# CS540 Introduction to Artificial Intelligence

# **Convolutional Neural Networks (I)**

Yudong Chen
University of Wisconsin-Madison

**November 2, 2021**

Slides created by Sharon Li [modified by Yudong Chen]

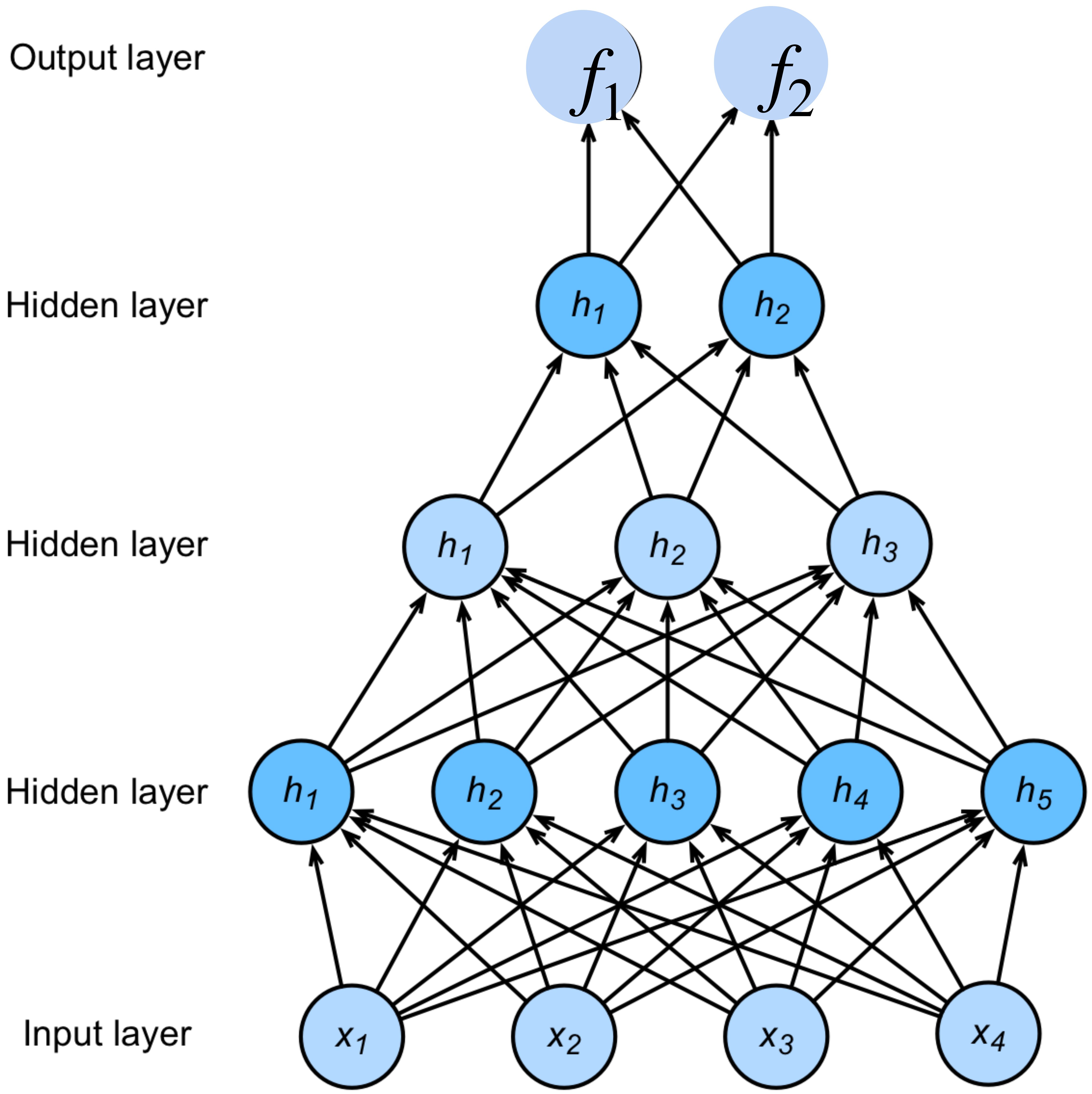
Congrats on getting midterm done!

**Reminder**: HW6 is due this Thursday

# Outline

- Intro of convolutional computations
    - 2D convolution
    - Padding, stride etc
    - Multiple input and output channels
    - Pooling
- Basic Convolutional Neural Networks
    - LeNet

# Review: Deep neural networks (DNNs)




$$\mathbf{h}^{(1)} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}^{(2)} = \sigma(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}_2)$$

$$\mathbf{h}^{(3)} = \sigma(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}_3)$$

$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}^{(3)} + \mathbf{b}_4$$

$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

NNs are composition of nonlinear functions

# How to classify

## Cats vs. dogs?

Dual

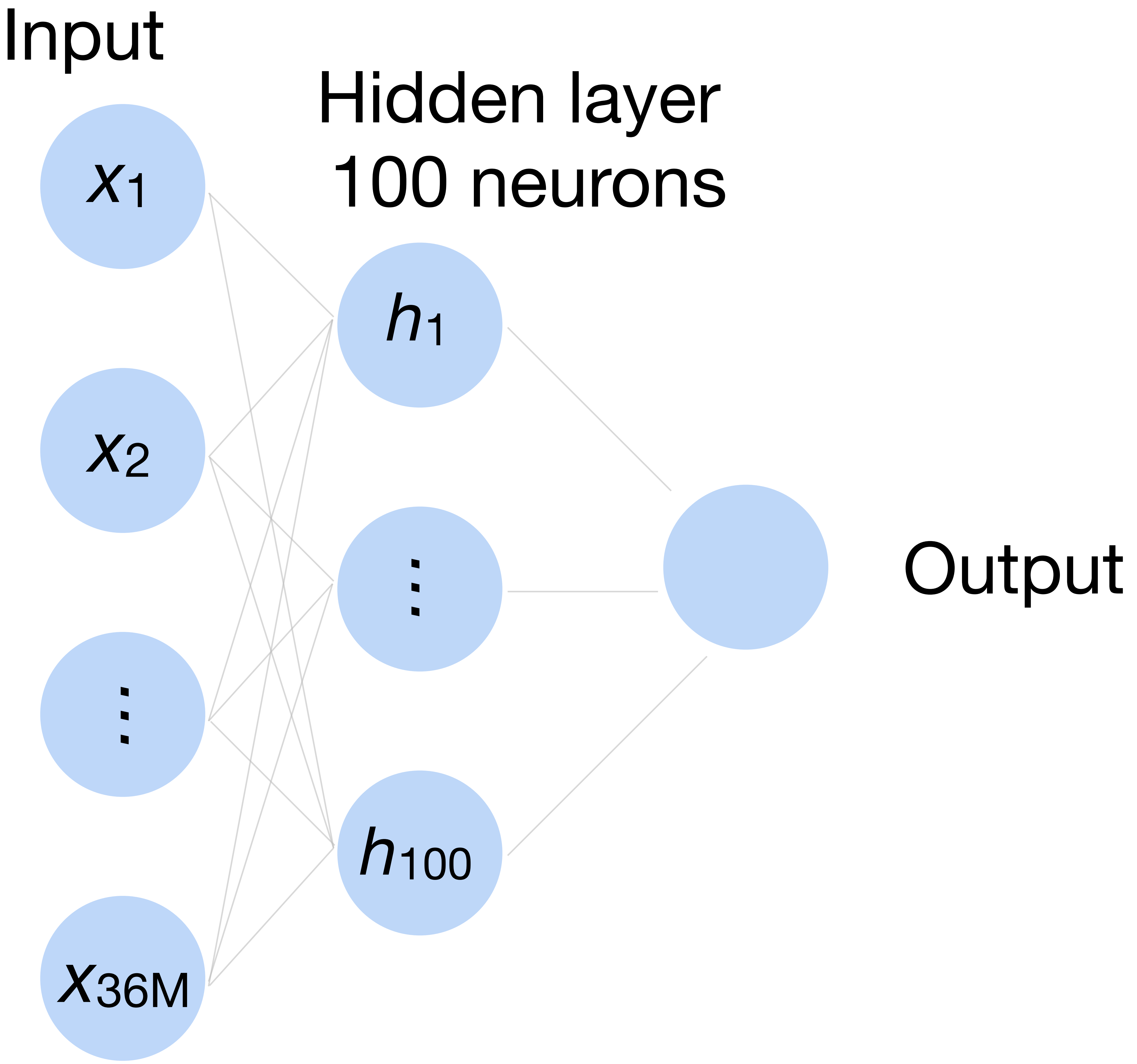
**12MP**

wide-angle and
telephoto cameras

**36M** numbers in a RGB image!

# Fully Connected Networks

**Cats vs. dogs?**




~ 36M elements x 100 = ~**3.6B** parameters!

# Convolutions come to rescue!

# Why Convolution?

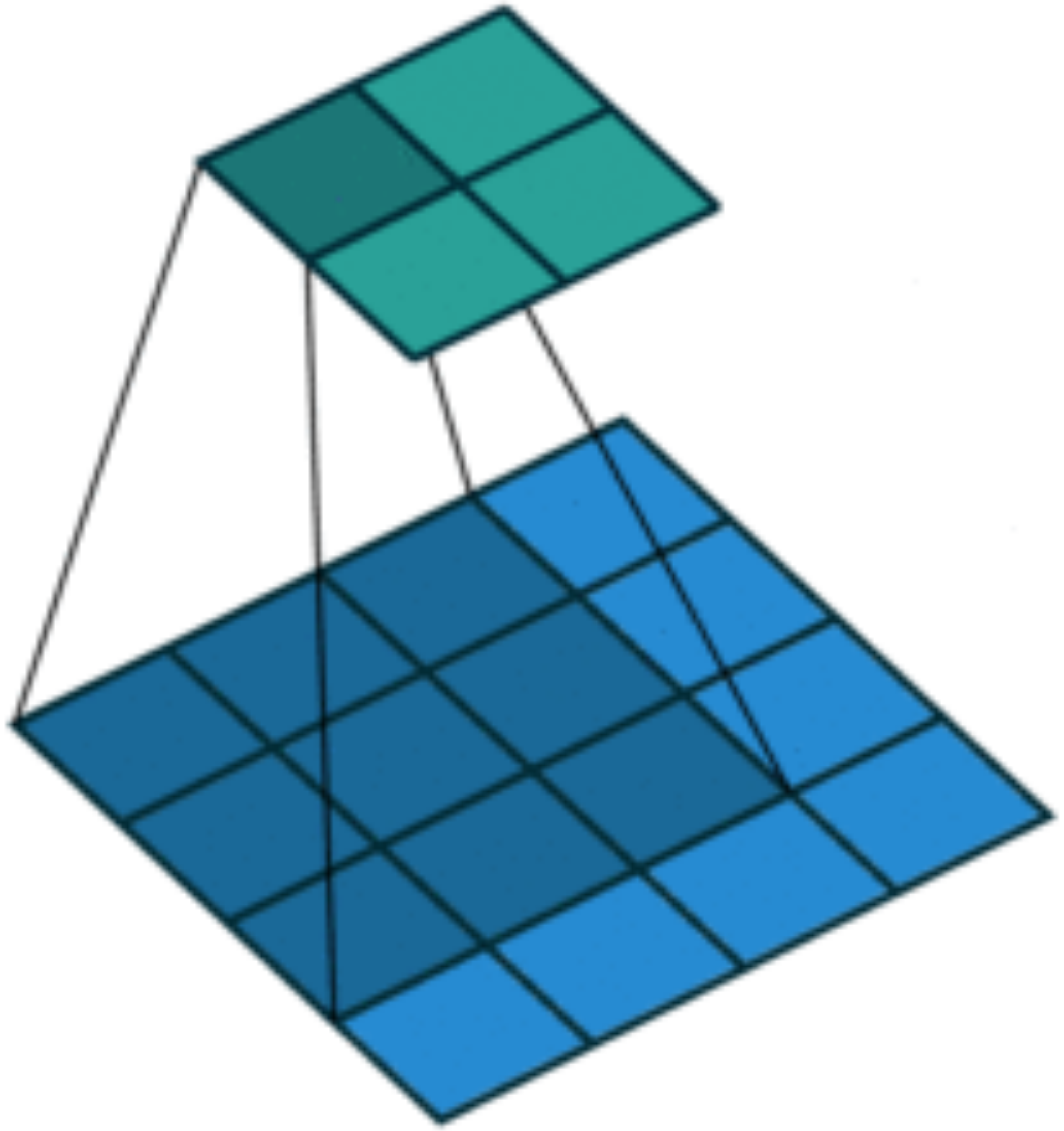
Translation Invariance

Locality

# 2-D Convolution

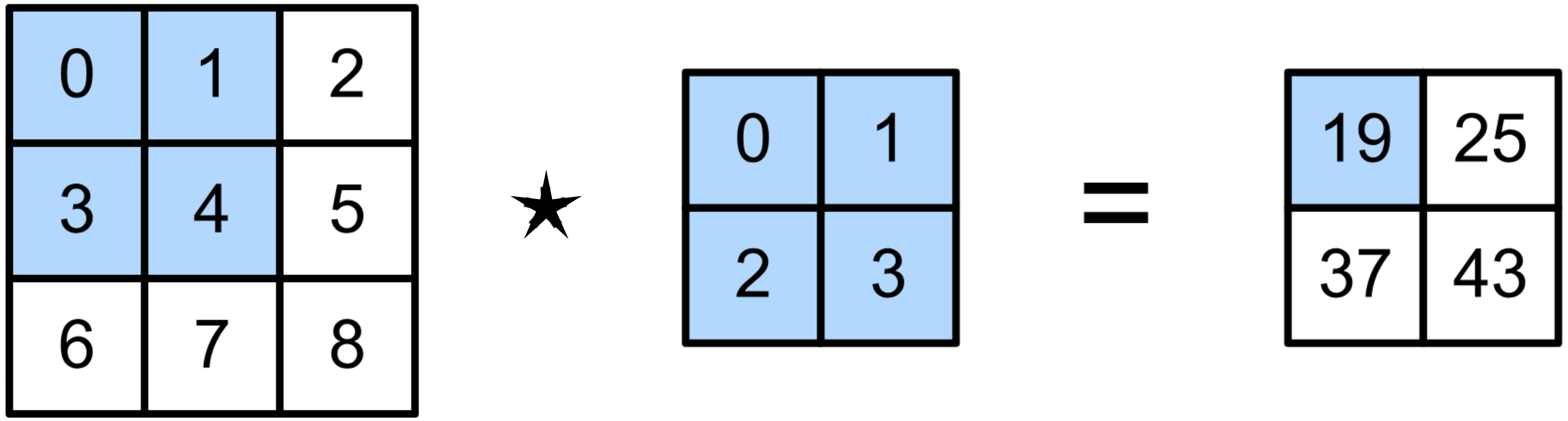
$$\underset{\text{Input}}{\begin{bmatrix}0 & 1 & 2\\ 3 & 4 & 5\\ 6 & 7 & 8\end{bmatrix}} \star \underset{\text{Kernel}}{\begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}} = \underset{\text{Output}}{\begin{bmatrix}19 & 25\\ 37 & 43\end{bmatrix}}$$

$$\begin{aligned}
0\times 0+1\times 1+3\times 2+4\times 3 &= 19,\\
1\times 0+2\times 1+4\times 2+5\times 3 &= 25,\\
3\times 0+4\times 1+6\times 2+7\times 3 &= 37,\\
4\times 0+5\times 1+7\times 2+8\times 3 &= 43.
\end{aligned}$$



(vdumoulin@ Github)

# 2-D Convolution Layer

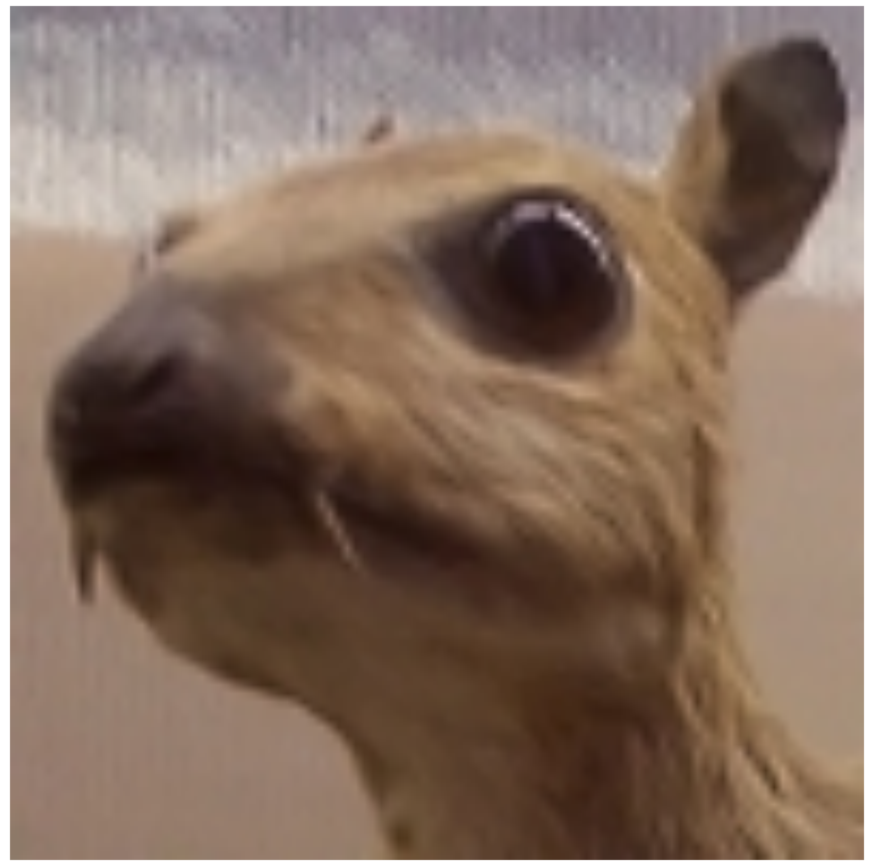



- $\mathbf{X} : n_h \times n_w$ input matrix
- $\mathbf{W} : k_h \times k_w$ kernel matrix
- $b$: scalar bias
- $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- **W** and $b$ are learnable parameters

# Examples

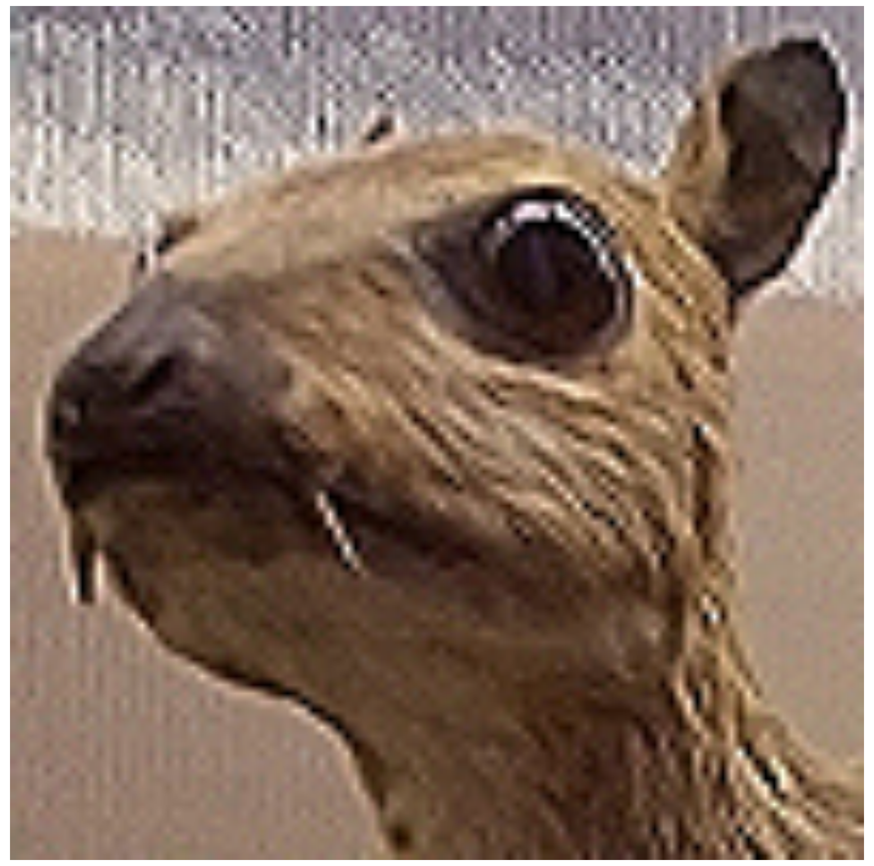


(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Sharpen

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

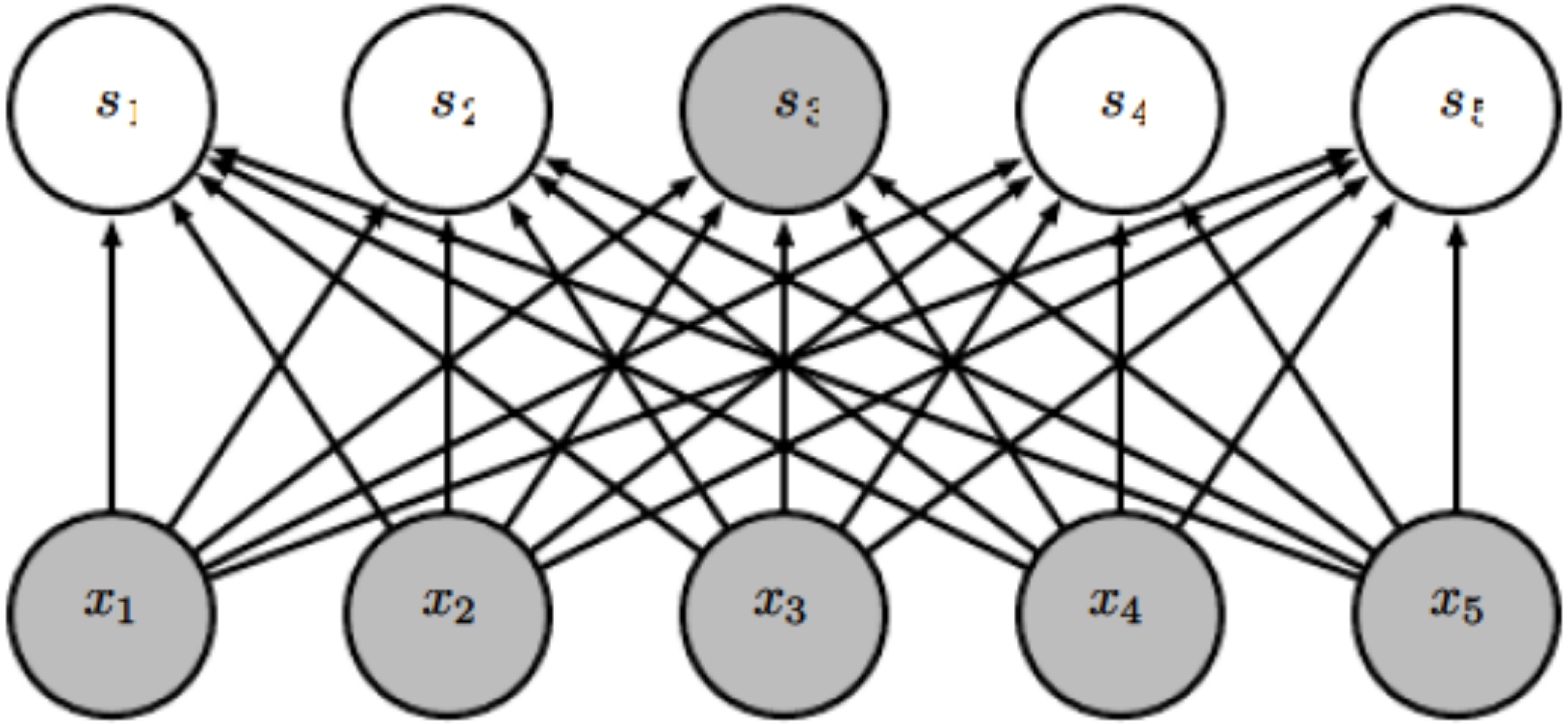
# Convolutional Neural Networks

- Strong empirical application performance

- Convolutional networks: neural networks that use convolution in place of general matrix multiplication in at least one of their layers

# FCNet vs ConvNet: dense vs sparse interaction

Fully connected layer, $m \times n$ edges



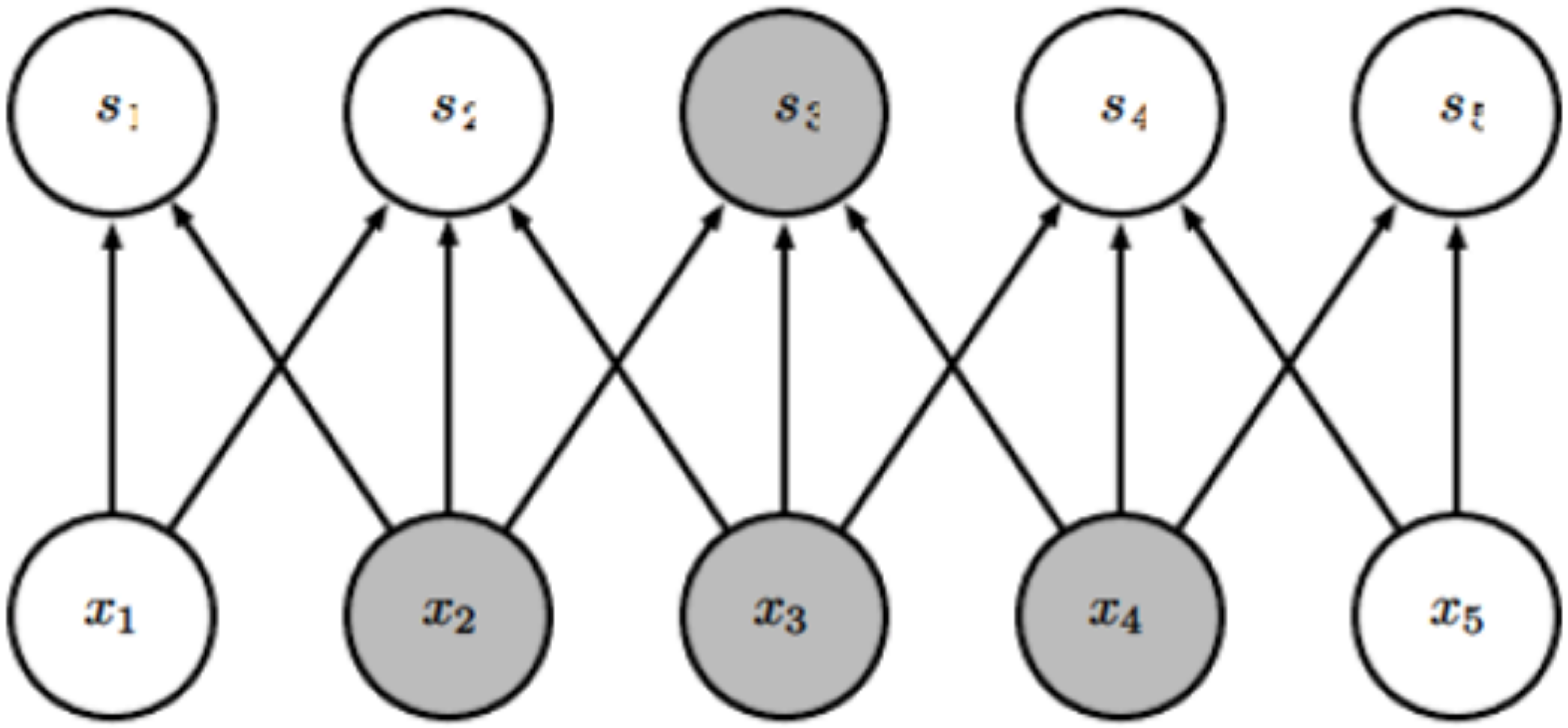

$m$ output nodes

$n$ input nodes

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

# FCNet vs ConvNet: dense vs sparse interaction

Convolutional layer, $\leq m \times k$ edges




$m$ output nodes

$k$ kernel size

$n$ input nodes

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

# Efficiency of Convolution

- Input size: 320 x 280
- Kernel Size: 2 x 1
- Output size: 319 x 280

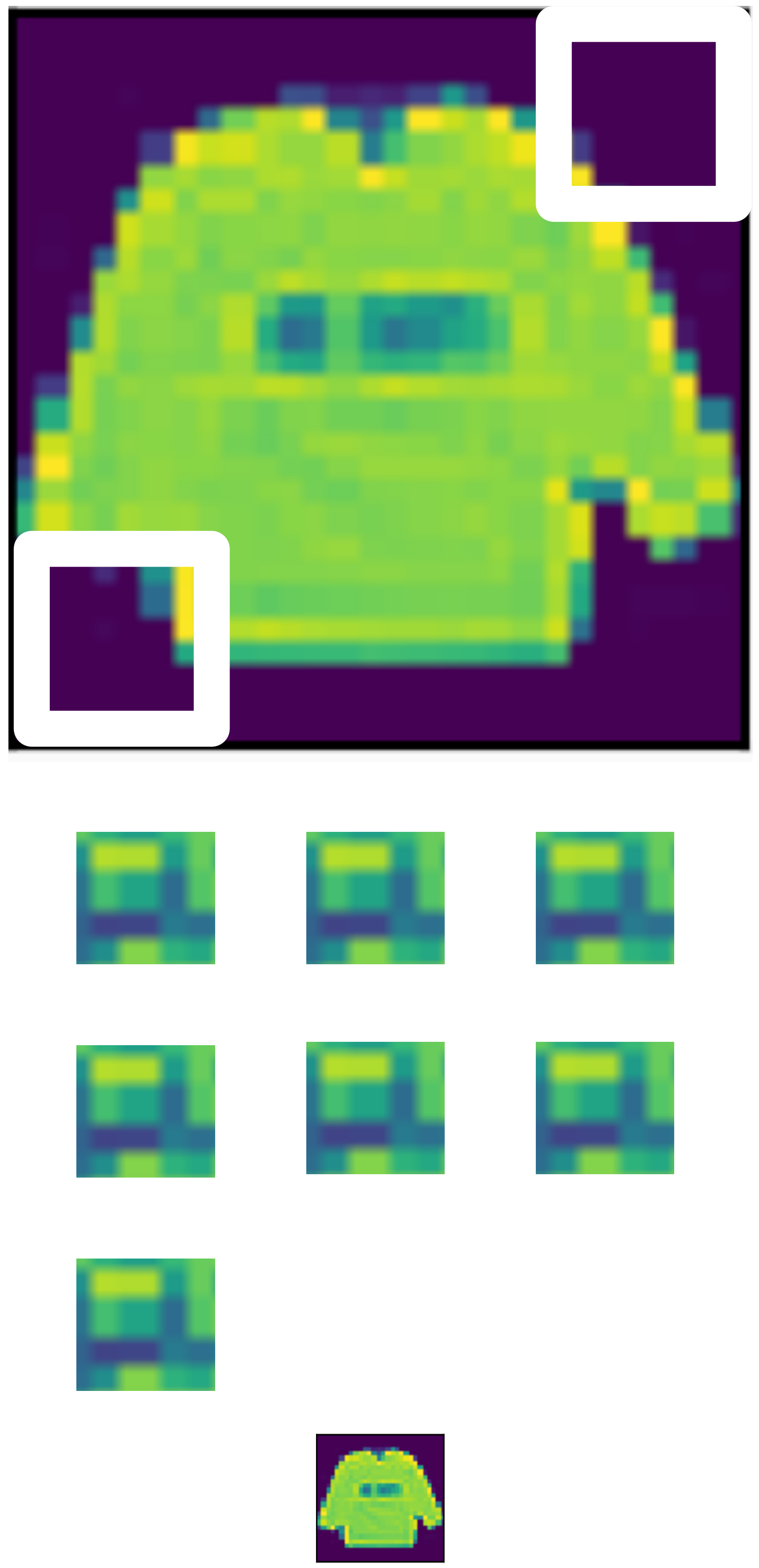
| | Convolution | Dense matrix |
|---|---|---|
| Stored floats | 2 | 319*280*320*280 $> 8e9$ |
| Float muls or adds | 319*280*3 = 267,960 | $> 16e9$ |

# Padding and Stride

# Padding

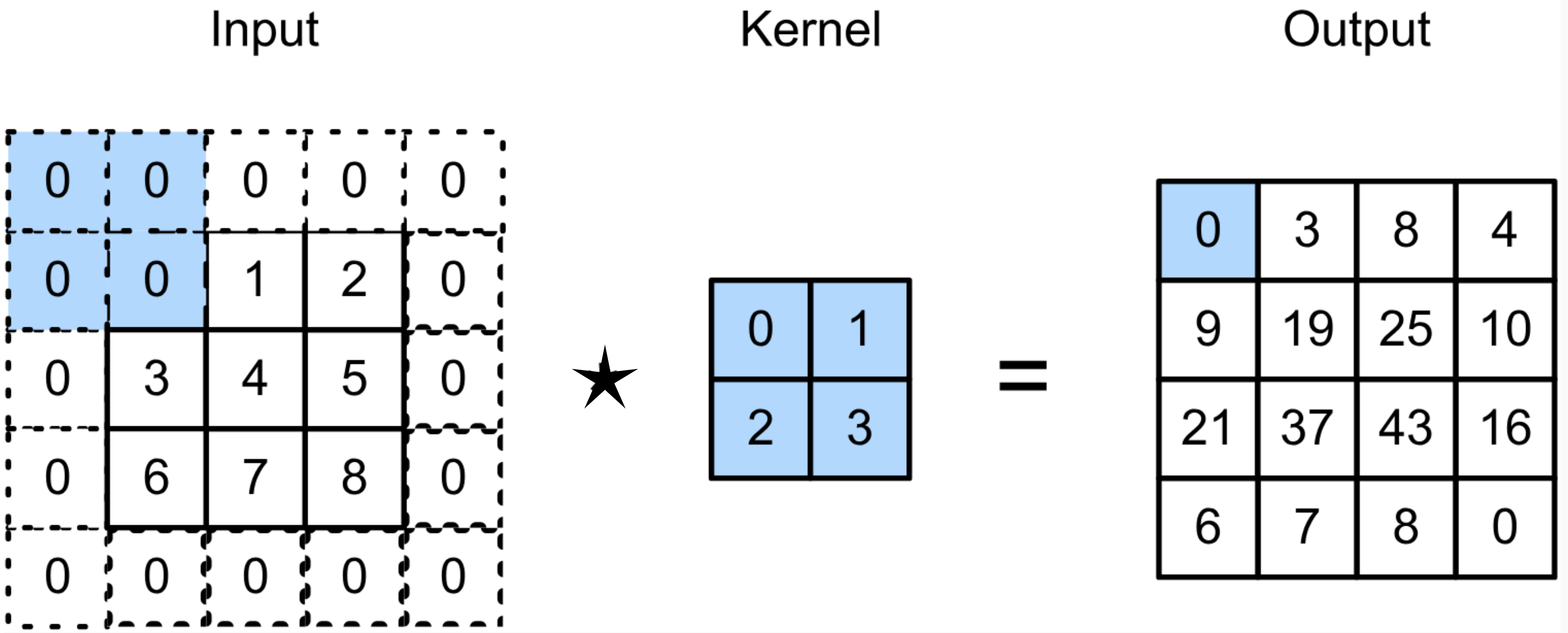
- Given a 32 x 32 input image
- Apply convolution with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 24 x 24 output with 2 layers
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to

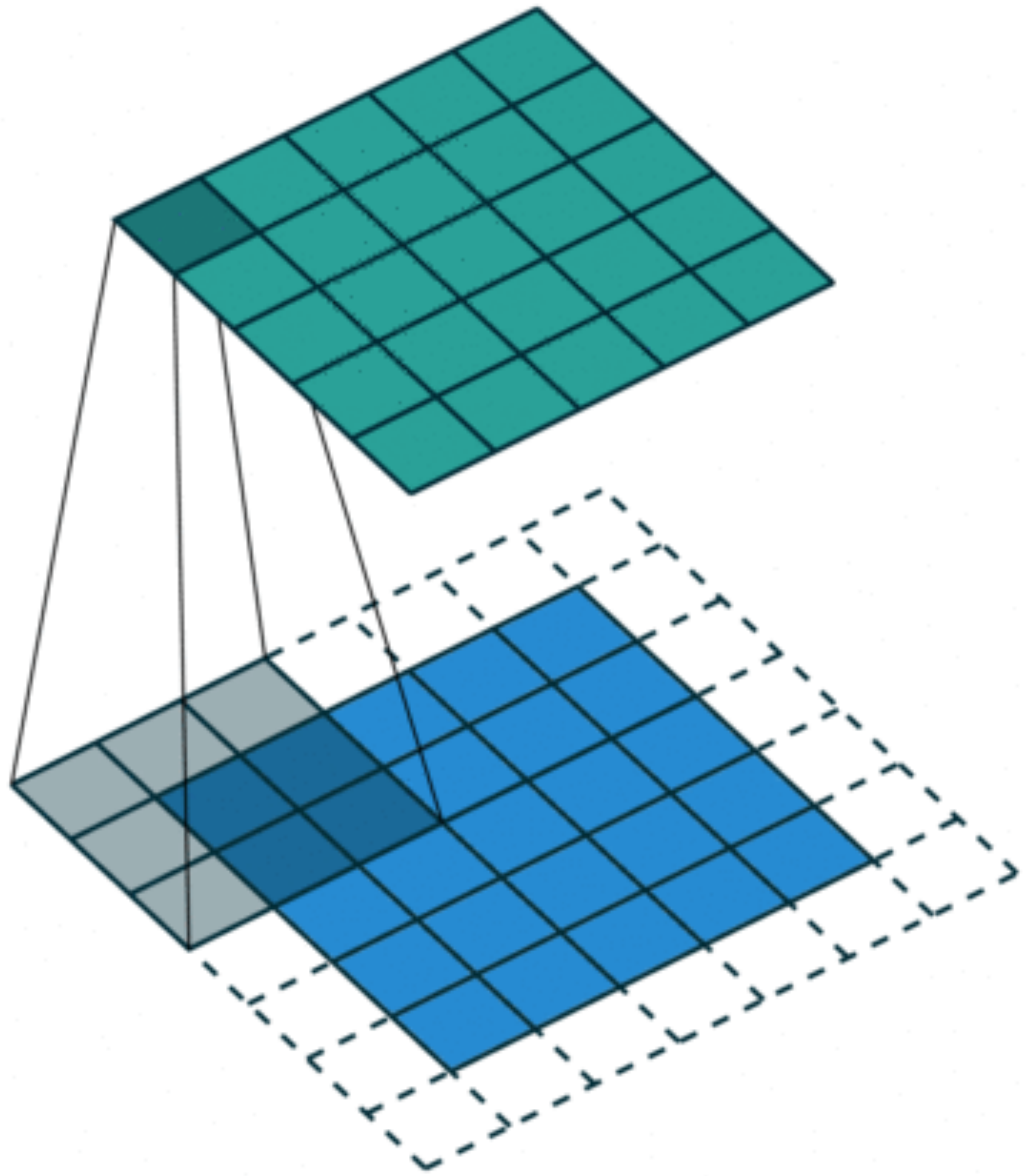
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$



# Padding

Padding adds rows/columns around input



# Padding

- Padding $p_h$ rows and $p_w$ columns, output shape will be

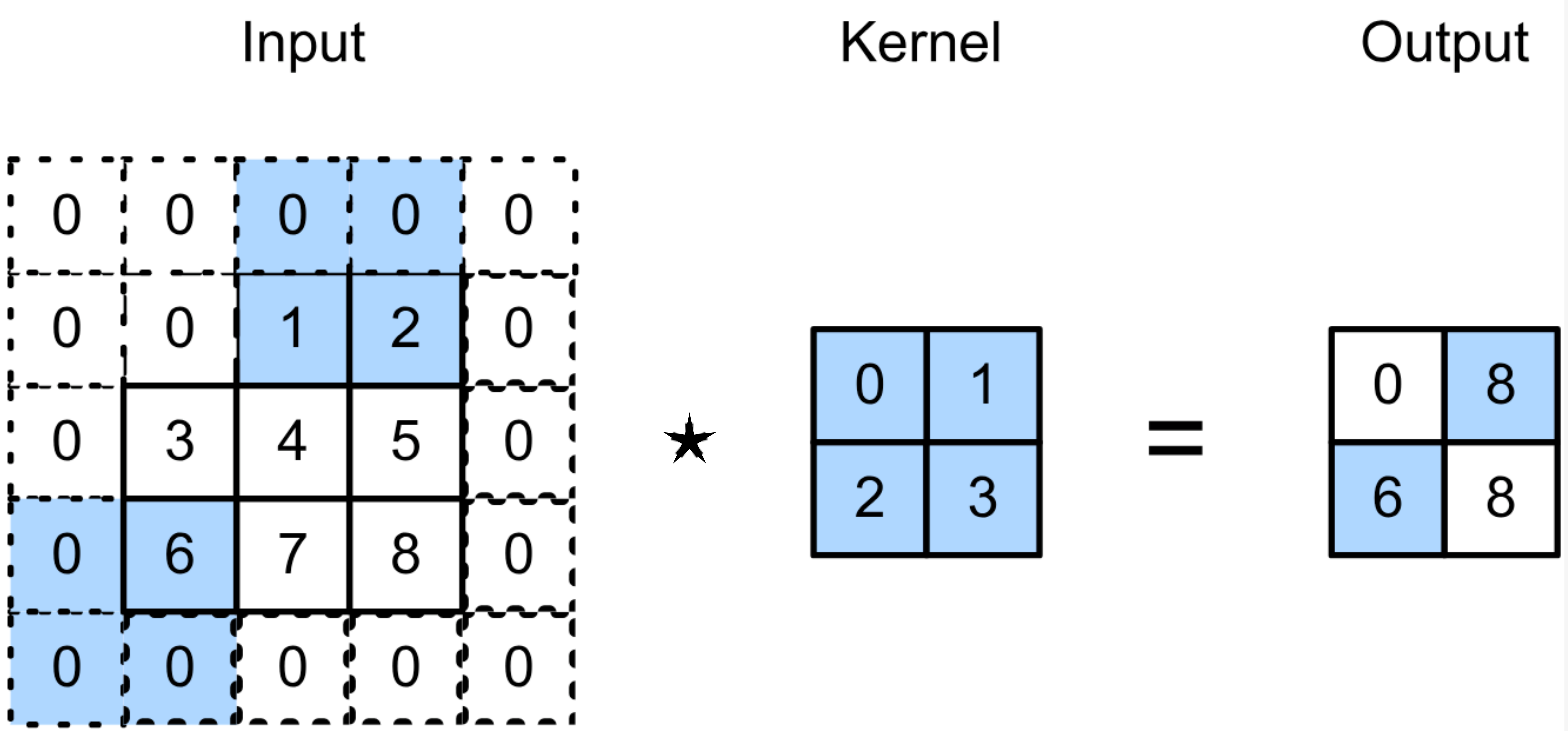
$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- A common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$
  - Odd $k_h$ : pad $p_h/2$ on both sides
  - Even $k_h$ : pad $\lceil p_h/2 \rceil$ on top, $\lfloor p_h/2 \rfloor$ on bottom

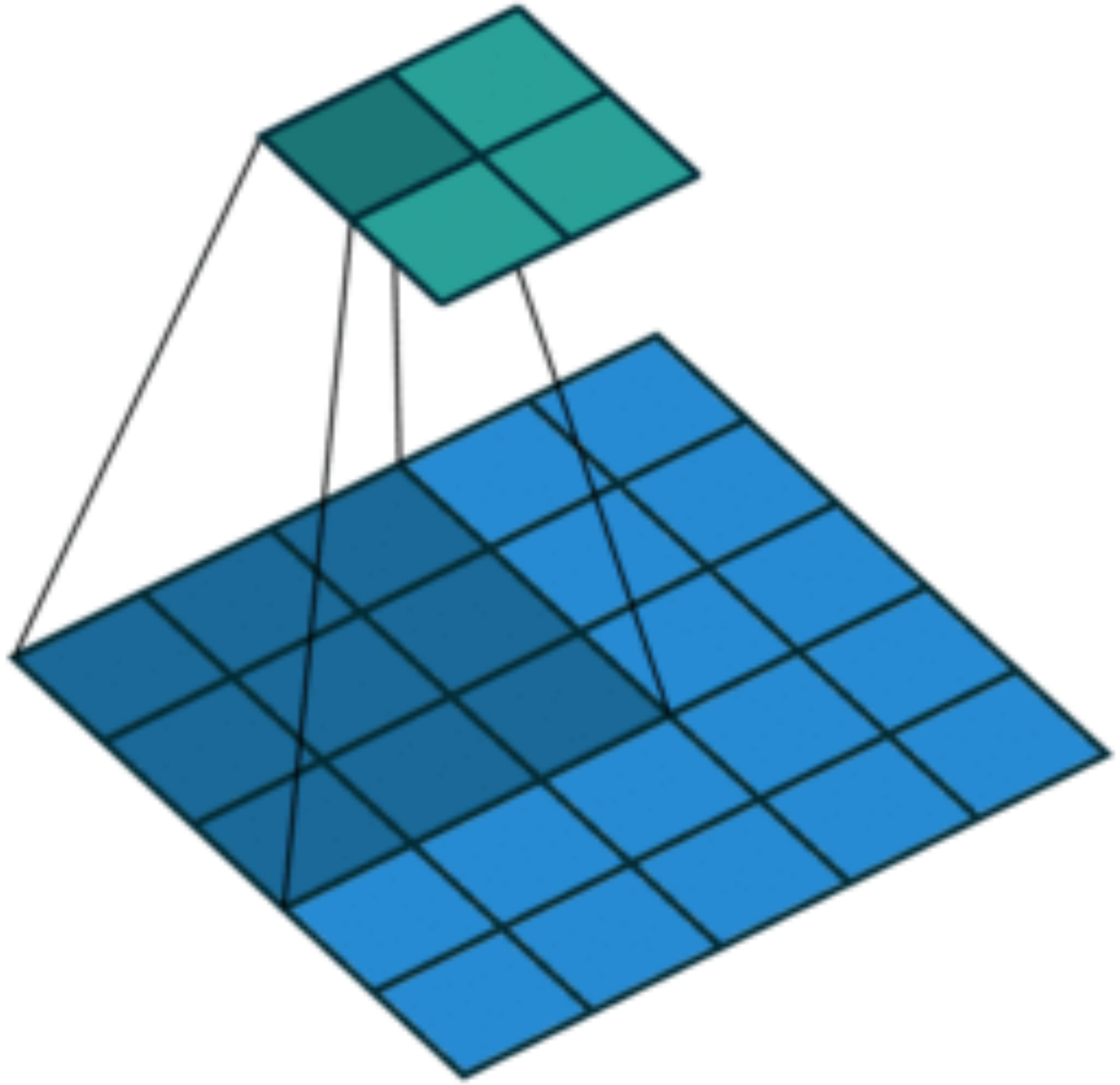
# Stride

- Convolution: slide over **1** row/column each time
- **Stride:** slide over **multiple** rows/columns each time

Strides of 3 and 2 for height and width




$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



# Stride

- Given stride $s_h$ for the height and stride $s_w$ for the width, the output shape is

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- With $p_h = k_h - 1$ and $p_w = k_w - 1$

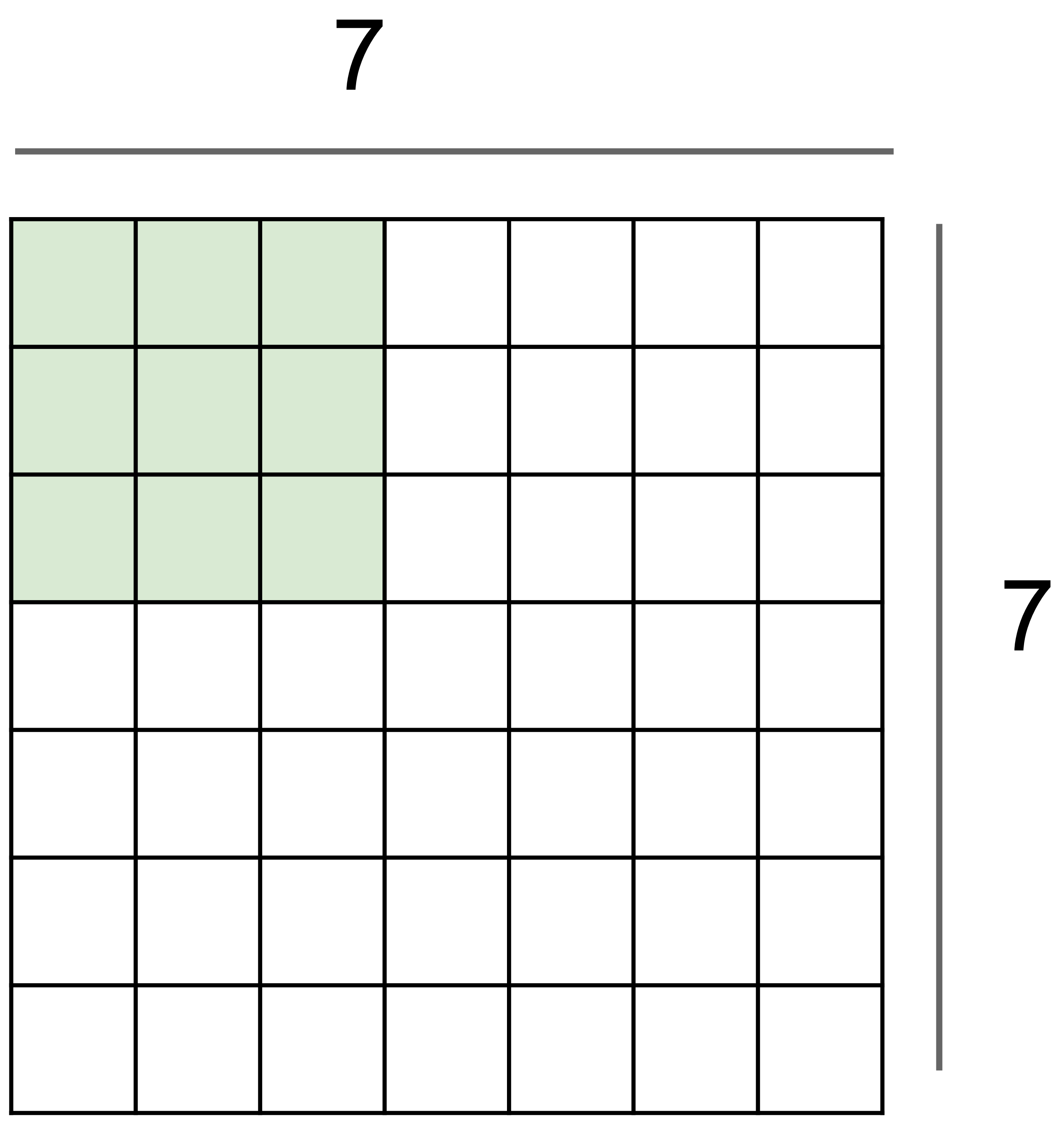
$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- If input height/width are divisible by strides

$$(n_h/s_h) \times (n_w/s_w)$$

Q1. Suppose we want to perform convolution on a single channel image of size 7x7 (no padding) with a kernel of size 3x3, and stride = 2. What is the dimension of the output?

A. 3x3
B. 7x7
C. 5x5
D. 2x2



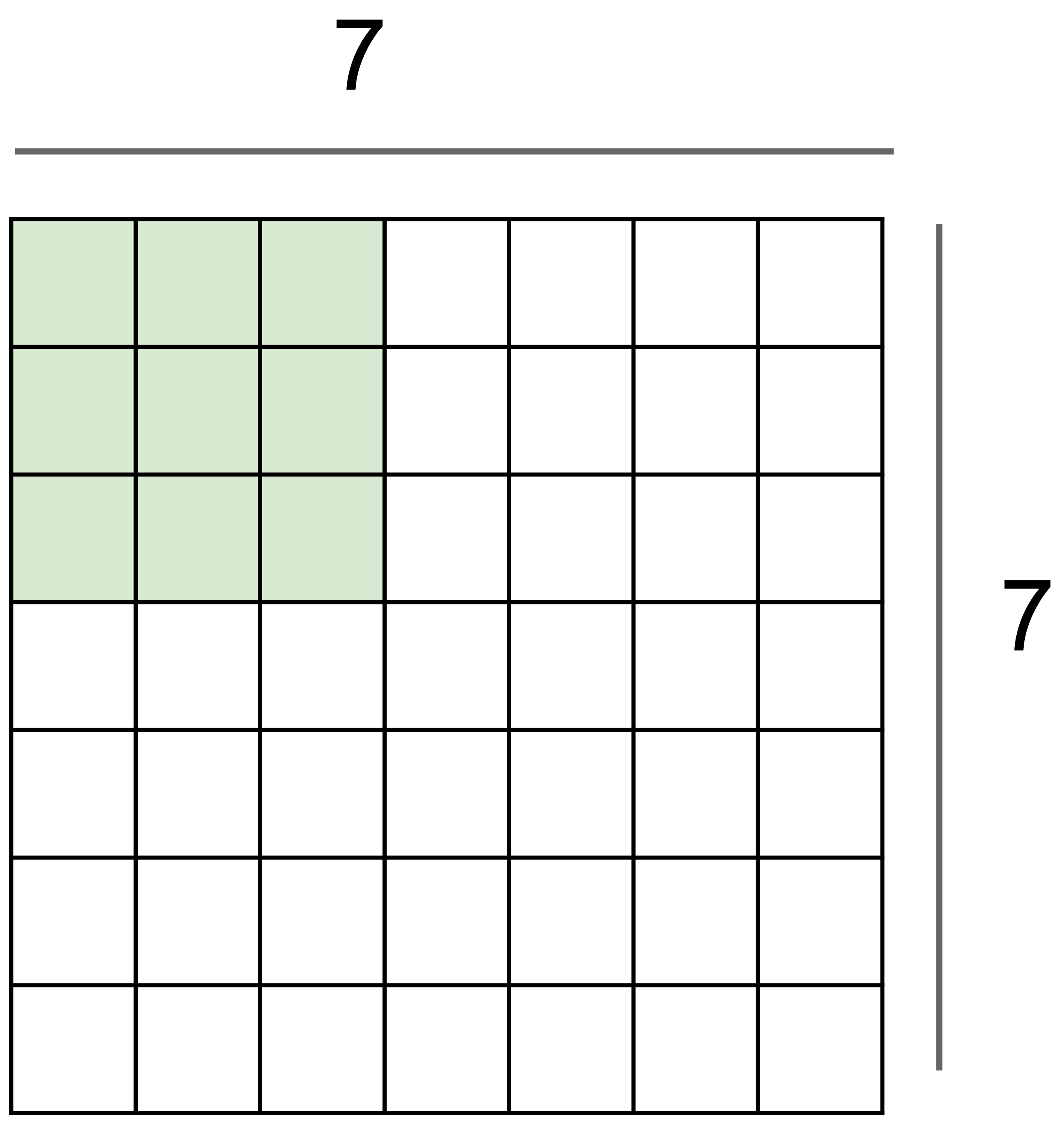
Q1. Suppose we want to perform convolution on a single channel image of size 7x7 (no padding) with a kernel of size 3x3, and stride = 2. What is the dimension of the output?

A. 3x3

B. 7x7

C. 5x5

D. 2x2




$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

# Multiple Input and Output Channels

# Multiple Input Channels

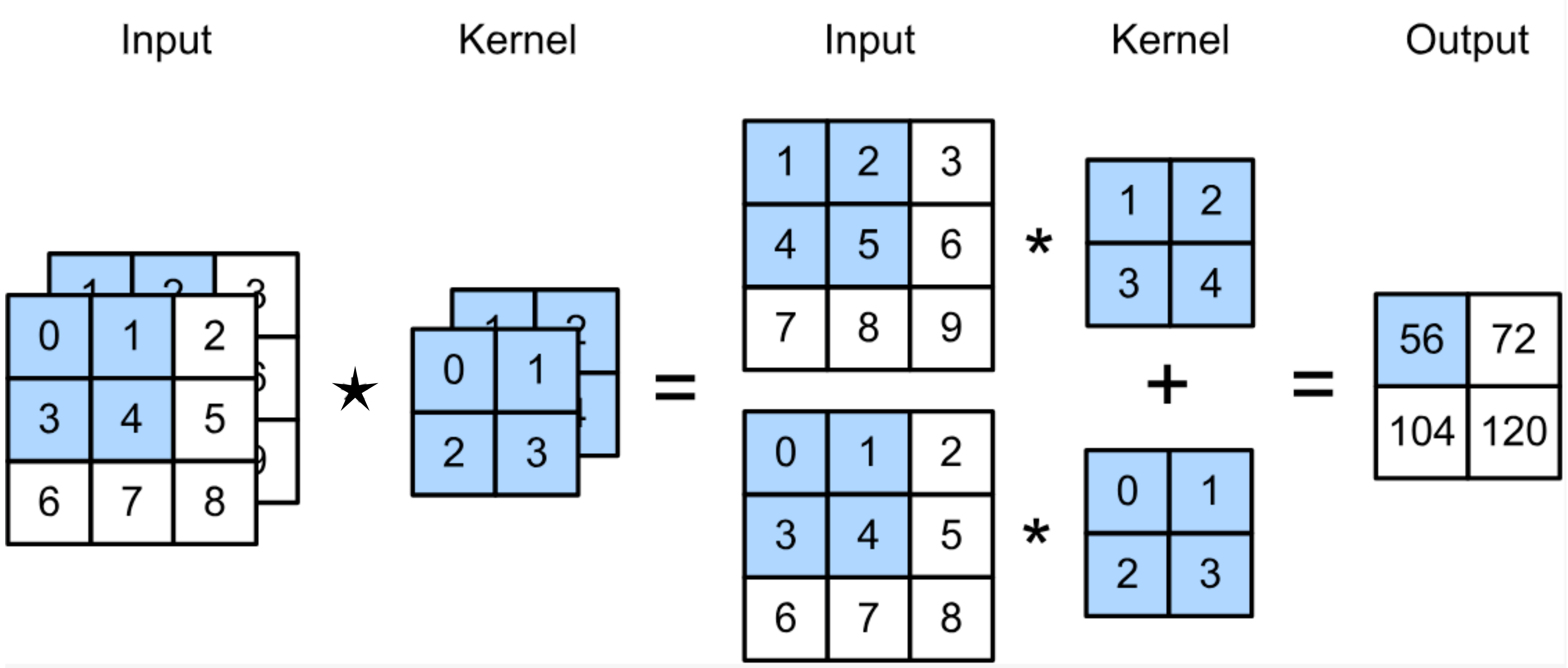
- Color image may have three RGB channels
- Converting to grayscale loses information

# Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

# Multiple Input Channels

- Have a kernel for each channel, and then sum results over channels




$$
\begin{aligned}
&(1\times 1 + 2\times 2 + 4\times 3 + 5\times 4)\\
&+(0\times 0 + 1\times 1 + 3\times 2 + 4\times 3)\\
&= 56
\end{aligned}
$$

# Multiple Input Channels

- $\mathbf{X} : c_i \times n_h \times n_w$ input
- $\mathbf{W} : c_i \times k_h \times k_w$ kernel
- $\mathbf{Y} : m_h \times m_w$ output

$$\mathbf{Y} = \sum_{j=0}^{c_i} \mathbf{X}_{j,:,:} \star \mathbf{W}_{j,:,:}$$

# Multiple Output Channels

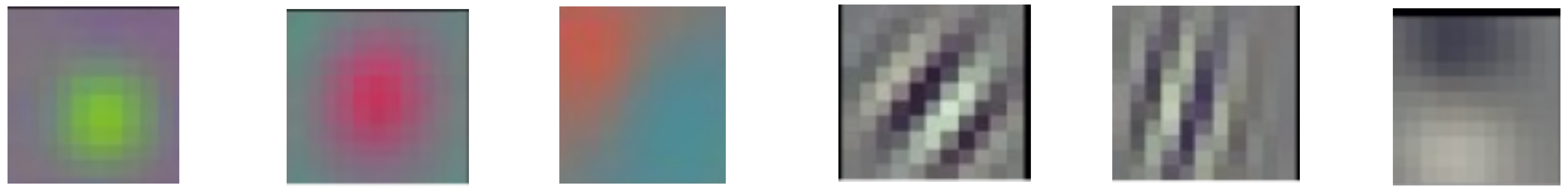
- No matter how many inputs channels, so far we always get a single output channel
- We can have **multiple 3-D kernels**, each one generates an output channel
- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$

$$\mathbf{Y}_{\ell,:,:} = \mathbf{X} \star \mathbf{W}_{\ell,:,:,:}$$

$$\text{for } \ell = 1,\ldots,c_o$$

# Multiple Input/Output Channels

- Each 3-D kernel may recognize a particular pattern



(Gabor filters)

Q3-1. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 3-D kernels of size 3x3. Stride = 1. Which is a reasonable estimate of the total number of scalar multiplications involved in this operation (without considering any optimization in matrix multiplication)?

A. 64x3x3x222x222
B. 64x3x3x222
C. 3x3x222x222
D. 64x3x3x3x222x222

Q3-1. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 3-D kernels of size 3x3. Stride = 1. Which is a reasonable estimate of the total number of scalar multiplications involved in this operation (without considering any optimization in matrix multiplication)?

A. 64x3x3x222x222

B. 64x3x3x222

C. 3x3x222x222

D. 64x3x3x3x222x222

Q 3-2. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 3-D kernels of size 3x3. Stride = 1. Which is a reasonable estimate of the total number of learnable parameters?

A. 64x222x222

B. 64x3x3x222

C. 3x3x3x64

D. (3x3x3+1)x64

Q 3-2. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 3-D kernels of size 3x3. Stride = 1. Which is a reasonable estimate of the total number of learnable parameters?

A. 64x222x222

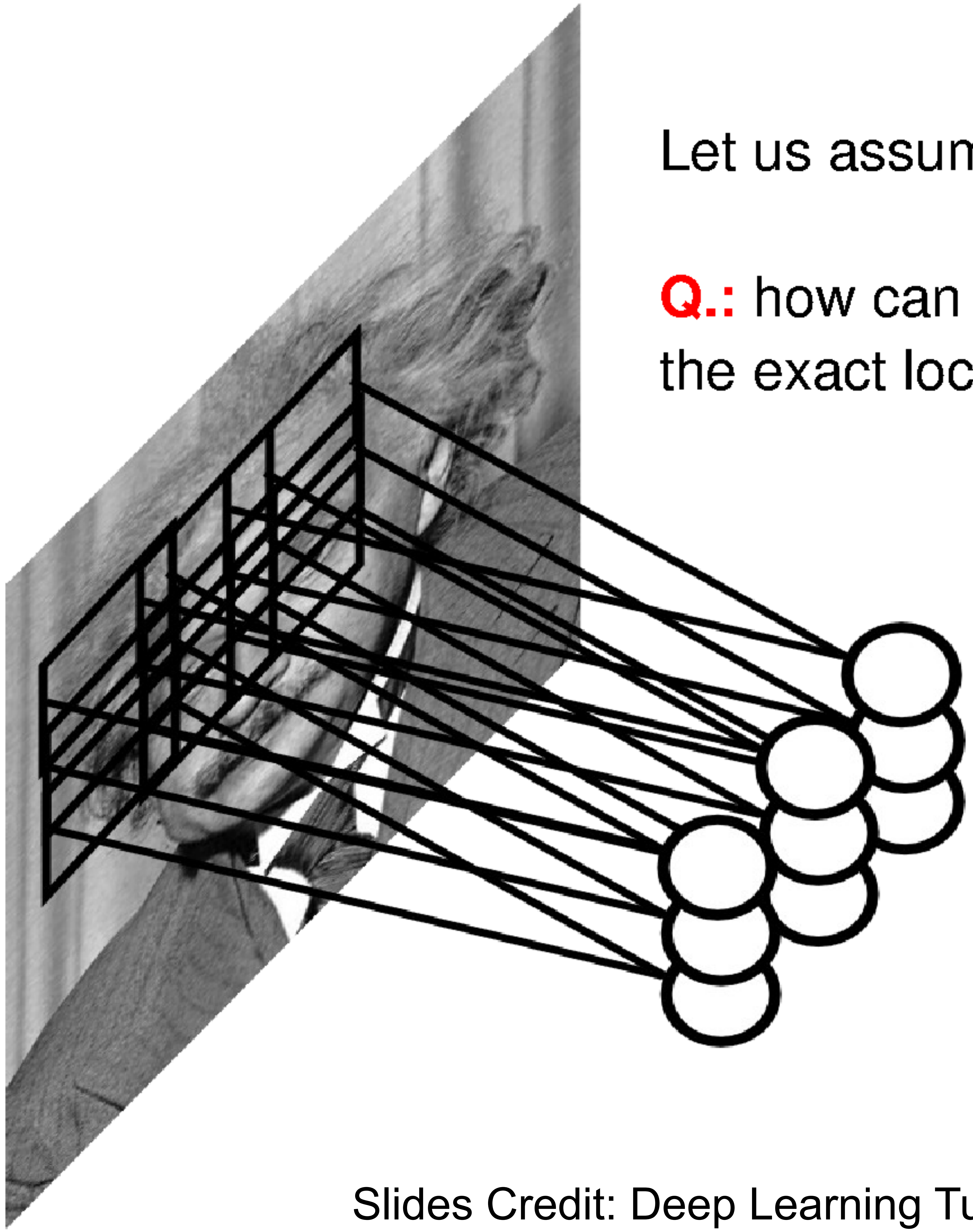
B. 64x3x3x222

C. 3x3x3x64

D. (3x3x3+1)x64

# Pooling Layer

# Pooling

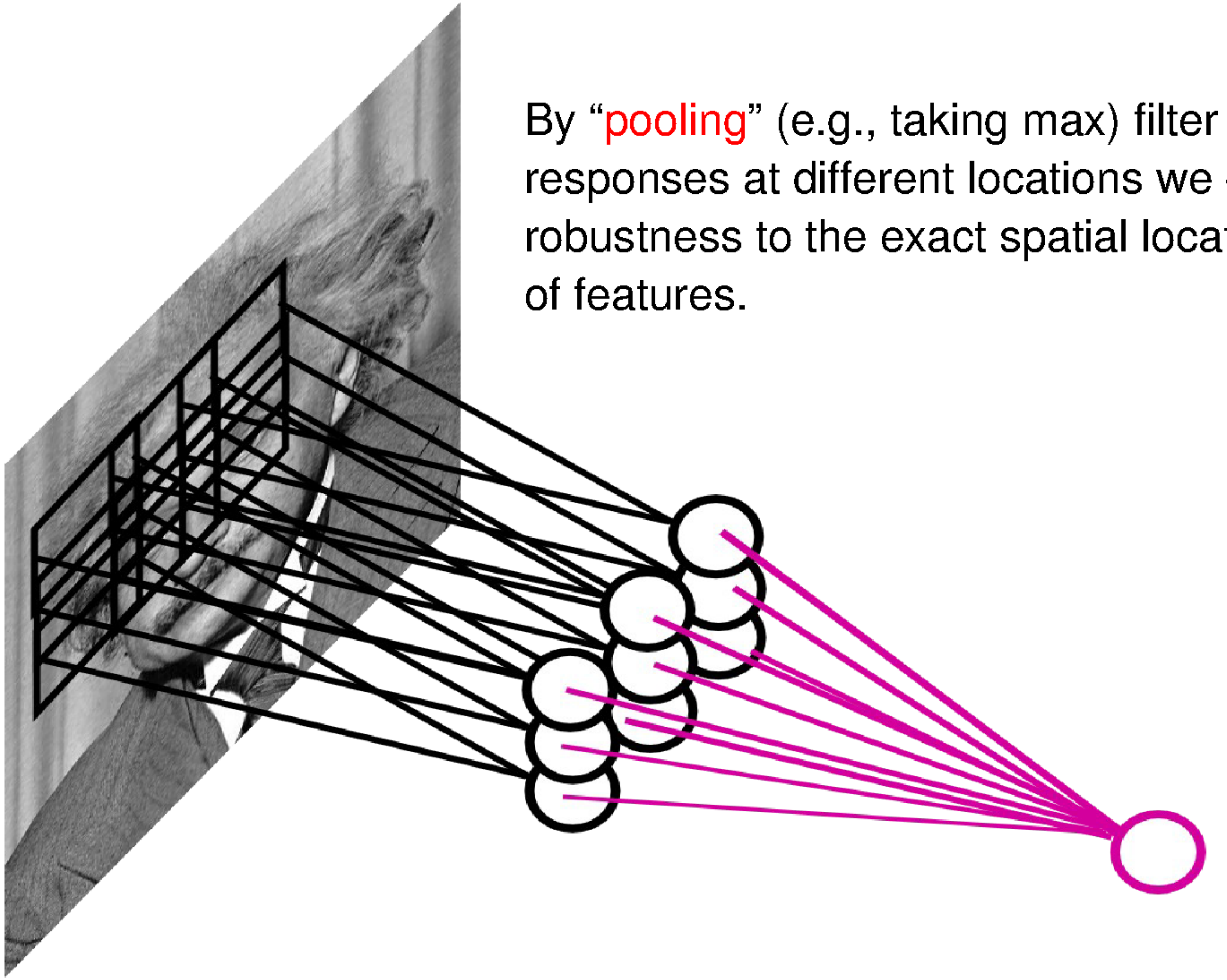


Let us assume filter is an "eye" detector.

**Q.:** how can we make the detection robust to the exact location of the eye?

Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

# Pooling

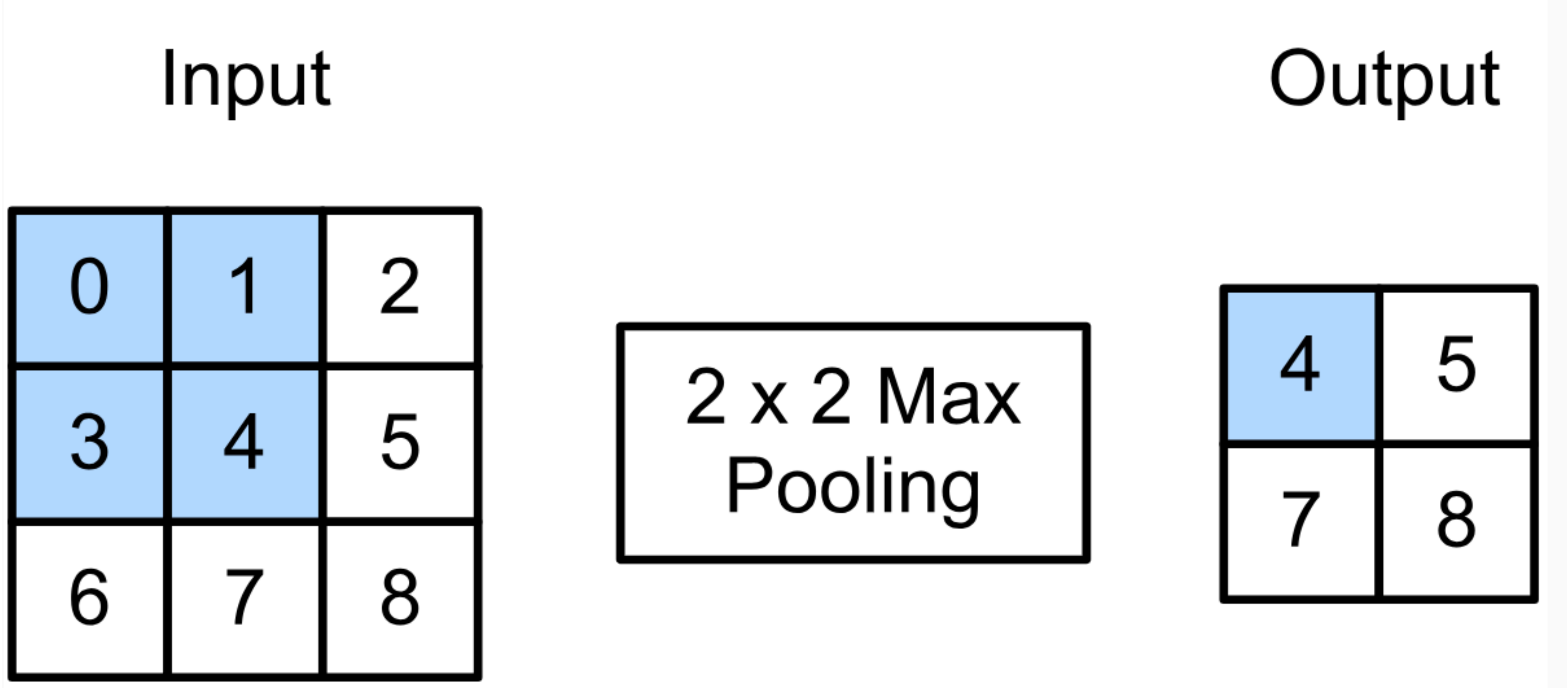


By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

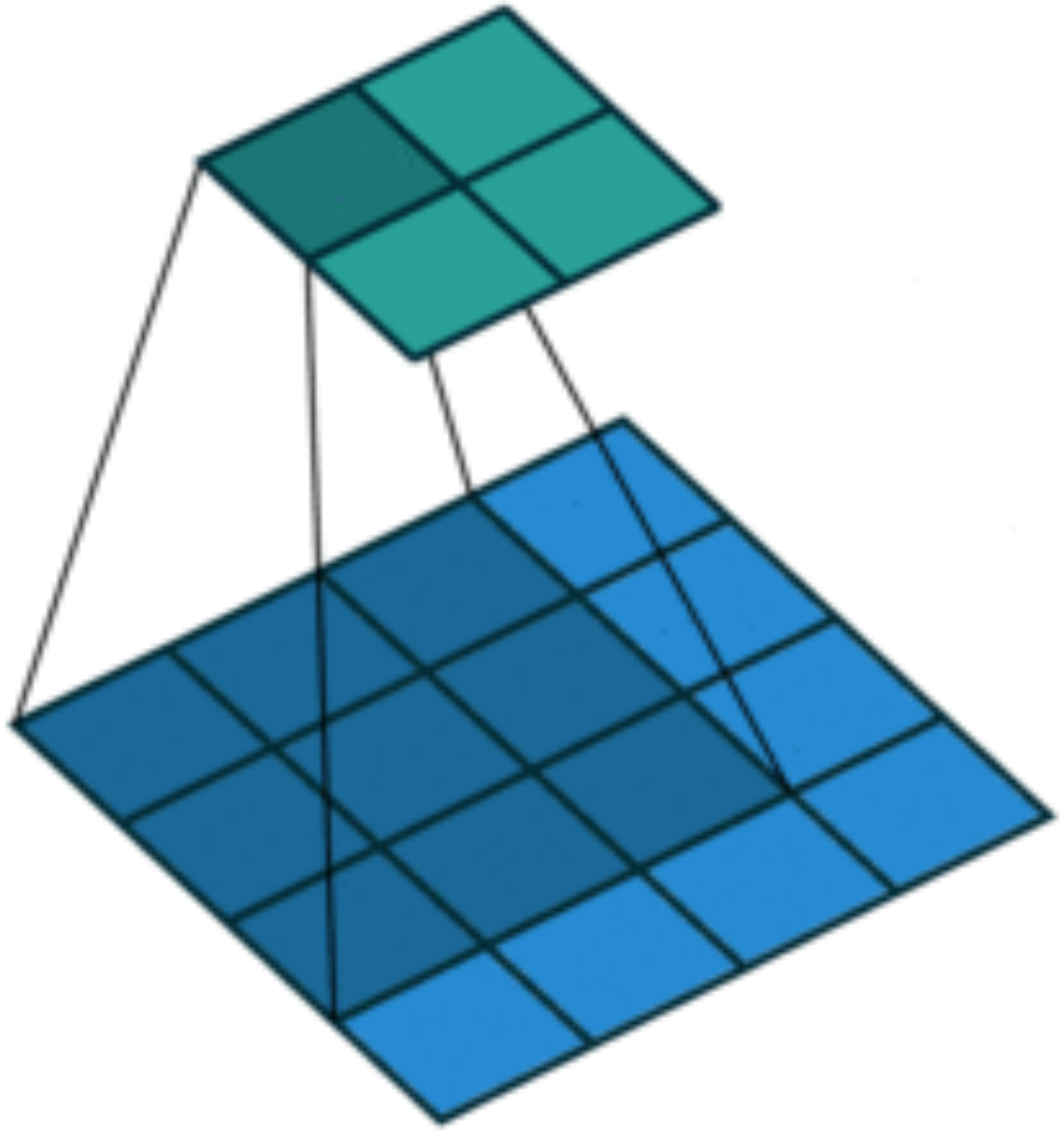
Slides Credit: Deep Learning Tutorial by Marc’Aurelio Ranzato

# 2-D Max Pooling

- Returns the maximal value in the sliding window



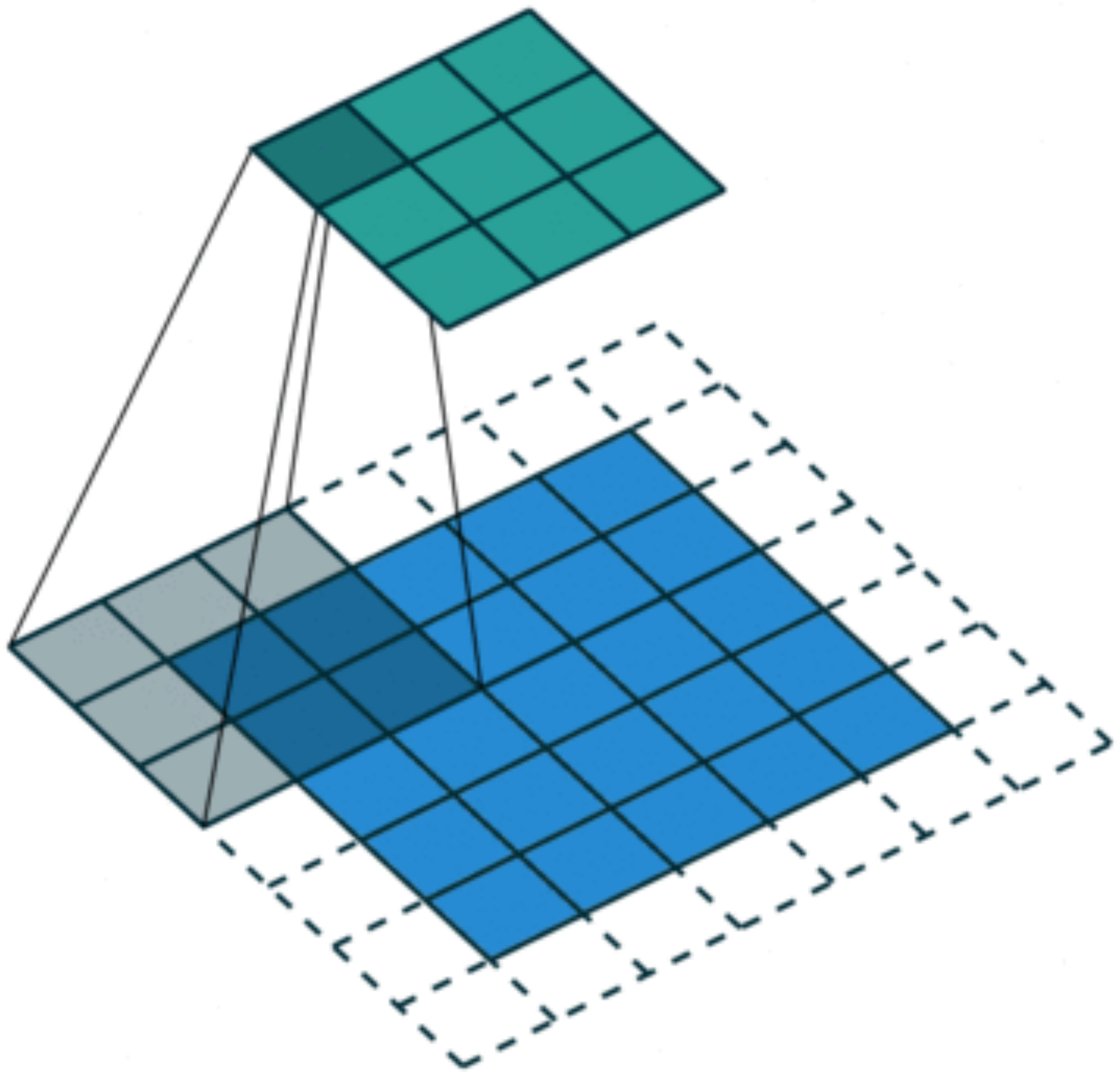

$$\max(0,1,3,4) = 4$$



# Padding, Stride, and Multiple Channels for Pooling

- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel

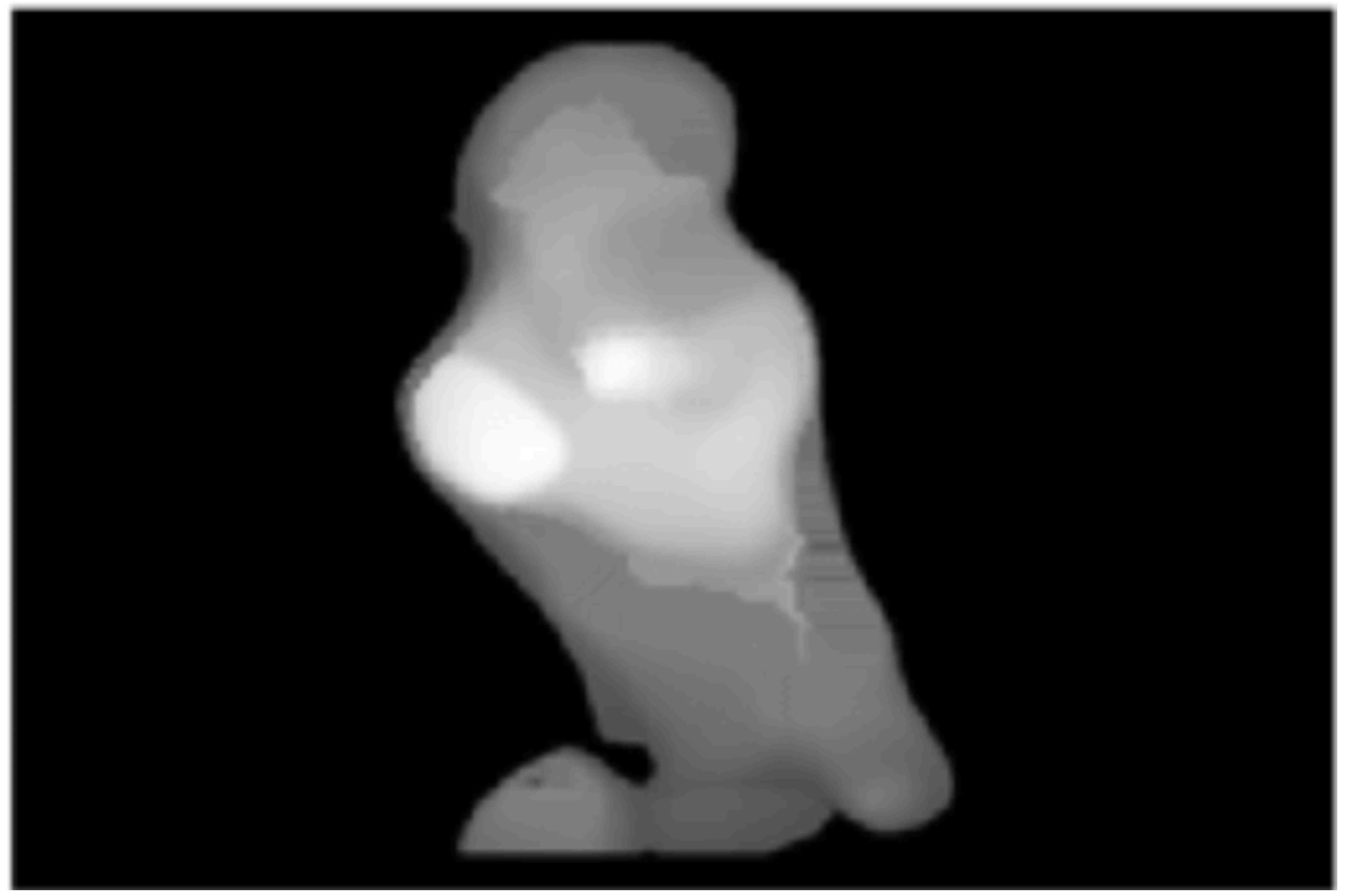
**#output channels = #input channels**



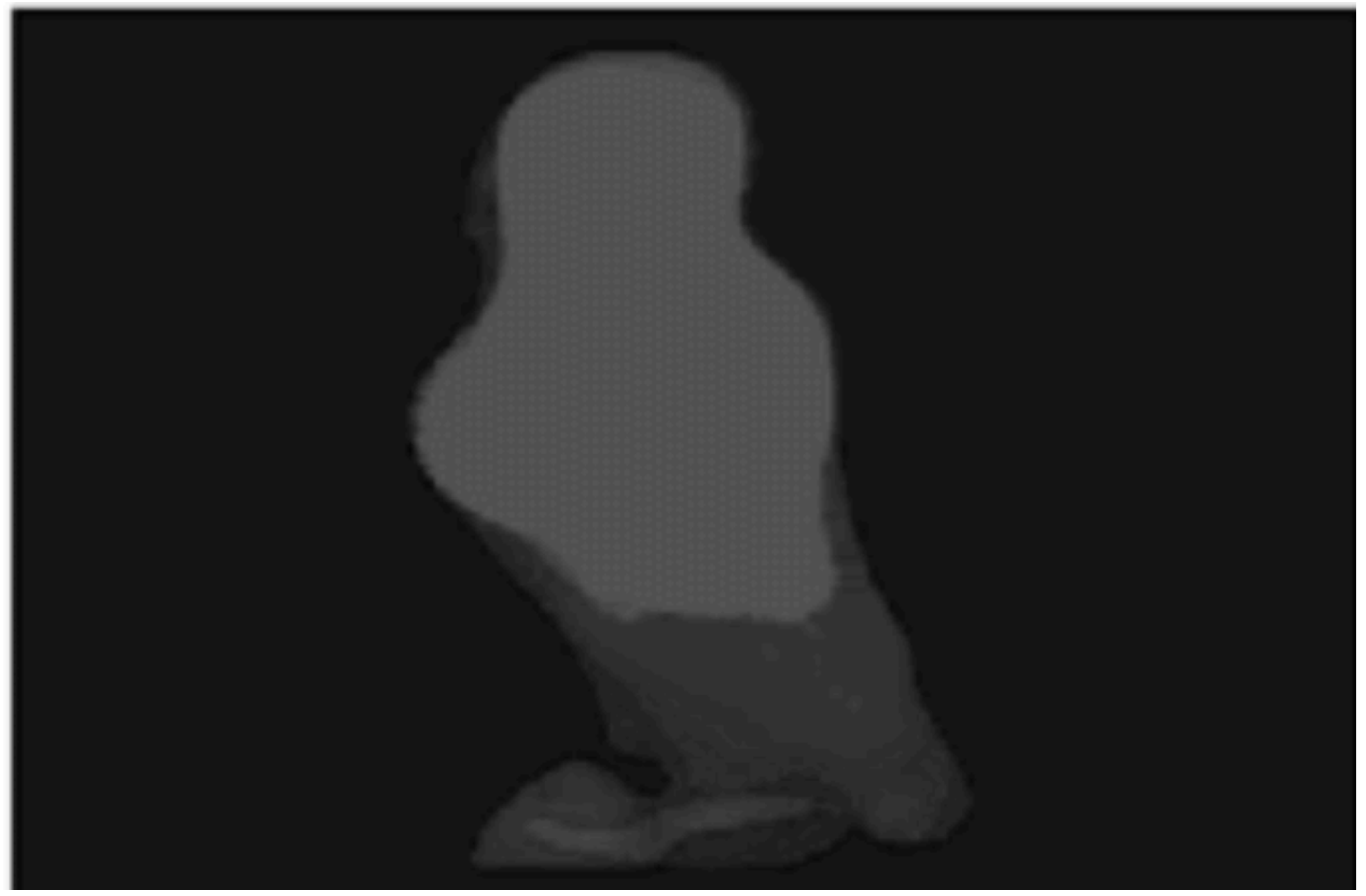
# Average Pooling

- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window

Max pooling



Average pooling



Q2-1. Suppose we want to perform 2x2 average pooling on the following single channel feature map of size 4x4 (no padding), and stride = 2. What is the output?

| | | | |
|---|---|---|---|
| 12 | 20 | 30 | 0 |
| 20 | 12 | 2 | 0 |
| 0 | 70 | 5 | 2 |
| 8 | 2 | 90 | 3 |

A.

| | |
|---|---|
| 20 | 30 |
| 70 | 90 |

B.

| | |
|---|---|
| 16 | 8 |
| 20 | 25 |

C.

| | |
|---|---|
| 20 | 30 |
| 20 | 25 |

D.

| | |
|---|---|
| 12 | 2 |
| 70 | 5 |

Q2-1. Suppose we want to perform 2x2 average pooling on the following single channel feature map of size 4x4 (no padding), and stride = 2. What is the output?

| | | | |
|---|---|---|---|
| 12 | 20 | 30 | 0 |
| 20 | 12 | 2 | 0 |
| 0 | 70 | 5 | 2 |
| 8 | 2 | 90 | 3 |

A.

| | |
|---|---|
| 20 | 30 |
| 70 | 90 |

B.

| | |
|---|---|
| 16 | 8 |
| 20 | 25 |

C.

| | |
|---|---|
| 20 | 30 |
| 20 | 25 |

D.

| | |
|---|---|
| 12 | 2 |
| 70 | 5 |

Q2-2. What is the output if we use 2 x 2 max pooling (other settings are the same)?

| | | | |
|---|---|---|---|
| 12 | 20 | 30 | 0 |
| 20 | 12 | 2 | 0 |
| 0 | 70 | 5 | 2 |
| 8 | 2 | 90 | 3 |

A.

| | |
|---|---|
| 20 | 30 |
| 70 | 90 |

B.

| | |
|---|---|
| 16 | 8 |
| 20 | 25 |

C.

| | |
|---|---|
| 20 | 30 |
| 20 | 25 |

D.

| | |
|---|---|
| 12 | 2 |
| 70 | 5 |

Q2-2. What is the output if we use 2 x 2 max pooling (other settings are the same)?

| | | | |
|---|---|---|---|
| 12 | 20 | 30 | 0 |
| 20 | 12 | 2 | 0 |
| 0 | 70 | 5 | 2 |
| 8 | 2 | 90 | 3 |

A.

| | |
|---|---|
| 20 | 30 |
| 70 | 90 |

B.

| | |
|---|---|
| 16 | 8 |
| 20 | 25 |

C.

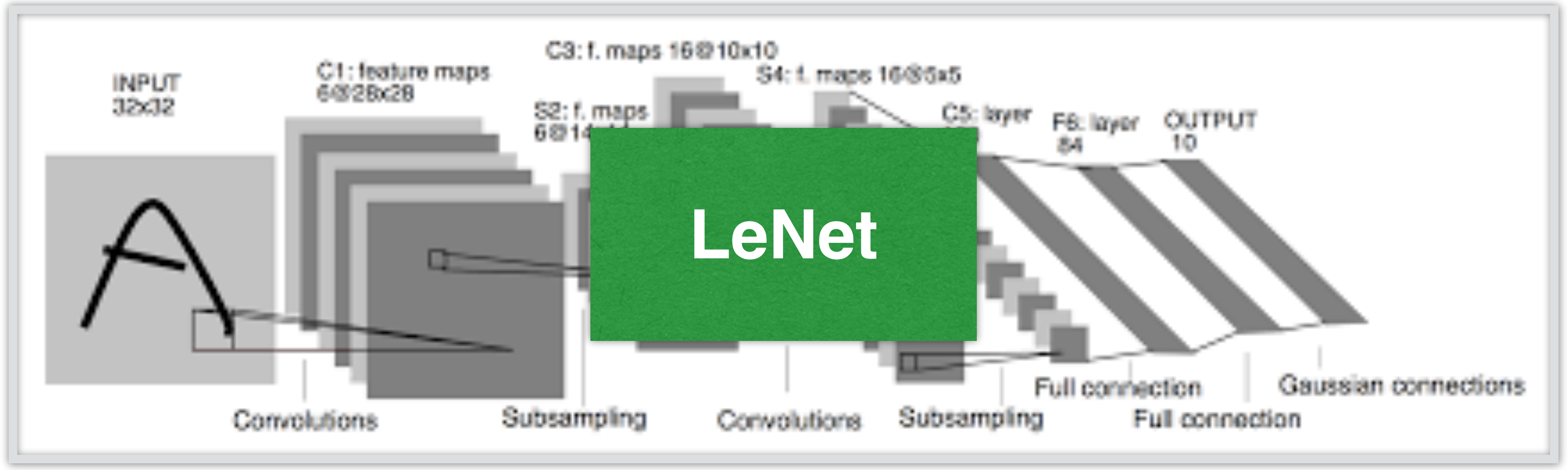
| | |
|---|---|
| 20 | 30 |
| 20 | 25 |

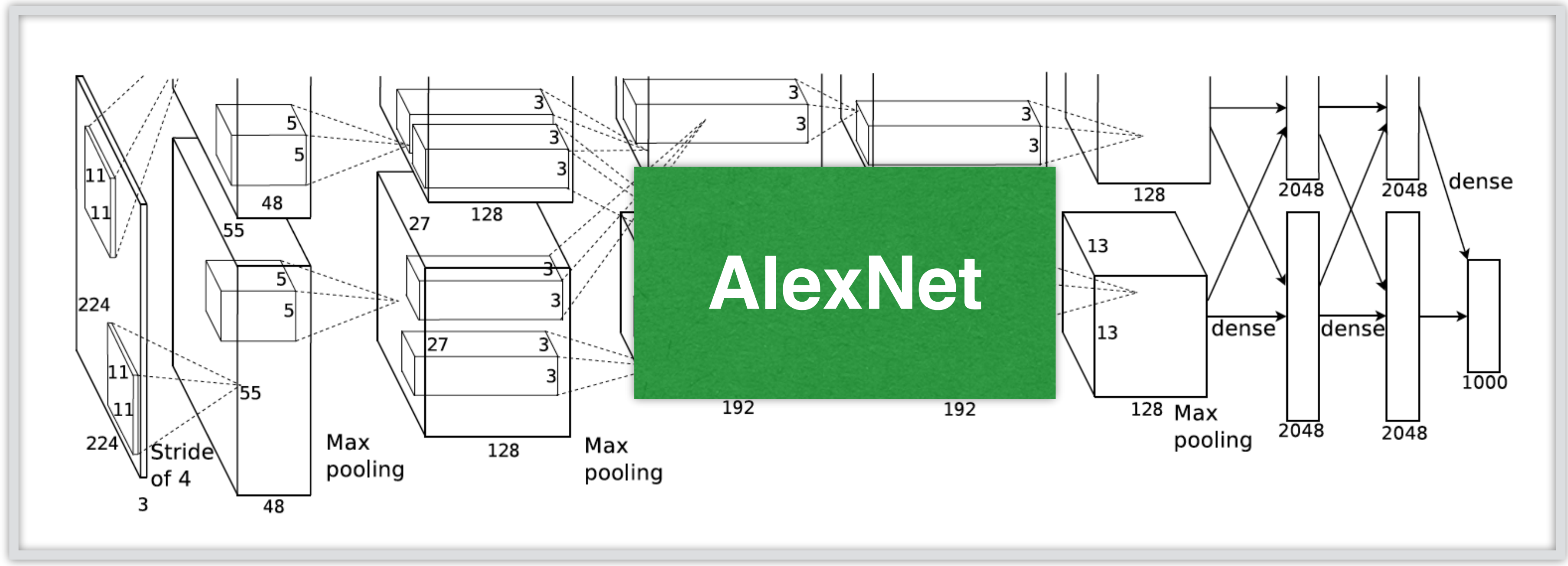
D.

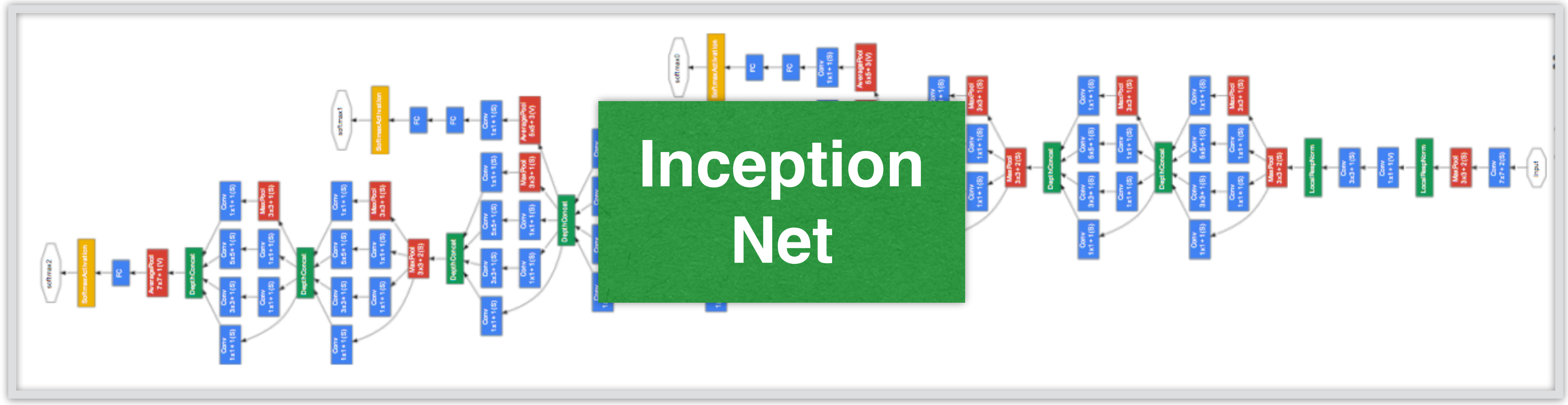
| | |
|---|---|
| 12 | 2 |
| 70 | 5 |

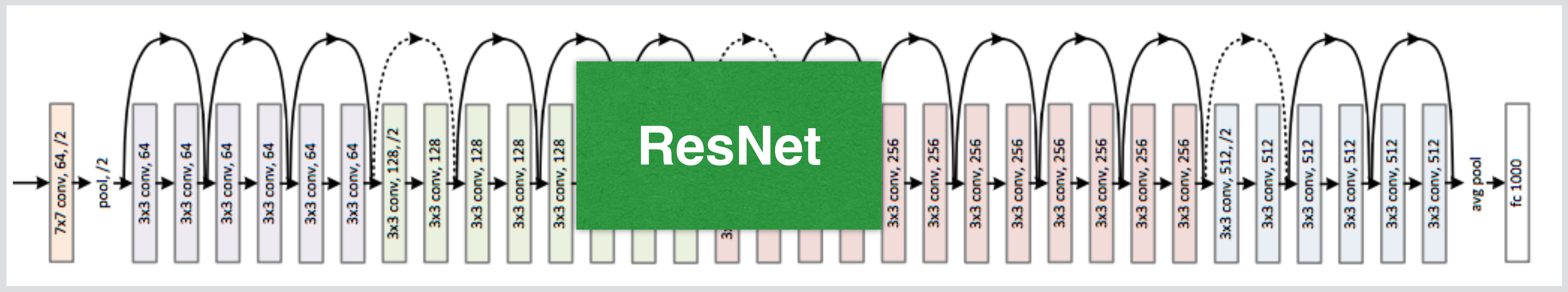
# Convolutional Neural Networks

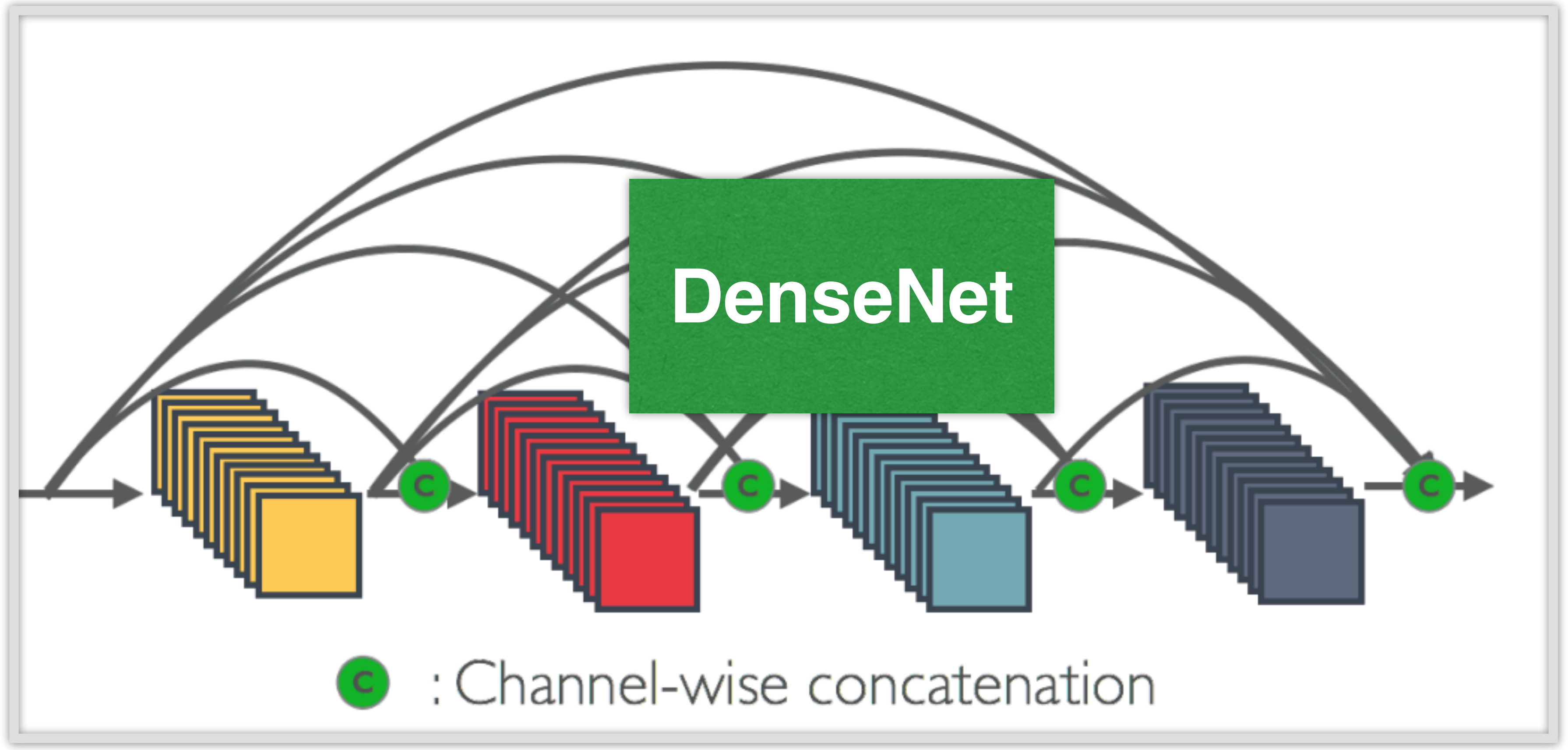
# Evolution of neural net architectures



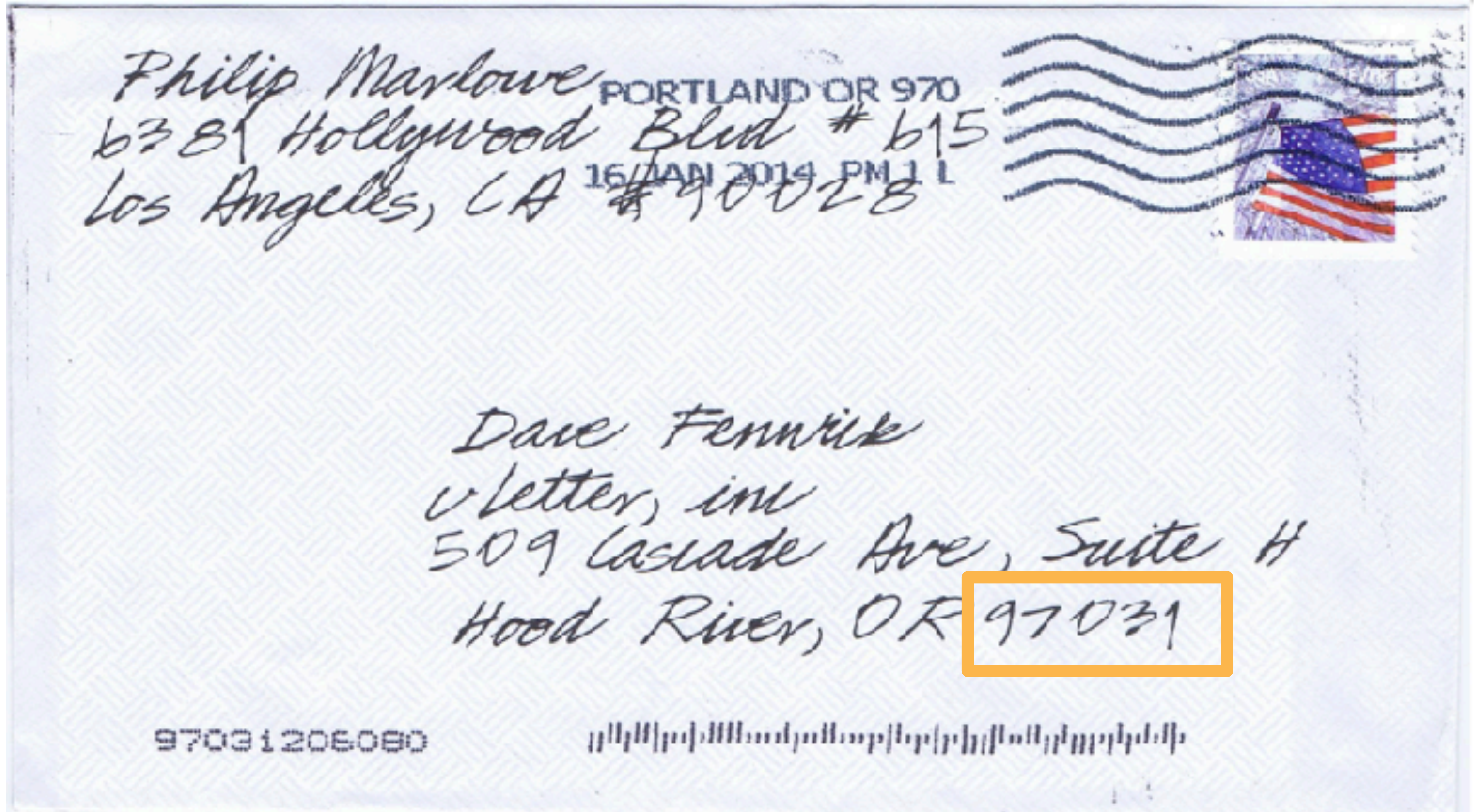



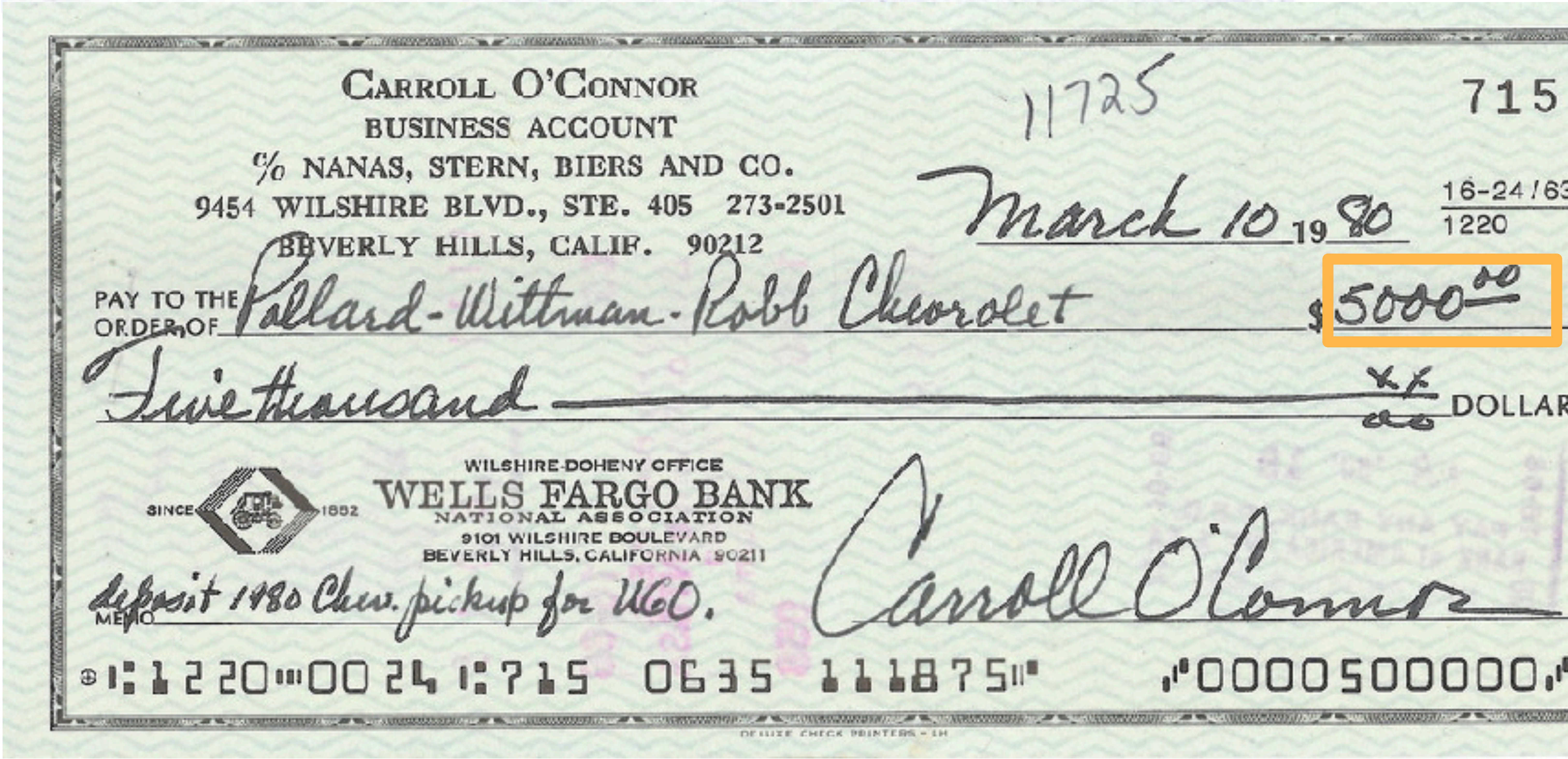



# LeNet Architecture


gluon-cv.mxnet.io

# Handwritten Digit Recognition



# MNIST

- Centered and scaled
- 50,000 training data
- 10,000 test data
- 28 x 28 images
- 10 classes





Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition

# LeNet Architecture

# LeNet in Pytorch

```
def __init__(self):
    super(LeNet5, self).__init__()
    # Convolution (In LeNet-5, 32x32 images are given as input. Hence padding of 2 is done below)
    self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1, padding=2, bias=True)
    # Max-pooling
    self.max_pool_1 = torch.nn.MaxPool2d(kernel_size=2)
    # Convolution
    self.conv2 = torch.nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1, padding=0, bias=True)
    # Max-pooling
    self.max_pool_2 = torch.nn.MaxPool2d(kernel_size=2)
    # Fully connected layer
    self.fc1 = torch.nn.Linear(16*5*5, 120)   # convert matrix with 16*5*5 (= 400) features to a matrix of 120 features (col
    self.fc2 = torch.nn.Linear(120, 84)       # convert matrix with 120 features to a matrix of 84 features (columns)
    self.fc3 = torch.nn.Linear(84, 10)        # convert matrix with 84 features to a matrix of 10 features (columns)
```

# Summary

- Intro of convolutional computations
  - 2D convolution
  - Padding, stride etc
  - Multiple input and output channels
  - Pooling
- Basic Convolutional Neural Networks
  - LeNet (first conv nets)

**Acknowledgement**:
Some of the slides in these lectures have been adapted from materials developed by Alex Smola and Mu Li:
https://courses.d2l.ai/berkeley-stat-157/index.html