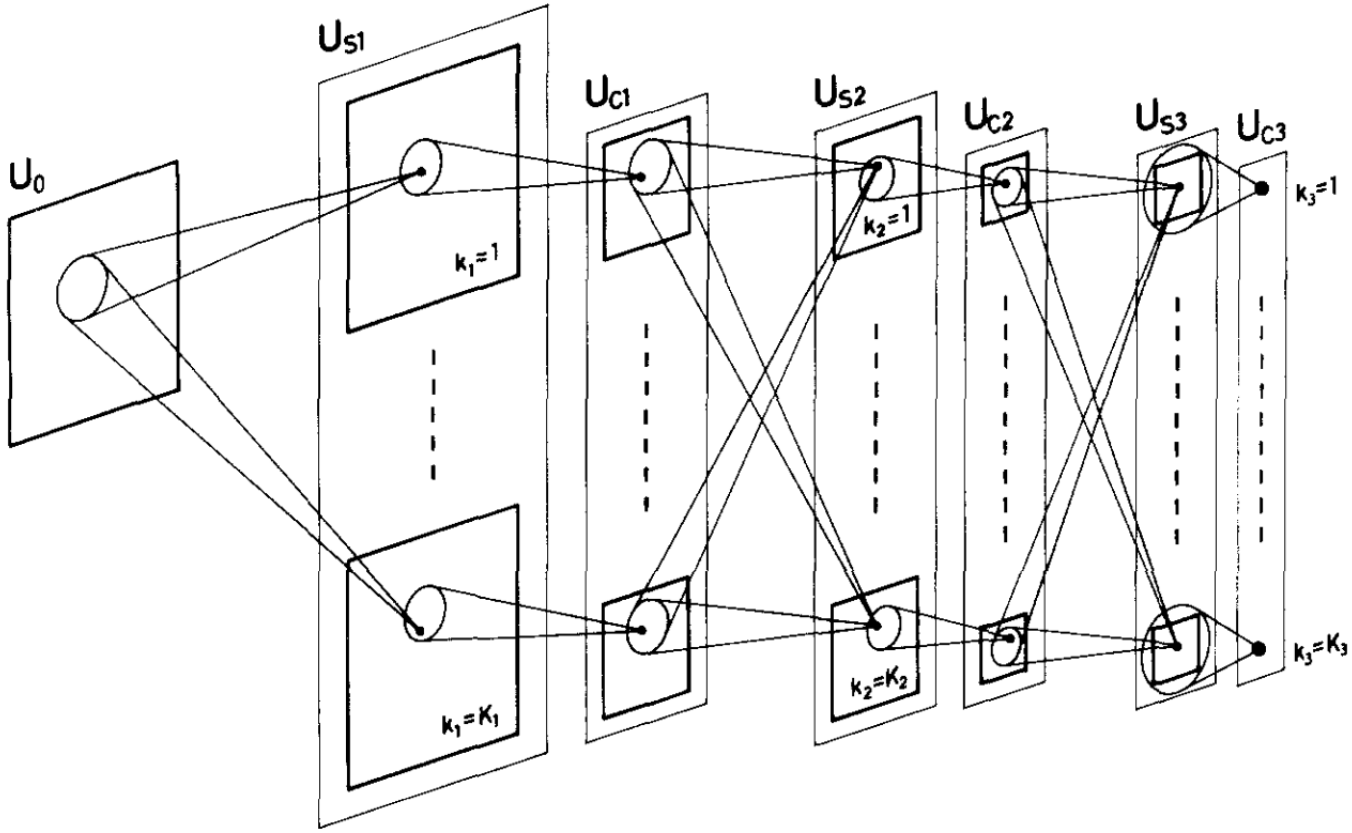
Lecture 9

Convolutional Neural Networks



Sara Beery, Kaiming He, Mina Konaković Luković, Vincent Sitzmann

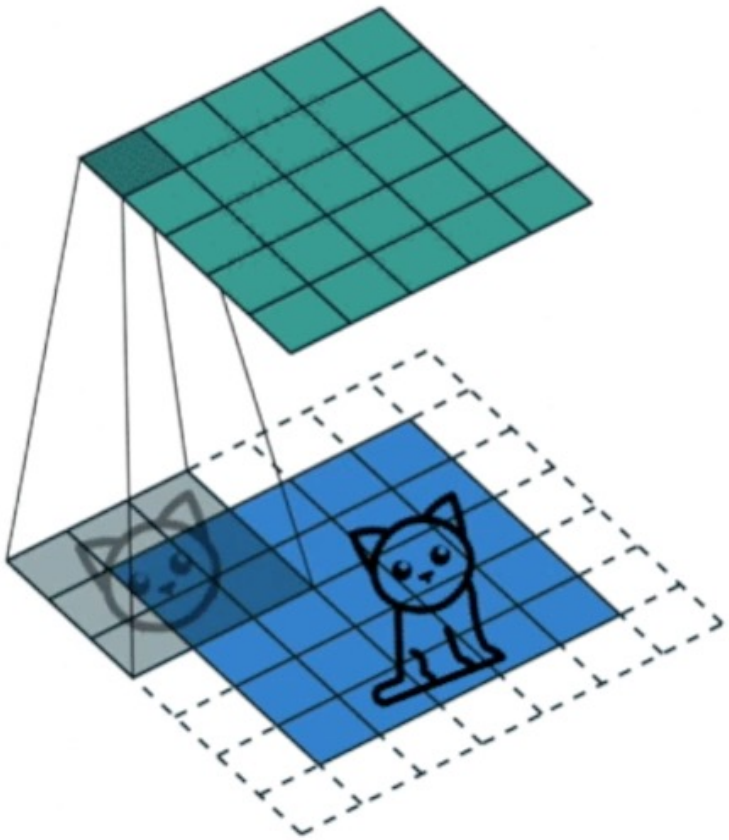
Overview

- Convolution
- Convolutional Neural Networks
- Case Study: LeNet, AlexNet, VGG
- Visualization

Convolution

Convolution is template matching …

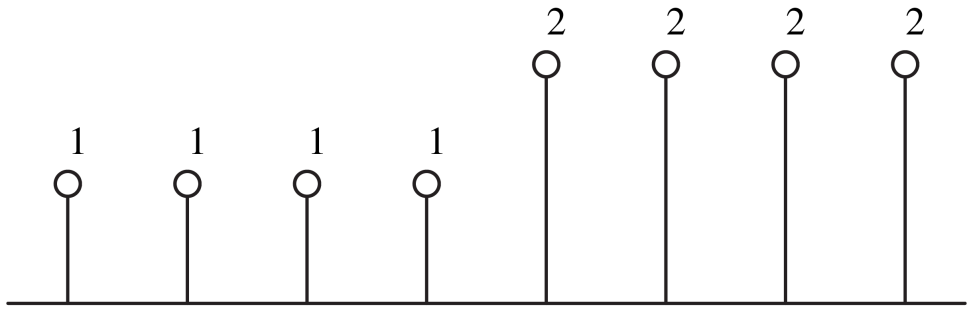
- with a sliding window
- abstract templates
- similarity measured by dot product
- stronger activation, better matching



Animation adapted from: https://lena-voita.github.io/nlp_course/models/convolutional.html

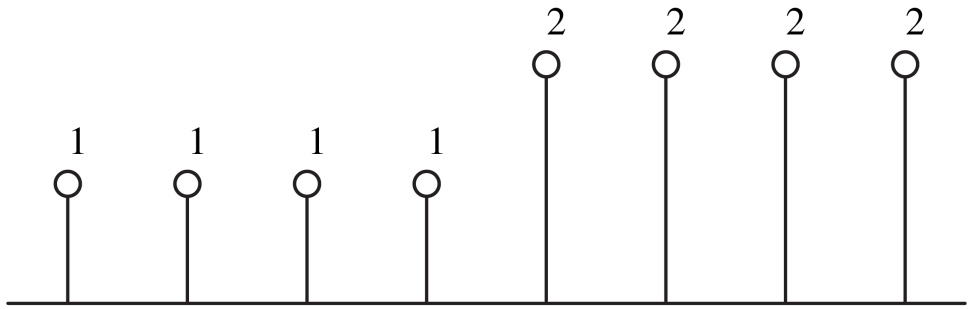
Convolution: a 1-D example

input

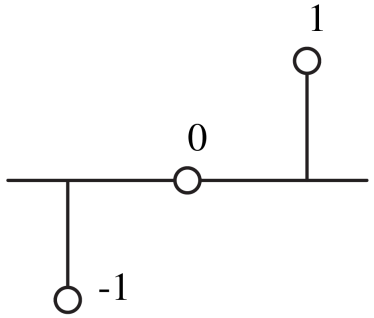


Convolution: a 1-D example

input



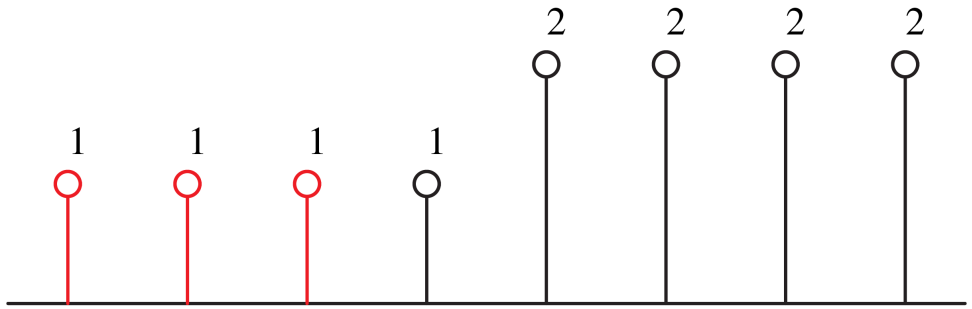
filter



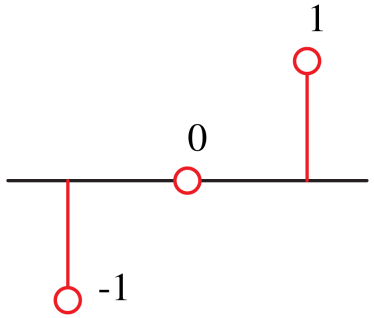
Convolution: a 1-D example

- sliding window
- dot product

input



filter



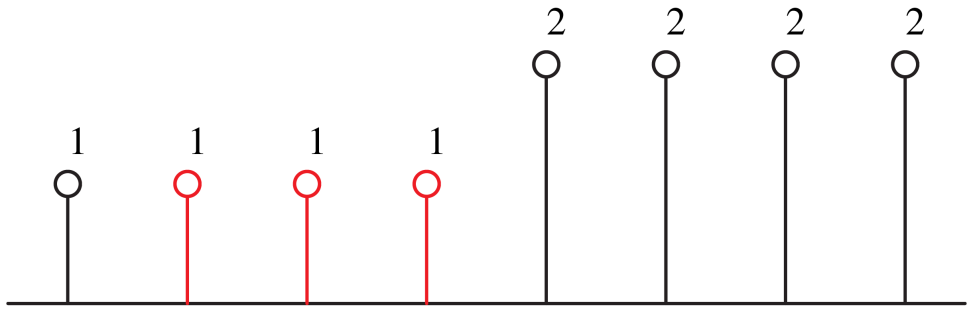
-1x1 + 0x1 + 1x1

output

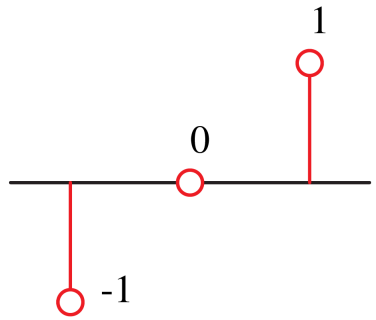
Convolution: a 1-D example

- sliding window
- dot product

input



filter

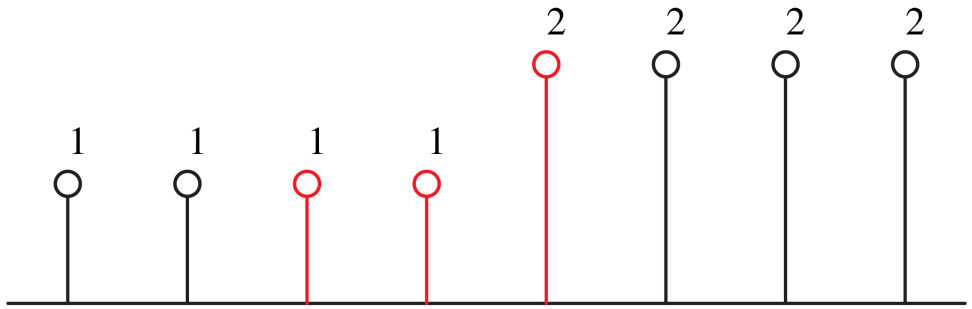


output

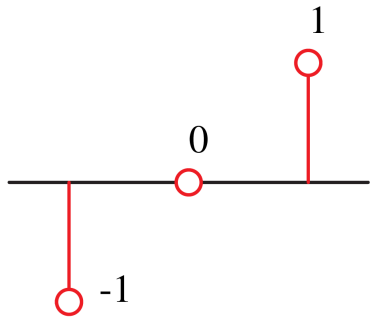
Convolution: a 1-D example

- sliding window
- dot product

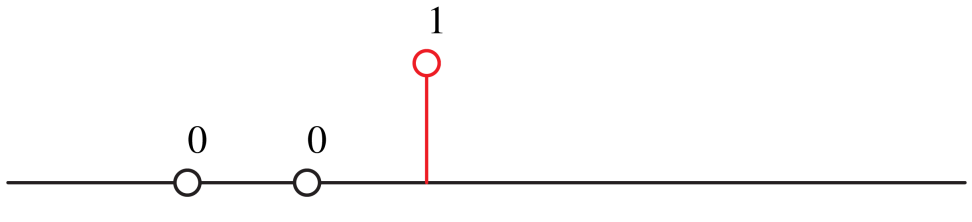
input



filter



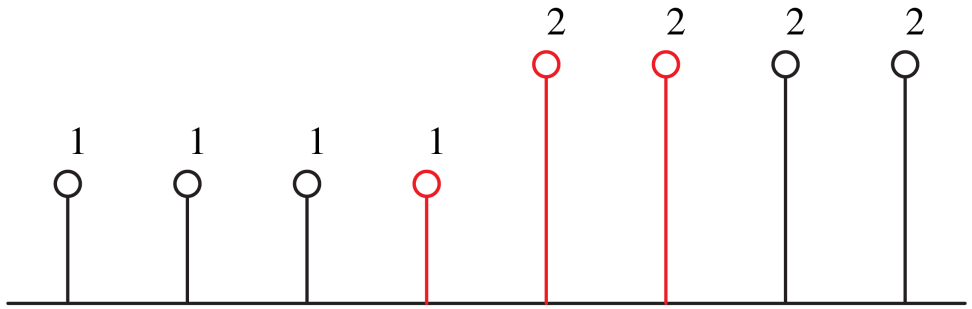
output



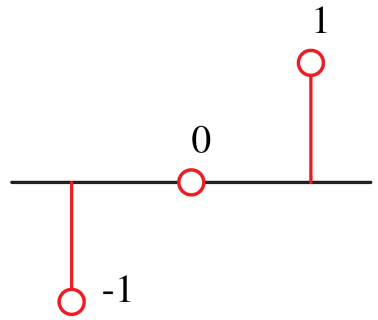
Convolution: a 1-D example

- sliding window
- dot product

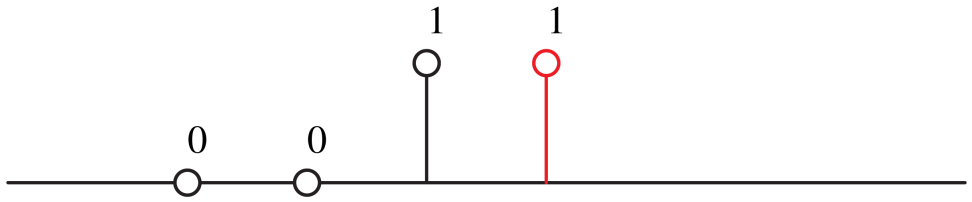
input



filter



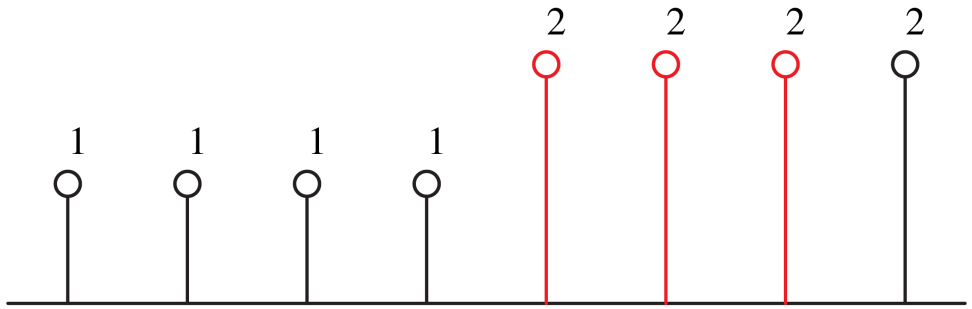
output



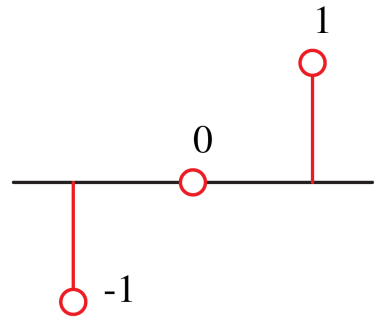
Convolution: a 1-D example

- sliding window
- dot product

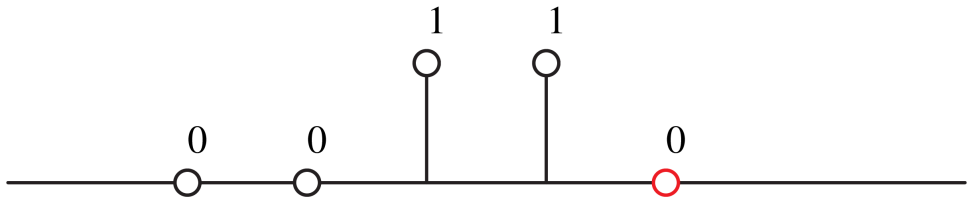
input



filter



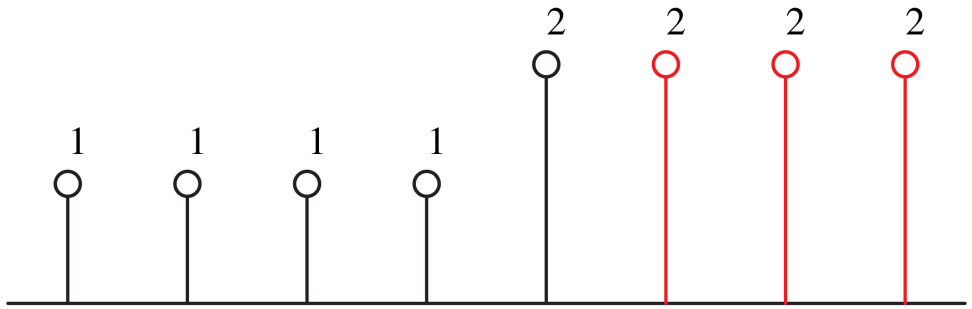
output



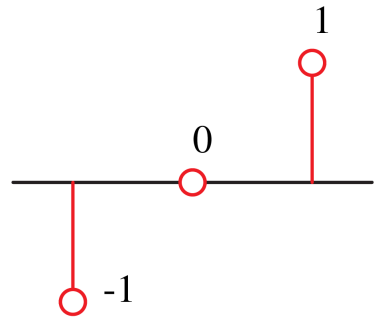
Convolution: a 1-D example

- sliding window
- dot product

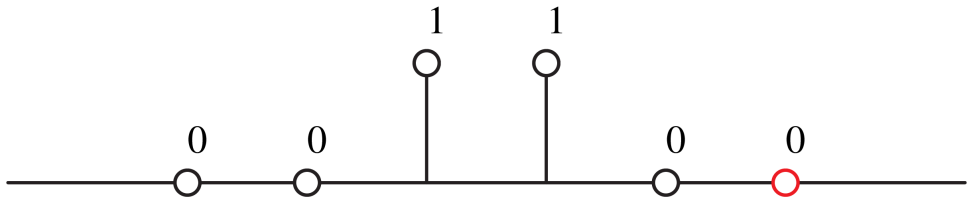
input



filter

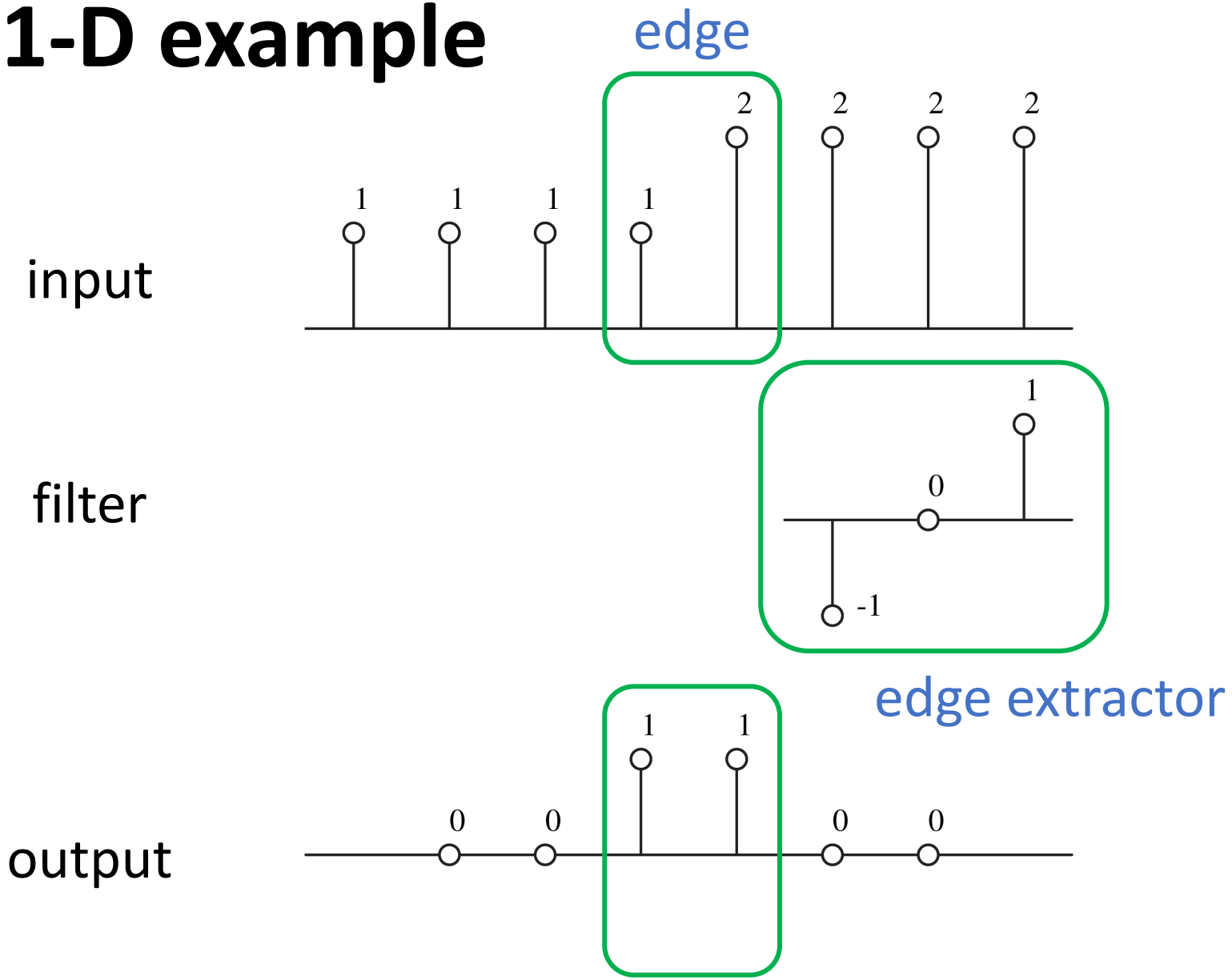


output



Convolution: a 1-D example

- sliding window
- dot product



Convolution: a 2-D example

input

0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

filter

1	**2**	**1**
0	**0**	**0**
-1	**-2**	**-1**

output

Convolution: a 2-D example

input

0 1	0 2	0 1	0	0	0	0	0
0 0	0 0	0 0	0	0	1	1	0
0 -1	1 -2	1 -1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

filter

1	**2**	**1**
0	**0**	**0**
-1	**-2**	**-1**

- sliding window
- dot product

output

-3					

Convolution: a 2-D example

input

0	0 1	0 2	0 1	0	0	0	0
0	0 0	0 0	0 0	0	1	1	0
0	1 -1	1 -2	1 -1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

filter

1	2	1
0	0	0
-1	-2	-1

- sliding window
- dot product

output

-3	-4				

Convolution: a 2-D example

input

0	0	0 1	0 2	0 1	0	0	0
0	0	0 0	0 0	0 0	1	1	0
0	1	1 -1	1 -2	1 -1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

filter

1	2	1
0	0	0
-1	-2	-1

- sliding window
- dot product

output

-3	-4	-4			

Convolution: a 2-D example

input

0	0	0	0_{1}	0_{2}	0_{1}	0	0
0	0	0	0_{0}	0_{0}	1_{0}	1	0
0	1	1	1_{-1}	1_{-2}	1_{-1}	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

filter

1	**2**	**1**
0	**0**	**0**
-1	**-2**	**-1**

- sliding window
- dot product

output

-3	-4	-4	-4		

Convolution: a 2-D example

input

0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	1	1	1	0 1	0 2	0 1
0	0	1	1	1	0 0	0 0	0 0
0	0	0	0	0	0 -1	0 -2	0 -1

filter

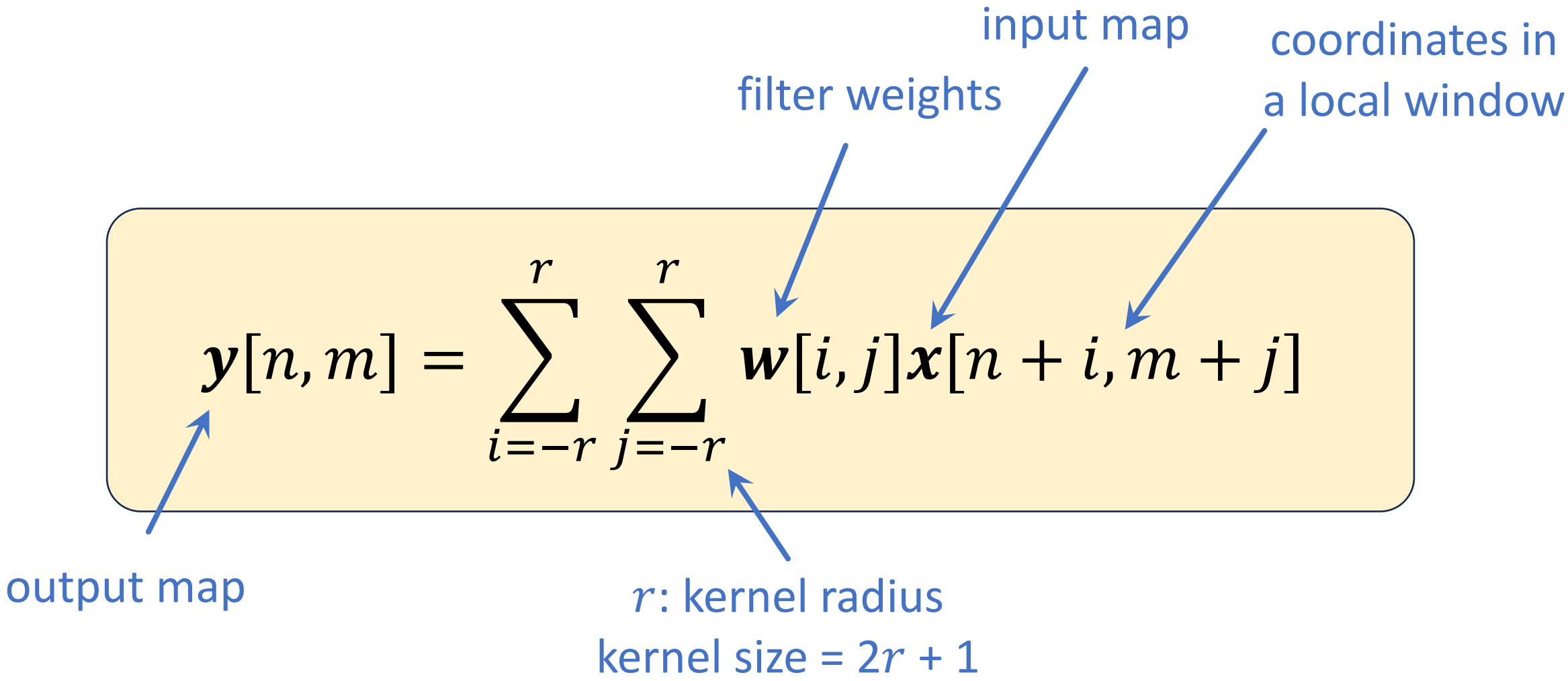
1	2	1
0	0	0
-1	-2	-1

- sliding window
- dot product

output

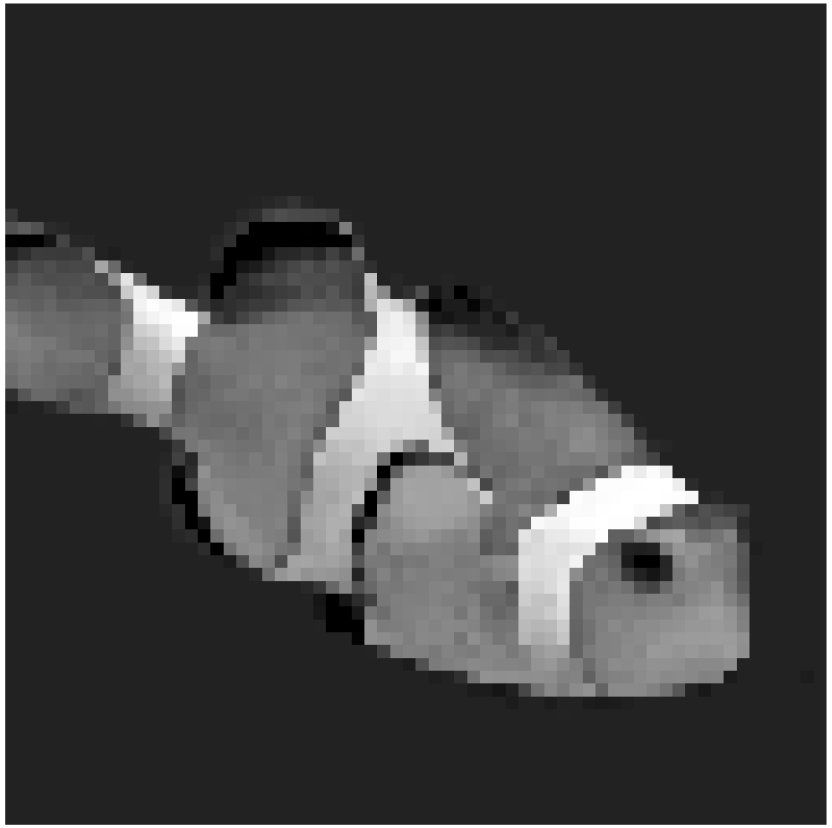
-3	-4	-4	-4	-4	-3
-3	-4	-4	-3	-1	0
0	0	0	0	0	0
2	1	0	1	3	3
2	1	0	1	3	3
1	3	4	3	1	0

Convolution: a 2-D example



* In ConvNets, convolution is often implemented as **cross-correlation** (no flipping)

Convolution: 2-D



*

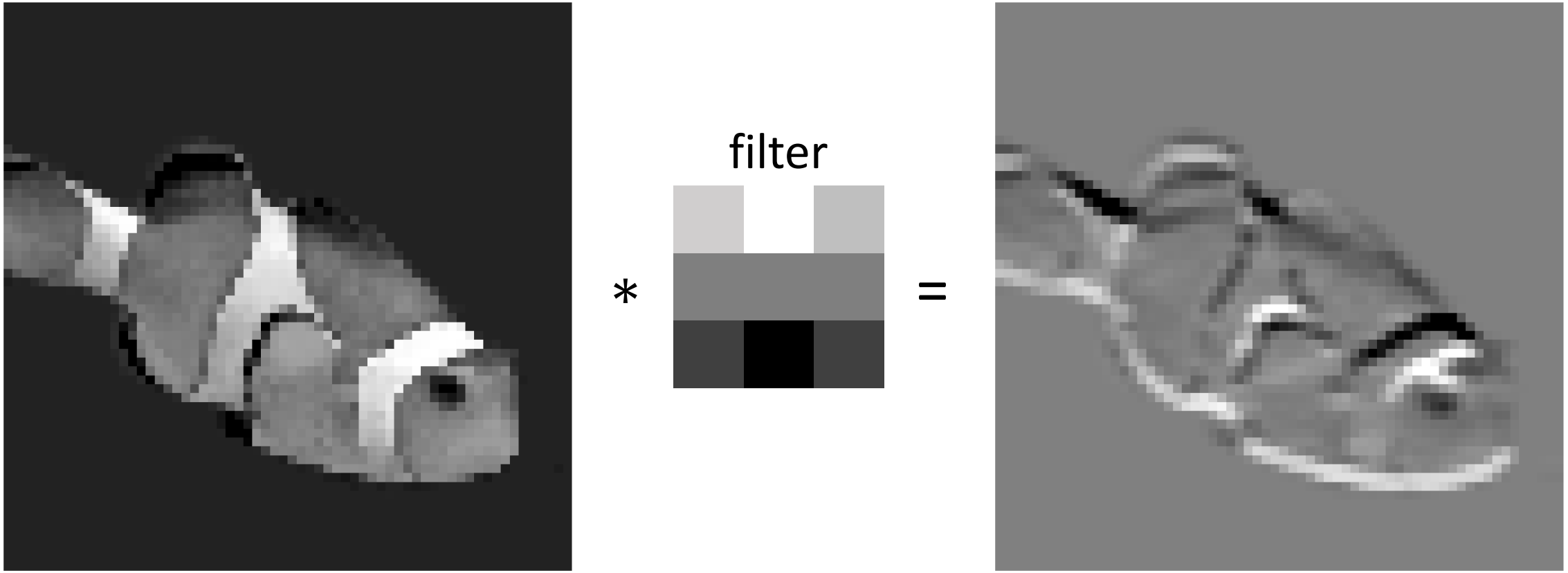
filter

1	2	1
0	0	0
-1	-2	-1

=

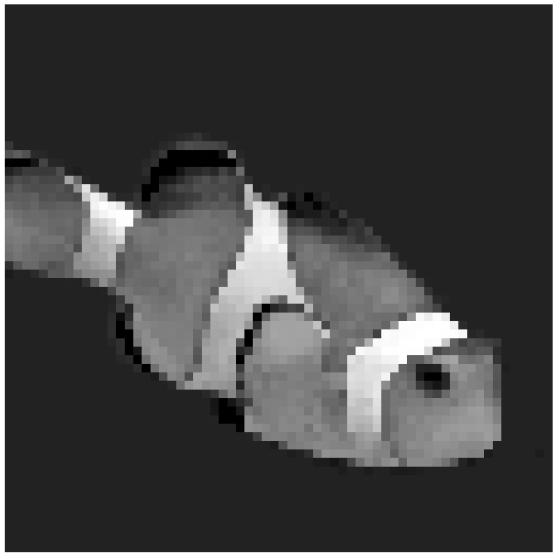
This example is a Sobel edge extractor of horizontal edges.

Convolution: 2-D



This example is a Sobel edge extractor of horizontal edges.

Convolution: Multi-channel outputs



$*$

1	2	1
0	0	0
-1	-2	-1

$=$

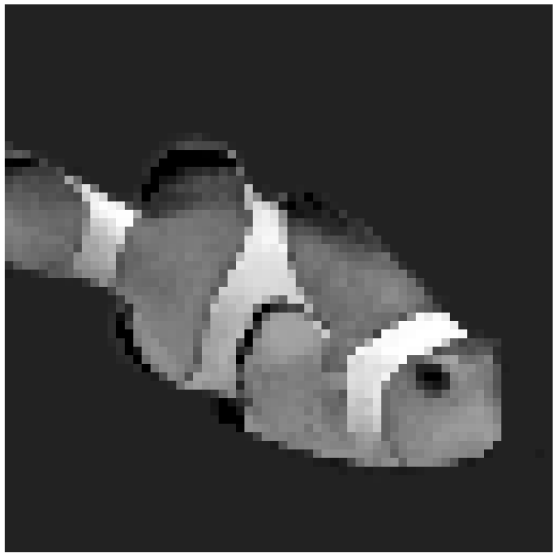
one filter, one feature

$*$

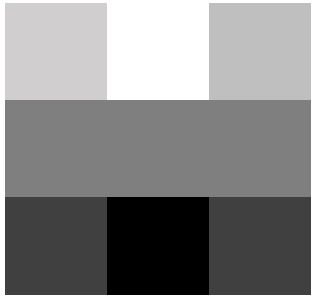
1	0	-1
2	0	-2
1	0	-1

$=$

Convolution: Multi-channel outputs



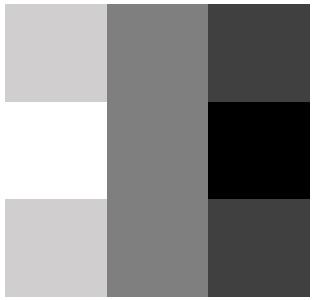
*



=

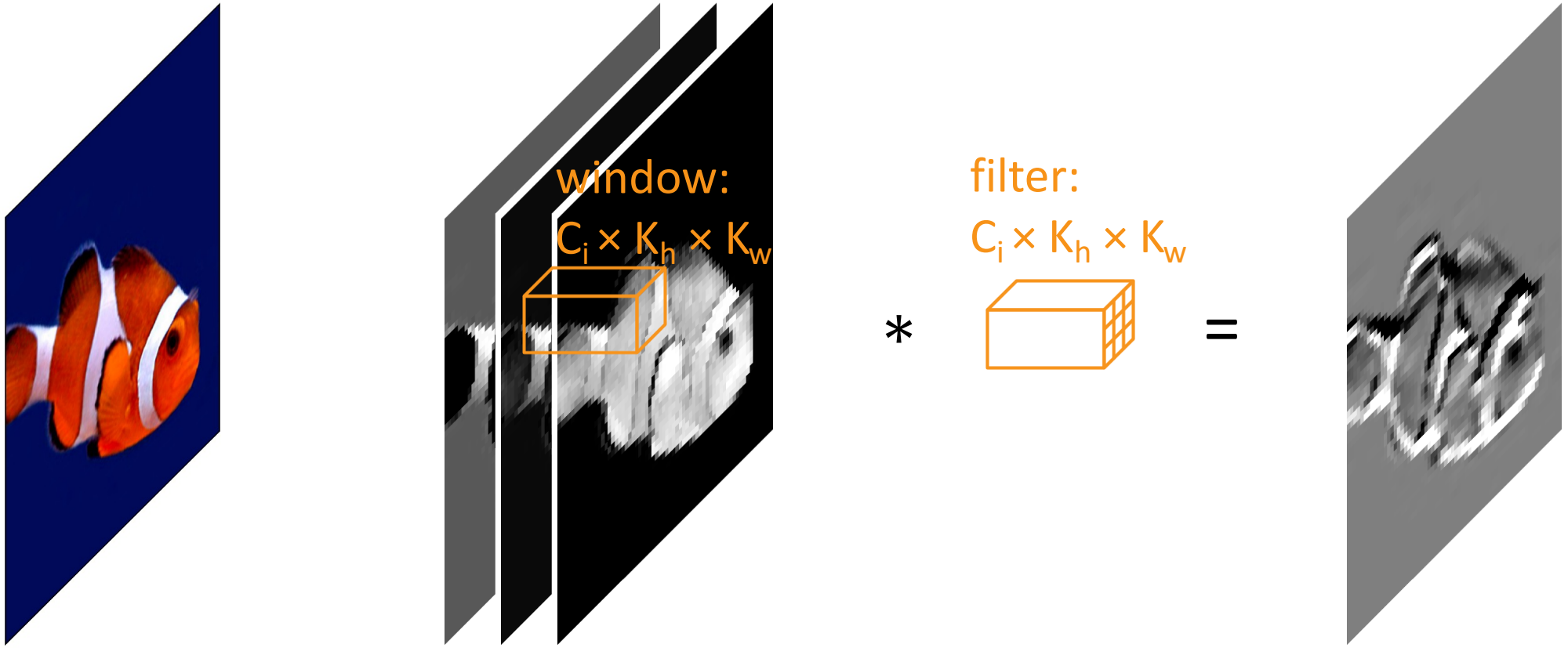
one filter, one feature

*

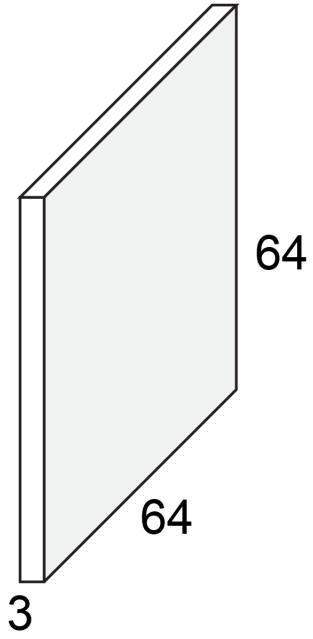


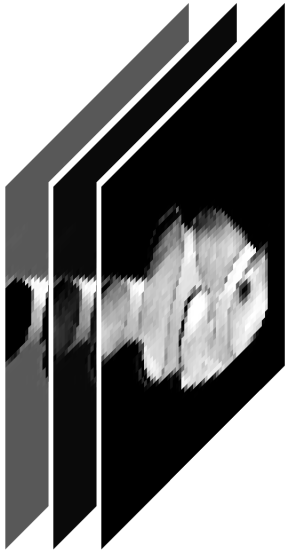
=

Convolution: Multi-channel inputs



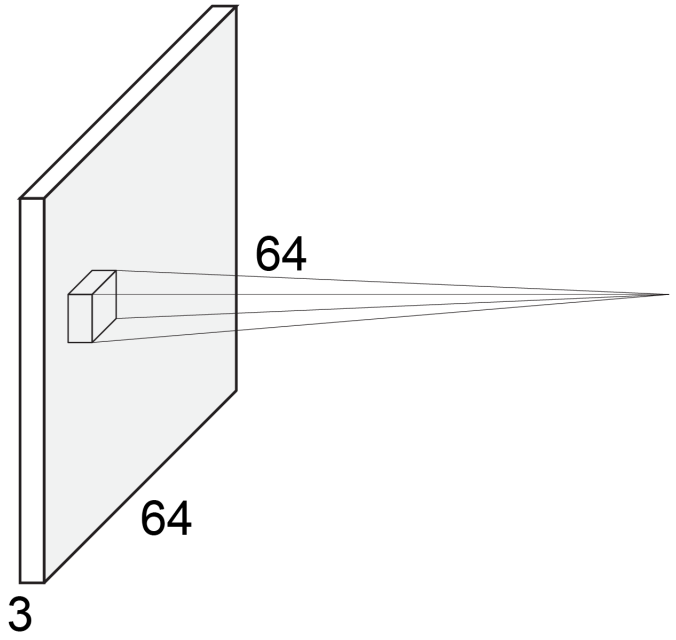
Convolution: tensor views

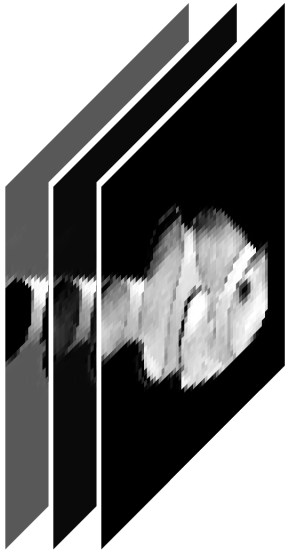




- Tensor: high-dimension array
- feature maps
  - 3-D tensor: C × H × W
  - C: channels
  - H: height
  - W: width

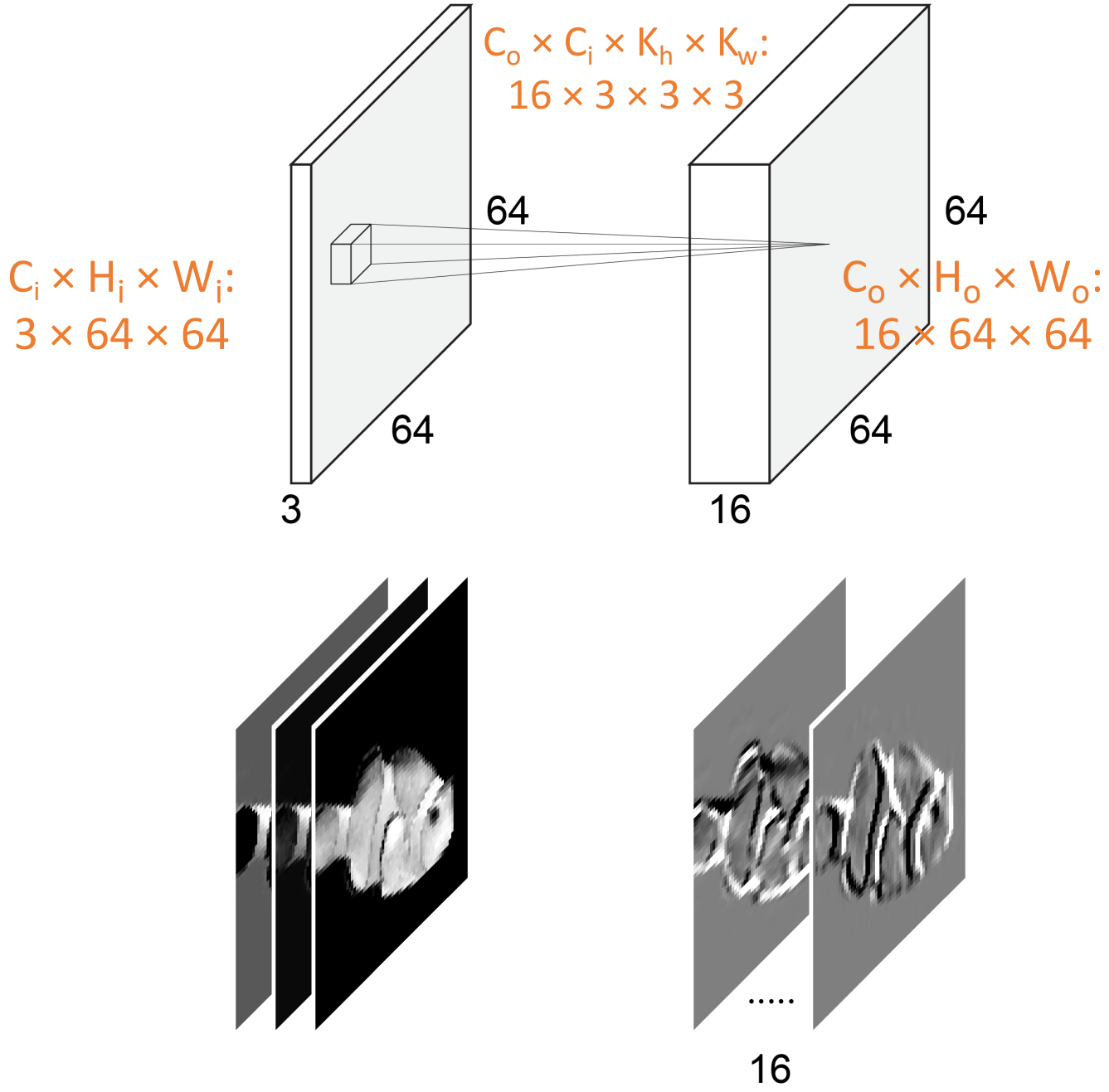
Convolution: tensor views





- Tensor: high-dimension array
- feature maps
  - 3-D tensor: $C \times H \times W$
  - C: channels
  - H: height
  - W: width
- filters
  - 4-D tensor: $C_o \times C_i \times K_h \times K_w$
  - C_o: output channels
  - C_i: input channels
  - K_h, K_w: filter height, width

Convolution: tensor views



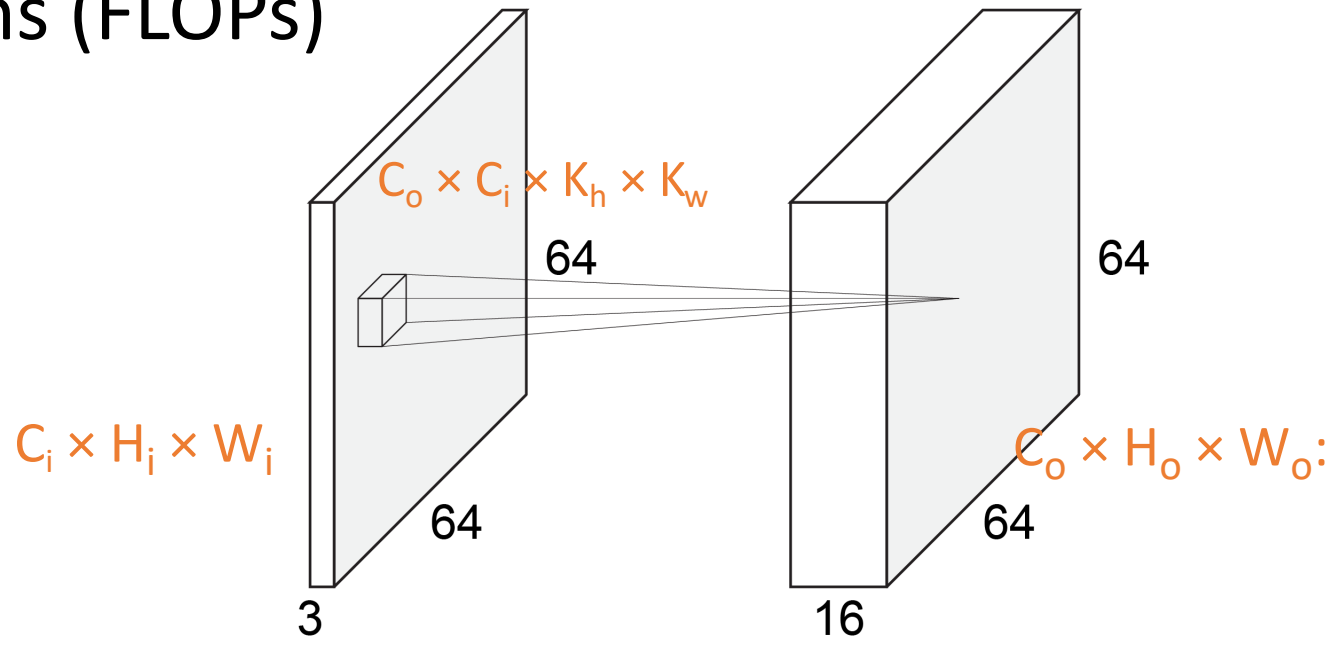
- Tensor: high-dimension array
- feature maps
  - 3-D tensor: $C \times H \times W$
  - C: channels
  - H: height
  - W: width
- filters
  - 4-D tensor: $C_o \times C_i \times K_h \times K_w$
  - C_o: output channels
  - C_i: input channels
  - K_h, K_w: filter height, width

Convolution: # parameters and # operations

- # parameters
  - weights: $C_o \times C_i \times K_h \times K_w$
  - bias: C_o

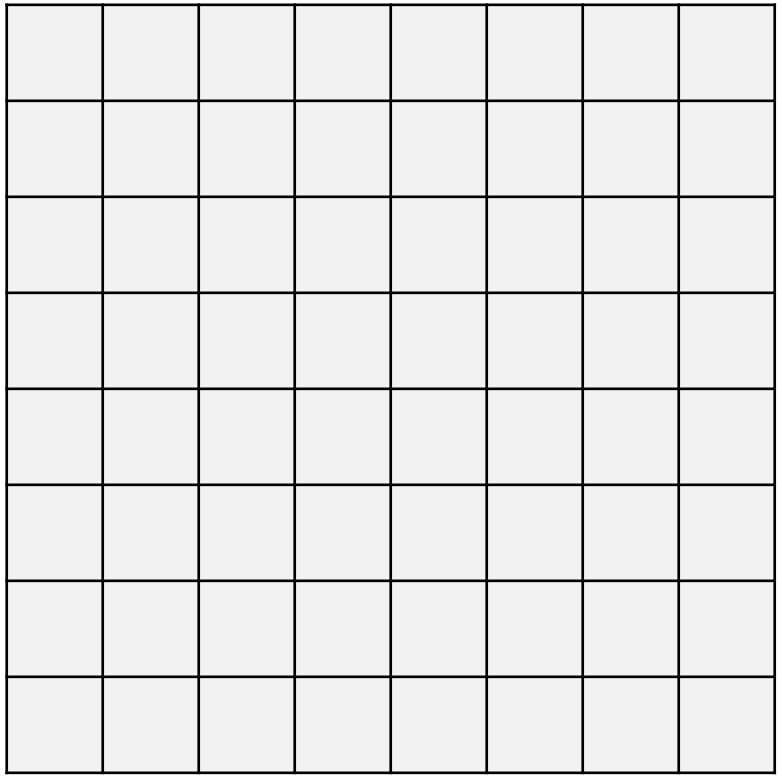
- # floating-point operations (FLOPs)
  - # params $\times H_o \times W_o$

- # params per FLOP
  - 1 / spatial_size
  - parameter-efficient

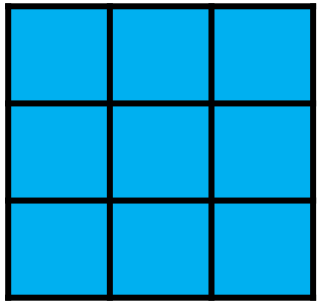


Convolution: padding

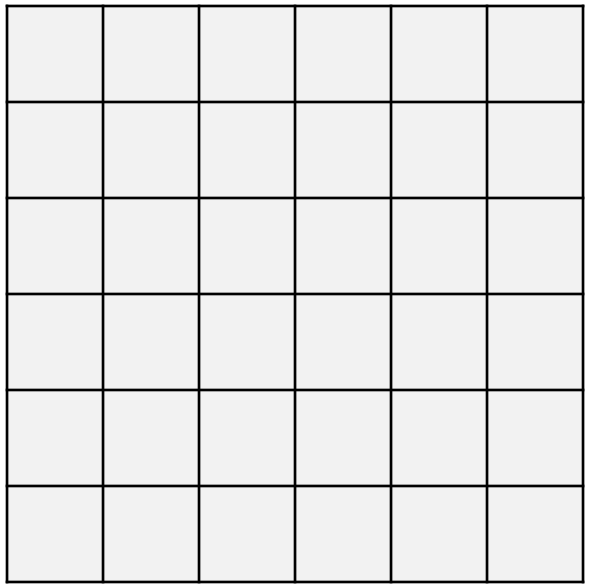
input: H × W = 8 × 8



filter

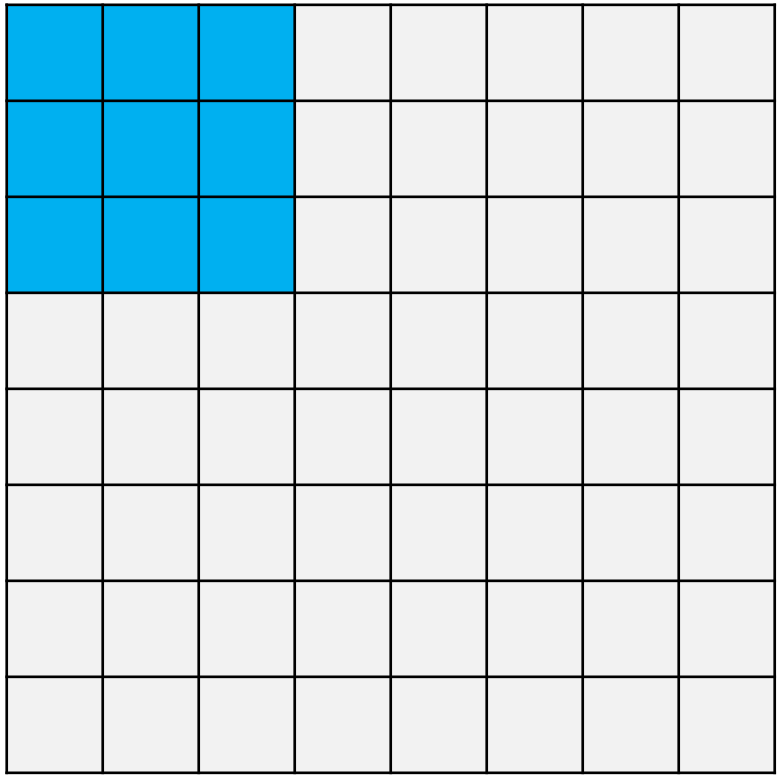


output: H × W = 6 × 6

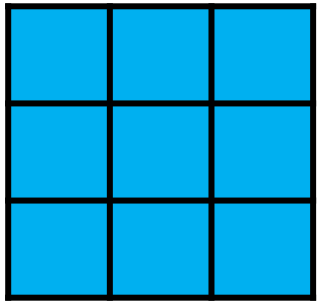


Convolution: padding

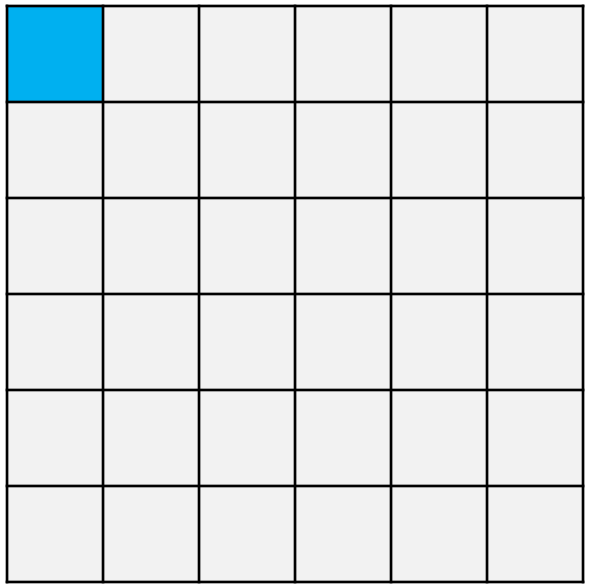
input: H × W = 8 × 8



filter

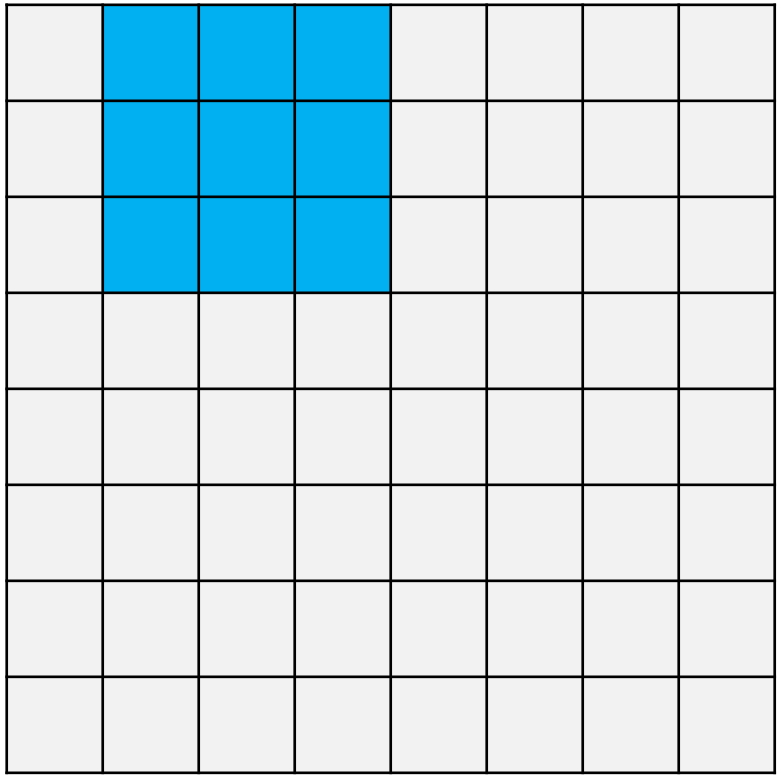


output: H × W = 6 × 6

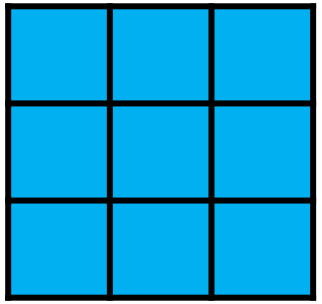


Convolution: padding

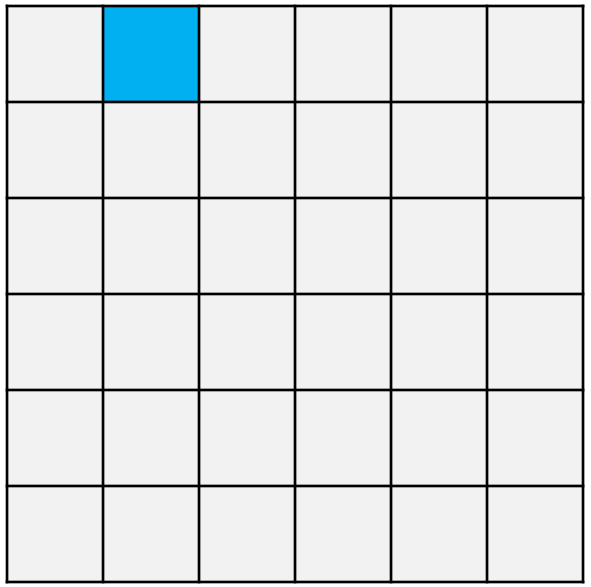
input: H × W = 8 × 8



filter

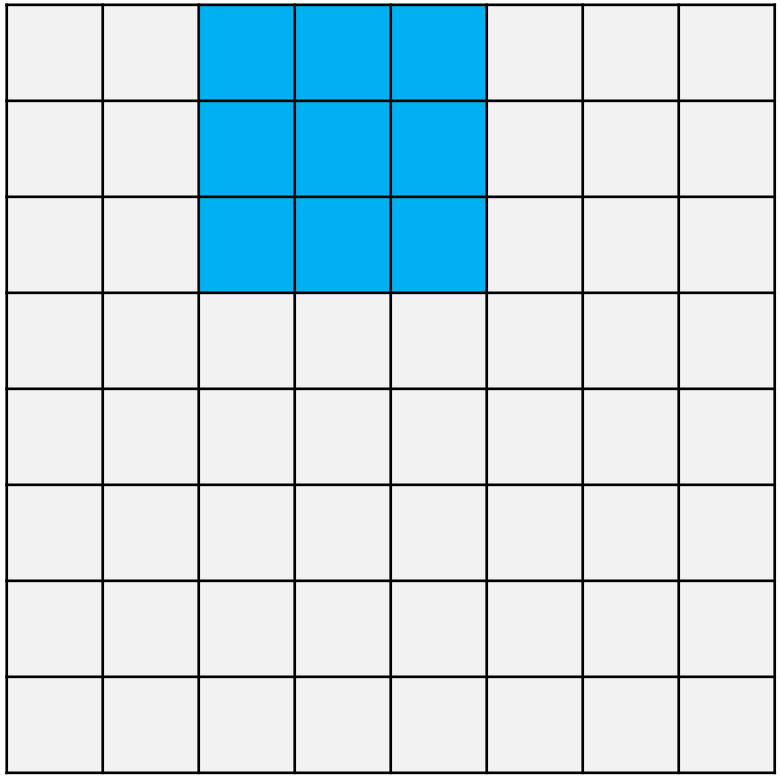


output: H × W = 6 × 6

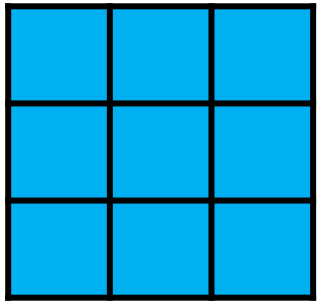


Convolution: padding

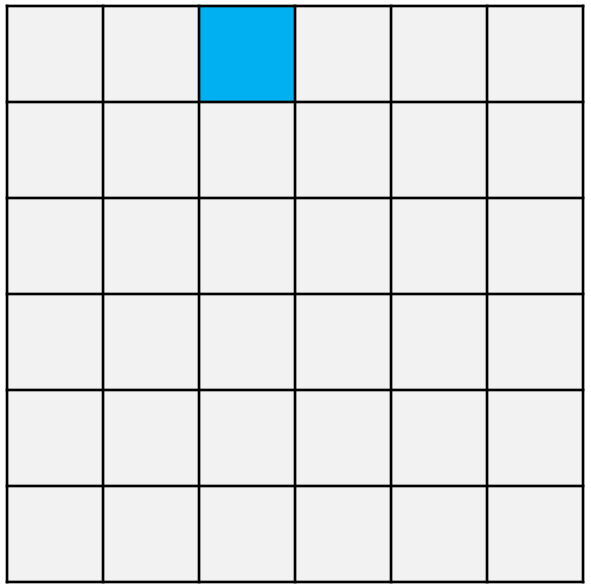
input: H × W = 8 × 8



filter

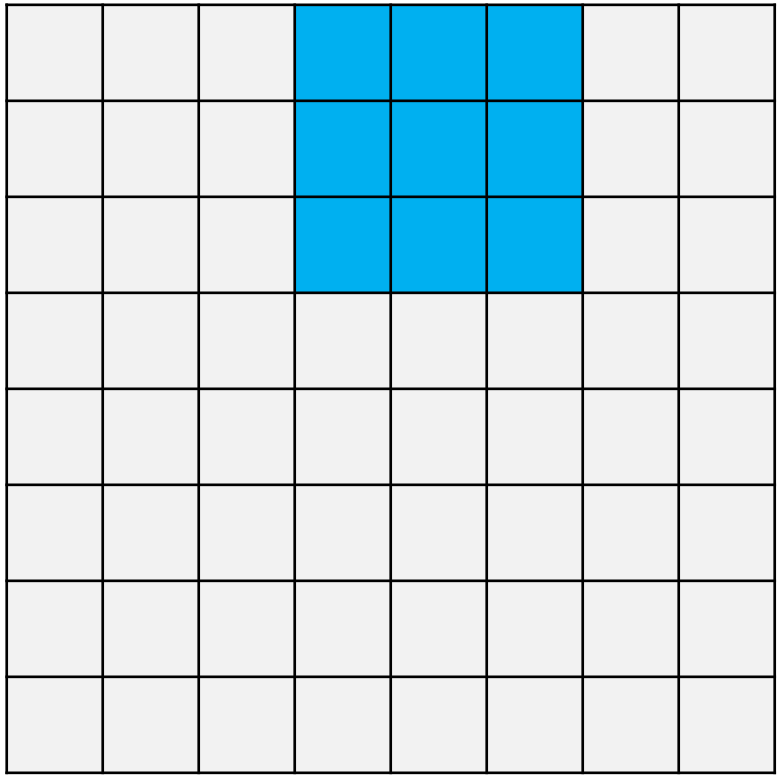


output: H × W = 6 × 6

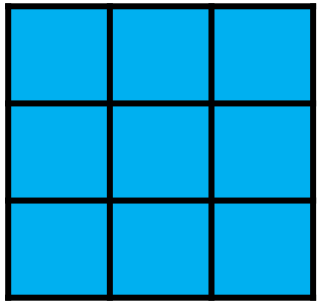


Convolution: padding

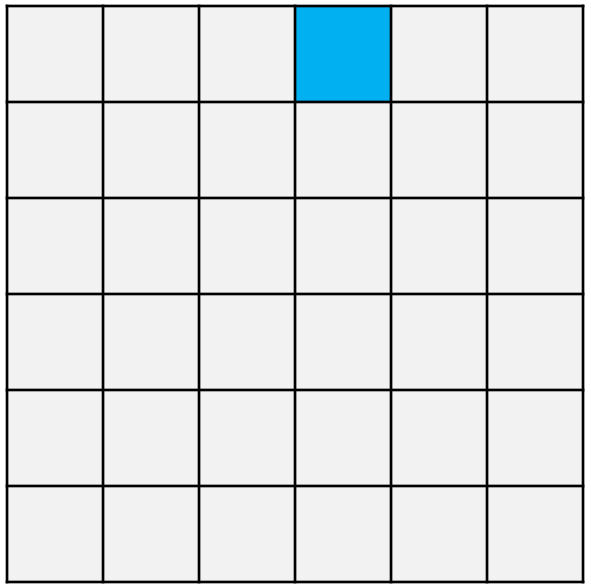
input: H × W = 8 × 8



filter

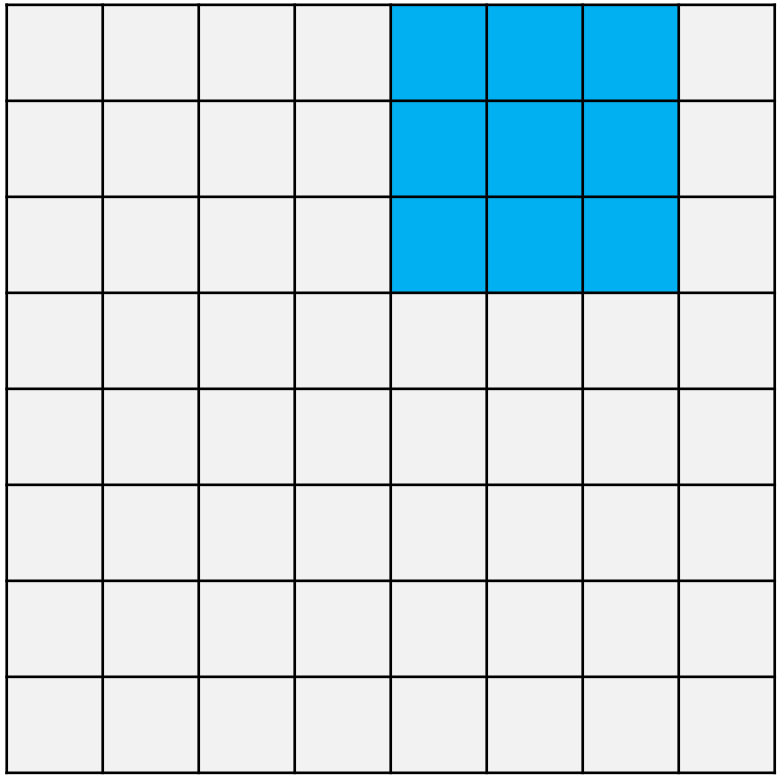


output: H × W = 6 × 6

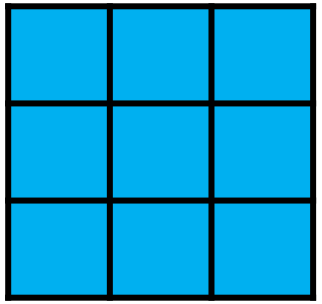


Convolution: padding

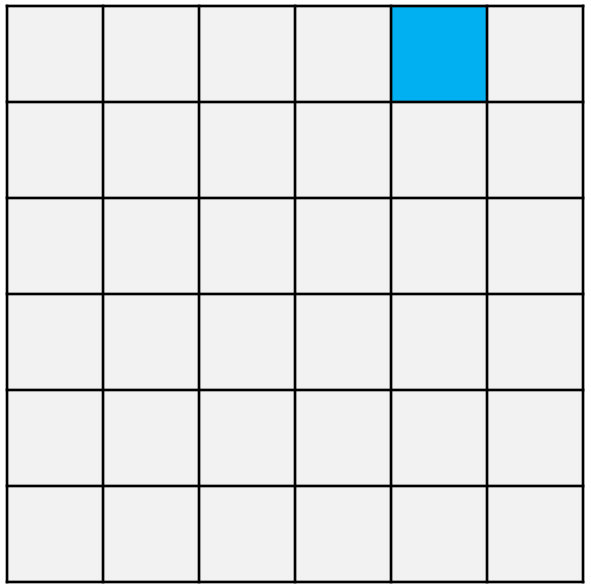
input: H × W = 8 × 8



filter

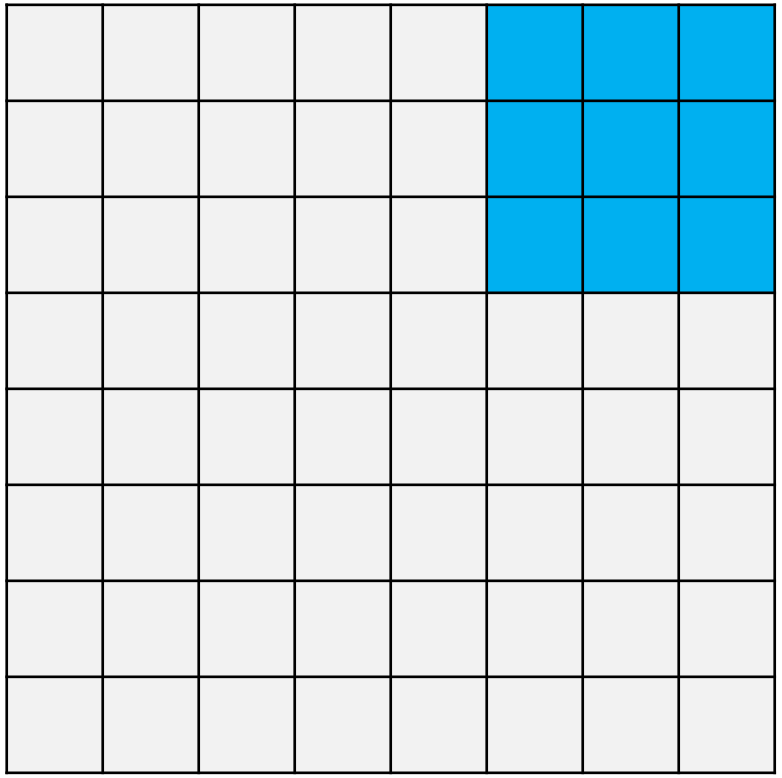


output: H × W = 6 × 6

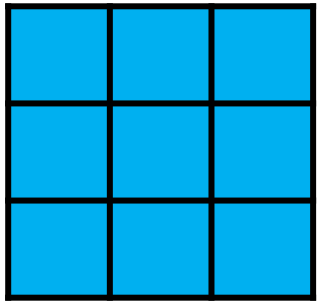


Convolution: padding

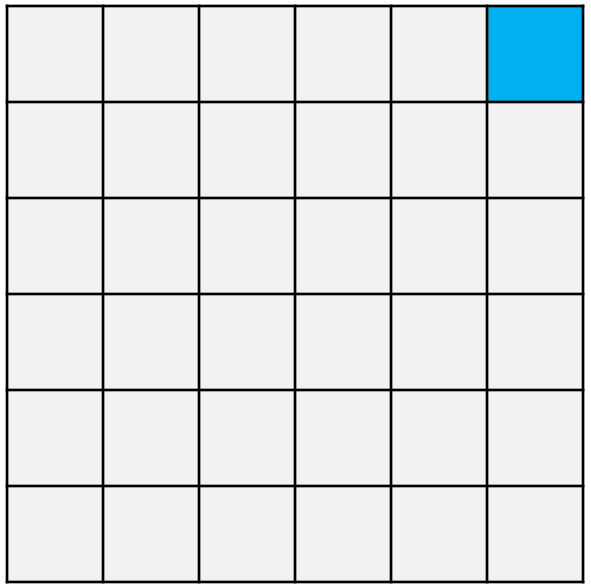
input: H × W = 8 × 8



filter



output: H × W = 6 × 6



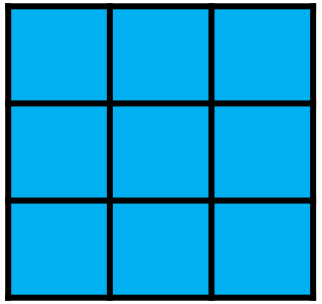
$$H_{out} = H_{in} - K_h + 1$$

Convolution: padding

input: 8 × 8, + pad

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

filter



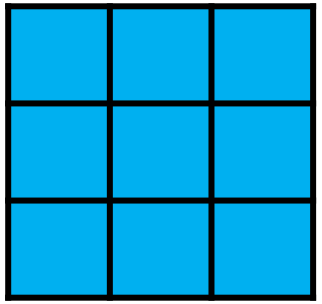
output: H × W = 8 × 8

Convolution: padding

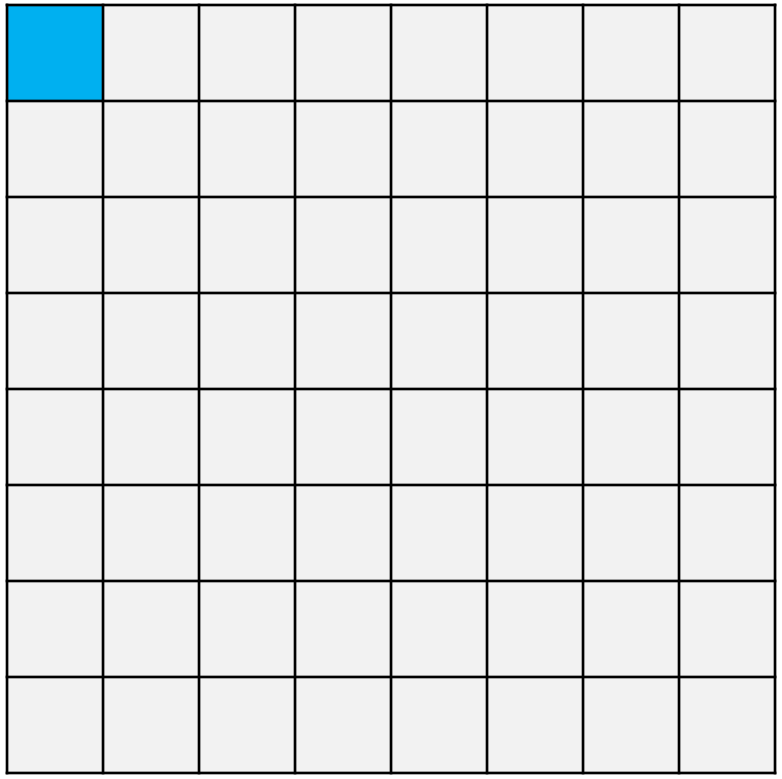
input: 8 × 8, + pad

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

filter



output: H × W = 8 × 8

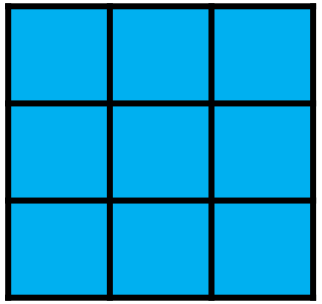


Convolution: padding

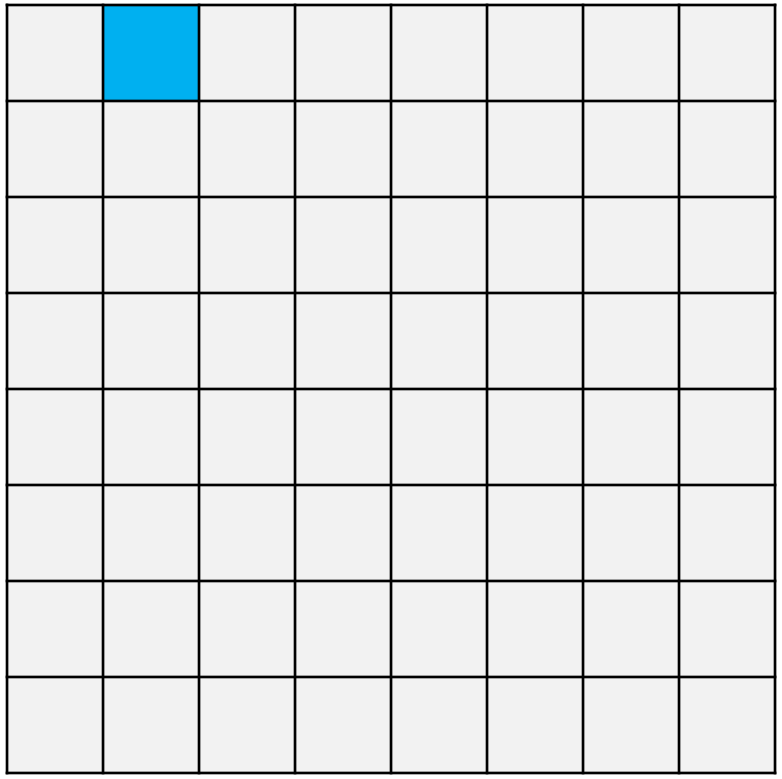
input: 8 × 8, + pad

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

filter



output: H × W = 8 × 8

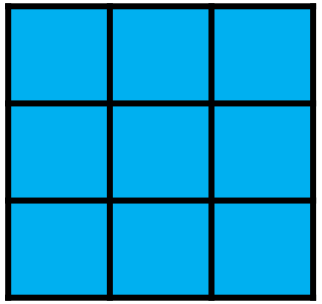


Convolution: padding

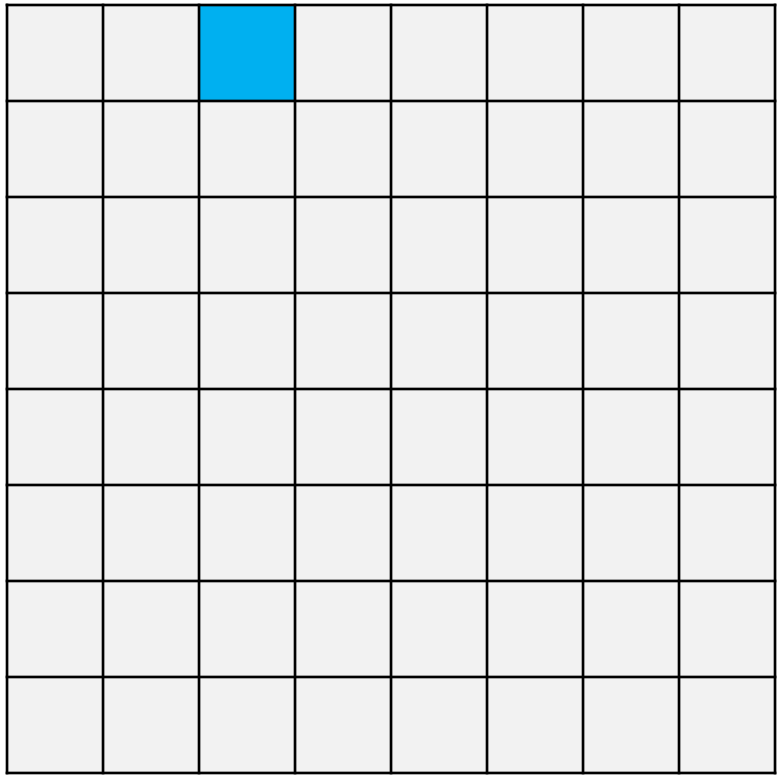
input: 8 × 8, + pad

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

filter

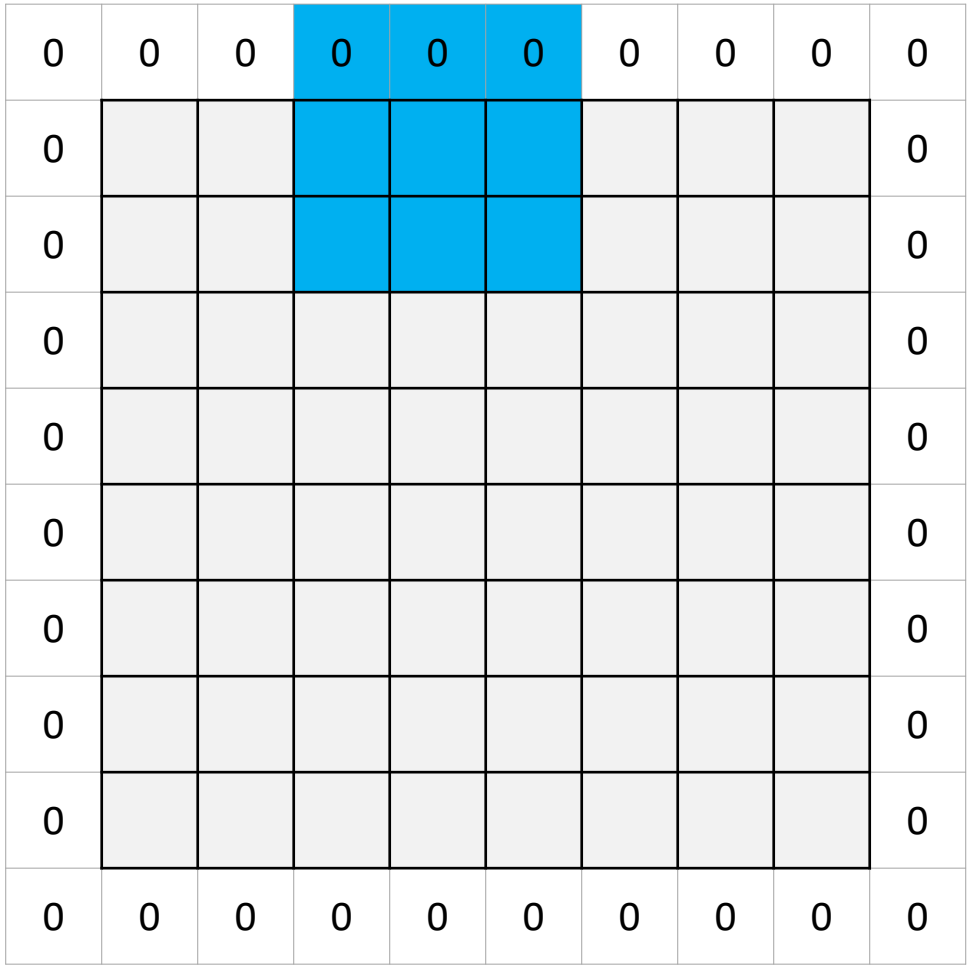


output: H × W = 8 × 8

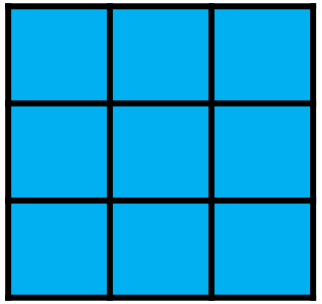


Convolution: padding

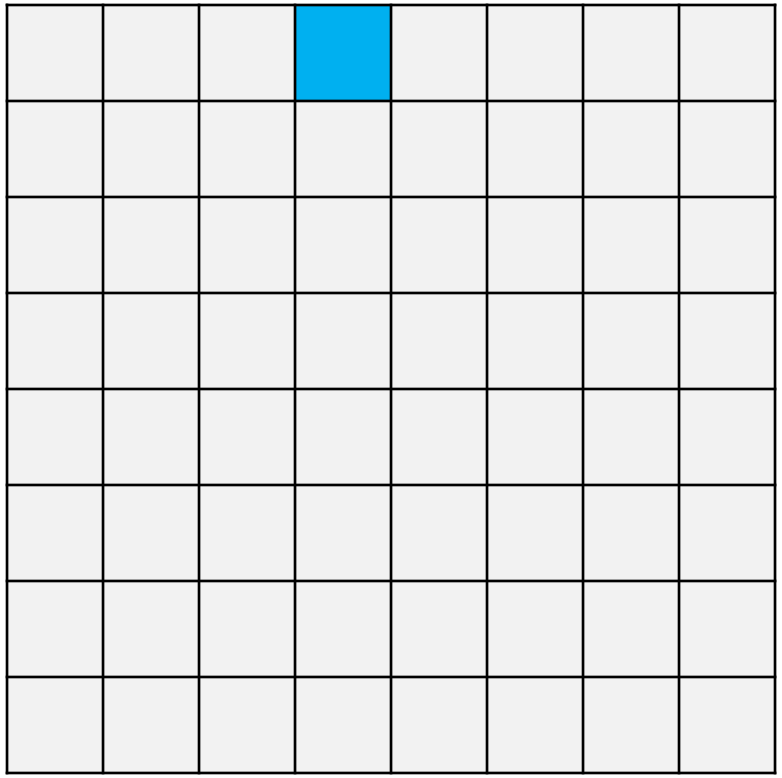
input: 8 × 8, + pad



filter

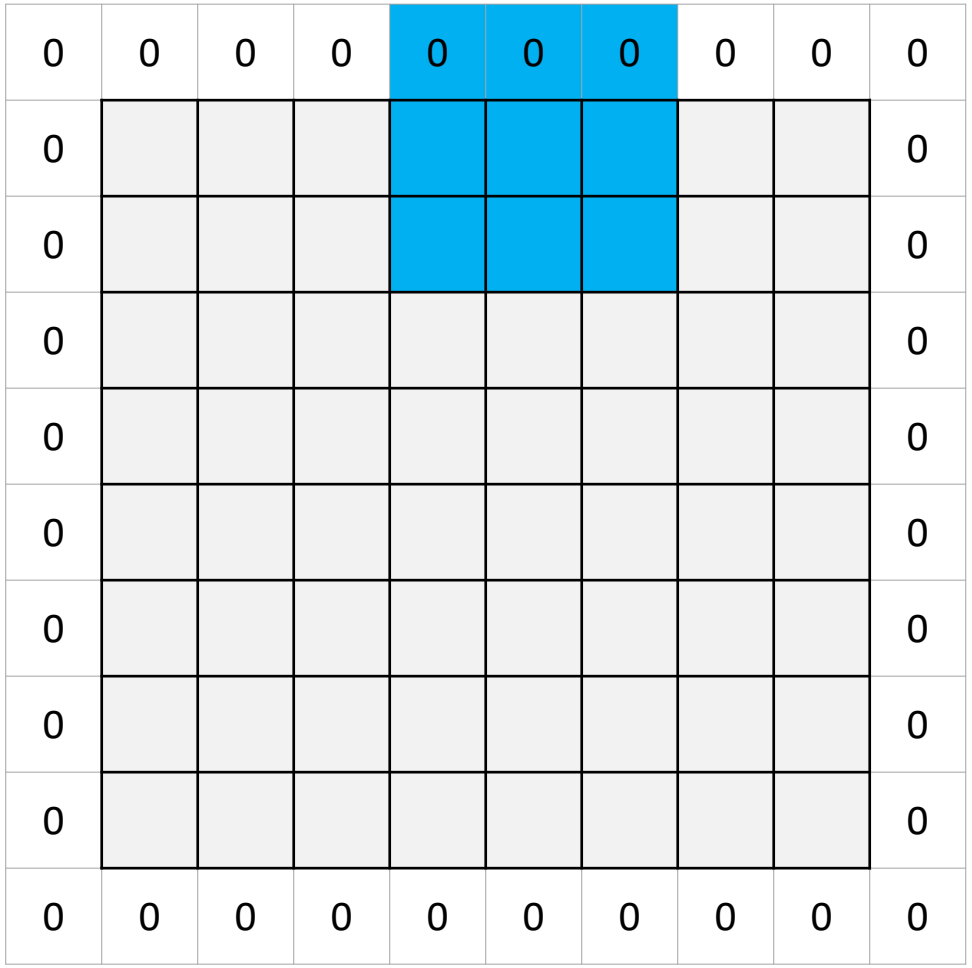


output: H × W = 8 × 8

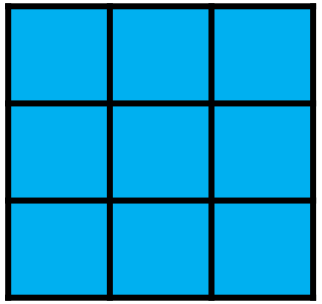


Convolution: padding

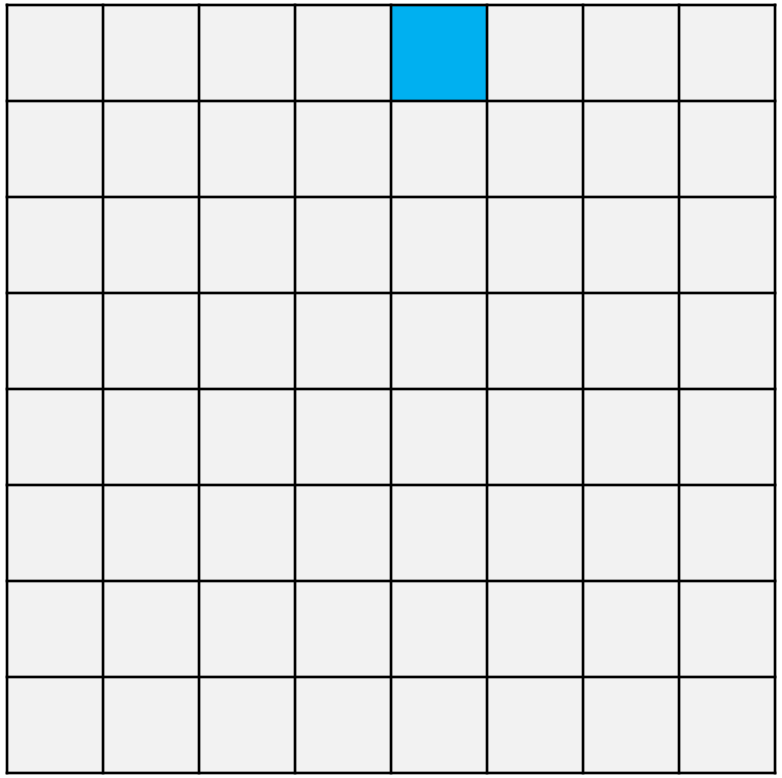
input: 8 × 8, + pad



filter

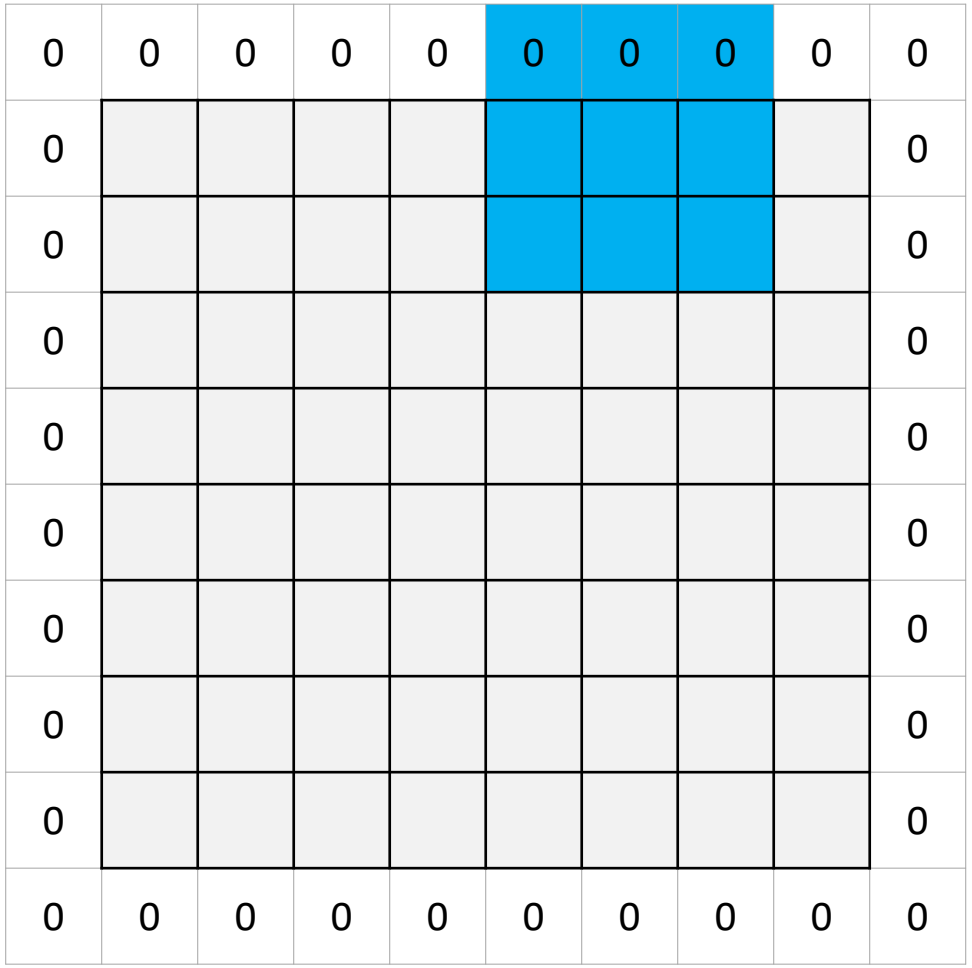


output: H × W = 8 × 8

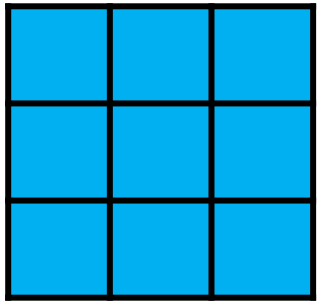


Convolution: padding

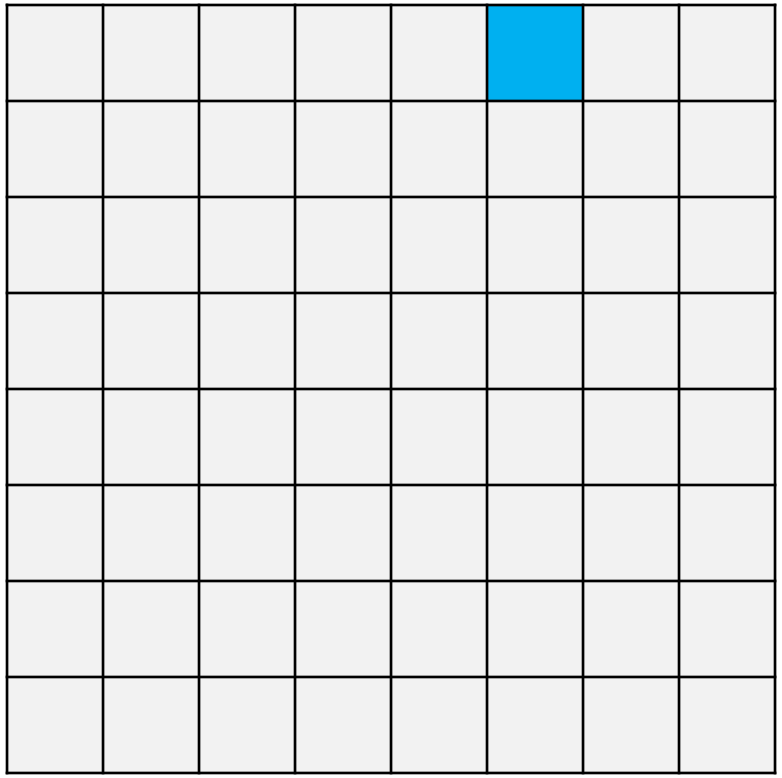
input: 8 × 8, + pad



filter

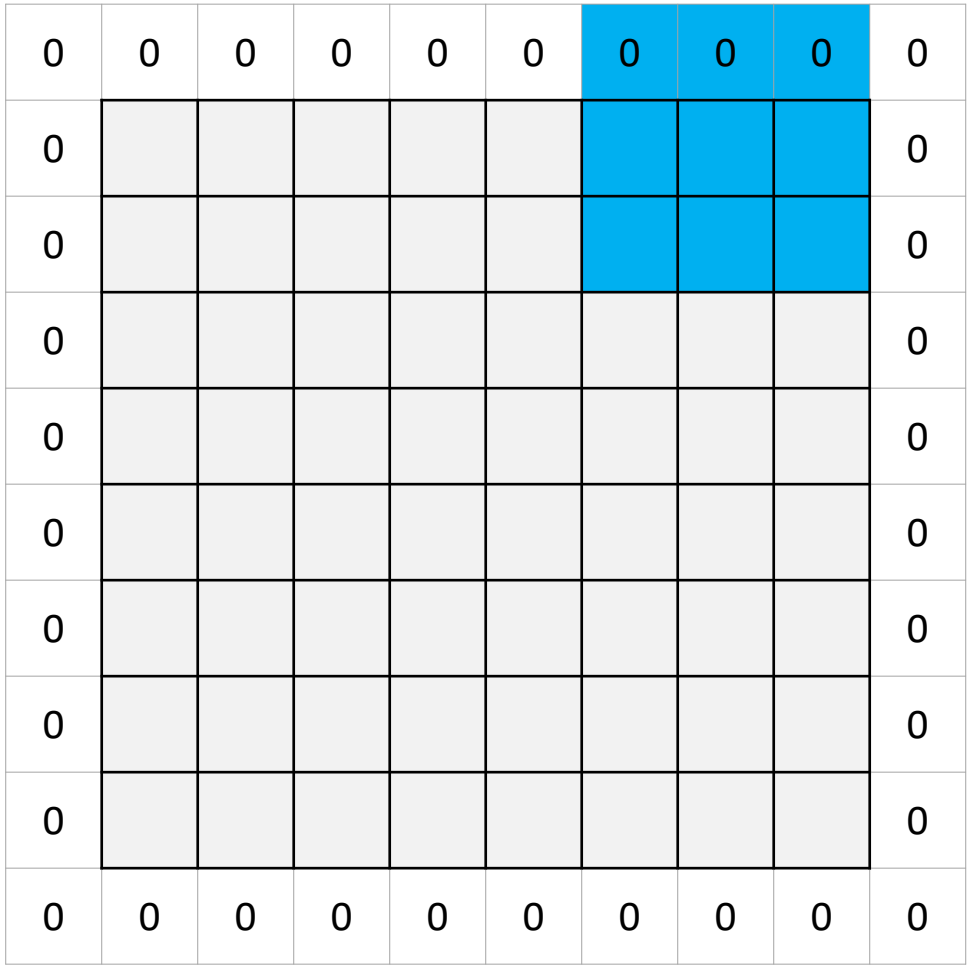


output: H × W = 8 × 8

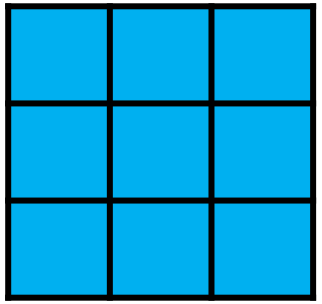


Convolution: padding

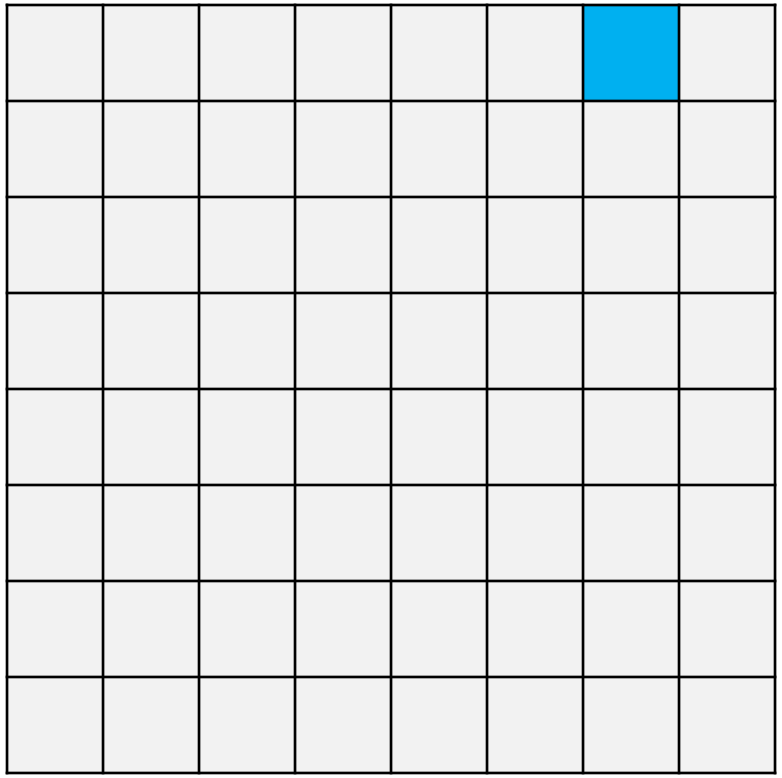
input: 8 × 8, + pad



filter



output: H × W = 8 × 8

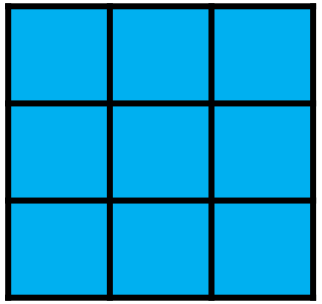


Convolution: padding

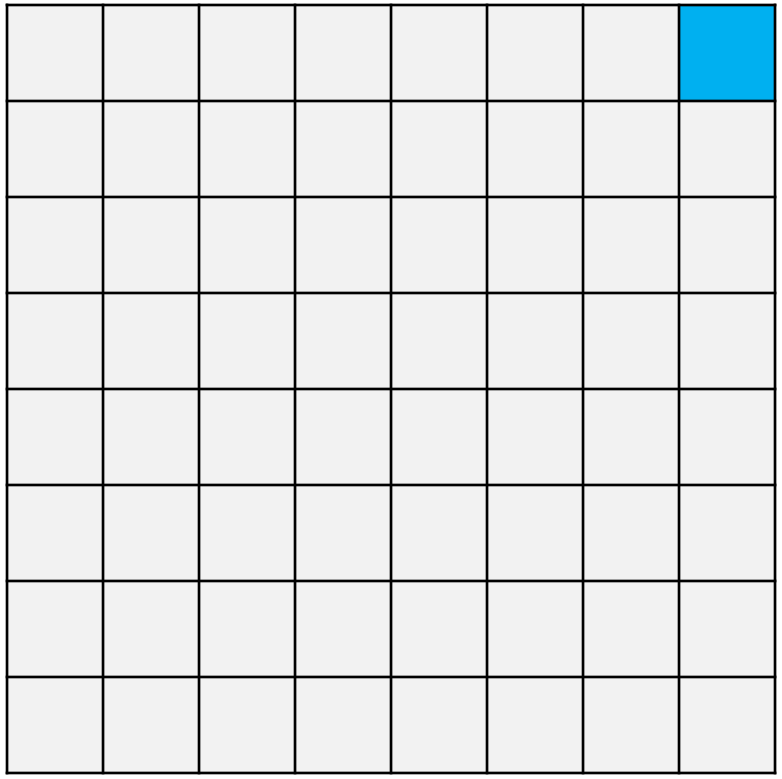
input: 8 × 8, + pad

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

filter



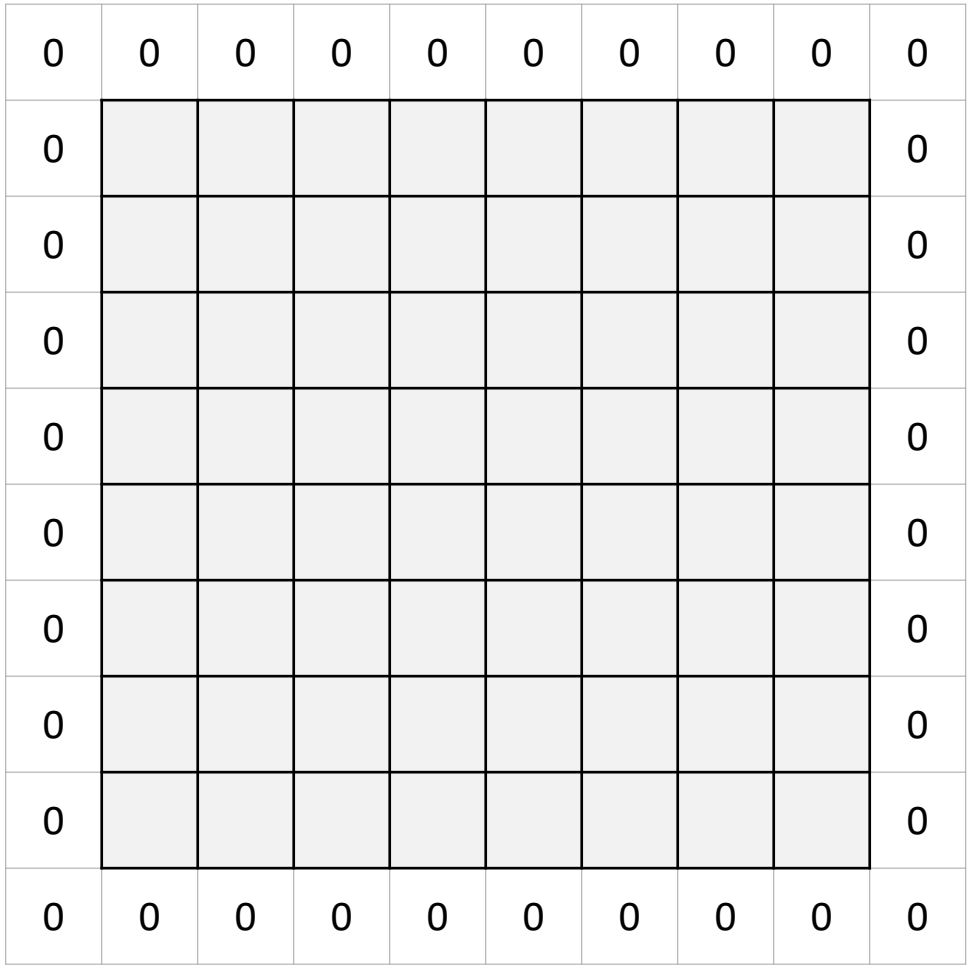
output: H × W = 8 × 8



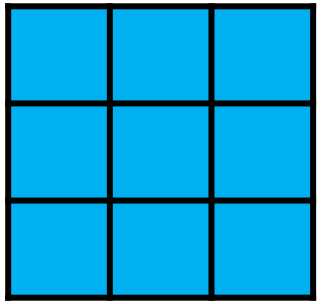
Convolution: padding

- pad = ⌊kernel_size / 2⌋
- maintains feature map size

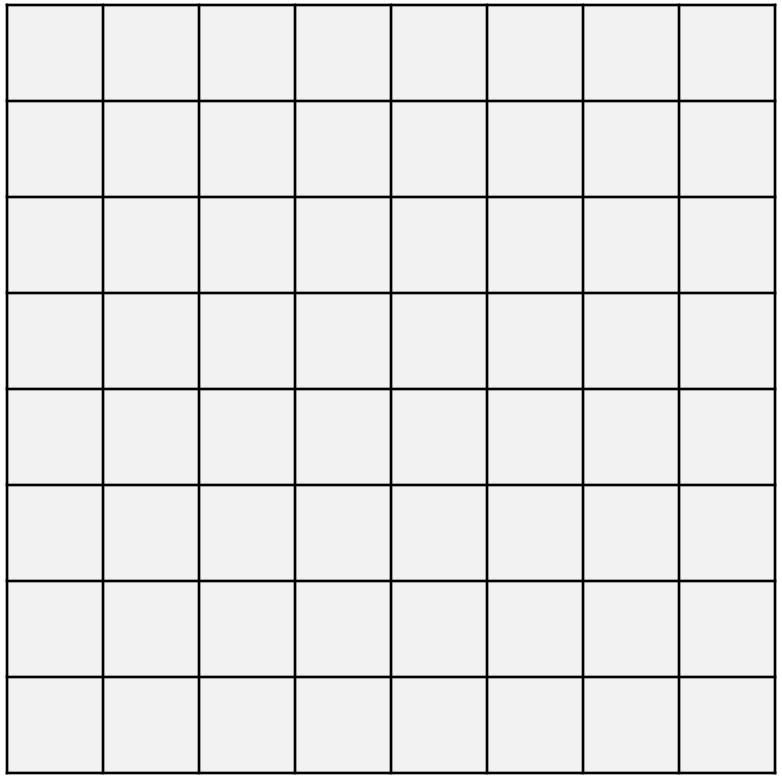
input: 8 × 8, + pad



filter



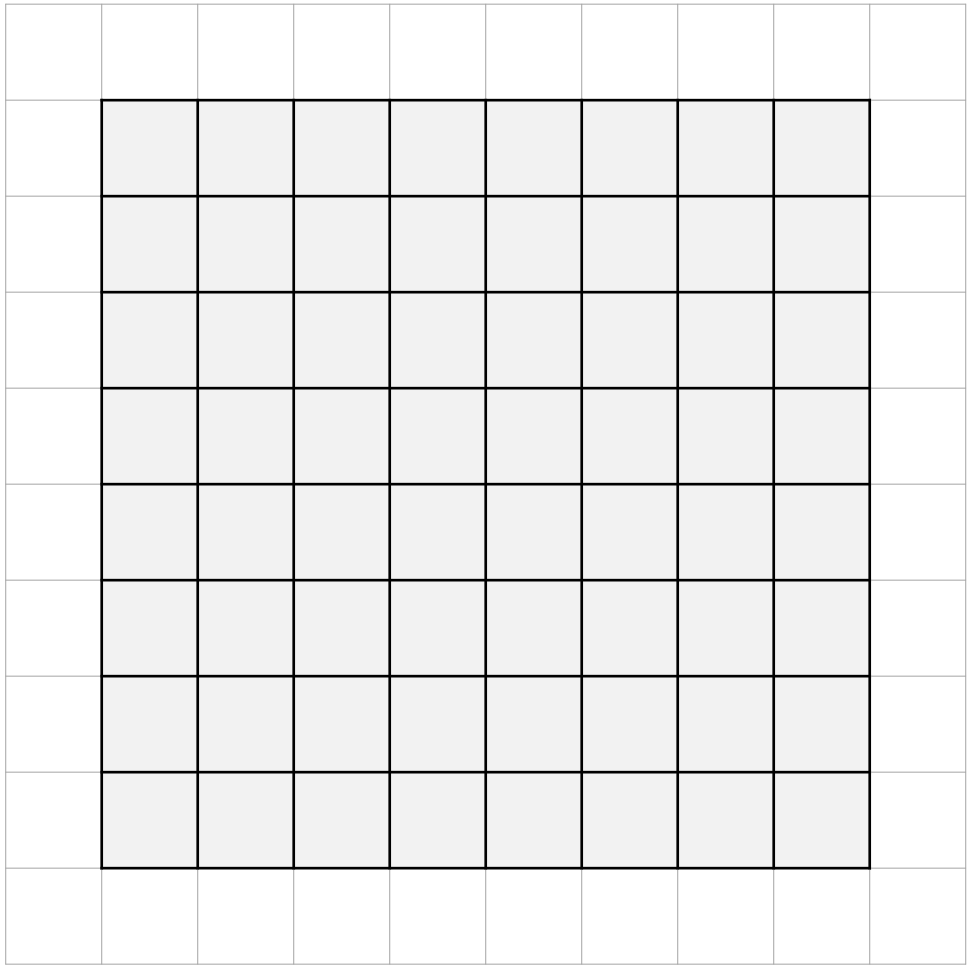
output: H × W = 8 × 8



$$H_{out} = H_{in} + 2pad_h - K_h + 1$$

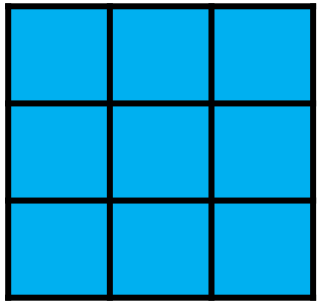
Convolution: stride

input

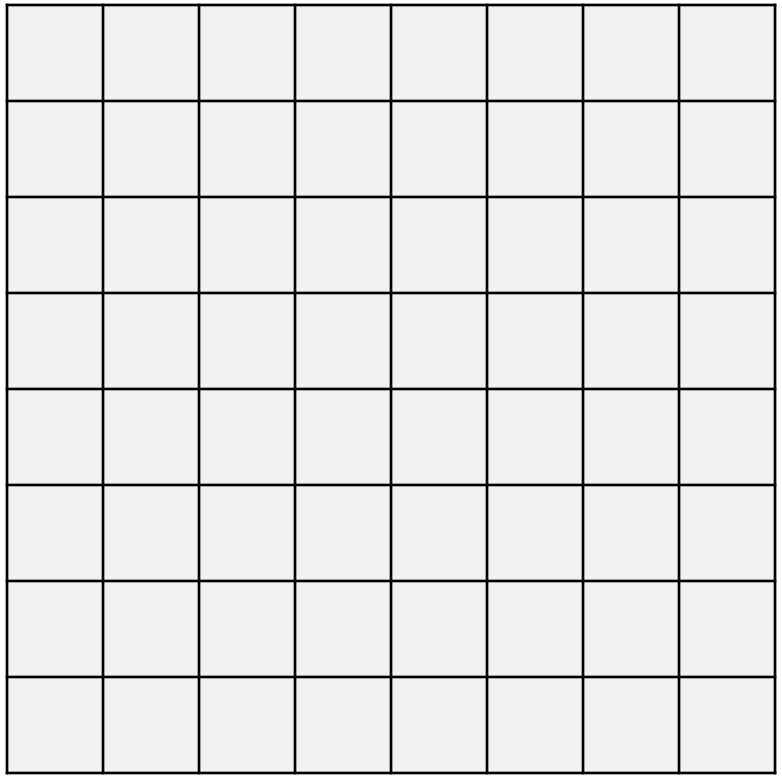


stride = 2

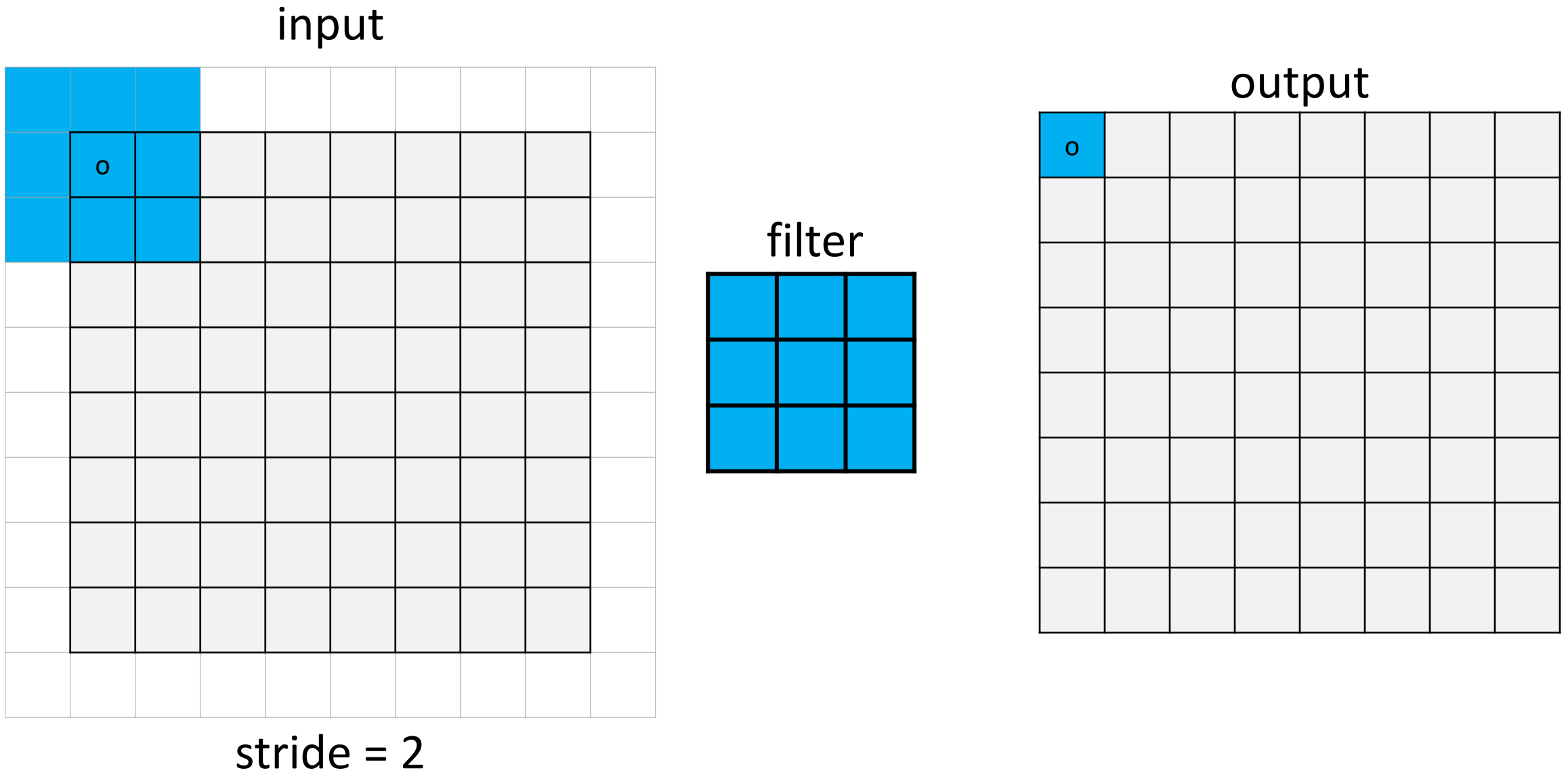
filter



output

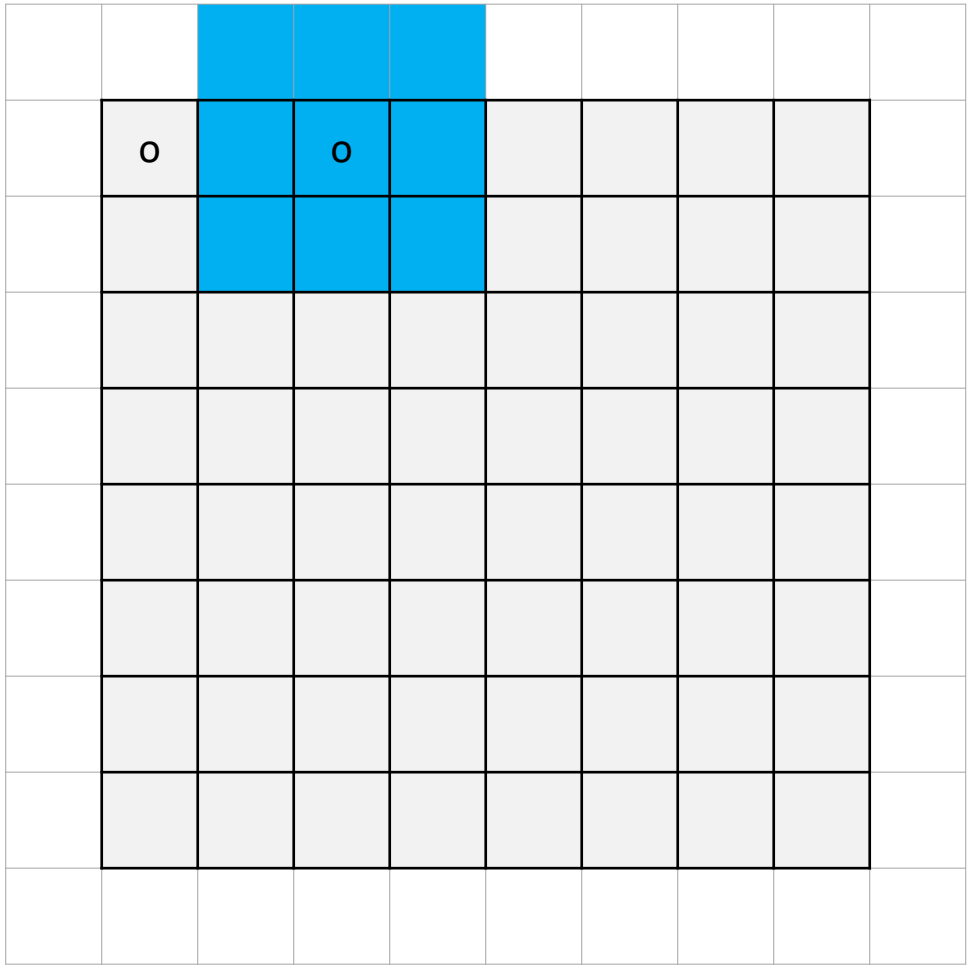


Convolution: stride



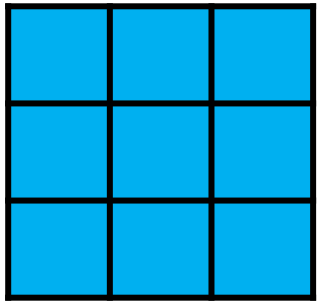
Convolution: stride

input

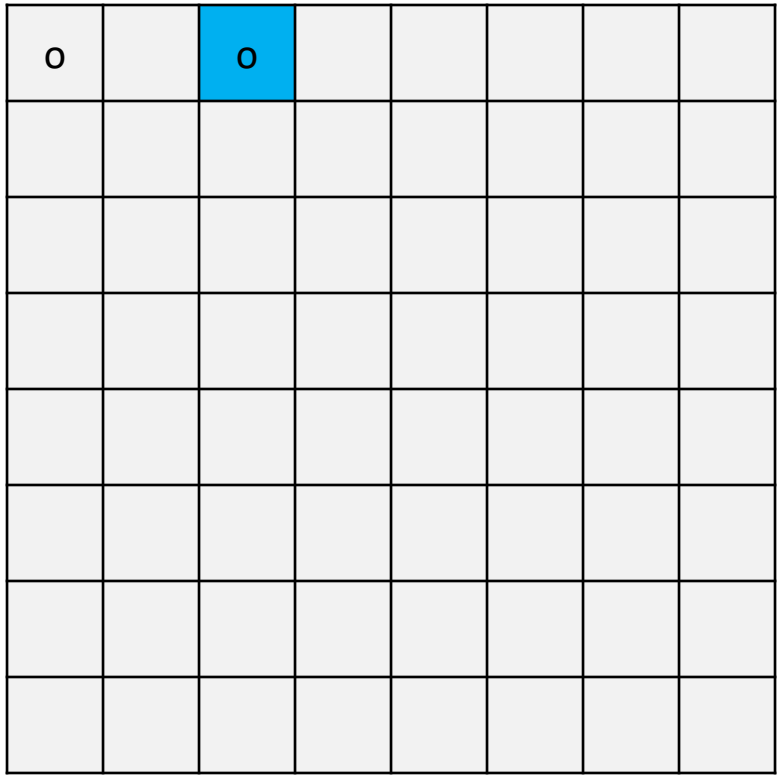


stride = 2

filter

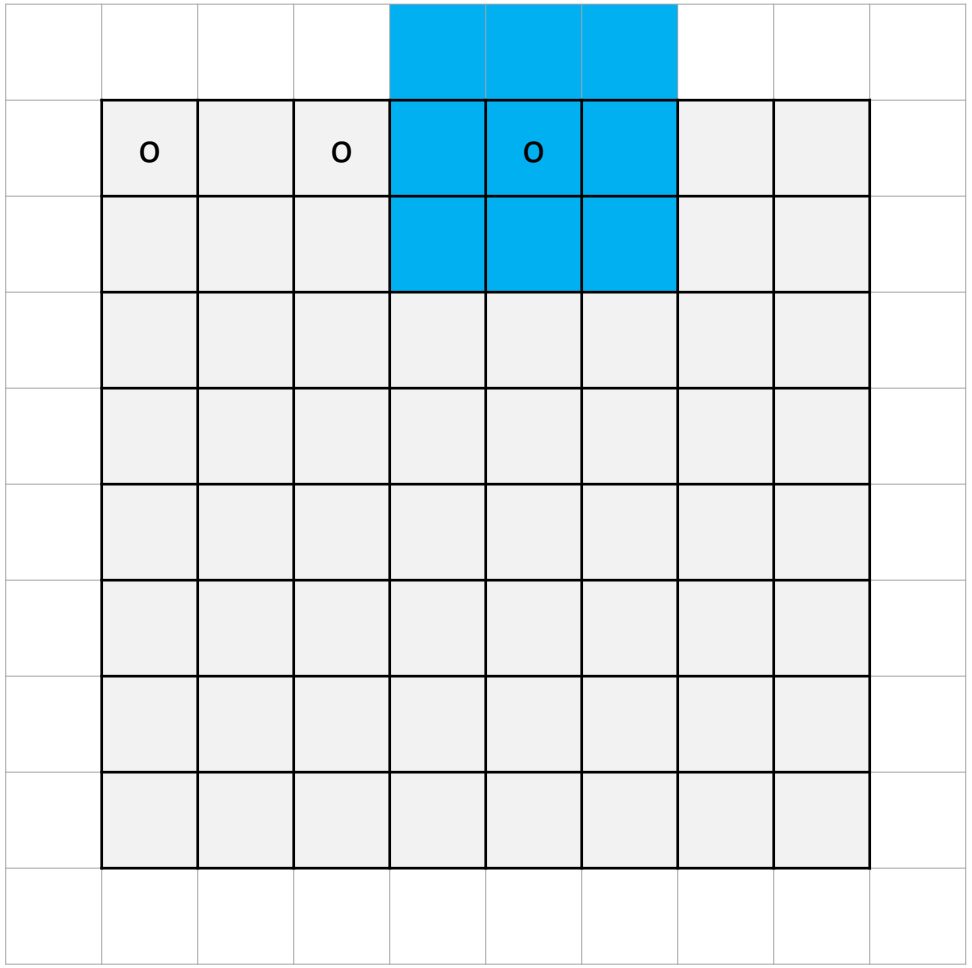


output



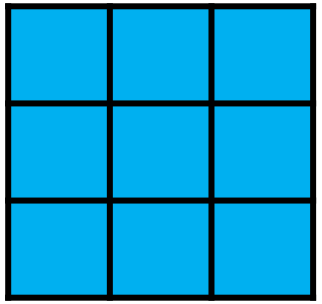
Convolution: stride

input

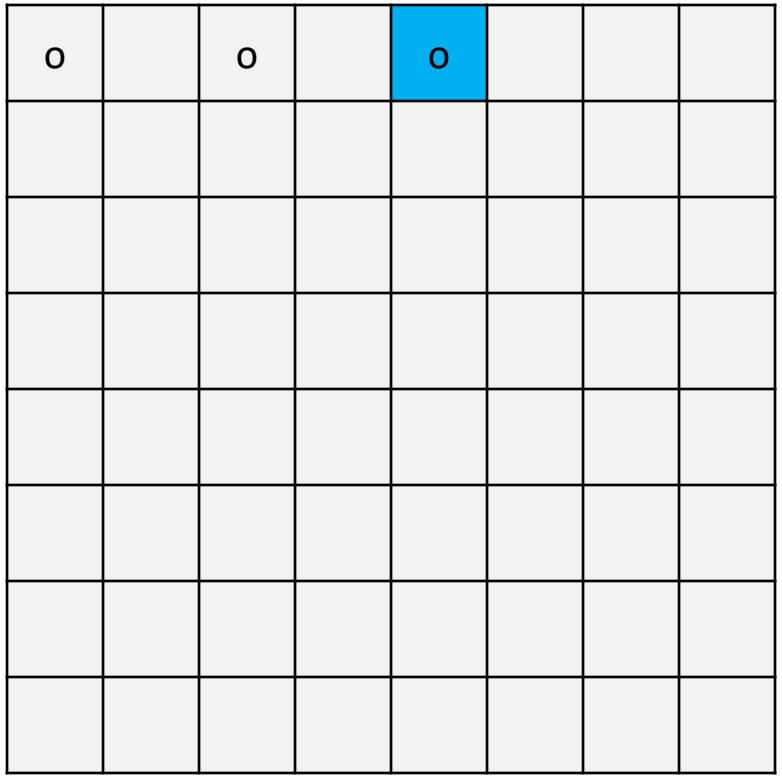


stride = 2

filter

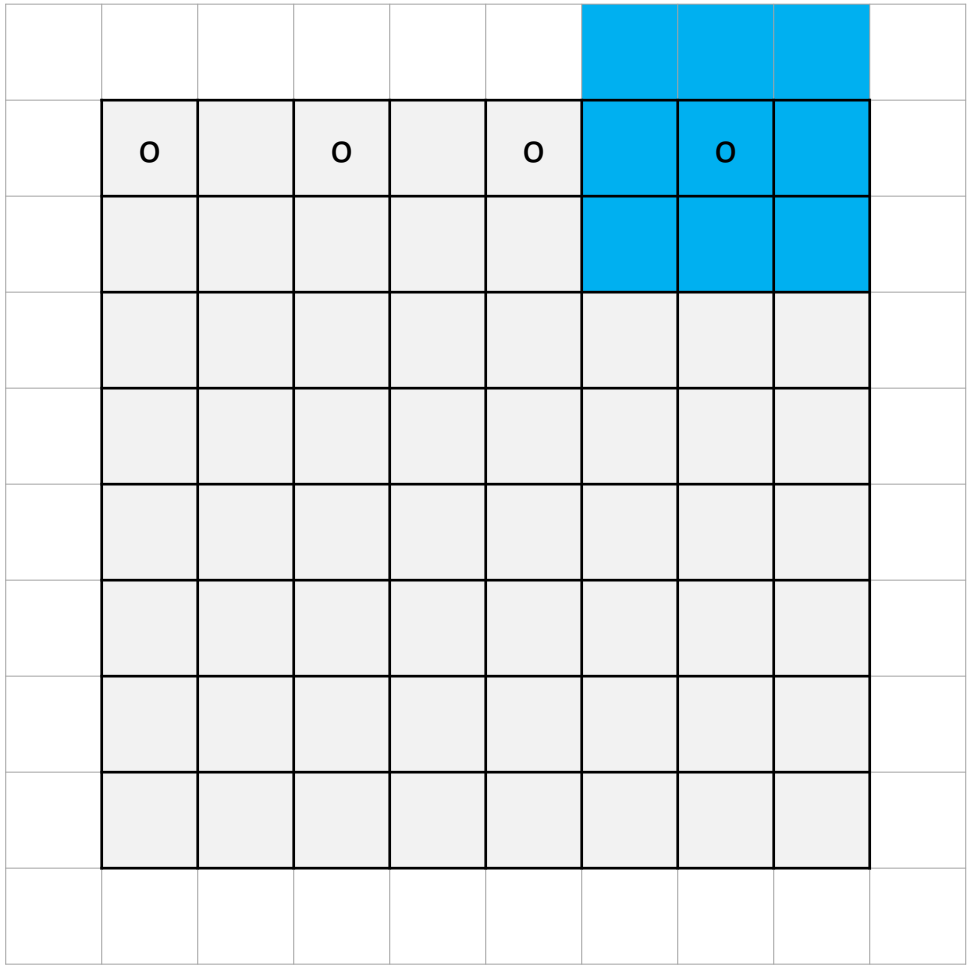


output



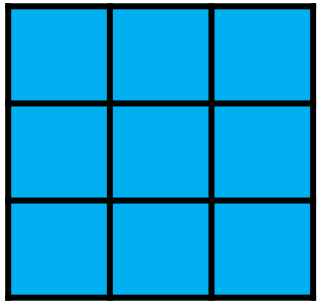
Convolution: stride

input

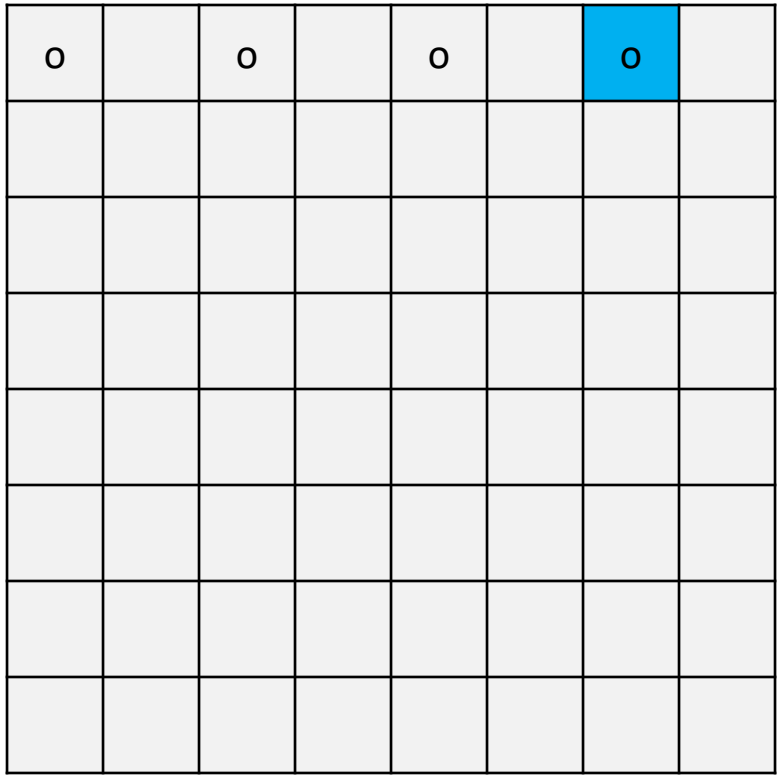


stride = 2

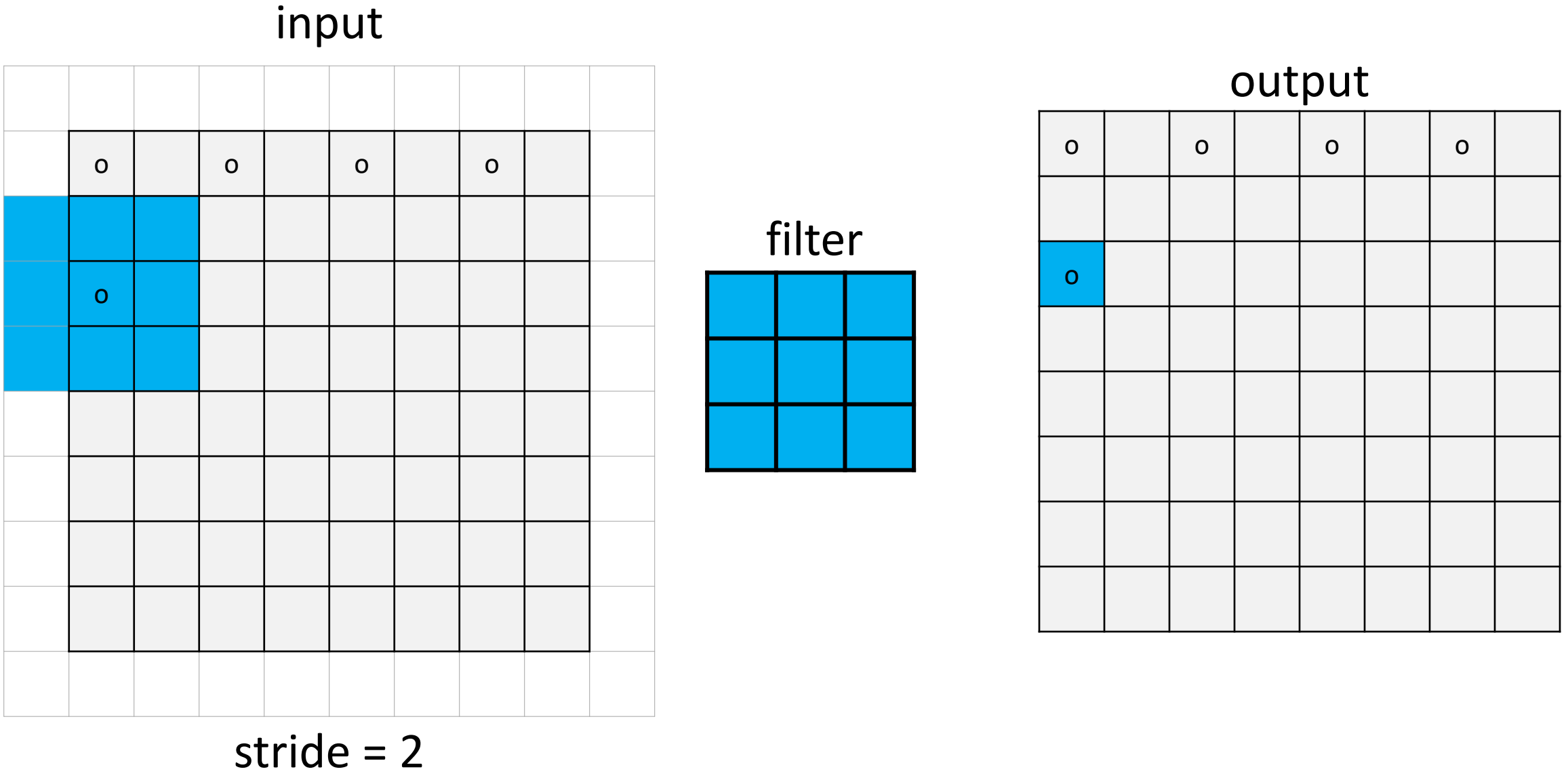
filter



output

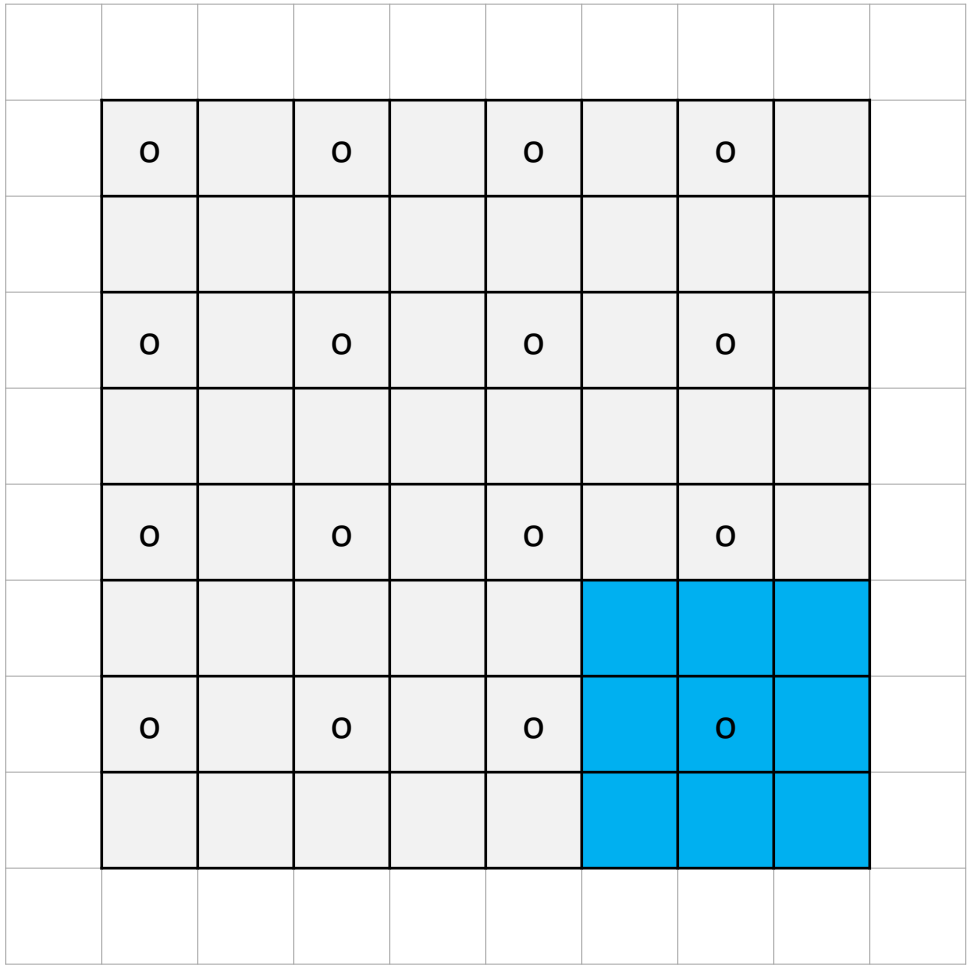


Convolution: stride

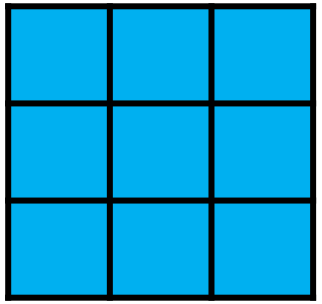


Convolution: stride

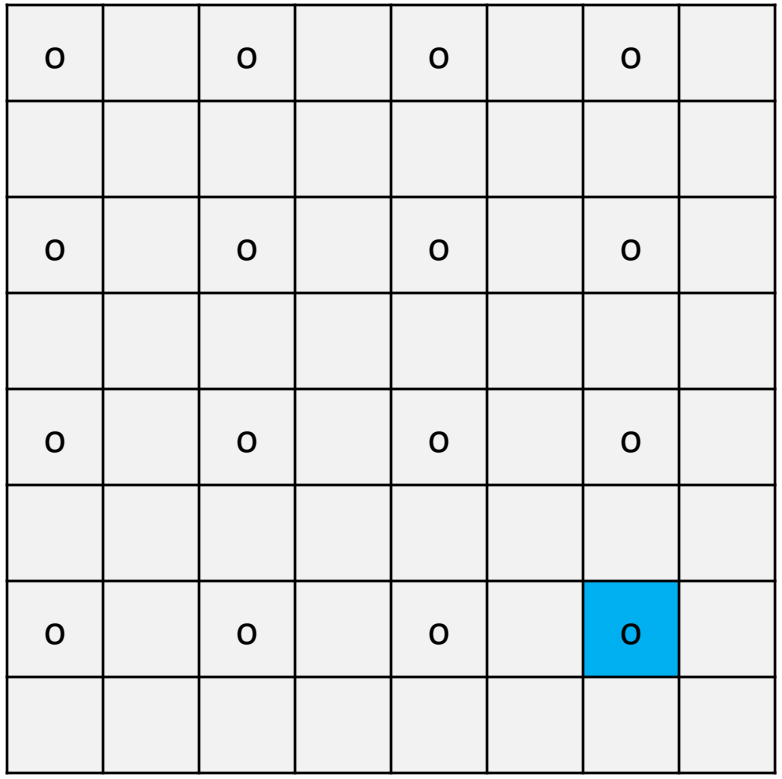
input



filter

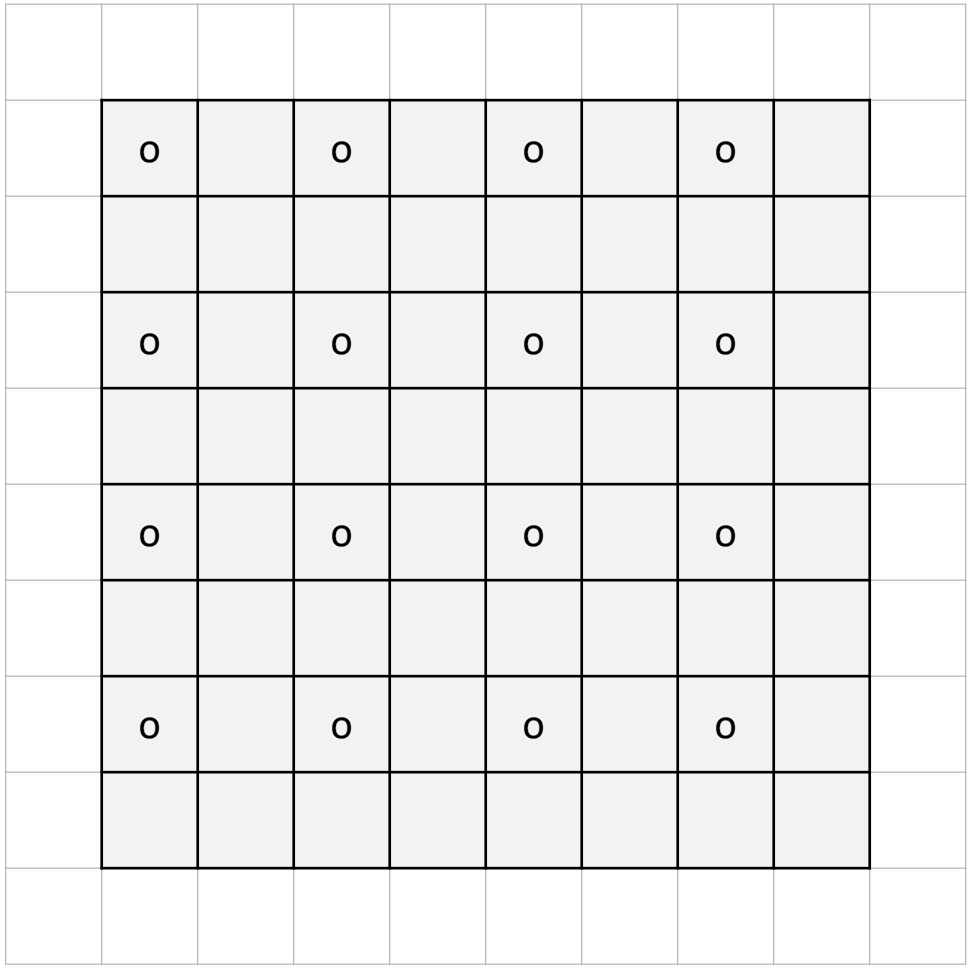


output



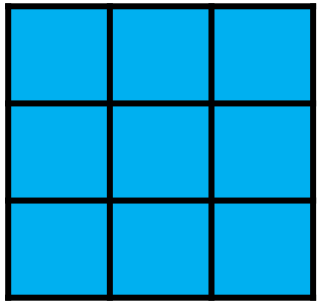
Convolution: stride

input: H × W = 8 × 8

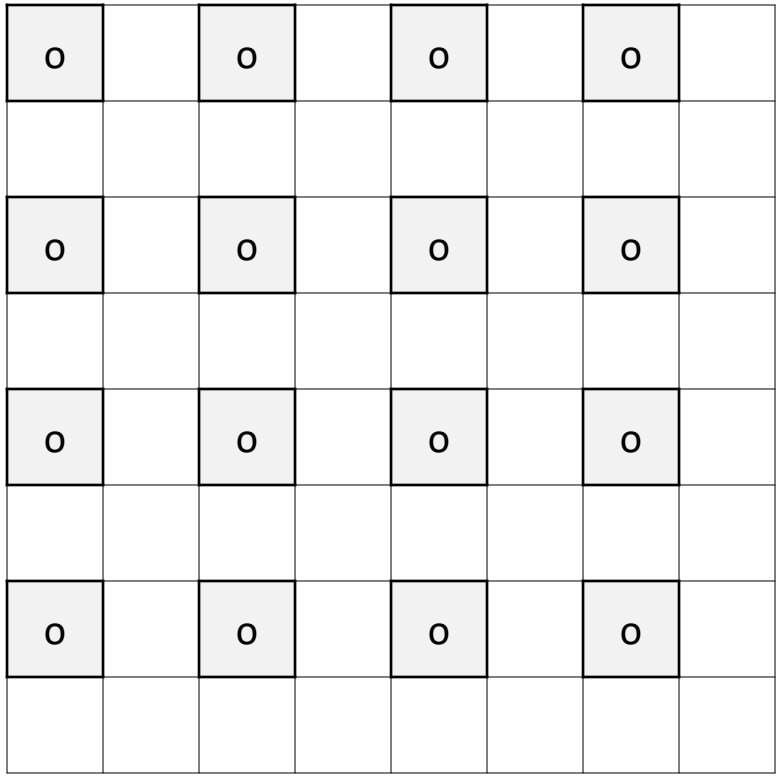


stride = 2

filter



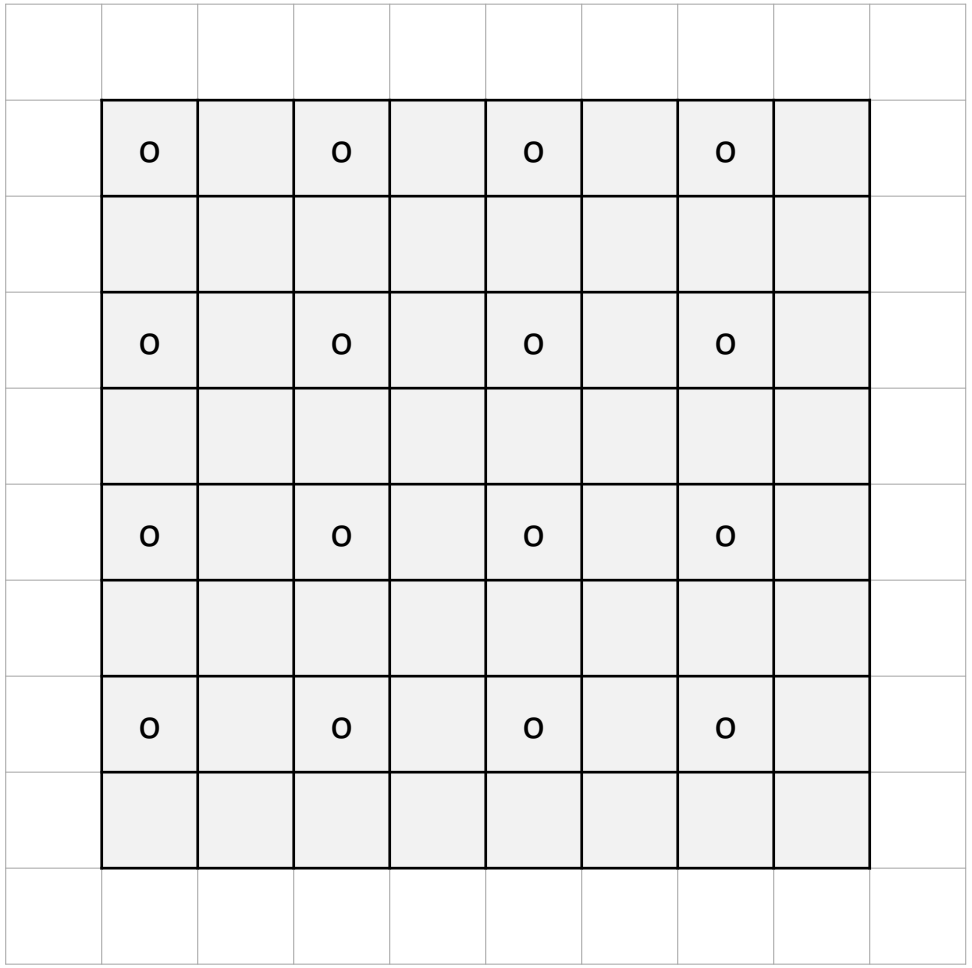
output: H × W = **4 × 4**



Convolution: stride

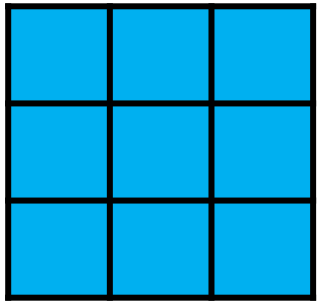
- reduces feature map size
- compress and abstract

input: H × W = 8 × 8



stride = 2

filter



output: H × W = **4 × 4**

o	o	o	o
o	o	o	o
o	o	o	o
o	o	o	o

$$H_{out} = \lfloor (H_{in} + 2pad_h - K_h) / str \rfloor + 1$$

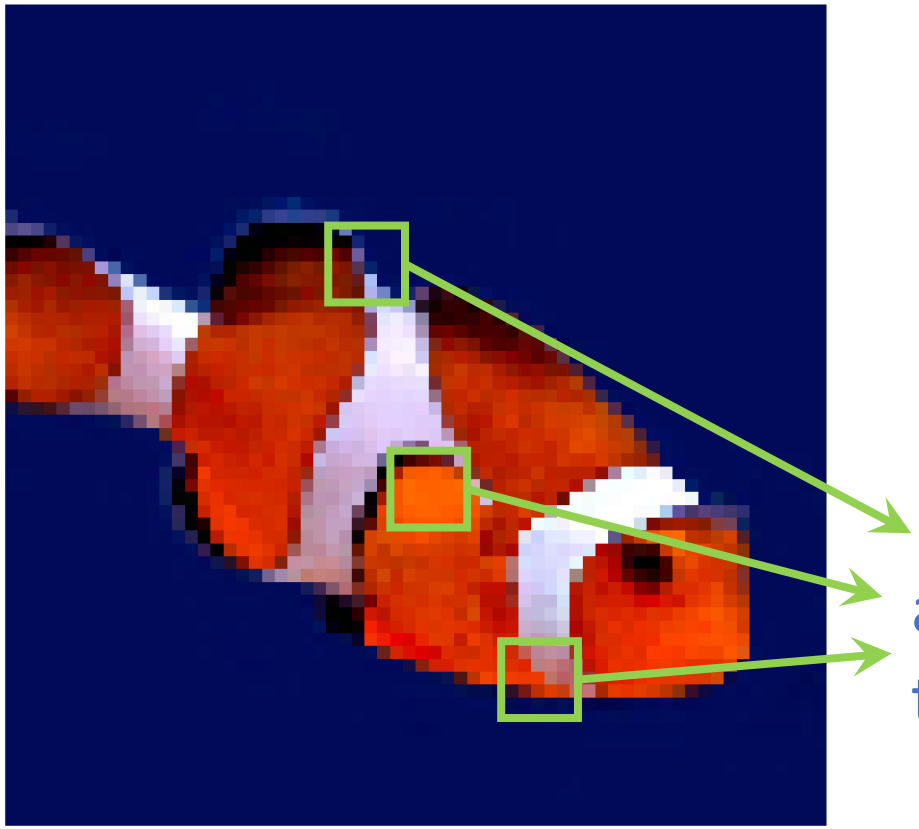
*rounding operation depends on libraries

Convolution: arguments

- kernel size: $K_h \times K_w$ (e.g., 3×3)
- output channels: C_o
  - input channels: C_i determined by input
- padding (e.g., 1)
- stride (e.g., 1 or 2)

Convolution: translation-invariance

- Process each window in the same way



apply the same weights regardless of the window location

Convolution: translation-invariance

- Process each window in the same way

$$\boldsymbol{y}[n,m] = \sum_{i=-r}^{r} \sum_{j=-r}^{r} \boldsymbol{w}[i,j]\boldsymbol{x}[n+i,m+j]$$

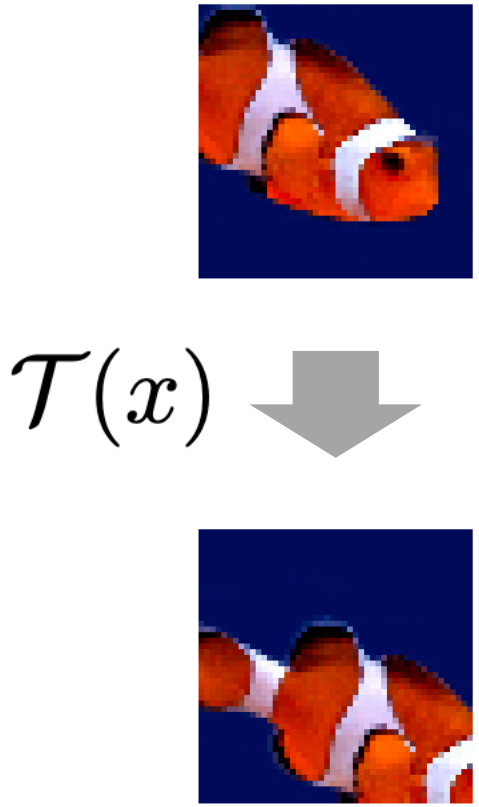
apply the same weights regardless of the window location

Convolution: translation-equivariance

- Equivariance: $f(\mathcal{T}(x)) = \mathcal{T}(f(x))$

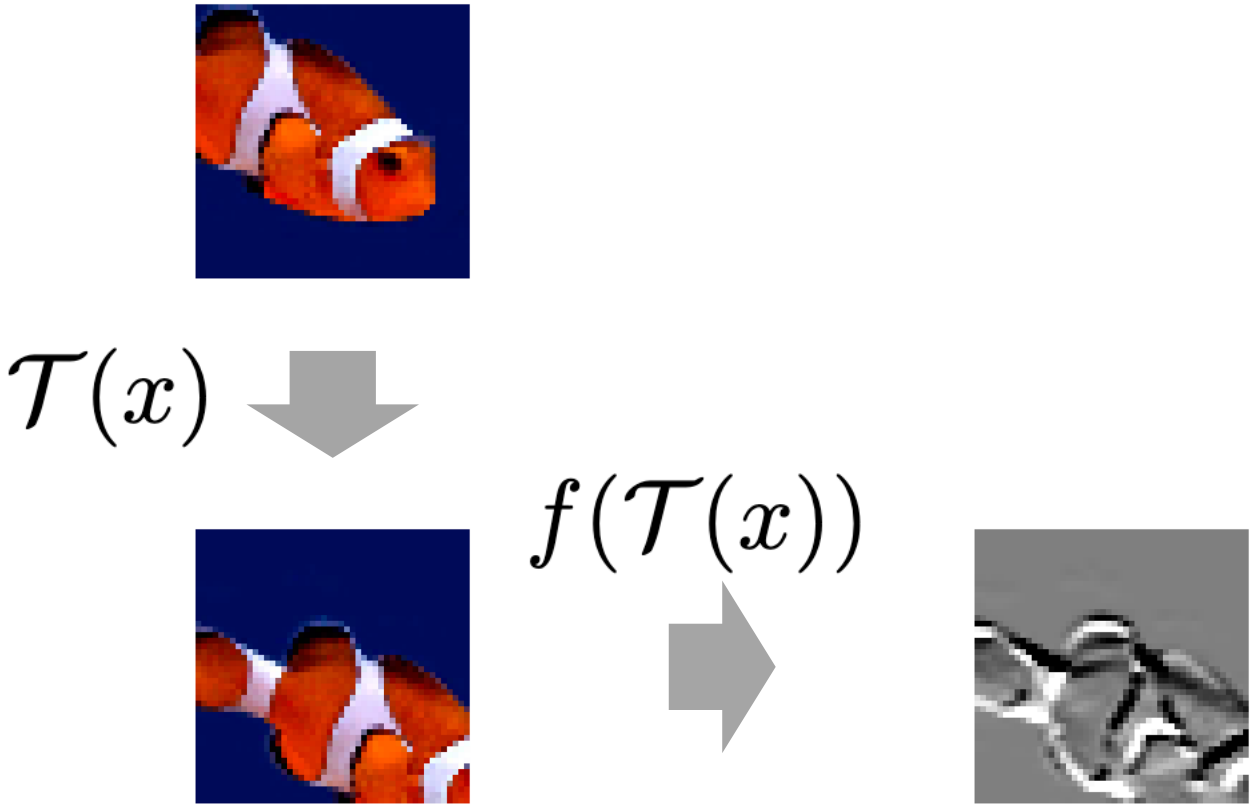
Convolution: translation-equivariance

- Equivariance: $f(\mathcal{T}(x)) = \mathcal{T}(f(x))$



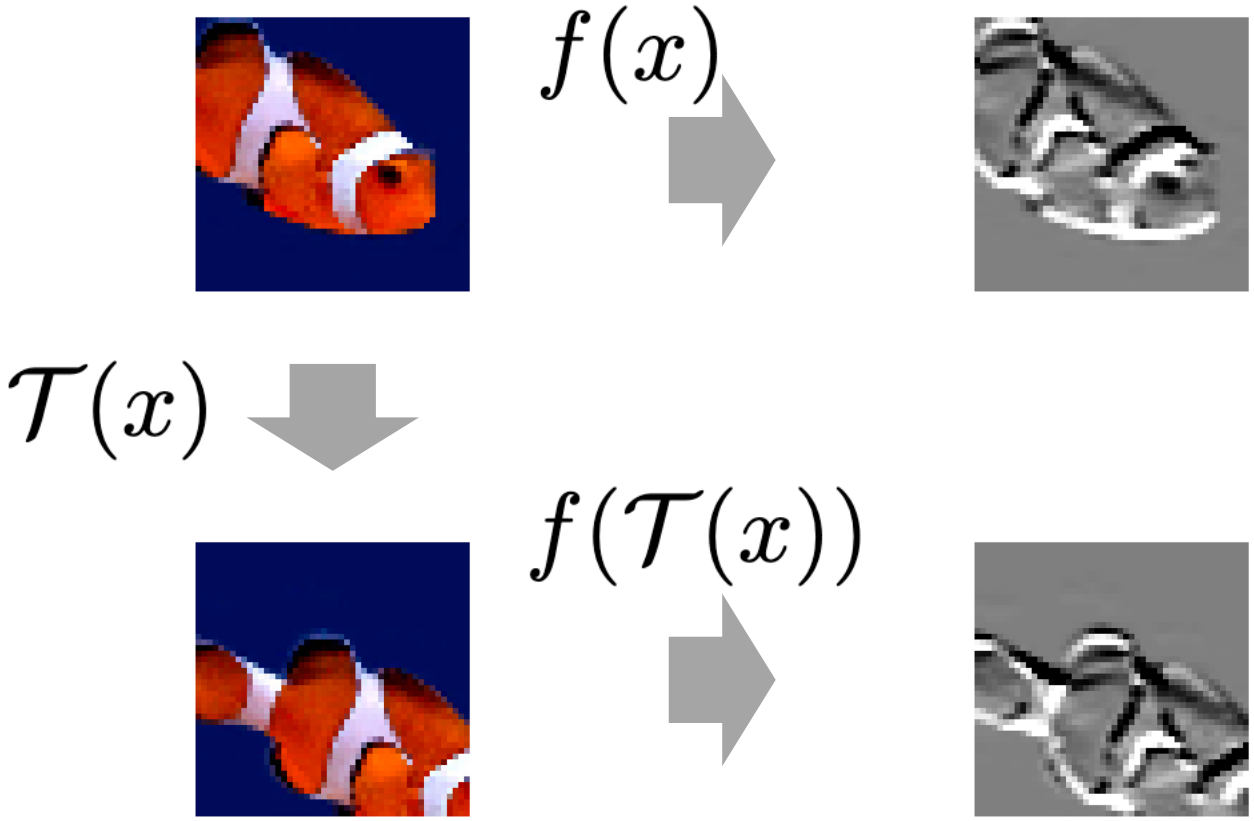
Convolution: translation-equivariance

- Equivariance: $f(\mathcal{T}(x)) = \mathcal{T}(f(x))$



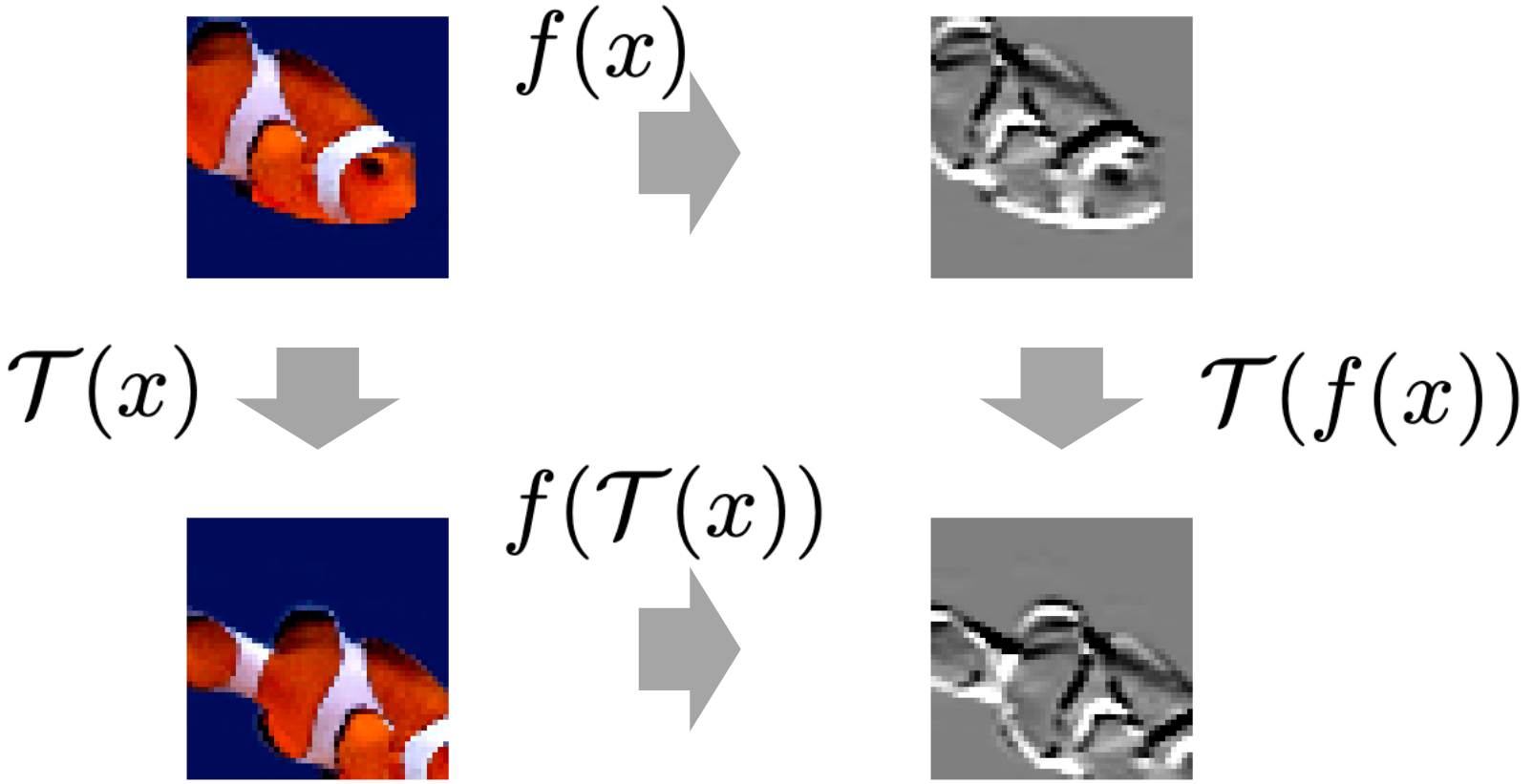
Convolution: translation-equivariance

- Equivariance: $f(\mathcal{T}(x)) = \mathcal{T}(f(x))$



Convolution: translation-equivariance

- Equivariance: $f(\mathcal{T}(x)) = \mathcal{T}(f(x))$

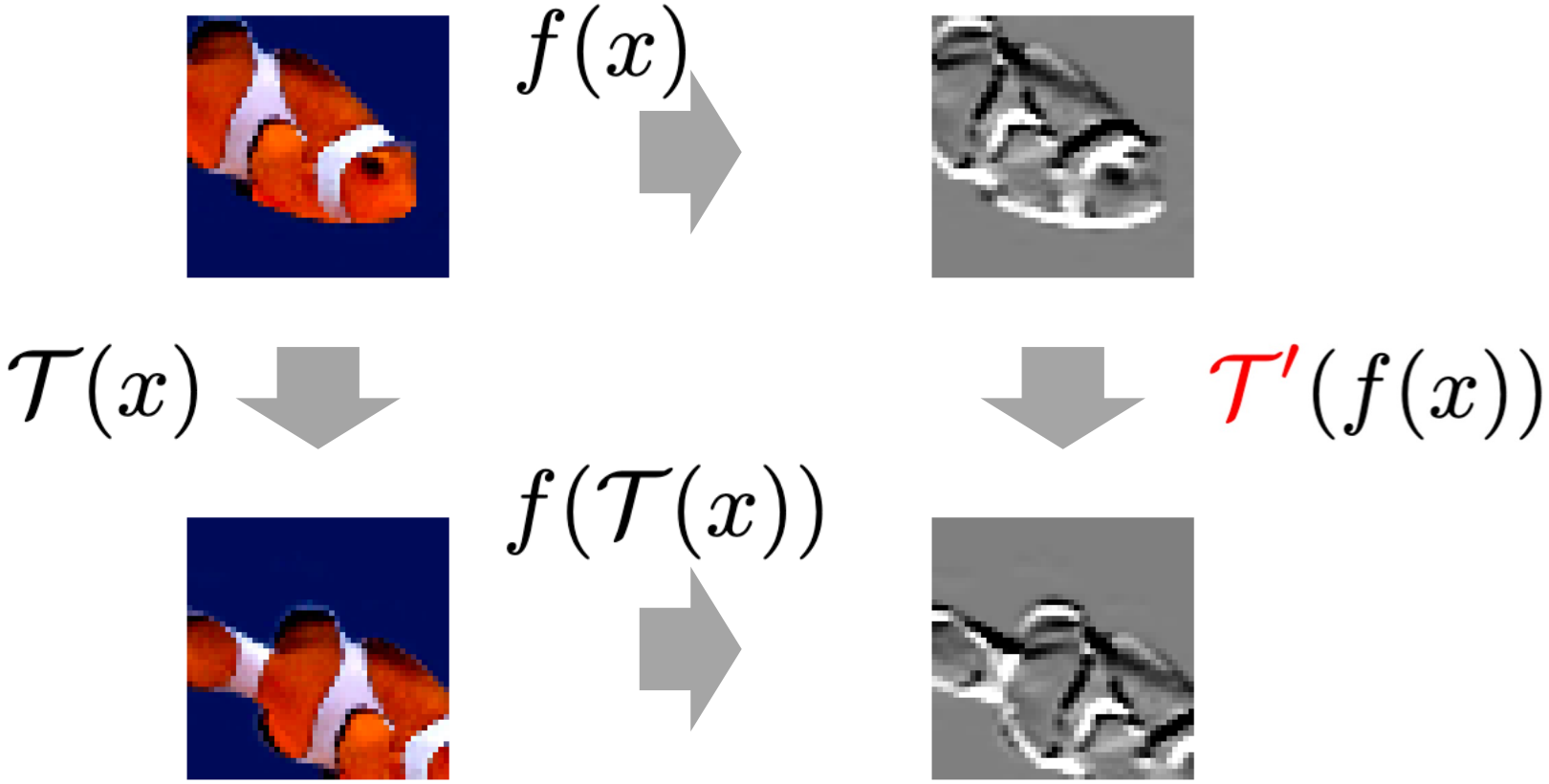


* assuming infinite canvas

Convolution: translation-equivariance

- Equivariance: $f(\mathcal{T}(x)) = \mathcal{T}'(f(x))$

$\mathcal{T}'$ does not depend on x



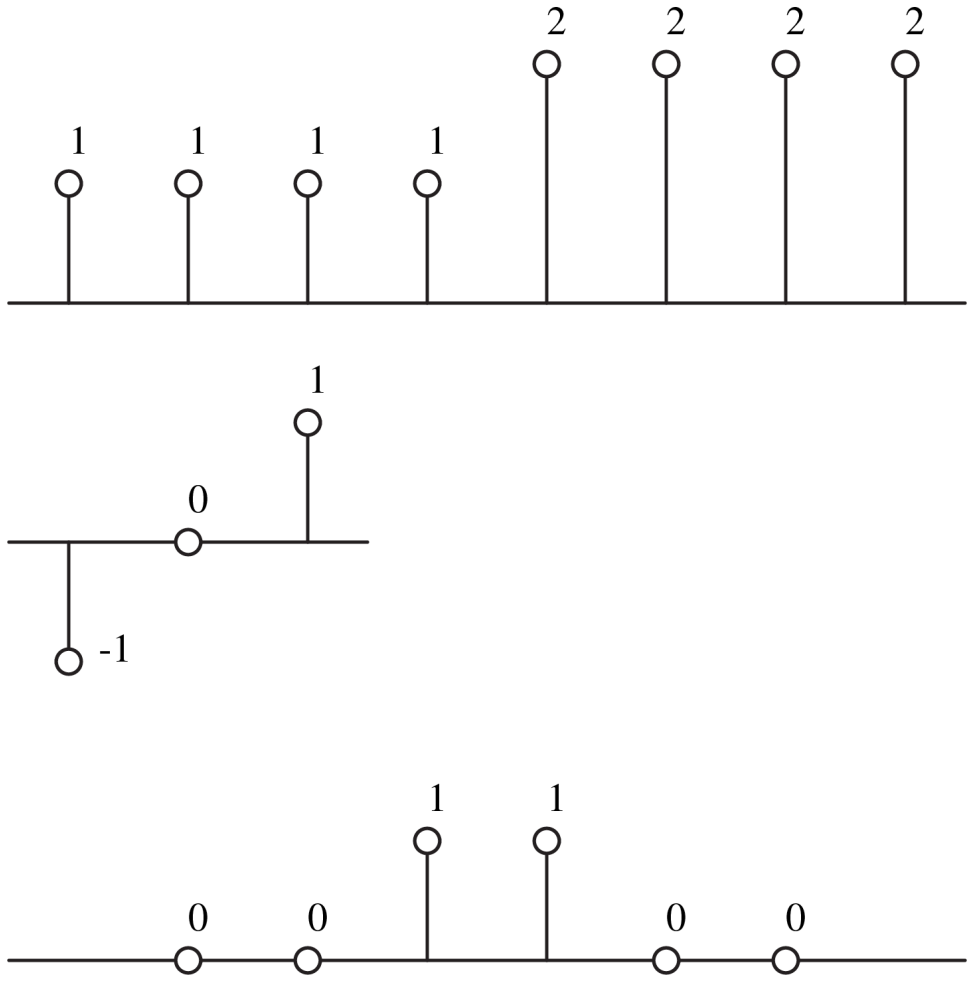
* assuming infinite canvas

Convolution = local connection + weight-sharing

input

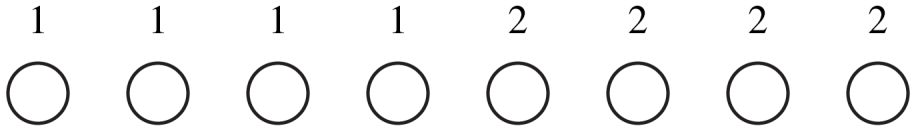
filter

output

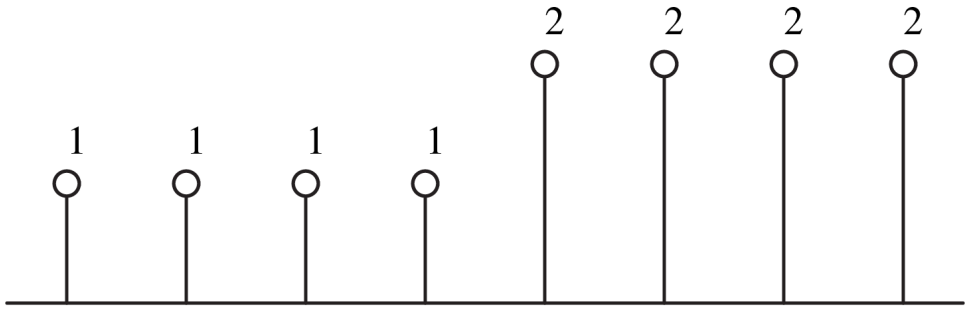


Convolution = local connection + weight-sharing

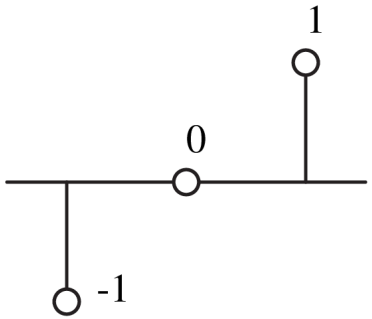
Let’s plot it in a different way!



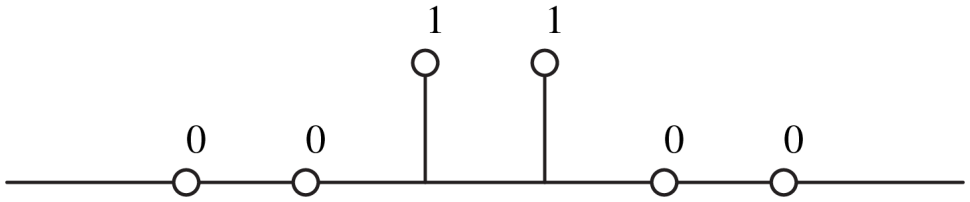
input



filter

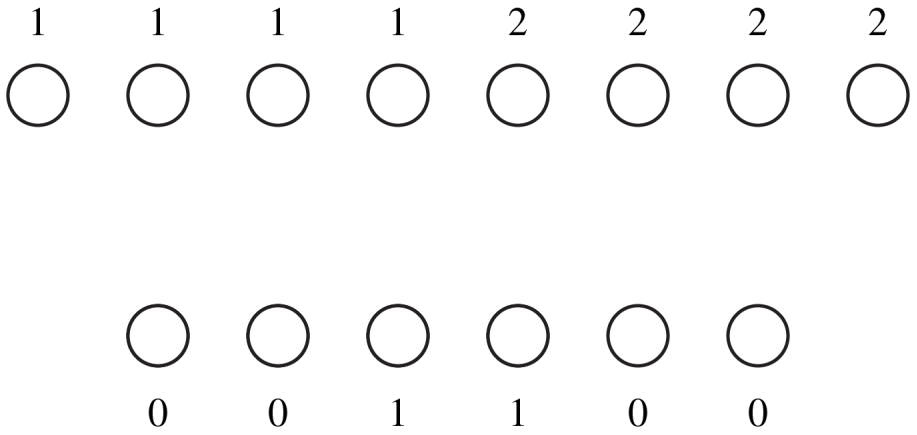


output

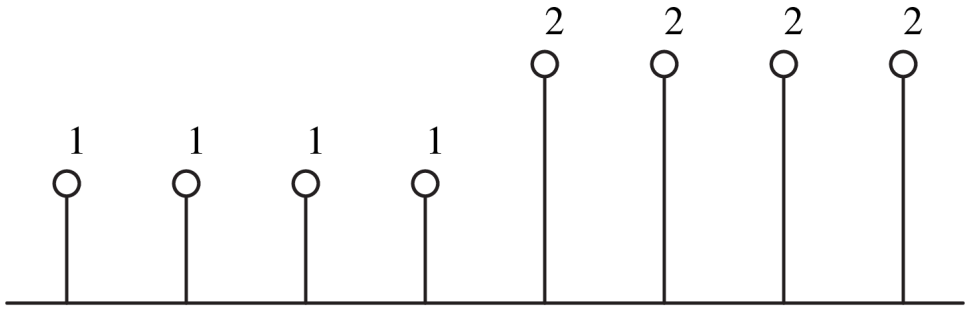


Convolution = local connection + weight-sharing

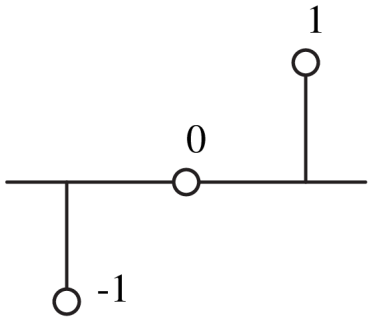
Let’s plot it in a different way!



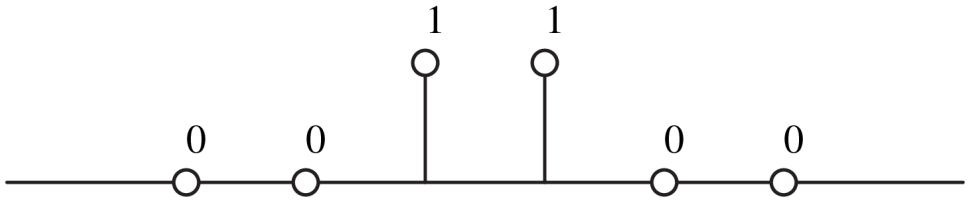
input



filter

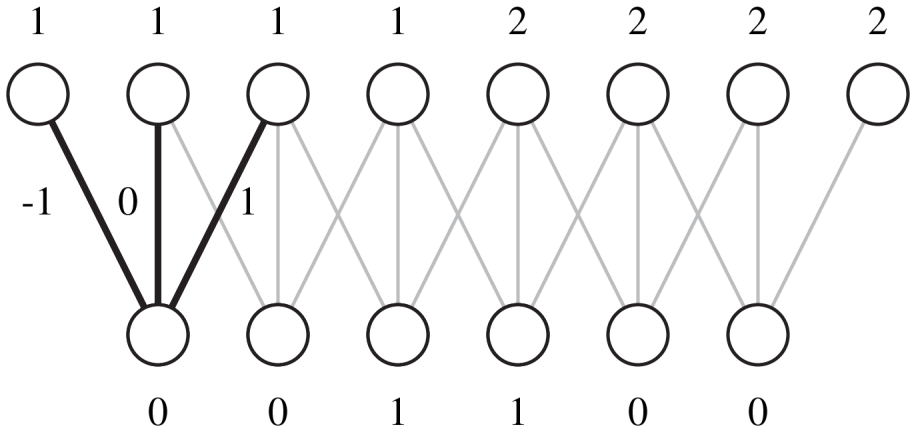


output

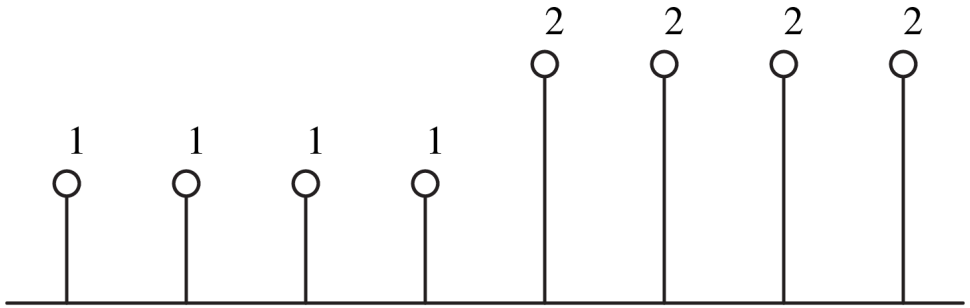


Convolution = local connection + weight-sharing

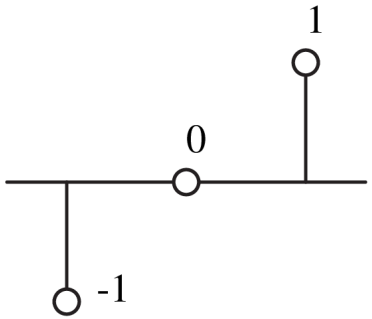
Let’s plot it in a different way!



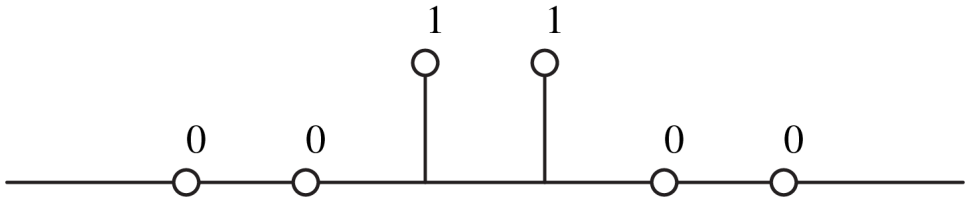
input



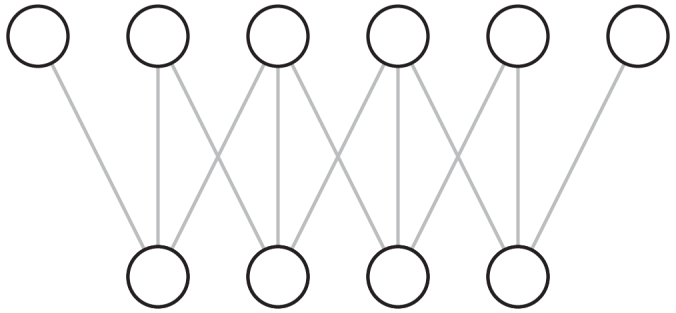
filter



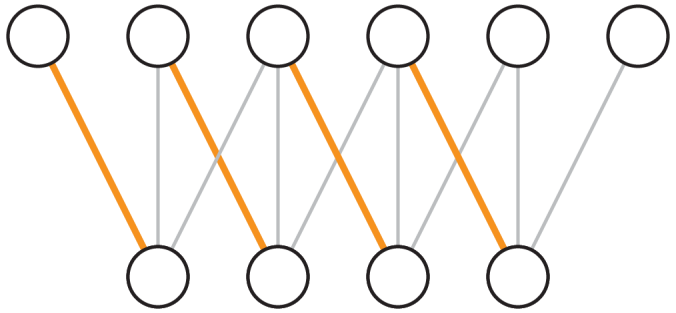
output



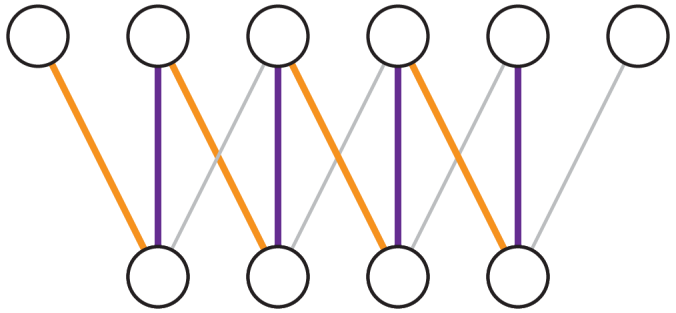
Convolution = local connection + weight-sharing



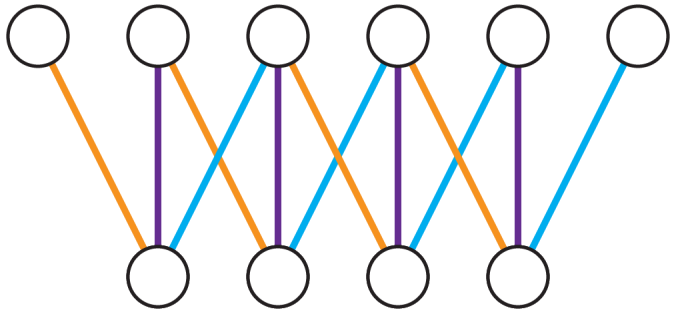
Convolution = local connection + weight-sharing



Convolution = local connection + weight-sharing

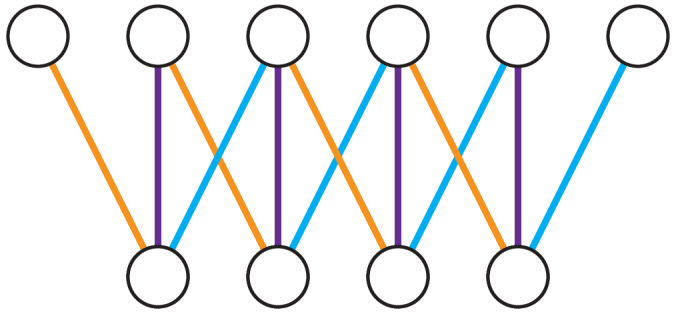


Convolution = local connection + weight-sharing



Convolution = local connection + weight-sharing

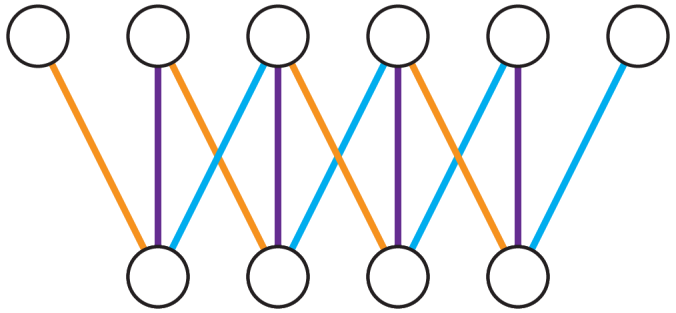
convolution



params = 3 here

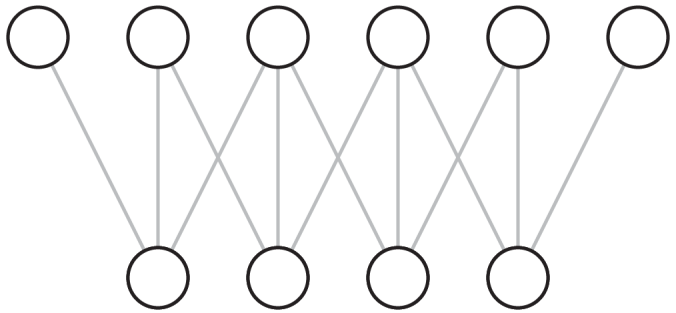
Convolution = local connection + weight-sharing

convolution



params = 3 here

locally connected

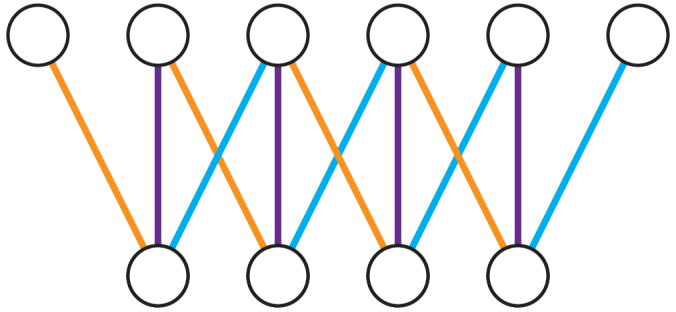


params = 12 here

no weight-sharing

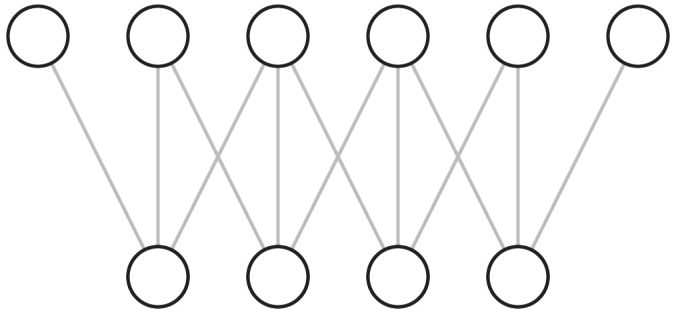
Convolution = local connection + weight-sharing

convolution



params = 3 here

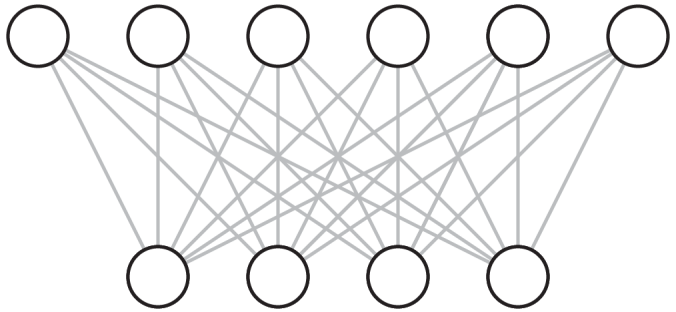
locally connected



params = 12 here

no weight-sharing

fully connected

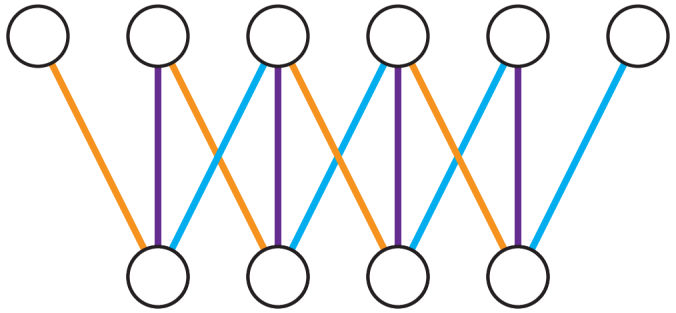


params = 24 here

no weight-sharing

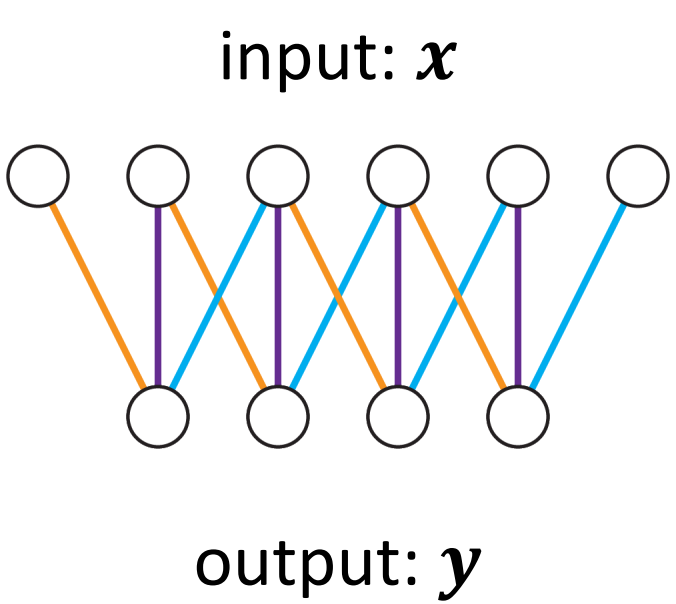
Convolution: linear transform

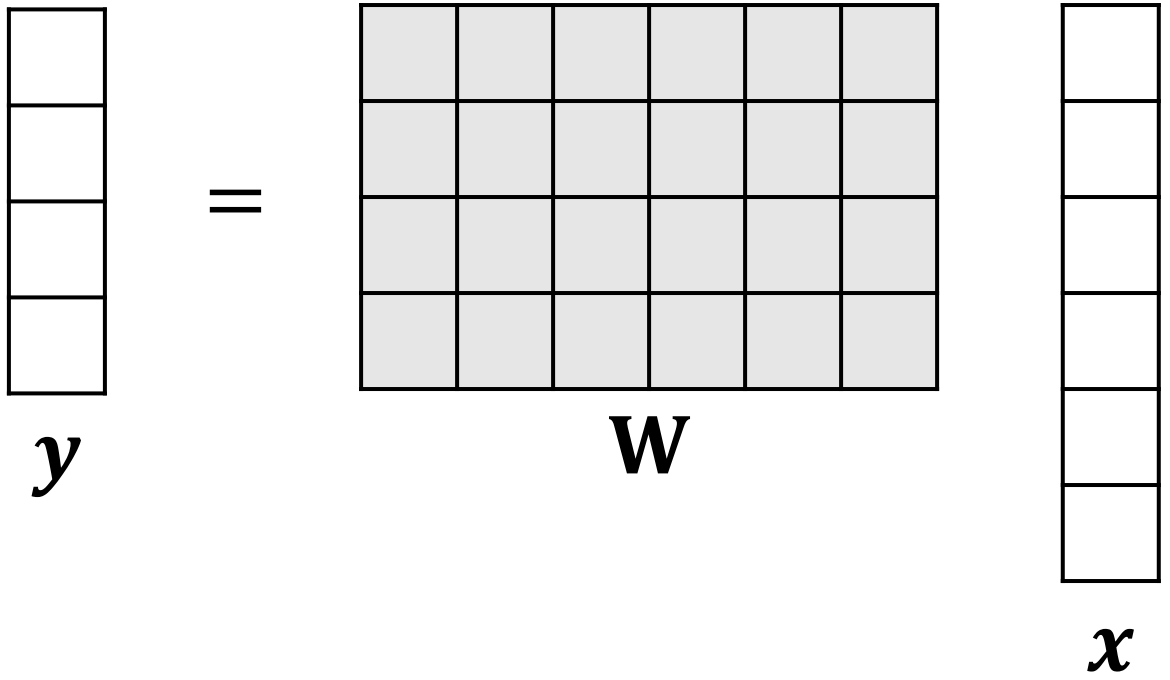
input: $\boldsymbol{x}$



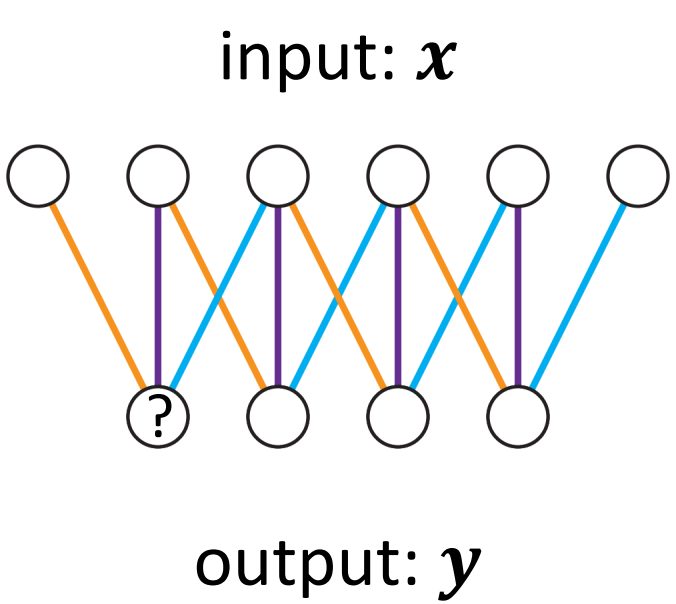
output: $\boldsymbol{y}$

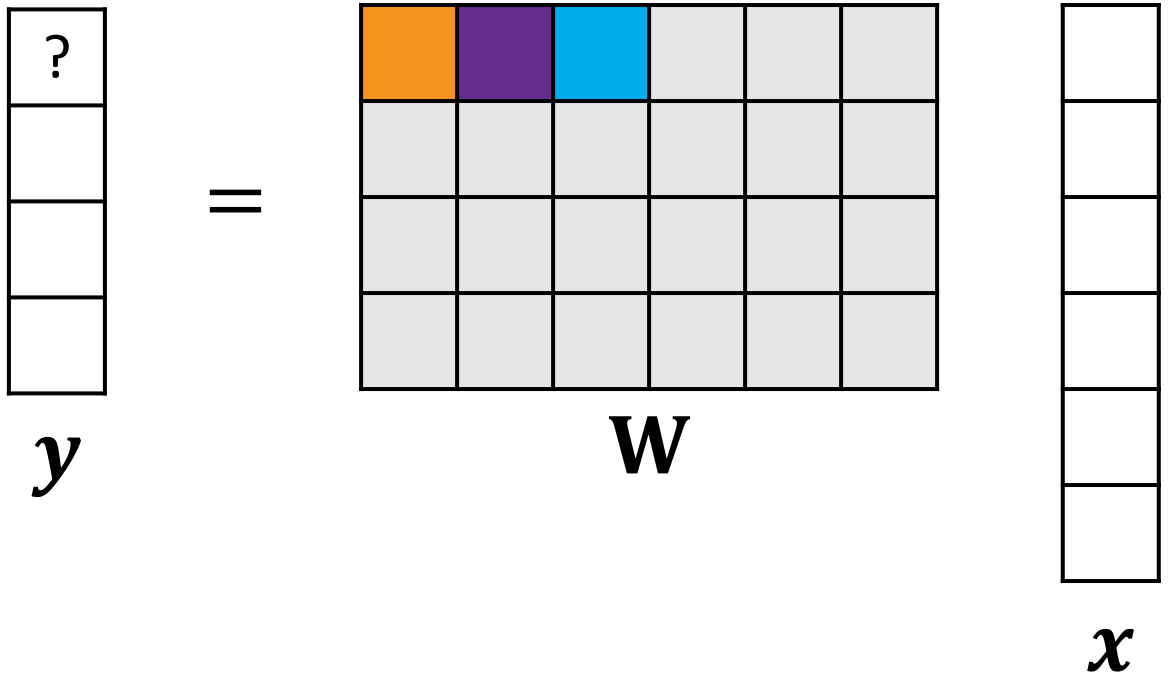
Convolution: linear transform



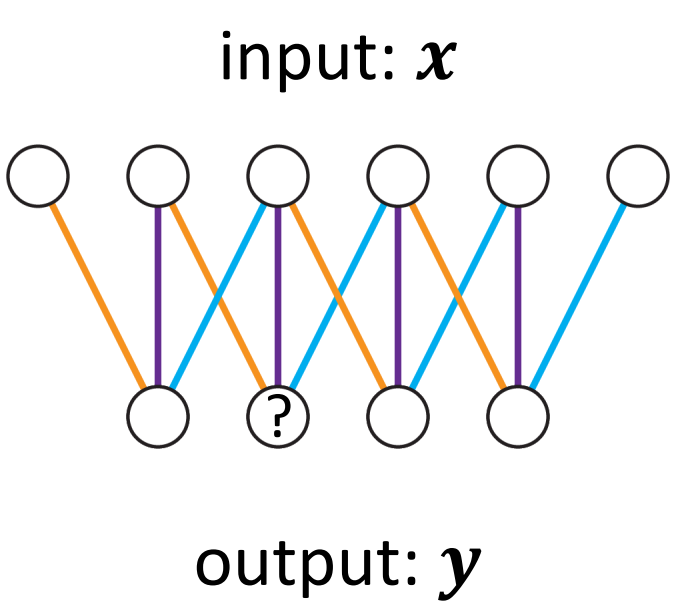


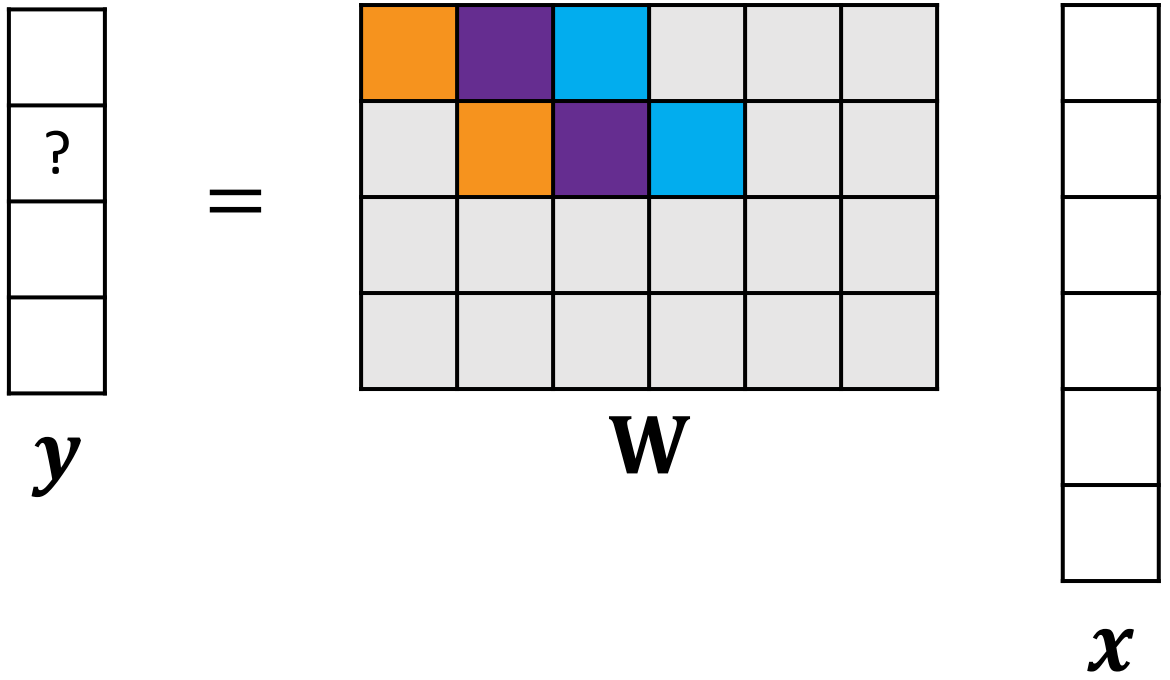
Convolution: linear transform



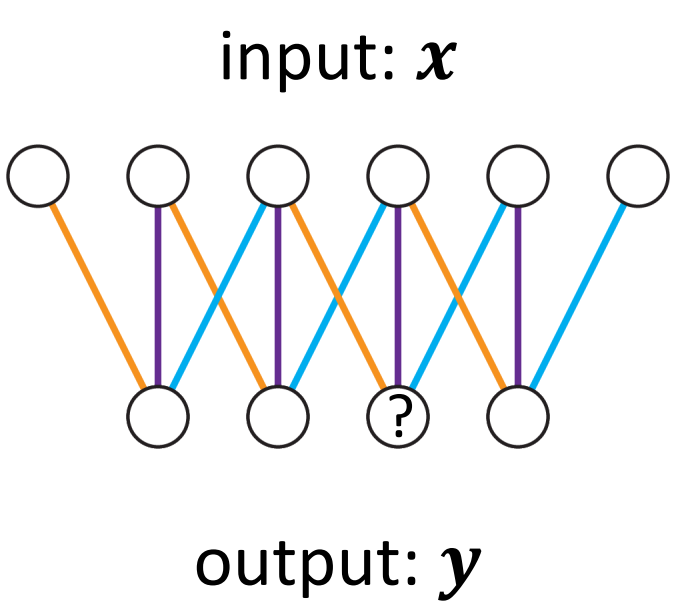


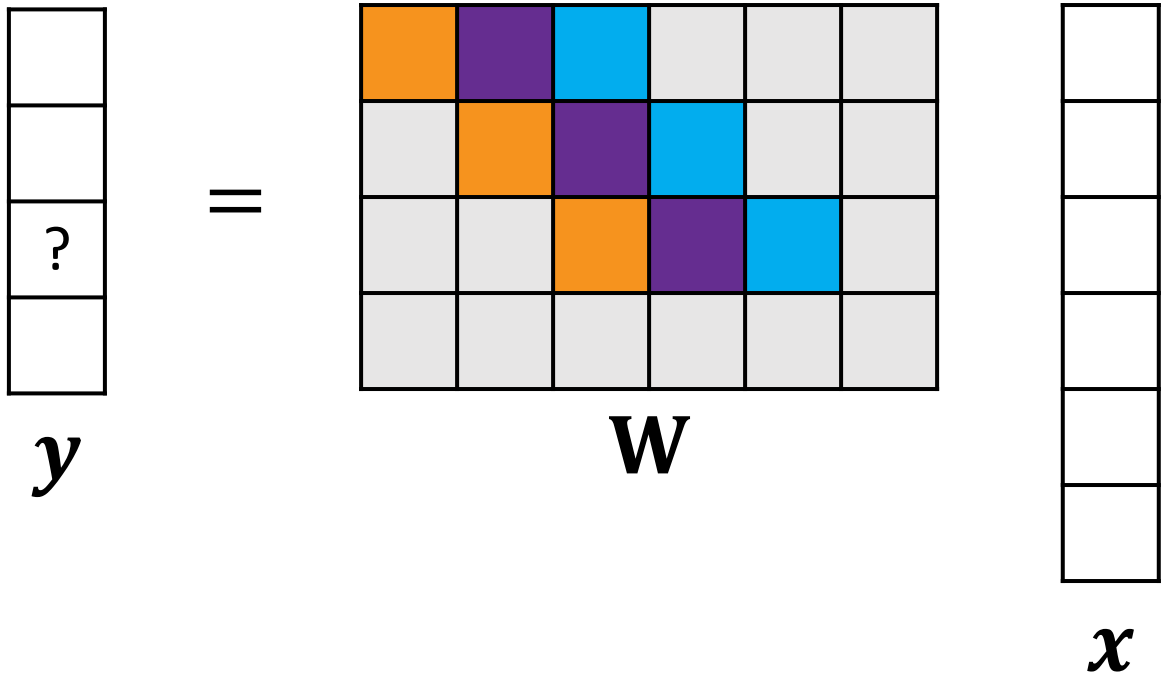
Convolution: linear transform



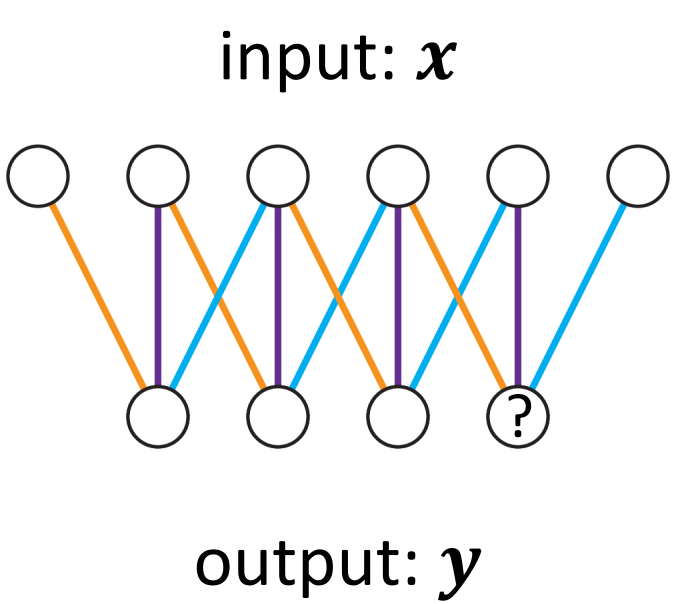


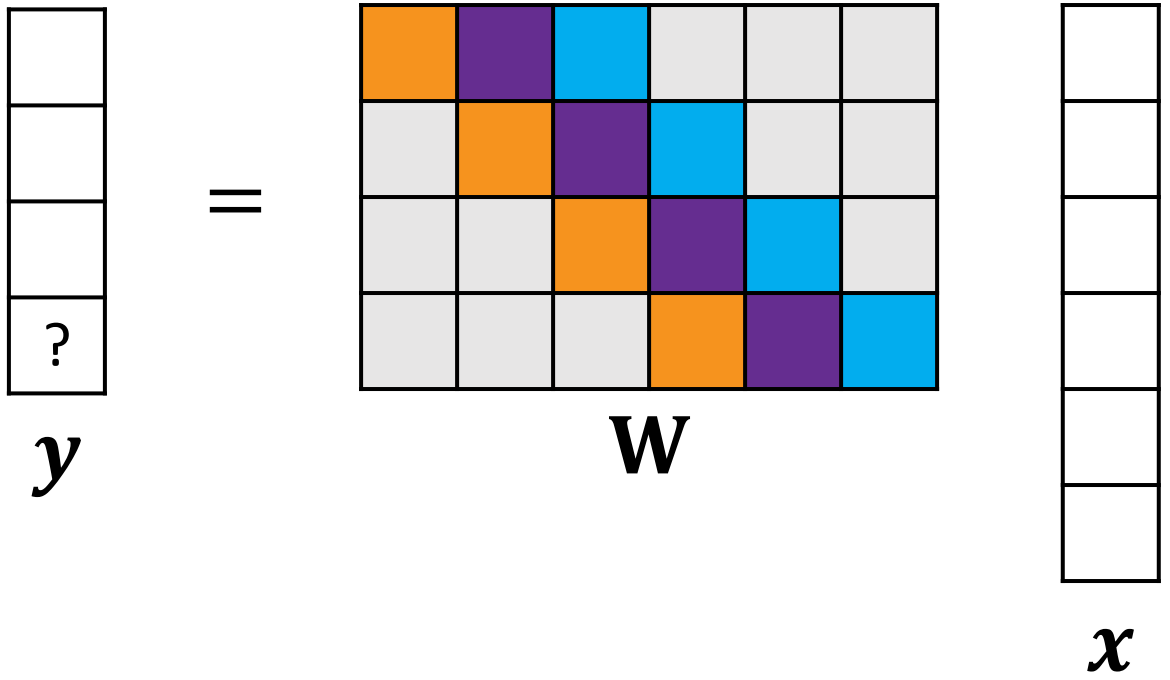
Convolution: linear transform



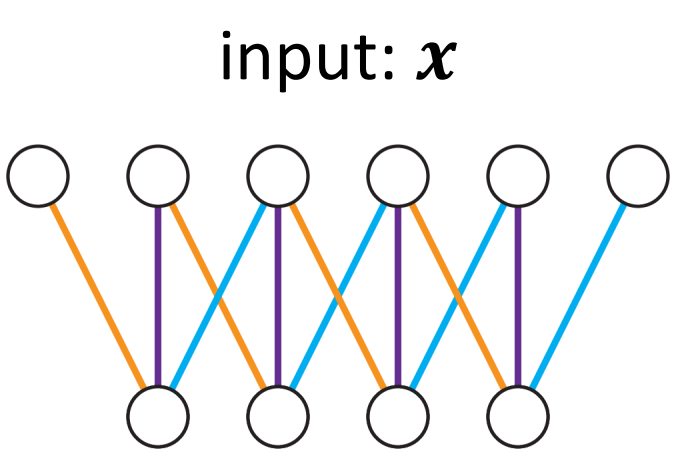


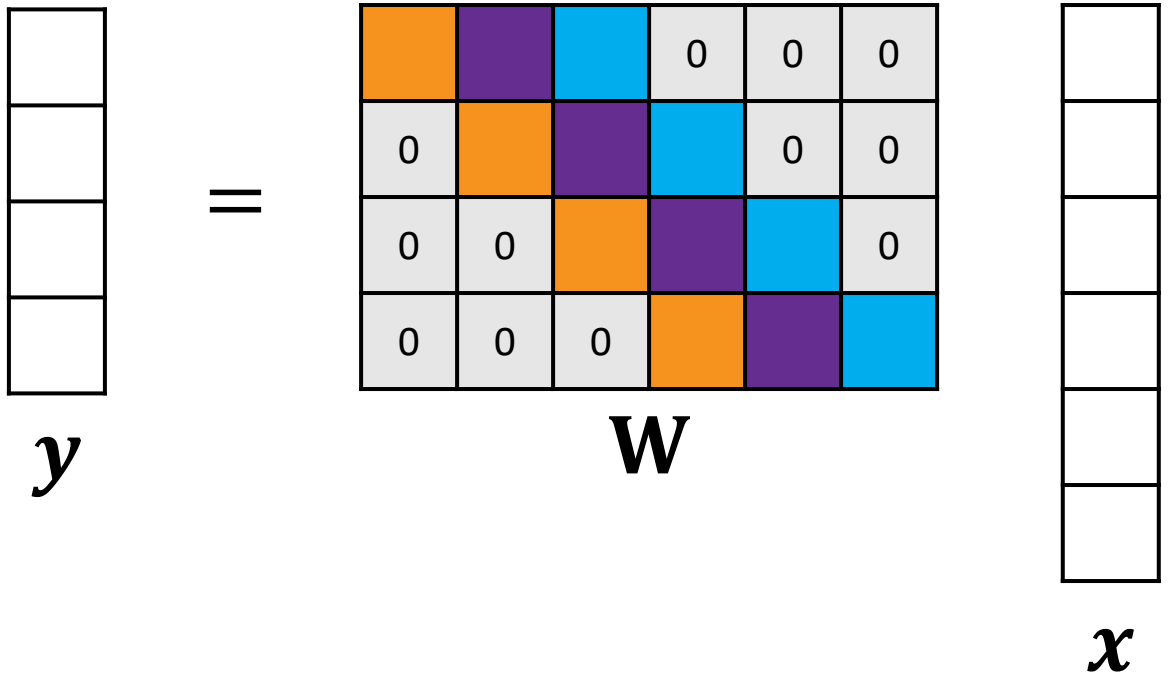
Convolution: linear transform





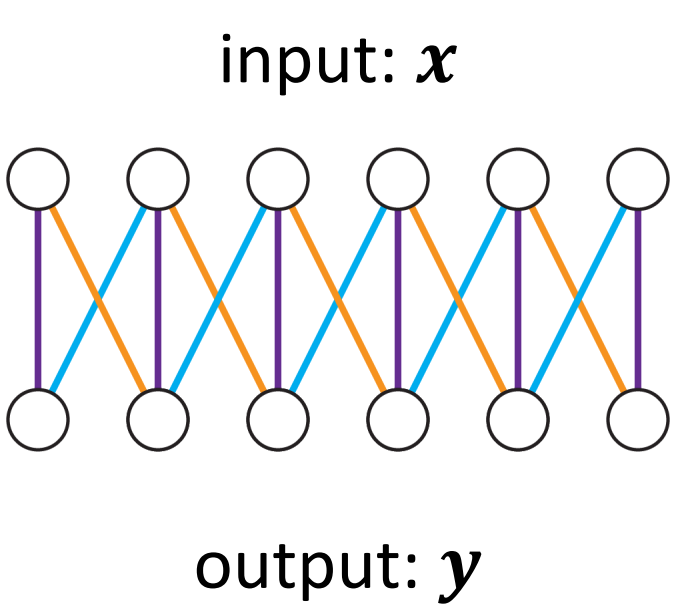
Convolution: linear transform

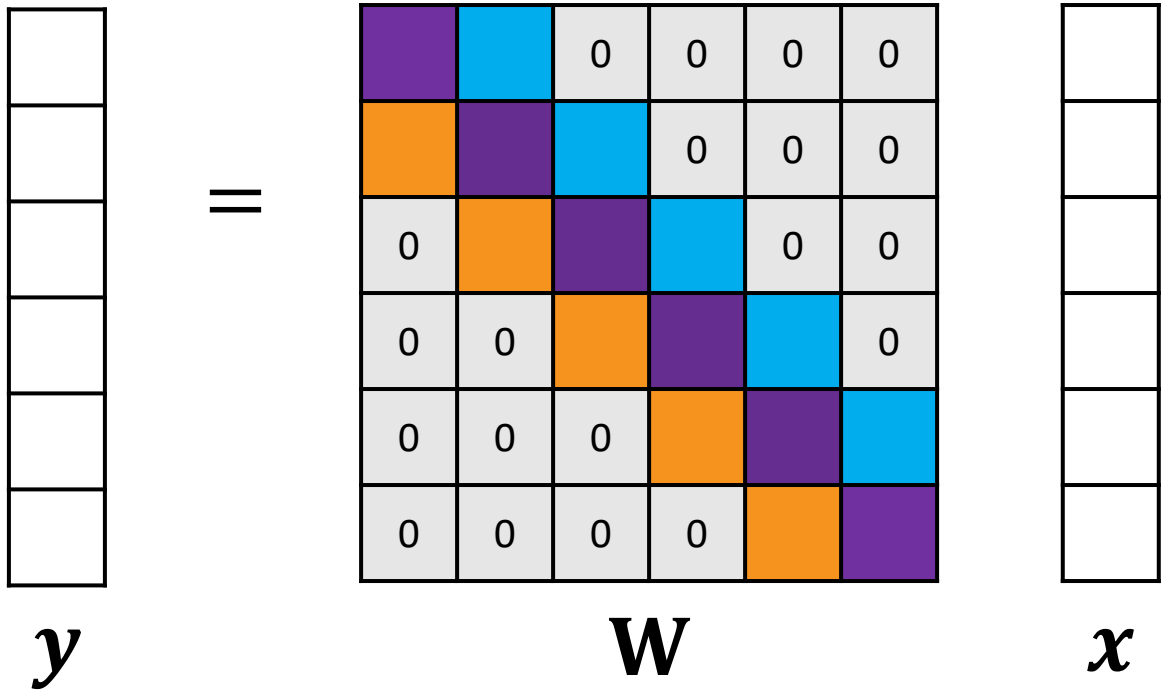




- **local connection**: **W** is sparse
- **weight-sharing**: # params = 3 here in W

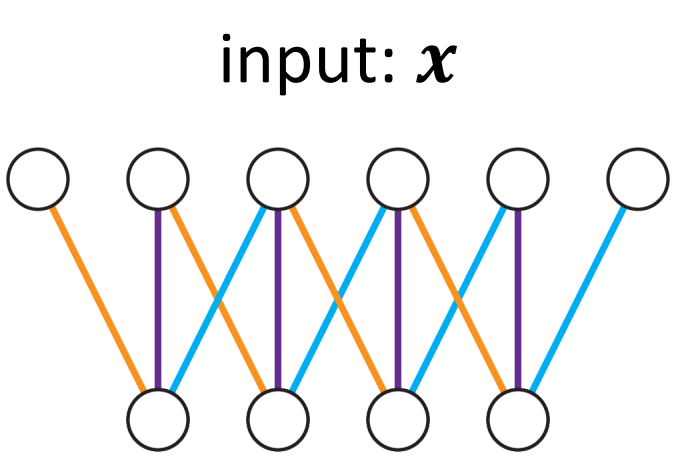
Convolution: linear transform

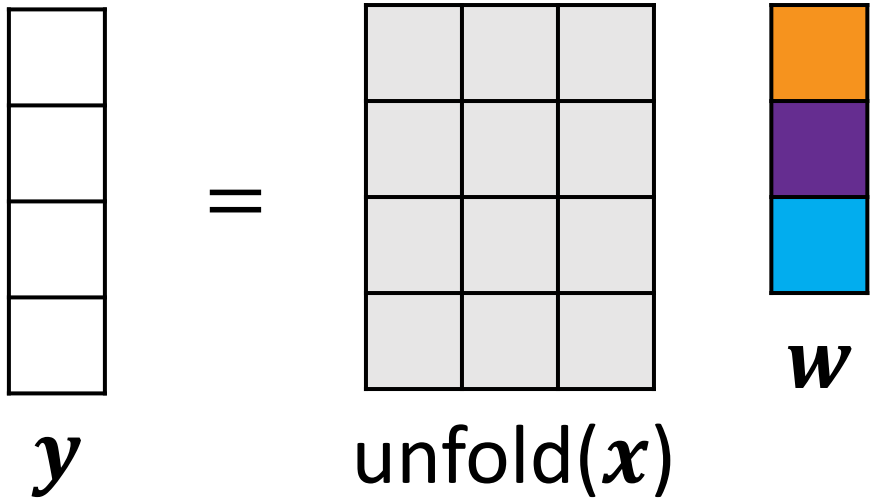




- This is the case with padding
- *Exercise*: what about stride = 2?
- *Exercise*: what about 2-D convolution?

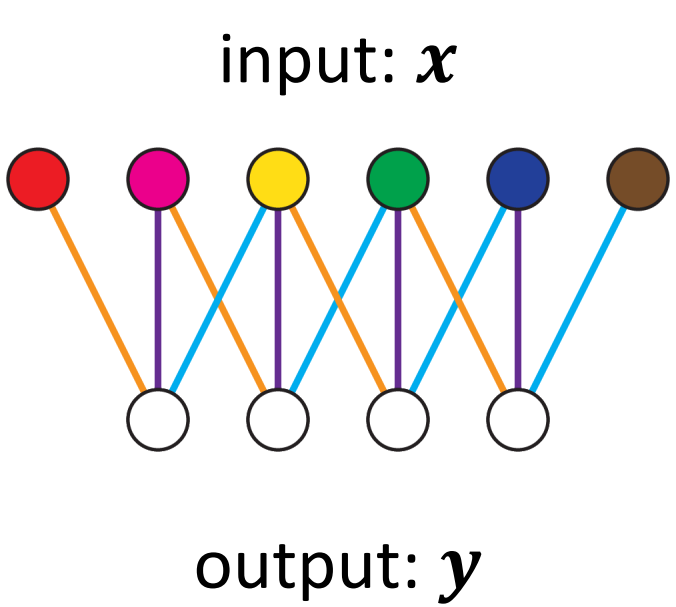
Convolution: unfold / im2col

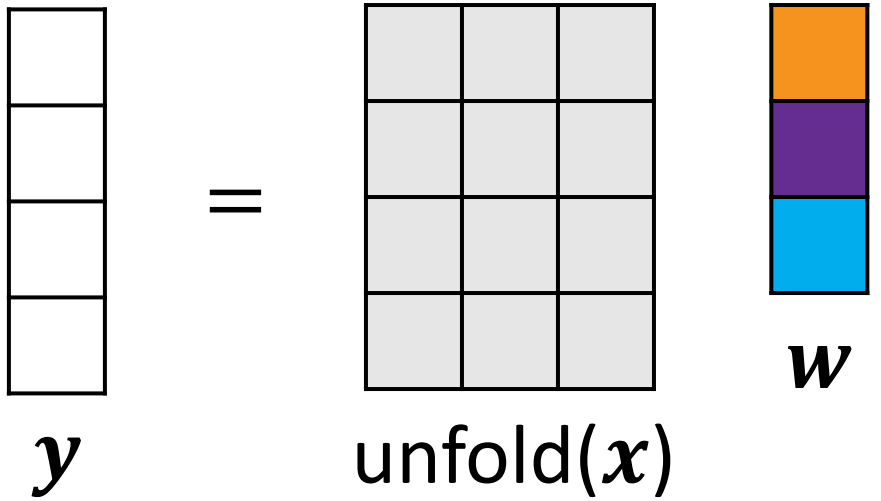




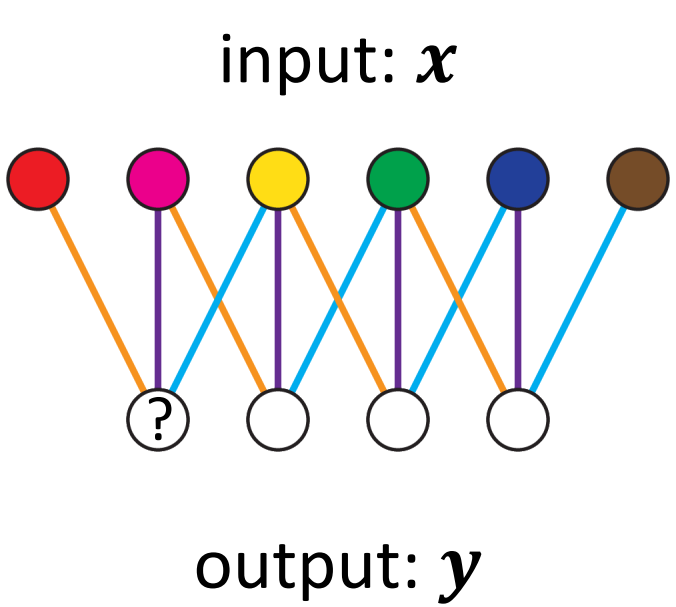
Can we find another linear transform,
such that we don't need sparse matmul?

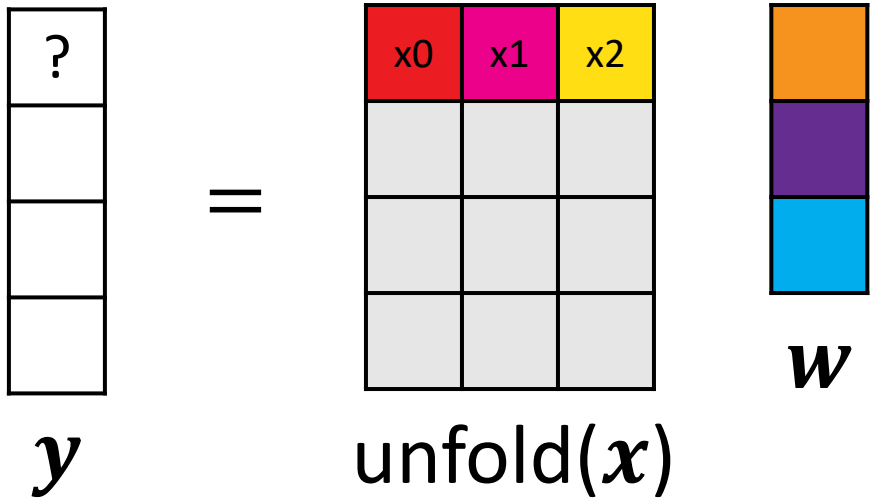
Convolution: unfold / im2col



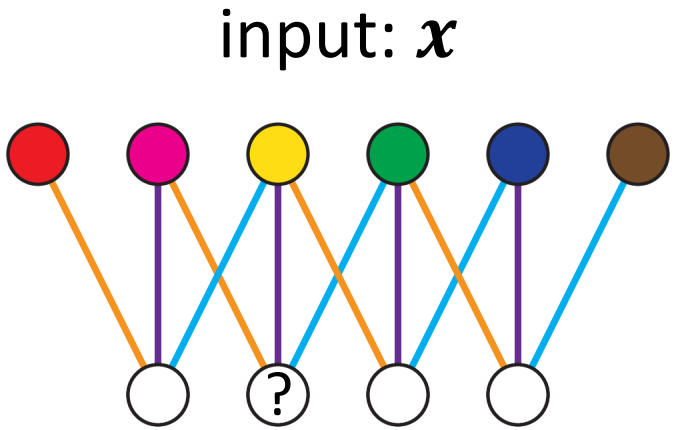


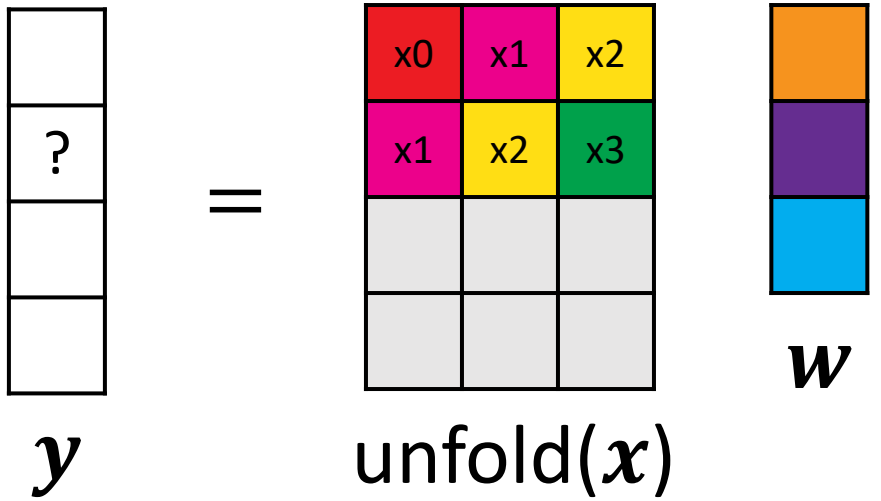
Convolution: unfold / im2col



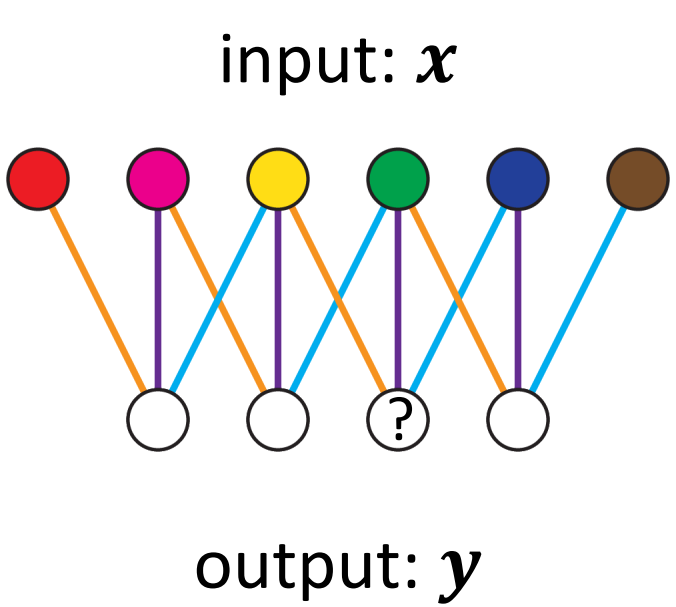


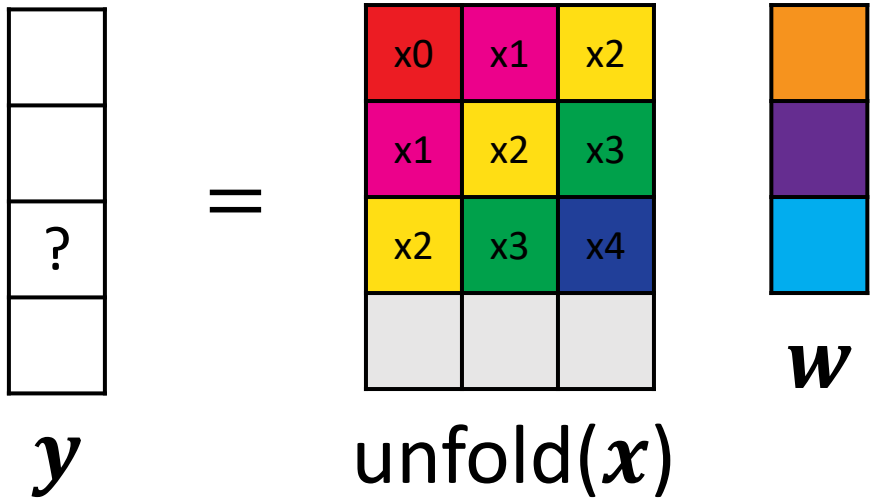
Convolution: unfold / im2col



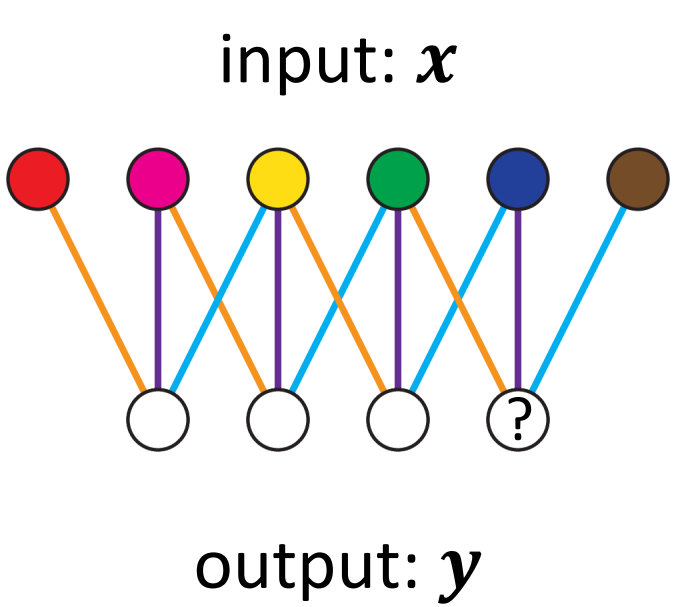


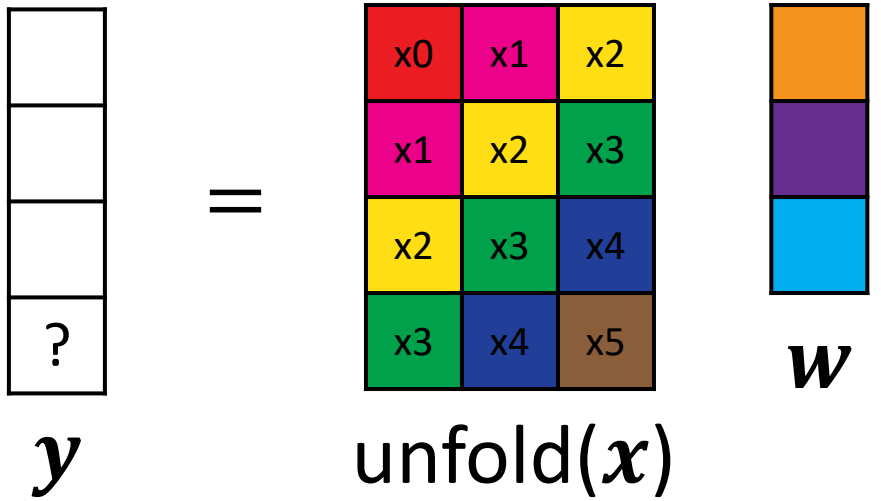
Convolution: unfold / im2col



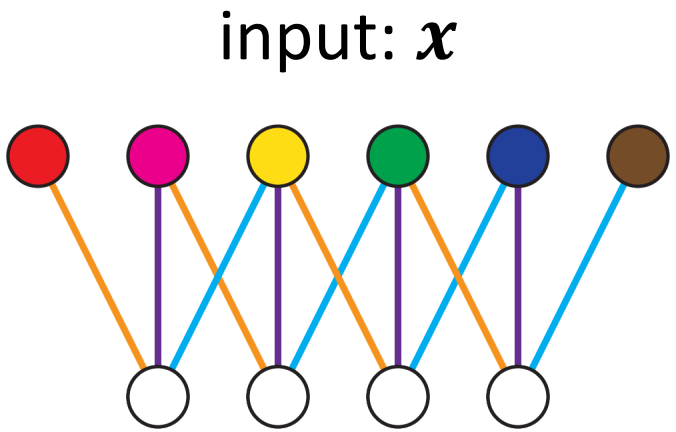


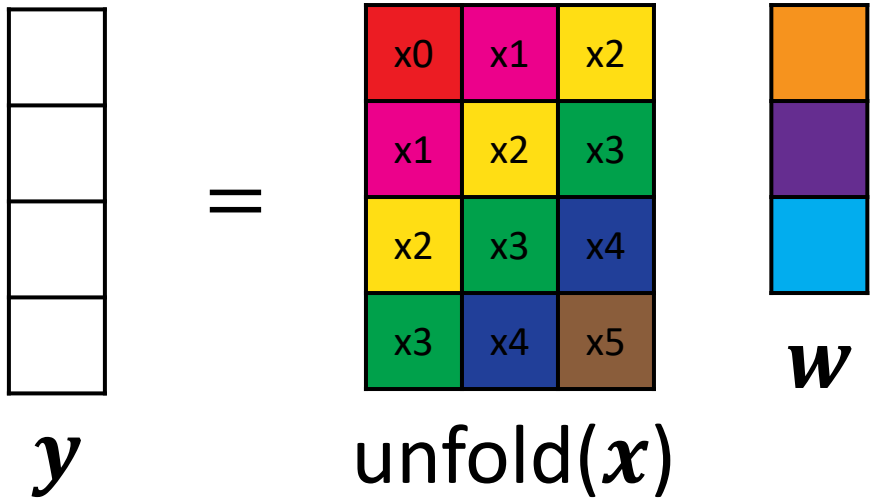
Convolution: unfold / im2col





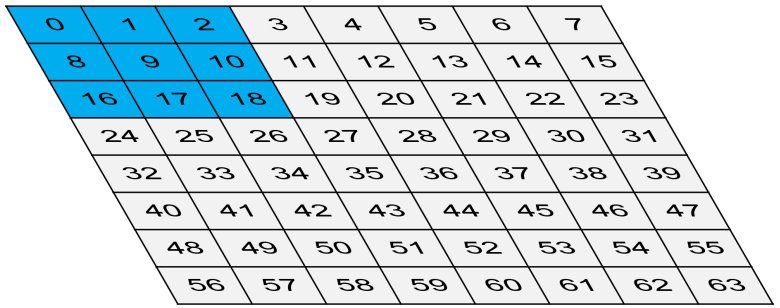
Convolution: unfold / im2col



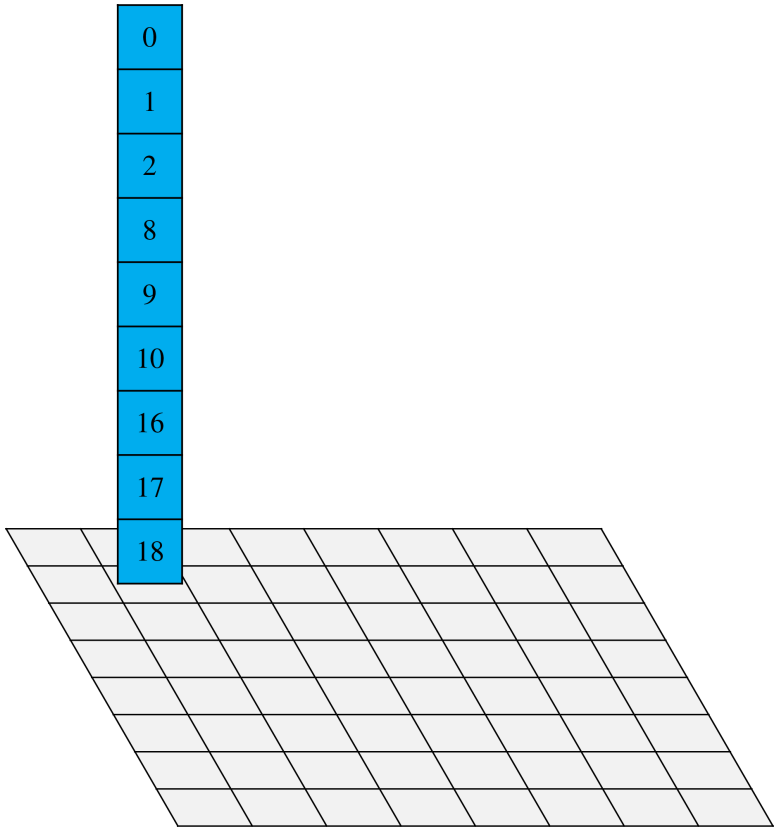


- this transform is called "**unfold**" (PyTorch)
- in 2D, it's also called "**im2col**" (Matlab)
- w is dense; can easily do multiple filters
- "**unfold**" in itself is convolution (*Exercise*: why?)

Convolution: unfold / im2col

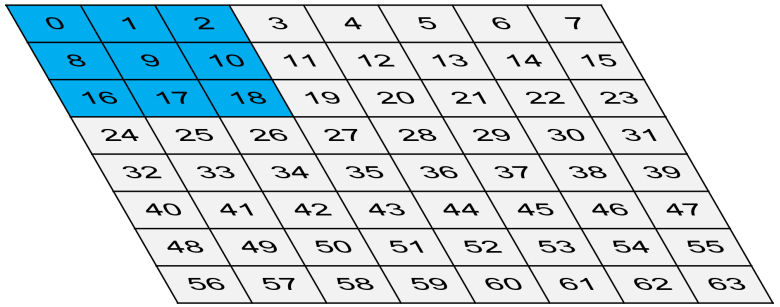


2D im

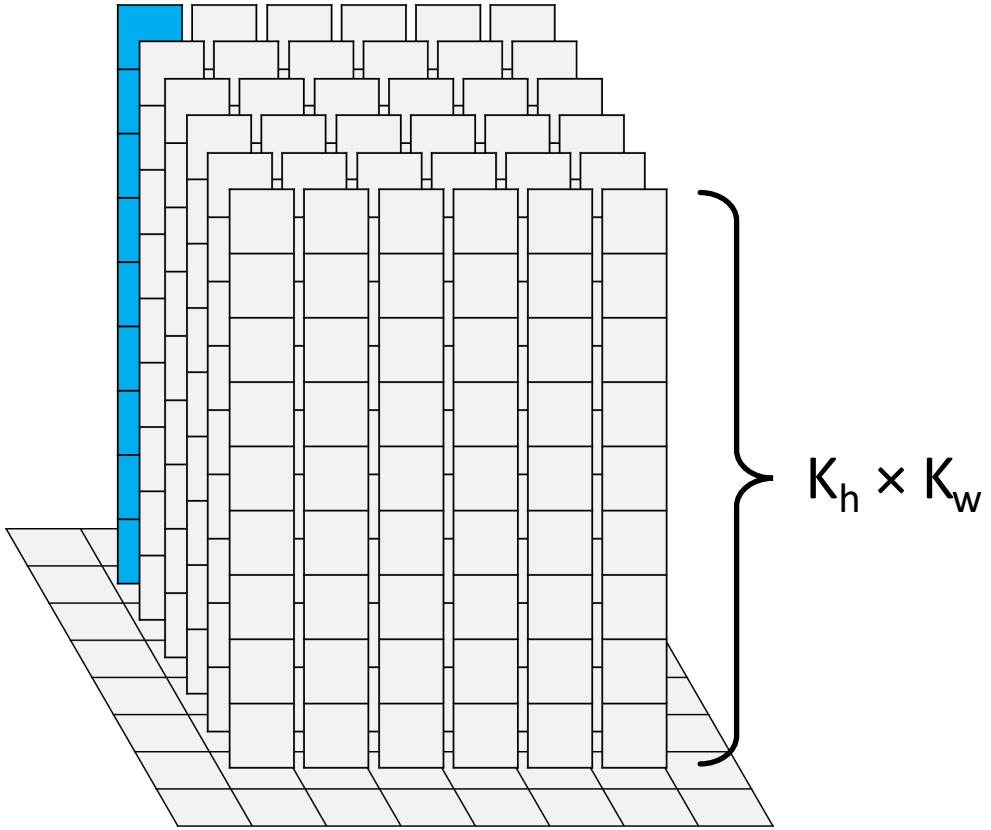


im2col

Convolution: unfold / im2col



2D im

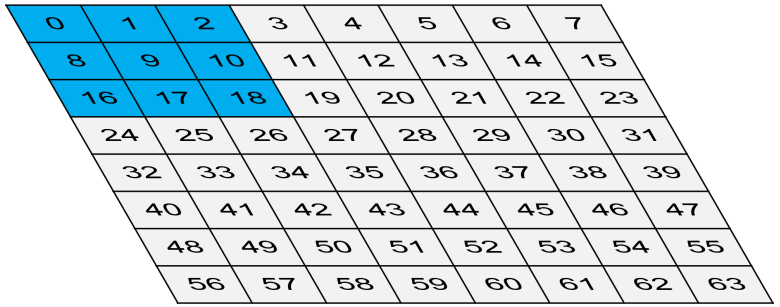


im2col

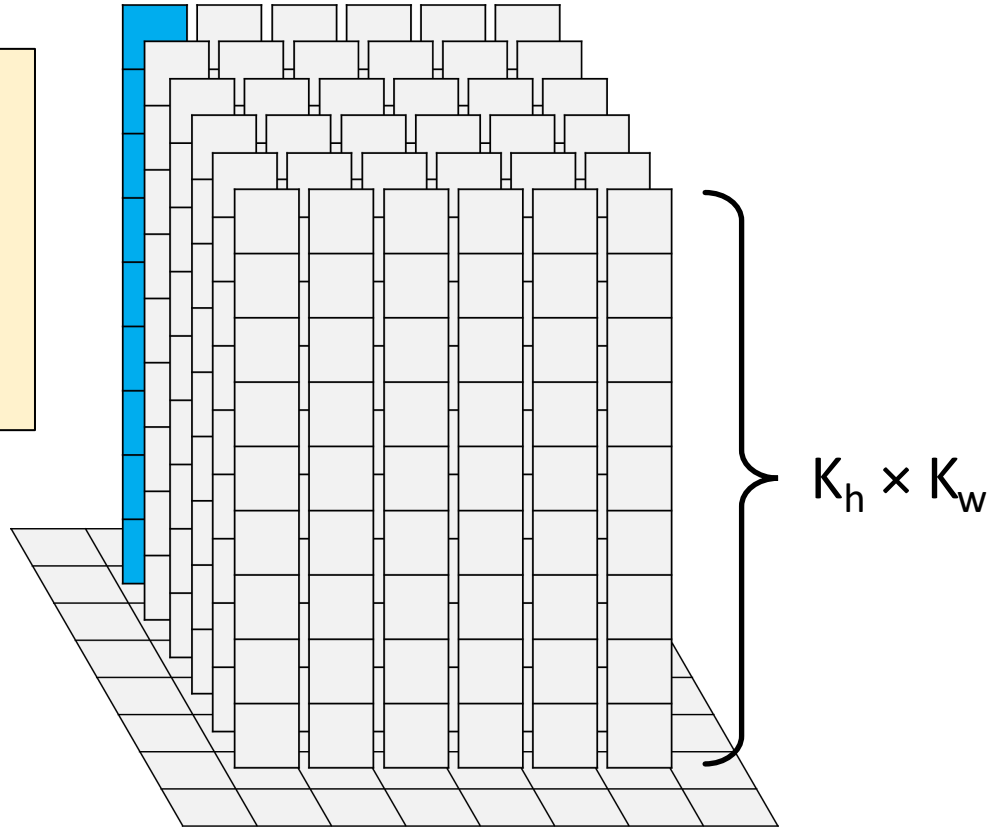
Convolution: unfold / im2col

im2col

- gather local information
- flatten into a “col” ($K_h \times K_w$ channels)
- followed by 1×1 conv



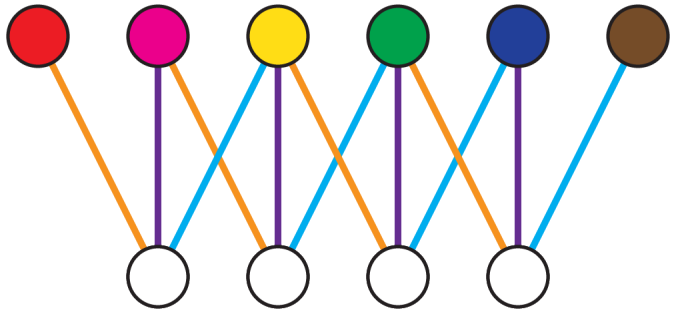
2D im



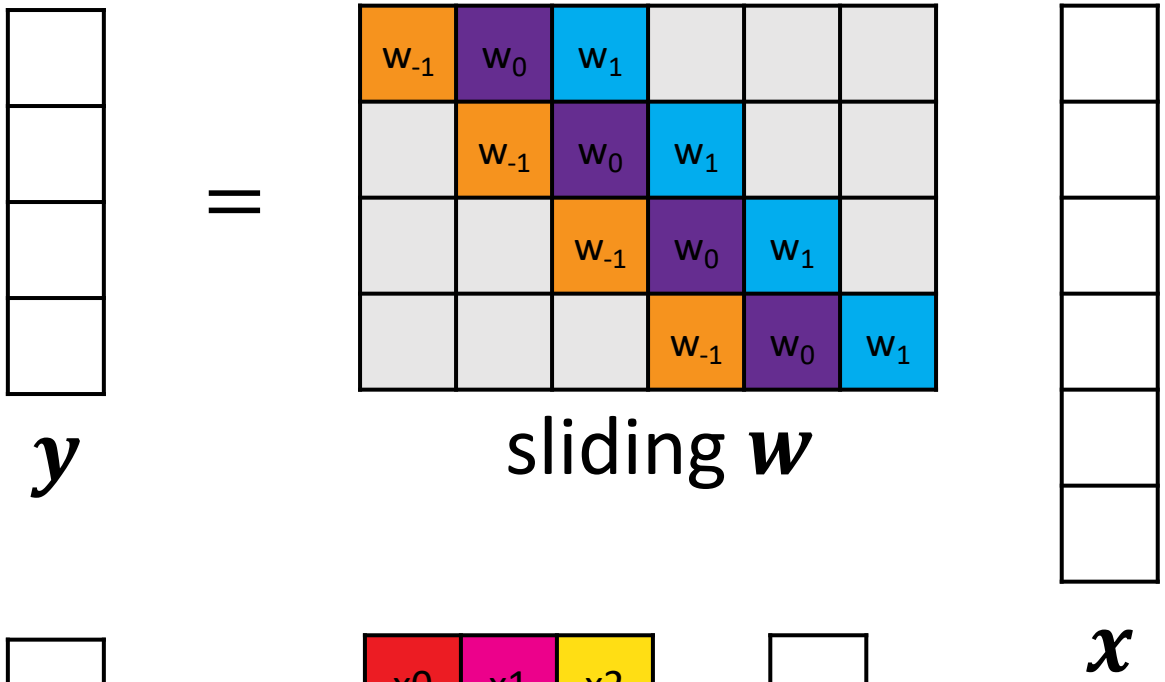
im2col

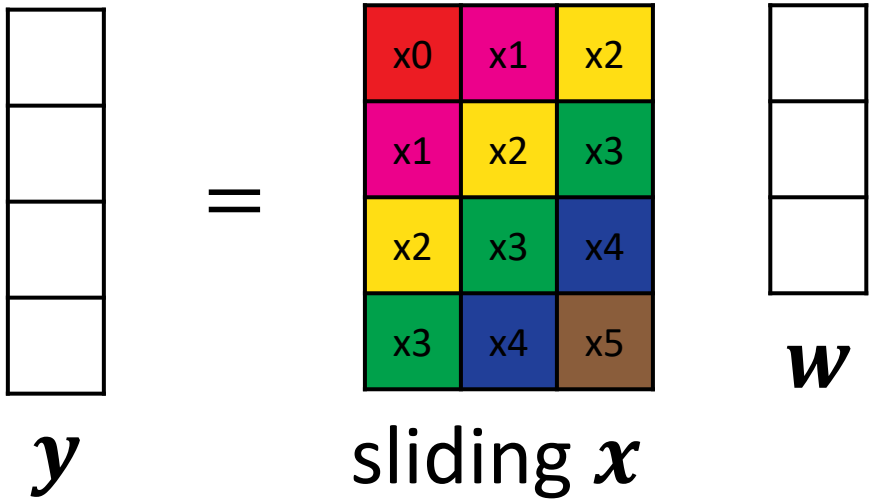
Two views of convolution

input: $\boldsymbol{x}$

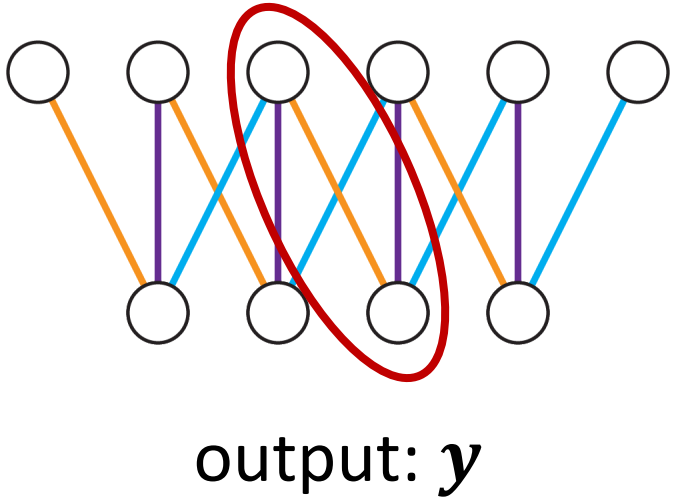


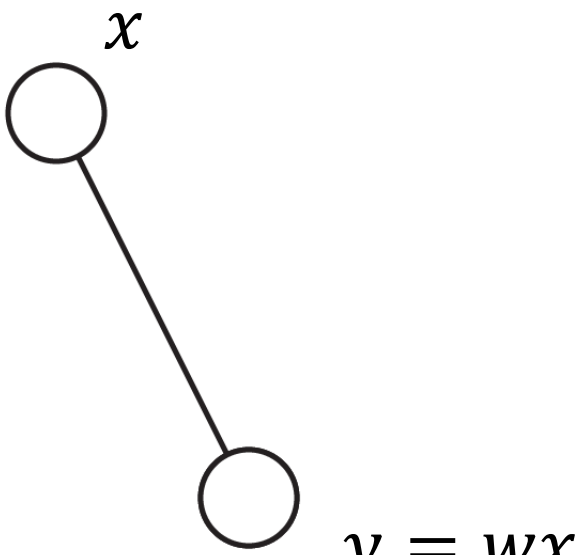
output: $\boldsymbol{y}$





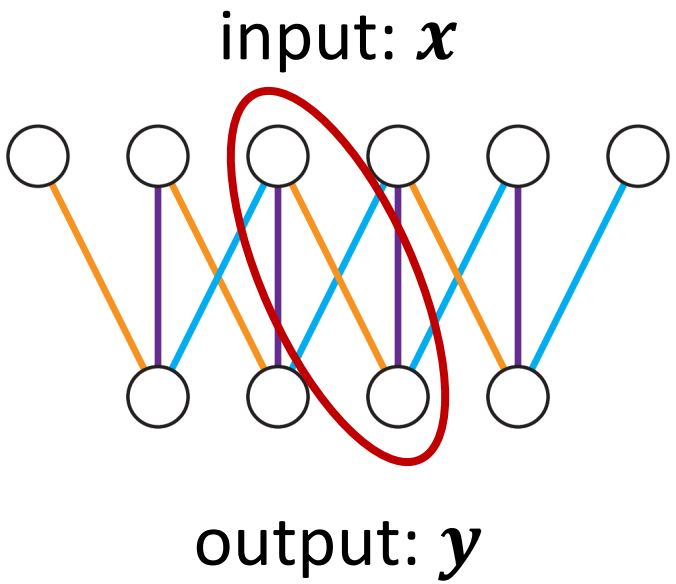
Convolution: BackProp

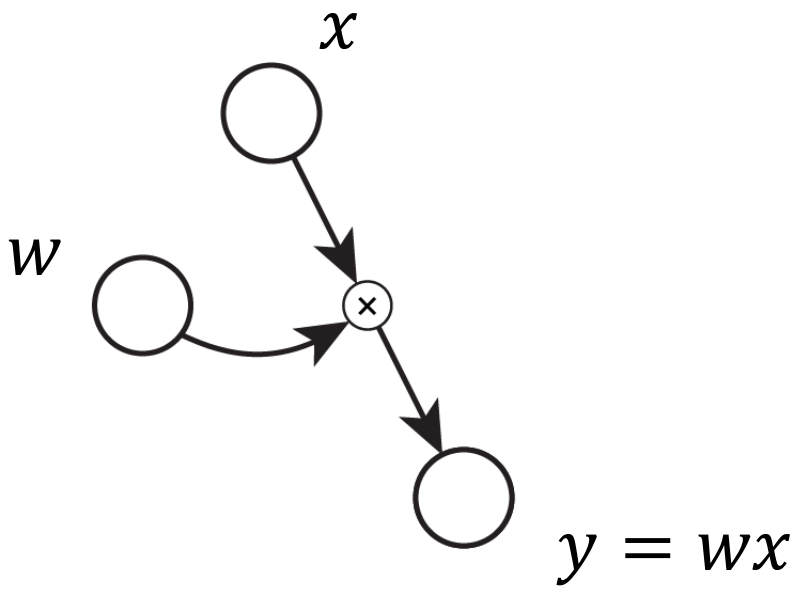




forward

Convolution: BackProp





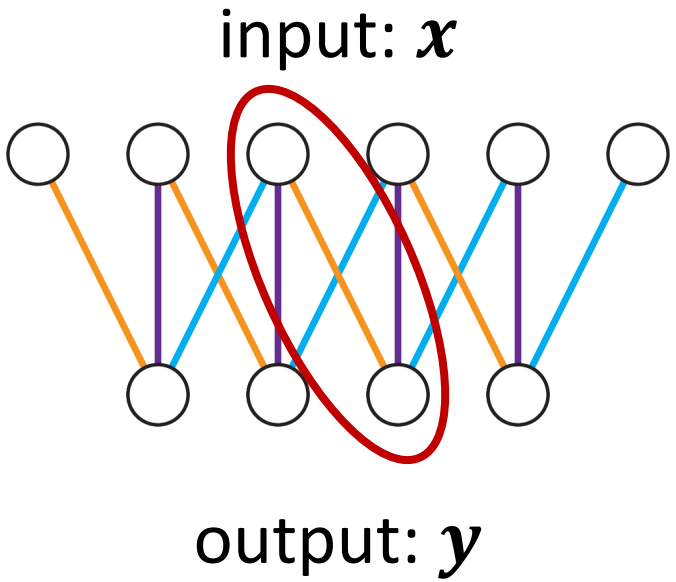
forward

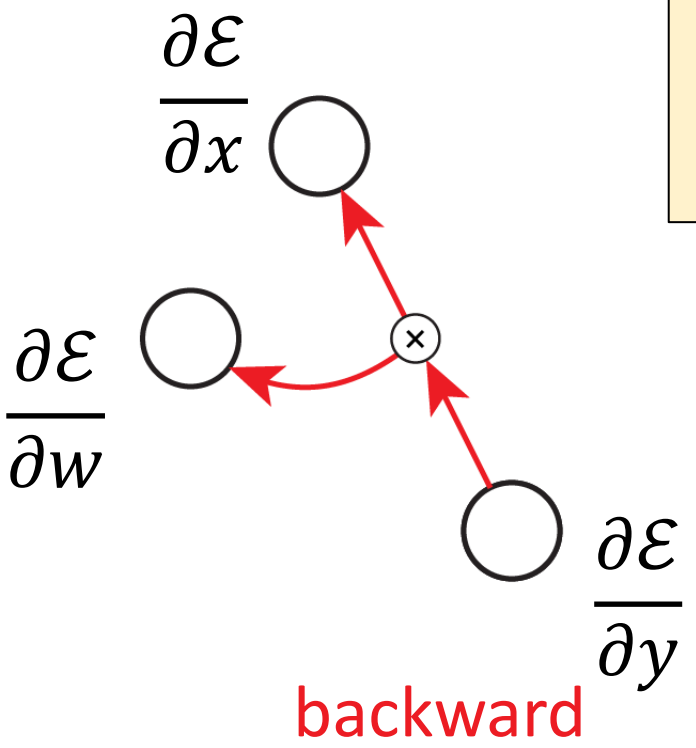
Convolution: BackProp

chain rule (scalars):

$$\frac{\partial \mathcal{E}}{\partial x} = \frac{\partial \mathcal{E}}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \mathcal{E}}{\partial y} w$$

$$\frac{\partial \mathcal{E}}{\partial w} = \frac{\partial \mathcal{E}}{\partial y}\frac{\partial y}{\partial w} = \frac{\partial \mathcal{E}}{\partial y} x$$

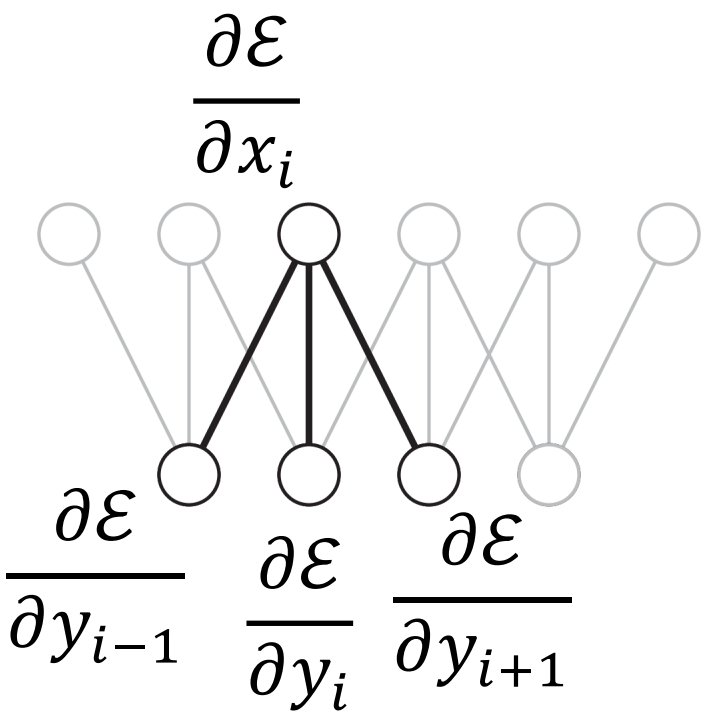


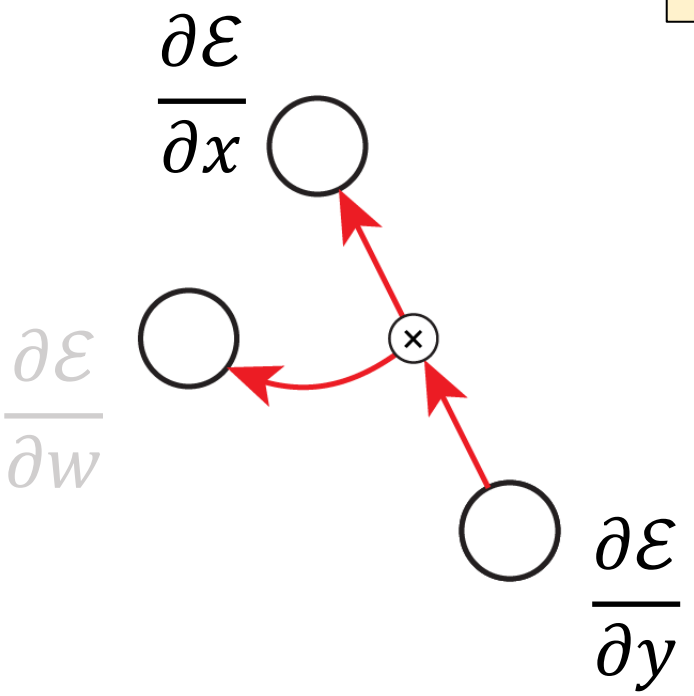


Convolution: BackProp

chain rule (scalars):

$$\frac{\partial \mathcal{E}}{\partial x} = \frac{\partial \mathcal{E}}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \mathcal{E}}{\partial y} w$$



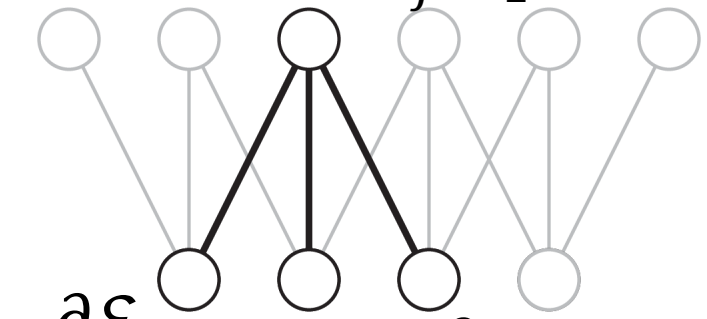


Convolution: BackProp

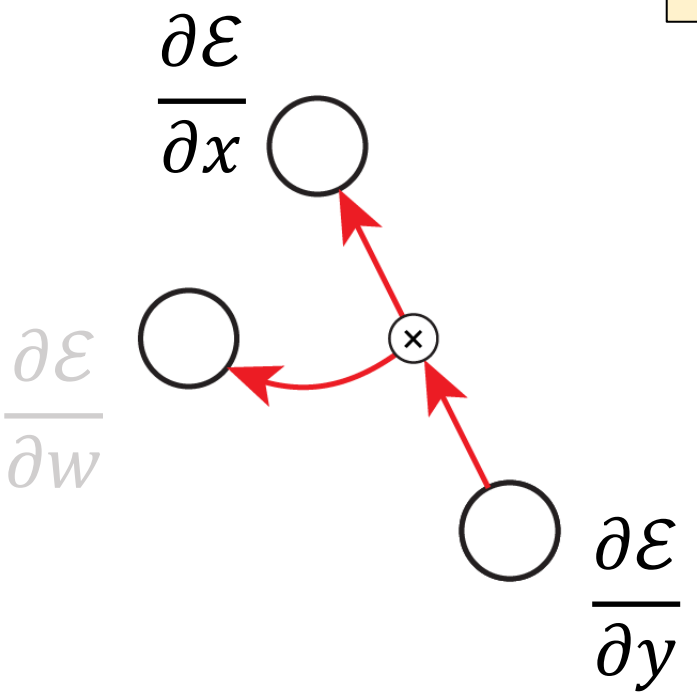
chain rule (scalars):

$$\frac{\partial \mathcal{E}}{\partial x} = \frac{\partial \mathcal{E}}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \mathcal{E}}{\partial y} w$$

$$\frac{\partial \mathcal{E}}{\partial x_i} = \sum_{j=-1}^{1} \frac{\partial \mathcal{E}}{\partial y_{i+j}} w_{-j}$$



$$\frac{\partial \mathcal{E}}{\partial y_{i-1}} \quad \frac{\partial \mathcal{E}}{\partial y_i} \quad \frac{\partial \mathcal{E}}{\partial y_{i+1}}$$

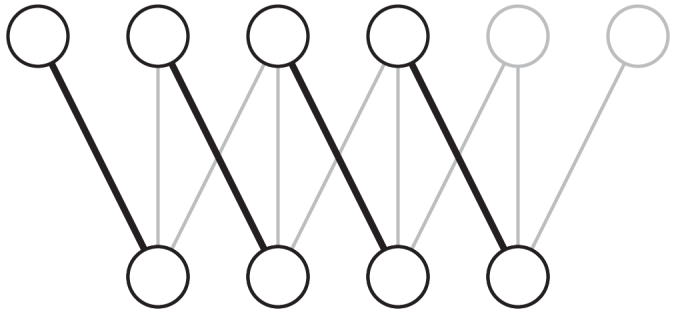


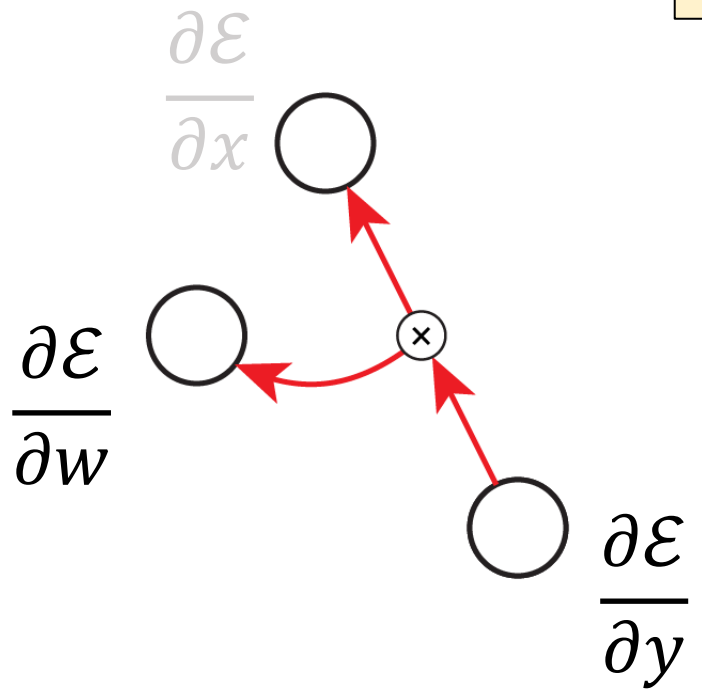
backprop wrt x is conv

Convolution: BackProp

chain rule (scalars):

$$\frac{\partial \mathcal{E}}{\partial w} = \frac{\partial \mathcal{E}}{\partial y}\frac{\partial y}{\partial w} = \frac{\partial \mathcal{E}}{\partial y}x$$



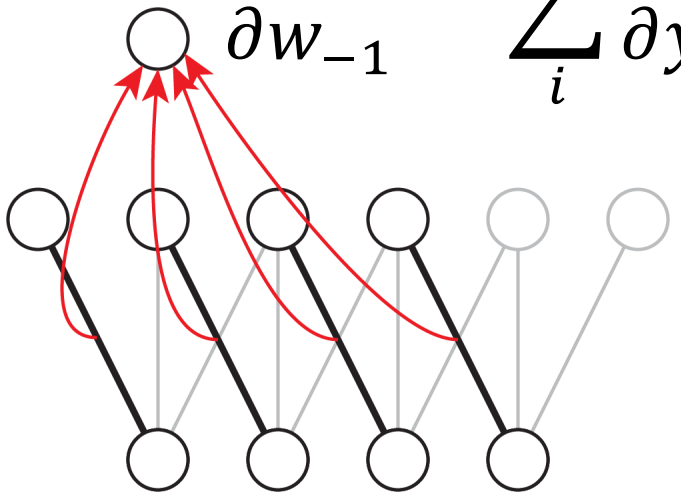


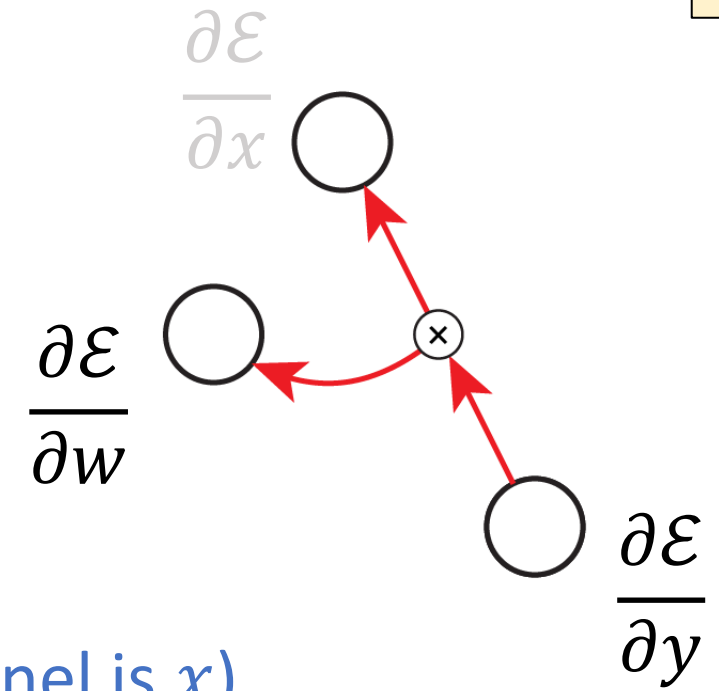
Convolution: BackProp

chain rule (scalars):

$$\frac{\partial \mathcal{E}}{\partial w} = \frac{\partial \mathcal{E}}{\partial y}\frac{\partial y}{\partial w} = \frac{\partial \mathcal{E}}{\partial y}x$$

$$\frac{\partial \mathcal{E}}{\partial w_{-1}} = \sum_i \frac{\partial \mathcal{E}}{\partial y_i} x_{i-1}$$





backprop wrt w is also conv (kernel is x)

Exercise: derive the backprop in matrix forms

Takeaways about Convolution

- sliding window
- local connection + weight-sharing
- linear transform
- translation-equivariant

Convolutional Neural Networks

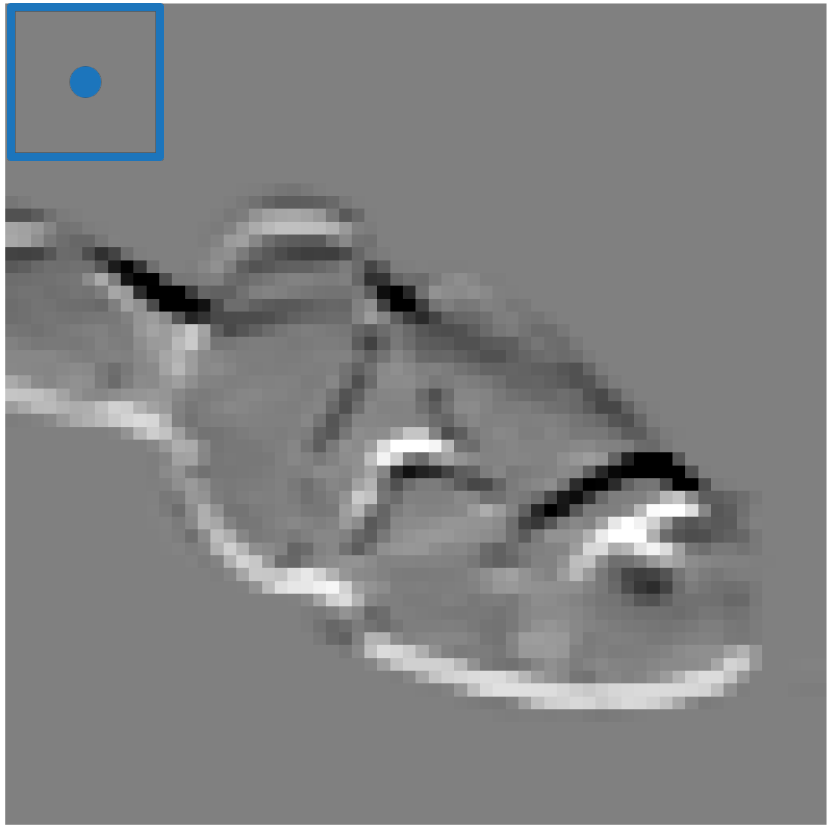
Convolutional Neural Network (ConvNet, CNN)

- Core idea of deep learning:
  Composing basic operations into complex functions

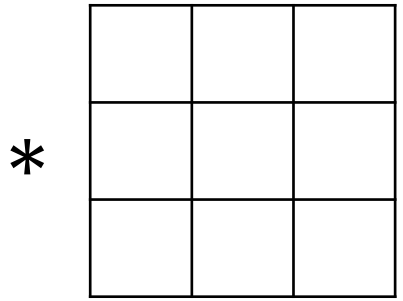
- Common operations of ConvNet
  - convolution
  - activation function
  - pooling
  - fully-connected layer
  - softmax

Composing basic operations

output of prev. layer

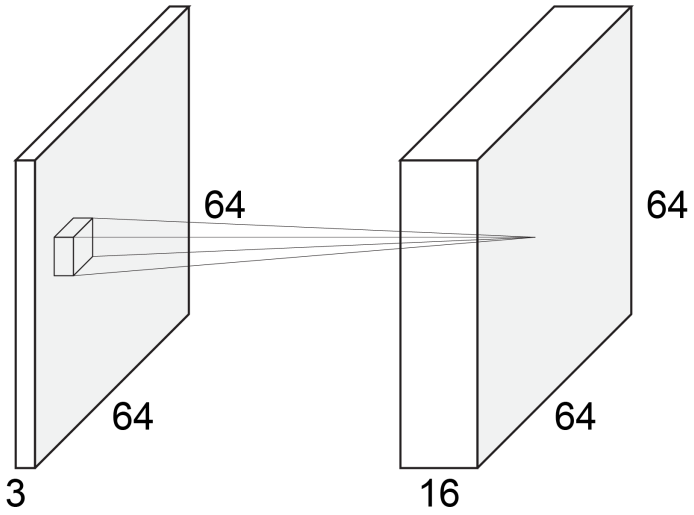


filter



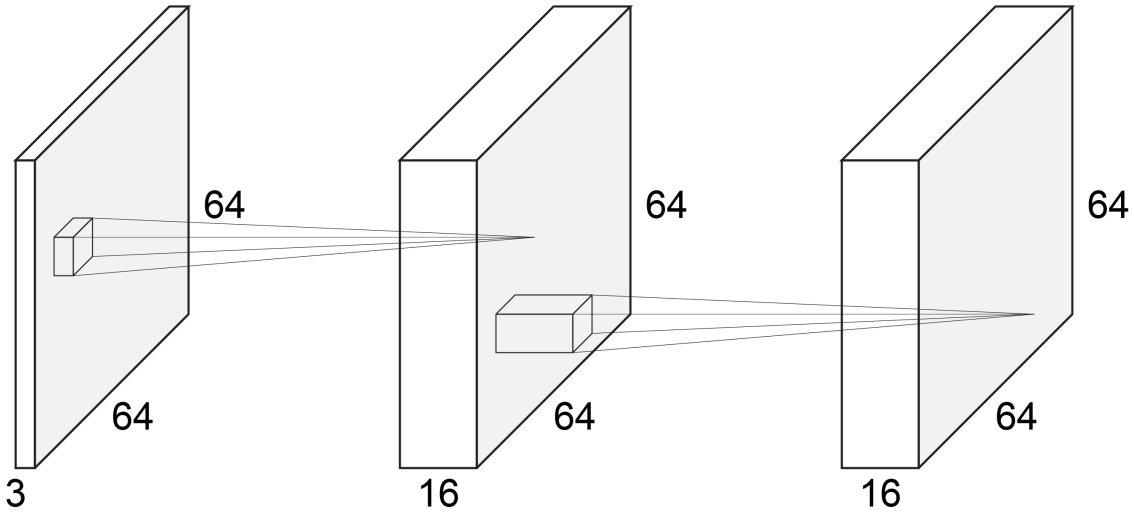
output of this layer

Composing basic operations



conv
3×3, 16

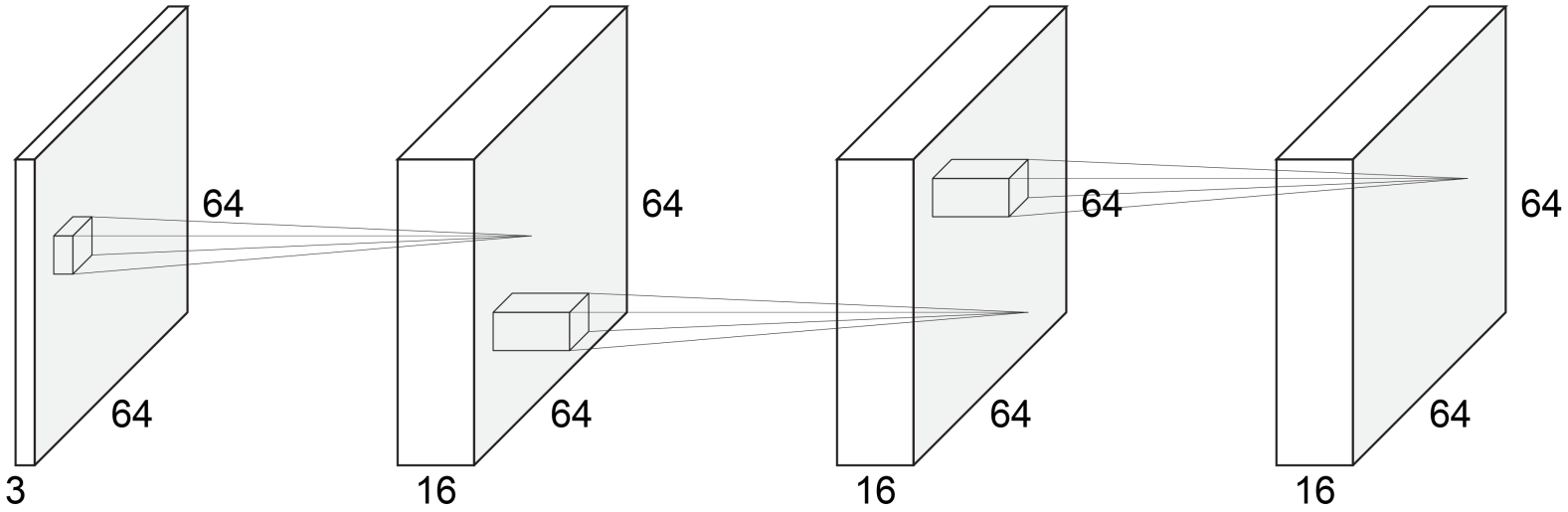
Composing basic operations



conv
3×3, 16

conv
3×3, 16

Composing basic operations

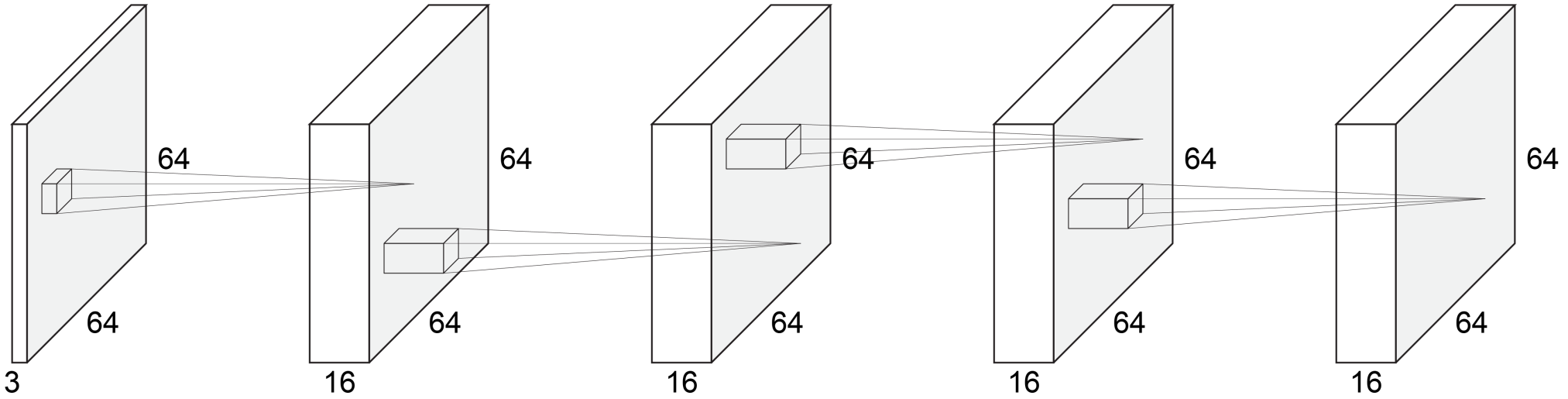


conv
3×3, 16

conv
3×3, 16

conv
3×3, 16

Composing basic operations



conv
3×3, 16

conv
3×3, 16

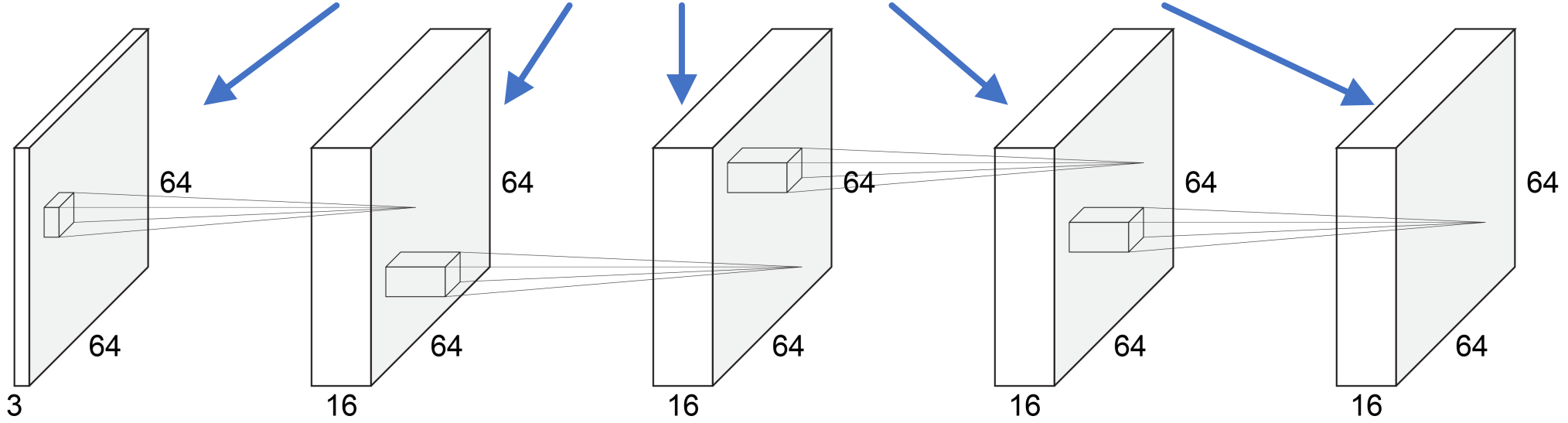
conv
3×3, 16

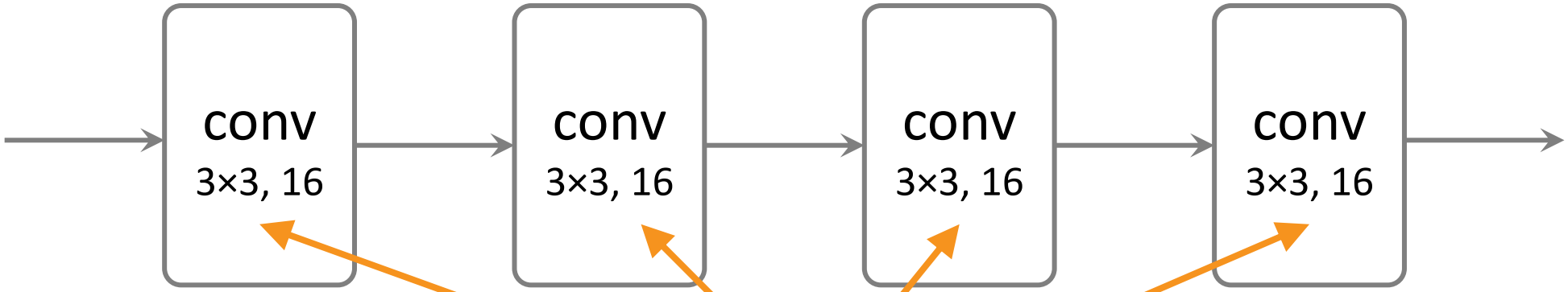
conv
3×3, 16

Composing basic operations

two ways of showing neural nets

these are activations (features, embeddings, tensors …)

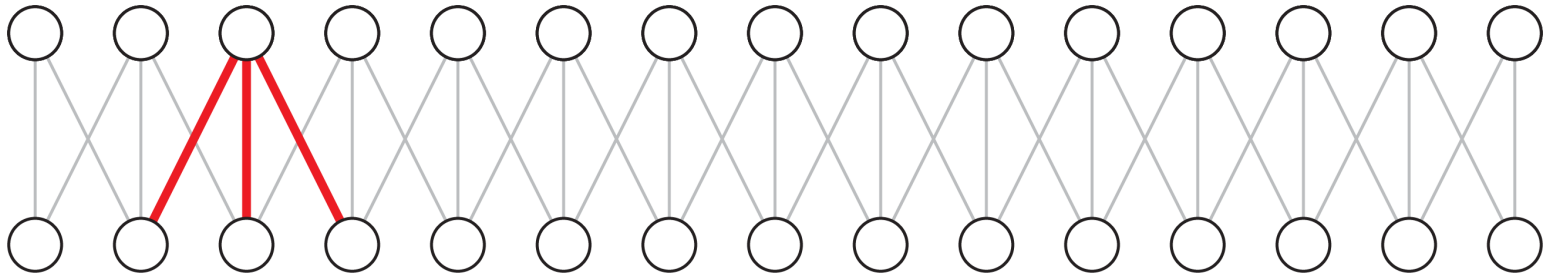




these are operations (functions, transforms, layers …)

Receptive Field

kernel = 3

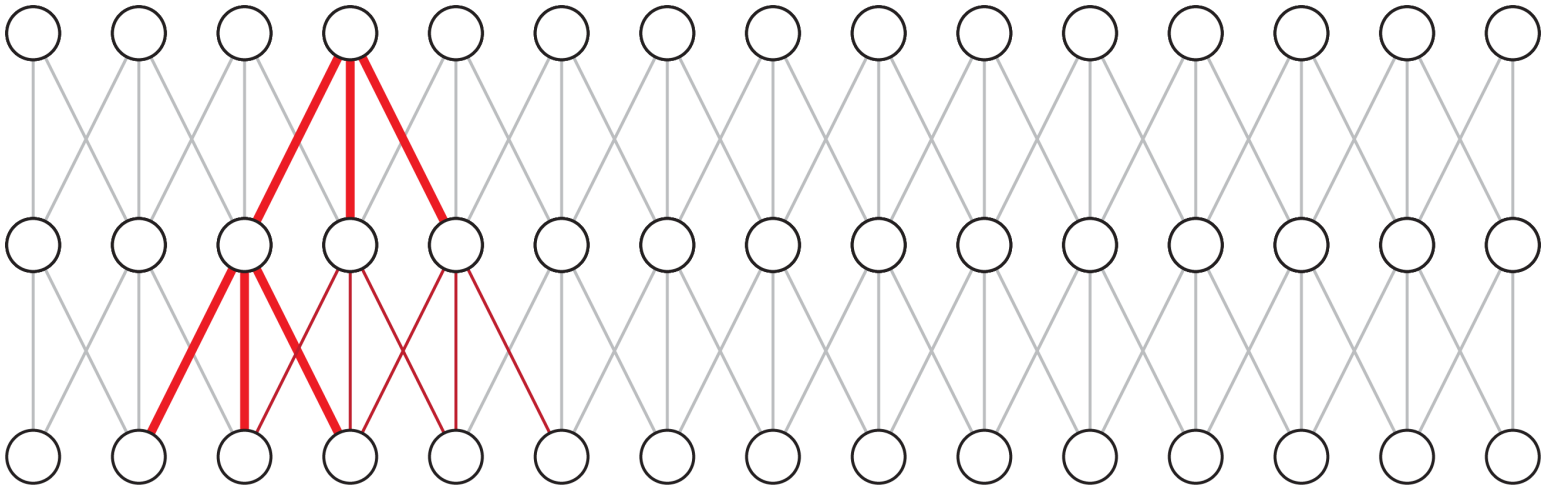


receptive field = 3

Receptive Field

kernel = 3

kernel = 3



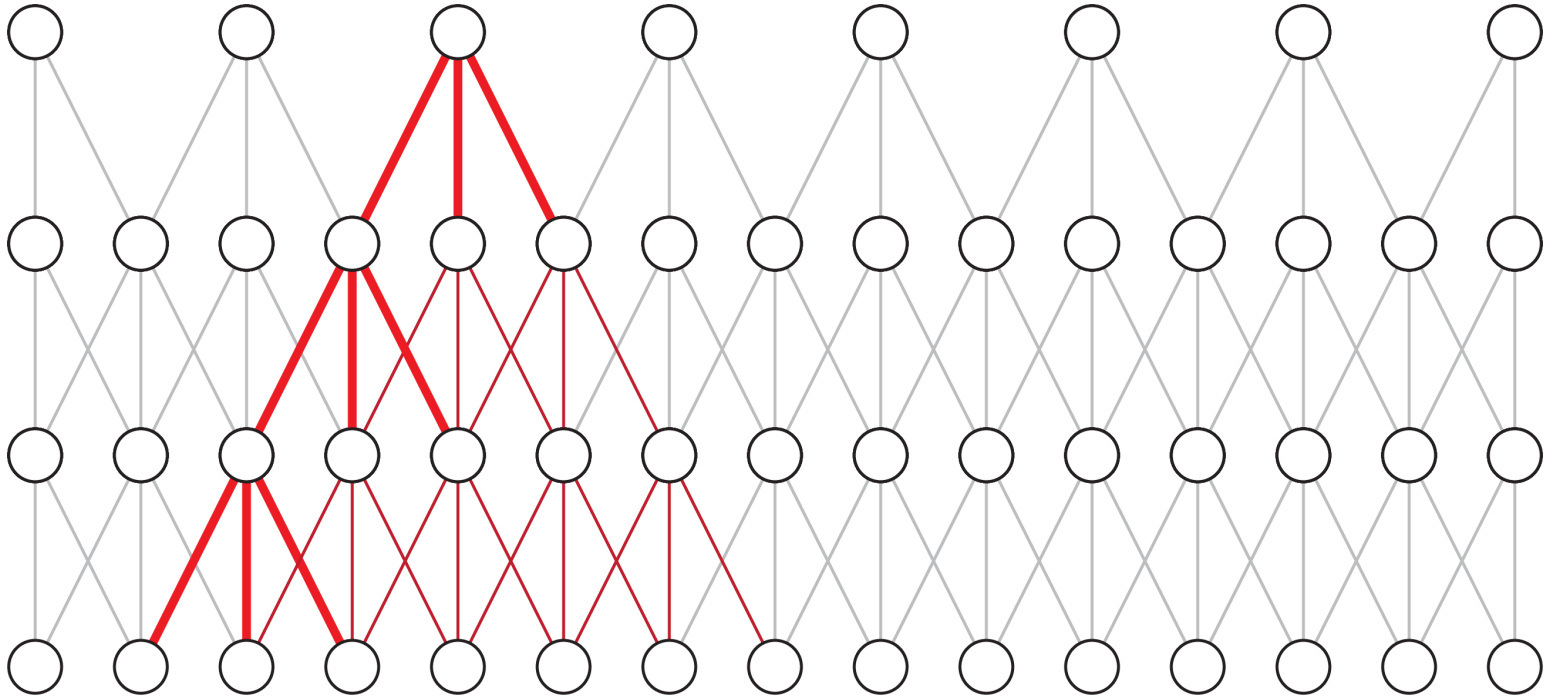
receptive field = 5

Receptive Field

kernel = 3, str = 2

kernel = 3

kernel = 3



receptive field = 7

Receptive Field

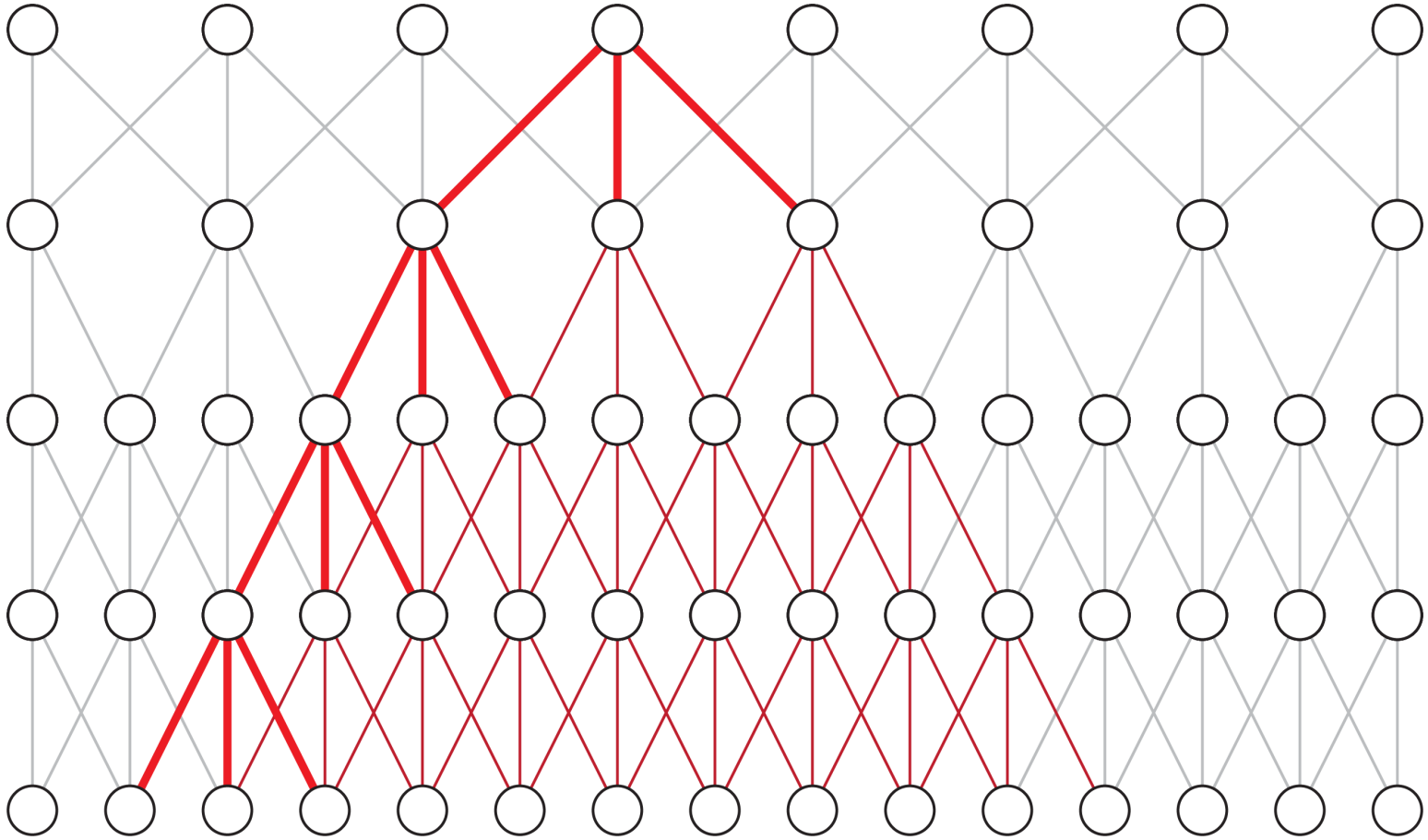
Composing many conv layers can create a large receptive field.

kernel = 3

kernel = 3, str = 2

kernel = 3

kernel = 3



receptive field = 11

Convolutional Neural Network (ConvNet, CNN)

- Core idea of deep learning:
  Composing basic operations into complex functions

- Common operations of ConvNet
  - convolution
  - activation function
  - pooling
  - fully-connected layer
  - softmax

Activation function

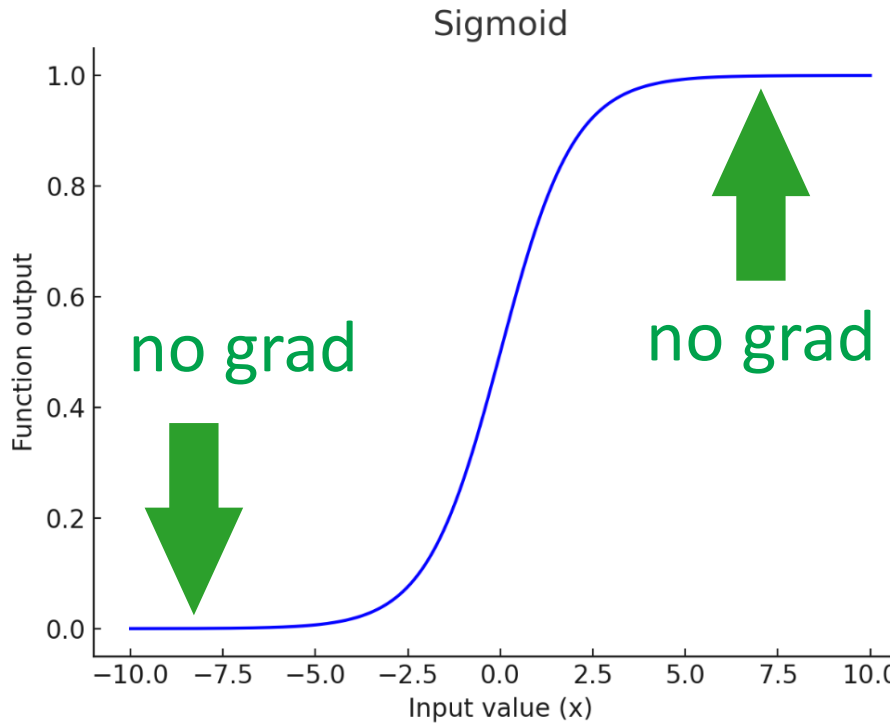
- Activation functions introduce non-linearity

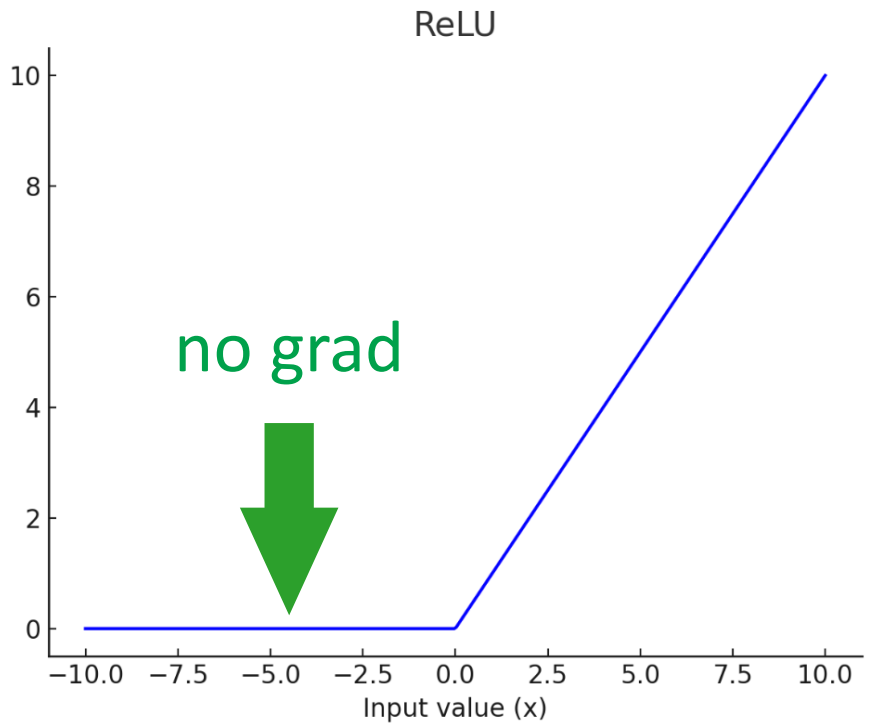
x

ReLU(x) = max(x, 0)

Activation function

- Rectified Linear Unit (ReLU) helps gradient propagation





Pooling

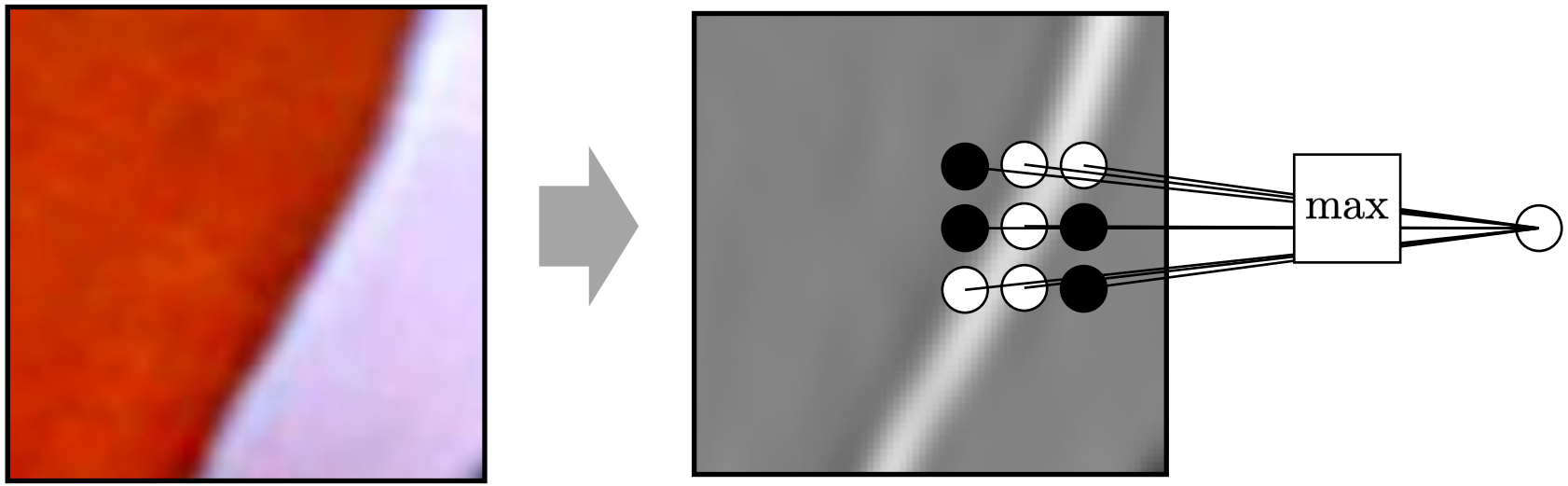
- Pooling is similar to convolution:
  - sliding window + aggregation
  - often stride > 1
  - local invariance w.r.t. translation

max pooling:

$$y[i] = \max_{j \in \mathcal{N}(i)} x[j]$$

average pooling:

$$y[i] = \frac{1}{|\mathcal{N}|} \operatorname{sum}_{j \in \mathcal{N}(i)} x[j]$$



"There is an edge!"

large activation

Pooling

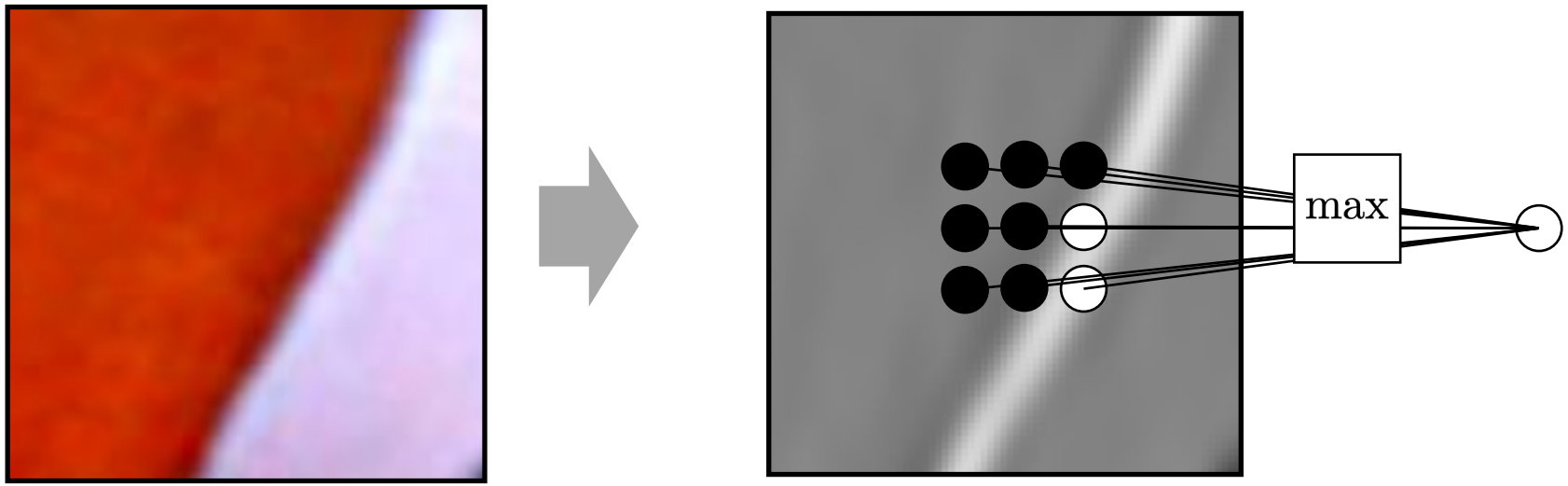
- Pooling is similar to convolution:
  - sliding window + aggregation
  - often stride > 1
  - local invariance w.r.t. translation

max pooling:

$$y[i] = \max_{j \in \mathcal{N}(i)} x[j]$$

average pooling:

$$y[i] = \frac{1}{|\mathcal{N}|} \operatorname{sum}_{j \in \mathcal{N}(i)} x[j]$$



"There is an edge!"

large activation

Pooling

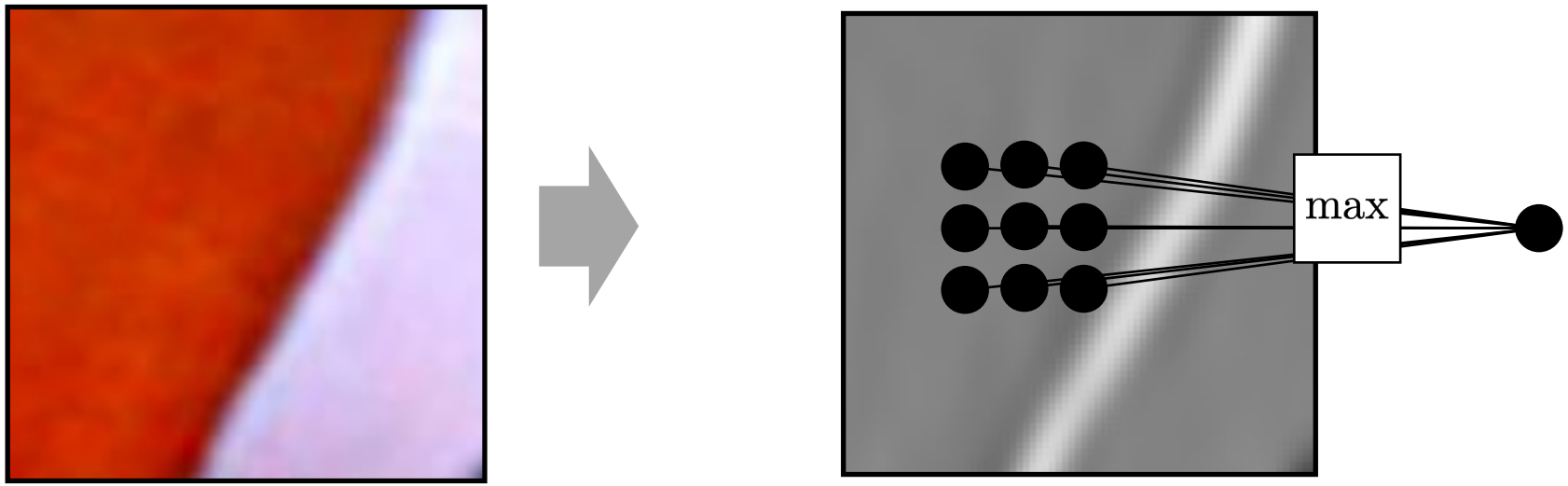
- Pooling is similar to convolution:
  - sliding window + aggregation
  - often stride > 1
  - local invariance w.r.t. translation

max pooling:

$$y[i] = \max_{j \in \mathcal{N}(i)} x[j]$$

average pooling:

$$y[i] = \frac{1}{|\mathcal{N}|} \operatorname{sum}_{j \in \mathcal{N}(i)} x[j]$$



"No edge!"

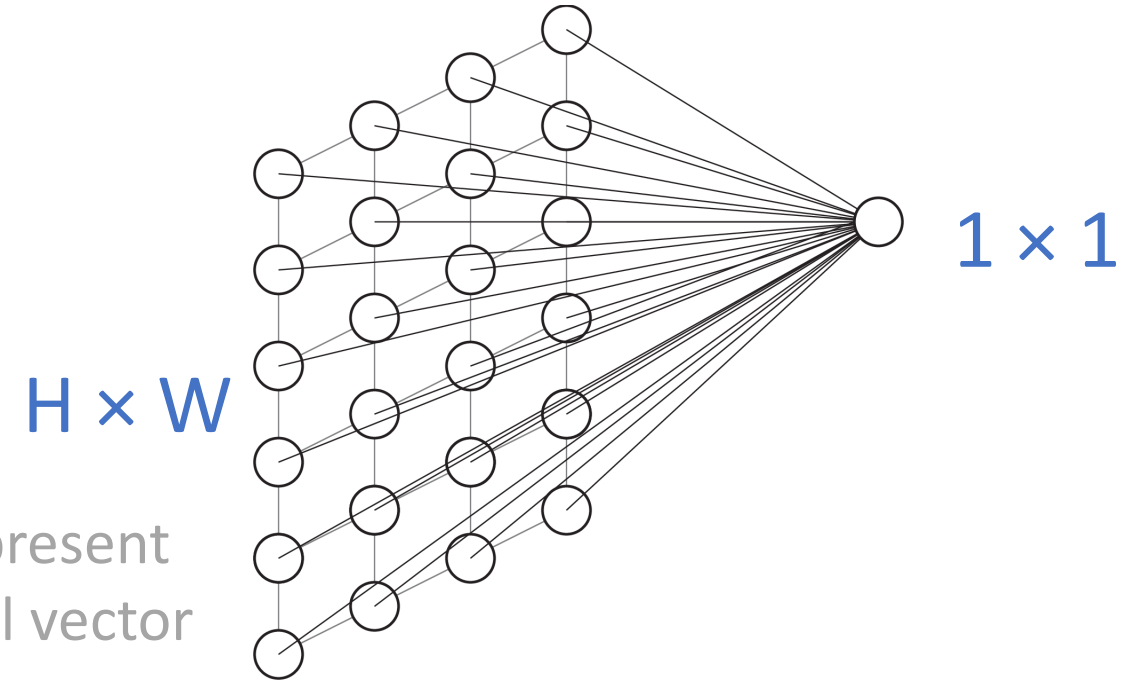
no activation

Fully-connected layer in ConvNet

- every output is connected to every input

or alternatively:

- convolution with kernel_size = H × W (w/o padding)
- output map size = 1 × 1
- useful for “fully conv net”

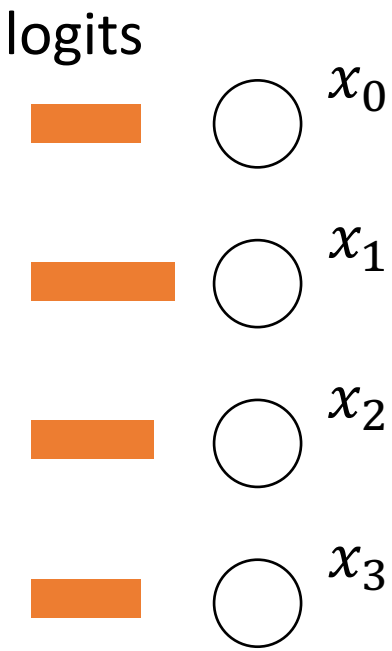


each node represent
a multi-channel vector

Softmax

$$\text{softmax}(\boldsymbol{x})_i = e^{x_i} / \sum e^{x_j}$$

- softmax = exponential activation & normalization

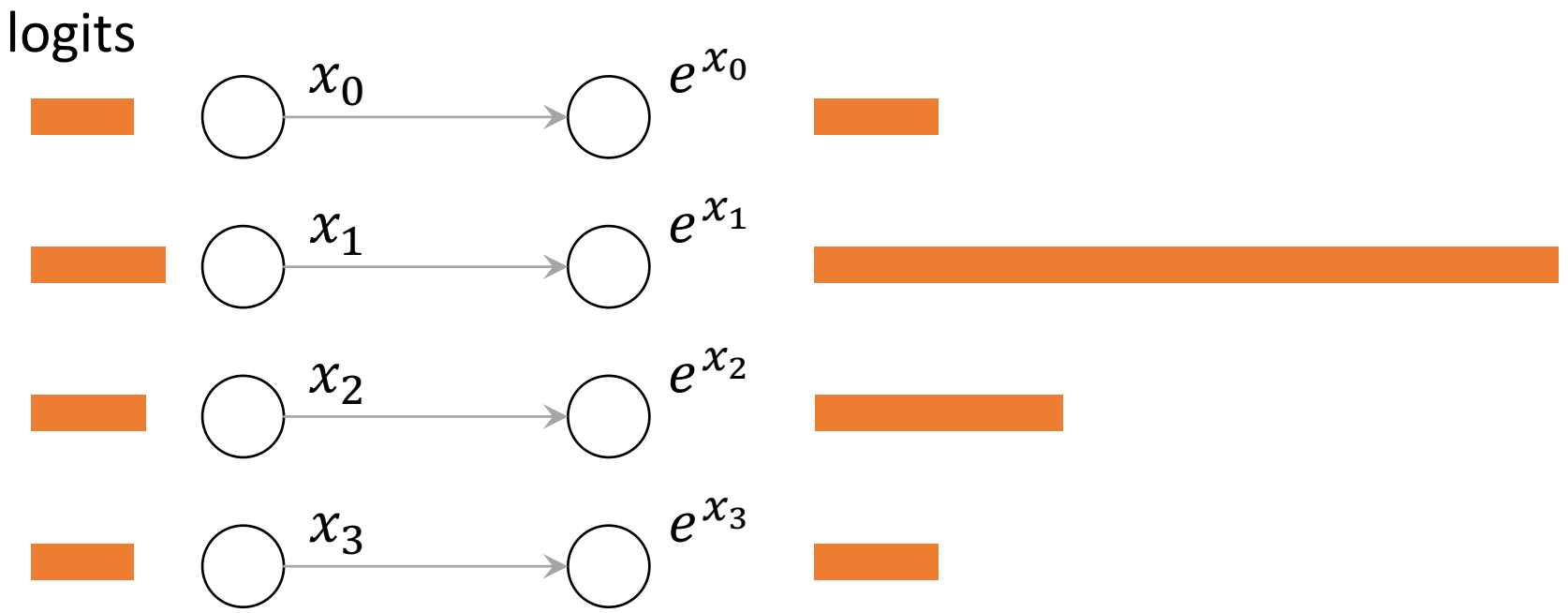


this example: 4 channels for 4-class classification

Softmax

$$\text{softmax}(\boldsymbol{x})_i = e^{x_i} / \sum e^{x_j}$$

- softmax = exponential activation & normalization



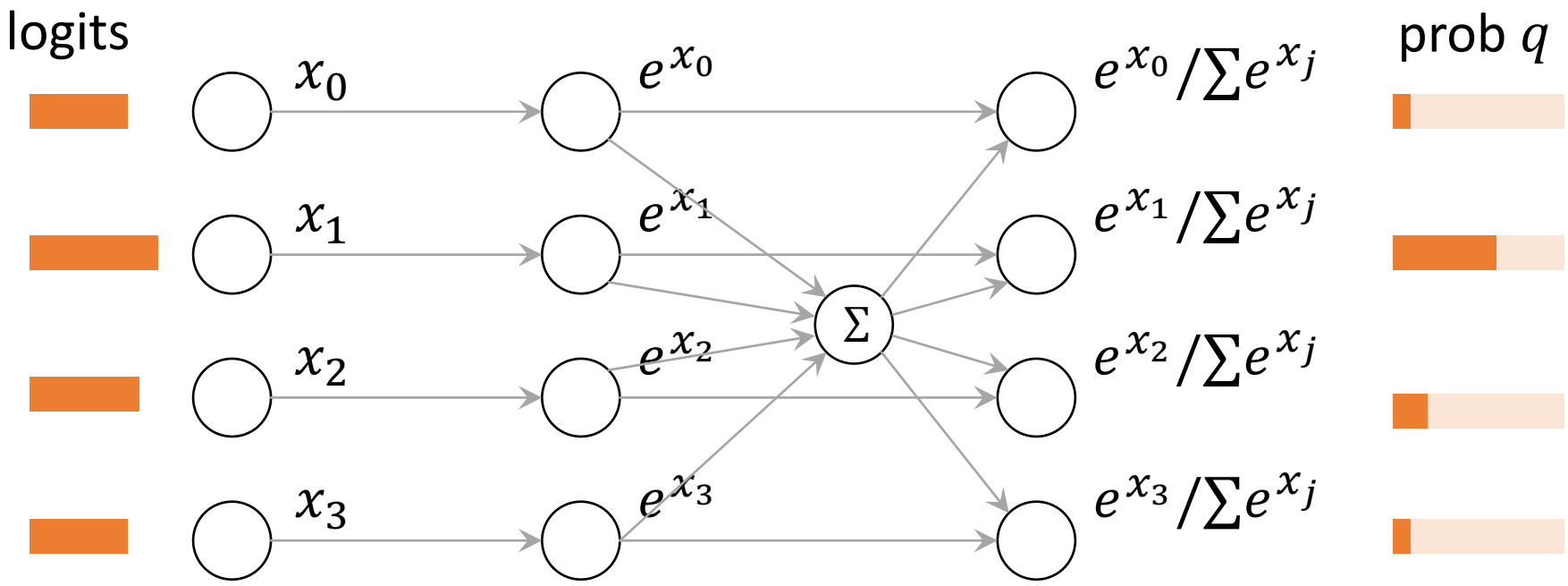
exponential:

- bigger activation gets even bigger
- suppresses non-maximum activation

Softmax

$$\text{softmax}(\boldsymbol{x})_i = e^{x_i} / \sum e^{x_j}$$

- softmax = exponential activation & normalization



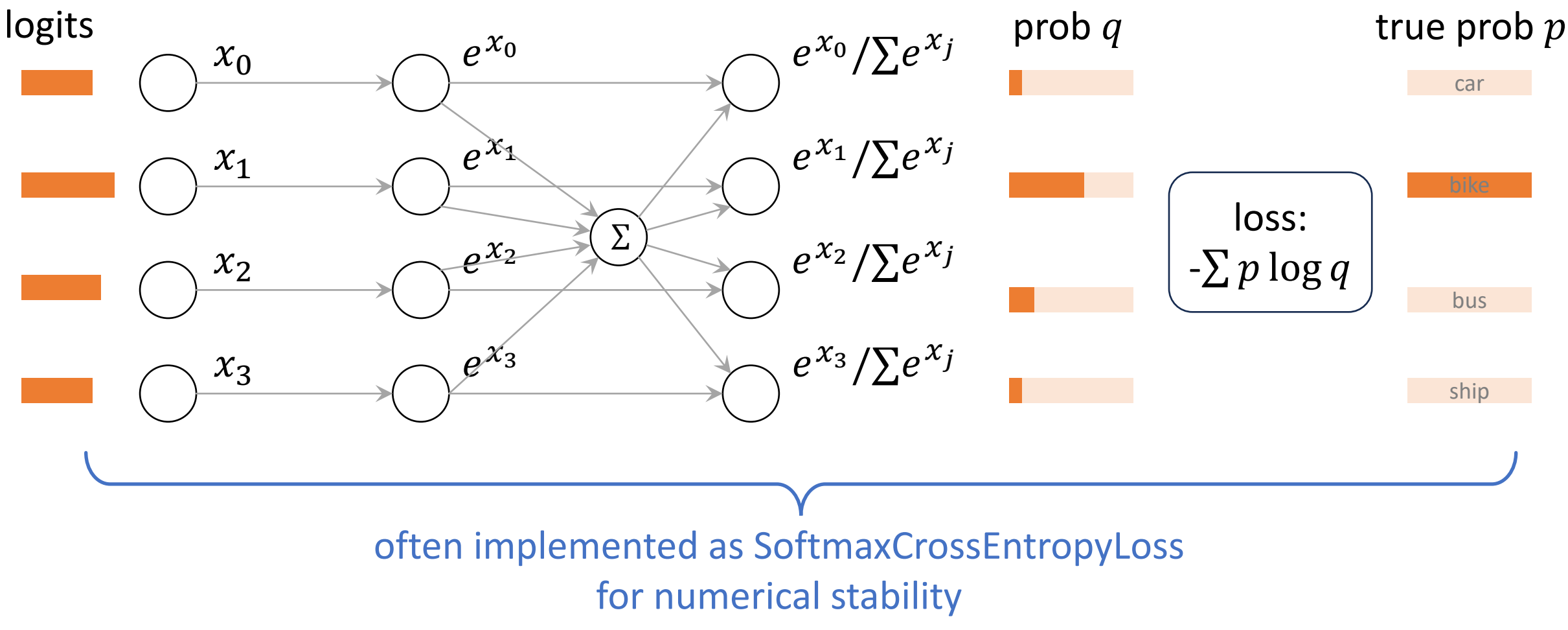
normalization:

- turn into a probability distribution

Softmax

$$\text{softmax}(\boldsymbol{x})_i = e^{x_i} / \sum e^{x_j}$$

- softmax = exponential activation & normalization



Convolutional Neural Network (ConvNet, CNN)

- Core idea of deep learning:
  Composing basic operations into complex functions

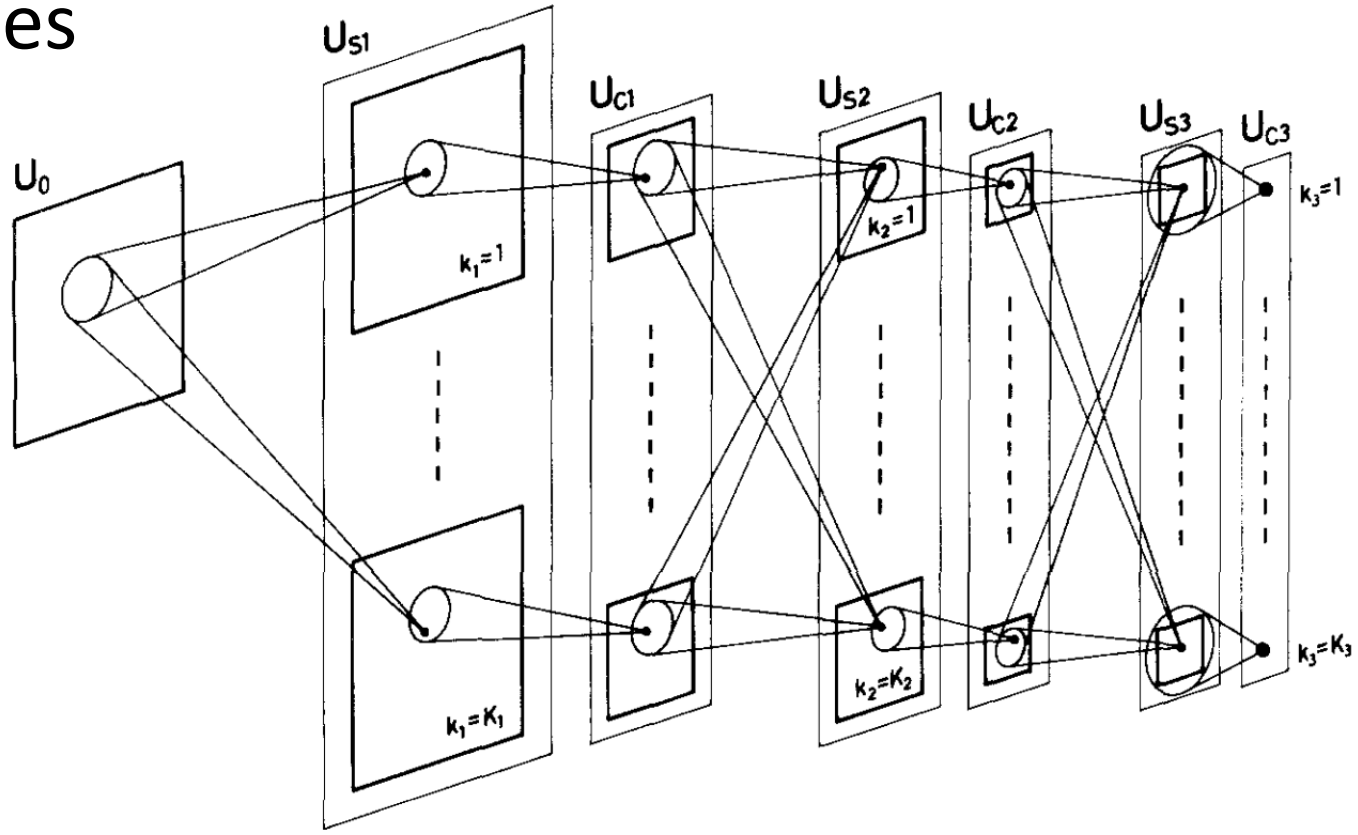
- Common operations of ConvNet
  - convolution
  - activation function
  - pooling
  - fully-connected layer
  - softmax

- **Train with BackProp!**

Convolutional Neural Networks: Case Study

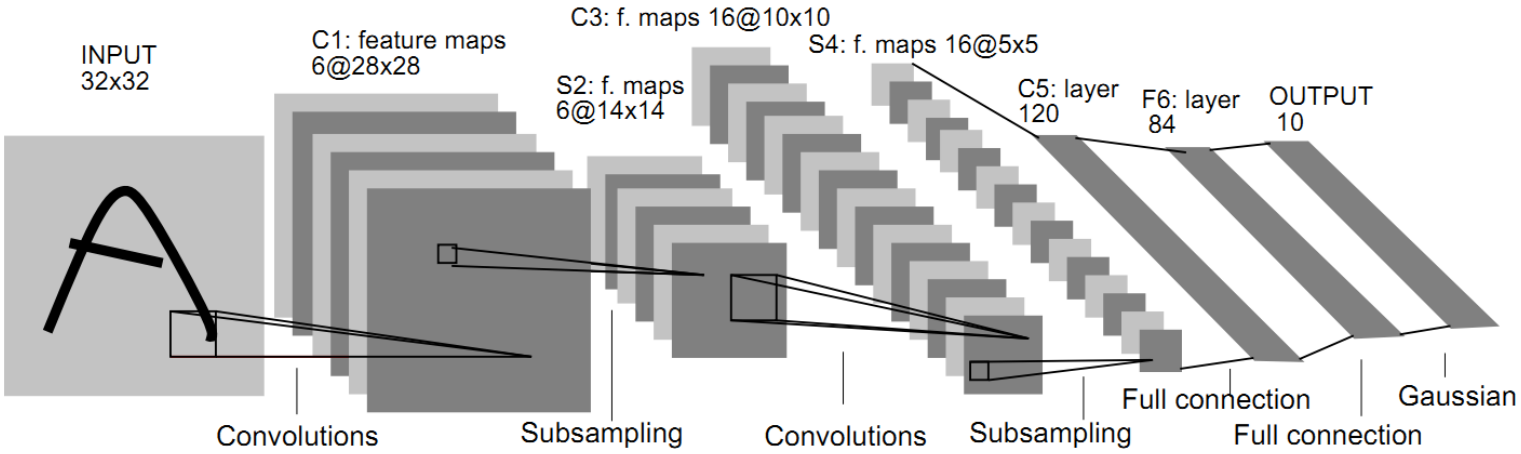
Neocognitron, 1979

- root of ConvNet
- sliding-window features
- composing basic modules
- no backprop



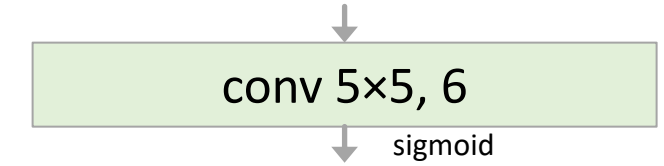
Kunihiko Fukushima, “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”, 1980

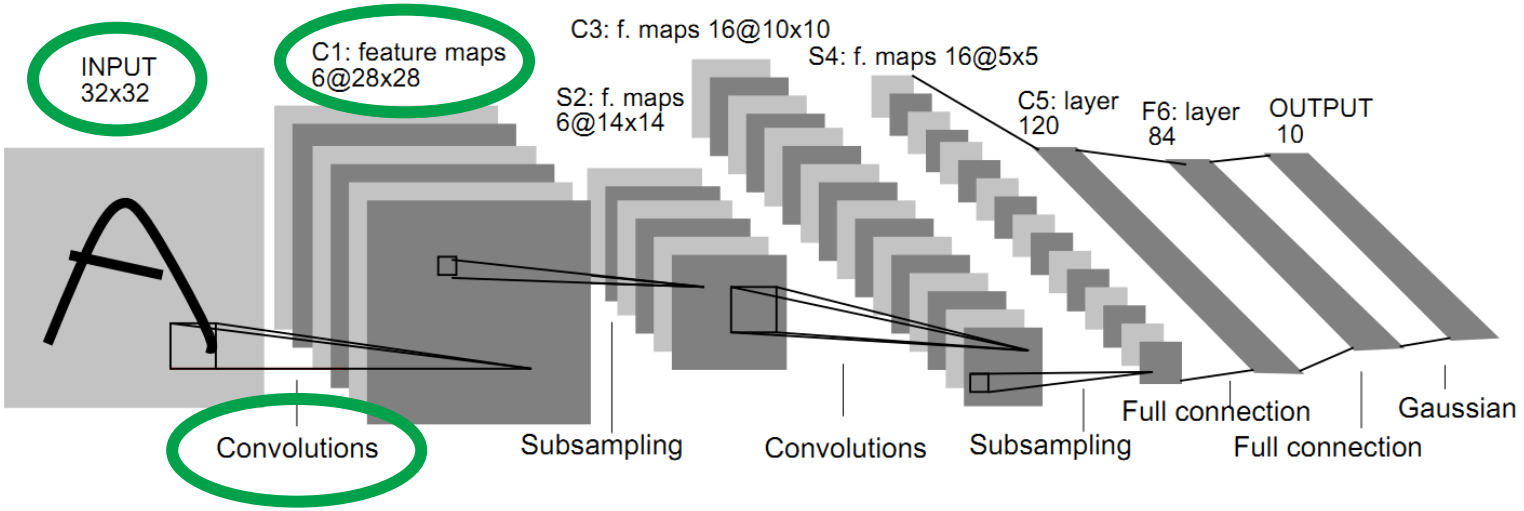
LeNet, 1989



"Backpropagation applied to handwritten zip code recognition", LeCun et al. 1989, "Gradient-based learning applied to document recognition", LeCun et al. 1998

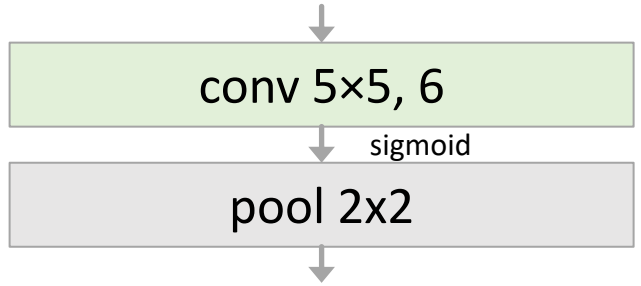
LeNet, 1989

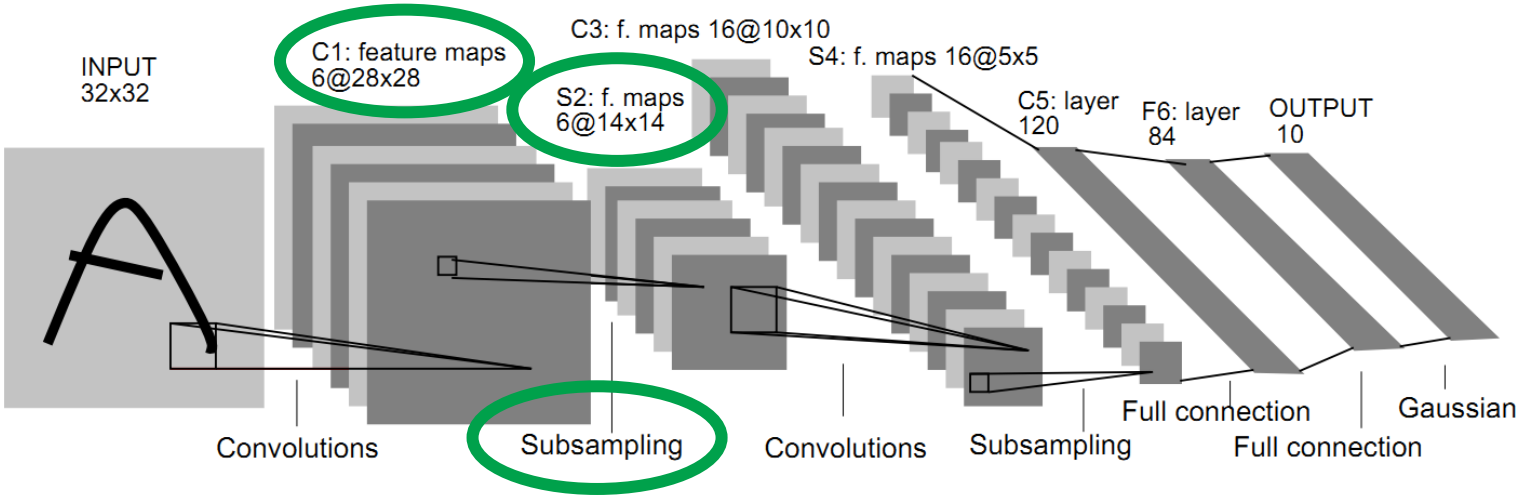




"Backpropagation applied to handwritten zip code recognition", LeCun et al. 1989, "Gradient-based learning applied to document recognition", LeCun et al. 1998

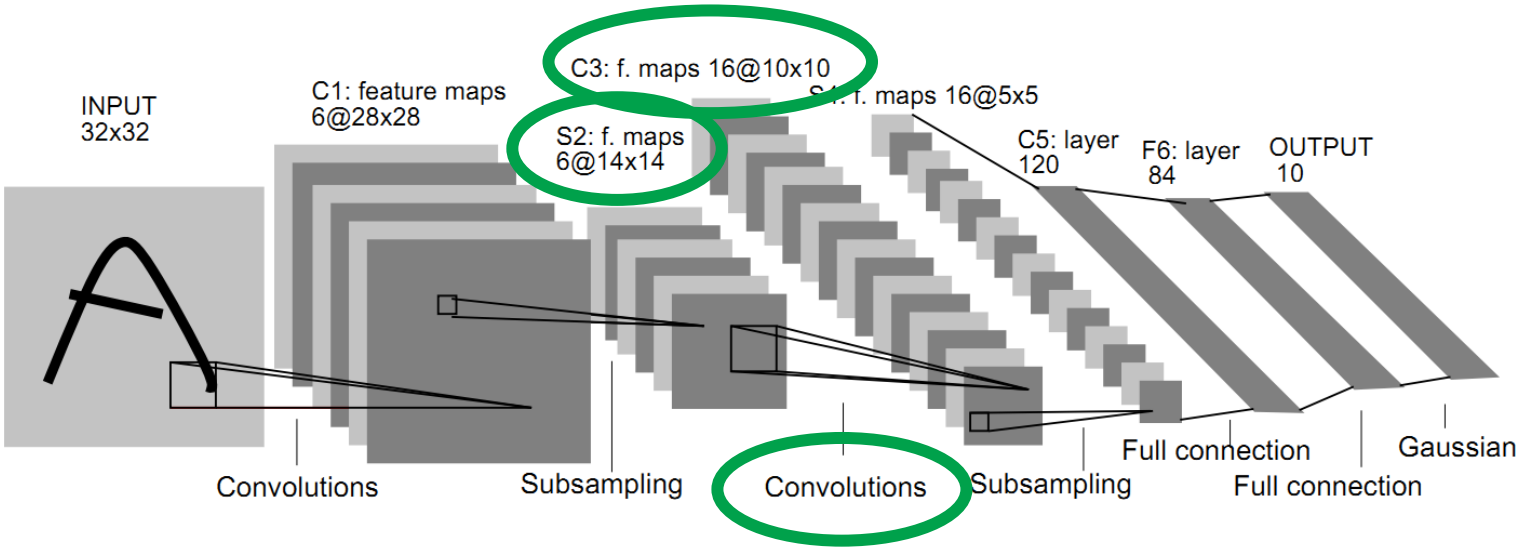
LeNet, 1989

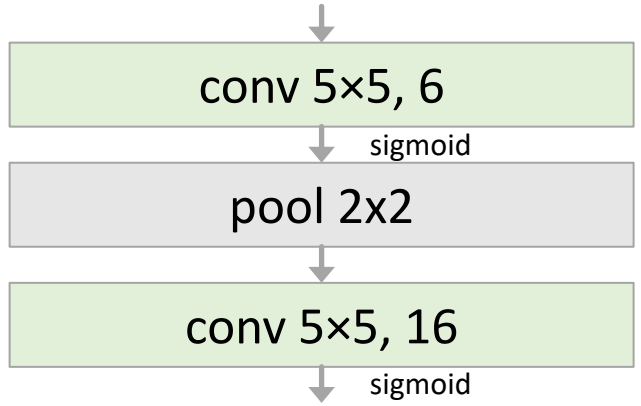




"Backpropagation applied to handwritten zip code recognition", LeCun et al. 1989, "Gradient-based learning applied to document recognition", LeCun et al. 1998

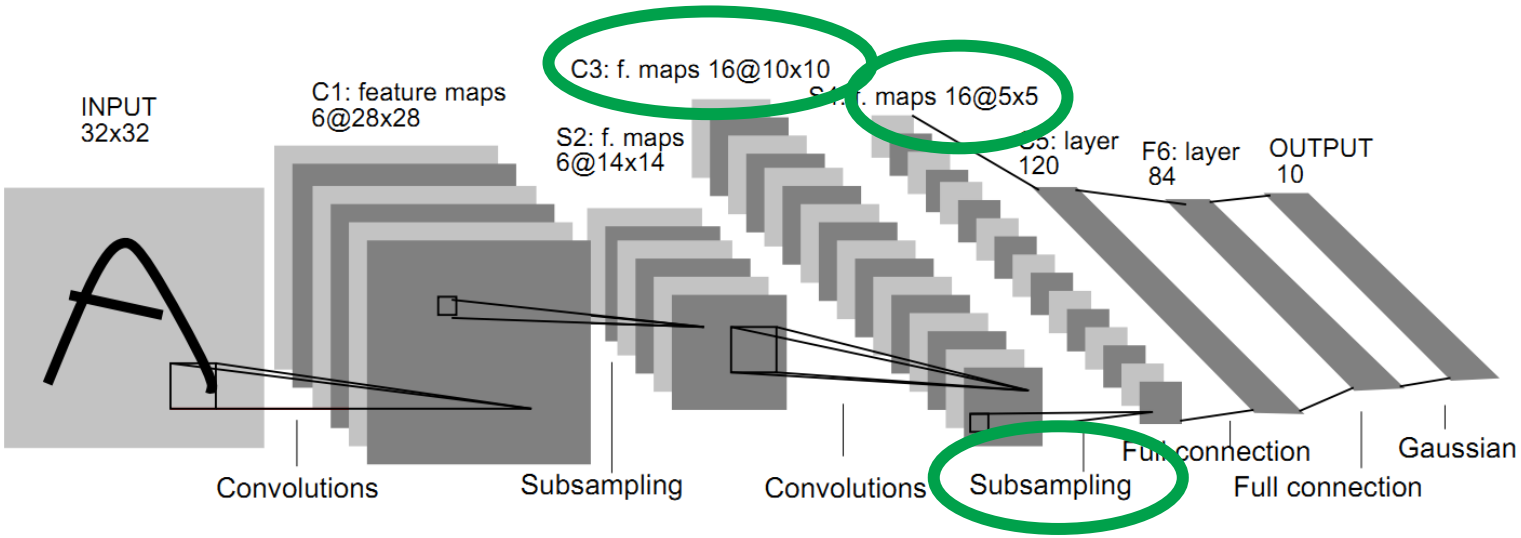
LeNet, 1989

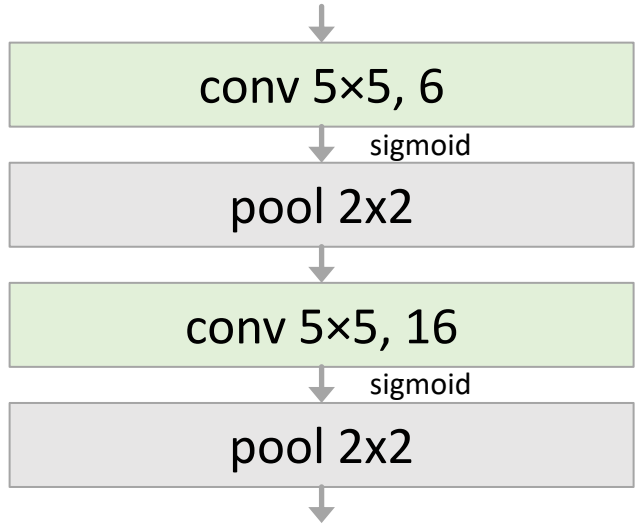




"Backpropagation applied to handwritten zip code recognition", LeCun et al. 1989, "Gradient-based learning applied to document recognition", LeCun et al. 1998

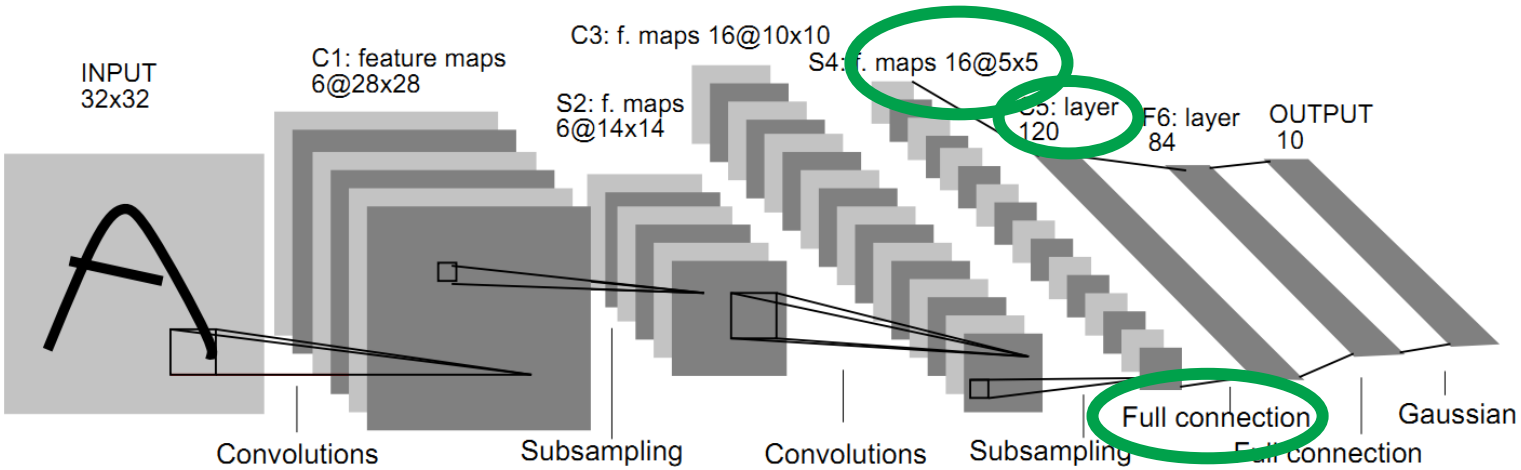
LeNet, 1989

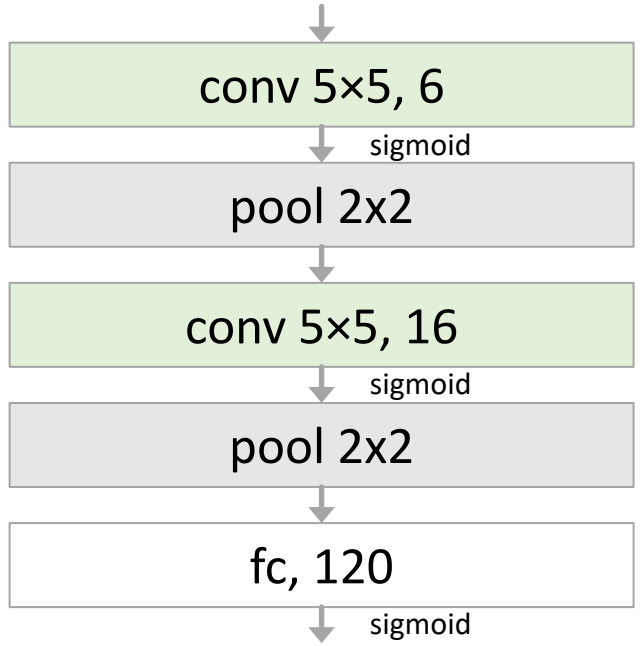




"Backpropagation applied to handwritten zip code recognition", LeCun et al. 1989, "Gradient-based learning applied to document recognition", LeCun et al. 1998

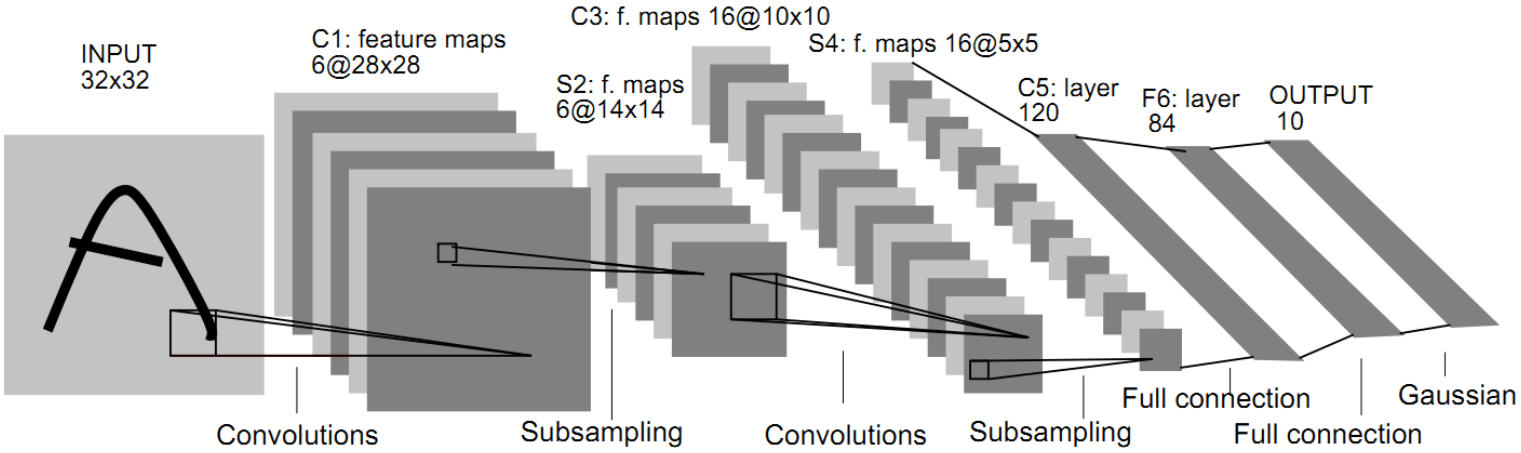
LeNet, 1989

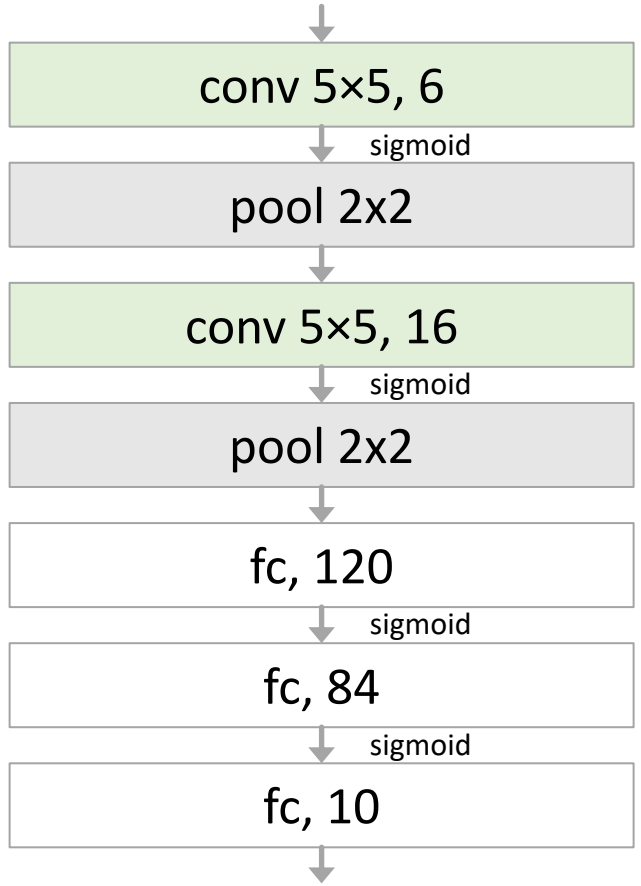




"Backpropagation applied to handwritten zip code recognition", LeCun et al. 1989, "Gradient-based learning applied to document recognition", LeCun et al. 1998

LeNet, 1989

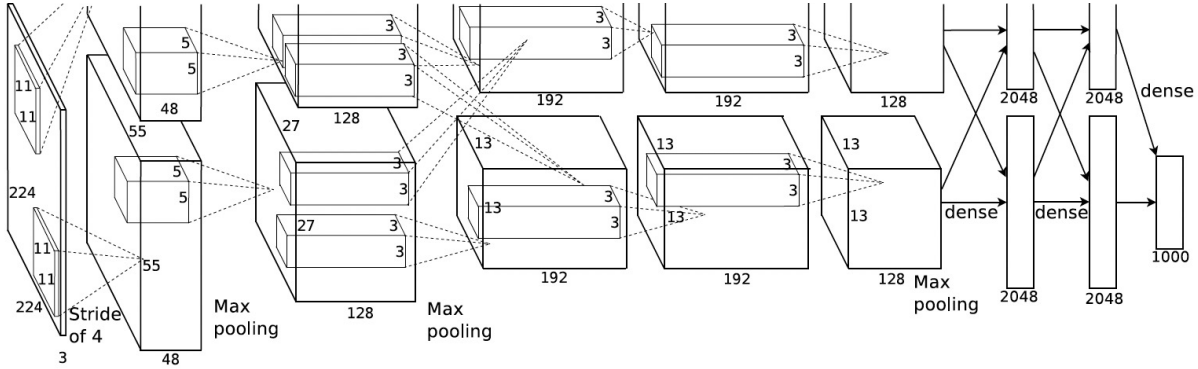




"Backpropagation applied to handwritten zip code recognition", LeCun et al. 1989, "Gradient-based learning applied to document recognition", LeCun et al. 1998

AlexNet, 2012

The Deep Learning Revolution



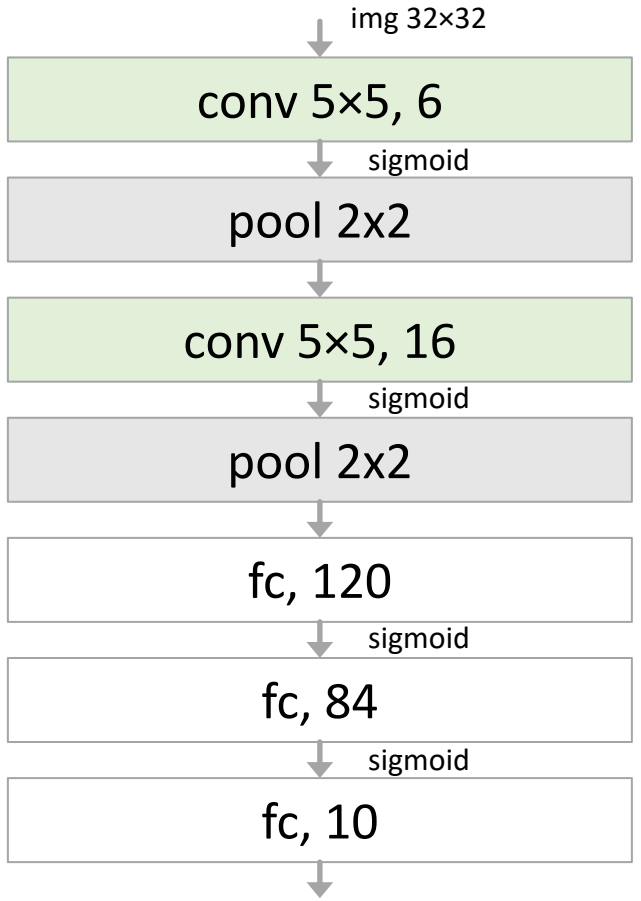
Scaling up ConvNet

- data scaling
  - ImageNet, 1.28 million images, 1000 classes
- model scaling
  - “60 million parameters and 650,000 neurons”
- reducing overfitting
  - data augmentation
  - dropout
- GPU implementation

“ImageNet Classification with Deep Convolutional Neural Networks”, Krizhevsky, Sutskever, Hinton. NeurIPS 2012

LeNet vs. AlexNet

- deeper (more layers)
- wider (more channels)
- ReLU (not sigmoid)



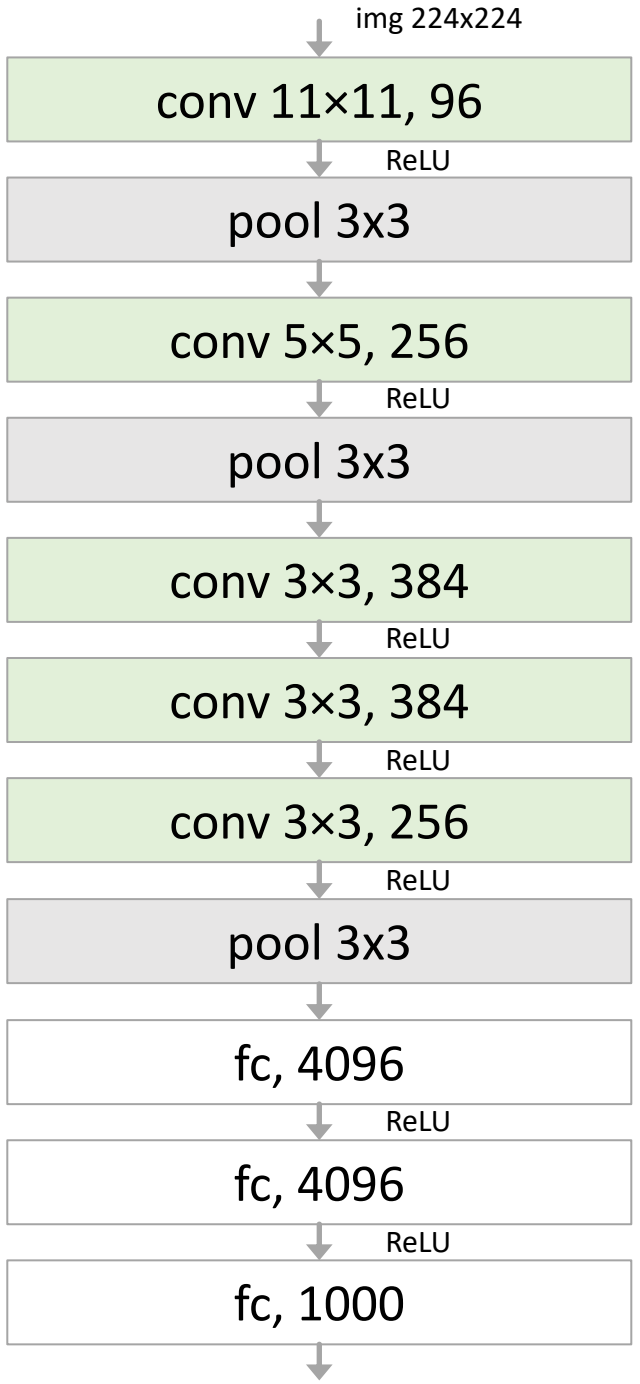
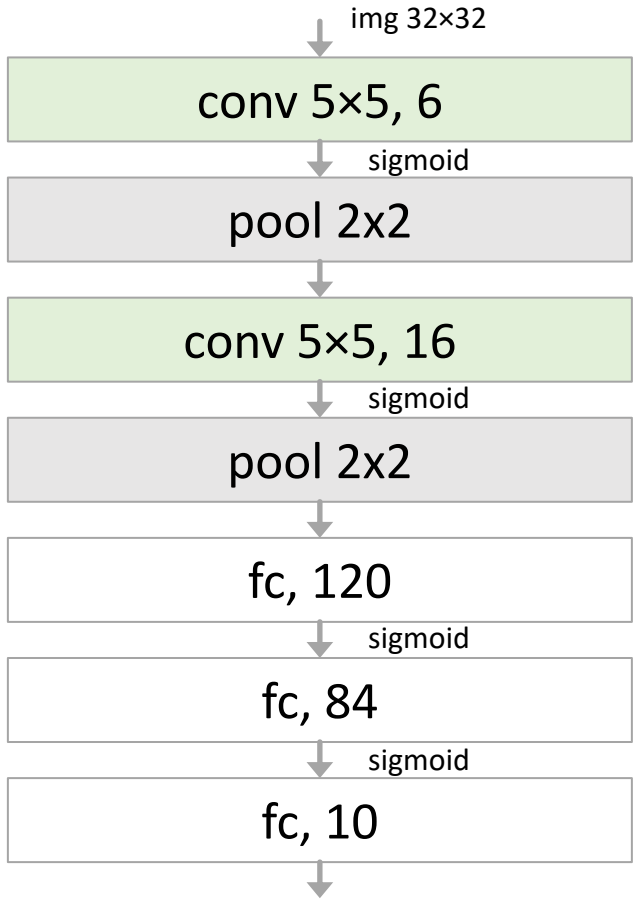


figure inspired by: https://en.wikipedia.org/wiki/LeNet

LeNet vs. AlexNet

- **deeper (more layers)**
- wider (more channels)
- ReLU (not sigmoid)



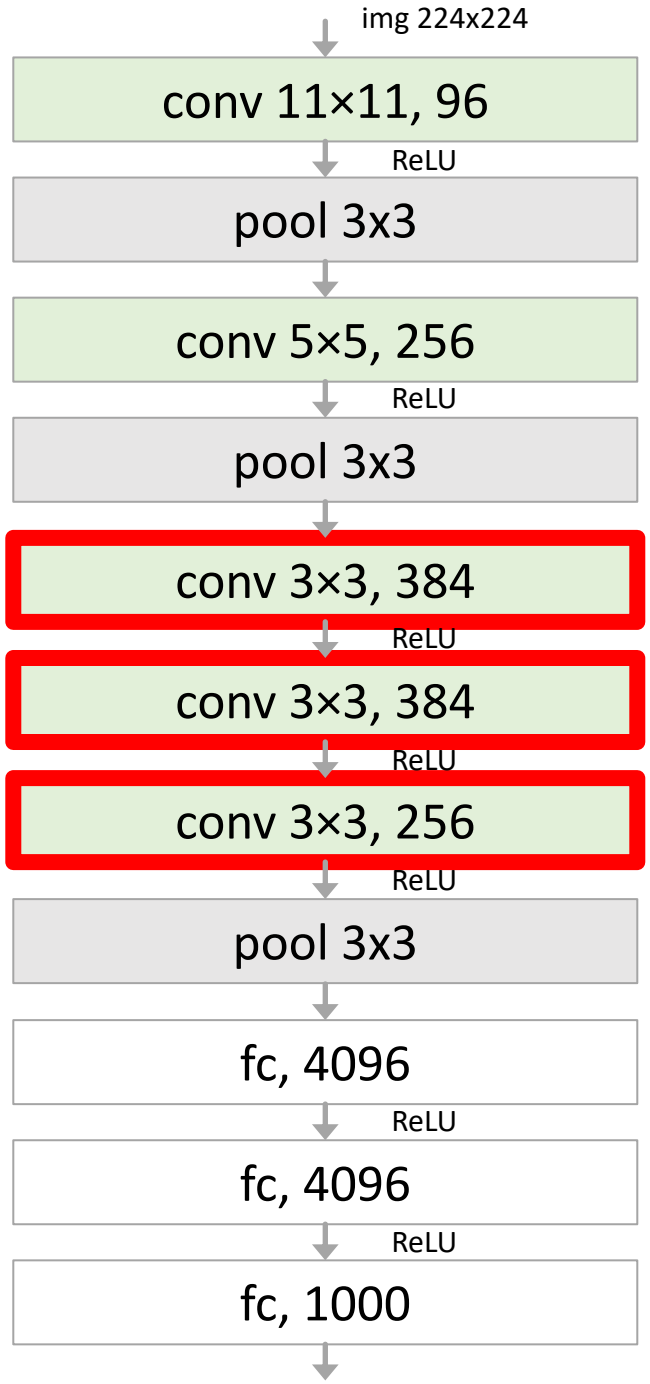
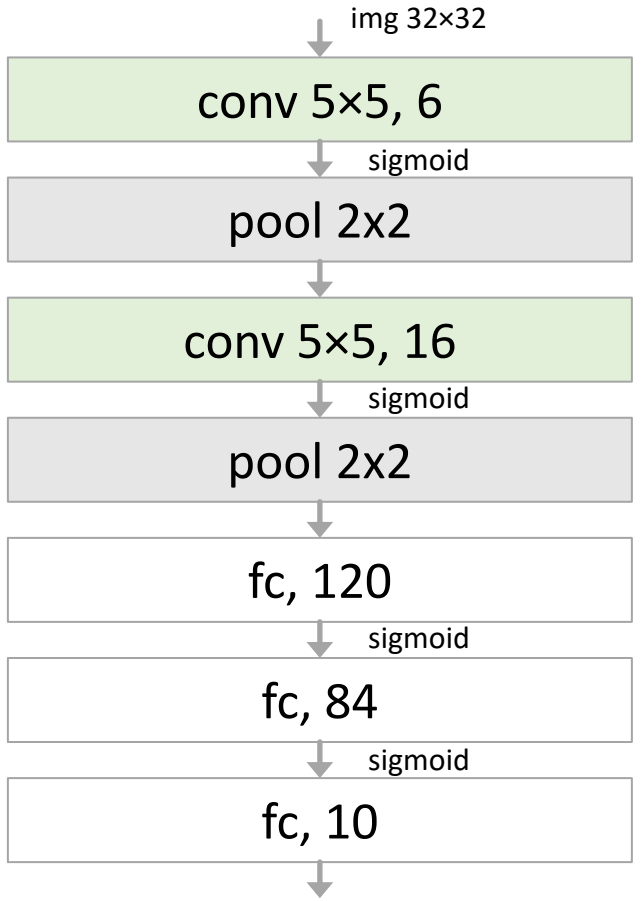


figure inspired by: https://en.wikipedia.org/wiki/LeNet

LeNet vs. AlexNet

- deeper (more layers)
- **wider (more channels)**
- ReLU (not sigmoid)



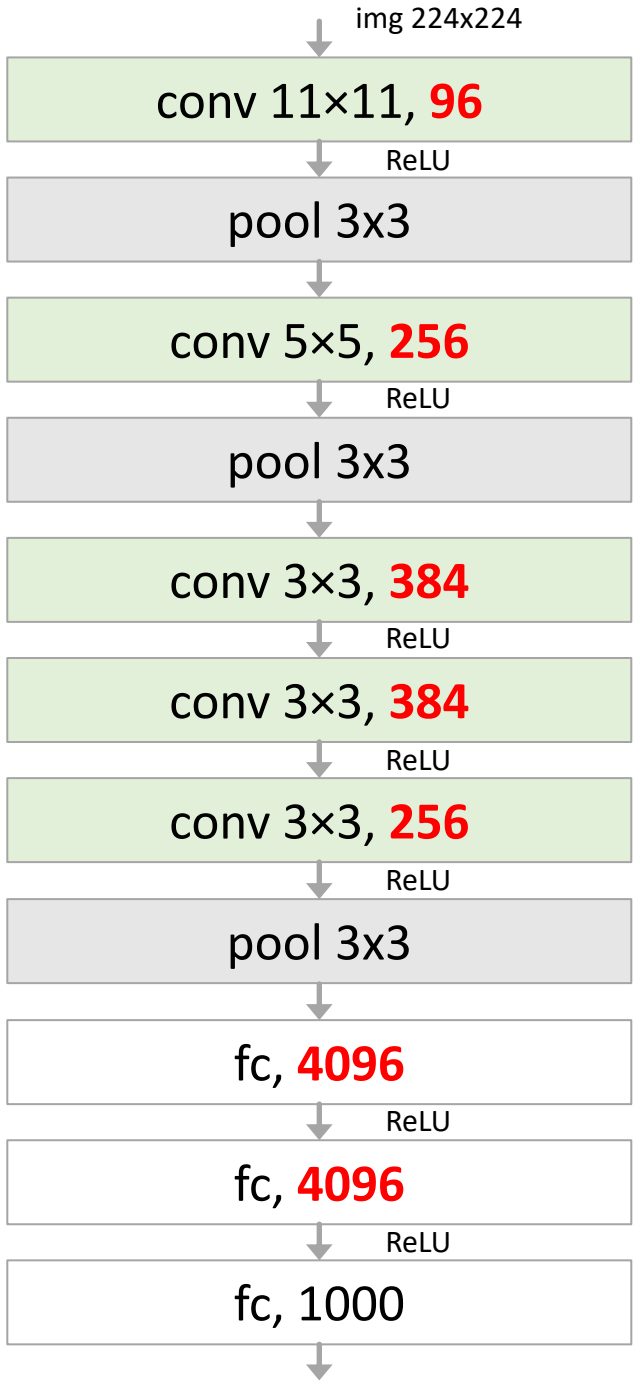
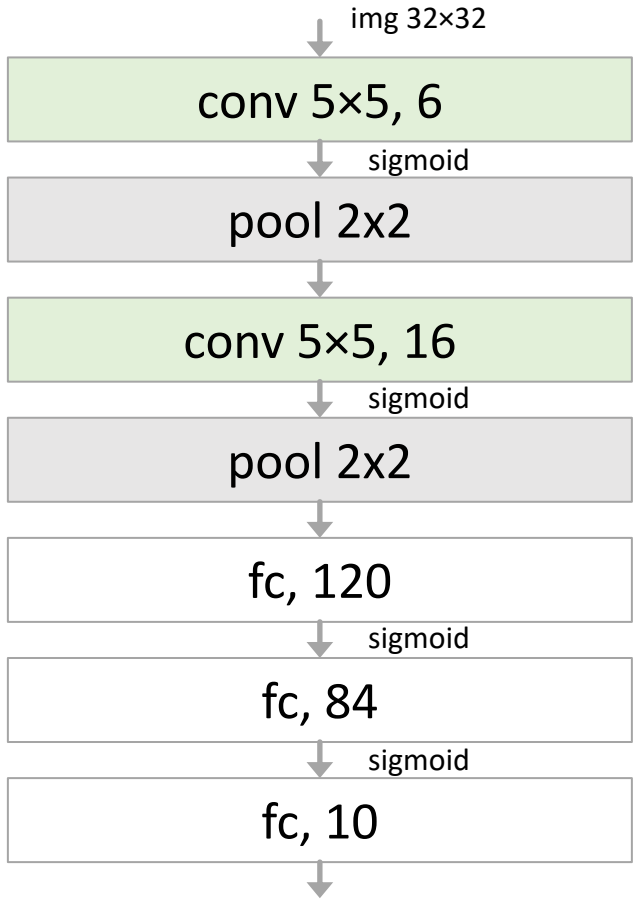


figure inspired by: https://en.wikipedia.org/wiki/LeNet

LeNet vs. AlexNet

- deeper (more layers)
- wider (more channels)
- **ReLU (not sigmoid)**



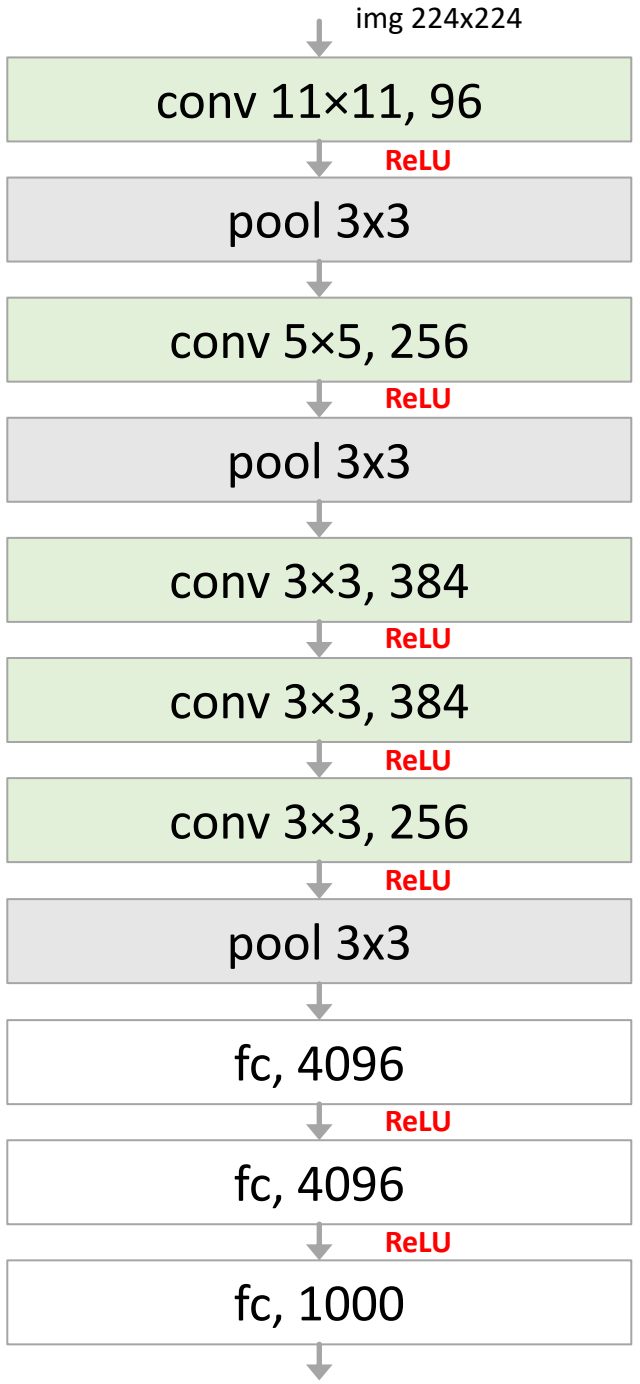
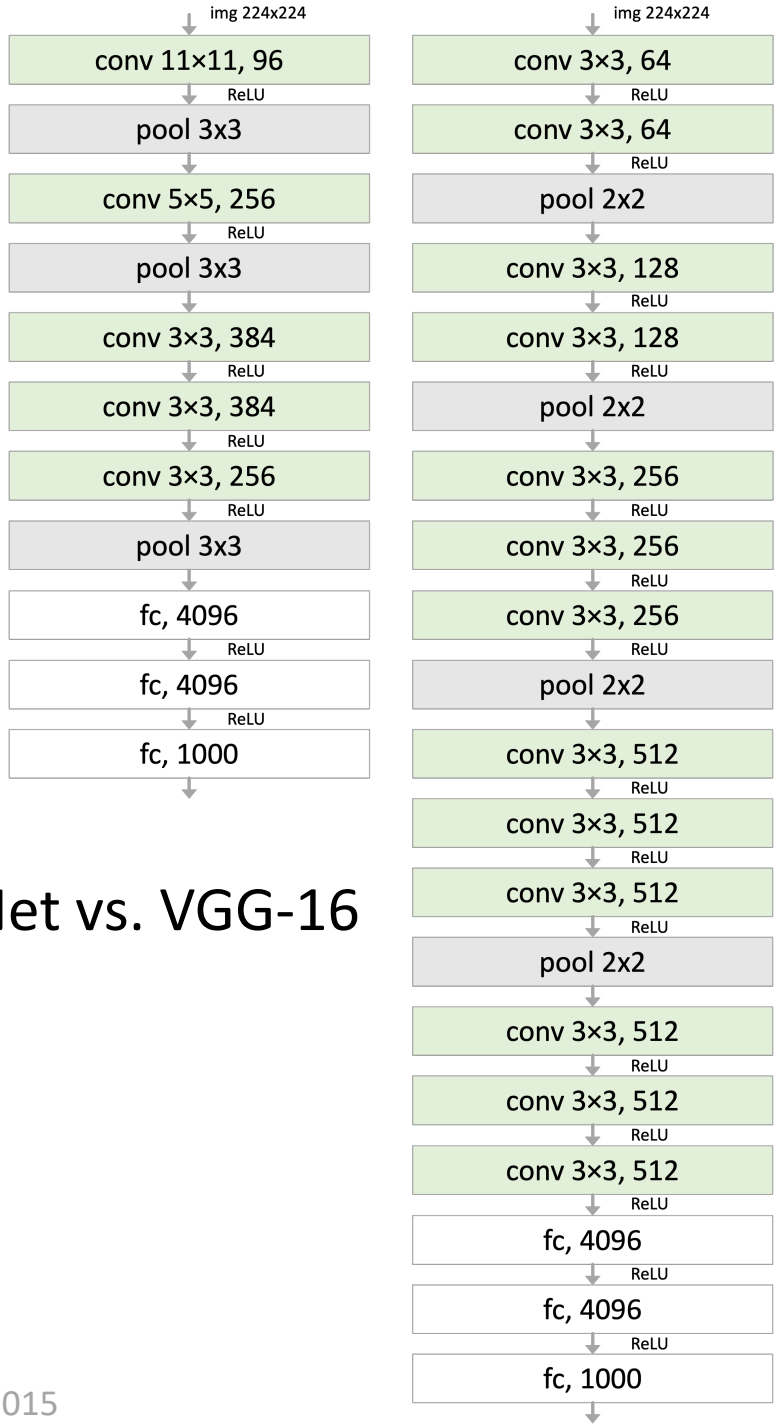


figure inspired by: https://en.wikipedia.org/wiki/LeNet

VGG nets, 2014

Very deep ConvNets

- stack the same modules
  - conv: all 3x3
- very deep
  - 16 or 19 layers
- clear evidence: deeper is better
- **Composing basic operations into complex functions**

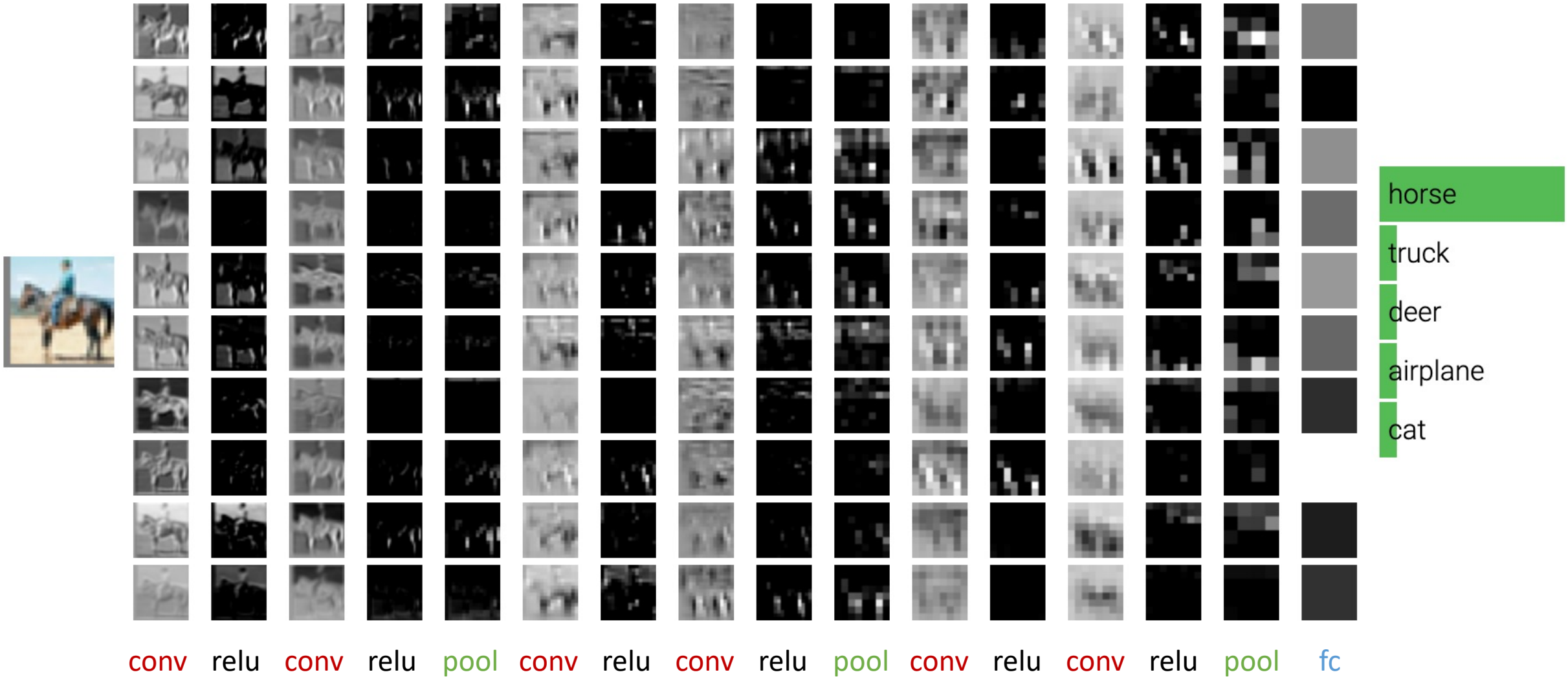


AlexNet vs. VGG-16

"Very Deep Convolutional Networks for Large-Scale Image Recognition", Simonyan & Zisserman. ICLR 2015

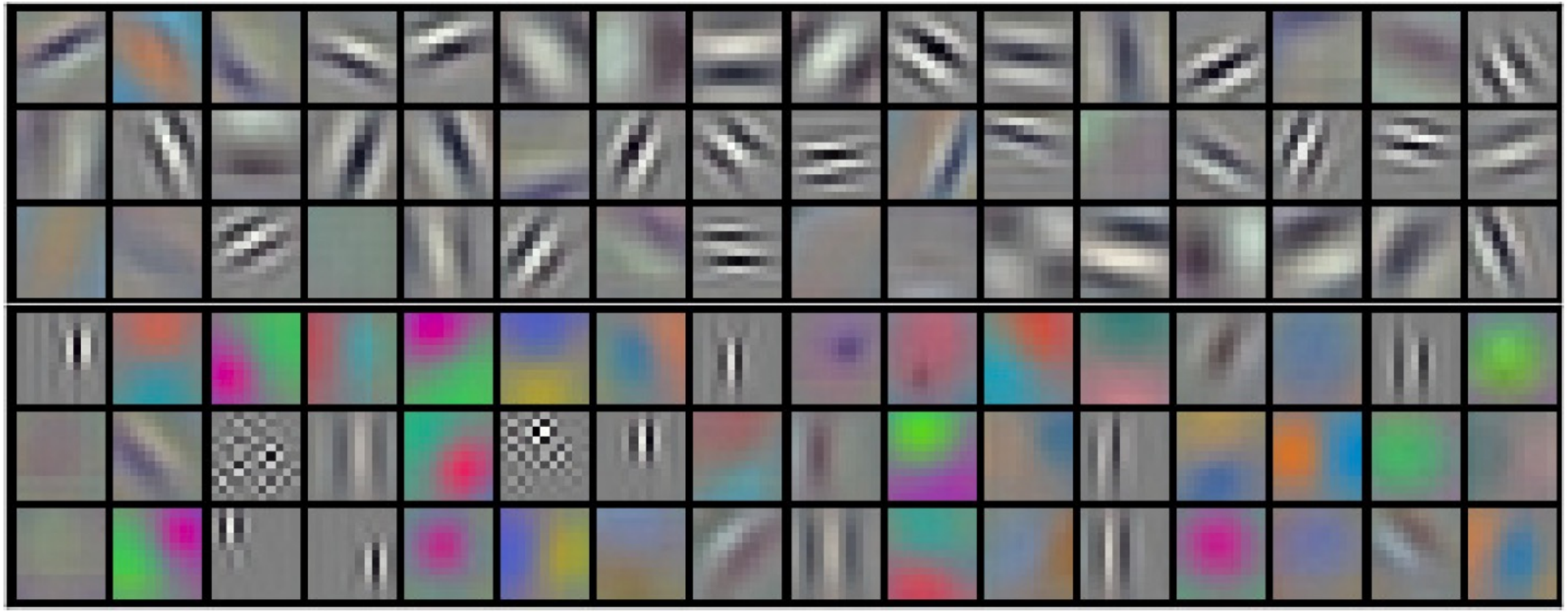
Visualizing Convolutional Neural Networks

Visualizing feature maps



see more examples in http://cs231n.stanford.edu/

Visualizing filters



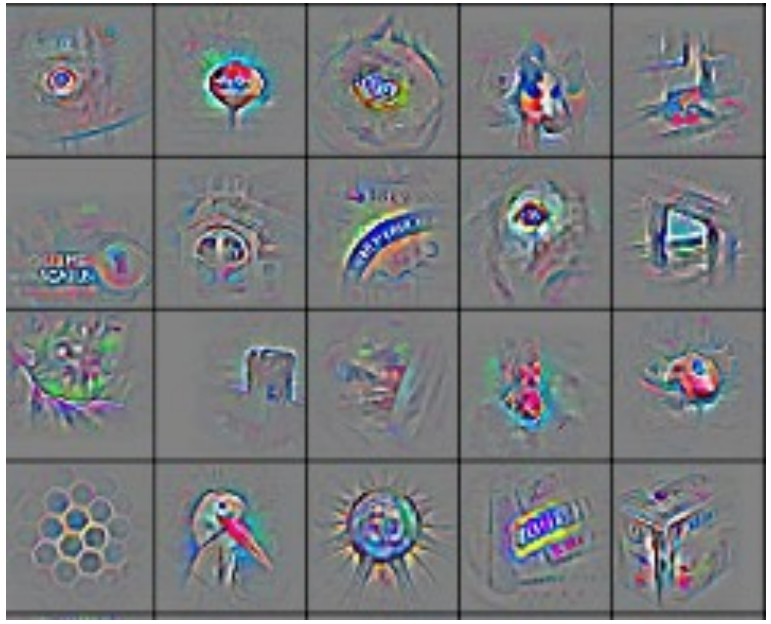
AlexNet's 1st conv layer (96 filters):
edge or color detectors

"ImageNet Classification with Deep Convolutional Neural Networks", Krizhevsky, Sutskever, Hinton. NeurIPS 2012

Visualizing 1-hot feature maps by BackProp

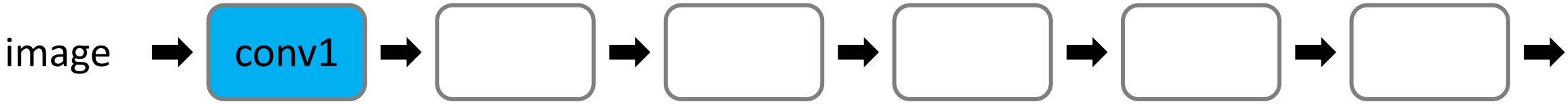
To find “what input can produce a specific feature”

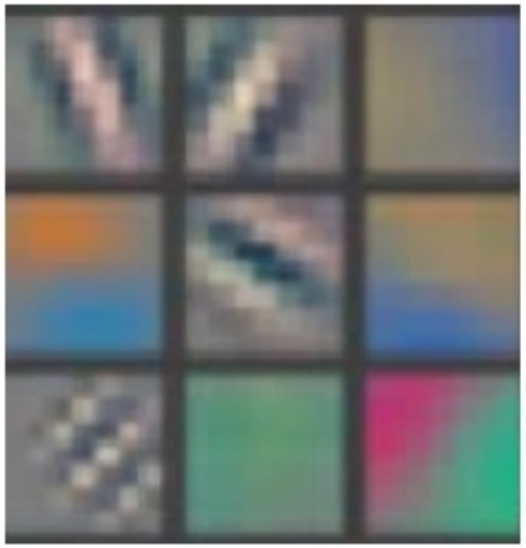
- set a one-hot feature map
- back-prop to pixels



“Visualizing and Understanding Convolutional Networks”, Zeiler & Fergus. arXiv 2013, ECCV 2014

Visualizing 1-hot feature maps by BackProp



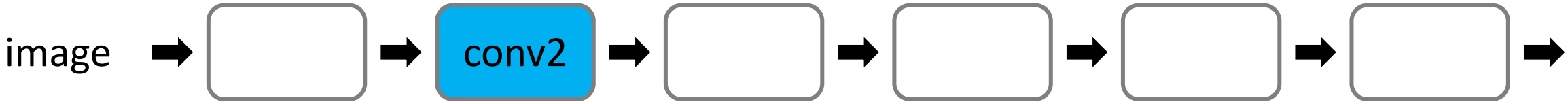


features

stimuli
(patches with the highest
1-hot activations)

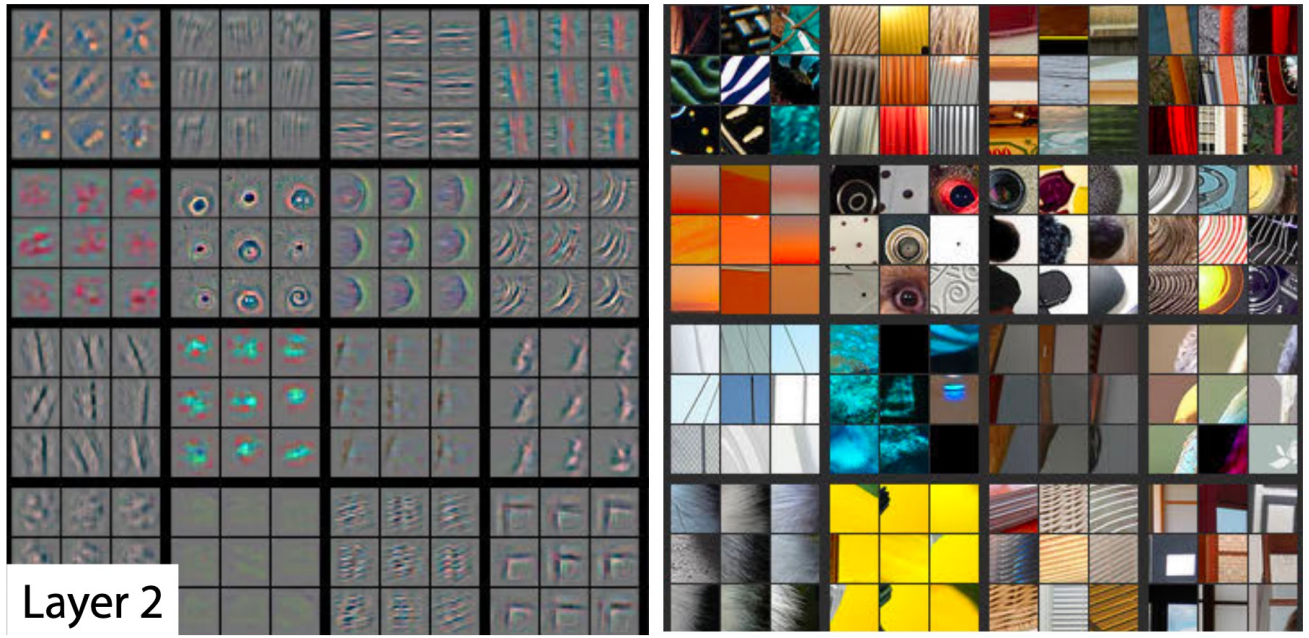
"Visualizing and Understanding Convolutional Networks", Zeiler & Fergus. ECCV 2014

Visualizing 1-hot feature maps by BackProp



features

stimuli

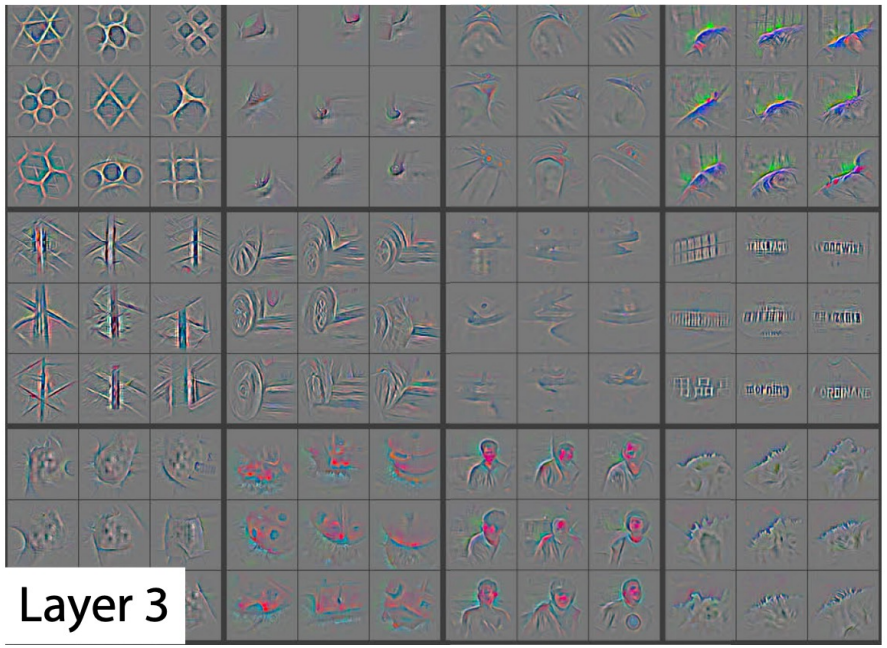


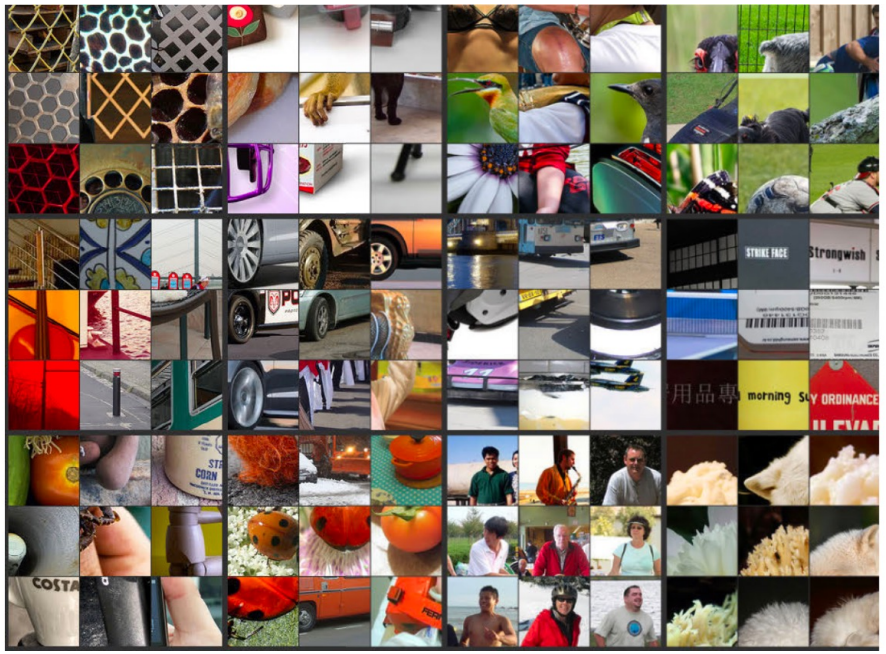
"Visualizing and Understanding Convolutional Networks", Zeiler & Fergus. ECCV 2014

Visualizing 1-hot feature maps by BackProp

features

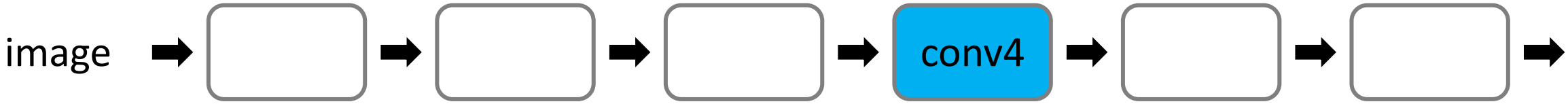
stimuli

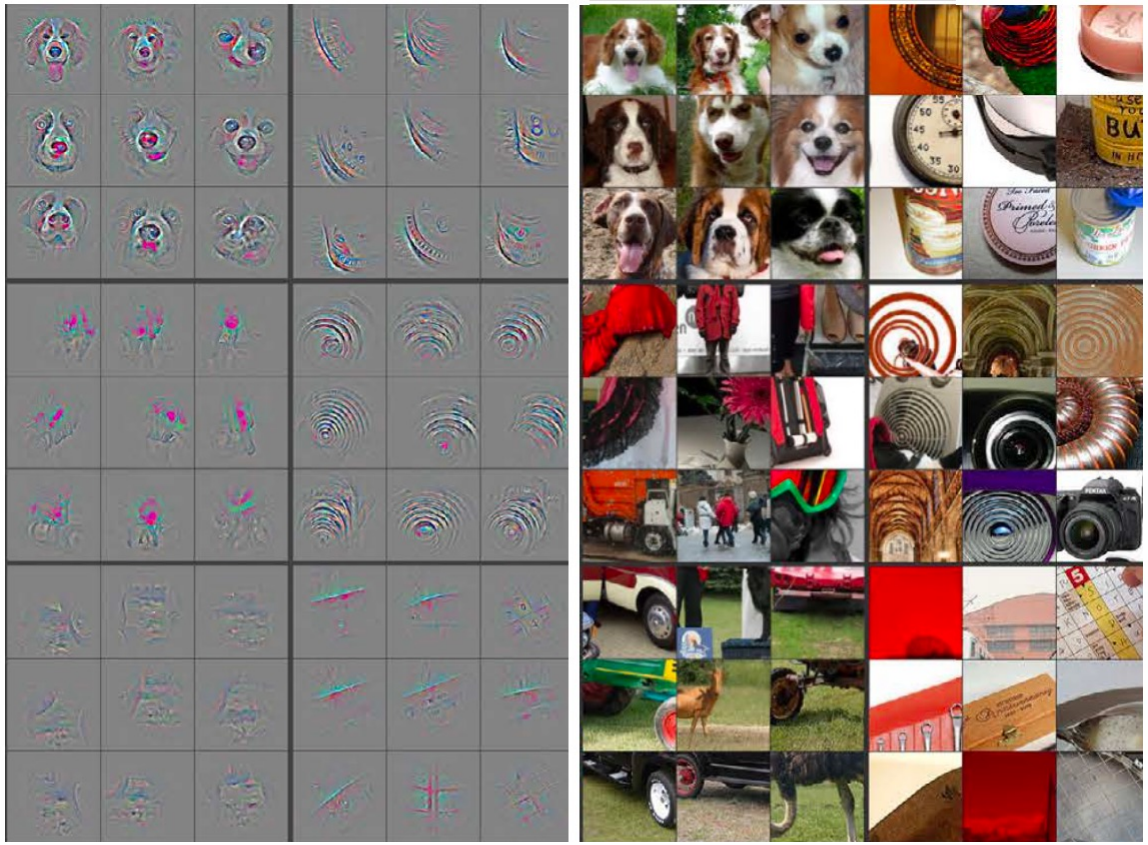


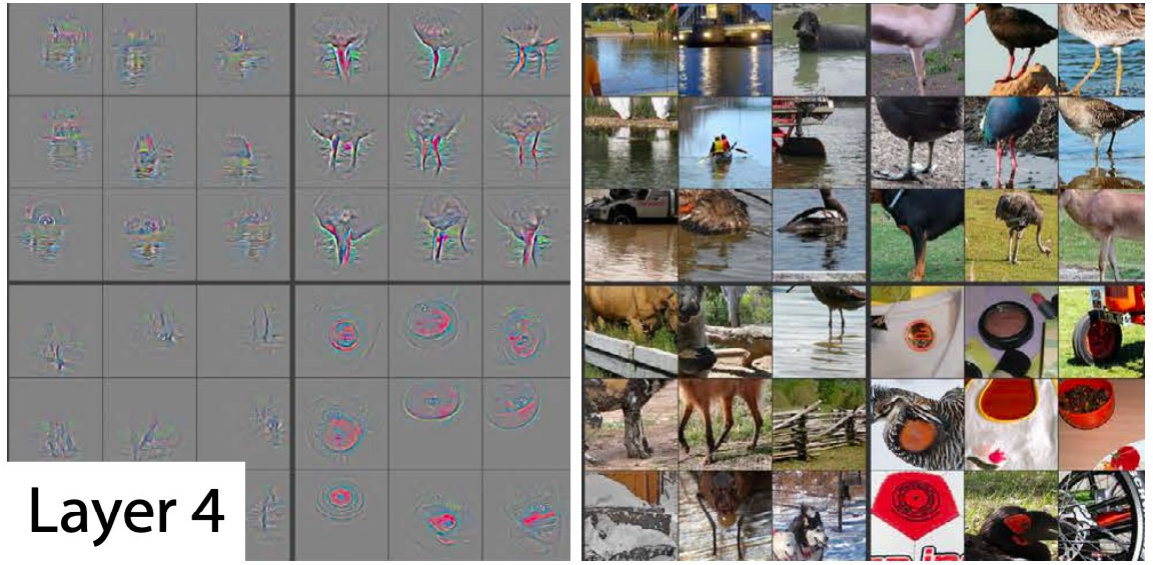


"Visualizing and Understanding Convolutional Networks", Zeiler & Fergus. ECCV 2014

Visualizing 1-hot feature maps by BackProp

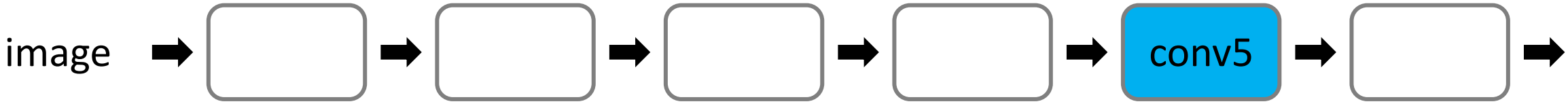


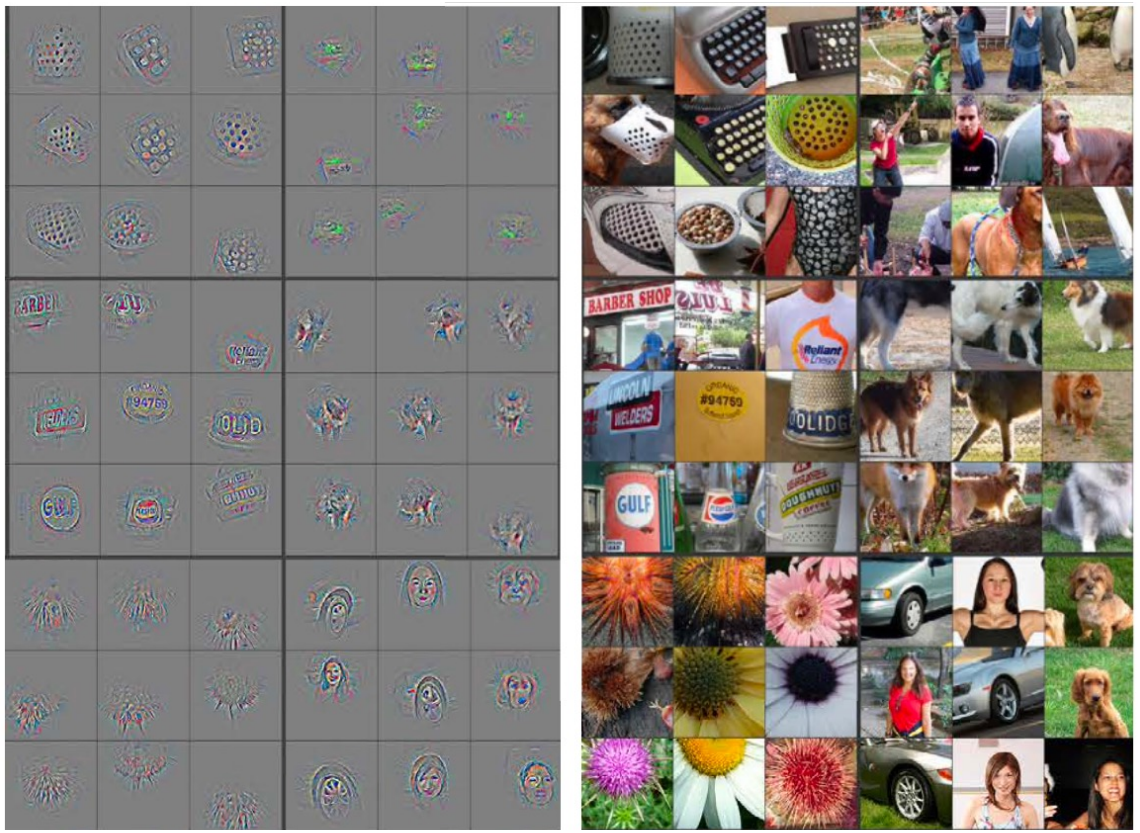


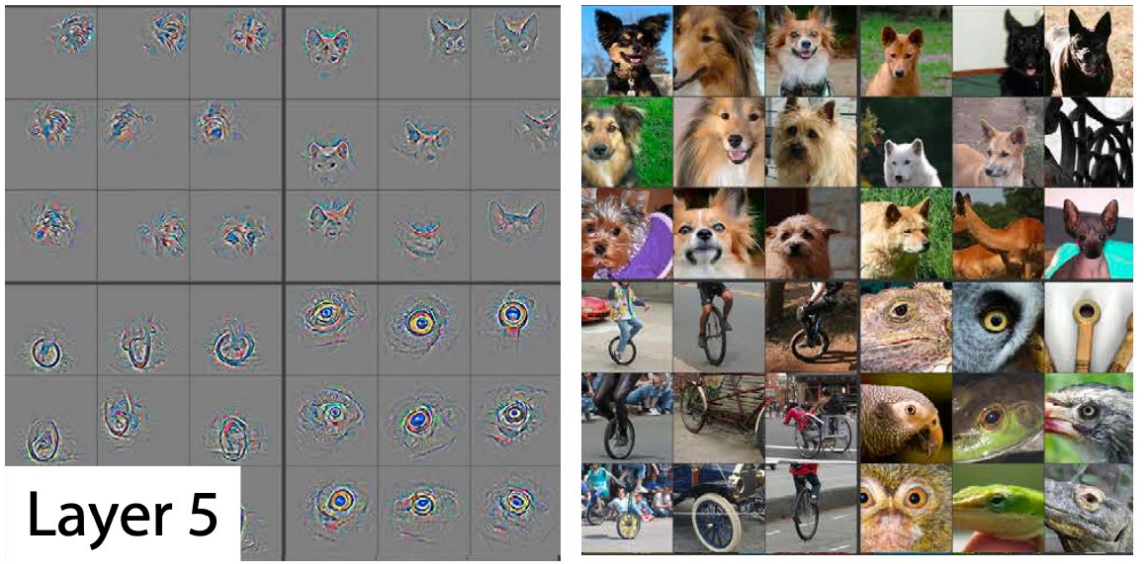


"Visualizing and Understanding Convolutional Networks", Zeiler & Fergus. ECCV 2014

Visualizing 1-hot feature maps by BackProp





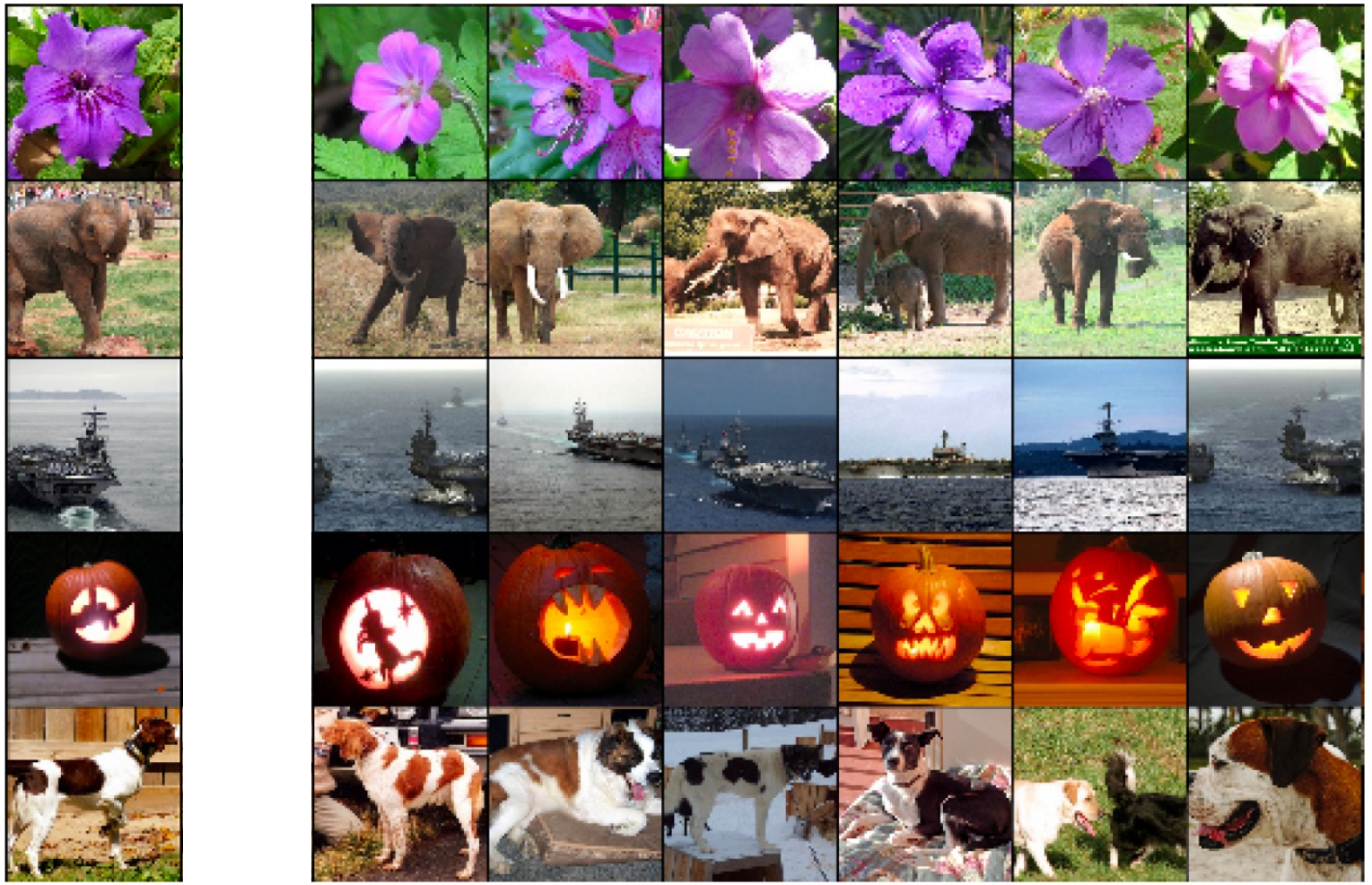


Deeper layers have “higher-level” features.

Visualizing by retrieval

query

nearest neighbors in feature space of AlexNet's last layer



"ImageNet Classification with Deep Convolutional Neural Networks", Krizhevsky, Sutskever, Hinton. NeurIPS 2012

Lecture 9: Convolutional Neural Networks

- Convolution
- Convolutional Neural Networks
- Case Study: LeNet, AlexNet, VGG
- Visualization