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

COMP 5212
Machine Learning
Lecture 19

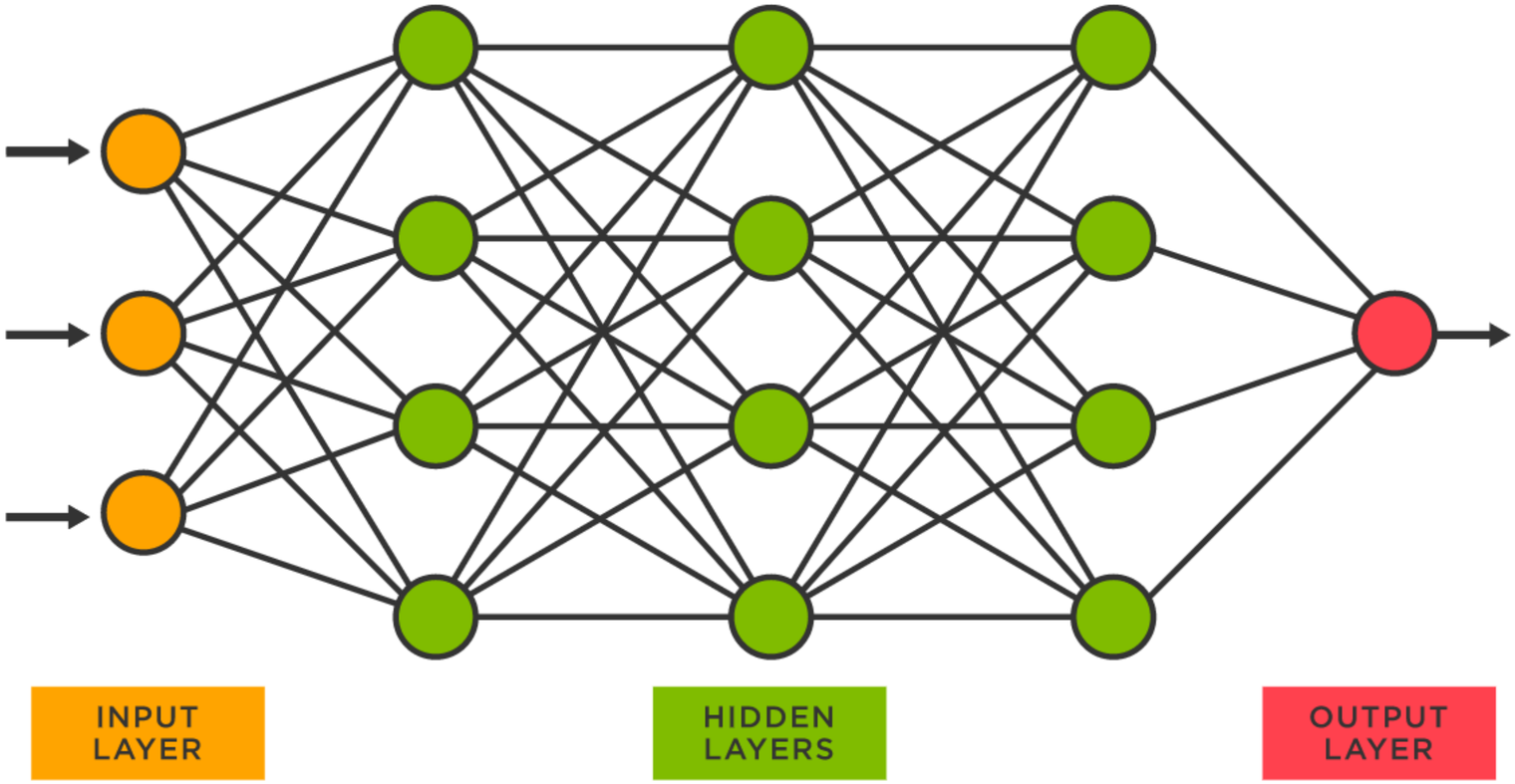
Neural Networks, Architectures

Junxian He
Nov 12, 2024

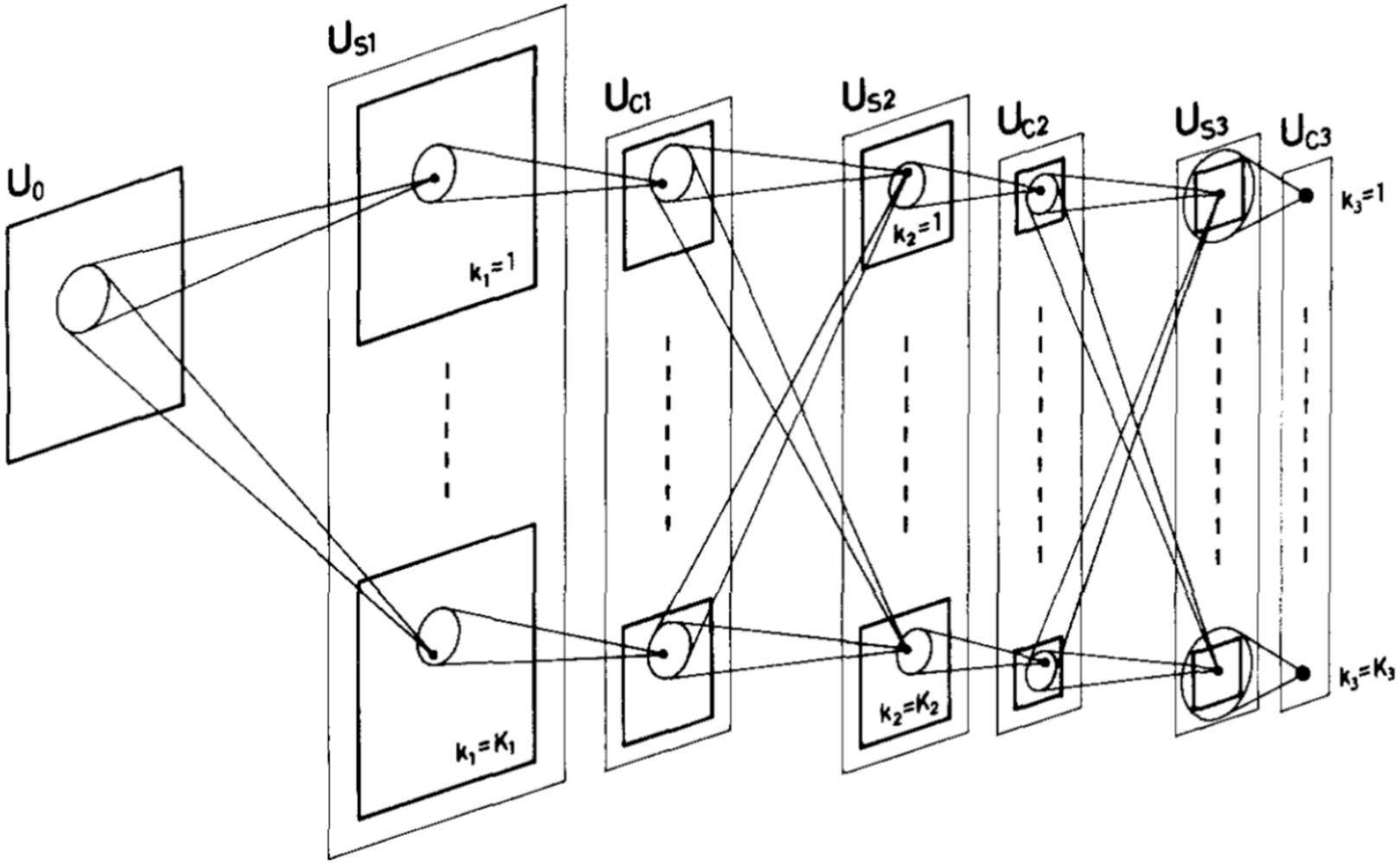
Announcements

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

Recap: Multilayer Perceptron Neural Networks (MLP)

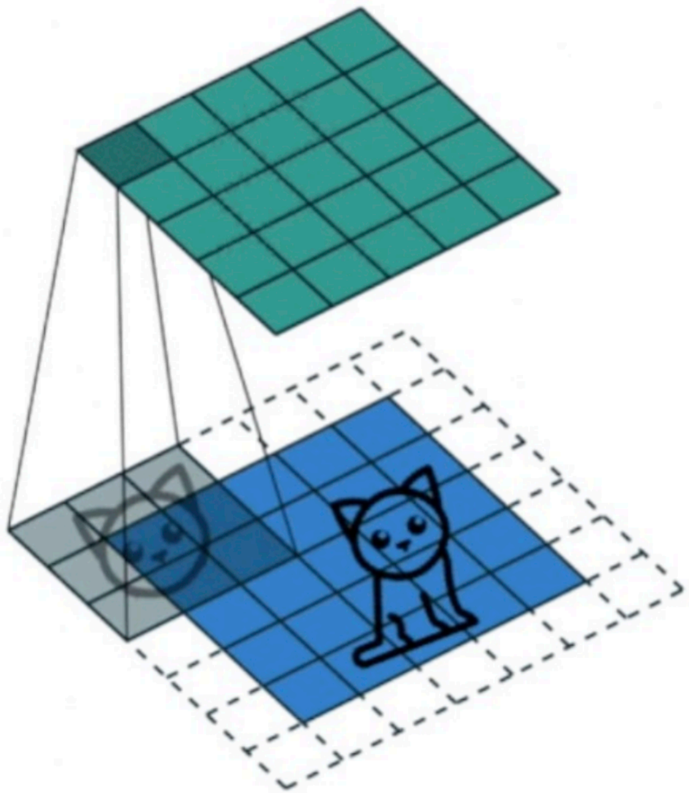


Convolutional Neural Networks



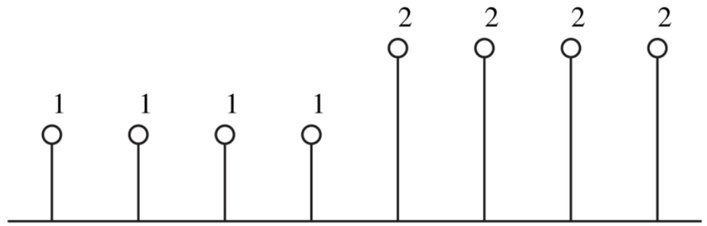
Convolution is template matching

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

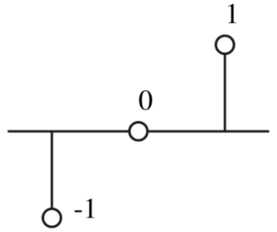


Convolution: a 1-D example

input

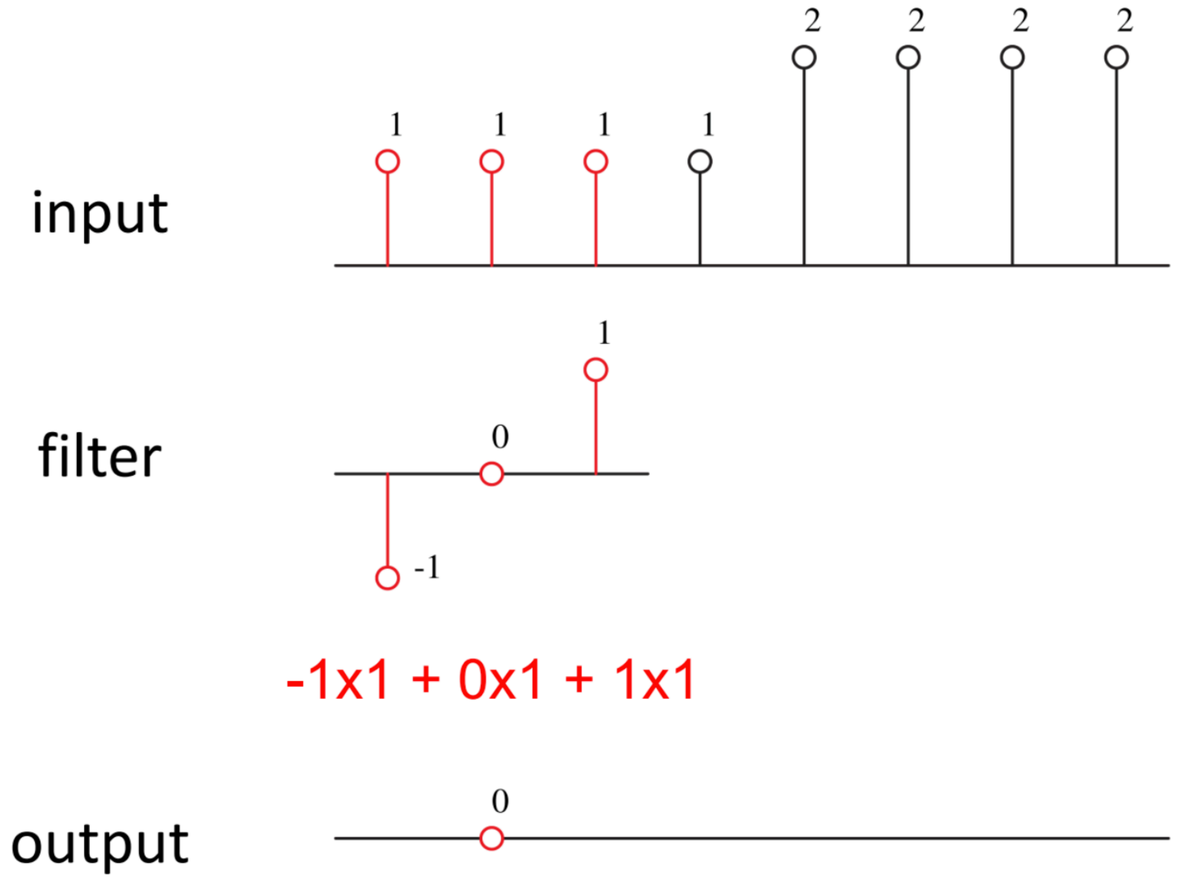


filter



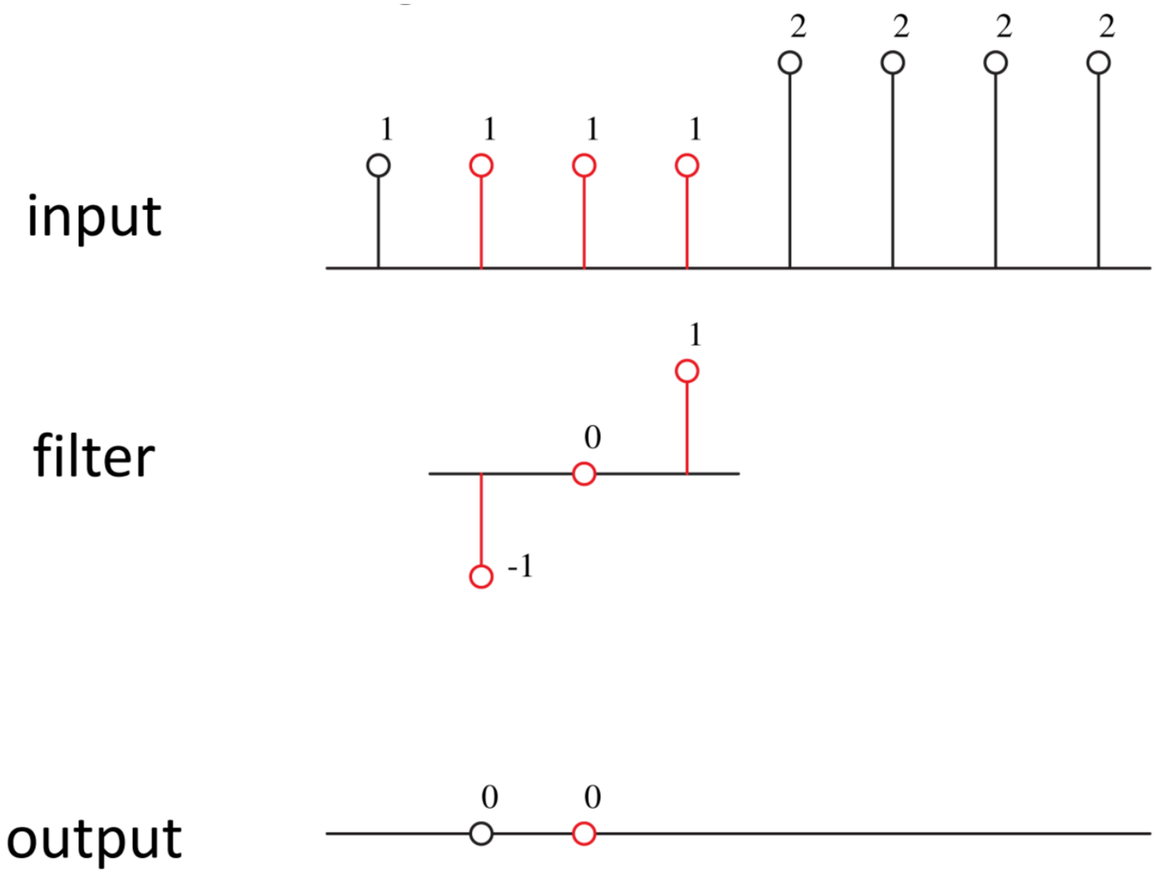
Convolution: a 1-D example

- sliding window
- dot product



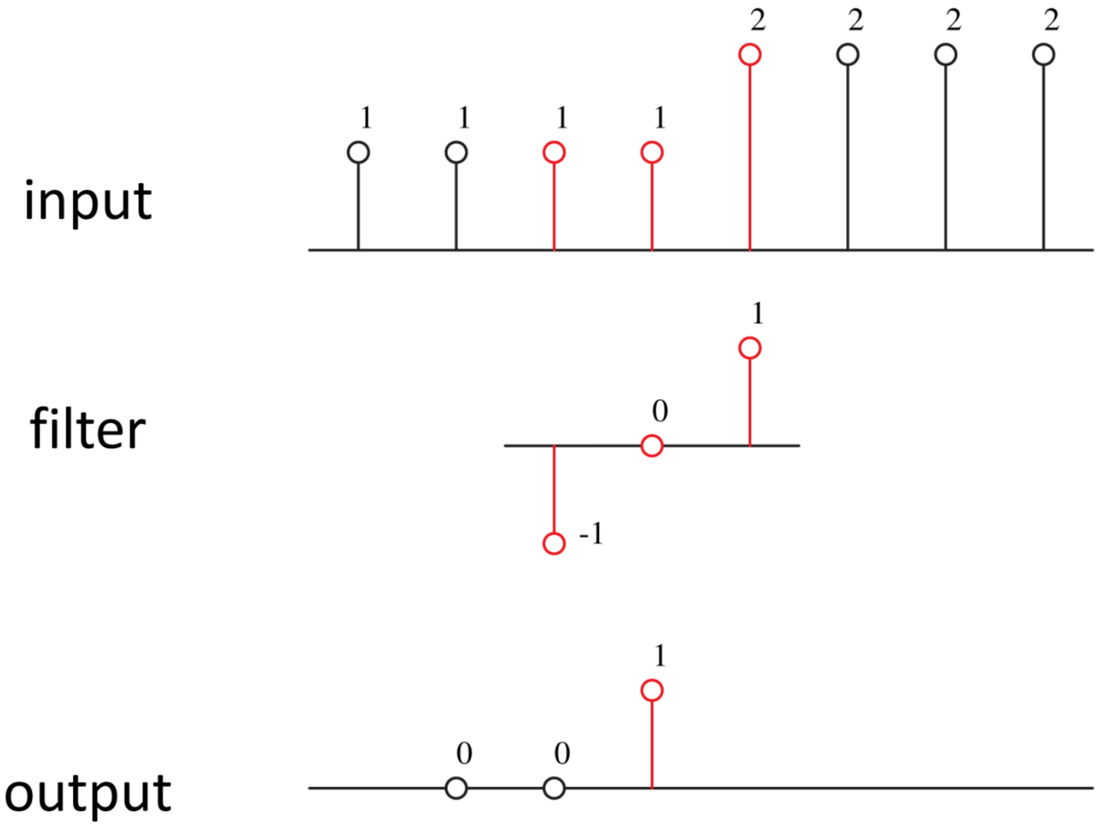
Convolution: a 1-D example

- sliding window
- dot product



Convolution: a 1-D example

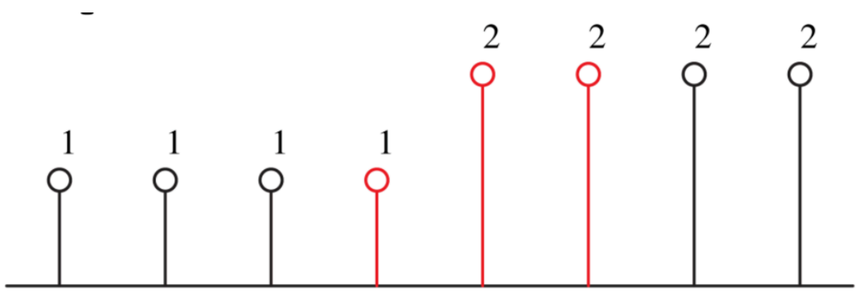
- sliding window
- dot product



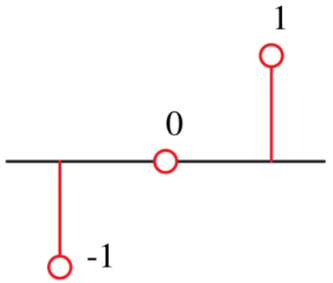
Convolution: a 1-D example

- sliding window
- dot product

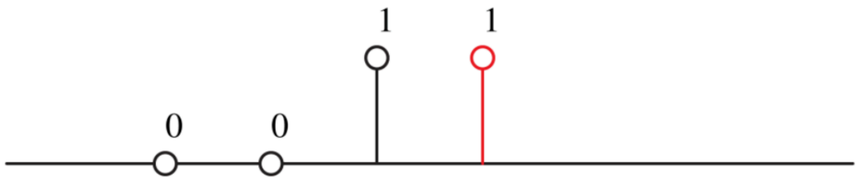
input



filter



output



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

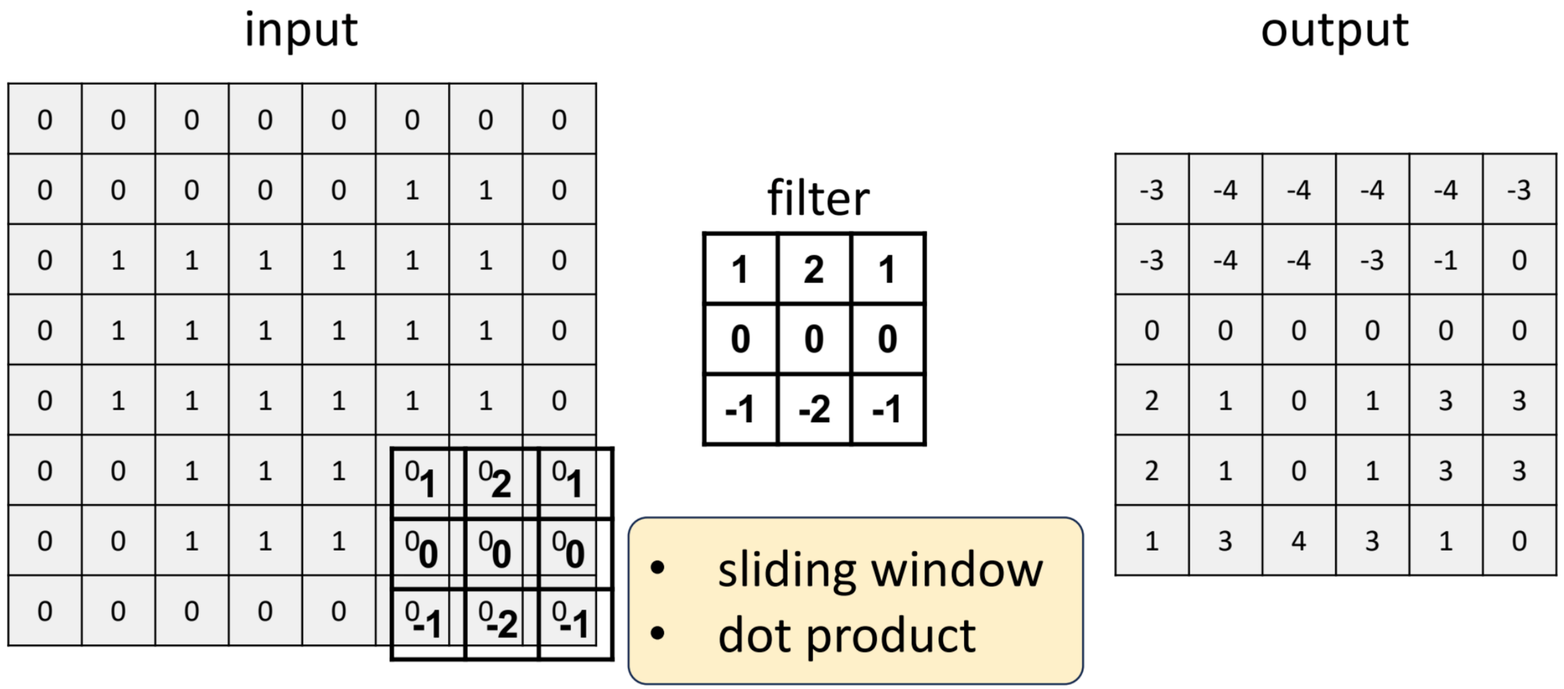
1	**2**	**1**
0	**0**	**0**
-1	**-2**	**-1**

- sliding window
- dot product

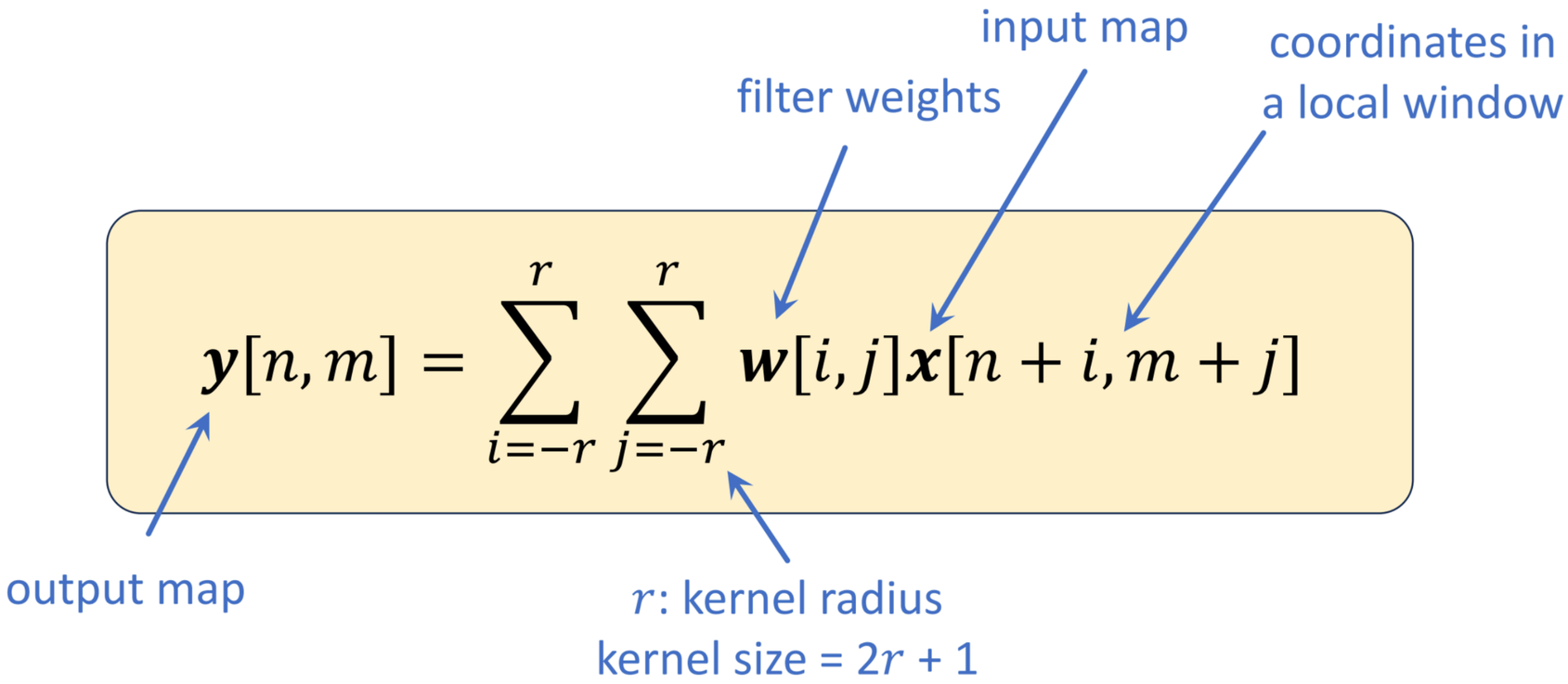
output

-3	-4	-4			

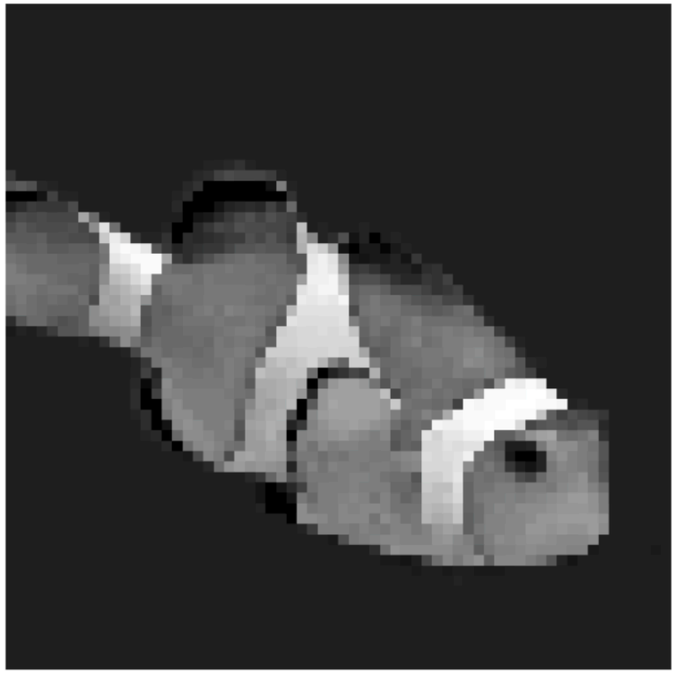
Convolution: a 2-D example



Convolution: a 2-D example



Convolution: 2-D



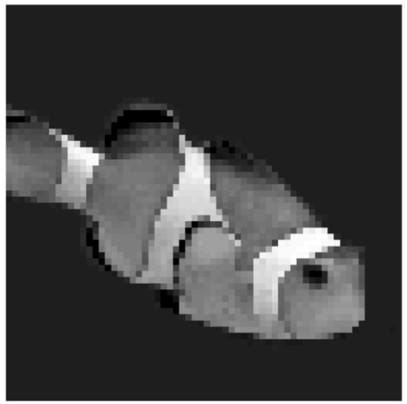
*

filter

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

=

Convolution: Multi-channel outputs



$*$

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

$=$

one filter, one feature

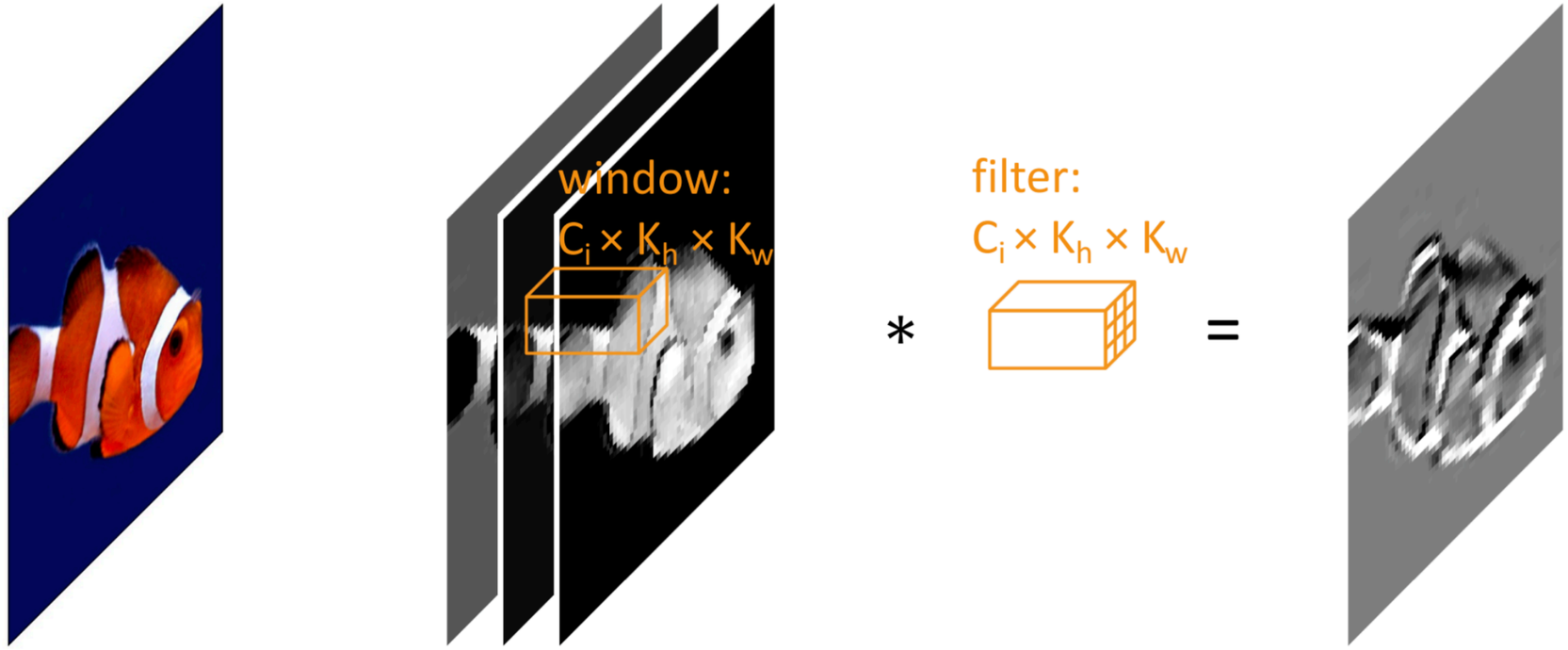
$*$

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

$=$

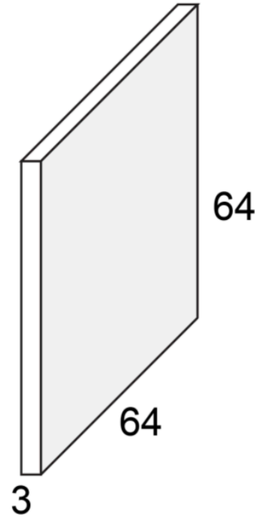
Understanding the filter/kernel as feature extractors

Convolution: Multi-channel inputs



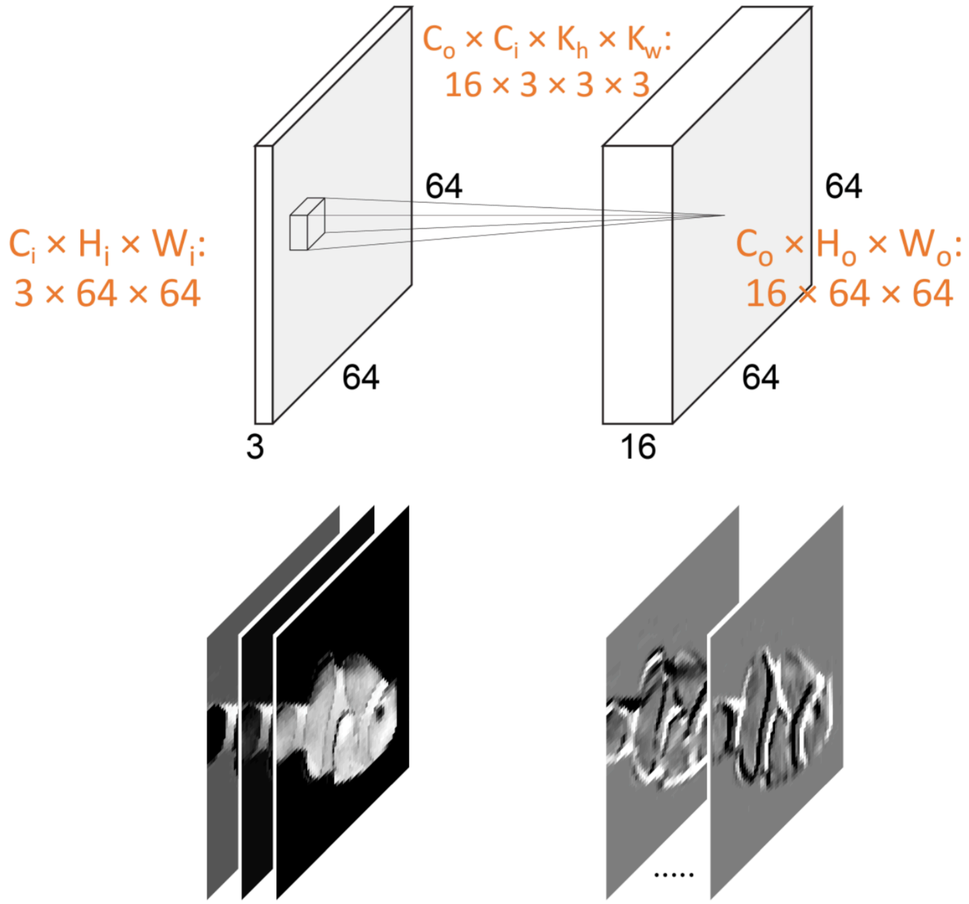
Like (R, G, B) color notations have three features

Convolution: tensor views



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

Convolution: tensor view



- Tensor: high-dimension array
- feature maps
  - 3-D tensor: C × H × 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

The same filter tensor applies to different locations

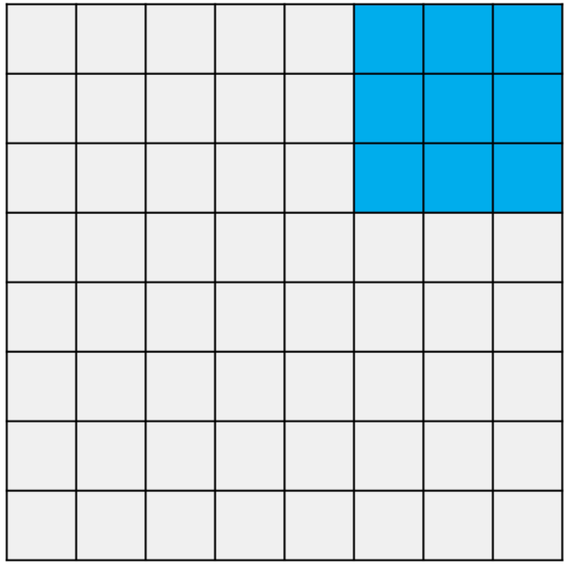
Convolution: # parameters and # operations

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

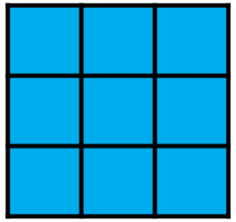
- # floating-point operations (FLOPs)
  - # params $\times H_o \times W_o$

Convolution: padding

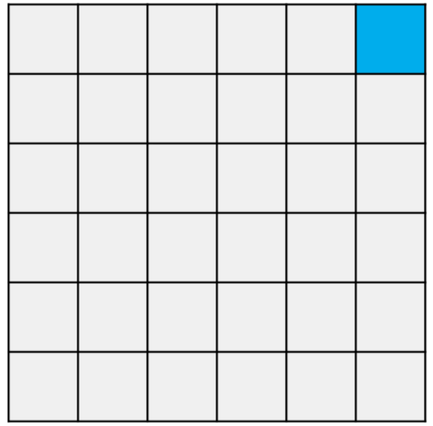
input: H × W = 8 × 8



filter



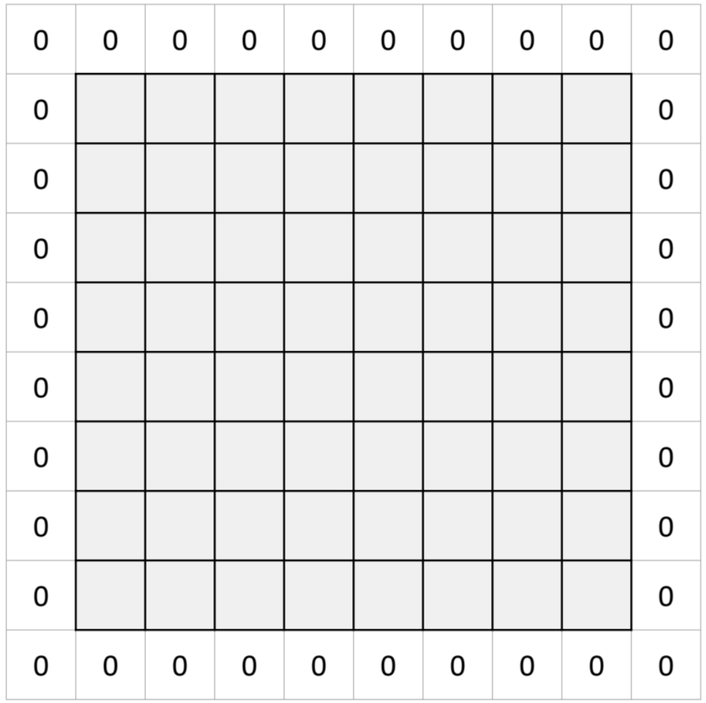
output: H × W = 6 × 6



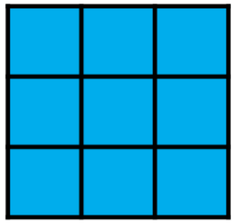
$$H_{out} = H_{in} - K_h + 1$$

Convolution: padding

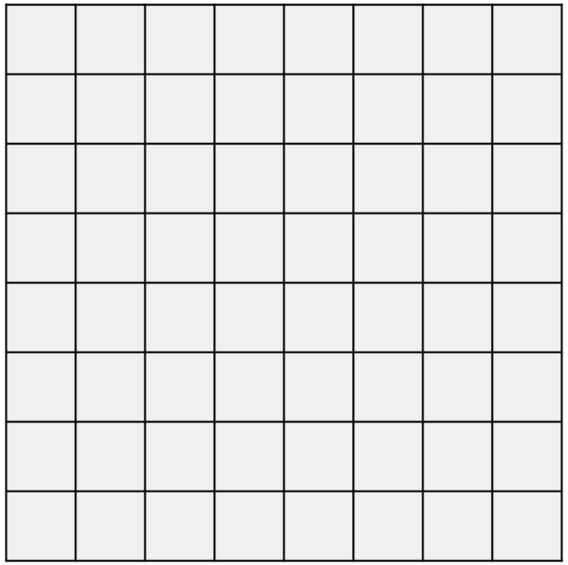
input: 8 × 8, + pad



filter



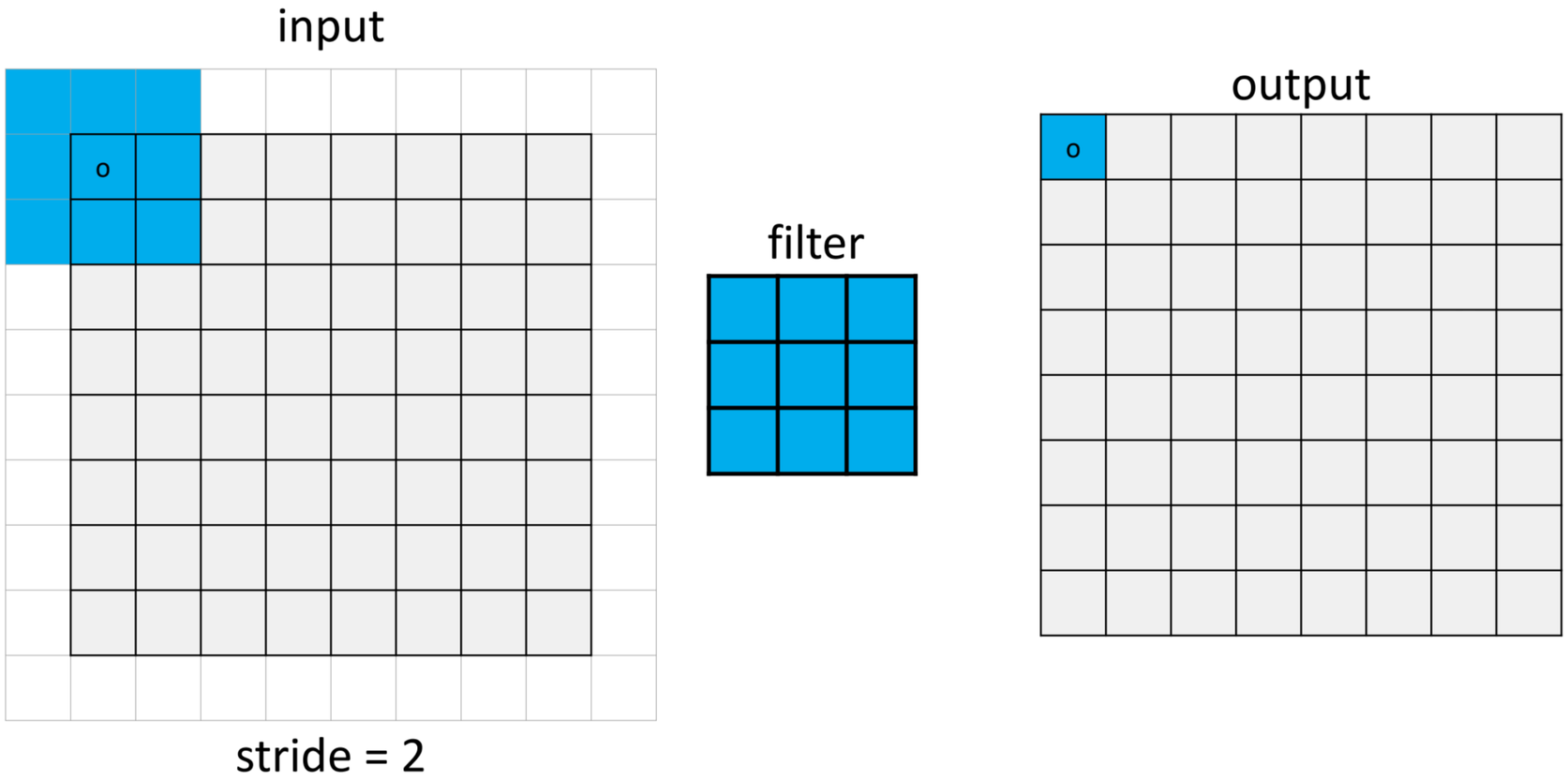
output: H × W = 8 × 8



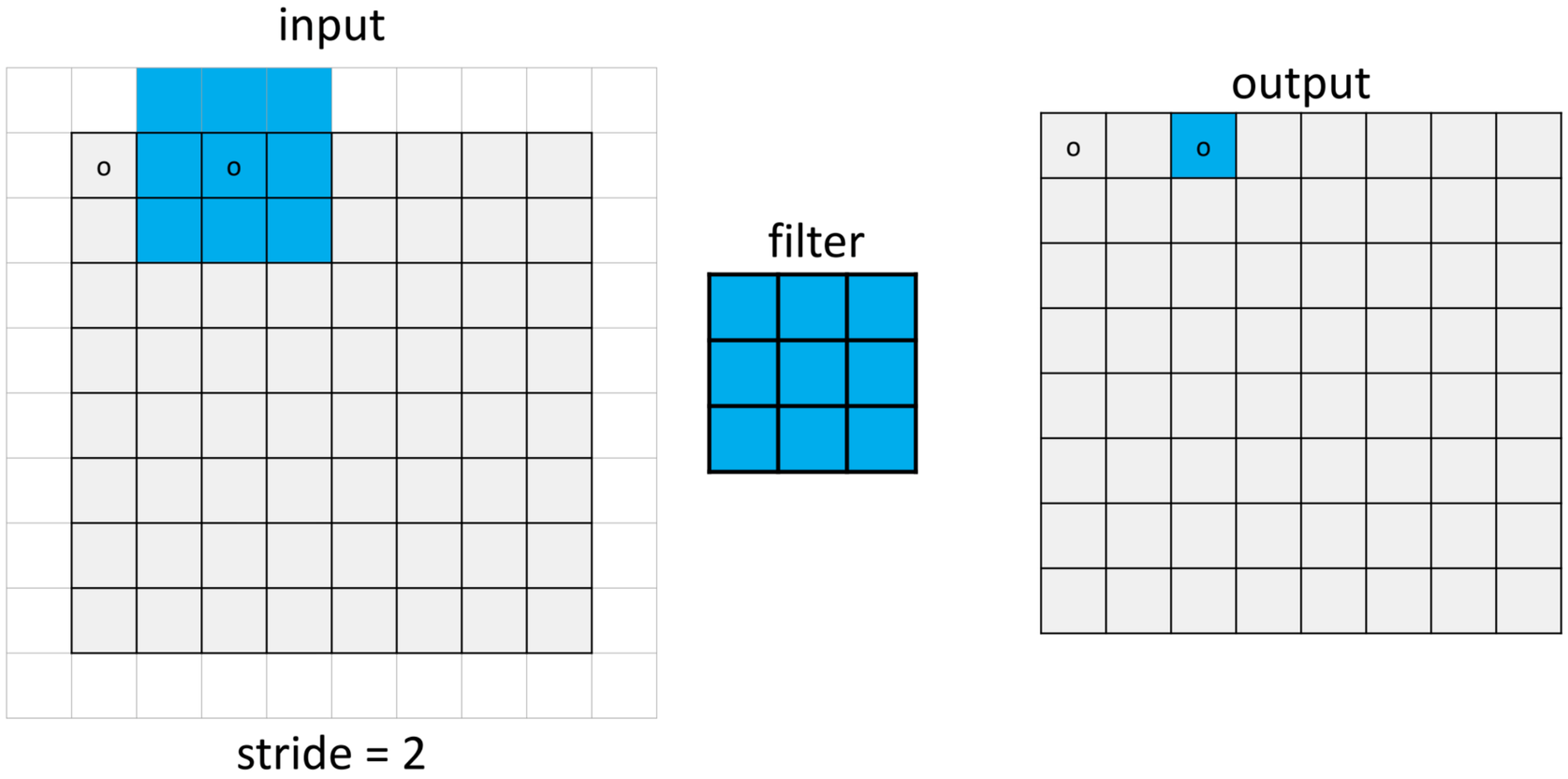
- pad = ⌊kernel_size / 2⌋
- maintains feature map size

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

Convolution: stride

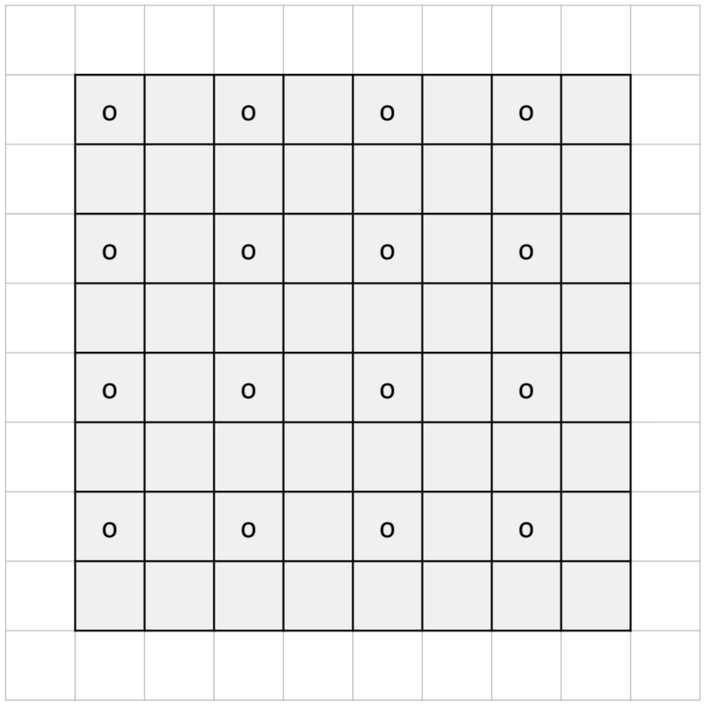


Convolution: stride



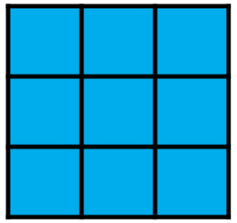
Convolution: stride

input: H × W = 8 × 8

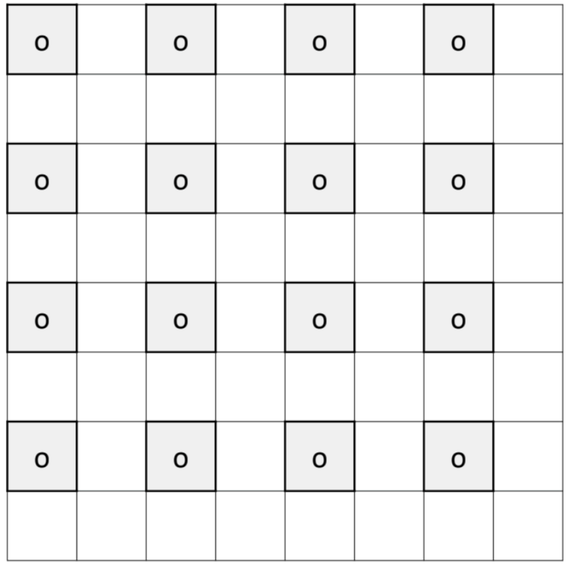


stride = 2

filter



output: H × W = **4 × 4**

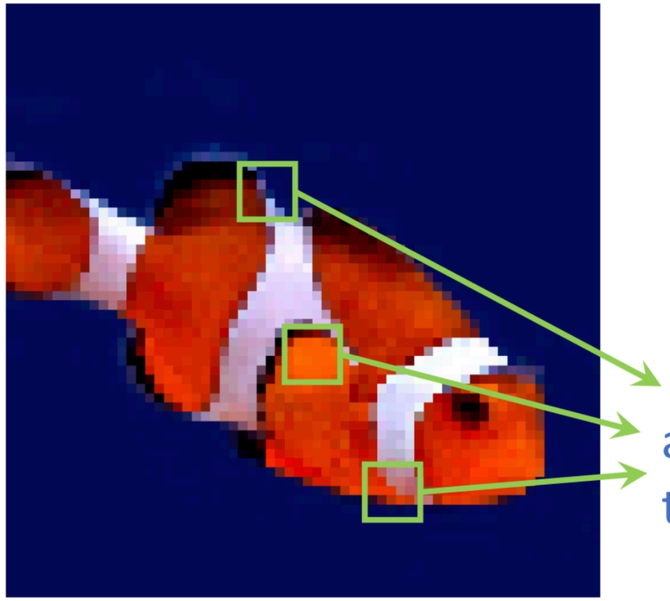


- reduces feature map size
- compress and abstract

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

Convolution: translation-invariance

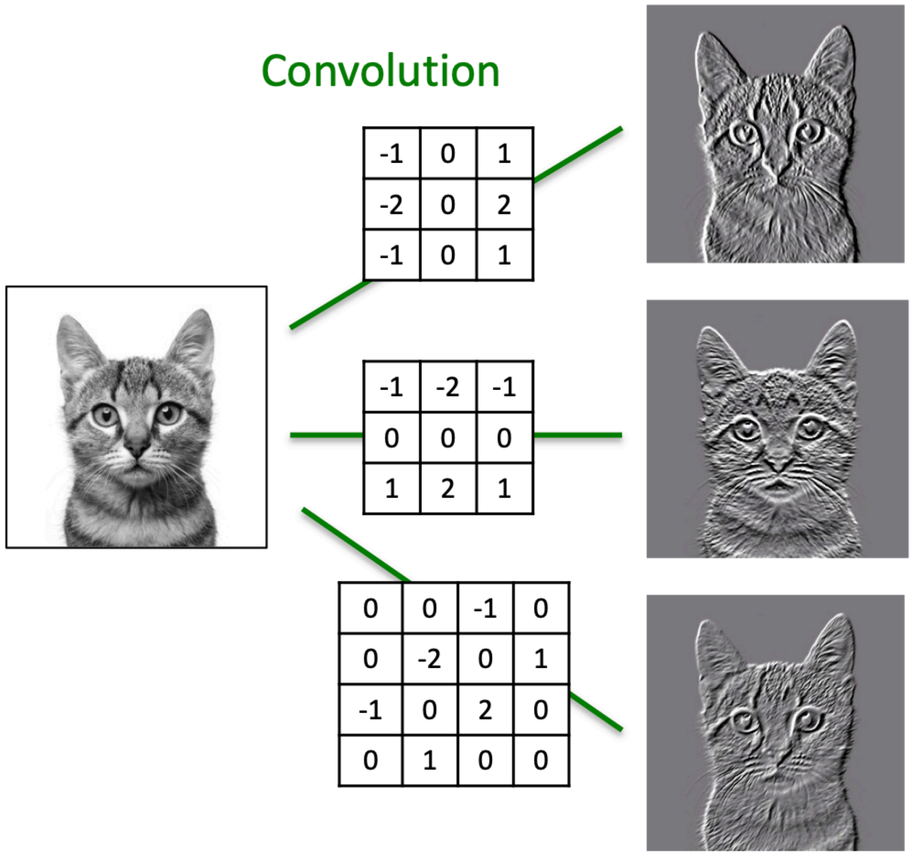
- Process each window in the same way

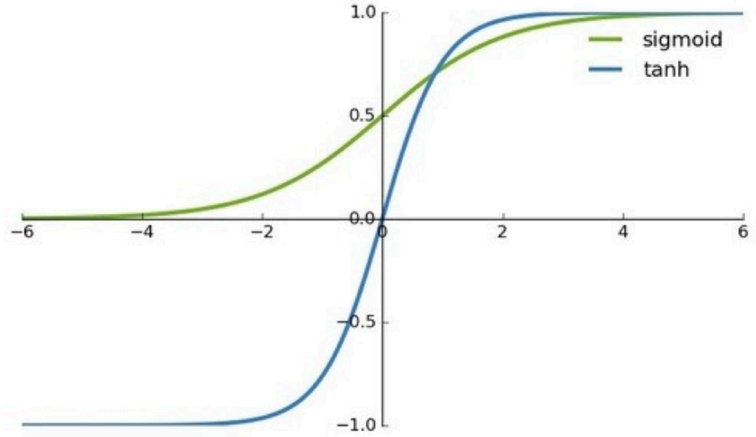


apply the same weights regardless of the window location

Deep Convolutional Networks

[Convolution + Nonlinear activation] + Pooling



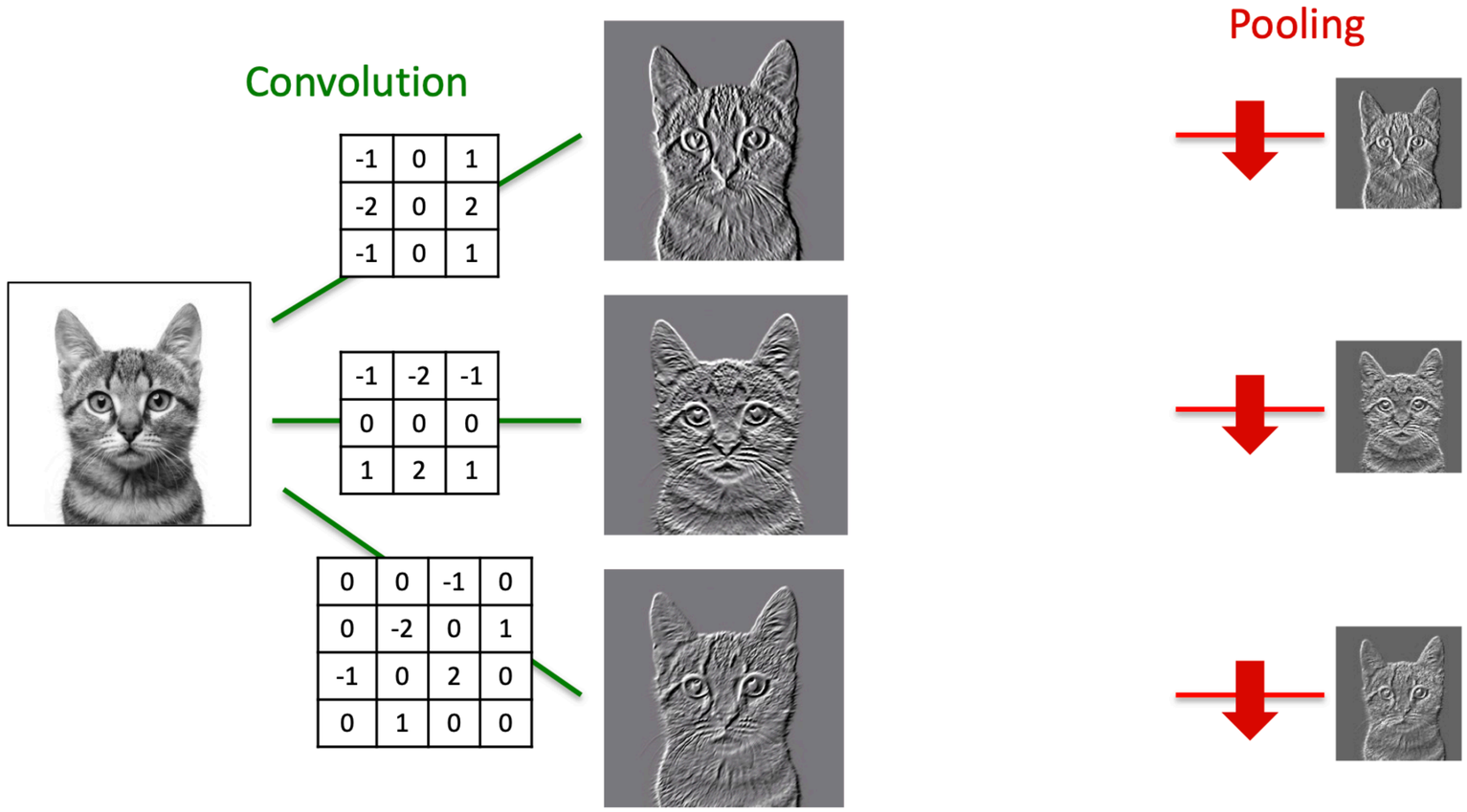


LeNet – tanh activation

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

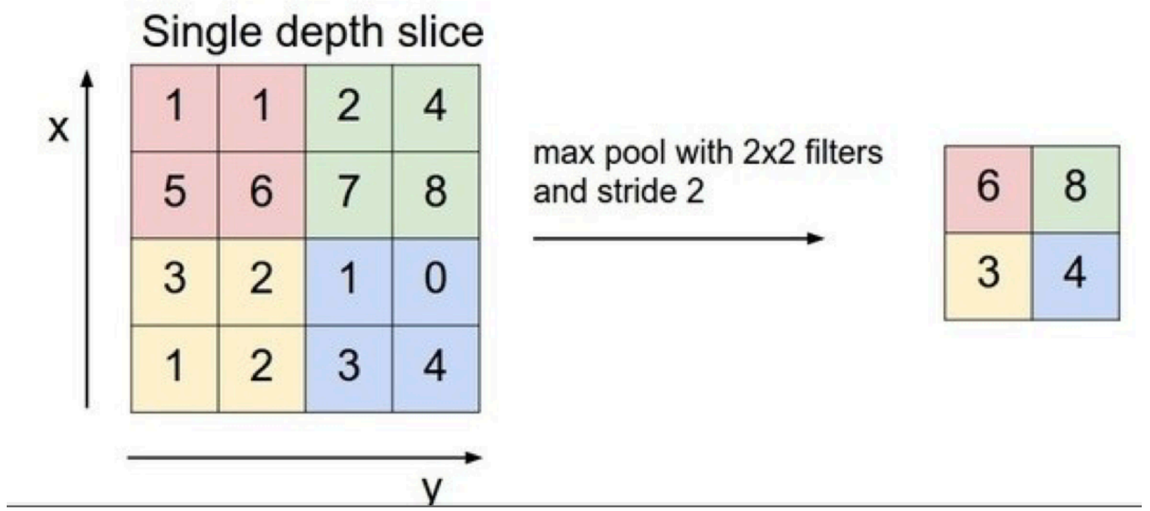
Deep Convolutional Networks

[Convolution + Nonlinear activation] + Pooling

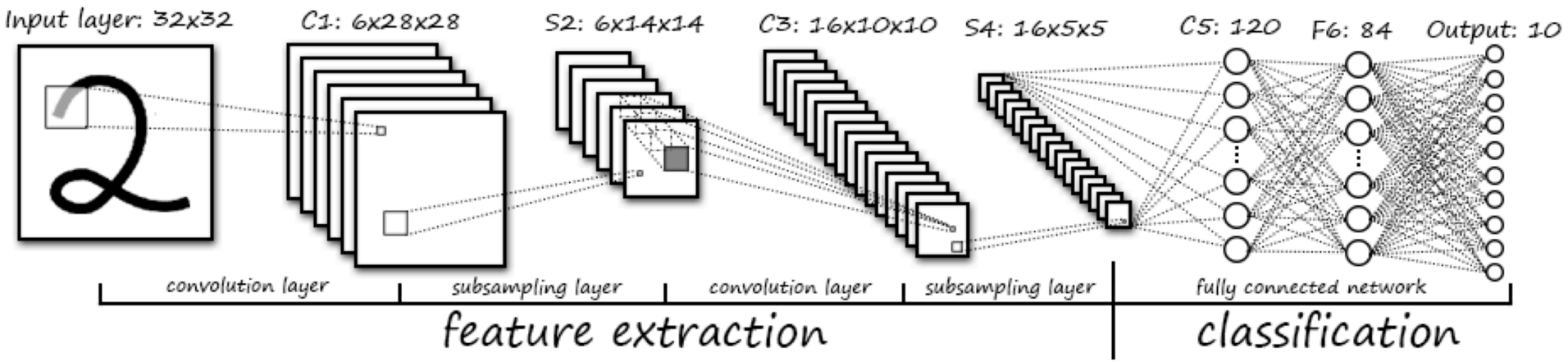


Pooling = Down-sampling

Max pooling



Deep Convolutional Networks



MNIST

60,000 original dataset

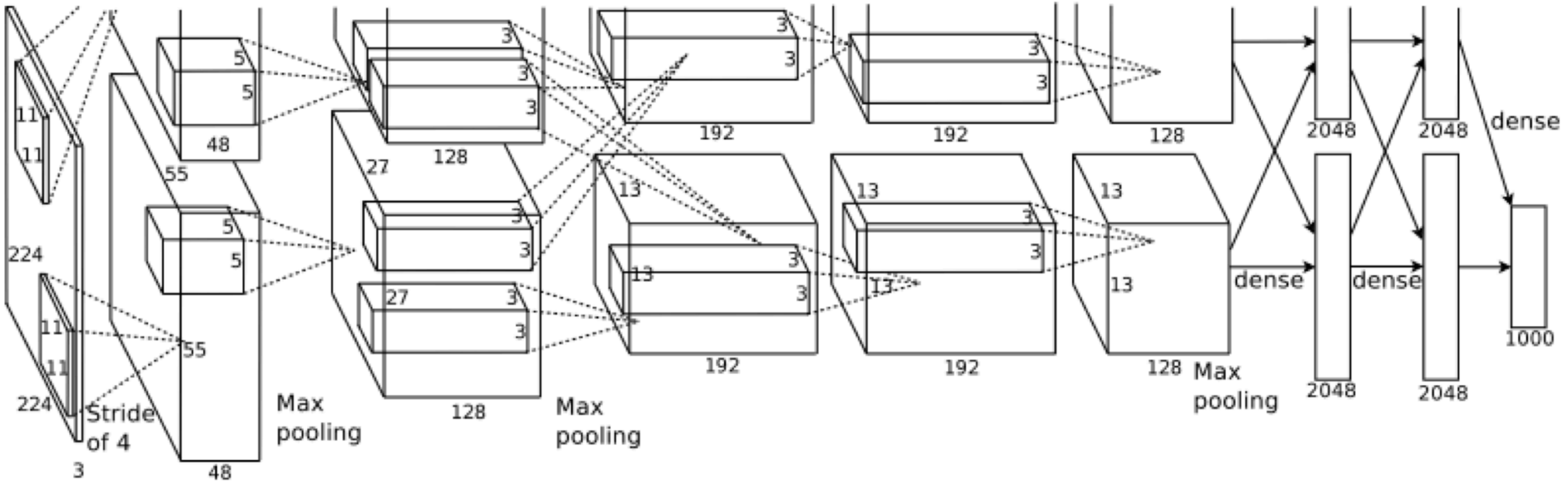
Test error: 0.95%

[1] LeNet 5, LeCun et al. 1998

Misclassified examples on MNIST

True label -> Predicted label

Alex Net



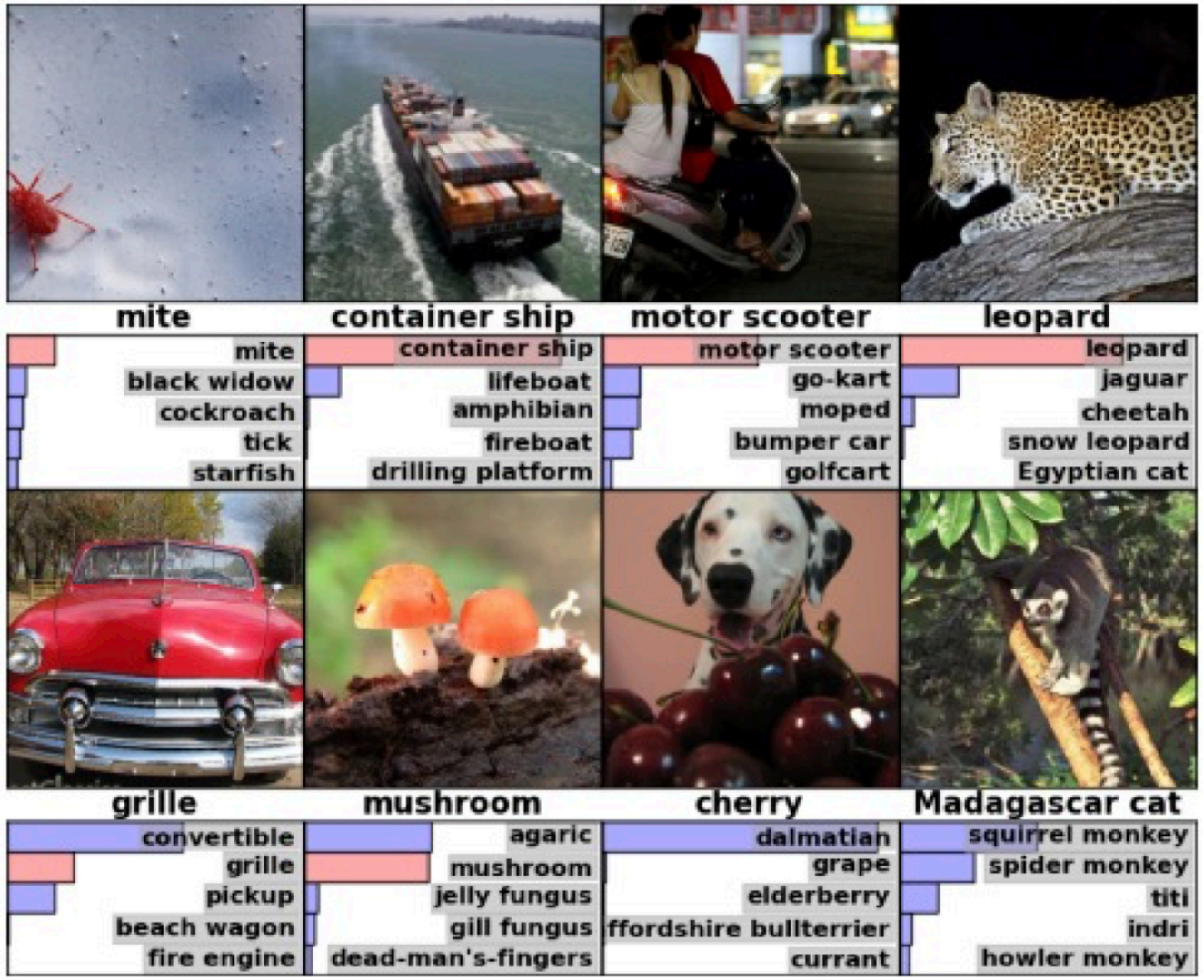
[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NeurIPS 2012.

ImageNet

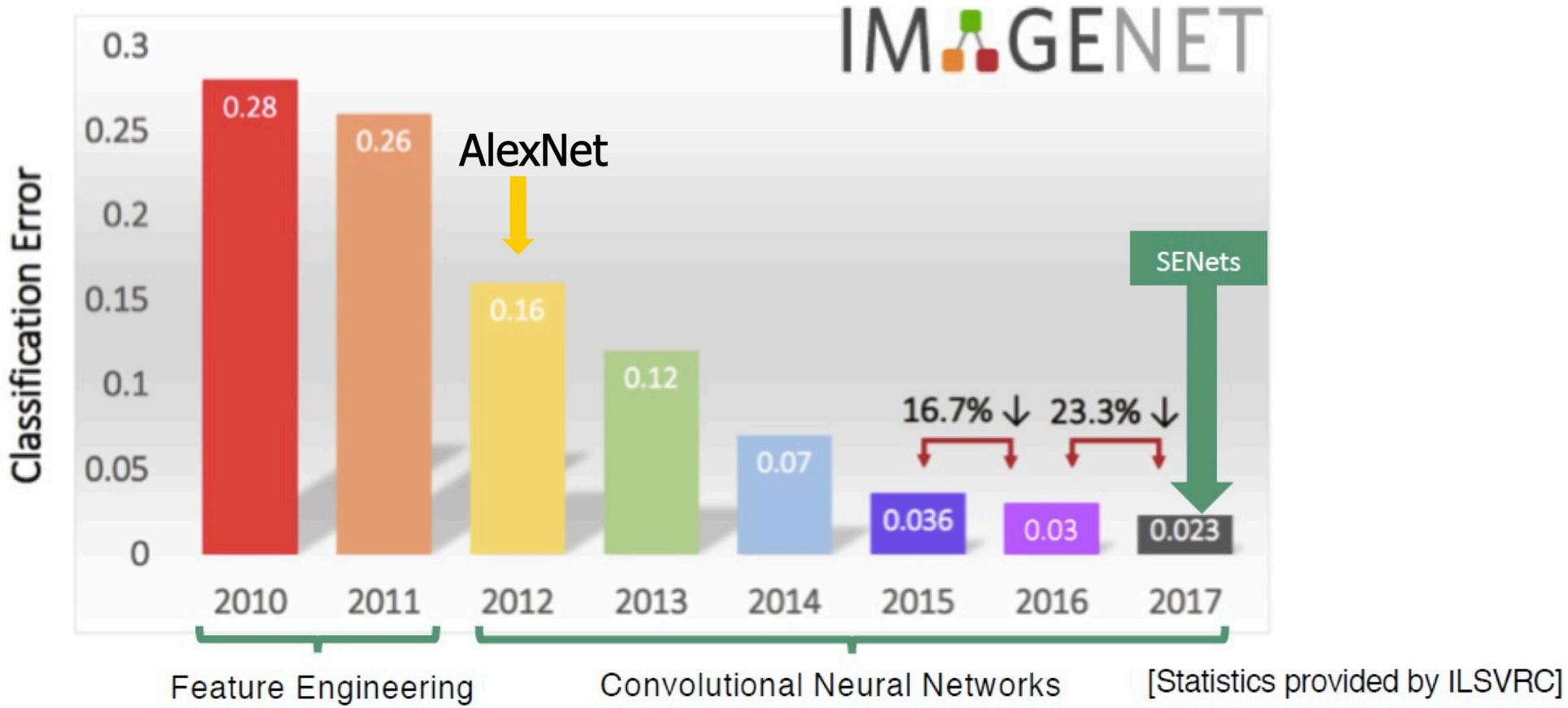
- 15M images
- 22K categories
- Images collected from Web
- Human labelers (Amazon's Mechanical Turk crowd-sourcing)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
  - 1K categories
  - 1.2M training images (~1000 per category)
  - 50,000 validation images
  - 150,000 testing images

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

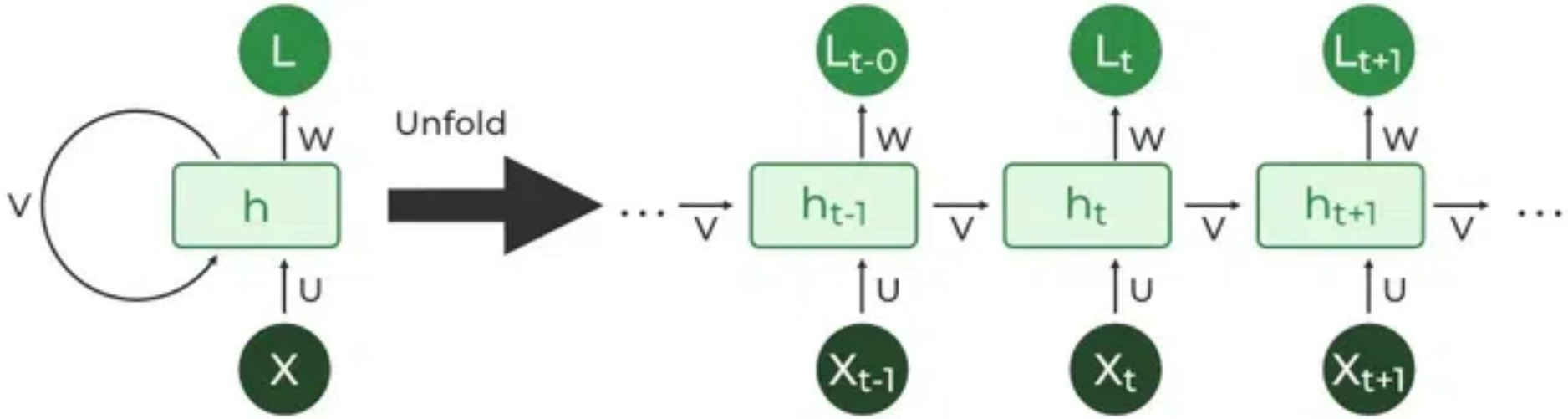
ImageNet Results



ImageNet Results



Recurrent Neural Networks (RNNs)

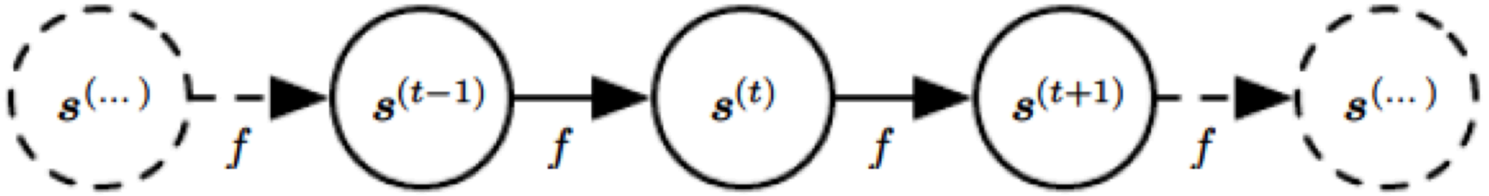


Recurrent Neural Networks

- 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

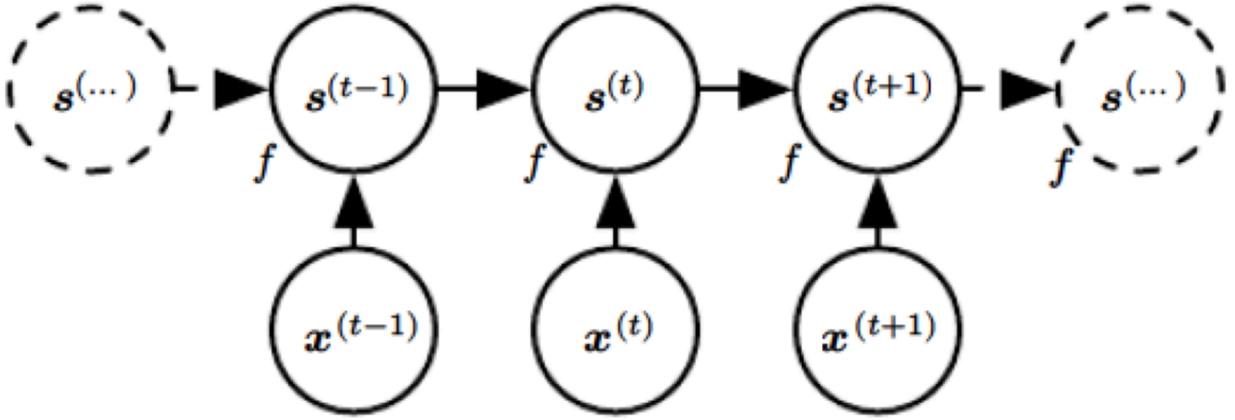


$$s^{(t+1)} = f(s^{(t)}; \theta)$$

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

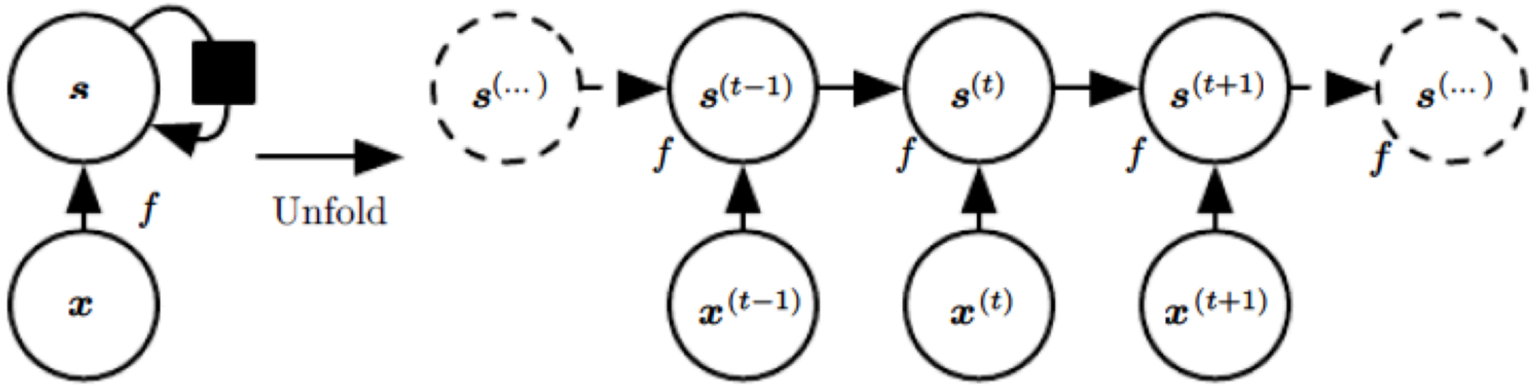
Like Markov model, but here $s^{(t+1)}$ is deterministic given $s^{(t)}$

Computation Graph



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

Compact view



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

Key: the same f and θ for all time steps

Like stationary HMM

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

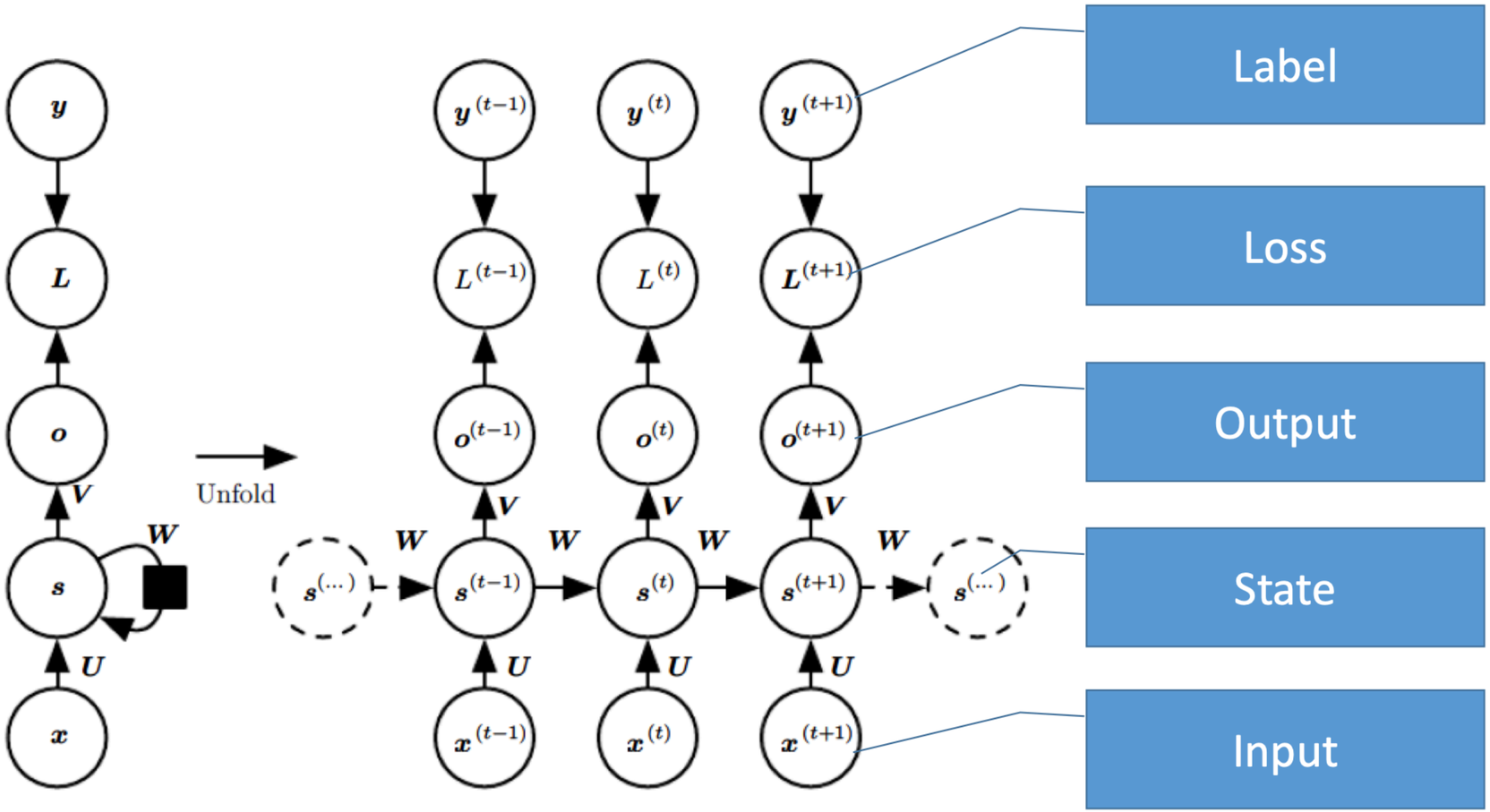
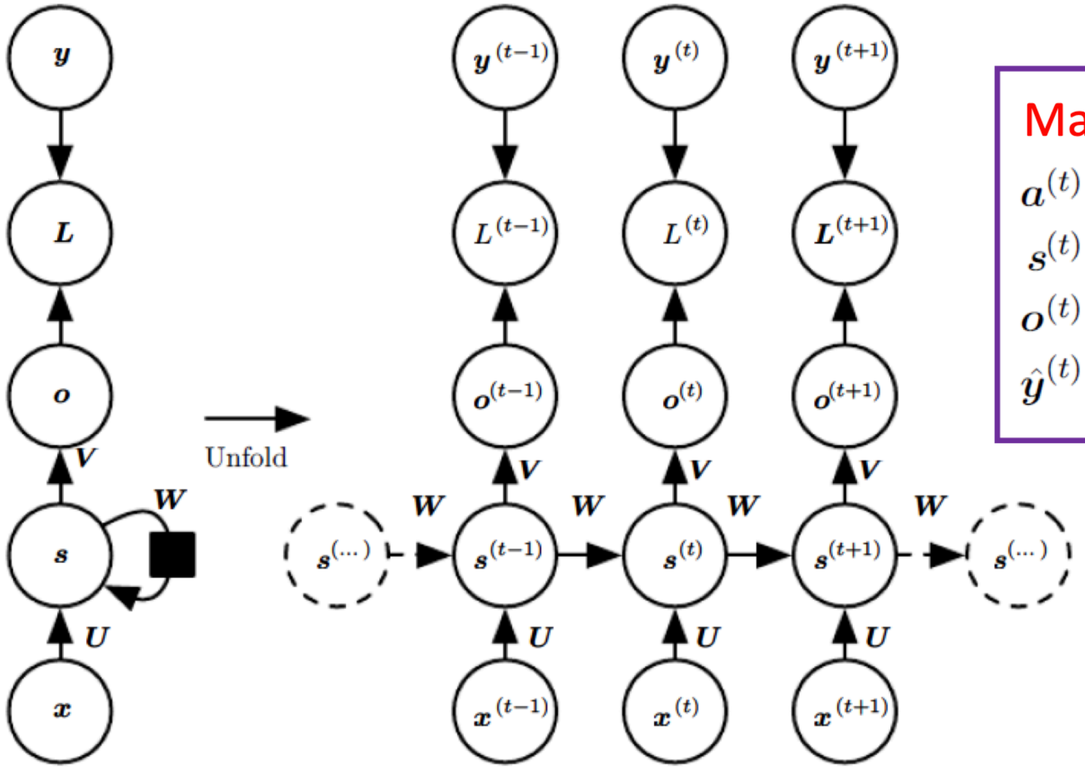


Figure from *Deep Learning*, by Goodfellow, Bengio and Courville

Recurrent Neural Networks



Math formula:

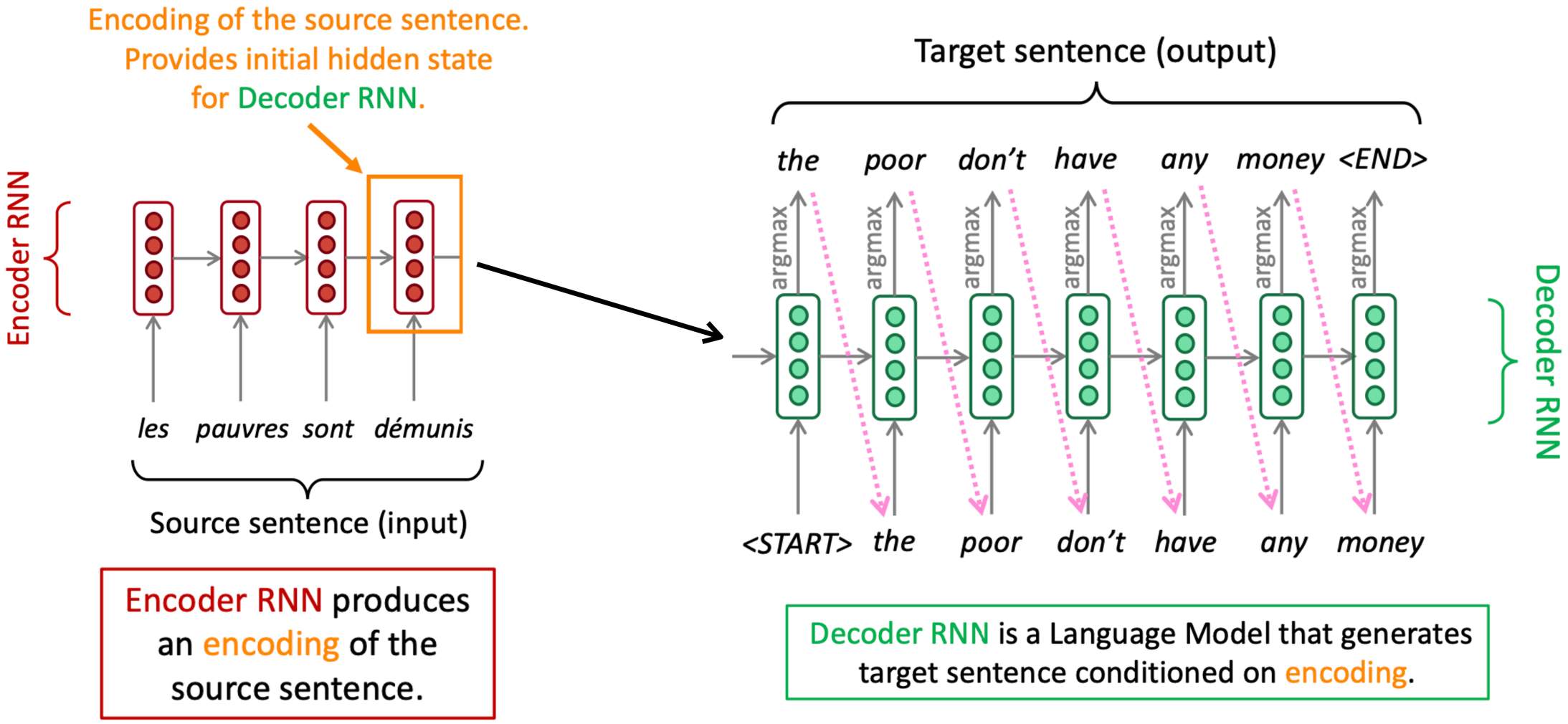
$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{s}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$
$$\boldsymbol{s}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{s}^{(t)}$$
$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$$

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



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

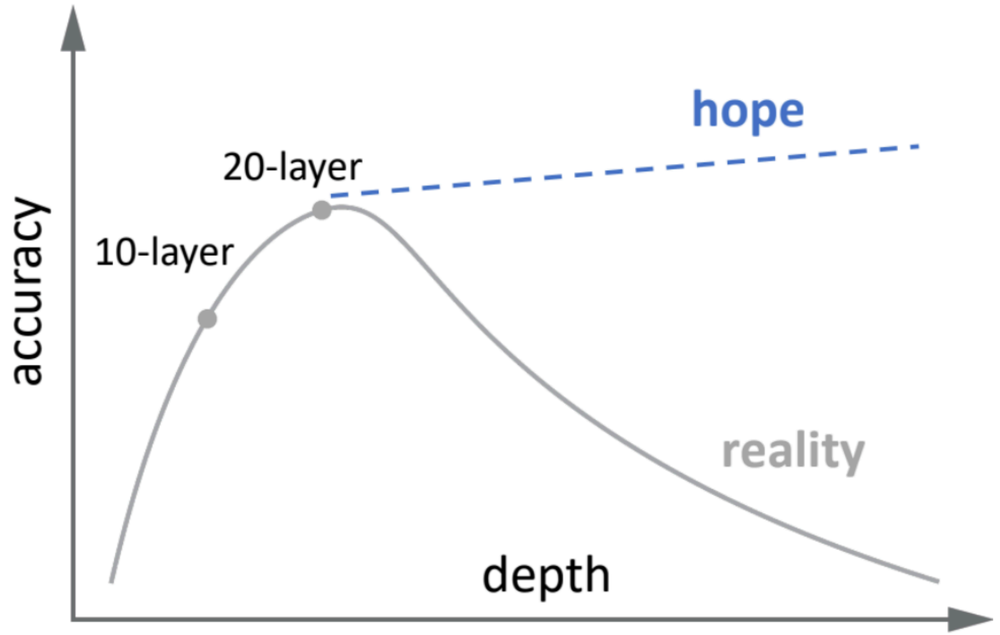
Normalization modules

Deep residual learning

The Degradation Problem

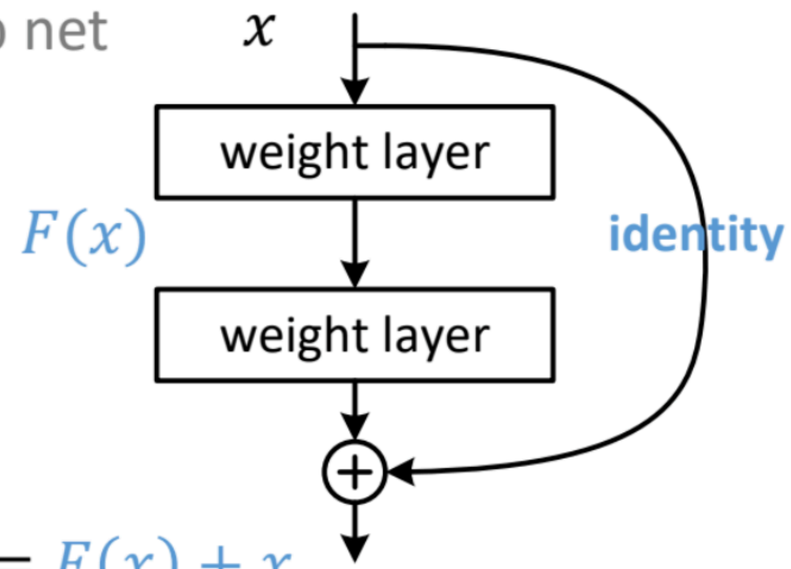
- Good init + norm enable training deeper models
- Simply stacking more layers?

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



Deep Residual Learning

a subnet in
a deep net



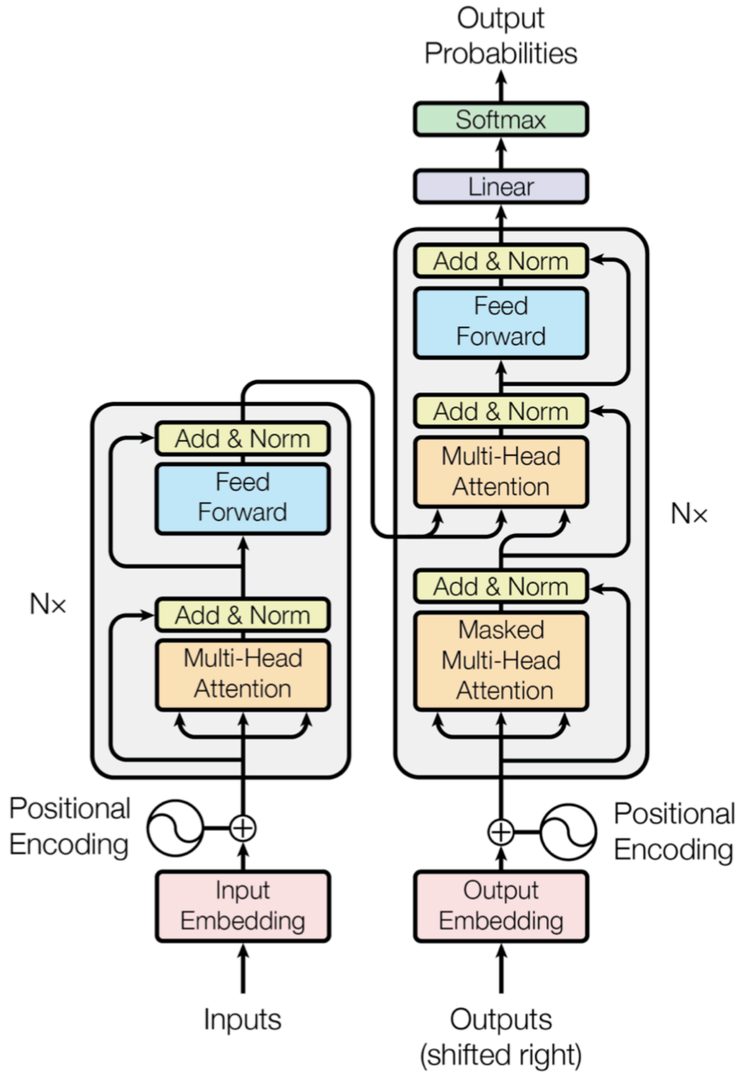
$$H(x) = F(x) + x$$

Deep Residual Networks (ResNet)

ResNet-152

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

Transformers



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