MATH60026/MATH70026
Methods for Data Science
Lecture 6

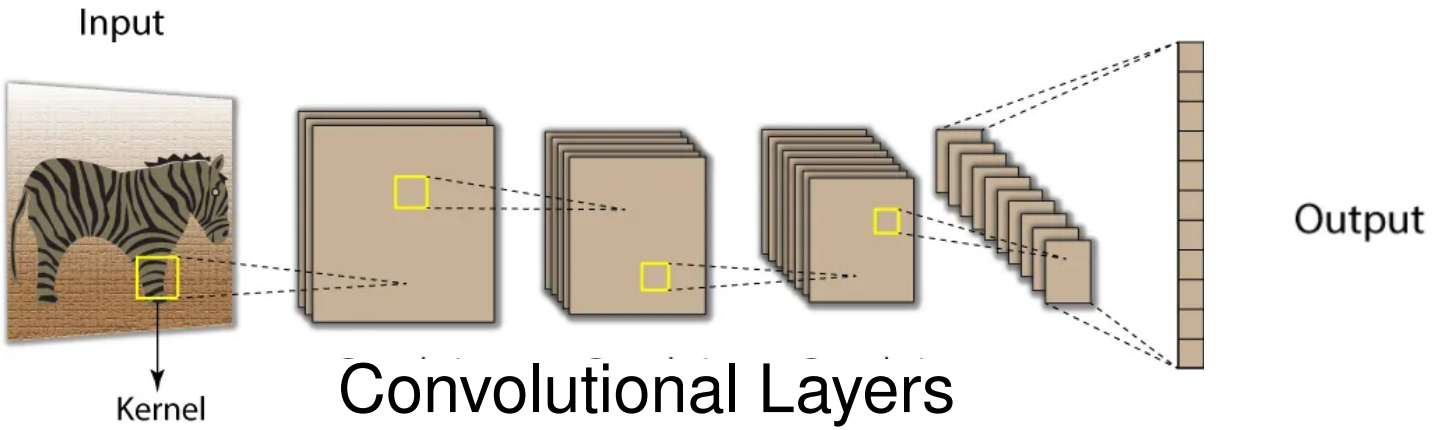
based on materials by Kevin Webster

Barbara Bravi, Imperial College London

Department of Mathematics, Academic year 2024-2025

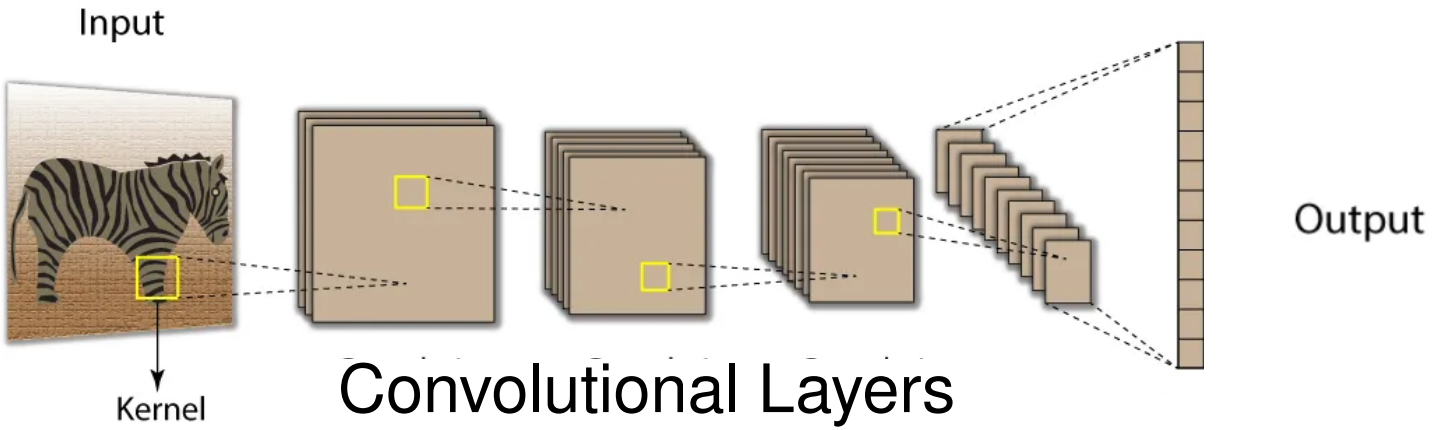
IMPERIAL

Convolutional NNs



"Success story" in deep learning: CNN have revolutionised tasks of **image recognition and classification.**

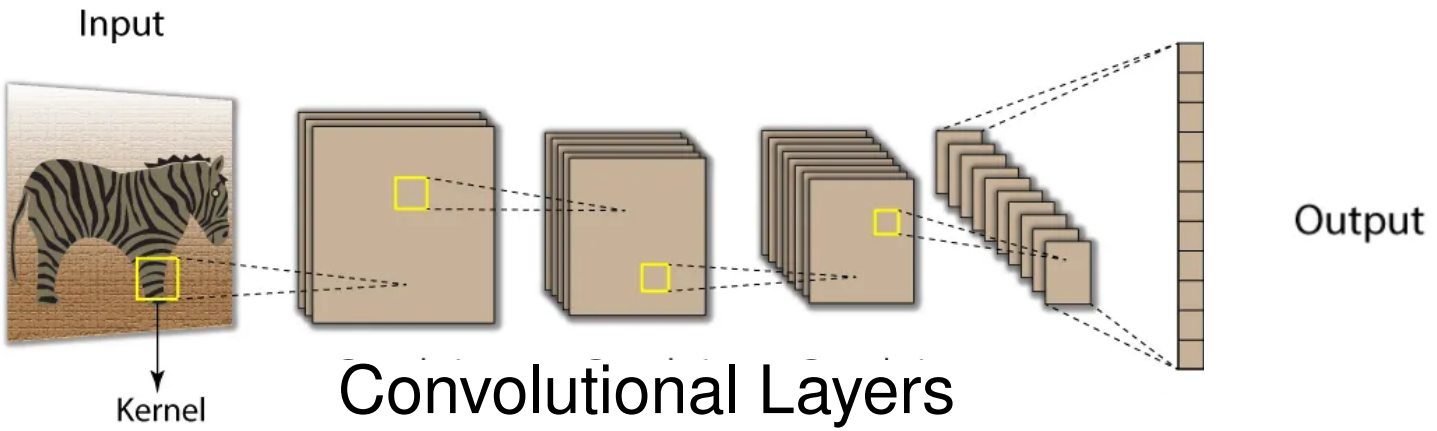
Convolutional NNs



"Success story" in deep learning: CNN have revolutionised tasks of **image recognition and classification.**

1. They leverage specific **"structural priors"**

Convolutional NNs



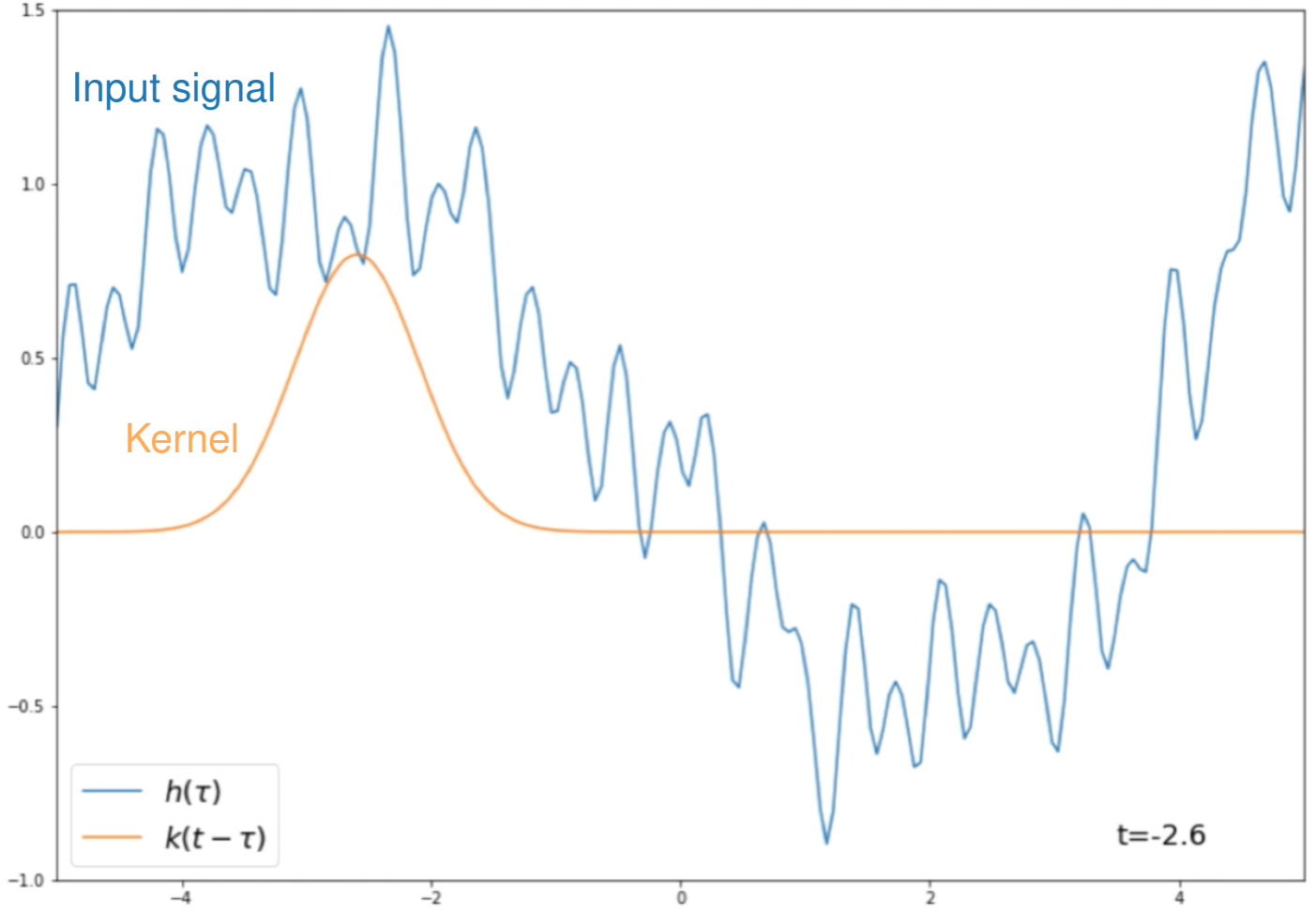
"Success story" in deep learning: CNN have revolutionised tasks of **image recognition and classification.**

1. They leverage specific **"structural priors"**

2. Convolutional layers implement **convolution operations**

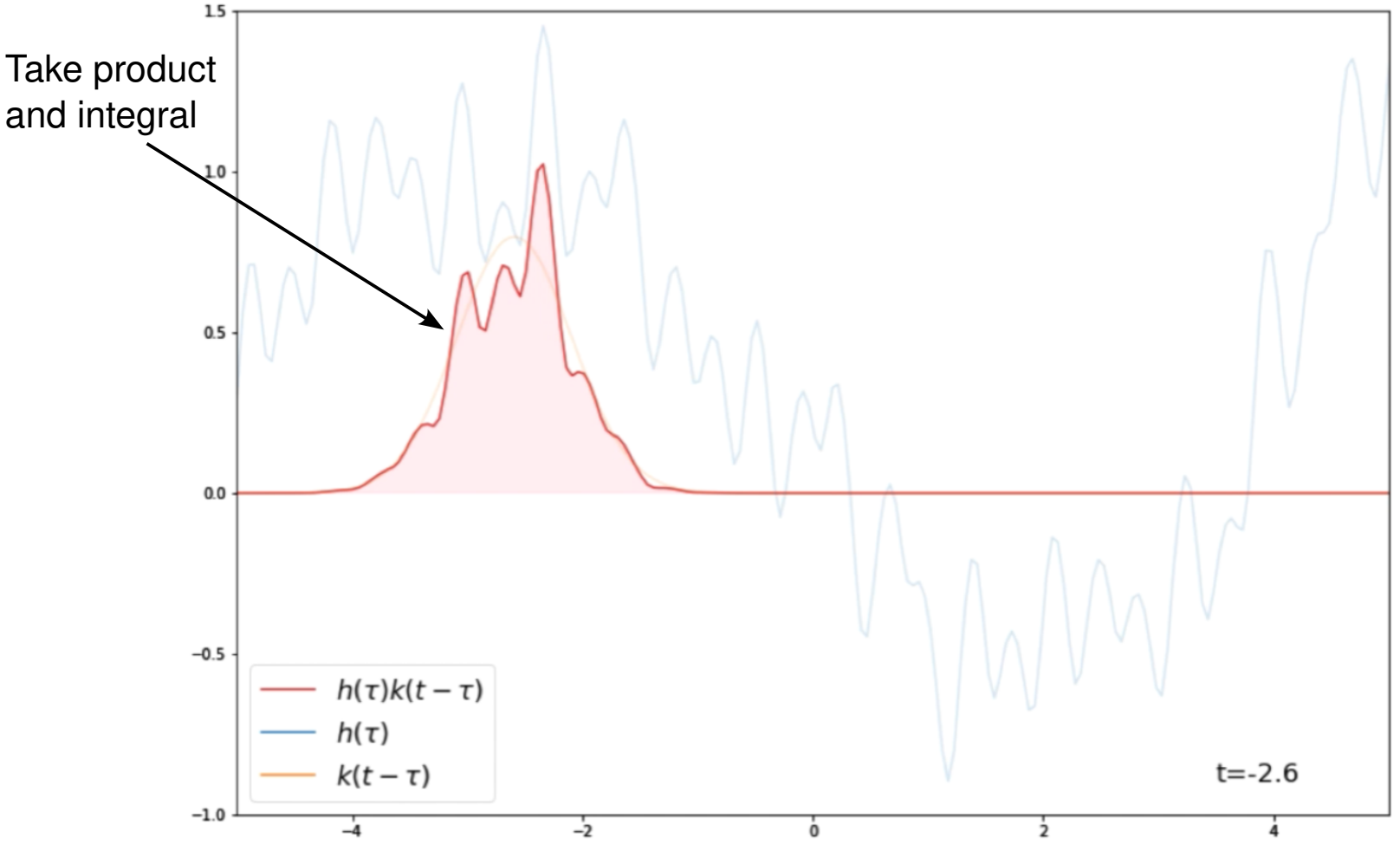
The convolution operation

$$(h * k)(t) = \int_{-\infty}^{\infty} h(\tau)k(t-\tau)d\tau$$



The convolution operation

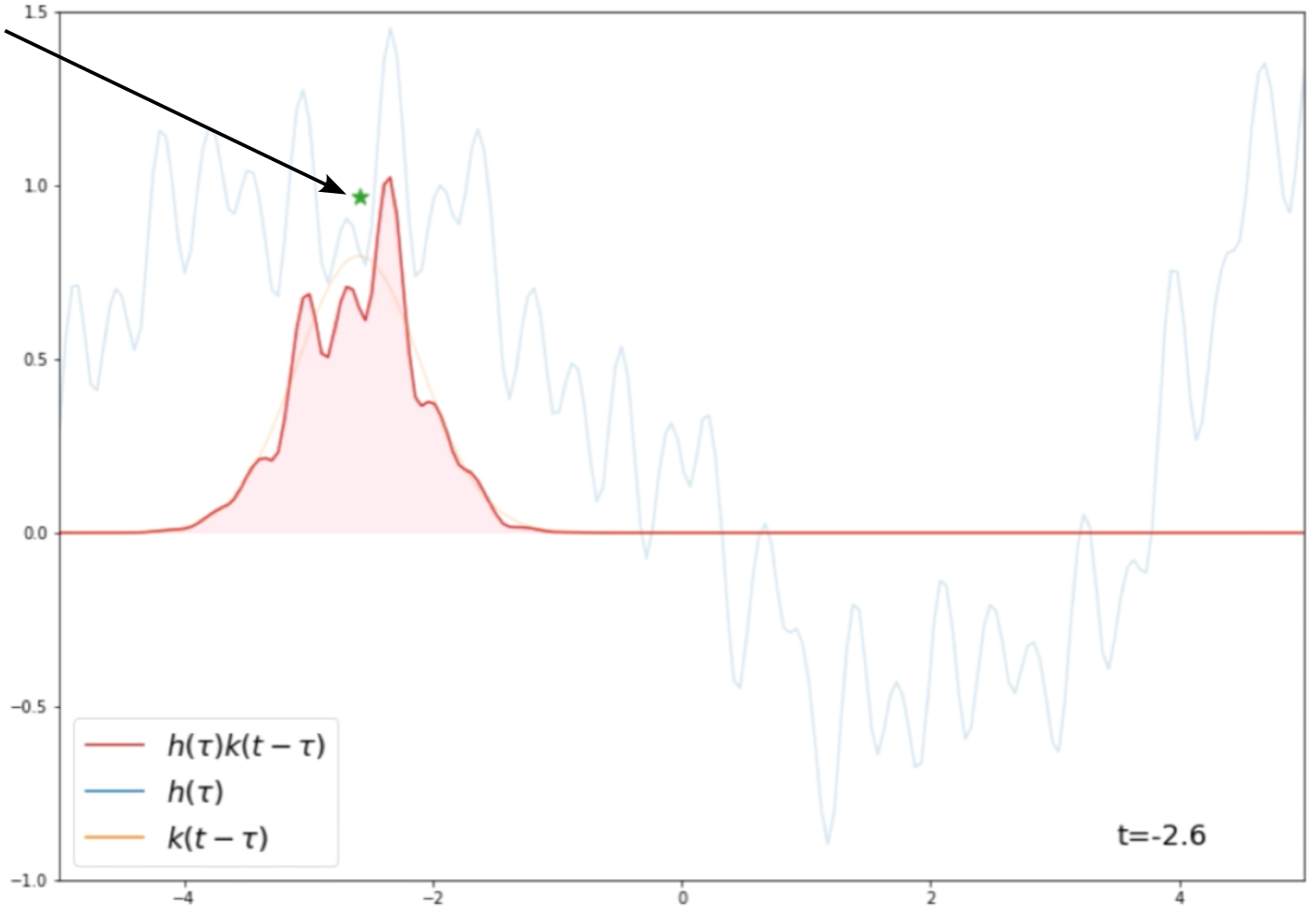
$$(h * k)(t) = \int_{-\infty}^{\infty} h(\tau)k(t-\tau)d\tau$$



The convolution operation

$$(h * k)(t) = \int_{-\infty}^{\infty} h(\tau)k(t-\tau)d\tau$$

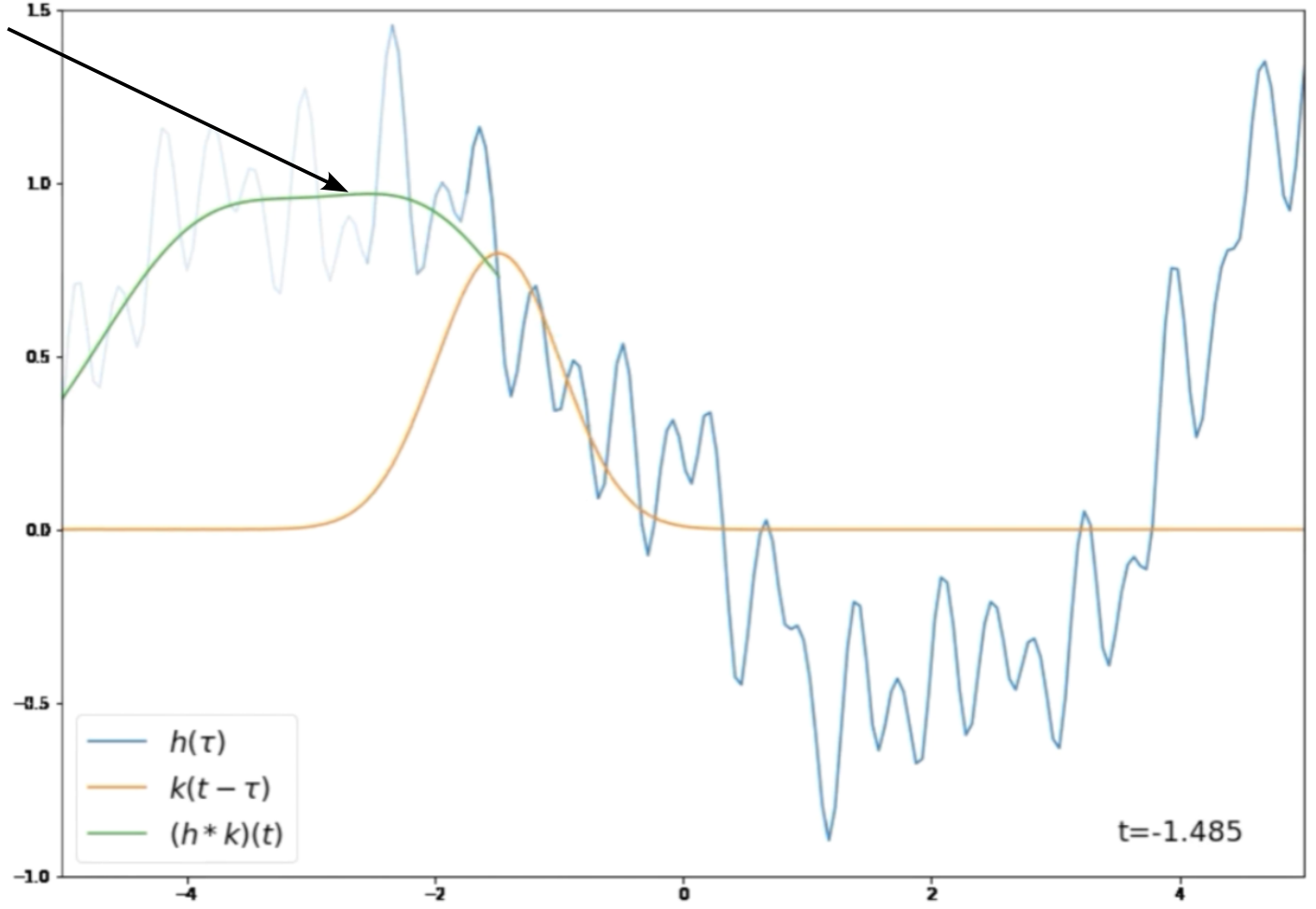
Convolution value at *t*



The convolution operation

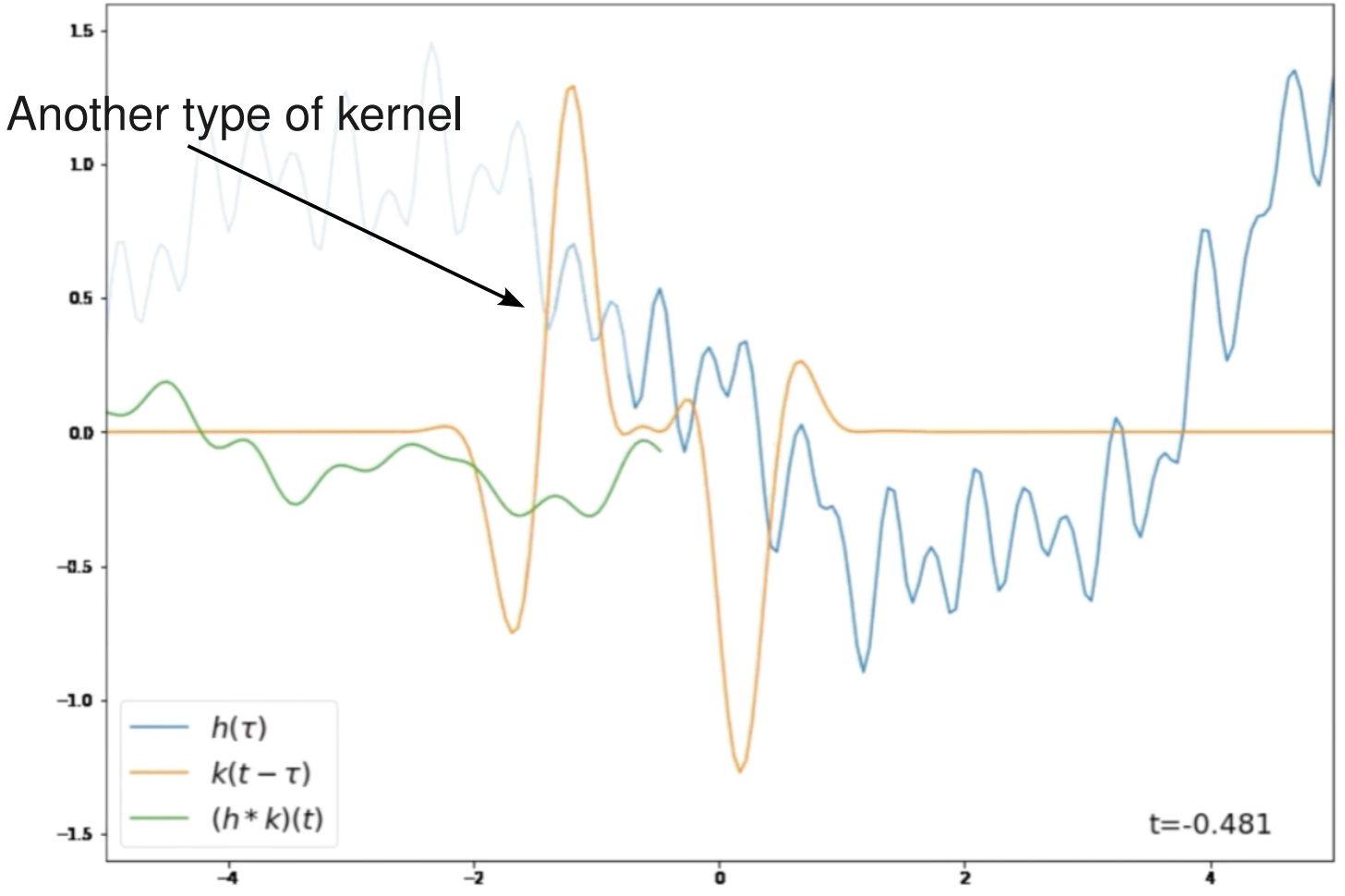
$$(h * k)(t) = \int_{-\infty}^{\infty} h(\tau)k(t-\tau)d\tau$$

Repeat *t* by *t*



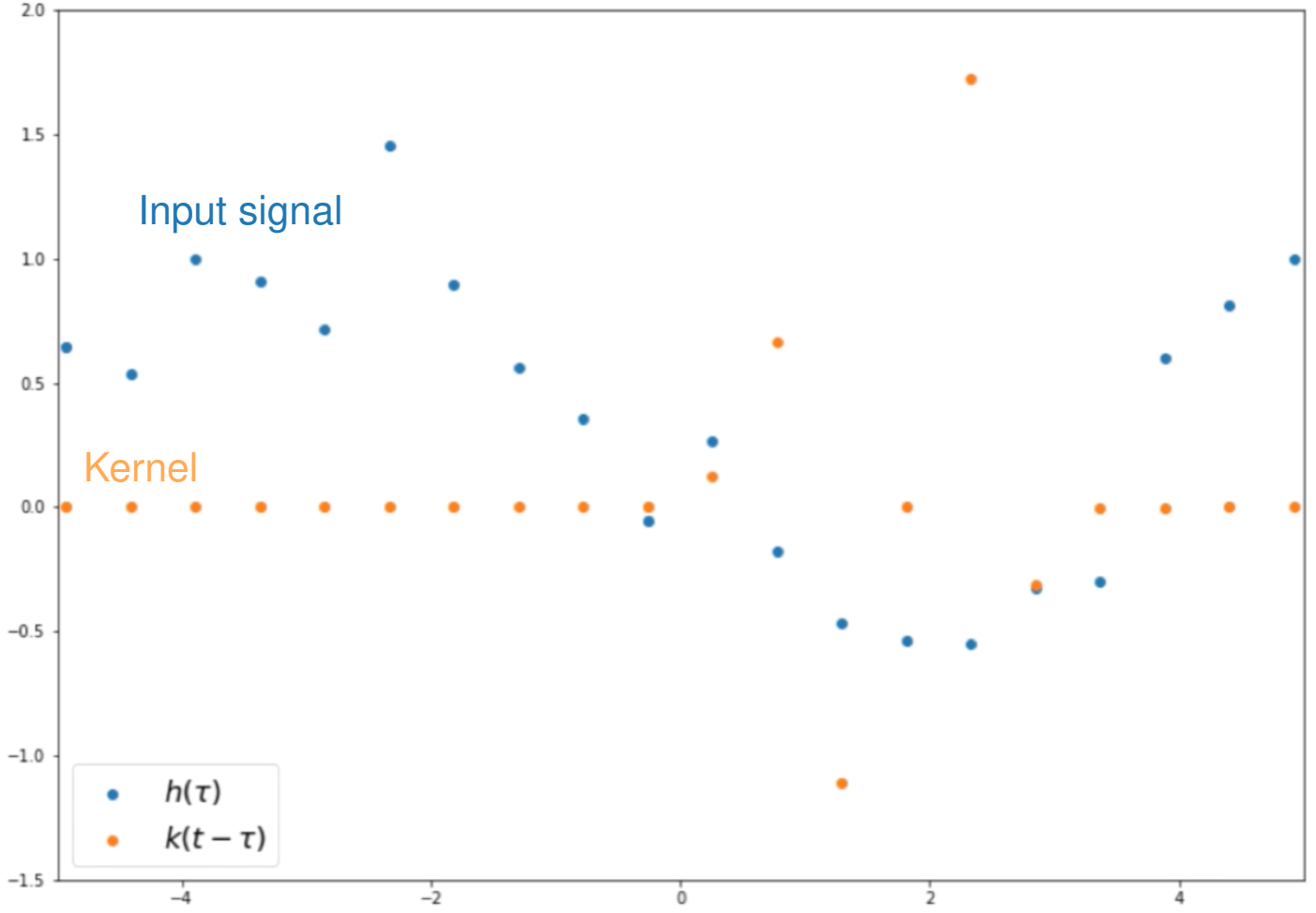
The convolution operation

$$(h * k)(t) = \int_{-\infty}^{\infty} h(\tau)k(t-\tau)d\tau$$



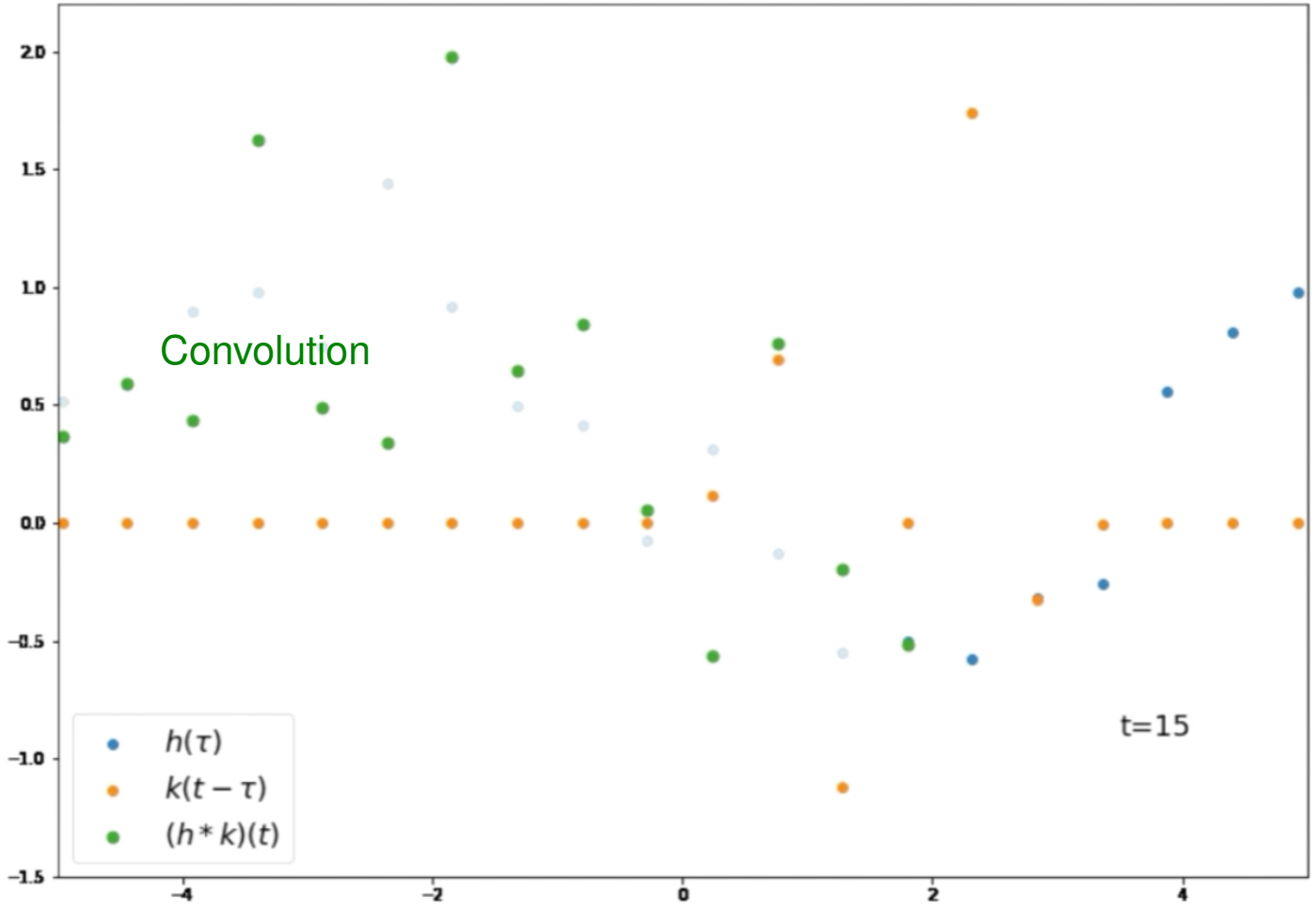
The discrete convolution

$$(h * k)(t) = \sum_{\tau=-\infty}^{\infty} h(\tau)k(t-\tau)$$



The discrete convolution

$$(h * k)(t) = \sum_{\tau=-\infty}^{\infty} h(\tau)k(t-\tau)$$



The discrete convolution

2D convolution: to apply to image data

$$(\mathbf{h} * \mathbf{k})(i,j) = \sum_{m,n} h(m,n)k(i-m, j-n)$$

$$\mathbf{h} \in \mathbb{R}^{n_h \times n_w} \qquad \mathbf{k} \in \mathbb{R}^{k_h \times k_w}$$

Input height and width

Kernel height and width (smaller)

The discrete convolution

2D convolution: to apply to image data

$$(\mathbf{h} * \mathbf{k})(i,j) = \sum_{m,n} h(m,n)k(i-m, j-n)$$

$$\mathbf{h} \in \mathbb{R}^{n_h \times n_w} \qquad \mathbf{k} \in \mathbb{R}^{k_h \times k_w}$$

Input height and width

Kernel height and width (smaller)

For computational convenience, implemented as the **cross-correlation operation:**

$$(\mathbf{h} * \mathbf{k})(i,j) = \sum_{m,n} h(i+m, j+n)k(m,n)$$

The convolutional layer

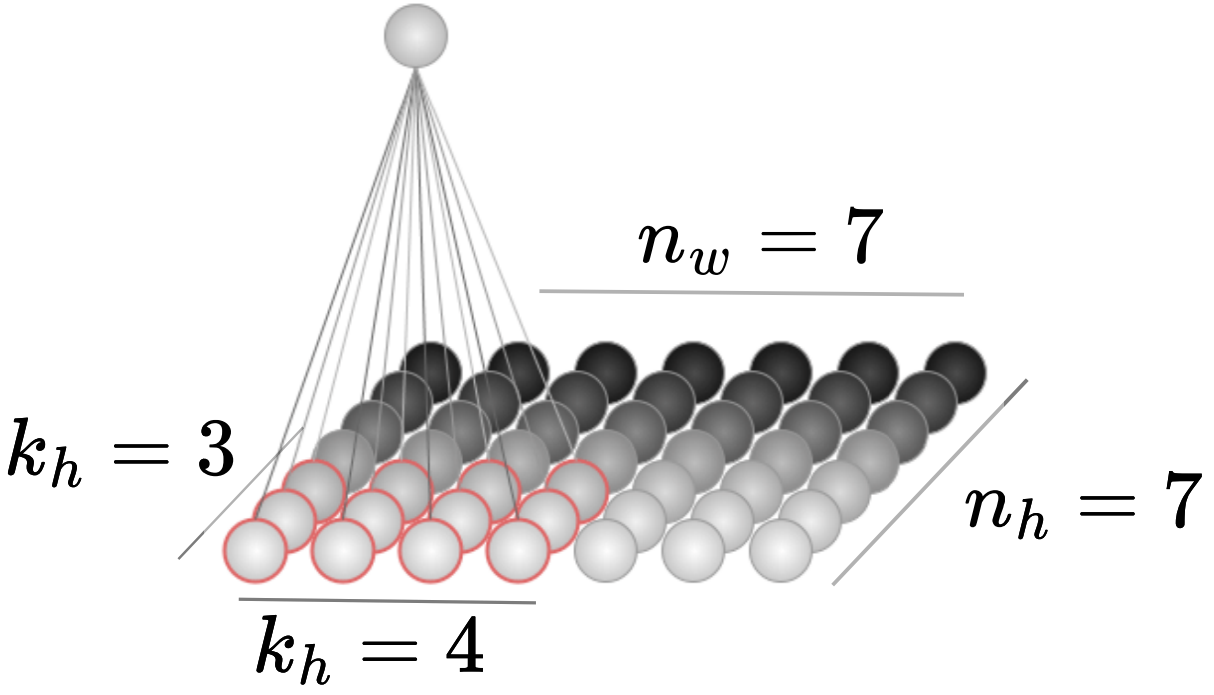
Input (or hidden activation):

$$\mathbf{h} \in \mathbb{R}^{n_h \times n_w}$$

Kernel or "Filter": it's a matrix of learnable weights

$$\mathbf{k} \in \mathbb{R}^{k_h \times k_w}$$

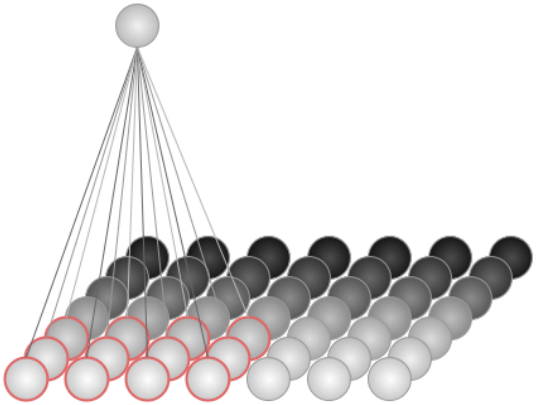
Example: $\mathbf{x} =: \mathbf{h}^{(0)} \in \mathbb{R}^{7 \times 7}$ Grayscale image



In the convolution operation:

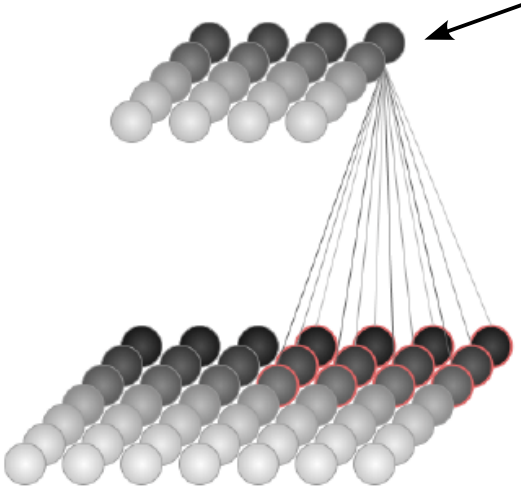
$$(\mathbf{h} * \mathbf{k})(i,j) = \sum_{m,n} h(i+m, j+n)k(m,n)$$

the kernel **k** is progressively swept through the input **h**.



Weight sharing: use of the same small filters across the input

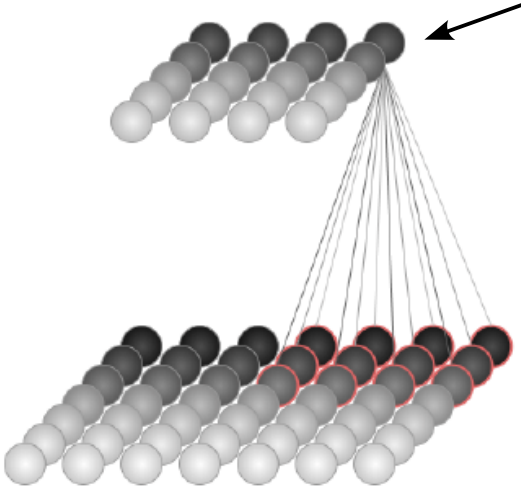
The convolutional layer



As in the hidden layers of the MLP, one applies **activation functions**

Convolutional layer combines **convolution + activation:**

The convolutional layer



As in the hidden layers of the MLP, one applies **activation functions**

Convolutional layer combines **convolution + activation:**

post-activation of layer / **Feature Map**

bias applied point-wise

$$\mathbf{h}^{(l)} = \sigma\left(\left(\mathbf{h}^{(l-1)} * \mathbf{k}^{(l-1)}\right) + b^{(l-1)}\right)$$

Convolution, of size:

$$\mathbb{R}^{(n_h - k_h + 1) \times (n_w - k_w + 1)}$$

(Typically smaller than the input!)

Assumptions & properties

Why do CNNs **leverage** specific **"structural priors"**?

"structural priors" = assumptions about structure of the data

Assumptions & properties

Why do CNNs **leverage** specific **"structural priors"**?

"structural priors" = assumptions about structure of the data

They can lead to *better* models, if applied to the *right* data.

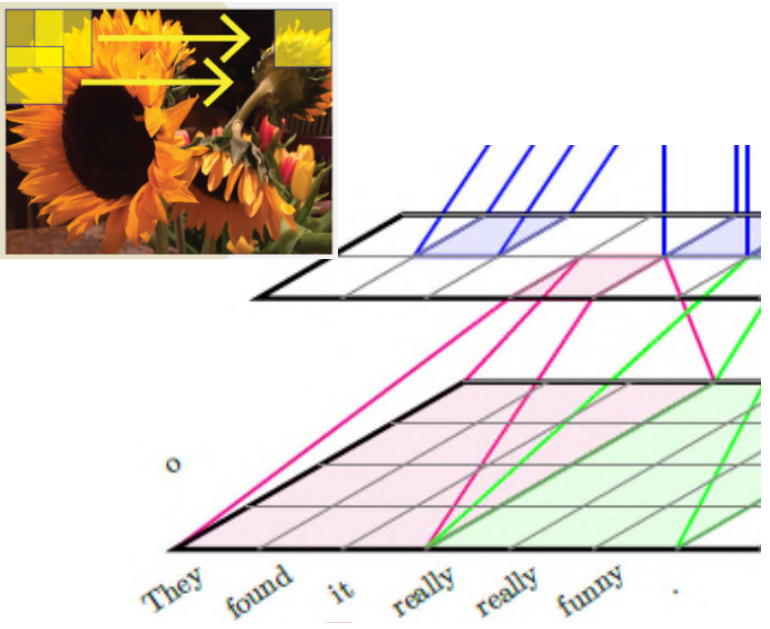
Assumptions & properties

Why do CNNs **leverage** specific **"structural priors"**?

"structural priors" = assumptions about structure of the data

They can lead to *better* models, if applied to the *right* data.

Right data: **images, text data** with combinations of features that are local and informative regardless of their exact position (shape edges in an image, keywords in a text).



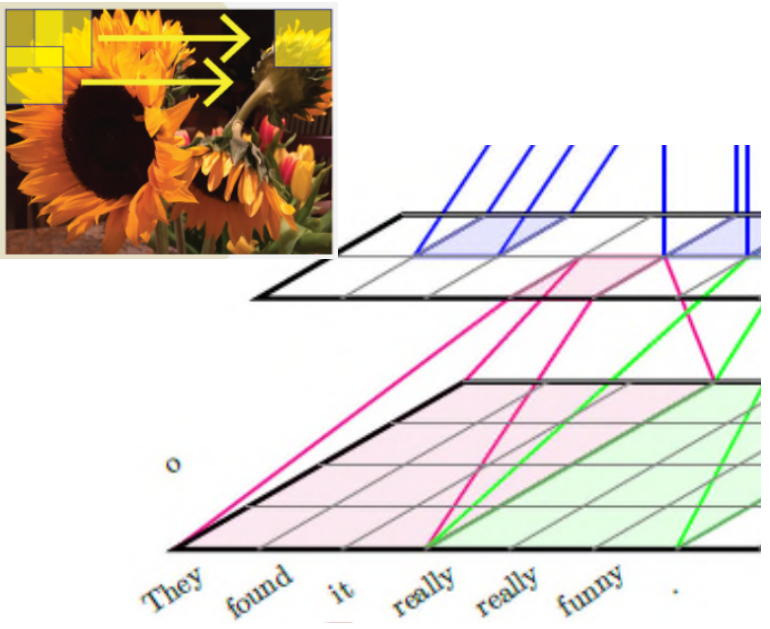
Assumptions & properties

Why do CNNs **leverage** specific **"structural priors"**?

"structural priors" = assumptions about structure of the data

They can lead to *better* models, if applied to the *right* data.

Right data: **images, text data** with combinations of features that are local and informative regardless of their exact position (shape edges in an image, keywords in a text).



Useful: **local feature detectors robust to exact location.**

Assumptions & properties

CNNs achieve local feature detection robust to features' exact location via **local connectivity** & **parameter sharing**.

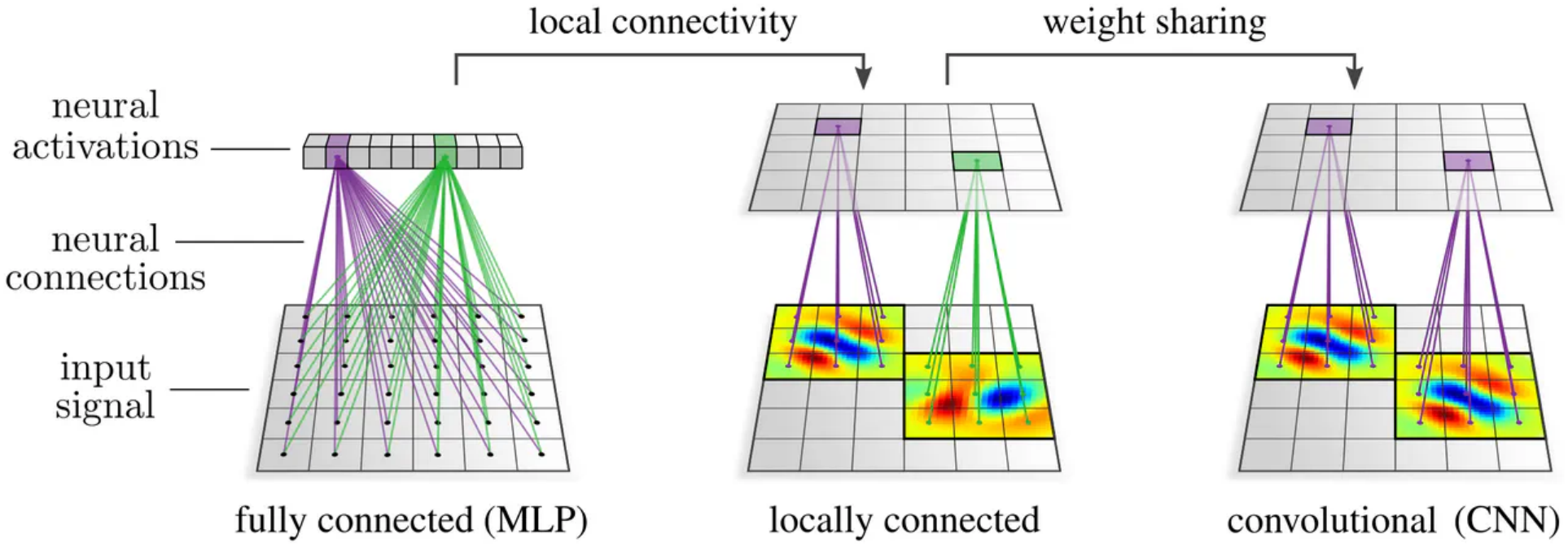


Image from https://maurice-weiler.gitlab.io/blog_post/cnn-book_2_conventional_cnns/

Assumptions & properties

It introduces **translational equivariance**: with small shifts of the input and the activations remain essentially the same.

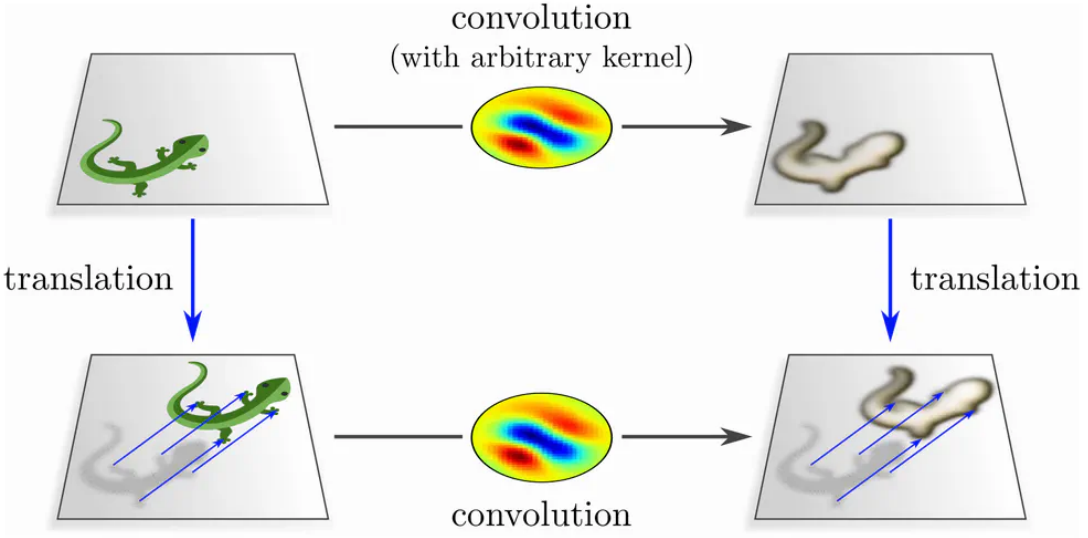
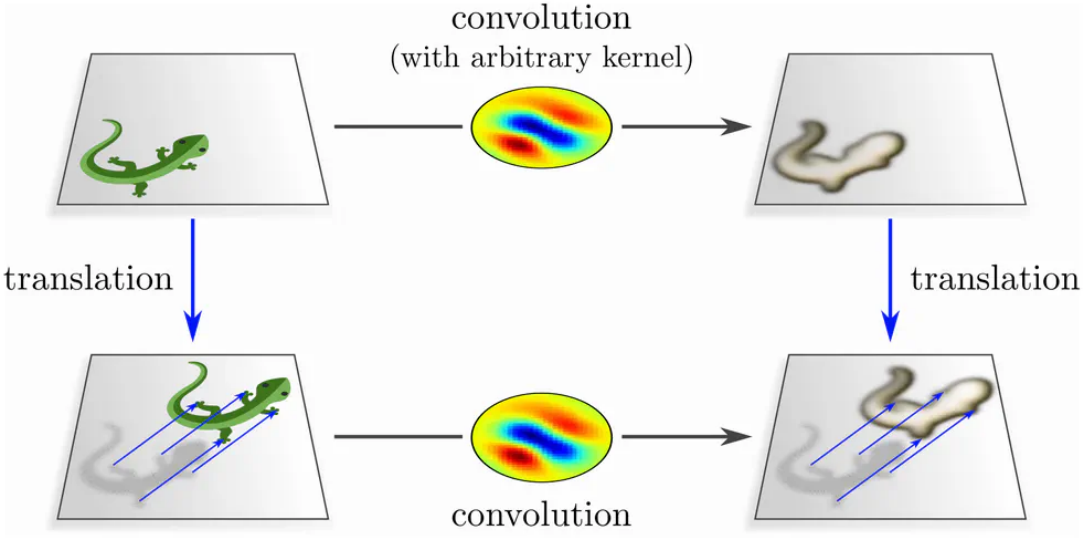


Image from https://maurice-weiler.gitlab.io/blog_post/cnn-book_2_conventional_cnns/

Assumptions & properties

It introduces **translational equivariance**: with small shifts of the input and the activations remain essentially the same.

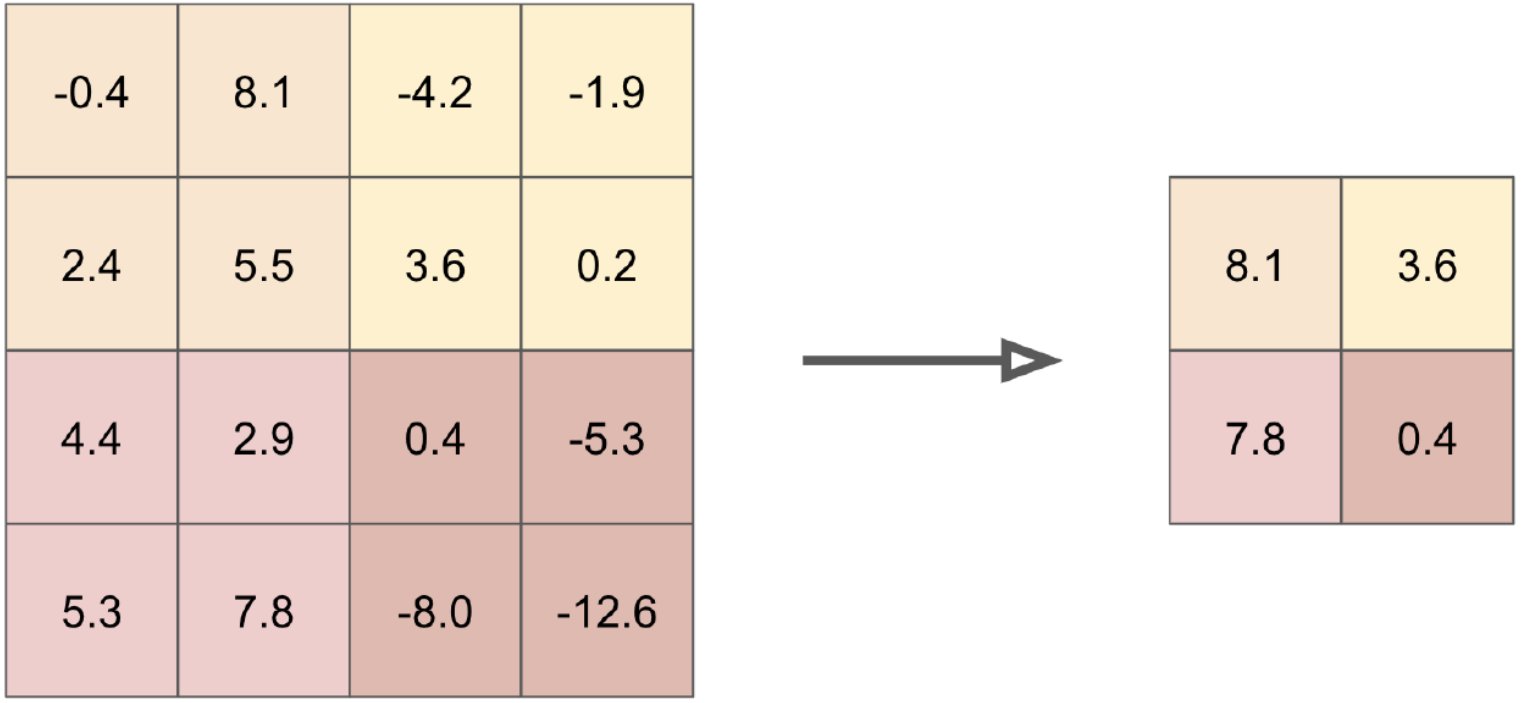


But also: fewer parameters (than other architectures with same depth) wtih faster convergence, less overfitting.

Image from https://maurice-weiler.gitlab.io/blog_post/cnn-book_2_conventional_cnns/

Pooling layer

To downsample the spatial dimensions of a layer:



Max Pooling with 2x2 window (also average pooling, others)

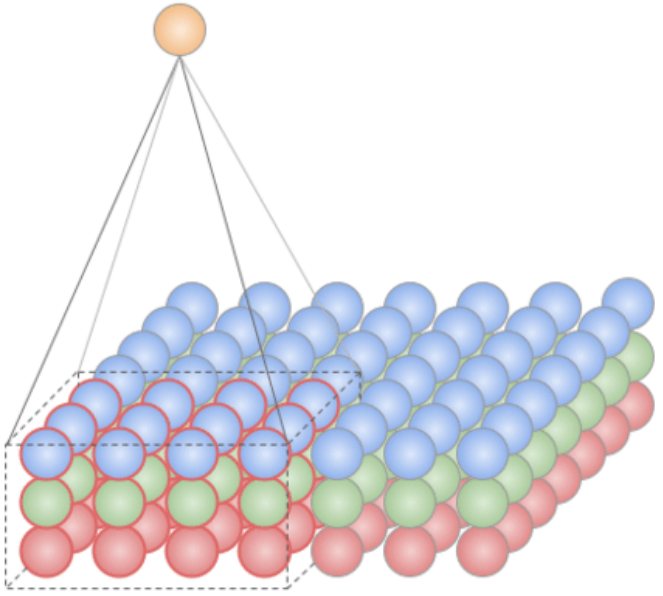
Extension to inputs with **multiple channels** of the convolution operation:

$$(\mathbf{h} * \mathbf{k})(i, j) = \sum_{m,n,p} h(i + m, j + n, p)k(m, n, p).$$

Extension to inputs with **multiple channels** of the convolution operation:

$$(\mathbf{h} * \mathbf{k})(i,j) = \sum_{m,n,p} h(i+m, j+n, p)k(m,n,p).$$

Main example: 3 channels for an input RGB image, giving $\mathbf{k} \in \mathbb{R}^{k_h \times k_w \times 3}$:



Here: $\mathbf{x} \in \mathbb{R}^{7\times7\times3}$ and $\mathbf{k} \in \mathbb{R}^{3\times4\times3}$

Many filters stacked together produce **multichannel outputs**:

$$(\mathbf{h} * \mathbf{k})(i, j, q) = \sum_{m,n,p} h(i + m, j + n, p)k(m, n, p, q).$$

Many filters stacked together produce **multichannel outputs**:

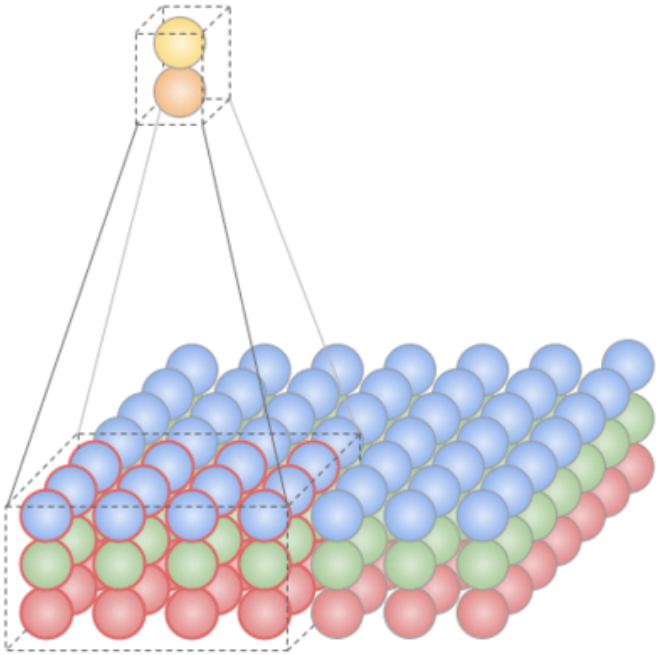
$$(\mathbf{h} * \mathbf{k})(i, j, q) = \sum_{m,n,p} h(i + m, j + n, p)k(m, n, p, q).$$

Hence $\mathbf{k} \in \mathbb{R}^{k_h \times k_w \times c_{in} \times c_{out}}$, with:
c_{in}: number of channels in the input;
c_{out}: number of channels in the output.

Many filters stacked together produce **multichannel outputs**:

$$(\mathbf{h} * \mathbf{k})(i, j, q) = \sum_{m,n,p} h(i + m, j + n, p)k(m, n, p, q).$$

Hence $\mathbf{k} \in \mathbb{R}^{k_h \times k_w \times c_{in} \times c_{out}}$, with:
c_{in}: number of channels in the input;
c_{out}: number of channels in the output.



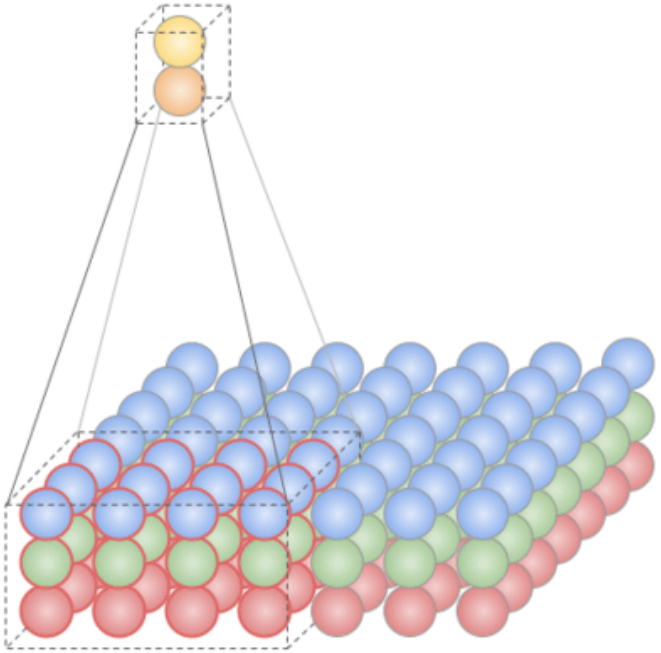
Here: $\mathbf{x} \in \mathbb{R}^{7 \times 7 \times 3}$ and $\mathbf{k} \in \mathbb{R}^{3 \times 4 \times 3 \times 2}$

The convolutional layer reads:

$$\mathbf{h}^{(l)} = \sigma\left((\mathbf{h}^{(l-1)} * \mathbf{k}^{(l-1)}) + \mathbf{b}^{(l-1)}\right),$$

The bias $\mathbf{b}^{(l-1)} \in \mathbb{R}^{c_{out}}$ is added pixel-wise to the convolution's output:

$$(\mathbf{h}^{(l-1)} * \mathbf{k}^{(l-1)}) \in \mathbb{R}^{(n_h - k_h + 1) \times (n_w - k_w + 1) \times c_{out}}.$$



Here: $\mathbf{x} \in \mathbb{R}^{7 \times 7 \times 3}$ and $\mathbf{k} \in \mathbb{R}^{3 \times 4 \times 3 \times 2}$

Recall: for a spatial dimension of size i and a kernel of width k:

$$\text{Output size} \quad o = i - k + 1$$

Recall: for a spatial dimension of size i and a kernel of width k:

$$\text{Output size} \quad o = i - k + 1$$

To keep the *same* spatial dimensions in the input and output of a convolutional layer, one can perform **padding with p zeros**:

$$o = i + p - k + 1$$

If $p = k - 1$ then $o = i$: **"SAME" padding**.

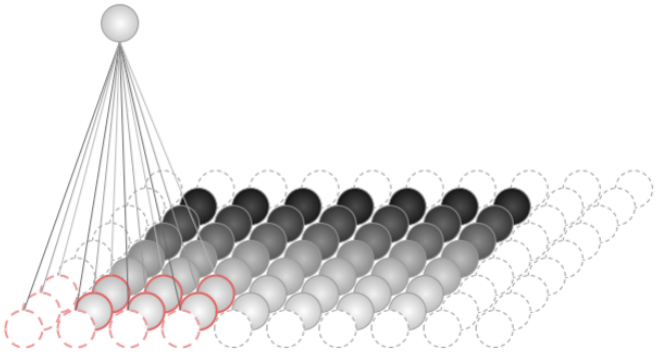
Recall: for a spatial dimension of size i and a kernel of width k:

$$\text{Output size} \quad o = i - k + 1$$

To keep the *same* spatial dimensions in the input and output of a convolutional layer, one can perform **padding with p zeros**:

$$o = i + p - k + 1$$

If $p = k - 1$ then $o = i$: **"SAME" padding**. Example: $p_h = 2$, $p_w = 3$



Additional flexibility over the spatial dimensions gained via **stride** s: the distance between consecutive positions of the kernel.

Additional flexibility over the spatial dimensions gained via **stride** s: the distance between consecutive positions of the kernel.

$s > 1 \rightarrow$ **downsampling of the input**

Additional flexibility over the spatial dimensions gained via **stride** s: the distance between consecutive positions of the kernel.

$s > 1 \rightarrow$ **downsampling of the input**

spatial dimension i + padding p + kernel width k + stride s:

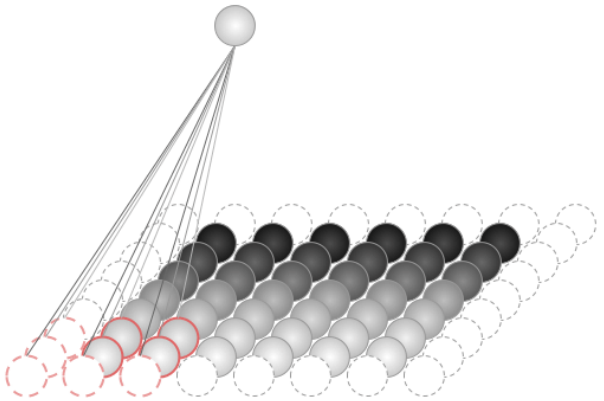
$$o = \left\lfloor \frac{i + p - k}{s} \right\rfloor + 1$$

Additional flexibility over the spatial dimensions gained via **stride** s: the distance between consecutive positions of the kernel.

$s > 1 \rightarrow$ **downsampling of the input**

spatial dimension i + padding p + kernel width k + stride s:

$$o = \left\lfloor \frac{i + p - k}{s} \right\rfloor + 1$$



Here: Kernel $\mathbf{k} \in \mathbb{R}^{3\times3\times1\times1}$, image $\mathbf{x} \in \mathbb{R}^{7\times6\times1}$, "SAME" padding, stride $(2, 2)$.

CNN: example of a classifier

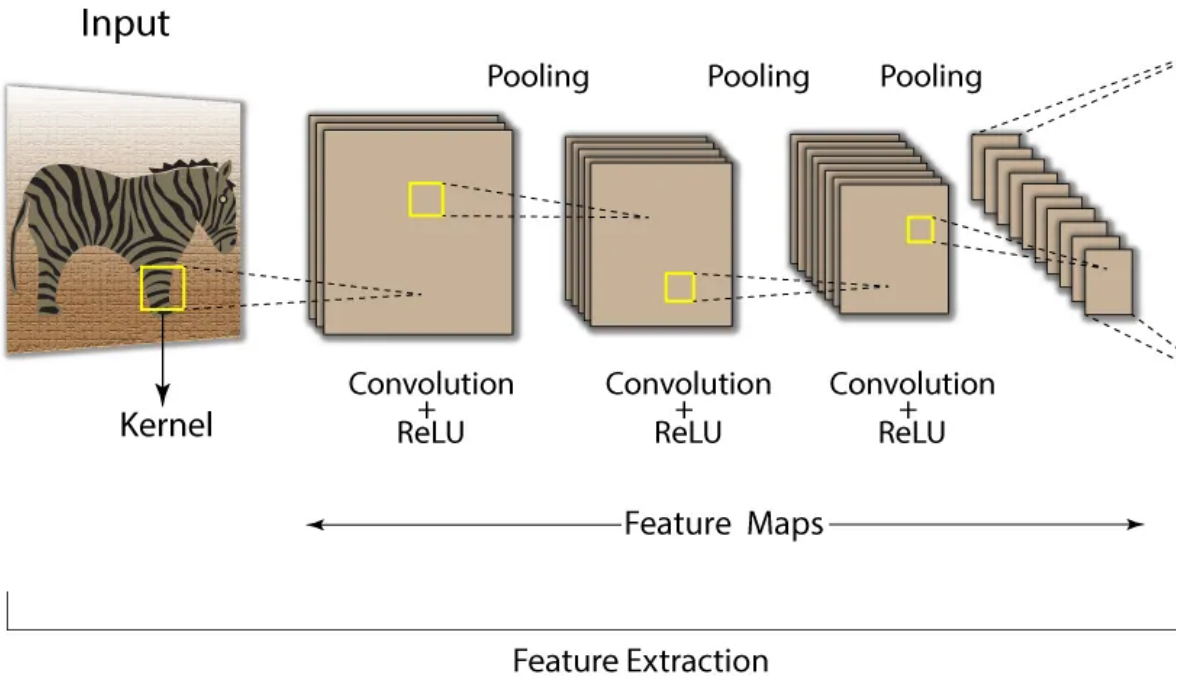


Image from: https://medium.com/@sinha.nikhil77/fundamental-of-image-classification-problem-using-convolution-neural-network-cnn-d538a14b26

CNN: example of a classifier

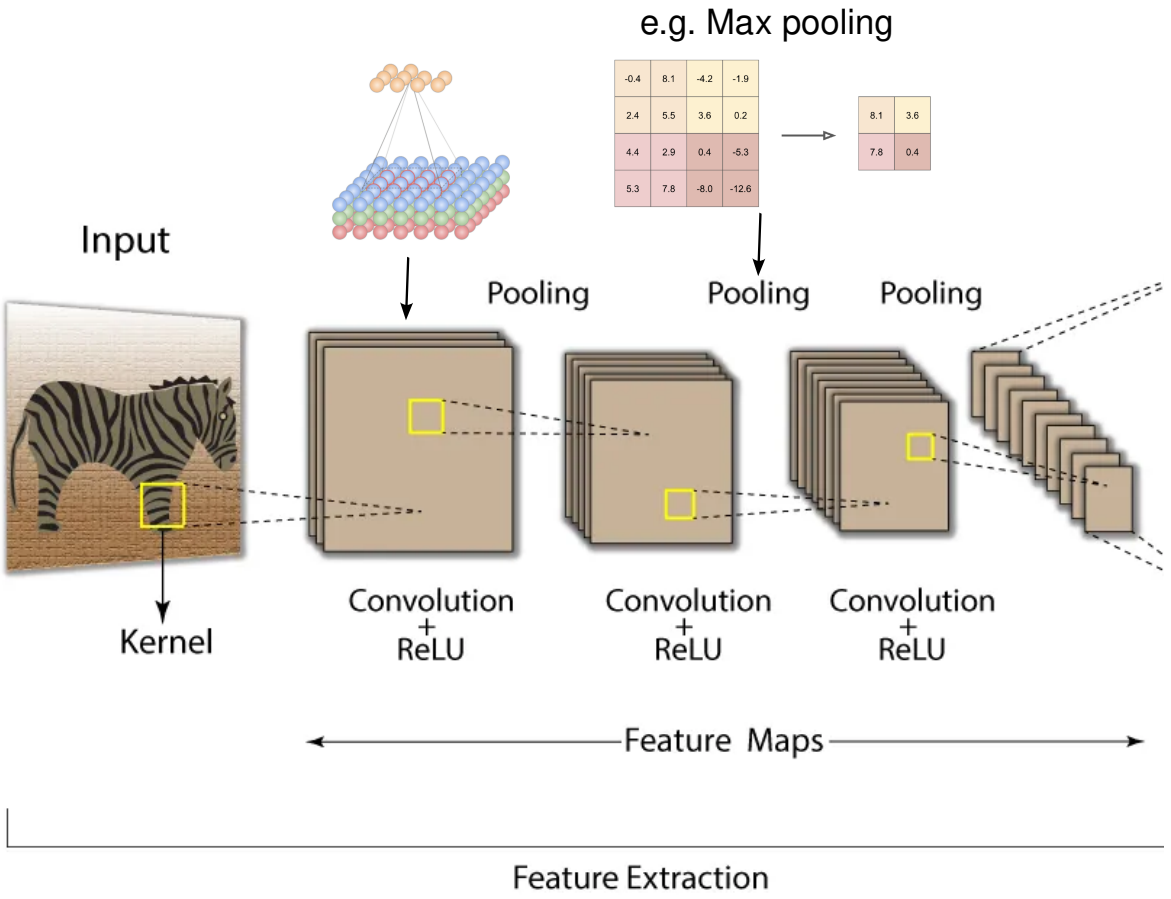


Image from: https://medium.com/@sinha.nikhil77/fundamental-of-image-classification-problem-using-convolution-neural-network-cnn-d538a14b26

CNN: example of a classifier

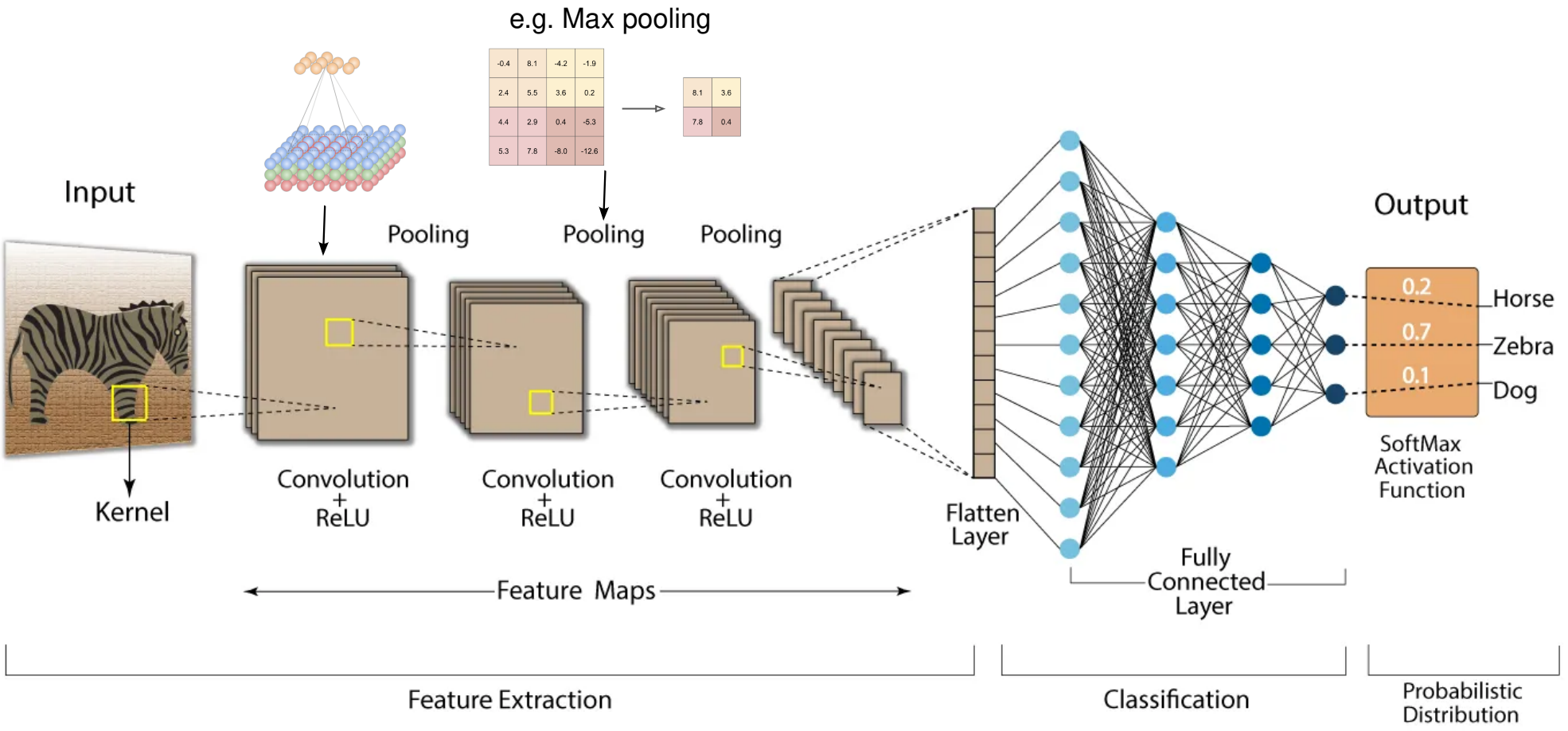
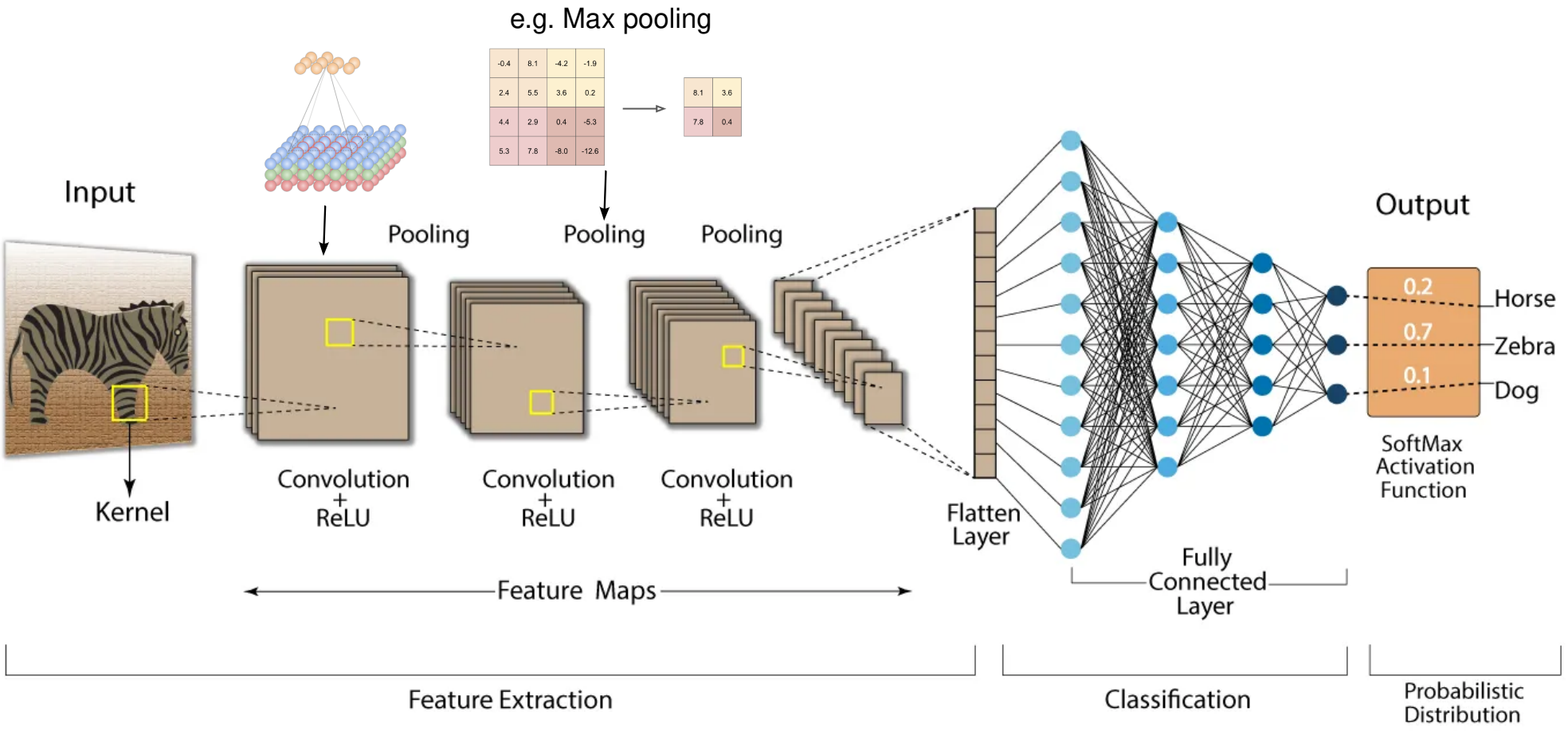


Image from: https://medium.com/@sinha.nikhil77/fundamental-of-image-classification-problem-using-convolution-neural-network-cnn-d538a14b26

CNN: example of a classifier

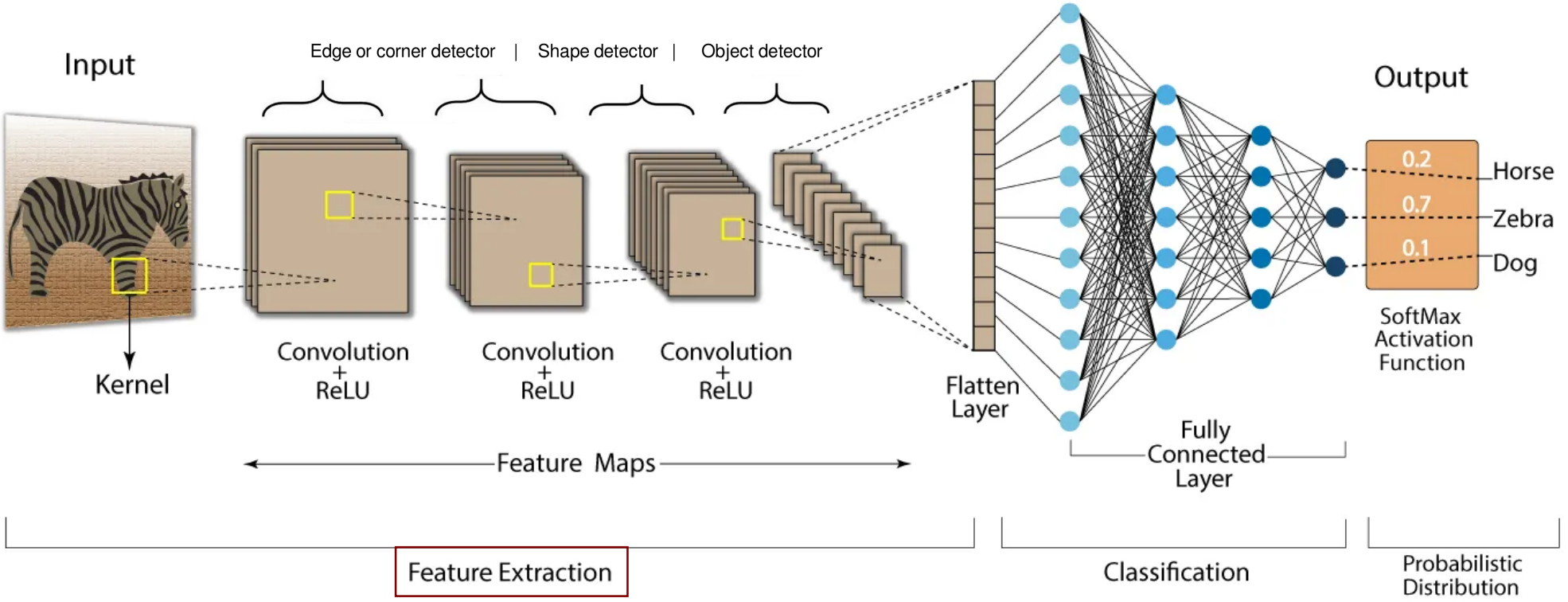


Training: classification loss function + error backpropagation to learn the weights of CNN filters, other weights and biases

Image from: https://medium.com/@sinha.nikhil77/fundamental-of-image-classification-problem-using-convolution-neural-network-cnn-d538a14b26

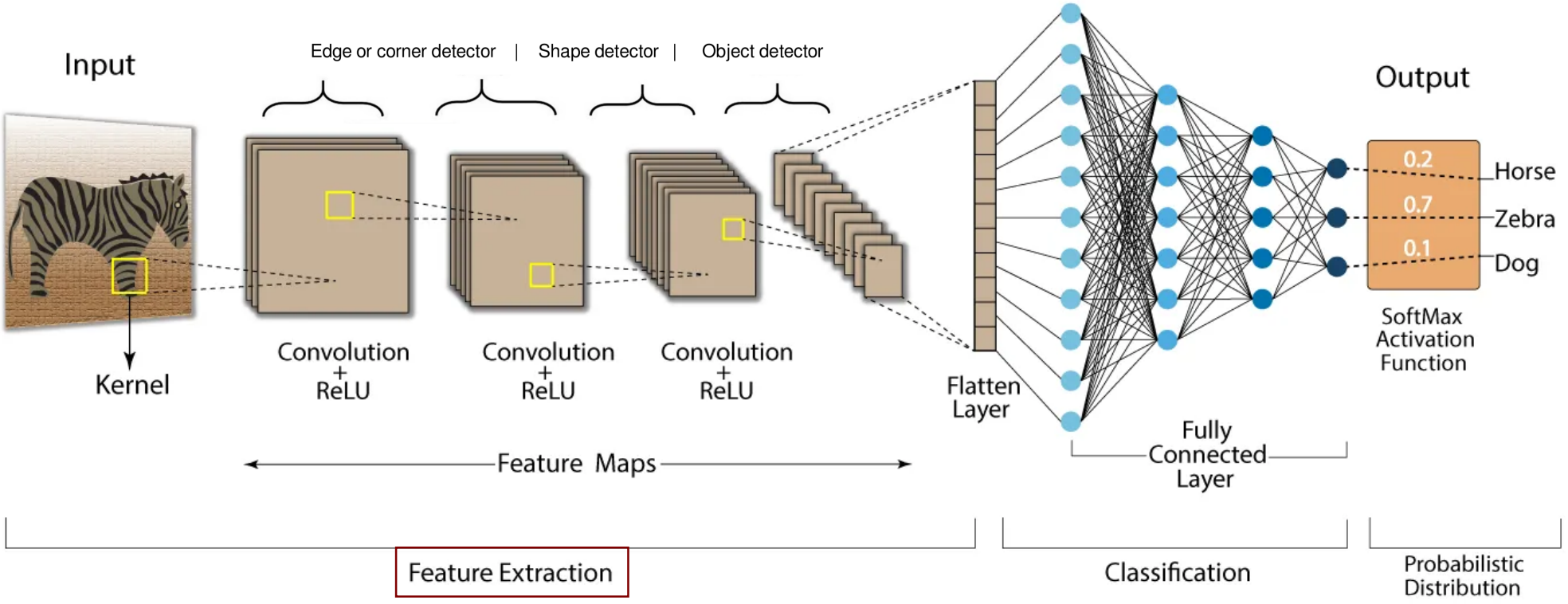
Hierarchical representations

They learn representations where layer activations learn increasingly more complex, coarser features combining lower-level features, see: LeCun, Bengio, Hinton, Deep Learning (2015)

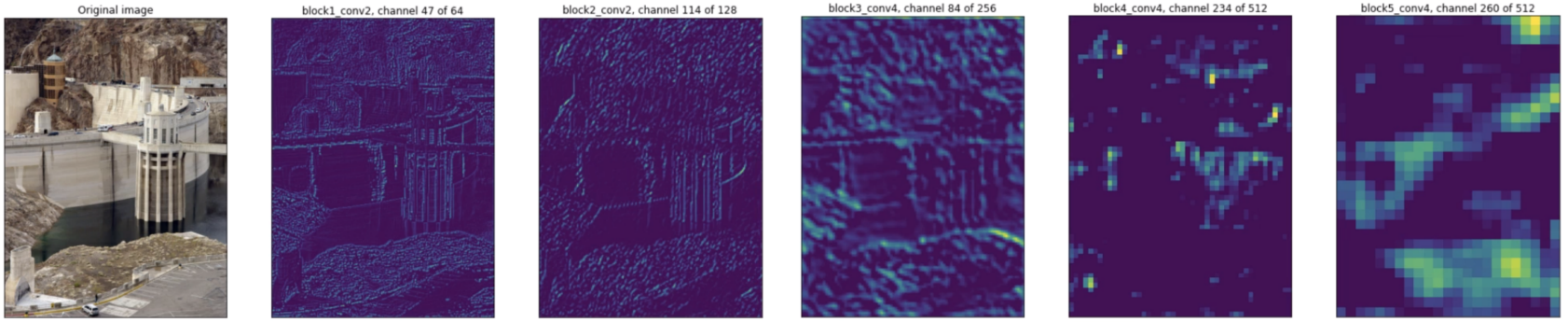


Hierarchical representations

They learn representations where layer activations learn increasingly more complex, coarser features combining lower-level features, see: LeCun, Bengio, Hinton, Deep Learning (2015)



Example: VGG-19 model (Simonyan and Zisserman , 2014)



CNN for street number transcription

Multi-digit numbers in Street View images

- Tens of millions of labelled examples

From Goodfellow et al., Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks (2014)
see also Goodfellow, Bengio, Courville, *Deep Learning*, Chap 11

CNN for street number transcription

Multi-digit numbers in Street View images

Best architecture has **11 layers (8 convolutional)**, reaches 96% accuracy

- Tens of millions of labelled examples

- Increase in representational capacity by adjusting 'depth' (# of hidden layers)

- Depth is essential to a complex task involving localization, segmentation, and recognition of motifs

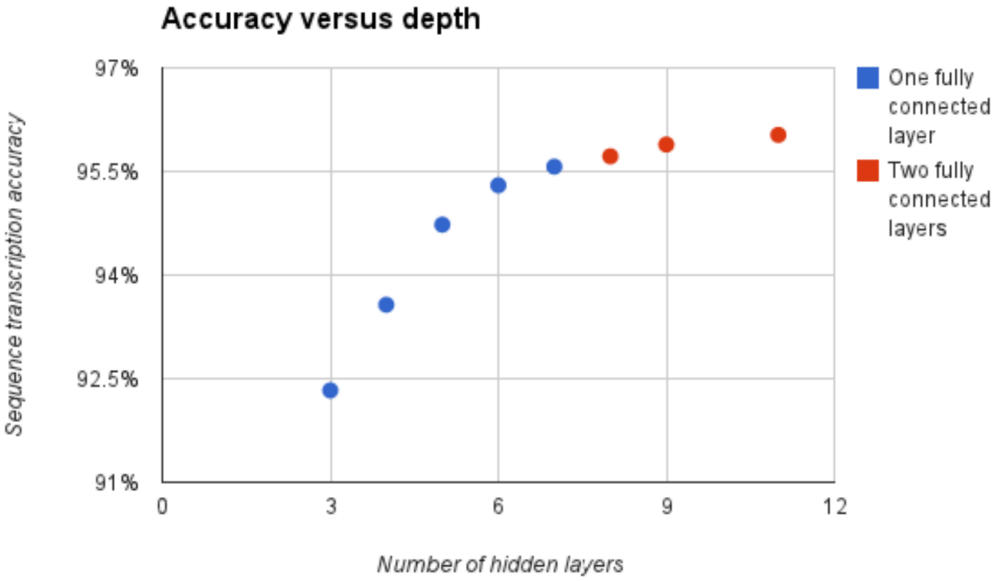


Figure 4: Performance analysis experiments on the public SVHN dataset show that fairly deep architectures are needed to obtain good performance on the sequence transcription task.

From Goodfellow et al., Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks (2014)
see also Goodfellow, Bengio, Courville, *Deep Learning*, Chap 11

ImageNet (ILSVRC)

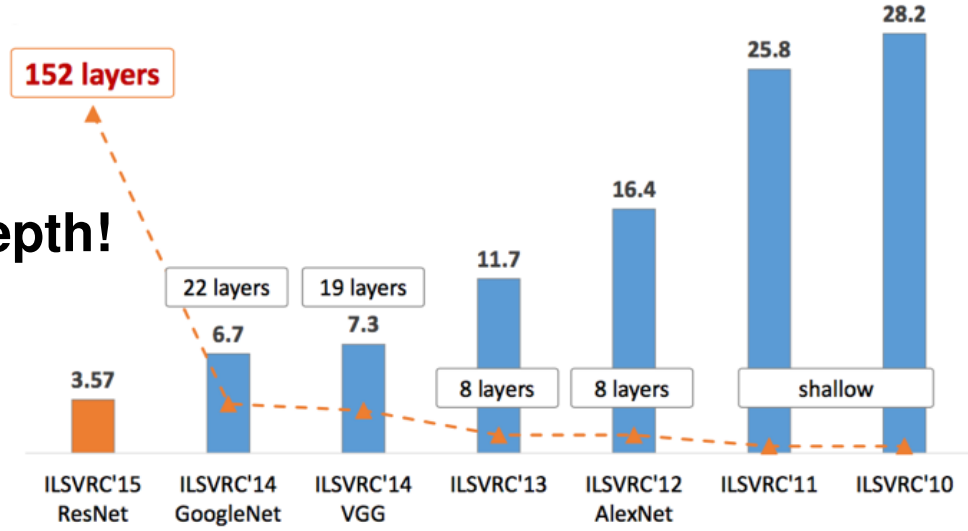
ILSVRC: Large Scale Visual Recognition Challenge

1.2 million images with 1000 classes for recognition

Annual competition run by ImageNet since 2010 for the task of image recognition and classification

CNN: major leaps forward in this competition

Misclassification error



Major increase in accuracy thanks to depth!

Figure from https://iq.opengenus.org/evolution-of-cnn-architectures/

Summary

1. The building block of CNNs is the convolutional layer, performing: convolution (with shared learnable kernel) + non-linearity;

Summary

1. The building block of CNNs is the convolutional layer, performing: convolution (with shared learnable kernel) + non-linearity;

2. Local connectivity + parameter sharing = local feature detection with translational equivariance, suitable for image and text data;

Summary

1. The building block of CNNs is the convolutional layer, performing: convolution (with shared learnable kernel) + non-linearity;

2. Local connectivity + parameter sharing = local feature detection with translational equivariance, suitable for image and text data;

3. Convolutional layers alternated with pooling layers (subsampling);

Summary

1. The building block of CNNs is the convolutional layer, performing: convolution (with shared learnable kernel) + non-linearity;

2. Local connectivity + parameter sharing = local feature detection with translational equivariance, suitable for image and text data;

3. Convolutional layers alternated with pooling layers (subsampling);

4. Convolutional layers handle multi-channel inputs and outputs;

Summary

1. The building block of CNNs is the convolutional layer, performing: convolution (with shared learnable kernel) + non-linearity;

2. Local connectivity + parameter sharing = local feature detection with translational equivariance, suitable for image and text data;

3. Convolutional layers alternated with pooling layers (subsampling);

4. Convolutional layers handle multi-channel inputs and outputs;

5. Padding and strides are used to adjust spatial dimensions;

Summary

1. The building block of CNNs is the convolutional layer, performing: convolution (with shared learnable kernel) + non-linearity;

2. Local connectivity + parameter sharing = local feature detection with translational equivariance, suitable for image and text data;

3. Convolutional layers alternated with pooling layers (subsampling);

4. Convolutional layers handle multi-channel inputs and outputs;

5. Padding and strides are used to adjust spatial dimensions;

6. Through depth, CNNs can learn hierarchical data representations.