Deep Learning

LM Computer Science, Data Science, Cybersecurity

2nd semester - 6 CFU

References:

Deep Learning Book (main book)

Mitchell (machine learining concepts)

Bishop (machine learning)

Convolutional Neural Networks

(chapter 9)

Motivation

Images have three properties that suggest the need for specialized architecture.

1. they are **high-dimensional**, e.g. 224×224 RGB values (i.e., 150,528 input dimensions). Hidden layers in fully connected networks are generally larger than the input size, and so even for a shallow network, the number of weights would exceed $150{,}528^2$ or 22 billion. Obvious practical problems in terms of the required training data, memory, and computation.

2. **Nearby image pixels are statistically related**. Fully connected networks have no notion of "nearby" and treats the relationship between every input equally; if the pixels of the training and test images were randomly permuted in the same way, the network could still be trained with no practical difference.

3. The interpretation of an image is **stable under geometric transformations**. An image of a tree is still an image of a tree if we shift it leftwards by a few pixels. However, this shift changes every input to the network, and so the model would have to learn the patterns of pixels that correspond to a tree separately at every position. This is clearly inefficient.

Translation invariance

- Image b is the same as a but translated by some pixels.
- Consider an image classification task
- We want both the images to be classified as “mountain”
- This property is called <u>invariance to translation</u>
- <u>CNNs are NOT translation invariant out of the box</u>

a)

b)

Translation equivariance

- Image d is the same as c but translated by some pixels.
- Consider now the task of semantic segmentation: each pixel in the image is classified according to the object it belongs to
- When the input image is translated we want the output (colored overlay) to translate accordingly
- We require the output to be <u>equivariant with respect to translation</u>.

c)

e)

d)

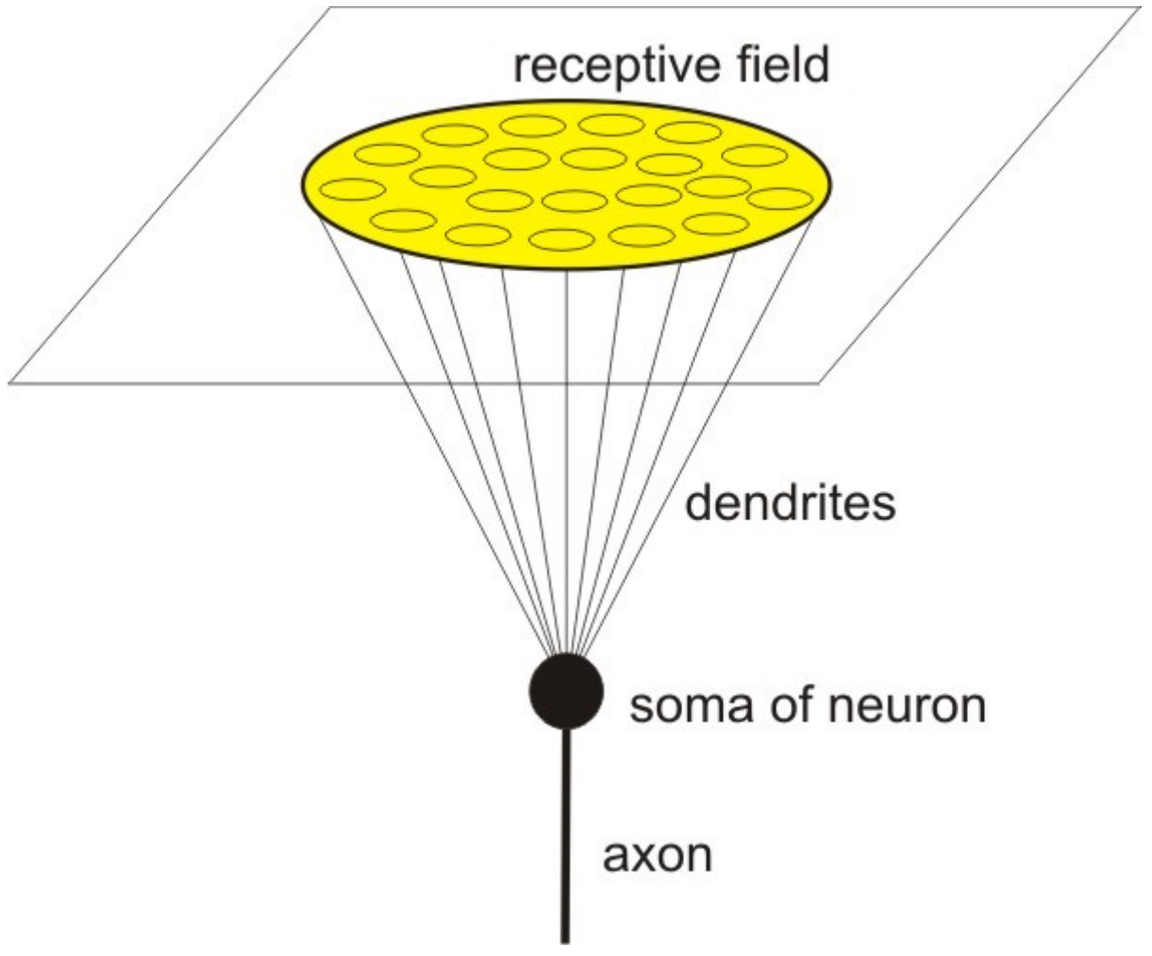
f)

CNNs

- Specialized networks for processing grid-like structures
  - Time-series data -> 1-D grid
  - Images -> 2-D grid
  - Videos -> 3-D grid
- Inspired by neuroscience (visual cortex)
- Based on the mathematical operation of **convolution**
- **Idea**: substitute matrix multiplication with convolution
  - All the rest stays the same

Vision

- **Receptive field** of a single sensory neuron:
- specific region of the retina in which something will affect the firing of that neuron (that is, will activate the neuron).
- Every sensory neuron cell has similar receptive fields
- Their fields are overlapping
- Biological inspiration for CNNs



Convolution operator (on images)

- General definition for **filter (or kernel)** f and **signal** x:

$$(w * x)(t) = \int_{-\infty}^{\infty} x(t-a)w(a)\,da$$

- For discrete signals :

Feature map → $(w * x)(t) = \sum_{a=-\infty}^{\infty} x(t-a)w(a)$

- In practice, since we need to store filter values, we define the filter to be zero everywhere except for a finite set of points

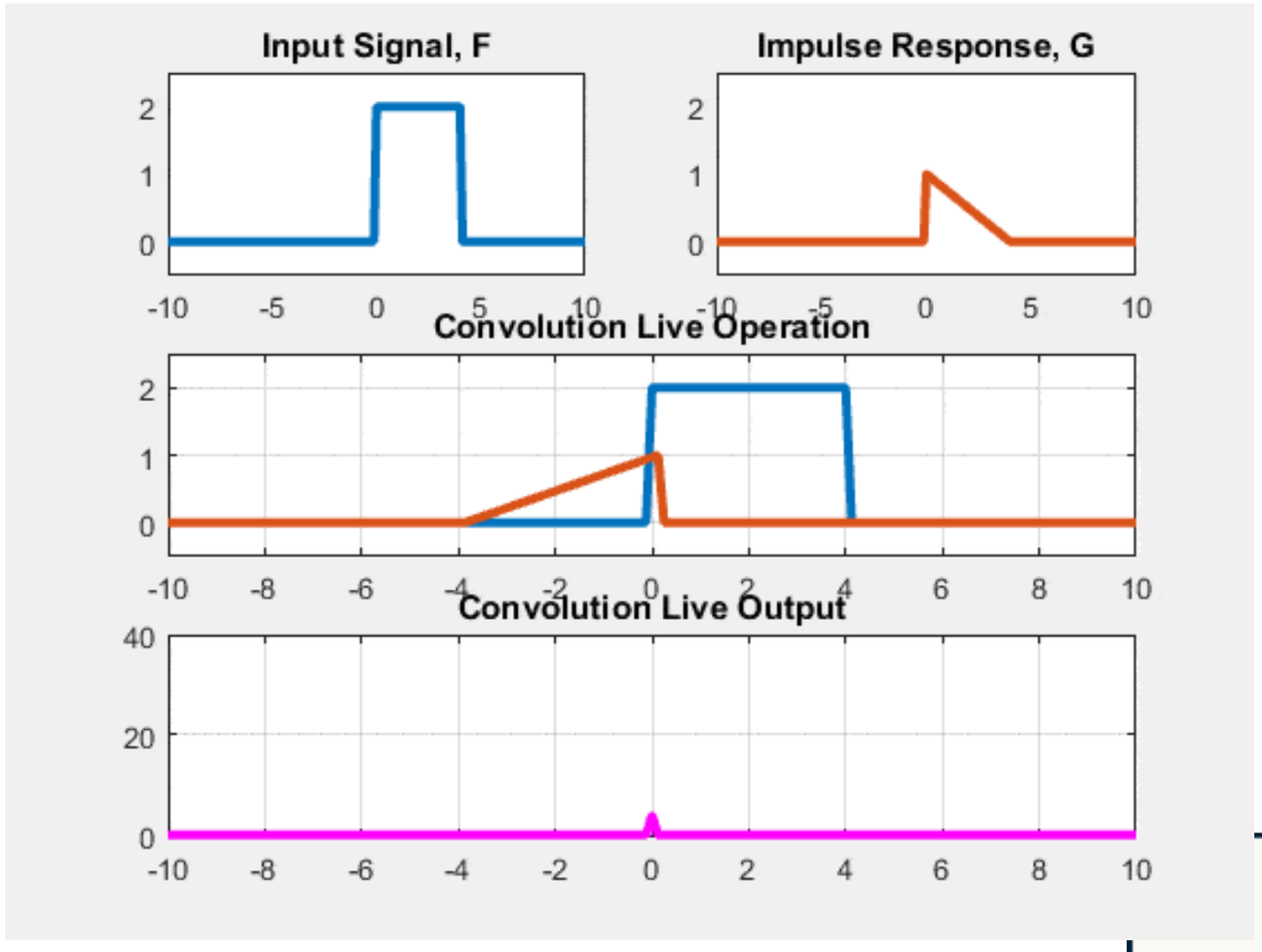
$$(w * x)(t) = \sum_{a=0}^{m} x(t-a)w(a)$$

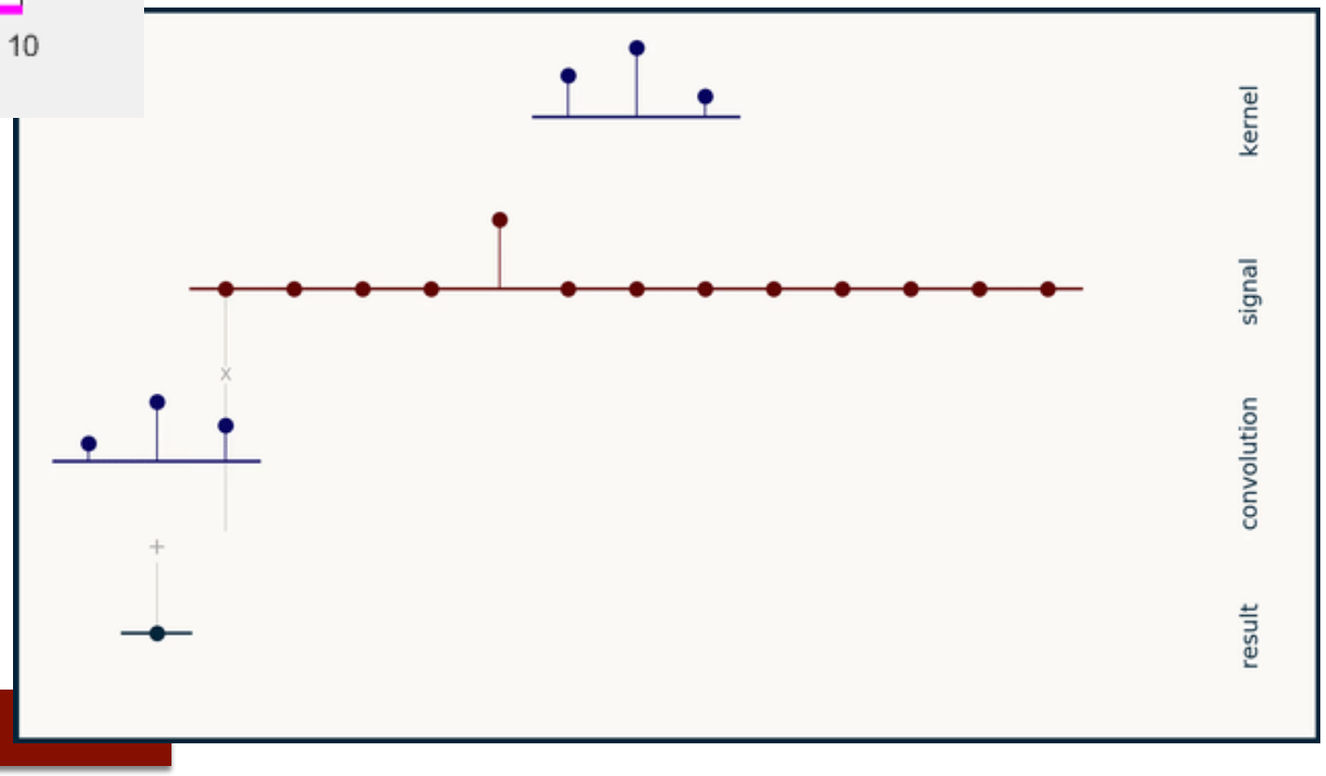
- In images (2-D signals), it corresponds to

$$(\boldsymbol{K} * \boldsymbol{I})\,(i,j) = \sum_{a=-m}^{m} \sum_{b=-n}^{n} K(a,b)I(i-a, j-b)$$

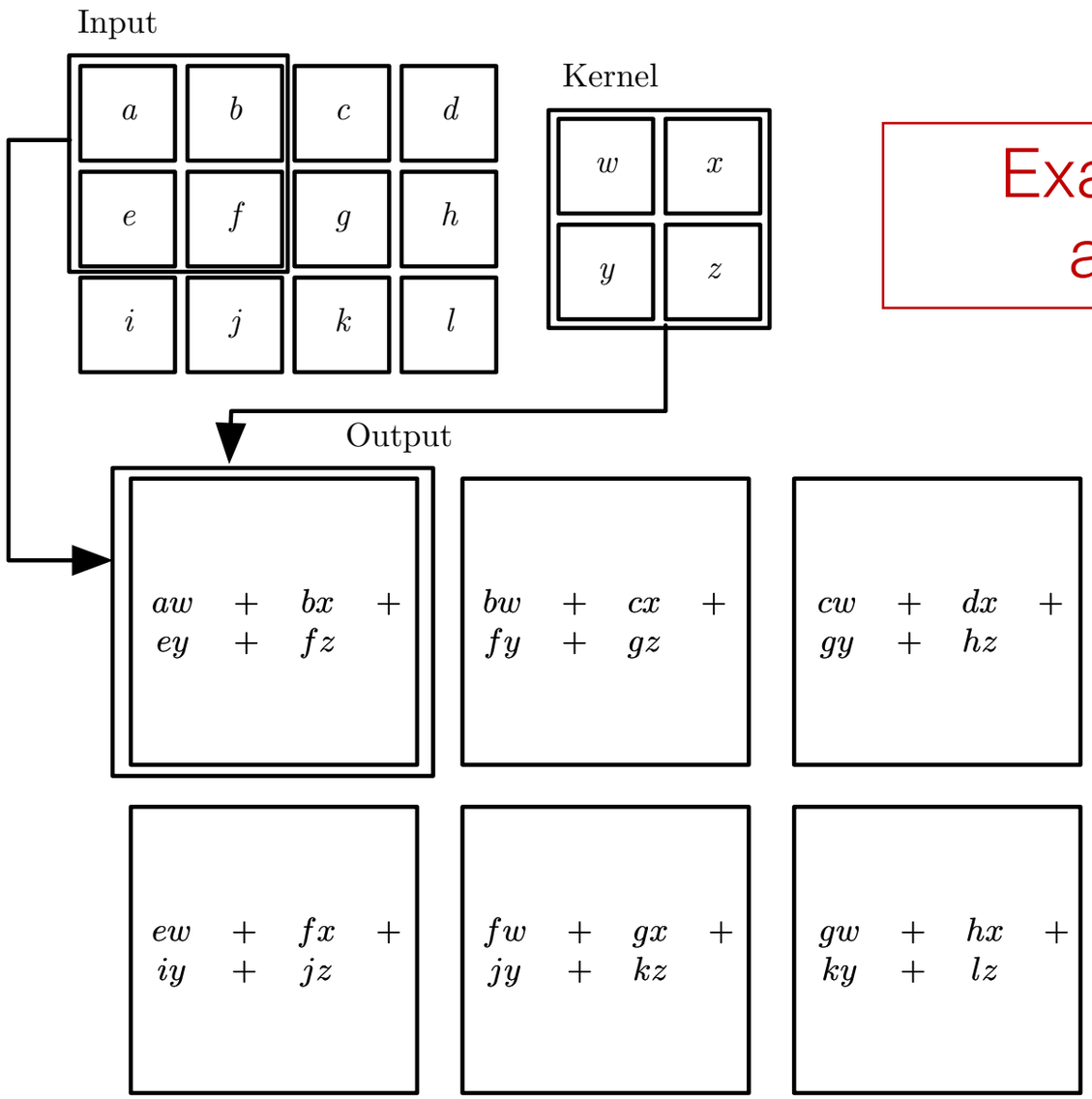
- Where $\boldsymbol{K}$ is a $2m \times 2n$ filter and $\boldsymbol{I}$ an image
- Cross-correlation: $(\boldsymbol{K} * \boldsymbol{I})\,(i,j) = \sum_{a=-m}^{m} \sum_{b=-n}^{n} K(a,b)I(i+a, j+b)$

Convolution operator

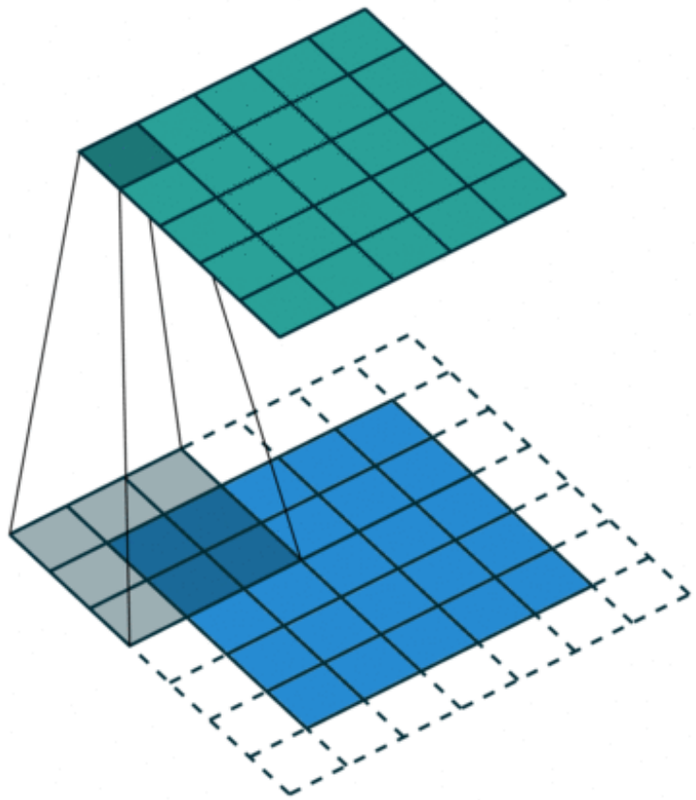




Convolution operator



Example of convolution (this is actually cross-correlation)



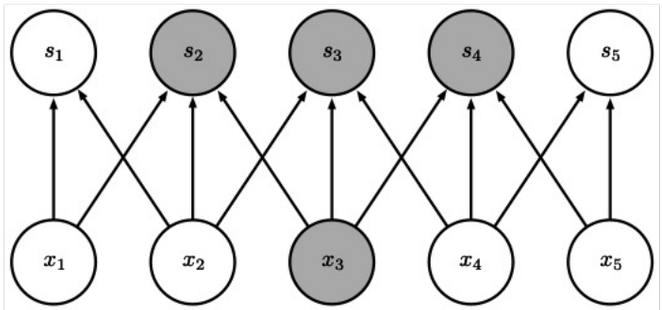
Convolution operator

Main properties (more details in the following slides):

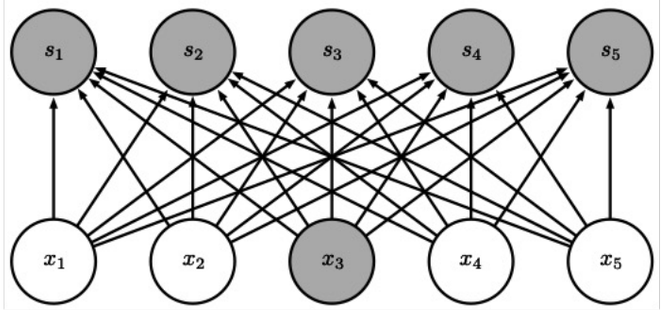
- Sparse interactions -> kernel smaller than input, efficiency
- Parameter sharing
- Equivariant representations
- Works with inputs of variable size

Sparse interactions

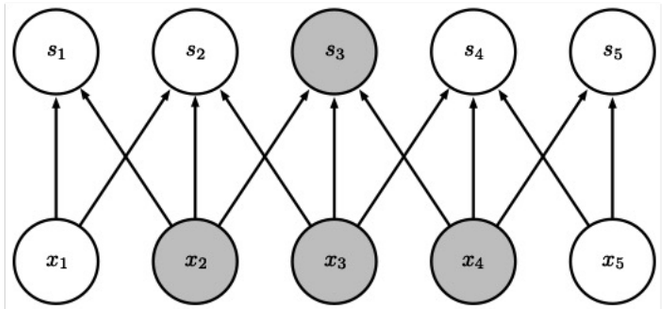
Sparse connections due to small convolution kernel



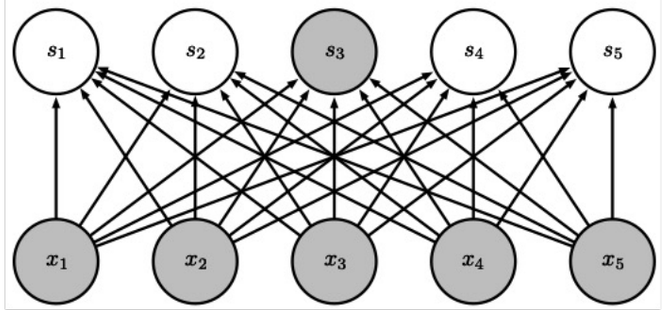
Dense connections



Sparse connections due to small convolution kernel



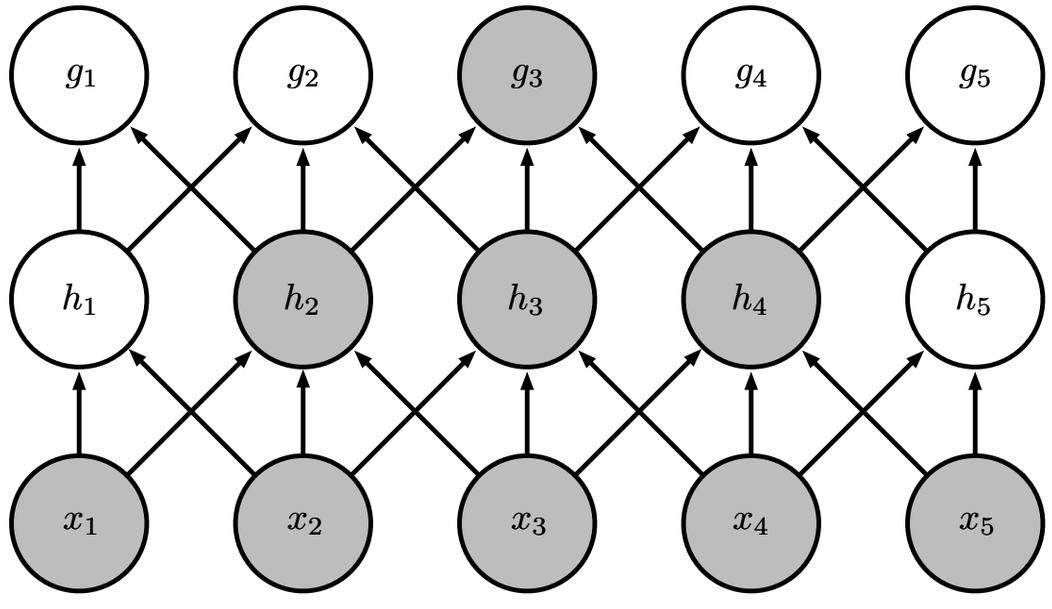
Dense connections



From below

From above

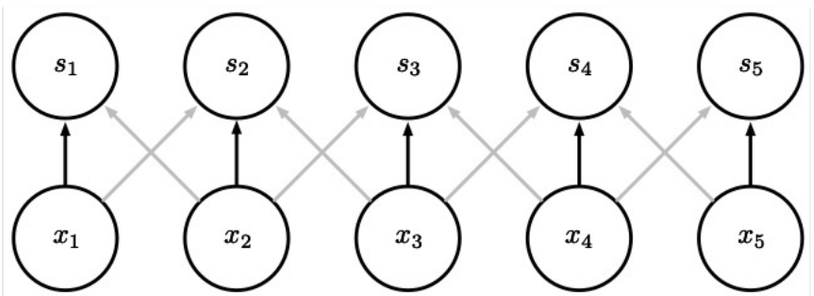
Receptive field grows with depth



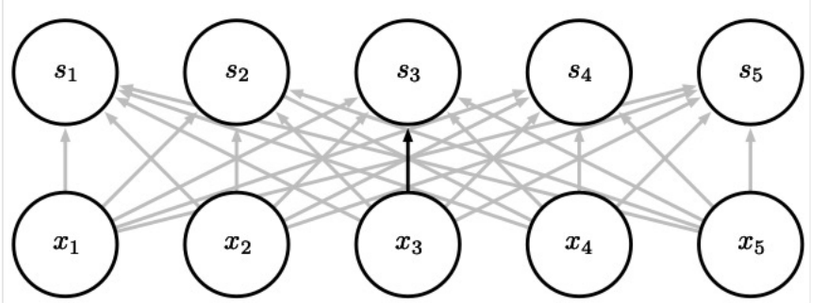
Parameter sharing (tied weights)

- Each member of the filter (kernel) is used at every position in the input
- Instead of learning a separate set of parameters for each location, we learn only one set
- Storage requirements: just the filter
  - Much fewer weights compared to fully connected networks
  - easier to learn

Convolution shares the same parameters across all spatial locations



Traditional matrix multiplication does not share any parameters



Efficiency of convolution

Input size: 320 by 280
Kernel size: 2 by 1
Output size: 319 by 280

	Convolution	Dense matrix	Sparse matrix
Stored floats	2	319*280*320*280 > 8e9	2*319*280 = 178,640
Float muls or adds	319*280*3 = 267,960	> 16e9	Same as convolution (267,960)

2 multiplications and 1 addition for each position

Equivariance to translation

- A function f is equivariant to a function g if

$$f\big(g(x)\big) = g(f(x))$$

Convolution layer

Translation operation (shift)

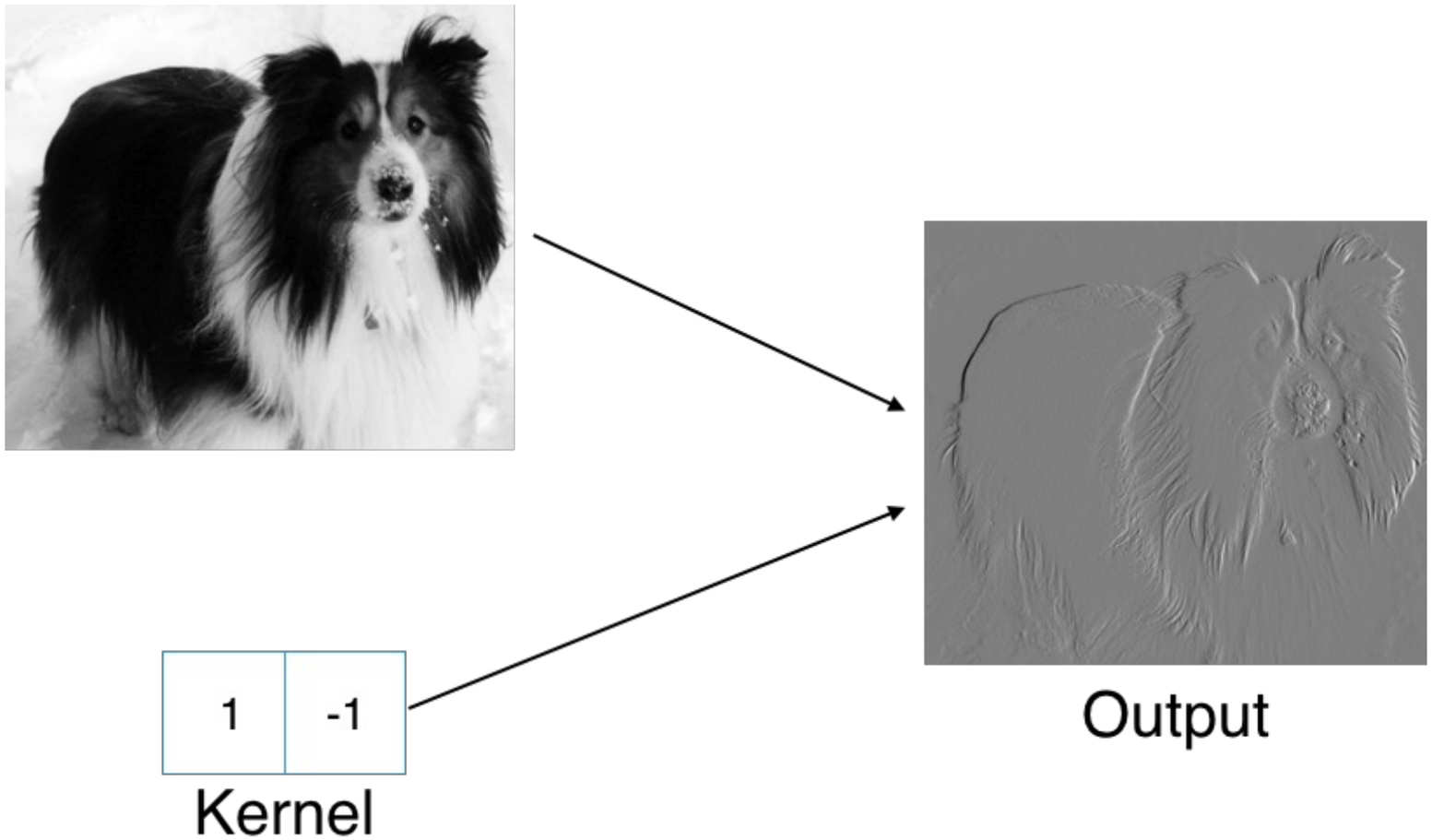
- If we think of g as a function translating an image to the right by one pixel, the feature map will be also translated to the right

- Convolution is **not** equivariant to:
  - Scale change
  - rotation

Inputs of variable size

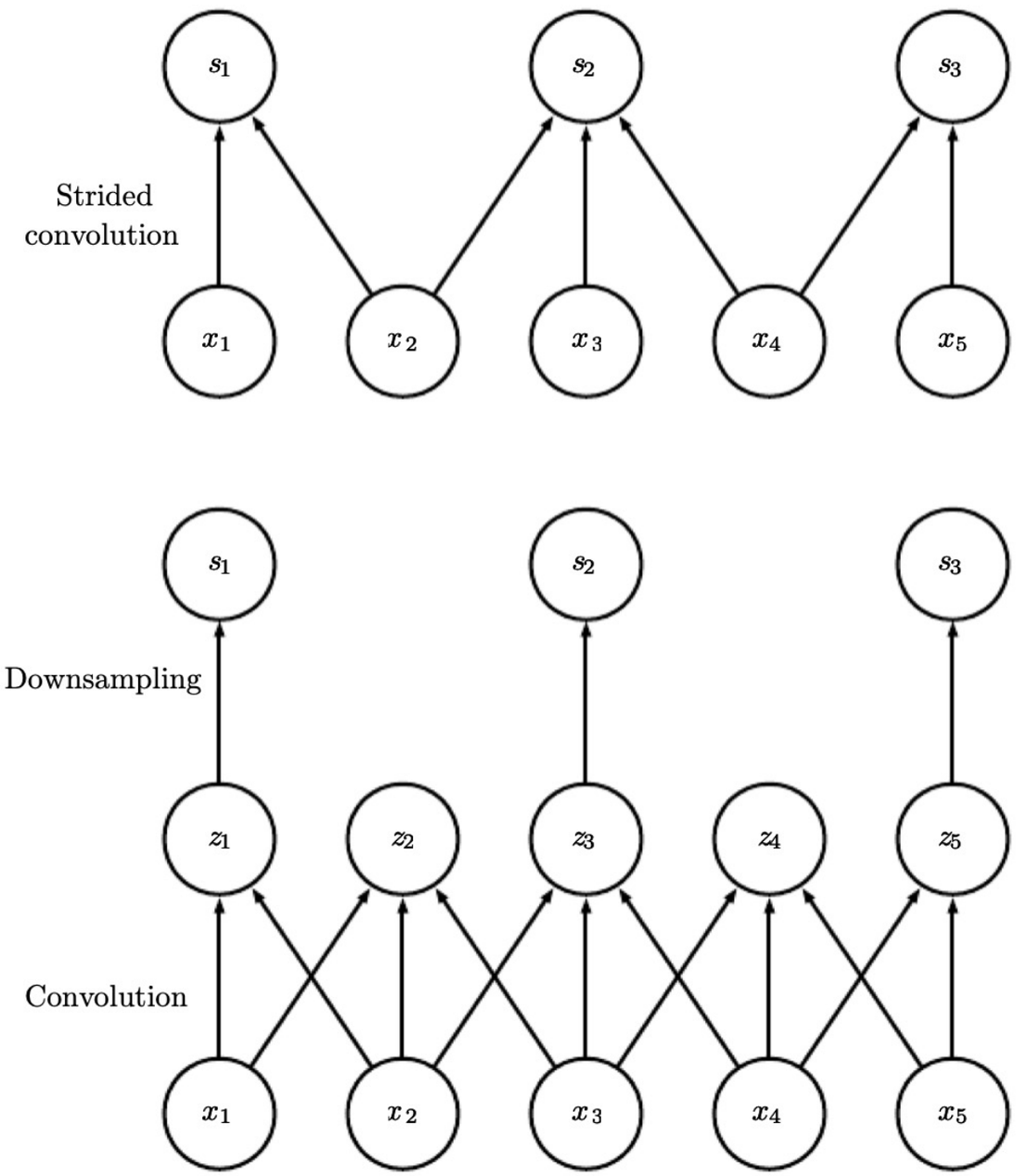
- E.g. images of different size
- We can apply the same convolution operator to all images,
  - The size of the feature map will change accordingly
  - Works well for producing masks
- If the output is fixed size, we need global pooling operations

E.g. Edge detection with convolution



Strided Convolution

- We may want to reduce the size of the feature map compared to the input
  - Downsampling the feature map
  - Improves efficiency
  - Increases receptive field of the following conv layer



Dilated Convolution

- Increasing the kernel size has the disadvantage of requiring more weights.
- Dilated convolutions: the kernel values are spaced with zeros.
  - E.g. in 1D: we can turn a kernel of size five into a dilated kernel of size three by setting the second and fourth elements to zero
  - We still integrate information from a larger input region but only require three weights to do this.
- The number of zeros we intersperse between the weights is termed the dilation rate.

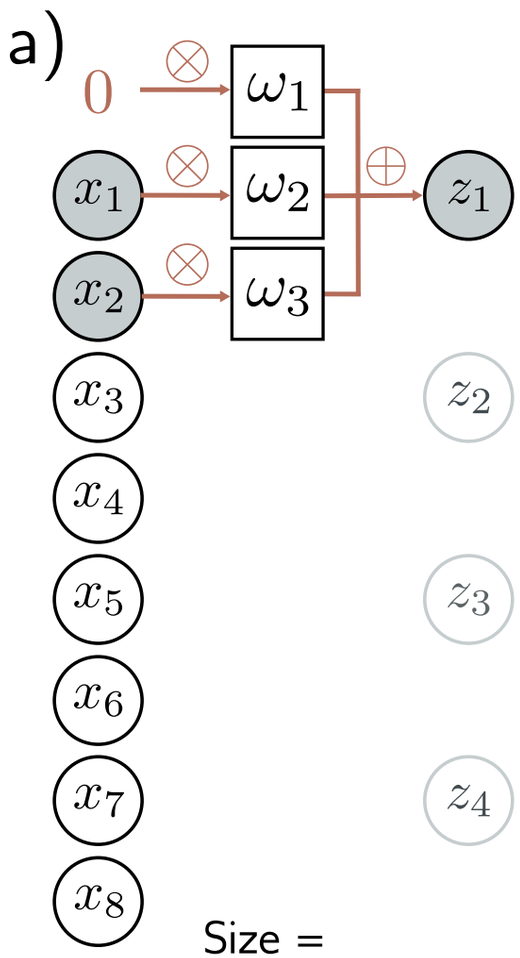
5x5 kernel (25 weights)

3x3 kernel, dilation 1 (9 weights)

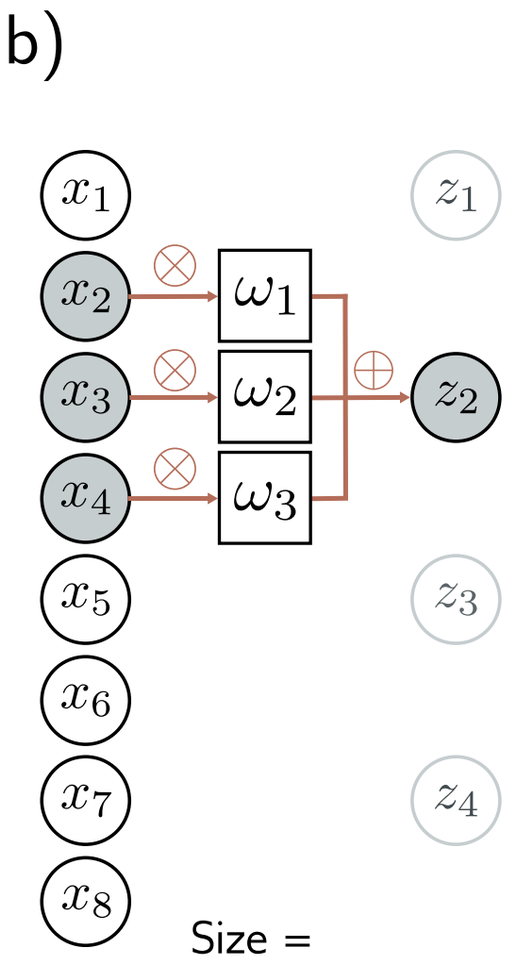
	0		0	
0	0	0	0	0
	0		0	
0	0	0	0	0
	0		0	

Exercise

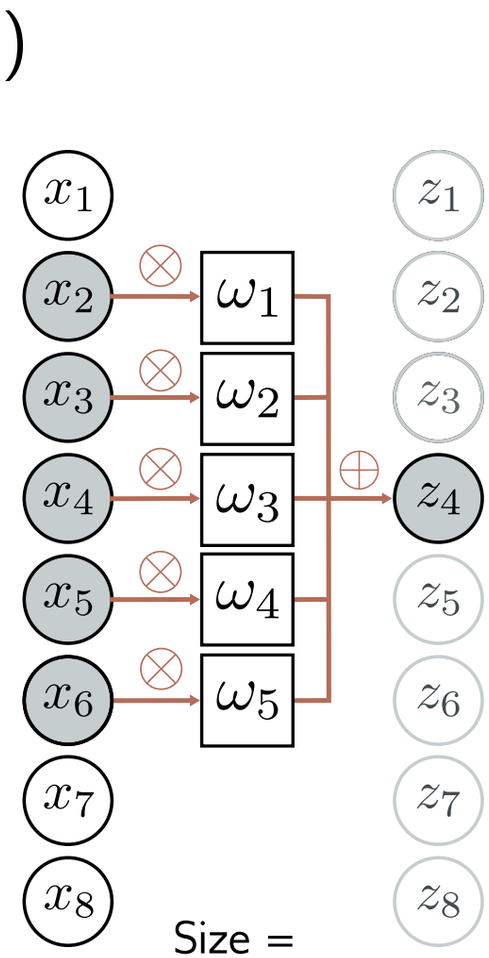
For each image: Size, Stride, Dilation



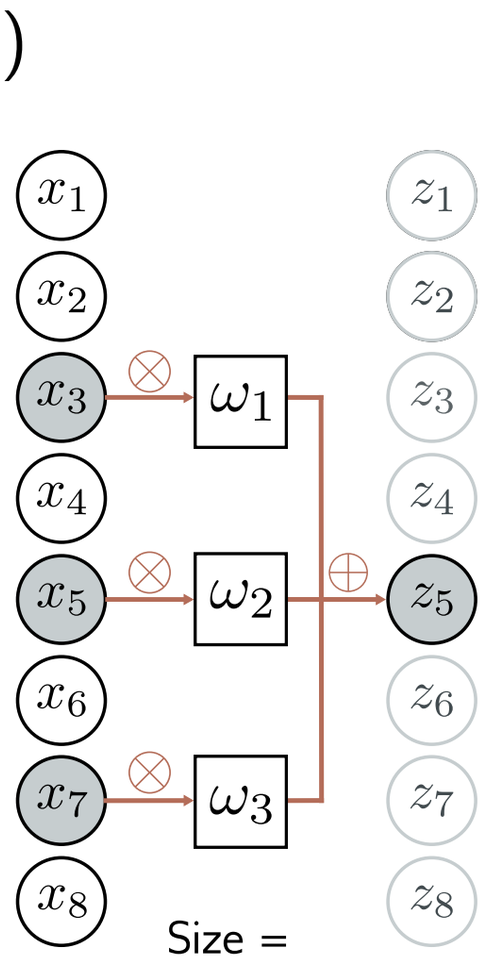
Size =
Stride =
Dilation =



Size =
Stride =
Dilation =

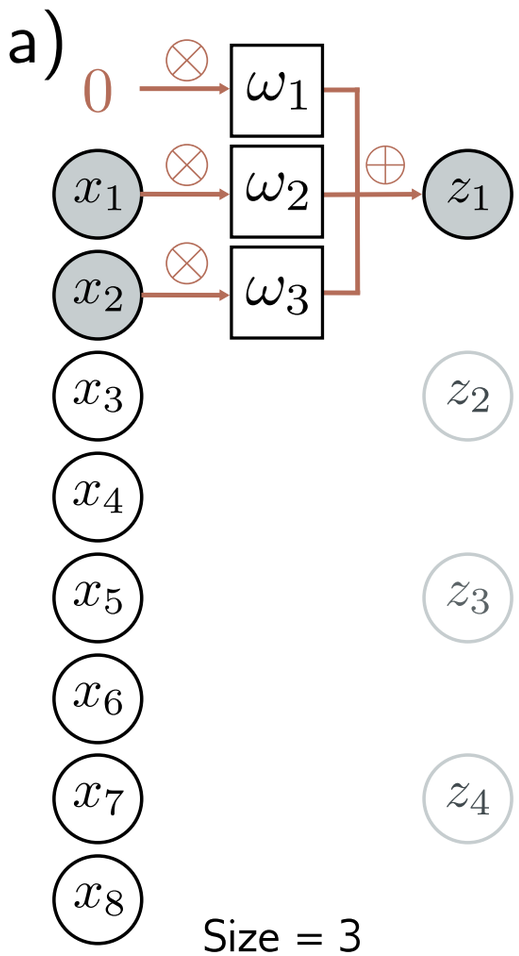


Size =
Stride =
Dilation =

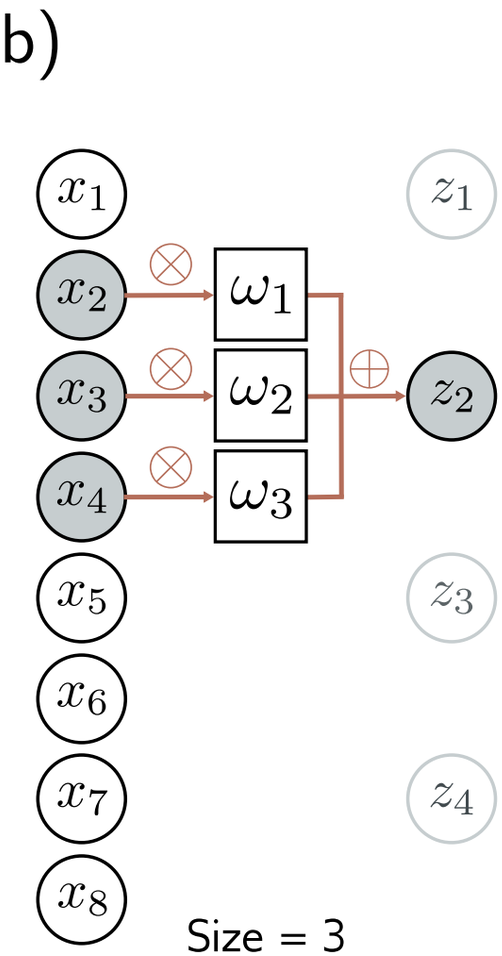


Size =
Stride =
Dilation =

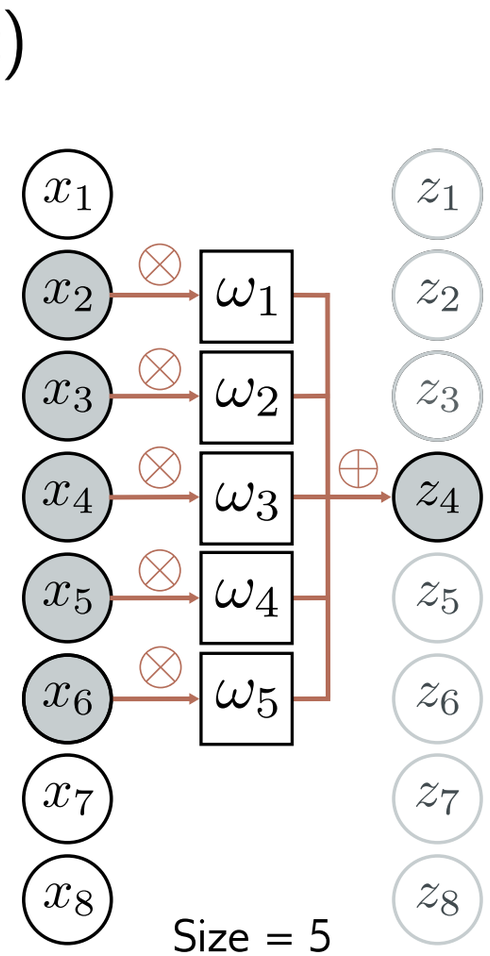
Exercise: solution



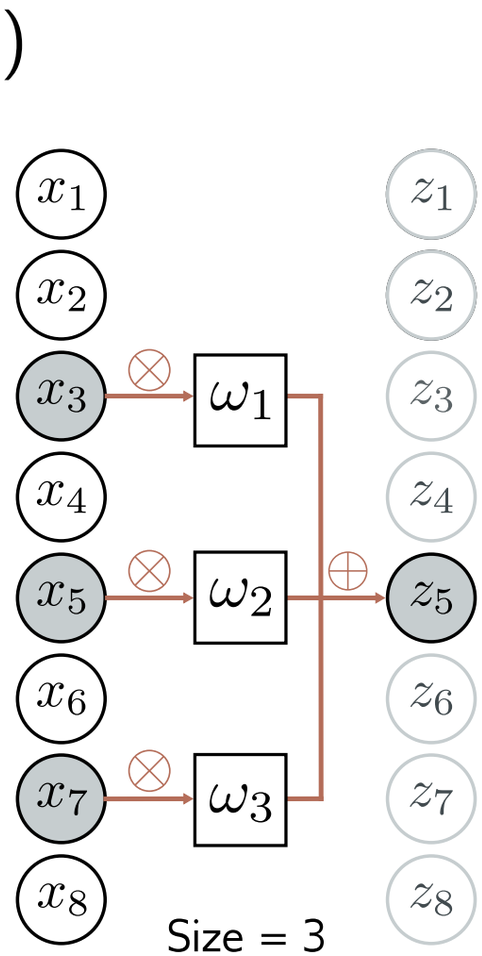
Size = 3
Stride = 2
Dilation = 0



Size = 3
Stride = 2
Dilation = 0



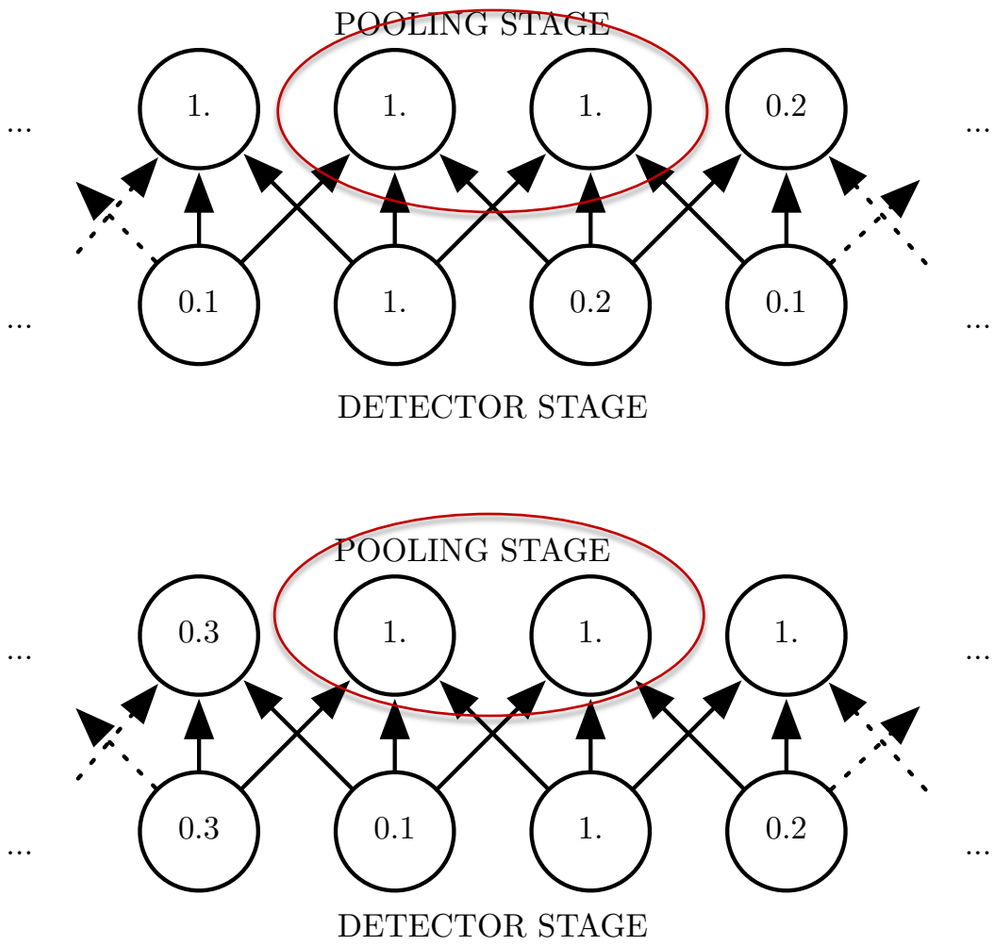
Size = 5
Stride = 1
Dilation = 0



Size = 3
Stride = 1
Dilation = 1

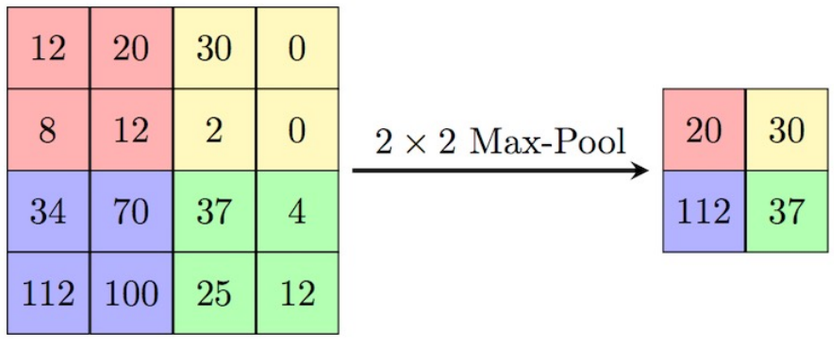
Spatial Pooling

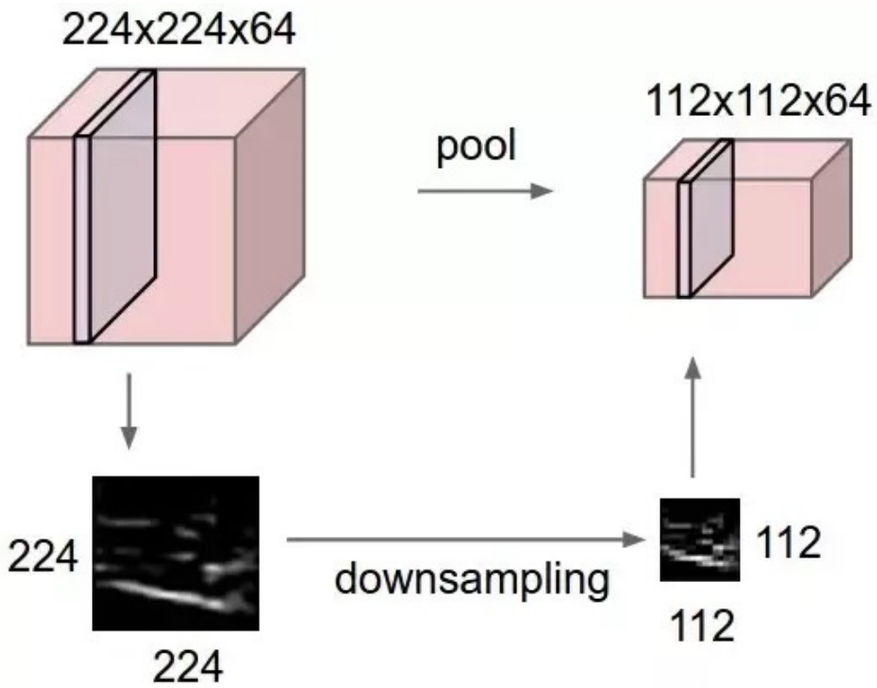
- Makes the network approximately **invariant** to small translations



Input shifted to the right

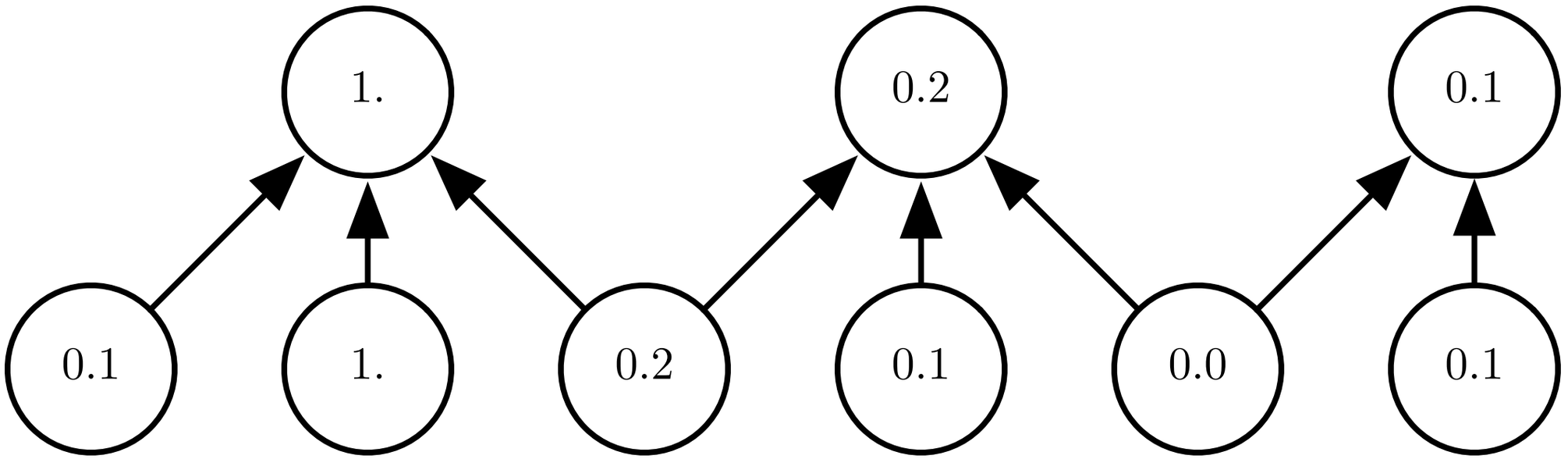
Usually combined with stride





Pooling to improve efficiency

- Downsampling: reduces the size of the input for the subsequent layers



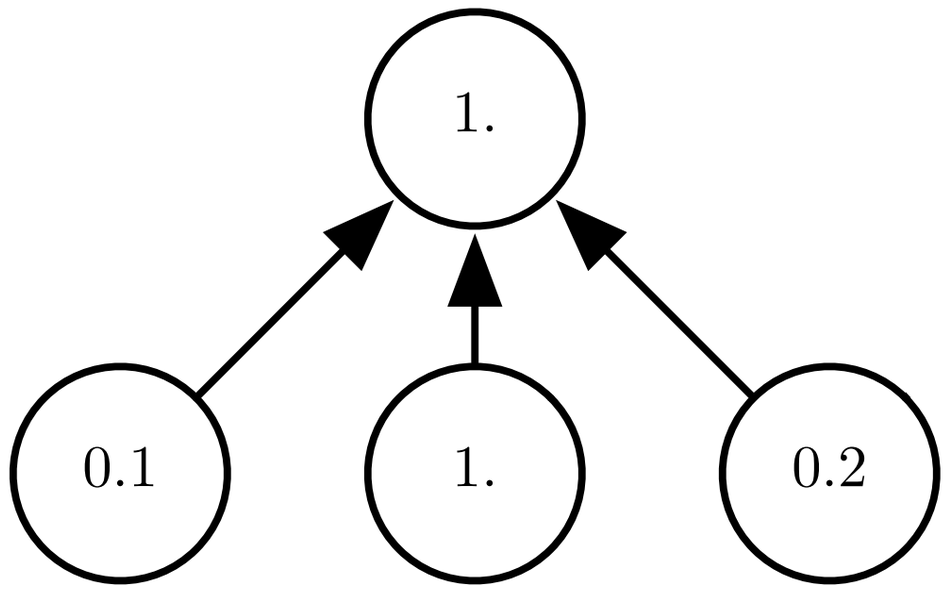
Pool width: 3, stride: 2

Pooling to generate a fixed-size repr.

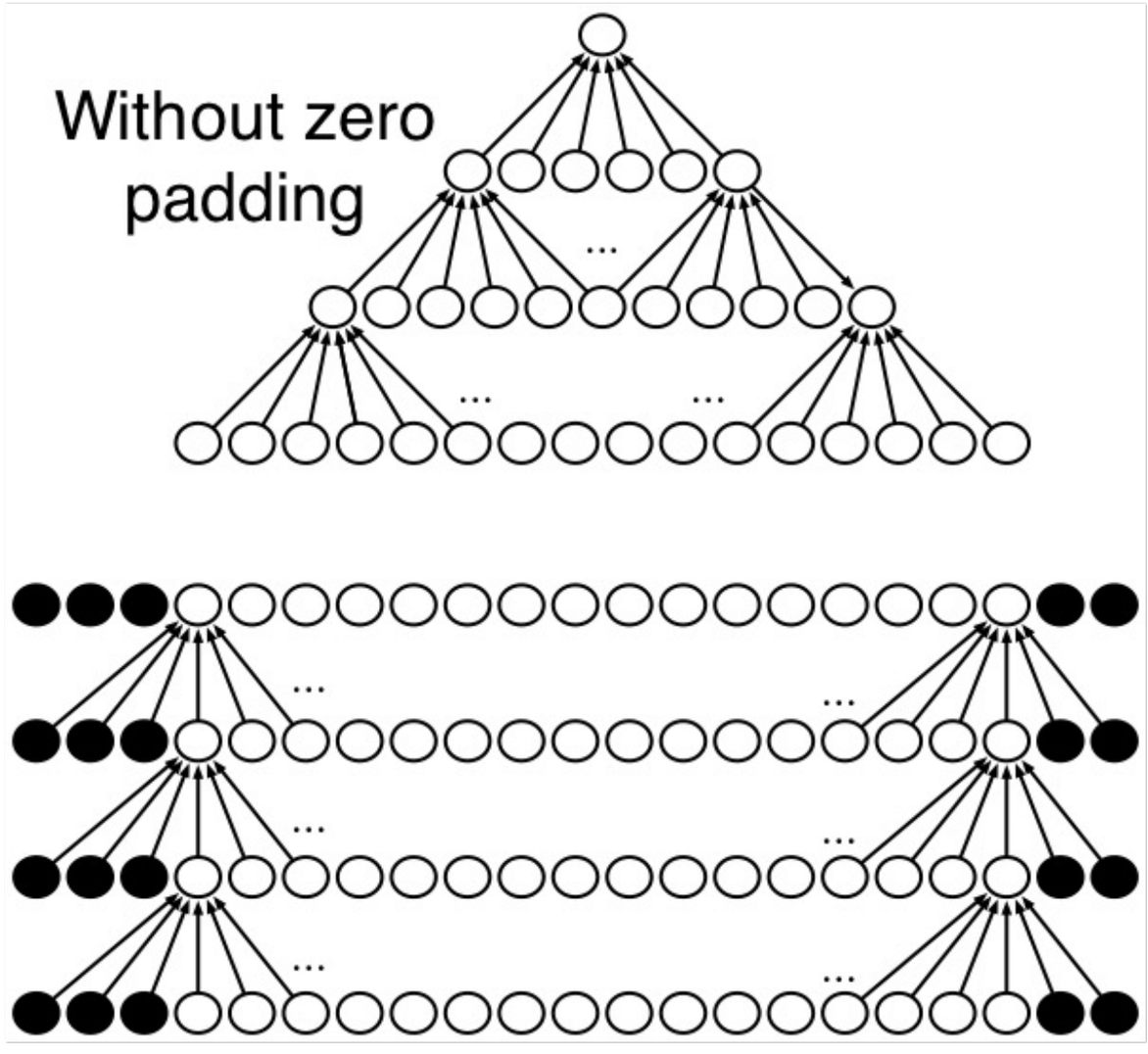
- Conv layers are usually followed by a stack of fully connected layers
  - needs a fixed-size vector as input
- CNNs can be applied to inputs of different sizes
  - The size of the feature map will change
- We can define a pooling layer that <u>splits the input in a fixed number of regions</u>
  - The downsampling size may depend on the size of the input

Global Pooling

- Aggregates a variable number of inputs in a single output
  - E.g. Global mean/max pooling



Padding



Padding

Padding = Valid

Filter

1	0
0	0.5

Stride X

Input

1	0	0.5	0.5
0	0.5	1	0
0	1	0.5	1
1	0.5	0.5	1

Stride Y

Output

1.25	0.5	0.5
0.5	0.75	1.5
0.25	1.25	1

outDim = (inpDim - filterDim +1)/strideDim

Padding = Same

Filter

1	0
0	0.5

Stride X

Input

0	0	0	0	0
0	1	0	0.5	0.5
0	0	0.5	1	0
0	0	1	0.5	1
0	1	0.5	0.5	1

Stride Y

Output

0.5	0	0.25	0.25
0	1.25	0.5	0.5
0	0.5	0.75	1.5
0.5	0.25	1.25	1

outDim = (inpDim)/strideDim

Multi-channel input

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

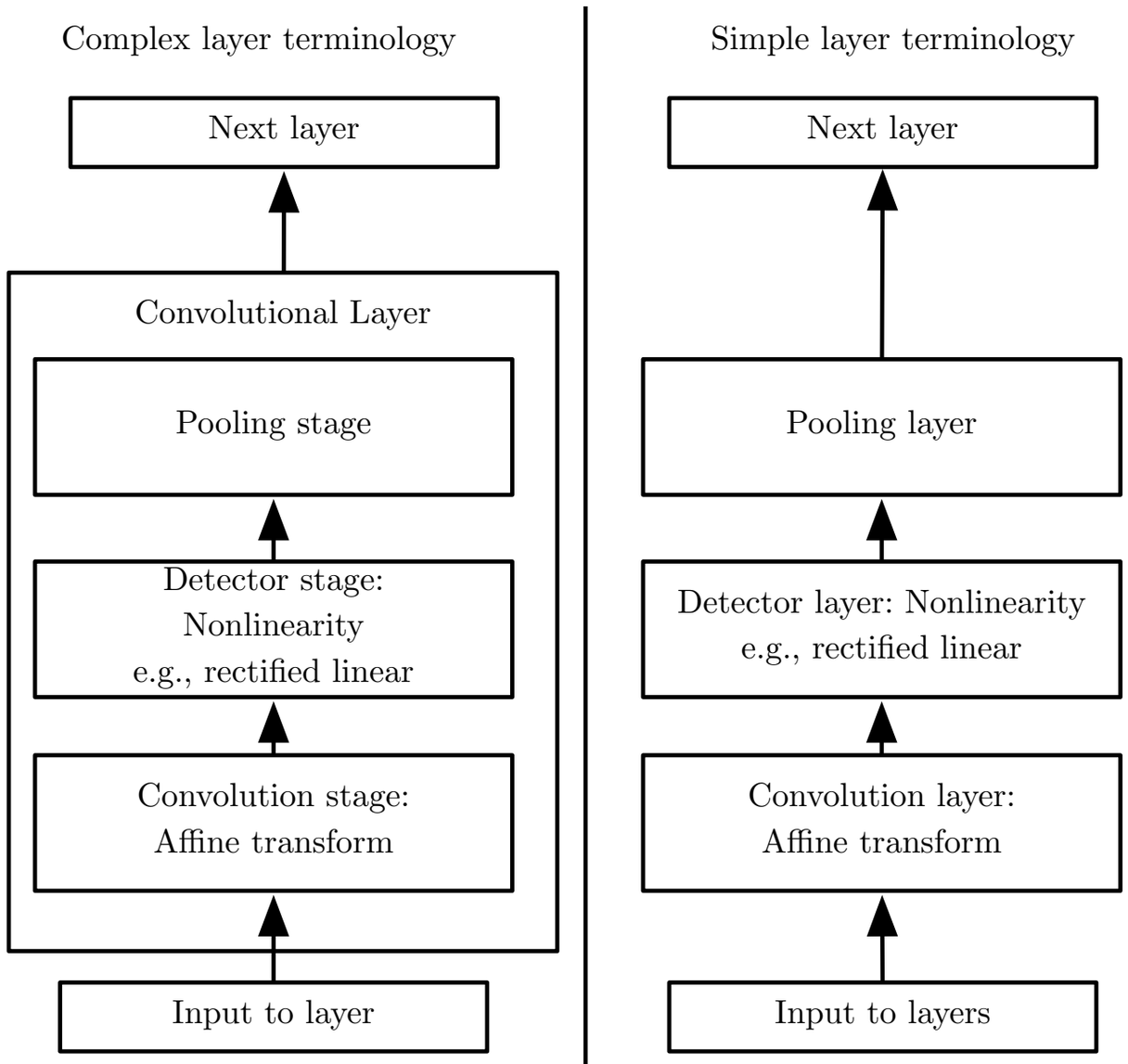
$308 + -498 + 164 + 1 = -25$

Bias = 1

Output

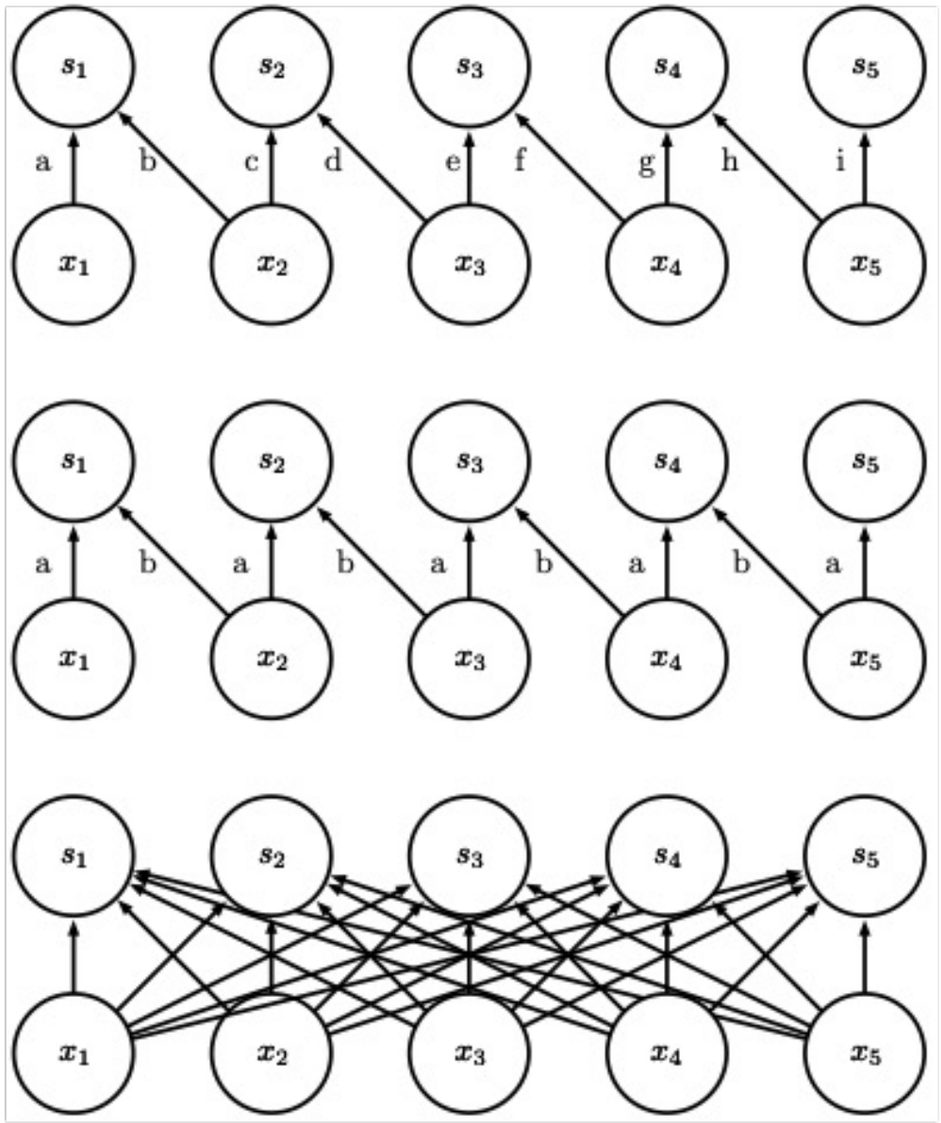
-25				...
				...
				...
				...
...	...	...	...	...

Convolutional Network Layer



Unshared convolution

- Locally connected layers
- Similar to convolution, <u>without</u> shared weights



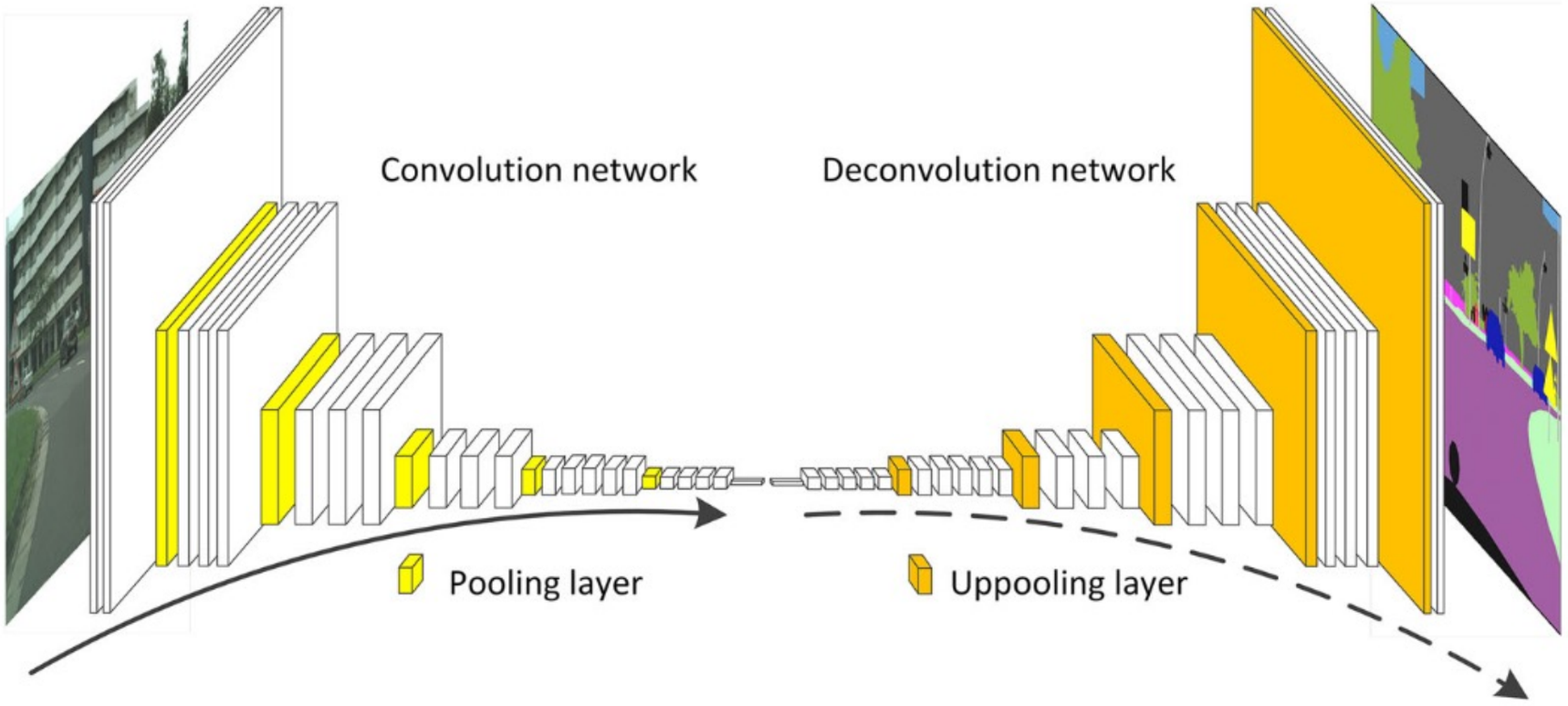
Local connection: like convolution, but no sharing

Convolution

Fully connected

Structured outputs

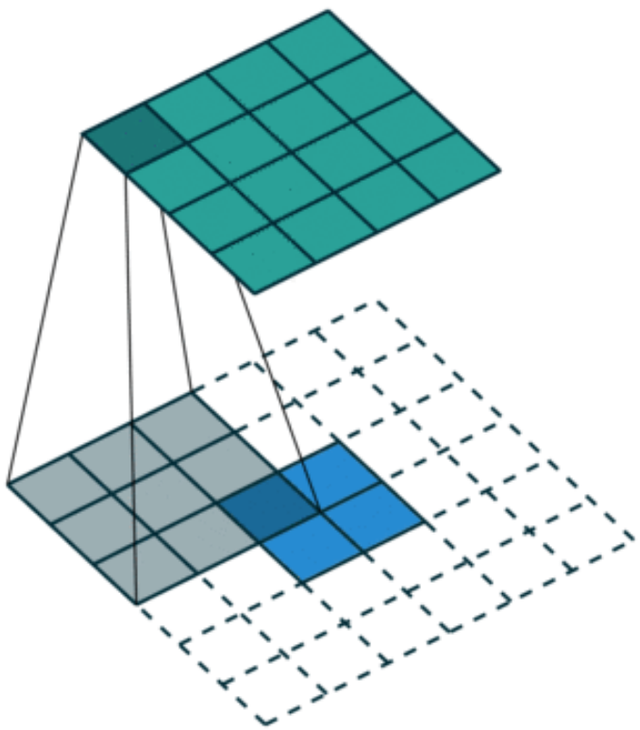
- E.g. output a tensor with the same shape of the input, with a class label for each pixel to produce object detection masks
  - No pooling/ padding same or
  - Convolution / Deconvolution



Deconvolution

- Or more properly Transposed convolution
- Conv. can be seen as matrix multiplication, being careful of the shared weights
- The mult. by the transpose of such matrix is needed for backprop (multiplication by the transpose weights) as well as to reconstruct the input from a feature map

Stride 1, padding: valid



Stride 2, padding: same