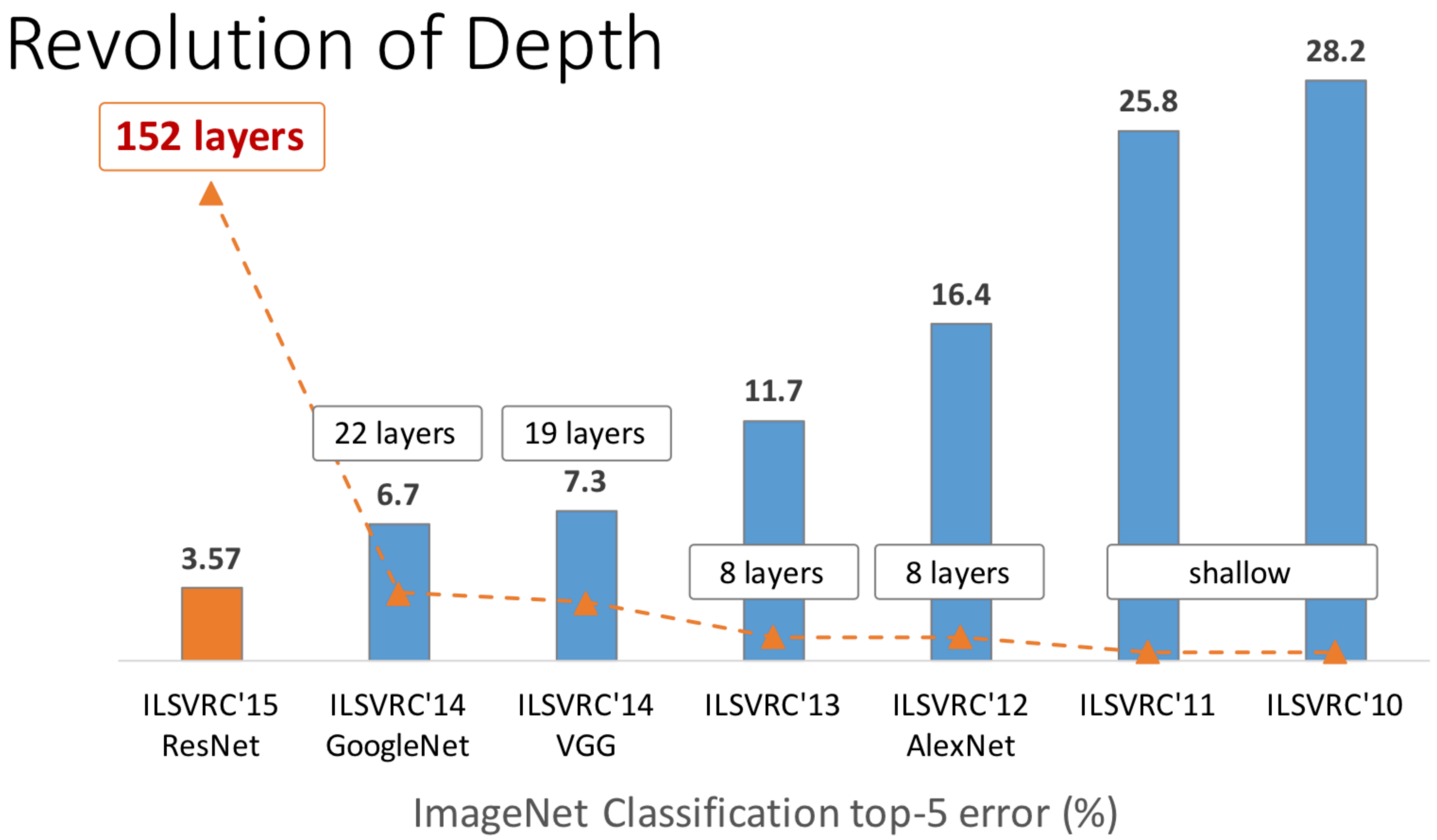
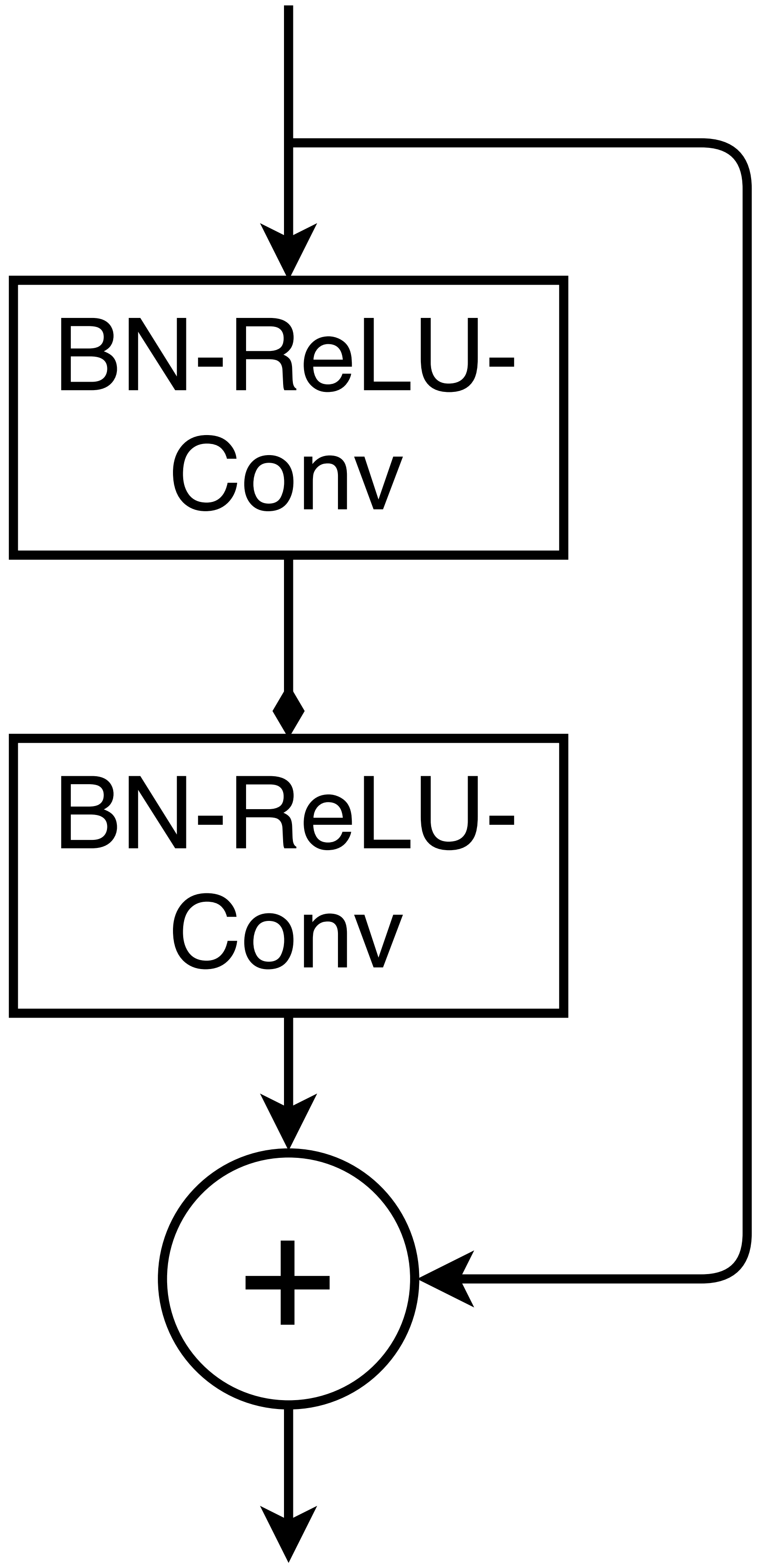
Neural Architecture

Ligeng Zhu
May 4th

The Blooming of CNNs



Bypass Connection



$$
\begin{aligned}
x_{\ell+1} &= F_\ell(x_\ell) + x_\ell \\
&= F_\ell(x_\ell) + F_{\ell-1}(x_{\ell-1}) + x_{\ell-1} \\
&= F_\ell(x_\ell) + F_{\ell-1}(x_{\ell-1}) + \ldots + F_1(x_1) \\
&= y_{\ell-1} + y_{\ell-2} + \ldots + y_1 .
\end{aligned}
$$

Direct gradient flow between any two layer, makes optimizer easy to optimize.

Cons of Residual Connection

- Information loss during summation (especially in deep case)

Cifar-10	param	error
Res-32	0.46M	7.51
Res-44	0.66M	7.17
Res-56	0.85M	6.97
Res-110	1.7M	**6.43**
Res-1202	19.4M	7.93

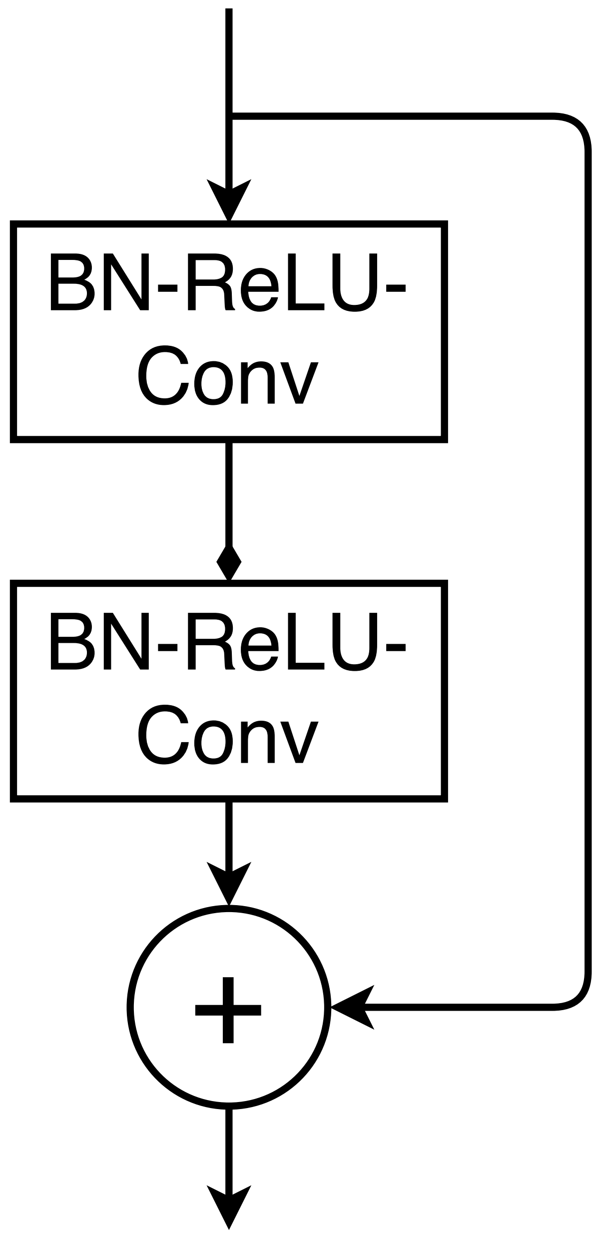
3 + 10 + 15 = 28 (easy)
28 = ? + ? + ? (difficult)

Improves of Residual Connection

- Avoid information loss via replacing sum with concat

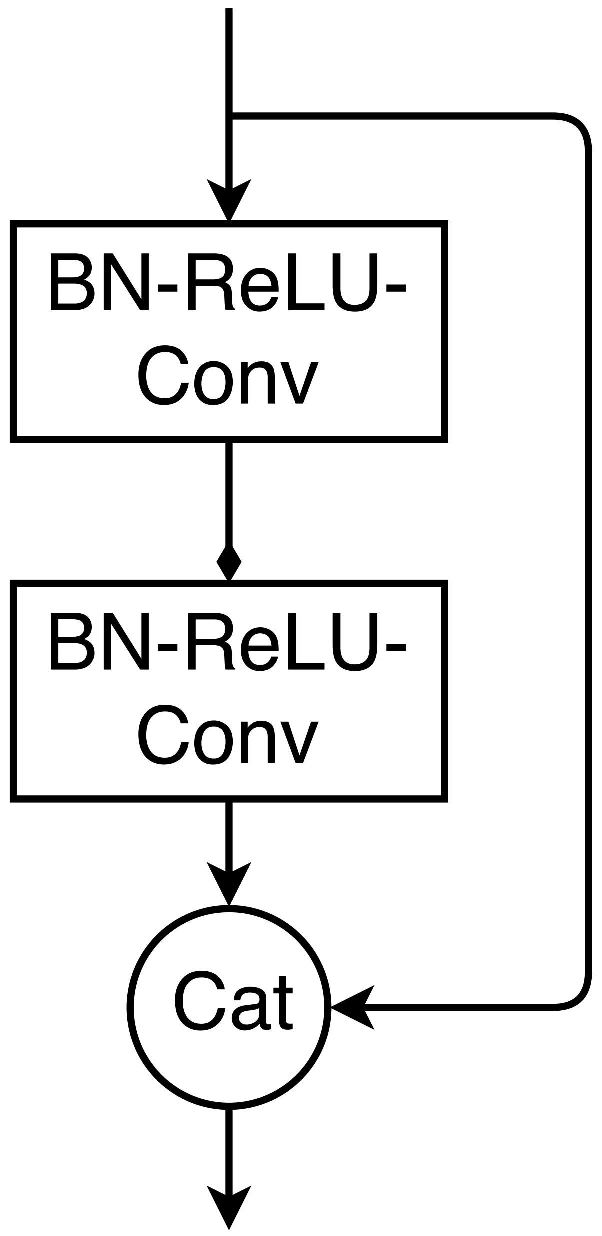
3 + 10 + 15 = 28 (easy)
28 = ? + ? + ? (difficult)

concat(3, 10, 15) = [3, 10, 15]
[3, 10, 15] = concat(3, 10, 15)




```
# ResNet pre-activation
def ResidualBlock(x):
    x1 = BN_ReLU_Conv(x)
    x2 = BN_ReLU_Conv(x1)
    return x + x2

for i in range(N):
    model.add(ResidualBlock)
```




```
# DenseNet BC structure
def DenseBlock(x):
    x1 = BN_ReLU_Conv(x)
    x2 = BN_ReLU_Conv(x1)
    return Concat([x, x2])

for i in range(N):
    model.add(DenseBlock)
```

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).

DenseNet

- Concat is more parameter-efficient than sum.

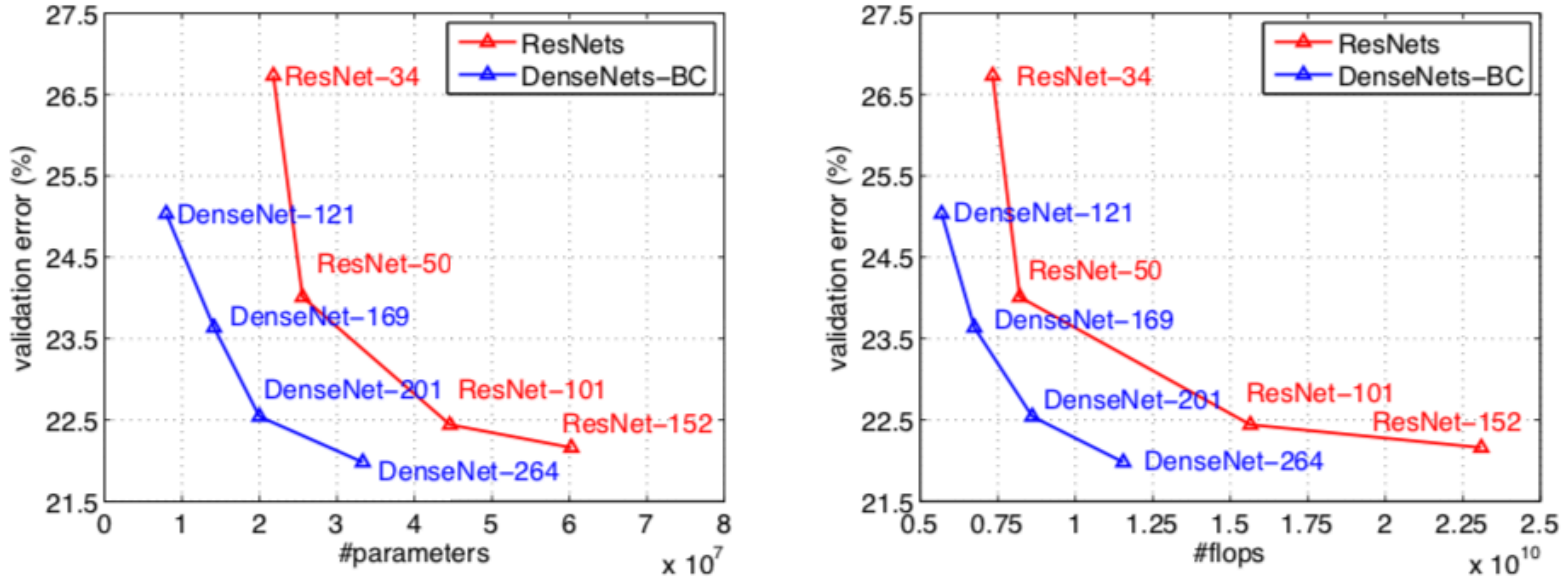


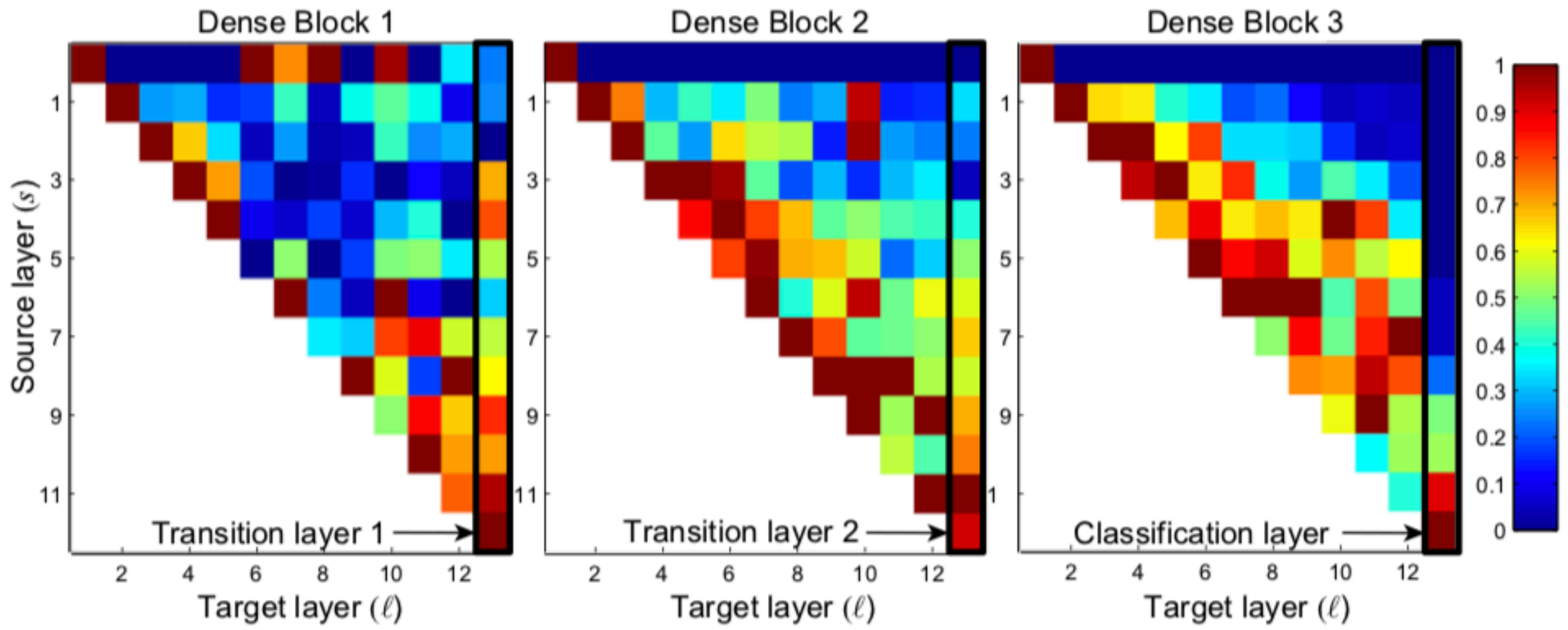
Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).

Cons of Concatenation

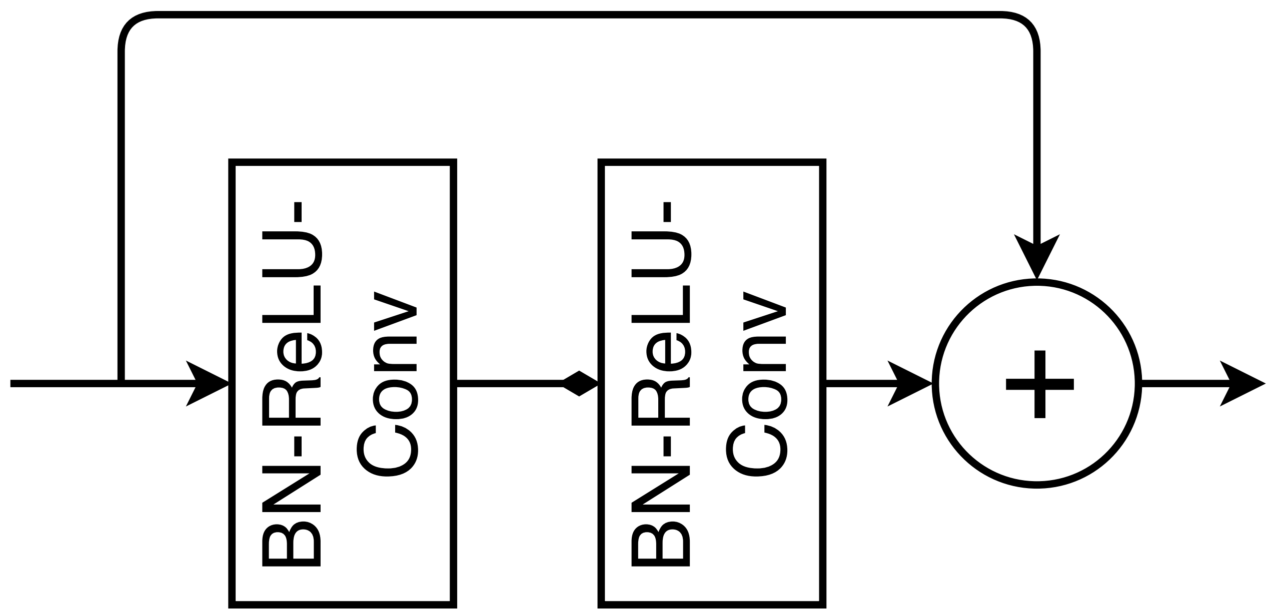
- Disadvantage :
  - Exploding parameters in deep networks-> O(n^2)
  - Redundant inputs in deeper layers

Dense-40-12	1.0M
Dense-100-12	7.0M
Dense-100-24	27.2M
Dense-200-12	OOM

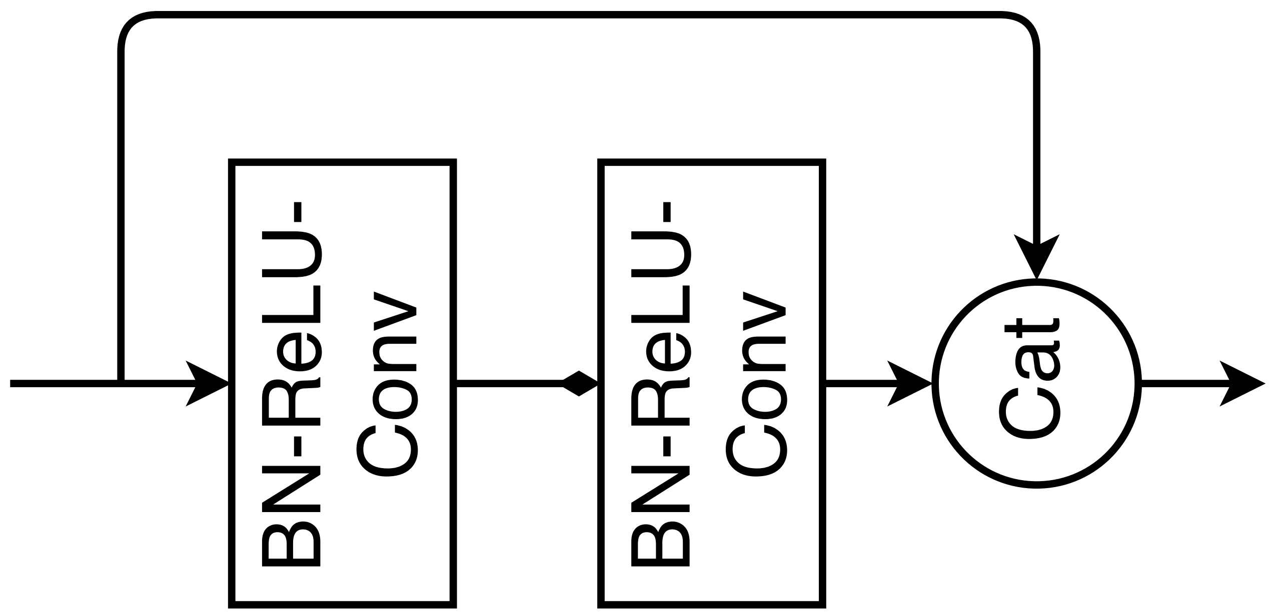


Rethink about ResNet and DenseNet

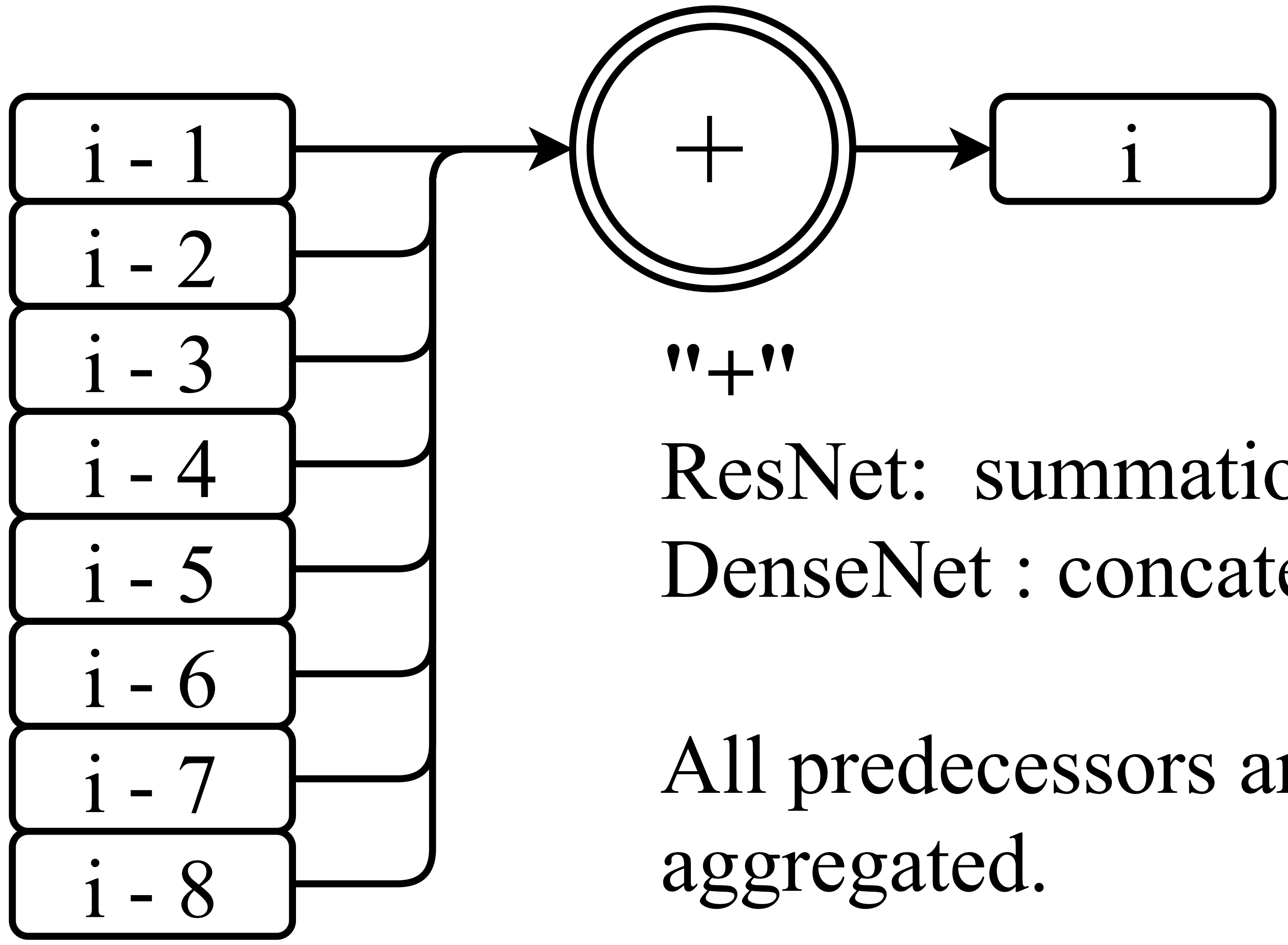
- Features are densely aggregated in both ResNet and DenseNet.



$$
\begin{aligned}
x_{\ell+1} &= F_\ell(x_\ell) + x_\ell \\
&= F_\ell(x_\ell) + F_{\ell-1}(x_{\ell-1}) + x_{\ell-1} \\
&= F_\ell(x_\ell) + F_{\ell-1}(x_{\ell-1}) + \ldots + F_1(x_1) \\
&\boxed{= y_{\ell-1} + y_{\ell-2} + \ldots + y_1.}
\end{aligned}
$$



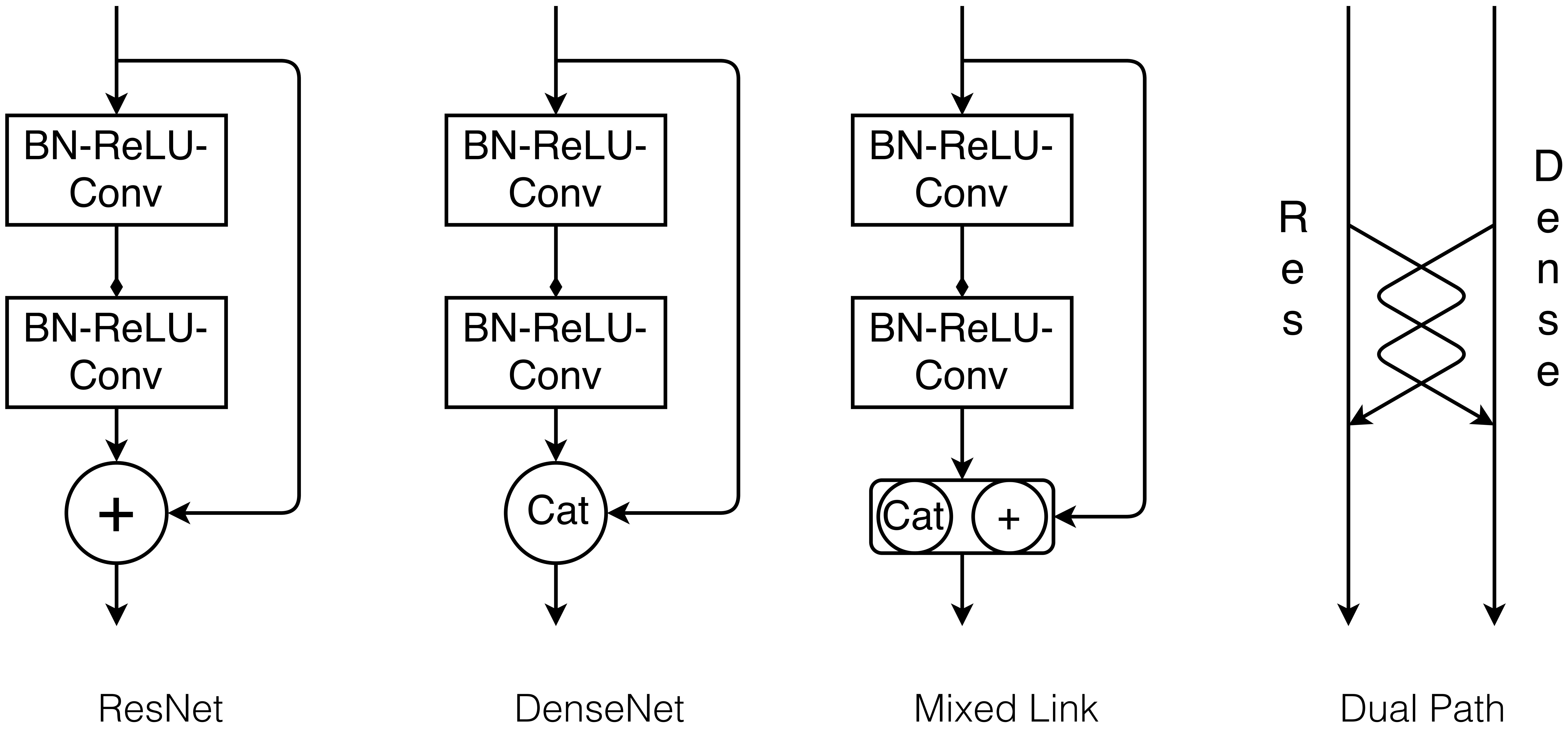
$$
\begin{aligned}
x_{\ell+1} &= F_\ell(x_\ell) \oplus x_\ell \\
&= F_\ell(x_\ell) \oplus F_{\ell-1}(x_{\ell-1}) \oplus x_{\ell-1} \\
&= F_\ell(x_\ell) \oplus F_{\ell-1}(x_{\ell-1}) \oplus \ldots \oplus F_1(x_1) \\
&\boxed{= y_{\ell-1} \oplus y_{\ell-2} \oplus \ldots \oplus y_1.}
\end{aligned}
$$



"+"
ResNet: summation
DenseNet : concatenation

All predecessors are aggregated.

Variations of dense aggregation (how to aggregate)

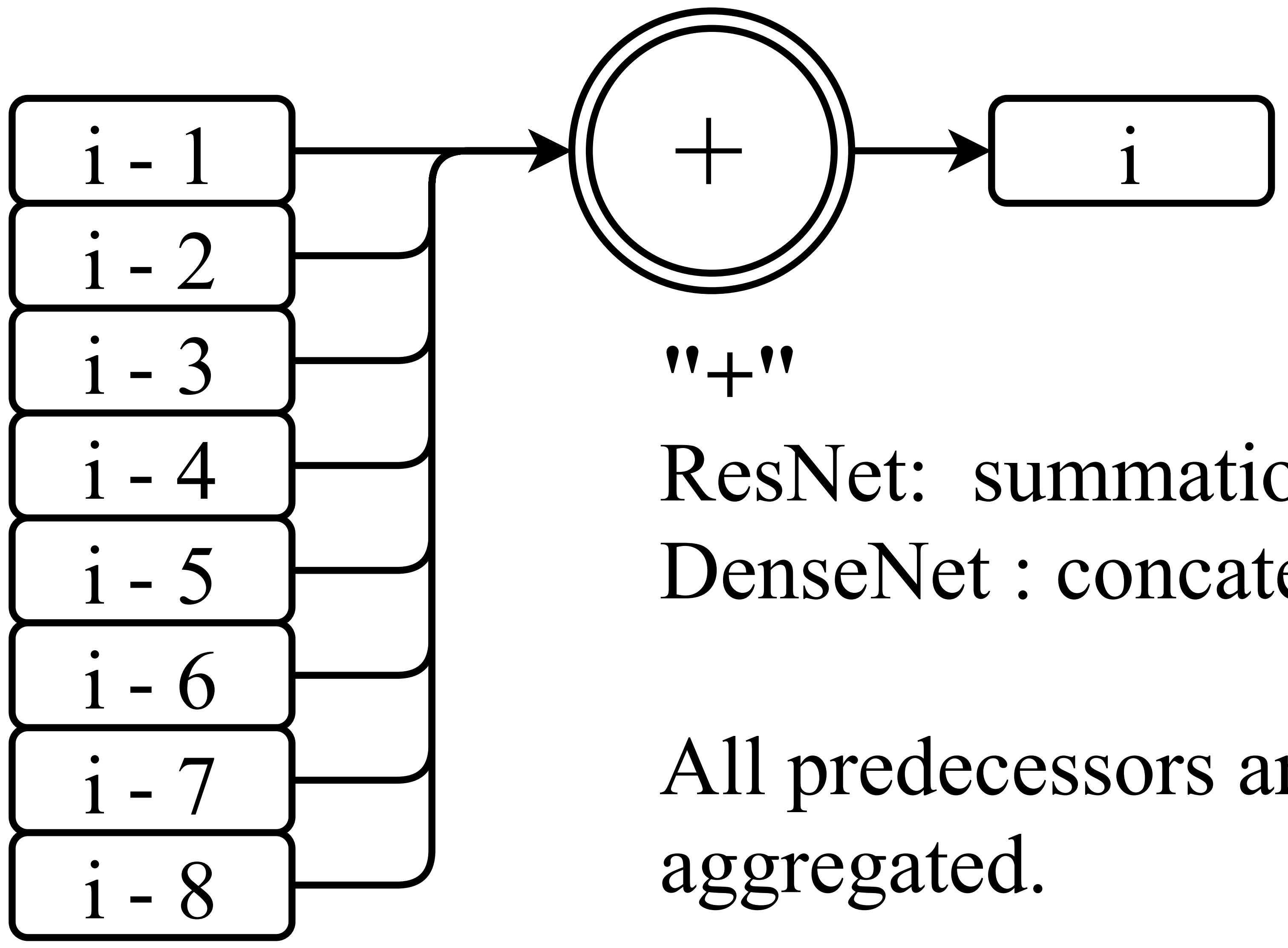


Sum and Concat

- ResNet and DenseNet are both dense aggregation structure.
- Summation appears to be powerful on gradients, BUT
  - Information loss leads to parameter deficiency
- Concat is a better way of aggregations, BUT
  - Blowing params and redundancy
- Any way to utilize both advantages without bringing new troubles?

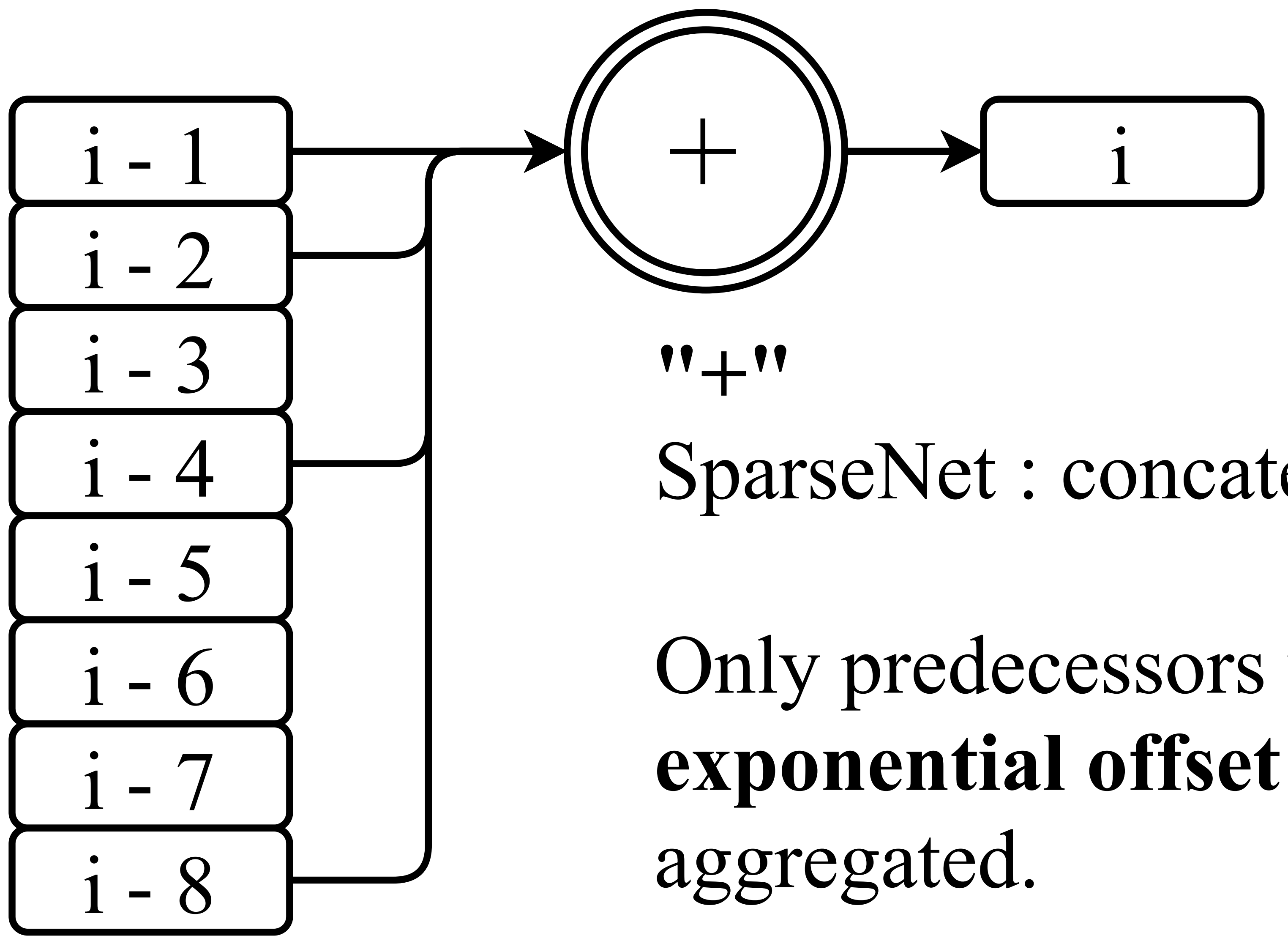
Sparsely Aggregated Convolutional Networks

- Instead of "**how to aggregate**", consider "**what to aggregate**"
- Only gather layers with exponential offsets



"+"
ResNet: summation
DenseNet : concatenation

All predecessors are aggregated.



"+"
SparseNet : concatenation

Only predecessors with **exponential offset** are aggregated.

Zhu, L., Deng, R., Maire, M., Deng, Z., Mori, G., & Tan, P. (2018). Sparsely aggregated convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 186-201).

Params and Gradient Flow Analysis

- The total skip connections (params)

$$log_c 1 + log_c 2 + ... + log_c N = log_c N! \approx log_c N^N = O(NlgN)$$

- The gradient flow between any two layers

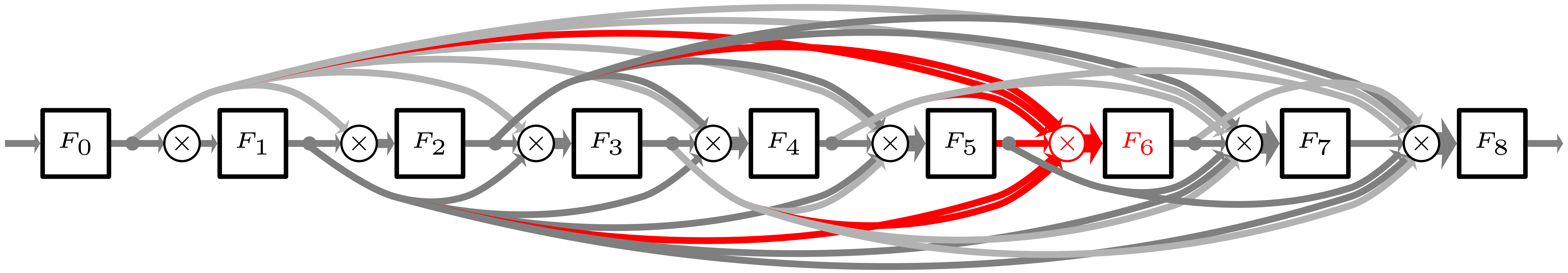
$$N \text{ offsets} => log_c N \times (c-1) \text{ steps}$$

	Parameters	Shortest Gradient Path	Aggregated Features
Plain	$O(N)$	$O(N)$	$O(1)$
ResNets	$O(N)$	$O(1)$	$O(\ell)$
DenseNets	$O(N^2)$	$O(1)$	$O(\ell)$
SparseNets (sum)	$O(N)$	$O(\log(N))$	$O(\log \ell)$
SparseNets (concat)	$O(N \log N)$	$O(\log(N))$	$O(\log \ell)$

Zhu, L., Deng, R., Maire, M., Deng, Z., Mori, G., & Tan, P. (2018). Sparsely aggregated convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 186-201).

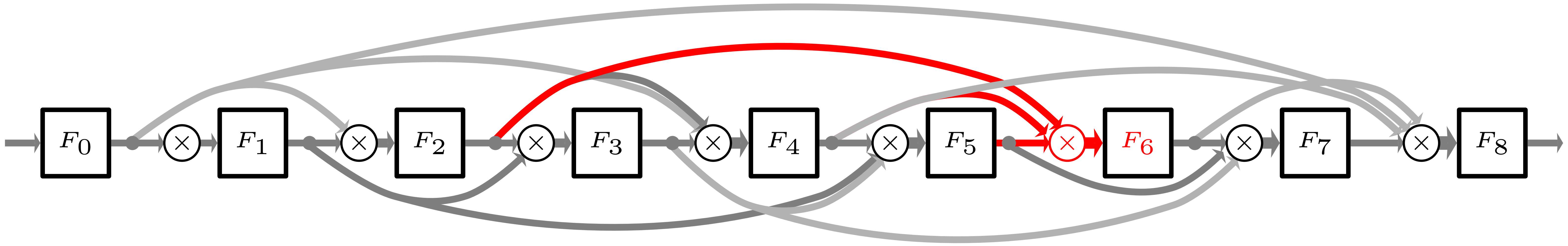
Dense Concatenation and Sparse Aggregation

(a) Dense Aggregation: Equivalent Exploded View of **(a)**



ResNet & DenseNet: each layer takes all previous outputs.

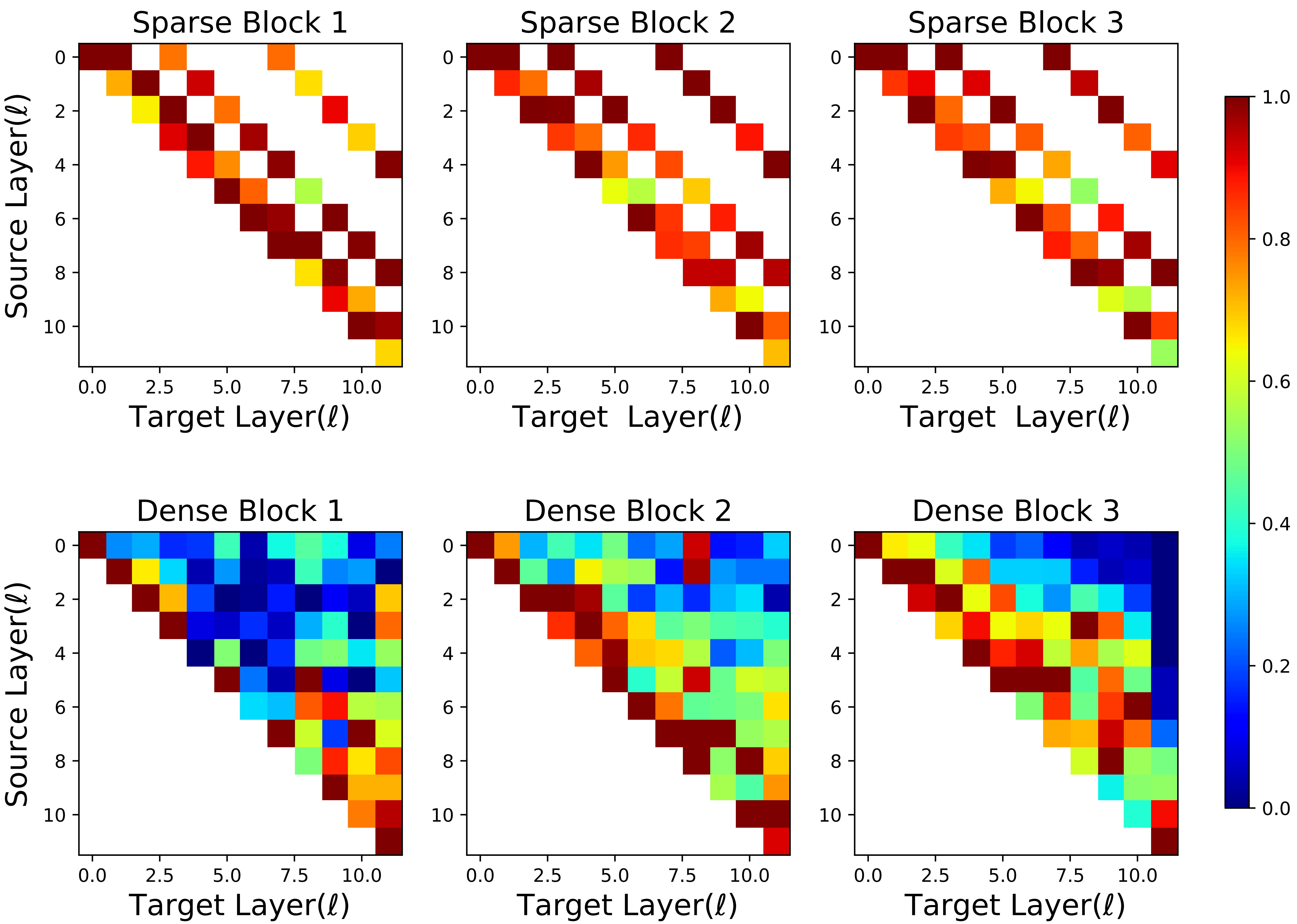
(b) Sparse Aggregation (Our Proposed Topology)



SparseNet: each layer takes all outputs with exponential offset (e.g., i-1, i - 2, i - 4, i - 8 …)

Zhu, L., Deng, R., Maire, M., Deng, Z., Mori, G., & Tan, P. (2018). Sparsely aggregated convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 186-201).

Better parameter utilization



Zhu, L., Deng, R., Maire, M., Deng, Z., Mori, G., & Tan, P. (2018). Sparsely aggregated convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 186-201).

Better Param-Perform Curve

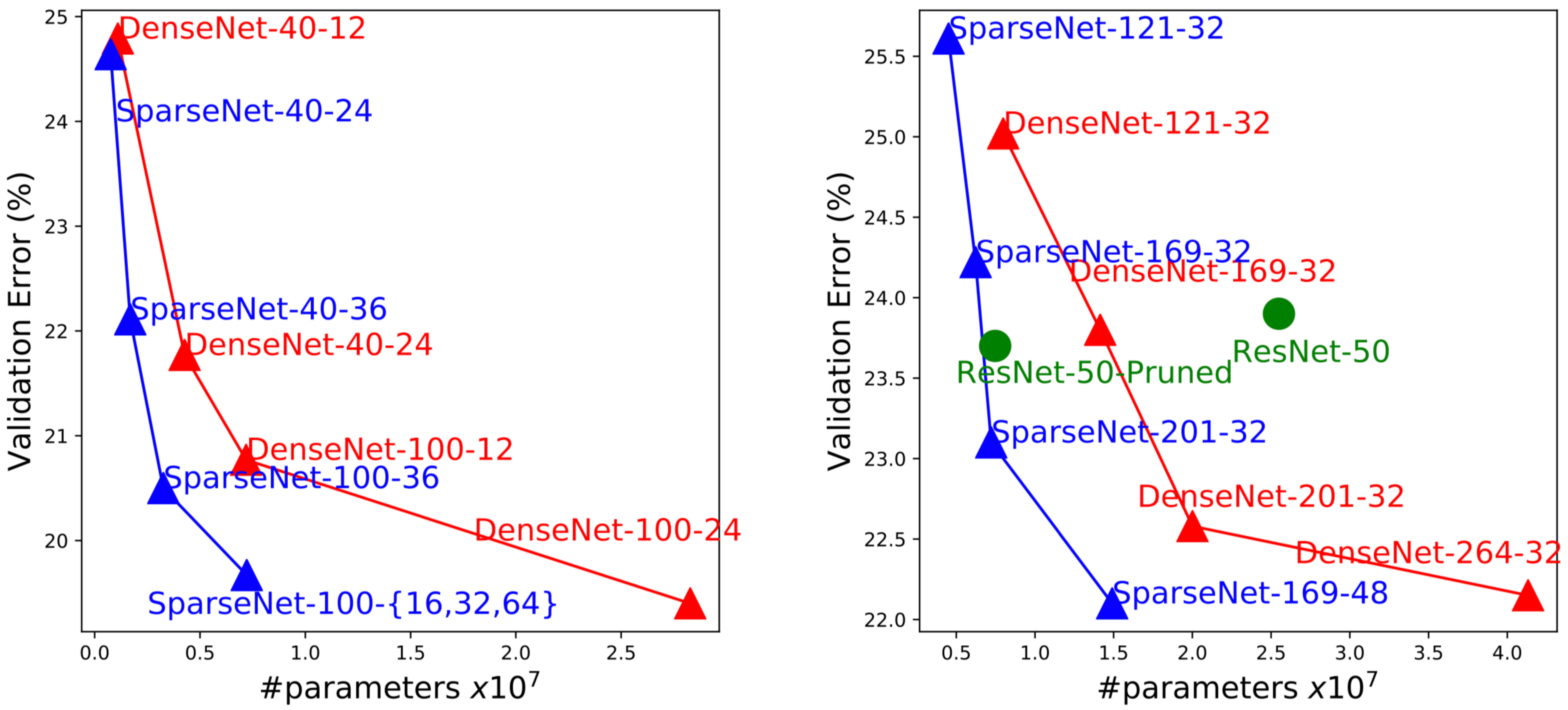
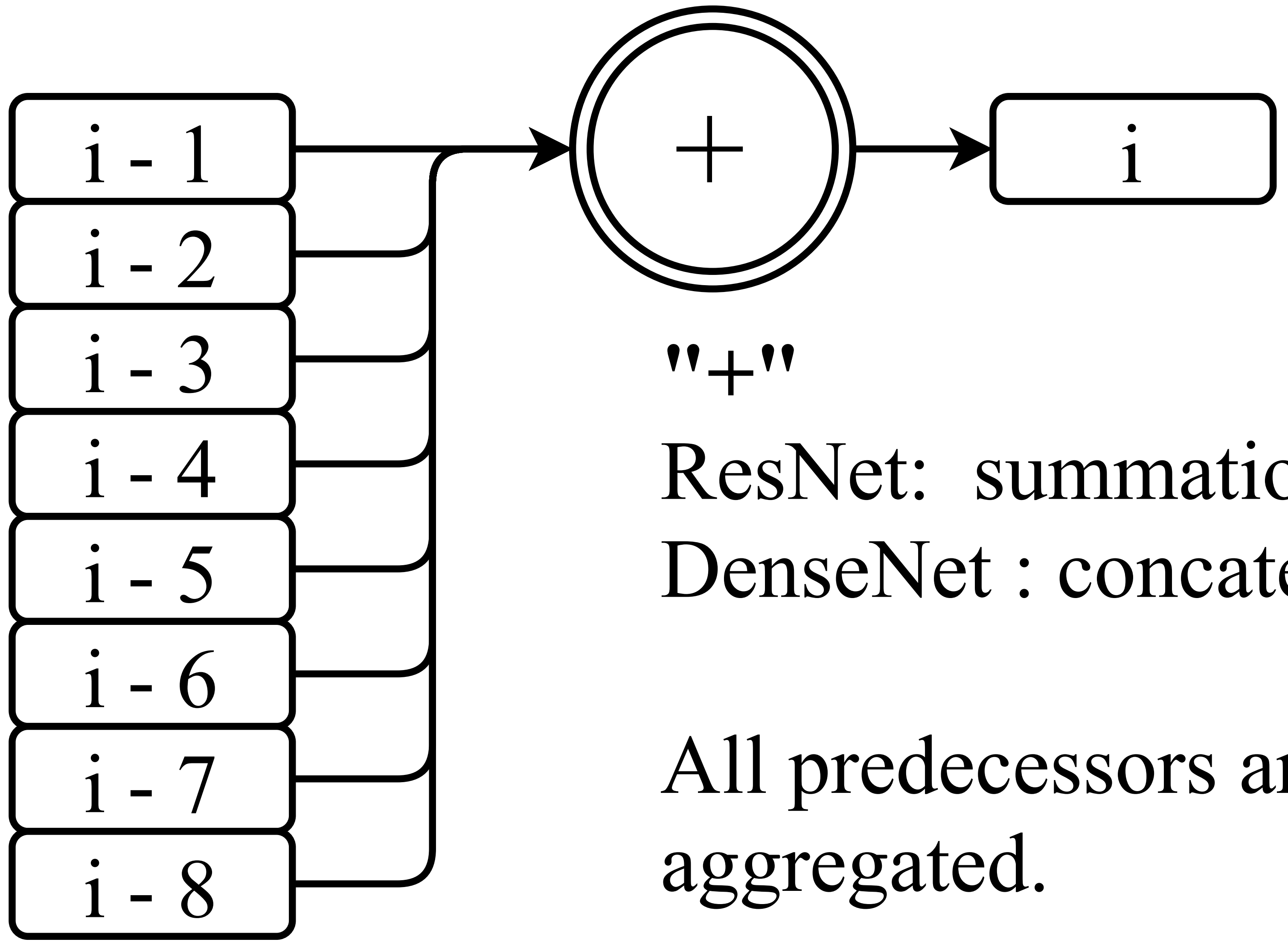


Fig. 2. Parameter efficiency. Comparison between DenseNets and SparseNets[⊕] on top-1% error and number of parameters with different configurations. ***Left:*** CIFAR. ***Right:*** ImageNet. SparseNets achieve lower error with fewer parameters.

Zhu, L., Deng, R., Maire, M., Deng, Z., Mori, G., & Tan, P. (2018). Sparsely aggregated convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 186-201).

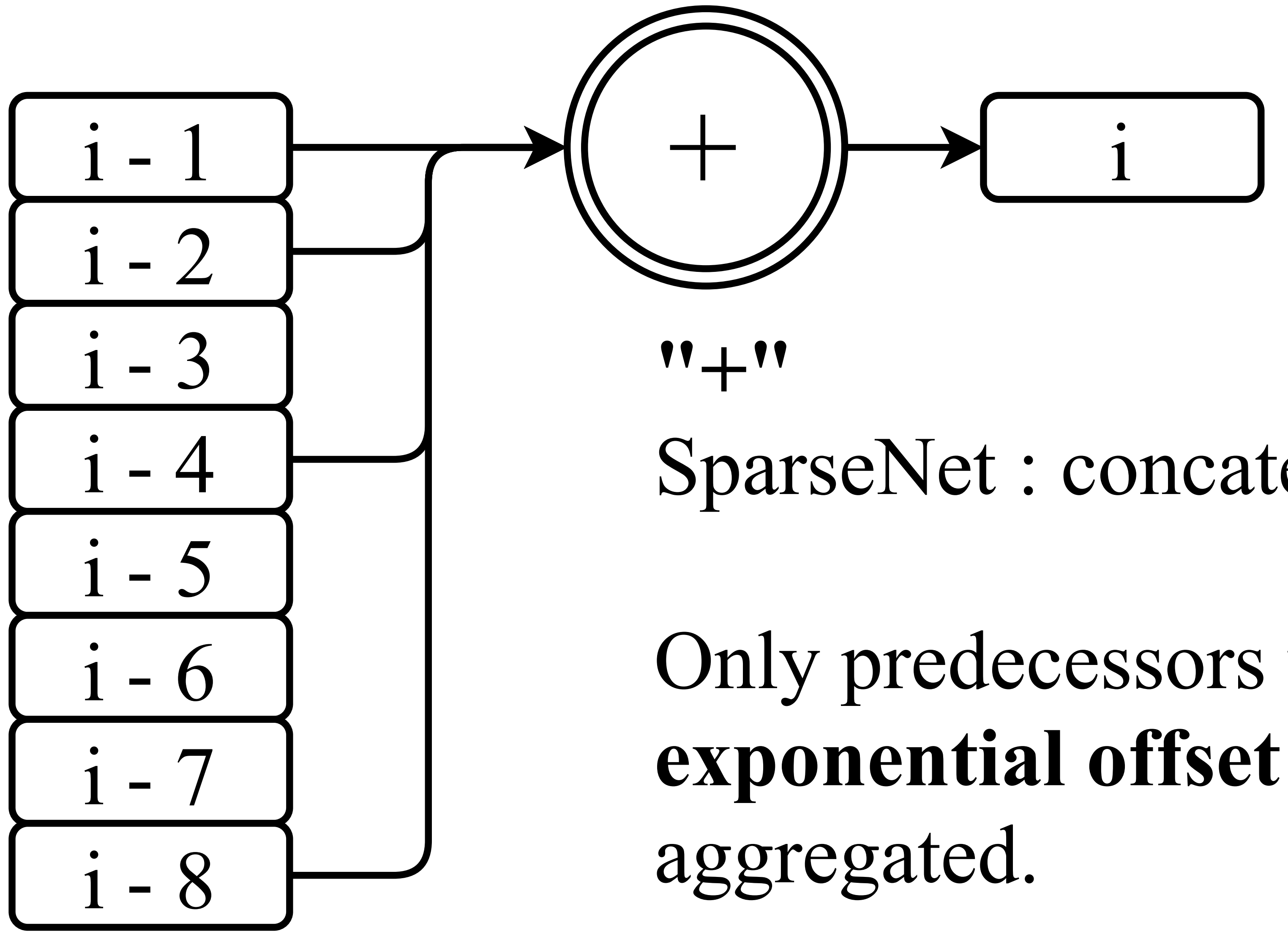
Remaining Question

- What if, let the network self-choose what to aggregate?



"+"
ResNet: summation
DenseNet : concatenation

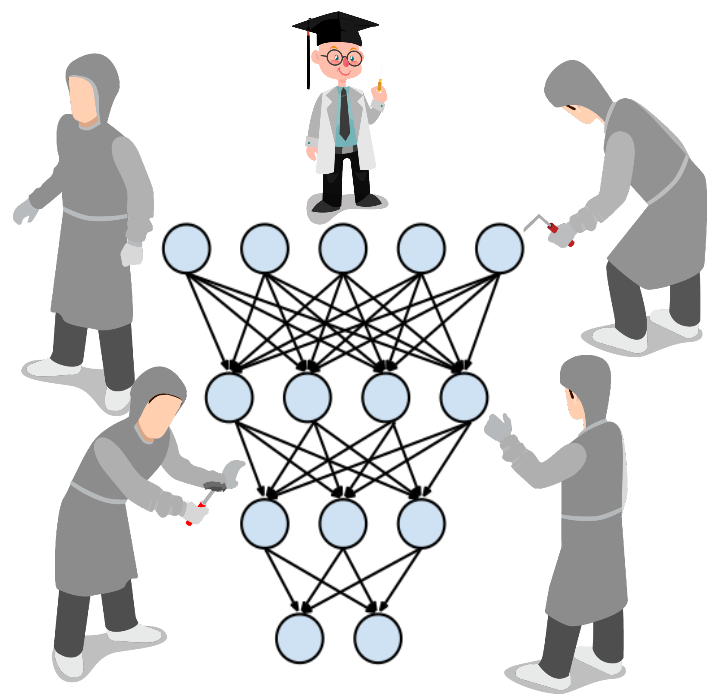
All predecessors are aggregated.



"+"
SparseNet : concatenation

Only predecessors with **exponential offset** are aggregated.

From Manual Design to Architecture Search

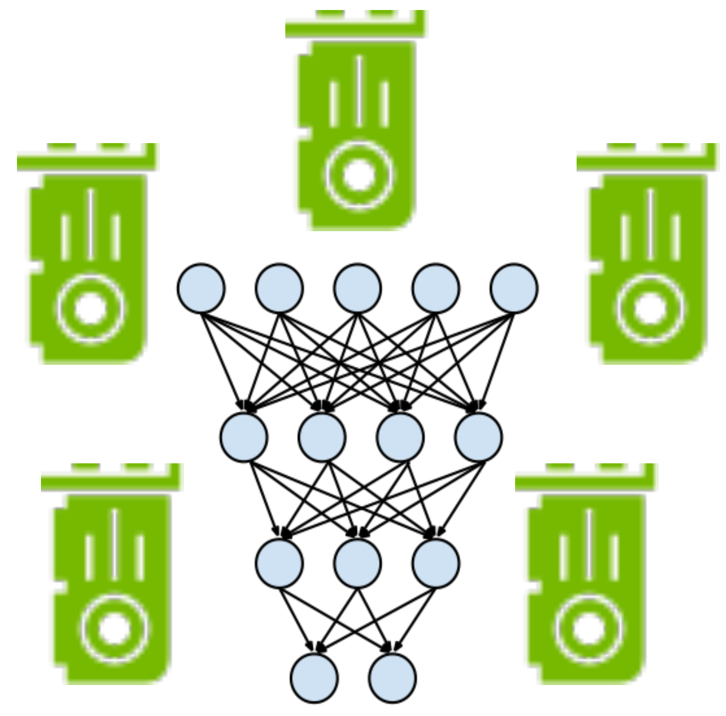


Human Expertise

Manual Architecture Design

VGGNets
Inception Models
ResNets
DenseNets
….

Computational Resources

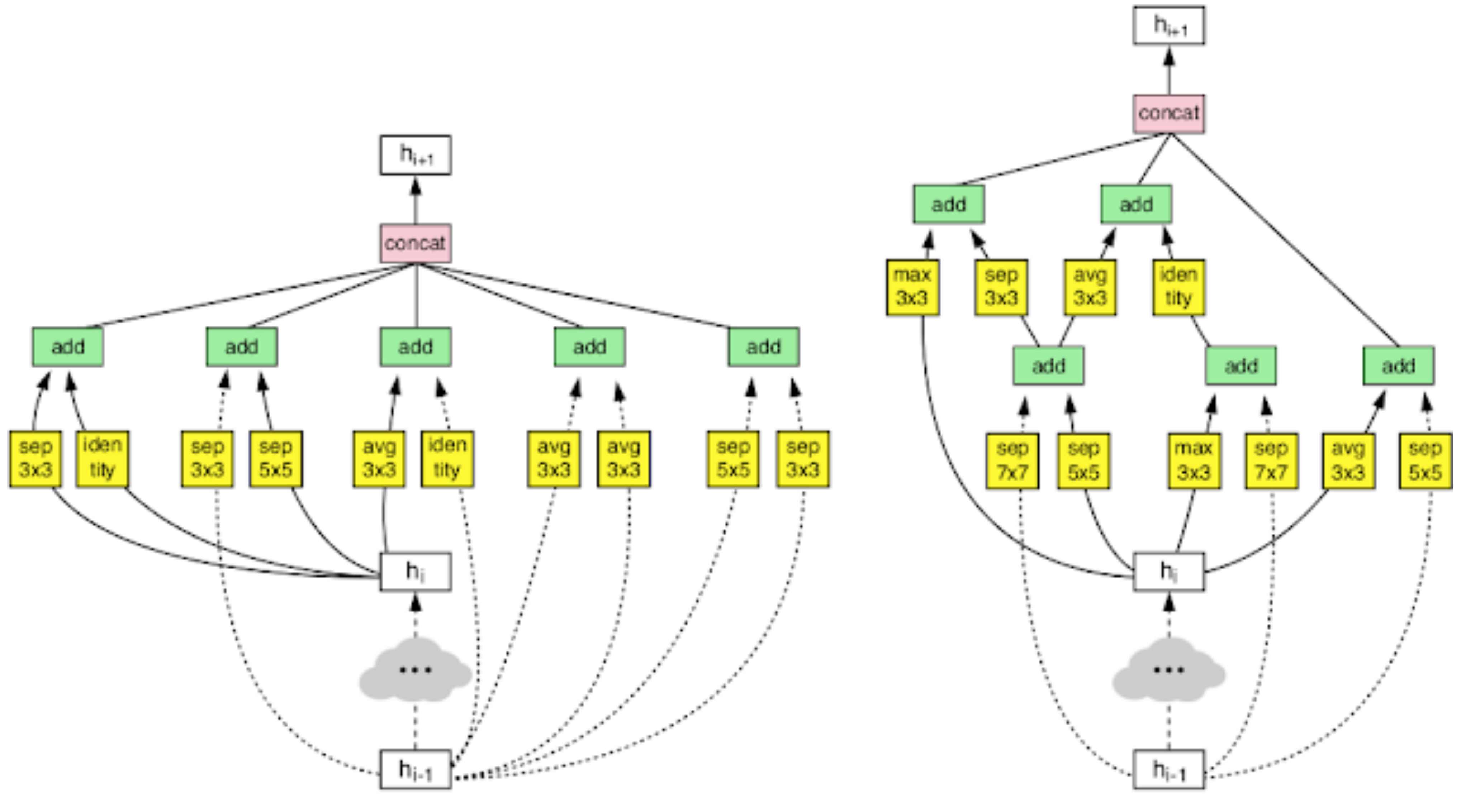


Machine Learning

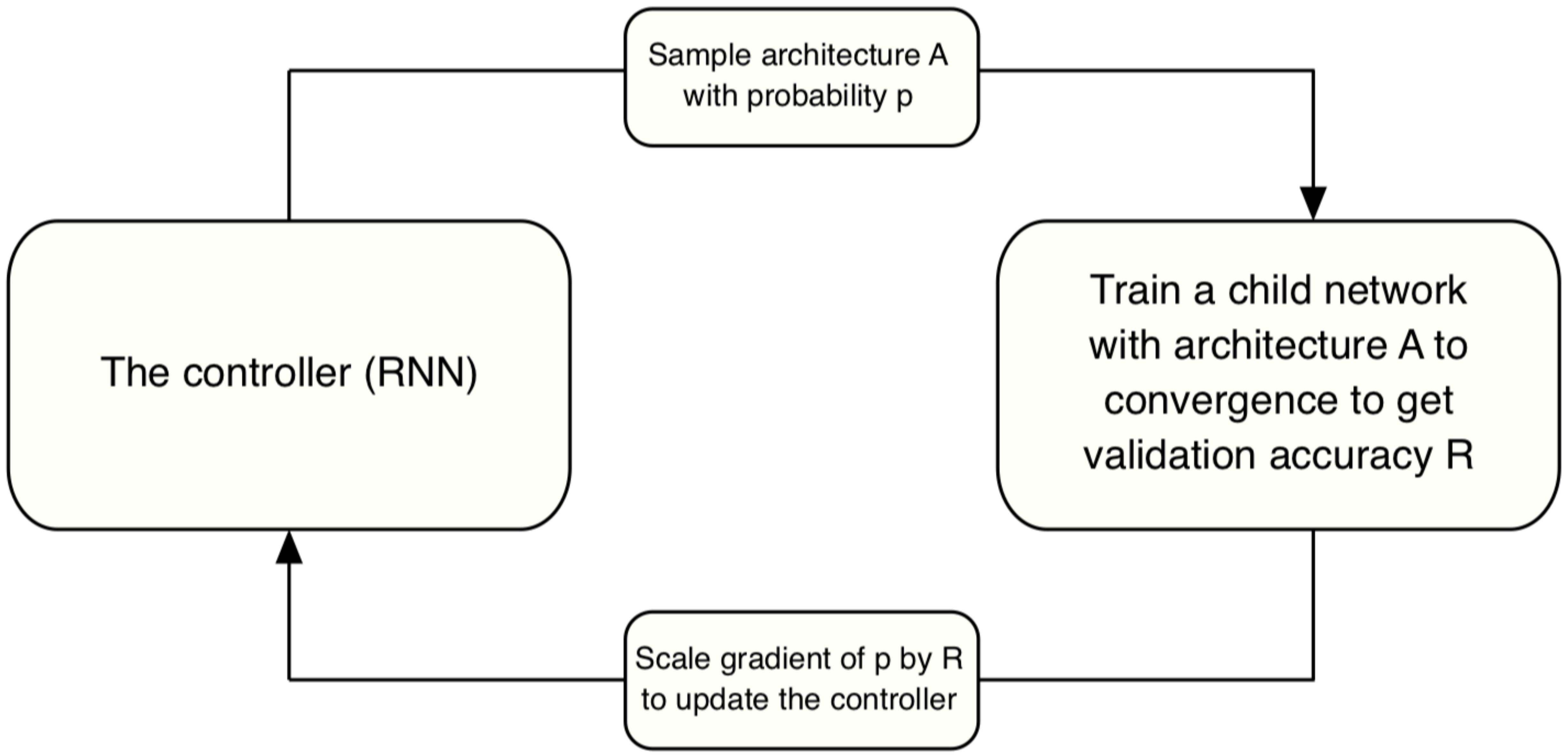
Automatic Architecture Search

Reinforcement Learning
Neuro-evolution
Bayesian Optimization
Monte Carlo Tree Search
…

NASNet



Normal Cell *Reduction Cell*



Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [29]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (5 @ 1538)	**299×299**	**10.9 M**	**2.35 B**	**78.6**	**94.2**
Inception V3 [60]	299×299	23.8 M	5.72 B	78.8	94.4
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [58]	299×299	55.8 M	13.2 B	80.1	95.1
NASNet-A (7 @ 1920)	**299×299**	**22.6 M**	**4.93 B**	**80.8**	**95.3**

Everything is good, except the cost

Learning Transferable Architectures for Scalable Image Recognition

In this section, we describe our experiments with the method described above to learn convolutional cells. In summary, all architecture searches are performed using the CIFAR-10 classification task [31]. The controller RNN was trained using Proximal Policy Optimization (PPO) [51] by employing a global workqueue system for generating a pool of child networks controlled by the RNN. In our experiments, the pool of workers in the workqueue consisted of 500 GPUs.

The result of this search process over 4 days yields several candidate convolutional cells. We note that this search procedure is almost 7× faster than previous approaches [71] that took 28 days.[1] Additionally, we demonstrate below that the resulting architecture is superior in accuracy.

Figure 4 shows a diagram of the top performing Normal Cell and Reduction Cell. Note the prevalence of separable

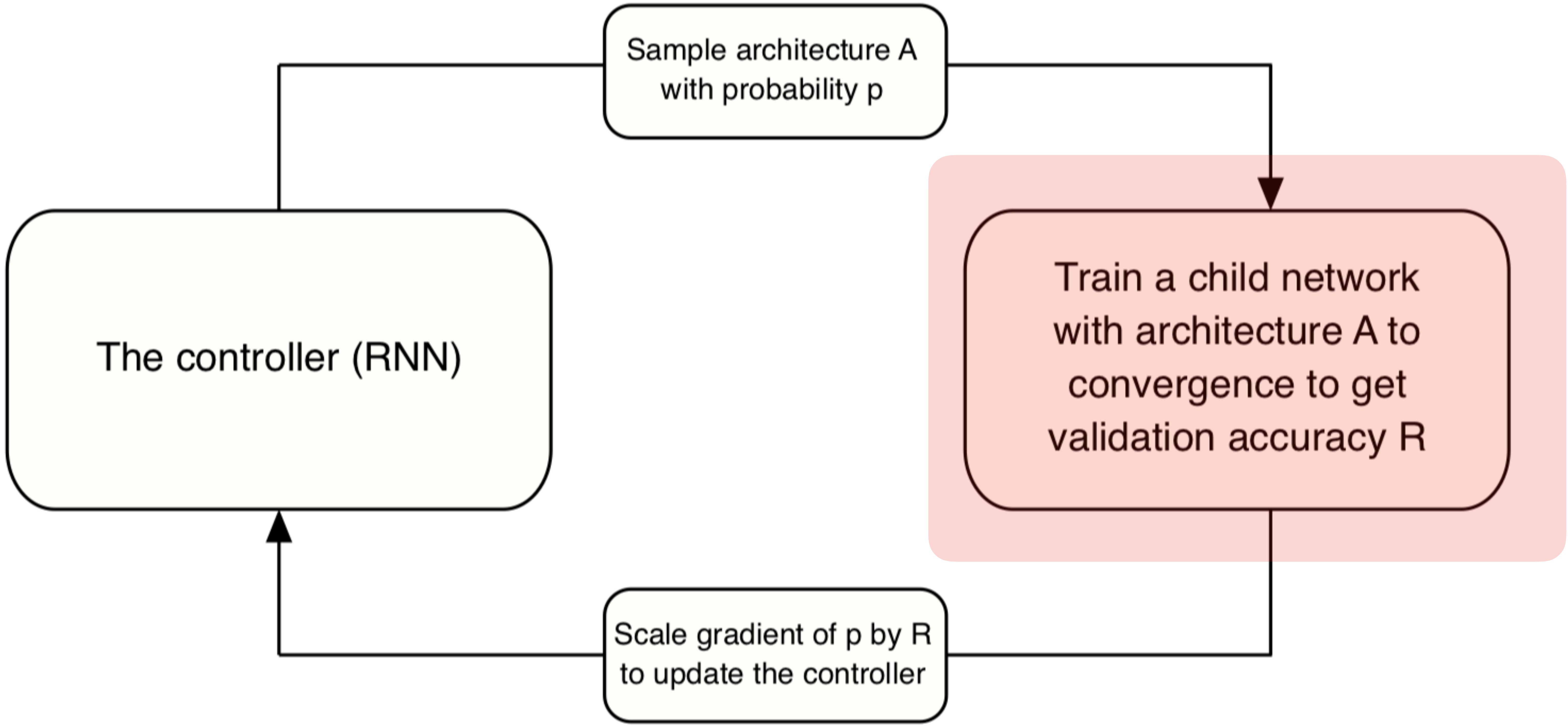
4 days * 24 hours * 500 GPUs = 48,000 GPU hours

Common Way: Proxy

- Search on a small dataset, then transfer to large one(s).
  - e.g., CIFAR -> ImageNet
- Search a subset(a single or few blocks), then repeats
- Train only a few epochs instead fully train the model.

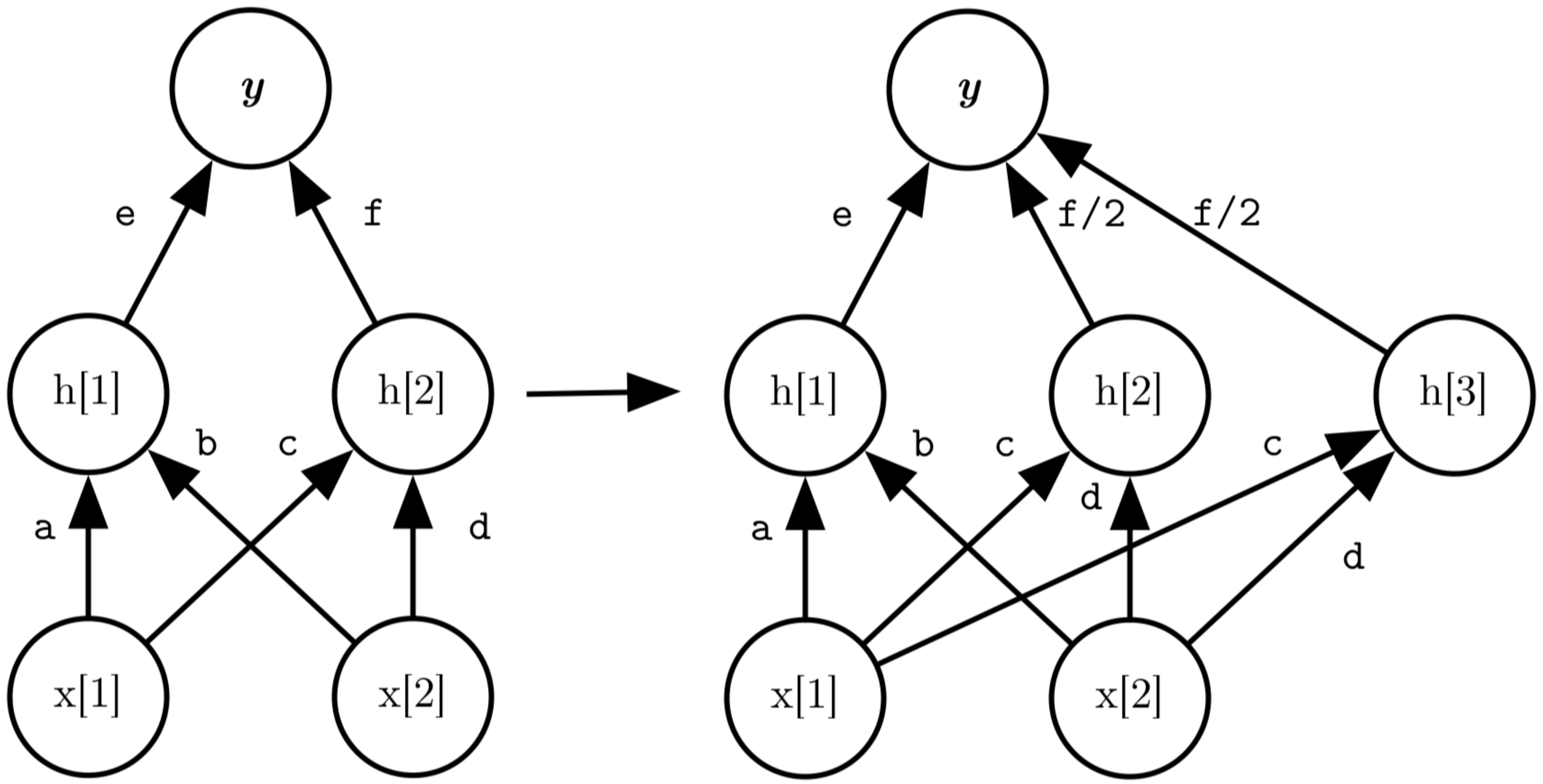
Proxy leads to sub-optimal!

Exploration on Efficient NAS

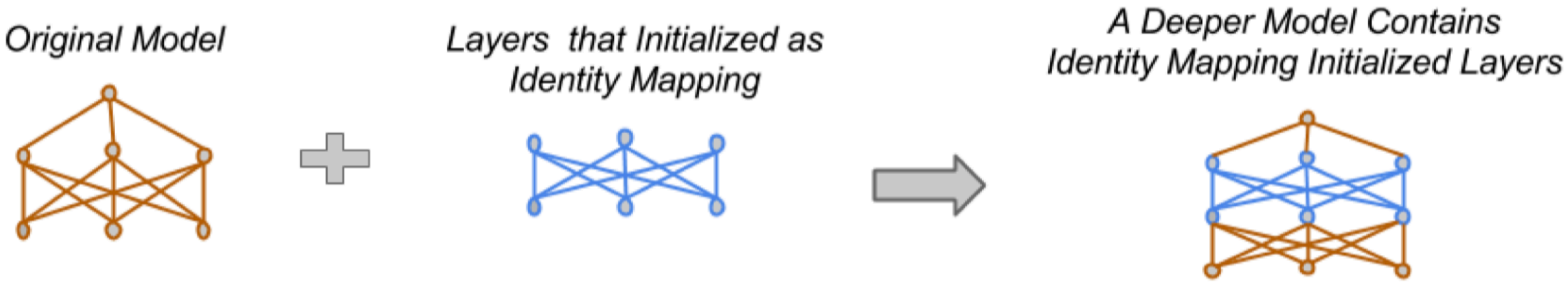


Efficient Architecture Search by Network Transformation

Net2Wider

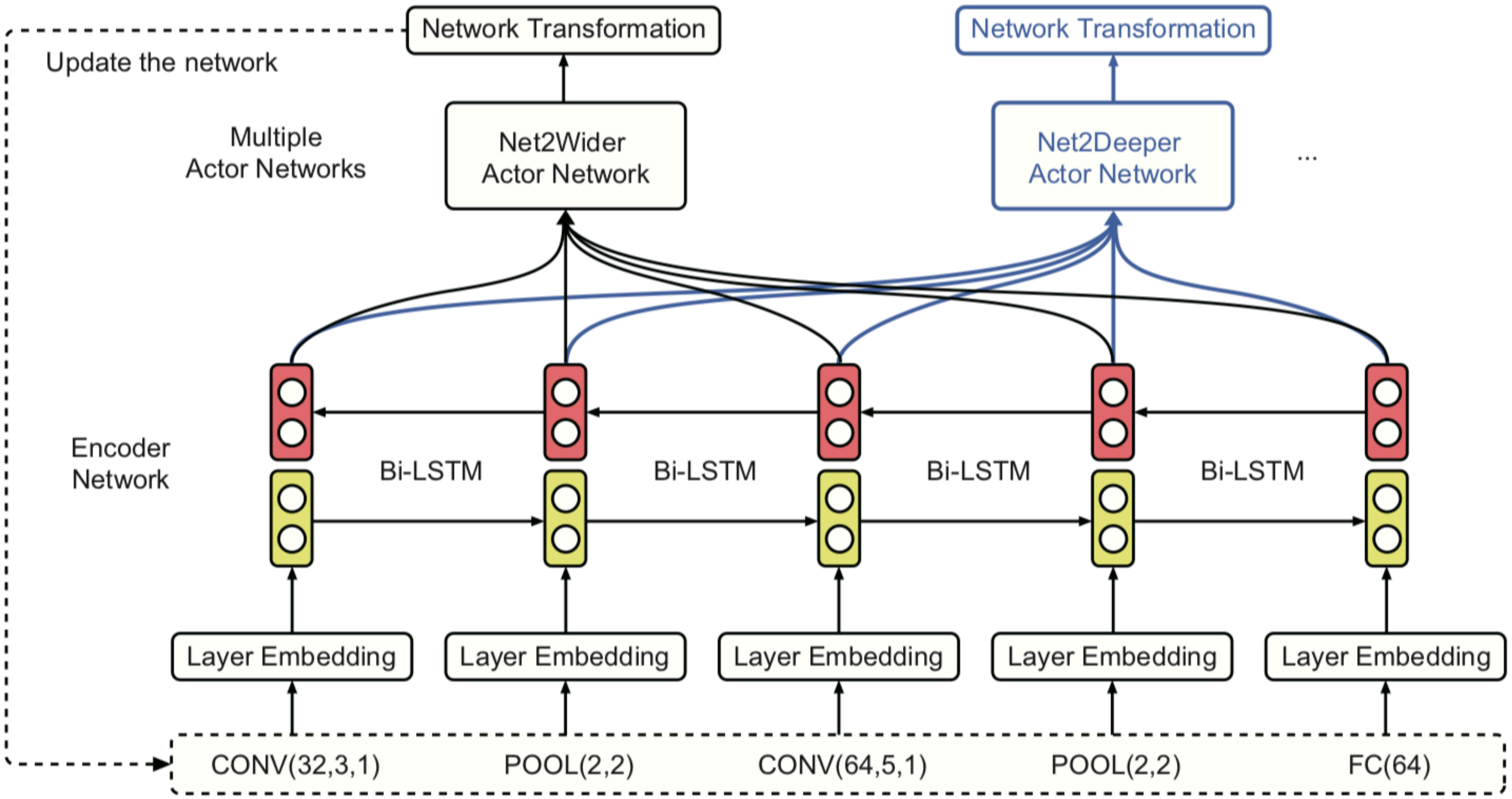


Net2Deeper

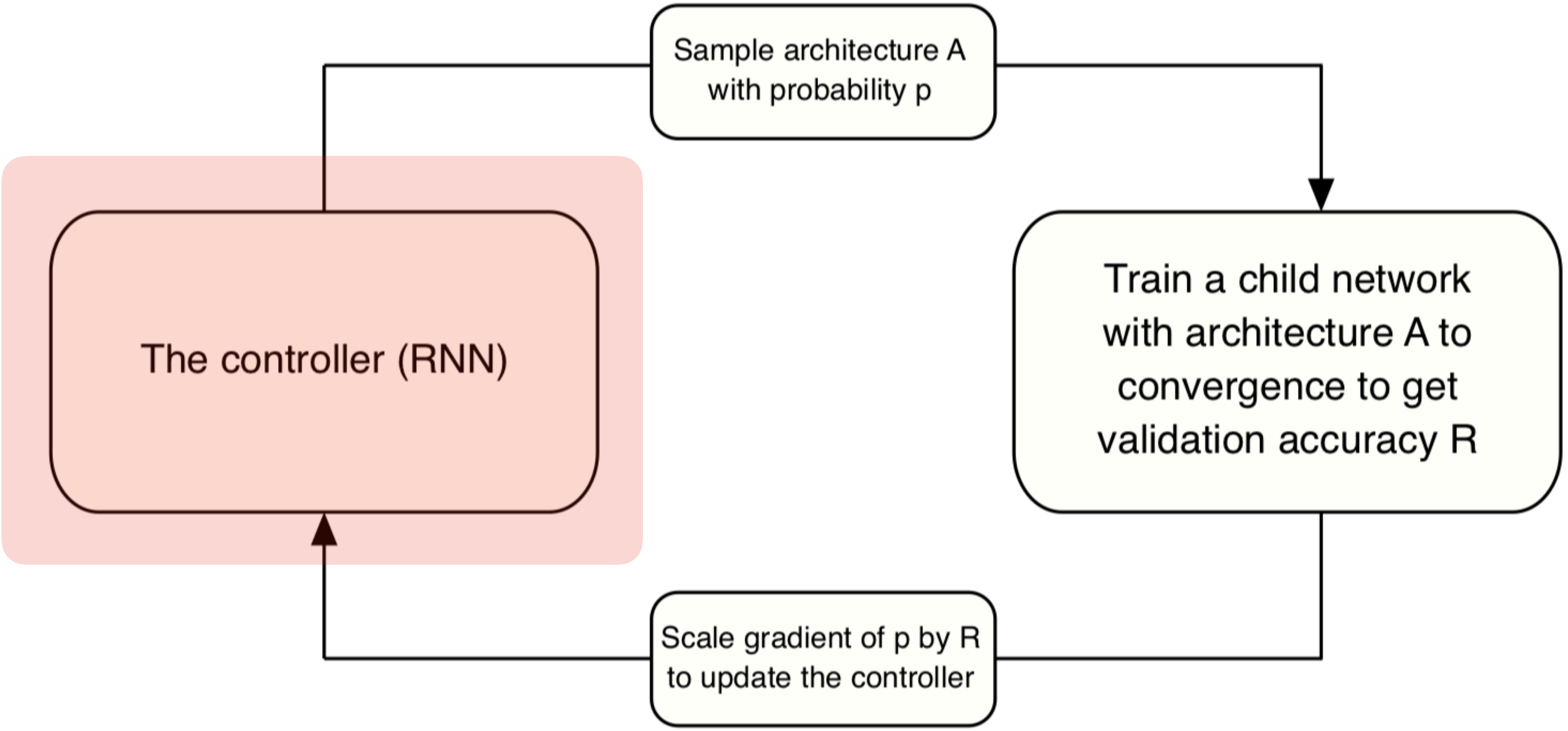


Efficient Architecture Search by Network Transformation

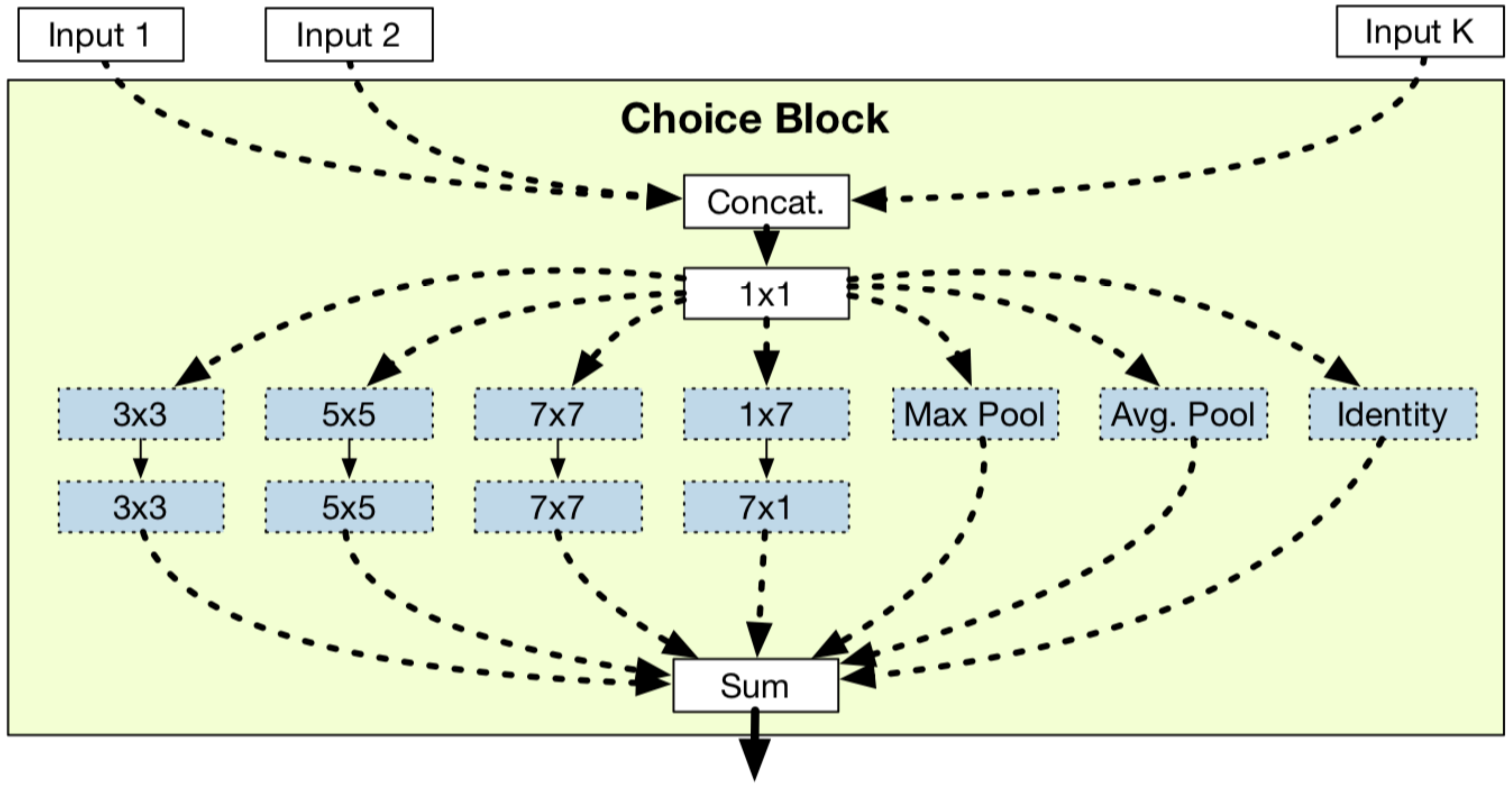
- Instead of sample a random layer, sample a equivalent transformation



Exploration on Efficient NAS



Understanding and Simplifying One-Shot Architecture Search

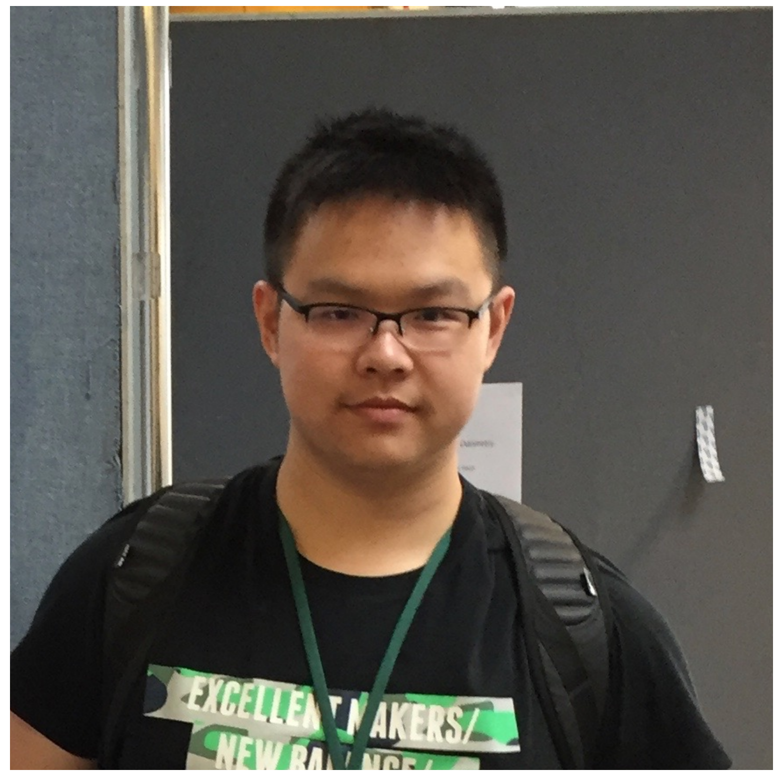


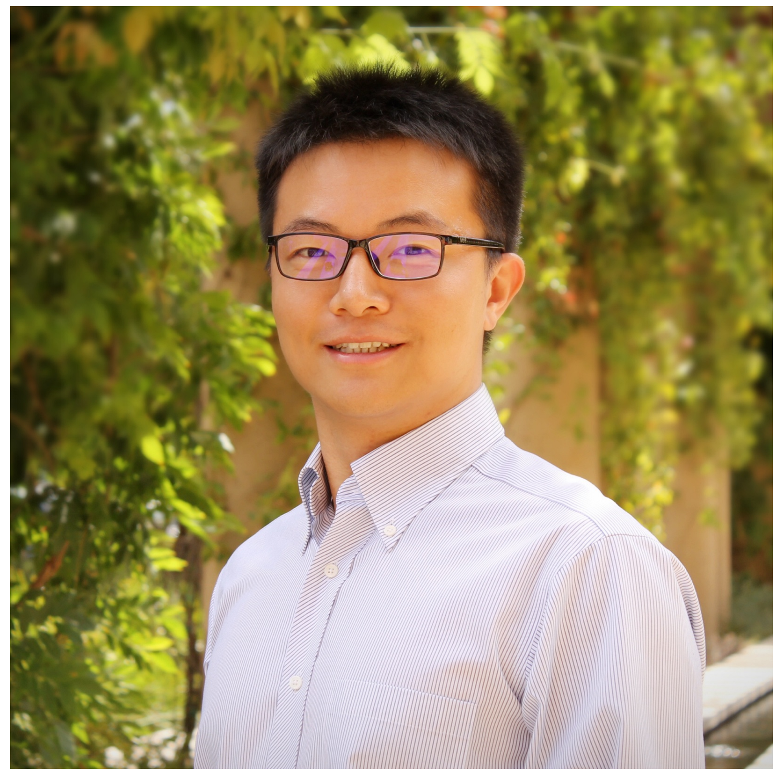
1. Train a larger network (with all candidates)
2. Sample a path, validate the performance.
3. Repeat step 2.
4. Choose the one with highest performance.

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware

Han Cai, Ligeng Zhu, Song Han

Massachusetts Institute of Technology

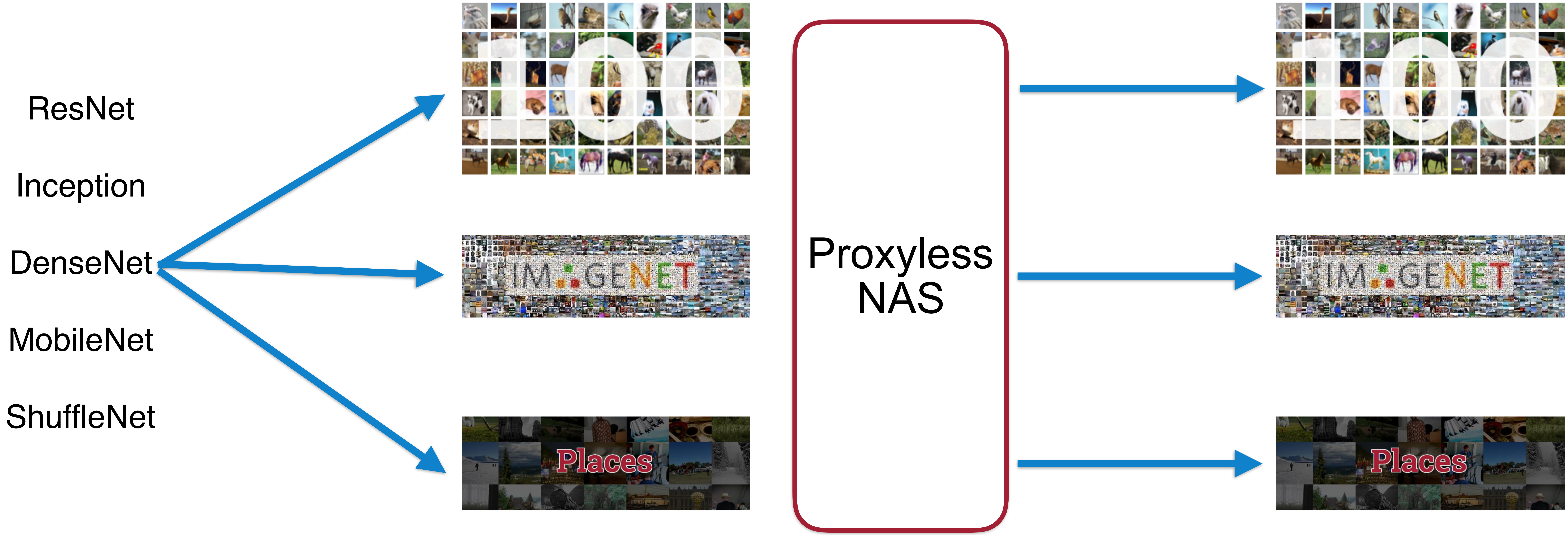




From General Design to Specialized CNN

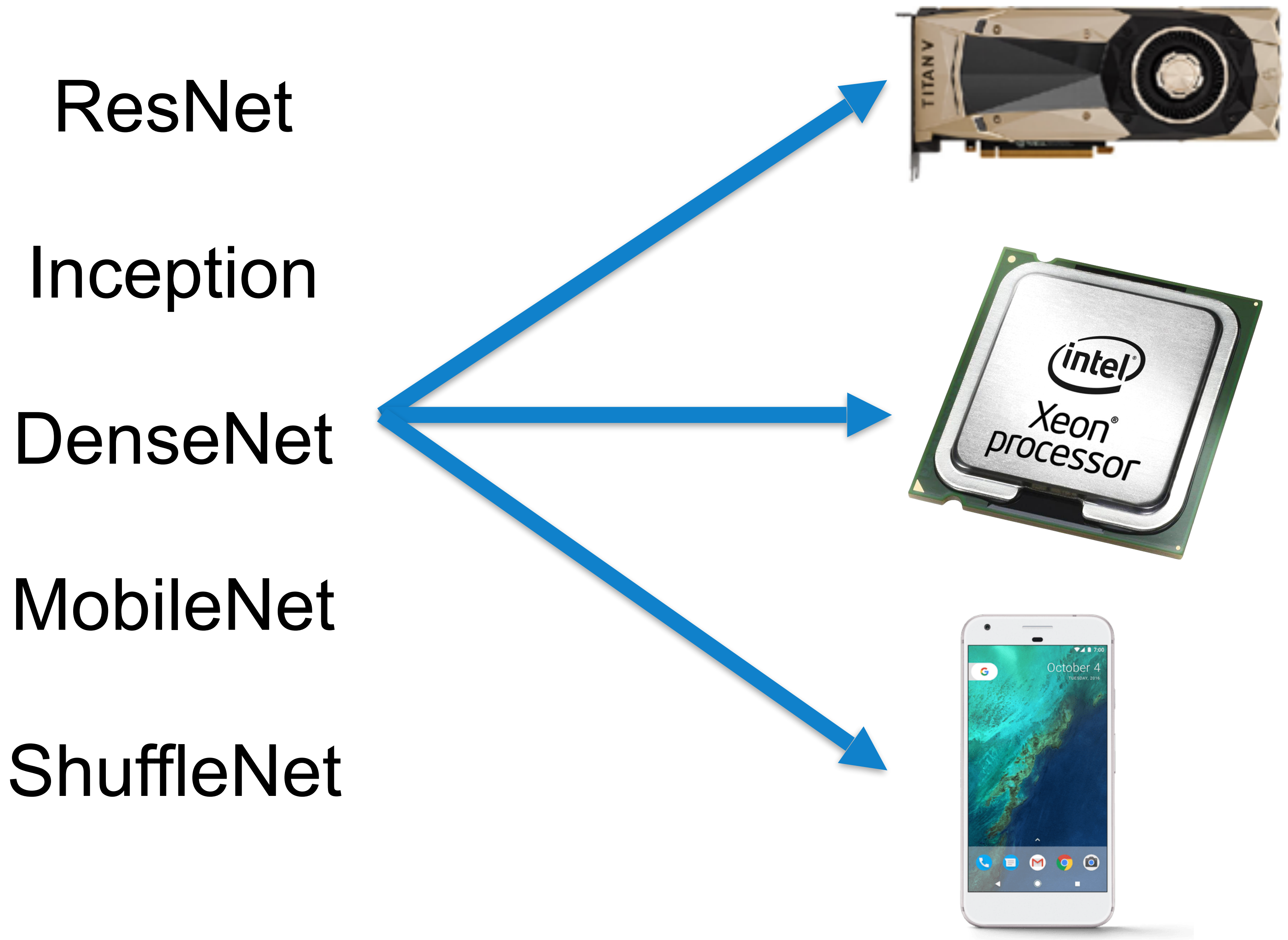
Previous Paradigm:
One CNN for all datasets.

Our Work:
Customize CNN for each dataset.

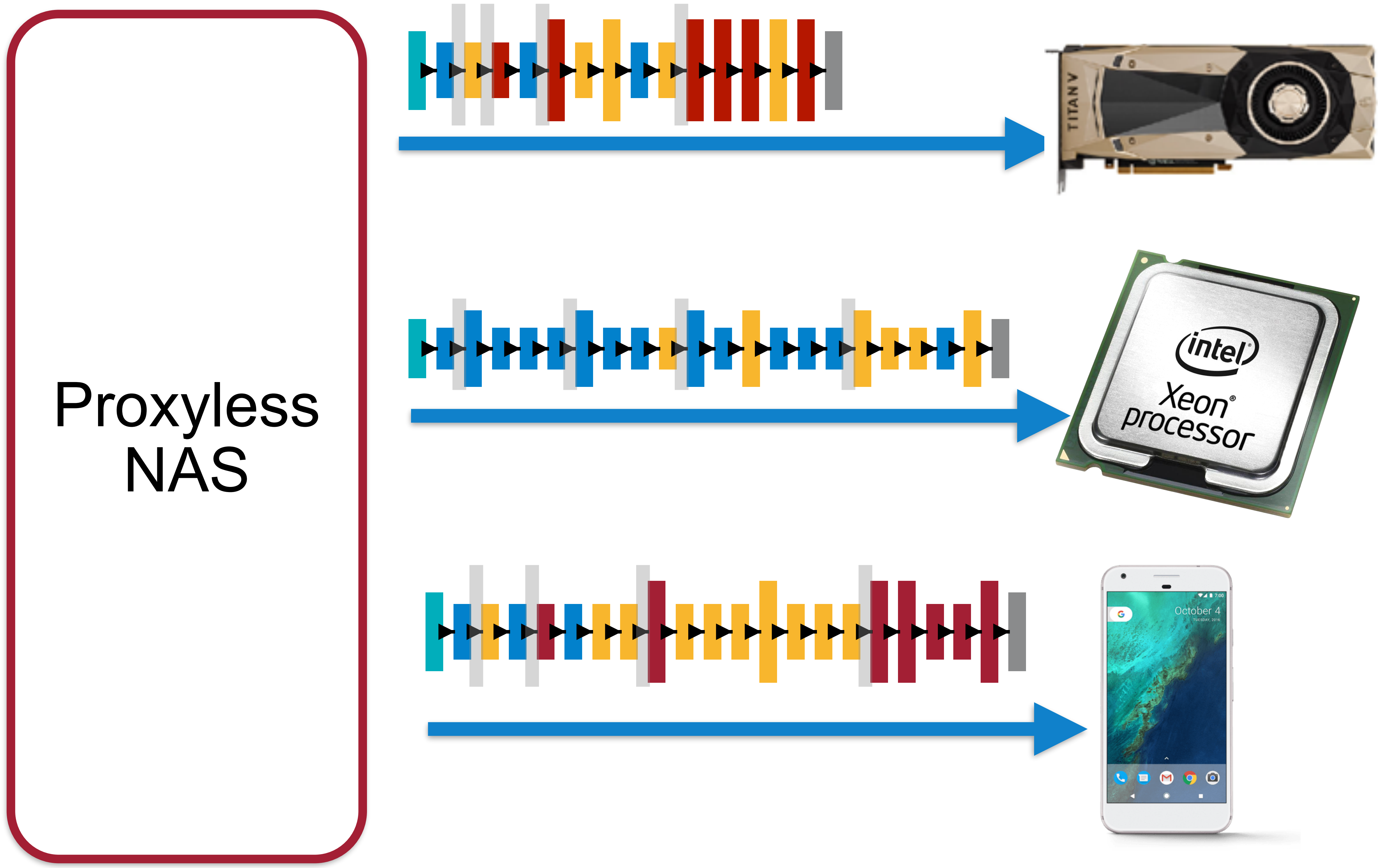


From General Design to Specialized CNN

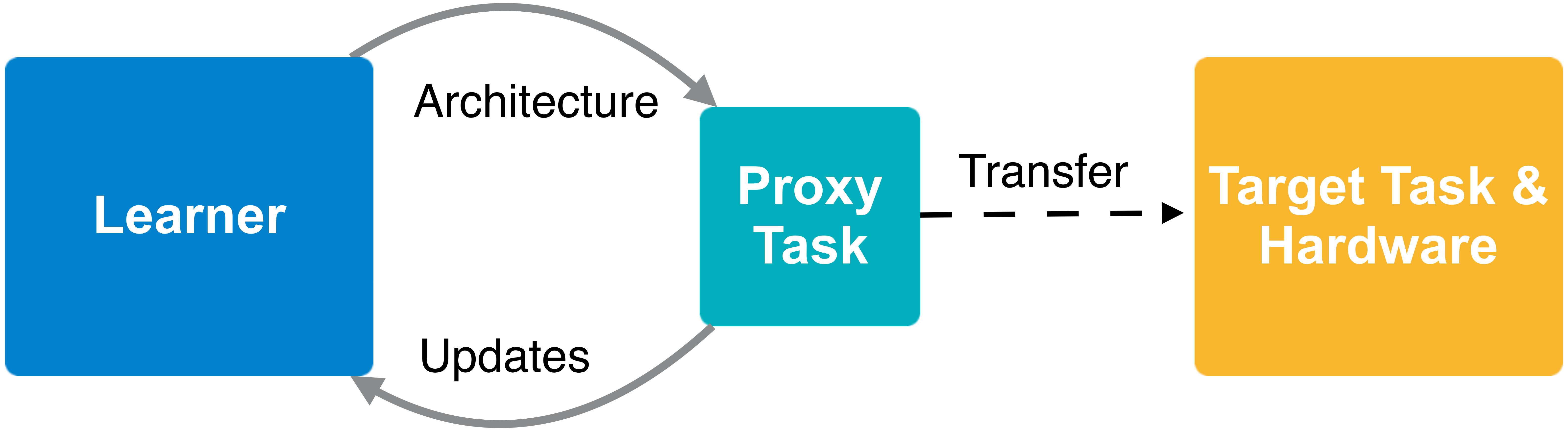
Previous Paradigm:
One CNN for all platforms.



Our Work:
Customize CNN for each platform.



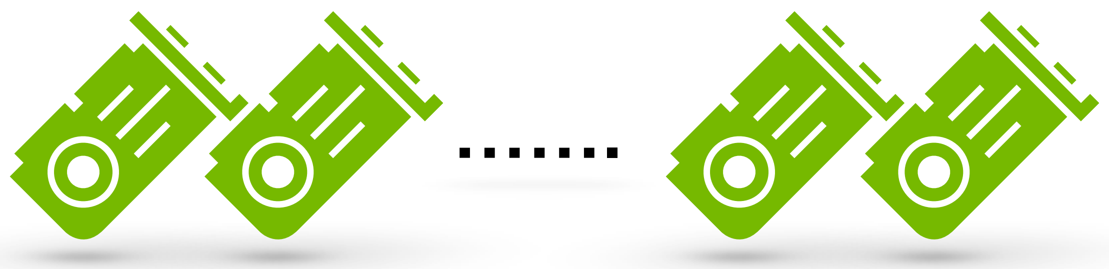
Conventional NAS: Computation Expensive



Current neural architecture search (NAS) is **VERY EXPENSIVE**.

- NASNet: 48,000 GPU hours ≈ 5 years on single GPU
- DARTS: 100Gb GPU memory* ≈ 9 times of modern GPU

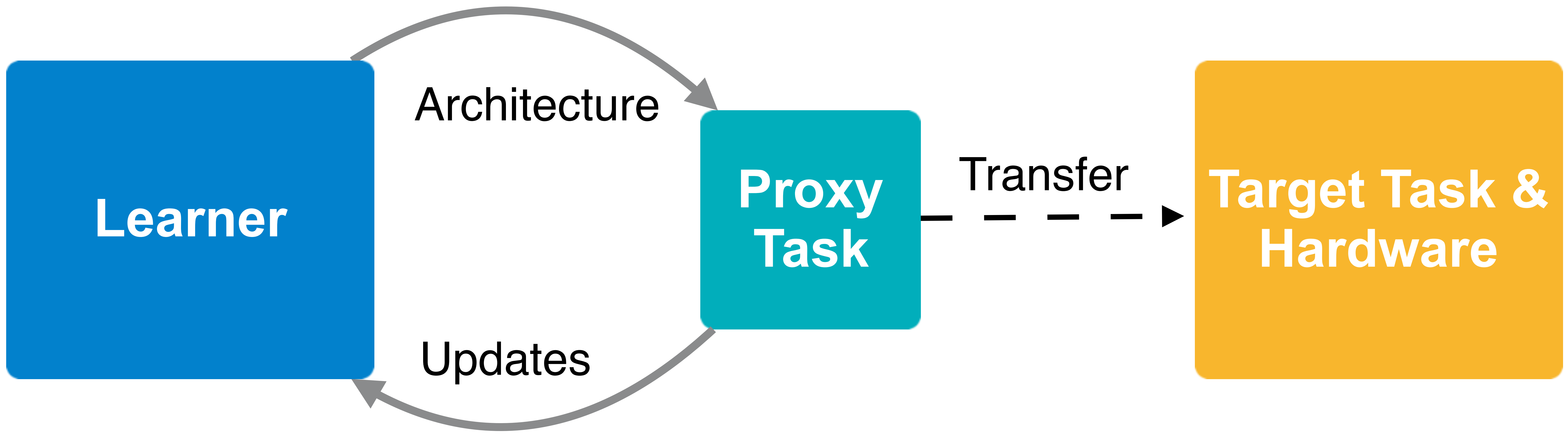
*if directly search on ImageNet, like us



Therefore, previous work have to utilize **proxy tasks:**

- CIFAR-10 -> ImageNet
- Small architecture space (e.g. low depth) -> large architecture space
- Fewer epochs training -> full training

Conventional NAS: proxy-based



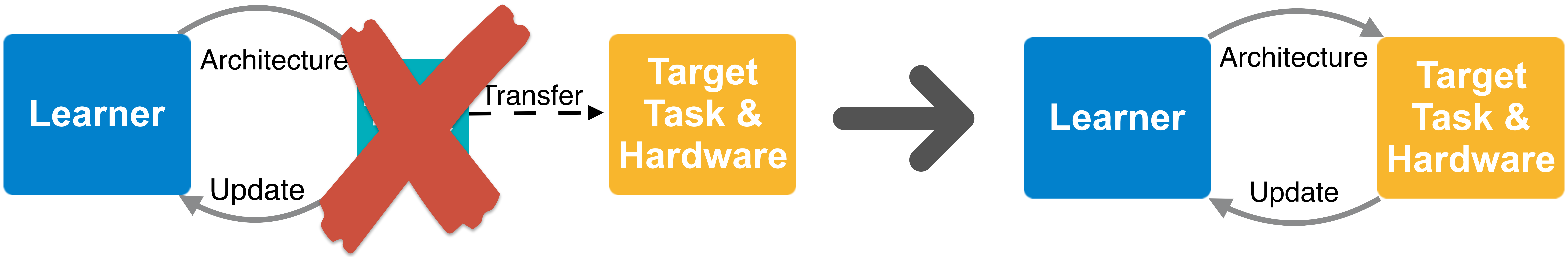
Proxies:

- CIFAR-10 -> ImageNet
- Small architecture space (e.g. low depth) -> large architecture space
- Fewer epochs training -> full training

Limitations of Proxy

- **Suboptimal** for the target task
- Blocks are forced to **share the same structure**.
- Cannot optimize for **specific hardware**.

Our Work: proxyless, save GPU hours by 200x



Goal: Directly learn architectures on the **target task** and **hardware**, while allowing all blocks to have different structures. We achieved by

1. Reducing the cost of NAS (GPU hours and memory) to the **same** level of regular training.
2. Cooperating **hardware feedback** (e.g. latency) into the search process.

To make NAS 200x more Efficient

Google, Facebook, NVIDIA

High-end GPU cluster

Many Engineers

AI research institutes:
Good weapon (GPU cluster)
Many Engineers

poor equipment, smart algorithm

poor weapon but smart students
Less GPUs but:
we have more efficient algorithm

Model Compression

Neural Architecture Search

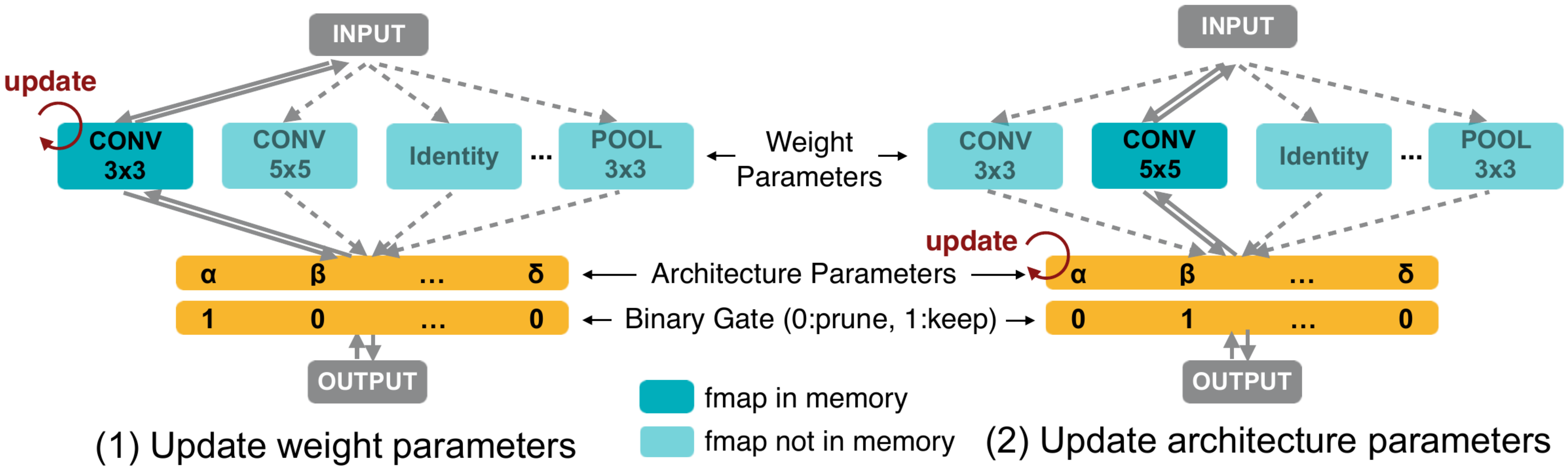
Pruning

Binarization

Save GPU hours

Save GPU Memory

Save GPU Hours



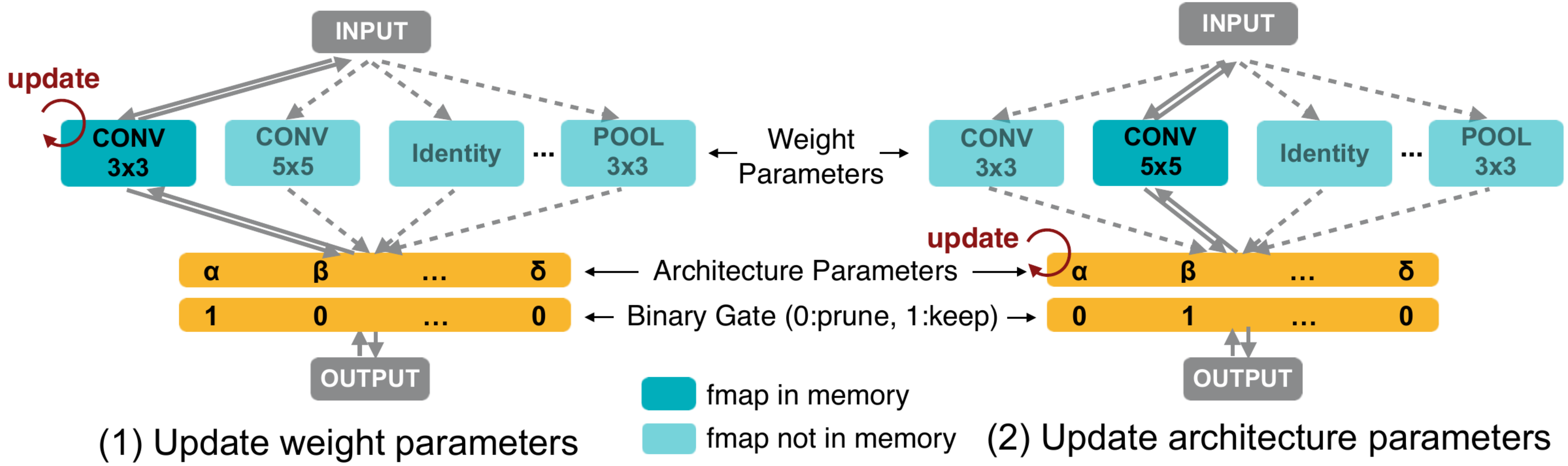
Pruning redundant paths based on architecture parameters

Simplify NAS to be a **single training process** of a over-parameterized network.

No meta controller. Stand on the shoulder of giants.

Build the cumbersome network **with all candidate paths**

Save GPU Memory

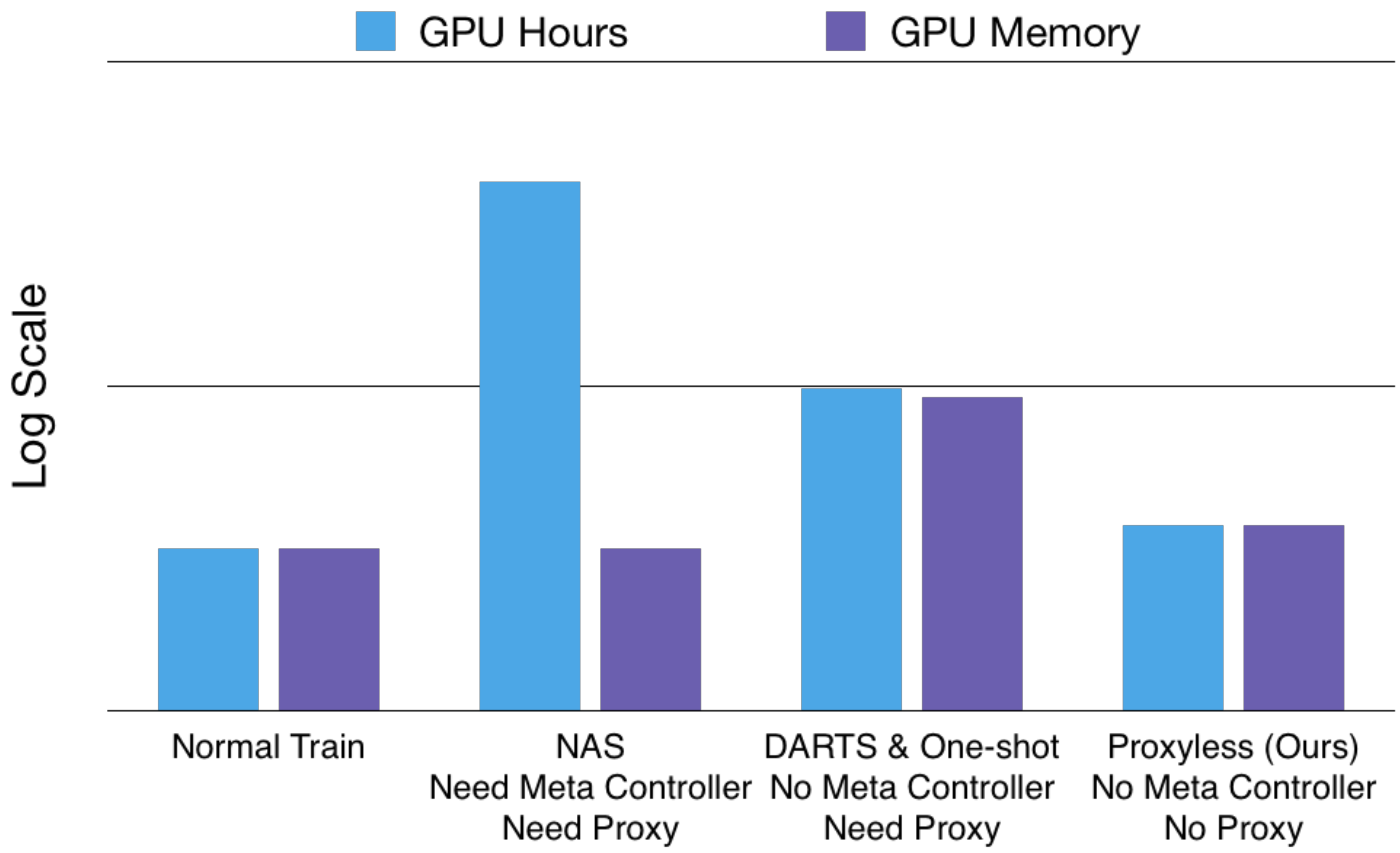


Binarize the architecture parameters and allow only **one path of activation to be active** in memory at run-time.

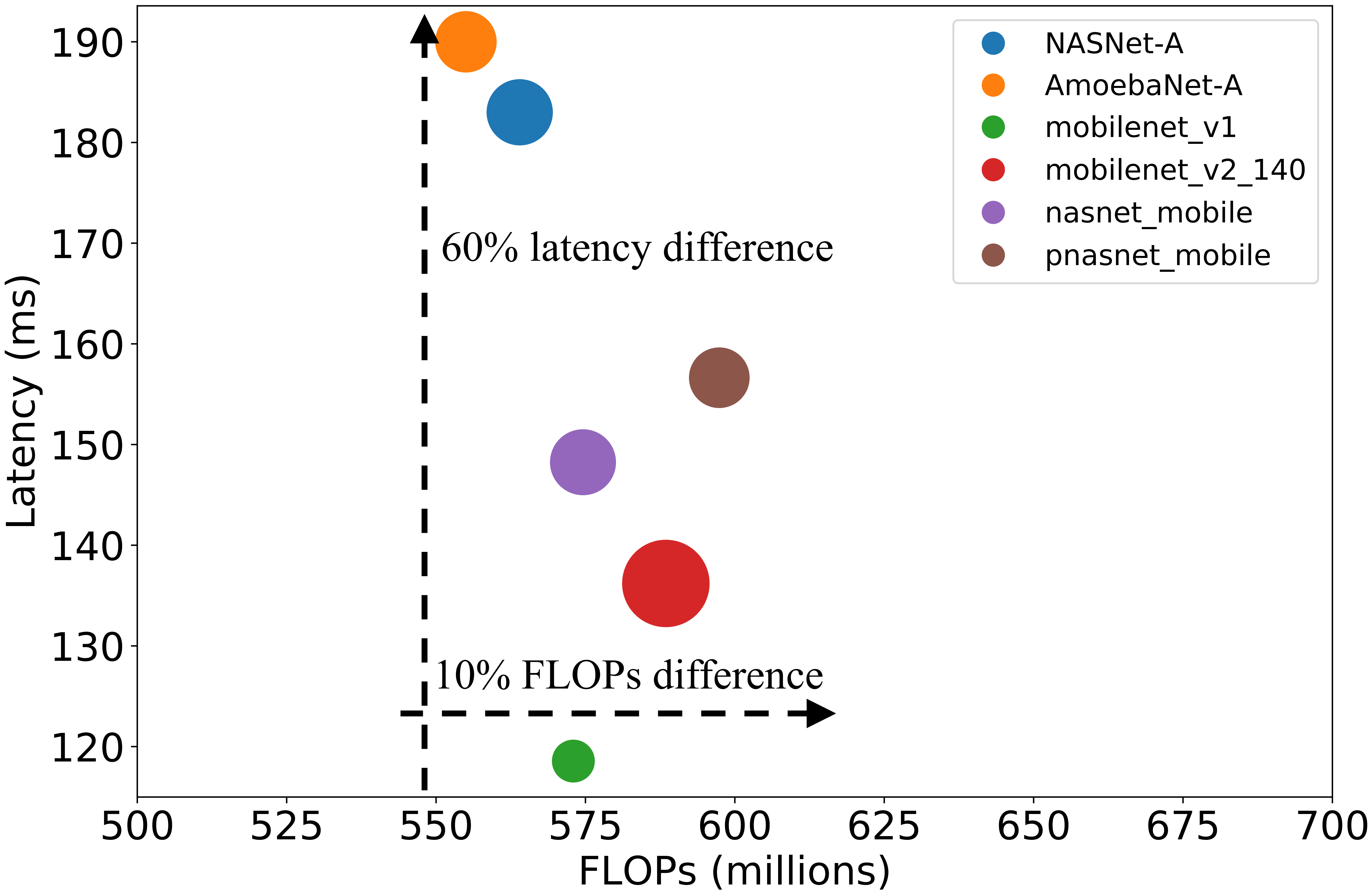
We propose **gradient-based** and **RL** methods to update the **binarized parameters**.

Thereby, the memory footprint reduces from **O(N) to O(1)**.

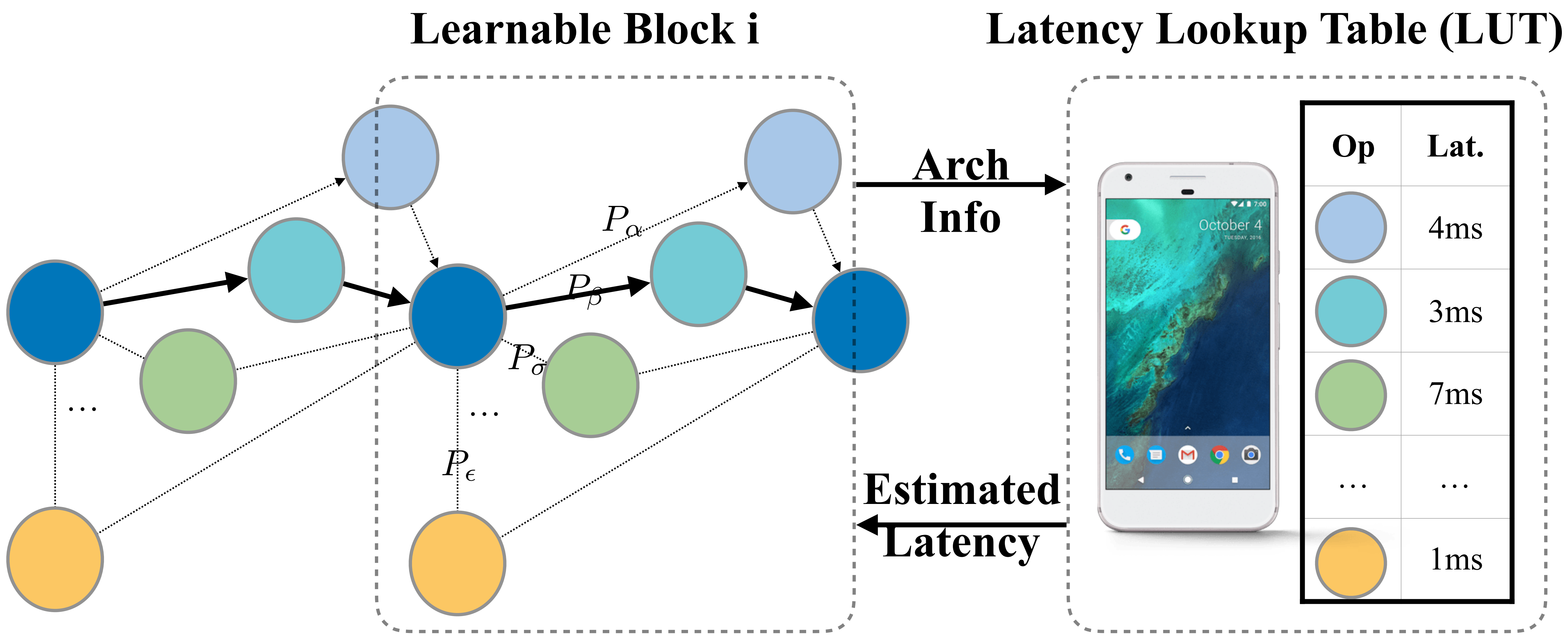
Search Cost


GPU Hours
GPU Memory
Log Scale
Normal Train
NAS
Need Meta Controller
Need Proxy
DARTS & One-shot
No Meta Controller
Need Proxy
Proxyless (Ours)
No Meta Controller
No Proxy

FLOPs != Latency



Hardware-aware Constraints



Query the latency from the lookup table.

Gradient Based

$$
\begin{aligned}
\mathbb{E}[\mathrm{LAT_i}] = & P_\alpha \times F(\mathrm{conv\_3x3}) + \\
& P_\beta \times F(\mathrm{conv\_5x5}) + \\
& P_\sigma \times F(\mathrm{identity}) + \\
& \ldots\ldots \\
& P_\zeta \times F(\mathrm{pool\_3x3})
\end{aligned}
$$

$$
\mathrm{E}[\mathrm{LAT}] = \sum_i^N \mathrm{E}[\mathrm{LAT}_i]
$$

$$
Loss = Loss_{CE} + \lambda_1 ||w||_2^2 + \lambda_2 \mathrm{E}[\mathrm{LAT}]
$$

Reinforce Based

$$
J(\alpha) = \mathbb{E}_{g\sim\alpha}[R(\mathcal{N}_g)] = \sum_i p_i R(\mathcal{N}(e = o_i)),
$$

$$
\begin{aligned}
\nabla_\alpha J(\alpha) &= \sum_i R(\mathcal{N}(e = o_i)) \nabla_\alpha p_i = \sum_i R(\mathcal{N}(e = o_i)) p_i \nabla_\alpha \log(p_i), \\
&= \mathbb{E}_{g\sim\alpha}[R(\mathcal{N}_g) \nabla_\alpha \log(p(g))] \approx \frac{1}{M} \sum_{i=1}^M R(\mathcal{N}_{g^i}) \nabla_\alpha \log(p(g^i)),
\end{aligned}
$$

Results: ProxylessNAS on CIFAR-10

Model	Params	Test error
DenseNet-BC (Huang et al., 2017)	25.6M	3.46
PyramidNet (Han et al., 2017)	26.0M	3.31
Shake-Shake + c/o (DeVries & Taylor, 2017)	26.2M	2.56
PyramidNet + SD (Yamada et al., 2018)	26.0M	2.31
ENAS + c/o (Pham et al., 2018)	4.6M	2.89
DARTS + c/o (Liu et al., 2018c)	3.4M	2.83
NASNet-A + c/o (Zoph et al., 2018)	27.6M	2.40
PathLevel EAS + c/o (Cai et al., 2018b)	14.3M	2.30
AmoebaNet-B + c/o (Real et al., 2018)	34.9M	2.13
Proxyless-R + c/o (ours)	5.8M	2.30
Proxyless-G + c/o (ours)	5.7M	**2.08**

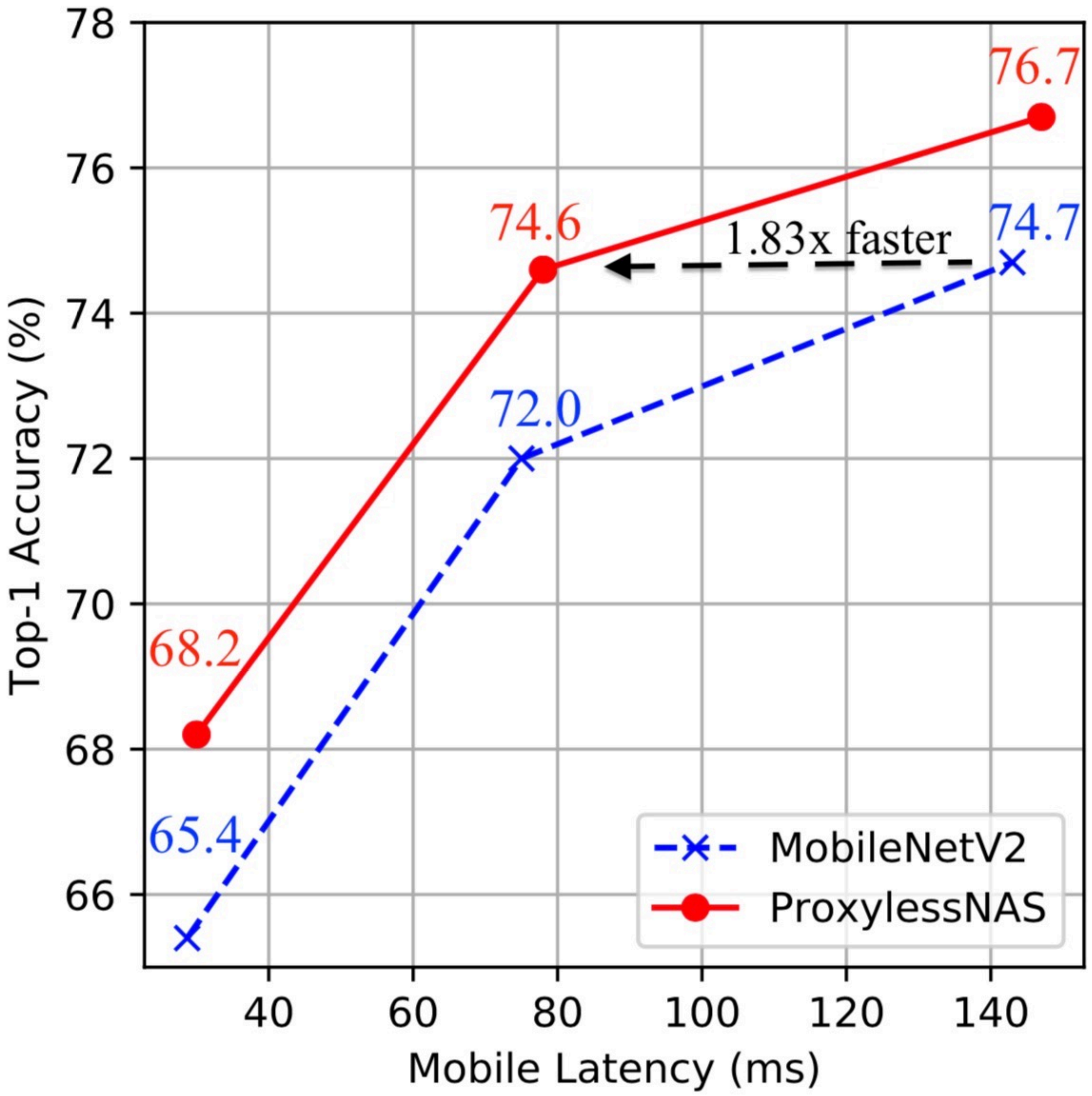
- Directly explore a huge space: 54 distinct blocks and possible architectures
- State-of-the-art test error with 6X fewer params (Compared to AmeobaNet-B)

Results: Proxyless-NAS on ImageNet, GPU Platform

Model	Top-1	Top-5	GPU latency
MobileNetV2 (Sandler et al., 2018)	72.0	91.0	6.1ms
ShuffleNetV2 (1.5) (Ma et al., 2018)	72.6	-	7.3ms
ResNet-34 (He et al., 2016)	73.3	91.4	8.0ms
NASNet-A (Zoph et al., 2018)	74.0	91.3	38.3ms
DARTS (Liu et al., 2018c)	73.1	91.0	-
MnasNet (Tan et al., 2018)	74.0	91.8	6.1ms
Proxyless (GPU)	**75.1**	**92.5**	**5.1ms**

When targeting GPU platform, the accuracy is further improved to 75.1%.
3.1% higher than MobilenetV2.

Results: ProxylessNAS on ImageNet, Mobile Platform



- With >74.5% top-1 accuracy, ProxylessNAS is **1.8x faster** than MobileNet-v2, the current industry standard.

ProxylessNAS for Hardware Specialization


Model	Top-1	GPU	CPU	Mobile
Specialized for GPU	75.1	**5.1ms**	204.9ms	124ms
Specialized for CPU	75.3	7.4ms	**138.7ms**	116ms
Specialized for Mobile	74.6	7.2ms	164.1ms	**78ms**

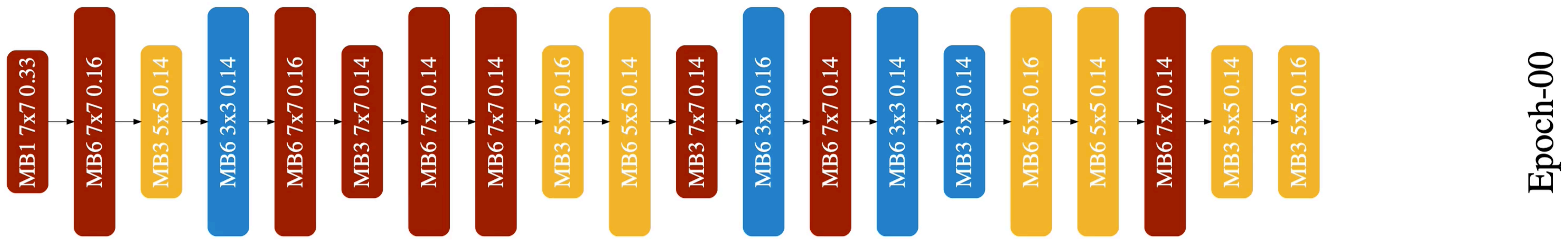
Results: ProxylessNAS on ImageNet, Mobile Platform

	Model	Top-1	Latency	Hardware Aware	No Proxy	No Repeat	Search Cost
Manually Designed	MobilenetV1	70.6	113ms	-	-	x	-
	MobilenetV2	72.0	75ms	-	-	x	-
NAS	NASNet-A	74.0	183ms	x	x	x	48000
	AmoebaNet-A	74.4	190ms	x	x	x	75600
	MNasNet	74.0	76ms	yes	x	x	40000
ProxylessNAS	ProxylessNAS-G	71.8	83ms	yes	yes	yes	200
	ProxylessNAS-G + LL	74.2	79ms	yes	Yes	yes	200
	ProxylessNAS-R	74.6	78ms	yes	Yes	yes	200
	ProxylessNAS-R + MIXUP	75.1	78ms	yes	yes	yes	200

ProxylessNAS achieves state-of-the art accuracy (%) on ImageNet (under mobile latency constraint ≤ 80ms) with 200× less search cost in GPU hours. “LL” indicates latency regularization loss.

The History of Architectures

(1) The history of finding efficient Mobile model

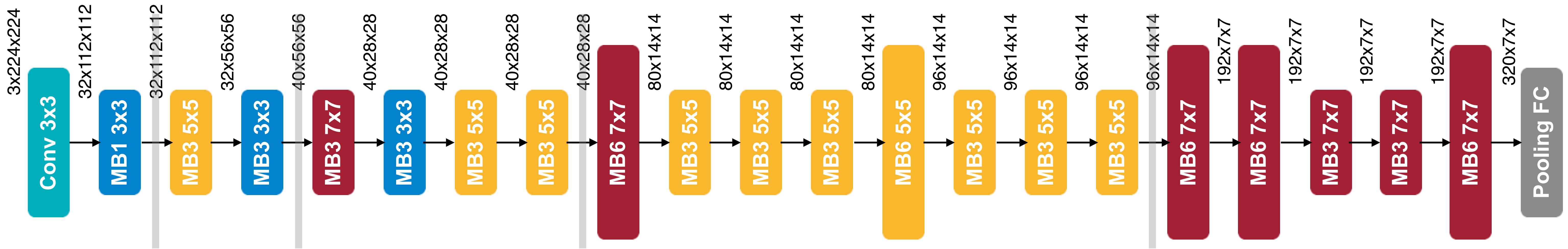


(2) The history of finding efficient CPU model

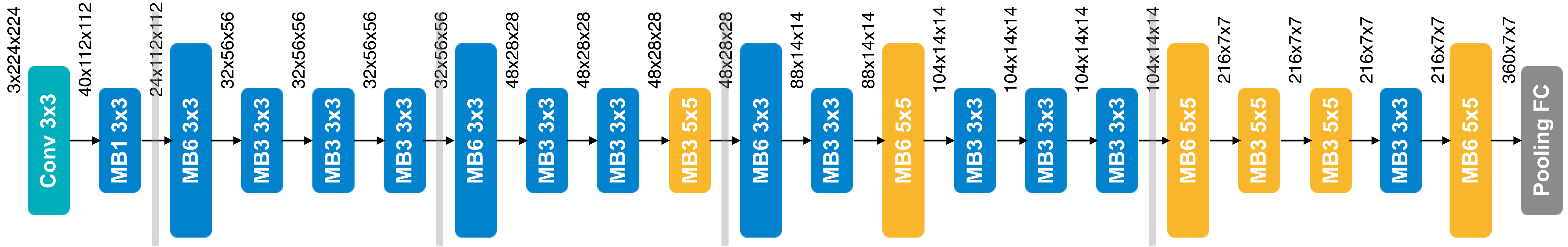
(3) The history of finding efficient GPU model

https://hanlab.mit.edu/files/proxylessNAS/visualization.mp4

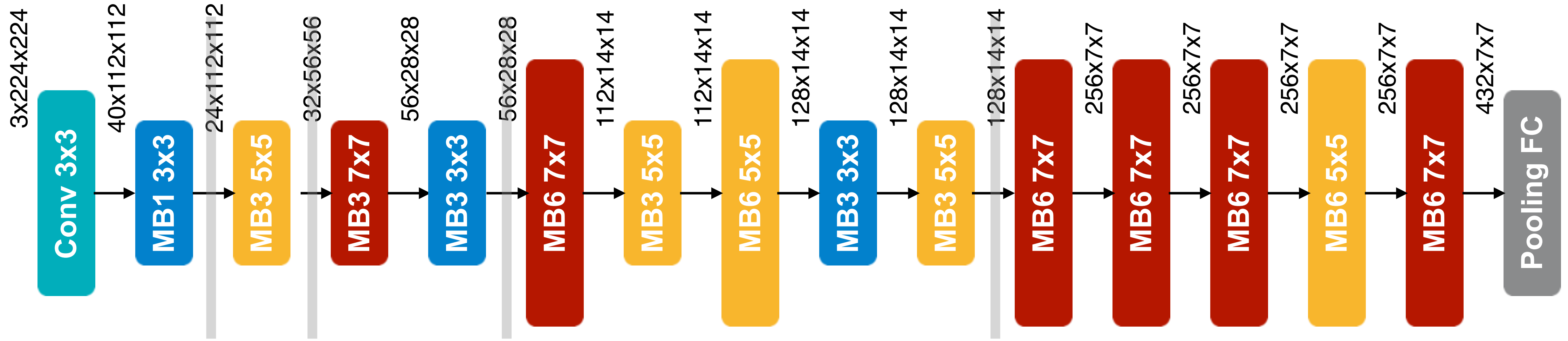
Detailed Architectures



(1) Efficient mobile architecture found by Proxy-less NAS.

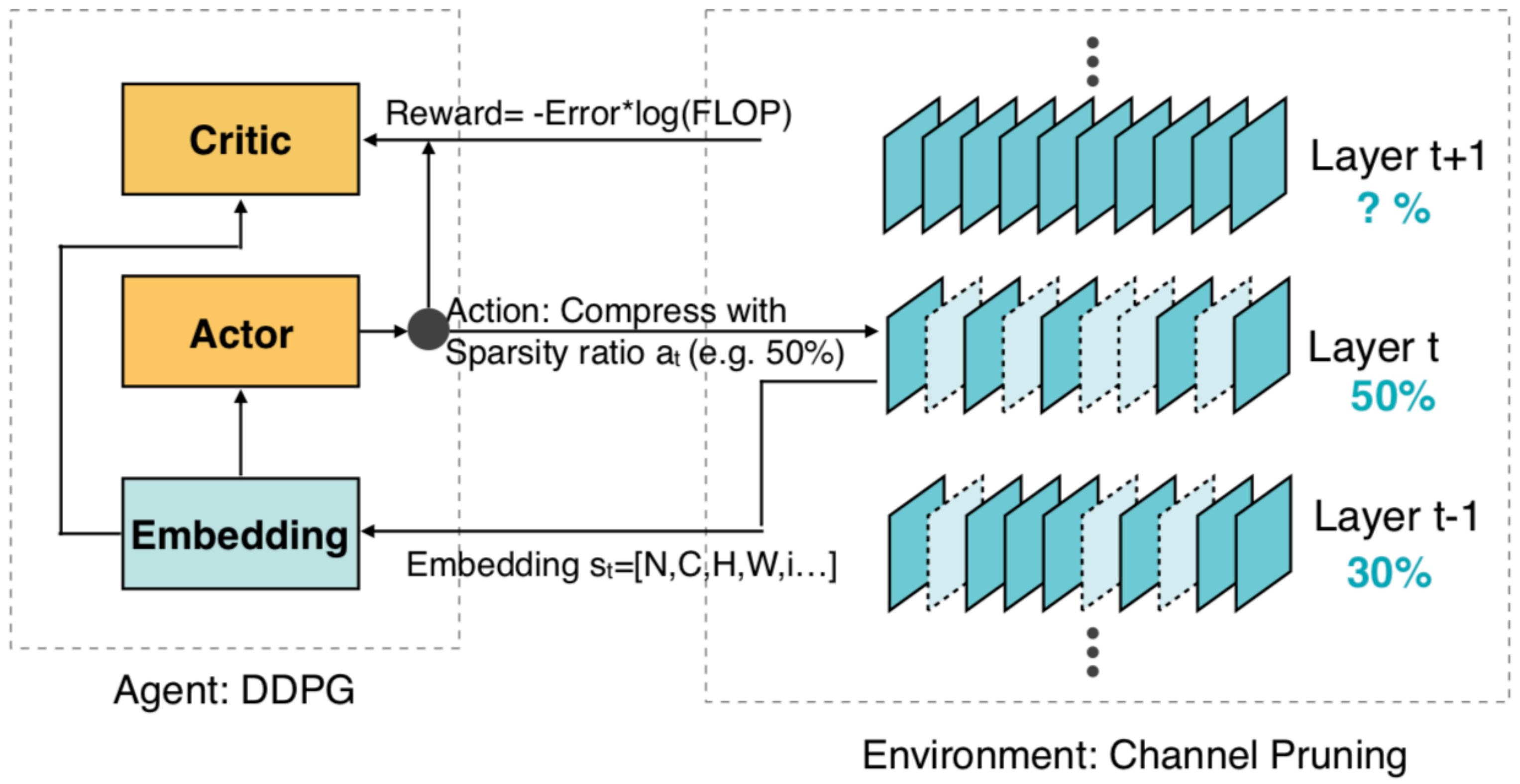


(2) Efficient CPU architecture found by Proxy-less NAS.



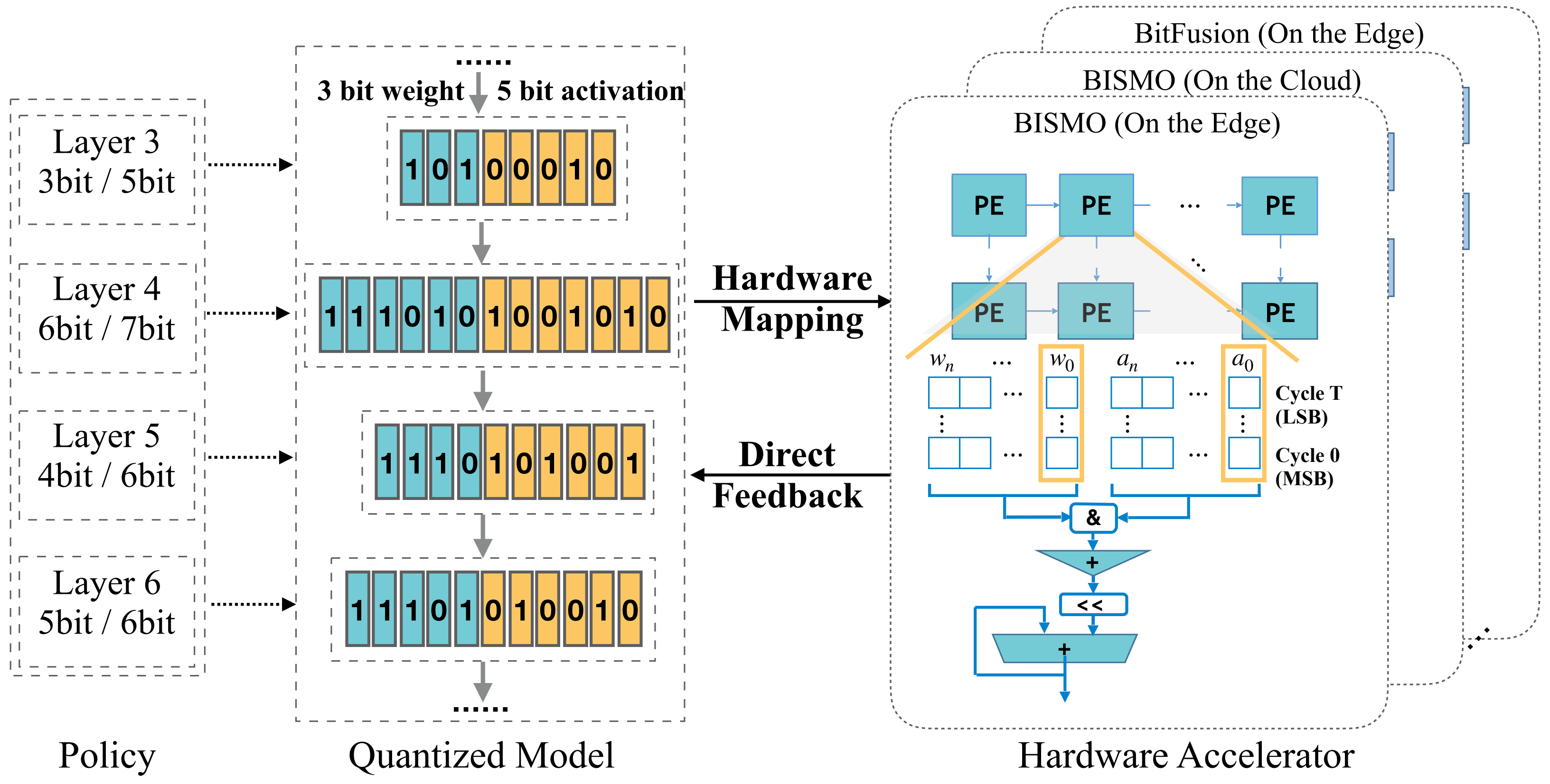
(3) Efficient GPU architecture found by Proxy-less NAS.

Machine learning expert
Hardware expert



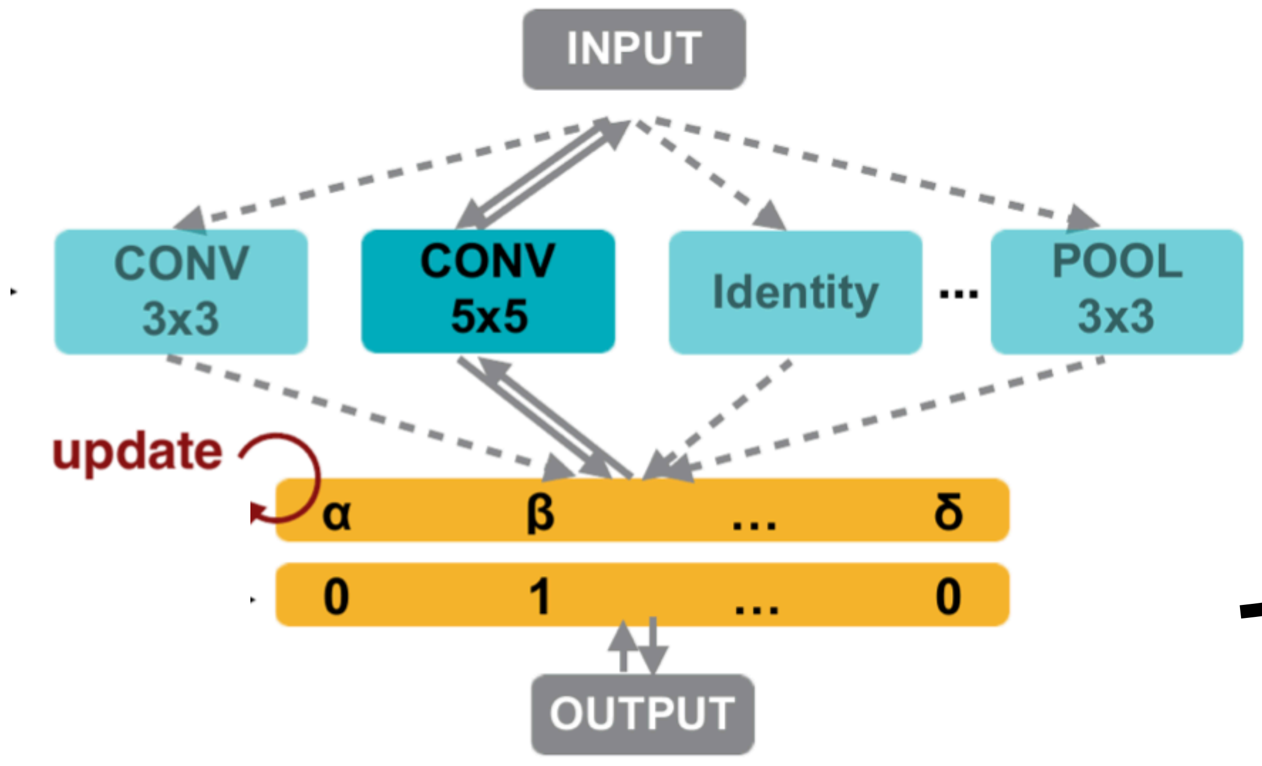
AMC: AutoML for Model Compression

He et al [ECCV'18]



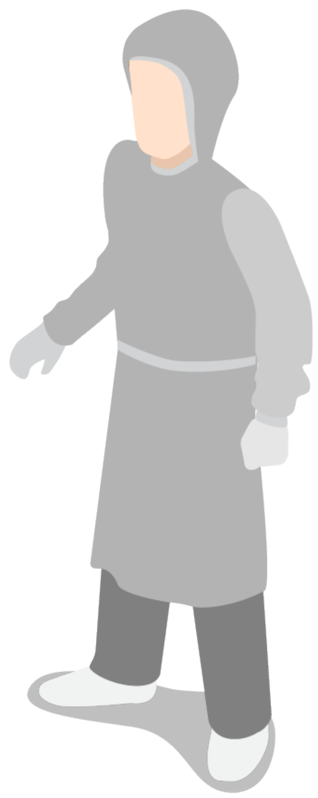
HAQ: Hardware-aware Automated Quantization

Wang et al [CVPR'19], oral



Proxyless Neural Architecture Search

Cai et al [ICLR'19]



+

Non expert

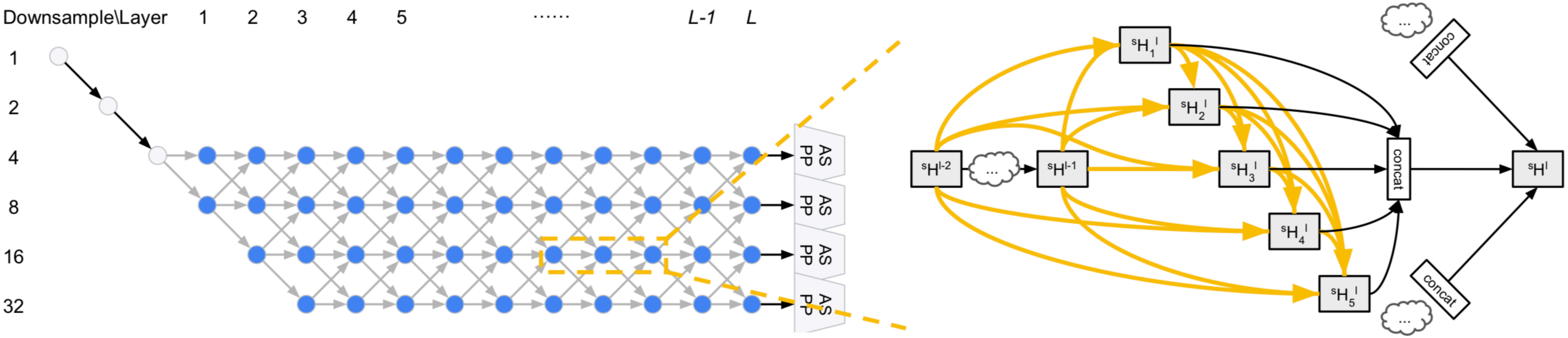
Hardware-Centric AutoML

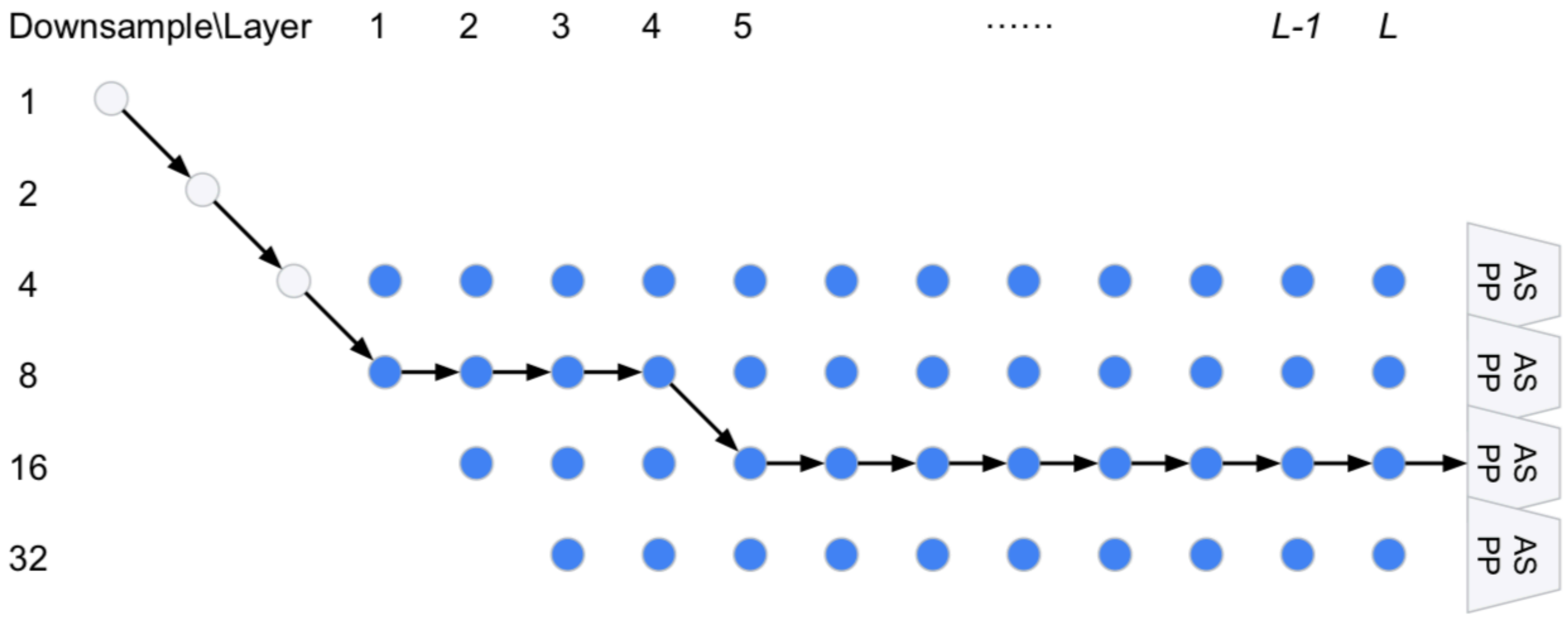
Embrace Open-source

- Our models are now released on Github with pre-trained weights.

```
# https://github.com/MIT-HAN-LAB/ProxylessNAS
from proxyless_nas import *
net = proxyless_cpu(pretrained=True)
net = proxyless_gpu(pretrained=True)
net = proxyless_mobile(pretrained=True)
```

AutoDeeplab

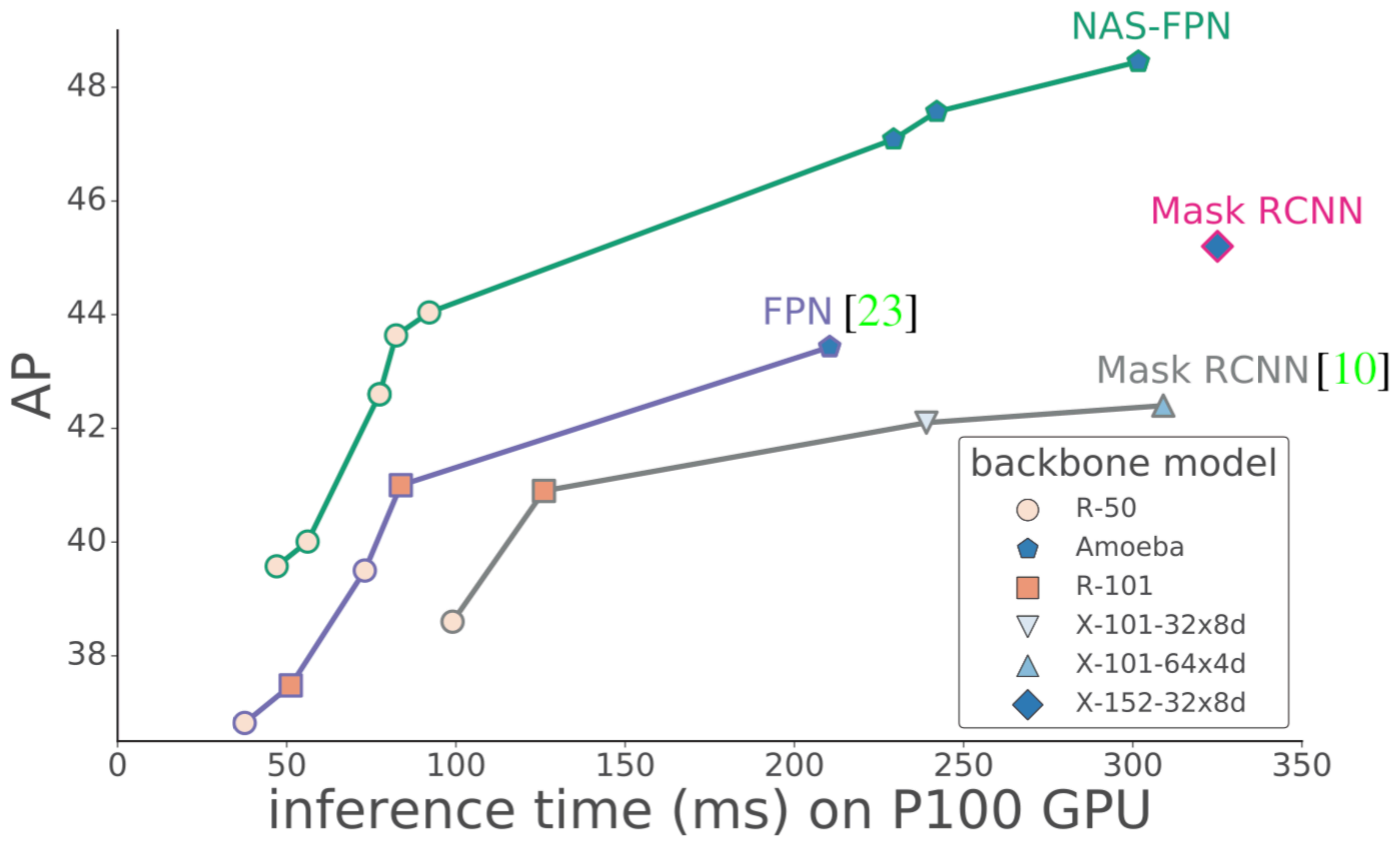


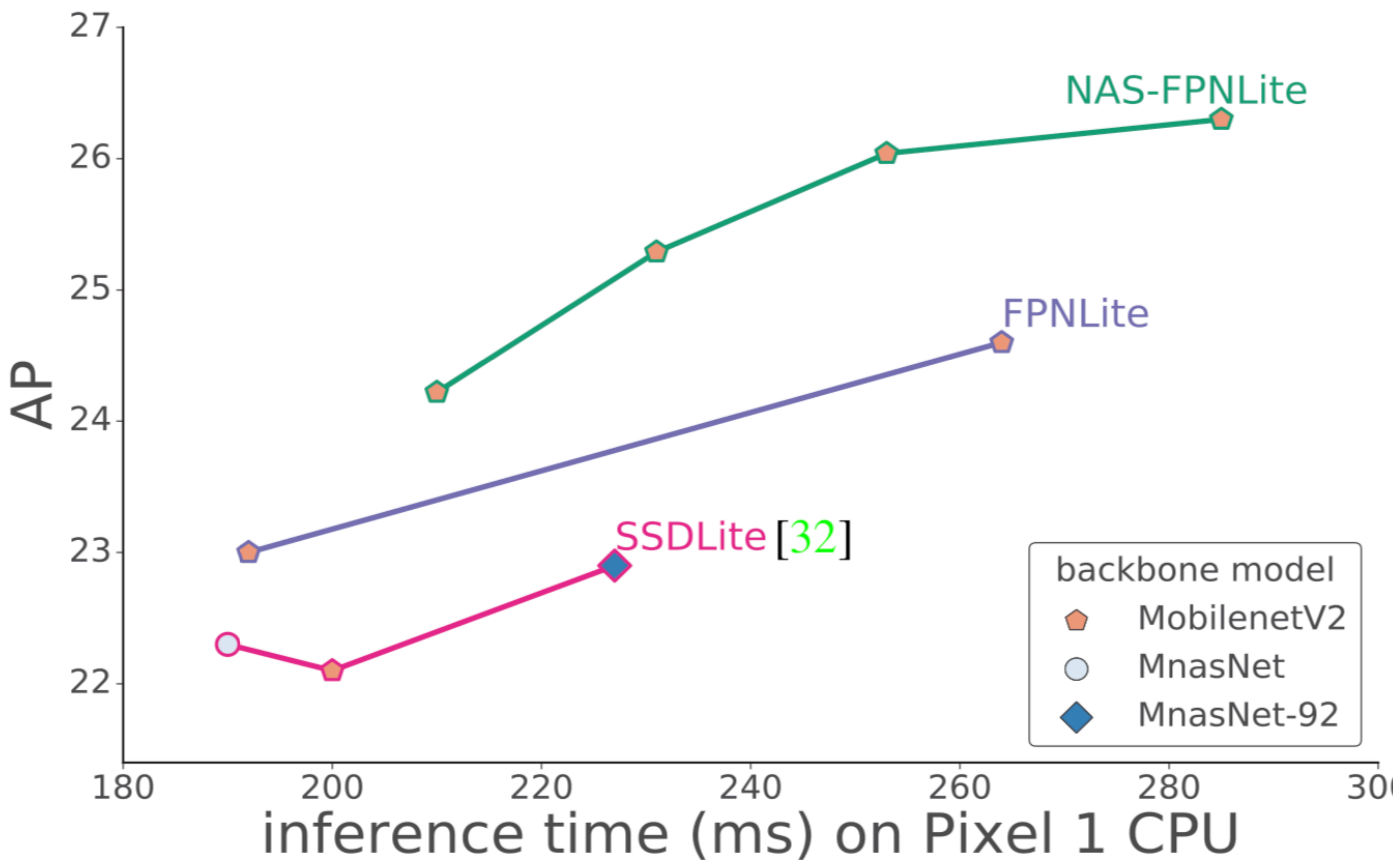


(a) Network level architecture used in DeepLabv3 [9].

Method	**ImageNet**	**Coarse**	mIOU (%)
FRRN-A [60]			63.0
GridNet [17]			69.5
FRRN-B [60]			71.8
Auto-DeepLab-S			79.9
Auto-DeepLab-L			80.4
Auto-DeepLab-S		✓	80.9
Auto-DeepLab-L		✓	82.1

NAS-FPN


NAS-FPN
Mask RCNN
FPN [23]
Mask RCNN [10]
backbone model
R-50
Amoeba
R-101
X-101-32x8d
X-101-64x4d
X-152-32x8d
AP
48
46
44
42
40
38
0
50
100
150
200
250
300
350
inference time (ms) on P100 GPU


NAS-FPNLite
FPNLite
SSDLite [32]
backbone model
MobilenetV2
MnasNet
MnasNet-92
AP
27
26
25
24
23
22
180
200
220
240
260
280
300
inference time (ms) on Pixel 1 CPU