



RHODES UNIVERSITY  
*Where leaders learn*

## CONVOLUTIONAL NETWORKS

**Marcellin Atemkeng**  
**(Rhodes University)**

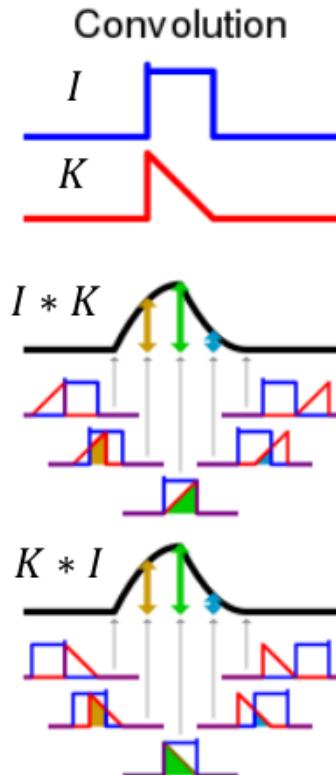


# Discrete convolutions

We'll focus on 1 & 2-D signals but signal dimensionality is arbitrary (usually 1,2,3,4-D)

The 2-D discrete convolution of two signals  $I$  and  $K$  is defined as:

$$\begin{aligned}(I * K)(i, j) &= \sum_m \sum_n I(m, n)K(i - m, j - n) \\ &= \sum_m \sum_n I(i - m, j - n)K(m, n)\end{aligned}$$



Where  $-\infty \leq m, n \leq \infty$ . Finite signals can be extended by adding zeros (more on this later)

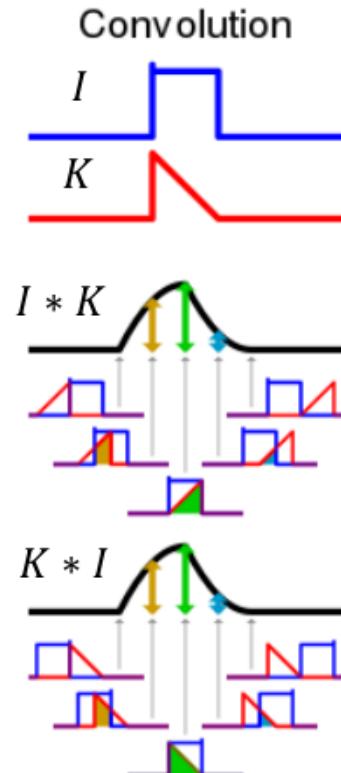
Image by Cmglee - Own work, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=20206863>

# Discrete convolutions

We'll focus on 1 & 2-D signals but signal dimensionality is arbitrary (usually 1,2,3,4-D)

The 2-D discrete convolution of two signals  $I$  and  $K$  is defined as:

$$\begin{aligned}(I * K)(i, j) &= \sum_m \sum_n I(m, n)K(i - m, j - n) \\ &= \sum_m \sum_n I(i - m, j - n)K(m, n)\end{aligned}$$



Where  $-\infty \leq m, n \leq \infty$ . Finite signals can be extended by adding zeros (more on this later)

Image by Cmglee - Own work, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=20206863>

# Convolution vs Cross-correlation

$$(I * K)(i, j) == \sum_m \sum_n I(m, n)K(i - m, j - n)$$

vs

$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

Cross-correlation quantifies **presence of  $I(i, j)$  in (shifted)  $K(i, j)$**

Most ML libraries implement convolutional layers as cross-correlation layers.

- Commutativity not important for most ConvNets
- Can avoid flipping one of the signals  $\Rightarrow$  **easier to implement**

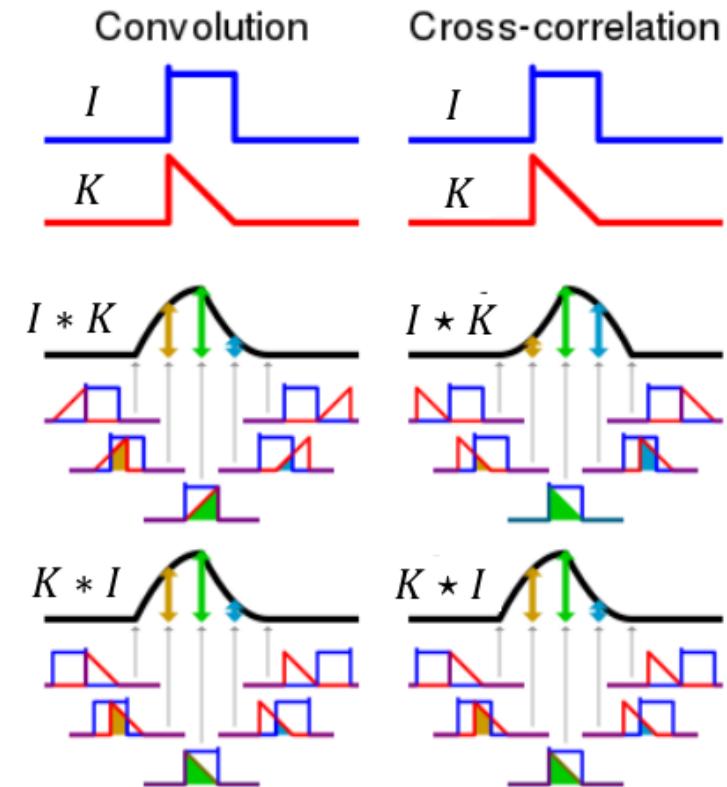


Image by Cmglee - Own work, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=20206883>

## Discrete cross-correlation: 2-D example

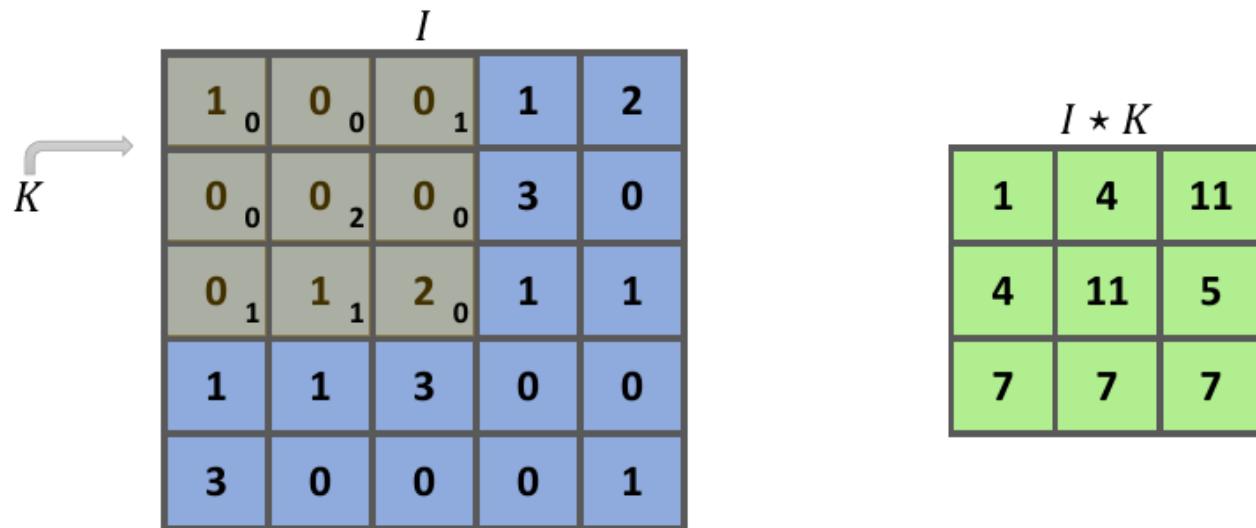
$$\begin{array}{c}
 I \\
 \hline
 \begin{matrix} 1 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 \end{matrix} \\
 \star \\
 K \\
 \hline
 \begin{matrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 0 \end{matrix}
 \end{array}$$

$$\begin{aligned}
 (I * K)(0,0) &= I(0,0)K(0,0) + I(0,1)K(0,1) + I(0,2)K(0,2) + \\
 &\quad I(1,0)K(1,0) + I(1,1)K(1,1) + I(1,2)K(1,2) + \\
 &\quad I(2,0)K(2,0) + I(2,1)K(2,1) + I(2,2)K(2,2) \\
 &= 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + \\
 &\quad 0 \cdot 0 + 0 \cdot 2 + 0 \cdot 0 + \\
 &\quad 0 \cdot 1 + 1 \cdot 1 + 2 \cdot 0 \\
 &= 1
 \end{aligned}$$

$$(I * K)(i,j) == \sum_m \sum_n I(m,n)K(i+m,j+n)$$

## Discrete cross-correlation: 2-D example

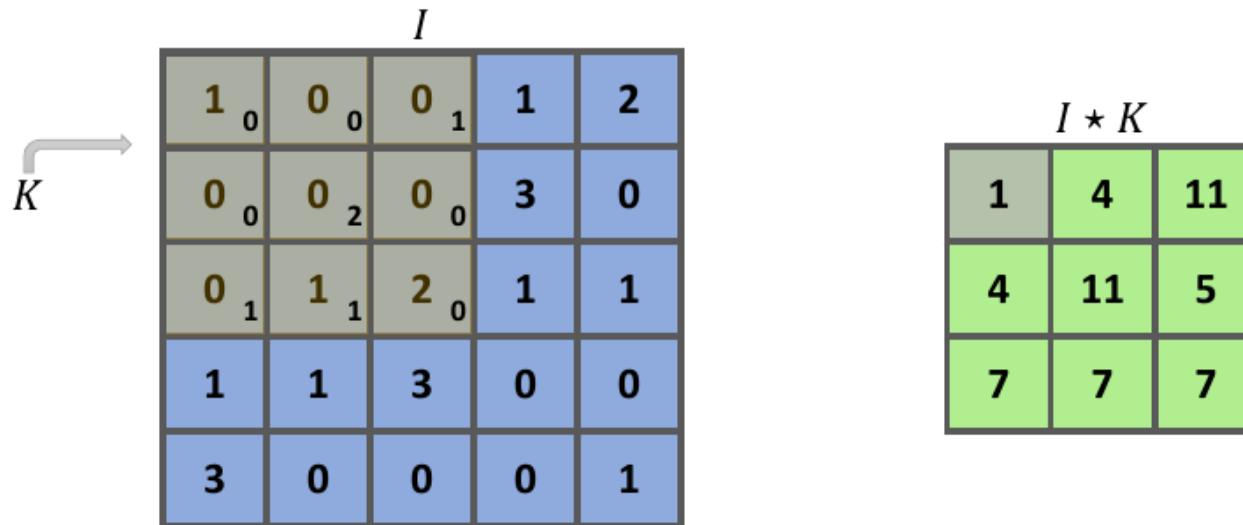
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

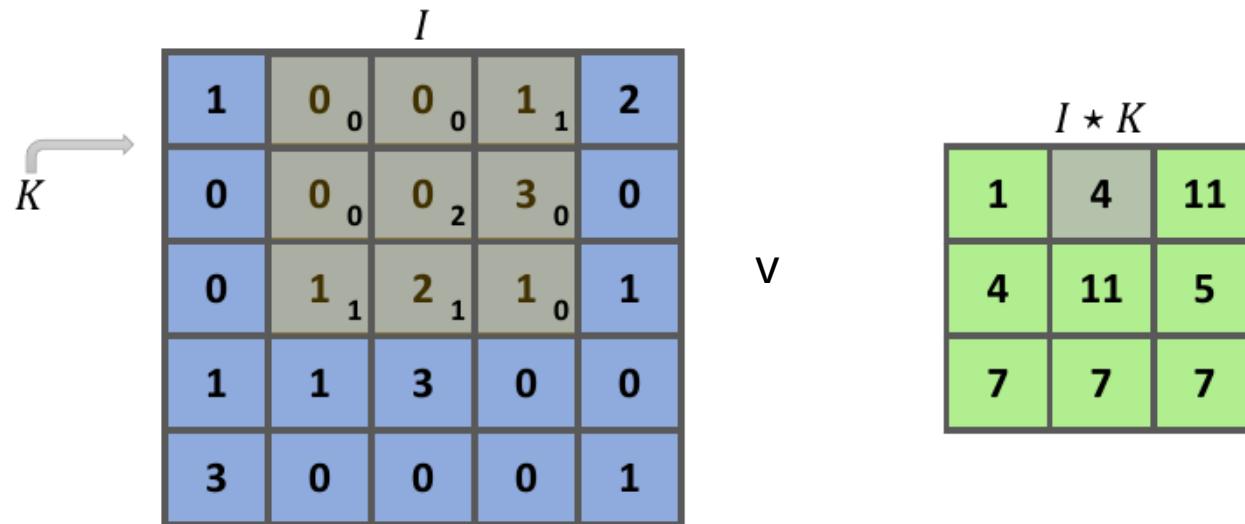
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

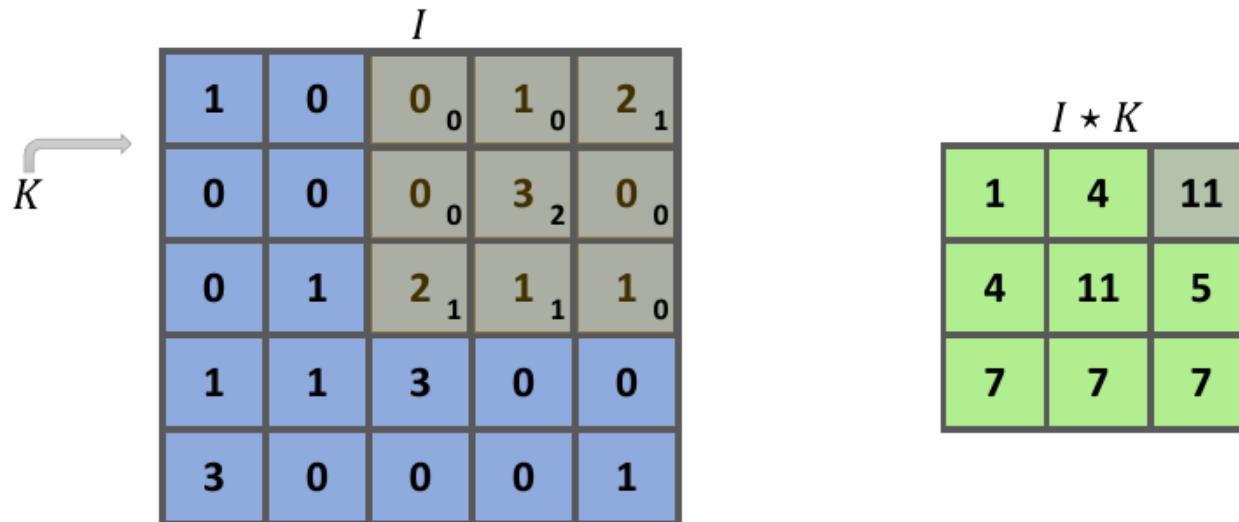
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

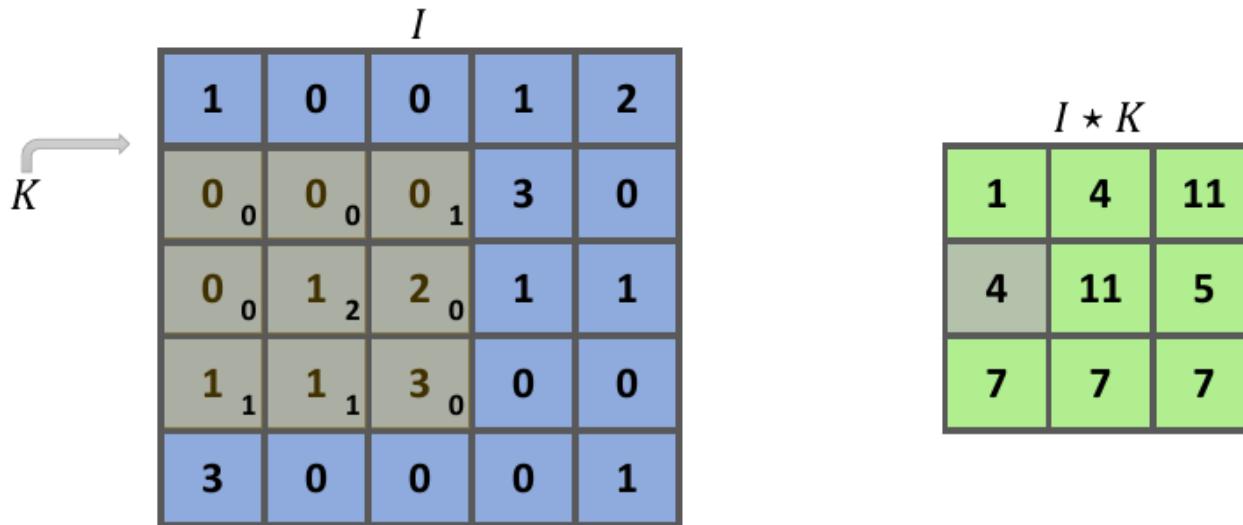
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

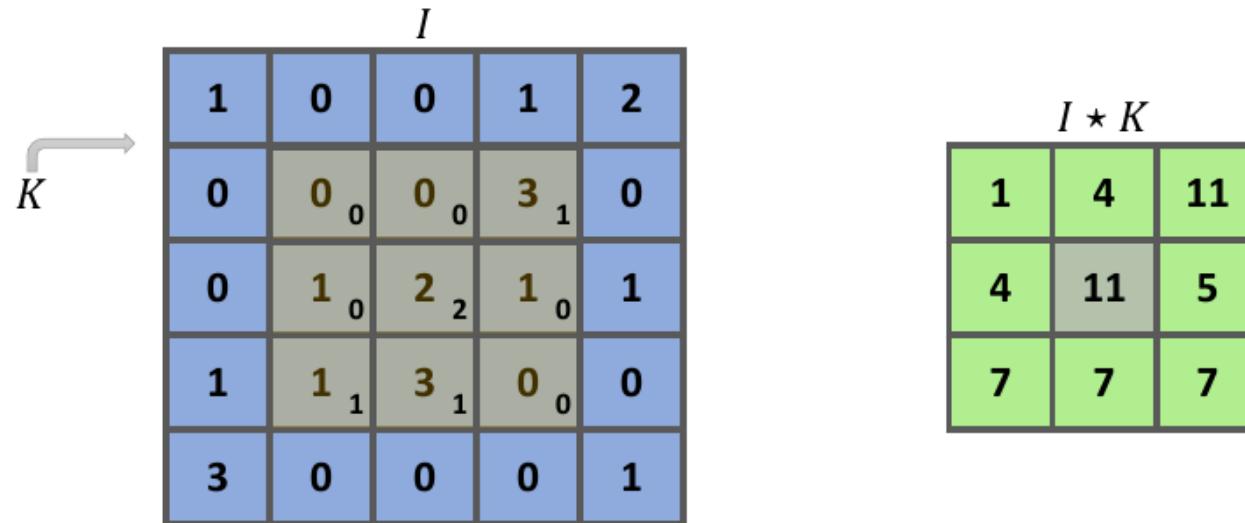
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

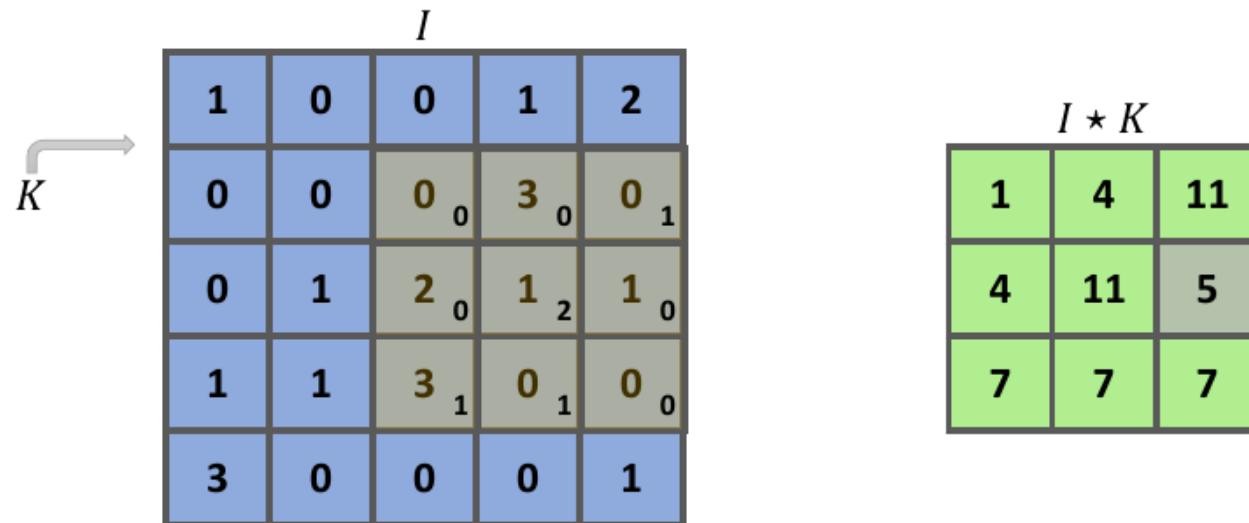
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

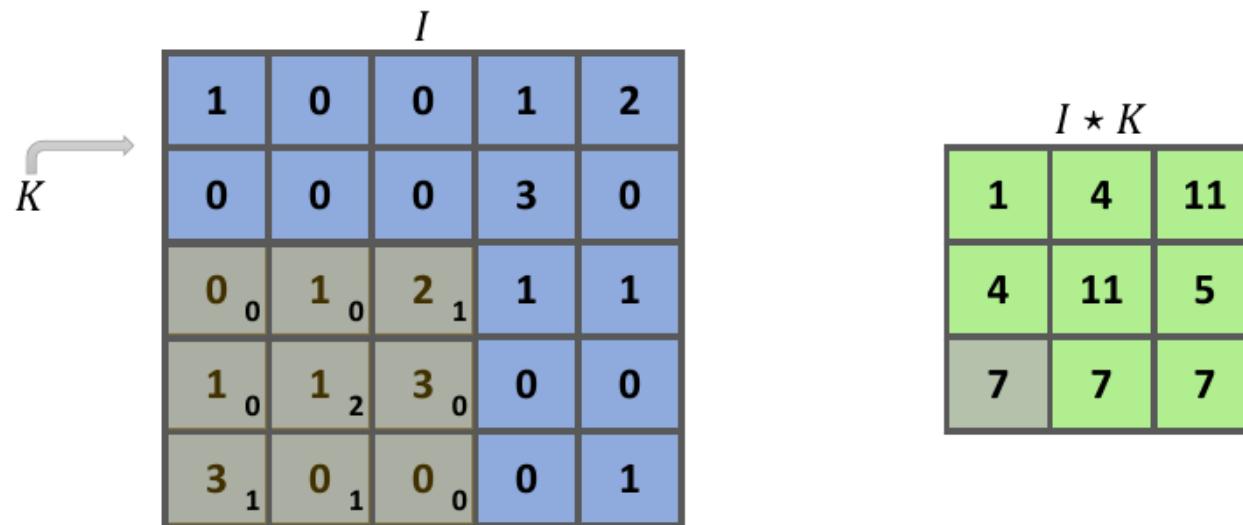
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

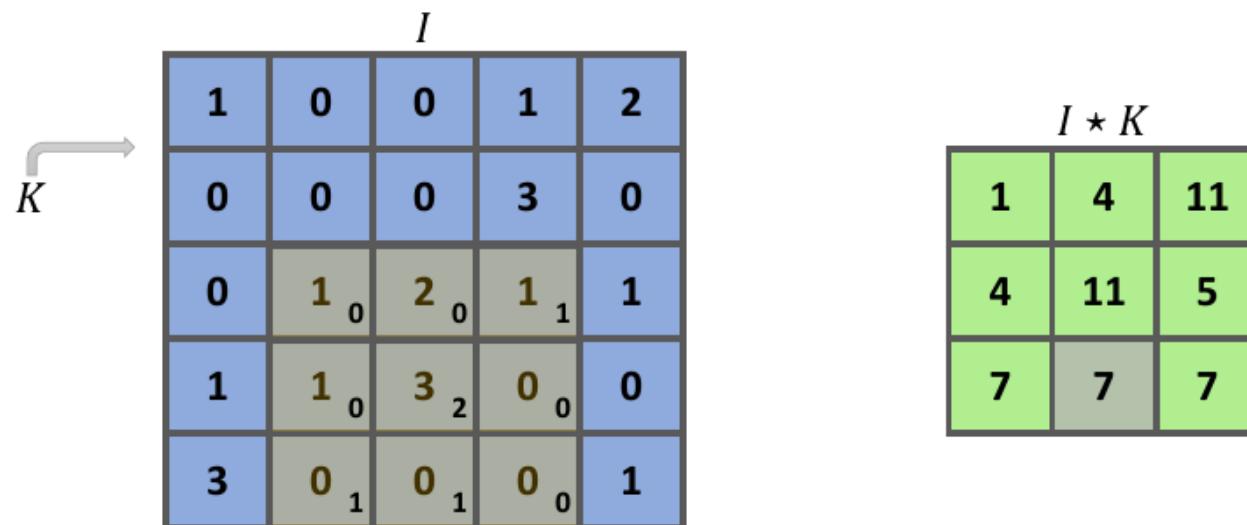
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

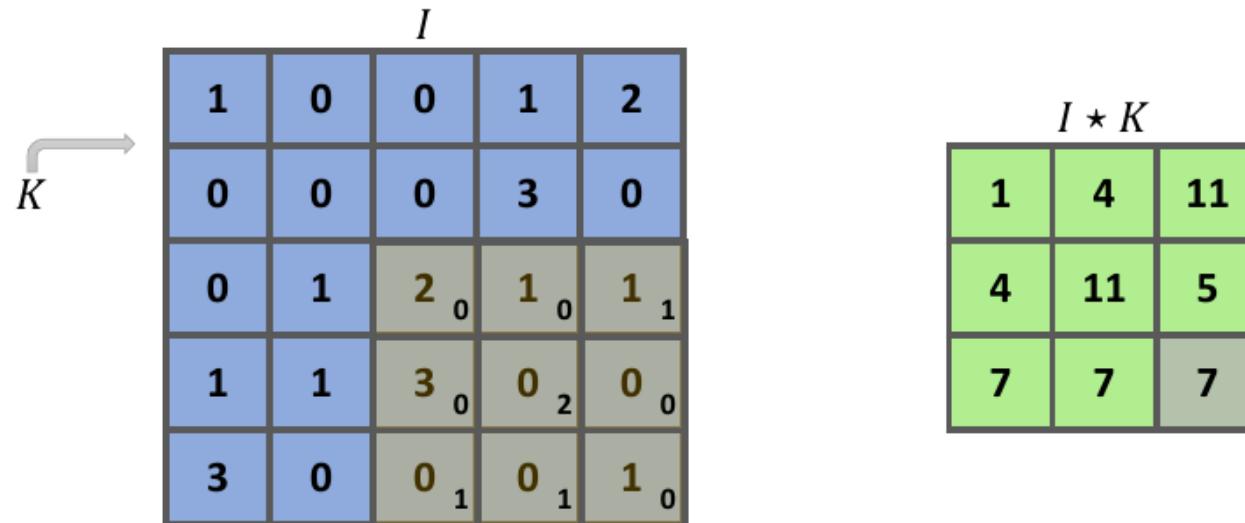
Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

Can be viewed as a “sliding window” operation:



$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

1	0	0	1	2
0	0	0	3	0
0	1	2	1	1
1	1	3	0	0
3	0	0	0	1

\*

0	0	1
0	2	0
1	1	0

=

1	4	11
4	11	5
7	7	7

Often called the  
feature map

$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: 2-D example

$$\begin{array}{|c|c|c|c|c|} \hline 1 & 0 & 0 & 1 & 2 \\ \hline 0 & 0 & 0 & 3 & 0 \\ \hline 0 & 1 & 2 & 1 & 1 \\ \hline 1 & 1 & 3 & 0 & 0 \\ \hline 3 & 0 & 0 & 0 & 1 \\ \hline \end{array} \star \begin{array}{|c|c|c|} \hline 0 & 0 & 1 \\ \hline 0 & 2 & 0 \\ \hline 1 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 1 & 4 & 11 \\ \hline 4 & 11 & 5 \\ \hline 7 & 7 & 7 \\ \hline \end{array}$$

Something's not quite right...

$$(I \star K)(i, j) == \sum_m \sum_n I(m, n)K(i + m, j + n)$$

## Discrete cross-correlation: padding

0 0	0 0	0 1	0	0	0	0	0	0
0 0	0 2	0 0	0	0	0	0	0	0
0 1	0 1	1 0	0	0	1	2	0	0
0 0	0 0	0 0	0	0	3	0	0	0
0 0	0 0	0 1	2	1	1	1	0	0
0 0	1 1	1 1	3	0	0	0	0	0
0 0	3 0	0 0	0	0	1	0	0	0
0 0	0 0	0 0	0	0	0	0	0	0
0 0	0 0	0 0	0	0	0	0	0	0

Technically, our signals have infinite extent...  
we solve by padding with zeros

1	4	11
4	11	5
7	7	7

## Discrete cross-correlation: padding

0 0	0 0	0 1	0	0	0	0	0	0
0 0	0 2	0 0	0	0	0	0	0	0
0 1	0 1	1 0	0	0	1	2	0	0
0	0	0	0	0	3	0	0	0
0	0	0	1	2	1	1	0	0
0	0	1	1	3	0	0	0	0
0	0	3	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

0								
	1	4	11					
	4	11	5					
	7	7	7					

## Discrete cross-correlation: padding

0	0 0	0 0	0 1	0	0	0	0	0
0	0 0	0 2	0 0	0	0	0	0	0
0	0 1	1 1	0 0	0	1	2	0	0
0	0	0	0	0	3	0	0	0
0	0	0	1	2	1	1	0	0
0	0	1	1	3	0	0	0	0
0	0	3	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

0	1							
1	4	11						
4	11	5						

## Discrete cross-correlation: padding

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	2	0	0
0	0	0	0	0	3	0	0	0
0	0	0	1	2	1	1	0	0
0	0	1	1	3	0	0	0	0
0	0	3	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

0	1	1						
1	4	11						
4	11	5						
7	7	7						

## Discrete cross-correlation: padding

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	2	0	0
0	0	0	0	3	0	0	0	0
0	0	0	1	2	1	1	0	0
0	0	1	1	3	0	0	0	0
0	0	3	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Normally we want the output to maintain the input size

0	1	1	0	1	3	2
0	2	0	0	5	7	0
1	0	1	4	11	2	1
0	1	4	11	5	2	0
0	6	7	7	7	1	1
1	7	3	0	0	2	0
3	0	0	0	1	0	0

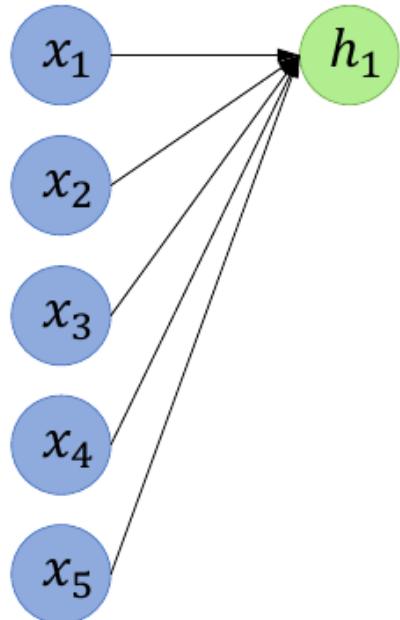
## Discrete cross-correlation: padding

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	2	0	0
0	0	0	0	0	3	0	0	0
0	0	0	1	2	1	1	0	0
0	0	1	1	3	0	0	0	0
0	0	3	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Normally we want the output to maintain the input size

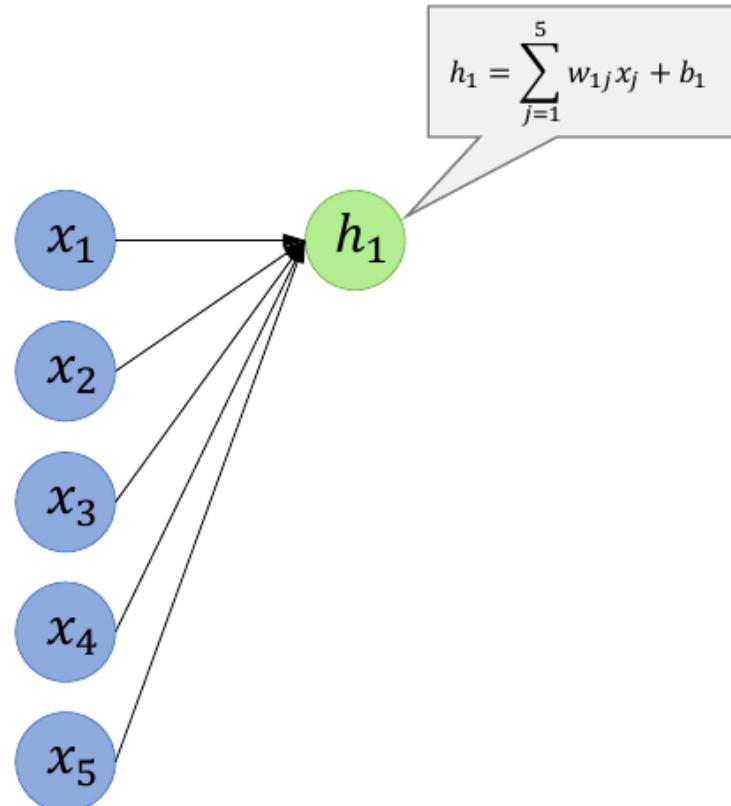
0	1	1	0	1	3	2
0	2	0	0	5	7	0
1	0	1	4	11	2	1
0	1	4	11	5	2	0
0	6	7	7	7	1	1
1	7	3	0	0	2	0
3	0	0	0	1	0	0

# Neural networks ⇒ Convolutional networks



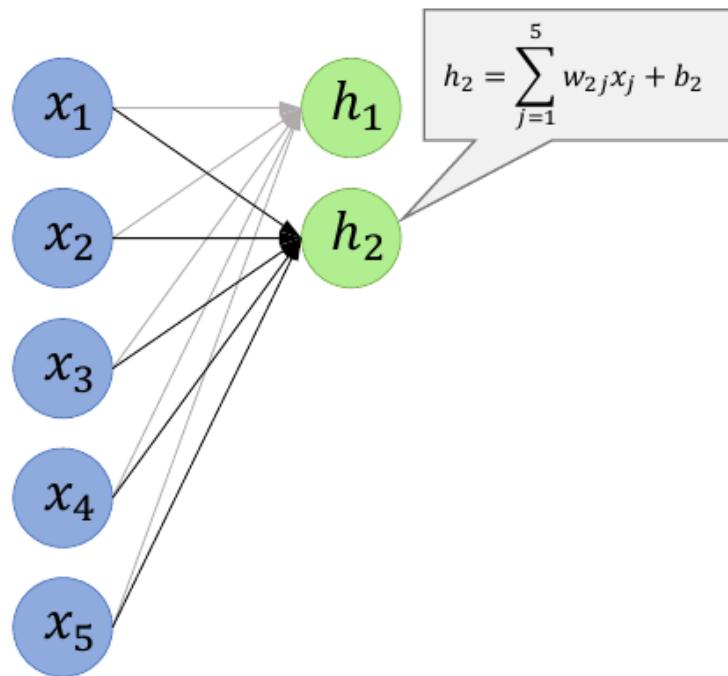
$$\mathbf{h} = \mathbf{Wx} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$

## Neural networks ⇒ Convolutional networks



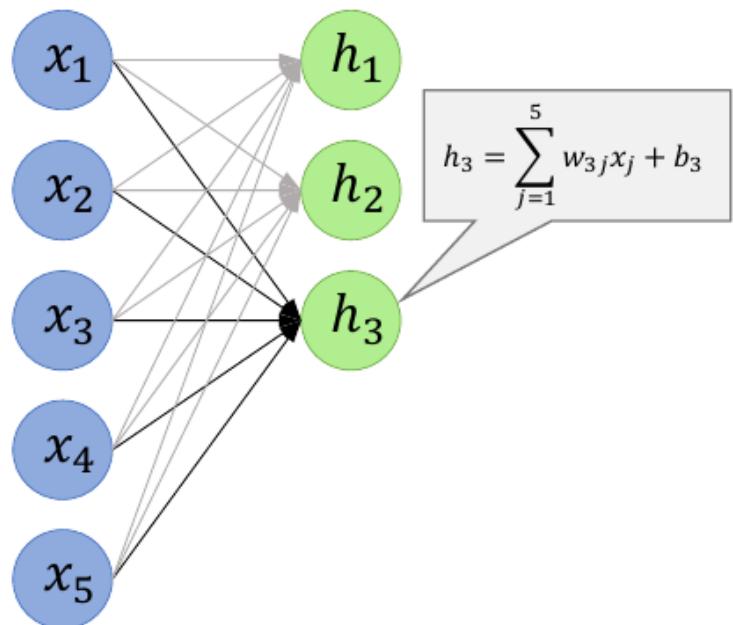
$$\mathbf{h} = \mathbf{Wx} + \mathbf{b}; h_i = \sum_j w_{ij} x_j + b_i$$

# Neural networks ⇒ Convolutional networks



$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$

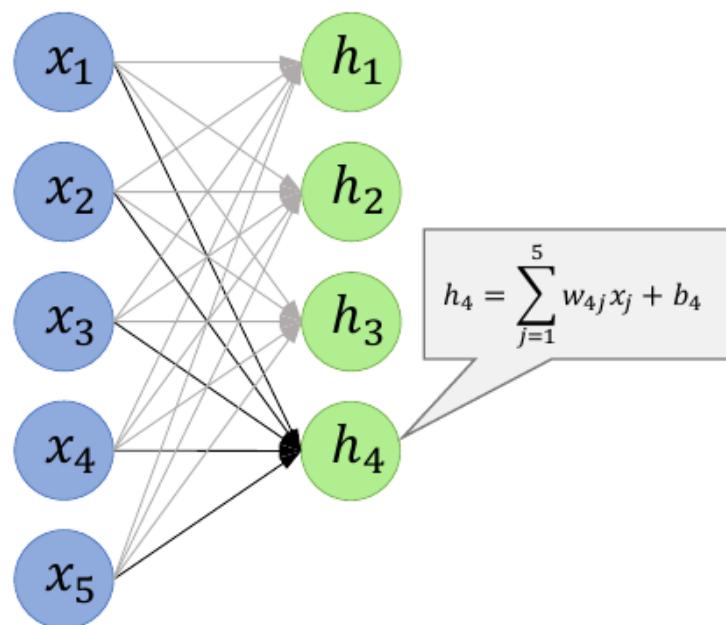
## Neural networks ⇒ Convolutional networks



$$h_3 = \sum_{j=1}^5 w_{3j}x_j + b_3$$

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$

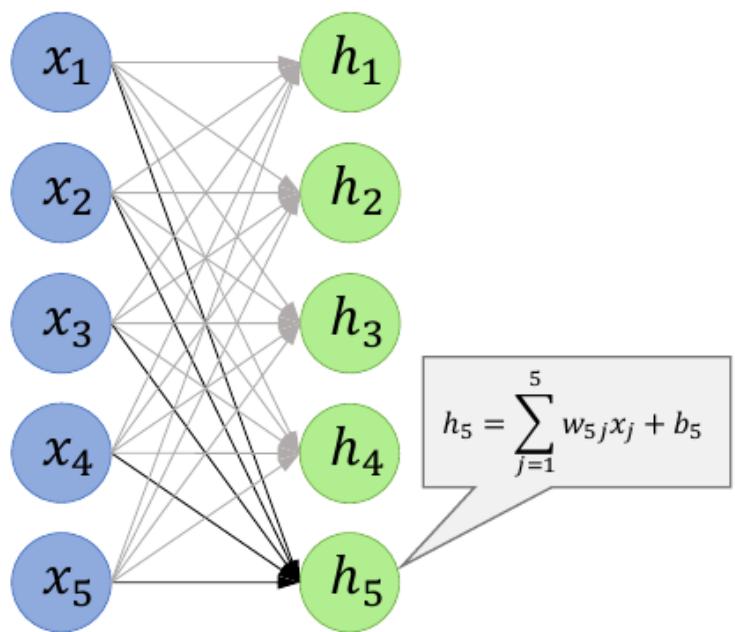
## Neural networks ⇒ Convolutional networks



$$h_4 = \sum_{j=1}^5 w_{4j} x_j + b_4$$

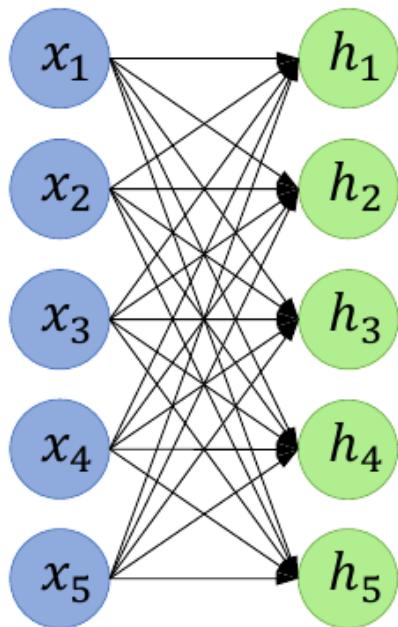
$$\mathbf{h} = \mathbf{Wx} + \mathbf{b}; h_i = \sum_j w_{ij} x_j + b_i$$

# Neural networks ⇒ Convolutional networks

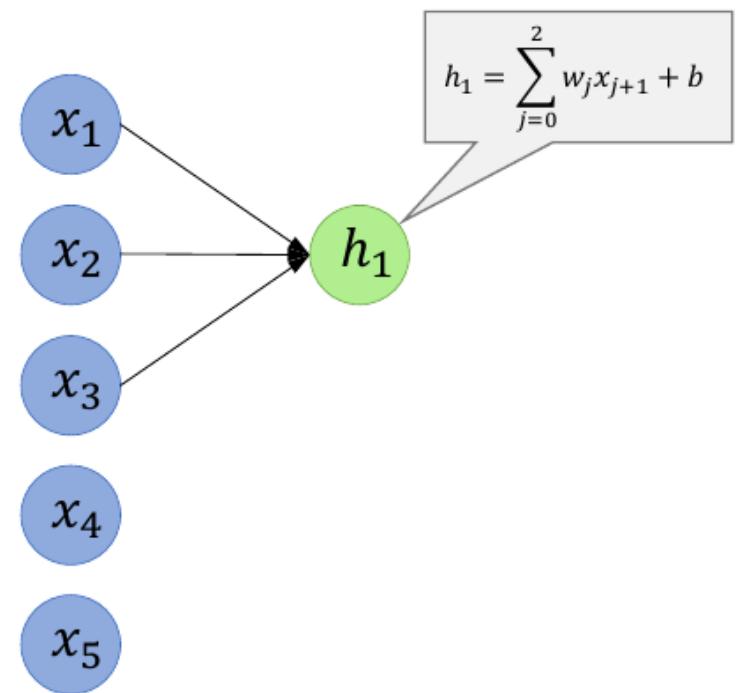


$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$

# Neural networks ⇒ Convolutional networks

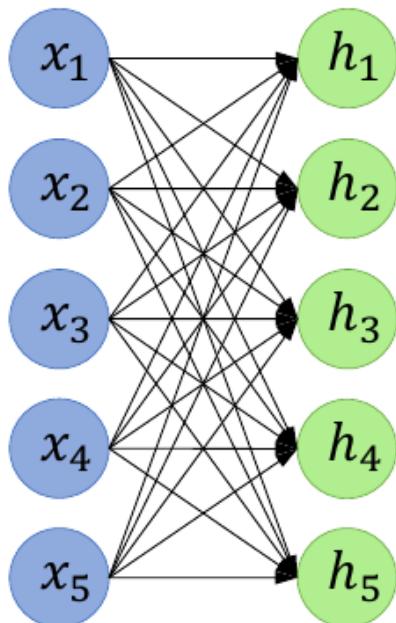


$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$

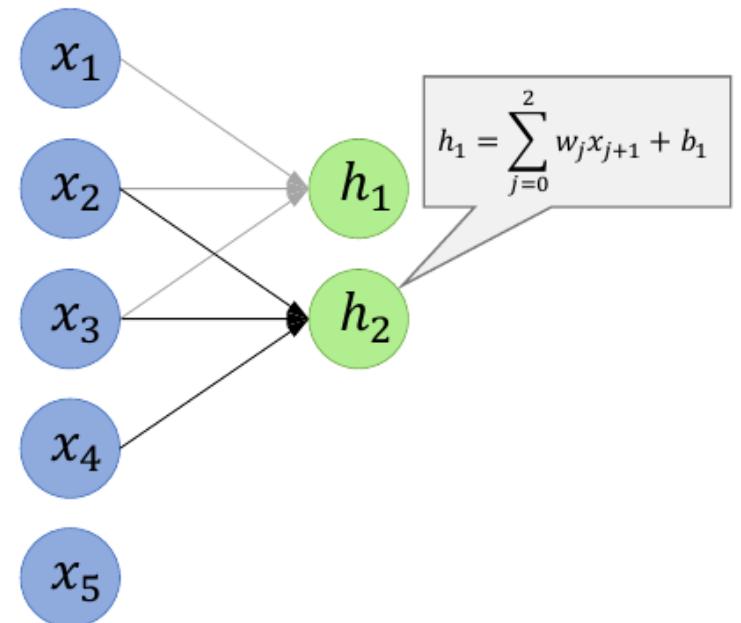


$$h_i = \sum_j w_j x_{j+i} + b$$

# Neural networks $\Rightarrow$ Convolutional networks



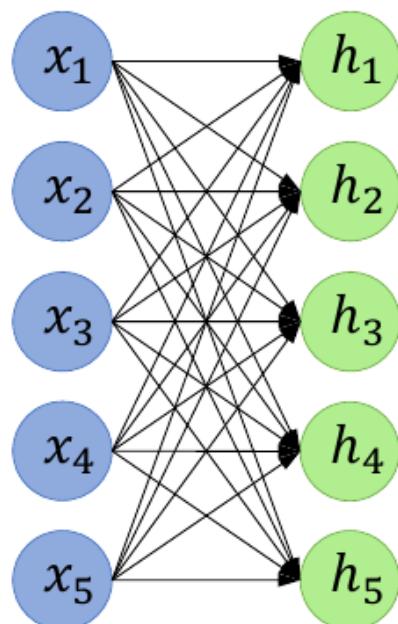
$$\mathbf{h} = \mathbf{Wx} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$



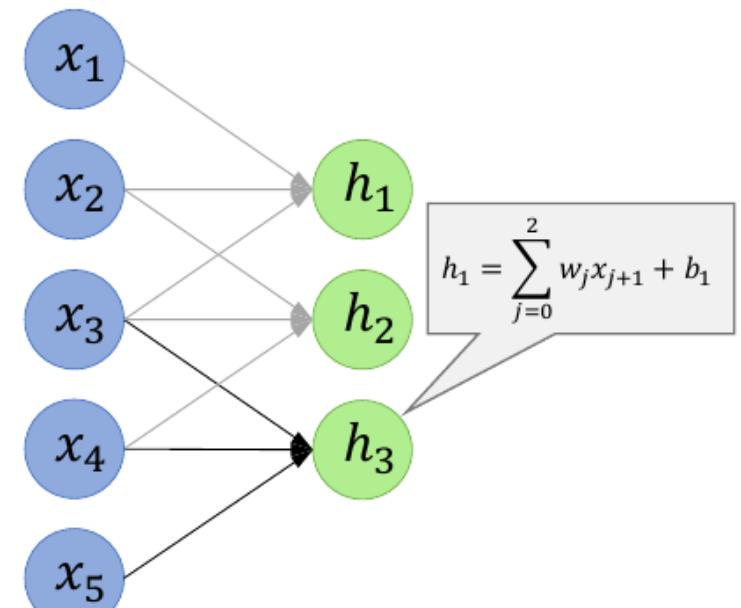
$$h_i = \sum_j w_j x_{j+i} + b$$

$$h_1 = \sum_{j=0}^2 w_j x_{j+1} + b_1$$

## Neural networks ⇒ Convolutional networks

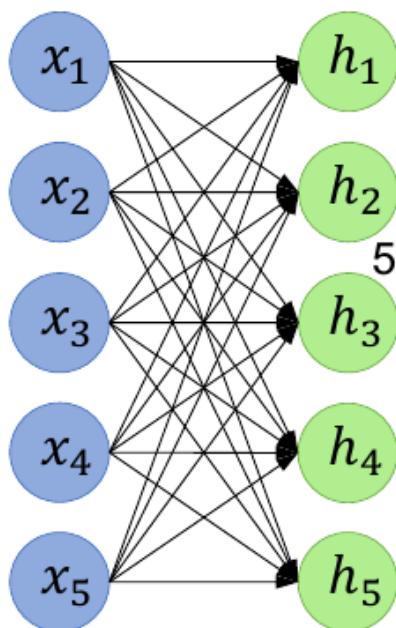


$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$



$$h_i = \sum_j w_j x_{j+i} + b$$

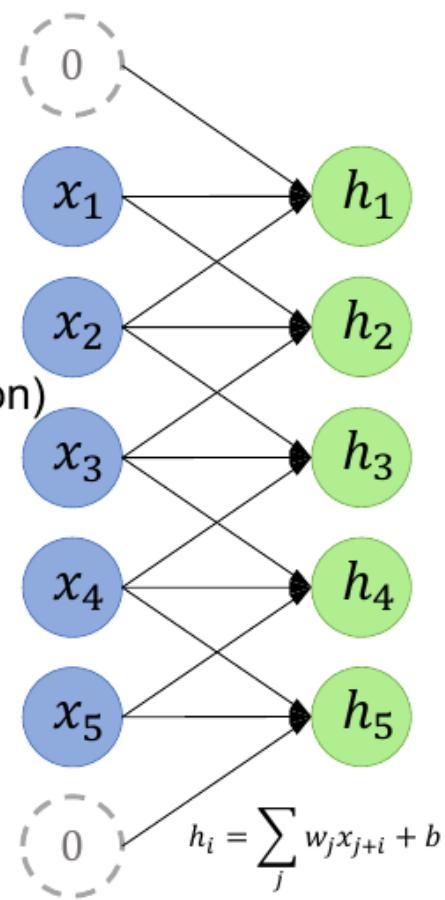
# Neural networks $\Rightarrow$ Convolutional networks



dense connectivity  
vs  
sparse connectivity

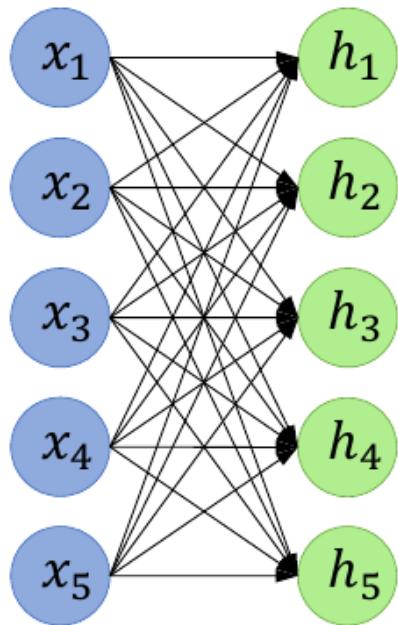
In our example:  
5x5 multiplications (naïve implementation)  
vs  
5x3 multiplications  
Sparse connectivity scales better  
e.g. 25x25 vs 25x3

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$



$$h_i = \sum_j w_j x_{j+i} + b$$

# Neural networks ⇒ Convolutional networks

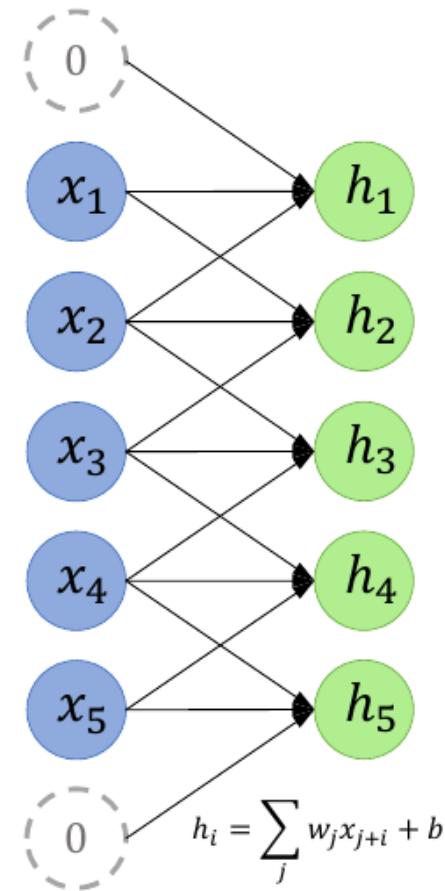


$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}; h_i = \sum_j w_{ij}x_j + b_i$$

unshared vs shared weights:

In our example:  
5x5 vs 3 weights

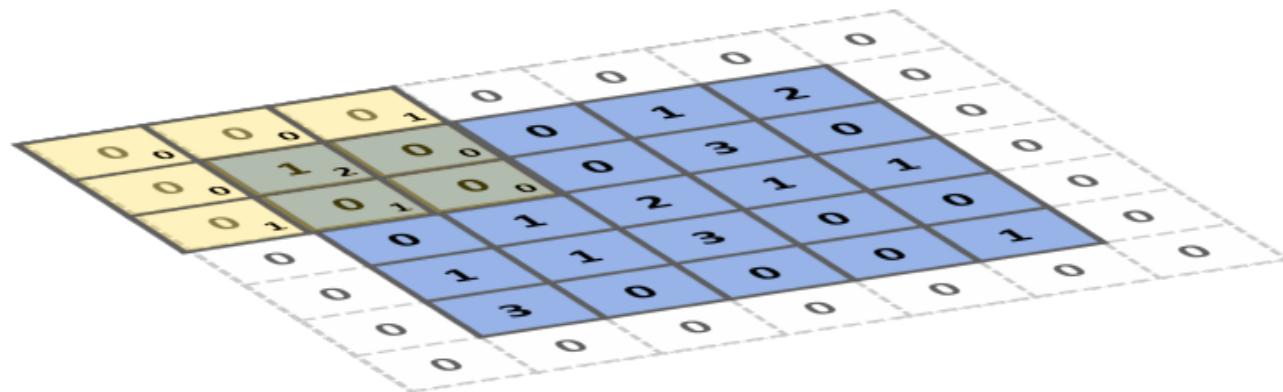
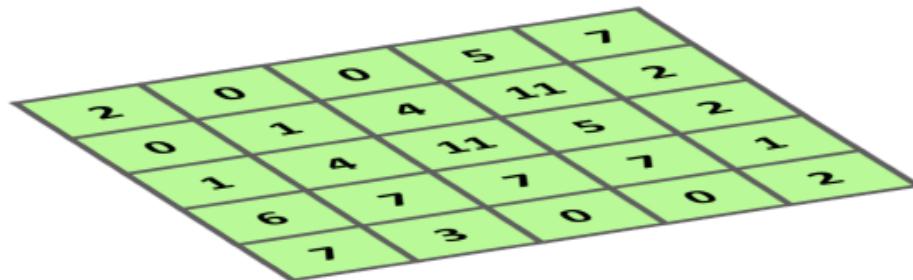
shared weights scale way better:  
e.g. 25x25 vs 3



$$h_i = \sum_j w_j x_{j+i} + b$$

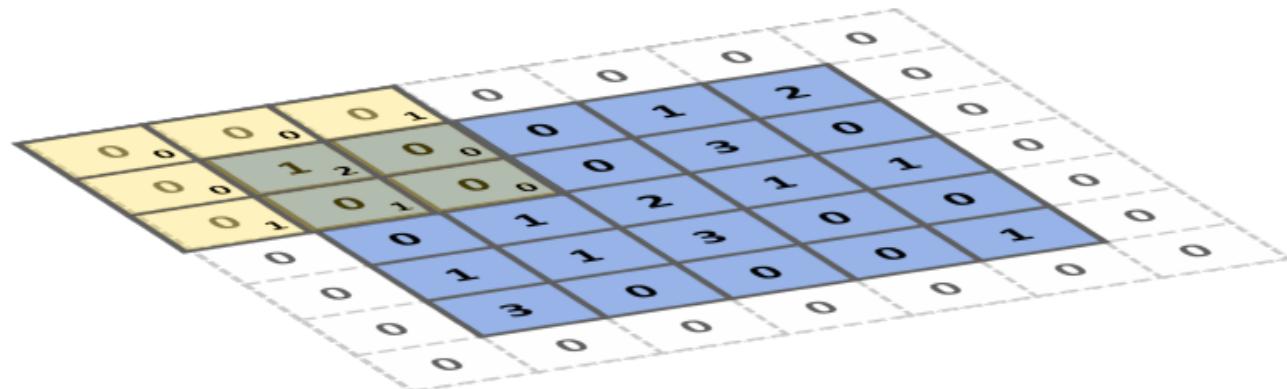
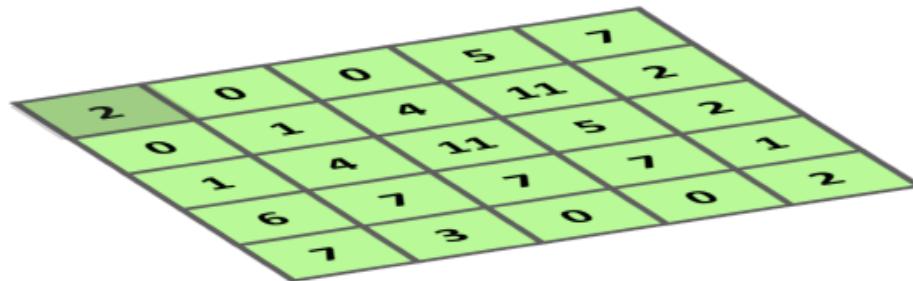
## The effect of different strides

Stride: distance between two consecutive positions of the kernel



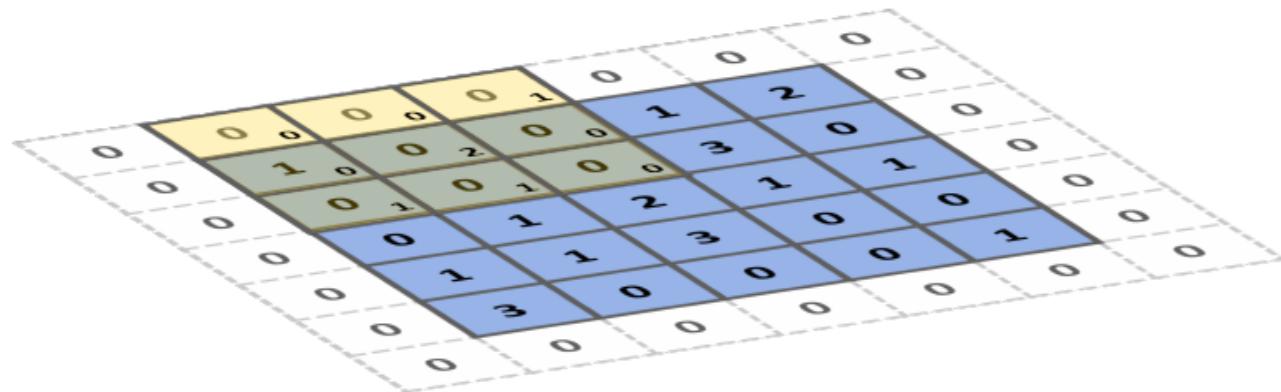
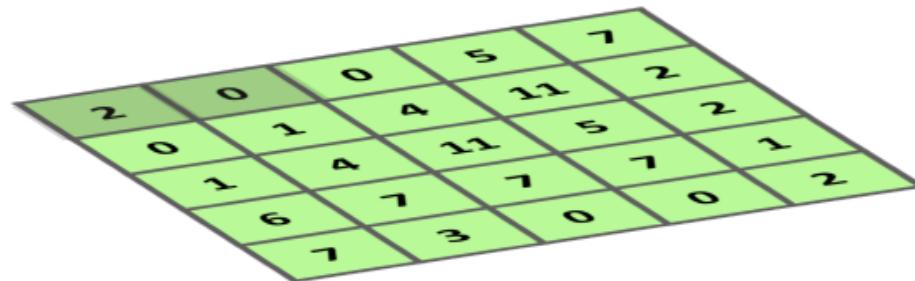
## The effect of different strides

Stride: 1x1



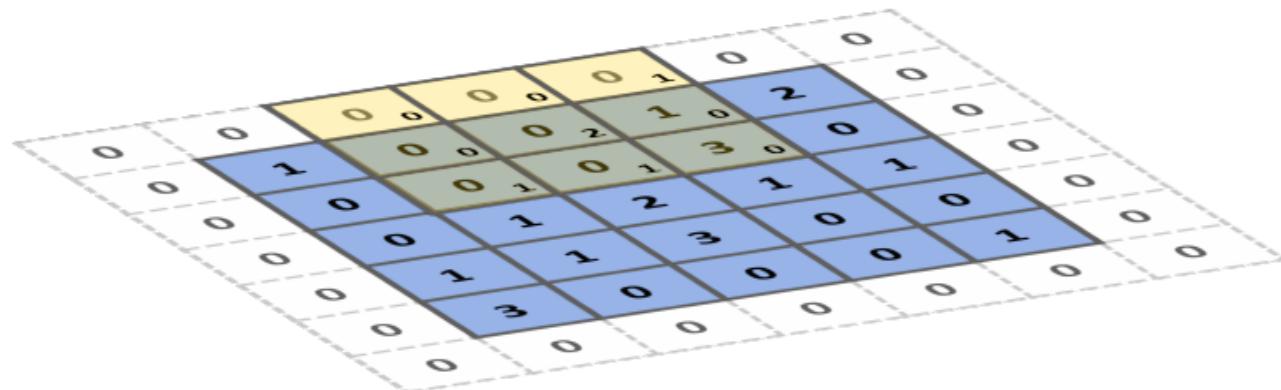
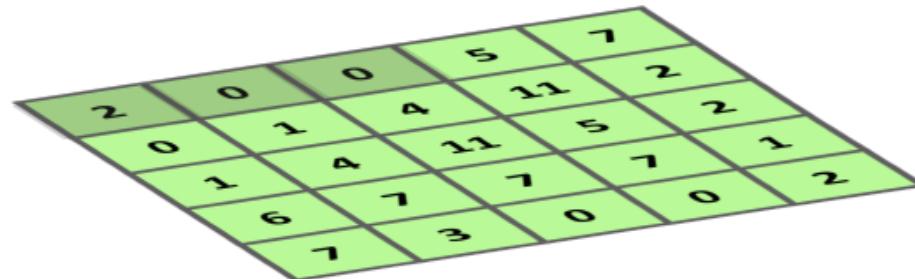
## The effect of different strides

Stride: 1x1



## The effect of different strides

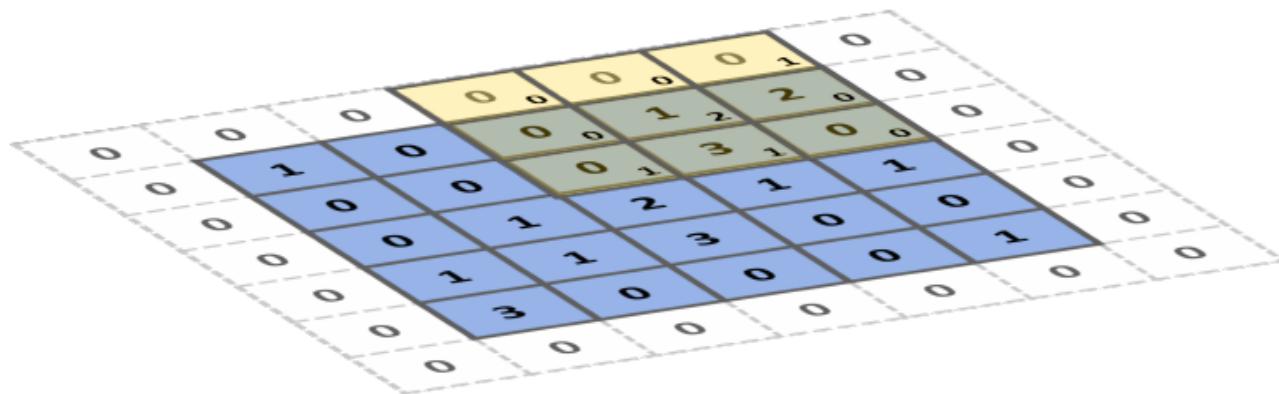
Stride: 1x1



# The effect of different strides

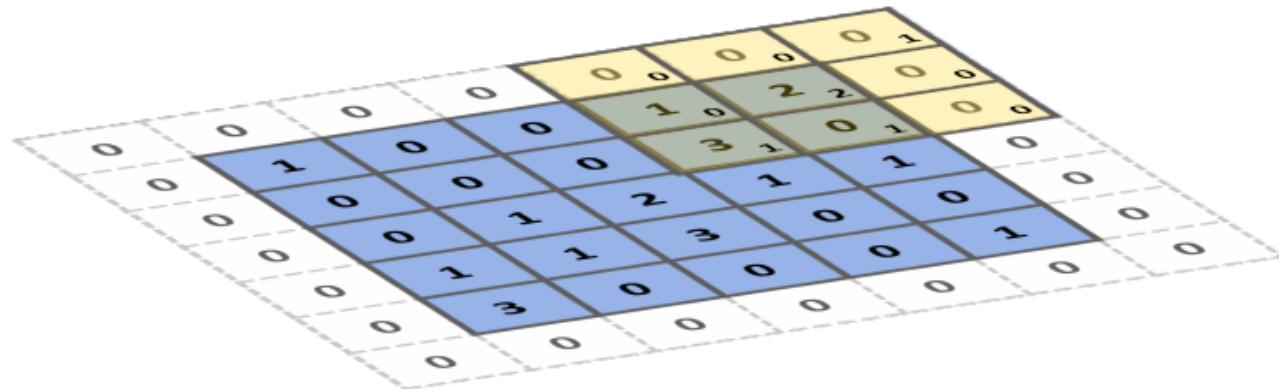
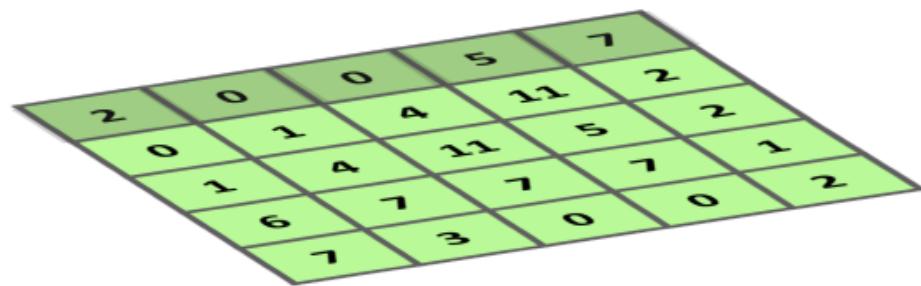
Stride: 1x1

2	0	0	0	5	7	2
0	1	4	11	5	2	1
1	4	11	5	7	1	2
6	7	7	0	0	0	2
7	3	0	0	0	0	0



# The effect of different strides

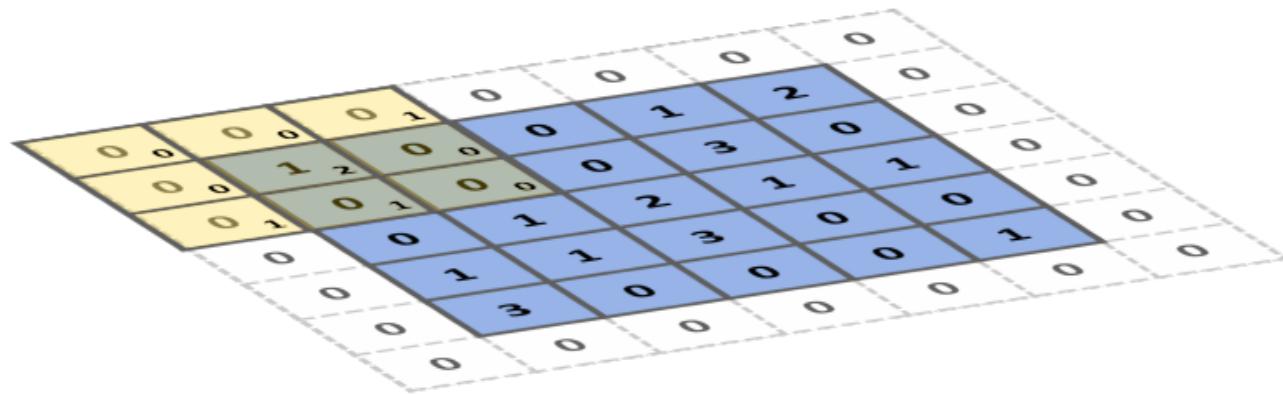
Stride: 1x1



# The effect of different strides

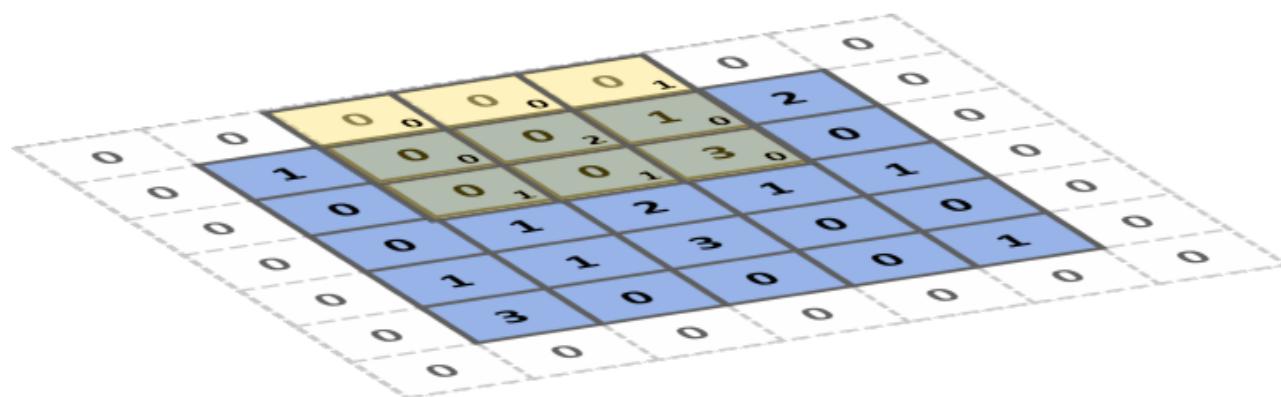
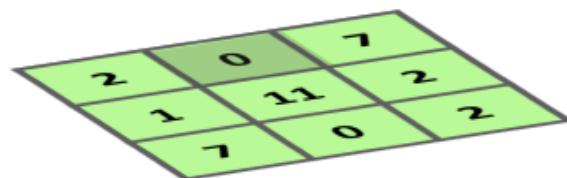
Stride: 2x2

2	0	7
1	11	2
7	0	2



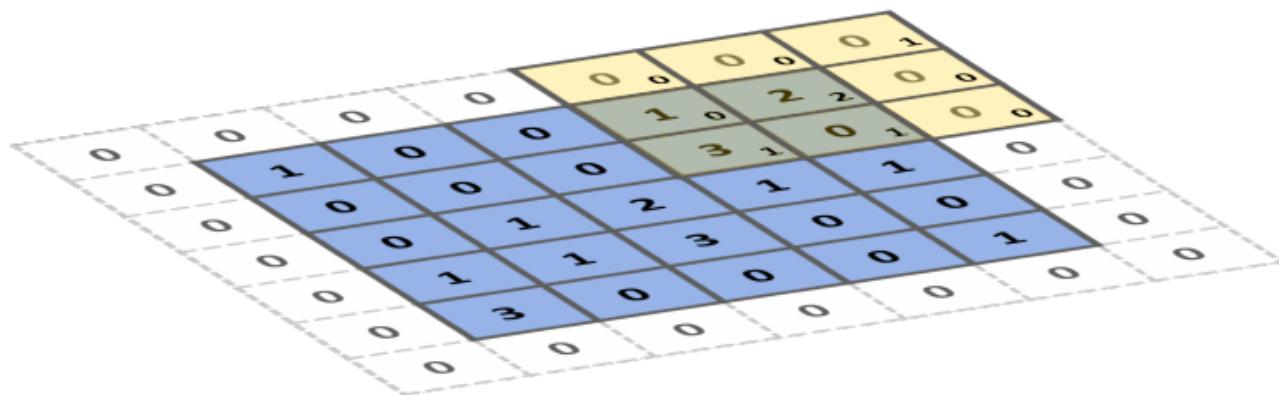
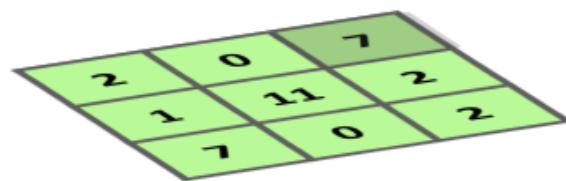
## The effect of different strides

Stride: 2x2



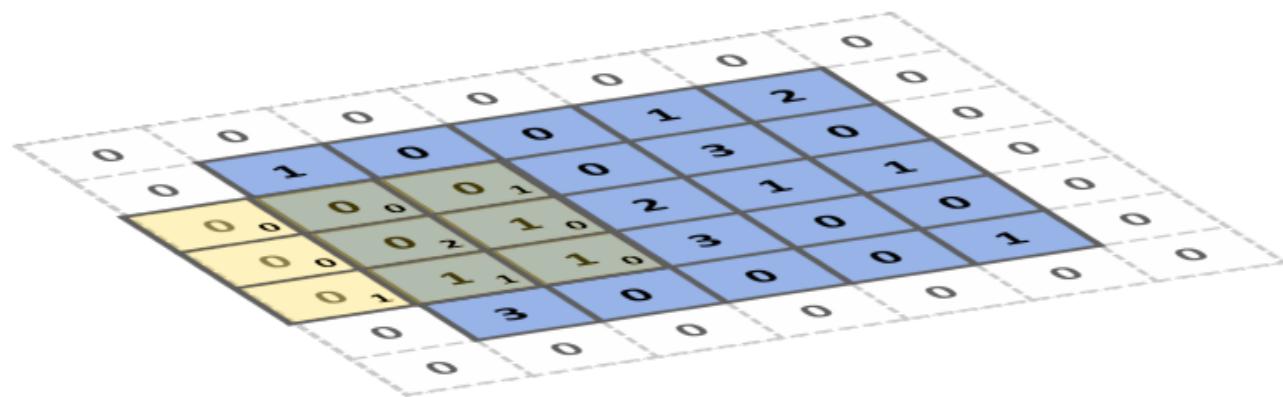
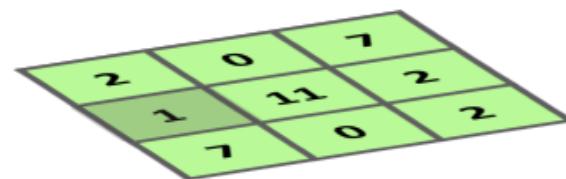
## The effect of different strides

Stride: 2x2



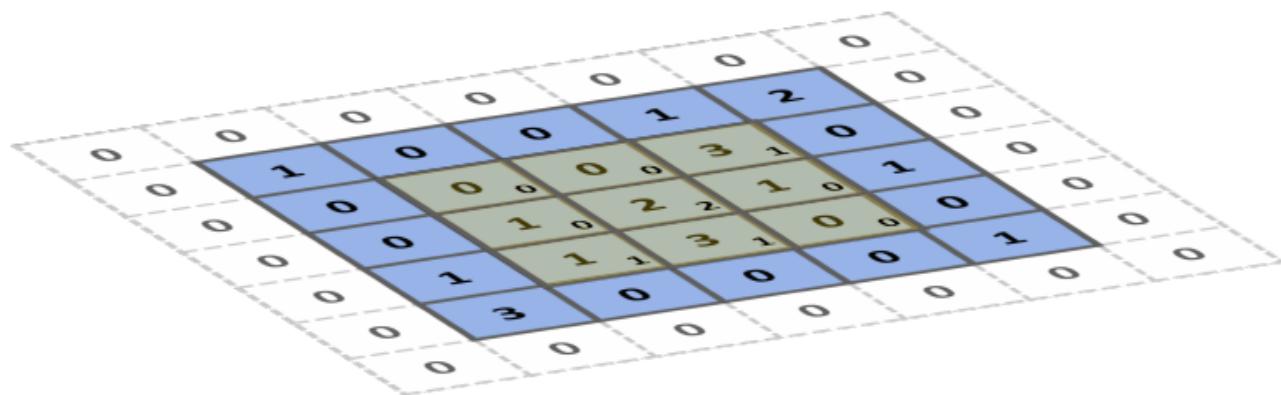
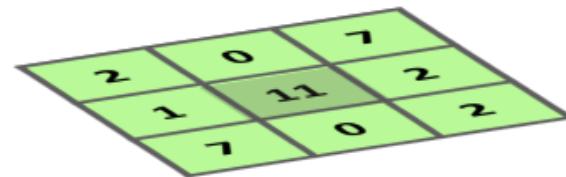
## The effect of different strides

Stride: 2x2



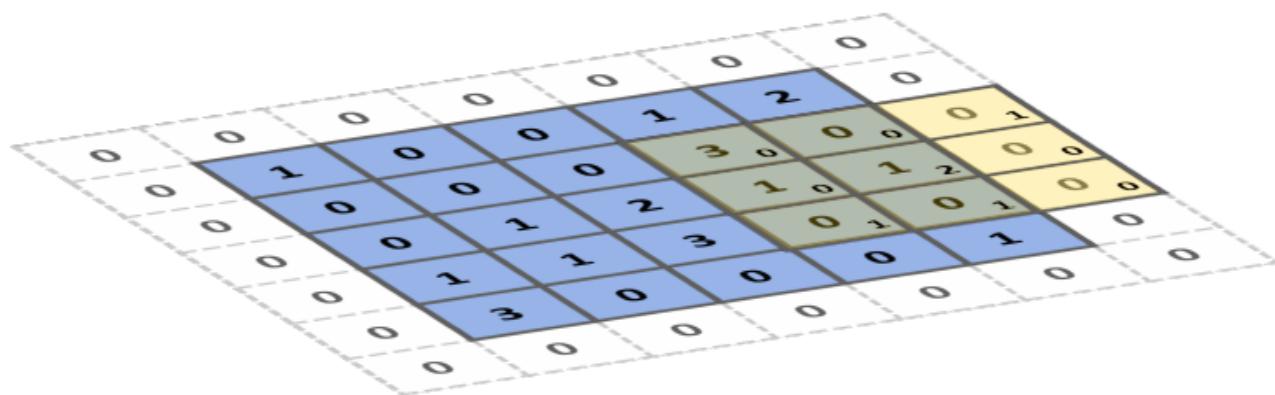
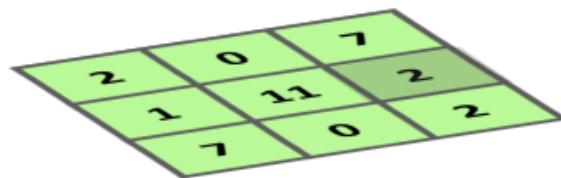
## The effect of different strides

Stride: 2x2



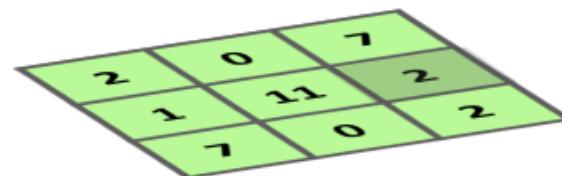
## The effect of different strides

Stride: 2x2



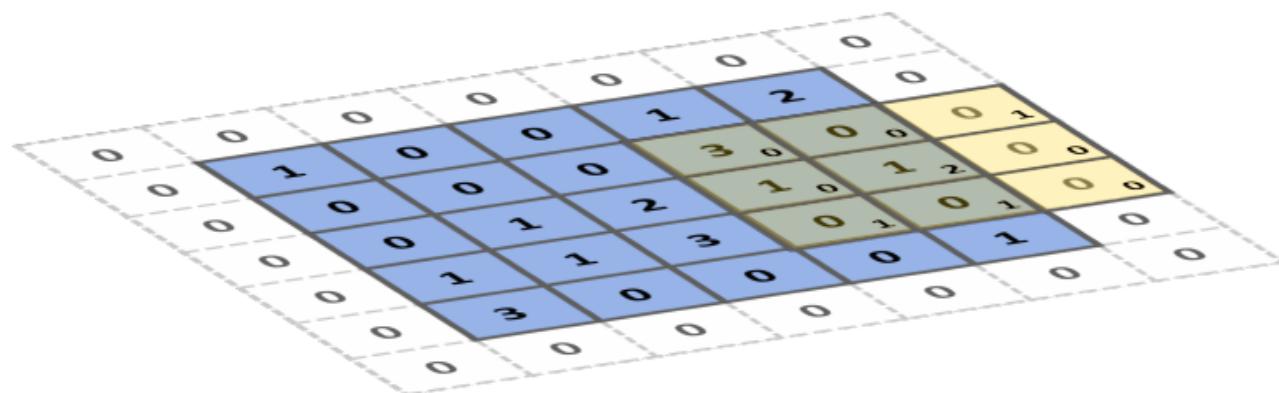
## The effect of different strides

Stride: 2x2



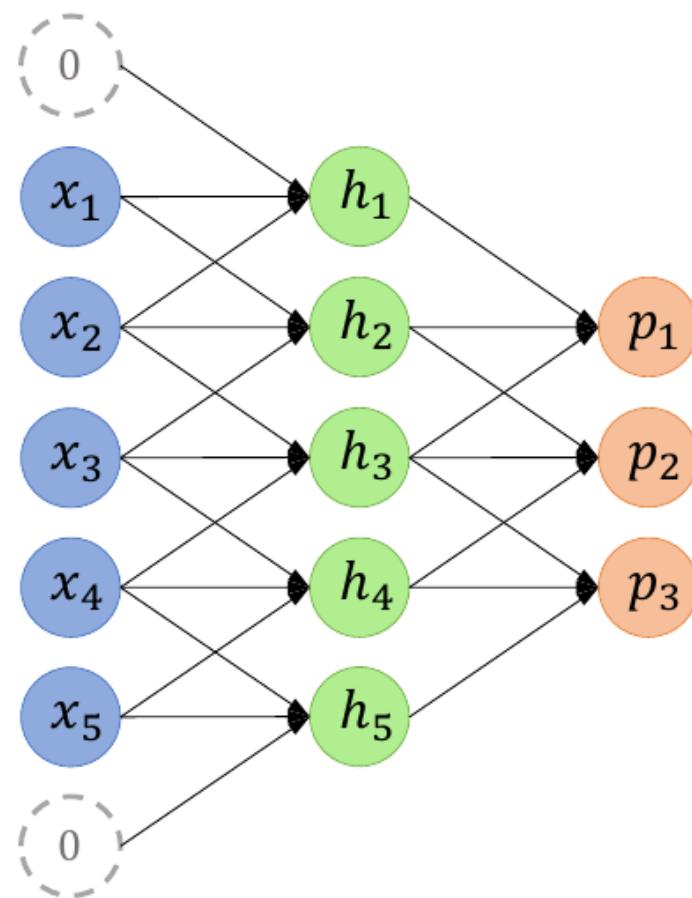
## Why use stride > 1?

- Reduce redundancy
  - Compress feature map



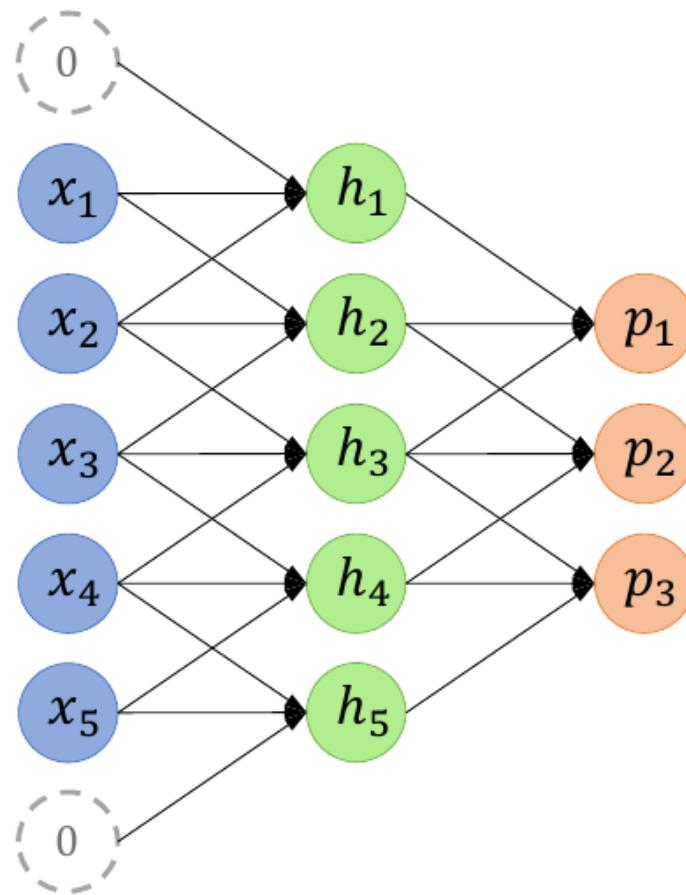
# Pooling

Operation to aggregate or “summarize” sub-region of input.



# Pooling

Operation to aggregate or “summarize” sub-region of input.



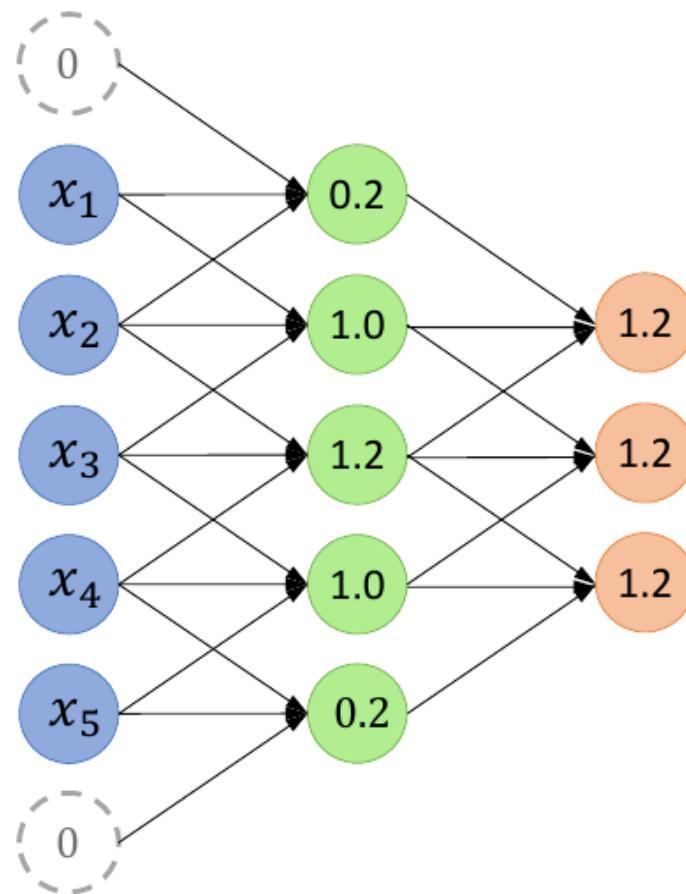
$$p = f(\mathbf{a})$$

$$f_{max}(\mathbf{a}) = \max_i(a_i)$$

$$f_{avg}(\mathbf{a}) = \frac{1}{N} \sum_{i=1}^N a_i$$

# Pooling

Operation to aggregate or “summarize” sub-region of input.



$$p = f(\mathbf{a})$$

$$f_{max}(\mathbf{a}) = \max_i(a_i)$$

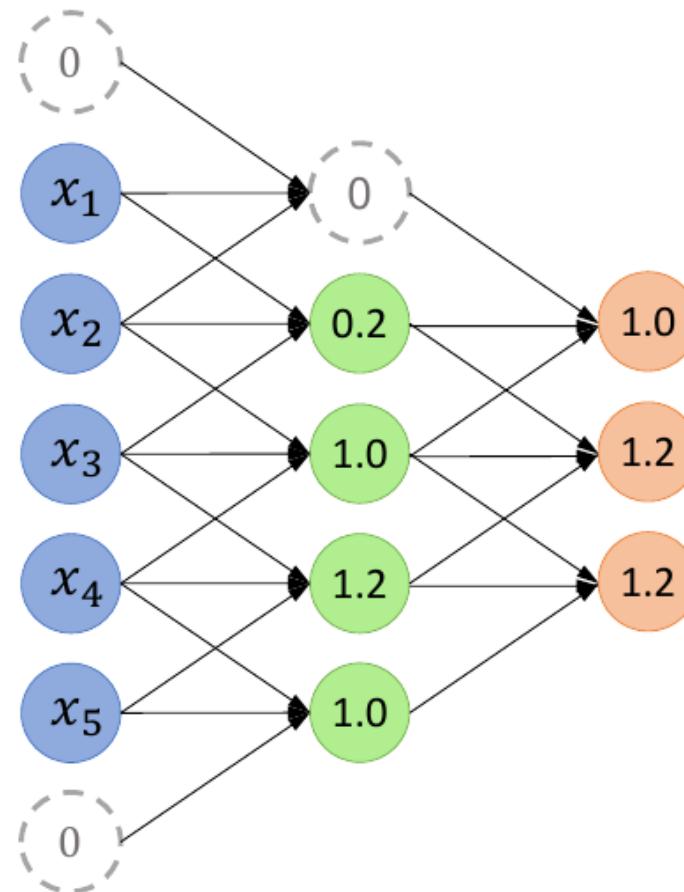
$$f_{avg}(\mathbf{a}) = \frac{1}{N} \sum_{i=1}^N a_i$$

# Pooling

Operation to aggregate or “summarize” sub-region of input.

Shift input by 1 position:

5/5 inputs change but only 1/3 pooled outputs



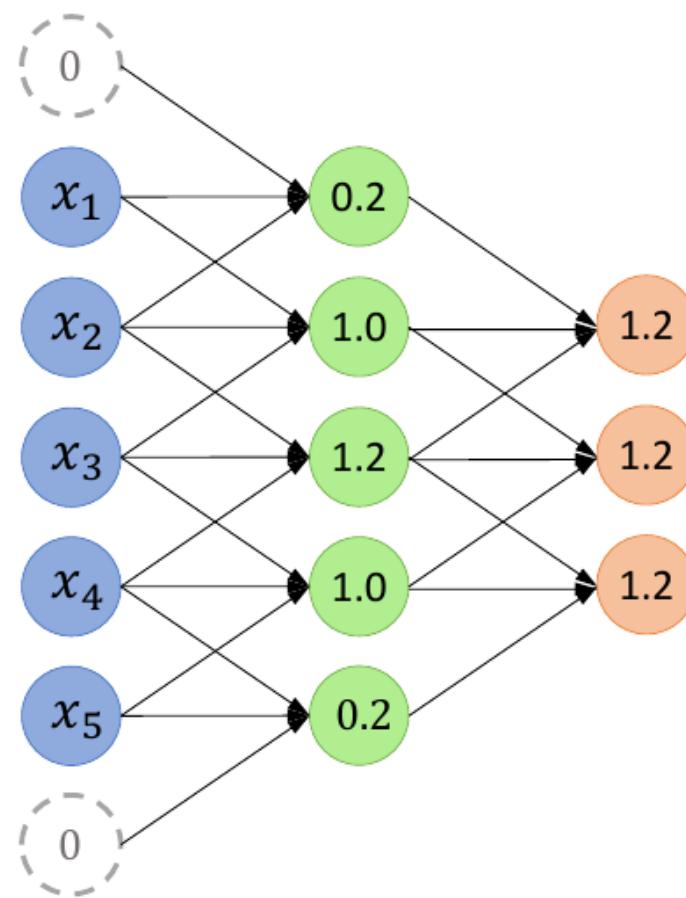
$$p = f(\mathbf{a})$$
$$f_{max}(\mathbf{a}) = \max_i(a_i)$$
$$f_{avg}(\mathbf{a}) = \frac{1}{N} \sum_{i=1}^N a_i$$

# Pooling

Operation to aggregate or “summarize” sub-region of input.

Shift input by 1 position:

5/5 inputs change but only 1/3 pooled outputs



$$p = f(\mathbf{a})$$

$$f_{max}(\mathbf{a}) = \max_i(a_i)$$

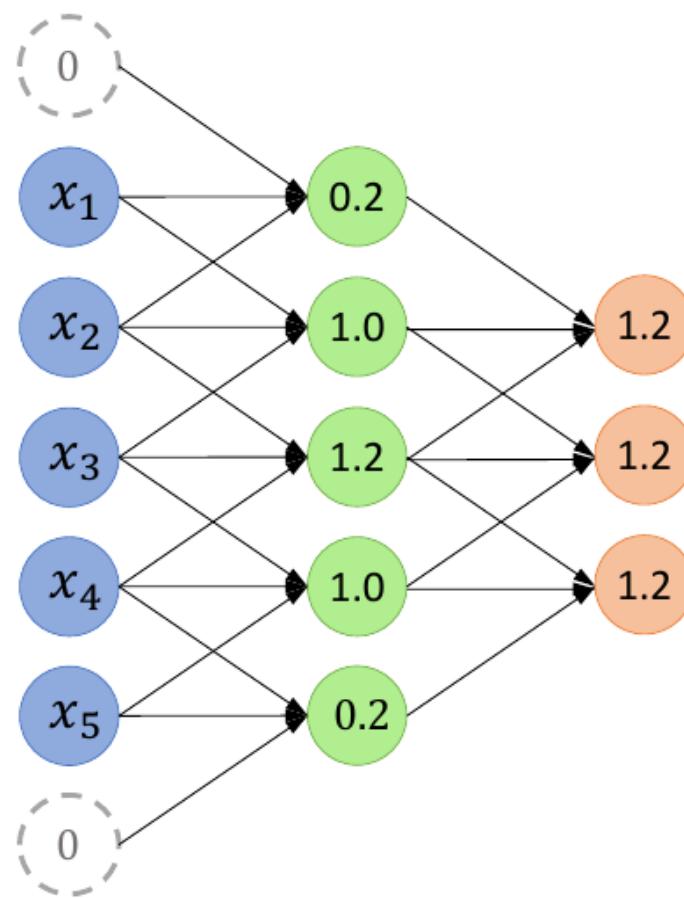
$$f_{avg}(\mathbf{a}) = \frac{1}{N} \sum_{i=1}^N a_i$$

## Pooling

Operation to aggregate or “summarize” sub-region of input.

Shift input by 1 position:

5/5 inputs change but only 1/3 pooled outputs



$$p = f(\mathbf{a})$$

$$f_{max}(\mathbf{a}) = \max_i(a_i)$$

$$f_{avg}(\mathbf{a}) = \frac{1}{N} \sum_{i=1}^N a_i$$

# Pooling

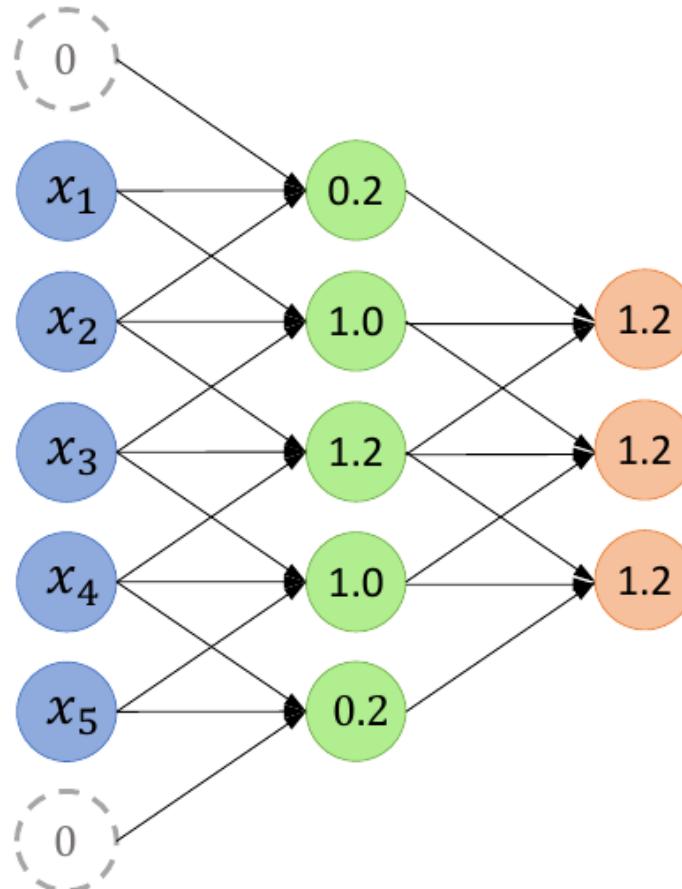
Operation to aggregate or “summarize” sub-region of input.

Shift input by 1 position:

5/5 inputs change but only 1/3 pooled outputs

Why is pooling useful?

- Adds **translation invariance** (useful when we care about “what” more than “where”)
- Can be used to **compress signal** (useful to improve computational efficiency)



$$p = f(\mathbf{a})$$

$$f_{max}(\mathbf{a}) = \max_i(a_i)$$

$$f_{avg}(\mathbf{a}) = \frac{1}{N} \sum_{i=1}^N a_i$$

# Pooling

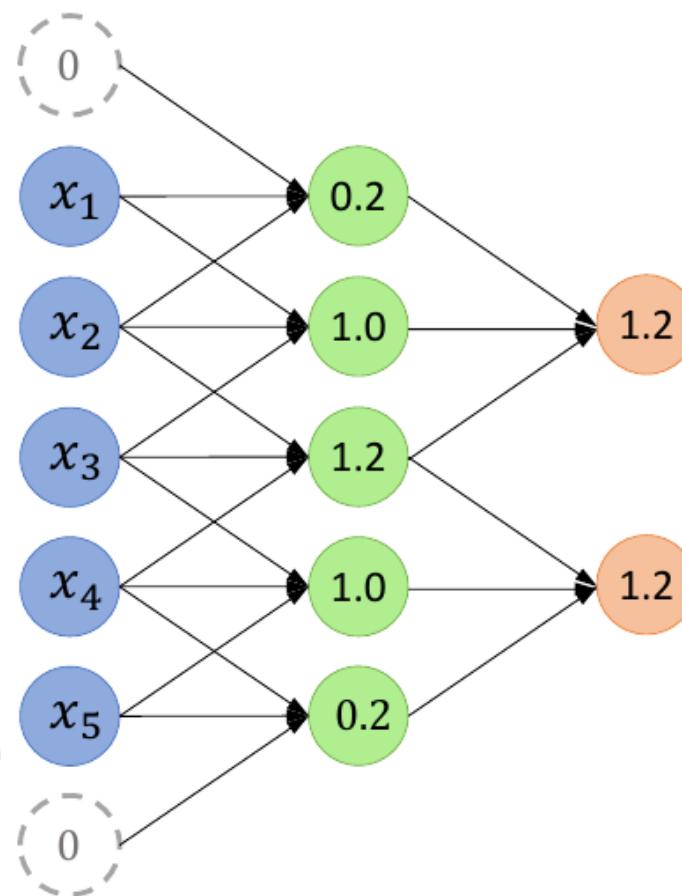
Operation to aggregate or “summarize” sub-region of input.

Shift input by 1 position:

5/5 inputs change but only 1/3 pooled outputs

Why is pooling useful?

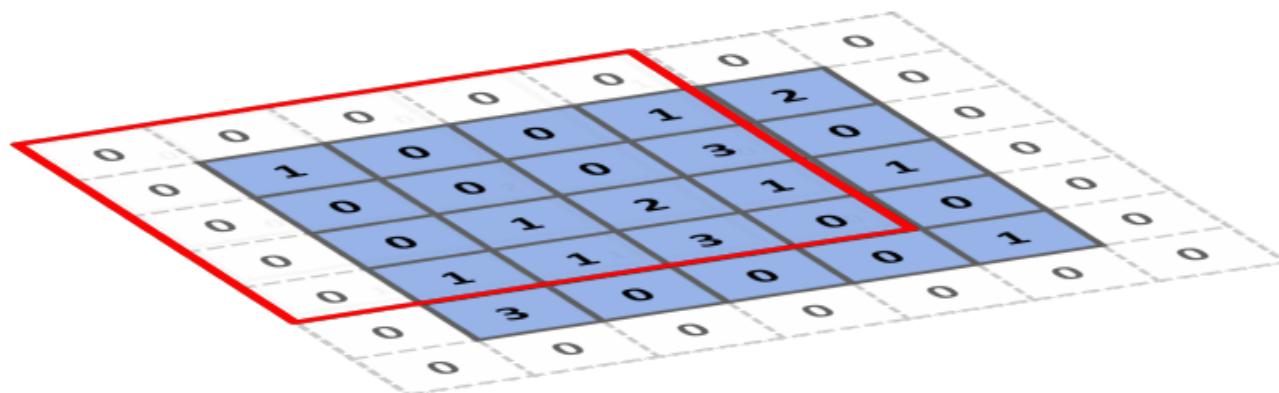
- Adds **translation invariance** (useful when we care about “what” more than “where”)
- Can be used to **compress signal** (useful to improve computational efficiency)



$$p = f(\mathbf{a})$$
$$f_{max}(\mathbf{a}) = \max_i(a_i)$$
$$f_{avg}(\mathbf{a}) = \frac{1}{N} \sum_{i=1}^N a_i$$

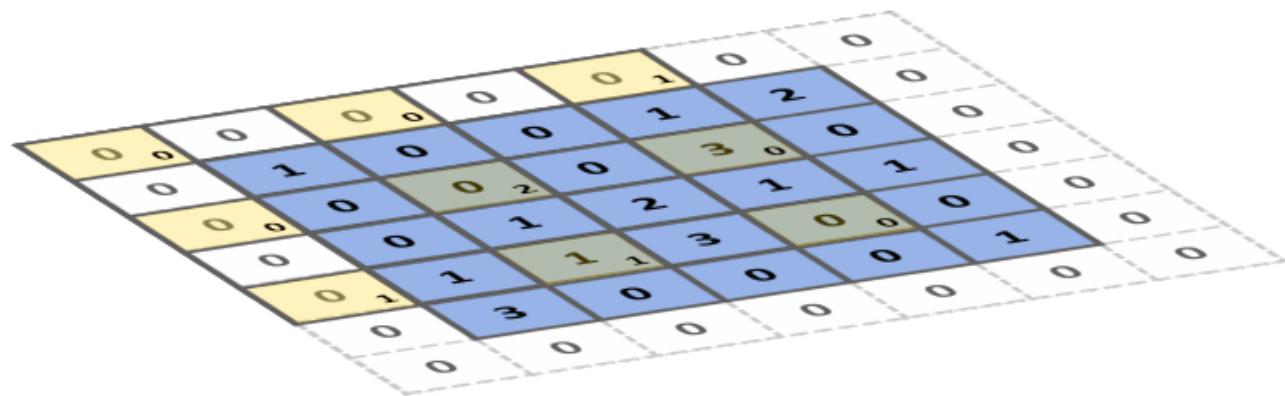
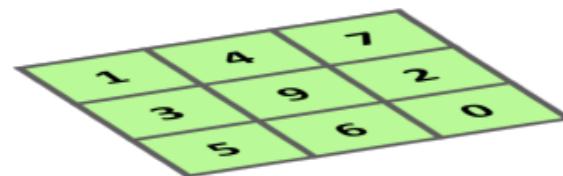
# Dilated convolutions

When you want to increase receptive field without increasing computation...



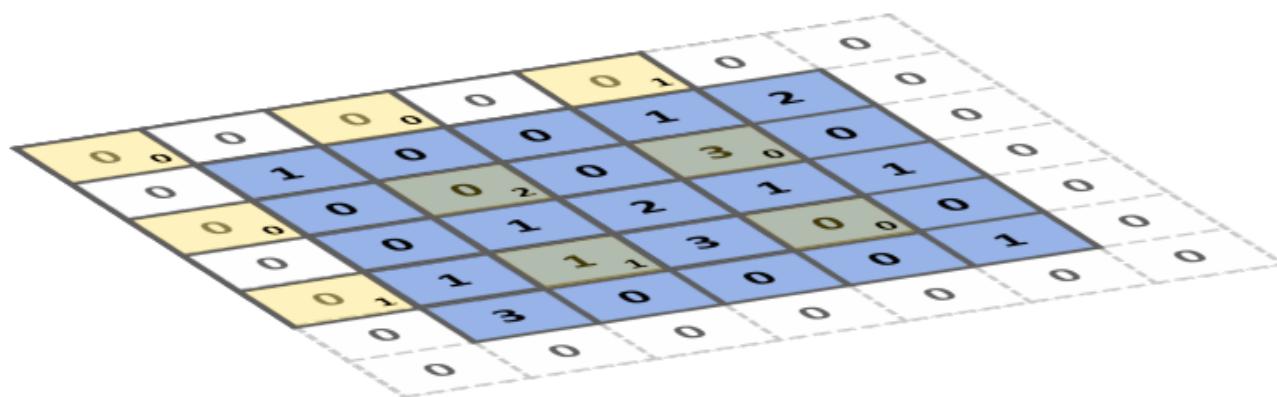
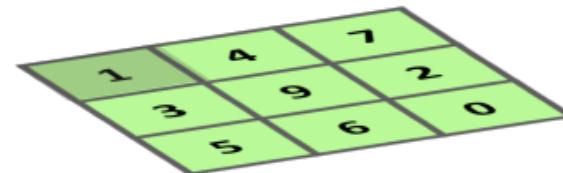
## Dilated convolutions

When you want to increase receptive field without increasing computation...



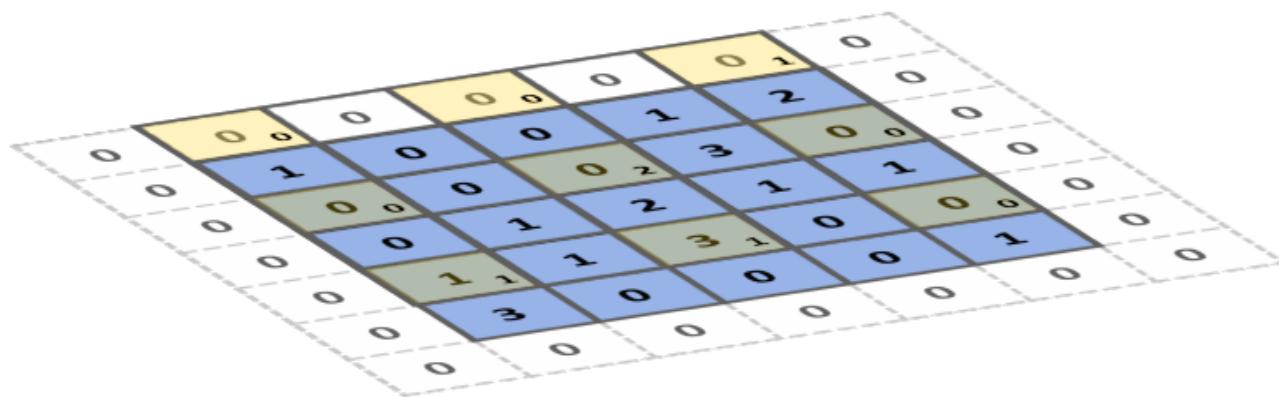
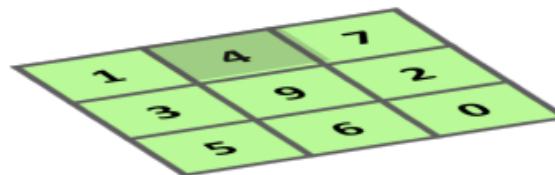
## Dilated convolutions

When you want to increase receptive field without increasing computation...



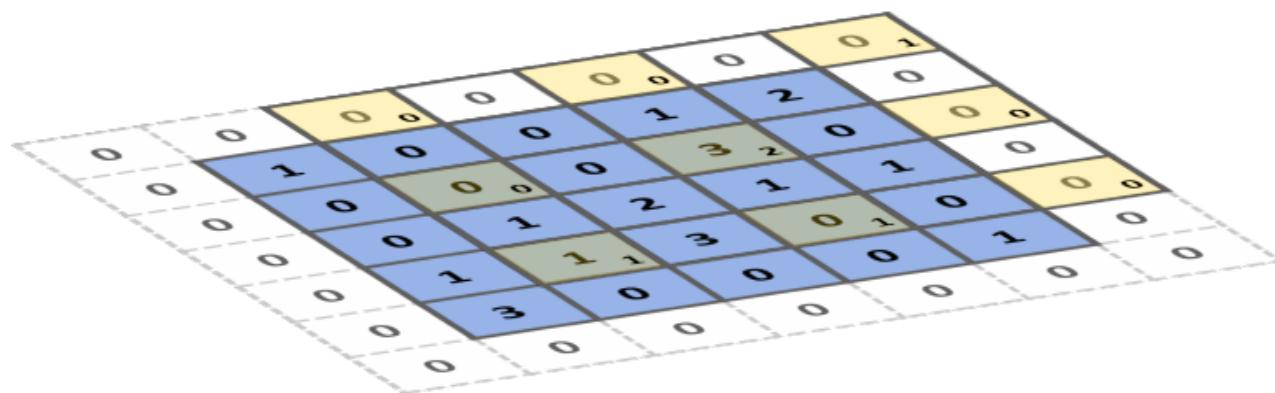
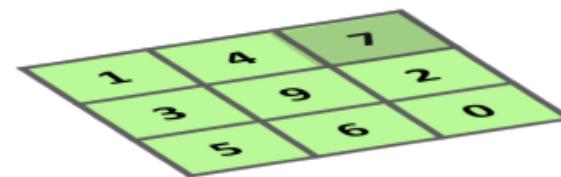
# Dilated convolutions

When you want to increase receptive field without increasing computation...



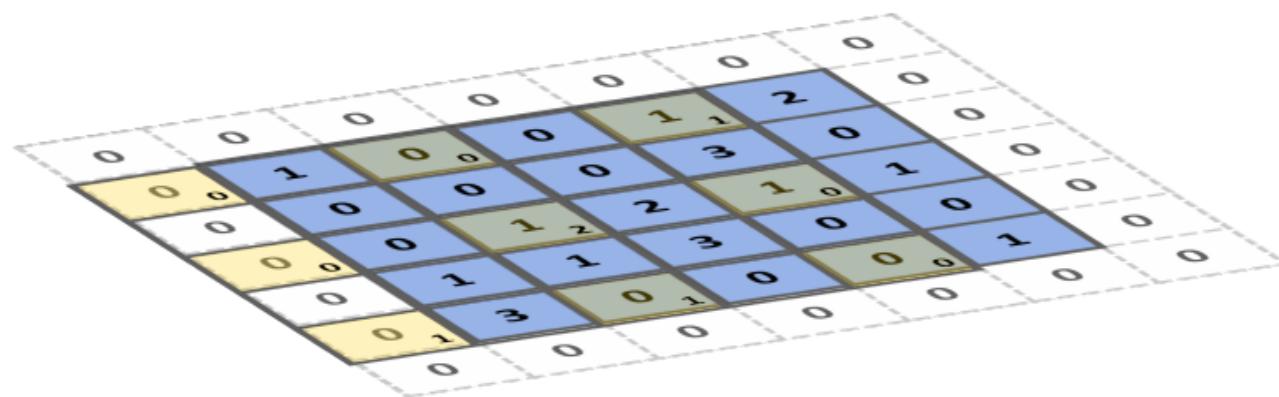
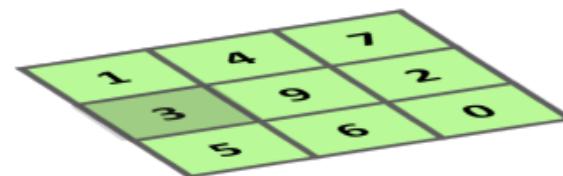
# Dilated convolutions

When you want to increase receptive field without increasing computation...



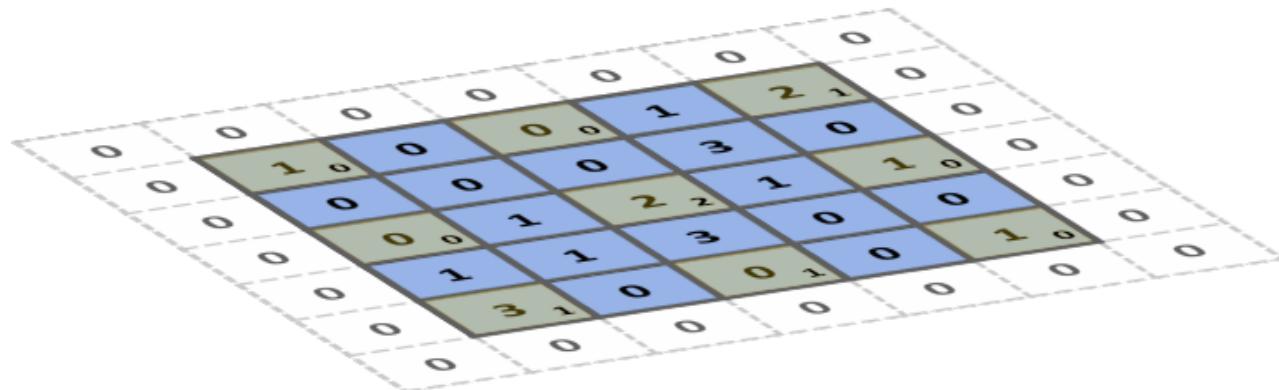
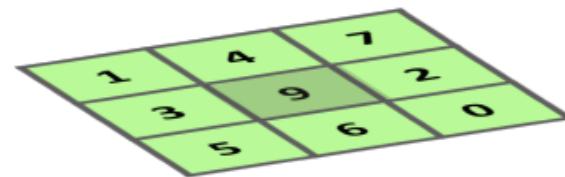
## Dilated convolutions

When you want to increase receptive field without increasing computation...



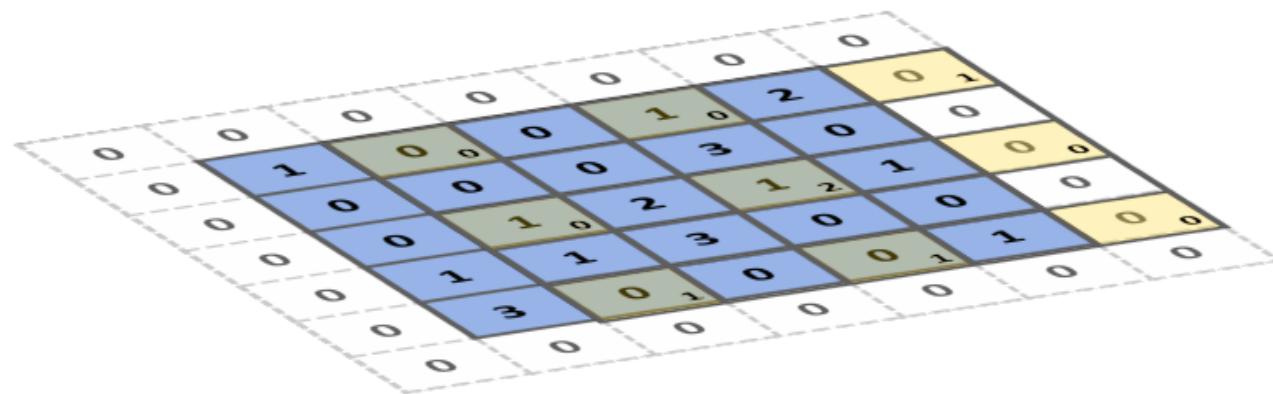
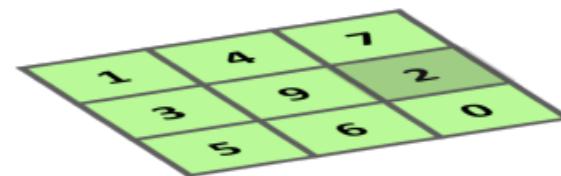
## Dilated convolutions

When you want to increase receptive field without increasing computation...



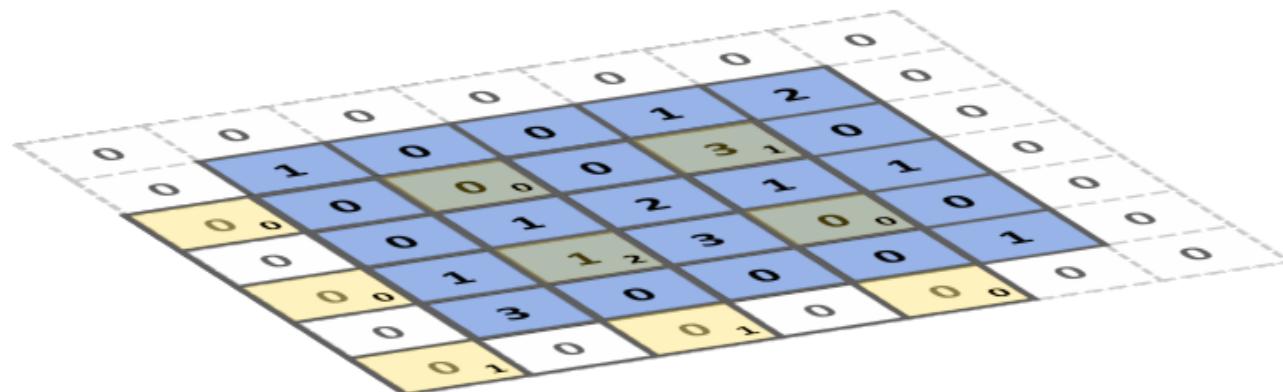
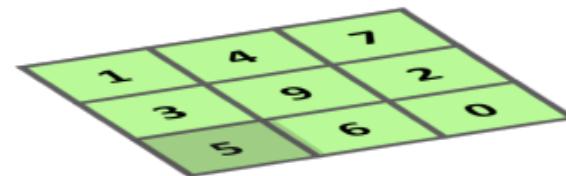
## Dilated convolutions

When you want to increase receptive field without increasing computation...



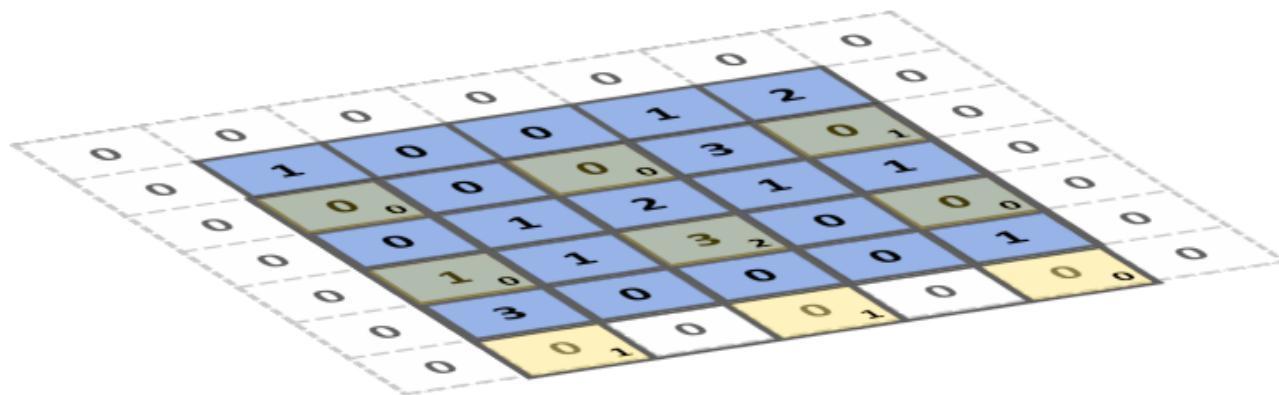
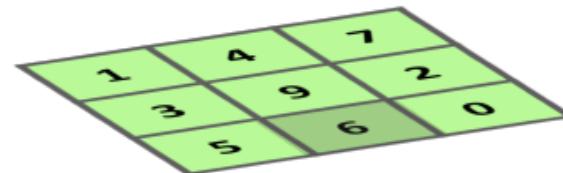
## Dilated convolutions

When you want to increase receptive field without increasing computation...



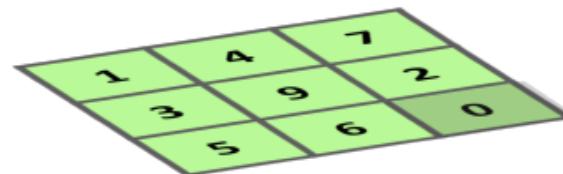
# Dilated convolutions

When you want to increase receptive field without increasing computation...

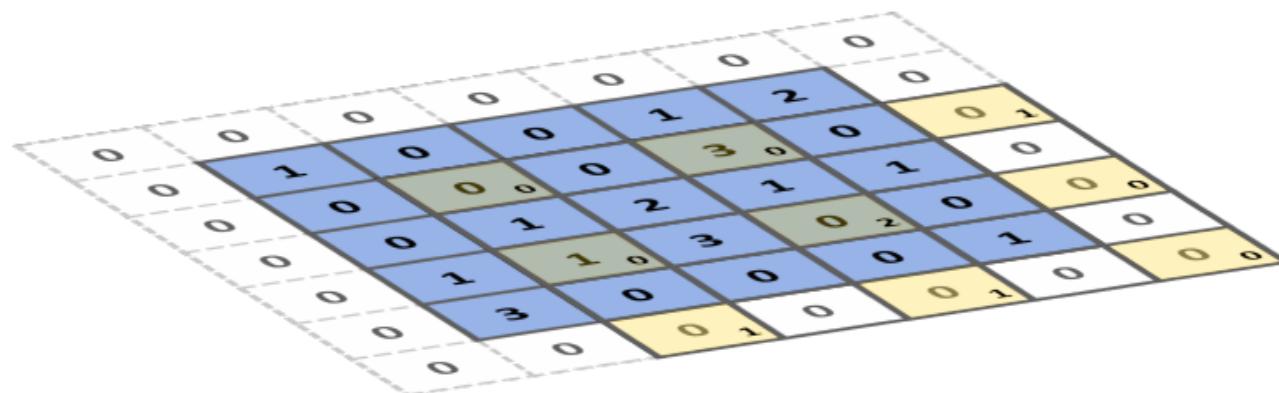


# Dilated convolutions

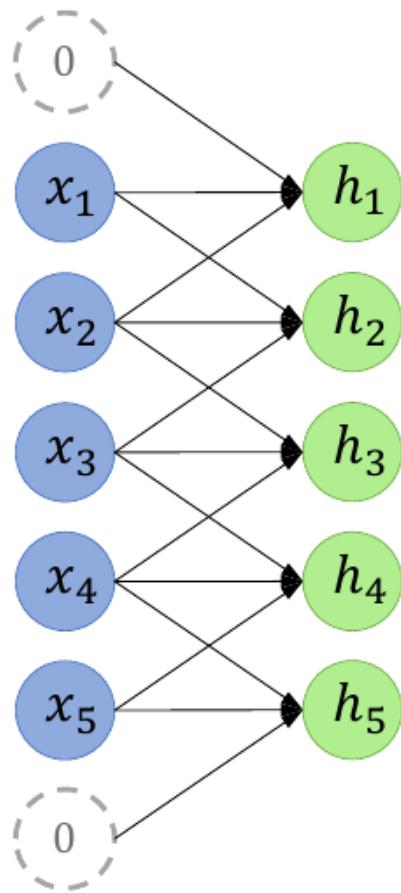
When you want to increase receptive field without increasing computation...



Also known as “à trous” (with holes) convolutions

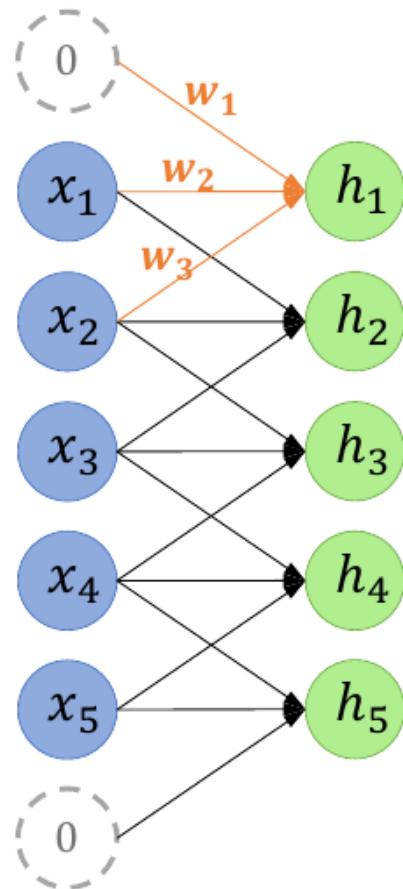


# Tiled convolutions



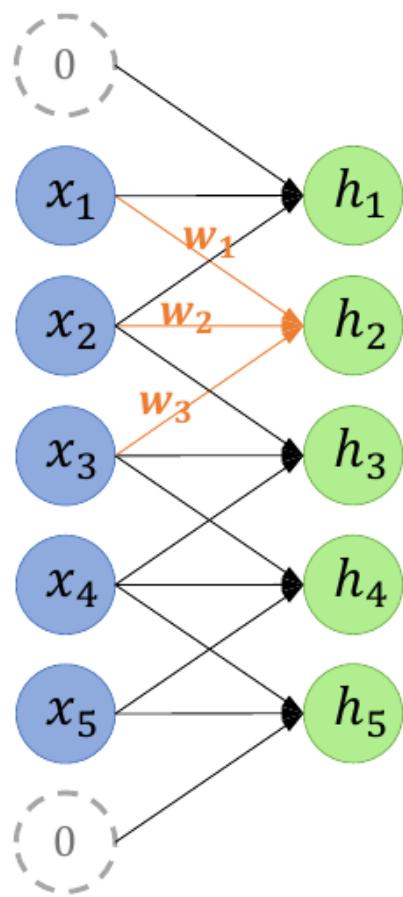
Reminder: regular convolutions slide same filter across signal

## Tiled convolutions



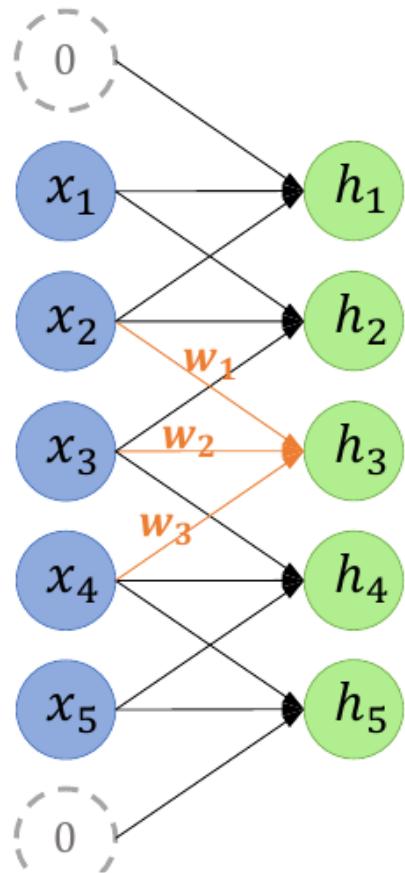
Reminder: regular convolutions slide same filter across signal

## Tiled convolutions



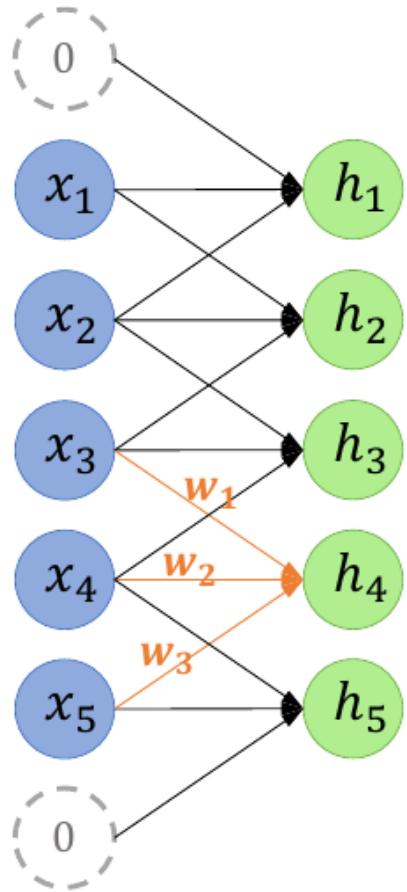
Reminder: regular convolutions slide same filter across signal

## Tiled convolutions



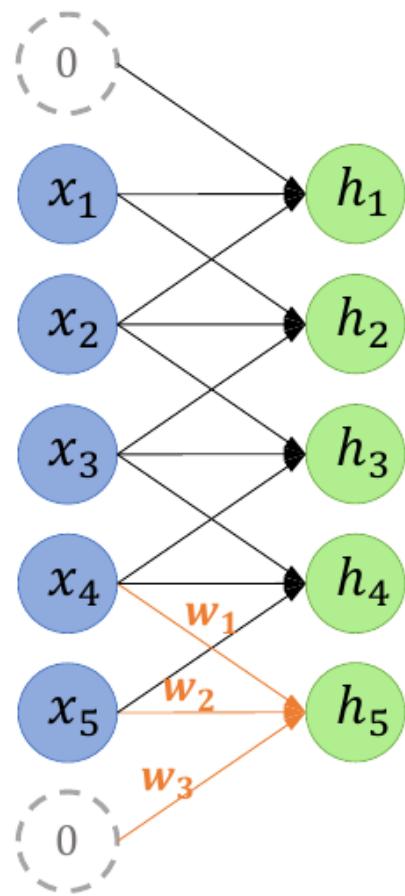
Reminder: regular convolutions slide same filter across signal

# Tiled convolutions



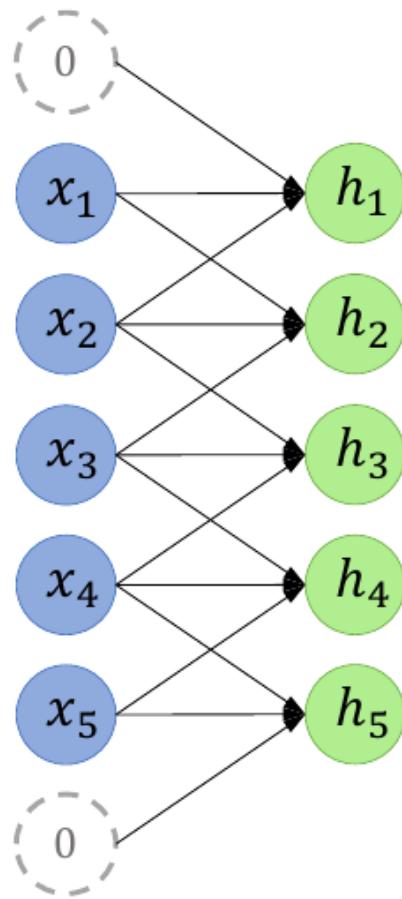
Reminder: regular convolutions slide same filter across signal

# Tiled convolutions

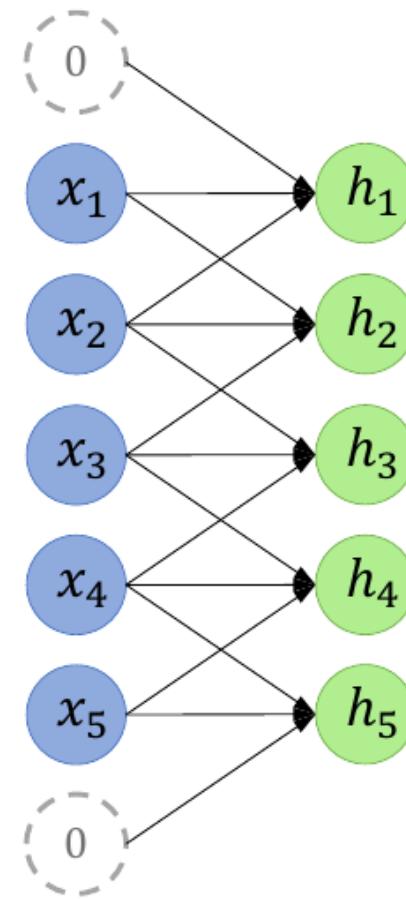


Reminder: regular convolutions slide same filter across signal

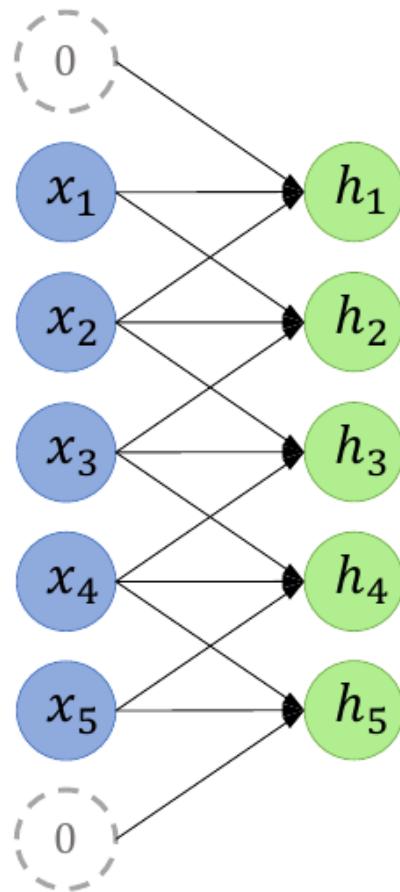
# Tiled convolutions



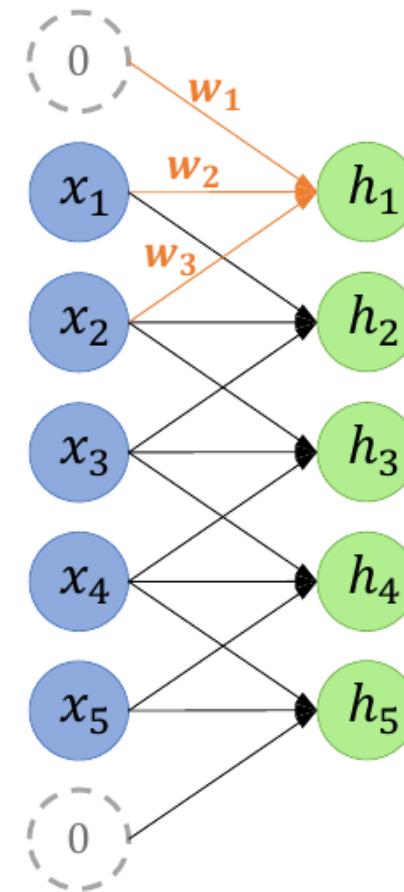
Cycle through different filters



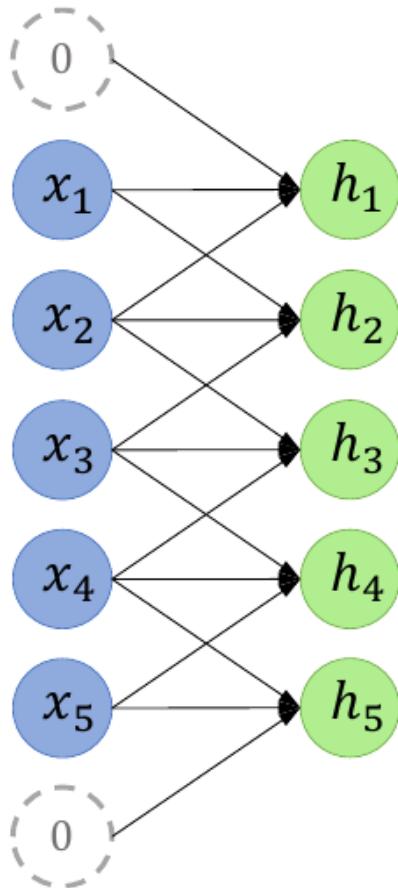
# Tiled convolutions



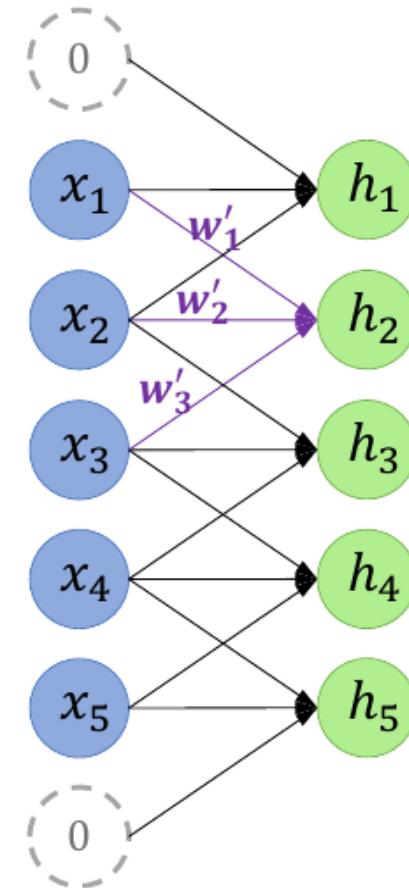
Cycle through different filters



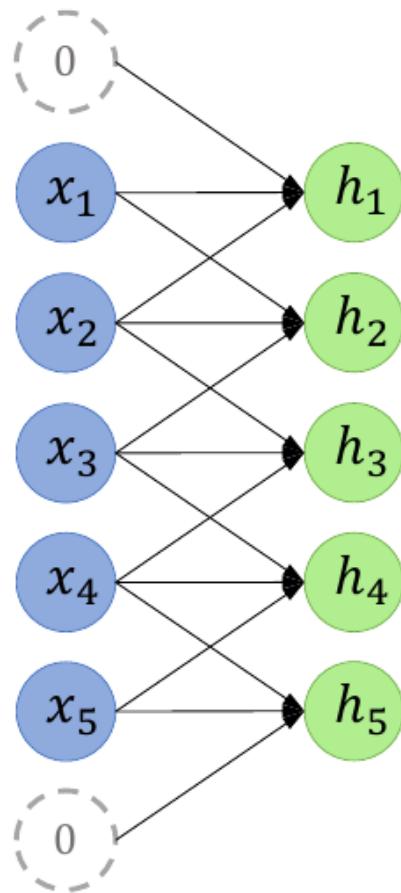
# Tiled convolutions



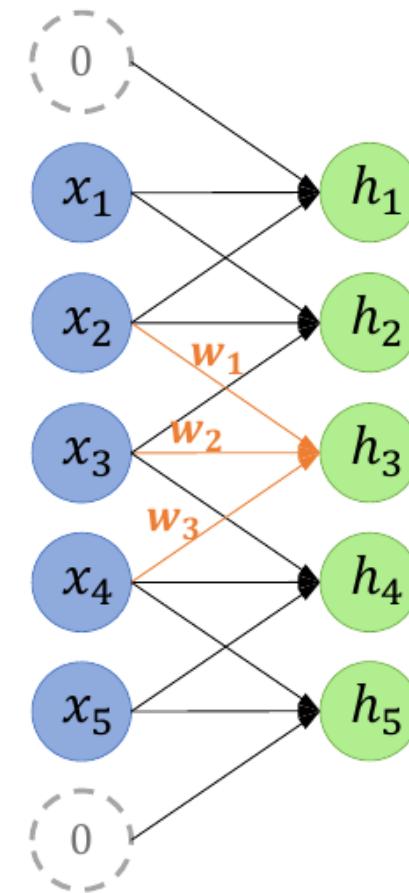
Cycle through different filters



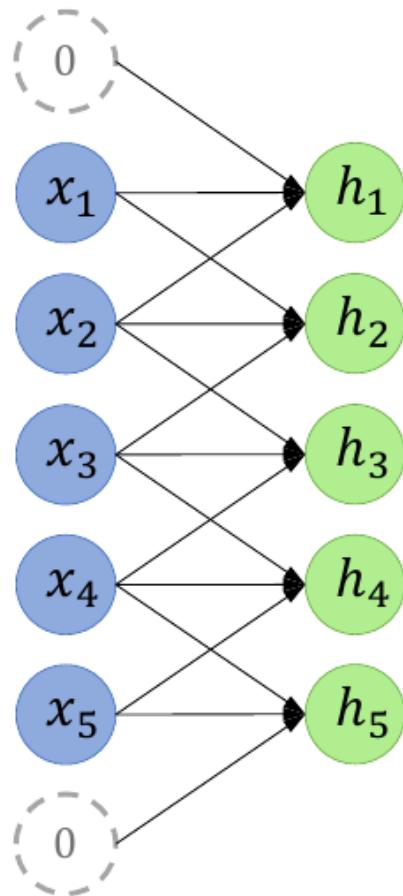
# Tiled convolutions



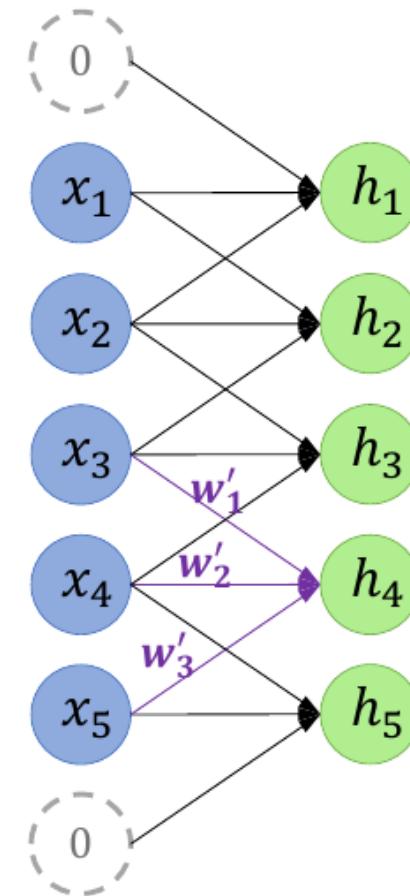
Cycle through different filters



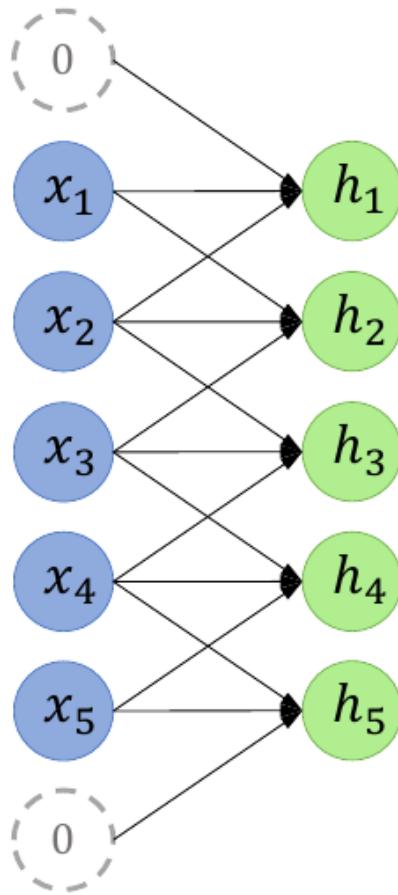
# Tiled convolutions



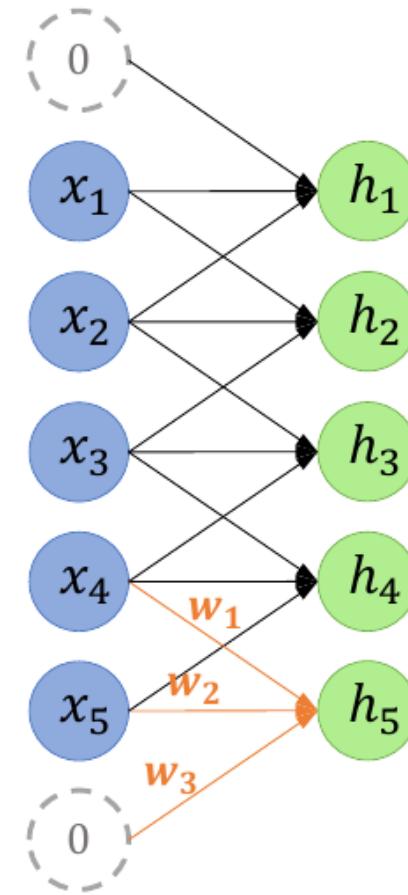
Cycle through different filters



# Tiled convolutions



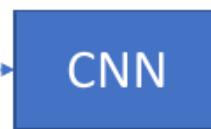
Cycle through different filters



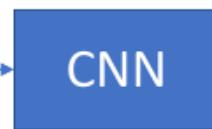
# Tiled convolutions

Why is this useful?

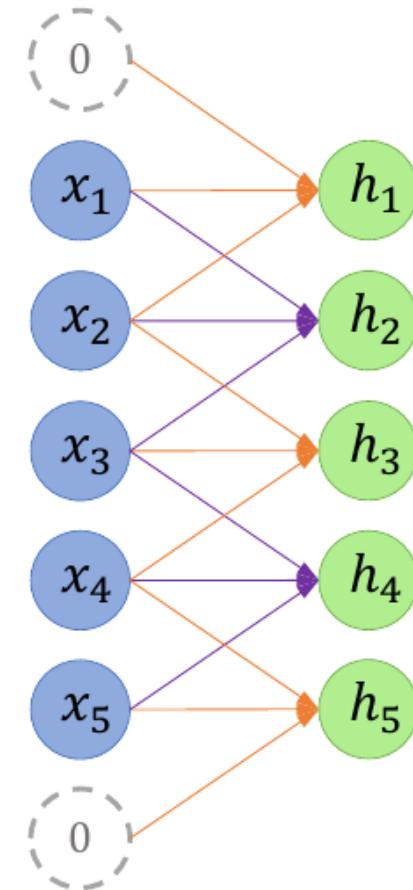
Can capture other types of invariance (e.g. rotation)



“butterfly”



“butterfly”



## ACKNOWLEDGEMENTS

Adapted version of the DeepLearning Indaba 2018