RATT

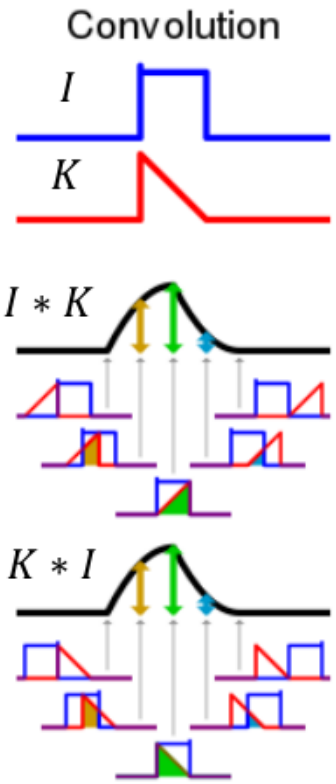
RHODES UNIVERSITY
Where leaders learn

CONVOLUTIONAL NETWORKS

Marcellin Atemkeng
(Rhodes University)

SKA AFRICA
SQUARE KILOMETRE ARRAY

Discrete convolutions



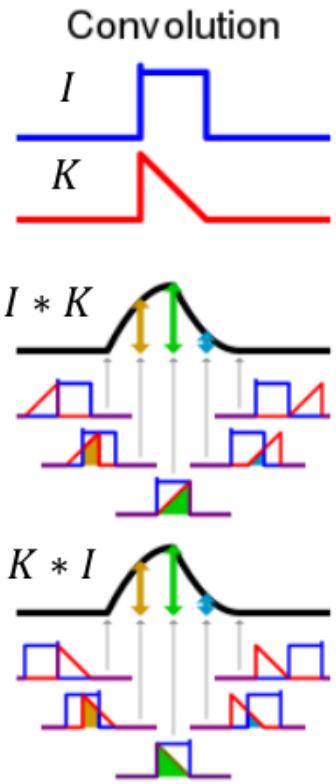
We'll focus on 1 & 2-D signals but signal dimensionality is arbitrary (usually 1,2,3,4-D)

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

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

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

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:

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

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

Convolution vs Cross-correlation

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

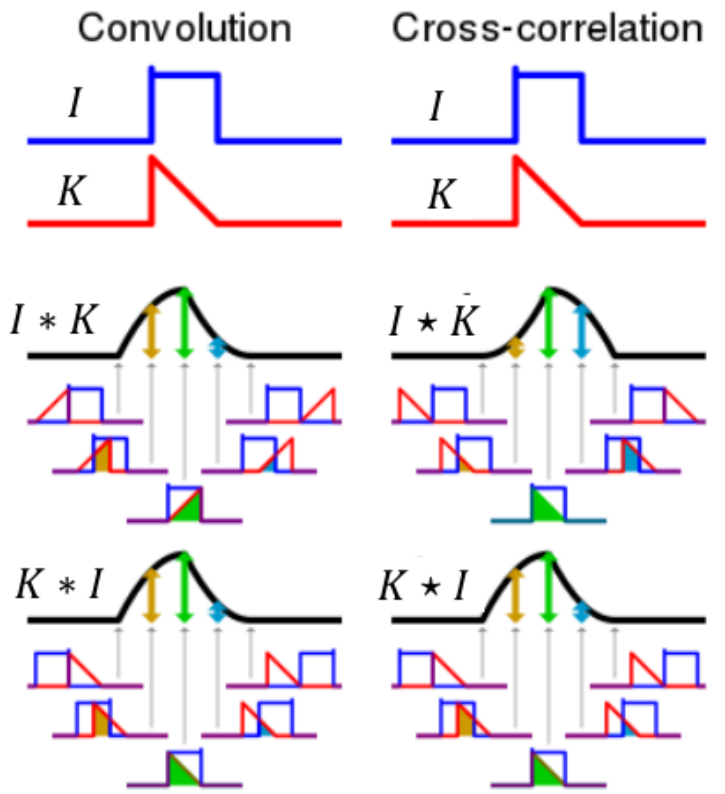
vs

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

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 ⇒ **easier to implement**



Discrete cross-correlation: 2-D example

I

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

$\star$

K

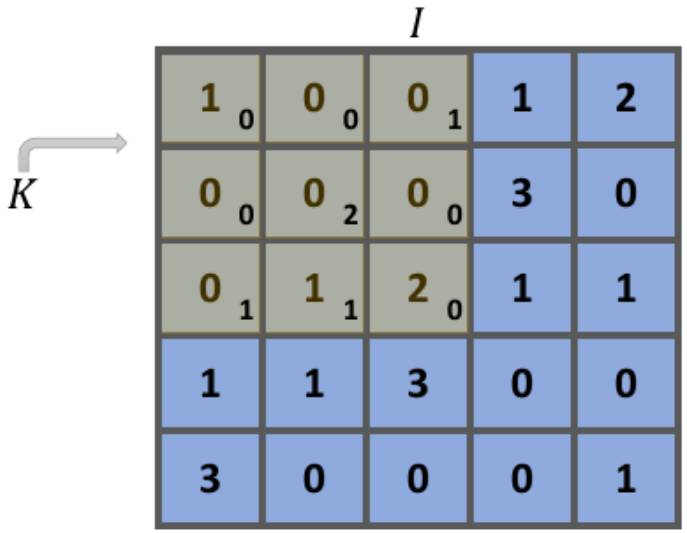
0	0	1
0	2	0
1	1	0

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

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

Discrete cross-correlation: 2-D example

Can be viewed as a “sliding window” operation:



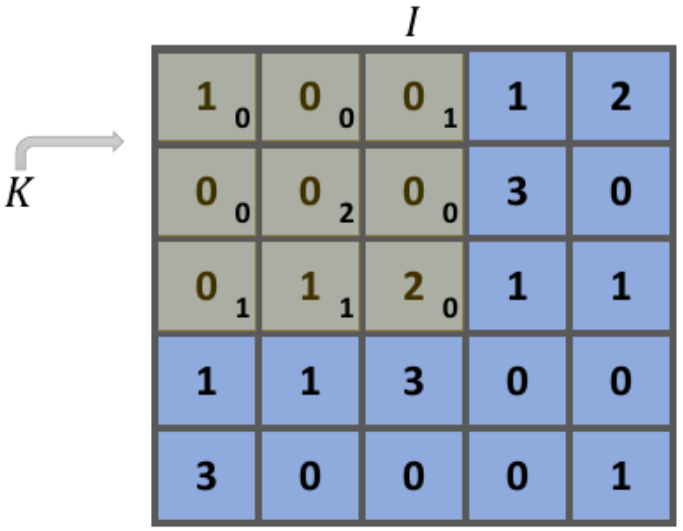
$I \star K$

1	4	11
4	11	5
7	7	7

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

Discrete cross-correlation: 2-D example

Can be viewed as a “sliding window” operation:



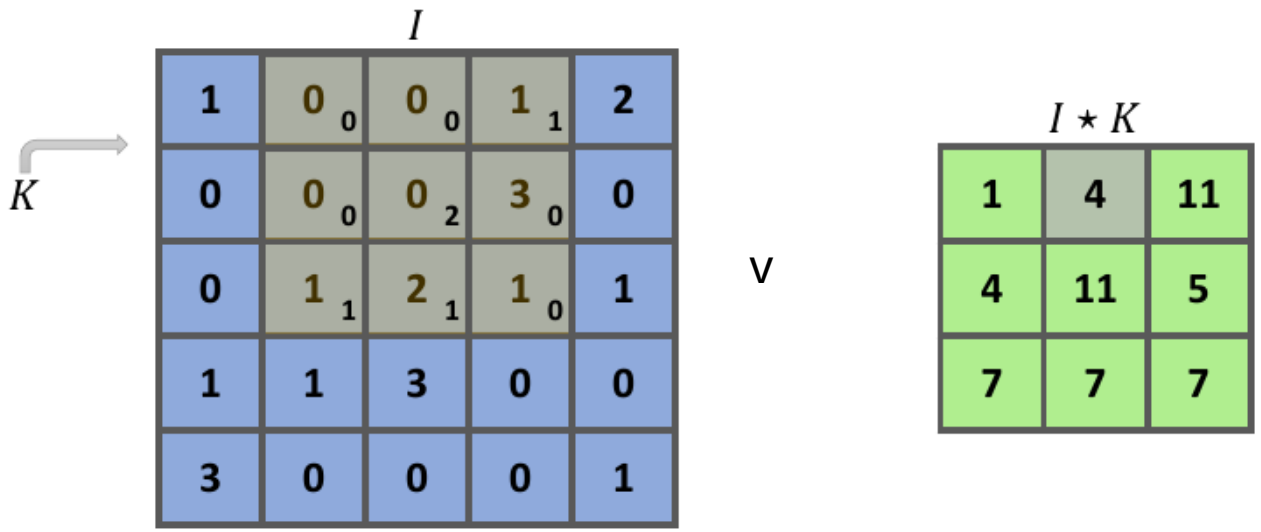
$I \star K$

1	4	11
4	11	5
7	7	7

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

Discrete cross-correlation: 2-D example

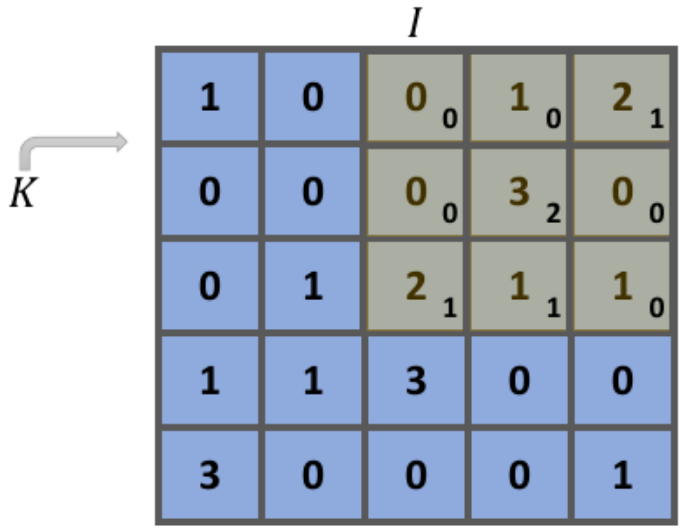
Can be viewed as a “sliding window” operation:

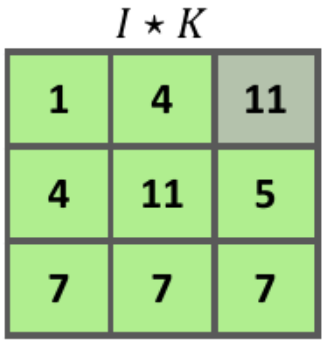


$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Discrete cross-correlation: 2-D example

Can be viewed as a “sliding window” operation:

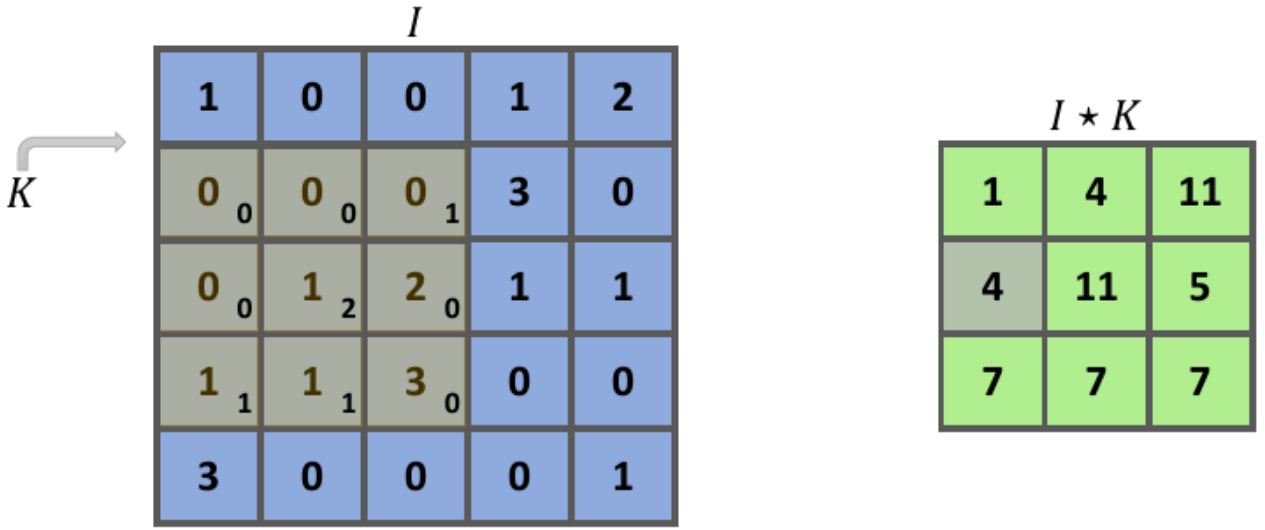




$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Discrete cross-correlation: 2-D example

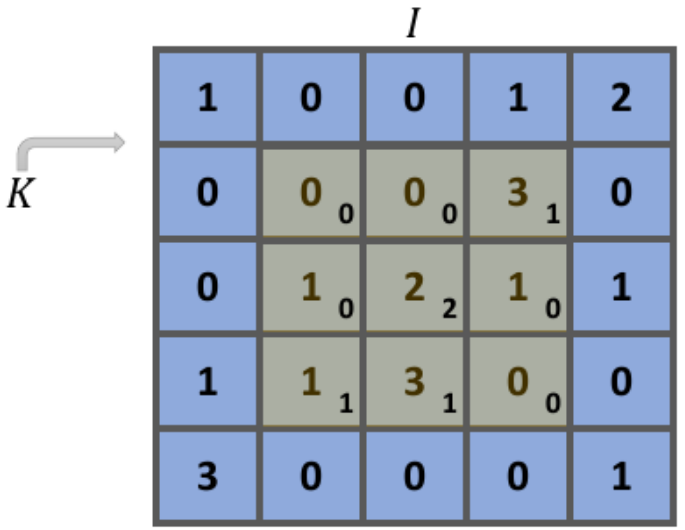
Can be viewed as a “sliding window” operation:

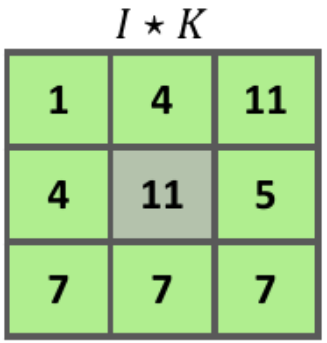


$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Discrete cross-correlation: 2-D example

Can be viewed as a “sliding window” operation:

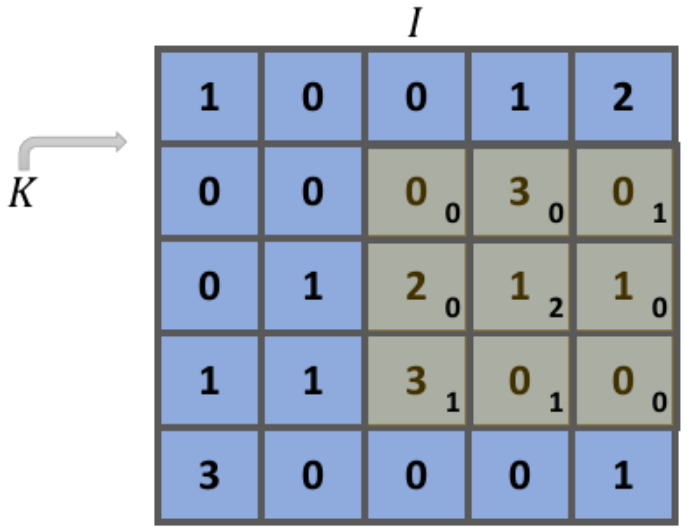




$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Discrete cross-correlation: 2-D example

Can be viewed as a “sliding window” operation:



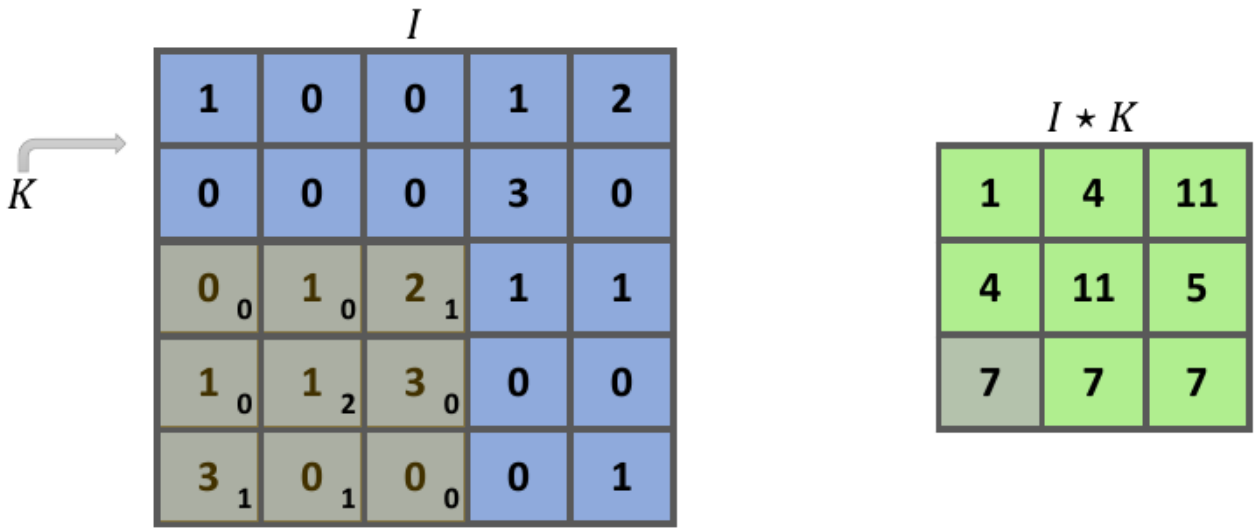
$I \star K$

1	4	11
4	11	5
7	7	7

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

Discrete cross-correlation: 2-D example

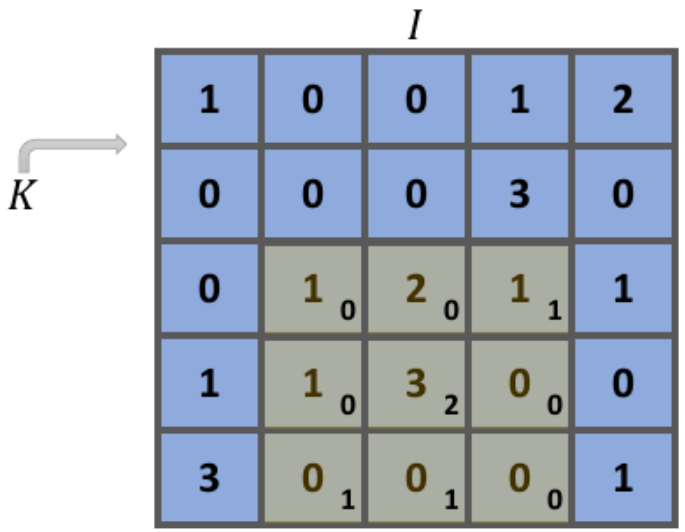
Can be viewed as a “sliding window” operation:

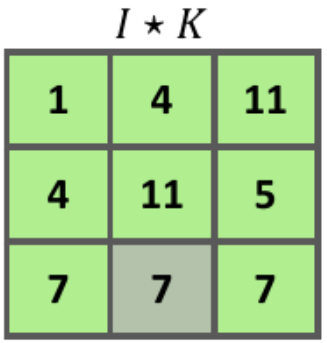


$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Discrete cross-correlation: 2-D example

Can be viewed as a “sliding window” operation:

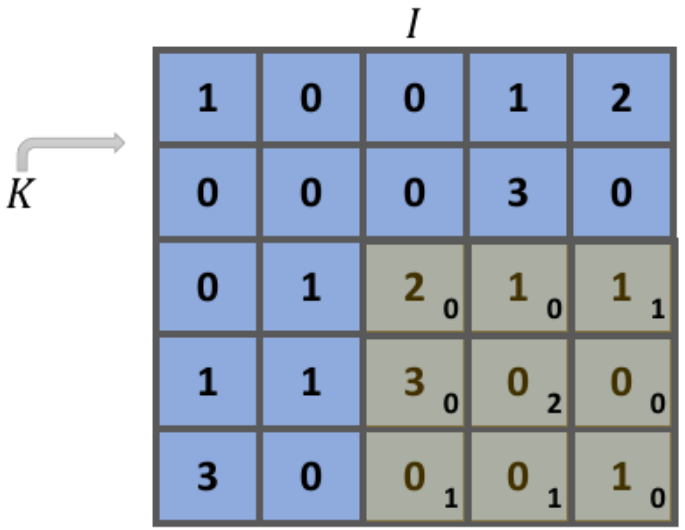


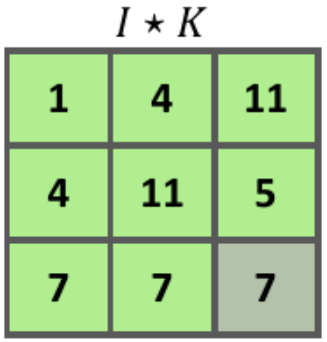


$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m,j+n)$$

Discrete cross-correlation: 2-D example

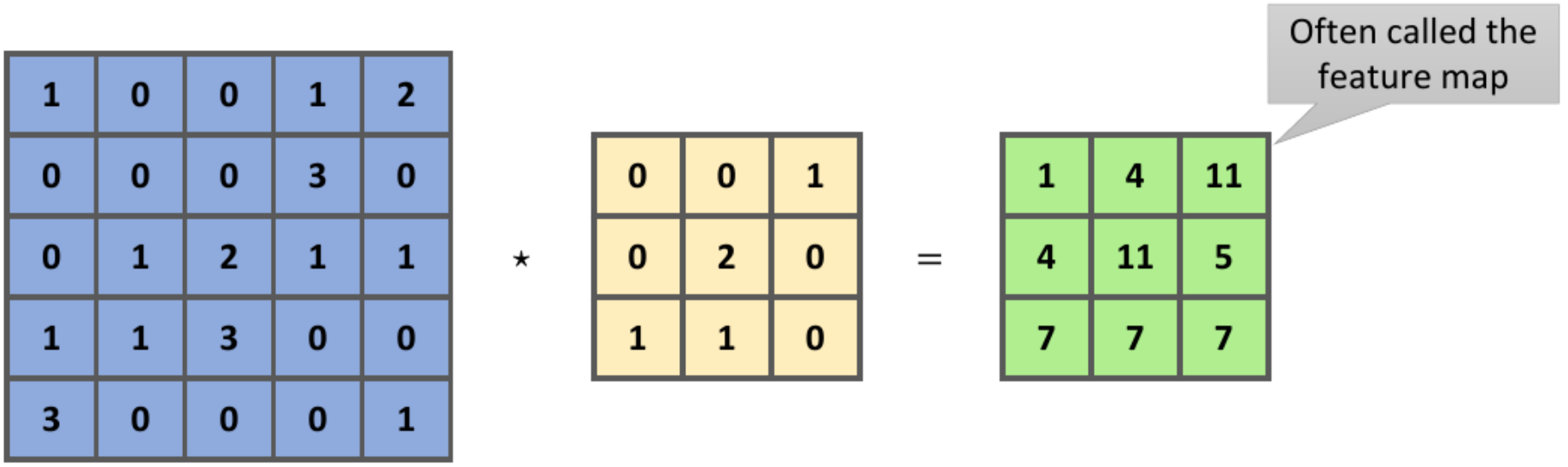
Can be viewed as a “sliding window” operation:





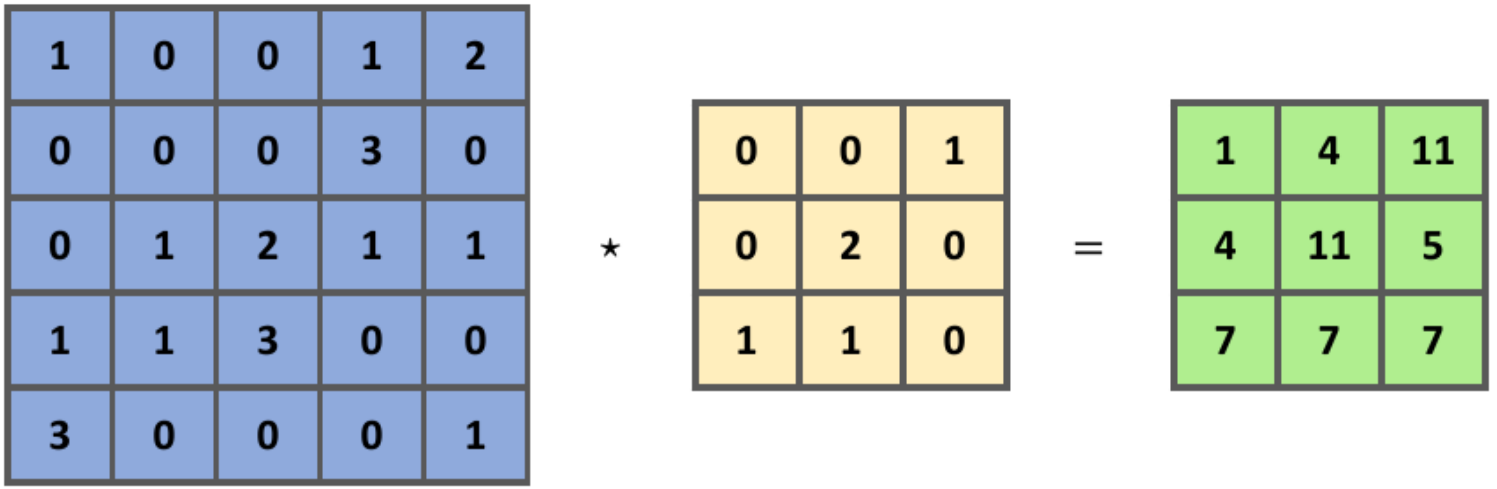
$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m,j+n)$$

Discrete cross-correlation: 2-D example



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

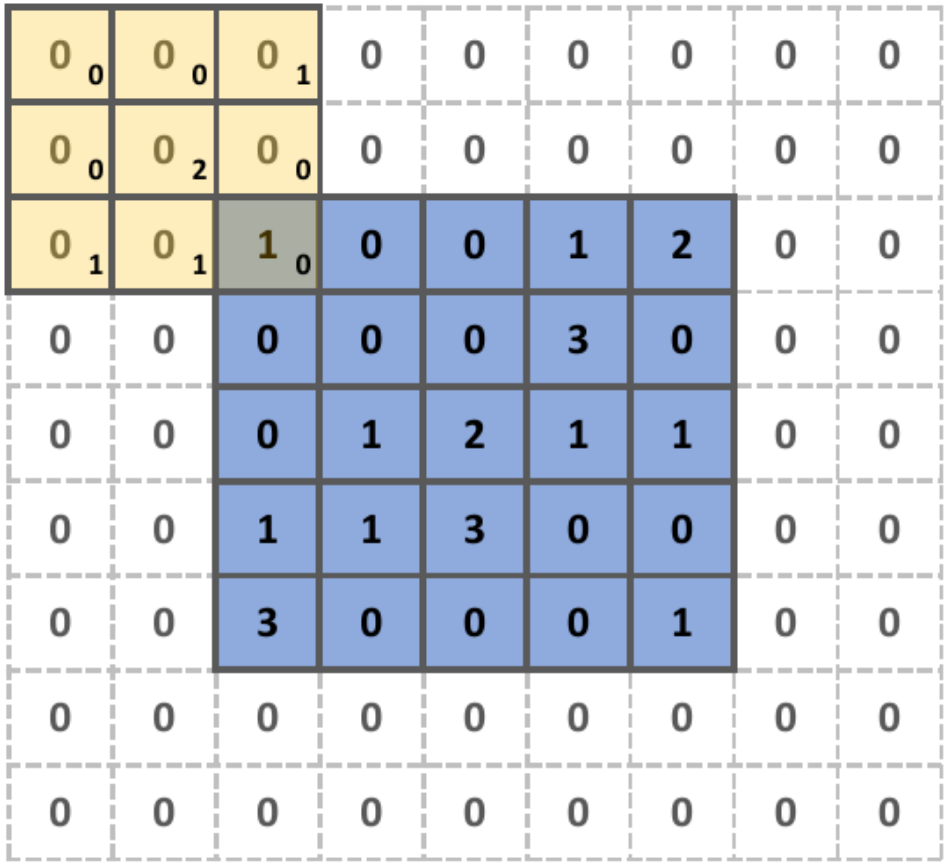
Discrete cross-correlation: 2-D example



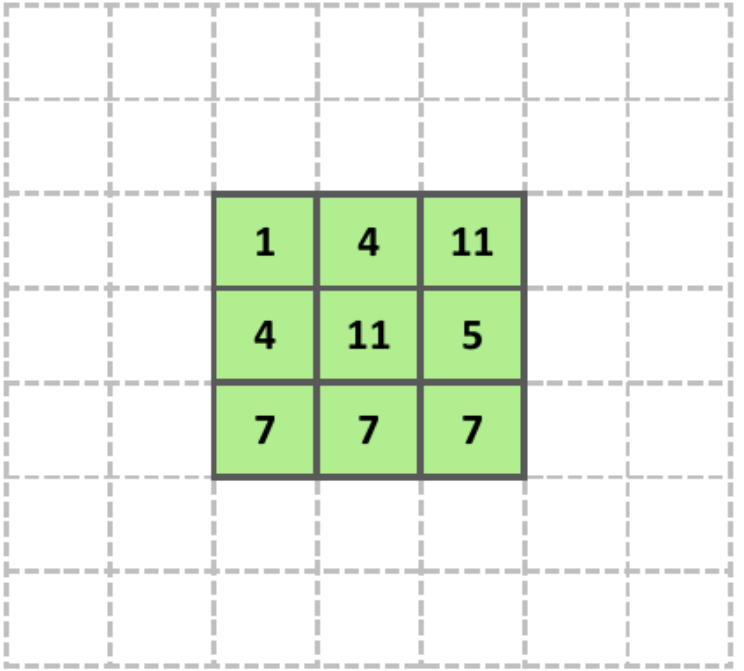
Something's not quite right…

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

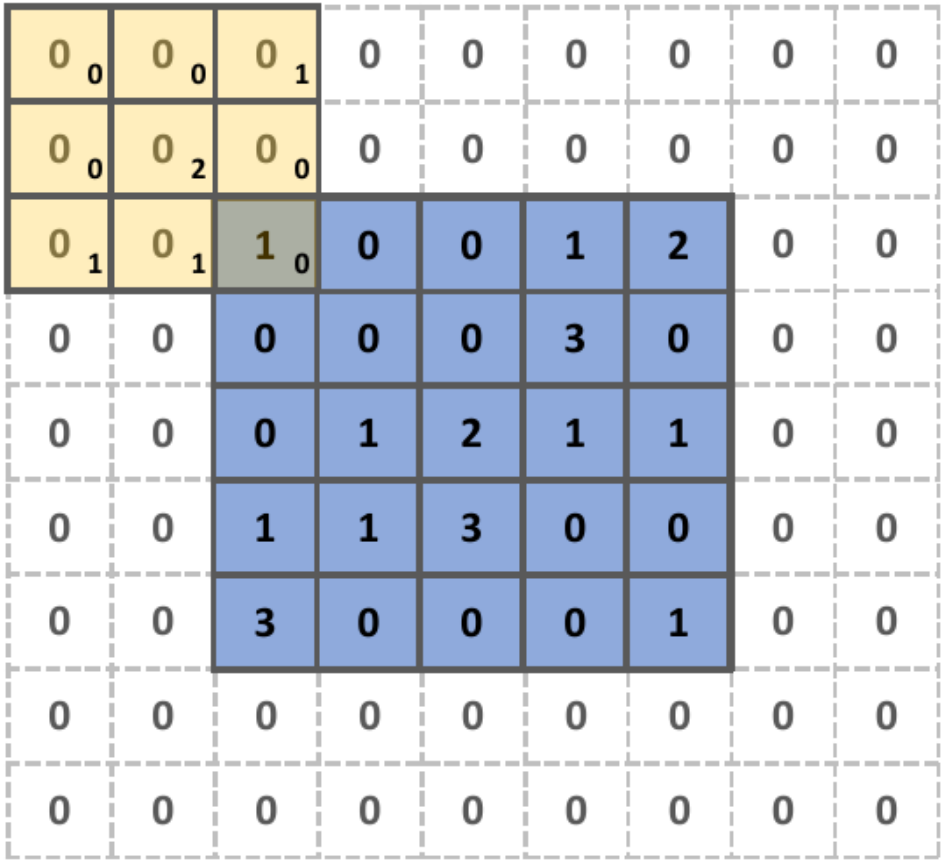
Discrete cross-correlation: padding

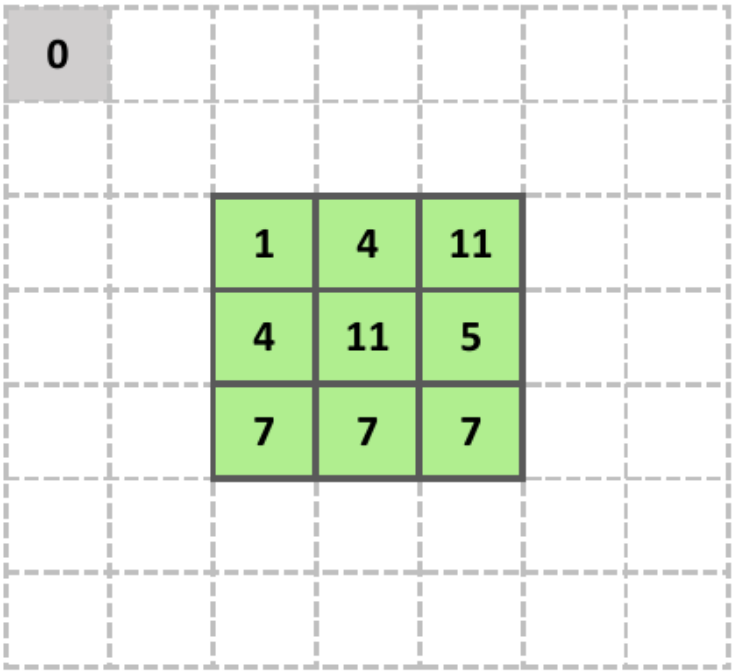


Technically, our signals have infinite extent…
we solve by padding with zeros

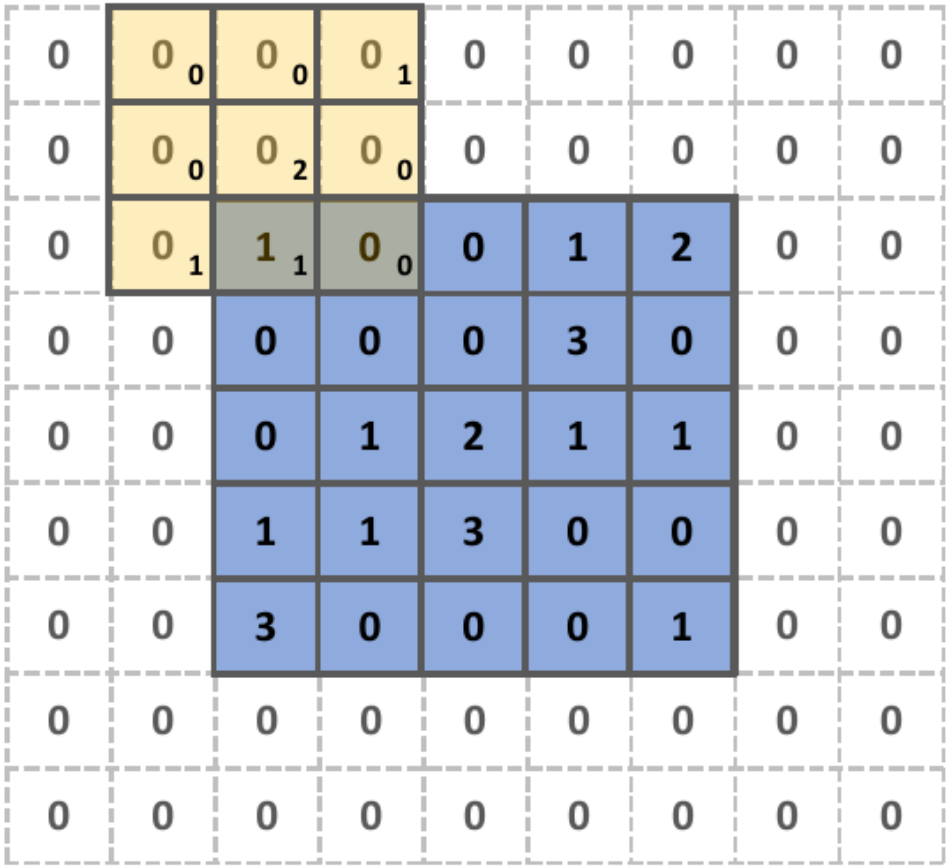


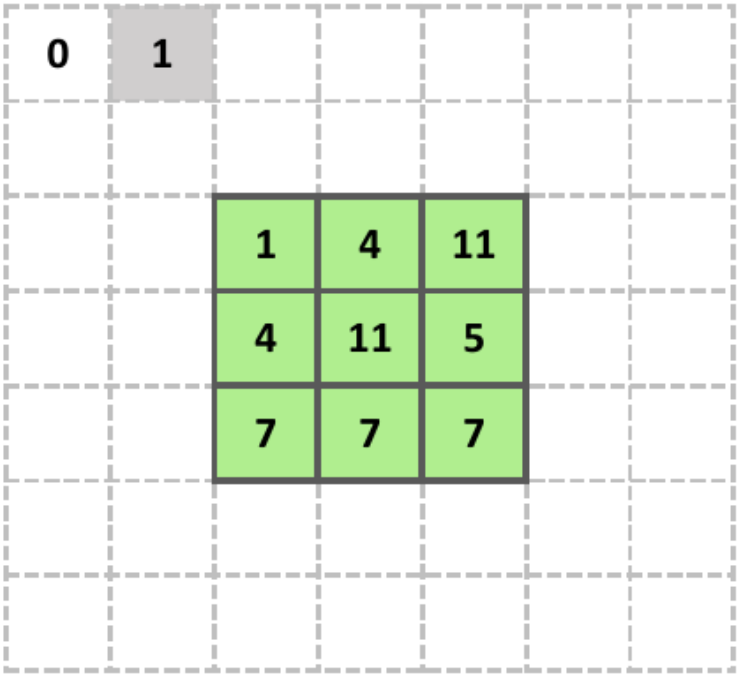
Discrete cross-correlation: padding



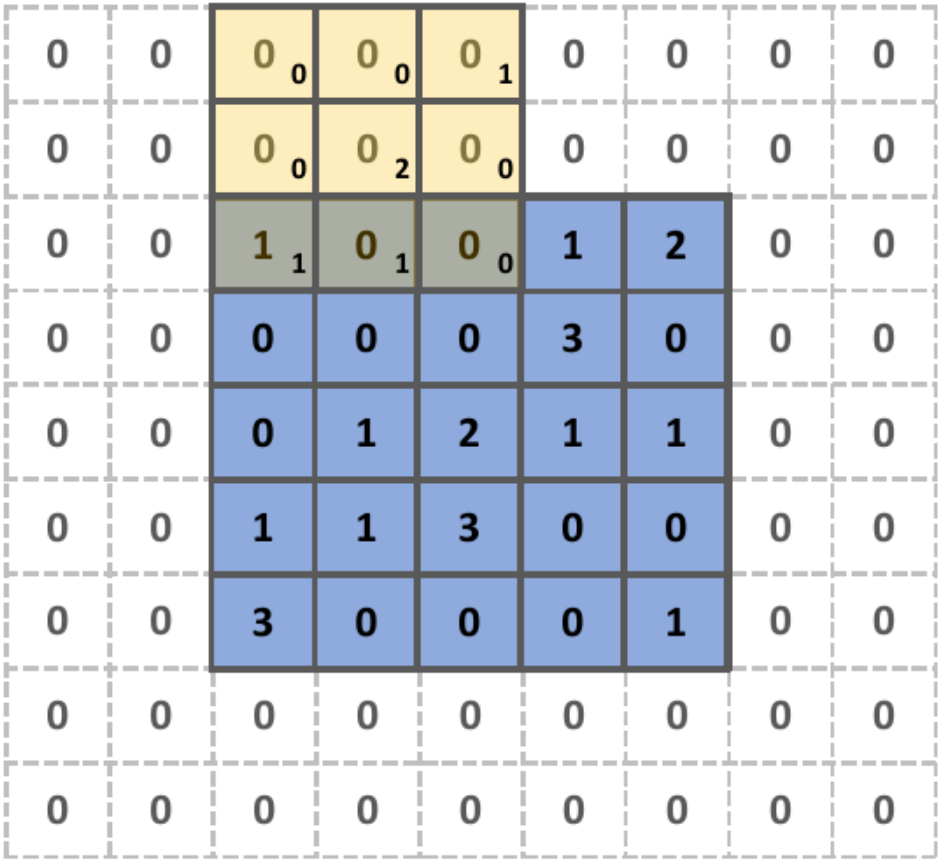


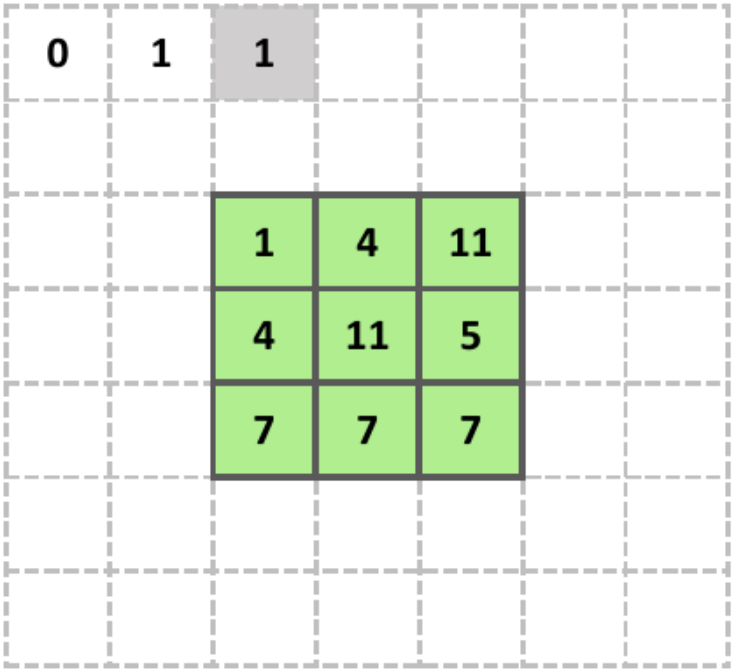
Discrete cross-correlation: padding



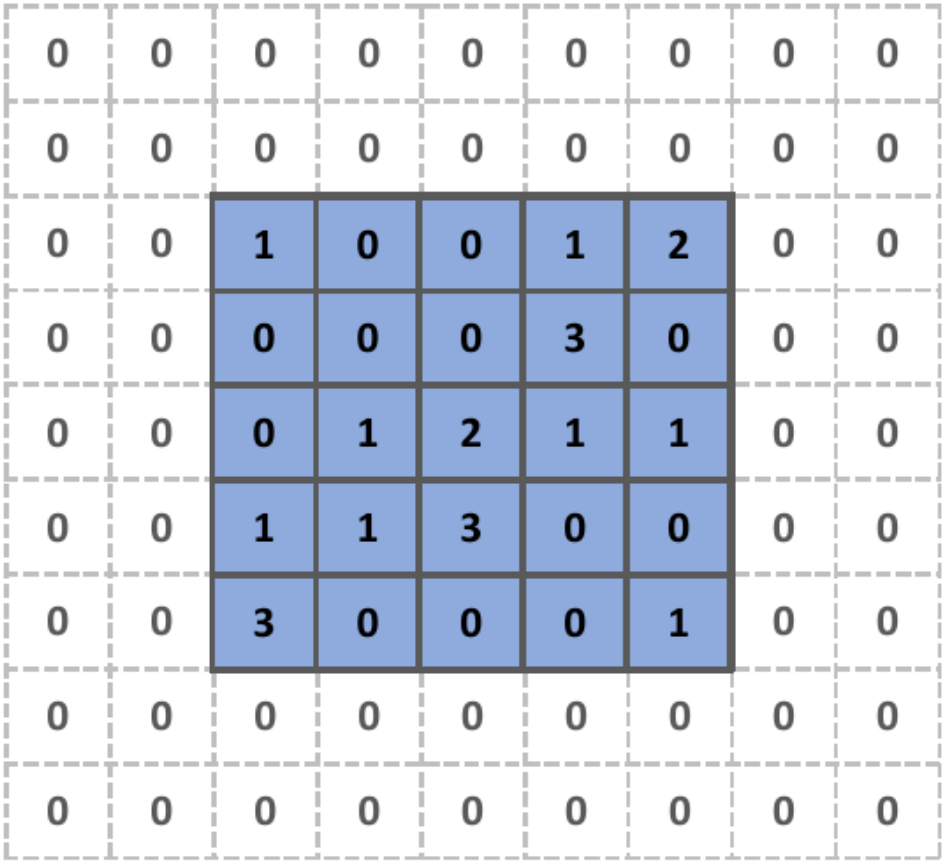


Discrete cross-correlation: padding

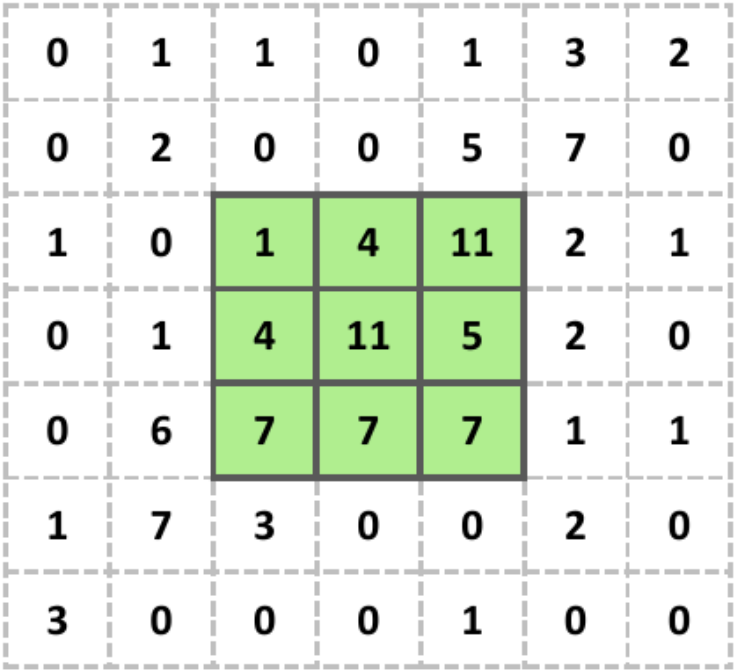




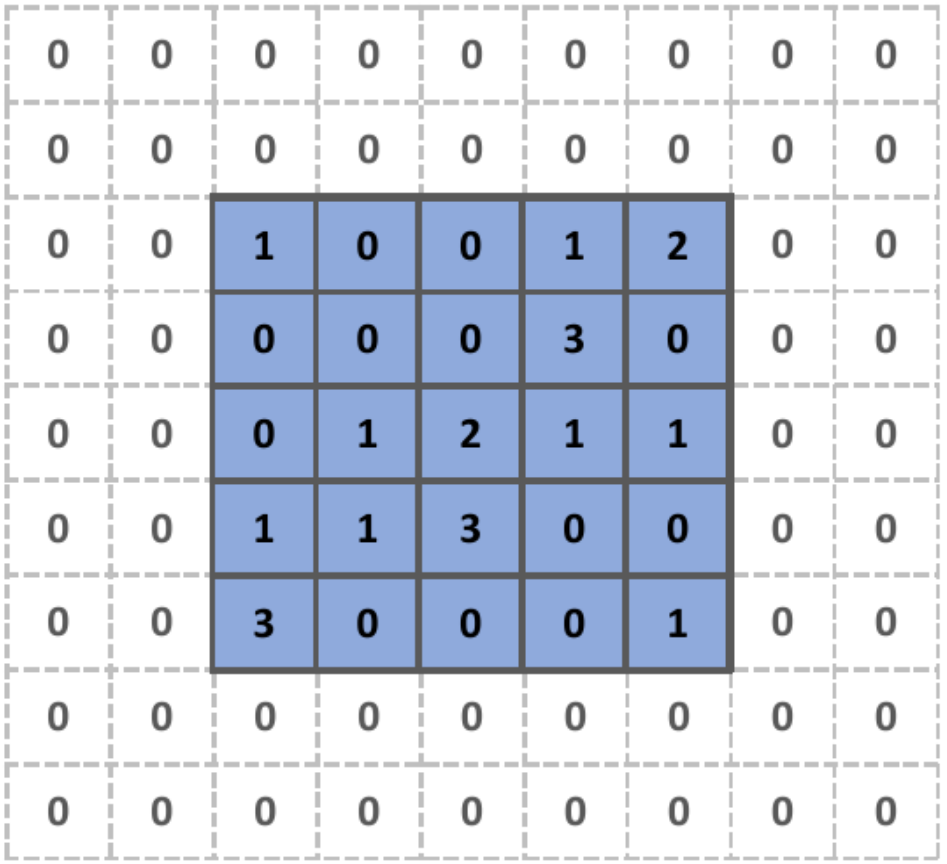
Discrete cross-correlation: padding



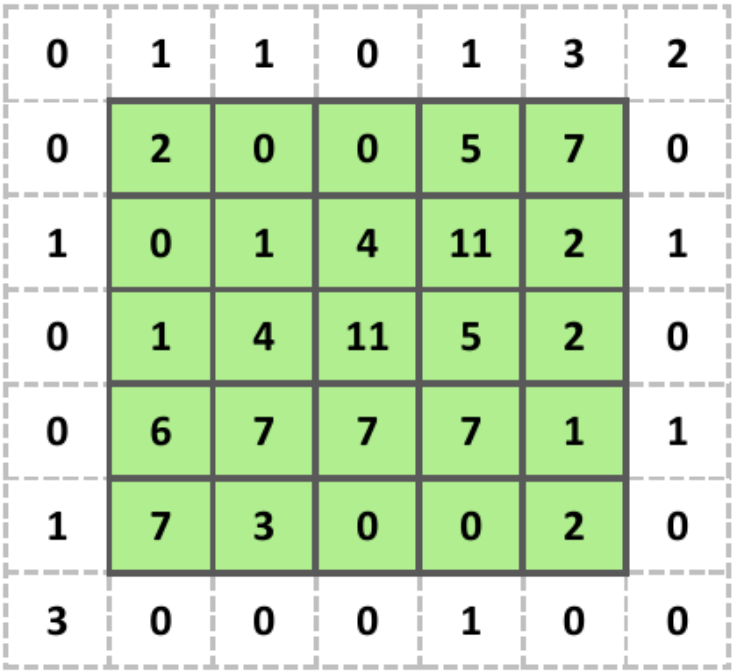
Normally we want the output to maintain the input size



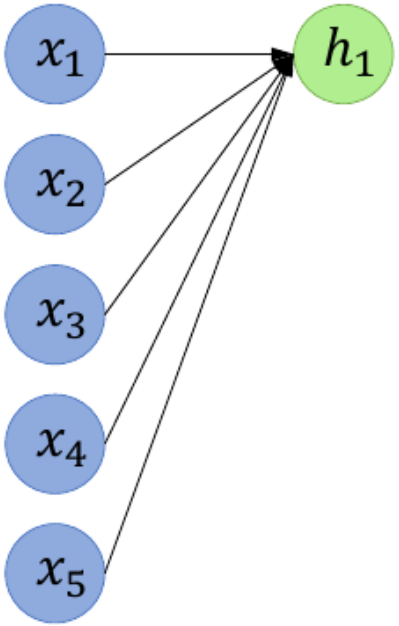
Discrete cross-correlation: padding



Normally we want the output to maintain the input size

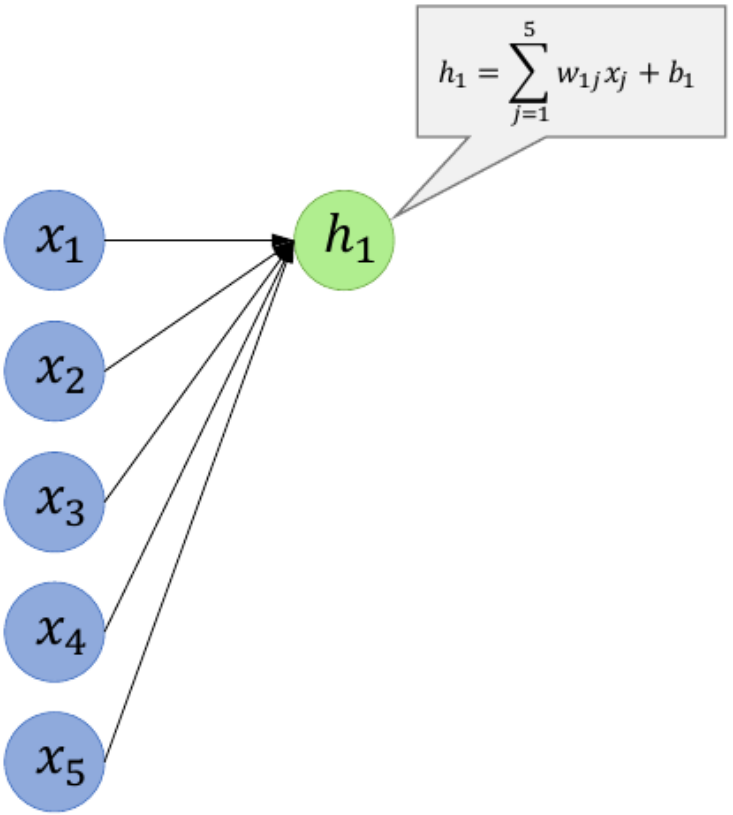


Neural networks ⇒ Convolutional networks



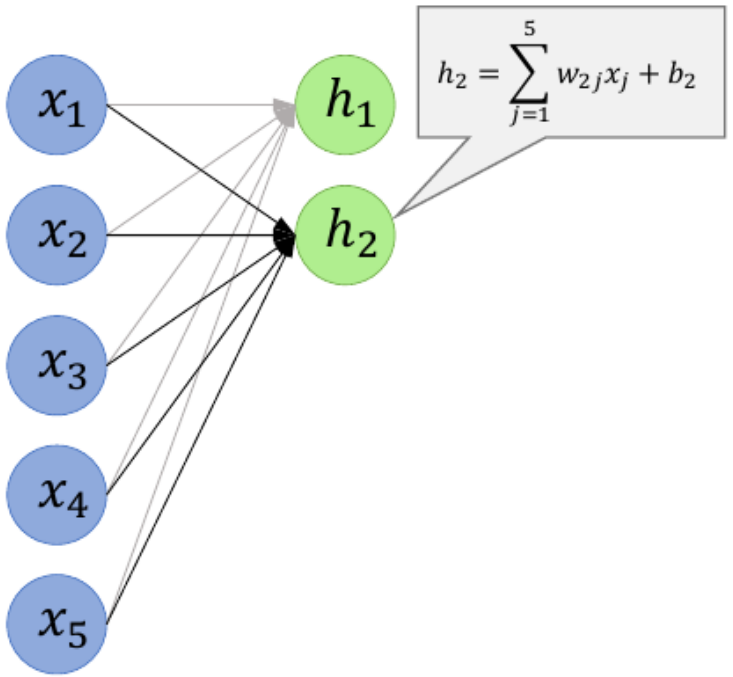
$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

Neural networks ⇒ Convolutional networks



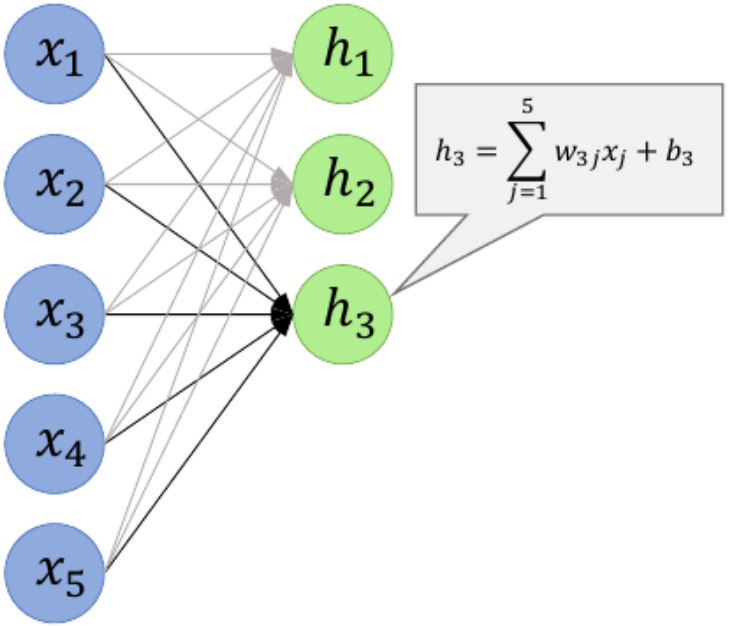
$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

Neural networks ⇒ Convolutional networks



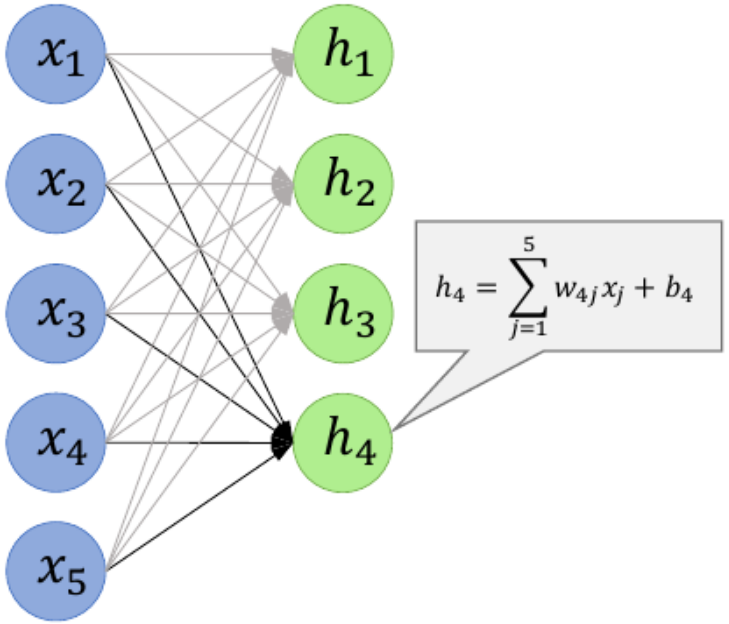
$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

Neural networks ⇒ Convolutional networks



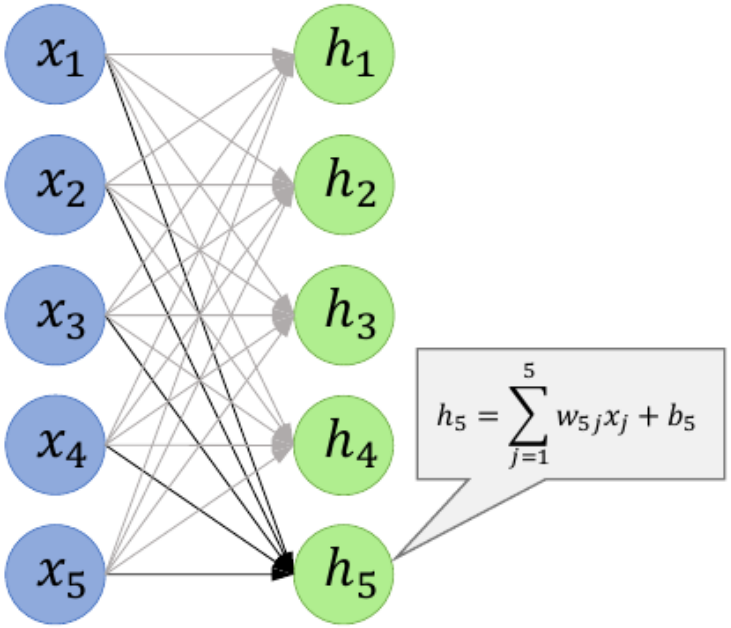
$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

Neural networks ⇒ Convolutional networks



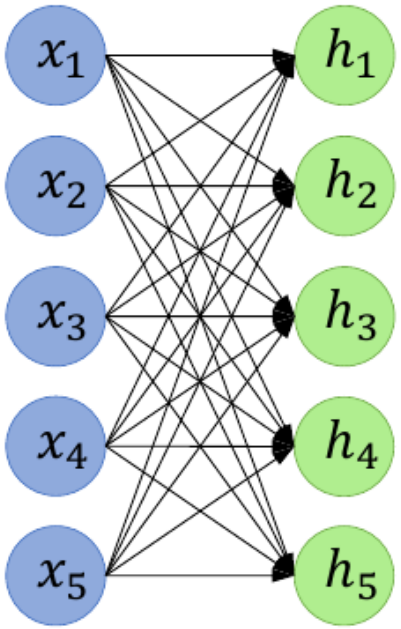
$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

Neural networks ⇒ Convolutional networks

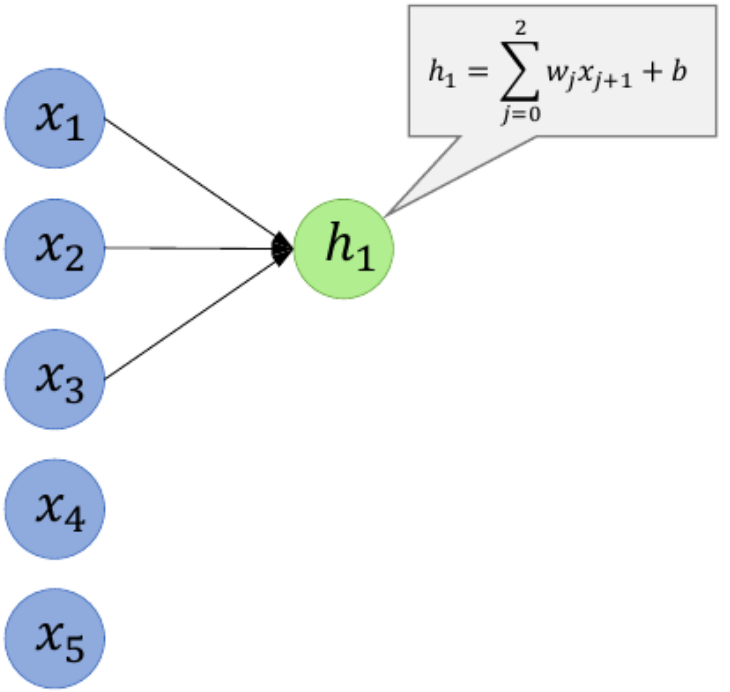


$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

Neural networks ⇒ Convolutional networks

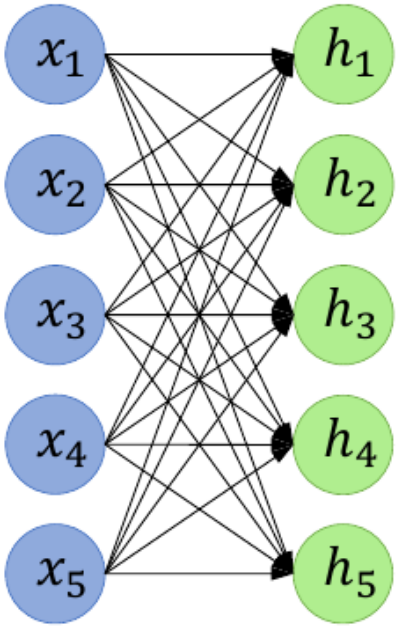


$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$

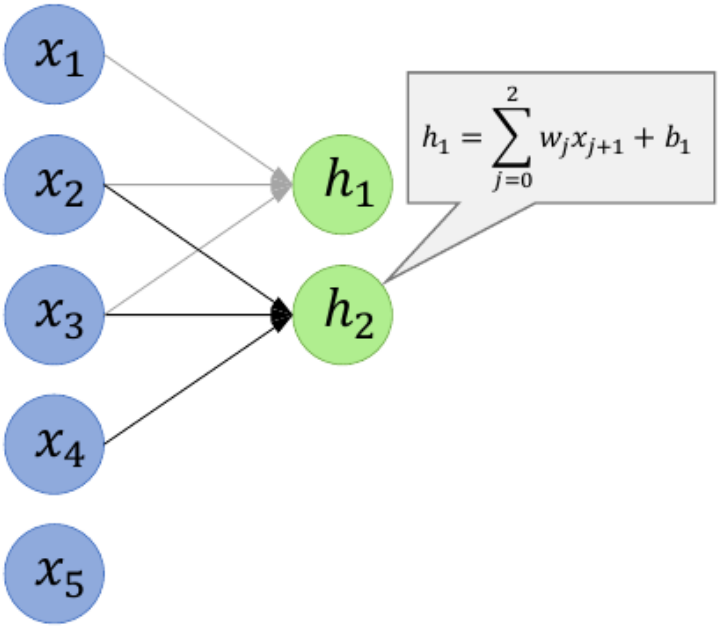


$h_i = \sum_j w_j x_{j+i} + b$

Neural networks ⇒ Convolutional networks

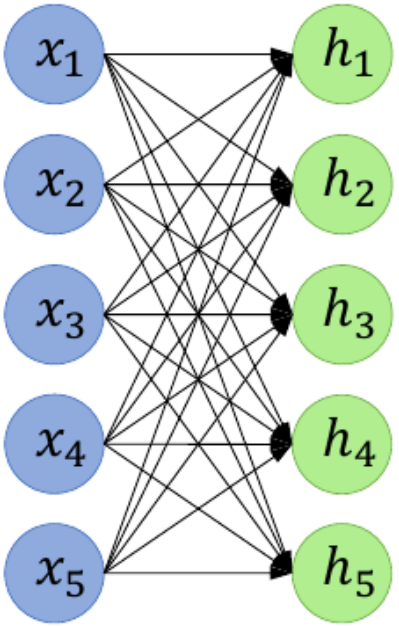


$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

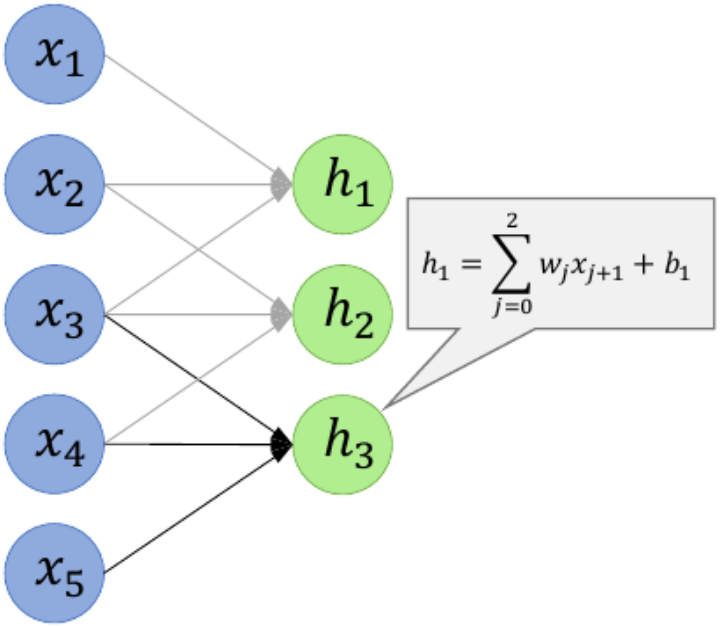


$$h_i = \sum_j w_j x_{j+i} + b$$

Neural networks ⇒ Convolutional networks

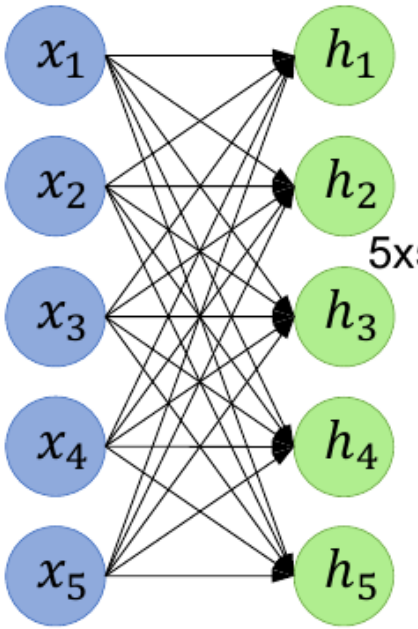


$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$



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

Neural networks ⇒ Convolutional networks

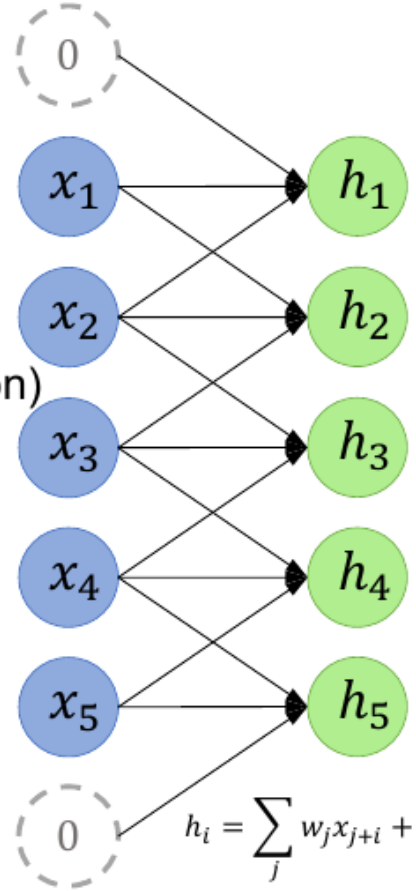


$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$

dense connectivity
vs
sparse connectivity

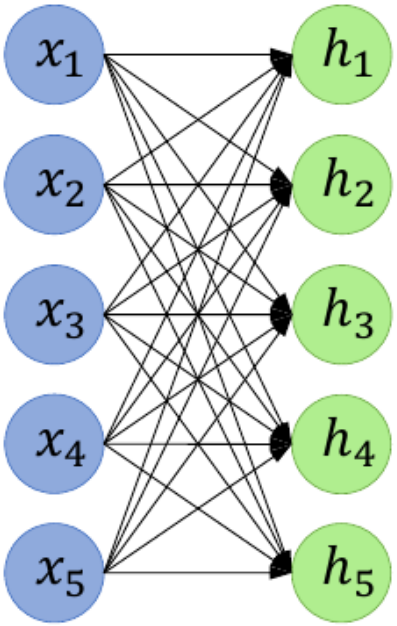
In our example:
5x5 multiplications (naïve implementation)
vs
5x3 multiplications

Sparse connectivity scales better
e.g. 25x25 vs 25x3



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

Neural networks ⇒ Convolutional networks

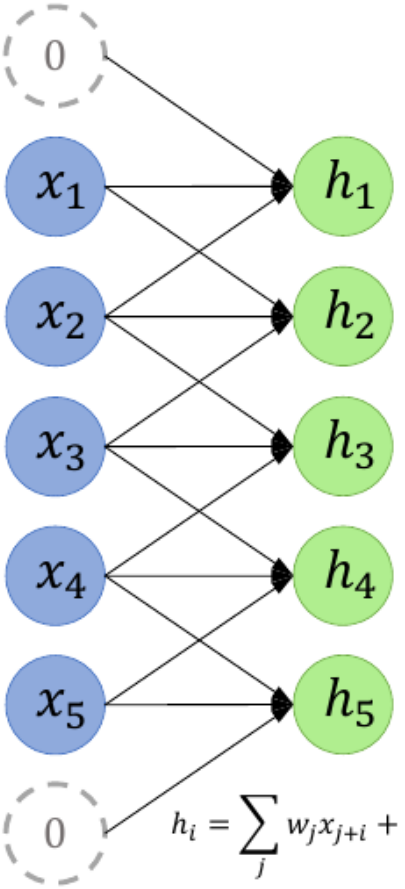


$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$

unshared vs shared weights:

In our example:
5x5 vs 3 weights

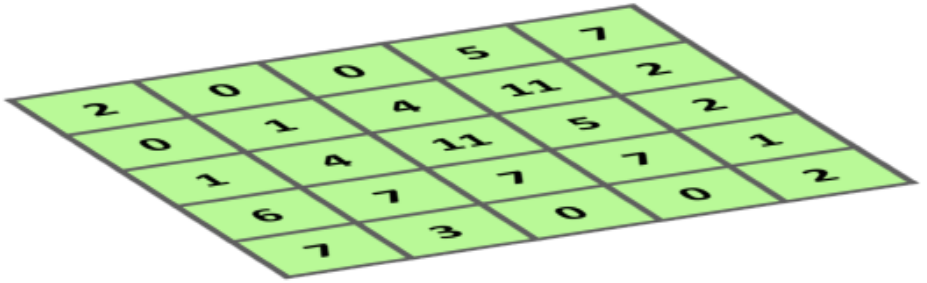
shared weights scale way better:
e.g. 25x25 vs 3

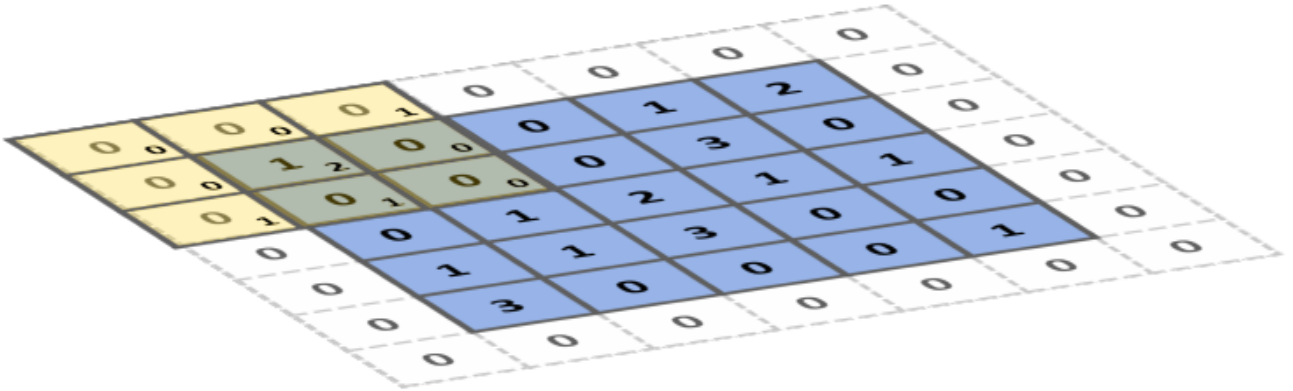


$h_i = \sum_j w_j x_{j+i} + b$

The effect of different strides

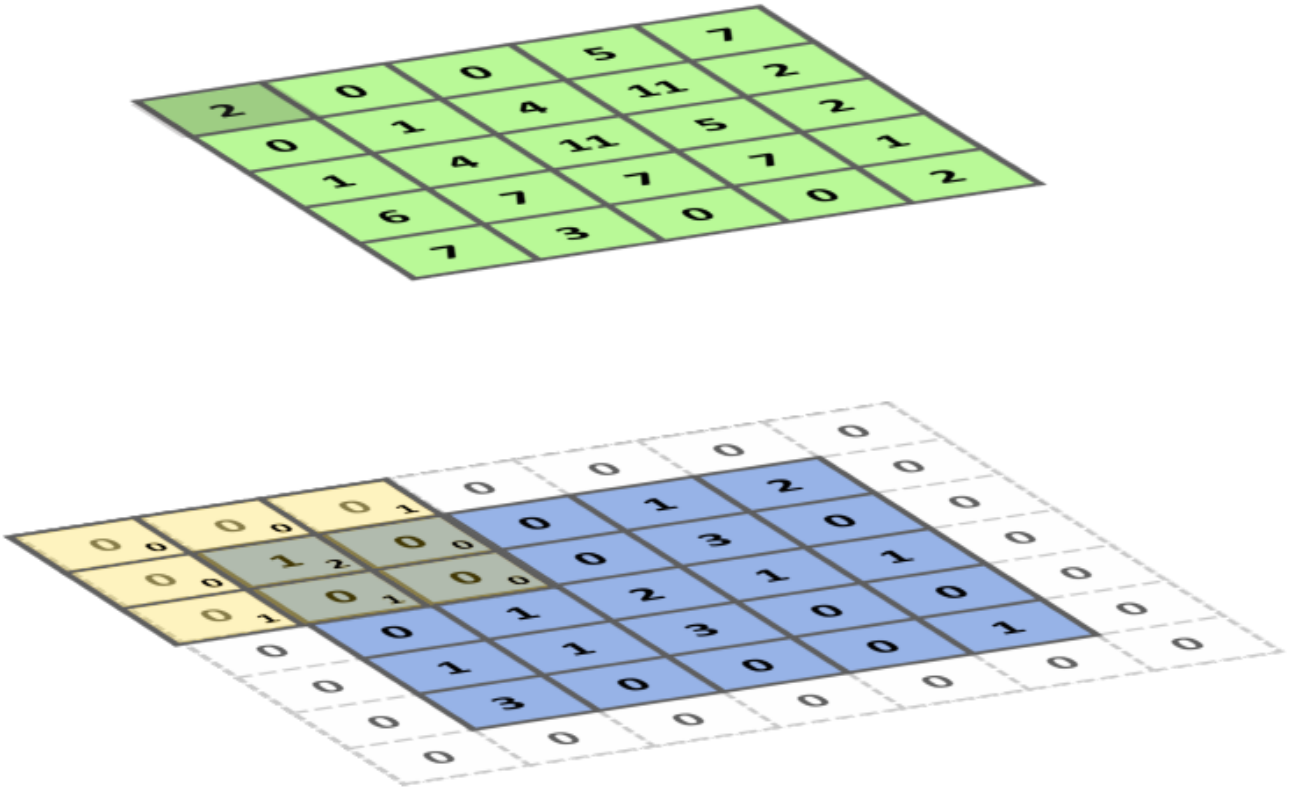
Stride: distance between two consecutive positions of the kernel





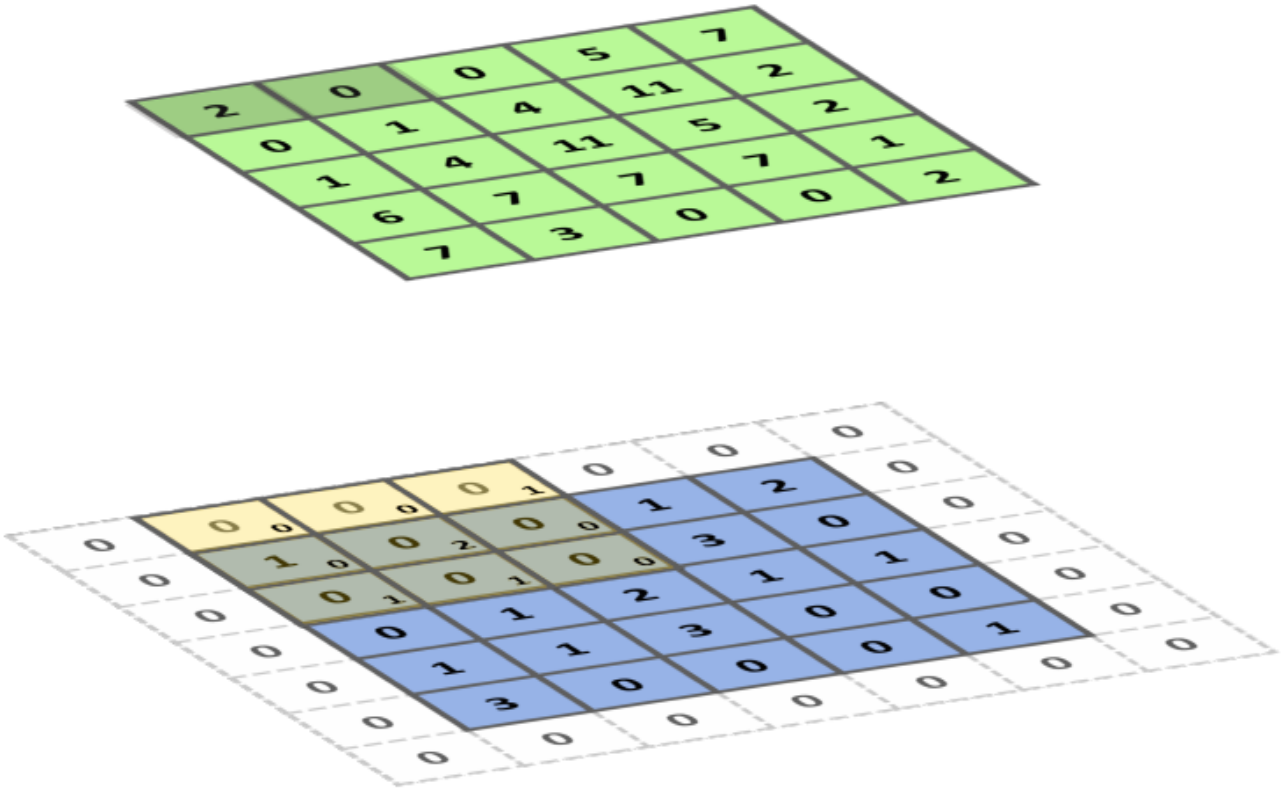
The effect of different strides

Stride: 1x1



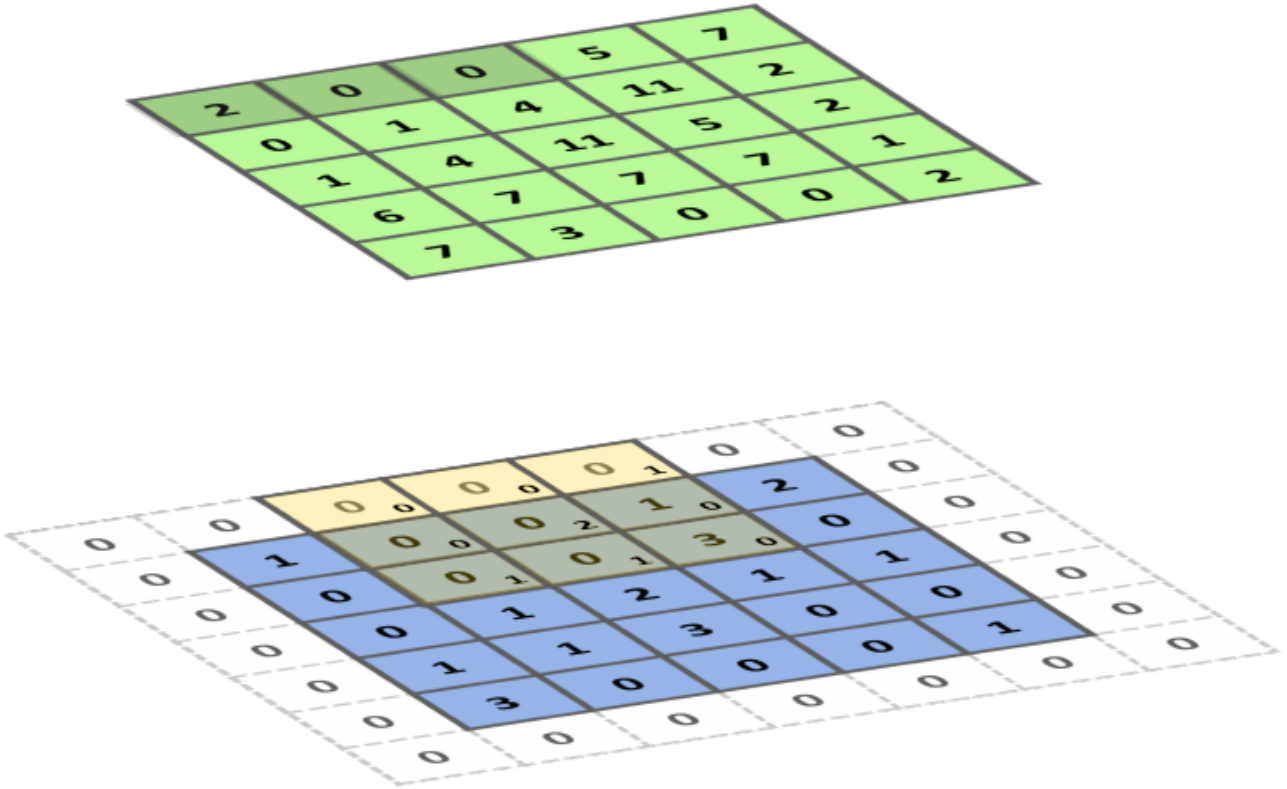
The effect of different strides

Stride: 1x1



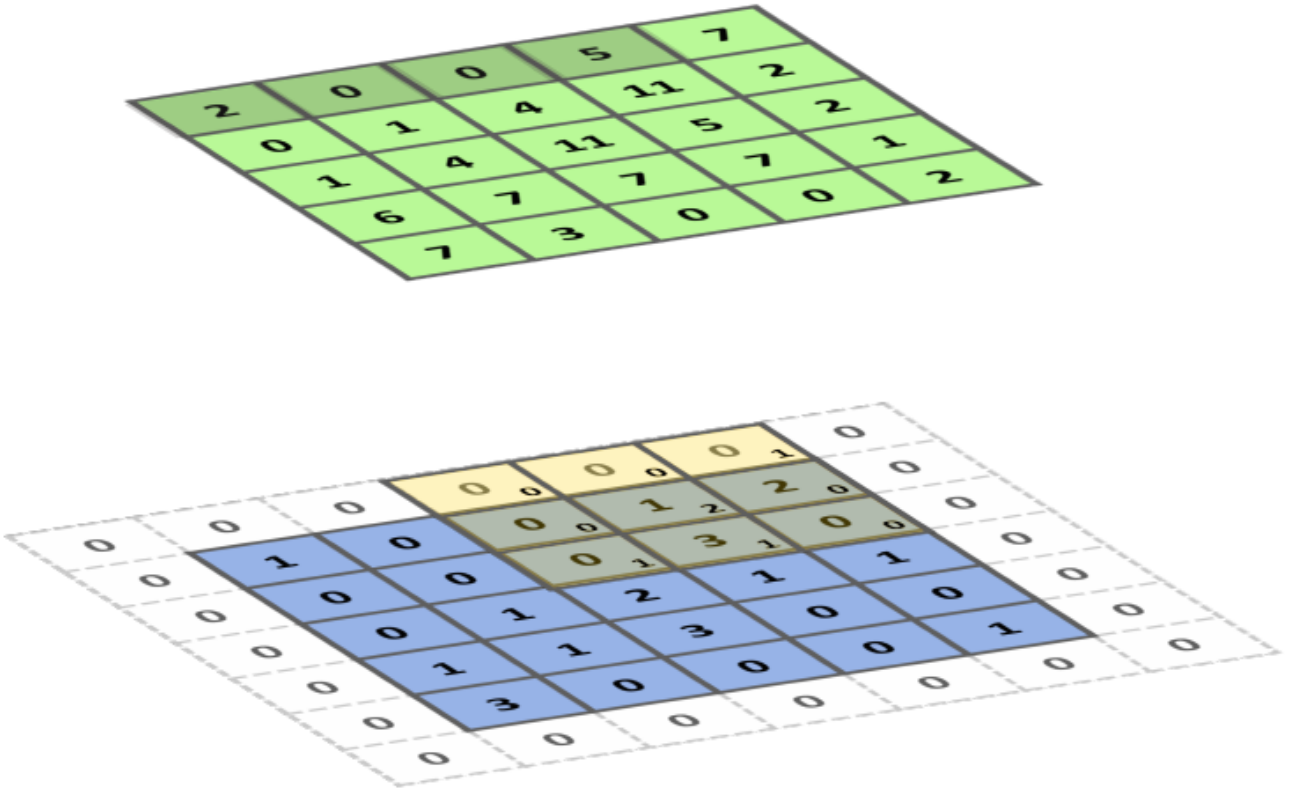
The effect of different strides

Stride: 1x1



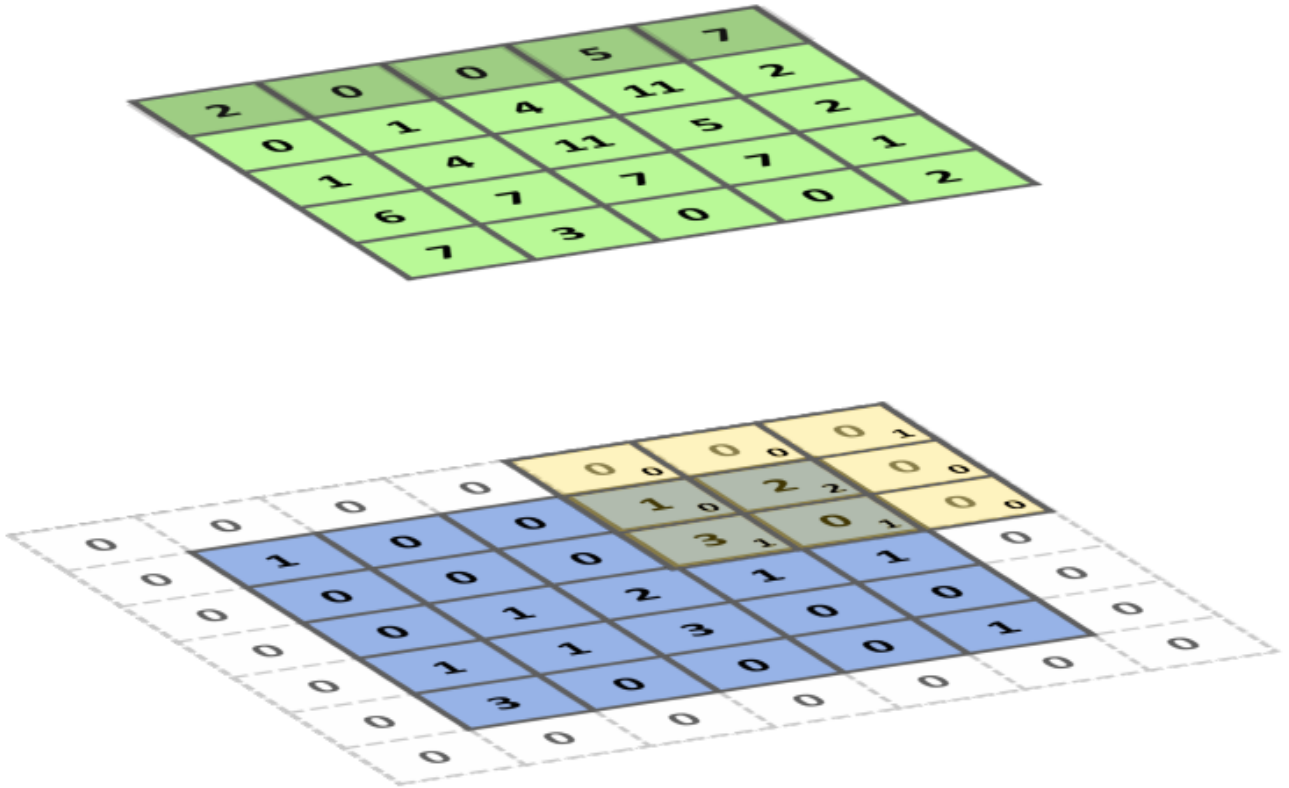
The effect of different strides

Stride: 1x1



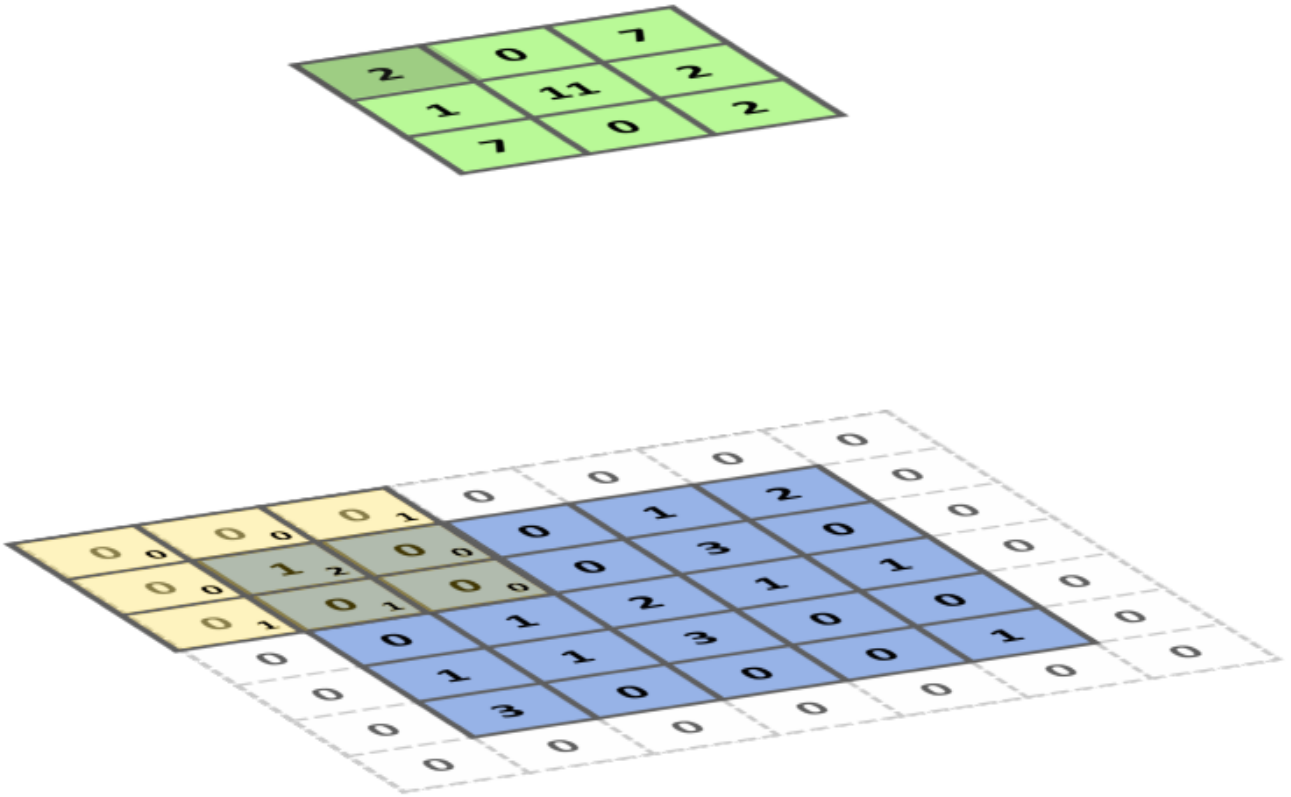
The effect of different strides

Stride: 1x1



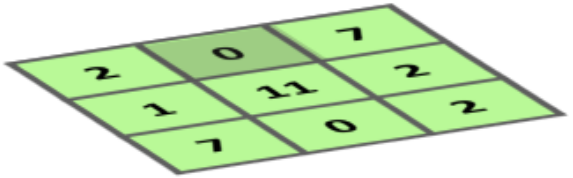
The effect of different strides

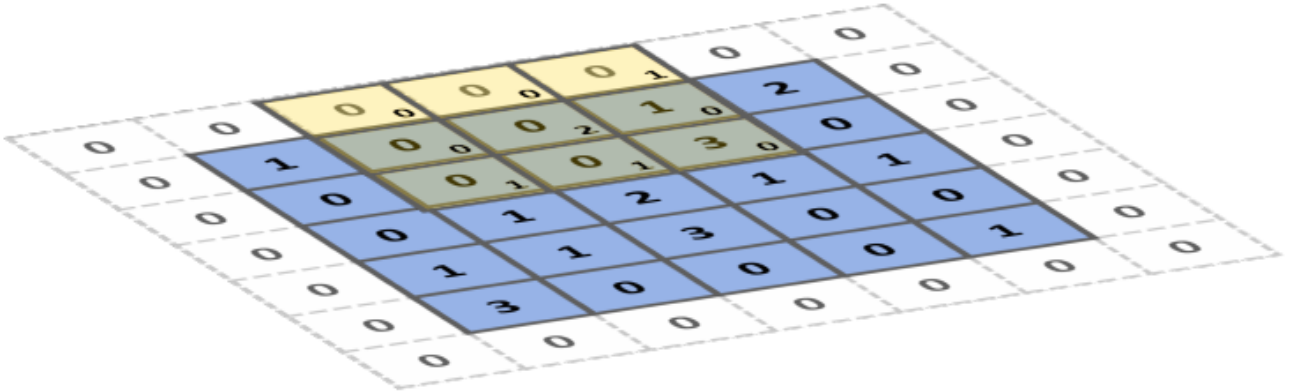
Stride: 2x2



The effect of different strides

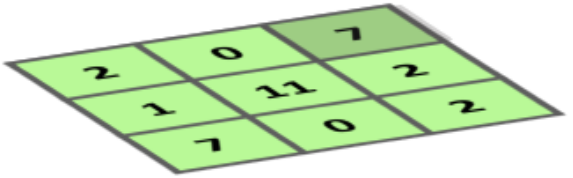
Stride: 2x2

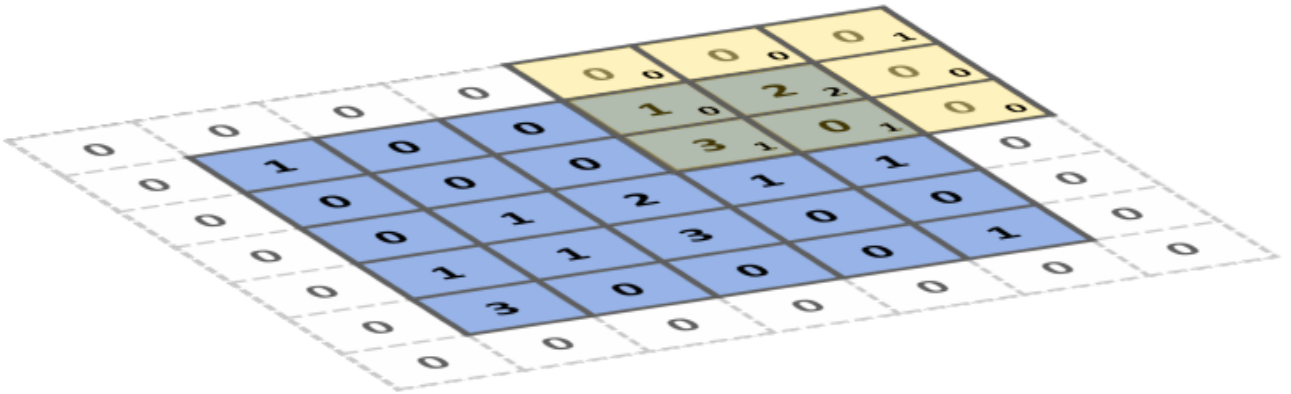




The effect of different strides

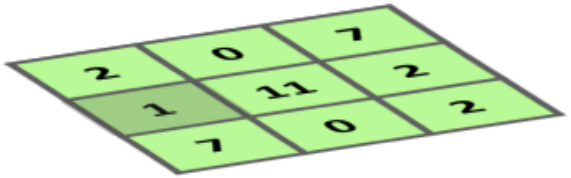
Stride: 2x2

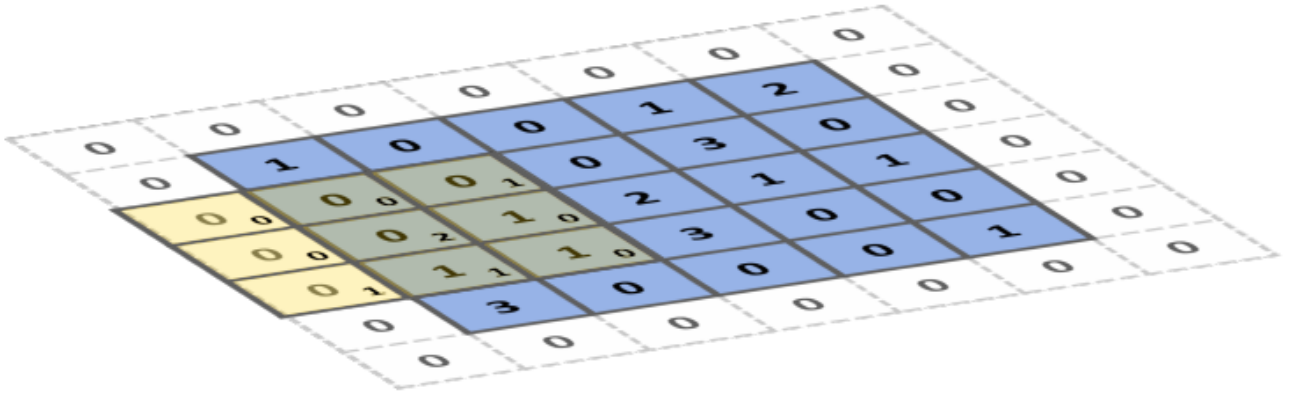




The effect of different strides

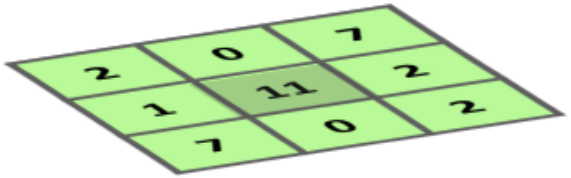
Stride: 2x2

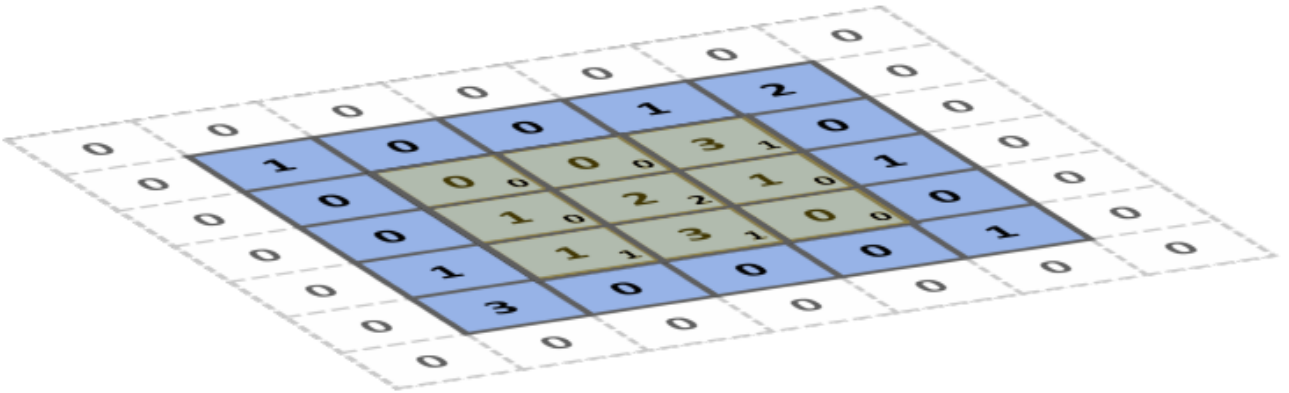




The effect of different strides

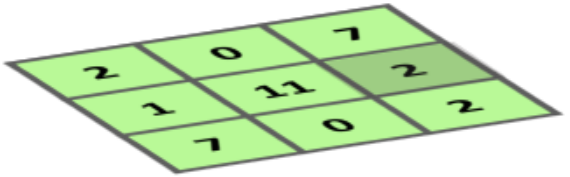
Stride: 2x2

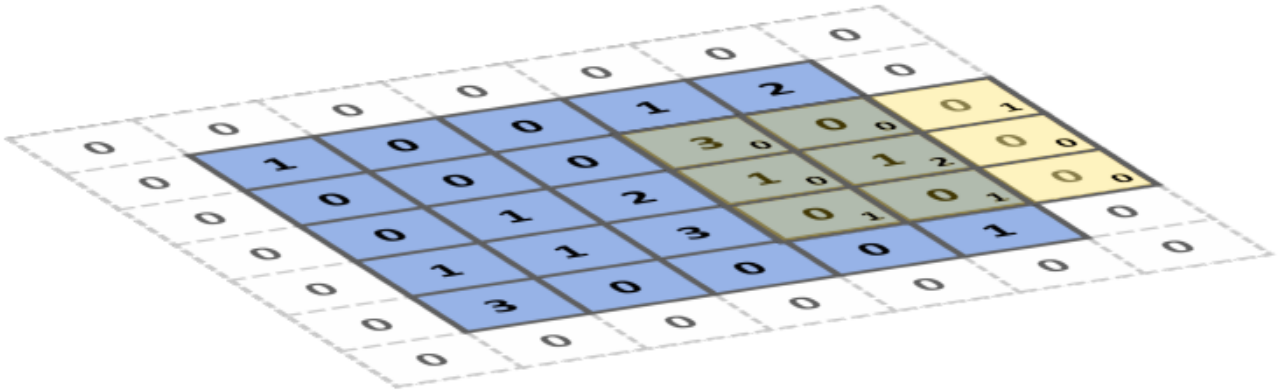




The effect of different strides

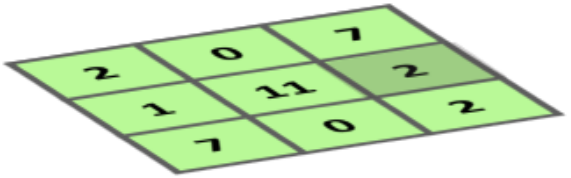
Stride: 2x2





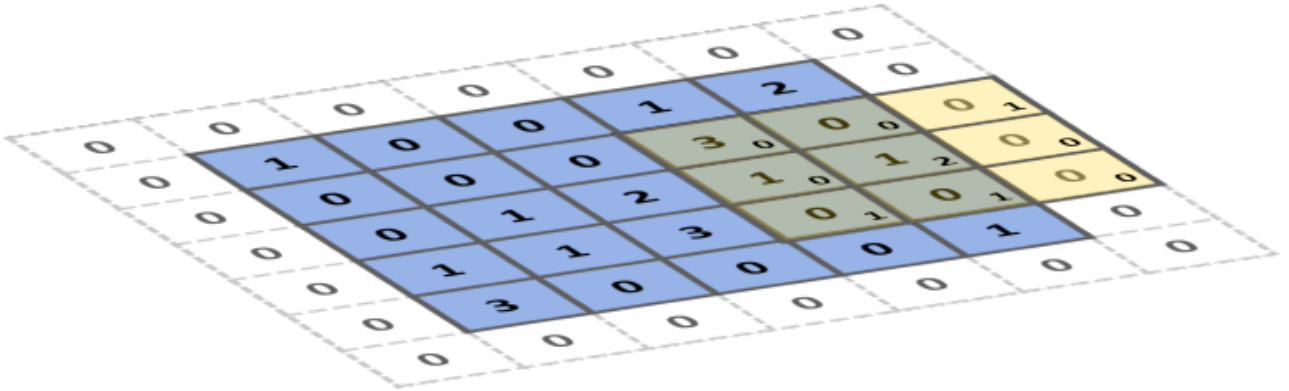
The effect of different strides

Stride: 2x2



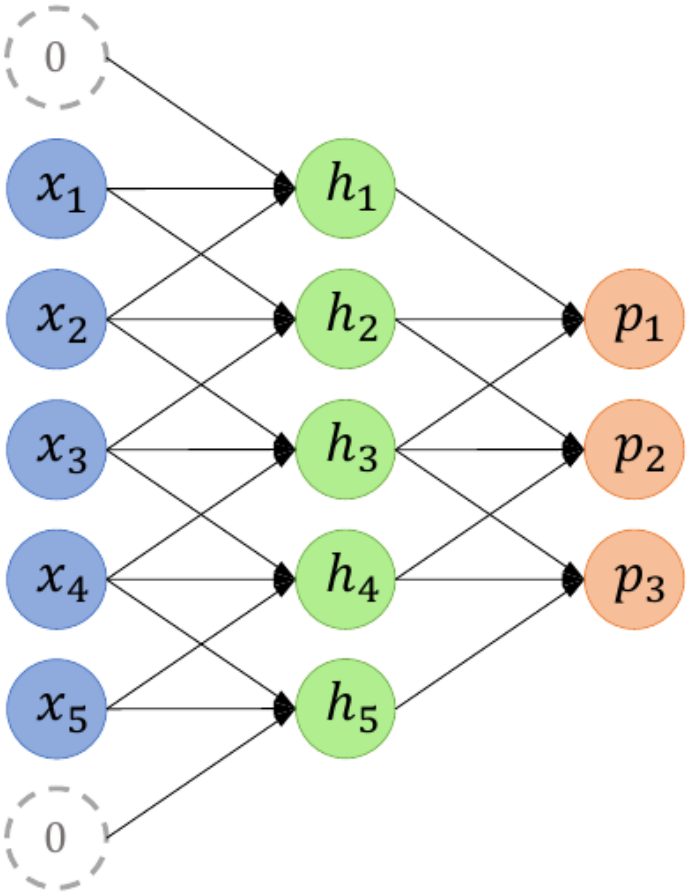
Why use stride > 1?

- Reduce redundancy
- Compress feature map



Pooling

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



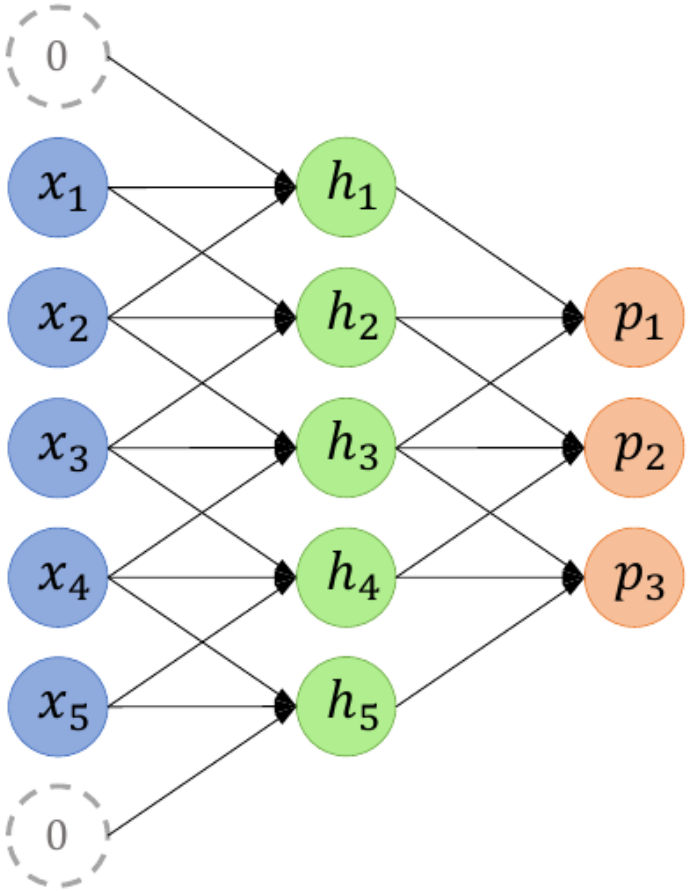
Pooling

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

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

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

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



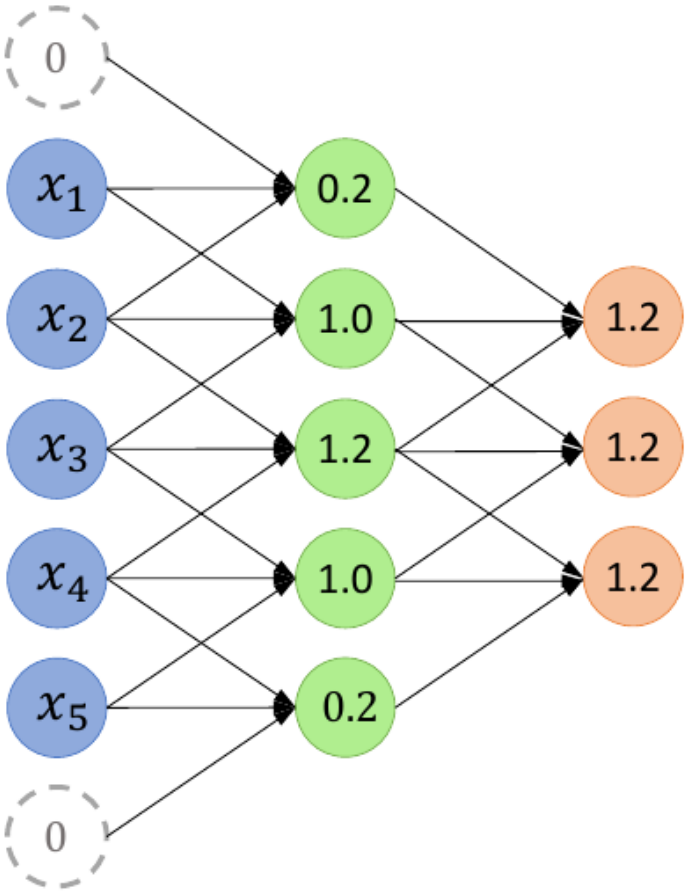
Pooling

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

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

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

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



Pooling

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

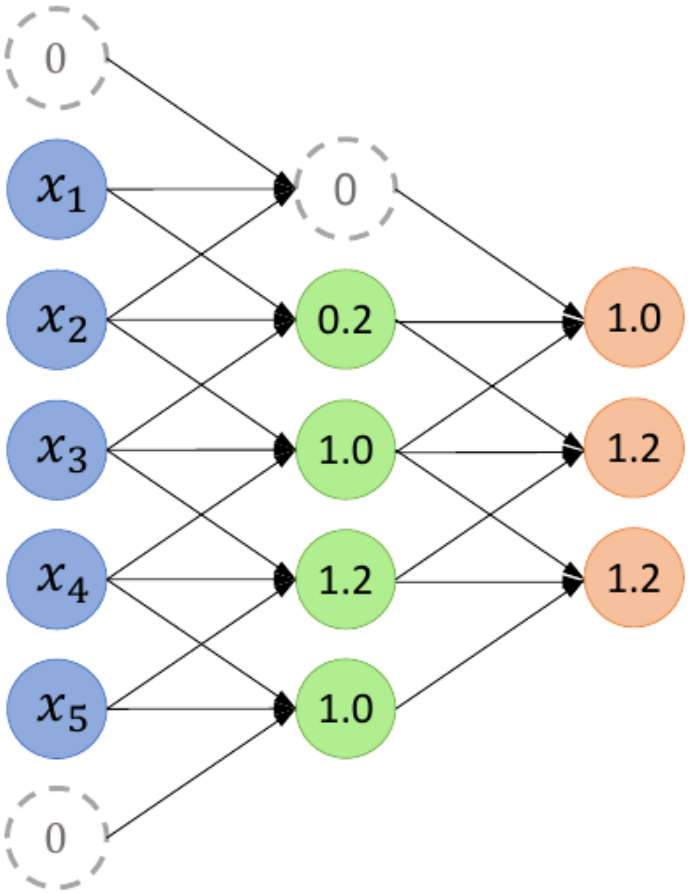
Shift input by 1 position:

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

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

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

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



Pooling

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

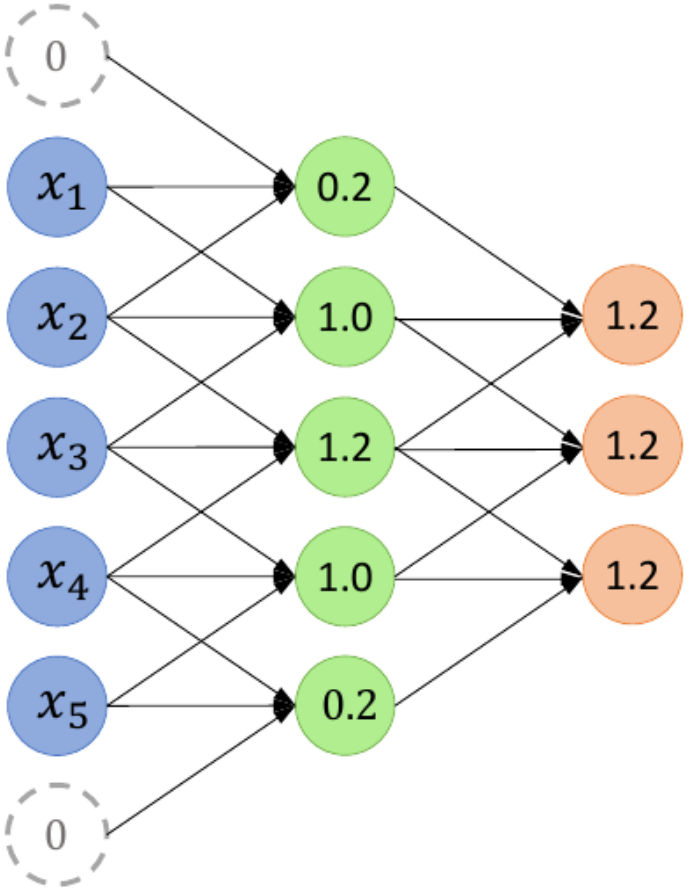
Shift input by 1 position:

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

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

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

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



Pooling

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

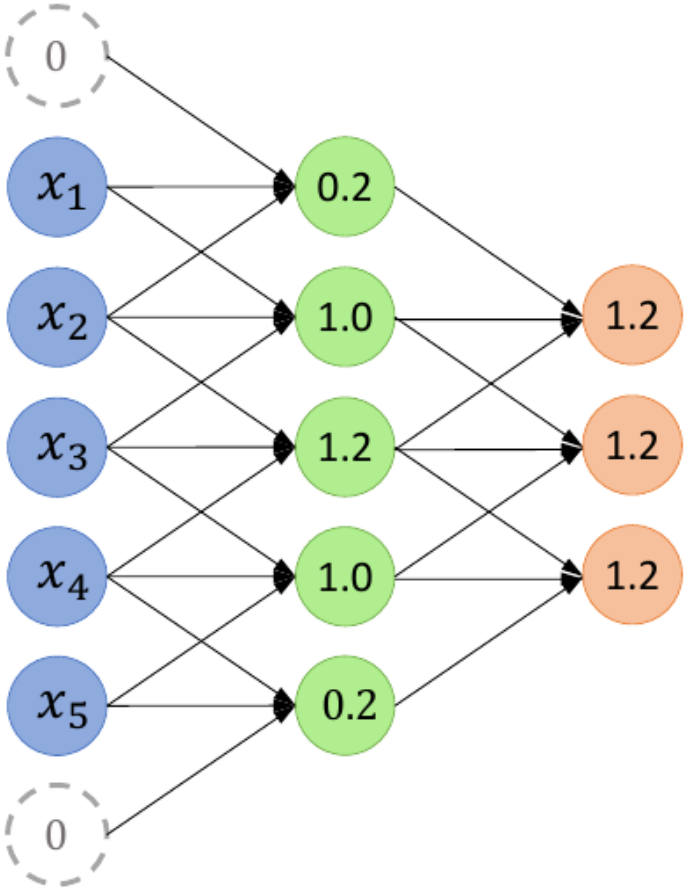
Shift input by 1 position:

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

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

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

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



Pooling

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

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

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

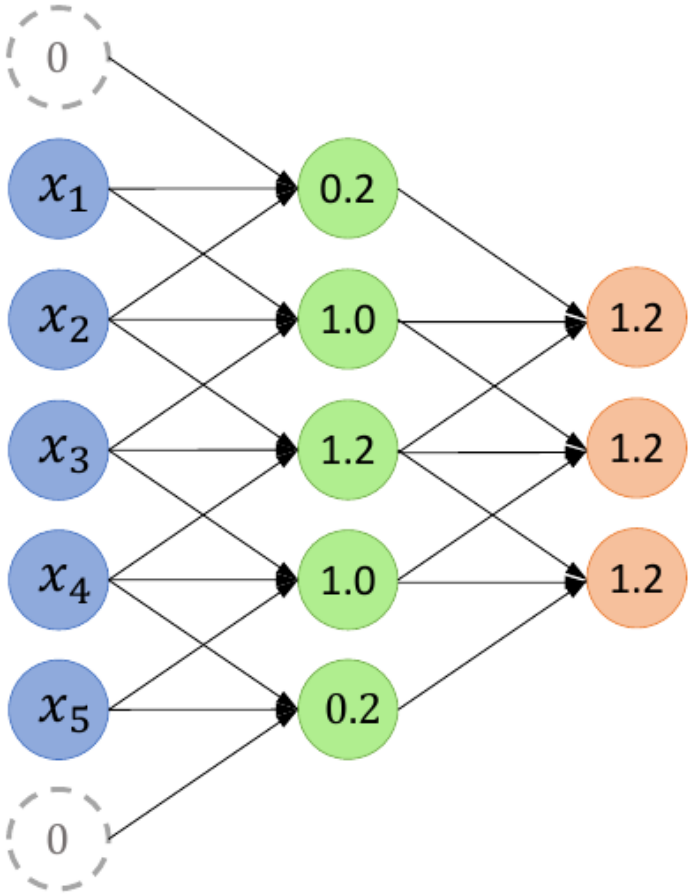
Operation to aggregate or “summarize” sub-region of input.

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)



Pooling

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

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

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

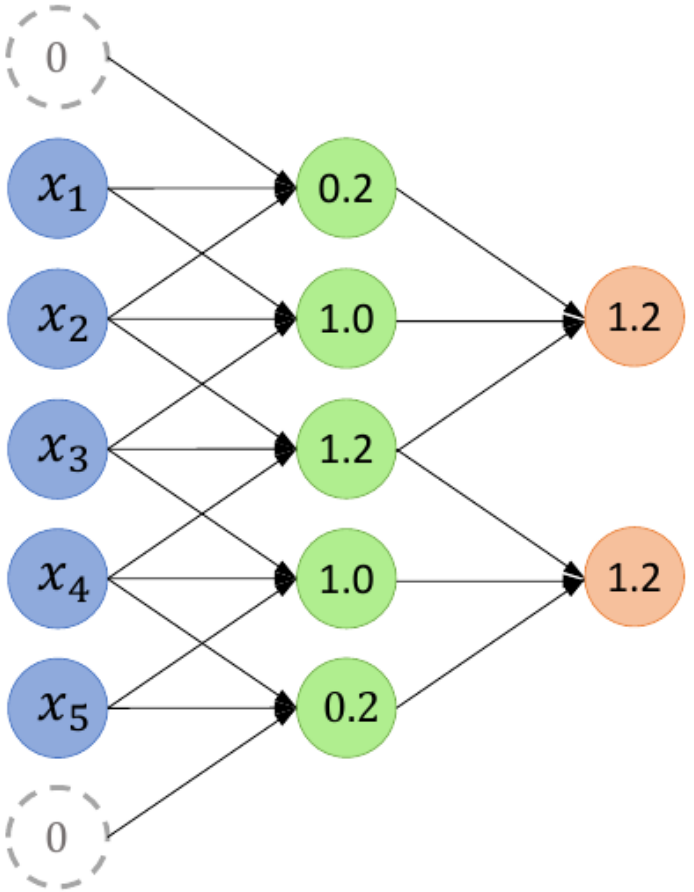
Operation to aggregate or "summarize" sub-region of input.

Shift input by 1 position:

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

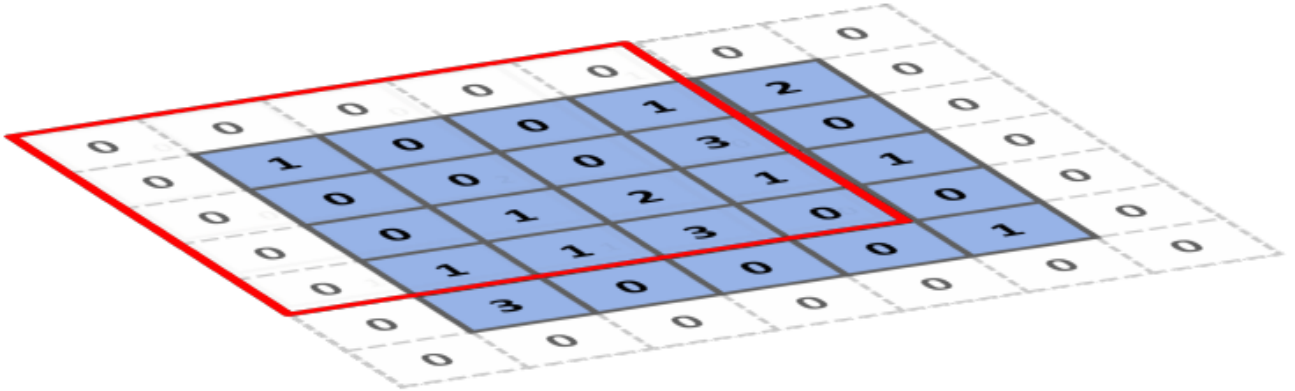
Why is pooling useful?

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



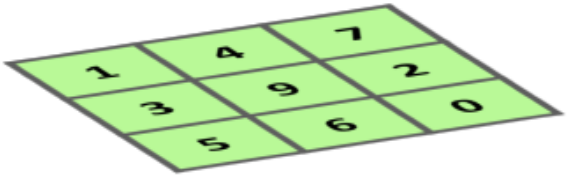
Dilated convolutions

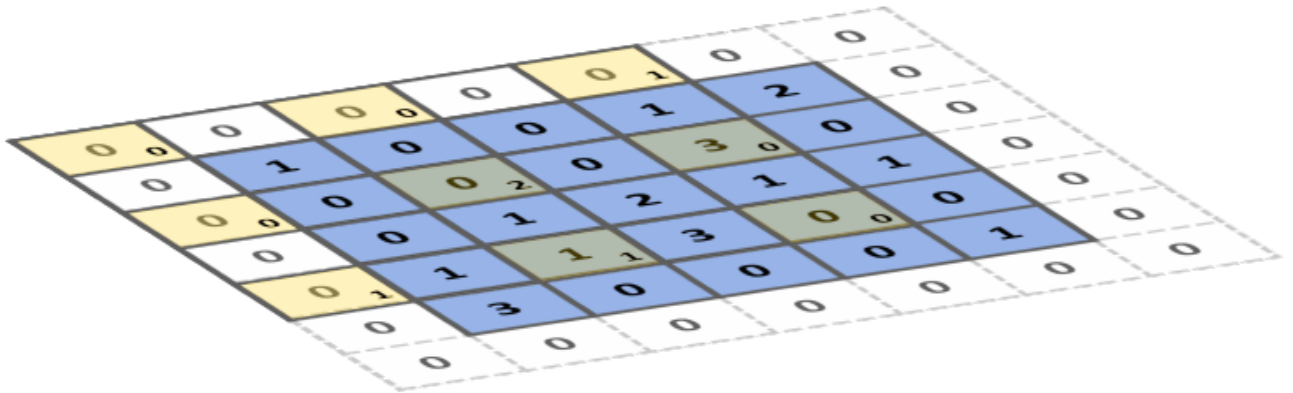
When you want to increase receptive field without increasing computation…



Dilated convolutions

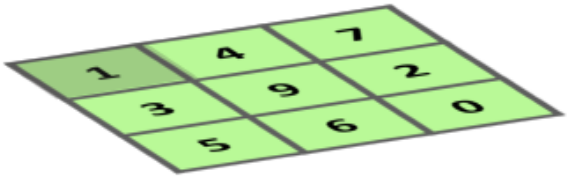
When you want to increase receptive field without increasing computation…

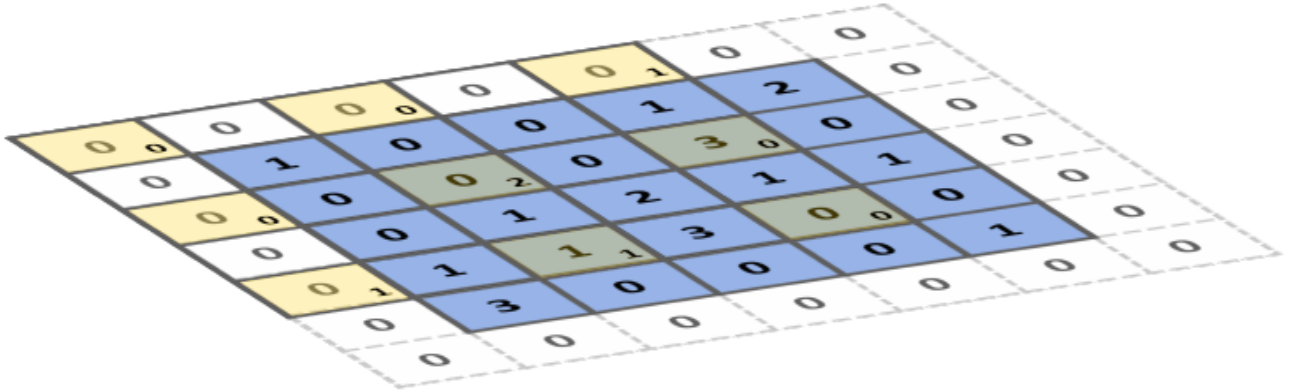




Dilated convolutions

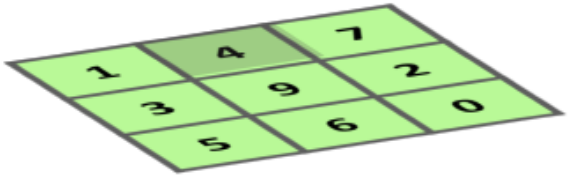
When you want to increase receptive field without increasing computation…

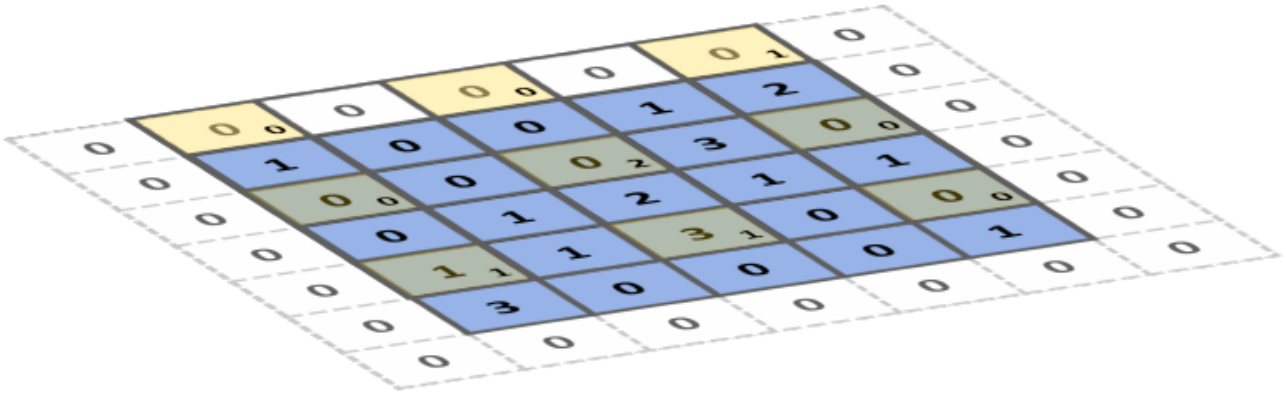




Dilated convolutions

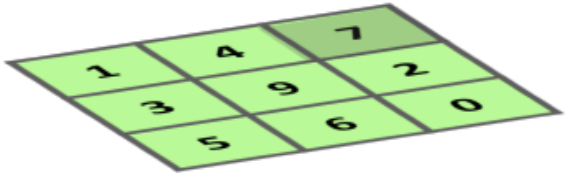
When you want to increase receptive field without increasing computation…

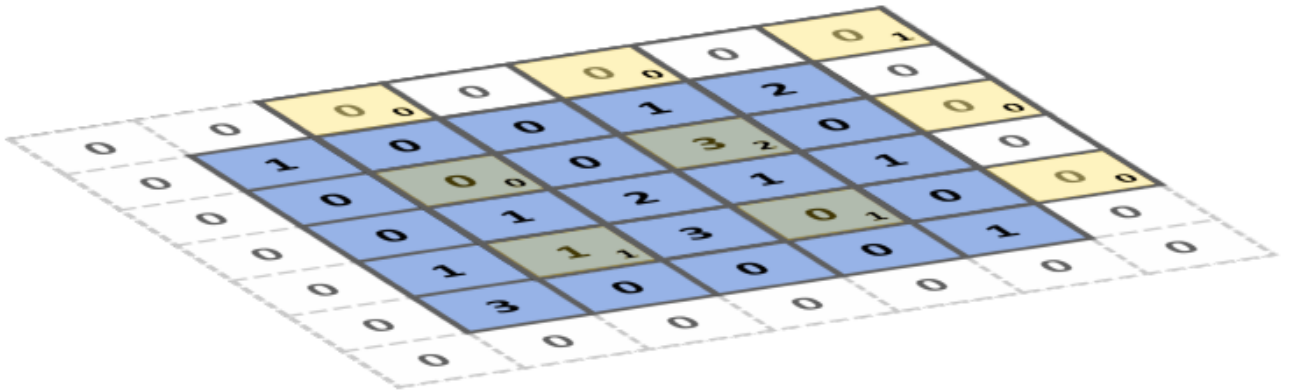




Dilated convolutions

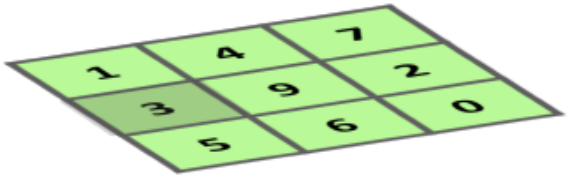
When you want to increase receptive field without increasing computation…

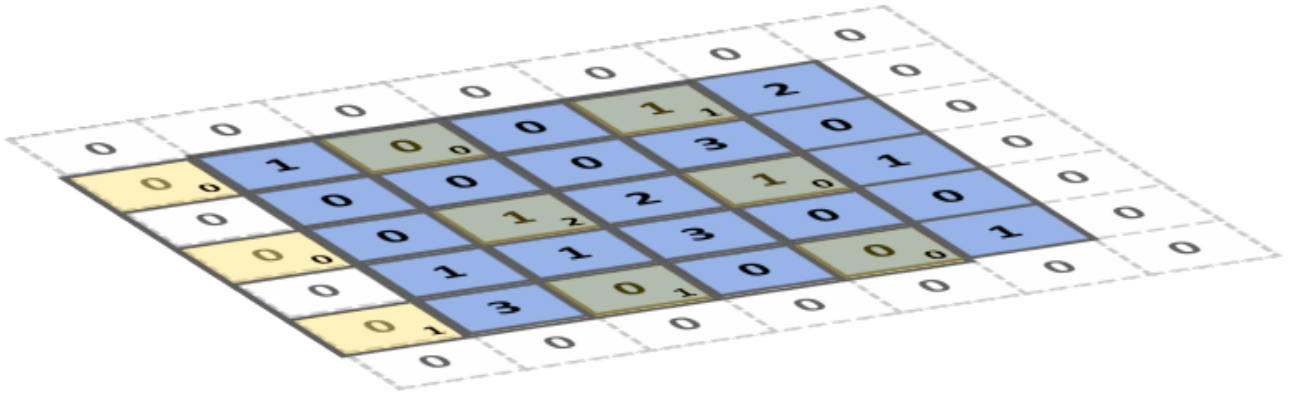




Dilated convolutions

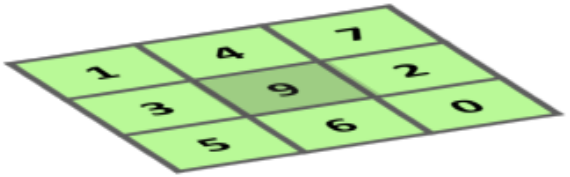
When you want to increase receptive field without increasing computation…

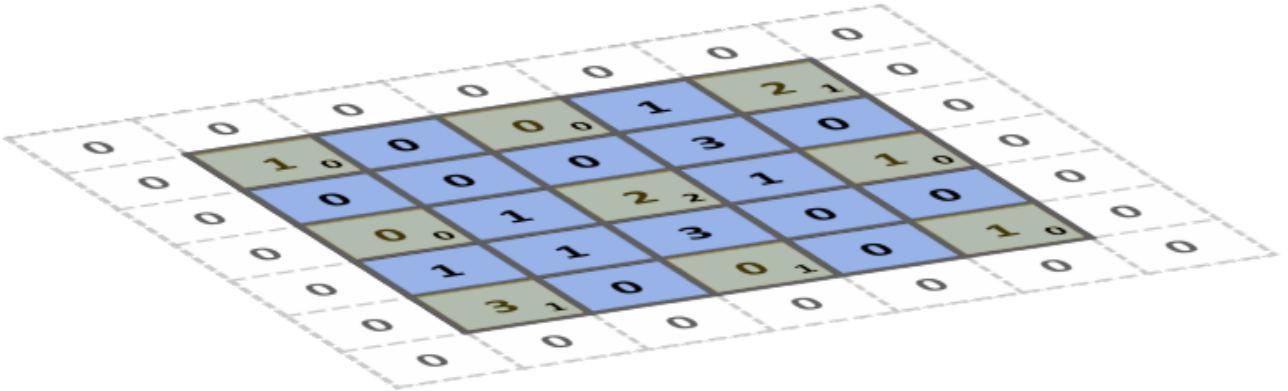




Dilated convolutions

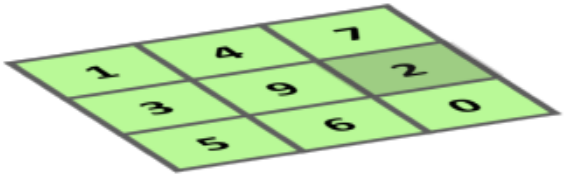
When you want to increase receptive field without increasing computation…

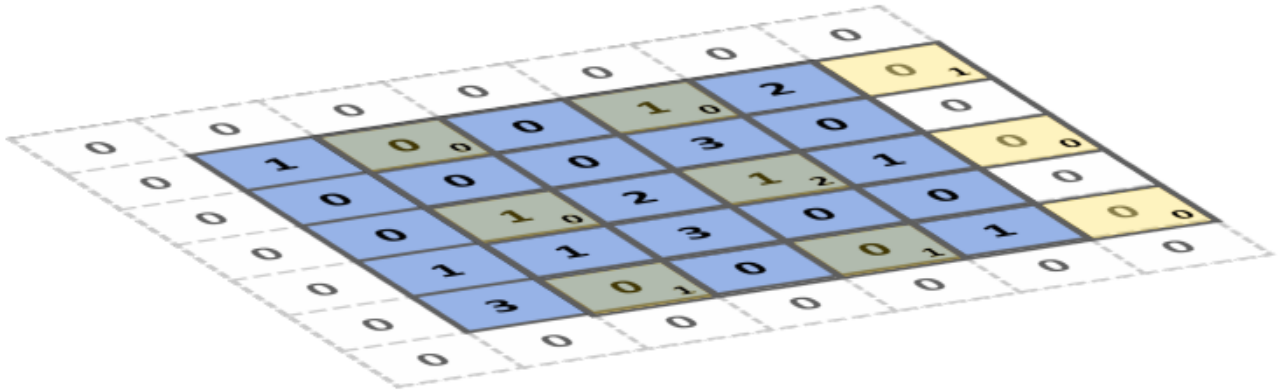




Dilated convolutions

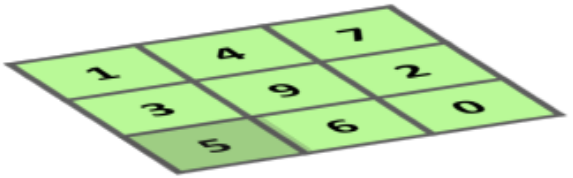
When you want to increase receptive field without increasing computation…

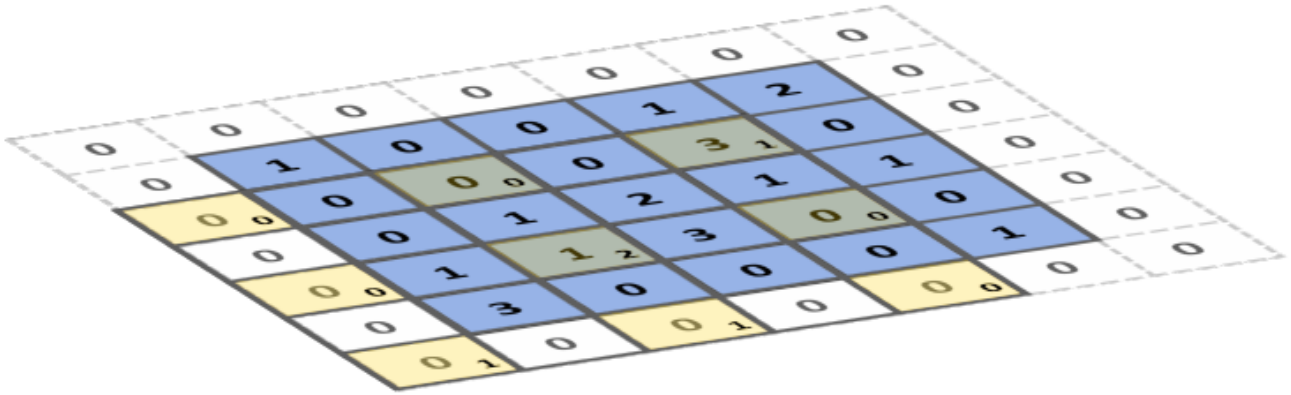




Dilated convolutions

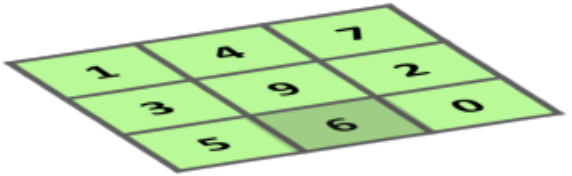
When you want to increase receptive field without increasing computation…

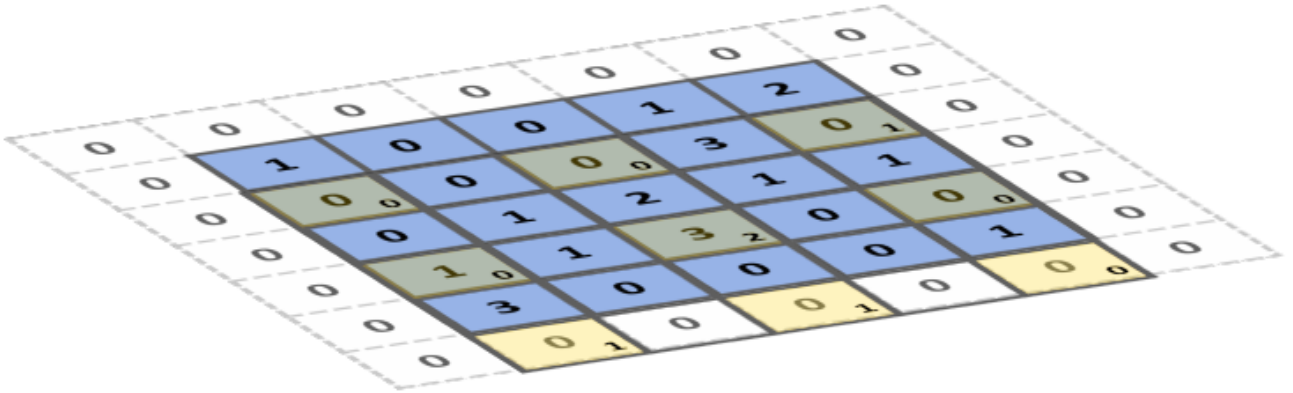




Dilated convolutions

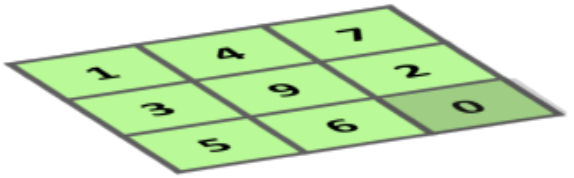
When you want to increase receptive field without increasing computation…



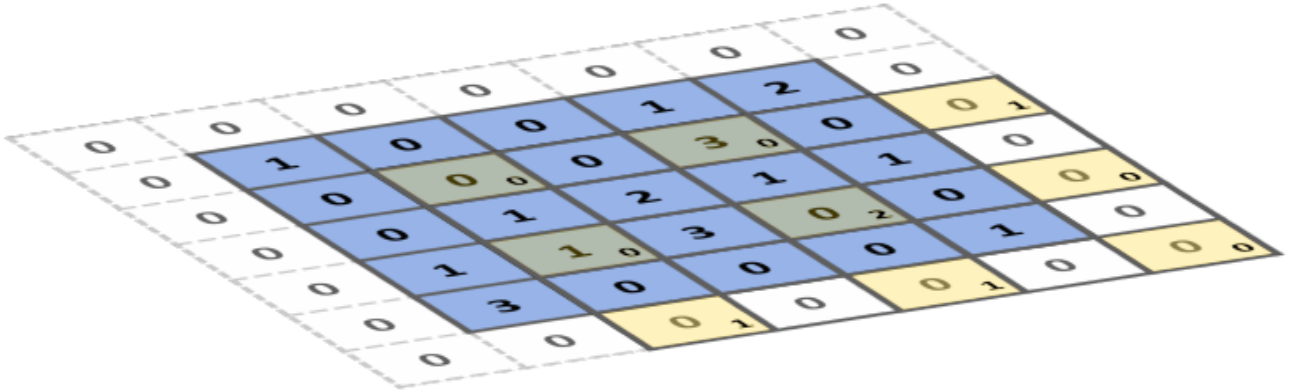


Dilated convolutions

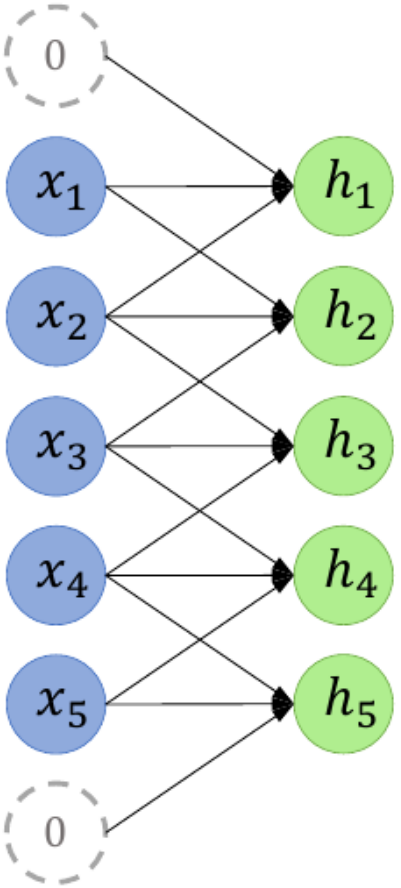
When you want to increase receptive field without increasing computation…



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

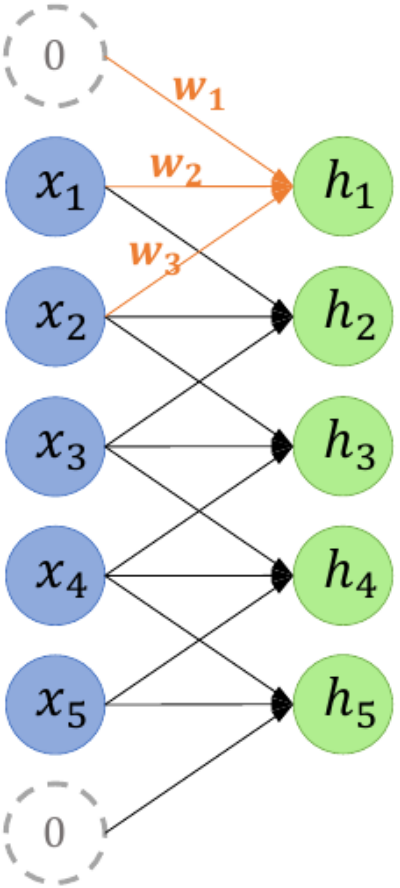


Tiled convolutions



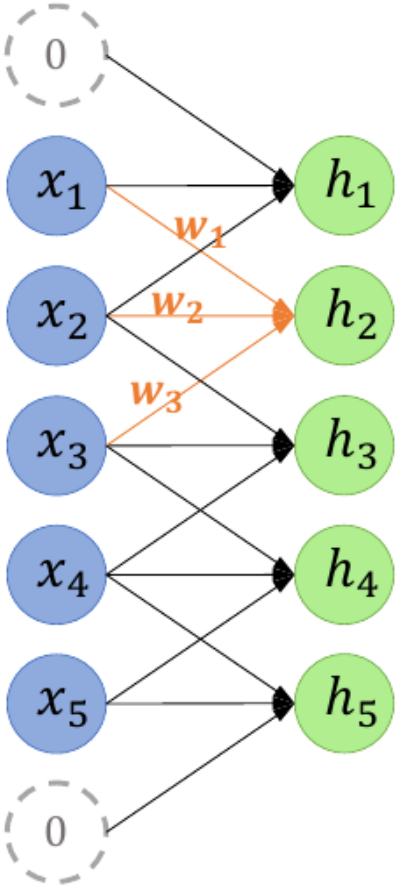
Reminder: regular convolutions slide same filter across signal

Tiled convolutions



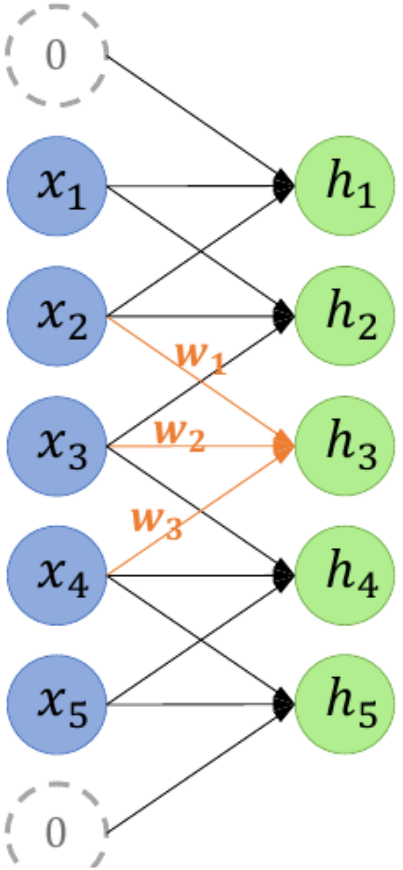
Reminder: regular convolutions slide same filter across signal

Tiled convolutions



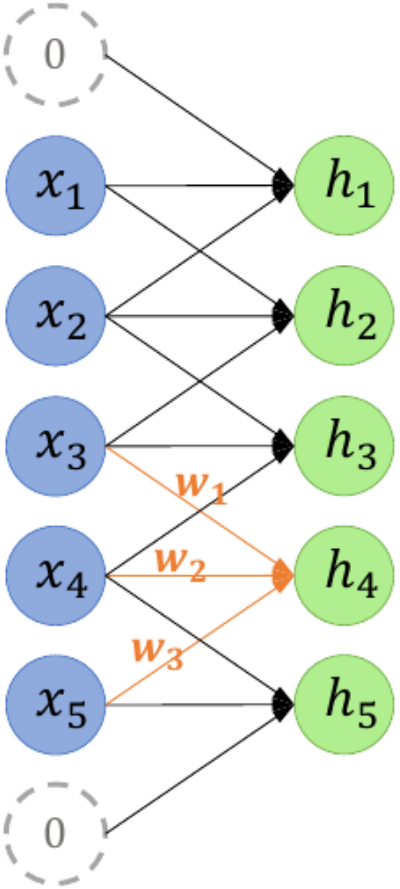
Reminder: regular convolutions slide same filter across signal

Tiled convolutions



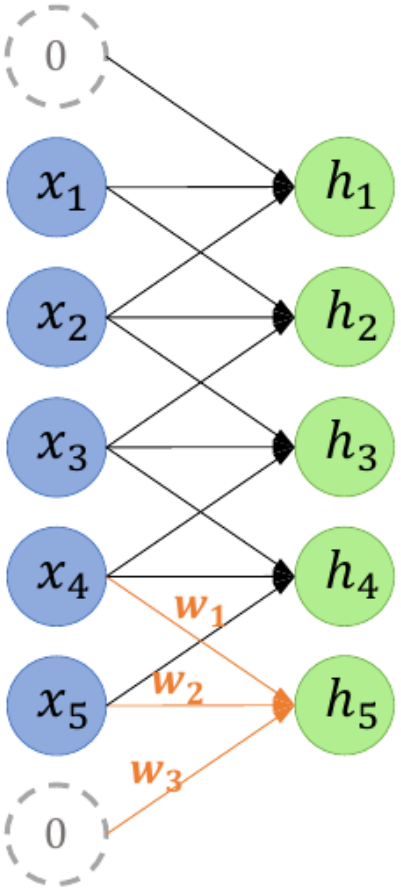
Reminder: regular convolutions slide same filter across signal

Tiled convolutions



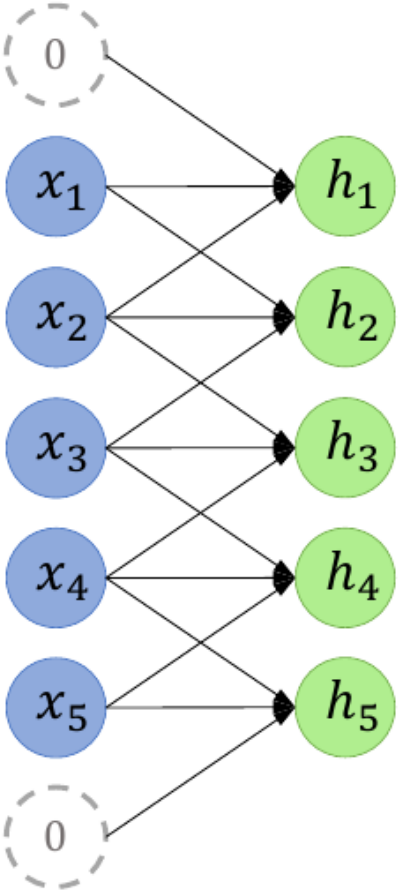
Reminder: regular convolutions slide same filter across signal

Tiled convolutions

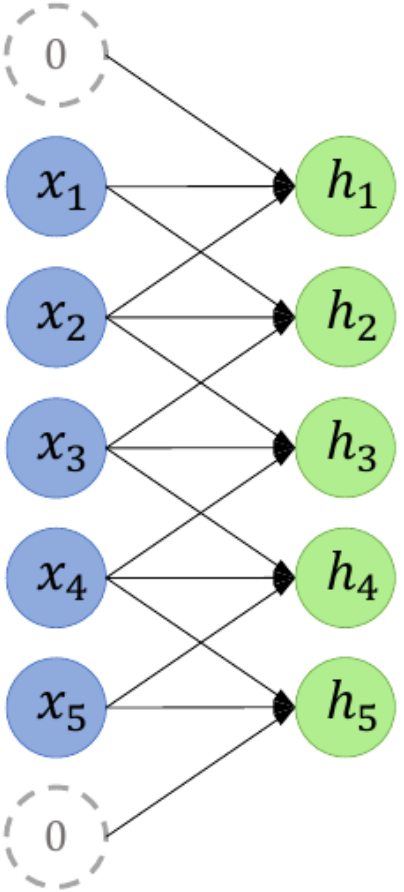


Reminder: regular convolutions slide same filter across signal

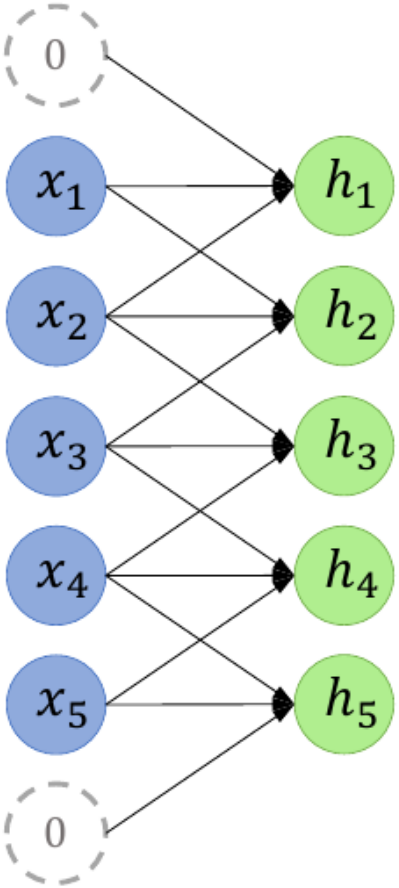
Tiled convolutions



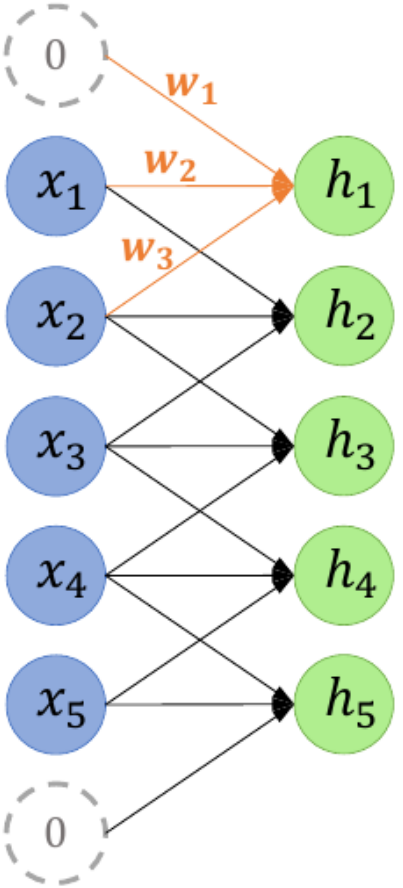
Cycle through different filters



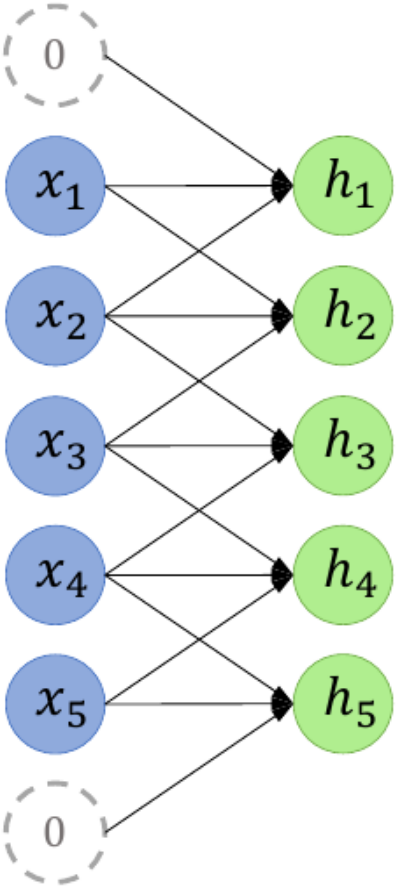
Tiled convolutions



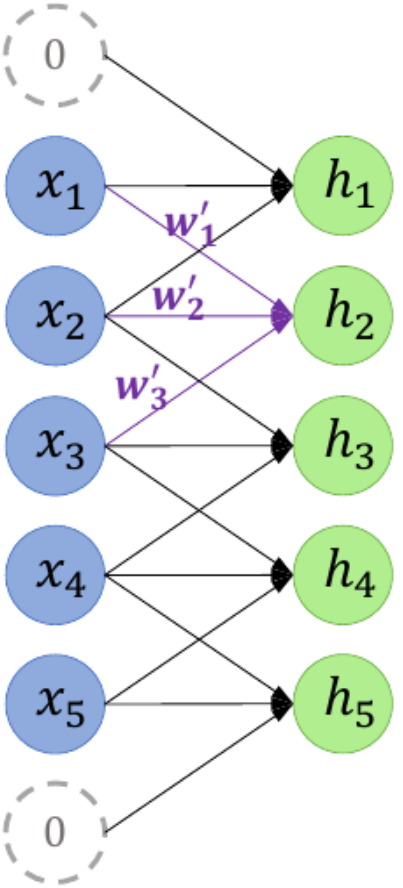
Cycle through different filters



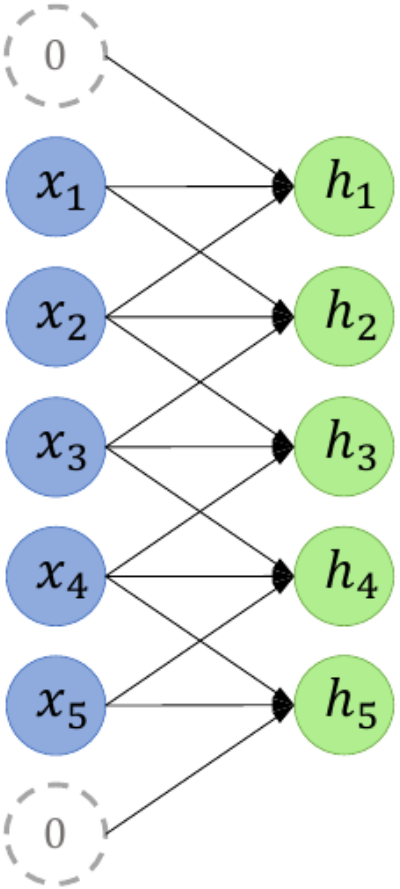
Tiled convolutions



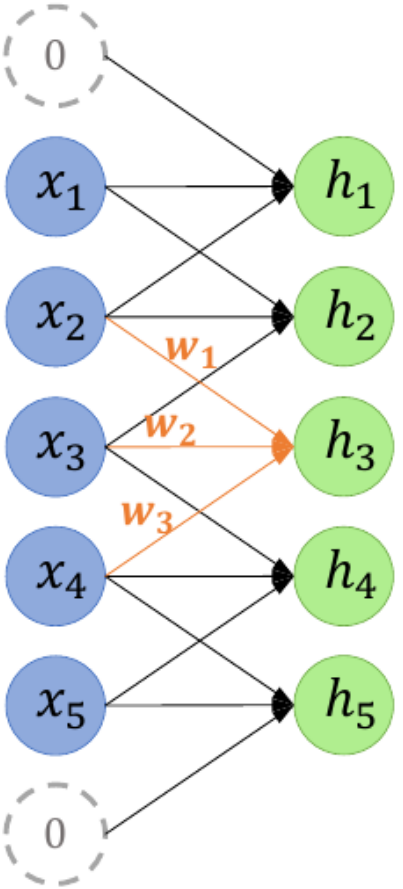
Cycle through different filters



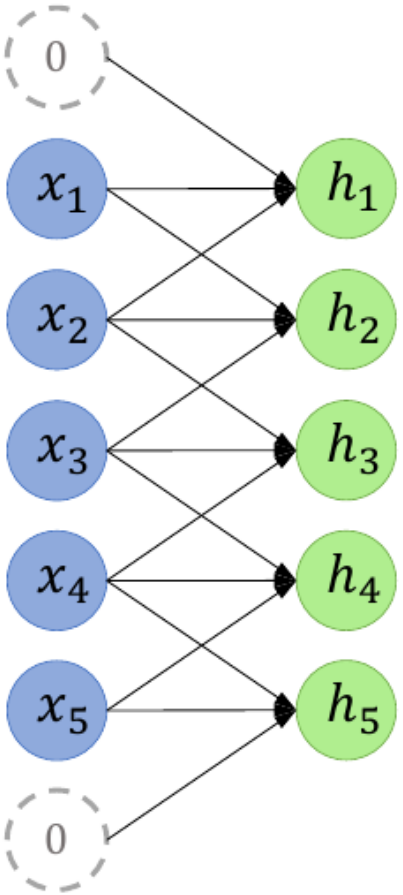
Tiled convolutions



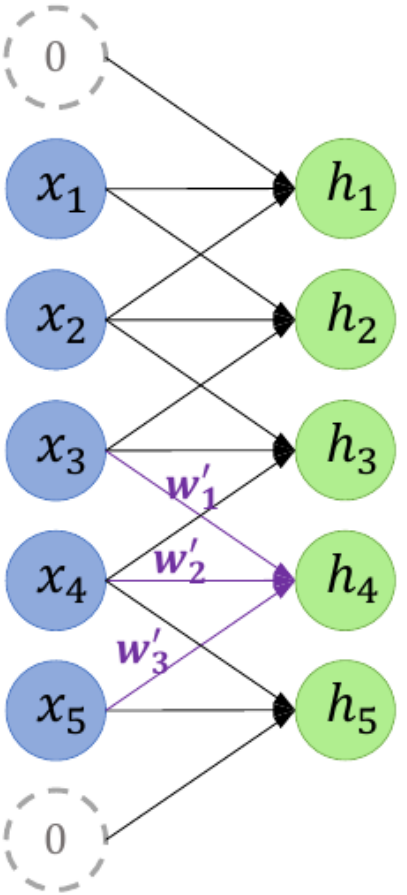
Cycle through different filters



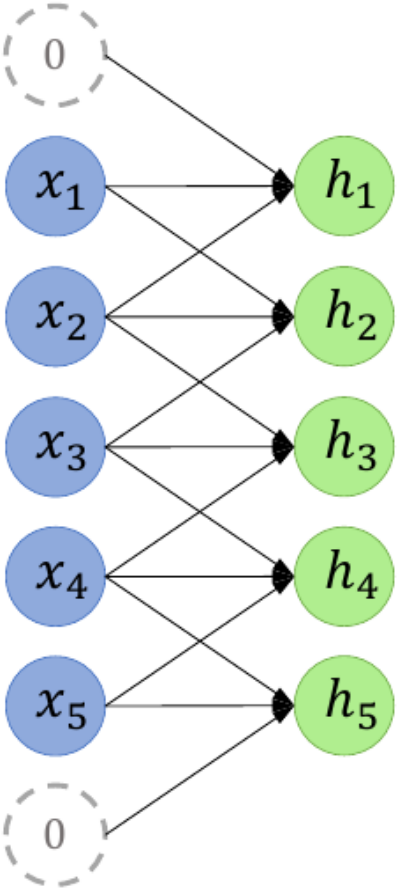
Tiled convolutions



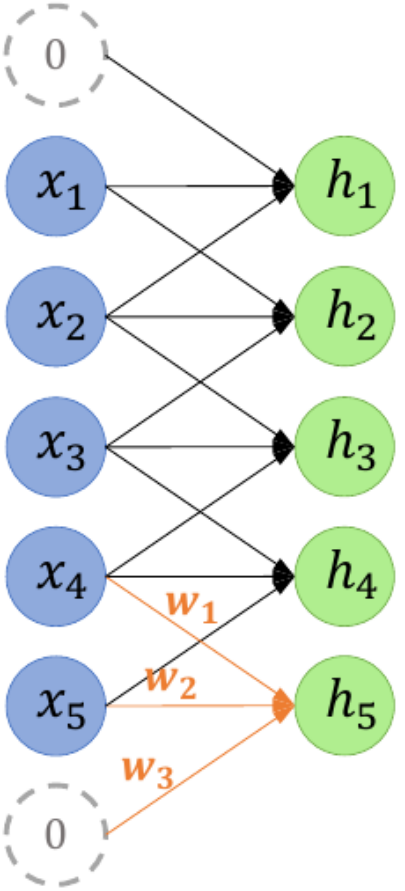
Cycle through different filters



Tiled convolutions



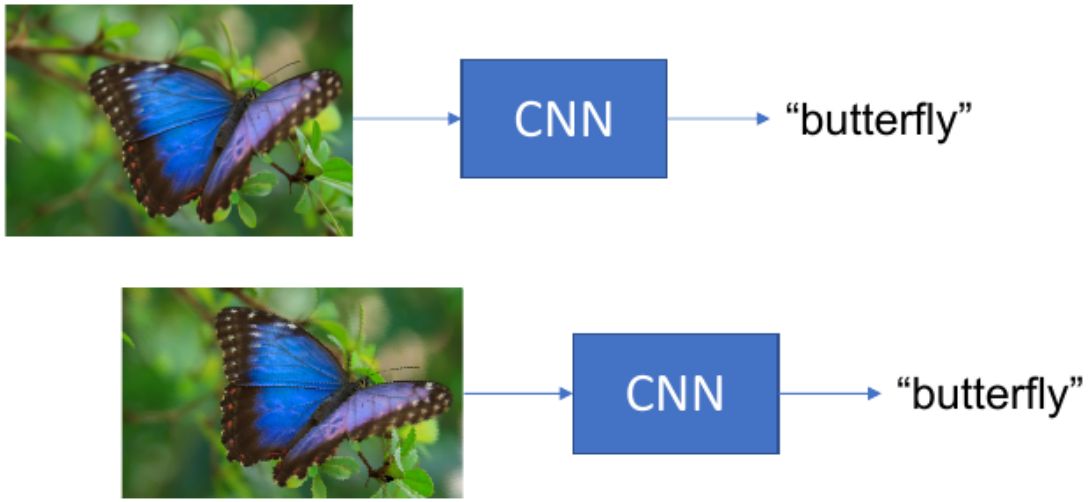
Cycle through different filters

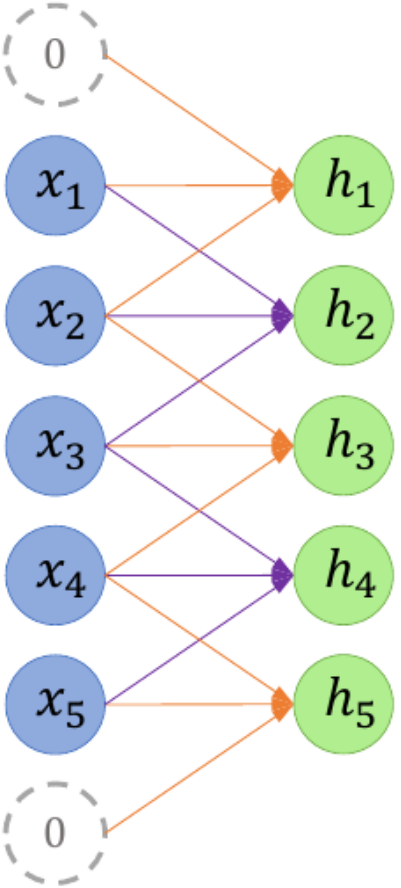


Tiled convolutions

Why is this useful?

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





ACKNOWLEDGEMENTS

Adapted version of the DeepLearning Indaba 2018