LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Deep Learning for Physicists

Lecture #5: Convolutional neural networks | Part 1

Kosmas Kepesidis

Previous lecture…

- Fully-connected networks as a basic network architecture
- Fully-connected networks can be used for both regression and classification tasks
- Reviewed related objective functions
- Challenges of training of deep neural networks
- Methods of network stabilization

About the course

- Course content:
  - Architectures of deep neural networks
    - Fully-connected neural networks
    - **Convolutional neural networks (CNNs)**
    - Recurrent neural networks (RNNs)
    - Graph networks
    - Hybrid architectures

Outline

- **Convolutional neural networks (CNNs) | Part 1**
  - Convolutions of image-like data
  - Convolutional layers
  - Multi-dimensional convolutions
  - Important operations in CNNs
  - Short- and long-range correlations
  - CNNs vs. fully-connected networks

Convolutional neural networks (CNNs) | Part 2
  - Reconstruction tasks
  - Advanced concepts
  - Applications in physics

Convolutional neural networks (CNNs)
Part 1

Convolutional neural networks (CNNs)

- For data with inherent symmetries, there exist dedicated network architectures that are superior to the basic fully-connected networks

- We focus our attention on data that have a particular structure: 2D grid-like data (such as pixels in an image)
  - Refer to this structure as **spatial domain** in contrast to **feature space**
  - **Features space** refers to higher-level patterns that can exist on top of the spatial domain

Convolutional neural networks (CNNs)

- Interpretation is detached from the spatial domain: **semantic meaning**
- Why are fully-connected neural networks not successful in such a context?
  - Input would consist of all pixels of the image
  - The applied transformation that brings information from one layer to the next will imply that each pixel will be correlated to every other: resulting to an enormous number of parameters to be adapted
  - We expect similar **semantic meaning** to populate neighboring areas on the image
- CNNs originated in the 1990s, developed based on the concept of **translation invariance**
- Although old concept, breakthrough began in 2012 (ImageNet)
- Now CNNs are one of the most important architectures, driving innovation in deep learning and computer vision

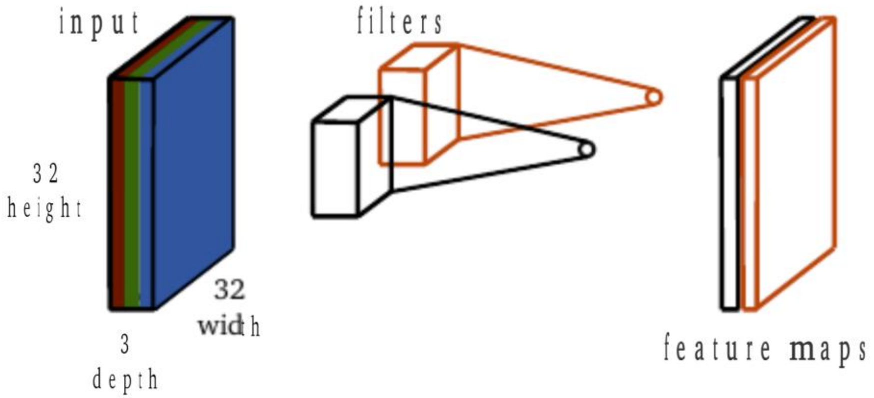
Y. Lecun et al., Gradient-based learning applied to document recognition, Proceedings of the IEEE 86, 11 (1998)
A. Krizhevsky et al., ImageNet classification with deep convolutional neural networks (2012)

Convolutions of image-like data

- CNNs analyzes data in segments and subsequently merge information
  - ➢ Process only local groups of pixels of an image, then combine the extracted features

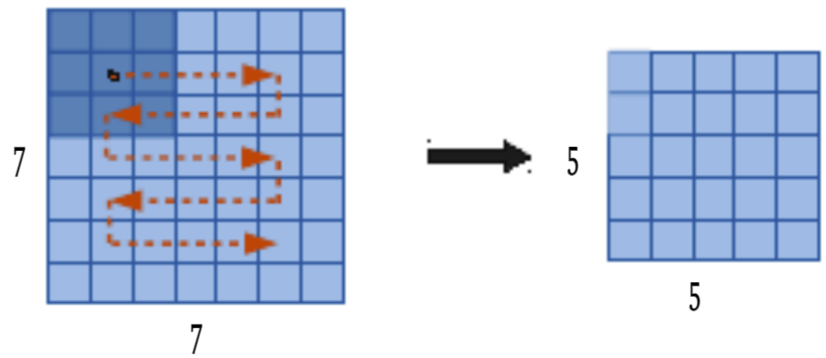
- This approach is implemented using **sliding filters** over an image

- Example: convolution using two filters for an image with three color channels (RGB=red, green, blue)



Convolutions of image-like data

- Each filter moves over the image and at each position, pixel values are weighted with the respective filter weights
- Resulting values are added up over the neighborhood – this results into a single value per filter position
- Assume single-channel image – transformation similar to that of fully-connected network:



$$x_i' = \sum_{x_j \in N_i} W_i \cdot x_j + b$$

➢ x_j runs over all pixels in the neighborhood N_i of pixel i
➢ W_i are (as usual) the adaptable weights and b the bias

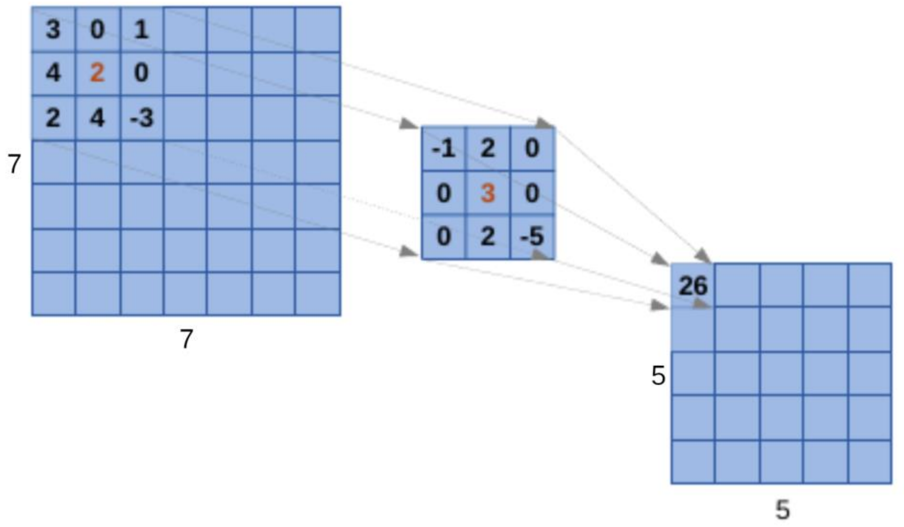
Convolutions of image-like data

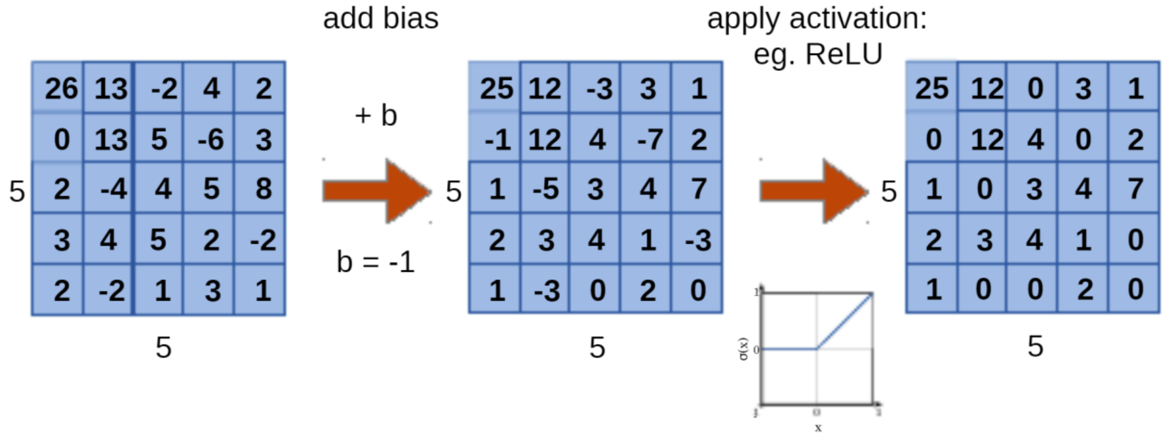
- In general, x_j and W_j are vectors accounting for multiple channels
  - For an RGB input, the filters have a depth of 3 and act on all input channels simultaneously

- Just as for fully-connected nets, a nonlinearity is applied using activation function: $y_i' = \sigma(x_i')$

- During a convolutional operation:
  - the information at a given filter position is aggregated over its neighborhood
  - Result is a single value by performing:
    1. linear transformation, i.e. summing over the neighborhood
    2. applying a nonlinearity

Convolutions of image-like data



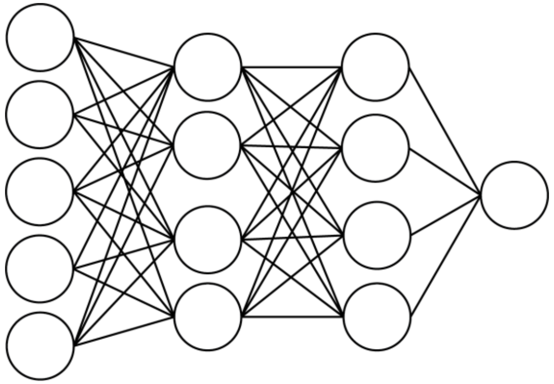


Convolutions of image-like data

- **Translational invariance**:
  - Parameters W_j and b do not depend on the position of the filter (no i-dependence)
    - $x_i = \sum_{x_j \in N_i} W_i \cdot x_j + b$
  - Same parameters applied all over the grid: **weight sharing**
  - Convolutional operation is **translational invariant**
  - Weight sharing also leads to lower # of parameters to be adapted

- **Feature mapping**: result of single-filter convolution operation
  - Feature map preserves “image-like” structure of data
  - Generate as many feature maps as filters: adapting the amount of retained information

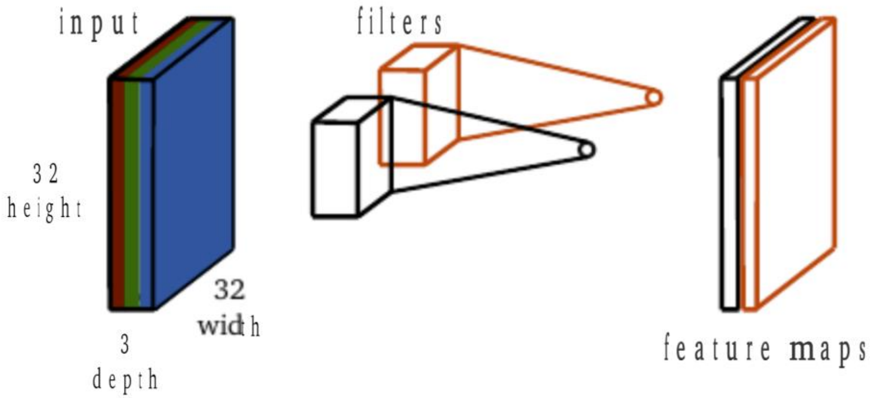
Convolutions of image-like data



- Thought example: weight sharing
  - ➢ Consider single filter of size equal to the size of the entire image
  - ➢ Each pixel is linked to every other pixel in the image and is scaled with an individual weight – so, no weight sharing possible
  - ➢ Result is a fully-connected layer with a single node
  - ➢ More filters would correspond to more nodes on a fully-connected layer
  - ➢ Thus, a fully-connected layer can be expressed using convolutional filters of maximum size
  - ➢ Moderate filter sizes are essential for exploiting spatial symmetry

Convolutional layers

- Feature maps produced by a convolution can be input to subsequent convolutions
- A set of convolutions, using n_f filters of given filter size and activation function σ, defines a **convolutional layer**
- Convolutional layers are building block of convolutional networks



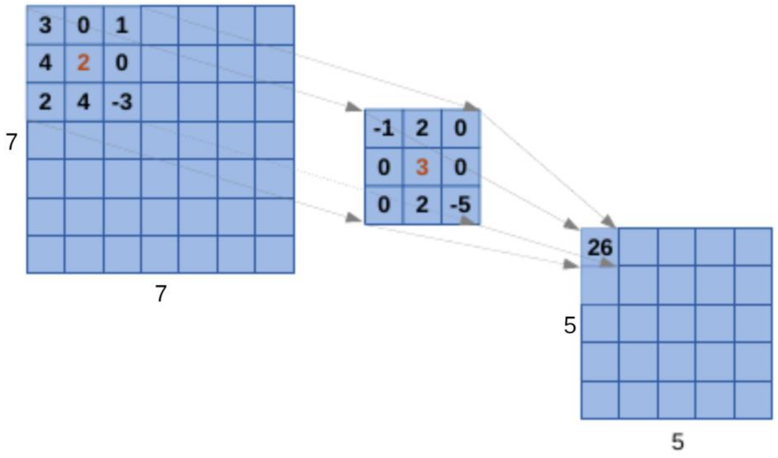
Convolutional layers

- Number of parameters of CNNs determines their efficiency
- As filter slides over the pixels of an image, each channel in the filter has its own set of parameters:

$$n_T = n_c \cdot n_w \cdot n_h \cdot n_f + n_f$$

  - n_T: total number of parameters per layer
  - n_c: number of channels
  - n_w: width
  - n_h: height of filter
  - n_f: total number of filters (and feature maps)

Convolutional layers

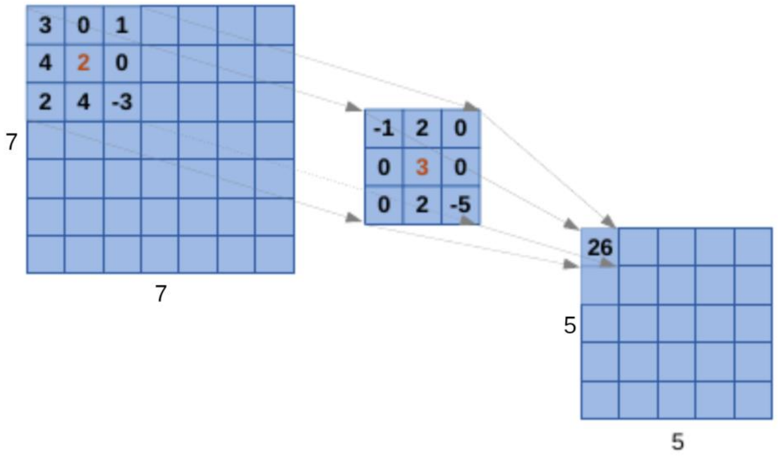


- Comprehensive example: parameter counting
  - Sliding two 3x3 filter over an RBG image
  - At each pixel, information of this pixel and the 8 neighboring ones is collected, for all three color channels
  - This information per pixel is multiplied by adaptive weights and collected into a single response
  - Subsequently, the nonlinearity is applied to the response

$$Y'_{i,j,f} = \sigma\left(\sum_{k=-1}^{1}\sum_{l=-1}^{1}\sum_{c=1}^{3} W_{k,l,c,f}\, X_{i+k,j+l,c} + b_f\right)$$

Convolutional layers

- Comprehensive example: parameter counting



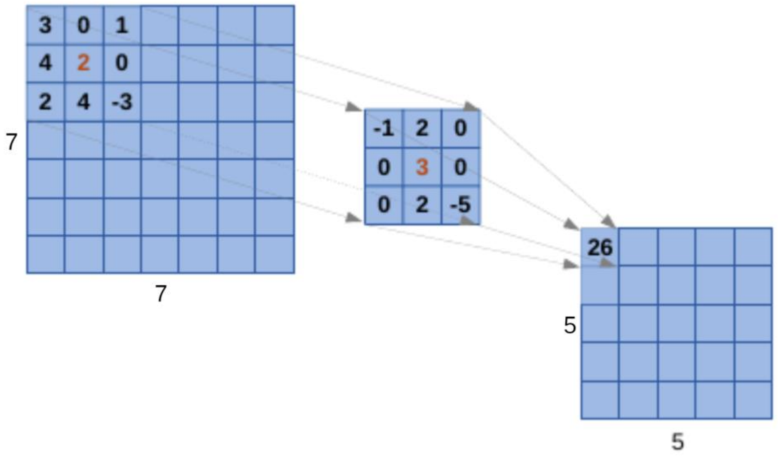
$$Y'_{i,j,f} = \sigma\left(\sum_{k=-1}^{1}\sum_{l=-1}^{1}\sum_{c=1}^{3} W_{k,l,c,f}\, X_{i+k,j+l,c} + b_f\right)$$

- (i, j): position of input (central) and output pixel
- c: color channel
- f: filter index
- k, l: indices for aggregation over neighboring pixels

- So, how many parameters?

Convolutional layers

- Comprehensive example: parameter counting



$$Y'_{i,j,f} = \sigma\left(\sum_{k=-1}^{1}\sum_{l=-1}^{1}\sum_{c=1}^{3} W_{k,l,c,f}\, X_{i+k,j+l,c} + b_f\right)$$

- So, how many parameters?
  - 3 x 3 x 3 = 27 adaptive weights per filter
  - 27 + 1 = 28, due to bias term
  - For two filters: 56 total parameters!

- In contrast, for a fully-connected layer, we would have:
  - (32 x 32 x 3 + 1) = 3073!

Multi-dimensional convolutions

- Concept of convolutions can be extended to N-dimensions:
  - N-dimensional filters scan N-dimensional grid and produce N-dimensional feature maps
  - However, few Deep Learning frameworks can support convolutions in more than 3-4 dimensions

- Interesting fact:
  - Important symmetries or correlations between certain dimensions might exist
  - To help the model take advantage of possible symmetries or extract important correlations, one should carefully choose filter sizes and share weights over specific dimensions

Multi-dimensional convolutions

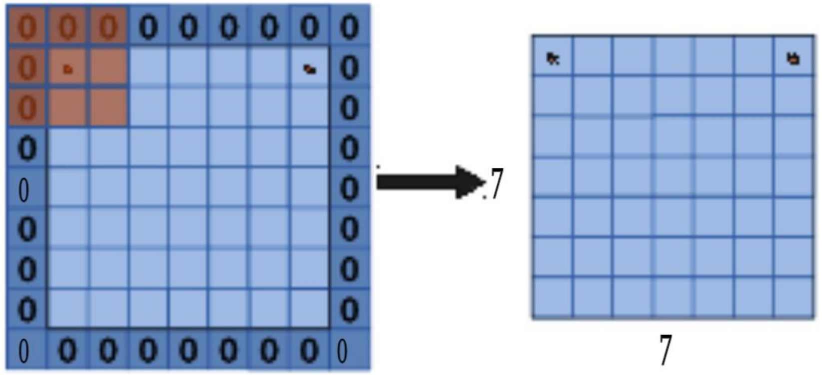
- Example: capturing symmetries
  - Assume a regular 2D grid of sensors that measure signal traces: $A_{x,y}(t)$
  - The traces of each detector might show similar patterns – could be useful to have parameter sharing along spatial dimensions
  - Appropriate filter would be 3-dimensional: (n_x, n_y, n_t)

Important operations in CNNs

- **Zero padding**:

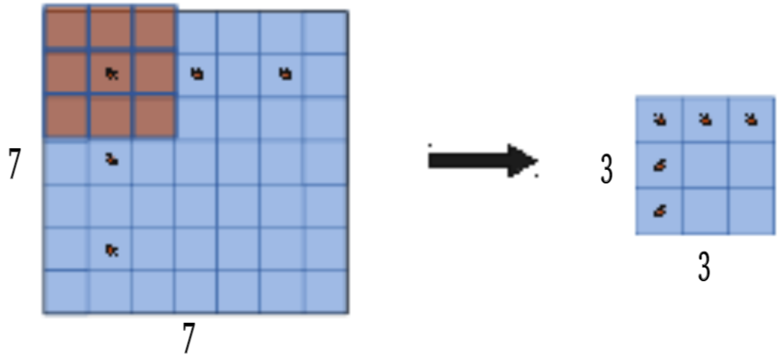
  - For being able to apply filters on the pixels at the edges of an image, as well as for preserving the initial resolution, **zero padding** is used

  - i.e. a frame of zeros is added around the image



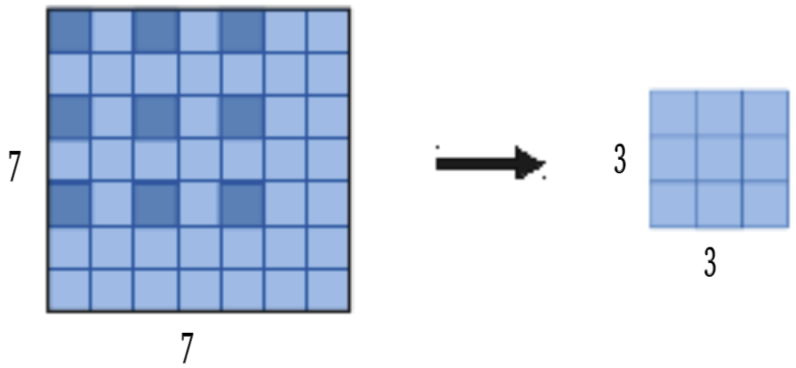
Important operations in CNNs

- **Striding**:
  - ➢ Feature maps' size can be controlled by increasing the size of steps between subsequent applications of a filter
  - ➢ It can be of benefit in cases of large-images to apply the filter every n steps
  - ➢ This is known as **strided convolution**



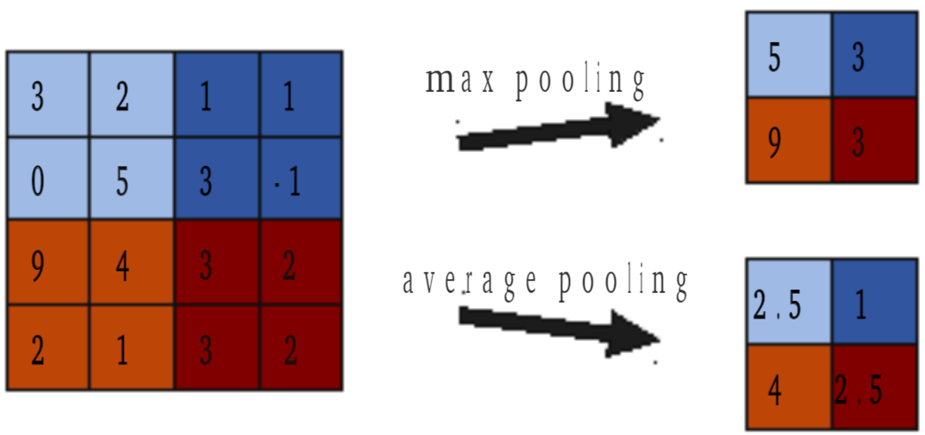
Important operations in CNNs

- **Dilating**:
  - ➢ In another variant, called **dilating**, the filter has gaps that rapidly reduce the image size
  - ➢ Typically used when a large receptive field of view is needed within a few layers
  - ➢ Size of gaps inside the filter is another hyperparameter to be tuned



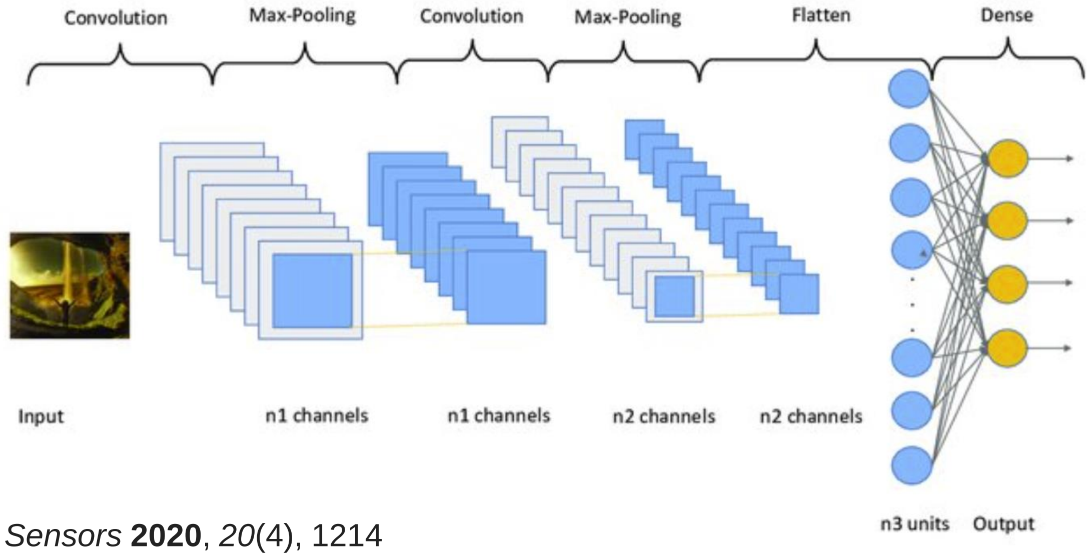
Important operations in CNNs

- **Pooling**:
    - ➢ Instead of modifying the convolutional operation (filter), **pooling** is a way of down-sampling
    - ➢ Aggregate information of neighborhood using a patch of size $a \times b$
    - ➢ Two flavors: **max pooling**, **average pooling**
    - ➢ Extension to the entire feature map: **global pooling**
        - ➢ entire feature maps are reduced to a single value
        - ➢ This technique reduces the number of outputs dramatically
        - ➢ It is usually used in the end of the architecture



Important operations in CNNs

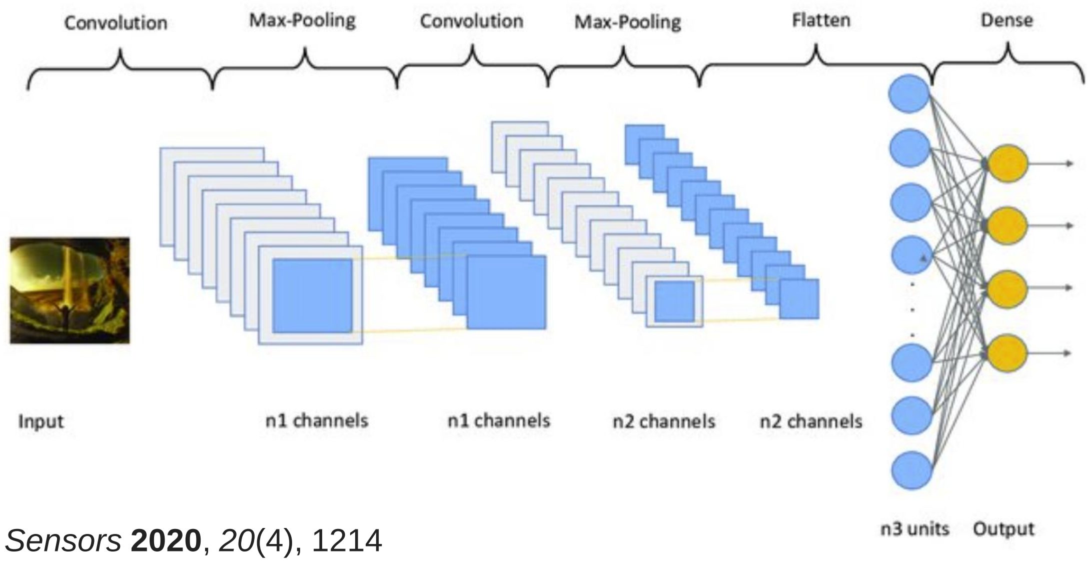
- Adding fully-connected layer:
  - ➢ After a few convolutional and pooling layers, sufficient number of features have been extracted
  - ➢ These features can be used a the input of a fully-connected layer
  - ➢ The fully-connected layer learns an extra mapping and is responsible for making the final predictions



Sensors **2020**, *20*(4), 1214

Important operations in CNNs

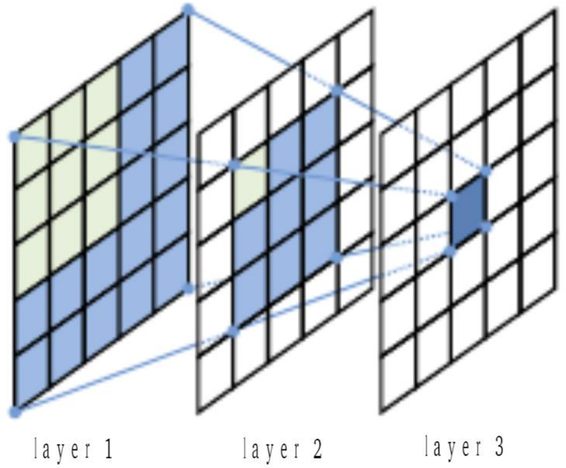
- Adding fully-connected layer:
  - ➢ The number of weights for fully-connected layers scale with the number of inputs
  - ➢ If feature maps are of high resolution, application of the fully-connected layer will be prone to overfitting
  - ➢ Appropriate reduction of the image size via regularization or global-pooling helps overcoming this issue



Sensors **2020**, *20*(4), 1214

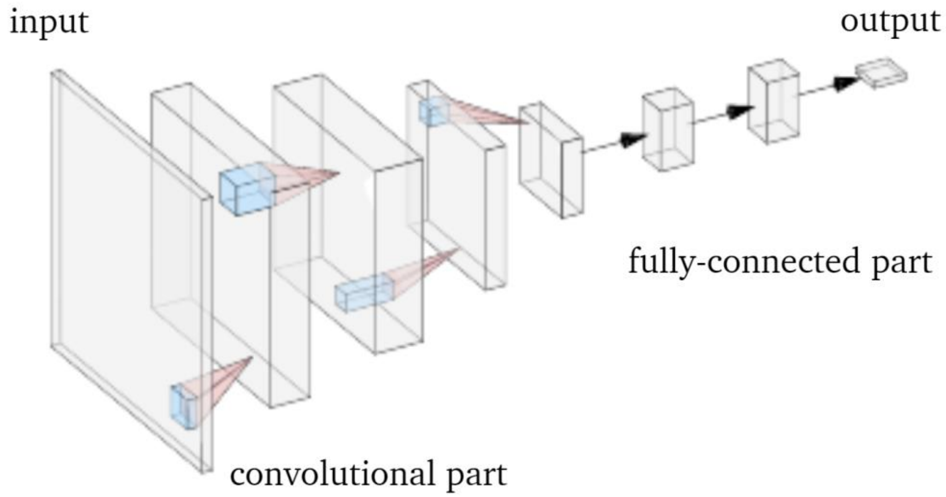
Important operations in CNNs

- Short- and long-range correlations
  - ➢ In CNNs, complex operations are divided into simpler layers of lower complexity: **hierarchy learning**
  - ➢ In a single convolutional layer, input pixels (or features) are linked to their immediate neighbors – how can long-range correlations be represented?
  - ➢ For subsequent layers, convolutional operations are applied to the responses (features) of the previous layer
  - ➢ In this way, the region where correlations can be exploited is increasing
  - ➢ In computer vision, this is called **receptive field of view** – what is the largest area in which correlations can be exploited



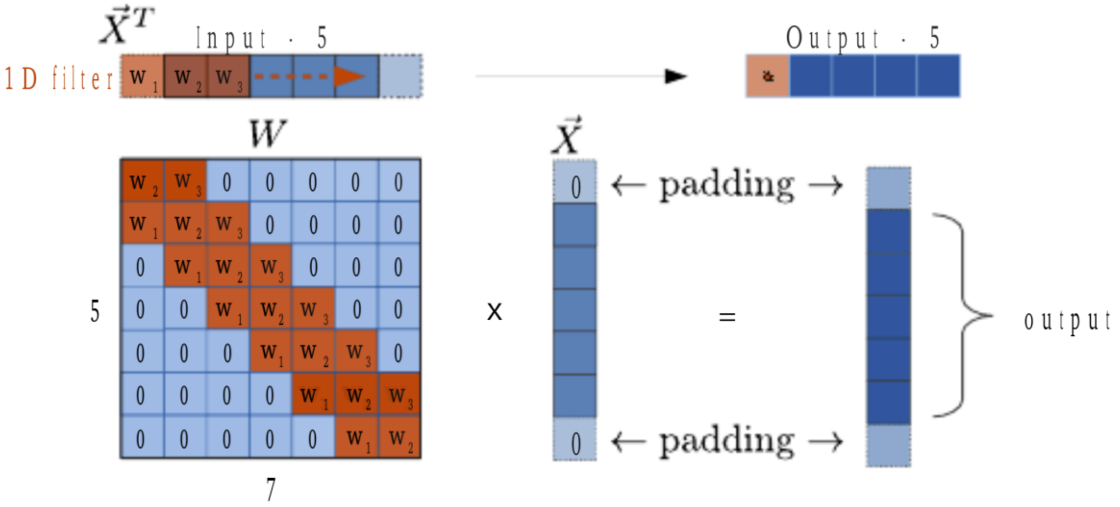
Important operations in CNNs

- Short- and long-range correlations
  - ➢ The constantly increasing **receptive field of view** with increasing the number of layers of CNNs is a demonstration of the concept of **hierarchy learning**
  - ➢ In the first layers, short-range correlations are exploited and lead to more global features
  - ➢ Long-range correlations and more detailed and task-specific features are learned in later layers



CNNs vs. fully-connected networks

- Assume 1D convolution and input data with regular structure:
  - ➢ time traces $A(t)$ measured in fixed intervals Δt
- The filter has length of 3 and is applied 5 times using zero padding
- The shape of weight matrix $\mathbf{W}$ for the linear mapping of a fully-connected network would be 5×7
- In the case of CNNs, only links to neighboring pixels are considered, the weight matrix has a quasi-diagonal form



CNNs vs. fully-connected networks

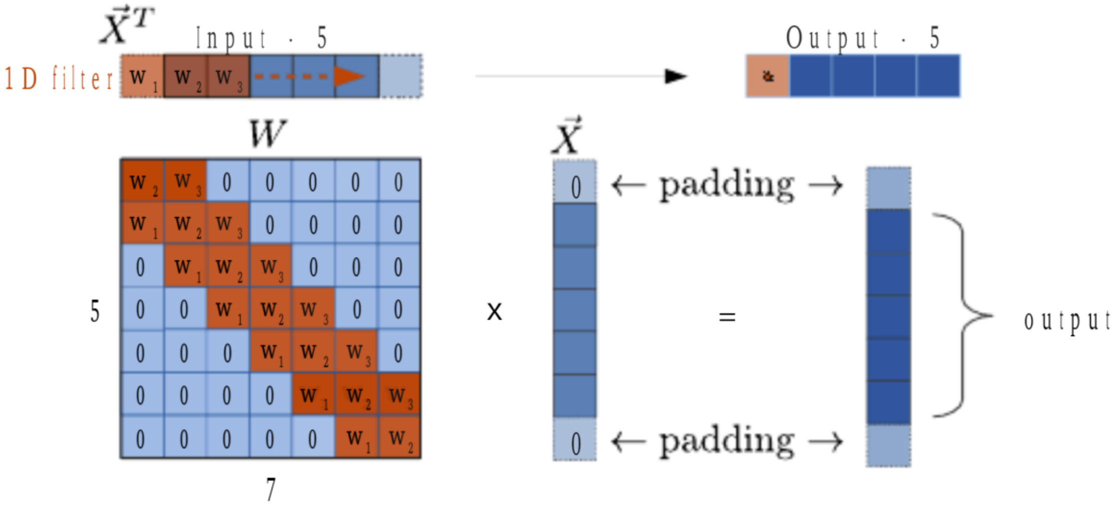
- Number of weights is further reduced as the shame filter is shared over the entire signal (weight sharing)

- This corresponds to a massive reduction of adaptive parameters compared to fully-connected networks

- However, this reduction is justified from symmetry considerations and thus does not affect the descriptive power of the model

- Advantages:
    1. Simplified optimization (training)
    2. Reducing the chances of overfitting

    ➢ Less parameters involved
    ➢ Weight-sharing reduces impact of fluctuations



CNNs vs. fully-connected networks

- Importance of weight sharing:
  - ➢ Assume we want to build a cat detector
  - ➢ Suppose that during the training of a CNN one filter was learned, which produces a high response when encountering the face or head of a cat
  - ➢ Whenever the response is strong, we can identify a cat
  - ➢ Since this filter is shared over the entire image, it is location-independent – it does not matter where the cat is on the image
    - ➢ Training benefits: less trained parameters
    - ➢ Practical advantage: not only detect the cat but also locate its position

Summary

- Convolutional neural networks (CNNs) are the standard architecture for building deep networks to process image-like data
- CNNs simplify the underlying numerical problem by using symmetries that exist in images
- By sliding small filters with adaptive weights over the input, the convolutional operation can deal with variable input and output sizes
- Exploiting symmetry in data allows to reduce the total number of model parameters