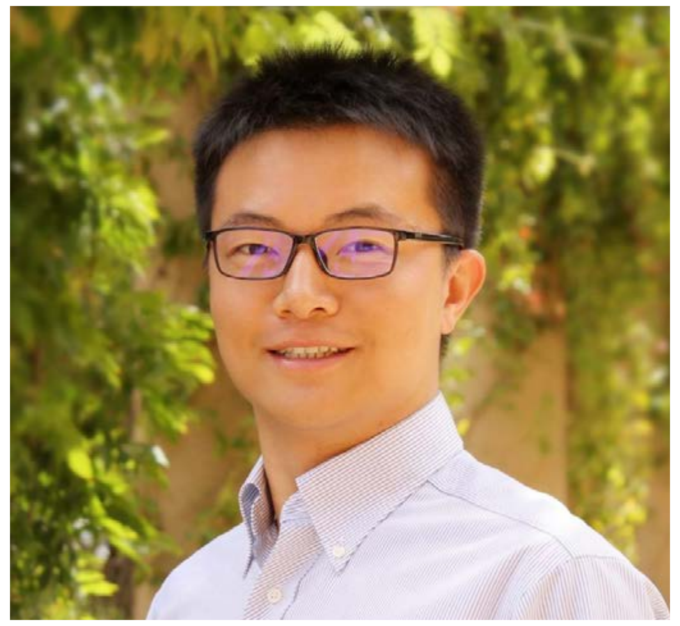
EfficientML.ai Lecture 02: Basics of Neural Networks



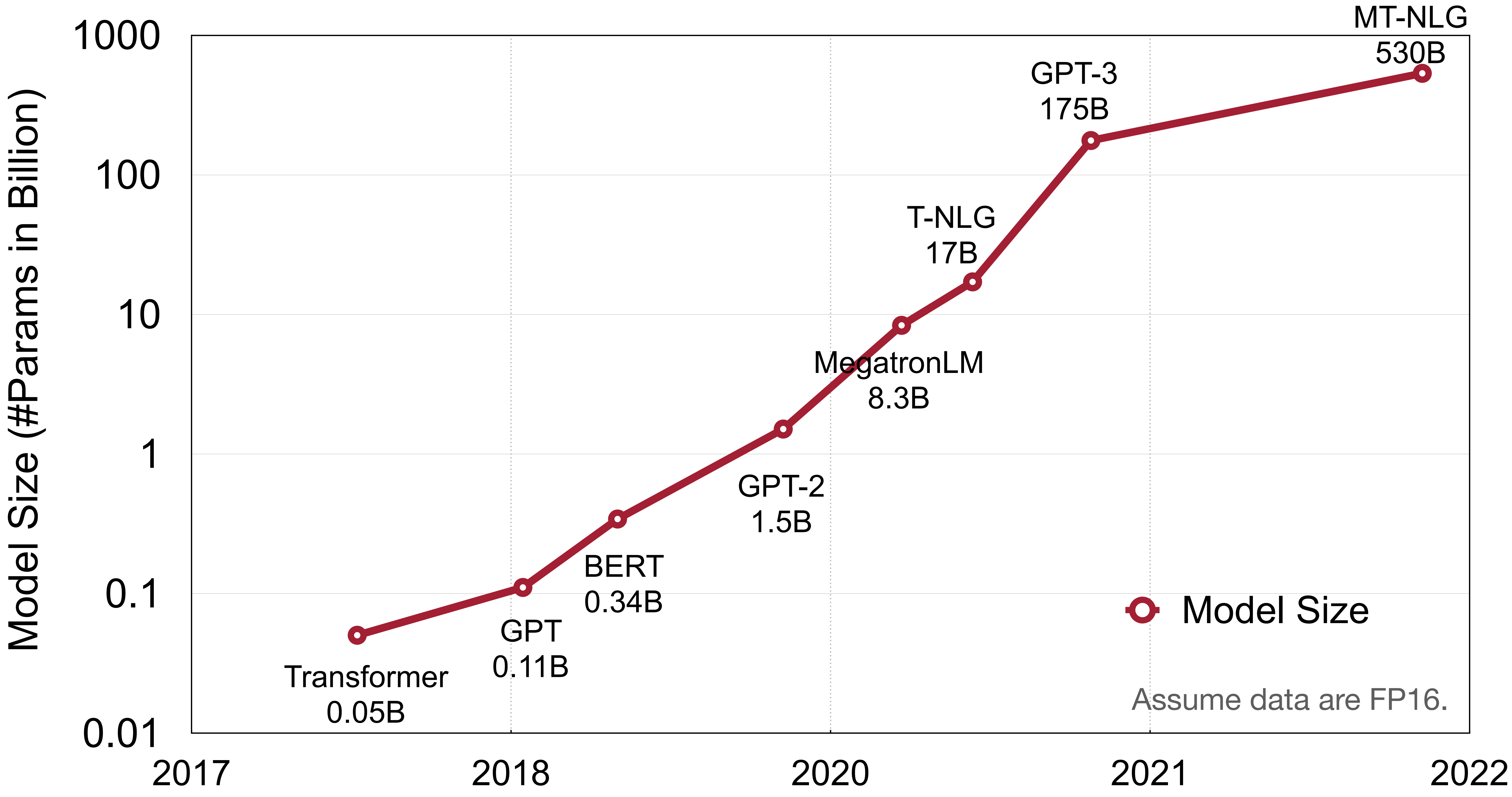
Song Han

Associate Professor, MIT
Distinguished Scientist, NVIDIA

@SongHan_MIT

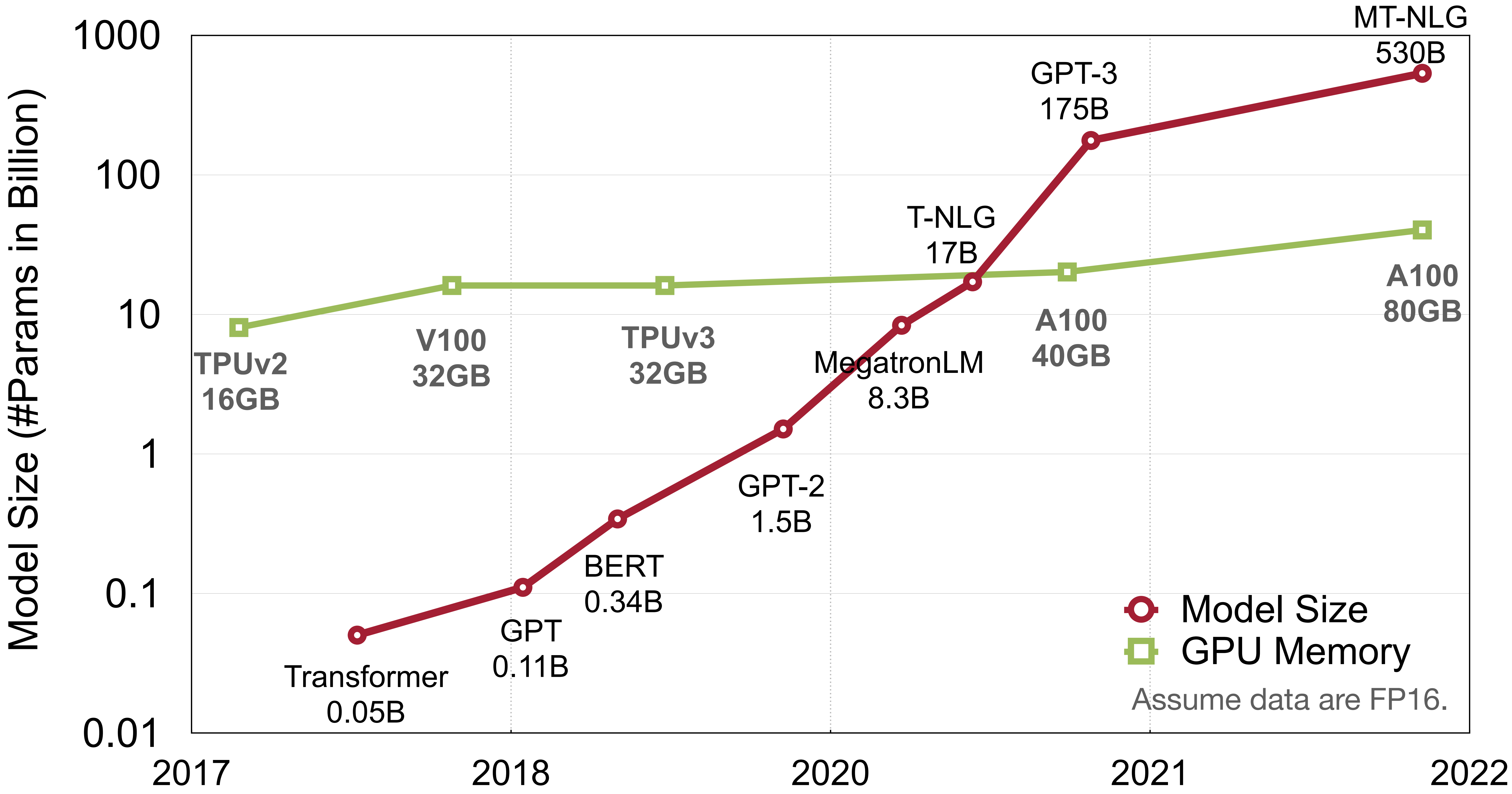
Deep Learning Continues to Scale

The demand of computation grows exponentially



Problem: DL Models Outgrow Hardware

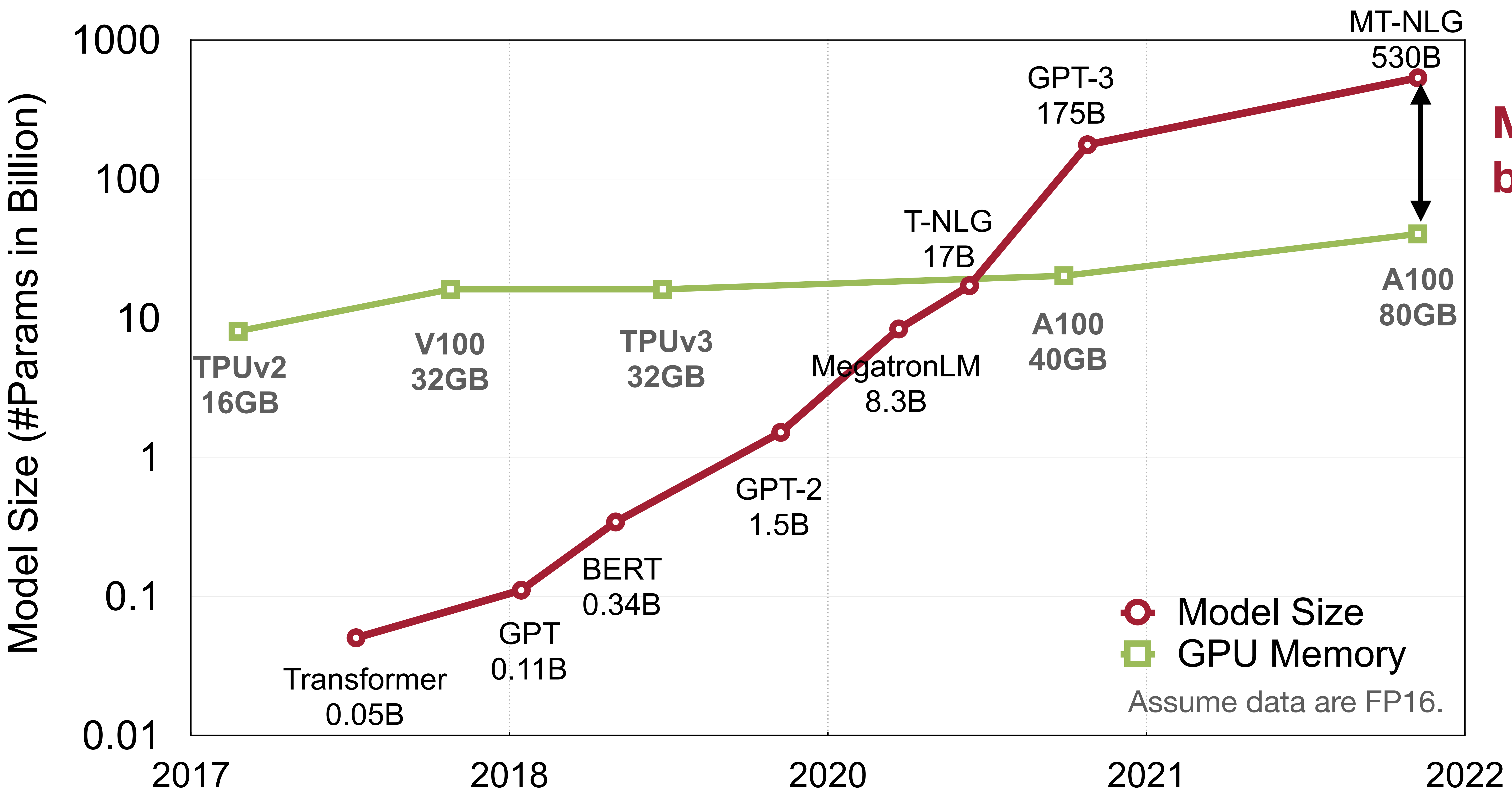
Moore's Law: 2x every 2 years; DL models: 4x every 2 years



Assume data are FP16.

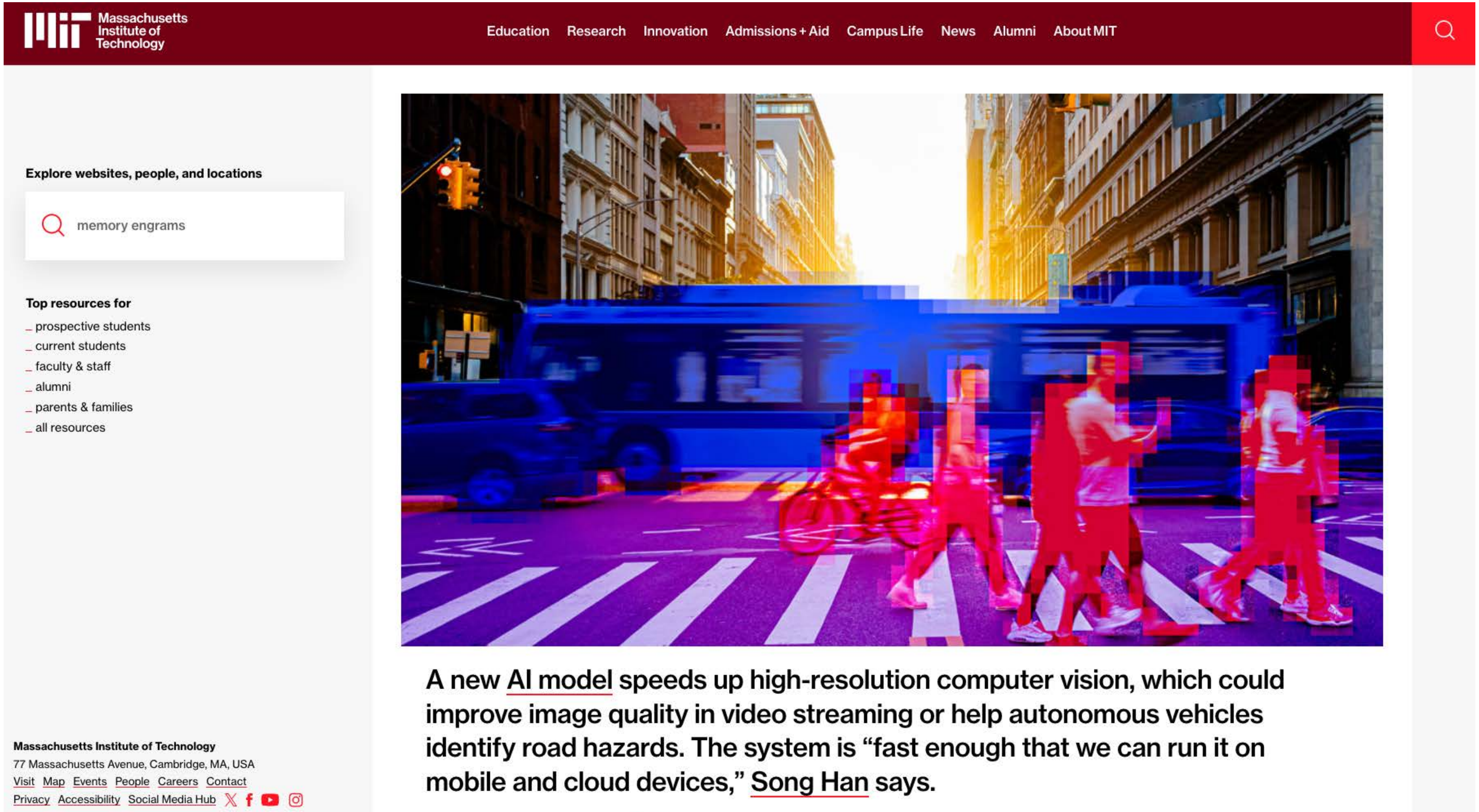
Efficient Deep Learning Techniques are Essential

Bridges the Gap between the Supply and Demand of Computation



Model compression bridges the gap.

EfficientViT: Speeds up High-Resolution Computer Vision



Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA, USA

A new AI model speeds up high-resolution computer vision, which could improve image quality in video streaming or help autonomous vehicles identify road hazards. The system is "fast enough that we can run it on mobile and cloud devices," Song Han says.

EfficientViT: Multi-Scale Linear Attention for High-Resolution Dense Prediction [Cai *et al.*, ICCV 2023]

EfficientViT: Speeds up High-Resolution Computer Vision

EfficientViT enables real-time street scene segmentation on edge

SegFormer
1.6FPS, 82.4mIoU

EfficientViT
21.8FPS, 82.7mIoU

Speed is measured on Nvidia Jetson AGX Orin with TensorRT, fp16, batch size 1.
Performance is measured on the Cityscapes dataset.

EfficientViT: Multi-Scale Linear Attention for High-Resolution Dense Prediction [Cai *et al.*, ICCV 2023]

Efficient Prompt Segmentation

EfficientViT accelerates Segment Anything by 70 times on GPU

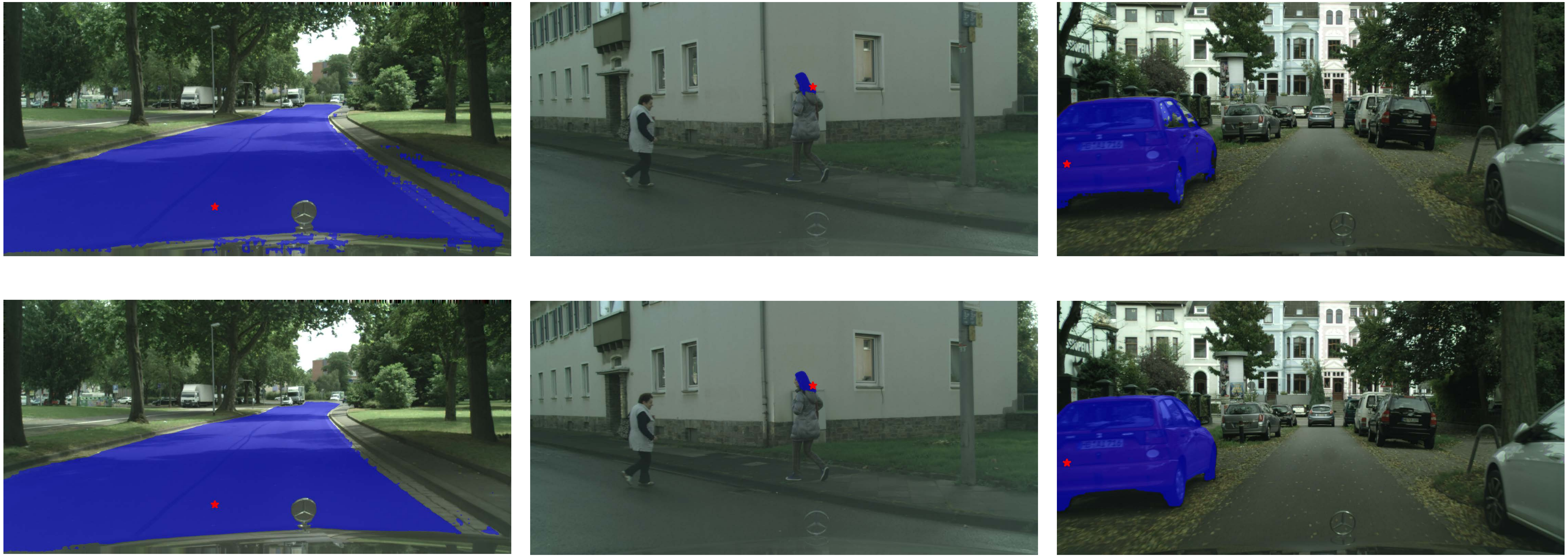
SAM ViT-Huge
12 image/s

EfficientViT
842 image/s

EfficientViT: Multi-Scale Linear Attention for High-Resolution Dense Prediction [Cai *et al.*, ICCV 2023]

Efficient Prompt Segmentation

EfficientViT accelerates Segment Anything by 70 times on GPU



SAM ViT-Huge
12 image/s

EfficientViT
842 image/s

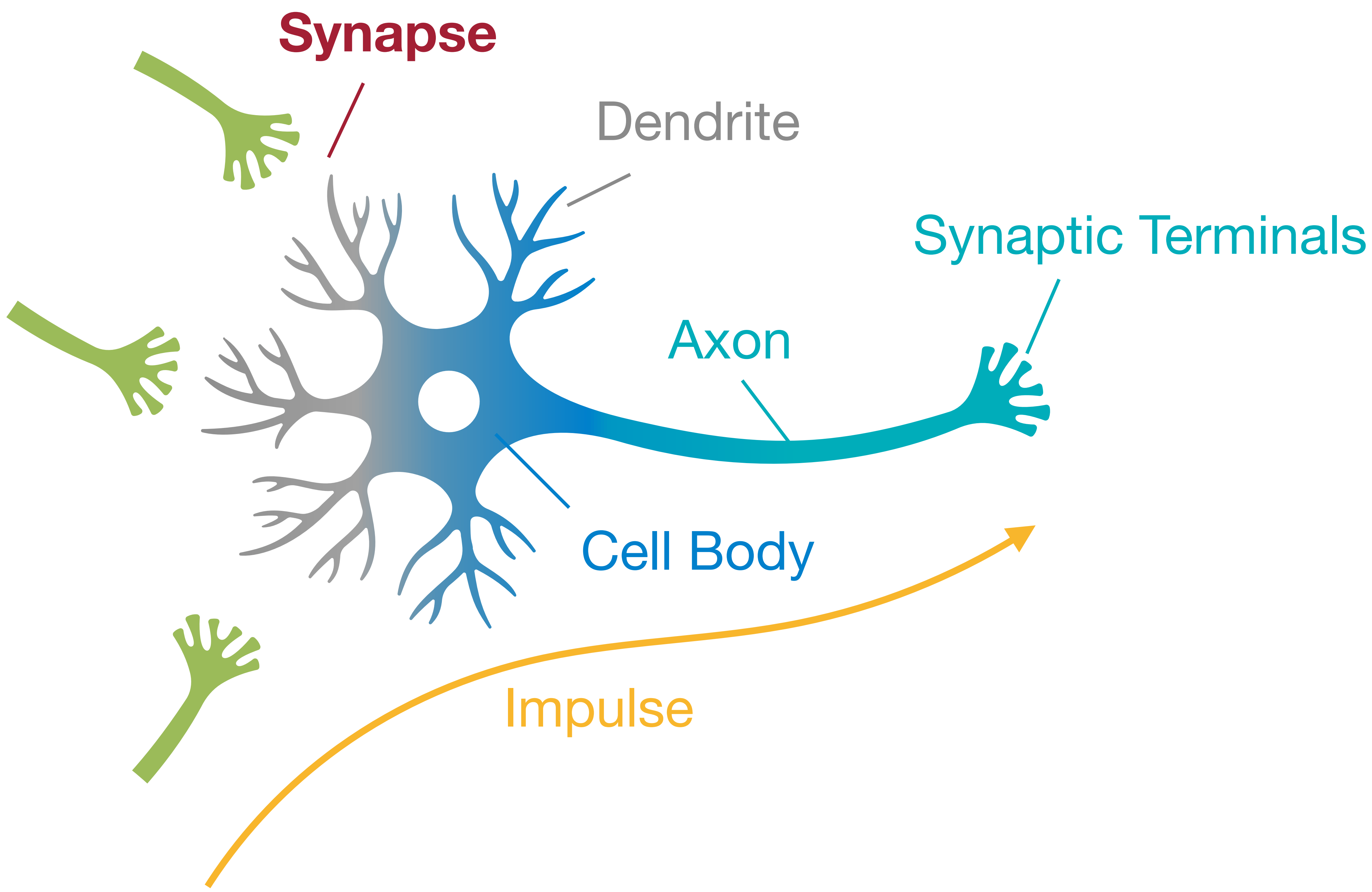
EfficientViT: Multi-Scale Linear Attention for High-Resolution Dense Prediction [Cai *et al.*, ICCV 2023]

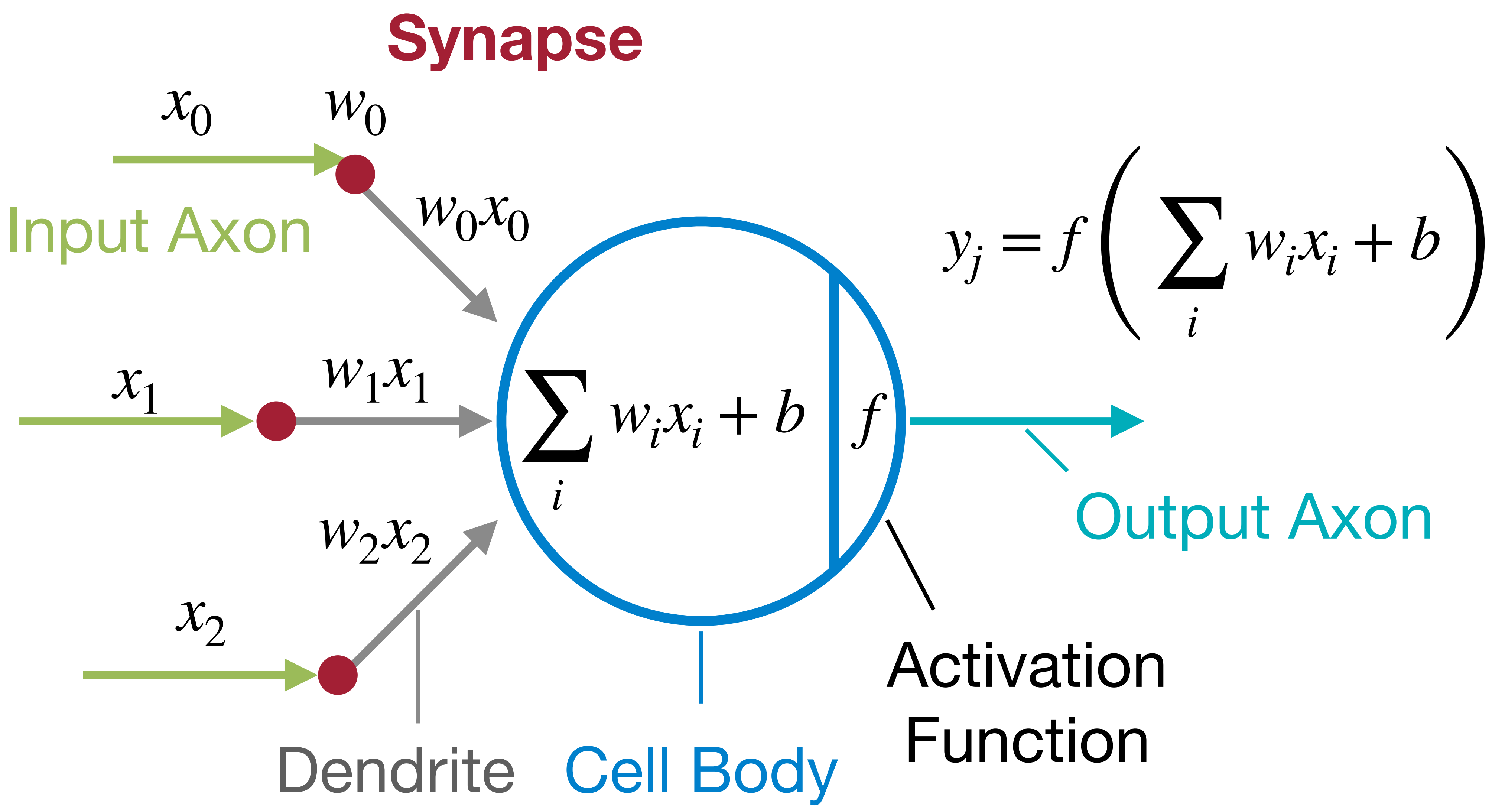
Lecture Plan

Today we will:

1. Review the **terminology of neural networks**
    - Neuron, Synapses, Activation, Feature, Weight, Parameter, *etc*
2. Review **popular building blocks** in a neural network
    - Fully-Connected, Convolution, Grouped Convolution, Depthwise Convolution
    - Pooling, Normalization, Transformer
3. Review **convolutional neural networks**’ architecture
    - AlexNet, VGG-16, ResNet-50, MobileNetV2
4. Introduce **popular efficiency metrics** for neural networks
    - #Parameters, Model Size, Peak #Activations, MAC, FLOP, FLOPS, OP, OPS, Latency, Throughput
5. Lab 0: Tutorial on PyTorch and lab exercises

Neuron and Synapse

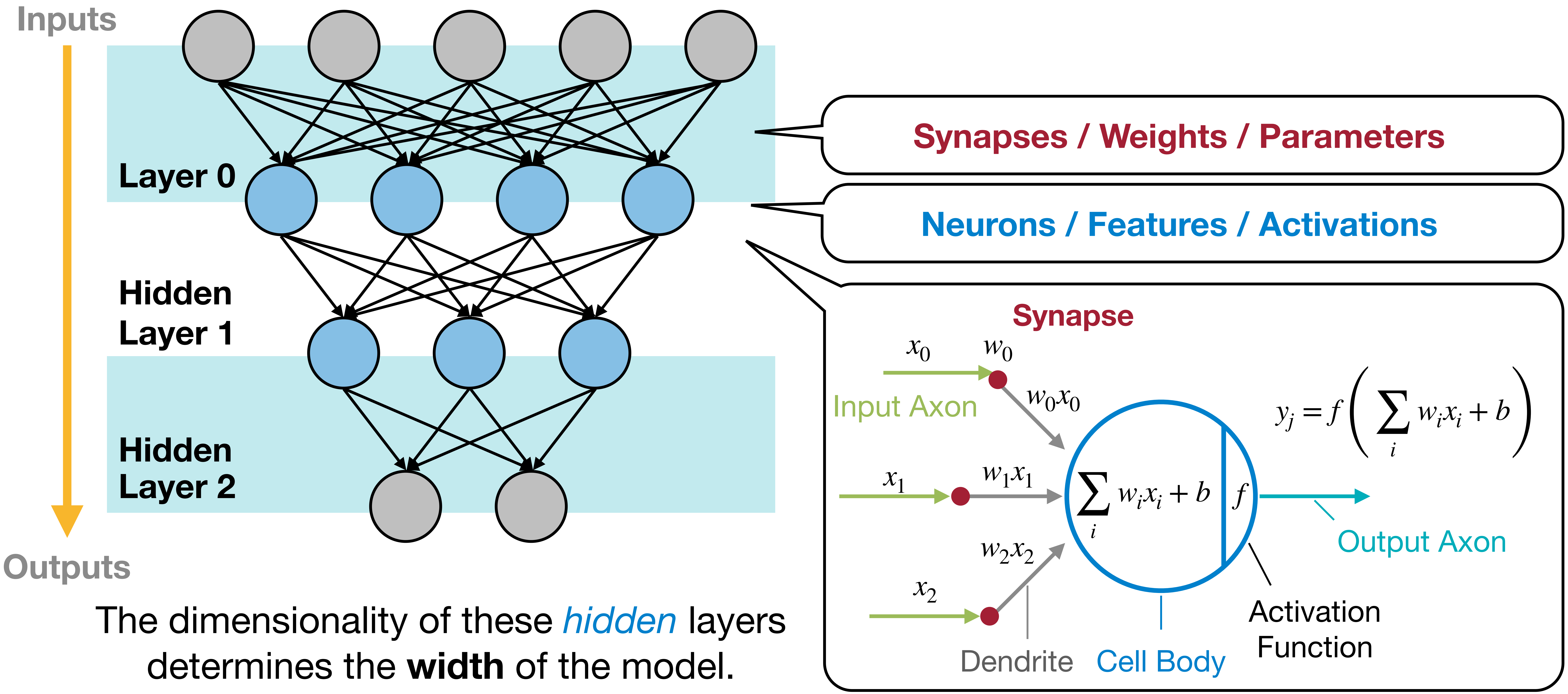




$$y_j = f\left(\sum_i w_i x_i + b\right)$$

Deep Neural Network

3-Layer Neural Network
With 2 Hidden Layers



Synapses / Weights / Parameters

Neurons / Features / Activations

$$y_j = f\left(\sum_i w_ix_i + b\right)$$

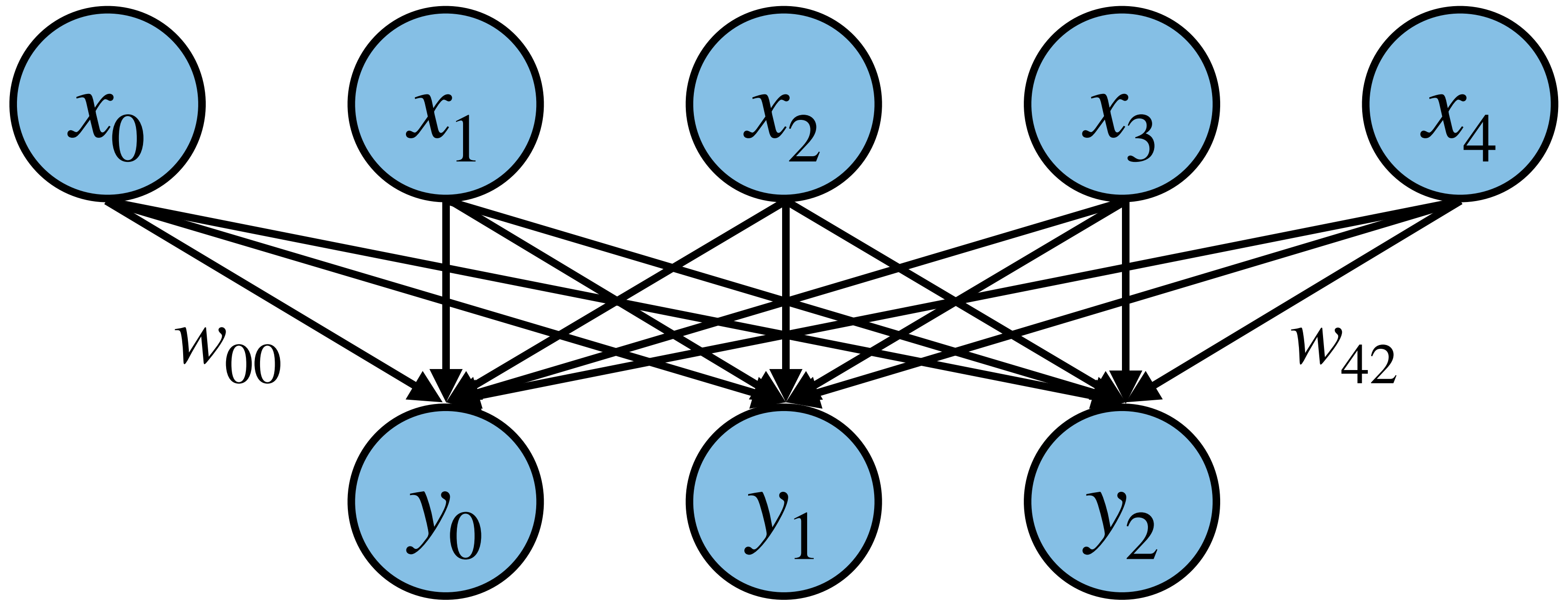
The dimensionality of these *hidden* layers determines the **width** of the model.

Popular Neural Network Layers

Fully-Connected Layer (Linear Layer)

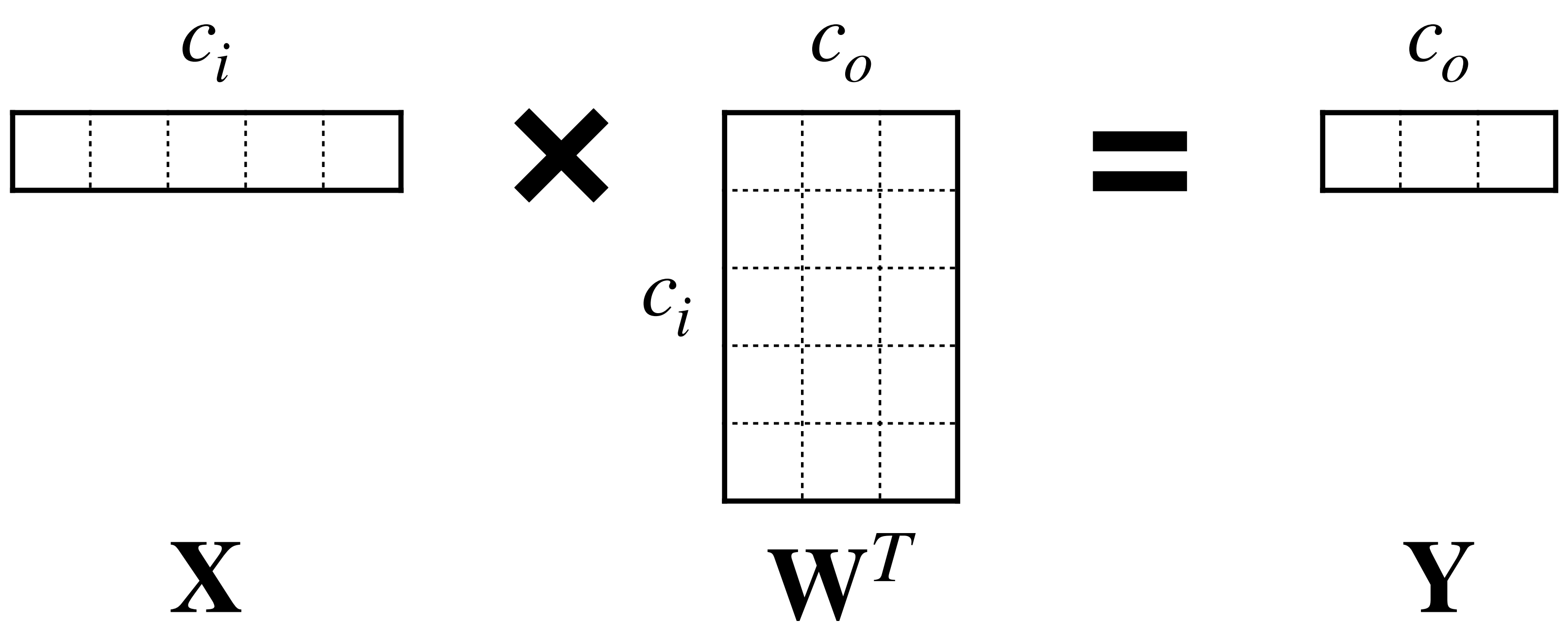
The output neuron is connected to all input neurons.

- **Shape of Tensors:**
  - Input Features $\mathbf{X}$: $(1, c_i)$
  - Output Features $\mathbf{Y}$: $(1, c_o)$
  - Weights $\mathbf{W}$: (c_o, c_i)
  - Bias $\mathbf{b}$: $(c_o,)$



$$y_i = \sum_j w_{ij} x_j + b_i$$

Notations	
c_i	Input Channels
c_o	Output Channels

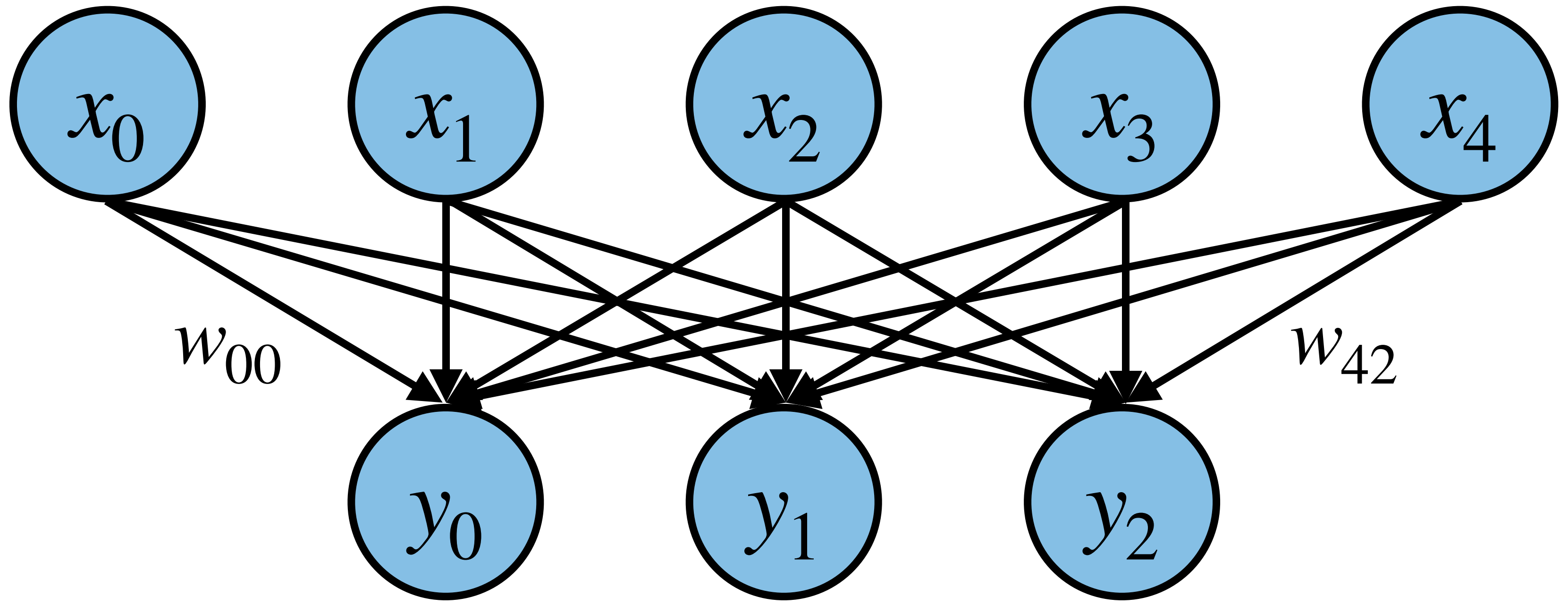


Fully-Connected Layer (Linear Layer)

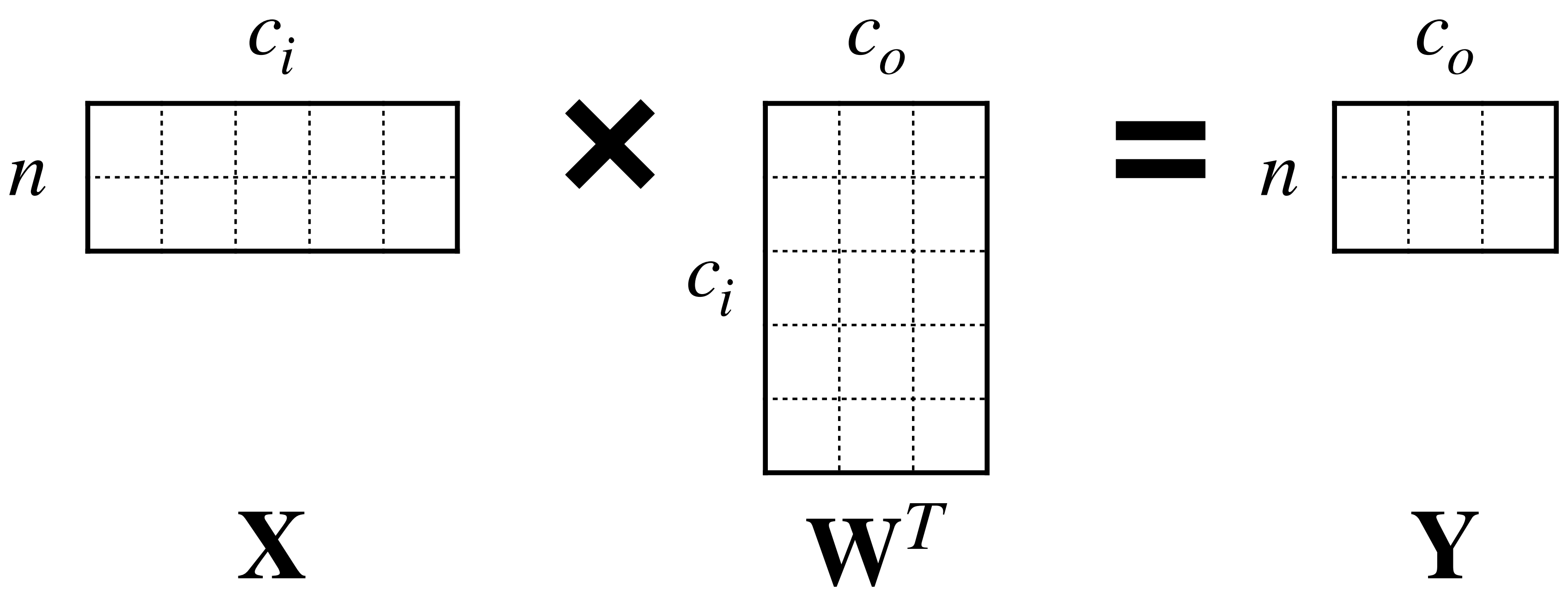
The output neuron is connected to all input neurons.

- **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



$$y_i = \sum_j w_{ij} x_j + b_i$$

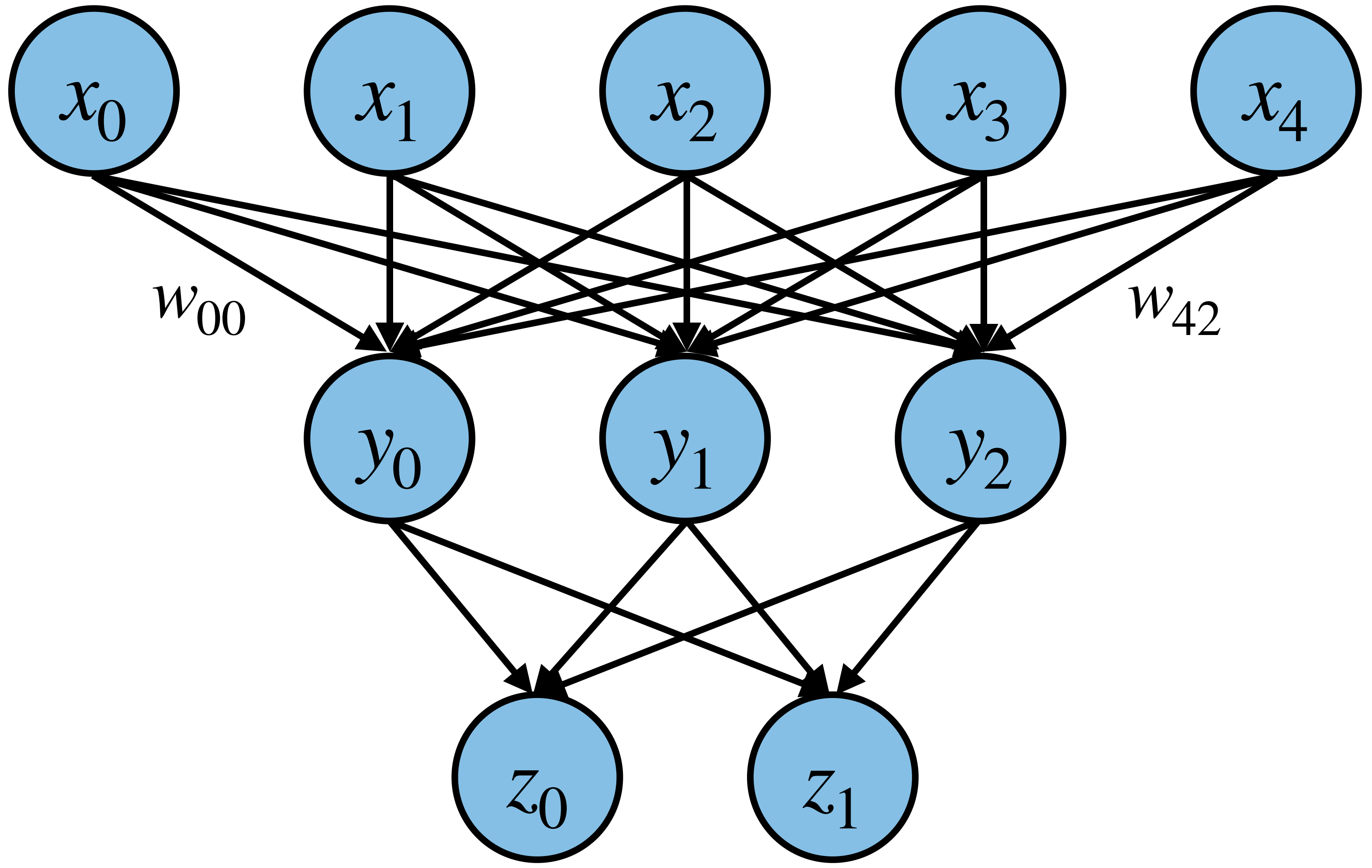


Fully-Connected Layer (Linear Layer)

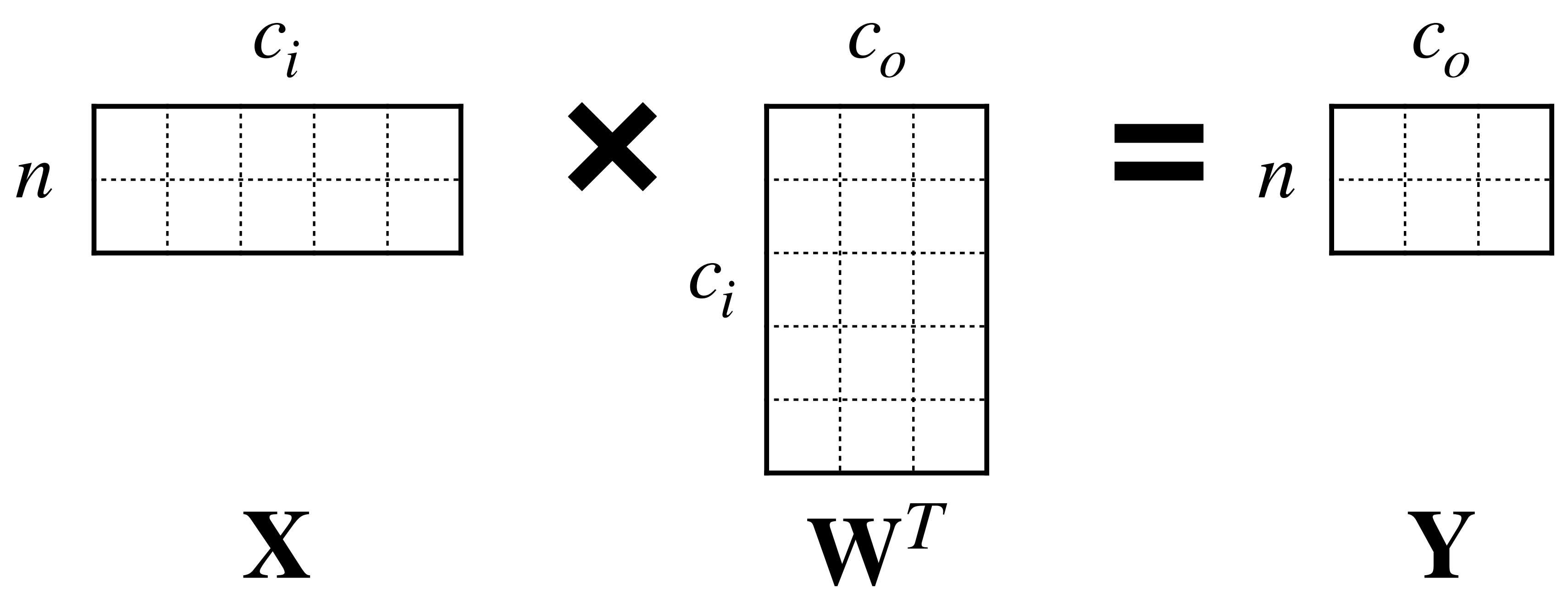
The output neuron is connected to all input neurons.

- **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)

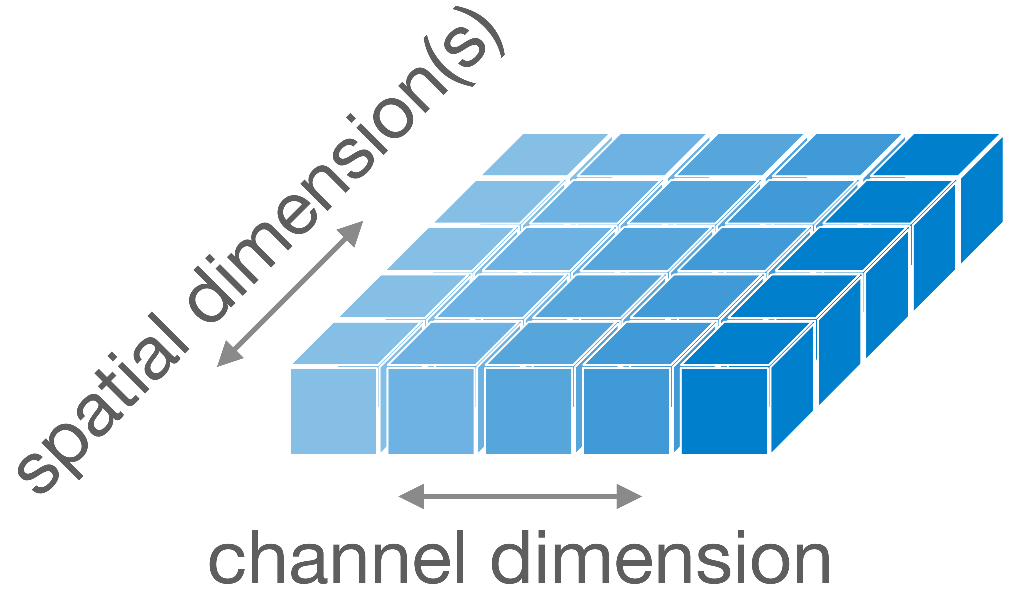


Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:** **1D Conv**
  - Input Features $\mathbf{X}$: ~~(n, c_i)~~ (n, c_i, w_i)
  - Output Features $\mathbf{Y}$: ~~(n, c_o)~~ (n, c_o, w_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
w_i, w_o	Input/Output Width

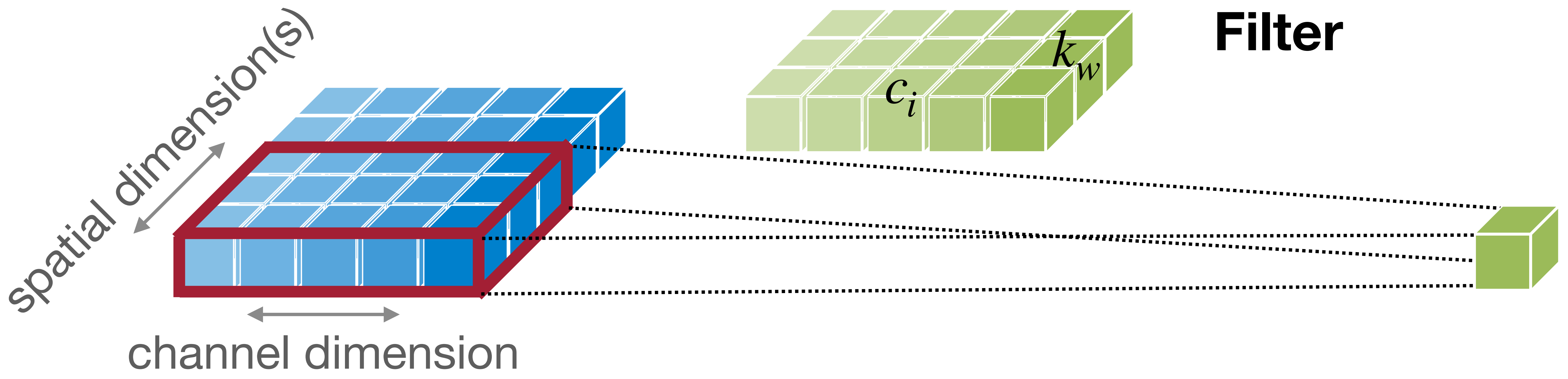


Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:** **1D Conv**
  - Input Features $\mathbf{X}$: ~~(n, c_i)~~ (n, c_i, w_i)
  - Output Features $\mathbf{Y}$: ~~(n, c_o)~~ (n, c_o, w_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
w_i, w_o	Input/Output Width
k_w	Kernel Width

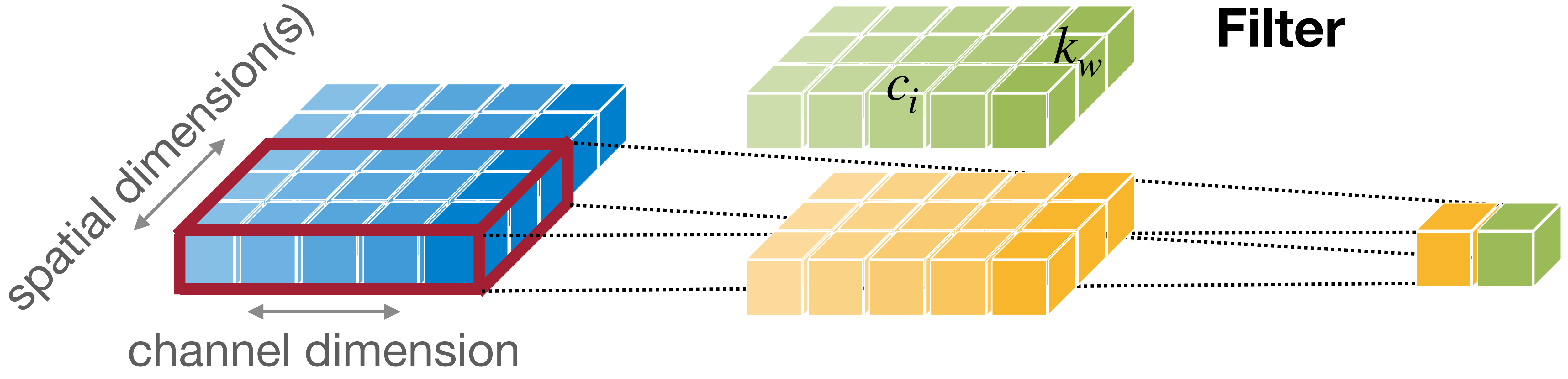


Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:** **1D Conv**
  - Input Features $\mathbf{X}$: $\cancel{(n, c_i)}$ (n, c_i, w_i)
  - Output Features $\mathbf{Y}$: $\cancel{(n, c_o)}$ (n, c_o, w_o)
  - Weights $\mathbf{W}$: $\cancel{(c_o, c_i)}$
  - Bias $\mathbf{b}$: $(c_o,)$

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

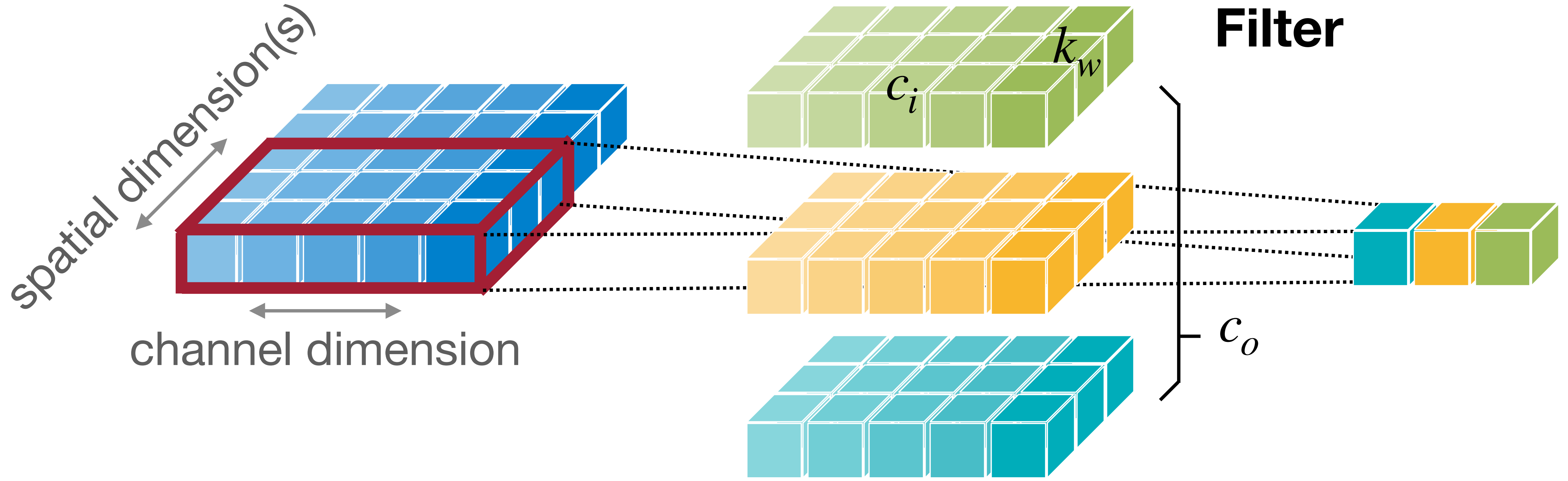


Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:** **1D Conv**
  - Input Features $\mathbf{X}$: ~~(n, c_i)~~ (n, c_i, w_i)
  - Output Features $\mathbf{Y}$: ~~(n, c_o)~~ (n, c_o, w_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
w_i, w_o	Input/Output Width
k_w	Kernel Width

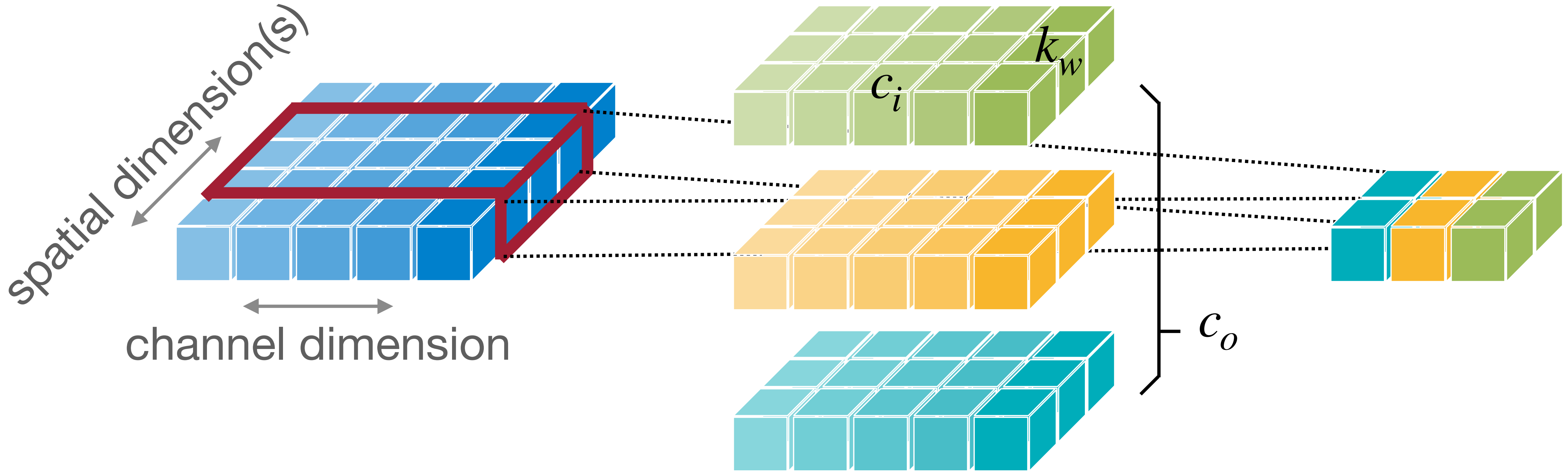


Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:** **1D Conv**
  - Input Features $\mathbf{X}$: ~~(n, c_i)~~ (n, c_i, w_i)
  - Output Features $\mathbf{Y}$: ~~(n, c_o)~~ (n, c_o, w_o)
  - Weights $\mathbf{W}$: ~~(c_o, c_i)~~ (c_o, c_i, k_w)
  - Bias $\mathbf{b}$: $(c_o,)$

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



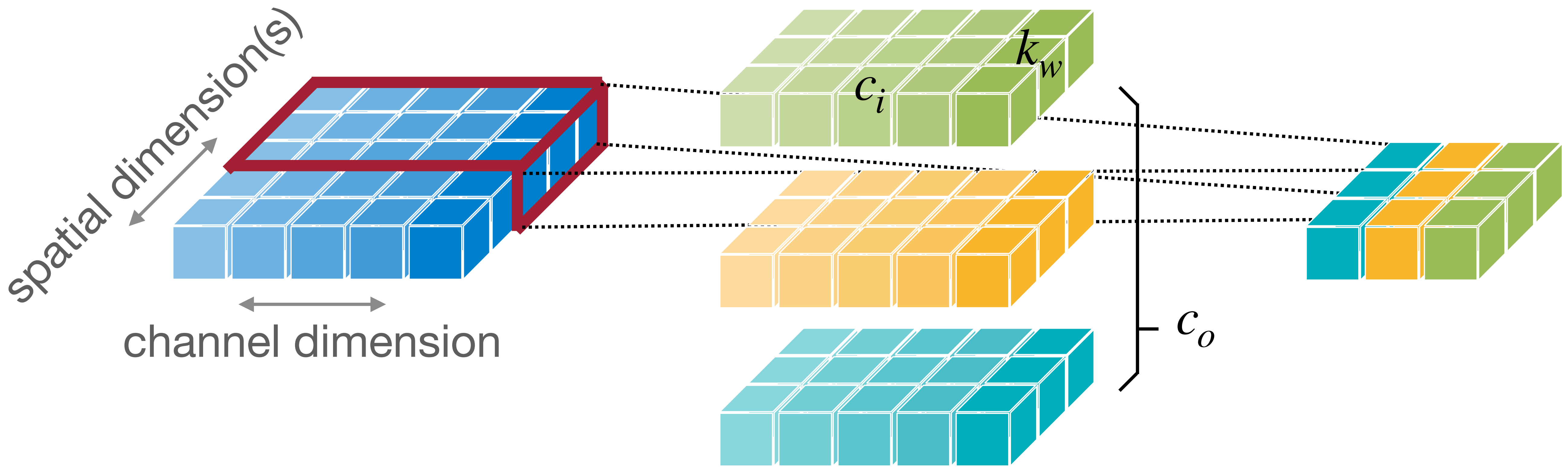
Weight Sharing

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:** **1D Conv**
  - Input Features $\mathbf{X}$: ~~(n, c_i)~~ (n, c_i, w_i)
  - Output Features $\mathbf{Y}$: ~~(n, c_o)~~ (n, c_o, w_o)
  - Weights $\mathbf{W}$: ~~(c_o, c_i)~~ (c_o, c_i, k_w)
  - Bias $\mathbf{b}$: $(c_o,)$

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



Weight Sharing

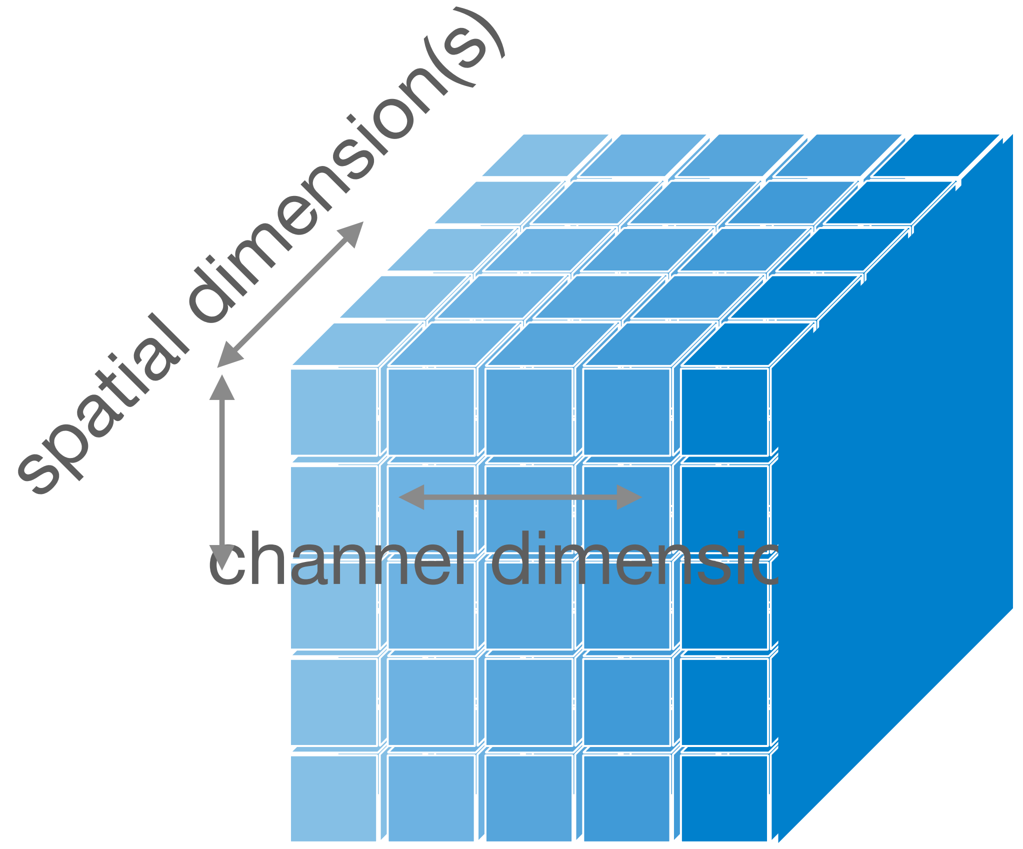
Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:**

	1D Conv	2D Conv
Input Features $\mathbf{X}$: $\cancel{(n, c_i)}$	(n, c_i, w_i)	(n, c_i, h_i, w_i)
Output Features $\mathbf{Y}$: $\cancel{(n, c_o)}$	(n, c_o, w_o)	(n, c_o, h_o, w_o)
Weights $\mathbf{W}$: $\cancel{(c_o, c_i)}$	(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



Activation Map / Feature Map

$h \times w$

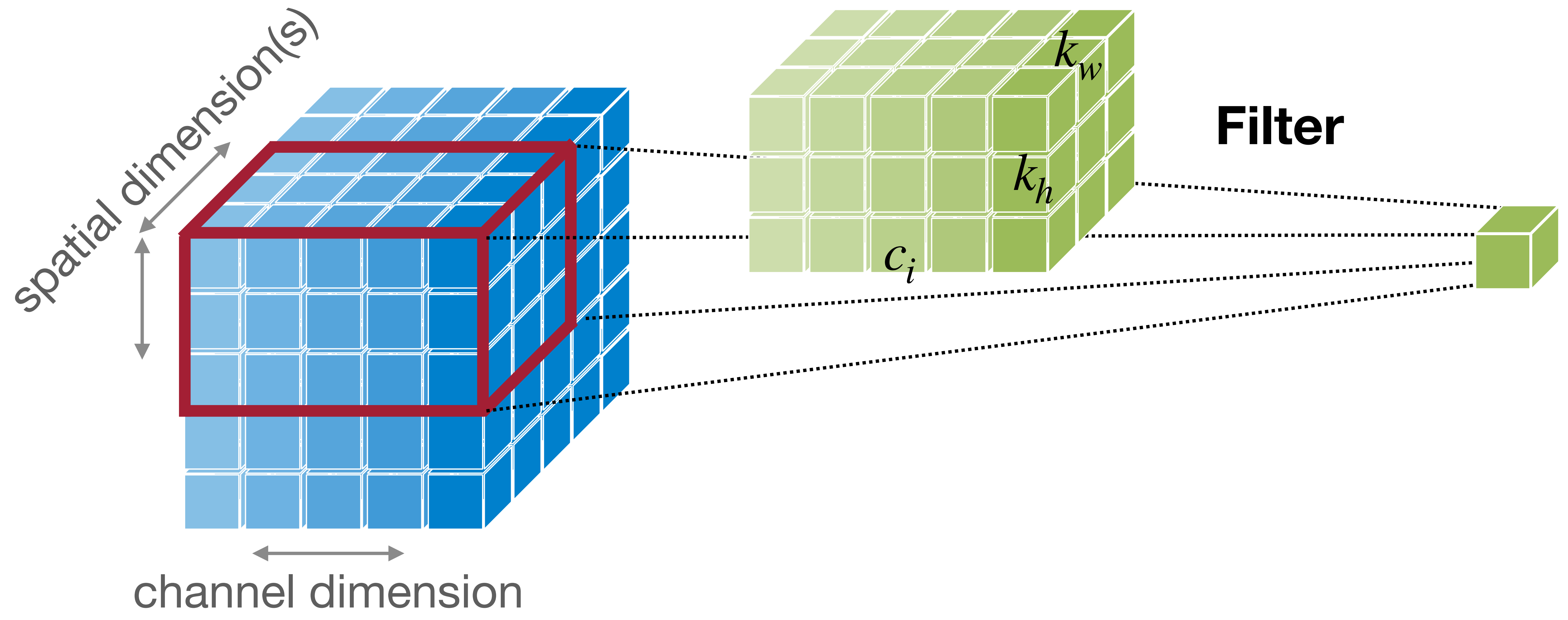
Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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

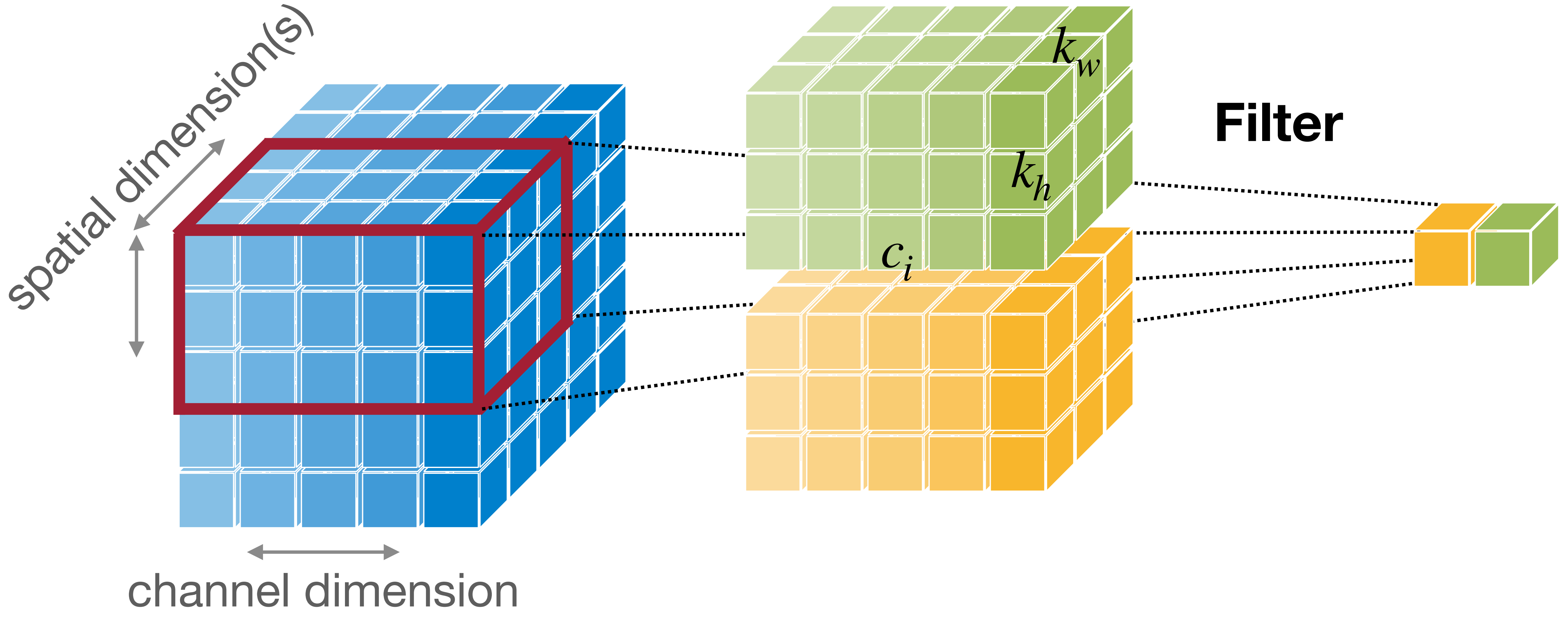


Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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



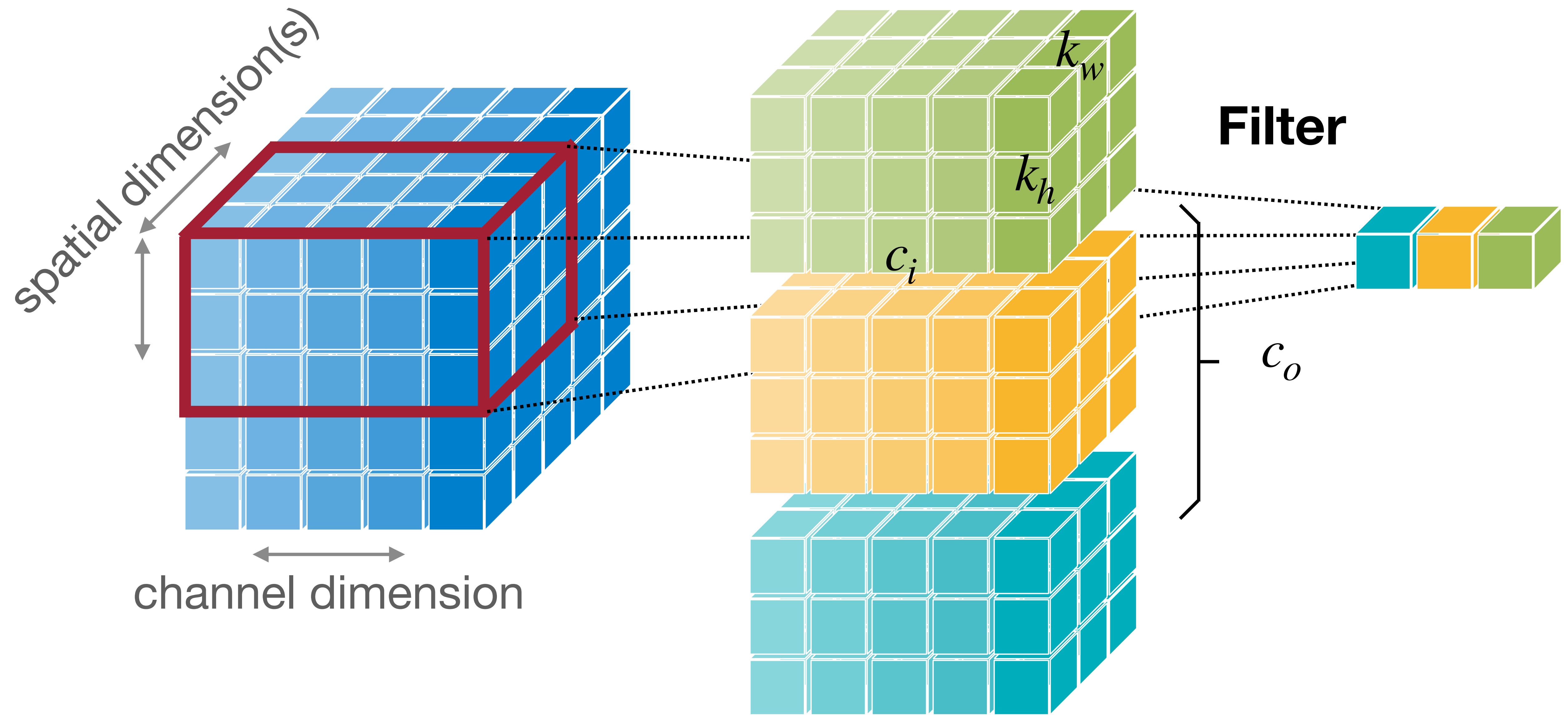
Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:**

	1D Conv	**2D Conv**
Input Features $\mathbf{X}$: $\cancel{(n, c_i)}$	(n, c_i, w_i)	(n, c_i, h_i, w_i)
Output Features $\mathbf{Y}$: $\cancel{(n, c_o)}$	(n, c_o, w_o)	(n, c_o, h_o, w_o)
Weights $\mathbf{W}$: $\cancel{(c_o, c_i)}$	(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

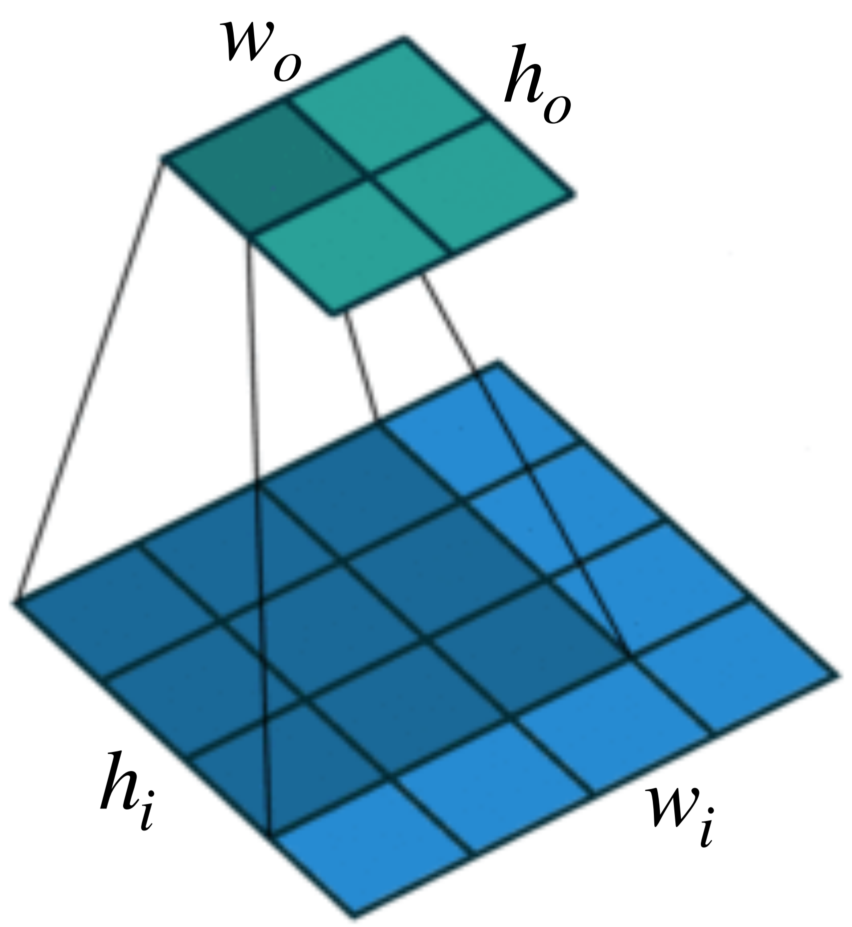


Convolution Layer

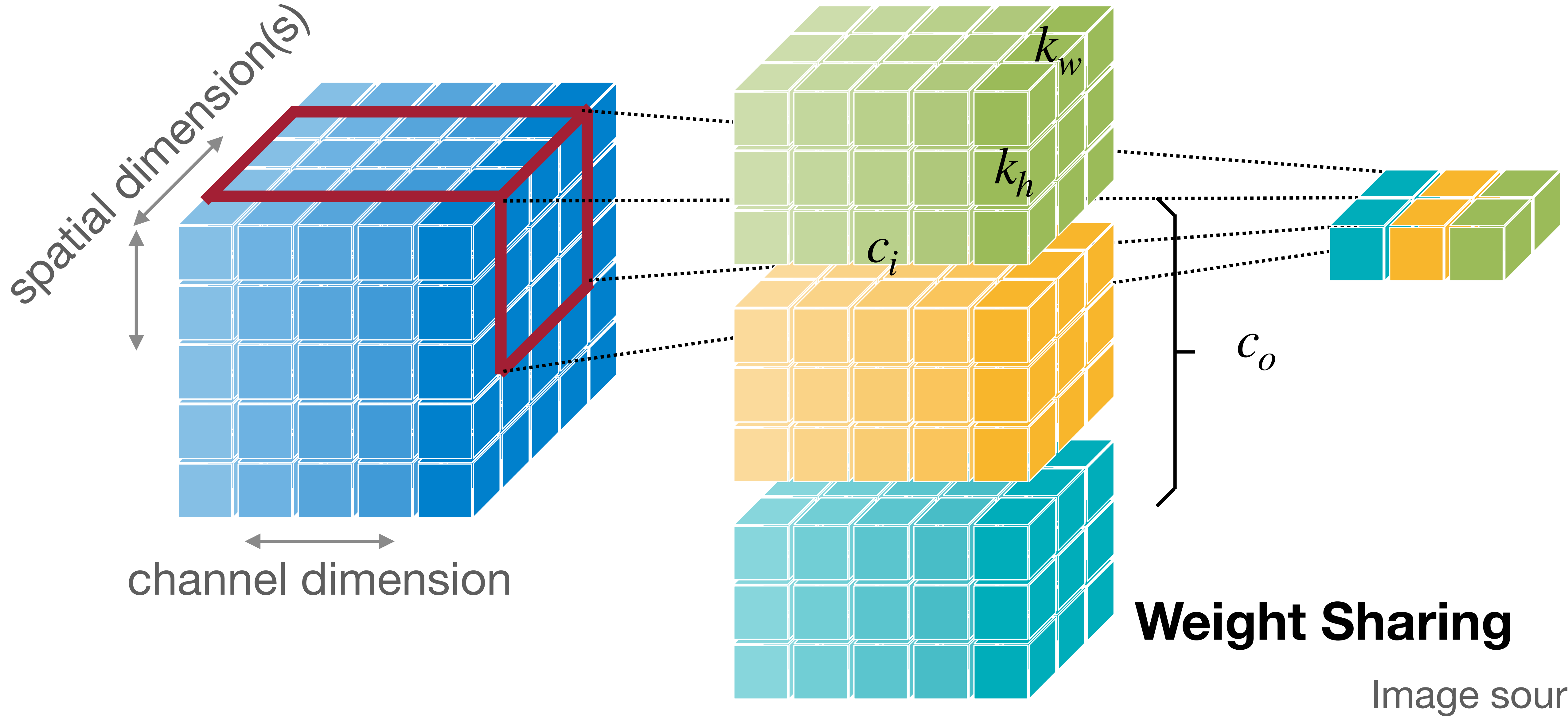
The output neuron is connected to input neurons in the receptive field.

- **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



Weight Sharing

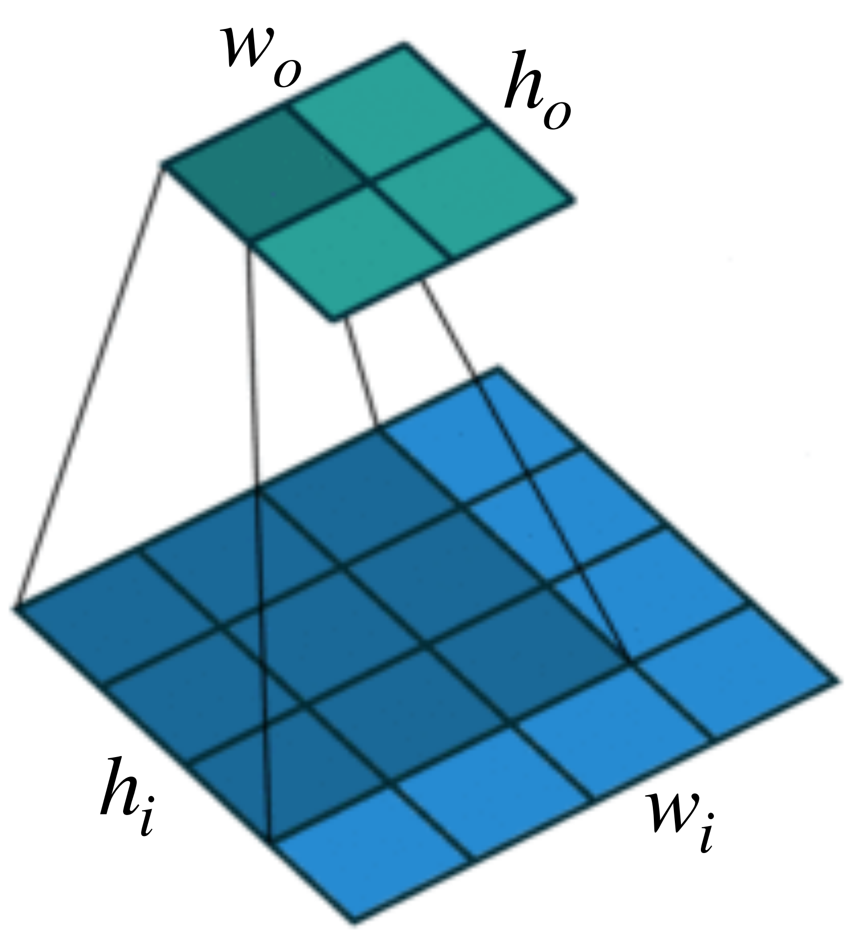
Image source: 1

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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

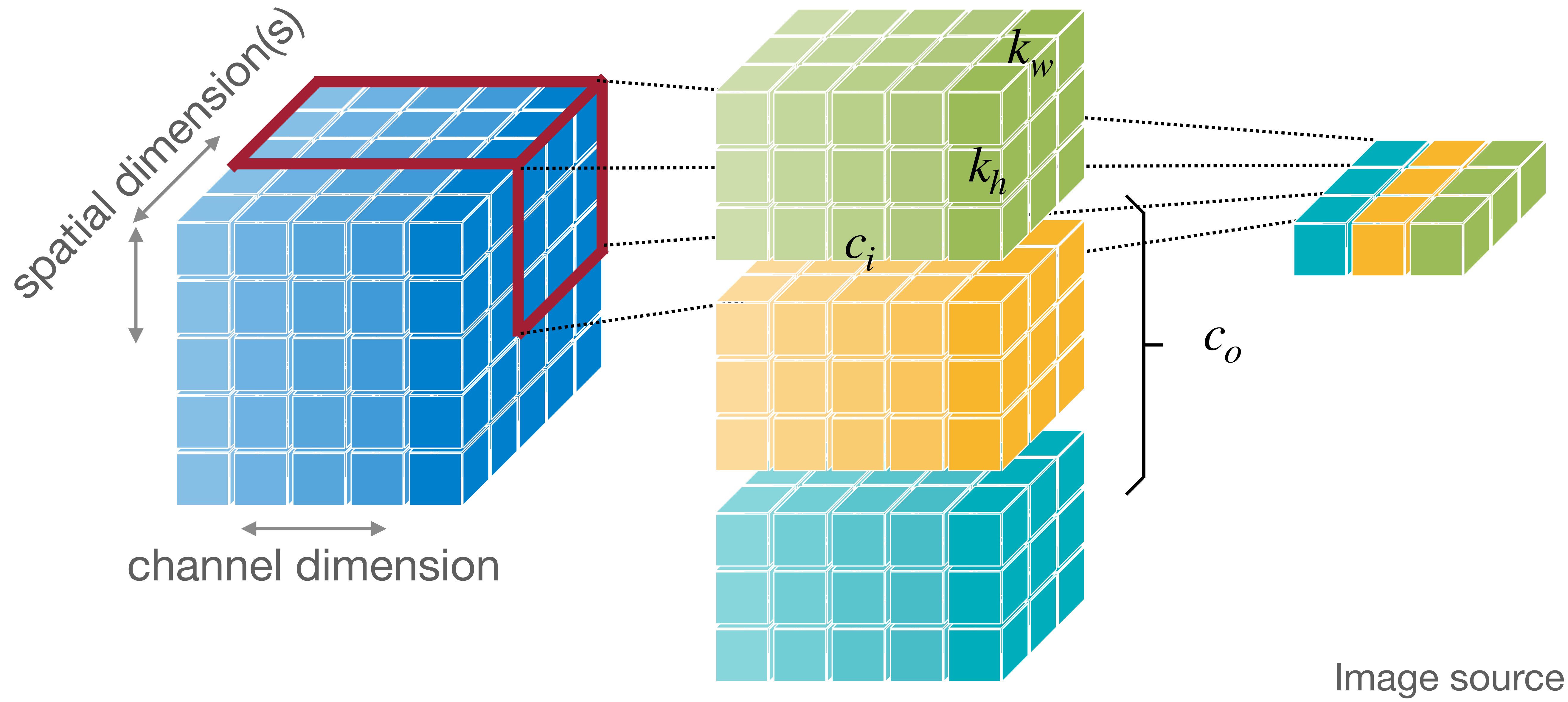


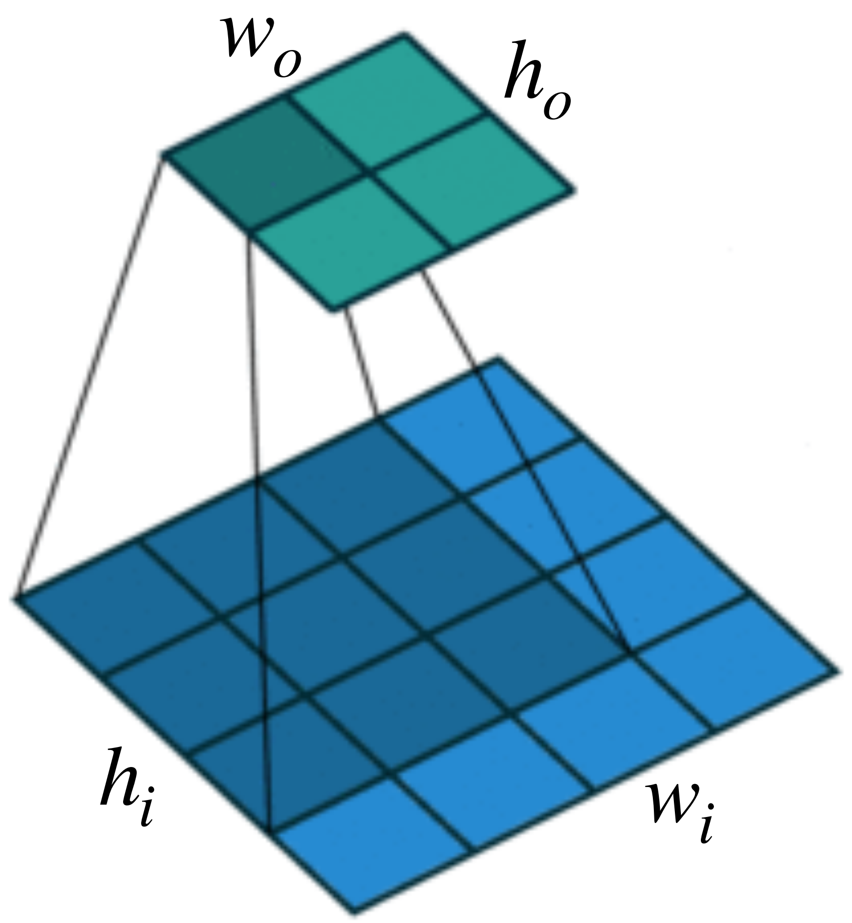
Image source: 1

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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

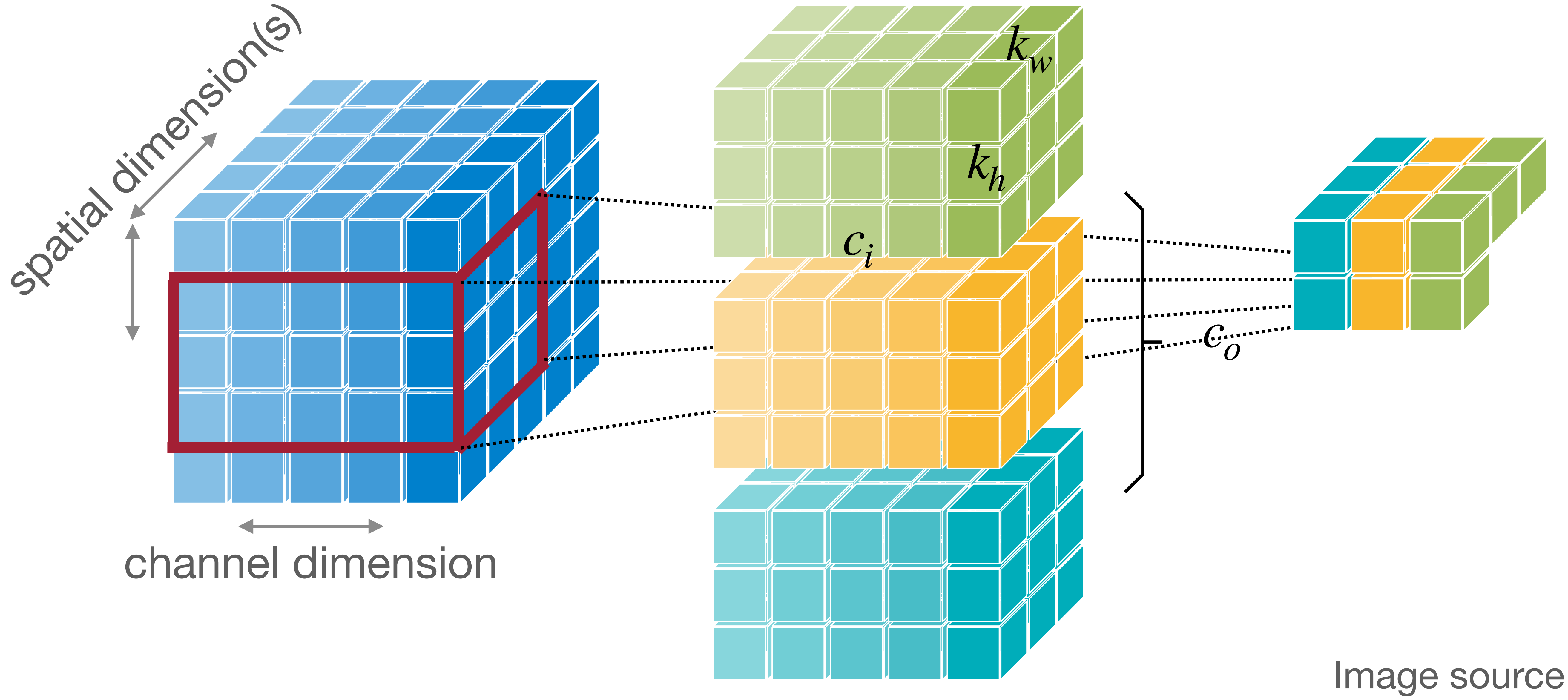


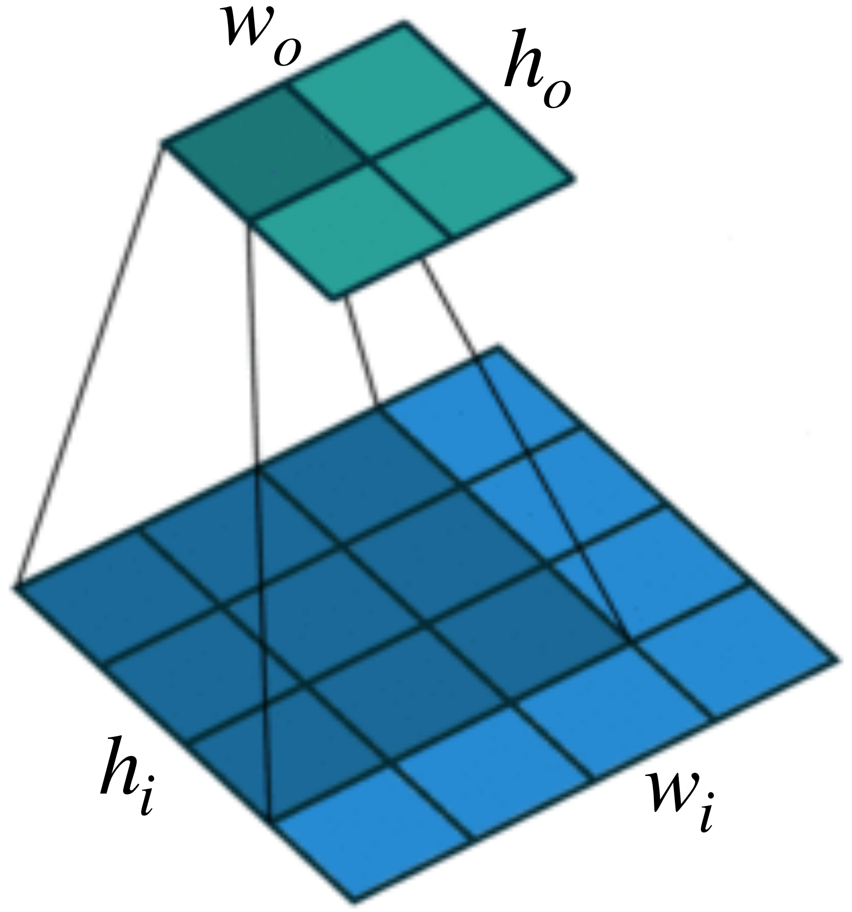
Image source: 1

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **Shape of Tensors:**

		1D Conv	**2D Conv**
Input Features $\mathbf{X}$:	$\cancel{(n, c_i)}$	(n, c_i, w_i)	(n, c_i, h_i, w_i)
Output Features $\mathbf{Y}$:	$\cancel{(n, c_o)}$	(n, c_o, w_o)	(n, c_o, h_o, w_o)
Weights $\mathbf{W}$:	$\cancel{(c_o, c_i)}$	(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

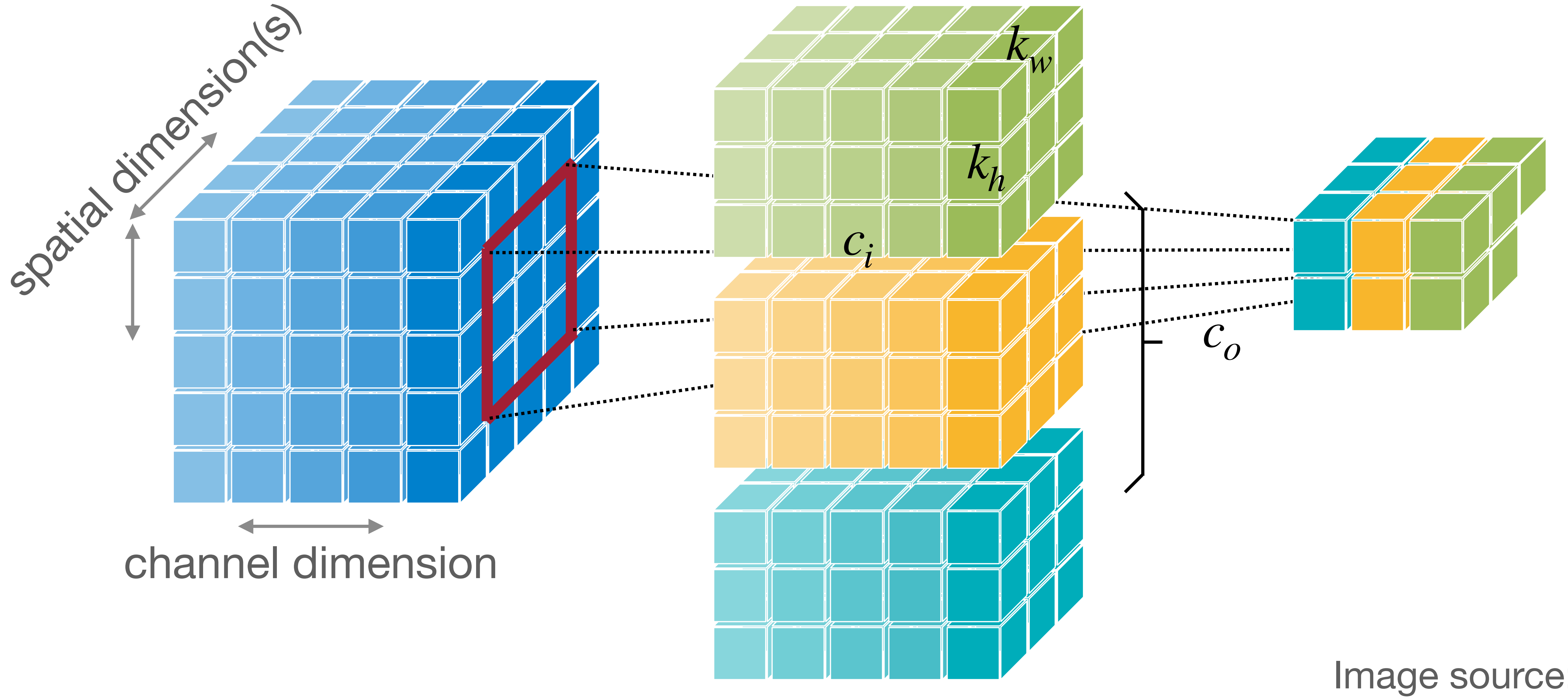


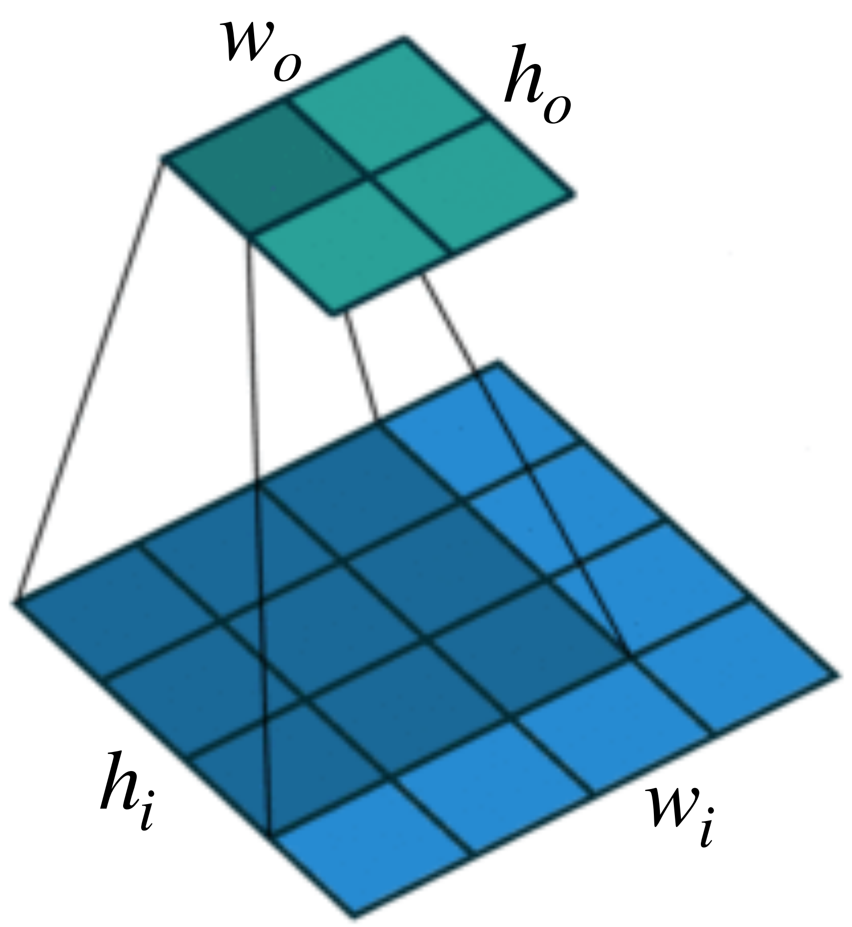
Image source: 1

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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

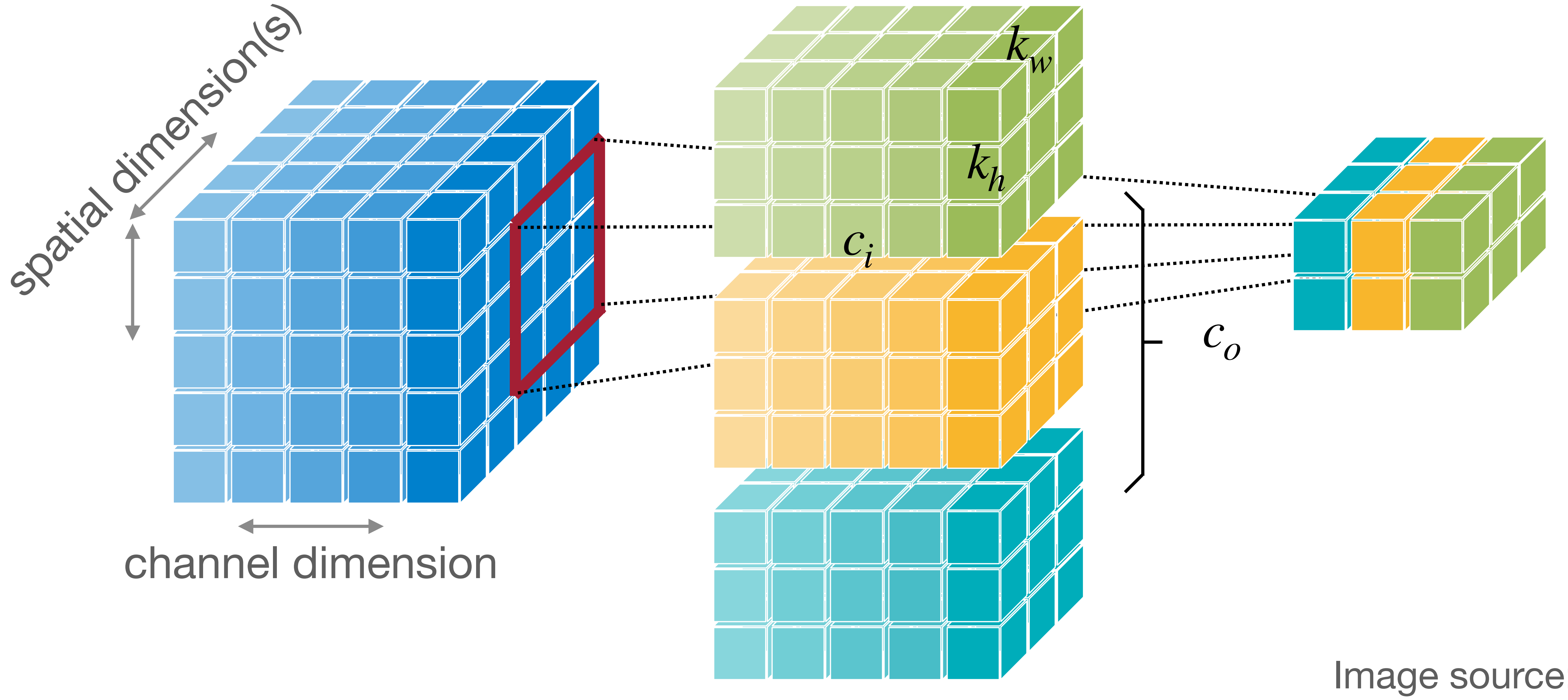


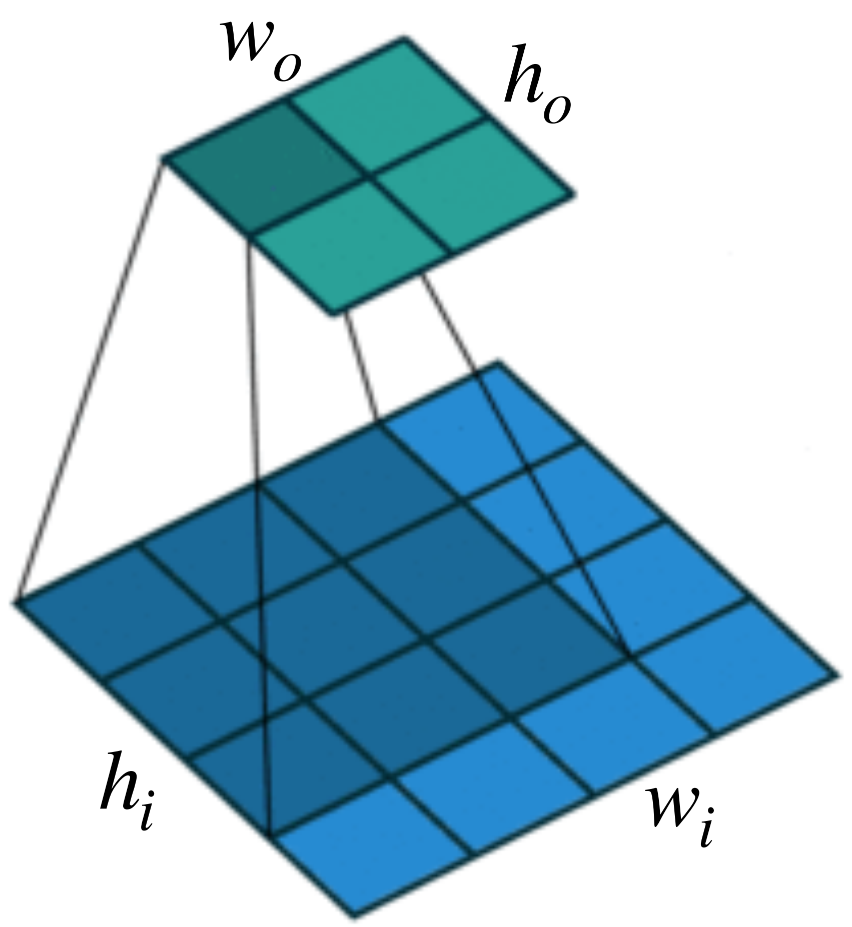
Image source: 1

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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

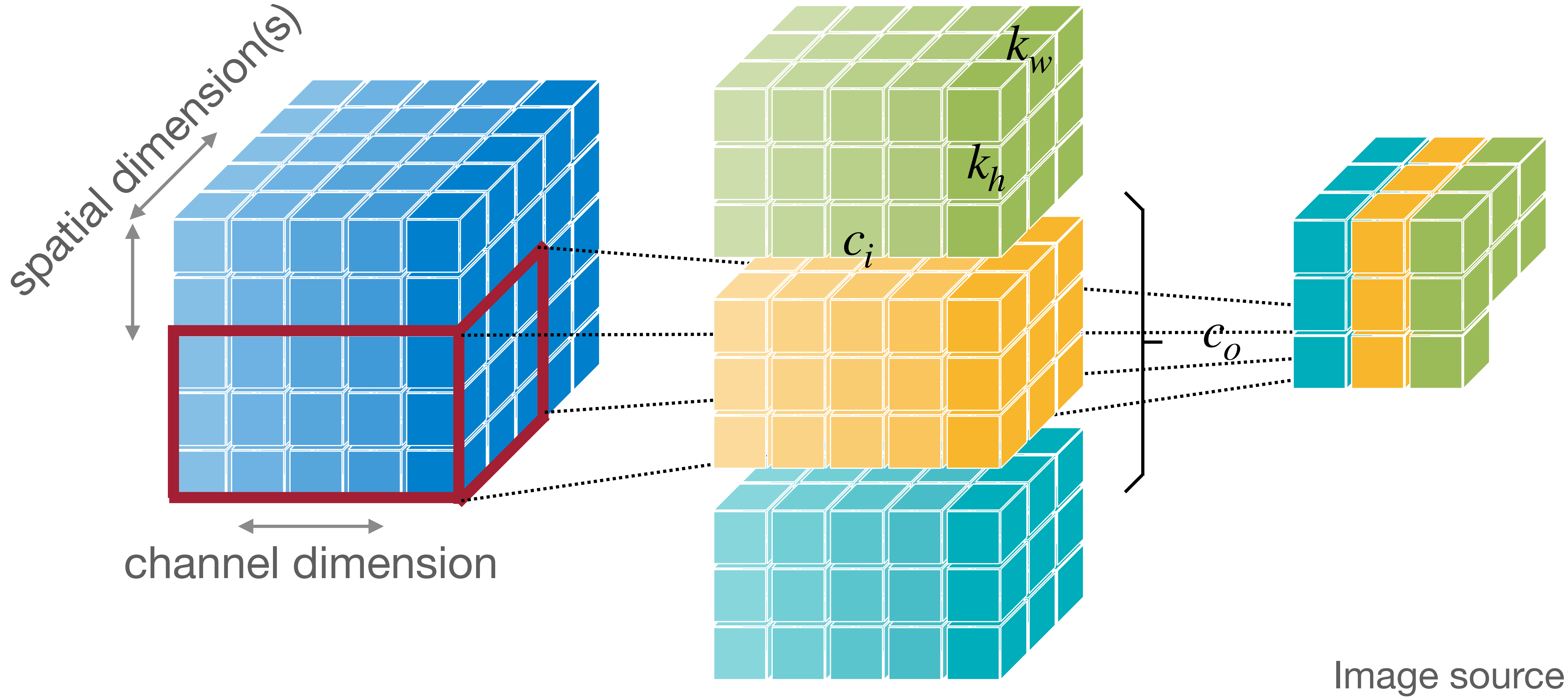


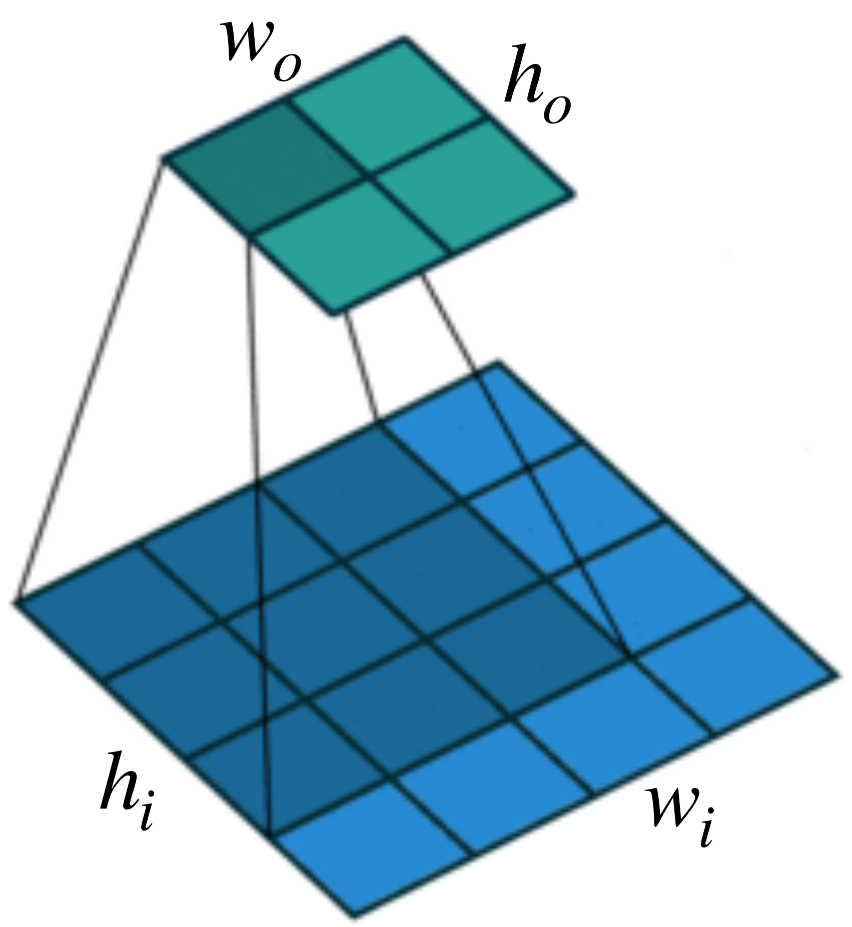
Image source: 1

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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

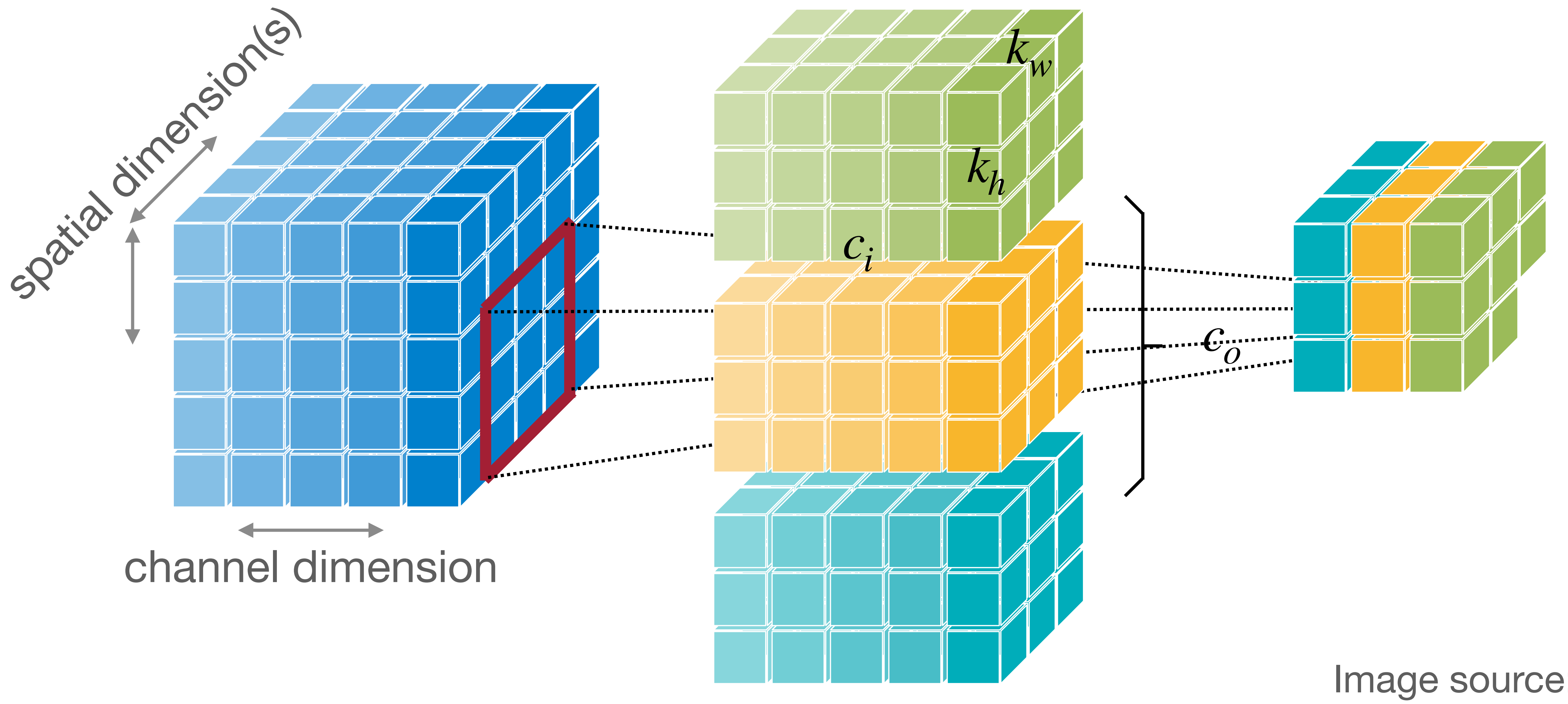


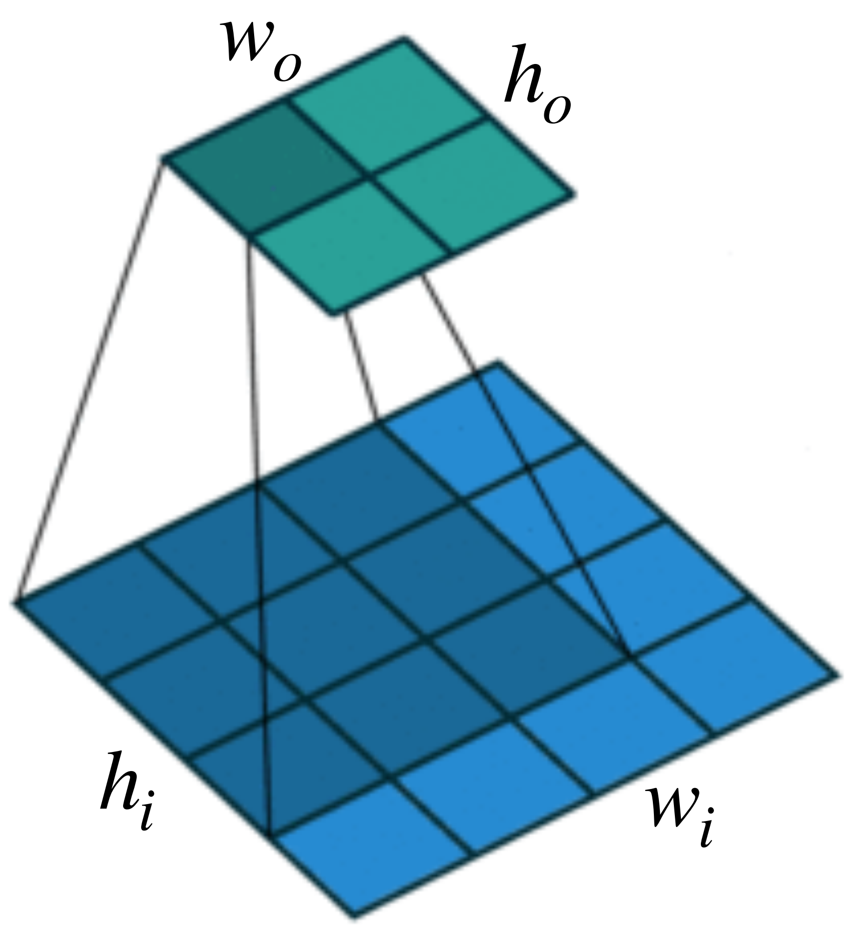
Image source: 1

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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

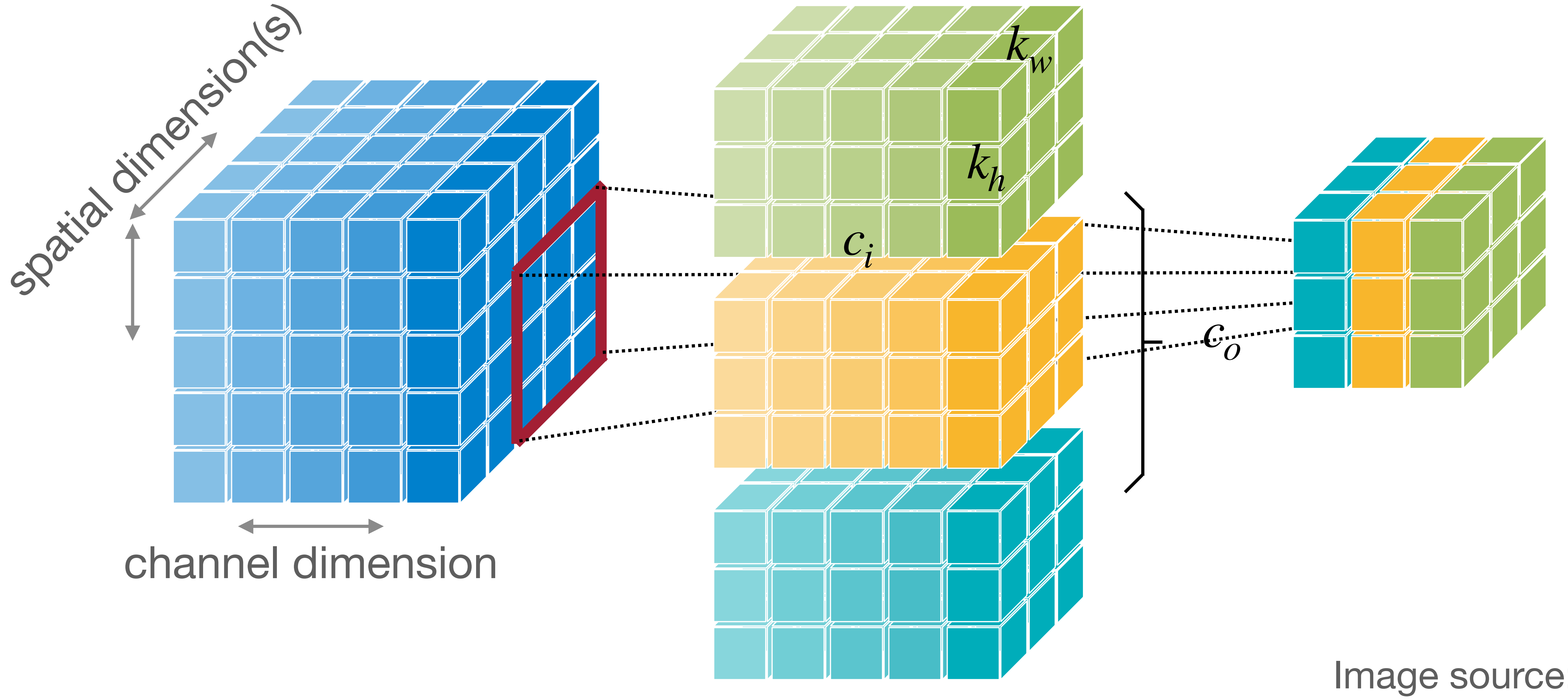


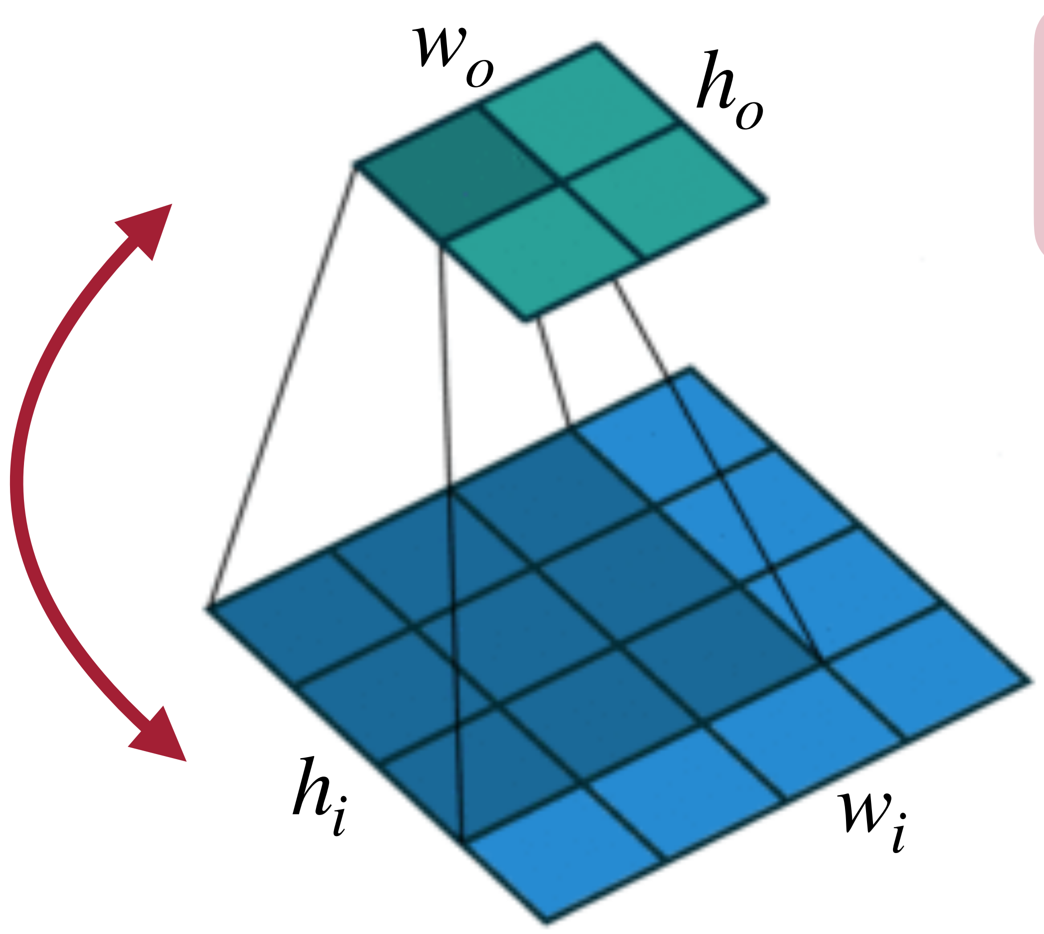
Image source: 1

Convolution Layer

The output neuron is connected to input neurons in the receptive field.

- **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)
  - 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



Feature map size becomes smaller.

$$h_o = h_i - k_h + 1$$

$$h_i = w_i = 4$$
$$k_h = k_w = 3$$
$$h_o = w_o$$
$$= 4 - 3 + 1$$
$$= 2$$

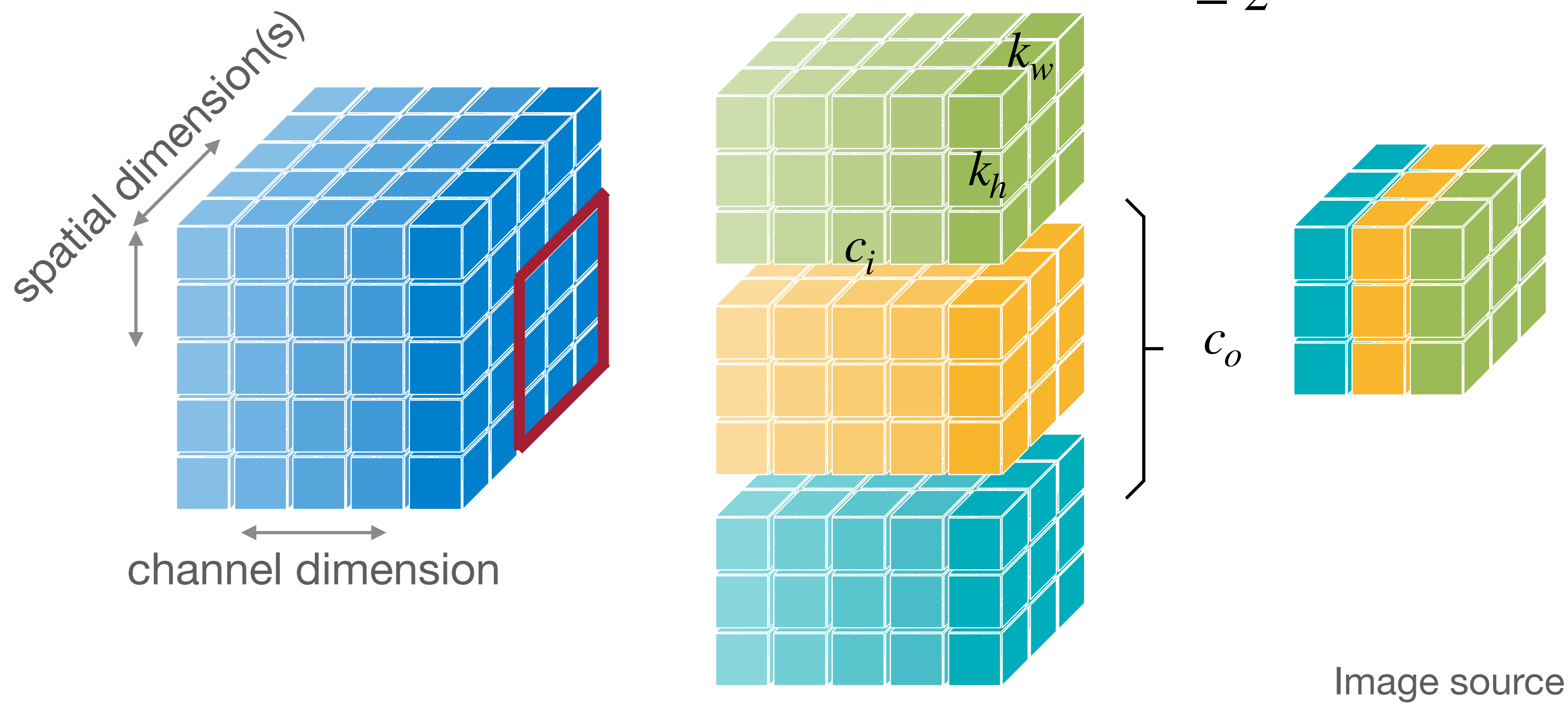
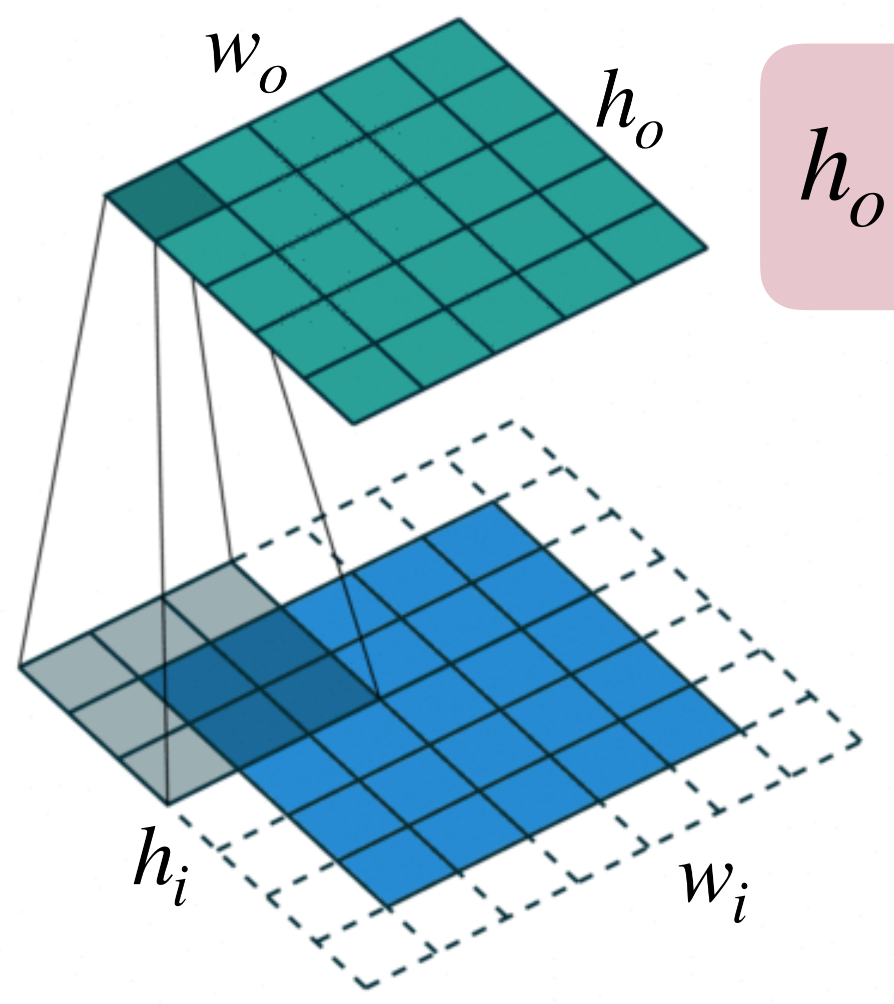


Image source: 1

Convolution Layer: Padding

- Padding can be used to keep the output feature map size is the same as input feature map size
  - **Zero Padding** pads the input boundaries with zero. (Default in PyTorch)
  - Other Paddings: Reflection Padding, Replication Padding, Constant Padding



$$h_o = h_i + 2p - k_h + 1$$

p is padding

$$h_i = w_i = 5$$
$$k_h = k_w = 3$$
$$h_o = w_o$$
$$= 5 + 2 \times 1 - 3 + 1$$
$$= 5$$

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

Zero Padding

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	2	3	0	0
0	0	4	5	6	0	0
0	0	7	8	9	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Reflection Padding

9	8	7	8	9	8	7
6	5	4	5	6	4	5
3	2	1	2	3	2	1
6	5	4	5	6	5	4
9	8	7	8	9	8	7
6	5	4	5	6	5	4
3	2	1	2	3	2	1

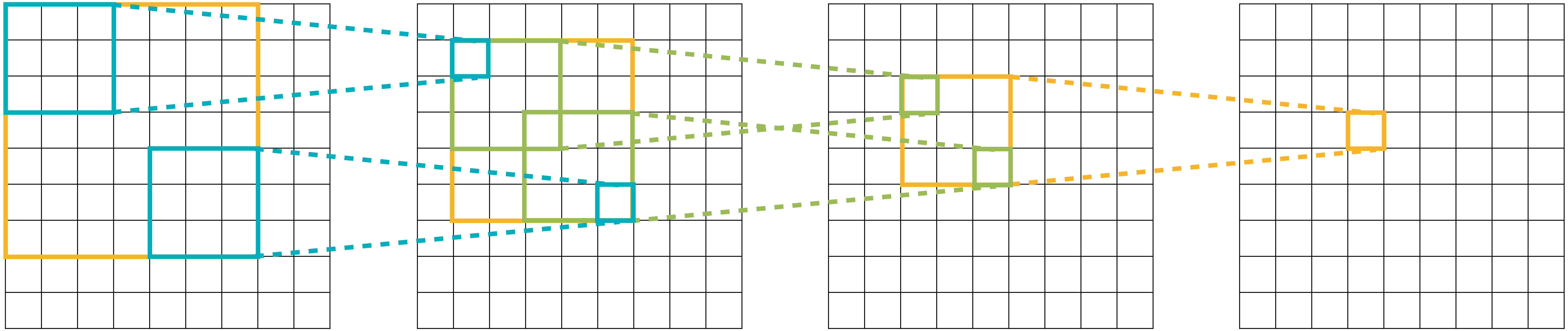
Replication Padding

1	1	1	2	3	3	3
1	1	1	2	3	3	3
1	1	1	2	3	3	3
4	4	4	5	6	6	6
7	7	7	8	9	9	9
7	7	7	8	9	9	9
7	7	7	8	9	9	9

Image source: 1

Convolution Layer: Receptive Field

- In convolution, each output element depends on $k_h \times k_w$ receptive field in the input.
- Each successive convolution adds $k-1$ to the receptive field size
- With L layers, the receptive field size is $L \cdot (k-1)+1$



For $L=2$ and $k=3$, the receptive field size is **5**

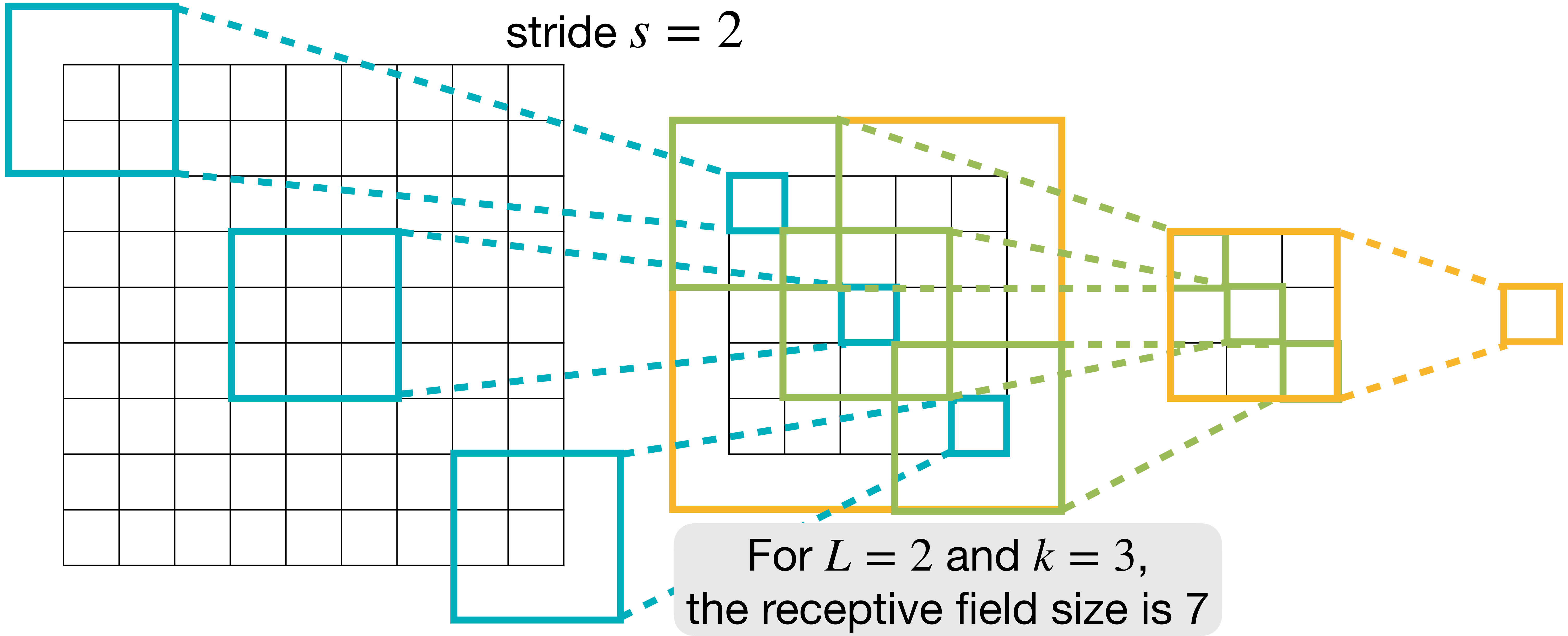
For $L=3$ and $k=3$, the receptive field size is **7**

Problem: For large images, we need many layers for each output to "see" the whole image

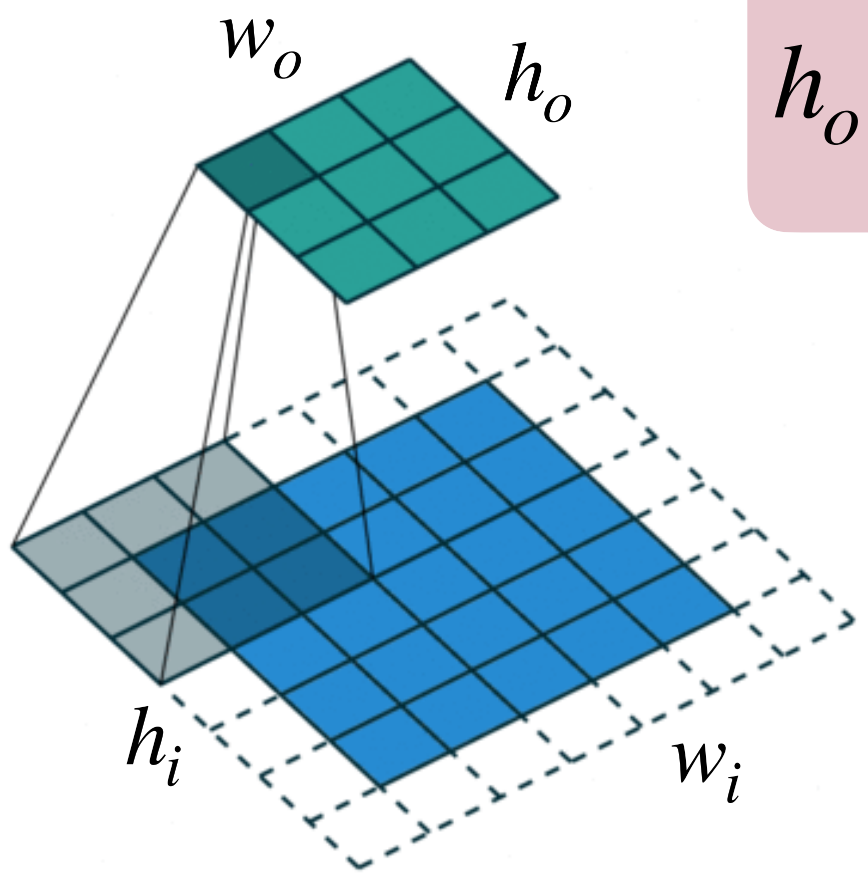
Solution: Downsample inside the neural network

Slide Inspiration: Ruohan Gao

Strided Convolution Layer


stride $s = 2$

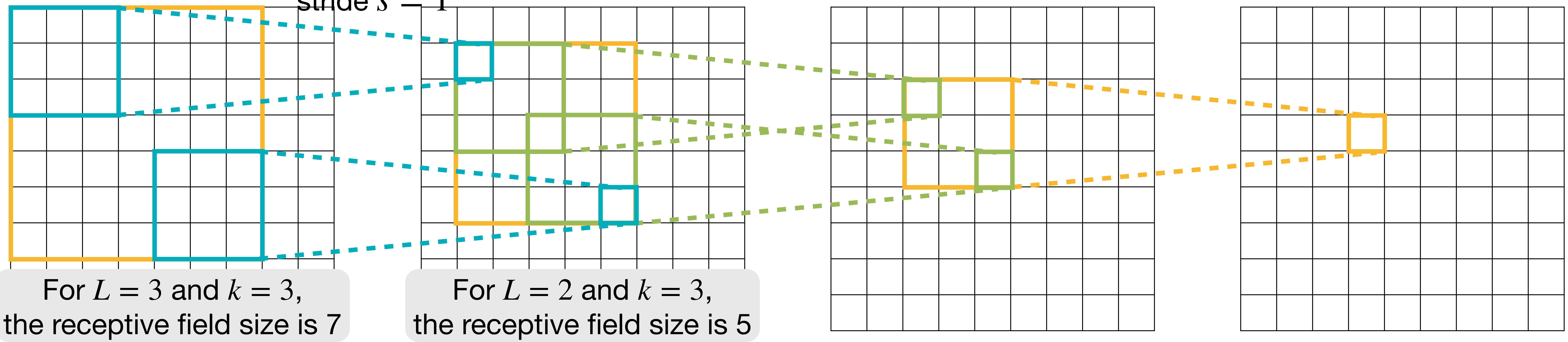
For $L = 2$ and $k = 3$, the receptive field size is 7



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

p is padding

s is stride


stride $s = 1$

For $L = 3$ and $k = 3$, the receptive field size is 7

For $L = 2$ and $k = 3$, the receptive field size is 5

Grouped Convolution Layer

A group of narrower convolutions

- **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 $\mathbf{b}$: $(c_o,)$

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

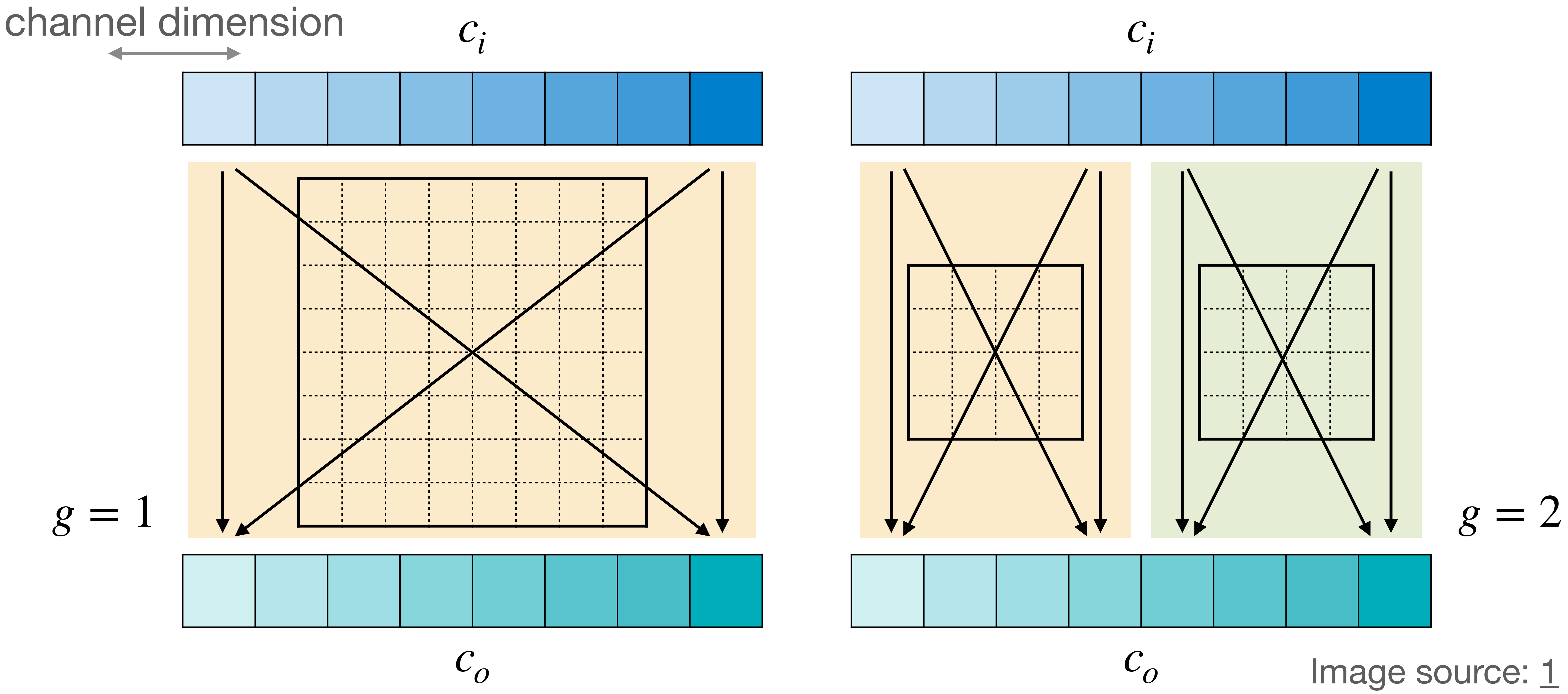


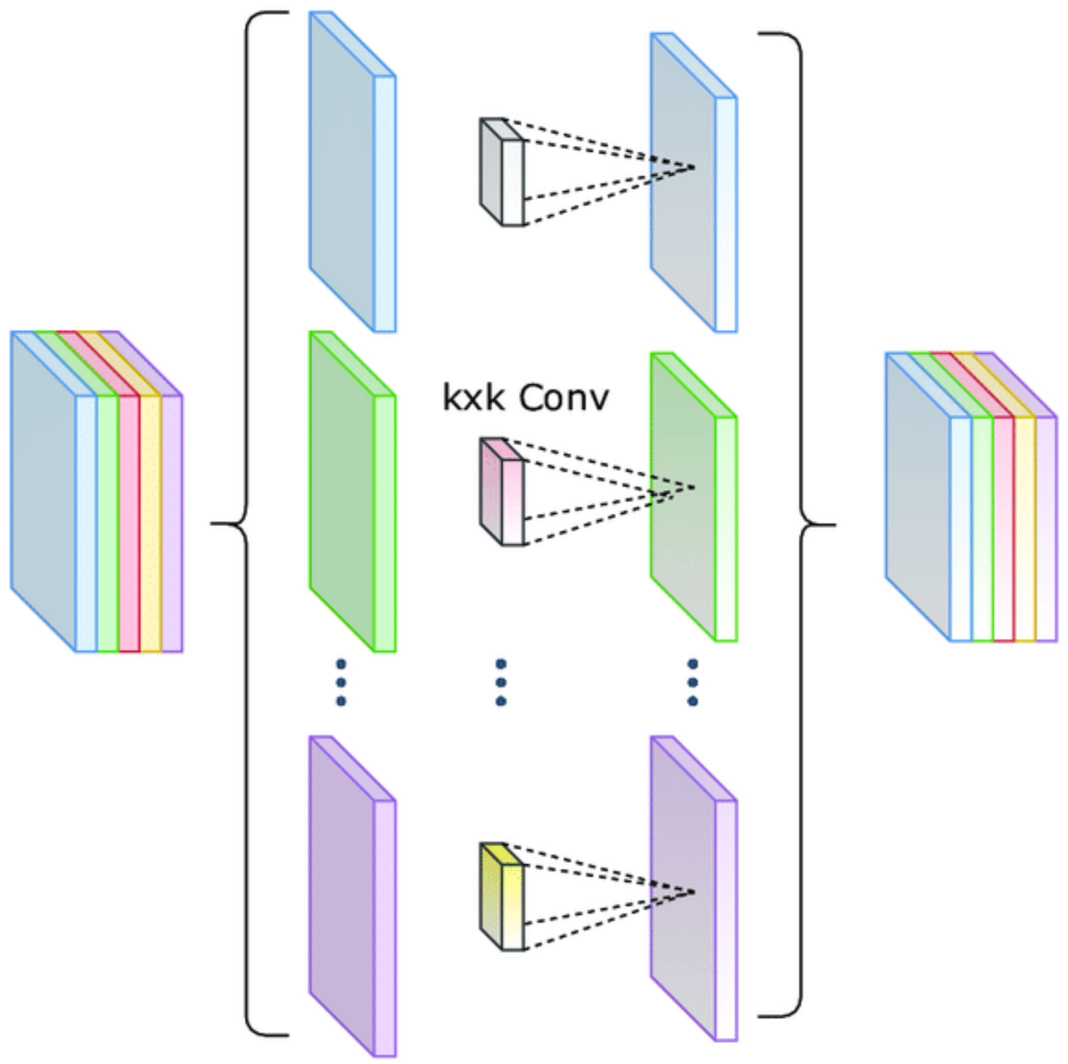
Image source: 1

Depthwise Convolution Layer

Independent filter for each channel: $g = c_i = c_o$ in grouped convolution

- **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 $\mathbf{b}$: $(c_o,)$

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



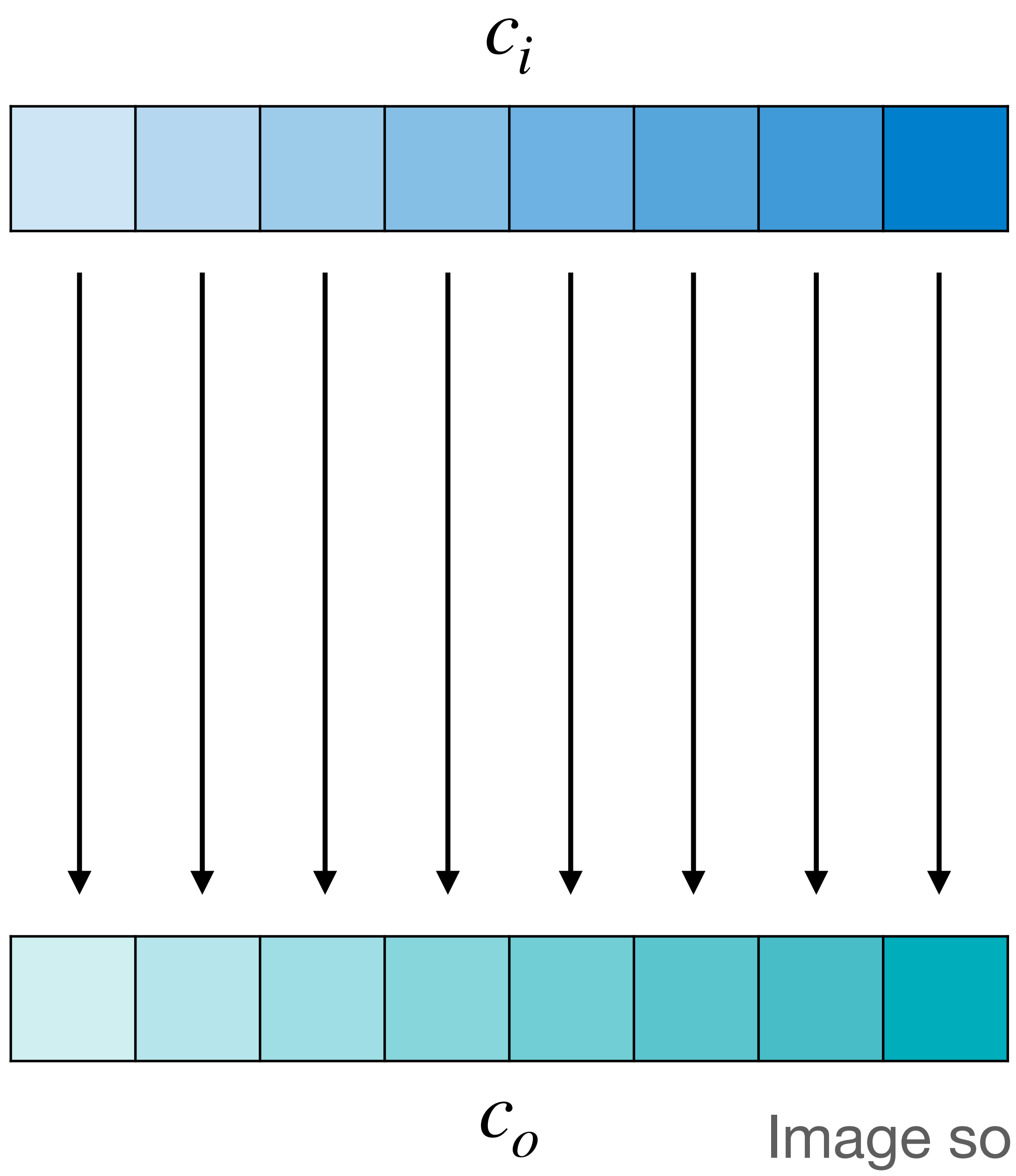
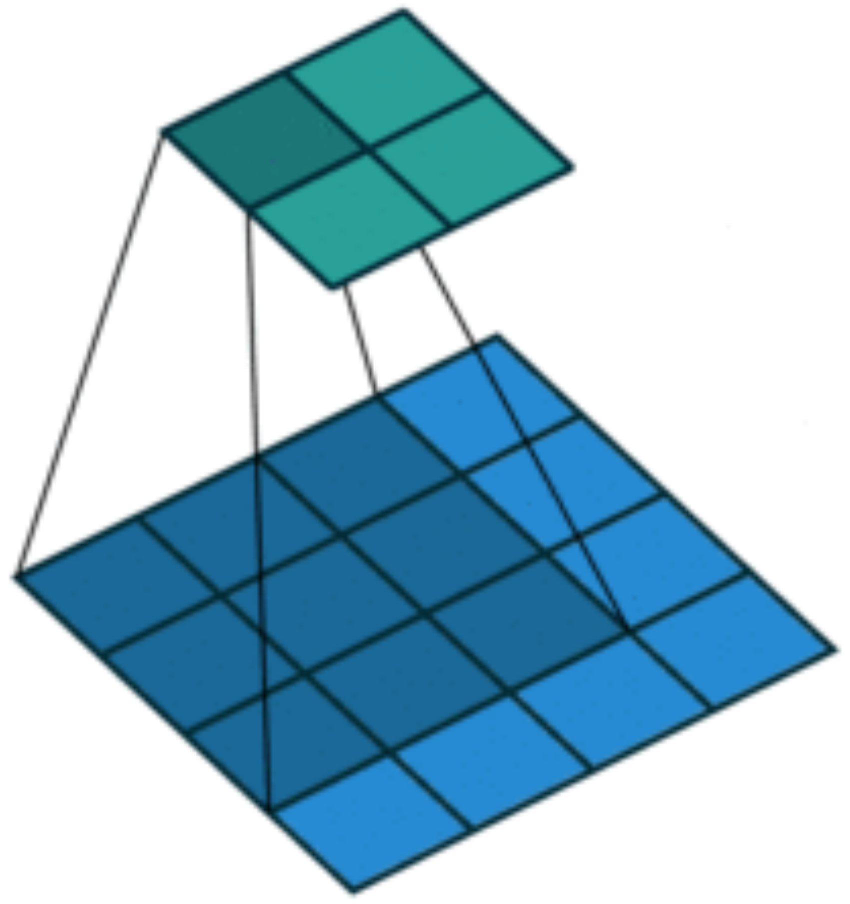


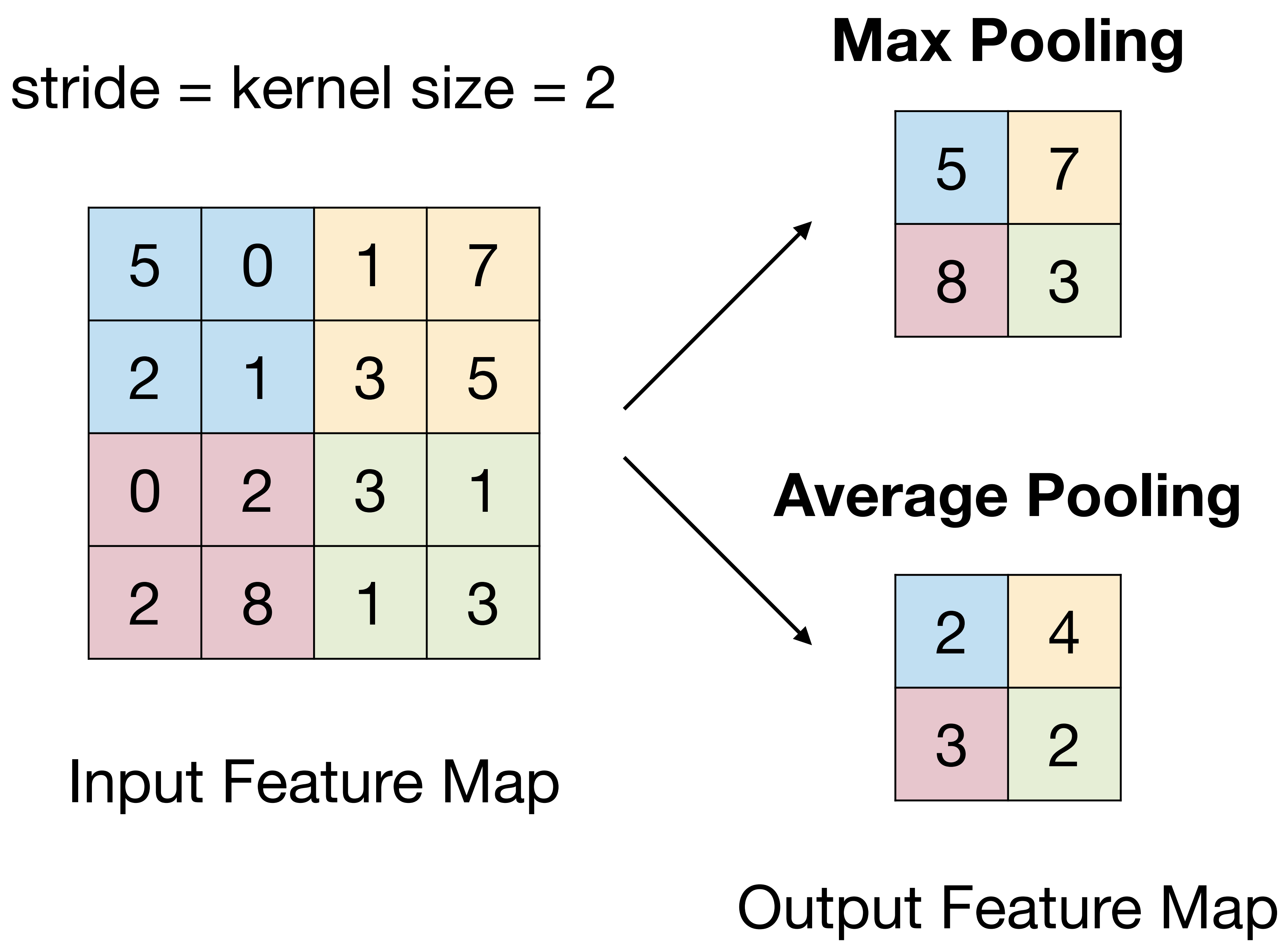
Image source: 1, 2

Pooling Layer

Downsample the feature map to a smaller size

- The output neuron pools the features in the receptive field, similar to convolution.
  - Usually, the stride is the same as the kernel size: $s = k$
- Pooling operates over each channel independently.
  - No learnable parameters





stride = kernel size = 2

5	0	1	7
2	1	3	5
0	2	3	1
2	8	1	3

Input Feature Map

Max Pooling

5	7
8	3

Average Pooling

2	4
3	2

Output Feature Map

Slide Inspiration: Ruohan Gao

Image source: 1

Normalization Layer

Normalizing the features makes optimization faster.

$$\mu_i = \frac{1}{m} \sum_{k \in \mathcal{S}_i} x_k$$

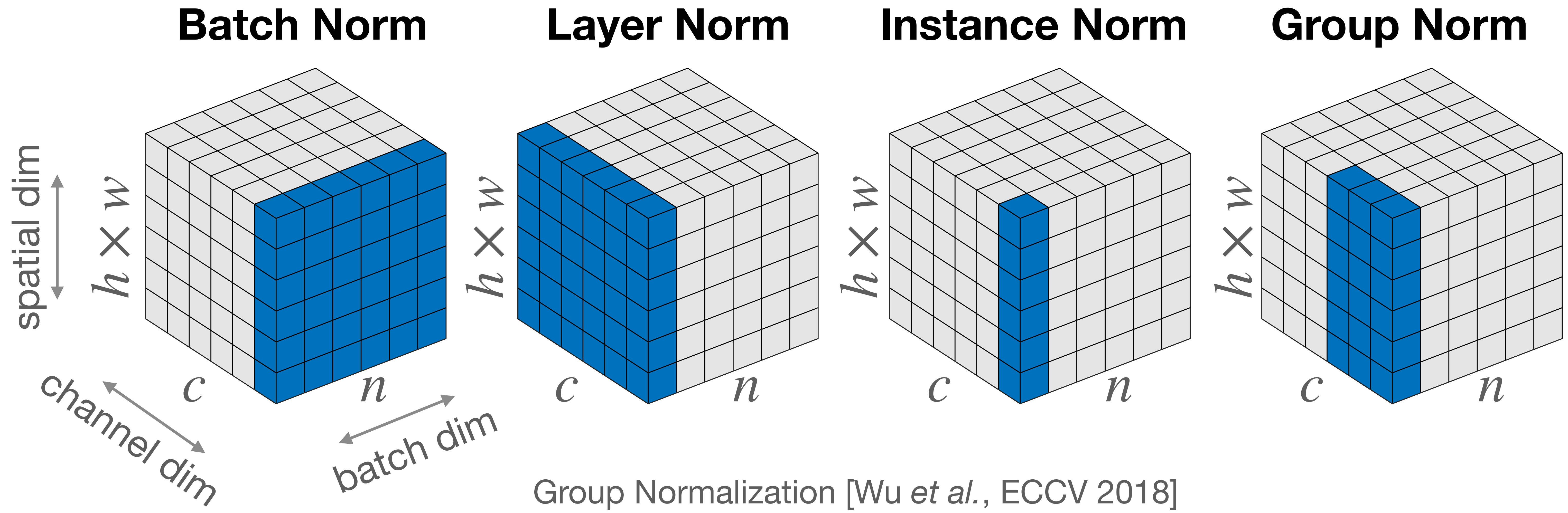
$$\sigma_i = \sqrt{\frac{1}{m} \sum_{k \in \mathcal{S}_i} (x_k - \mu_i)^2 + \epsilon}$$

- Normalization layer normalizes the features as follows,

$$\hat{x}_i = \frac{1}{\sigma}\left(x_i - \mu_i\right)$$

  - μ_i is the mean, and σ_i is the standard deviation (std) over the set of pixels $\mathcal{S}_i$.
- Then learns a *per-channel* linear transform (trainable scale γ and shift β) to compensate for the possible lost of representational ability.

$$y = \gamma_{i_c} \hat{x}_i + \beta_{i_c}$$

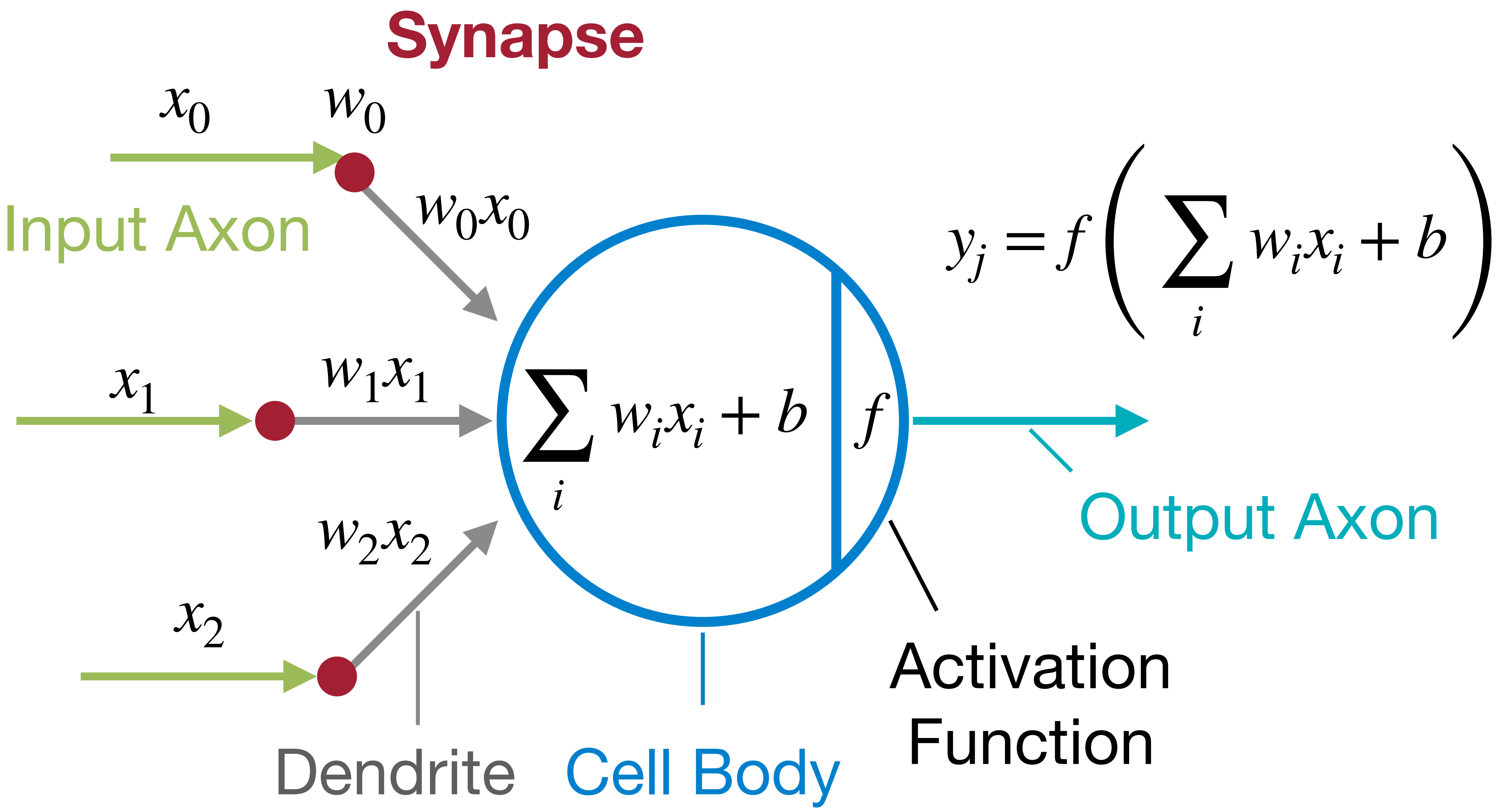


Different normalizations use different definitions of the set $\mathcal{S}_i$ (colored in blue)

Group Normalization [Wu *et al.*, ECCV 2018]

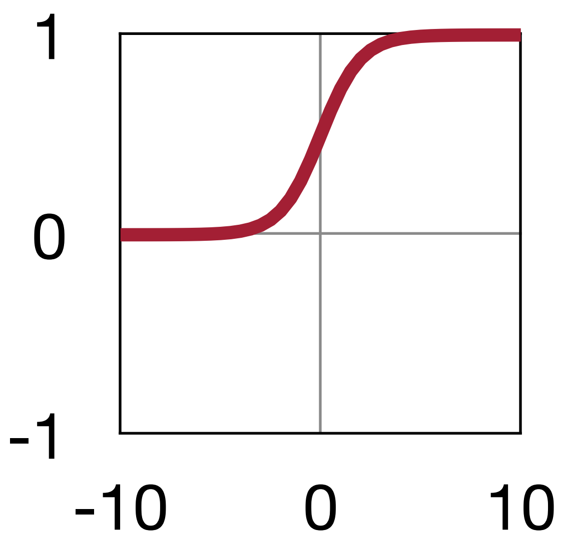
Activation Function

Activation functions are typically non-linear functions



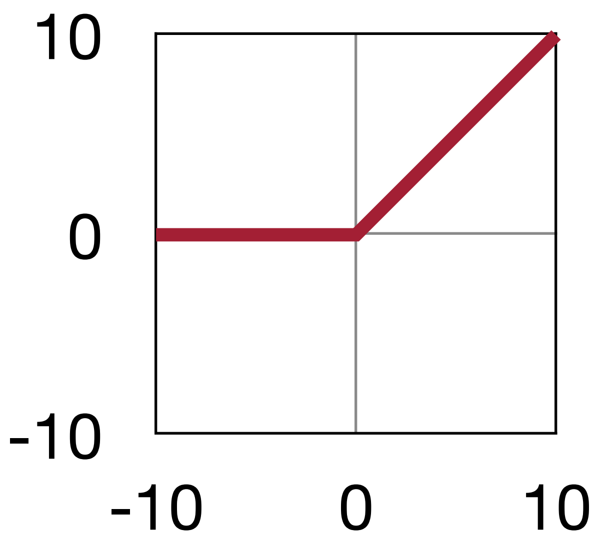
$$y_j = f\left(\sum_i w_i x_i + b\right)$$

Sigmoid



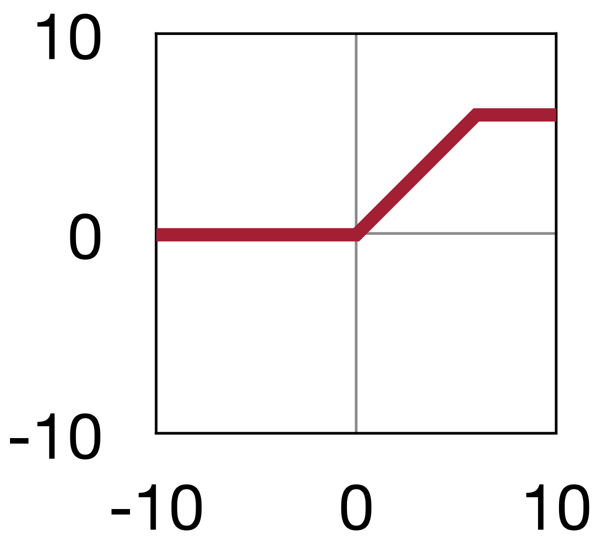
$$y = 1/(1 + e^{-x})$$

ReLU



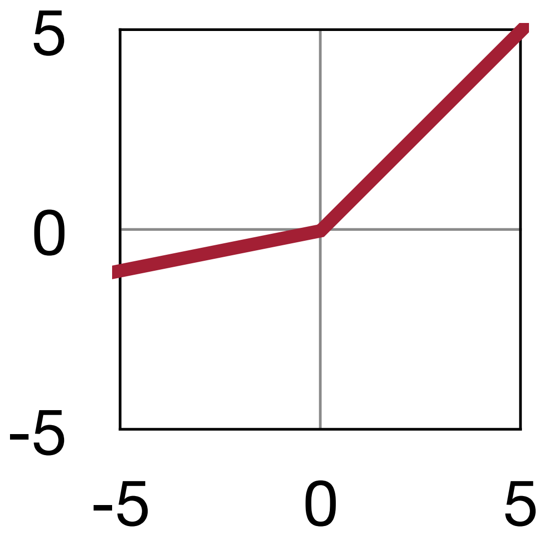
$$y = \max(0, x)$$

ReLU6



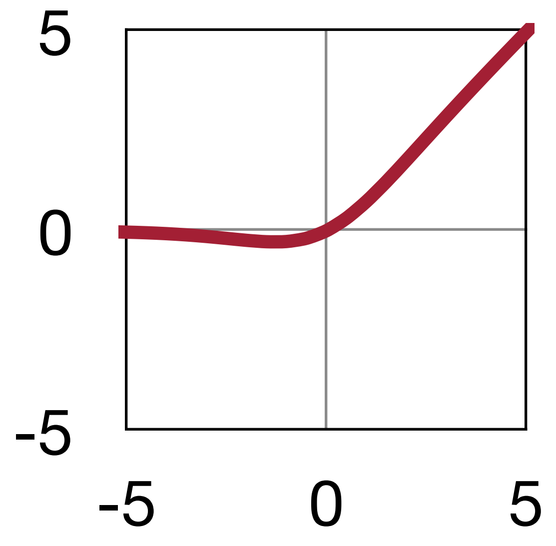
$$y = \min\big(\max(0, x), 6\big)$$

Leaky ReLU



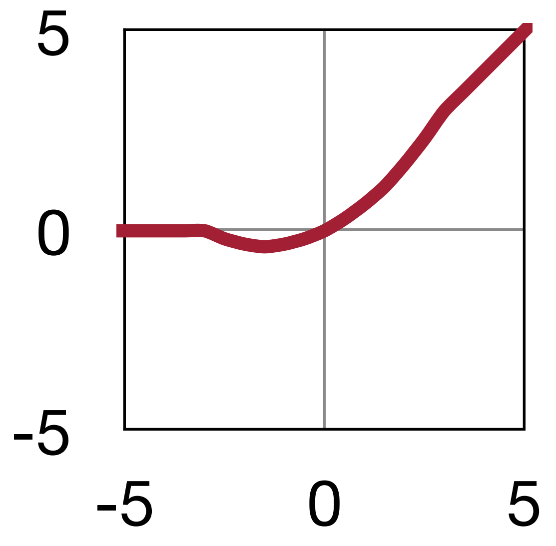
$$y = \max(\alpha x, x)$$

Swish



$$y = x/(1 + e^{-x})$$

Hard Swish

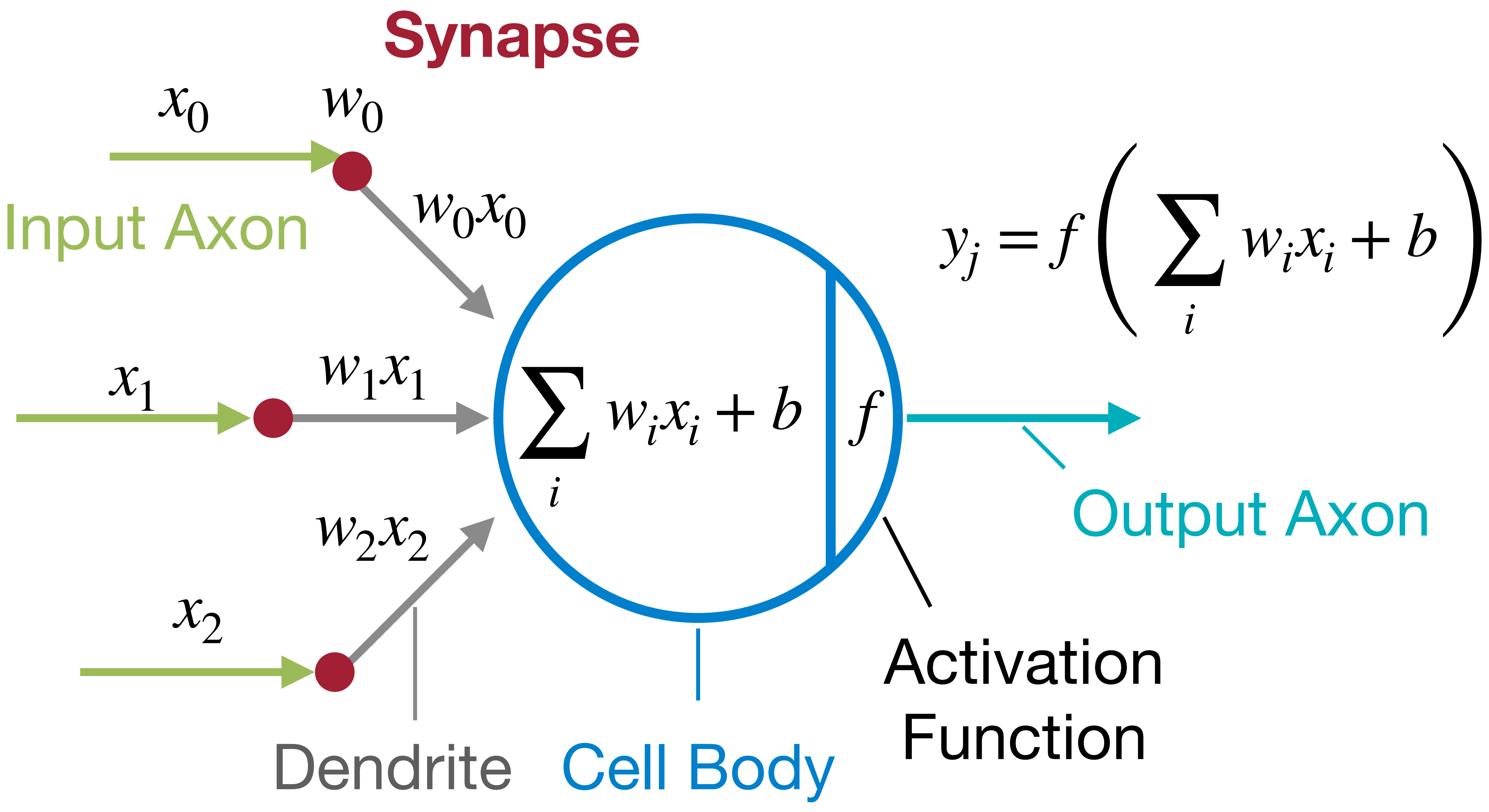


$$y = \begin{cases} 0, & x \le -3 \\ x, & x \ge 3 \\ x \cdot (x+3)/6, & otherwise \end{cases}$$

Other Activation Functions: Tanh, GELU, ELU, Mish…

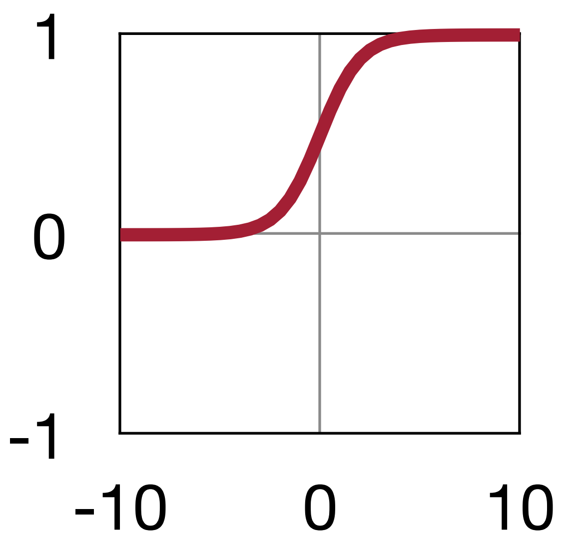
Activation Function

Activation functions are typically non-linear functions



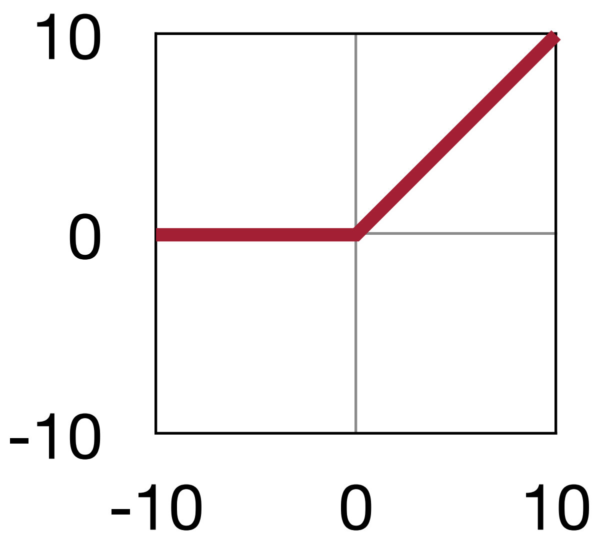
$$y_j = f\left(\sum_i w_i x_i + b\right)$$

Sigmoid



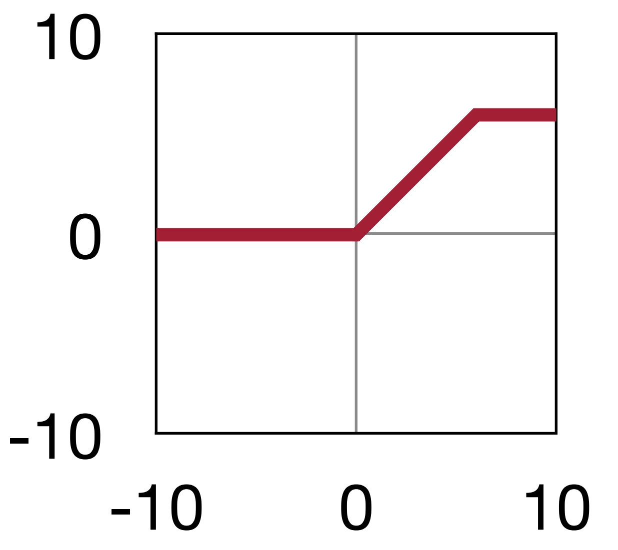
$$y = 1/(1 + e^{-x})$$

ReLU



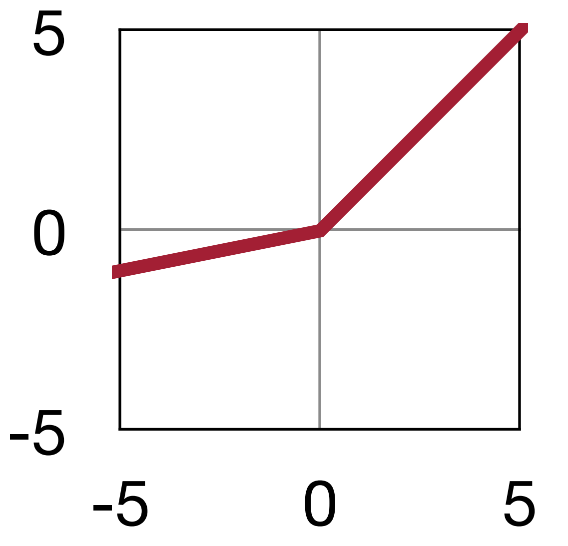
$$y = \max(0, x)$$

ReLU6



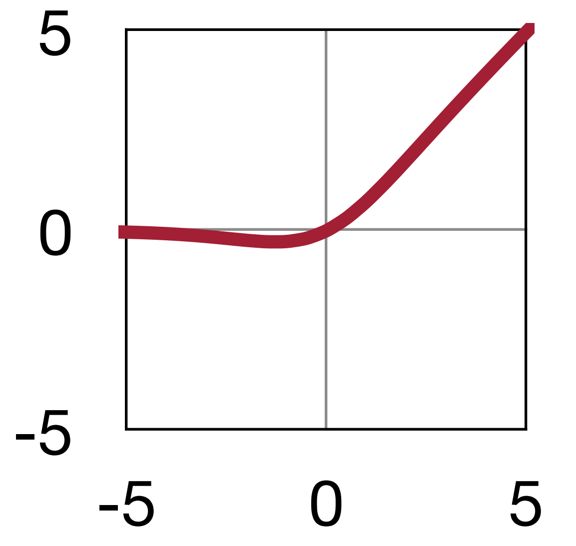
$$y = \min\left(\max(0, x), 6\right)$$

Leaky ReLU



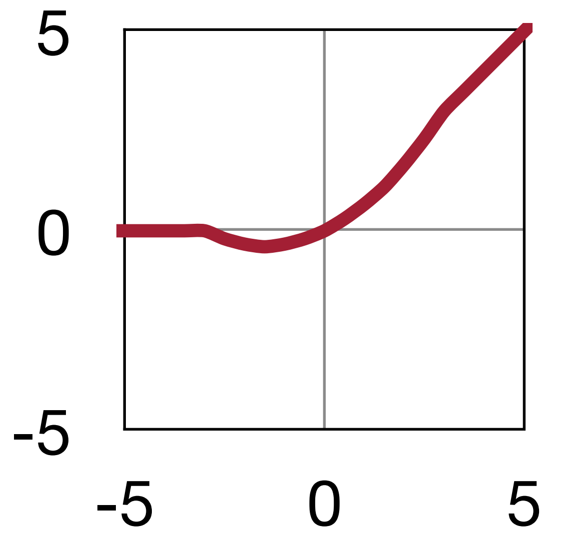
$$y = \max(\alpha x, x)$$

Swish



$$y = x/(1 + e^{-x})$$

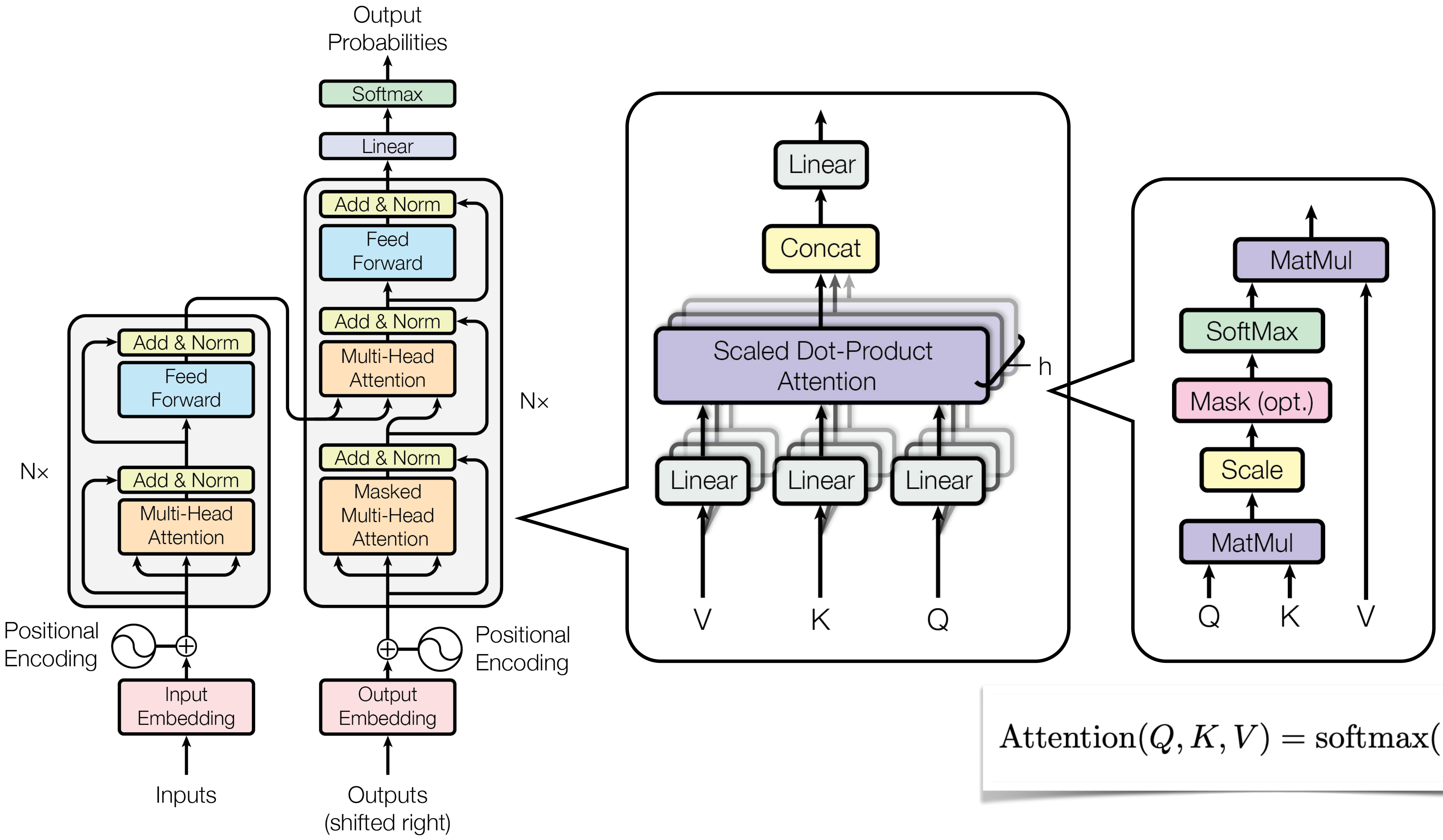
Hard Swish



$$y = \begin{cases} 0, & x \leq -3 \\ x, & x \geq 3 \\ x \cdot (x+3)/6, & otherwise \end{cases}$$

Other Activation Functions: Tanh, GELU, ELU, Mish…

Transformer



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

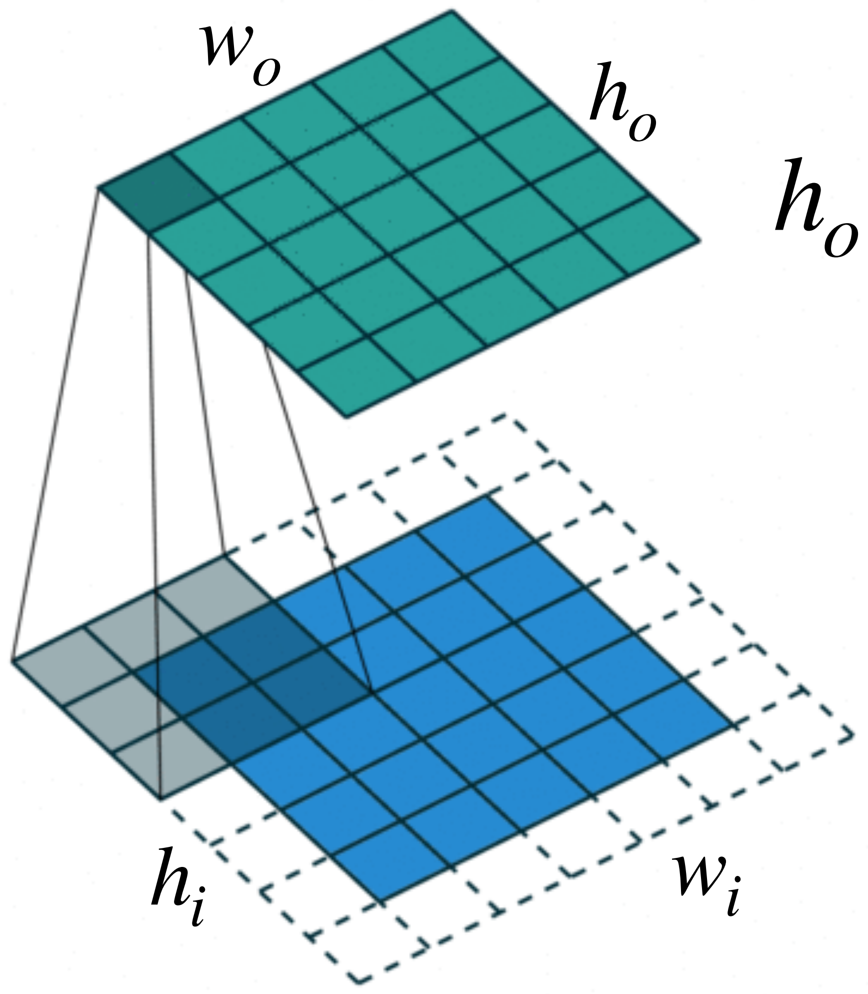
Attention is All You Need [Vaswani *et al.*, NeurIPS 2017]

Popular CNN Architectures

AlexNet

AlexNet	C × H × W	H, W
Image (3×224×224)	3×224×224	
11×11 Conv, channel 96, stride 4, pad 2	96×55×55	$\frac{224+2\times2-11}{4}+1=55$
3x3 MaxPool, stride 2	96×27×27	$\frac{55+0-3}{2}+1=27$
5x5 Conv, channel 256, pad 2, groups 2	256×27×27	$\frac{27+2\times2-5}{1}+1=27$
3x3 MaxPool, stride 2	256×13×13	$\frac{27+0-3}{2}+1=13$
3x3 Conv, channel 384, pad 1	384×13×13	$\frac{13+2\times1-3}{1}+1=13$
3x3 Conv, channel 384, pad 1, groups 2	384×13×13	$\frac{13+2\times1-3}{1}+1=13$
3x3 Conv, channel 256, pad 1, groups 2	256×13×13	$\frac{13+2\times1-3}{1}+1=13$
3x3 MaxPool, stride 2	256×6×6	$\frac{13+0-3}{2}+1=6$
Linear, channel 4096	4096	
Linear, channel 4096	4096	
Linear, channel 1000	1000	

Convolution Layer / Pooling Layer

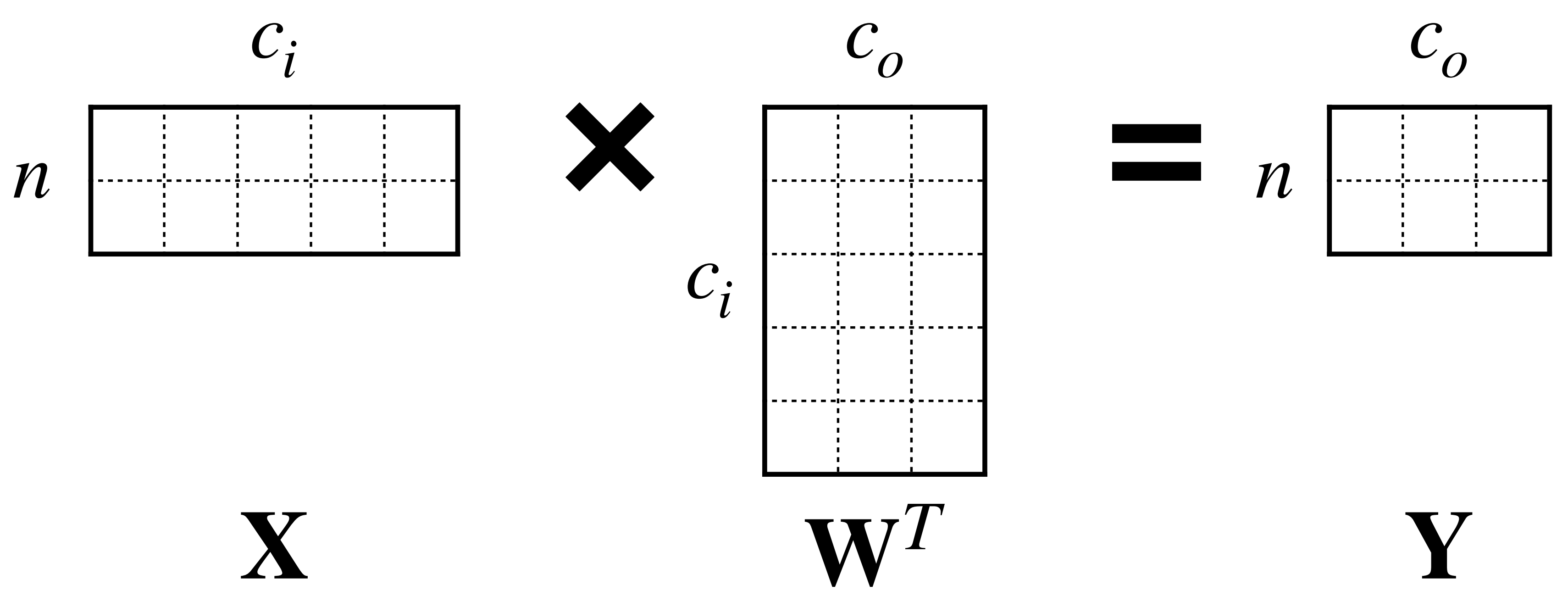


$$h_o = \frac{h_i + 2p - k_h}{s} + 1$$

p is padding

s is stride

Linear Layer



ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky *et al.*, NeurIPS 2012]

VGG-16

AlexNet

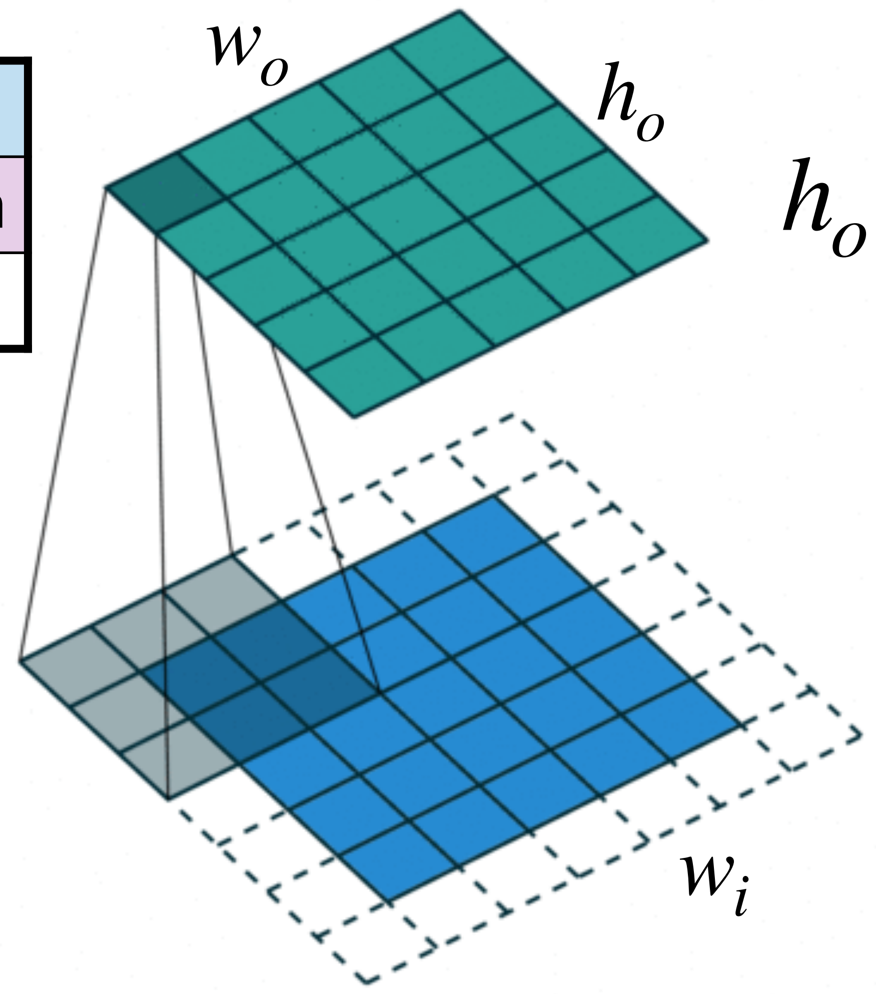
- Image (3×224×224)
- 11×11 Conv, channel 96, stride 4, pad 2
- 3×3 MaxPool, stride 2
- 5×5 Conv, channel 256, pad 2, groups 2
- 3×3 MaxPool, stride 2
- 3×3 Conv, channel 384, pad 1
- 3×3 Conv, channel 384, pad 1, groups 2
- 3×3 Conv, channel 256, pad 1, groups 2
- 3×3 MaxPool, stride 2
- Linear, channel 4096
- Linear, channel 4096
- Linear, channel 1000

VGG-16

	Image (3×224×224)
1	3x3 Conv, channel 64, pad 1
2	3x3 Conv, channel 64, pad 1
	2x2 MaxPool, stride 2
3	3x3 Conv, channel 128, pad 1
4	3x3 Conv, channel 128, pad 1
	2x2 MaxPool, stride 2
5	3x3 Conv, channel 256, pad 1
6	3x3 Conv, channel 256, pad 1
7	3x3 Conv, channel 256, pad 1
	2x2 MaxPool, stride 2
8	3x3 Conv, channel 512, pad 1
9	3x3 Conv, channel 512, pad 1
10	3x3 Conv, channel 512, pad 1
	2x2 MaxPool, stride 2
11	3x3 Conv, channel 512, pad 1
12	3x3 Conv, channel 512, pad 1
13	3x3 Conv, channel 512, pad 1
	2x2 MaxPool, stride 2
14	Linear, channel 4096
15	Linear, channel 4096
16	Linear, channel 1000

Conv
BatchNorm
ReLU

Convolution Layer / Pooling Layer

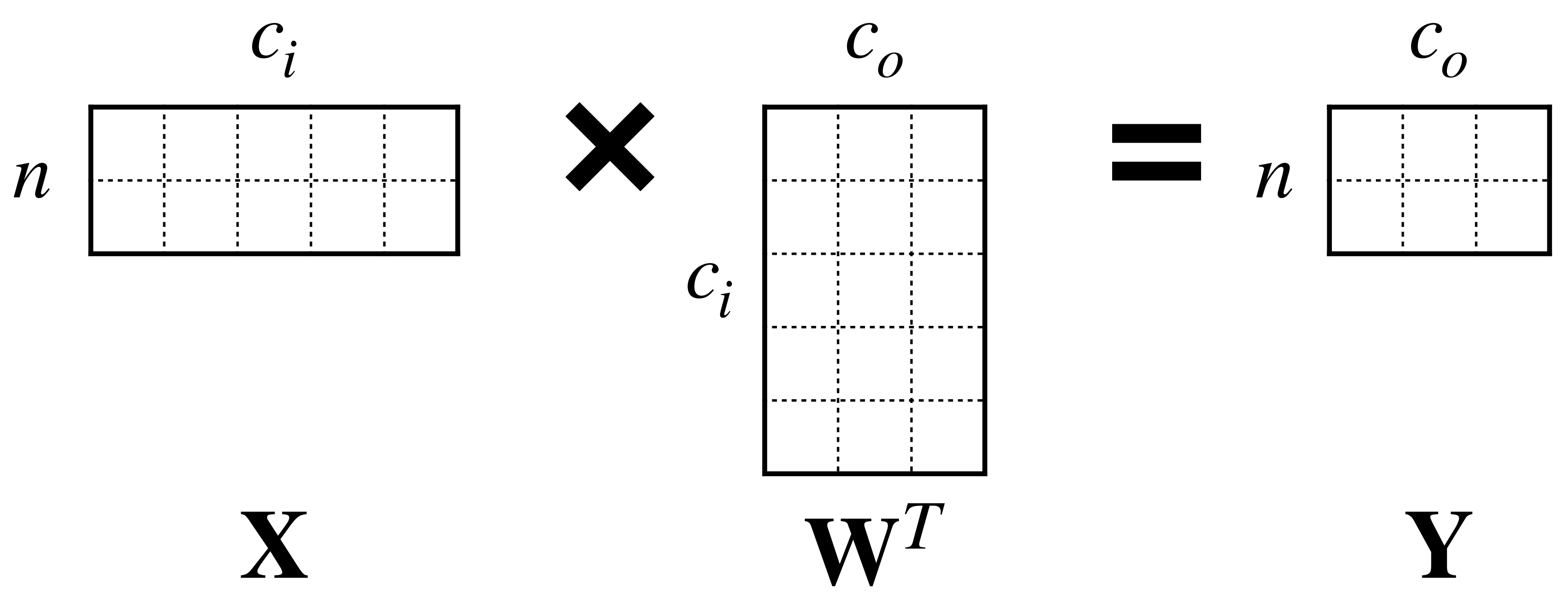


$$h_o = \frac{h_i + 2p - k_h}{s} + 1$$

p is padding

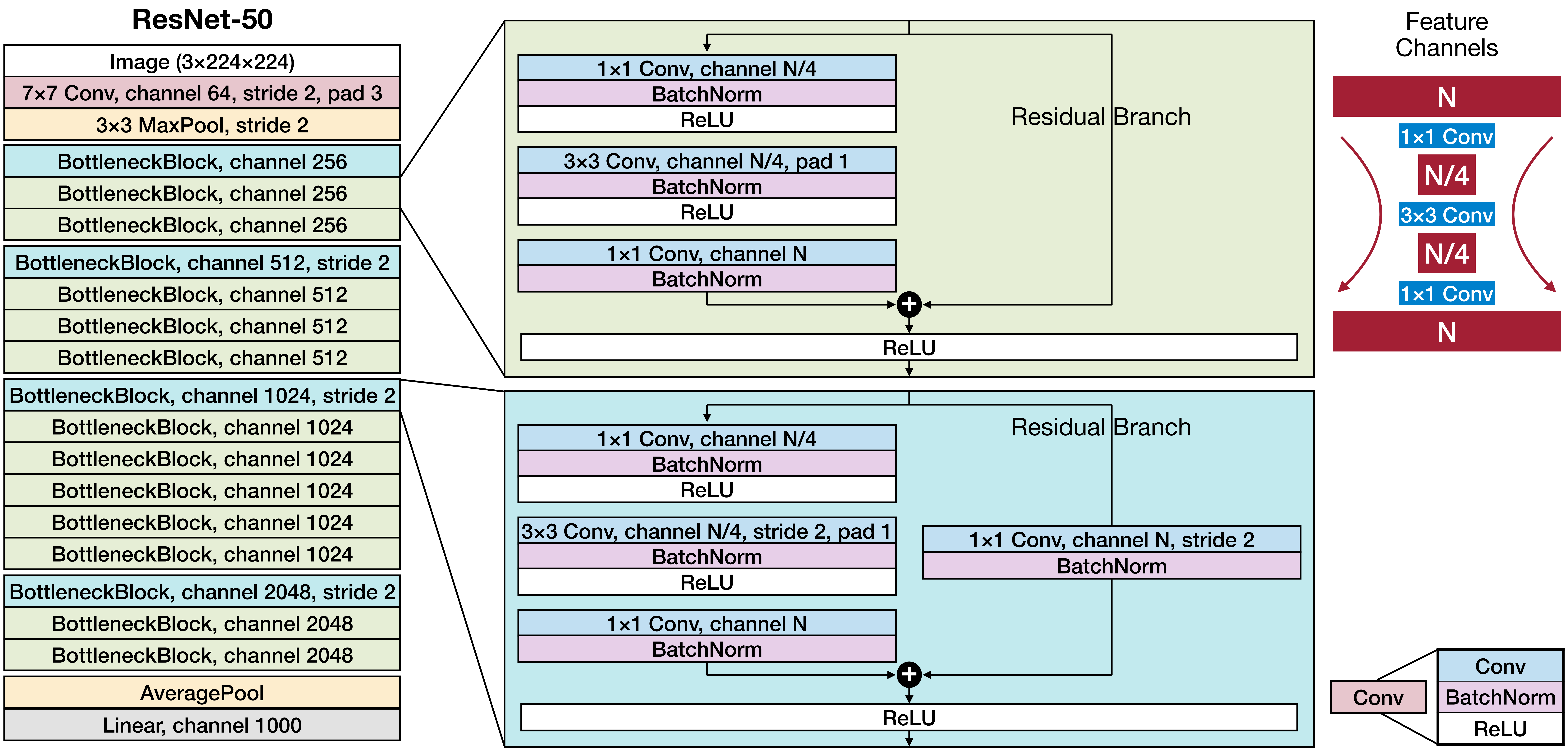
s is stride

Linear Layer



Very Deep Convolutional Networks for Large-Scale Image Recognition [Simonyan *et al.*, ICLR 2015]

ResNet-50



ResNet-50

Image (3×224×224)
7×7 Conv, channel 64, stride 2, pad 3
3×3 MaxPool, stride 2
BottleneckBlock, channel 256
BottleneckBlock, channel 256
BottleneckBlock, channel 256
BottleneckBlock, channel 512, stride 2
BottleneckBlock, channel 512
BottleneckBlock, channel 512
BottleneckBlock, channel 512
BottleneckBlock, channel 1024, stride 2
BottleneckBlock, channel 1024
BottleneckBlock, channel 1024
BottleneckBlock, channel 1024
BottleneckBlock, channel 1024
BottleneckBlock, channel 1024
BottleneckBlock, channel 2048, stride 2
BottleneckBlock, channel 2048
BottleneckBlock, channel 2048
AveragePool
Linear, channel 1000

1×1 Conv, channel N/4
BatchNorm
ReLU
3×3 Conv, channel N/4, pad 1
BatchNorm
ReLU
1×1 Conv, channel N
BatchNorm
Residual Branch
ReLU

1×1 Conv, channel N/4
BatchNorm
ReLU
3×3 Conv, channel N/4, stride 2, pad 1
BatchNorm
ReLU
1×1 Conv, channel N
BatchNorm
Residual Branch
1×1 Conv, channel N, stride 2
BatchNorm
ReLU

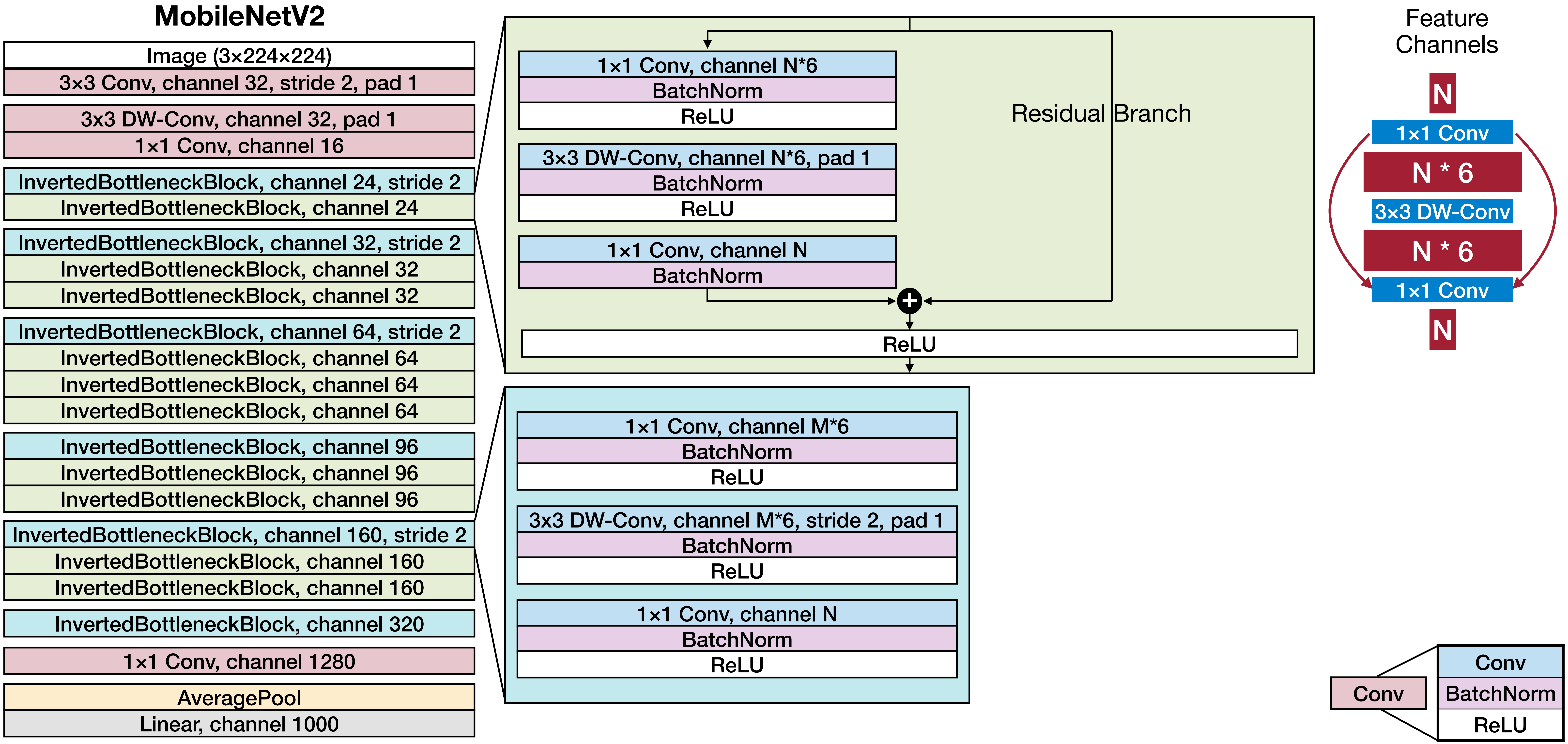
Feature Channels
N
1×1 Conv
N/4
3×3 Conv
N/4
1×1 Conv
N

Conv
Conv
BatchNorm
ReLU

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

Figure Inspiration: 1

MobileNetV2



MobileNetV2

Image (3×224×224)
3×3 Conv, channel 32, stride 2, pad 1

3x3 DW-Conv, channel 32, pad 1
1×1 Conv, channel 16

InvertedBottleneckBlock, channel 24, stride 2
InvertedBottleneckBlock, channel 24

InvertedBottleneckBlock, channel 32, stride 2
InvertedBottleneckBlock, channel 32
InvertedBottleneckBlock, channel 32

InvertedBottleneckBlock, channel 64, stride 2
InvertedBottleneckBlock, channel 64
InvertedBottleneckBlock, channel 64
InvertedBottleneckBlock, channel 64

InvertedBottleneckBlock, channel 96
InvertedBottleneckBlock, channel 96
InvertedBottleneckBlock, channel 96

InvertedBottleneckBlock, channel 160, stride 2
InvertedBottleneckBlock, channel 160
InvertedBottleneckBlock, channel 160

InvertedBottleneckBlock, channel 320

1×1 Conv, channel 1280

AveragePool
Linear, channel 1000

1×1 Conv, channel N*6
BatchNorm
ReLU

3×3 DW-Conv, channel N*6, pad 1
BatchNorm
ReLU

1×1 Conv, channel N
BatchNorm

Residual Branch

ReLU

1×1 Conv, channel M*6
BatchNorm
ReLU

3x3 DW-Conv, channel M*6, stride 2, pad 1
BatchNorm
ReLU

1×1 Conv, channel N
BatchNorm
ReLU

Feature Channels

N
1×1 Conv
N * 6
3×3 DW-Conv
N * 6
1×1 Conv
N

Conv

Conv
BatchNorm
ReLU

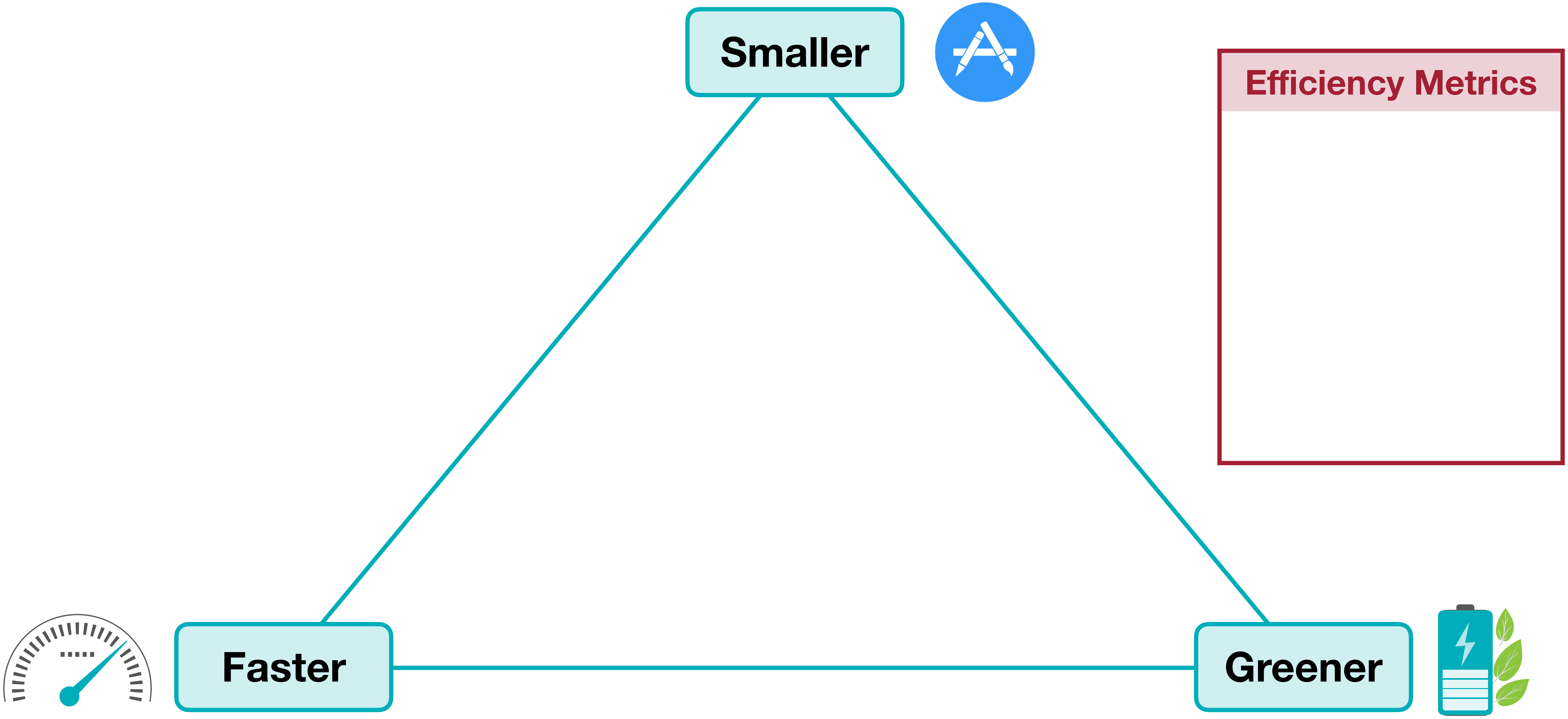
MobileNetV2: Inverted Residuals and Linear Bottlenecks [Sandler *et al.*, CVPR 2018]

Figure Inspiration: 1

Efficiency Metrics

How should we measure the efficiency of neural networks?

Efficiency of Neural Networks



Smaller

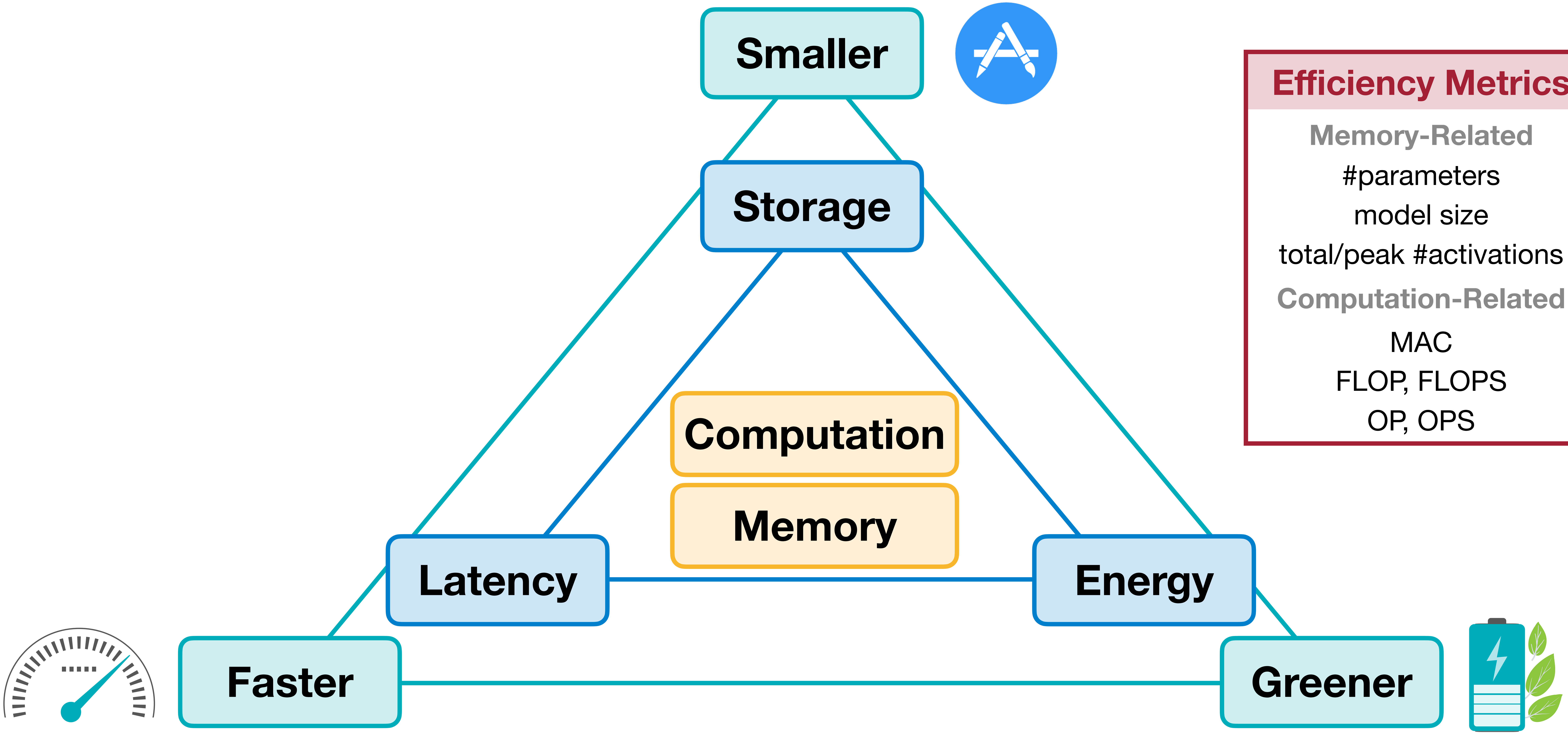
Faster

Greener

Efficiency Metrics

Image source: 1

Efficiency of Neural Networks



Efficiency Metrics

Memory-Related

#parameters

model size

total/peak #activations

Computation-Related

MAC

FLOP, FLOPS

OP, OPS

Image source: 1

Latency

Measures the delay for a specific task

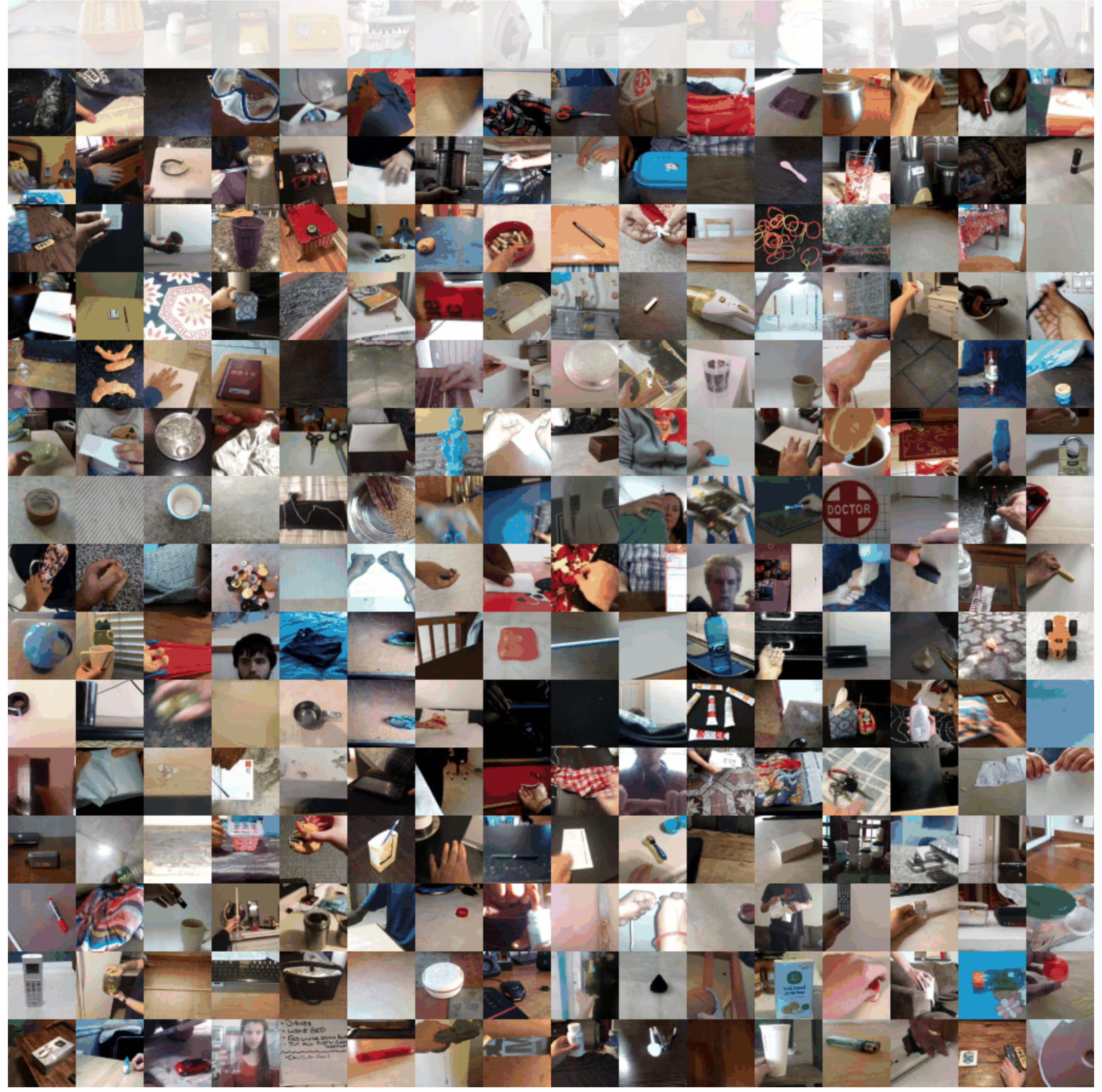
High Latency
638ms

Low Latency
46ms

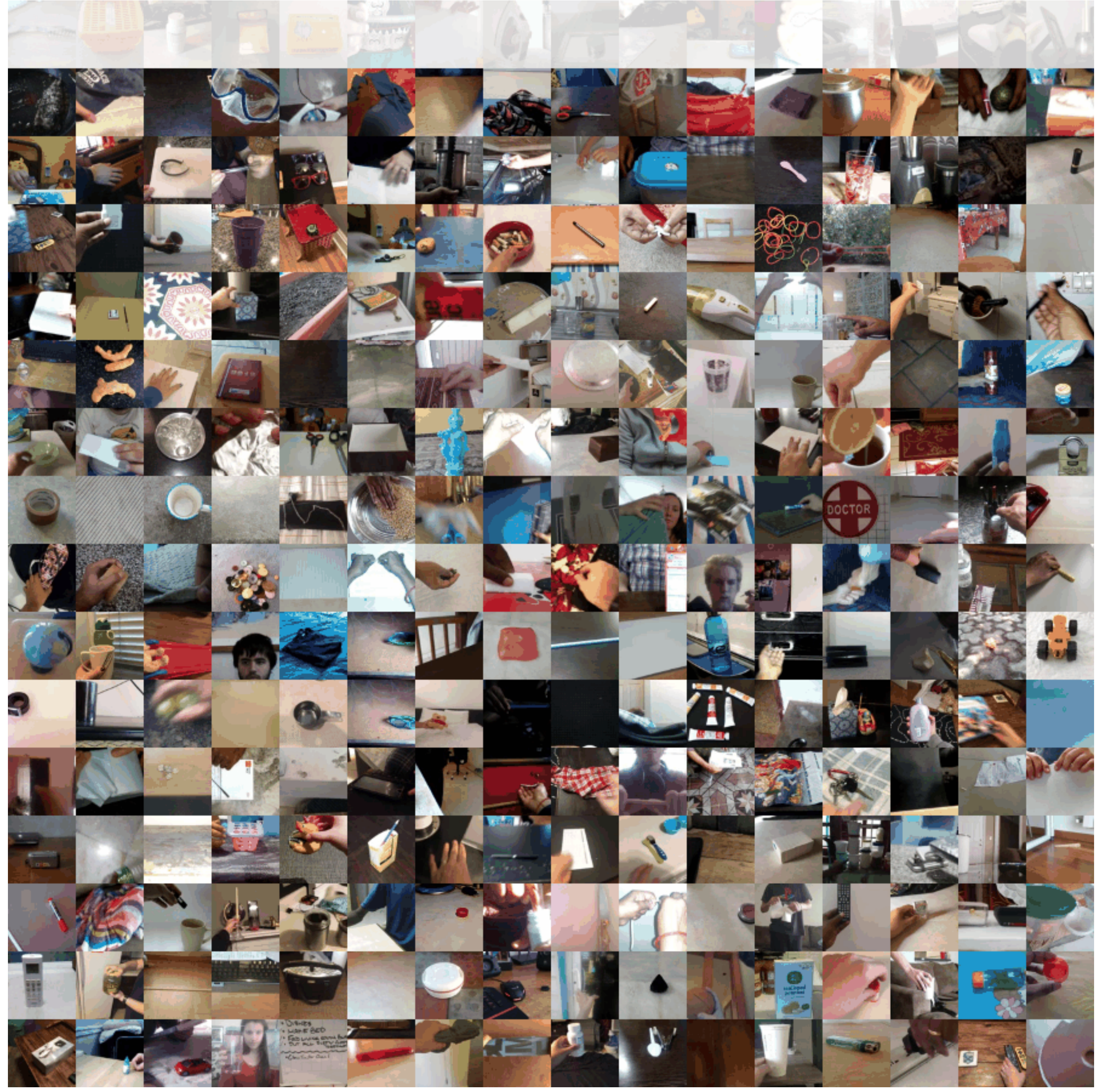
Speed is measured on Nvidia Jetson AGX Orin with TensorRT, fp16, batch size 1.

Throughput

Measures the rate at which data is processed



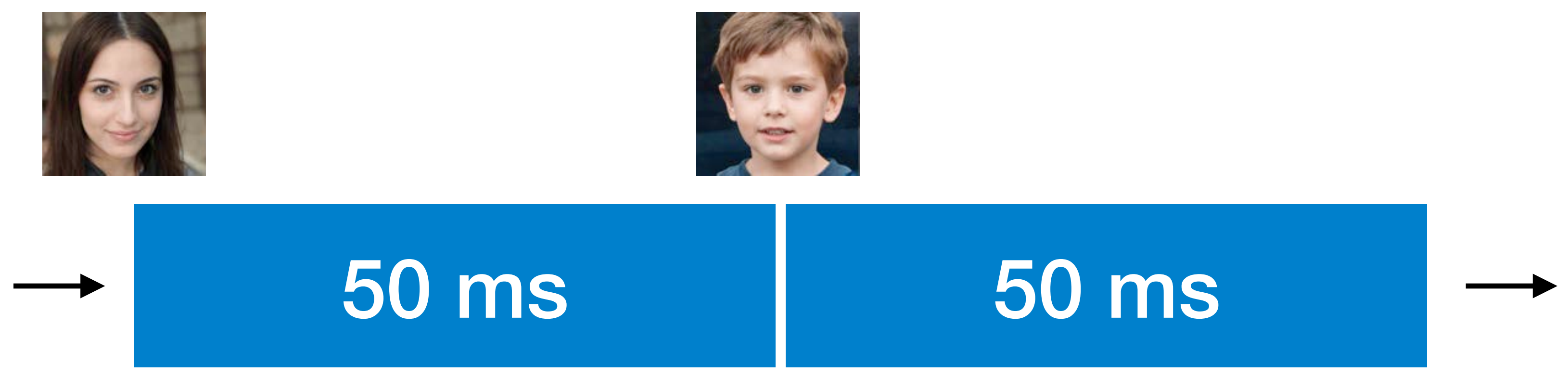
Low Throughput = 6.1 video/s



High Throughput = 77.4 video/s

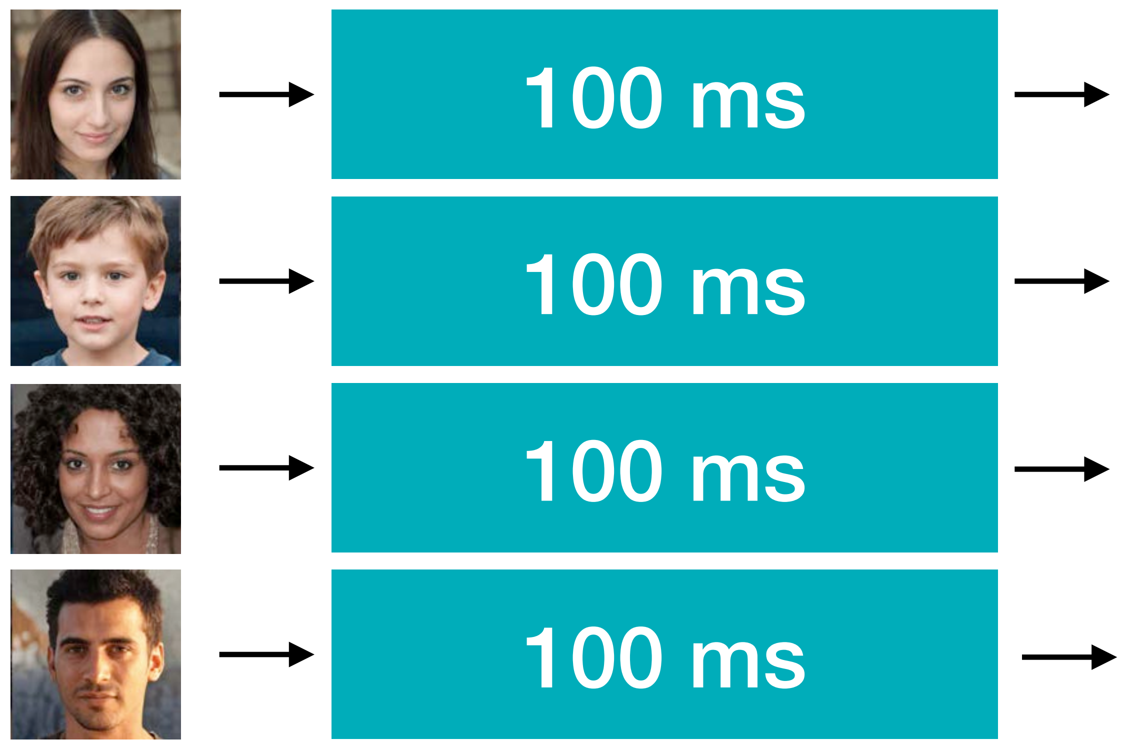
Latency vs. Throughput

- Does higher throughput translate to lower latency? Why?
- Does lower latency translate to higher throughput? Why?



Design 1

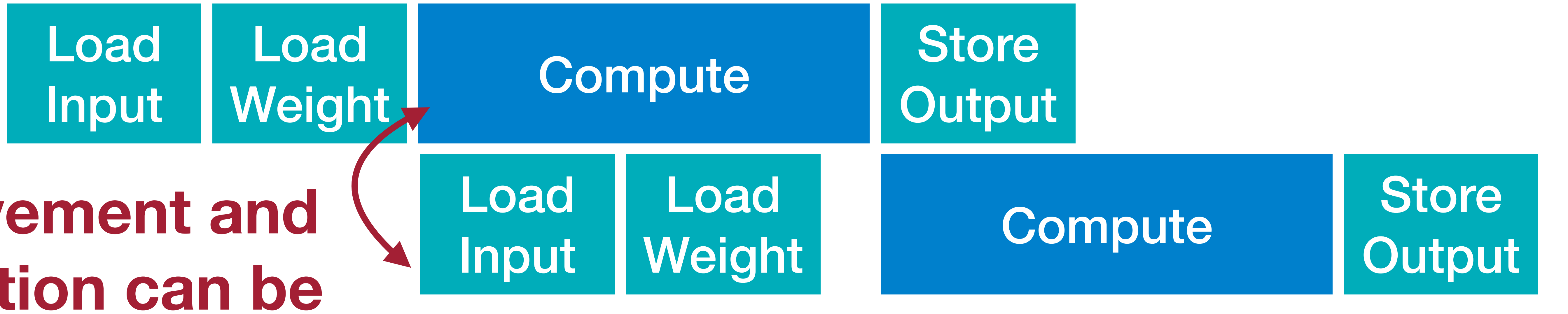
Latency: 50 ms
Throughput: 20 image/s



Design 2

Latency: 100 ms
Throughput: 40 image/s

Latency



Data Movement and Computation can be overlapped.

$$Latency \approx \max\left(T_{computation}, T_{memory}\right)$$

$$T_{\text{computation}} \approx \frac{\text{Number of Operations in Neural Network Model}}{\text{Number of Operations that Processor can Process Per Second}}$$

NN Specification

Hardware Specification

$$T_{\text{memory}} \approx T_{\text{data movement of activations}} + T_{\text{data movement of weights}}$$

$$T_{\text{data movement of weights}} \approx \frac{\text{Neural Network Model Size}}{\text{Memory Bandwidth of Processor}}$$

NN Specification

Hardware Specification

$$T_{\text{data movement of activations}} \approx \frac{\text{Input Activation Size + Output Activation Size}}{\text{Memory Bandwidth of Processor}}$$

NN Specification

Hardware Specification

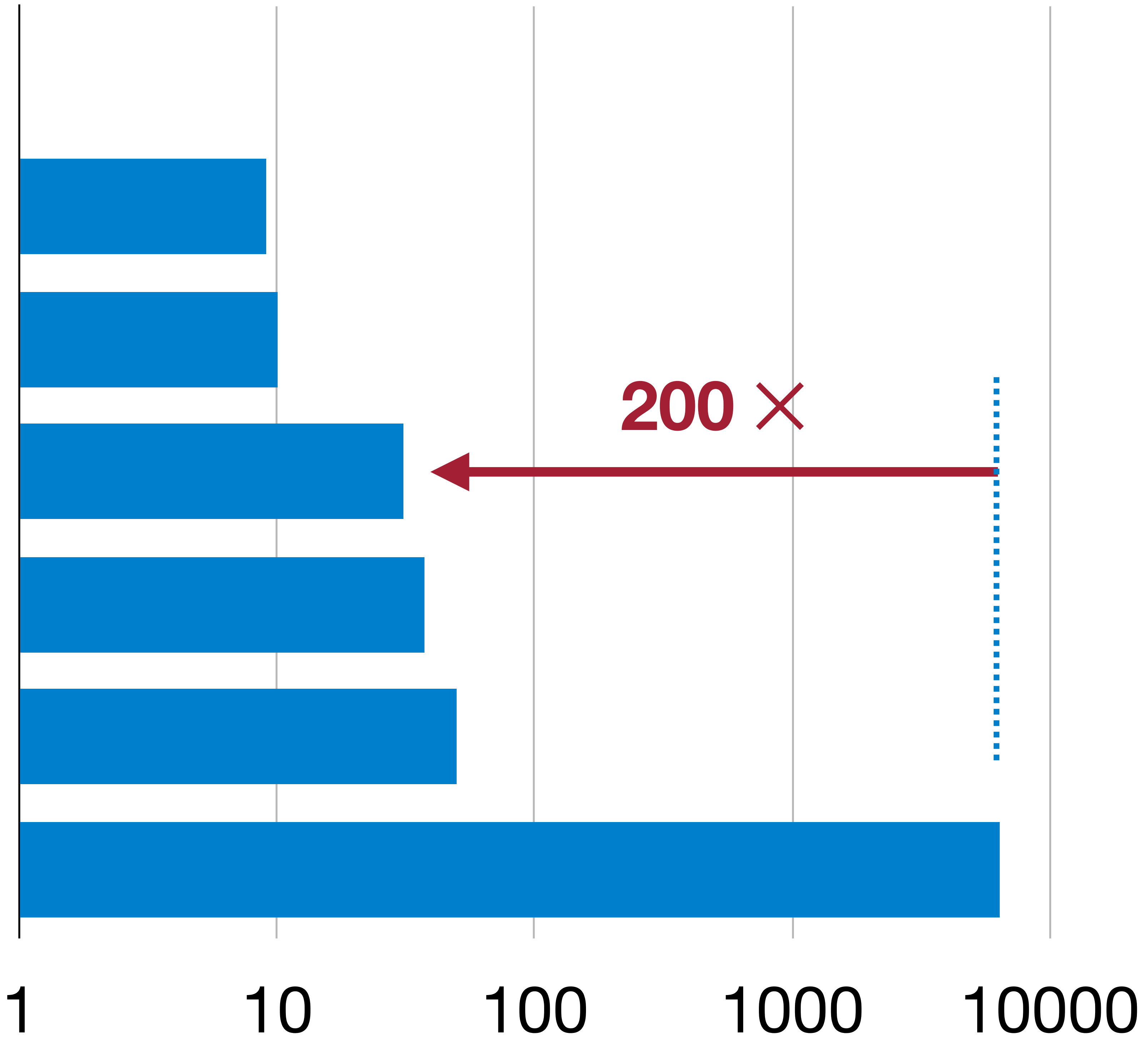
Energy Consumption

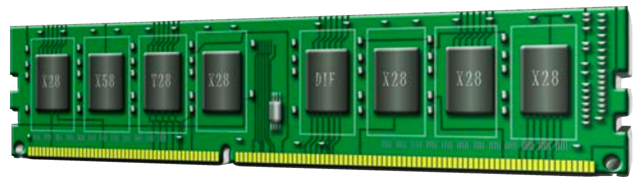
Data movement → more memory reference → more energy

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640

Rough Energy Cost For Various Operations in 45nm 0.9V

Relative Energy Cost



1  = 200 ×+

This image is in the public domain

Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014]

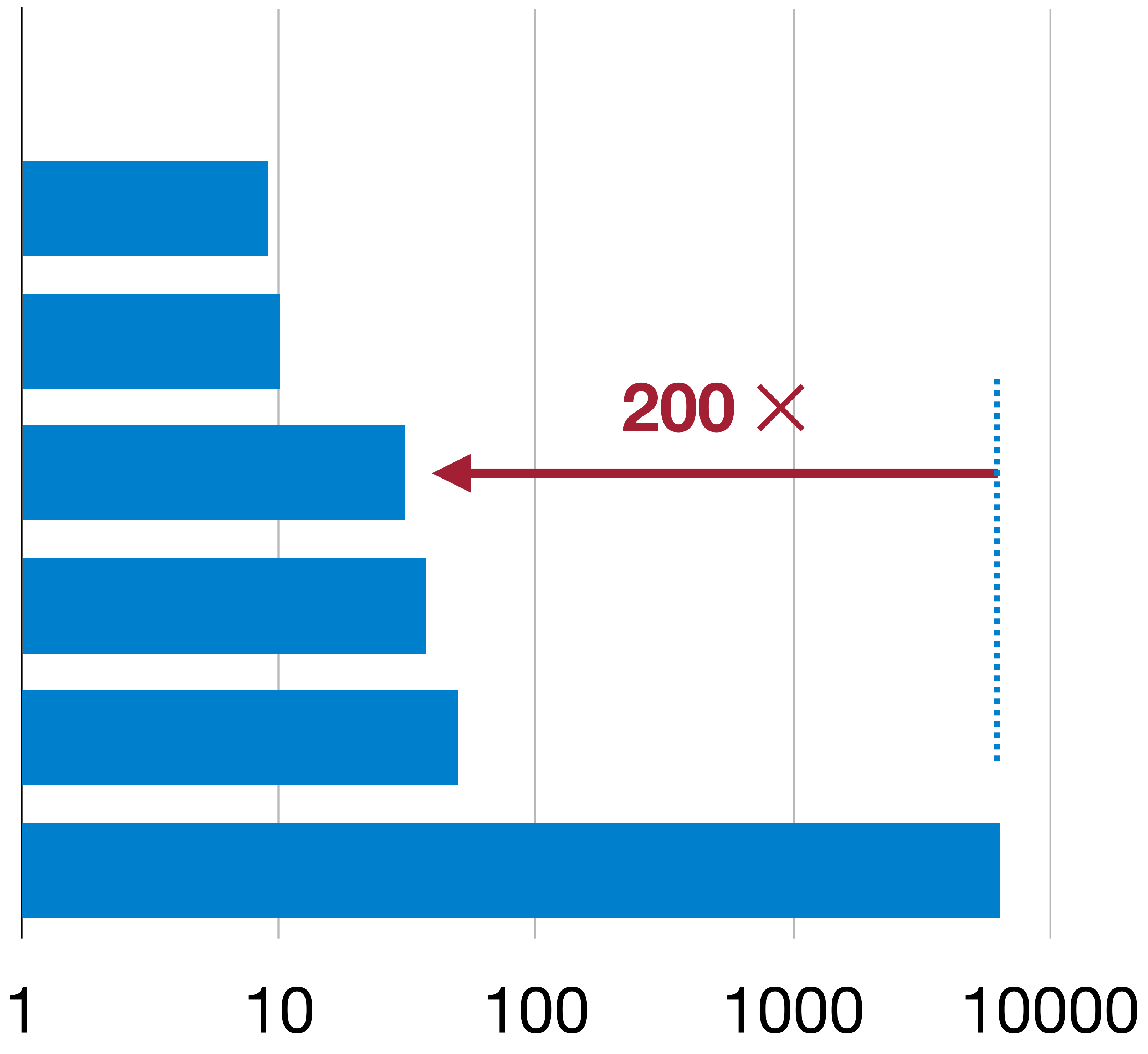
Energy Consumption

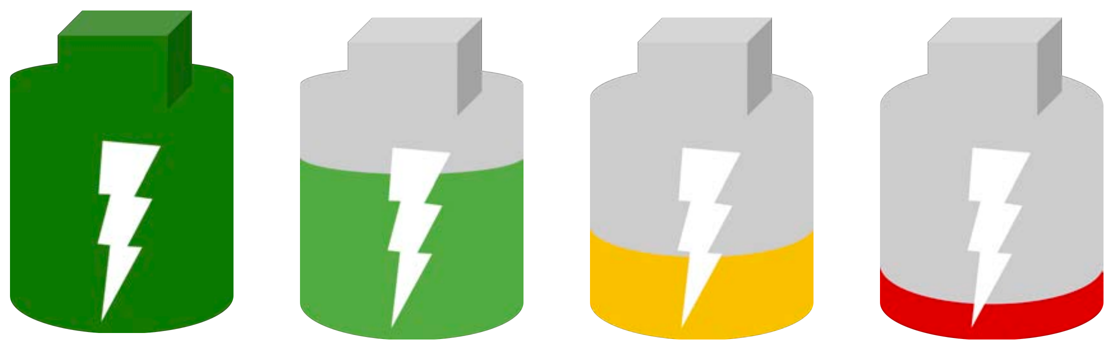
Data movement → more memory reference → more energy

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640

Rough Energy Cost For Various Operations in 45nm 0.9V

Relative Energy Cost

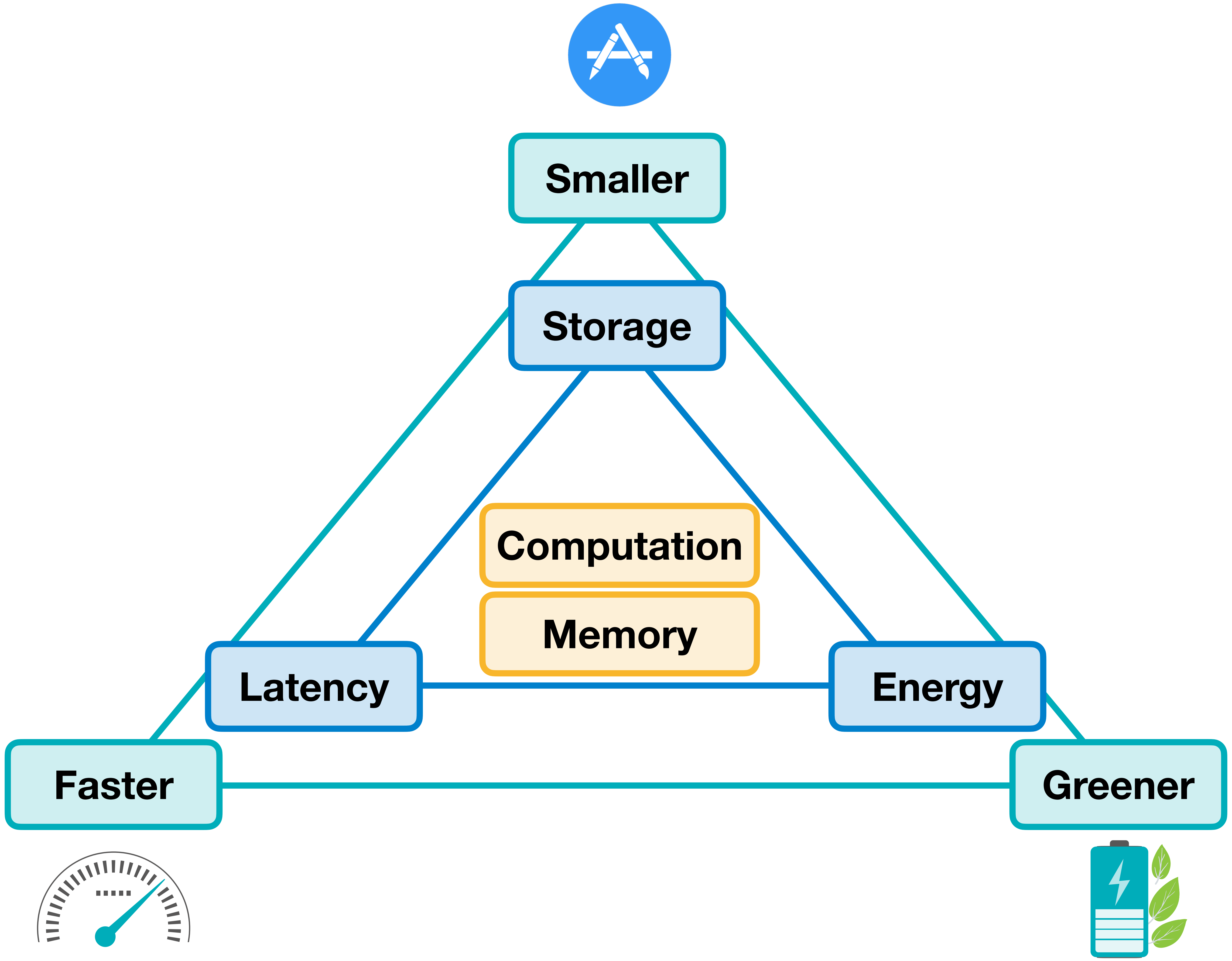




Battery images are in the public domain
Image 1, image 2, image 2, image 4

Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014]

Efficiency of Neural Networks



Efficiency Metrics

Memory-Related

#parameters

model size

total/peak #activations

Computation-Related

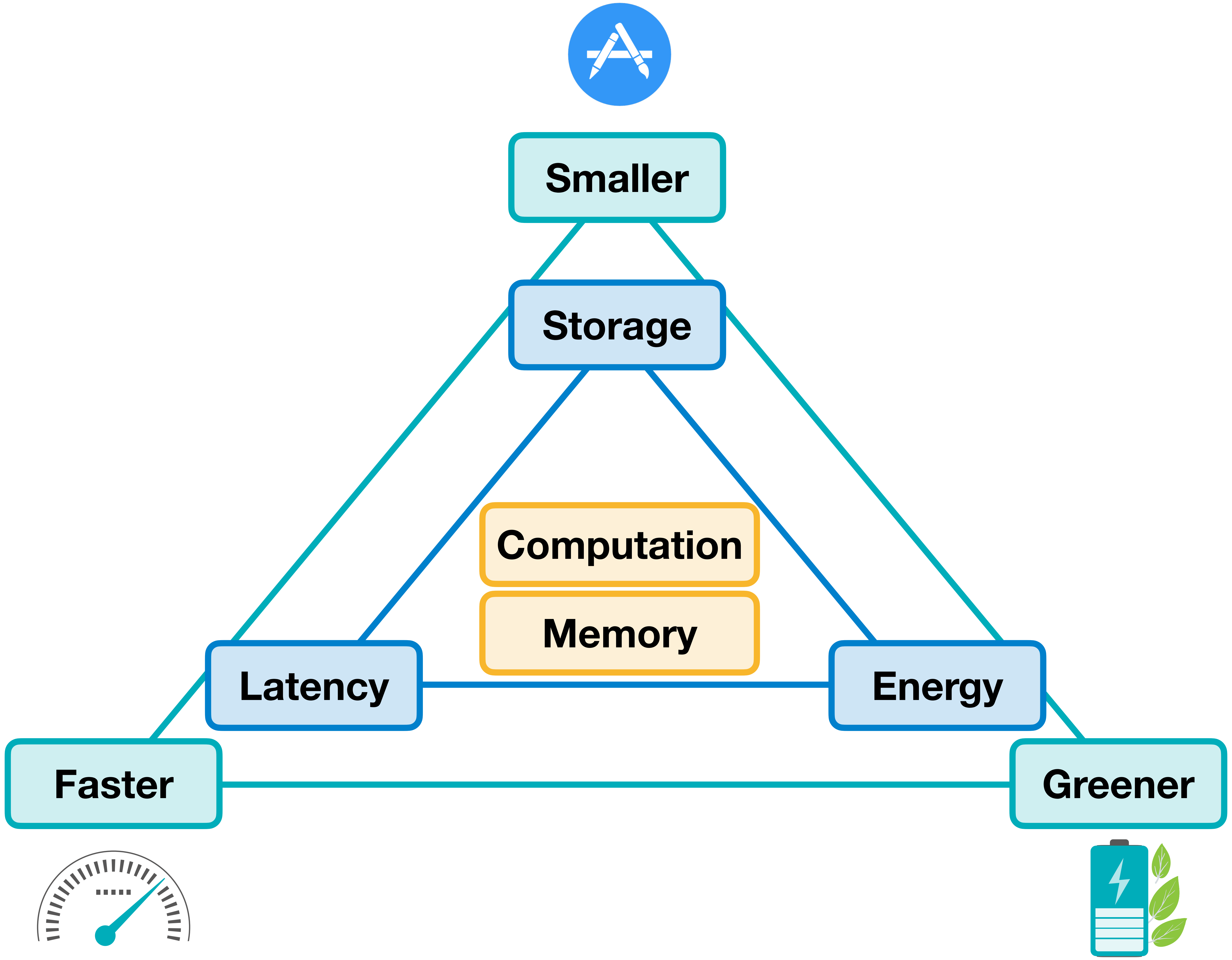
MAC

FLOP, FLOPS

OP, OPS

Image source: 1

Efficiency of Neural Networks



Efficiency Metrics

Memory-Related

#parameters

model size

total/peak #activations

Computation-Related

MAC

FLOP, FLOPS

OP, OPS

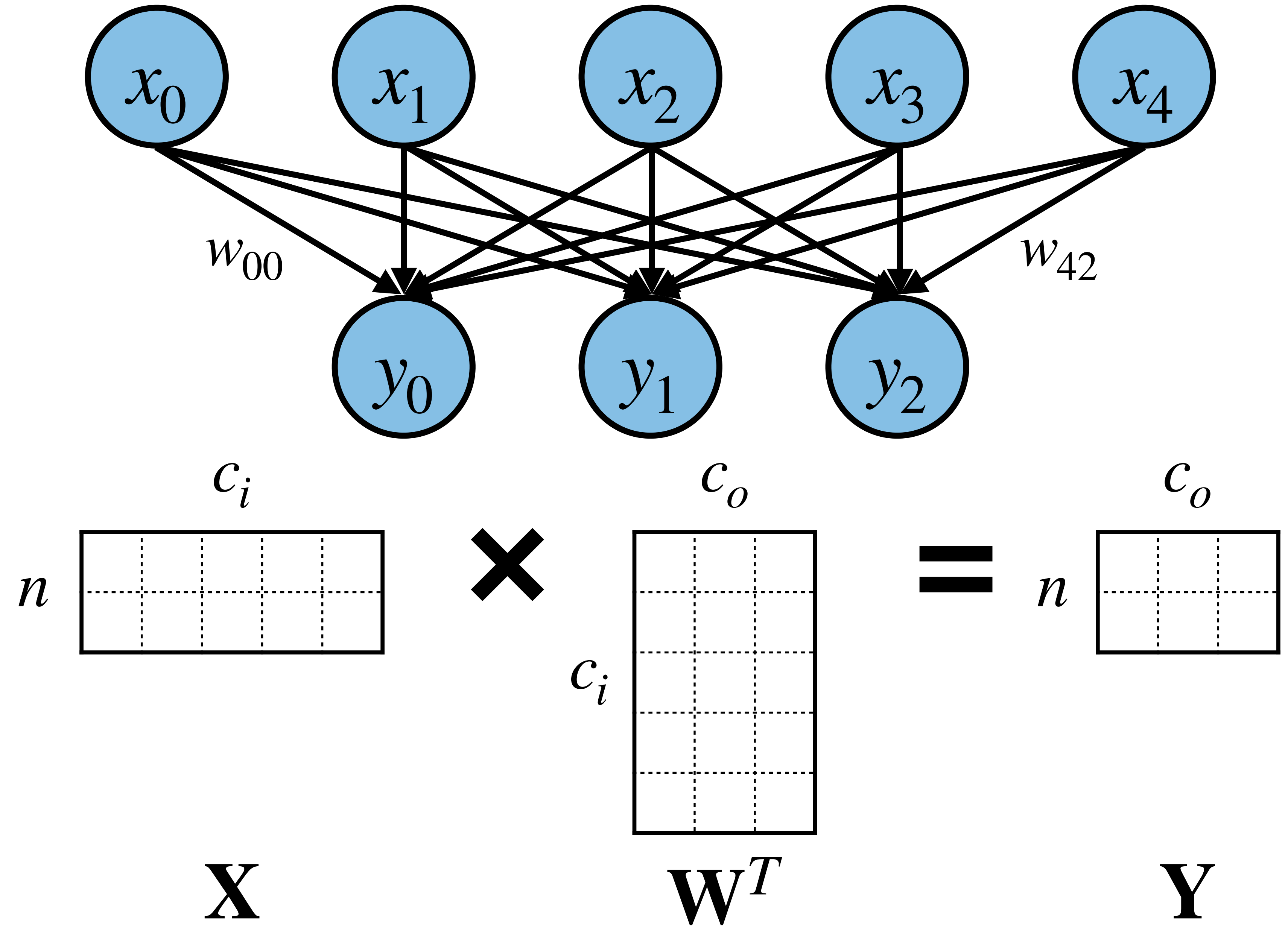
Image source: 1

Number of Parameters (#Parameters)

- #Parameters is the parameter (synapse/weight) count of the given neural network, *i.e.*, the number of elements in the weight tensors.

Number of Parameters (#Parameters)

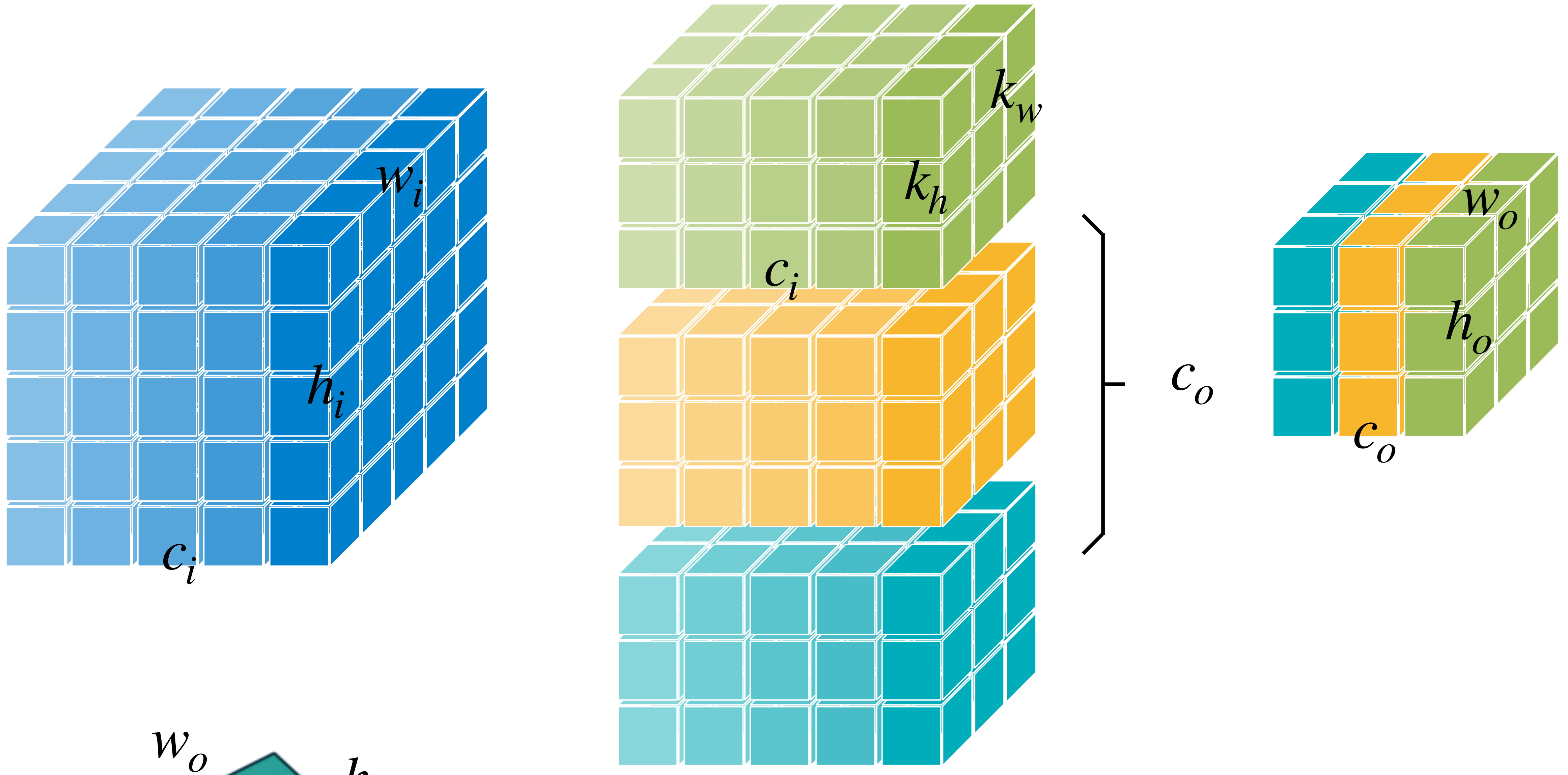
Layer	#Parameters (bias is ignored)
Linear Layer	$c_o \cdot c_i$
Convolution	
Grouped Convolution	
Depthwise Convolution	



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

Number of Parameters (#Parameters)

Layer	#Parameters (bias is ignored)
Linear Layer	$c_o \cdot c_i$
Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w$
Grouped Convolution	
Depthwise Convolution	



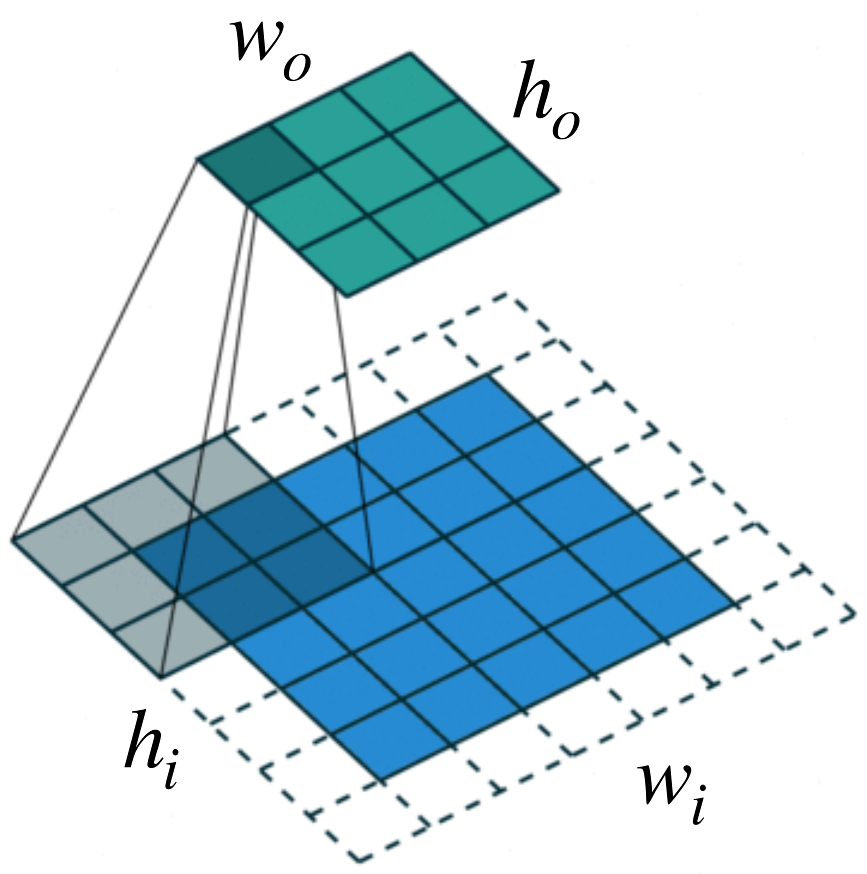


Image source: 1

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

Number of Parameters (#Parameters)

Layer	#Parameters (bias is ignored)
Linear Layer	$c_o \cdot c_i$
Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w$
Grouped Convolution	$c_o/g \cdot c_i/g \cdot k_h \cdot k_w \cdot g$ $= c_o \cdot c_i \cdot k_h \cdot k_w/g$
Depthwise Convolution	

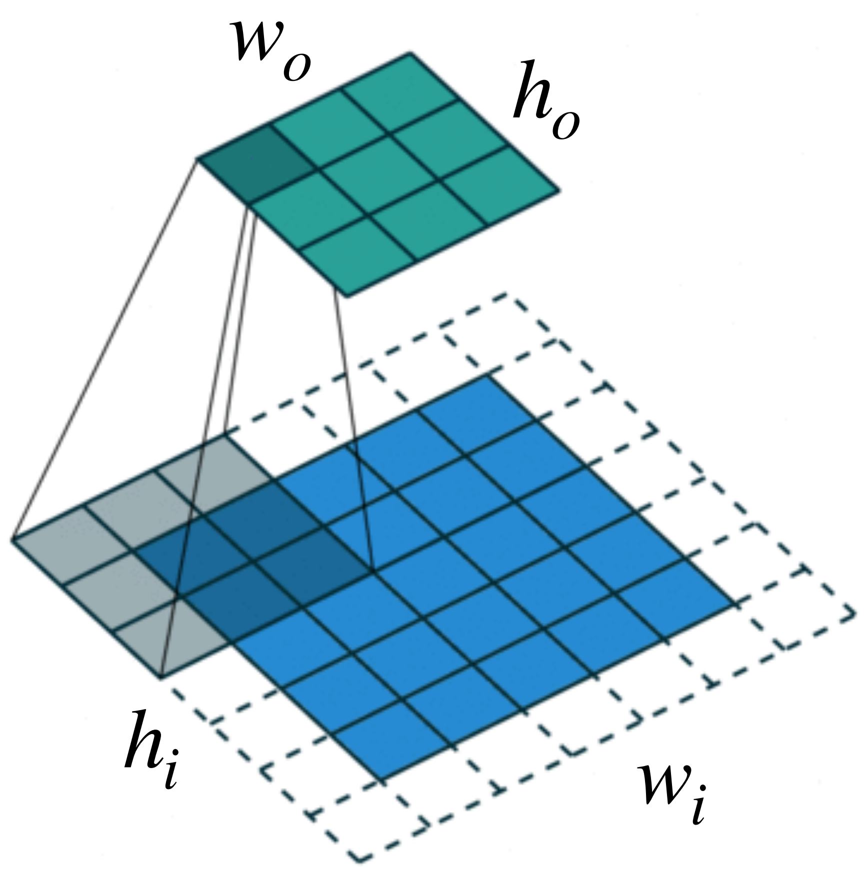
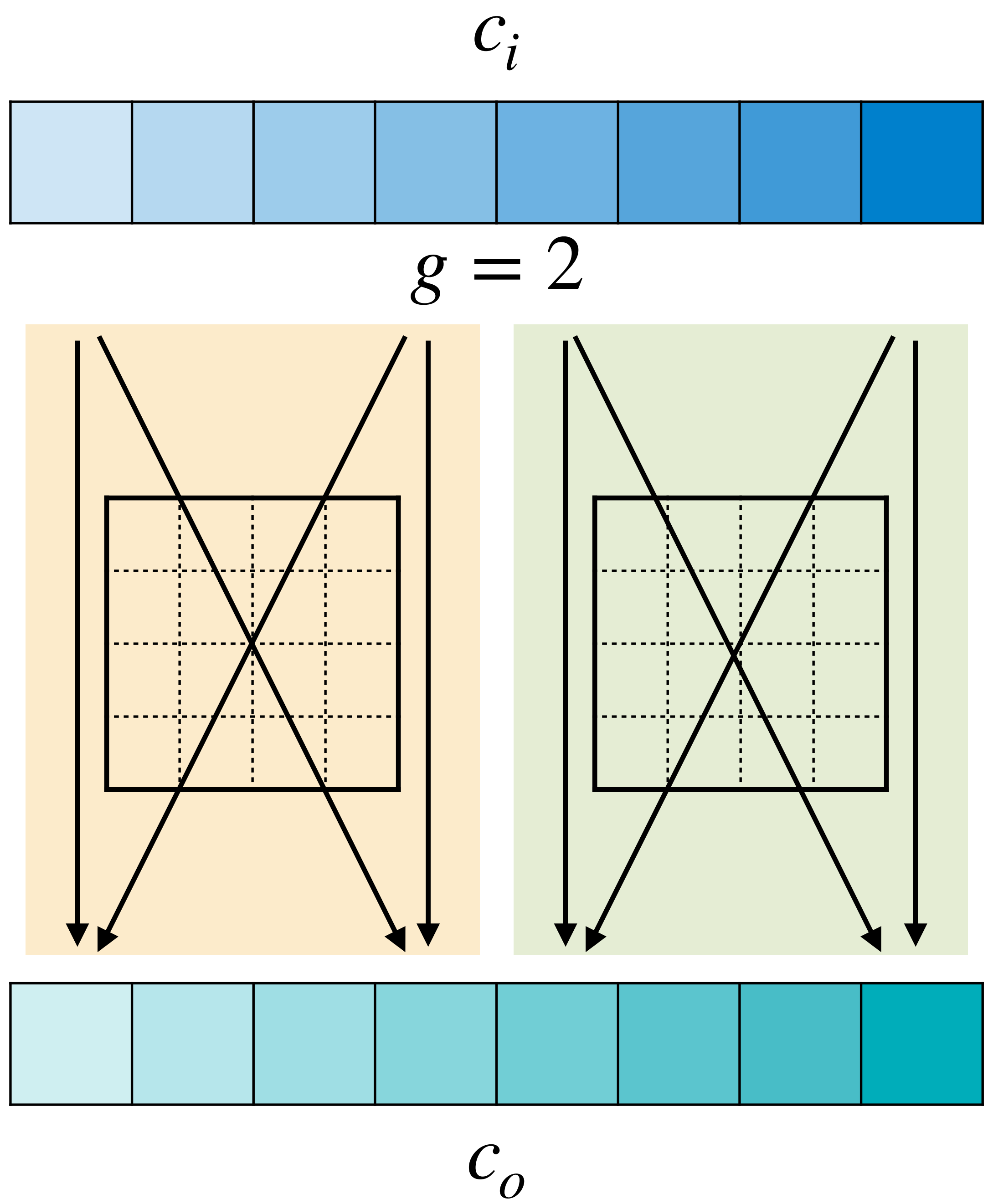


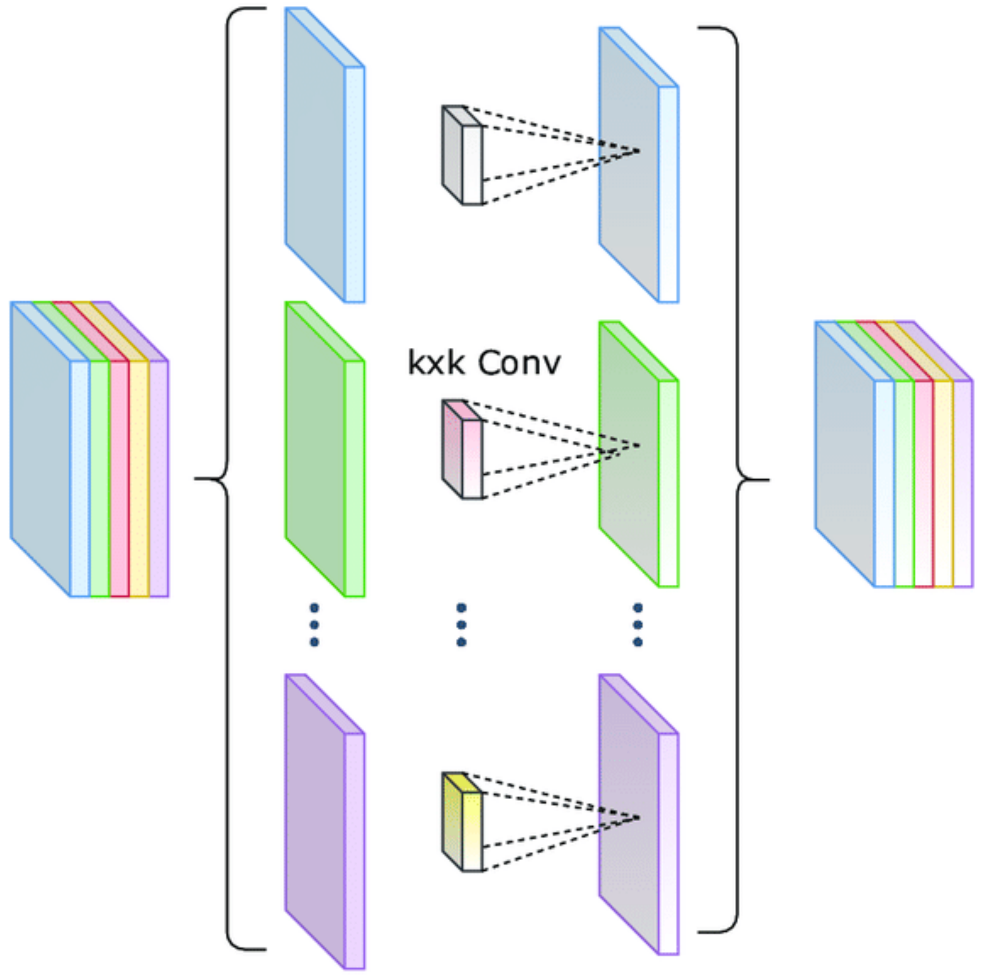
Image source: 1



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

Number of Parameters (#Parameters)

Layer	#Parameters (bias is ignored)
Linear Layer	$c_o \cdot c_i$
Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w$
Grouped Convolution	$c_o/g \cdot c_i/g \cdot k_h \cdot k_w \cdot g$ $= c_o \cdot c_i \cdot k_h \cdot k_w/g$
Depthwise Convolution	$c_o \cdot k_h \cdot k_w$



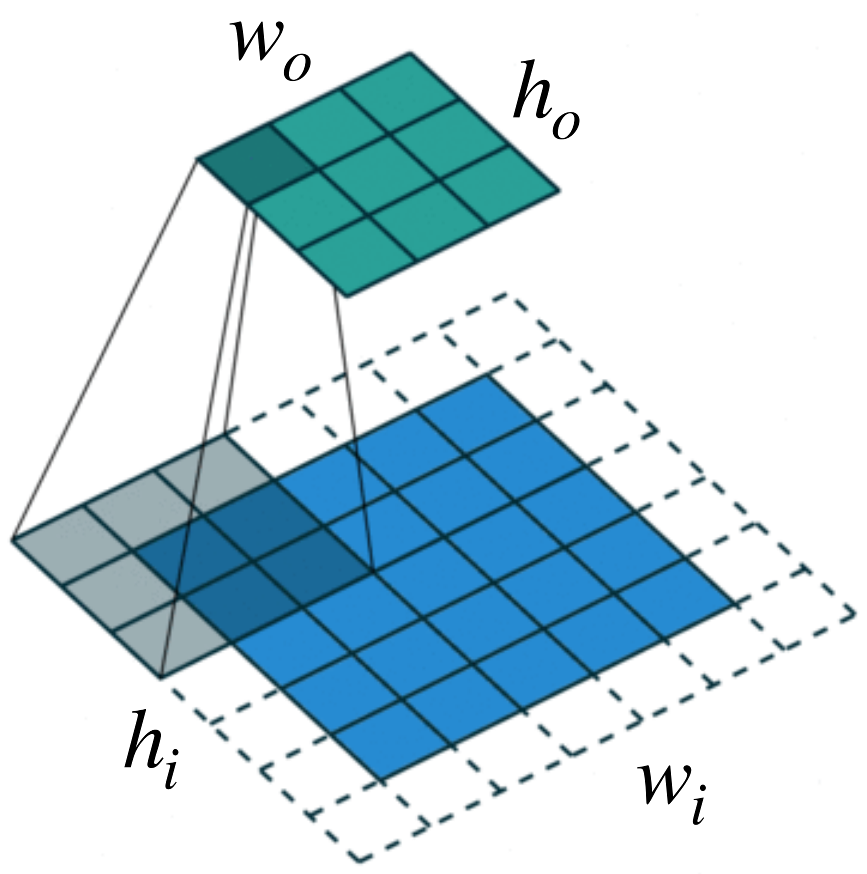
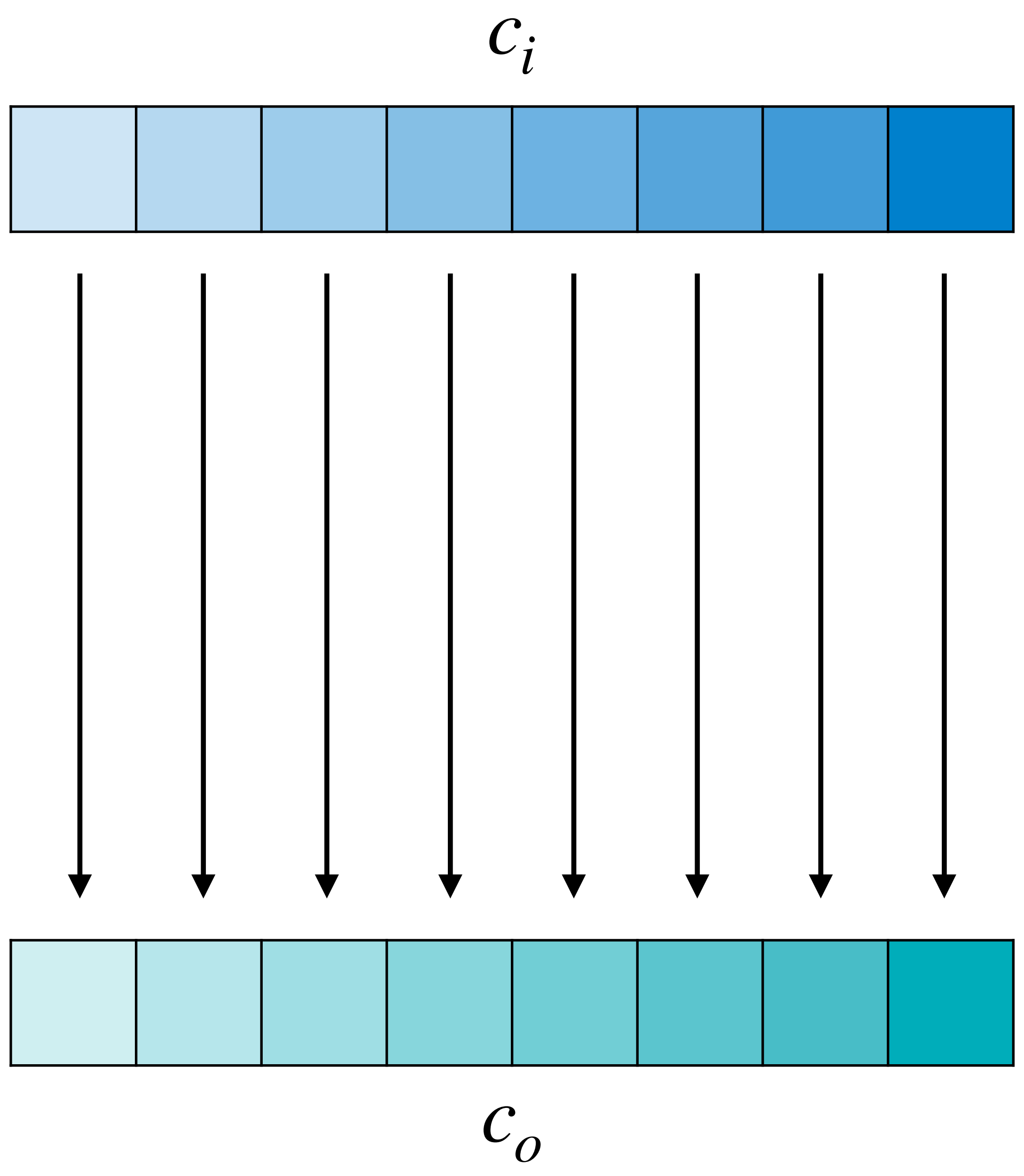


Image source: 1



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

AlexNet: #Parameters

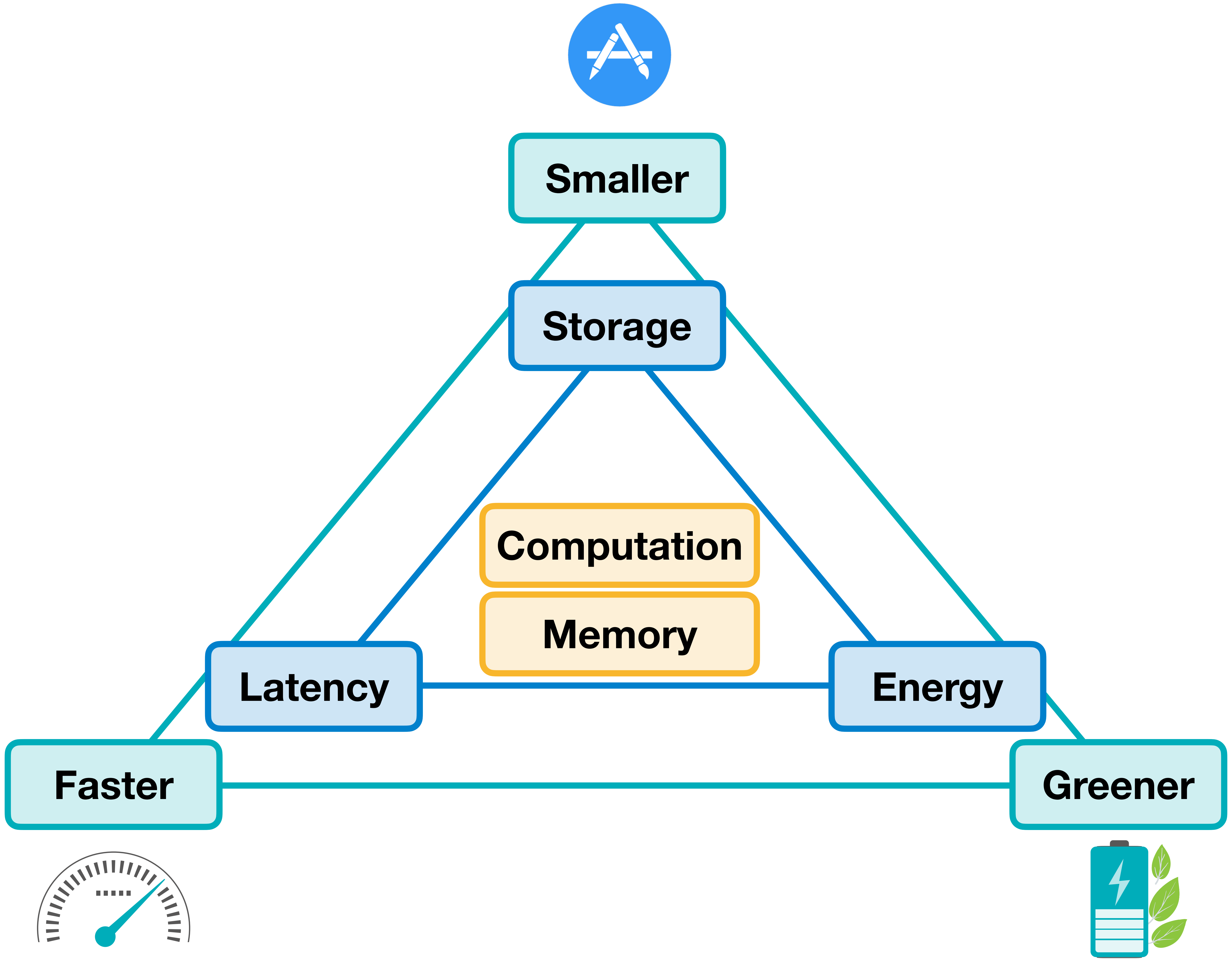
AlexNet	C × H × W	#Parameters (bias is ignored)
Image (3×224×224)	3×224×224	
11×11 Conv, channel 96, stride 4, pad 2	96×55×55	96×3×11×11 = 24, 848
3x3 MaxPool, stride 2	96×27×27	
5x5 Conv, channel 256, pad 2, groups 2	256×27×27	256×96×5×5 / 2 = 307, 200
3x3 MaxPool, stride 2	256×13×13	
3x3 Conv, channel 384, pad 1	384×13×13	384×256×3×3 = 884, 736
3x3 Conv, channel 384, pad 1, groups 2	384×13×13	384×384×3×3 / 2 = 663, 552
3x3 Conv, channel 256, pad 1, groups 2	256×13×13	256[illegible]×3 / 2 = 442, 368
3x3 MaxPool, stride 2	256×6×6	
Linear, channel 4096	4096	4096×(256×6×6) = 37, 748, 736
Linear, channel 4096	4096	4096×4096 = 16, 777, 216
Linear, channel 1000	1000	1000×4096 = 4, 096, 000

Layer	#Parameters
Linear Layer	$c_o \cdot c_i$
Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w$
Grouped Convolution	$c_o \cdot c_i \cdot k_h \cdot k_w / g$
Depthwise Convolution	$c_o \cdot k_h \cdot k_w$

61M in total

ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky *et al.*, NeurIPS 2012]

Efficiency of Neural Networks



Efficiency Metrics

Memory-Related

#parameters

model size

total/peak #activations

Computation-Related

MAC

FLOP, FLOPS

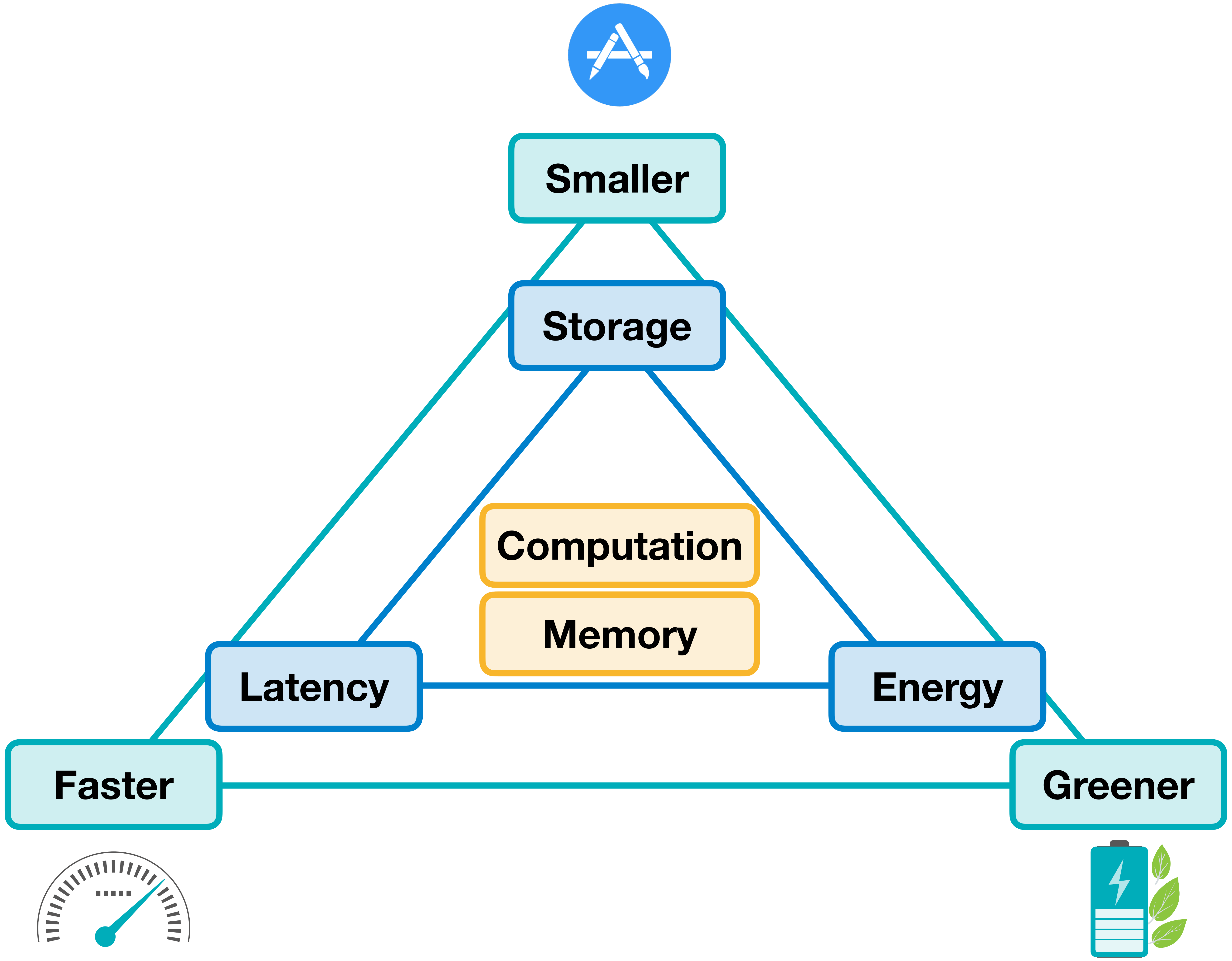
OP, OPS

Image source: 1

Model Size

- Model size measures the storage for the weights of the given neural network.
  - The common units for model size are: MB (megabyte), KB (kilobyte), bits.
- In general, if the whole neural network uses the same data type (*e.g.*, floating-point),
  - $Model \;\; Size = \#Parameters \cdot Bit\ Width$
- Example: AlexNet has 61M parameters.
  - If all weights are stored with 32-bit numbers, total storage will be about
    - 61M × 4 Bytes (32 bits) = 224 MB (224×10^6 Bytes)
  - If all weights are stored with 8-bit numbers, total storage will be about
    - 61M × 1 Byte (8 bits) = 61 MB

Efficiency of Neural Networks



Efficiency Metrics

Memory-Related

#parameters

model size

total/peak #activations

Computation-Related

MAC

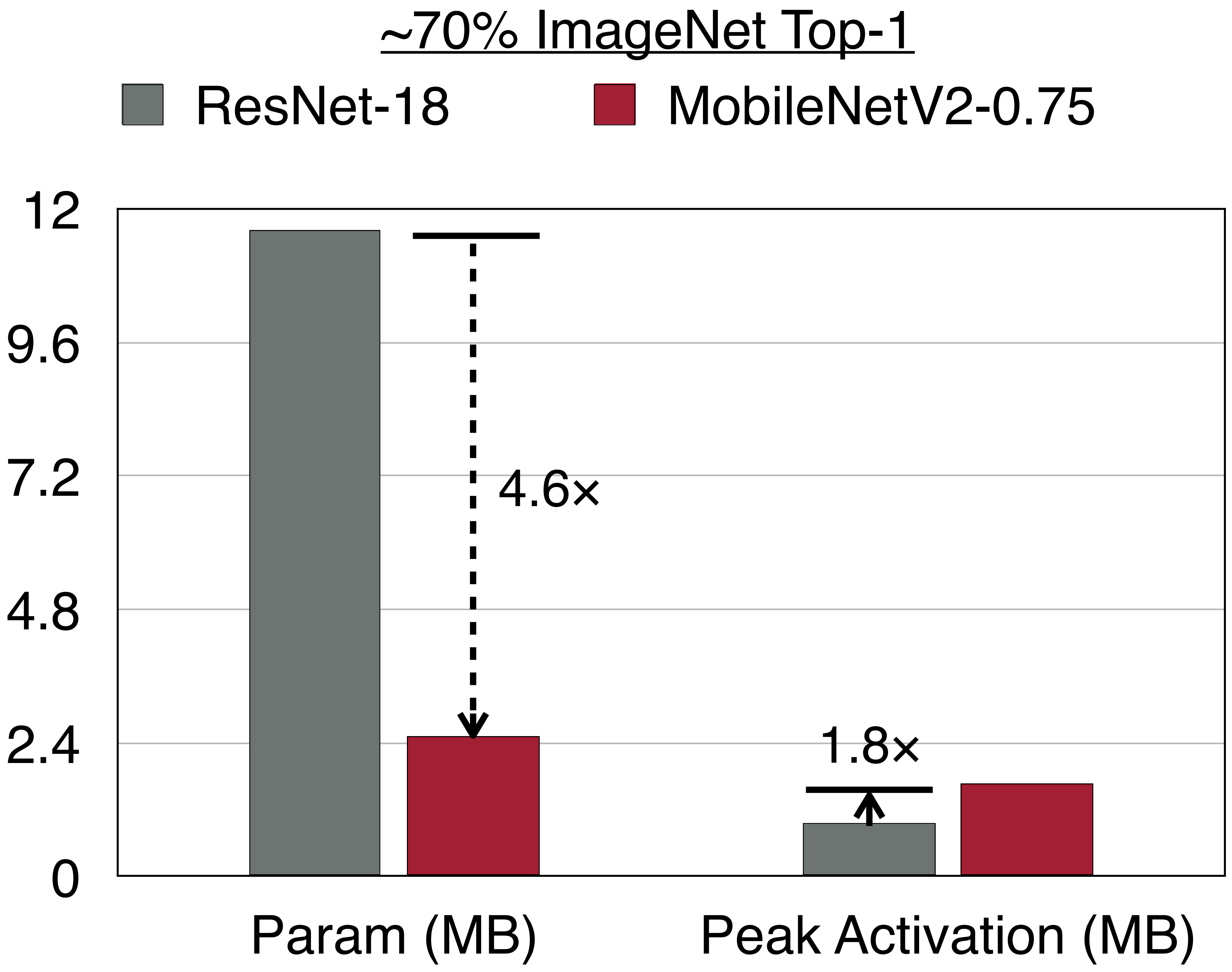
FLOP, FLOPS

OP, OPS

Image source: 1

Number of Activations (#Activations)

#Activation is the memory bottleneck in inference on IoT, not #Parameters.

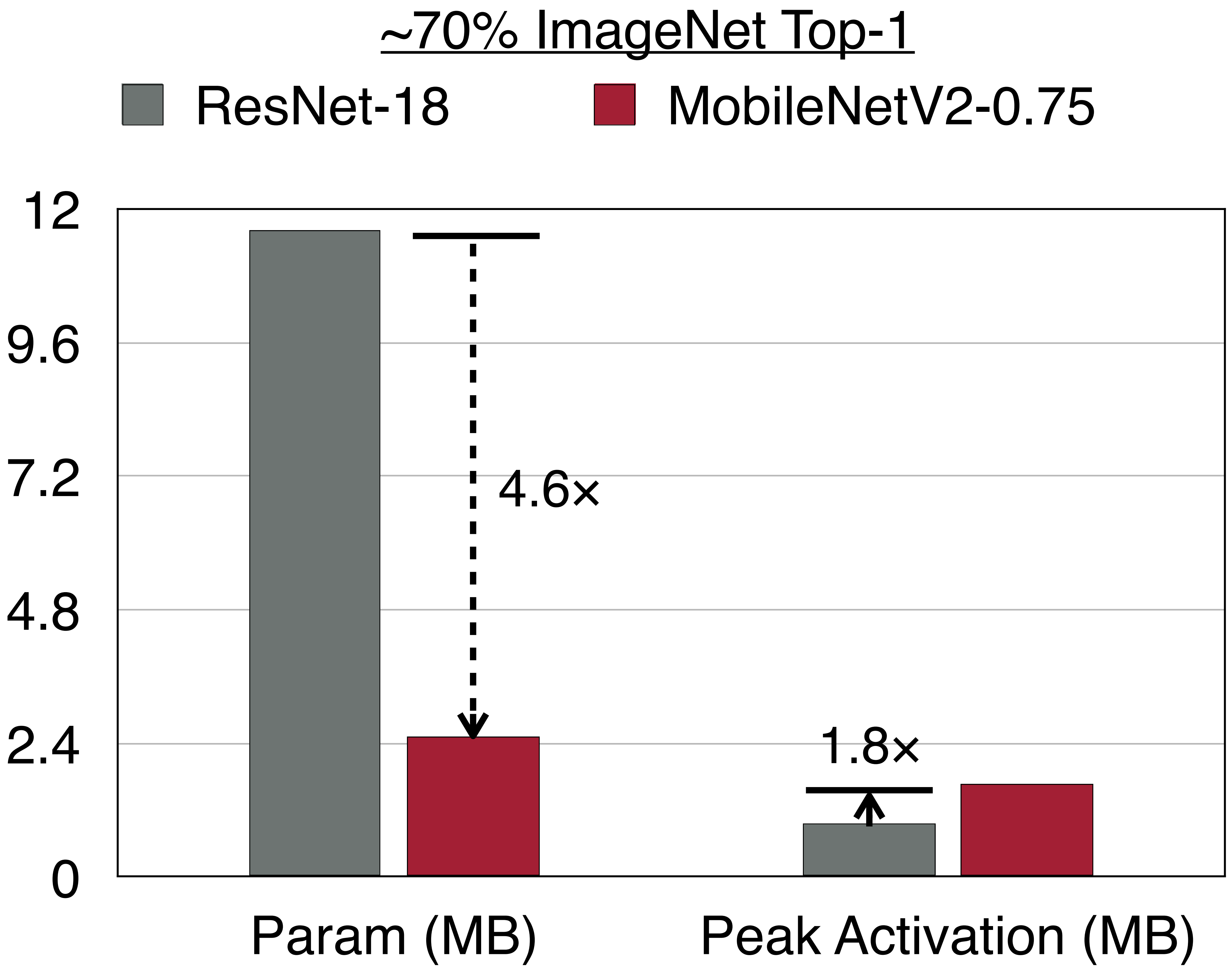


*** All parameters and activations are Integer numbers (8 bits).**

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

Number of Activations (#Activations)

#Activation didn't improve from ResNet to MobileNet-v2

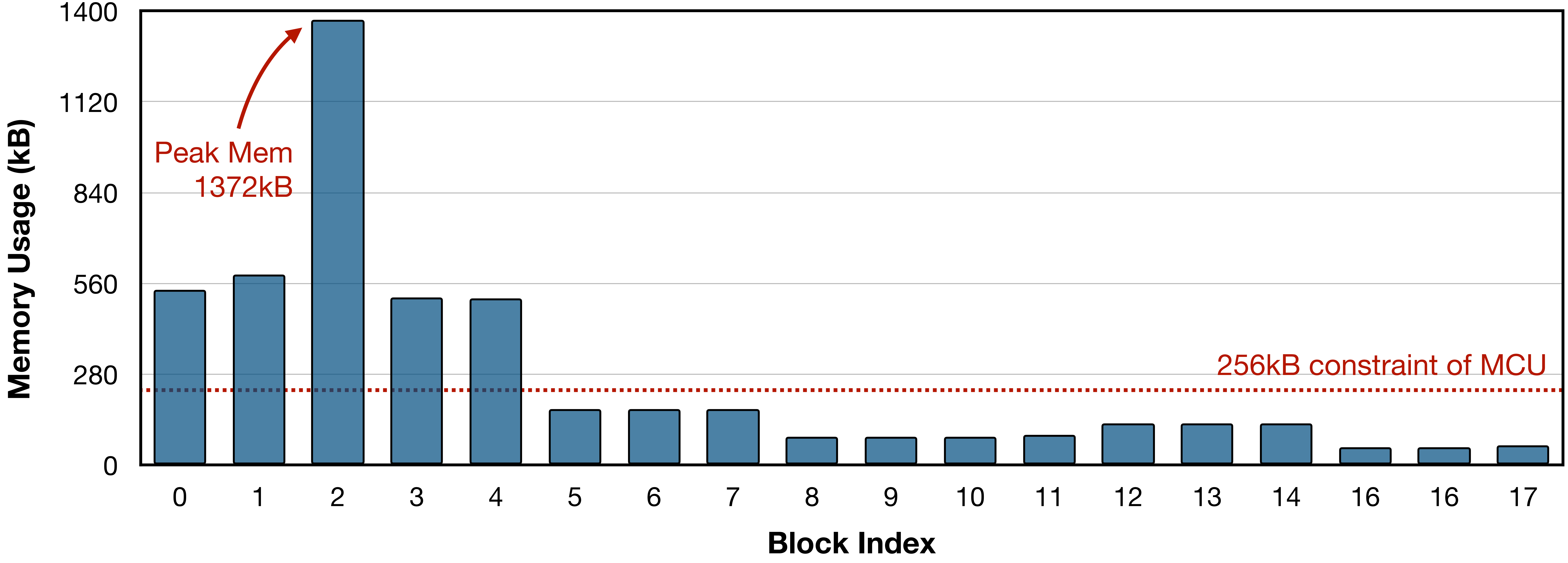


*** All parameters and activations are Integer numbers (8 bits).**

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

Number of Activations (#Activations)

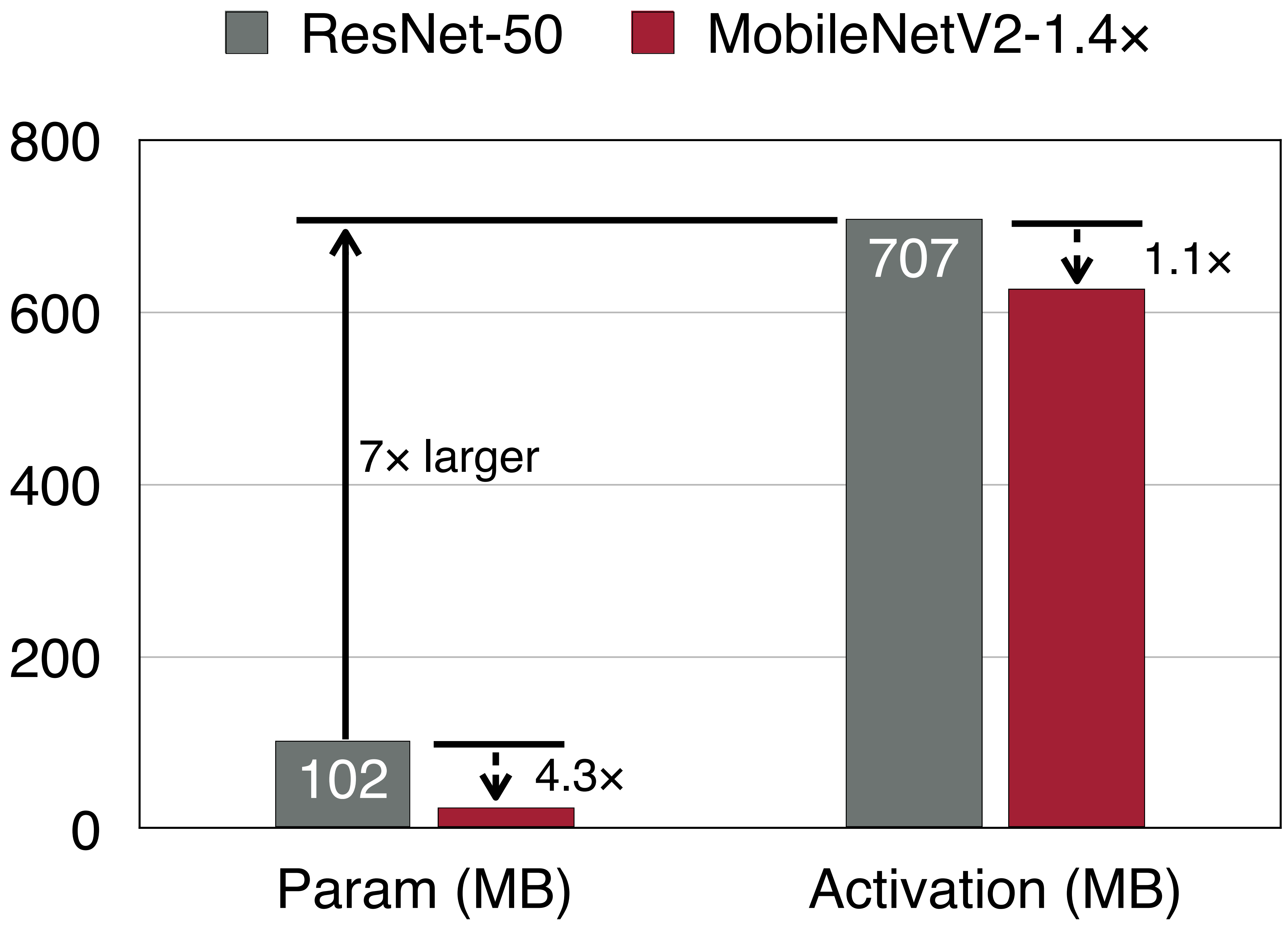
Imbalanced memory distribution of MobileNetV2



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

Number of Activations (#Activations)

#Activation is the memory bottleneck in training, not #Parameters.



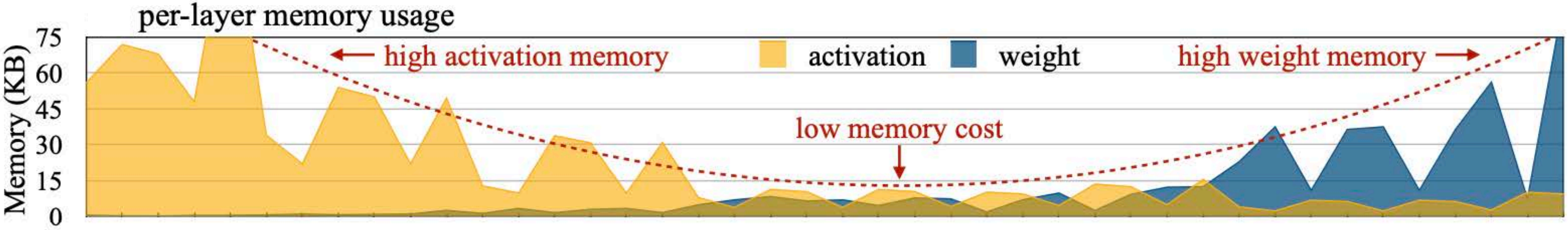
The main memory bottleneck does not improve much.

*** All parameters and activations are Floating-Point numbers (32 bits).**

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

Number of Activations (#Activations)

Activation and weight memory distribution of MCUNet



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

AlexNet: #Activations

AlexNet	C × H × W	
Image (3×224×224)	3×224×224	=150,528
11×11 Conv, channel 96, stride 4, pad 2	96×55×55	=290,400
3x3 MaxPool, stride 2	96×27×27	=69,984
5×5 Conv, channel 256, pad 2, groups 2	256×27×27	=186,624
3x3 MaxPool, stride 2	256×13×13	=43,264
3x3 Conv, channel 384, pad 1	384×13×13	=64,896
3x3 Conv, channel 384, pad 1, groups 2	384×13×13	=64,896
3x3 Conv, channel 256, pad 1, groups 2	256×13×13	=43,264
3x3 MaxPool, stride 2	256×6×6	=9,216
Linear, channel 4096	4096	=4,096
Linear, channel 4096	4096	=4,096
Linear, channel 1000	1000	=1,000

Total #Activation: 932,264

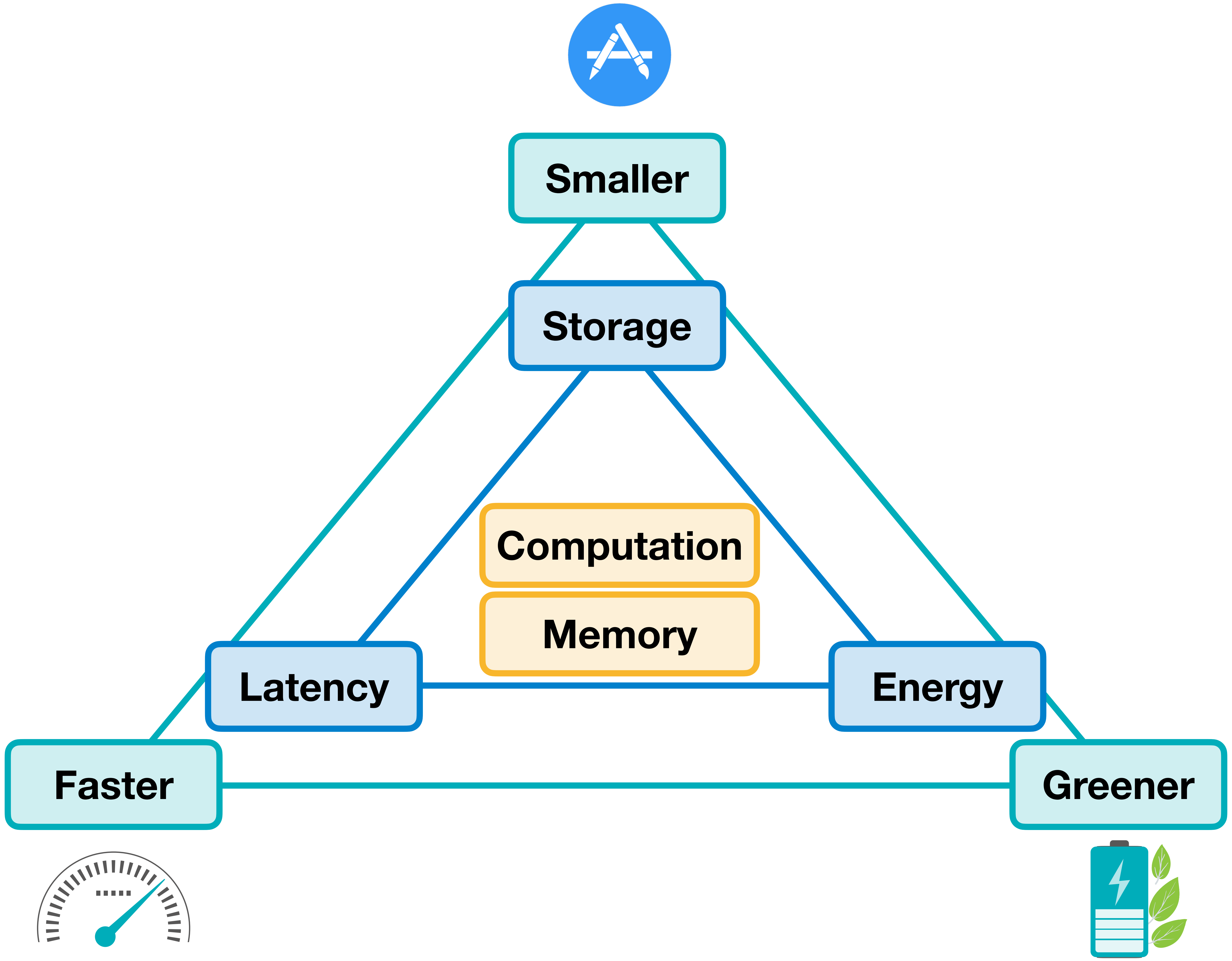
Peak #Activation:

≈ #input activation + #output activation

= 150,528 + 290,400 = 440,928

ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky *et al.*, NeurIPS 2012]

Efficiency of Neural Networks



Efficiency Metrics

Memory-Related

#parameters

model size

total/peak #activations

Computation-Related

MAC

FLOP, FLOPS

OP, OPS

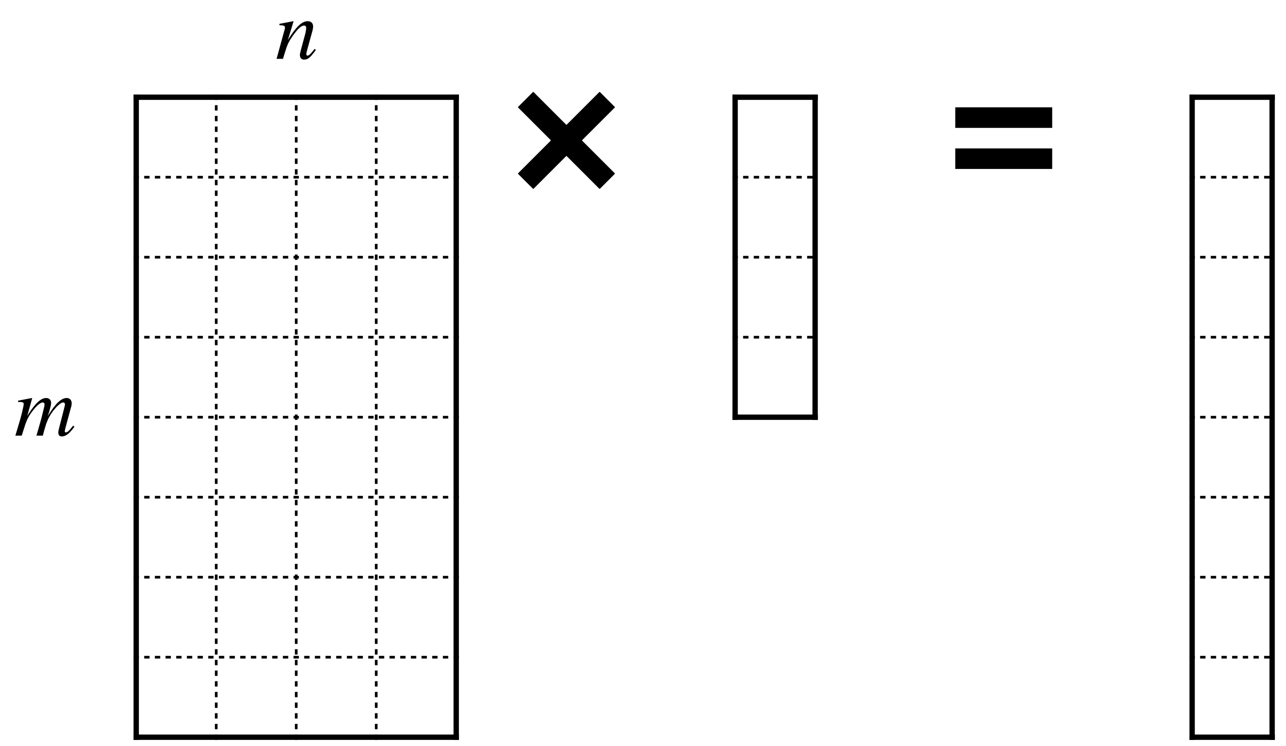
Image source: 1

Number of Multiply-Accumulate Operations

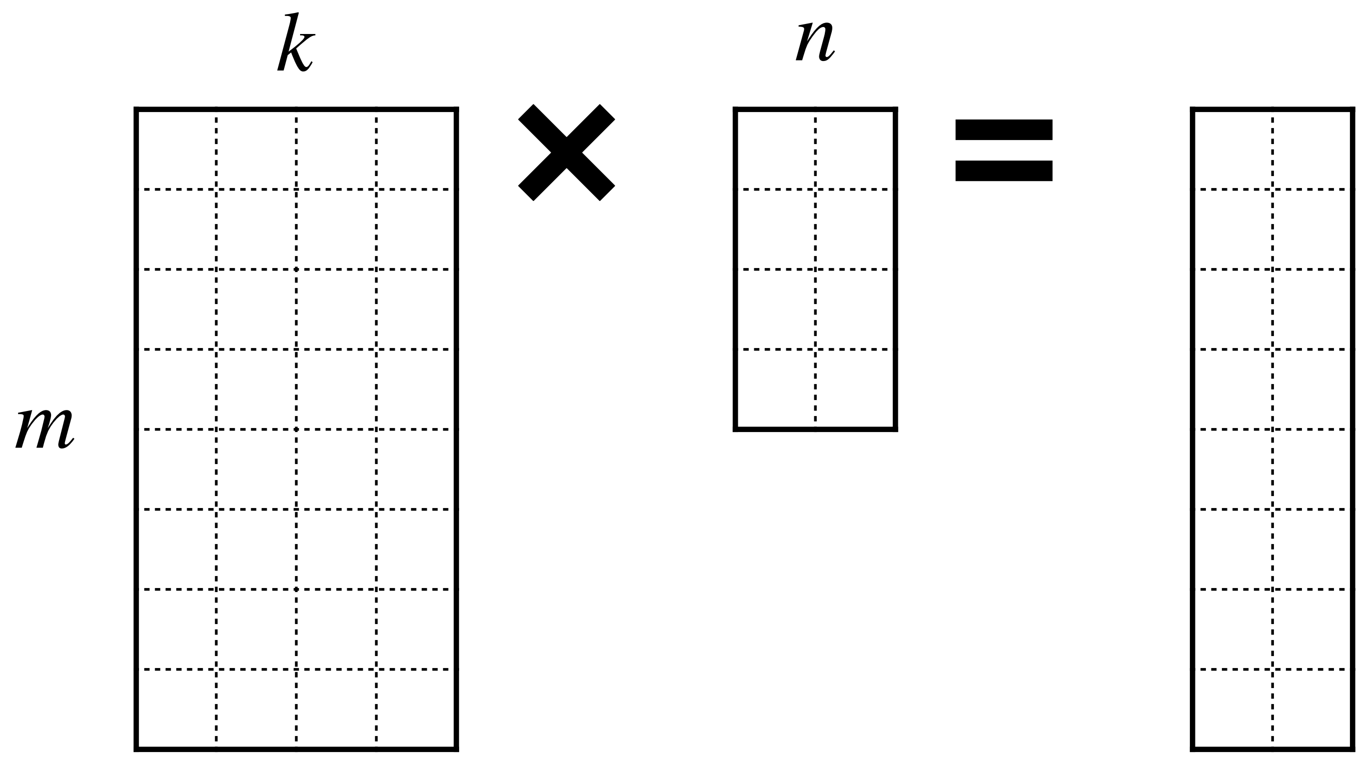
MAC

- Multiply-Accumulate operation (MAC)
  - $a \leftarrow a + b \cdot c$

- Matrix-Vector Multiplication (MV)
  - $MACs = m \cdot n$



- General Matrix-Matrix Multiplication (GEMM)
  - $MACs = m \cdot n \cdot k$

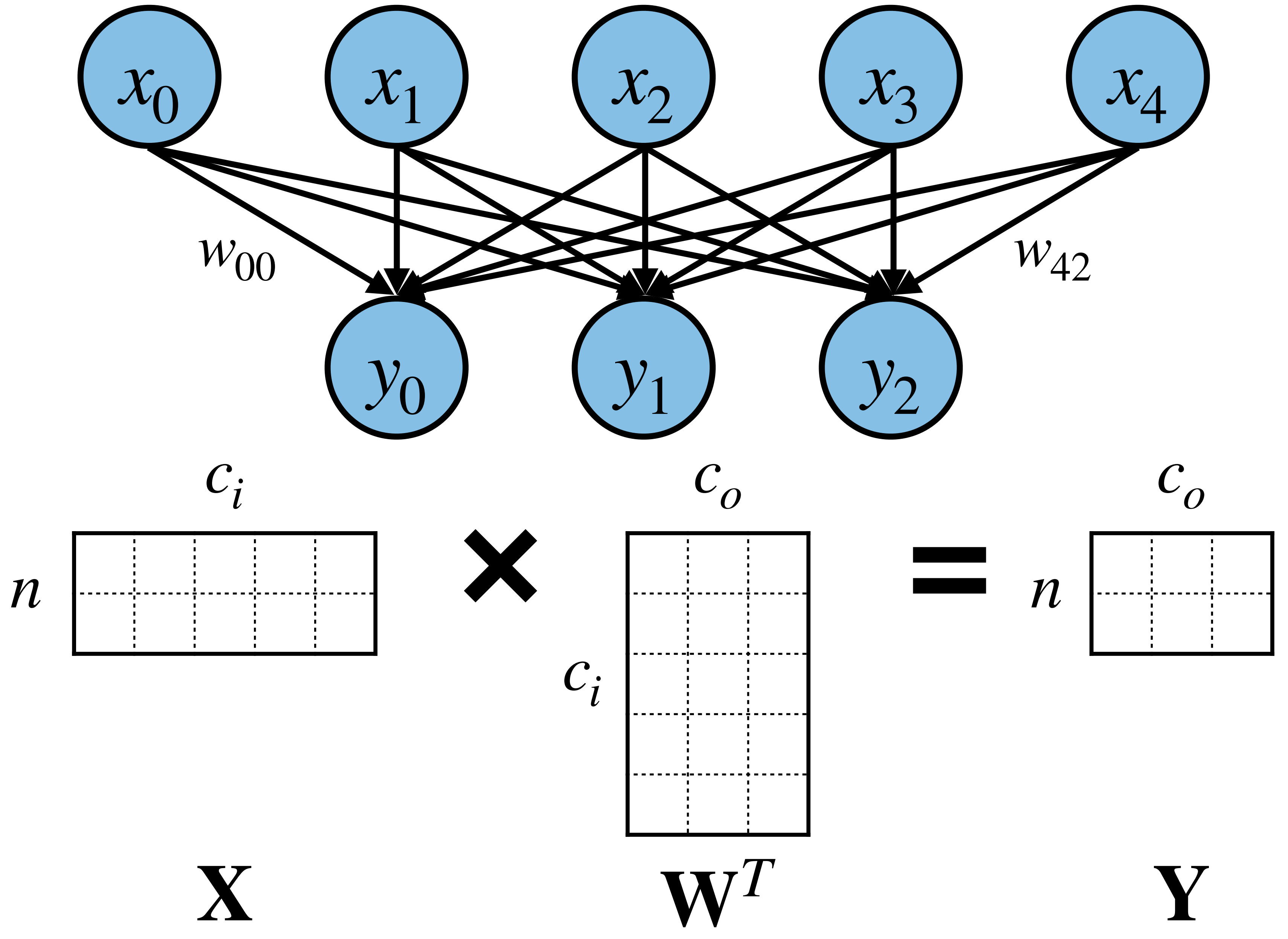


Number of Multiply-Accumulate Operations

MAC

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

* bias is ignored



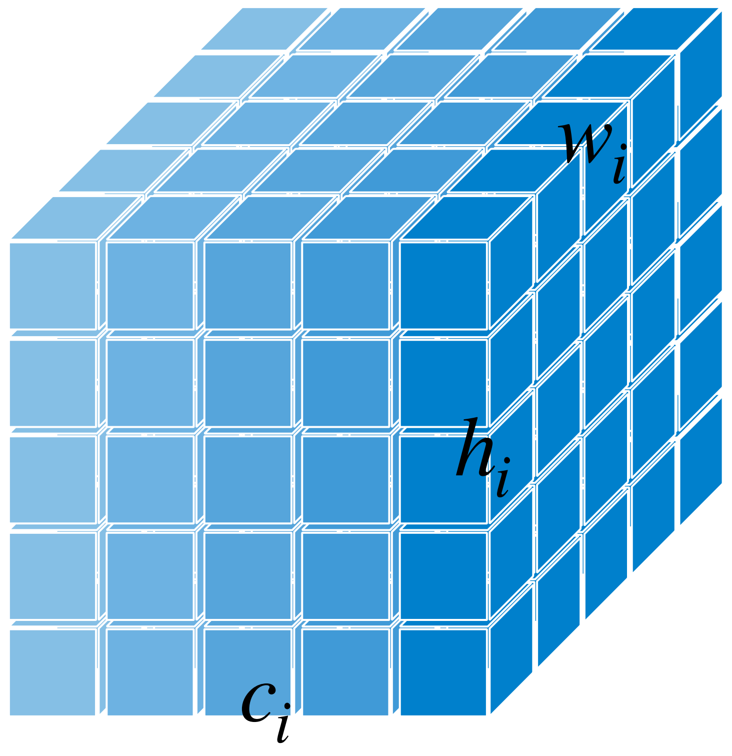
Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h, k_w	Kernel Height/Width
g	Groups

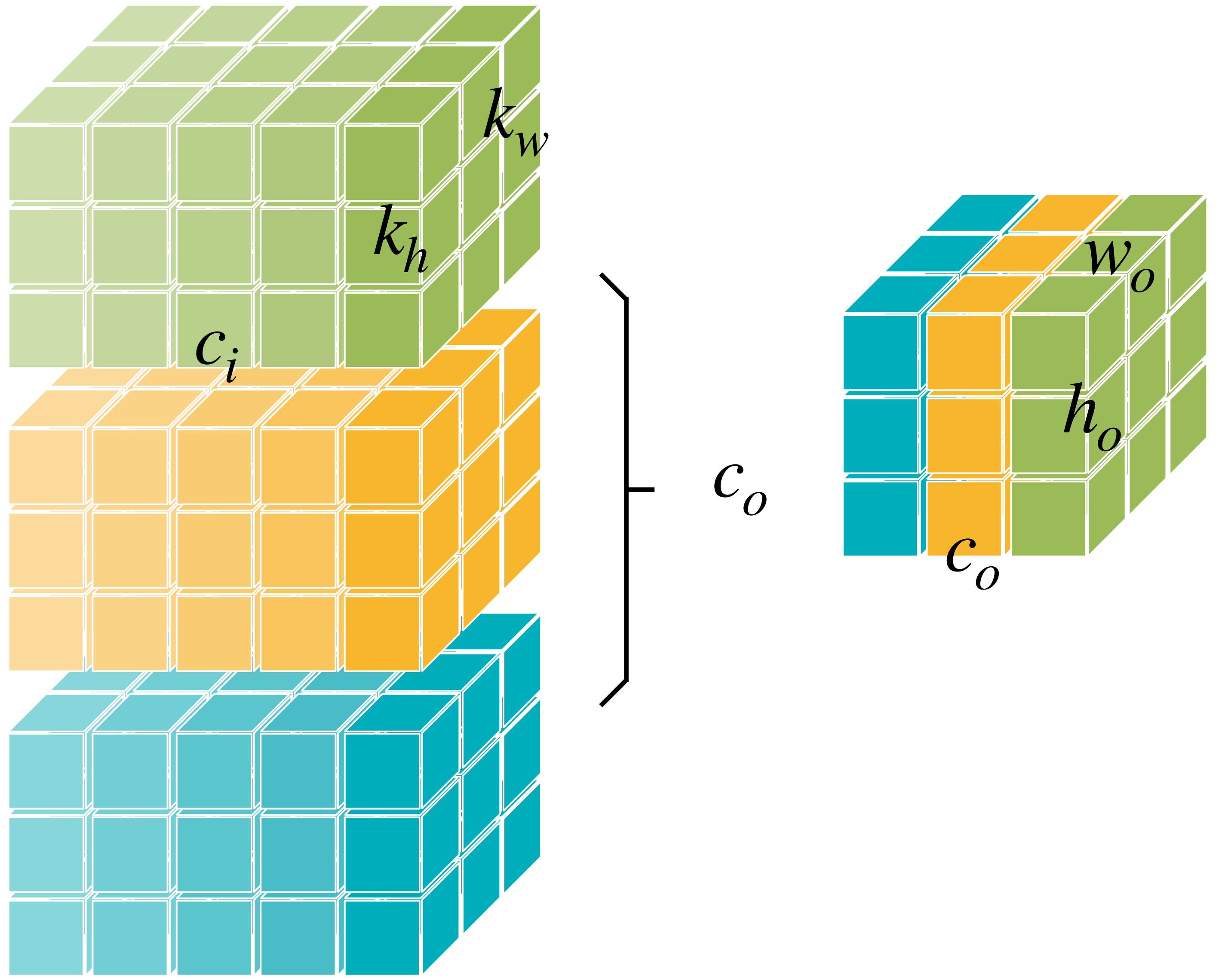
Number of Multiply-Accumulate Operations

MAC

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

* bias is ignored





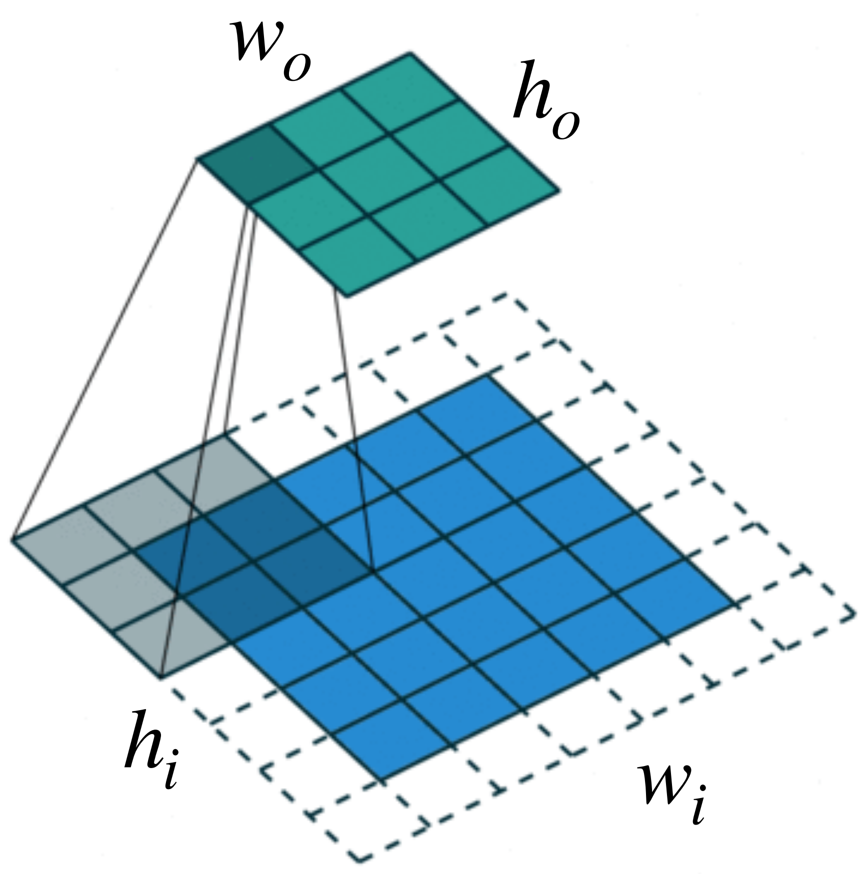


Image source: 1

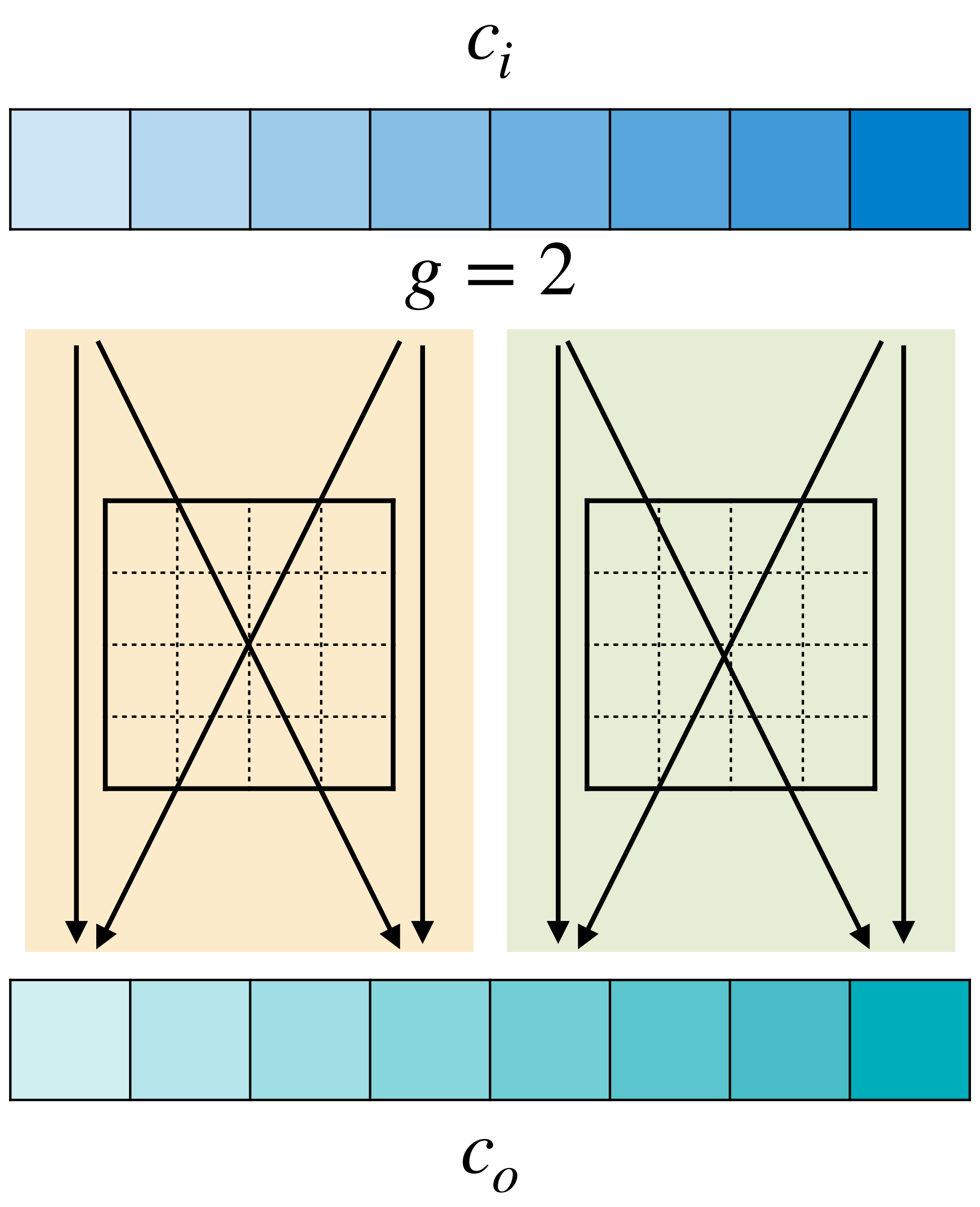
Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h, k_w	Kernel Height/Width
g	Groups

Number of Multiply-Accumulate Operations

MAC

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

* bias is ignored



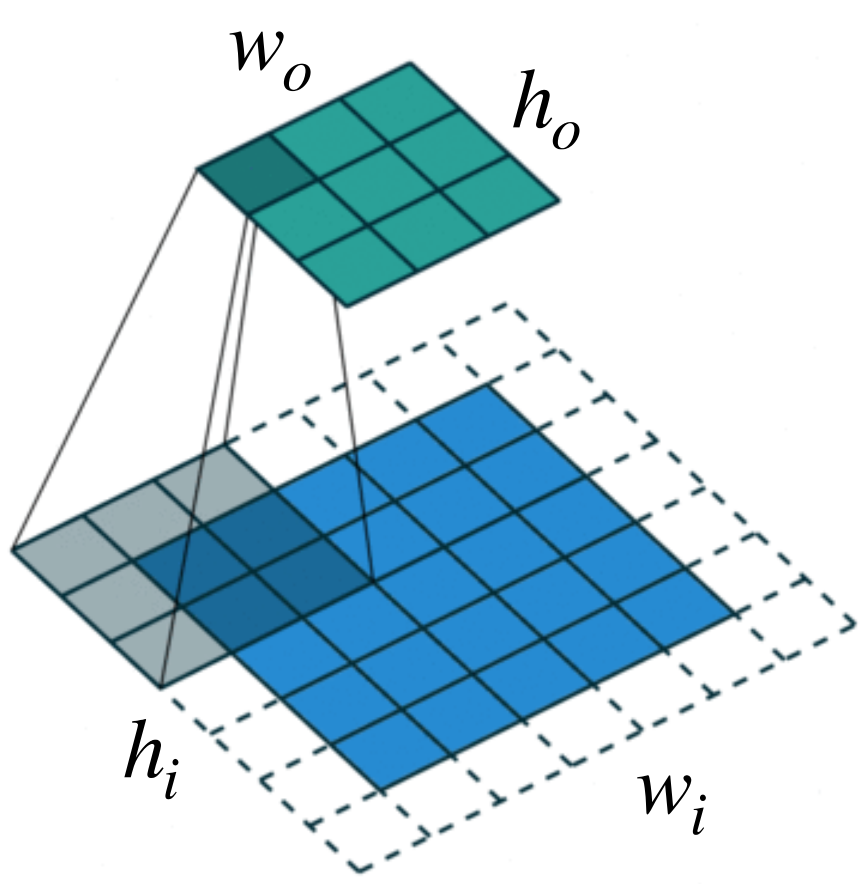


Image source: 1

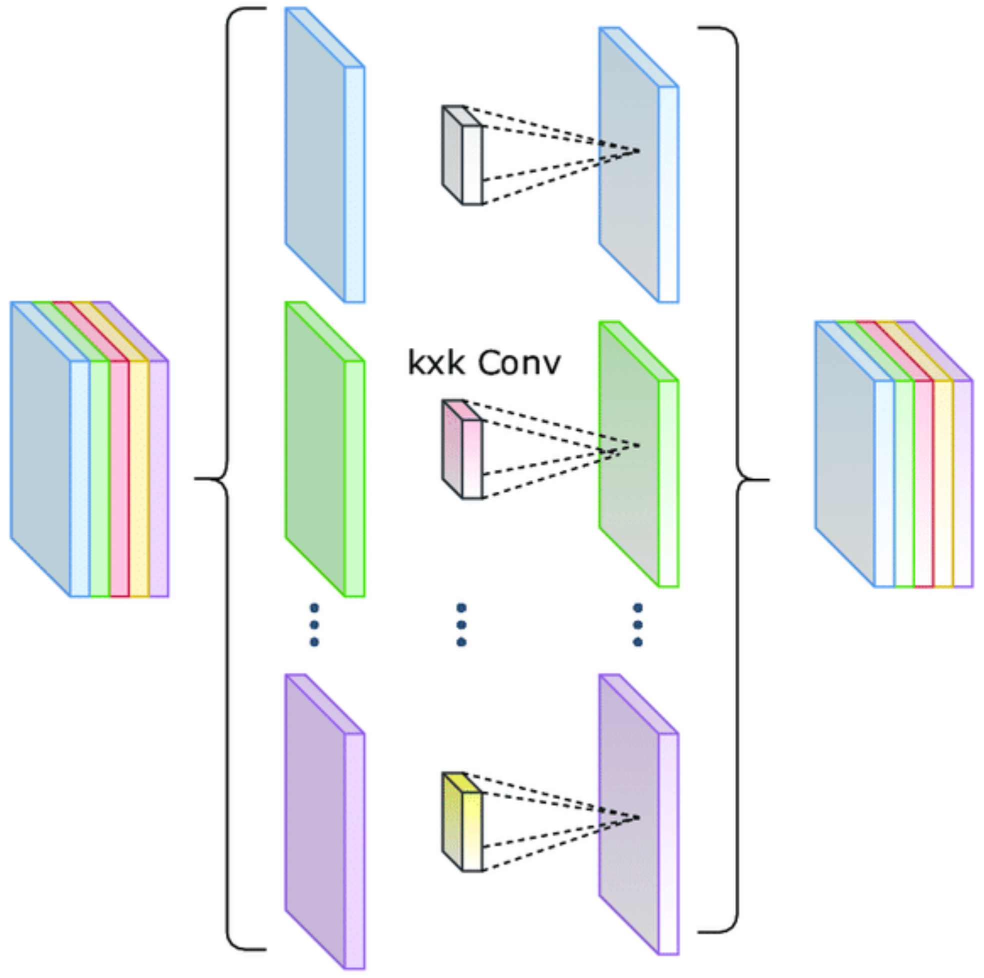
Notations	
n	Batch Size
c_i	Input Channels
c_o	Output Channels
h_i, h_o	Input/Output Height
w_i, w_o	Input/Output Width
k_h, k_w	Kernel Height/Width
g	Groups

Number of Multiply-Accumulate Operations

MAC

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

* bias is ignored



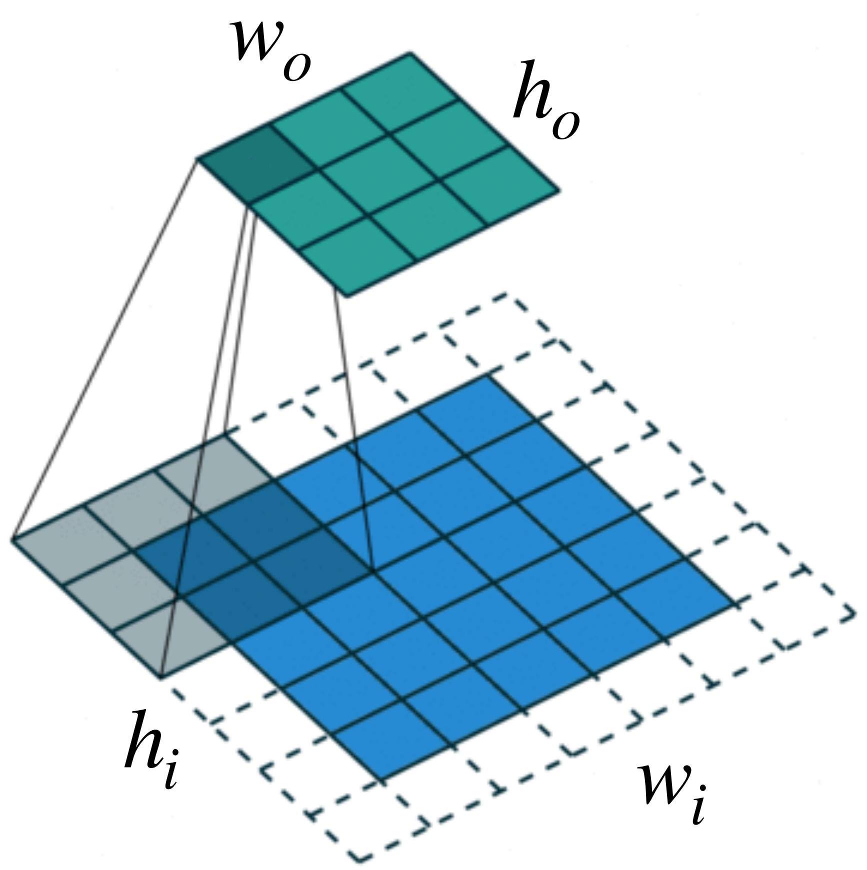
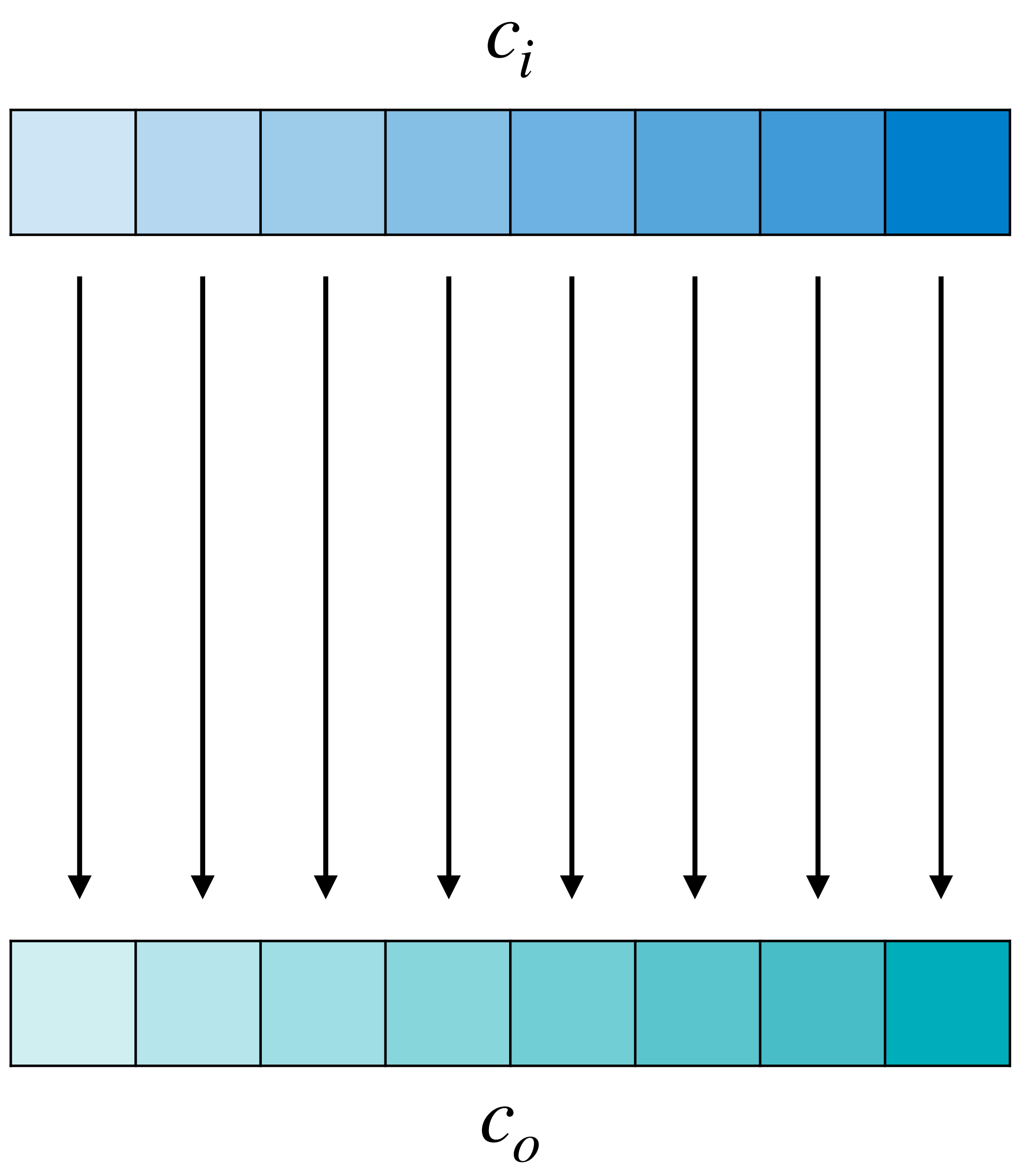


Image source: 1



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

AlexNet: #MACs

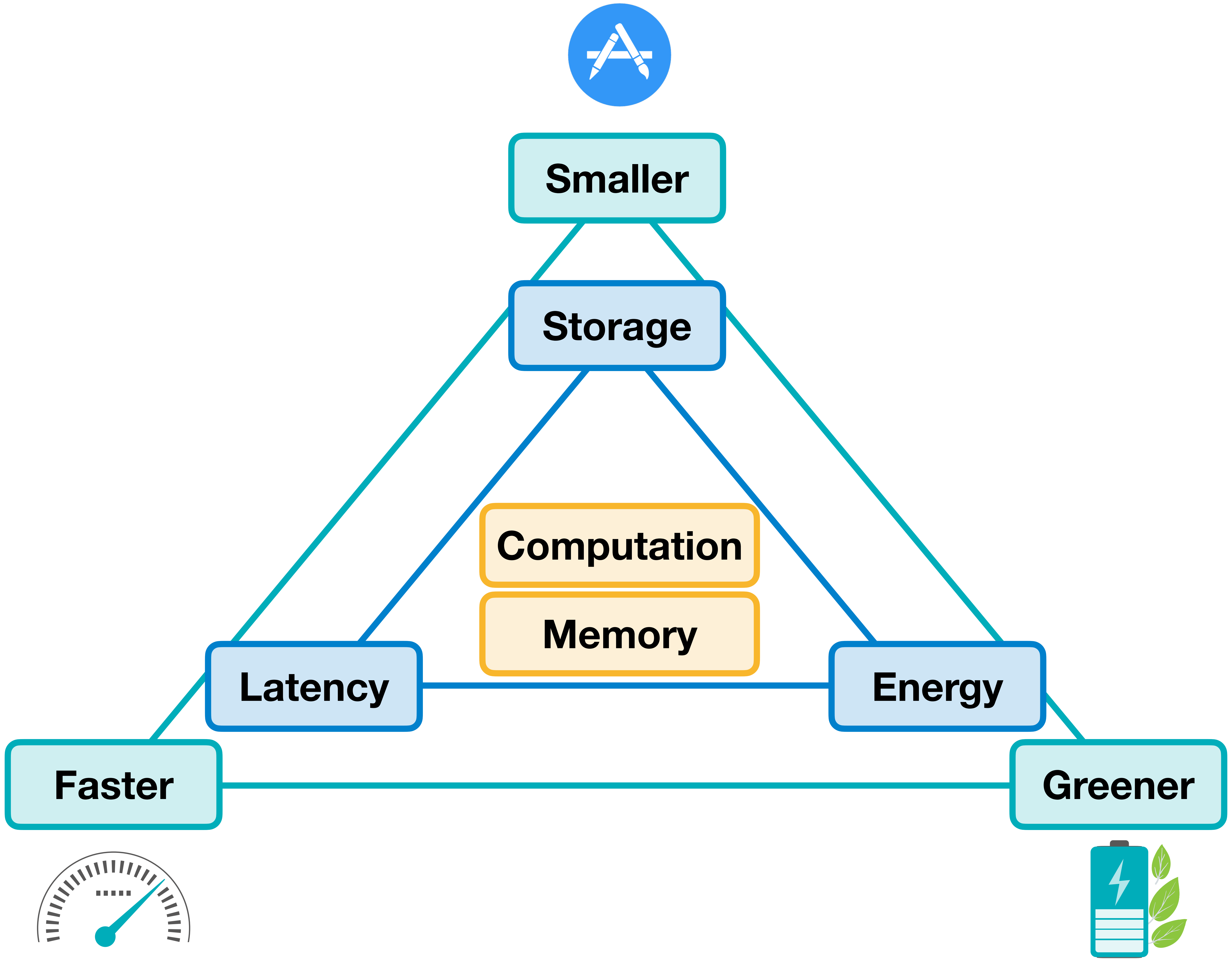
AlexNet	C × H × W	MACs
Image (3×224×224)	3×224×224	
11×11 Conv, channel 96, stride 4, pad 2	96×55×55	96×3×11×11×55×55 = 105,415,200
3x3 MaxPool, stride 2	96×27×27	
5x5 Conv, channel 256, pad 2, groups 2	256×27×27	256×96×5×5×27×27 / 2 = 223,948,800
3x3 MaxPool, stride 2	256×13×13	
3x3 Conv, channel 384, pad 1	384×13×13	384×256×3×3×13×13 = 149,520,384
3x3 Conv, channel 384, pad 1, groups 2	384×13×13	384×384×3×3×13×13 / 2 = 112,140,288
3x3 Conv, channel 256, pad 1, groups 2	256×13×13	256×[illegible]3×3×13×13 / 2 = 74,760,1[illegible] ?
3x3 MaxPool, stride 2	256×6×6	
Linear, channel 4096	4096	4096×(256×6×6) = 37,748,736
Linear, channel 4096	4096	4096×4096 = 16,777,216
Linear, channel 1000	1000	1000×4096 = 4,096,000

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	$c_o \cdot k_h \cdot k_w \cdot h_o \cdot w_o$

724M in total !

ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky *et al.*, NeurIPS 2012]

Efficiency of Neural Networks



Efficiency Metrics

Memory-Related

#parameters

model size

total/peak #activations

Computation-Related

MAC

FLOP, FLOPS

OP, OPS

Image source: 1

Number of Floating Point Operations (FLOP)

- A multiply is a floating point operation
- An add is a floating point operation
- ***One*** multiply-accumulate operation is ***two*** floating point operations (FLOP)
  - Example: AlexNet has 724M MACs, total number of floating point operations will be
    - 724M × 2 = 1.4G FLOPs

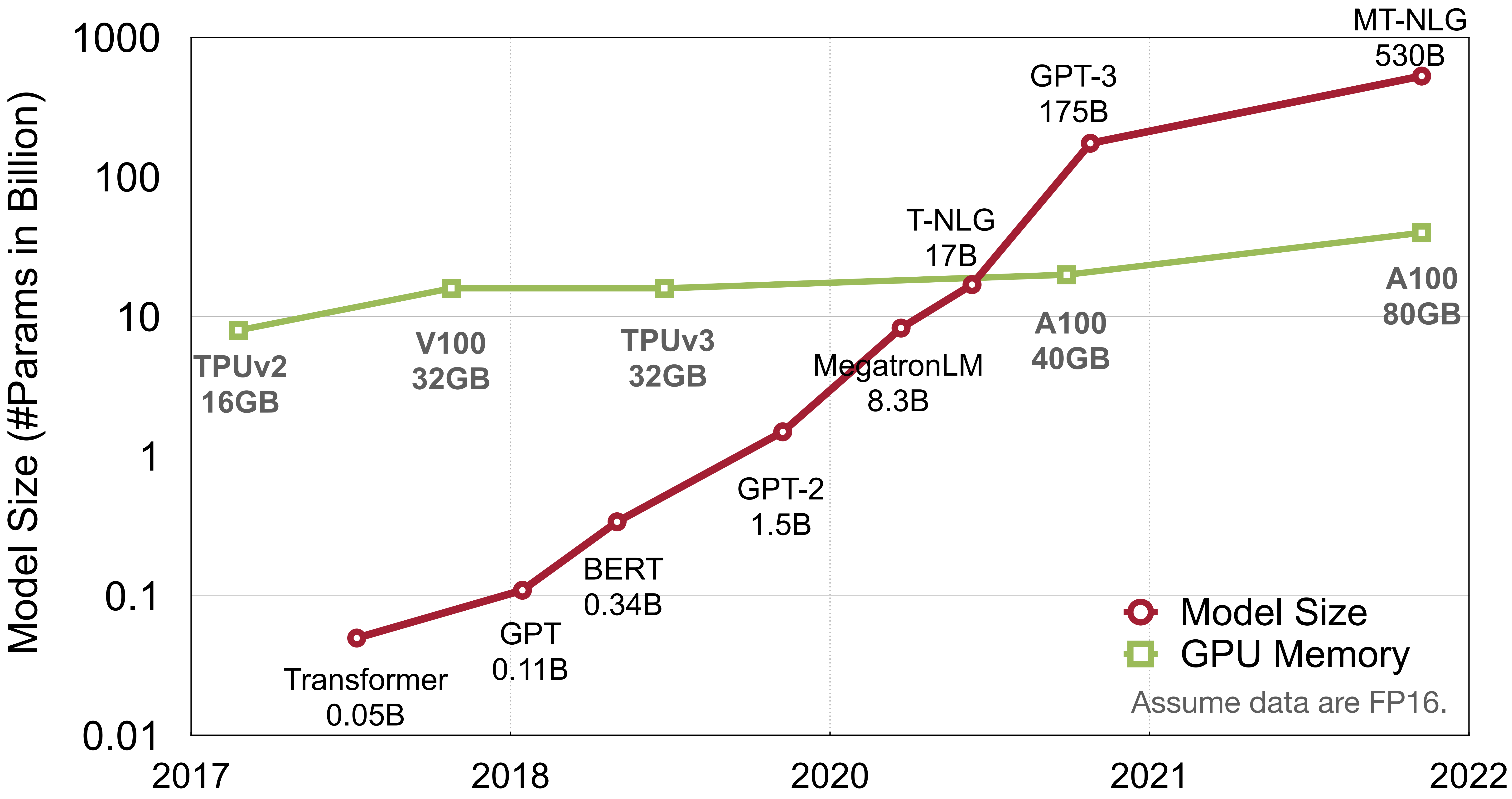
- Floating Point Operation Per Second (FLOPS)
  - $FLOPS = \frac{FLOPs}{second}$

Number of Operations (OP)

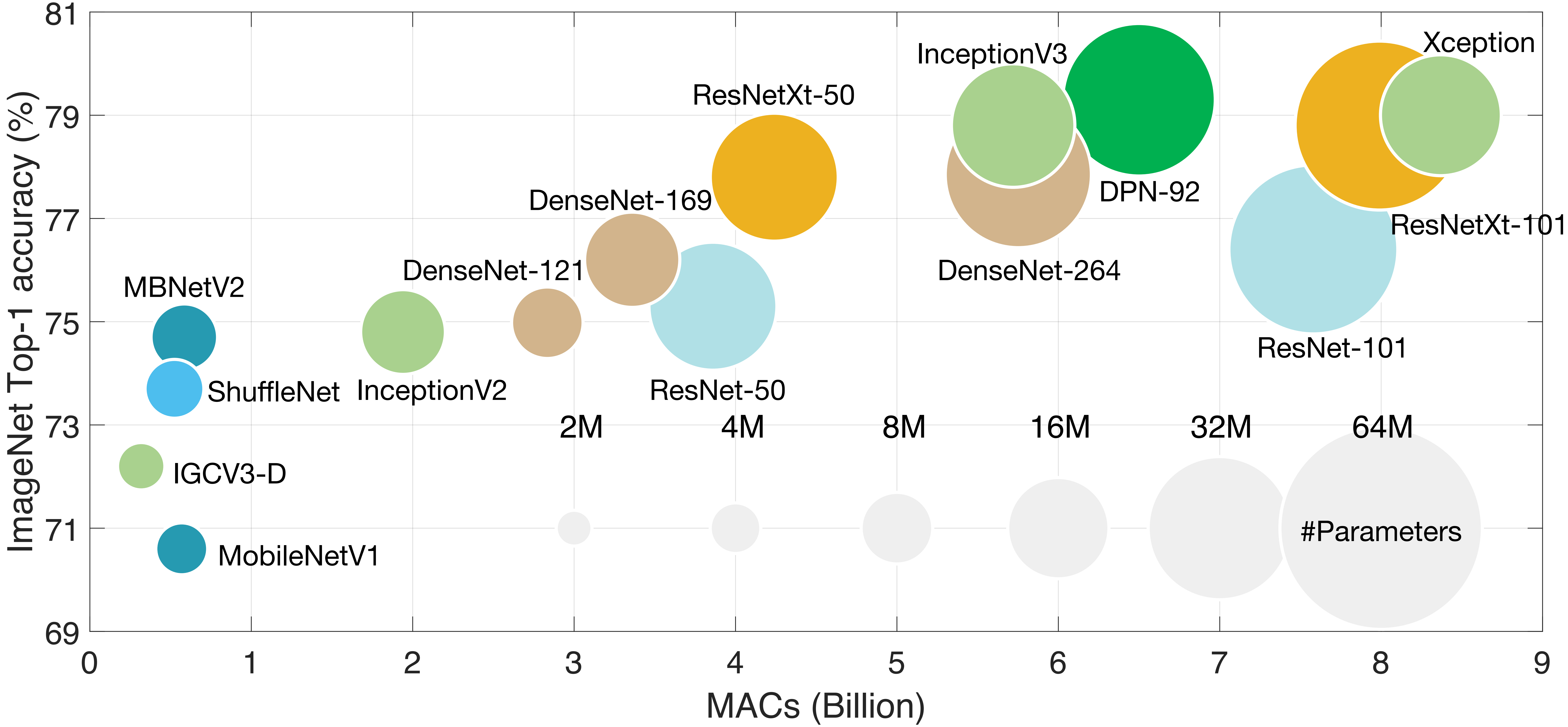
- Activations/weights in neural network computing are not always floating point.
- To generalize, number of operations is used to measure the computation amount.
  - Example: AlexNet has 724M MACs, total number of floating point operations will be
    - 724M × 2 = 1.4G OPs

- Similarly, Operation Per Second (OPS)
  - $OPS = \frac{OPs}{second}$

Today's AI is too BIG!

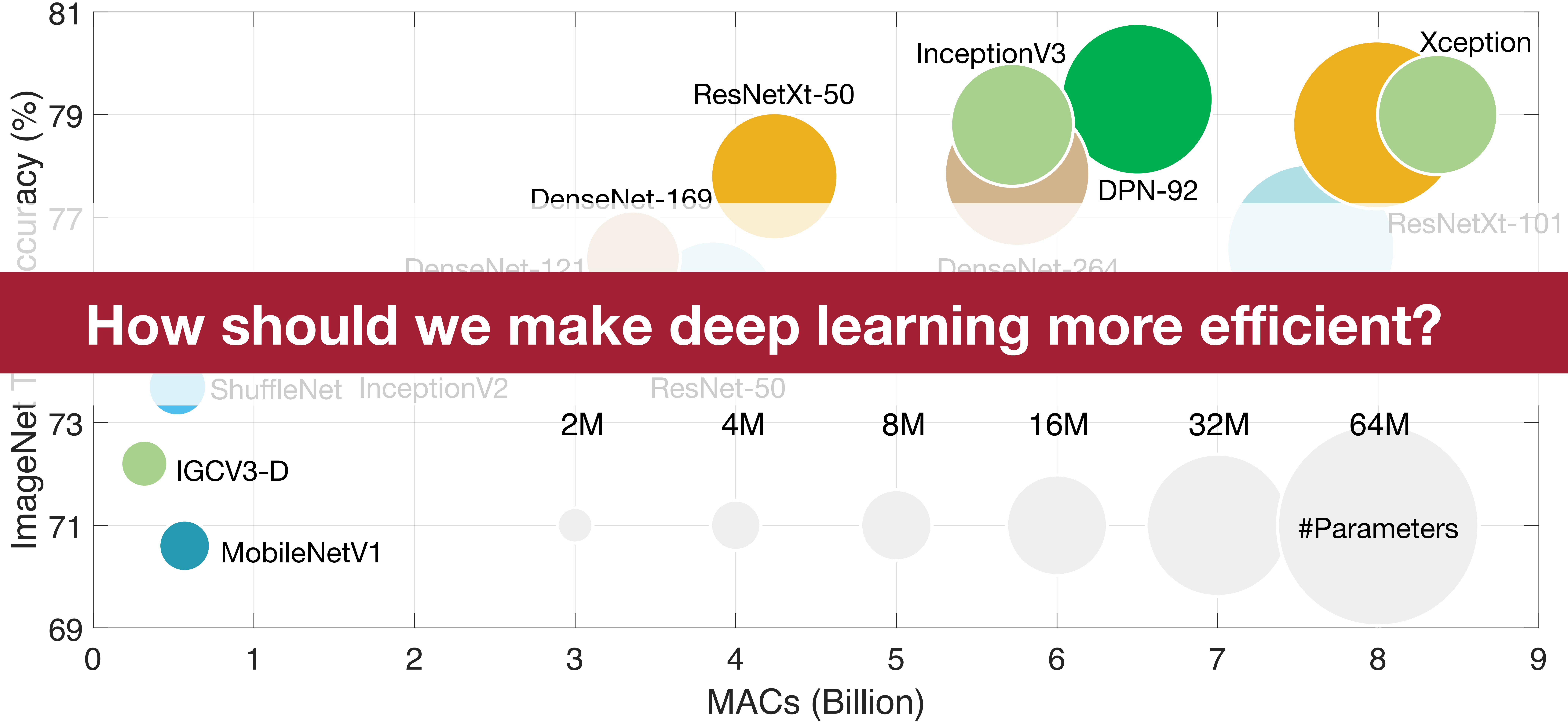


Today's AI is too BIG!



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

Today's AI is too BIG!



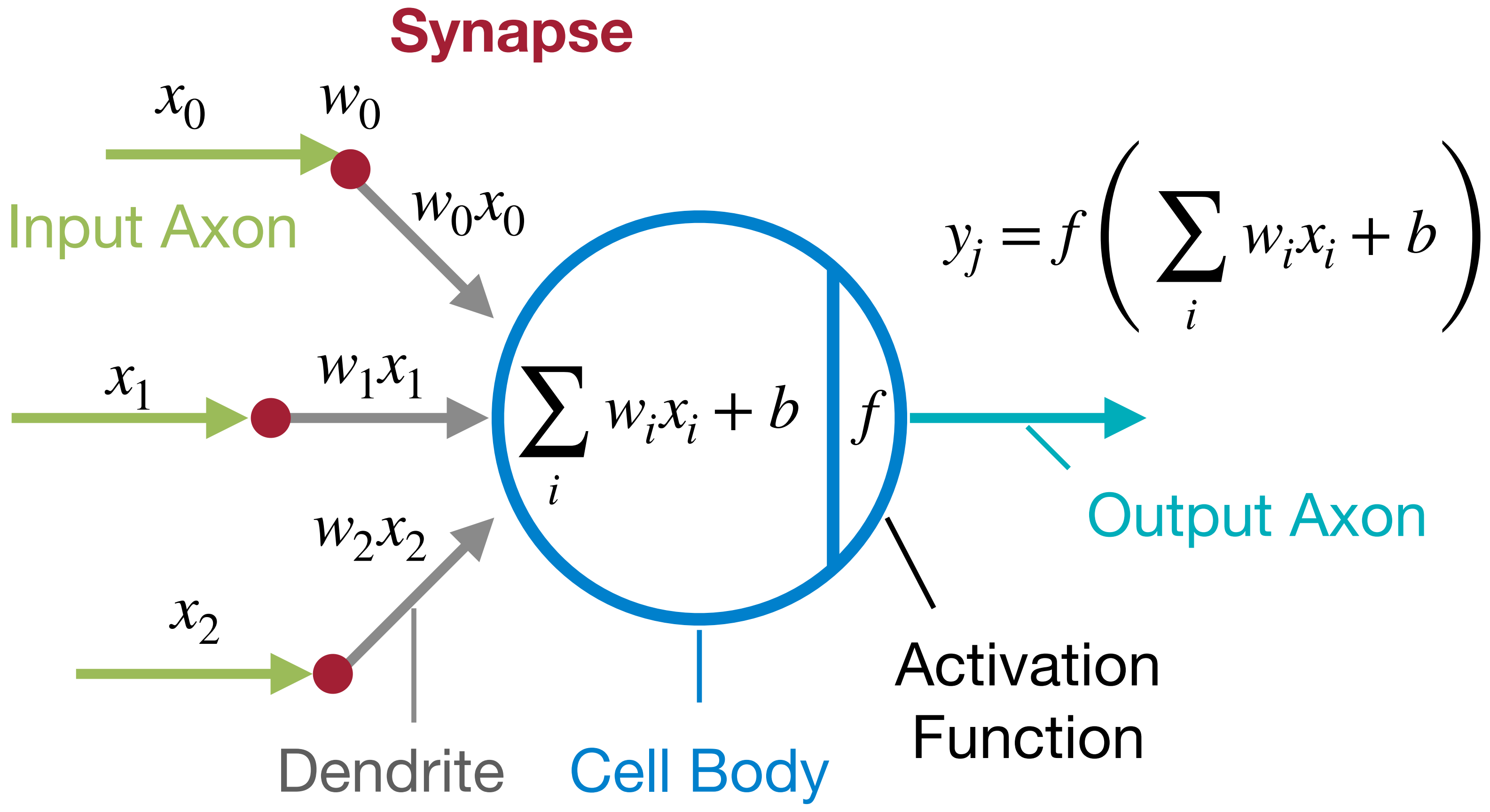
How should we make deep learning more efficient?

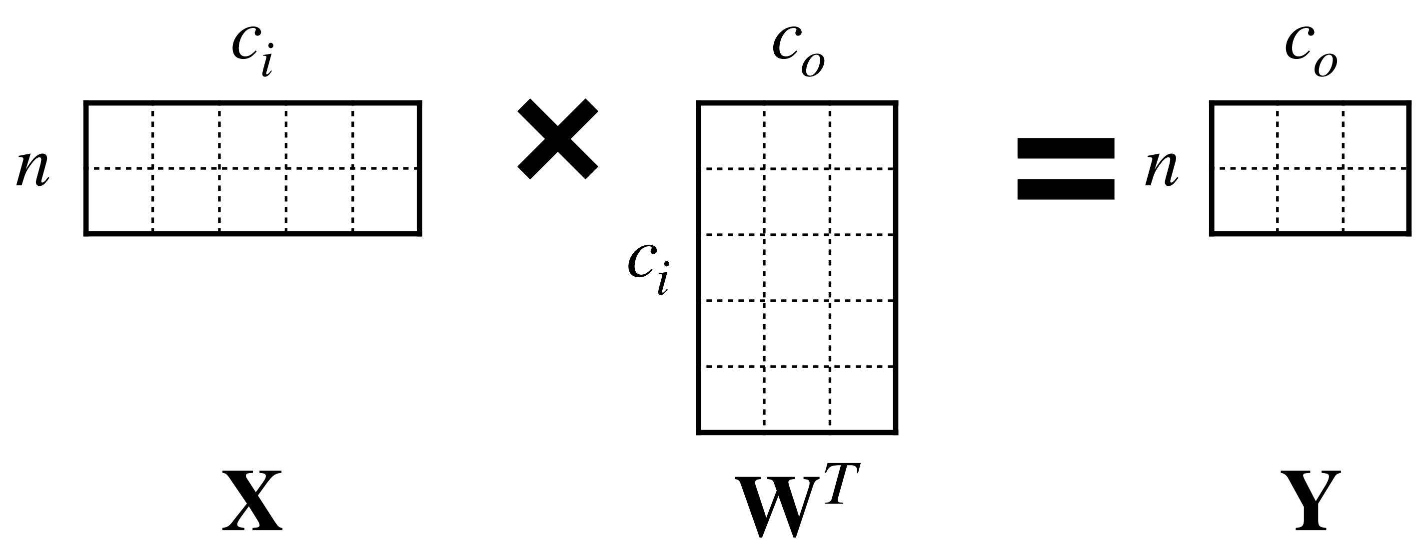
Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey [Deng *et al.*, IEEE 2020]

Summary of Today's Lecture

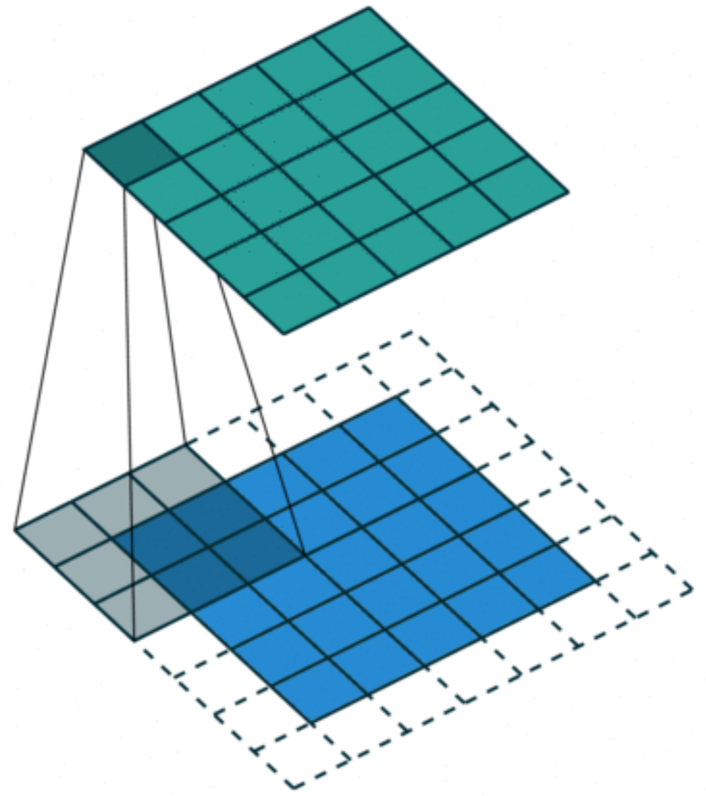
In this lecture, we

- Reviewed the basics of neural networks
  - terminology: neuron output → activation, synapses → weight
  - popular layers: fully-connected, convolution, depthwise convolution, pooling, normalization
  - classic neural networks: AlexNet, VGG-16, ResNet-50, MobileNetV2
- Introduced popular efficiency metrics for neural networks
  - memory cost: #Parameters, Model Size, #Activations
  - computation cost: MACs, FLOP, FLOPS, OP, OPS





Linear Layer



Convolution Layer

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

Image source: 1

Lab 0: Getting Started

Tutorial on PyTorch and Laboratory Exercises

References

1. Convolution arithmetic [Github Repo]
2. Image Classification with CNNs [Stanford CS231n Lecture 5]
3. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe *et al.*, ICML 2015]
4. Group Normalization [Wu *et al.*, ECCV 2018]
5. ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky *et al.*, NeurIPS 2012]
6. Very Deep Convolutional Networks for Large-Scale Image Recognition [Simonyan *et al.*, ICLR 2015]
7. Deep Residual Learning for Image Recognition [He *et al.*, CVPR 2016]
8. MobileNetV2: Inverted Residuals and Linear Bottlenecks [Sandler *et al.*, CVPR 2018]
9. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey [Deng et al., IEEE 2020]