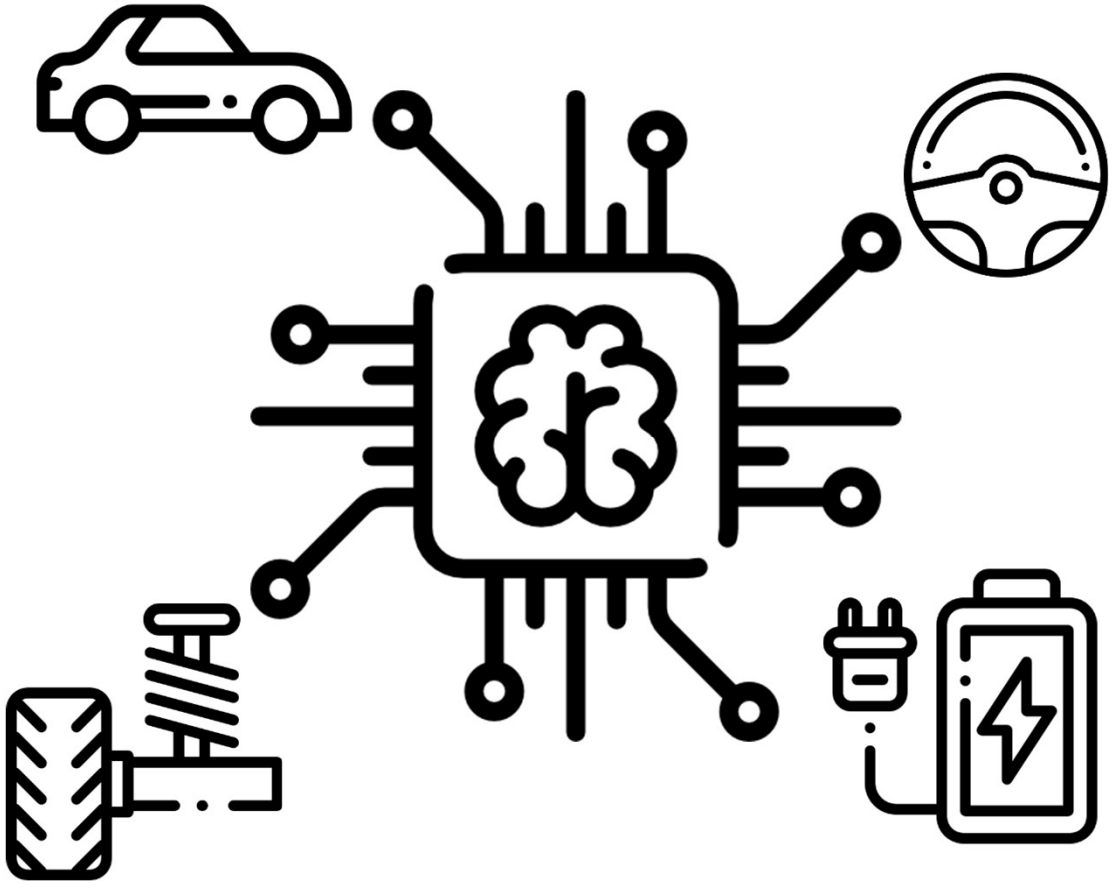
Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

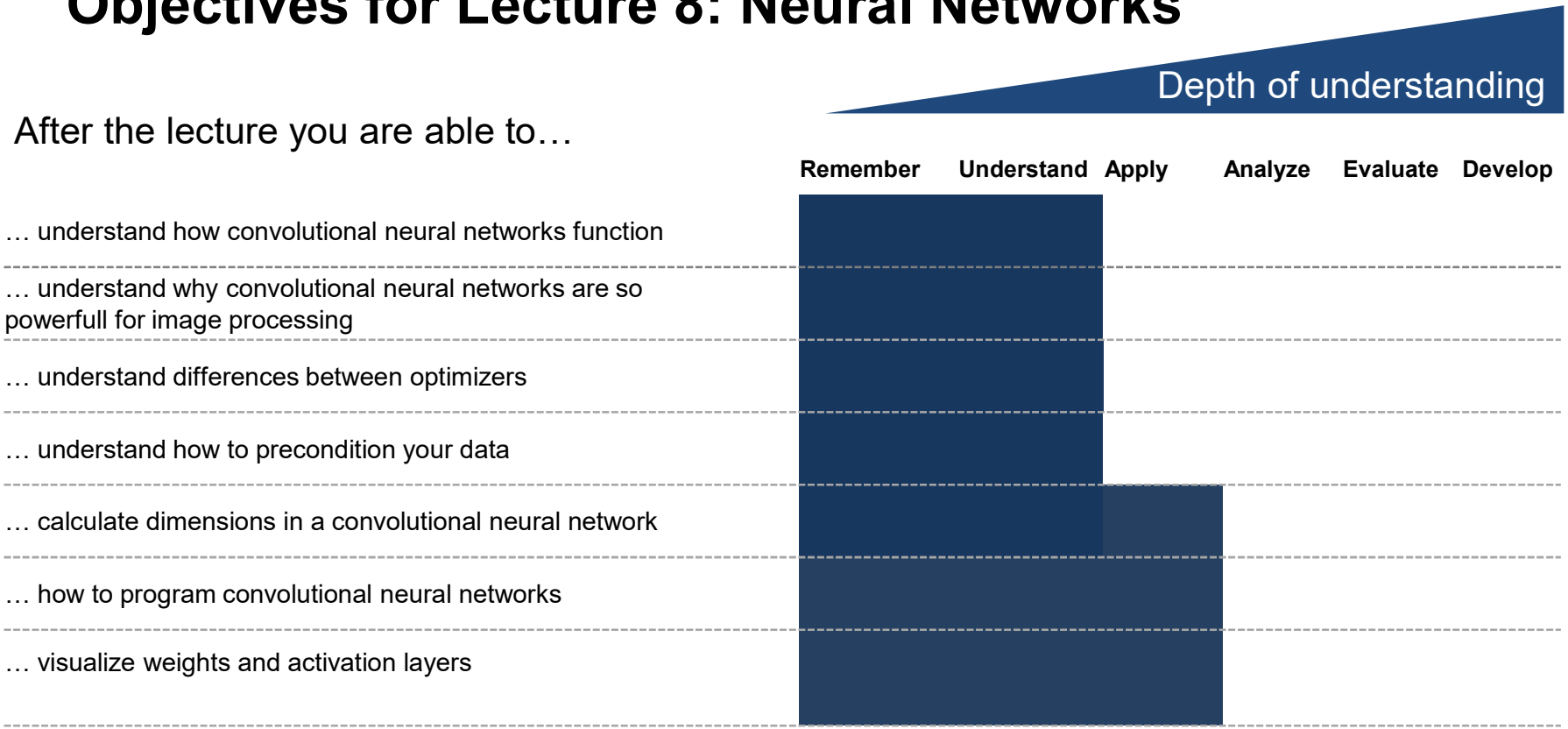
Artificial Intelligence in Automotive Technology

Johannes Betz / Prof. Dr.-Ing. Markus Lienkamp / Prof. Dr.-Ing. Boris Lohmann



Lecture 16:15 – 17:45	Practice 17:45 – 18:30
1 Introduction: Artificial Intelligence 17.10.2019 – Johannes Betz	**Practice 1** 17.10.2019 – Johannes Betz
2 Perception 24.10.2019 – Johannes Betz	**Practice 2** 24.10.2019 – Johannes Betz
3 Supervised Learning: Regression 31.10.2019 – Alexander Wischnewski	**Practice 3** 31.10.2019 – Alexander Wischnewski
4 Supervised Learning: Classification 7.11.2019 – Jan Cedric Mertens	**Practice 4** 7.11.2019 – Jan Cedric Mertens
5 Unsupervised Learning: Clustering 14.11.2019 – Jan Cedric Mertens	**Practice 5** 14.11.2019 – Jan Cedric Mertens
6 Pathfinding: From British Museum to A* 21.11.2019 – Lennart Adenaw	**Practice 6** 21.11.2019 – Lennart Adenaw
7 Introduction: Artificial Neural Networks 28.11.2019 – Lennart Adenaw	**Practice 7** 28.11.2019 – Lennart Adenaw
8 Deep Neural Networks 5.12.2019 – Jean-Michael Georg	**Practice 8** 5.12.2019 – Jean-Michael Georg
9 Convolutional Neural Networks 12.12.2019 – Jean-Michael Georg	**Practice 9** 12.12.2019 – Jean-Michael Georg
10 Recurrent Neural Networks 19.12.2019 – Christian Dengler	**Practice 10** 19.12.2019 – Christian Dengler
11 Reinforcement Learning 09.01.2020 – Christian Dengler	**Practice 11** 09.01.2020 – Christian Dengler
12 AI-Development 16.01.2020 – Johannes Betz	**Practice 12** 16.01.2020 – Johannes Betz
13 Guest Lecture: VW Data:Lab 23.01.2020 –	

Objectives for Lecture 8: Neural Networks

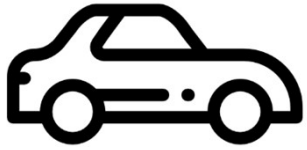


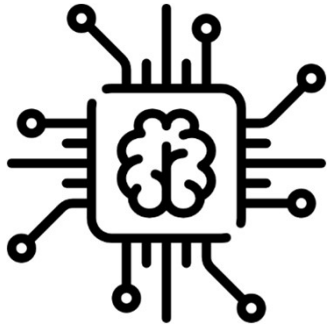
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

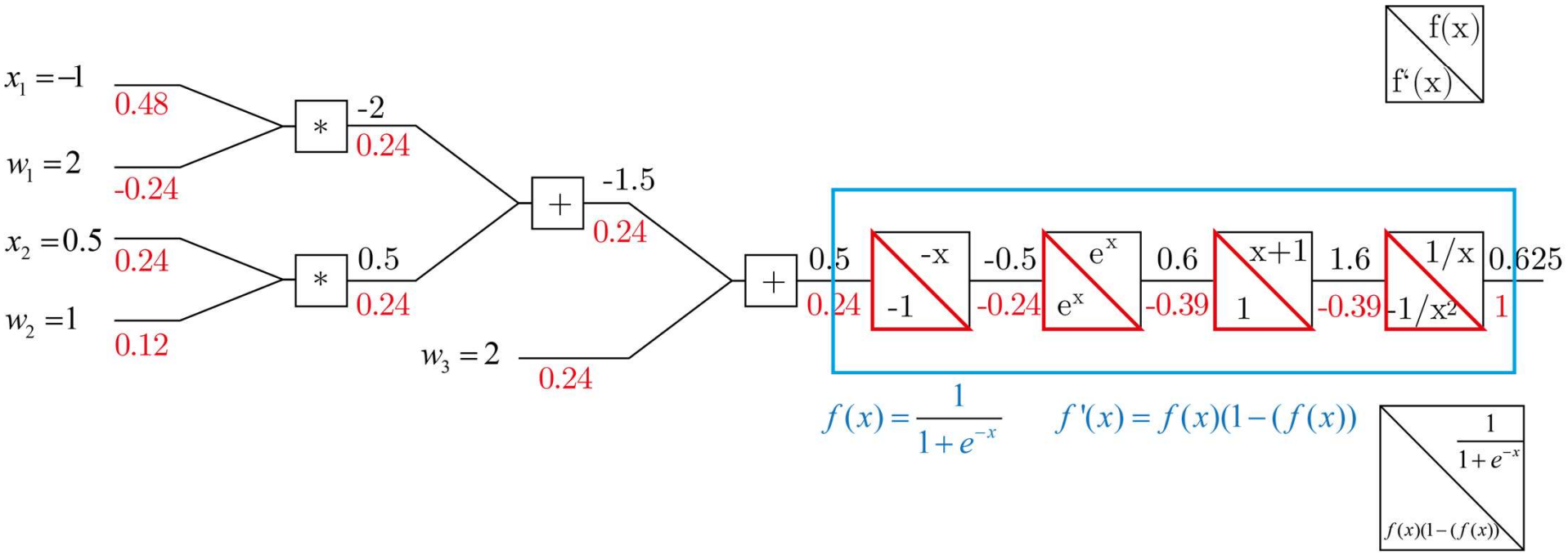
1. Chapter: Convolutional Neural Networks
    - 1.1 Motivation
    - 1.2 Introduction
    - 1.3 Dimensions, Padding, Stride
2. Chapter: Pooling
3. Chapter: CNN in Action
4. Chapter: Advanced Topics
    - 2.1 Optimizer
    - 2.2 Data Formatting



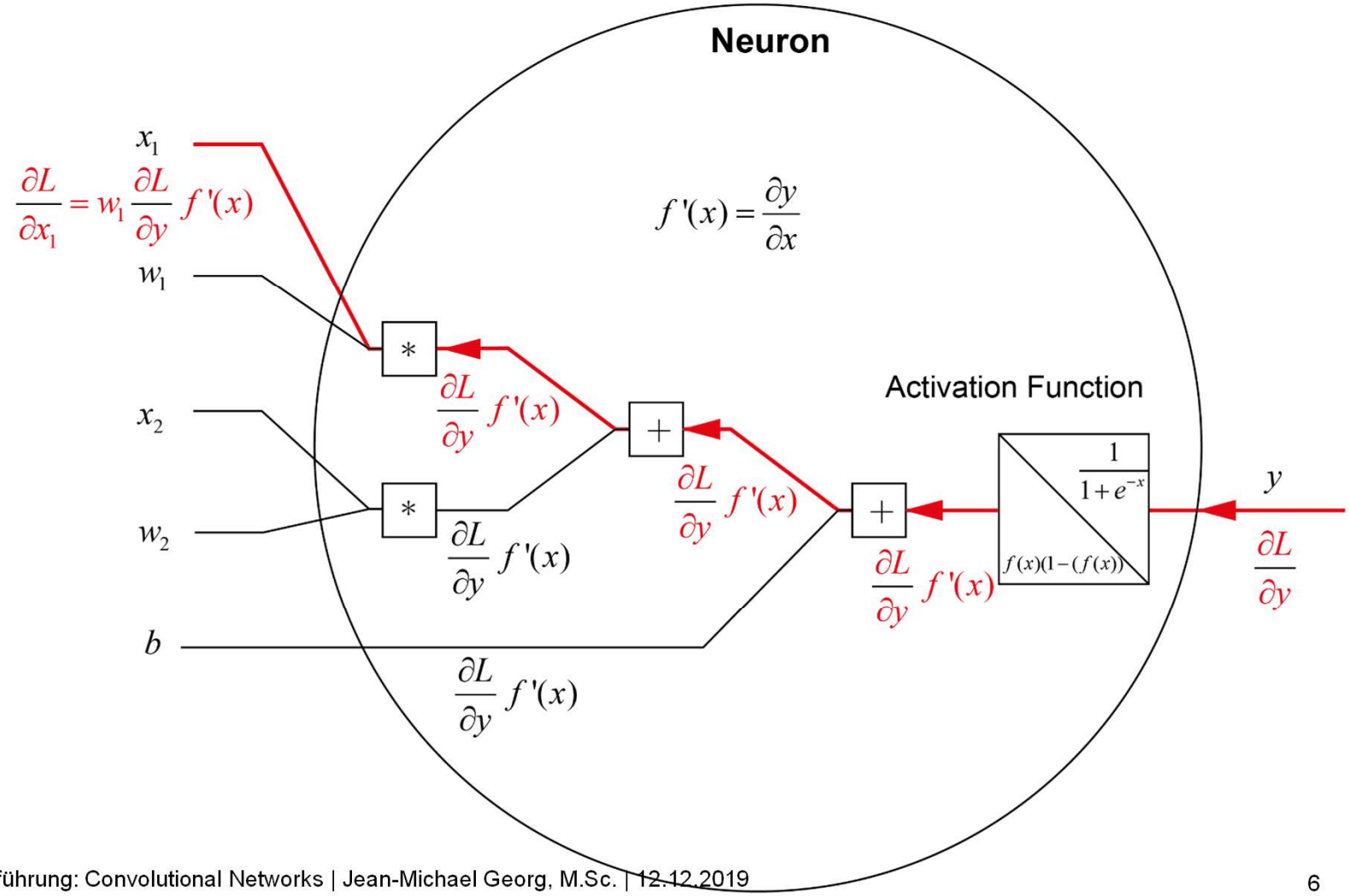


Computational Graph – Complex Example

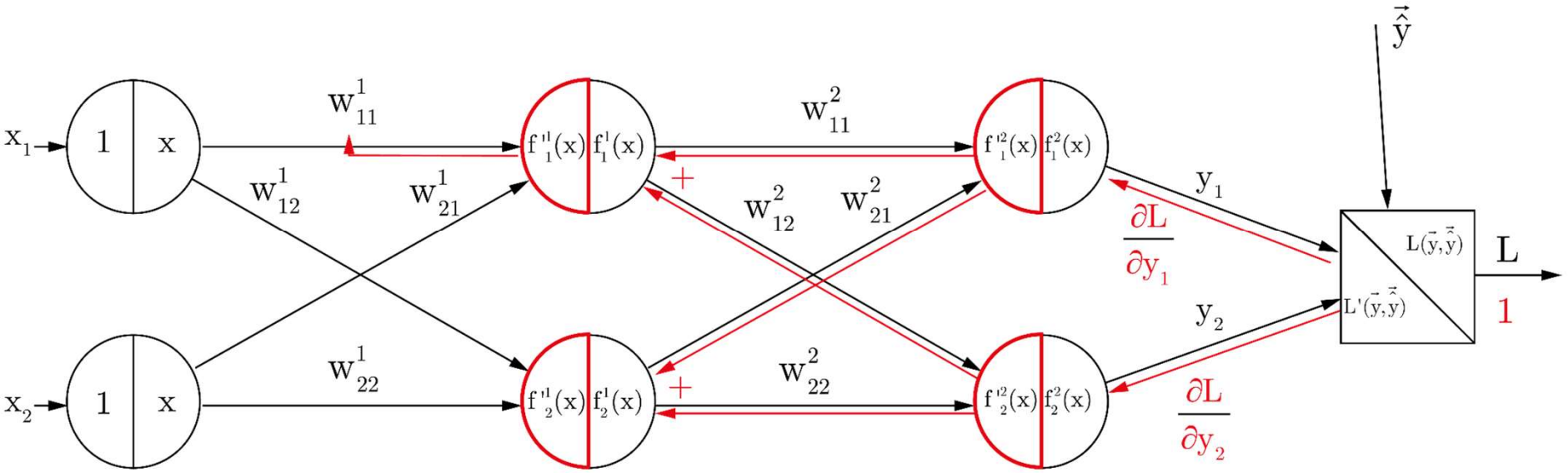
$$f(x_1, x_2) = \frac{1}{1 + e^{-(x_1 w_1 + x_2 w_2 + w_3)}}$$



Computational Graph – Neuron

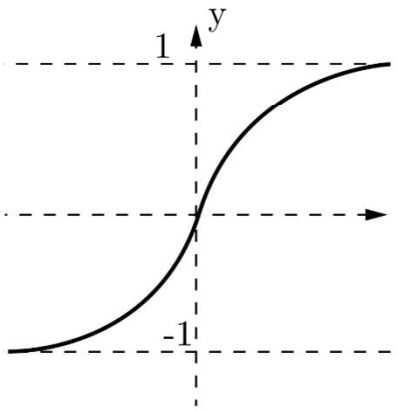


Backpropagation– Neural Network 2 Hidden Layers



Activation Functions

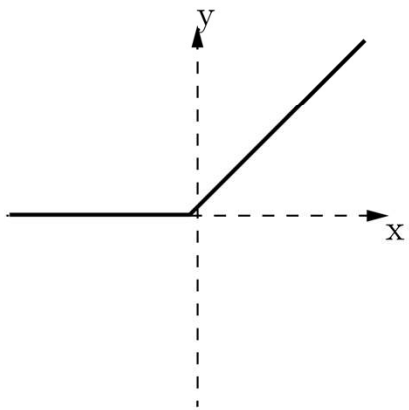
tanh



$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$f'(x) = \frac{4}{(e^{x} + e^{-x})^{2}}$$

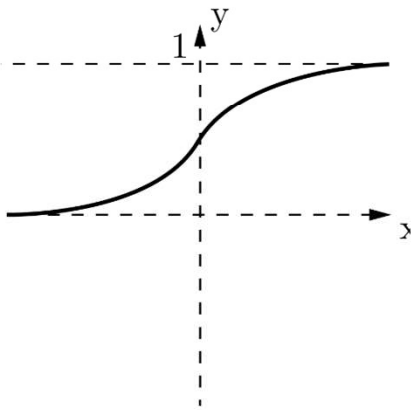
ReLu



$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

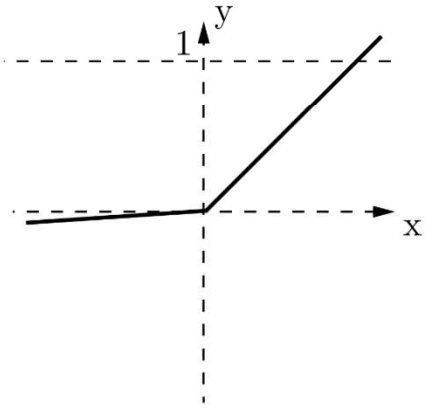
Sigmoid



$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

leaky/parametric Relu



$$f(x) = \begin{cases} x, & x \geq 0 \\ ax, & x < 0 \end{cases}$$

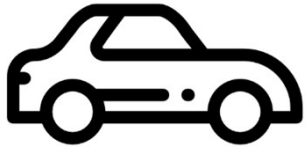
$$f'(x) = \begin{cases} 1, & x \geq 0 \\ a, & x < 0 \end{cases}$$

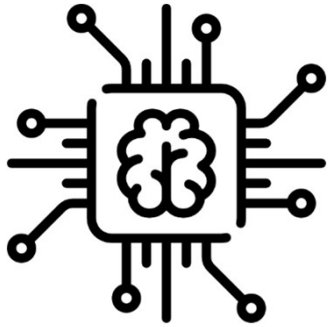
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

1. Chapter: Convolutional Neural Networks
   - 1.1 Motivation
   - 1.2 Introduction
   - 1.3 Dimensions, Padding, Stride
2. Chapter: Pooling
3. Chapter: CNN in Action
4. Chapter: Advanced Topics
   - 2.1 Optimizer
   - 2.2 Data Formatting





Video in our everyday life





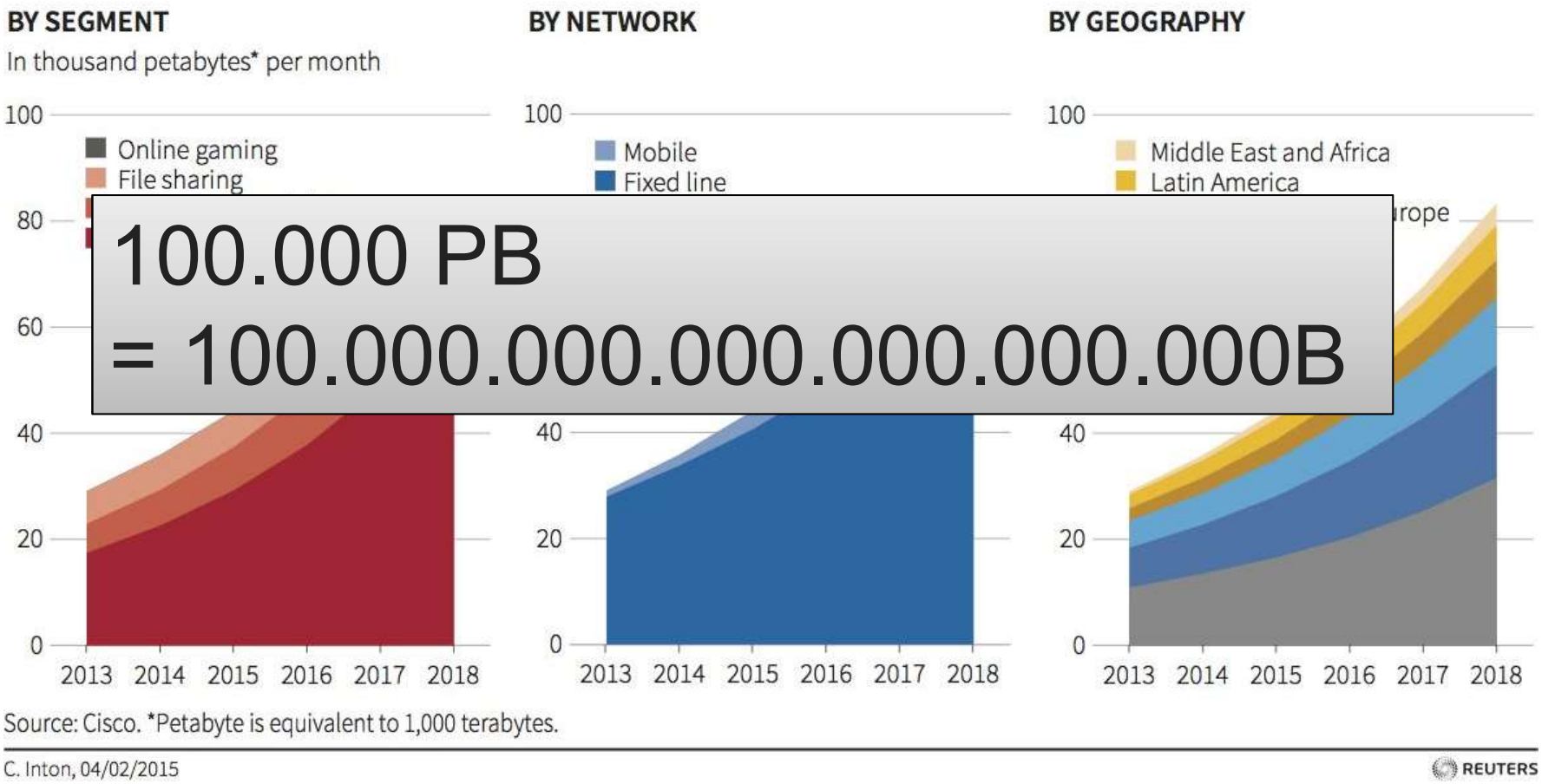





Einführung: Convolutional Networks | Jean-Michael Georg, M.Sc. | 12.12.2019

Consumer internet traffic

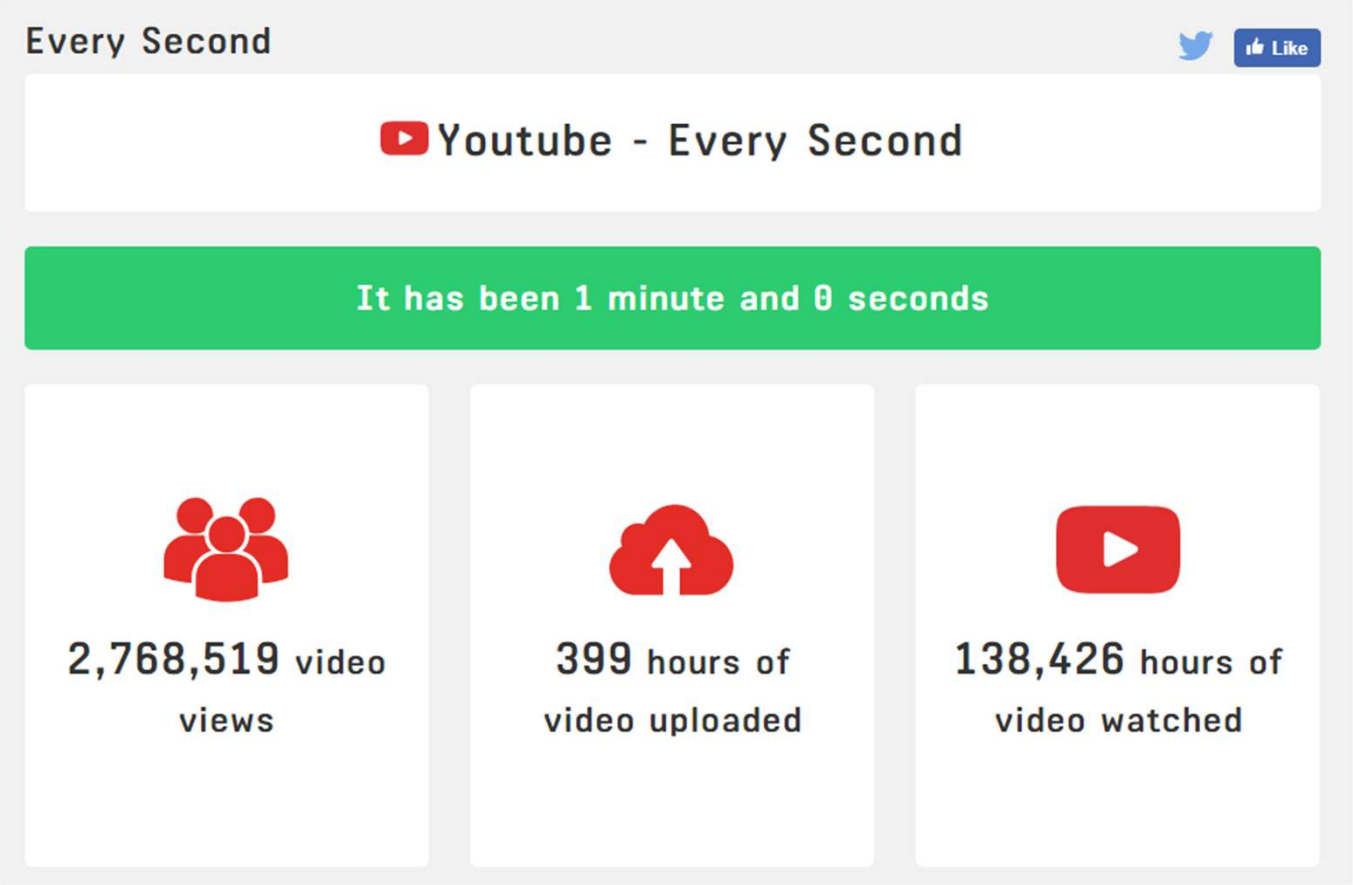
Internet video traffic will rise from 60 to 75 percent of total consumer internet traffic by 2018, according to estimates by Cisco.



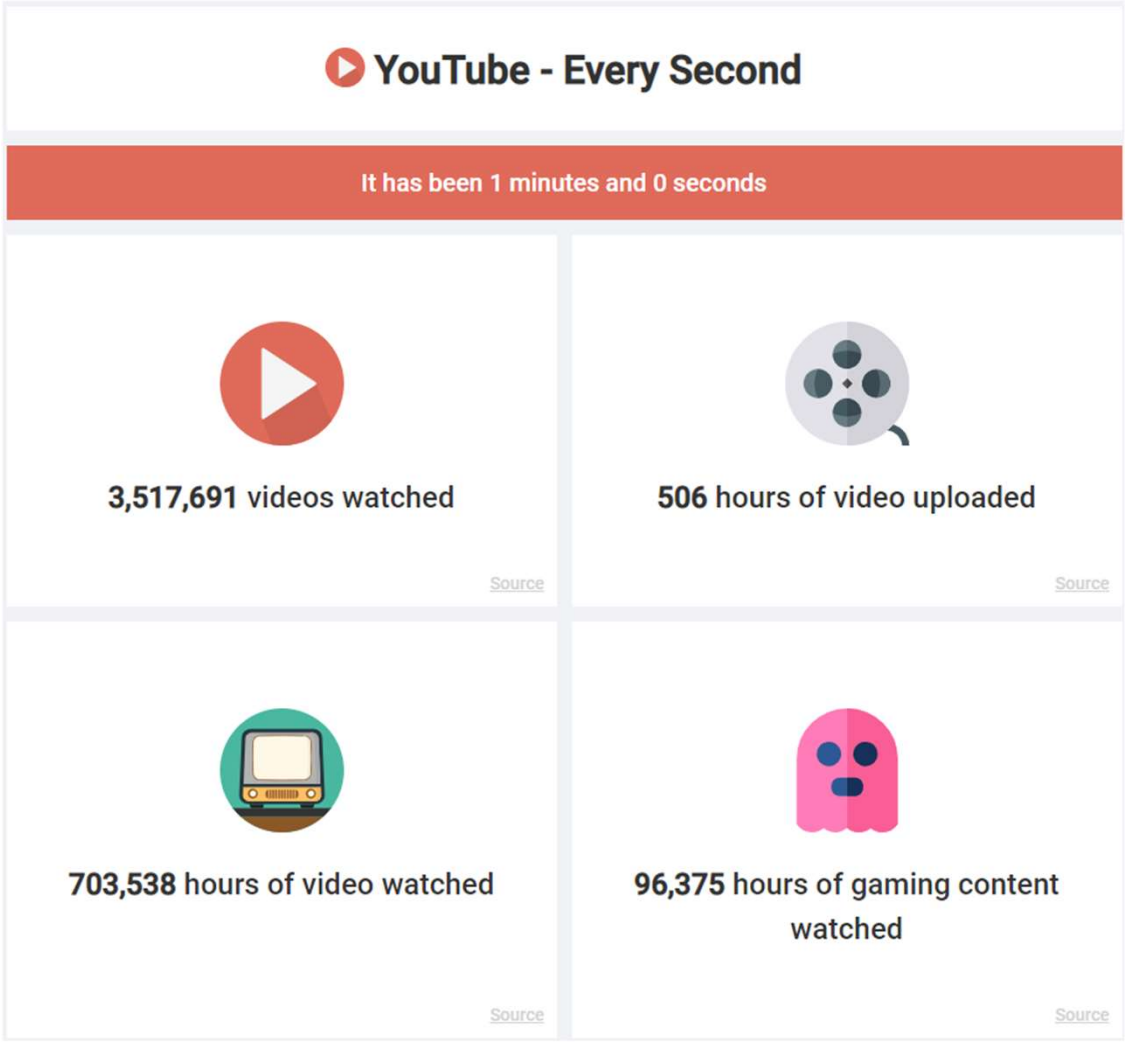
100.000 PB
= 100.000.000.000.000.000.000B

Youtube Every Second - 2018

- http://www.everysecond.io/youtube



Youtube Every Second - 2019



Cameras in autonomous cars

Additional Slides

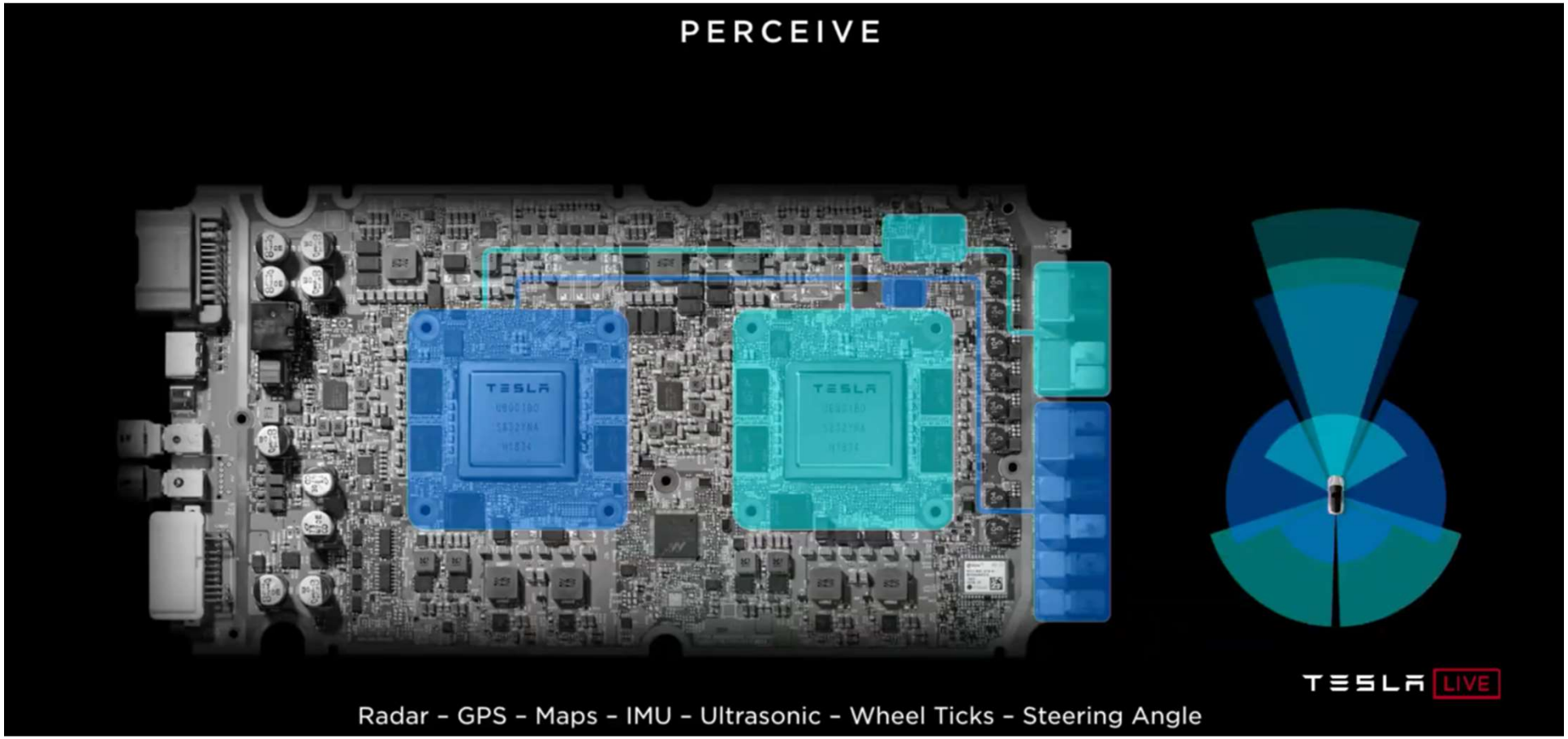
As video streams become increasingly part in our daily life, understandig the content of the camera data becomes an import task for researchers and companies. This is shown by the total worldwide internet traffic, where the video-stream traffic has the largest share, and the amount of cameras which are used in autonmous vehicles. But analyzing this data has many challenges…

Tesla Model 3

https://www.techrepublic.com/article/teslas-autopilot-cheat-sheet/

Tesla Autonomy Day

22.04.2019

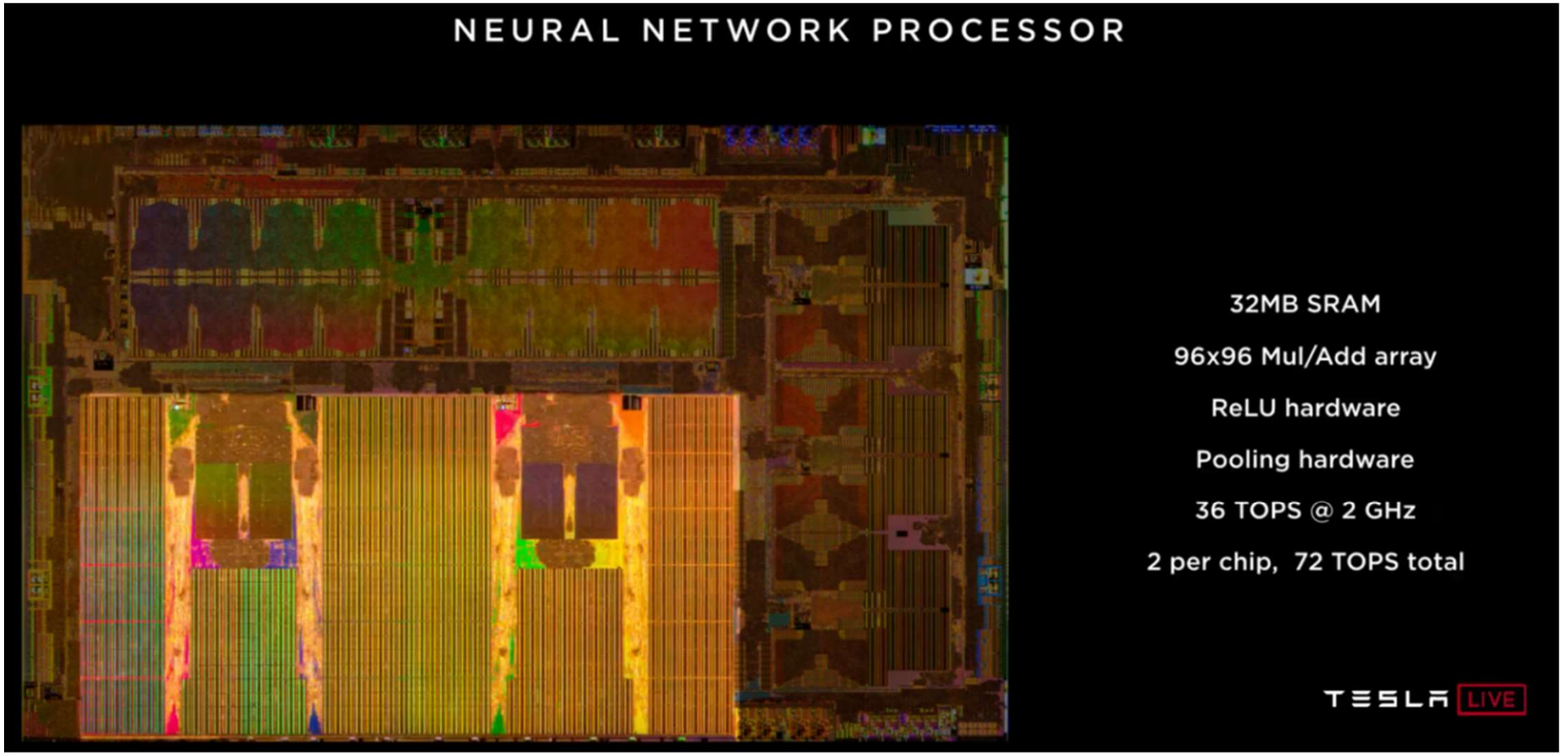


„Lidar is doomed“ – Elon Musk, „Tesla der Konkurrenz mind. 4 Jahre voraus“ – ARK Invest analyst James Wang

https://theteslashow.com/tesla-autonomy-investor-day/

Tesla Autonomy Day

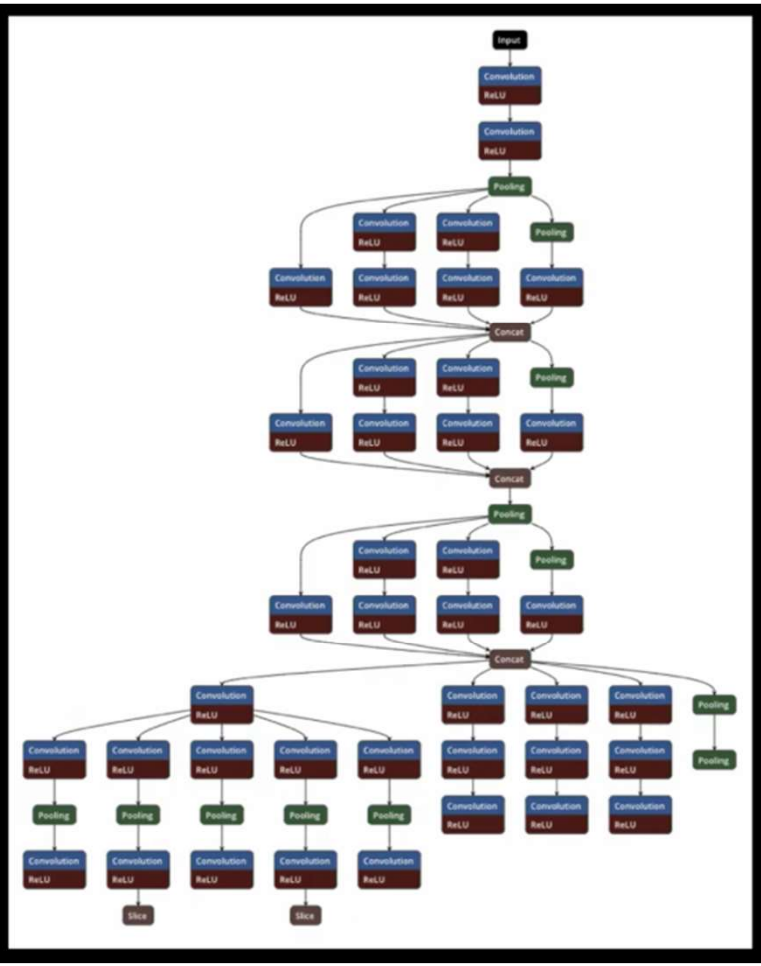
NN Chip



https://theteslashow.com/tesla-autonomy-investor-day/

Tesla Autonomy Day

NN Architektur



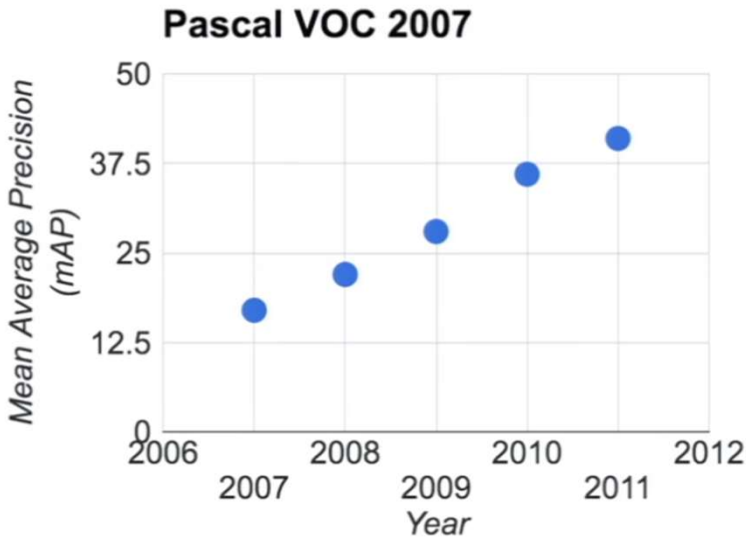
Operation	MOPS	%
Convolution	34275	98.1
Deconvolution	576	1.6
ReLU	123	0.1
Pooling	13	0.2

99.7% of operations are multiply add

TESLA LIVE

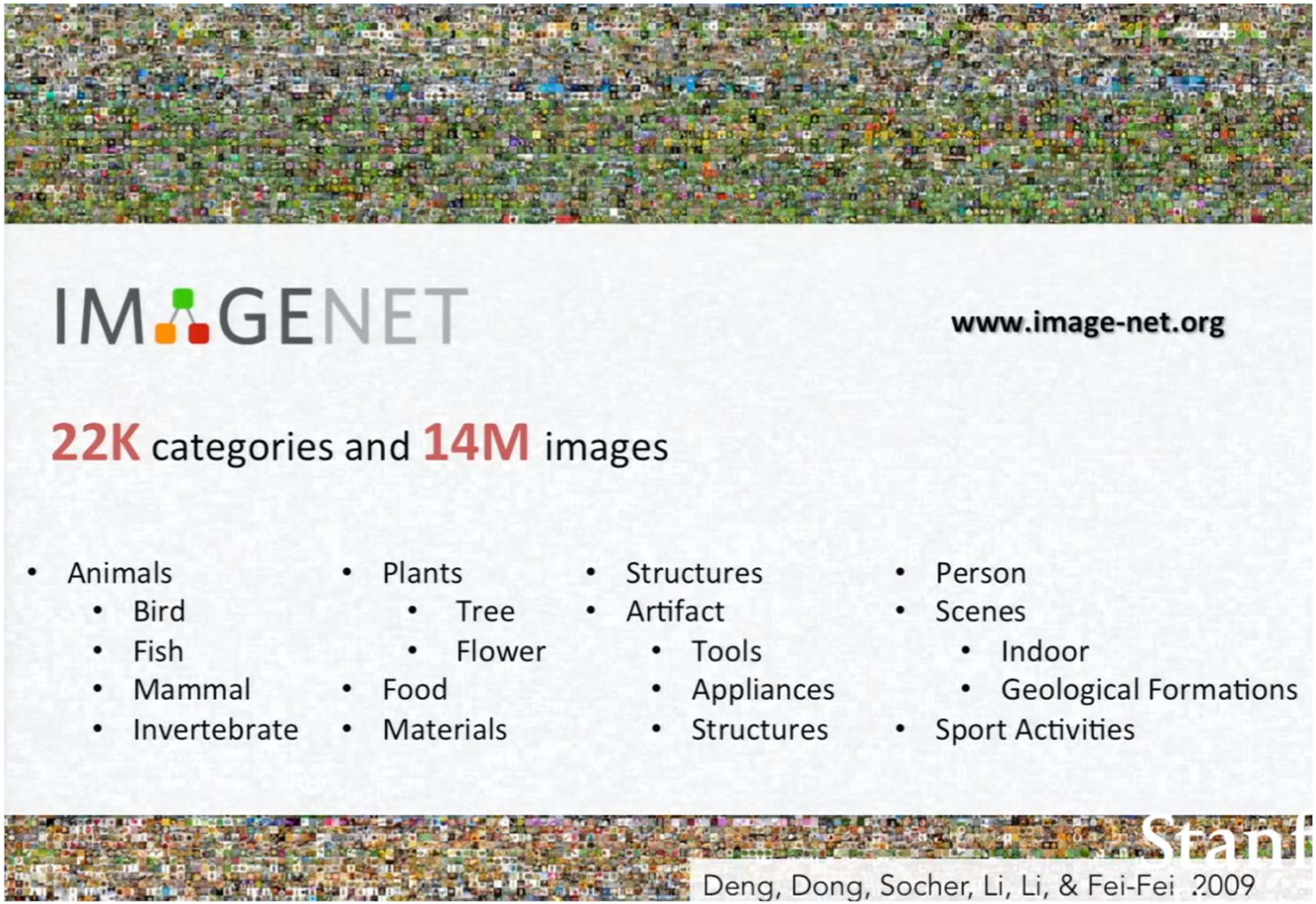
https://theteslashow.com/tesla-autonomy-investor-day/

PASCAL Visual Object Challenge

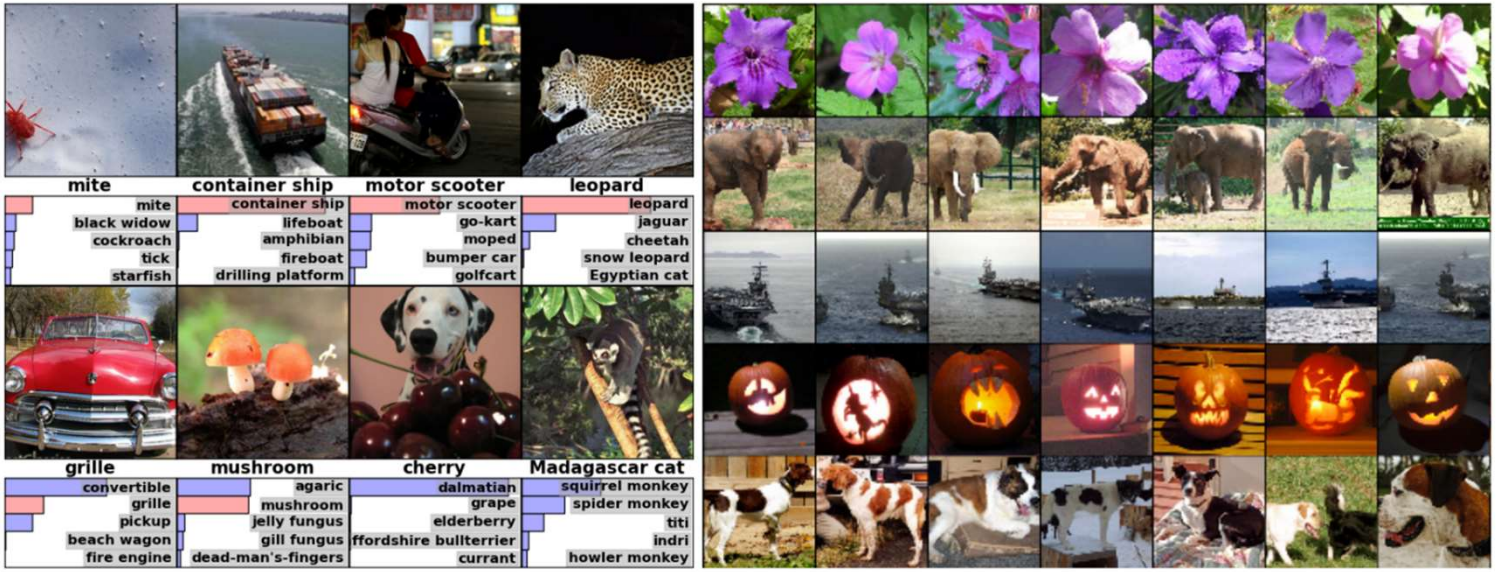


[Everingham et al. 2006-2012]

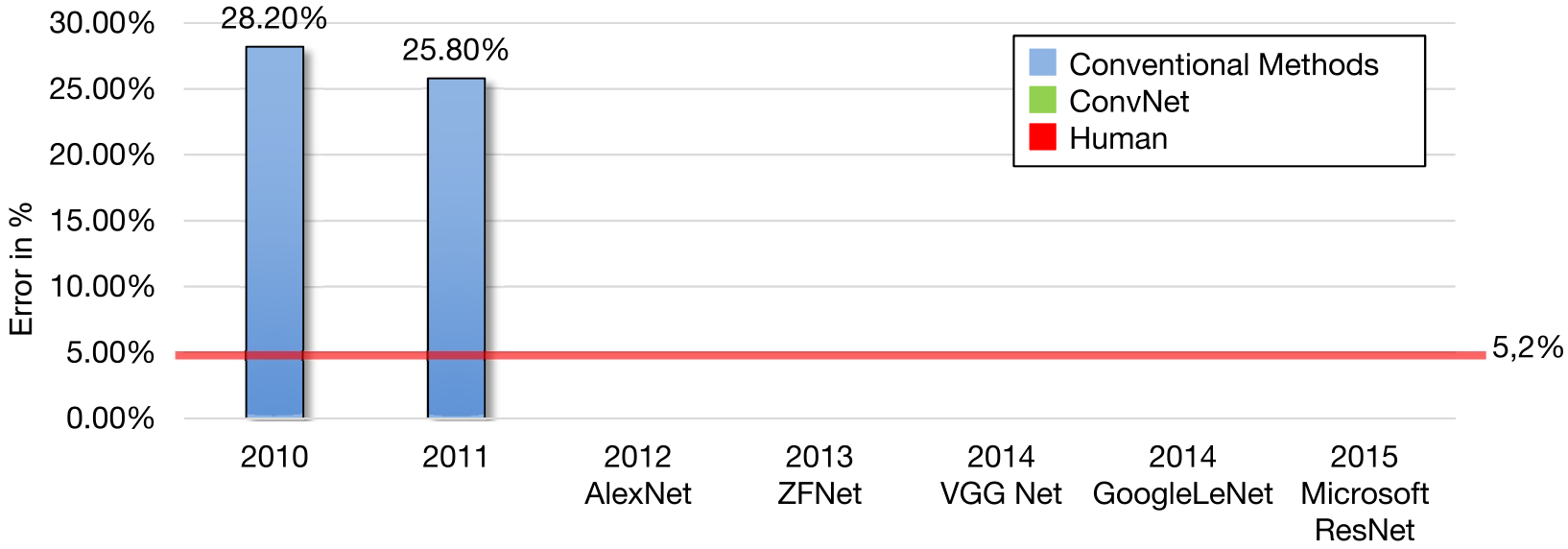
Imagenet


IMAGENET
www.image-net.org
22K categories and 14M images
• Animals
• Bird
• Fish
• Mammal
• Invertebrate
• Plants
• Tree
• Flower
• Food
• Materials
• Structures
• Artifact
• Tools
• Appliances
• Structures
• Person
• Scenes
• Indoor
• Geological Formations
• Sport Activities
Deng, Dong, Socher, Li, Li, & Fei-Fei 2009

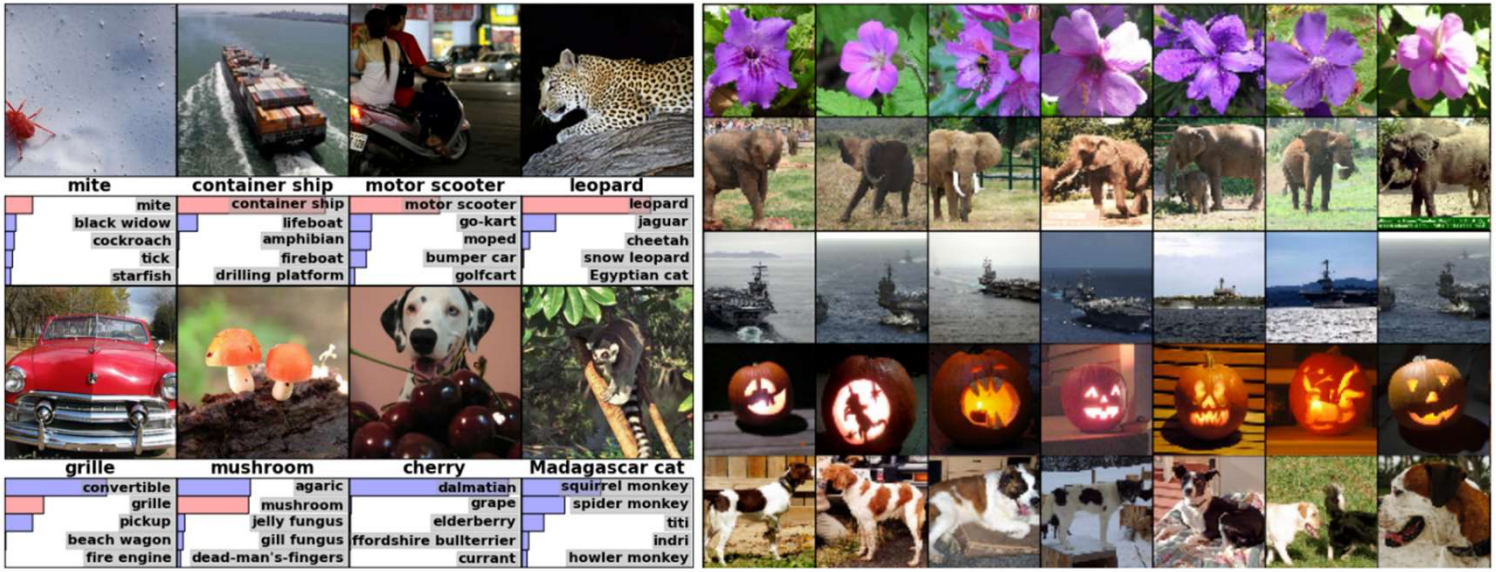
ImageNet Wettbewerb - Bildklassifizierung



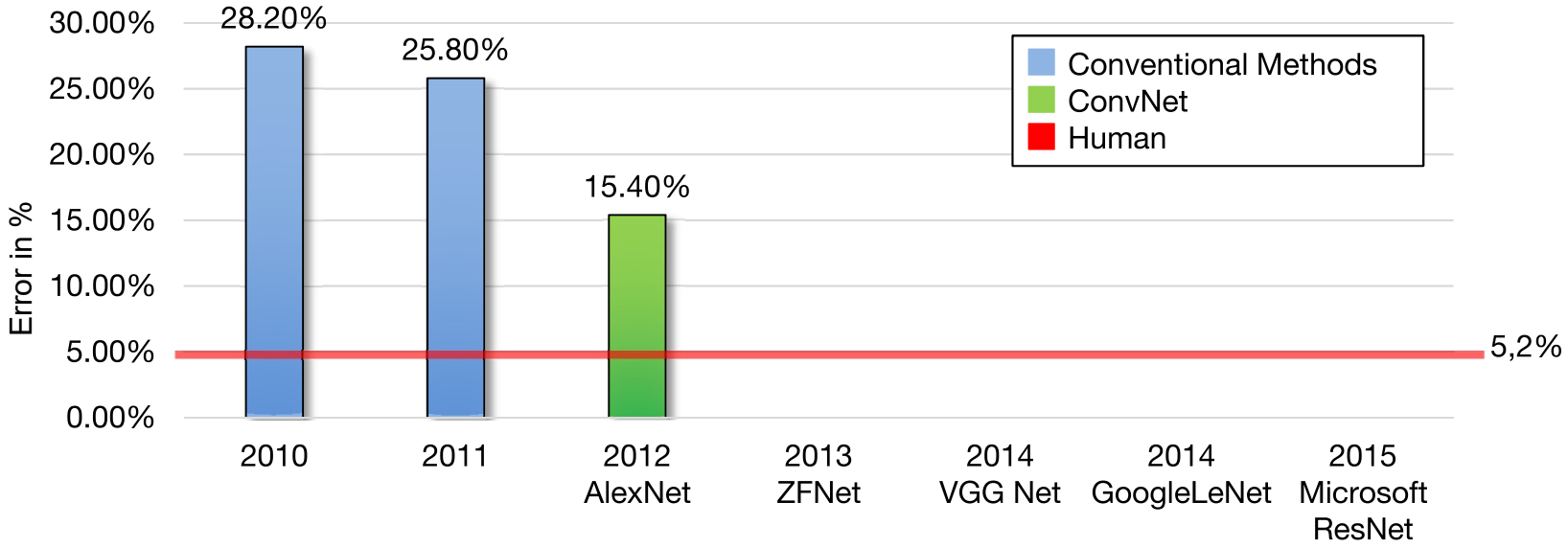
Results



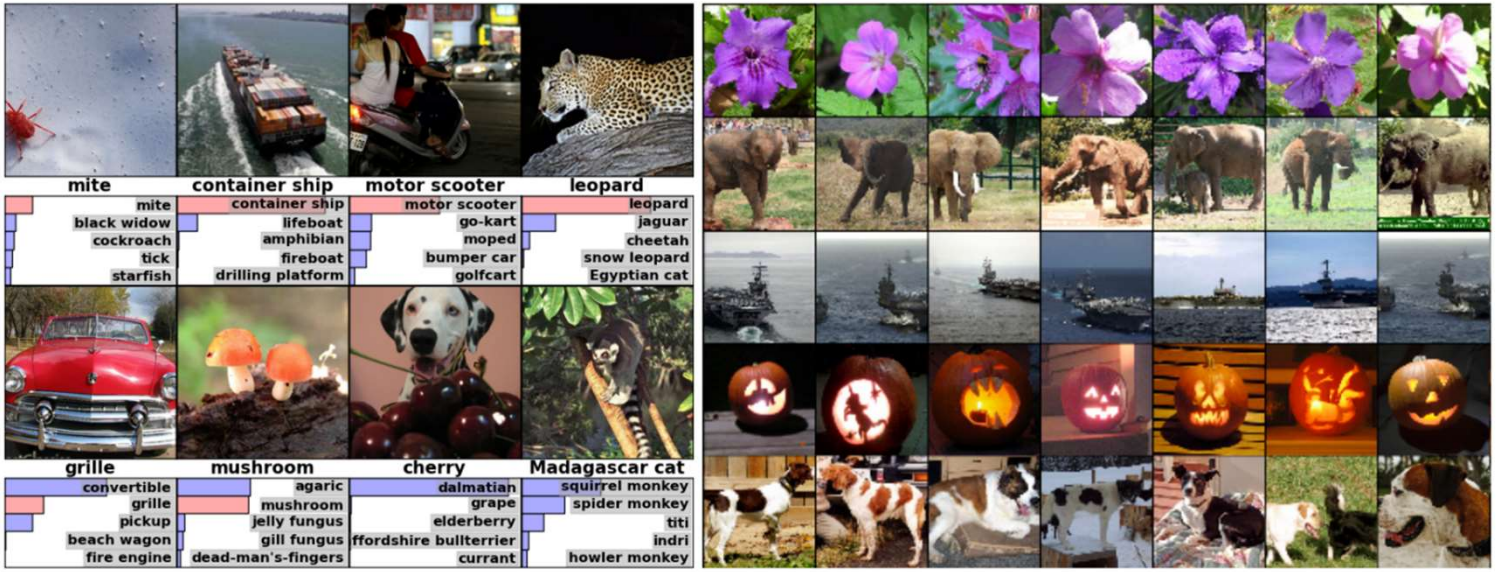
ImageNet Wettbewerb - Bildklassifizierung



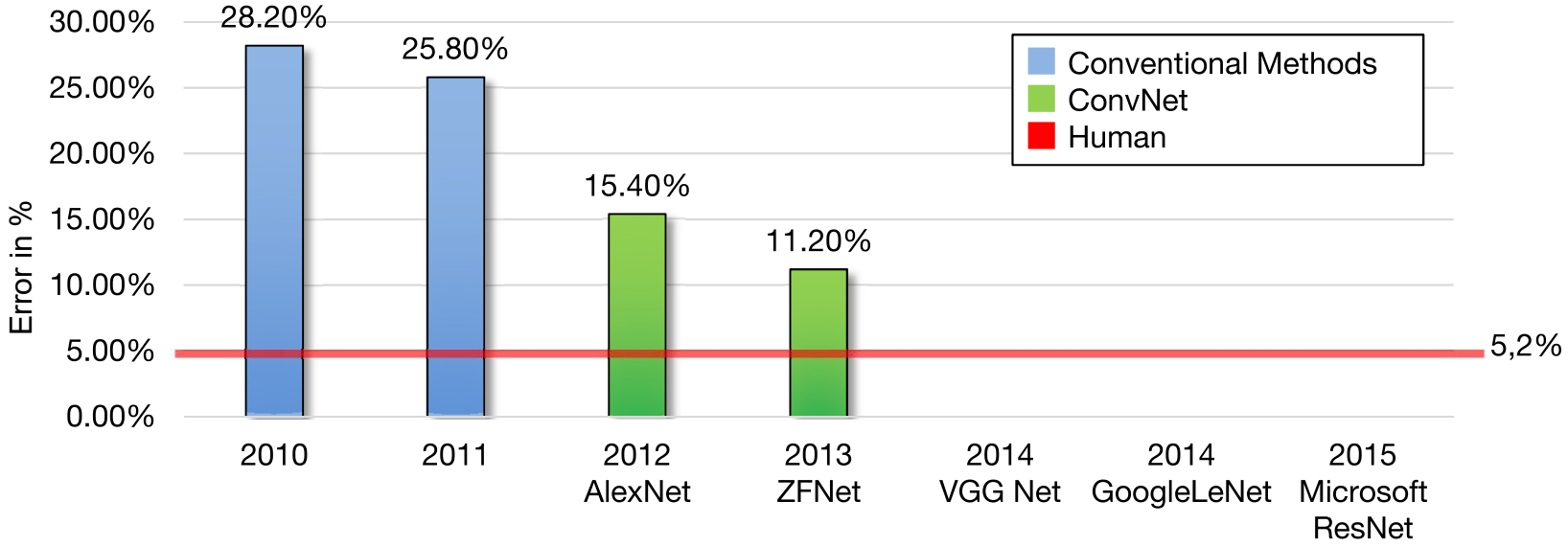
Results



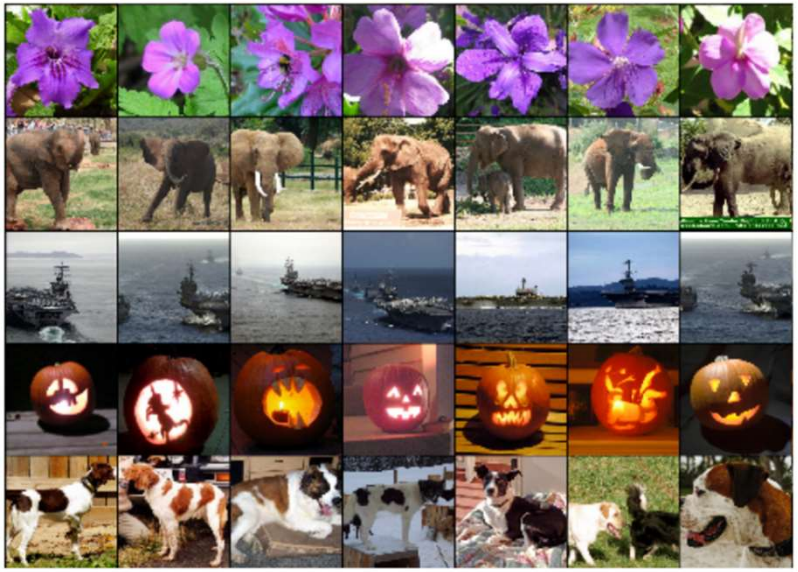
ImageNet Wettbewerb - Bildklassifizierung



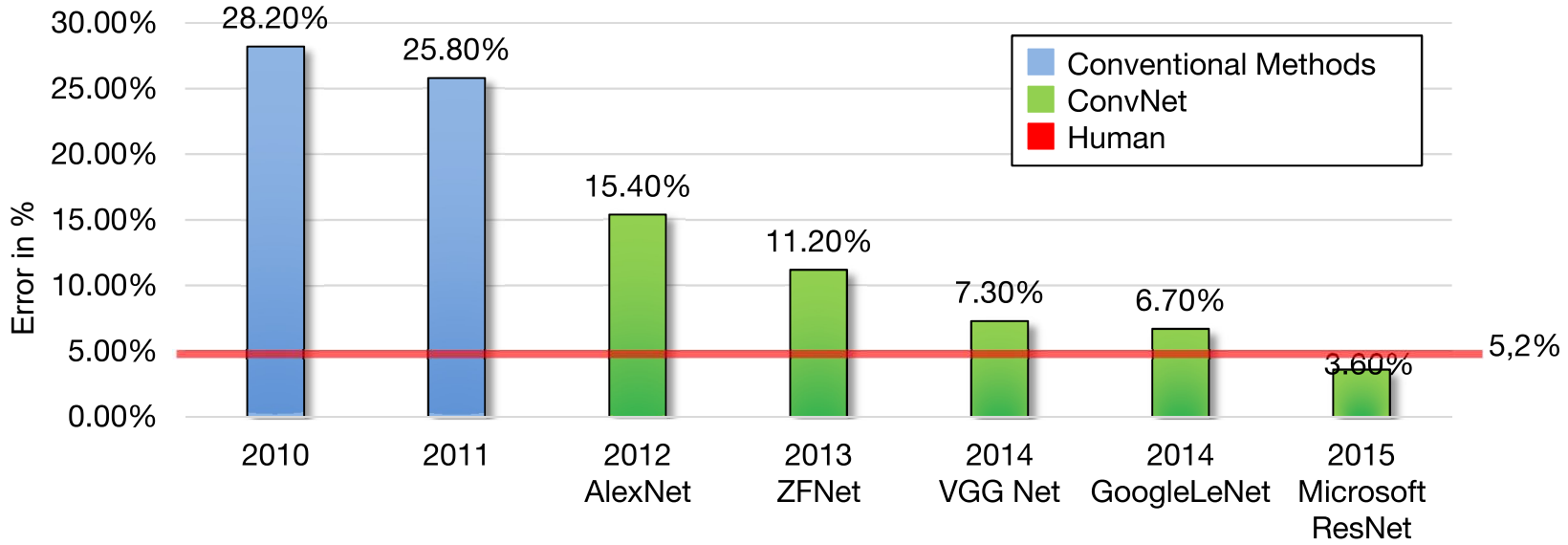
Results



ImageNet Wettbewerb - Bildklassifizierung



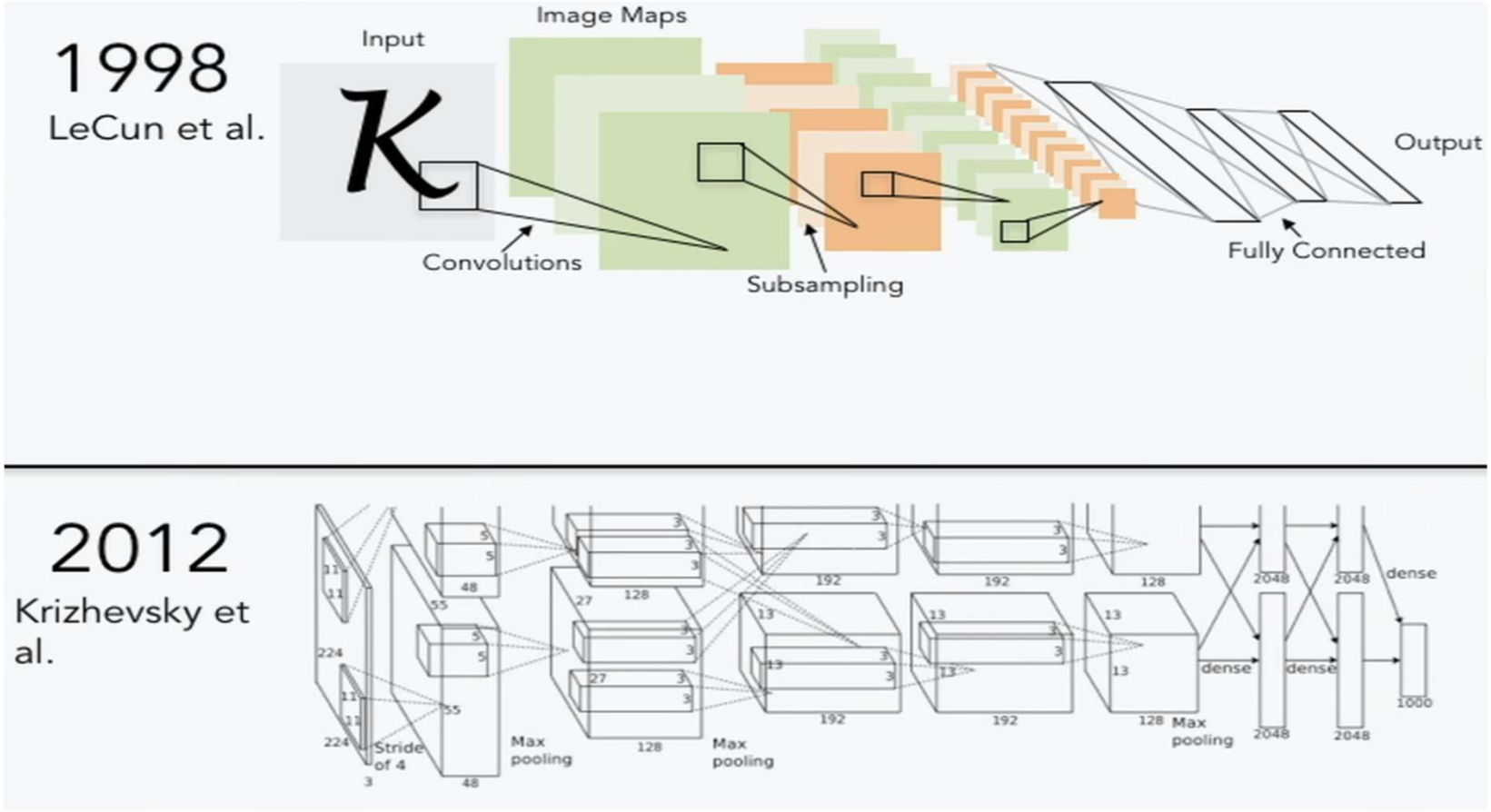
Results



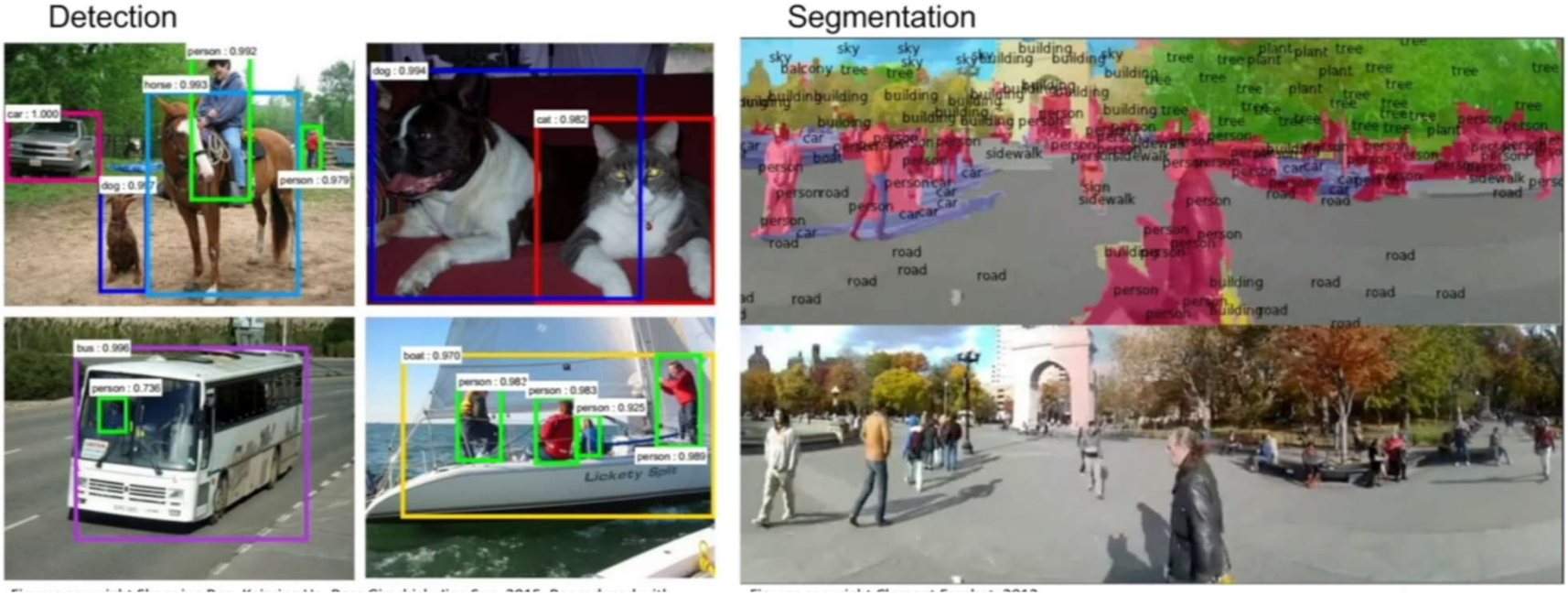
Additional Slides

The first breakthrough of nerual networks started with the imageNet competition in 2010. The challenge was to classify the objects shown in the images. The first conventional algorithms reached an error rate of around 28%. In 2012 the first neural network (AlexNet) managed to reduce the error rate by around 13%, essentialy halfing the best coventional method. The following years neural network architectures improved up to a point where the neural network outperformed the human. This amazing development is only possible because of **convolutional** neural networks.

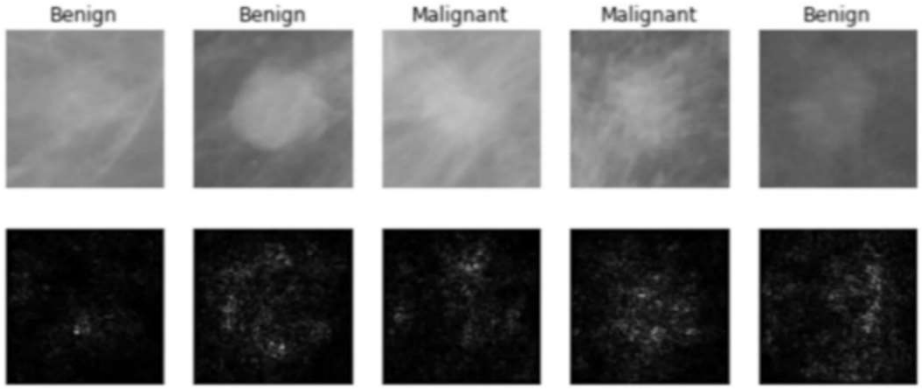
ConvNets exited before 2012…



What else can ConvNets to today ?



What else can ConvNets to today ?



[Levy et al. 2016]

[Dieleman et al. 2014]

[Sermanet et al. 2011]
[Ciresan et al.]

Most important: Convnets can make you rich!

SPIEGEL ONLINE SPIEGEL+ Anmelden

Menü | Politik Meinung Wirtschaft Panorama Sport Kultur Netzwelt Wissenschaft mehr ▾

NETZWELT

Schlagzeilen | DAX 11.066,41 | TV-Programm | Abo

Nachrichten > Netzwelt > Web > Christie's > Künstliche Intelligenz: Christie's erzielt mit KI-Gemälde 432.500 Dollar

KI-Kunstwerk versteigert

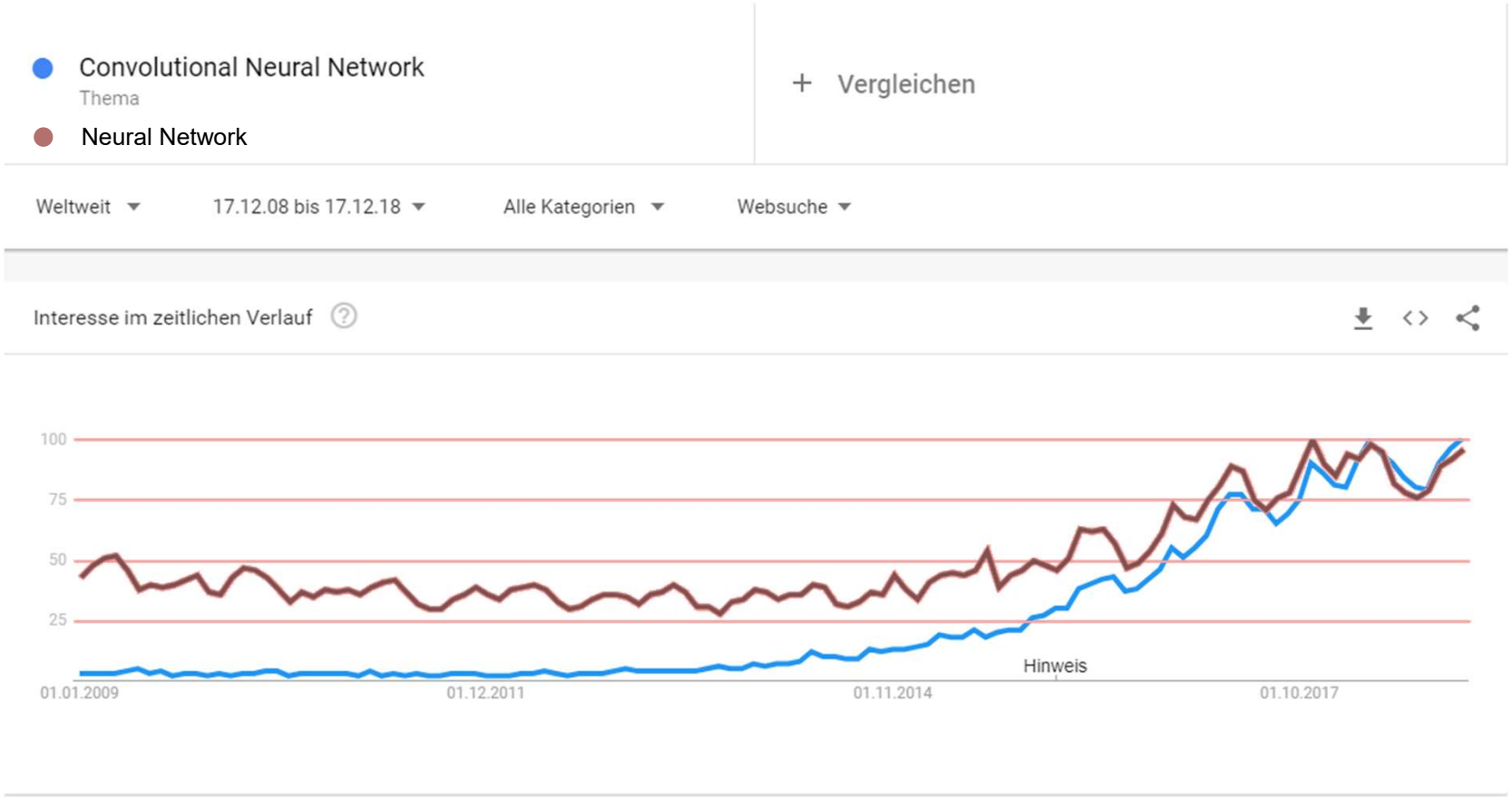
Gemälde von "min G max D Ex[log(D(x))]+Ez[log(1-D(G(z)))]" erzielt 432.500 Dollar

Das Werk "Edmond de Belamy" wurde von einem künstlichen neuronalen Netzwerk geschaffen, nun hat das Auktionshaus Christie's es versteigert - und für das KI-Gemälde deutlich mehr bekommen als ursprünglich geschätzt.

26.10.2018

Convolutional Neural Networks

Google Trend

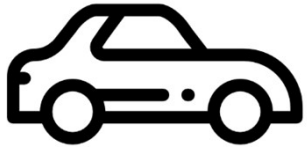


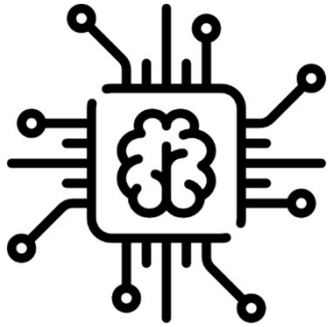
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

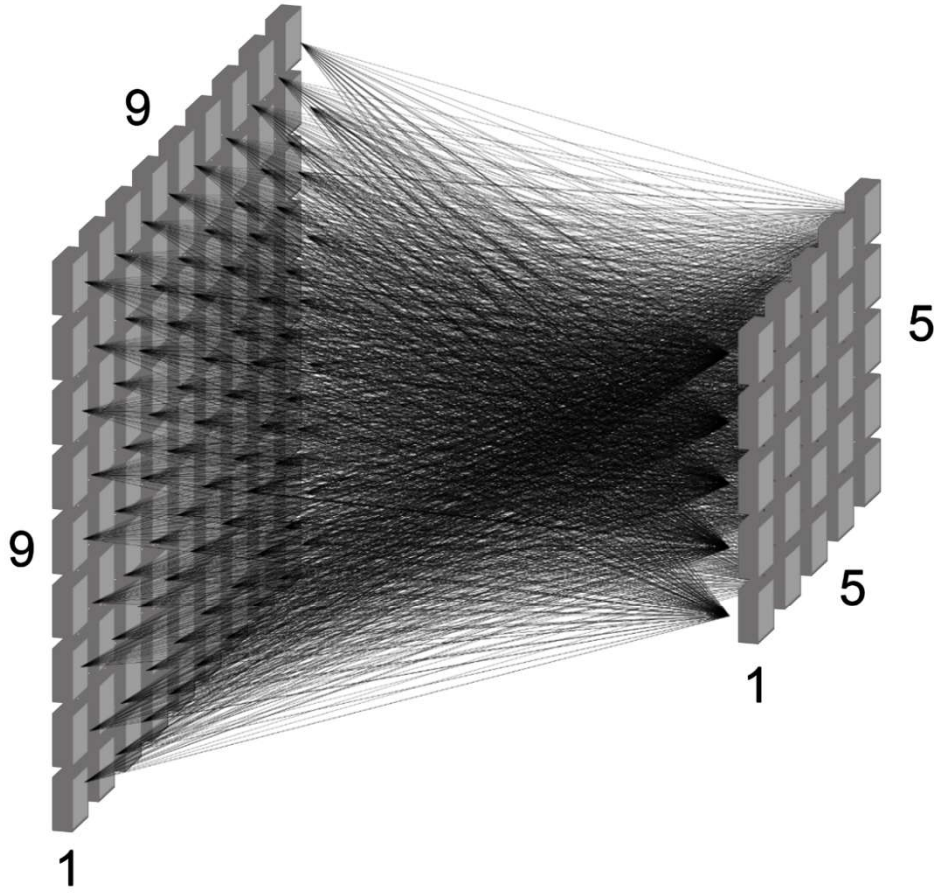
1. Chapter: Convolutional Neural Networks
    - 1.1 Motivation
    - 1.2 Introduction
    - 1.3 Dimensions, Padding, Stride
2. Chapter: Pooling
3. Chapter: CNN in Action
4. Chapter: Advanced Topics
    - 2.1 Optimizer
    - 2.2 Data Formatting



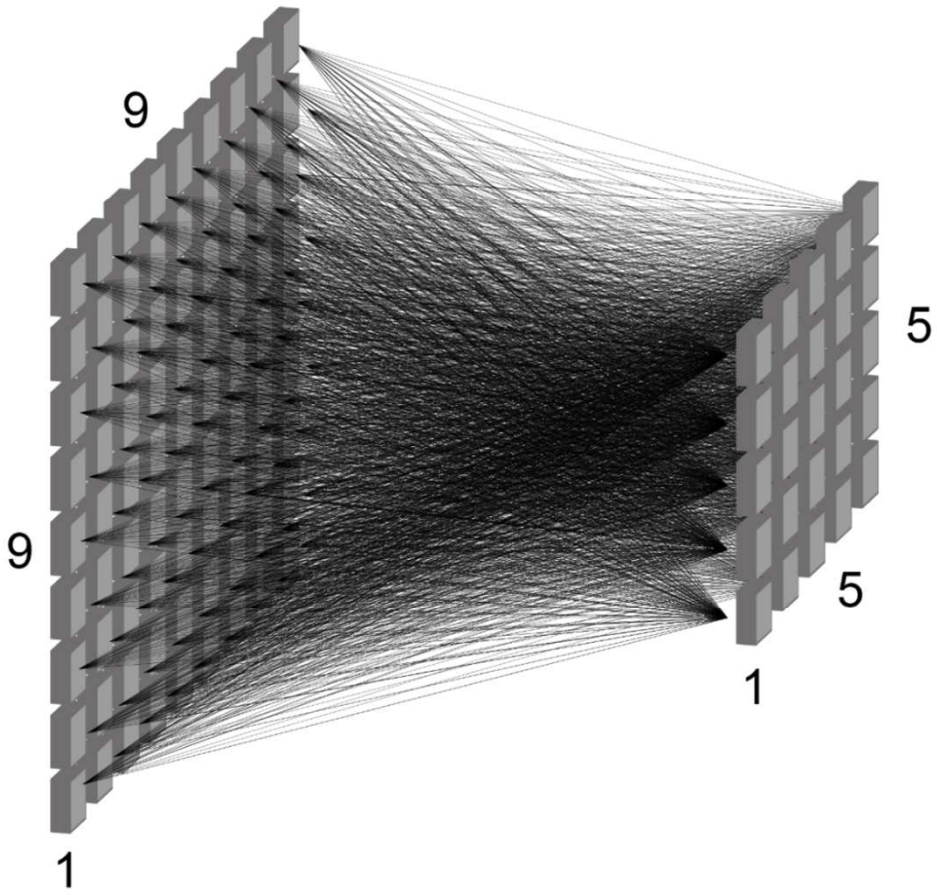


But…
…what are Convolutional Neural Networks?
...how do they work?
...and why do we need them?

Fully Connected Layer



Fully Connected Layer



$$\vec{x} \in K^{1\times 81}$$

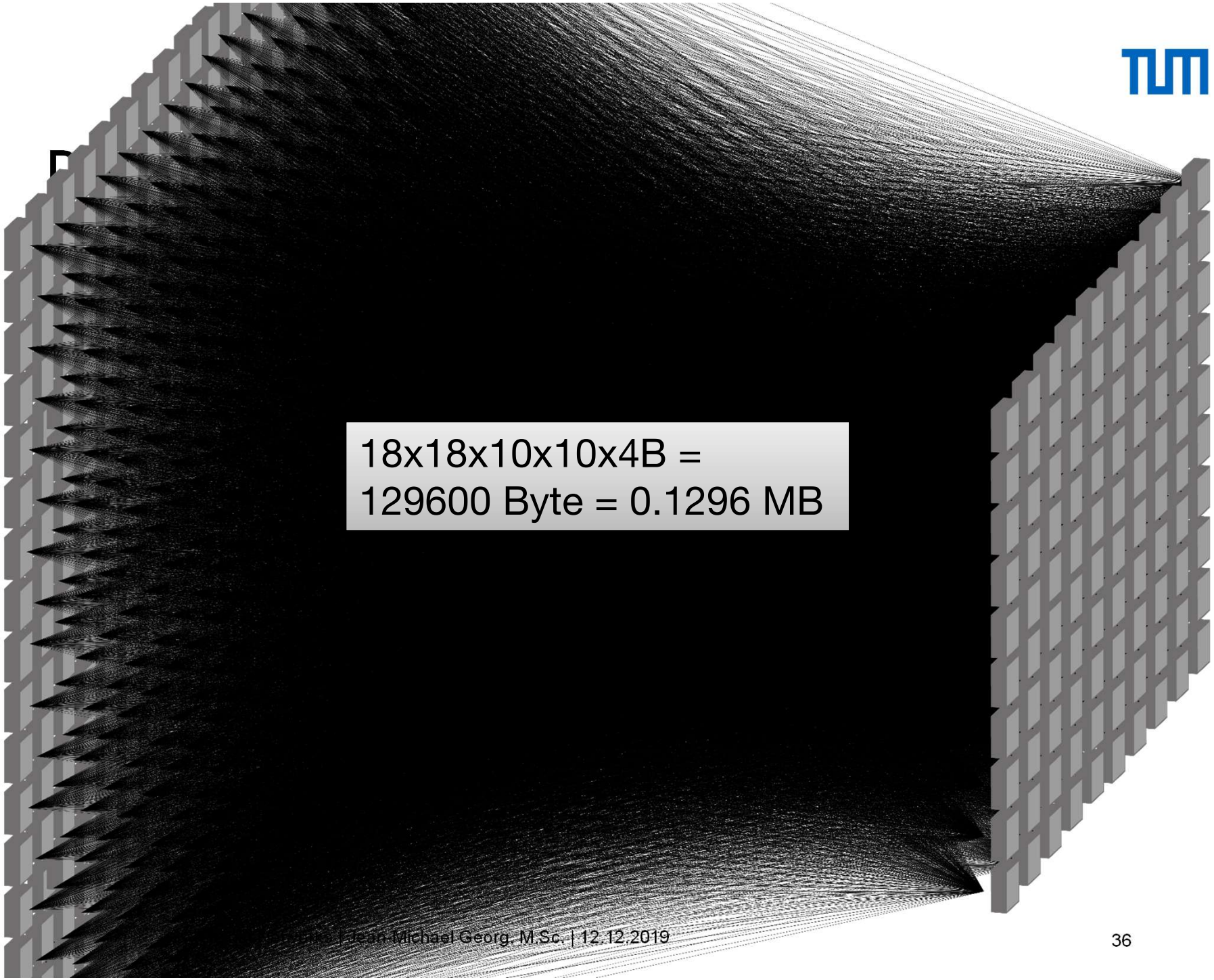
$$\vec{y} \in K^{1\times 25}$$

$$W \in K^{81x25}$$

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1,25} \\ \vdots & \ddots & \vdots \\ w_{81,1} & \cdots & w_{81,25} \end{pmatrix}$$

$$\Rightarrow 9 \cdot 9 \cdot 5 \cdot 5 = 2025 \text{ Parameter}$$

2025x4B =
8100 Byte = 8 kB


18x18x10x10x4B =
129600 Byte = 0.1296 MB

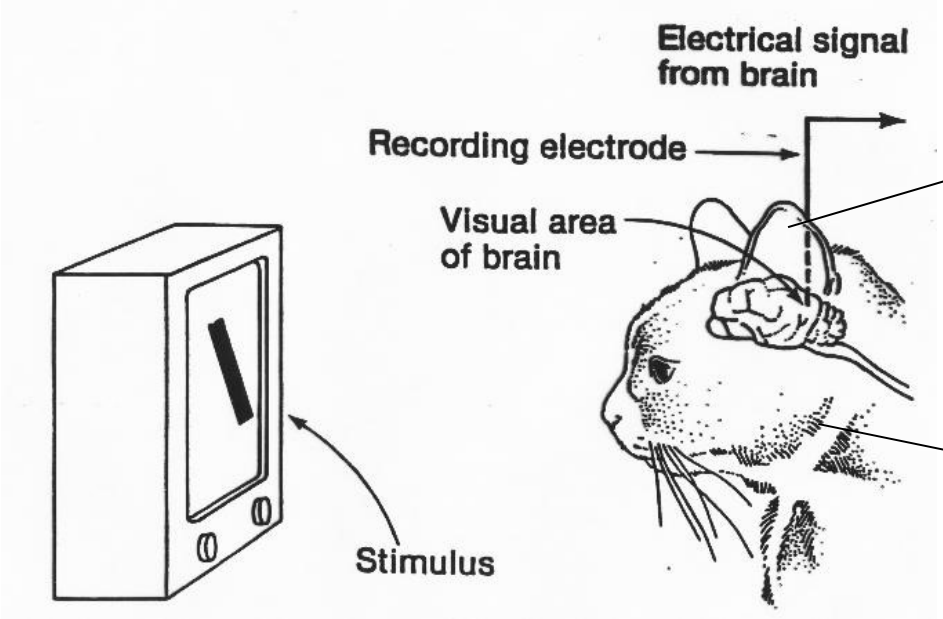
FullHD x FullHD?
1920x1080x3 x 1920x1080x3x 4B =
1.54729341e+14 B =
154.293 GB =
154 TB =
Internet Traffic of 12800 people in 2018
=> For images we need something else

Additional Slides

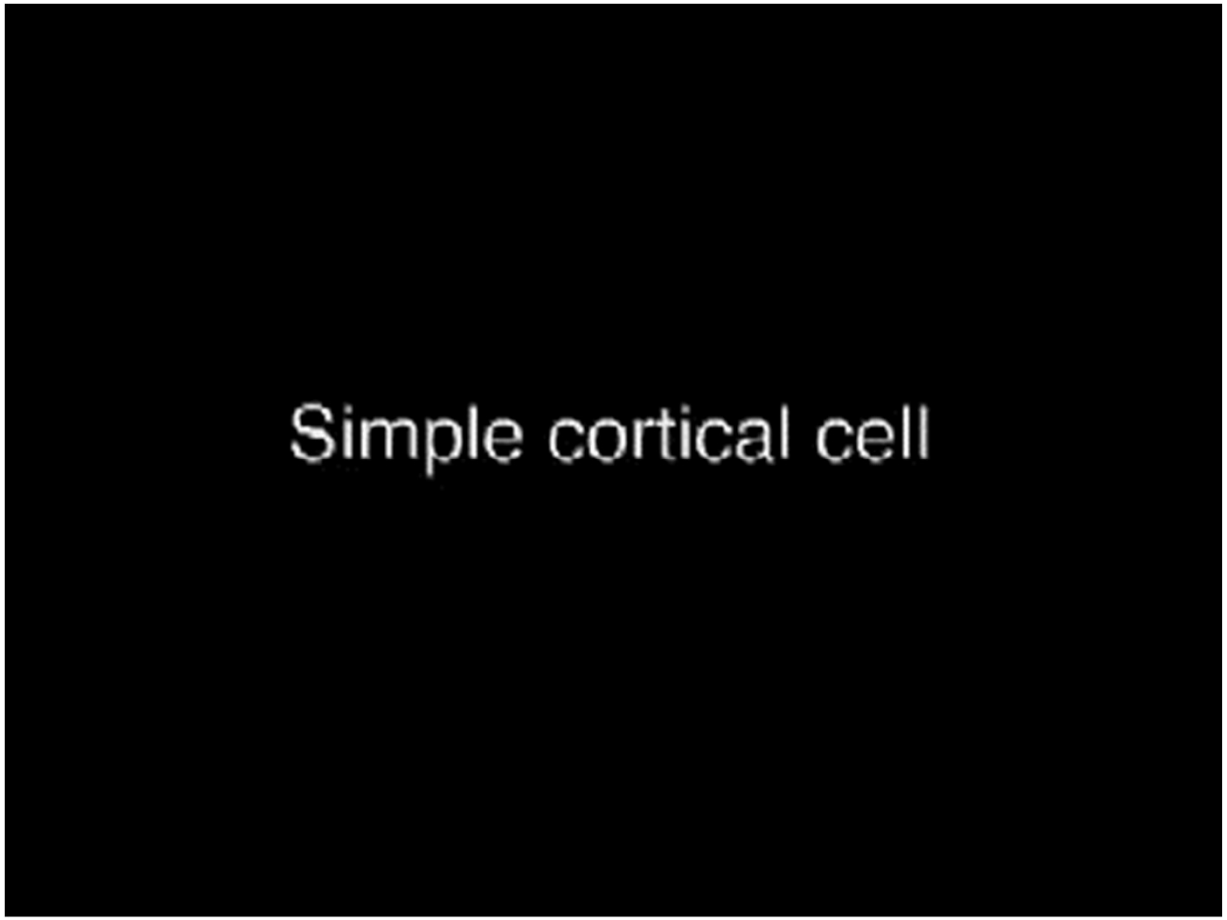
Analyzing images with a fully connected network is possible, but extremly memory expensive. The examples shown an input image with an 18x18 pixel input image and a 10x10 output image. Each square represents a neuron. In a fully connected layer all neurons from the imput layer are connected with all neurons from the output layer. Resulting in 18x18x10x10 = 32400 weights. Assuming every weight is a float, which uses 4Byte, we need 0.1298Mb to store the weights on our working memory.

But usually images are not that small. For analyzing an Full HD image we would need 154TB of working memory to store the weights of a single fully connected layer, which is not feasaile.

Hubel and Wiesel Experiment 1959



Hubel and Wiesel Experiment Video

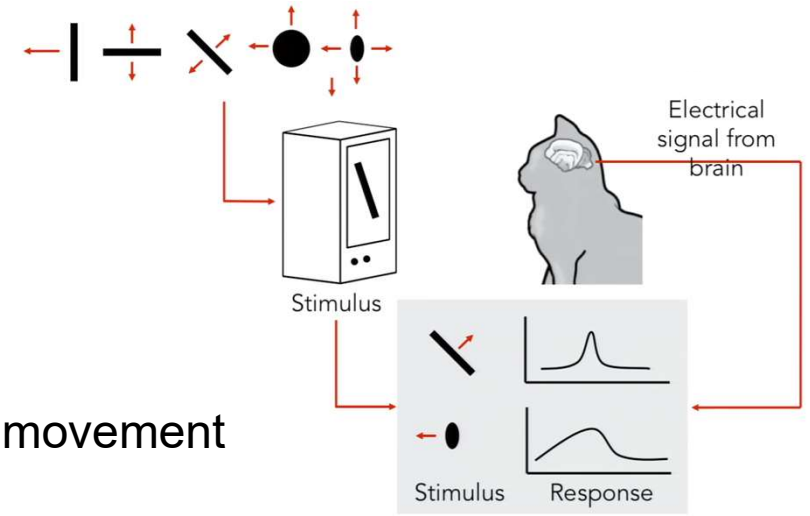


https://www.youtube.com

Hubel and Wiesel

Results

- **Simple cell:**
  - Responses to light orientation
- **Complex cell:**
  - Responses to light orientation and movement
- **Hypercomplex cells:**
  - Repsonse to movement with and end point



https://en.wikipedia.org

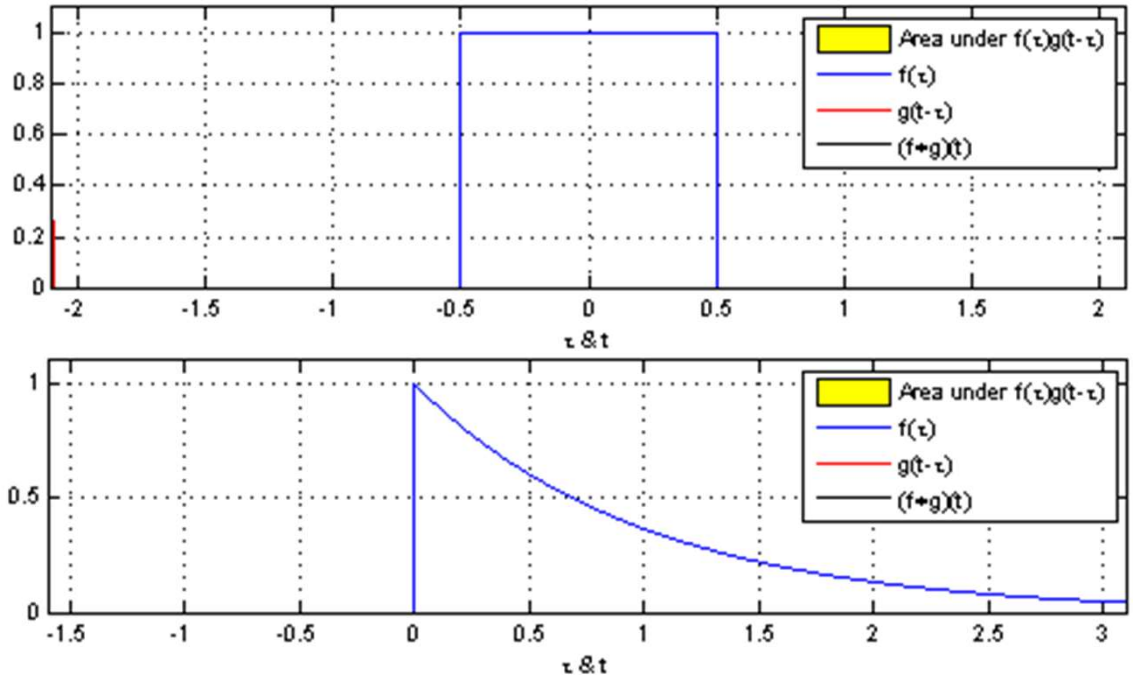
Additional Slides

Before we look at convolutional neural networks we have a look at an experiment Hubel and Wiesel did in 1959. In this experiment they connected a neurotransmitter to a single neuron inside the visual cortex of the cats brain. With this transmitter it was possible to monitor the activation of the neuron. They observed that certain neurons where activated by different input images and found three basic types of neurons.

Convolution Layer

Math and other domains

Definition: $$(f * g)(x) = \int_{\mathbb{R}^n} f(\tau) g(x - \tau) \, d\tau$$



$$f(x, y) * g(x, y) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f(n_1, n_2) \cdot g(x - n_1, y - n_2)$$

https://en.wikipedia.org/wiki/Convolution

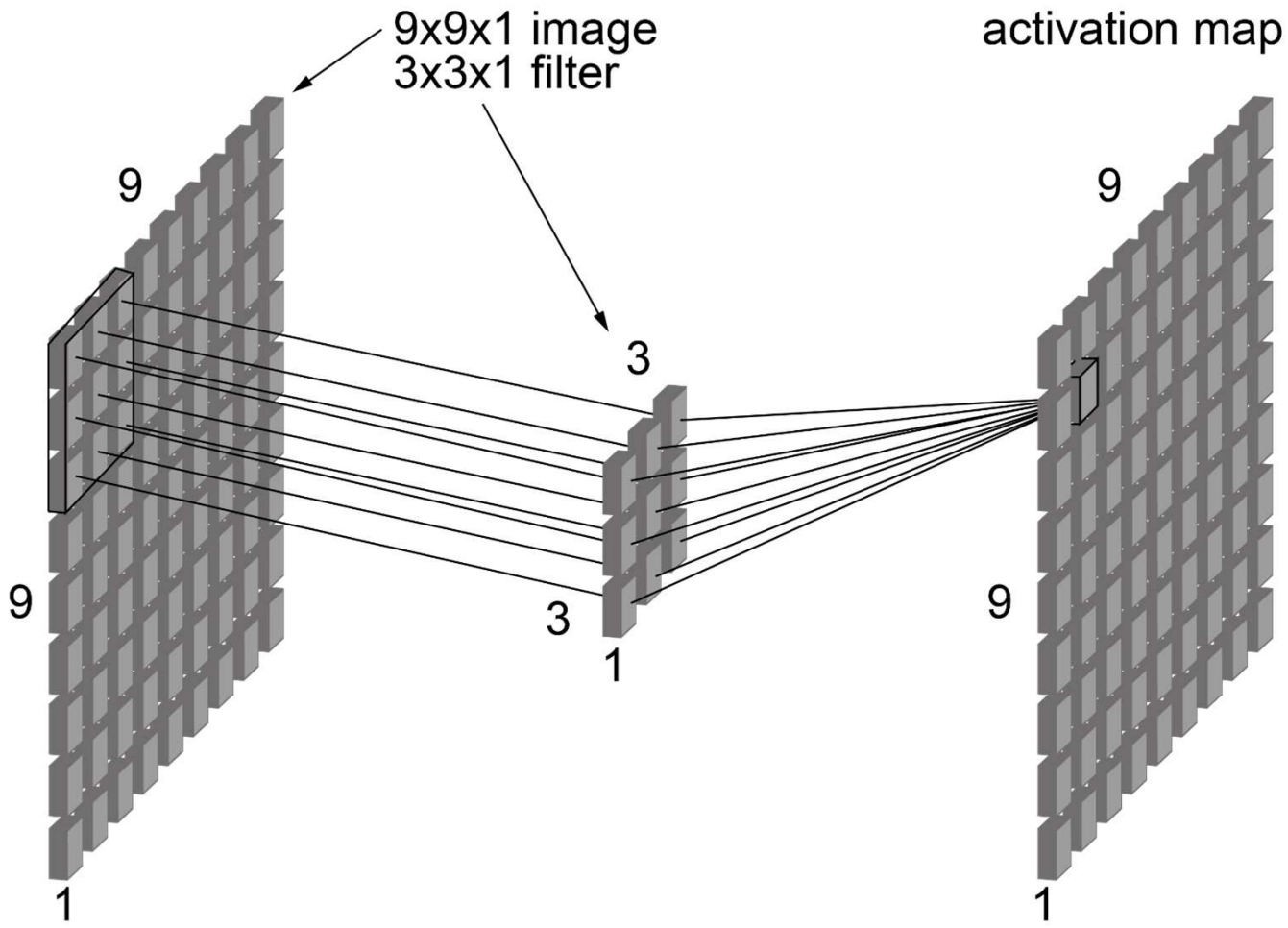
Additional Slides

Convolutions mainly found in other domains. Please visit, to see the animation:

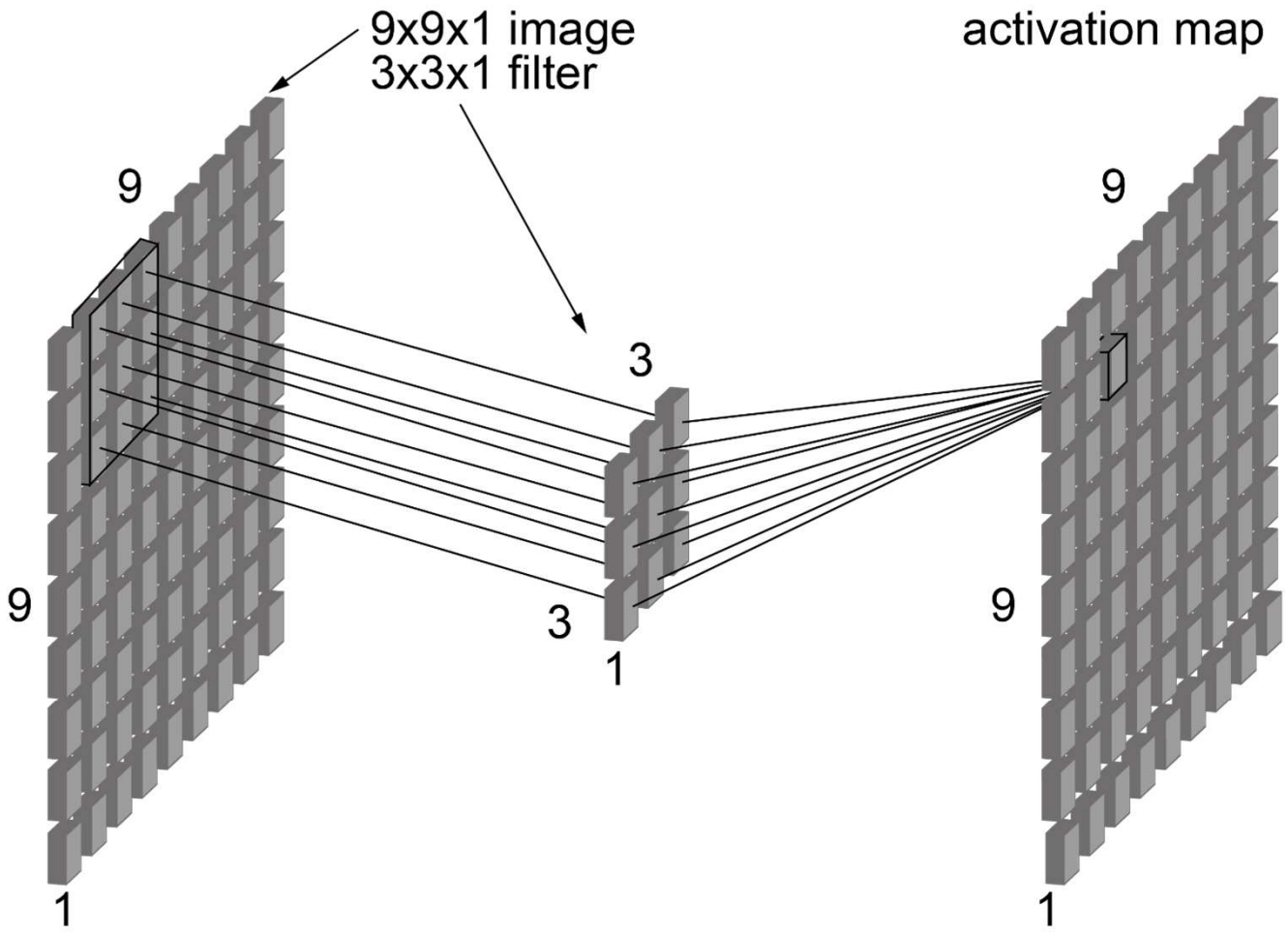
https://en.wikipedia.org/wiki/Convolution

The result of convolution is basically the common area two function share, when on function is moved over the other. Therefore the convolution can approximate the similarity between two functions.

Convolution Layer

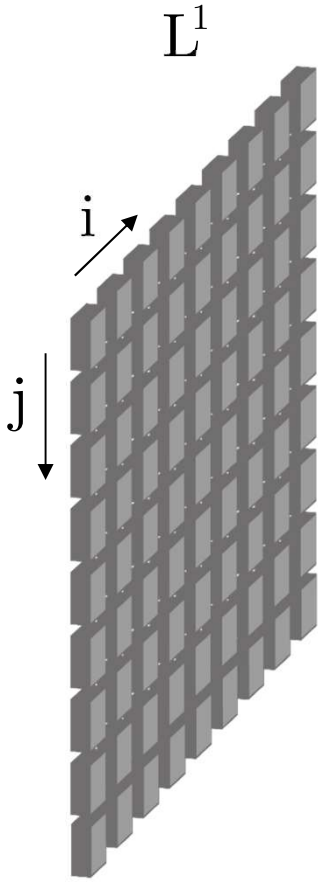


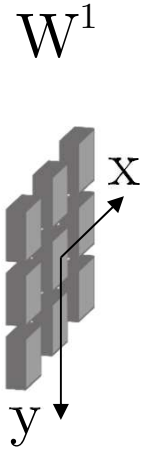
Convolution Layer



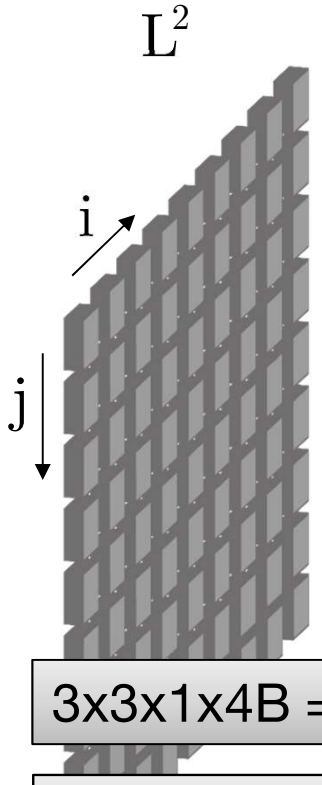
Convolution Layer

$$L^2_{i,j} = f\left(\sum_{x=-1}^{1}\sum_{y=-1}^{1} L^1_{x+i,y+j} \cdot w_{x,y} + b_{i,j}\right)$$





$$W^1 = \begin{pmatrix} w_{-1,-1} & w_{0,-1} & w_{1,-1} \\ w_{-1,0} & w_{0,0} & w_{1,0} \\ w_{-1,1} & w_{0,1} & w_{1,1} \end{pmatrix}$$



3x3x1x4B = 36 B

FullHD?
3x3x1x4B = 36 B

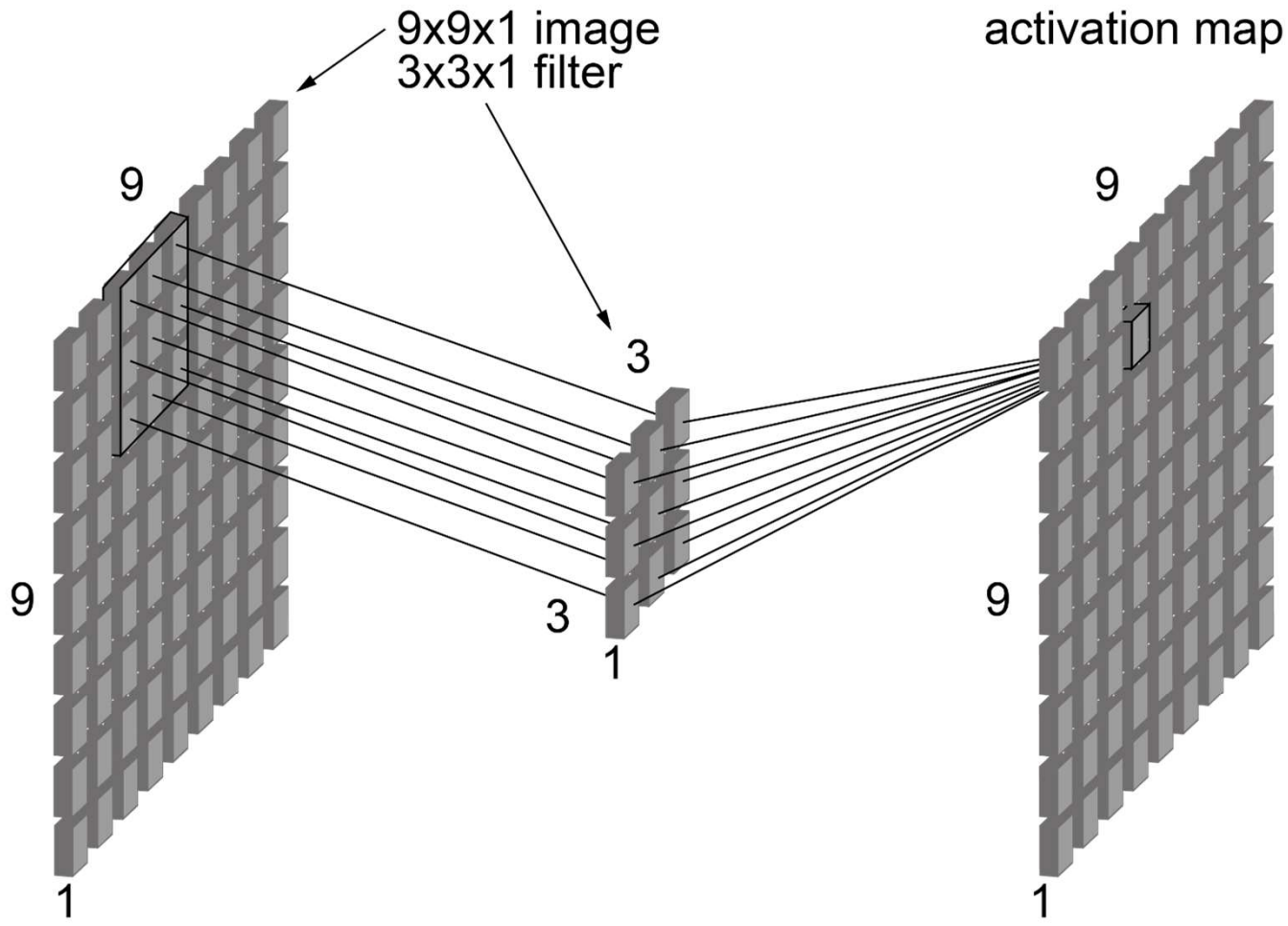
Additional Slides

Similiar as the convolution in maths, we have a „Filter“ (here 3x3) which is moved over the input layer. From left to right and top to bottom. At every position the convolution calculates how similiar the part of the input image is with the filter. The result is a similarity for each region which is the output layer of the convolution, also called activation map.

The „similarity“ is calculated by the sum of multipling the input pixel with the corresponding filter pixel.

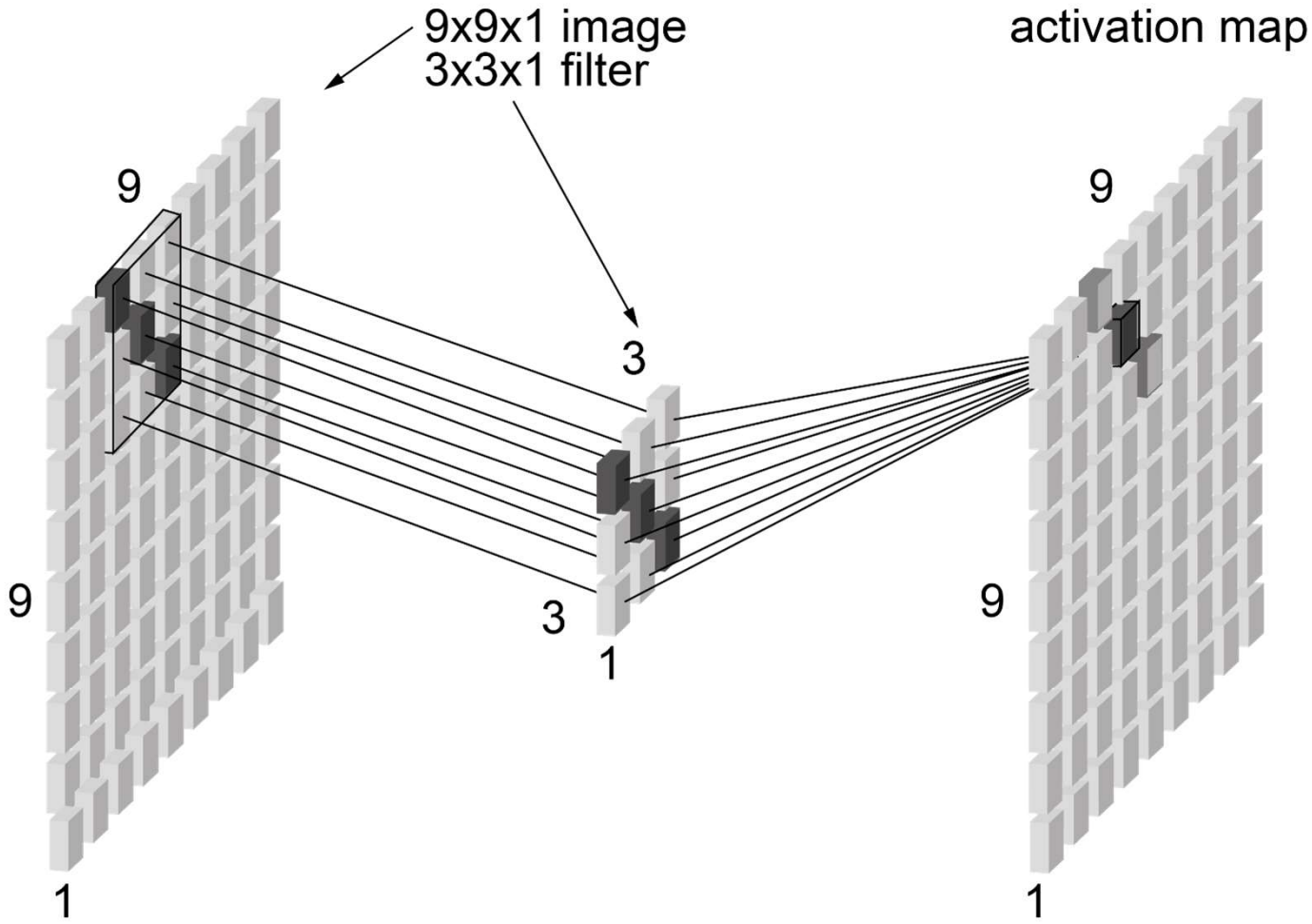
The memory usage is now only determined by the size of the filter and not the size of the images, which reduces the neceissary storage immensily.

Convolution Layer



Convolution Layer

$$L^2_{i,j} = f\left(\sum_{x=-1}^{1}\sum_{y=-1}^{1} L^1_{x+i,y+j} \cdot w_{x,y} + b_{i,j}\right)$$



Additional Slides

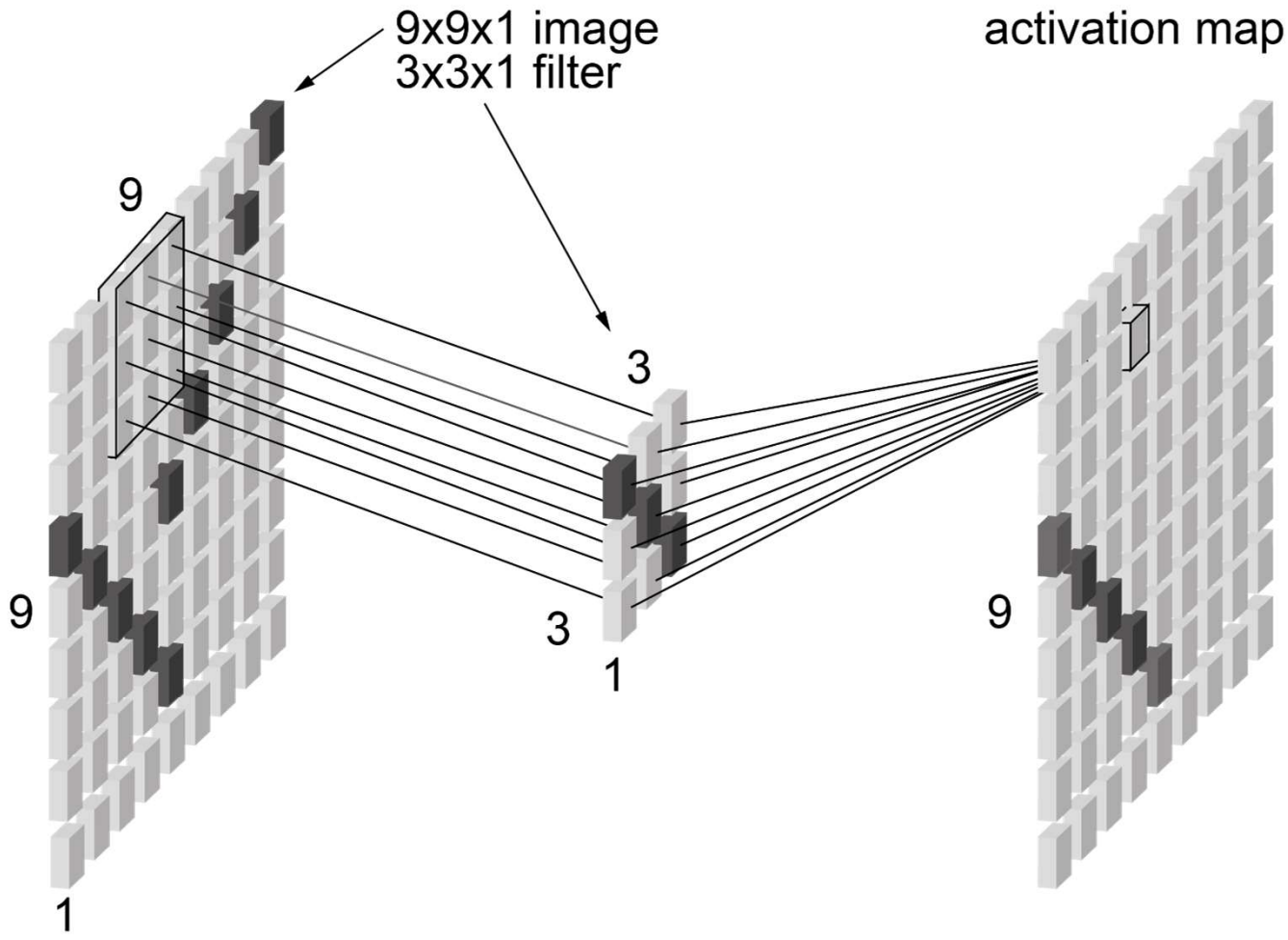
If dark pixel have the value 1 and white pixels have the value 0 the activation of the current convolution in the current position would be 3.

1*1 + 0*0 + 0*0 +
0*0 + 1*1 + 0*0 +
0*0 + 0*0 + 1*1 = 3

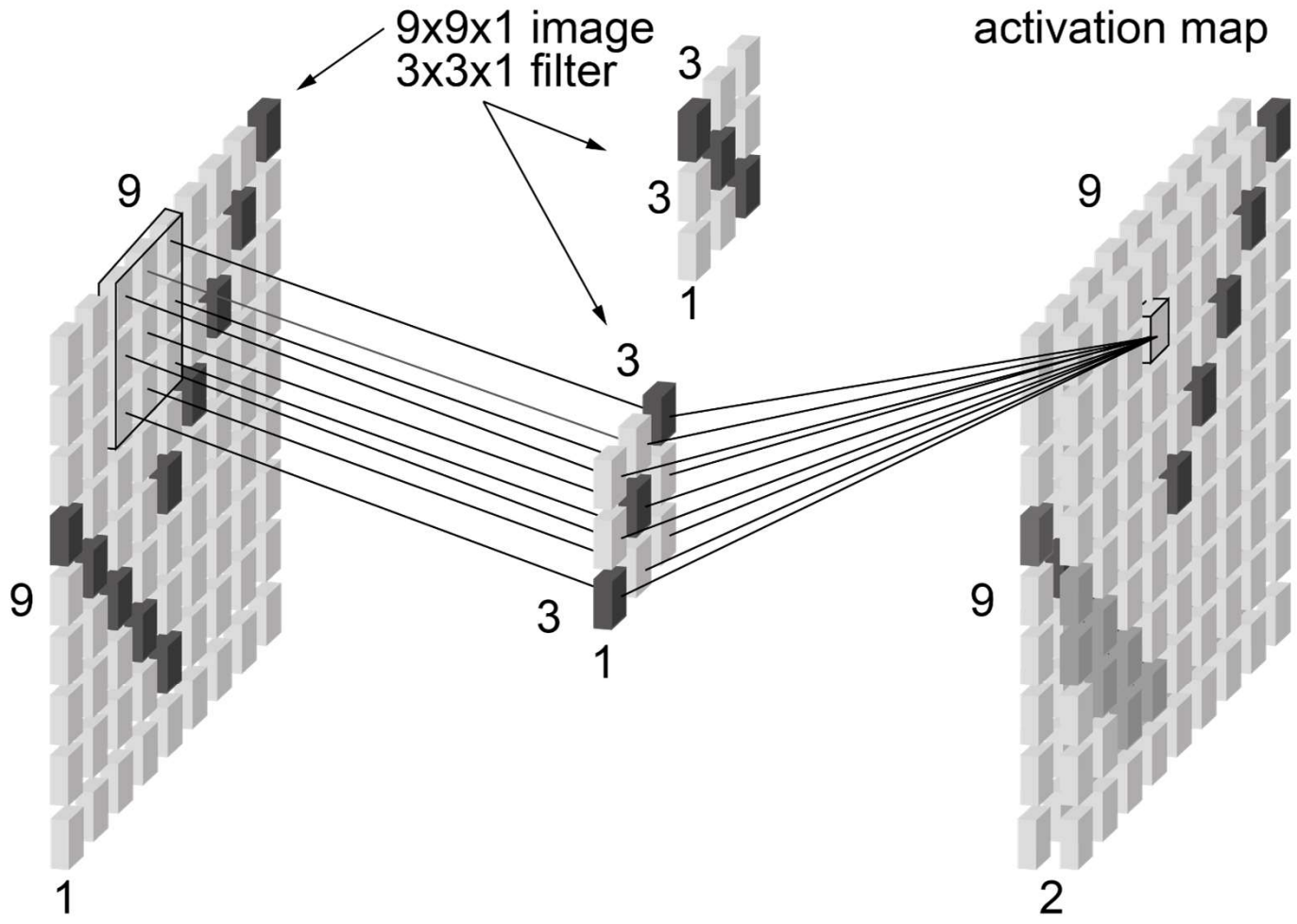
The grey pixel have a smaller value, since the convolution in this part only has two dark input pixel

1*1 + 0*0 + 0*0 +
0*0 + 1*1 + 0*0 +
0*0 + 0*0 + 0*0 = 2

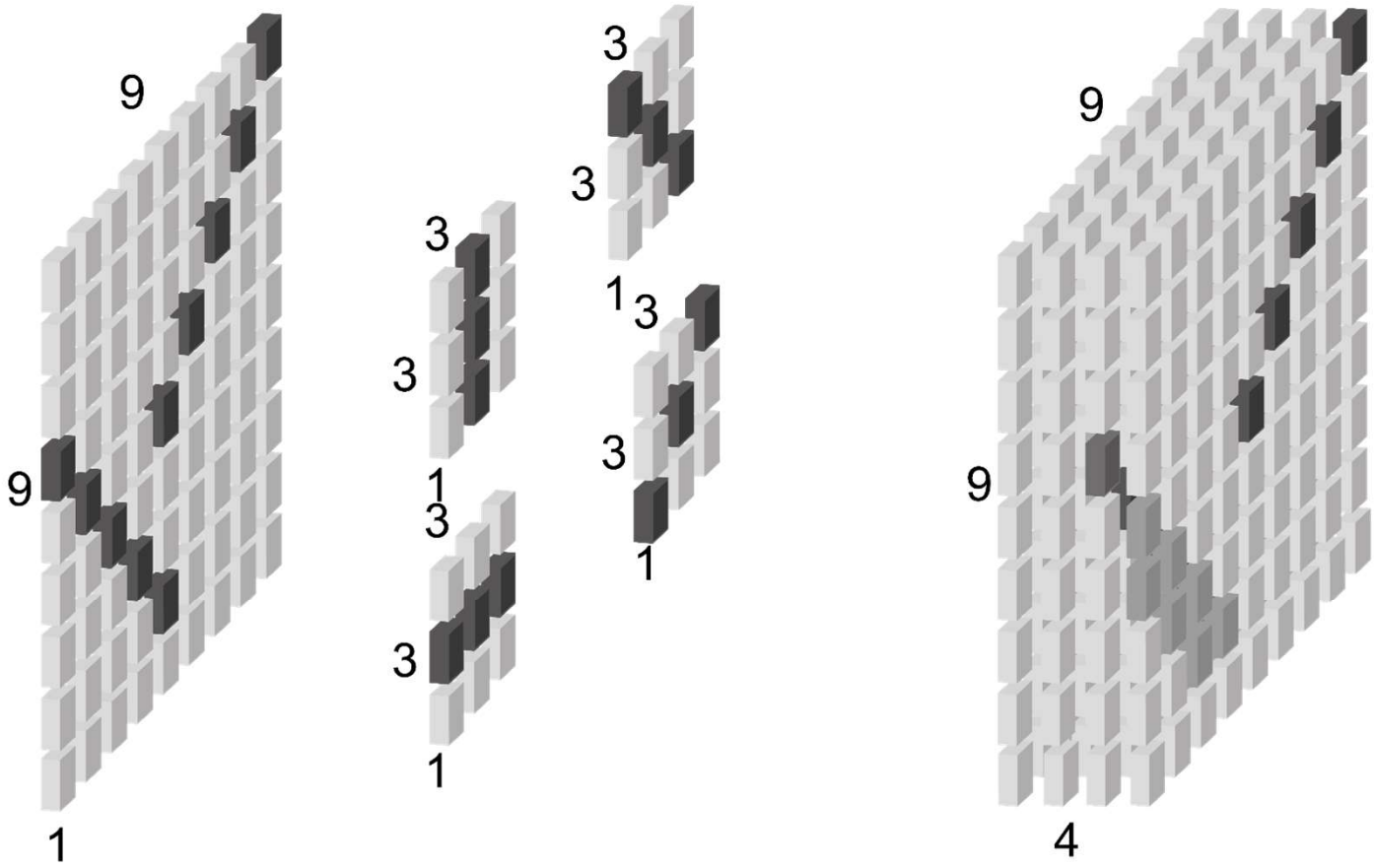
Convolution Layer



Convolution Layer



Convolution Layer

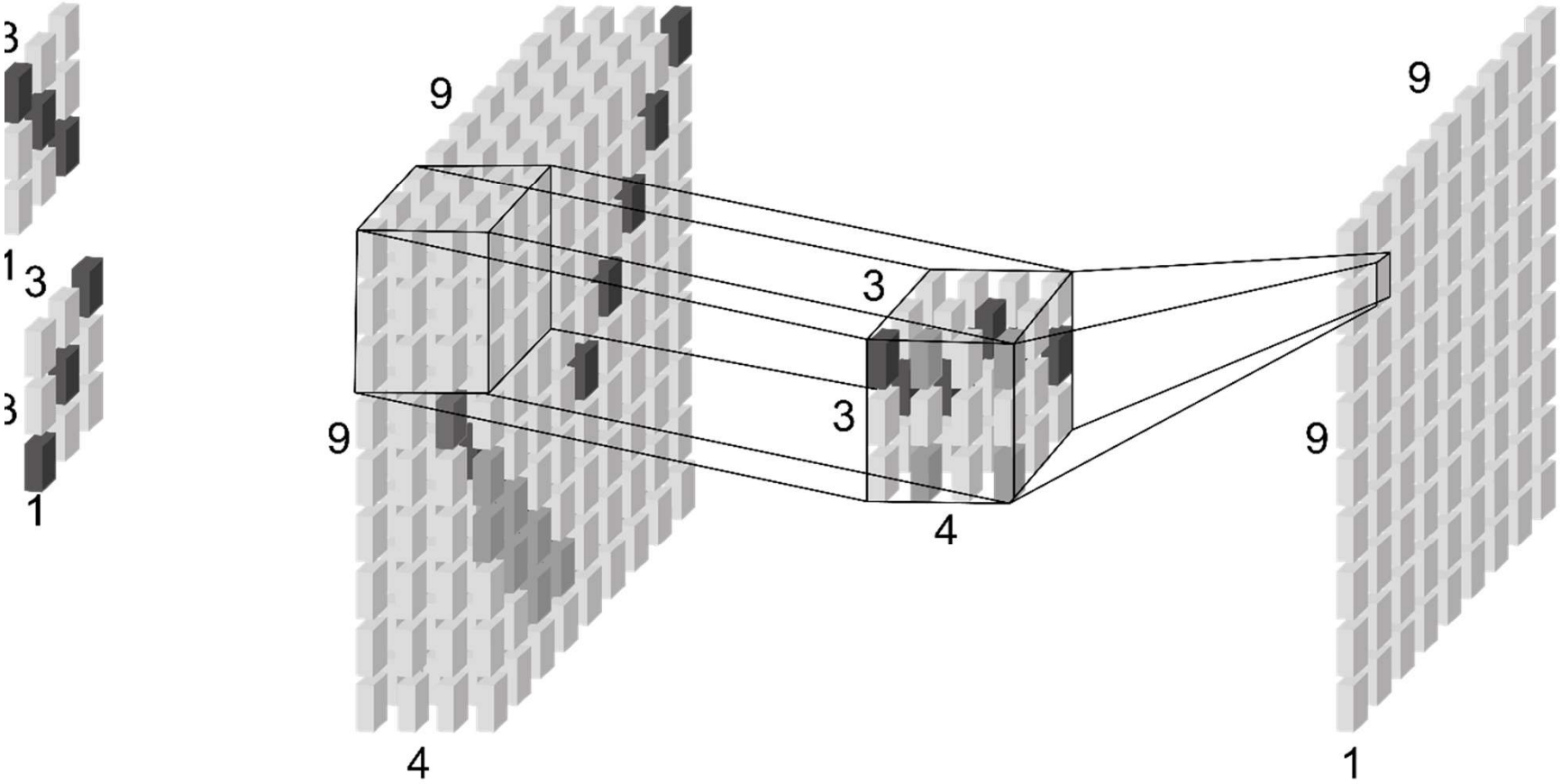


Additional Slides

Usually a convolutional layer does not only have one single filter, but multiple filters. Each filter usually detects a different feature in the image like for example horizontal or vertical lines. The activation maps are stacked on top of each other, resulting in a 3 dimensional output layer.

Convolution Layer

$$L^2_{i,j,k} = f(\sum_{x=-1}^{1} \sum_{y=-1}^{1} \sum_{z=1}^{4} L^1_{x+i,y+j,z} \cdot w^k_{i,j,z} + b^k_{i,j})$$



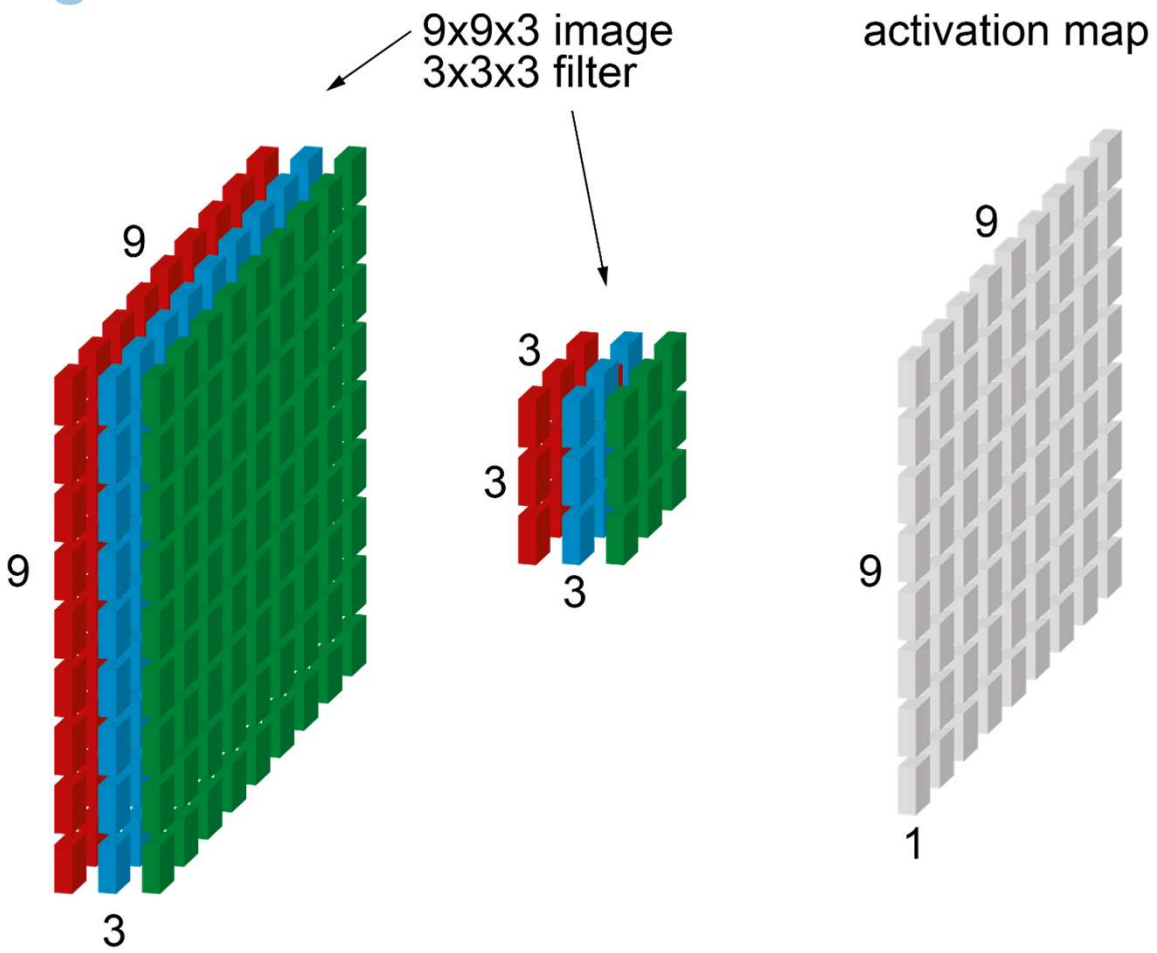
Additional Slides

If the input layer has 3 dimension the filter also becomes 3 dimensional, detecting features over all input layers. Therefore it is becomes increasingly harder to understand features / filters in deeper level of convolutional neural networks.

But how does the neural networks know how the filters have to look like? The neural network doesn‘t! Usually the filters are initialized with random weights and changed during the training by the backpropagation algorithm.

Convolution Layer

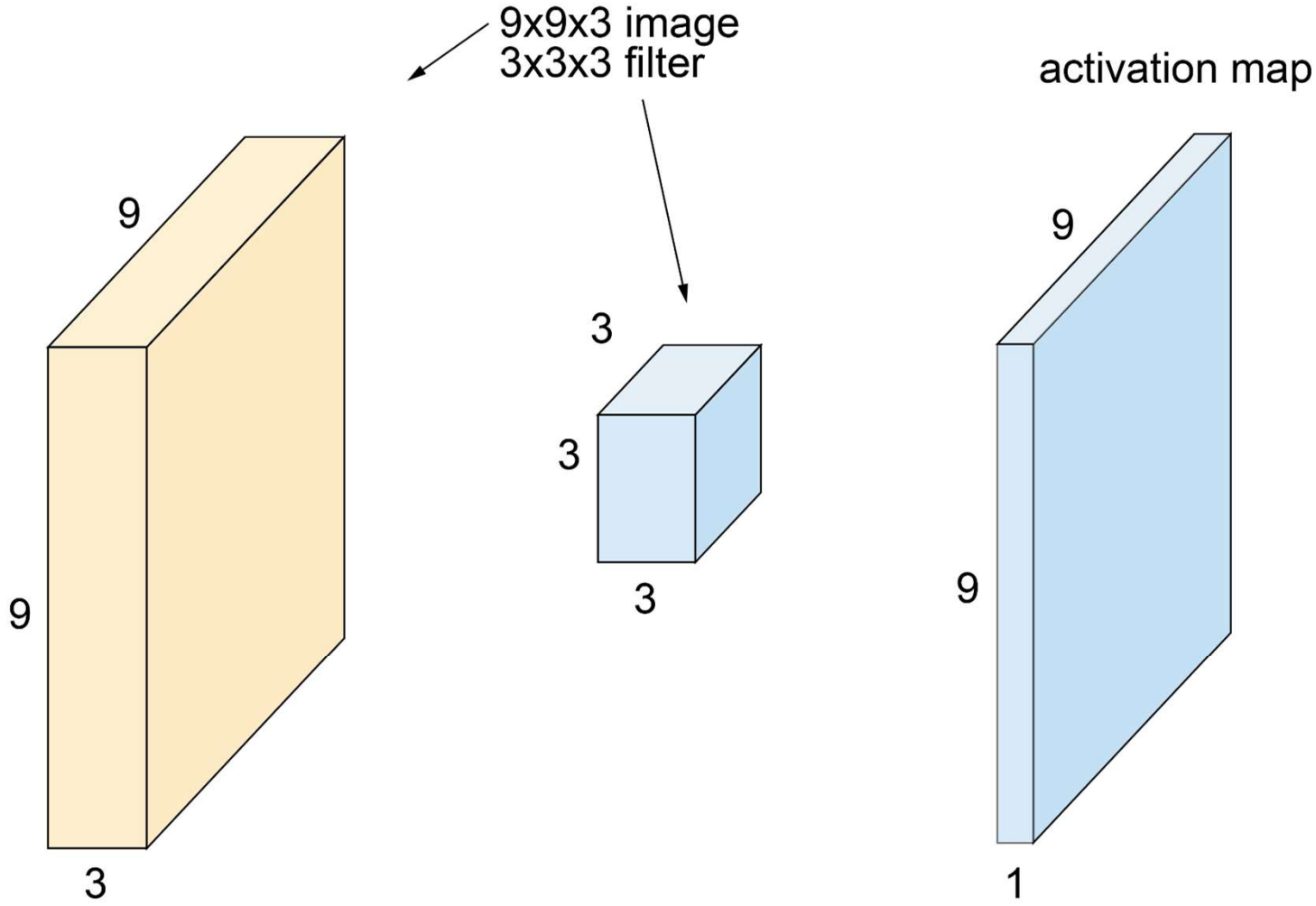
RGB Image



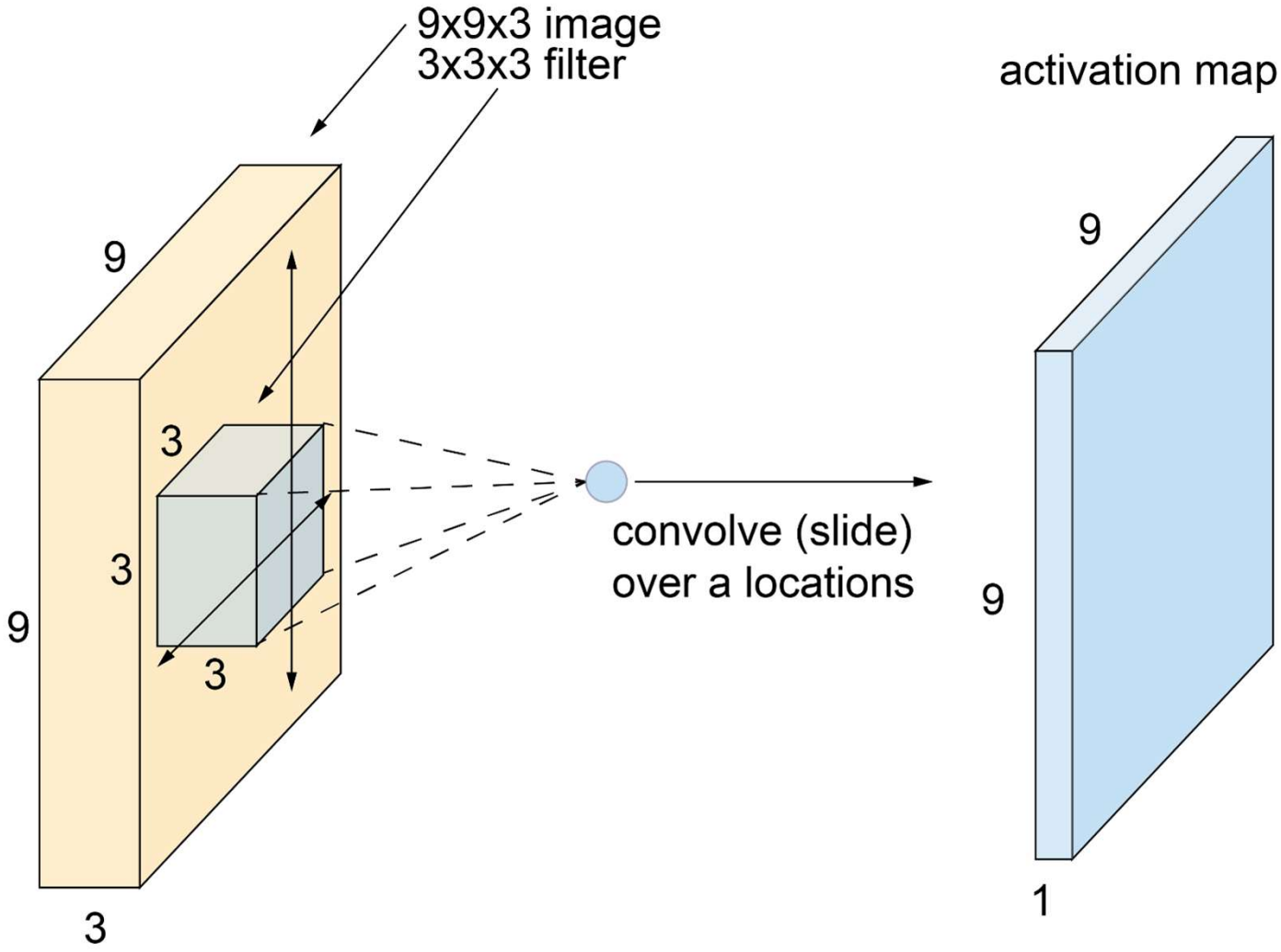
Additional Slides

Usually image data in computer vision consists of the three color components red, green and blue (rgb). Since each pixel has ist own color values, the representation of the image has three dimensions: Width*Height*Colors (= 9 x 9 x 3 in this example). Therefore the first filter of this convolutional layer also has three dimensions, where each plane detects features in the corresponding image plane. Since the planes of the filter corresponds to the rgb planes of the image, the filter itself can be interpreted as an image. There it is possible to visualize filters of the first layer usually quite easilly.

Convolution Layer



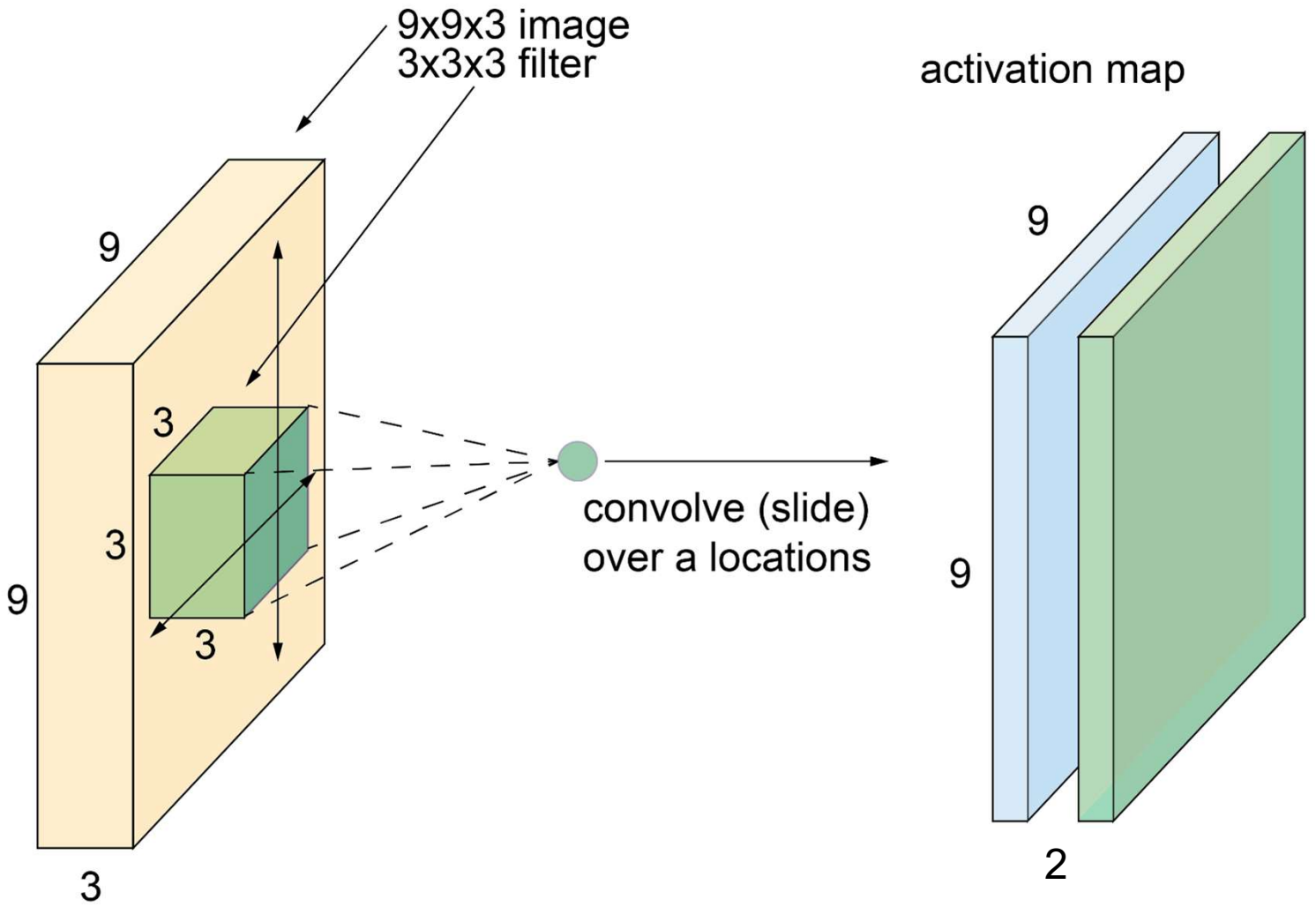
Convolution Layer



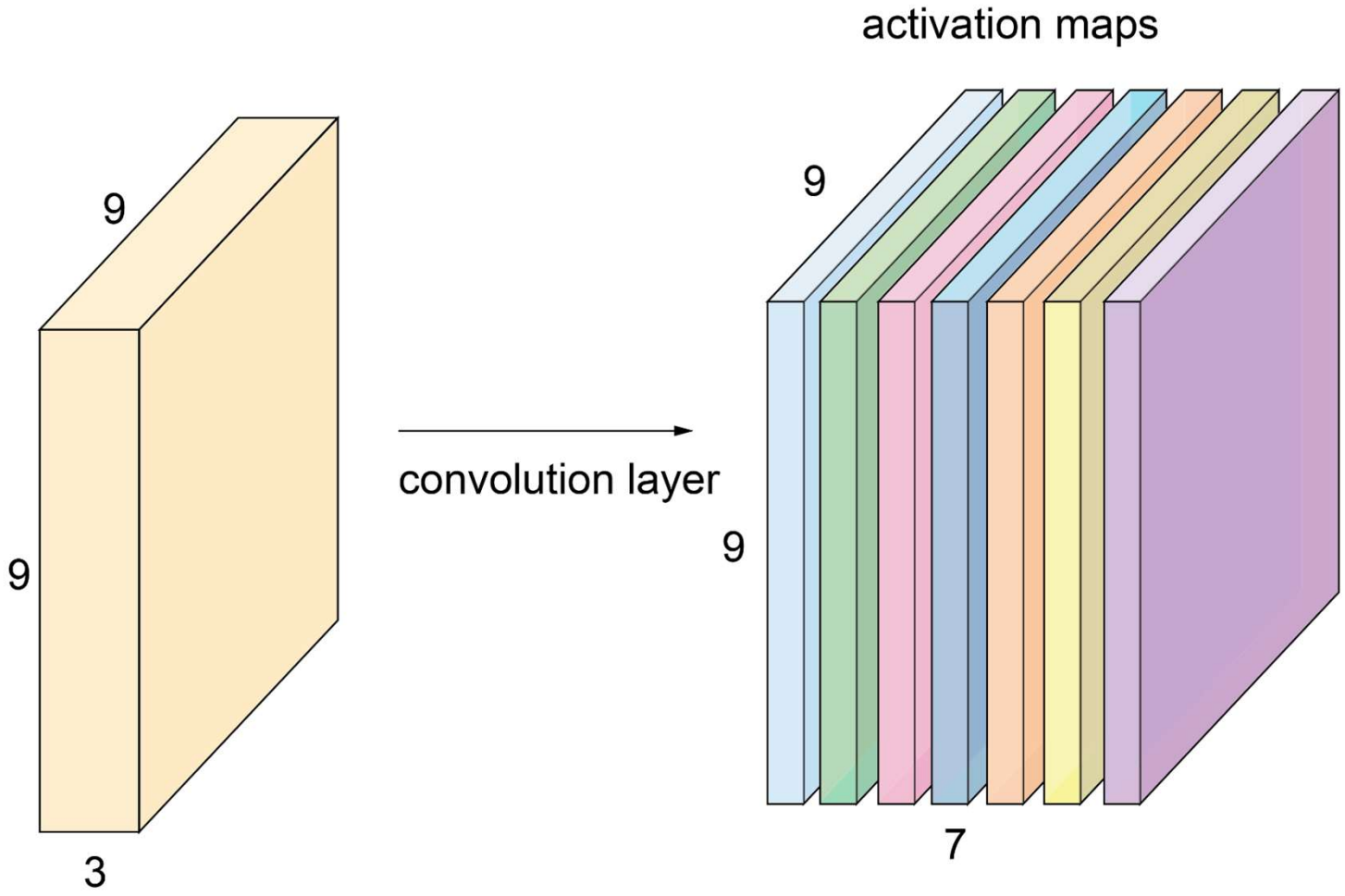
Additional Slides

For the next slides, we will stop using the pixel representation and start using three dimensional cubes, which have the same meaning. The convolution operation can then be intrepreted as moving the filter-cube inside the input cube with a discrete step size.

Convolution Layer



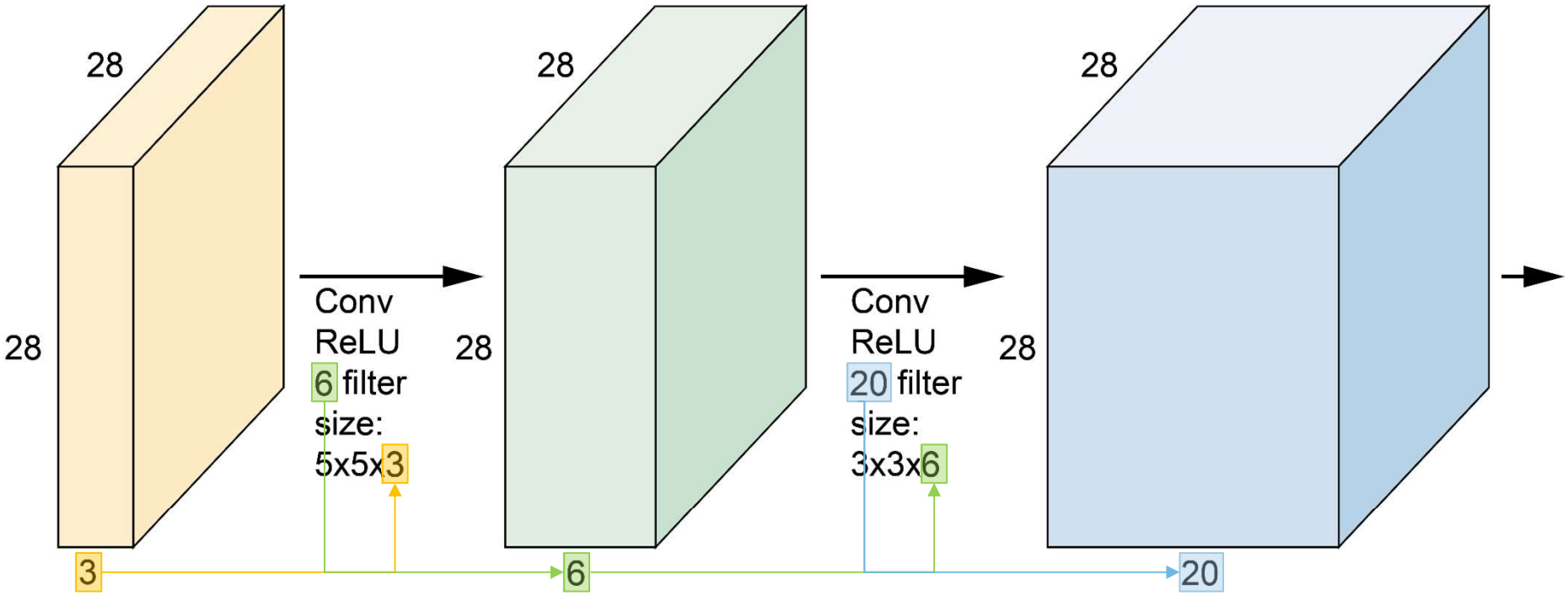
Convolution Layer



Additional Slides

Also here multiple filters produce each an activation map. Those activation maps get stacked resulting in an three dimensional output layer. The output dimensions are proportional to the input layers width and height as well as the convolutional layer filter count.

Convolution Layer

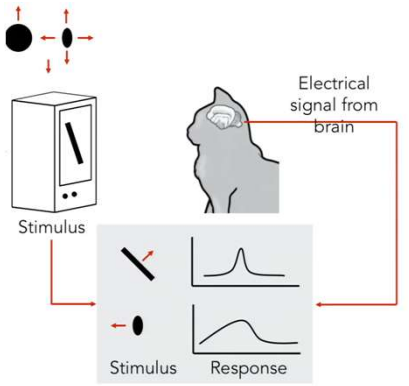


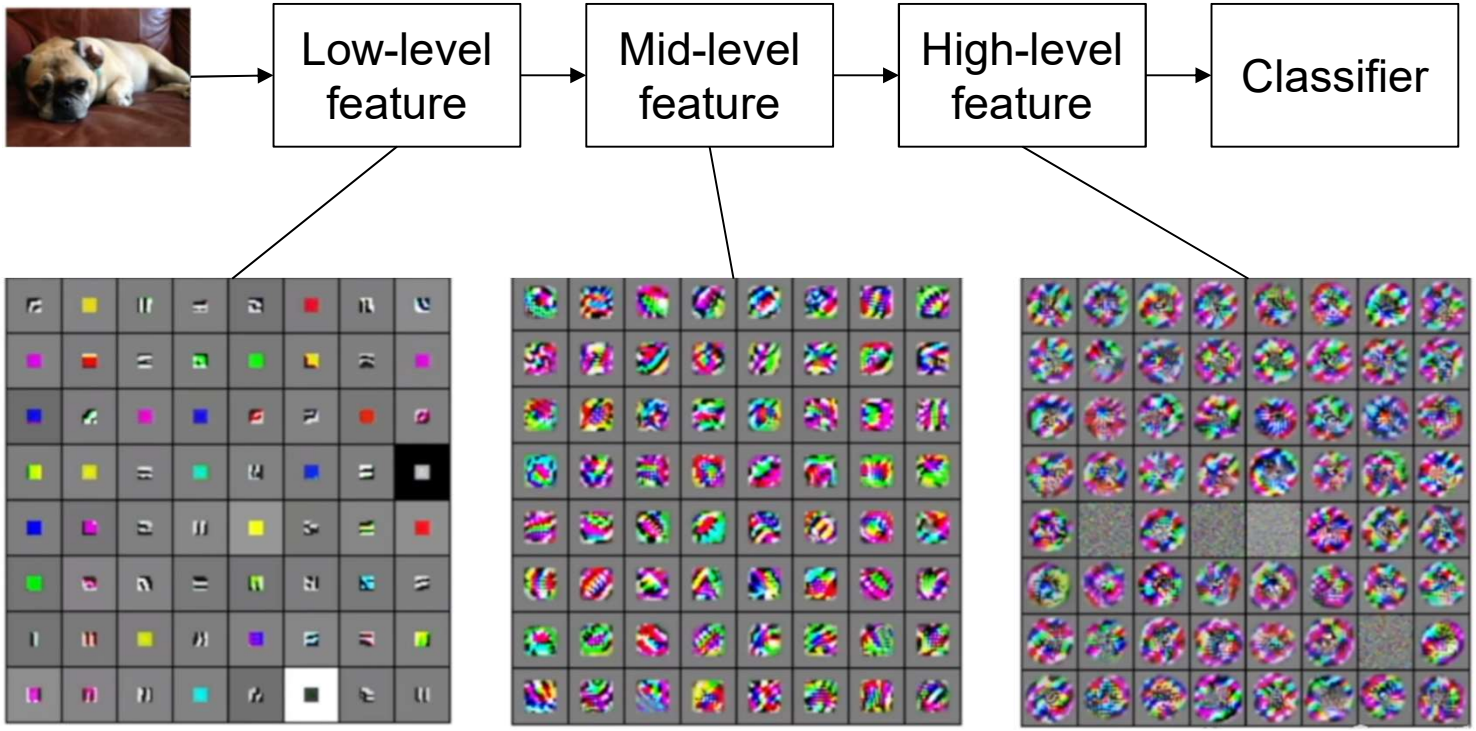
Additional Slides

The third dimension of the filter corresponds to the third dimension of the input layer. The Third dimension of the output layer corresponds to the filter count of the convolutional layer.

Convolution Layer

Filter Visualisation



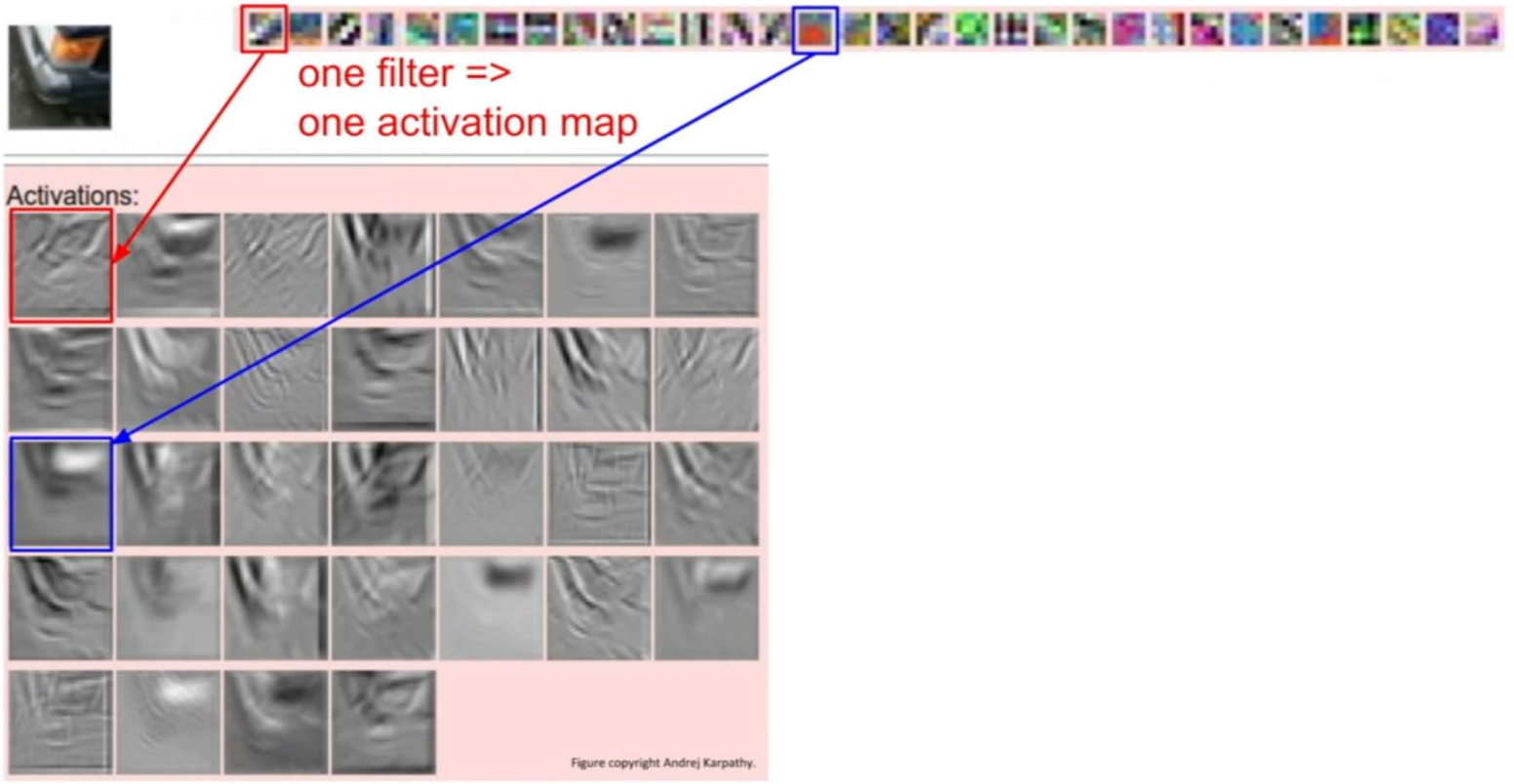


Additional Slides

Usually multiple convolutional layer are used in series. The features which are detechted by each layer usually get more complex with deeper layer count. As explained befored, low level features like lines and colors are detected in the first convolutional layer. Higher layer features get more complex and hard to interprete. This follows the findings of huber and wiesel, where they looked into different neuron types in a cats brain.

Convolution Layer

Activation Map Visualisation

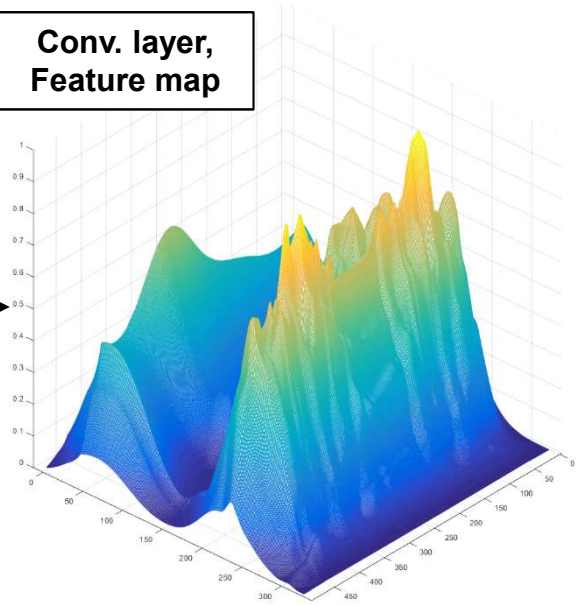


Convolution Layer

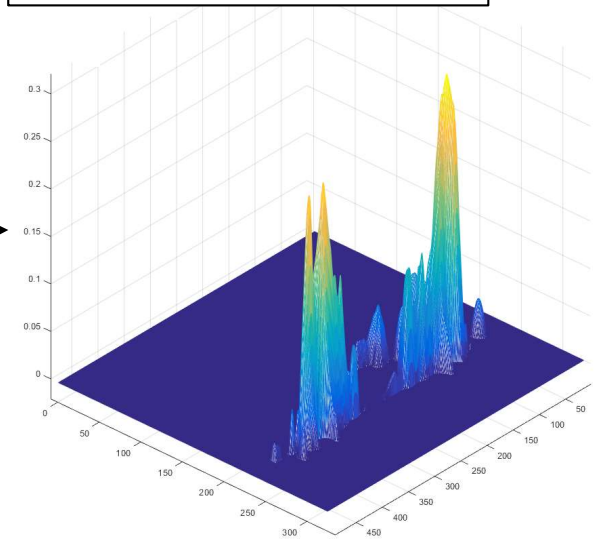
Visualisation

Input Layer

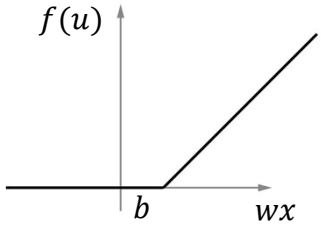
Conv. layer, Feature map



Relu Aktivierungfuntkion b=-0.7



Filter 1



Additional Slides

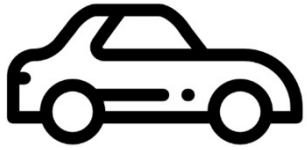
Convolutional layers often use a relu activation function with an additional bias. In this example we can visualize the effect of the bias on the output activation map for a specific filter. The Filter is used to detect circles (tires) in am image. The result of the convolution of the inpu timage and Filter 1 is shown in the middle picture. Cleary two spikes occur where the wheels are. Too isoloate the two spikes an additional bias is added and the relu function is applied. Since adding a bias is equivalant to moving the function to the right, low activation (<b) don‘t result in an activation at all. Therefore on the two spikes remain in the output activation (image on the right). The bias is also learned by the nerual network algorithm thorugh backpropagation.

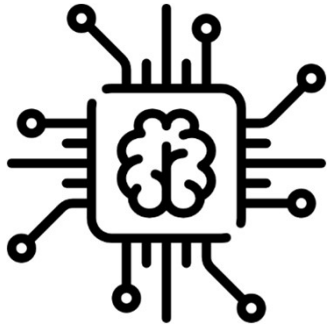
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

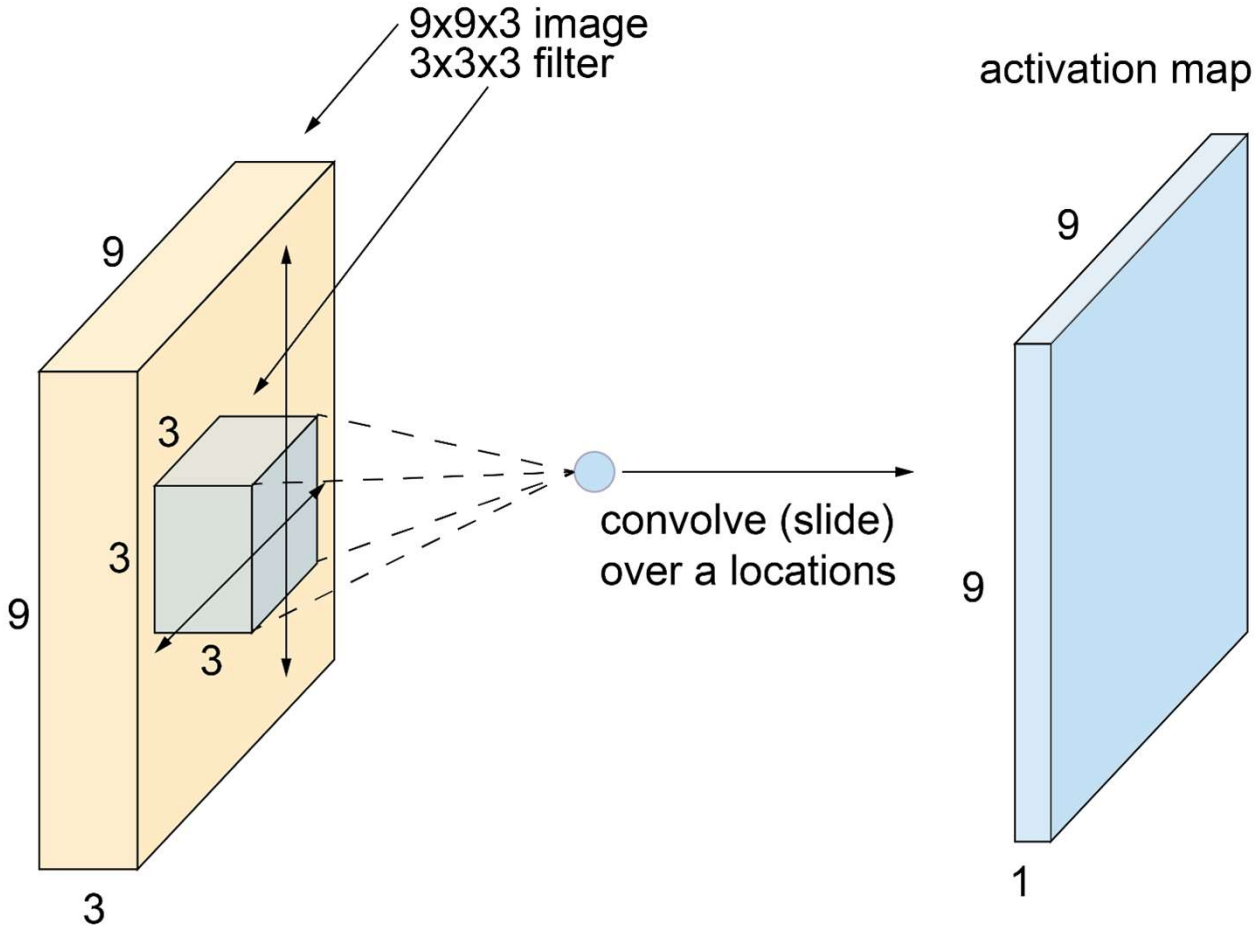
1. Chapter: Convolutional Neural Networks
    - 1.1 Motivation
    - 1.2 Introduction
    - 1.3 Dimensions, Padding, Stride
2. Chapter: Pooling
3. Chapter: CNN in Action
4. Chapter: Advanced Topics
    - 2.1 Optimizer
    - 2.2 Data Formatting





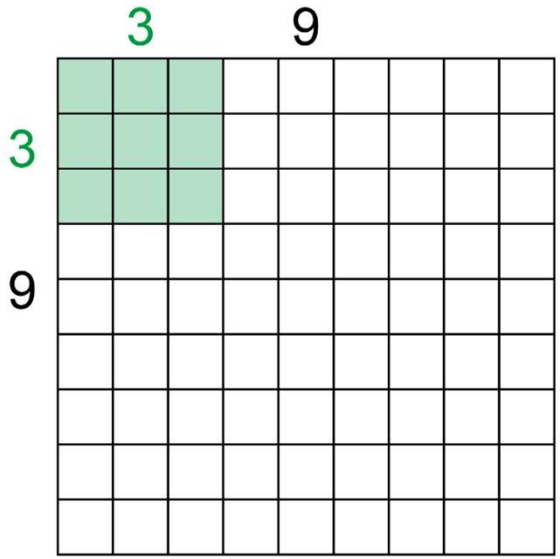
Convolution Layer

Dimensions



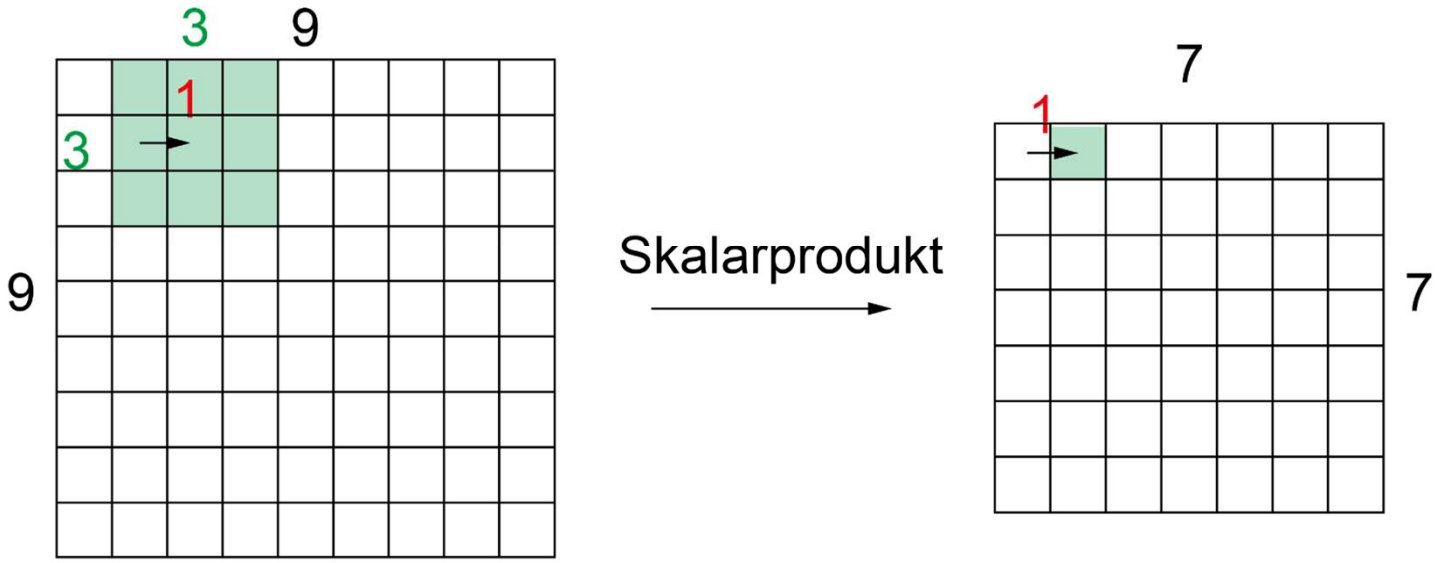
Convolution Layer

Dimensions



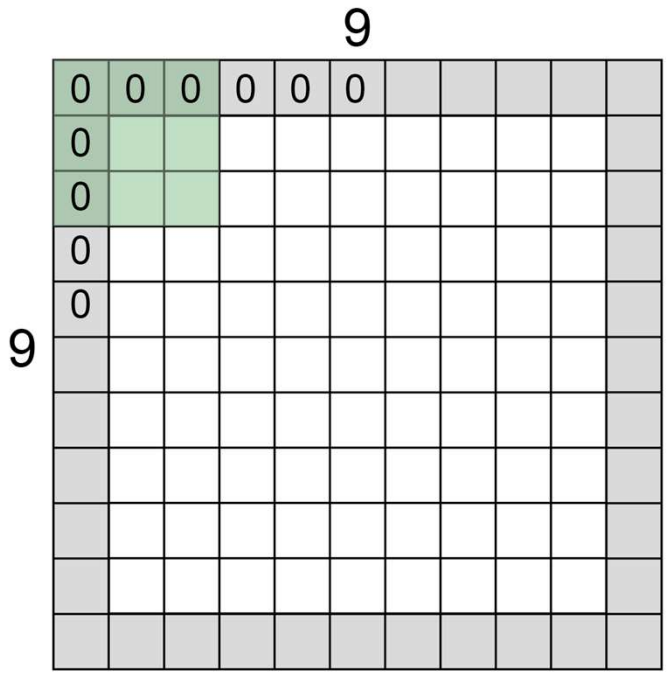
Convolution Layer

Dimensions



Convolution Layer

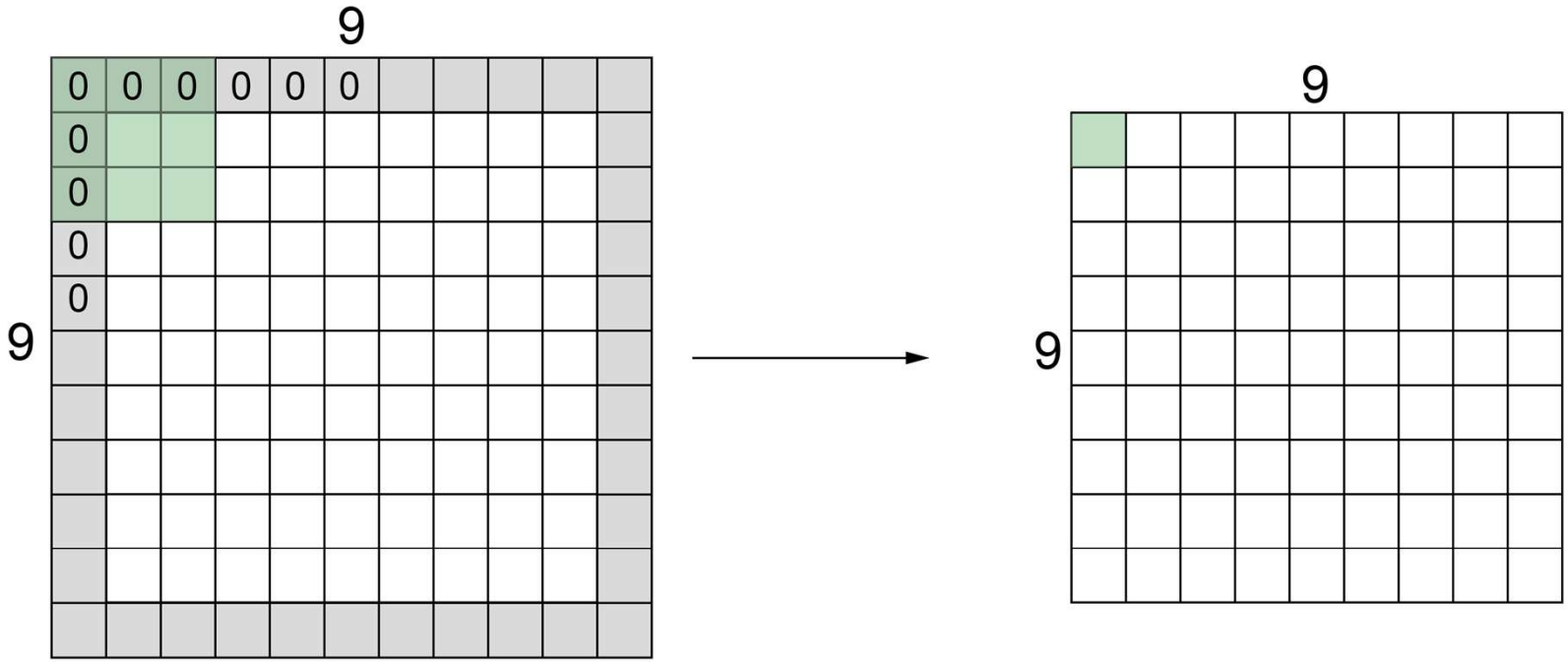
Dimensions: Padding



input 9x9
3x3 filter, applied with stride 1
what is the output?

Convolution Layer

Dimensions: Padding

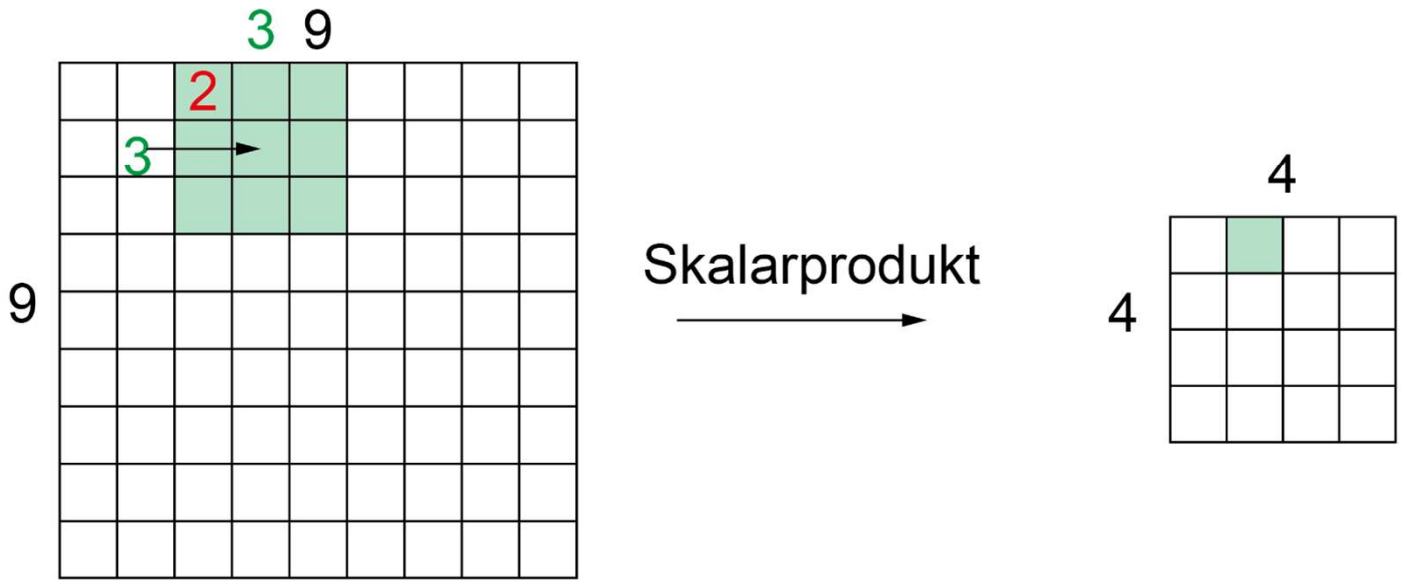


Additional Slides

Let‘s have a closer look at the dimensions of a convolutional layer. There are different factors which have an influence on the output dimensions of the layer. The convolution of a 3x3 filter over a 9x9 input layer, with the stepsize 1 results in a 7x7 output layer. This would shrink the images with every convolution layer by a couple of pixels resulting in a small output image for langer neural networks. The common practice, is to keep the output dimension the same by adding a padding around the edges of the input images. Those values usually get set to zero.

Convolution Layer

Dimensions: Stride



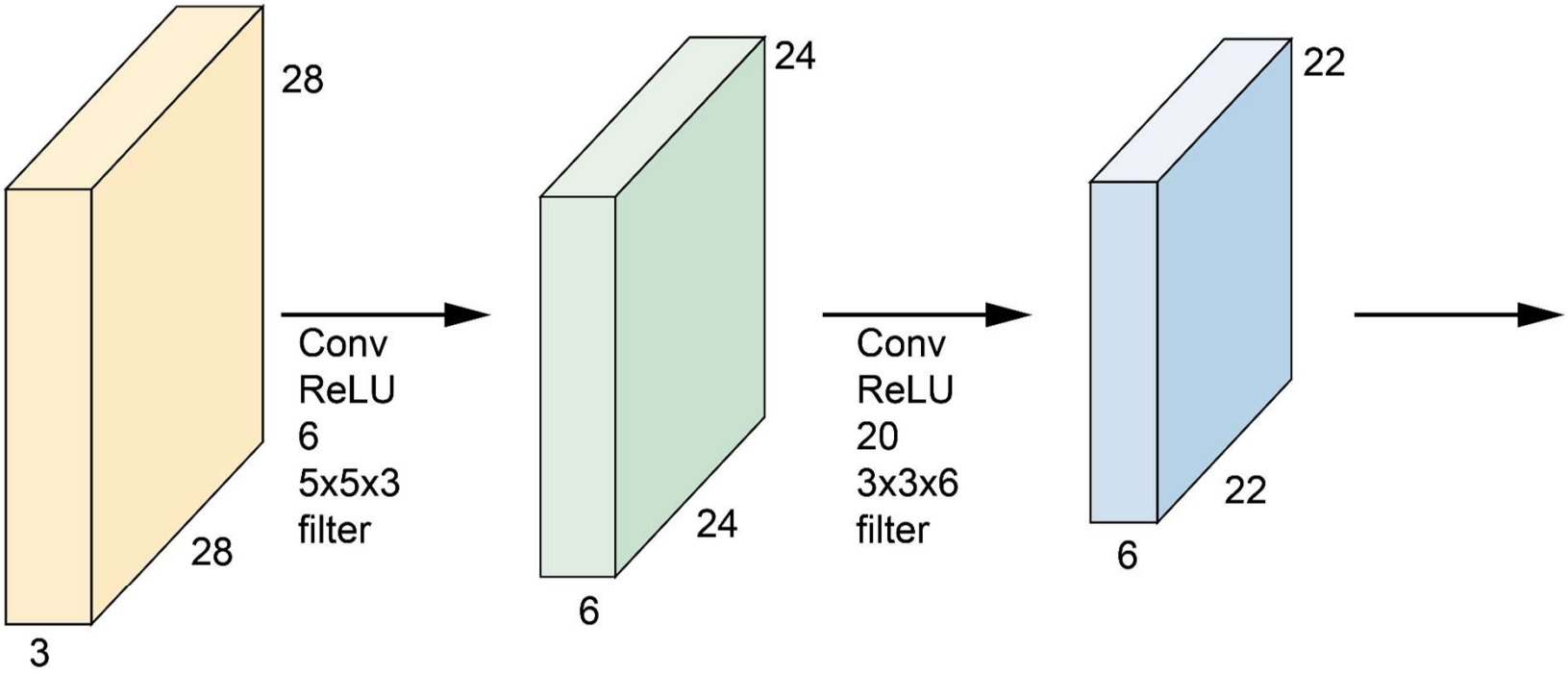
Additional Slides

Another factor is the stepsize. Sometimes it makes sense to use larger stepsize than one, but usually one is the default stepsize.

Convolution Layer

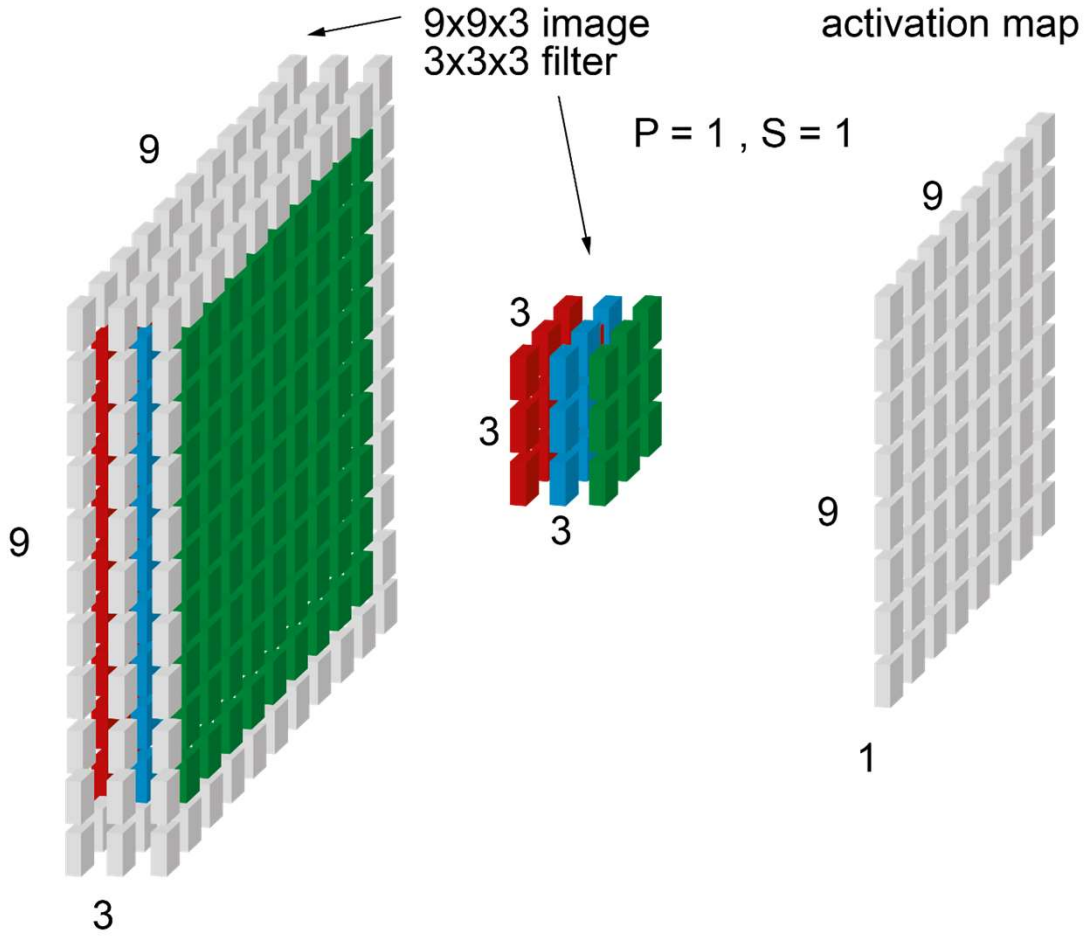
Padding

Without padding, volume shrinks with every convolution layer. This can cause problems in very deep neuronal networks. Use padding!



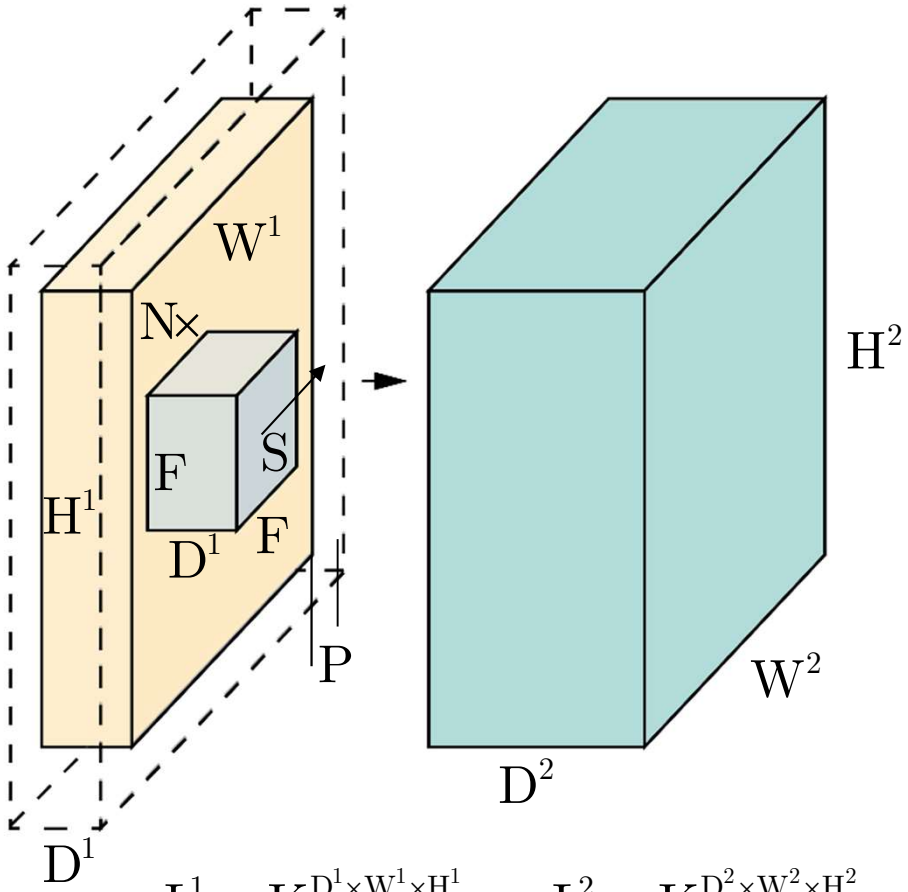
Convolution Layer

Padding with RGB Image



Convolution Layer

Dimensions Formula



$$L^1 \in K^{D^1 \times W^1 \times H^1} \rightarrow L^2 \in K^{D^2 \times W^2 \times H^2}$$

N : Number of Filters

F : Filtersize

S : Stride

P : Padding

New Dimensions:

$$W^2 = \frac{W^1 + 2P - F}{S} + 1$$

$$H^2 = \frac{H^1 + 2P - F}{S} + 1$$

$$D^2 = N$$

Keep Dimensions:

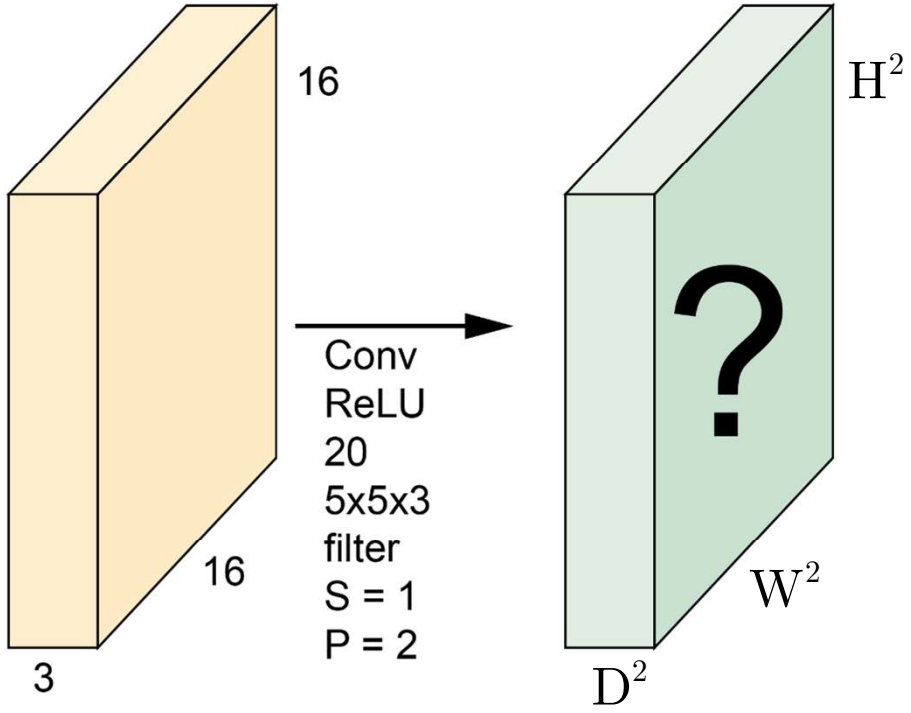
$$F = 3 \Rightarrow P = 1$$

$$F = 5 \Rightarrow P = 2$$

$$F = 7 \Rightarrow P = 3$$

Convolution Layer

Example



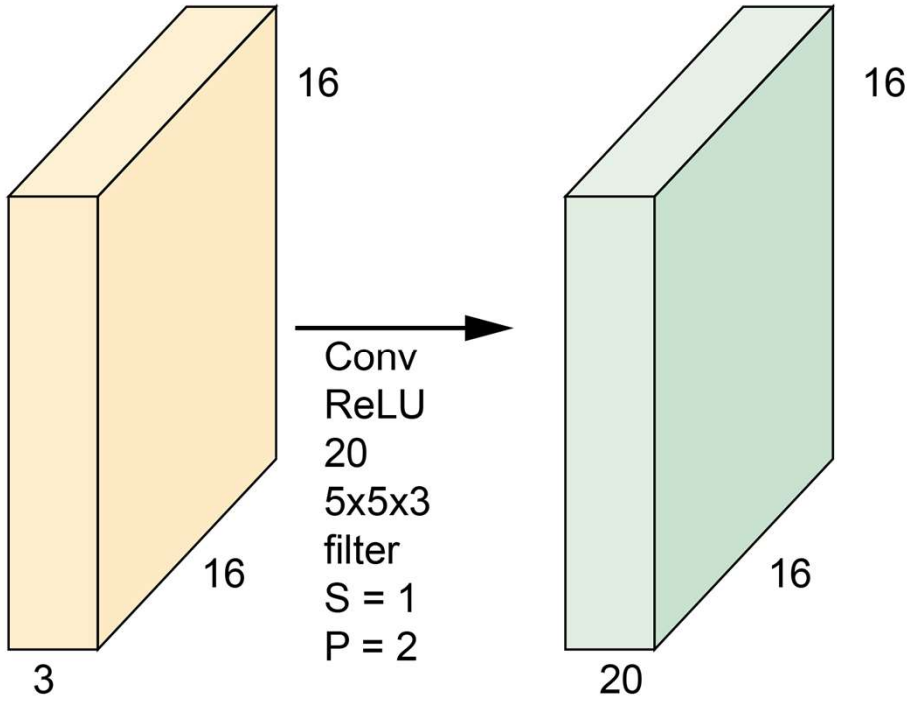
$$W^2 = \frac{W^1 + 2P - F}{S} + 1$$

$$H^2 = \frac{H^1 + 2P - F}{S} + 1$$

$$D^2 = N$$

Convolution Layer

Example



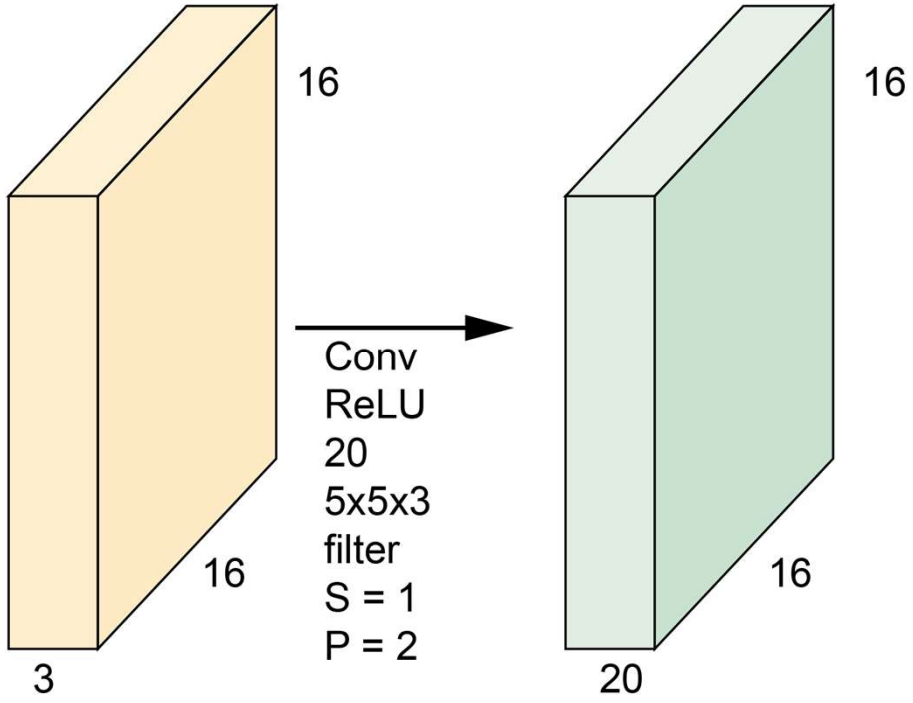
$$W^2 = \frac{16 + 2 \cdot 2 - 5}{1} + 1 = 16$$

$$H^2 = \frac{16 + 2 \cdot 2 - 5}{1} + 1 = 16$$

$$D^2 = 20$$

Convolution Layer

Example



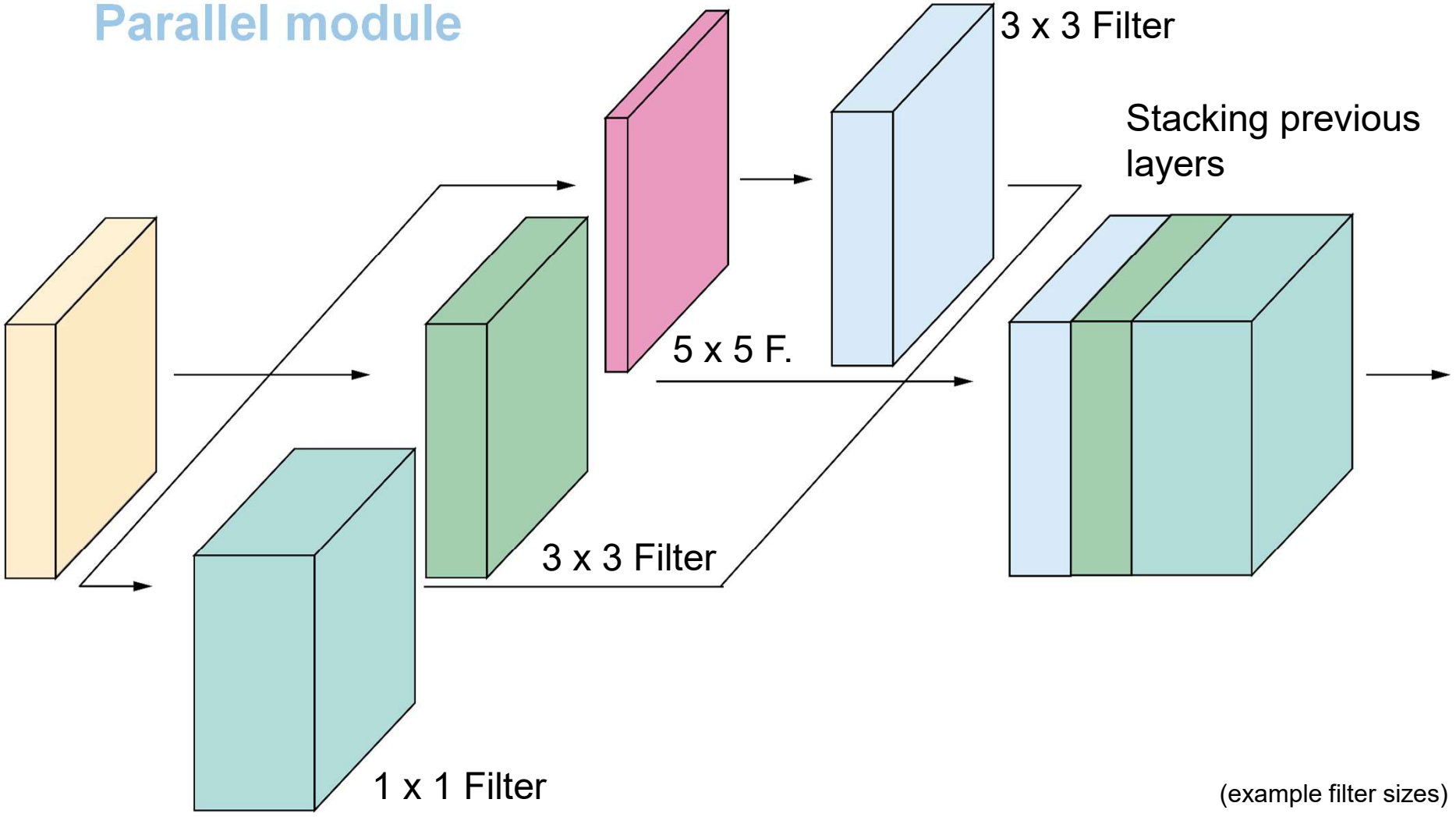
Number of parameters in this layer?

$5 \times 5 \times 3 + 1 = 76$ Paramter per filter

$20 \cdot 76 = 1520$ Paramter

Convolution Layer

Parallel module



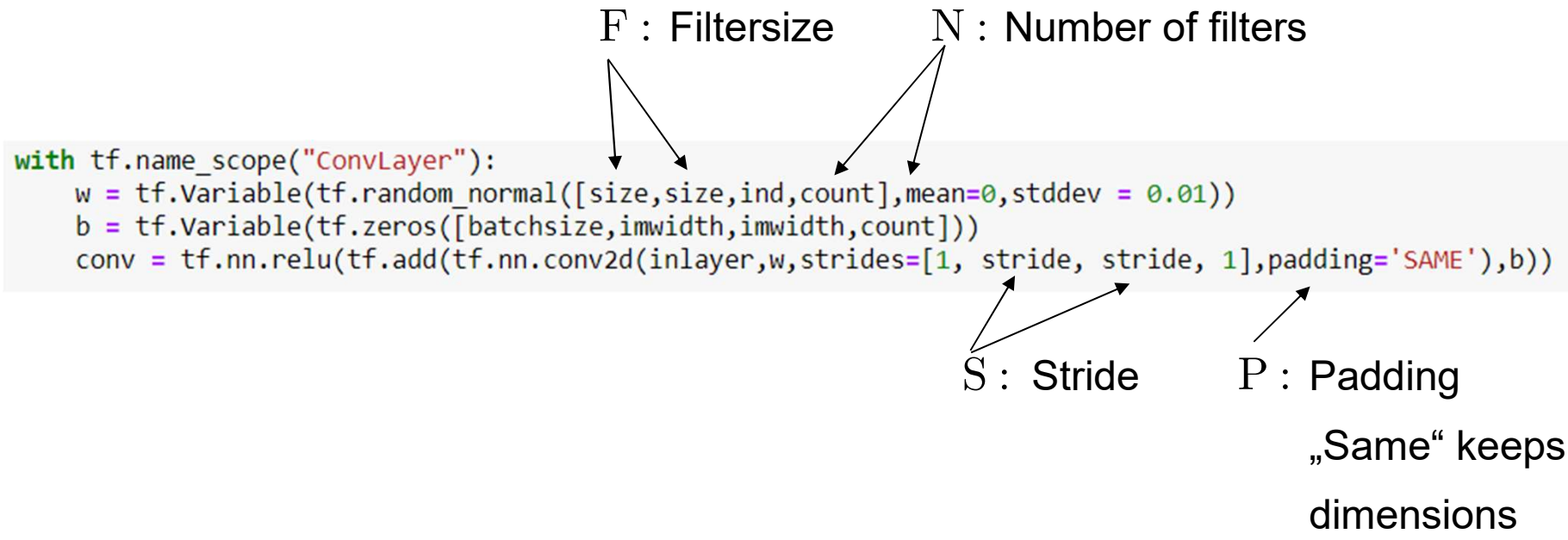
(example filter sizes)

Additional Slides

In more advanced neural networks convolutional layers are often used in parallel. Each of those layers uses a different filter size. Where small filters detect smaller features and large filters detected larger features. The results are than stacked again. The resulting layer preserves more information, since it consists of the results of different convolutional layers.

Convolution Layer

Code Example

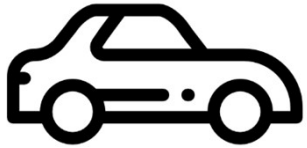


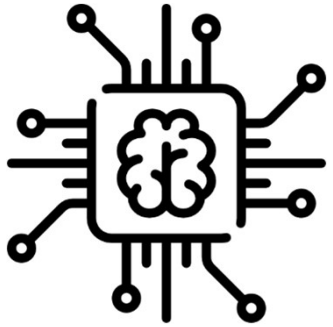
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

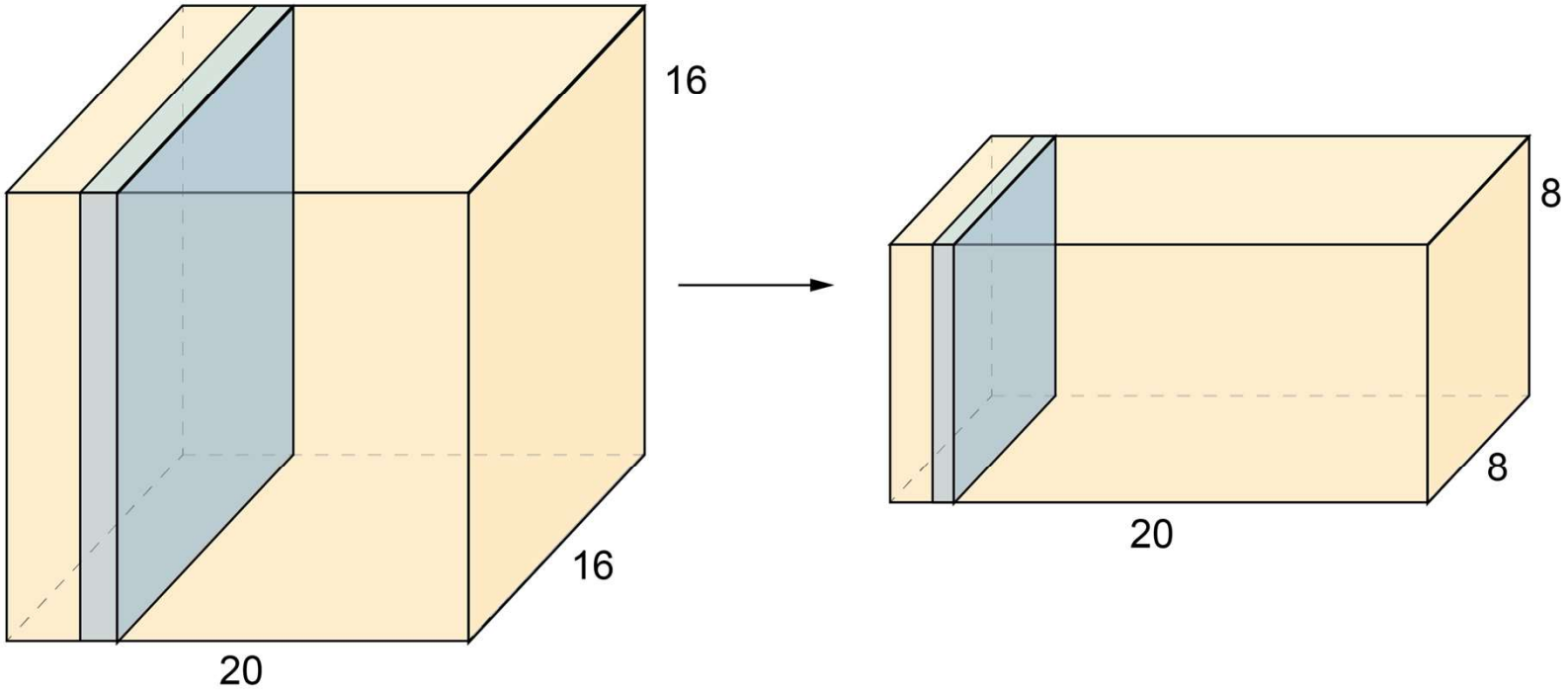
1. Chapter: Convolutional Neural Networks
    - 1.1 Motivation
    - 1.2 Introduction
    - 1.3 Dimensions, Padding, Stride
2. Chapter: Pooling
3. Chapter: CNN in Action
4. Chapter: Advanced Topics
    - 2.1 Optimizer
    - 2.2 Data Formatting





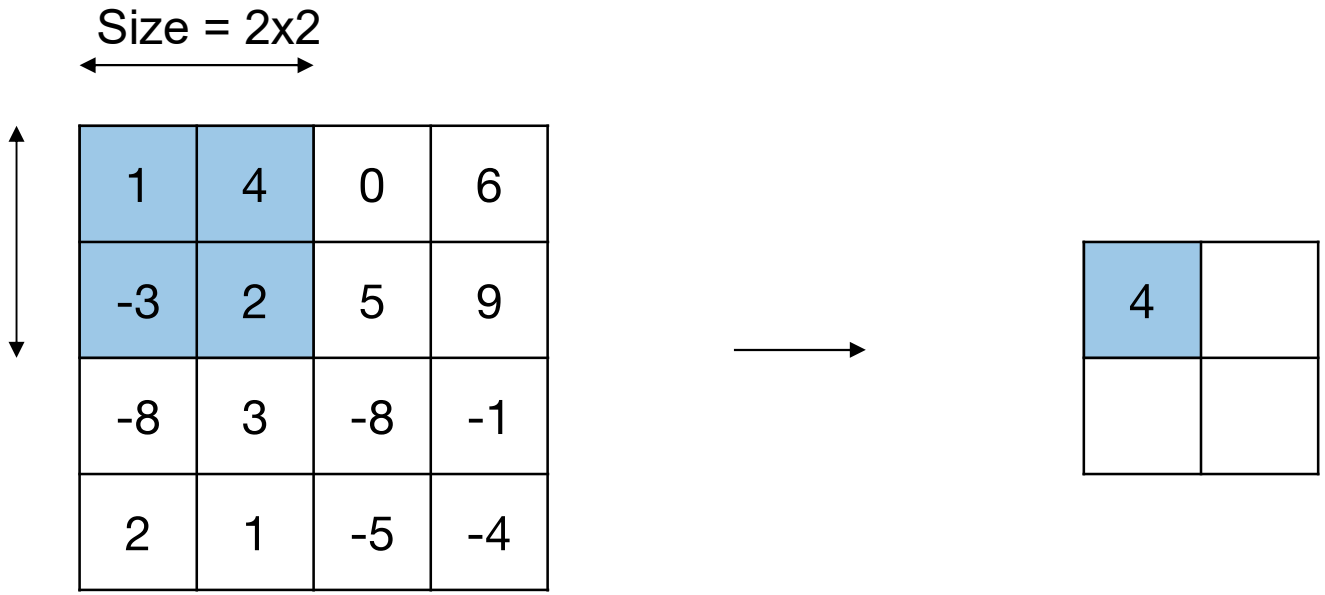
Pooling

- reduces the spacial output dimensions
- operates over each activation map idependently



Pooling

Max Pool



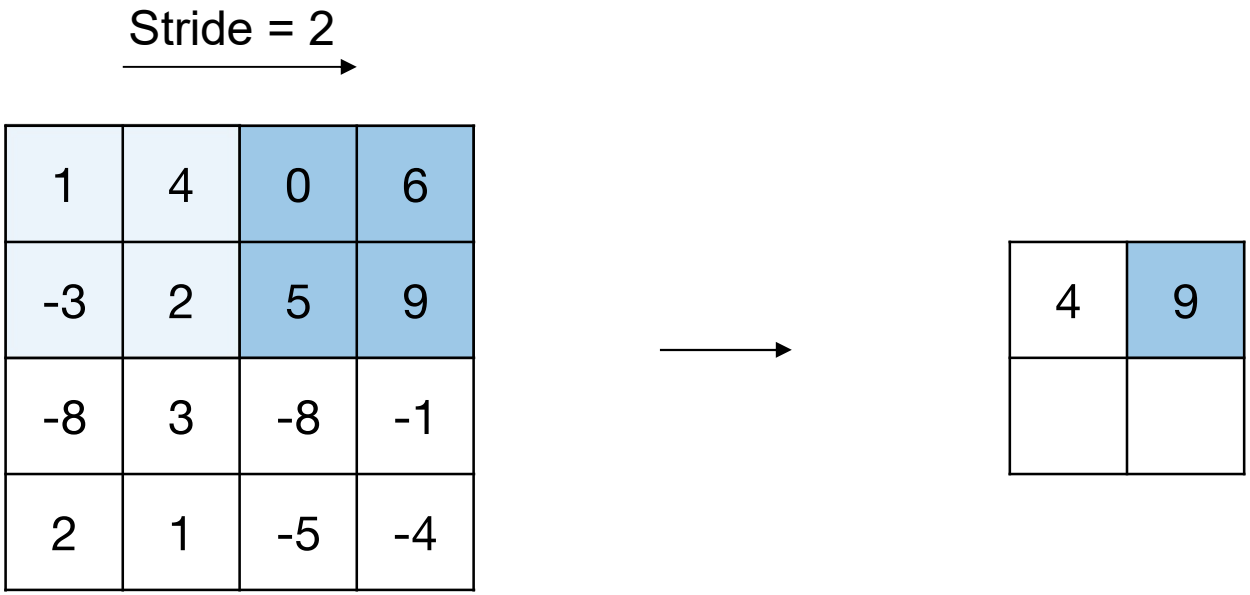
$$L^2_{i,j} = \max(L^1_{i,j}, L^1_{i-1,j}, L^1_{i,j-1}, L^1_{i-1,j-1})$$

Additional Slides

Pooling is a way to reduce dimensions in a neural network. There a different types of pooling, like max pooling or average pooling. Usual pooling layer have a size of 2x2 and stride or stepsize of 2. This means that for a max pool layer the max values of a 2x2 area a propagted to the next layer. But there is no overlap between those areas, resulting in output dimensions half the size. Also the pooling operation usually works in slices and not on the three dimensional part of the input cube.

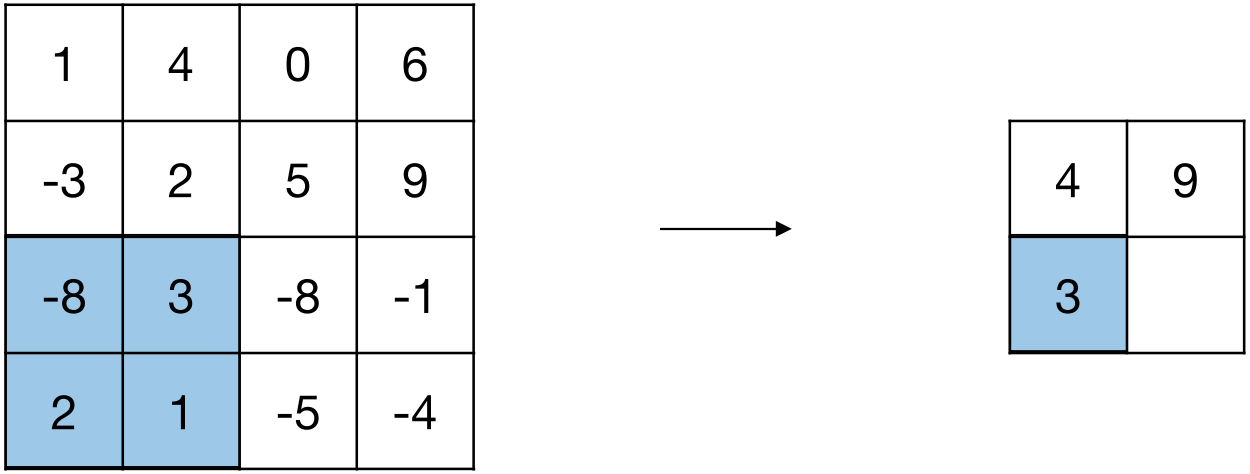
Pooling

Max Pool



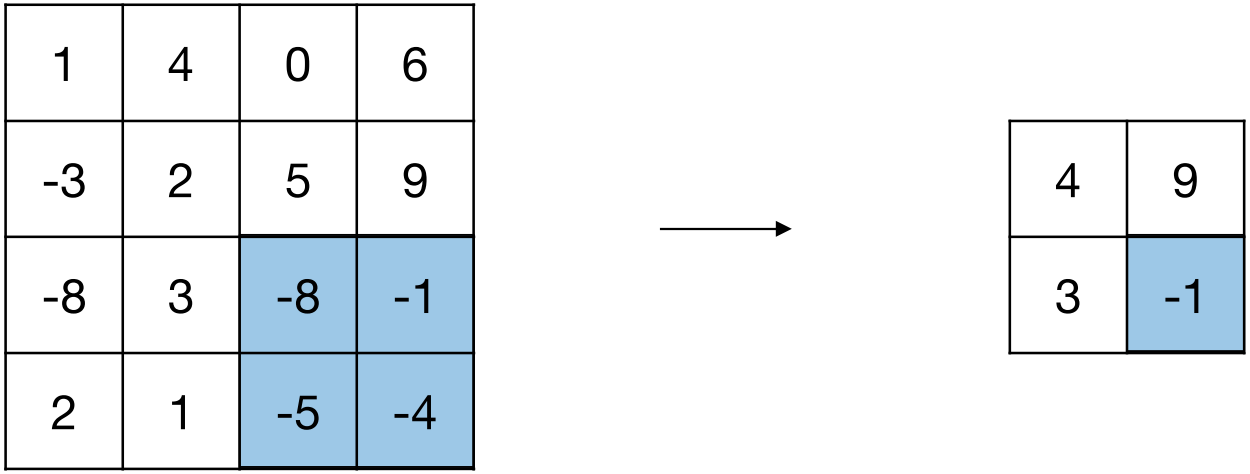
Pooling

Max Pool



Pooling

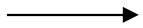
Max Pool



Pooling

Max Pool

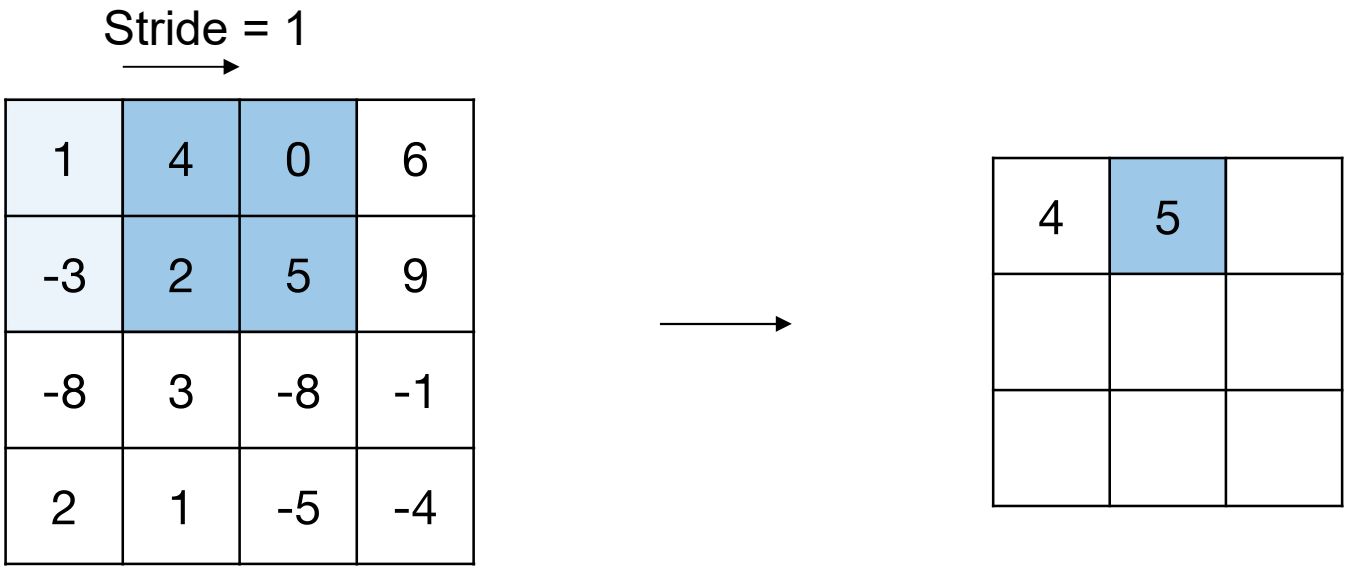
1	4	0	6
-3	2	5	9
-8	3	-8	-1
2	1	-5	-4



4		

Pooling

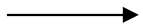
Max Pool



Pooling

Max Pool

1	4	0	6
-3	2	5	9
-8	3	-8	-1
2	1	-5	-4

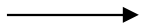


4	5	9
2	5	9
3	3	-1

Pooling

Average Pool

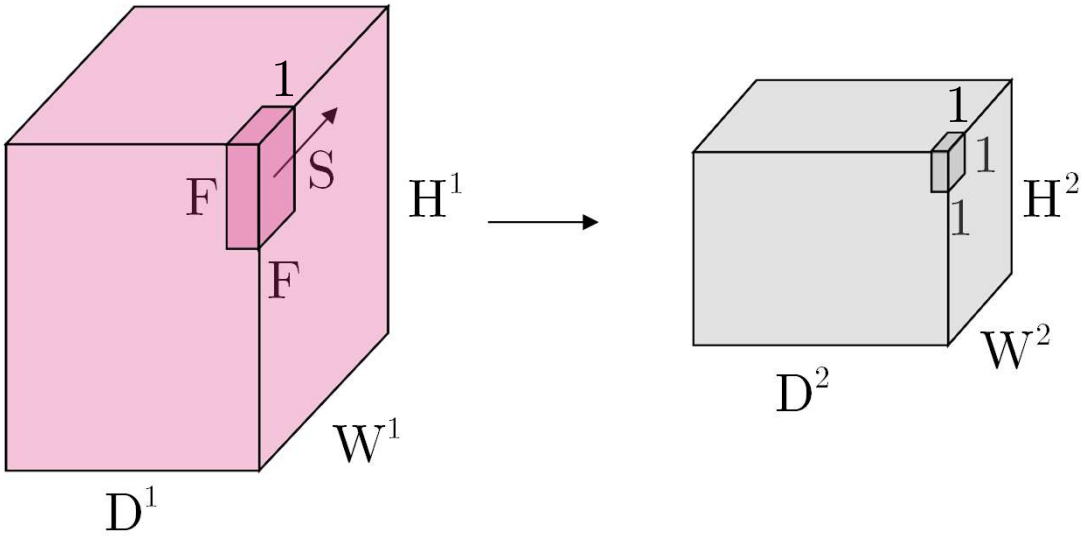
1	4	0	6
-3	2	5	9
-8	3	-8	-1
2	1	-5	-4



1	5
-0.5	-4.5

$$L^2_{i,j} = \frac{1}{4} \sum_{x=0}^{1} \sum_{y=0}^{1} L^1_{2i-x,2j-y}$$

Pooling
Dimensions



$$L^1 \in K^{D^1 \times W^1 \times H^1} \rightarrow L^2 \in K^{D^2 \times W^2 \times H^2}$$

New Dimensions:

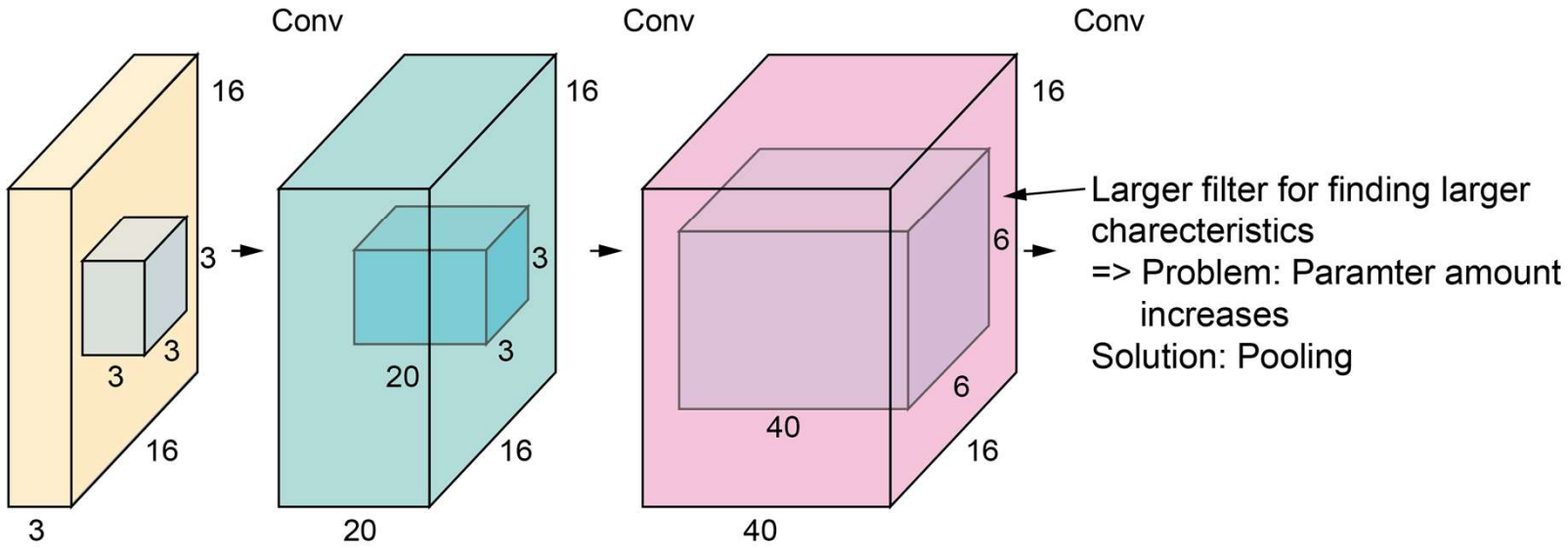
$$W^2 = \frac{W^1 - F}{S} + 1$$

$$H^2 = \frac{H^1 - F}{S} + 1$$

$$D^2 = D^1$$

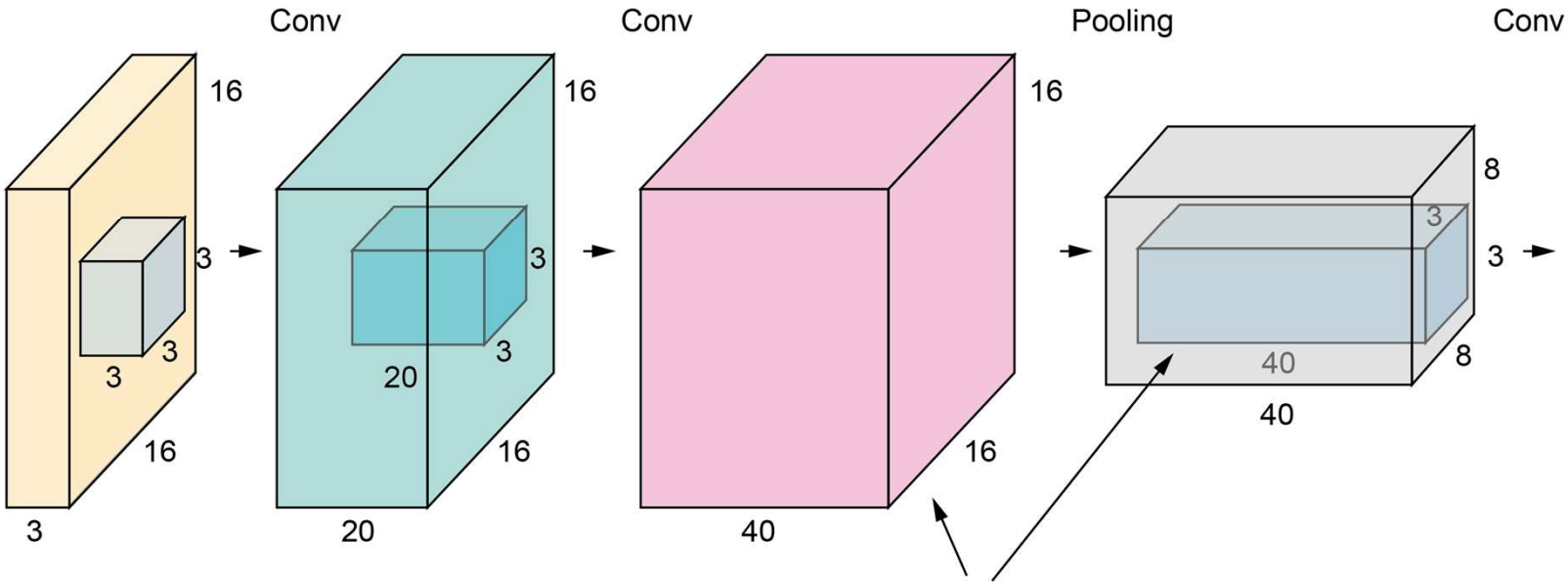
Pooling

With Convolution layer



Pooling

With Convolution layer



With the use of Pooling layers larger charisitics can be found with smaller filters. Reducing the parameter amount in this example to 25 %

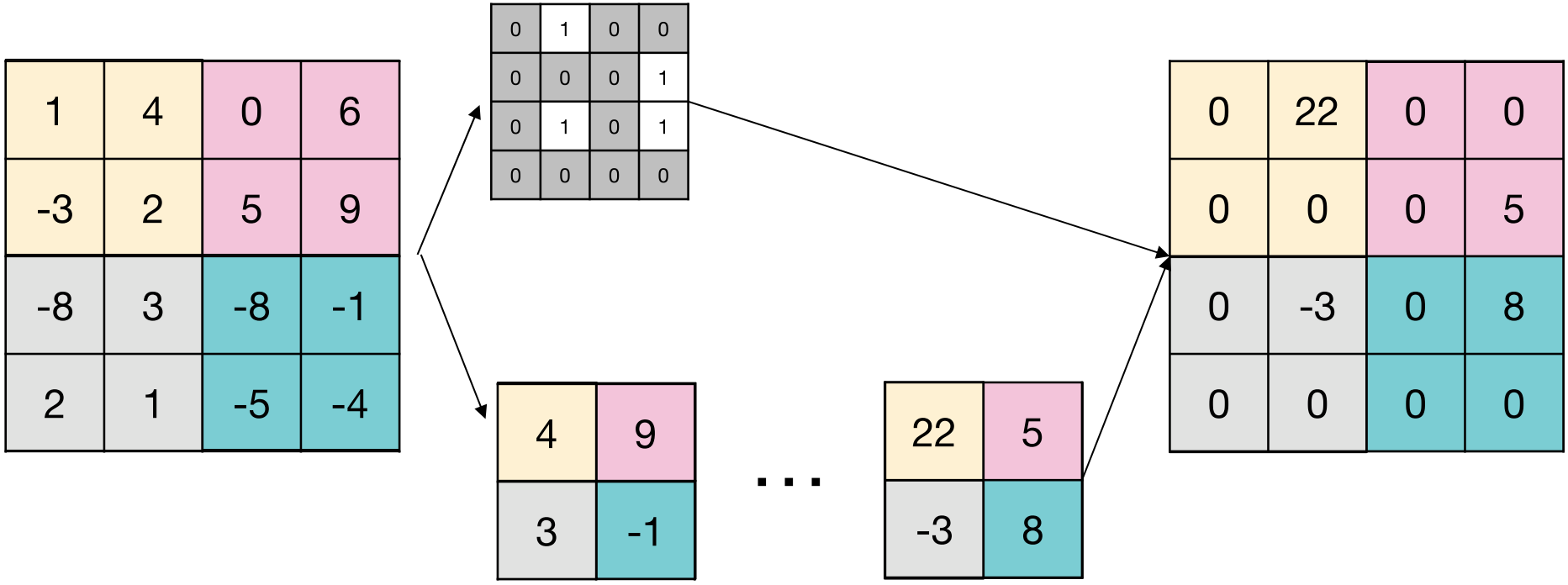
Additional Slides

But why are pooling layer used?

The problem lies in the filter sizes of convolutional layers. The maximum feature size which can be detected in a convolutional layer is determined by the size of the filter. Lets say a tire occupies 50 x 50 pixel in an input image. If the filter is only 3x3 pixels, it is not possible to detect the tire as a feature. The solution is to either increase the filter size, or to decrease the input image dimensions. Since increasing the filter size results in a higher computational load, pooling layers are used. Features detected in previous layers are not lost, since the highest activation is passed along.

Unpooling

Max Unpool

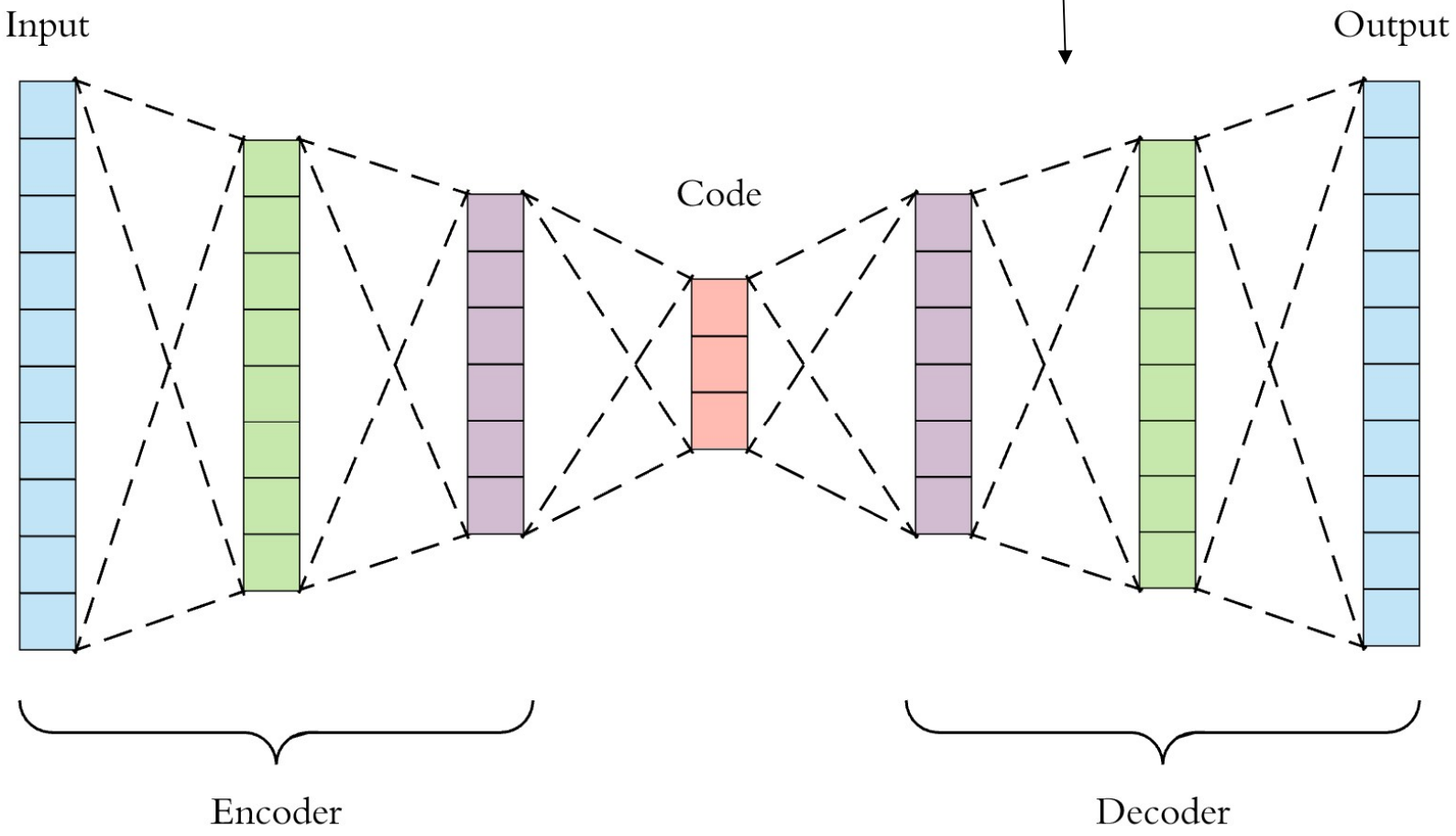


Additional Slides

There are also options to reverse the unpool operation, for example to enlarge an image after pooling. This is can be done by saving the positions of the correspoding high values during the pooling and than using these positions in an later unpool layer.

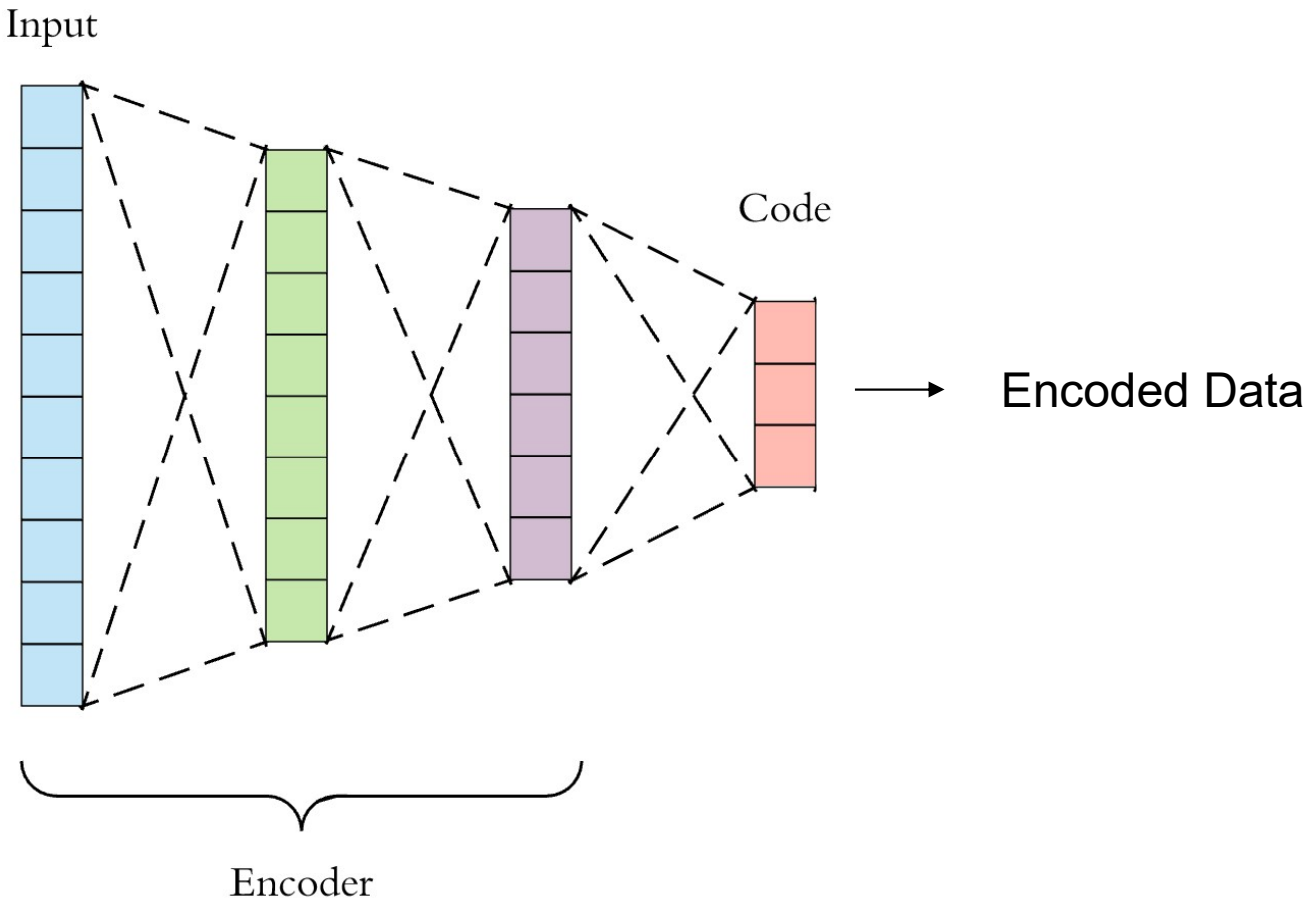
Dimension Reduction

Fully Connected - Autoencoder



Dimension Reduction

Fully Connected - Autoencoder



Additional Slides

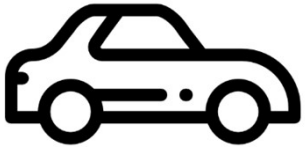
An interesting concept is the auto-encoder. Here the neural network is trained to reproduce the same output data as the input data. Since the data density is reduced with each layer in the encoder part the network has to reproduce the same data in the decoder part as the input data from less information. Therefore all important information is encoded in the layer with the smallest neuron count. The encoder and decoder part of the network can than be separeted and used idependently. This can be usefull for data compression, data filtering ….

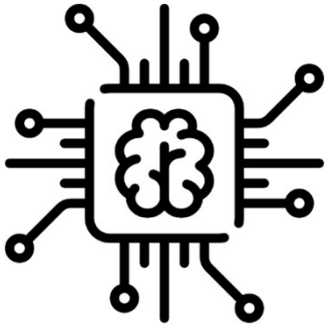
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

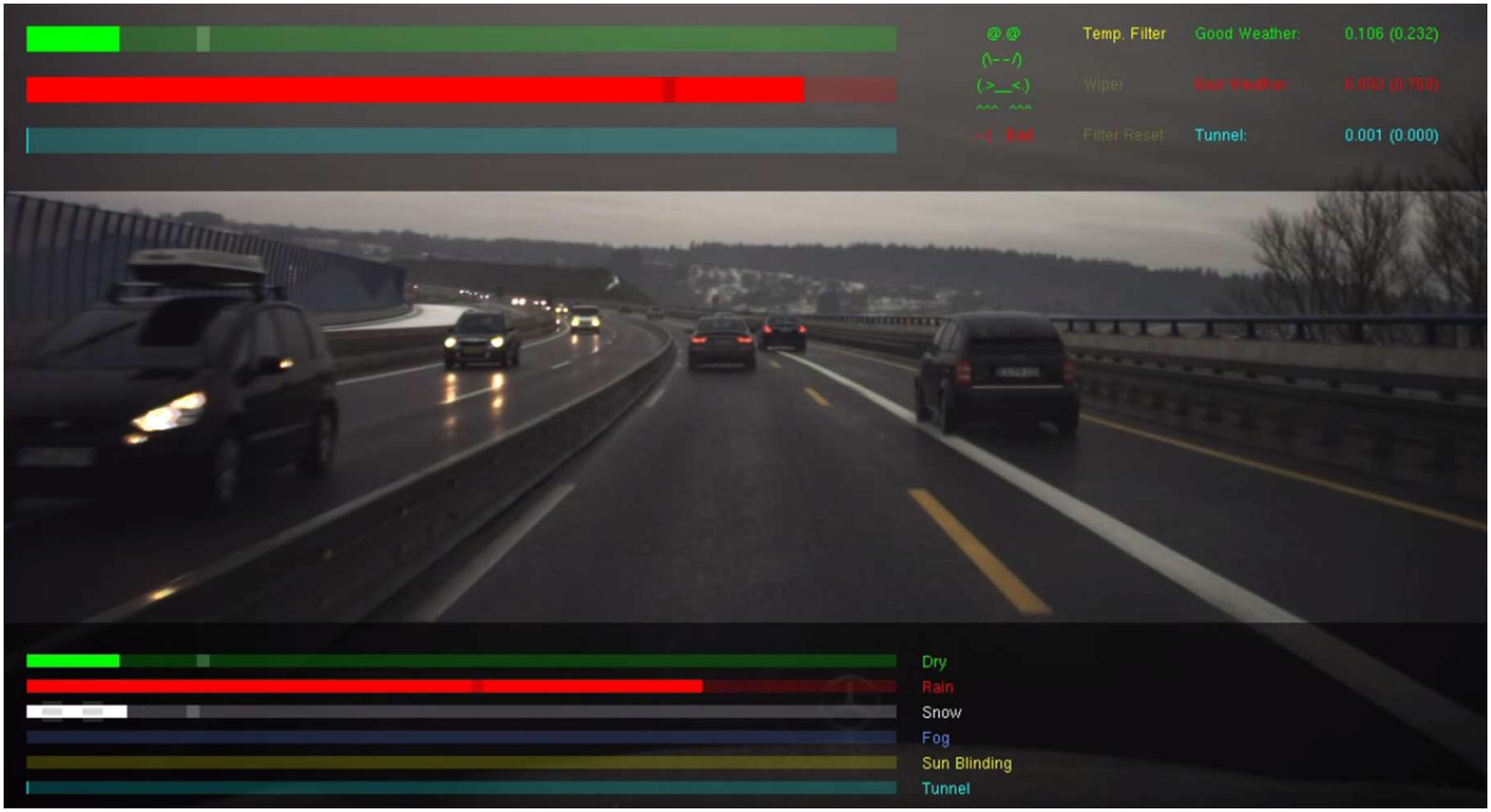
1. Chapter: Convolutional Neural Networks
    1.1 Motivation
    1.2 Introduction
    1.3 Dimensions, Padding, Stride
2. Chapter: Pooling
3. Chapter: CNN in Action
4. Chapter: Advanced Topics
    2.1 Optimizer
    2.2 Data Formatting





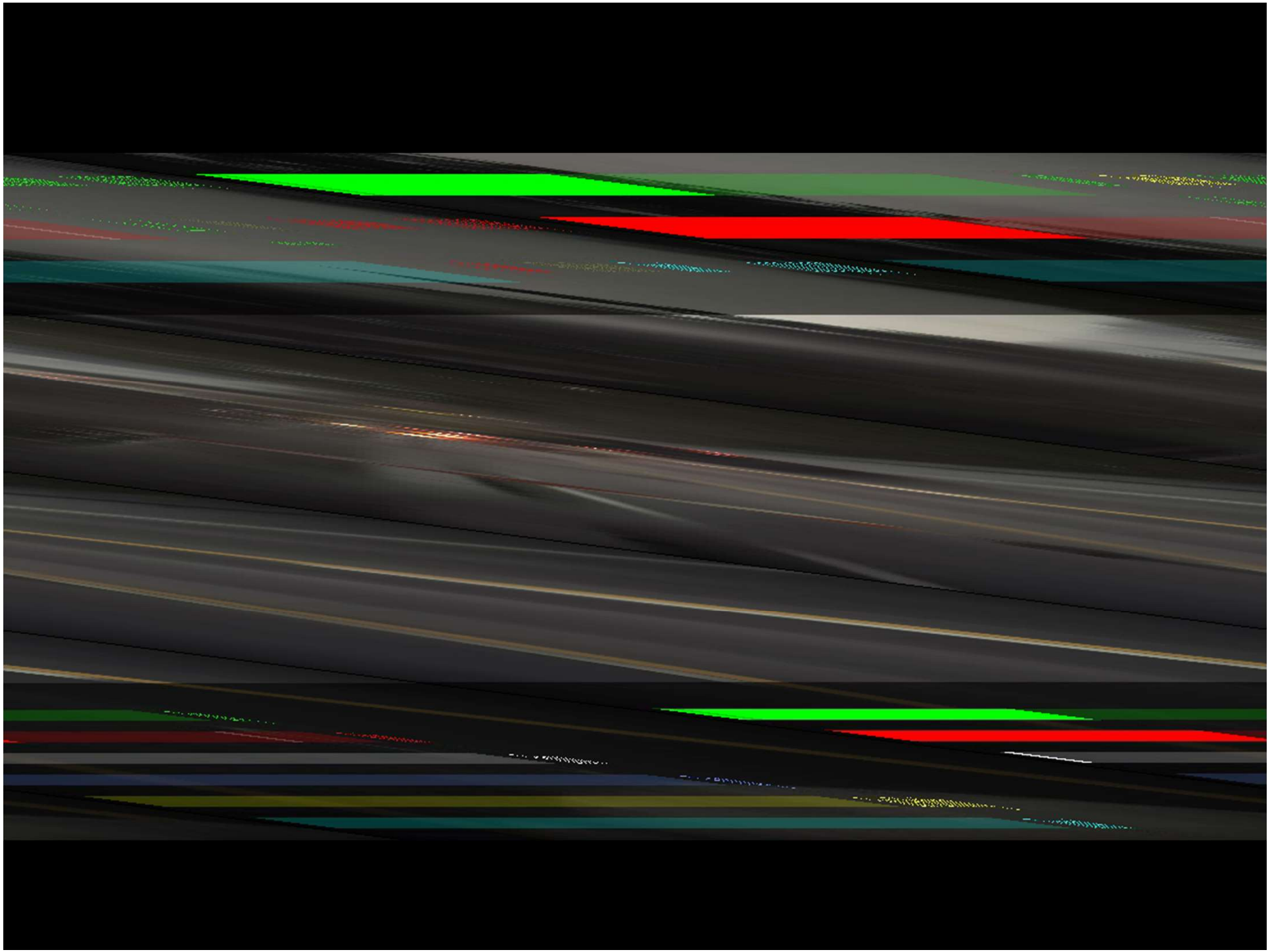
Daimler: Weather Prediction

Dr. Uwe Franke et al.

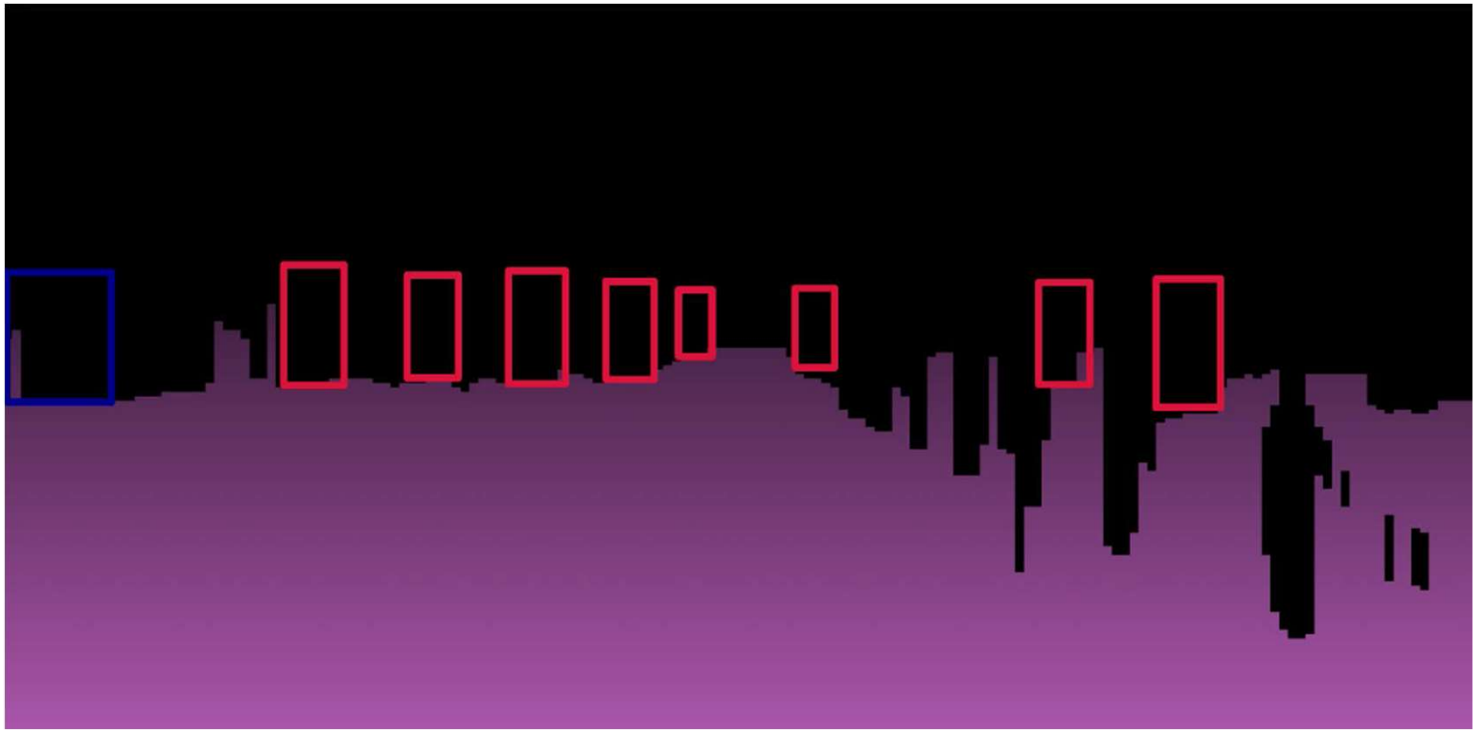


Additional Slides

The video shows the results of AlexNet on classifing the current weather in video data.



Bertha Benz 1888 -> 2013 autonom



Bertha Benz Fahrt

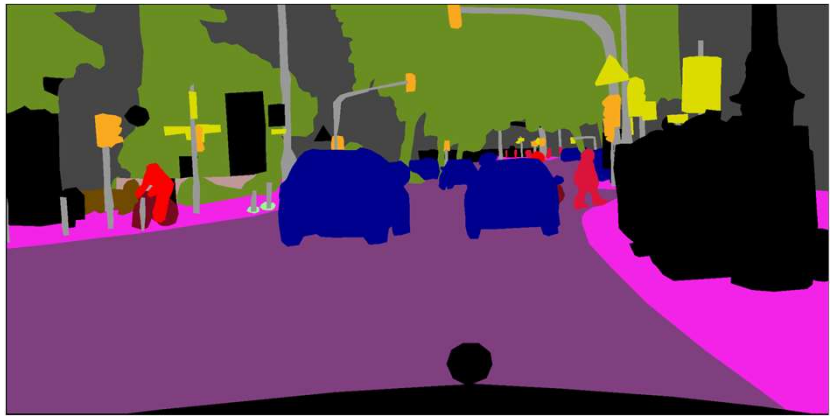
Beispielszene

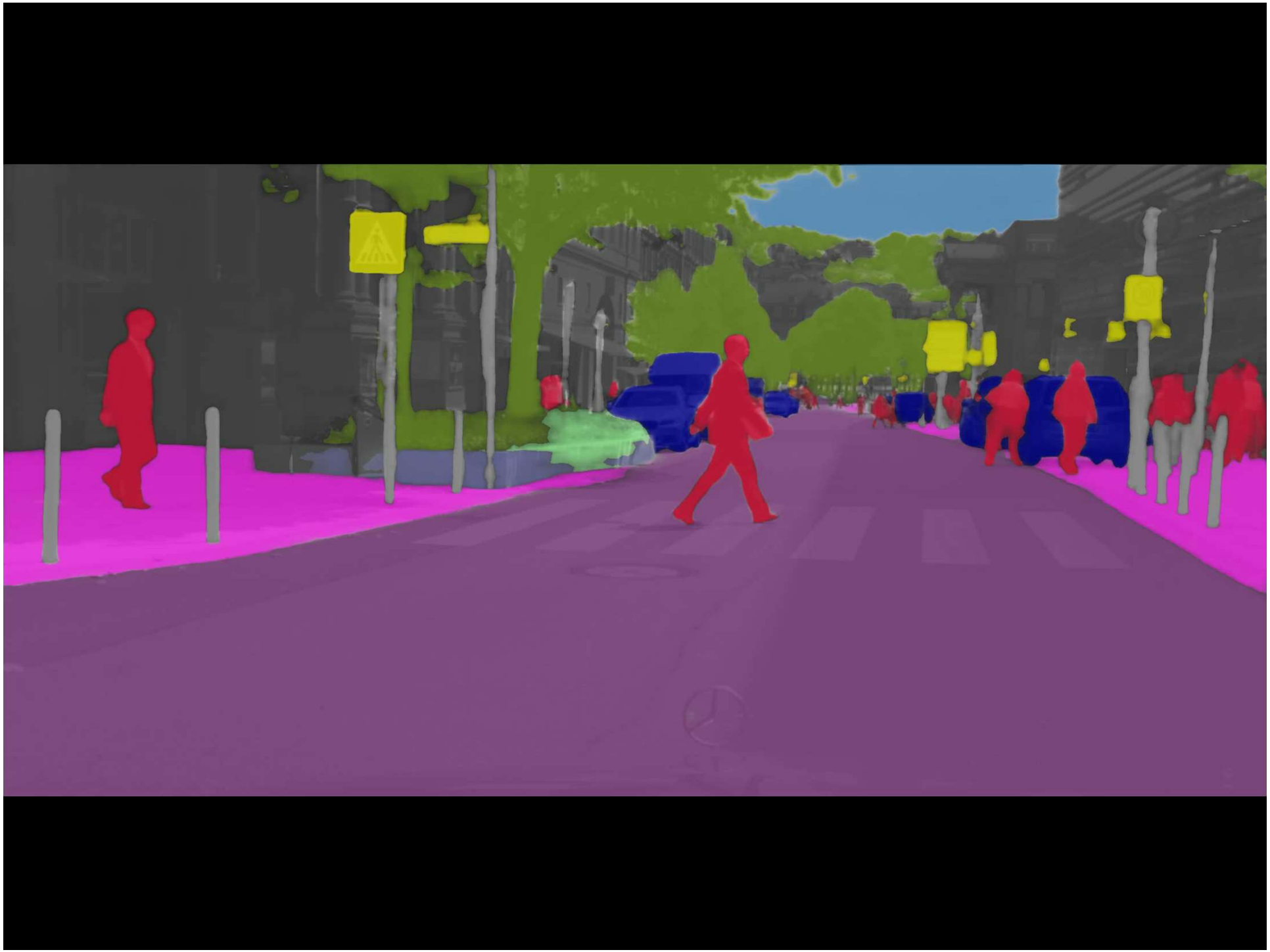
Beispielszene

Daimler: Cityscapes Dataset

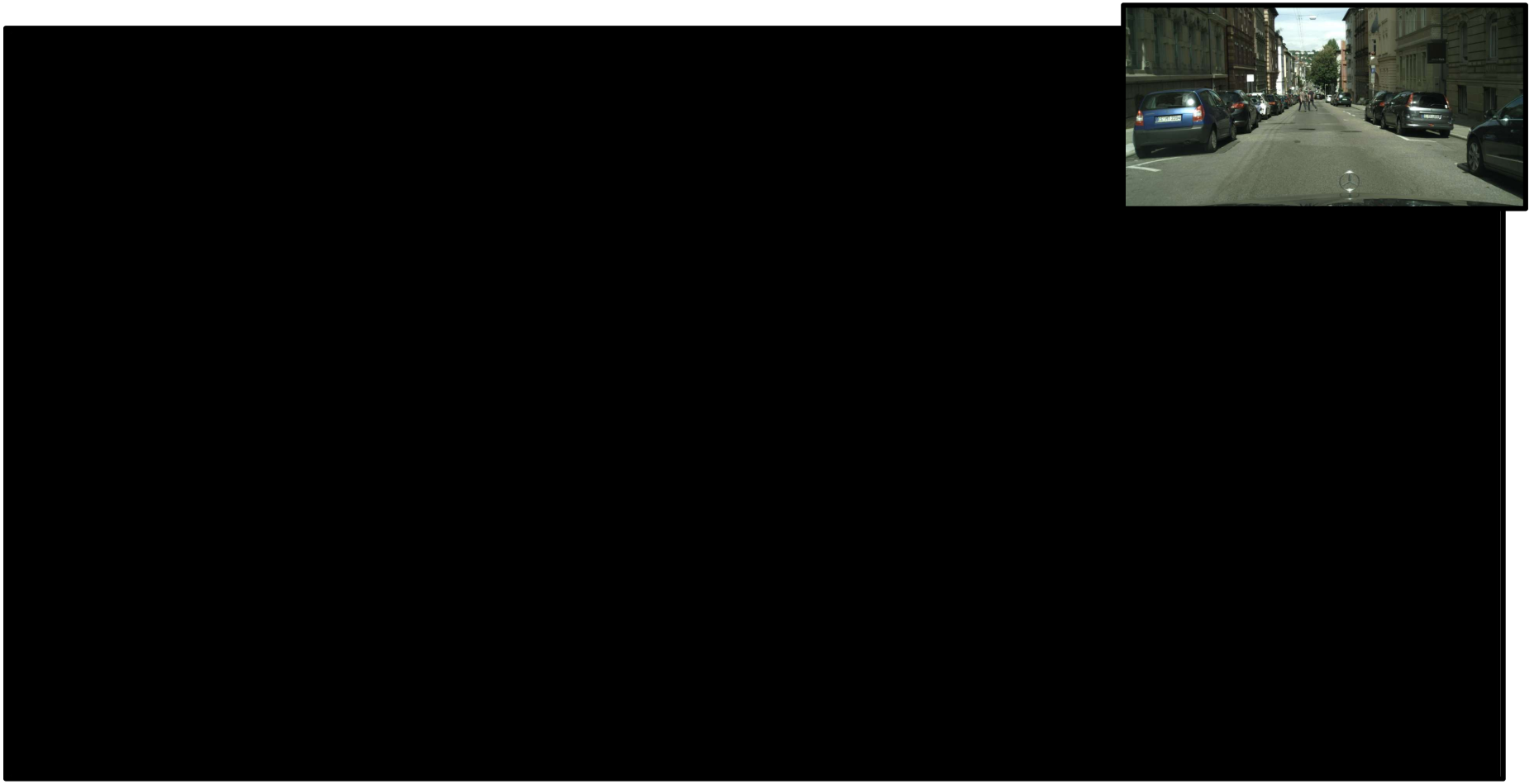
x

$\hat{y}$



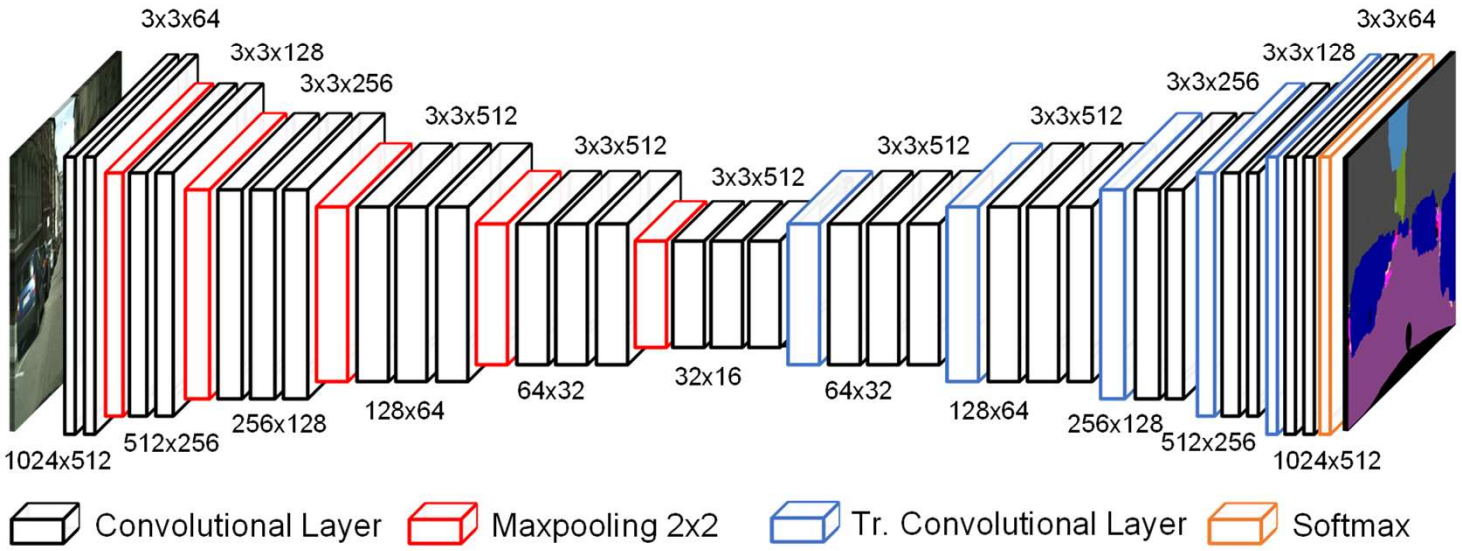


Cityscapes FCN: Learning Process



CNN-Struktur

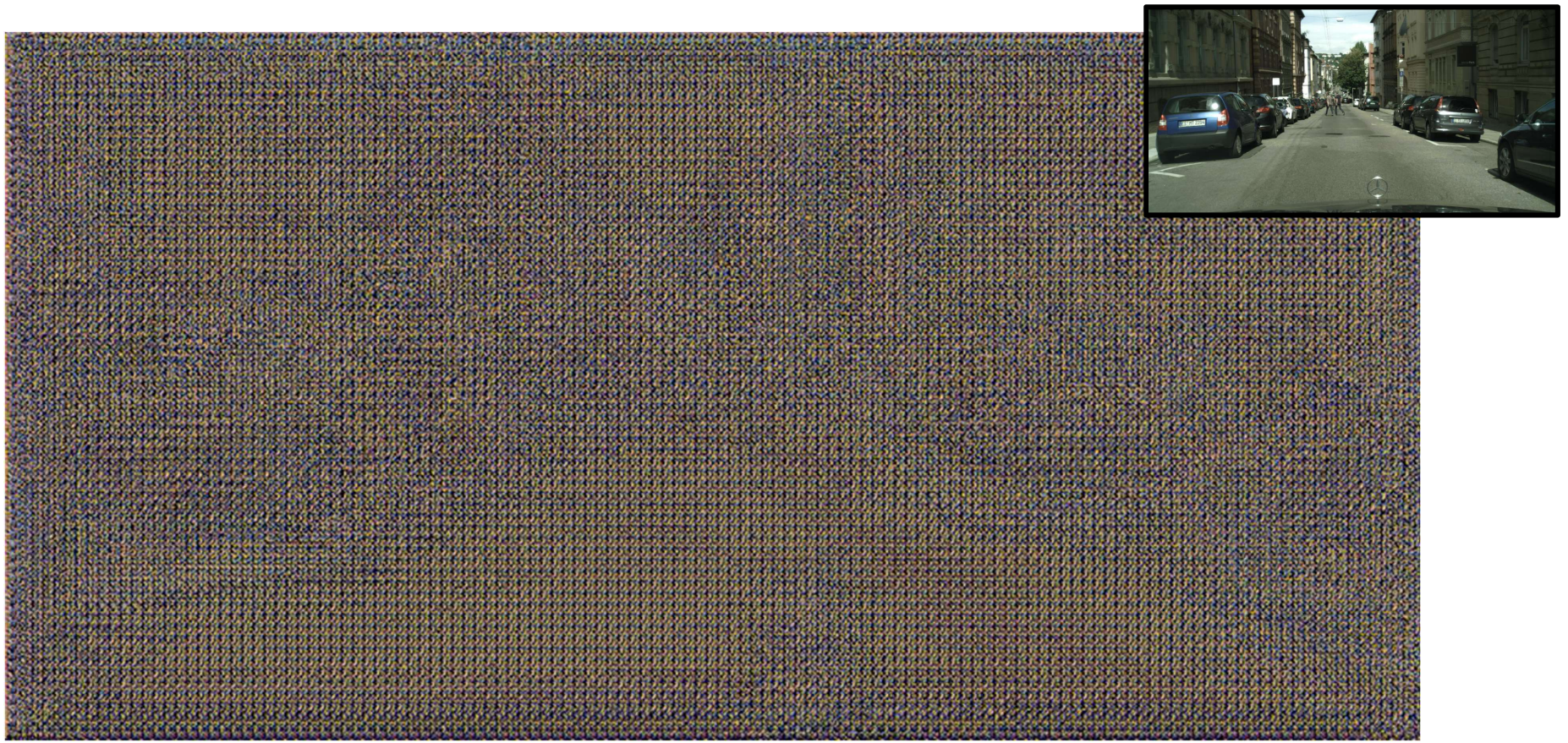
VGG16 Mirrored like code (VGG) (feature_drop_before_cvtr)



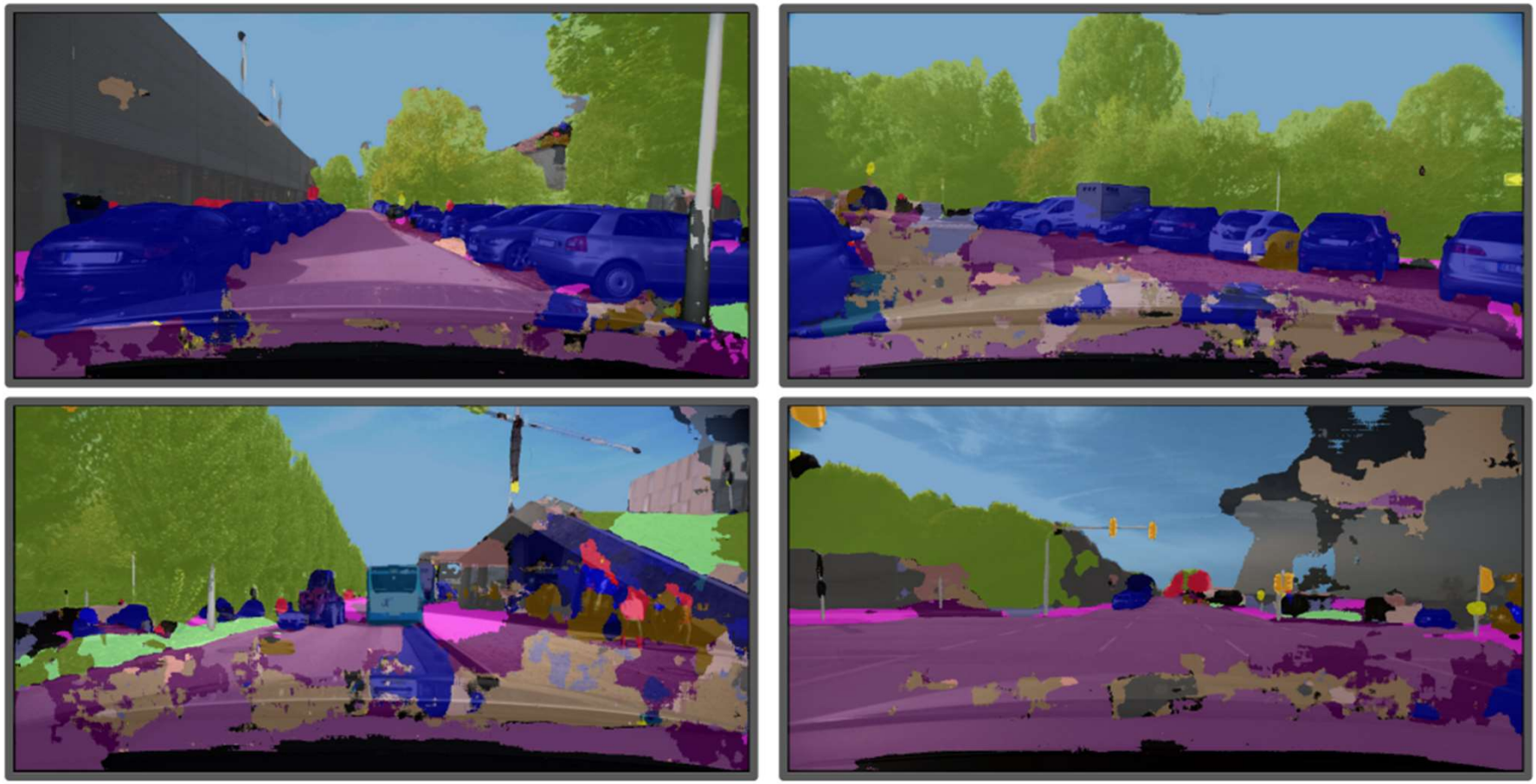
Additional Slides

In this image you can see a typical convolutional neural network, the VGG-Net. From the image representation the network architecture can be directly derived. The interesting thing is, that the neural network can produce an accurate semantic segmentation of an 1024x512 images based on an 32x16 image in the center of the network .

Cityscapes CNN: Learning Process



Pylon Camera Results

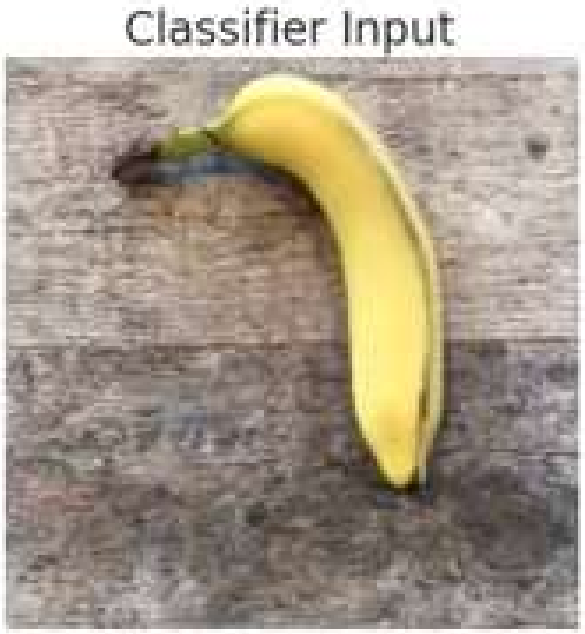


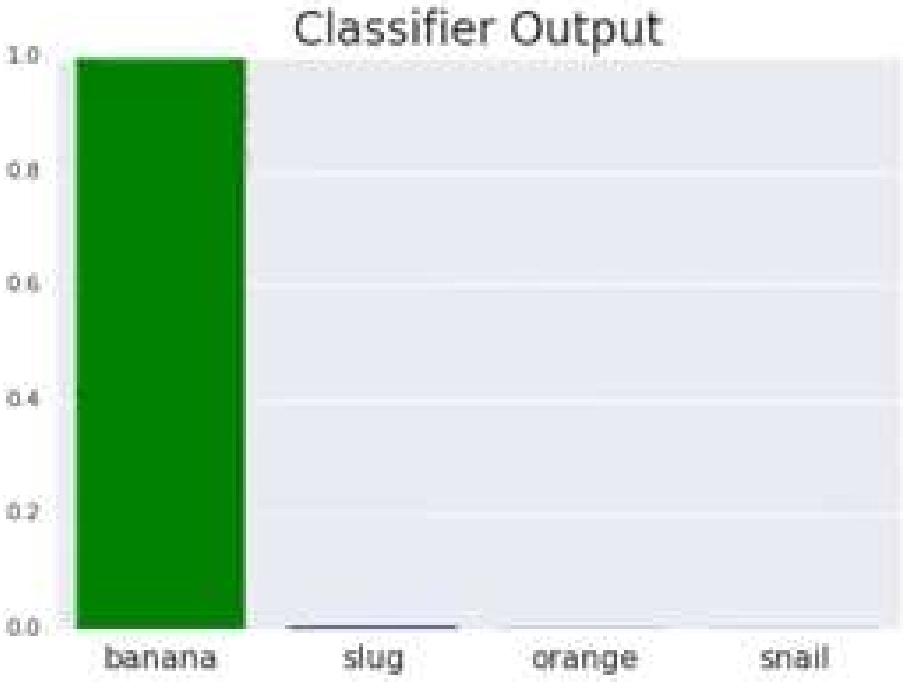
Galaxy S7 Results

Additional Slides

If the training data is recorded with one camera, but the neural network is running on a different camera, the results will be probably not useable. Therefore the dataset should be as diverse as possible in order for the network to generalize the information.

Video: Adversial Patch





Additional Slides

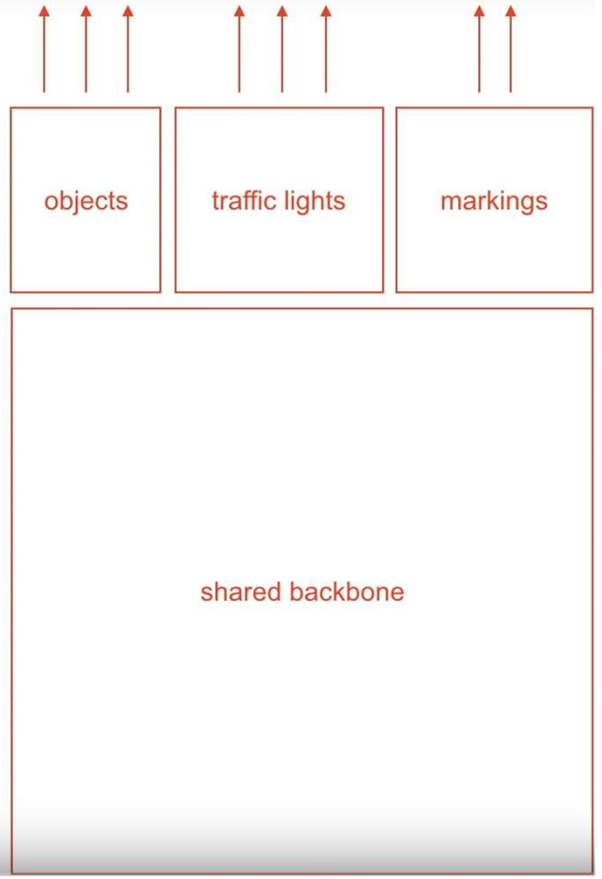
Neural networks can be hacked with so called „Adversial Patches“. This is still an ongoing research topic

https://www.youtube.com/watch?v=i1sp4X57TL4

Tesla – 6.11.2019

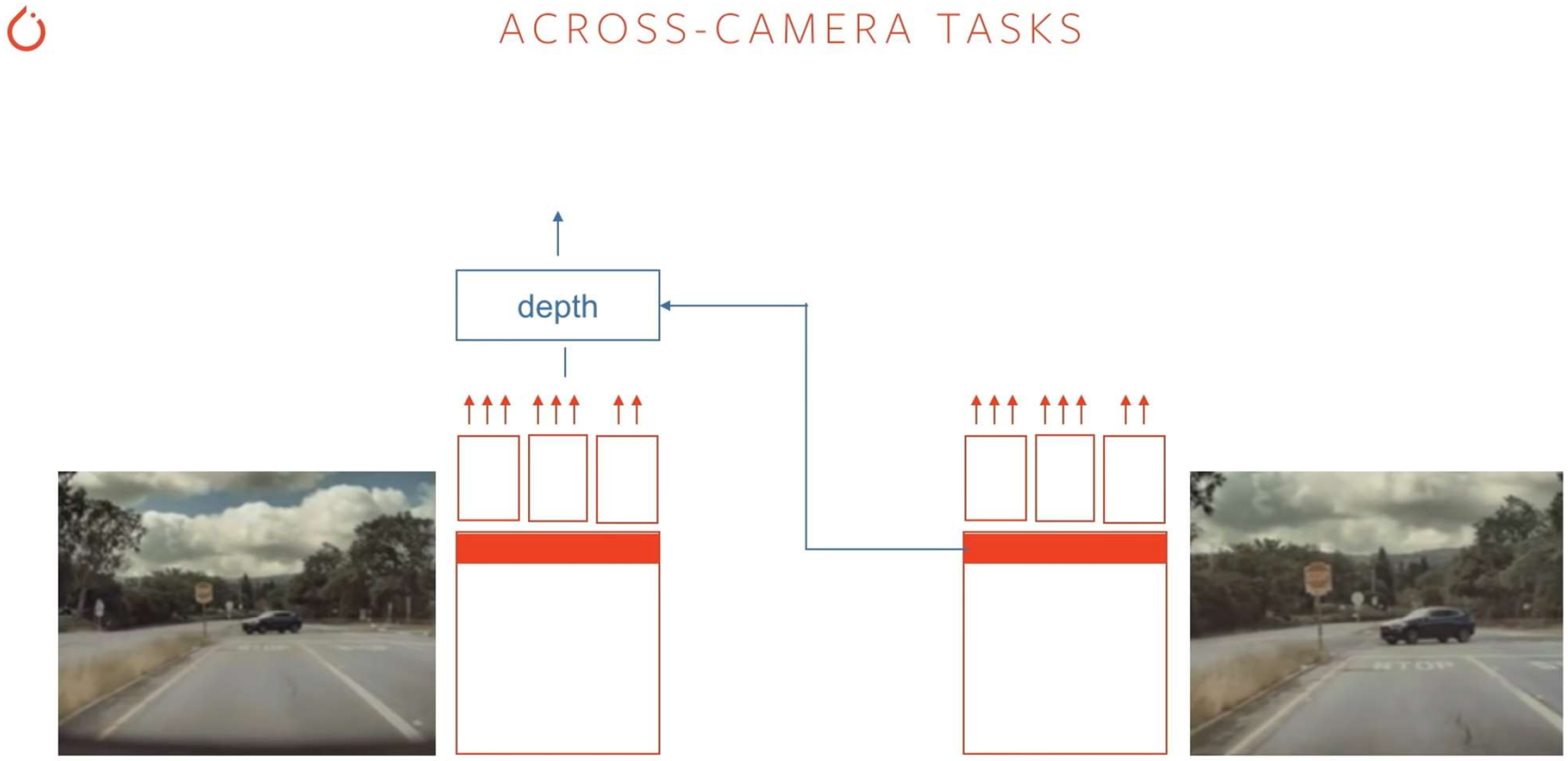
SINGLE VIEW TASKS

- (3, 960, 1280) input images
- “ResNet-50 like” dilated backbones
- FPN / DeepLabV3 / UNet -like heads
- ~15 tasks => “prototypes framework”



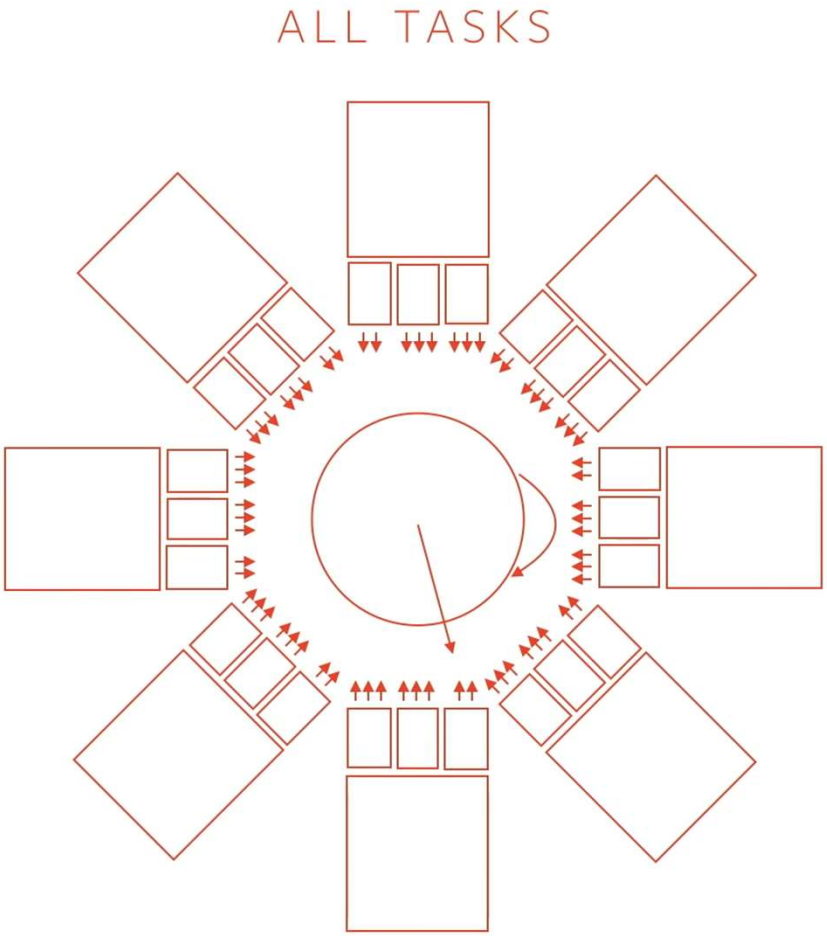
https://www.youtube.com/watch?v=oBkIltKXtDE

Tesla



https://www.youtube.com/watch?v=oBkIltKXtDE

Tesla

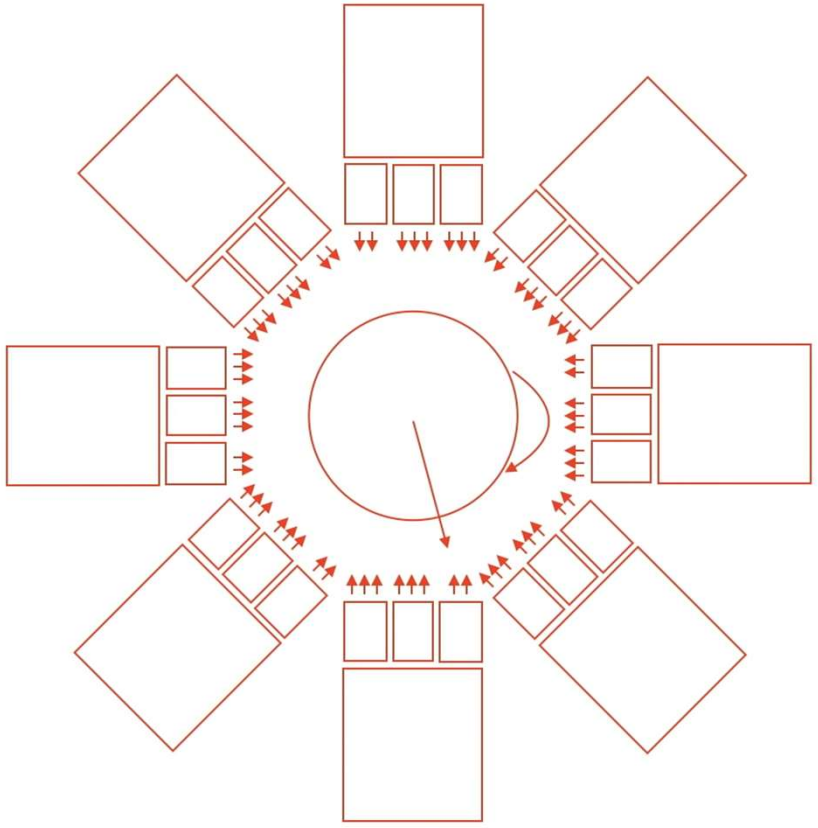


https://www.youtube.com/watch?v=oBkIltKXtDE

Tesla

e.g.:
8 cameras, 16 time steps, batch size 32:
8 x 16 x 32 = **4096 images**
in a single forward pass

Requires a combination of data parallel and model parallel training



https://www.youtube.com/watch?v=oBkIltKXtDE

Tesla

A FULL BUILD

(Compiling the datasets into Software 2.0 code)

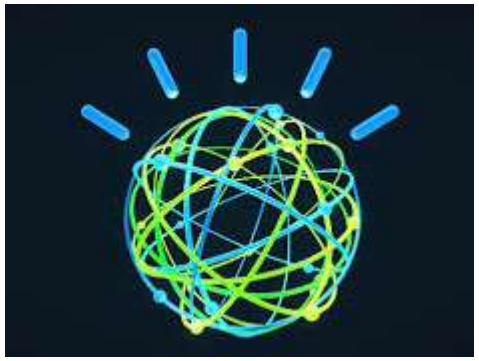
48
networks

1,000
distinct predictions
(#output tensors)

70,000
GPU hours

https://www.youtube.com/watch?v=oBkIltKXtDE

Mensch vs. Maschine



	Mensch	IBM Watson, 2012
Geschwindigkeit	10.000.000 Mia Op/s	80.000 Mia. Op/s
Energie	20 Watt	200 Kilo Watt
Effizienz	500x10^6 Mega Flops/Watt	400 Mega Flops/Watt

=> Mehr als 50% des Hirns beschäftigt sich mit Sehen

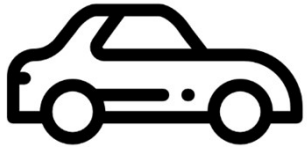
Source: Yu Wang, Tsinghua University, Feb 2016; https://www.quora.com/How-much-of-the-brain-is-involved-with-vision

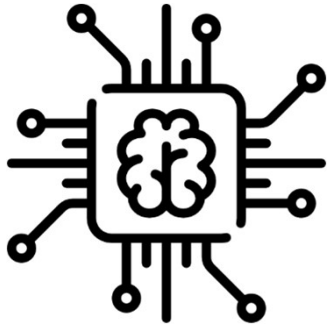
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

Agenda

1. Chapter: Convolutional Neural Networks
    - 1.1 Motivation
    - 1.2 Introduction
    - 1.3 Dimensions, Padding, Stride
2. Chapter: Pooling
3. Chapter: CNN in Action
4. Chapter: Advanced Topics
    - 2.1 Optimizer
    - 2.2 Data Formatting





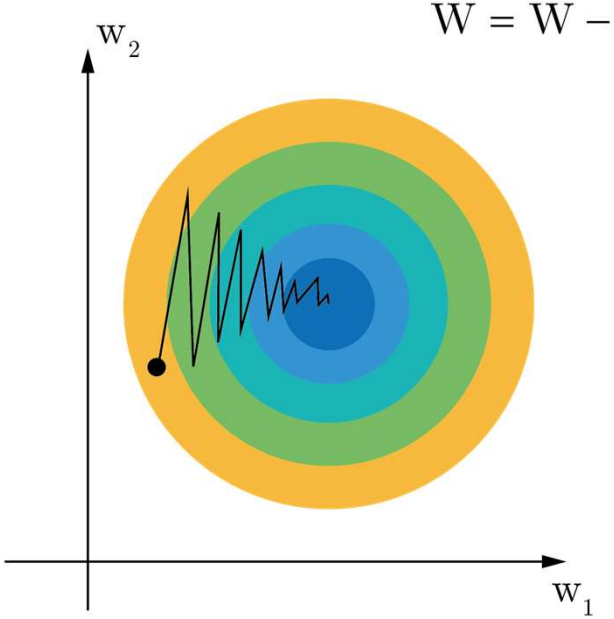
Optimizer

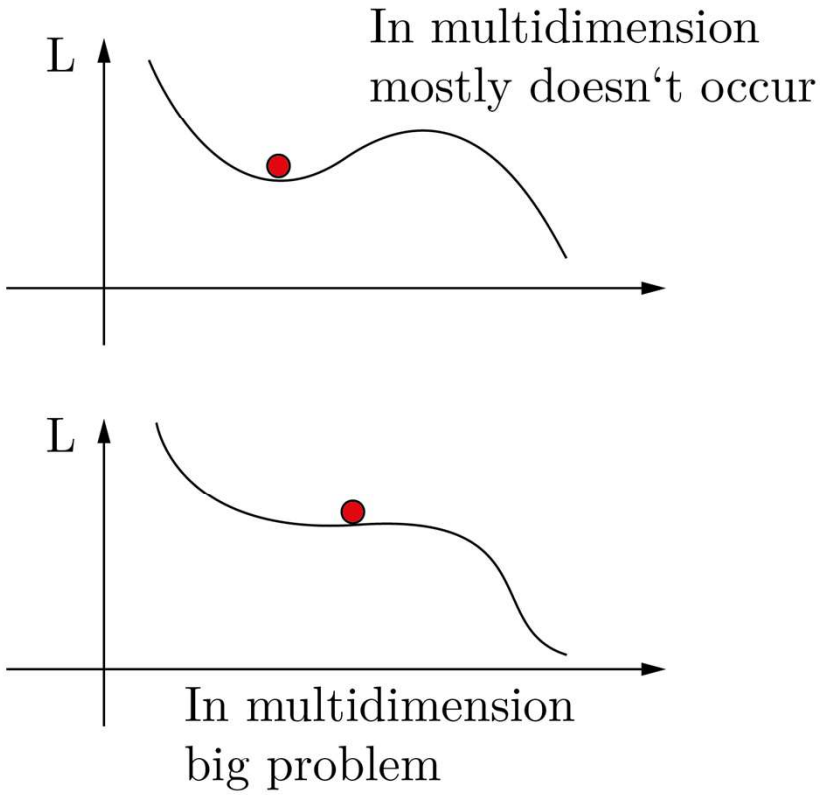
Stochastic Gradient Decent

Problems with SGD:

- Loss might change quickly in one direction, but slow in the other!
- local minima and sattle points

$$W = W - \alpha \nabla_W L$$



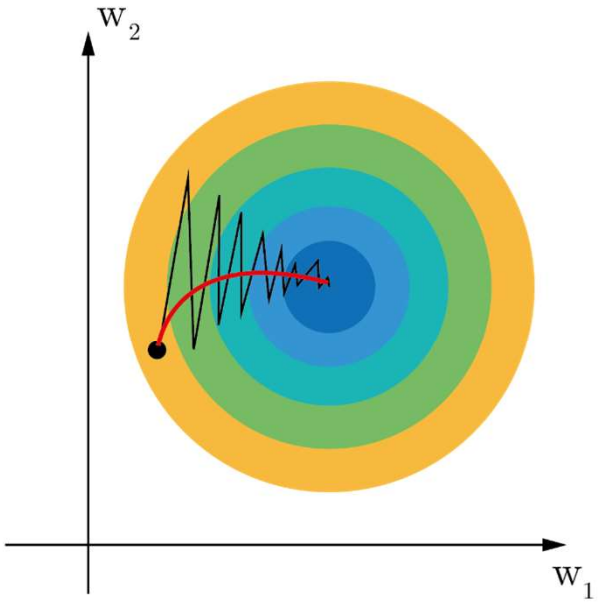


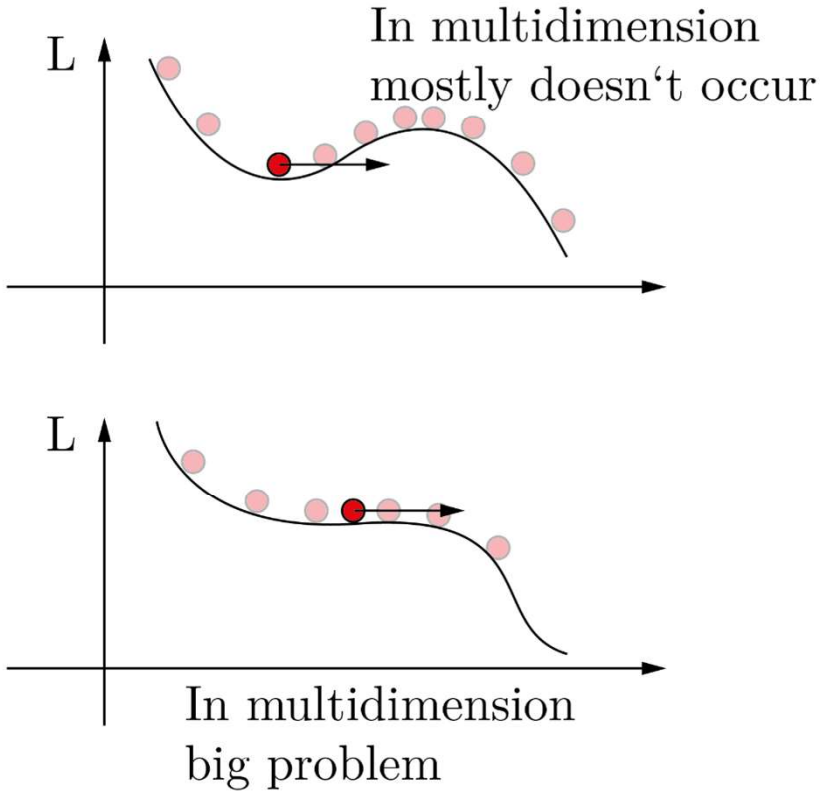
Optimizer

Stochastic Gradient Decent + Momentum

$$W_{t+1} = W_t - \alpha v_t \qquad v : \text{velocity}$$

$$v_{t+1} = \rho v_t + \nabla_W L_t \qquad \rho : \text{friction}$$





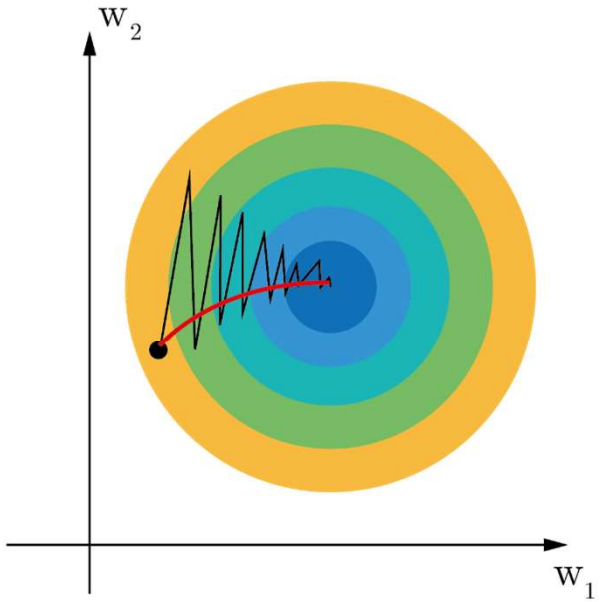
Additional Slides

Stochastic gradient decent might get stuck in a local minima or sattlepoint during training. To overcome this problem a momentum can be added to the weight update step. The momentum contains the current weight-update $\nabla_W L_t$ as well as previous scaled weight-updates ρv_t . If the current weight-update is zero, the previous weight-updates will still change the current weights, reducing the likelihood to get stuck during training

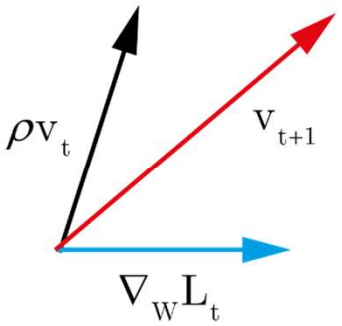
Optimizer

Nesterov+ Momentum

$$W_{t+1} = W_t - \alpha v_t$$
$$v_{t+1} = \rho v_t + \nabla_W L_t(x + v_t)$$

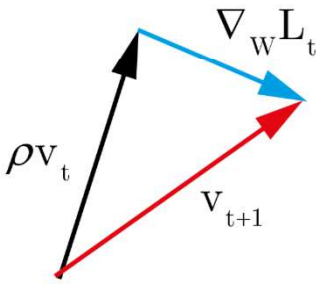


SGD + Momentum



$$v_{t+1} = \rho v_t + \nabla_W L_t(x)$$

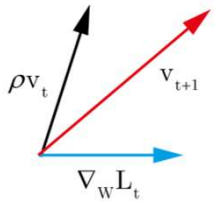
Nesterov Momentum



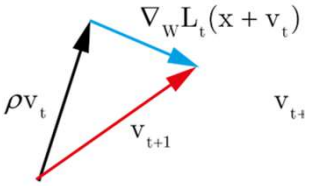
$$v_{t+1} = \rho v_t + \nabla_W L_t(x + v_t)$$

Additional Slides

If you look at the momentum principle, the direction of weight change is given by the previous weight-updates and the current weight-update. For two weights this can be represented as a vector addition, where the new weight-update is the sum of both vectors. .



Another idea is the Nesterov Momentum: In the Nesterov Momentum the weights are first changed according to the previous weight-updates, than the current weight-update is calculated.



Optimizer

AdaGrad and RMSProp

$$\text{AdaGrad}:$$

$$\Phi_{t+1} = \Phi_t + \nabla_W L_t \cdot \nabla_W L_t$$

$$W_{t+1} = W_t - \frac{\alpha}{\sqrt{\Phi_t + \varepsilon}} \cdot \nabla_W L_t$$

$$\text{AdaDelta / RMSProp}:$$

$$\Phi_{t+1} = \delta\Phi_t + (1-\delta)\nabla_W L_t \cdot \nabla_W L_t$$

$$W_{t+1} = W_t - \frac{\alpha}{\sqrt{\Phi_t + \varepsilon}} \cdot \nabla_W L_t$$

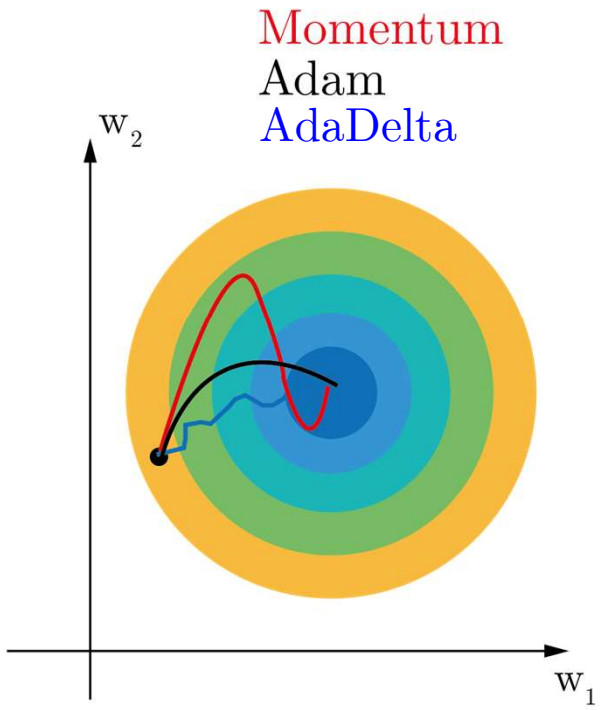
$$\delta : \text{decay rate}$$

Additional Slides

The idea of AdaGrad is to scale gradients in such a way, that weight updates happen equal and more consistent along each dimension. Therefore the current-weight update is scaled by the previous weight-updates. The result is that large weight-updates in one dimension get scaled by a smaller factor, resulting in an smaller update. And small weigt-updates in the other dimension, get scaled by a larger factor, resulting in an larger update. Since the scaling factor is calculated by the sum of the weight-updates squared, the factor and therefore the weight updates get smaller with every training iterator. RMSProp solves this problem by adding a decay rate to the old factors.

Optimizer

Adam



Combining AdaGrad and Momentum

$$\Upsilon_{t+1} = \delta_1 \Upsilon_t + (1-\delta_1)\nabla_W L$$
$$\Phi_{t+1} = \delta_2 \Phi_t + (1-\delta_2)\nabla_W L_t * \nabla_W L_t$$
$$W_{t+1} = W_t - \alpha \frac{\Upsilon_t}{\sqrt{\Phi_t + \varepsilon}}$$

Problem: What happens at the beginning?

$$\Upsilon_{t+1} = \delta_1 \Upsilon_t + (1-\delta_1)\nabla_W L$$
$$\Phi_{t+1} = \delta_2 \Phi_t + (1-\delta_2)\nabla_W L_t * \nabla_W L_t$$
$$\hat{\Upsilon}_{t+1} = \frac{\Upsilon_{t+1}}{1-\delta_1^t}$$
$$\hat{\Phi}_{t+1} = \frac{\Phi_{t+1}}{1-\delta_2^t}$$
$$W_{t+1} = W_t - \alpha \frac{\hat{\Upsilon}_t}{\sqrt{\hat{\Phi}_t + \varepsilon}}$$

$\delta_1 = 0.9$

$\delta_2 = 0.999$

$\alpha = 0.001$

Additional Slides

The Adam Optimizer combines the AdaGrad and Momentum idea, adding the velocity term as well as the gradient scaling factor. A Problem occurs during the first time steps of training. After the first interation the AdaGrad Term Φ_{t+1} is usually still close to zero. Since we divide the momentum term Υ_{t+1} by the AdaGrad Term, the result is large weight-update. Therefore we include a bias term which scales both factors during the first training iterations.

$$\hat{\Upsilon}_{t+1} = \frac{\Upsilon_{t+1}}{1-\delta_1^t}$$

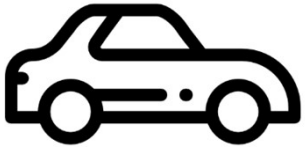
$$\hat{\Phi}_{t+1} = \frac{\Phi_{t+1}}{1-\delta_2^t}$$

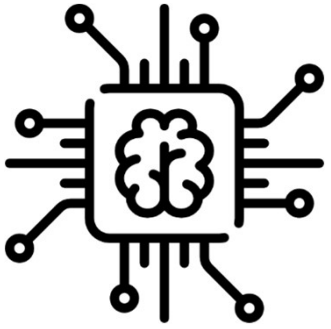
Deep Neural Networks

Johannes Betz / Prof. Dr. Markus Lienkamp / Prof. Dr. Boris Lohmann
(Jean-Michael Georg, M. Sc.)

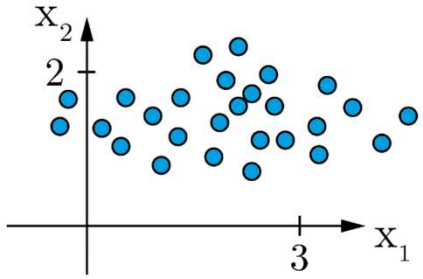
Agenda

1. Chapter: Convolutional Neural Networks
    - 1.1 Motivation
    - 1.2 Introduction
    - 1.3 Dimensions, Padding, Stride
2. Chapter: Pooling
3. Chapter: CNN in Action
4. Chapter: Advanced Topics
    - 2.1 Optimizer
    - 2.2 Data Formatting

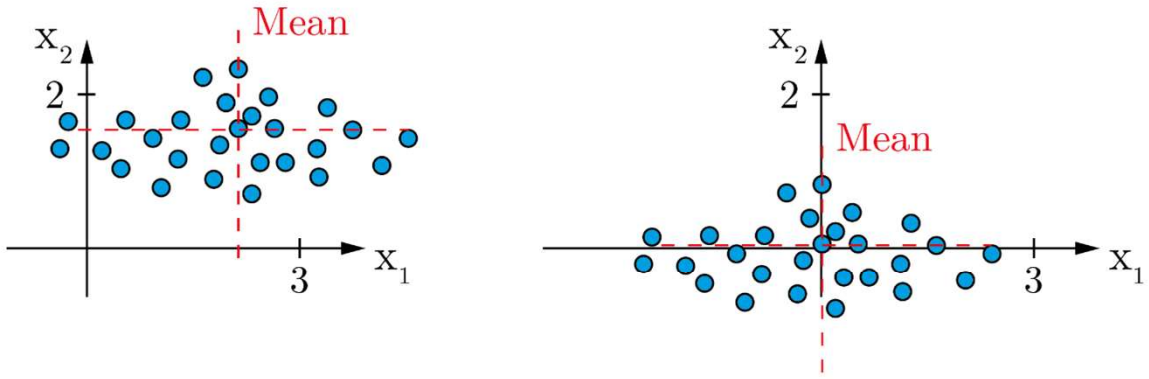




Data Distribution of the Input Data



Data Distribution of the Input Data

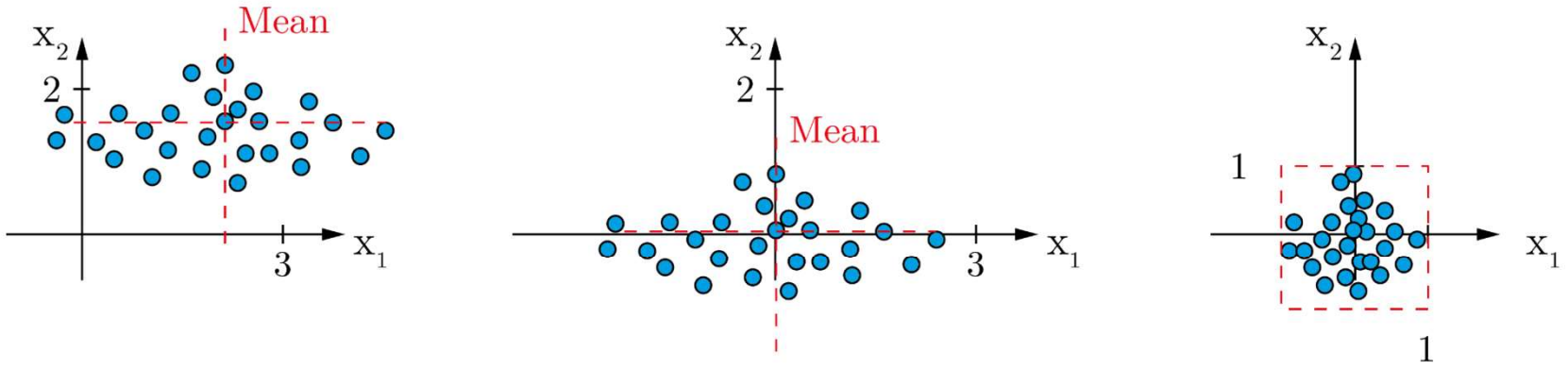


1. Subsract Mean

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$x_i = x_i - \mu$$

Data Distribution of the Input Data



1. Subsract Mean

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$x_i = x_i - \mu$$

2. Normalize Variance

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x_i^2$$

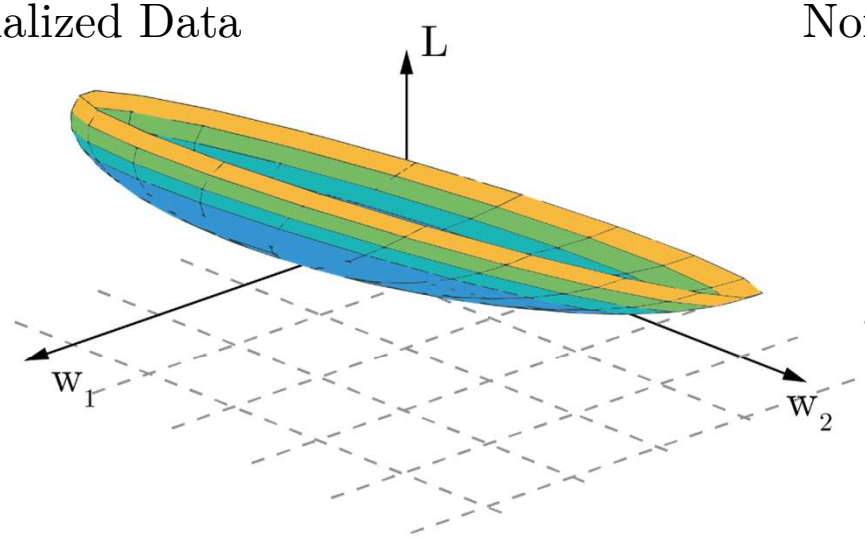
$$x_i = \frac{x_i}{\sigma^2}$$

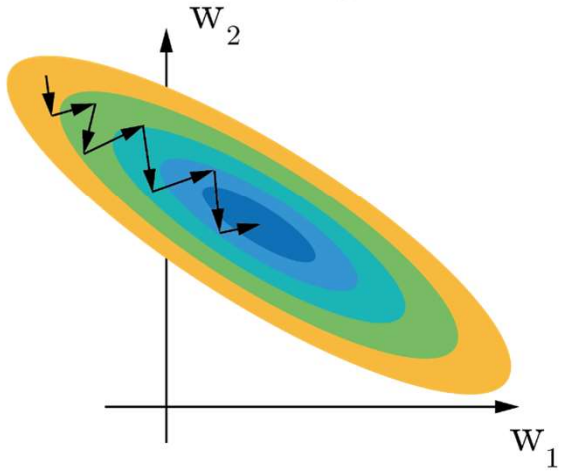
Additional Slides

Before actually training the network, the training data needs to be in the right format. For images usually the data is in the rgb color space and the value for each color is between 0-255. But most activation functions as well as classifiers expect data in the range from -1 to 1. Therefore the input data is first centered on the zero axis and than scaled by the variance.

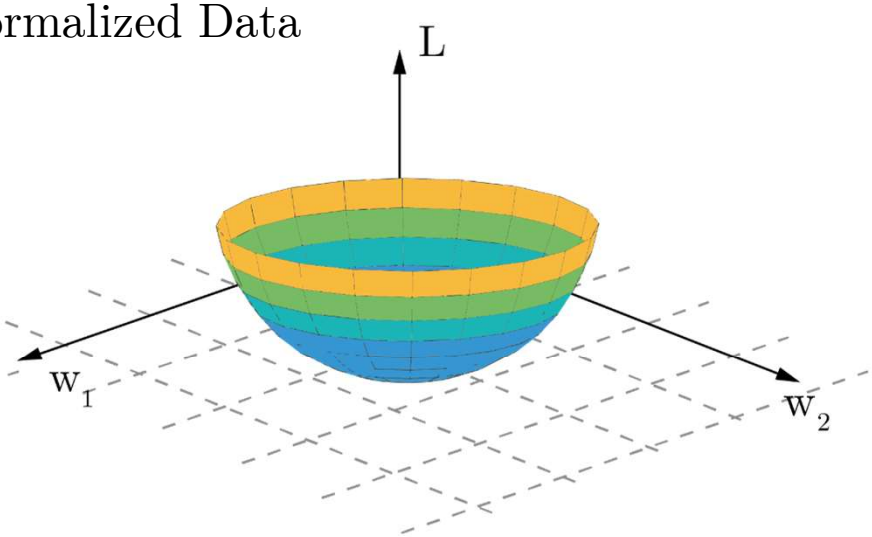
Data Distribution of the Input Data

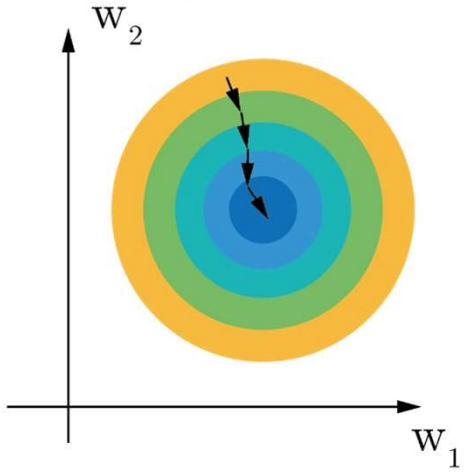
Unormalized Data





Normalized Data





End

Excursion

Regularization

Until now:

$$\Theta \equiv \{W^1, b^1, ..., W^L, b^l\}$$

Training step :

$$\Theta = \Theta - \alpha \Delta \Theta$$

$$\Delta \Theta = \vec{\nabla}_{\Theta} L(x, y)$$

Adding regularization term:

$$\Delta \Theta = \vec{\nabla}_{\Theta} L(x, y) + \lambda \vec{\nabla}_{\Theta} \Omega(\Theta)$$

Why?

- penalize complicated neural networks
- method to stop specialized neural networks and train more generalized networks
- works well against overfitting

L1 regularization :

$$\Omega(\Theta) = \sum_{l} \sum_{i} \sum_{j} | w^l_{i,j} |$$

$$\vec{\nabla}_{\Theta^l} \Omega(\Theta) = \text{sign}(W^l)$$

L2 regularization :

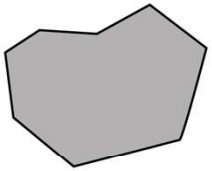
$$\Omega(\Theta) = \sum_{l} \sum_{i} \sum_{j} (w^l_{i,j})^2$$

$$\vec{\nabla}_{\Theta^l} \Omega(\Theta) = 2W^l$$

Regularization

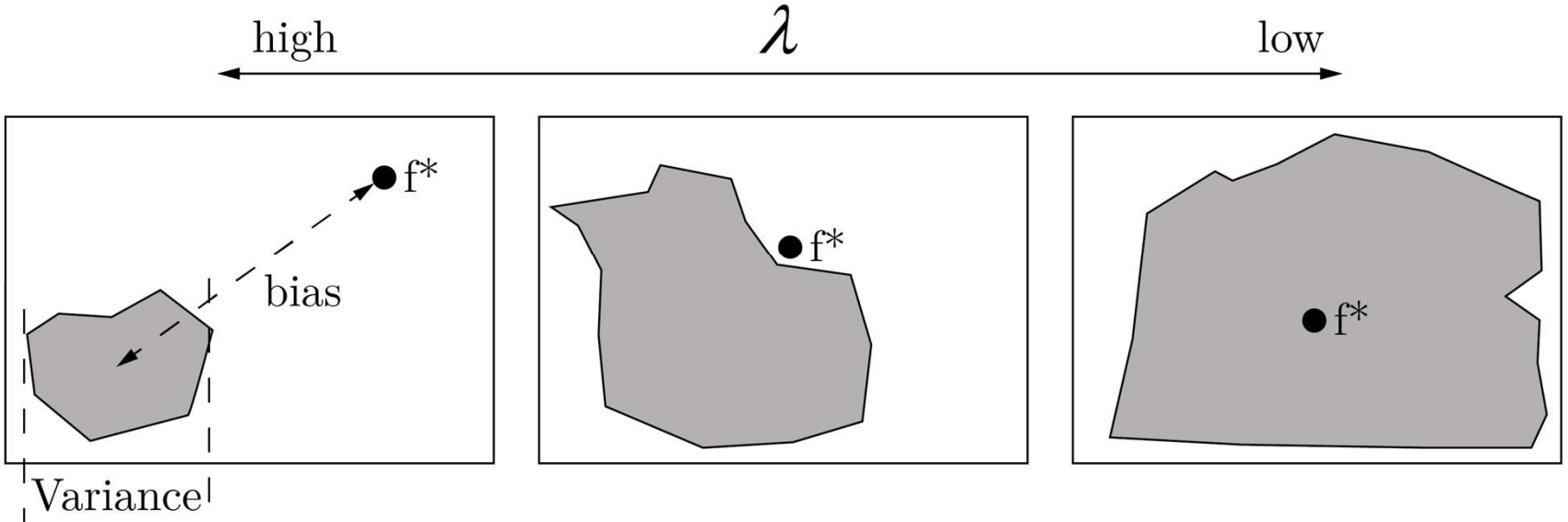
Bias and Variance of Neural Networks:

- f* : the real function

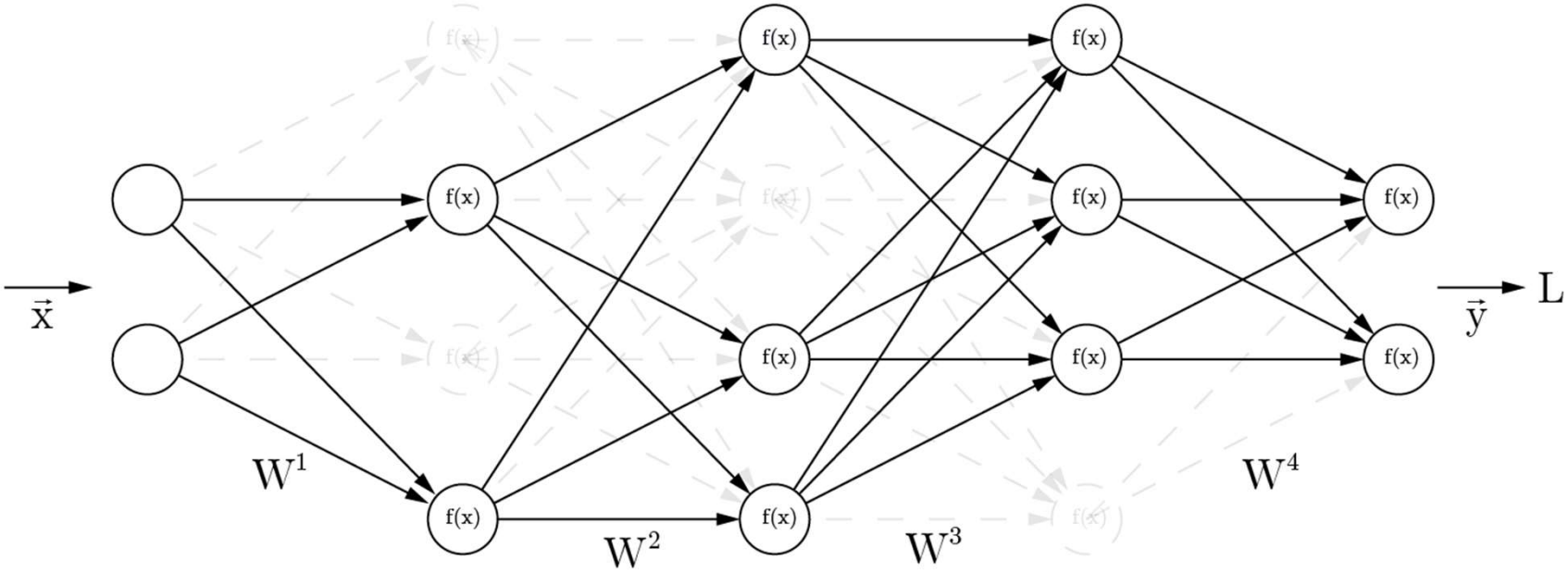


:possible Neural network function

$$\Delta\Theta = \vec{\nabla}_{\Theta} L(x, y) + \lambda \vec{\nabla}_{\Theta} \Omega(\Theta)$$



Regularization: Dropout



Kommentarfolie

*Dies ist eine Kommentarfolie.

Wir nutzen diese Folien, um zusätzeliche Inhalte zu erläutern (Beispielsweise größere Rechnungen, Zusatztexte in langer Prosa etc.)

Diese Folie wird NICHT in der Vorlesung gezeigt, sondern auskommentiert

Diese Folie kommt in das Skript für die Studenten und ist Zusatzinformation.*