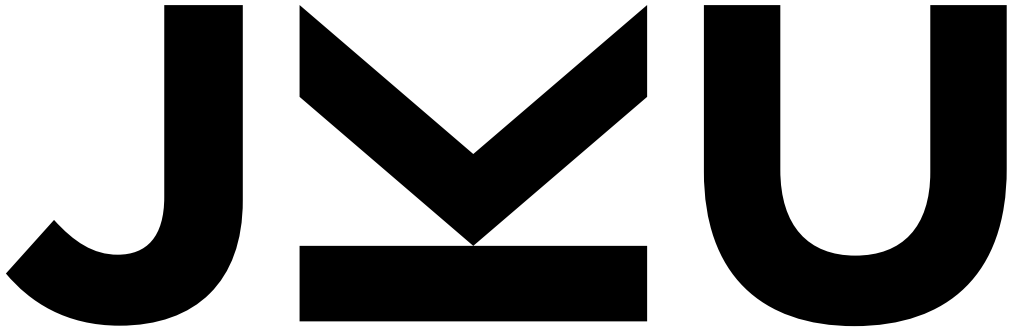
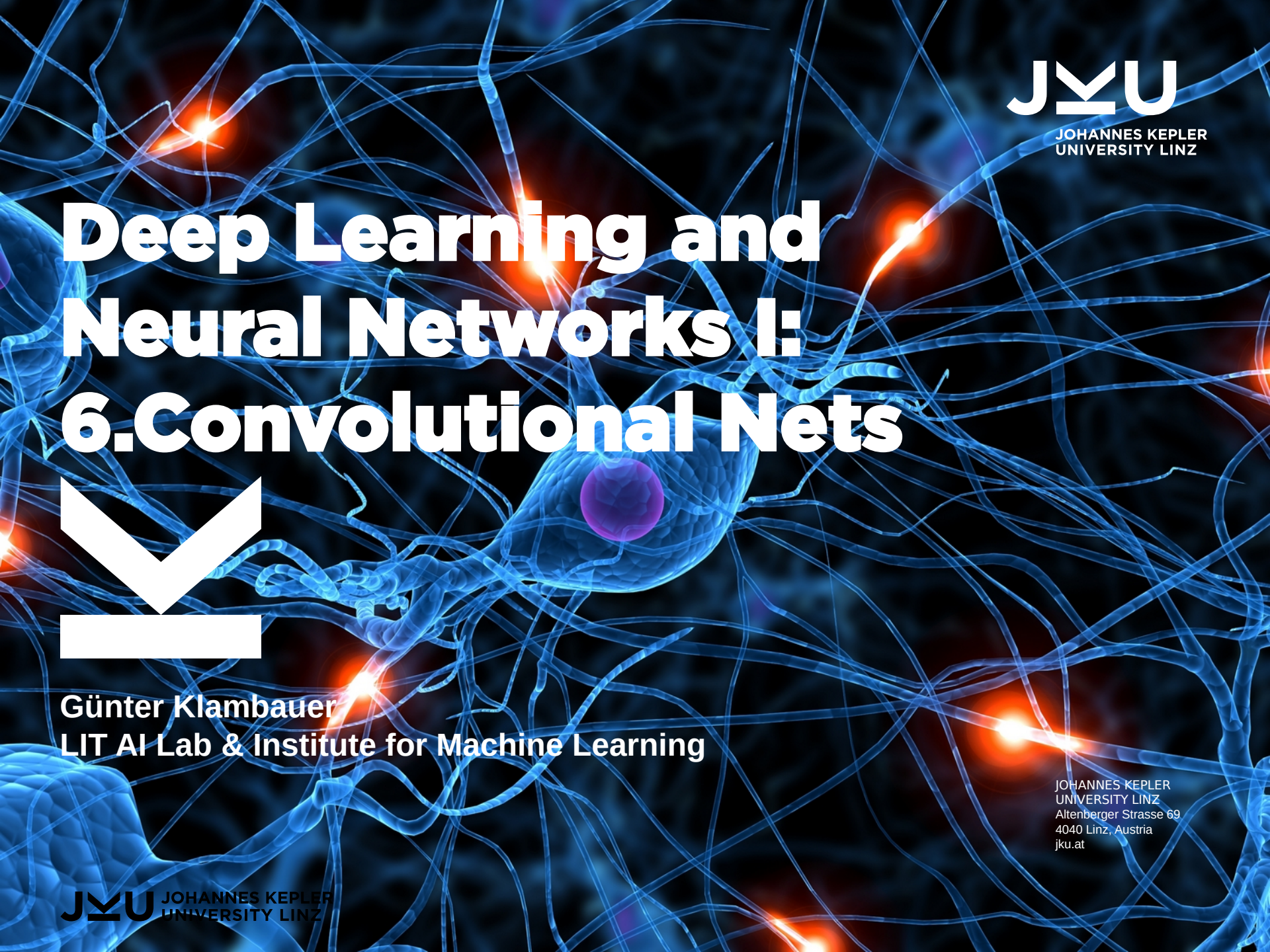


JOHANNES KEPLER
UNIVERSITY LINZ



JOHANNES KEPLER
UNIVERSITY LINZ

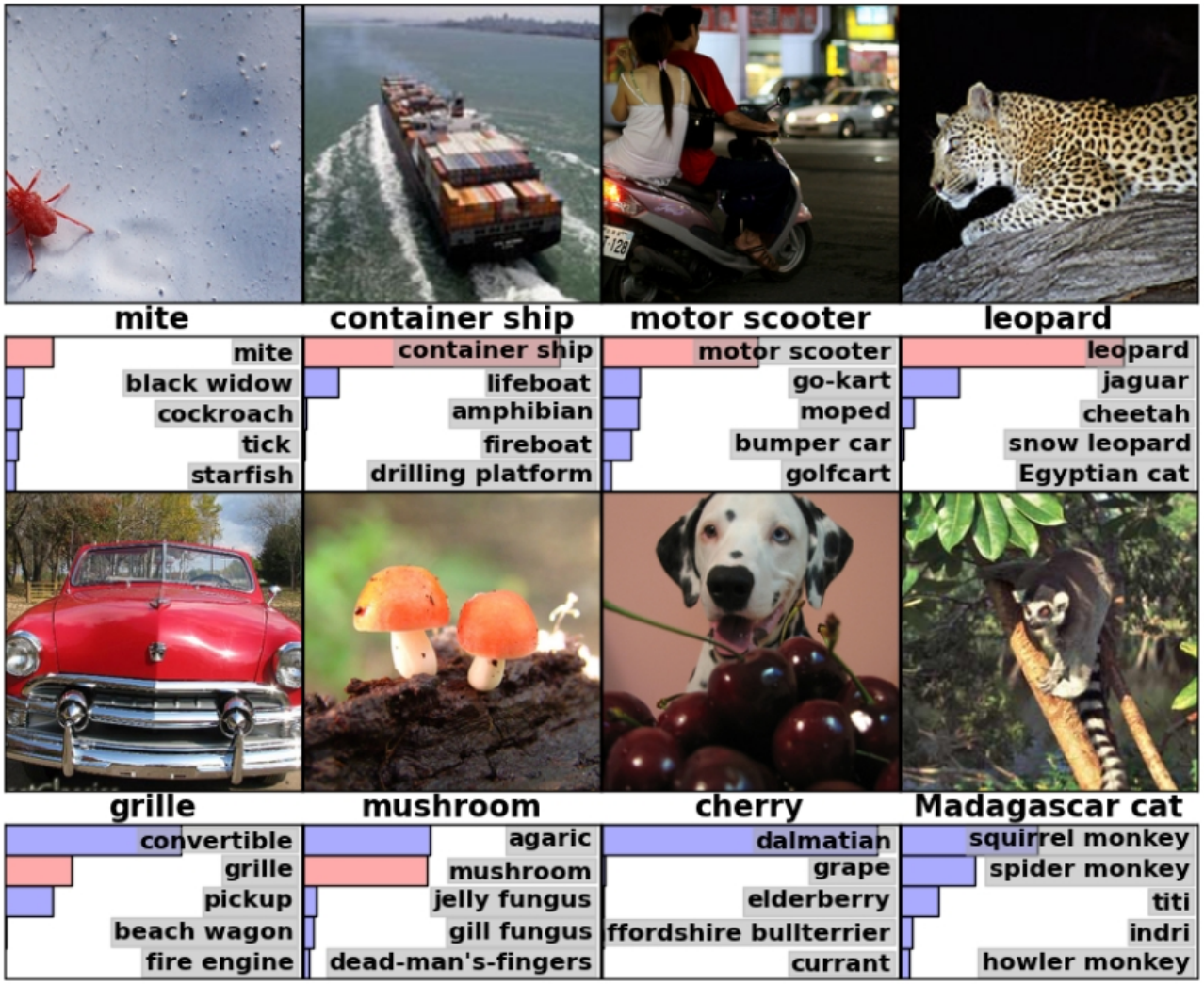
Deep Learning and Neural Networks I: 6.Convolutional Nets

Günter Klambauer
LIT AI Lab & Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Strasse 69
4040 Linz, Austria
jku.at

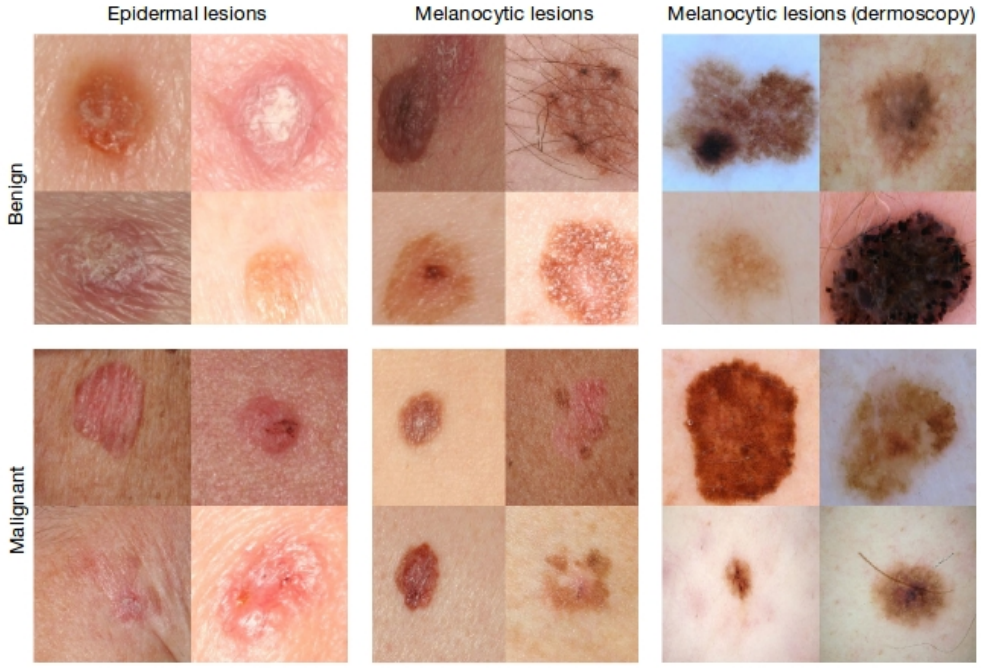
JOHANNES KEPLER
UNIVERSITY LINZ

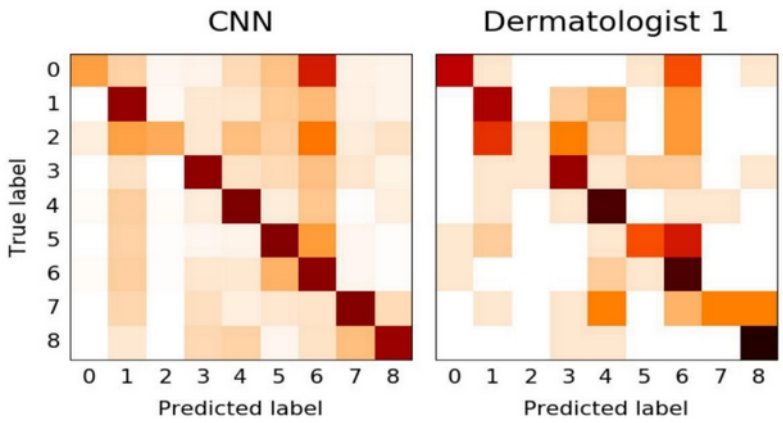
Motivation: Image recognition tasks (1/3)



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90.

Motivation: Image recognition tasks (2/3)





Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639), 115-118.

Motivation: recent example (3/3)

SNDCNN: SELF-NORMALIZING DEEP CNNs WITH SCALED EXPONENTIAL LINEAR UNITS FOR SPEECH RECOGNITION

Zhen Huang, Tim Ng, Leo Liu, Henry Mason, Xiaodan Zhuang, Daben Liu

Apple Inc., USA
{zhen_huang,tim_ng,lliu9,hmason,xiaodan_zhuang,daben_liu}@apple.com

ABSTRACT

Very deep CNNs achieve state-of-the-art results in both computer vision and speech recognition, but are difficult to train. The most popular way to train very deep CNNs is to use shortcut connec-

shown to improve inference speed on CPUs [13] for neural networks of all types. Specifically for speech recognition, running inference at a decreased frame rate [14] has also been shown to reduce computation cost without affecting accuracy.

- 50-layer convnet for speech recognition
- 60-80% faster than before
- Lower error-rate than before

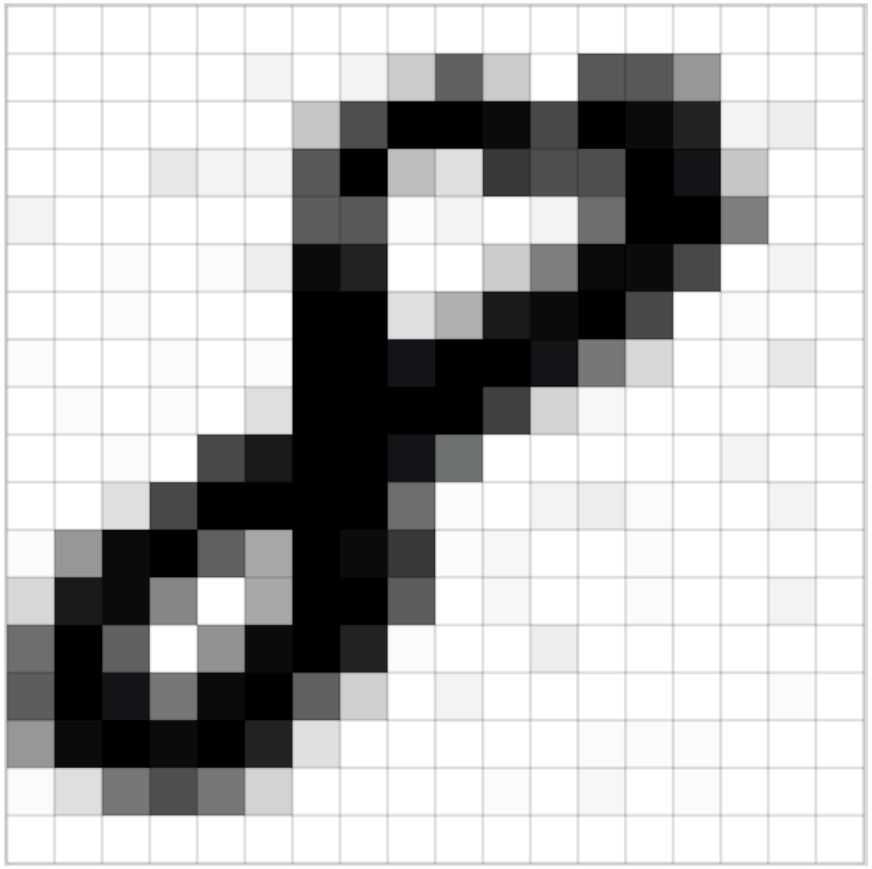
Overview: Chapter 6

- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

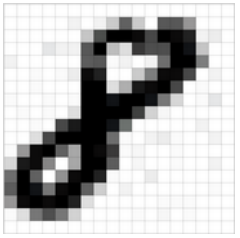
Images as input: convolutional NNs

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Images as input: convolutional NNs



The MNIST dataset

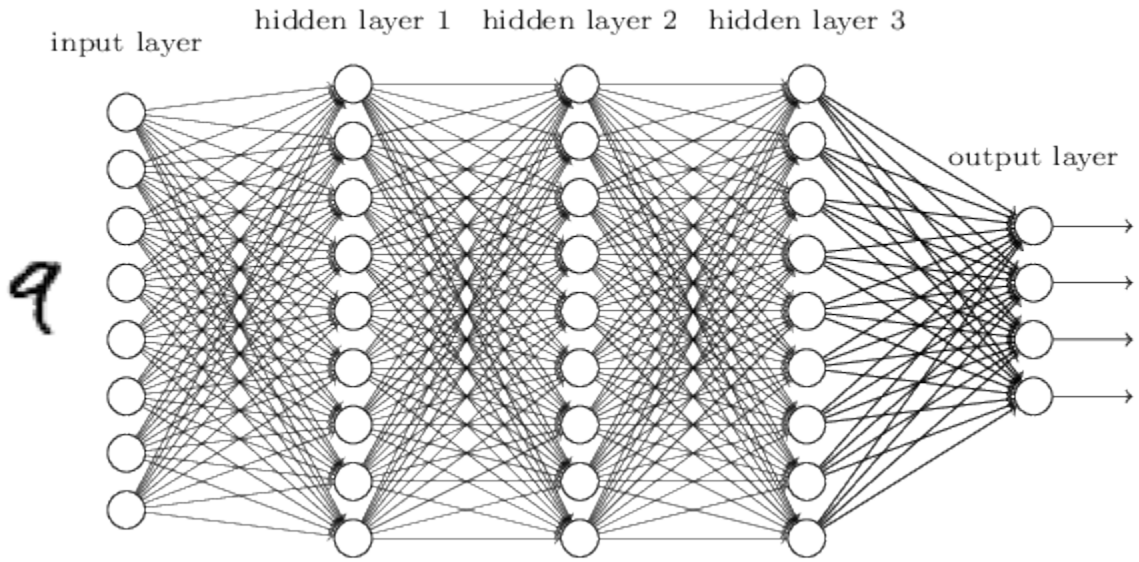


- classic dataset
- small (28x28) handwritten grayscale digits
- developed in the 1990s
- “hello world” for introducing deep learning
- 60,000 images for training
- 10,000 for testing
- Even simple methods can reach >99% accuracy

Overview: Chapter 6

- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

MNIST with fully-connected networks



The convolution operation

- A discrete convolution between two functions h and k is:

$$(h * k)(a) = \sum_{i=-\infty}^{\infty} h(a-i)\ k(i)$$

Convolution and cross-correlation

- For images, convolutions across two axes are used:

$$(h * k)(a, b) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(a - i, b - j)\ k(i, j),$$

- What is actually used is the “cross-correlation”:

$$(h * k)(a, b) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(a + i, b + j)\ k(i, j).$$

  In ML still termed “convolution”

Images and the convolution operation (1/2)

$$(h * k)(a, b) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(a+i, b+j)\ k(i, j).$$

- We identify the image with the function h:
  image $\boldsymbol{x} \in \mathbb{R}^{H \times W}$ with height $H$and width W
- We identify the kernel function k with:
  a weight matrix $\boldsymbol{W} \in \mathbb{R}^{R \times R}$ with size R
  (quadratic only for simplicity)

Images and the convolution operation (2/2)

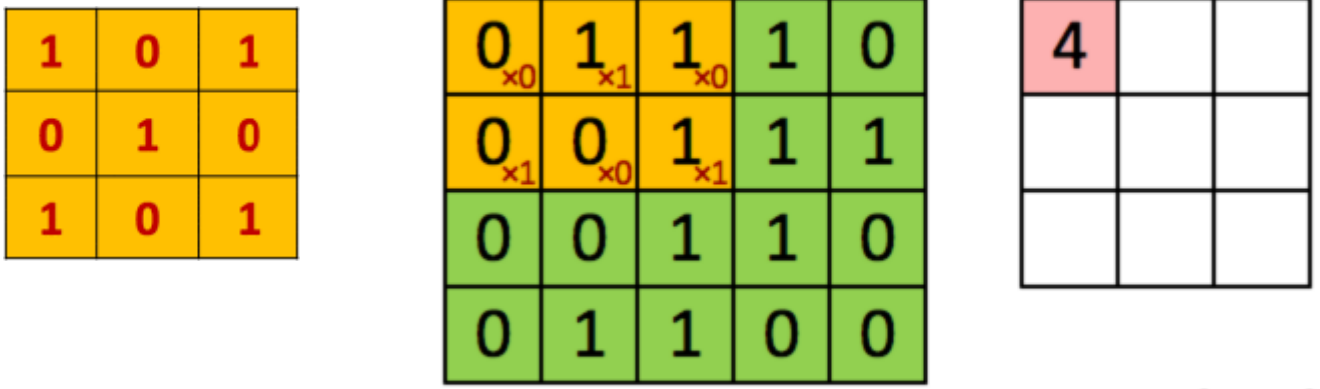
- The weight matrix, also called kernel, is "shifted" across the image to obtain a *feature map*:

$$s_{a,b} = \sum_{i=0}^{R-1} \sum_{j=0}^{R-1} w_{i,j}\ x_{a+i,b+j} = (\boldsymbol{W} * \boldsymbol{x})_{a,b}.$$

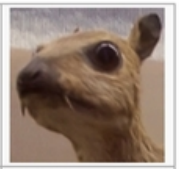
- The quantity $s_{a,b}$ can be, again, seen as pre-activation.
- The convolution operation is a linear operation

Note: indices start with 0 in this chapter!!!!!

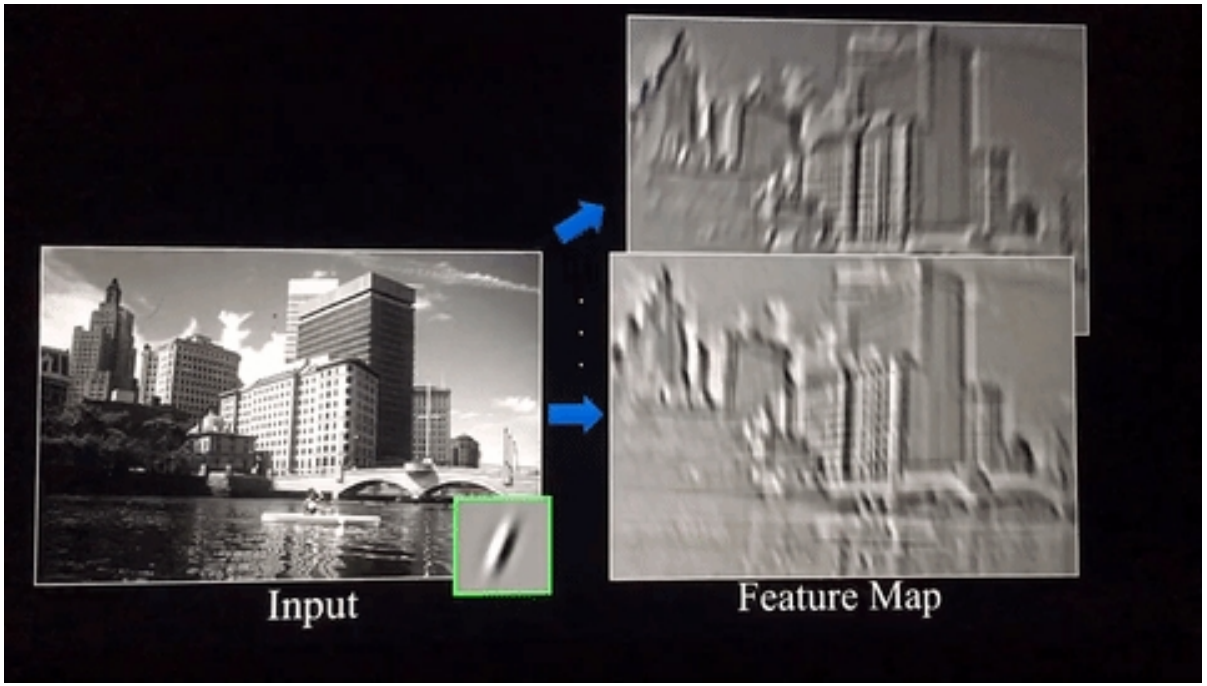
CNNs: The convolution operation explained



W x $s = W * x$

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

CNNs: The convolution operation explained



Demo

- Excel sheets with convolutions

Overview: Chapter 6

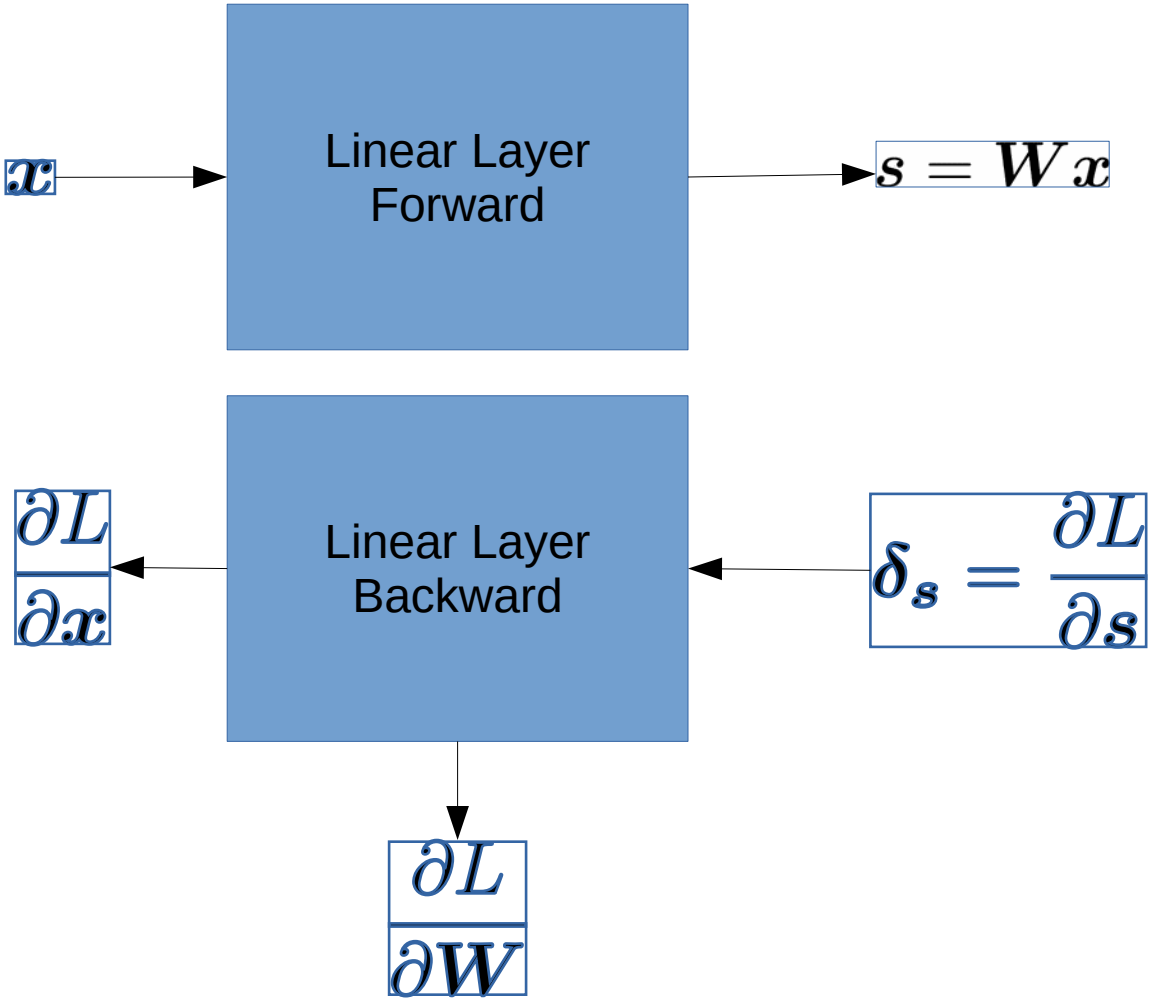
- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

Backpropagation through convolutions

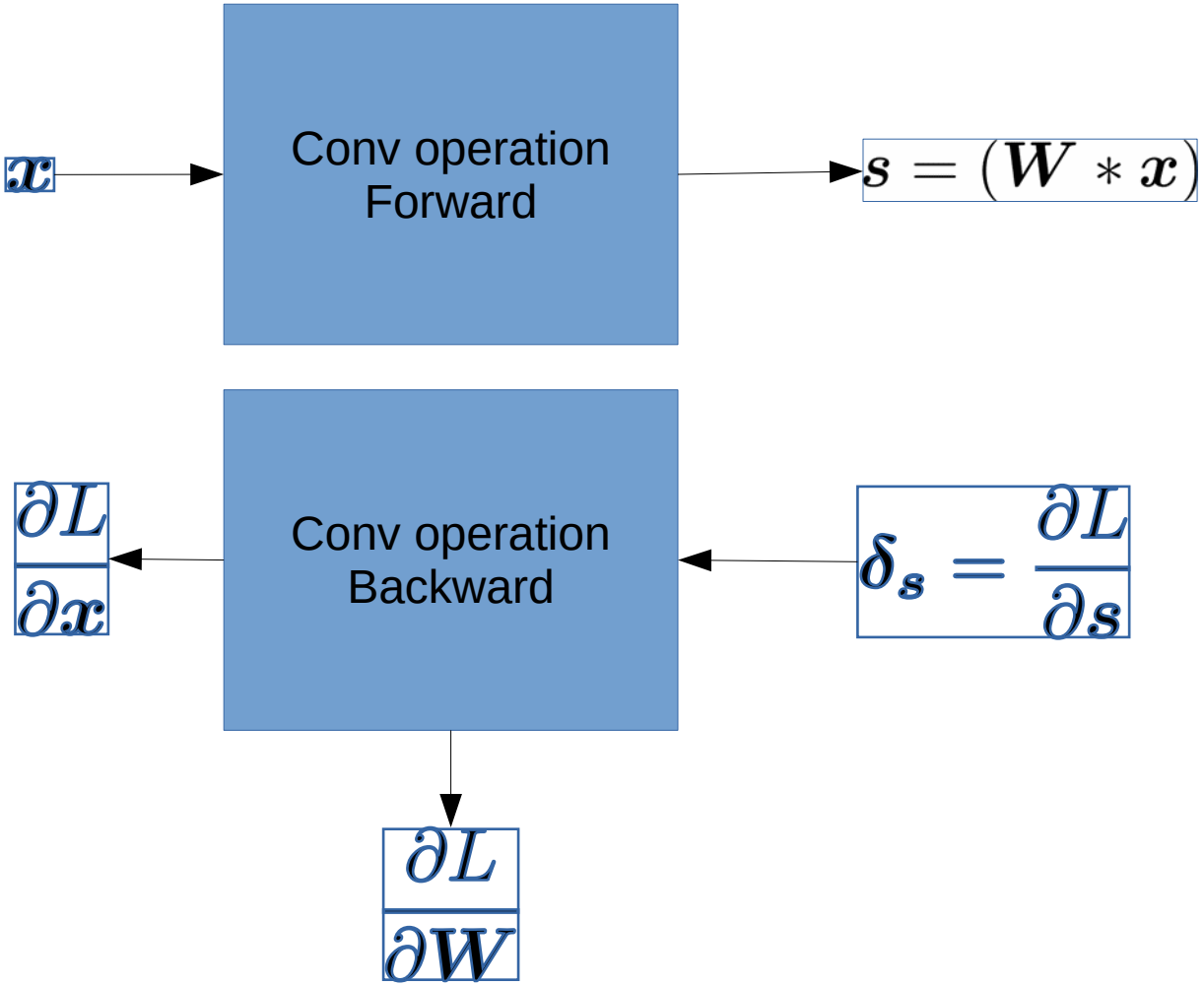
$$s_{a,b} = \sum_{i=0}^{R-1} \sum_{j=0}^{R-1} w_{i,j}\ x_{a+i,b+j} = (\boldsymbol{W} * \boldsymbol{x})_{a,b}.$$

- We assume
  - to have a neural network, which has a “convolutional layer”, i.e. a layer that uses the conv operation.
  - We can differentiate the loss with respect to the preactivations of this conv layer $\frac{\partial L}{\partial s_{a,b}}$
- We want to find:
  - $\frac{\partial L}{\partial w_{i,j}}$: to update the weights
  - $\frac{\partial L}{\partial x_{a,b}}$: to provide deltas for the layer below

Structure of forward and backward pass in FCN



Structure of forward and backward pass in CNNs



Derivative w.r.t. weights

$$s_{a,b} = \sum_{i=0}^{R-1} \sum_{j=0}^{R-1} w_{i,j}\ x_{a+i,b+j} = (\boldsymbol{W} * \boldsymbol{x})_{a,b}$$

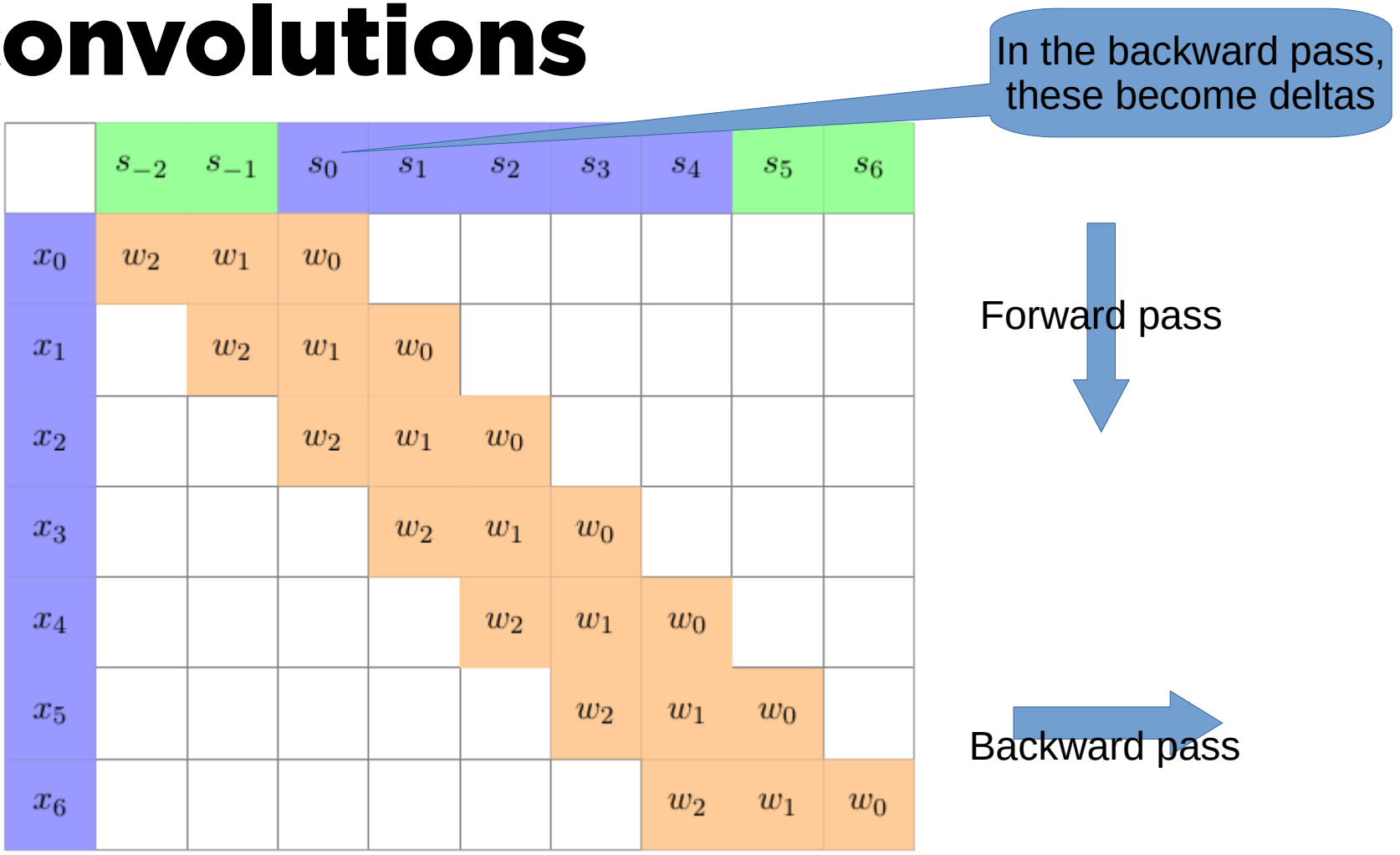
$$\begin{aligned}
\frac{\partial L}{\partial w_{i,j}} &= \sum_{a=0}^{A-R} \sum_{b=0}^{B-R} \frac{\partial L}{\partial s_{a,b}} \frac{\partial s_{a,b}}{\partial w_{i,j}} \\
&= \sum_{a=0}^{A-R} \sum_{b=0}^{B-R} \underbrace{\frac{\partial L}{\partial s_{a,b}}}_{:=\delta_{a,b}} x_{a+i,b+j} \\
&= (\boldsymbol{\delta} * \boldsymbol{x})_{i,j}
\end{aligned}$$

Derivative w.r.t. inputs

$$s_{a,b} = \sum_{i=0}^{R-1} \sum_{j=0}^{R-1} w_{i,j}\ x_{a+i,b+j} = (\boldsymbol{W} * \boldsymbol{x})_{a,b}.$$

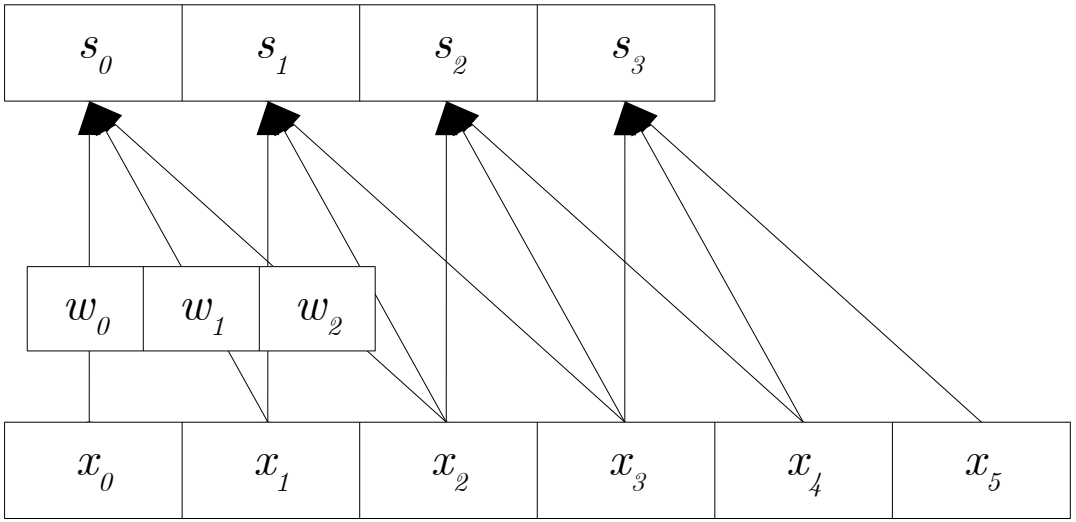
$$\begin{aligned}
\frac{\partial L}{\partial x_{a,b}} &= \sum_{i=0}^{A-R} \sum_{j=0}^{B-R} \frac{\partial L}{\partial s_{i,j}} \frac{\partial s_{i,j}}{\partial x_{a,b}} \\
&= \sum_{i=0}^{A-R} \sum_{j=0}^{B-R} \underbrace{\frac{\partial L}{\partial s_{i,j}}}_{:=\delta_{i,j}} \sum_{u=0}^{R-1} \sum_{v=0}^{R-1} w_{u,v} \frac{\partial x_{i+u,j+v}}{\partial x_{a,b}} \\
&= \sum_{i=0}^{A-R} \sum_{j=0}^{B-R} \delta_{i,j} \sum_{u=0}^{R-1} \sum_{v=0}^{R-1} w_{u,v} \mathbf{1}_{(i+u=a,j+v=b)} \\
&= \sum_{i=a-R+1}^{a} \sum_{j=b-R+1}^{b} \delta_{i,j}\ w_{a-i,b-j} \\
&= \sum_{i'=0}^{R-1} \sum_{j'=0}^{R-1} w_{R-1-i',R-1-j'}\ \delta_{a+i'-R+1,b+j'-R+1}, \\
&= \sum_{i'=0}^{R-1} \sum_{j'=0}^{R-1} \hat{w}_{i',j'}\ \hat{\delta}_{a+i',b+j'} \\
&= (\hat{\boldsymbol{W}} * \hat{\boldsymbol{\delta}})_{a,b}
\end{aligned}$$

Backprop explained with 1-D convolutions

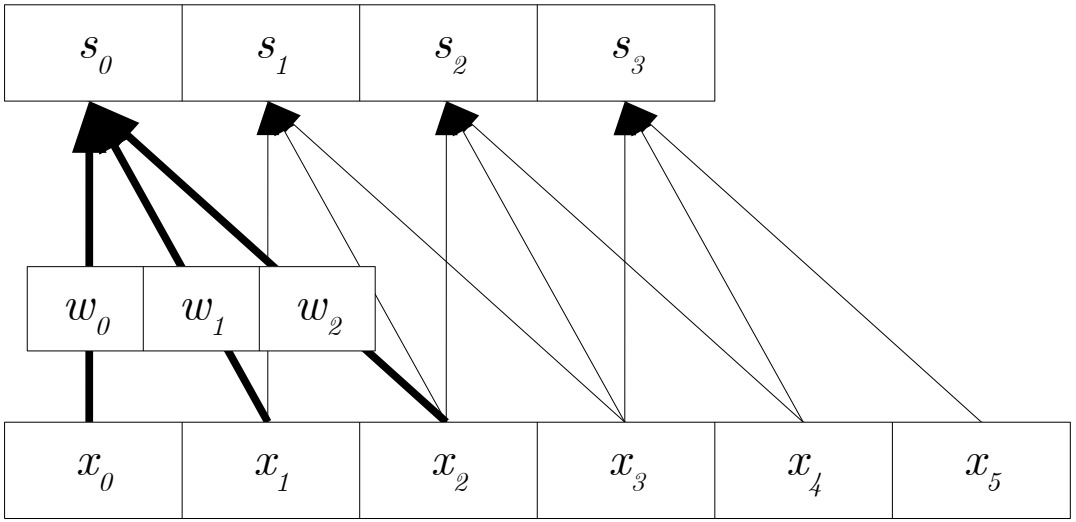


	s_{-2}	s_{-1}	s_0	s_1	s_2	s_3	s_4	s_5	s_6
x_0	w_2	w_1	w_0						
x_1		w_2	w_1	w_0					
x_2			w_2	w_1	w_0				
x_3				w_2	w_1	w_0			
x_4					w_2	w_1	w_0		
x_5						w_2	w_1	w_0	
x_6							w_2	w_1	w_0

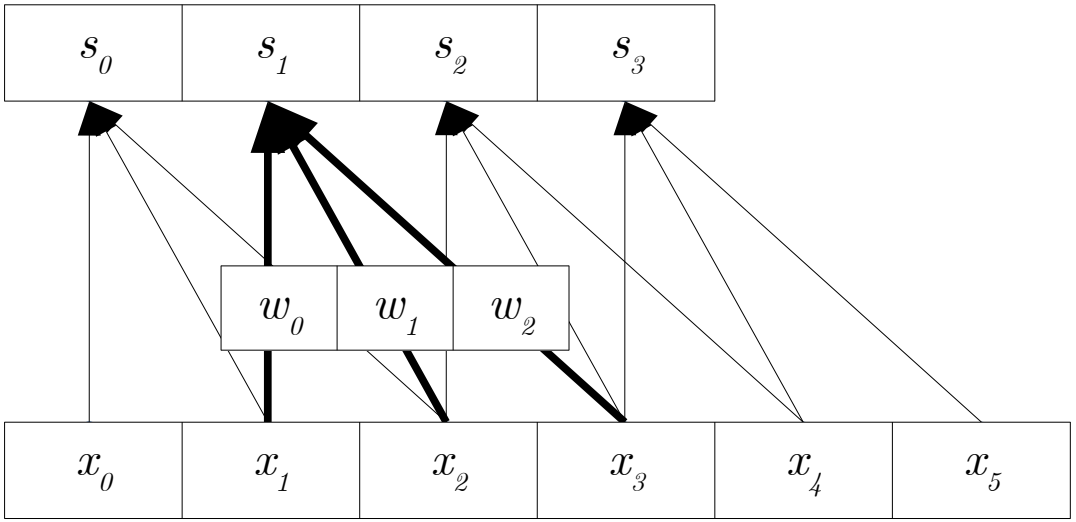
Backprop explained with 1-D convolutions



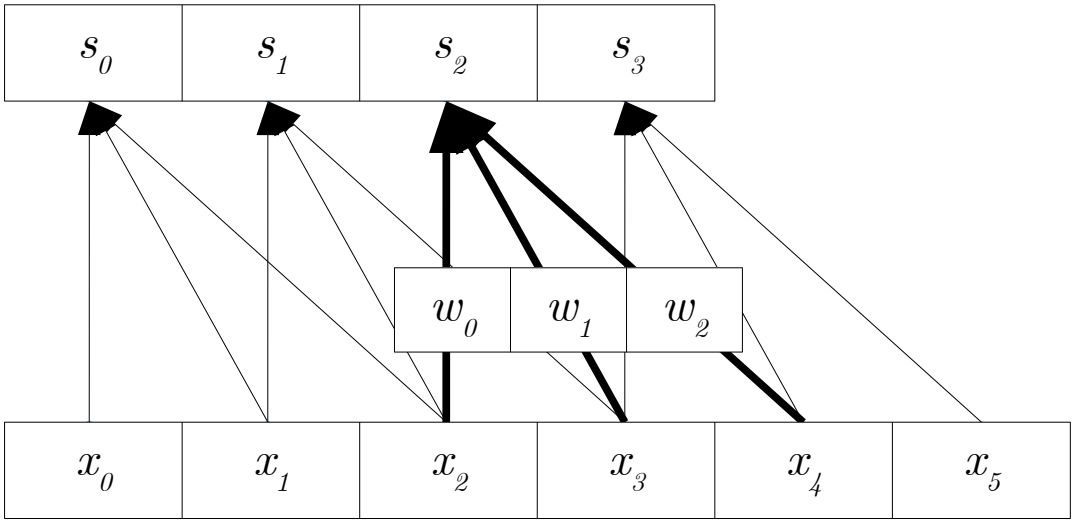
Backprop explained with 1-D convolutions



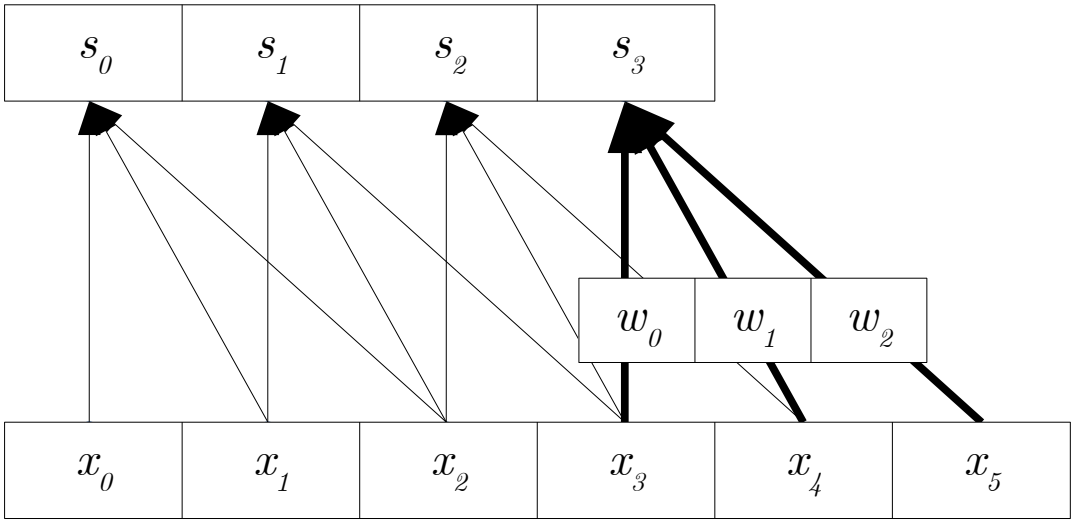
Backprop explained with 1-D convolutions



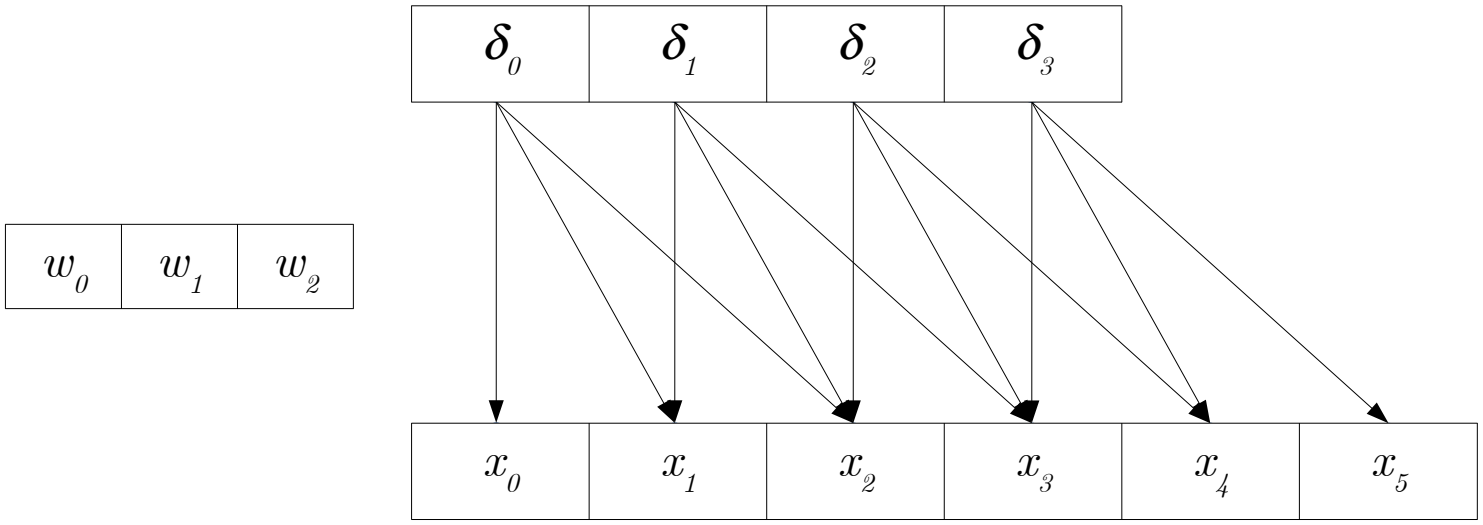
Backprop explained with 1-D convolutions



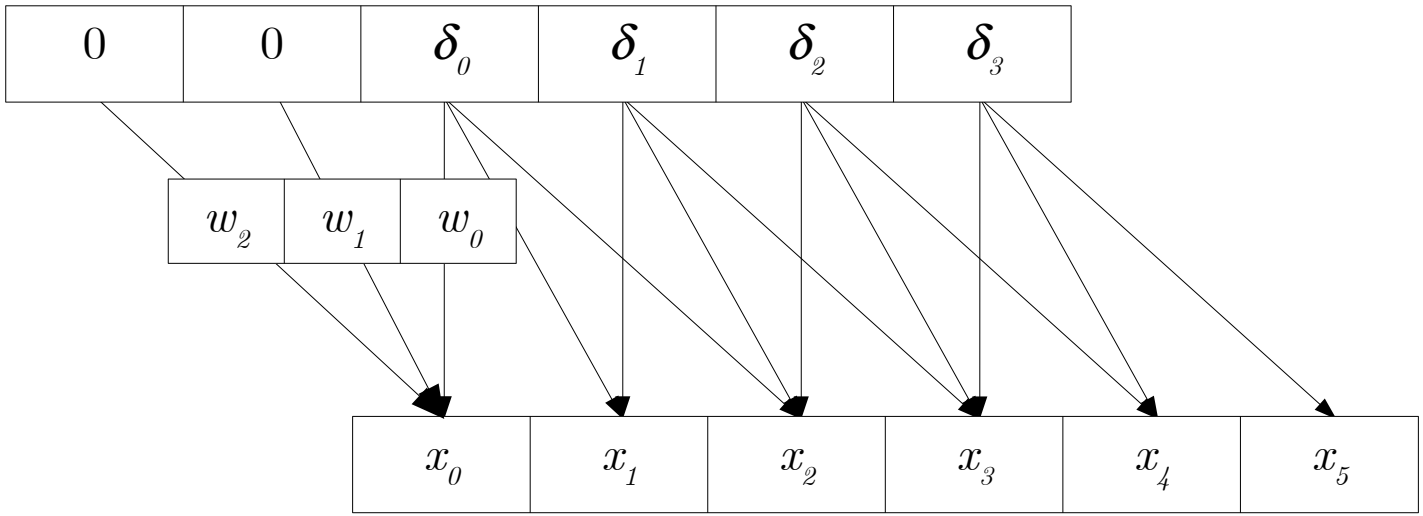
Backprop explained with 1-D convolutions



Backprop explained with 1-D convolutions

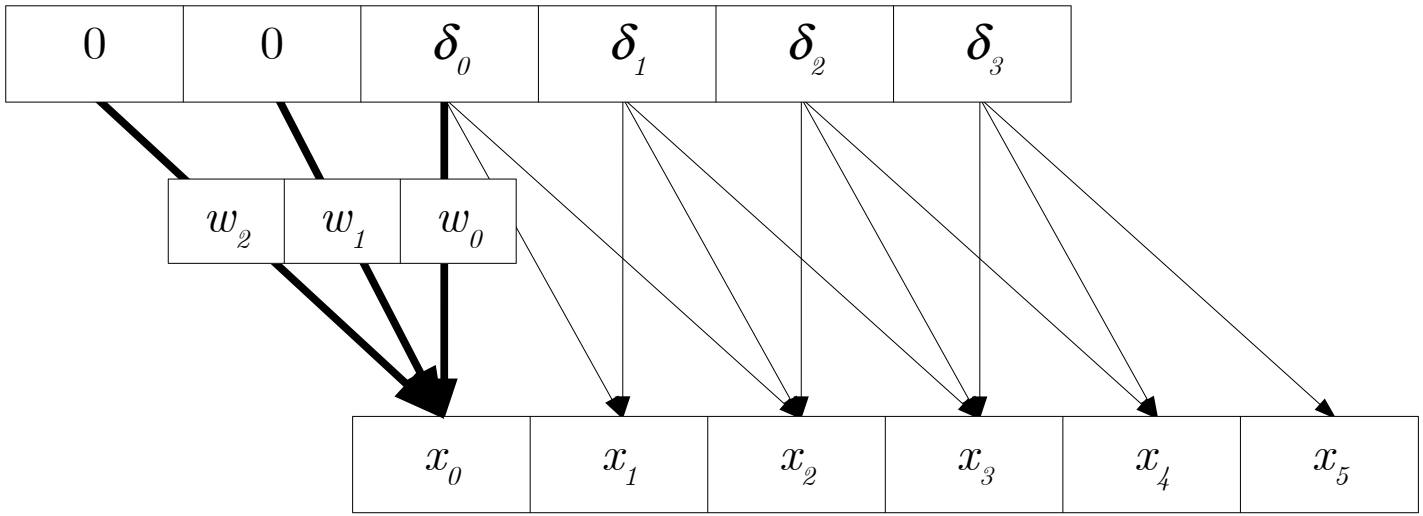


Backprop explained with 1-D convolutions

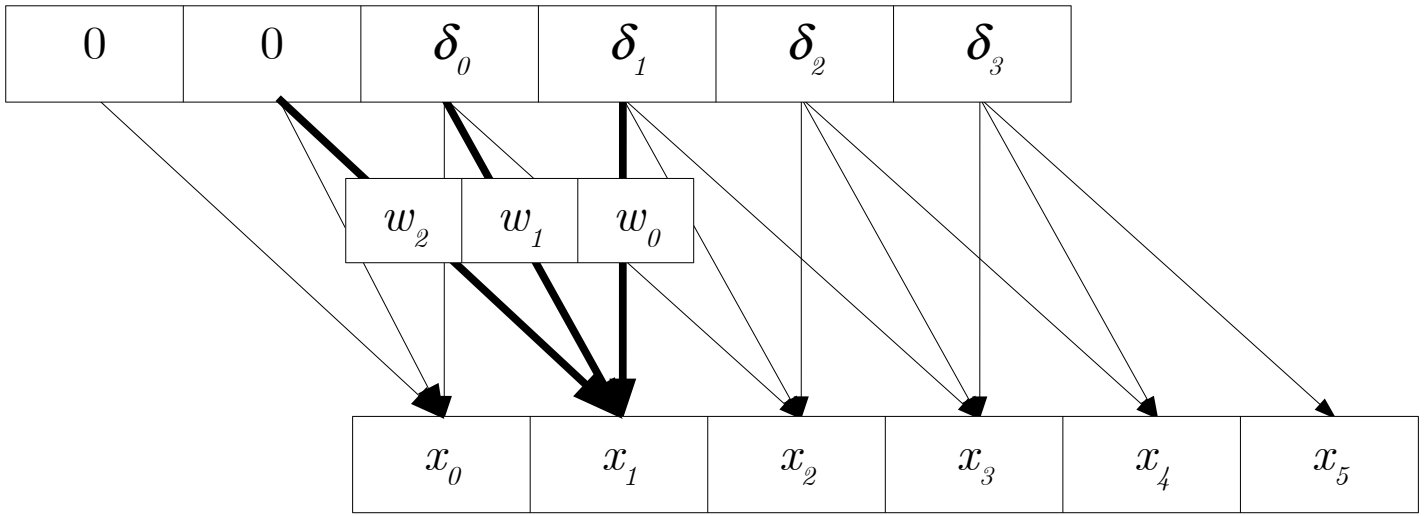


Note: zero padding
and
reversing weights

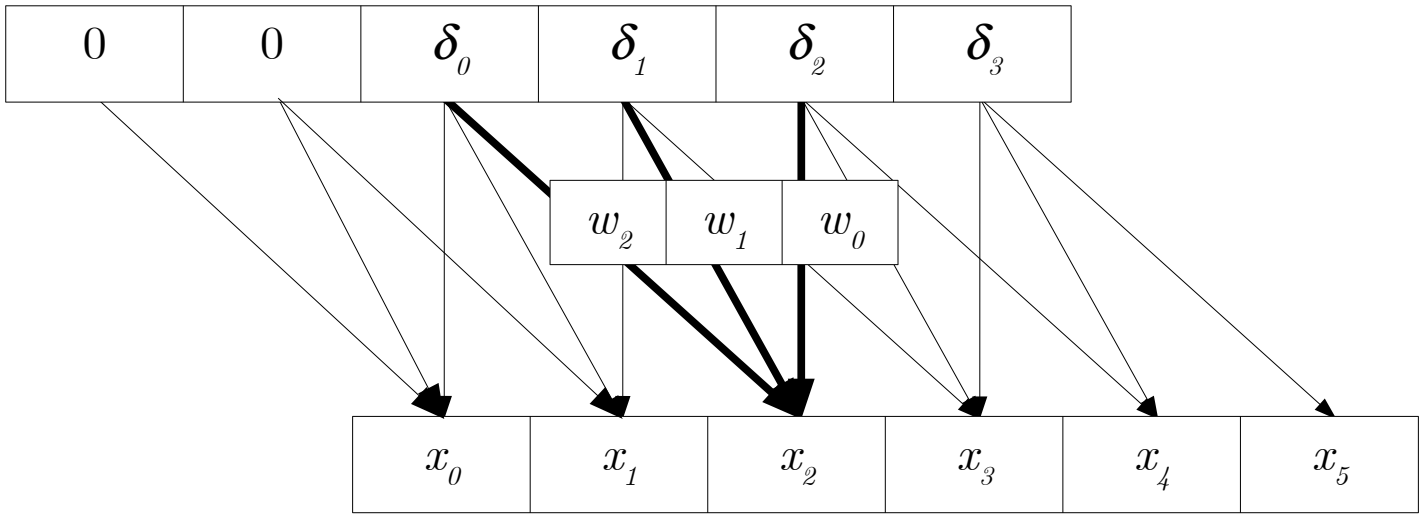
Backprop explained with 1-D convolutions



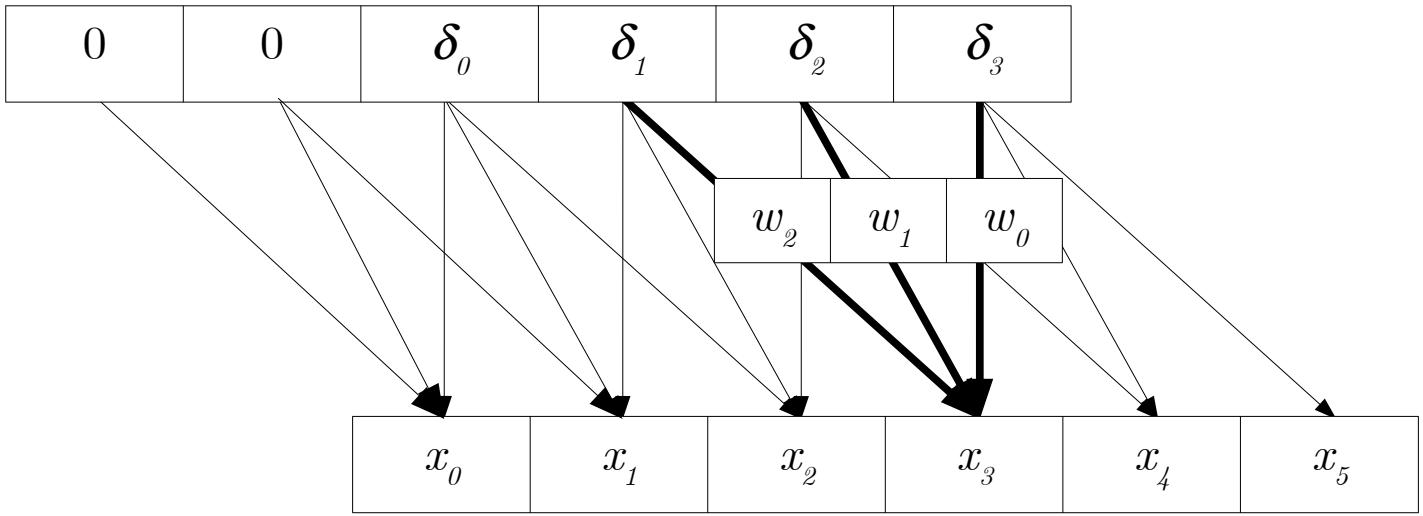
Backprop explained with 1-D convolutions



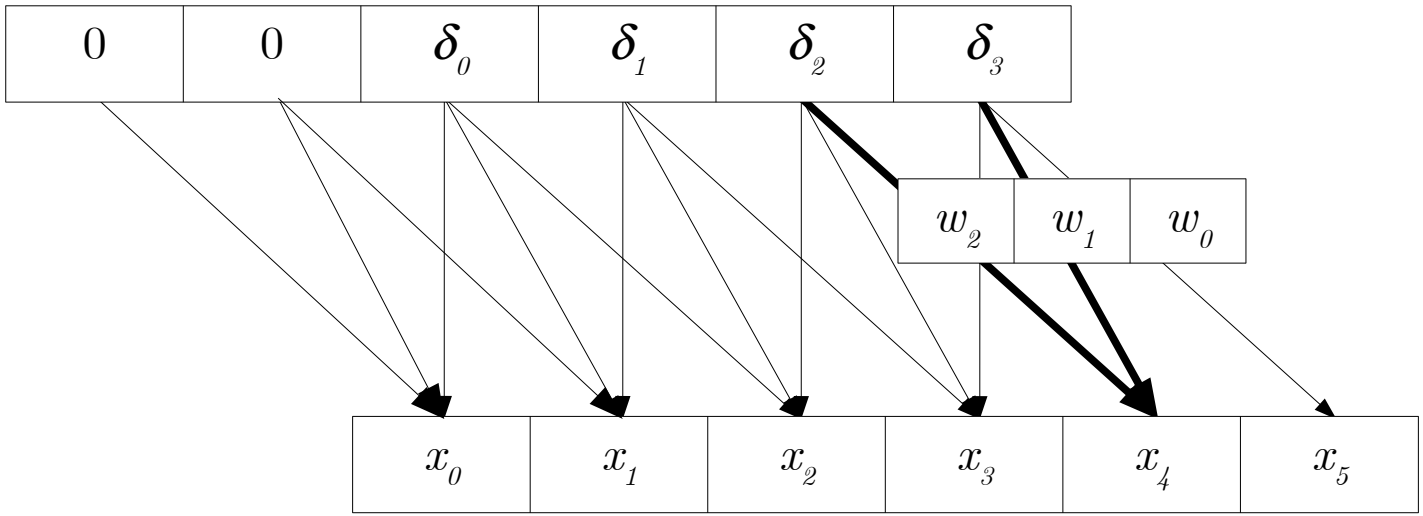
Backprop explained with 1-D convolutions



Backprop explained with 1-D convolutions

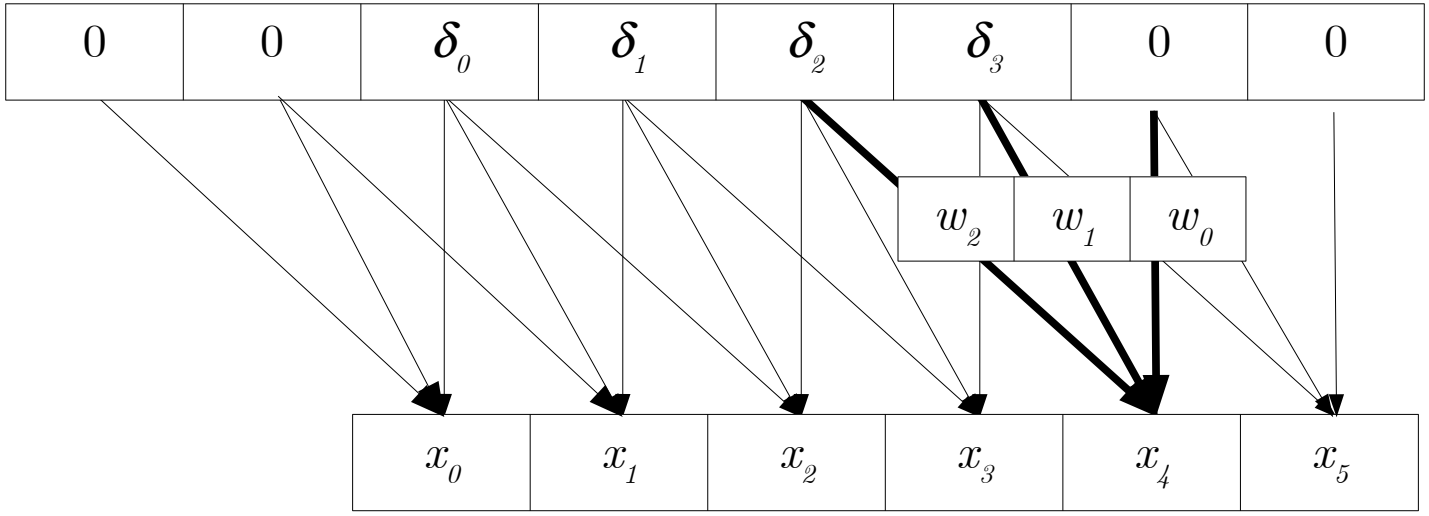


Backprop explained with 1-D convolutions

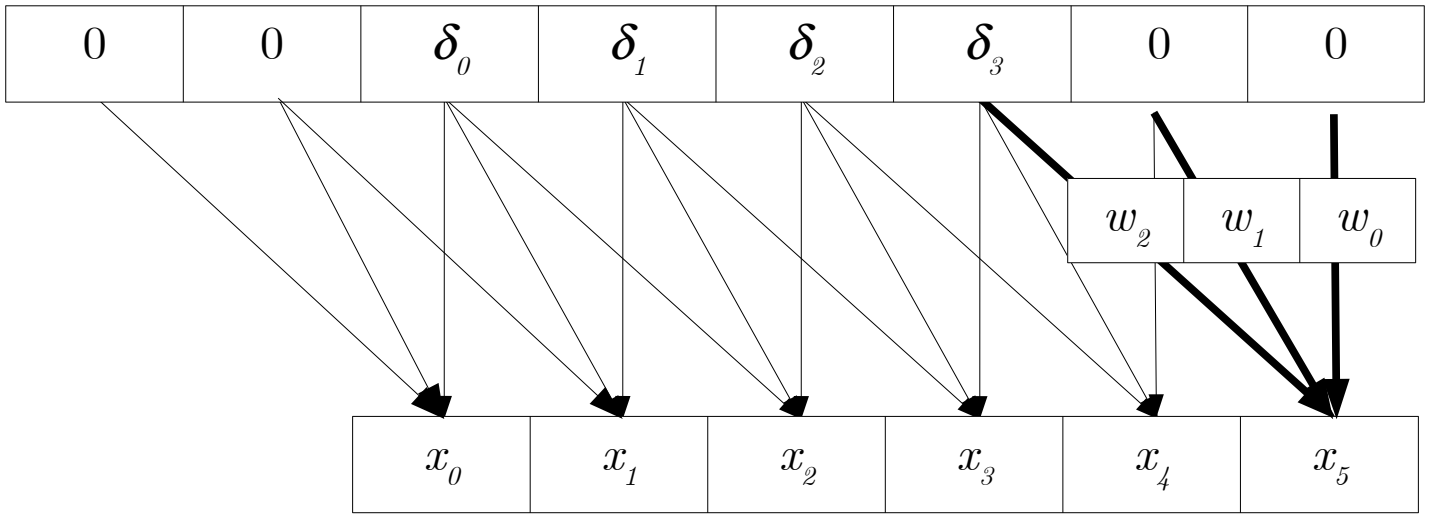


Backprop explained with 1-D convolutions

Note: zero padding again

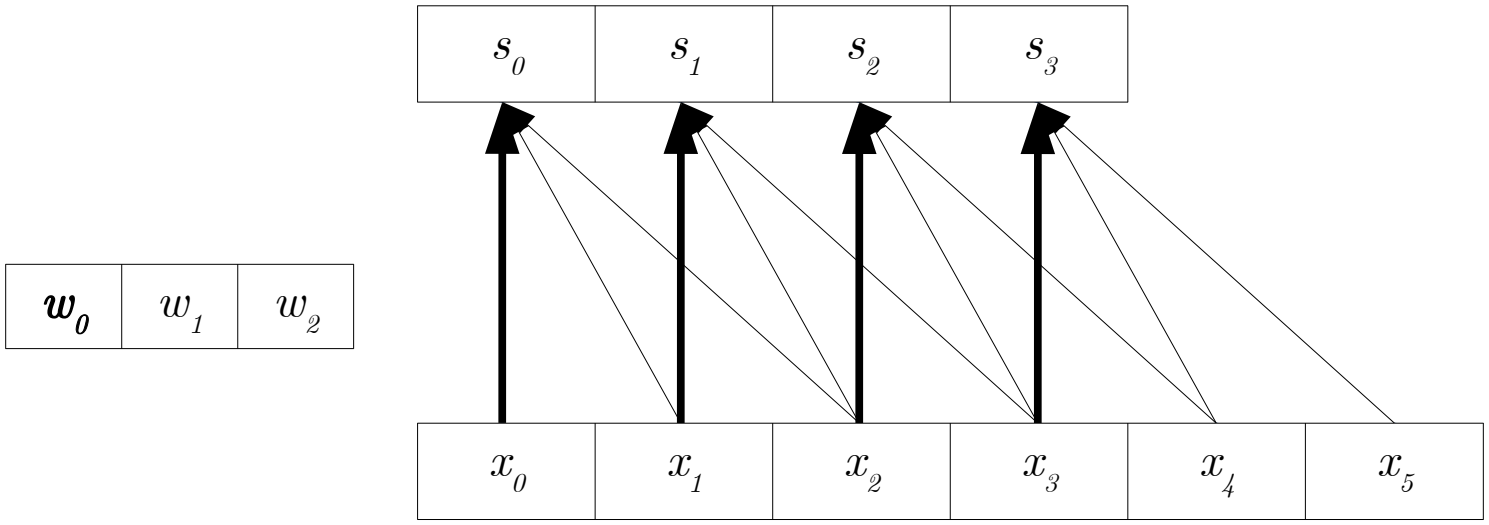


Backprop explained with 1-D convolutions

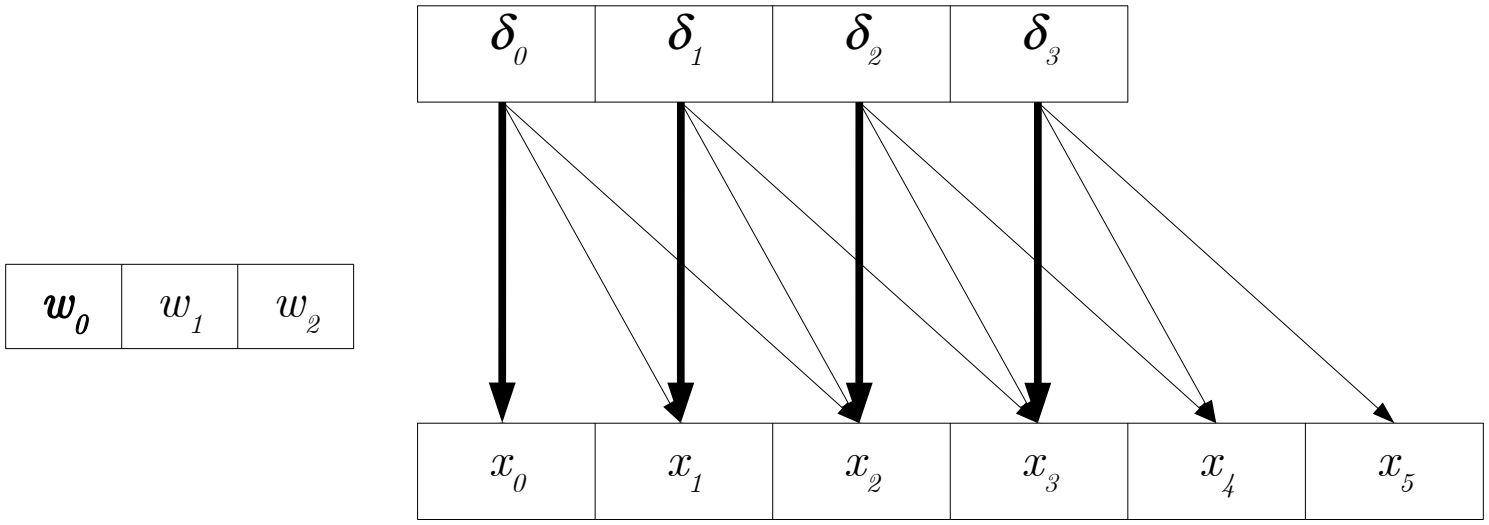


Backprop explained with 1-D convolutions: weights

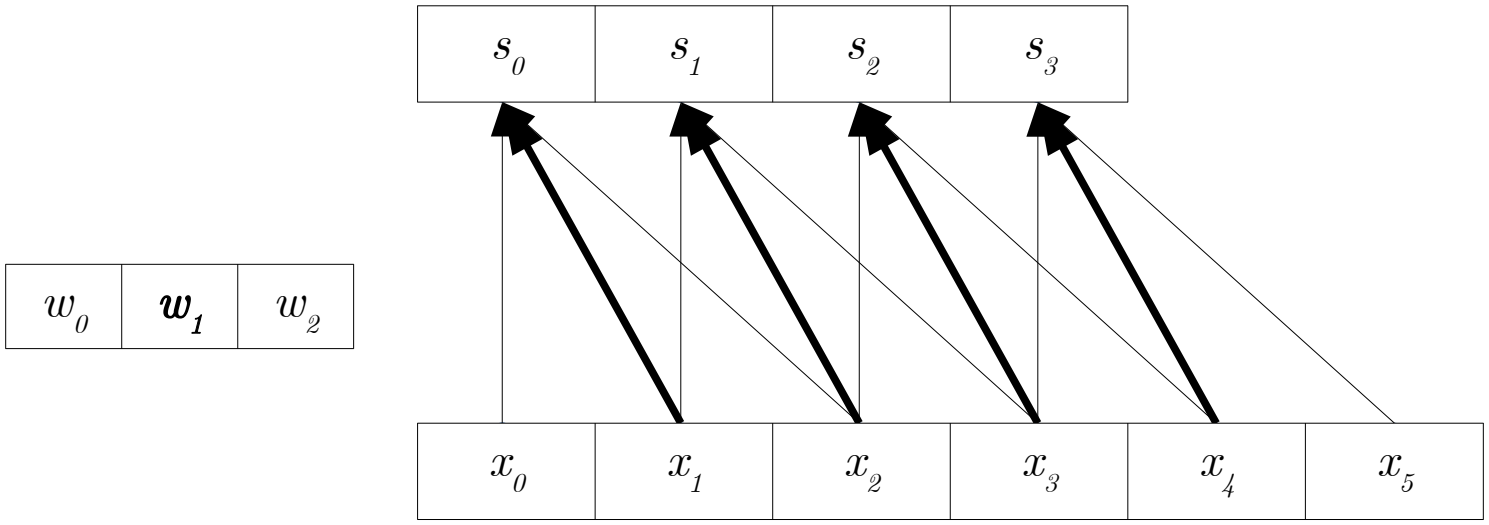
Backprop explained with 1-D convolutions: weights



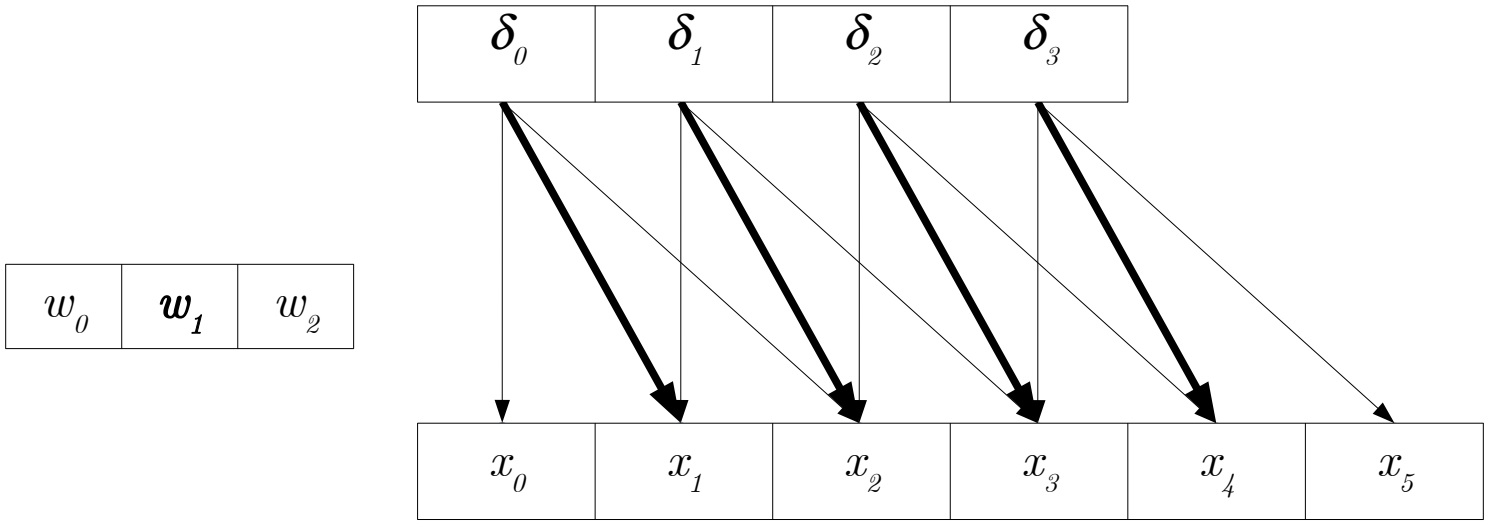
Backprop explained with 1-D convolutions: weights



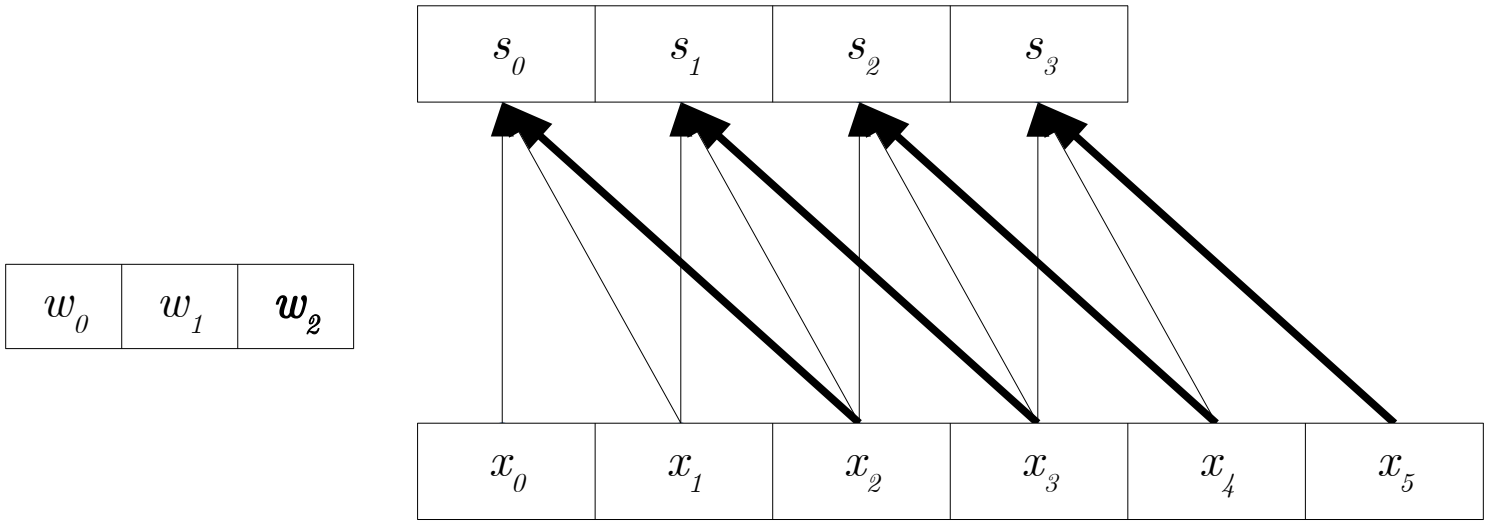
Backprop explained with 1-D convolutions: weights



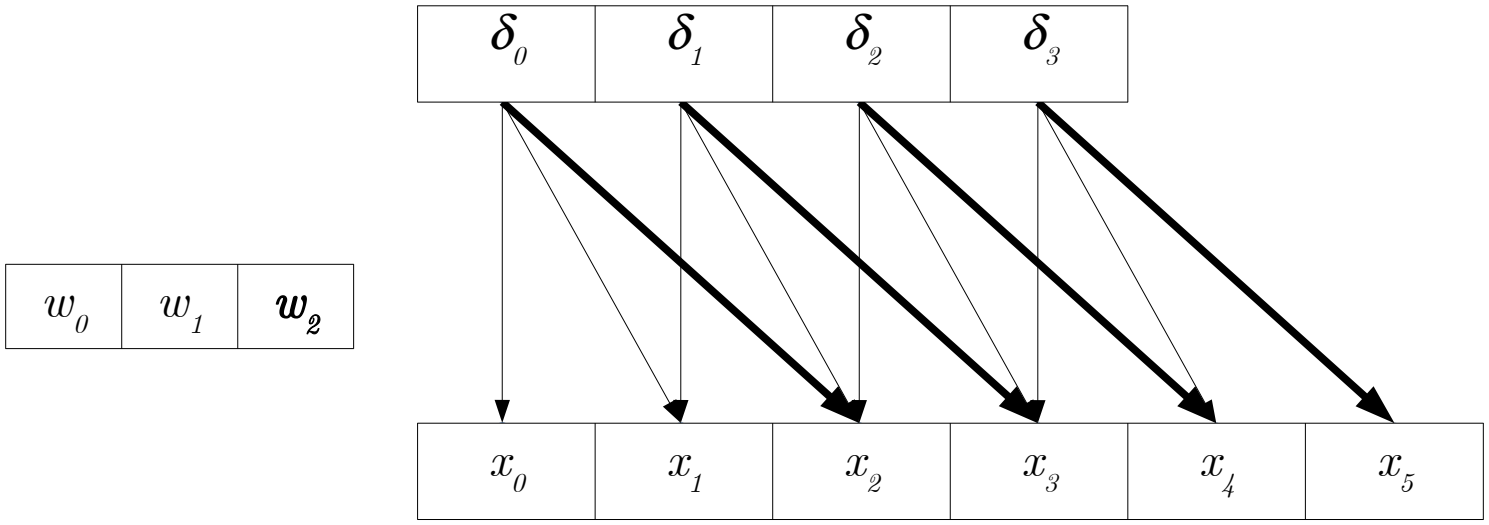
Backprop explained with 1-D convolutions: weights



Backprop explained with 1-D convolutions: weights



Backprop explained with 1-D convolutions: weights

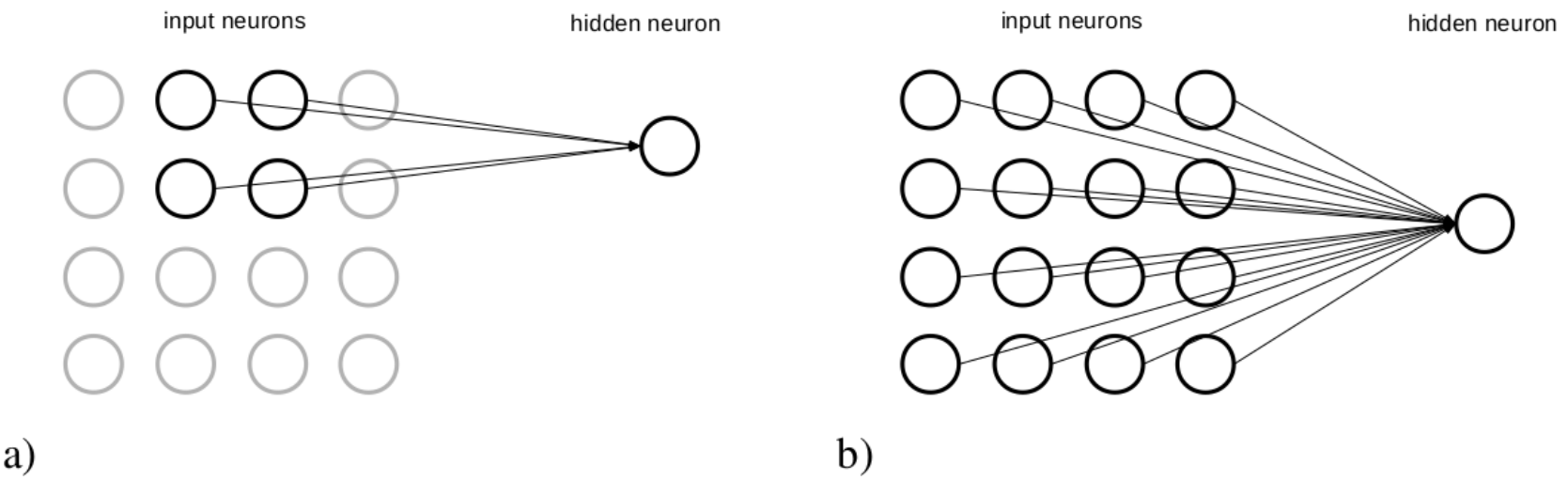


Overview: Chapter 6

- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

Local receptive fields

- Fully-connected networks: each neuron of a layer is connected to all neurons in the layer below
- Convolutional neural networks: connections in small localized regions

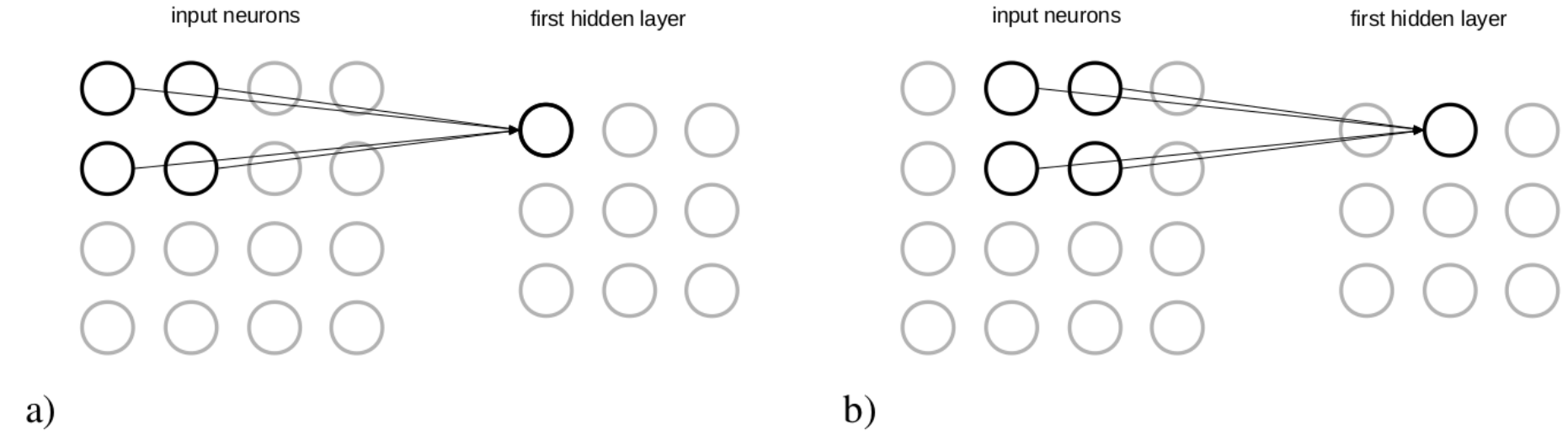


Overview: Chapter 6

- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

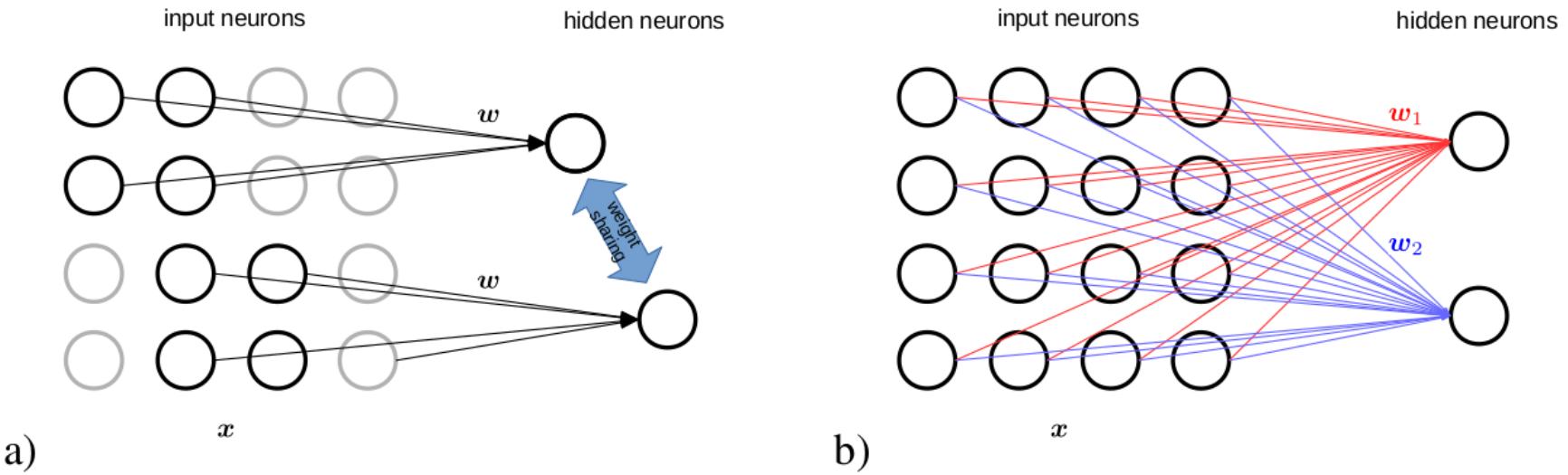
Weight sharing and shifting

- Sharing of weights across positions: the same feature is detected at multiple positions in the image



Weight sharing and shifting

- Sharing of weights across positions: the same feature is detected at multiple positions in the image

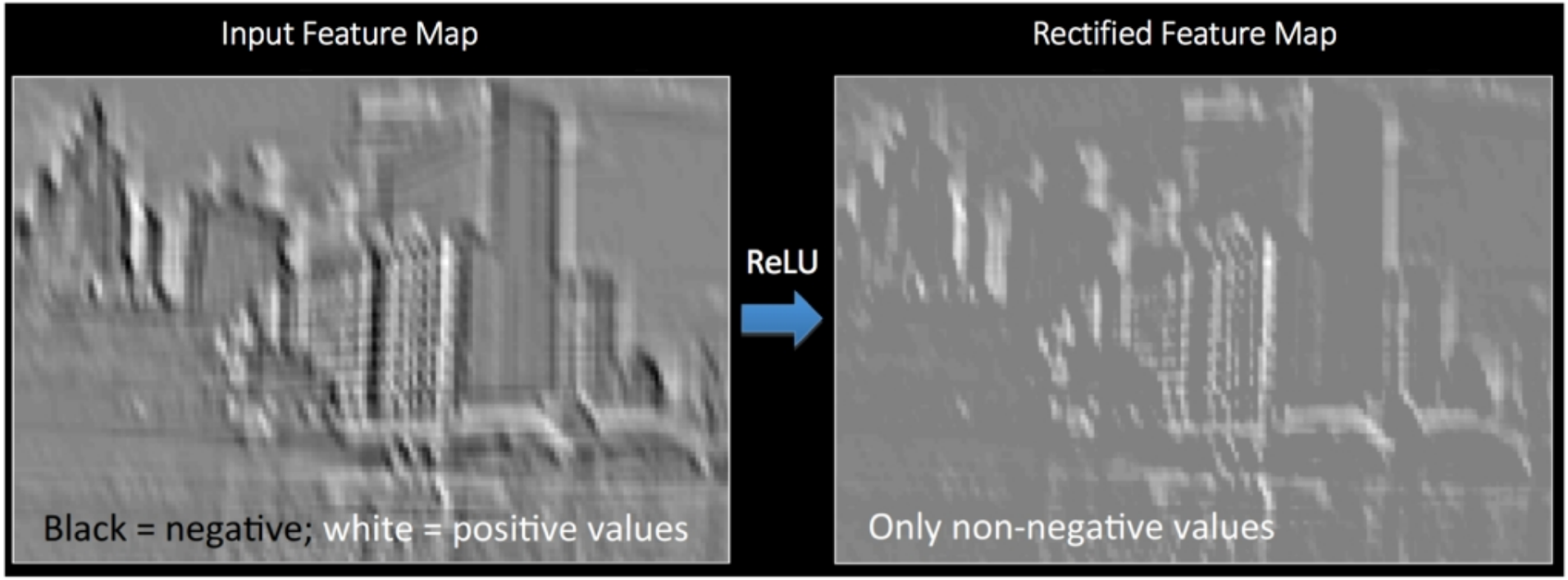


Overview: Chapter 6

- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

Non-linearities for CNNs

- Typically: ReLU, leaky ReLU, (S)ELU



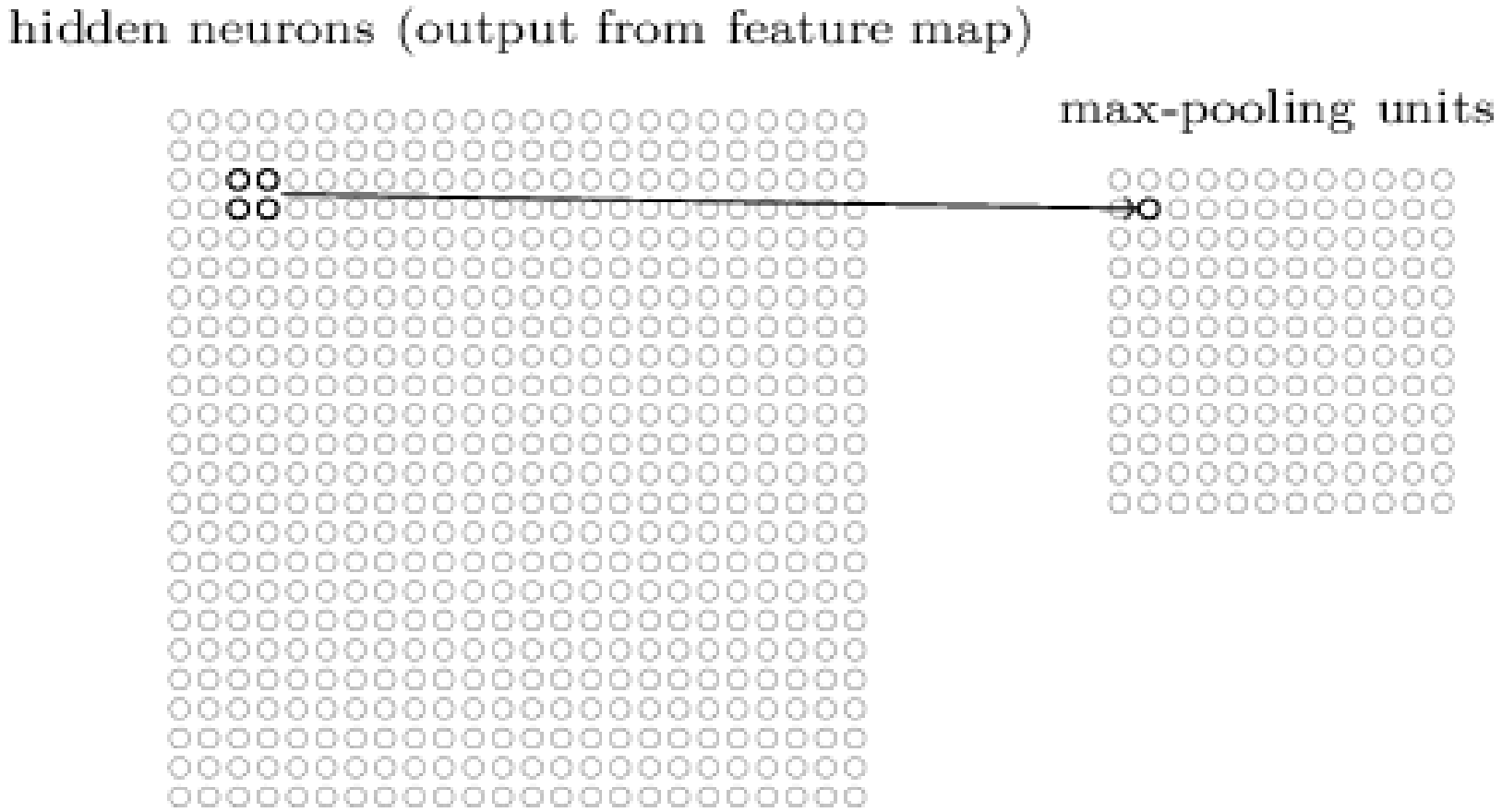
Overview: Chapter 6

- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

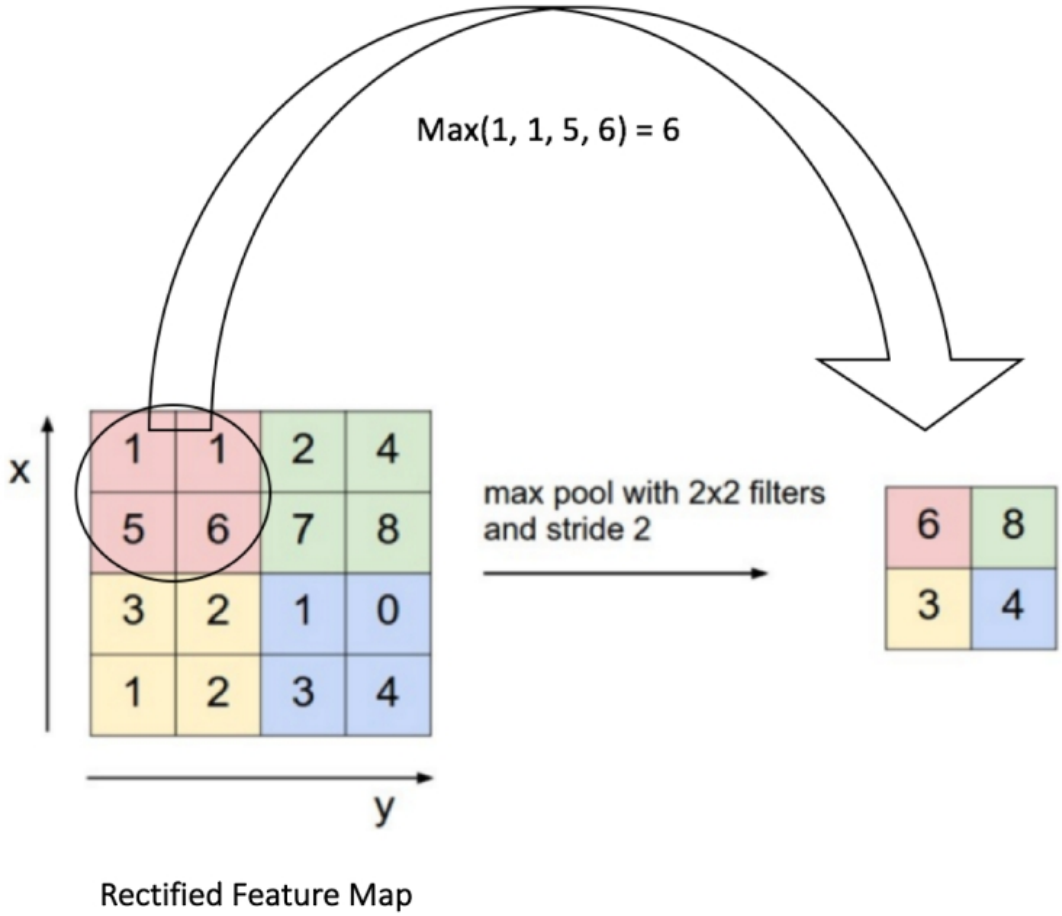
Pooling

- Reduction of multiple numbers into one
  - e.g. maximum, mean, weighted mean (attention)
- Pooling a set of nearby pixels → “zooming out”
- This can also be done for feature maps
- Provides some local invariance
- Reduces computational effort

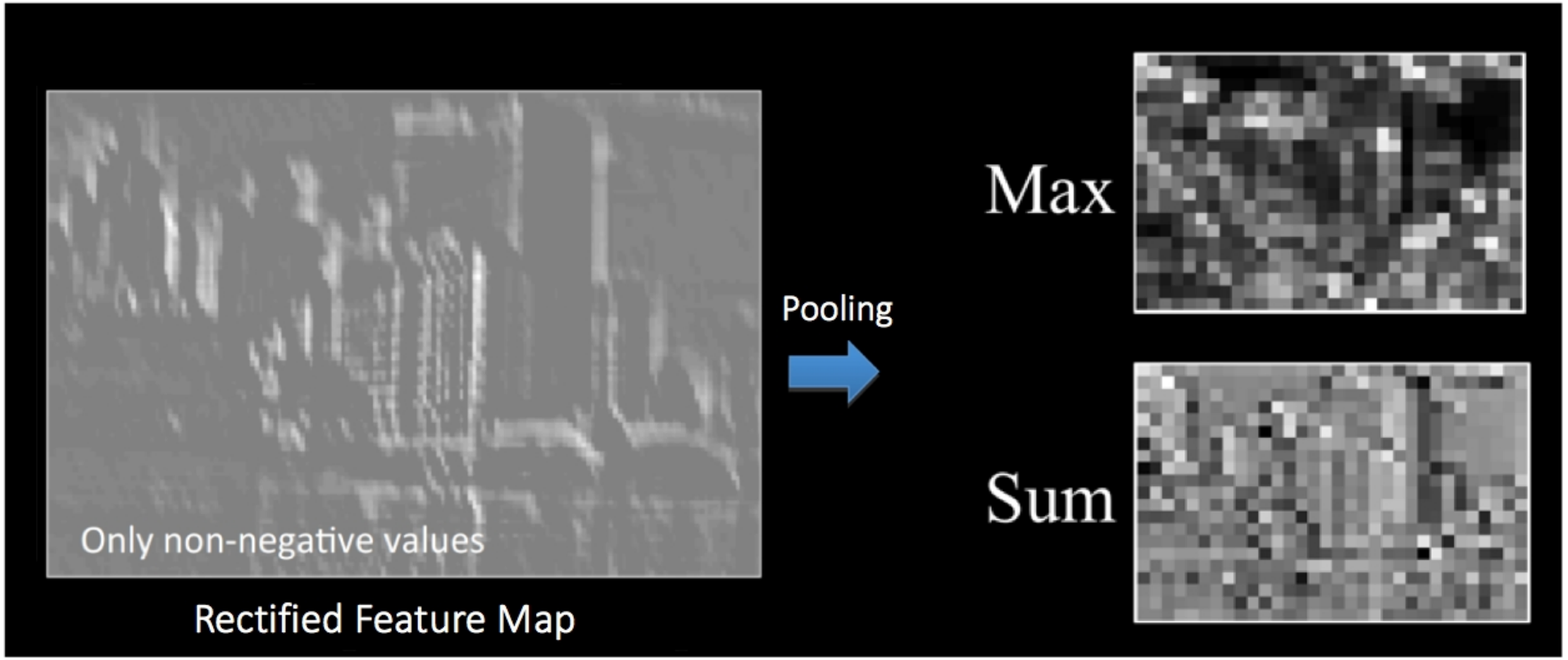
CNNs: The max-pooling operation



CNNs: The max-pooling operation



CNNs: The max-pooling operation

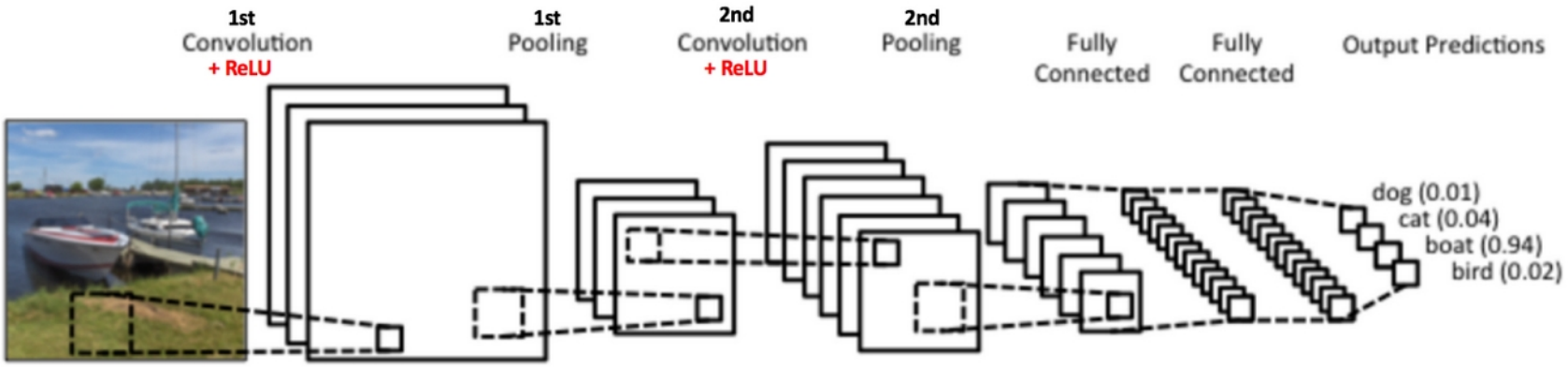


Overview: Chapter 6

- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

Recap: CNNs

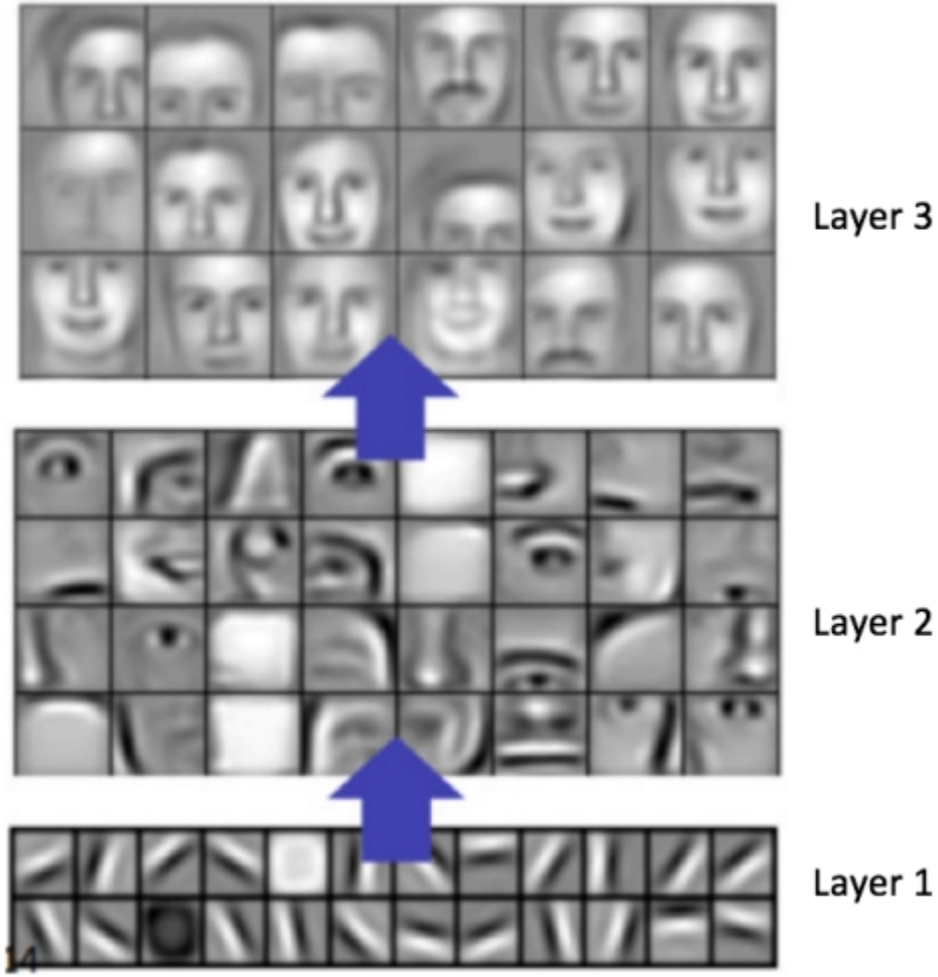
- Typically multiple steps of convolution and pooling
- Recently: strided convolutions instead of pooling



Recap: CNNs

- Kernel: weights of a convolutional layer; can be interpreted as feature detectors;
- Filter: used interchangeably with kernel
- Kernel size: size of the weight matrix; often 3x3 or 5x5
- Shared weights: For 2D convs, each neuron in the output has C R^2 weights, where C is the number of channels (or feature maps) in the input. These weights are shared by the output units.
- Stride: A filter can also be shifted by two pixels at once. The size of this shifting is called stride.
- Padding: Adding zeros to the columns or rows of an image.
- Feature map: Output of a filter applied to the input;
- Convolutional layer: A layer in a CNN that applies one or more conv operations to a structured input object; typically has several kernels

A simple convolutional neural network: learned features in different layers



Remark 1: Size of feature maps

- The dimensions of the feature maps are:

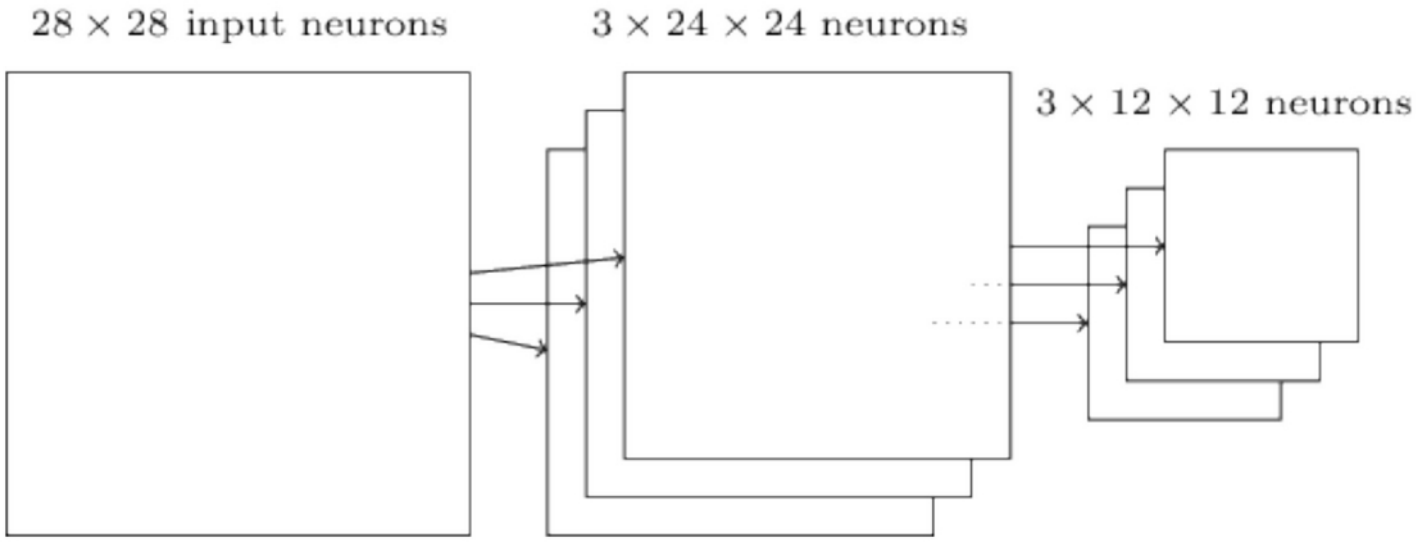
$$A^{[l+1]} \times B^{[l+1]} \times D^{[l+1]},$$

with $A^{[l+1]} = (A^{[l]} - R + 2P)/S + 1$ and $B^{[l+1]} = (B^{[l]} - R + 2P)/S + 1$, where

- $A^{[l]}$ is the height of the feature maps or the input image in the previous layer
- $B^{[l]}$ is the width of the feature maps or the input image in the previous layer
- P is the amount of (symmetric) zero padding,
- R is the kernel size,
- S is the stride, and
- $D^{[l+1]}$ is the number of output feature maps of the convolutional layer.

Remark 2: Multiple filters

- We want to find different features
  - Thus, we need multiple filters
  - Each filter generates a feature map



Remark 3: Images with depth

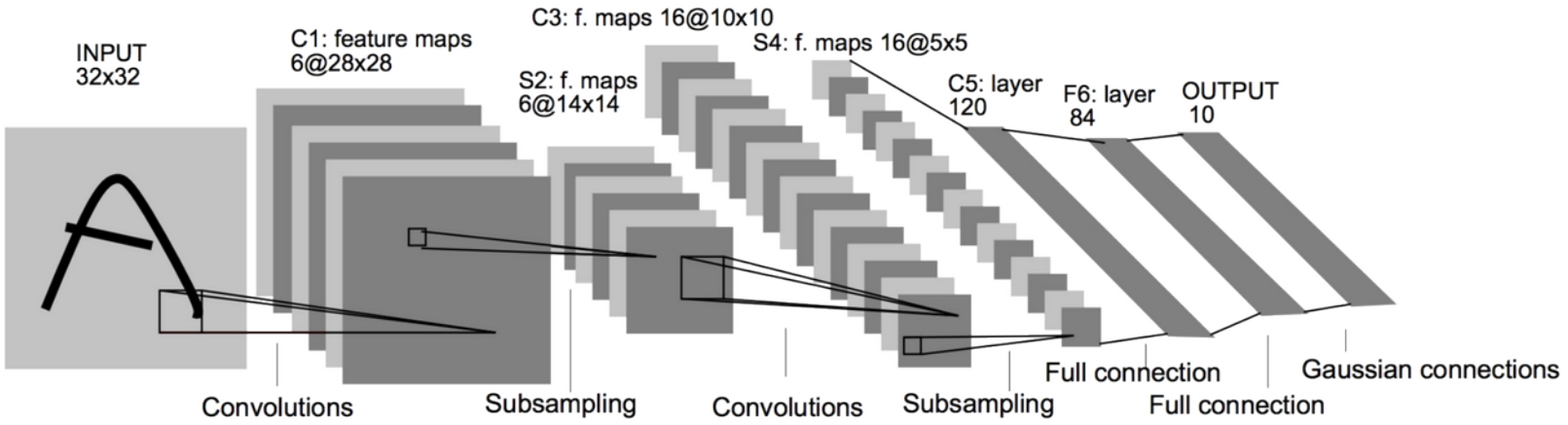
- Typically, we have 3 (RGB) input channels
  - Grayscale: 1 channel
  - Channels: additional dimension (W, H, C)
- In hidden conv layers, feature maps correspond to channels
- How do we deal with it with CNNs?

$$s_{a,b} = \sum_{c=0}^{C-1} \sum_{i=0}^{R-1} \sum_{j=0}^{R-1} w_{i,j,c}\, x_{a+i,b+j,c} = (\boldsymbol{W} * \boldsymbol{x})_{a,b}$$

Overview: Chapter 6

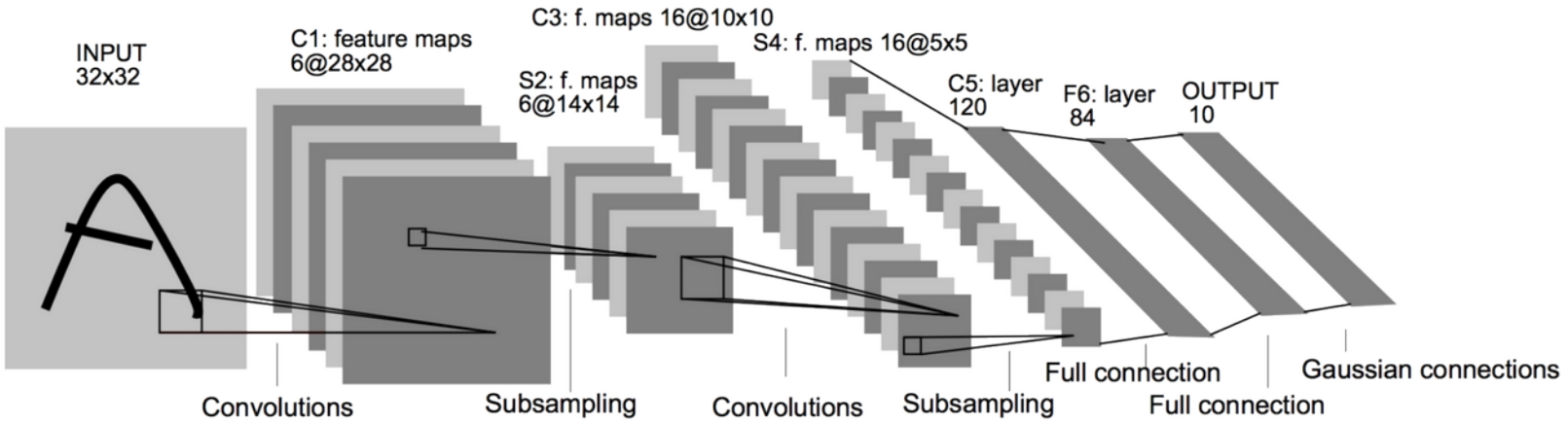
- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

Example: LeNet



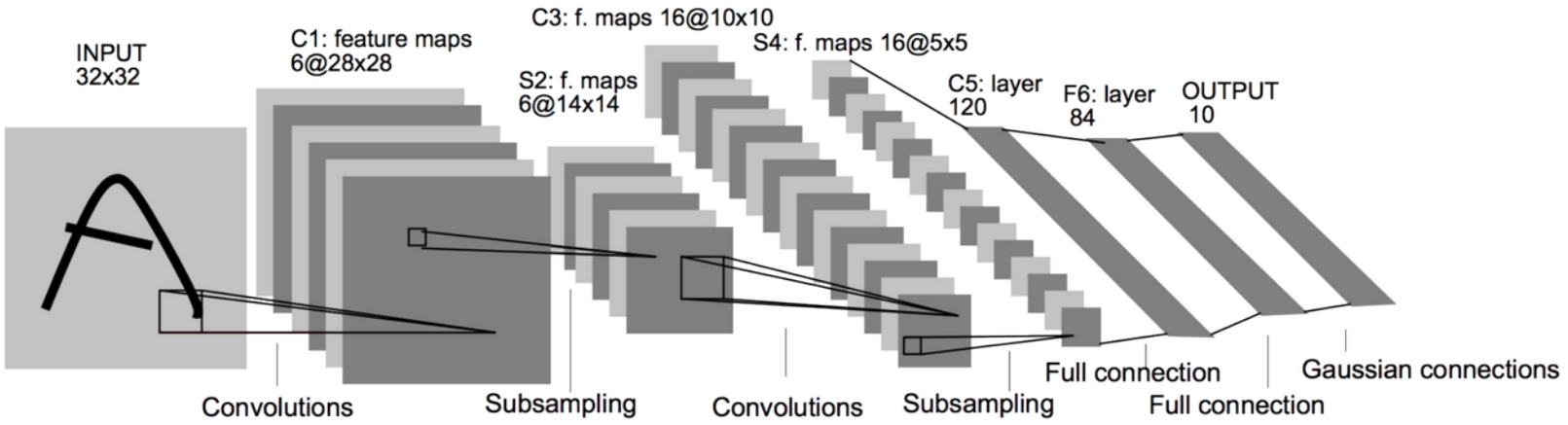
- **Input:** gray scale image of size 32x32 or 32x32x1
- **C1:** ConvLayer with 6 kernels of size (5,5) and stride 1
  - Output dimensions are: 28x28x6
  - Number of trainable parameters: (1*5*5+1)*6=156
- **S2:** Pooling with size (2,2) and stride 2.
  - Output dimensions are: 14x14x6

Example: LeNet



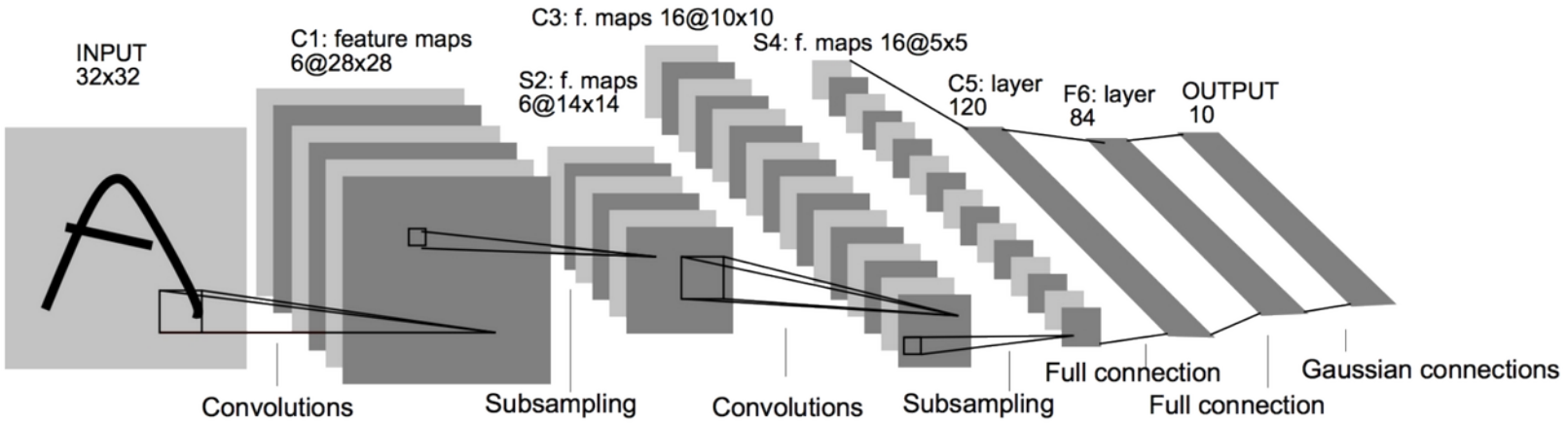
- **C3:** ConvLayer with 16 kernels of size (5,5) and stride 1
  - Output dimensions are: 10x10x16
  - Number of trainable parameters: (6*5*5+1)*16=2416
- **S4:** Pooling with size (2,2) and stride 2.
  - Output dimensions are: 5x5x16

Example: LeNet



- **C5:** ConvLayer with 120 kernels of size (5,5) and stride 1
  - Output dimensions are: 1x1x120
    - Practically means “full-connection”
  - Number of trainable parameters: (16*5*5+1)*120=48120
- **F6:** Fully-connected layer with 84 units.
  - Number of trainable parameters: (120+1)*84

Example: LeNet

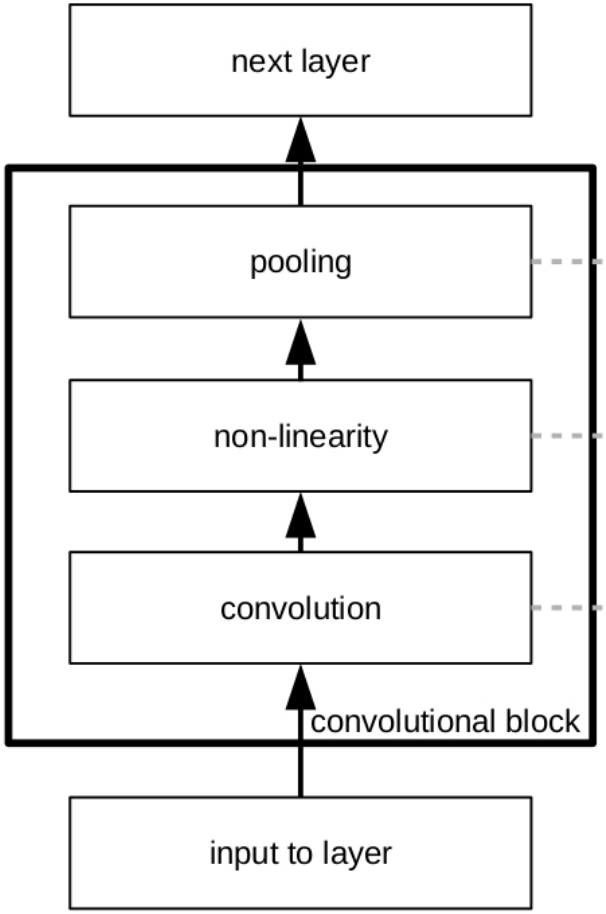


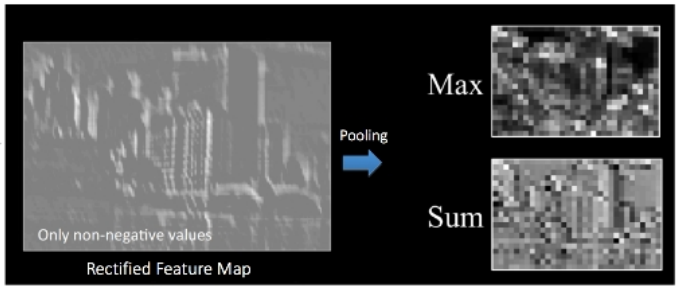
- **Output:** Fully-connected layer with 10 units.
  - Number of trainable parameters: (84+1)*10

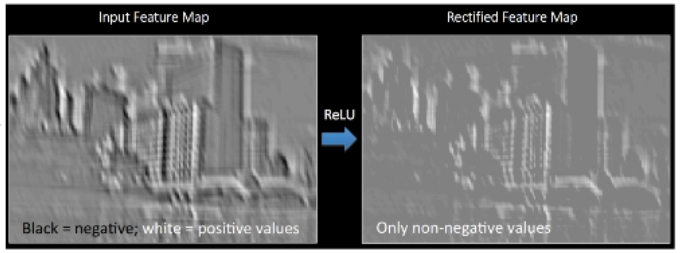
Remark: CNNs reduce number of weights

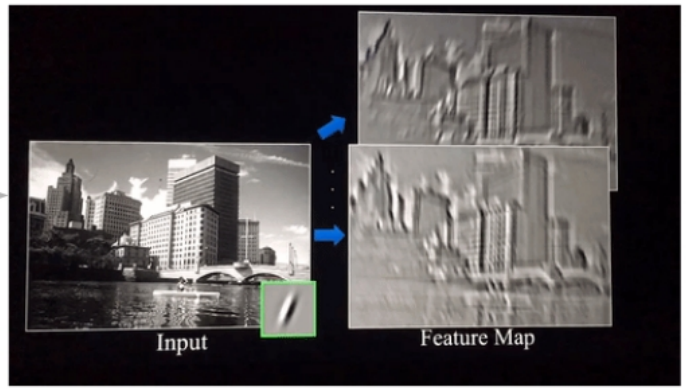
- First layer in LeNet required (5*5+1) parameters per kernel
  - 20 neurons/kernels: 520 weights (26*20)
- First layer in a fully-connected network would require 28*28 per neuron
  - 20 neurons: 17,700 weights (784*20+20)

Convolutional blocks: Core elements of a CNN





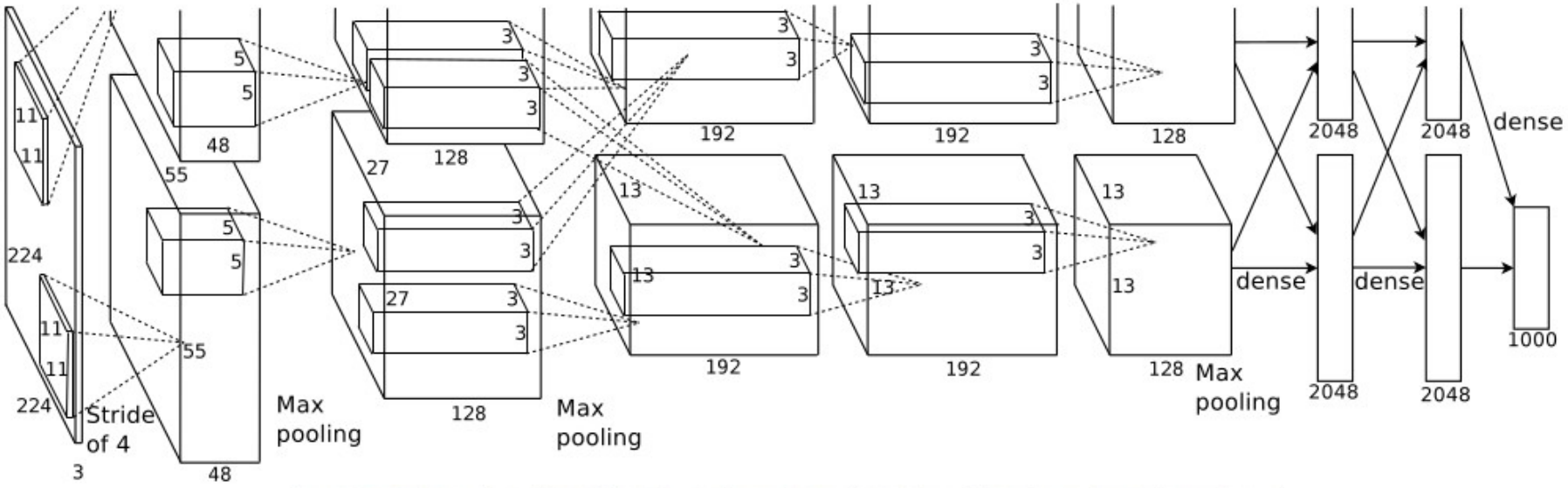


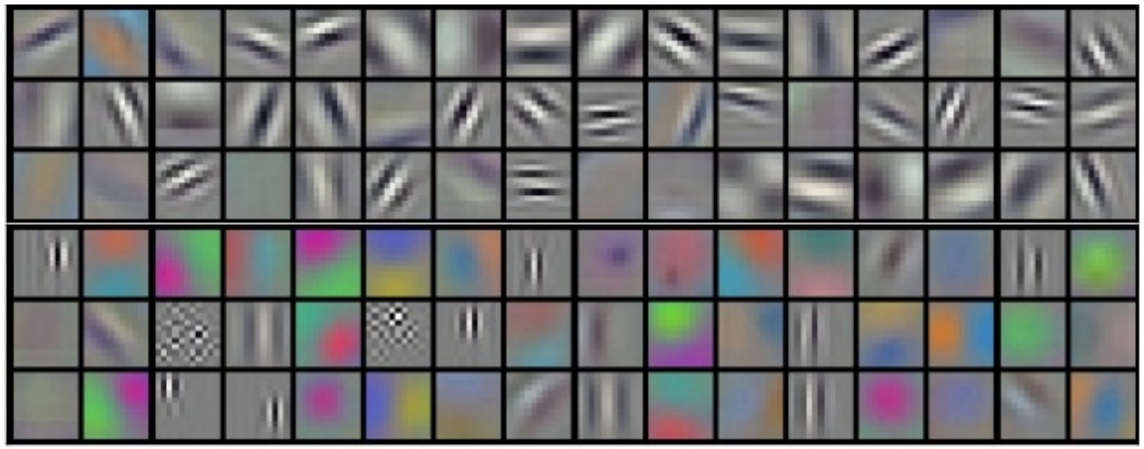


Outlook: Advanced CNN architectures (1/2)

- Currently, deep CNN networks are used
  - e.g. ResNet with 150 layers
- Special techniques are necessary to counter the VGP

Outlook: Advanced CNN architectures (2/2)



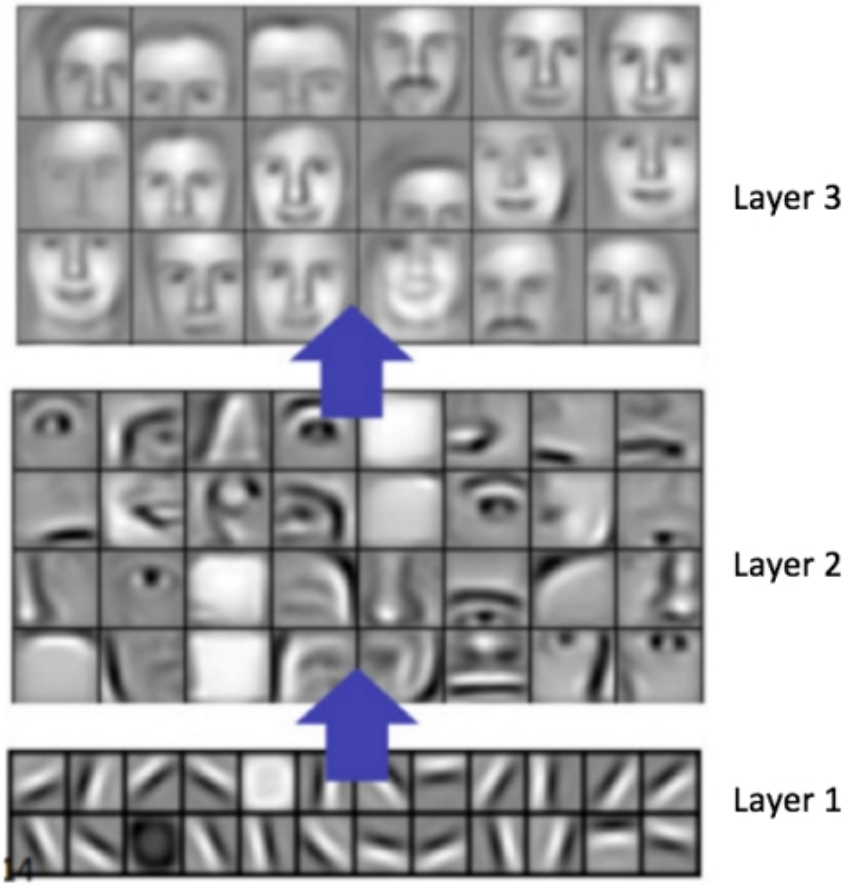


Overview: Chapter 6

- 6.1. Image data and their characteristics
- 6.2. The convolution operation
- 6.3. Backpropagation through conv
- 6.4. Local receptive fields
- 6.5. Shared weights and biases
- 6.6. Non-linearities for CNNs
- 6.7. Pooling
- 6.8. Recap
- 6.9. Example: LeNet
- 6.10. Convolutional blocks
- 6.11. Visualizing and understanding CNNs

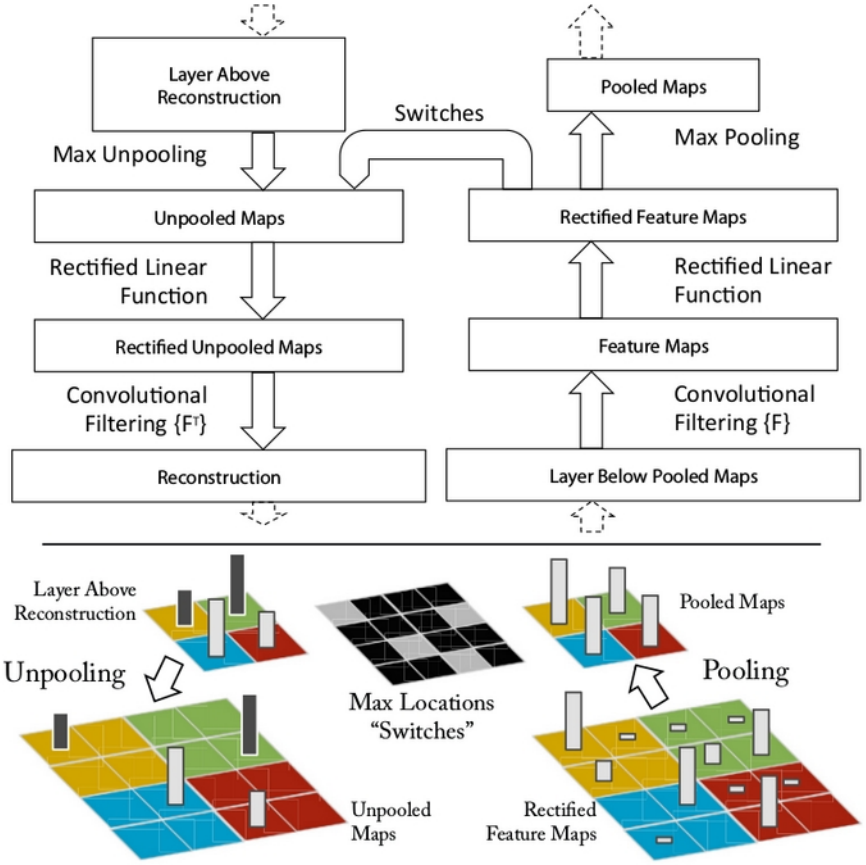
Visualizing and understanding CNNs

- Increasingly abstract features through layers
- Resemblance to biological neurons in visual cortex
- CNNs can be considered as feature extractors



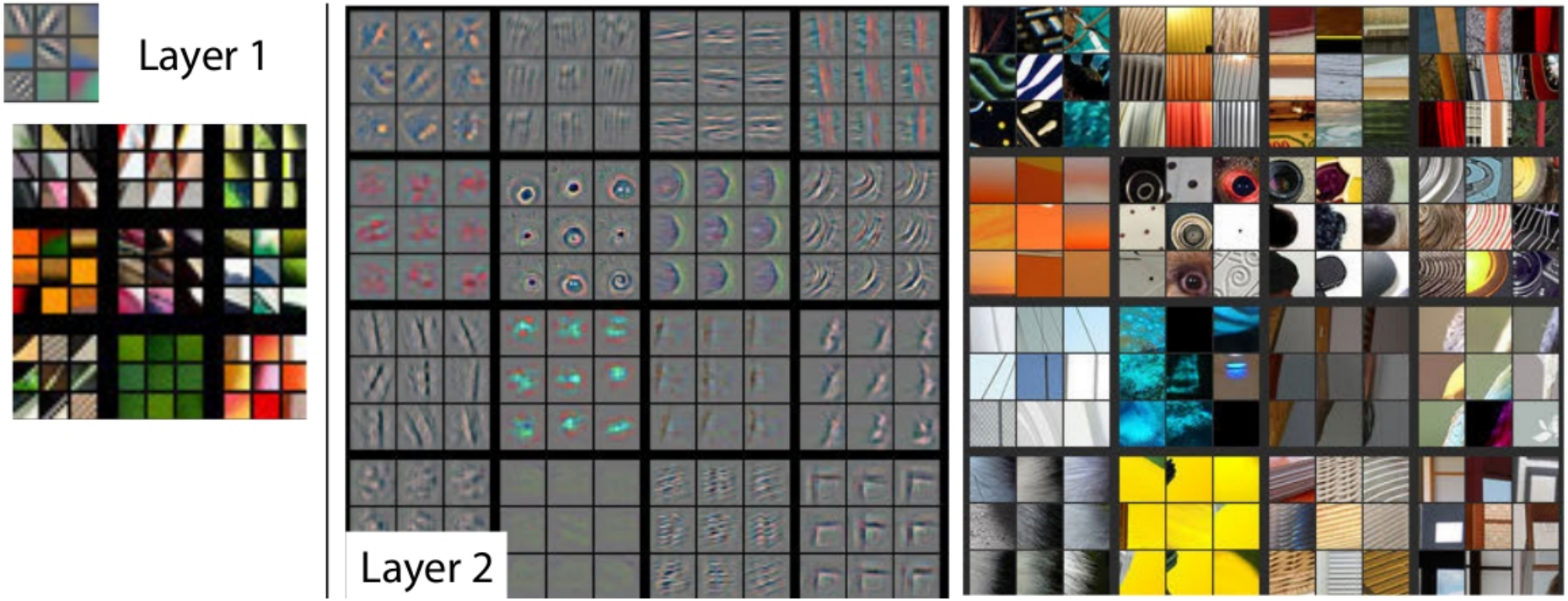
Deconv net for visualizing and understanding CNNs

- Examination: all activations except one set to zero
- Pass the activation to the deconv layer
  - Unpooling: switches
  - Rectify: ReLU
  - Filter: deconv (transposed version of filters)
- This reconstructs activity in layer below that gave rise to this activation
- Similar to backprop for a single strong activation
- Continue process until input layer



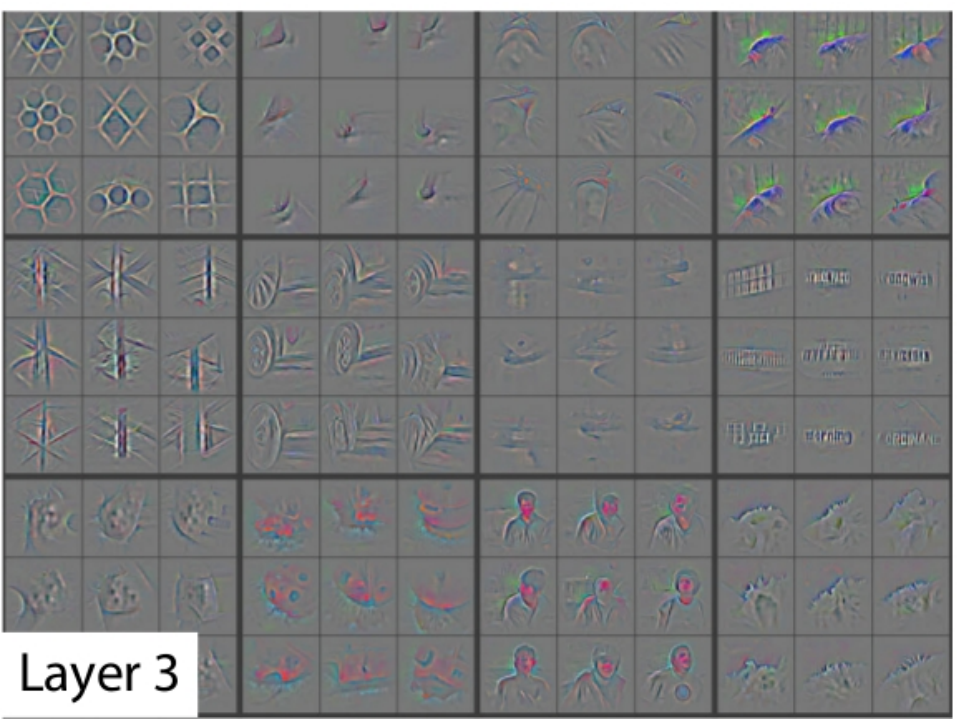
Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

Visualizing and understanding CNNs



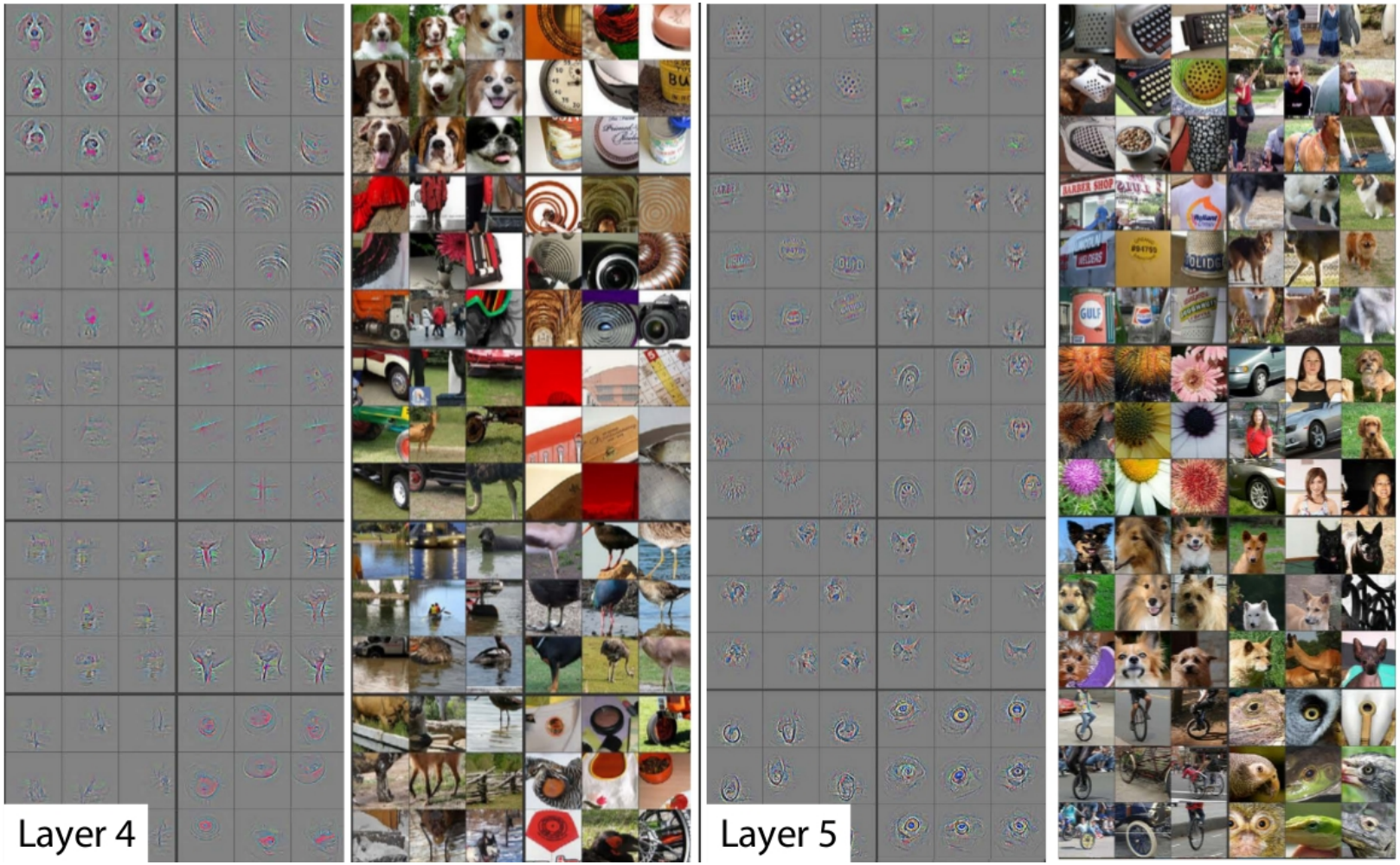
Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

Visualizing and understanding CNNs



Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

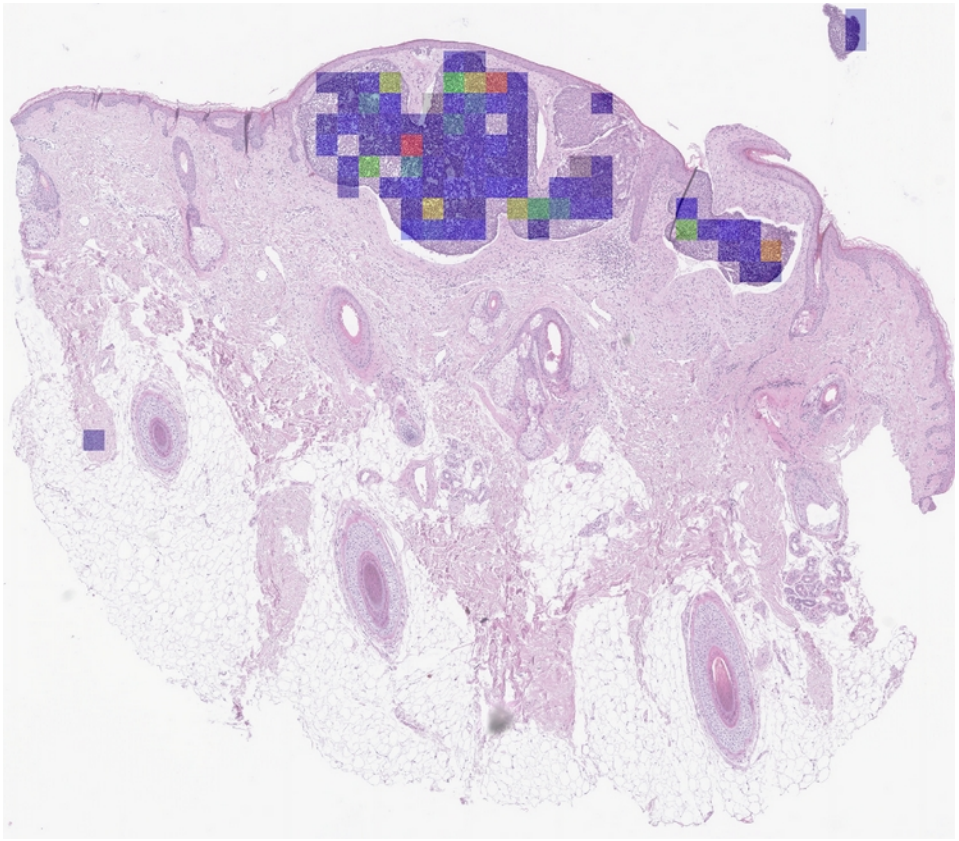
Visualizing and understanding CNNs



Applications of CNNs – a non-exhaustive list

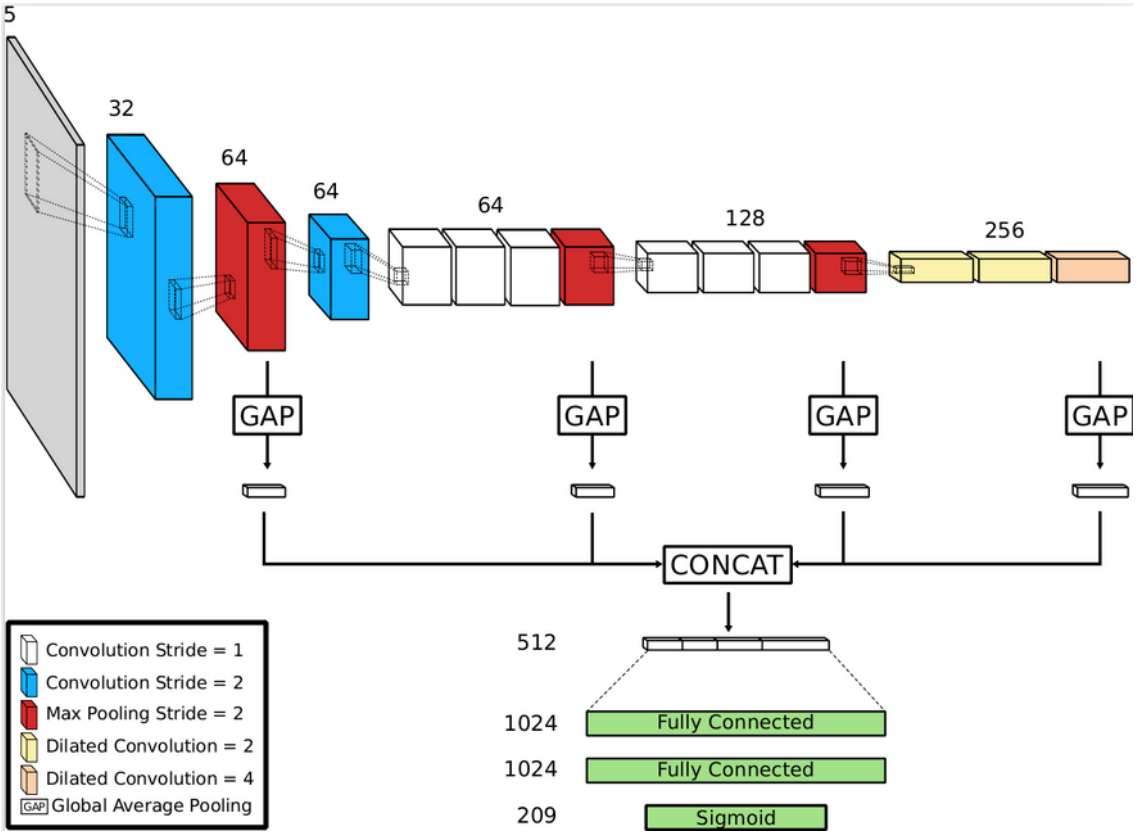
- Computer vision
  - Image recognition
  - Face recognition
  - Scene labeling
  - Action recognition
  - Human pose estimation
  - Object detection
  - Semantic segmentation
- ...
- Natural language processing
  - Speech recognition
  - Text classification
- Medicine/healthcare
  - Medical imaging
- Molecular biology
- ...

Recent works on CNNs at IML and LIT AI LAB



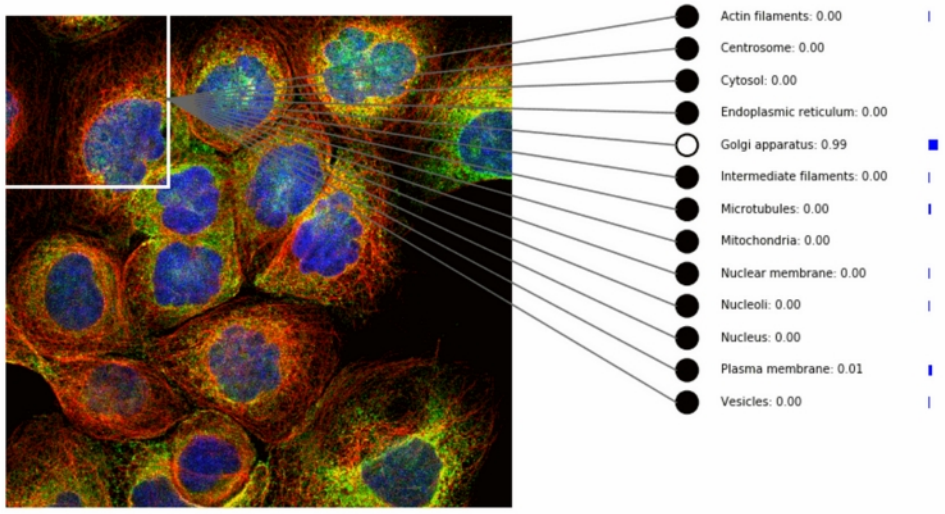
Kimeswenger, S., Rumetshofer, E., Hofmarcher, M., Tschandl, P., Kittler, H., Hochreiter, S., ... & Klambauer, G. (2019). Detecting cutaneous basal cell carcinomas in ultra-high resolution and weakly labelled histopathological images. Machine Learning For Health workshop at NeurIPS 2019. arXiv preprint arXiv:1911.06616.

Recent works on CNNs at IML and LIT AI LAB: GapNet



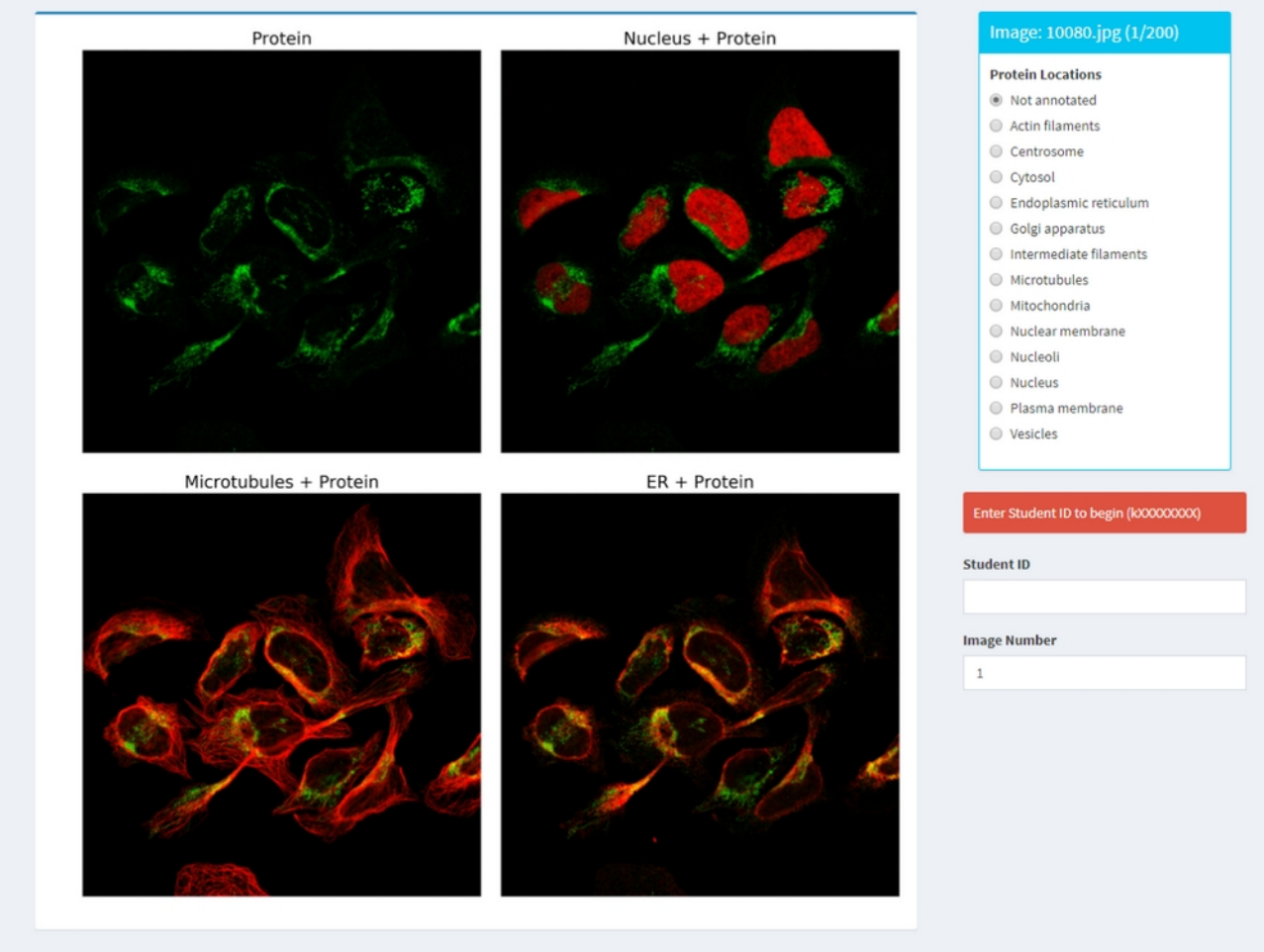
Rumetshofer, E., Hofmarcher, M., Röhrl, C., Hochreiter, S., & Klambauer, G. (2018). Human-level Protein Localization with Convolutional Neural Networks. *ICLR 2019*.

Recent works on CNNs at IML and LIT AI LAB

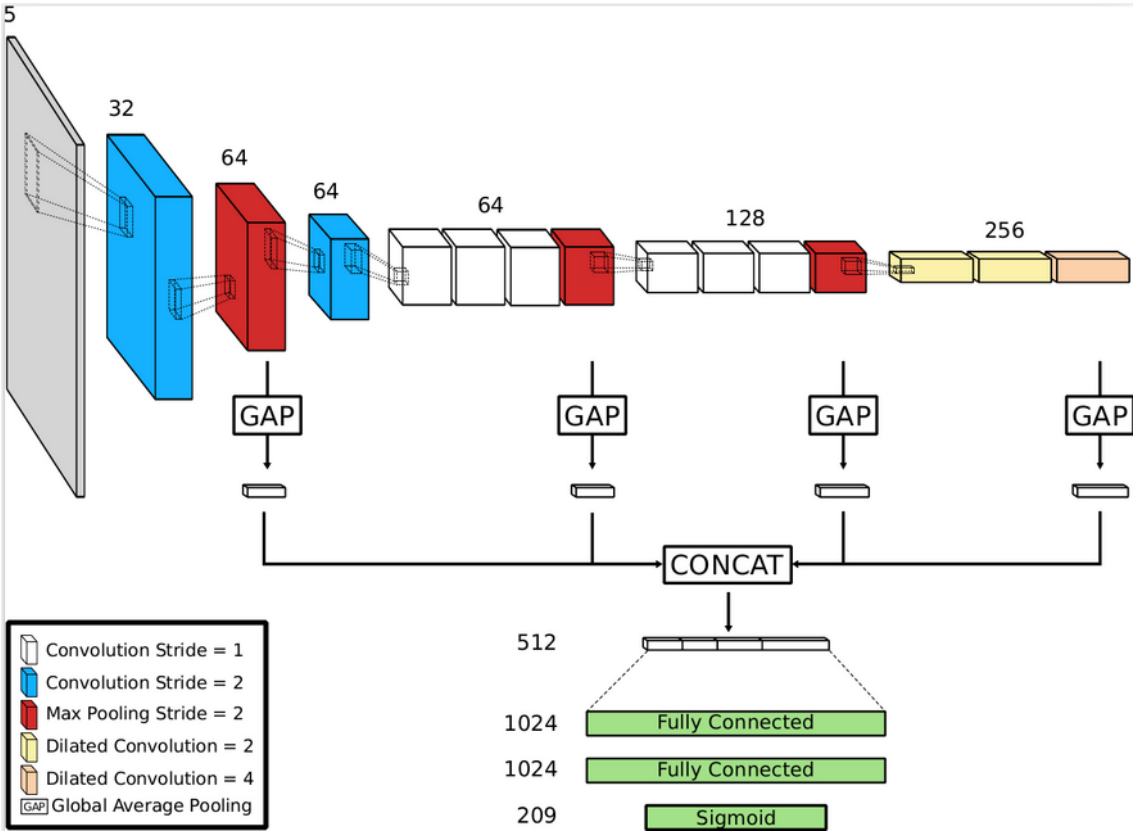


Rumetshofer, E., Hofmarcher, M., Röhrl, C., Hochreiter, S., & Klambauer, G. (2019). Human-level Protein Localization with Convolutional Neural Networks. International conference on learning representations (ICLR) 2019.

Finding a protein in cell compartments is difficult...



GapNet-PL network



Rumetshofer, E., Hofmarcher, M., Röhrl, C., Hochreiter, S., & Klambauer, G. (2018). Human-level Protein Localization with Convolutional Neural Networks. *ICLR 2019*.

Outperforms other networks...

Method	F1 Score	p-value	Precision	Recall	AUC
GapNet-PL	**0.78 ± 0.09**		0.84	**0.75**	**0.98 ± 0.02**
M-CNN	0.75 ± 0.12	0.0232	**0.90**	0.66	0.97 ± 0.02
DenseNet-121	0.73 ± 0.12	0.0434	0.75	0.74	0.97 ± 0.02
DeepLoc	0.52 ± 0.28	0.0009	0.79	0.45	0.91 ± 0.07
FCN-Seg	0.50 ± 0.18	0.0007	0.63	0.44	0.71 ± 0.08
Convolutional MIL	0.47 ± 0.31	0.0007	0.71	0.40	0.94 ± 0.06
Liimatainen et al. [2][1]	0.51	-	0.45	0.68	-

[1] performance estimate calculated on a different test dataset

Table: Performance comparison of different CNN architectures for protein localization. For each method, the table reports the average F1 score and its standard deviation across 13 prediction tasks. p-value reports the result of a one-sided paired Wilcoxon test with the h_0 that GapNet-PL and the respective method perform equally across prediction tasks.

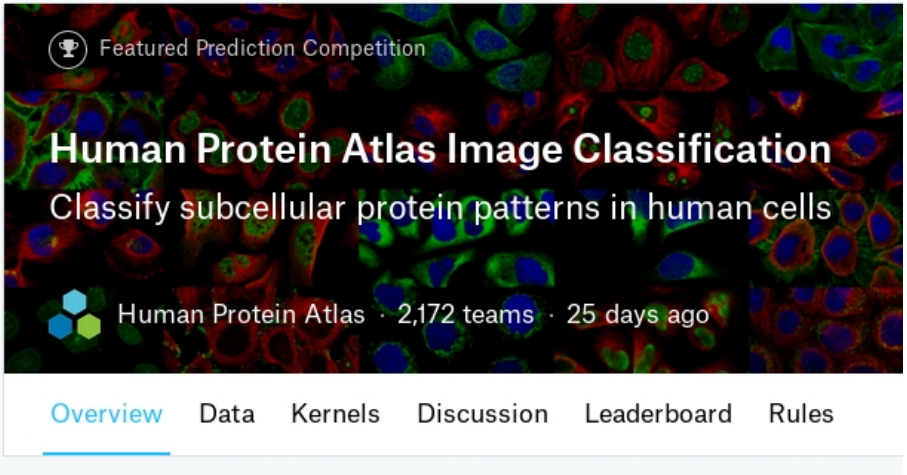
GapNet-PL against human experts....
... and scholars

GapNet-PL against human experts.... ... and scholars

Method	Accuracy	F1 Score	Precision	Recall
GapNet-PL	**0.91**	**0.82**	**0.75**	**0.95**
Expert 1	0.72	0.57	0.55	0.67
Expert 2	0.66	0.49	0.44	0.64
Expert 3	0.64	0.58	0.73	0.58
Mean Scholars	0.51	0.36	0.39	0.46
Ensemble of experts	0.71	0.60	0.60	0.68
Ensemble of scholars	0.71	0.57	0.60	0.65

Table: Predictive performance of GapNet-PL, 3 human experts and the group of scholars. The table shows the overall accuracy and average F1 score, precision and recall over 13 tasks. In each column, the best performing method is written bold.

Kaggle Challenge: Gold medal with our method



GAPNet

Immediately, reading the GAPNet paper, I had the idea to change the illustrated architecture to use a pretrained backbone instead.

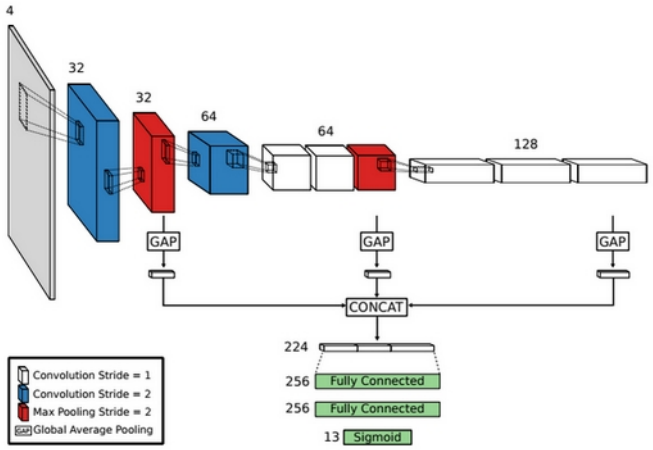


Figure 4: GapNet-PL architecture

I think one important advantage of the GAPNet architecture is its ability for multiscale. So I tried different backbones and ended up with ResNet18, which also enabled to use a batchsize of 32 on a GTX1080Ti. I also saw minor improvements adding SE-Blocks before the Average Pooling layers with nearly no computational cost, so I added those. I saw no improvement in using RGBY images. I used a weighted bce and f1 loss and a cosine annealing lr schedule and only trained for 20 epochs. After applying our thresholding method to the predictions GAPNet trained on 512x512 RGB images also using the HPA external data a single 5-fold model was able to achieve 0.602 Public LB. I also experimented with different internal/external data proportions, RGBY and 512cropping from 1024 images so I had 4 5-fold GAPNet models which I could ensemble resulted in LB 0.609

Summary

- CNNs are one of the core elements of the current AI success
- We got to know the *convolution operation*
- Main operation in a convolutional layer
  - Contains a kernel/filter with adaptive weights
- Backward pass of a conv layer also uses the conv operation
- CNNs consist of convolutional blocks: (1) conv, (2) non-linearity, and (3) subsampling/pooling
- Detailed example of LeNet and MNIST