Institut
des algorithmes
d'apprentissage
de Montréal

MILA

Convolution arithmetic

Vincent Dumoulin, Francesco Visin

Introduction

Why should I care?

- Convolutional neural networks are not a new idea (LeCun *et al.* did it in the nineties).
- CNNs started getting a *lot* of traction when Krizhevsky *et al.* used a deep CNN to achieve SOTA on ImageNet in 2012.
- Used pretty extensively in computer vision.

Learning to use CNNs is an intimidating experience.

Complexity: NN vs. CNN

Fully-connected networks

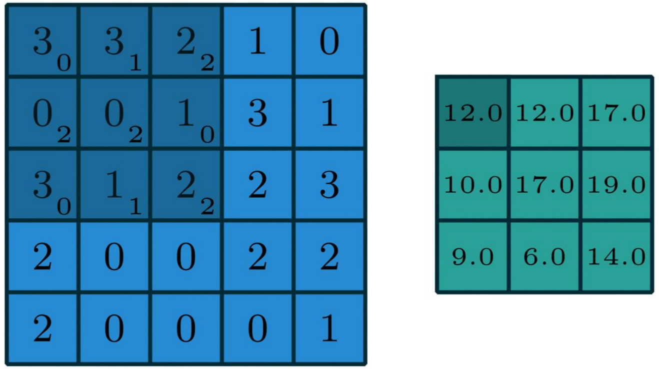
$$\begin{tikzcd}
\text{W.shape[1]} \arrow[d] \\
\text{Output size}
\end{tikzcd}$$

Convolutional neural networks

$$\begin{tikzcd}
 & \text{Kernel shape} \arrow[d] & \\
\text{Input shape} \arrow[r] & \text{Output shape} & \text{Strides} \arrow[l] \\
 & \text{Zero padding} \arrow[u] &
\end{tikzcd}$$

Back to basics: discrete convolutions

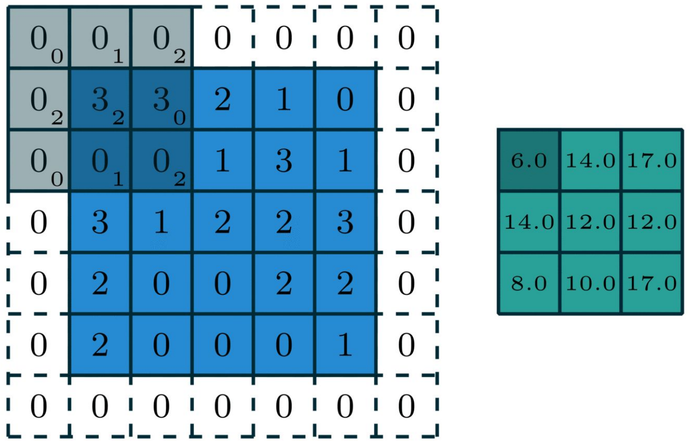
- **Kernel** (*shaded area*) slides over the **input feature map** (*blue*).
- At each kernel position, elementwise product is computed between the kernel and the overlapped input subset. Result is summed up.
- Results constitute the **output feature map** (*cyan*).



Back to basics: discrete convolutions

Properties affecting a discrete convolution:

- ***i*** : input size
- ***k*** : kernel size
- ***s*** : stride (distance between consecutive positions of the kernel)
- ***p*** : zero padding (number of zeros inserted at the beginning and at the end of an axis)



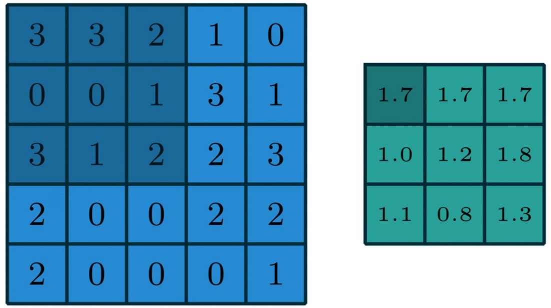
i = 5, *k* = 3, *s* = 2, *p* = 1

Back to basics: pooling

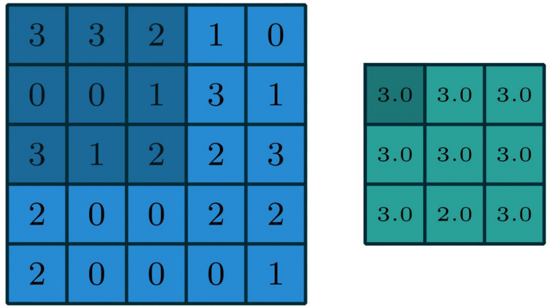
Pooling: sliding a window over the input and summarizing the content of the window as a single number (e.g., average pooling, max pooling).

Properties affecting pooling:

- ***i*** : input size
- ***k*** : pooling window size
- ***s*** : stride (distance between consecutive positions of the pooling window)



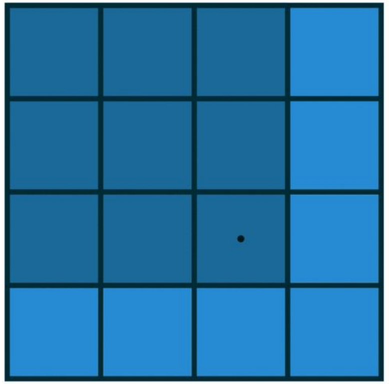
Average pooling



Max pooling

Convolution arithmetic

Counting kernel positions



- Kernel starts at top left side of the input
- Count the number of hops to go to the right side
- Add 1 to account for the original kernel position
- Repeat for the up-down axis

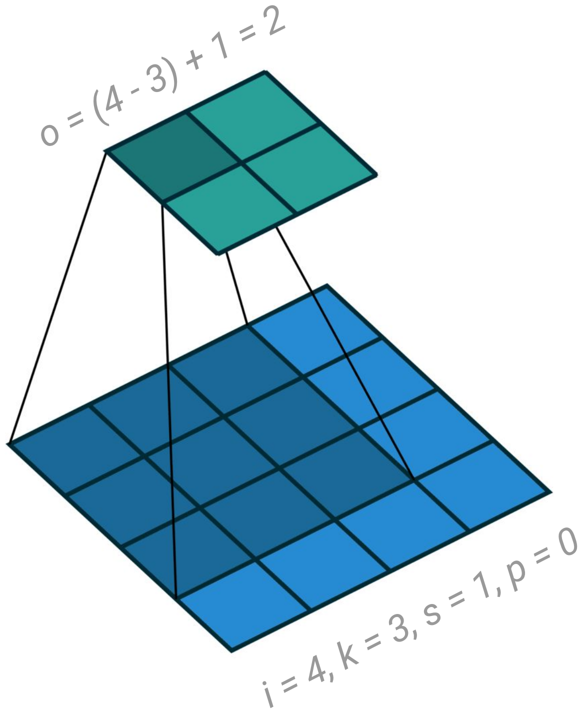
Yields

$$o = (i - k) + 1$$

No padding, no strides

Count the number of possible kernel positions in both axes:

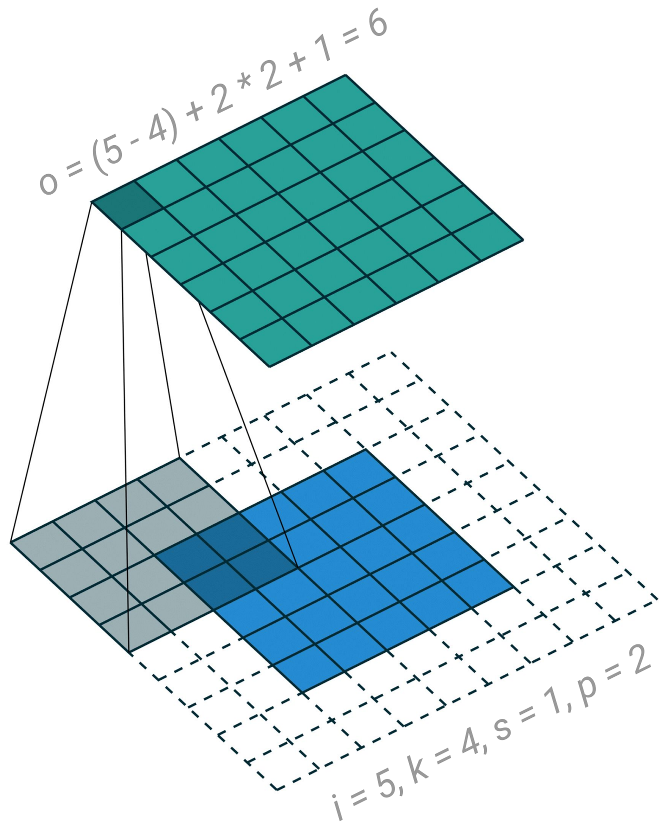
$$o = (i - k) + 1$$



Arbitrary padding, no strides

Zero padding changes the effective input size (it adds *2p* to the input size):

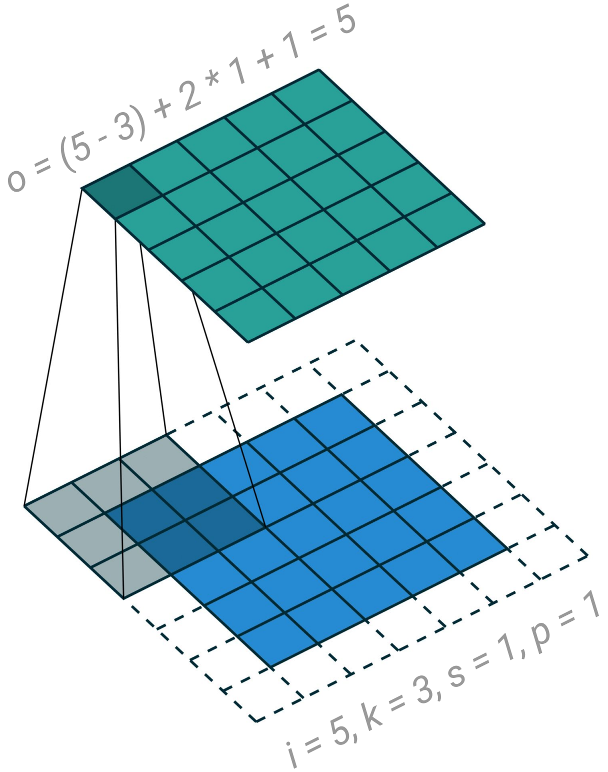
$$o = (i - k) + 2p + 1$$



Half (same) padding, no strides

If k is odd ($k = 2n + 1$), and if $p = k \mathbin{//} 2 = n$, the output size is equal to the input size:

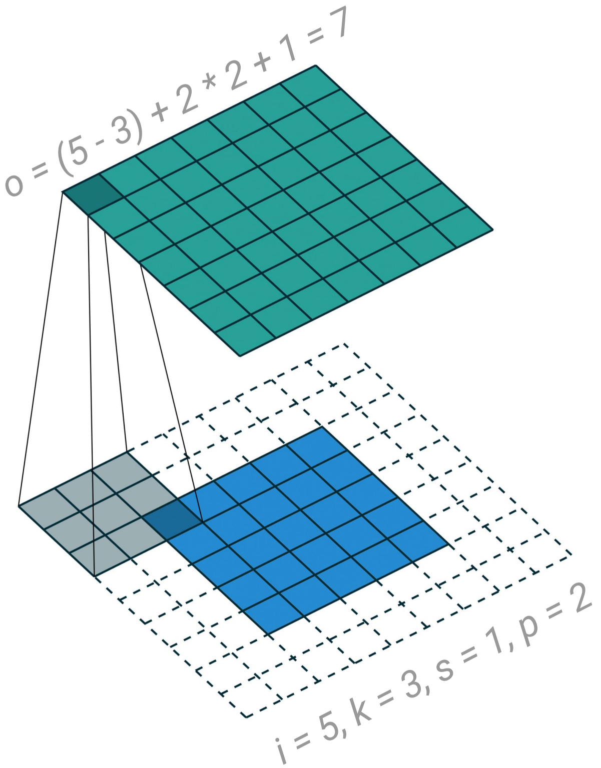
$$\begin{aligned} o &= (i - k) + 2p + 1 \\ &= i - 2n - 1 + 2n + 1 \\ &= i \end{aligned}$$



Full padding, no strides

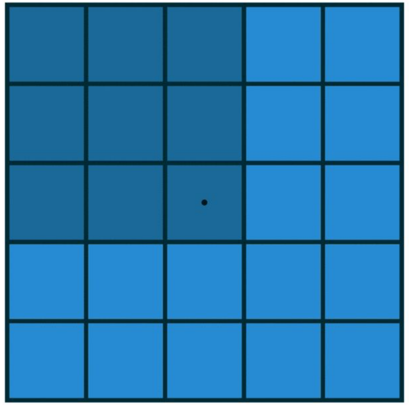
If *p* = *k* - *1*, the size **increases** by *k* - *1*:

$$
\begin{aligned}
o &= (i - k) + 2p + 1 \\
&= i - (k - 1) + 2(k - 1) \\
&= i + (k - 1)
\end{aligned}
$$



What about those pesky strides?

Counting kernel positions (strides)



- Kernel starts at top left side of the input
- Count the number of hops of size *s* to go to the right side
- Add 1 to account for the original kernel position
- Repeat for the up-down axis

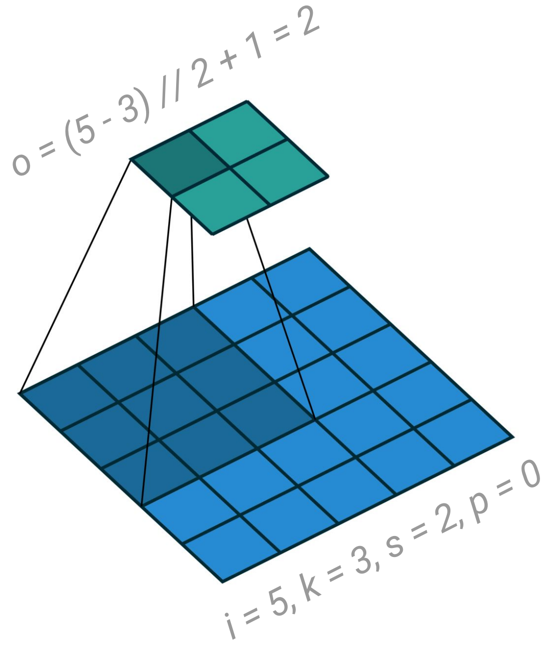
Yields

$$o = (i - k) \mathbin{//} s + 1$$

No padding, strides

Count the number of possible kernel positions in both axes:

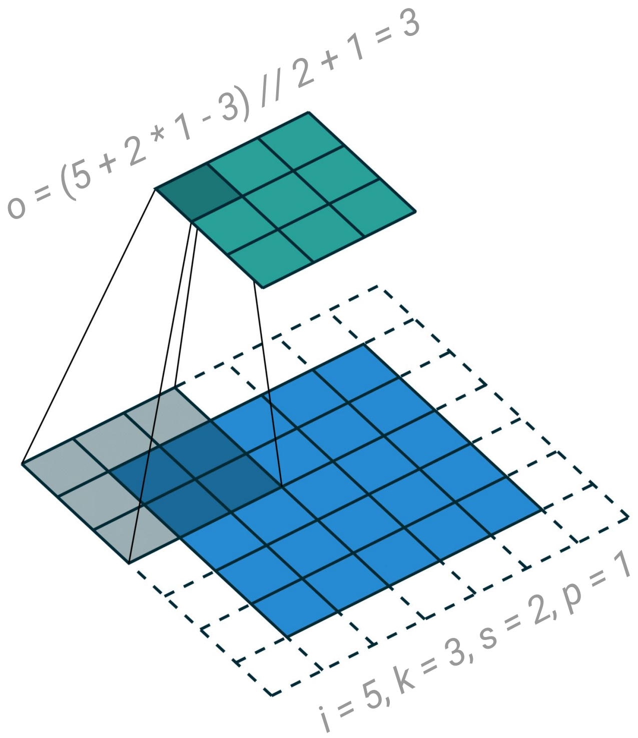
$$o = (i - k) \mathbin{//} s + 1$$



Arbitrary padding, strides

Combine previous rule with the fact that zero padding adds *2p* to the input size:

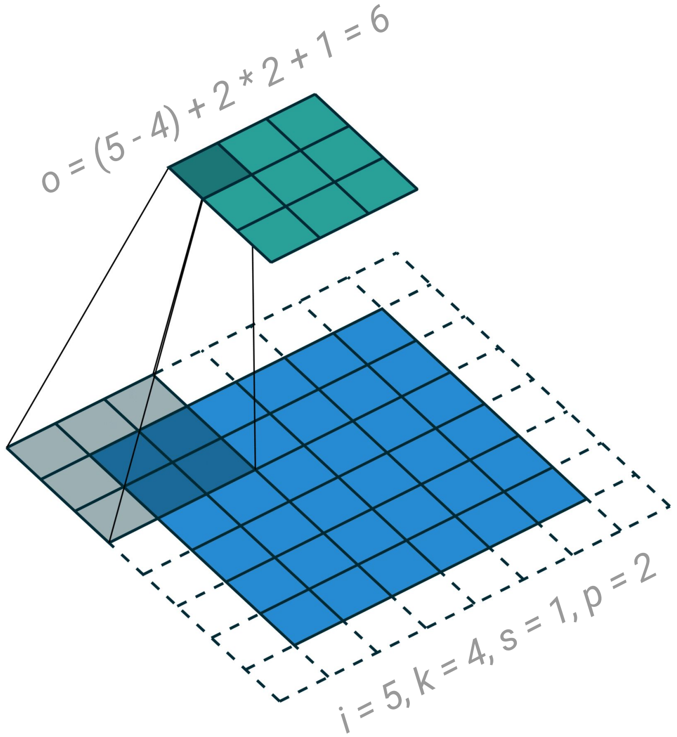
$$o = (i + 2p - k) \,//\, s + 1$$



Arbitrary padding, strides

Note: sometimes, *i + 2p - k* is **not** a multiple of *s*.

This means that for *s > 1* multiple input sizes share the same output size.



Pooling arithmetic

Freebie!

Did you notice we did ***not*** talk about convolutions in the previous section?

Transposed convolution arithmetic

Transposed convolutions

What is it for, anyways?

Used for

- Gradient backpropagation
- Decoder layers in convolutional autoencoder
- Project feature maps into a higher-dimensional space (*upsampling*)

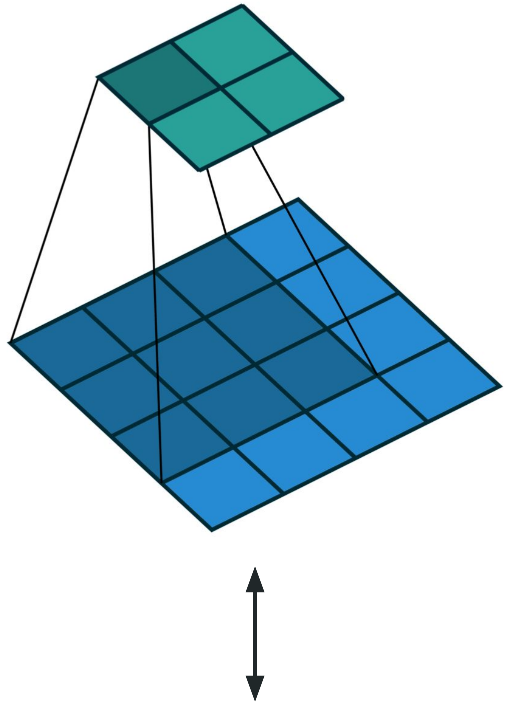
Convolution: matrix view

A convolution can be represented as a sparse and weight-sharing matrix.

Forward pass: 16-D$\rightarrow$ **C** $\rightarrow$ 4-D

Backward pass: 4-D $\rightarrow$ $\mathbf{C}^{T}$ $\rightarrow$ 16-D

Both computations share the same connectivity pattern.



$$\mathbf{C} = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Transposed convolution: matrix view

A transposed convolution **swaps the forward and backward passes**.

Forward pass: 4-D$\rightarrow$ $\mathbf{C}^T$ $\rightarrow$ 16-D

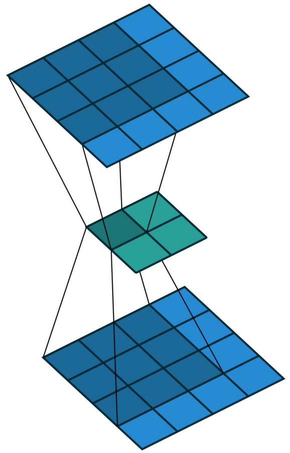
Backward pass: 16-D $\rightarrow$ $\mathbf{C}$ $\rightarrow$ 4-D

In other words, a transposed convolution is the gradient of some convolution with respect to its input.

How can you wrap your mind around that?

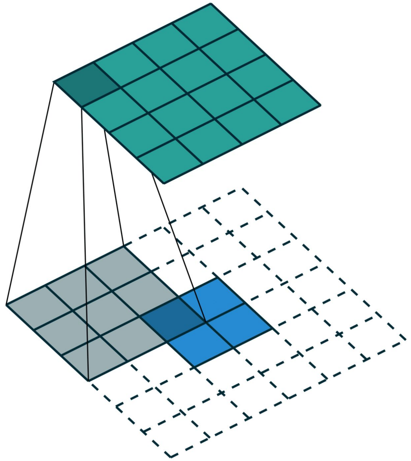
Visualizing transposed convolutions

Transposed convolutions can be conceptualized in terms of convolutions by processing the input in a clever fashion.



Convolution followed by its transpose,

i = 4, k = 3, s = 1, p = 0



Convolution equivalent to the transposed convolution,

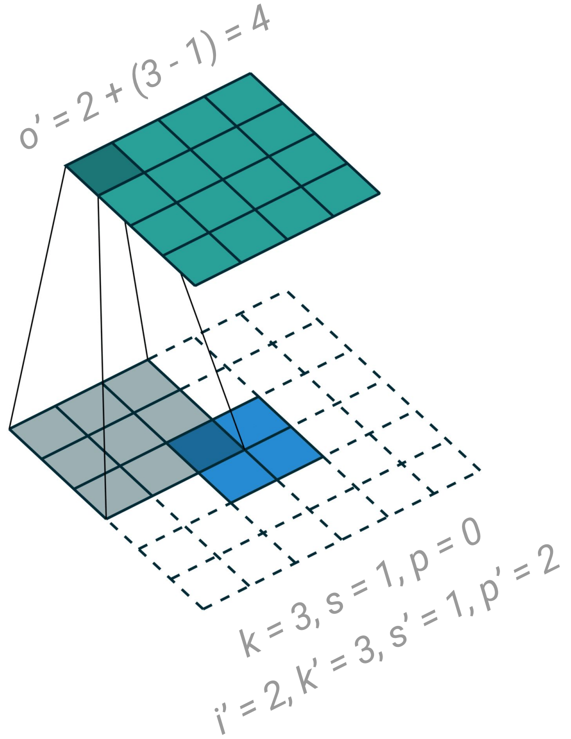
i’ = 6, k = 3, s = 1, p = 0

[No padding, no strides]^T

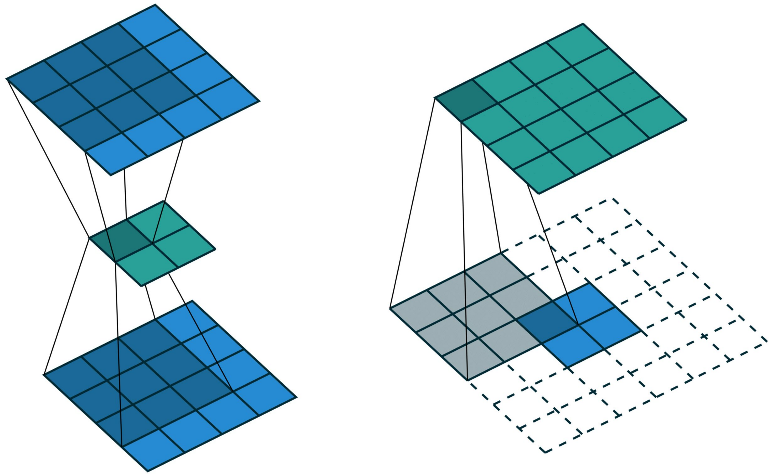
Equivalent convolution is defined by

$$k' = k, \quad s' = 1, \quad p' = k - 1,$$

$$o' = i' + (k - 1)$$



Why zero padding?



We are trying to match connectivity patterns.

Zero padding lets the kernel move fully over an input unit.

[Arbitrary padding, no strides]^T

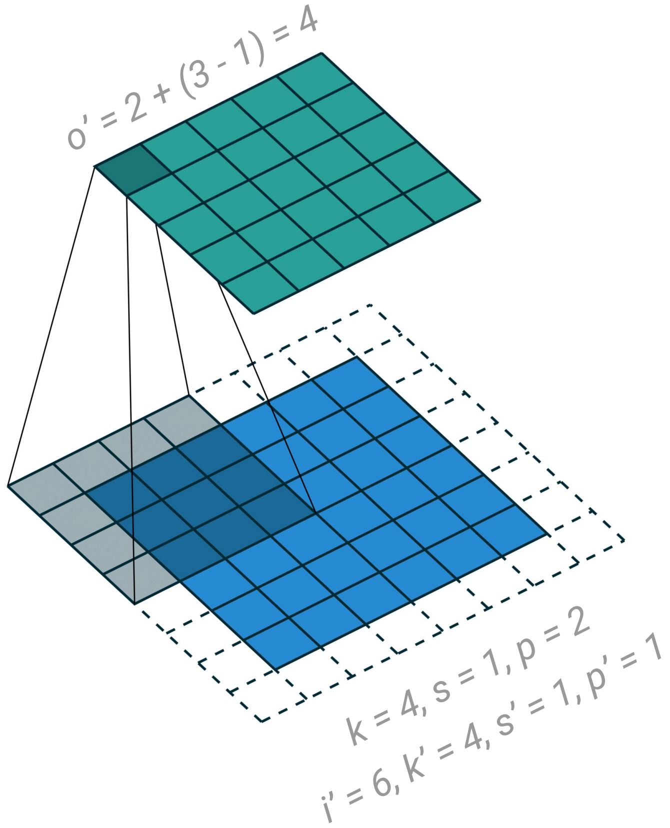
[No padding, no strides]^T: equivalent convolution *adds* zero padding.

[Arbitrary padding, no strides]^T: equivalent convolution *removes* padding.

Equivalent convolution is defined by

$$k' = k, \quad s' = 1, \quad p' = k - p - 1,$$

$$o' = i' + (k - 1) - 2p$$

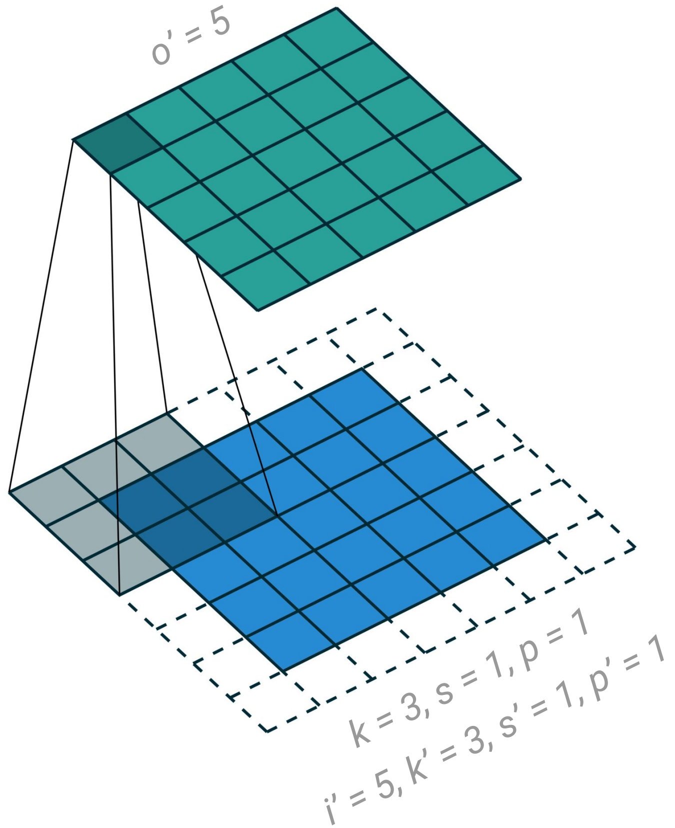


[Half (same) padding, no strides]^T

Equivalent convolution is [half padding, no strides] itself!

$$k' = k, \quad s' = 1, \quad p' = p,$$

$$o' = i'$$



[Full padding, no strides]^T

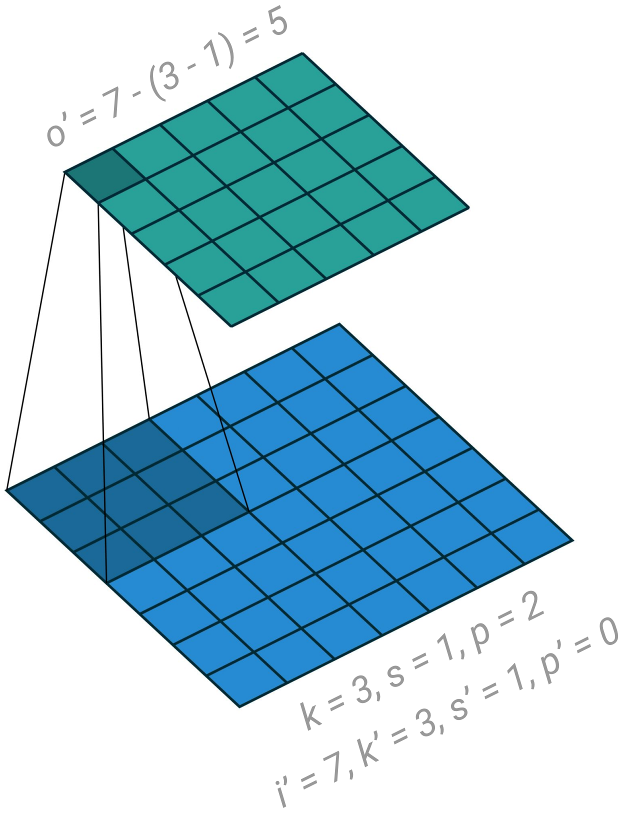
[No padding, no strides]^T = [Full padding, no strides]

[Full padding, no strides]^T = [No padding, no strides]

Equivalent convolution is defined by

$$k' = k, \quad s' = 1, \quad p' = 0,$$

$$o' = i' - (k - 1)$$



[No padding, strides]^T

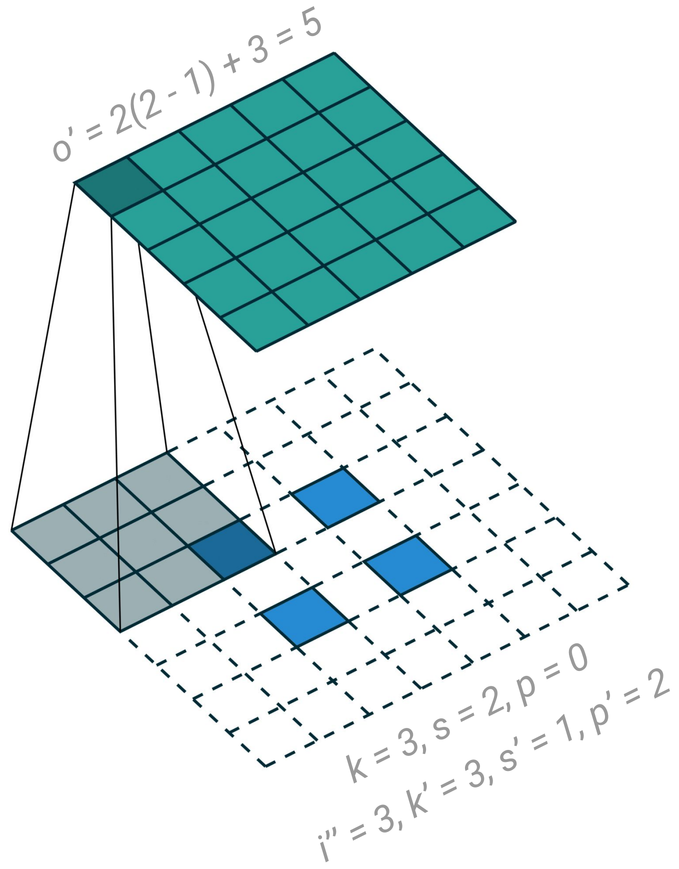
Equivalent convolution is defined by

$$k' = k, \quad s' = 1, \quad p' = k - 1,$$

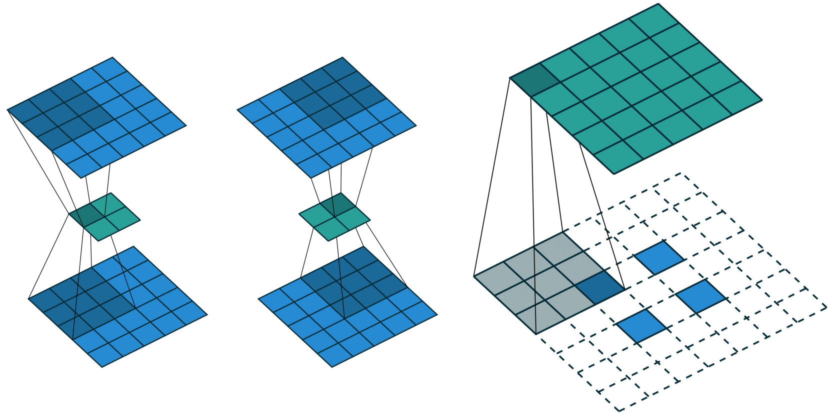
$$i'' = (s - 1)(i' - 1)$$

(we insert $s - 1$ zeros between inputs)

$$o' = s(i' - 1) + k$$



Why insert zeros?



We are trying to match connectivity patterns.

Inserting zeros adds the needed space to compensate for the strides.

[Arbitrary padding, strides]^T

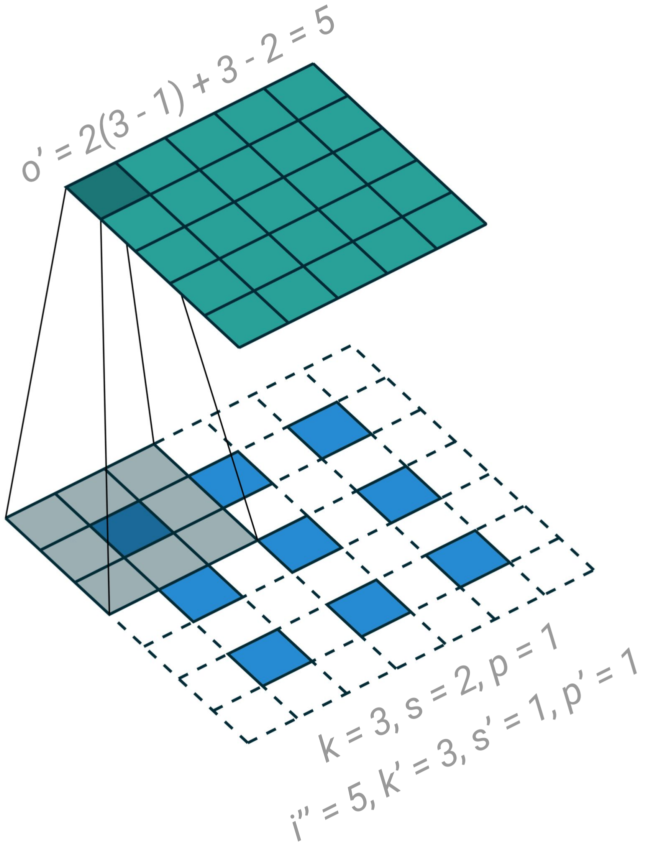
Equivalent convolution is defined by

$$k' = k, \quad s' = 1, \quad p' = k - p - 1,$$

$$i'' = (s - 1)(i' - 1)$$

(we insert $s - 1$ zeros between inputs)

$$o' = s(i' - 1) + k - 2p$$



[Arbitrary padding, strides]^T

Equivalent convolution is defined by

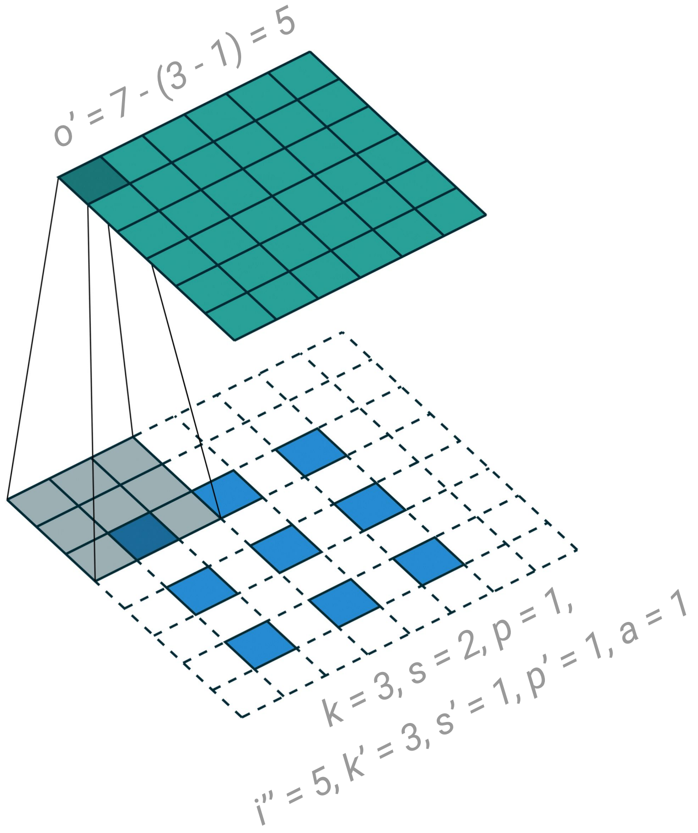
$$k' = k, \quad s' = 1, \quad p' = k - p - 1,$$

$$i'' = (s - 1)(i' - 1),$$

$$a = (i + 2p - k) \bmod s$$

(we insert $s - 1$ zeros between inputs)

$$o' = s(i' - 1) + a + k - 2p$$



Acknowledgements, guide and code

Many thanks to David Warde-Farley, Guillaume Alain and Caglar Gulcehre for their valuable feedback, as well as everyone else who offered input after the guide was put on arXiv.

Special thanks to Ethan Schoonover, creator of the Solarized color scheme, whose colors were used for the figures.

You can find the guide on arXiv here:

http://arxiv.org/abs/1603.07285

You can find the code for the guide and the figures here:

https://github.com/vdumoulin/conv_arithmetic