Foundations of Edge AI

Lecture05

Basics of Edge AI

Lanyu (Lori) Xu

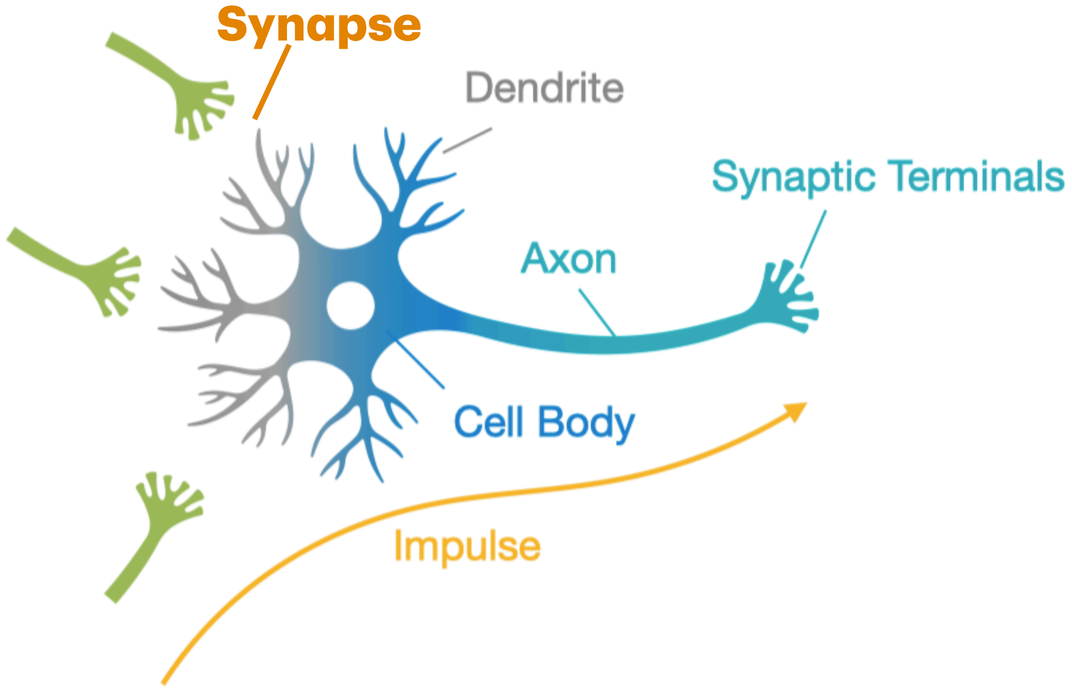
Email: lxu@oakland.edu
Homepage: https://lori930.github.io/
Office: EC 524

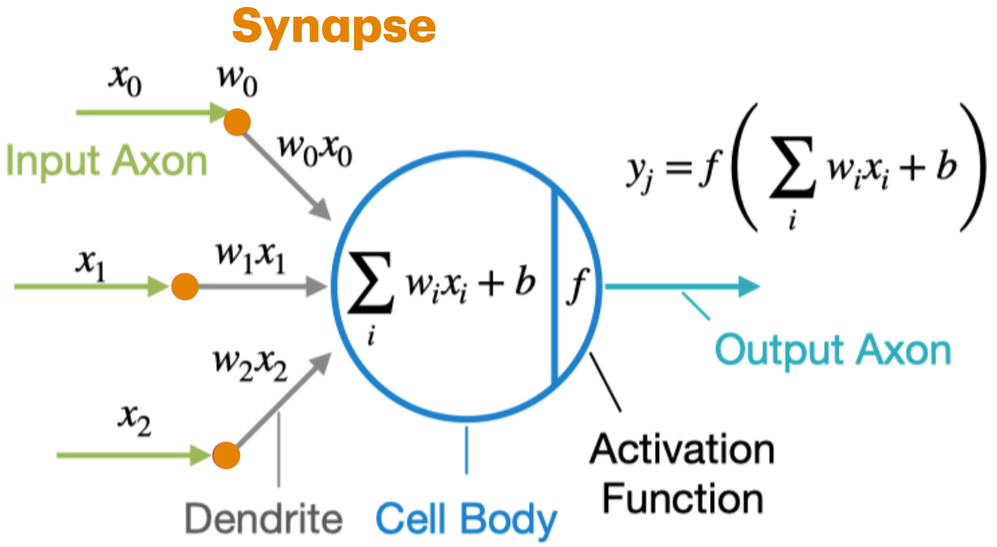
Lecture Plan

Today we will:

- Review the terminology of neural networks
  - Neuron, synapses, activation, feature, weight, parameter, etc.
- Review popular building blocks in a neural network
  - Fully-connected, convolution, grouped convolution, depthwise convolution
  - Stride, pooling, mormalization, transformer
- Review popular convolutional neural networks' architecture
  - AlexNet, VGG-16, ResNet-50, MobileNetV2
- Introduce popular efficiency metrics for neural networks
  - #Parameters, model size, peak #activations, MAC, FLOP, FLOPS, OP, OPS, latency, throughput
- Warmup Lab

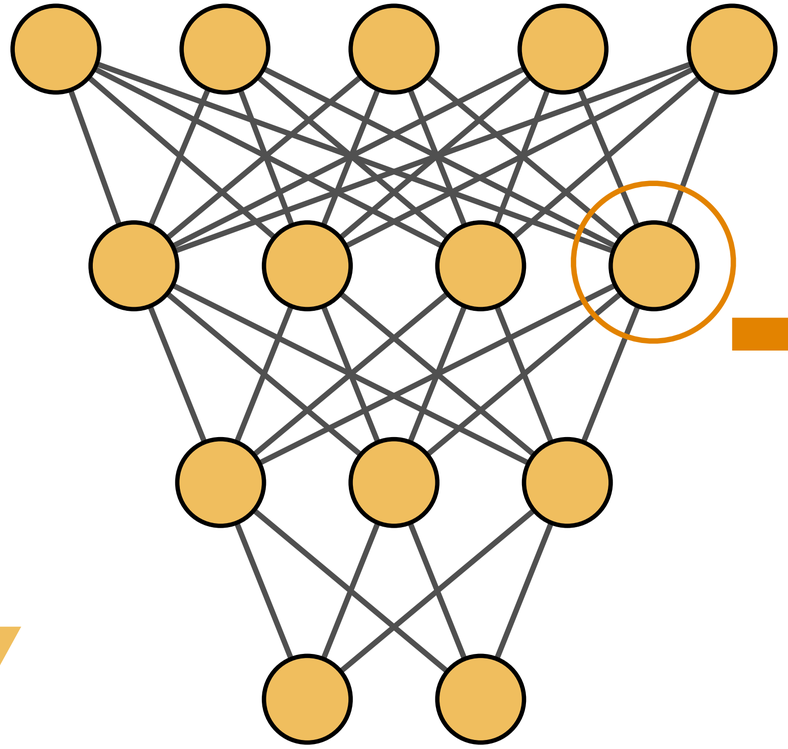
Neuron and Synapse





Deep Neural Network

Example: 3-layer neural network with 2 hidden layers



Input Layer $\in \mathbb{R}^5$

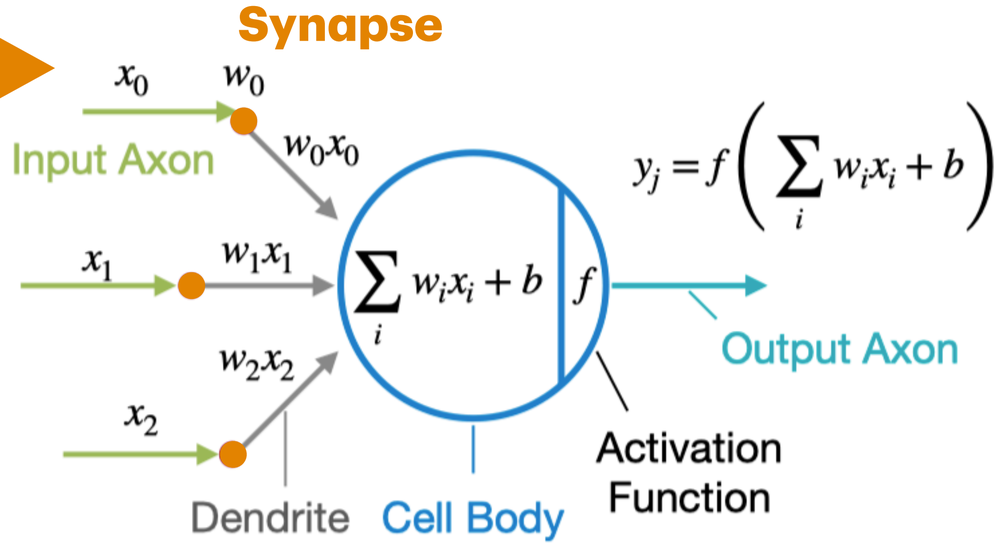
Hidden Layer $\in \mathbb{R}^4$

Hidden Layer $\in \mathbb{R}^3$

Output Layer $\in \mathbb{R}^2$

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

- Synapses = Weights = Parameters
- Neurons = Features = Activations



With the same amount of parameters, you create two NN models

A. Shallow and wide

B. Narrow and deep

_____ is more hardware efficient

_____ is more accurate.

A **B**

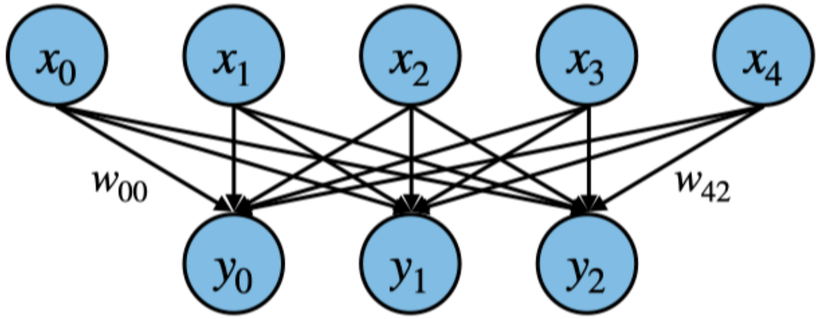
Popular Neural Network Layers

Fully-Connected Layer (Linear Layer)

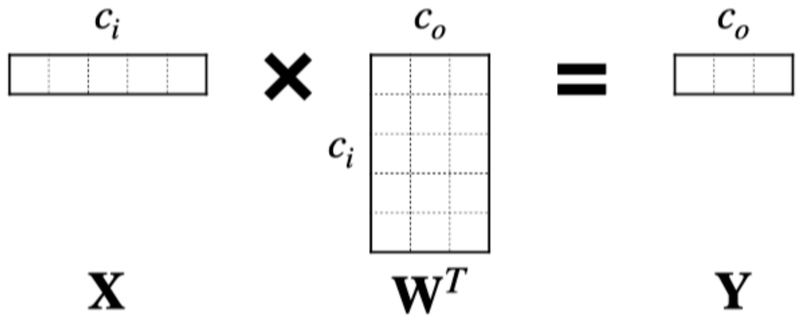
The output neuron is connected to all input neurons

- Shape of tensors
  - Input features $X : (1, c_i)$
  - Output features $Y : (1, c_o)$
  - Weights $W : (c_o, c_i)$
  - Bias $b : (c_o,)$

Notations	
c_i	Input channels
c_o	Output Channels



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

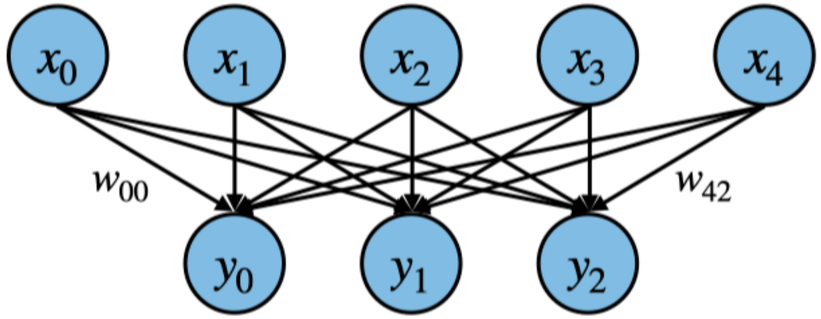


Fully-Connected Layer (Linear Layer)

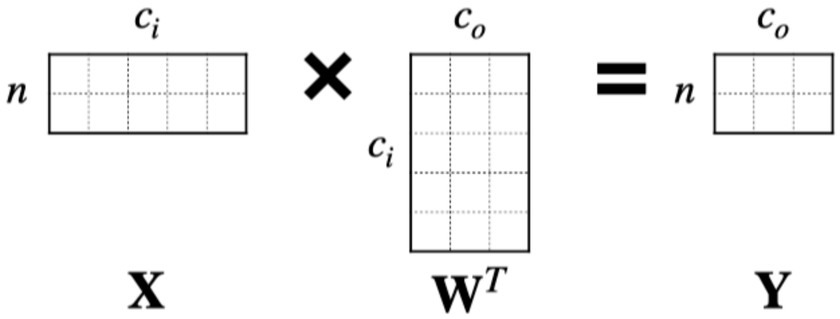
The output neuron is connected to all input neurons

- Shape of tensors
  - Input features X : (n, c_i)
  - Output features Y : (n, c_o)
  - Weights W : (c_o, c_i)
  - Bias b : $(c_o,)$

Notations	
`ci`	Input channels
`co`	Output channels
`n`	Batch size



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

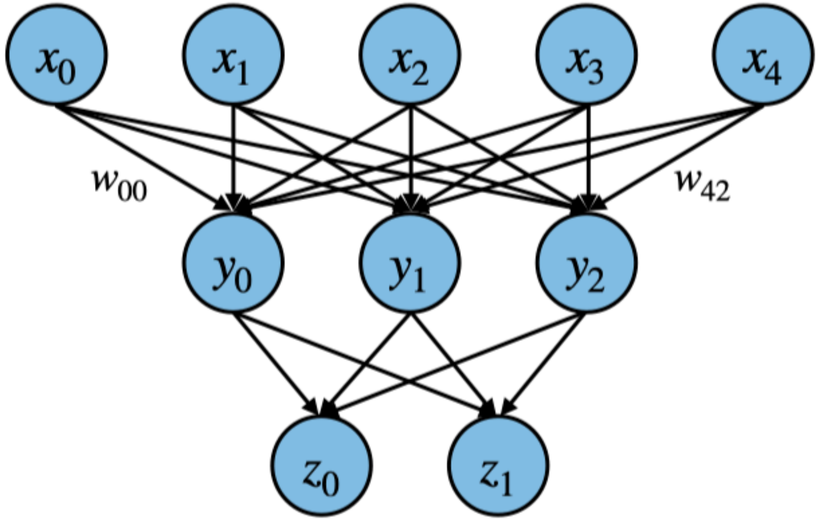


Fully-Connected Layer (Linear Layer)

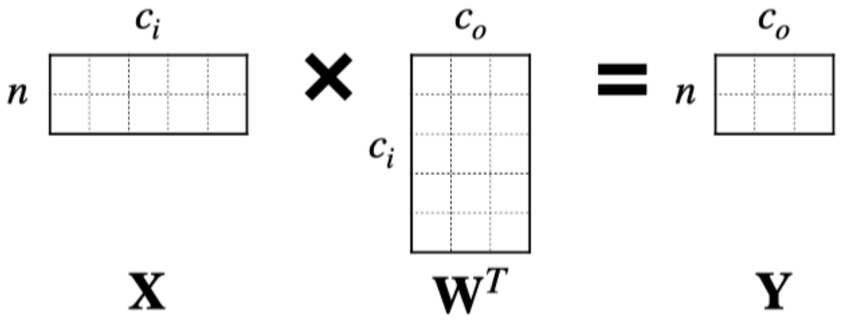
The output neuron is connected to all input neurons

- Shape of tensors
  - Input features X : (n, c_i)
  - Output features Y : (n, c_o)
  - Weights W : (c_o, c_i)
  - Bias b : $(c_o,)$

Notations	
c_i	Input channels
c_o	Output channels
n	Batch size



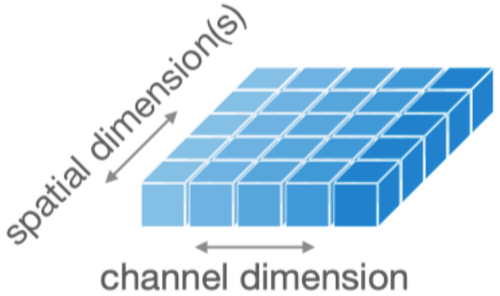
Multilayer Perceptron (MLP)



Convolution Layer (1D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, w_i)$
  - Output features $Y : (n, c_o, w_o)$
  - Weights $W : (c_o, c_i, ?)$
  - Bias $b : (c_o,)$



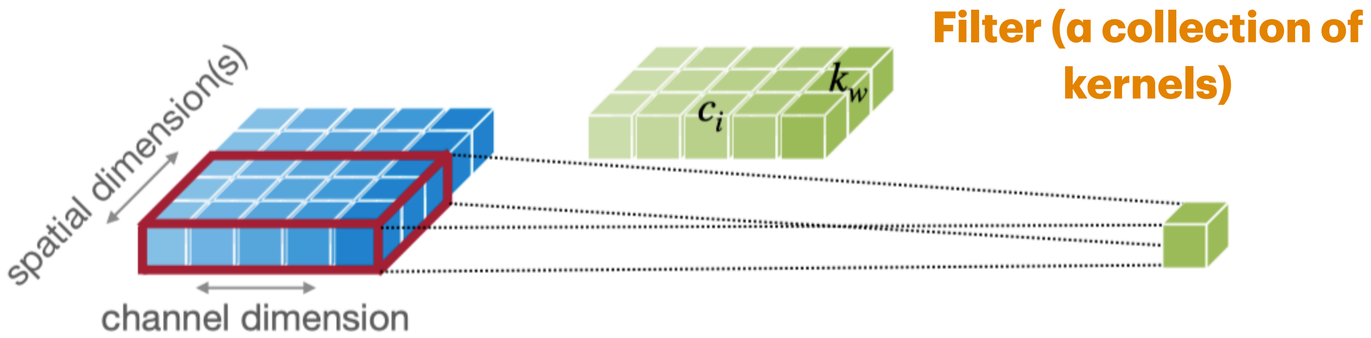
What could be the application?

Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width

Convolution Layer (1D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, w_i)$
  - Output features $Y : (n, c_o, w_o)$
  - Weights $W : (c_o, c_i, k_w)$
  - Bias $b : (c_o,)$

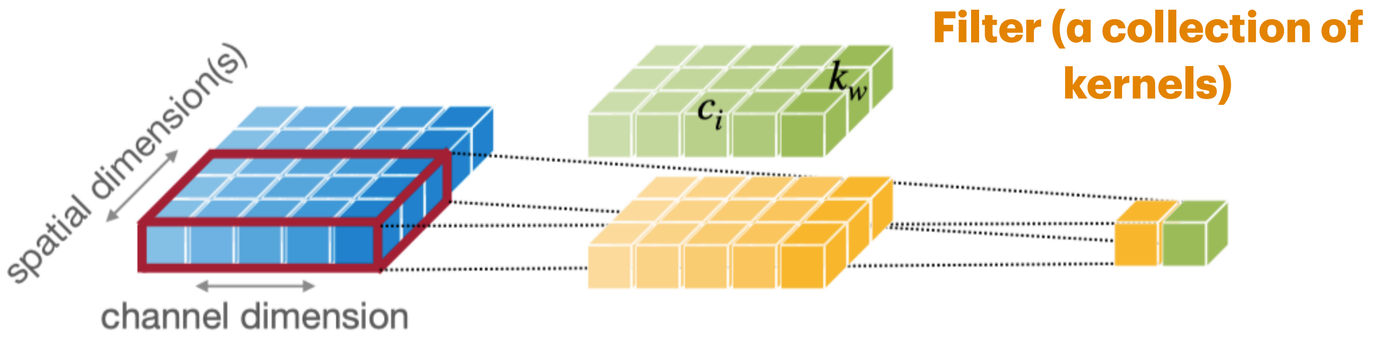


Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
k_w	Kernel width

Convolution Layer (1D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, w_i)$
  - Output features $Y : (n, c_o, w_o)$
  - Weights $W : (c_o, c_i, k_w)$
  - Bias $b : (c_o,)$

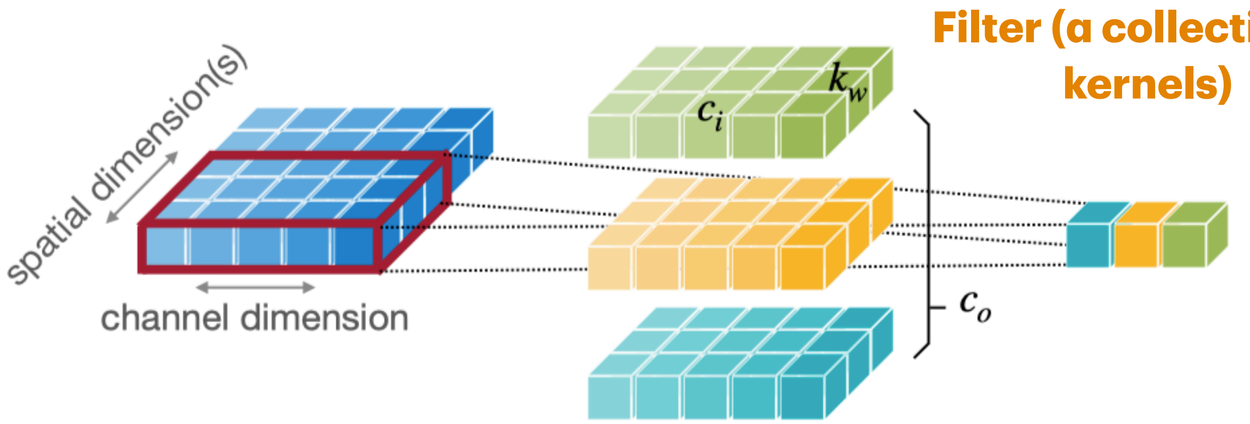


Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
k_w	Kernel width

Convolution Layer (1D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, w_i)$
  - Output features $Y : (n, c_o, w_o)$
  - Weights $W : (c_o, c_i, k_w)$
  - Bias $b : (c_o,)$

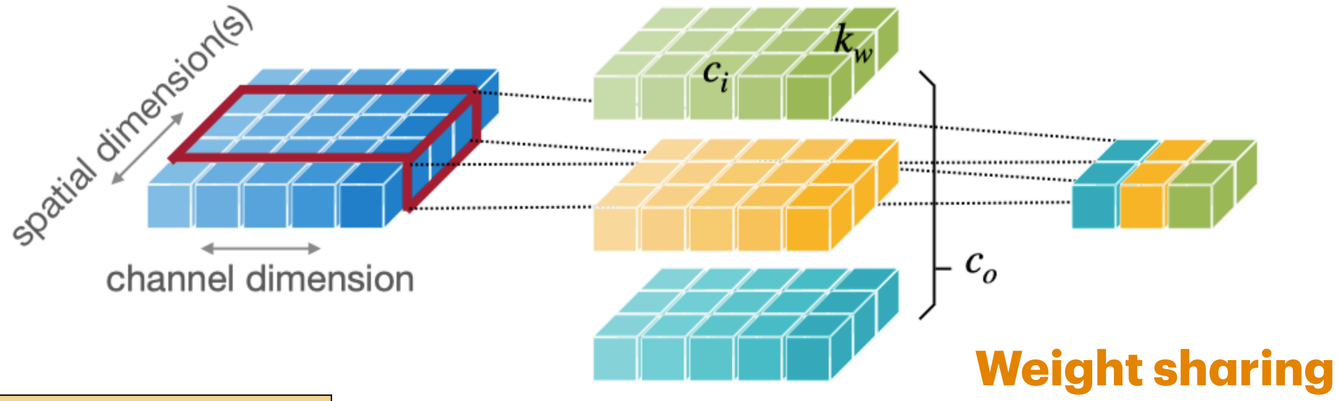


Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
k_w	Kernel width

Convolution Layer (1D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, w_i)$
  - Output features $Y : (n, c_o, w_o)$
  - Weights $W : (c_o, c_i, k_w)$
  - Bias $b : (c_o,)$



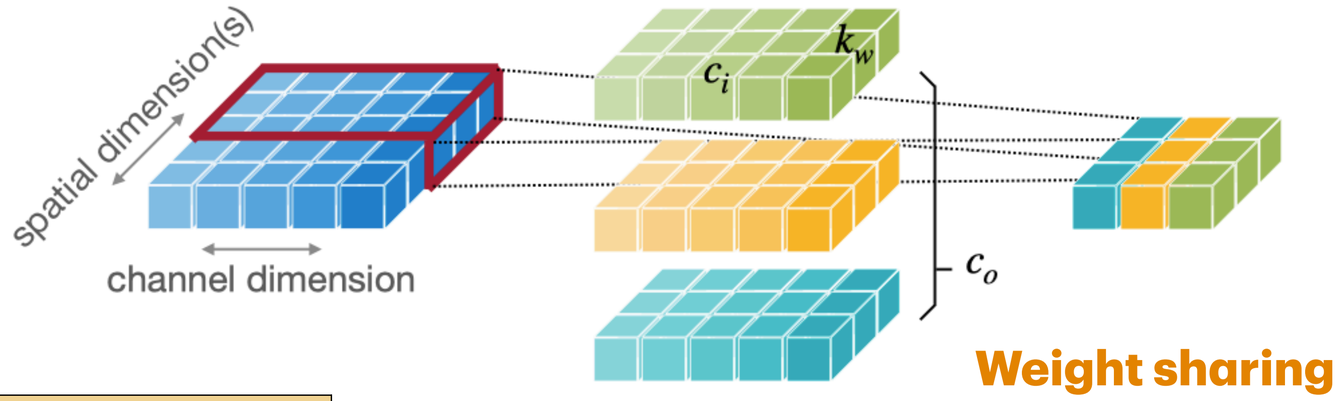
Weight sharing

Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
k_w	Kernel width

Convolution Layer (1D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, w_i)$
  - Output features $Y : (n, c_o, w_o)$
  - Weights $W : (c_o, c_i, k_w)$
  - Bias $b : (c_o,)$



Weight sharing

Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
k_w	Kernel width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, ?, w_i)$
  - Output features $Y : (n, c_o, ?, w_o)$
  - Weights $W : (c_o, c_i, ?, k_w)$
  - Bias $b : (c_o,)$

Activation map (aka feature map)

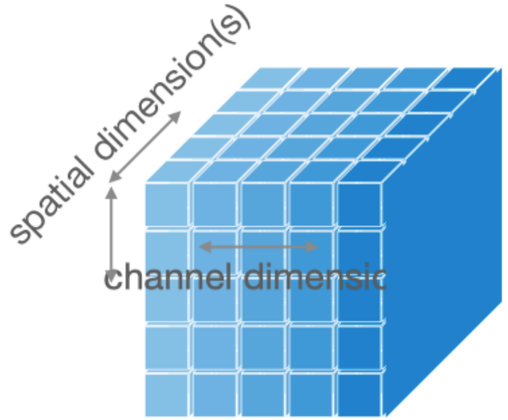
$h \times w$

Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
k_w	Kernel width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$

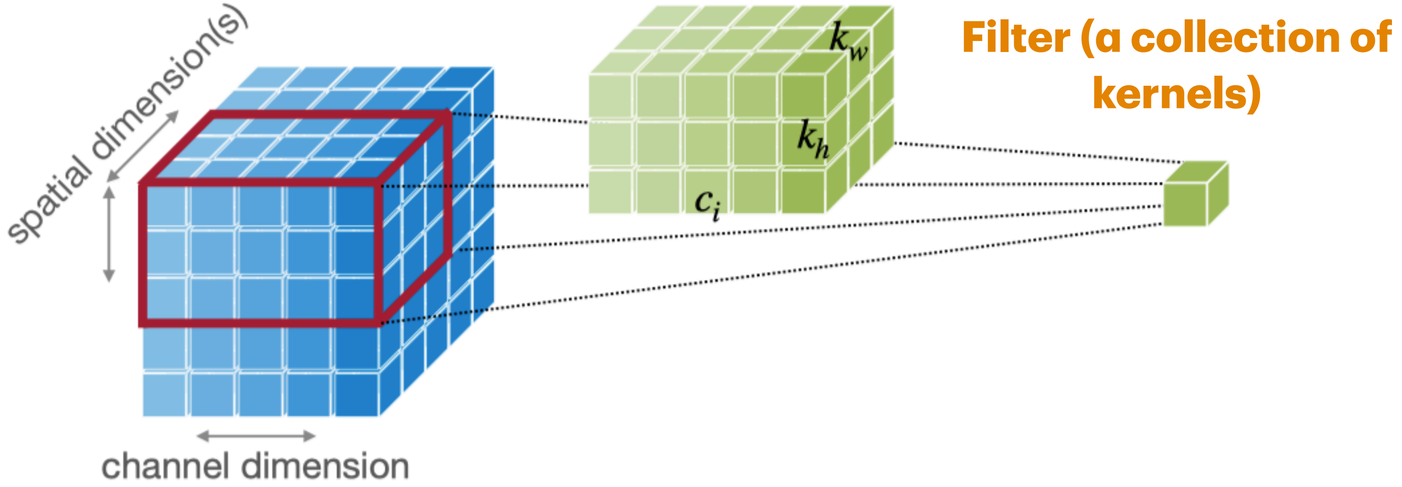


Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$

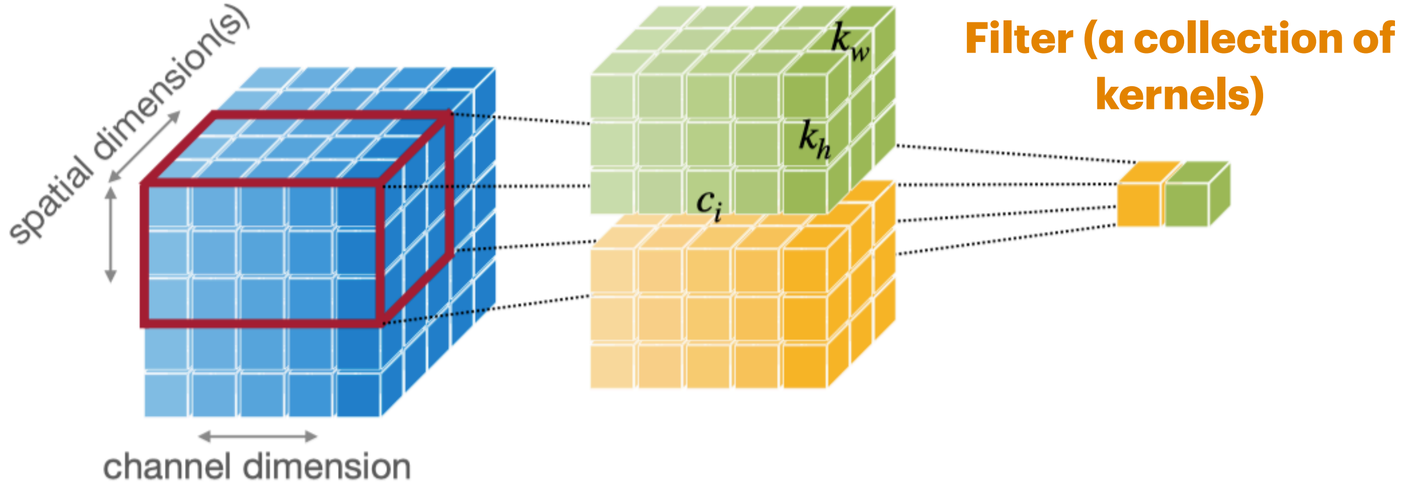


Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$

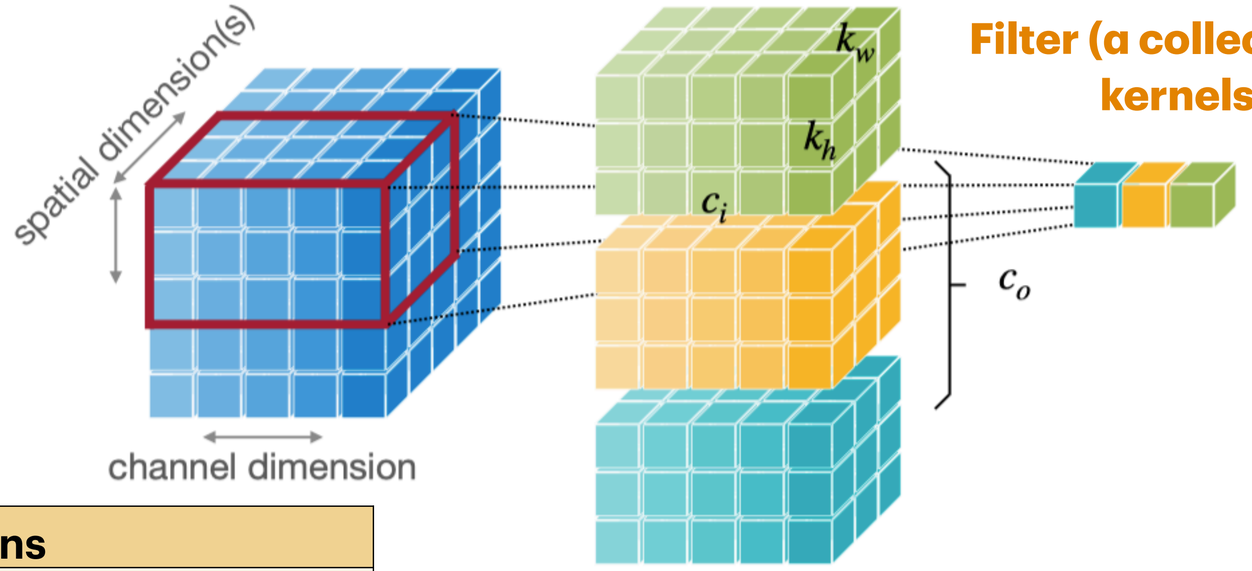


Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$

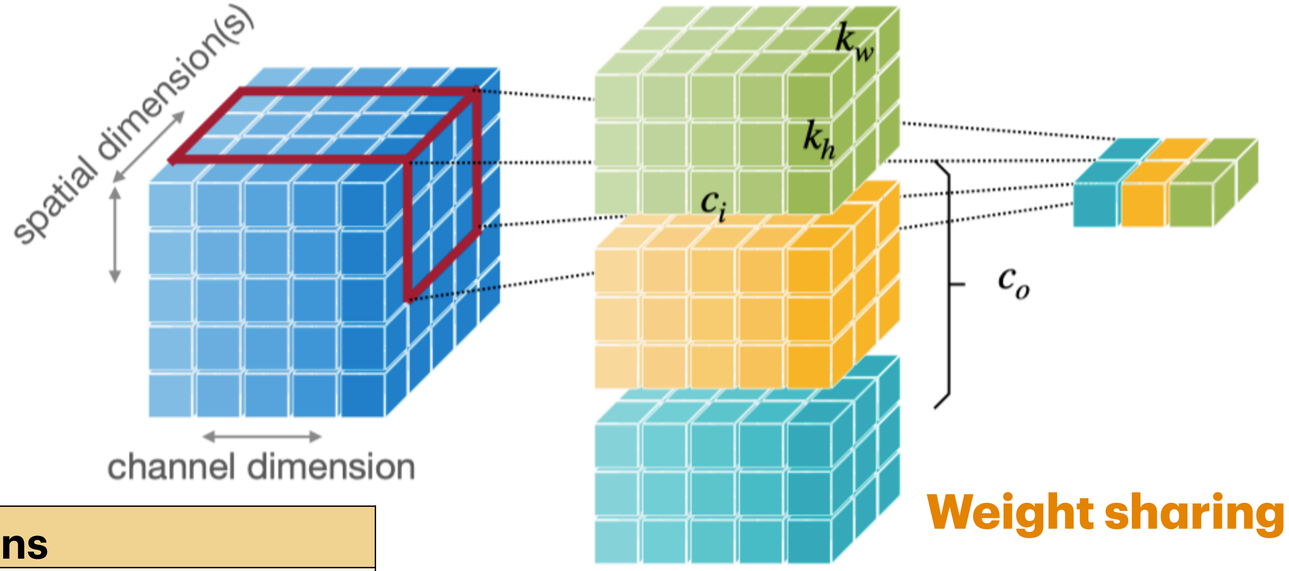


Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

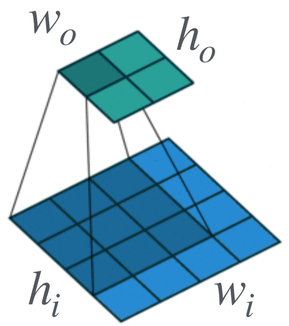
Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$



Weight sharing

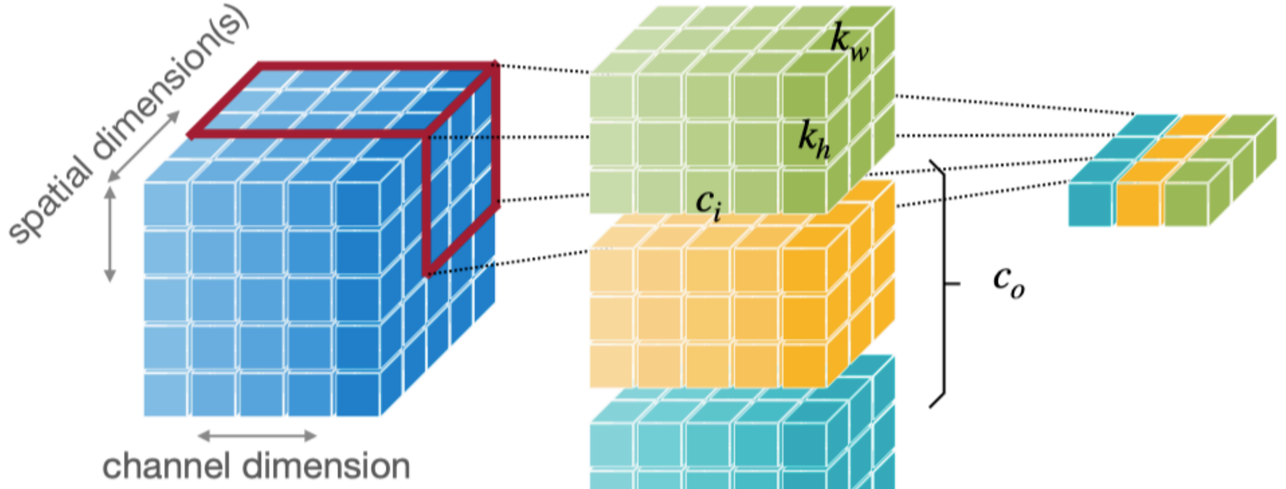


Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

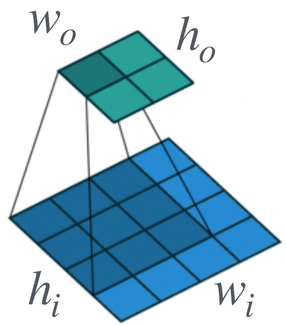
Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$



Weight sharing

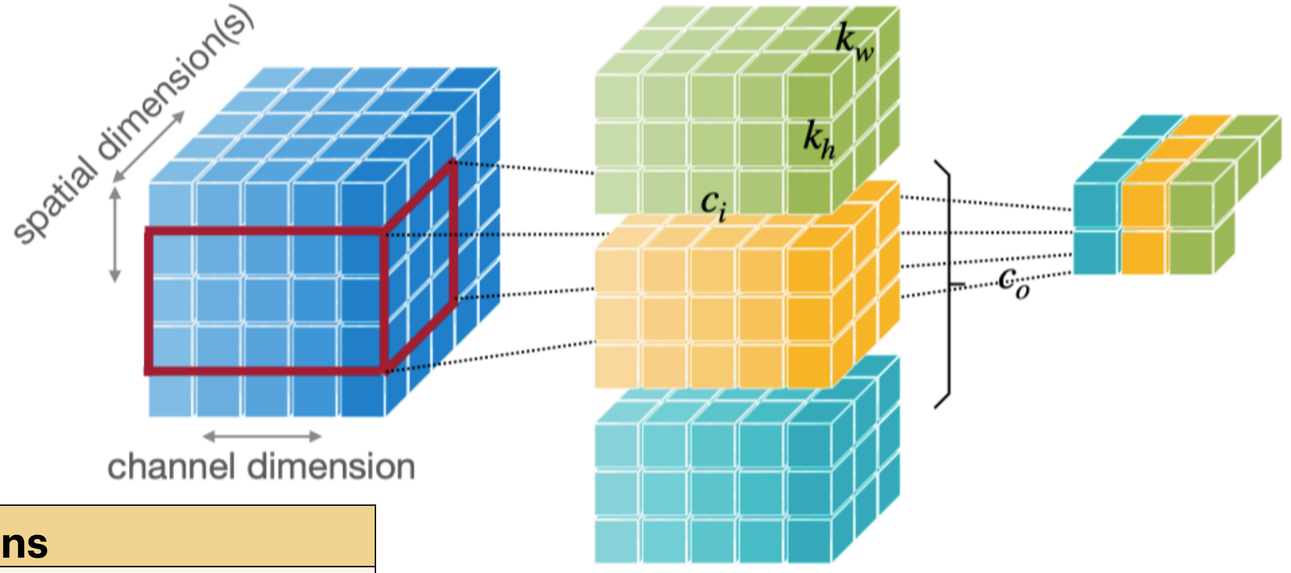


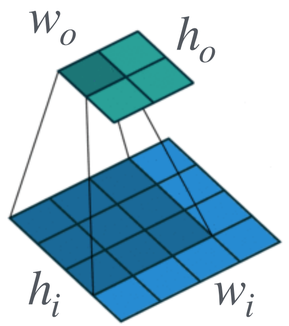
Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$



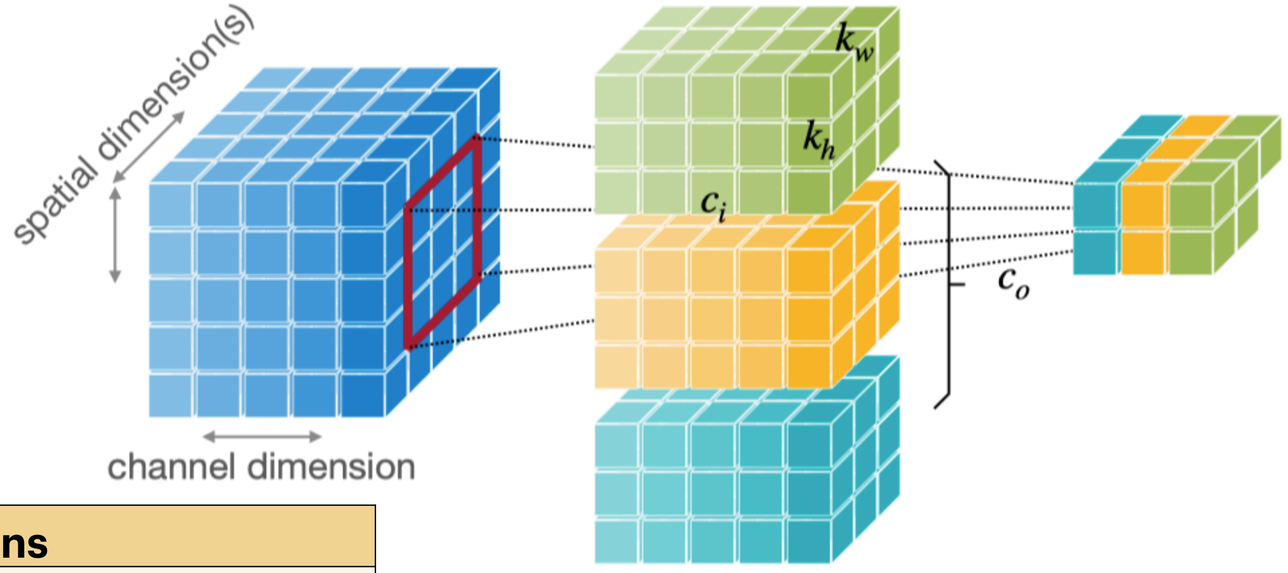


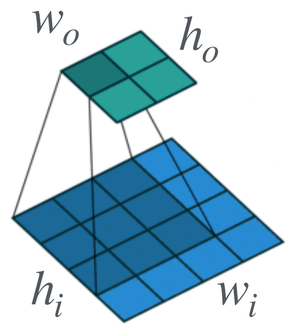
Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$



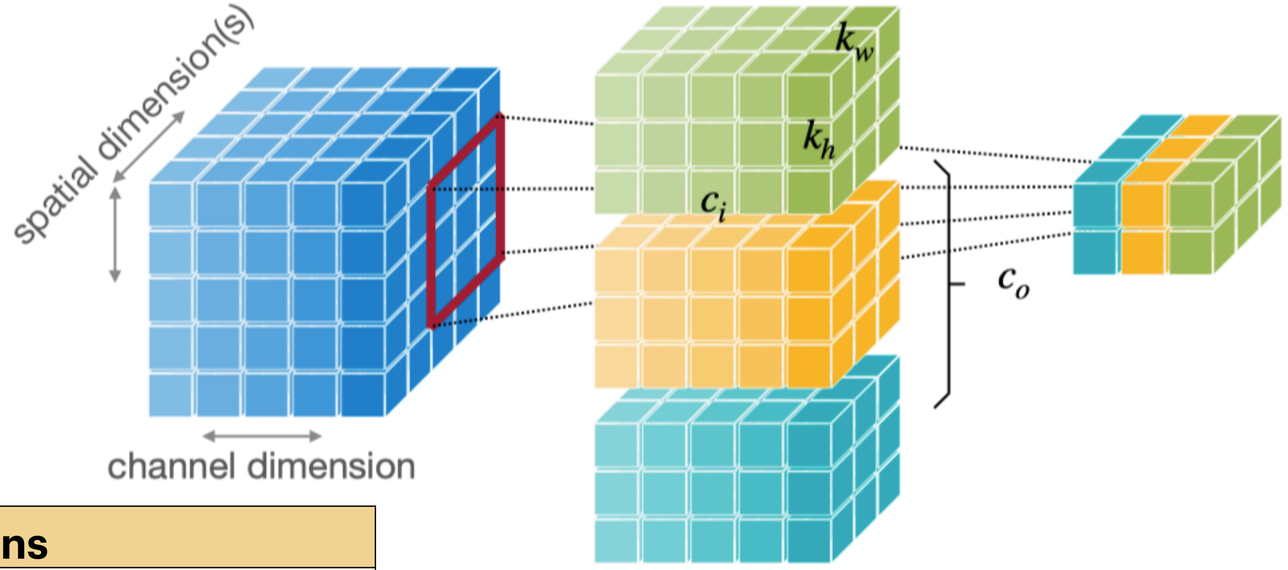


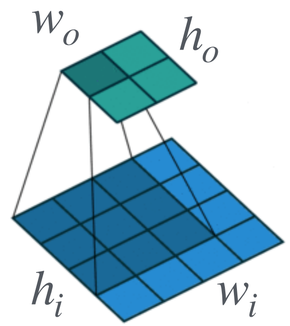
Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$



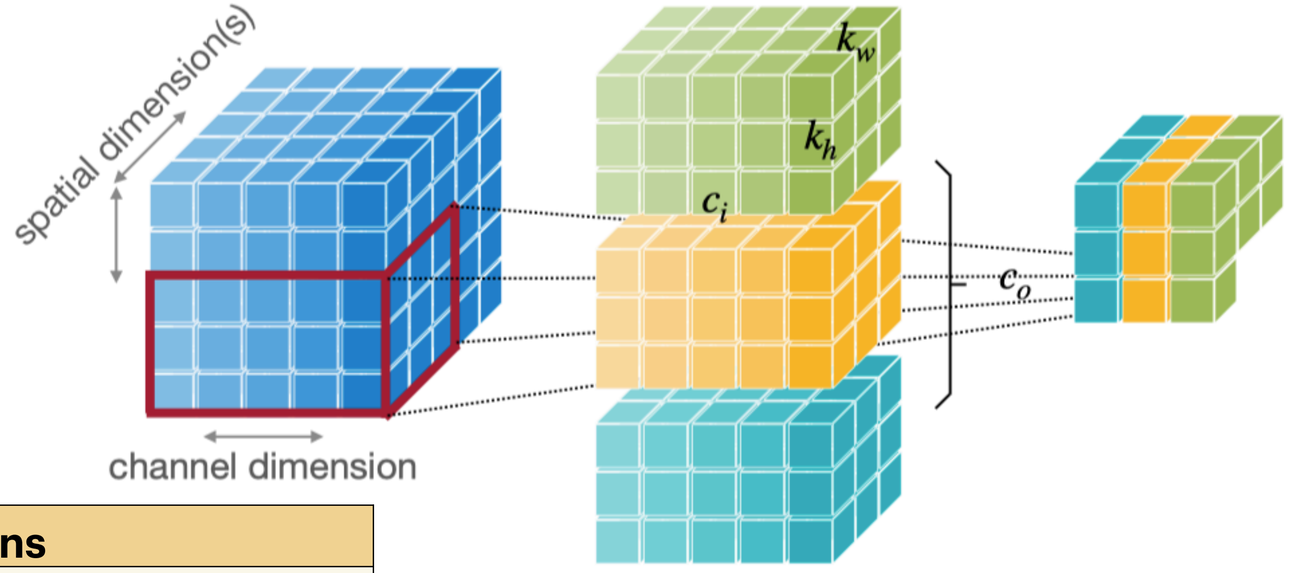


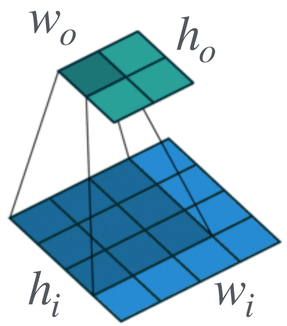
Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$



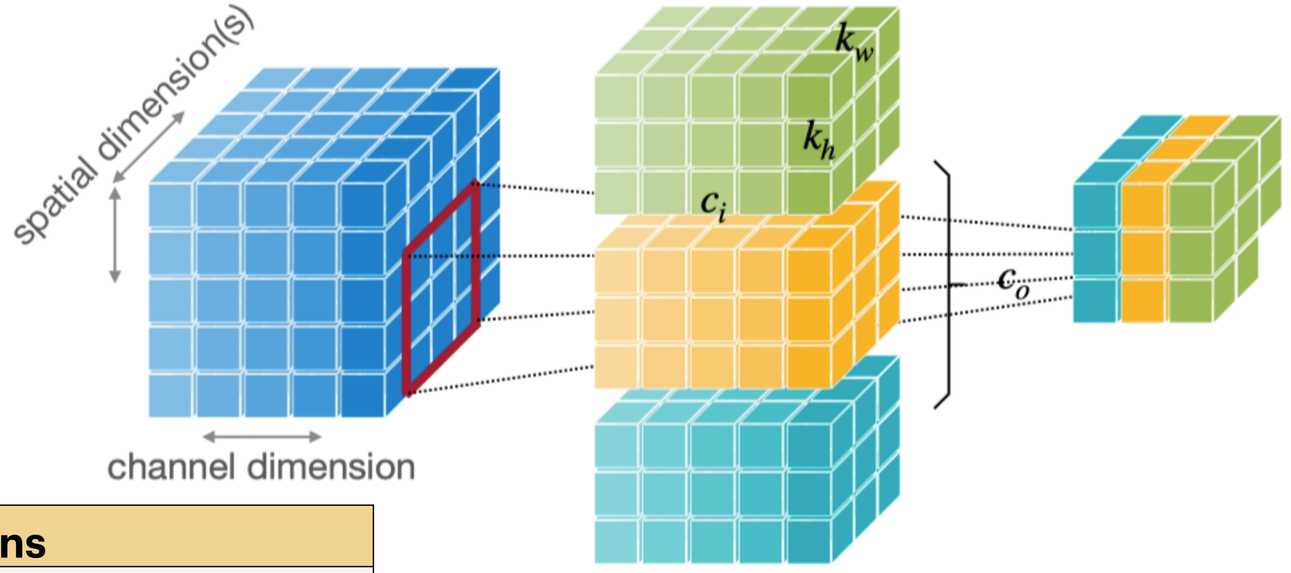


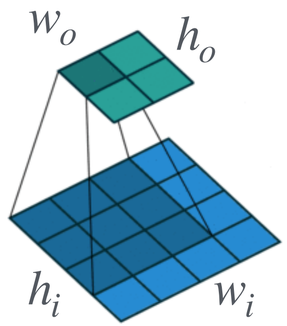
Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$



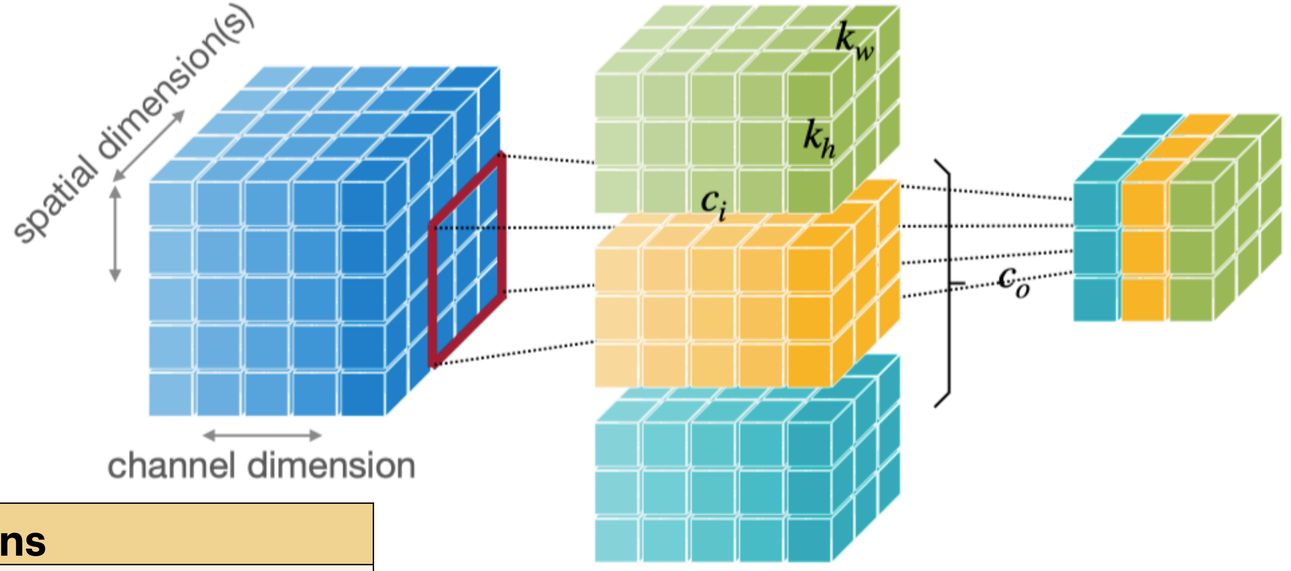


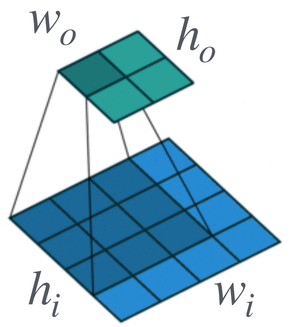
Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

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

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$



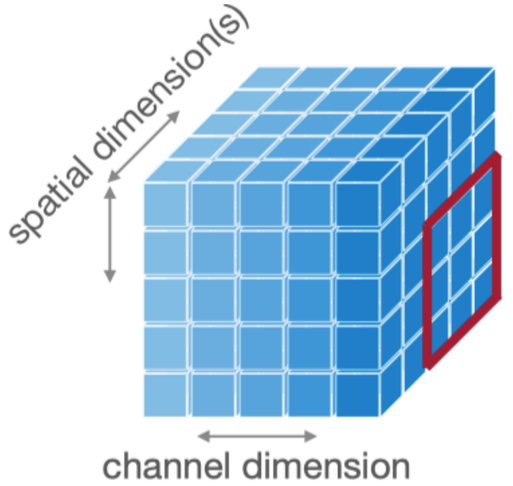


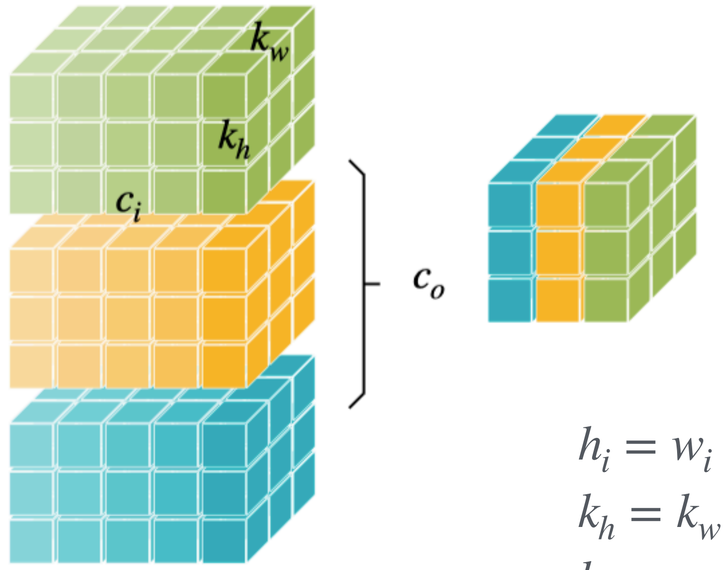
Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer (2D Conv)

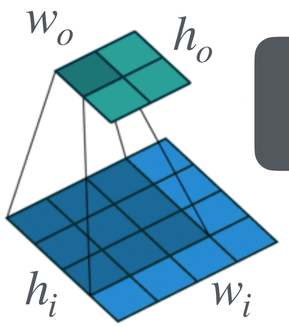
The output neuron is connected to input neurons in the receptive field

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : (c_o, c_i, k_h, k_w)$
  - Bias $b : (c_o,)$





Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width



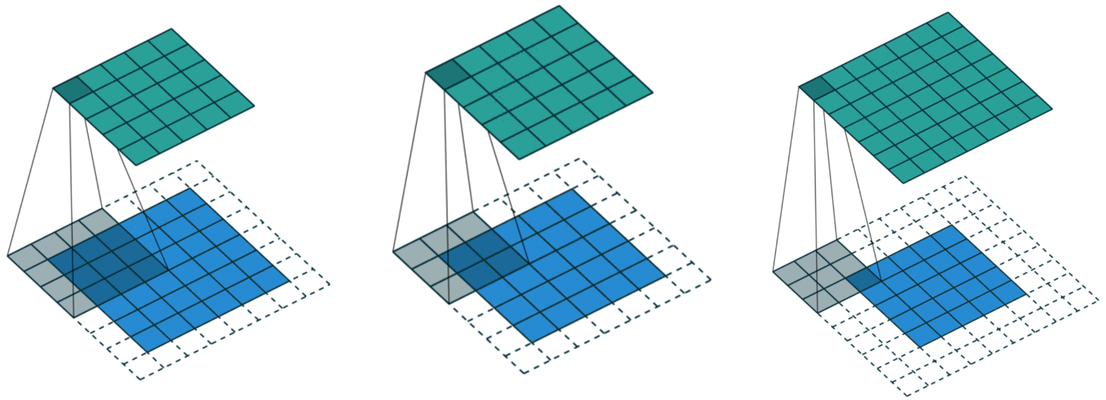
$$h_o = h_i - k_h + 1$$

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

Feature map size becomes smaller.

Convolution Layer: Padding

- Padding can be used to **keep** the output feature map **size the same** as the input feature map size (preserve dimensions)
- Padding enables the conversion filter to process the **edges of the input**
- Padding adds **flexibility** to control whether the output is smaller, equal, or larger than the input



Arbitrary padding | Same padding | Full padding

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

p is padding

$h_i = w_i = 6$
$k_h = k_w = 4$
$p = 1$
$h_o = w_o$
$= 6 + 2 \times 1 - 4 + 1$
$= 5$

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

$h_i = w_i = 5$
$k_h = k_w = 3$
$p = 2$
$h_o = w_o$
$= 5 + 2 \times 2 - 3 + 1$
$= 7$

Notations	
c_i	Input channels
c_o	Output channels
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Convolution Layer: Padding

- **Zero padding** pads the input boundaries with zero (default in PyTorch)
- Other paddings: Reflection padding, replication padding, constant padding (https://pytorch.org/docs/stable/nn.html#padding-layers)

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

Zero padding

9	8	7	8	9	8	7
6	5	4	5	6	5	4
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

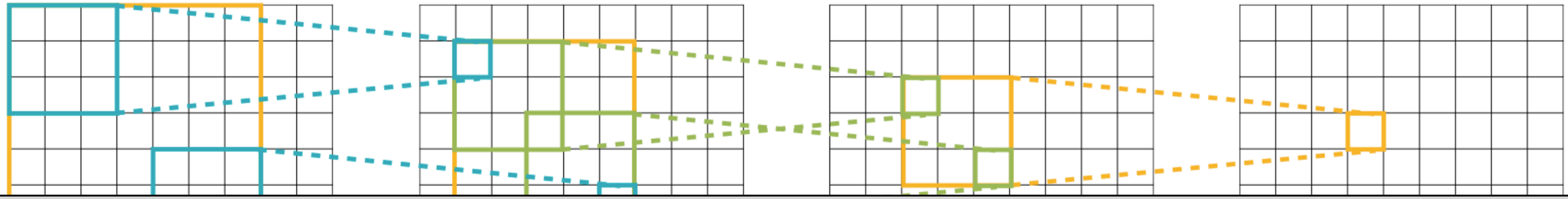
Reflection padding

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

Replication padding

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 $L \cdot (k - 1) + 1$

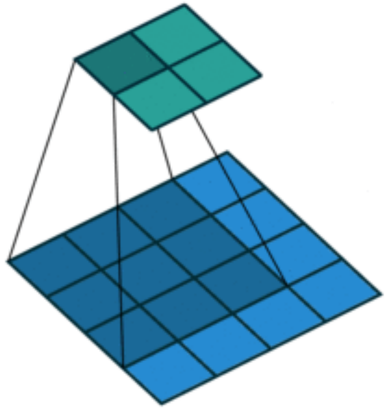


Problem: for large images, each output needs many layers to "see" the whole image
Solution: **Downsample** inside the neural network (e.g., stride, pooling)

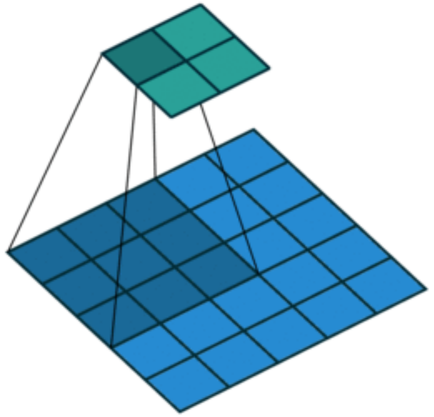
For $L = 2, k = 3$, the receptive field size is 5; for $L = 3, k = 3$, the receptive field size is 7

Convolution Layer: Stride

- Stride controls how much the convolutional filter moves across the input data
  - Reduce the size of the output feature map, decrease the number of computations and parameters, and reduce computational complexity



No stride (stride = 1)



Stride = 2

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

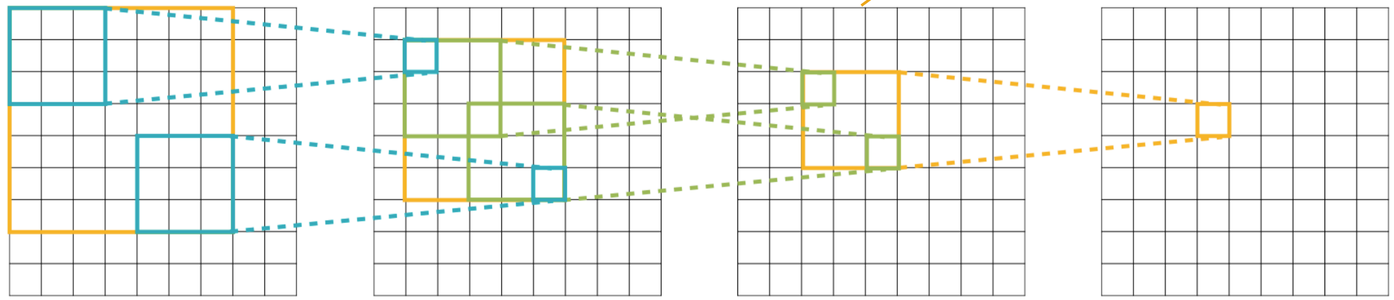
p is padding

s is stride

When stride $s = 2$

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

$$h_o = w_o = 3$$
$$k_h = k_w = 3$$
$$p = 0$$
$$h_i = w_i$$
$$= (3 - 1) \times 2 + 3$$
$$= 7$$

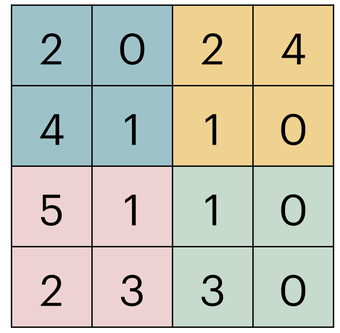


For $L = 2, k = 3$, the receptive field size is 5; for $L = 3, k = 3$, the receptive field size is 7

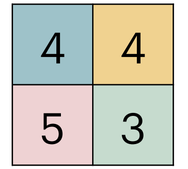
Convolution Layer: Pooling Layer

Downsample the feature map to a smaller size

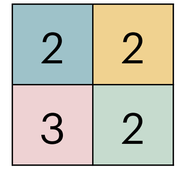
- The output neuron pools the features in the receptive field, similar to convolution
  - Usually, the stride is the same as the kernel size: $s = k$
- Pooling operates over each channel independently
  - 🤩 No learnable parameters



Stride = kernel size = 2



Max pooling



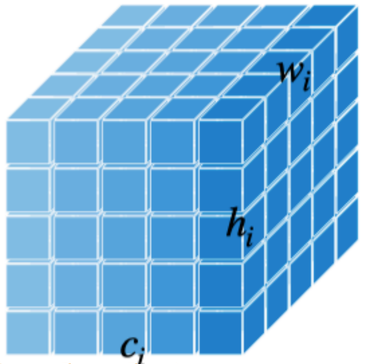
Average pooling

Input feature map **Output feature map**

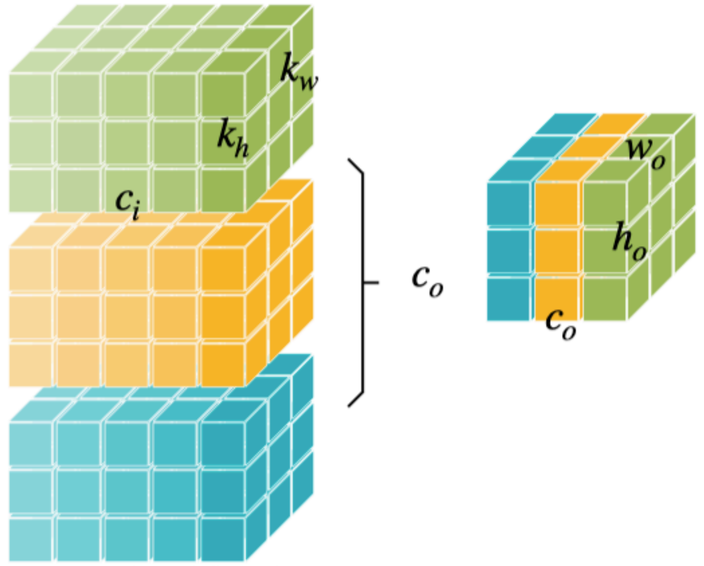
Grouped Convolution Layer

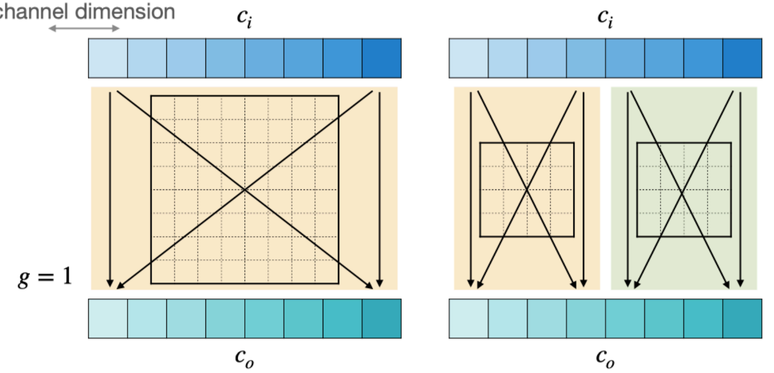
A group of narrower convolutions

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : \sout{(c_o, c_i, k_h, k_w)}$ $(c_o/g \cdot g, c_i/g, k_h, k_w)$
  - Bias $b : (c_o,)$



$c_i \to c_i/g$





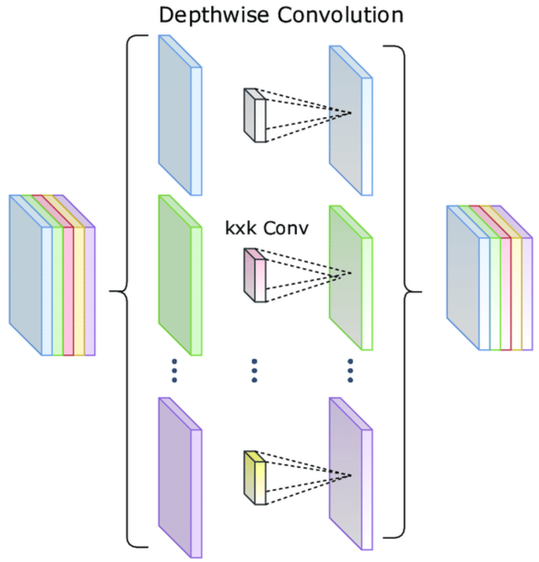
Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Depthwise Convolution Layer

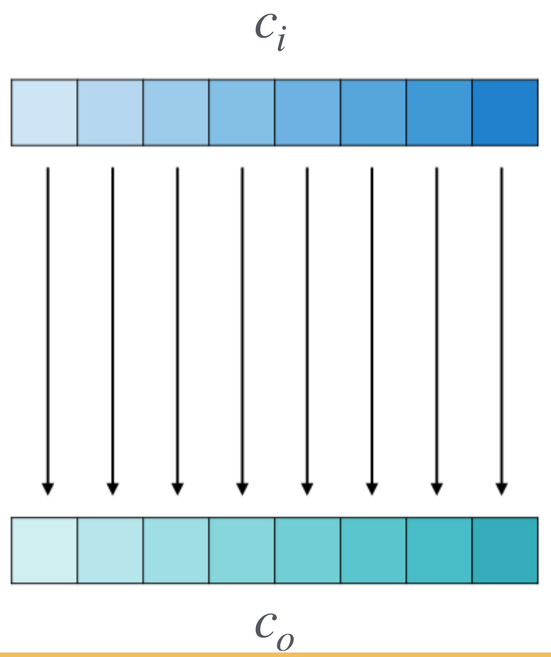
Independent filter for each channel: $g = c_i = c_o$ in grouped convolution

- Shape of tensors
  - Input features $X : (n, c_i, h_i, w_i)$
  - Output features $Y : (n, c_o, h_o, w_o)$
  - Weights $W : ~~(c_o/g \cdot g, c_i/g, k_h, k_w)~~ \; (c_o, 1, k_h, k_w)$
  - Bias $b : (c_o,)$

Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width



Source



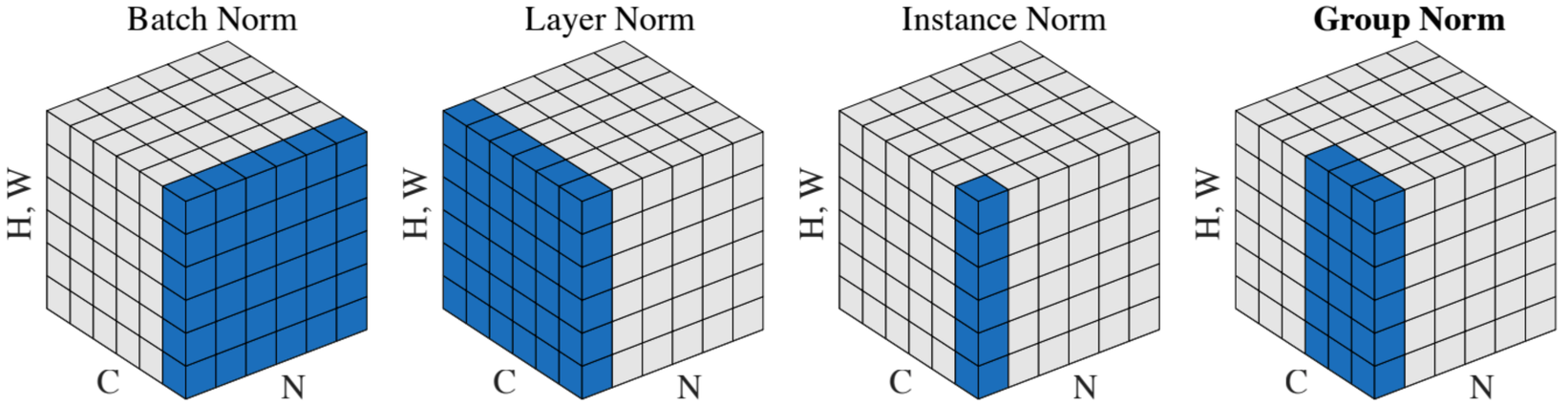
Normalization Layer

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

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

Normalizing the features makes optimization faster

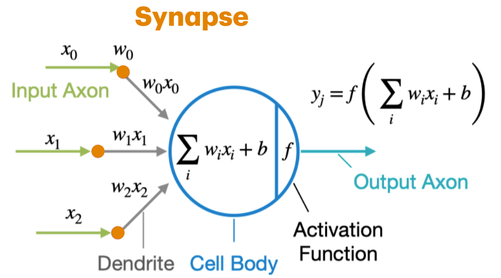
- Normalize the features according to: $\hat{x}_i = \frac{1}{\sigma}(x_i - \mu_i)$
  - μ_i is the mean, and σ_i is the standard deviation (std) over the set of pixels $\mathcal{S}_i$
- Then, learn a per-channel linear transform (trainable scale γ and shift β) to compensate for the possible loss of representational ability as follows: $y = \gamma_{i_c}\hat{x}_i + \beta_{i_c}$



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

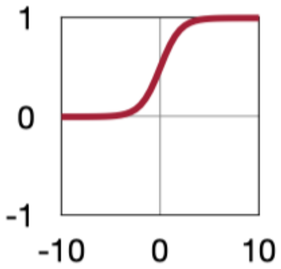
Ioffe, S. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Activation Function



Activation functions are typically non-linear functions

Sigmoid

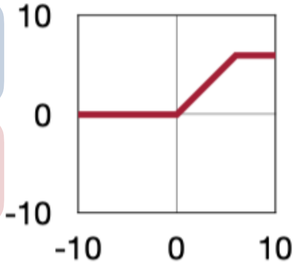


$y = 1/(1 + e^{-x})$

🤩 Easy to quantize

😖 Gradient vanishing

ReLU6

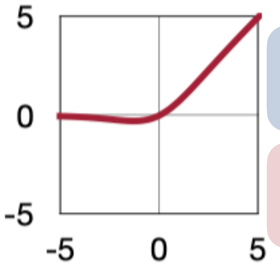


$y = \min\left(\max(0, x), 6\right)$

🤩 Constraint quantization range

😖 No gradient in negative

Swish

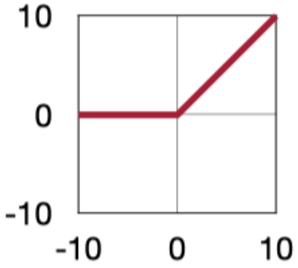


$y = x/(1 + e^{-x})$

🤩 Find by NAS

😖 Hard to implement

ReLU



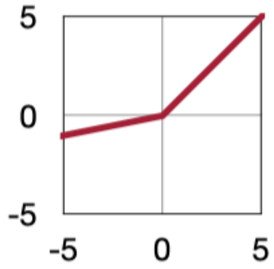
$y = \max(0, x)$

🤩 Easy to sparsity

😖 Hard to quantize

😖 No gradient in negative

Leaky ReLU

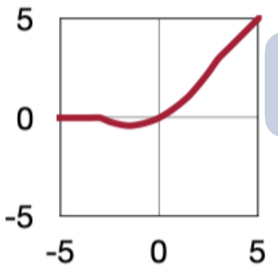


$y = \max(\alpha x, x)$

🤩 Has gradience for negative

😖 No sparsity benefit

Hard Swish

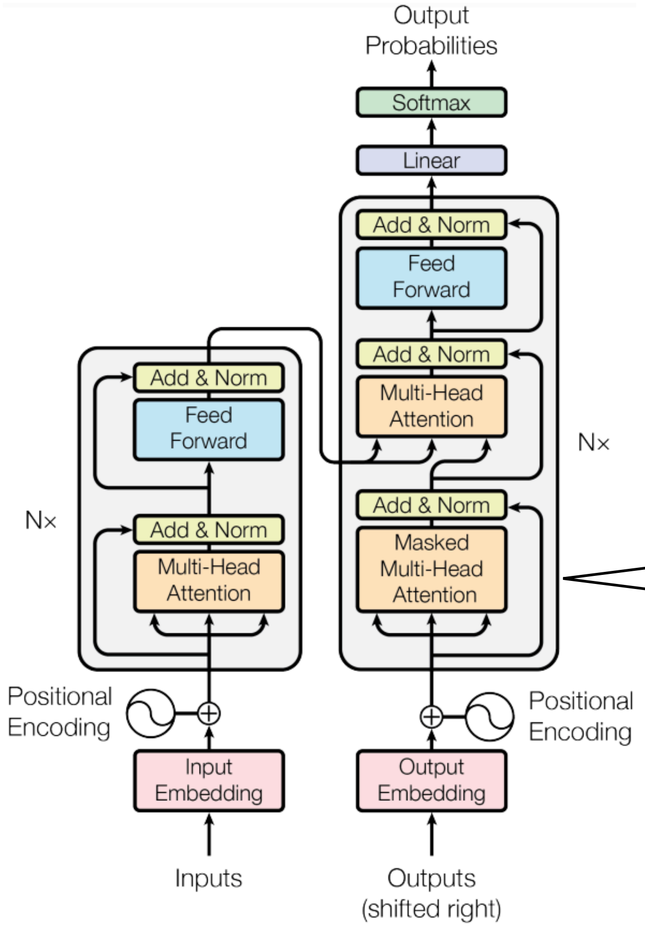


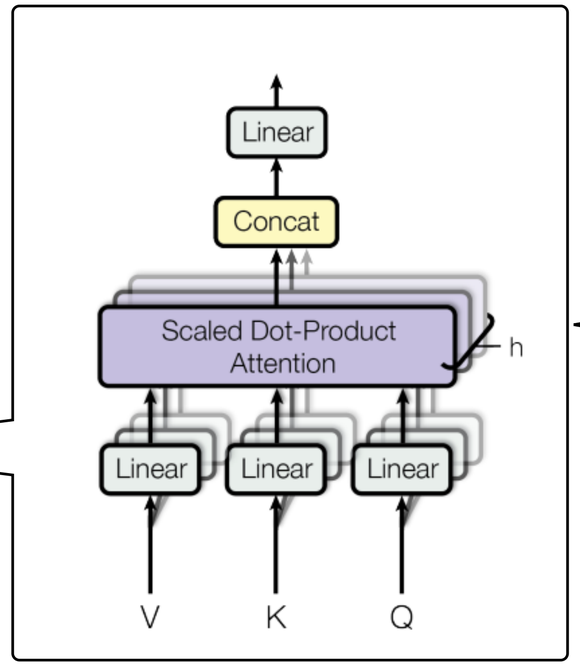
$$y = \begin{cases} 0, & x \le -3 \\ x, & x \ge 3 \\ x \cdot (x+3)/6, & otherwise \end{cases}$$

🤩 Easy to implement

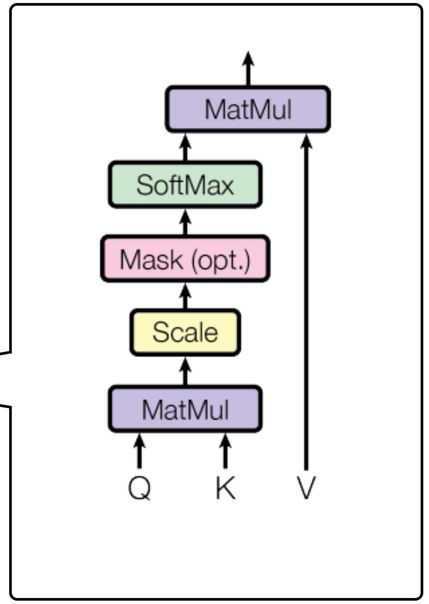
Transformer

We will have a separate lecture on transformer architecture in Nov.





Attention mechanism



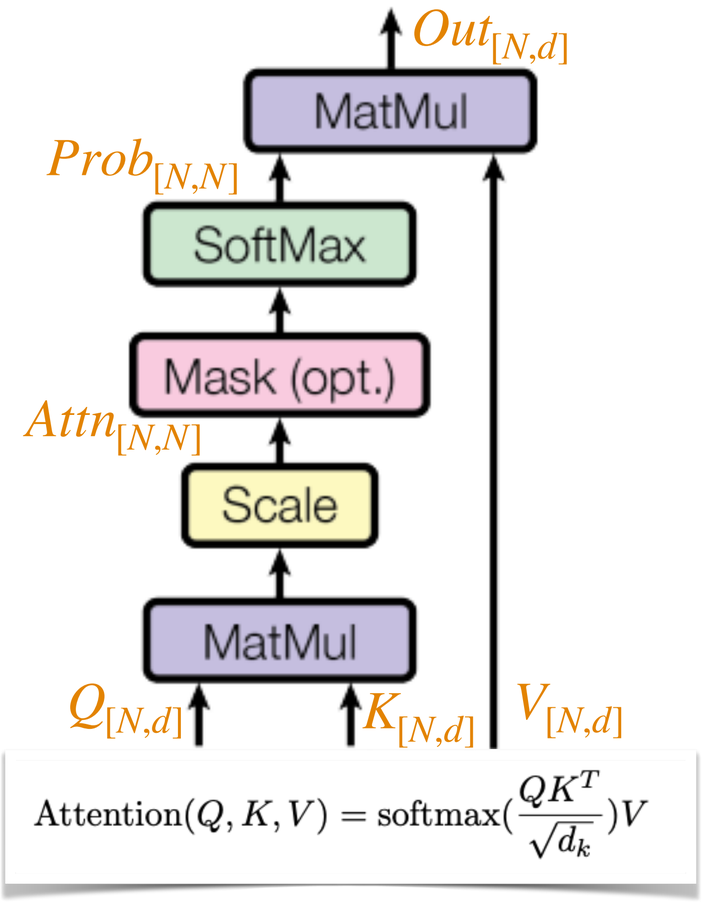
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Vaswani, A. (2017). Attention is all you need. Advances in Neural Information Processing Systems.

Attention in Transformer

We will have a separate lecture on transformer architecture in Nov.

- The Query-Key-Value design is analogous to a retrieval system (use YouTube search as an example)
  - Query: text prompt in the search bar
  - Key: the title/descriptions of videos
  - Value: the corresponding videos
- Multiply Q and K to get the inner product (normalize by $\sqrt{d}$)
- Followed by Softmax to get the attention weights of shape $N \times N$
- Multiply attention weights with V to get the output



Popular CNN Architectures

AlexNet

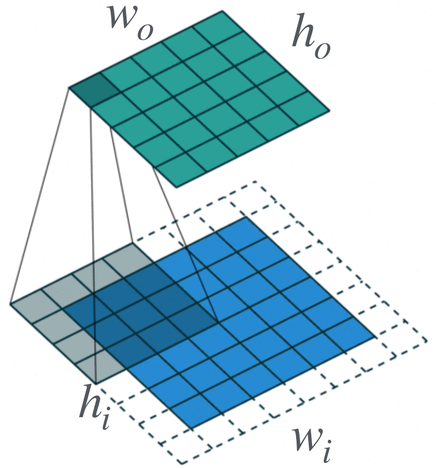
AlexNet	C x H x W	H, W
Image (3x224x224)	3x224x224	
11x11 Conv, channel 96, stride 4, pad 2	96x55x55	$\frac{224+2\times2-11}{4}+1=55$
3x3 MaxPool, stride 2	96x27x27	$\frac{55+0-3}{2}+1=27$
5x5 Conv, channel 256, pad 2, groups 2	256x27x27	$\frac{27+2\times2-5}{1}+1=27$
3x3 MaxPool, stride 2	256x13x13	$\frac{27+0-3}{2}+1=13$
3x3 Conv, channel 384, pad 1		
3x3 Conv, channel 384, pad 1, groups 2		
3x3 Conv, channel 256, pad 1, groups 2		
3x3 MaxPool, stride 2		
Linear, channel 4096		
Linear, channel 4096		
Linear, channel 1000		

🤔🤔🤔

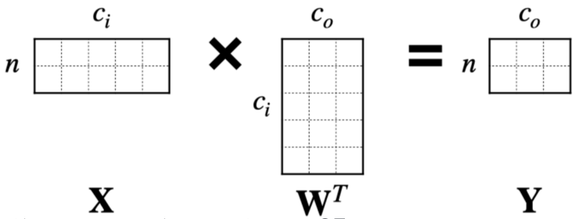
Convolution Layer/Pooling Layer

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

p is padding, s is stride



Linear Layer



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

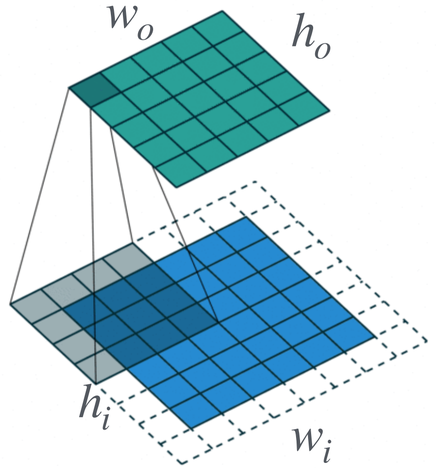
AlexNet

AlexNet	C x H x W	H, W
Image (3x224x224)	3x224x224	
11x11 Conv, channel 96, stride 4, pad 2	96x55x55	$\frac{224+2\times2-11}{4}+1=55$
3x3 MaxPool, stride 2	96x27x27	$\frac{55+0-3}{2}+1=27$
5x5 Conv, channel 256, pad 2, groups 2	256x27x27	$\frac{27+2\times2-5}{1}+1=27$
3x3 MaxPool, stride 2	256x13x13	$\frac{27+0-3}{2}+1=13$
3x3 Conv, channel 384, pad 1	384x13x13	$\frac{13+2\times1-3}{1}+1=13$
3x3 Conv, channel 384, pad 1, groups 2	384x13x13	$\frac{13+2\times1-3}{1}+1=13$
3x3 Conv, channel 256, pad 1, groups 2	256x13x13	$\frac{13+2\times1-3}{1}+1=13$
3x3 MaxPool, stride 2	256x6x6	$\frac{13+0-3}{2}+1=6$
Linear, channel 4096	4096	
Linear, channel 4096	4096	
Linear, channel 1000	1000	

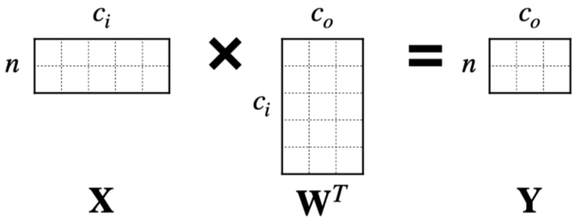
Convolution Layer/Pooling Layer

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

p is padding, s is stride



Linear Layer

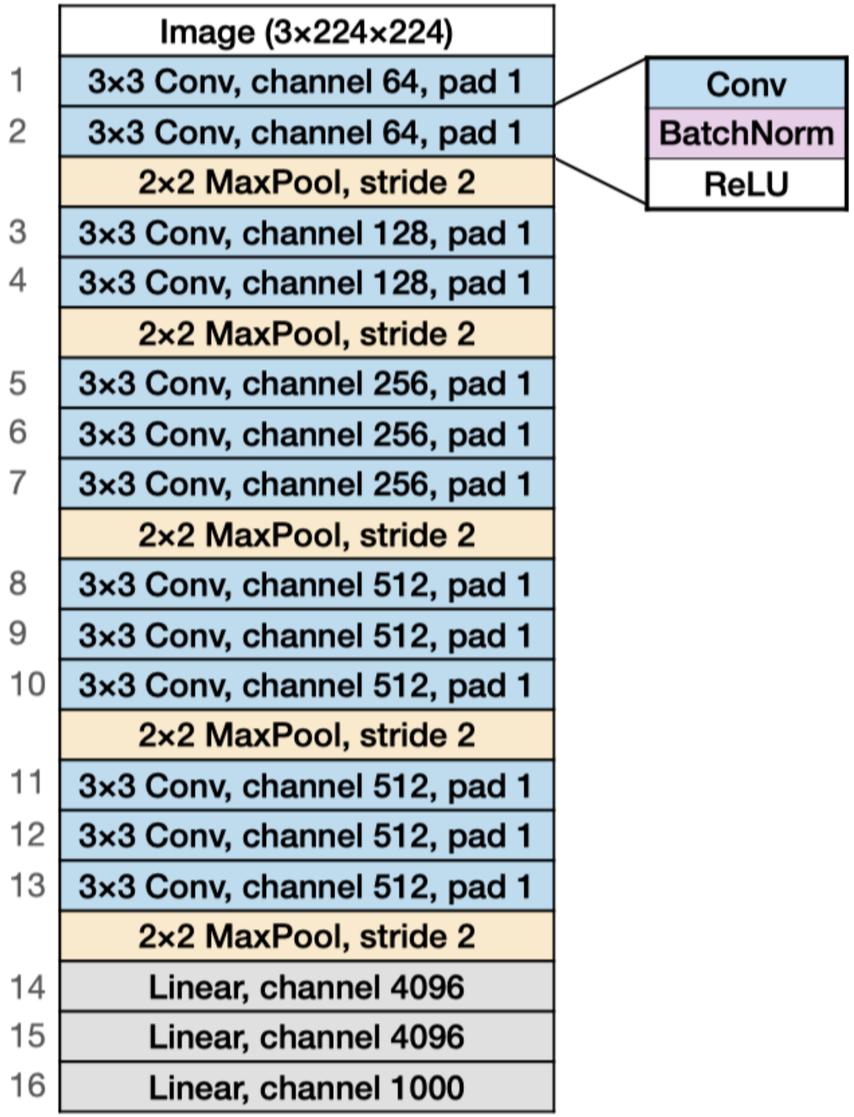


VGG-16

AlexNet

Image (3x224x224)
11x11 Conv, channel 96, stride 4, pad 2
3x3 MaxPool, stride 2
5x5 Conv, channel 256, pad 2, groups 2
3x3 MaxPool, stride 2
3x3 Conv, channel 384, pad 1
3x3 Conv, channel 384, pad 1, groups 2
3x3 Conv, channel 256, pad 1, groups 2
3x3 MaxPool, stride 2
Linear, channel 4096
Linear, channel 4096
Linear, channel 1000

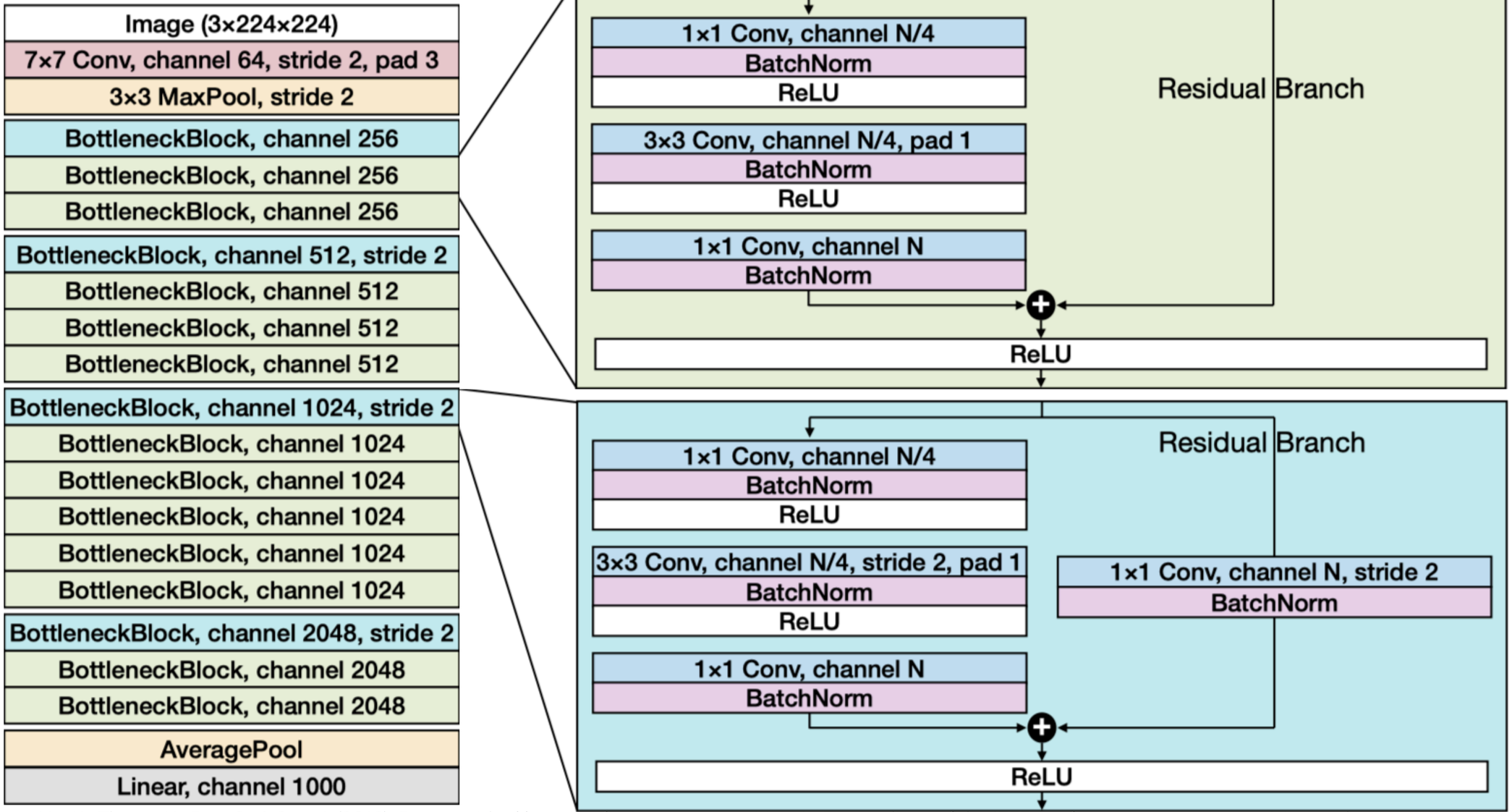
VGG-16



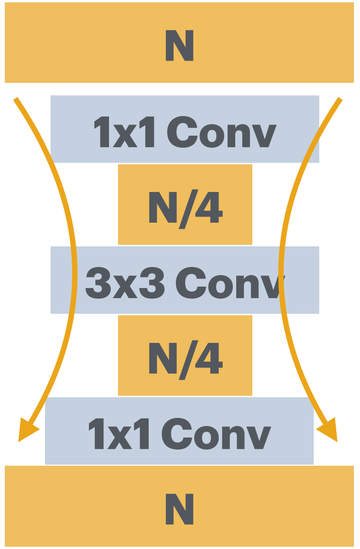
Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

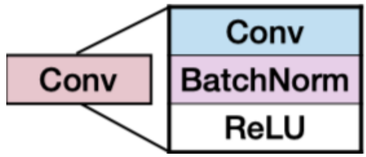
Figure source

ResNet-50



Feature Channels

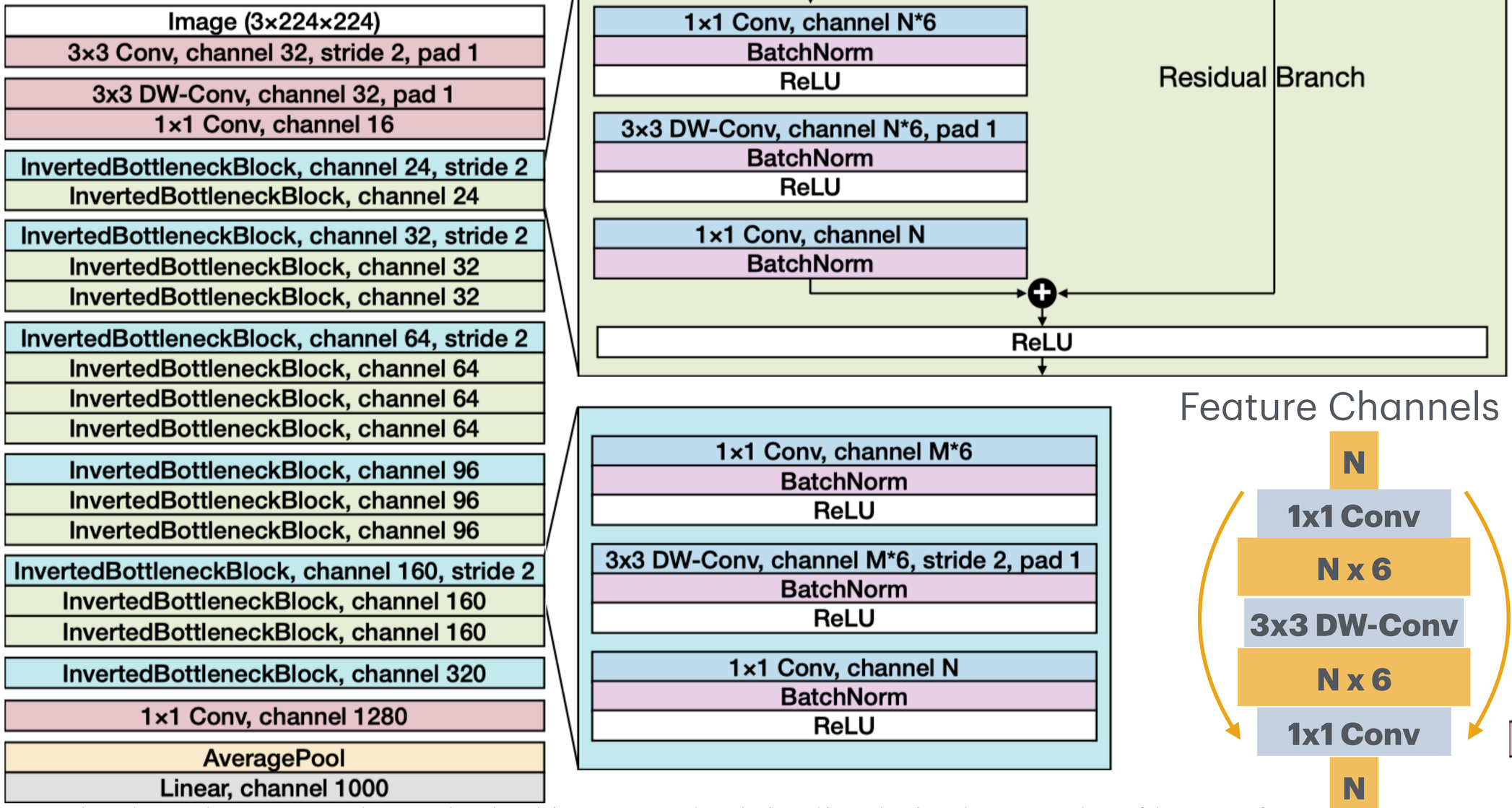




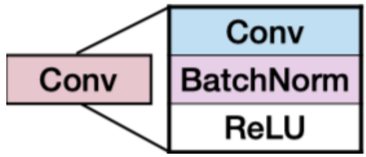
He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Figure source

MobileNetV2



Feature Channels



Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

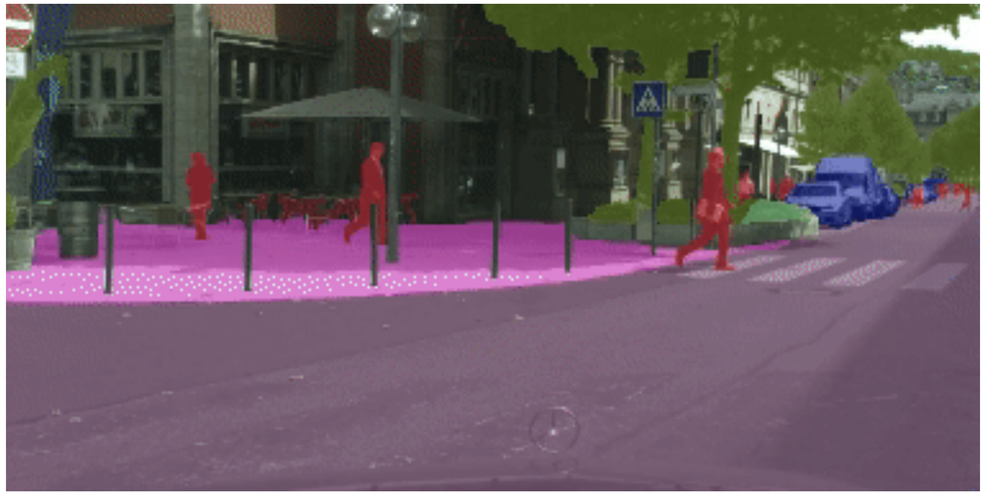
Figure source

Efficiency Metrics

How should we measure the efficiency of neural networks?

Latency and Throughput

- Latency measures the **delay** for a specific task



EfficientViT
46ms, 82.7mIOU
(Nvidia Jetson AGX Orin w/ TensorRT, FP16, batch size 1)

- Throughput measures the **rate** at which data is processed
  - $$\text{Throughput} = \frac{\text{Batch size}}{\text{Total time taken for the batch}}$$

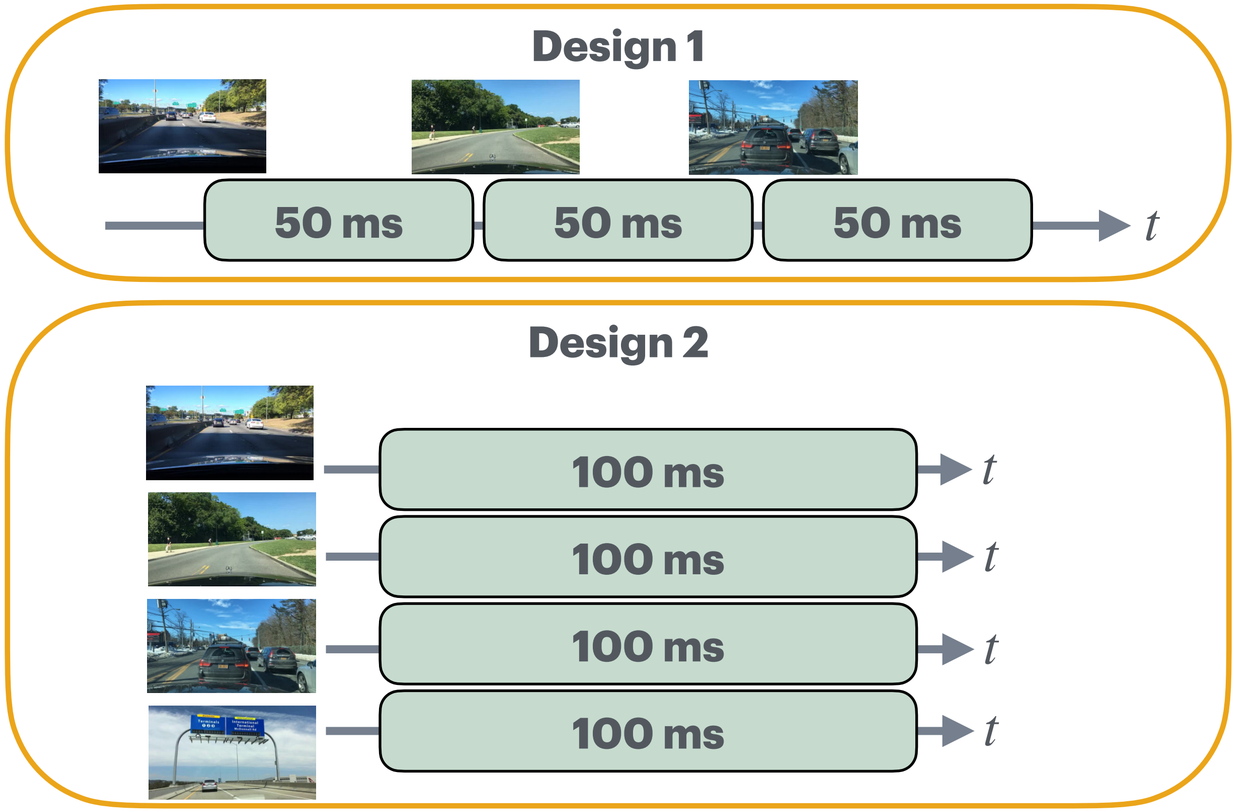
Cai, H., Li, J., Hu, M., Gan, C., & Han, S. (2022). Efficientvit: Multi-scale linear attention for high-resolution dense prediction. arXiv preprint arXiv:2205.14756.

What is the relationship between latency and throughput?

Does higher throughput translate to lower latency? Why?

Does lower latency translate to higher throughput? Why?

Latency vs. Throughput



Latency: 50ms
Throughput: 20 image/s

Latency: 100ms
Throughput: 40 image/s

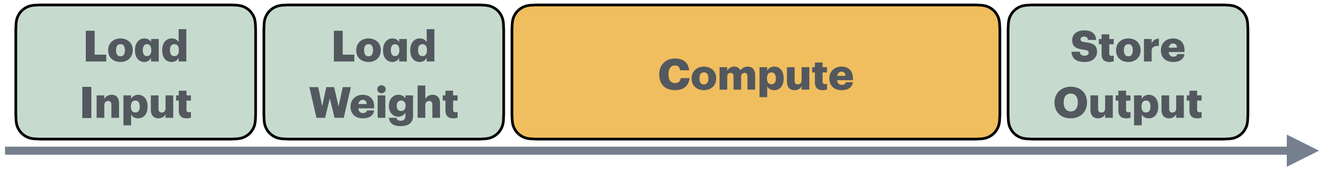
- Does higher throughput translate to lower latency? Why?
- Does lower latency translate to higher throughput? Why?

Which one is harder to improve?

Latency

NN specification

HW specification

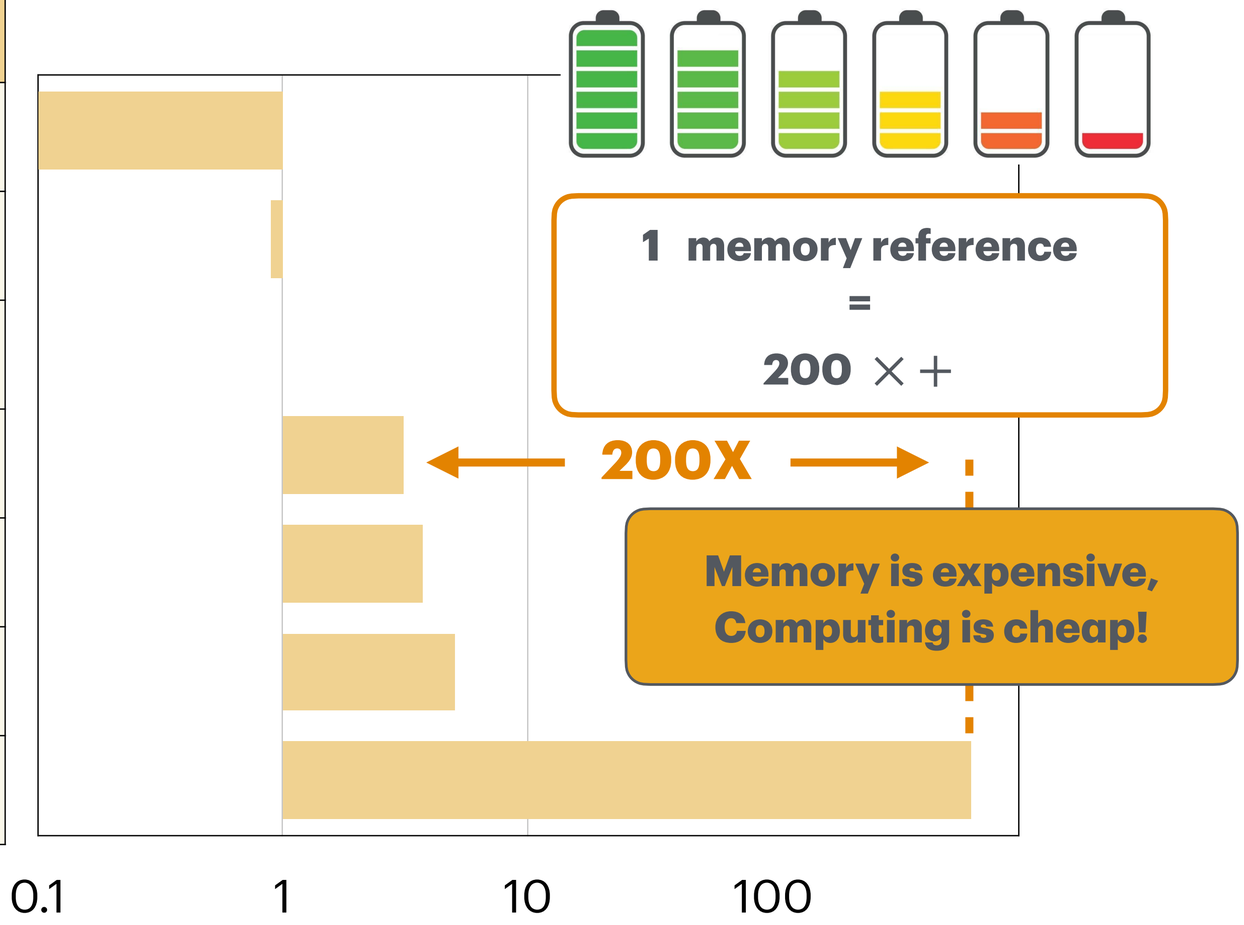


- $Latency \approx \max(T_{computation}, T_{memory})$
- $T_{computation} \approx \dfrac{\text{Number of operations in neural network model}}{\text{Number of operations that processor can process per second}}$
- $T_{memory} \approx T_{\text{data movement of activations}} + T_{\text{data movement of weights}}$
- $T_{\text{data movement of weights}} \approx \dfrac{\text{Neural network model size}}{\text{Memory bandwidth of processor}}$
- $T_{\text{data movement of activations}} \approx \dfrac{\text{Input activation size + Output activation size}}{\text{Memory bandwidth of processor}}$

Energy Consumption

Data movement → More memory reference → More energy

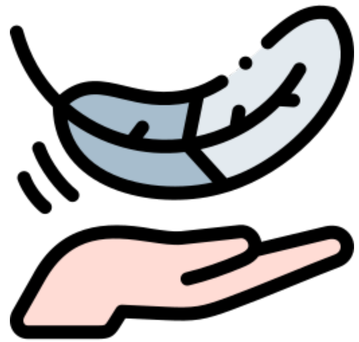
Operation	Energy (pJ)
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM memory	640



Horowitz, M. (2014, February). 1.1 computing's energy problem (and what we can do about it). In 2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC) (pp. 10-14). IEEE.

Efficiency of Neural Networks

Faster, smaller, greener



Memory related

 # parameters

Model size

Total/peak #activations

Computation related

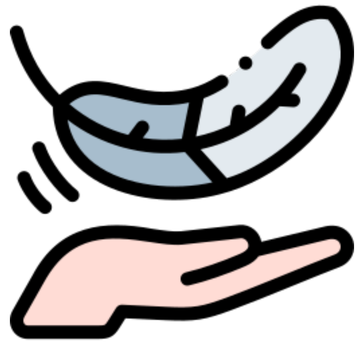
MAC

FLOP, FLOPS

OP, OPS

Efficiency of Neural Networks

Faster, smaller, greener



Memory related

 # parameters

Model size

Total/peak #activations

Computation related

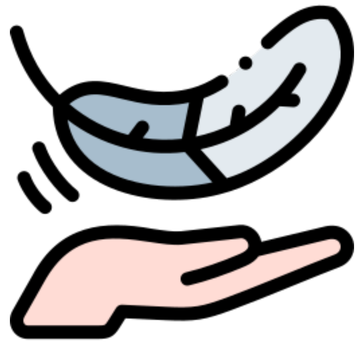
MAC

FLOP, FLOPS

OP, OPS

Efficiency of Neural Networks

Faster, smaller, greener



Memory related

 # parameters

Model size

Total/peak #activations

Computation related

MAC

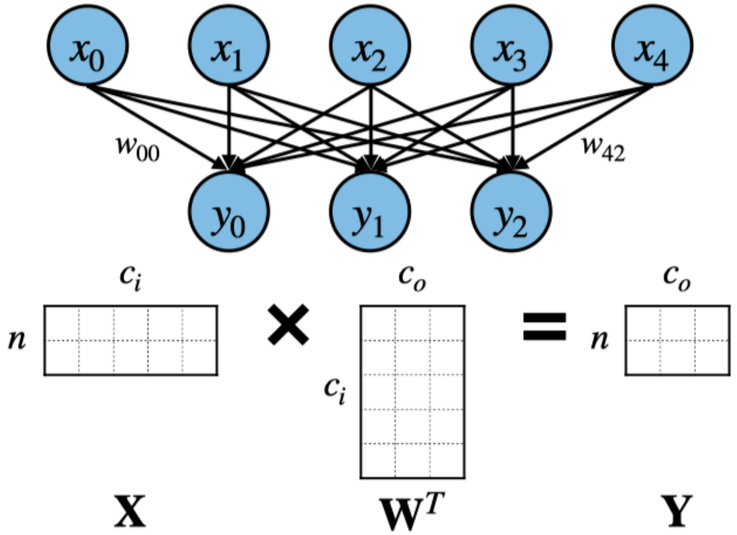
FLOP, FLOPS

OP, OPS

Parameters

Parameter (synapse/weight) count of the given neural network, i.e., # elements in the weight tensors

Layer	#Parameters (Bias is ignored)
Linear Layer	$c_o \cdot c_i$

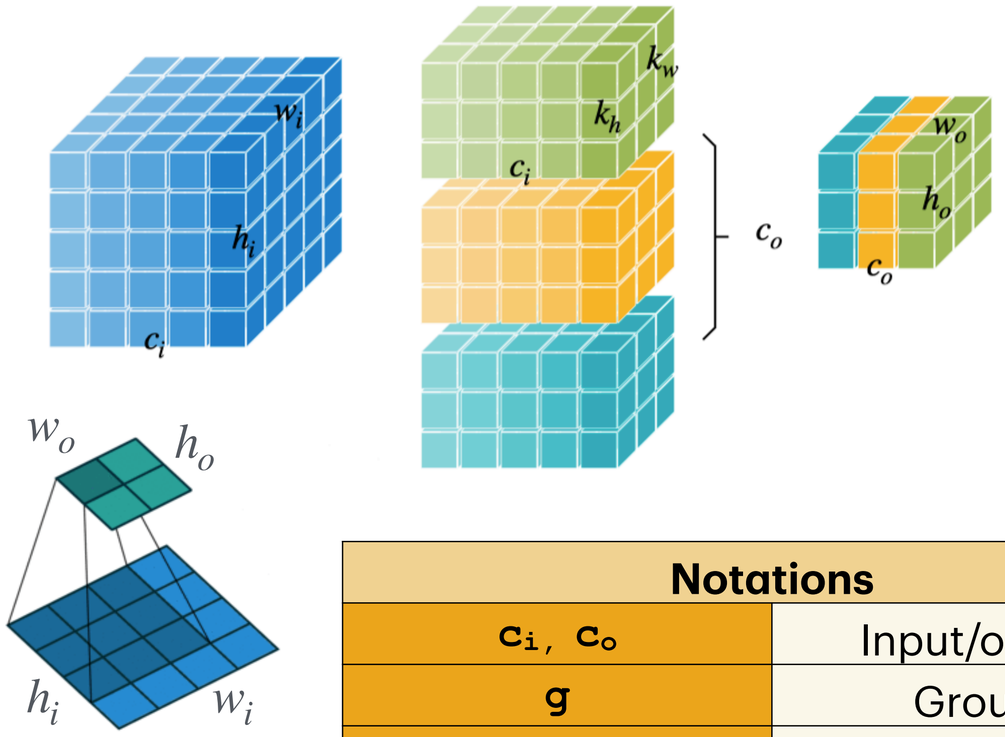


Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Parameters

Parameter (synapse/weight) count of the given neural network, i.e., # elements in the weight tensors

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

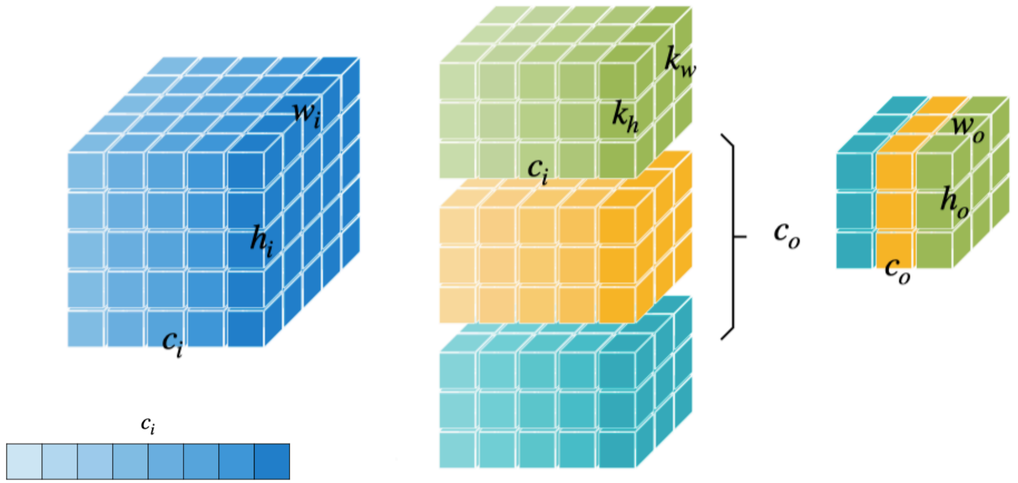


Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Parameters

Parameter (synapse/weight) count of the given neural network, i.e., # elements in the weight tensors

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	$c_o \cdot c_i/g \cdot k_h \cdot k_w$ $= c_o \cdot c_i \cdot k_h \cdot k_w/g$

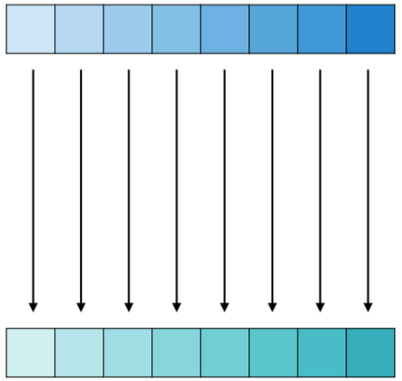


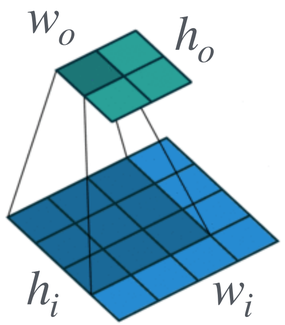
Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

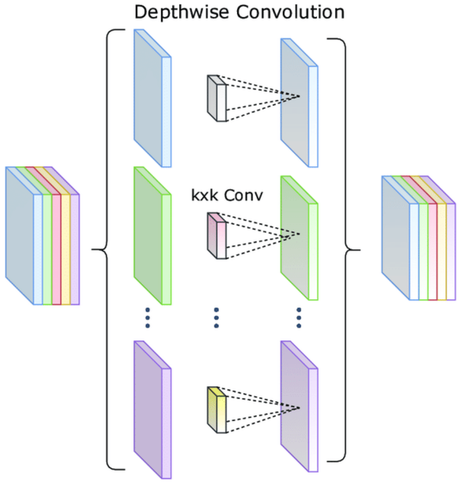
Parameters

Parameter (synapse/weight) count of the given neural network, i.e., # elements in the weight tensors

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	$c_o \cdot c_i \cdot k_h \cdot k_w / g$
Depthwise Convolution	$c_o \cdot k_h \cdot k_w$







Source

Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

AlexNet: #Parameters

AlexNet	C x H x W	# Parameters (bias is ignored)
Image (3x224x224)	3x224x224	
11x11 Conv, channel 96, stride 4, pad 2	96x55x55	96x3x11x11 = 34,848
3x3 MaxPool, stride 2	96x27x27	
5x5 Conv, channel 256, pad 2, groups 2	256x27x27	256x96x5x5/2 = 307,200
3x3 MaxPool, stride 2	256x13x13	
3x3 Conv, channel 384, pad 1	384x13x13	
3x3 Conv, channel 384, pad 1, groups 2	384x13x13	
3x3 Conv, channel 256, pad 1, groups 2	256x13x13	
3x3 MaxPool, stride 2	256x6x6	
Linear, channel 4096	4096	
Linear, channel 4096	4096	
Linear, channel 1000	1000	

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	$c_o \cdot k_h \cdot k_w$

AlexNet: #Parameters

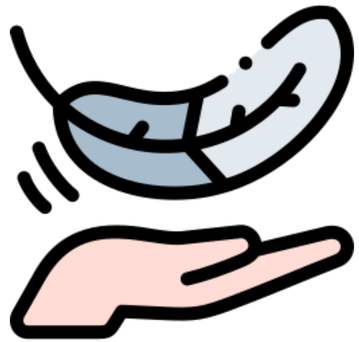
AlexNet	C x H x W	# Parameters (bias is ignored)
Image (3x224x224)	3x224x224	
11x11 Conv, channel 96, stride 4, pad 2	96x55x55	96x3x11x11 = 34,848
3x3 MaxPool, stride 2	96x27x27	
5x5 Conv, channel 256, pad 2, groups 2	256x27x27	256x96x5x5/2 = 307,200
3x3 MaxPool, stride 2	256x13x13	
3x3 Conv, channel 384, pad 1	384x13x13	384x256x3x3 = 884,736
3x3 Conv, channel 384, pad 1, groups 2	384x13x13	384x384x3x3/2 = 663,552
3x3 Conv, channel 256, pad 1, groups 2	256x13x13	256x384x3x3/2 = 442,368
3x3 MaxPool, stride 2	256x6x6	
Linear, channel 4096	4096	4096x(256x6x6) = 37,748,736
Linear, channel 4096	4096	4096x4096 = 16,777,216
Linear, channel 1000	1000	1000x4096 = 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

Efficiency of Neural Networks

Faster, smaller, greener



Memory related

 # parameters

Model size

Total/peak #activations

Computation related

MAC

FLOP, FLOPS

OP, OPS

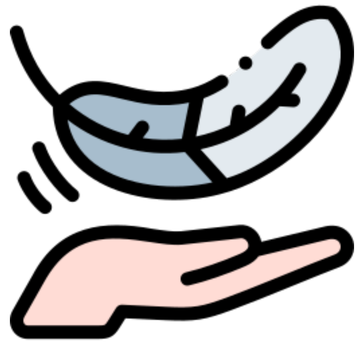
Model Size

Measures the storage for the weights of the given neural network. Common units: MB, KB, bits

- In general, if the whole neural network uses the same data type (e.g., floating-point)
  - Model size = #Parameters · Bit Width
- AlexNet has 61M parameters
  - If all weights are stored with 32-bit numbers, total storage will be about
    - $61M \times 4\text{Bytes}\ (32\text{bits}) = 244MB(244 \times 10^6\text{Bytes})$
  - If all weights are stored with 8-bit numbers, total storage will be about
    - $61M \times 1\text{Bytes}(8\text{bits}) = 61MB$

Efficiency of Neural Networks

Faster, smaller, greener



Memory related

 # parameters

Model size

Total/peak #activations

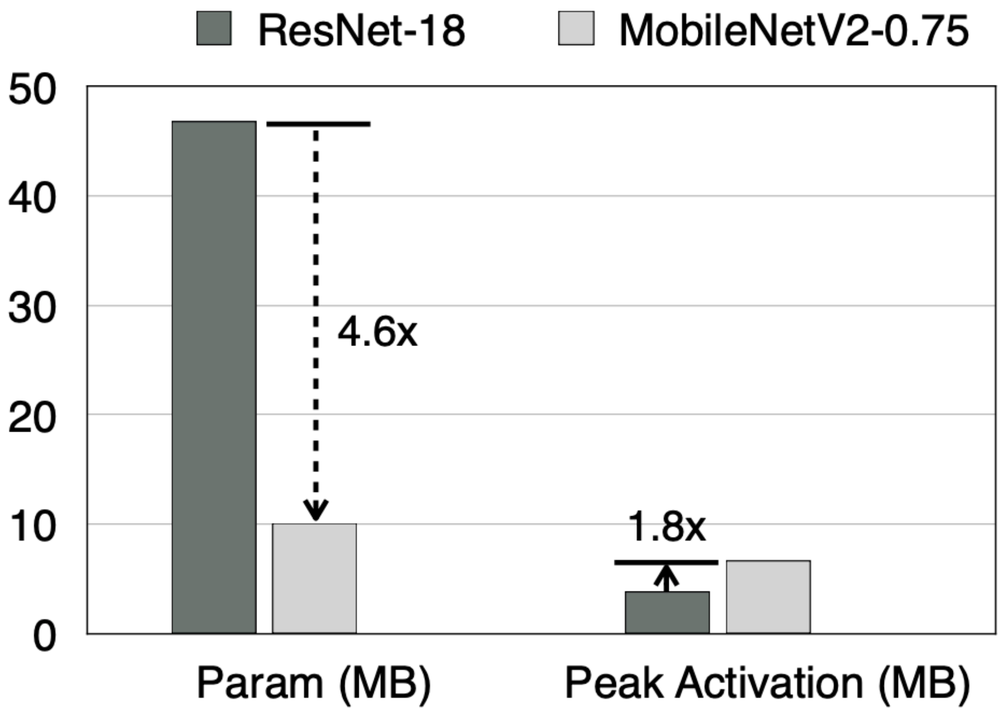
Computation related

MAC

FLOP, FLOPS

OP, OPS

Activations



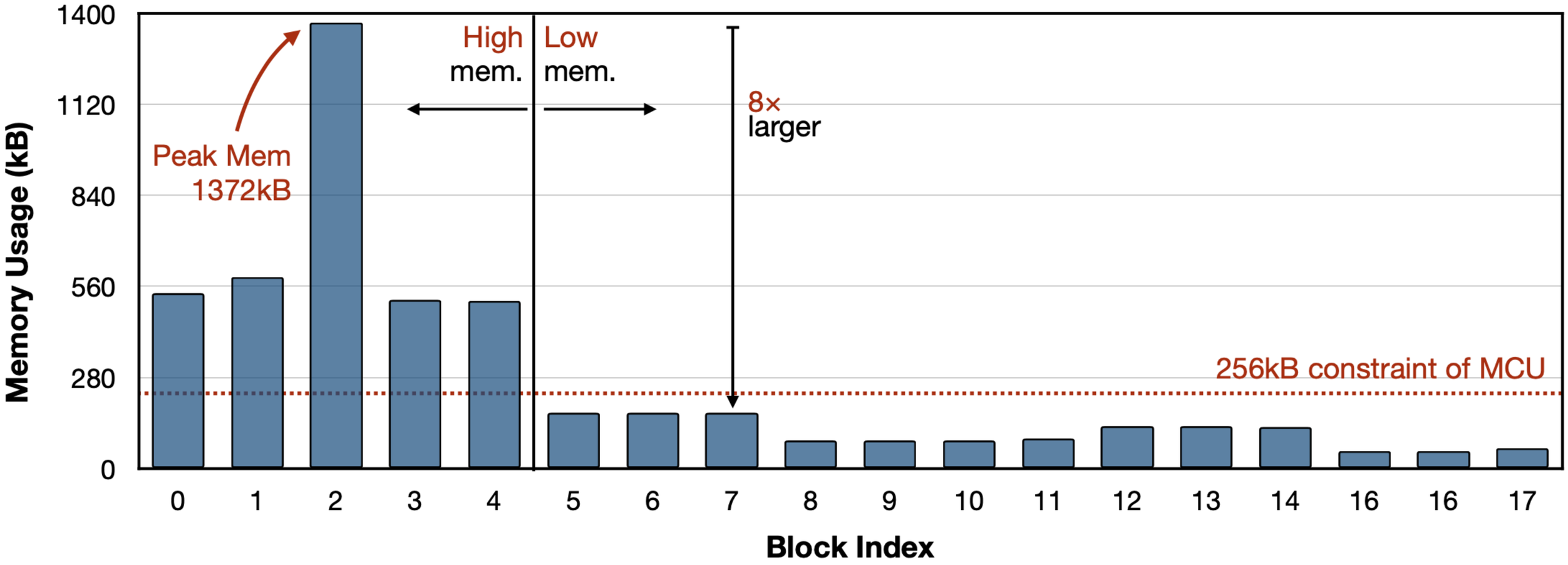
- From ResNet to MobileNet-v2, the model size was reduced, but not the activation size.

The number of Activations is the memory bottleneck in CNN inference, not #Parameters.

Lin, J., Chen, W. M., Lin, Y., Gan, C., & Han, S. (2020). Mcunet: Tiny deep learning on iot devices. Advances in neural information processing systems, 33, 11711-11722.

Activations

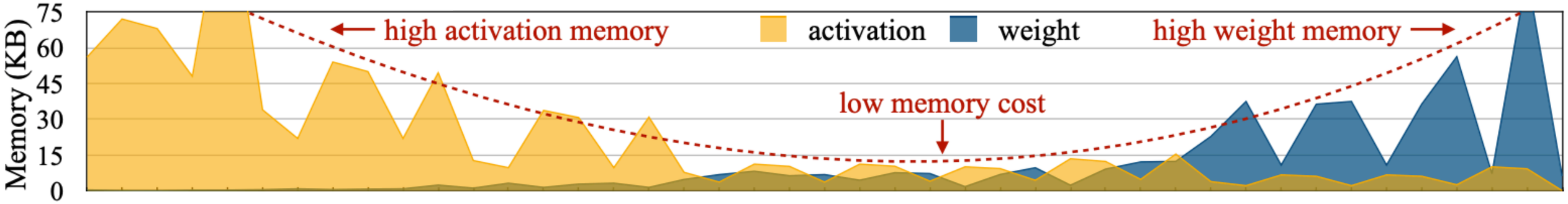
Imbalanced memory distribution of MobileNetV2



Lin, J., Chen, W. M., Cai, H., Gan, C., & Han, S. (2021). Memory-efficient patch-based inference for tiny deep learning. *Advances in Neural Information Processing Systems*, *34*, 2346-2358.

Activations

Activation and weight memory distribution of MCUNet



Per layer memory usage

🤔

Can you explain this U-shape pattern?

Lin, J., Zhu, L., Chen, W. M., Wang, W. C., Gan, C., & Han, S. (2022). On-device training under 256kb memory. Advances in Neural Information Processing Systems, 35, 22941-22954.

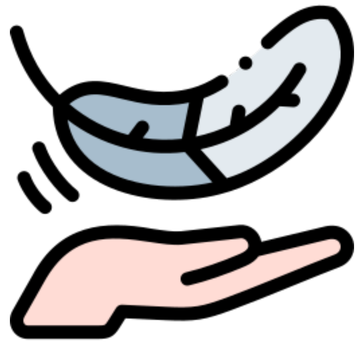
AlexNet: #Activations

AlexNet	C x H x W	#Activations
Image (3x224x224)	3x224x224	= 150,528
11x11 Conv, channel 96, stride 4, pad 2	96x55x55	= 290,400
3x3 MaxPool, stride 2	96x27x27	= 69,984
5x5 Conv, channel 256, pad 2, groups 2	256x27x27	= 186,624
3x3 MaxPool, stride 2	256x13x13	= 43,264
3x3 Conv, channel 384, pad 1	384x13x13	= 64,896
3x3 Conv, channel 384, pad 1, groups 2	384x13x13	= 64,896
3x3 Conv, channel 256, pad 1, groups 2	256x13x13	= 43,264
3x3 MaxPool, stride 2	256x6x6	= 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

Efficiency of Neural Networks

Faster, smaller, greener



Memory related

 # parameters

Model size

Total/peak #activations

Computation related

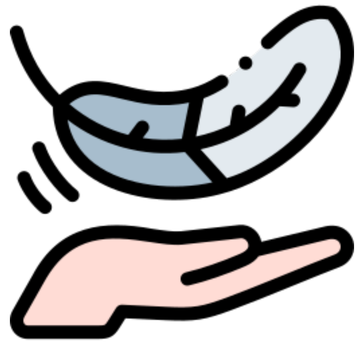
MAC

FLOP, FLOPS

OP, OPS

Efficiency of Neural Networks

Faster, smaller, greener



Memory related

 # parameters

Model size

Total/peak #activations

Computation related

MAC

FLOP, FLOPS

OP, OPS

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)
  - $MACs = m \cdot n \cdot k$

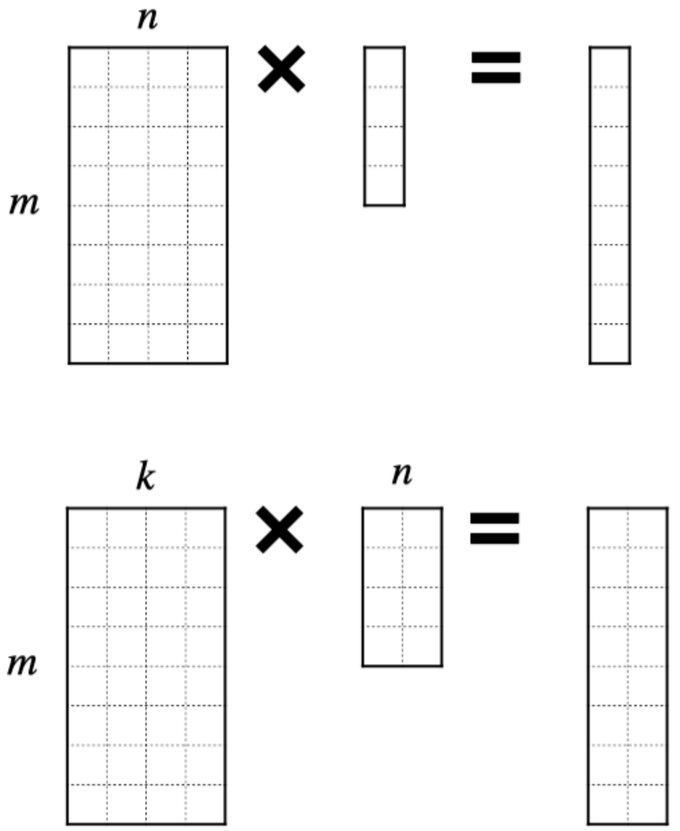
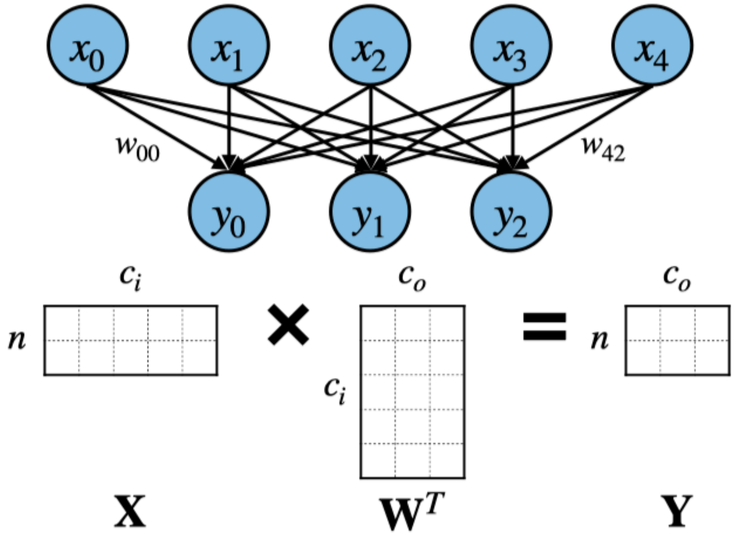


Figure source

Multiply-Accumulate Operations

MAC

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

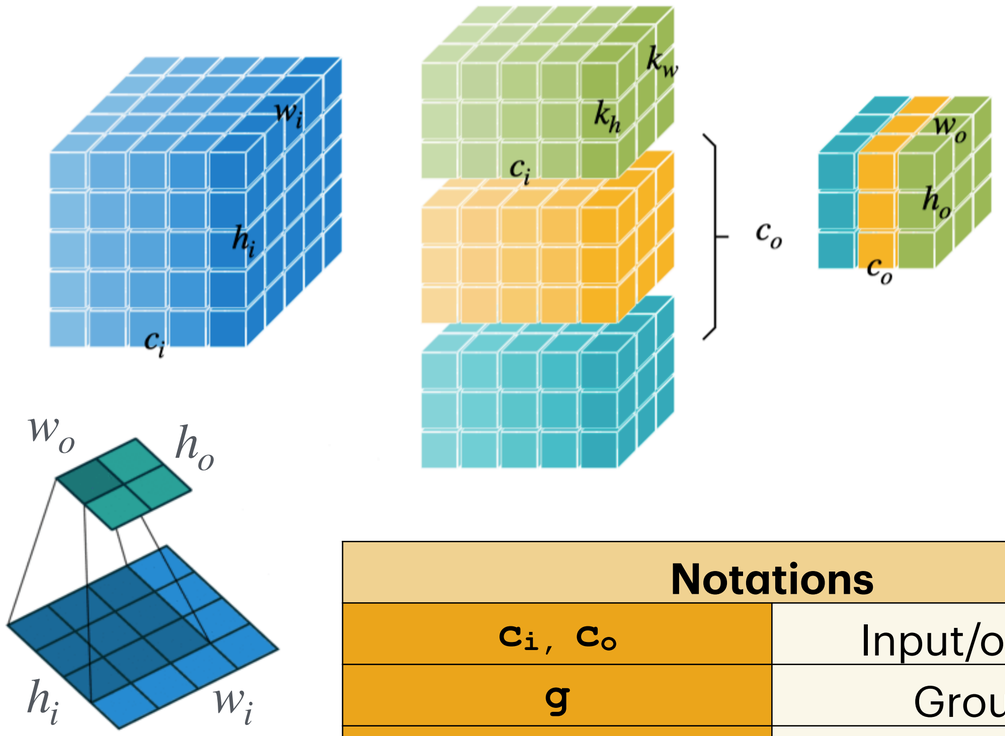


Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Multiply-Accumulate Operations

MAC

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

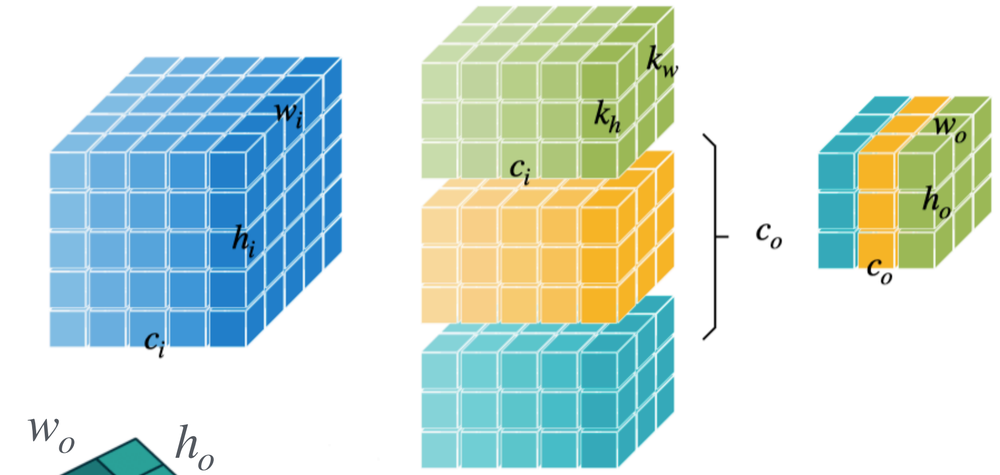


Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

Multiply-Accumulate Operations

MAC

Layer	#Parameters (Bias is ignored)
Linear Layer	$c_o \cdot c_i$
Convolution	$c_i \cdot k_h \cdot k_w \cdot c_o \cdot h_o \cdot w_o$
Grouped Convolution	$c_i/g \cdot k_h \cdot k_w \cdot c_o \cdot h_o \cdot w_o$

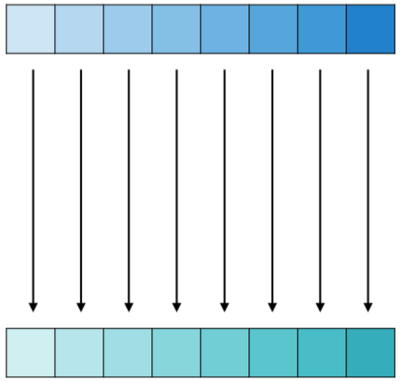


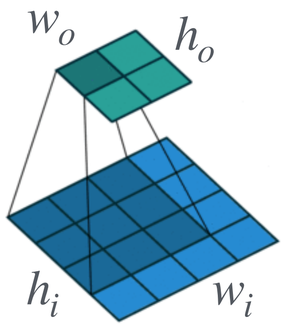
Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

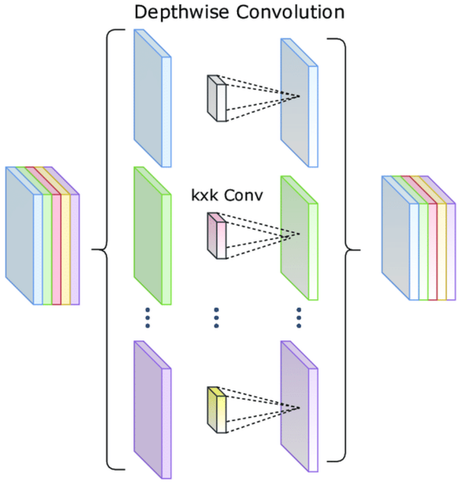
Multiply-Accumulate Operations

MAC

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







Source

Notations	
c_i, c_o	Input/output
g	Groups
n	Batch size
w_i, w_o	Input/output width
h_i, h_o	Input/output height
k_h, k_w	Kernel height/width

AlexNet: #MACs

AlexNet	C x H x W	#MACs
Image (3x224x224)	3x224x224	
11x11 Conv, channel 96, stride 4, pad 2	96x55x55	= 3x11x11x96x55x55 = 105,415,200
3x3 MaxPool, stride 2	96x27x27	
5x5 Conv, channel 256, pad 2, groups 2	256x27x27	= 96/2x5x5x256x27x27 = 223,948,800
3x3 MaxPool, stride 2	256x13x13	
3x3 Conv, channel 384, pad 1	384x13x13	
3x3 Conv, channel 384, pad 1, groups 2	384x13x13	
3x3 Conv, channel 256, pad 1, groups 2	256x13x13	
3x3 MaxPool, stride 2	256x6x6	
Linear, channel 4096	4096	
Linear, channel 4096	4096	
Linear, channel 1000	1000	

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

🤔🤔🤔

AlexNet: #MACs

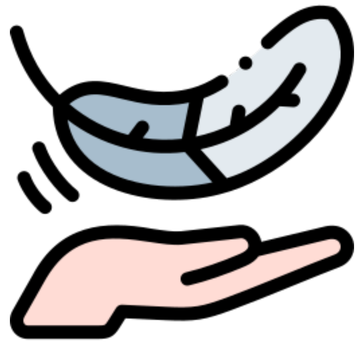
AlexNet	C x H x W	#MACs
Image (3x224x224)	3x224x224	
11x11 Conv, channel 96, stride 4, pad 2	96x55x55	= 3x11x11x96x55x55 = 105,415,200
3x3 MaxPool, stride 2	96x27x27	
5x5 Conv, channel 256, pad 2, groups 2	256x27x27	= 96/2x5x5x256x27x27 = 223,948,800
3x3 MaxPool, stride 2	256x13x13	
3x3 Conv, channel 384, pad 1	384x13x13	= 256x3x3x384x13x13 = 149,520,384
3x3 Conv, channel 384, pad 1, groups 2	384x13x13	= 384/2x3x3x384x13x13 = 112,140,288
3x3 Conv, channel 256, pad 1, groups 2	256x13x13	= 384/2x3x3x256x13x13 = 74,760,192
3x3 MaxPool, stride 2	256x6x6	
Linear, channel 4096	4096	= 4096x(256x6x6) = 37,748,736
Linear, channel 4096	4096	= 4096 x 4096 = 16,777,216
Linear, channel 1000	1000	= 1000 x 4096 = 4,096,000

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

~724M in total

Efficiency of Neural Networks

Faster, smaller, greener



Memory related

 # parameters

Model size

Total/peak #activations

Computation related

MAC

FLOP, FLOPS

OP, OPS

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)
  - AlexNet has 724M MACs, total number of floating point operations are
    - $724M \times 2 = 1.4G$ FLOPS
- Floating Point Operation Per Second (FLOPS)
  - $\text{FLOPS} = \frac{\text{FLOPs}}{\text{second}}$

What if the model is quantized to int8?

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
  - AlexNet has 724M MACs, total number of operations will be
    - $724M \times 2 = 1.4G$ OPs
  - Similarly, Operation Per Second (OPS)
    - $\text{OPS} = \frac{\text{OPs}}{\text{second}}$

Warmup Lab

Tutorial on PyTorch and Laboratory Exercises

References

- Convolution arithmetic [Github Repo]
- Image Classification with CNNs [Stanford CS231n Lecture 5]
- Basics of Neural Networks [MIT 6.5940]
- Wu, Y., & He, K. (2018). Group normalization. In Proceedings of the European conference on computer vision (ECCV) (pp. 3-19).
- Ioffe, S. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
- Vaswani, A. (2017). Attention is all you need. Advances in Neural Information Processing Systems.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.
- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).
- Deng, L., Li, G., Han, S., Shi, L., & Xie, Y. (2020). Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE, 108(4), 485-532.