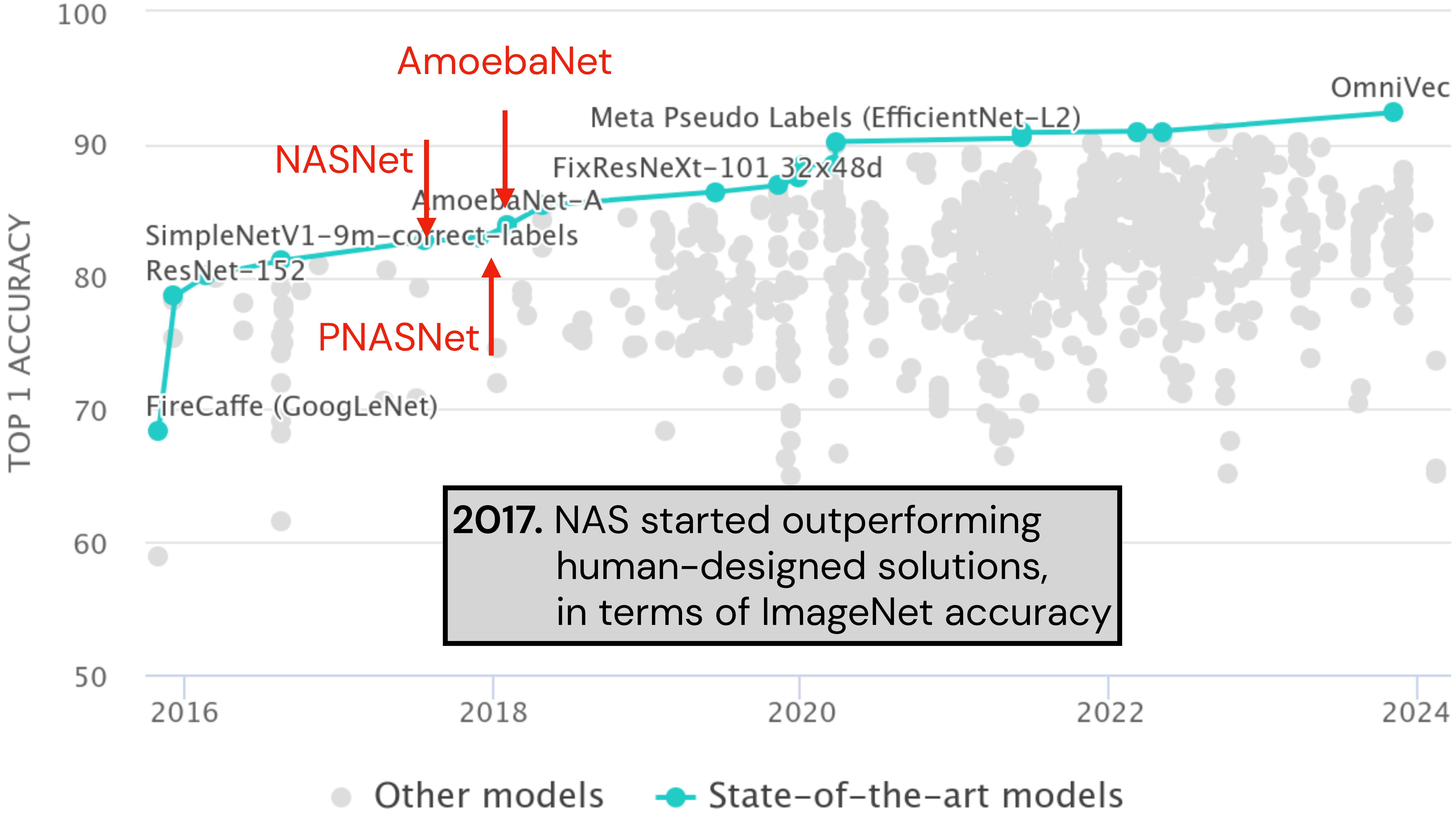
- **Old solution.** Let 👱‍♂️ optimize
  - Number of layers
  - Number of channels (in each layer)
  - Activation function
  - Operator type
  - (....)
- **NAS.** Let 🤖 optimize!

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Table 1. Specification for MobileNetV3-Large. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes the type of nonlinearity used. Here, HS denotes h-swish and RE denotes ReLU. NBN denotes no batch normalization. s denotes stride.

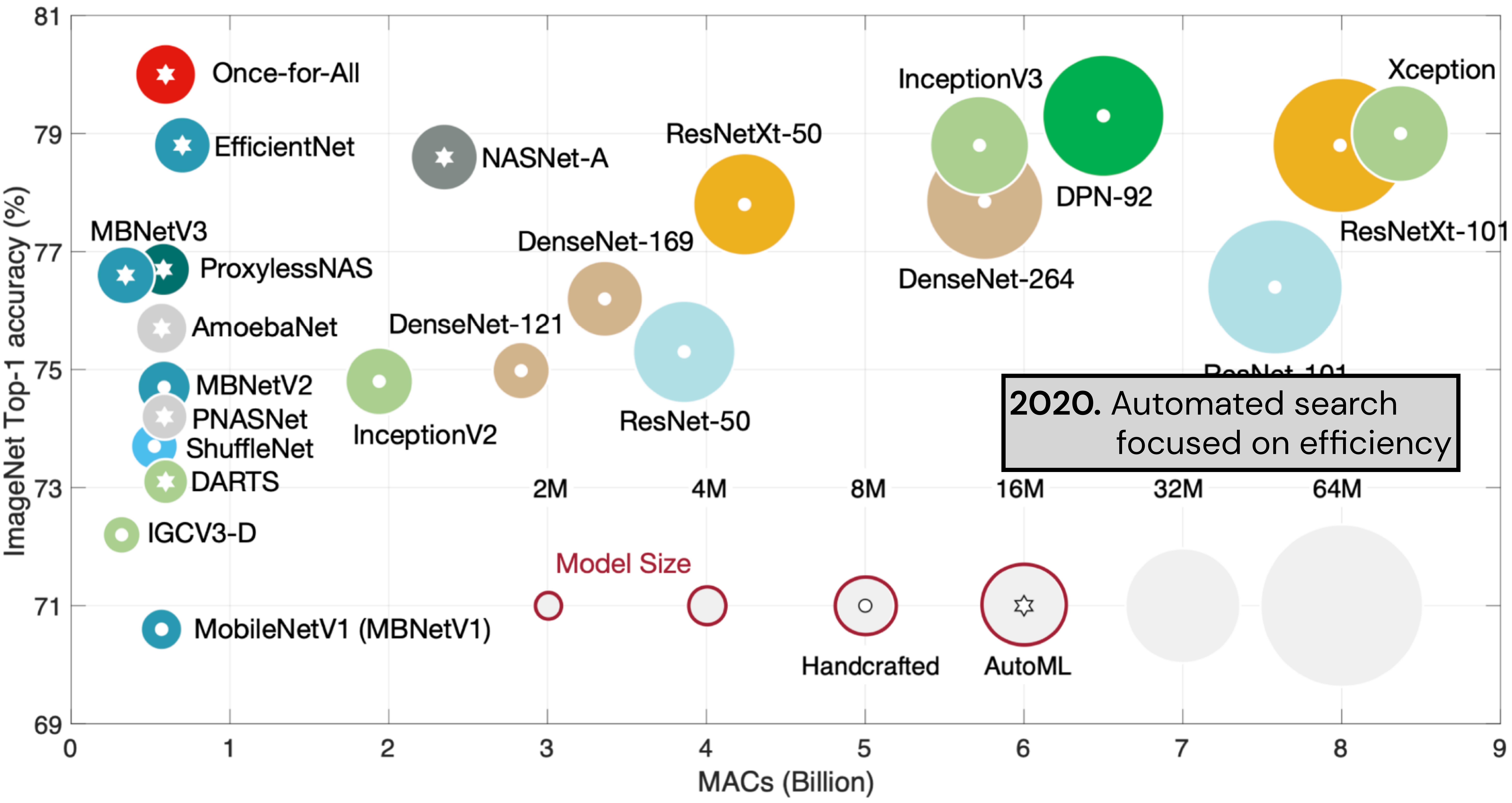
Howard et al., "Searching for MobileNetv3," ICCV 2019

Motivation



2017. NAS started outperforming human-designed solutions, in terms of ImageNet accuracy

paperswithcode.com



Cai et al., "Once-for-all: Train one network and specialize it for efficient deployment," ICLR 2020

Basic idea

Idea

- Ultimately, NAS is about solving

$$\min_{a \in \mathscr{A}} \quad \ell(\mathrm{a})$$

  - $\mathscr{A}$: Search space (e.g., all possible neural nets)
  - $\ell(\,\cdot\,)$: Test loss after training

- **Problem.**
  - Search space is too big
  - Search space is discrete
  - Evaluating loss $\ell(\,\cdot\,)$ takes much compute

Idea

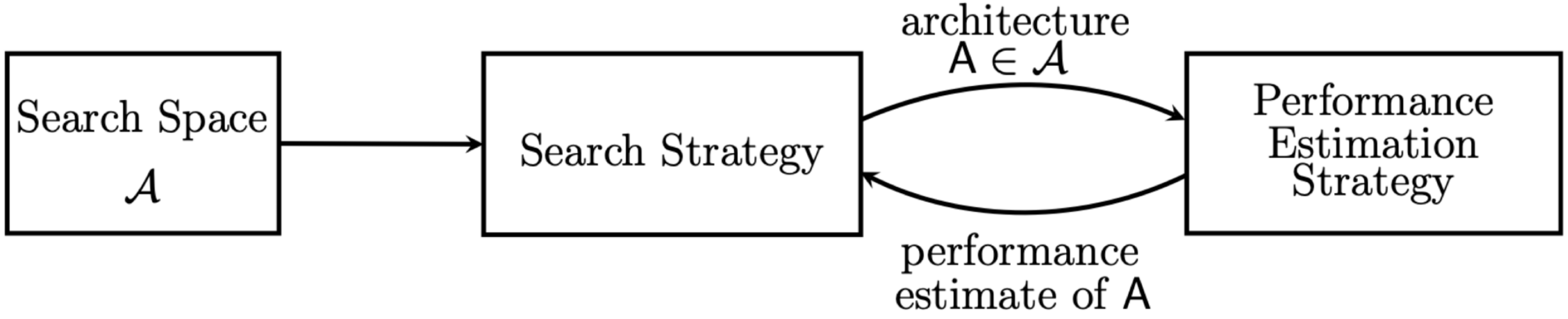
$$\min_{a \in \mathscr{A}} \quad \ell(\mathrm{a})$$

- **Dumb Approach.** A computational nightmare
    - Construct $\mathscr{A}$ as a set of all possible neural nets
        - Pick a model $a \in \mathscr{A}$
        - Train it until convergence
        - Evaluate $\ell(a)$
        - Repeat, until we evaluate all models

Elements

$$\min_{a \in \mathcal{A}} \ell(a)$$

- **Trick.** Simplify the problem in three senses:
  - Search space. Use human / experience-based priors
  - Search strategy. Discrete search algorithms or relaxation
  - Evaluation strategy. Use cheaper proxies



Elsken et al., "Neural Architecture Search: A Survey," JMLR 2019

Search space

Search space

$$\min_{a \in \mathscr{A}} \quad \ell(\mathrm{a})$$

- Defines which architecture can be represented
- **Idea.** Narrow down with human priors
- Look at many different levels:
  - Elementary Ops
  - Blocks
  - Cells

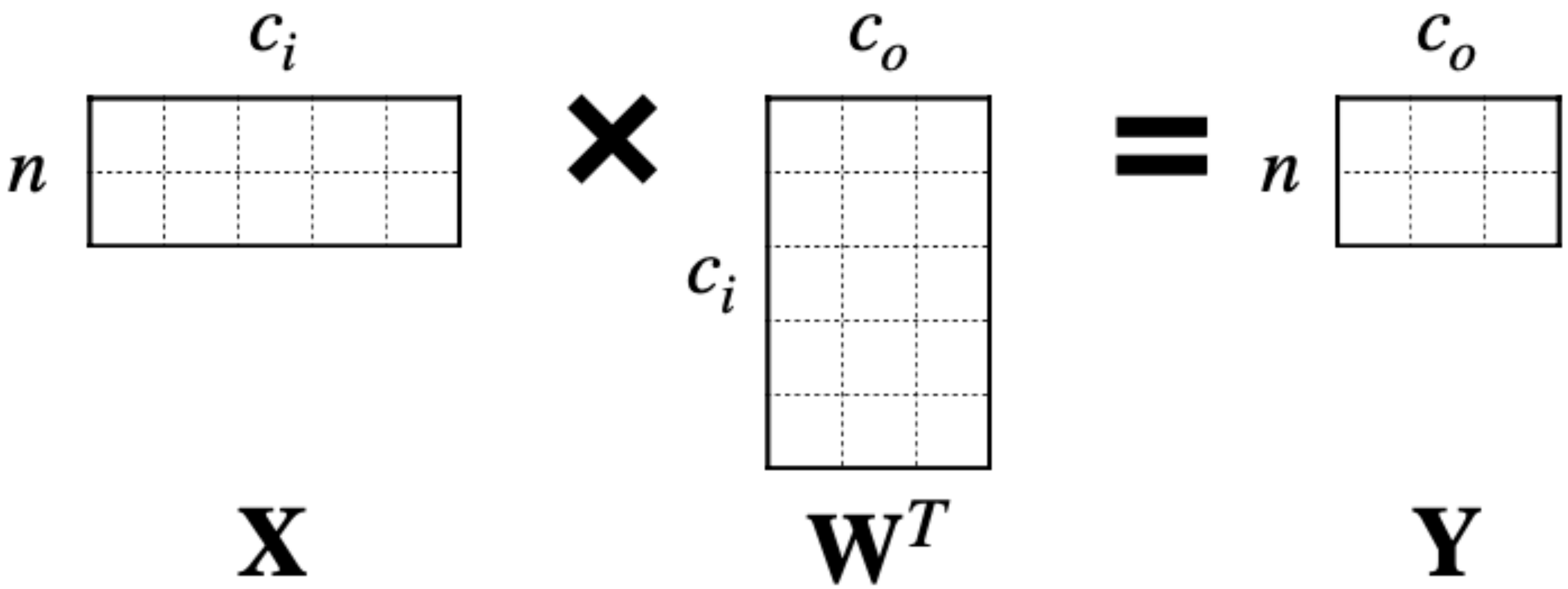
Elementary Ops

- Already much effort to improve the efficiency
    - Recap

Linear

- a.k.a. Dense layer / Fully-connected layer
  - Matrix multiplication

- **Params.** $c_i \cdot c_o$
- **Compute.** $c_i \cdot c_o$



Convolution

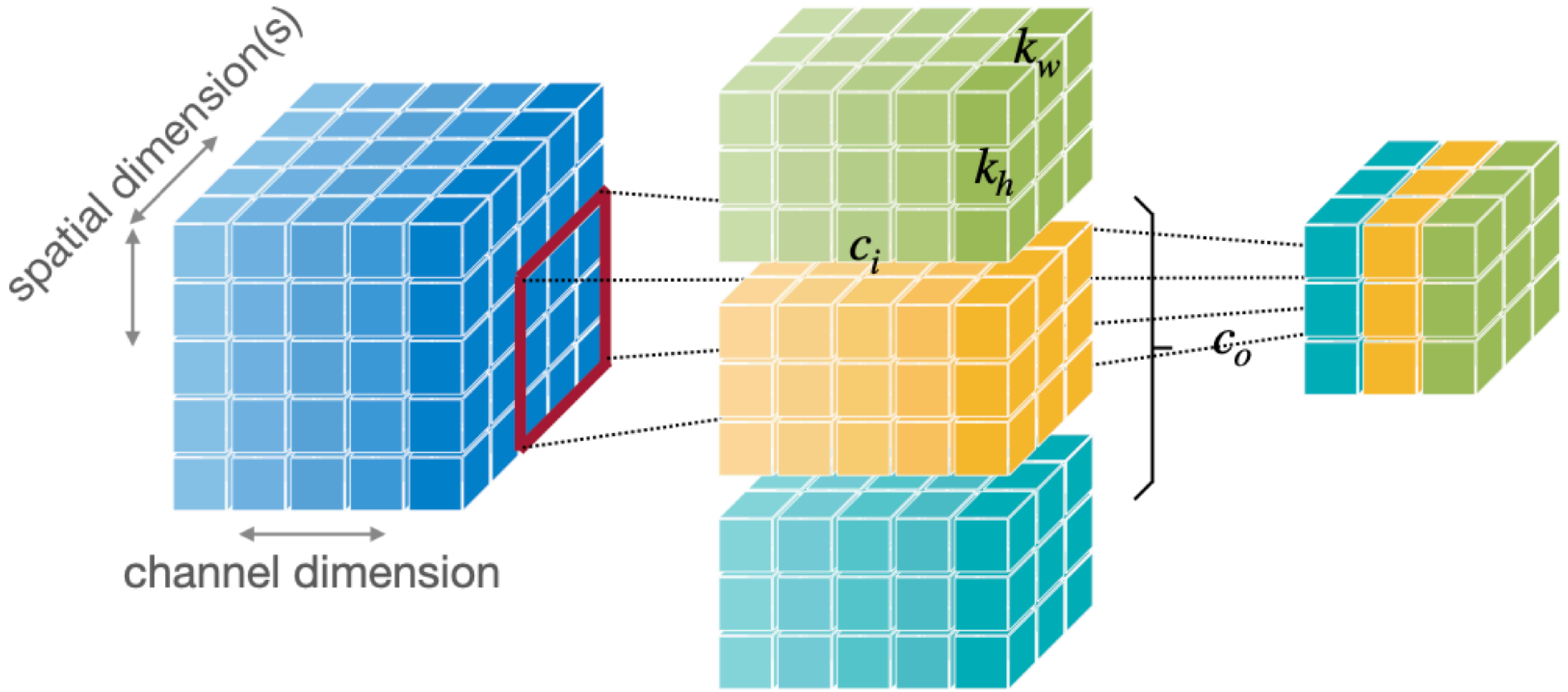
- Parameter sharing for efficiency

- **Params.** $k_h k_w c_i c_o$

  (reduced by $h_i w_i h_o w_o / k_h k_w$)

- **Compute.** $k_h k_w c_i c_o h_o w_o$

  (reduced by $h_i w_i / k_i k_o$)



Grouped convolution

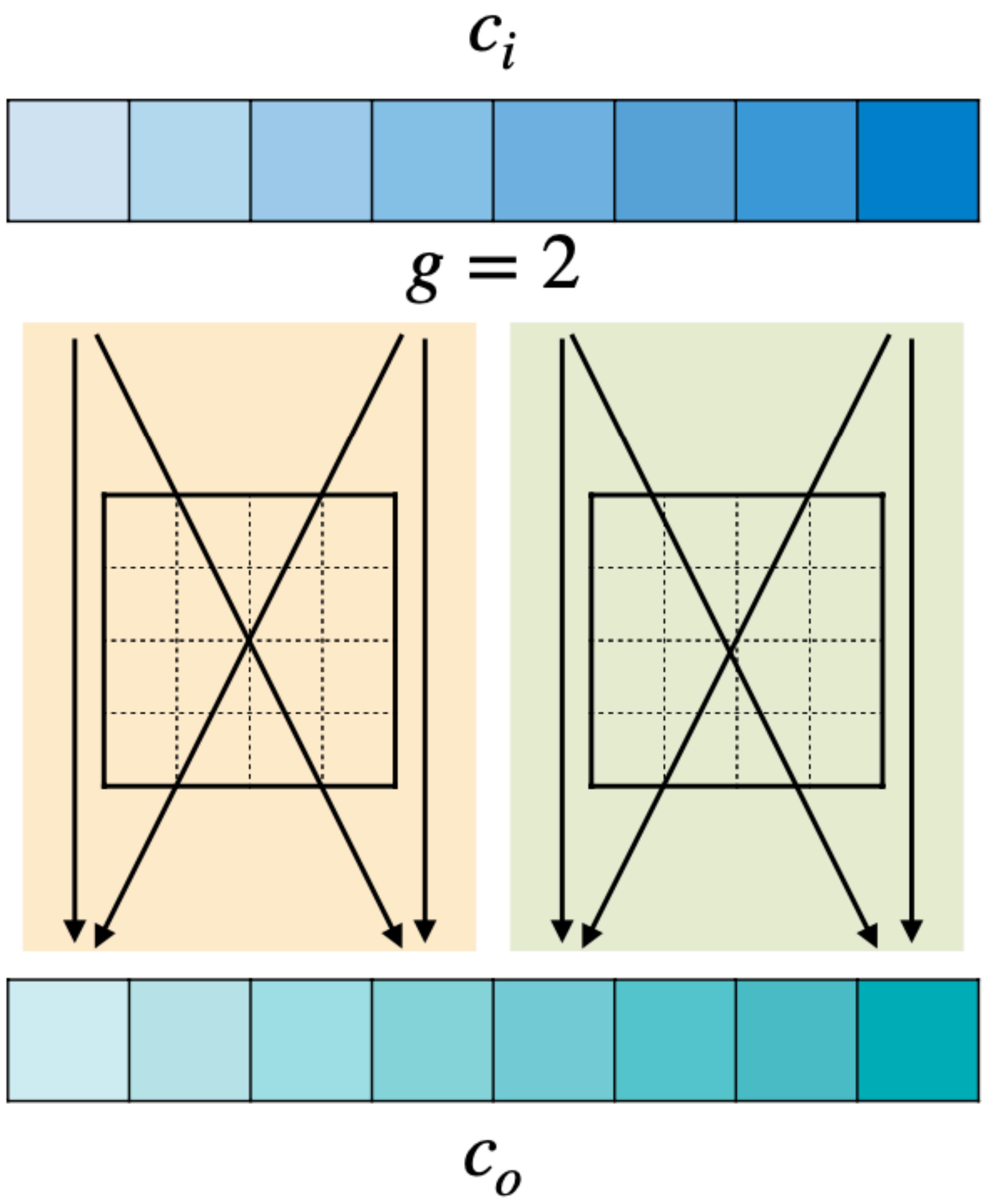
- Certain input channels only affect certain output channels

- **Params.** $k_h k_w c_i c_o / g$

  (reduced by g)

- **Compute.** $k_h k_w c_i c_o h_o w_o / g$

  (reduced by g)

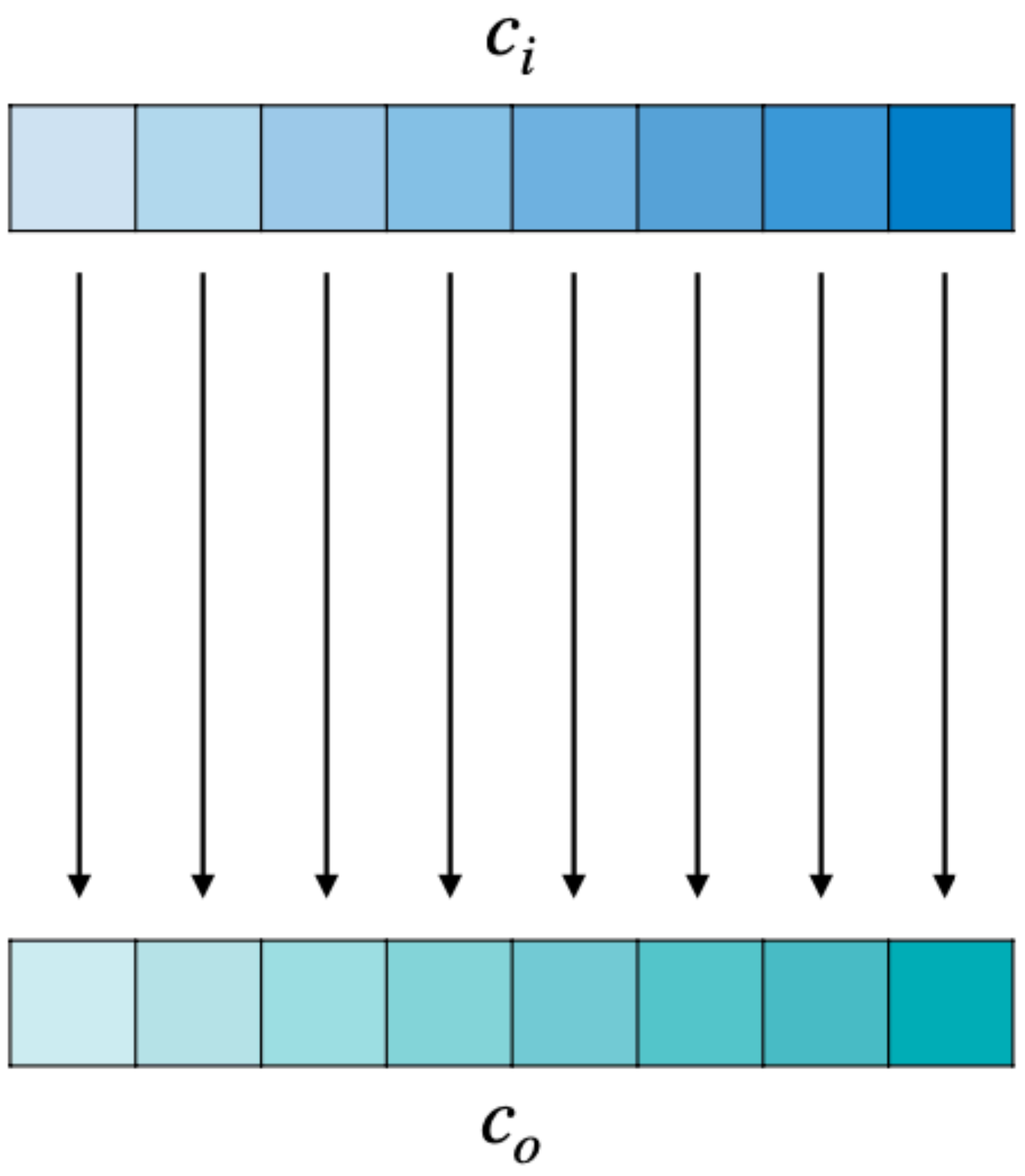


Depthwise convolution

- Group for every channel
    - Linear increase in cost in terms of the #channels (⇔ quadratic)
- **Params.** $k_h k_w c$

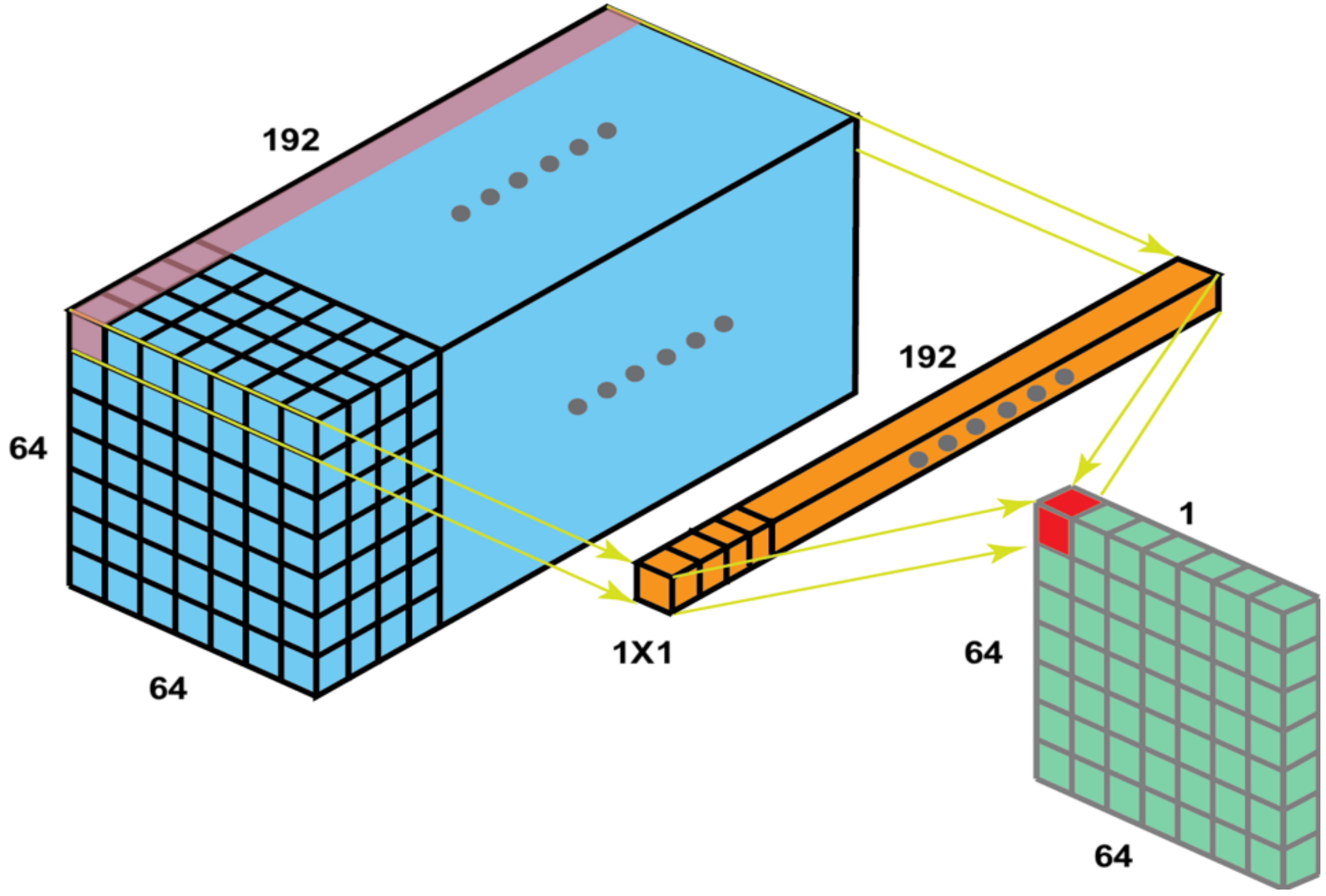
  (reduced by c over conv2d)
- **Compute.** $k_h k_w h_o w_o c$

  (reduced by c over conv2d)



1x1 convolution

- Only mixes information between channels
    - Complementary to depthwise
- **Params.** c^2
- **Compute.** $h_i w_i c^2$

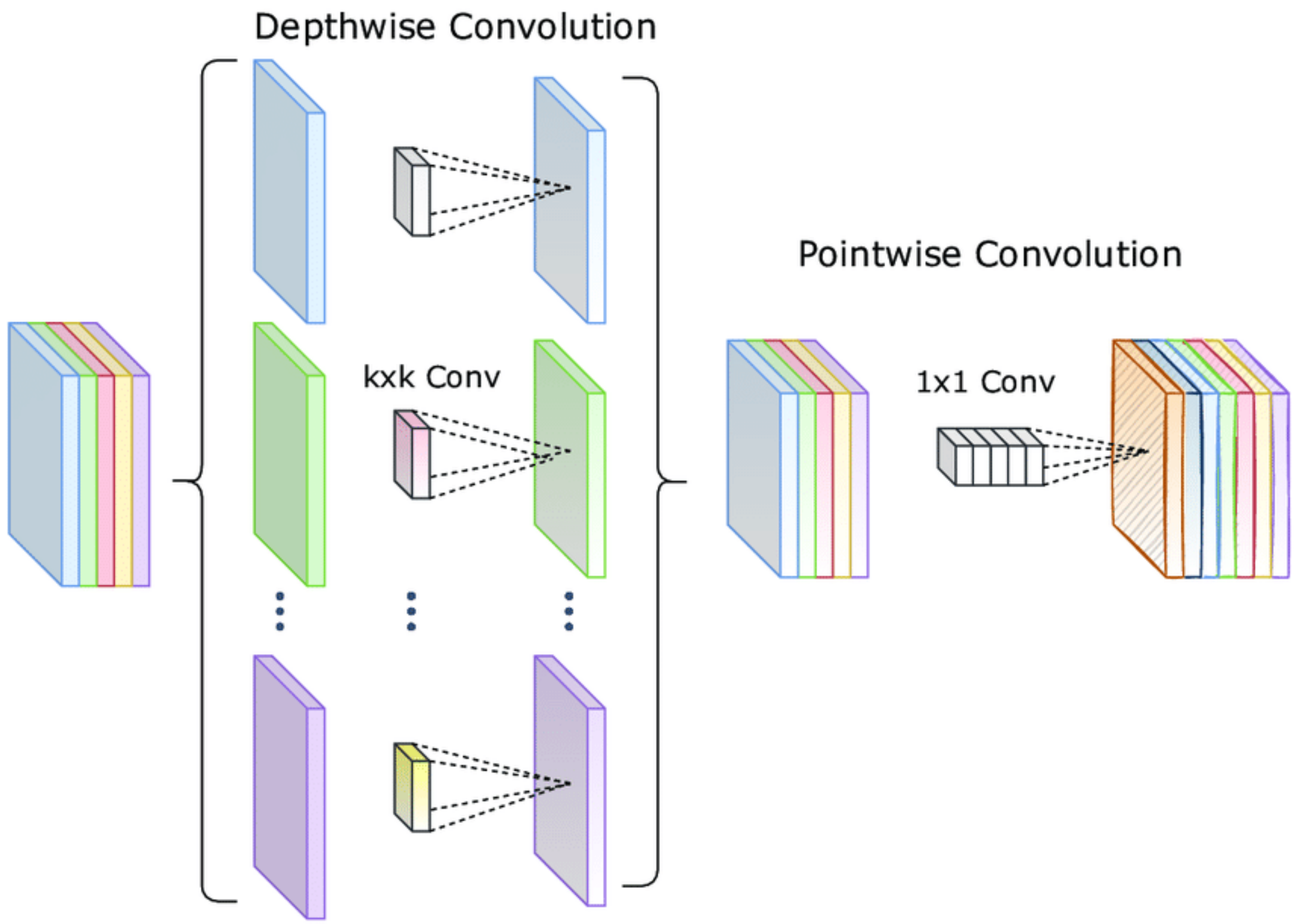


Handmade blocks

- Many works have already combined several ops into **blocks**
  - Focused on efficiency
  - Will be a motivation of how we build search space

MobileNet: Depthwise + 1x1

- Depthwise Conv → 1x1 Conv
  (intra-channel) (inter-channel)
  - c.f. transformers



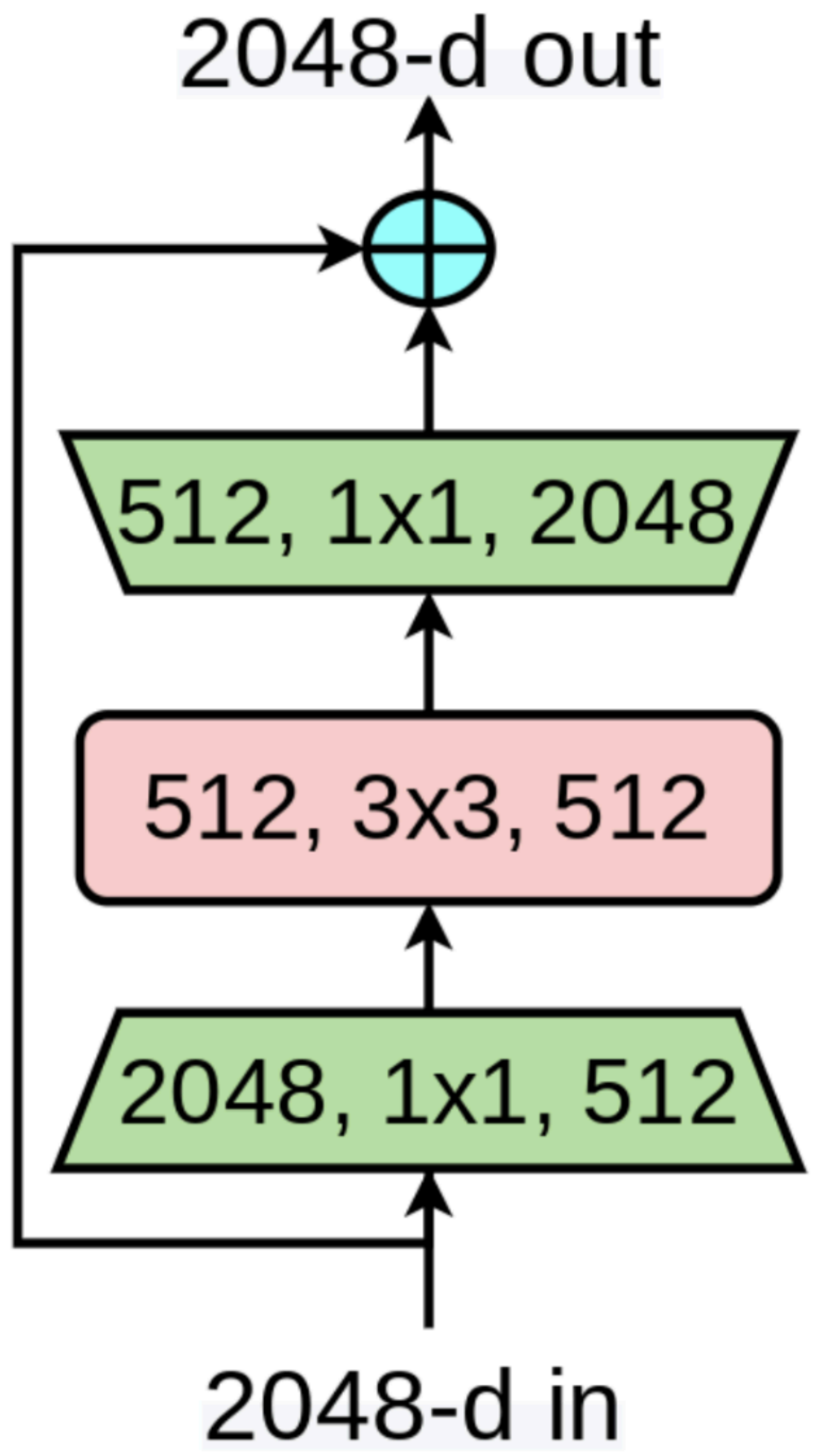
- By replacing Conv → Depthwise + 1x1:
  - **Params.** $k^2c^2 \rightarrow k^2c + c^2$
  - **Compute.** $hwk^2c^2 \rightarrow hw(k^2c + c^2)$

https://www.researchgate.net/figure/Depthwise-separable-convolutions_fig1_358585116
Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv 2017

ResNet: Bottlenecks

- Decrease #channels → do 3x3 → increase back

- By replacing Conv → Bottleneck:
  (with channel reduction factor r)
  - **Params.** $k^2c^2 \rightarrow (2/r + k^2/r^2)c^2$
  - **Compute.** $hwk^2c^2 \rightarrow hw(2c^2/r + k^2c^2/r^2)$



He et al., "Deep residual learning for image recognition," CVPR 2016

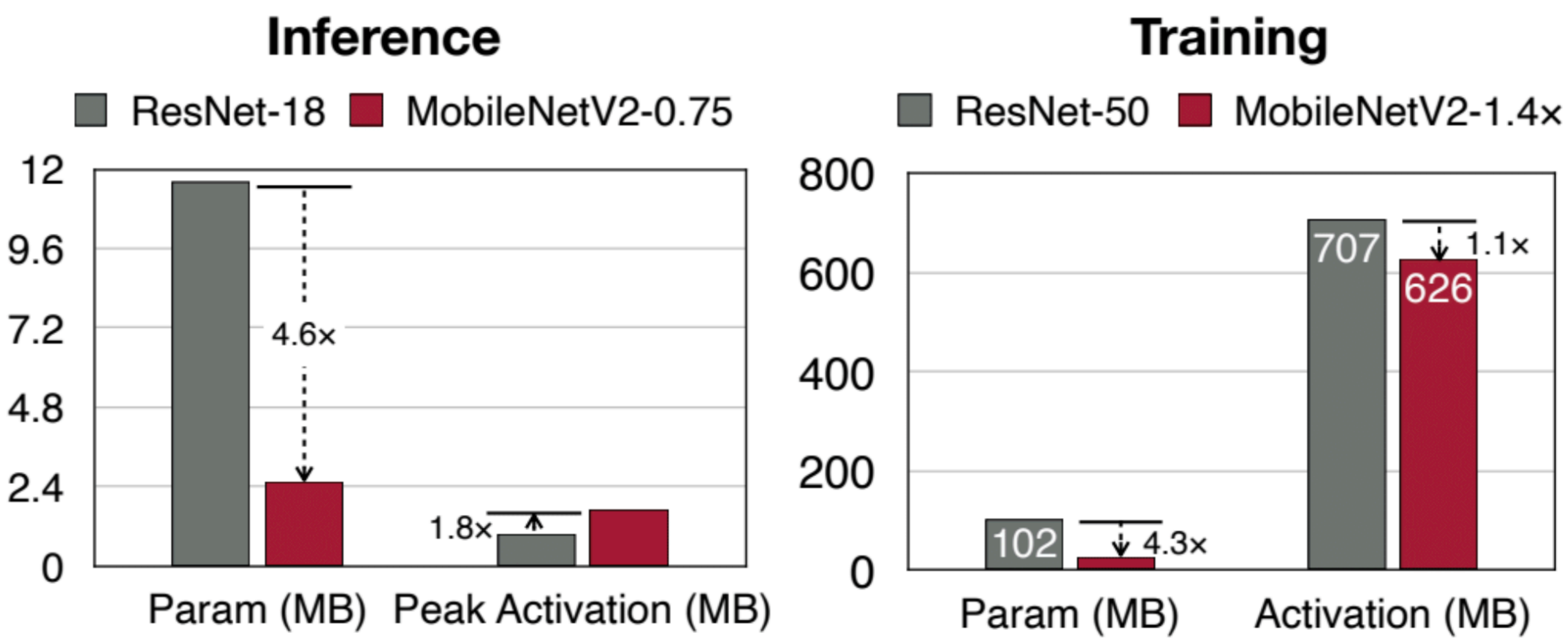
ResNeXT: Grouped bottlenecks

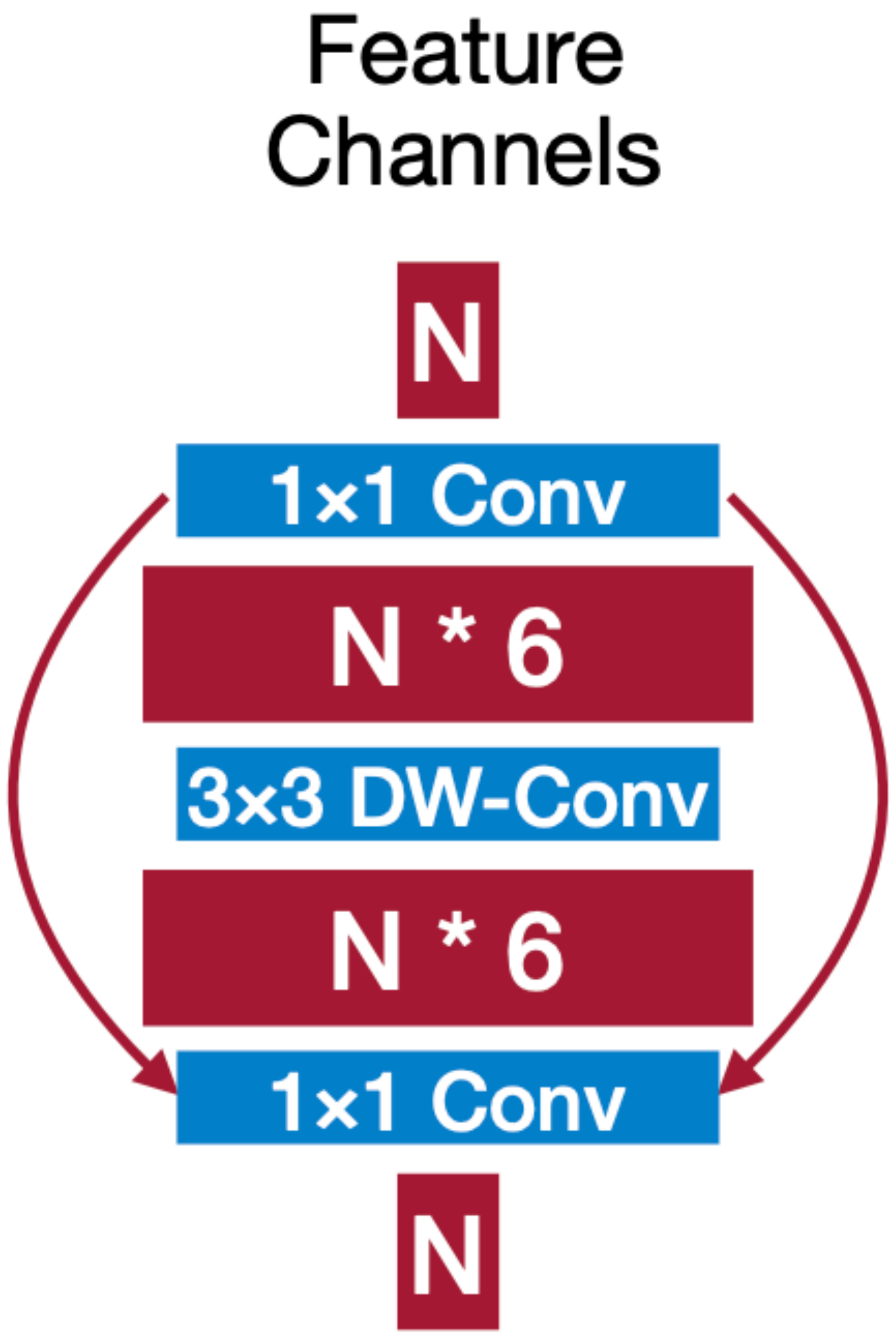
- Combine bottleneck with grouped convolution
  - Equivalent to a multi-path block

Xie et al., "Aggregated Residual Transformations for Deep Neural Networks," CVPR 2017

MobileNetv2: Inverted Bottleneck

- Increase #channels → Depthwise convolution → Decrease #Channels
  - Works better than simple depthwise + 1x1 without channel inflation
  - **Drawback.** Much activation memory

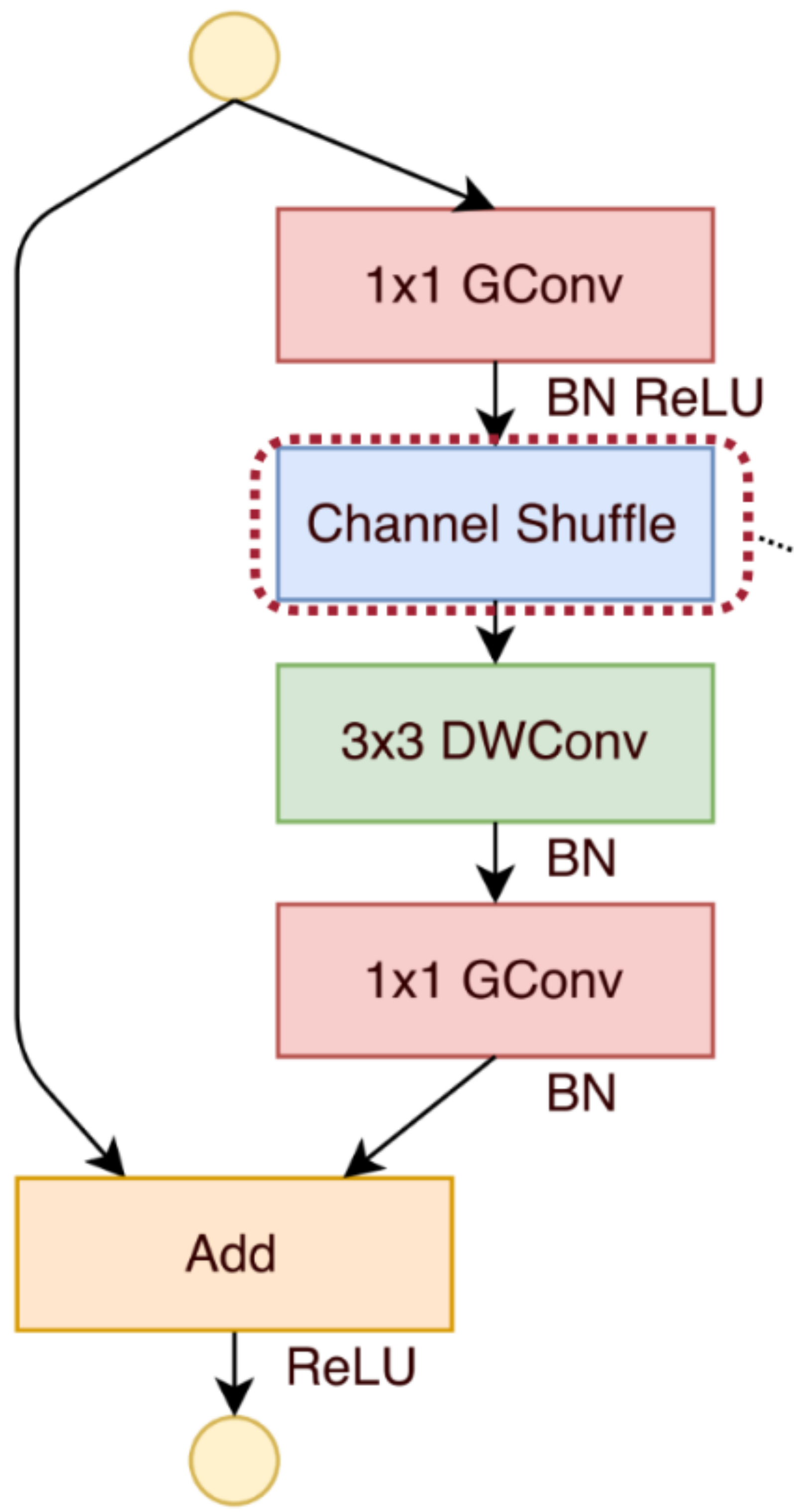


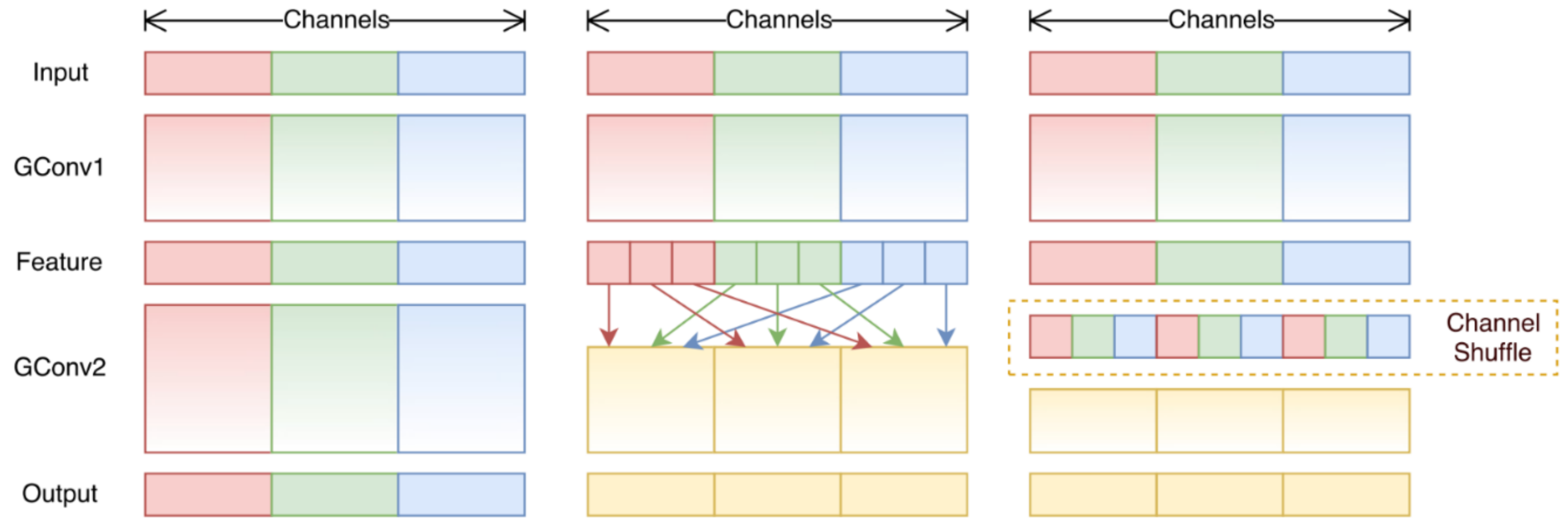


Sandler et al., "MobileNetv2: Inverted residuals and linear bottlenecks," CVPR 2018

ShuffleNet: Inverted Bottleneck

- Replace 1x1 conv with 1x1 grouped convolution
  - Then, do channel shuffling

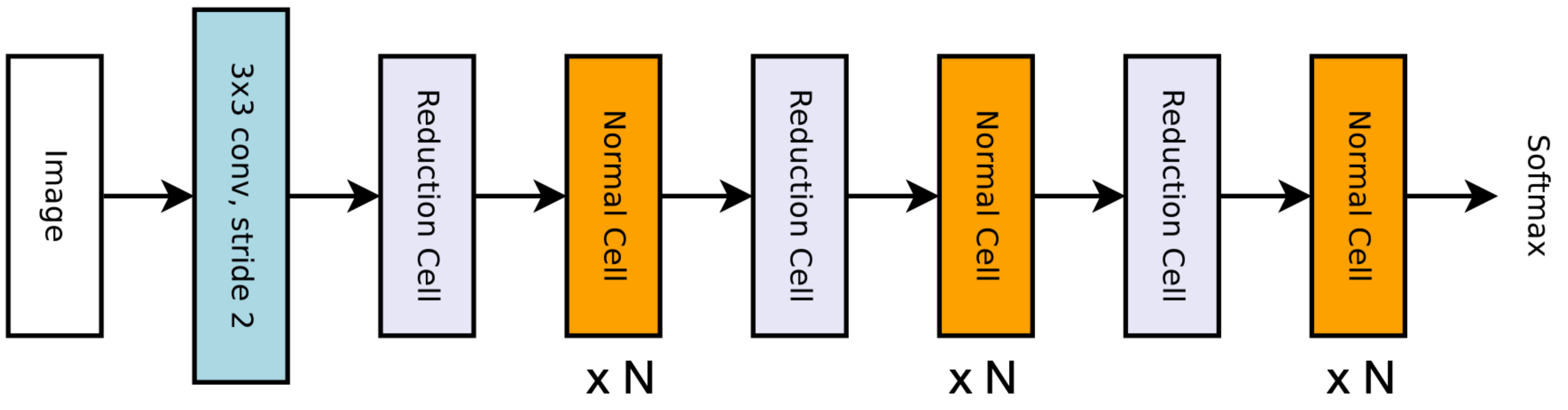




Zhang et al., "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," CVPR 2018

Building the search space

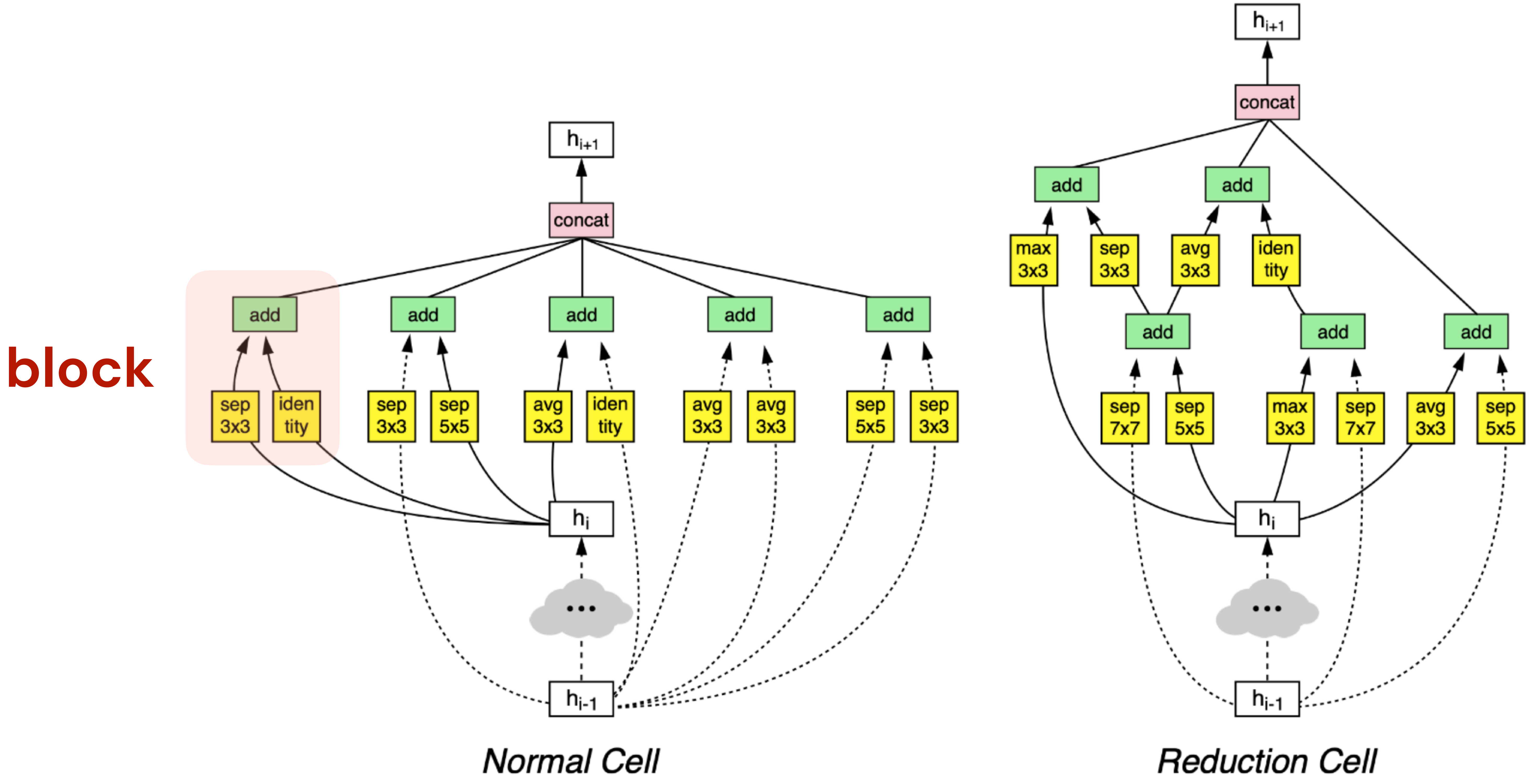
- For image-processing units, typically use **cell-based representation**
  - We describe the approach in NASNet
- A net consists of repeated cells + reduction cells (downsampling)
  - Inspired by successful models
  - Reduced search space



Zoph et al., "Learning transferable architectures for scalable image recognition" CVPR 2018

Building the search space

- A cell consist of **multiple blocks**
  - Placed in parallel, or in series
  - Example. NASNet searched on CIFAR-10, set to have five blocks

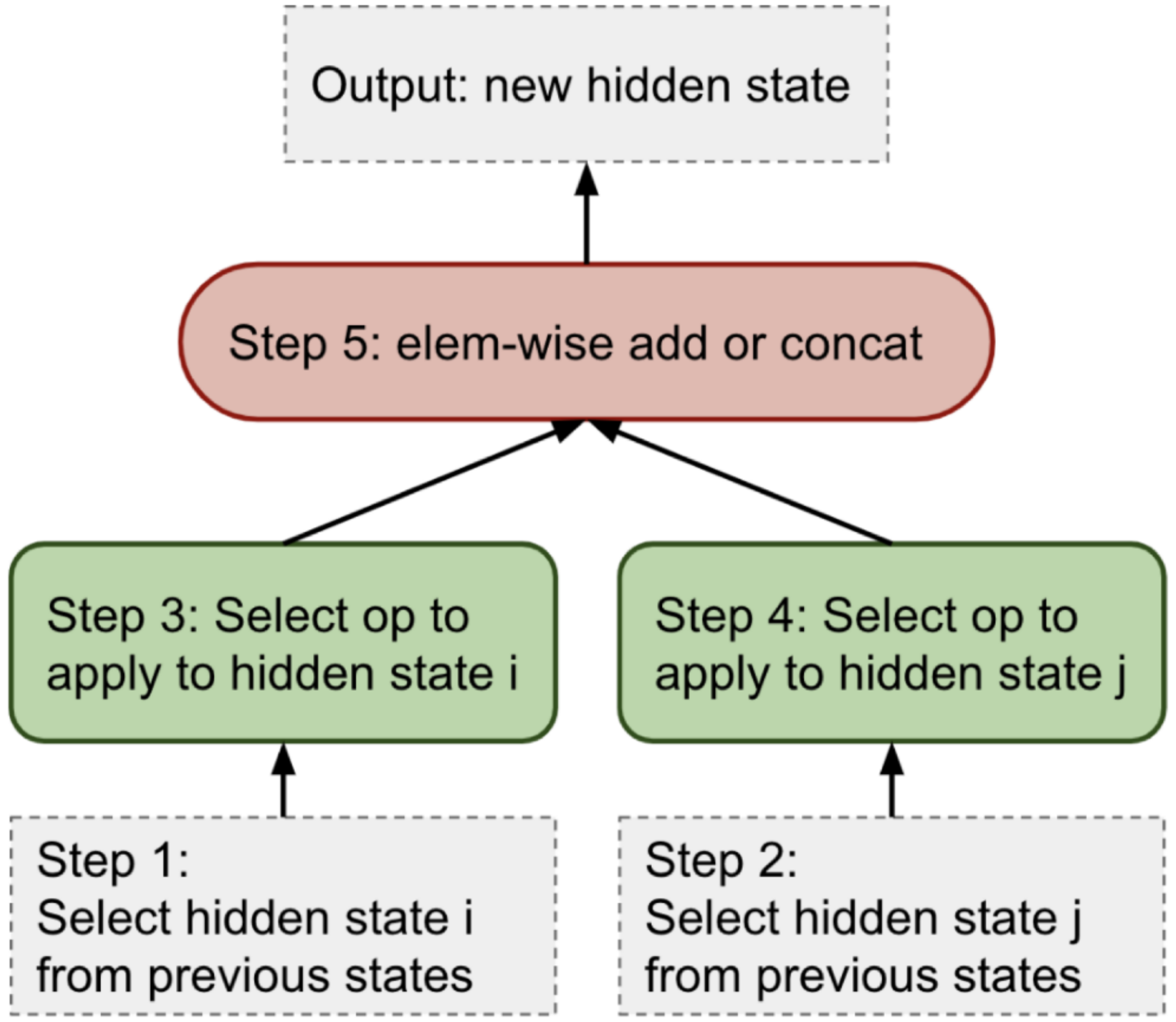


Zoph et al., "Learning transferable architectures for scalable image recognition" CVPR 2018

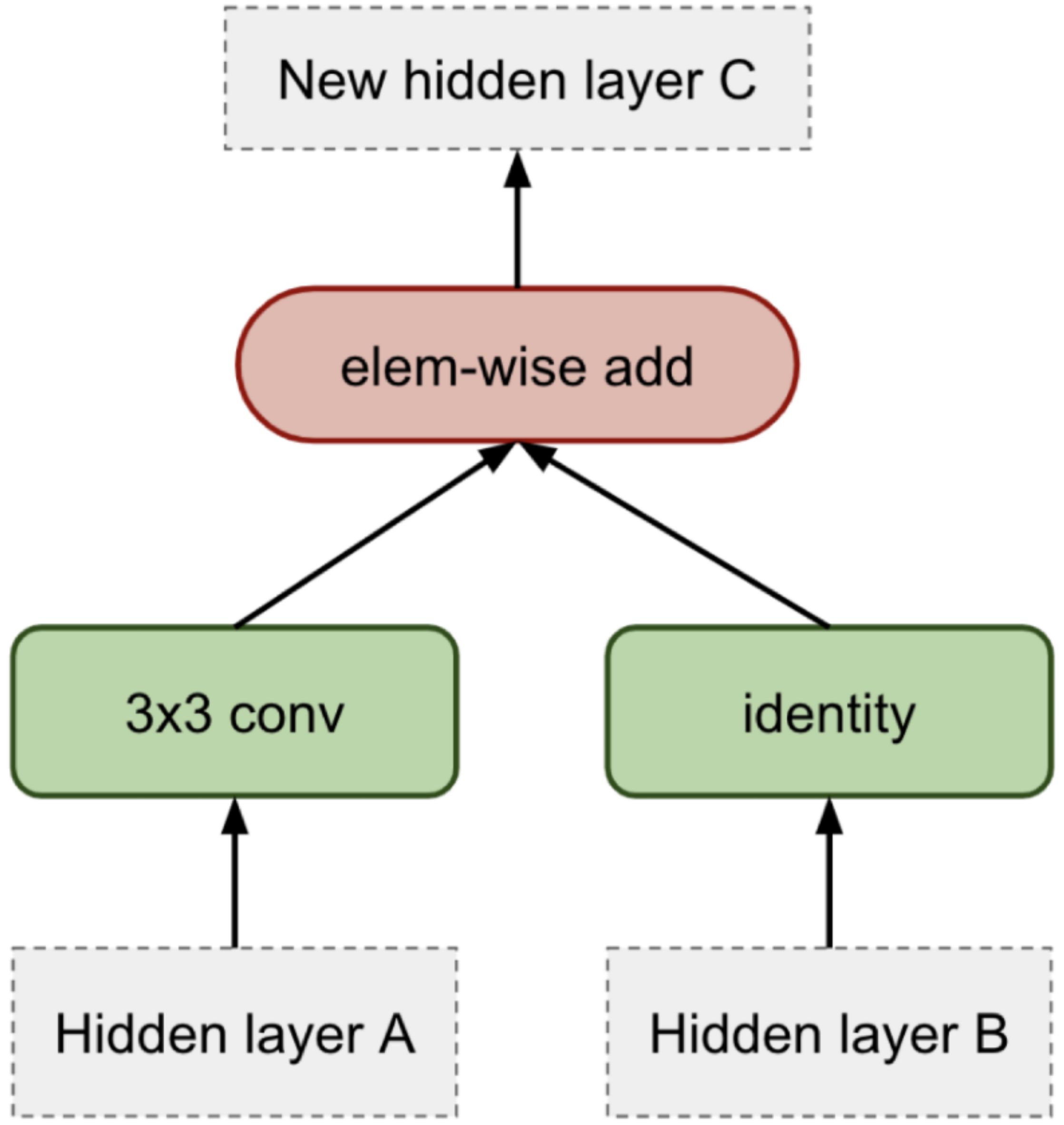
Building the search space

- A block consist of **five discrete choices**
  - Select inputs, process, aggregate
  - Processing ops are pre-handpicked (⇒)

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv



(a) 5 discrete choices in each block



(b) A concrete example

Zoph et al., "Learning transferable architectures for scalable image recognition" CVPR 2018

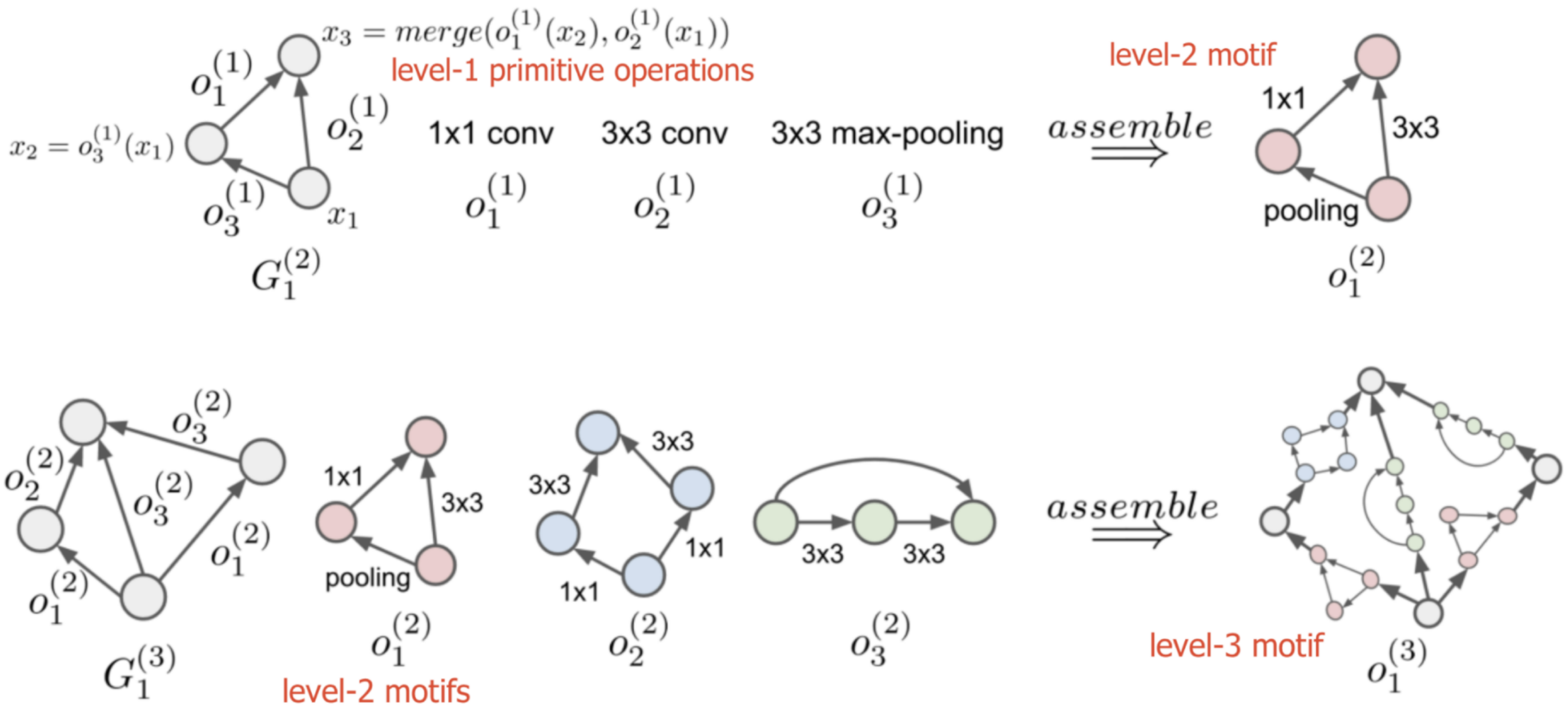
Extending the search space

- Note that we made several arbitrary choices:
  - Number of cell repetitions
  - Kernel size
  - Degrees of downsampling
  - Input resolution
  - (…)
- These were also searched in later works
  - e.g., MNASNet, RegNet, ProxylessNAS, OFANet

Zoph et al., “Learning transferable architectures for scalable image recognition” CVPR 2018

Extending the search space

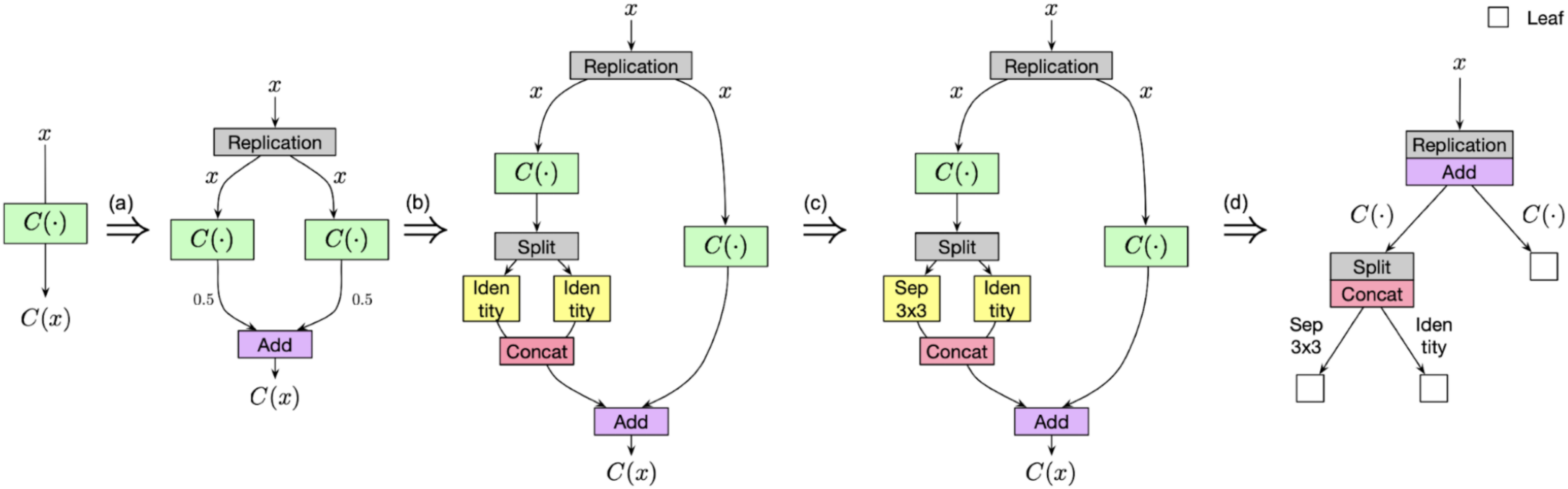
- Some works proposed more complicated structures than repeated cells
  - <u>Example</u>. Hierarchical NAS



Liu et al., "Hierarchical representations for efficient architecture search" ICLR 2018

Extending the search space

- <u>Example</u>. Tree-like branching structures



Cai et al., "Path-level network transformations for efficient architecture search" ICML 2018

Search strategy

Searching

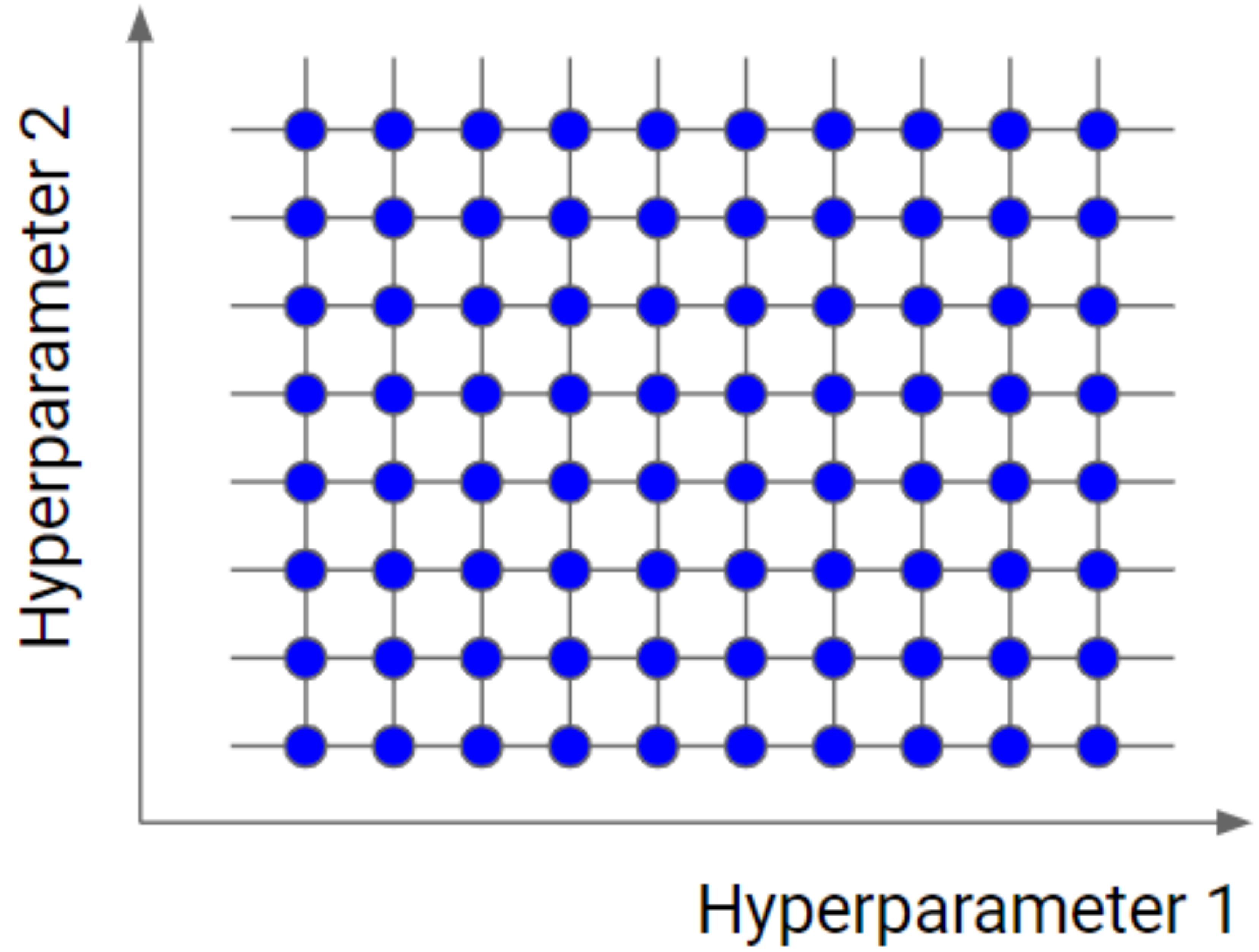
$$\min_{a \in \mathcal{A}} \quad \ell(\mathrm{a})$$

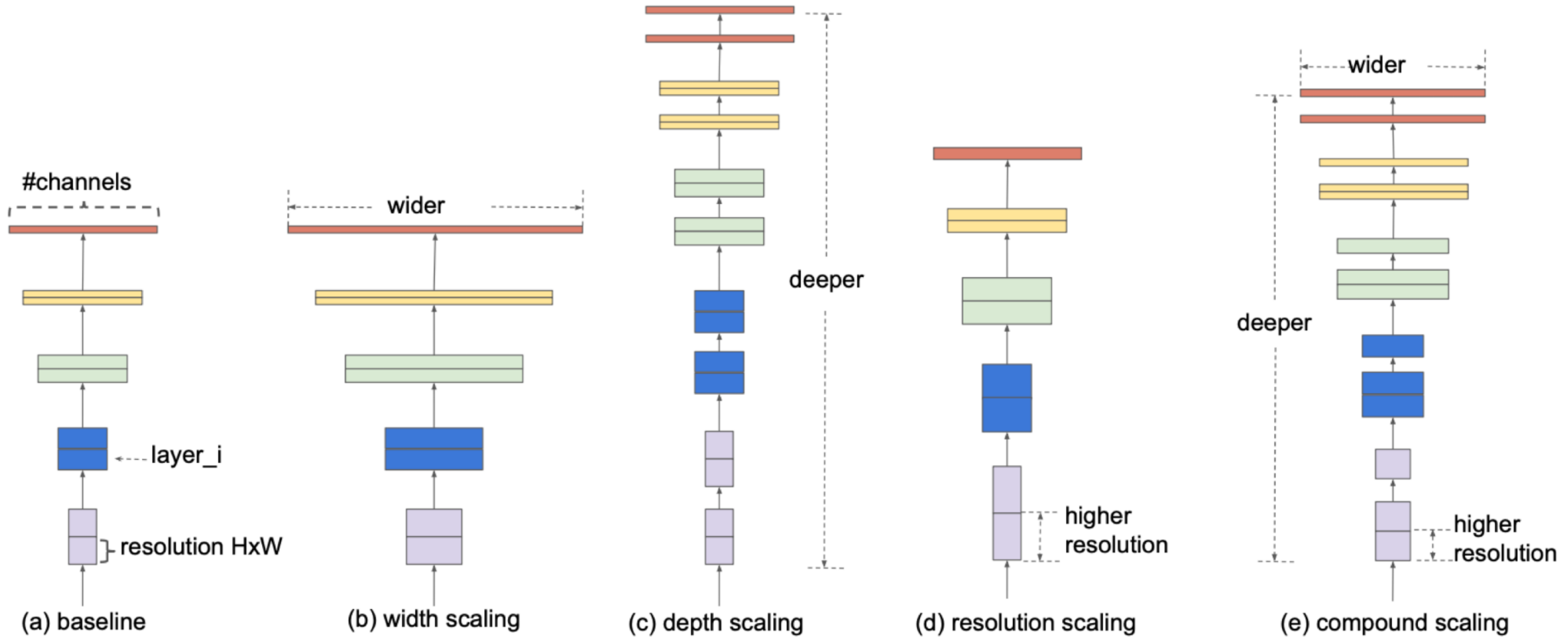
- **Problem.** Searching over **discrete** space
  - Grid / random search
  - Reinforcement learning
  - Evolutionary method
  - Progressive search

  - Differentiable options $\Leftarrow$ discussed in the next section

Grid Search

- Simply list all possible choices
  - Exponential growth in scale
  - Can be used for extremely simplified search space
    - e.g., EfficientNet

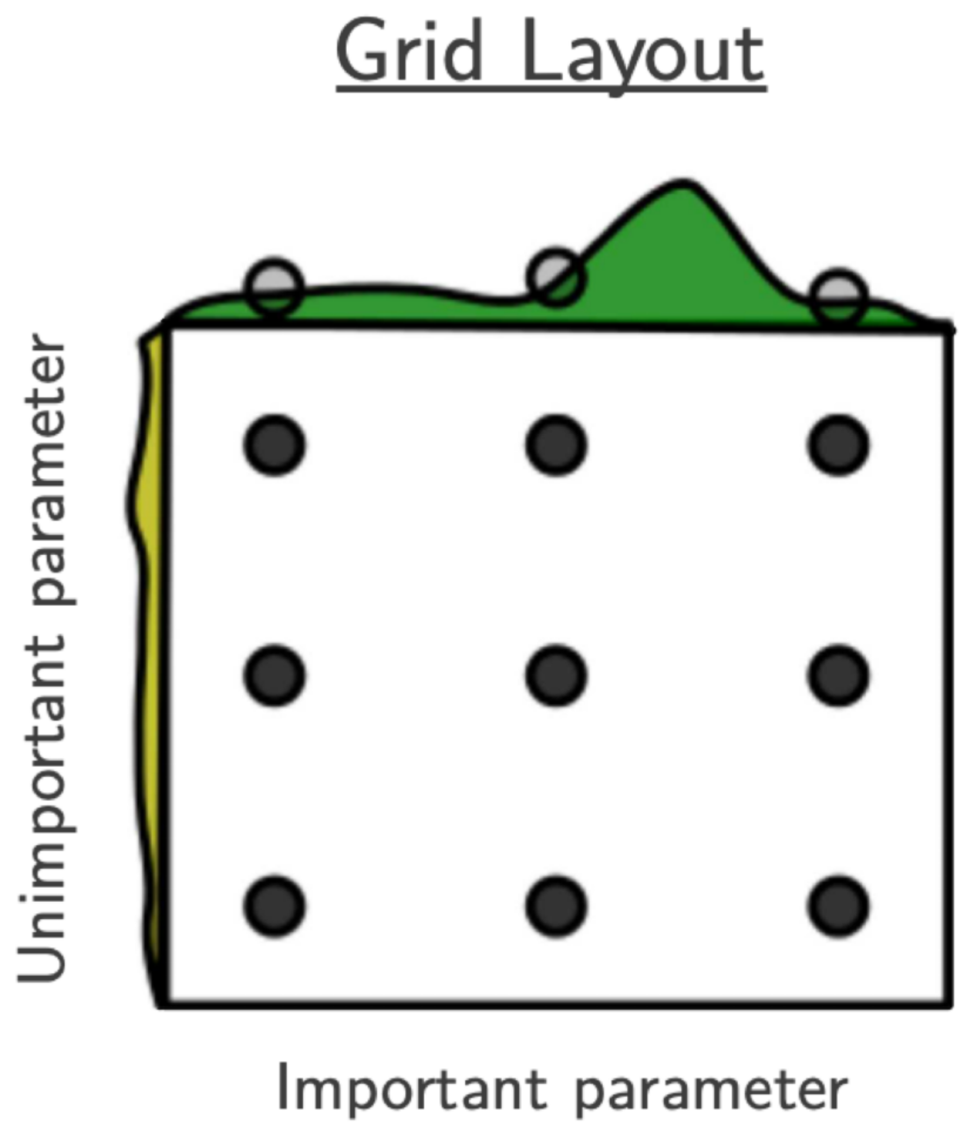


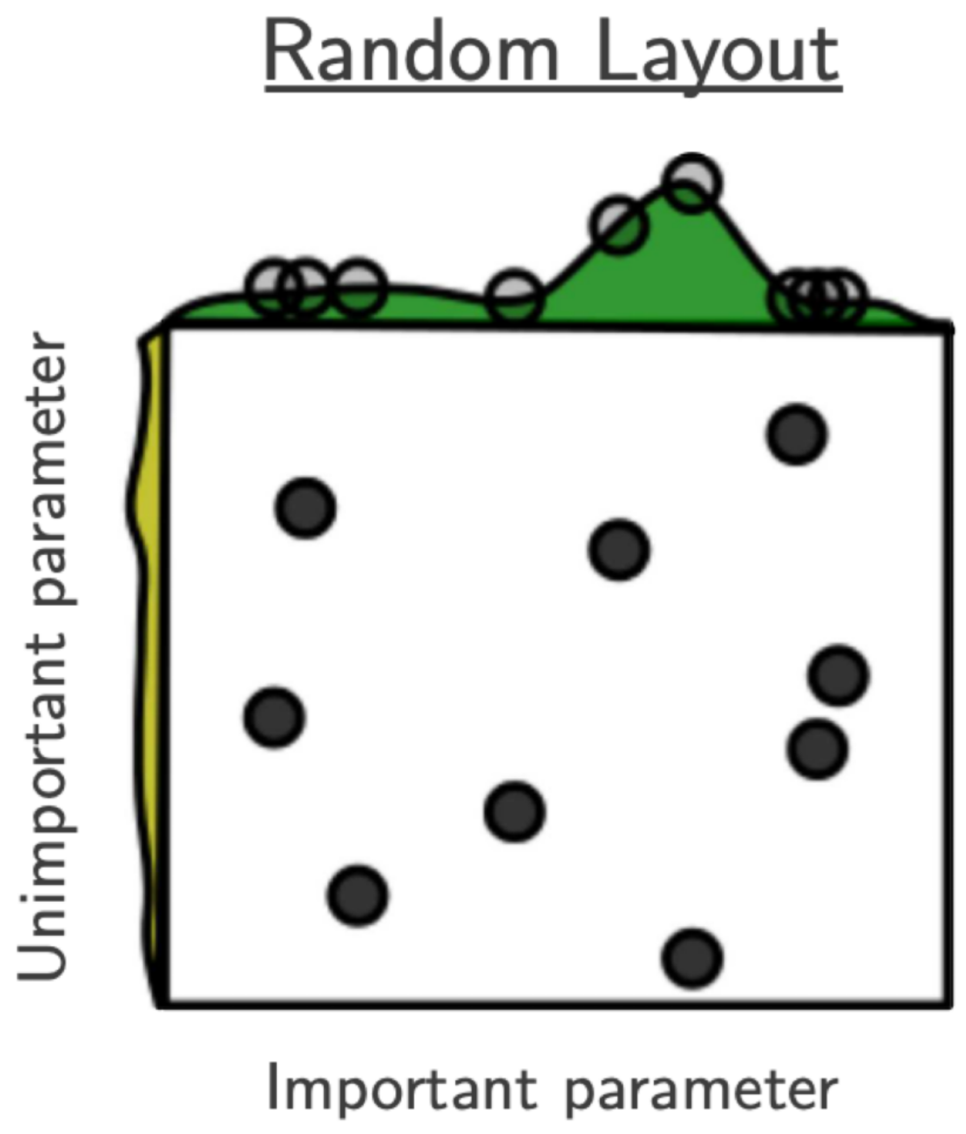


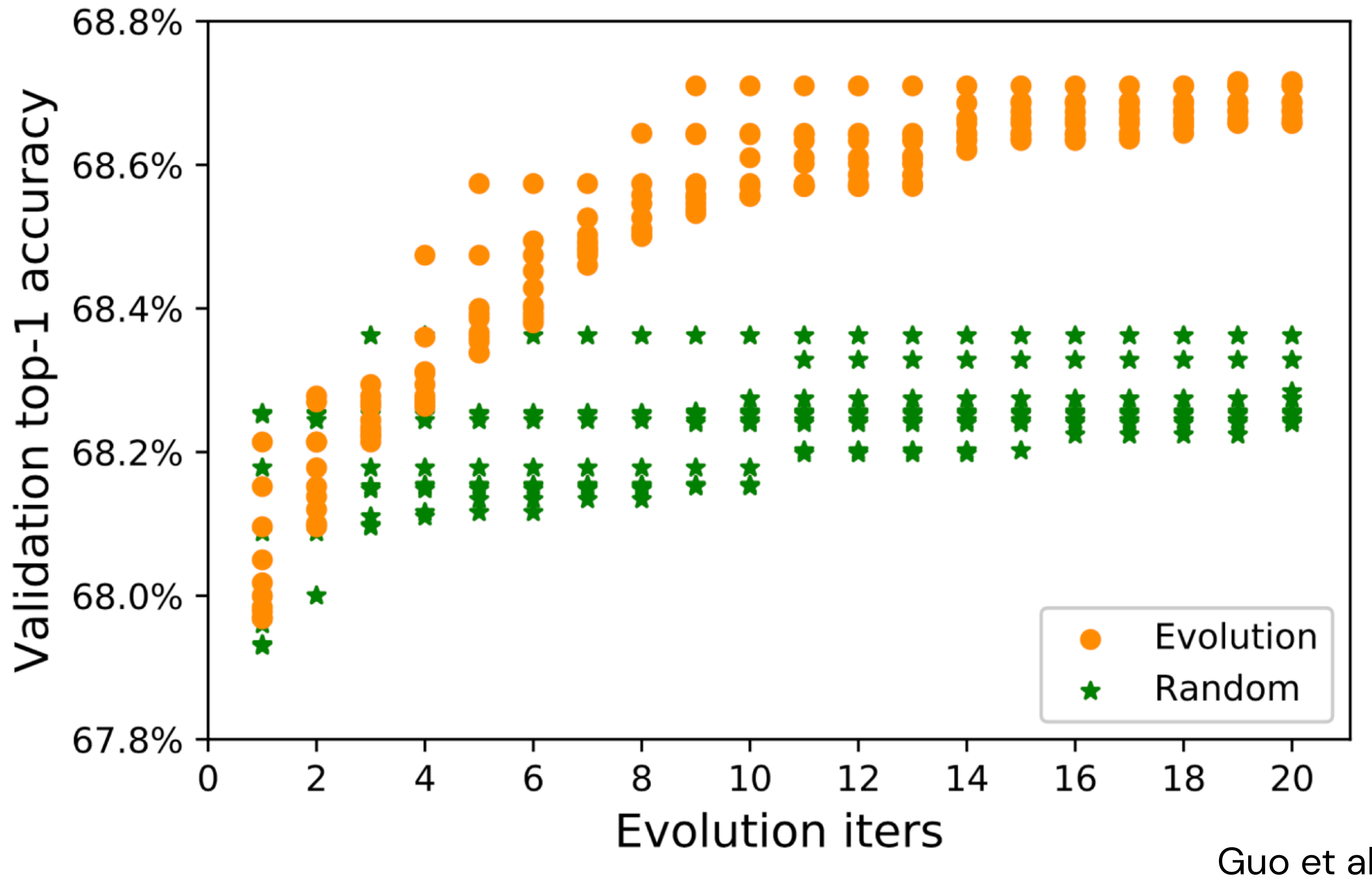
Liu et al., "Hierarchical representations for efficient architecture search" ICLR 2018

Random Search

- More effective utilization of trials
  - With grid search, effective number of samples is 3
  - Very simple, but effective



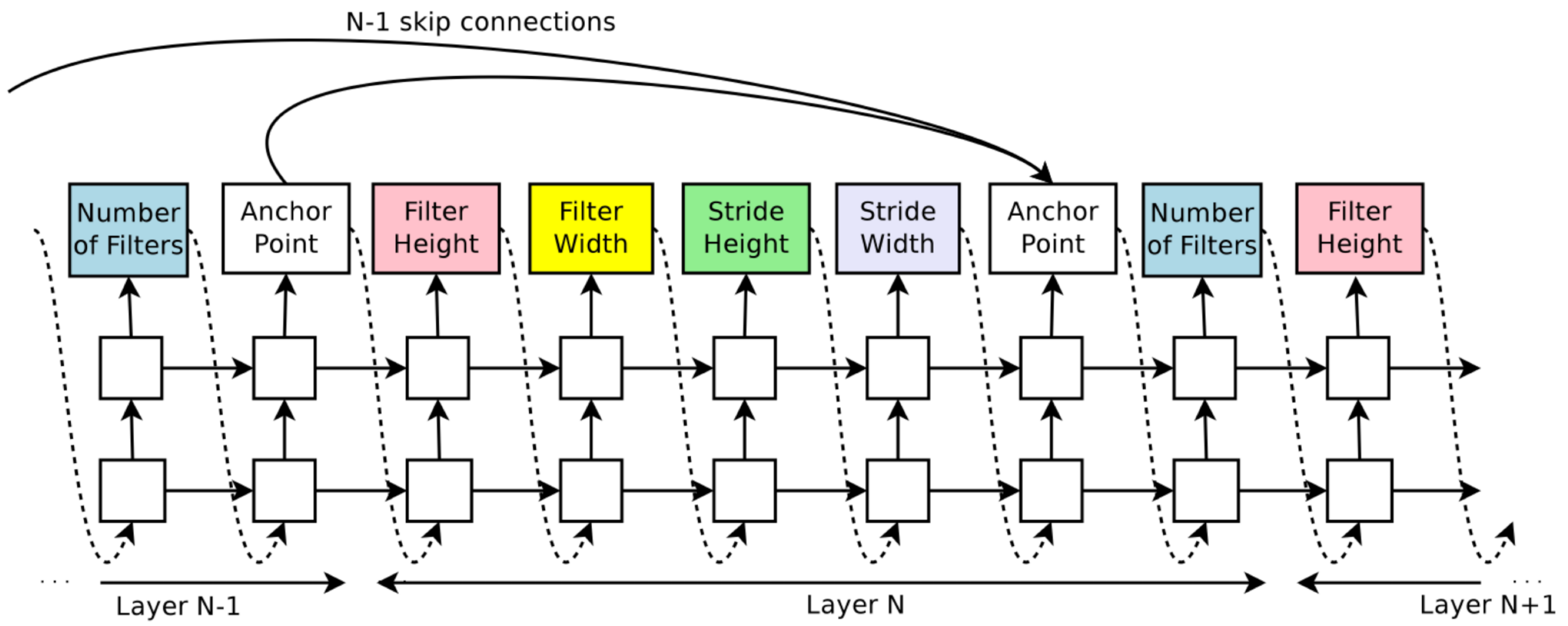




Guo et al. "Single Path One-Shot NAS with Uniform Sampling" ECCV 2020

Reinforcement learning

- Train a **policy** which generates hyperparameters sequentially
  - Update policy parameters, based on reward
  - <u>Example</u>. Zoph and Le (2017) uses an RNN controller



Zoph and Le, “NAS with RL,” ICLR 2017

Reinforcement learning

- **Training.** Evaluate policy gradient with respect to the **REINFORCE loss**
  (later works used PPO, Q Learning, MTCS, ...)
  - Given the model parameter θ, RNN generates HPs with distribution

$$p_\theta(a_{1:T})$$

  - Want-to-do. Maximize the expected reward

$$J(\theta) = \mathbb{E}_{p_\theta}[R]$$

    - R is the validation accuracy of the model configured by $a_{1:T}$

Zoph and Le, "NAS with RL," ICLR 2017

Reinforcement Learning

- Update RNN controllers using the gradient:

$$\nabla_\theta J(\theta) = \nabla_\theta \int \left(\prod_{t=1}^{T} p_\theta(a_t \mid a_{1:t-1}) \cdot R \right) \mathrm{d}a_{1:T}$$

$$= \int \left(\sum_{t=1}^{T} \frac{\nabla_\theta p_\theta(a_t \mid a_{1:t-1})}{p_\theta(a_t \mid a_{1:t-1})} \cdot p_\theta(a_{1:T}) \cdot R \right) \mathrm{d}a_{1:T}$$

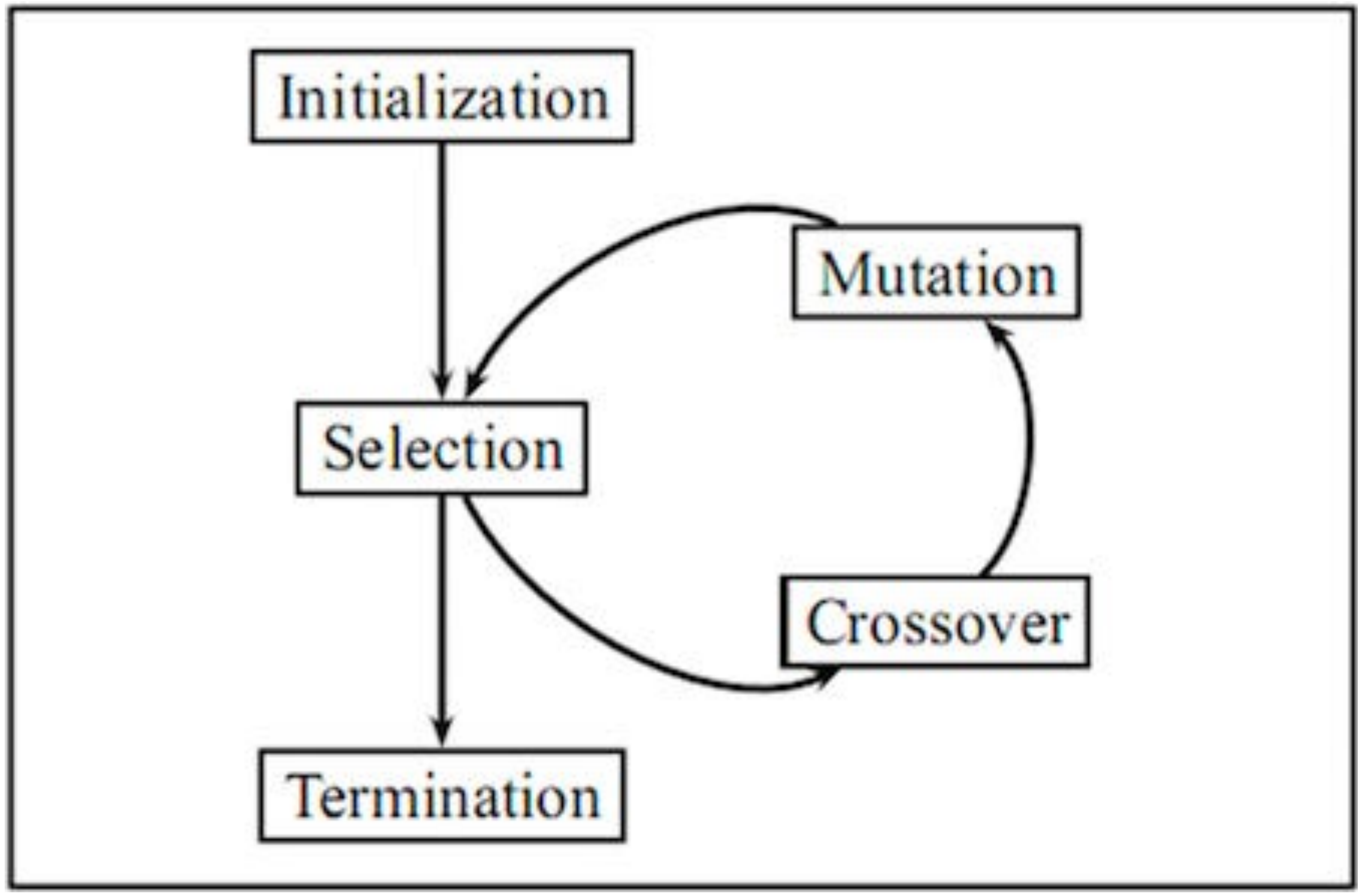
$$= \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log p_\theta(a_t \mid a_{1:t-1}) \cdot R]$$

  - If R was high, strong positive feedback to generate similar HPs
  - If R was low, weak positive feedback (thus called "reinforce," not penalize)

Williams, "Simple statistical gradient-following algorithms for connectionist RL" Machine Learning, 1992

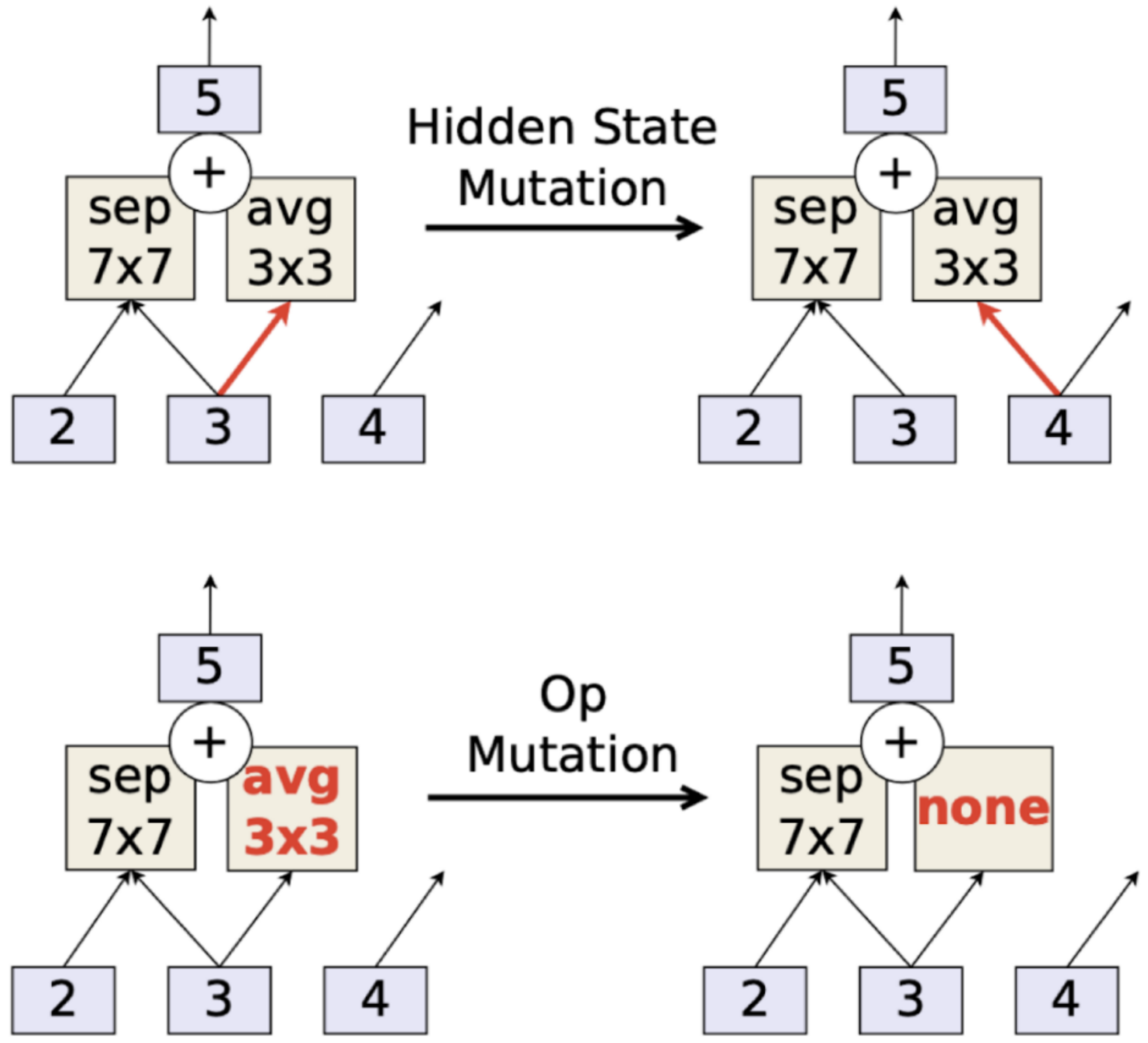
Evolutionary method

- Do the following:
  - Start from a set of solutions
  - Repeat:
    - Pick a solution
    - Randomly mutate it
      - If good, add it to population (optionally, remove one)



Evolutionary method

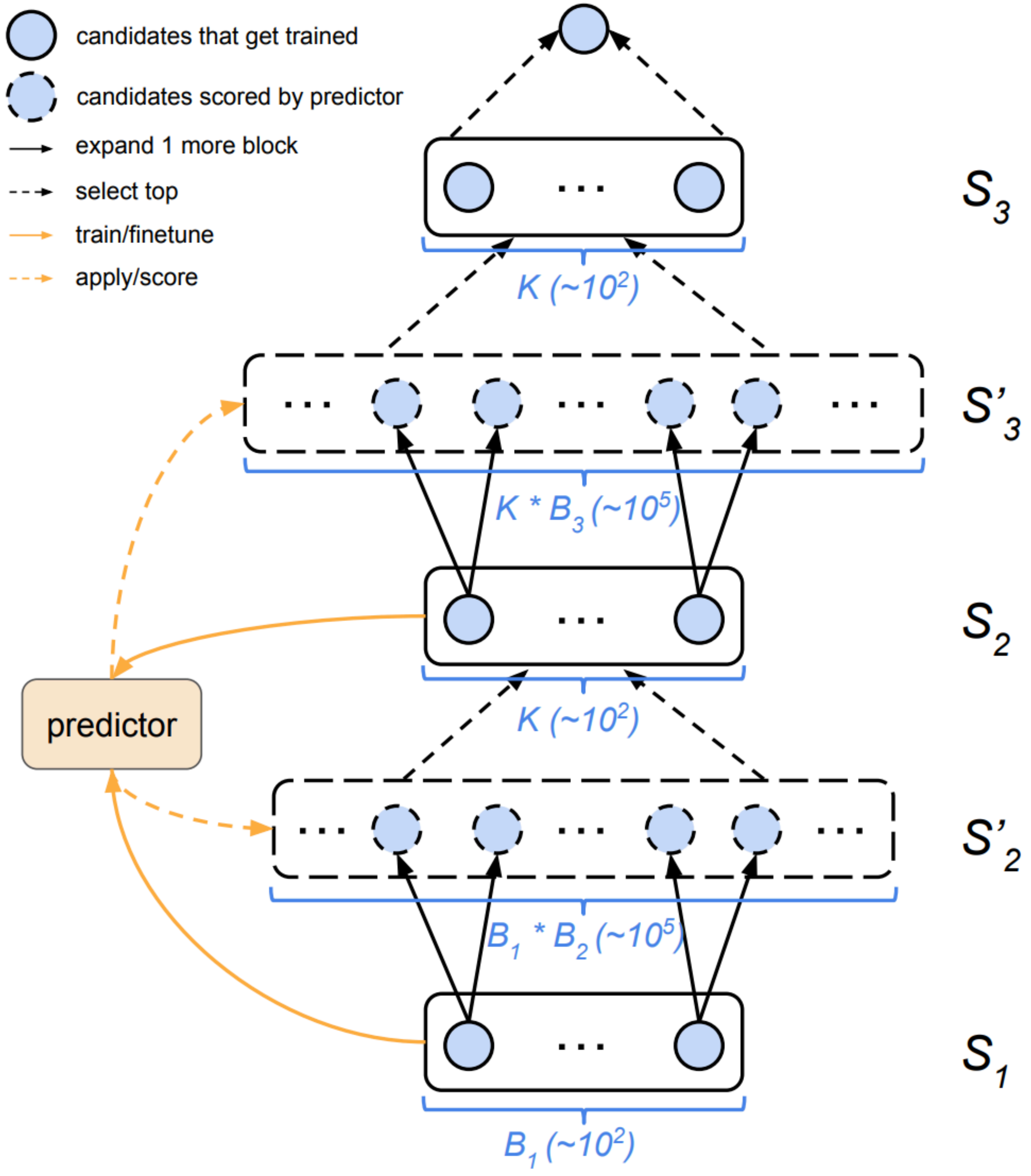
- Example. AmoebaNet
- Uses **tournament selection**
  - Sample S models from the population
    - Pick highest acc. model as parent
  - Mutate parent to get a child
  - Train child and evaluate
  - Add child to the population



Real et al., "Regularized Evolution for Image Classifier Architecture Search" AAAI 2019

Progressive search

- These ideas are often combined with progressive search
- Example. Progressive NAS
    - Search for 1-Block cells
    - Select top-k cells
    - Add one block to top-K cells
    - (repeat)



Liu et al., "Progressive NAS" ECCV 2018

Evaluation strategy

Evaluation strategy

$$\min_{a \in \mathcal{A}} \quad \ell(\mathrm{a})$$

- **Idea.** Use a cheaper proxy
    - Smaller duration (less epochs)
    - Smaller data (less #data, less resolution)
    - Smaller model (less #channels, less repeated blocks)

    - Re-use trained weights
    - Joint training

Shorter training

- **Problem.** Simply selecting the best solution may not be good enough
    - Poor correlation with final accuracy

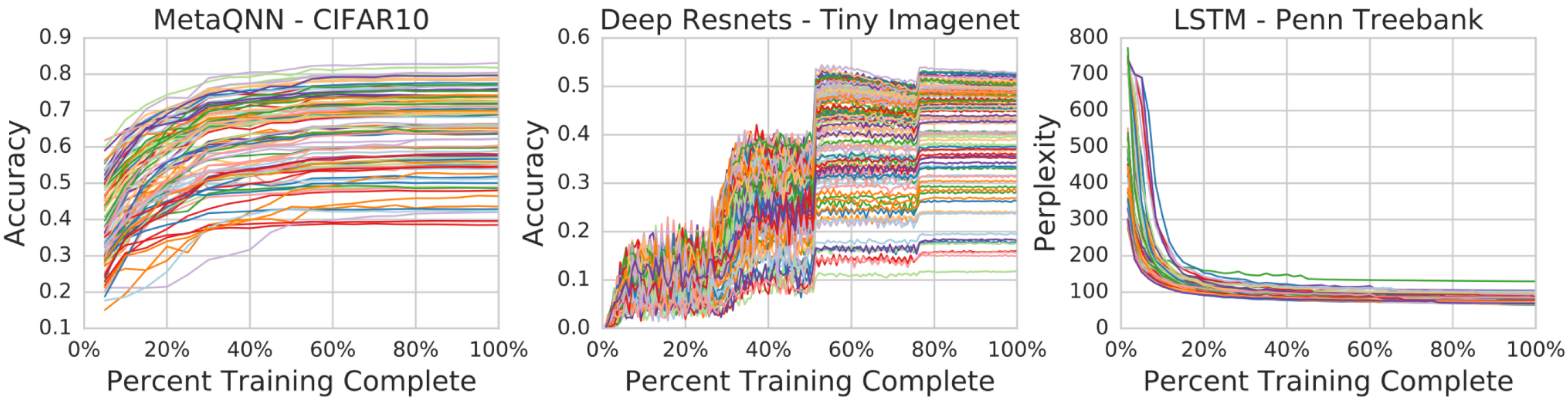
Table 1: Spearman rank correlation coefficients of the validation errors between different budgets. The correlation is high between every budget and the next larger one, but degrades quickly beyond that.

	1200s	1h	3h
400s	0.87	0.31	0.05
1200s		0.88	0.64
1h			0.86

Zela et al., "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," ICML workshop 2018

Shorter training

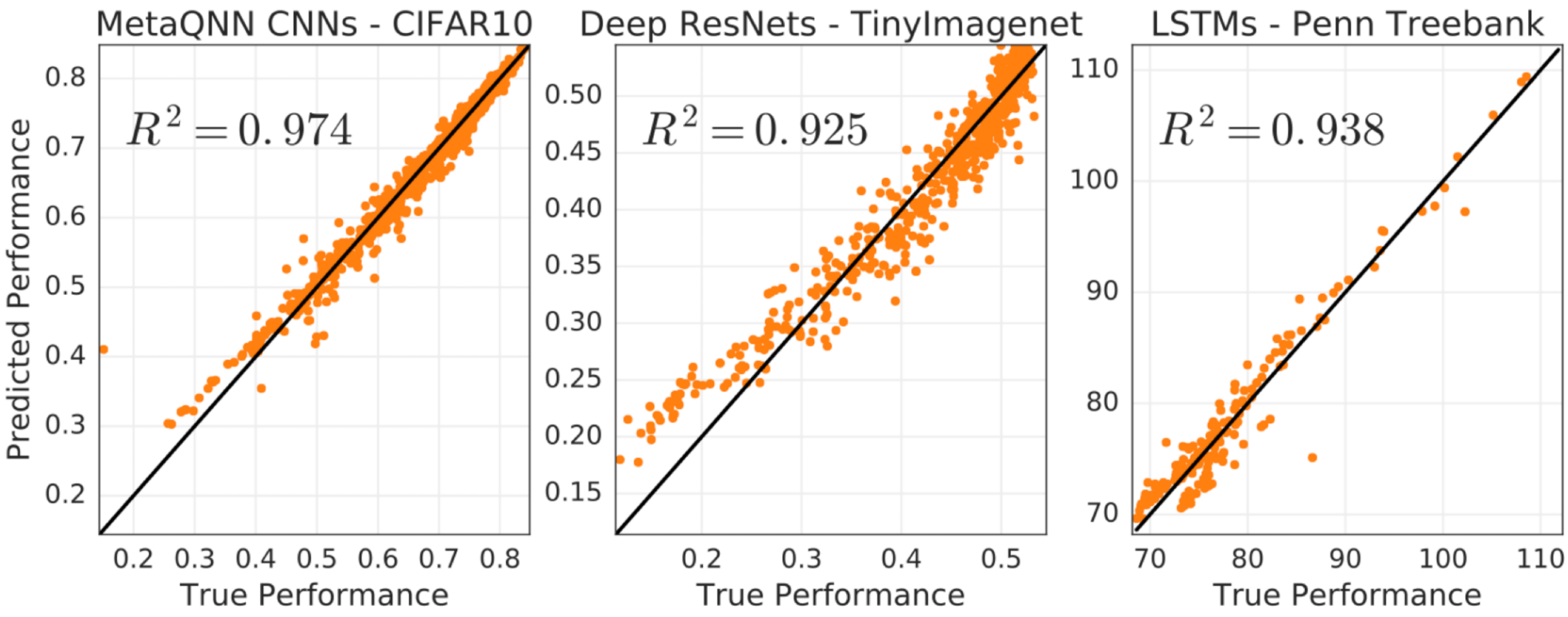
- **Solution.** Train a loss predictor
  - Example. Baker et al. (2018) observes that models have similar loss curves



Baker et al., "Accelerating NAS using performance prediction," ICLR 2018

Shorter training

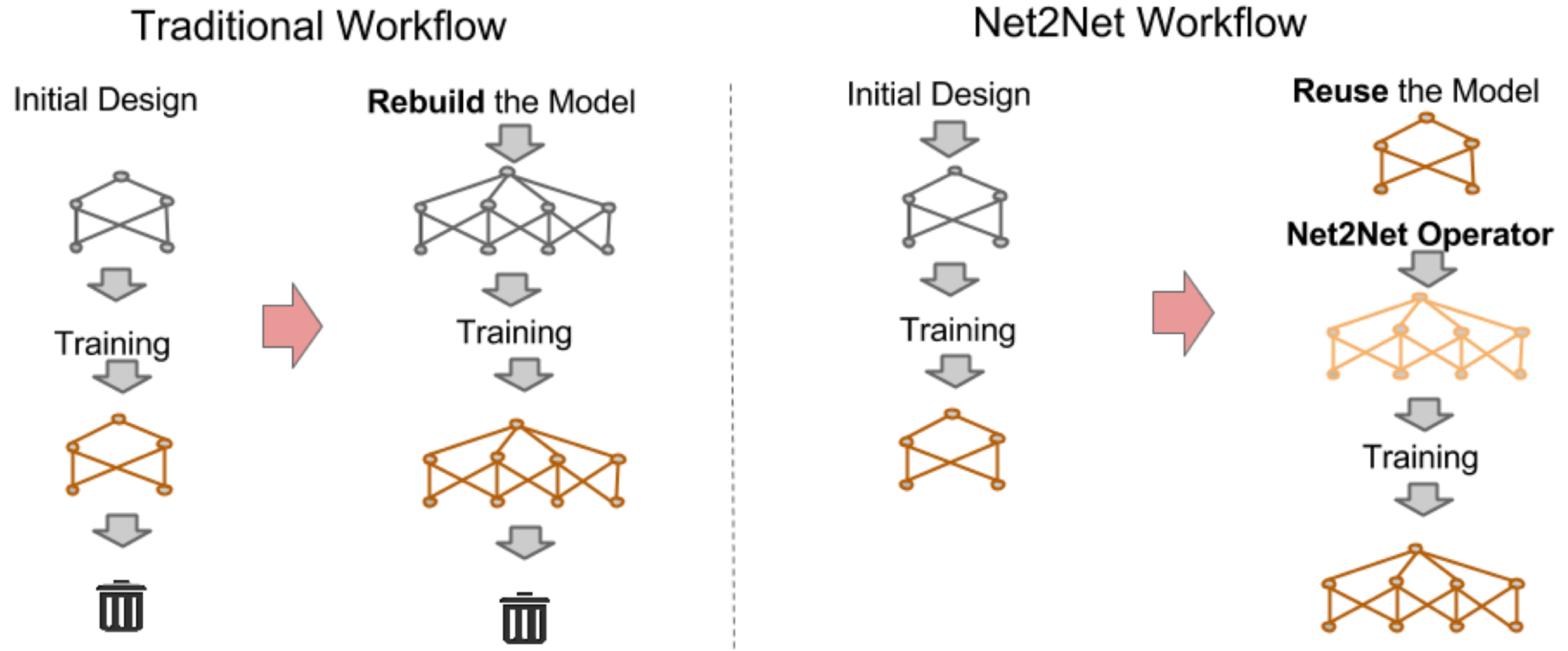
- Baker et al. (2018) used ν-SVR to predict the full curve from the early 25%



Baker et al., "Accelerating NAS using performance prediction," ICLR 2018

Training re-use

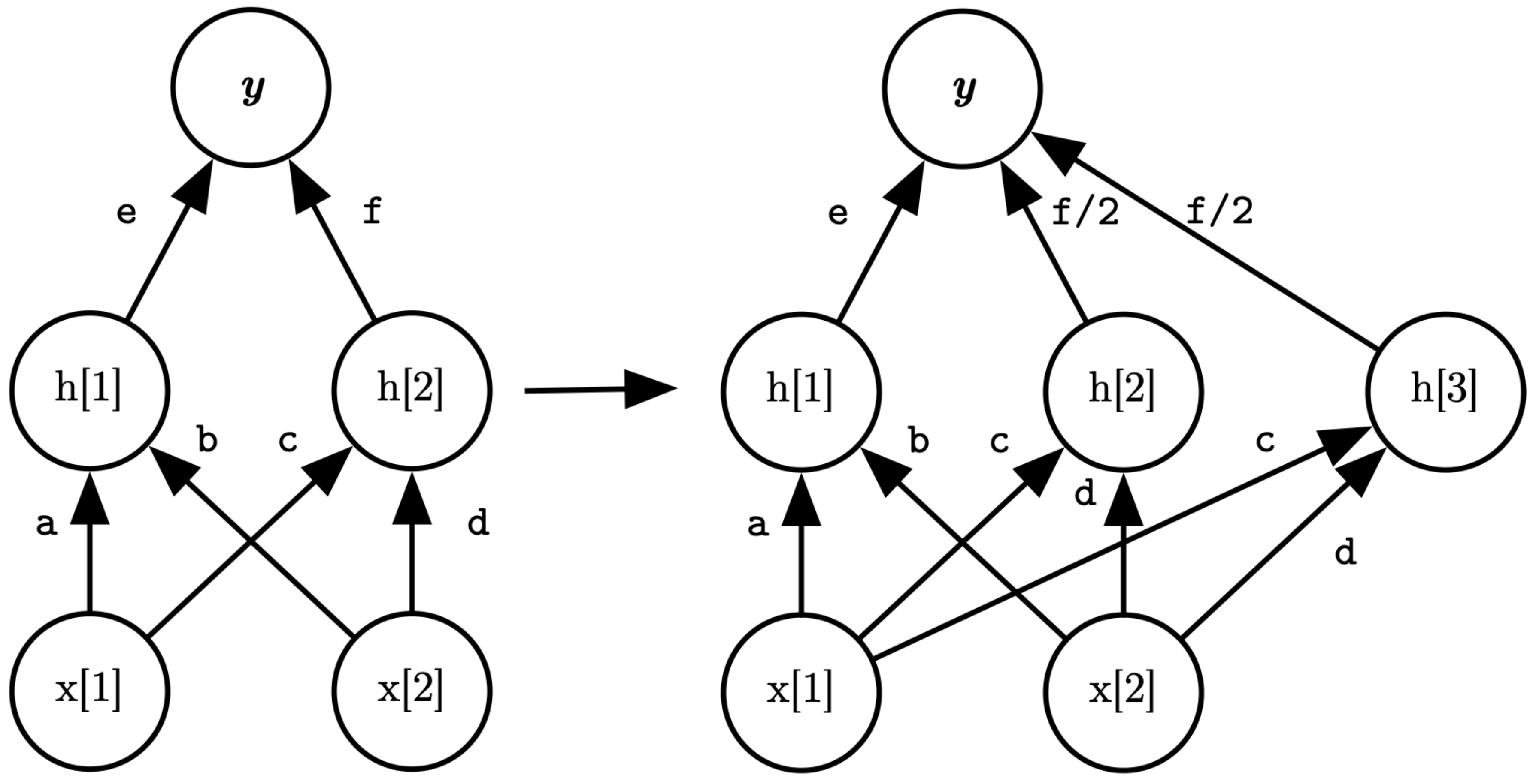
- **Idea.** Reuse the weights trained from prior runs
  - Related. Net2Net (2016) transfers weights to other tasks for adaptation

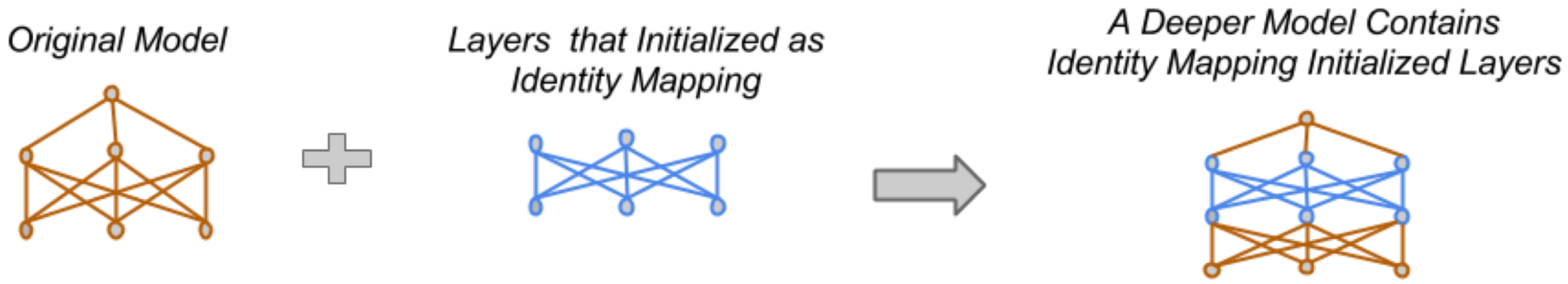


Chen et al., "Net2Net: Accelerating learning via knowledge transfer," ICLR 2016

Training re-use

- **Expanding width.** Distribute weights by half
- **Expanding depth.** Identity function





Chen et al., "Net2Net: Accelerating learning via knowledge transfer," ICLR 2016

Training re-use

- **App. to NAS.** EfficientNAS views NAS as finding a subgraph of a giant net
  - Update weights with SGD
  - Select subgraph with RNN

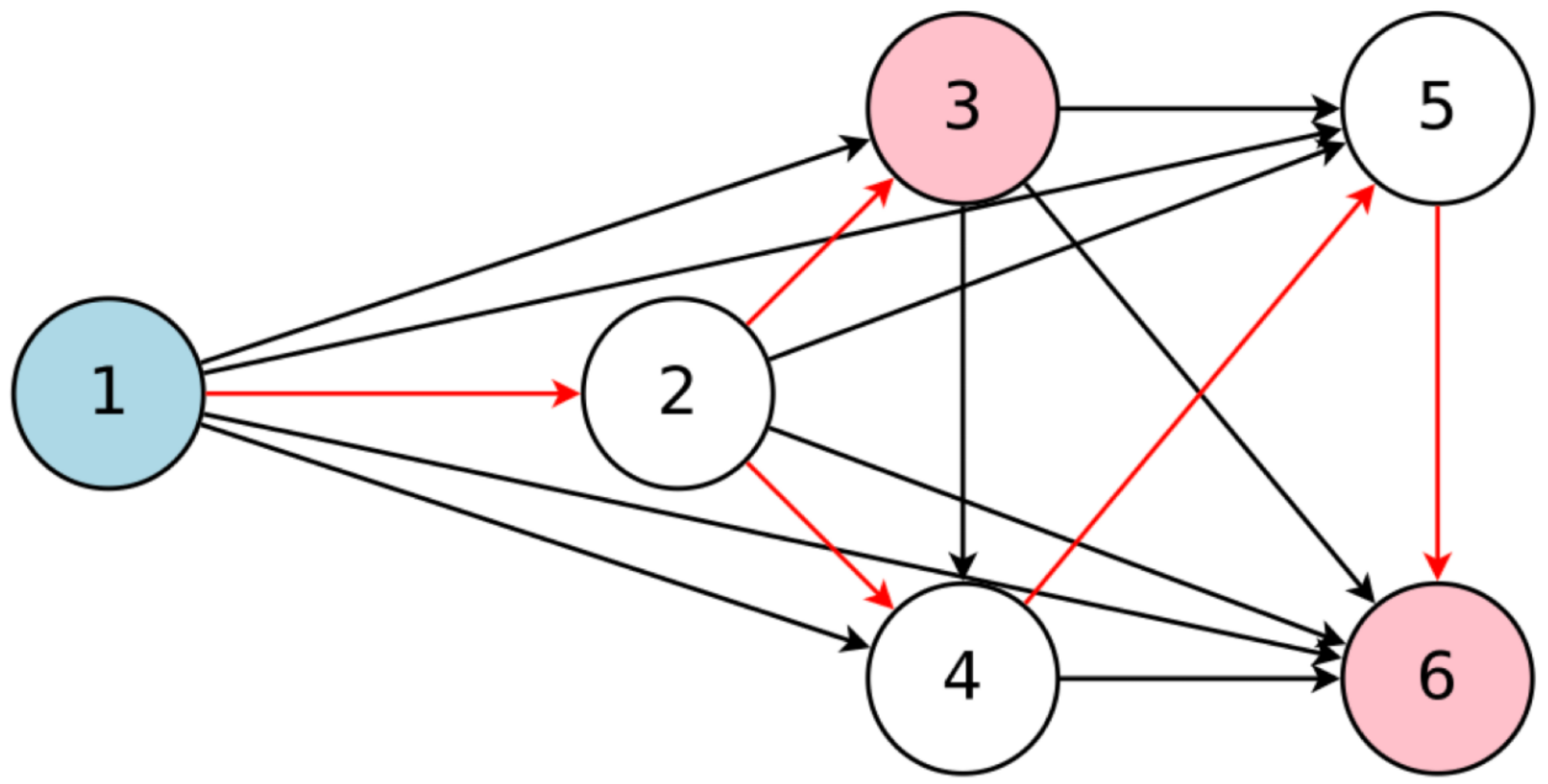
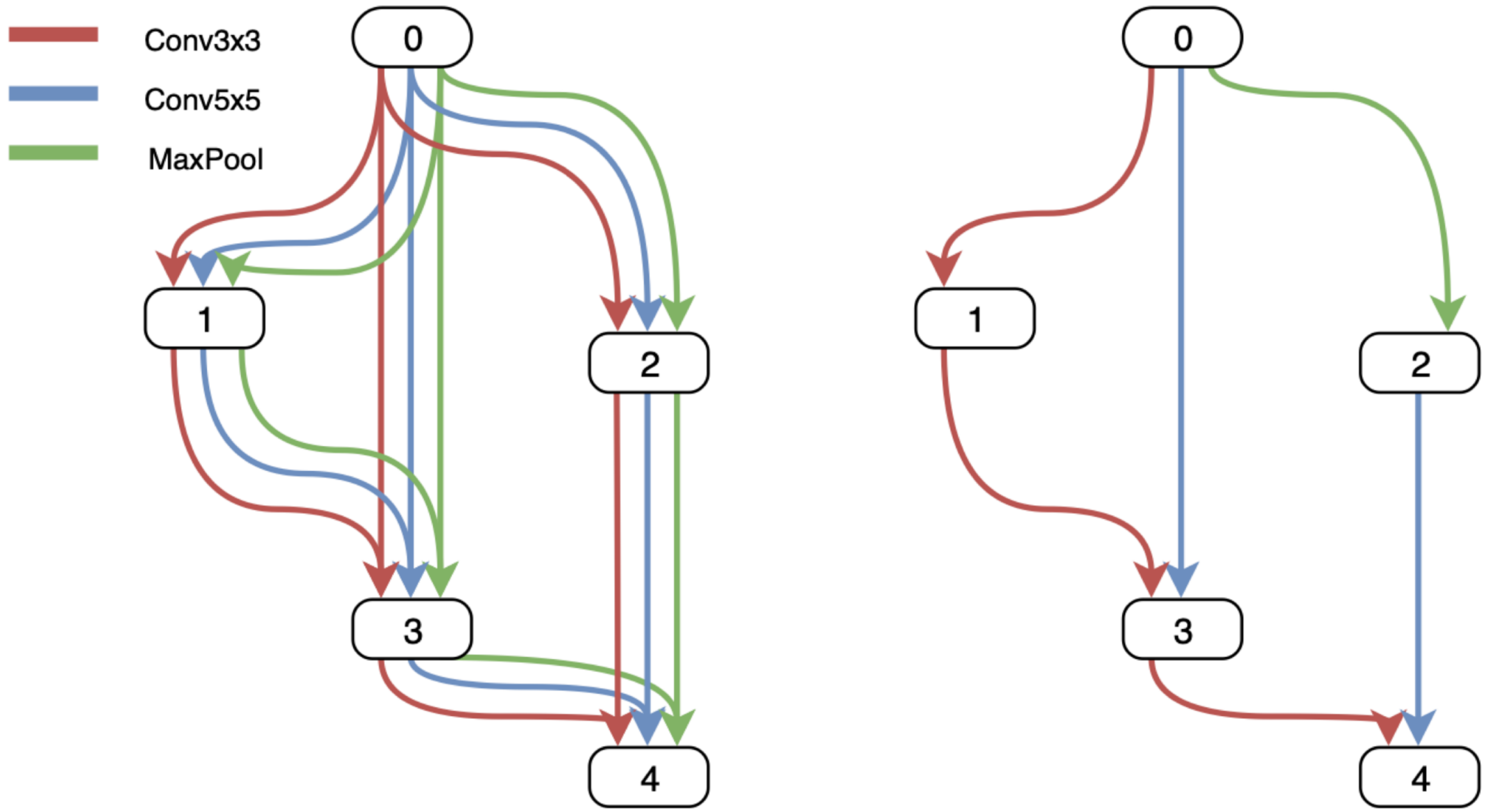


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

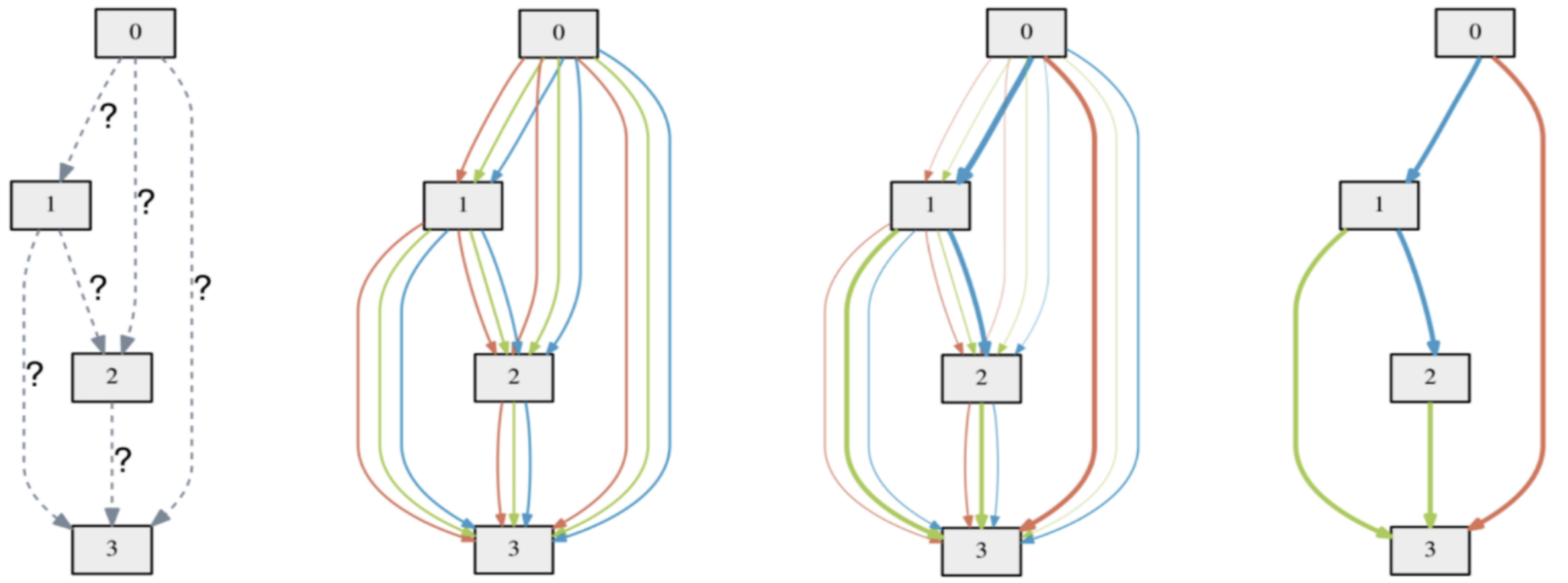
Pham et al., "Efficient NAS via parameter sharing," arXiv 2018

Joint training

- We can use differentiable relaxation for optimizing the connectivity as well
  - Example. DARTS uses GD for finding the subgraph as well
    - Intermediate features are connected with a mixture of modules



Liu et al., "DARTS: Differentiable architecture search," ICLR 2019



(a) Initially unknown operations on the edges.

(b) Continuous relaxation by placing a mixture of operations on each edge.

(c) Bilevel optimization to jointly train mixing probabilities and weights.

(d) Finalized the model based on the learned mixing probabilities.

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$$

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

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)
while *not converged* **do**
1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$
($\xi = 0$ if using first-order approximation)
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned α.

Liu et al., "DARTS: Differentiable architecture search," ICLR 2019

Further reading

- Zero-shot NAS
  - Mellor et al., “NAS without training” ICML 2021
  - Abdelfattah et al., “Zero-cost proxies for lightweight NAS” ICLR 2021
- Efficiency-aware NAS
  - ProxylessNAS: Use “latency” as a reward as well
  - MobileNAS: Construct search space with efficient modules
  - MCUNet: Maximize FLOPs for better memory-accuracy tradeoff
  - ChamNet: Train proxies for efficiency metrics

Zoph et al., “Learning transferable architectures for scalable image recognition” CVPR 2018

Next Class

- Efficient Training & Tuning

That’s it for today 🙌