UNIVERSITÄT BONN

Introduction to Convolutional Neural Networks

Elena Trunz

February 24, 2025

Visual Computing Group, University of Bonn

Overview

Why Convolution?

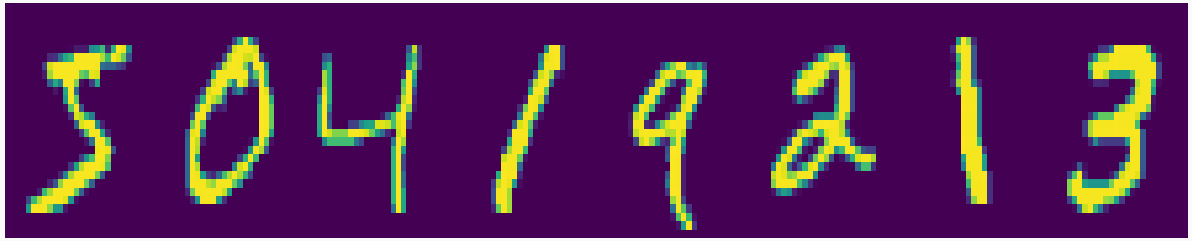
Convolution operation in machine learning

Understanding convolution

Convolutional neural networks

Why Convolution?

Invariance in the network structure



- Consider the task of recognizing handwritten digits:
  - Input image: set of pixel intensities
  - Desired output: posterior probability over the ten digits
- A system recognizing digits should be invariant to
  - Translations
  - Scaling
  - (Small) rotations
  - Some elastic deformations

Fully connected network?

- Given enough training data a fully connected network solves the task.
- However, we can do better:
  - Pixels near to each other are more correlated to each other than pixels far apart.
  - Local features which are useful in one region of the image are likely to be useful elsewhere as well, e.g. for detecting a translated object.

The invention of convolutional neural networks

In 1989 Yann LeCun proposed to build invariance properties into the structure of a neural network. This is the basis for convolutional neural networks.

Convolution operation in machine learning

Defining discrete convolution

- For two one-dimensional signals $x \in \mathbb{R}^T$ and $k \in \mathbb{R}^T$, convolution is defined as

$$s(t) = (x * k)(t) = \sum_{m=0}^{T} x(m)k(t-m), \tag{1}$$

  where t denote the position in the output sequence.

- In 2D, we require a kernel matrix $K \in \mathbb{R}^{O,P}$ and a image matrix $I \in \mathbb{K}^{N,M}$

$$S(i,j) = (K * I)(i,j) = \sum_{m}^{M} \sum_{n}^{N} I(i-m, j-n)K(n,m) \tag{2}$$

  where (i,j) denote the position in the output image.

Defining cross-correlation

- Cross-correlation is convolution without flipping the kernel:

$$S(i,j) = (K * I)(i,j) = \sum_{m}^{M} \sum_{n}^{N} I(i+m, j+n) K(m,n) \quad (3)$$

- Many machine-learning libraries implement cross-correlation and call it convolution.
- In this course we will follow their example.

Convolution

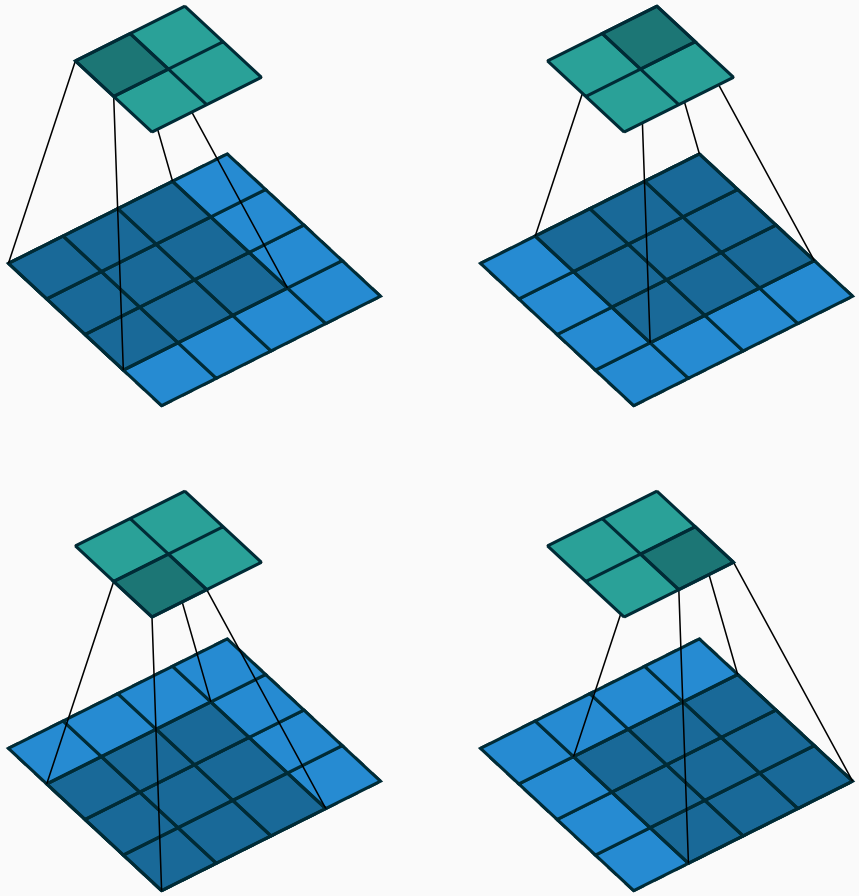


Illustration of the convolution operation [DV16].

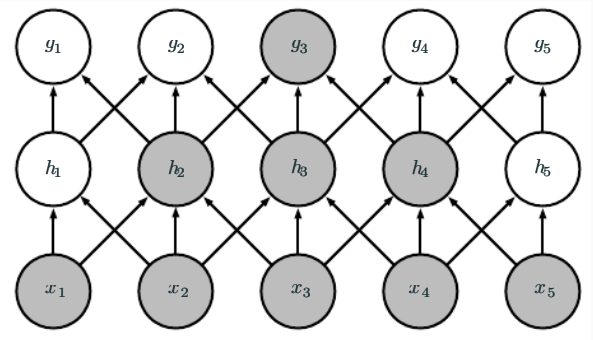
Advantages of convolutions

The convolution operation allows us to do this for inputs of varying sizes in a convenient way, and affords advantages of

1. local receptive fields (sparse connectivity limited in range vs. fully connected)
2. parameter sharing (allows for detection of same features all over the image)
3. equivariant representations and subsampling (reduces information hierarchically)

Sparse connectivity and interaction

- In CNNs, the kernel or filter is often much smaller than the input size. This allows us to process images of varying sizes. For an input image of millions of pixels we can still detect meaningful features which are local, such as edges.
- A small kernel does not mean that the area of interaction is limited. For a deep network, the units may *indirectly* interact with a larger portion of the input as information propagates up the layers:



Unit g_3 indirectly interacts with the whole input [GBC16]

Parameter sharing

- Standard feed-forward network has unit-specific weights.
- Each weight is used only once when computing the activations for subsequent layers.
- In a CNN, the weights of each kernel is applied to every position of the input.
- This *parameter sharing* reduces the amount of storage per model.

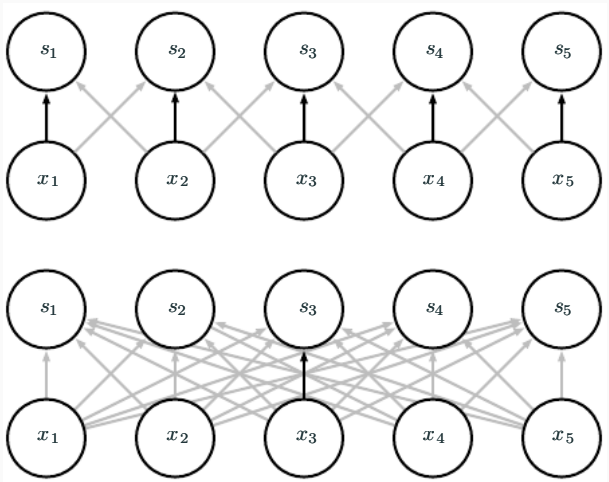


Image was taken from [GBC16]

Translation equivariance

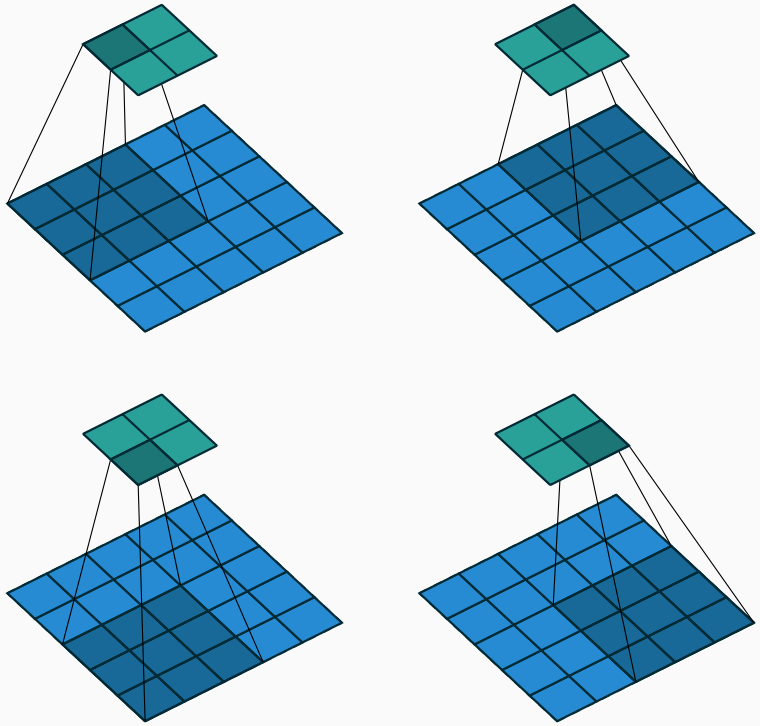
- An *equivariant* function is one where for some given change in the input, the output changes in the same way.
- With images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.

Multiple filters / kernels

- In the context of CNNs, we refer to convolution not in the strict mathematical sense, but an operation that applies many convolutions in parallel.
- Convolution with a single kernel extracts only one kind of feature (at many spatial locations).
- For each layer, however, we are interested in extracting many kinds of features (at many spatial locations).

Strided convolution

- To save computational expense, we can skip over some positions of the kernel, i.e. not extract our feature response as finely.



Stride two convolutions [DV16].

Padded convolution

- We can control the size of convolution inputs and outputs through padding.
- Without padding, the spatial extent of the network would shrink, with the rate of shrinking directly proportional to the size of the filter or kernel.

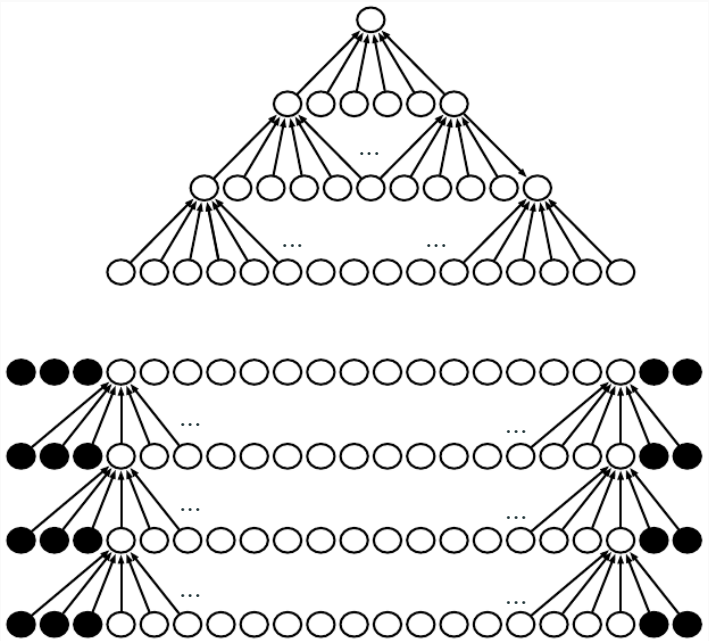
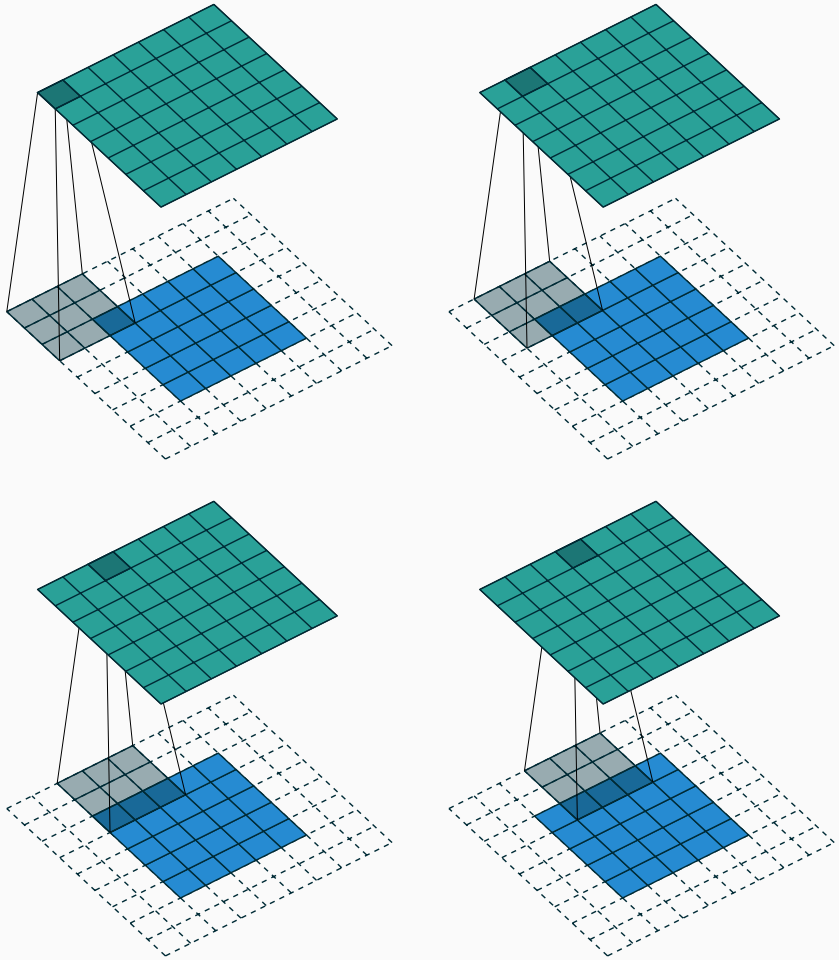


Image was taken from [GBC16]

Visualization of padded convolution



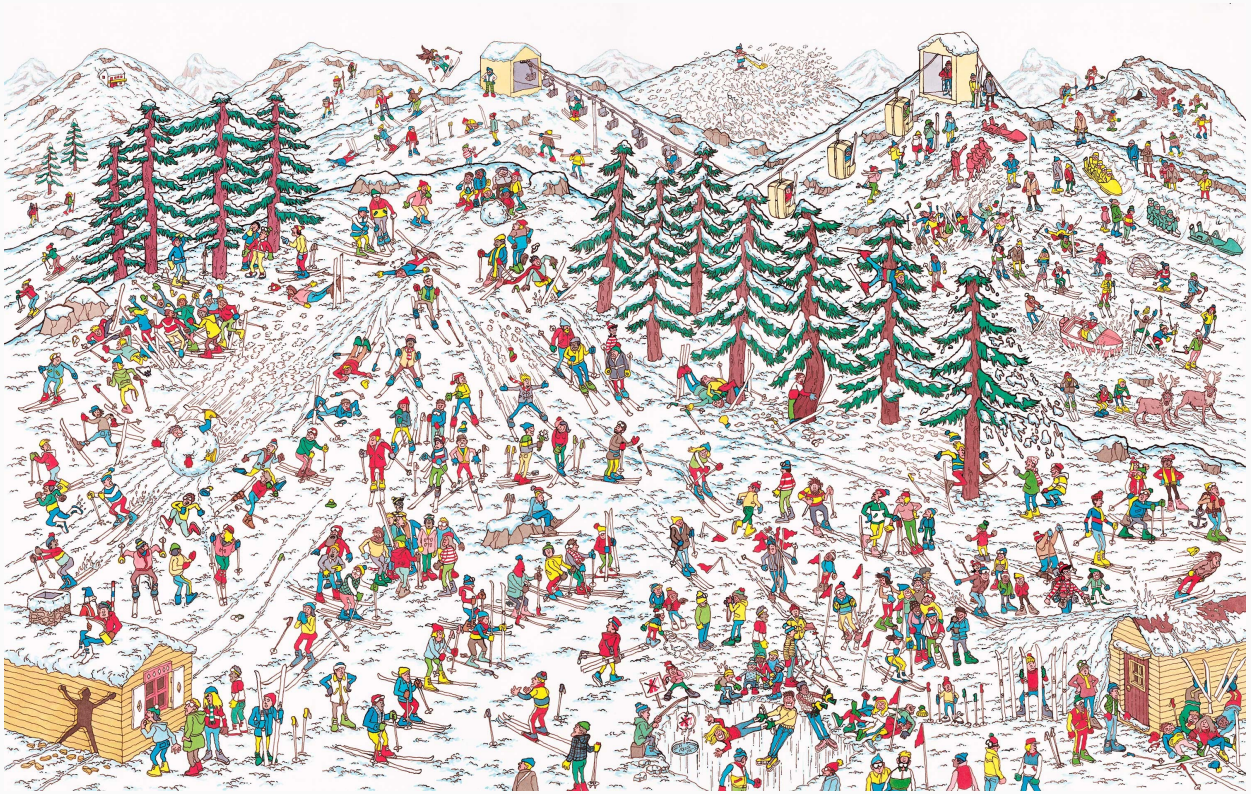
Fully padded convolutions with unit strides [DV16].

Summary

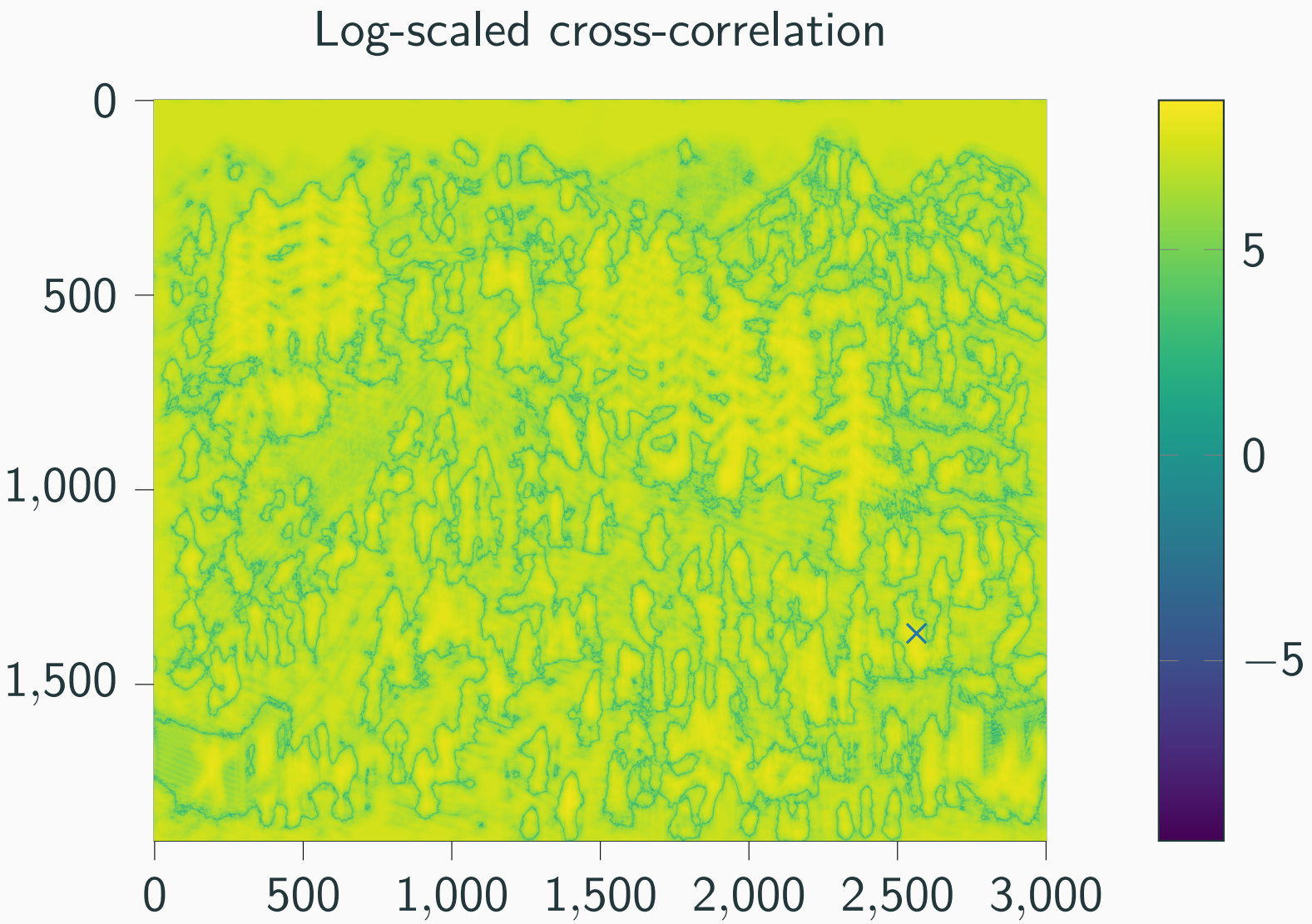
- The convolution operation slides convolution kernels over an image.
- Padding avoids losing pixels on the side.
- Strided convolutions downsample the input.
- Moving in steps of two pixels, for example, cuts the resolution in half.

Understanding convolution

Getting computers to find Waldo



Finding Waldo via cross-correlation.



Summary

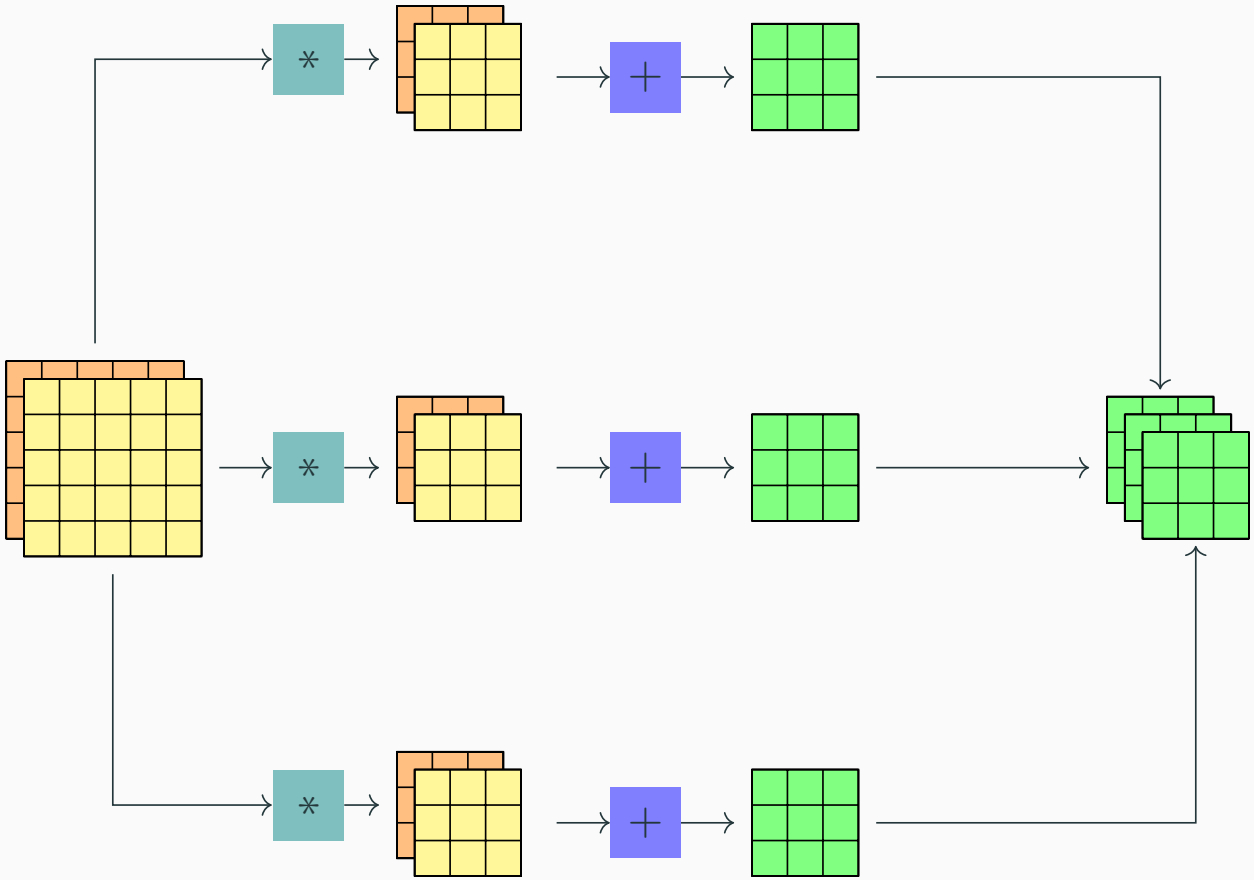
- Cross-correlation is called convolution in the machine learning literature.
- Patterns can be located in signals via cross-correlation.

Convolutional neural networks

Motivating convolutional neural networks (CNN)

- Fixed filters work if we are looking for a very specific waldo.
- In other cases, we need a better solution.
- Convolutional neural networks rely on filter optimization via back-propagation.
- Filter optimization turns CNNs into very versatile tools!

Multichannel convolution



The plot shows a convolution computation using a $3x2x3x3$ kernel on a $2x5x5$ input. The kernel pairs convolve with the input, producing $3x3$ results. $+$ adds the two channels for each of the three tensors. Finally, everything is stacked. Inspired by [DV16, page 9].

Computing the output shape of a CNN layer

One can determine the output shape for each dimension individually. Without zero padding and a stride size of one,

$$o = (i - k) + 1 \tag{4}$$

can be used to compute the output size. i denotes the input size, and k is the kernel size. [DV16] covers all cases which appear in practice.

Image to column and the forward pass

We already know how to train dense network layers using matrix multiplication. Training a CNN the same way requires restructuring the image to express convolution as matrix multiplication,

$$\overline{\mathbf{h}} = \mathbf{K}_f \mathbf{v}_l + \mathbf{b}, \tag{5}$$

$$\mathbf{h}_f = f(\overline{\mathbf{h}}). \tag{6}$$

$\mathbf{v}_l \in \mathbb{R}$ denotes the restructured image input. $\mathbf{K}_f \in \mathbb{R}^{k_o, k_i \cdot k_h \cdot k_w}$ the flattened restructured kernel. o, i, h, w denote the output, input, height, and width dimensions, respectively.

The backward pass

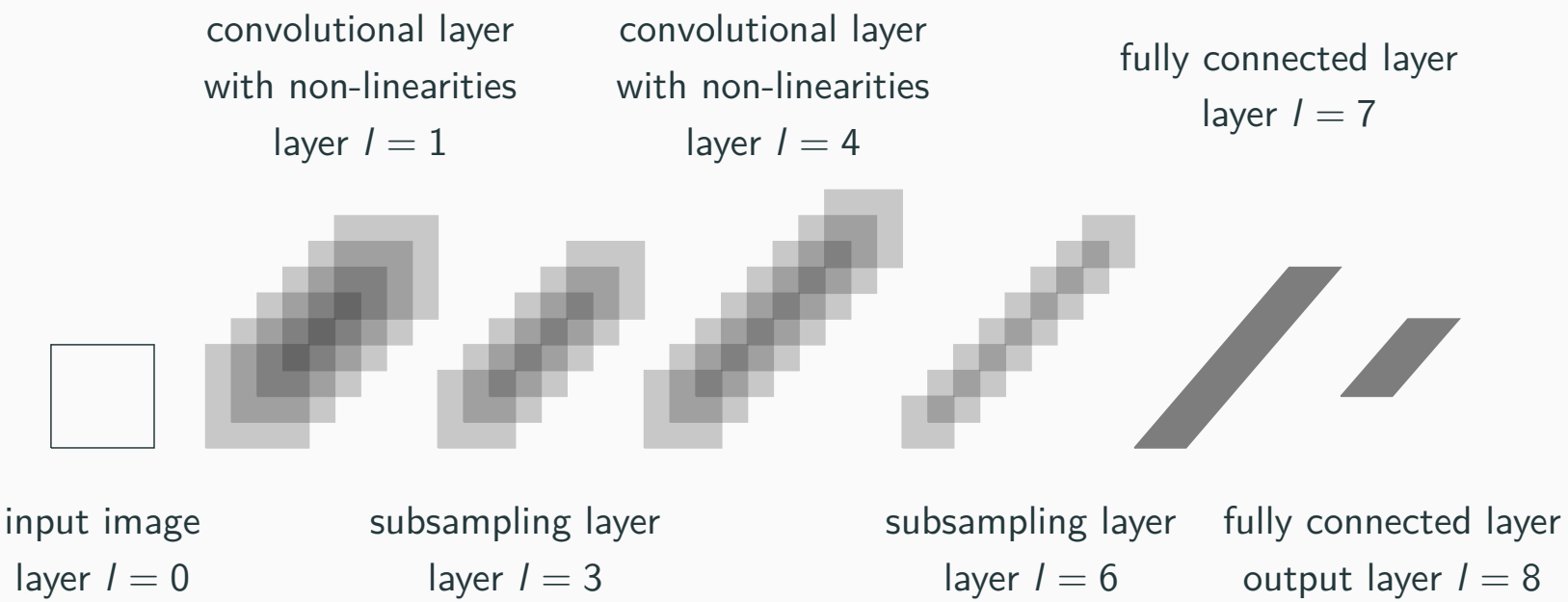
We apply the rules for dense layers to the restructured convolutional layer data,

$$\delta \mathbf{K}_f = [f'(\overline{\mathbf{h}}) \odot \triangle]_f \mathbf{v}_I^T, \qquad \delta \mathbf{b} = f'(\overline{\mathbf{h}}) \odot \triangle, \tag{7}$$

$$\delta \mathbf{x} = \left(\mathbf{K}_f^T [f'(\overline{\mathbf{h}}) \odot \triangle]_f\right)_{I^{-1}}. \tag{8}$$

With I and I^{-1} denoting the `im2col` and `col2im` operations. All major deep learning frameworks have both operations built in.

The classifier at the end



The LeNet-architecture[LeC+89] as illustrated by [Stu20].

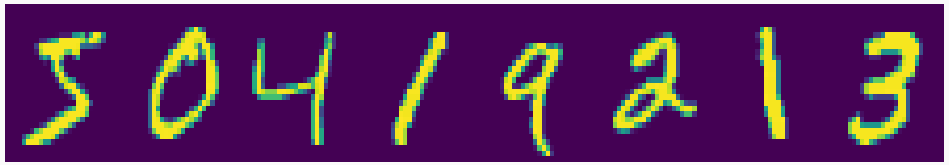
The shifting input problem

- With the tools we have seen, shifting an input also shifts the CNN output before the dense classifier.
- Shifting the input would shift the input in front of the final dense-classifier neurons.
- We want invariance to translation.

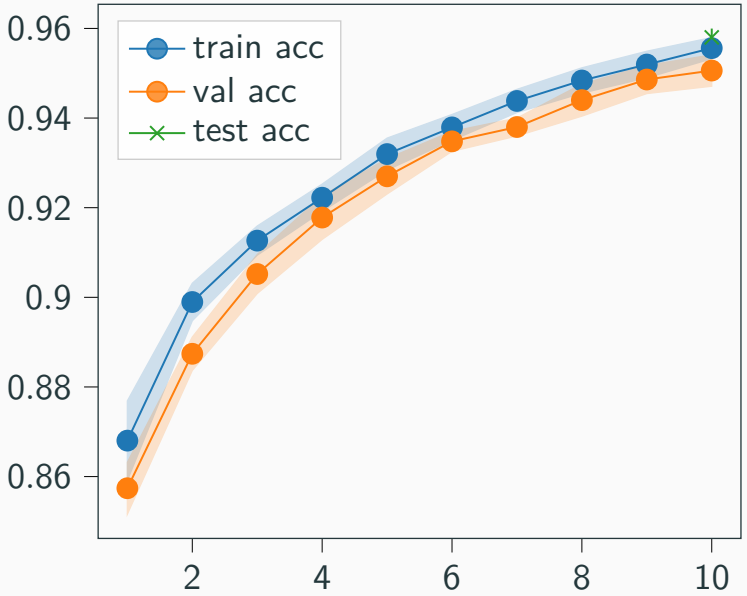
Pooling

Max pooling layers choose maximum values in predefined regions. Two by two max pooling, for example, picks the maximum in neighboring areas of four pixels. If an input is shifted by two pixels, the result will remain the same! Pooling layers are used repeatedly for a cumulative effect.

MNIST

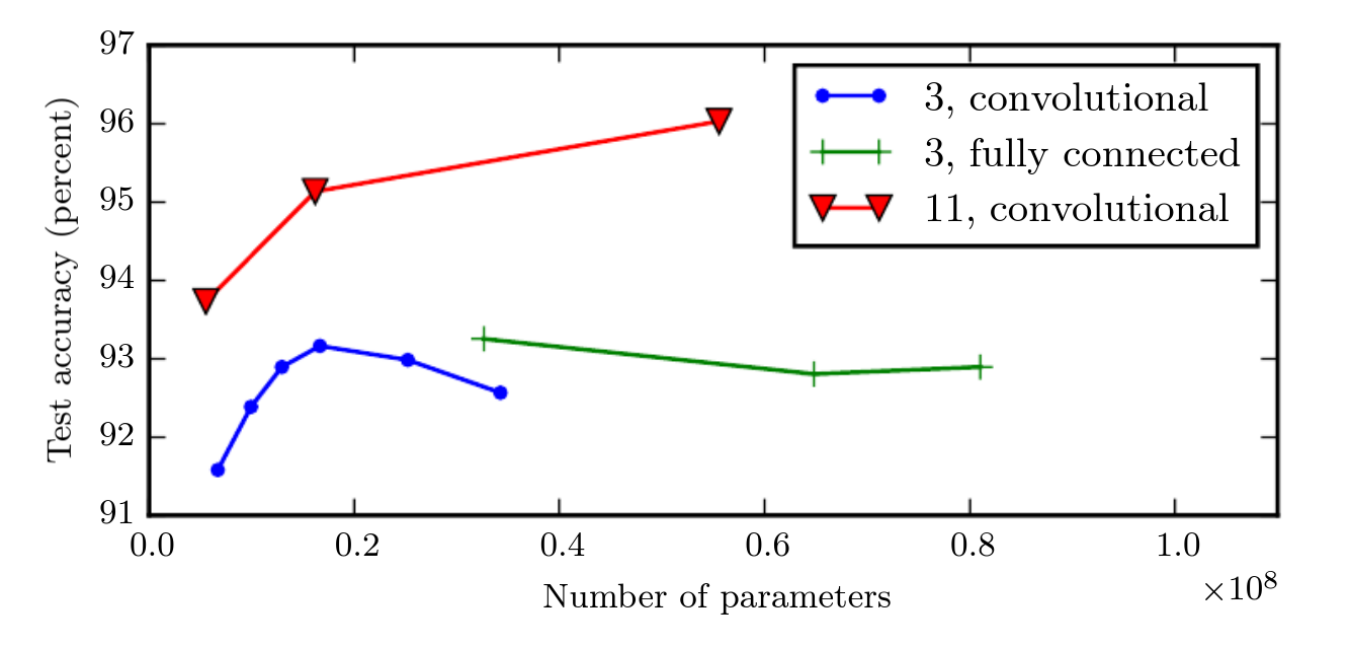


Sample digits from the MNIST-database.



Mean convergence of two-layer CNN with a dense classifier.

Deep convolutional neural networks



Comparing deep networks with and without convolutional structures on the Google-Street view dataset [GBC16, page 199].

References

[DV16] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning." In: *arXiv preprint arXiv:1603.07285* (2016).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Literature ii

[LeC+89] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. "Handwritten digit recognition with a back-propagation network." In: *Advances in neural information processing systems* 2 (1989).

[Stu20] David Stutz. *illustrating-convolutional-neural-networks*. https://davidstutz.de/illustrating-convolutional-neural-networks-in-latex-with-tikz/. Accessed: 2023-03-11. 2020.