MA-INF 4209 - Seminar Principles of Data mining and Learning Algorithms

Convolutional Networks

Chapter 9 (9.1-9.4)

Presenter: Ani Karapetyan

January 12, 2021

UNIVERSITÄT BONN

Convolutional Networks / Motivation

- **Convolutional networks (CNNs)** are neural networks that use **convolution** in place of general matrix multiplication in at least one of the layers.
- Convolutional models preserve the **spatial structure** of the input unlike fully-connected (FC) models which *flattens* the input.
- Dedicated for processing data with **grid-like** topology:
  - ➢ *Time-series data* - 1D grid of samples at regular time intervals.
  - ➢ *Image data* - 2D grid of pixels.
- Very popular in modern *Computer vision* and *NLP* applications.

The Convolution Operation

- ***Continuous case:*** Given *x(t)* and *w(t)* with *real-valued* arguments *(e.g. Fig. 1)*:

❖ $s(t) = (x * w)(t) = \int_{-\infty}^{+\infty} x(\tau)w(t-\tau)d\tau$

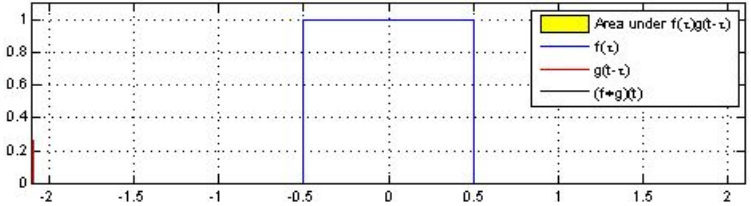


Fig. 1: Convolving a box signal with itself *[source]*

- ***Discrete case:*** Given *x(t)* and *w(t)* with *integer-valued* arguments:

❖ $s(t) = (x * w)(t) = \sum_{a=-\infty}^{+\infty} x(a)w(t-a)$

Feature map

Input

Kernel

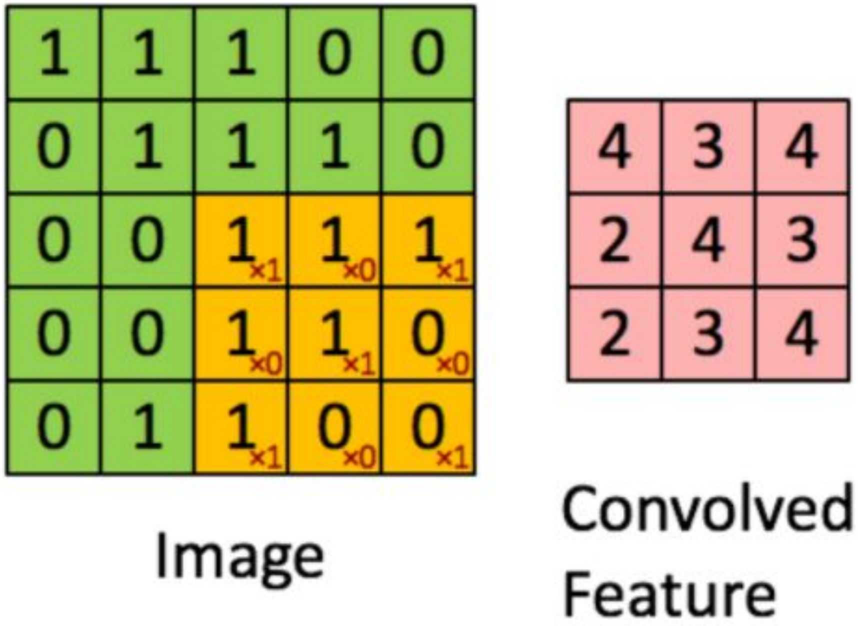
Finite sum in practice

The Convolution Operation

- *2D case* (convolving a 2D kernel *K* over an input image *I*):
  - $s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$
- Convolution is **commutative** (because of *flipped* kernel):
  - $s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$
- Cross-correlation:
  - $s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$

2D Convolution Example

Example: Convolution of an input image with a **2D** Kernel *(Fig. 2)*.



1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$

4	3	4
2	4	3
2	3	4

Fig. 2: Convolution with 3x3 kernel *[source]*

Famous Convolution Kernels

- Convolution is used for **feature extraction** in image processing.
  - ➢ Gaussian blur kernel: $\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$
  - ➢ Sharpen kernel: $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$
  - ➢ Right sobel kernel: $\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$

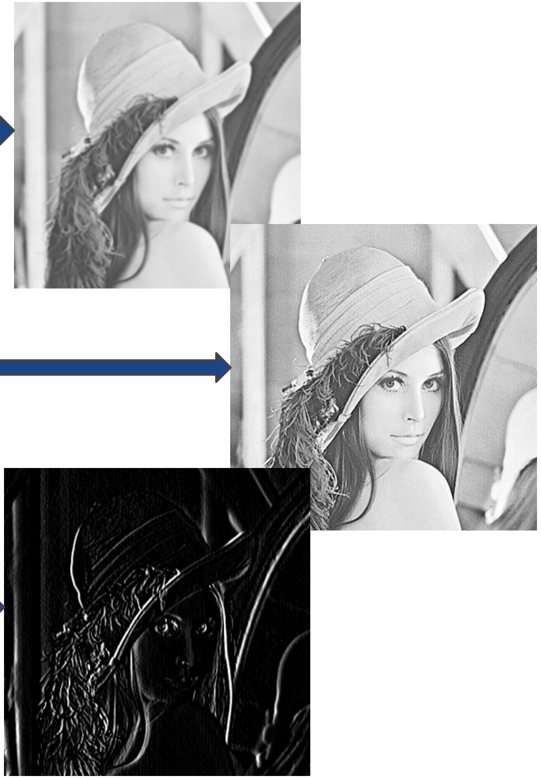


Fig. 3: Examples of image kernels *[source]*.

Convolution Implementation as Matrix Multiplication

Discrete convolution can be implemented as **matrix multiplication**:

➢ *Example (1D case):*

$$x = (1,2,3,4,5,6),\, k = (1,1,1) \quad x * k = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 6 \\ 9 \\ 12 \\ 15 \end{pmatrix}$$

Toeplitz matrix

➢ *Example (2D case):*

$$x = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix},\, k = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad x * k = \mathrm{reshape}\left(\begin{pmatrix} \mathbf{1} & \mathbf{2} & 0 & \mathbf{3} & \mathbf{4} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{2} & 0 & \mathbf{3} & \mathbf{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{2} & 0 & \mathbf{3} & \mathbf{4} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{2} & 0 & \mathbf{3} & \mathbf{4} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, (2,2) \right)$$

Doubly-block Toeplitz matrix

Convolutional layer structure

A typical convolutional layer consists of *3 stages*:

- ➢ **Convolution** of input with several filters in parallel.
- ➢ Non-linear activation function (e.g. **ReLU**).
- ➢ **Pooling** (max, average, L2 norm, etc).

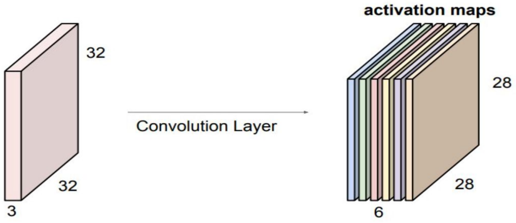


Fig. 5: Convolution with 6 5x5 filters *[source]*.

Convolution stage:

- ***Input:*** 3D tensor of shape W x H x C.
- ***Parameters:*** weights of 3D learnable *filters.*
- ***Output:*** set of 2D *activation maps* (activ. map per filter) stacked together along the depth dimension.

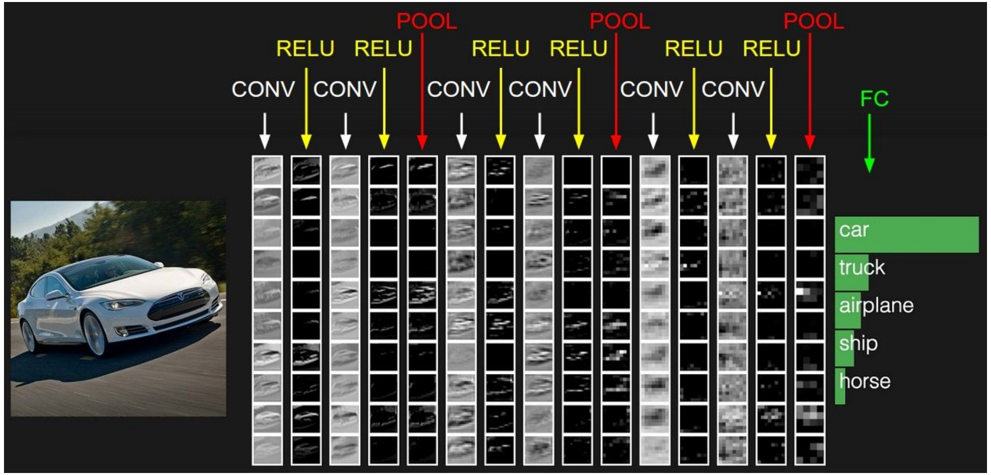


Fig. 4: General CNN architecture for object recognition *[source]*.

Convolutional layer properties

- Convolution introduces 3 key advantages over fully connected layers:
  - ➢ **Sparse interactions**
  - ➢ **Parameter sharing**
  - ➢ **Translation equivariance**

- Convolution allows to work with **inputs of variable size.**
  - ➢ Vary the stride of convolution or pooling or use global pooling.
  - ➢ Scales well to large input sizes.

Convolution / Sparse interactions

- In FC layer, every output unit interacts with every input unit *(Fig. 6, below diagram)*:
  - ➢ **m*n** parameters, **O(m*n)** runtime!
    - ❖ m - number of input neurons
    - ❖ n - number of output neurons
- Convolutional layers have **sparse connectivity** *(Fig. 6, above diagram)*, by making the kernel much *smaller* than the input:
  - ➢ **k*n** parameters, **O(k*n)** runtime!
    - ❖ k - kernel size
    - ❖ n - number of output neurons

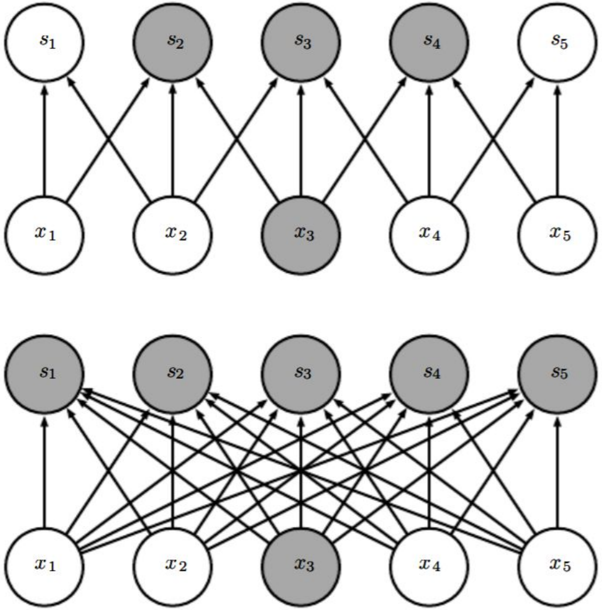


Fig. 6: Sparse connectivity *[1, Fig. 9.2]*

Convolution / Sparse interactions

- CNNs build complex interactions between different units by stacking simple building blocks of sparse connectivity.
- Units in deeper layers *indirectly* interact with larger parts of input.
  - ➢ Larger **receptive field** *(Fig. 7)*.

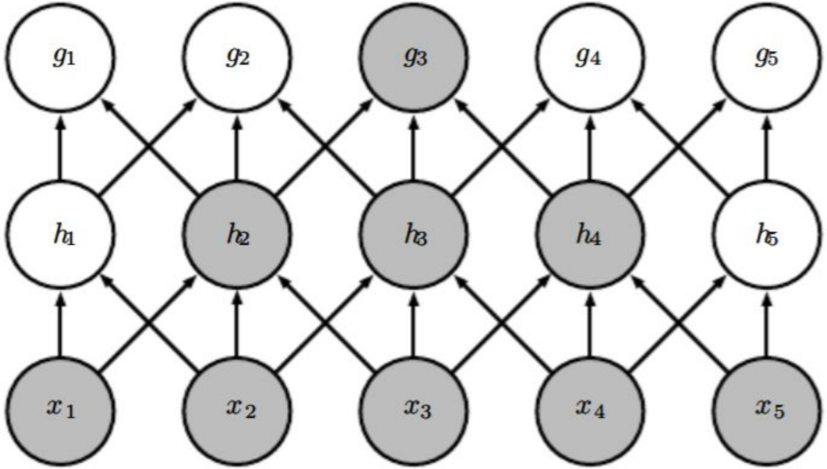


Fig. 7: Receptive field *[1, Fig. 9.4]*

Convolution / Parameter Sharing

Convolutional layers have **tied weights** - kernel weights are shared across the input *(Fig. 8)*.

- ➢ **k** parameters (**k << m*n**) => reduced memory!
- ➢ **O(k*n)** runtime *(as before)*.

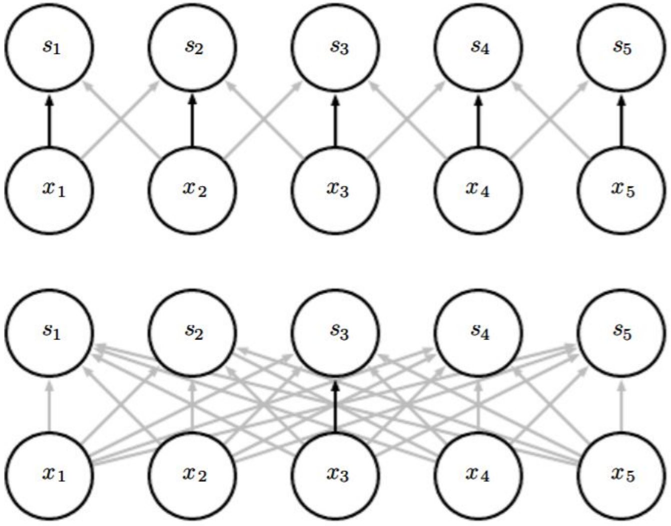


Fig. 8: Parameter sharing *[1, Fig. 9.5]*

Convolution / Parameter Sharing

Example: Efficiency of edge detection *(Fig. 9)*.

- **K = [1, -1]** - vertical edge detection kernel.
- Input size: **280x320.**
- Output size: **280x319.**

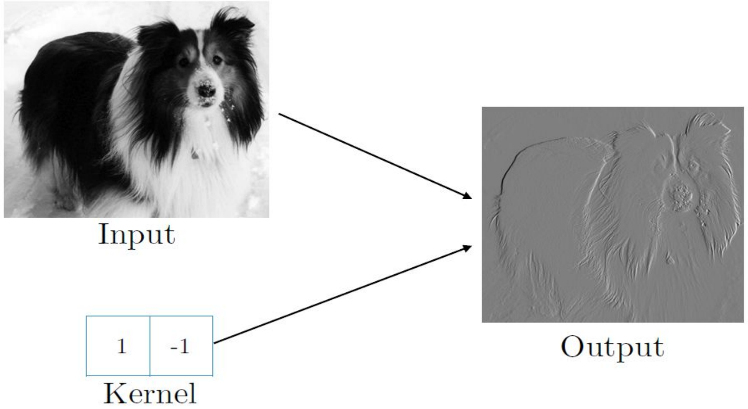


Fig. 9: Vertical edge detection by convolution *[1, Fig. 9.6]*

	Convolution	*Fully connected (full matrix multiplication)*
Number of stored weights (floats)	2	319*280*320*280 > 8e9
Floating point operations (addition, multiplication)	319*280*3 = 267960	319*280*320*280*2 > 16e9

Convolution / Translation Equivariance

- Convolution is **equivariant to translation**!
  - ➢ **F(T(Im)) = T(F(Im))**
    - ❖ T - translates the input by one pixel to the right: *T(Im)(x, y) = Im(x-1, y)*
    - ❖ F - convolution operation in this case.

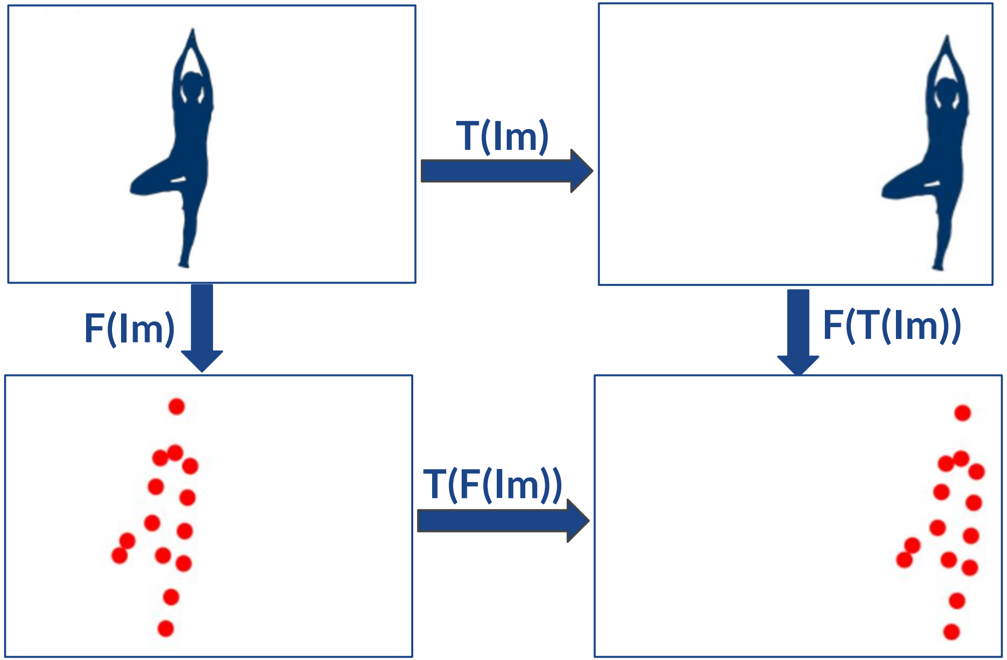
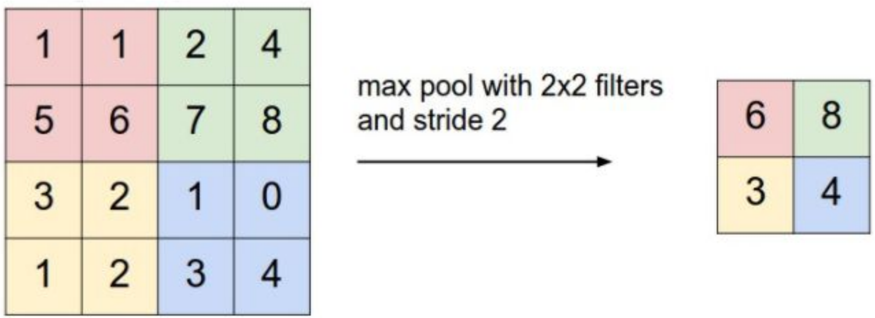


Fig. 10: Illustration of translation equivariance property *[source]*.

- Convolution is **not** equivariant to image **scaling** and **rotation**!

Pooling

- Combines the output of several neighboring units (from a rectangular region) into a **summary statistic** at that location.
  - ➢ Max *(Fig. 11)*, Average, L2 norm.
  - ➢ Weighted average based on distance from central pixel.
- Can progressively **reduce spatial size** of the input *(downsampling)*.
  - ➢ Reduced number of *parameters* and computation.
  - ➢ *Overfitting* control.



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

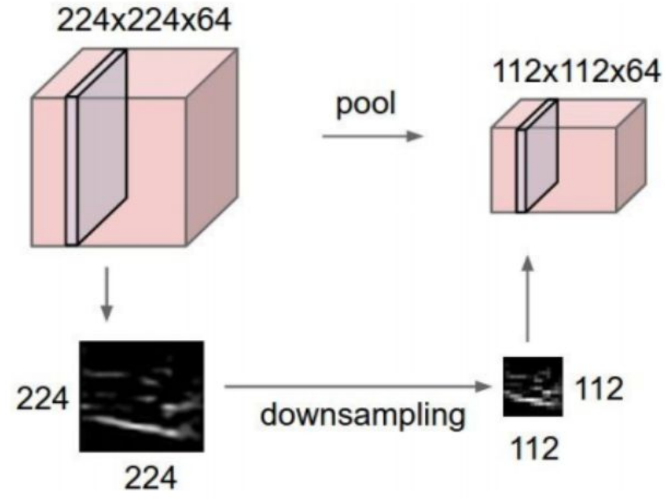


Fig. 11: Max pooling with 2x2 filter *[source]*.

Pooling

- Pooling achieves approximate **invariance** to *small* **translations** *(Fig. 12)*!
  - ➢ Useful if we care more about the presence of a specific feature than its exact location.

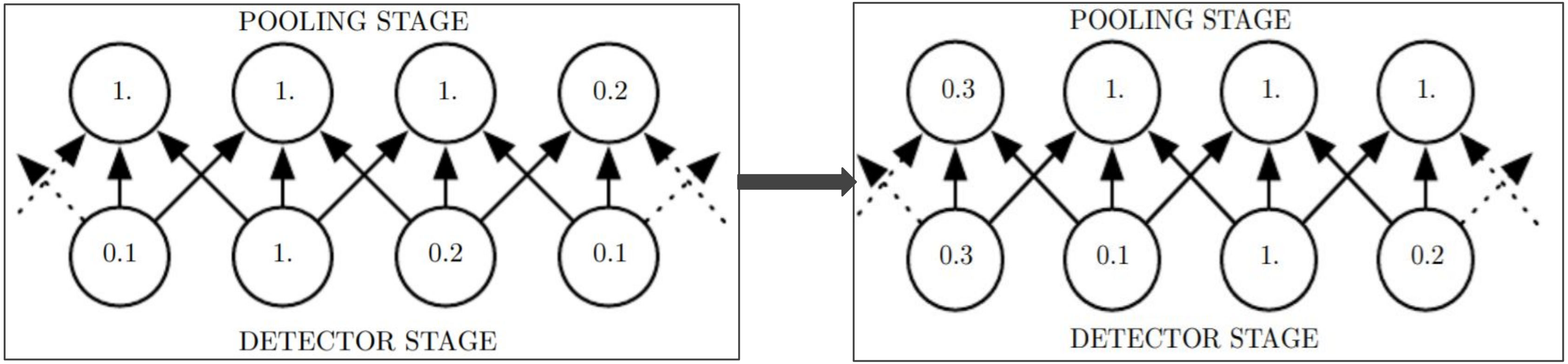


Fig. 12: Max pooling introduces invariance. *[1, Fig. 9.8]*

Pooling

- Pooling units over different *channels* of convolution output (*with multiple filters*) can learn to become invariant to different transformations other than simple translation.
  - ➢ *Example*: Learnt **invariance** to *small* **rotations** (*Fig. 13*).

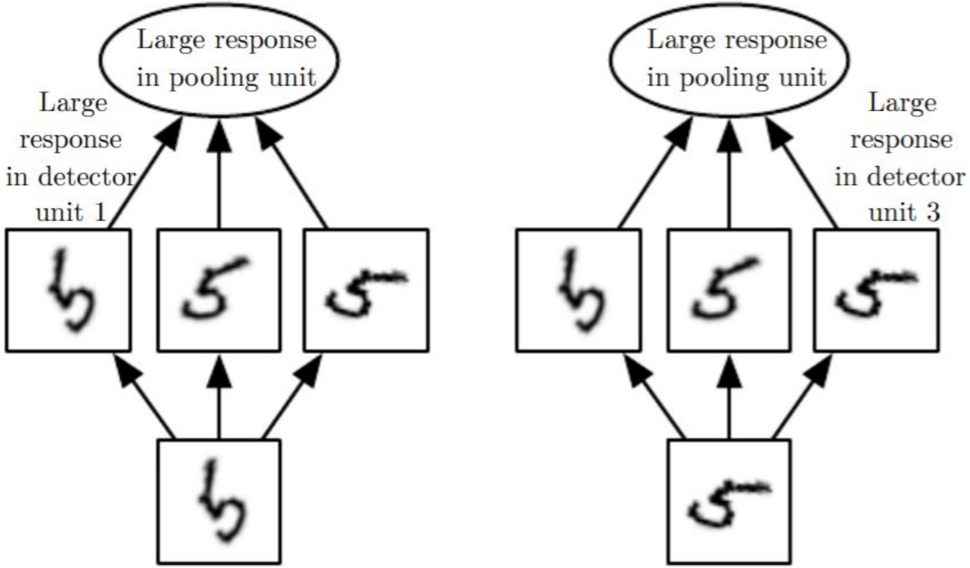

Large response in pooling unit
Large response in detector unit 1
Large response in pooling unit
Large response in detector unit 3

Fig. 13: Example of learned invariances. *[1, Fig. 9.9]*

Pooling

- Pooling is useful in handling **inputs of variable size**.
  - ➢ Adapt the **pooling stride** s.t. the FC layer (e.g. classification layer) always gets the same number of summary statistics.
  - ➢ Use **global pooling** before FC layer to transform a tensor of shape W x H x C to tensor of shape 1 x 1 x C, regardless of the input size.
- Pooling **extensions**:
  - ➢ *Cluster* features in order to make pooling local in the configuration space [2].
  - ➢ Learn a single pooling pattern *(adaptive receptive field)* and apply it to all images [3].

Convolution and Pooling as Infinitely Strong Prior

- **Strength of the prior** depends on how concentrated the probability density is.
  - ➢ How much evidence the model needs to see until it deviates from the prior assumptions about its parameters.
- Convolutional net is a FC net with an **infinitely strong prior** on its weights!
  - ➢ Most of the weights in the matrix are constrained to be 0.
  - ➢ Weights for one hidden unit must be the same as the weights of its neighbor shifted in space.
- Pooling puts an **infinitely strong prior** that each unit should be invariant to small translations!

Convolution and Pooling as Infinitely Strong Prior

- Convolution and pooling can cause **underfitting**!
  - ➢ If precise spatial information is relevant, *pooling* can increase training error.
  - ➢ When it's important to incorporate features from distant locations in the input, *convolution* may be inappropriate.

- We should only *compare* convolutional models to other convolutional models!
  - ➢ Non-convolutional models can learn even if we **permute** all the pixels in the image - they learn the concept of topology via training.

References

[1] I. Goodfellow, Y. Bengio and A. Courville: Deep Learning, MIT Press, 2016.
[2] Y. Boureau et al: Ask the locals: multi-way local pooling for image recognition, 2011.
[3] Y. Jia et al: Beyond spatial pyramids: Receptive field learning for pooled image features, 2012.
[4] https://cs231n.github.io/convolutional-networks
[5] https://towardsdatascience.com/how-are-convolutions-actually-performed-under-the-hood-226523ce7fbf

Convolutional Networks

Seminar Principles of Data Mining and Learning Algorithms
University Bonn

Nils Becker

January 12, 2021

Variants of the Basic Convolution Function

In practice, the used convolution differs from the usual definition slightly.

- many convolutions in parallel to extract many features
- multichannel convolution
  - e.g. images with channels for red, green, blue
  - leads to 3-D tensors
- skipping positions to reduce the computational cost

Multichannel Convolution

We can think about the input, not as a grid, with real values rather than a grid of vectors.

Definition

Let Z be a 3-D tensor, let V be our input with the same shape like Z and K our kernel, a 4-D tensor. The value of an output unit $Z_{i,j,k}$, with i=channel, j=row, k=column is given by:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l-1,j+m-1,k+n-1} K_{i,l,m,n}$$

where l, m, n are values for which the tensor indexing is valid.

For simplicity, the above formula requires that Z and V has the same shape. In general, this is not required, we can have more or less channels in the output than on the input.

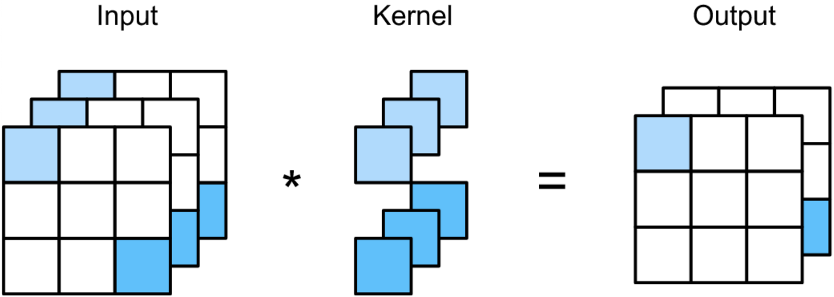


Figure: Example for 1x1 Cross-Correlation with three input channels and two output channels. Each output channel has its own kernel, such that the overall kernel is a 4-D Tensor [1].

Strided Convolution

When using the strided convolution, some positions are skipped. This leads to a reduced amount of computation. A strided convolution equals a normal one followed by downsampling.

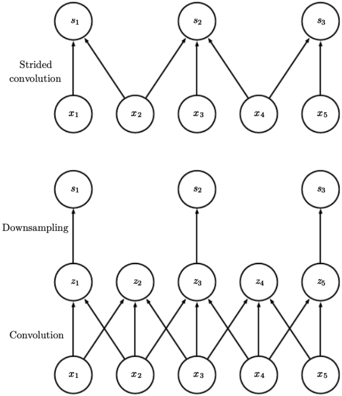


Figure: Example for a strided convolution of size 2 [2, P. 350].

Zero Padding

Convolution shrinks the size of the output by one less than the kernel width. This limits the number of possible layers. To avoid this, we can fill up the input with zeros.

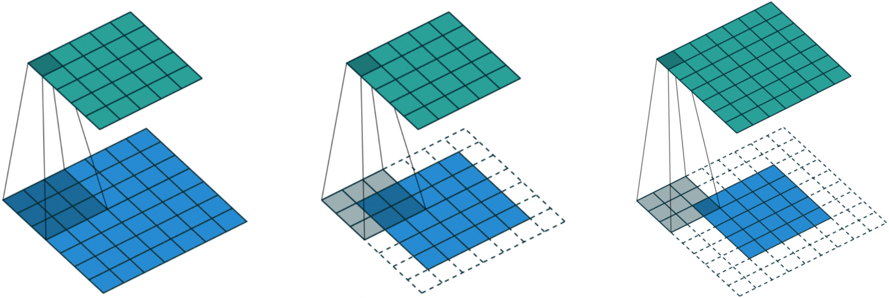


Figure: Left: Valid convolution. Middle: Same convolution. Right: Full convolution [3]

Comparison of Convolutions

Valid Convolution / no zero padding:

- shrinks output size.
- Input pixel influences output equally.

Same Convolution:

- keeps size
- arbitrary layers
- Border pixels influence less output pixels.
  - Border pixels underrepresented.

Full Convolution:

- Pixels are visited equally often.
- Output pixels near the border are a function of fewer pixels.
  - Difficult to perform well at all positions.

Optimal size lies between same and valid convolution.

Unshared Convolution

Unshared convolution

- consists of locally connected layers, where the parameters are not shared.
- can be regarded as that each output pixel has its own kernel.
- is useful when the occurrence of a pattern is restricted in space.

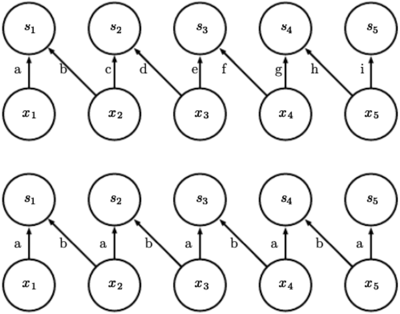


Figure: Comparison between unshared (Top) and normal convolution (Bottom) with a kernel width of two [2].

Tiled Convolution

Tiled Convolution

- is a compromise between unshared and shared convolution.
- uses multiple kernel, which rotate trough when moving through space.
- has different kernels for neighboring locations.
- memory only increases with number of kernels.

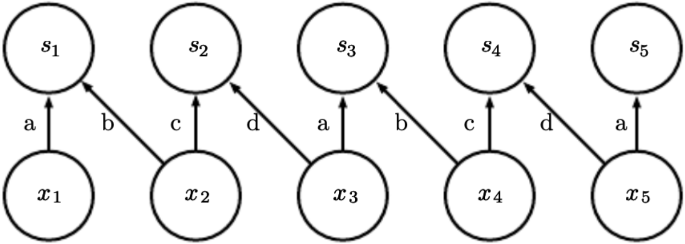


Figure: Example for tiled convolution with two kernels (a,b and c,d) with width of two [2].

Structured Outputs

- Convolutional Networks can output a structured object.
  - represented by a multidimensional tensor

  Example: Label each pixel of an image to a certain class. The convolutional network can output a tensor where each pixel has a vector of probabilities.

Problem: The network shrinks the image. We need to upsample it again.

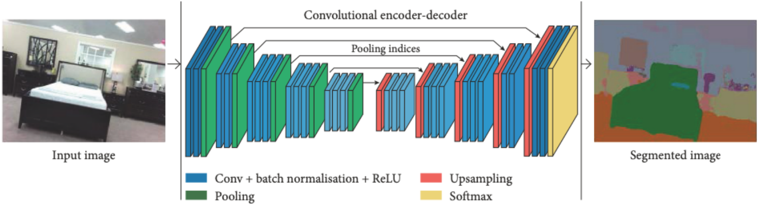


Figure: Example of shrinking input by convolutional layers [4].

Working with shrinked output

Solutions:

- avoid pooling
- pooling only with unit stride
- use low resolution
- refine initial guess with recurrent network

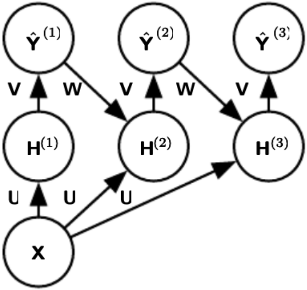


Figure: Stepwise refinement of an image using a recurrent network. In the initial step $\hat{Y}$ is zero [2, P. 359]

Data Types

- Convolutional networks can can handle data with varying size.
- Kernel is applied according to the size of the input.
    - Output scales with input.

But, what is, when we need a fixed size of the output?

- Insert a pooling layer, which scales proportional to input size.

Usages for CNNs

Because of the useful properties of convolutional networks, we can use them on:

Single channel

- audio files
- audio data in frequency domain
- volumetric data / CT scans

Multi channel

- animation data (channels representing angles of joints)
- colored images
- colored videos

Efficient Convolutional Algorithms

Pointwise convolution on millions of units is computational intense.

- Regard input as signals.
- Use Fourier transformation. ⇒ Convolution becomes multiplication.
- Transform result back.

If a d-dimensional kernel is separable (kernel can be expressed as product of d vectors), we can do d one-dimensional convolutions. Let w be the kernel width in each dimension, we get following runtime and memory consumption:

- naive approach: $O(w^d)$
- separable kernel: $O(w * d)$

Summary

- Kinds of convolution
  - Multichannel Convolution
  - Strided Convolution
  - Unshared Convoluton
- Zero Padding
  - Valid Convolution
  - Same Convolution
  - Full Convolution
- Structured outputs
- Data types and usage of CNNs
- Efficient Convolutional Algorithms

References

[1] https://d2l.ai/chapter_convolutional-neural-networks/channels.html.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning.
MIT Press, 2016.
http://www.deeplearningbook.org.

[3] Vincent Dumoulin and Francesco Visin.
A guide to convolution arithmetic for deep learning, 2018.

[4] Cesare Valenti, Bijen Khagi, and Goo-Rak Kwon.
Pixel-label-based segmentation of cross-sectional brain mri using simplified segnet architecture-based cnn.
Journal of Healthcare Engineering, 2018:3640705, 2018.

Appendix

$$\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$$

Figure: Example for a separable kernel. Here: gaussian filter used in image editing. (https://de.wikipedia.org/wiki/Separierbarkeit)