EECE695D: Efficient ML Systems

Neural Architecture Search

(part 1)

Motivation

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

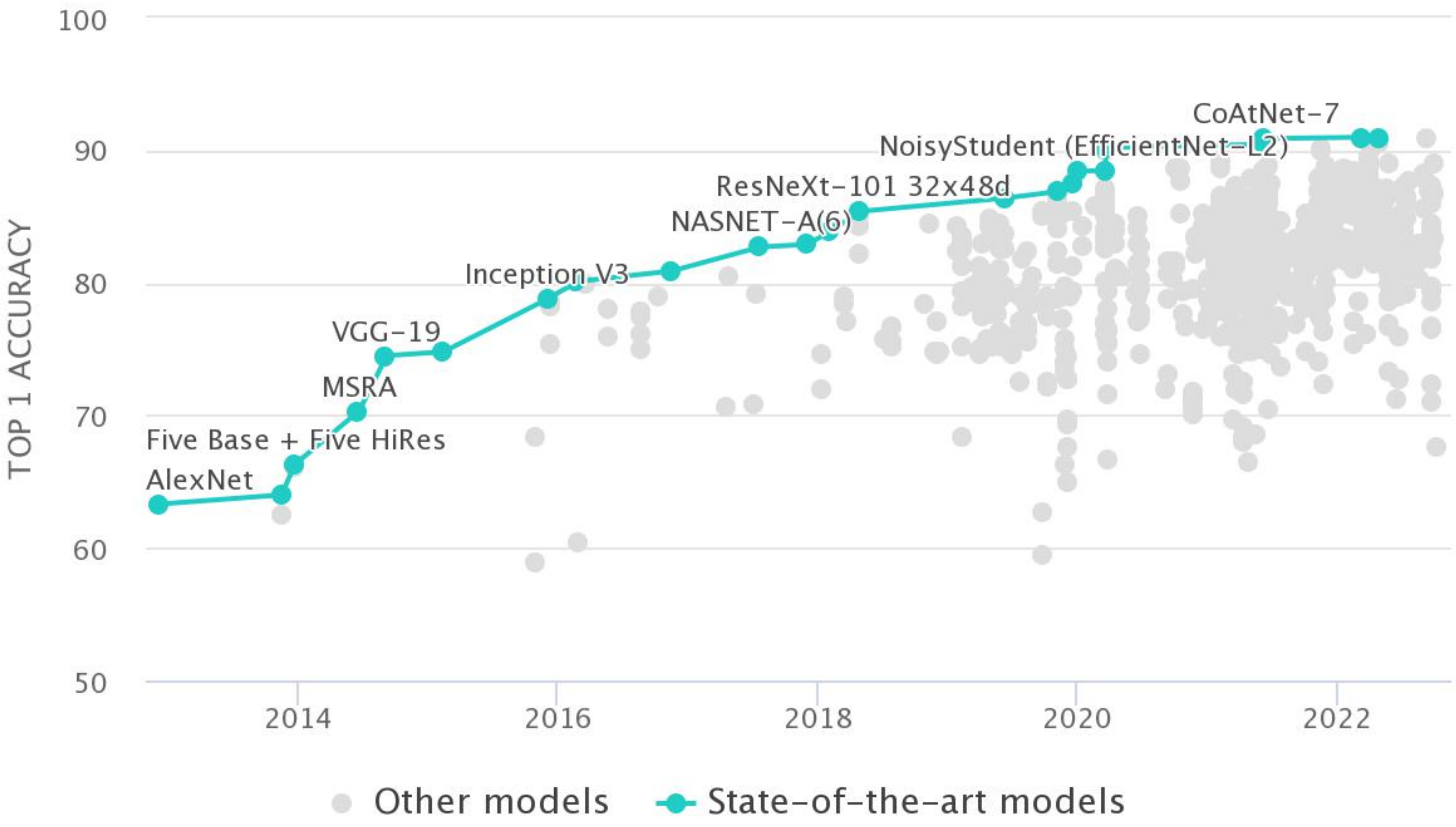
Table 1. Specification for MobileNetV3-Large. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes the type of nonlinearity used. Here, HS denotes h-swish and RE denotes ReLU. NBN denotes no batch normalization. s denotes stride.

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

Today's NN architectures are very carefully designed!
There are so many things that we can tune:

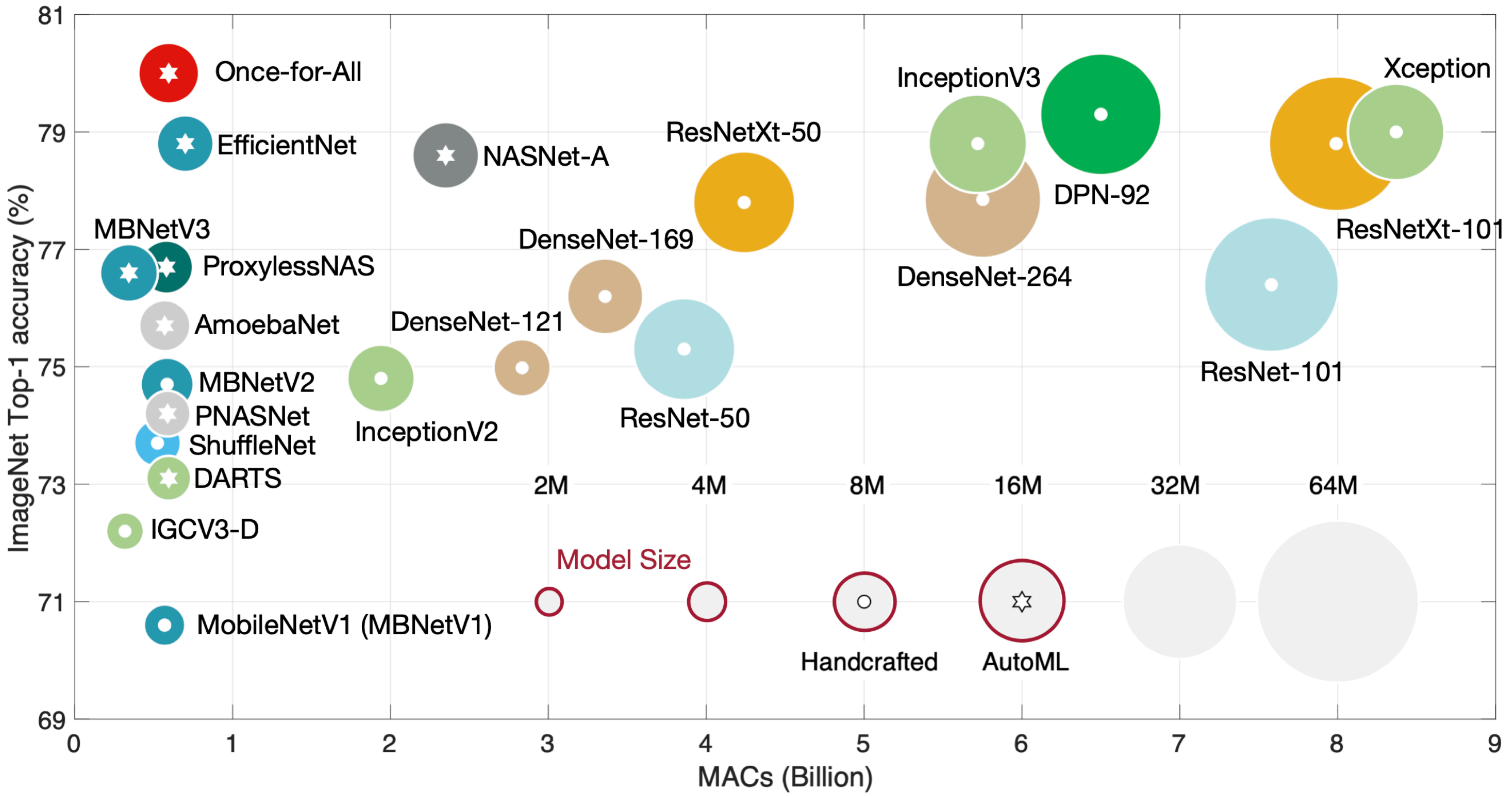
- Number of Layers — # ℕ
- #channels in each layer — # ℕ
- Activation function for each layer — # Swish? ReLU? GeLU?
- Type of the operator — # bottleneck? conv2d? else?
- Kernel size — # 3x3? 5x5? 7x7?
- Where to downsample? — # ℕ
- How to downsample? — # Stride? Pool?
- Overall topology — # Recurrent? Skip-connect? ...

Large Search Space? Let Machine do it!



2017. Machine-tuned networks already took over in terms of test accuracy

From paperswithcode.com



2020. Machine-tuned networks are dominant in terms of compute-efficiency.

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

Neural Architecture Search

You search over the space of possible NN architectures, and try to look for the best one.

Dumb Strategy. Here is a dumb way to do it (brute-force search).

- List up all design hyperparameters and determine their range # No restriction
- Make a lot of NN by combining all possible choices of hyperparameters # Exponential growth
- Fully train each network until convergence, and select the best-performing one. # Much compute

$$\text{Total Compute} \approx (\text{Training Budget per Model}) \times \prod_{\text{Hyperparam}} (\#\ \text{Choices for the HP})$$

Question. What is a better strategy?

Revisit the dumb strategy...

- List up all design hyperparameters and determine their range — 1. Well-designed Search Space
- Make a lot of NN by combining all possible choices of hyperparameters — 2. Smarter Exploration
- Fully train each network until convergence, and select the best-performing one. — 3. Cheaper evaluation

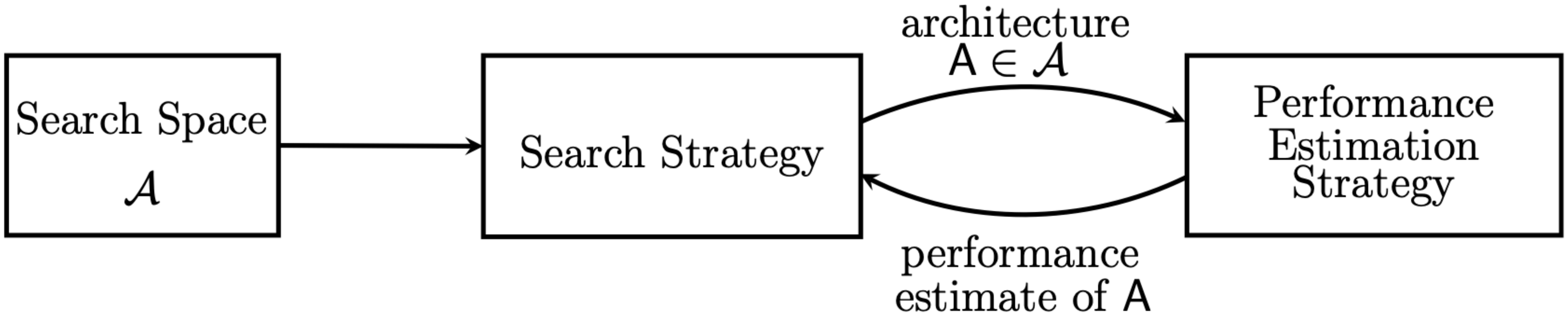
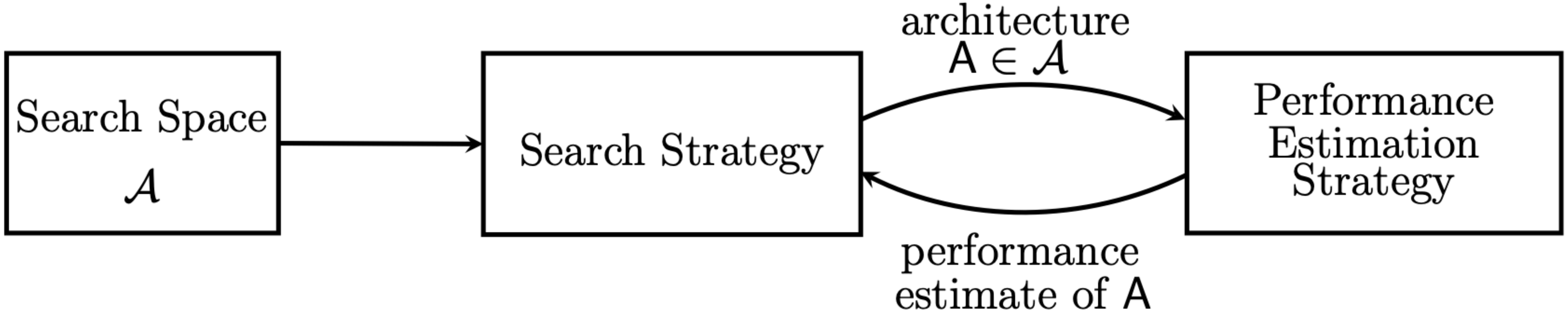


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

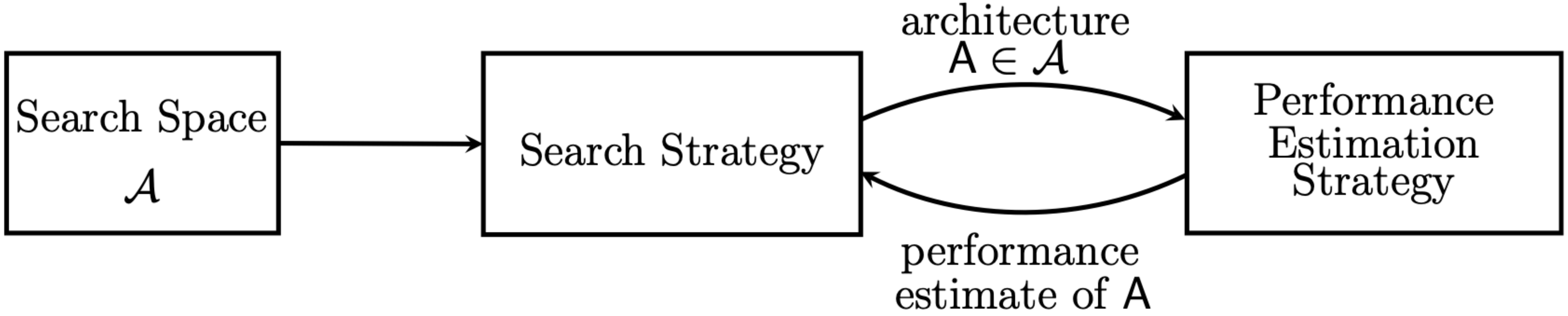
Elsken et al., "Neural Architecture Search: A Survey" JMLR 2019.



Search Space.

- Defines which architectures can be represented in principle.
- Can be reduced in size, by incorporating prior knowledge about typical properties of architectures well-suited for a task (e.g., convolution for vision)
  - But this introduce human bias; better to avoid if we have enough compute.

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



Search Strategy.

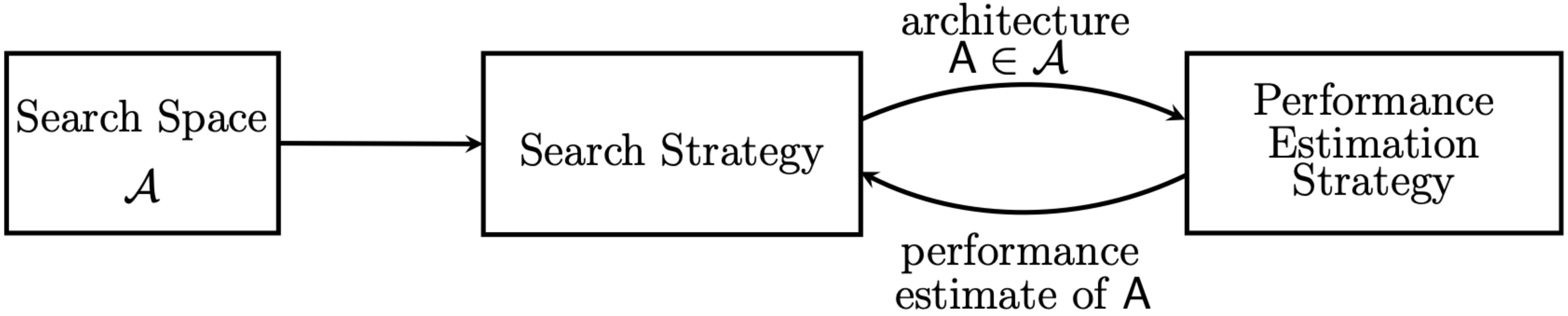
- Details how to explore the search space.
  (Can be exponentially large or unbounded)

- Encompasses the classical exploration-exploitation trade-off

  - Desirable to find well-performing architecture quickly

  - Premature convergence to a region of suboptimal architecture should be avoided

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



Performance Estimation Strategy.

- Estimate the predictive performance of a model.
- Simplest: to perform a standard training and measure validation performance
  - Too expensive
- Much efforts made in reducing the cost.

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

An Elementary Form

To give you an idea, here is an elementary version of neural architecture search:

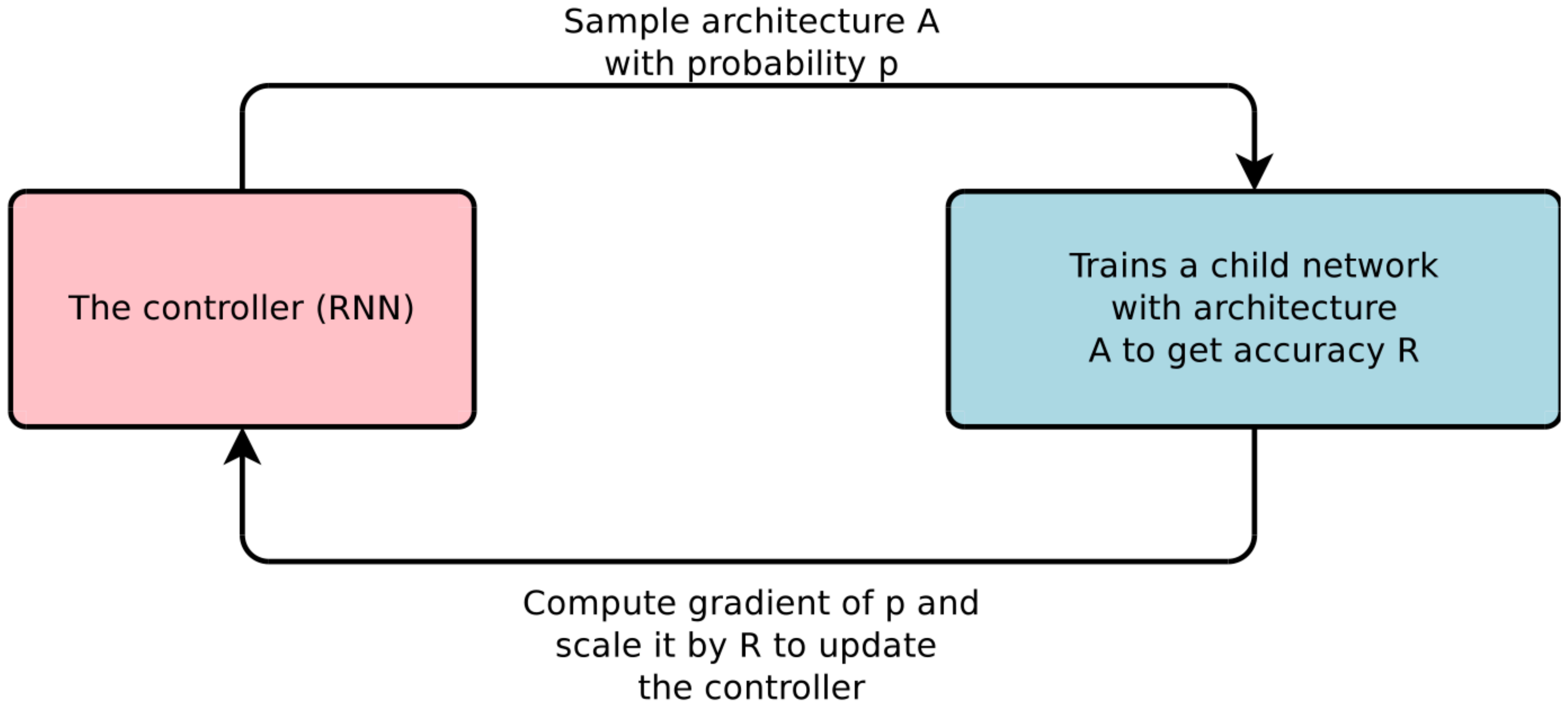
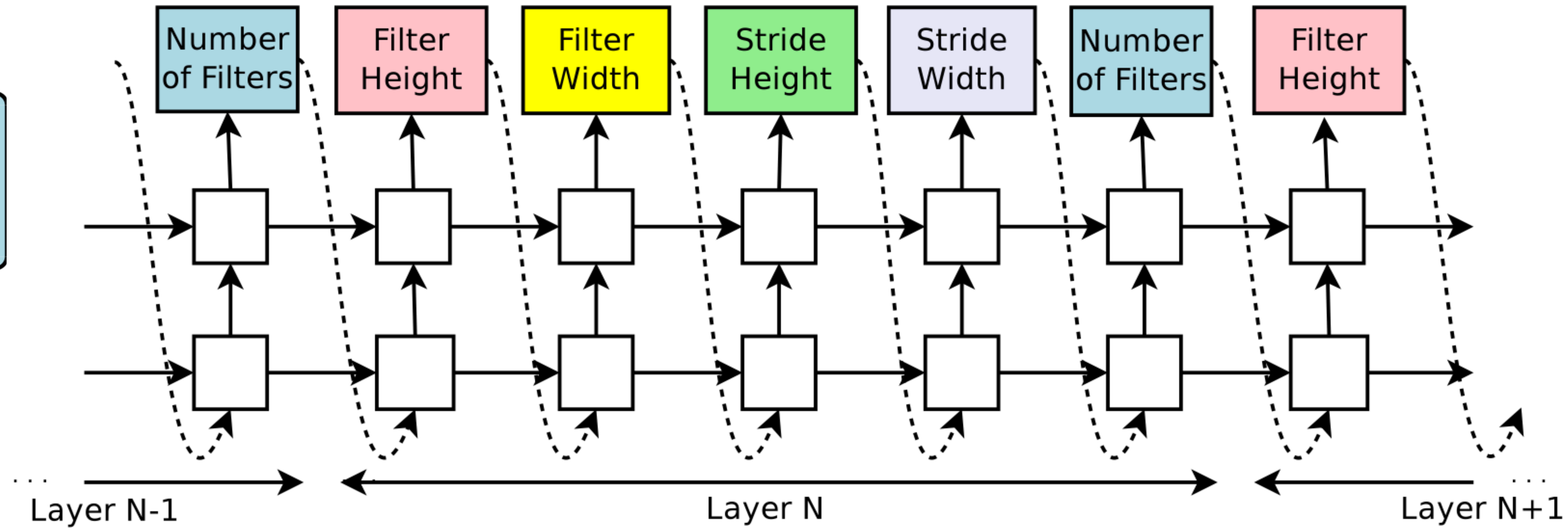


Figure 1: An overview of Neural Architecture Search.



- **Search Space:** Convolutional neural network with L layers.
- **Search Strategy:** Reinforcement Learning with RNN controller (policy); reward given by the performance
- **Performance Estimation:** Train for t steps

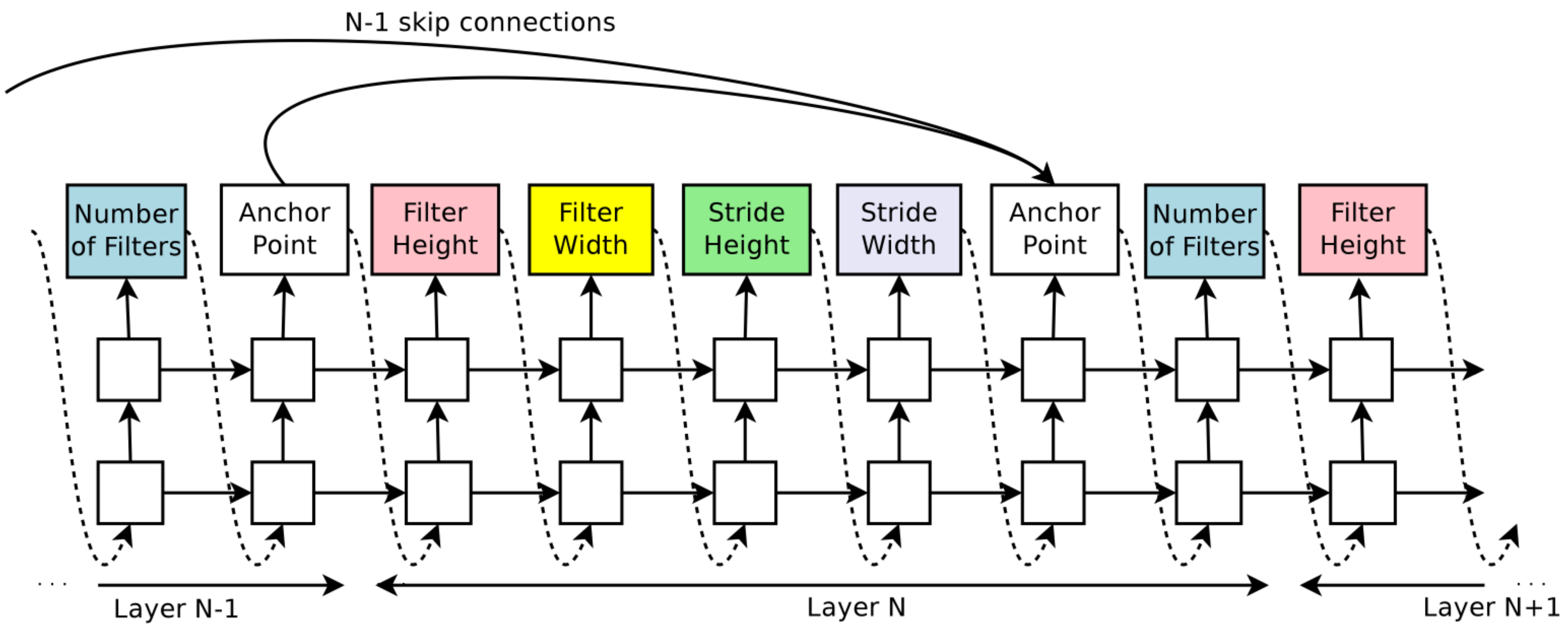
Let's tackle each of these separately...

Zoph and Le, "NAS with RL" ICLR 2017

Search Space

A simplest method construct the search space is to view NN as a sequence of layers (forget about topologies), and predict the HP for each layer separately — first predict type, and predict relevant parameters.

Example. Zoph and Le (2017) constrains the search space to a fully convolutional network,
and aim to predict #filters / kernel size / strides / residuals (and do additional tricks, too)



Zoph and Le, "NAS with RL" ICLR 2017

Predicting all parameters separately is a tedious thing to do—Zoph and Le (2017) uses 800 GPUs for 28 days!

Cell-based representation. Same cell (also called *motifs*) may be used repeatedly as a block.

(This is quite common in successful in handcrafted nets, so why not?)

- Reduces the search space—less effort for search
- Known to have better transferrability across tasks

NASNet (Zoph et al., 2018) organizes the layers into multiple groups of normal / reduction cells, and try to learn how each normal / reduction cell should look like.

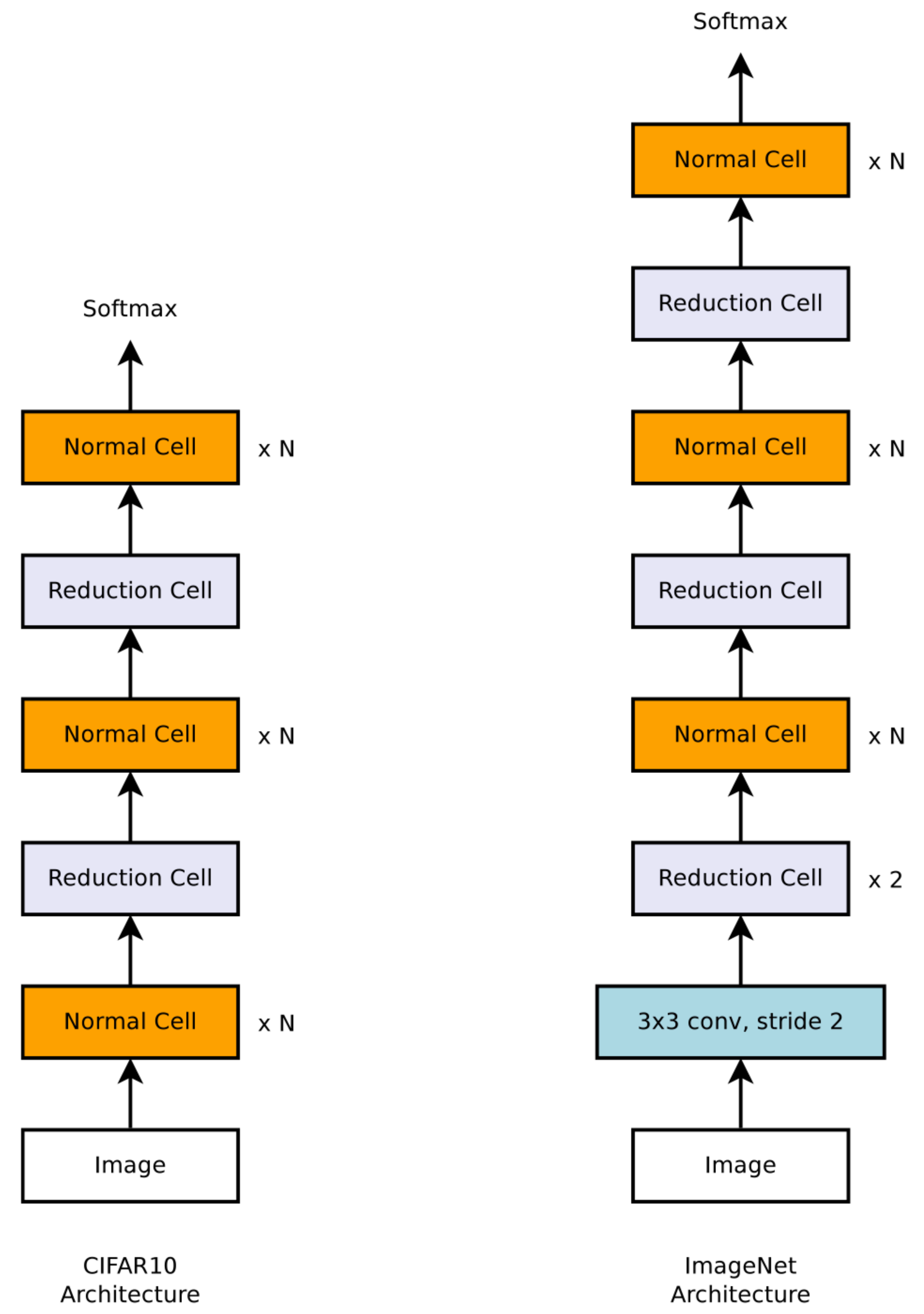


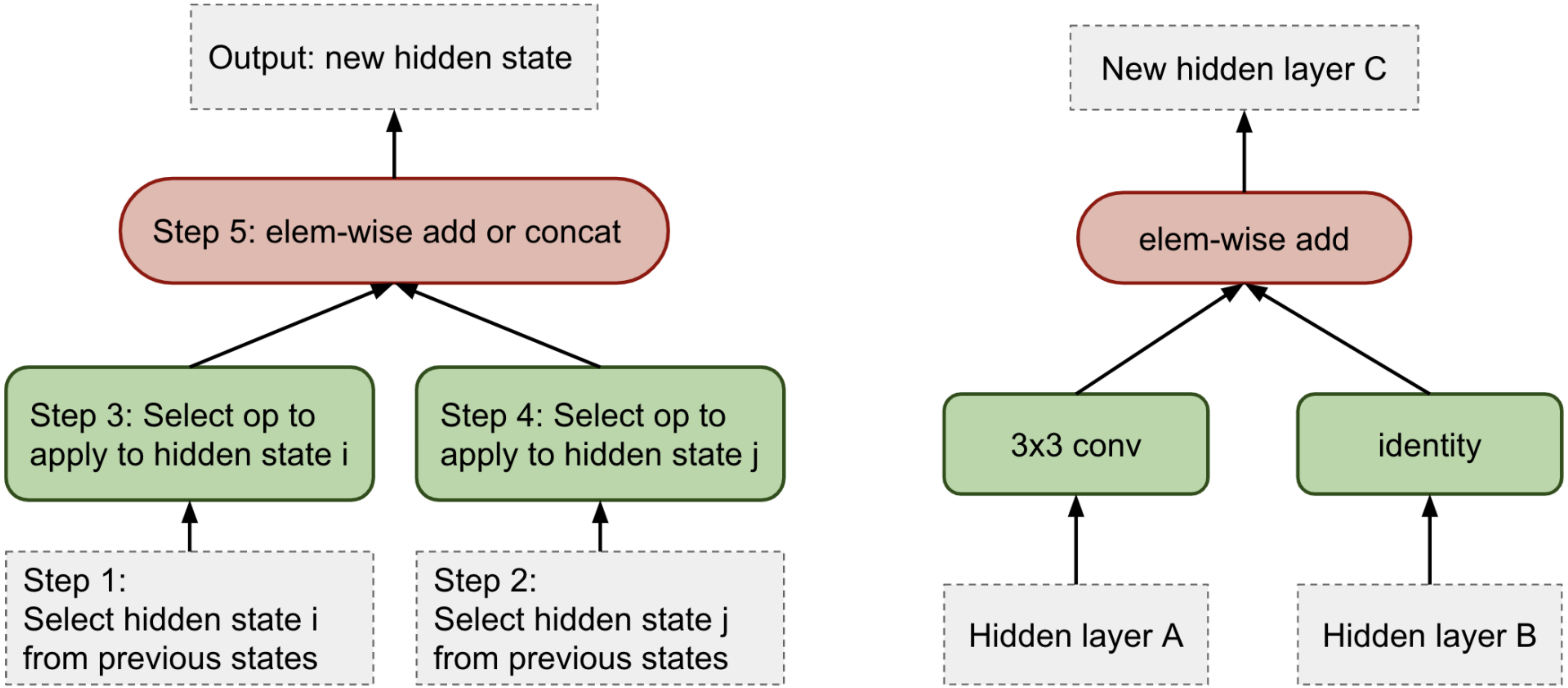
Figure 2. Scalable architectures for image classification consist of two repeated motifs termed *Normal Cell* and *Reduction Cell*. This diagram highlights the model architecture for CIFAR-10 and ImageNet. The choice for the number of times the Normal Cells that gets stacked between reduction cells, N, can vary in our experiments.

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

Each cell consists of $B = 5$ blocks.

Adding each block requires a five-step decision.
(We use RNN to make five decisions sequentially)

- 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

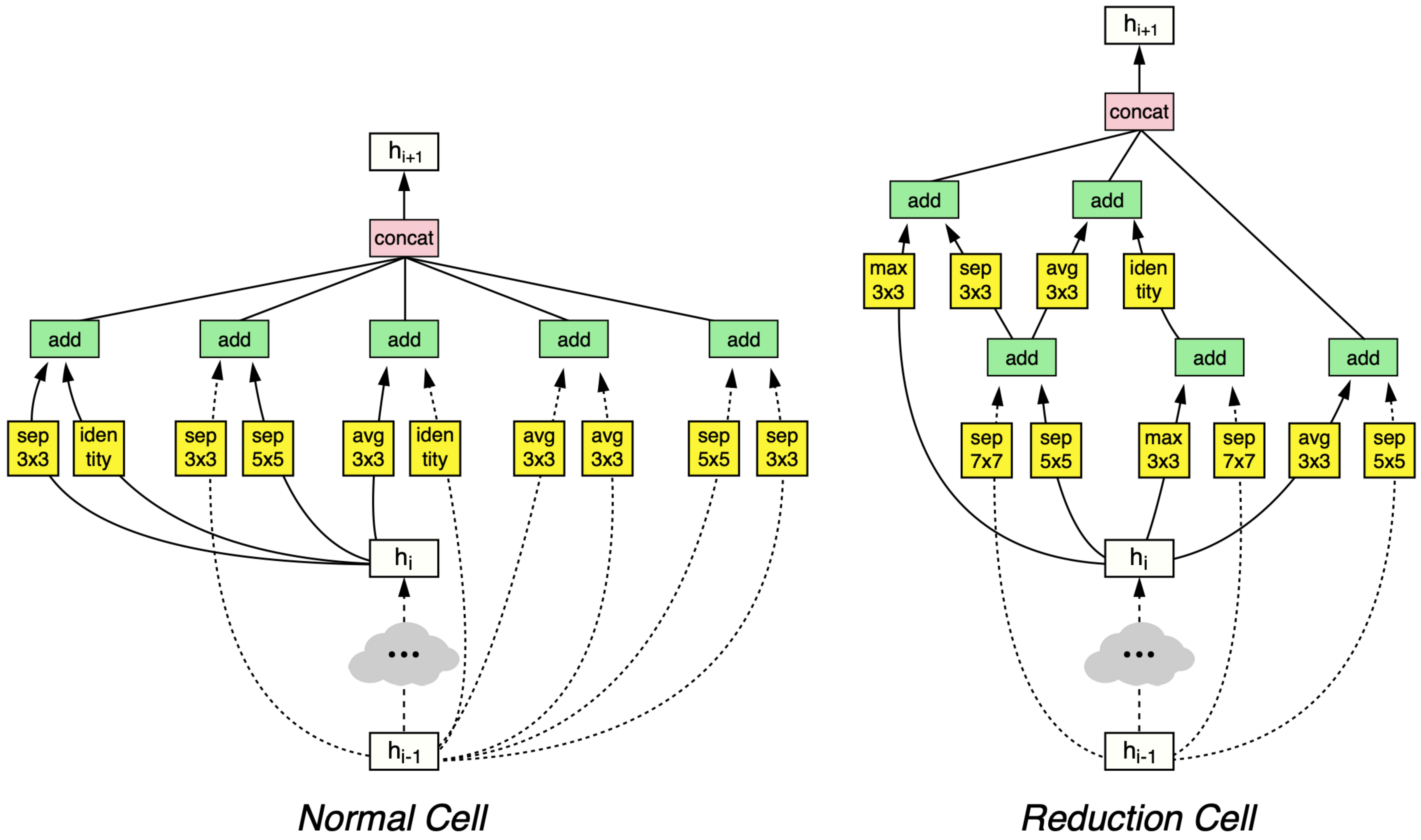


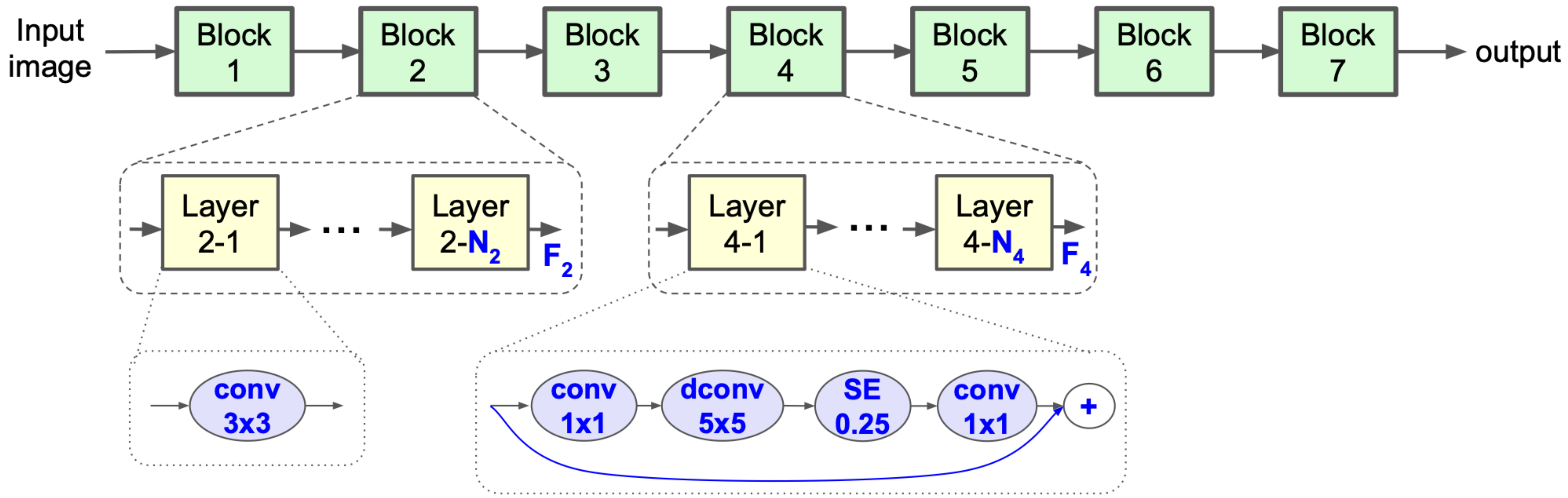
Figure 4. Architecture of the best convolutional cells (NASNet-A) with $B = 5$ blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green). Note that colors correspond to operations in Figure 3.

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

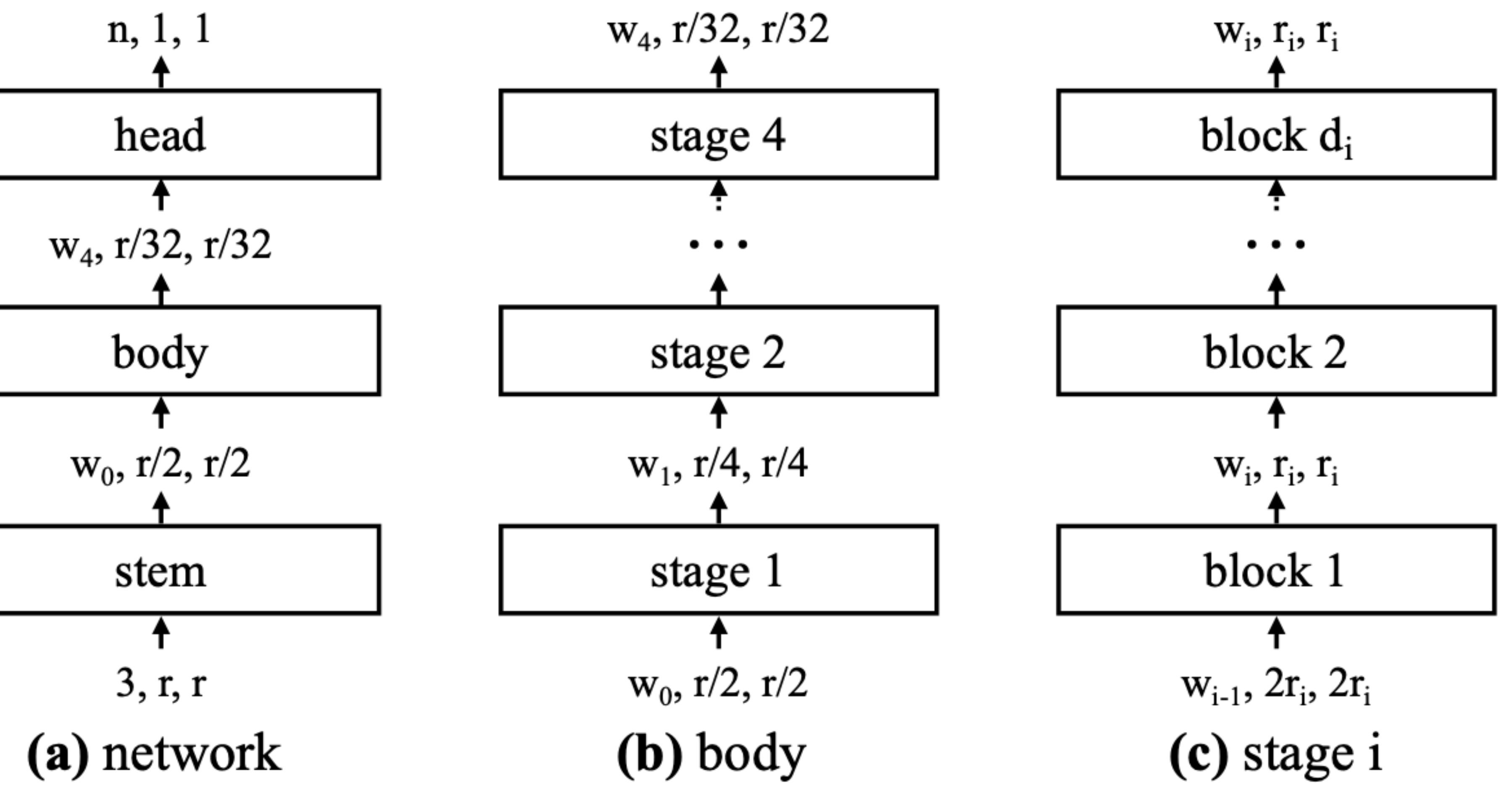
Search Space, with topologies

In previous examples, we can only change a very limited number of things at a network-level (⇔ cell-level)

In fact, we can only change the number of layers in many cases, with a fixed hierarchy.



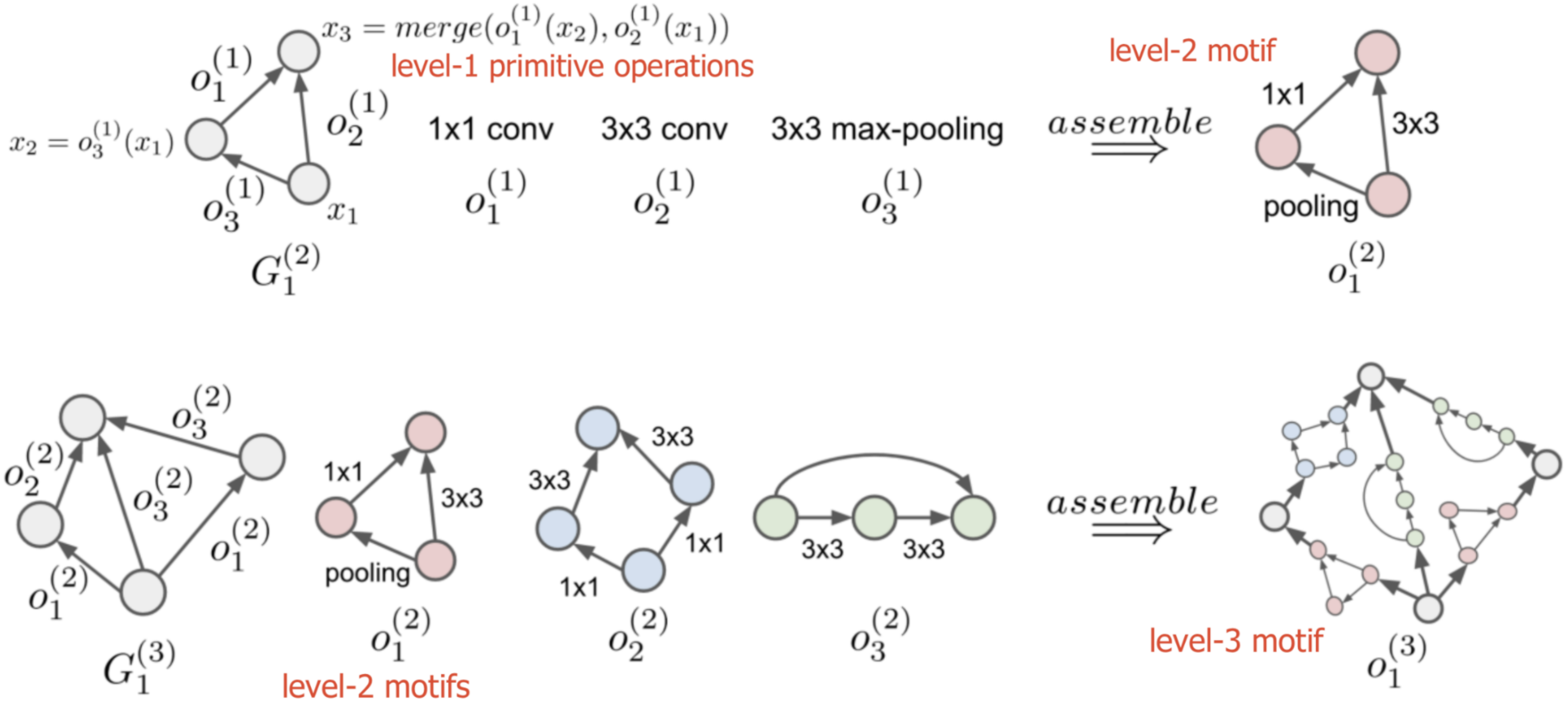
MNAS-Net
(CVPR 2019)



RegNet
(CVPR 2020)

There are several approaches that try to step outside this boundary.
(but they are not as popular as feedforward-like ones, as far as I know)

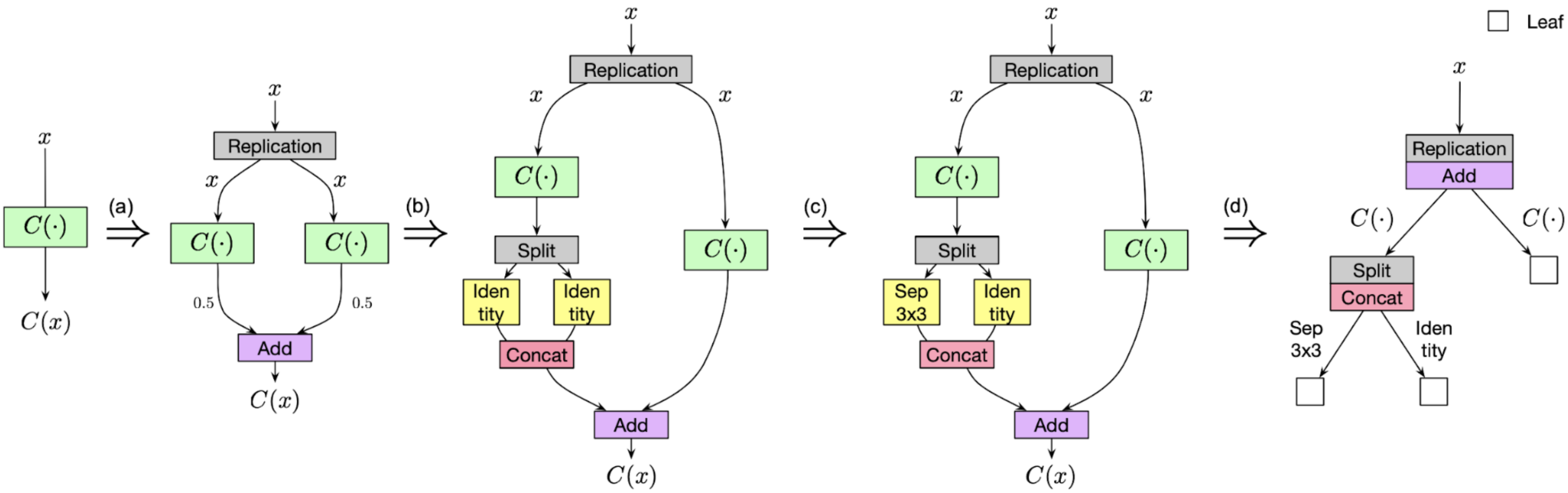
Example. Liu et al. (2018) proposes Hierarchical NAS, which uses *graph motifs* as a building block.



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

There are several approaches that try to step outside this boundary.
(but they are not as popular as feedforward-like ones, as far as I know)

Example. Cai et al. (2018) proposes to use tree-like branching process.



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

Next Up. Search strategies
Performance estimation strategies
Recent papers (transformers?)