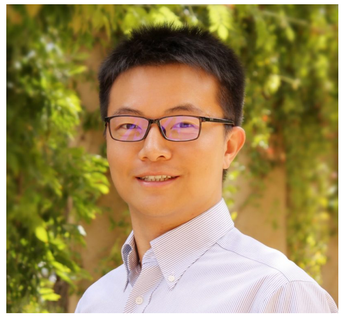
MIT HAN LAB

EfficientML.ai Lecture 07 Neural Architecture Search

Part I



Song Han

Associate Professor, MIT
Distinguished Scientist, NVIDIA

@SongHan_MIT

Lecture Plan

Today we will:

1. Review primitive operations of deep neural networks.

2. Introduce popular building blocks.

3. Introduce **Neural Architecture Search (NAS)**, an automatic technique for designing neural network architectures.

Trade-off Between Efficiency and Accuracy

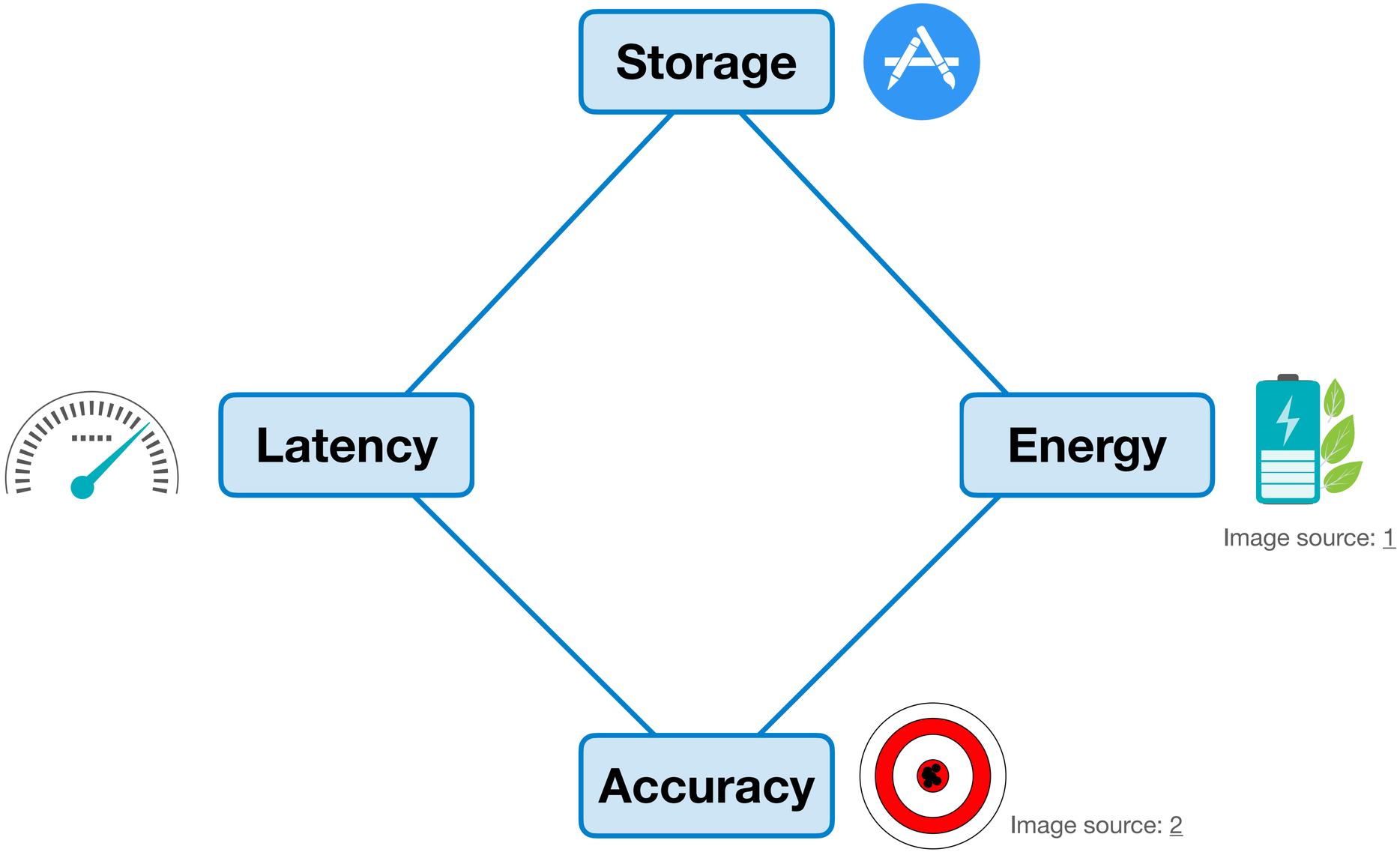
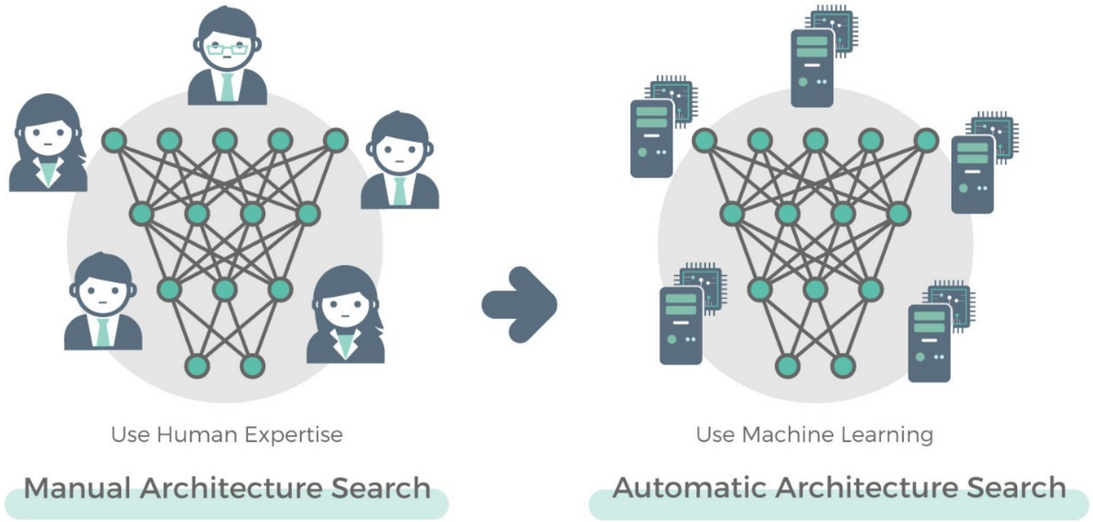


Image source: 1

Image source: 2

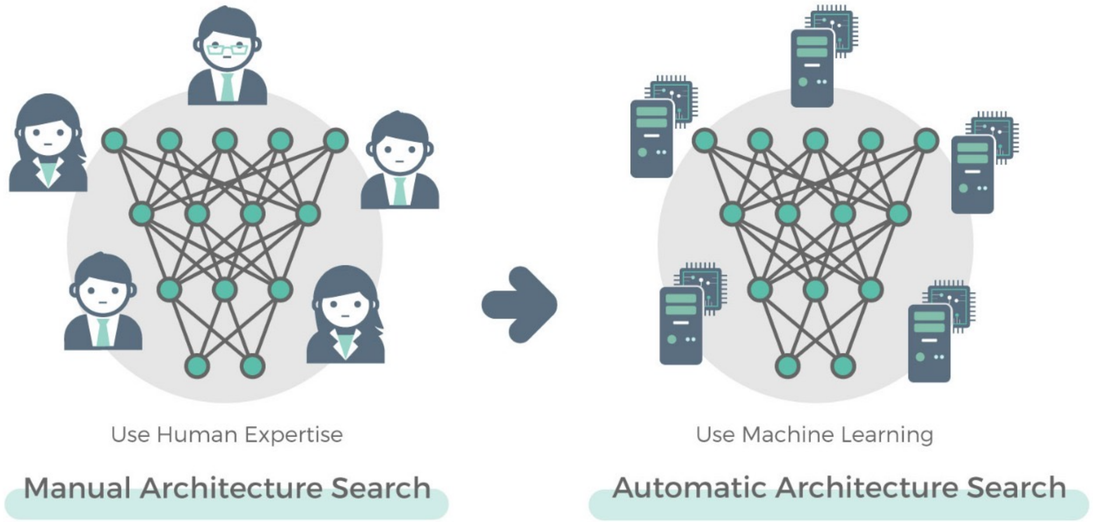
Neural Architecture Search

- **Primitive operations**
- **Classic building blocks**
- **Introduction to neural architecture search (NAS)**
  - What is NAS?
  - Search space
  - Design the search space
  - Search strategy
- Efficient and Hardware-aware NAS
  - Performance estimation strategy
  - Hardware-aware NAS
  - Zero-shot NAS
  - Neural-hardware architecture co-search
- NAS applications
  - NLP, GAN, point cloud, pose



Neural Architecture Search

- **Primitive operations**
- Classic building blocks
- Introduction to neural architecture search (NAS)
  - What is NAS?
  - Search space
  - Design the search space
  - Search strategy
- Efficient and Hardware-aware NAS
  - Performance estimation strategy
  - Hardware-aware NAS
  - Zero-shot NAS
  - Neural-hardware architecture co-search
- NAS applications
  - NLP, GAN, point cloud, pose

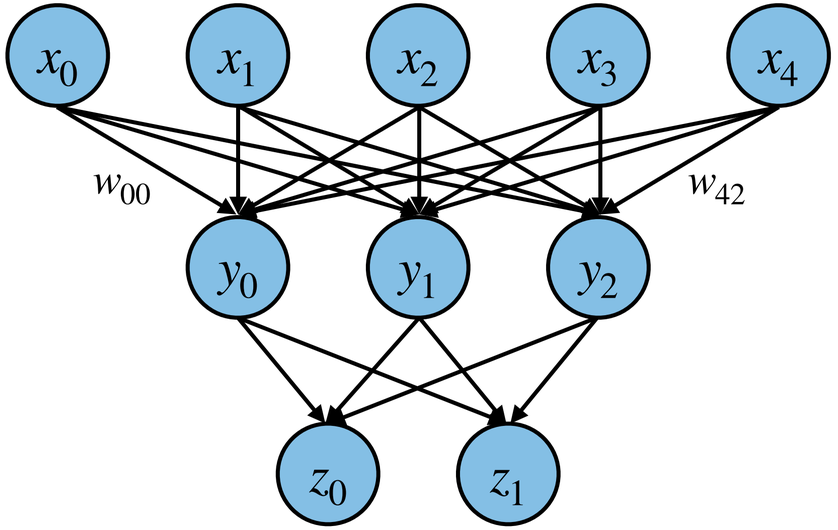


Recap: Primitive Operations

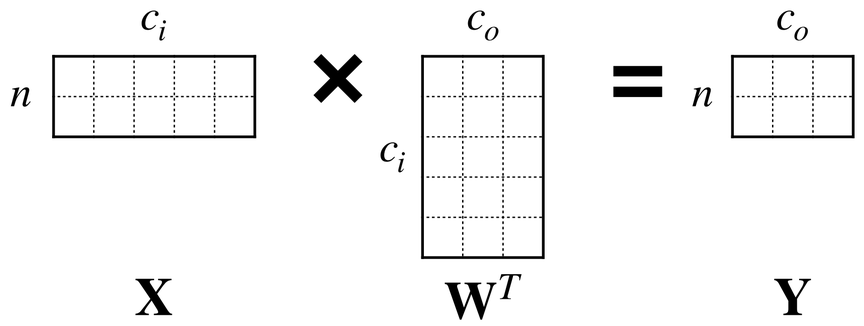
Fully-connected layer / linear layer

- **Shape of Tensors:**
  - Input Features $\mathbf{X}$: (n, c_i)
  - Output Features $\mathbf{Y}$: (n, c_o)
  - Weights $\mathbf{W}$: (c_o, c_i)
  - Bias $\mathbf{b}$: $(c_o,)$

Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels



Multilayer Perceptron (MLP)

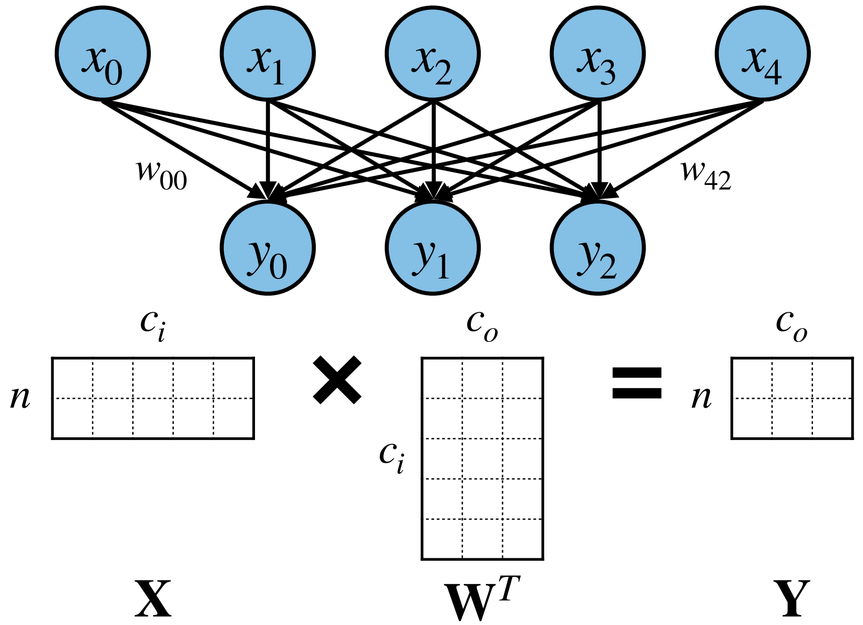


Recap: Primitive Operations

Fully-connected layer / linear layer

Layer	MACs (batch size n=1)
Linear Layer	$c_o \cdot c_i$

* bias is ignored



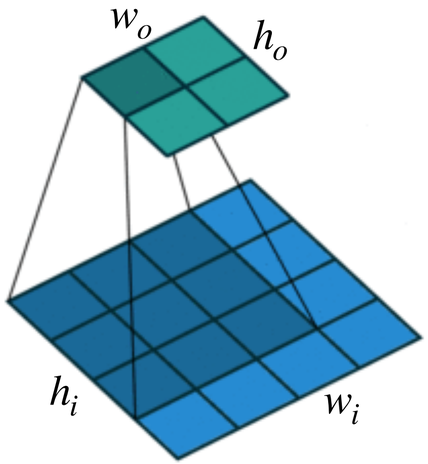
Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h, k_w	Kernel Height/Width
g	Groups

Recap: Primitive Operations

Convolution layer

- **Shape of Tensors:**

	1D Conv	2D Conv
• Input Features $\mathbf{X}$: ~~(n, c_i)~~	(n, c_i, w_i)	(n, c_i, h_i, w_i)
• Output Features $\mathbf{Y}$: ~~(n, c_o)~~	(n, c_o, w_o)	(n, c_o, h_o, w_o)
• Weights $\mathbf{W}$: ~~(c_o, c_i)~~	(c_o, c_i, k_w)	(c_o, c_i, k_h, k_w)
• Bias $\mathbf{b}$: $(c_o,)$		



Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h	Kernel Height
k_w	Kernel Width

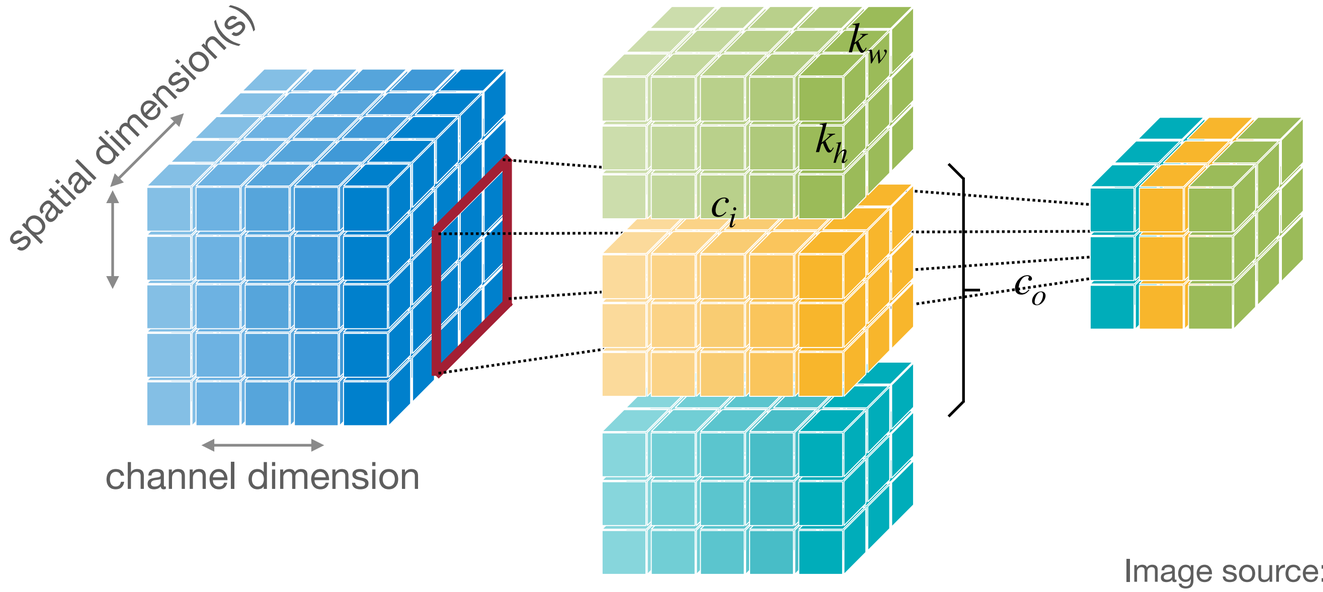


Image source: 1

Recap: Primitive Operations

Convolution layer

Layer	MACs (batch size n=1)
Linear Layer	$c_o \cdot c_i$
Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w \cdot h_o \cdot w_o$

* bias is ignored

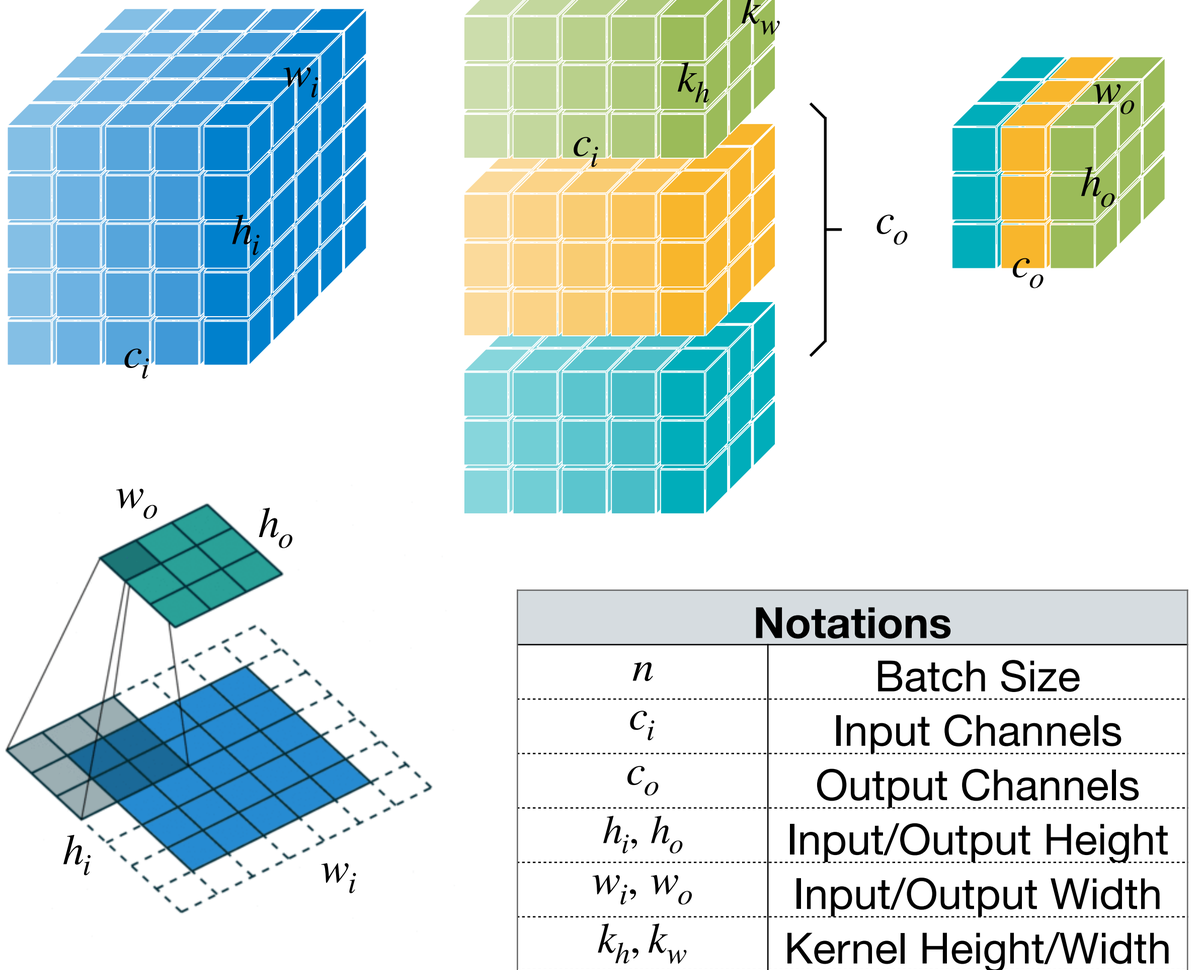


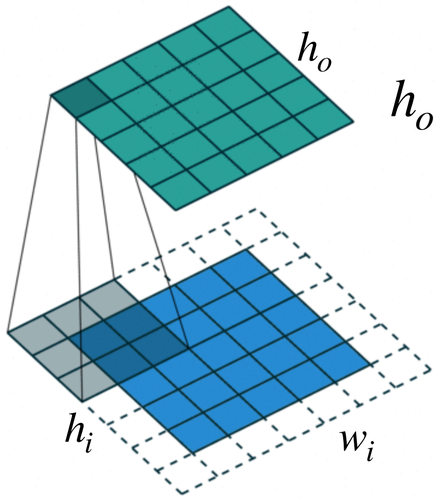
Image source: 1

Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h, k_w	Kernel Height/Width
g	Groups

Recap: Primitive Operations

Grouped convolution layer

- **Shape of Tensors:**
  - Input Features $\mathbf{X}$: (n, c_i, h_i, w_i)
  - Output Features $\mathbf{Y}$: (n, c_o, h_o, w_o)
  - Weights $\mathbf{W}$: ~~(c_o, c_i, k_h, k_w)~~ $(g \cdot c_o/g, c_i/g, k_h, k_w)$
  - Bias $\mathbf{b}$: $(c_o,)$



$$h_o = \frac{h_i + 2p - k_h}{s} + 1$$

p is padding

s is stride

Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h	Kernel Height
k_w	Kernel Width
g	Groups

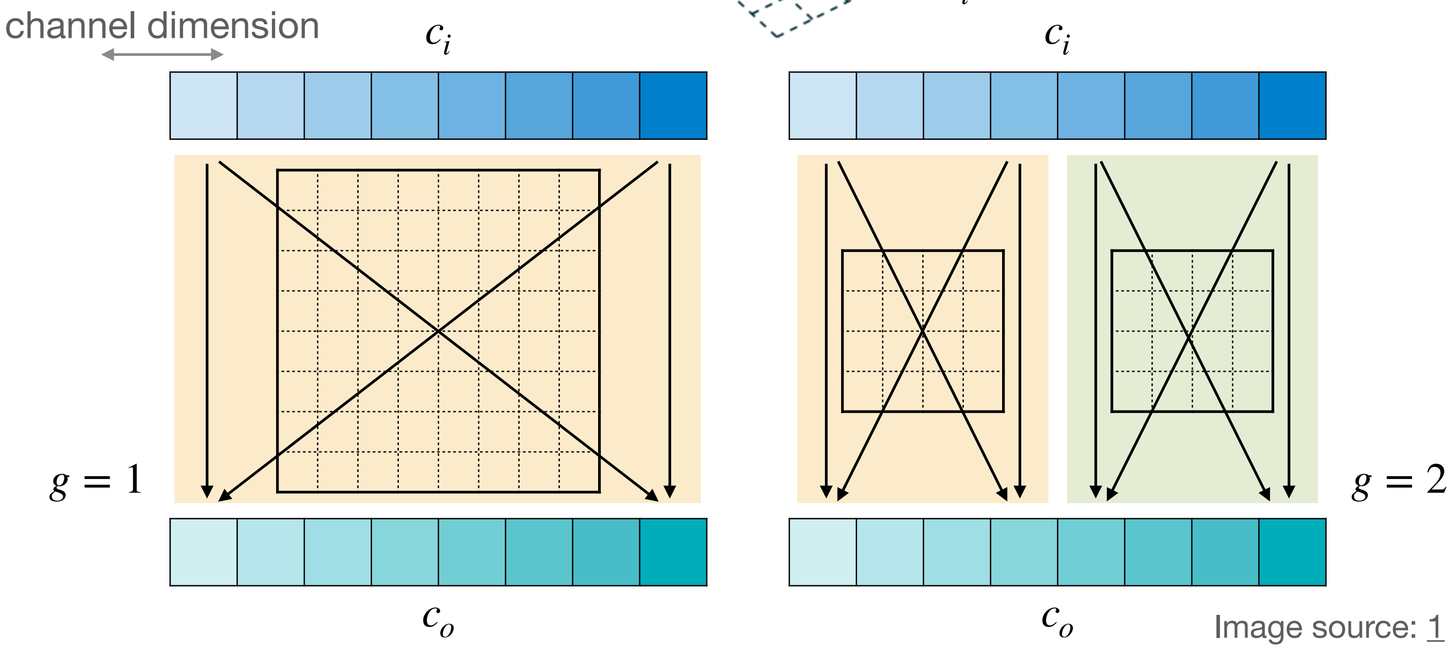


Image source: 1

Recap: Primitive Operations

Grouped convolution layer

Layer	MACs (batch size n=1)
Linear Layer	$c_o \cdot c_i$
Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w \cdot h_o \cdot w_o$
Grouped Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w \cdot h_o \cdot w_o / g$

* bias is ignored

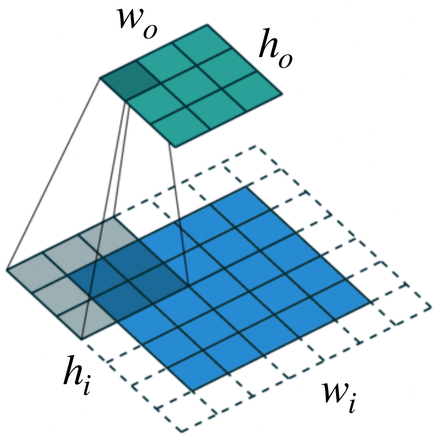
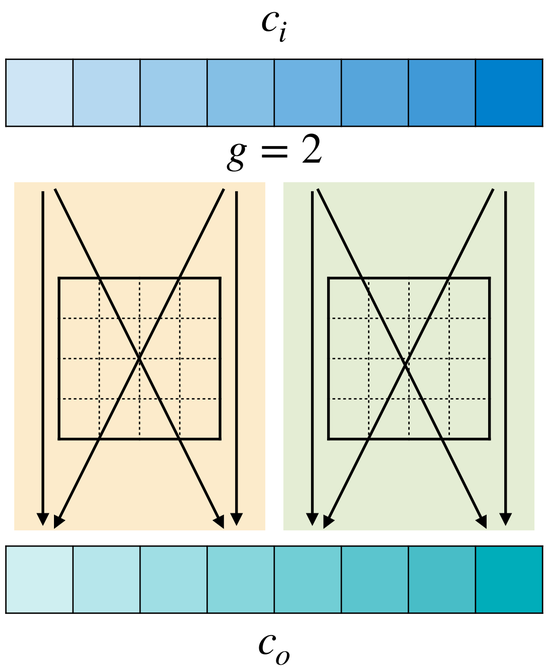


Image source: 1

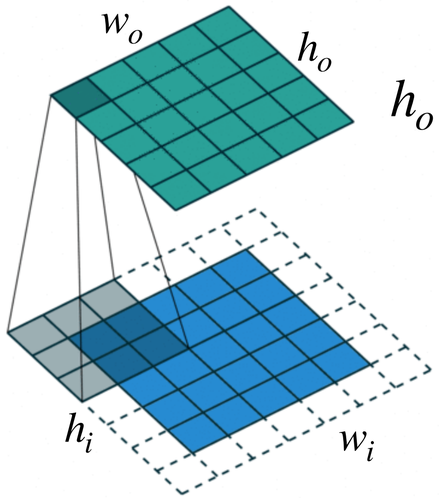


Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h, k_w	Kernel Height/Width
g	Groups

Recap: Primitive Operations

Depthwise convolution layer

- **Shape of Tensors:**
  - Input Features $\mathbf{X}$: (n, c_i, h_i, w_i)
  - Output Features $\mathbf{Y}$: (n, c_o, h_o, w_o)
  - Weights $\mathbf{W}$: ~~(c_o, c_i, k_h, k_w)~~ (c, k_h, k_w)
  - Bias $\mathbf{b}$: $(c_o,)$

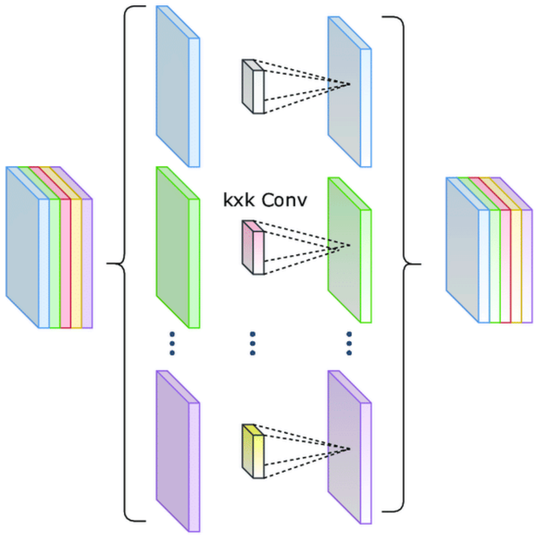


$$h_o = \frac{h_i + 2p - k_h}{s} + 1$$

p is padding

s is stride

Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h	Kernel Height
k_w	Kernel Width
g	Groups



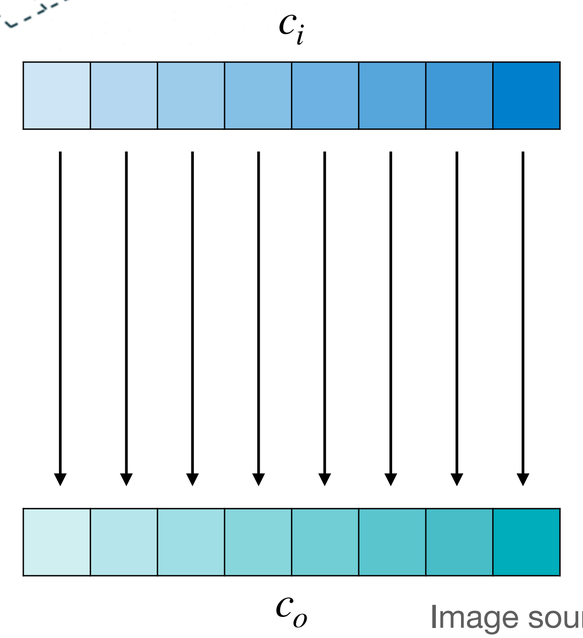


Image source: 1, 2

Recap: Primitive Operations

Depthwise convolution layer

Layer	MACs (batch size n=1)
Linear Layer	$c_o \cdot c_i$
Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w \cdot h_o \cdot w_o$
Grouped Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w \cdot h_o \cdot w_o / g$
Depthwise Convolution	$c_o \cdot k_h \cdot k_w \cdot h_o \cdot w_o$

* bias is ignored

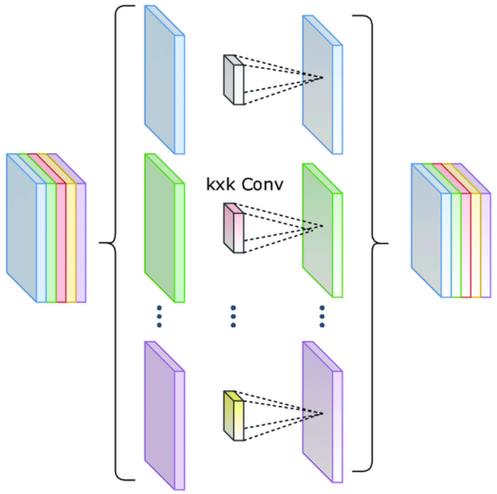
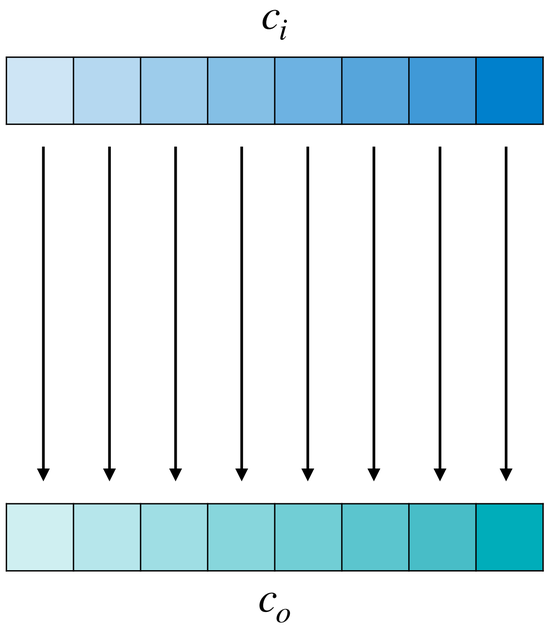


Image source: 1



Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h, k_w	Kernel Height/Width
g	Groups

Recap: Primitive Operations

1x1 convolution layer

Layer	MACs (batch size n=1)
Linear Layer	$c_o \cdot c_i$
Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w \cdot h_o \cdot w_o$
Grouped Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w \cdot h_o \cdot w_o / g$
Depthwise Convolution	$c_o \cdot k_h \cdot k_w \cdot h_o \cdot w_o$
1x1 Convolution	$c_o \cdot c_i \cdot h_o \cdot w_o$

* bias is ignored

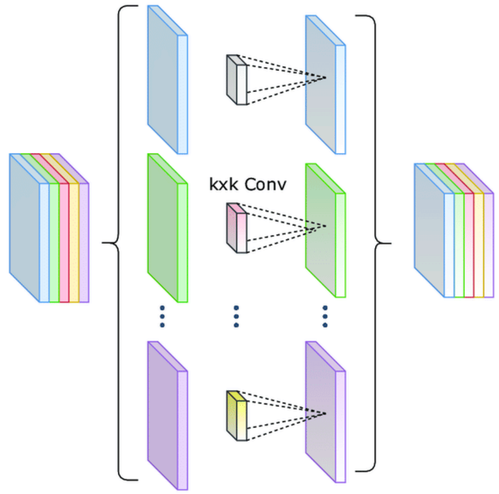
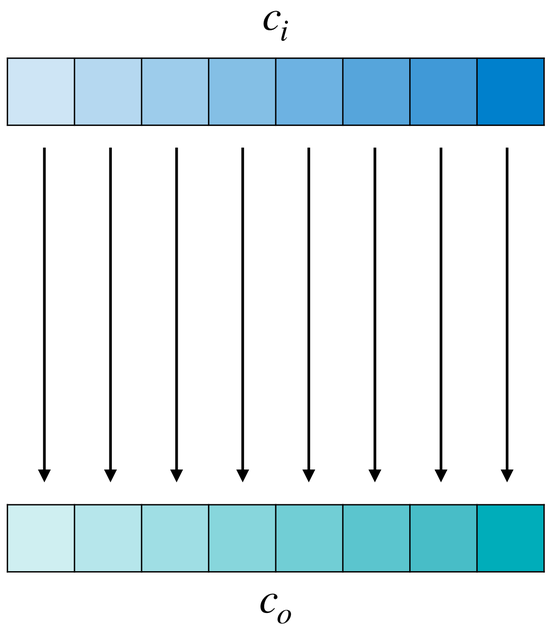


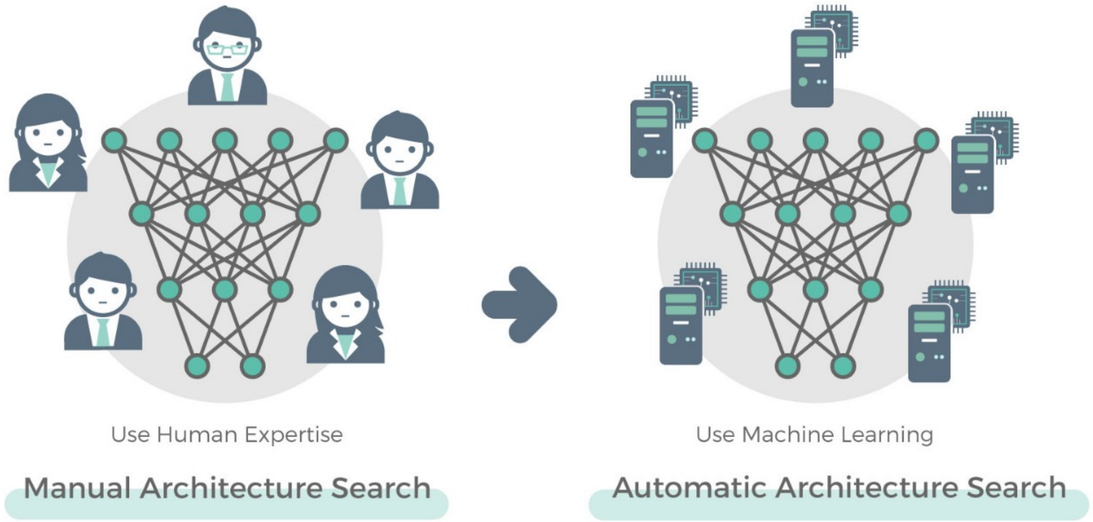
Image source: 1



Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h, k_w	Kernel Height/Width
g	Groups

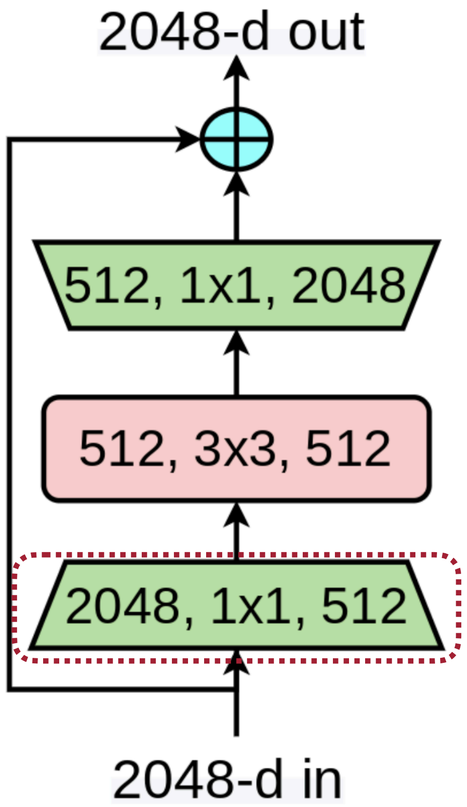
Neural Architecture Search

- **Primitive operations**
- **Classic building blocks**
- Introduction to neural architecture search (NAS)
  - What is NAS?
  - Search space
  - Design the search space
  - Search strategy
- Efficient and Hardware-aware NAS
  - Performance estimation strategy
  - Hardware-aware NAS
  - Zero-shot NAS
  - Neural-hardware architecture co-search
- NAS applications
  - NLP, GAN, point cloud, pose



Classic Building Blocks

ResNet50: bottleneck block



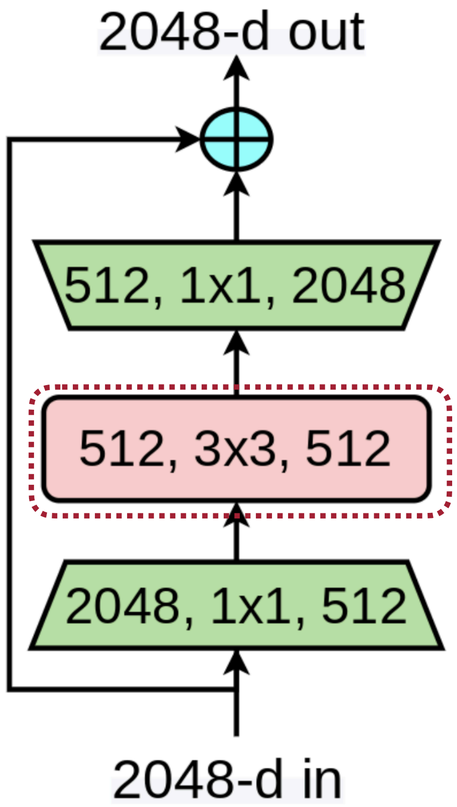
Reduce the number of channels by 4x
via 1x1 convolution

image source: link

Deep Residual Learning for Image Recognition [He *et al.*, CVPR 2016]

Classic Building Blocks

ResNet50: bottleneck block



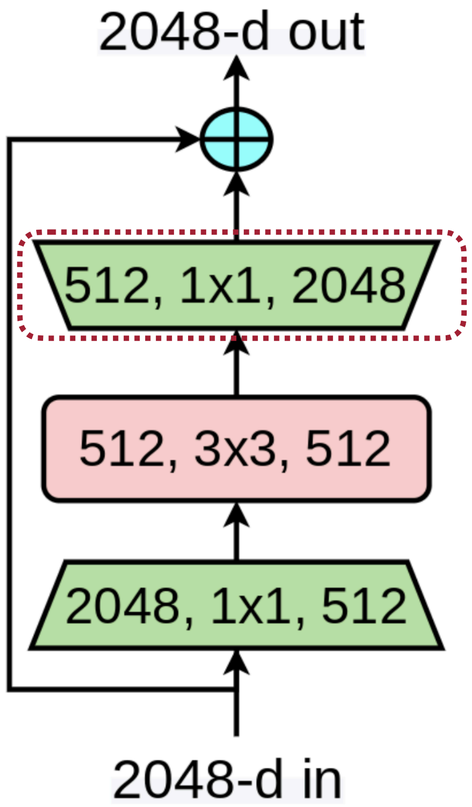
Feed reduced feature map to 3x3 convolution

image source: link

Deep Residual Learning for Image Recognition [He *et al.*, CVPR 2016]

Classic Building Blocks

ResNet50: bottleneck block



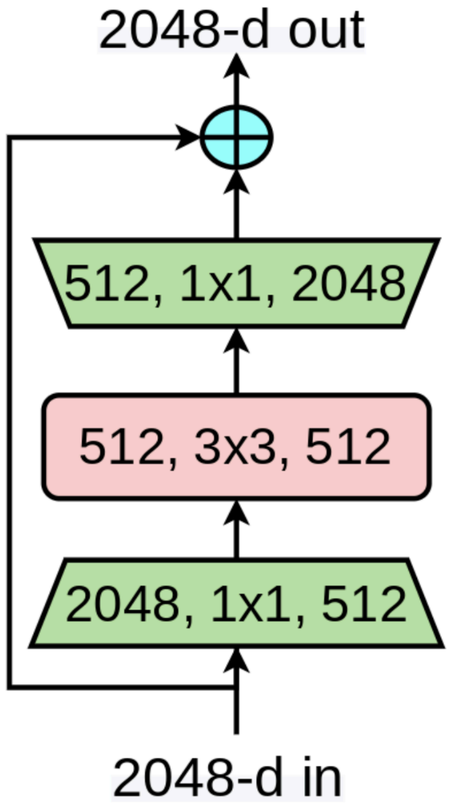
Expand the number of channels via 1x1 convolution

image source: link

Deep Residual Learning for Image Recognition [He *et al.*, CVPR 2016]

Classic Building Blocks

ResNet50: bottleneck block



#MACs

2048 x 512 x H x W x 1

512 x 512 x H x W x 9

2048 x 512 x H x W x 1

image source: link

Deep Residual Learning for Image Recognition [He *et al.*, CVPR 2016]

Classic Building Blocks

ResNet50: bottleneck block

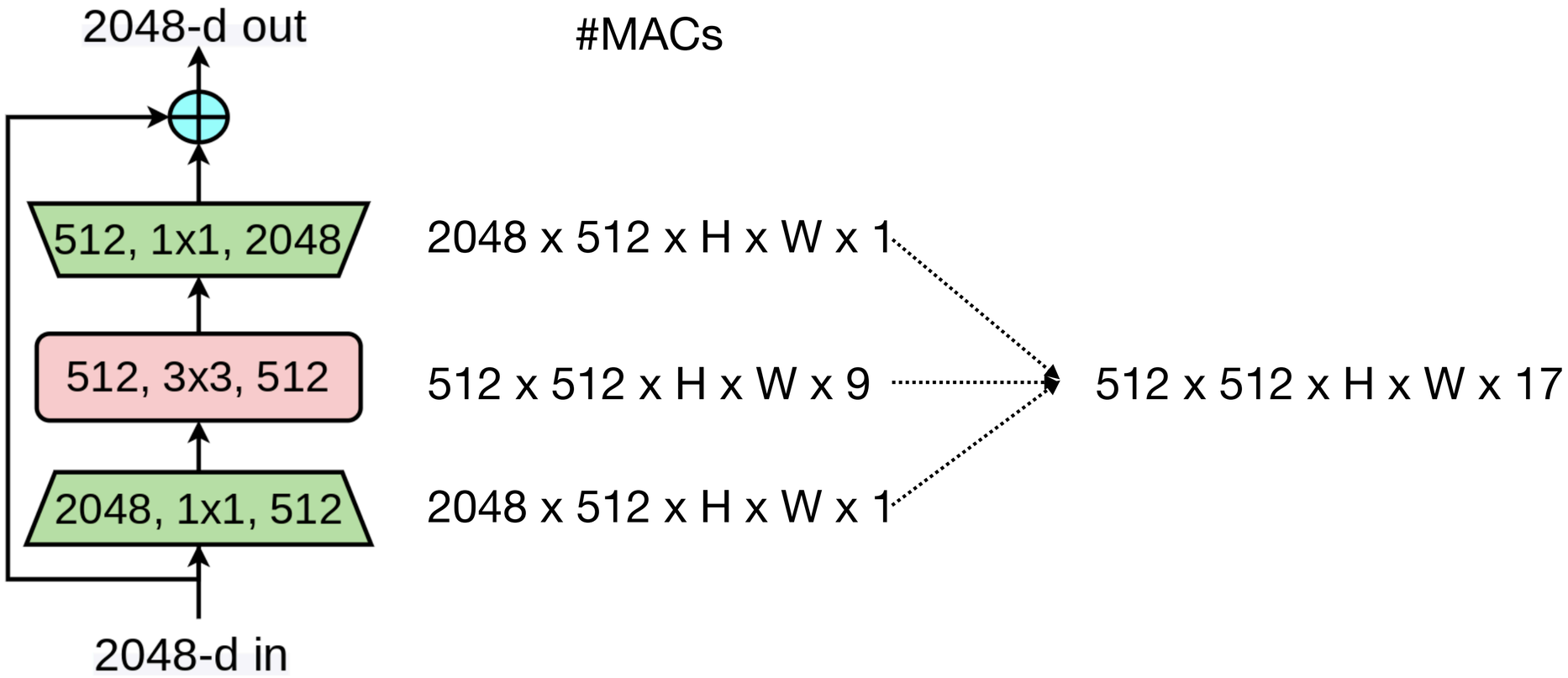
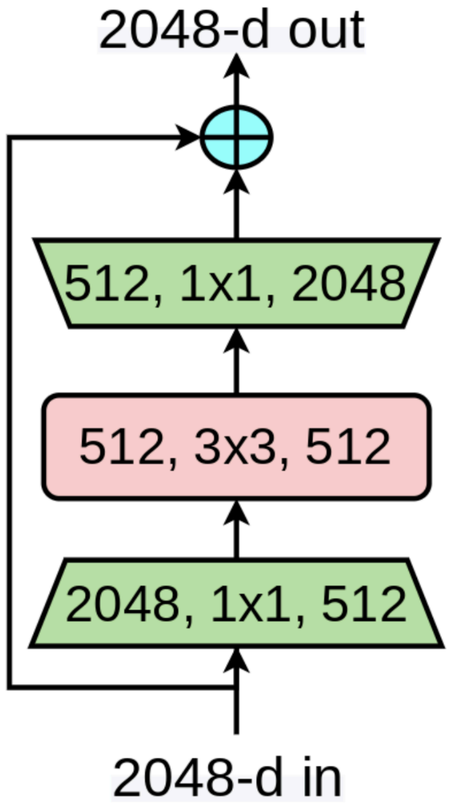


image source: link

Deep Residual Learning for Image Recognition [He *et al.*, CVPR 2016]

Classic Building Blocks

ResNet50: bottleneck block



#MACs

2048 x 512 x H x W x 1

512 x 512 x H x W x 9

2048 x 512 x H x W x 1

2048 x 2048 x H x W x 9

= 512 x 512 x H x W x 144

8.5x reduction

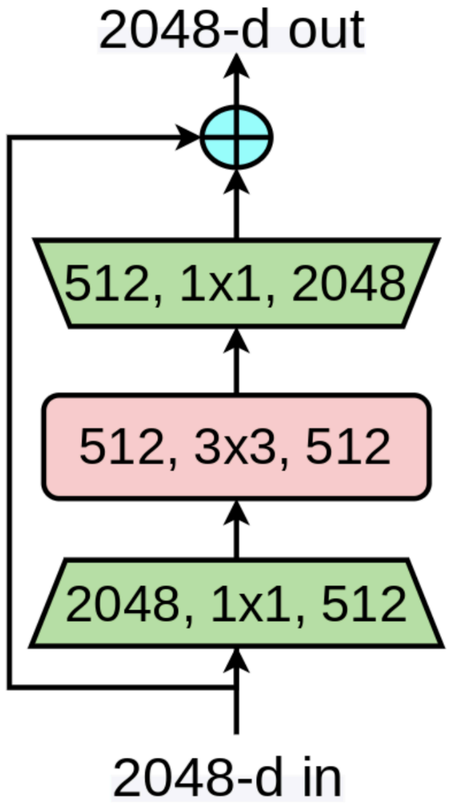
512 x 512 x H x W x 17

image source: link

Deep Residual Learning for Image Recognition [He *et al.*, CVPR 2016]

Classic Building Blocks

ResNet50: bottleneck block



#Params

2048 x 512 x 1

512 x 512 x 9

2048 x 512 x 1

2048 x 2048 x 9

= 512 x 512 x 144

8.5x reduction

512 x 512 x 17

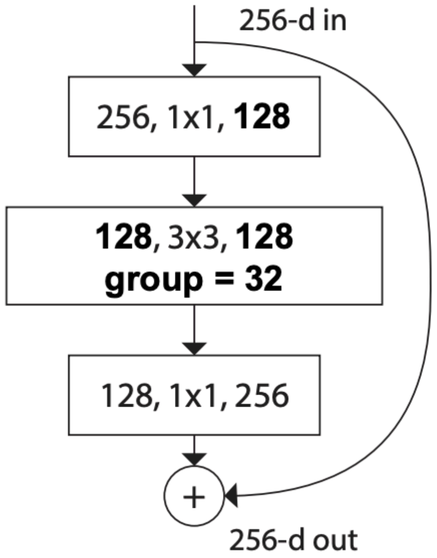
image source: link

Deep Residual Learning for Image Recognition [He *et al.*, CVPR 2016]

Classic Building Blocks

ResNeXt: grouped convolution

- Replace 3x3 convolution with 3x3 grouped convolution.

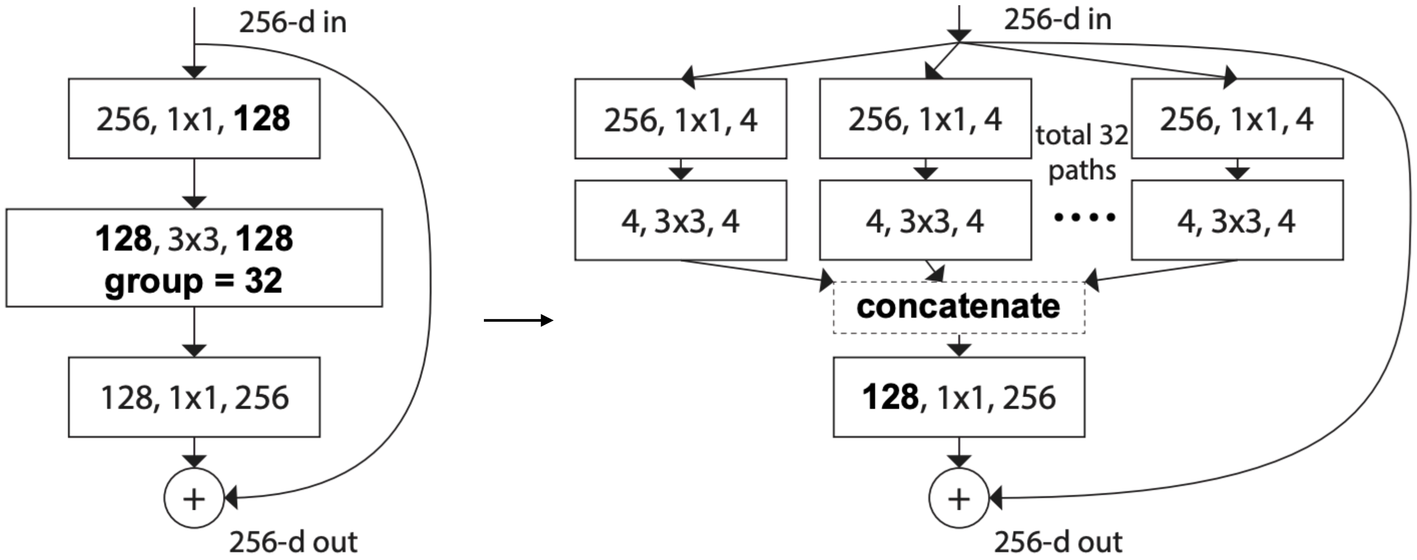


Aggregated Residual Transformations for Deep Neural Networks [Xie *et al.*, CVPR 2017]

Classic Building Blocks

ResNeXt: grouped convolution

- Replace 3x3 convolution with 3x3 grouped convolution.
- Equivalent to a multi-path block.

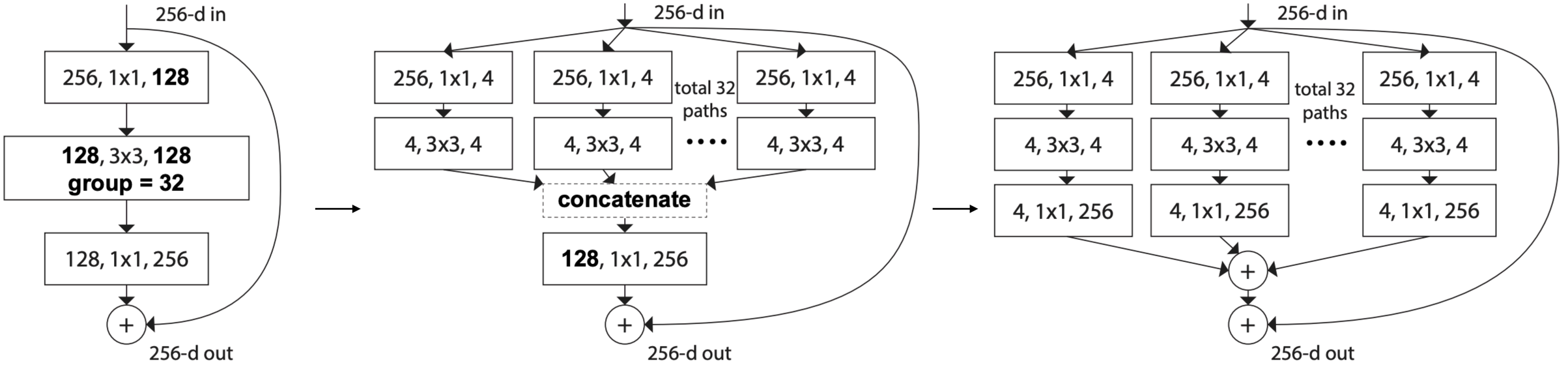


Aggregated Residual Transformations for Deep Neural Networks [Xie *et al.*, CVPR 2017]

Classic Building Blocks

ResNeXt: grouped convolution

- Replace 3x3 convolution with 3x3 grouped convolution.
- Equivalent to a multi-path block.



Aggregated Residual Transformations for Deep Neural Networks [Xie *et al.*, CVPR 2017]

Classic Building Blocks

MobileNet: depthwise-separable block

- Depthwise convolution is an extreme case of group convolution where the group number equals the number of input channels.

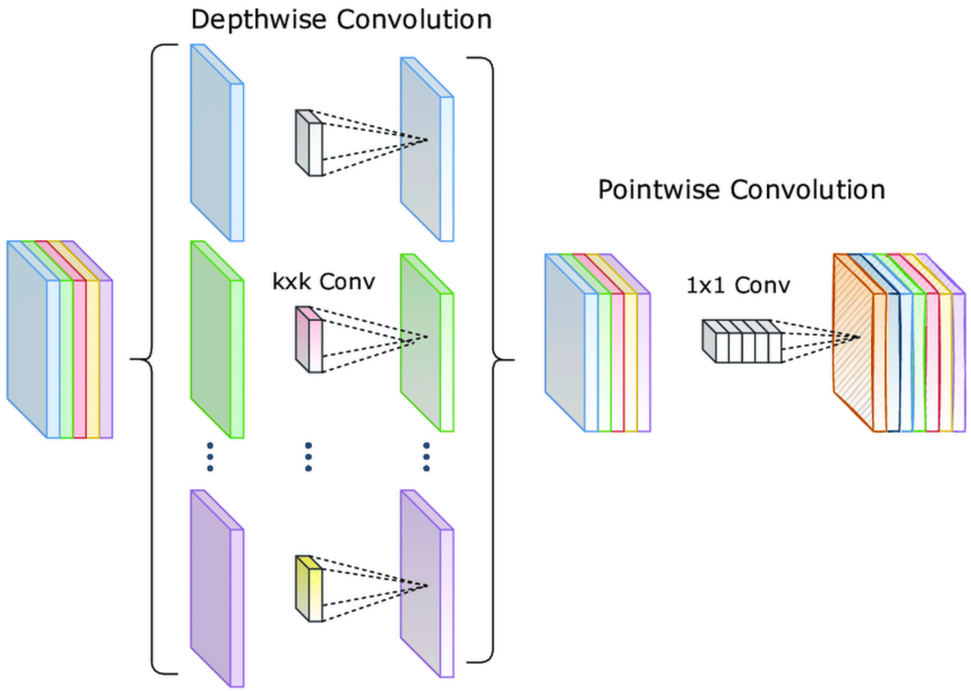


image source: link

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Howard *et al.*, arXiv 2017]

Classic Building Blocks

MobileNet: depthwise-separable block

- Depthwise convolution is an extreme case of group convolution where the group number equals the number of input channels.
- Use depthwise convolution to capture spatial information.

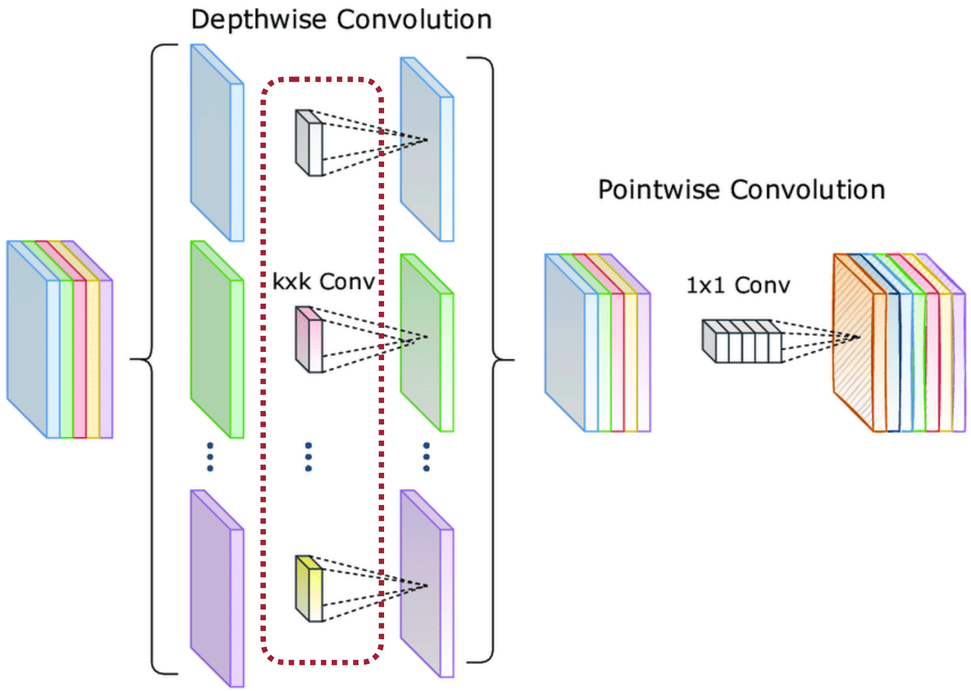


image source: link

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Howard *et al.*, arXiv 2017]

Classic Building Blocks

MobileNet: depthwise-separable block

- Depthwise convolution is an extreme case of group convolution where the group number equals the number of input channels.
- Use depthwise convolution to capture spatial information.
- Use 1x1 convolution to fuse/exchange information across different channels.

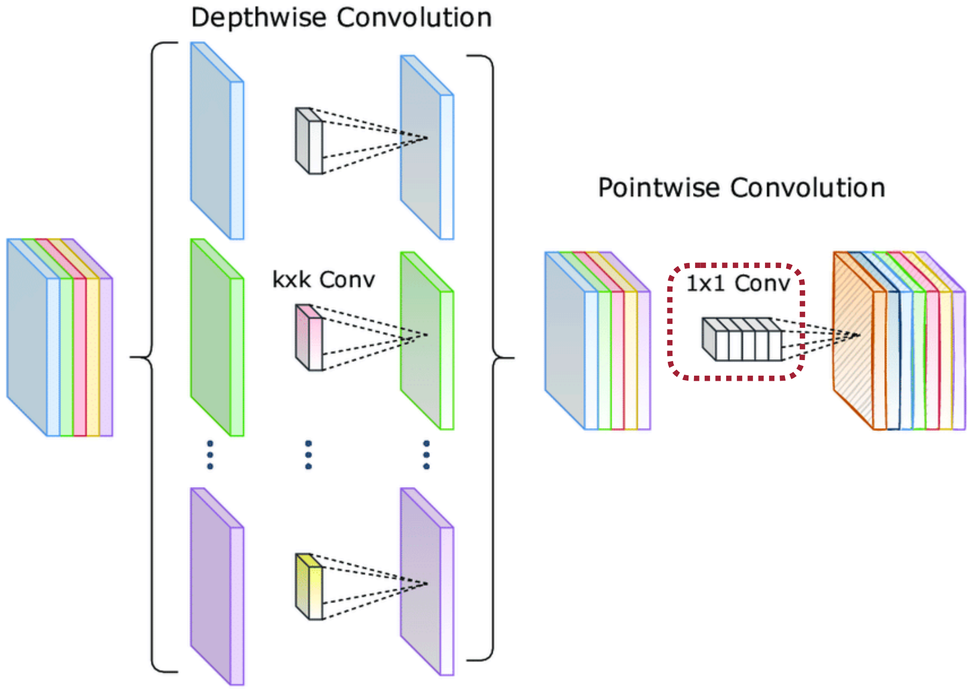


image source: link

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Howard *et al.*, arXiv 2017]

Classic Building Blocks

MobileNetV2: inverted bottleneck block

- Depthwise convolution has a much lower capacity compared to normal convolution.
  - Increase the depthwise convolution's input and output channels to improve its capacity.
  - Depthwise convolution’s cost only grows linearly. Therefore, the cost is still affordable.

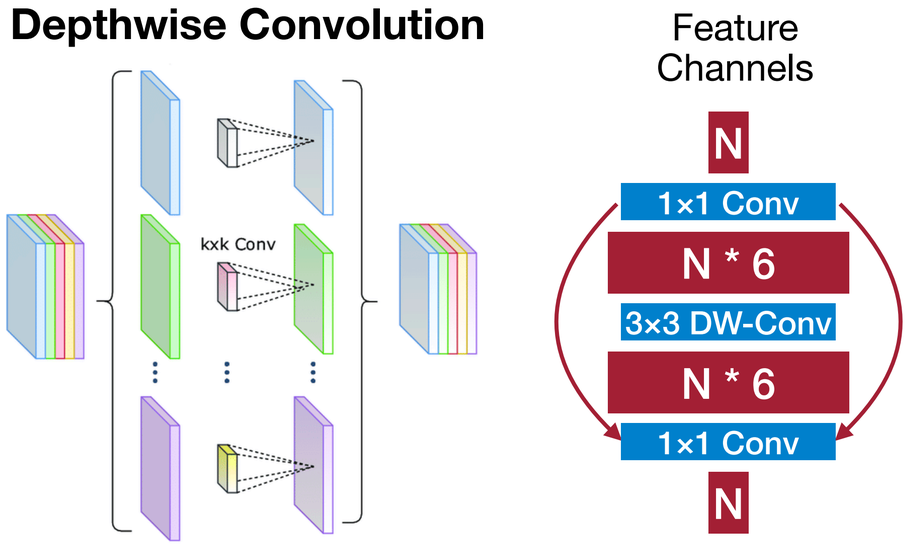


Image source: 1

Image source: 1

MobileNetV2: Inverted Residuals and Linear Bottlenecks [Sandler *et al.*, CVPR 2018]

Classic Building Blocks

MobileNetV2: inverted bottleneck block

- Depthwise convolution has a much lower capacity compared to normal convolution.
  - Increase the depthwise convolution's input and output channels to improve its capacity.
  - Depthwise convolution's cost only grows linearly. Therefore, the cost is still affordable.

Depthwise Convolution

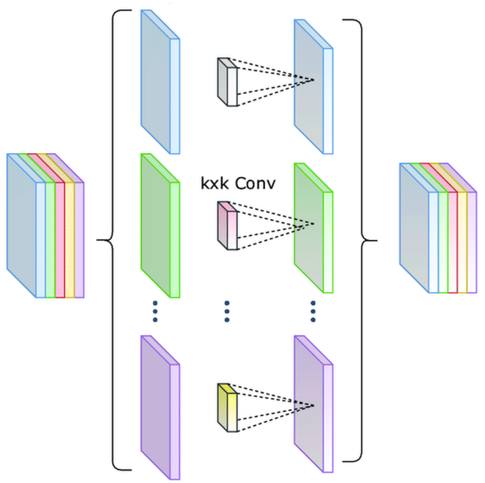
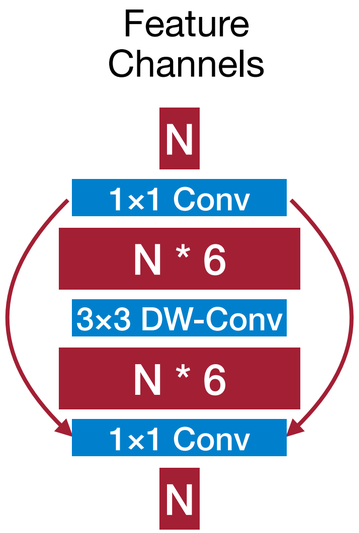


Image source: 1



#MACs (N = 160)

160 x 960 x H x W x 1

960 x H x W x 9

160 x 960 x H x W x 1

160 x 160 x H x W x 9

= 960 x H x W x 240

1 : 1.37

960 x H x W x 329

Image source: 1

MobileNetV2: Inverted Residuals and Linear Bottlenecks [Sandler *et al.*, CVPR 2018]

Classic Building Blocks

MobileNetV2: inverted bottleneck block

- Depthwise convolution has a much lower capacity compared to normal convolution.
  - Increase the depthwise convolution's input and output channels to improve its capacity.
  - Depthwise convolution's cost only grows linearly. Therefore, the cost is still affordable.

Depthwise Convolution

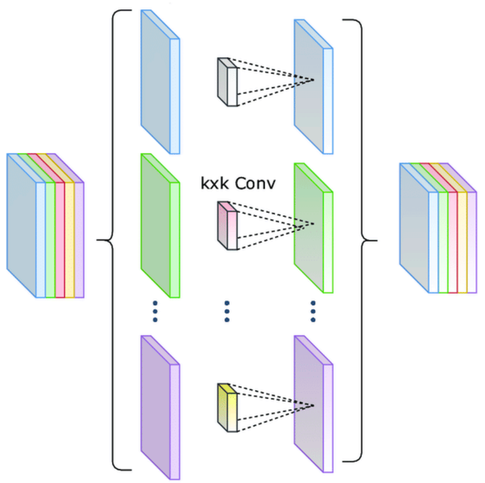
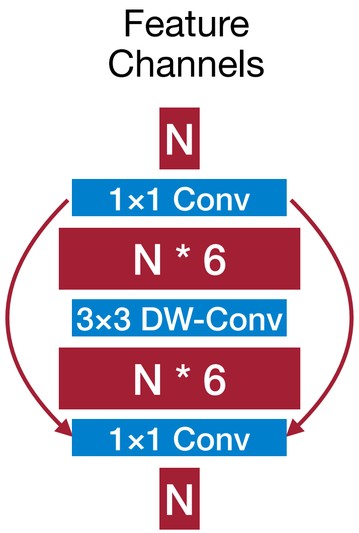


Image source: 1



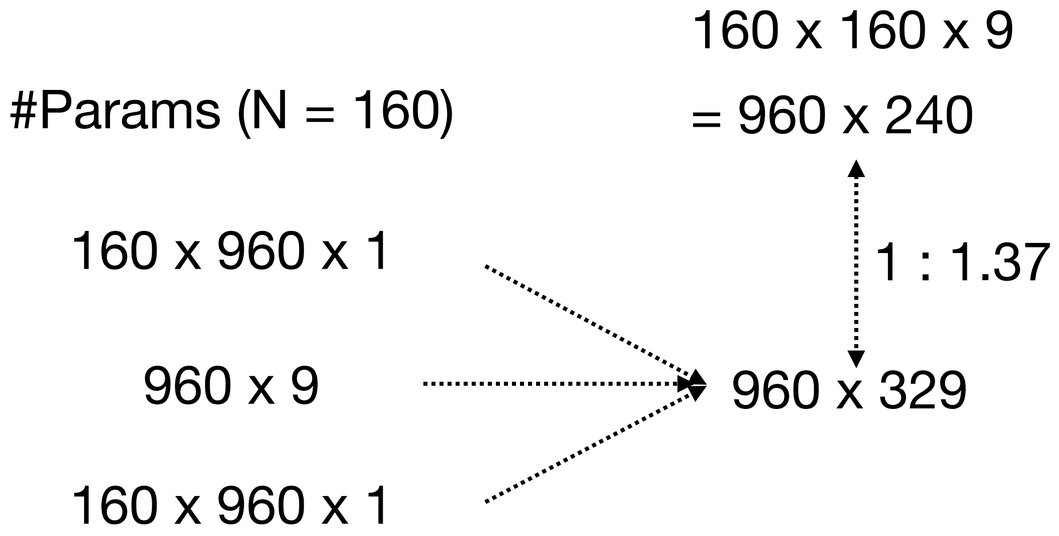


Image source: 1

MobileNetV2: Inverted Residuals and Linear Bottlenecks [Sandler *et al.*, CVPR 2018]

Classic Building Blocks

MobileNetV2: inverted bottleneck block

- Depthwise convolution has a much lower capacity compared to normal convolution.
  - Increase the depthwise convolution's input and output channels to improve its capacity.
  - Depthwise convolution’s cost only grows linearly. Therefore, the cost is still affordable.
  - However, this design is not memory-efficient for both inference and training.

Depthwise Convolution

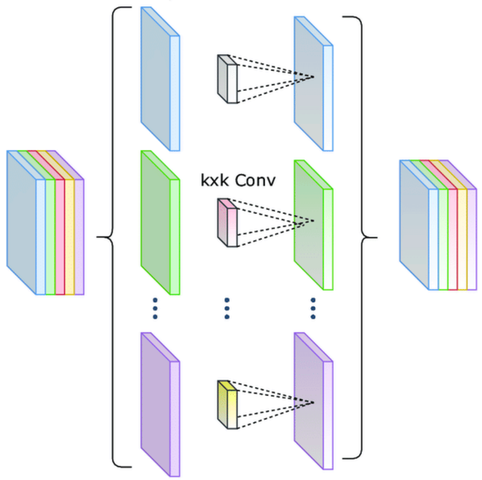
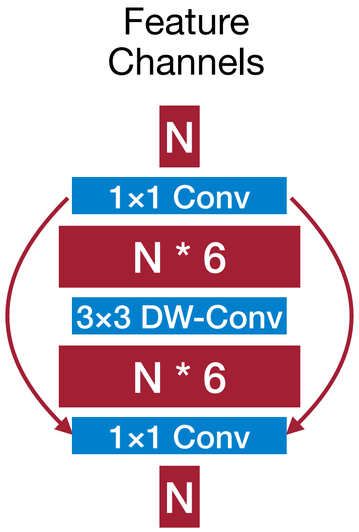
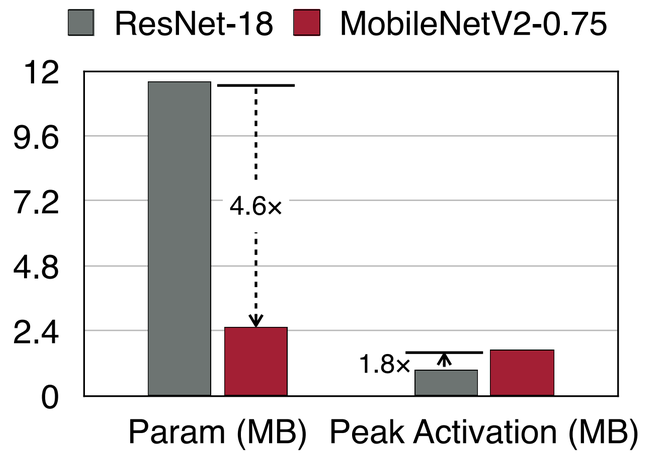


Image source: 1

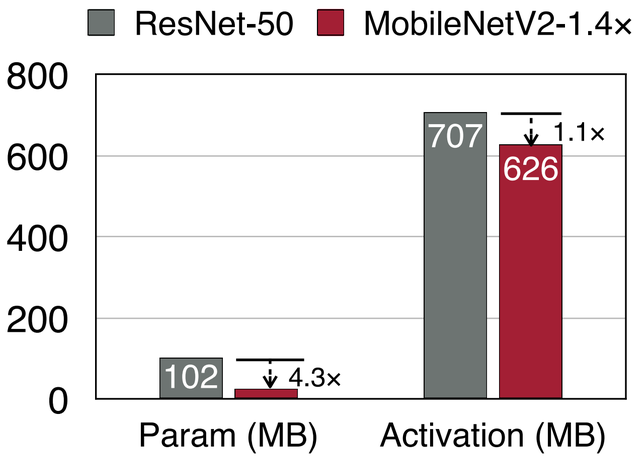


Inference



*** All parameters and activations are Integer numbers (8 bits). Batch size is 1.**

Training



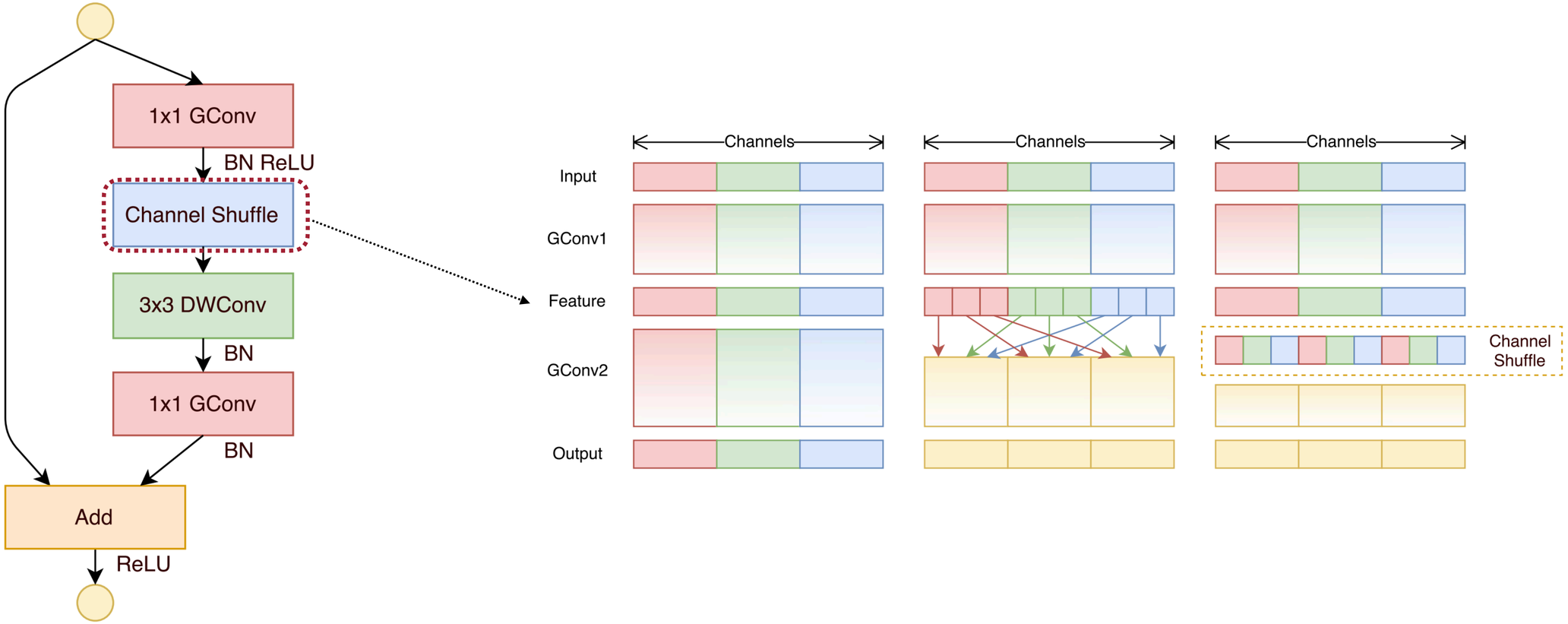
*** All parameters and activations are Floating-Point numbers (32 bits). Batch size is 16.**

MobileNetV2: Inverted Residuals and Linear Bottlenecks [Sandler *et al.*, CVPR 2018]

Classic Building Blocks

ShuffleNet: 1x1 group convolution & channel shuffle

- Further reduce the cost by replacing 1x1 convolution with 1x1 group convolution.
- Exchange information across different groups via channel shuffle.



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

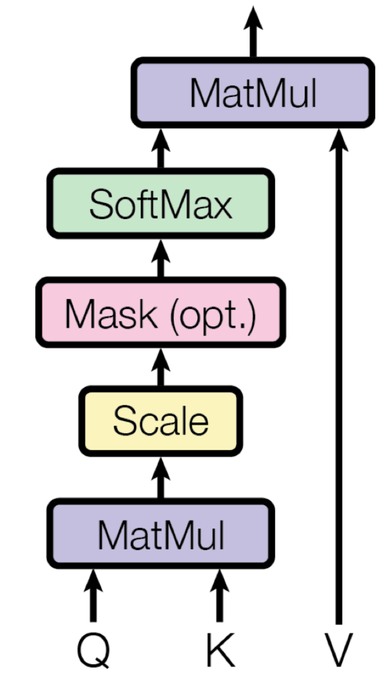
Classic Building Blocks

Transformer: Multi-Head Self-Attention (MHSA)

- **Project** Q, K and V with h **different**, learned linear projections.
- Perform the **scaled dot-product attention** function on each of these projected versions of Q, K and V **in parallel**.
- **Concatenate** the output values.
- **Project** the output values again, resulting in the final values.

$$\text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Scaled Dot-Product Attention



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Attention Is All You Need [Vaswani *et al.*, NeurIPS 2017]

Classic Building Blocks

Transformer: Multi-Head Self-Attention (MHSA)

- **Project** Q, K and V with h **different**, learned linear projections.
- Perform the **scaled dot-product attention** function on each of these projected versions of Q, K and V **in parallel**.
- **Concatenate** the output values.
- **Project** the output values again, resulting in the final values.

$$\text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

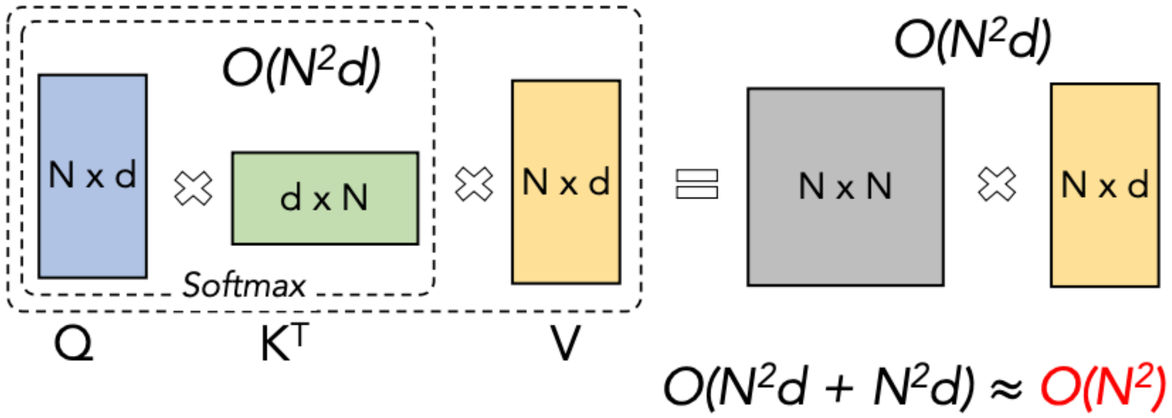


Image source: 1

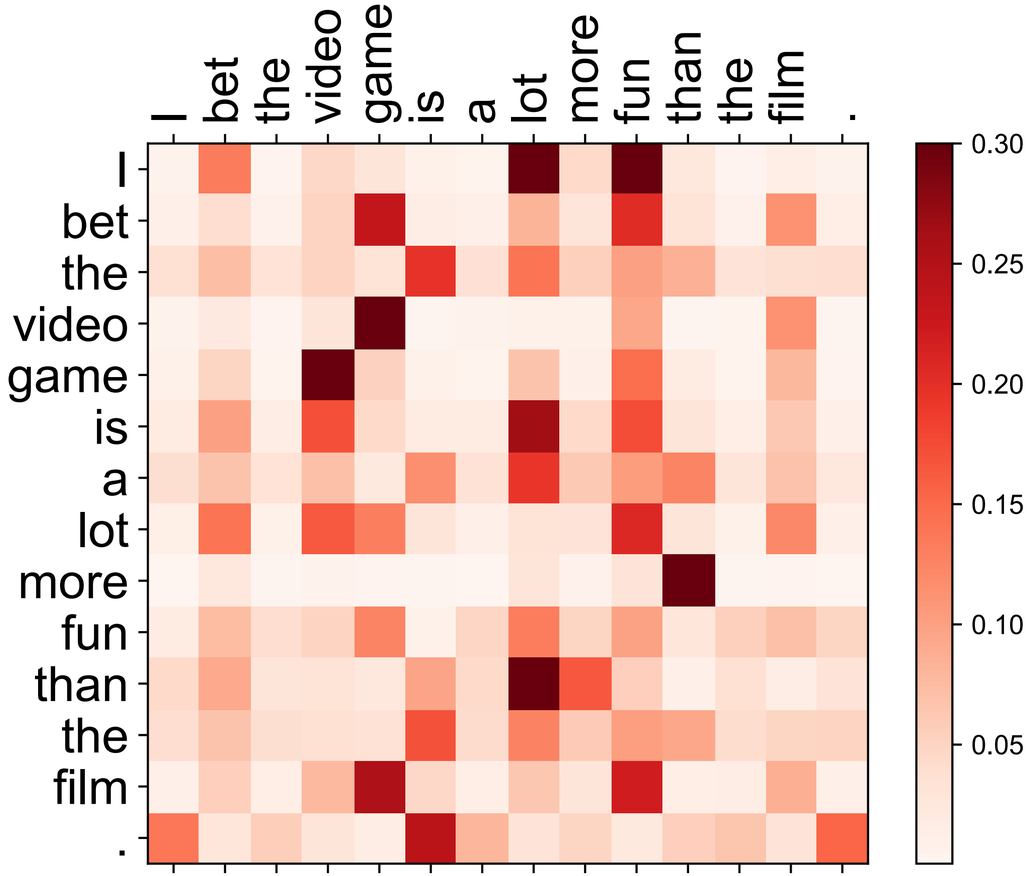
Attention Is All You Need [Vaswani *et al.*, NeurIPS 2017]

Classic Building Blocks

Transformer: Multi-Head Self-Attention (MHSA)

- **Project** Q, K and V with h **different**, learned linear projections.
- Perform the **scaled dot-product attention** function on each of these projected versions of Q, K and V **in parallel**.
- **Concatenate** the output values.
- **Project** the output values again, resulting in the final values.

$$\text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Attention Is All You Need [Vaswani *et al.*, NeurIPS 2017]

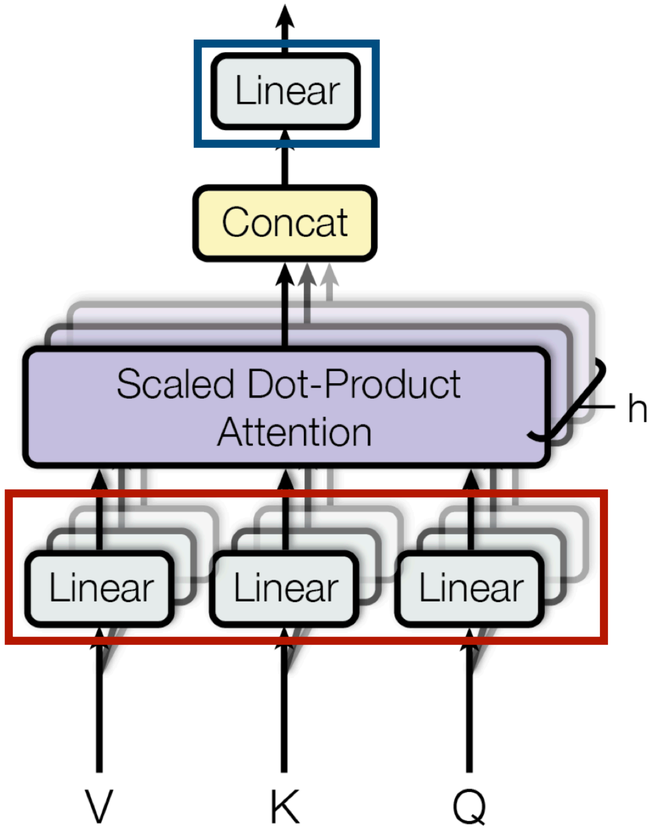
Classic Building Blocks

Transformer: Multi-Head Self-Attention (MHSA)

- **Project** Q, K and V with h **different**, learned linear projections.
- Perform the **scaled dot-product attention** function on each of these projected versions of Q, K and V **in parallel**.
- **Concatenate** the output values.
- **Project** the output values again, resulting in the final values.

$$\text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

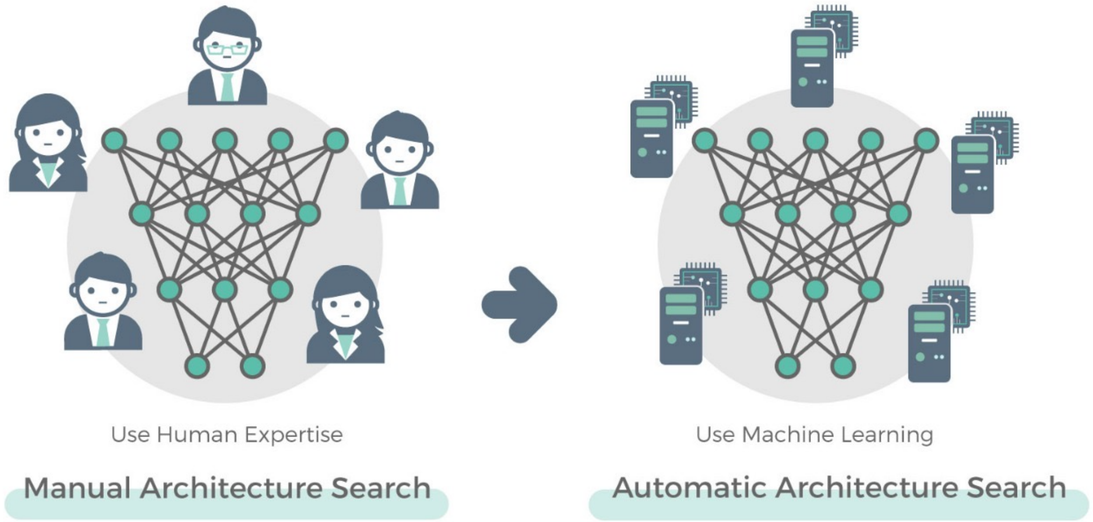
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$



Attention Is All You Need [Vaswani *et al.*, NeurIPS 2017]

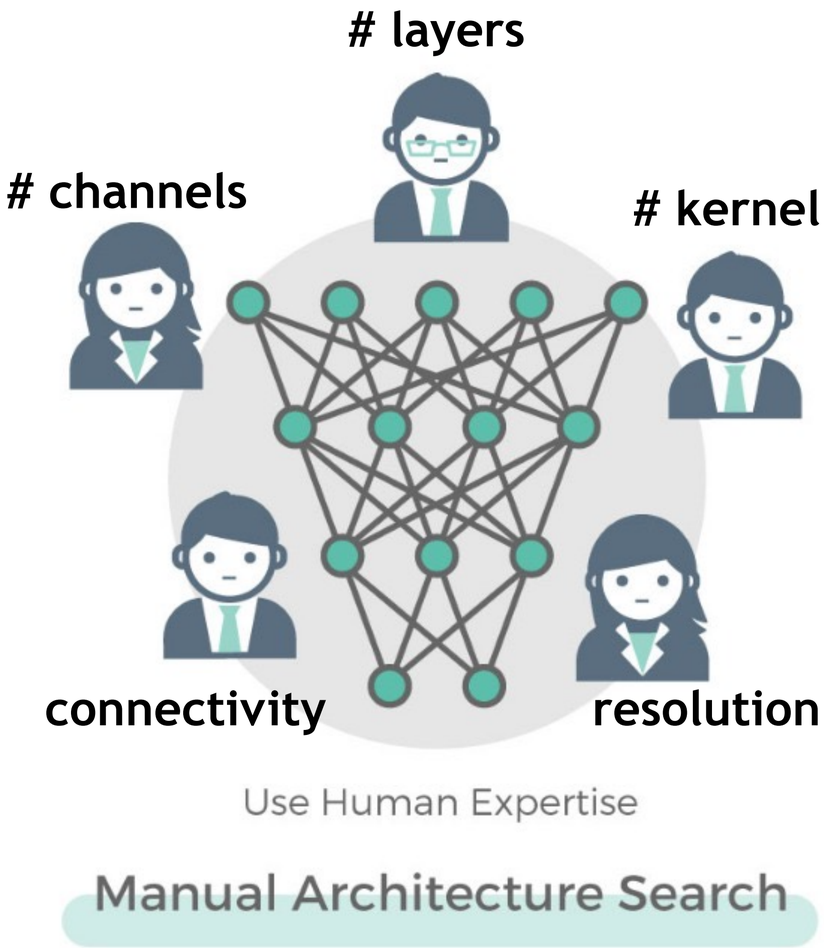
Neural Architecture Search

- **Primitive operations**
- **Classic building blocks**
- **Introduction to neural architecture search (NAS)**
  - What is NAS?
    - Search space
    - Design the search space
    - Search strategy
- Efficient and Hardware-aware NAS
  - Performance estimation strategy
  - Hardware-aware NAS
  - Zero-shot NAS
  - Neural-hardware architecture co-search
- NAS applications
  - NLP, GAN, point cloud, pose



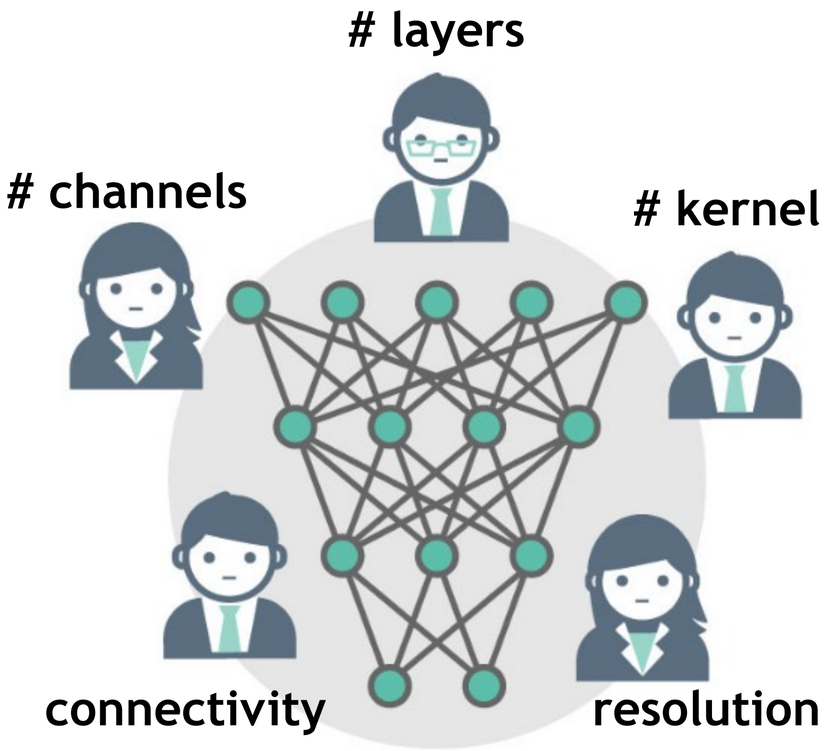
From Manual Design to Automatic Design

Huge design space, manual design is unscalable



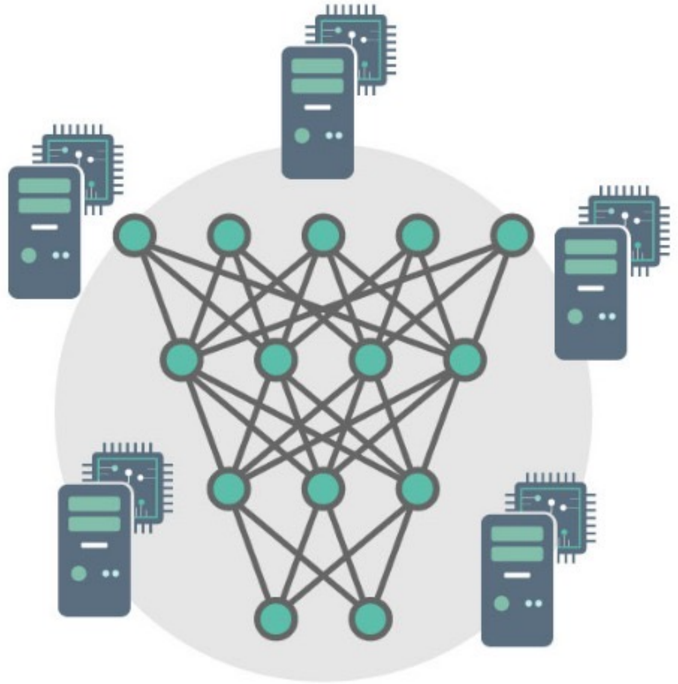
From Manual Design to Automatic Design

Huge design space, manual design is unscalable



Use Human Expertise

Manual Architecture Search

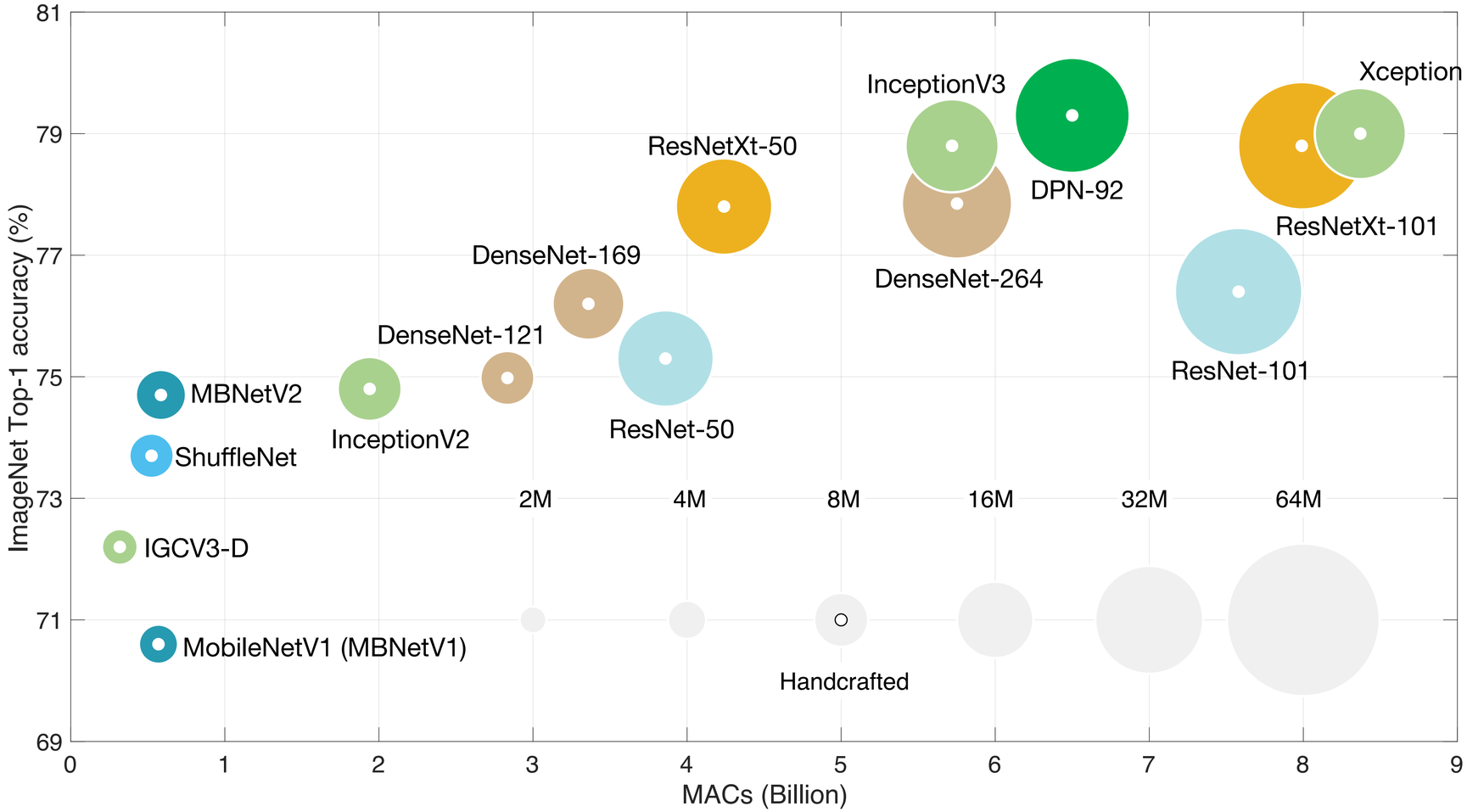


Use Machine Learning

Automatic Architecture Search

Manually-Designed Neural Networks

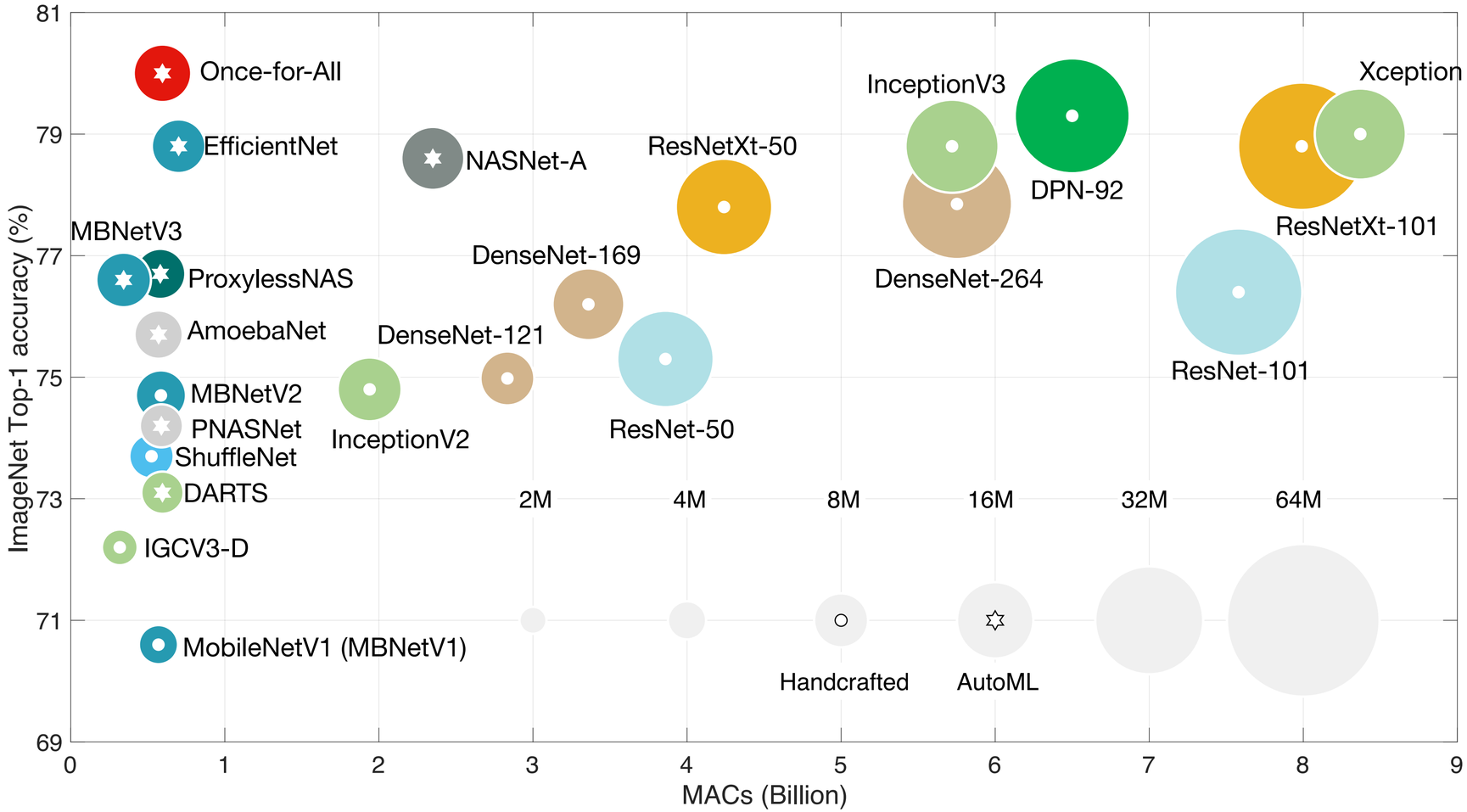
Accuracy-efficiency trade-off on ImageNet



Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey [Deng *et al.*, IEEE 2020]

From Manual Design to Automatic Design

Huge design space, manual design is unscalable

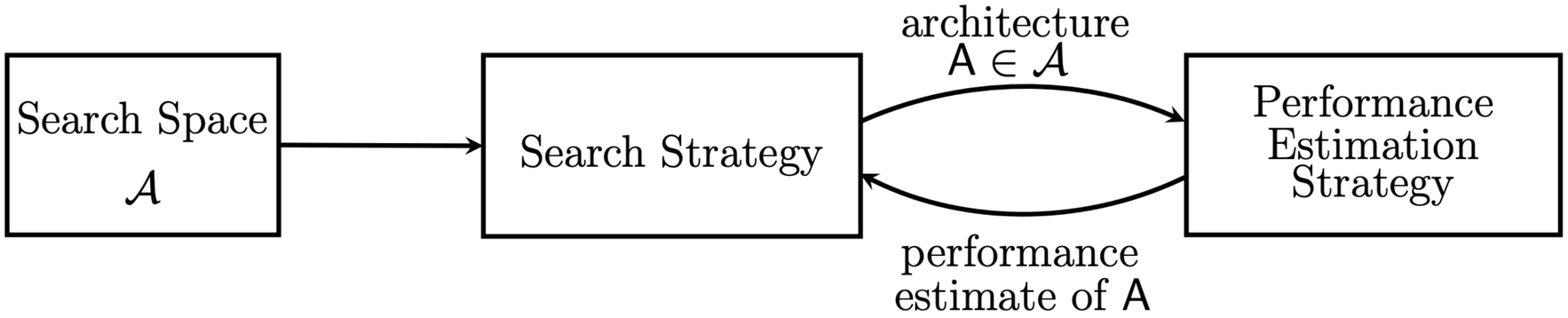


Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Illustration of NAS

Components and goal

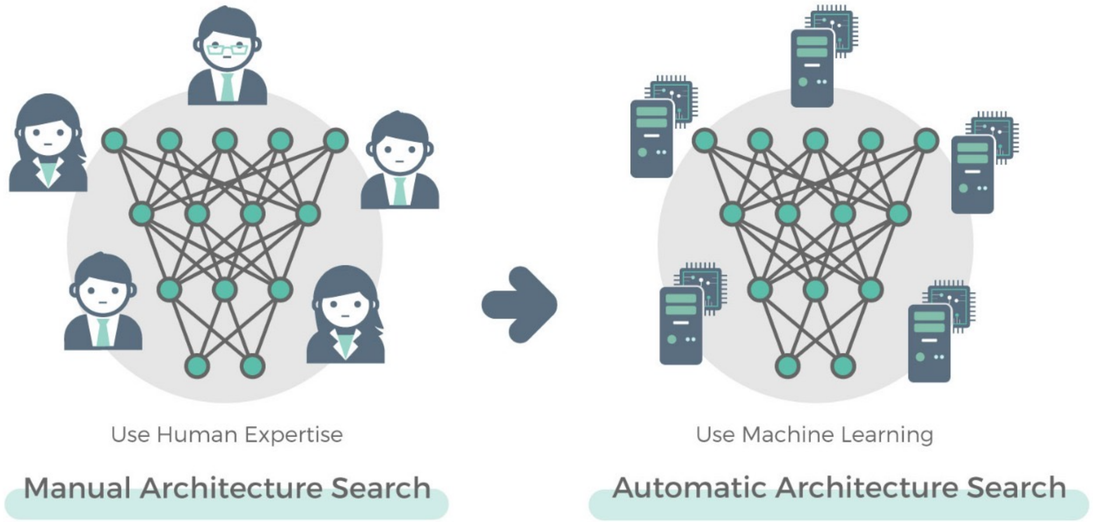
- The goal of NAS is to find the best neural network architecture in the search space, maximizing the objective of interest (e.g., accuracy, efficiency, etc).



Neural Architecture Search: A Survey [Elskan *et al.*, JMLR 2019]

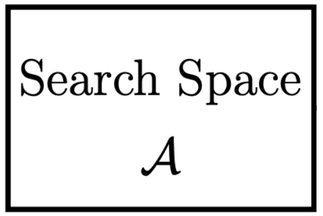
Neural Architecture Search

- **Primitive operations**
- **Classic building blocks**
- **Introduction to neural architecture search (NAS)**
  - What is NAS?
  - Search space
  - Design the search space
  - Search strategy
- Efficient and Hardware-aware NAS
  - Performance estimation strategy
  - Hardware-aware NAS
  - Zero-shot NAS
  - Neural-hardware architecture co-search
- NAS applications
  - NLP, GAN, point cloud, pose



Search Space

- Search space is a set of candidate neural network architectures.

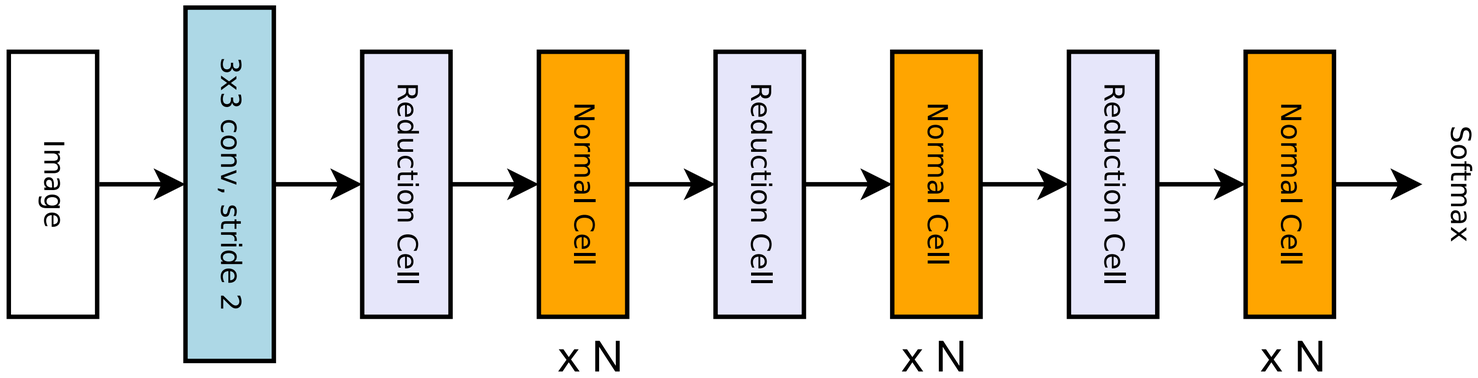


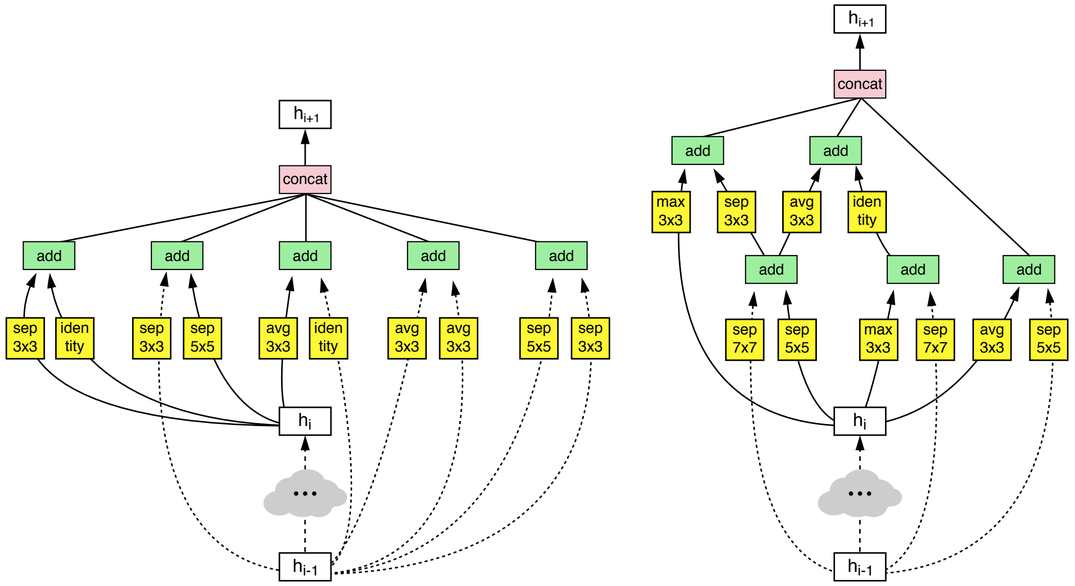
- Cell-level
- Network-level

Neural Architecture Search: A Survey [Elskan *et al.*, JMLR 2019]

Search Space

Cell-level search space





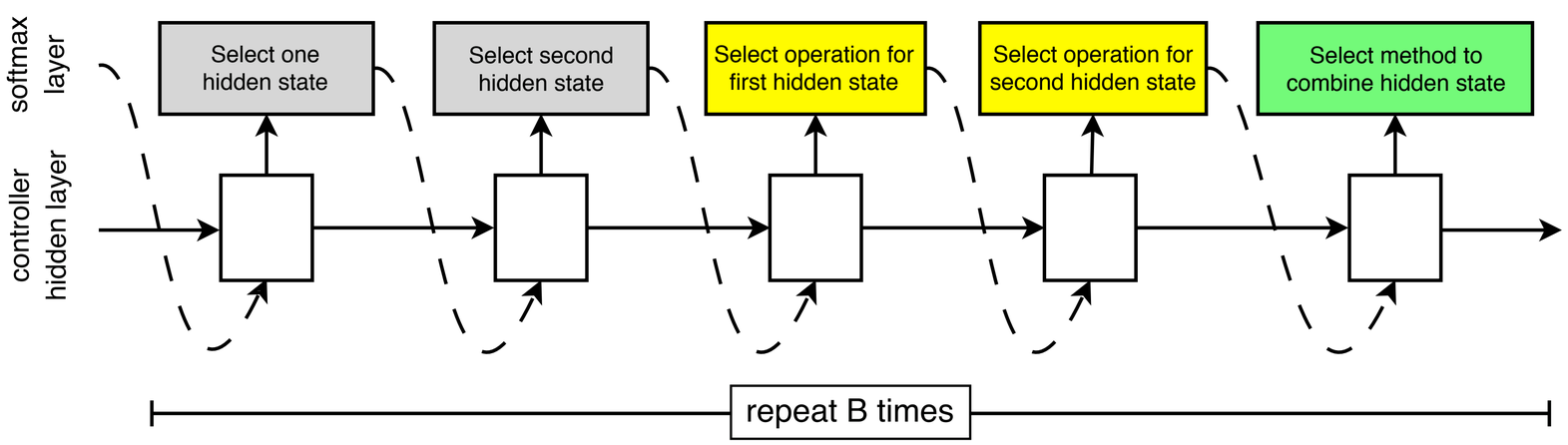
Normal Cell *Reduction Cell*

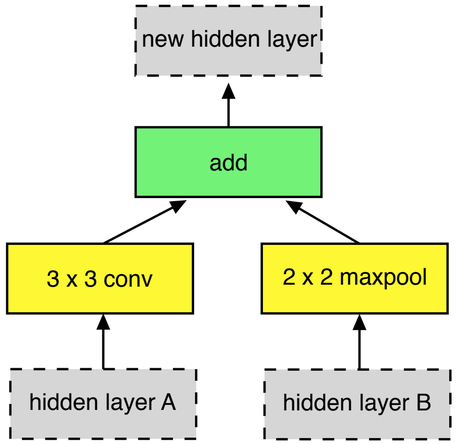
Learning Transferable Architectures for Scalable Image Recognition [Zoph *et al.*, CVPR 2018]

Search Space

Cell-level search space

- Left: **An RNN controller** generates the candidate cells five steps: finding two inputs, selecting two input transformation operations (e.g. convolution / pooling / identity), and finally selecting the method to combine the results. These five steps will be repeated for B times.
- Right: A cell generated after one step.



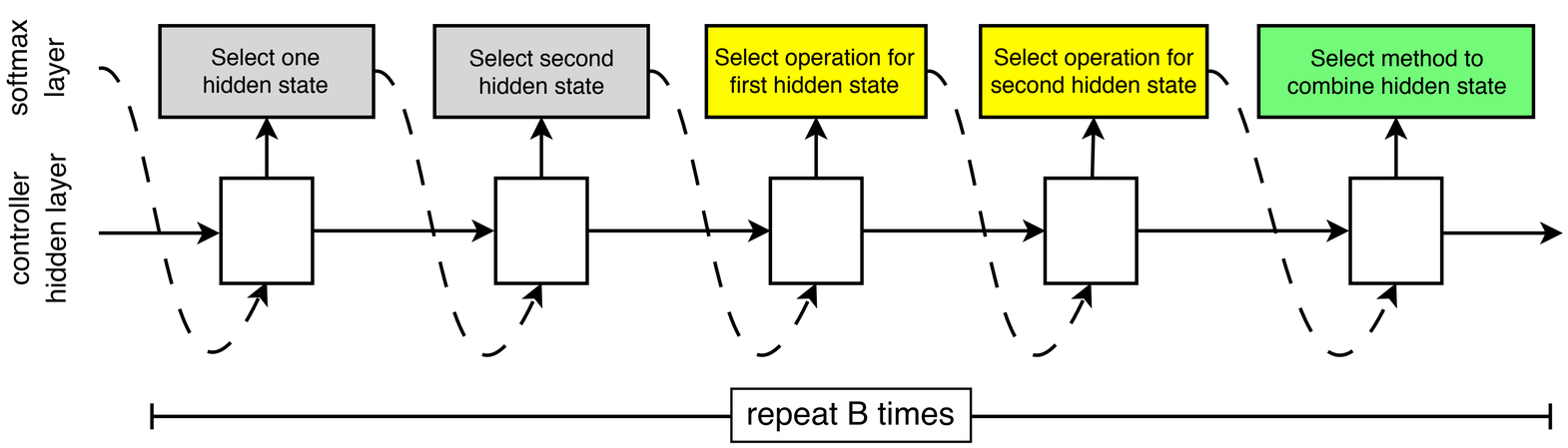


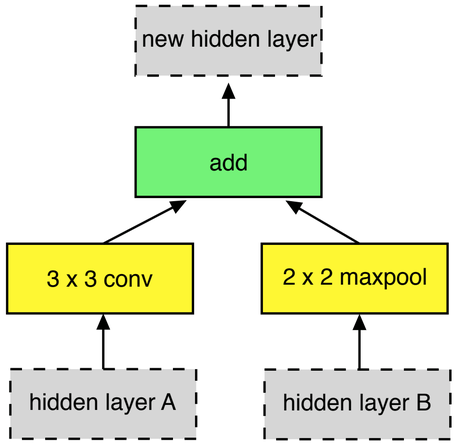
Learning Transferable Architectures for Scalable Image Recognition [Zoph *et al.*, CVPR 2018]

Search Space

Cell-level search space

- Question: Assuming that we have two candidate inputs, M candidate operations to transform the inputs and N potential operations to combine hidden states, what is the size of the search space in NASNet if we have B layers?
- Hint: Consider it step by step!



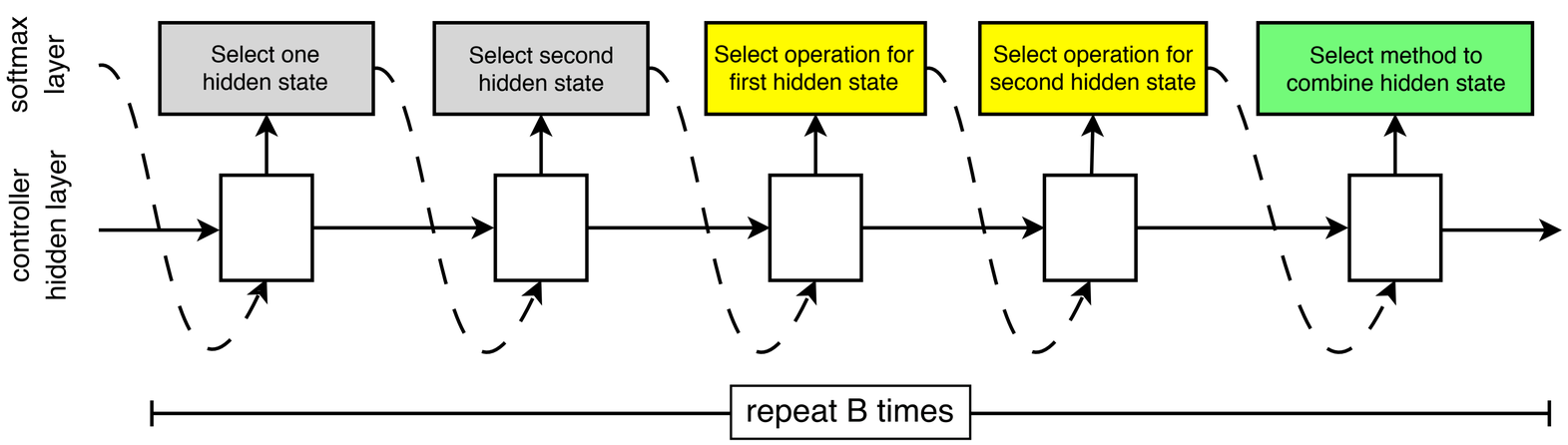


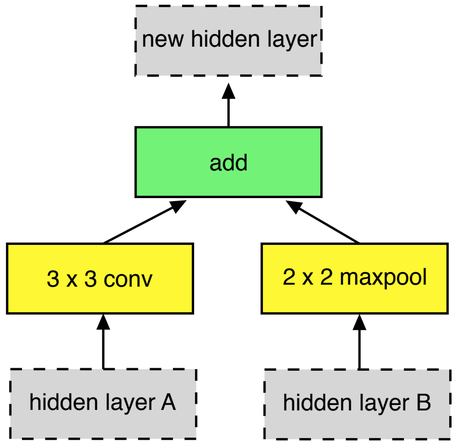
Learning Transferable Architectures for Scalable Image Recognition [Zoph *et al.*, CVPR 2018]

Search Space

Cell-level search space

- Question: Assuming that we have two candidate inputs, M candidate operations to transform the inputs and N potential operations to combine hidden states, what is the size of the search space in NASNet if we have B layers?
- Hint: Consider it step by step!
- Answer: $(2 \times 2 \times M \times M \times N)^B = 4^B M^{2B} N^B$.
- Assume M=5, N=2, B=5, we have 3.2×10^{11} candidates in the design space.

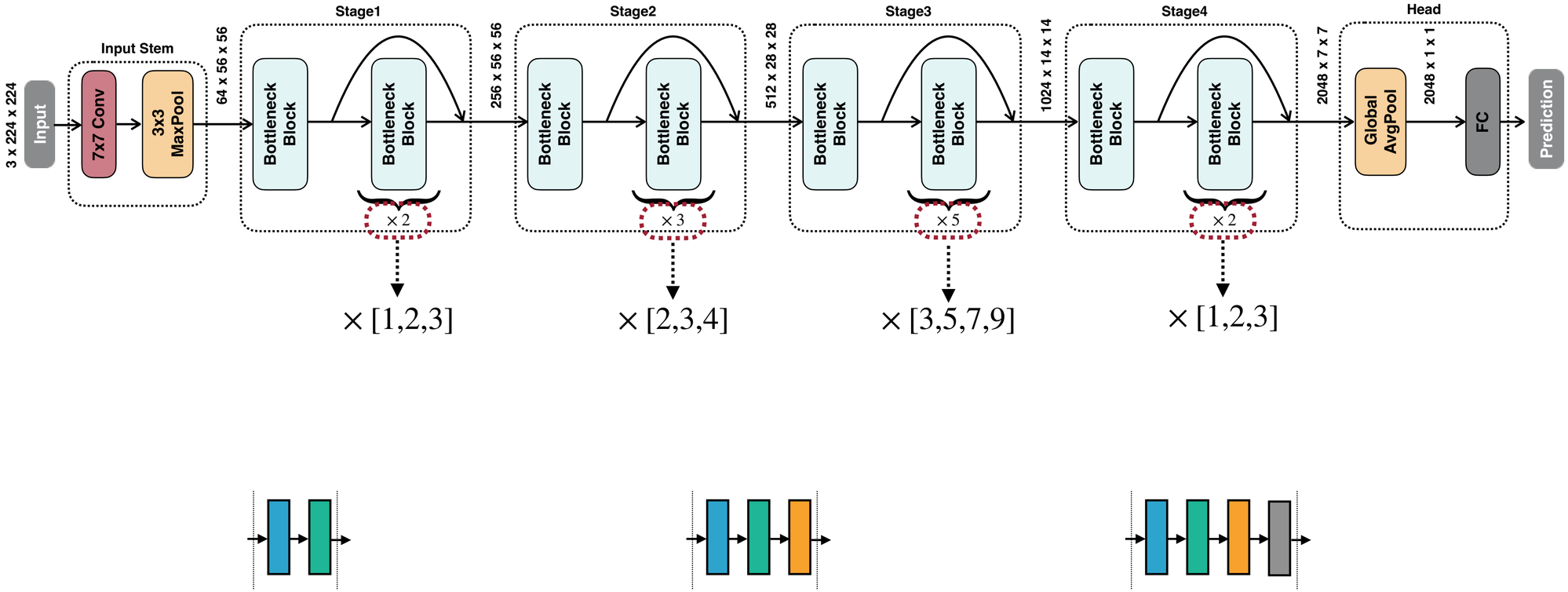




Learning Transferable Architectures for Scalable Image Recognition [Zoph *et al.*, CVPR 2018]

Search Space

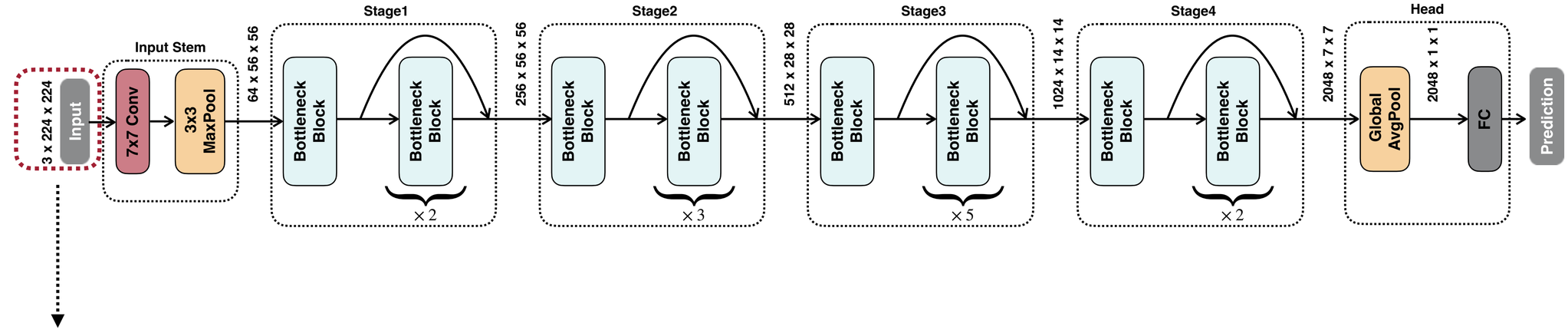
Network-level search space: depth dimension



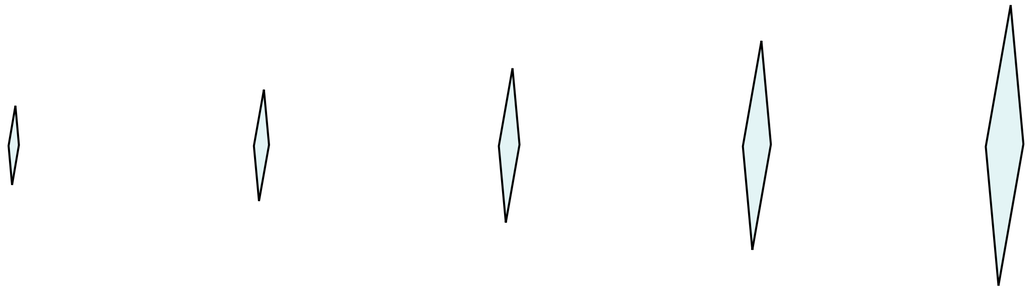
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Search Space

Network-level search space: resolution dimension



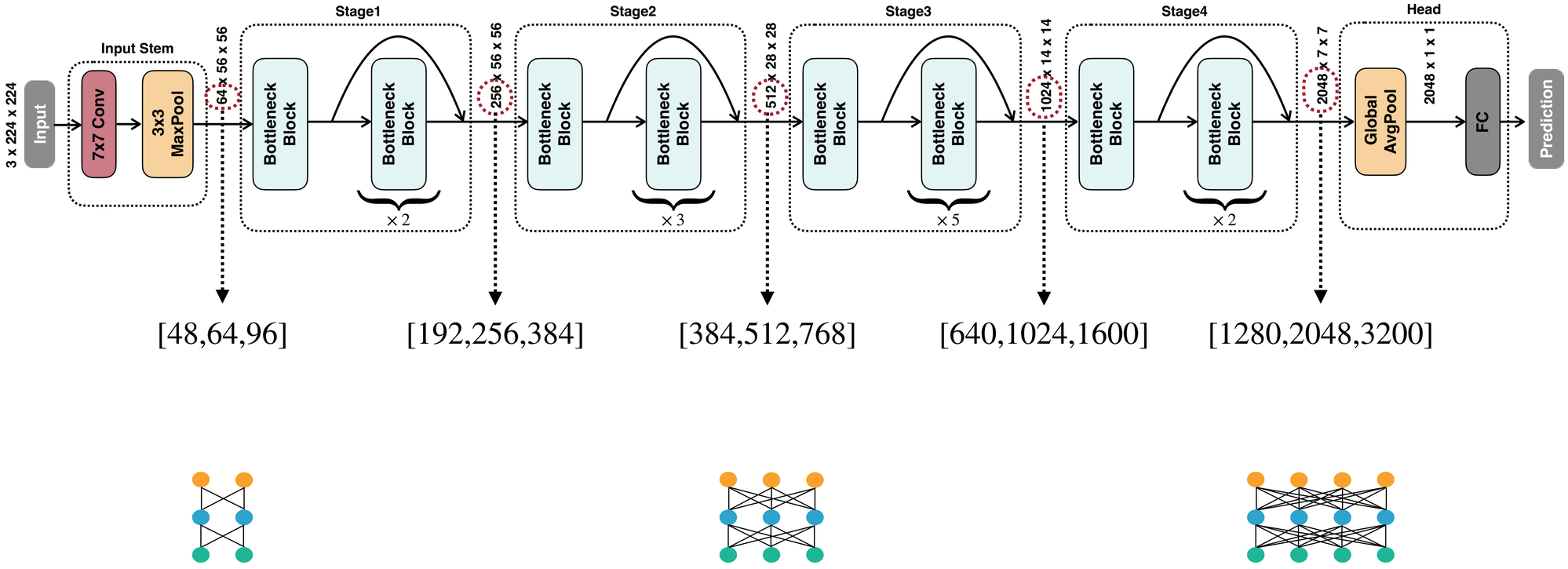
[(3,128,128); (3,160,160); (3,192,192); (3,224,224); (3,256,256)]



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Search Space

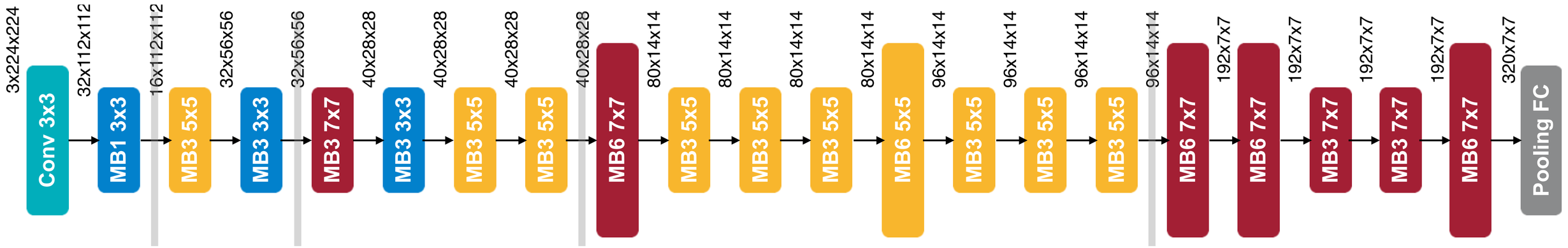
Network-level search space: width dimension



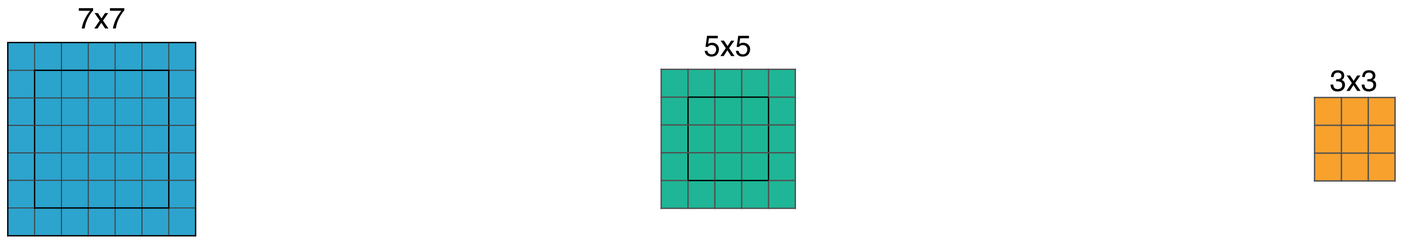
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Search Space

Network-level search space: kernel size dimension



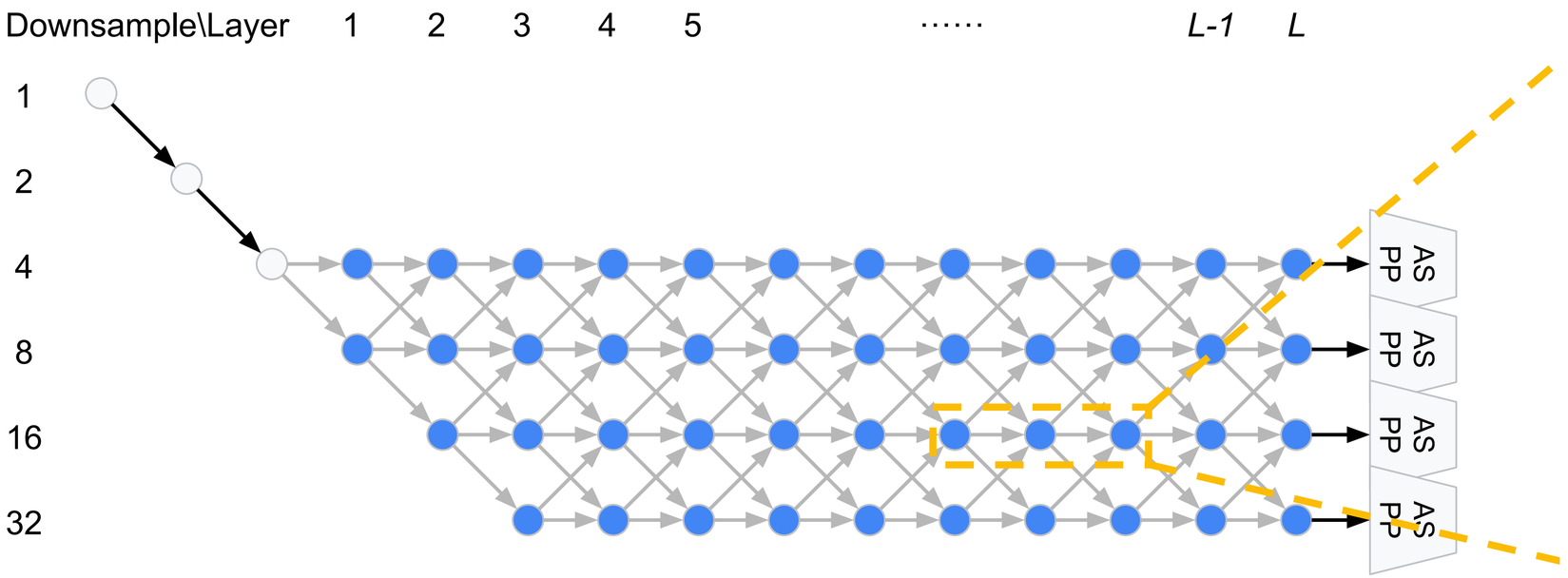
For models that use depthwise convolution, we can choose the kernel size for each depthwise convolution.

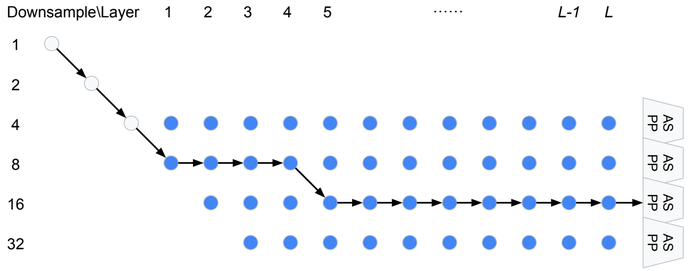


ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

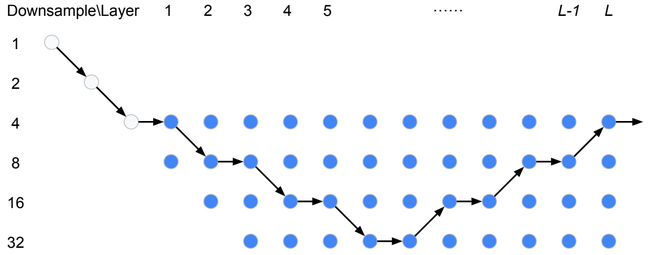
Search Space

Network-level search space: topology connection

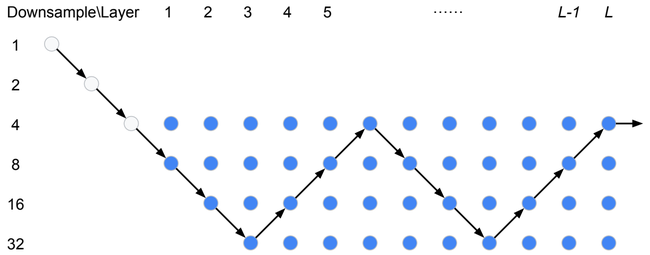




(a) Network level architecture used in DeepLabv3 [9].



(b) Network level architecture used in Conv-Deconv [56].



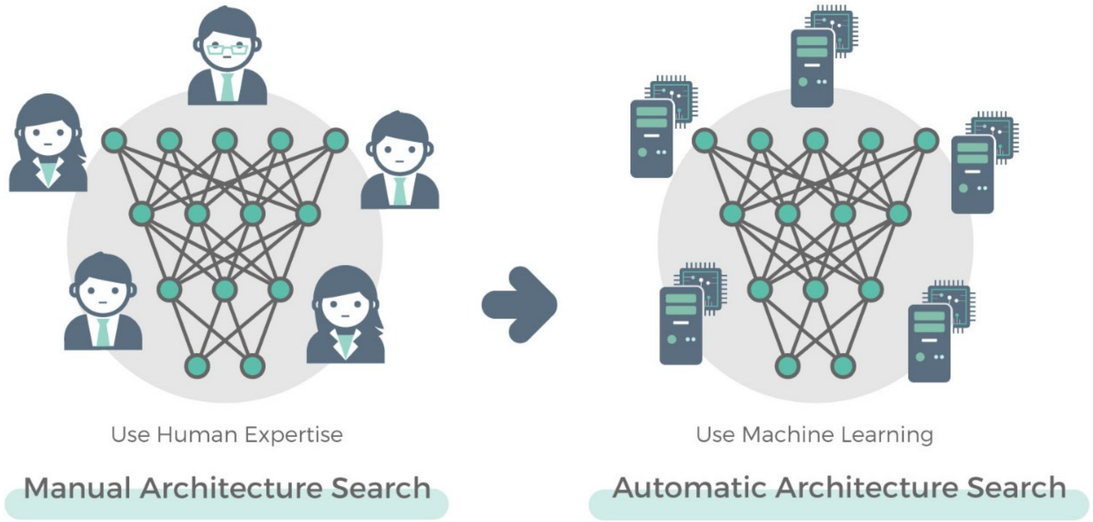
(c) Network level architecture used in Stacked Hourglass [55].

- Left: the network-level search space in AutoDeepLab. **Each path along the blue nodes** corresponds to a network architecture in the search space.
- Right: representative manual designs can be represented using this formulation. E.g.: DeepLabV3, UNet, Stacked Hourglass.

Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation [Liu *et al*., CVPR 2019]

Neural Architecture Search

- **Primitive operations**
- **Classic building blocks**
- **Introduction to neural architecture search (NAS)**
  - What is NAS?
  - Search space
  - Design the search space
  - Search strategy
- Efficient and Hardware-aware NAS
  - Performance estimation strategy
  - Hardware-aware NAS
  - Zero-shot NAS
  - Neural-hardware architecture co-search
- NAS applications
  - NLP, GAN, point cloud, pose

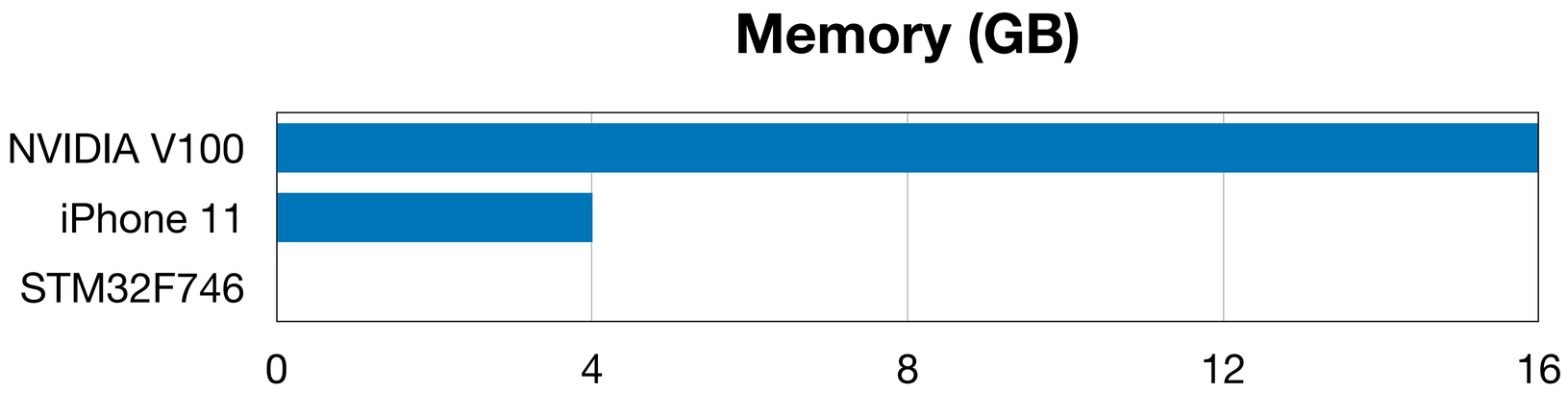


Design the Search Space

Design the Search Space

Design the search space for TinyML: memory is important for TinyML

	Cloud AI (NVIDIA V100)	→	Mobile AI (iPhone 11)	→	Tiny AI (STM32F746)		ResNet-50	MobileNetV2	MobileNetV2 (int8)
Memory	16 GB	4× →	4 GB	3100× →	**320 kB**	← gap →	7.2 MB	6.8 MB	1.7 MB
Storage	TB~PB	1000× →	>64 GB	64000× →	**1 MB**	← gap →	102MB	13.6 MB	3.4 MB



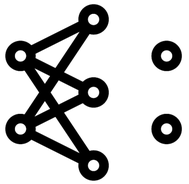
MCUNet: Tiny Deep Learning on IoT Devices [Lin *et al.*, NeurIPS 2020]

Design the Search Space

Design the search space for TinyML: memory is important for TinyML

Different from Mobile AI

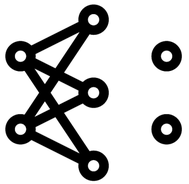
+



Latency Constraint

Energy Constraint

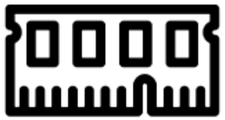
+



Latency Constraint

Energy Constraint

Memory Constraint



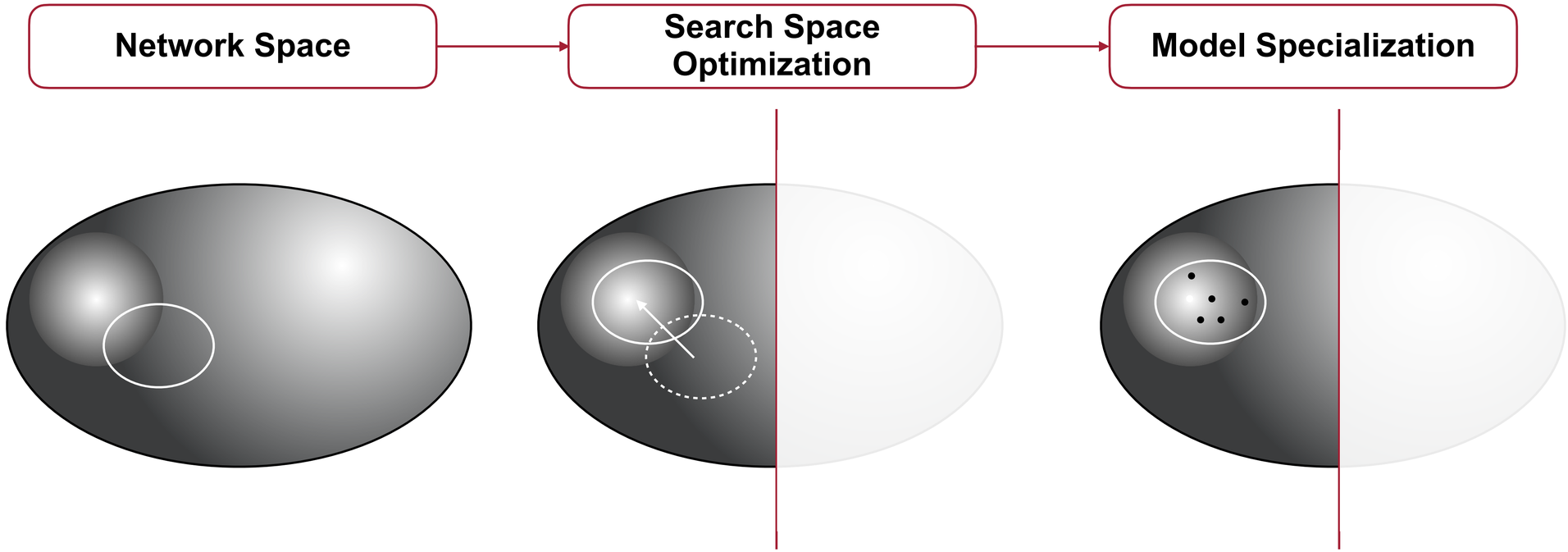
MCUNet: Tiny Deep Learning on IoT Devices [Lin *et al.*, NeurIPS 2020]

Design the Search Space

Design the search space for TinyML

Search space design is crucial for NAS performance
There is no prior expertise on MCU model design

TinyNAS:
(1) Automated search space optimization
(2) Resource-constrained model specialization

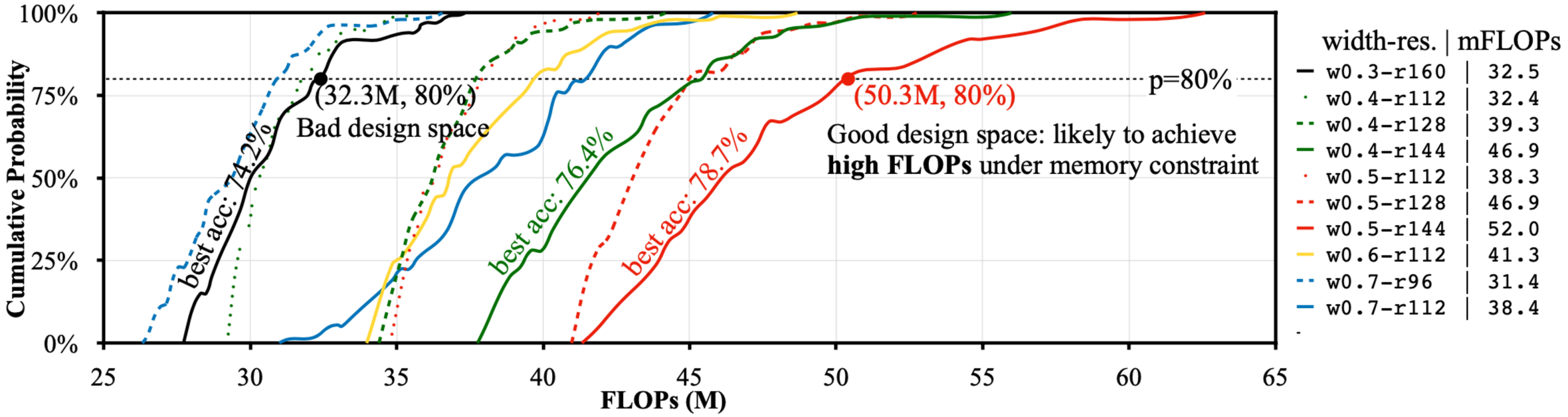


MCUNet: Tiny Deep Learning on IoT Devices [Lin *et al.*, NeurIPS 2020]

Design the Search Space

Design the search space for TinyML

Analyzing **FLOPs distribution** of satisfying models:
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy



MCUNet: Tiny Deep Learning on IoT Devices [Lin *et al.*, NeurIPS 2020]

Design the Search Space

Design the search space for TinyML

- **Discussion**: what are the advantages / disadvantages of these two methods (RegNet, MCUNet) for search space design?

	R-18@224	Rand Space	Huge Space	Our Space
Acc.	80.3%	74.7±1.9%	77.0%	**78.7%**

Table 5. Our search space achieves the best accuracy, closer to ResNet-18@224 resolution (OOM). Randomly sampled and a huge space (contain many configs) leads to worse accuracy.

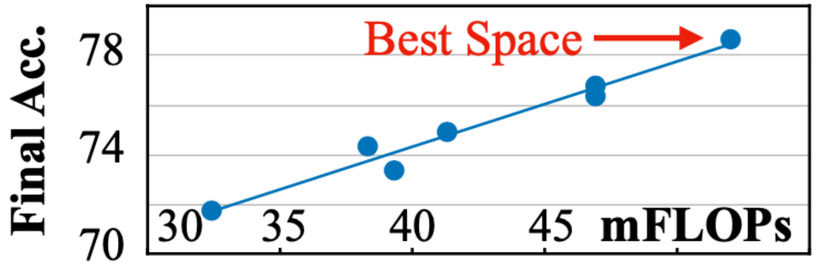


Figure 8. Search space with higher mean FLOPs leads to higher final accuracy.

Better search space, better final accuracy.

MCUNet: Tiny Deep Learning on IoT Devices [Lin *et al.*, NeurIPS 2020]

Neural Architecture Search

- **Primitive operations**
- **Classic building blocks**
- **Introduction to neural architecture search (NAS)**
  - What is NAS?
  - Search space
  - Design the search space
  - Search strategy
- Efficient and Hardware-aware NAS
  - Performance estimation strategy
  - Hardware-aware NAS
  - Zero-shot NAS
  - Neural-hardware architecture co-search
- NAS applications
  - NLP, GAN, point cloud, pose

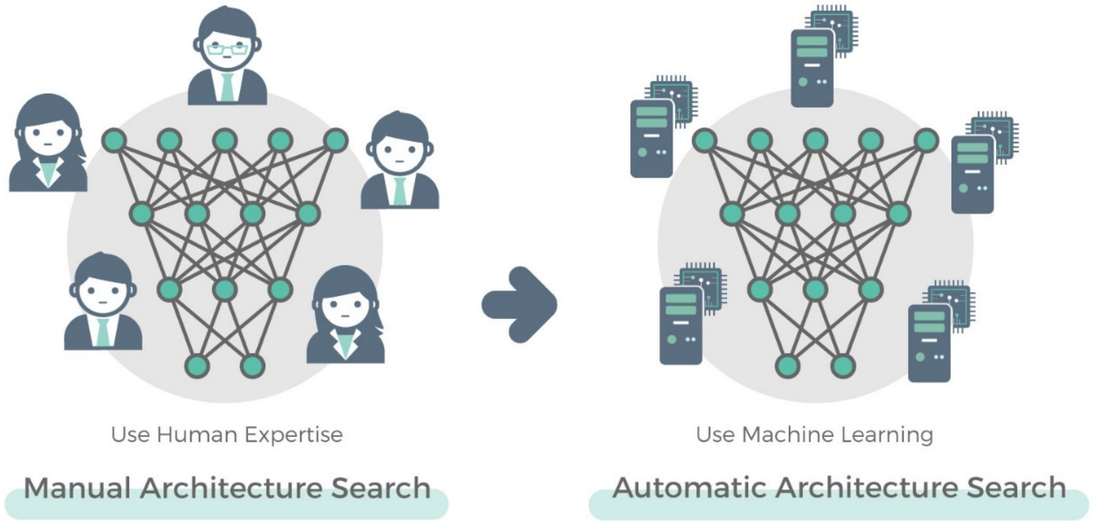
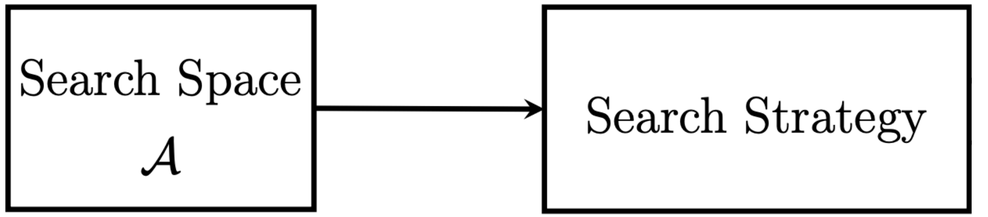


Illustration of NAS

Search strategy

- Search space is a set of candidate neural network architectures.
- Search strategy defines how to explore the search space.



- Cell-level
- Network-level

- Grid search
- Random search
- Reinforcement learning
- Gradient descent
- Evolutionary search

Neural Architecture Search: A Survey [Elskan *et al.*, JMLR 2019]

Search Strategy

Grid search

- Grid search is the traditional way of hyper parameter optimization. The entire design space is represented as the Cartesian product of single dimension design spaces (e.g. resolution in [1.0x, 1.1x,1.2x], width in [1.0x, 1.1x, 1.2x]).
- To obtain the accuracy of each candidate network, we train them from scratch.

Resolution / Width	**1.0x**	**1.1x**	**1.2x**
1.0x	50.0%	53.0%	**54.9%**
1.1x	51.0%	53.5%	55.4%
1.2x	52.0%	54.1%	56.2%

Satisfies the latency constraint

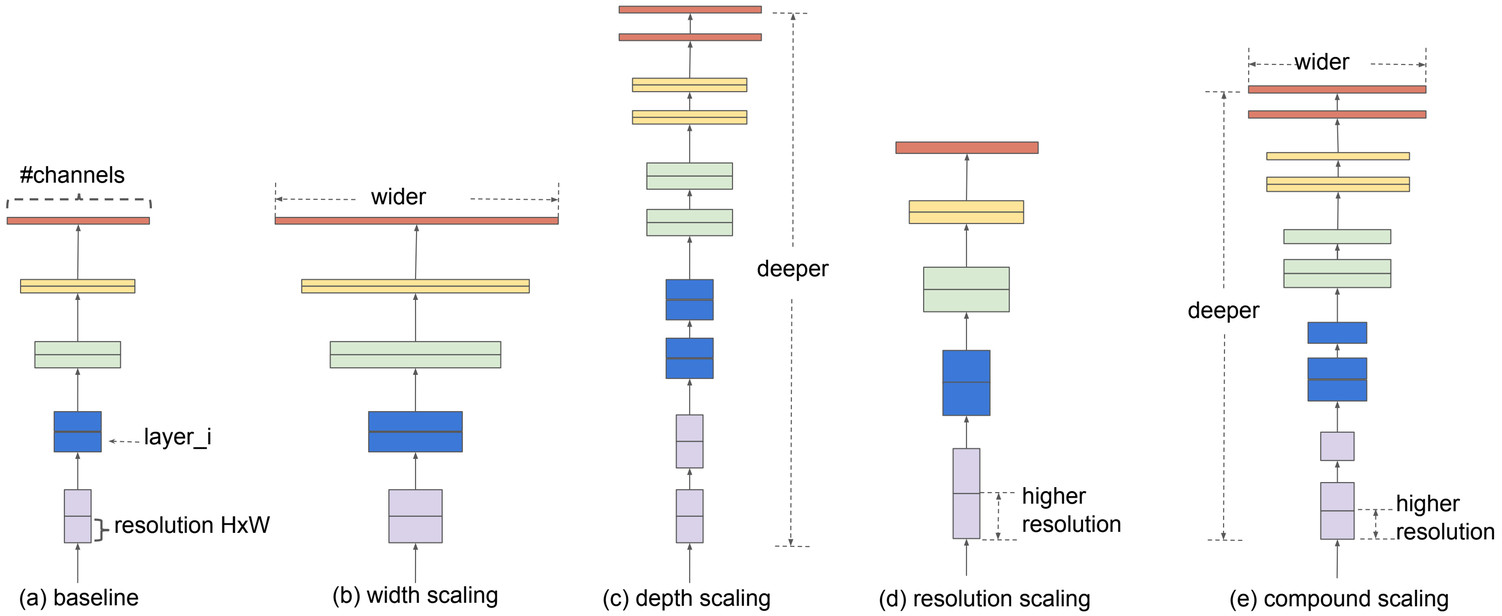
Breaks the latency constraint

Numbers in the grids correspond to the accuracy of candidate networks.

Search Strategy

Grid search

- EfficientNet applies **compound scaling** on **depth, width and resolution** to a starting network. It performs **grid search** on α, β, γ values such that the total FLOPs of the new model will be $2\times$ of the original network.



$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks [Tan and Le, ICML 2019]

Search Strategy

Random search

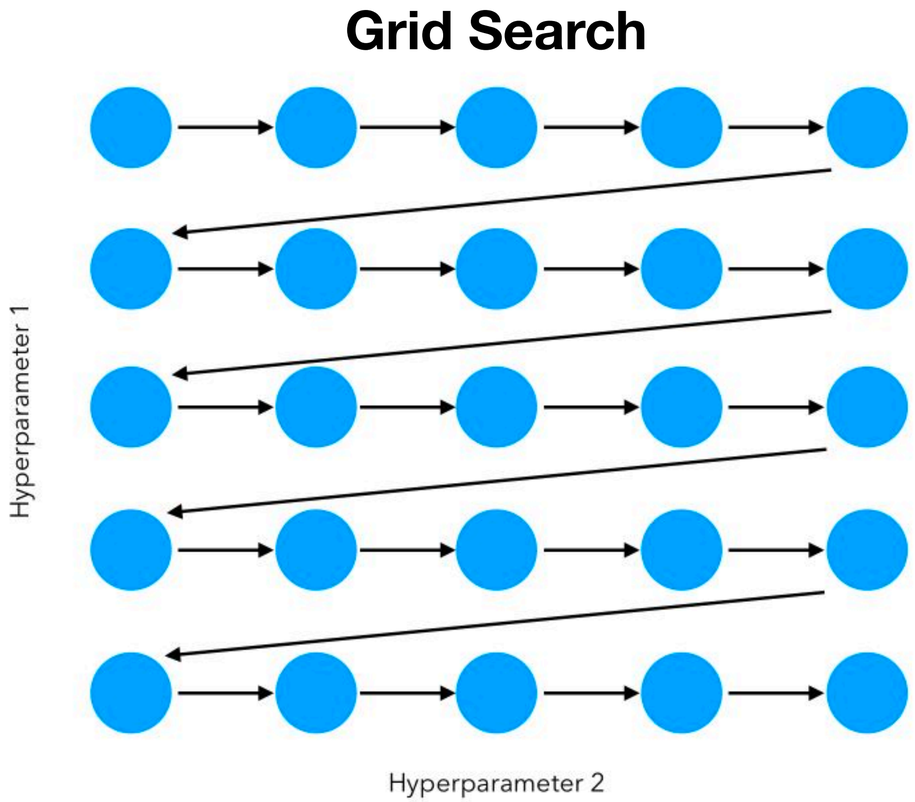


image source: 1

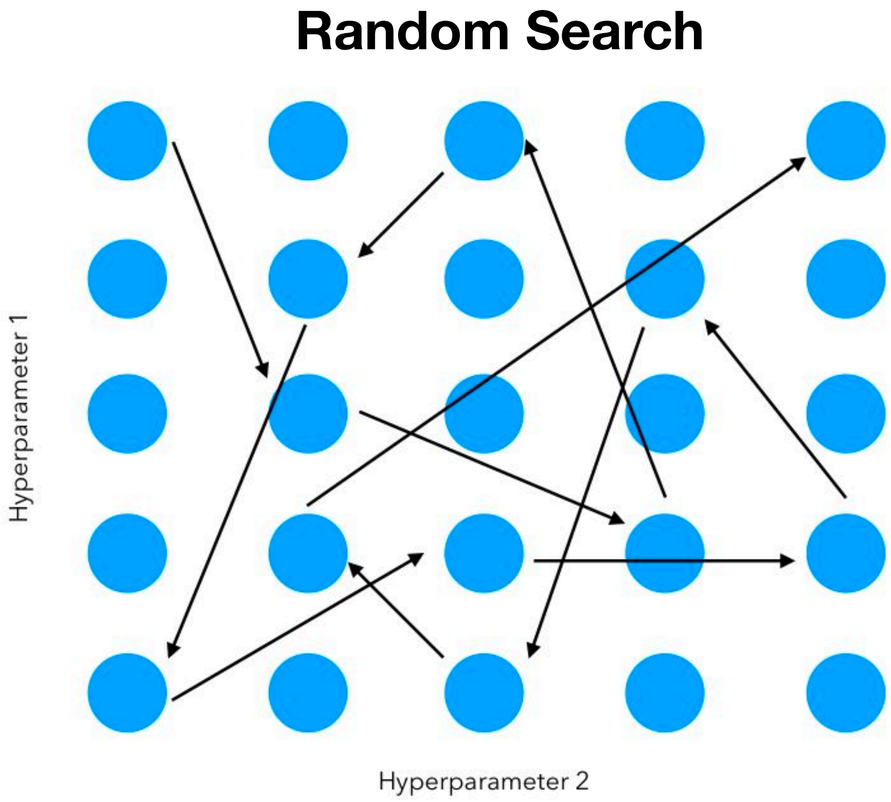
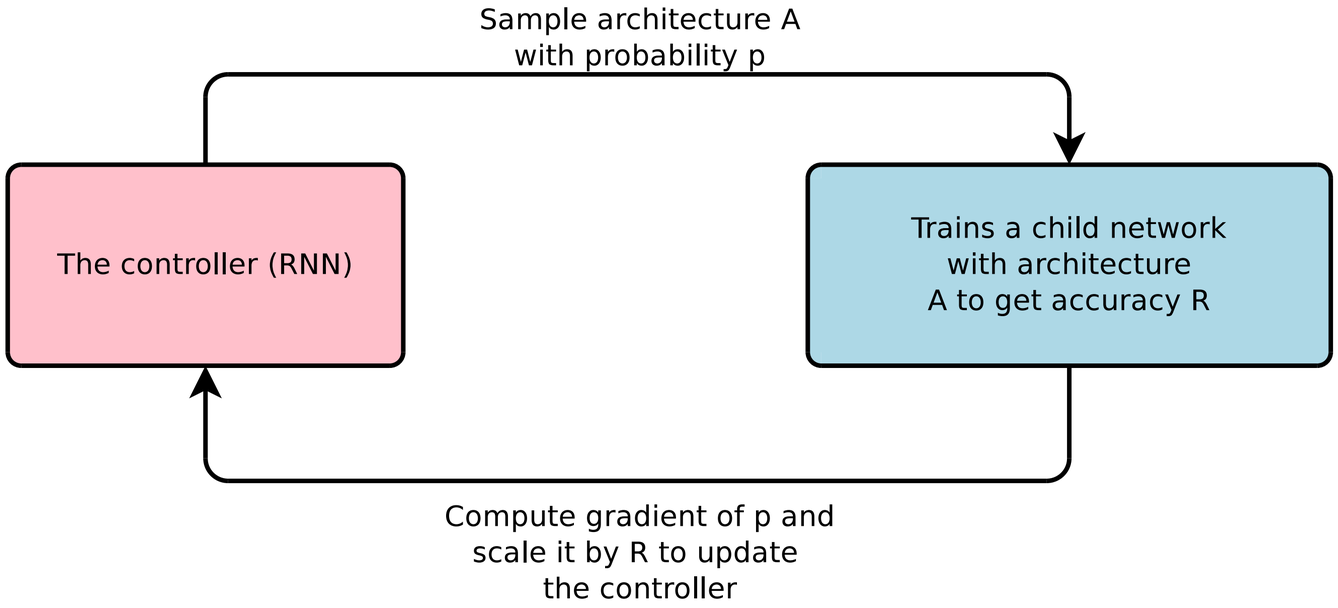


image source: 2

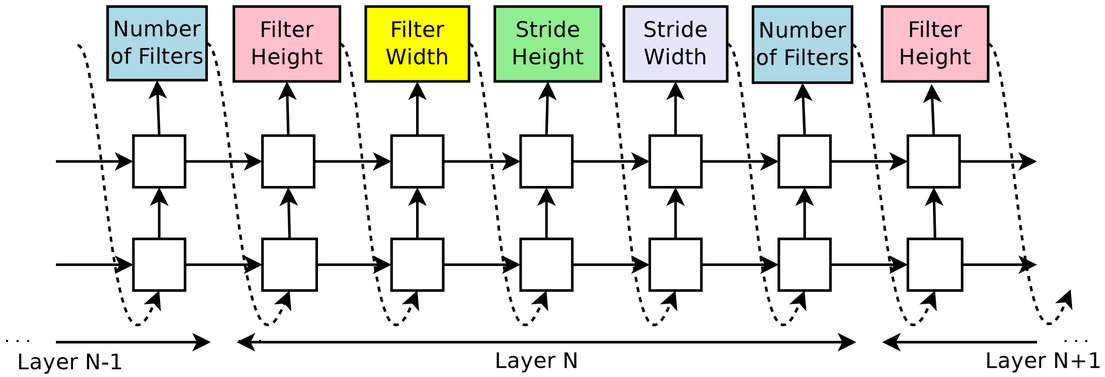
Search Strategy

Reinforcement learning

- Model neural architecture design as a sequential decision-making problem.
- Use reinforcement learning to train the controller that is implemented with an RNN.



Overview of RL-based NAS



The RNN controller

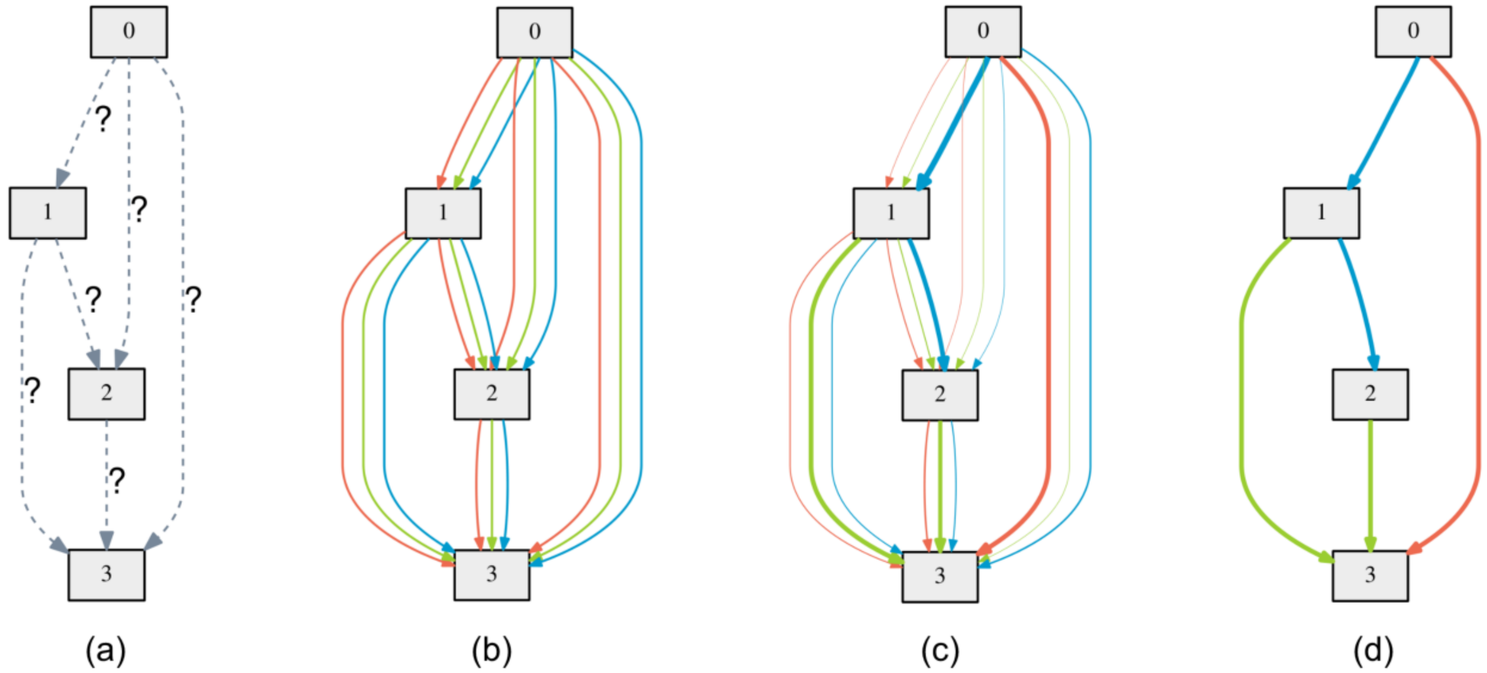
Neural Architecture Search with Reinforcement Learning [Zoph and Le, ICLR 2017]

Search Strategy

Gradient descent

- Represent the output at each node as a weighted sum of outputs from different edges.

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

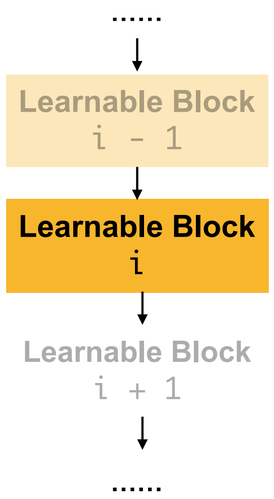


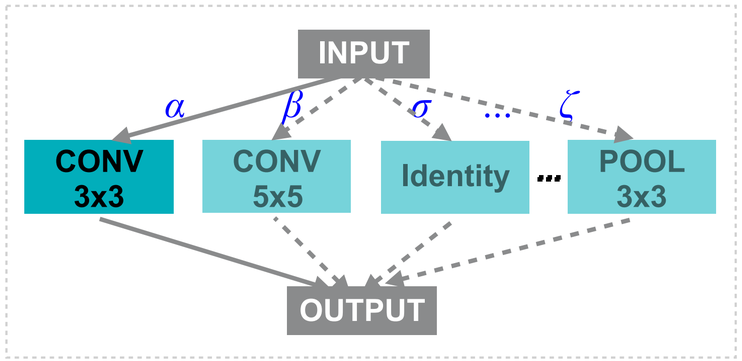
DARTS: Differentiable Architecture Search [Liu *et al.*, ICLR 2019]

Search Strategy

Gradient descent

- It is also possible to **take latency into account** for gradient-based search.
- Here, F is a latency prediction model (typically a regressor or a lookup table). With such formulation, we can calculate an additional gradient for the architecture parameters from the latency penalty term.





$$\mathbb{E}[\text{Latency}] = \alpha \times F(\text{conv\_3x3}) + \beta \times F(\text{conv\_5x5}) + \sigma \times F(\text{identity}) + \ldots\ldots + \zeta \times F(\text{pool\_3x3})$$

$$\mathbb{E}[\text{latency}] = \sum_i \mathbb{E}[\text{latency}_i]$$

$$Loss = Loss_{CE} + \lambda_1 ||w||_2^2 + \lambda_2 \mathbb{E}[\text{latency}]$$

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Search Strategy

Evolutionary search

- Mimic the evolution process in biology

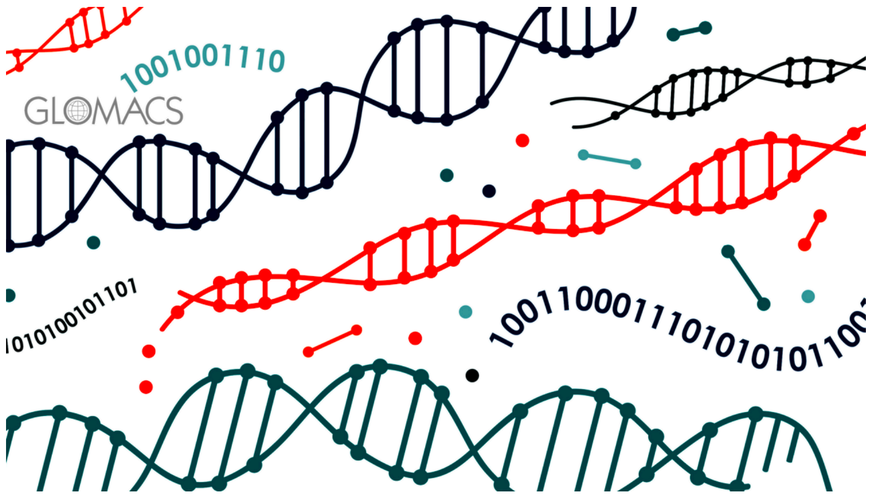


image source: 1

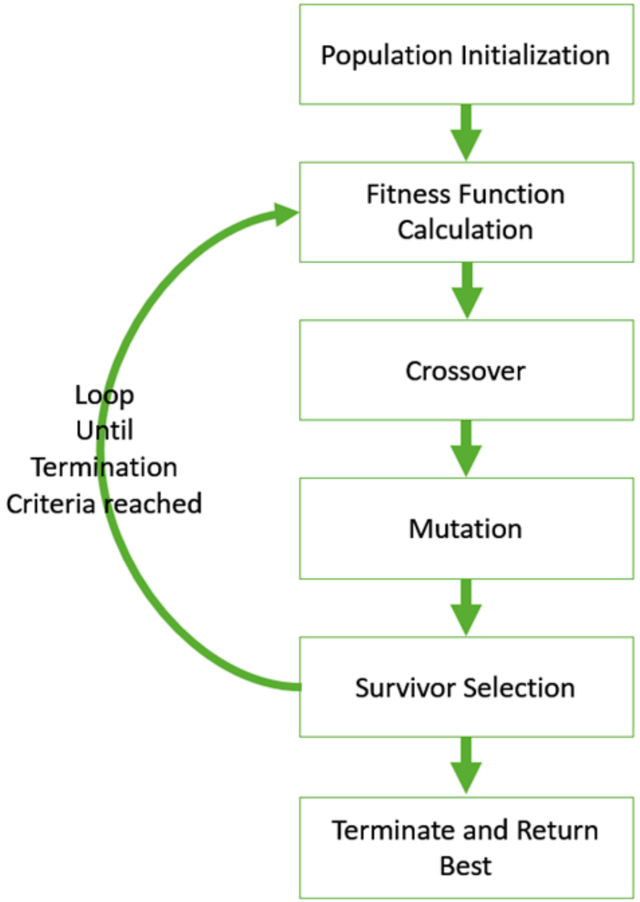
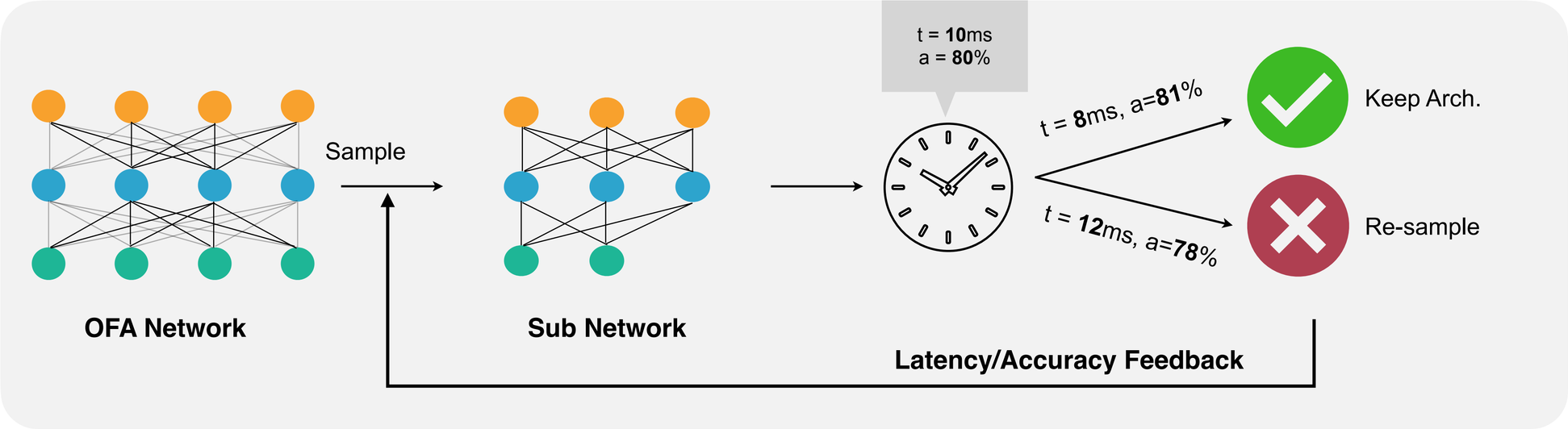


image source: 2

Search Strategy

Evolutionary search: fitness function

- Fitness function: F(accuracy, efficiency)



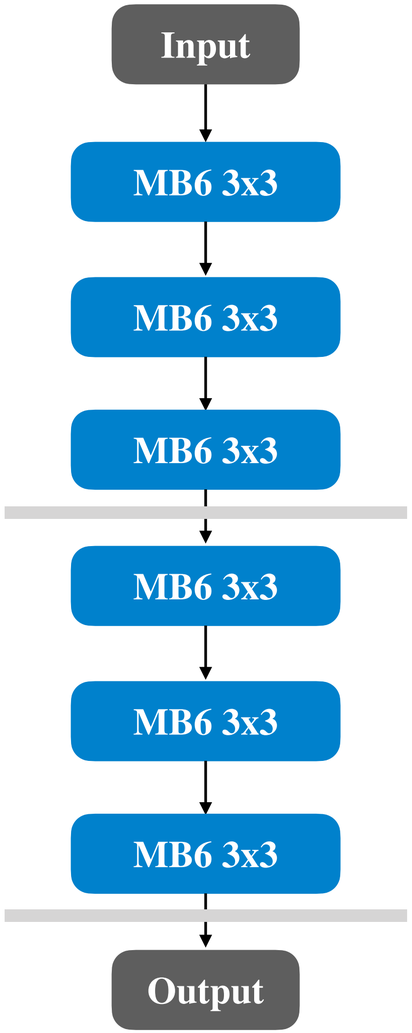
PVNAS: 3D Neural Architecture Search with Point-Voxel Convolution [Liu *et al.*, TPAMI 2021]

Search Strategy

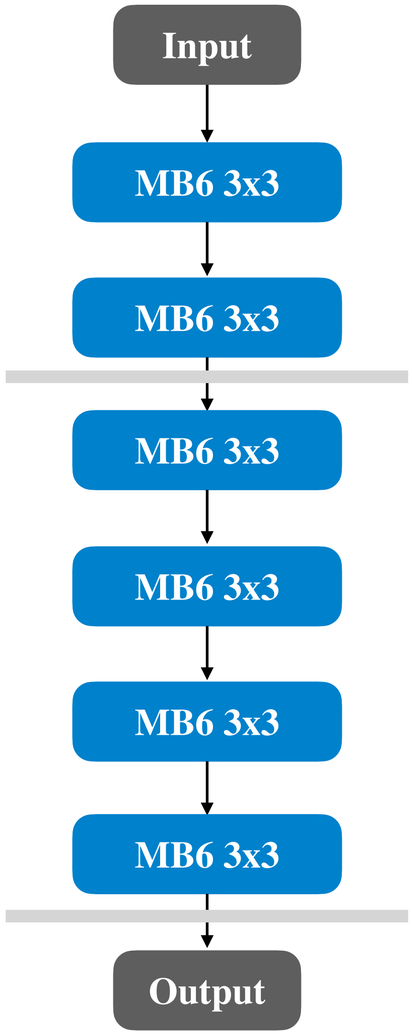
Evolutionary search: mutation

Stage 1: depth = 3

Stage 2: depth = 3



Mutation on depth



Stage 1: depth = 2

Stage 2: depth = 4

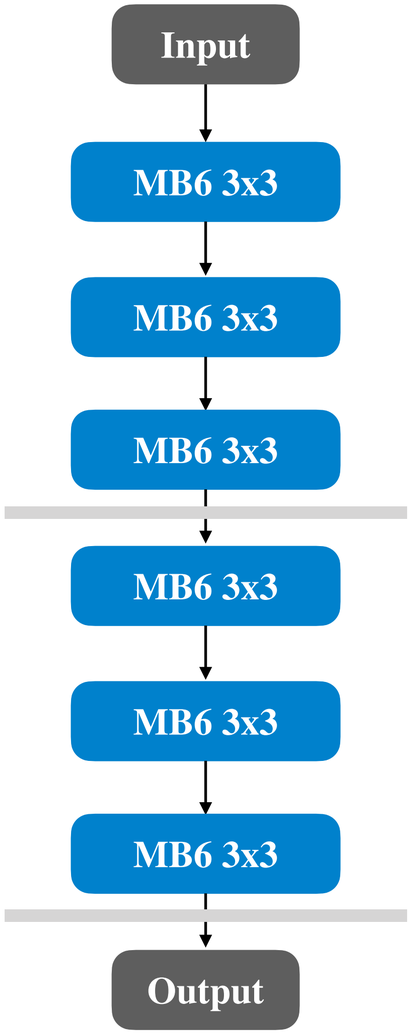
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Search Strategy

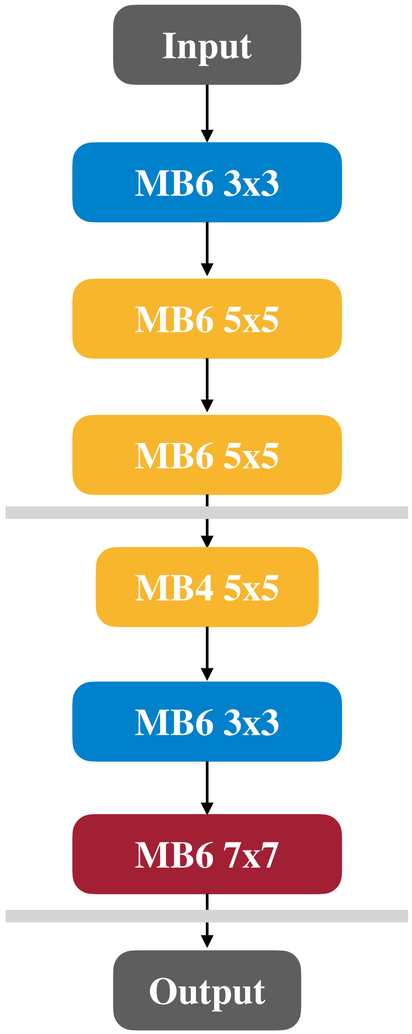
Evolutionary search: mutation

Stage 1: depth = 3

Stage 2: depth = 3



Mutation on operator



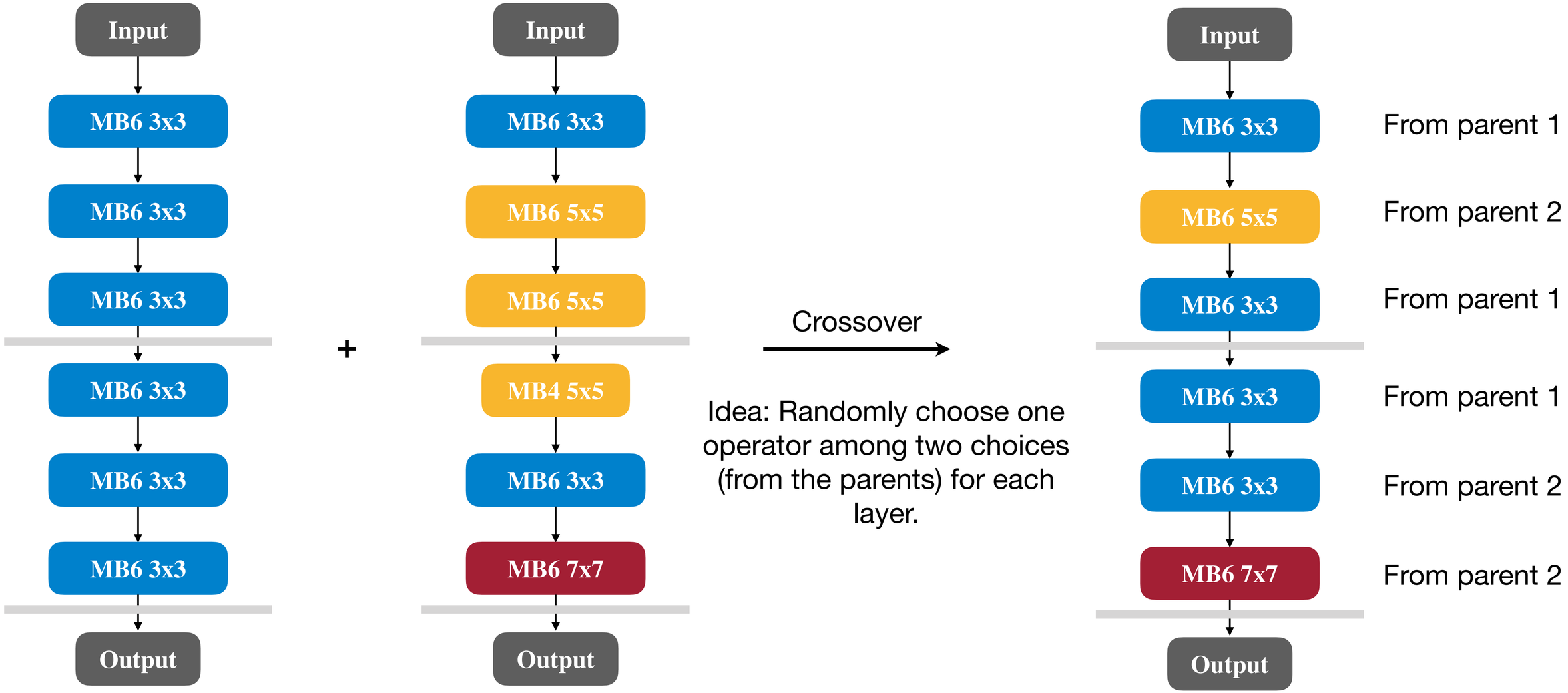
Stage 1: depth = 3

Stage 2: depth = 3

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Search Strategy

Evolutionary search: crossover

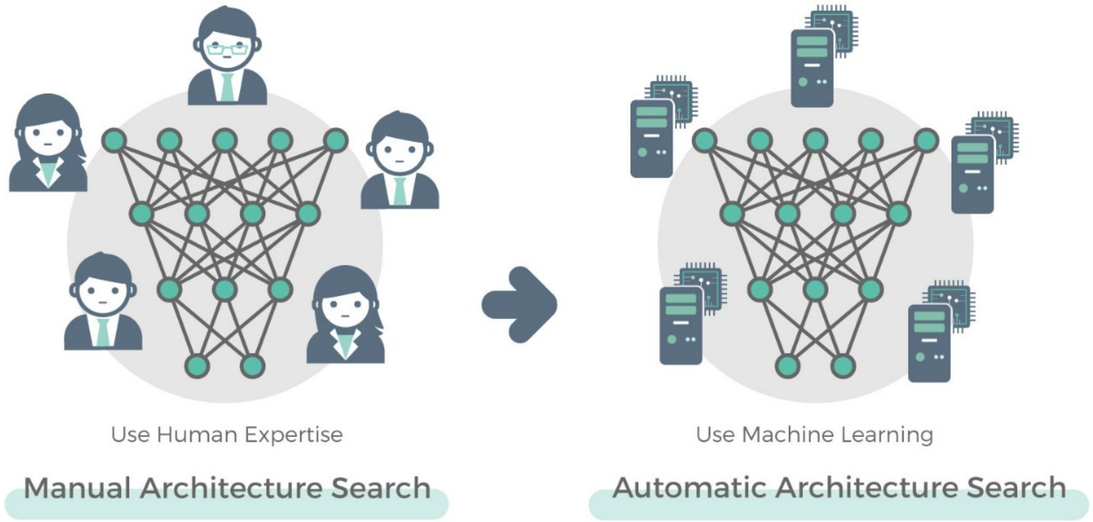


Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Summary of Today's Lecture

- Primitive operations
- Classic building blocks
- NAS search space
- Design the search space
- NAS search strategy

- We will cover in later lectures:
  - Performance estimation strategy in NAS
  - Hardware-aware NAS
  - Zero-shot NAS
  - Neural-hardware architecture search
  - NAS applications



References

1. Deep Residual Learning for Image Recognition [He et al., CVPR 2016]
2. ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky et al., NeurIPS 2012]
3. Very Deep Convolutional Networks for Large-scale Image Recognition [Simonyan et al., ICLR 2015]
4. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size [Iandola et al., arXiv 2016]
5. Aggregated Residual Transformations for Deep Neural Networks [Xie et al., CVPR 2017]
6. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Howard et al., arXiv 2017]
7. MobileNetV2: Inverted Residuals and Linear Bottlenecks [Sandler et al., CVPR 2018]
8. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices [Zhang et al., CVPR 2018]
9. Neural Architecture Search: A Survey [Elskan et al., JMLR 2019]
10. Learning Transferable Architectures for Scalable Image Recognition [Zoph et al., CVPR 2018]
11. DARTS: Differentiable Architecture Search [Liu et al., ICLR 2019]

References

- 12. MnasNet: Platform-Aware Neural Architecture Search for Mobile [Tan *et al.*, CVPR 2019]
- 13. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]
- 14. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search [Wu et al., CVPR 2019]
- 15. Designing Network Design Spaces [Radosavovic *et al.*, CVPR 2020]
- 16. Single Path One-Shot Neural Architecture Search with Uniform Sampling [Guo *et al.*, ECCV 2020]
- 17. Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]
- 18. Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation [Liu *et al*., CVPR 2019]
- 19. NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection [Ghaisi et al., CVPR 2019]
- 20. Exploring Randomly Wired Neural Networks for Image Recognition [Xie et al., ICCV 2019]
- 21. MCUNet: Tiny Deep Learning on IoT Devices [Lin et al., NeurIPS 2020]

References

- 22. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks [Tan and Le, ICML 2019]
- 23. Neural Architecture Search with Reinforcement Learning [Zoph and Le, ICLR 2017]
- 24. PVNAS: 3D Neural Architecture Search with Point-Voxel Convolution [Liu *et al.*, TPAMI 2021]
- 25. Regularized Evolution for Image Classifier Architecture Search [Real *et al.*, AAAI 2019]