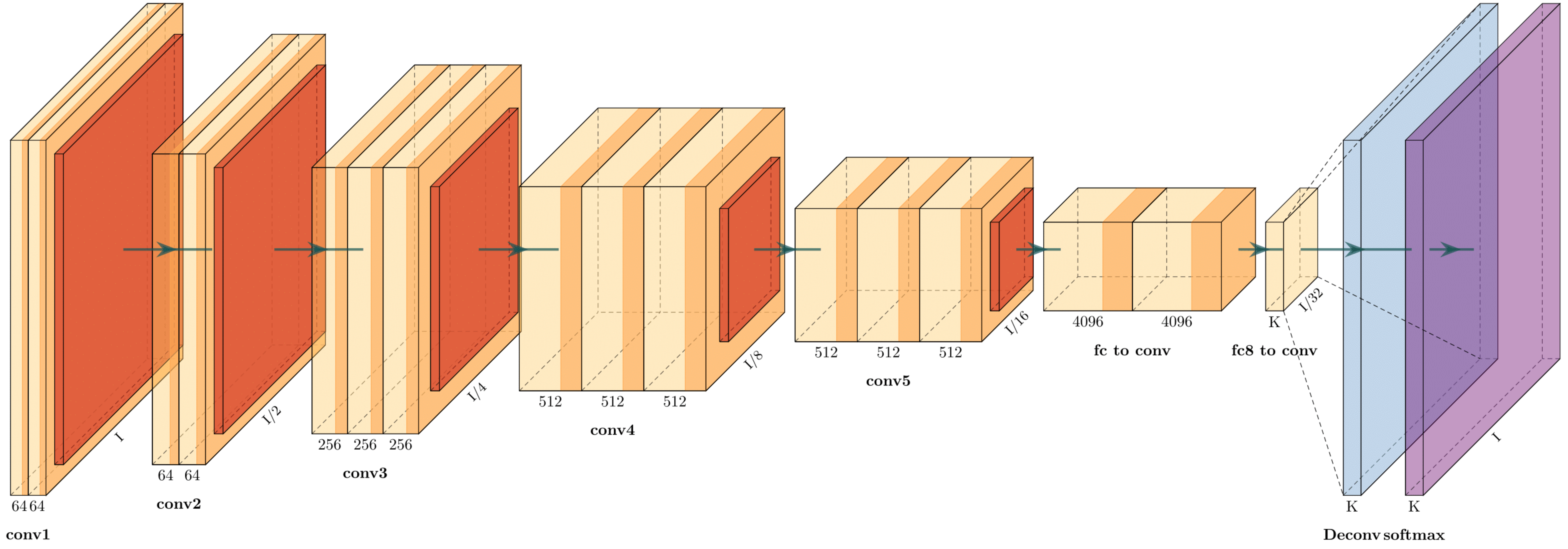
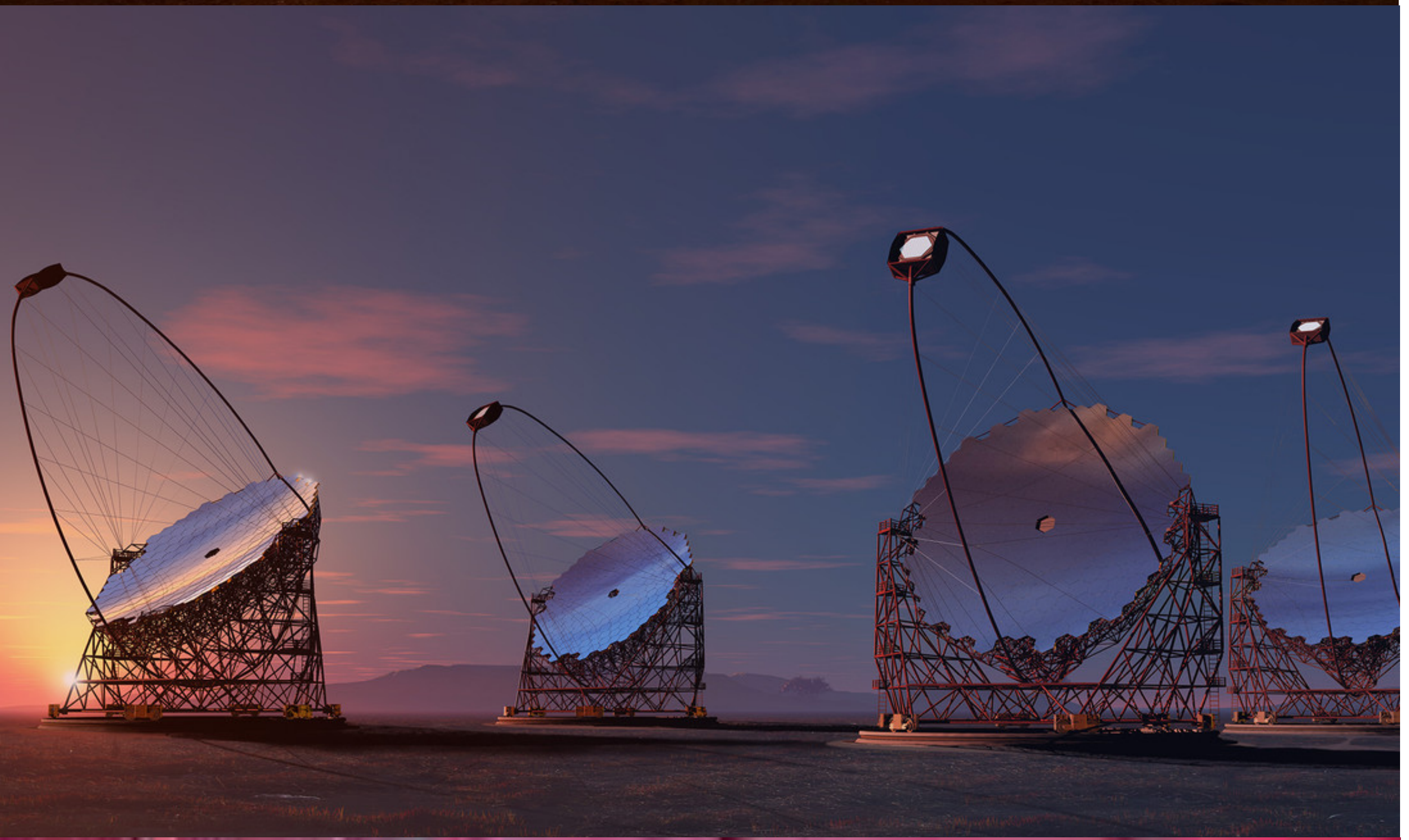
Convolutional neural networks (CNNs)

Stefan Funk - Erlangen Centre for astroparticle physics (ECAP)

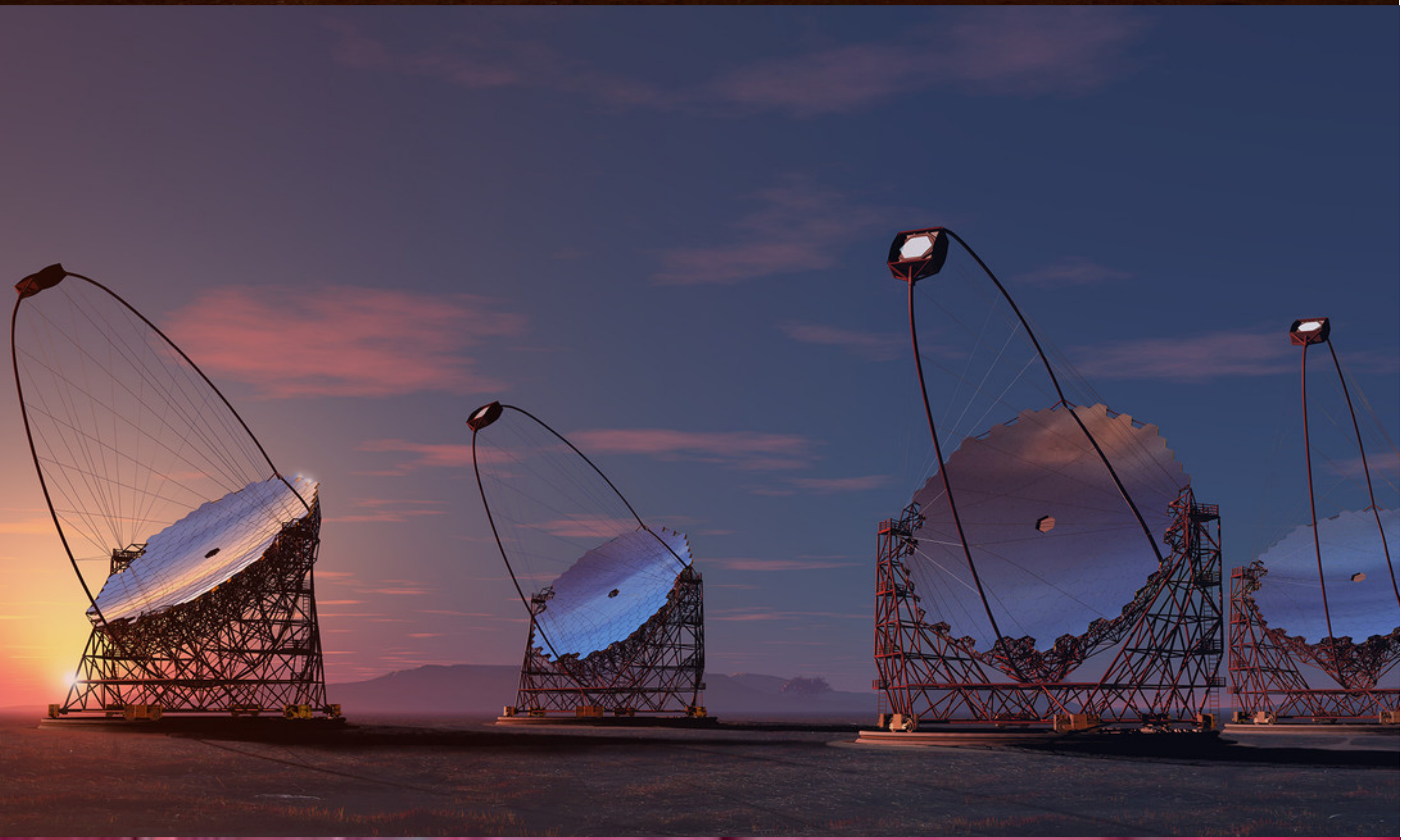
With input from a lot of people, in particular M. Meyer (U. Hamburg) and D. Malyshev (FAU)



me



me



me
and
you

CNNS IN THE PHYSICS DEPARTMENT AT FAU

Friedrich-Alexander-Universität
Faculty of Sciences

- At FAU (Physics department):
  - 1st semester: Datenverarbeitung
  - Later Wahlfächer like: “Machine learning and data analysis in science” and “Machine learning for physicists”
  - https://florianmarquardt.github.io/deep_learning_basics_linkmap.html
  - Currently no single course in physics focussed on Deep Learning
- But of course also courses in other departments (e.g. Data Sciences, …)
- What I will describe is one or two lectures on CNNs in the context of the above lectures
  - Will assume that everything about machine learning, ANN, back propagation, … has been covered

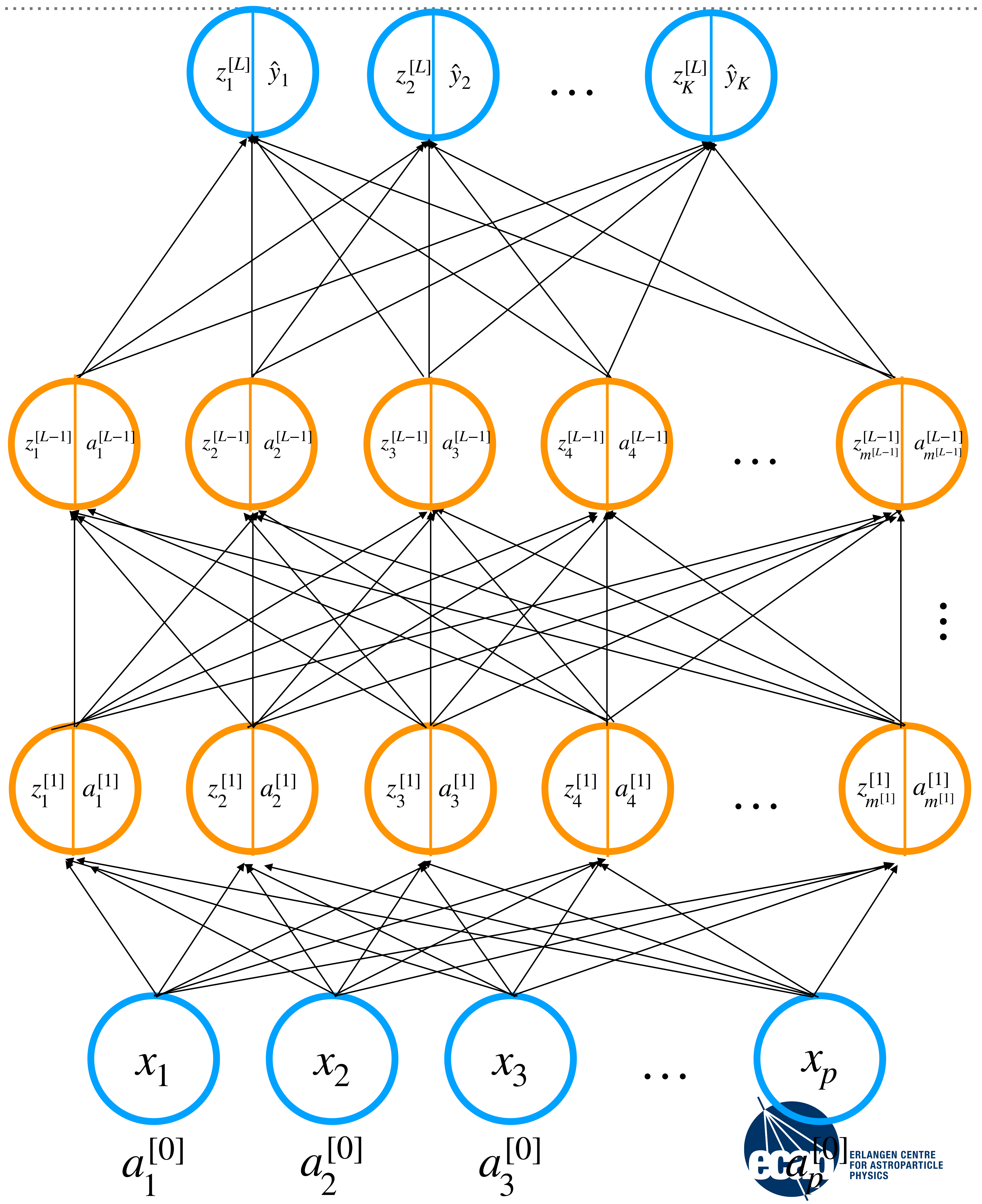
GOAL OF THE LECTURE

- At the end of this lecture, the students should
  - Understand the basic building blocks of a convolutional neural network
  - Understand why CNNs are very powerful in extracting features
  - Understand how modern CNNs are built
  - Be able to build their own CNN for a classification task

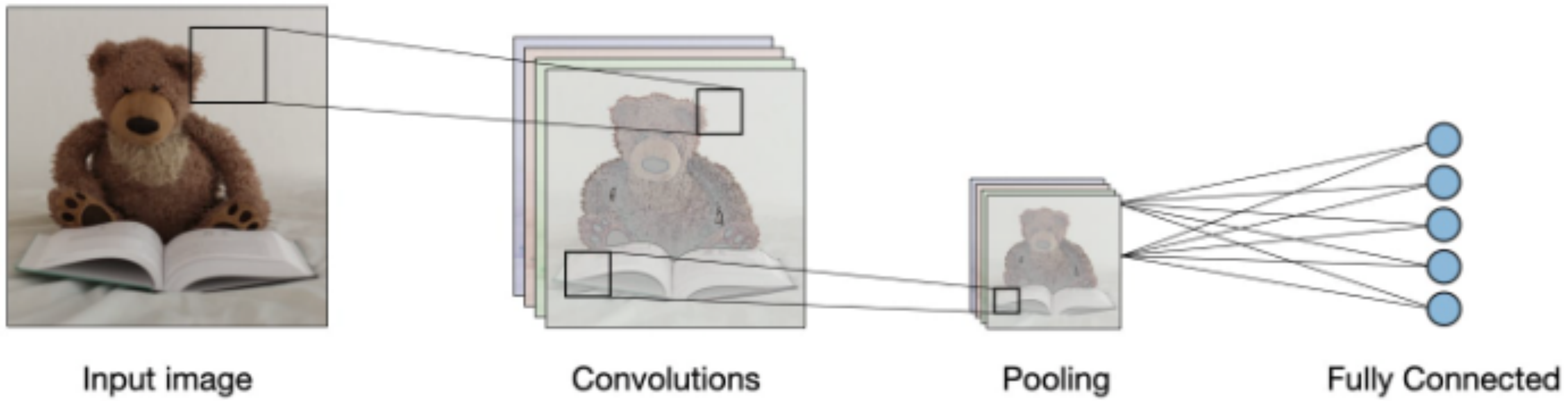
https://www.istockphoto.com/de/vektor/gruppe-von-studenten-gm910098436-250650793

REMINDER LAST LECTURE

- We introduced fully connected neural networks (aka multi-layer perceptrons) for 2-class and *K*-class classification and regression
  - Learned about weights, learned about activation function (e.g. ReLU)
- We learned how to train a network for a given cost function J, using **back propagation**
- Neural networks are prone to overfitting, often necessary to include **regularization**



CNNS IN A NUT-SHELL



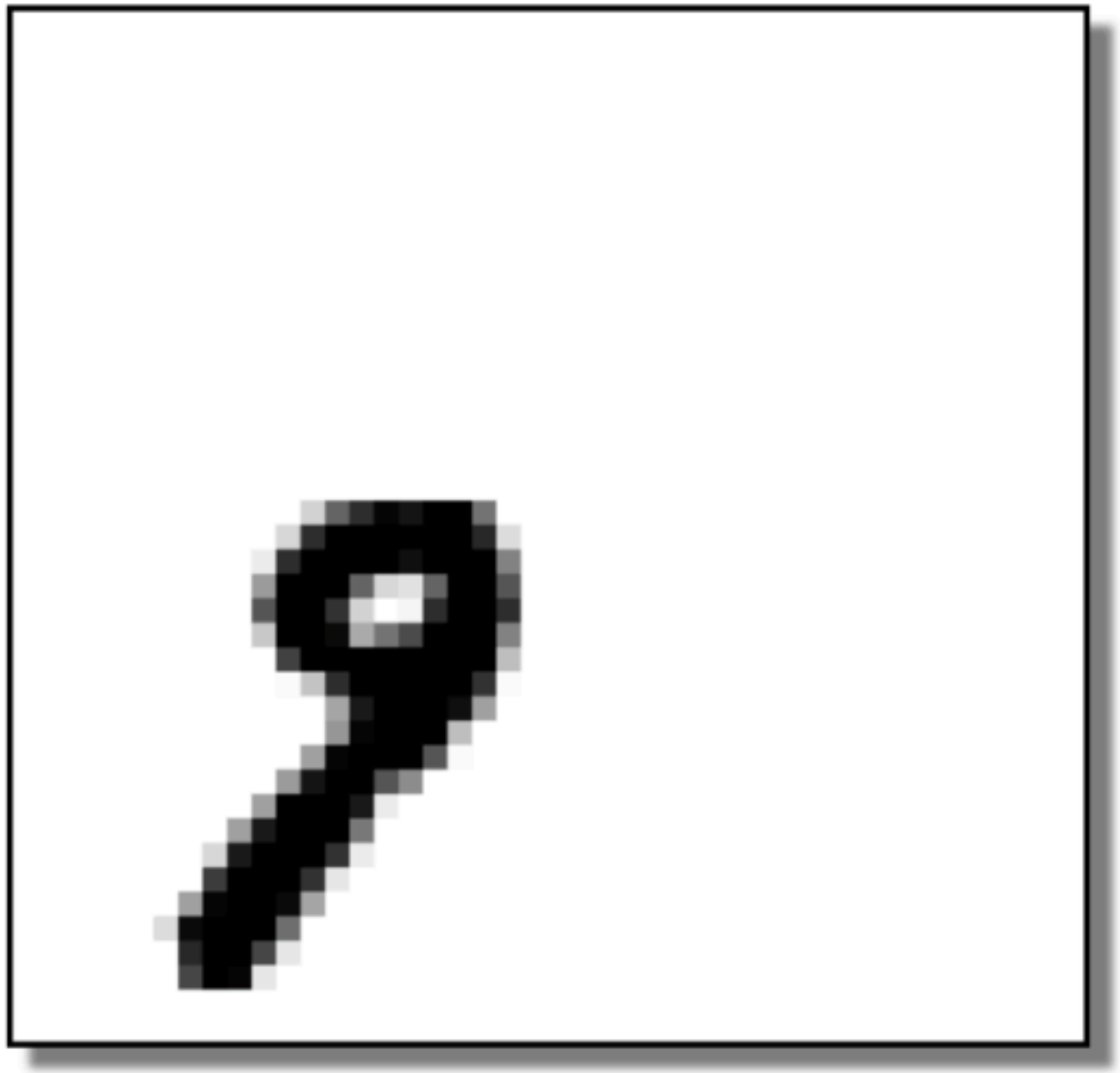
- ➤ Inspired by visual cortex (small region of cells sensitive to specific regions of the visual field)
  - ➤ Some neurons fire when they see vertical edges, others for diagonal ones (e.g. https://arxiv.org/pdf/1406.3284.pdf)
- ➤ CNN learns to look features (=values of the filters) on its own through learning
  - ➤ Technically: slide convolution 'filter' over input volume
  - ➤ Learning part: determine optimal parameter of filters
- ➤ Able to derive patterns in a highly-complex input space

THE CHALLENGE

What We See

What Computers See

THE POWER OF CNNS



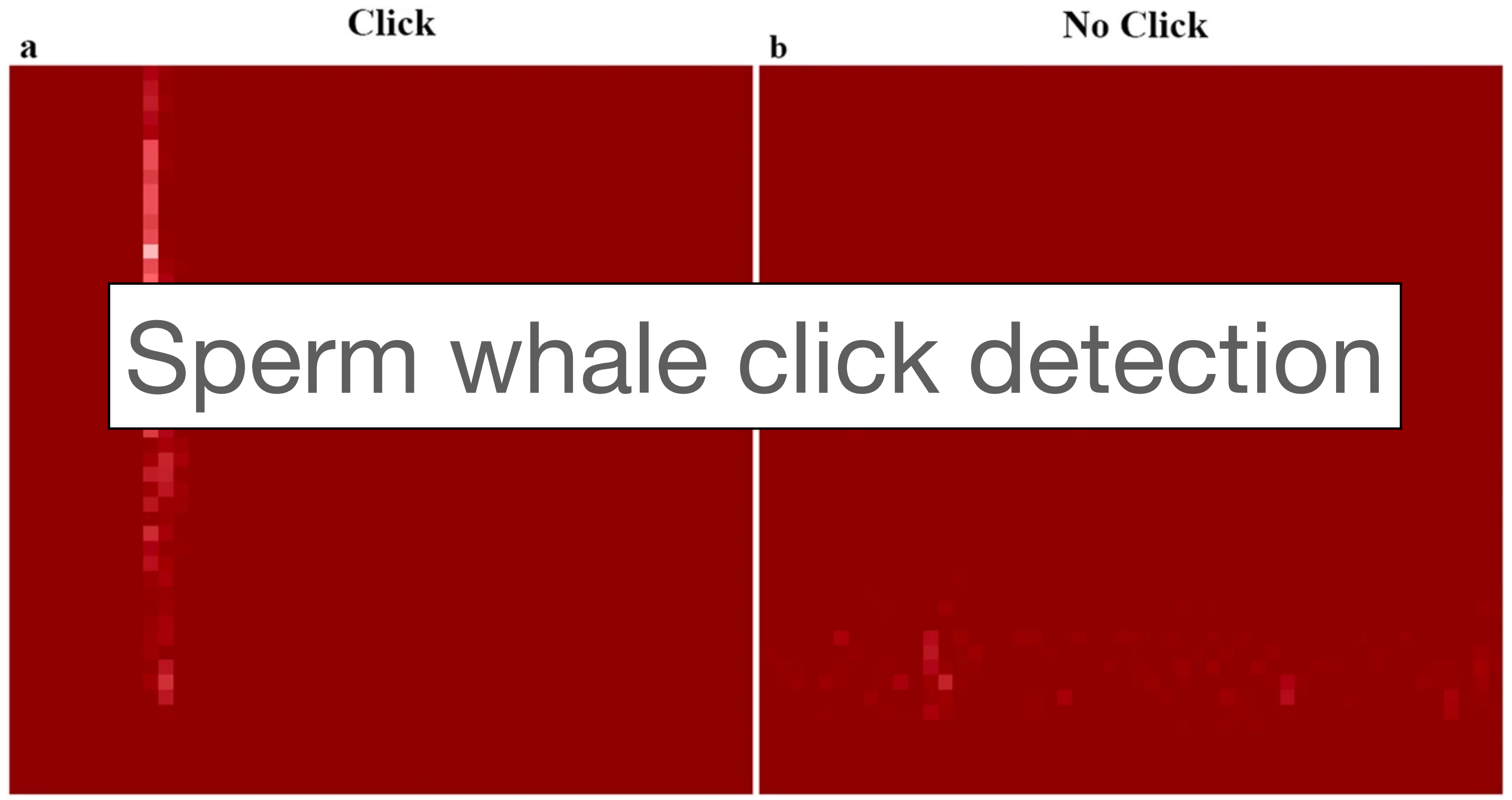
- Different image - same meaning

https://image-net.org/about.php

https://image-net.org/about.php

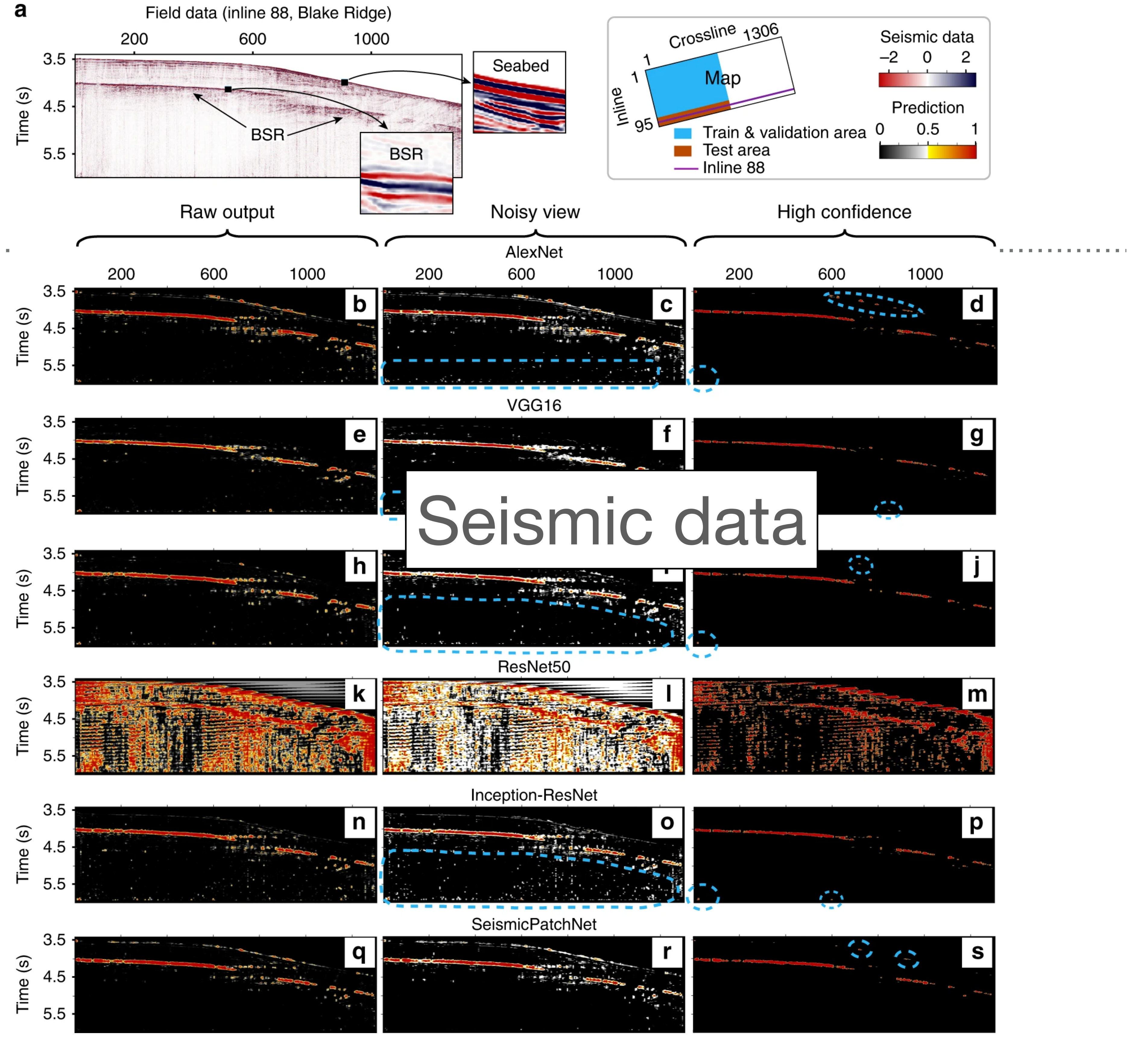

Person
Car
Motorcycle
Helmet
https://image-net.org/about.php

CNNS ARE EVERYWHERE



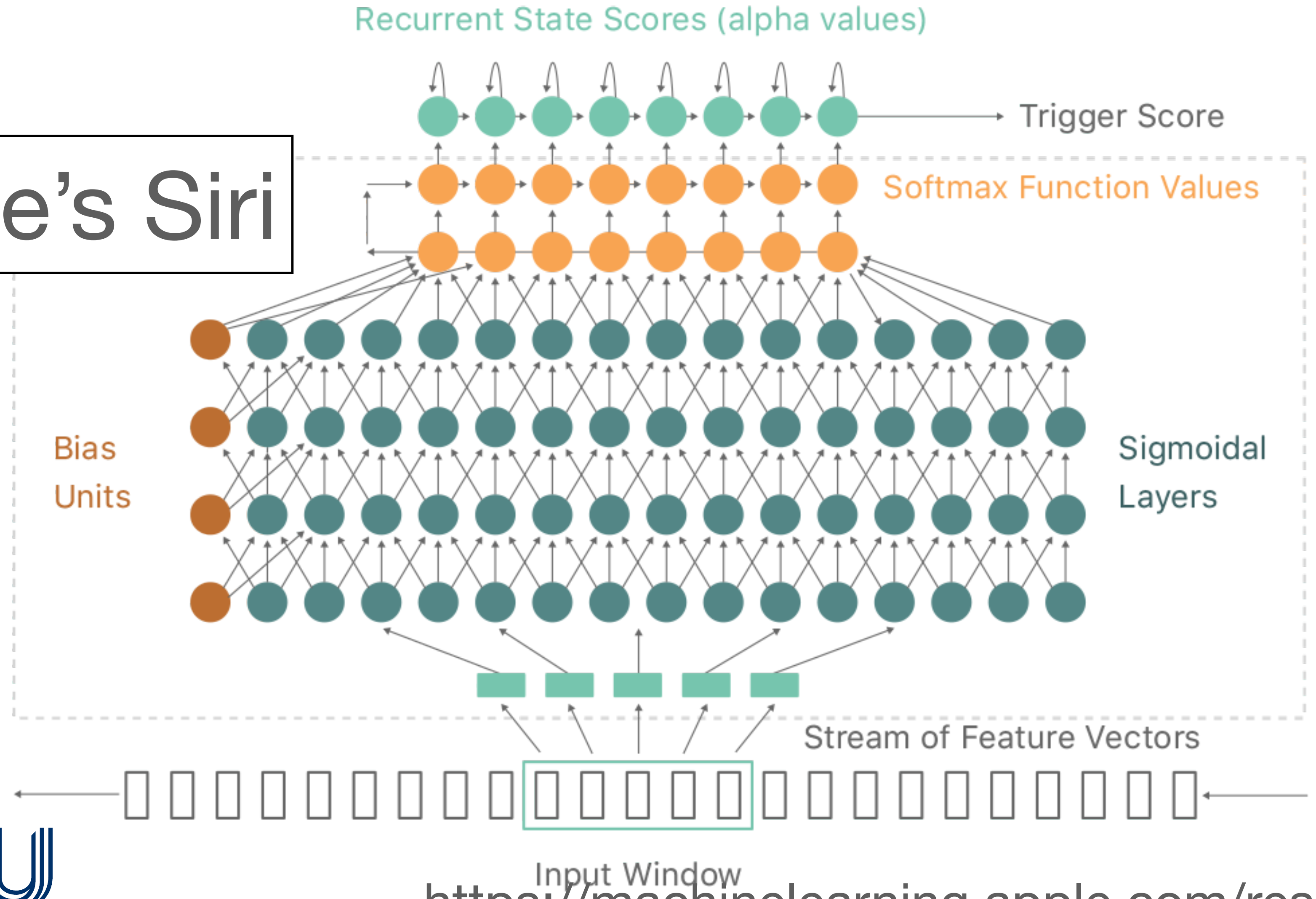
Sperm whale click detection

https://www.nature.com/articles/s41598-019-48909-4



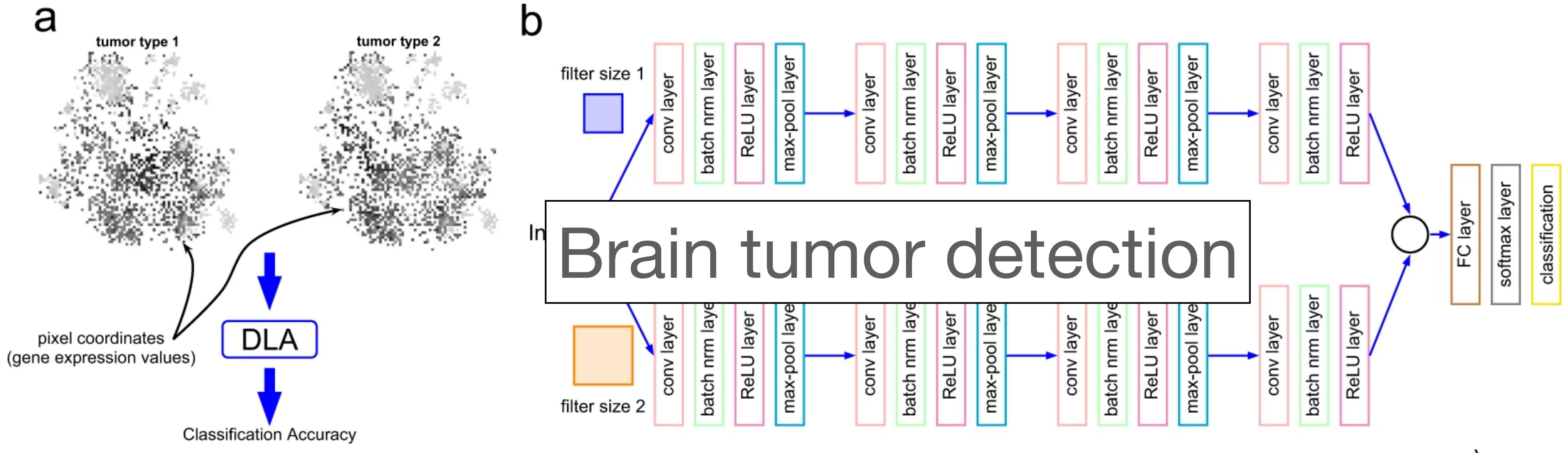
Seismic data

https://www.nature.com/articles/s41467-020-17123-6



Apple’s Siri

https://machinelearning.apple.com/research/hey-siri



Brain tumor detection

https://www.nature.com/articles/s41598-019-47765-6

OK, let's get started understanding CNNs

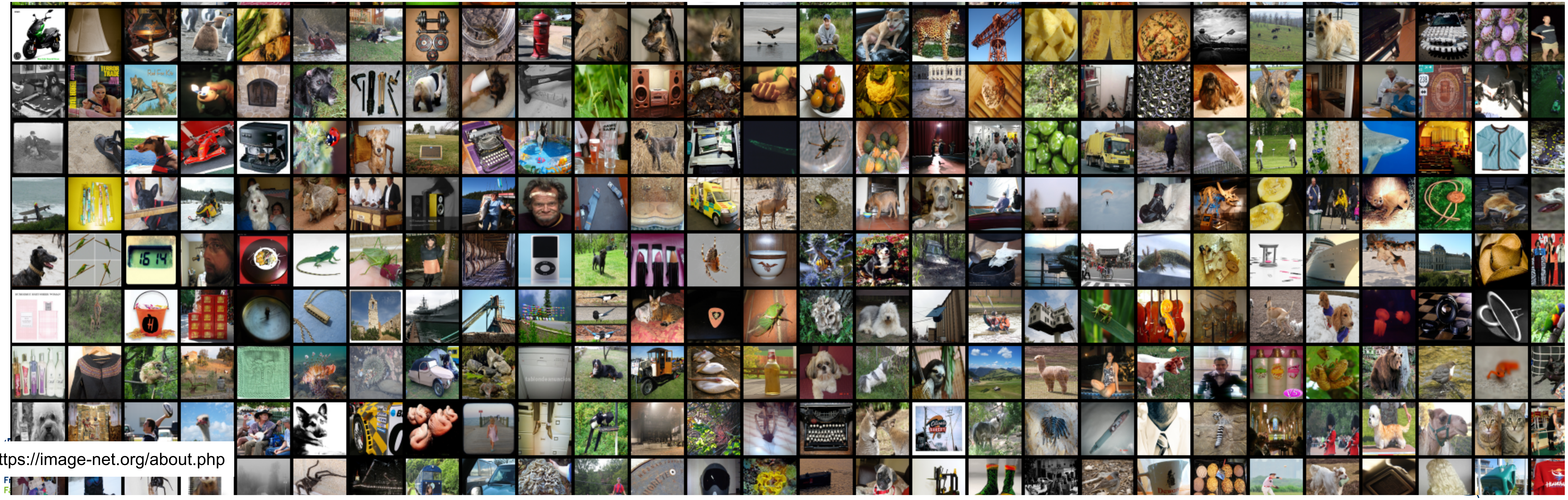
STEPS THAT SHOULD BE COVERED IN A LECTURE ON CNNS

- The basic blocks of machine learning and CNNs

1. The core of CNNs: convolution in artificial neural networks
2. Learning in CNNs
3. (Useful tools: Striding, zero padding and pooling)
4. Typical CNN architectures
5. Data augmentation and pre-processing

1 – THE CORE OF CNNS: THE TECHNICAL DETAILS THAT MATTER

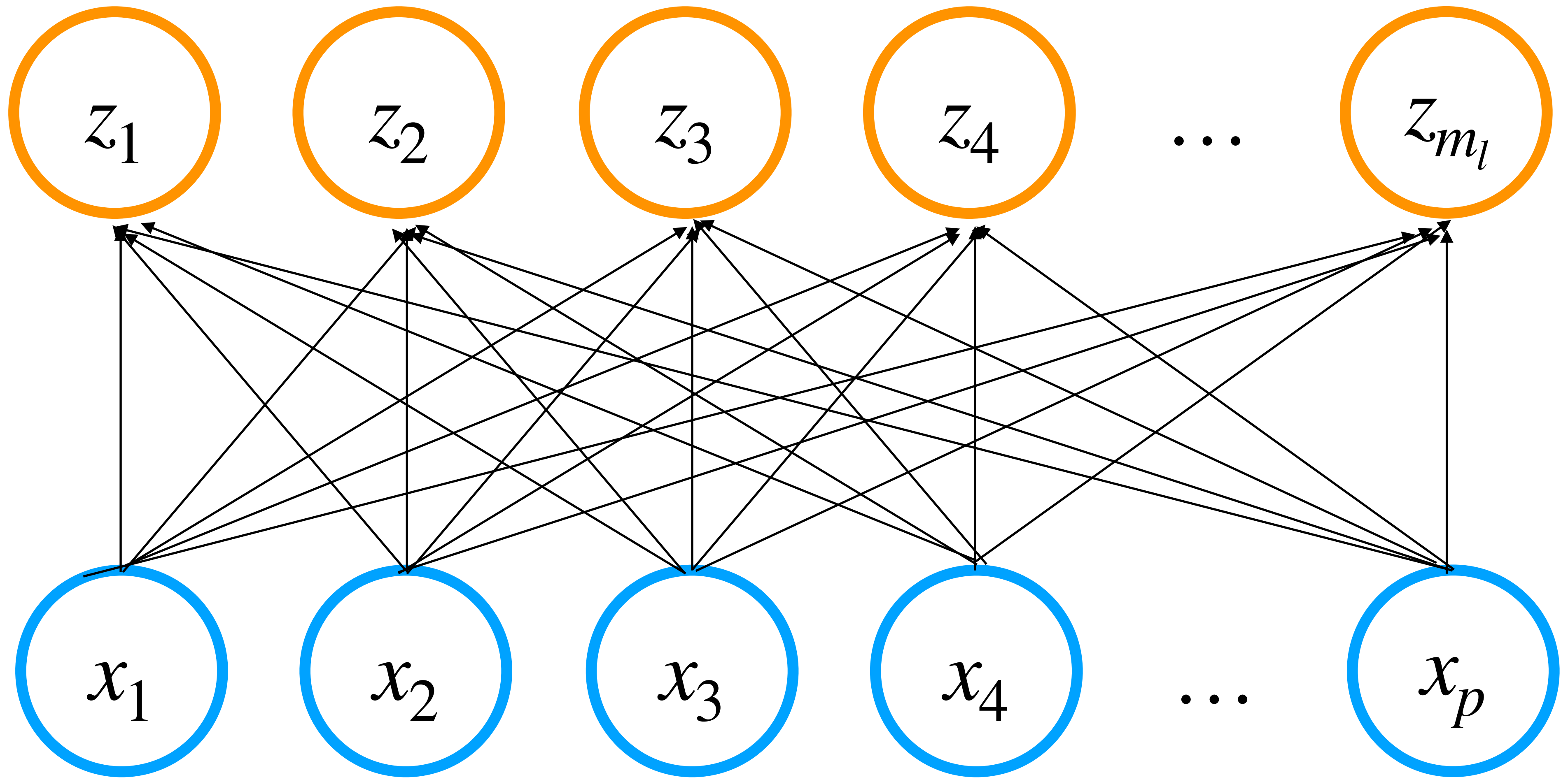
- Basic assumption I: input is one or more images (2-dimensional data on a square grid, e.g. RGB)
- Basic assumption II: fully connected networks don't scale well (as the images get larger)



https://image-net.org/about.php

1 – THE CORE OF CNNS: FULLY CONNECTED NNS DON'T SCALE WELL

Fully connected NN:

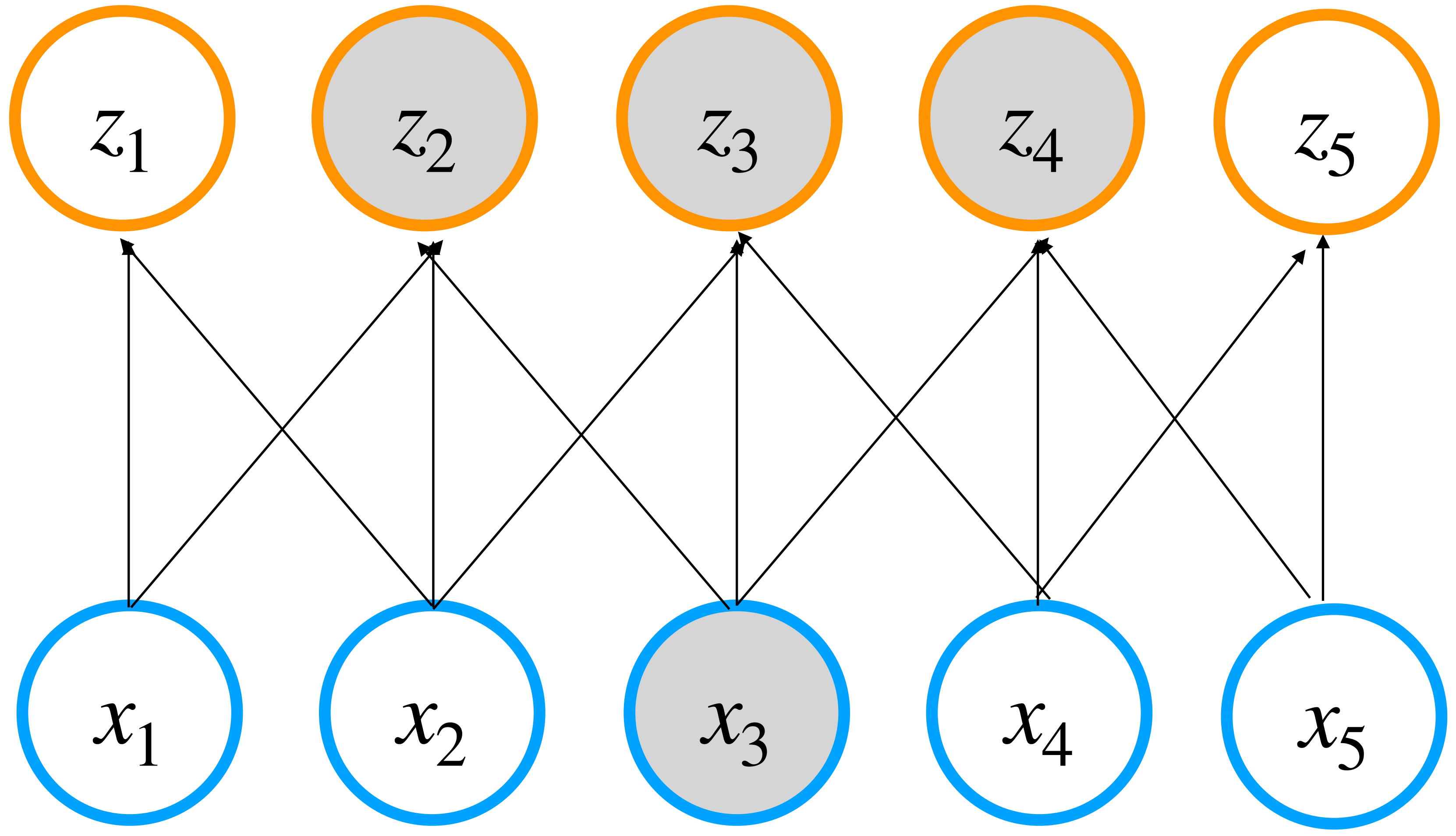


- Imagine an RGB image 32x32 pixels (i.e. 32x32x3=3072 input values)
- Imagine a set of six 5x5 filters to learn the features of the image.
- Output: 28x28 for each of the six filters (=4704 output values).
- Fully connected network: 3072x4704 = 14M weights.

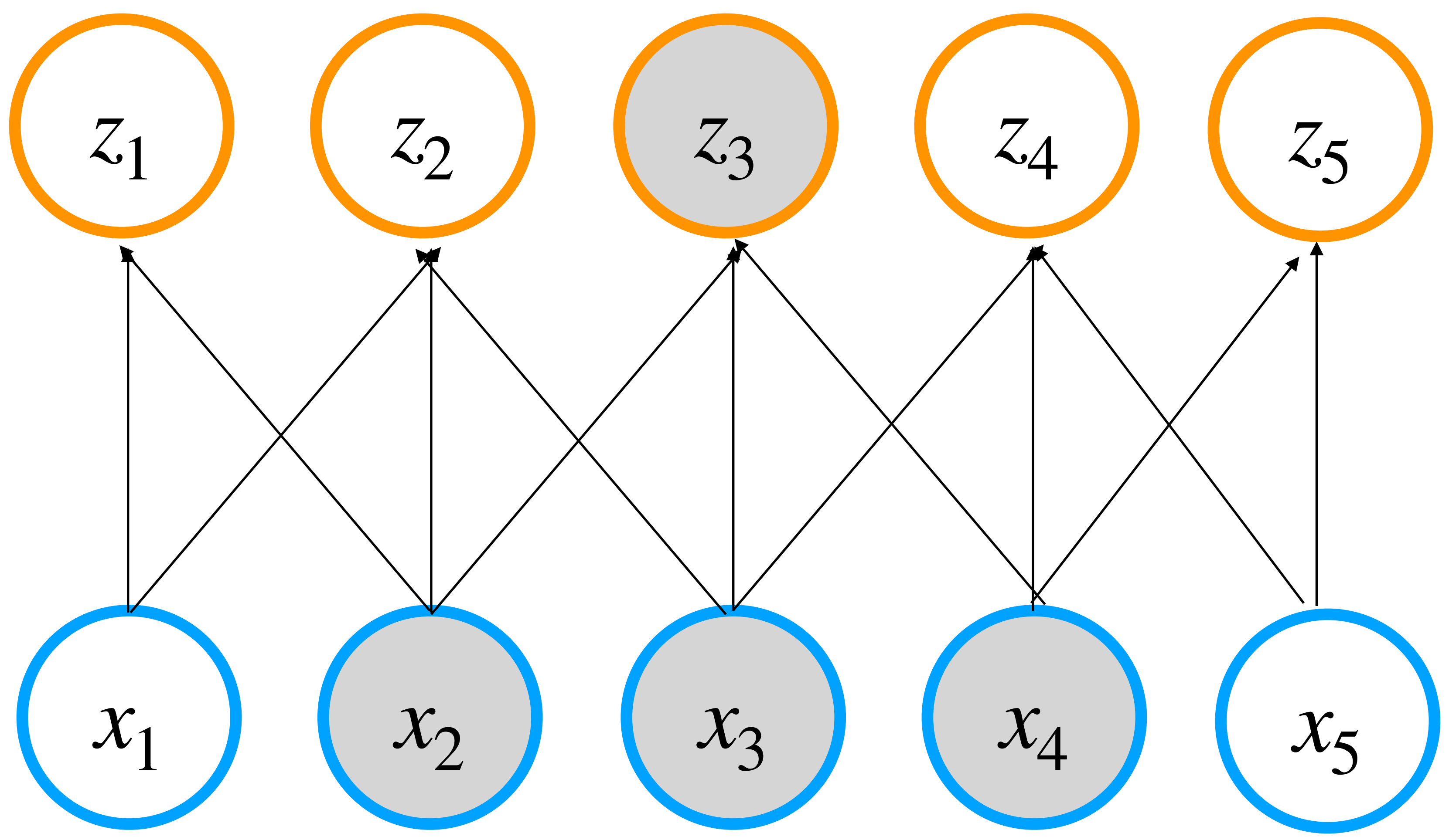
1 – THE CORE OF CNNS: THE TECHNICAL DETAILS THAT MATTER

- Basic assumption I: input is one or more images (2-dimensional data on a square grid, e.g. RGB)
- Basic assumption II: fully connected networks don't scale well (as the images get larger)
- Therefore create a network that contains the following features:
  - **Sparse interactions** (Kernel much smaller than input). Nodes are connected locally (convolution) but not fully connected (exact location of feature in image does not matter)
  - **Parameter sharing** (rather than learning a separate set of parameters for every location, only one set is learned - the Kernel weights - over the whole image)
  - **Translational equivariance** (if we shift the input, we also shift the output)

1 – THE CORE OF CNNS – SPARSE INTERACTIONS



1 – THE CORE OF CNNS: SPARSE INTERACTIONS

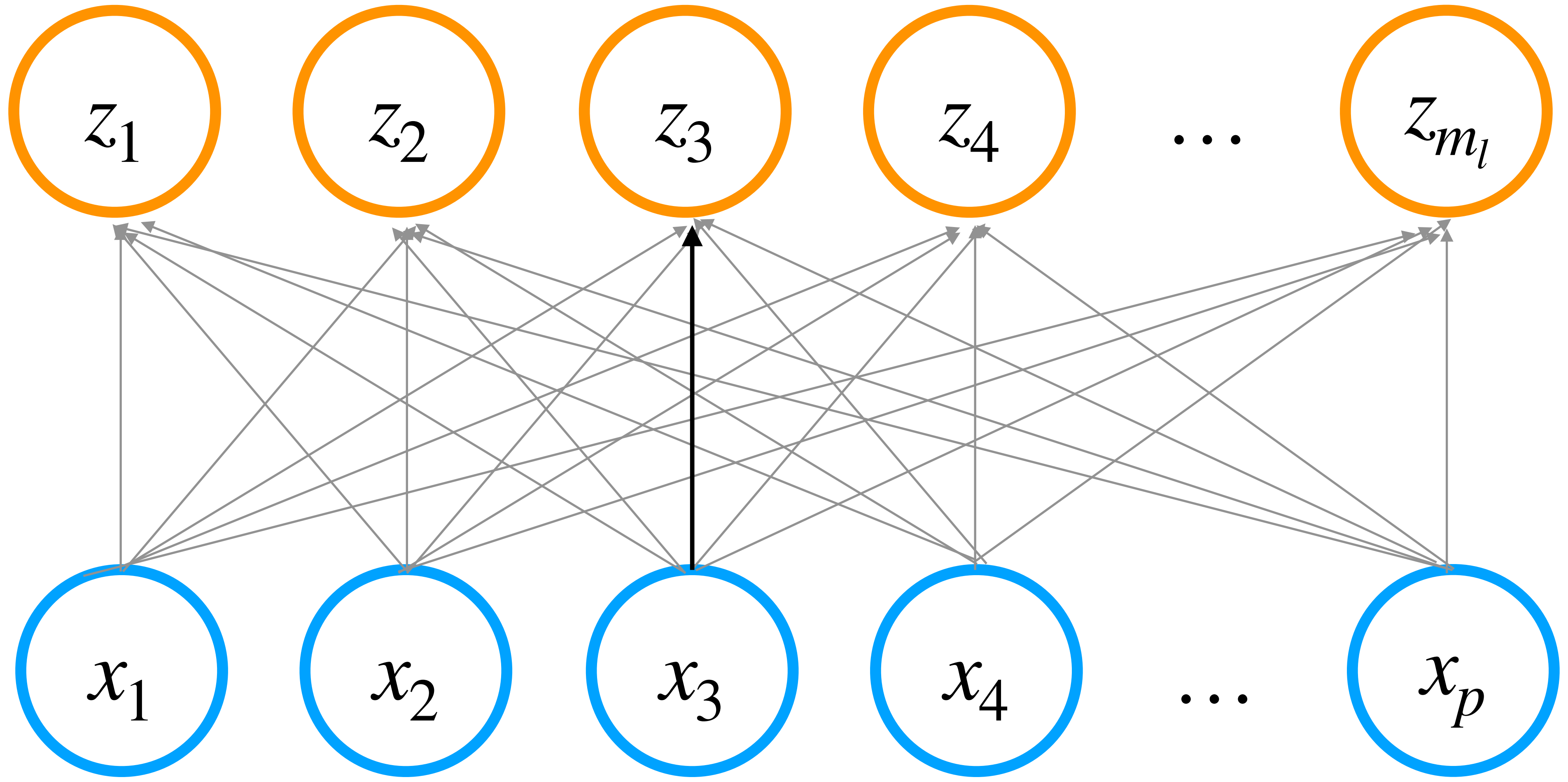


Receptive field

The receptive field in Convolutional Neural Networks (CNN) is the region of the input space that affects a particular unit of the network

1 – THE CORE OF CNNS: WEIGHT SHARING

Fully connected NN: each weight only used once

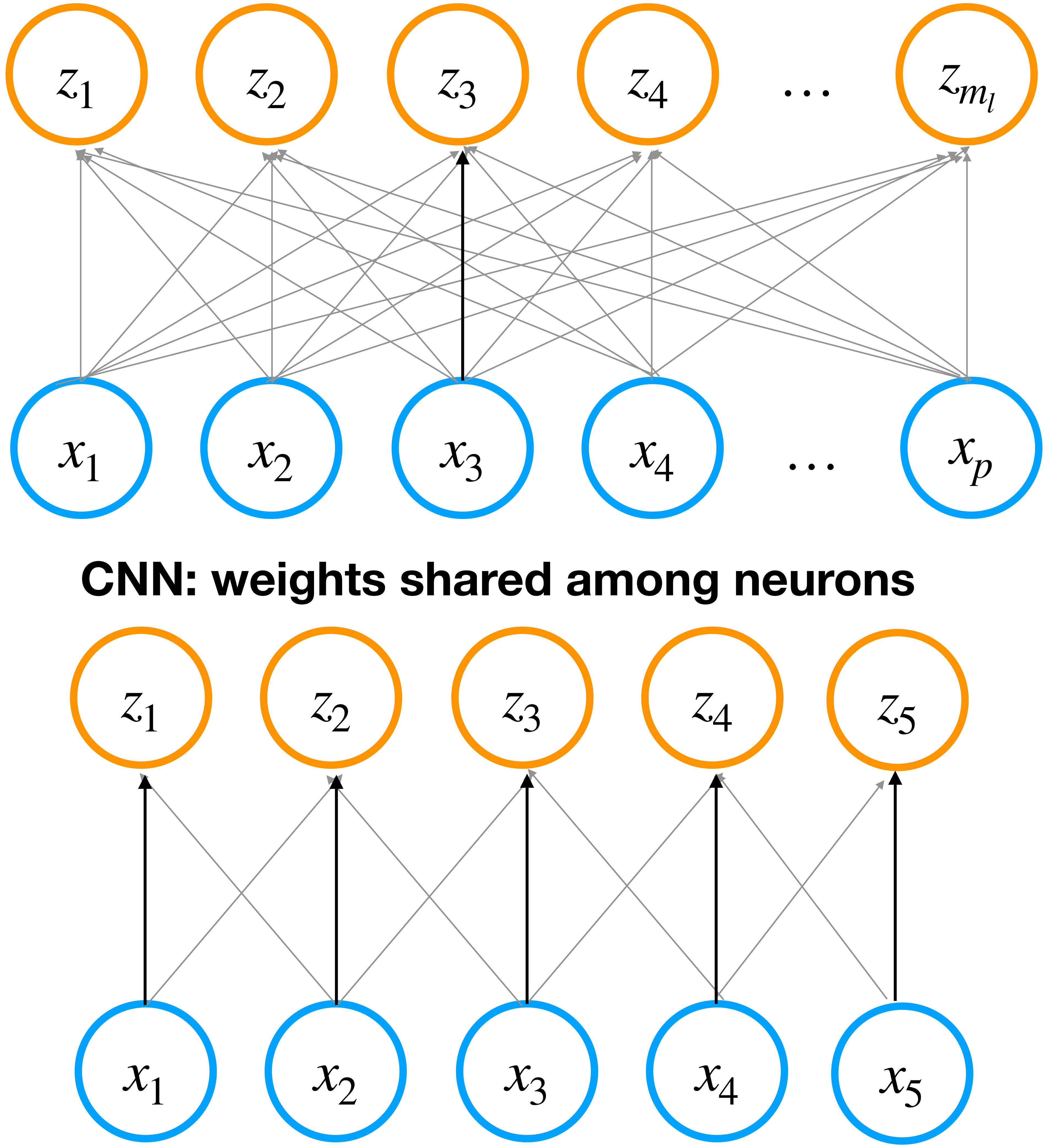


- What is important in one part of the image, is likely also important in another part of the image

1 – THE CORE OF CNNS: WEIGHT SHARING

Fully connected NN: each weight only used once

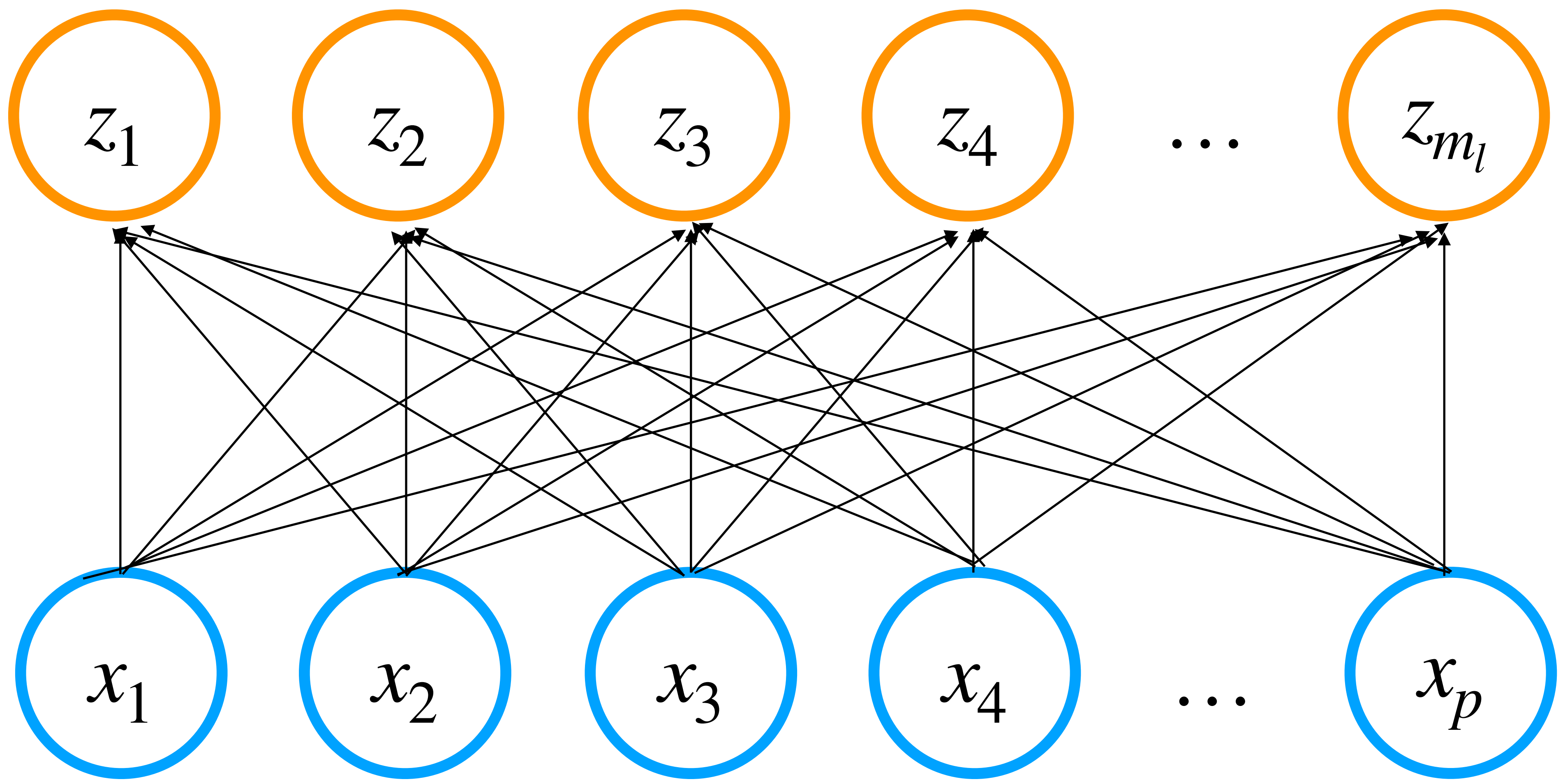
CNN: weights shared among neurons



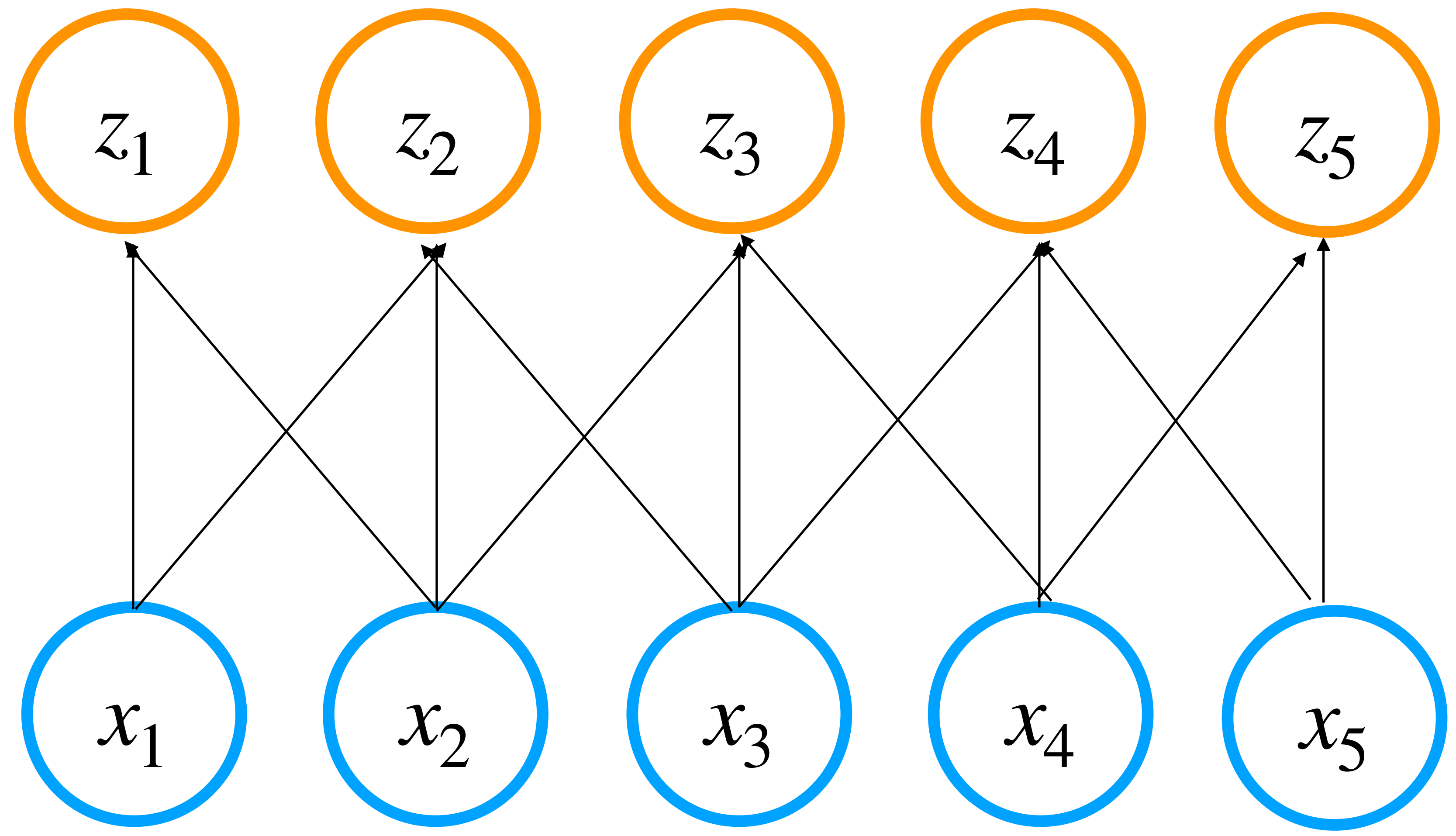
- What is important in one part of the image, is likely also important in another part of the image

1 – THE CORE OF CNNS: SOME NUMBERS (SPARSE INTERACTIONS AND PARAMETER SHARING)

Fully connected NN:



CNN:



- Imagine an RGB image 32x32 pixels (i.e. 32x32x3=3072 input values)
- Imagine a set of six 5x5 filters to learn the features of the image.
- Output: 28x28 for each of the six filters (=4704 output values).
- Fully connected network: 3072x4704 = 14M weights.
- For CNN: Each filter has (5x5+1) parameters, for the three layers, i.e. (5x5+1)x3x6 = 468 weights

Friedrich-Alexander-Universität
Faculty of Sciences

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

1 – THE CORE OF CNNS: UNDERSTANDING CONVOLUTION – MATHEMATICAL BASICS

- In mathematics, convolution defined as operation $(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$ (f, g real-valued functions)

- Convolution is commutative:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau, \quad \text{Substitution: } x = t - \tau, \frac{dx}{d\tau} = -1$$

$$= -\int_{\infty}^{-\infty} f(t-x)g(x)dx = \int_{-\infty}^{\infty} g(\tau)f(t-\tau)dx = (g * f)(t)$$

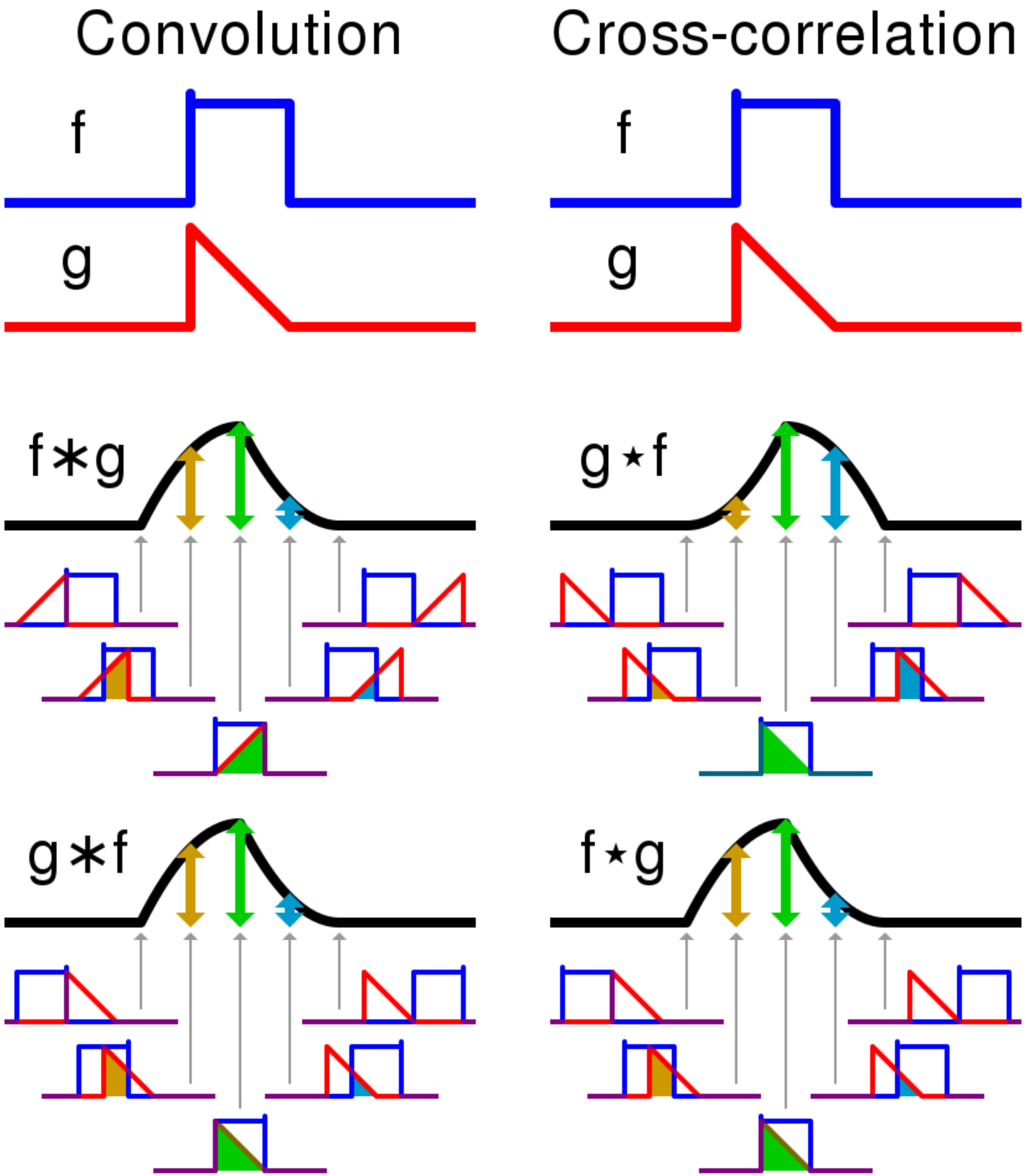
- In discrete form convolution can be written as: $(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m)$

- In the CNN: two-dim image $I(i,j)$ and kernel (or filter) functions $K(i,j)$, for which a convolution can be written as

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

, with $K(i,j)$, $I(i,j) = 0$ if i, j outside the image or kernel

CONVOLUTION AND CROSS CORRELATION



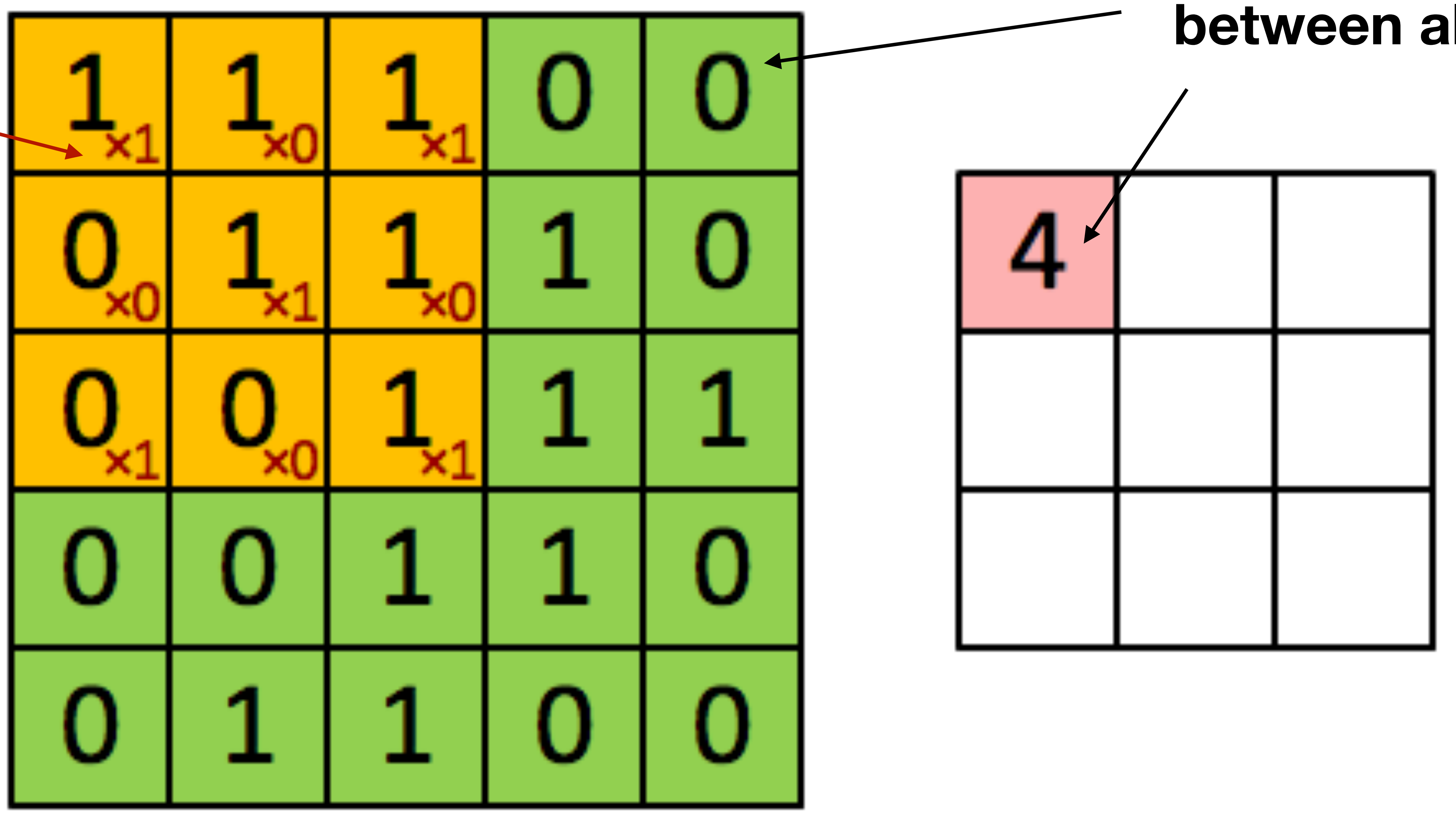
1 – THE CORE OF CNNS: IMAGE CONVO LE

1	0	1
0	1	0
1	0	1

Kernel

Weights in kernel stay constant as we move across the image

No interaction between all neurons

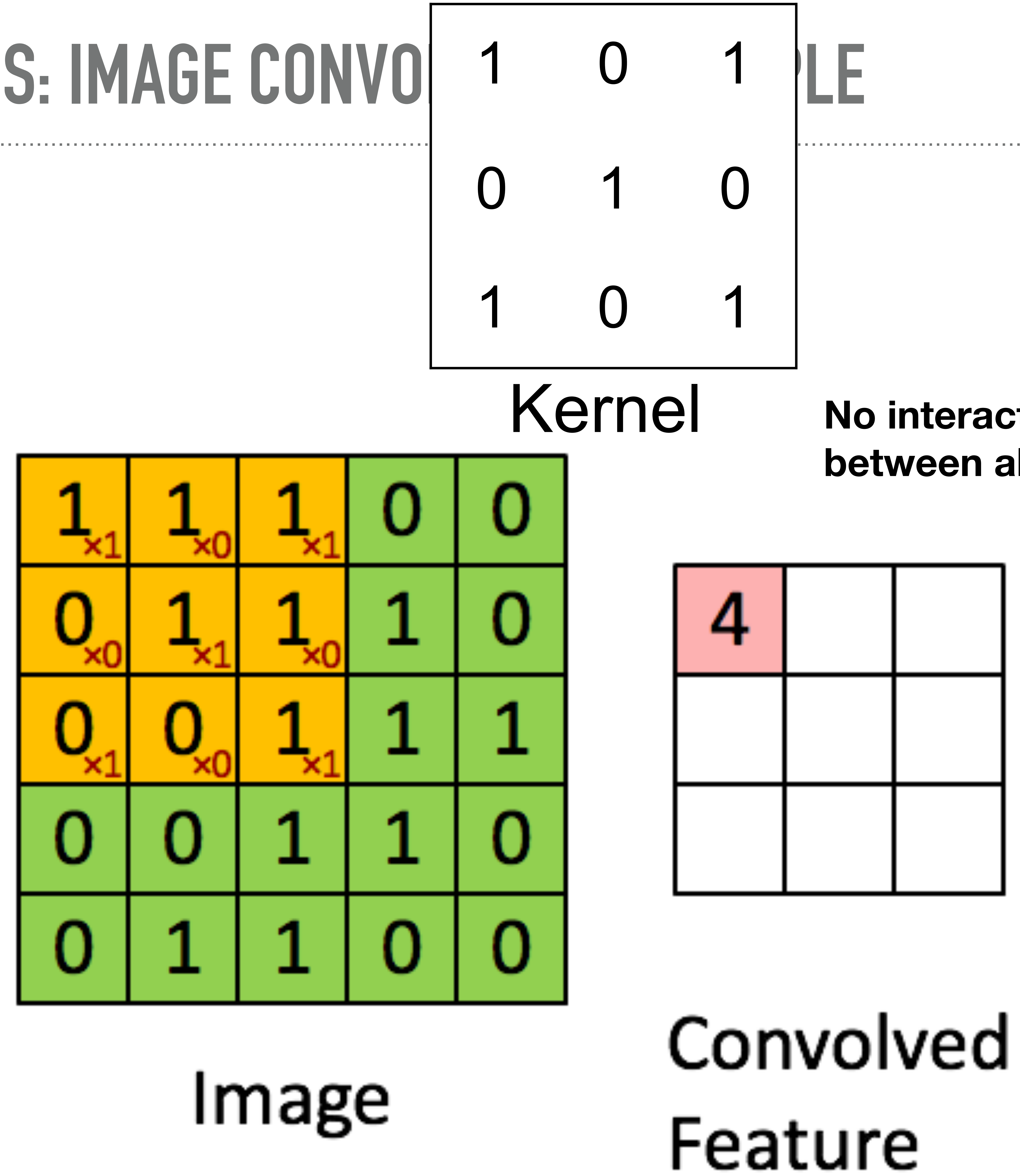


Image

Convolved Feature

1 – THE CORE OF CNNS: IMAGE CONVO LE

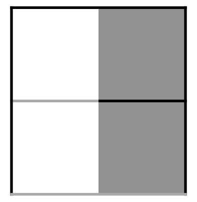
Weights in kernel stay constant as we move across the image



No interaction between all neurons

1 – THE CORE OF CNNS: EXAMPLE CONVOLUTION FOR EDGE DETECTION

Vertical edge detection



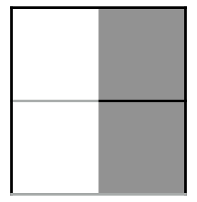
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$\times$

$=$

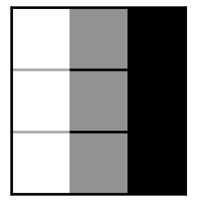
1 – THE CORE OF CNNS: EXAMPLE CONVOLUTION FOR EDGE DETECTION

Vertical edge detection



10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

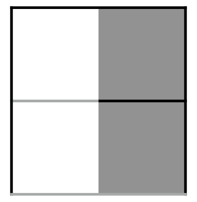
×

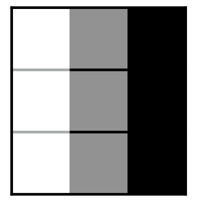


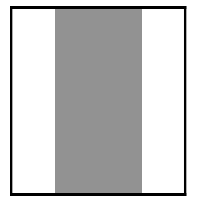
1	0	-1
1	0	-1
1	0	-1

=

Vertical edge detection







$$
\begin{bmatrix}
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0 \\
10 & 10 & 10 & 0 & 0 & 0
\end{bmatrix}
\times
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
=
\begin{bmatrix}
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0 \\
0 & 30 & 30 & 0
\end{bmatrix}
$$

1 – THE CORE OF CNNS: EXAMPLE CONVOLUTION FOR EDGE DETECTION

Horizontal edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

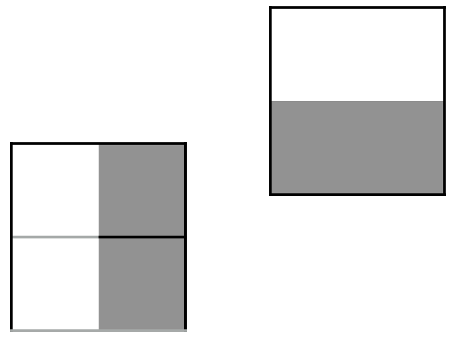
$\times$

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

$=$

1 – THE CORE OF CNNS: EXAMPLE CONVOLUTION FOR EDGE DETECTION

Horizontal edge detection

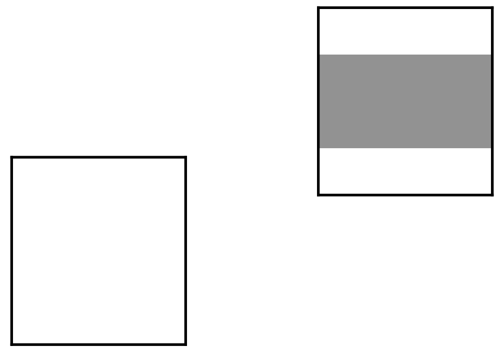


10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$\times$

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

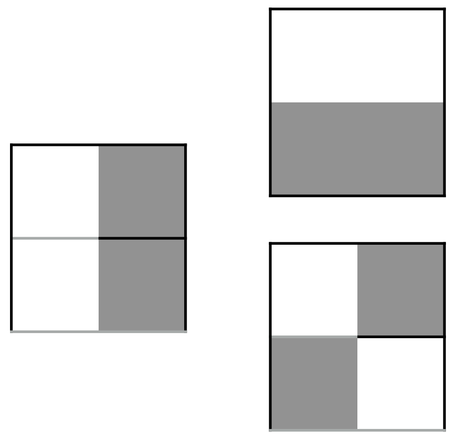
$=$



0	0	0	0
30	30	30	30
30	30	30	30
0	0	0	0

1 – THE CORE OF CNNS: EXAMPLE CONVOLUTION FOR EDGE DETECTION

Horizontal edge detection

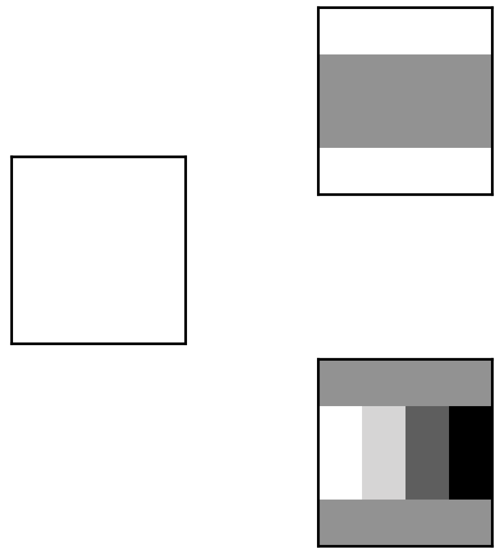


10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$\times$

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

$=$



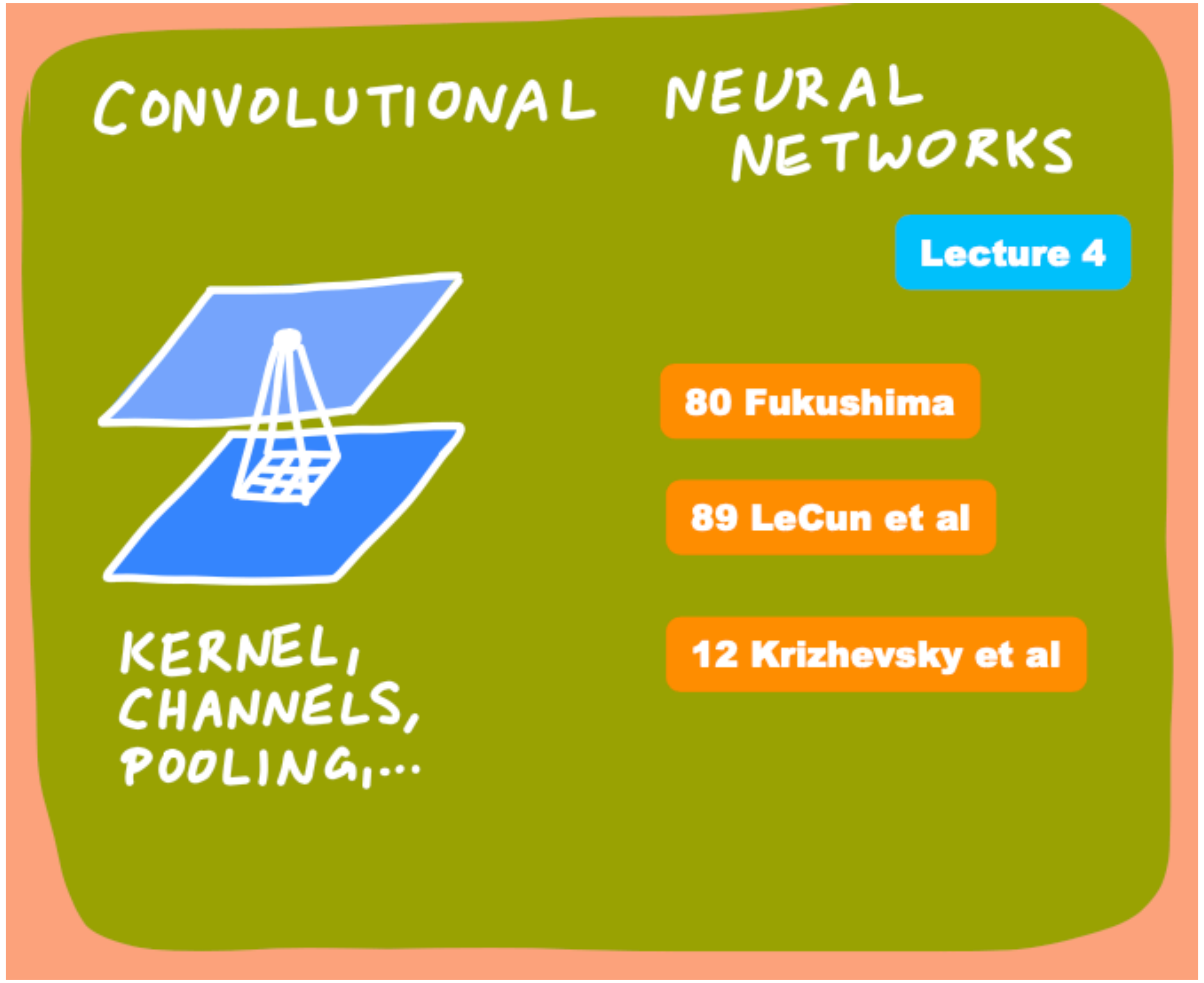
0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

1 – THE CORE OF CNNS: EDGE DETECTION AT WORK

https://en.wikipedia.org/wiki/Edge_detection#/media/File:Ääretuvastuse_näide.png

1 – THE CORE OF CNNS: WHY ARE CNNS SO USEFUL?

- Reduce number of computations (**sparse interactions, parameter sharing**) and memory requirements
- Intuitively: CNN will learn filters that activate for some kind of feature in the image like an edge or blotch of some color; often useful to learn that regardless of position in image **translational equivariance:**
- Equivariance not **always** useful: e.g. when different features are present at different positions in an image, like a face in the upper part. Then don't use weight sharing



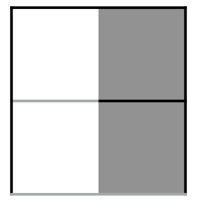
https://florianmarquardt.github.io/deep_learning_basics_linkmap.html

Now that we understand the basic mechanics of a CNN, how does the learning work?

Friedrich-Alexander-Universität
Faculty of Sciences

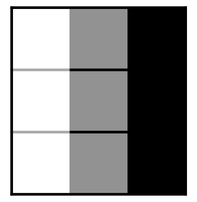
2 UNDERSTAND LEARNING IN CNNS

I assume this has been covered in the general introduction to ANNs



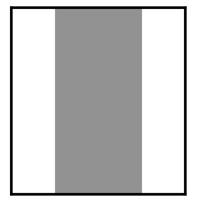
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

×



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

=

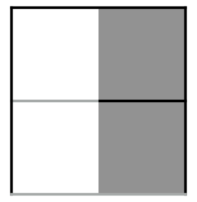


0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

2 UNDERSTAND LEARNING IN CNNS

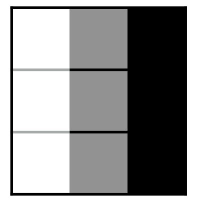
I assume this has been covered in the general introduction to ANNs

1	0	-1
2	0	-2
1	0	-1



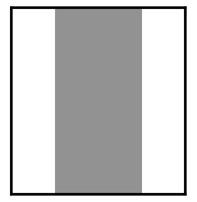
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$\times$



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

$=$



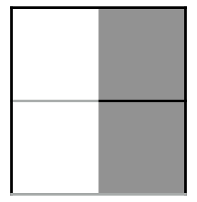
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

2 UNDERSTAND LEARNING IN CNNS

Sobel filter

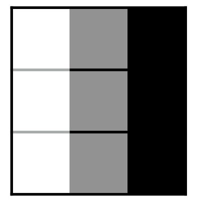
1	0	-1
2	0	-2
1	0	-1

I assume this has been covered in the general introduction to ANNs



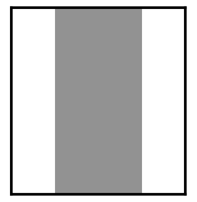
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$\times$



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

$=$



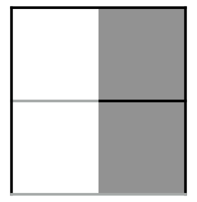
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

2 UNDERSTAND LEARNING IN CNNS

Sobel filter

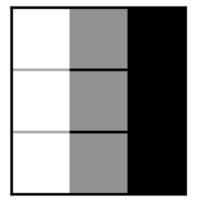
1	0	-1
2	0	-2
1	0	-1

I assume this has been covered in the general introduction to ANNs



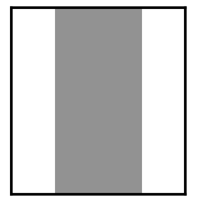
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$\times$



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

$=$

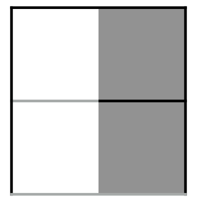


0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

3	0	-3
10	0	-10
3	0	-3

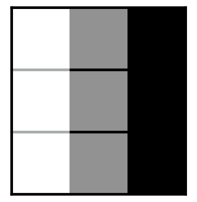
2 UNDERSTAND LEARNING IN CNNS

I assume this has been covered in the general introduction to ANNs



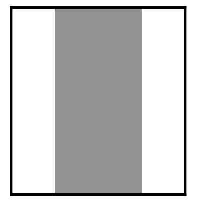
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

×



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

=



0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Sobel filter

1	0	-1
2	0	-2
1	0	-1

Schorr filter

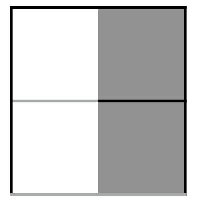
3	0	-3
10	0	-10
3	0	-3

2 UNDERSTAND LEARNING IN CNNS

I assume this has been covered in the general introduction to ANNs

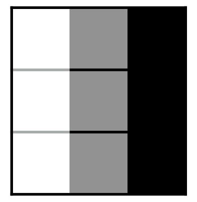
Sobel filter

1	0	-1
2	0	-2
1	0	-1



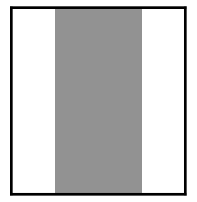
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$\times$



w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$=$



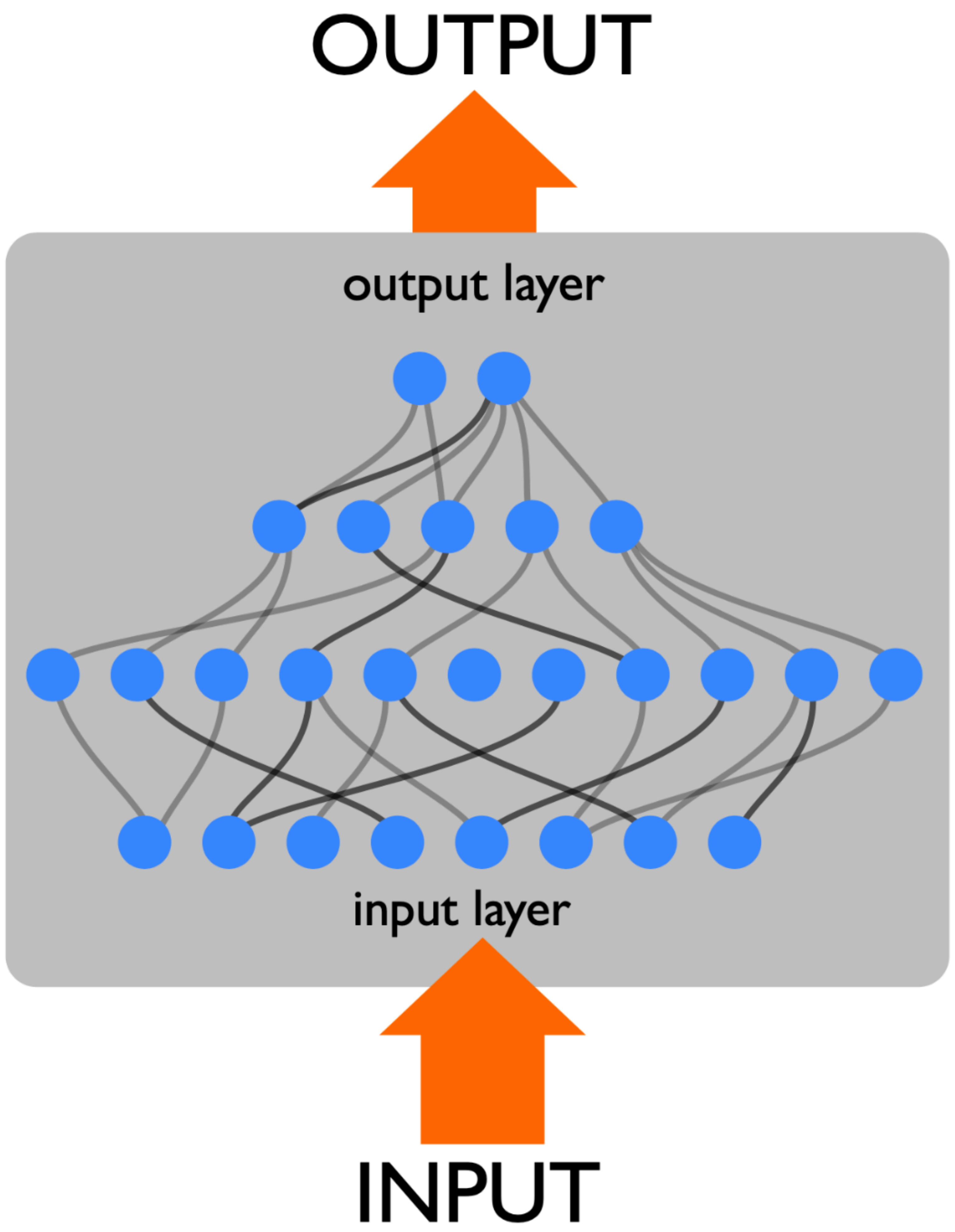
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Schorr filter

3	0	-3
10	0	-10
3	0	-3

2 UNDERSTAND LEARNING IN CNNS: REMEMBER – TRAINING OF WEIGHTS

A neural network



=

OUTPUT

Complicated nonlinear function that depends on all the weights and biases

$$y^{\text{out}} = F_w(y^{\text{in}})$$

INPUT

I assume this has been covered in the general introduction to ANNs

https://owncloud.gwdg.de/index.php/s/qetLJgXMW6u3FwC

2 UNDERSTAND LEARNING IN CNNS: REMEMBER – TRAINING OF WEIGHTS

We have: $y^{\text{out}} = F_w(y^{\text{in}})$

neural network
(w here also stands for the b

I assume this has been covered in the general introduction to ANNs

We would like: $y^{\text{out}} \approx F(y^{\text{in}})$

desired "target" function

Cost function measures deviation:

$$C(w) = \frac{1}{2}\langle \| F_w(y^{\text{in}}) - F(y^{\text{in}}) \|^2 \rangle$$

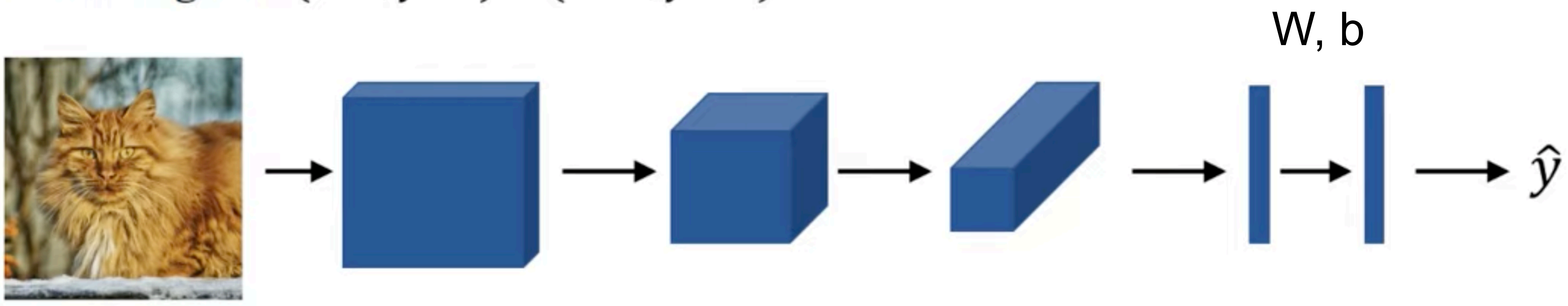
vector norm

average over all samples

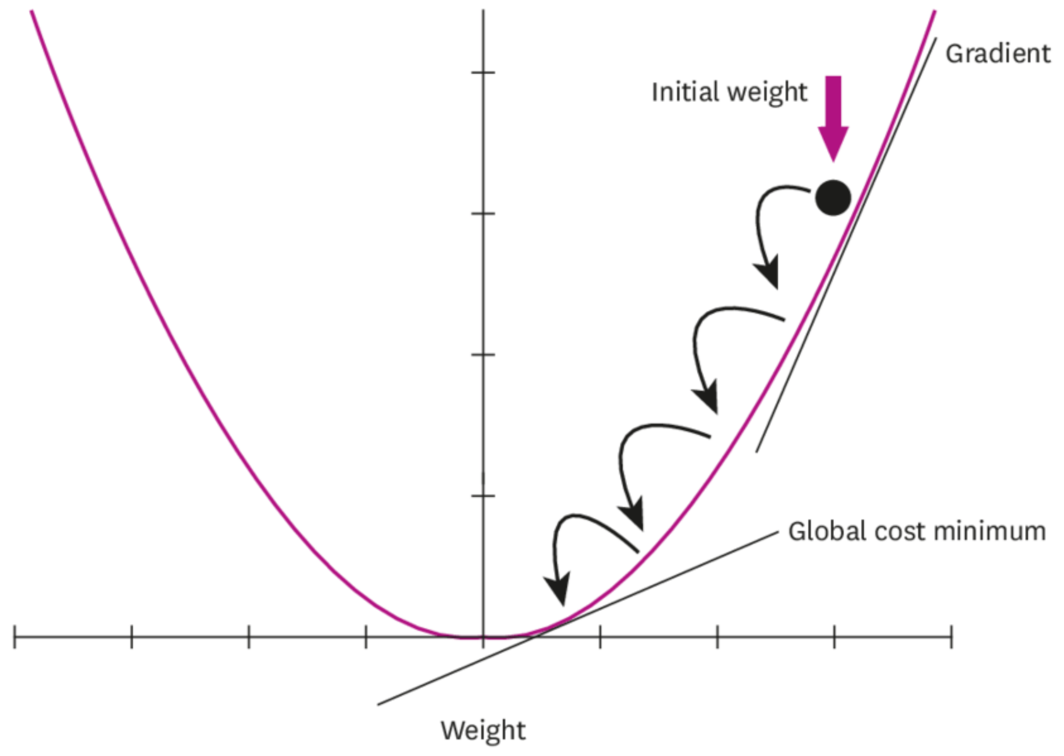
https://owncloud.gwdg.de/index.php/s/qetLJgXMW6u3FwC

2 UNDERSTAND LEARNING IN CNNS

Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$.



- Define a cost function. $$J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$
- Use gradient descent to derive the set of W, b, for which the cost is minimised



Before we can build our CNN, we need three more things

3 USEFUL TOOLS – STRIDING, PADDING, POOLING

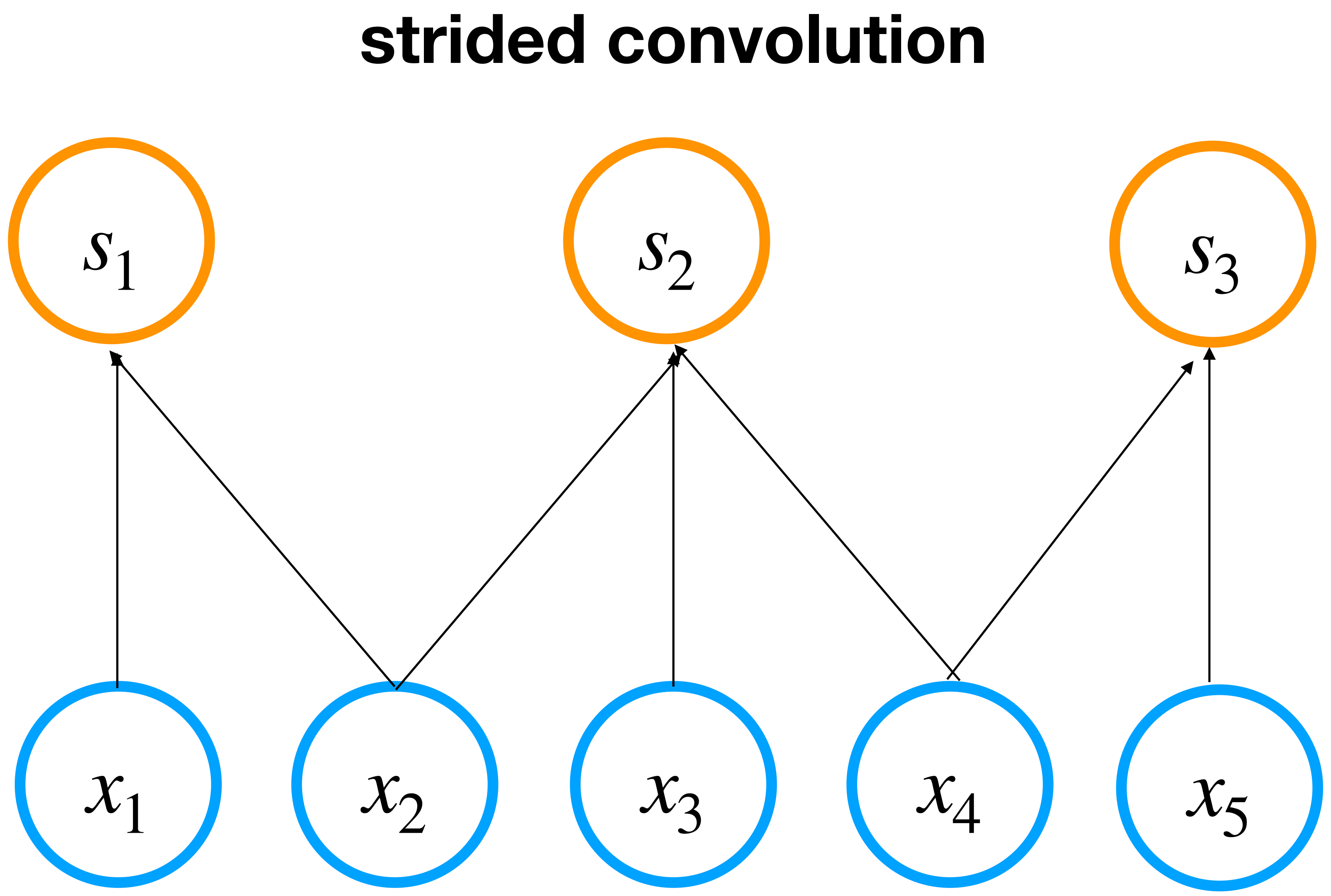
- To reduce computational cost we can shift the kernel by more the one pixel, that is, choose a stride $s > 1$

- This is equivalent to a down sampled convolution:

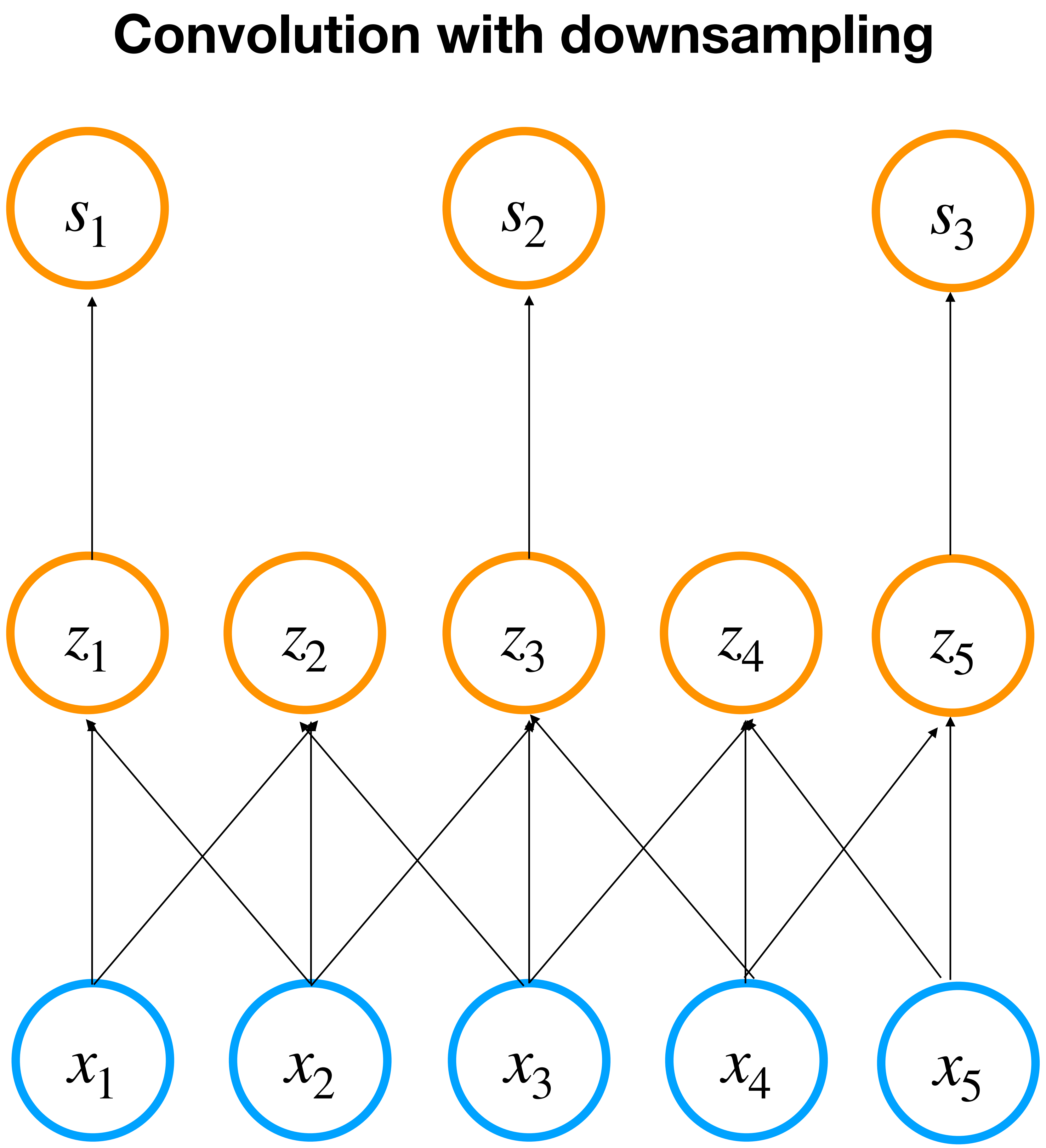
$$z^{[l]}_{i,j,k} = c(K, a^{[l-1]}, s)_{i,j,k} = \sum_m \sum_n \sum_p a^{[l-1]}_{m,(j-1)\times s+n,(k-1)\times s+p} K^{[l]}_{i,m,n,p}$$

- Which comes at the expense that features can not be extracted as finely

3 USEFUL TOOLS – STRIDING, PADDING, POOLING



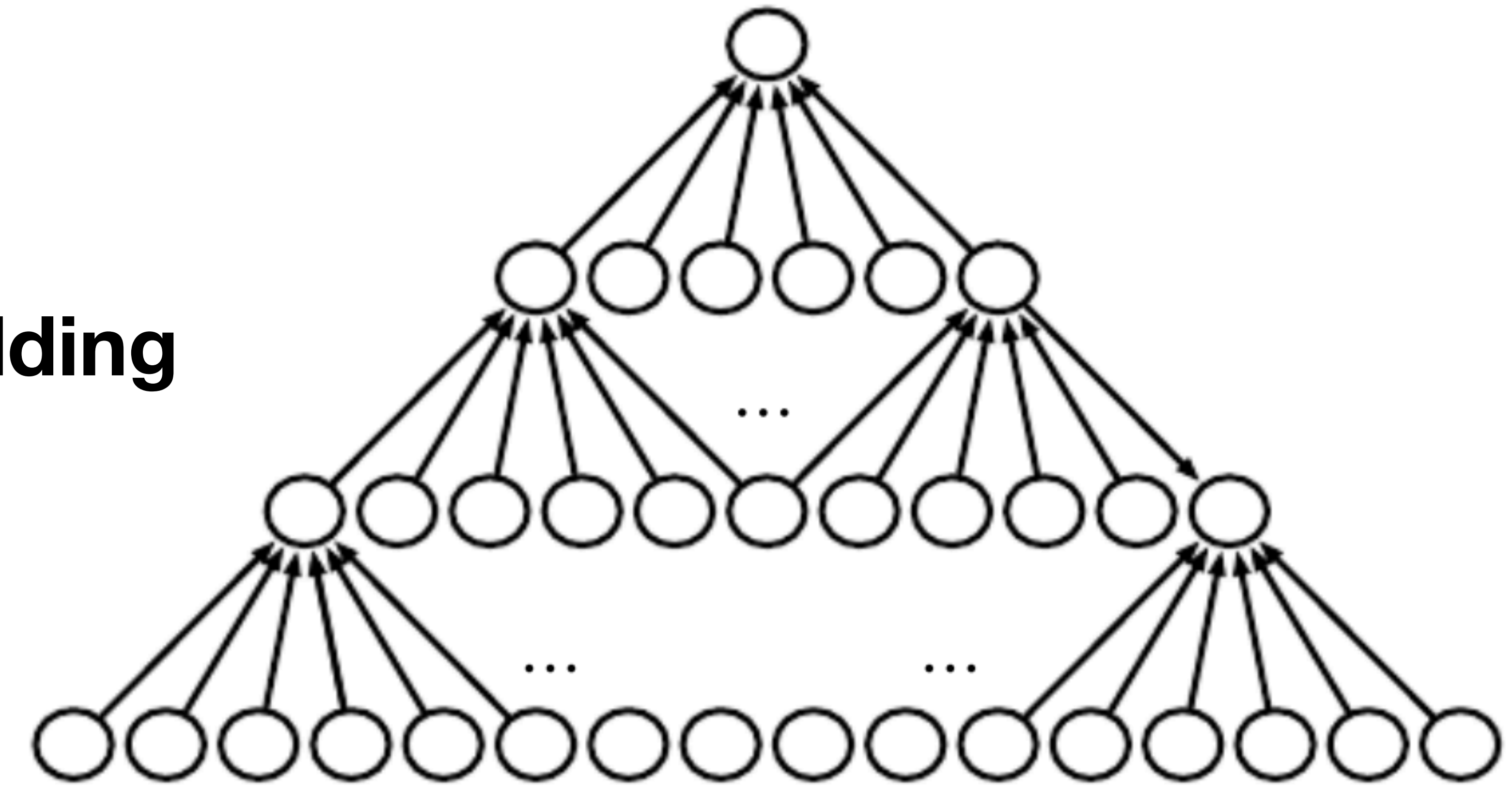
=



3 USEFUL TOOLS - STRIDING, PADDING, POOLING

- If only **valid convolutions** (=kernel fully contained in image) are considered, image will quickly shrink in size

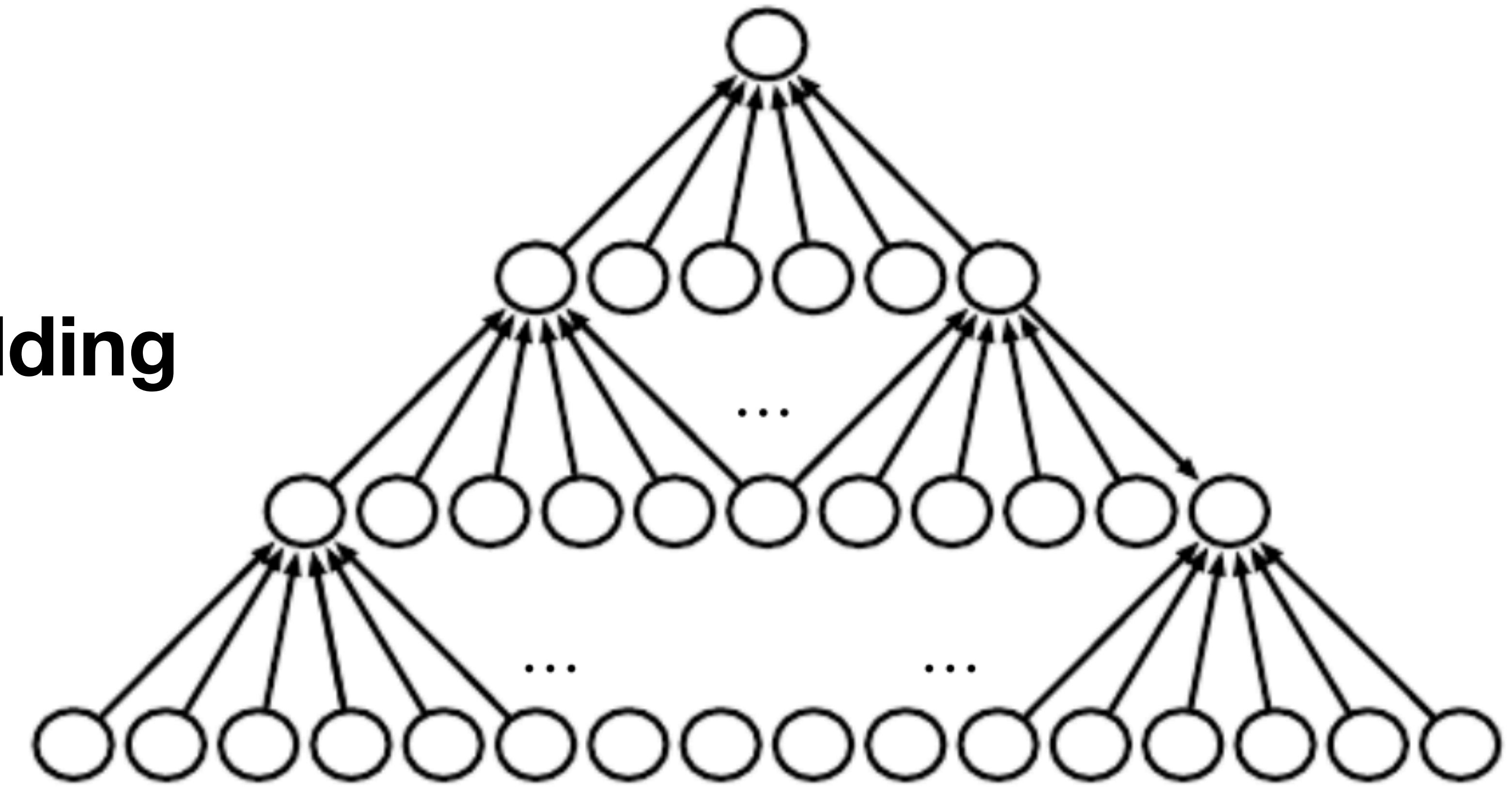
Valid convolution: no zero-padding



3 USEFUL TOOLS – STRIDING, PADDING, POOLING

➤ If only **valid convolutions** (=kernel fully contained in image) are considered, image will quickly shrink in size

Valid convolution: no zero-padding



Same convolution: zero padding so that dimensions don't change

- Padding edges with p zeros, allows one to make arbitrarily deep CNNs

- Avoids some edge effects

Friedrich-Alexander-Universität
Faculty of Sciences

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

3 USEFUL TOOLS - STRIDING, PADDING - OUTPUT SIZE OF CNN

➤ With input of width $W^{[l-1]}$, and kernel with receptive field of size $f^{[l]}$, stride $s^{[l]}$ and zero padding $p^{[l]}$, one output image in the convolutional layer will have width

$$W^{[l]} = (W^{[l-1]} - f^{[l]} + 2p^{[l]})/s^{[l]} + 1$$

➤ Strides constrained so that $W^{[l]}$ is an integer

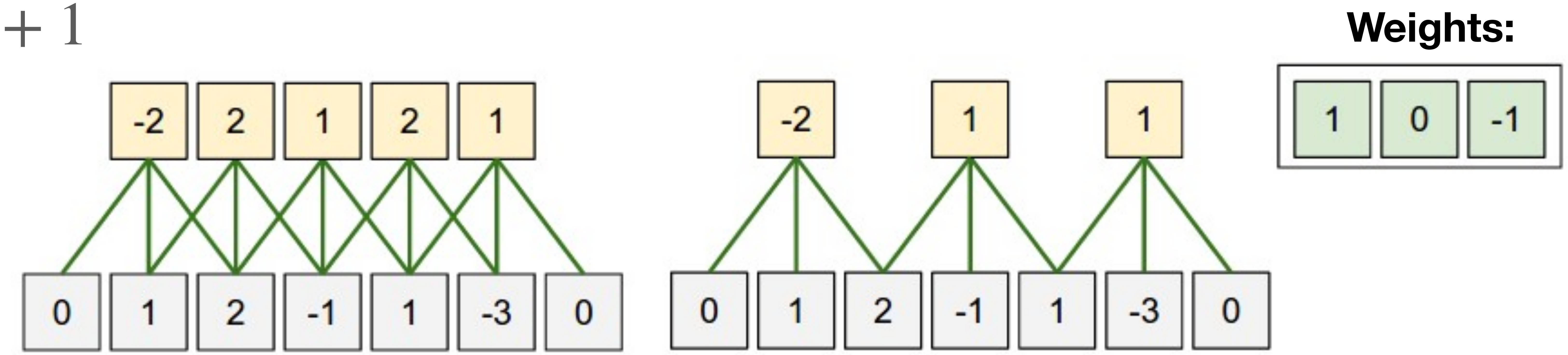
➤ With $s^{[l]} = 1$, zero padding with $p^{[l]} = (f^{[l]} - 1)/2$ will give $W^{[l]} = W^{[l-1]}$

3 USEFUL TOOLS – STRIDING, PADDING, POOLING – OUTPUT SIZE

- With input of width w, and kernel with receptive field of size f, stride s and zero padding p, one output image in the convolutional layer will have width

$$w_2 = (w - f + 2p)/s + 1$$

- Strides constrained so that w_2 is an integer
- With $s = 1$, zero padding with $p = (f - 1)/2$ will give $w_2 = w$



$$W^{[1]} = 7$$
$$f^{[2]} = 3$$
$$p^{[2]} = 0$$
$$s^{[2]} = 1$$

$$\Rightarrow W^{[2]} = \frac{7 - 3 + 2 \cdot 0}{1} + 1 = 5$$

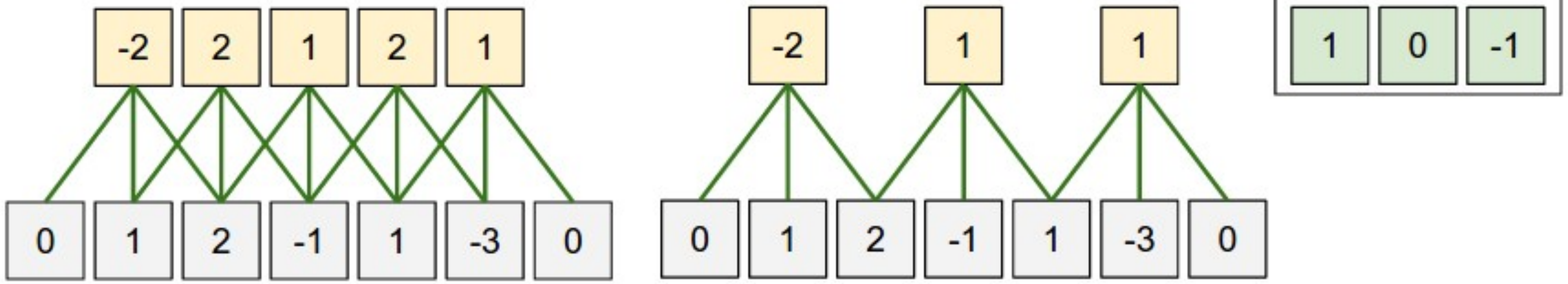
3 USEFUL TOOLS – STRIDING, PADDING, POOLING – OUTPUT SIZE

- With input of width w, and kernel with receptive field of size f, stride s and zero padding p, one output image in the convolutional layer will have width

$$w_2 = (w - f + 2p)/s + 1$$

- Strides constrained so that w_2 is an integer
- With $s = 1$, zero padding with $p = (f-1)/2$ will give $w_2 = w$

Weights:



$$W^{[1]} = 7$$
$$f^{[2]} = 3$$
$$p^{[2]} = 0$$
$$s^{[2]} = 1$$

$$\Rightarrow W^{[2]} = \frac{7 - 3 + 2 \cdot 0}{1} + 1 = 5$$

$$W^{[1]} =$$
$$f^{[2]} =$$
$$p^{[2]} =$$
$$s^{[2]} =$$

Friedrich-Alexander-Universität
Faculty of Sciences

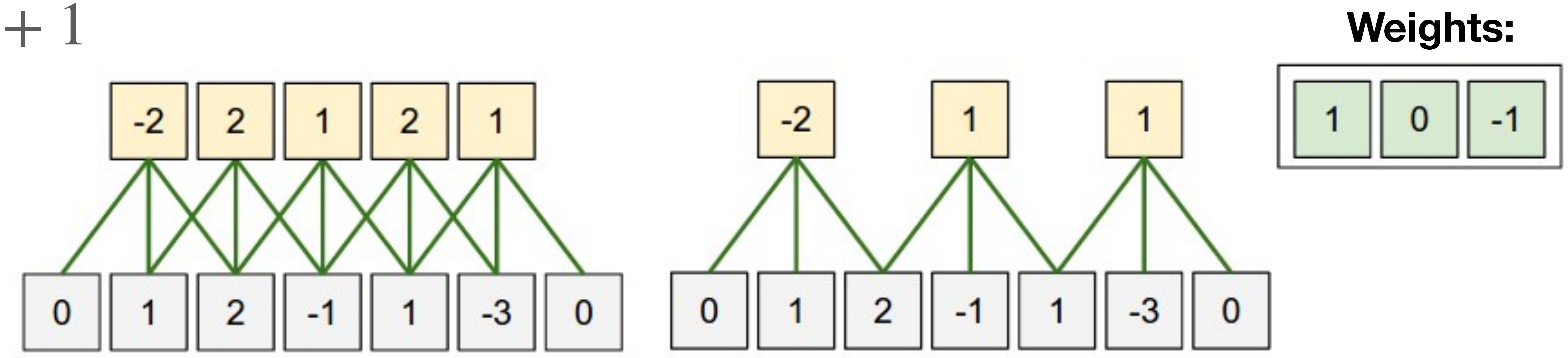
ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

3 USEFUL TOOLS – STRIDING, PADDING, POOLING – OUTPUT SIZE

- With input of width w, and kernel with receptive field of size f, stride s and zero padding p, one output image in the convolutional layer will have width

$$w_2 = (w - f + 2p)/s + 1$$

- Strides constrained so that w_2 is an integer
- With $s = 1$, zero padding with $p = (f - 1)/2$ will give $w_2 = w$



$$W^{[1]} = 7 \qquad f^{[2]} = 3 \qquad p^{[2]} = 0 \qquad s^{[2]} = 1$$

$$W^{[1]} = 7 \qquad f^{[2]} = 3 \qquad p^{[2]} = 0 \qquad s^{[2]} = 2$$

$$\Rightarrow W^{[2]} = \frac{7 - 3 + 2 \cdot 0}{1} + 1 = 5$$

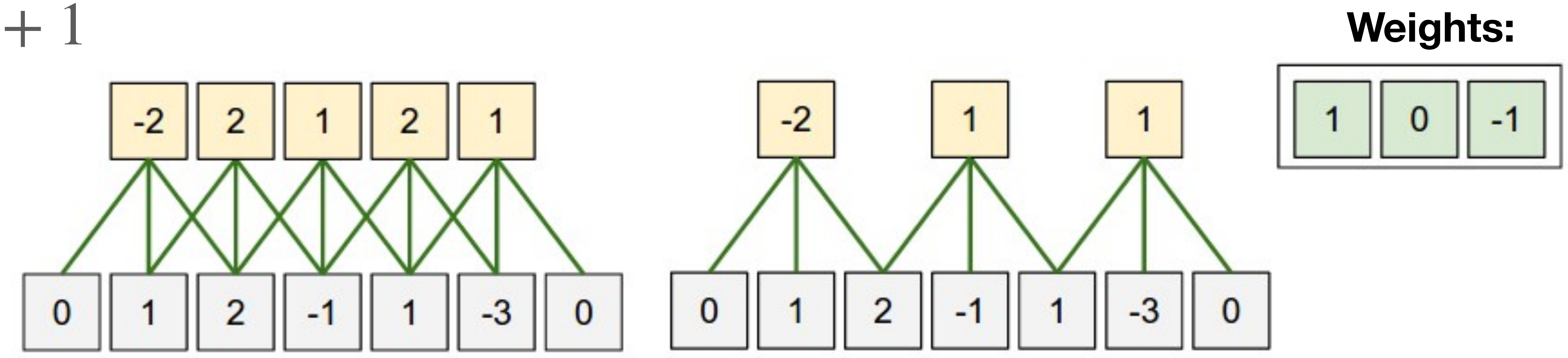
3 USEFUL TOOLS – STRIDING, PADDING, POOLING – OUTPUT SIZE

- With input of width w, and kernel with receptive field of size f, stride s and zero padding p, one output image in the convolutional layer will have width

$$w_2 = (w - f + 2p)/s + 1$$

- Strides constrained so that w_2 is an integer

- With $s = 1$, zero padding with $p = (f-1)/2$ will give $w_2 = w$



$$W^{[1]} = 7 \qquad f^{[2]} = 3 \qquad p^{[2]} = 0 \qquad s^{[2]} = 1$$

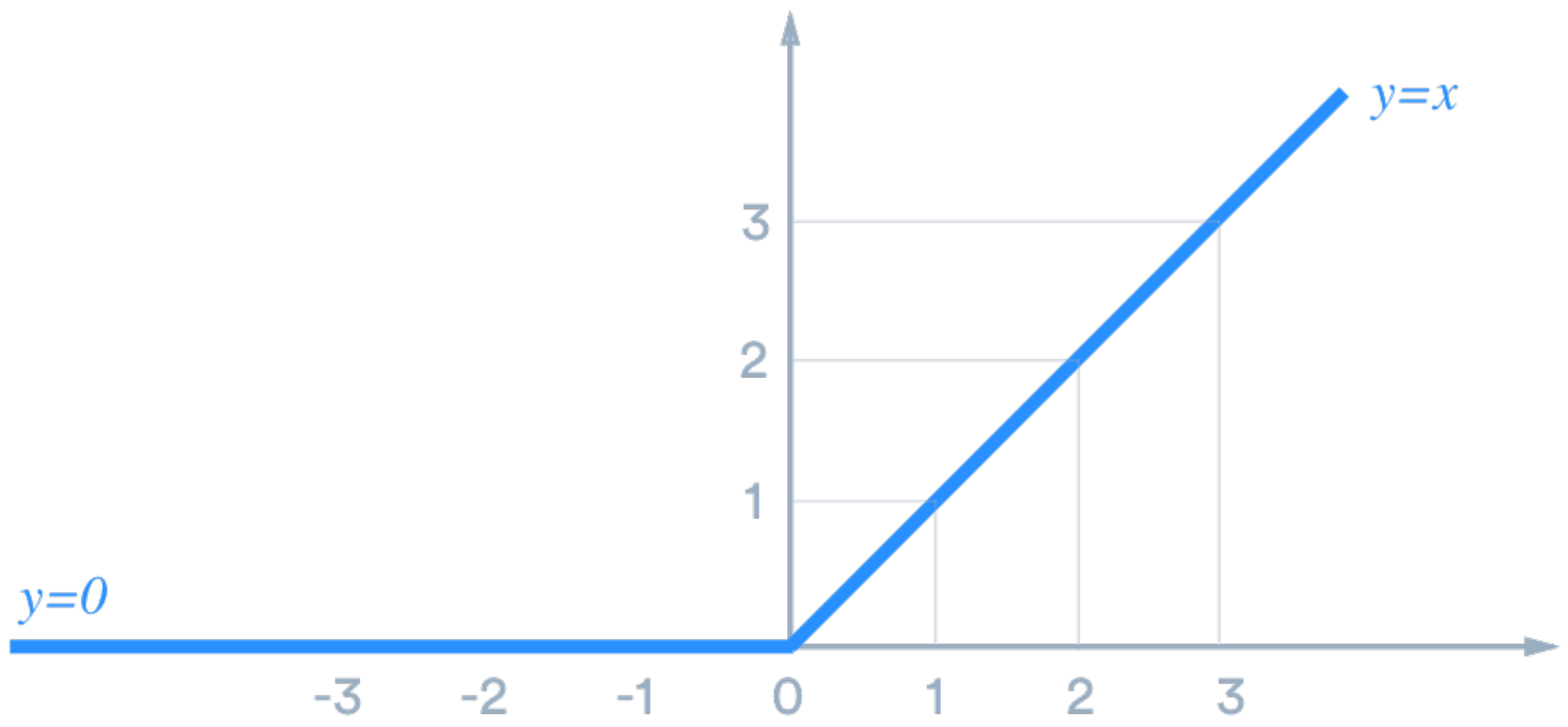
$$\Rightarrow W^{[2]} = \frac{7 - 3 + 2 \cdot 0}{1} + 1 = 5$$

$$W^{[1]} = 7 \qquad f^{[2]} = 3 \qquad p^{[2]} = 0 \qquad s^{[2]} = 2$$

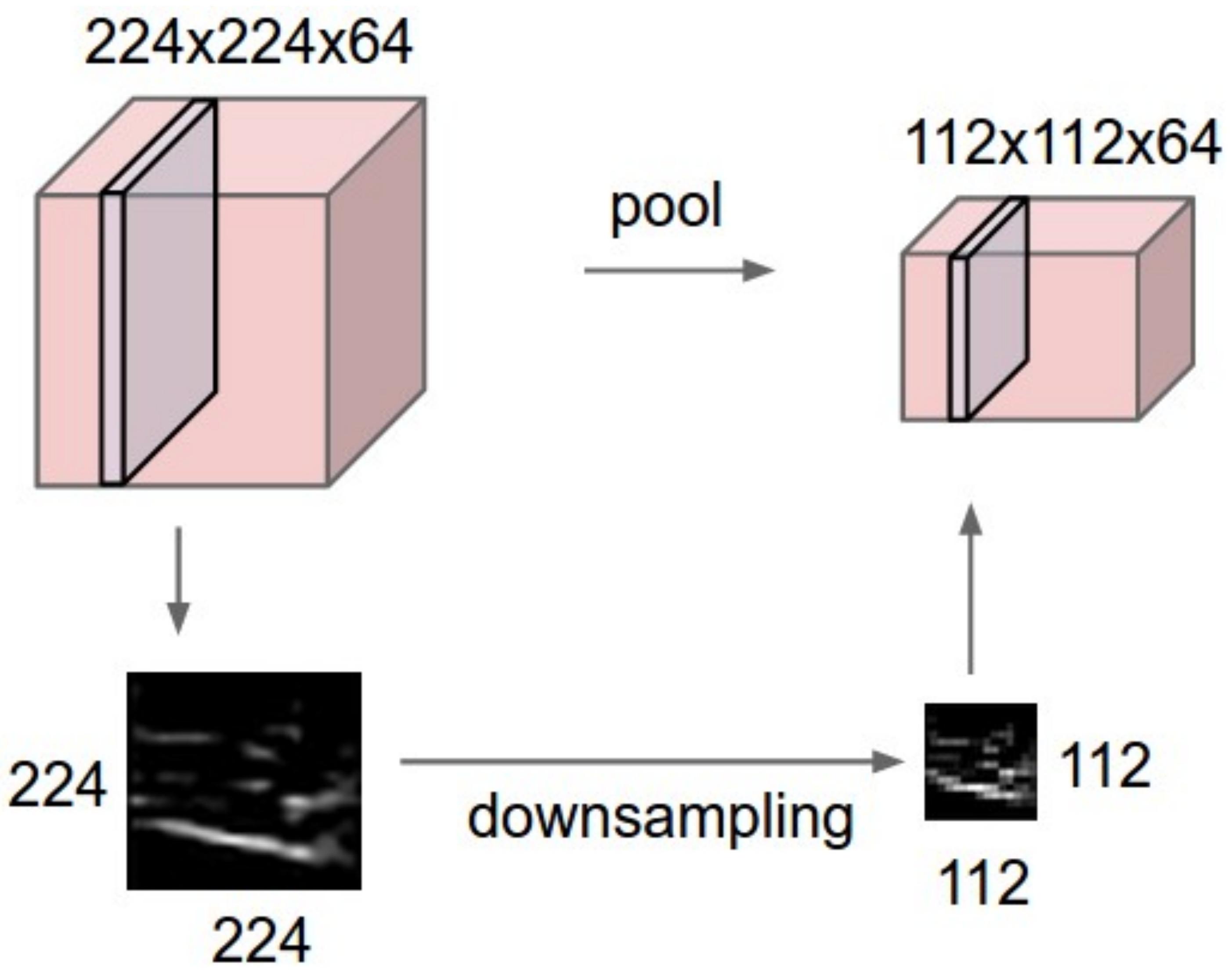
$$\Rightarrow W^{[2]} = \frac{7 - 3 + 2 \cdot 0}{2} + 1 = 3$$

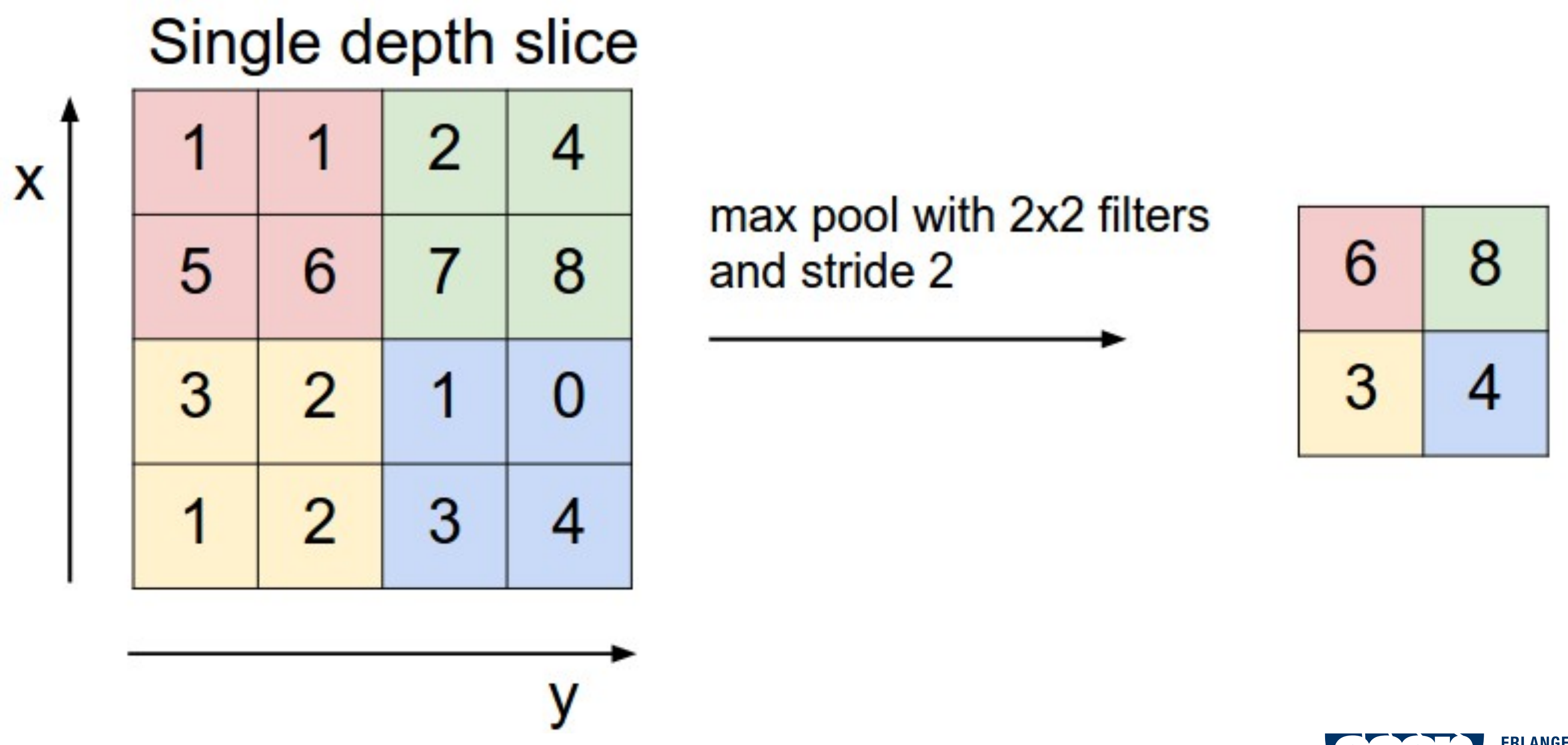
3 USEFUL TOOLS – STRIDING, PADDING, POOLING

- Result of convolution passed to non-linear activation function (e.g. Rectified Linear Unit RELU)



- Result of non-linear activation function usually passed to a 'pooling layer' that reduces spatial size (like a downsampling)
  - Most common: 2x2 filter with stride 2 that selects the maximum of the input fields (MAX pool)

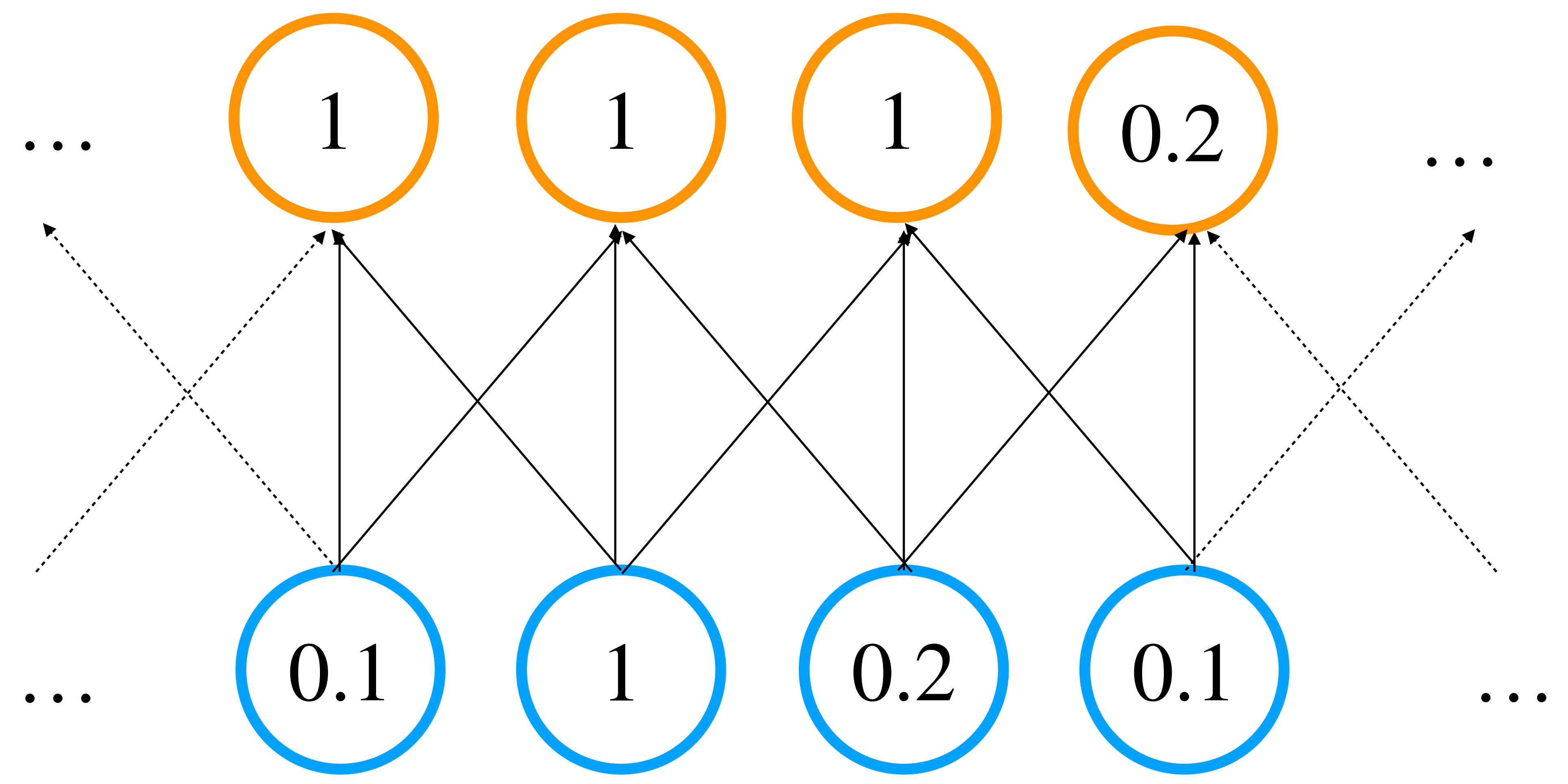




POOLING

- Helps to make representation **invariant** against small translations
- Invariance to local translations useful if we care more about a feature itself than its position in the image

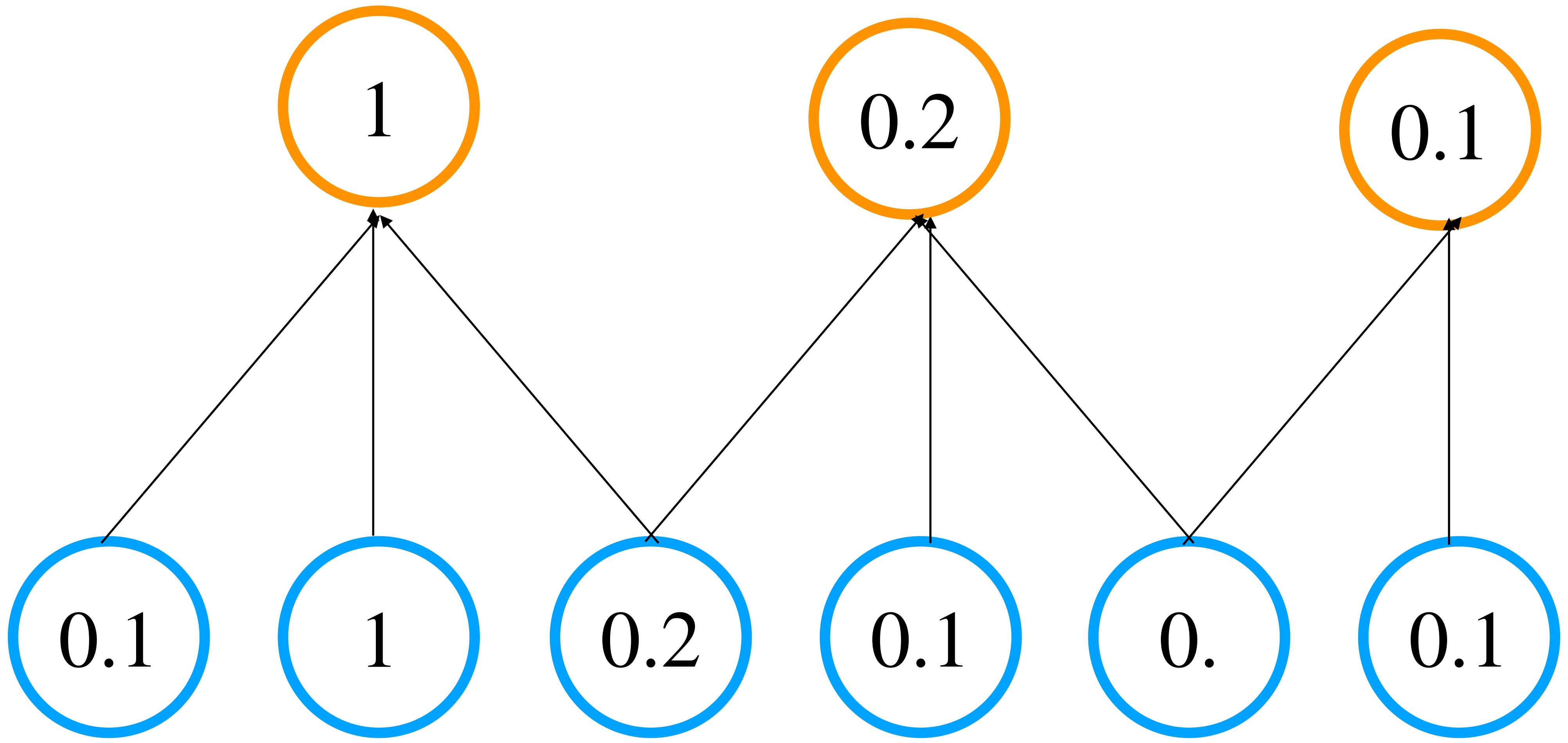
Pooling Stage (MAXPool)



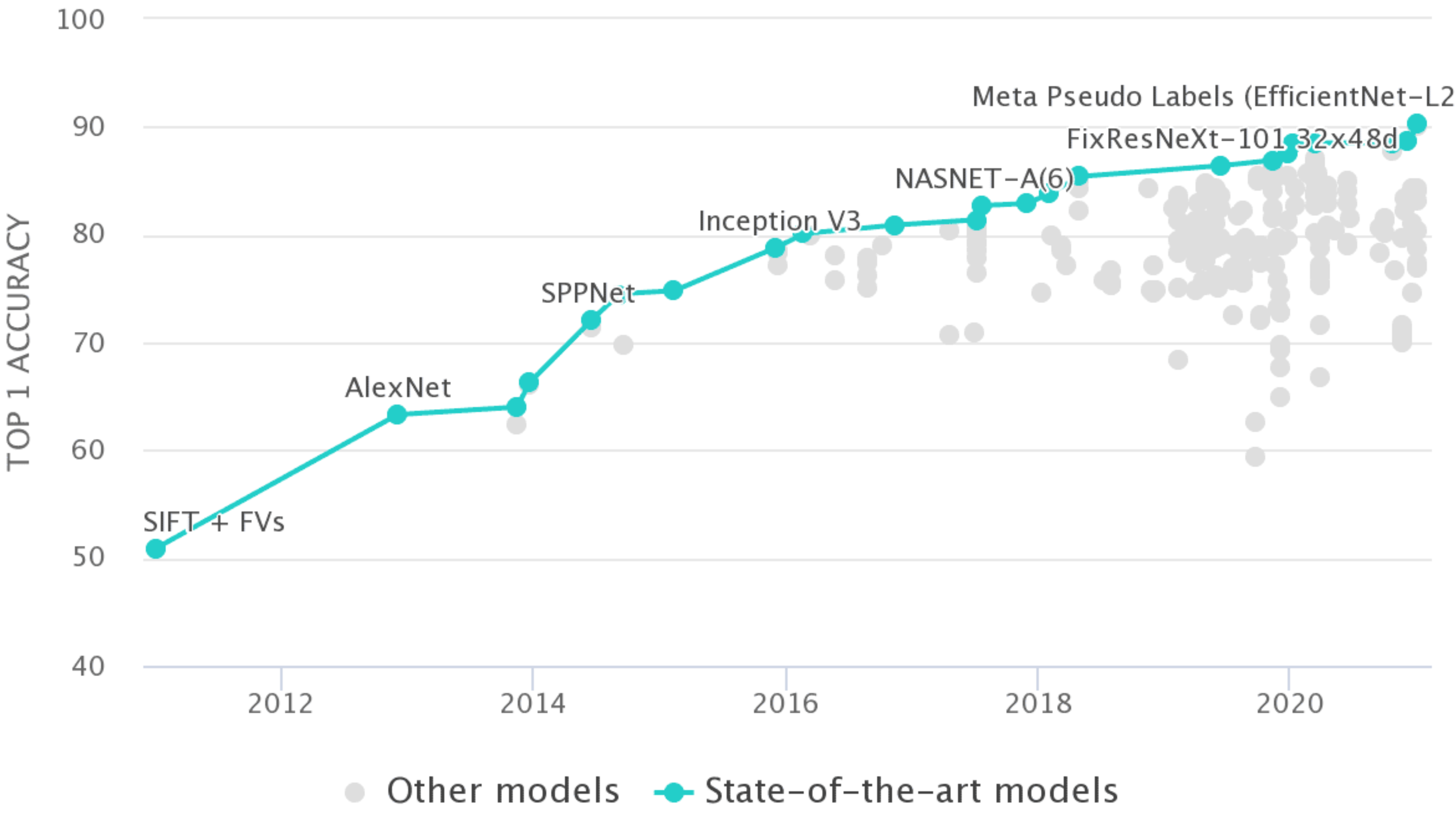
Detector Stage

POOLING WITH DOWNSAMPLING

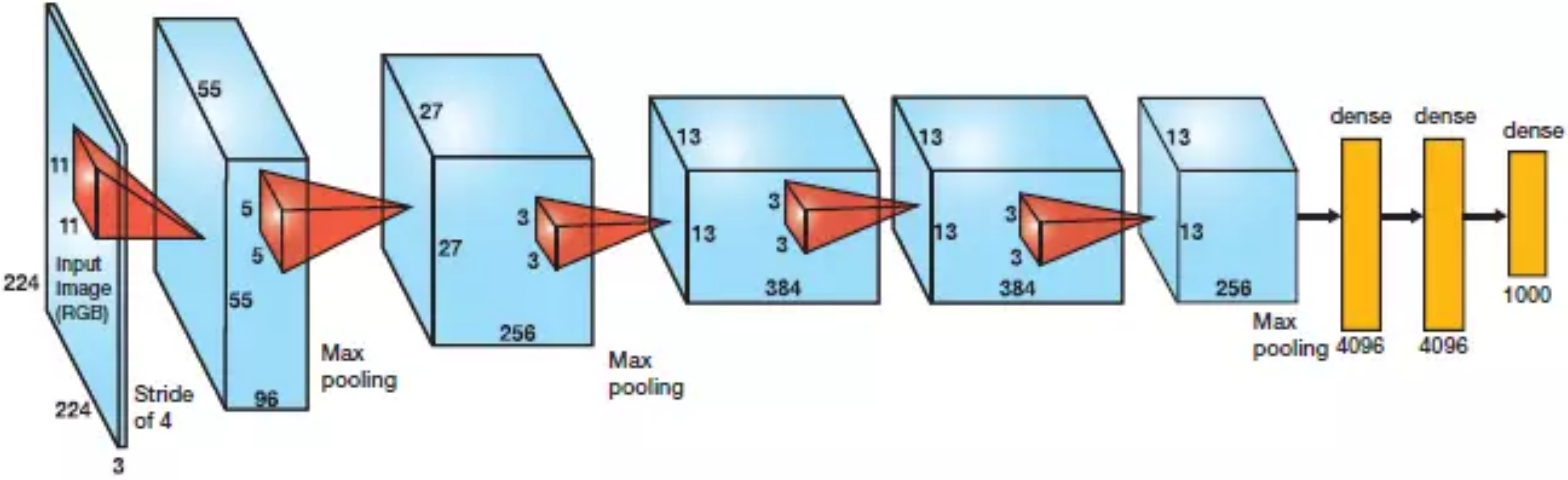
- Pooling summarizes activations of whole neighborhood
- Thus, makes sense to use fewer pooling units than detector units
- Example with stride $s = 2$:



OK, so we have the building blocks. Let’s build a CNN.


TOP 1 ACCURACY
100
90
80
70
60
50
40
2012
2014
2016
2018
2020
SIFT + FVs
AlexNet
SPPNet
Inception V3
NASNET-A(6)
FixResNeXt-101 32x48d
Meta Pseudo Labels (EfficientNet-L2
Other models
State-of-the-art models

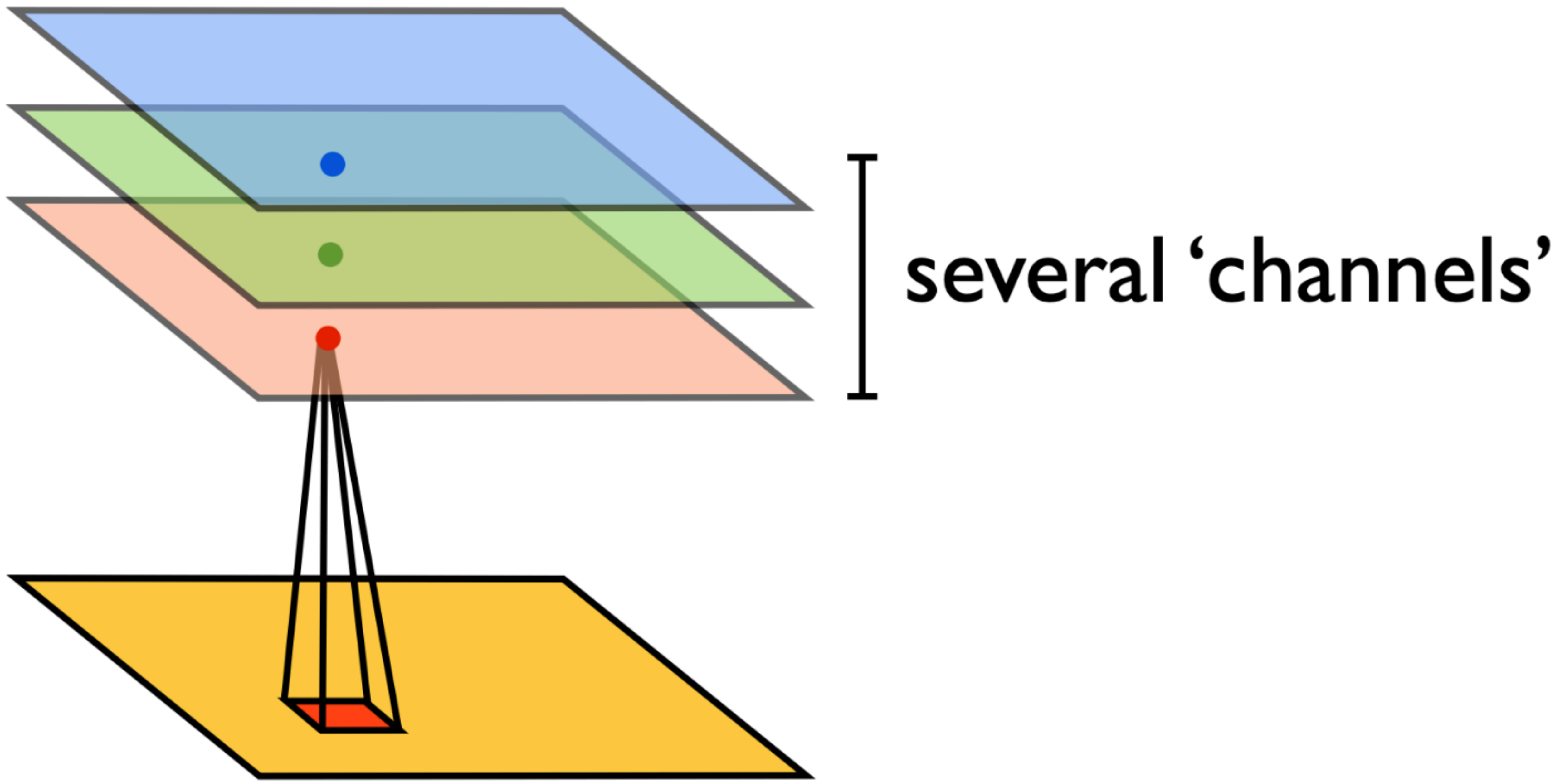
4 TYPICAL CNN ARCHITECTURES: THE FIRST SUCCESSFUL CNN – ALEX NET



https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

Several filters (kernels)

e.g. one for smoothing, one for contours, etc.



4 TYPICAL CNN ARCHITECTURES: THE FIRST SUCCESSFUL CNN – ALEX NET

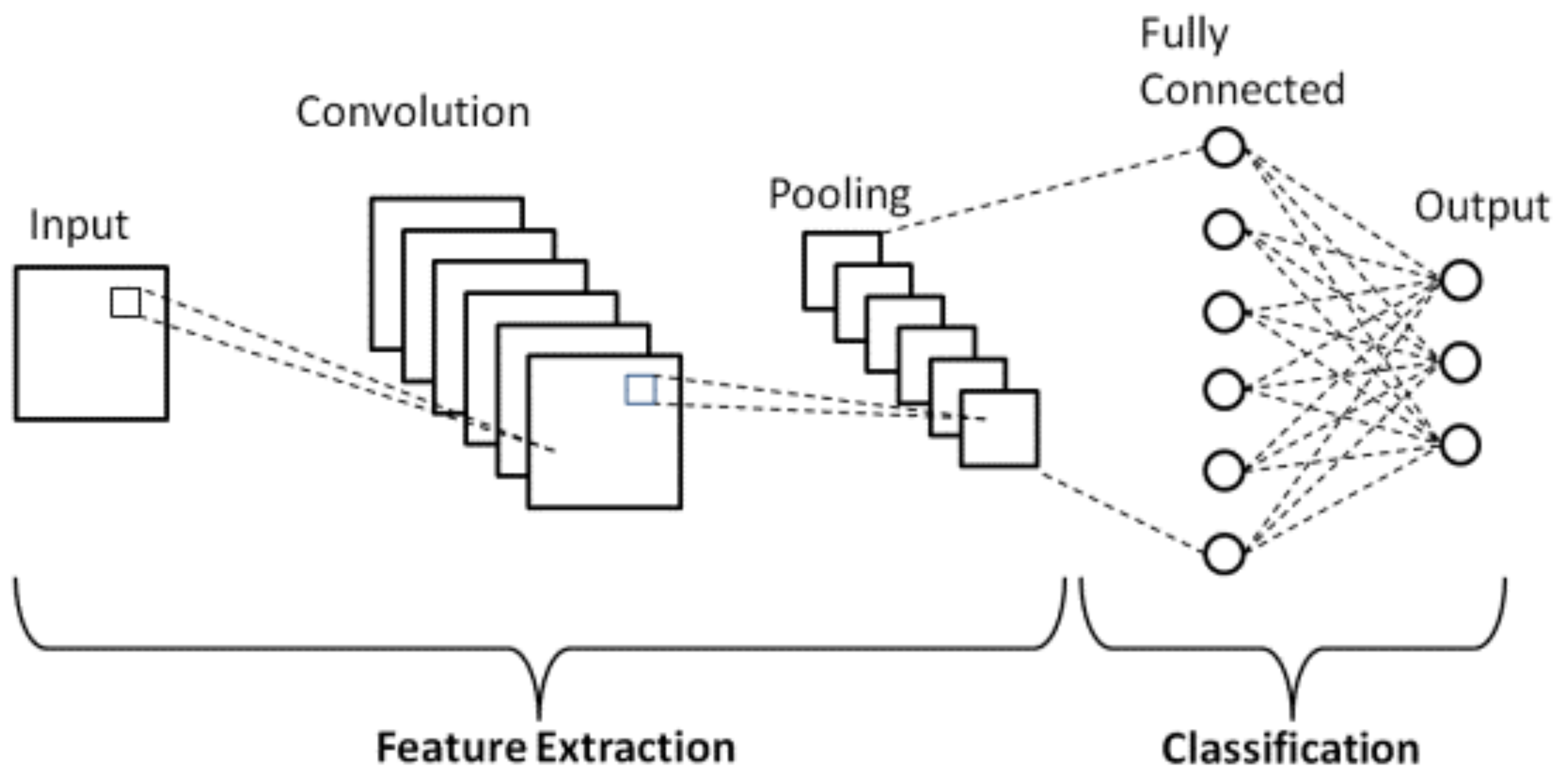
- developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. First to popularize CNNs for computer vision, won 2012 ImageNet contest. Significantly outperformed runner up
- First to implement maxpool layers, ReLU activation and dropout layers
- nowadays can be implemented in 35 lines of Torch code
- How was the exact configuration chosen? Trial and error

```python
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000) -> None:
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )

        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

4 TYPICAL CNN ARCHITECTURES



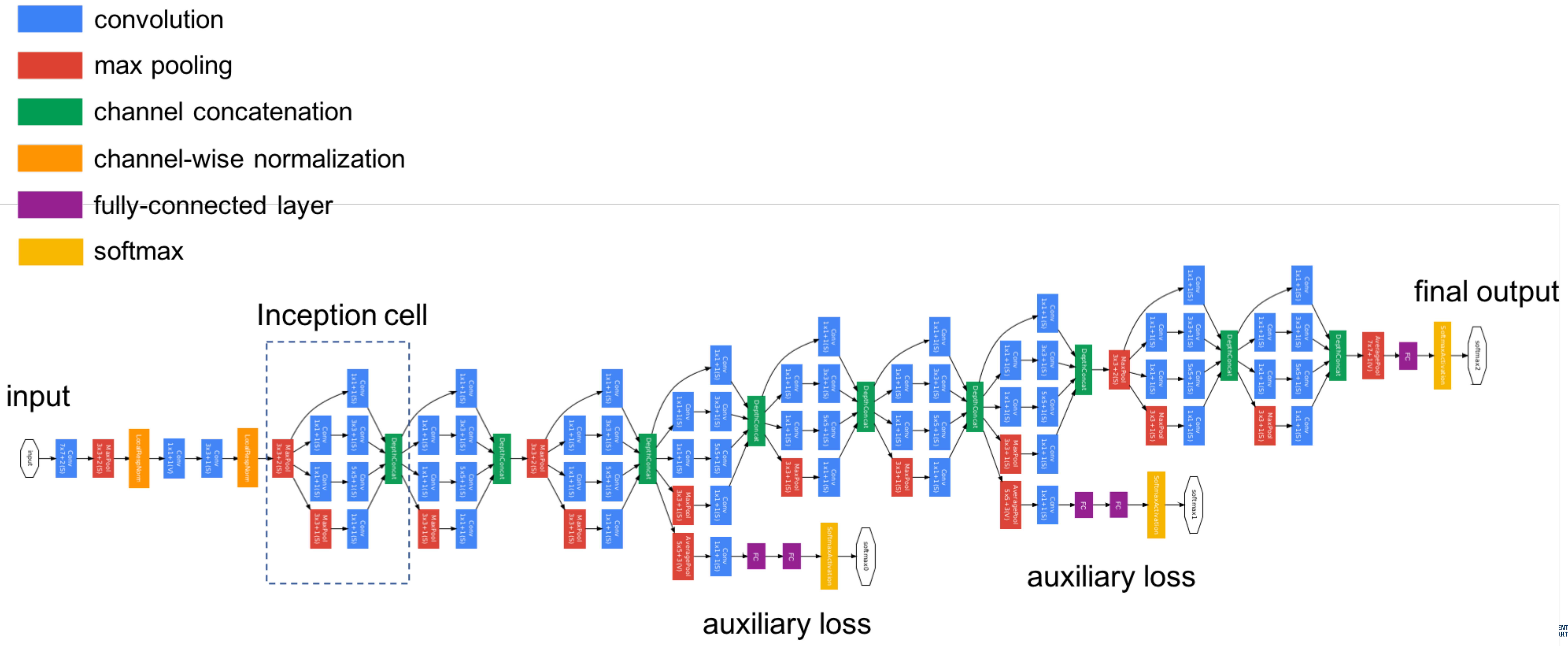
- Typical architecture of CNN generally will look like:

$$\text{Input} \to \underbrace{((\text{Conv} \to \text{ReLU}) \times N \to \text{Pooling}\,?) \times M}_{\text{Feature extraction}} \to \underbrace{(\text{FC} \to \text{ReLU}) \times K \to \text{FC}}_{\text{Classification}}$$

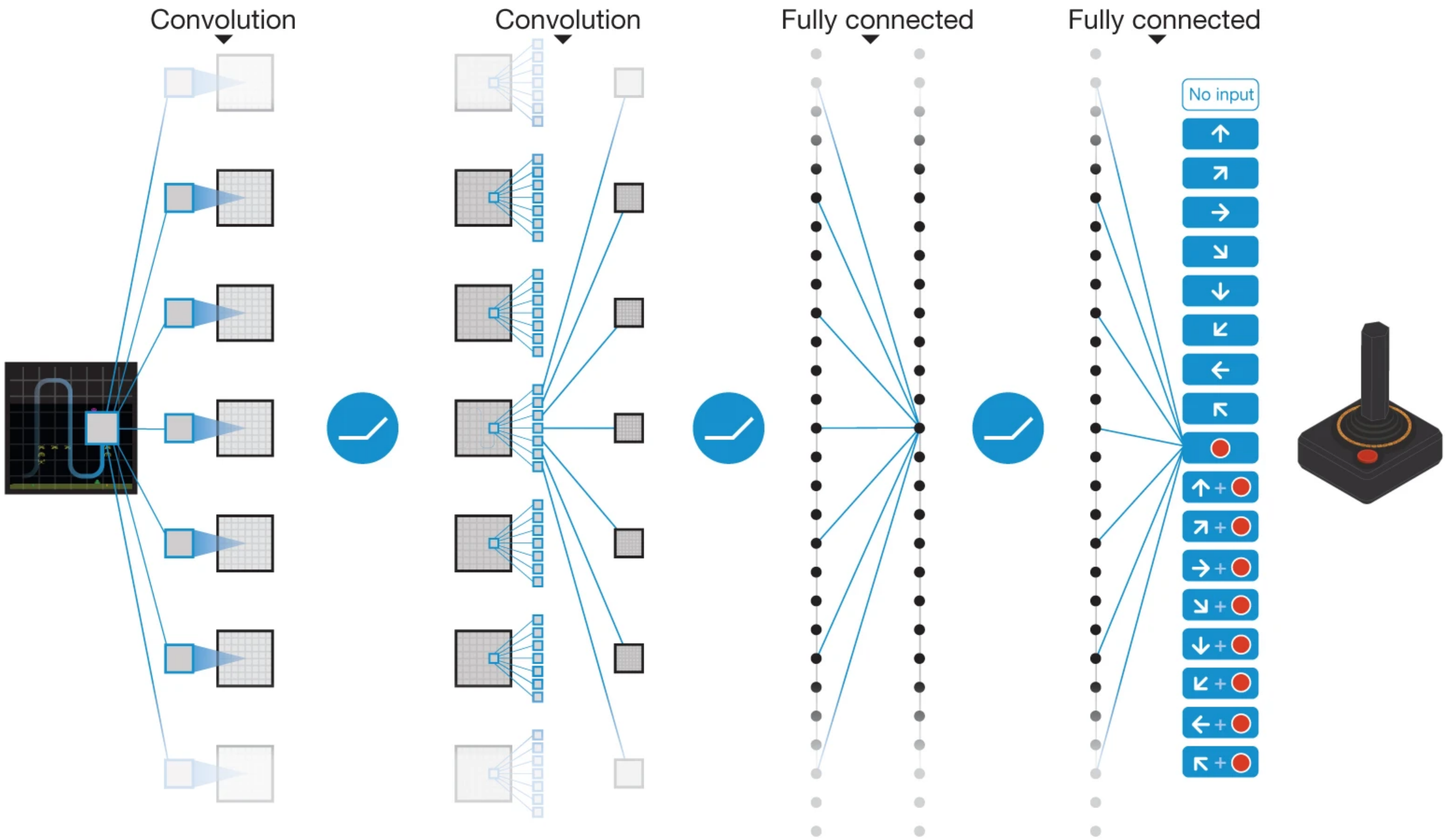
- In general: prefer repeated convolutions with small kernels over one convolution with large kernel
- Use zero padding such that convolution does not change spatial dimensions, only use pooling for that. Otherwise, you have to make sure that striding works
- Fully connected layers to classify based on the high-level features learned in the convolution part
- Consider re-using pre-trained networks and tweak them for your problem

4 TYPICAL CNN ARCHITECTURES: DEVELOPMENT FOLLOWING ALEXNET

➤ Deeper, e.g. Inception (GoogLeNet)

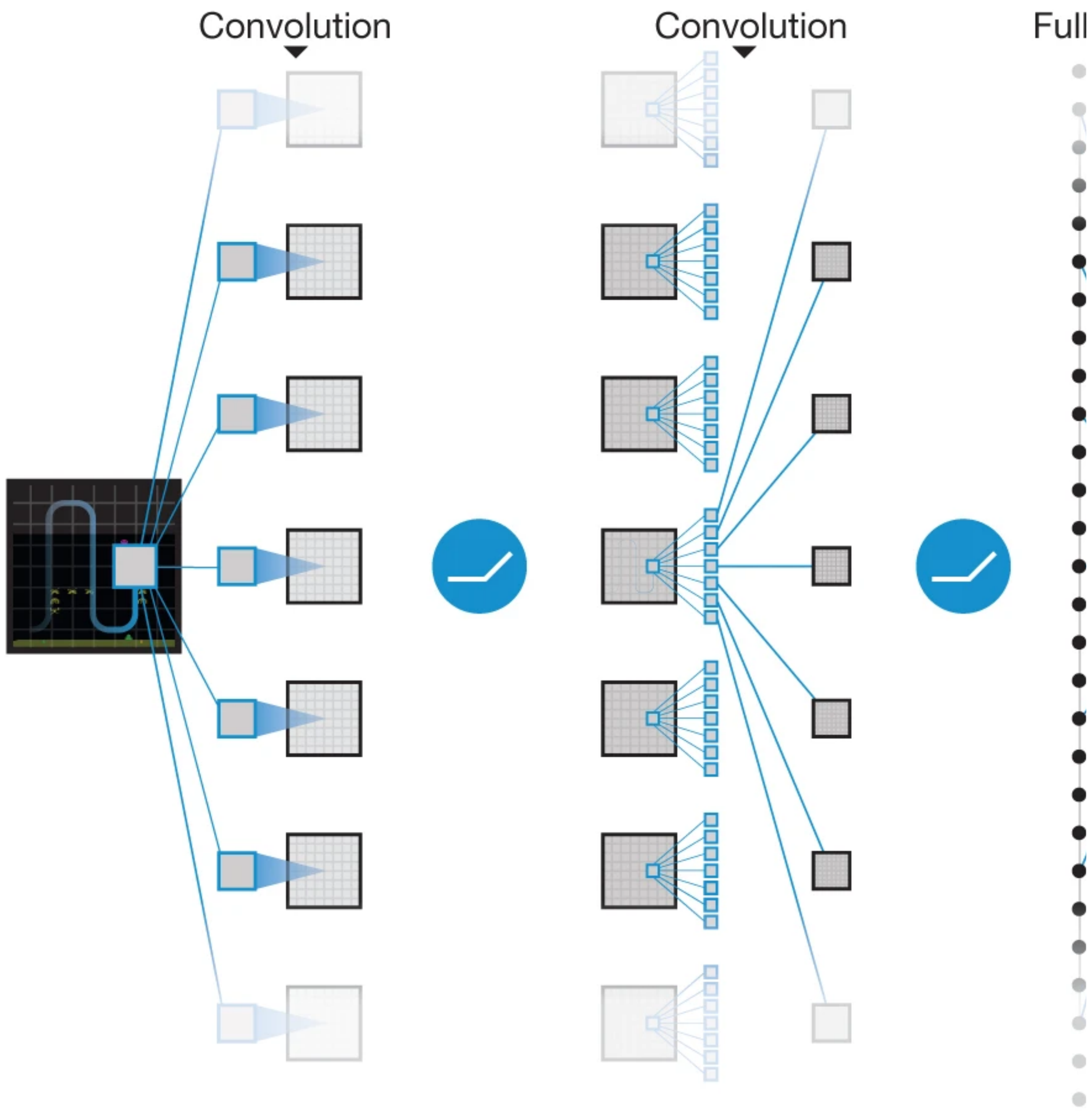


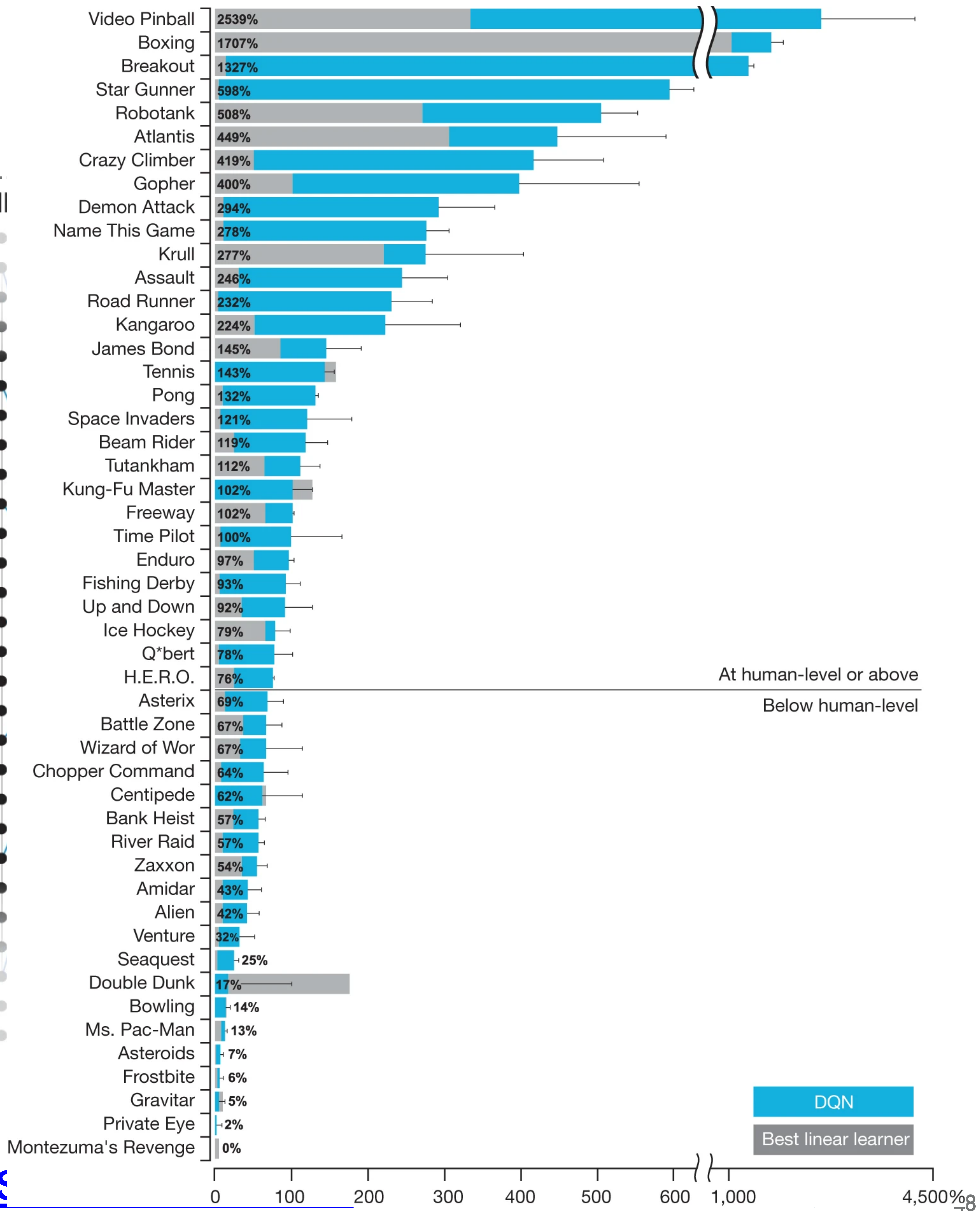
FUN EXAMPLE: PLAYING THE ATARI SUITE



https://www.nature.com/articles/nature14236

FUN EXAMPLE: PLAYING THE ATARI SUITE

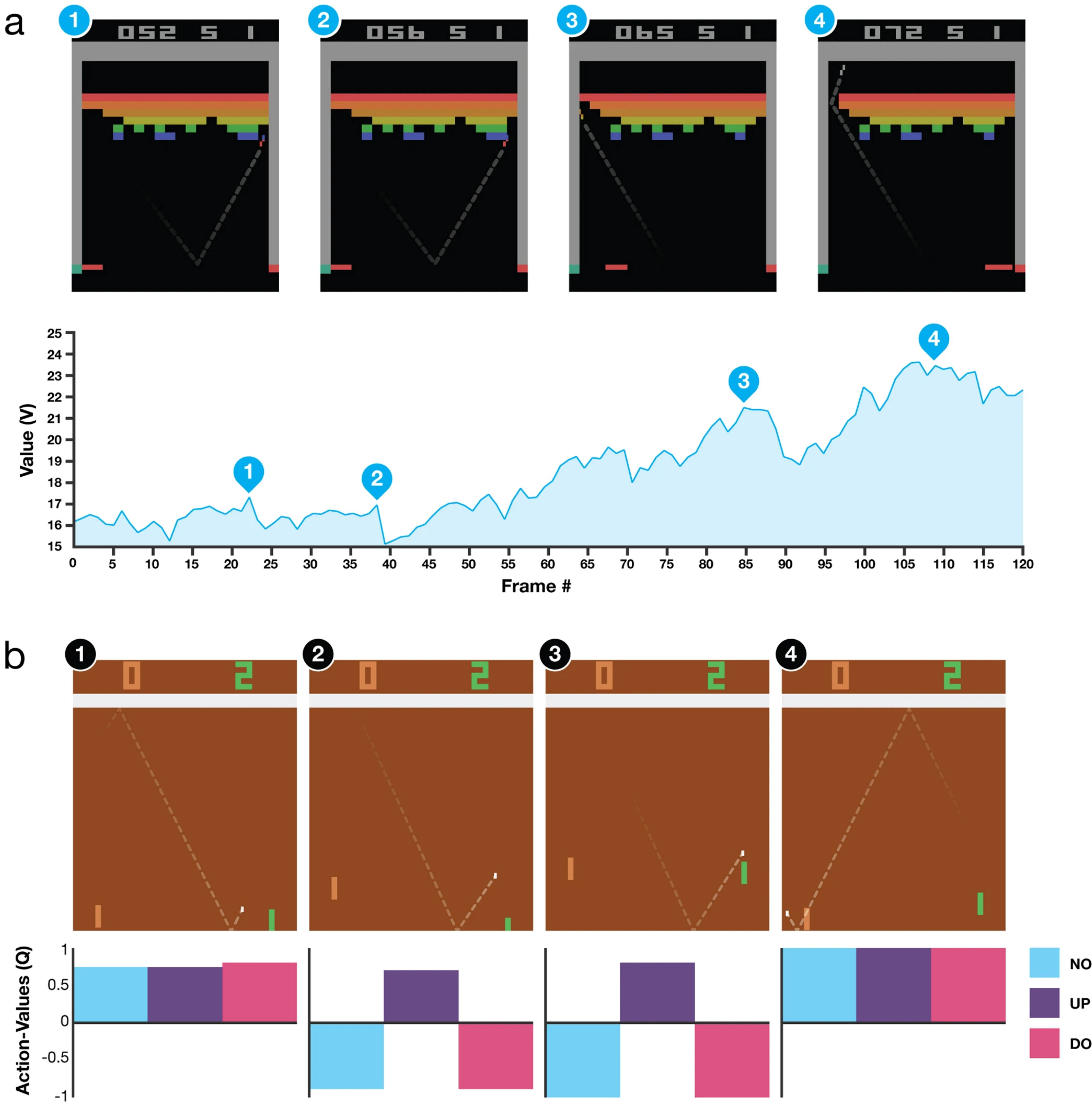




https://www.nature.com/articles

FUN EXAMPLE: PLAYING THE ATARI S

- https://static-content.springer.com/esm/art%3A10.1038%2Fnature14236/MediaObjects/41586_2015_BFnature14236_MOESM124_ESM.mov

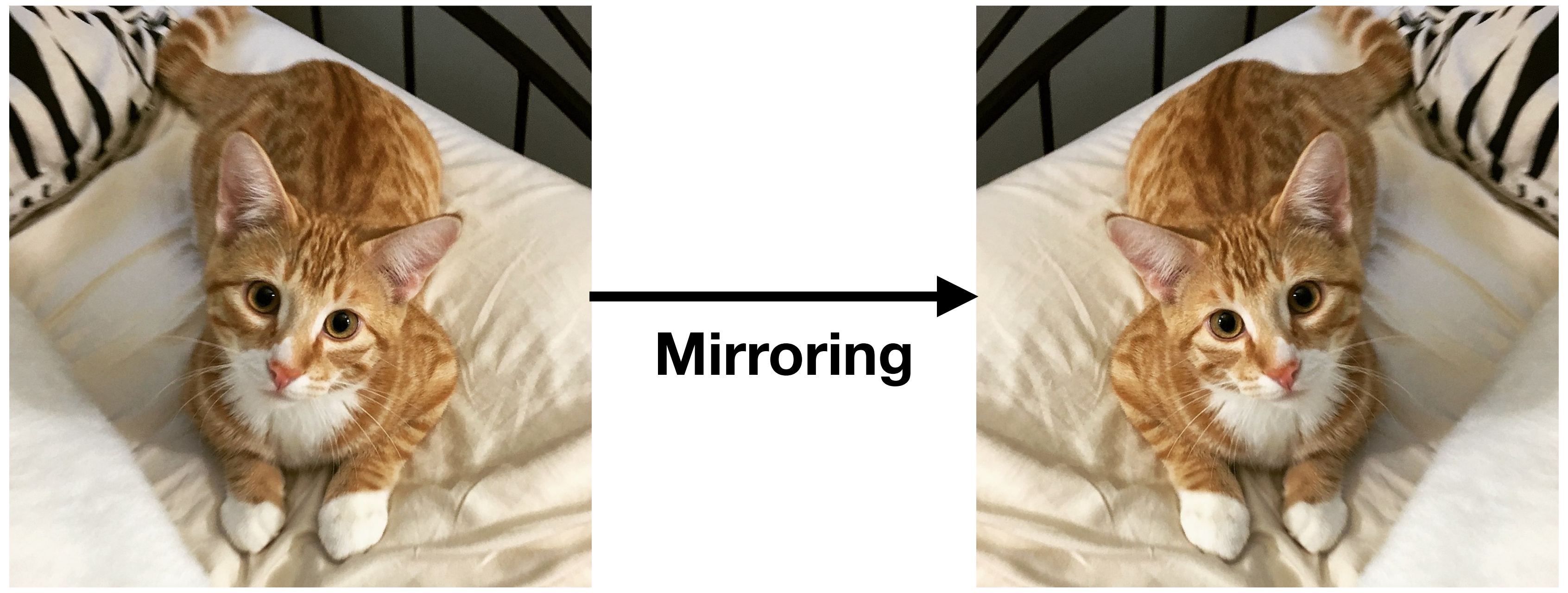


DATA AUGMENTATION

- Improve generalization error of classifier by adding copies of data samples that have been transformed in such a way that class does not change

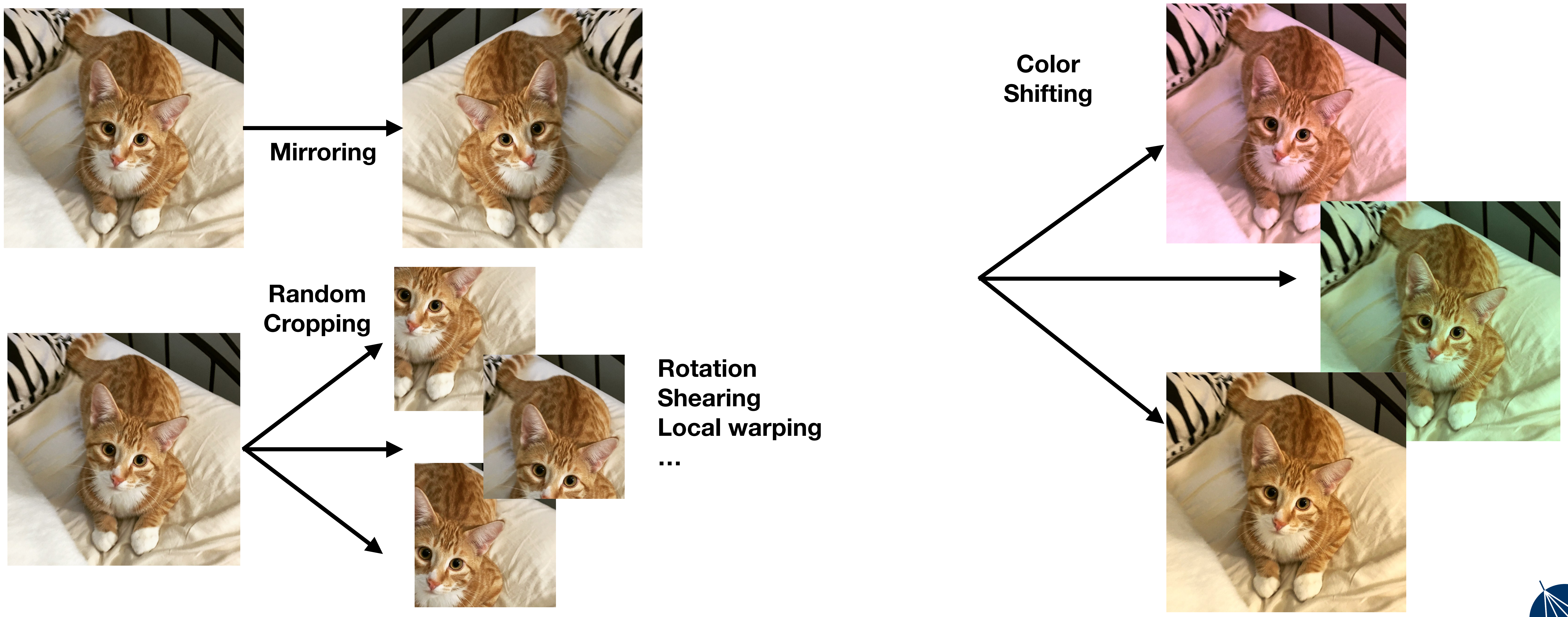
DATA AUGMENTATION

- Improve generalization error of classifier by adding copies of data samples that have been transformed in such a way that class does not change



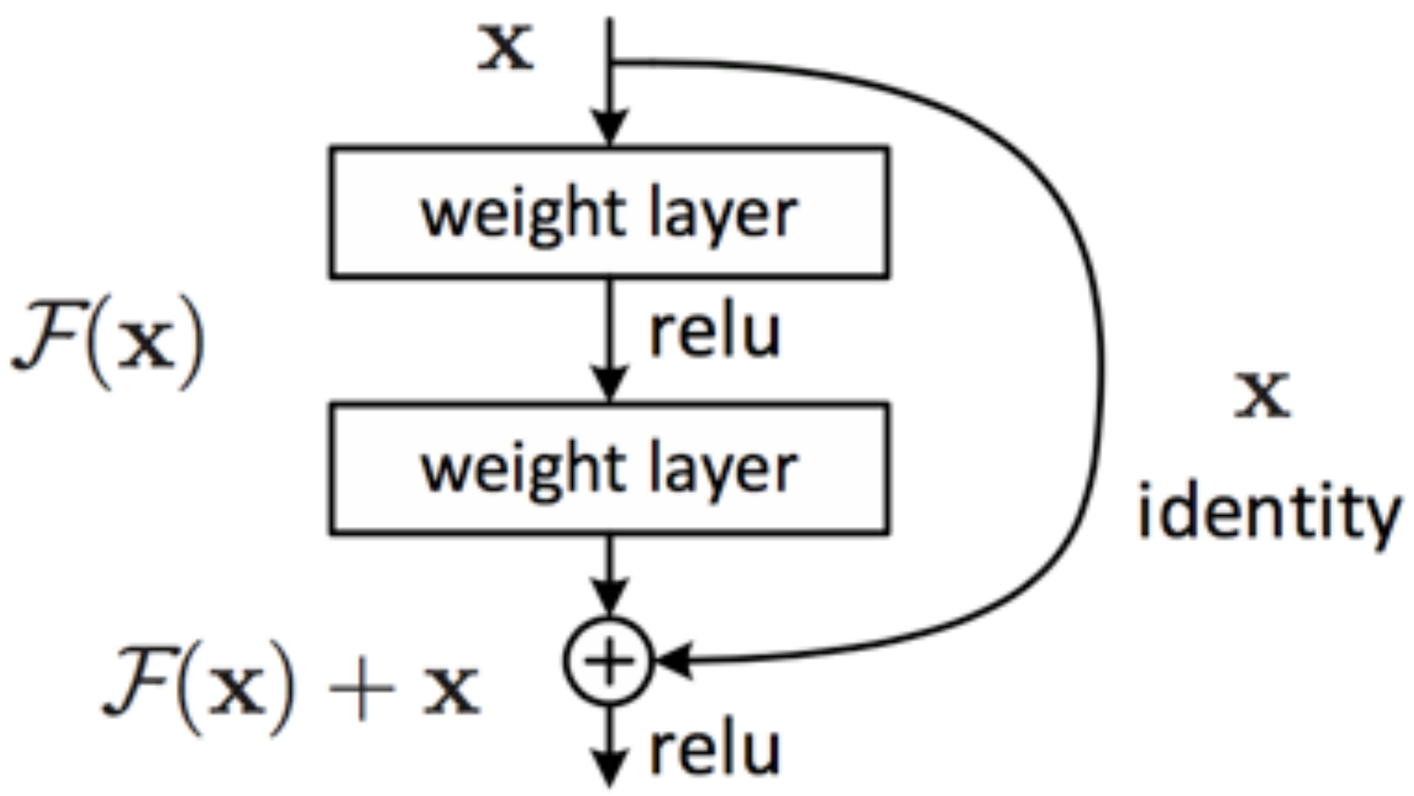
DATA AUGMENTATION

➤ Improve generalization error of classifier by adding copies of data samples that have been transformed in such a way that class does not change



4 TYPICAL CNN ARCHITECTURES: COULD GO MUCH DEEPER HERE

- Degradation problem, batch normalisation, residual blocks, ResNet
- Dense convolutional networks
- Network efficiency, …
-

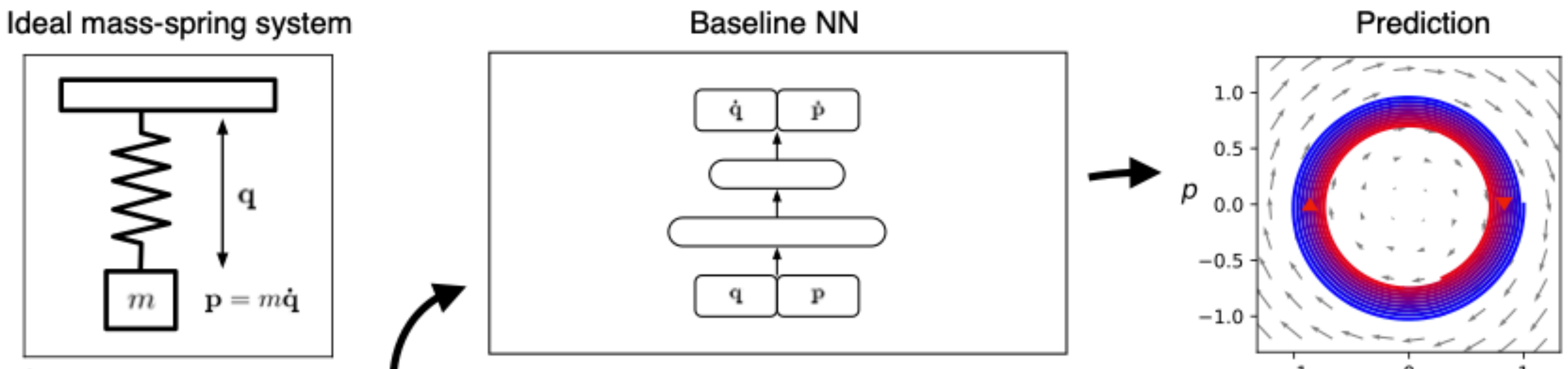


CURRENT EFFORTS TO IMPROVE CNNS

- Physics-guided CNNs (provide physics constraints on the network output) - e.g. Zhang et al. 2020 or Wu et al., 2018
  - Use physics laws as a guideline for constructing NNs (e.g. Hamiltonian NN)
  - Letting ML find optimal observables (e.g. Datta et al. 2019)
- Improve computational efficiency of CNNs, real-time processing
- CNNs on FPGAs (e.g. for L1T at LHC, HLS4ML open source, multi backend)
- Network causality - how are decisions taken? (e.g. Kindermans et al. 2017)
- Uncertainty quantification (typically with GANs)
- Refinement of MC simulations to match data (also with GANs)

CURRENT EFFORTS TO IMPROVE CNNS

- Physics-guided CNNs (provide physics constraints on the network output) - e.g. Zhang et al. 2020 or Wu et al., 2018
  - Use physics laws as a guideline for constructing NNs (e.g. Hamiltonian NN)
  - Letting ML find optimal observables (e.g. Datta et al. 2019)
- Improve computational efficiency of CNNs, real-time processing
- CNNs on FPGAs (e.g. for L1T at LHC, HLS4ML open source, multi backend)
- Network causality - how are decisions taken? (e.g. Kindermans et al. 2017)
- Uncertainty quantification (typically with GANs)
- Refinement of MC simul

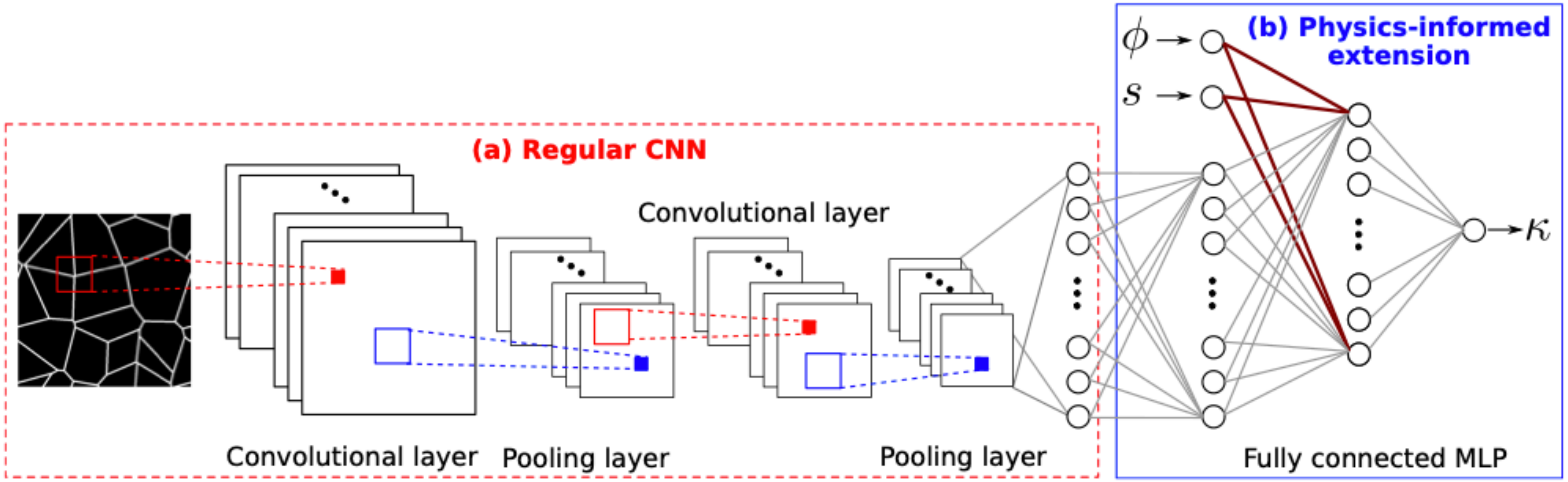


CURRENT EFFORTS TO IMPROVE CNNS

- Physics-guided CNNs (provide physics constraints on the network output) - e.g. Zhang et al. 2020 or Wu et al., 2018
  - Use physics laws as a guideline for constructing NNs (e.g. Hamiltonian NN)
  - Letting ML find optimal observables (e.g. Datta et al. 2019)
- Improve computational efficiency of CNNs, real-time processing
- CNNs on FPGAs (e.g. for L1T at LHC, HLS4ML open source, multi backend)
- Network causality - how are decisions taken? (e.g. Kindermans et al. 2017)
- Uncertainty quantification (typically with GANs)
- Refinement of MC simulations to match data (also with GANs)

CURRENT EFFORTS TO IMPROVE CNNS

- Physics-guided CNNs
  2020 or Wu et al., 201
  - Use physics laws as
  - Letting ML find opti



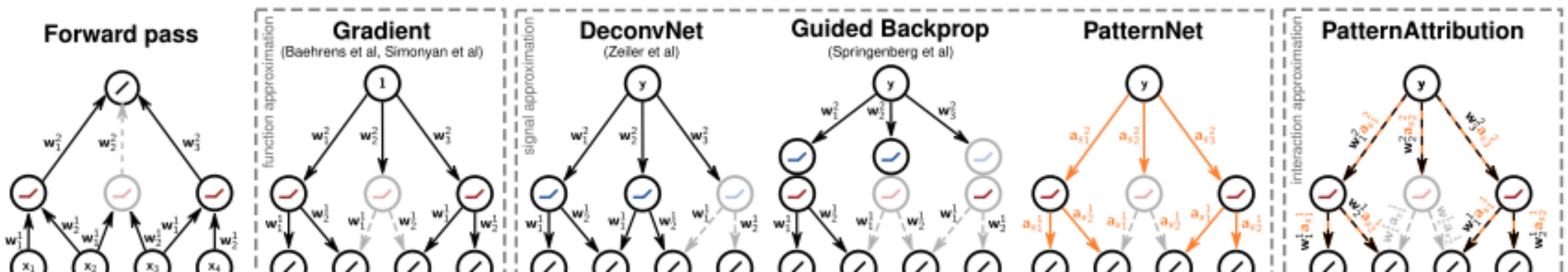
- Improve computationa
- CNNs on FPGAs (e.g. for L1T at LHC, HLS4ML open source, multi backend)
- Network causality - how are decisions taken? (e.g. Kindermans et al. 2017)
- Uncertainty quantification (typically with GANs)
- Refinement of MC simulations to match data (also with GANs)

CURRENT EFFORTS TO IMPROVE CNNS

- Physics-guided CNNs (provide physics constraints on the network output) - e.g. Zhang et al. 2020 or Wu et al., 2018
  - Use physics laws as a guideline for constructing NNs (e.g. Hamiltonian NN)
  - Letting ML find optimal observables (e.g. Datta et al. 2019)
- Improve computational efficiency of CNNs, real-time processing
- CNNs on FPGAs (e.g. for L1T at LHC, HLS4ML open source, multi backend)
- Network causality - how are decisions taken? (e.g. Kindermans et al. 2017)
- Uncertainty quantification (typically with GANs)
- Refinement of MC simulations to match data (also with GANs)

CURRENT EFFORTS TO IMPROVE CNNS

- Physics-guided CNNs (provide physics constraints on the network output) - e.g. Zhang et al. 2020 or Wu et al., 2018
  - Use physics laws as a guideline for constructing NNs (e.g. Hamiltonian NN)
  - Letting ML find optimal observables (e.g. Datta et al. 2019)
- Improve computational efficiency of CNNs, real-time processing
- CNNs on FPGAs (e.g. for L1T at LHC, HLS4ML open source, multi backend)
- Network causality - how are decisions taken? (e.g. Kindermans et al. 2017)
- Uncertainty quantification (typically with GANs)
- Refinement of MC simula



CURRENT EFFORTS TO IMPROVE CNNS

- Physics-guided CNNs (provide physics constraints on the network output) - e.g. Zhang et al. 2020 or Wu et al., 2018
  - Use physics laws as a guideline for constructing NNs (e.g. Hamiltonian NN)
  - Letting ML find optimal observables (e.g. Datta et al. 2019)
- Improve computational efficiency of CNNs, real-time processing
- CNNs on FPGAs (e.g. for L1T at LHC, HLS4ML open source, multi backend)
- Network causality - how are decisions taken? (e.g. Kindermans et al. 2017)
- Uncertainty quantification (typically with GANs)
- Refinement of MC simulations to match data (also with GANs)

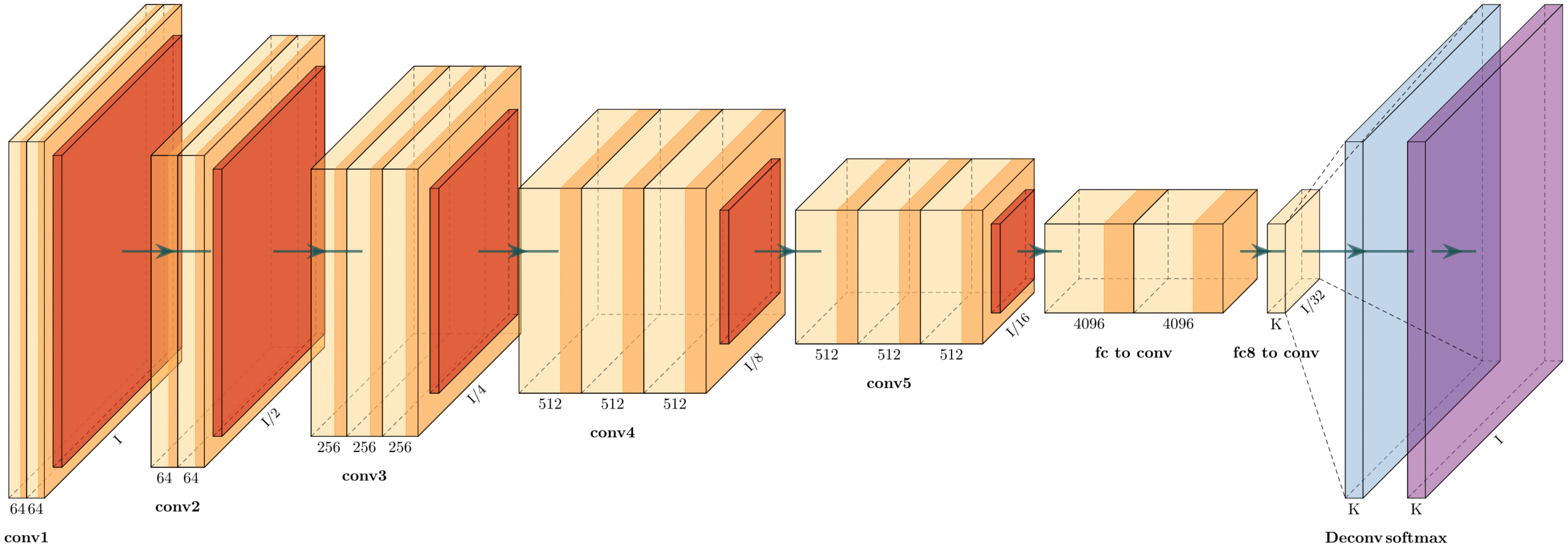
NOW LET'S APPLY IT – TENSORFLOW, OR KERAS

- Convenient neural network package for python
- Set up and training of a network in a few lines
- Based on underlying neural network package (also provides run-time compilation to CPU and GPU) either *tensorflow* or *theano*.

SUMMARY

➤ Machines exploit physics contained in data deeper than before

➤ CNNs are the workhorse for many of the more advanced applications, as example in investigations of causality, stability, uncertainties



Friedrich-Alexander-Universität
Faculty of Sciences

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS