香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

COMP 5212
Machine Learning
Lecture 19

Neural Networks, Architectures

Junxian He
Nov 12, 2024

Announcements

1. Midterm Exam paper check session is at this evening (7-8pm), 3520
2. Programming Assignment is out, please start early ✓ Dec 8
3. HW3 is out
4. HW4 will be easy (multi-choice QA only)

public leaderboard 50%

private leaderboard 50%

Nov 25

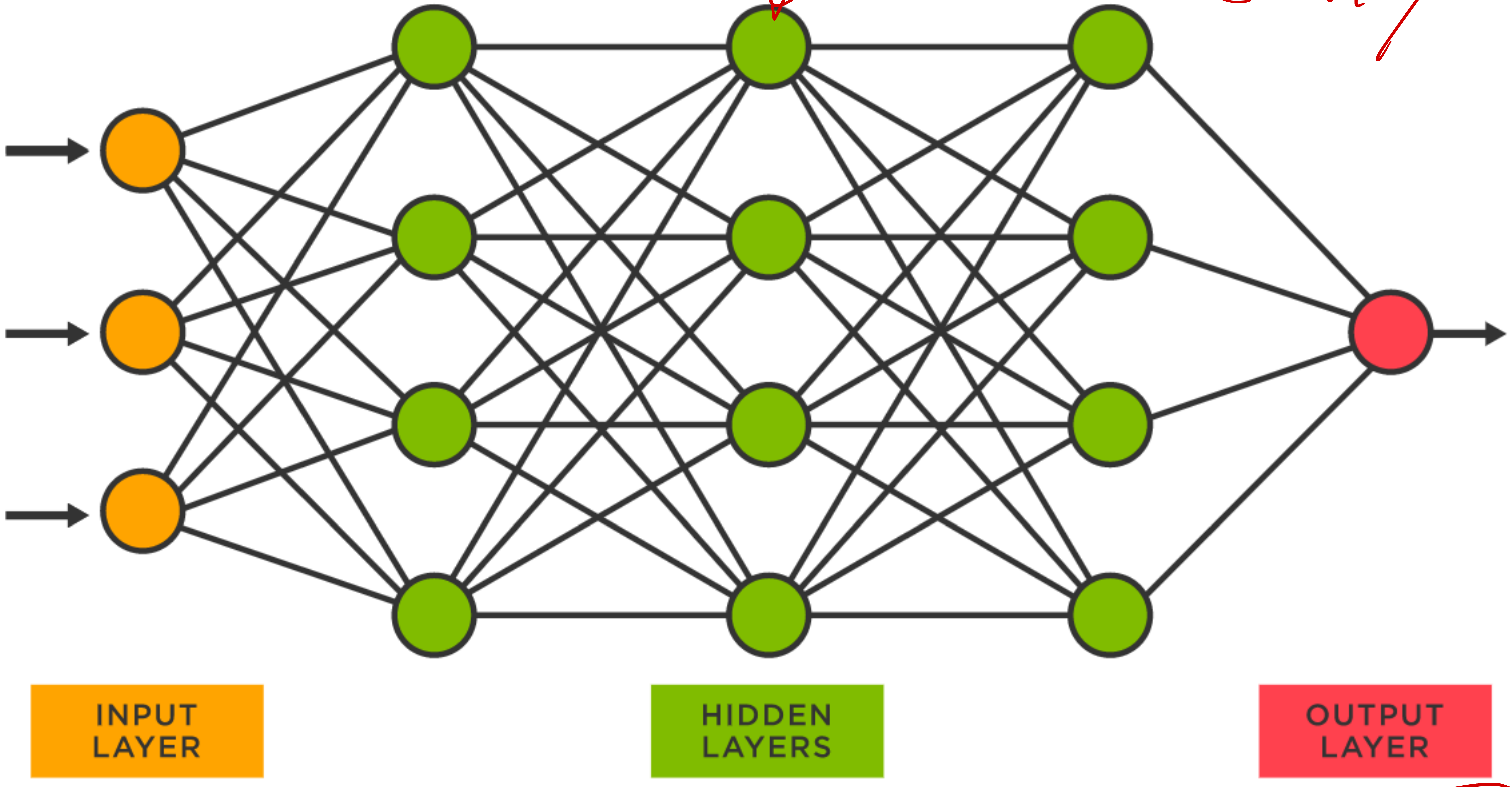
Dec 5

Recap: Multilayer Perceptron Neural Networks (MLP)

Relu
tanh

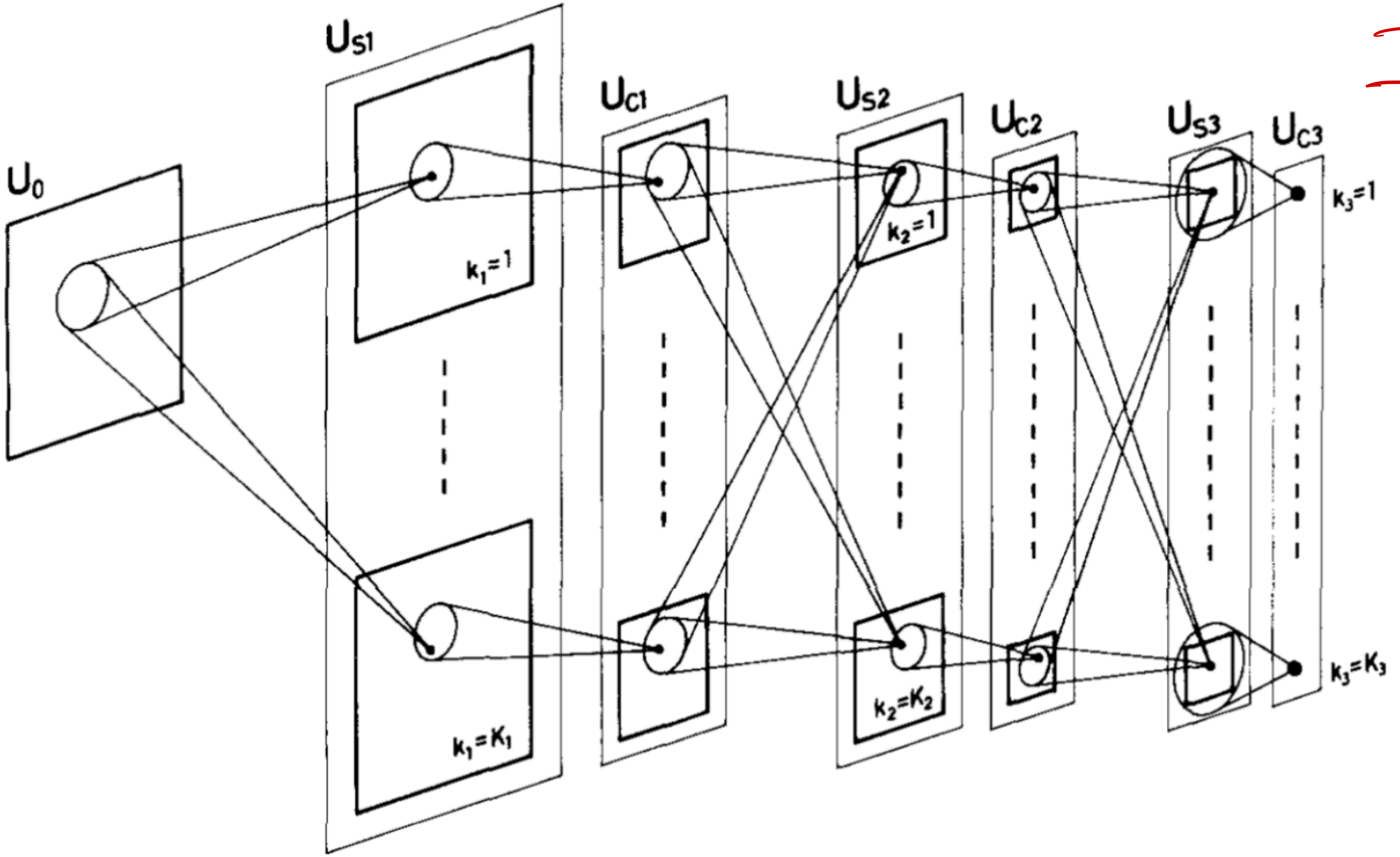
non linear activation

fully-connected



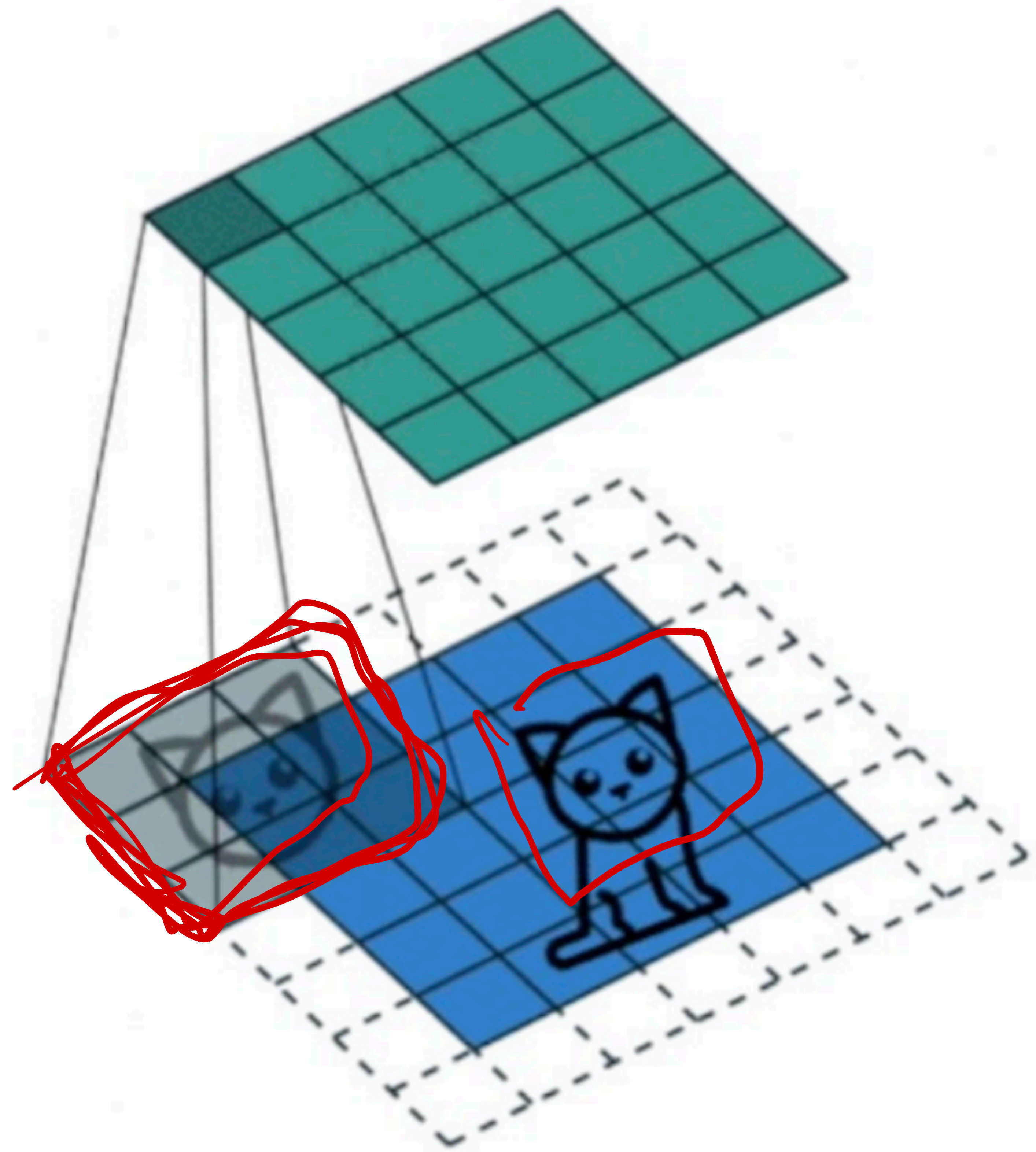
Convolutional Neural Networks

CNN



Convolution is template matching

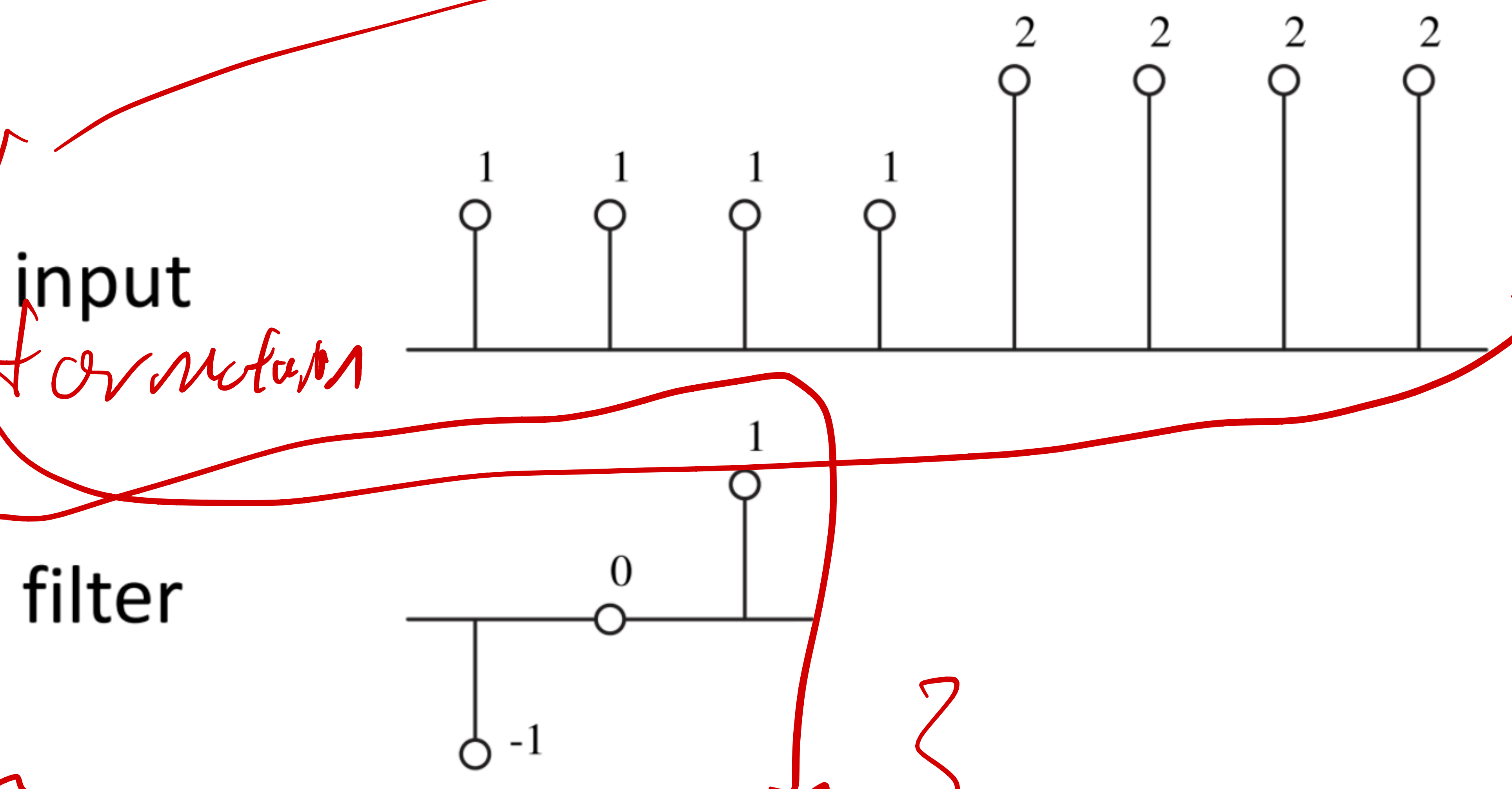
- with a sliding window
- abstract templates
- similarity measured by dot product
- stronger activation, better matching



dot pro

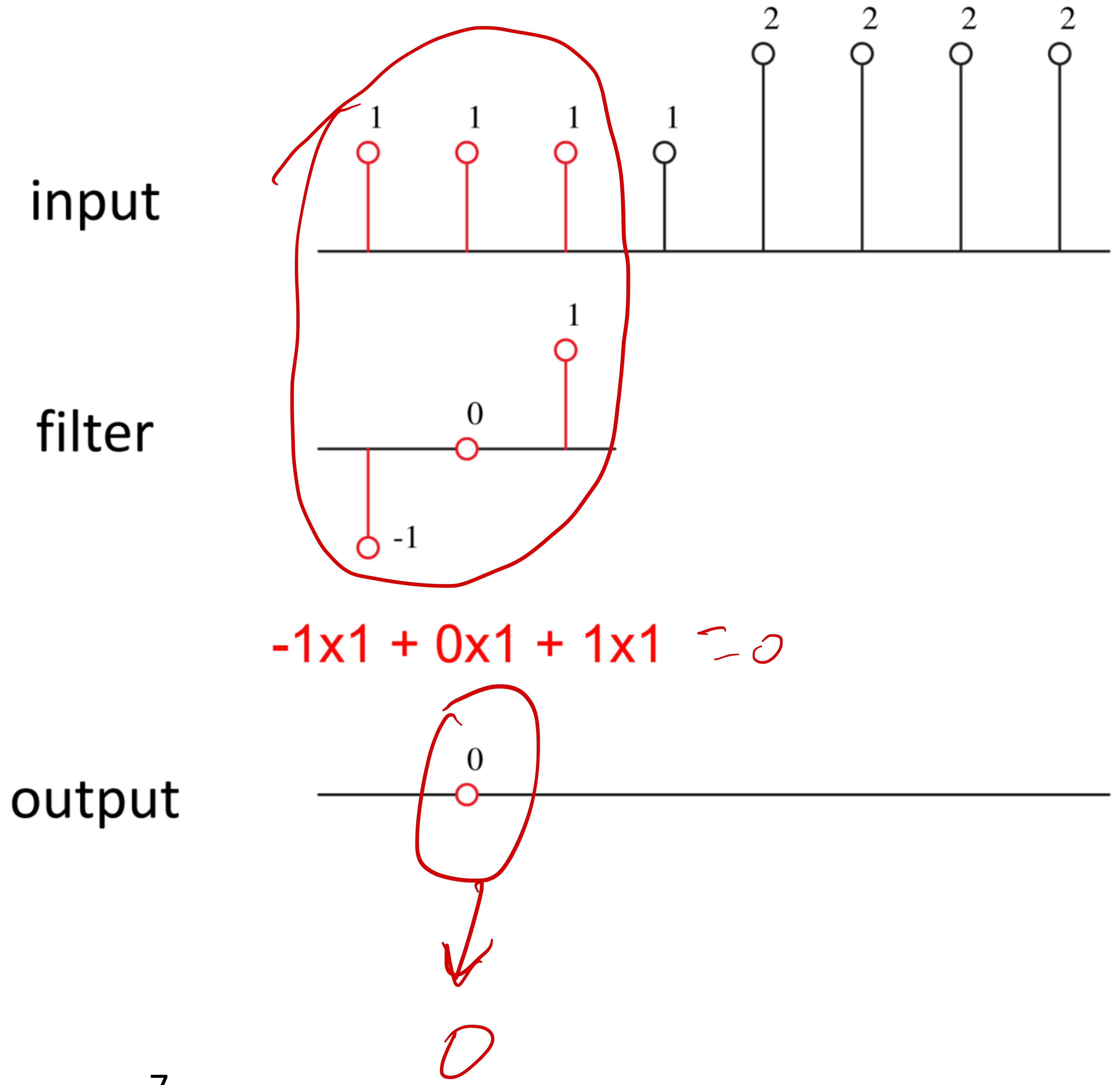
Convolution: a 1-D example

Fourier transformation



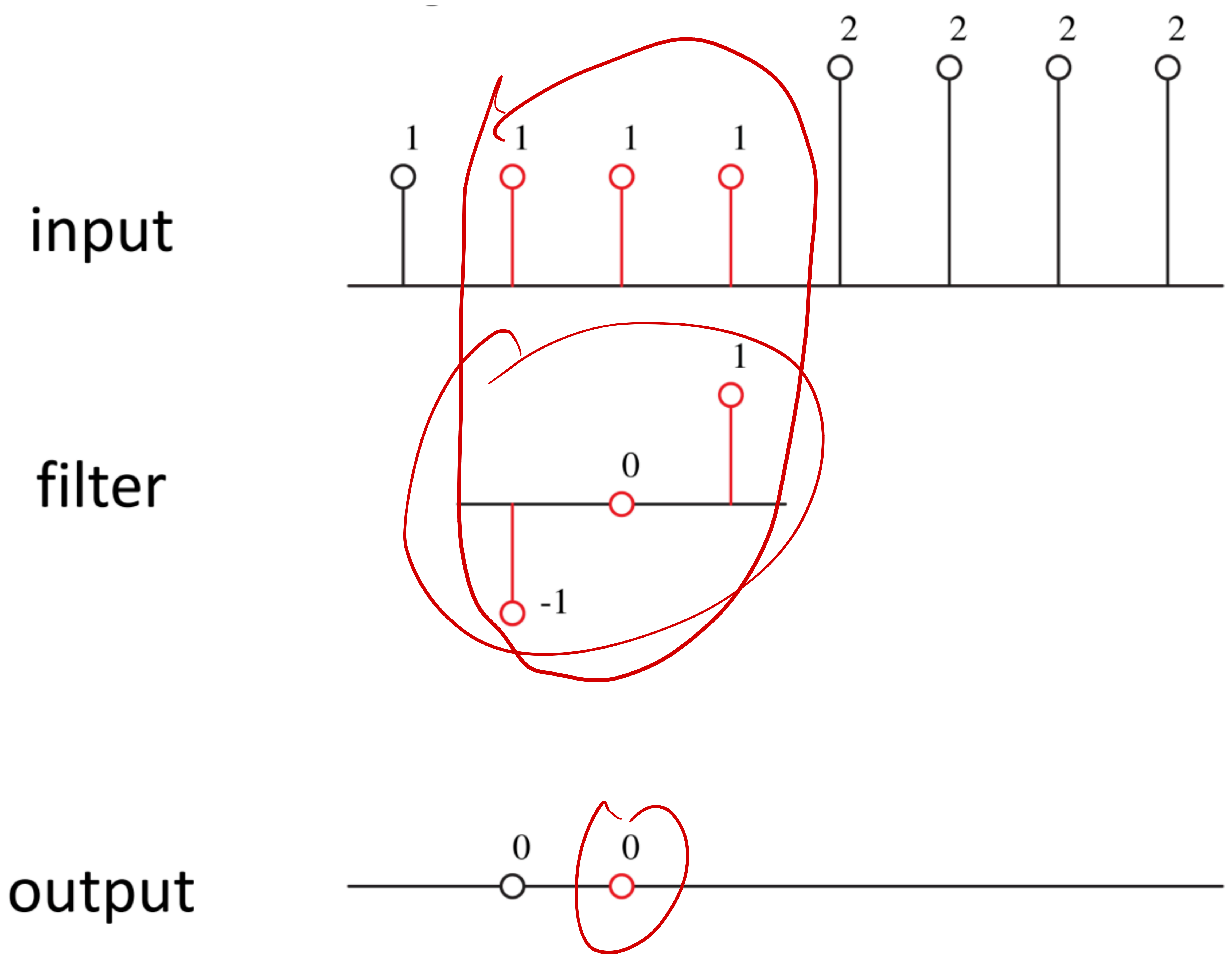
Convolution: a 1-D example

- sliding window
- dot product



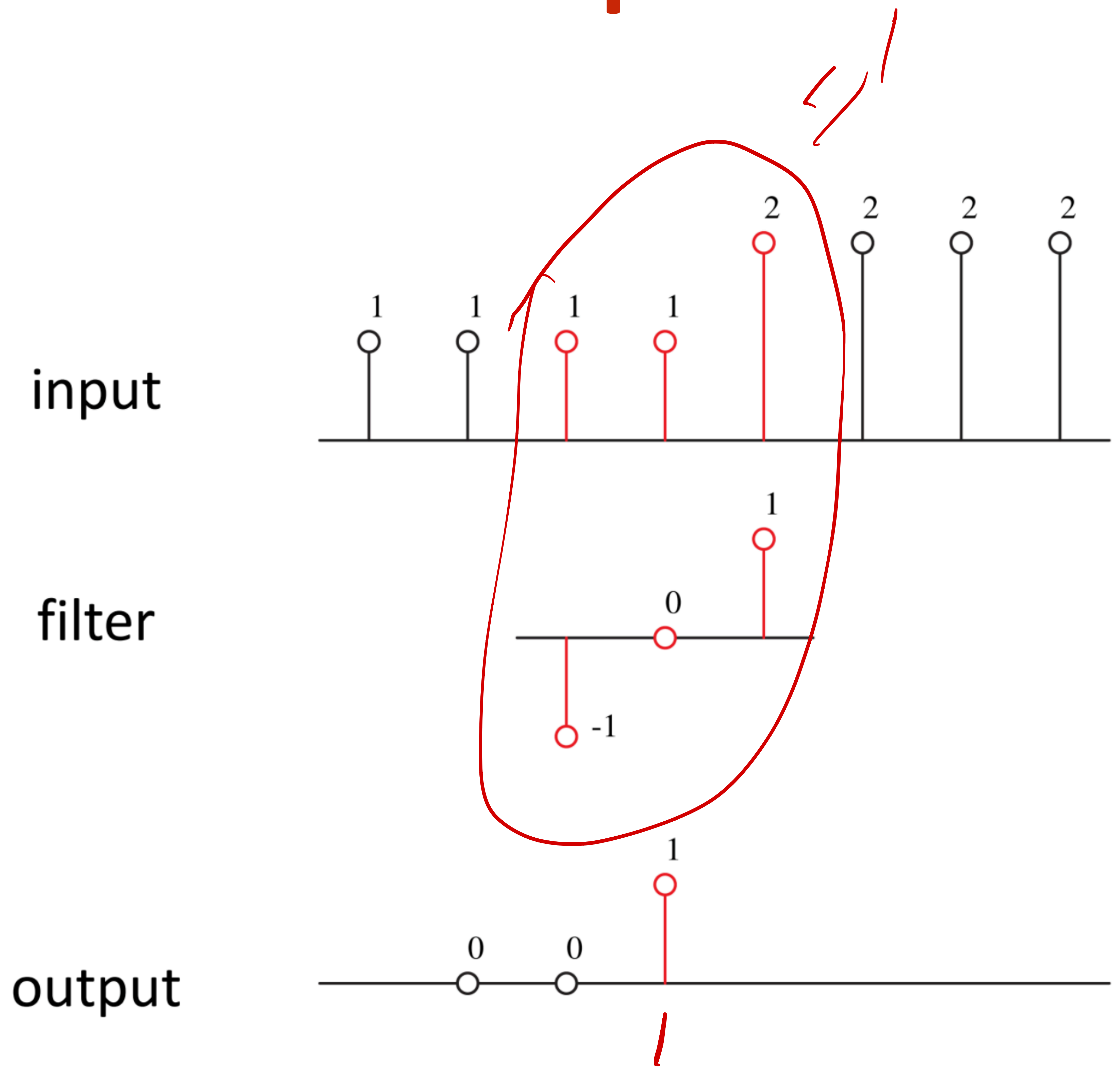
Convolution: a 1-D example

- sliding window
- dot product



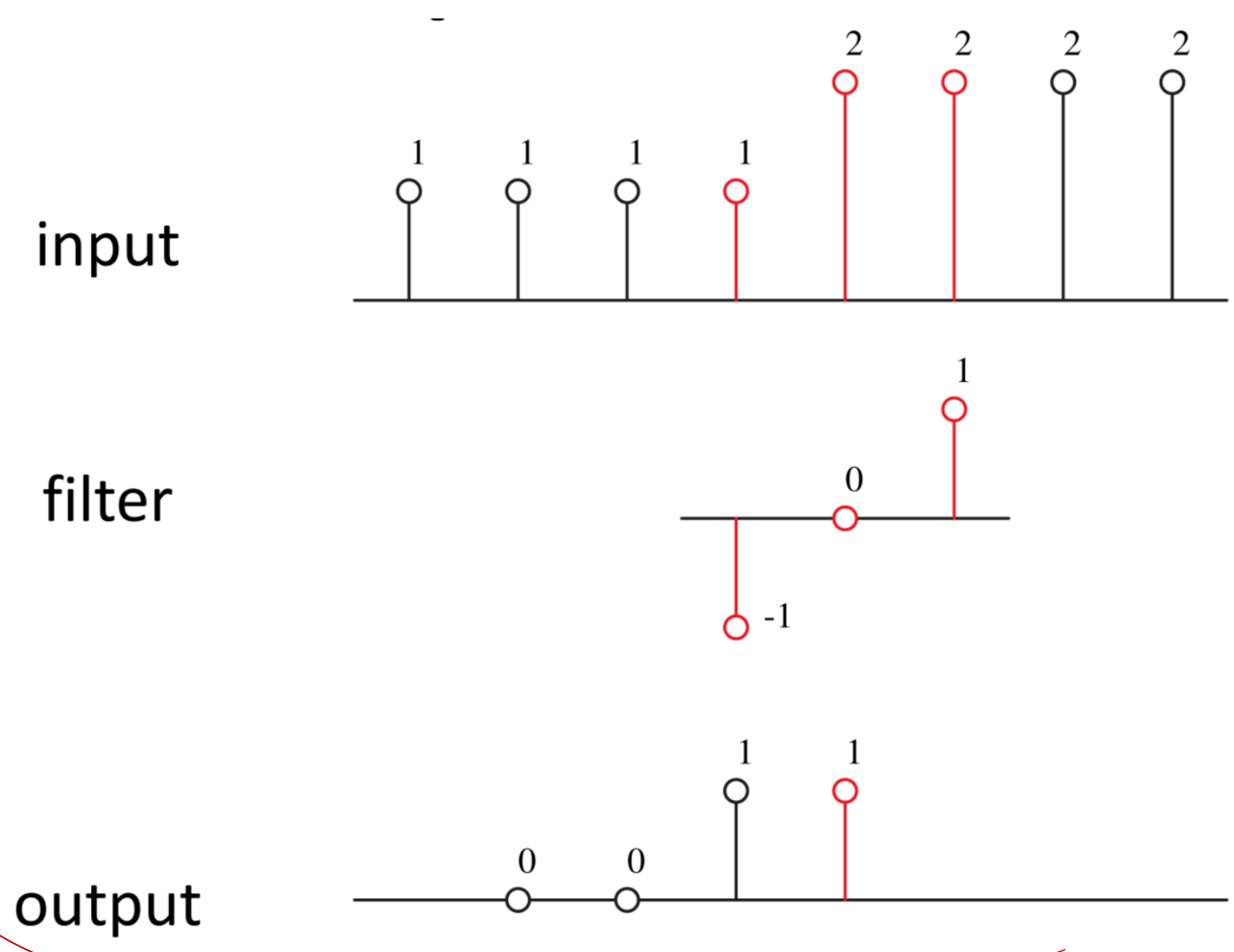
Convolution: a 1-D example

- sliding window
- dot product



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

0

$1 \times (-2) + 1 \times (-1) = -3$

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

-3					

- sliding window
- dot product

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	$^{0}\mathbf{1}$	$^{0}\mathbf{2}$	$^{0}\mathbf{1}$	0	0	0
0	0	$^{0}\mathbf{0}$	$^{0}\mathbf{0}$	$^{0}\mathbf{0}$	1	1	0
0	1	$^{1}\mathbf{-1}$	$^{1}\mathbf{-2}$	$^{1}\mathbf{-1}$	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

image

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

-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

- sliding window
- dot product

MLP

cat dog

house

image

Convolution: a 2-D example

filter weights

input map

coordinates in a local window

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

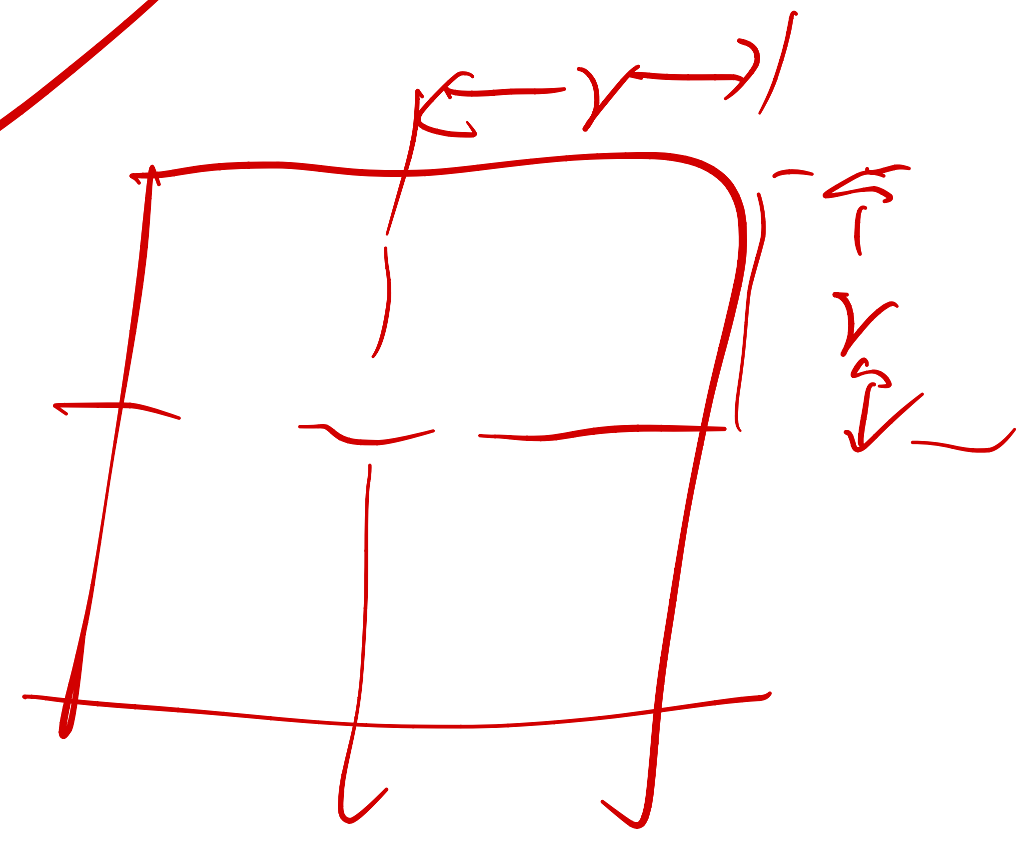
dot product

r

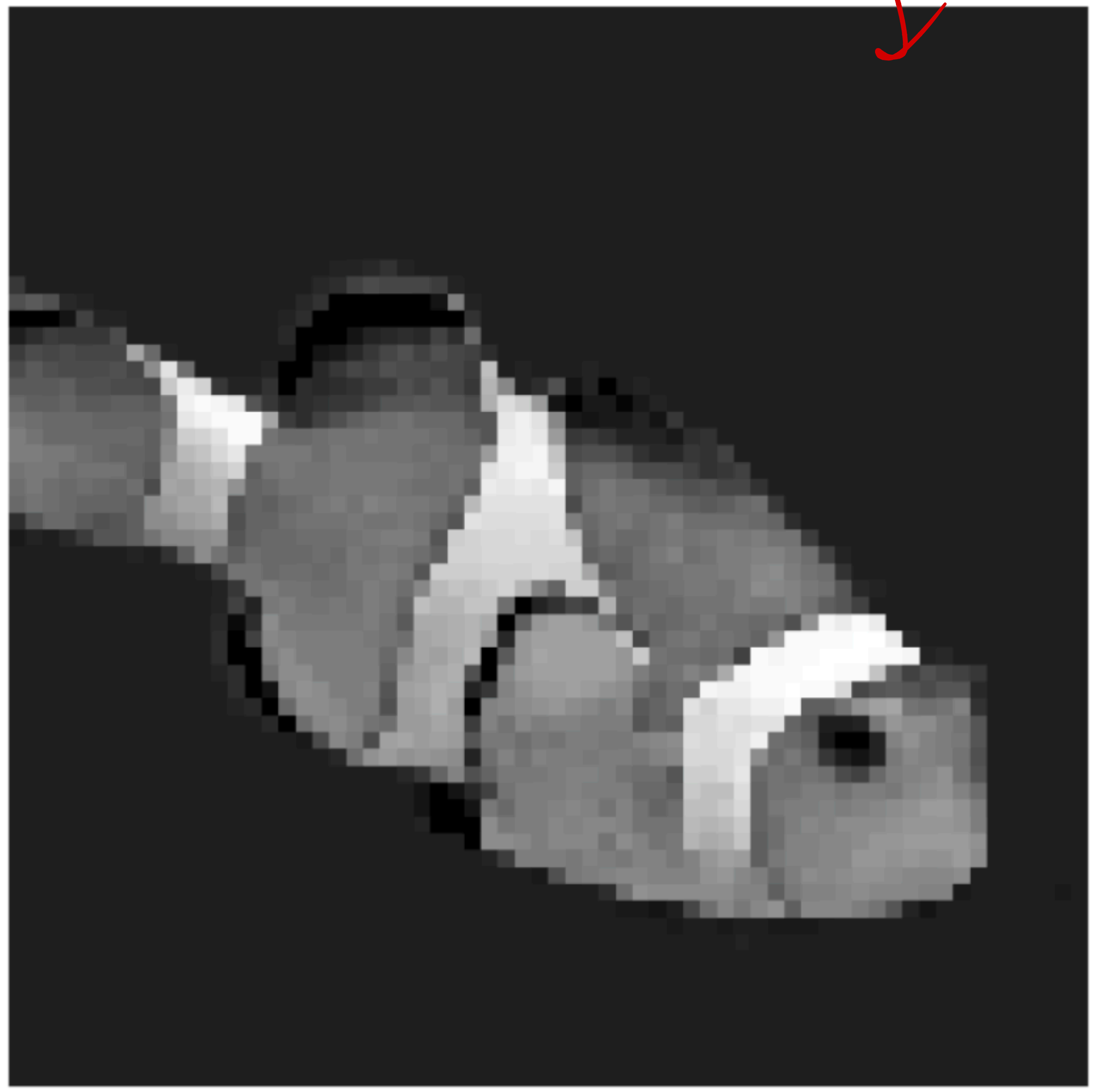
output map

r: kernel radius

kernel size = $2r + 1$



Convolution: 2-D



*

filter

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

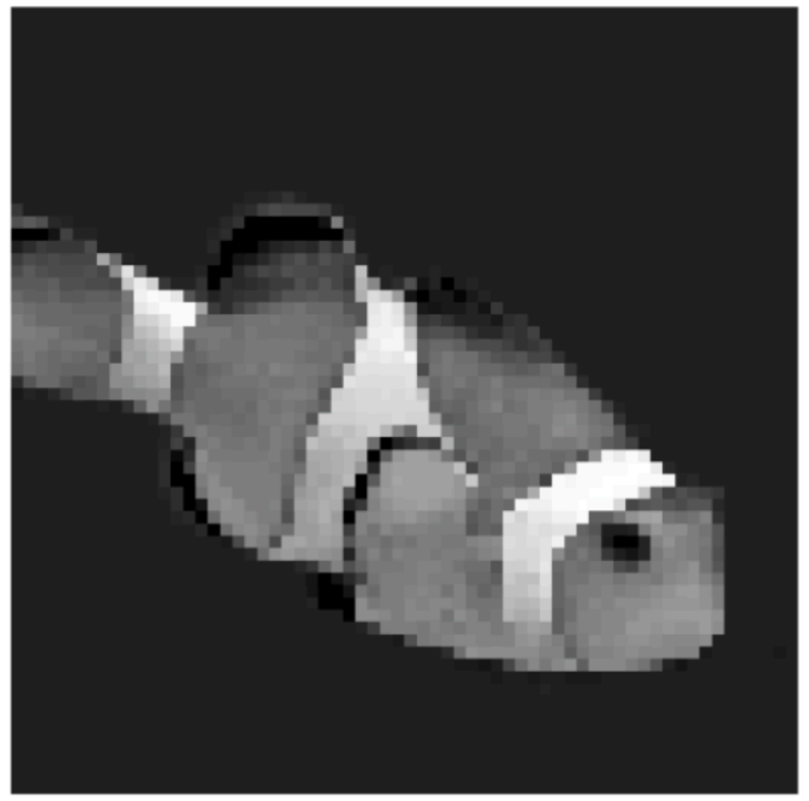
=

Convolution

Convolution: Multi-channel outputs

representation learning

feature extraction



*

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

=

one filter, one feature

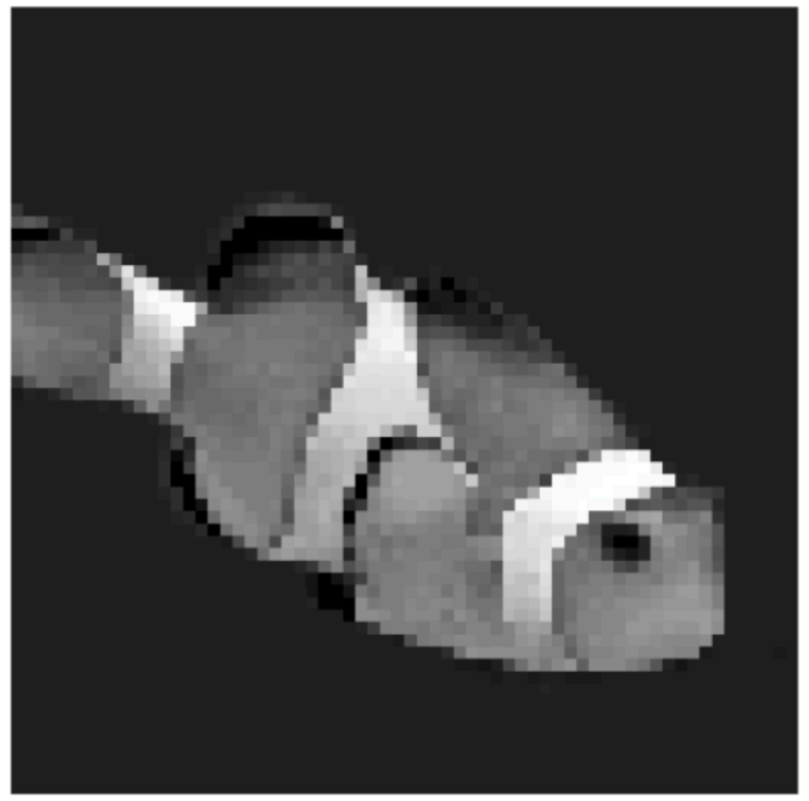
*

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

=

color change

Convolution: Multi-channel outputs



$$* \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} =$$

one filter, one feature

$$* \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} =$$

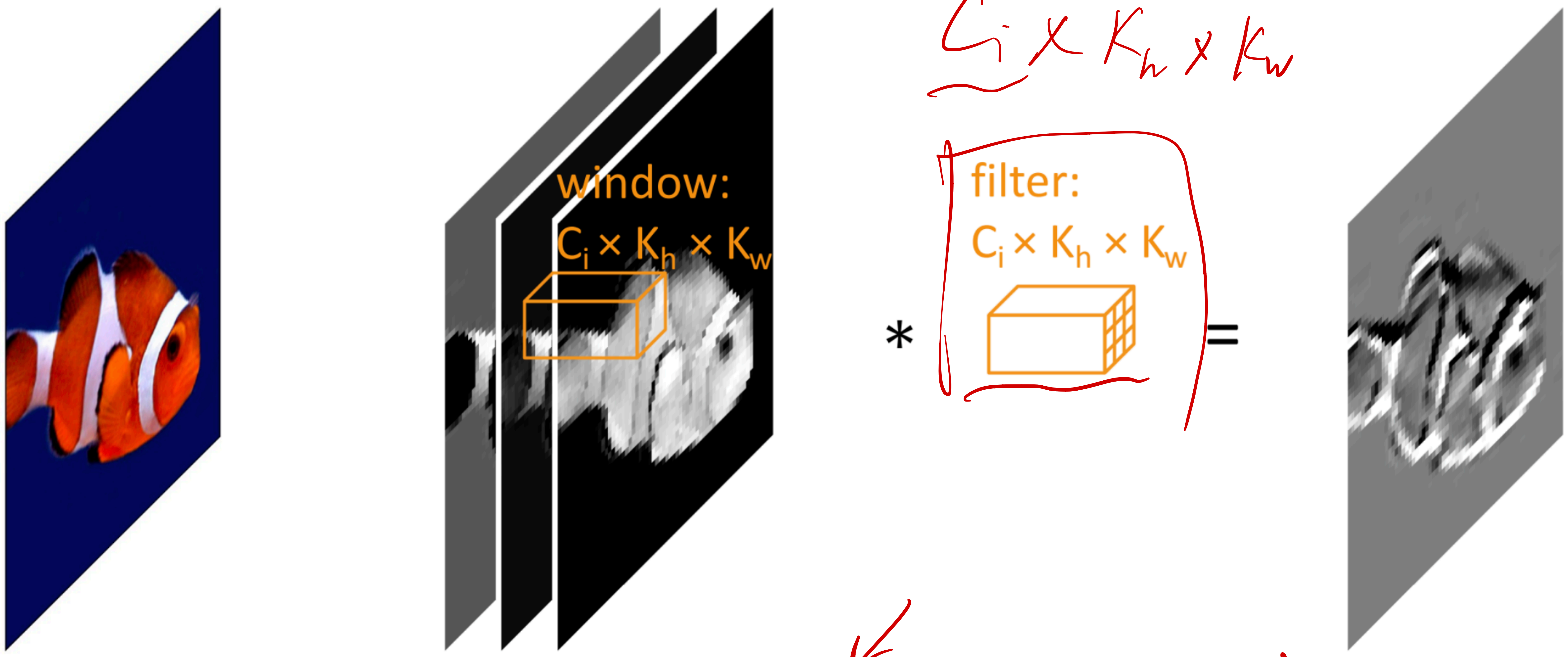
filter

Understanding the filter/kernel as feature extractors

Convolution: Multi-channel inputs

input channel size = 3

$C_i \times K_h \times K_w$

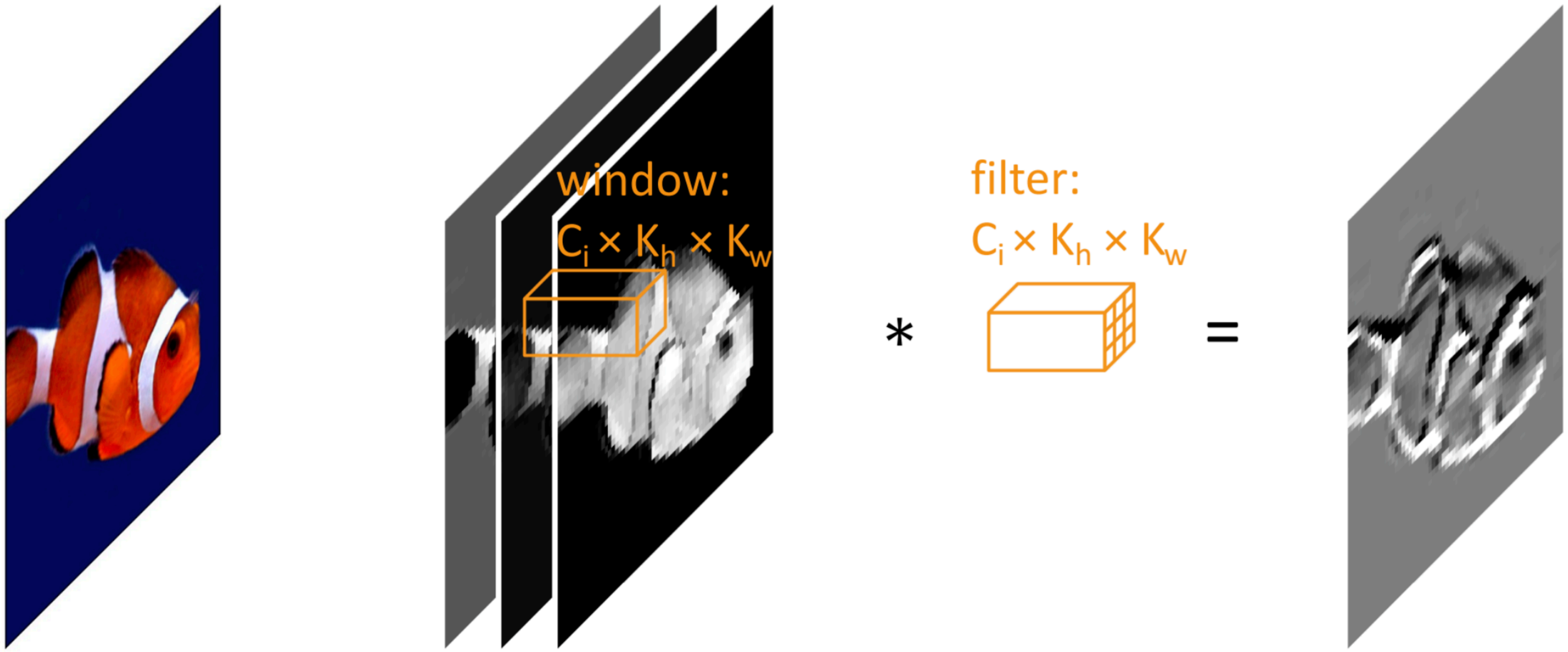


(R, G, B)

K_w

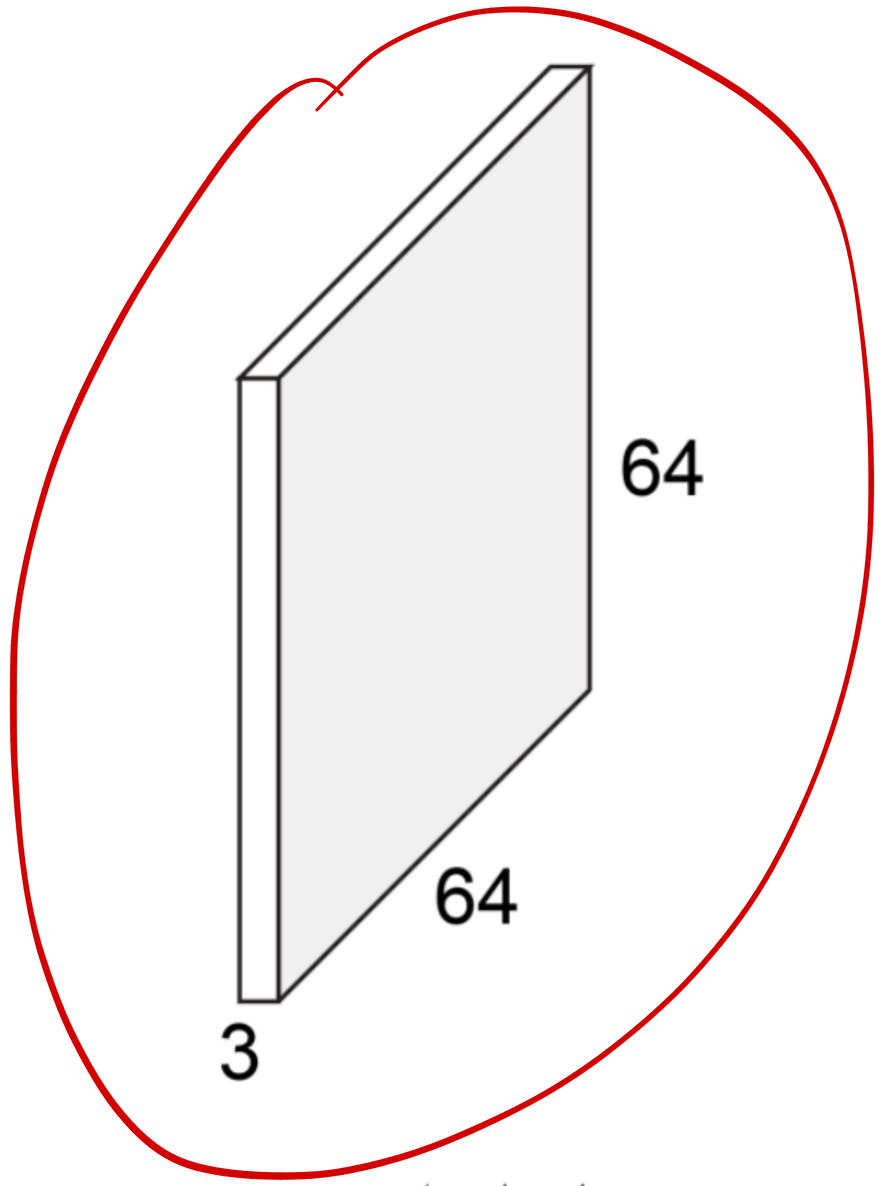
K_h

Convolution: Multi-channel inputs



Like (R, G, B) color notations have three features

Convolution: tensor views

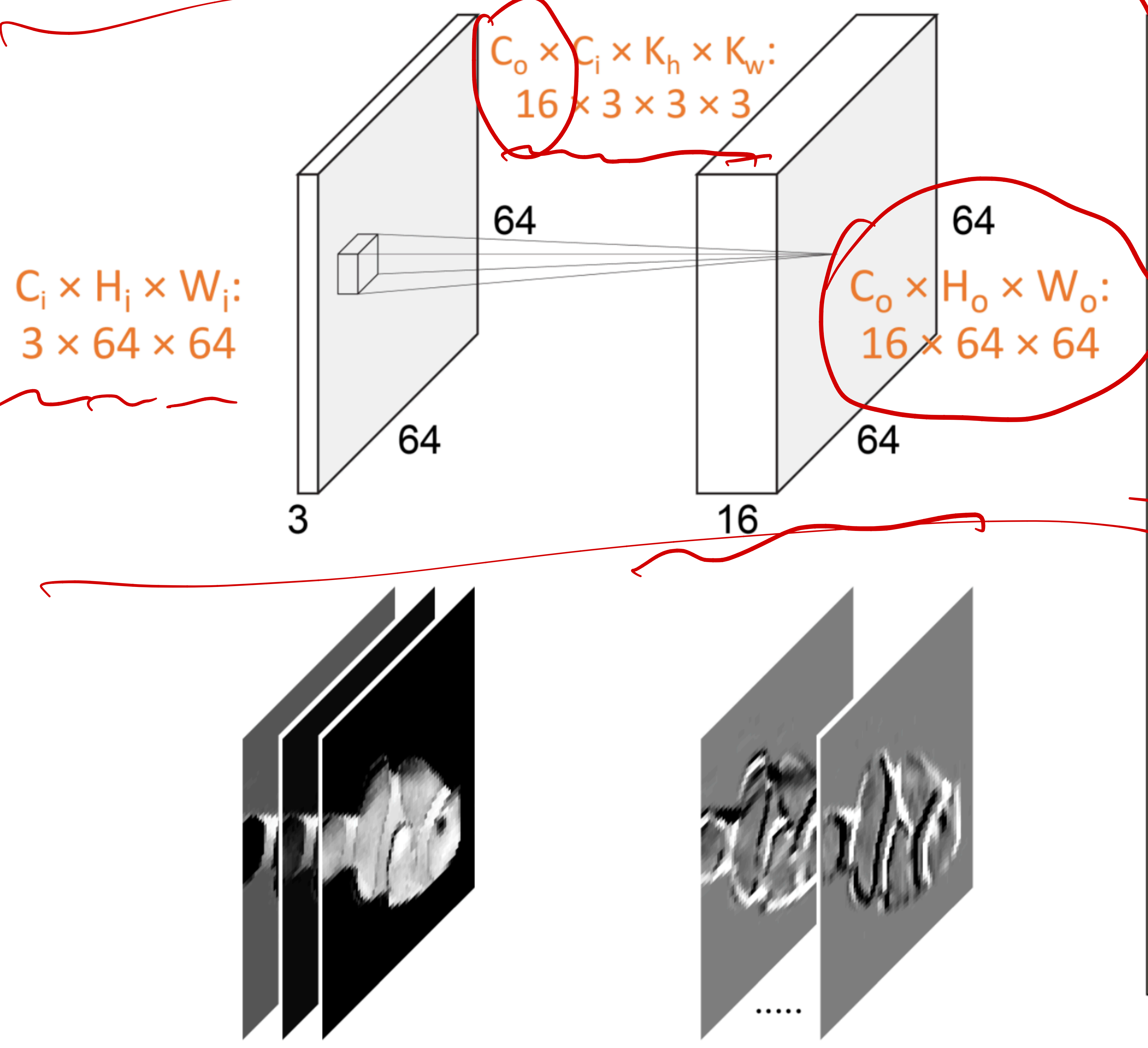


3×64×64

RGB

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

Convolution: tensor view

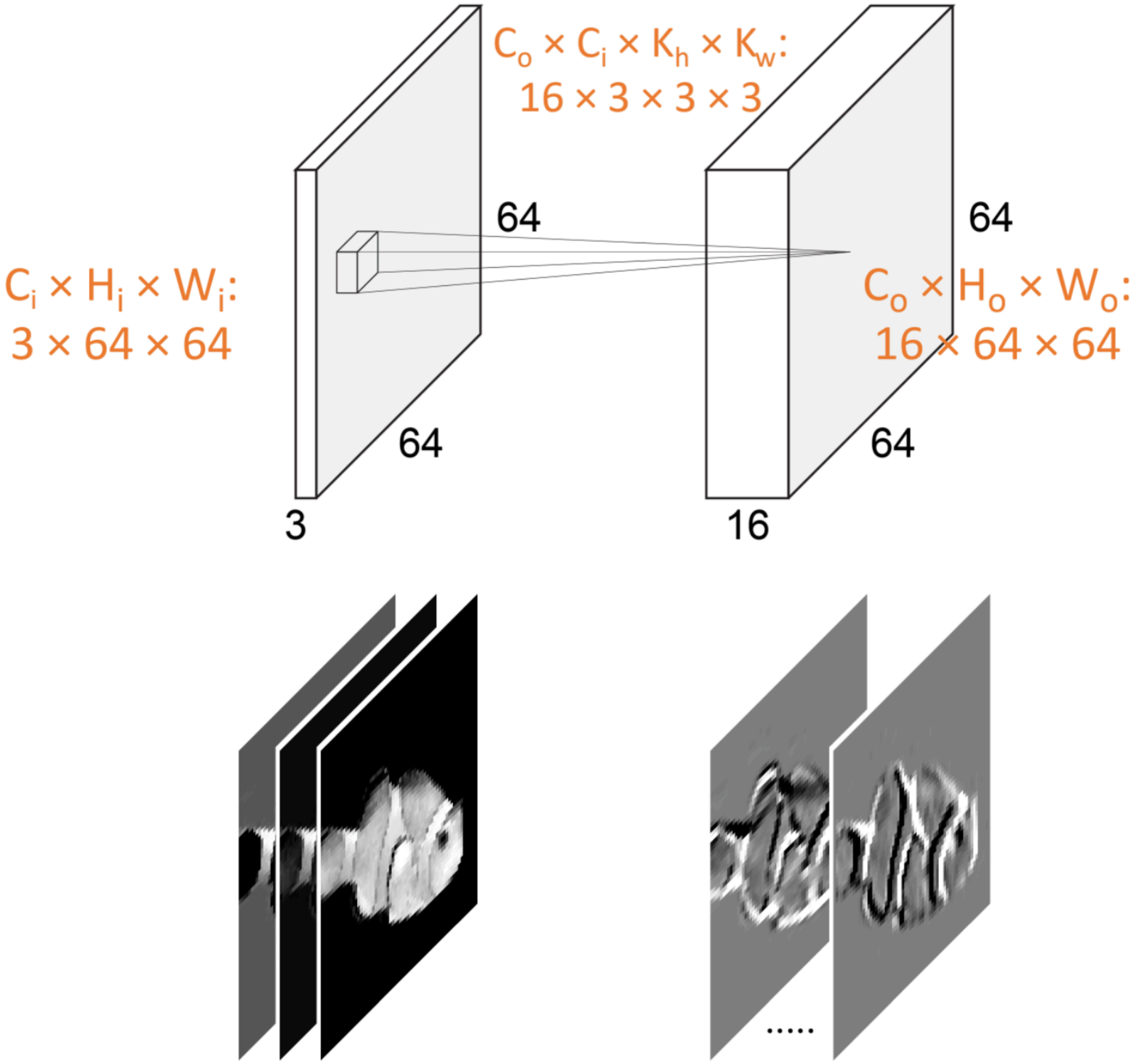


16

- 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

C_o

Convolution: tensor view



- 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

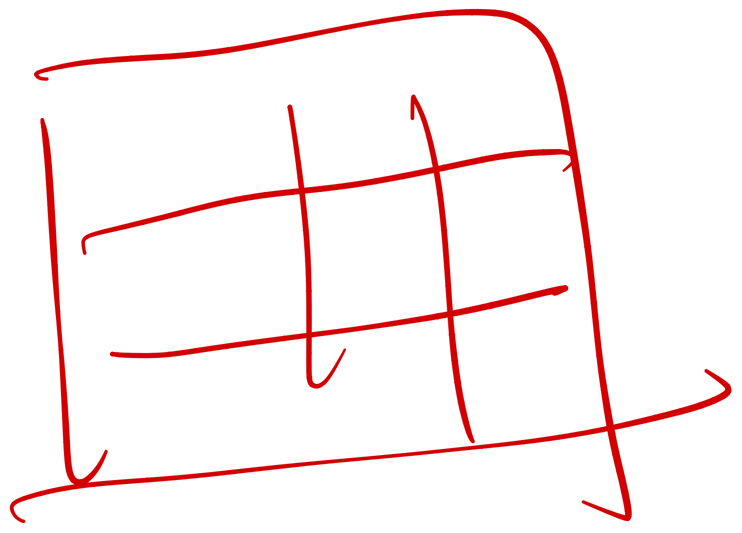
HMM

stationary

The same filter tensor applies to different locations

Convolution: # parameters and # operations

Kernel



only parameter

Convolution: # parameters and # operations

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

$wx + \boxed{b}$

bias

Convolution: # parameters and # operations

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

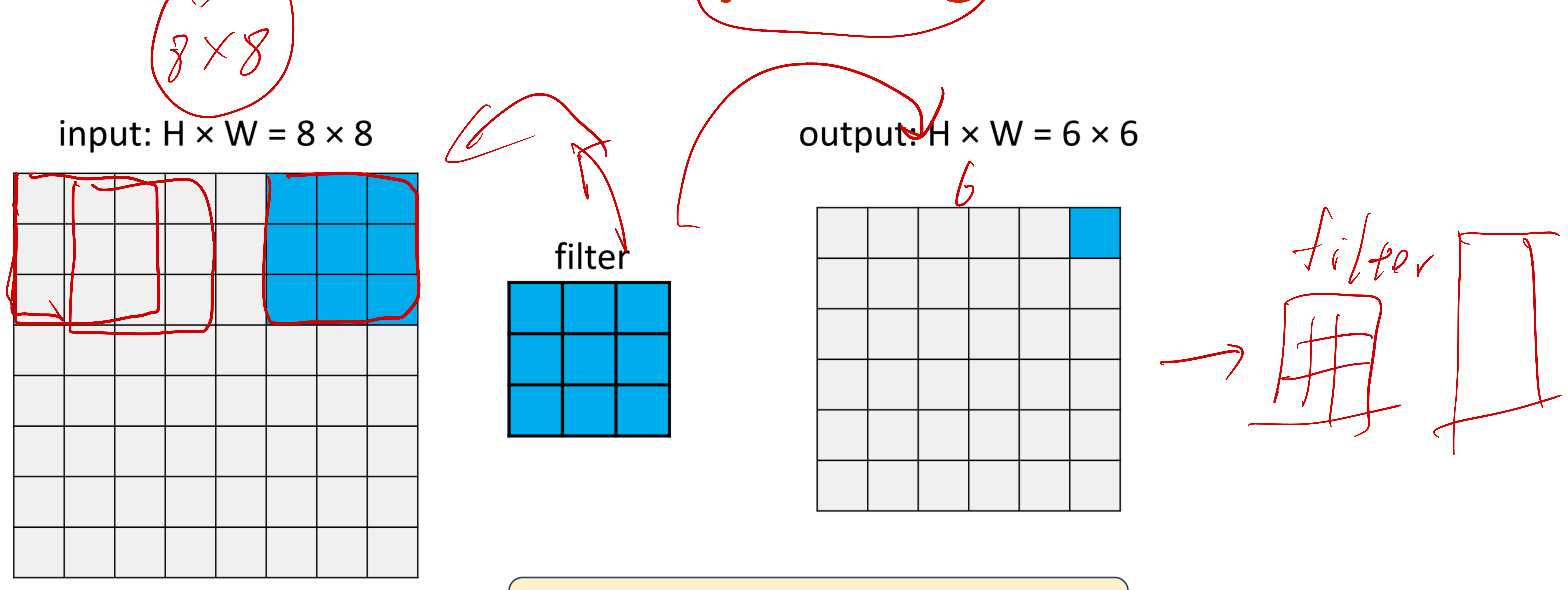
- # floating-point operations (FLOPs)
  - # params $\times H_o \times W_o$

$O(\boxed{H_o \times W_o} \times C_o \times C_i \times K_h \times K_w)$

dot product : $O(C_o \times C_i \times K_h \times K_w)$

dot product ?

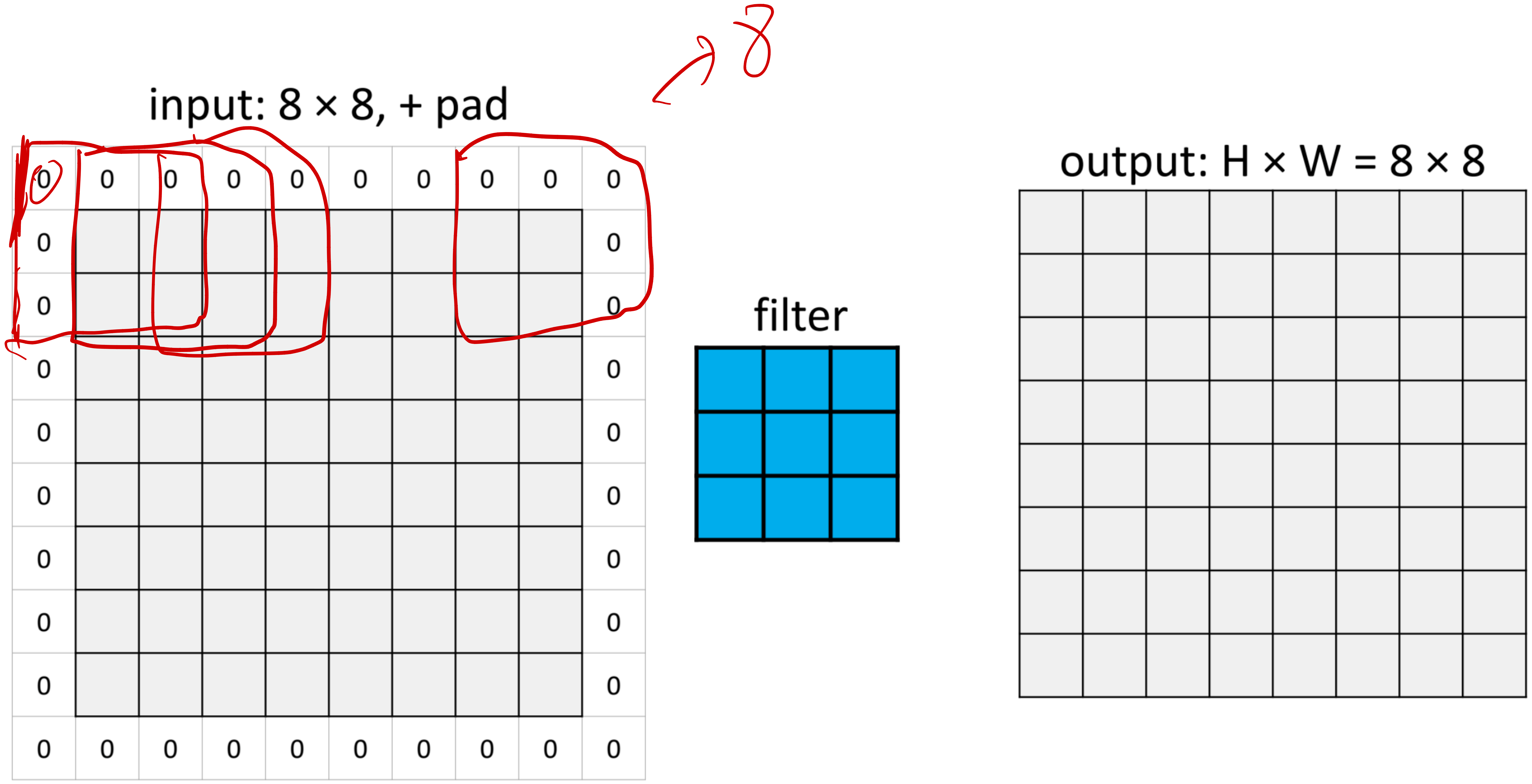
Convolution: padding



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

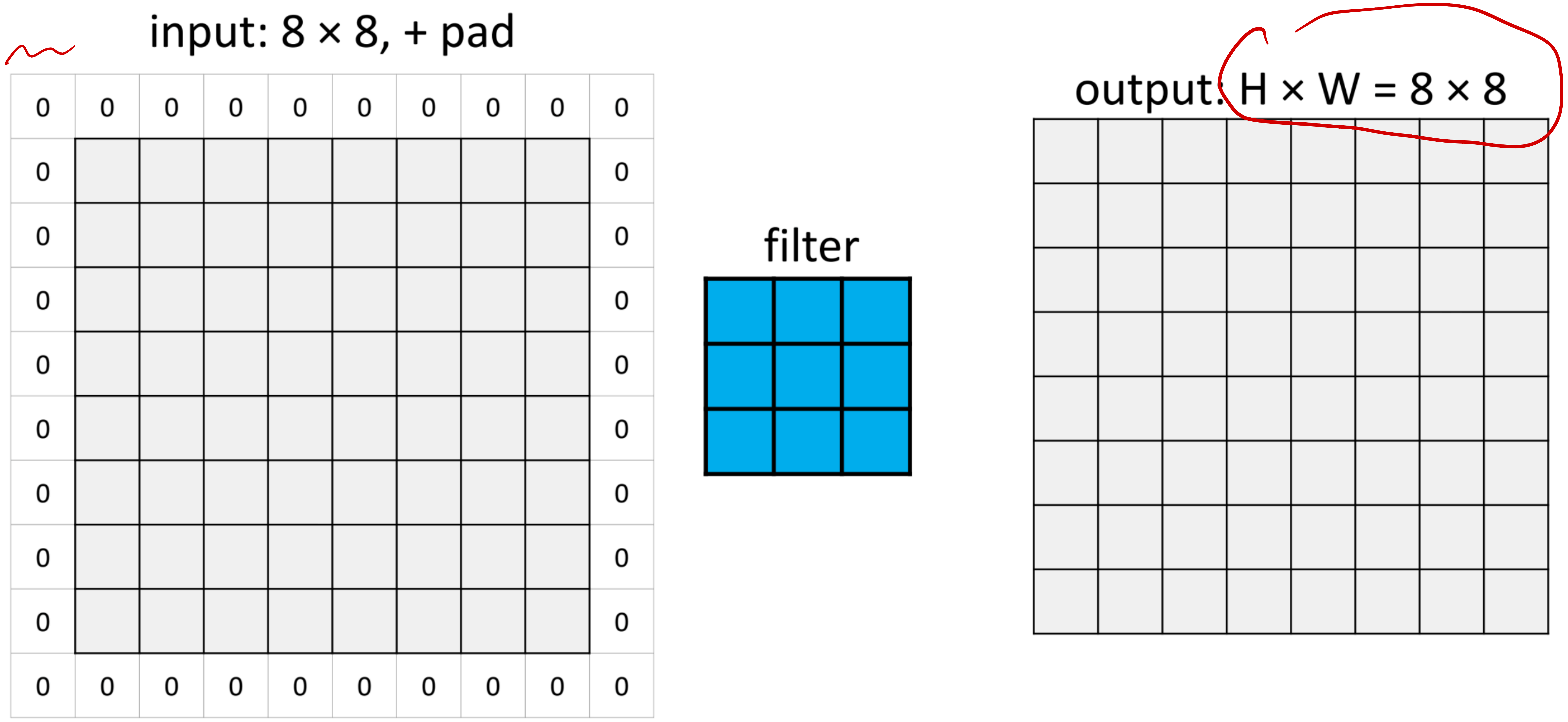
$8-3+1=6$

Convolution: padding



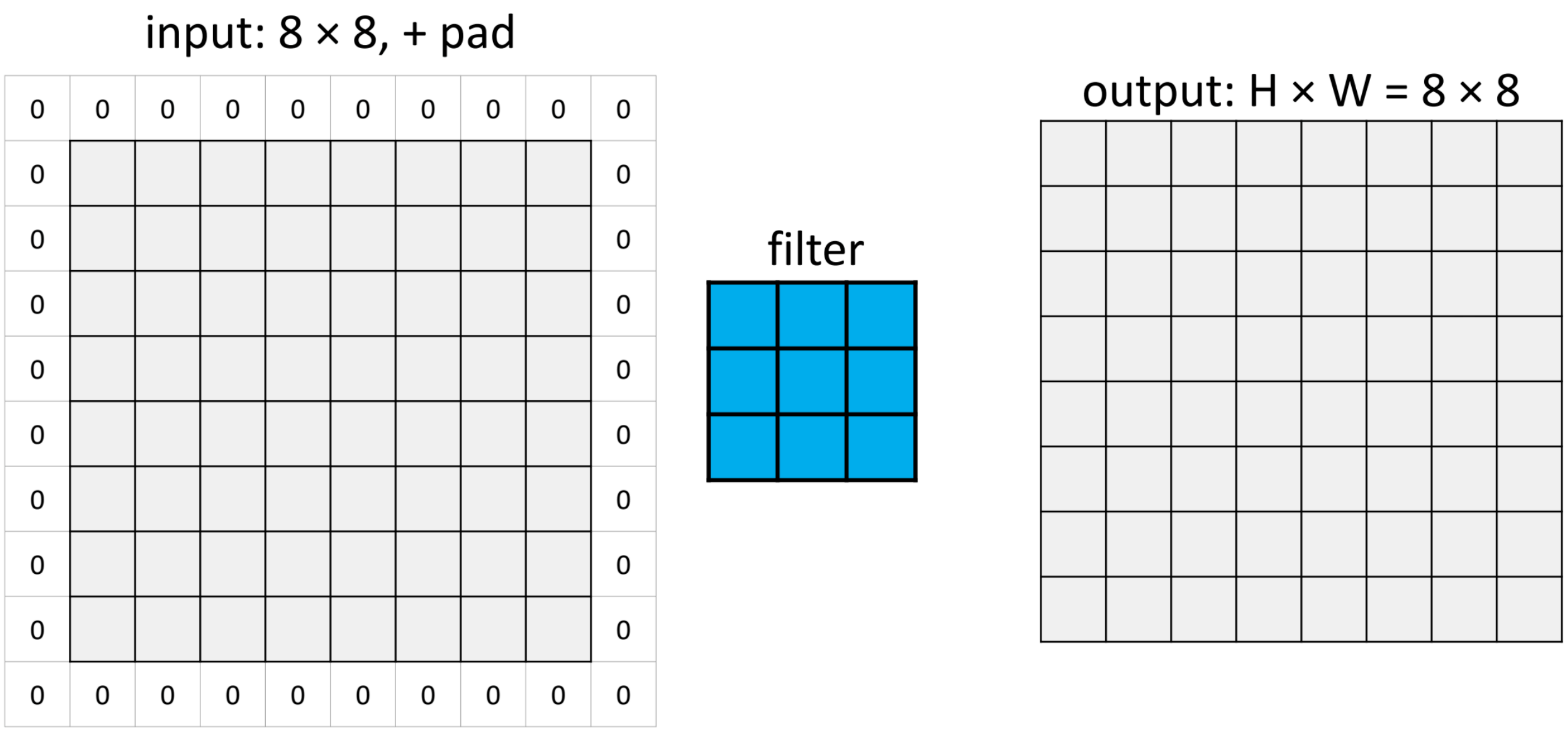
Convolution: padding

pad size



- pad = $\lfloor$kernel_size / 2$\rfloor$
- maintains feature map size

Convolution: padding

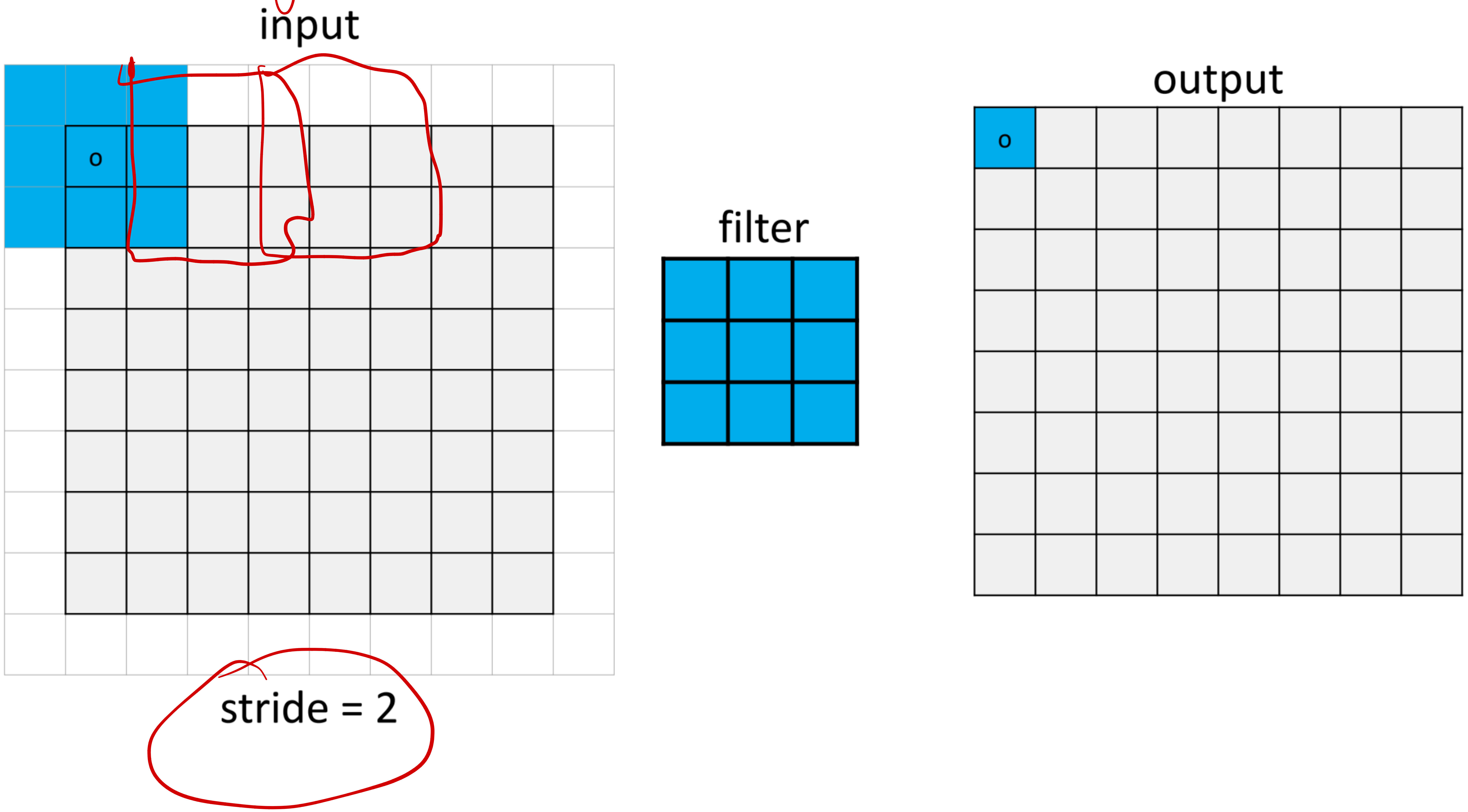


- pad = $\lfloor \text{kernel\_size} / 2 \rfloor$
- maintains feature map size

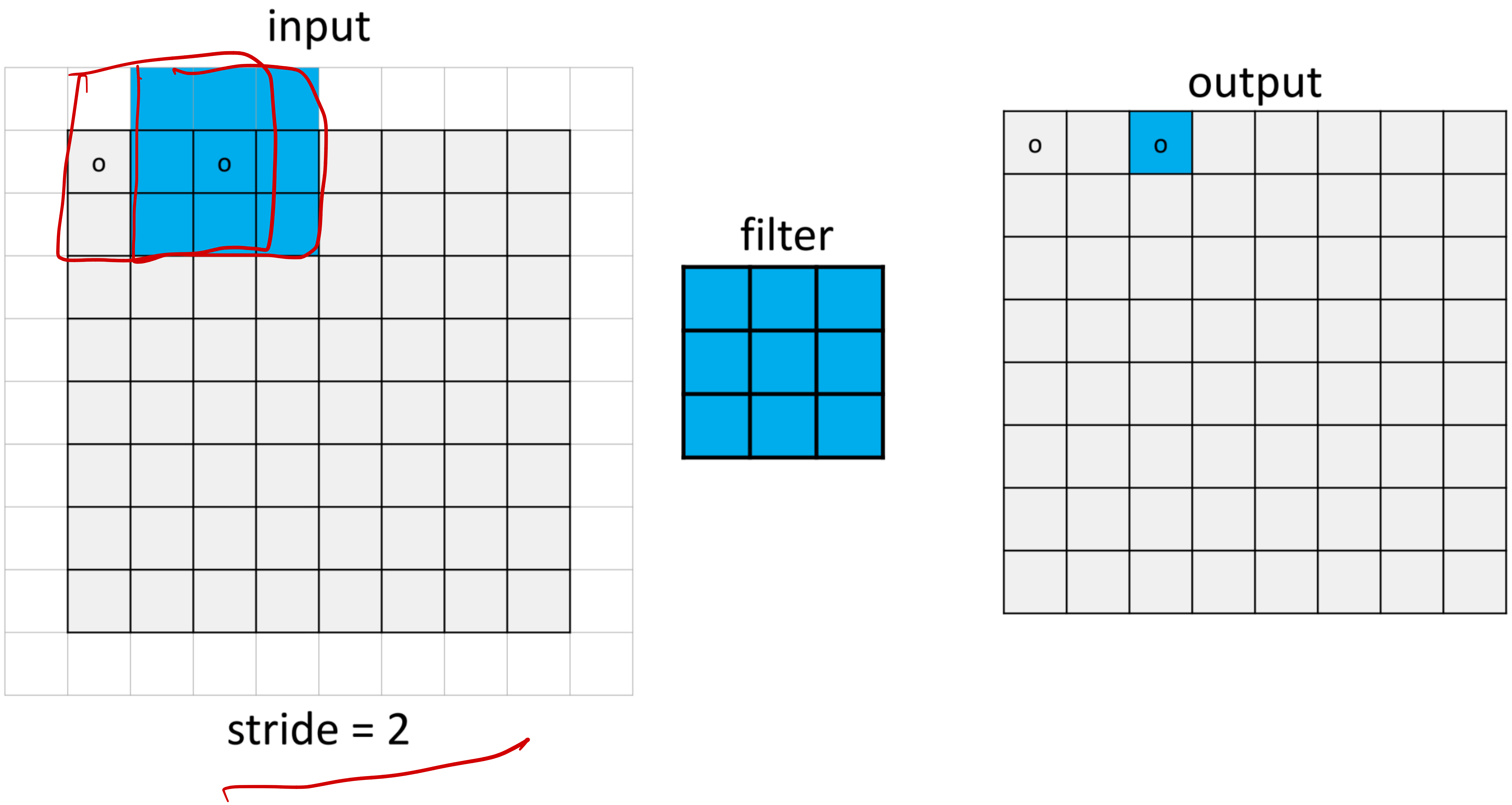
$$H_{out} = H_{in} + 2pad_h - K_h + 1$$

Convolution: stride

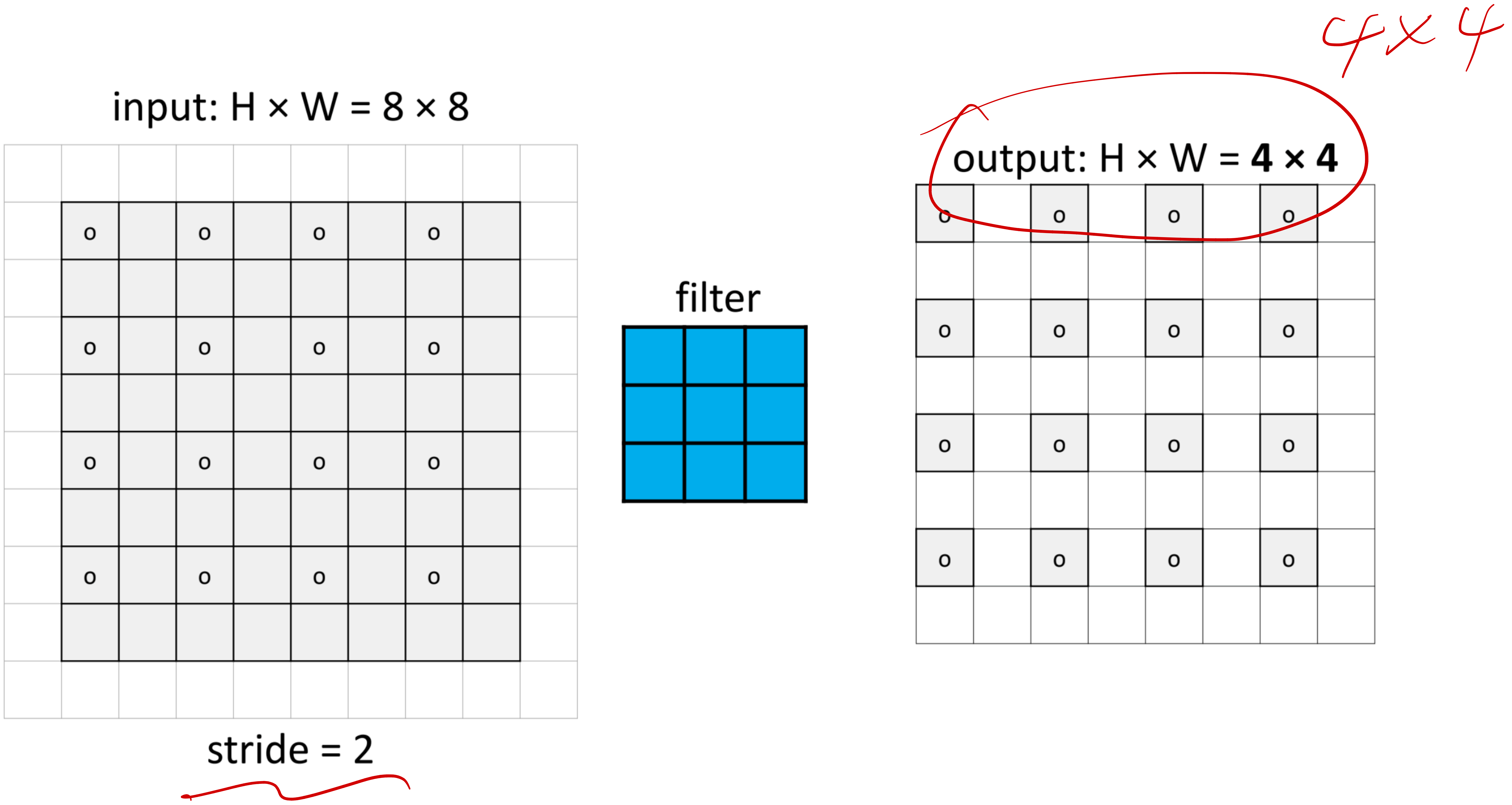
sliding step size



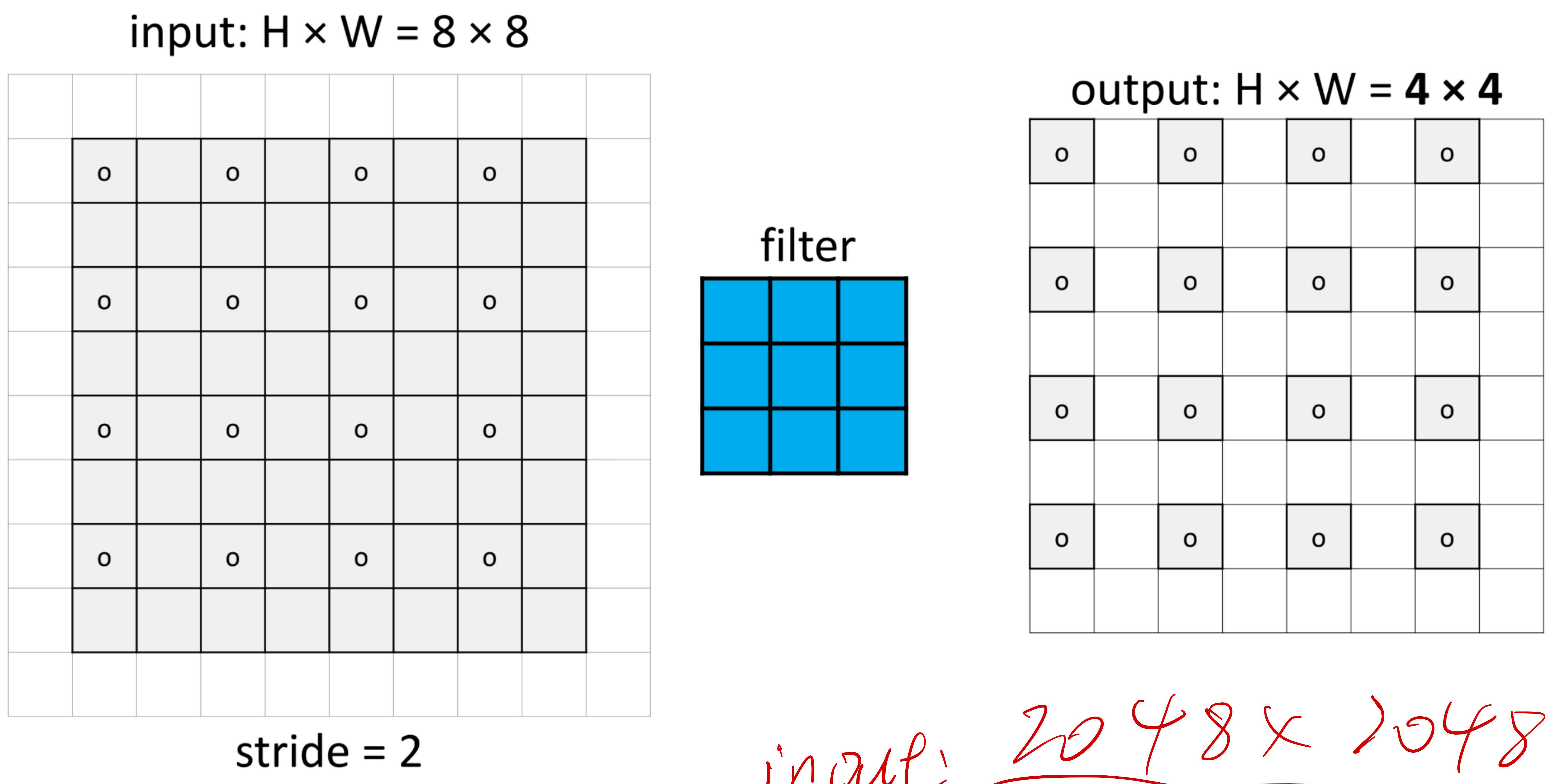
Convolution: stride



Convolution: stride



Convolution: stride

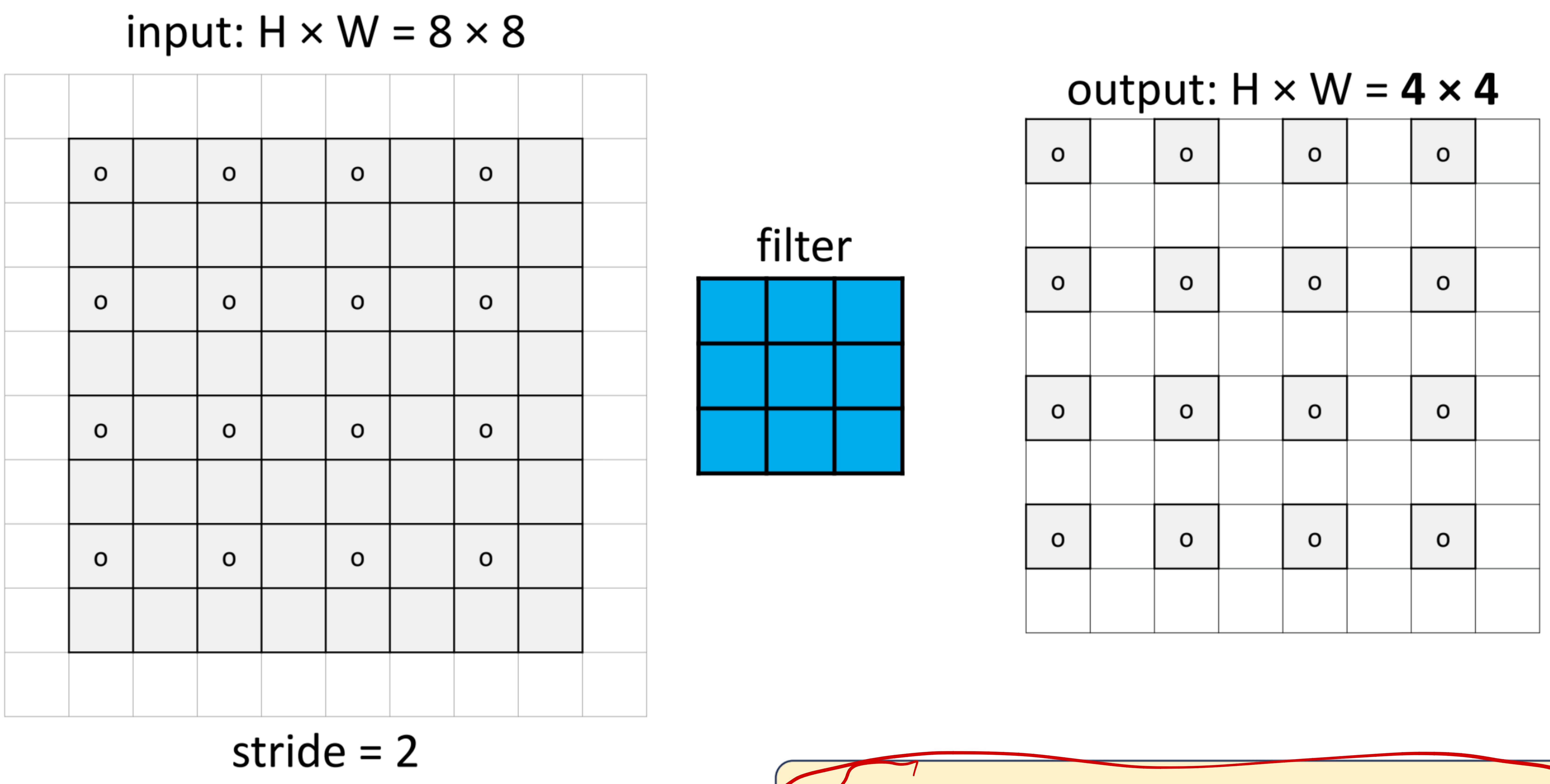


- reduces feature map size
- compress and abstract

input: 2048 × 2048

512 × 512

Convolution: stride

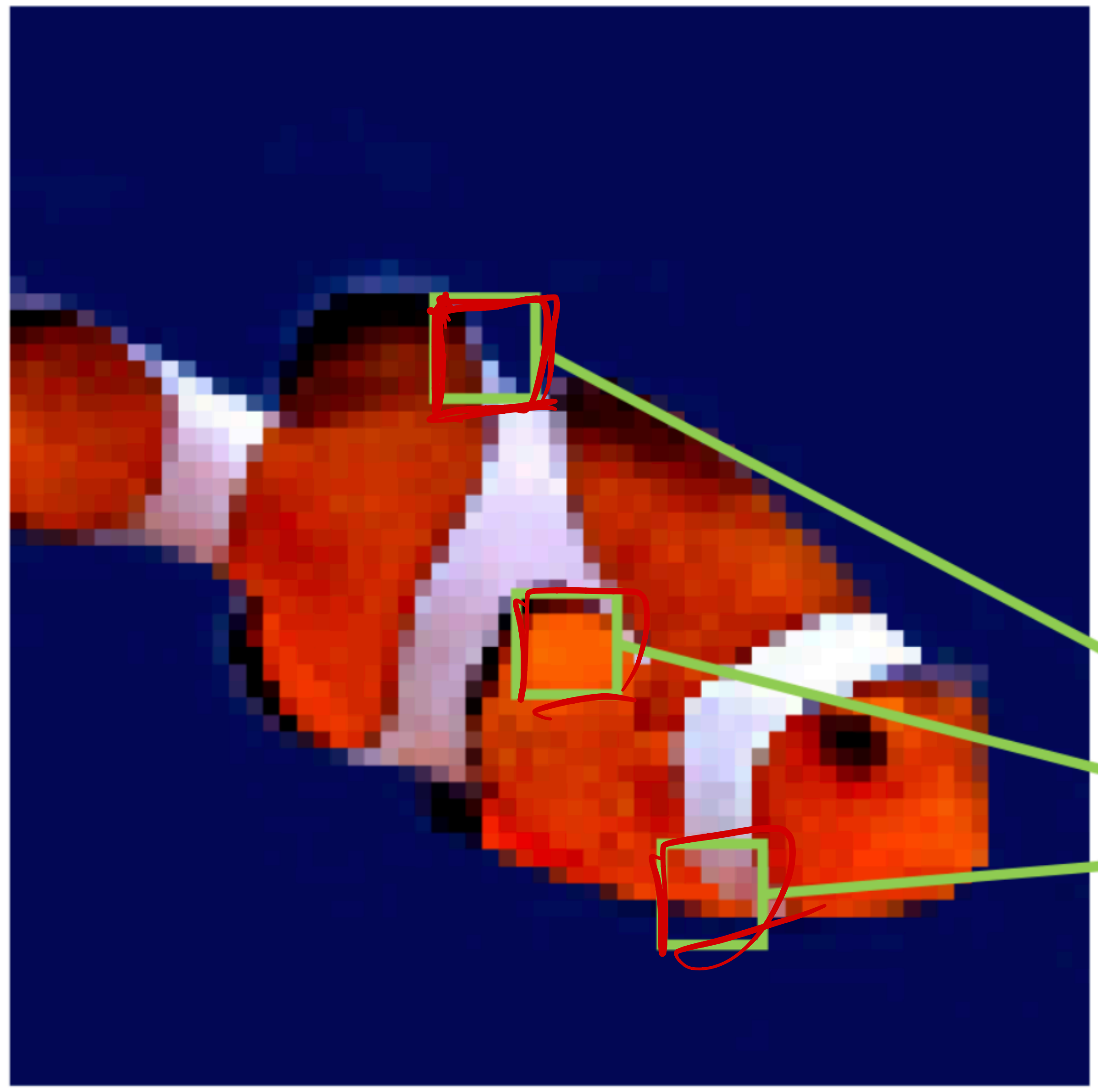


- reduces feature map size
- compress and abstract

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

Convolution: translation-invariance

- Process each window in the same way



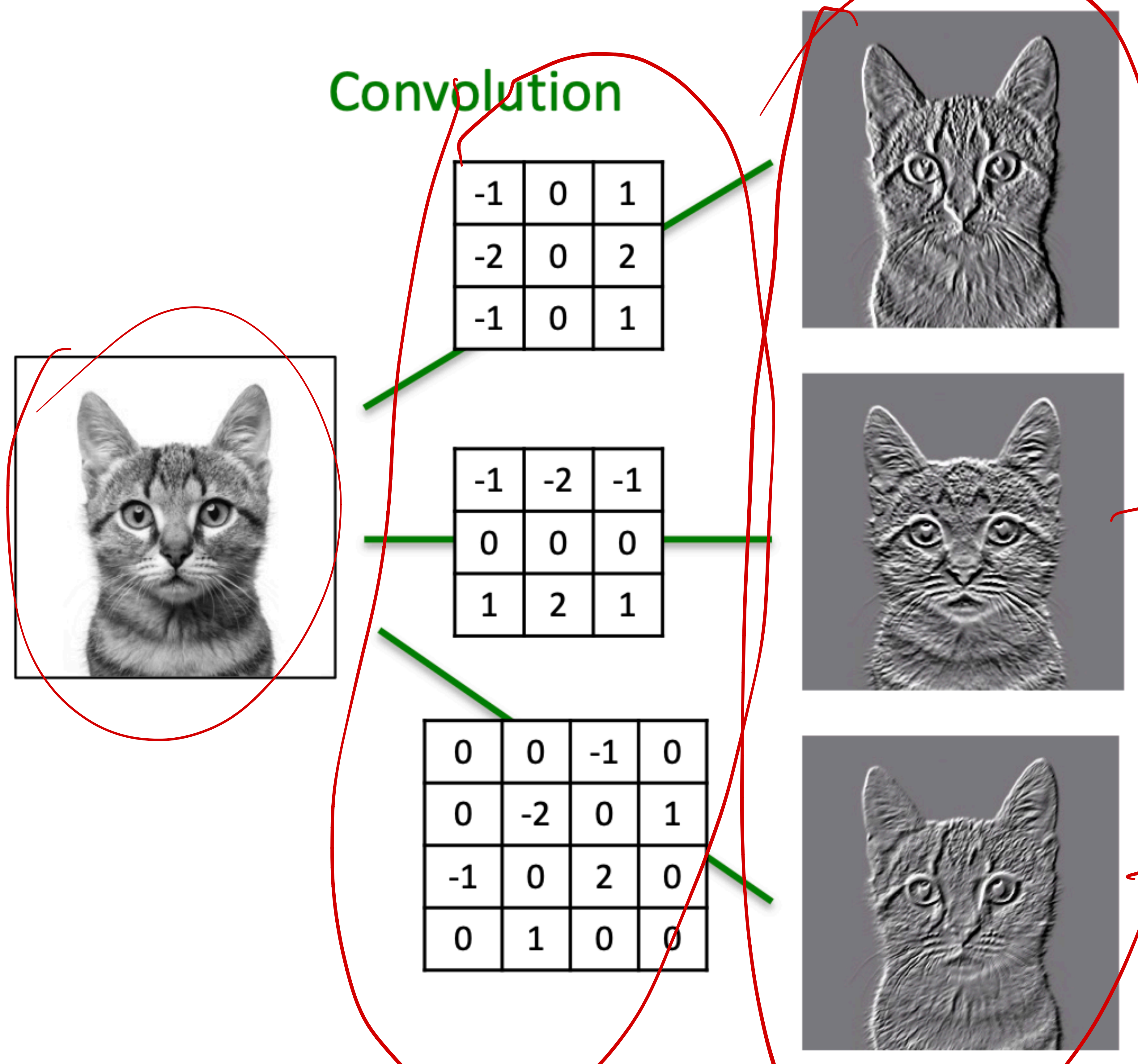
apply the same weights regardless of the window location

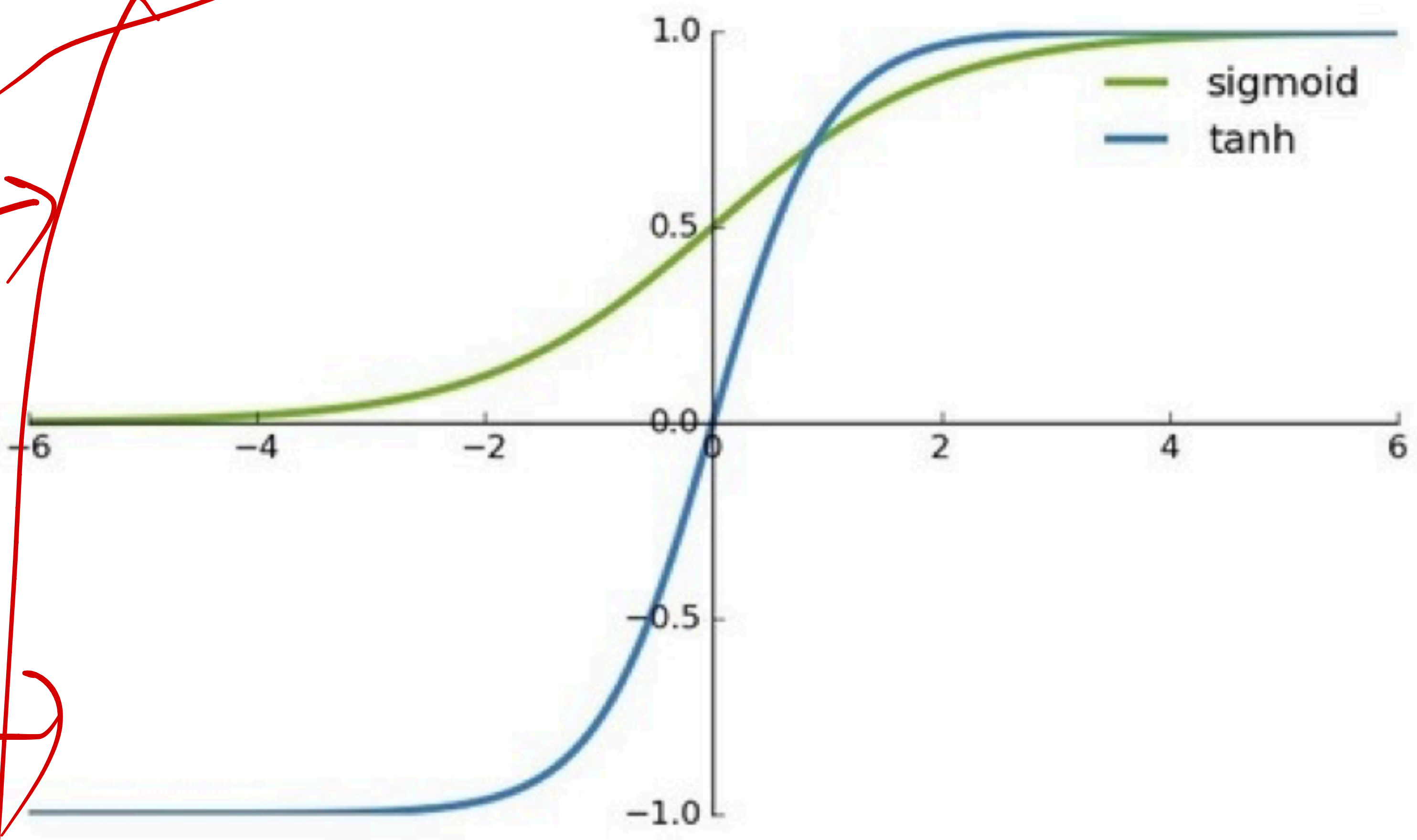
MLP

$wx + b$

Deep Convolutional Networks

[Convolution + Nonlinear activation] + Pooling



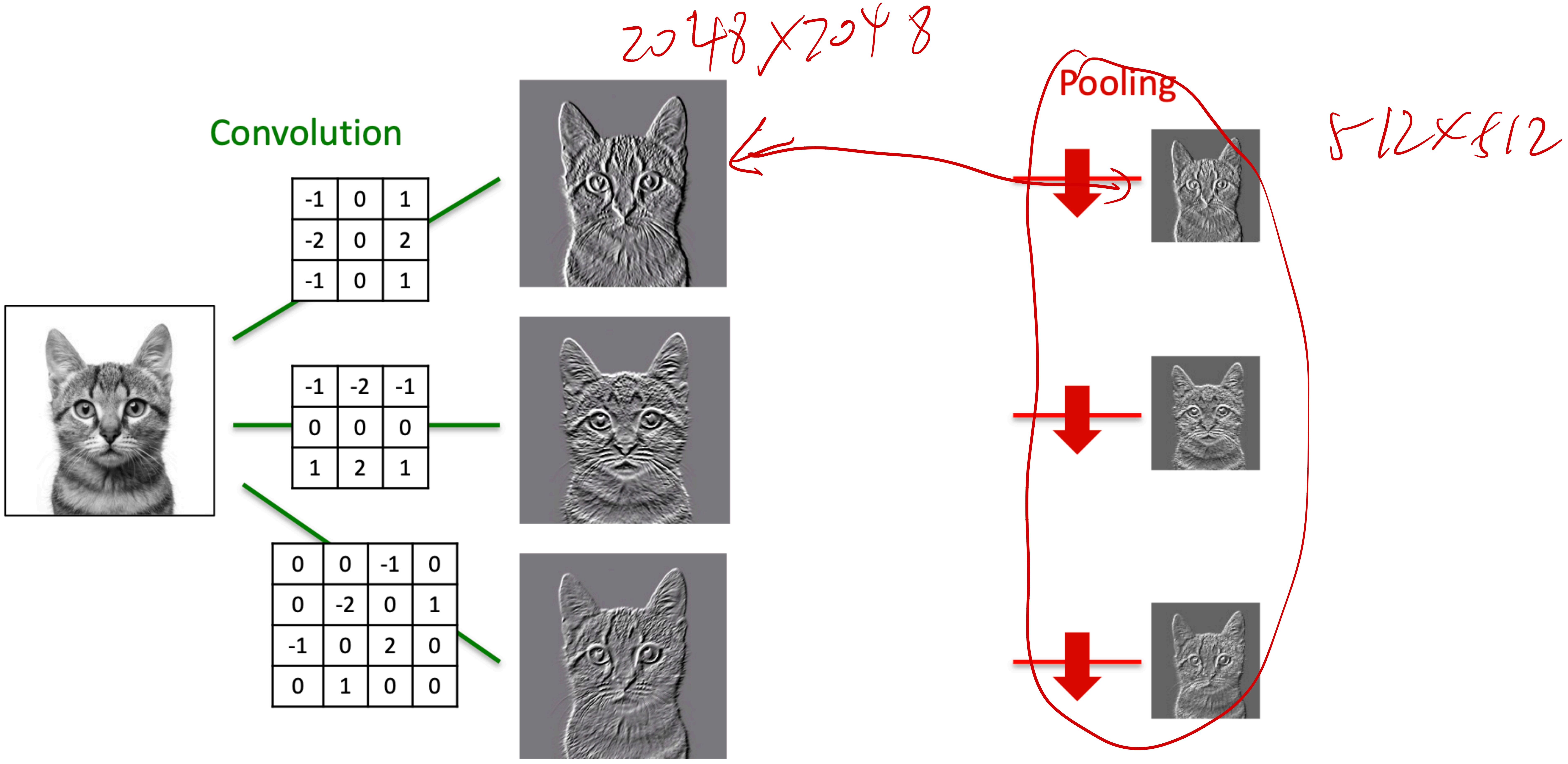


LeNet – tanh activation

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Deep Convolutional Networks

[Convolution + Nonlinear activation] + Pooling



1 do ~~not~~ like a cat

Pooling = Down-sampling

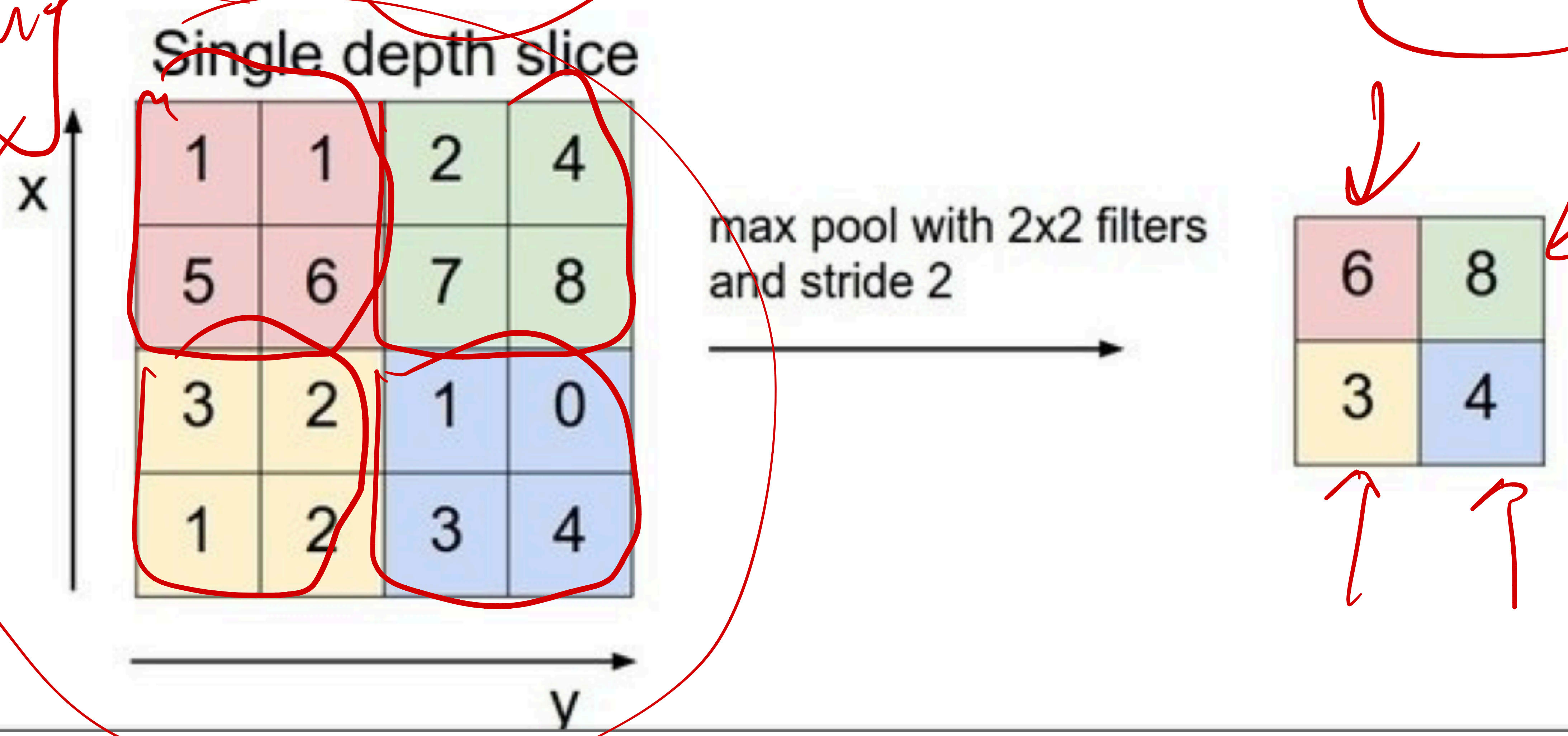
Max pooling

mean pooling

4x4

2x2

max



2x2

1024 x 1024

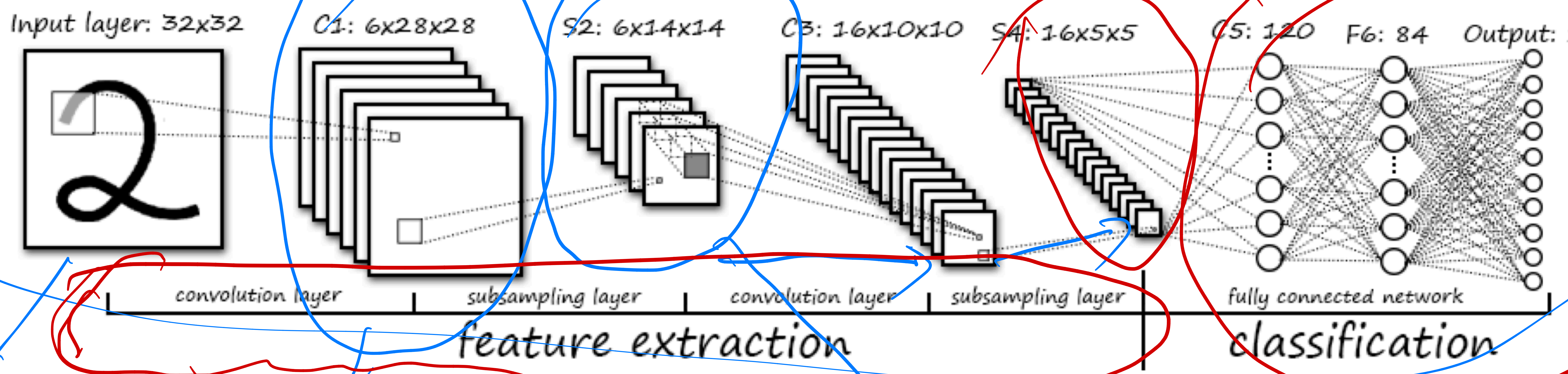
512 x 512

mean pooling:

6.5

Deep Convolutional Networks

2-layer MLP



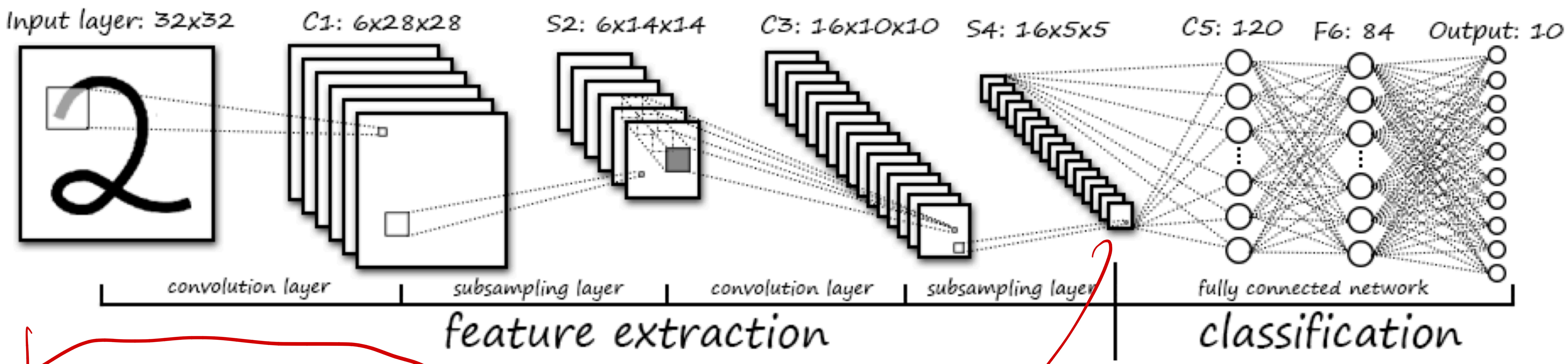
32x32 6x28x28

6x14x14

output channel size

[1] LeNet 5, LeCun et al. 1998

Deep Convolutional Networks



MNIST

60,000 original dataset

Test error: 0.95%

[1] LeNet 5, LeCun et al. 1998

Misclassified examples on MNIST

True label -> Predicted label

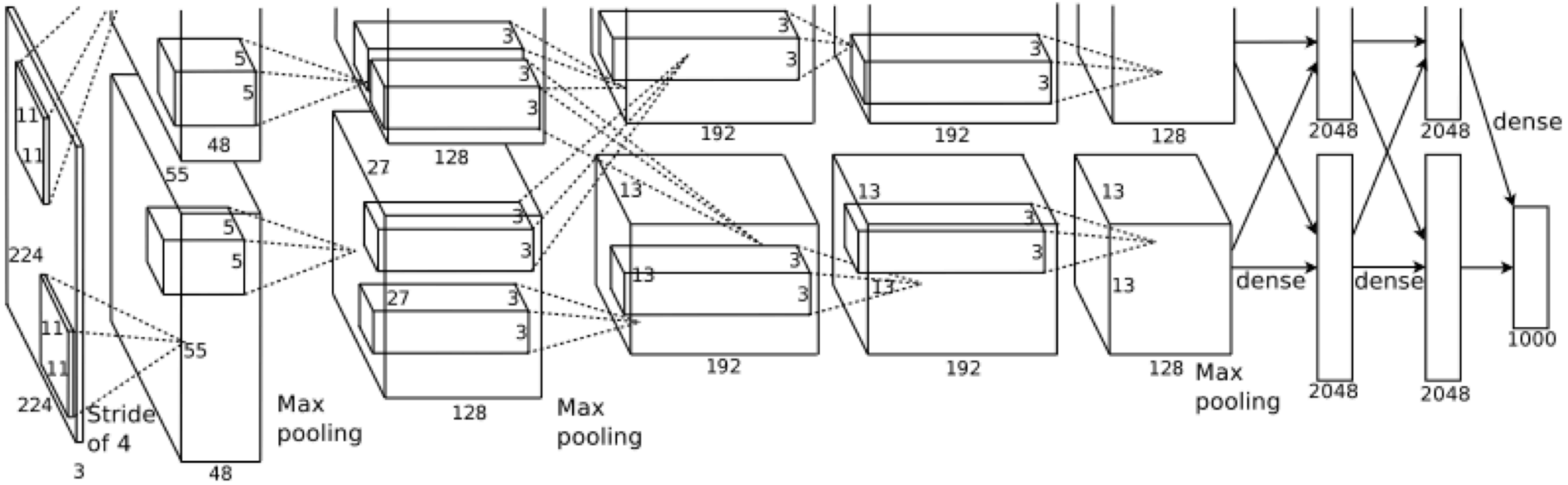
2->1

9->5

4->9

Alex Net

milestone



[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NeurIPS 2012.

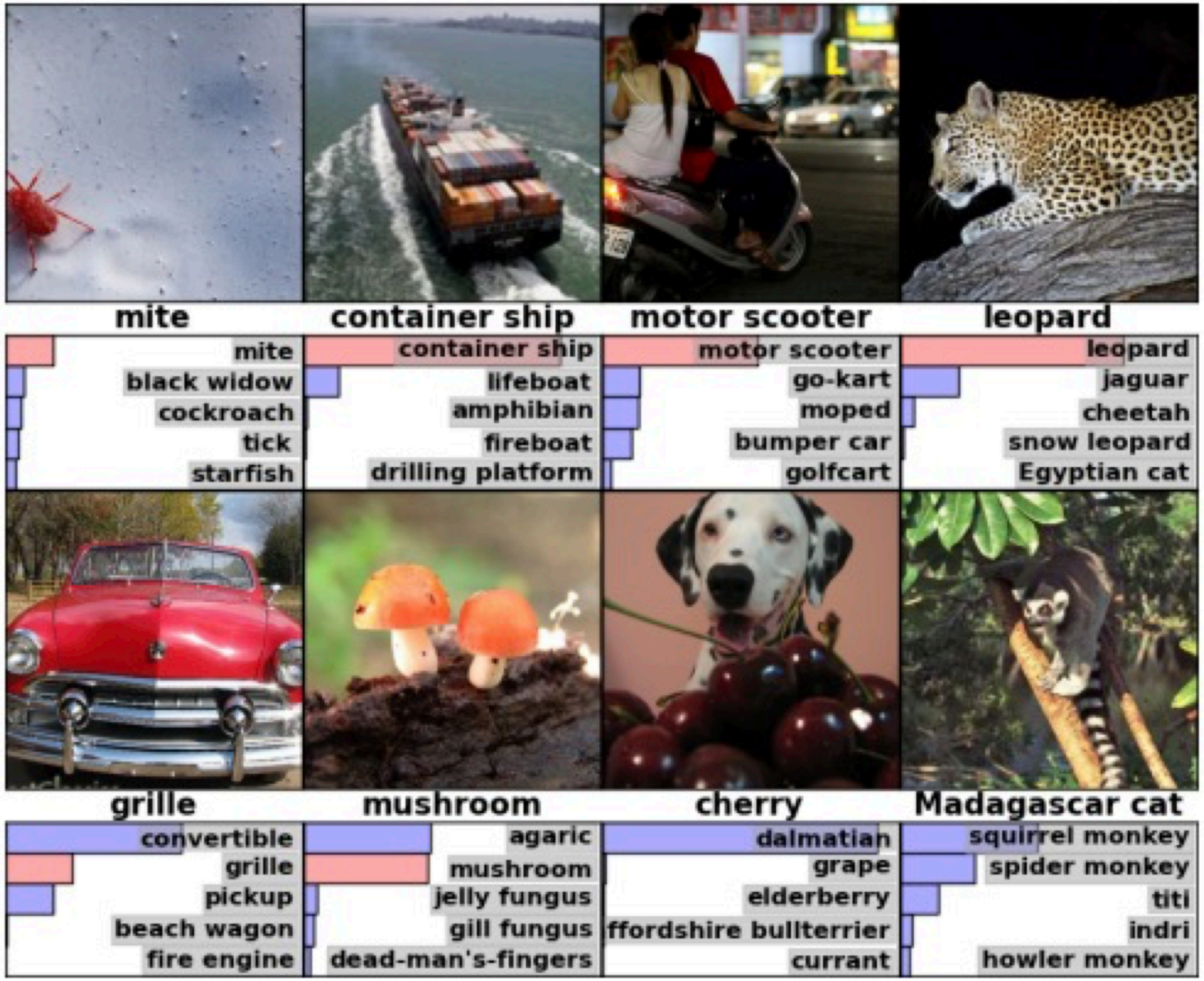
scale is large

ImageNet

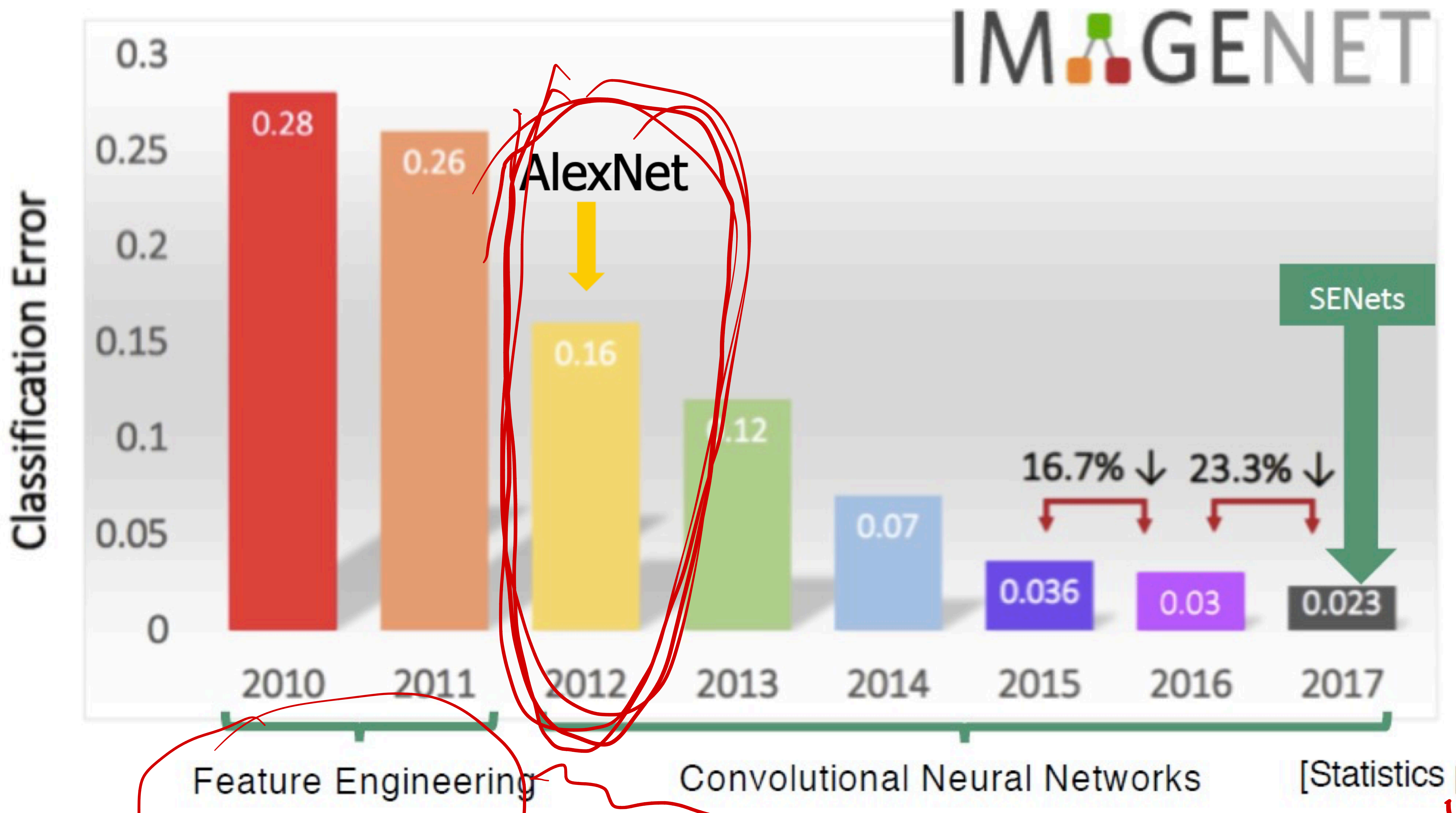
- ❑ 15M images
- ❑ 22K categories
- ❑ Images collected from Web
- ❑ Human labelers (Amazon's Mechanical Turk crowd-sourcing)
- ❑ ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
    - 1K categories
    - 1.2M training images (~1000 per category)
    - 50,000 validation images
    - 150,000 testing images

- ❑ RGB images
- ❑ Variable-resolution, but this architecture scales them to 256x256 size

ImageNet Results



ImageNet Results

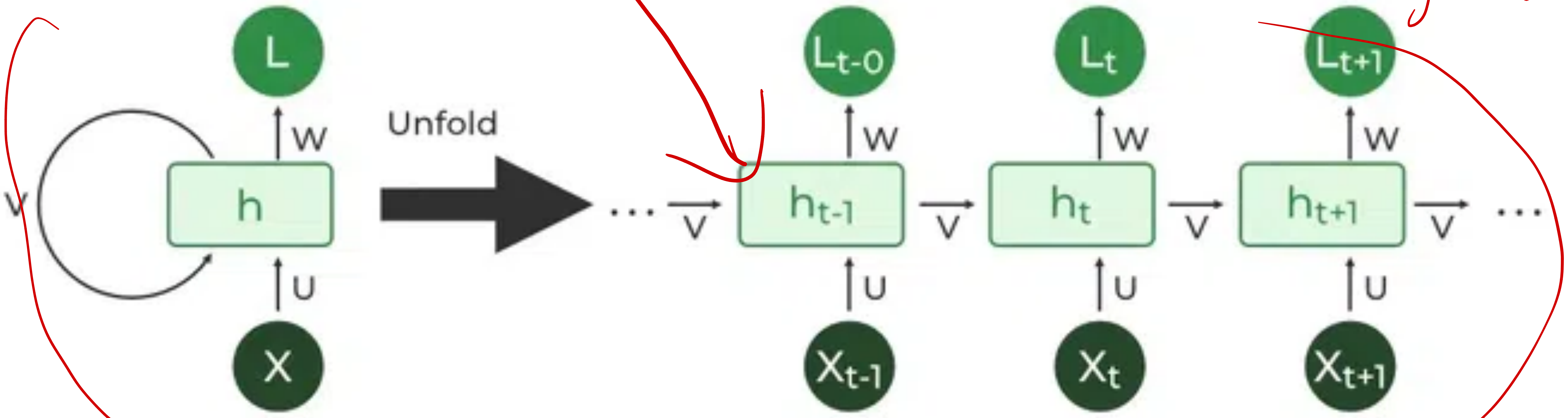


[Statistics provided by ILSVRC]

Recurrent Neural Networks (RNNs)

CNN for image

RNN for language



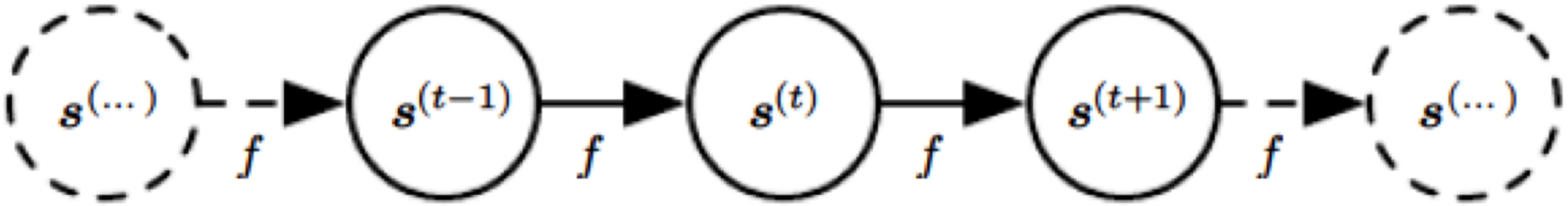
sequence

Recurrent Neural Networks

- Dates back to (Rumelhart *et al.*, 1986)
- A family of neural networks for handling sequential data, which involves variable length inputs or outputs

- Especially, for natural language processing (NLP)

Computation Graph



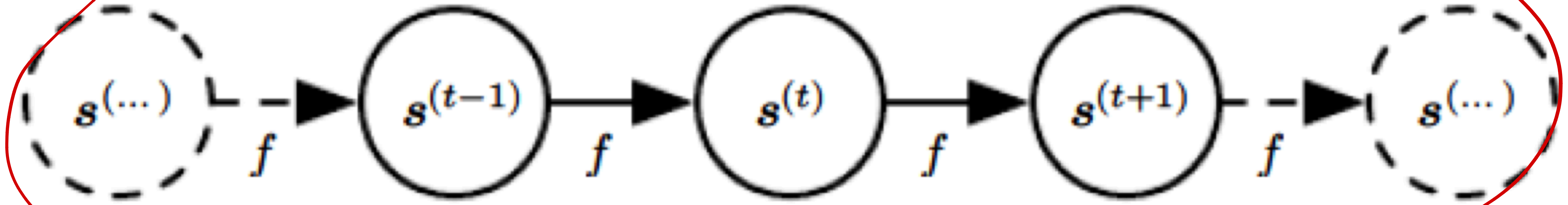
$$s^{(t+1)} = f(s^{(t)}; \theta)$$

Figure from *Deep Learning*, Goodfellow, Bengio and Courville

$$s^{(t+1)} = f(s^{(t)}; \theta)$$

parameters

Computation Graph

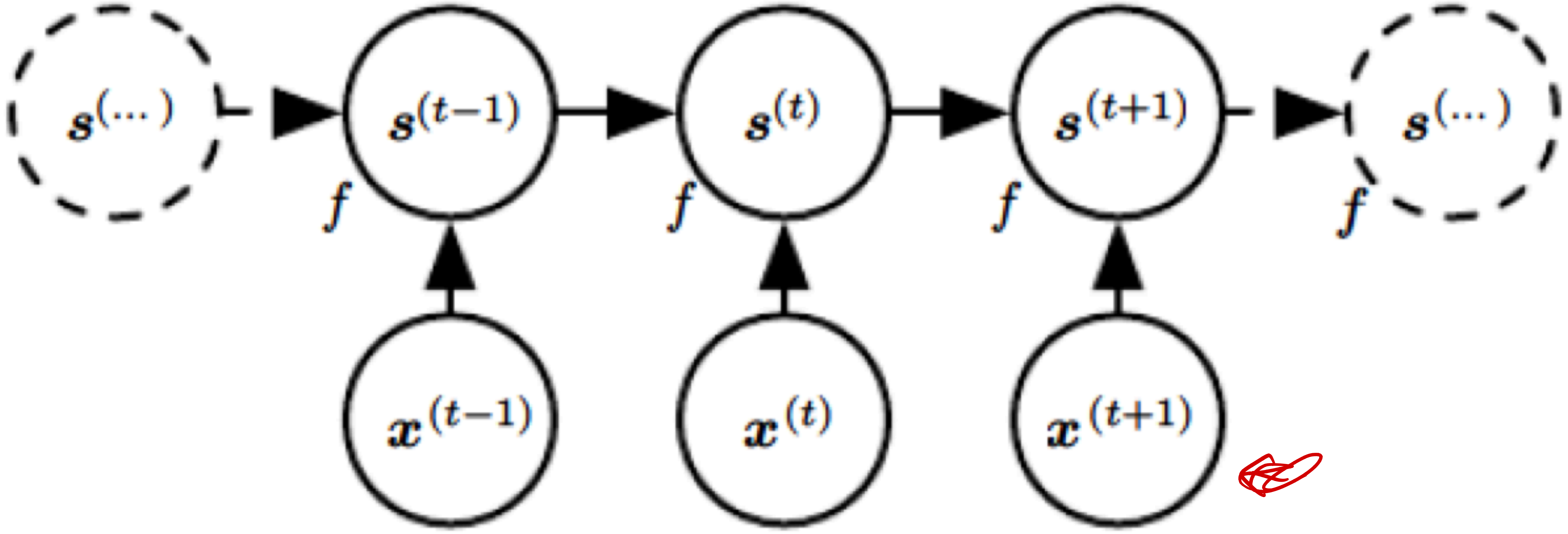


$$s^{(t+1)} = f(s^{(t)}; \theta)$$

Figure from *Deep Learning*, Goodfellow, Bengio and Courville

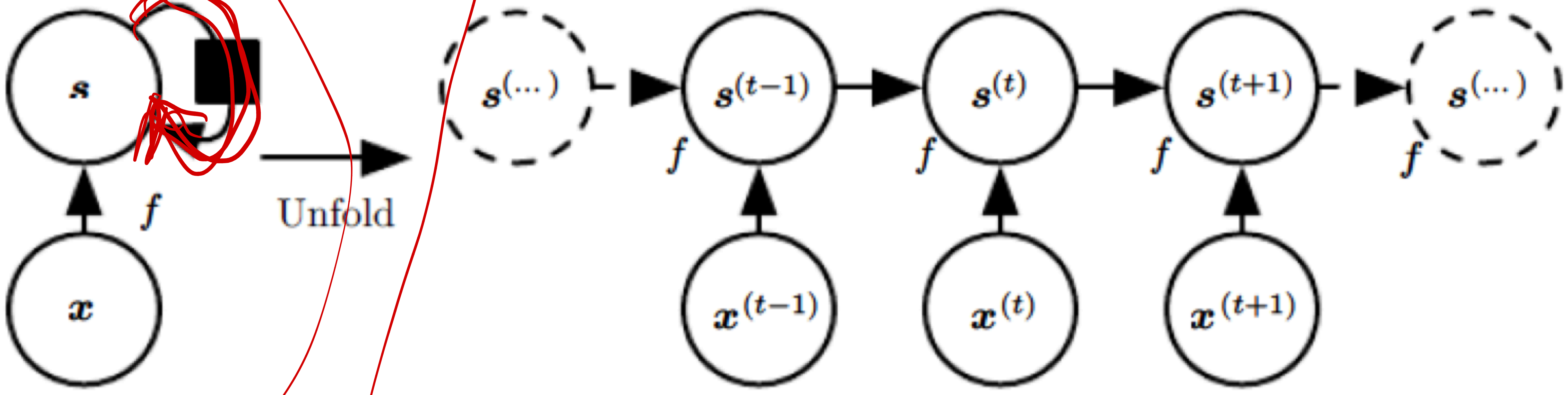
Like Markov model, but here $s^{(t+1)}$ is deterministic given $s^{(t)}$

Computation Graph



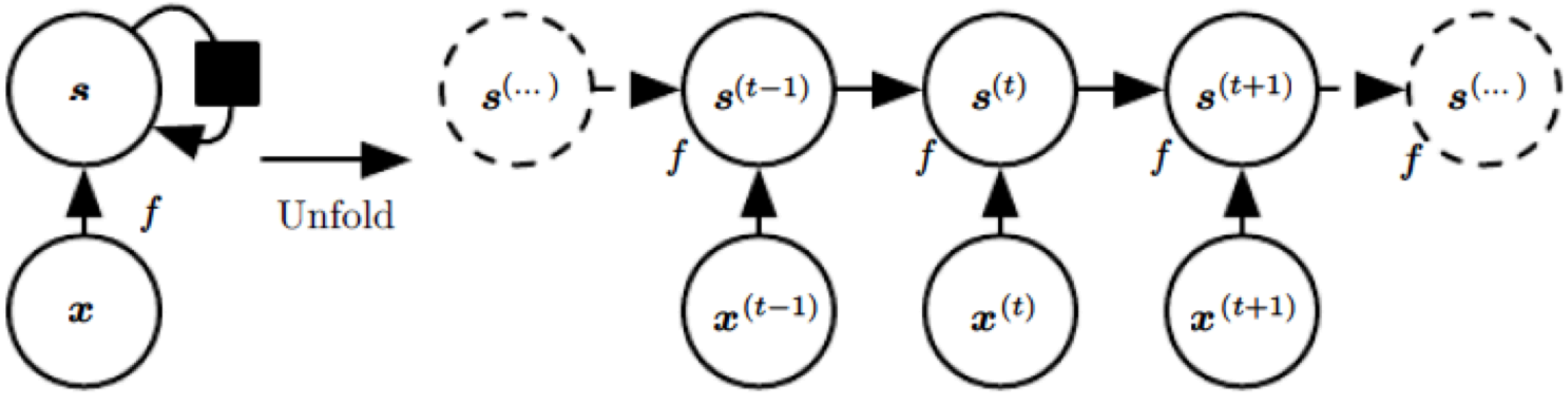
$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

Compact view



$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

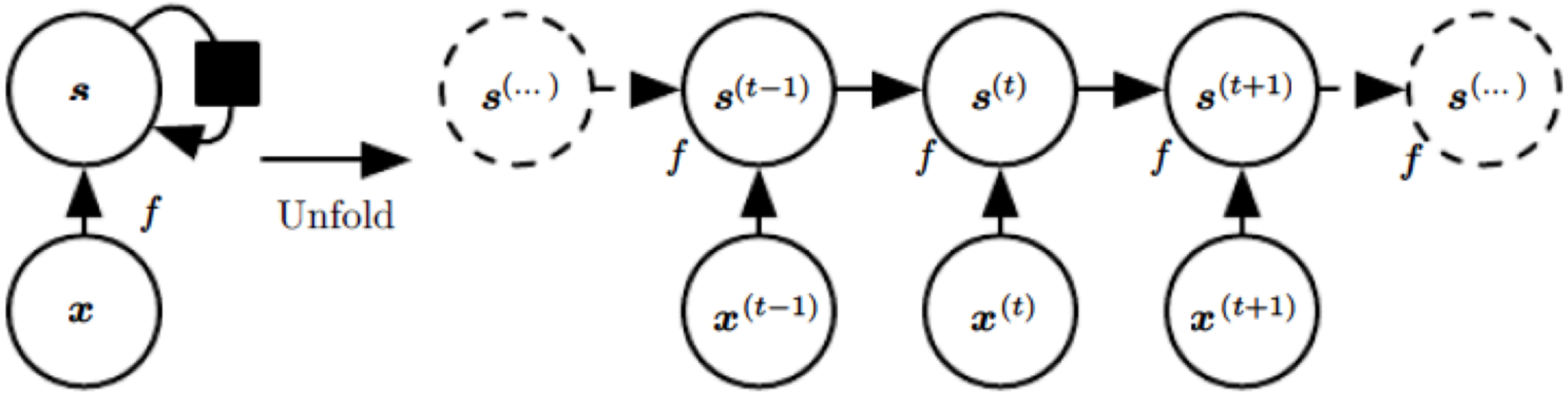
Compact view



$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

Key: the same f and θ for all time steps

Compact view



θ f

$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

Key: the same f and θ for all time steps

Like stationary HMM

w

Recurrent Neural Networks

Recurrent Neural Networks

- Use the same computational function and parameters across different time steps of the sequence

Recurrent Neural Networks

- Use the same computational function and parameters across different time steps of the sequence

- Each time step: takes the input entry and the previous hidden state to compute the output entry

Recurrent Neural Networks

- Use the same computational function and parameters across different time steps of the sequence

- Each time step: takes the input entry and the previous hidden state to compute the output entry

- Loss: typically computed every time step

Recurrent Neural Networks

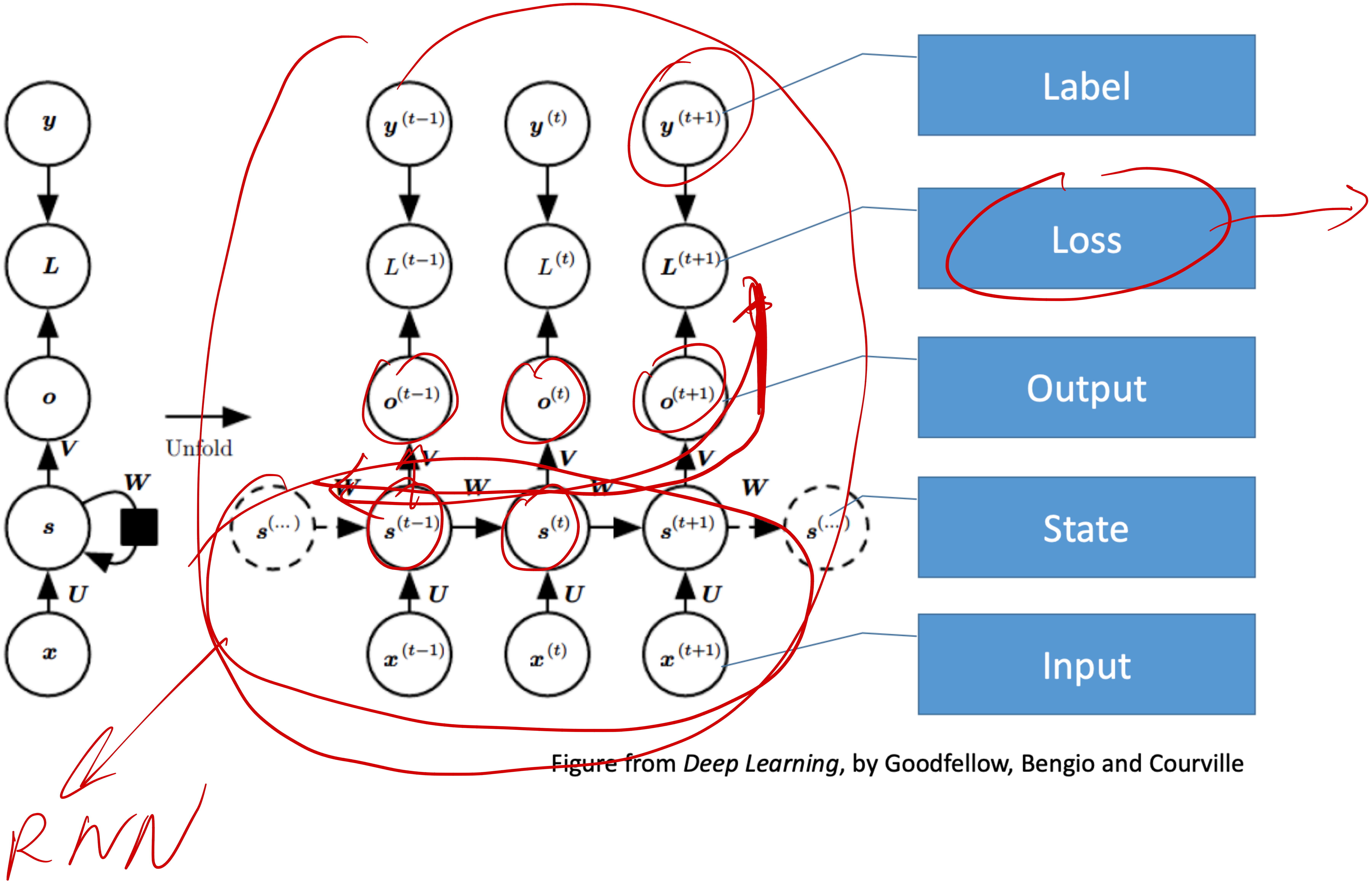
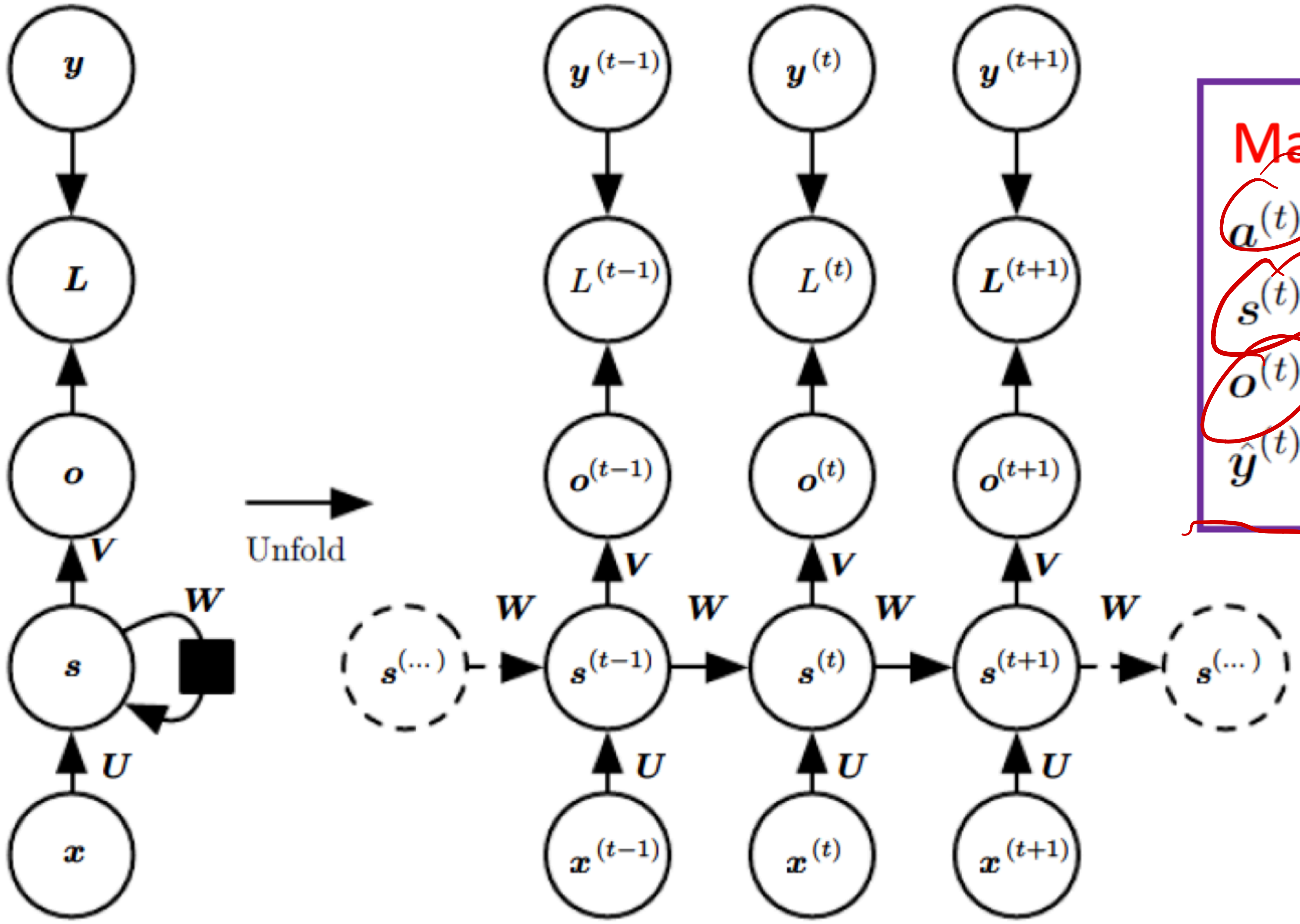


Figure from *Deep Learning*, by Goodfellow, Bengio and Courville

Recurrent Neural Networks



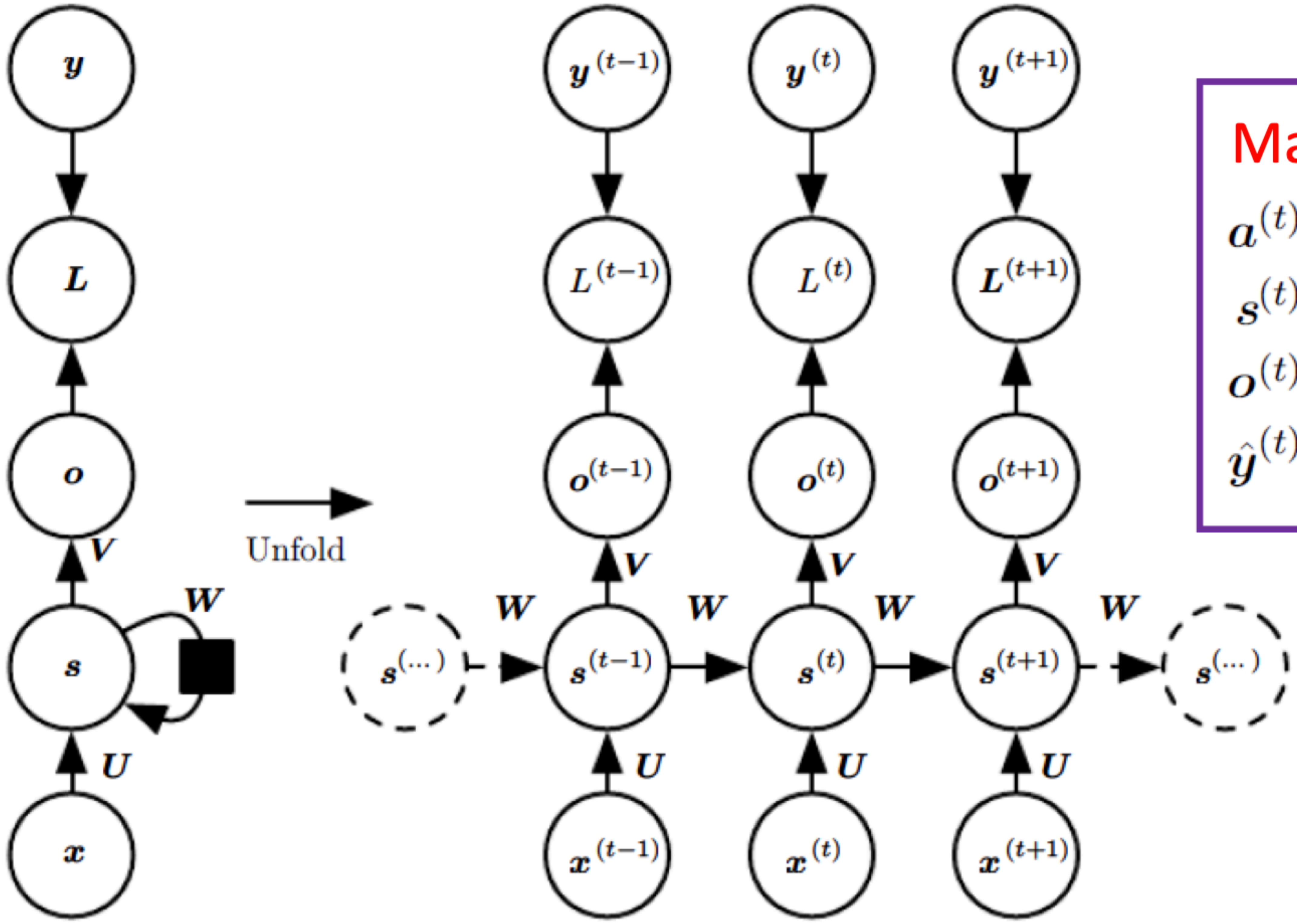
Math formula:

$$\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{s}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{s}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{s}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}$$

non linear

Figure from *Deep Learning*,
Goodfellow, Bengio and Courville

Recurrent Neural Networks



Math formula:

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{s}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{s}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{s}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Figure from *Deep Learning*, Goodfellow, Bengio and Courville

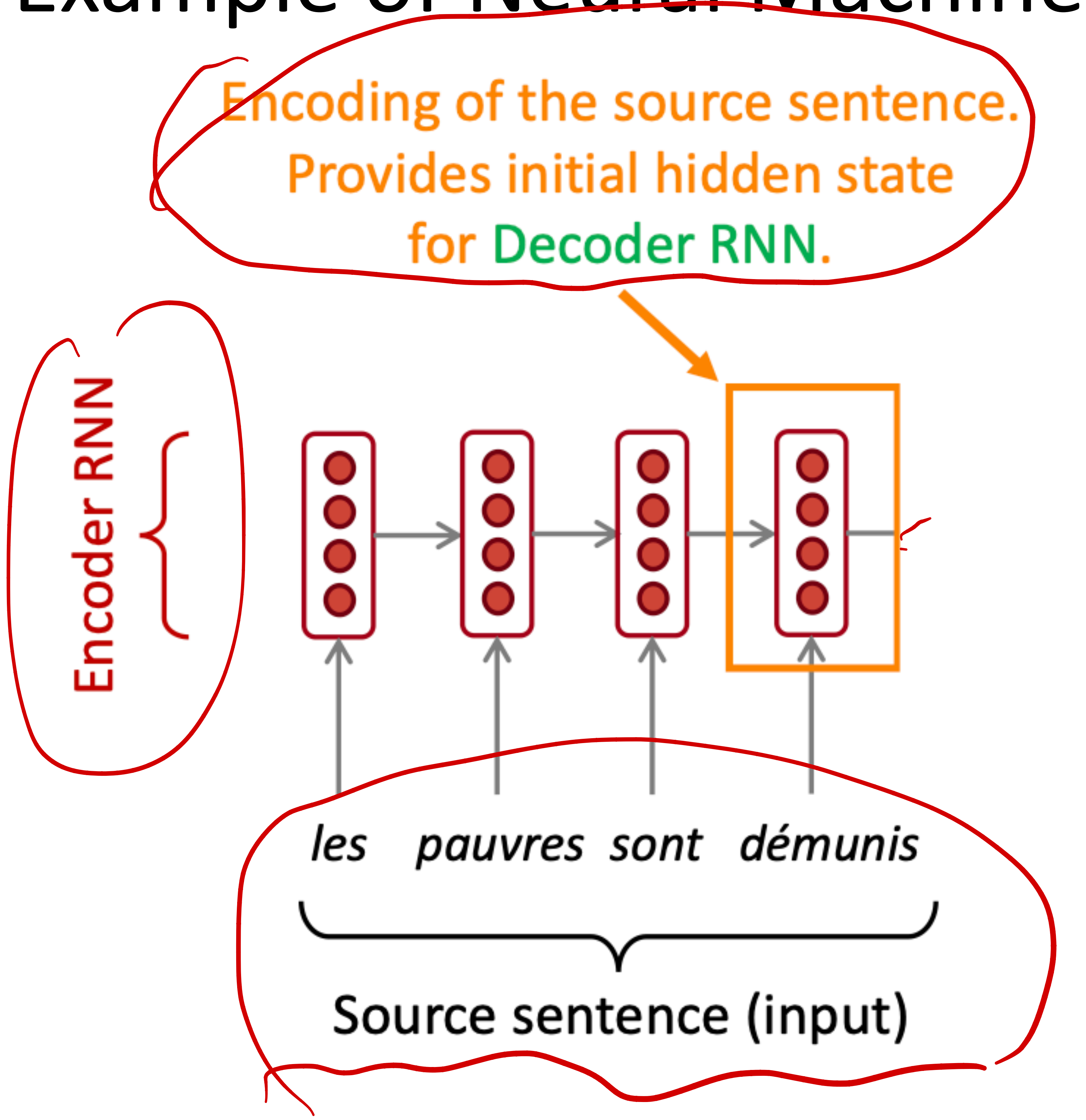
There are many variants of RNNs since the functional form to compute $s^{(t)}$ can vary, e.g., LSTM

Sequence-to-Sequence Learning

Example of Neural Machine Translation

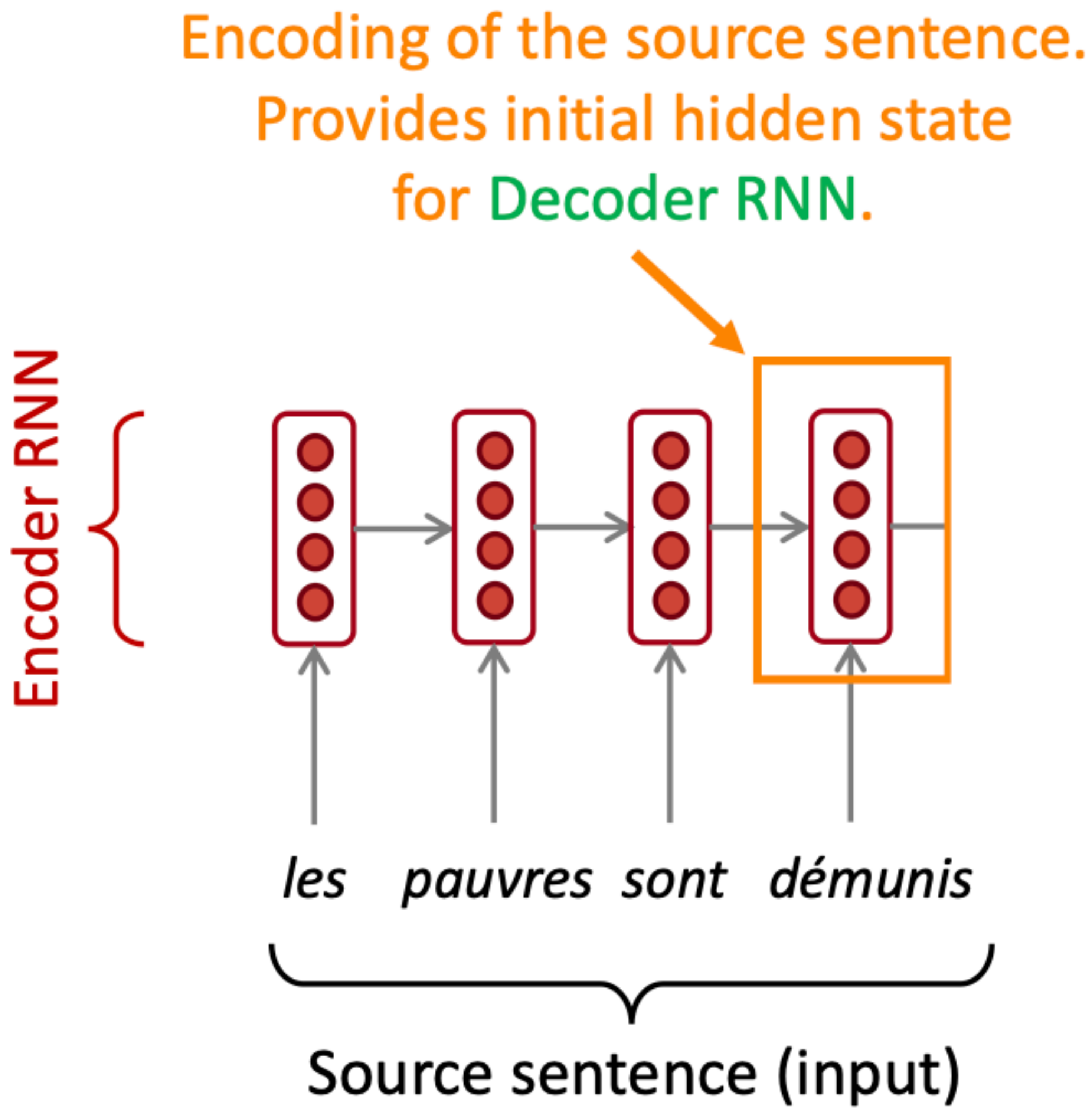
Sequence-to-Sequence Learning

Example of Neural Machine Translation



Sequence-to-Sequence Learning

Example of Neural Machine Translation

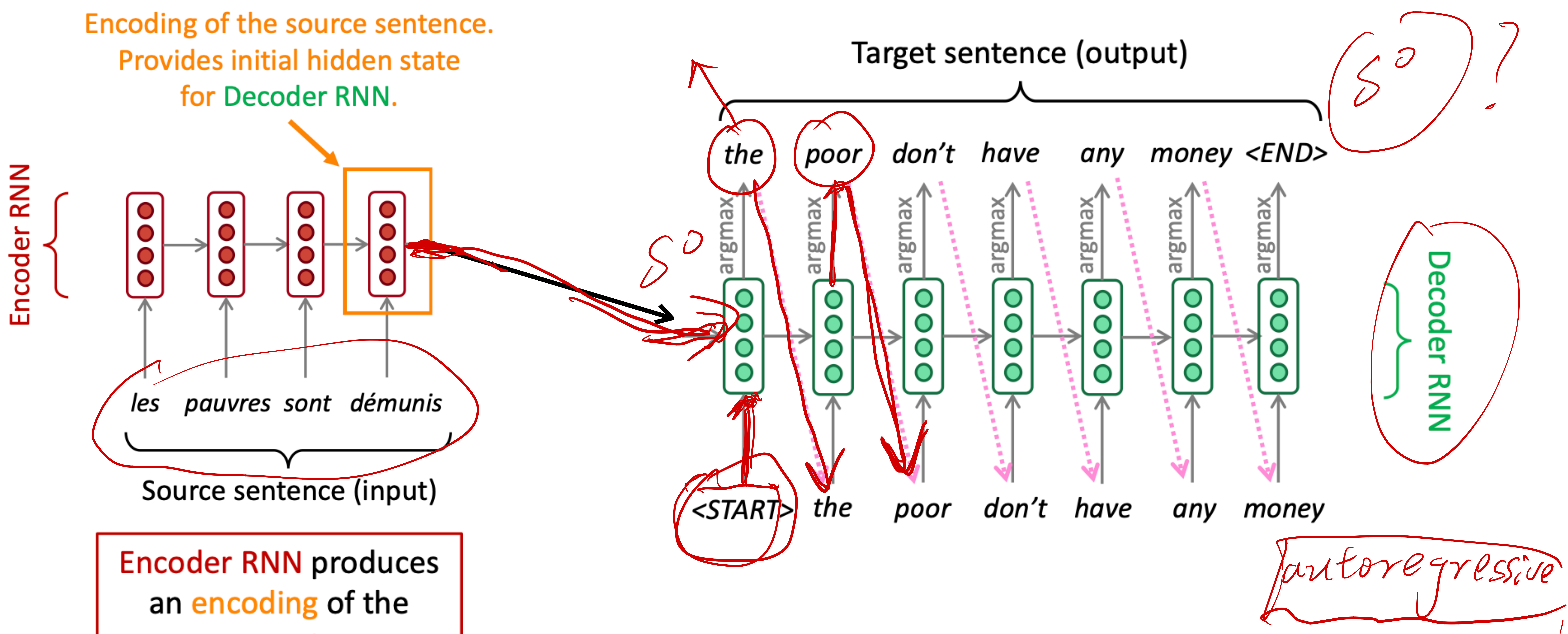


Encoder RNN produces an encoding of the source sentence.

Sequence-to-Sequence Learning

$s^t = f(s^{t-1}, x)$

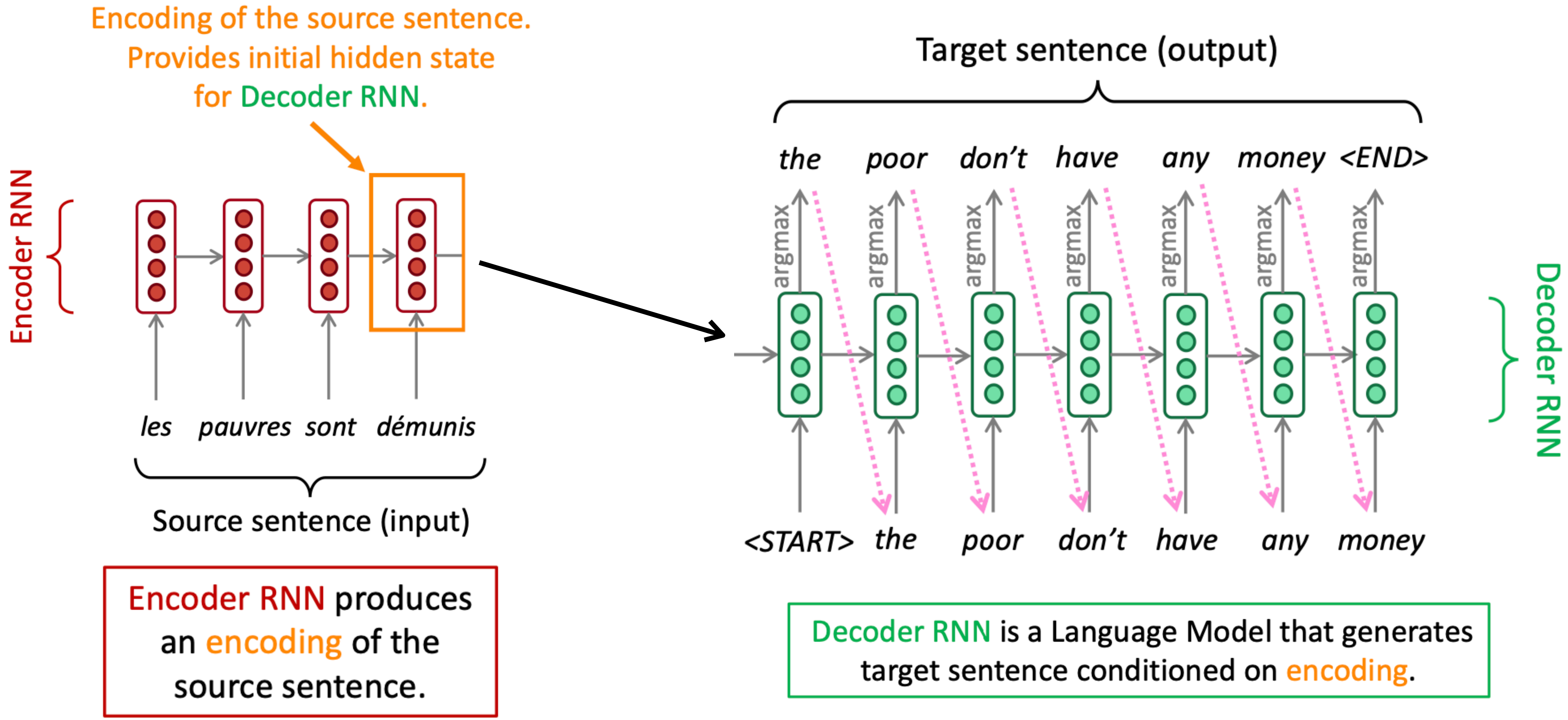
Example of Neural Machine Translation



Encoder RNN produces an encoding of the source sentence.

Sequence-to-Sequence Learning

Example of Neural Machine Translation



Residual Connection

We want deeper and deeper NNs, but going deep is difficult

Residual Connection

We want deeper and deeper NNs, but going deep is difficult

- Troubles accumulate w/ more layers
- Signals get distorted when propagated
- in forward and backward passes

large gradient explode

vanish

Residual Connection

We want deeper and deeper NNs, but going deep is difficult

- Troubles accumulate w/ more layers
- Signals get distorted when propagated
- in forward and backward passes

Commonly used techniques to train "Deep" NNs:

Weight initialization

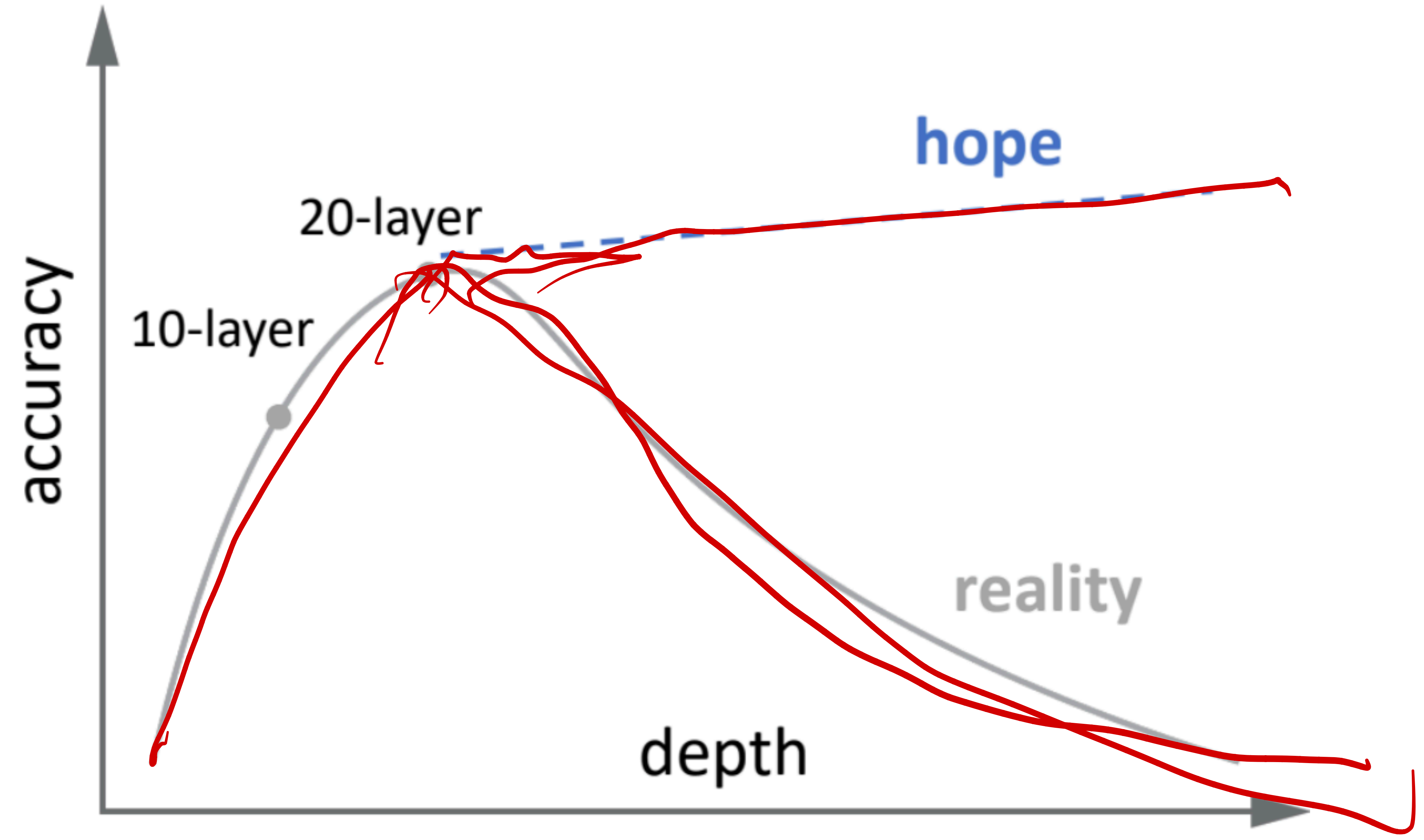
Normalization modules

Deep residual learning

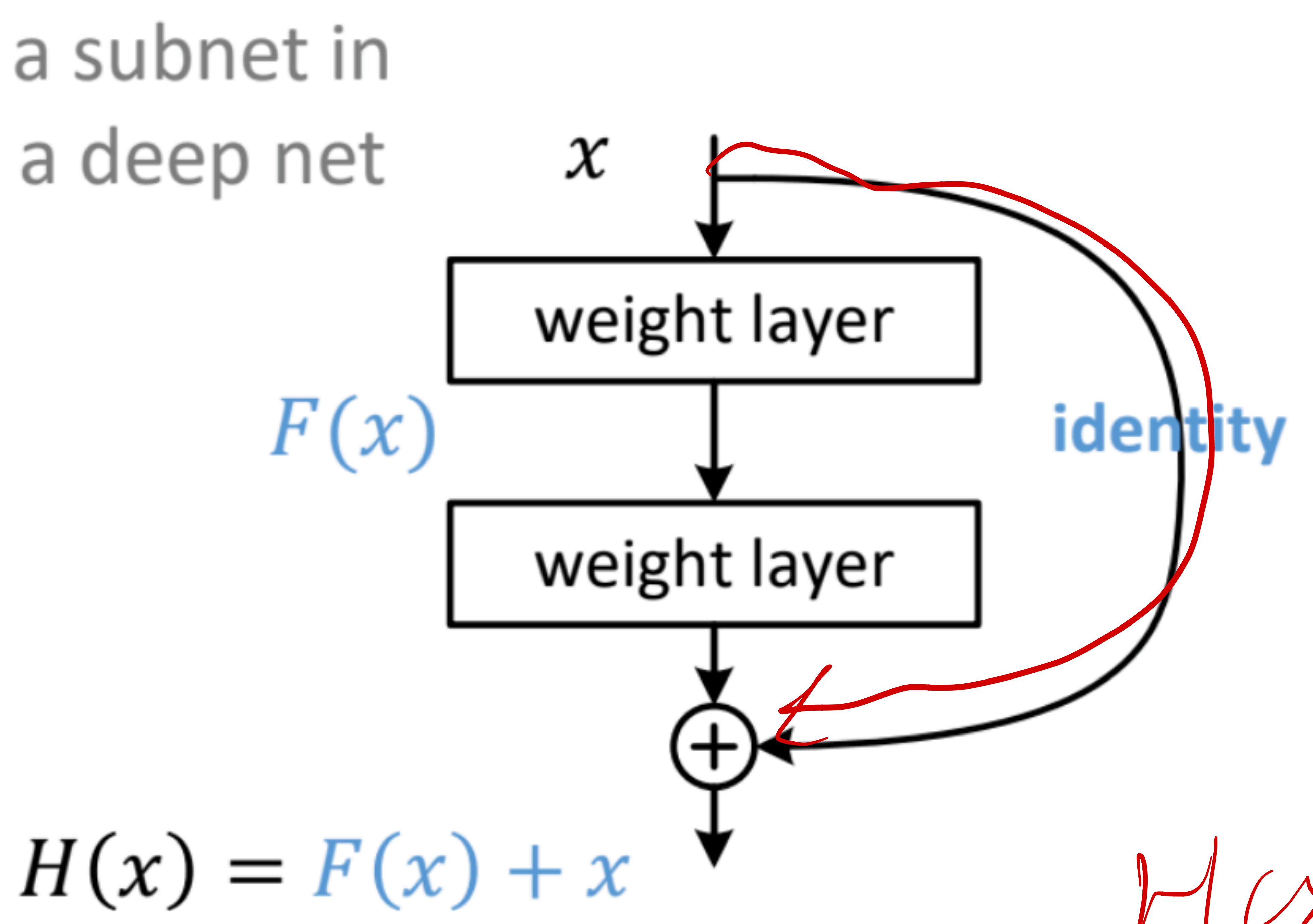
The Degradation Problem

- Good init + norm enable training deeper models
- Simply stacking more layers?

- Degrade after ~20 layers
- Not overfitting
- Difficult to train



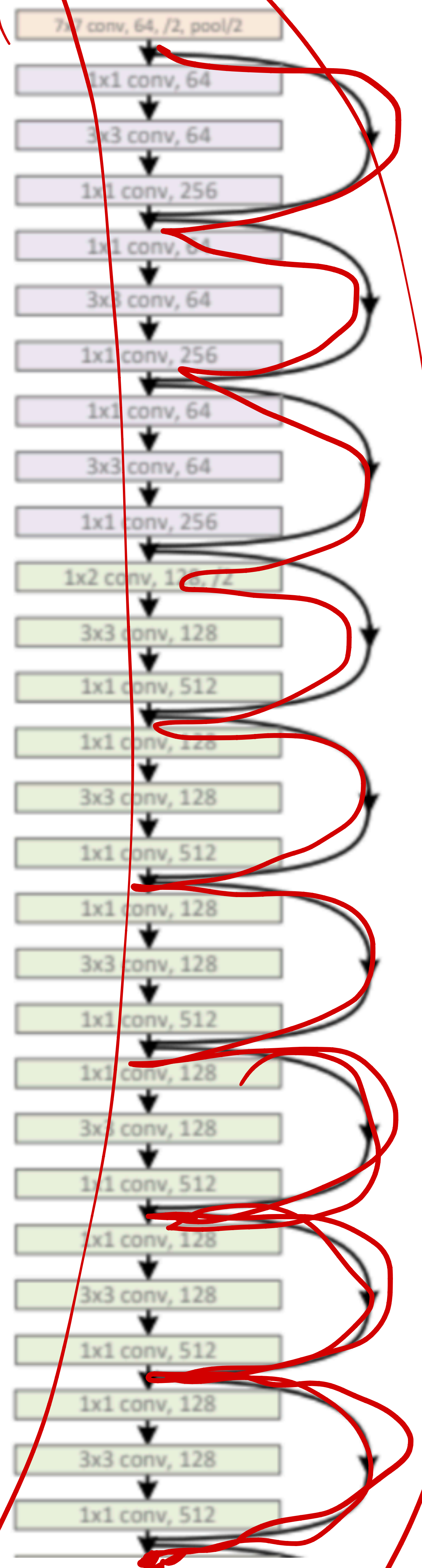
Deep Residual Learning



$H(x) = F(x) + x$

Deep Residual Networks (ResNet)

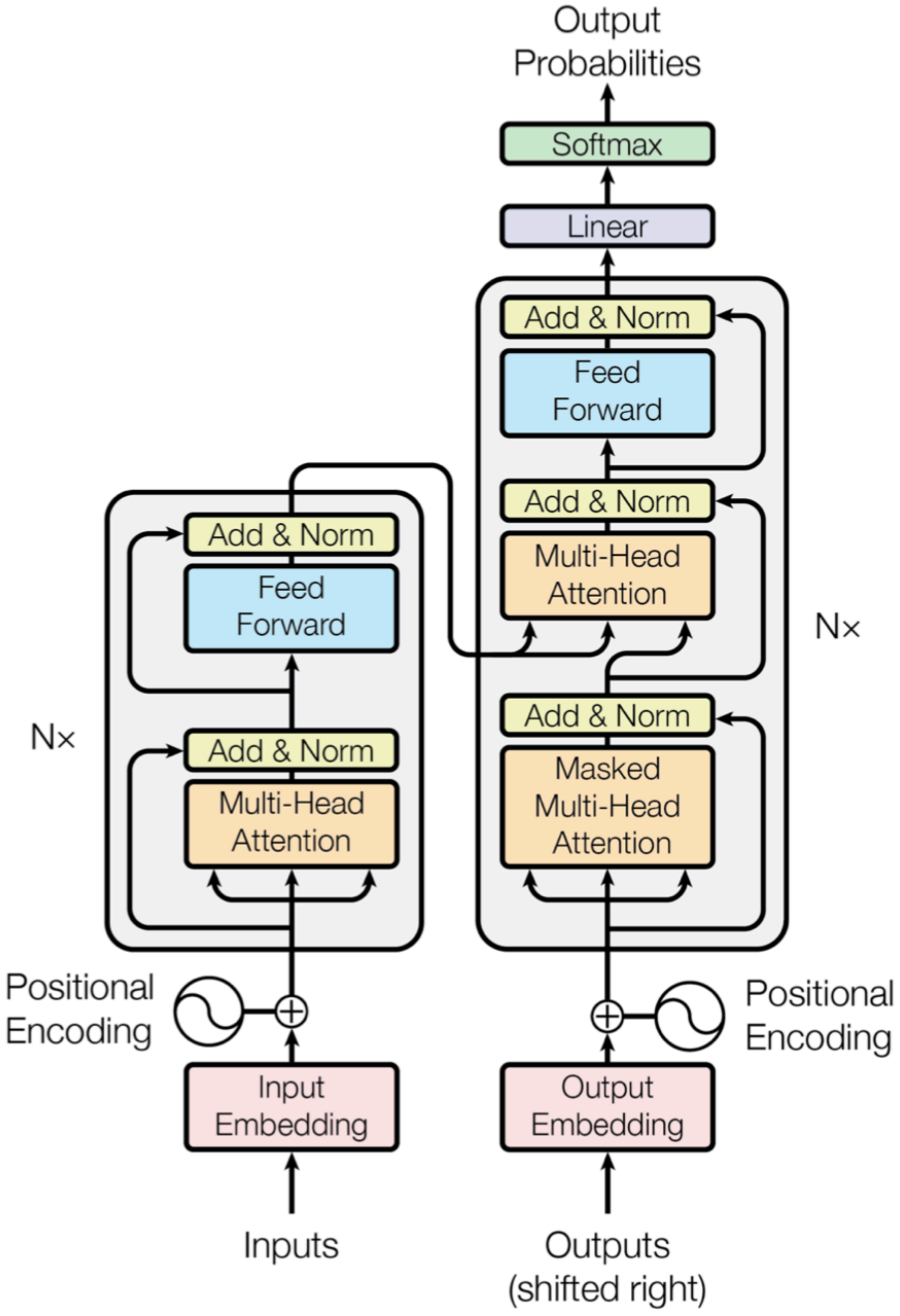
ResNet-152



deep

Kaiming He et al. Deep Residual Learning for Image Recognition. CVPR 2016.

Transformers



Vaswani et al. Attention is All You Need. NeurIPS 2017.