CS540 Introduction to Artificial Intelligence

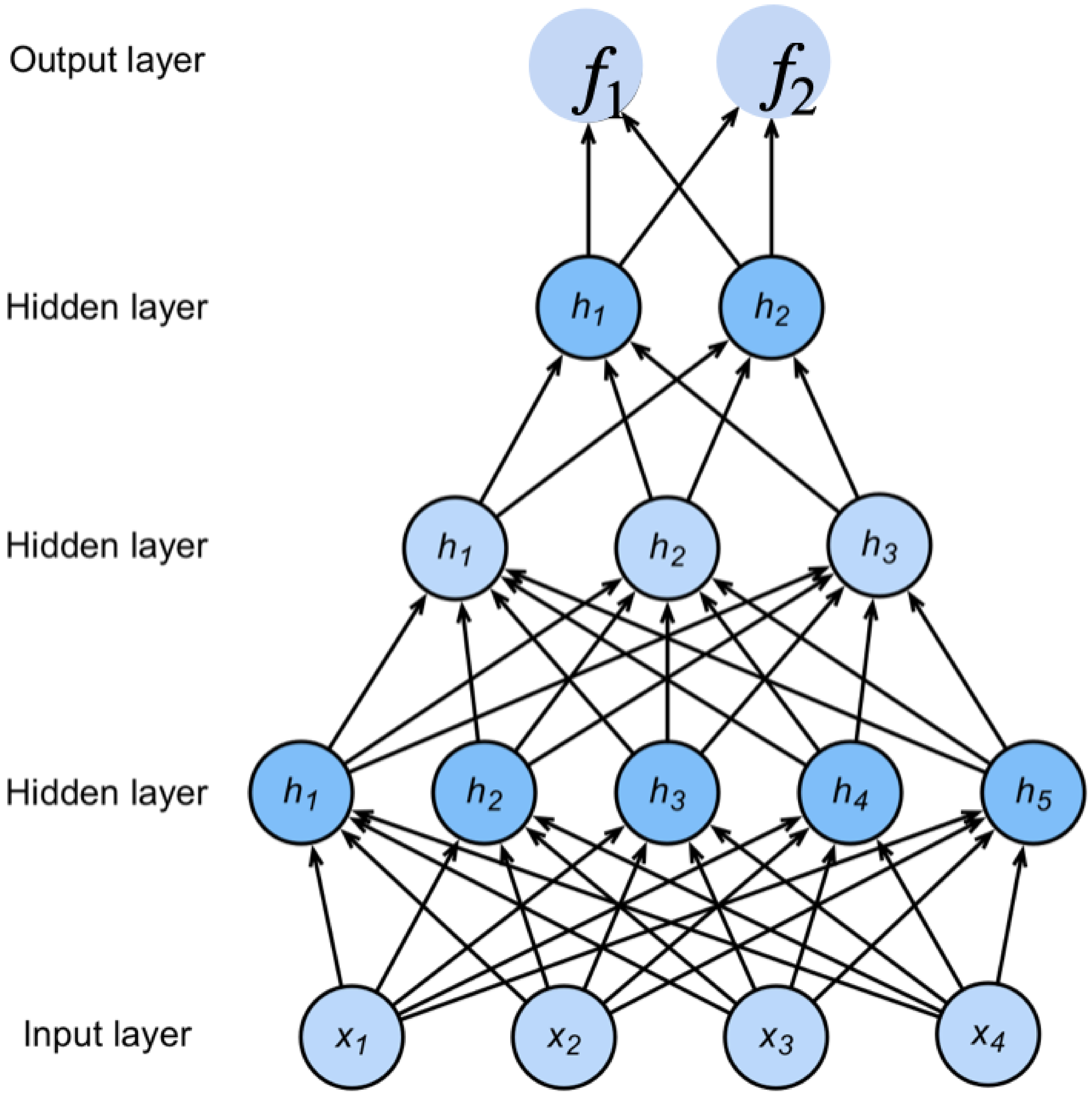
Deep Learning I: Convolutional Neural Networks

University of Wisconsin-Madison

Outline

- Intro of convolutional computations
    - 2D convolution
    - Padding, stride
    - Multiple input and output channels
    - Pooling

Review: Deep Neural Networks



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

$$\mathbf{h}_2 = \sigma(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)})$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)})$$

$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}_3 + \mathbf{b}^{(4)}$$

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

NNs are composition of nonlinear functions

How to classify

Cats vs. dogs?

Dual

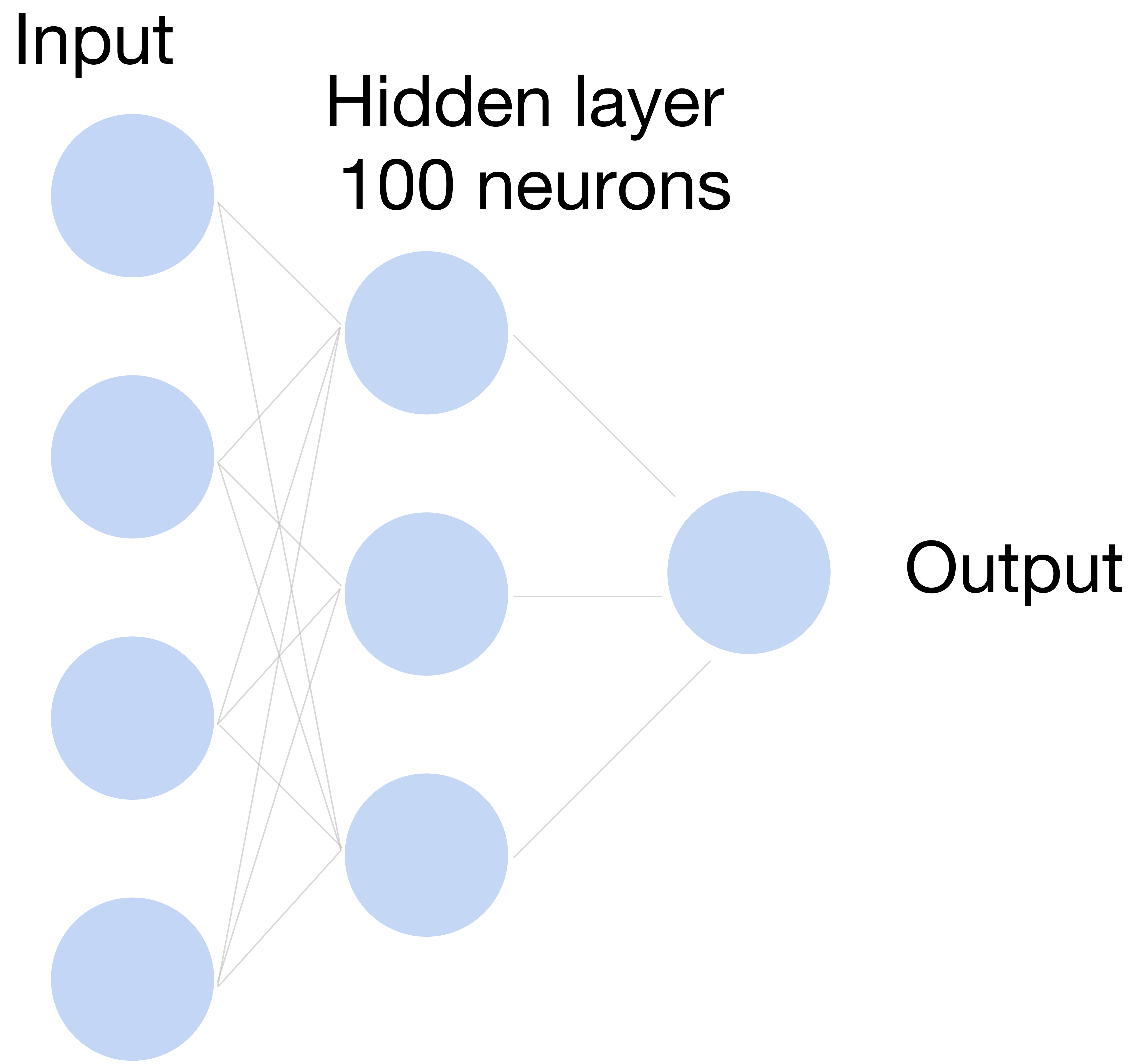
12MP

wide-angle and
telephoto cameras

36M floats in a RGB image!

Fully Connected Networks

Cats vs. dogs?



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

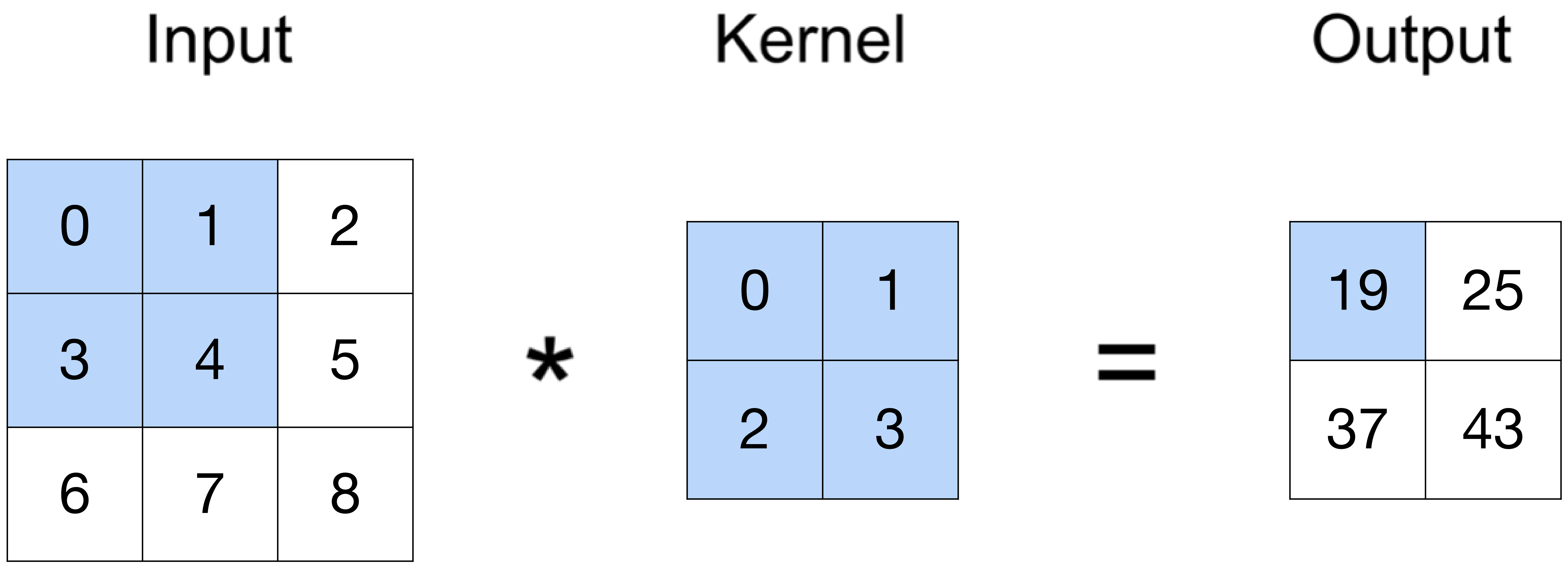
Convolutions come to rescue!

Where is Waldo?

Why Convolution?

- Translation Invariance
- Locality

2-D Convolution



0x0 + 1x1 + 3x2 + 4x3 = 19

2-D Convolution

Input

0	1	2
3	4	5
6	7	8

*

Kernel

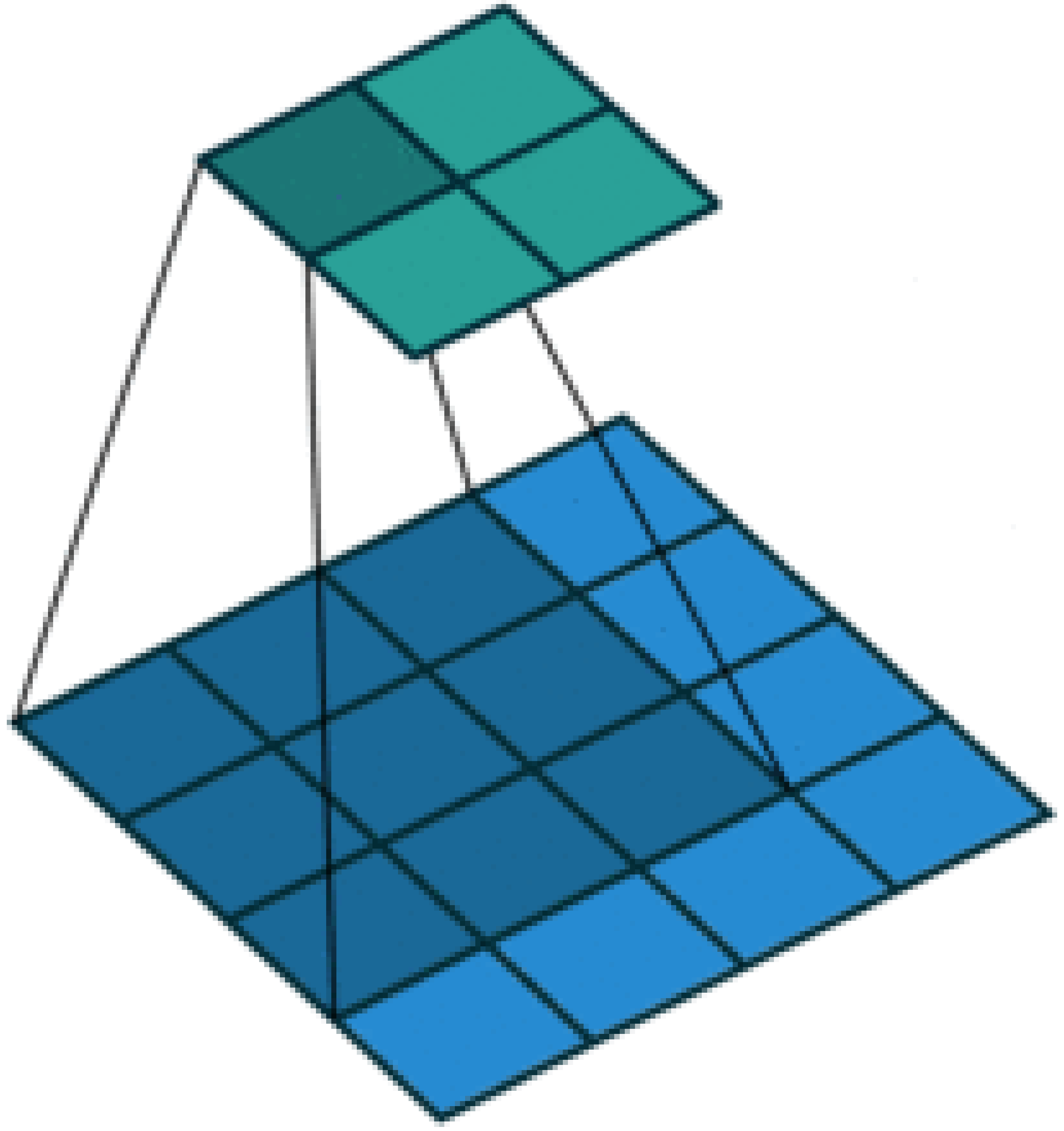
0	1
2	3

=

Output

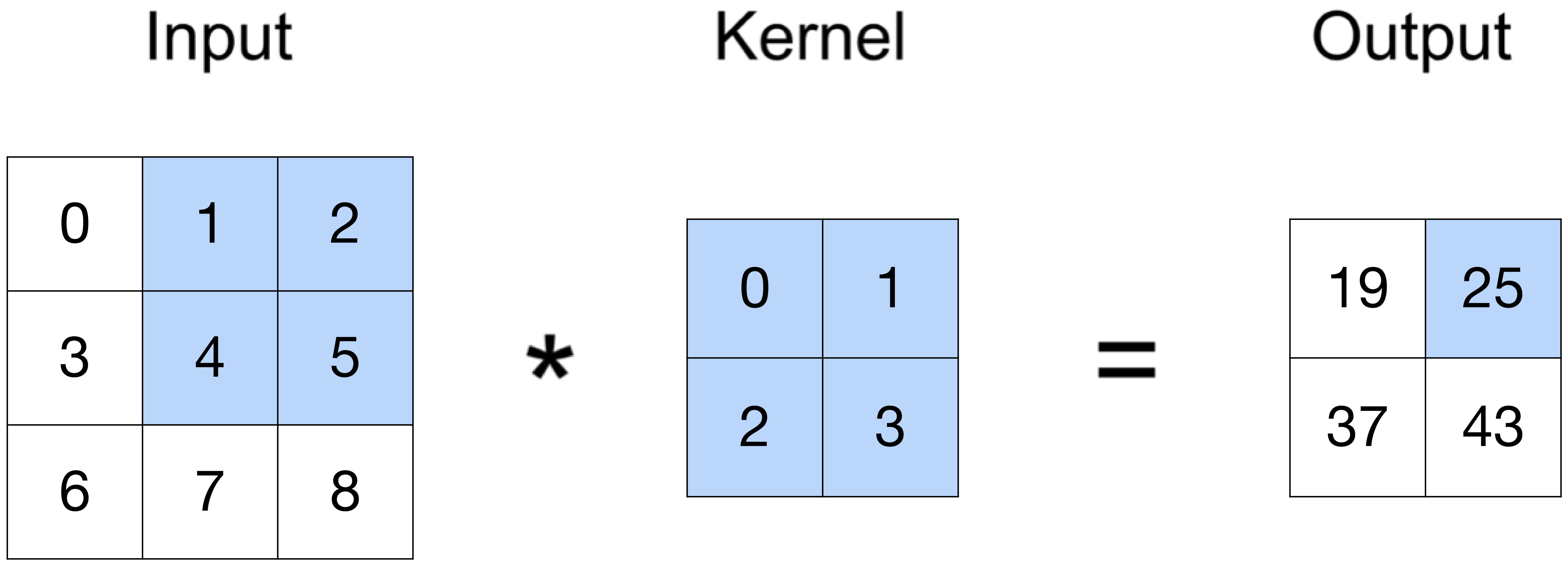
19	25
37	43

$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$



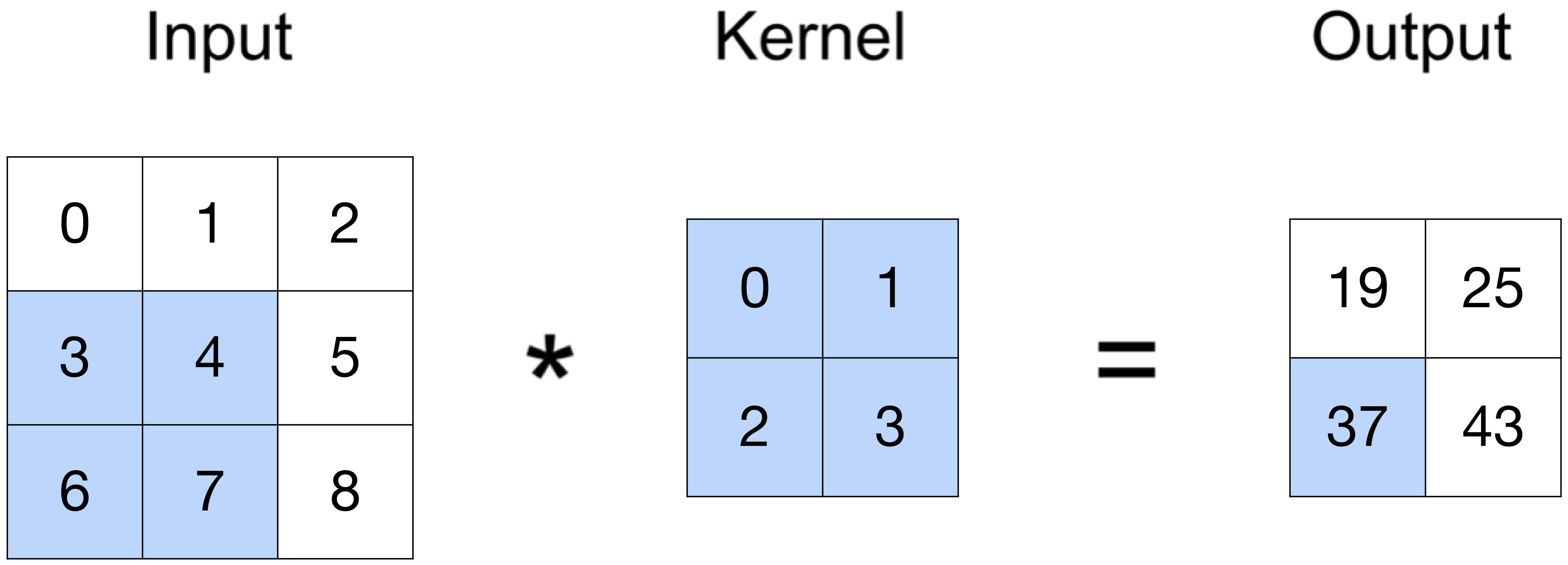
(vdumoulin@ Github)

2-D Convolution



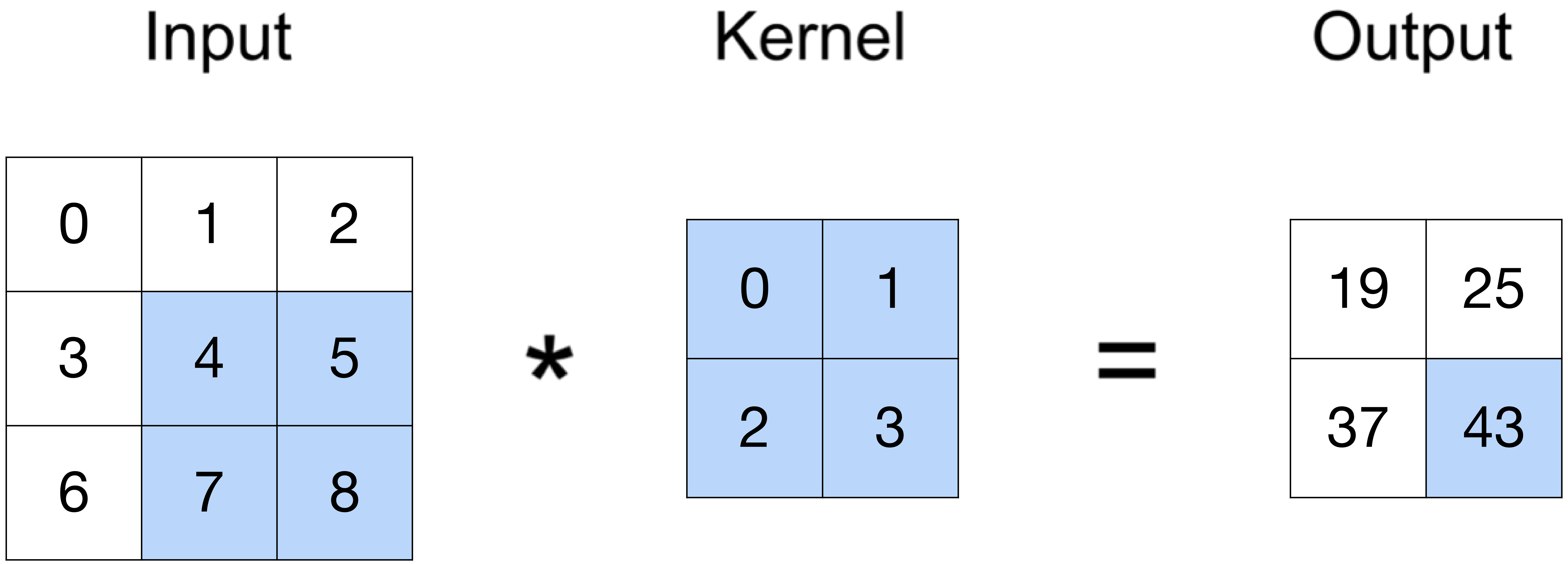
1x0 + 2x1 + 4x2 + 5x3 = 25

2-D Convolution



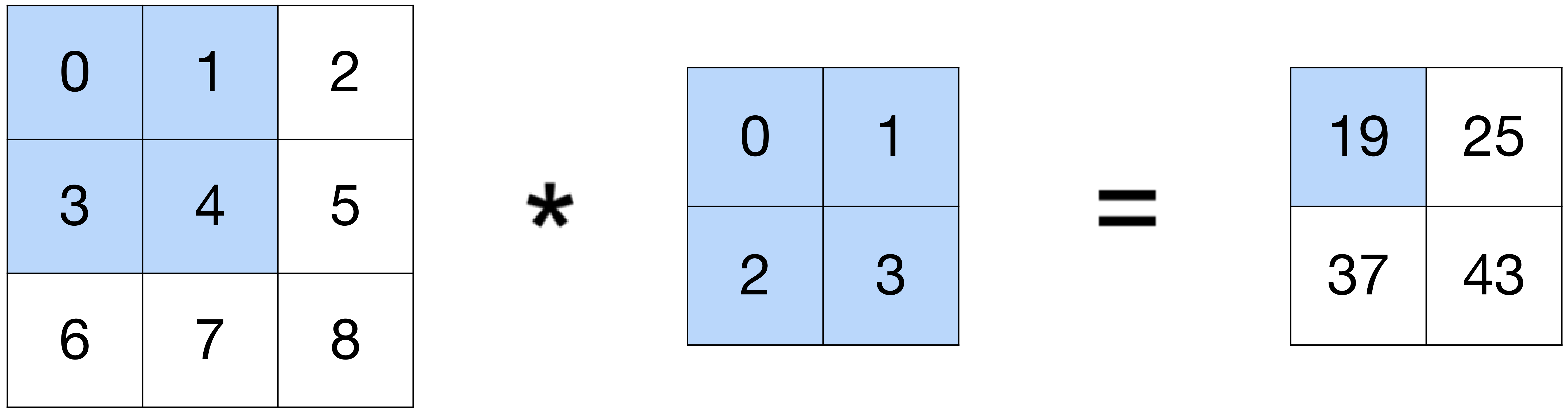
3x0 + 4x1 + 6x2 + 7x3 = 37

2-D Convolution



4x0 + 5x1 + 7x2 + 8x3 = 43

2-D Convolution Layer



- **X**: n_h x n_w input matrix
- **W**: k_h x k_w kernel matrix
- **Y**: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

Y = X * W

2-D Convolution Layer

$$\begin{bmatrix}0 & 1 & 2\\ 3 & 4 & 5\\ 6 & 7 & 8\end{bmatrix} * \begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix} + 1 = \begin{bmatrix}20 & 26\\ 38 & 44\end{bmatrix}$$

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

Y = X * 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

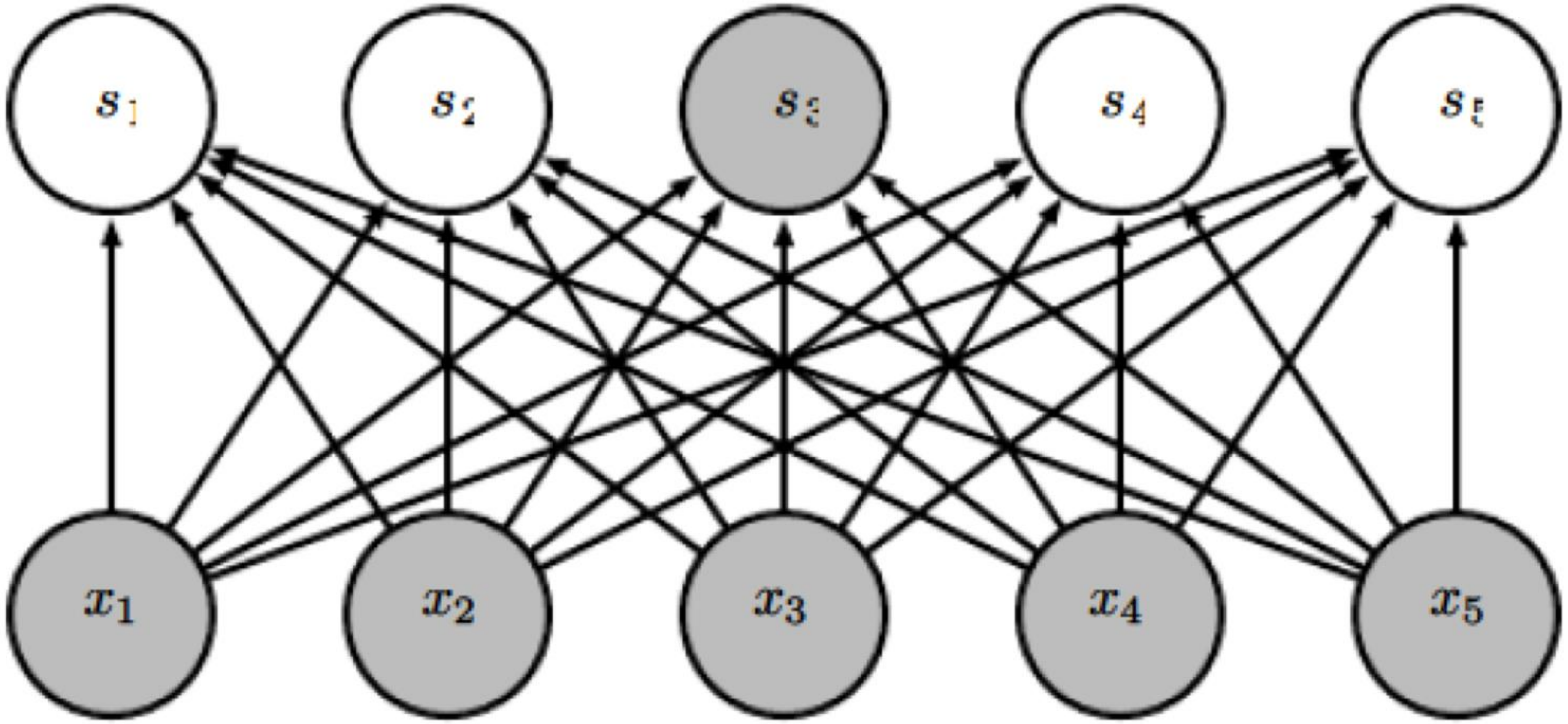
Convolutional Neural Networks

- Strong empirical application performance

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

Advantage: sparse interaction

Fully connected layer, $m \times n$ edges



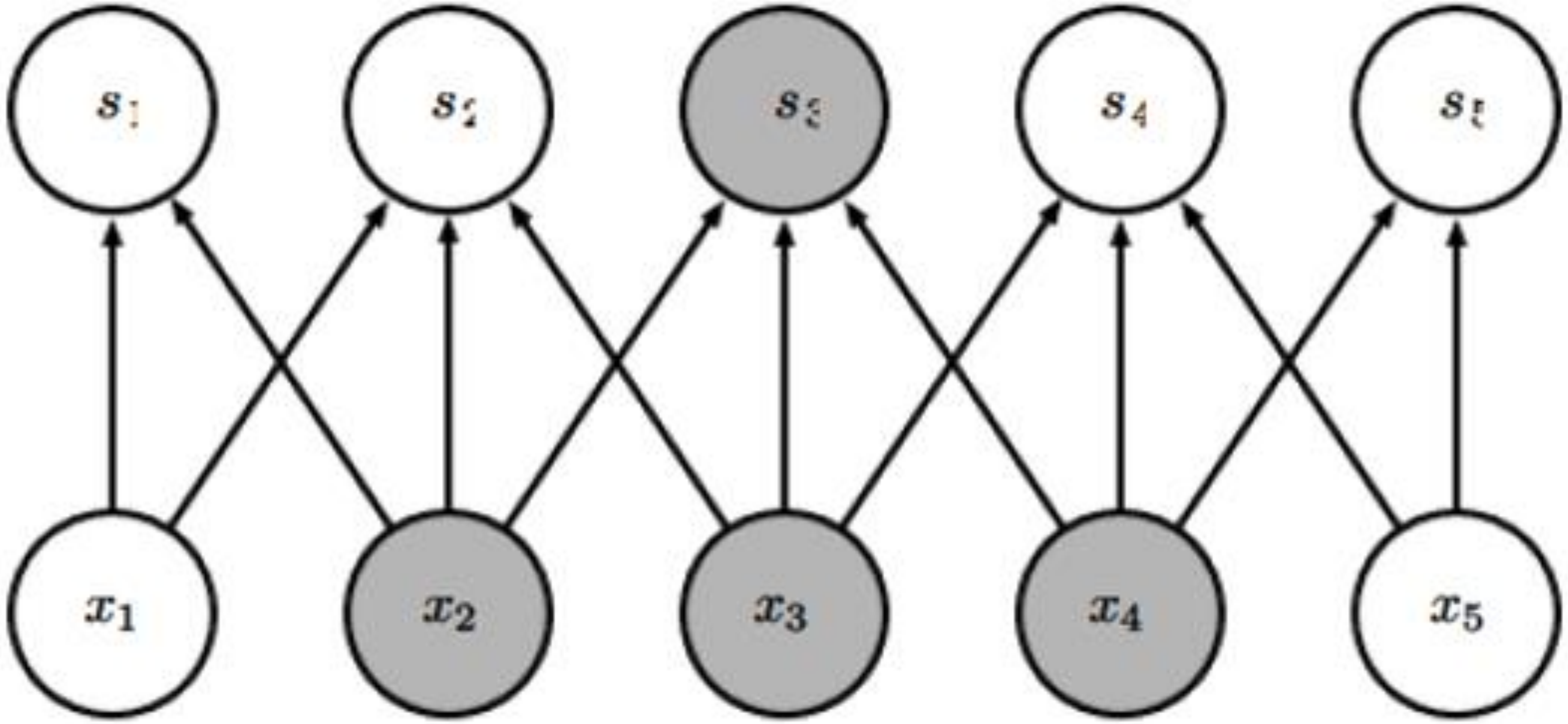
m output nodes

n input nodes

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

Advantage: 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

Q1. Suppose we want to perform convolution as follows. What's the output?

0	1	2
3	4	5
6	7	8

$*$

0	1
1	-1

$+1 = ?$

A.

1	2
4	5

A.

1	2
3	4

A.

1	3
3	5

0	1
3	4

Q1. Suppose we want to perform convolution as follows. What's the output?

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \boxed{+1} = \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix}$$

A. $\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix}$

B. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

$\begin{bmatrix} 1 & 3 \\ 3 & 5 \end{bmatrix}$

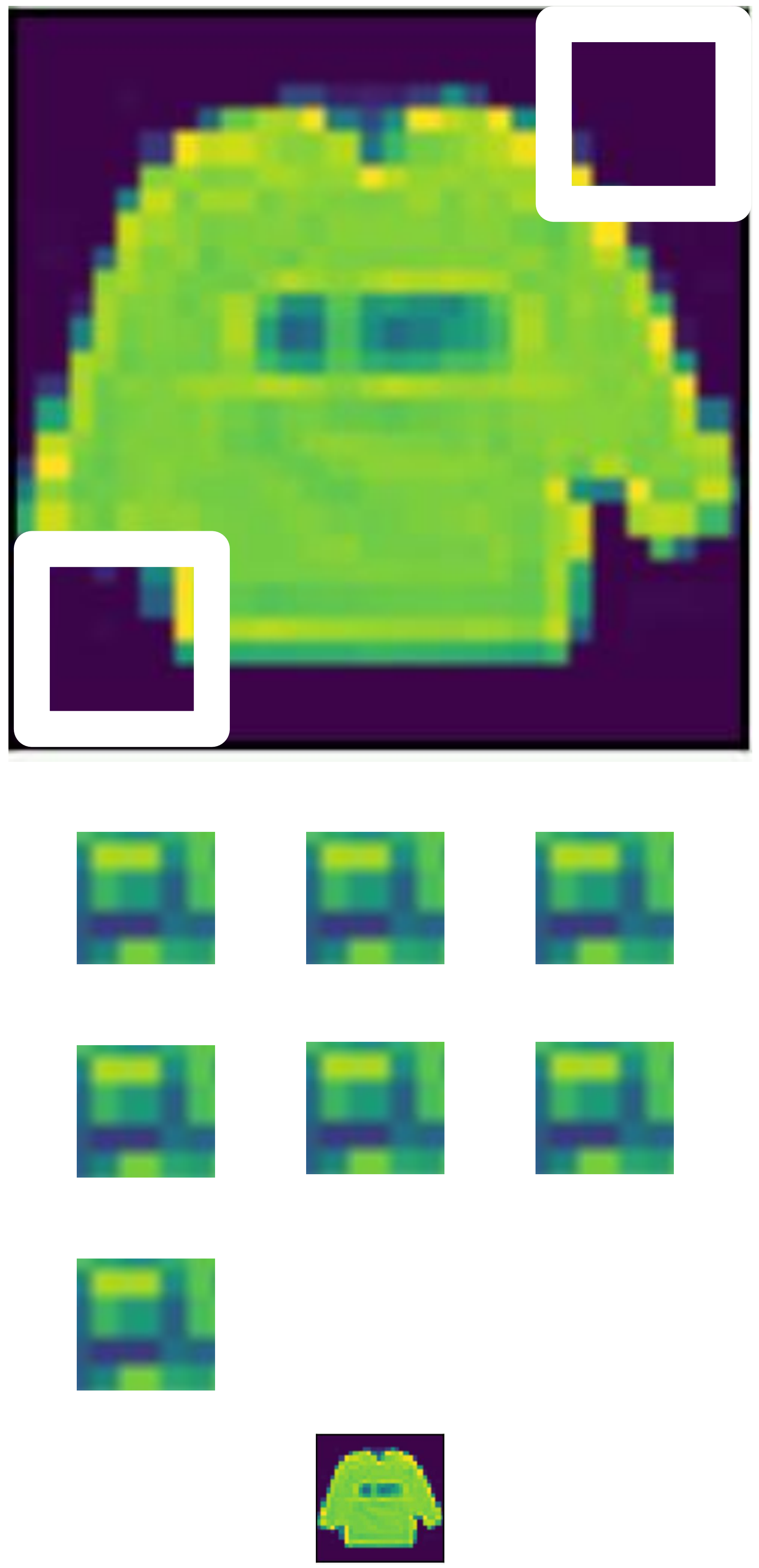
B. $\begin{bmatrix} 0 & 1 \\ 3 & 4 \end{bmatrix}$

$$0\times0 + 1\times1 + 3\times1 + 4\times(-1) + 1 = 1$$
$$1\times0 + 2\times1 + 4\times1 + 5\times(-1) + 1 = 2$$
$$3\times0 + 4\times1 + 6\times1 + 7\times(-1) + 1 = 4$$
$$4\times0 + 5\times1 + 7\times1 + 8\times(-1) + 1 = 5$$

Padding and Stride

Padding

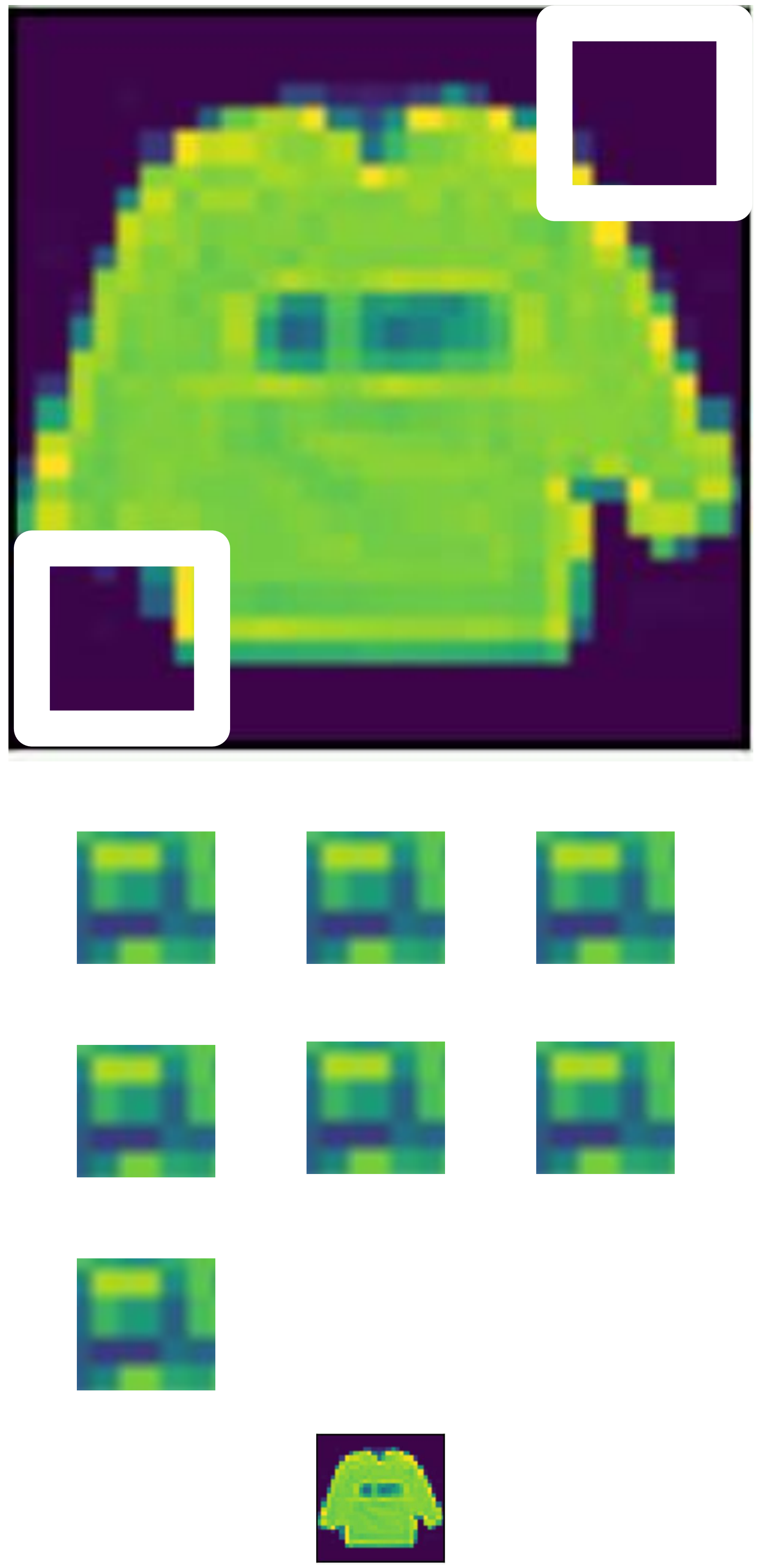
- Given a 32 x 32 input image
- Apply convolution with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers



Padding

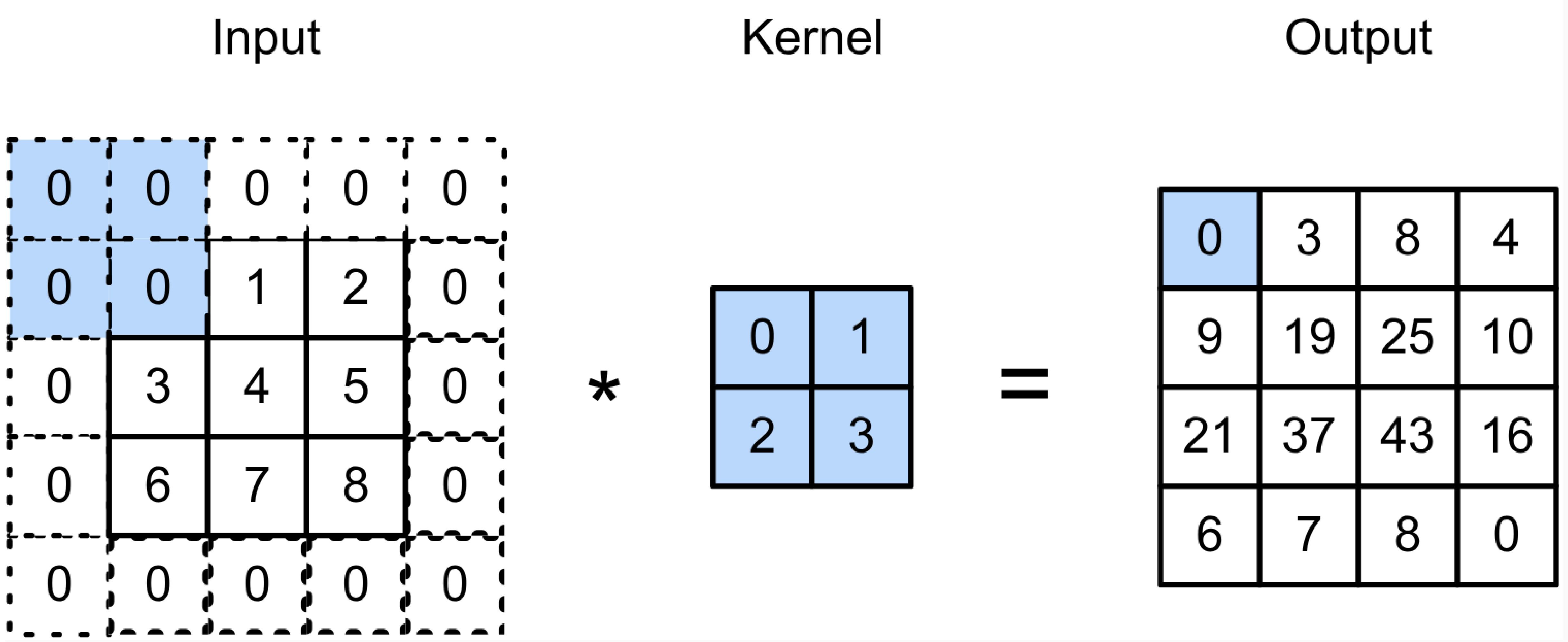
- Given a 32 x 32 input image
- Apply convolution with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to

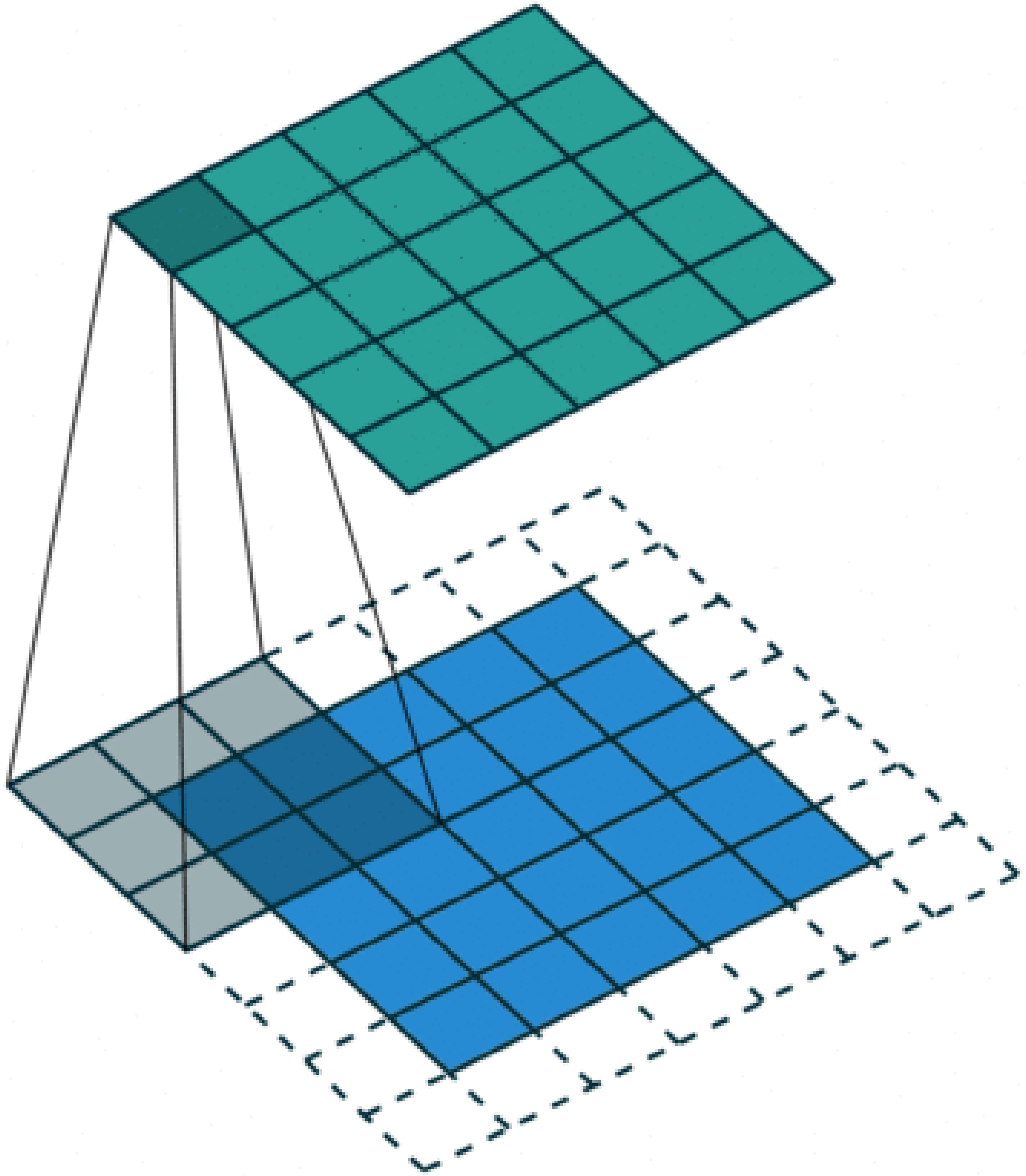
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$



Convolutional Layers: Padding

Padding adds rows/columns around input



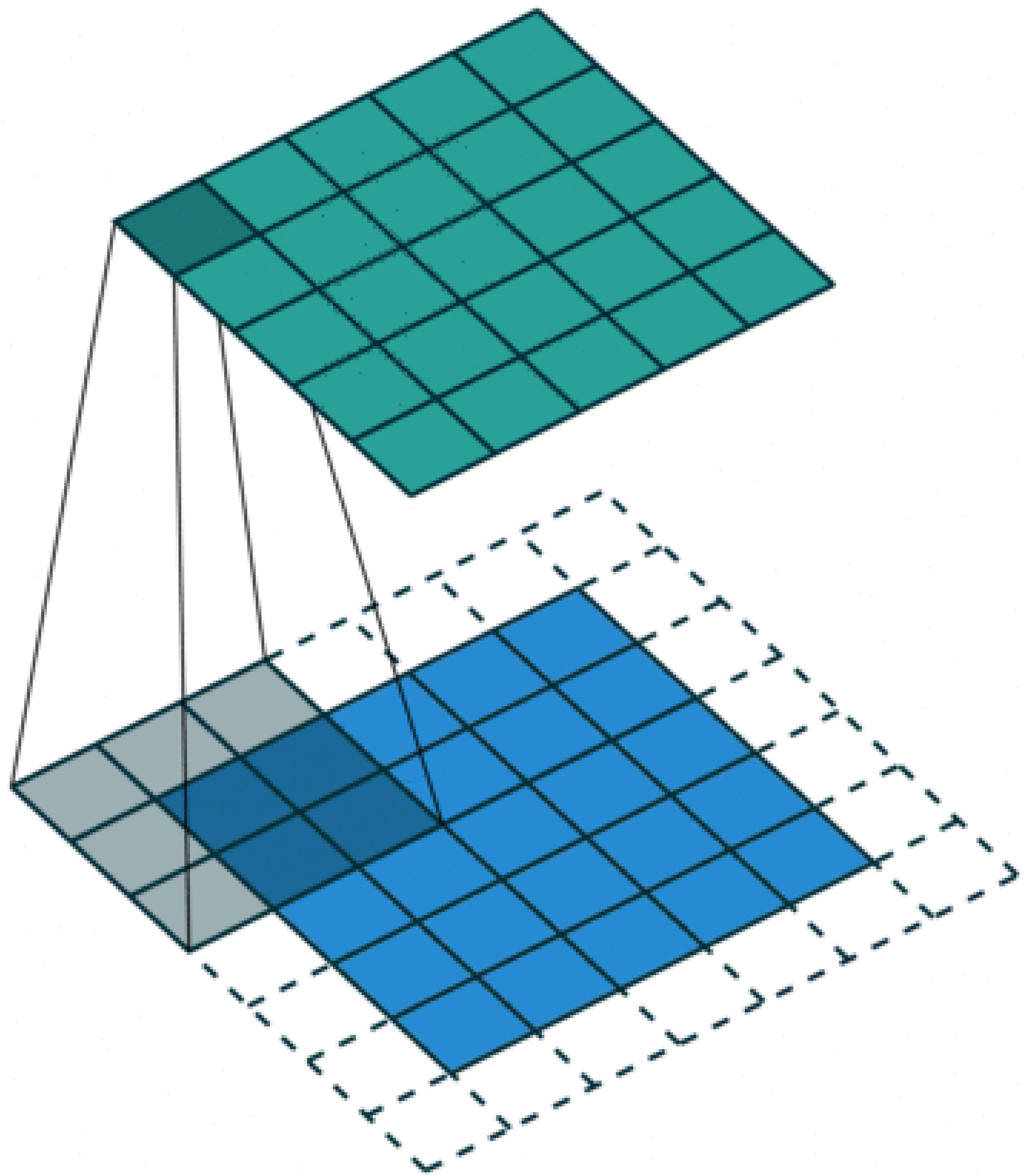


Convolutional Layers: Padding

Padding adds rows/columns around input

- Why?

1. Keeps **edge information**
2. Preserves sizes / allows deep networks
   - ie, for a 32x32 input image, 5x5 kernel, after 1 layer, get 28x28, after 7 layers, **only 4x4**
3. Can combine different filter sizes



Convolutional Layers: Padding

- Padding p_h rows and p_w columns, output shape is

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

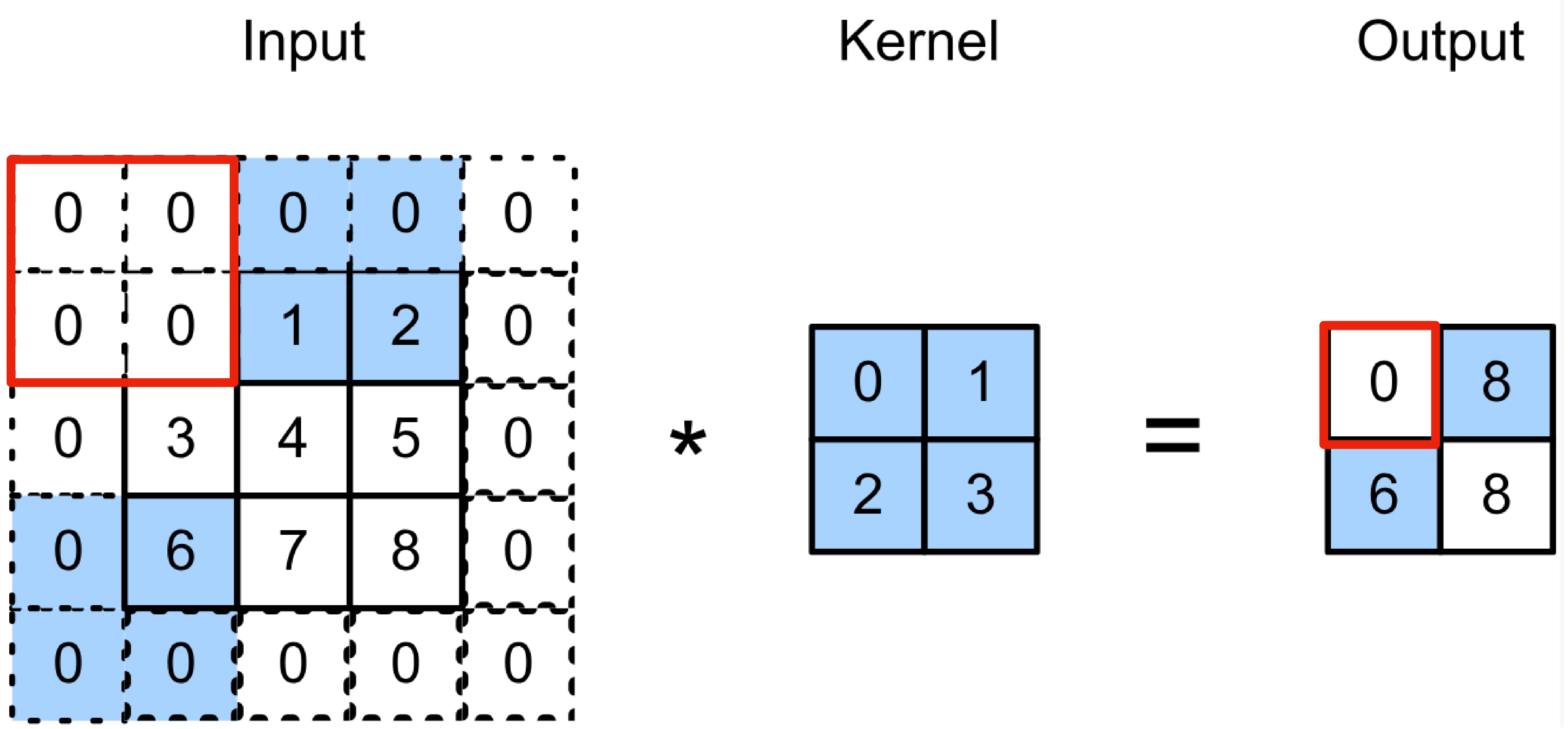
- Common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$

  - Odd k_h: pad $p_h/2$ on both sides
  - Even k_h: pad ceil($p_h/2$) on top, floor($p_h/2$) on bottom

Stride

- Stride is the #rows/#columns per slide

Strides of 3 and 2 for height and width

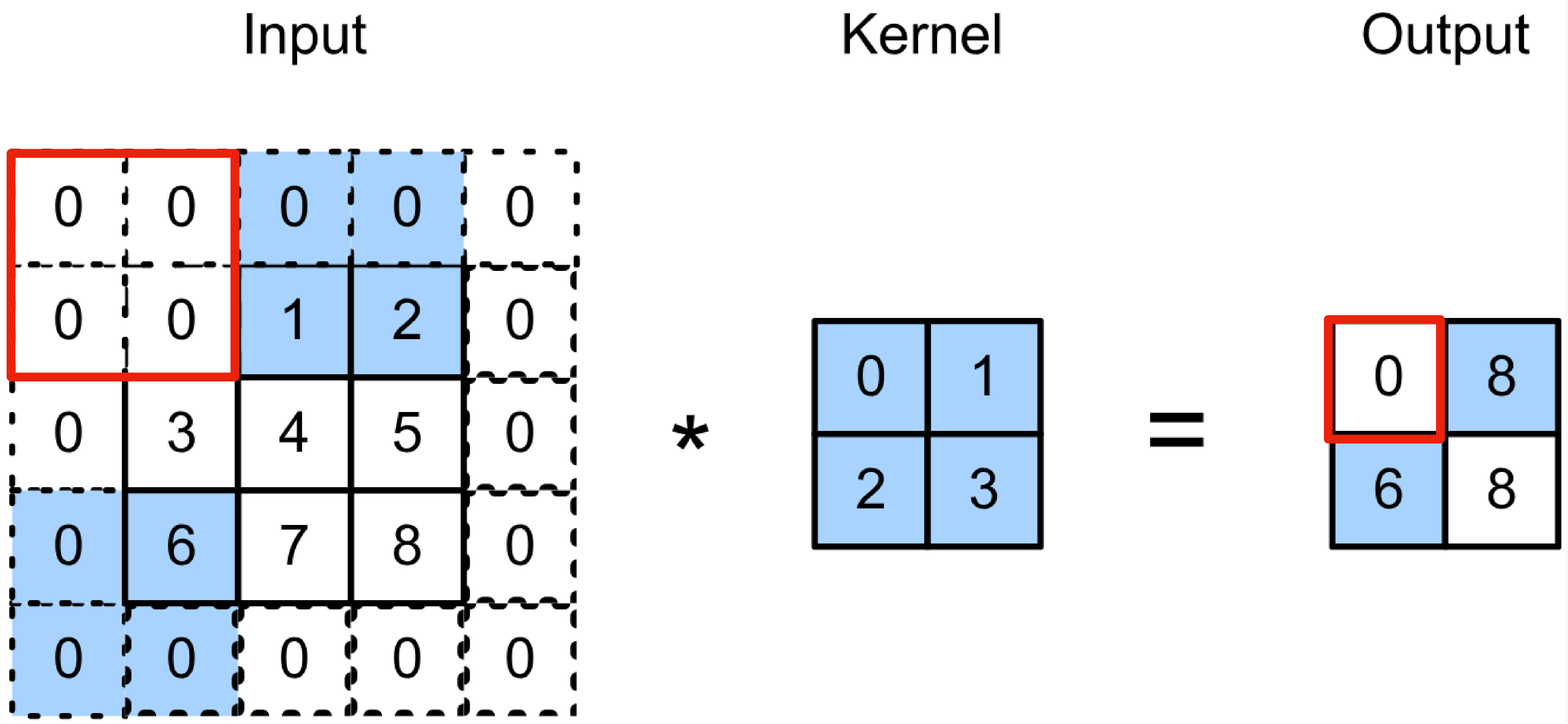


$$0\times0+0\times1+1\times2+2\times3=8$$
$$0\times0+6\times1+0\times2+0\times3=6$$

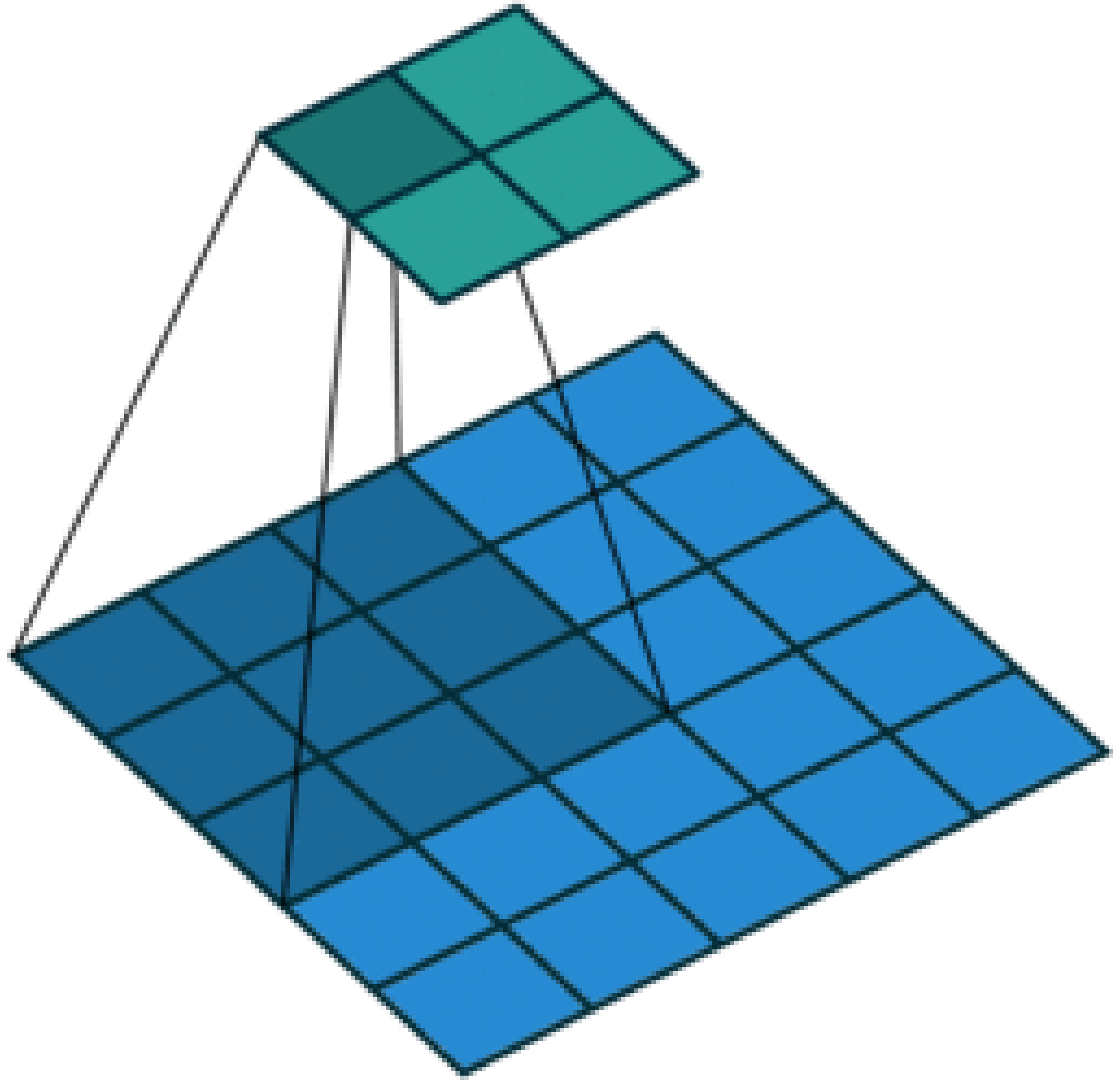
Stride

- Stride is the #rows/#columns per slide

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 2,2

Convolutional Layers: Stride

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

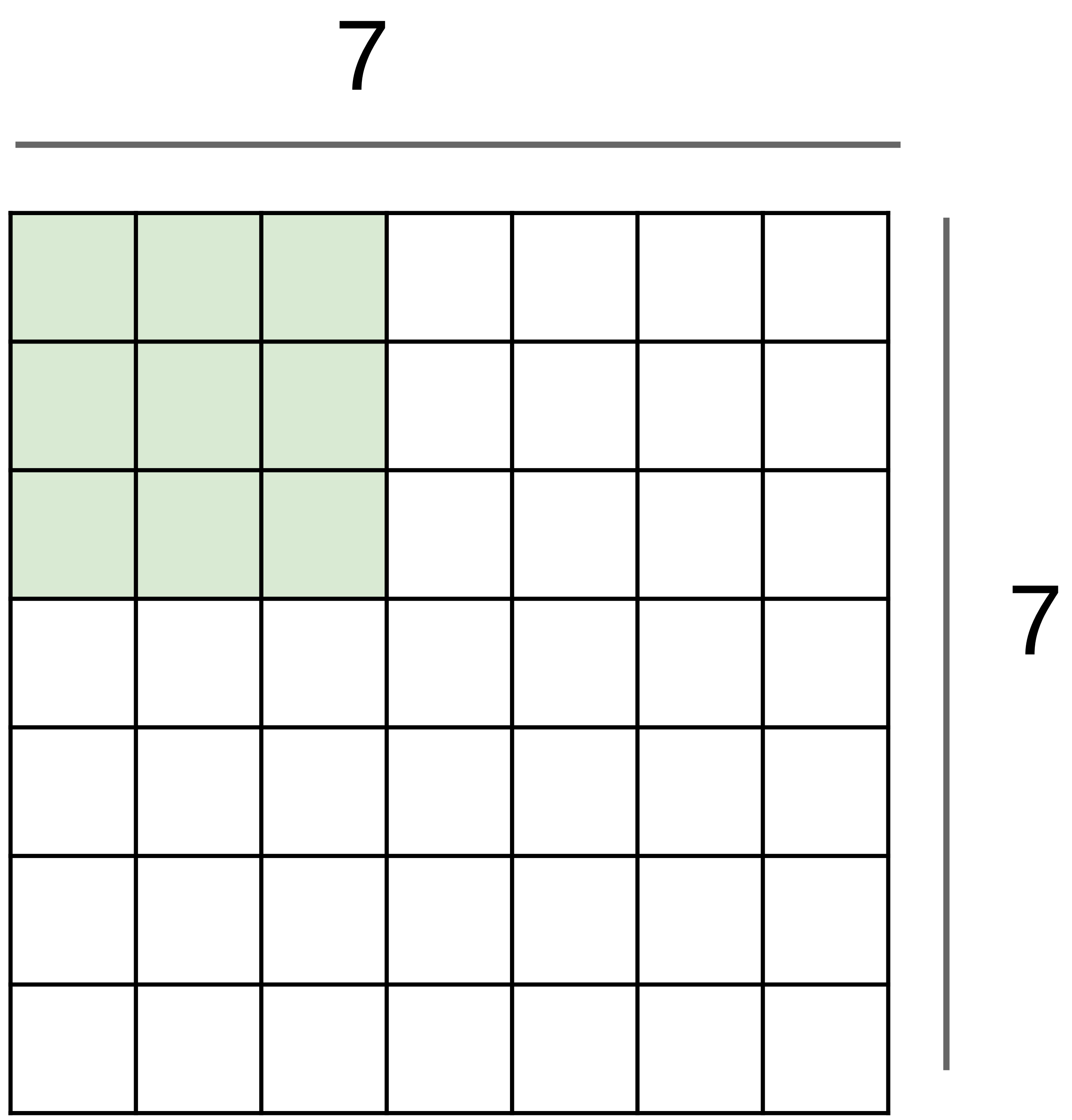
$$\lfloor(n_h-k_h+p_h+s_h)/s_h\rfloor \times \lfloor(n_w-k_w+p_w+s_w)/s_w\rfloor$$

- Set $p_h = k_h-1$, $p_w = k_w-1$, then get

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

Q2. 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



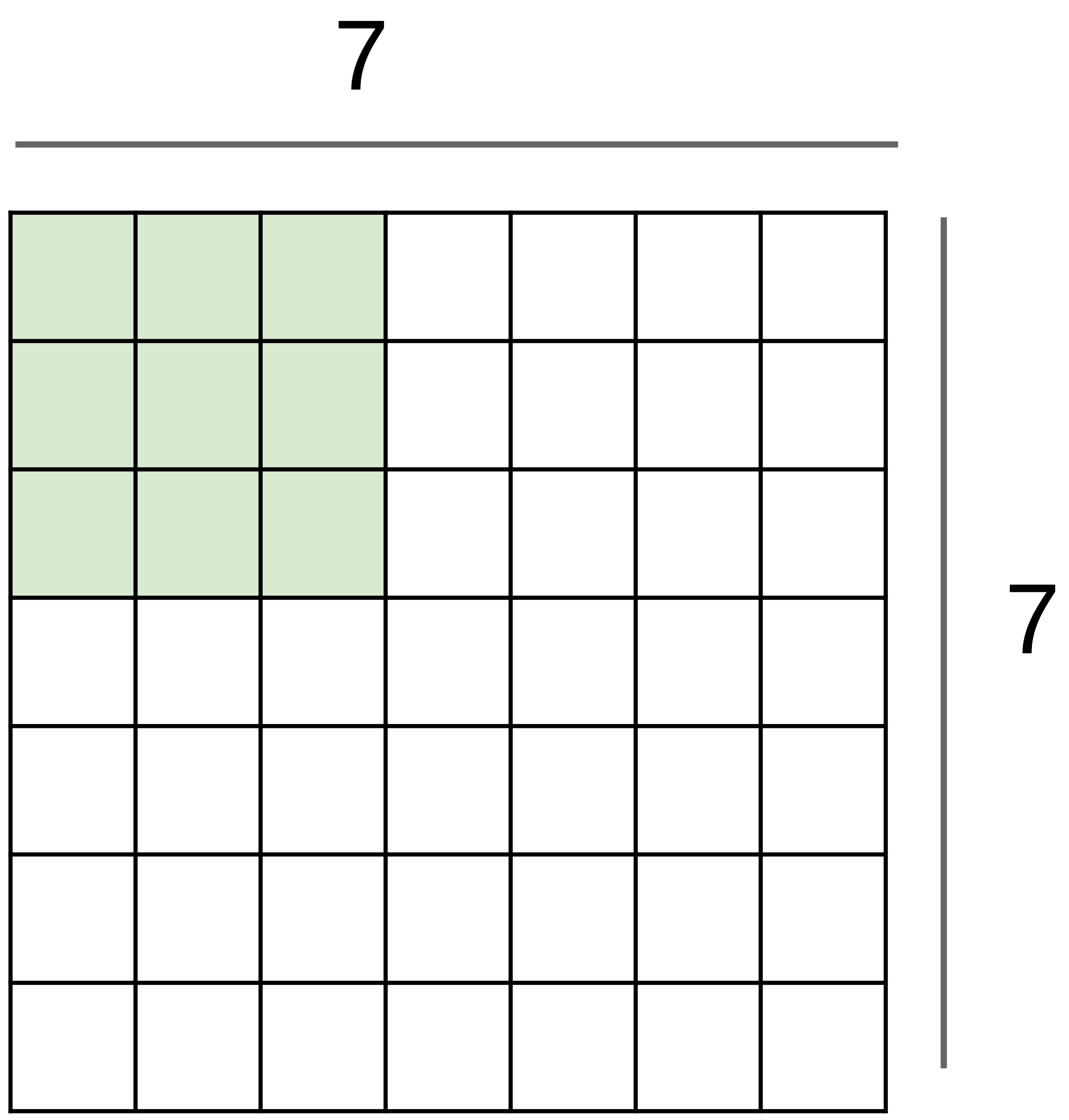
Q2. Suppose we want to perform convolution on a single channel image of size 7x7 (no padding) with a kernel of size 3x3, and stride = 2. What is the dimension of the output?

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

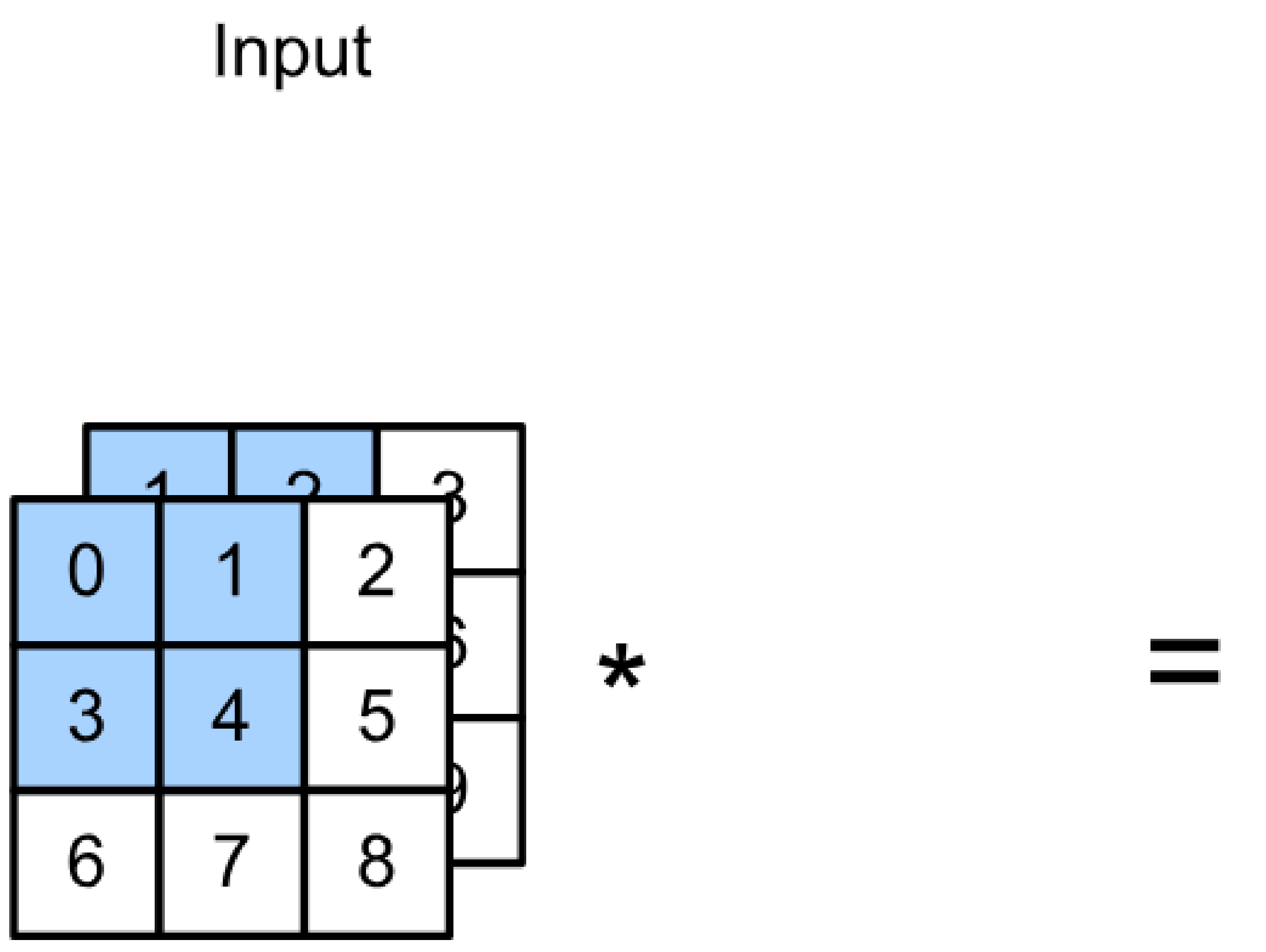
- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

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

Multiple Input Channels

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



Convolutional Layers: Channels

- How to integrate multiple channels?
  - Have a kernel for each channel, and then sum results over channels

$$\mathbf{X} : c_i \times n_h \times n_w$$

$$\mathbf{W} : c_i \times k_h \times k_w$$

$$\mathbf{Y} : m_h \times m_w$$

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

Multiple Output Channels

- No matter how many inputs channels, so far we always get single output channel
- We can have **multiple 3-D kernels**, each one generates an output channel

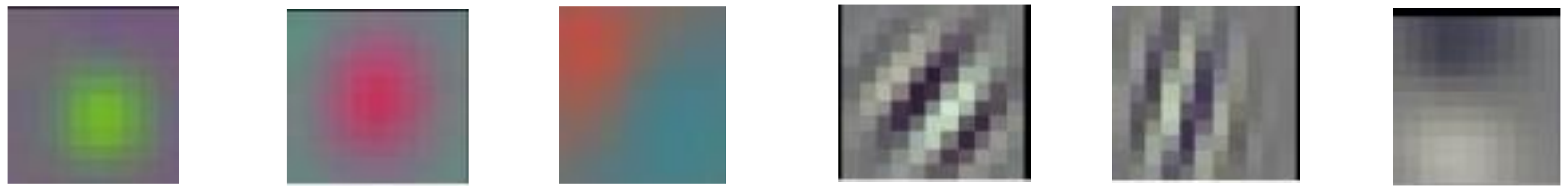
Multiple Output Channels

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

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$
$$\text{for } i = 1, \ldots, c_o$$

Multiple Input/Output Channels

- Each 3-D kernel may recognize a particular pattern



(Gabor filters)

Q3. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 kernels, each with height 3 and width 3. 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. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 kernels, each with height 3 and width 3. 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

For each kernel, we slide the window to 222x222 different locations. For each location, the number of multiplication is 3x3x3. So in total 64x3x3x3x222x222

Q4. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 kernels, each with height 3 and width 3. Stride = 1. The convolution layer has bias parameters. Which is a reasonable estimate of the total number of learnable parameters?

A. 64x222x222

B. 64x3x3x222

C. 3x3x3x64

D. (3x3x3+1)x64

Q4. Suppose we want to perform convolution on a RGB image of size 224x224 (no padding) with 64 kernels, each with height 3 and width 3. Stride = 1. The convolution layer has bias parameters. Which is a reasonable estimate of the total number of learnable parameters?

A. 64x222x222

B. 64x3x3x222

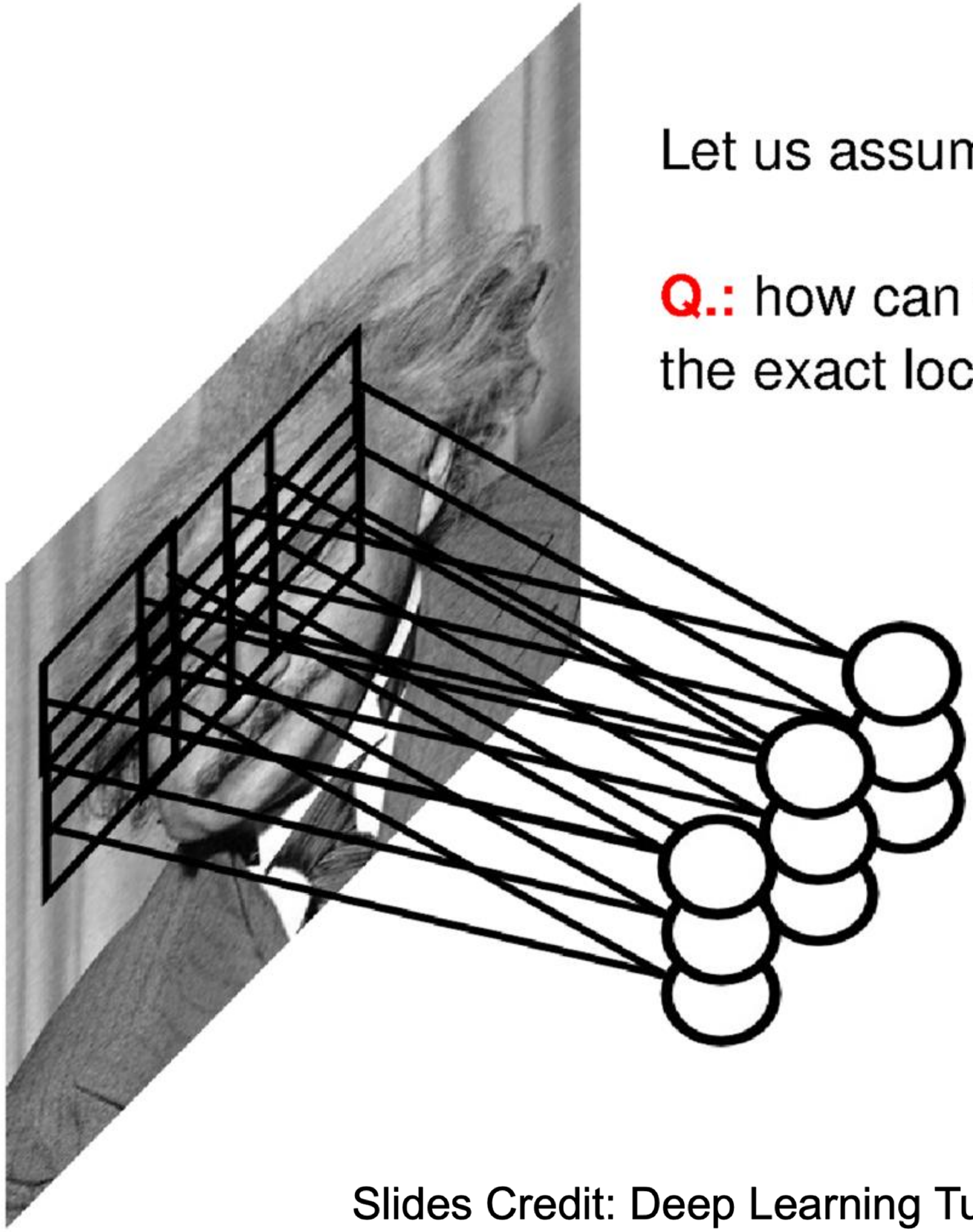
C. 3x3x3x64

D. (3x3x3+1)x64

Each kernel is 3D kernel across 3 input channels, so has 3x3x3 parameters. Each kernel has 1 bias parameter. So in total (3x3x3+1)x64

Pooling Layer

Pooling

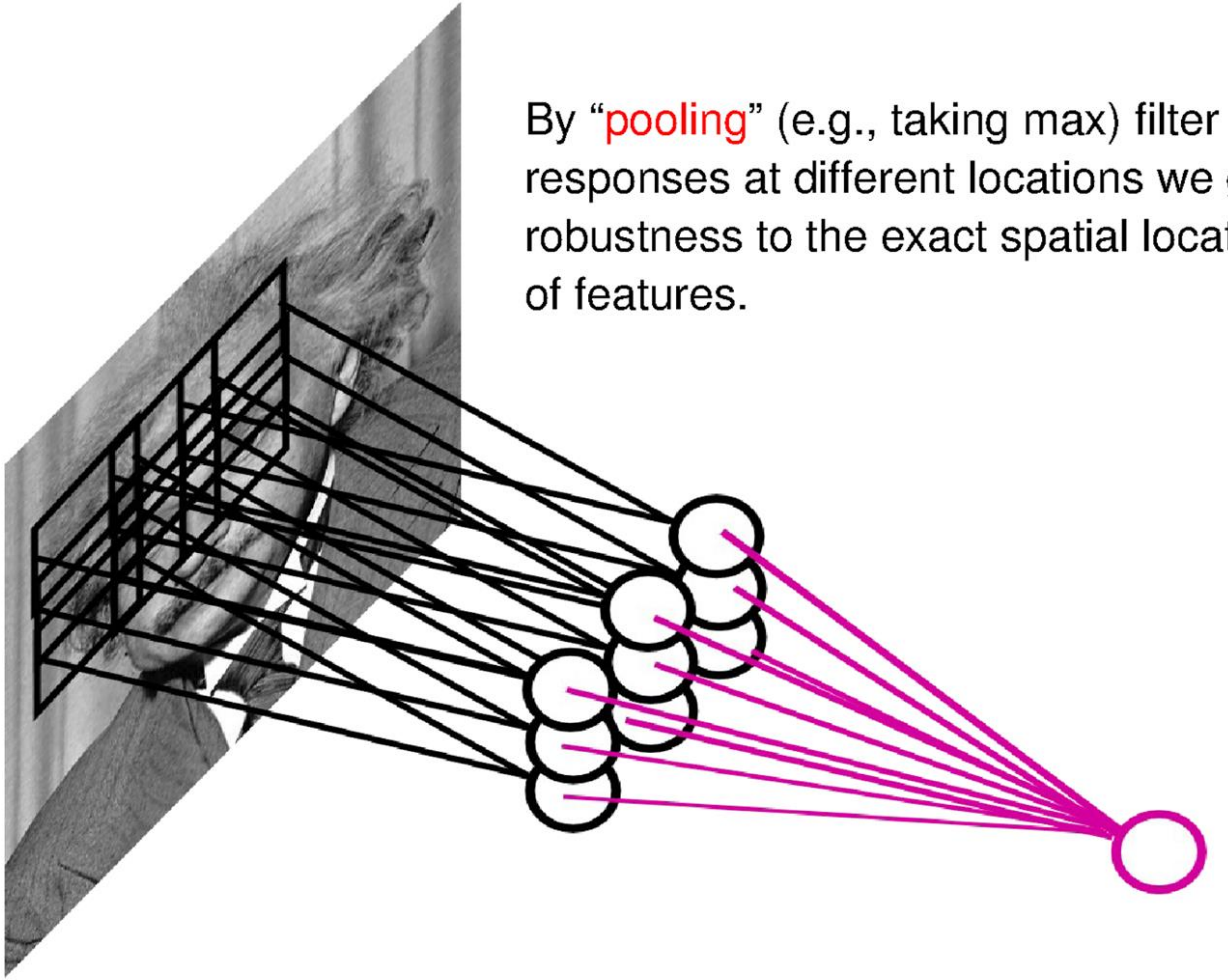


Let us assume filter is an “eye” detector.

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

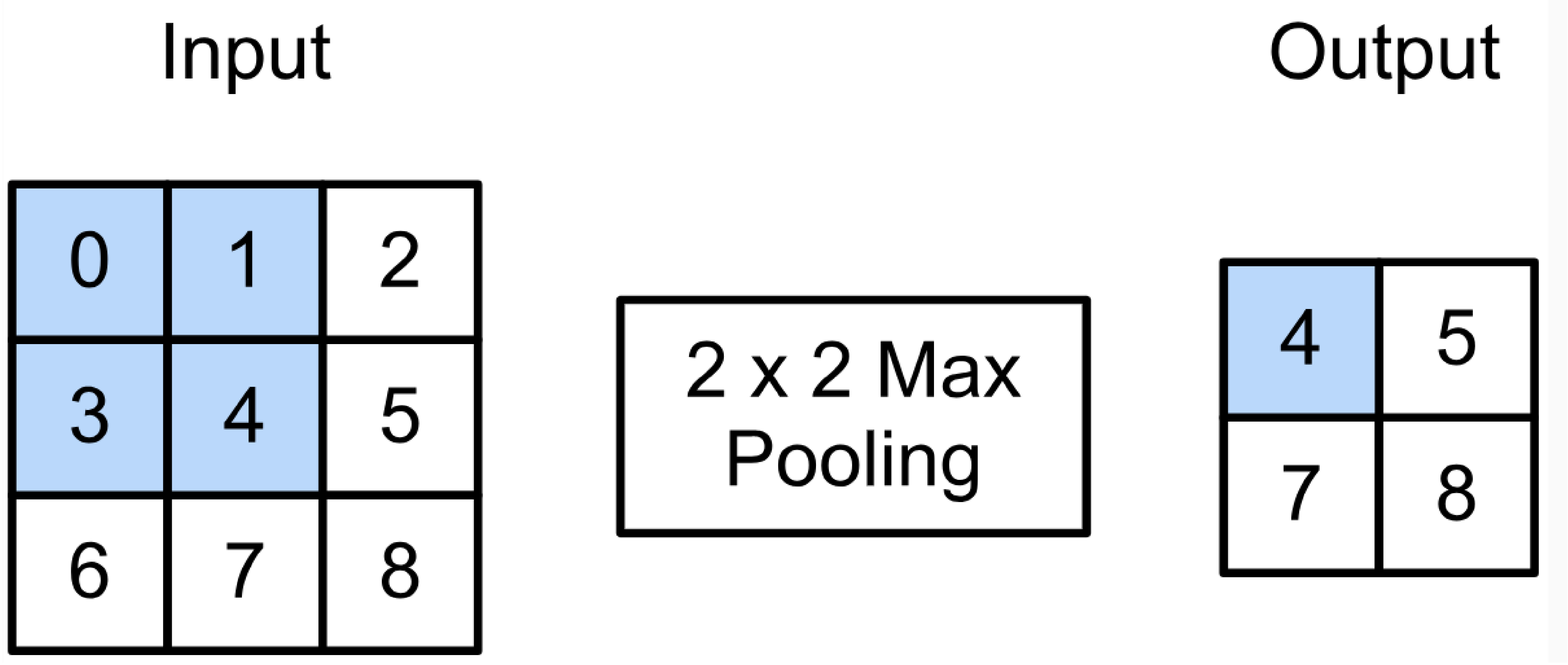


By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Slides Credit: Deep Learning Tutorial by Marc'Aurelio Ranzato

2-D Max Pooling

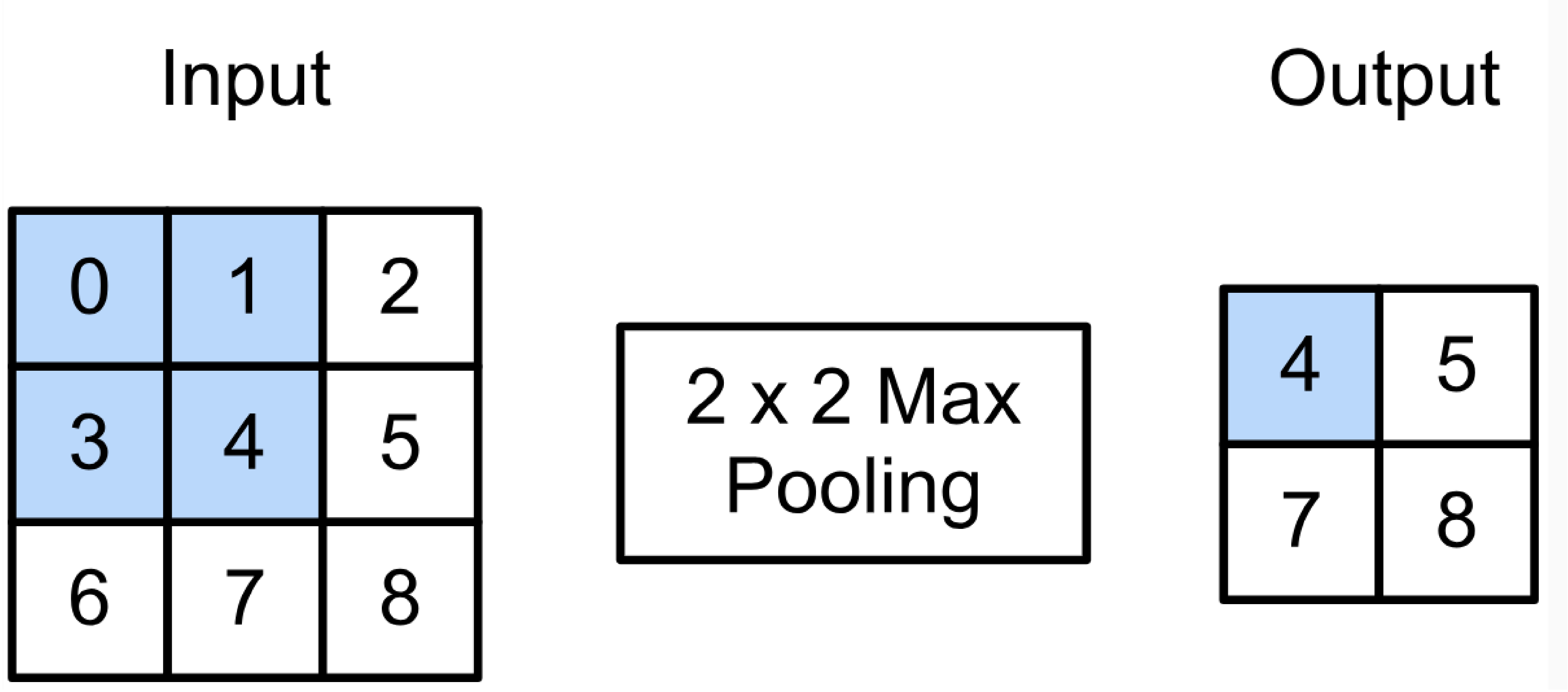
- Returns the maximal value in the sliding window



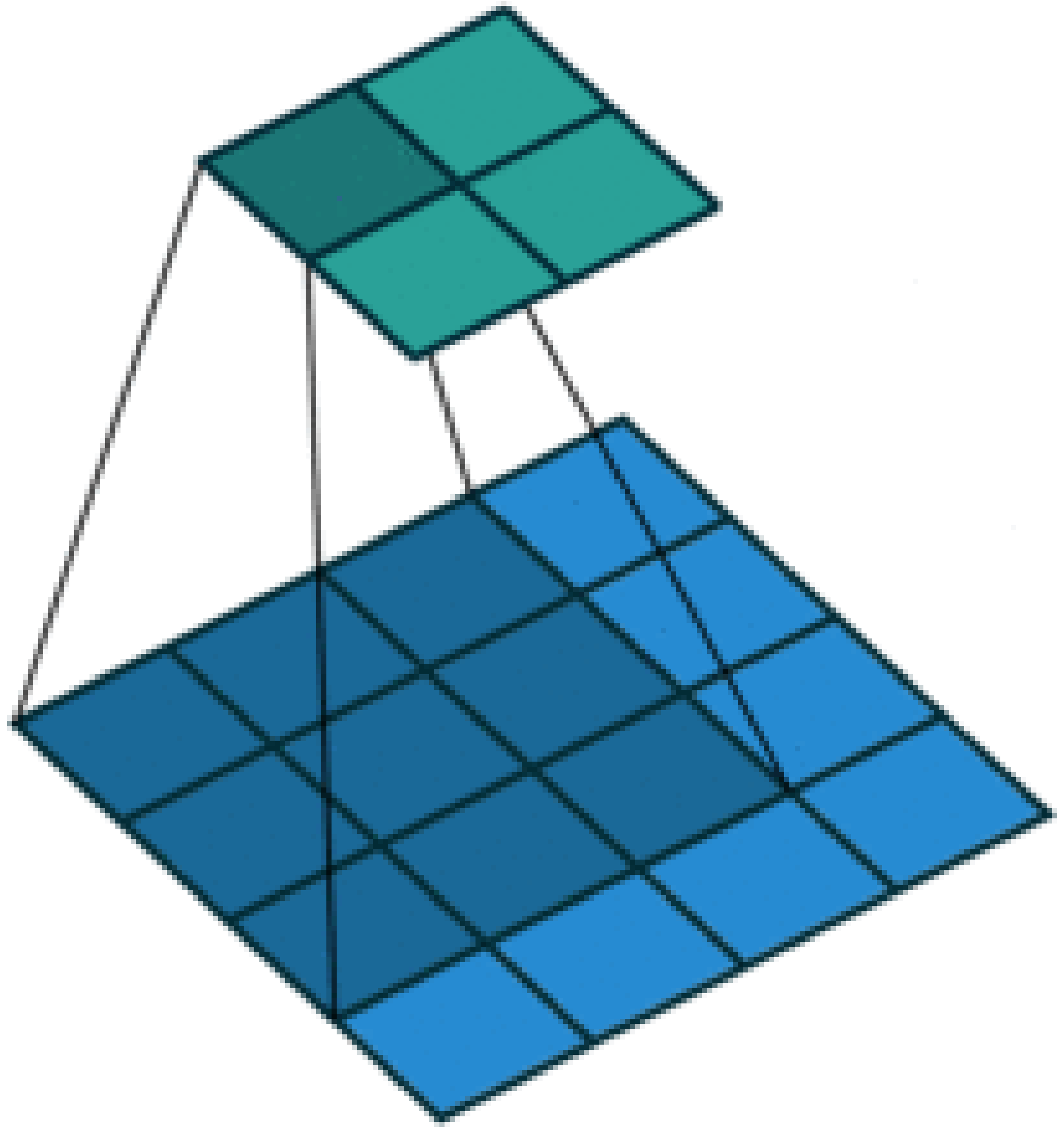
$max(0,1,3,4) =$

2-D Max Pooling

- Returns the maximal value in the sliding window



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



Padding, Stride, and Multiple Channels

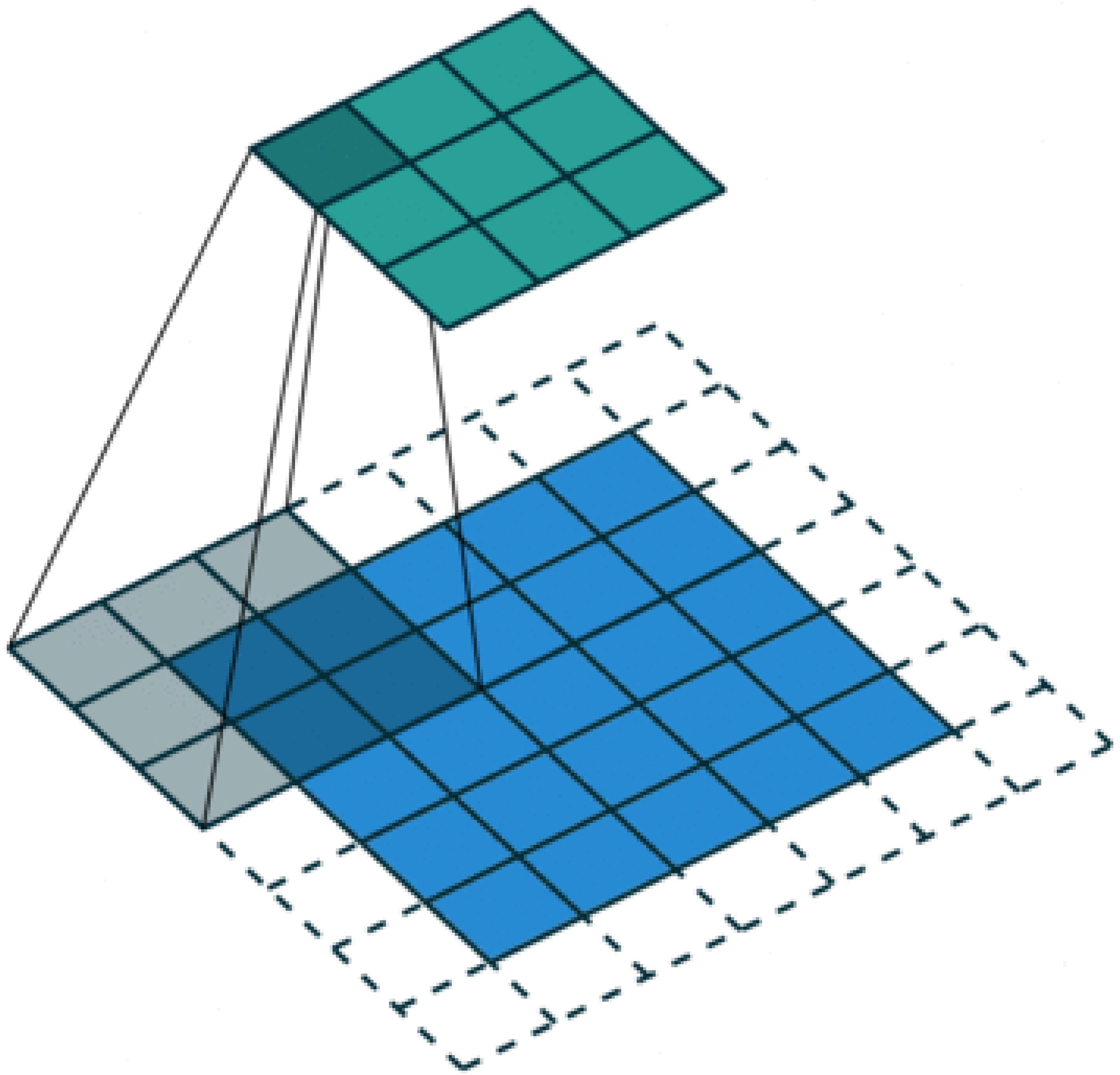
- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel

  #output channels = #input channels

Padding, Stride, and Multiple Channels

- 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

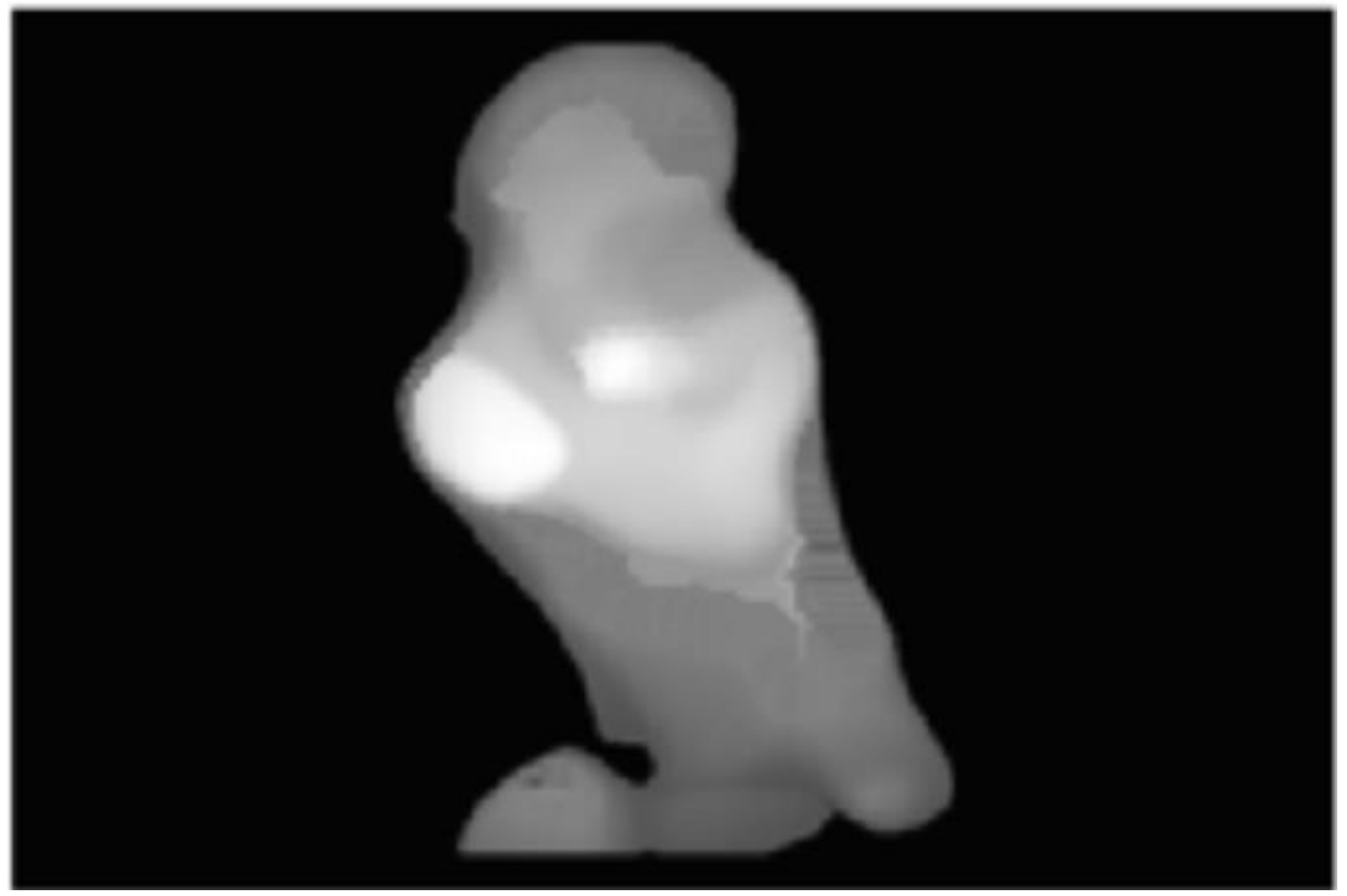
  #output channels = #input channels



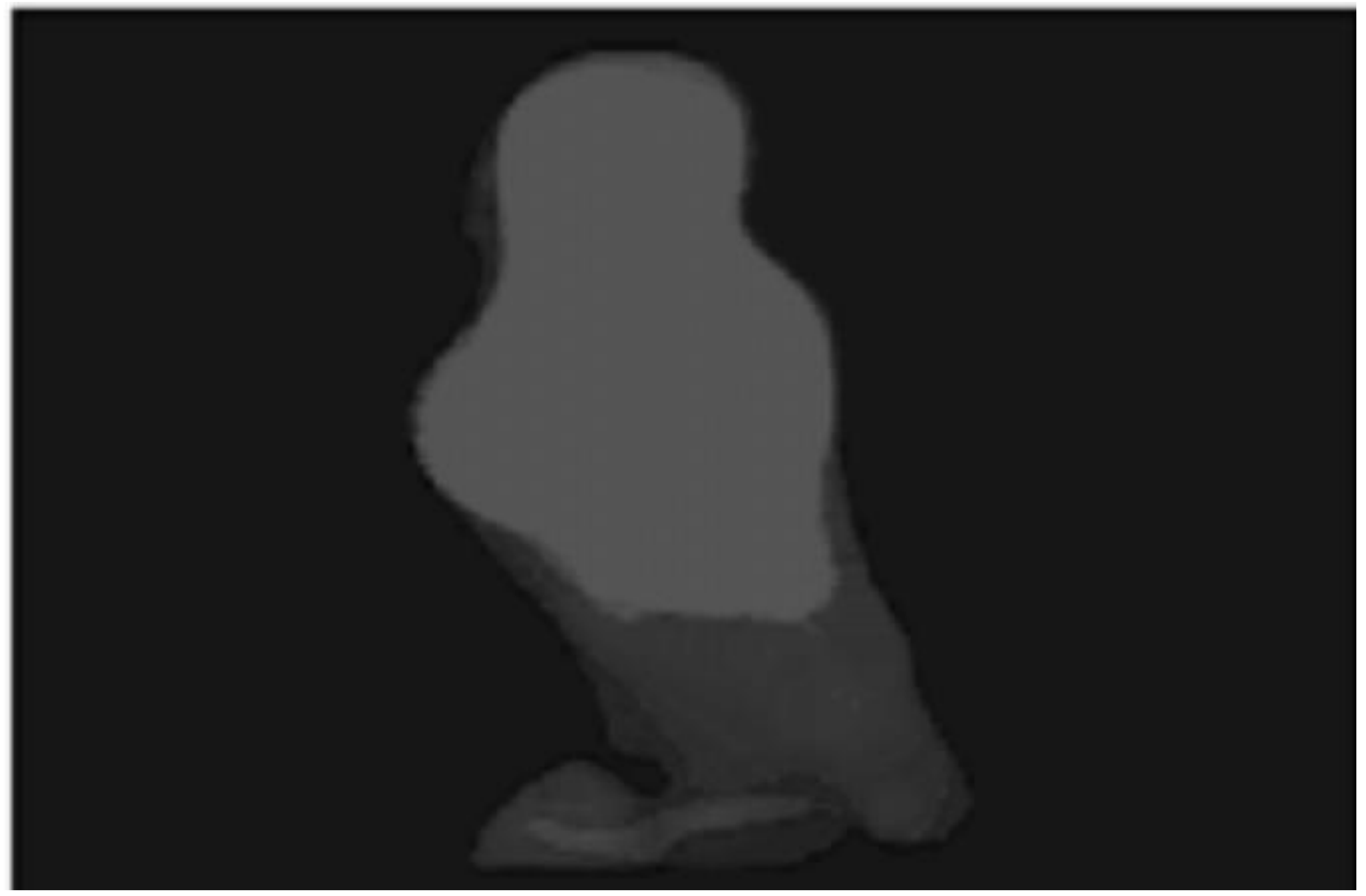
Average Pooling

- 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



Q5. 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

Q5. 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

Q6. What is the output if we replace average pooling with 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

Q6. What is the output if we replace average pooling with 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

Summary

- Intro of convolutional computations
  - 2D convolution
  - Padding, stride
  - Multiple input and output channels
  - Pooling

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