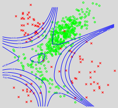
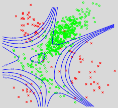


Machine Learning

Blockkurs *Neuronale Netze und Deep Learning* vom 24.9.2019

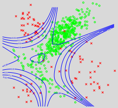
Mario Stanke
Institut für Mathematik und Informatik
Universität Greifswald

Problems with Fully Connected Neural Nets (only `Dense` layers)



Problems with Fully Connected Neural Nets (only `Dense` layers)

- high number of parameters
- when images are input:



Problems with Fully Connected Neural Nets (only `Dense` layers)

- high number of parameters
- when images are input:
  - no notion of pixel neighborhoods
  - no translation invariance

Cross-correlation (2-dimensional)

Definition 1

Let $A = (a_{ij})_{0 \le i,j < m}$ be a square $m \times m$-dimensional matrix and

$$B = \left(b_{ij}\right)_{\substack{0 \le i < h \\ 0 \le j < w}}$$

be another matrix of shape $h \times w$.
The $h - n + 1 \times w - n + 1$-dimensional matrix C with entries

$$c_{i,j} := \sum_{i'=0}^{m-1} \sum_{j'=0}^{m-1} a_{i',j'} \cdot b_{i+i',j+j'}$$

is the 2-dimensional cross-correlation of A and B. We write $C = A * B$.

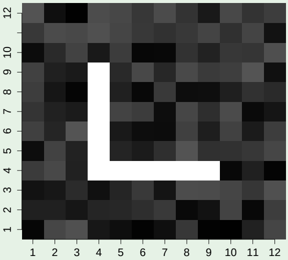
Example 2

$m - 2$, $h = 4$, $w = 5$.

$$A = \begin{pmatrix} 1 & -1 \\ 2 & -3 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & -3 & 0 & 2 & -1 \\ 0 & 1 & 4 & 0 & 1 \\ 2 & -2 & 7 & 3 & 0 \\ -1 & 0 & 1 & 0 & 4 \end{pmatrix} \qquad C = \begin{pmatrix} 2 & -13 & 6 & 0 \\ 9 & -28 & 9 & 5 \\ 2 & -12 & 6 & -9 \end{pmatrix}$$

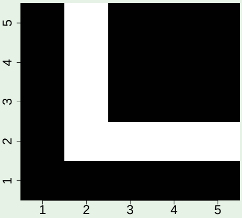
Cross-Correlation of an Image

$$
\begin{pmatrix}
0.56 & 0.54 & 0.59 & 0.41 & 0.56 & 0.54 & 0.45 & 0.50 & 0.44 & 0.58 & 0.44 & 0.54 \\
0.48 & 0.42 & 0.53 & 0.48 & 0.53 & 0.46 & 0.42 & 0.52 & 0.60 & 0.52 & 0.56 & 0.52 \\
0.40 & 0.56 & 0.56 & 0.42 & 0.51 & 0.55 & 0.58 & 0.47 & 0.55 & 0.53 & 0.51 & 0.57 \\
0.40 & 0.44 & 0.58 & 1.00 & 0.51 & 0.47 & 0.51 & 0.43 & 0.54 & 0.48 & 0.56 & 0.46 \\
0.53 & 0.42 & 0.58 & 1.00 & 0.59 & 0.55 & 0.56 & 0.43 & 0.57 & 0.52 & 0.53 & 0.52 \\
0.44 & 0.53 & 0.45 & 1.00 & 0.51 & 0.43 & 0.43 & 0.53 & 0.49 & 0.42 & 0.52 & 0.57 \\
0.41 & 0.51 & 0.53 & 1.00 & 0.46 & 0.43 & 0.54 & 0.42 & 0.57 & 0.42 & 0.53 & 0.53 \\
0.43 & 0.49 & 0.49 & 1.00 & 0.48 & 0.46 & 0.56 & 0.48 & 0.50 & 0.54 & 0.56 & 0.57 \\
0.45 & 0.49 & 0.44 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 1.00 & 0.46 & 0.59 & 0.58 \\
0.59 & 0.45 & 0.50 & 0.48 & 0.48 & 0.60 & 0.46 & 0.41 & 0.46 & 0.55 & 0.57 & 0.40 \\
0.56 & 0.48 & 0.45 & 0.57 & 0.55 & 0.49 & 0.48 & 0.45 & 0.47 & 0.50 & 0.58 & 0.43 \\
0.41 & 0.48 & 0.44 & 0.54 & 0.43 & 0.55 & 0.52 & 0.54 & 0.55 & 0.43 & 0.53 & 0.59
\end{pmatrix}
$$

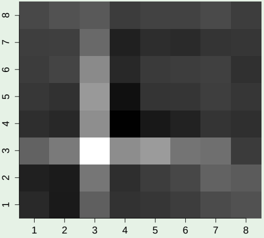


input image B

“filter” A



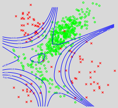
$$
\begin{pmatrix}
-1 & 4 & -1 & -1 & -1 \\
-1 & 4 & -1 & -1 & -1 \\
-1 & 4 & -1 & -1 & -1 \\
-1 & 4 & 4 & 4 & 4 \\
-1 & -1 & -1 & -1 & -1
\end{pmatrix}
$$



output image

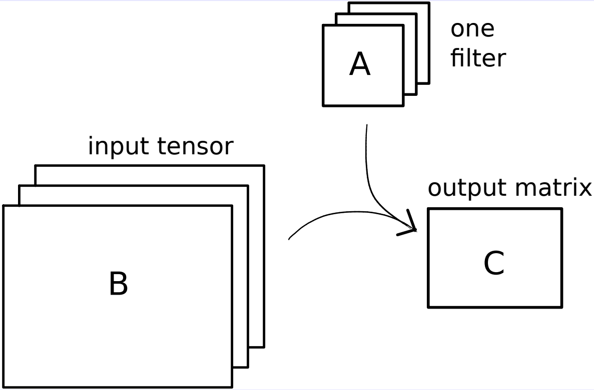
$C = A * B$

$$
\begin{pmatrix}
6.6 & 7.4 & 5.0 & 4.19 & 4.7 & 4.3 & 4.7 & 5.2 \\
6.2 & 7.9 & 8.8 & 4.56 & 5.3 & 5.4 & 4.6 & 6.2 \\
5.2 & 5.9 & 9.2 & 3.27 & 4.6 & 5.2 & 3.8 & 5.6 \\
4.7 & 5.2 & 12.1 & 2.57 & 4.6 & 5.0 & 4.0 & 5.5 \\
4.4 & 4.1 & 11.0 & 0.85 & 2.0 & 3.7 & 3.1 & 5.1 \\
7.7 & 9.3 & 19.3 & 11.13 & 11.9 & 10.8 & 8.4 & 7.2 \\
2.7 & 2.7 & 9.6 & 3.69 & 4.4 & 5.7 & 5.4 & 6.7 \\
3.7 & 3.1 & 7.9 & 4.14 & 4.8 & 5.2 & 5.3 & 6.0
\end{pmatrix}
$$

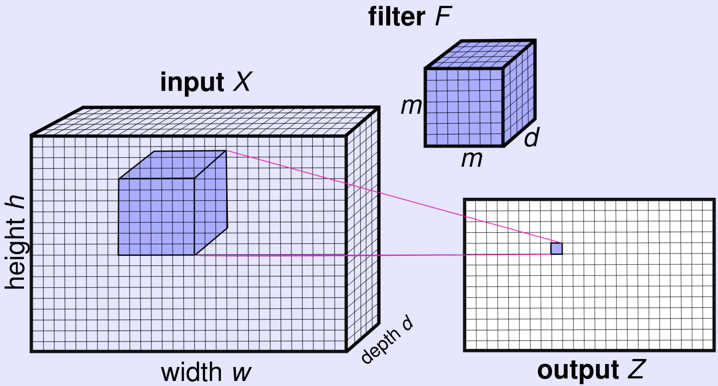


3-dimensional input

- Want to
  1. use multiple filters in parallel and
  2. stack several (convolutional) layers.
- Also, color images are naturally encoded as 3-dimensional (each pixel has a red, green and blue value).
- Solution: Define convolution for 3-dimensional tensor input as well.

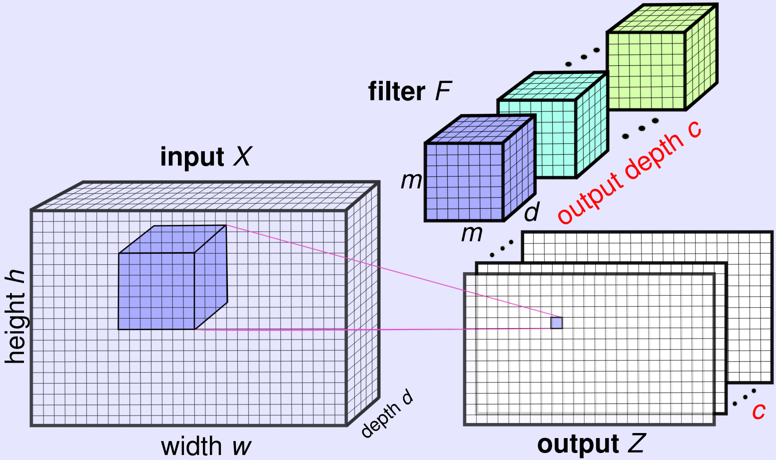


Convolutional Layer



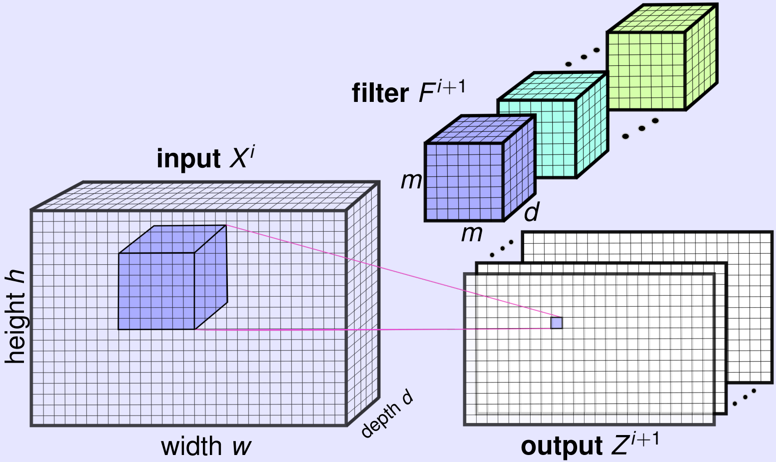
$$z_{i,j} = \sum_{i'=0}^{m-1} \sum_{j'=0}^{m-1} \sum_{k=0}^{d-1} x_{i+i',j+j',k} \cdot f_{i',j',k}$$

Convolutional Layer

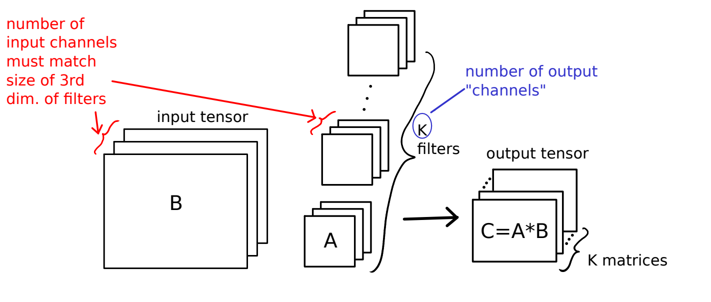


$$z_{i,j,r} = \sum_{i'=0}^{m-1} \sum_{j'=0}^{m-1} \sum_{k=0}^{d-1} x_{i+i',j+j',k} \cdot f_{i',j',k,r} + \underbrace{b_r}_{\text{bias}}$$

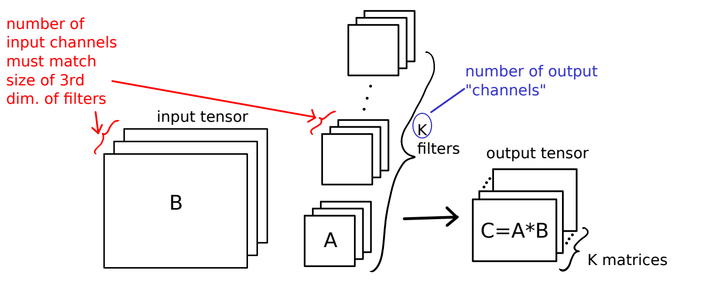
Convolutional Layer



Convolutional Layer

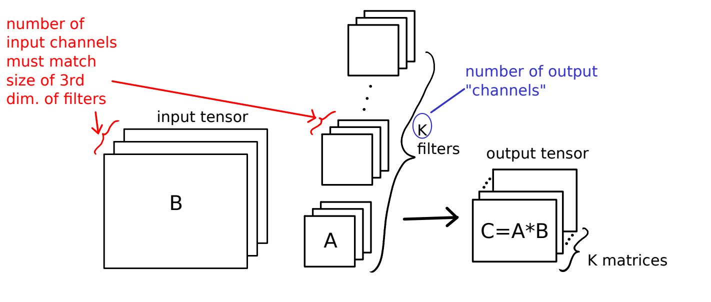


Convolutional Layer



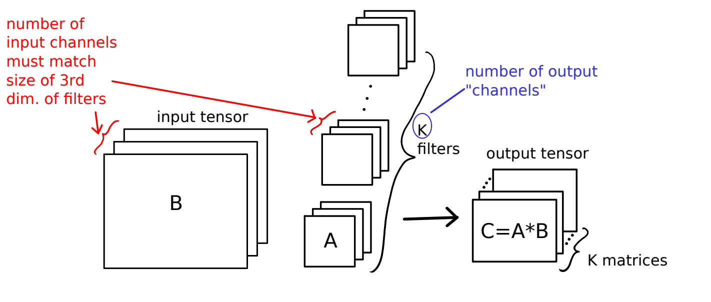
- The input width and height can be conserved in the ouput layer by zero-padding of input (`padding = 'same'`)

Convolutional Layer



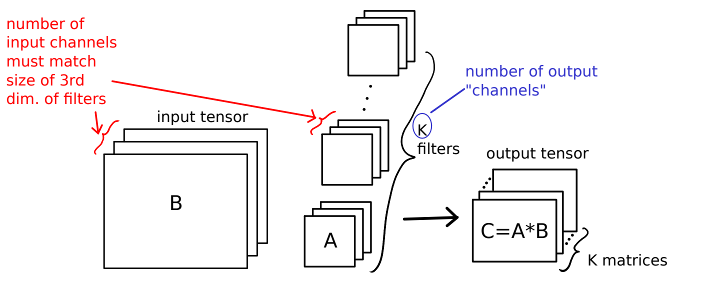
- The input width and height can be conserved in the ouput layer by zero-padding of input (`padding = 'same'`)
- Stride (Schrittweite) s: Skip $s-1$ positions in each direction when 'sliding' A over $B \Rightarrow$ decreases output layer size up to a factor of s^2.

Convolutional Layer



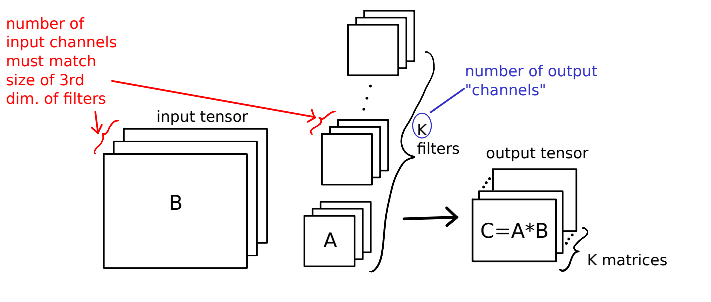
- The input width and height can be conserved in the ouput layer by zero-padding of input (`padding = 'same'`)
- Stride (Schrittweite) s: Skip $s-1$ positions in each direction when 'sliding' A over $B \Rightarrow$ decreases output layer size up to a factor of s^2.
- The matrices A are learned, not set manually. The derivative wrt. to the filter matrix parameters is computed during BackProp.

Convolutional Layer



- The input width and height can be conserved in the ouput layer by zero-padding of input (`padding = 'same'`)
- Stride (Schrittweite) s: Skip $s - 1$ positions in each direction when 'sliding' A over $B \Rightarrow$ decreases output layer size up to a factor of s^2.
- The matrices A are learned, not set manually. The derivative wrt. to the filter matrix parameters is computed during BackProp.
- Convolution is a special case of a fully-connected layer, in which certain parameters are shared (*parameter sharing*).

Convolutional Layer



- The input width and height can be conserved in the ouput layer by zero-padding of input (`padding = 'same'`)
- Stride (Schrittweite) s: Skip $s - 1$ positions in each direction when 'sliding' A over $B \Rightarrow$ decreases output layer size up to a factor of s^2.
- The matrices A are learned, not set manually. The derivative wrt. to the filter matrix parameters is computed during BackProp.
- Convolution is a special case of a fully-connected layer, in which certain parameters are shared (*parameter sharing*).
- Output neurons of convolution can detect lower-level features like ("lower left corner", "pupil") and be combined in deeper layers.

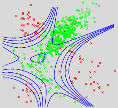
Pooling-Layers

Max-Pooling (`tf.keras.layers.MaxPool2D`)

- similar to a convolutional layer
- requires a `pool_size` m like the filter size
- does not have any parameters
- computes output

$$z_{i,j,r} = \max_{\substack{i' \in [0,\, m) \\ j' \in [0,\, m)}} x_{s \cdot i + i', s \cdot j + j', r}$$

- is usually applied with a stride $s \geq 2$ and therefore reduces height and width
- intuition:



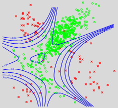
Pooling-Layers

Max-Pooling (`tf.keras.layers.MaxPool2D`)

- similar to a convolutional layer
- requires a `pool_size` m like the filter size
- does not have any parameters
- computes output

$$z_{i,j,r} = \max_{\substack{i' \in [0, m) \\ j' \in [0, m)}} x_{s \cdot i+i', s \cdot j+j', r}$$

- is usually applied with a stride $s \geq 2$ and therefore reduces height and width
- intuition: checks local presence of *at least one* feature (e.g. a green stroke) rather than summing up if there are multiple



Pooling-Layers

Max-Pooling (`tf.keras.layers.MaxPool2D`)

- similar to a convolutional layer
- requires a `pool_size` m like the filter size
- does not have any parameters
- computes output

$$z_{i,j,r} = \max_{\substack{i' \in [0,\, m) \\ j' \in [0,\, m)}} x_{s\cdot i+i',\, s\cdot j+j',\, r}$$

- is usually applied with a stride $s \geq 2$ and therefore reduces height and width
- intuition: checks local presence of *at least one* feature (e.g. a green stroke) rather than summing up if there are multiple

With an analogous definition, average pooling *averages* over regions of size $m \times m$, but is used more rarely.