CS540 Introduction to Artificial Intelligence

Deep Learning I: Convolutional Neural Networks

Yingyu Liang
University of Wisconsin-Madison

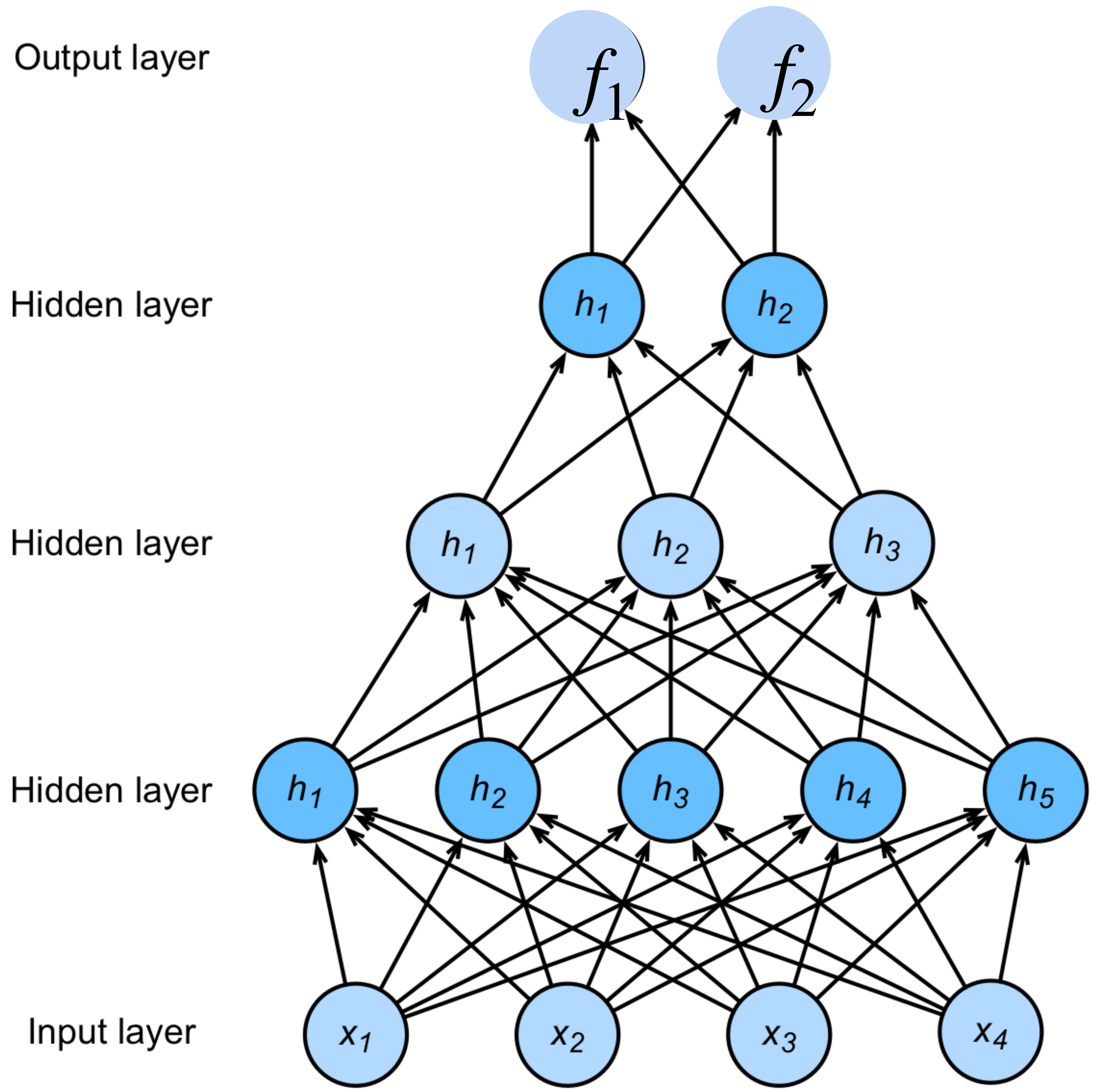
Nov 2, 2021

Slides created by Sharon Li [modified by Yingyu Liang]

Outline

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

Review: Deep neural networks (DNNs)



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

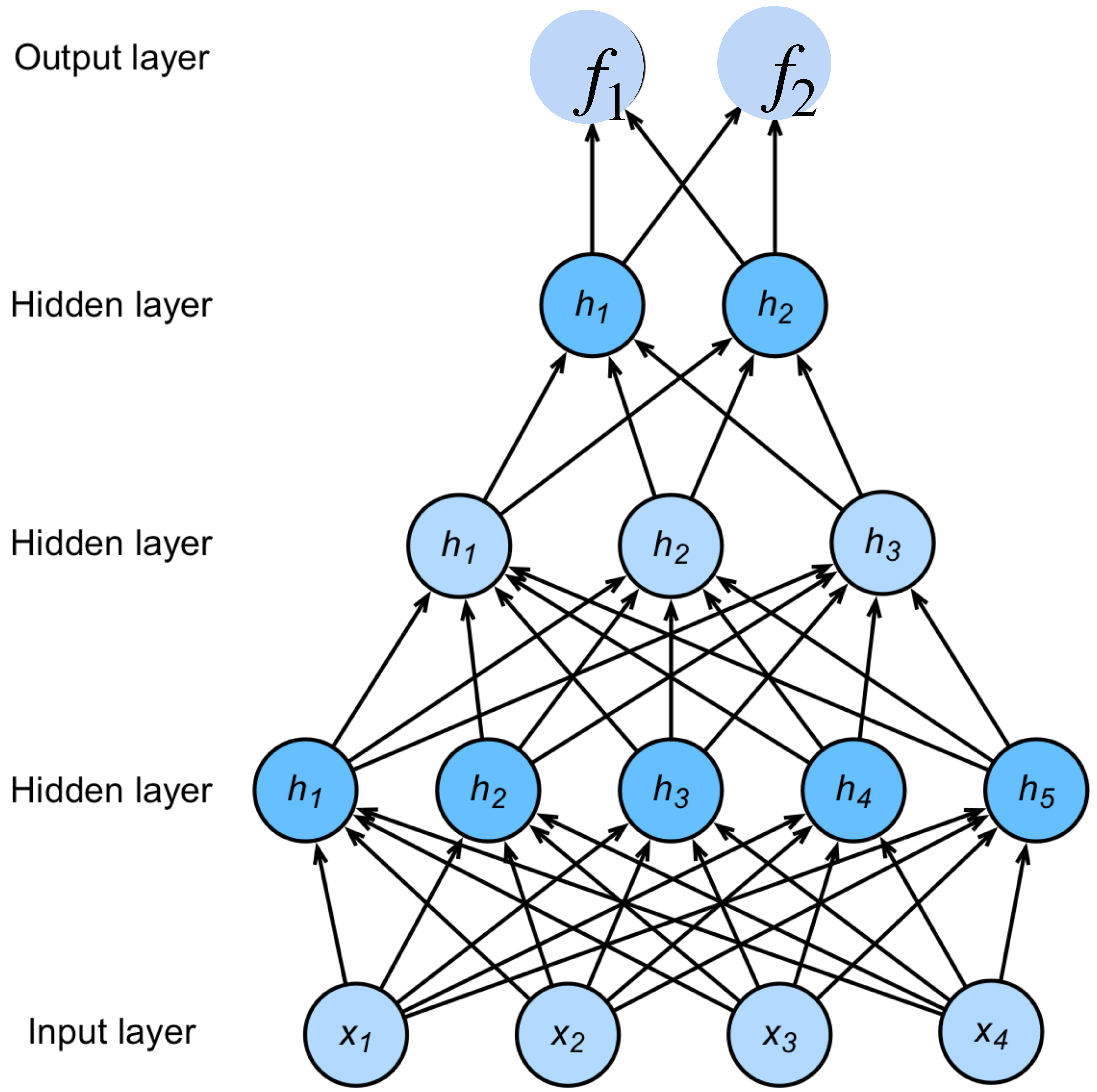
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$

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

$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$

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

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?

How to classify

Cats vs. dogs?

How to classify

Cats vs. dogs?

Dual

12MP

wide-angle and
telephoto cameras

How to classify

Cats vs. dogs?

Dual

12MP

wide-angle and
telephoto cameras

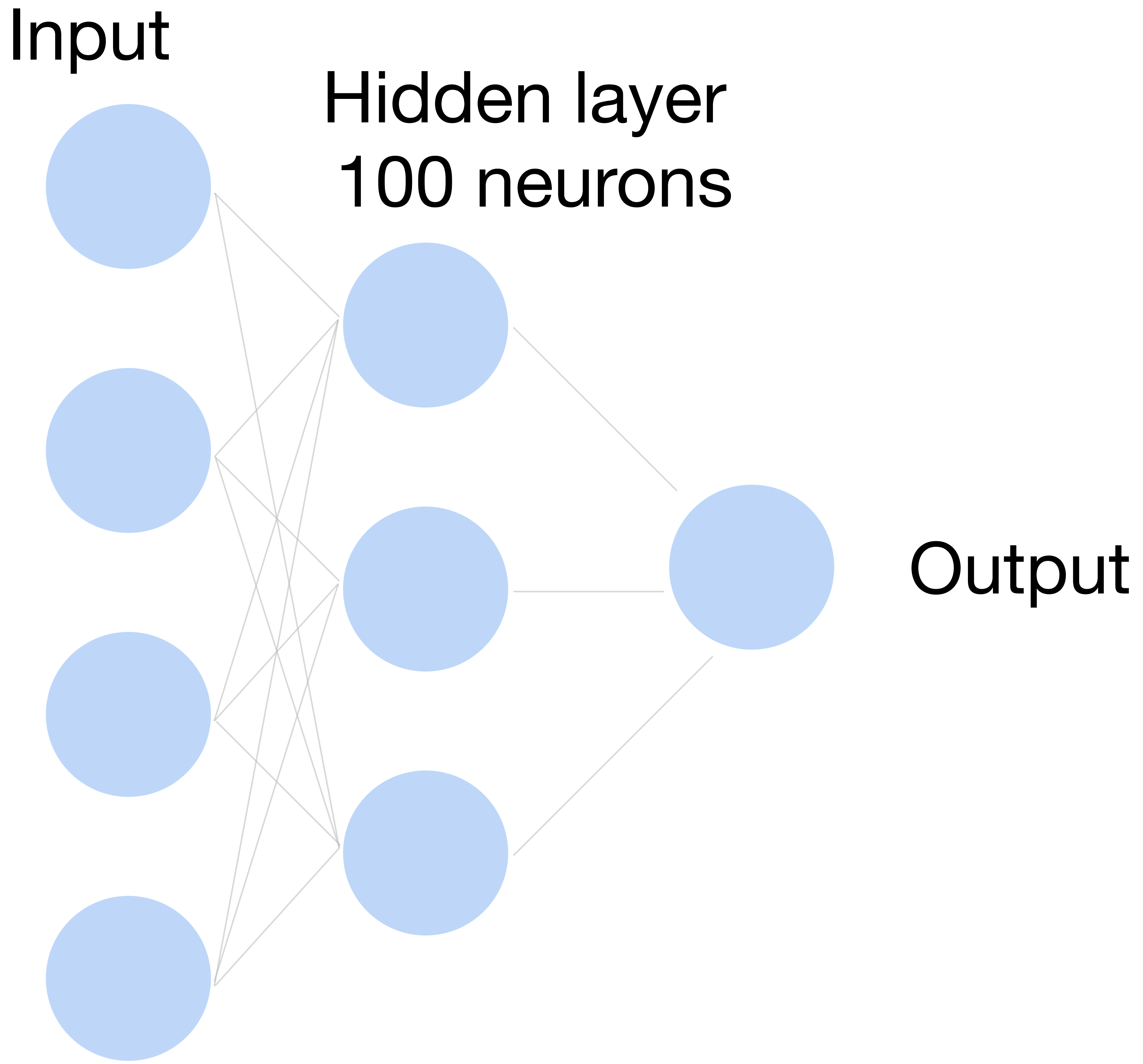
36M floats in a RGB image!

Fully Connected Networks

Cats vs. dogs?

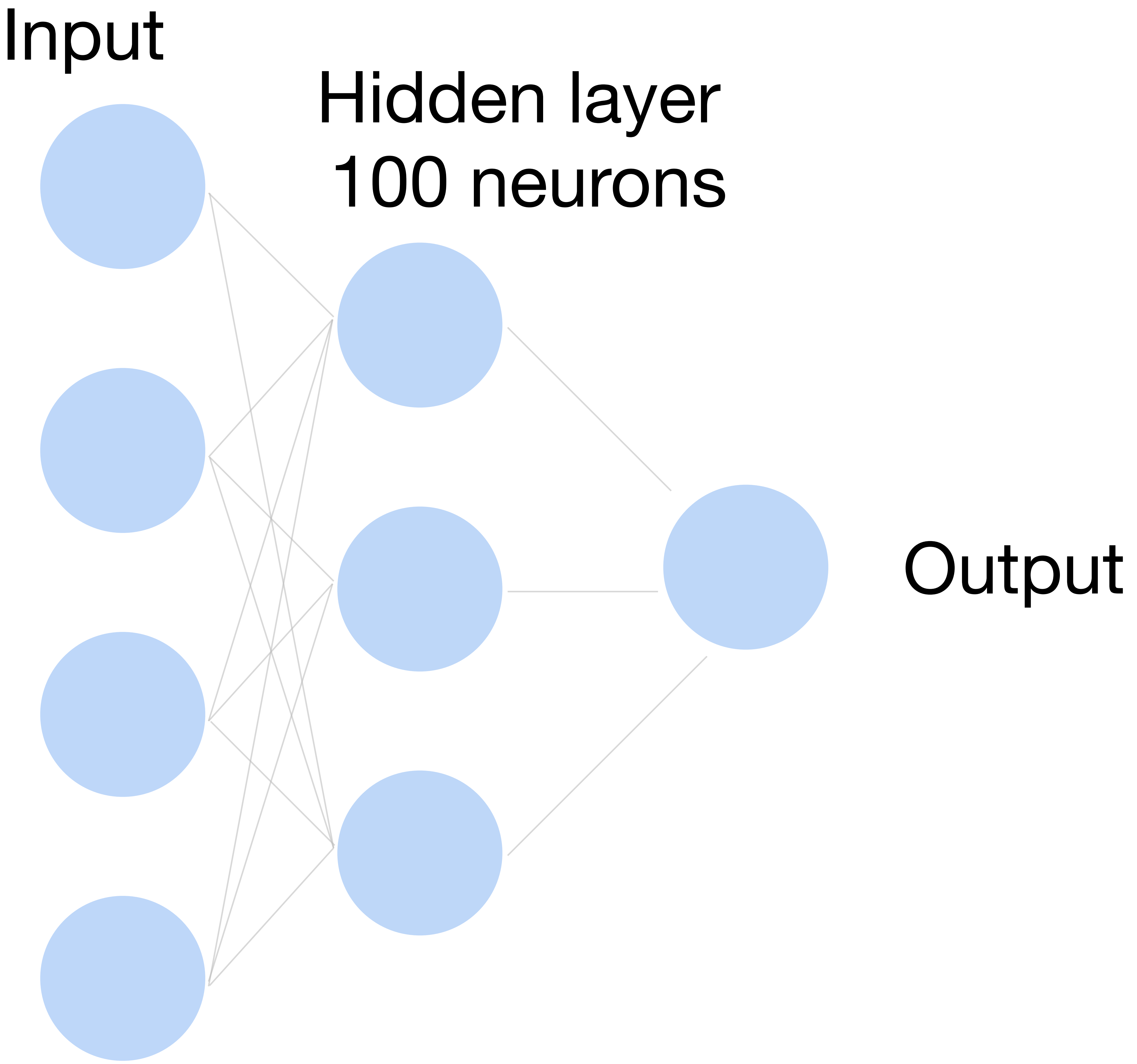
Fully Connected Networks

Cats vs. dogs?



Fully Connected Networks

Cats vs. dogs?



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

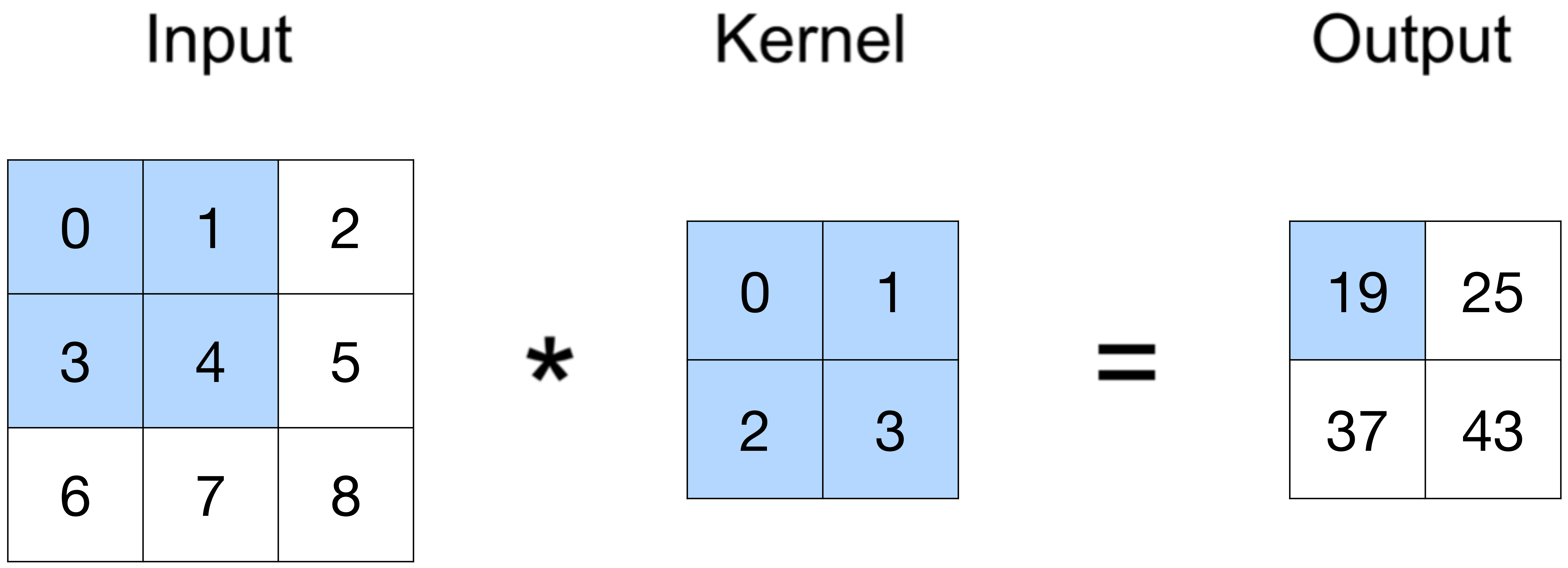
Convolutions come to rescue!

Where is Waldo?

Why Convolution?

- Translation Invariance
- Locality

2-D Convolution



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

2-D Convolution

Input

0	1	2
3	4	5
6	7	8

*

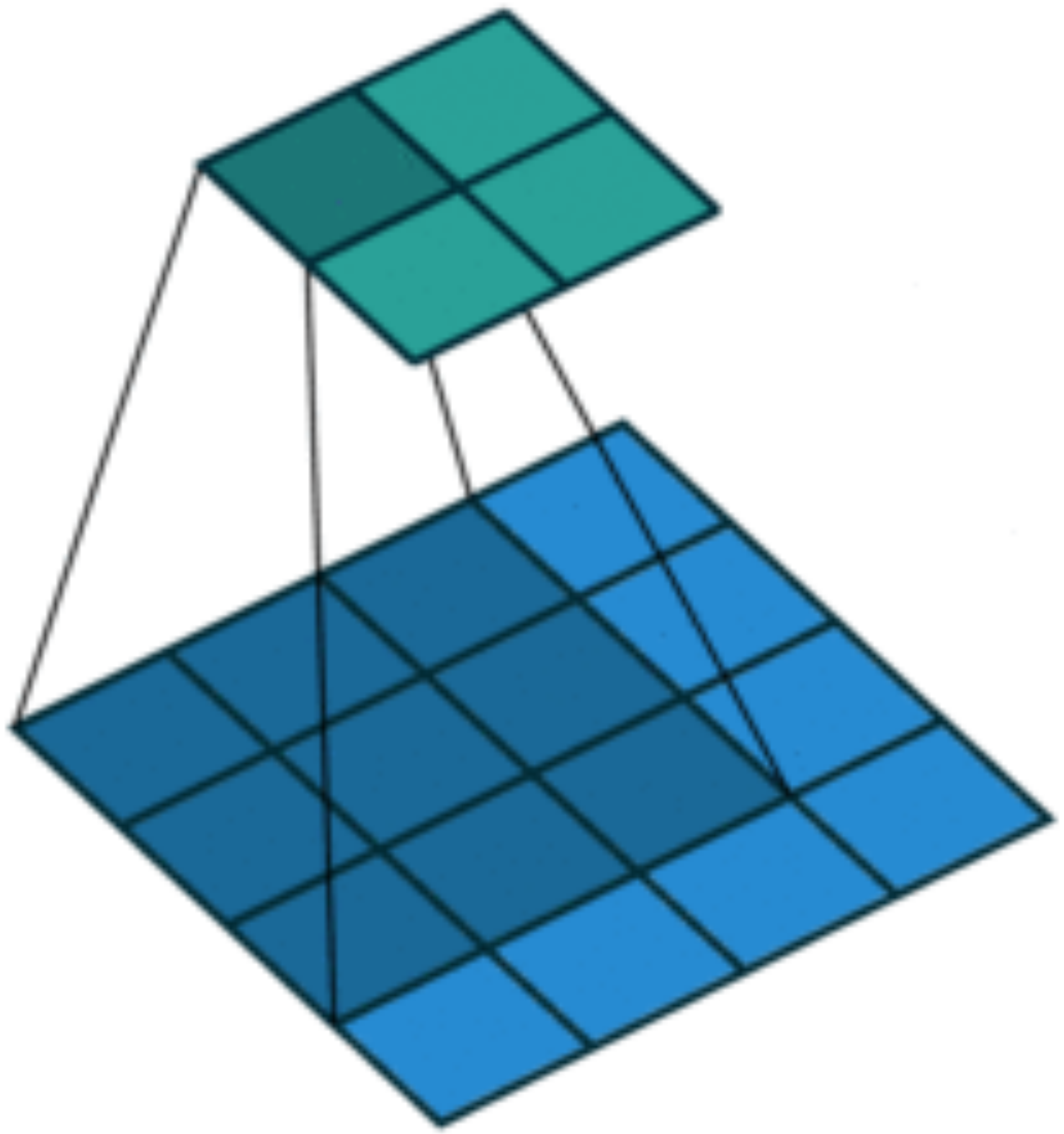
Kernel

0	1
2	3

=

Output

19	25
37	43



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

2-D Convolution

Input

0	1	2
3	4	5
6	7	8

*

Kernel

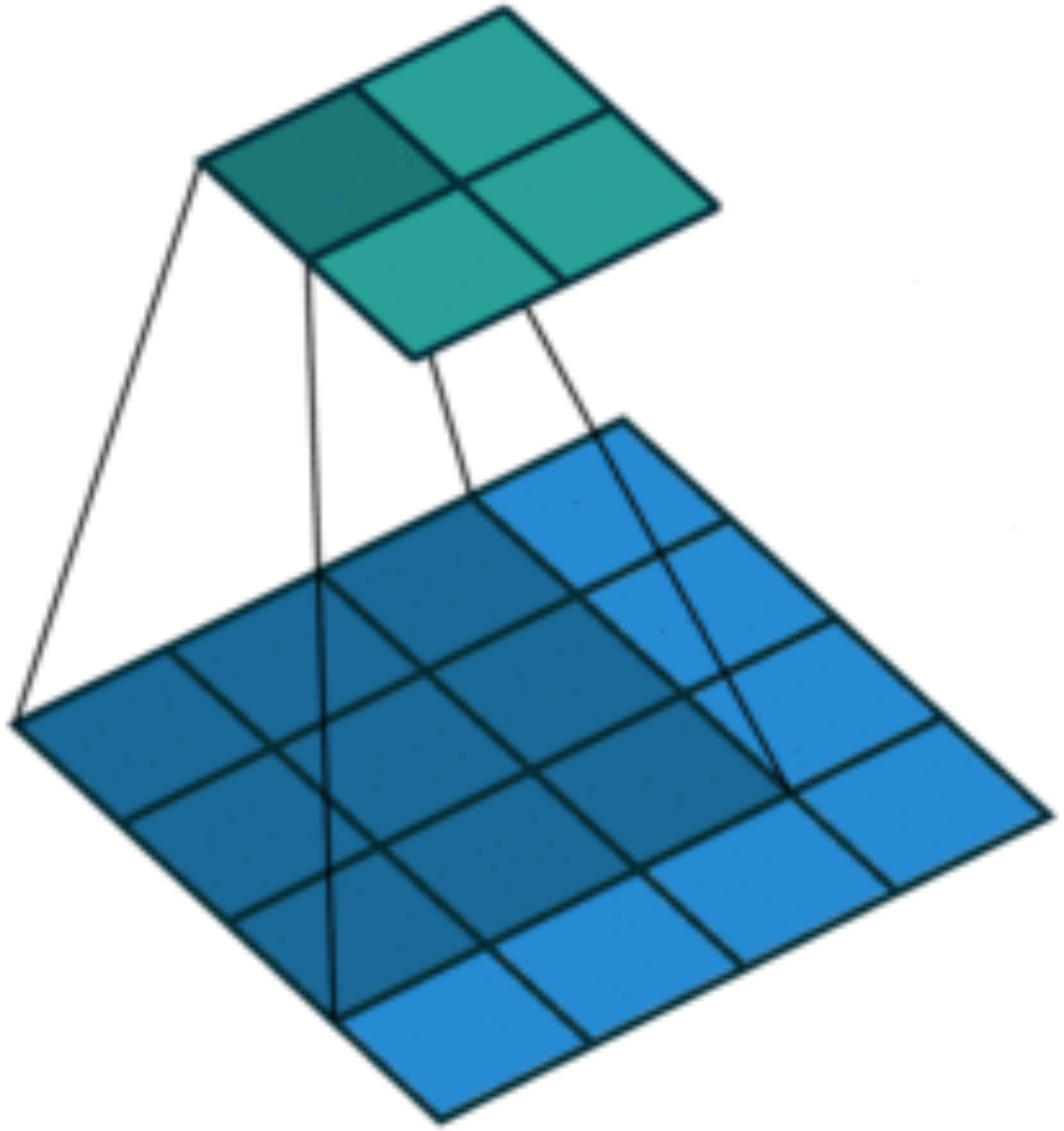
0	1
2	3

=

Output

19	25
37	43

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



(vdumoulin@ Github)

2-D Convolution

Input

0	1	2
3	4	5
6	7	8

*

Kernel

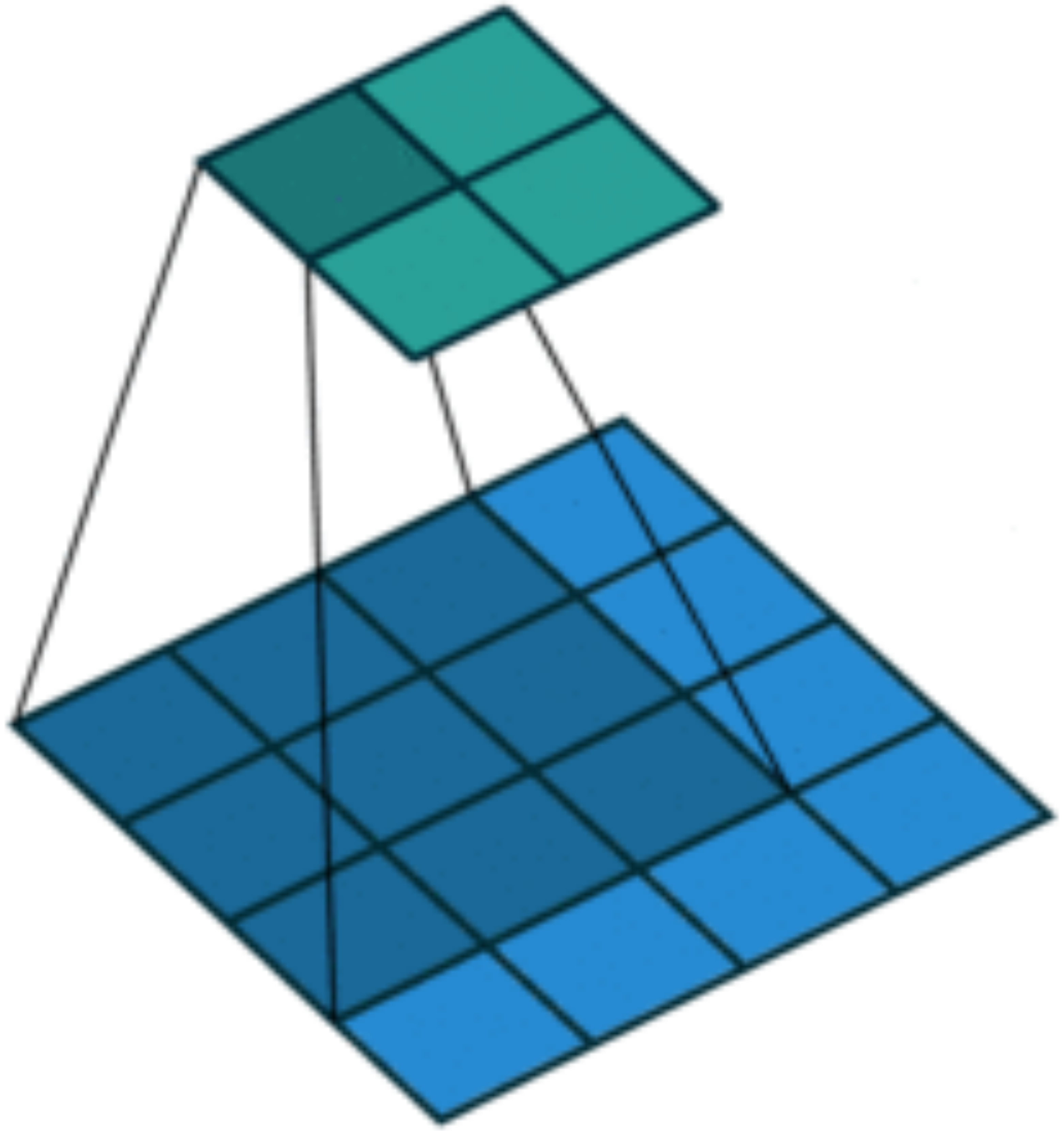
0	1
2	3

=

Output

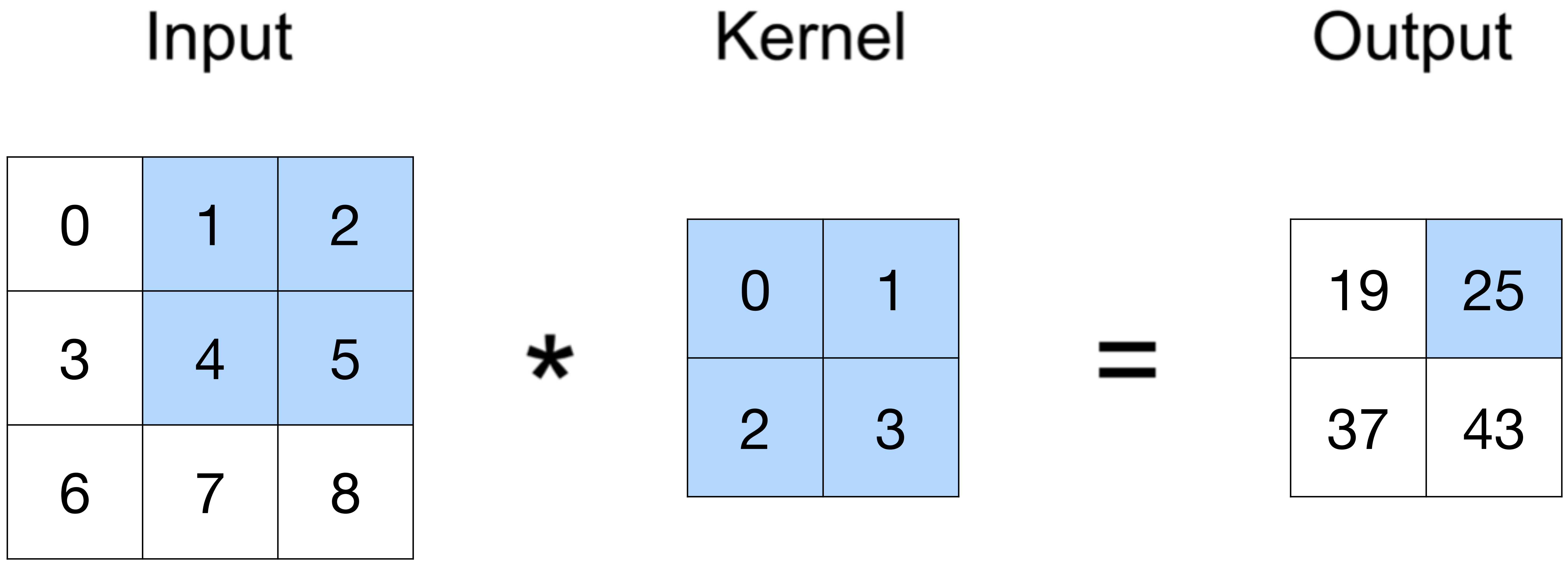
19	25
37	43

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

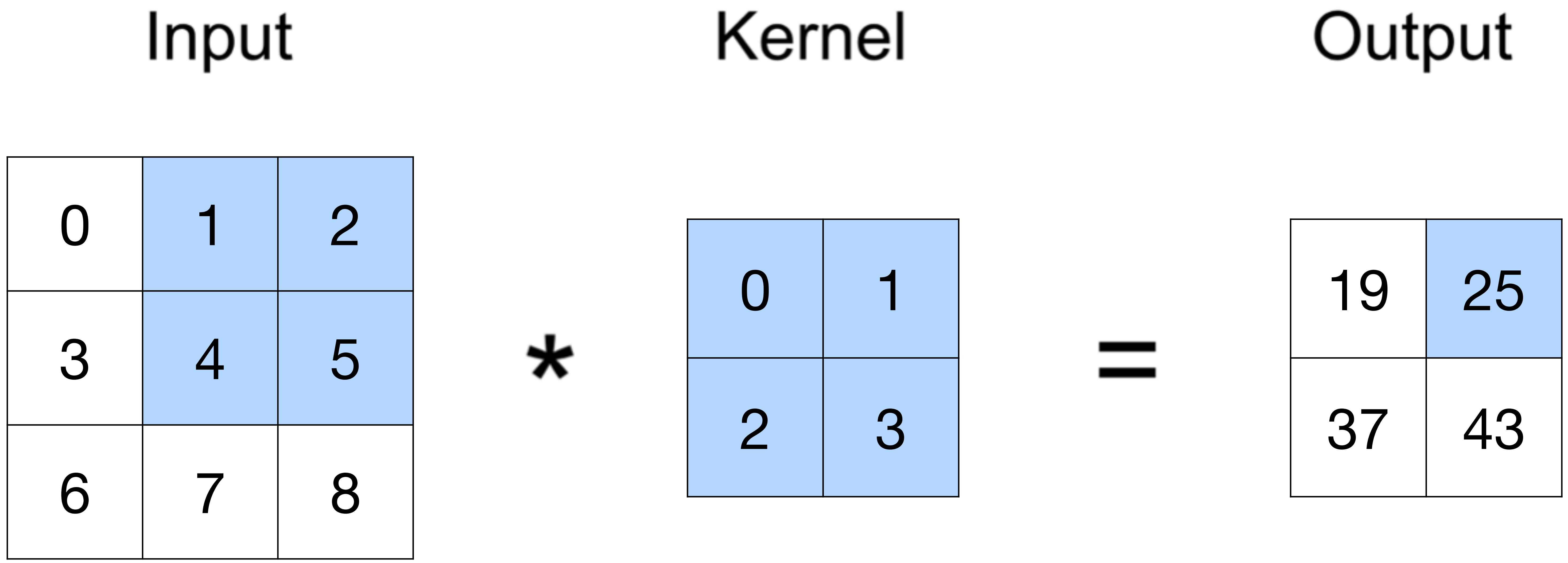


(vdumoulin@ Github)

2-D Convolution

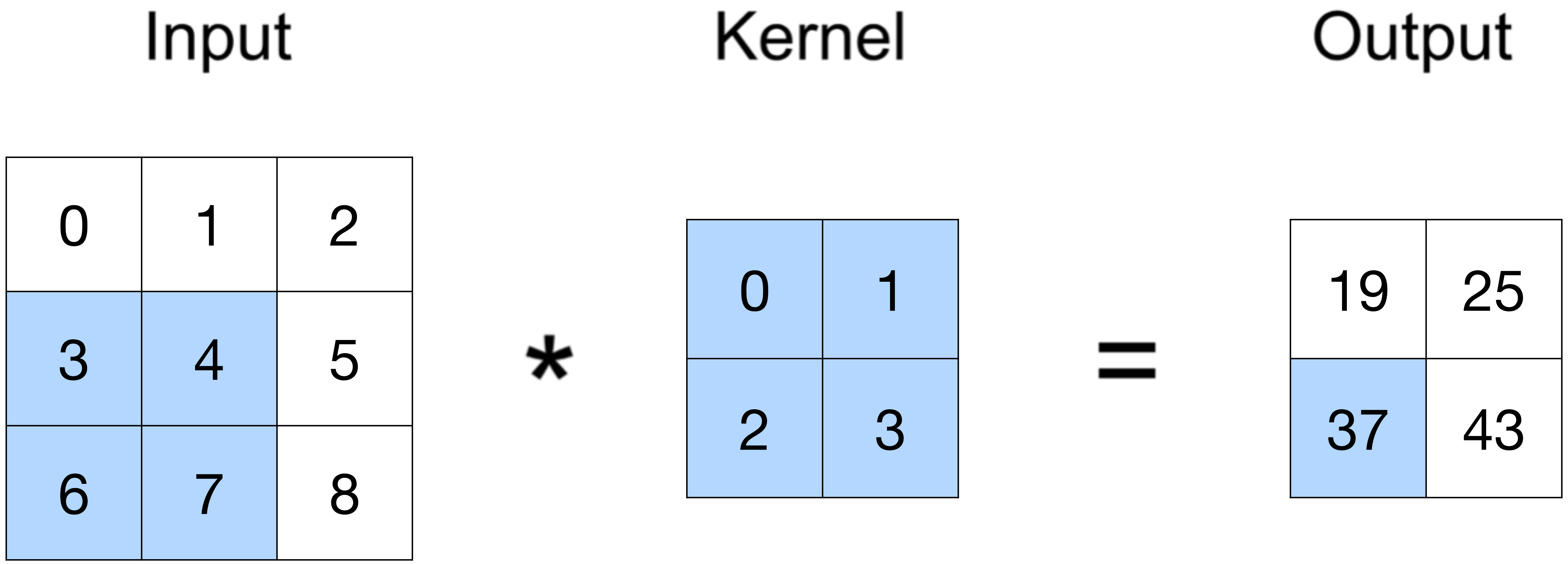


2-D Convolution

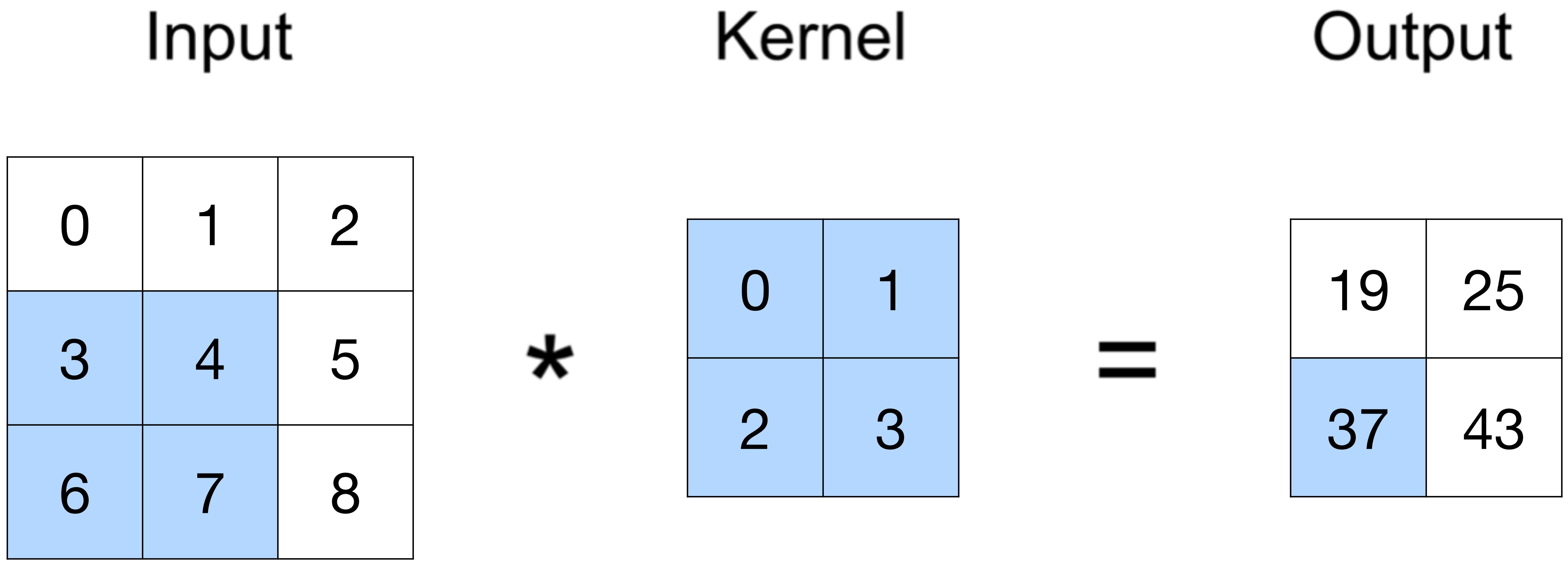


$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25$$

2-D Convolution

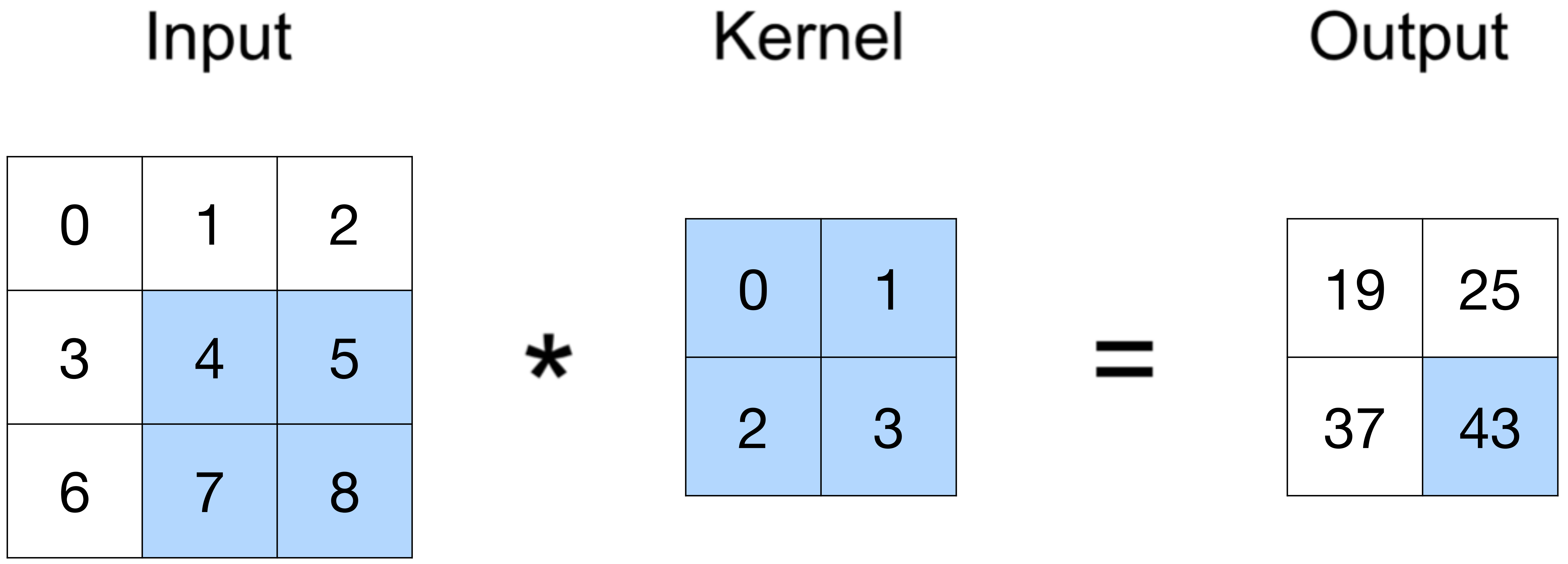


2-D Convolution

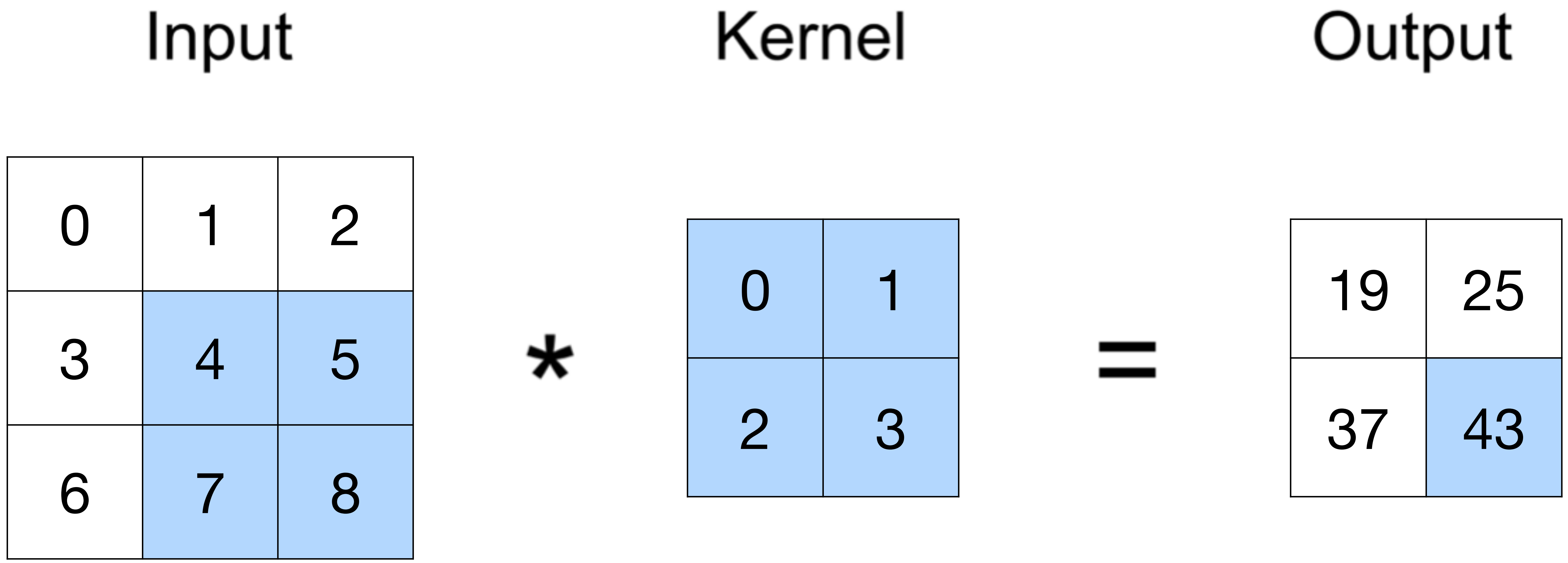


$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37$$

2-D Convolution

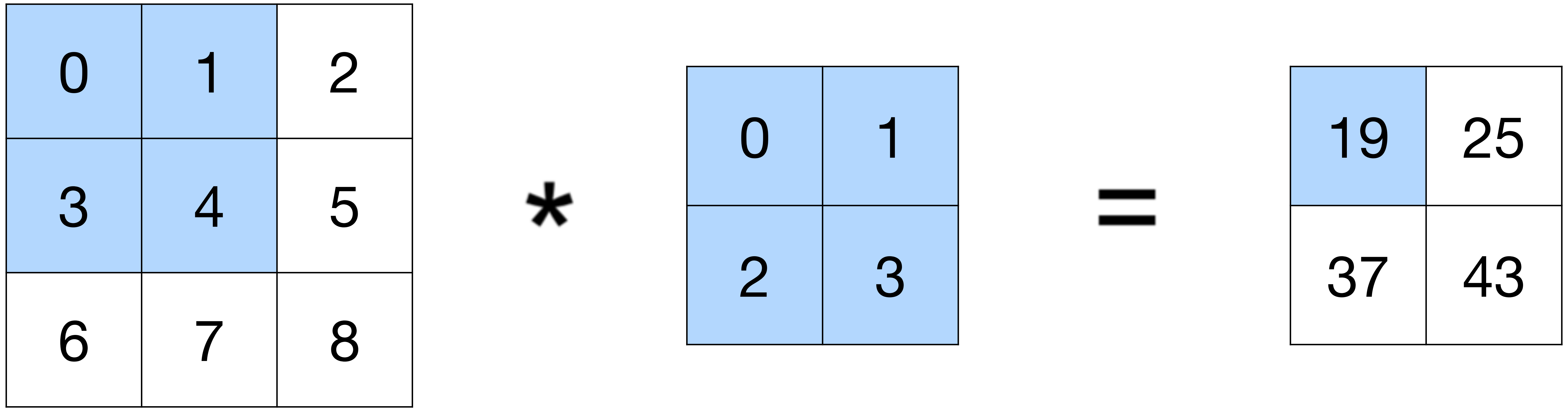


2-D Convolution



$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43$$

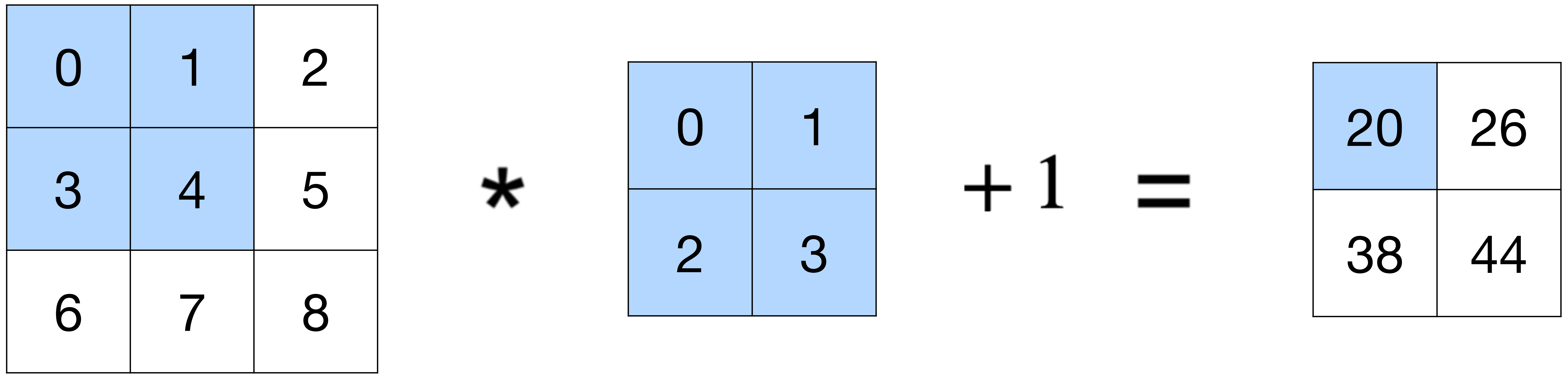
2-D Convolution Layer



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

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

2-D Convolution Layer

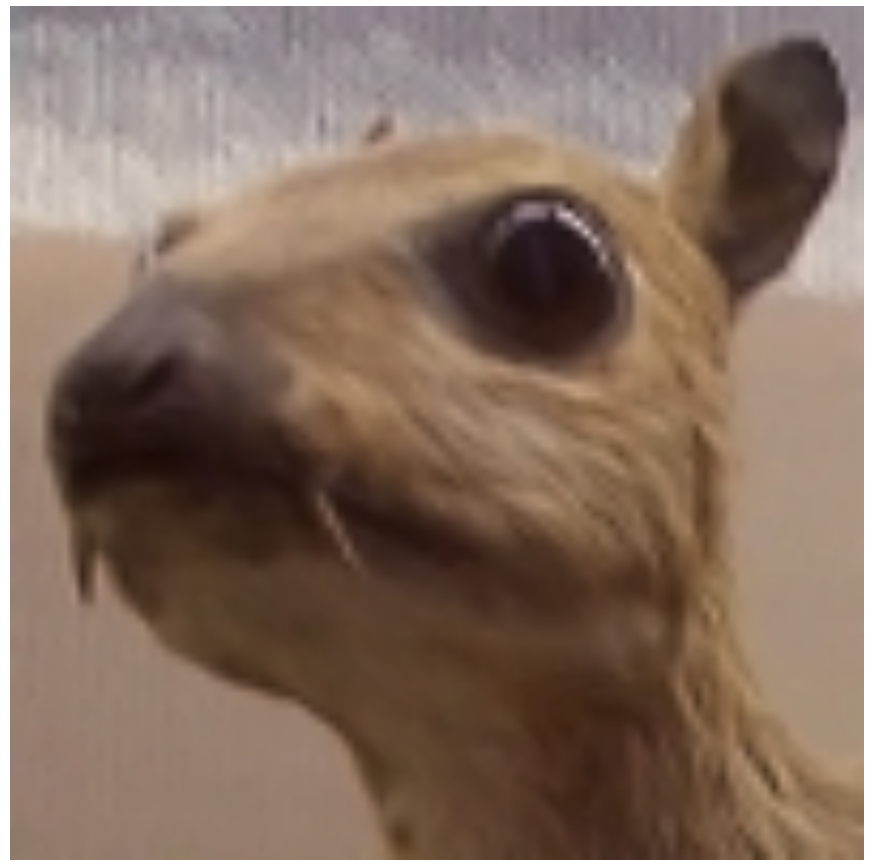


- $\mathbf{X} : n_h \times n_w$ input matrix
- $\mathbf{W} : k_h \times k_w$ kernel matrix
- b: scalar bias
- $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

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

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

Examples

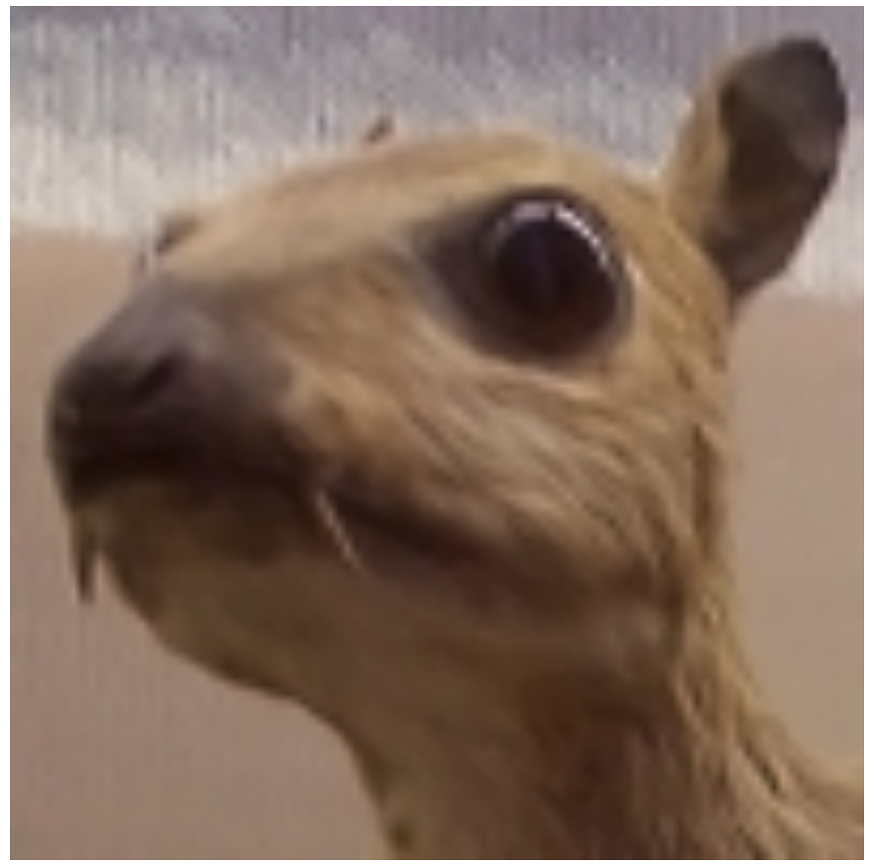


(wikipedia)

Examples

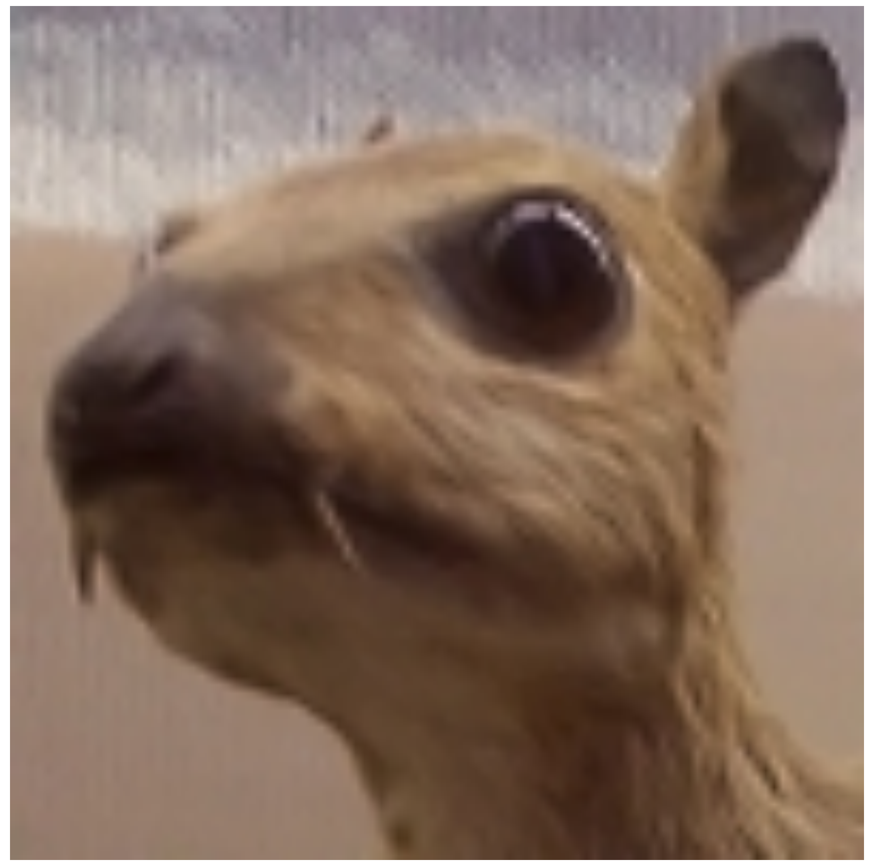
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection
(Viewed black and white)



(wikipedia)

Examples

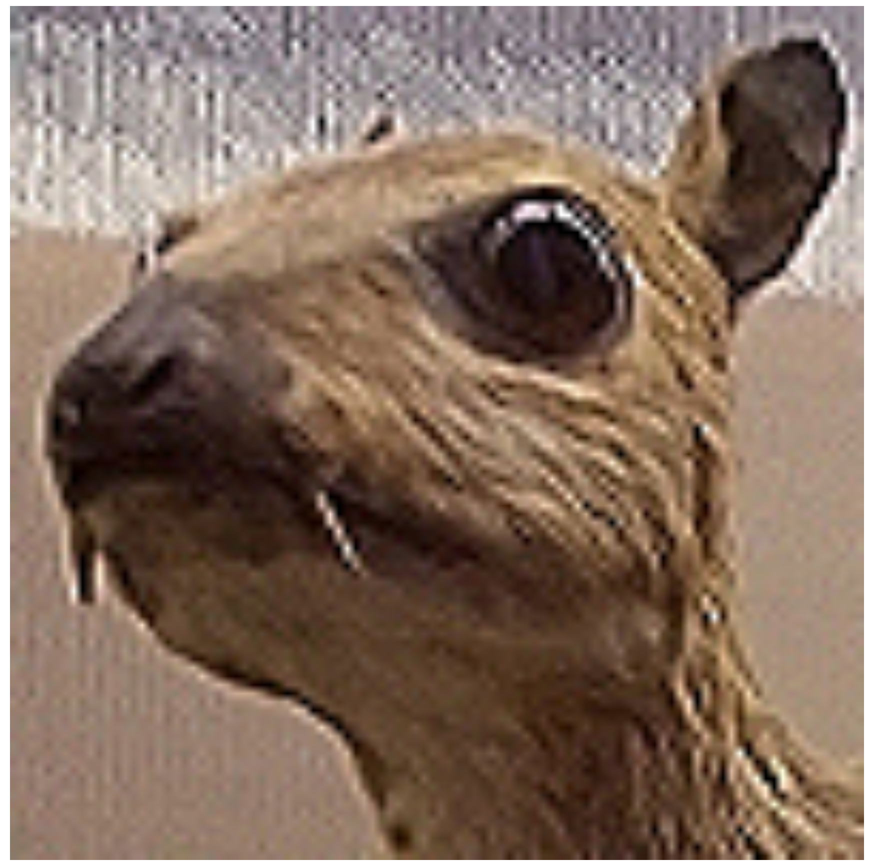


(wikipedia)

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

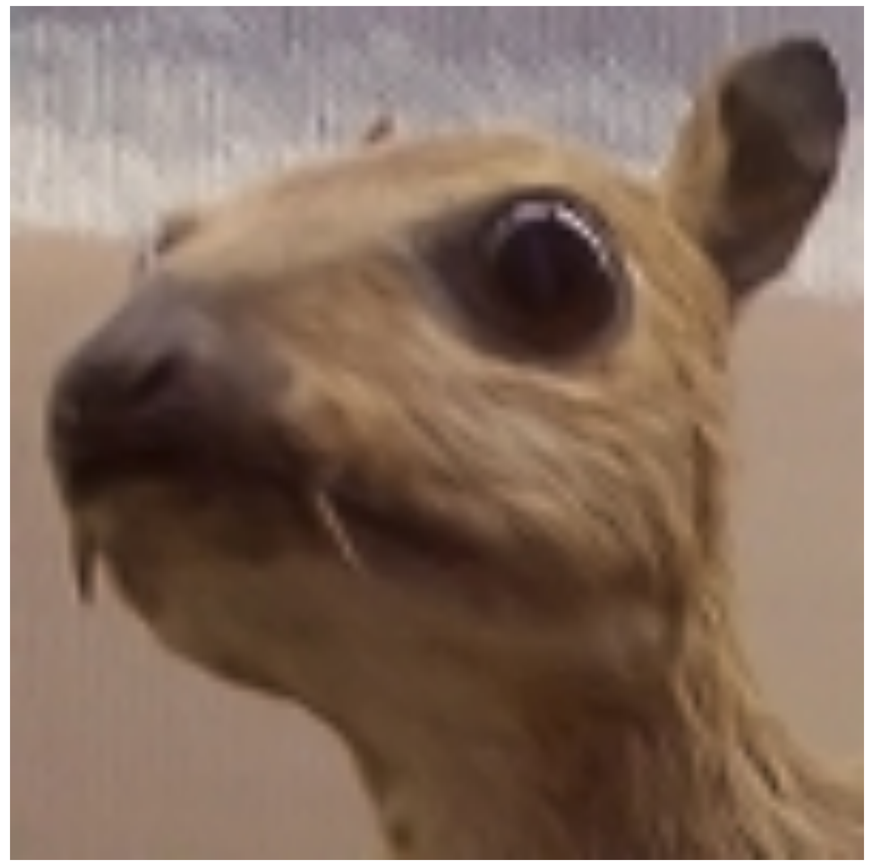
Edge Detection
(Viewed black and white)

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



Sharpen

Examples

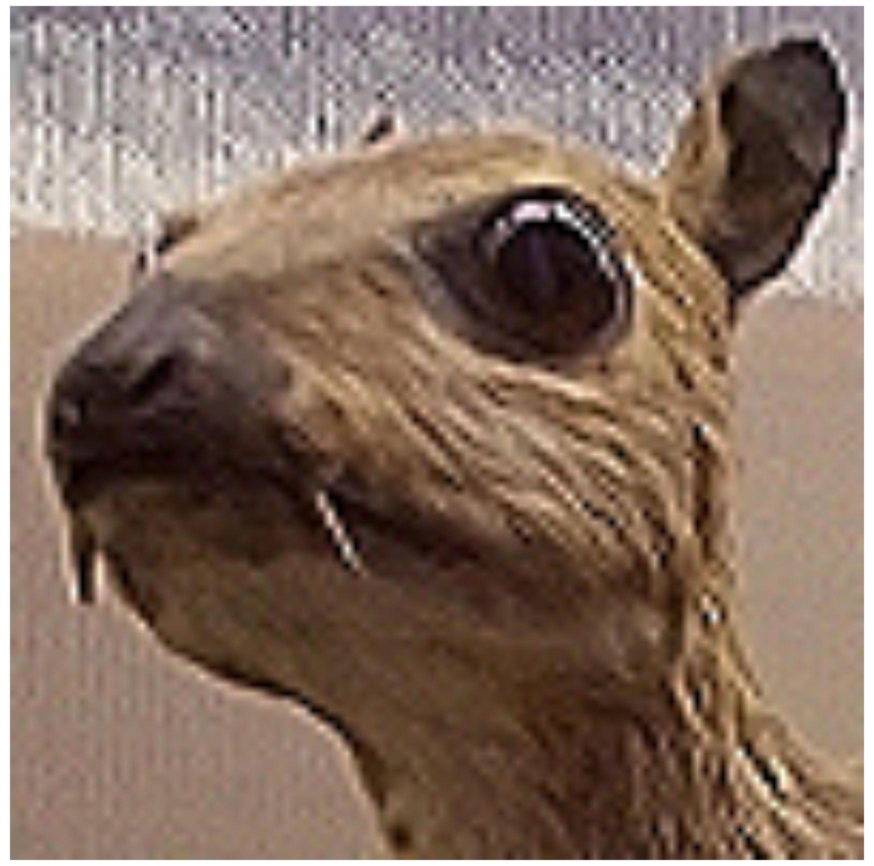


(wikipedia)

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

Edge Detection
(Viewed black and white)

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



Sharpen

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

Gaussian Blur

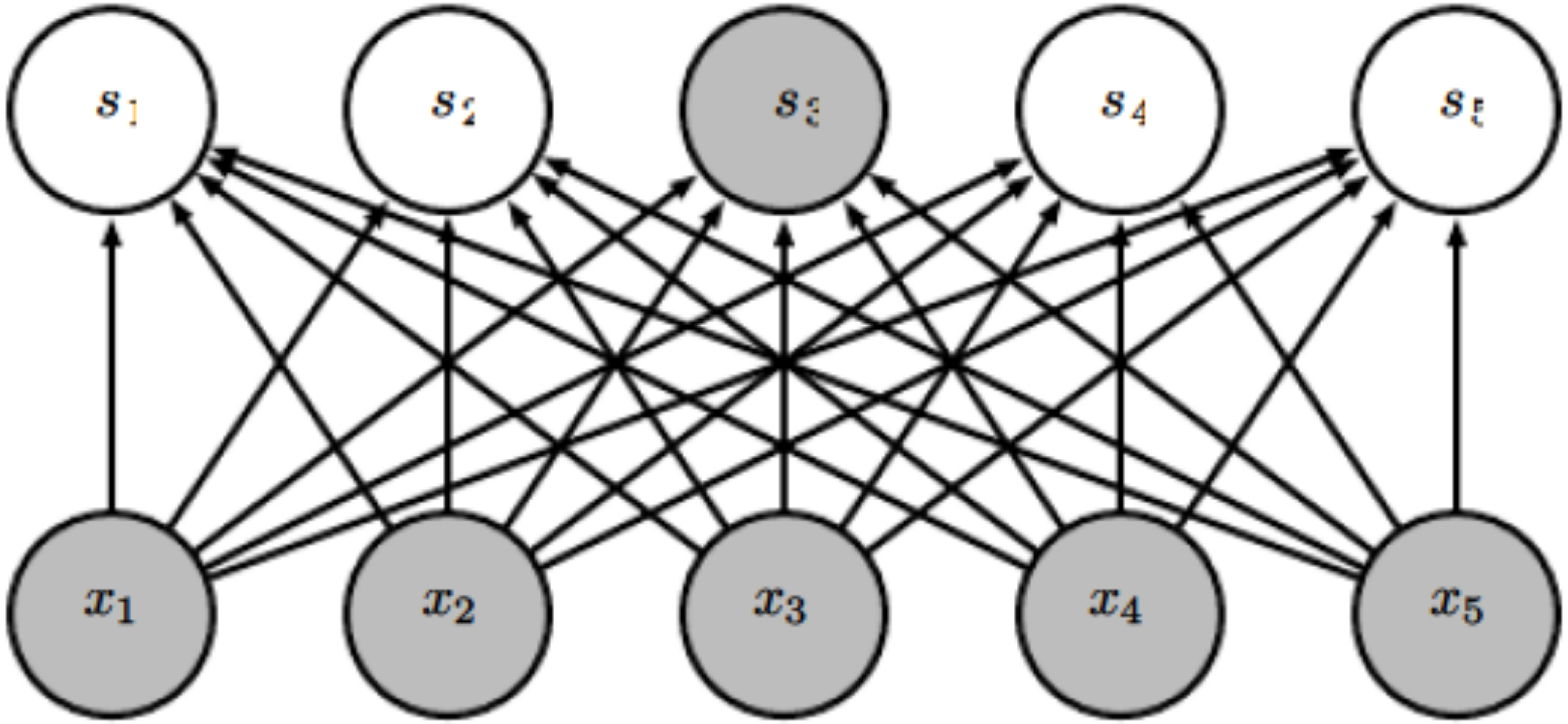
Convolutional Neural Networks

- 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



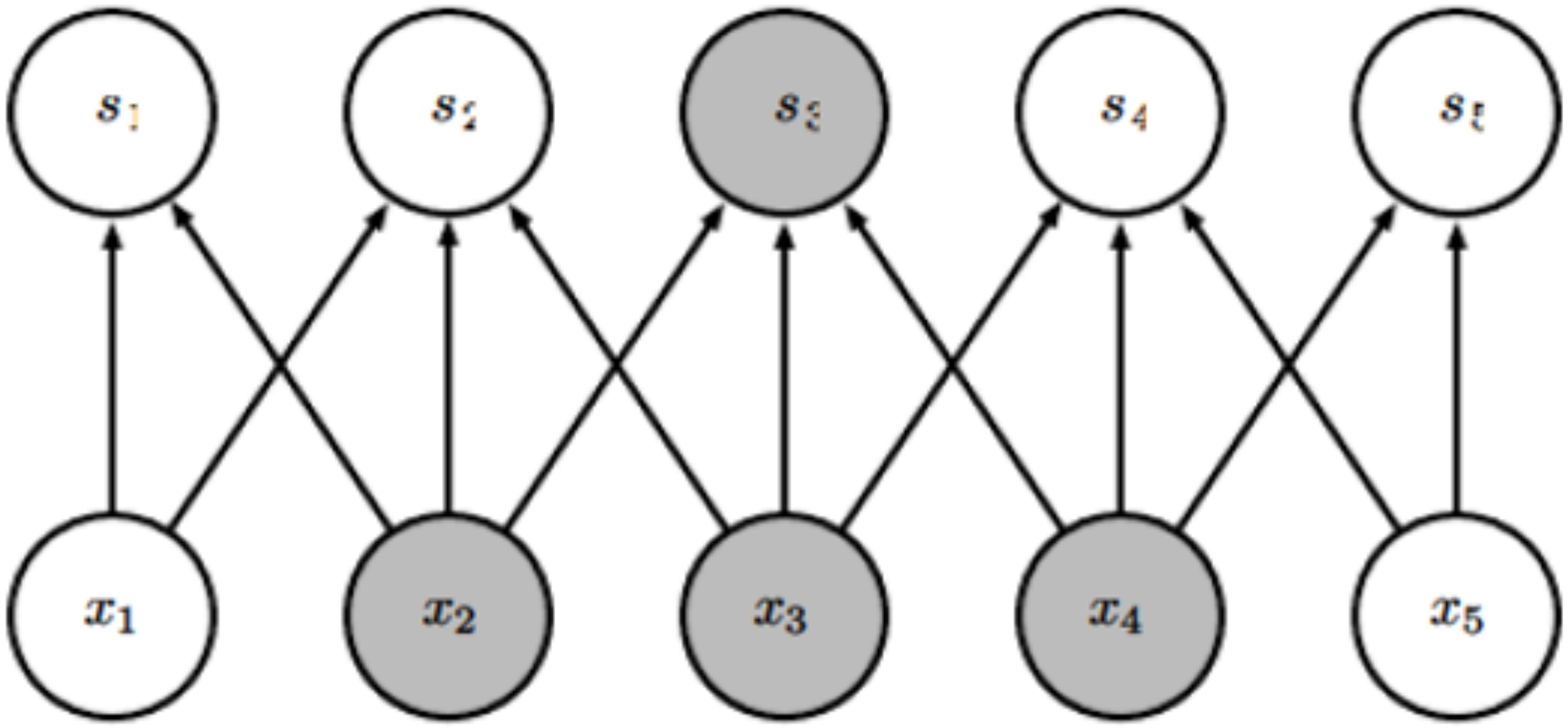
m output nodes

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

Efficiency of Convolution

Efficiency of Convolution

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

Efficiency of Convolution

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

	Convolution	Dense matrix
Stored floats		
Float muls or adds		

Efficiency of Convolution

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

	Convolution	Dense matrix
Stored floats	2	
Float muls or adds		

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		

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	

Efficiency of Convolution

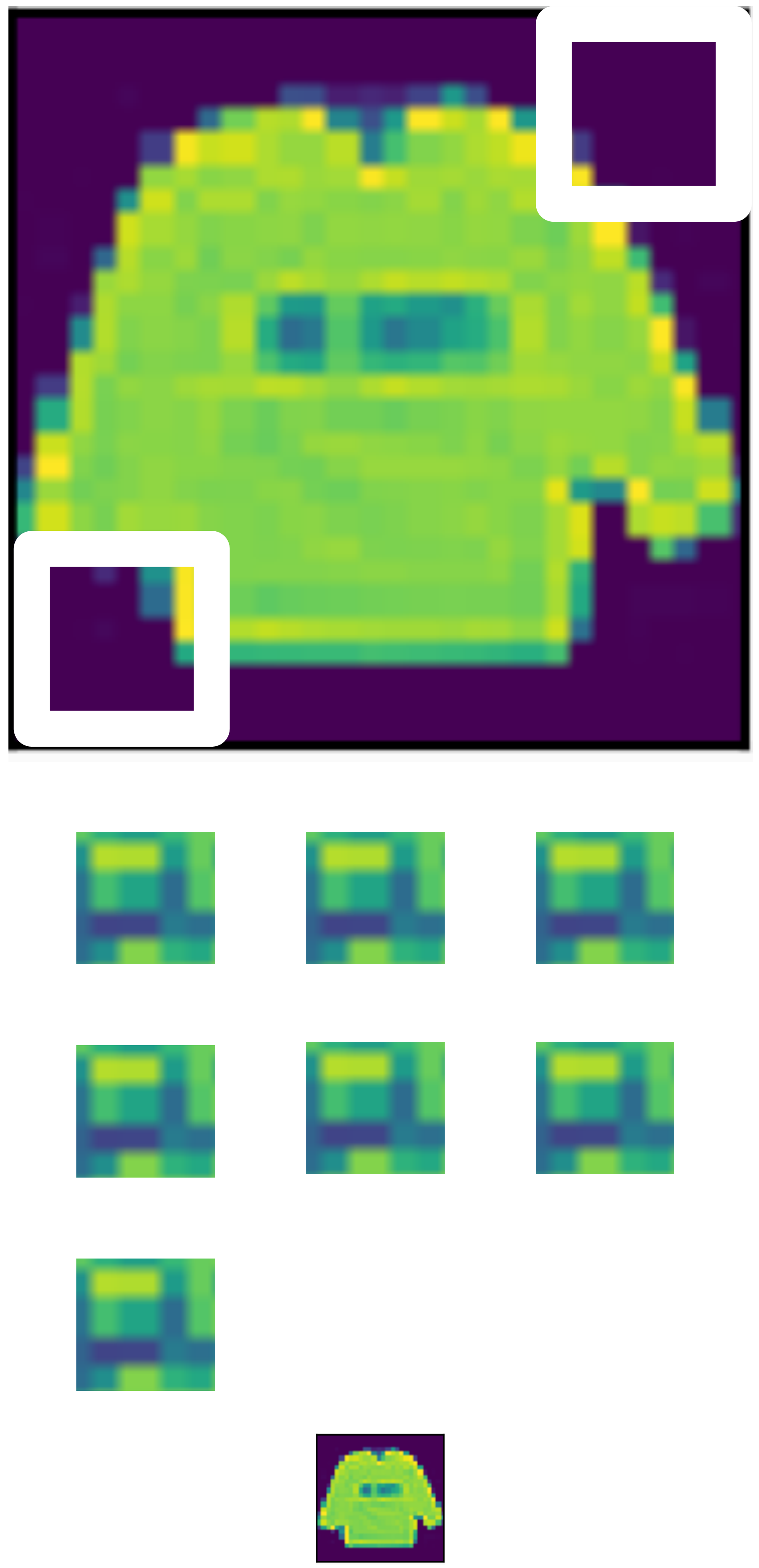
- Input size: 320 x 280
- Kernel Size: 2 x 1
- Output size: 319 x 280

	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

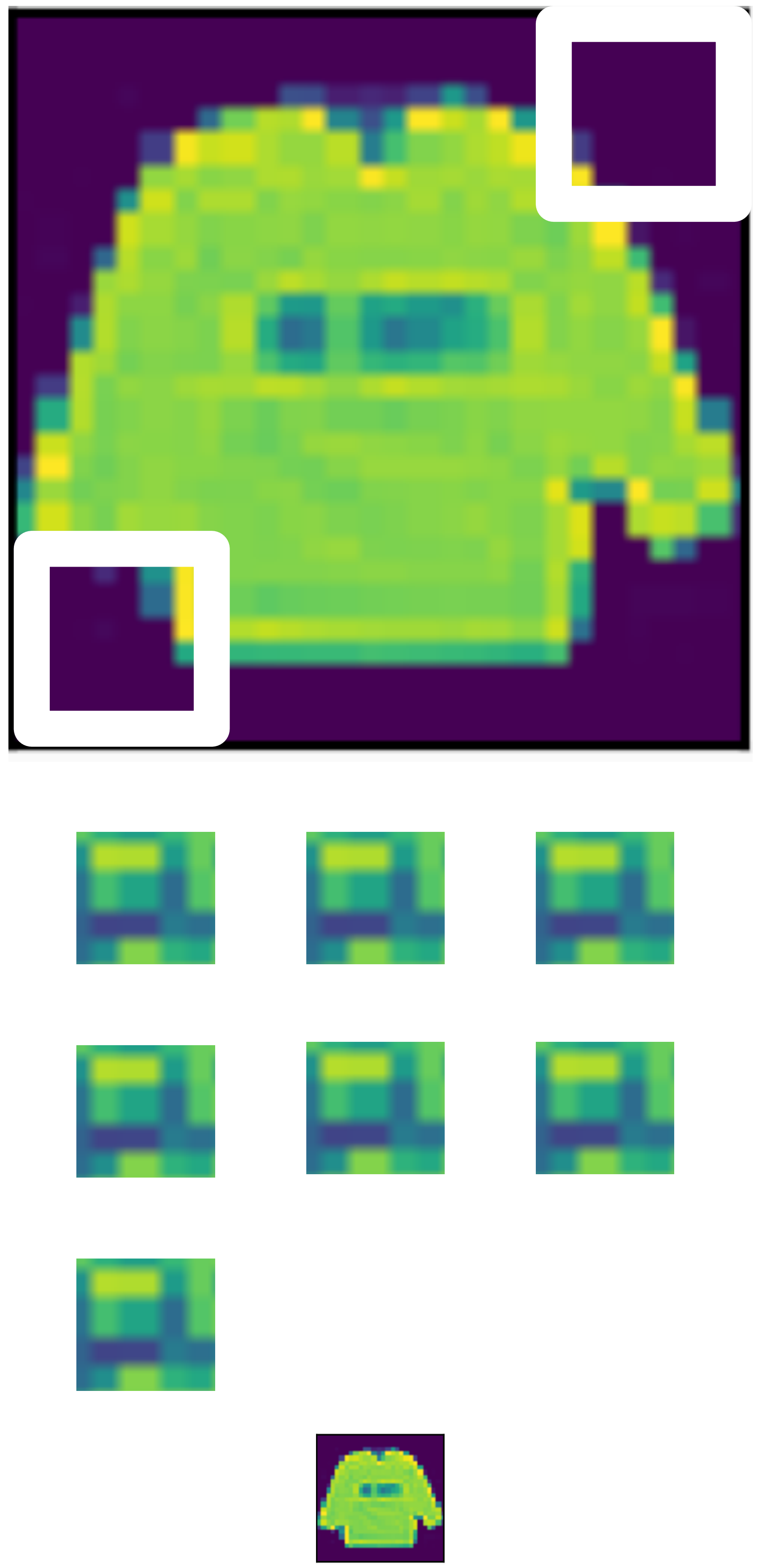
- Given a 32 x 32 input image
- Apply convolution with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers



Padding

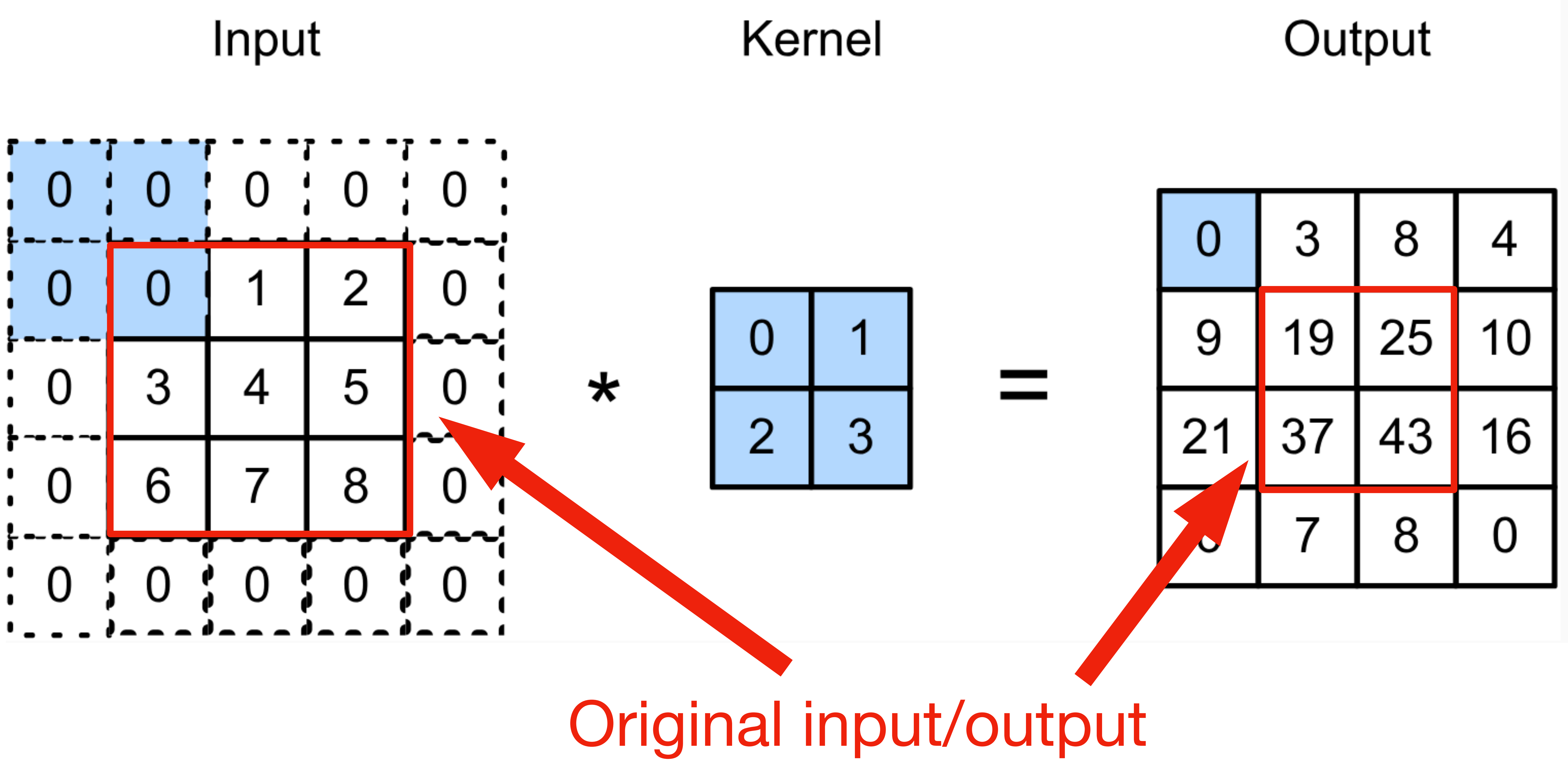
- Given a 32 x 32 input image
- Apply convolution with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to

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



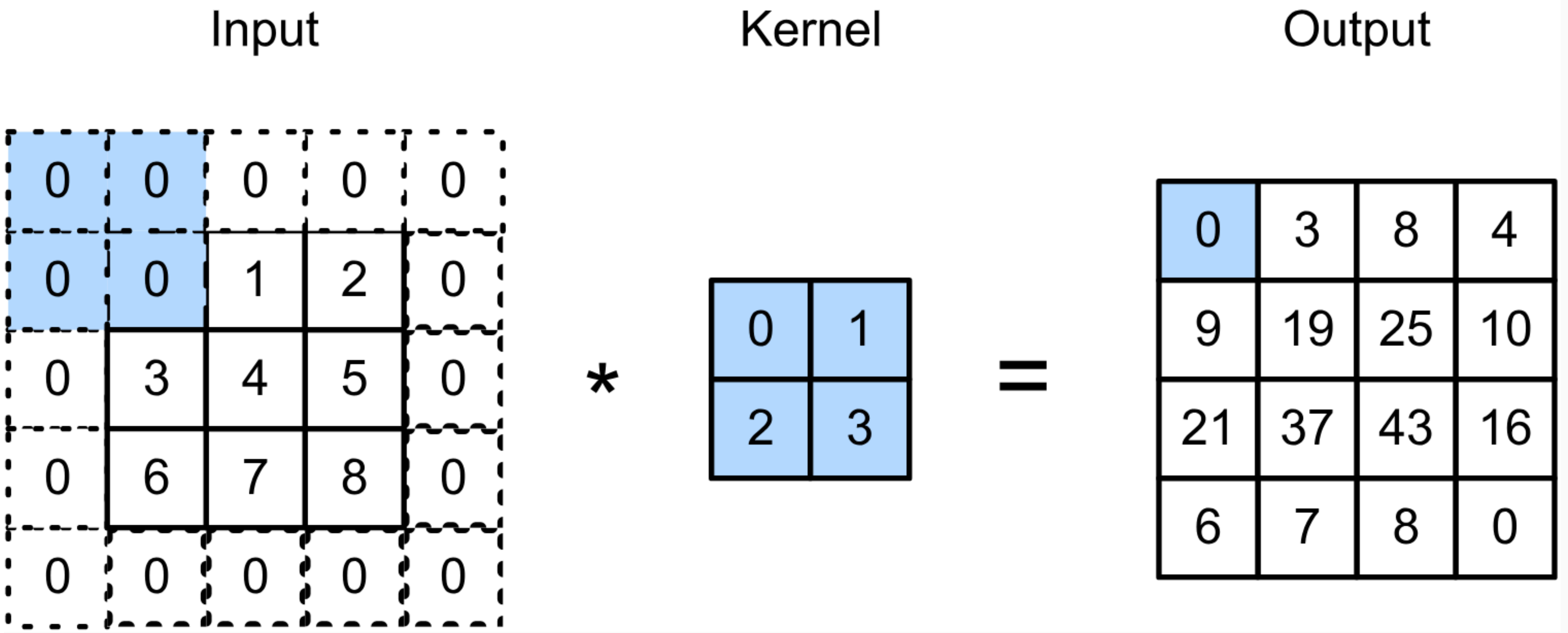
Padding

Padding adds rows/columns around input



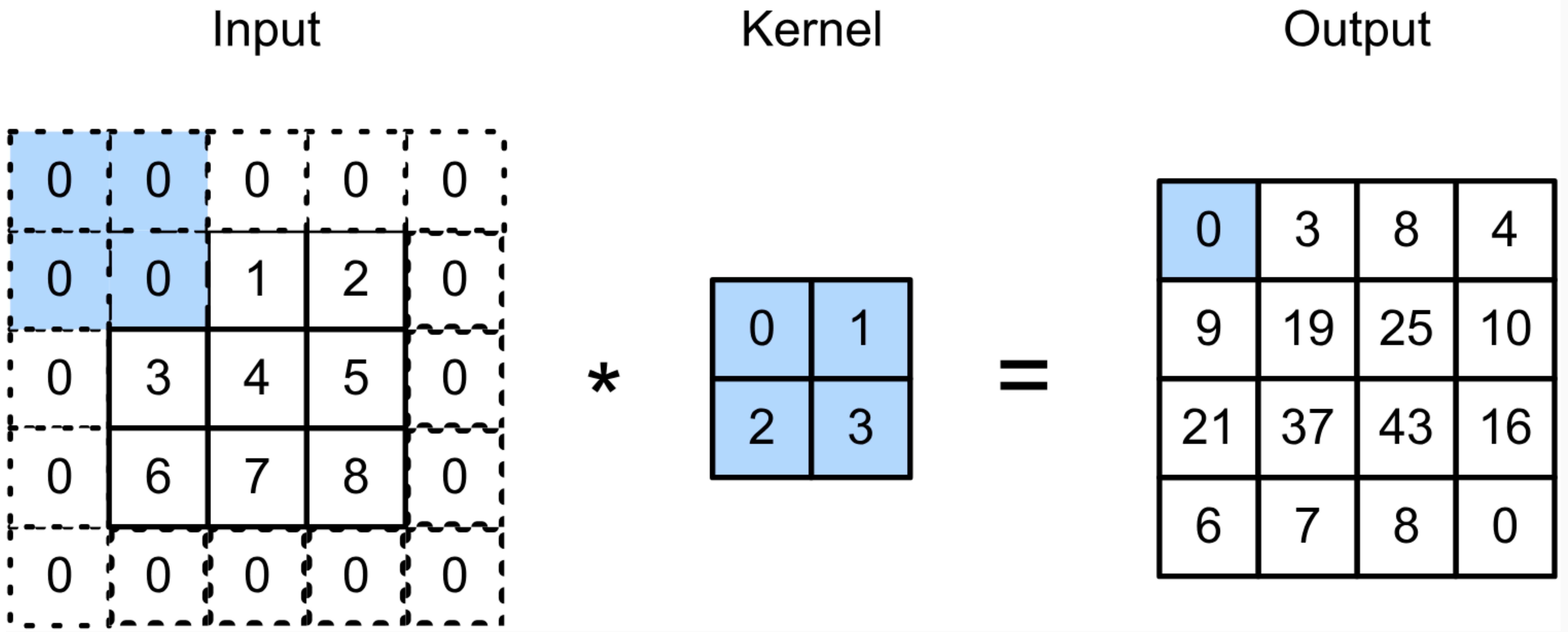
Padding

Padding adds rows/columns around input



Padding

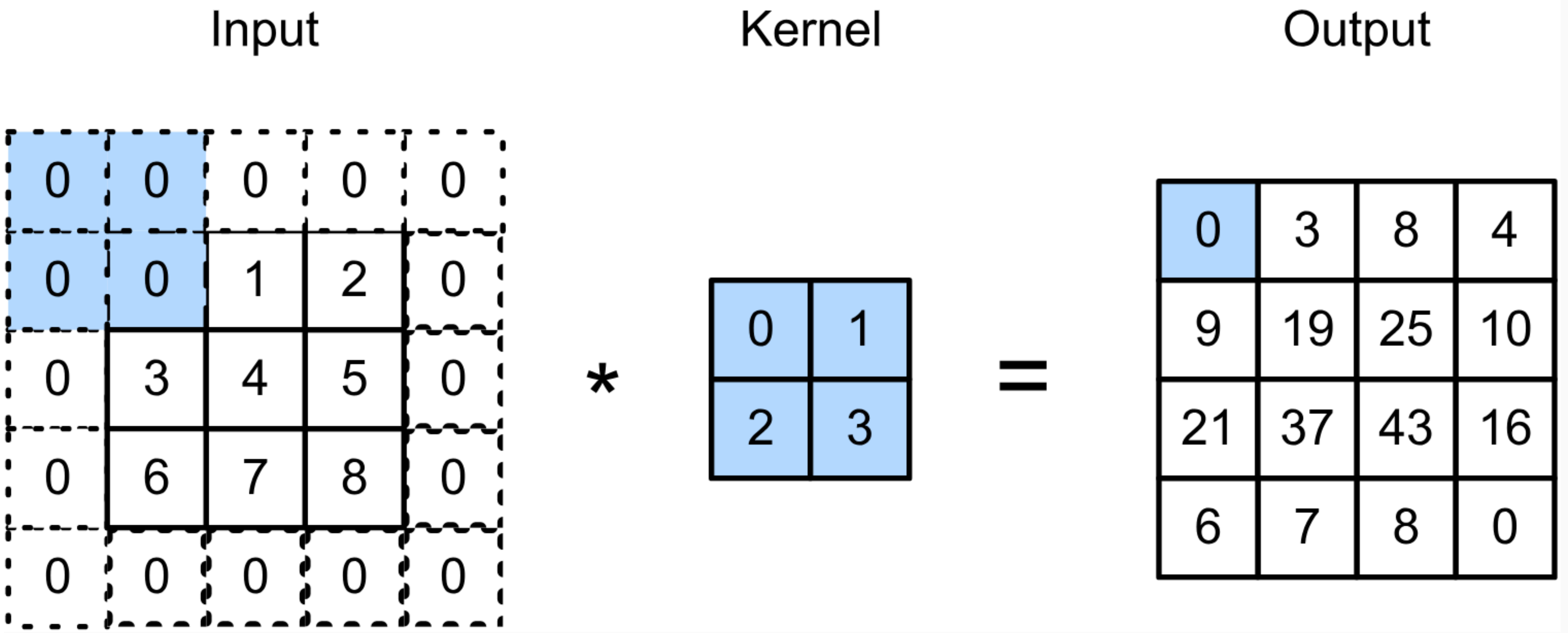
Padding adds rows/columns around input

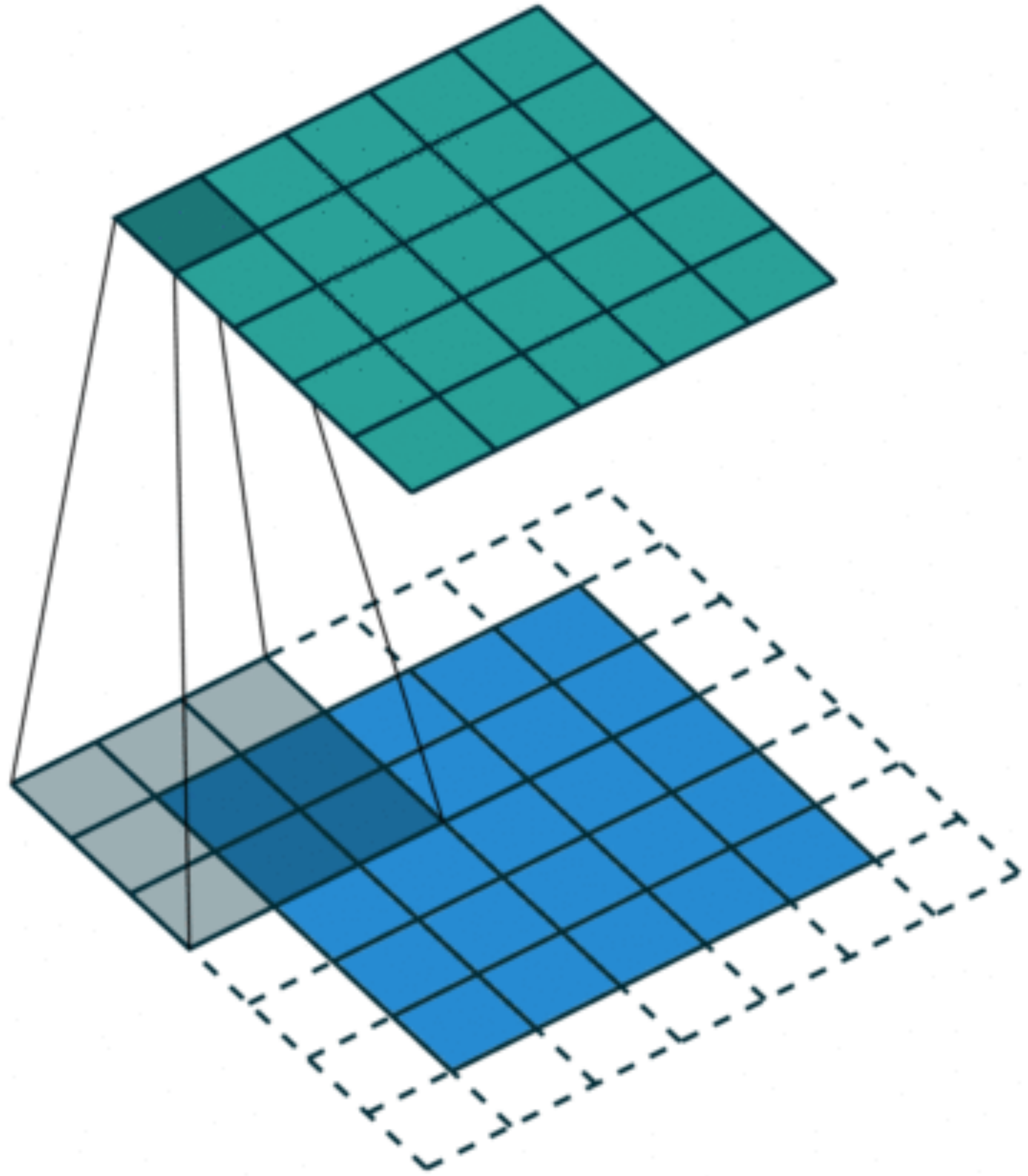


$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Padding

Padding adds rows/columns around input

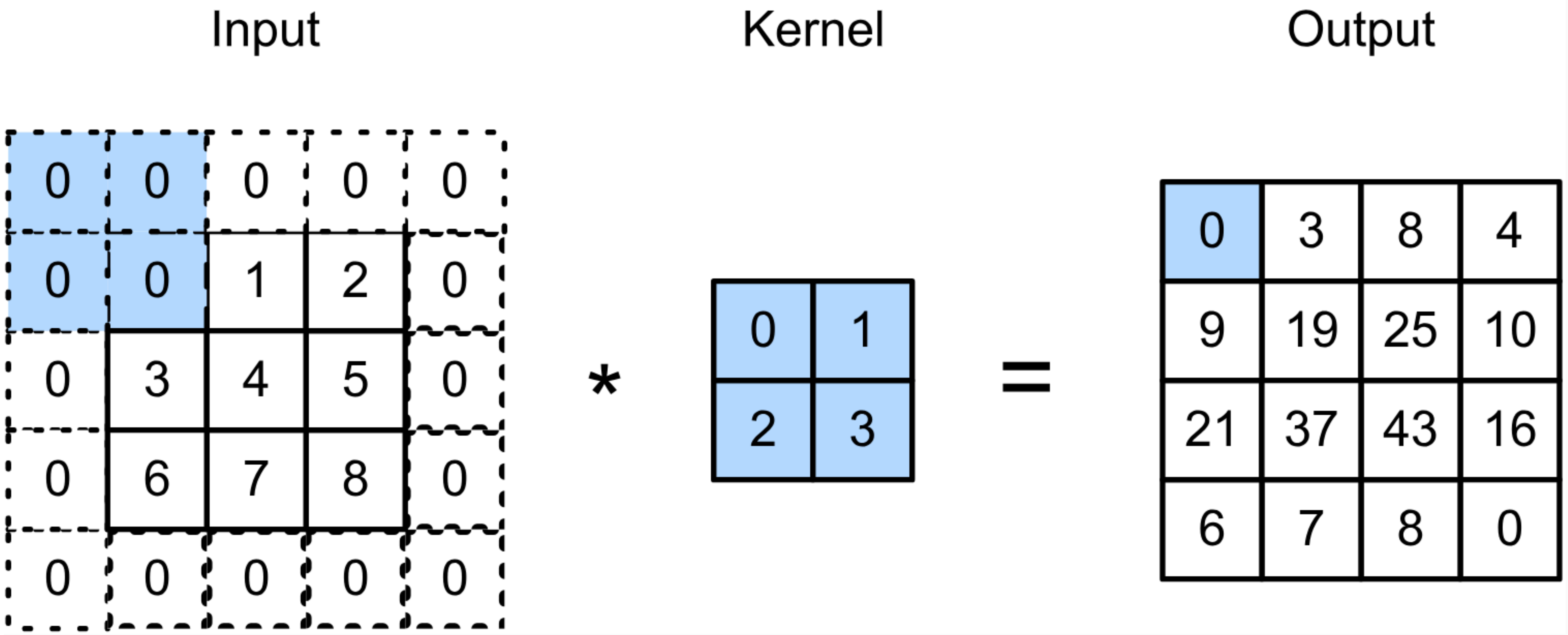


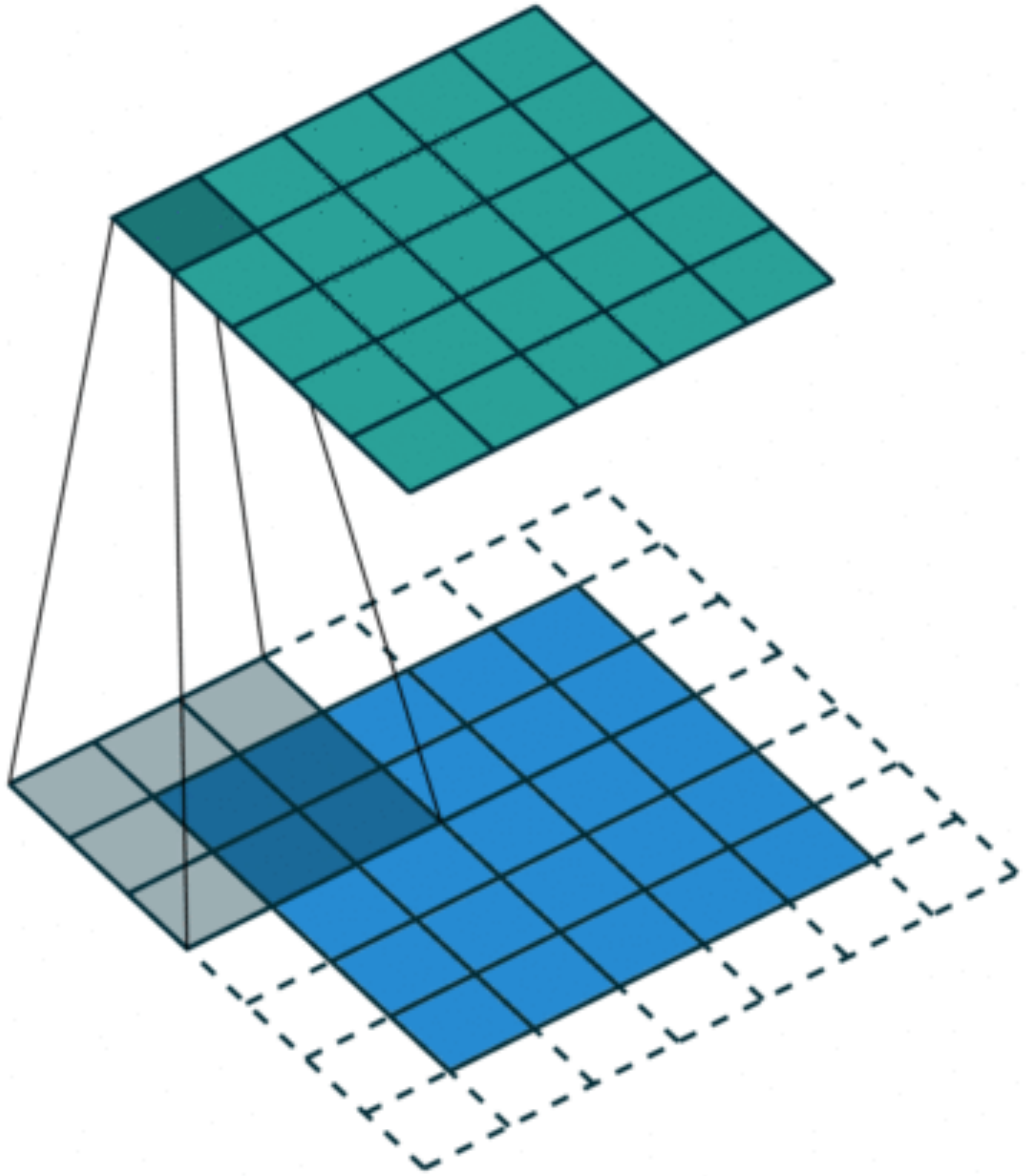


$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Padding

Padding adds rows/columns around input





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

Padding

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

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

Padding

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

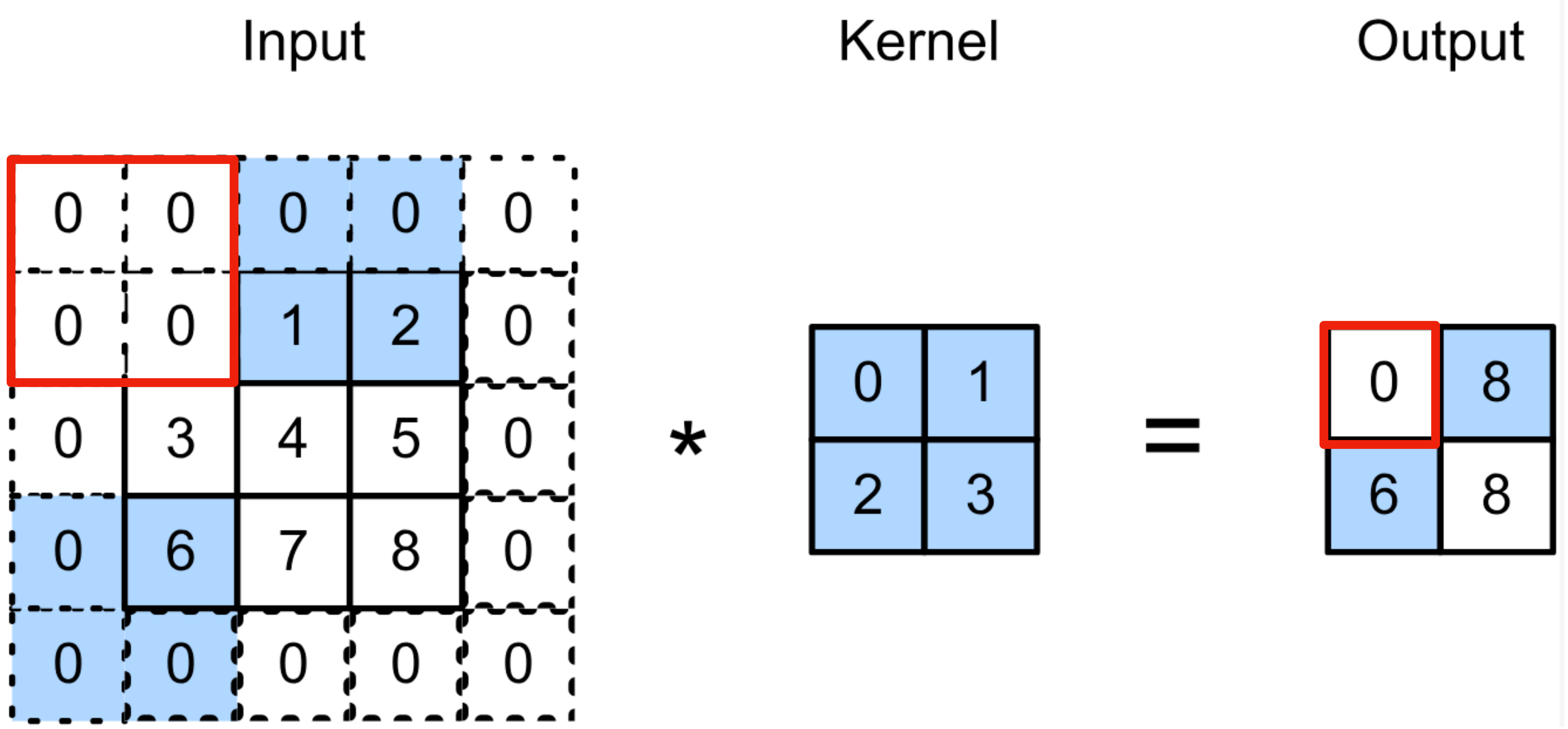
$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- 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

- Stride is the #rows/#columns per slide

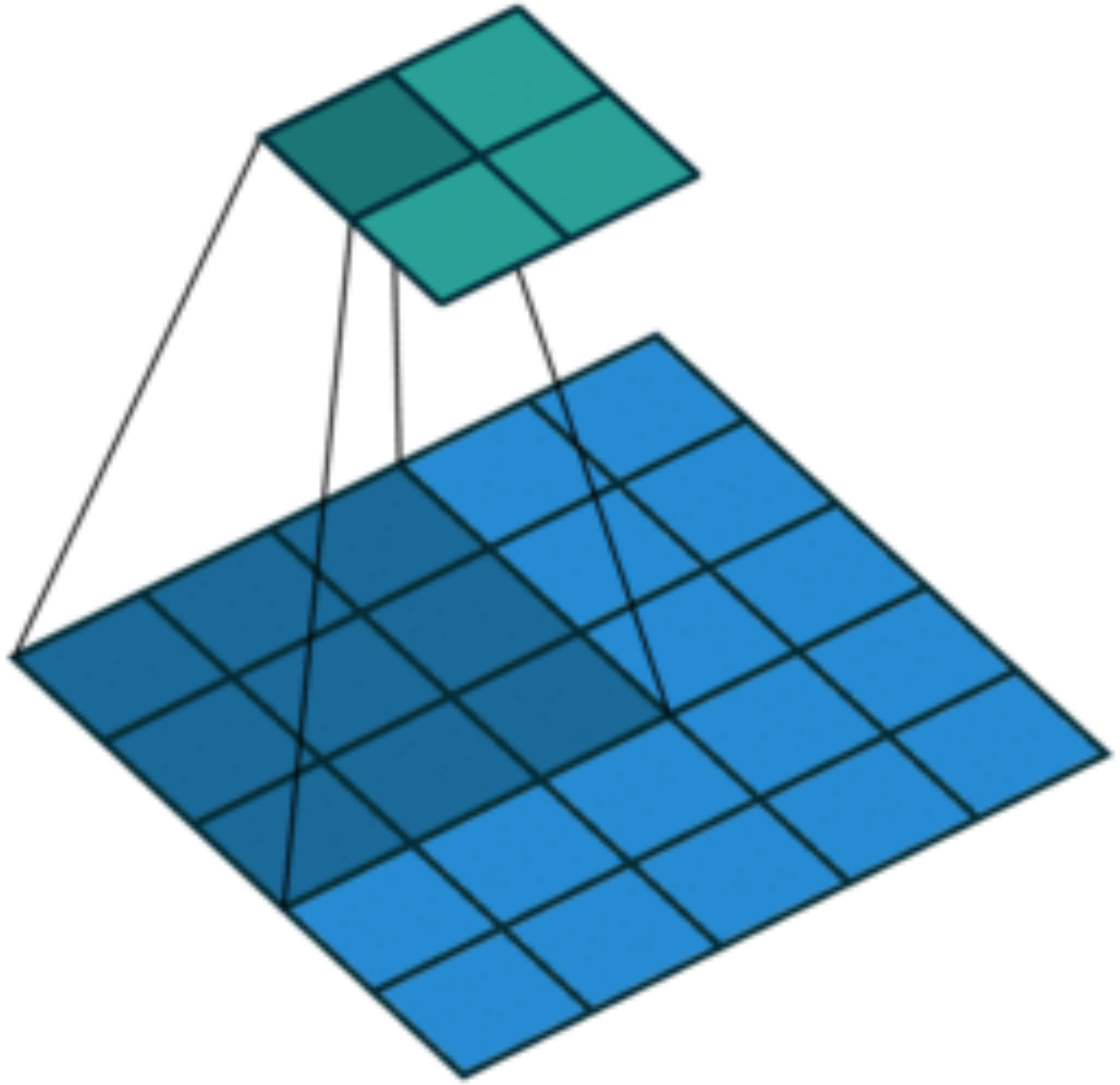
Strides of 3 and 2 for height and width



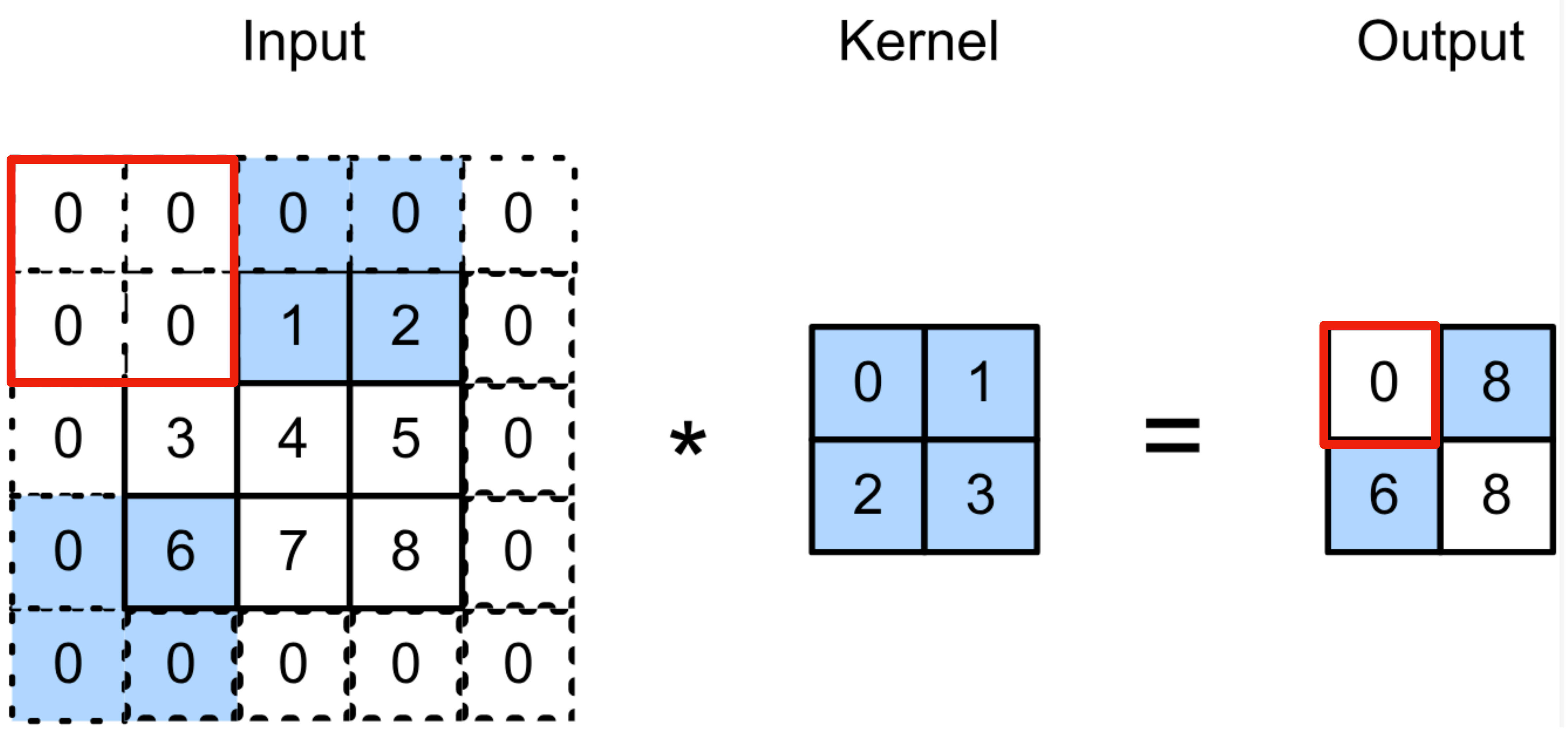
$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

Stride

- Stride is the #rows/#columns per slide



Strides of 3 and 2 for height and width

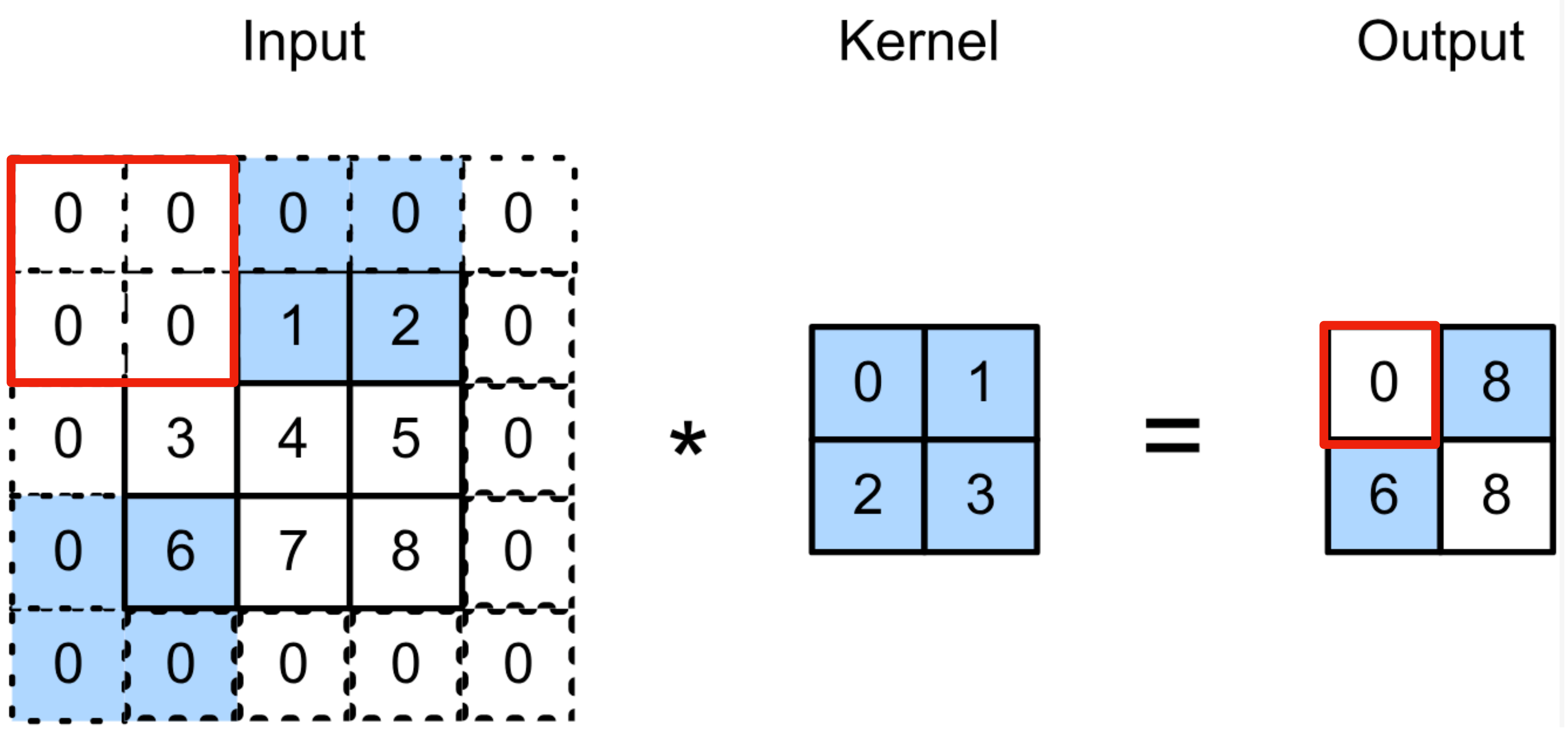


$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

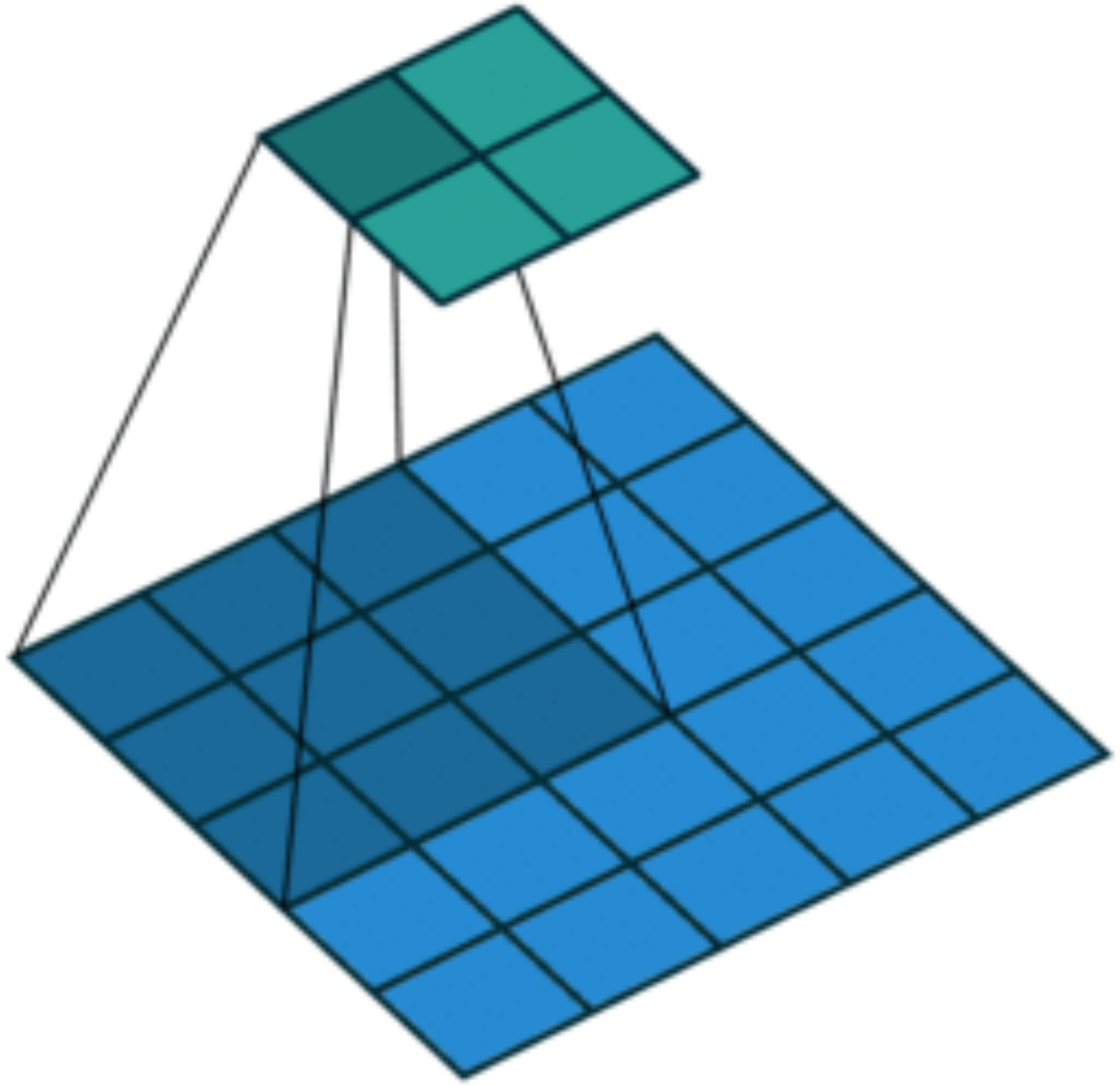
Stride

- 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

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

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

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

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

Multiple Input and Output Channels

Multiple Input Channels

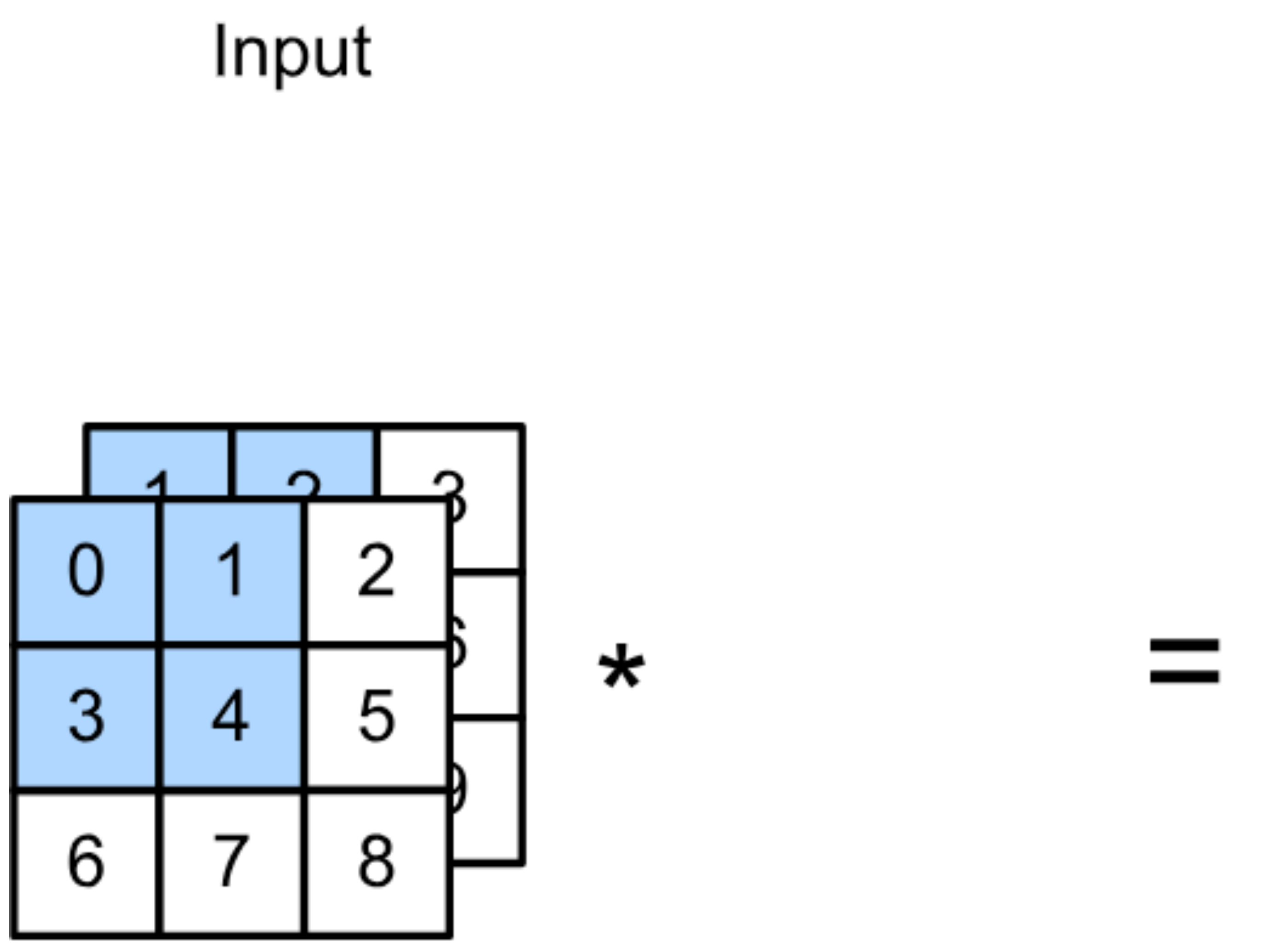
- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

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

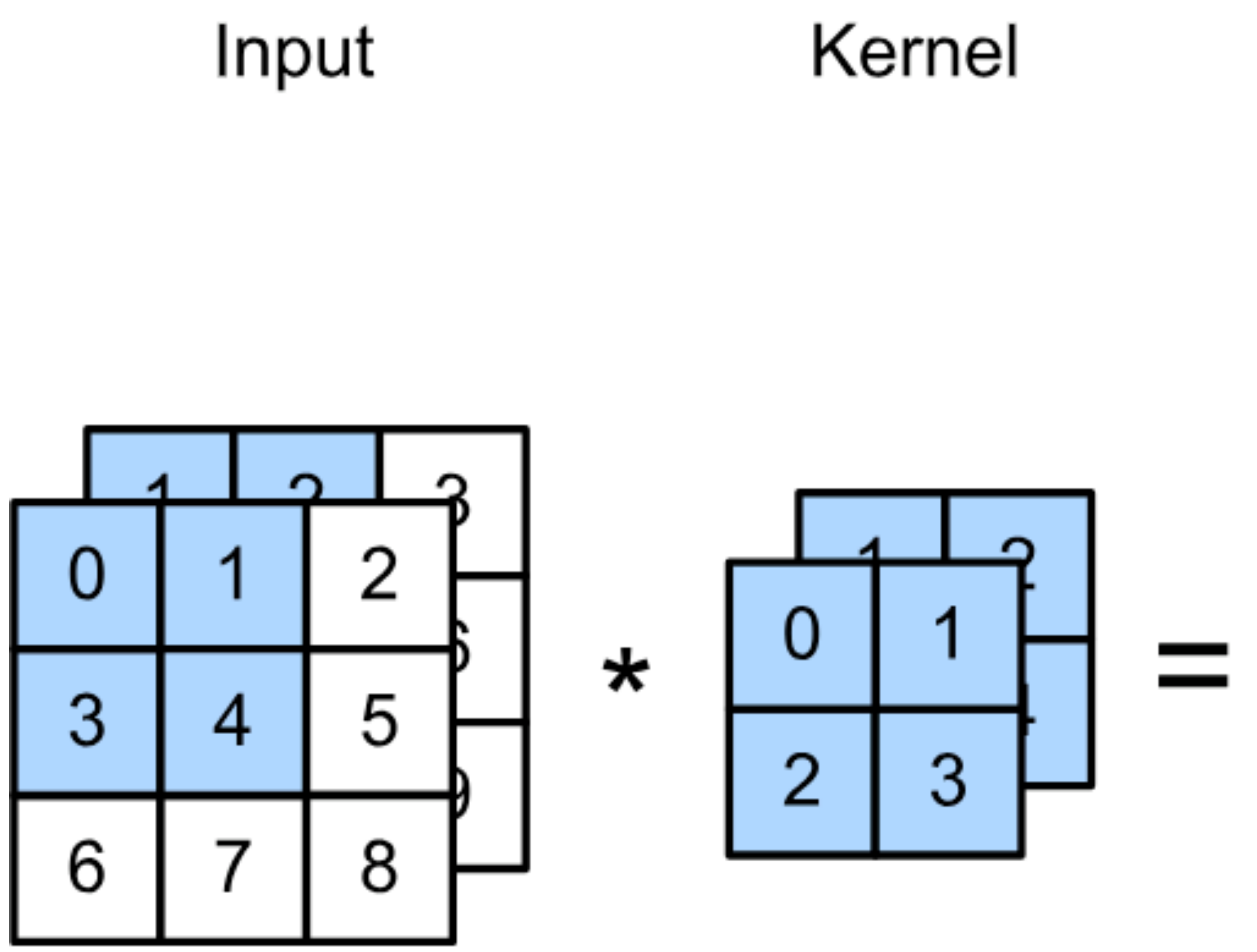
Multiple Input Channels

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



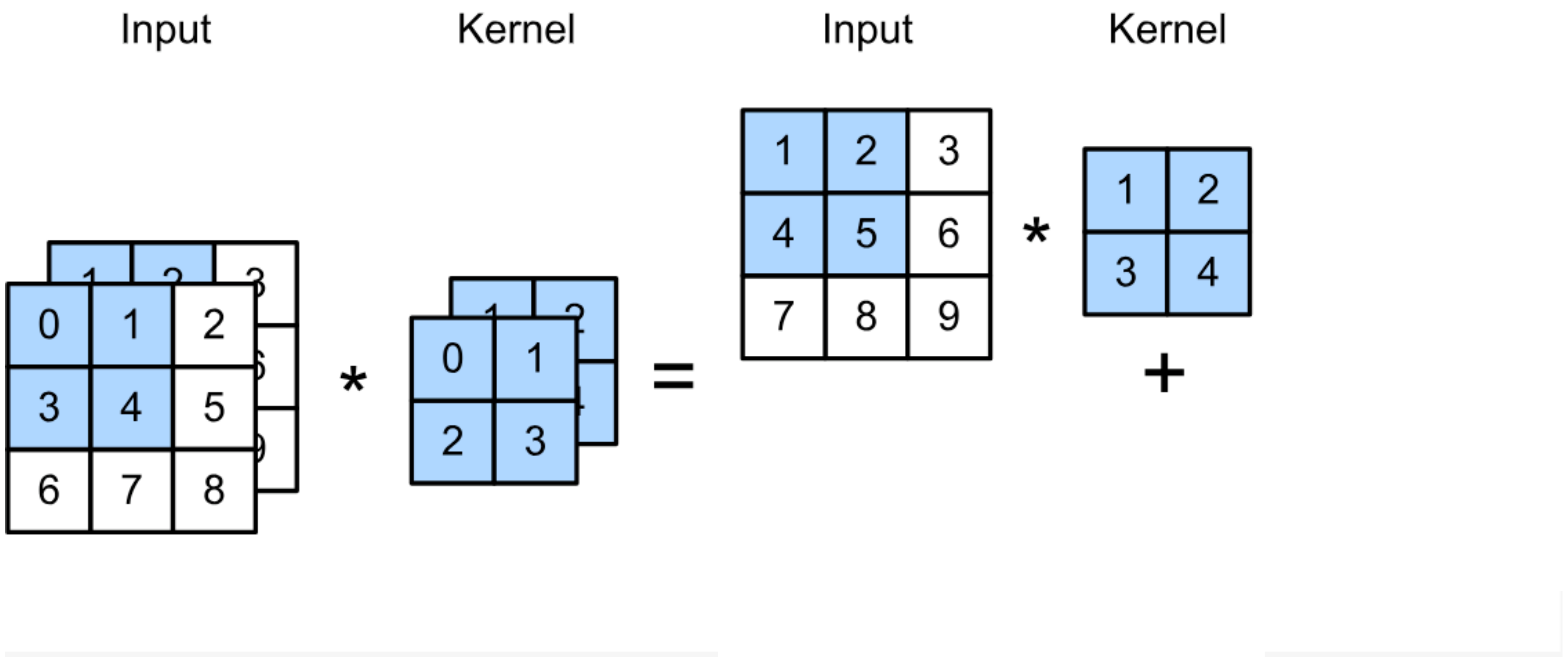
Multiple Input Channels

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



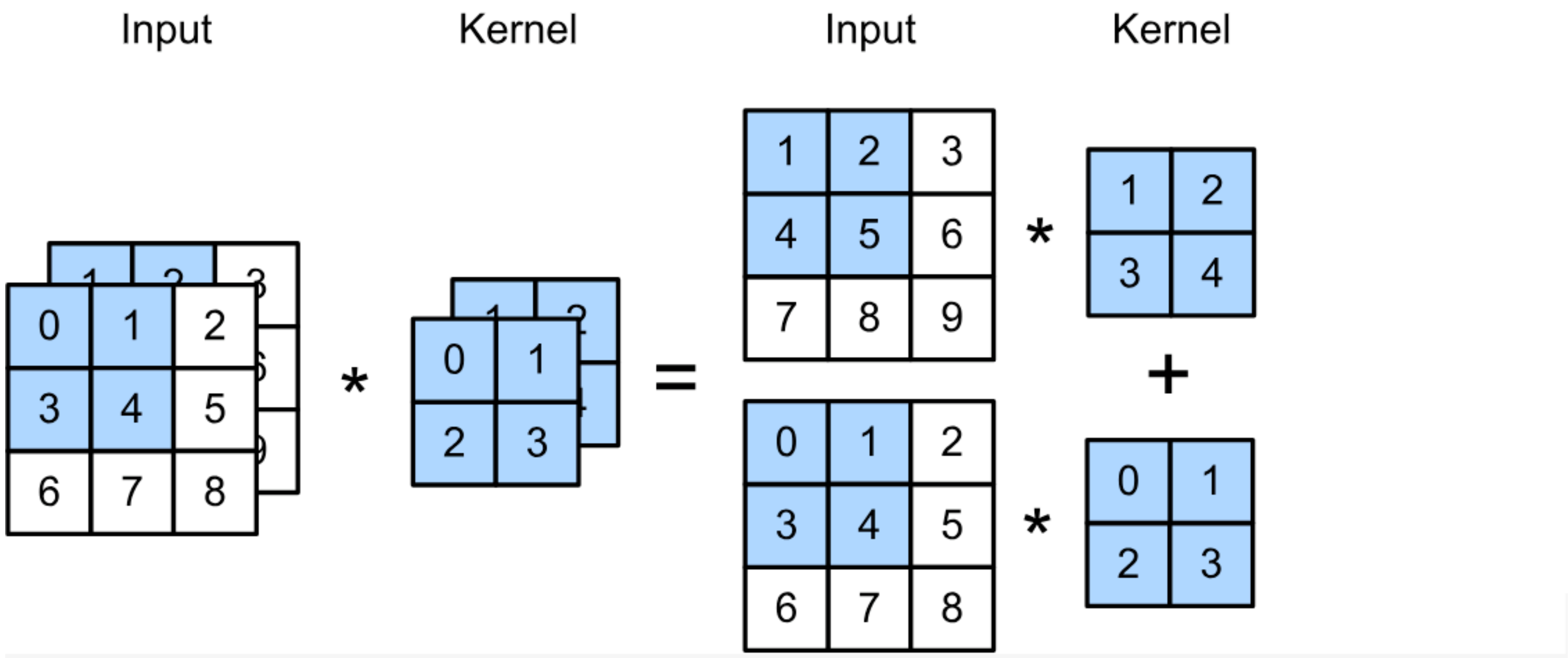
Multiple Input Channels

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



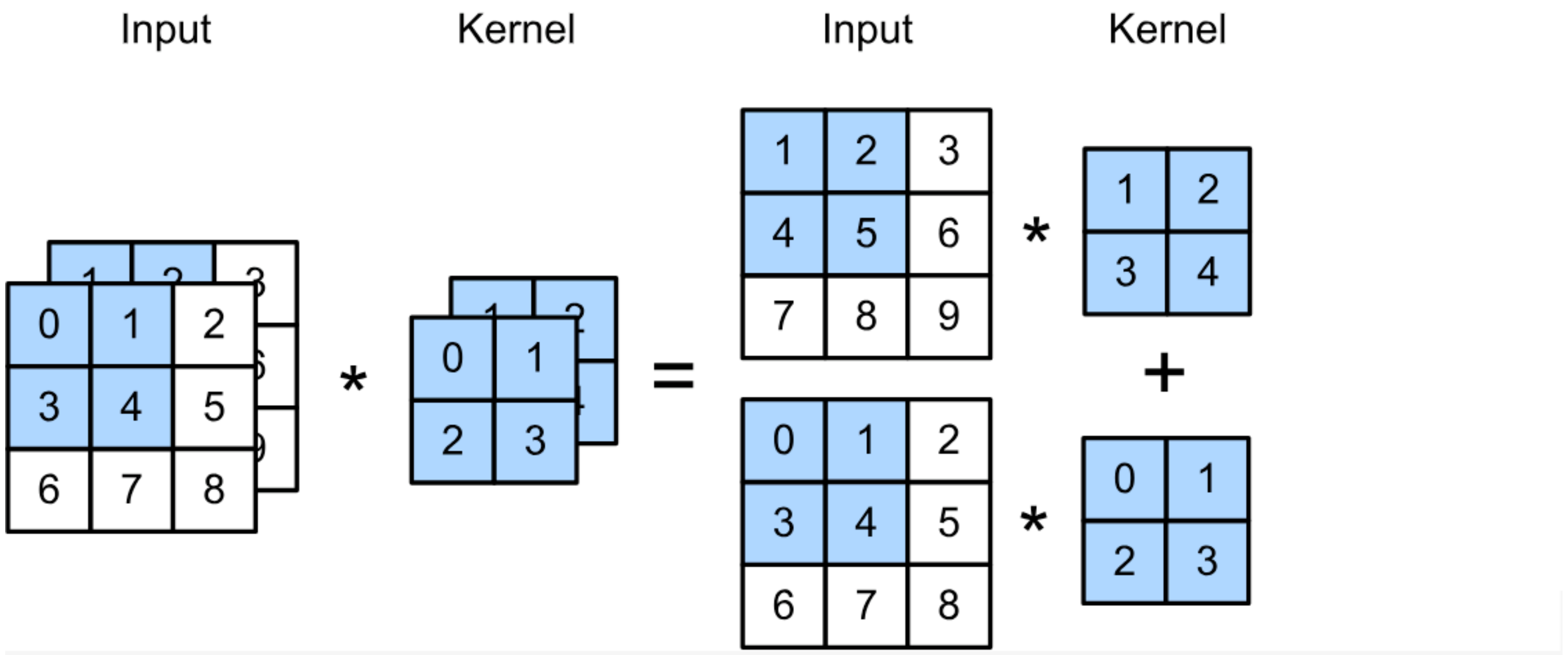
Multiple Input Channels

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



Multiple Input Channels

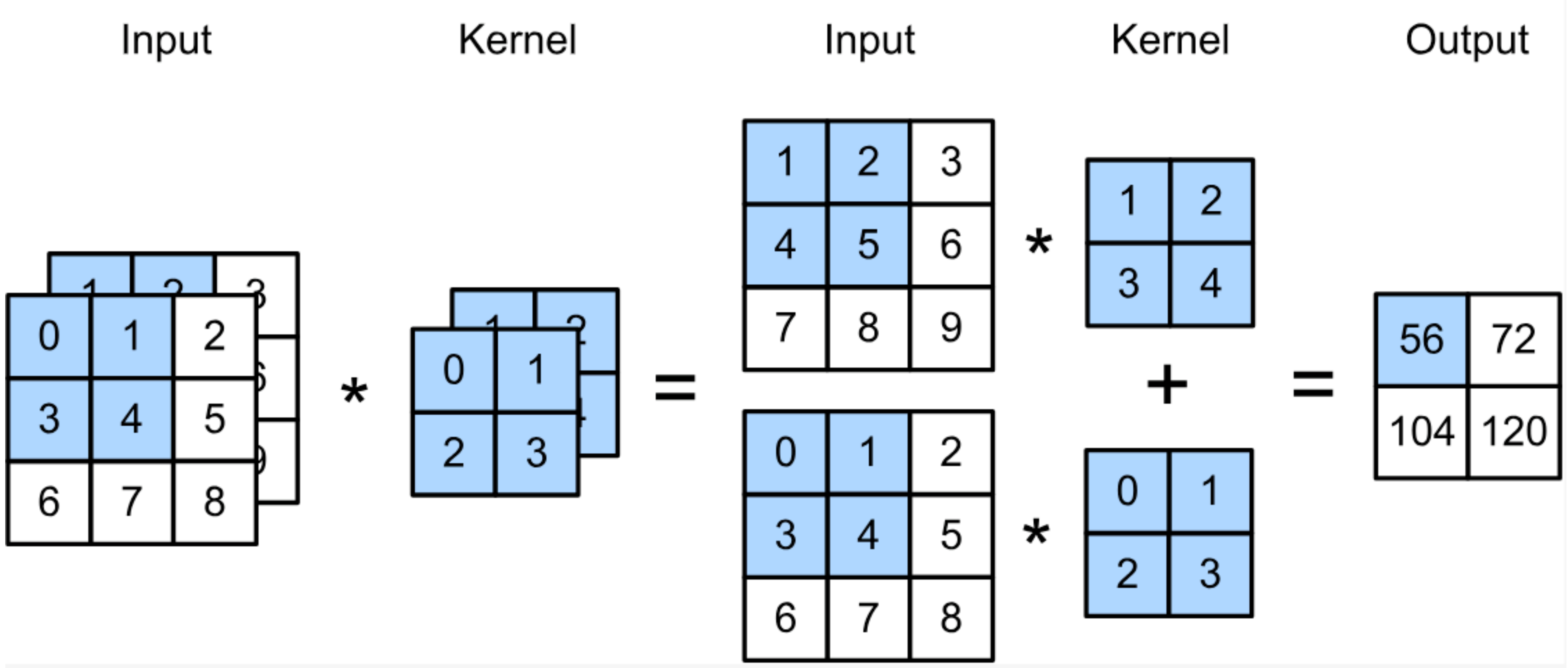
- Have a kernel for each channel, and then sum results over channels



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

Multiple Input Channels

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



$$(1\times1+2\times2+4\times3+5\times4)$$
$$+(0\times0+1\times1+3\times2+4\times3)$$
$$=56$$

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

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

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

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

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

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

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

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

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

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

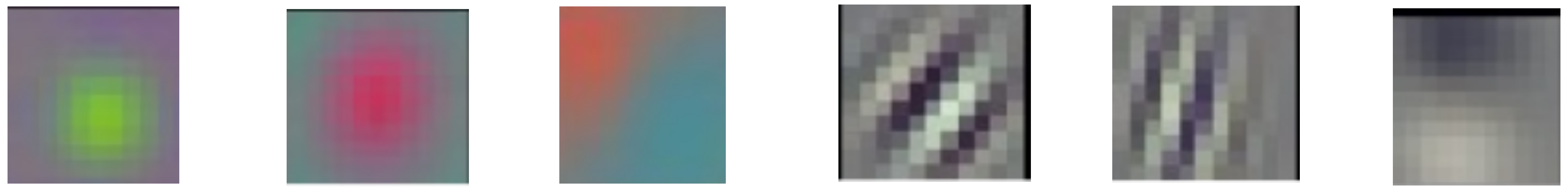
$$\text{for } i = 1, \ldots, c_o$$

Multiple Input/Output Channels

- Each 3-D kernel may recognize a particular pattern

Multiple Input/Output Channels

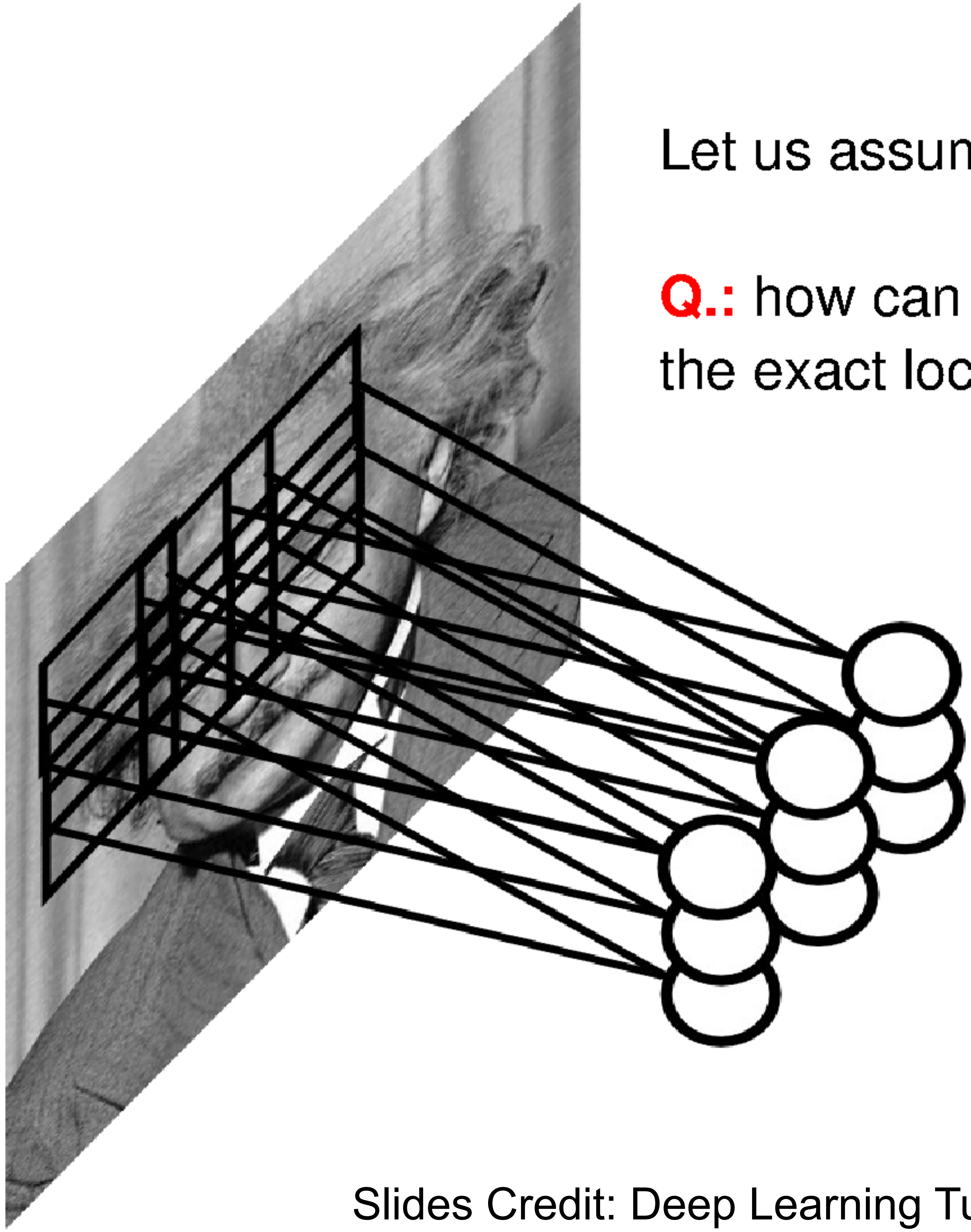
- Each 3-D kernel may recognize a particular pattern



(Gabor filters)

Pooling Layer

Pooling

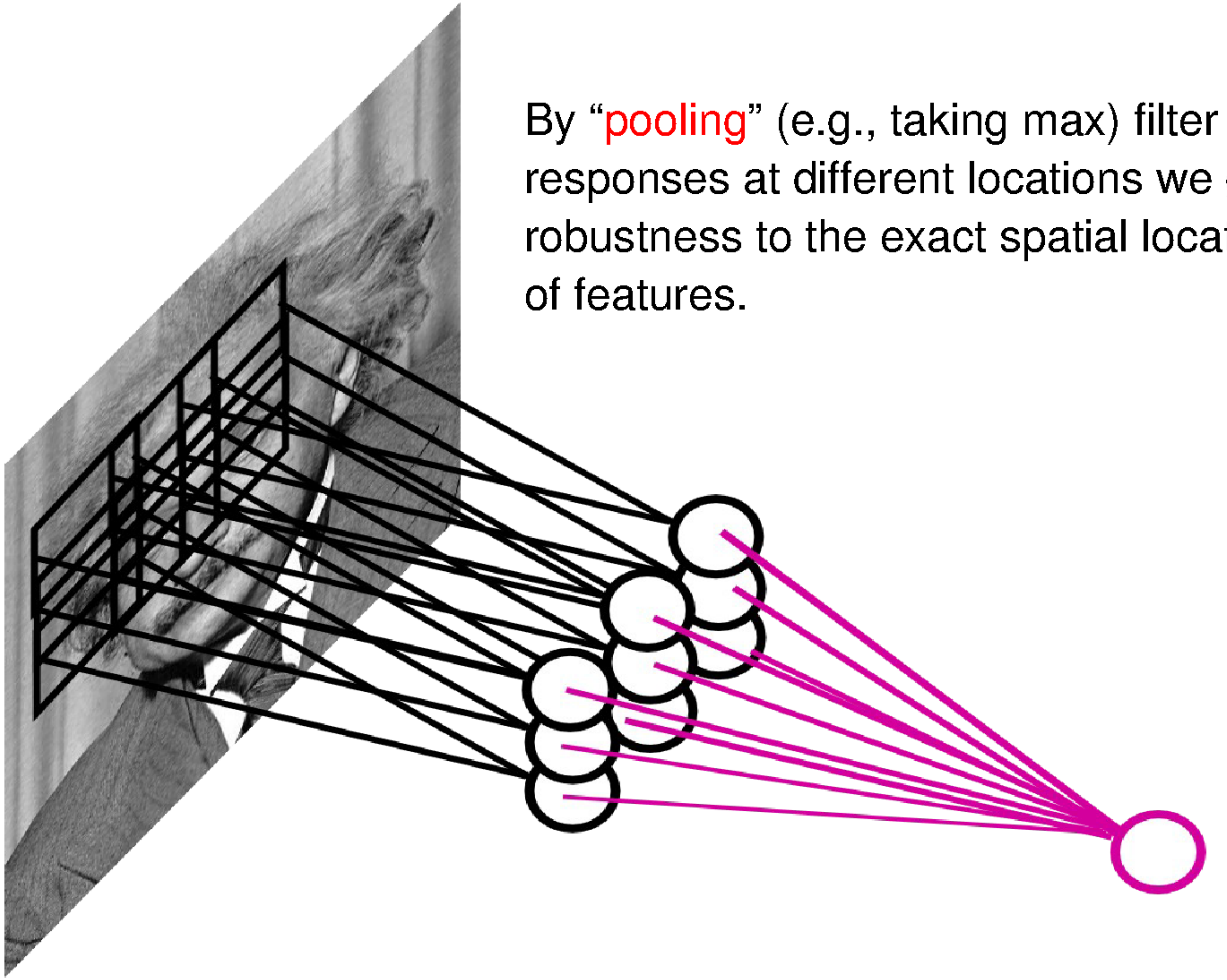


Let us assume filter is an “eye” detector.

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

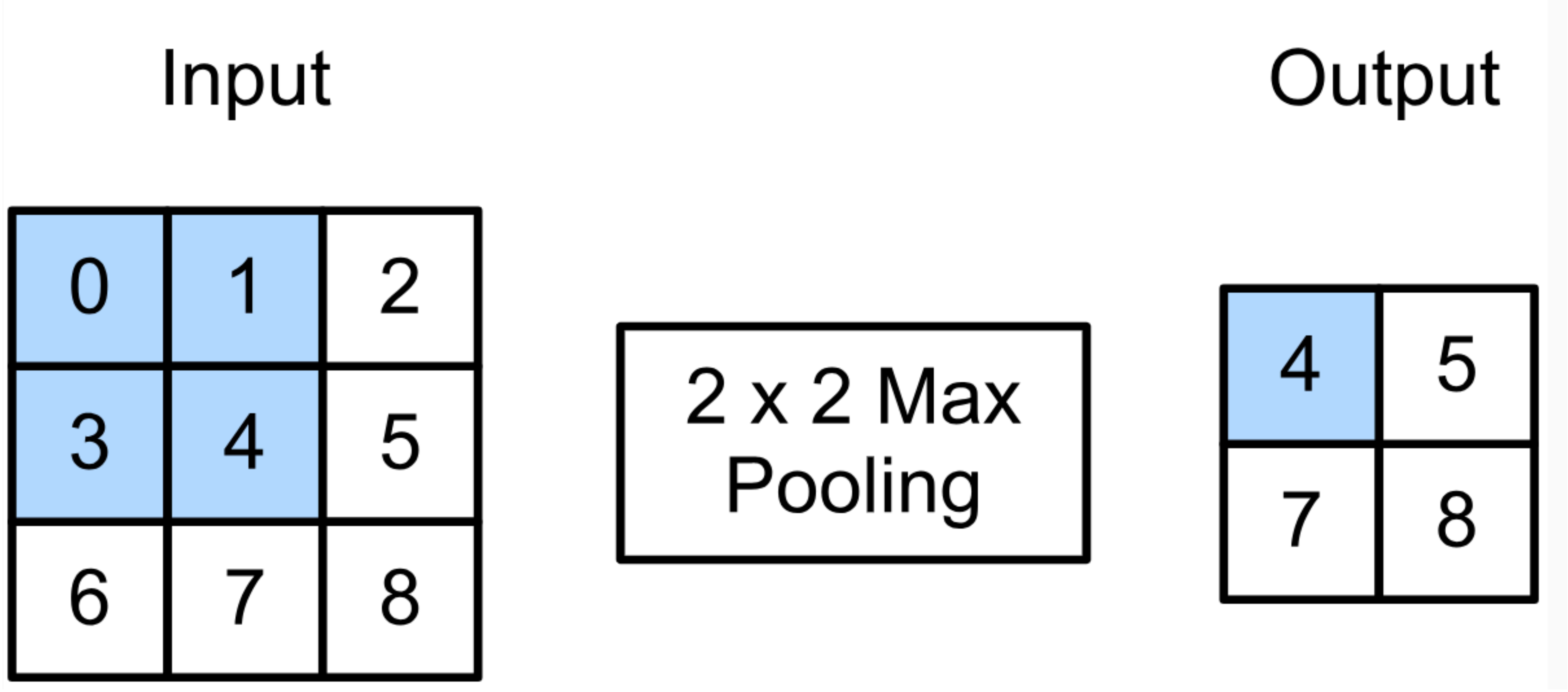


By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

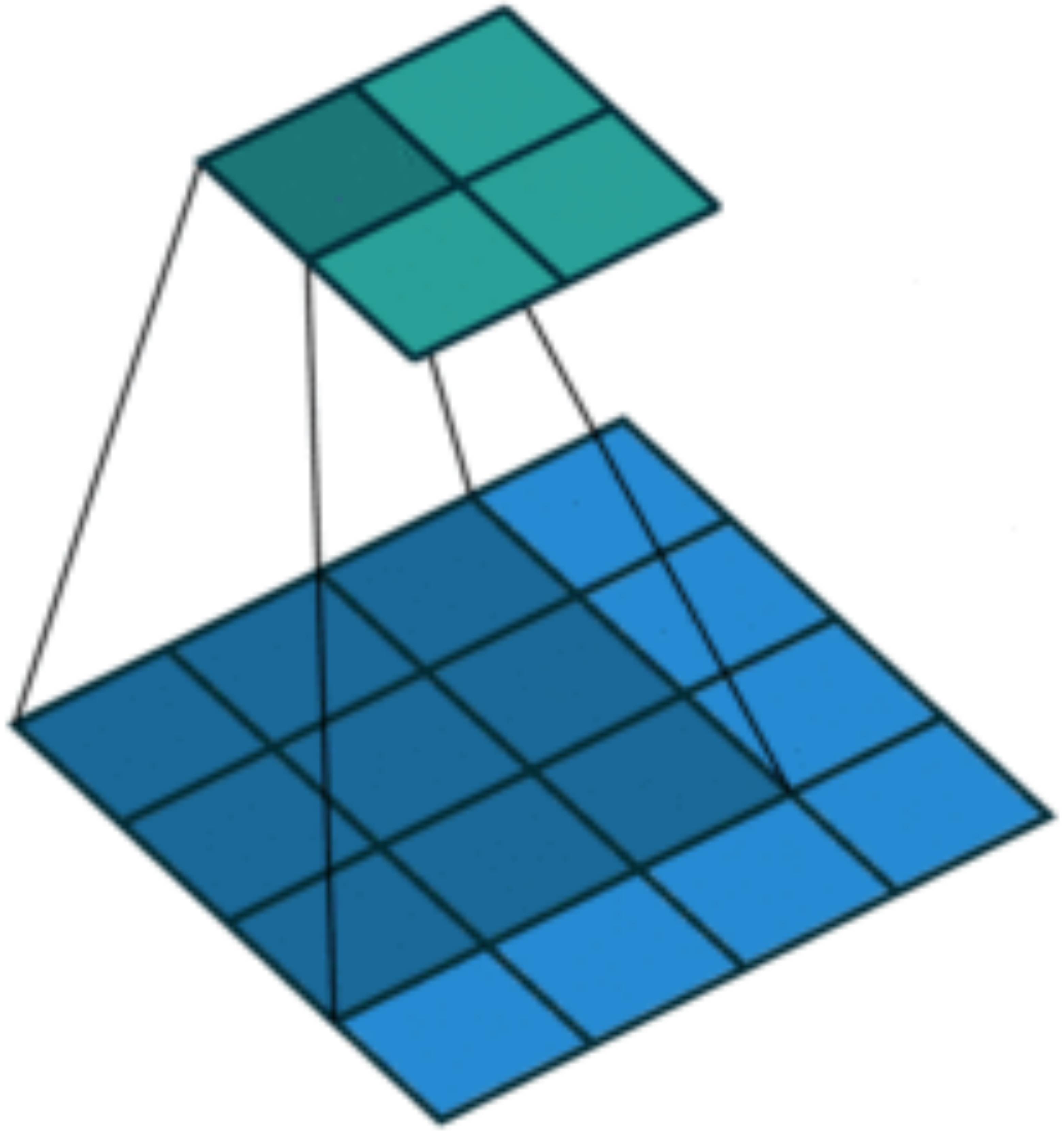
Slides Credit: Deep Learning Tutorial by Marc’Aurelio Ranzato

2-D Max Pooling

- Returns the maximal value in the sliding window

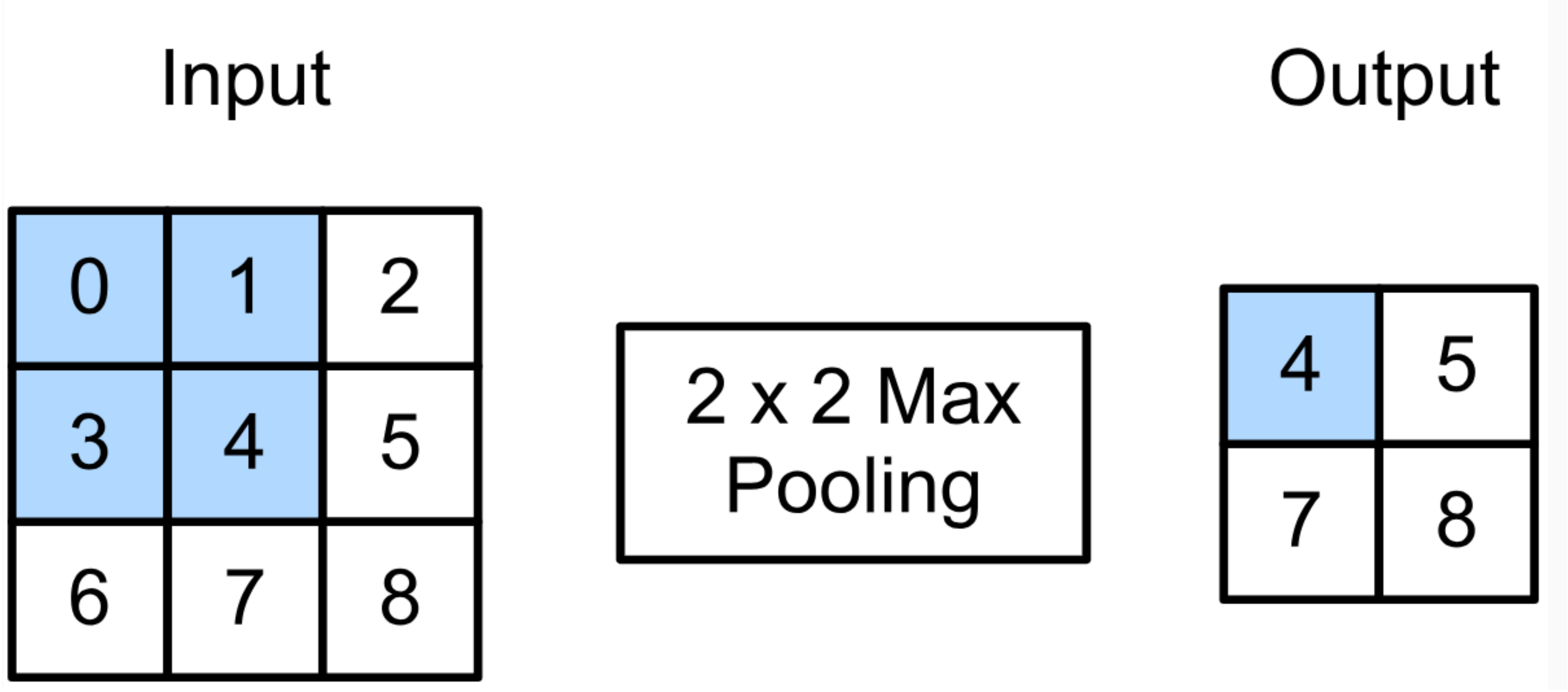


$$\max(0,1,3,4) = 4$$

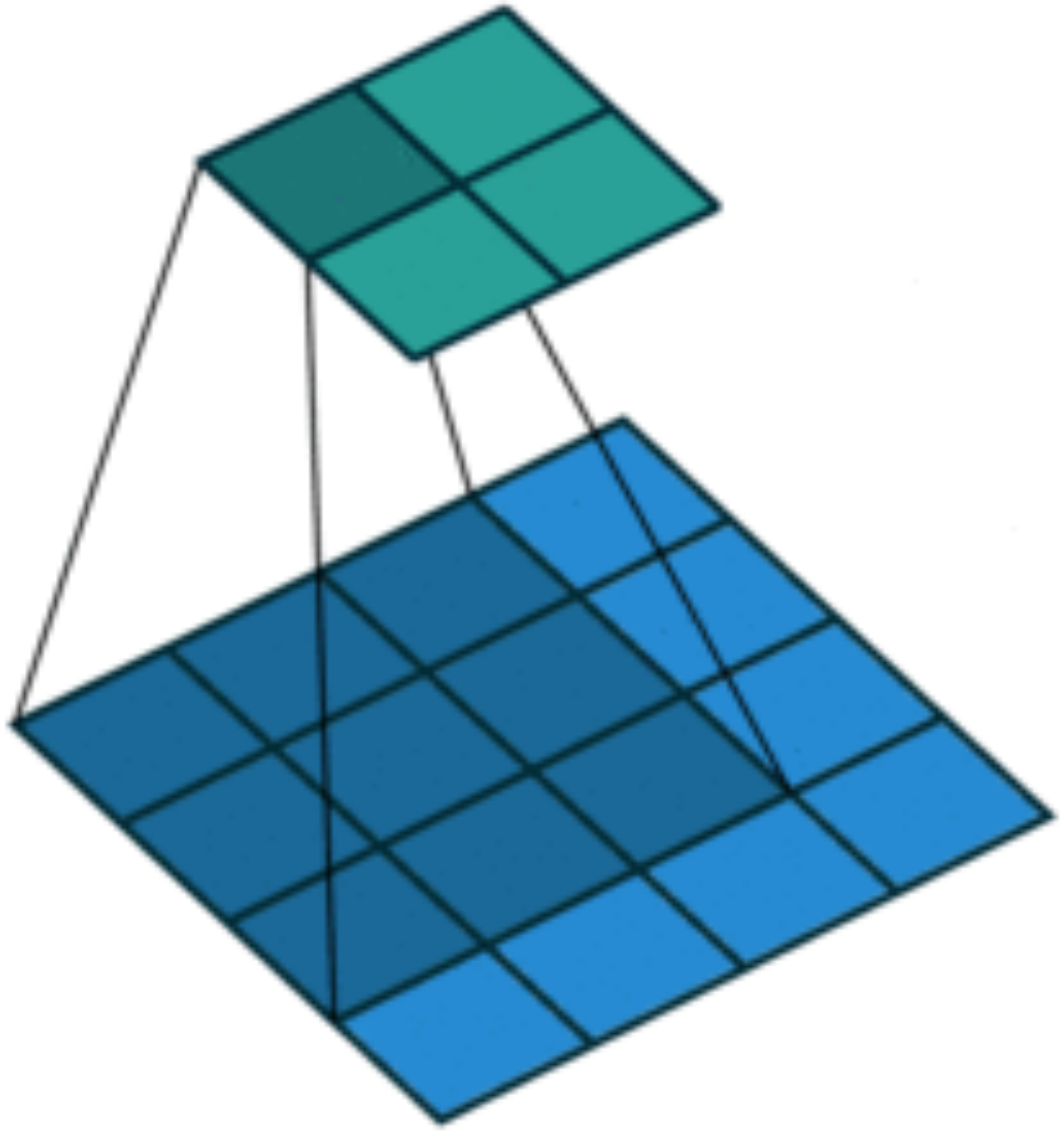


2-D Max Pooling

- Returns the maximal value in the sliding window



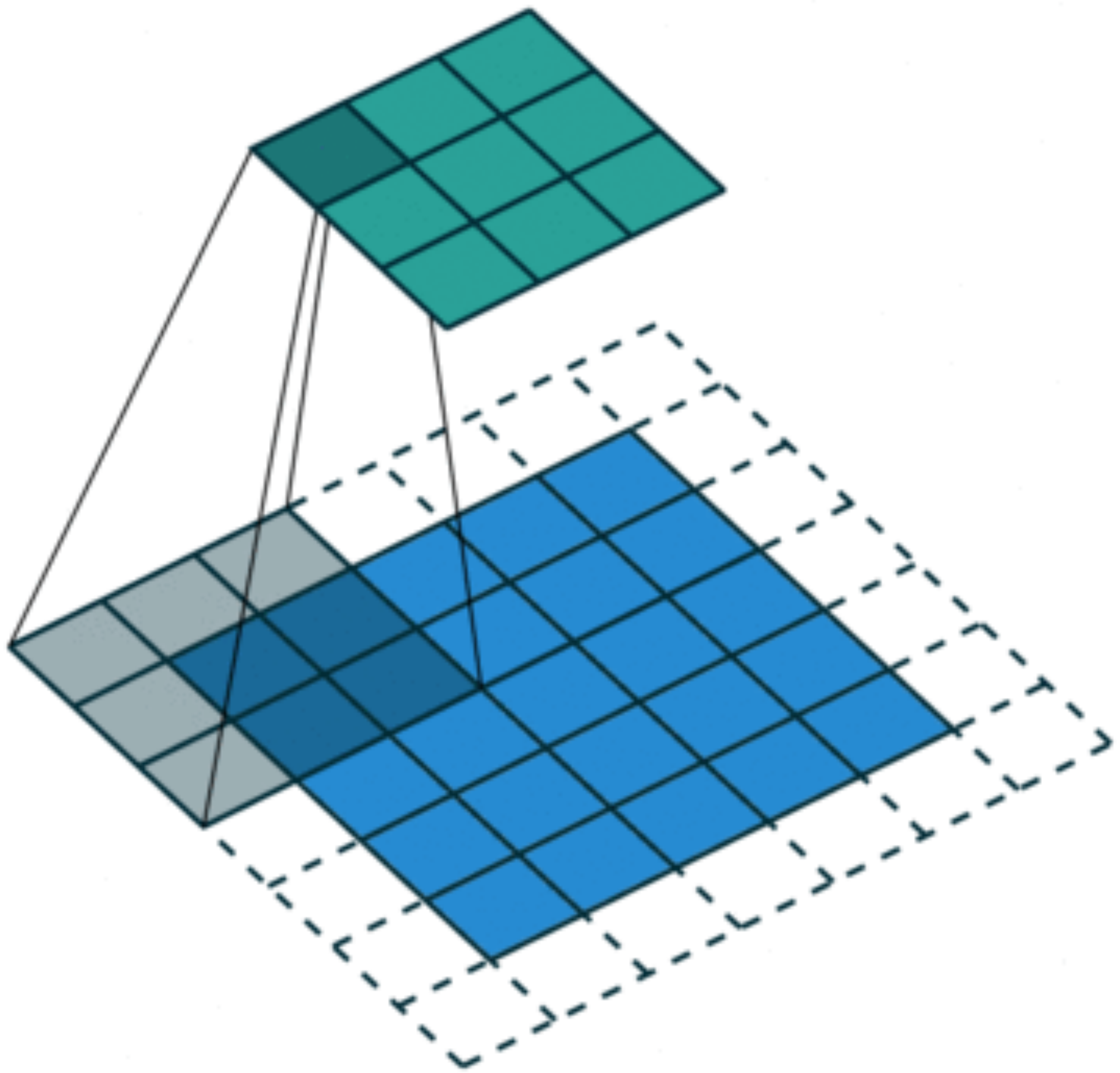
$$\max(0,1,3,4) = 4$$



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

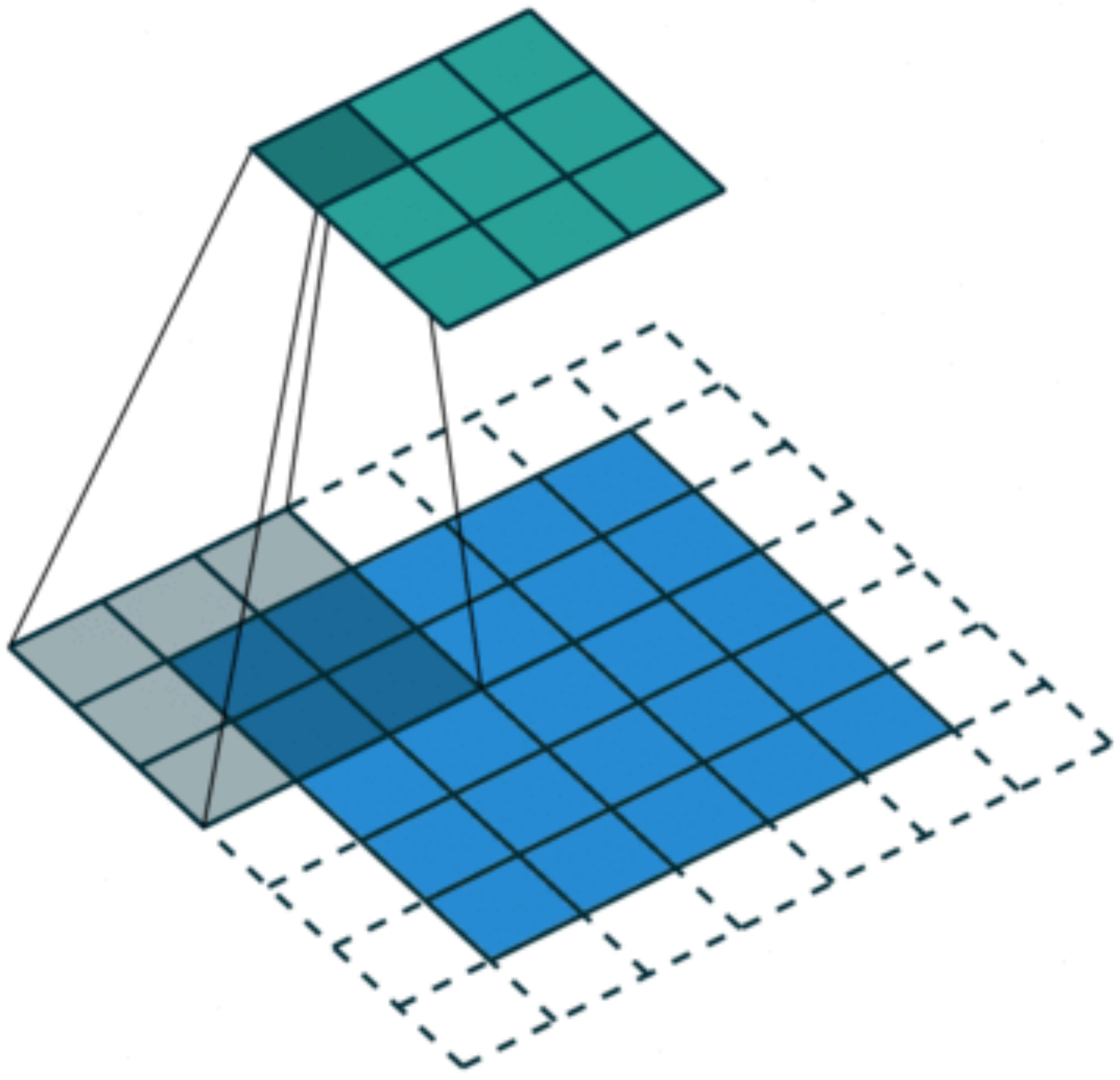
  #output channels = #input channels



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

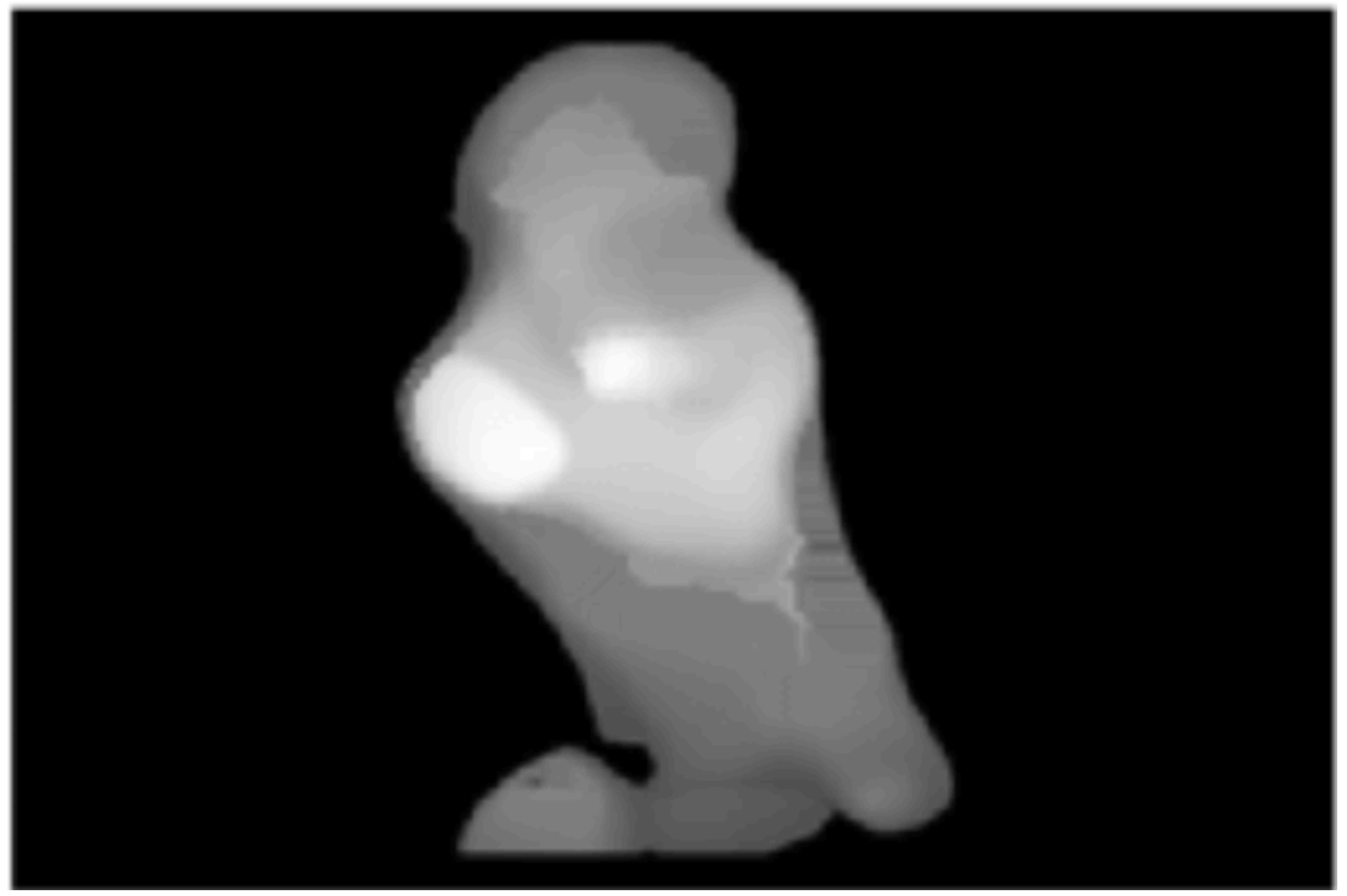
  #output channels = #input channels



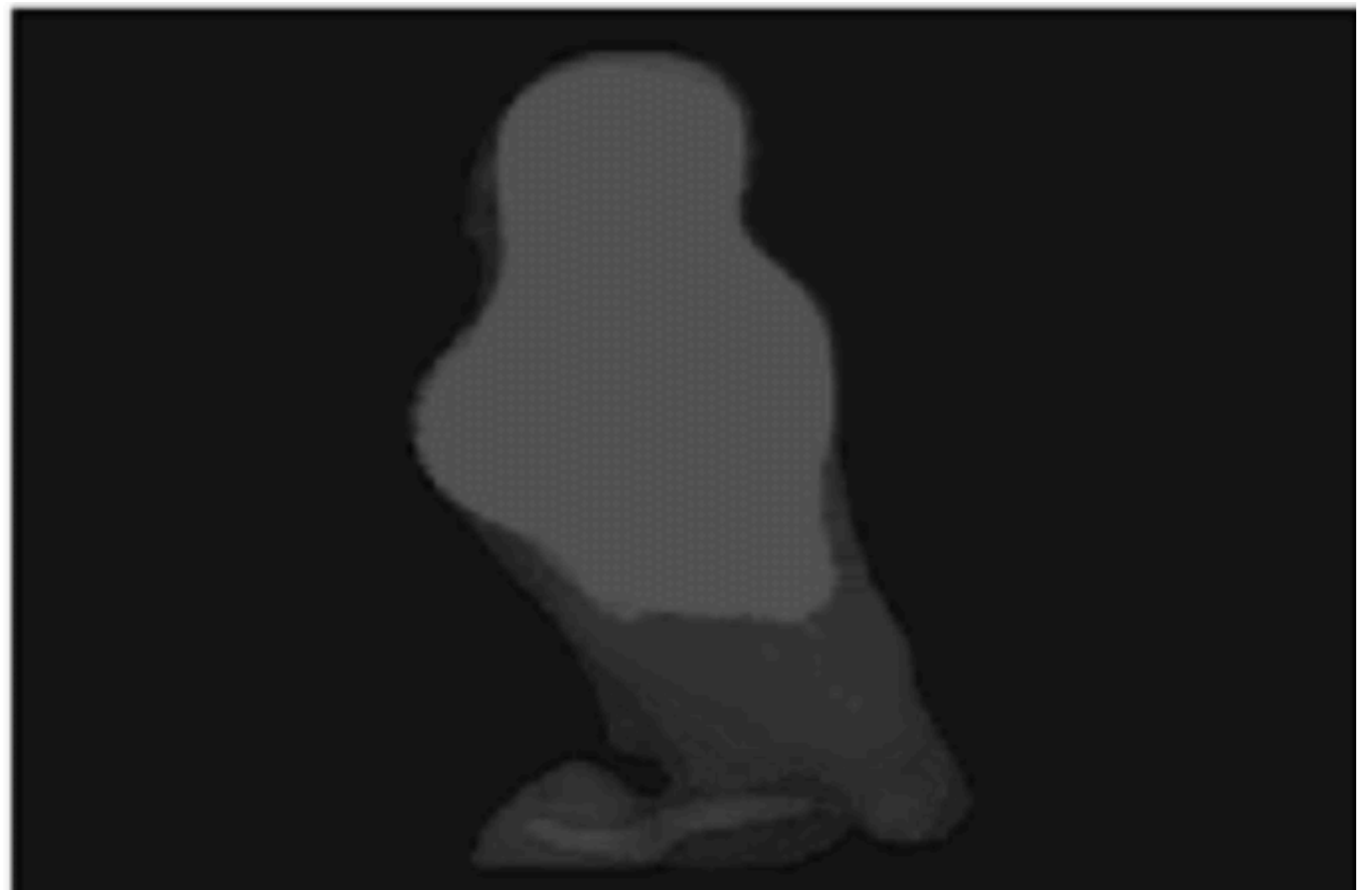
Average Pooling

- 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



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