Karlsruher Institut für Technologie

# Maschinelles Lernen II - Fortgeschrittene Verfahren
# V05 Deep Learning II - Convolutional Neural Networks

**Sommersemester 2017**

Dipl.-Inform. Michael Weber

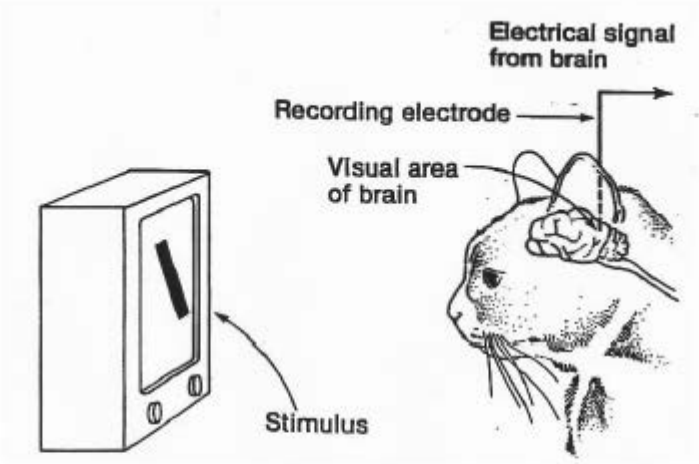
INSTITUT FÜR ANGEWANDTE INFORMATIK UND FORMALE BESCHREIBUNGSVERFAHREN
INSTITUT FÜR ANTHROPOMATIK UND ROBOTIK

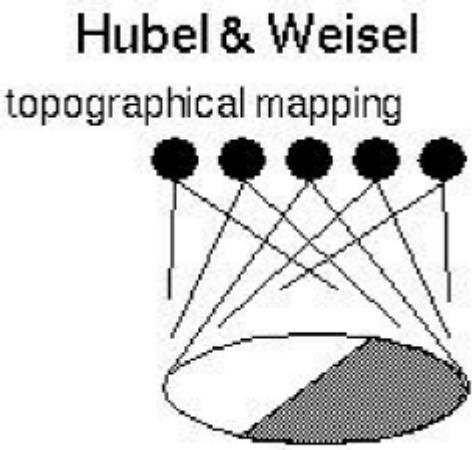
# Inhaltsübersicht

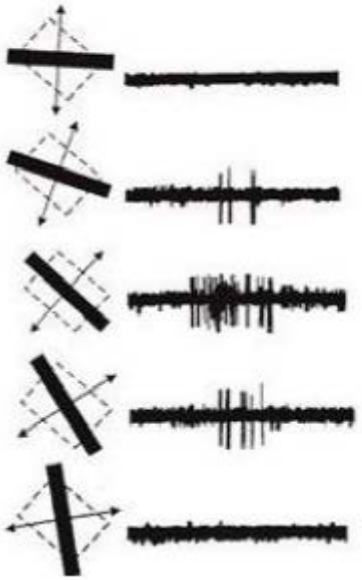
- Motivation
  - “Bildverarbeitung” bei Säugetieren
  - Erste künstliche Ansätze
- Grundlagen
  - Faltung (Convolution)
- Convolutional Neural Networks
  - Architektur
  - Layer-Typen
  - Visualisierung der Filter
- Detektion von Objekten mit CNNs

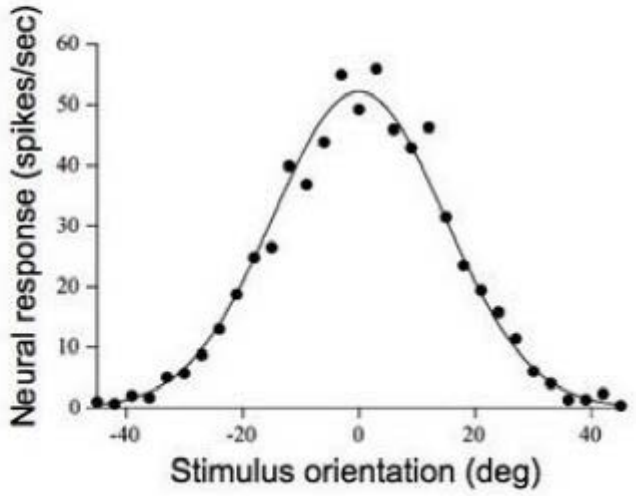
# MOTIVATION

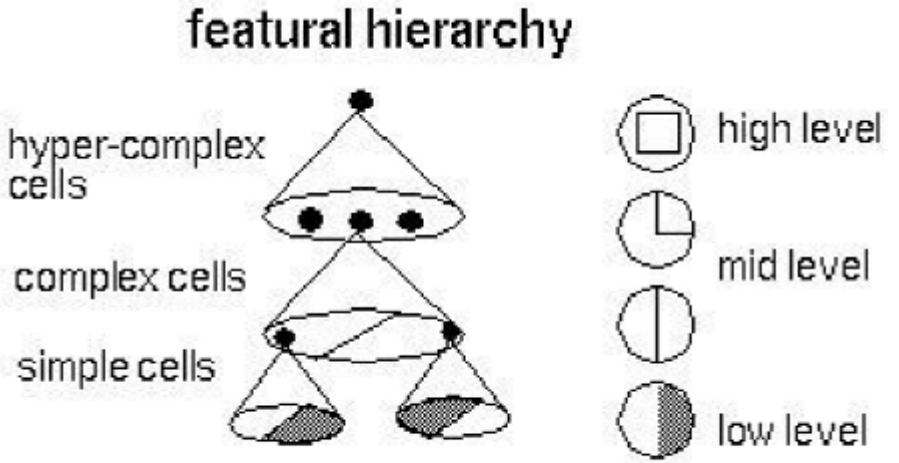
# Hubel & Wiesel (ab 1959)



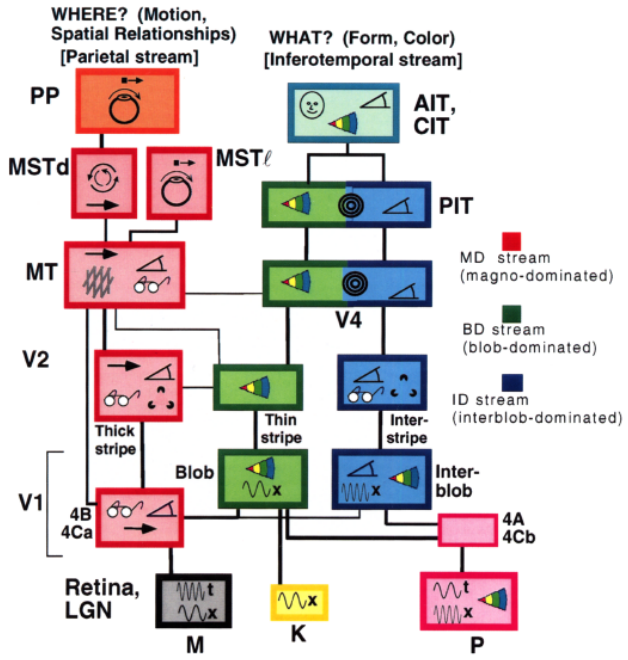





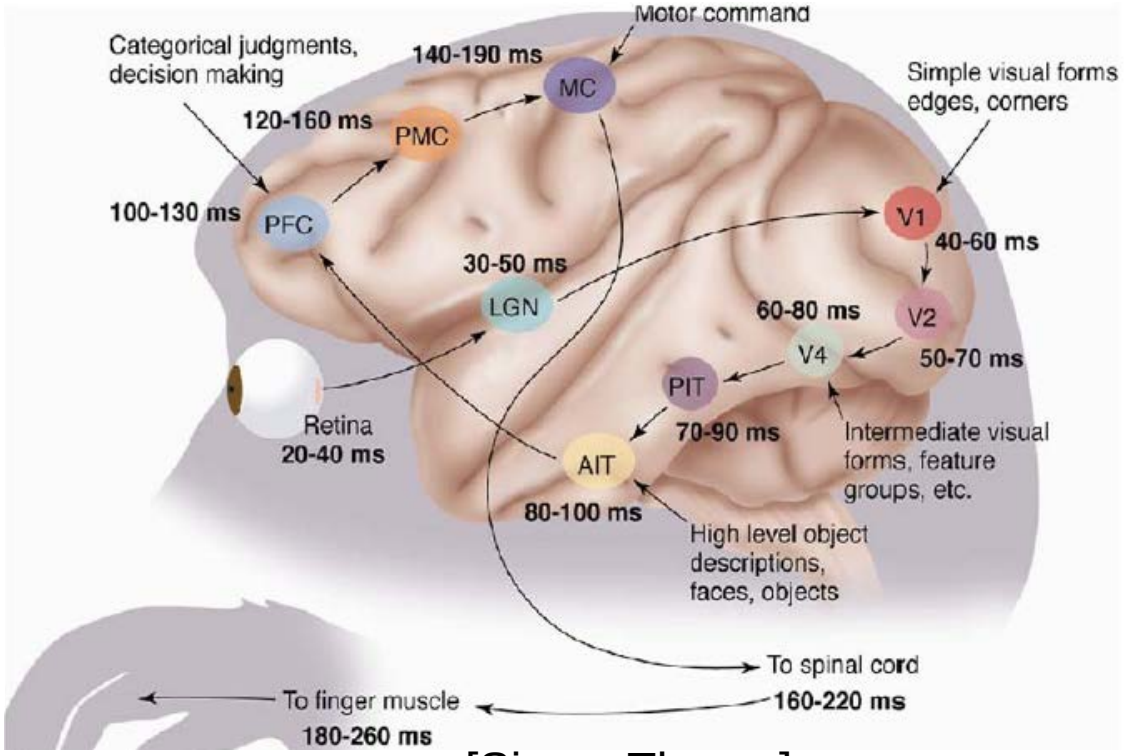



# „Bildverarbeitung“ bei Säugetieren

- Der ventrale Pfad im visuellen Cortex
- Retina – LGN – V1 – V2 – V4 – PIT - AIT



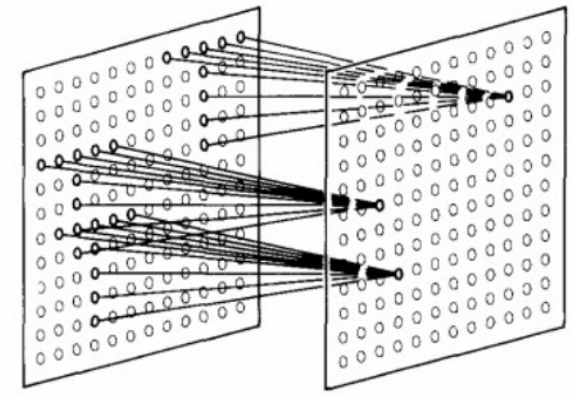

[Gallant, van Essen]

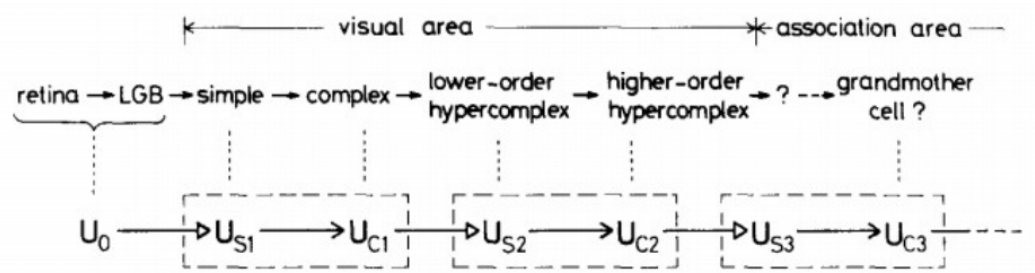


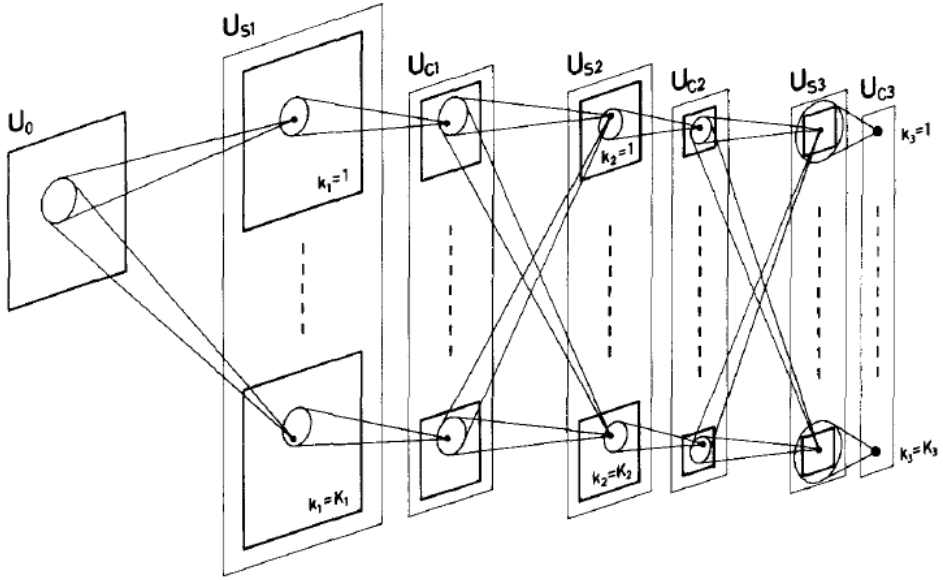

[Simon Thorpe]

# Erste künstliche Ansätze

Neurocognition
[Fukushima 1980]





# GRUNDLAGEN

# Faltung

- Das „Convolution“ in Convolutional Neural Networks
- In diesem Fall:
  - Diskrete Faltung
- Bekannt u. a. aus der Bildverarbeitung
  - Kantenfilter
  - Bildglättung
  - Bildschärfung

# Faltung theoretisch

- Gegeben: Graustufenbild I

$$I: \{1, \dots, m_1\} \times \{1, \dots, m_2\} \to W \subseteq \mathbb{R}, (i, j) \mapsto I_{i,j}$$

- und Filter $\mathrm{K} \in \mathbb{R}^{2n_1+1 \times 2n_2+1}$
- entspricht deren Faltung $(\mathrm{I} * \mathrm{K})$

$$(I * K)_{r,s} := \sum_{u=-n_1}^{n_1} \sum_{v=-n_2}^{n_2} K_{u,v} I_{r+u,s+v}$$

- Mit gegebenem Filter $K$

$$K = \begin{pmatrix} K_{-n_1,-n_2} & \cdots & K_{-n_1,n_2} \\ \vdots & K_{0,0} & \vdots \\ K_{n_1,-n_2} & \cdots & K_{n_1,n_2} \end{pmatrix}$$

# Faltung praktisch im Beispiel

Eingabebild

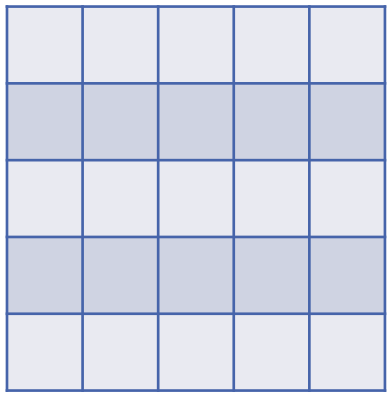
$m \times m$

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter(-Kernel)

$n \times n$

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |



Ergebnisbild

$m_{neu} \times m_{neu}$

$m_{neu} = m - n + 1$

# Faltung praktisch im Beispiel

Eingabebild

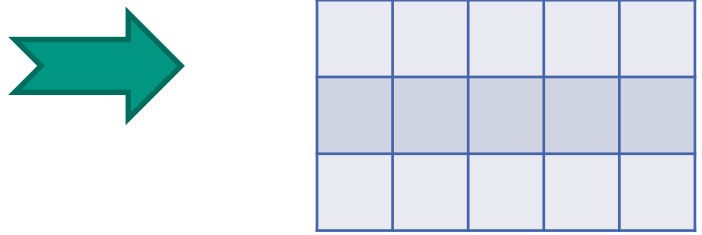
$m \times m$

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter(-Kernel)

$n \times n$

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |




Ergebnisbild

$m_{neu} \times m_{neu}$

$m_{neu} = m - n + 1$

# Faltung praktisch im Beispiel

Eingabebild

$m \times m$

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter(-Kernel)

$n \times n$

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 1 | 1 | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Ergebnisbild

$m_{neu} \times m_{neu}$

$m_{neu} = m - n + 1$

# Faltung praktisch im Beispiel

Eingabebild

$m \times m$

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter(-Kernel)

$n \times n$

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Ergebnisbild

$m_{neu} \times m_{neu}$

$m_{neu} = m - n + 1$

# Faltung praktisch im Beispiel

Eingabebild

$m \times m$

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter(-Kernel)

$n \times n$

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

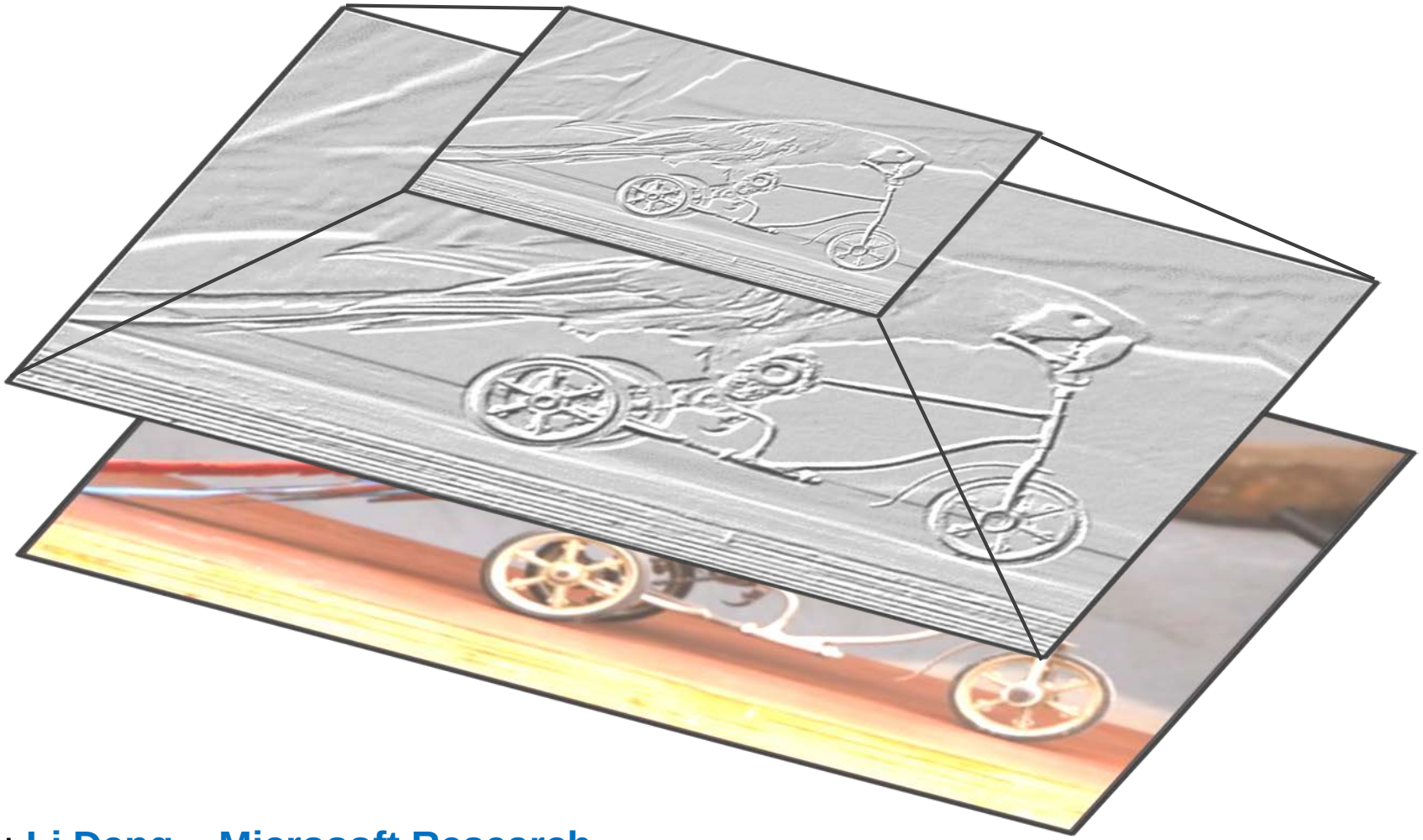
| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 |
| 1 | 2 | 2 | 2 | 2 |

Ergebnisbild

$m_{neu} \times m_{neu}$

$m_{neu} = m - n + 1$

# Faltung anschaulich



Pooling

Convolution

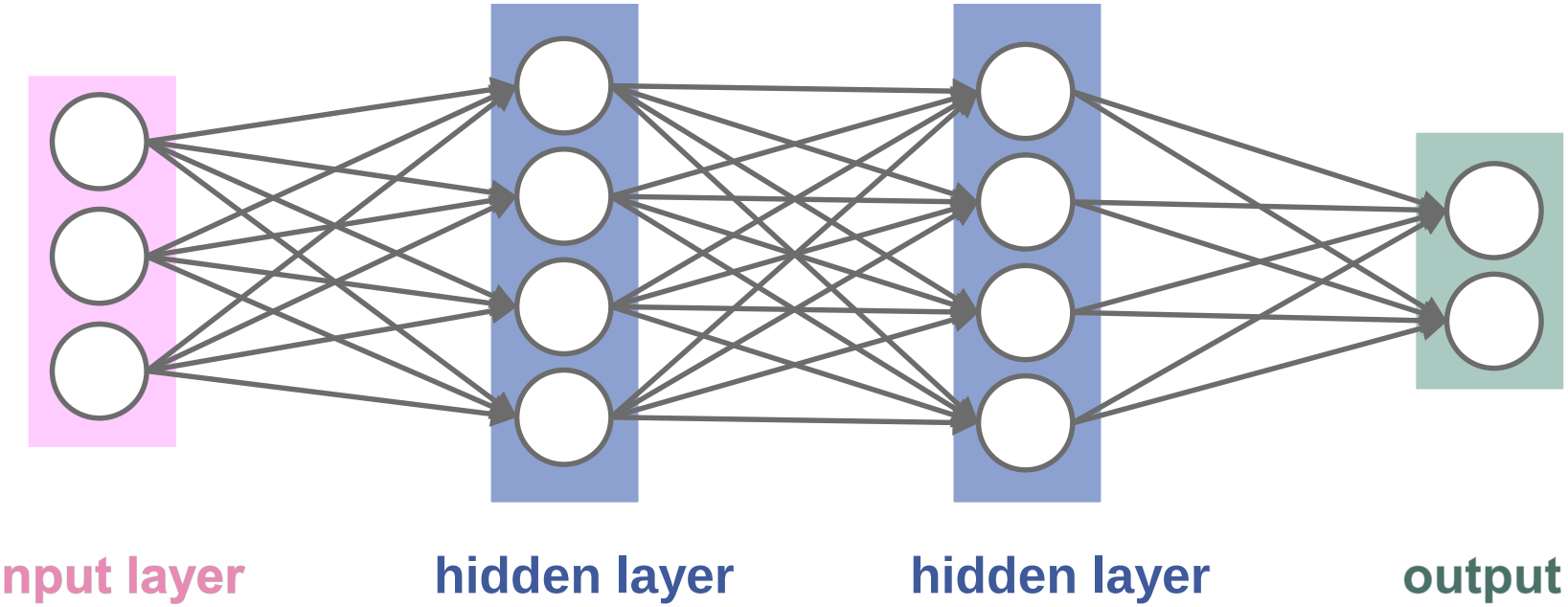
Image

Quelle: **Li Deng – Microsoft Research**

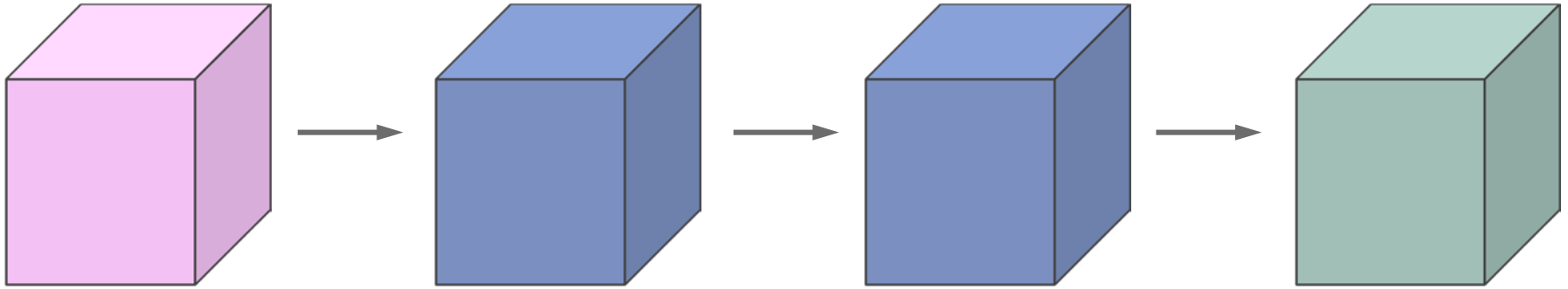
# CONVOLUTIONAL NEURAL NETWORKS

# Von KNN zu CNN

Bisher (KNN)

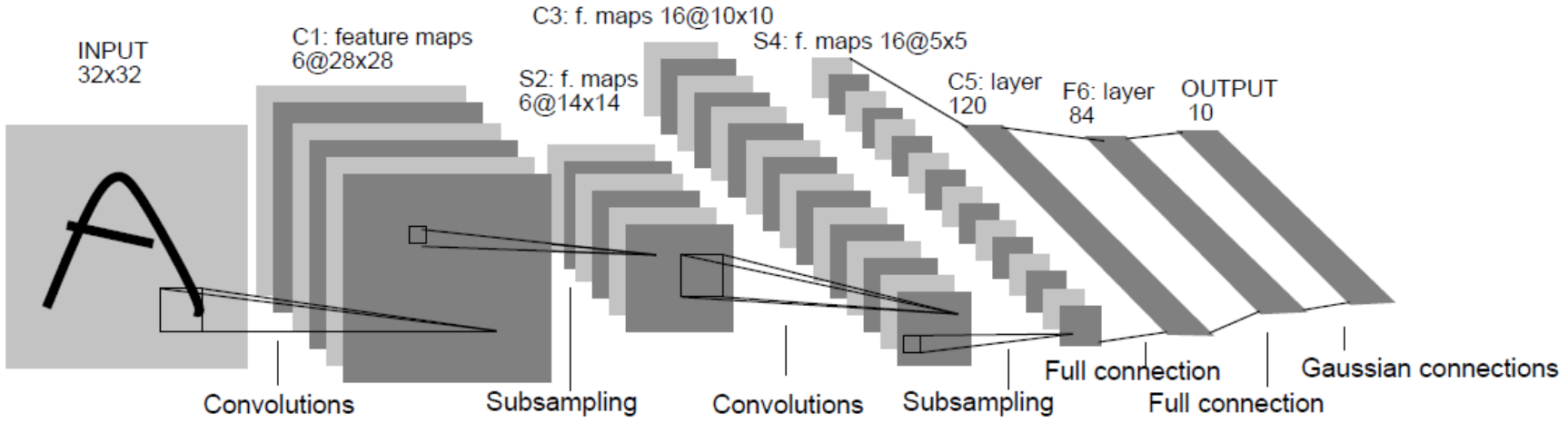



Jetzt (CNN)



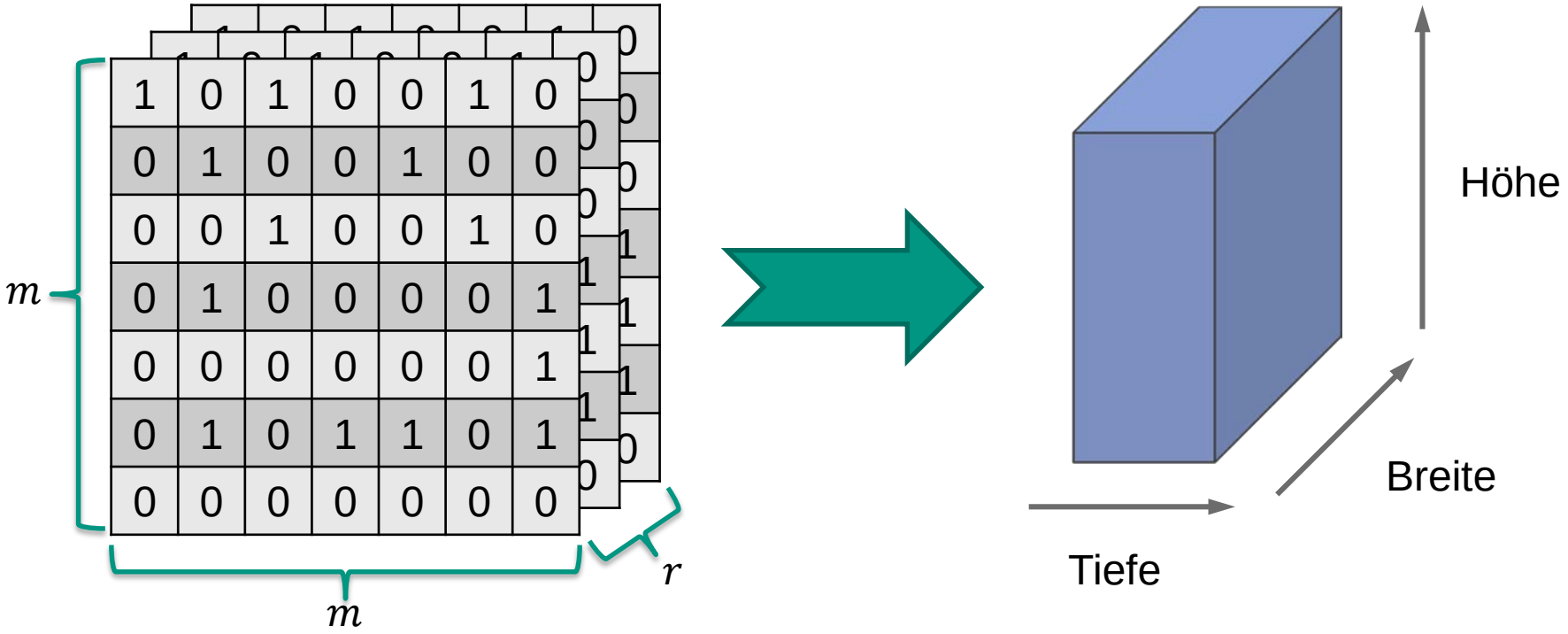
# LeNet

- Entwickelt zur Handschrifterkennung [LeCun 89]
- 7 Layer (exkl. Input)
  - 2x Convolution, 2x Subsampling (Pooling)
  - 3x Fully Connected NN (incl. Output)



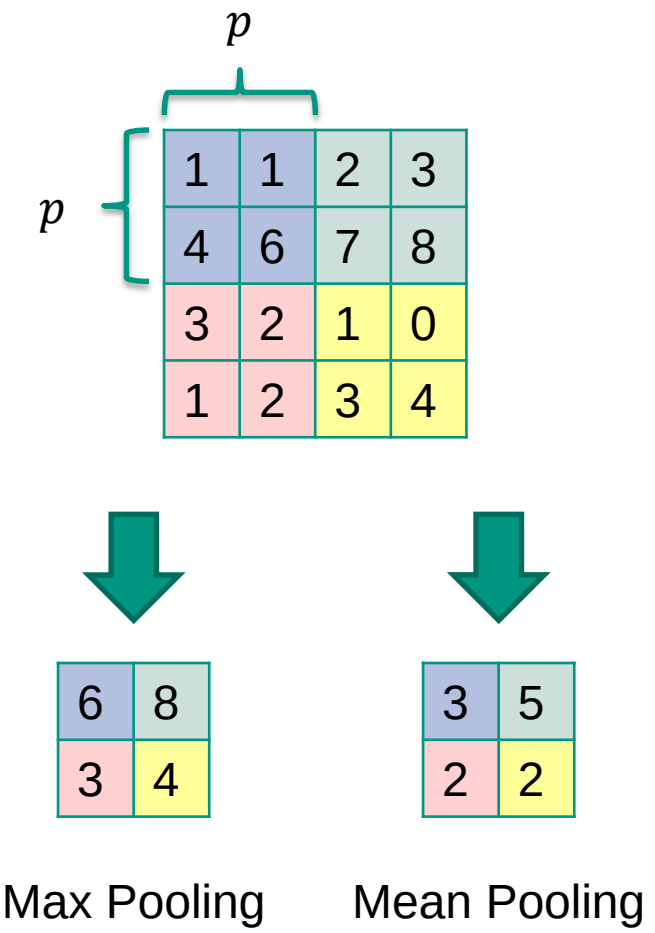
# Datenfluss im Netz

- Daten werden in Form von Feature Maps von Layer zu Layer übertragen
  - Jeder Layer erhält als Eingabe $r$ Feature Maps ($m \times m$)
  - Im Input Layer gilt: Feature Maps ≡ Eingabebild (RGB Eingabe: $m \times m \times 3$)
  - Alle Feature Maps zwischen zwei Layern entsprechen einem Tensor



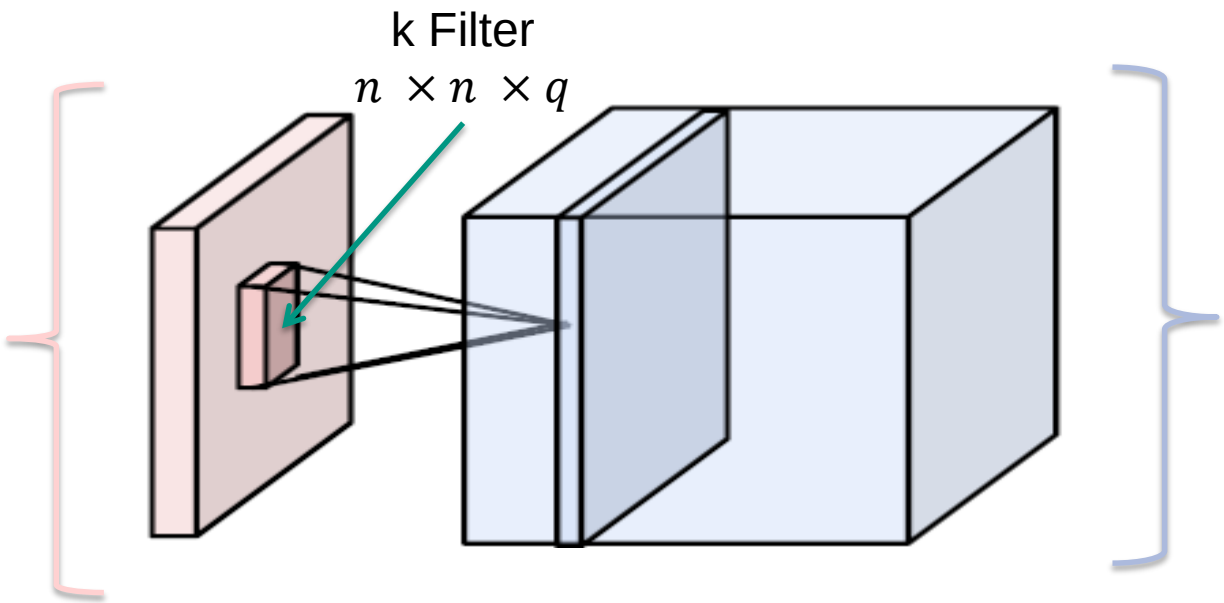
# Pooling Layer

- Zusammenfassung kleiner Bildbereiche
  - $p \times p$ Bereich
  - $p$ gewählt nach Größe des Eingabebildes
  - Meist $p \in [2,5]$
  - 2 für eher kleine Inputs, 5 eher große
- Verschiedene Strategien
  - Max Pooling $\max\{ a \in A \}$
  - Mean Pooling $\frac{1}{|A|} \sum_{a \in A} a$
  - Stochastic Pooling
- Nutzen
  - Lokale Translationsinvarianz
  - Invarianz gegen leichte Veränderungen und Verzerrungen
  - Datenreduktion



# Convolution Layer

- Anwendung von Faltungsoperationen auf die Eingabe
- Eingabe: Feature Maps des vorherigen Layers
- Ausgabe: Feature Maps des Layers
  - Entstanden durch Faltung der Eingabe mit k Filtern




Ausgabe

$$m_{neu} \times m_{neu} \times k$$

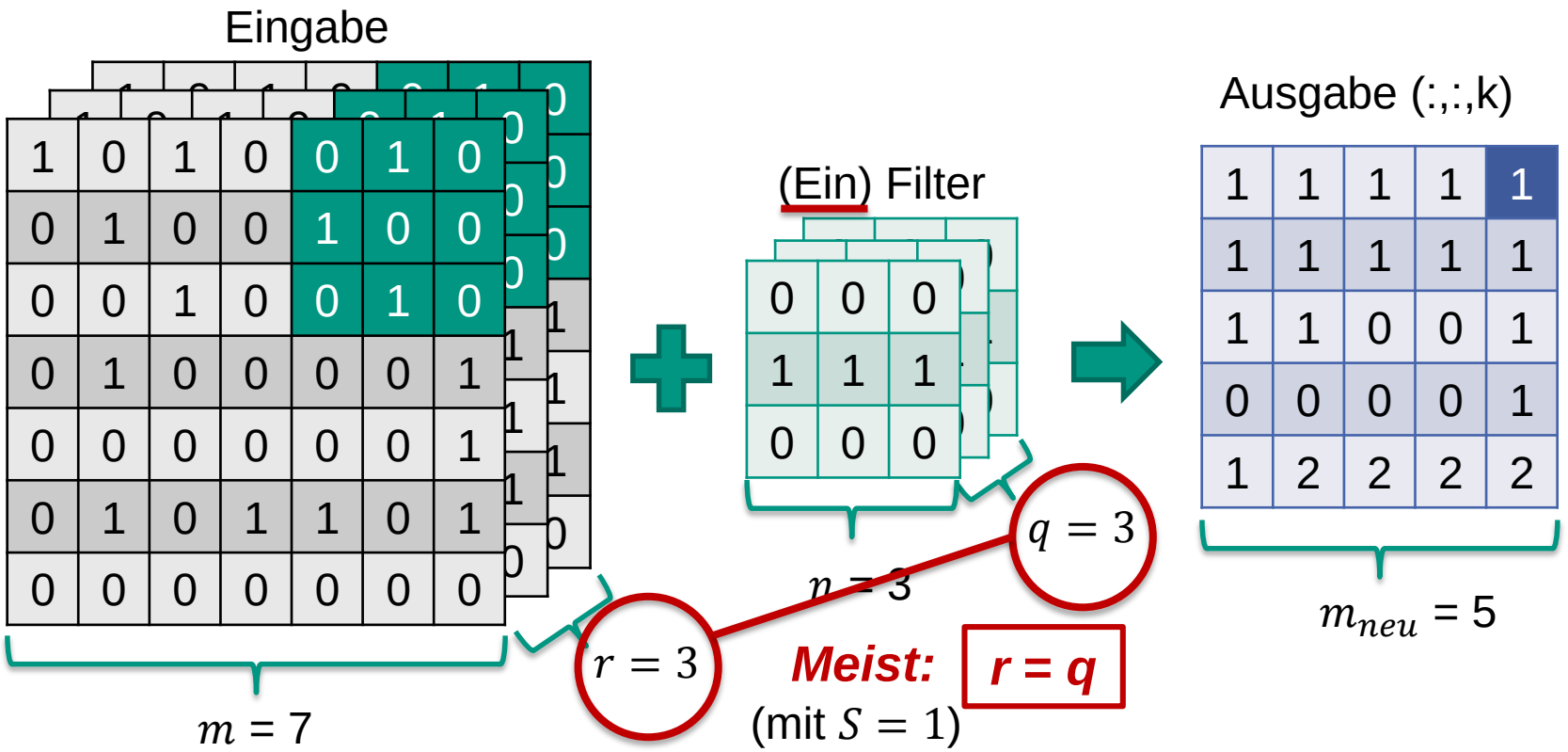
wobei

$$m_{neu} = \frac{(m-n)}{s} + 1$$

s = Stride

# Faltung im Detail

- Faltung erfolgt auf beliebigen Schichten der Eingabe
- Beispielhaft für einen Filter:



# Stride

- „Schrittweite“ des Faltungskernels
  - In beiden Dimensionen
- Feature Map Größe:
  - z.B. $m = 7, n = 3$:
    - Stride 1 → $\frac{(7-3)}{1} + 1 = 5$
    - Stride 2 → $\frac{(7-3)}{2} + 1 = 3$
    - Stride 3 → $\frac{(7-3)}{3} + 1 = \cdots$

$$m_{neu} = \frac{(m-n)}{s} + 1$$

- Stride muss zur Größe der Eingabe passen
  - In Abhängigkeit zur Größe des Filterkernels
- Bei Convolution- und Pooling Layern
  - Bei Pooling Layern meist s = 1

Stride = 1

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Stride = 2

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |

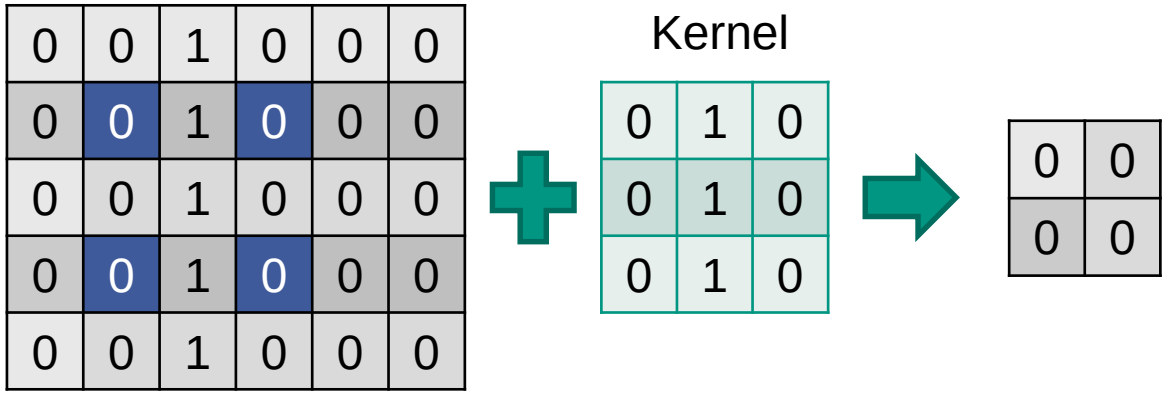
Blau = Mittelpunkte des Filter-Kernels

# Größenreduktion: Stride vs. Pooling

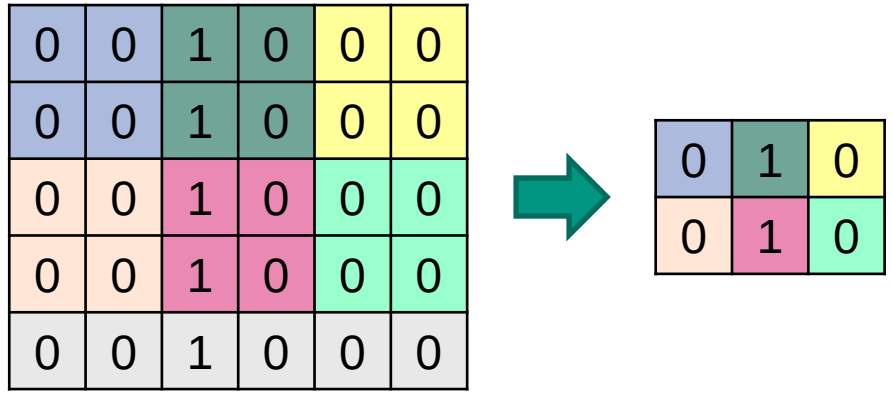
- Größenreduktion durch ConvLayer Stride möglich
- Wieso extra Layer?
- Wichtiger Unterschied:
  - ConvLayer Stride erreicht Reduktion durch Auslassen
  - Max Pooling betrachtet alle Werte bei Reduktion

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |

Convolution - Stride 2

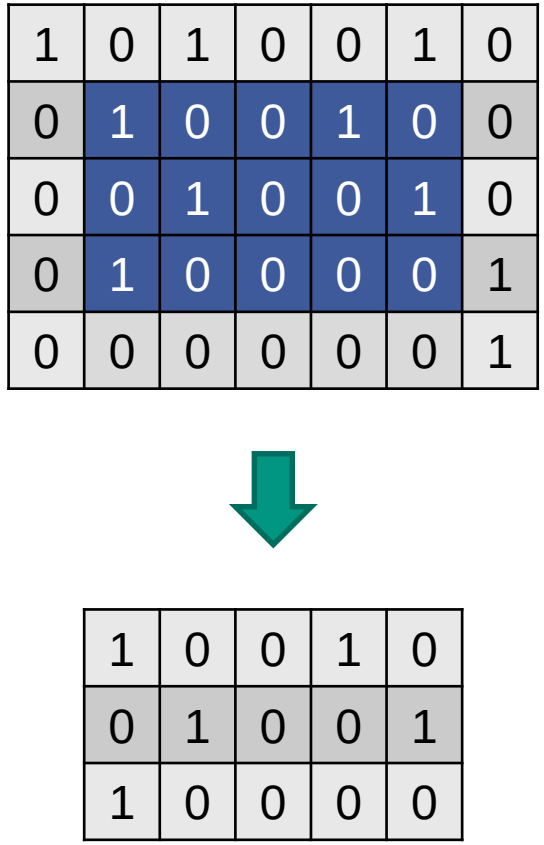



2x2 Max Pooling - Stride 2



# Randbetrachtung

- Relevant v. A. bei Convolution Layern
  - Bei Convolution Filtern größer $1 \times 1$
- Ursprungsgröße geht durch Randeffekte bei Faltung verloren
  - Mögliche Strides für jeden Convolution Layer andere
- Möglichkeiten zur Behandlung:
  - Don‘t care:
    - Berechnung des inneren Bereichs
    - Reduktion der Ergebnisgröße
  - Padding
    - Auffüllen des Randbereichs



# Padding

- Stabilisiert Größenverlauf der Layer
- Padding
  - Auffüllen der verlorenen Zeilen / Spalten
  - Ursprungsgröße bleibt erhalten
- Varianten:
  - Zero Padding:
    - Auffüllen der Randbereiche mit 0
  - Nearest Padding:
    - Duplizieren der Randpixel
  - Reflect Padding:
    - Matrix nach außen spiegeln

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Zero Padding

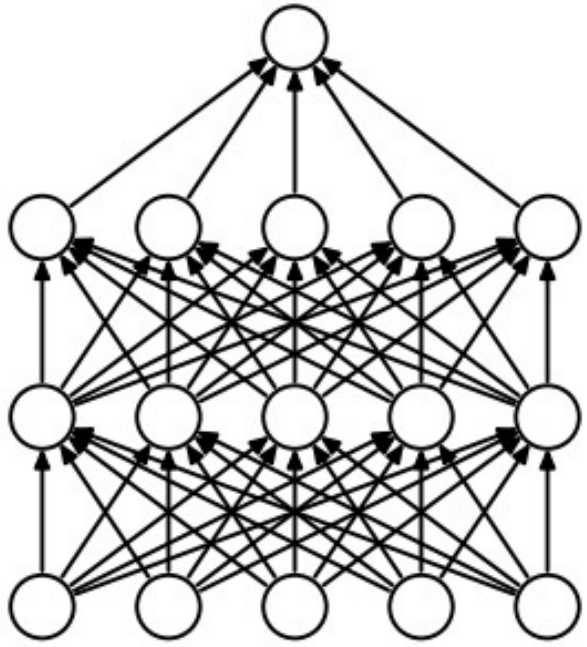
# Padding – Mögliche Probleme

- Ohne Padding:
  - Informationen am Bildrand gehen sukzessive verloren
  - Tritt in jedem Convolution Layer ohne Padding auf
    - Große Filter beschleunigen das Problem
    - Pooling mit großen p beschleunigt das Problem ebenso

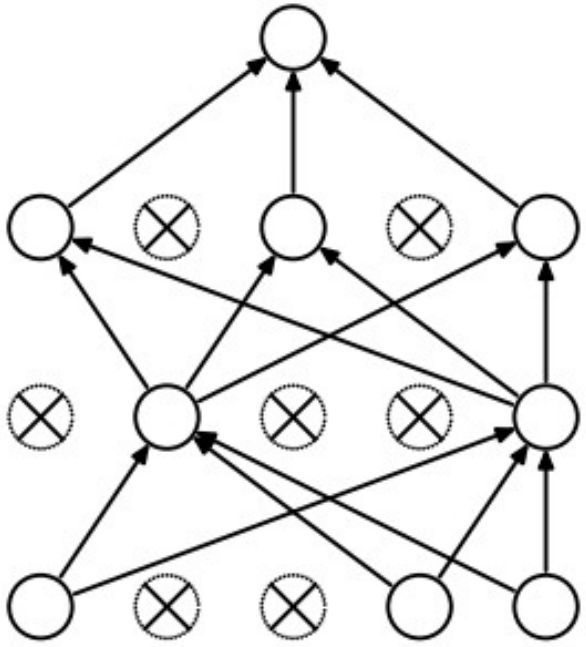
- Mit Padding:
  - Gefahr, dass am Rand Strukturinformationen entstehen, die im Eingabebild nicht vorhanden waren
  - z. B. Kanten

# Dropout

- Ziel
  - Reduktion von Overfitting
- Methode
  - Während des Trainings
    - Deaktivieren einzelner Neuronen mit Wahrscheinlichkeit p
    - Entscheidung über Deaktivierung in jedem Trainingsschritt
- Dropout zählt zu den Regularisierungsmethoden für CNNs



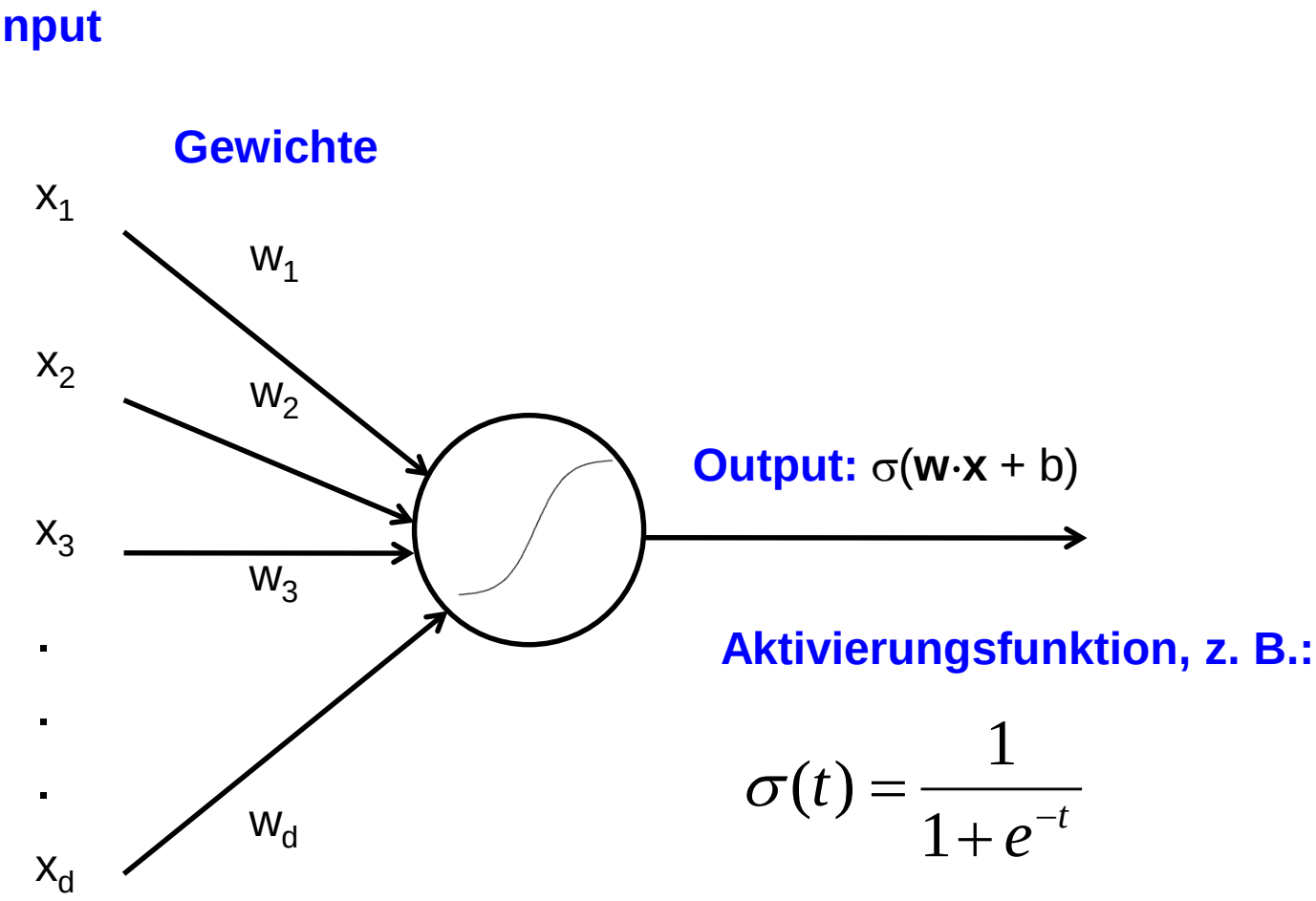
(a) Standard Neural Net



(b) After applying dropout.

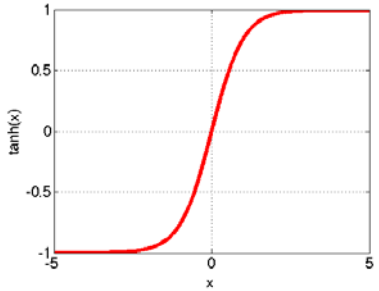
Srivastava et al. - *Dropout. A Simple Way to Prevent Neural Networks from Overfitting*

# Rückblick Perceptron

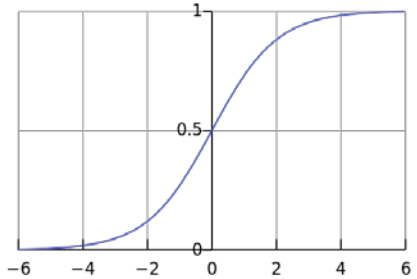


# Activation Layer

- Nichtlineare Aktivierung
- In klassischen neuronalen Netzen:
  - Sigmoid
  - Tanh




Tanh

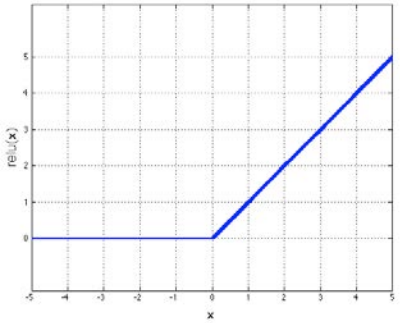


Sigmoid

- Für Convolutional Neural Networks:
  - Z. B. Rectified Linear Unit (ReLu)
    - $f(x) = \max(0, x)$
  - Einsatz nach Conv-Layer
  - Teilw. als Teil des Conv-Layer gesehen

| | | | | |
|---|---|---|---|---|
| -1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| -1 | 0 | 0 | 0 | 1 |
| 1 | 2 | 2 | 2 | 2 |

Ausgabe Conv-Layer



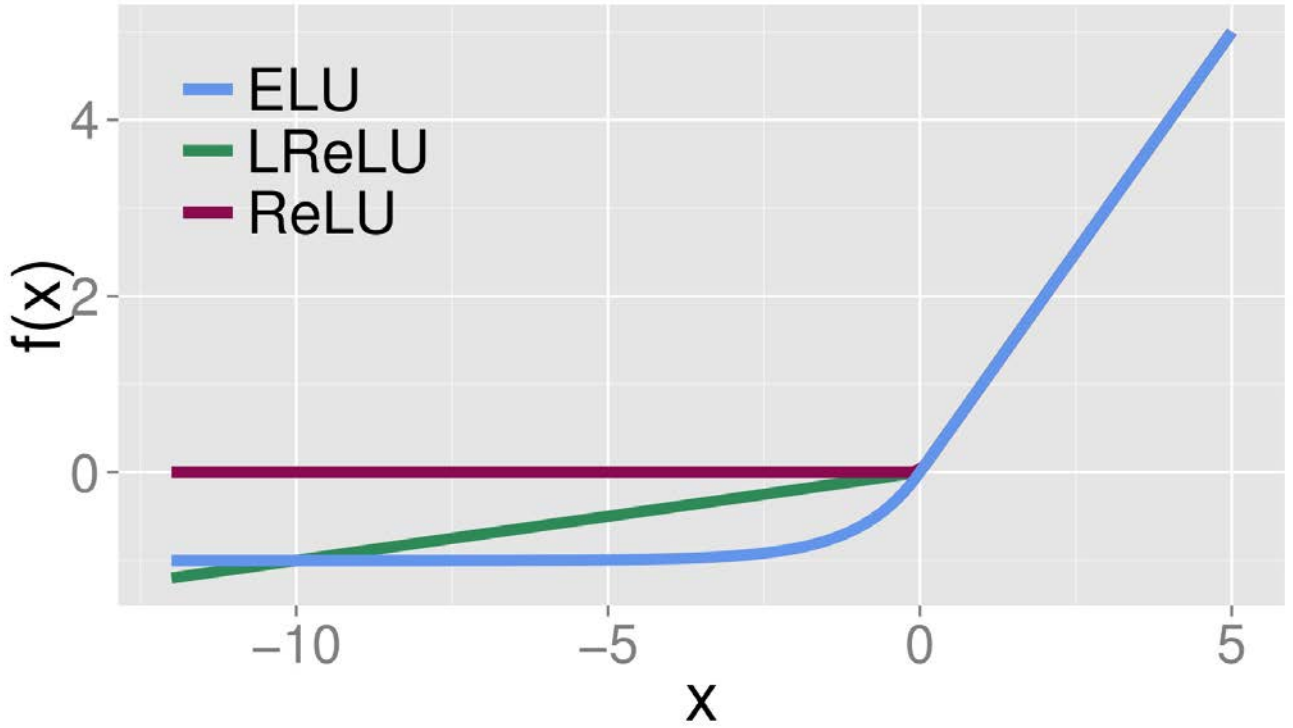

ReLu

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 |
| 1 | 2 | 2 | 2 | 2 |

Eingabe Folgelayer

# Aktivierungsfunktionen

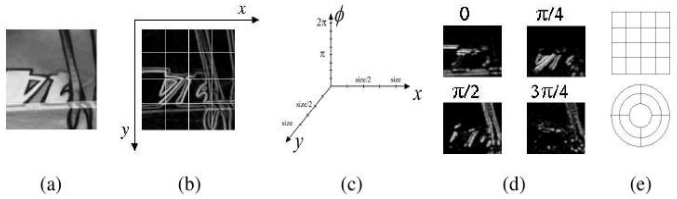
- Rectified Linear Unit (ReLU)
  - $f(x) = \max(0, x)$
- Leaky ReLU (LReLU)
  - $f(x) = \begin{cases} 0.01x & if \quad x < 0 \\ x & if \quad x \geq 0 \end{cases}$
- Parametric ReLU (PReLU)
  - Generalisierung von LReLU
  - $f(x) = \begin{cases} \alpha\, x & if \quad x < 0 \\ x & if \quad x \geq 0 \end{cases}$
- Exponential Linear Unit (ELU)
  - $f(x) = \begin{cases} \alpha(e^x - 1) & if \quad x < 0 \\ x & if \quad x \geq 0 \end{cases}$



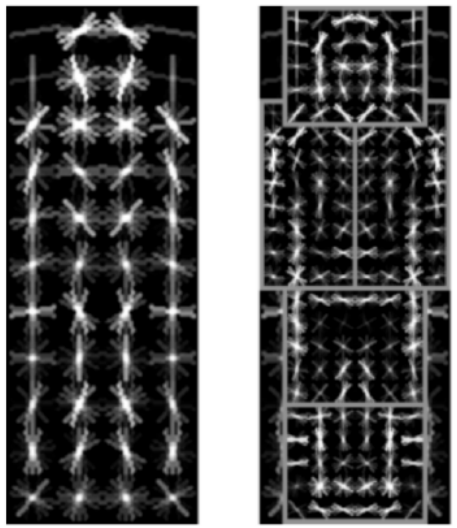

Bildquelle: ELU-Networks – Heusel et al.

# Features

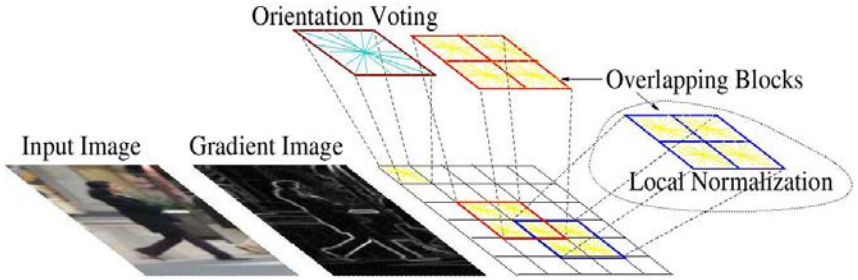
- Bis vor einigen Jahren in der Objekterkennung:
  - Wahl der Features Schlüssel zum Erfolg
  - Vielfältige manuell erstellte Features
    - SIFT, HOG, FREAK …



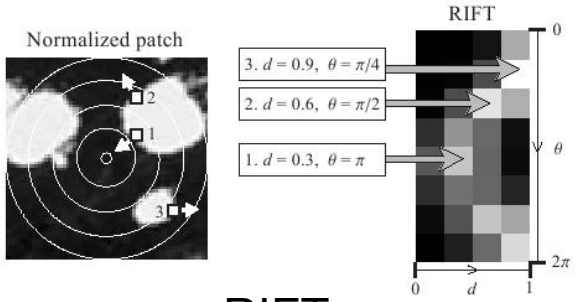

GLOH



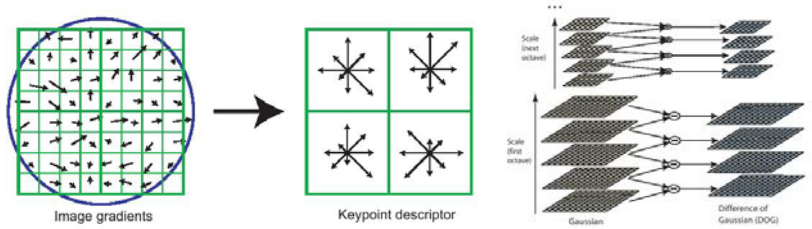
Felzenszwalb et al. 2007




HoG



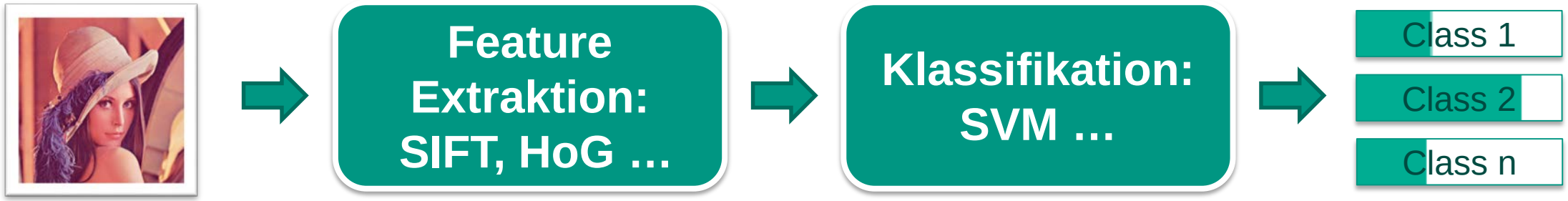

RIFT




SIFT

# Features

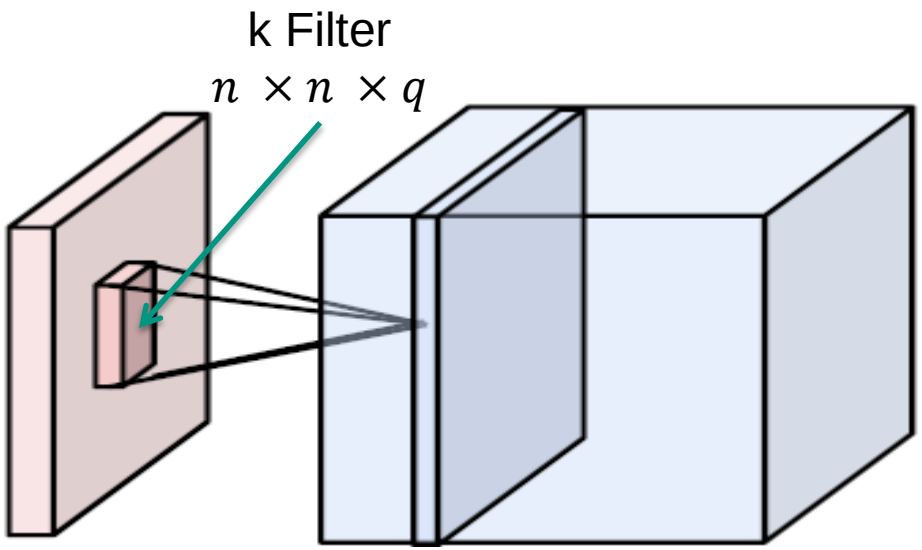
- Klassische Pipeline




- Probleme manuell erstellter Features
  - Expertenwissen notwendig
  - Features nur so gut wie der Experte
- Warum nicht gute Features für eine bestimmte Domäne lernen?
  - Features passen so immer zur aktuellen Aufgabe
- CNNs machen genau das

# Features in CNNs

- Wo sind diese Features im CNN?




- In den Filtern der Convolution Layer

# Hierarchische Features

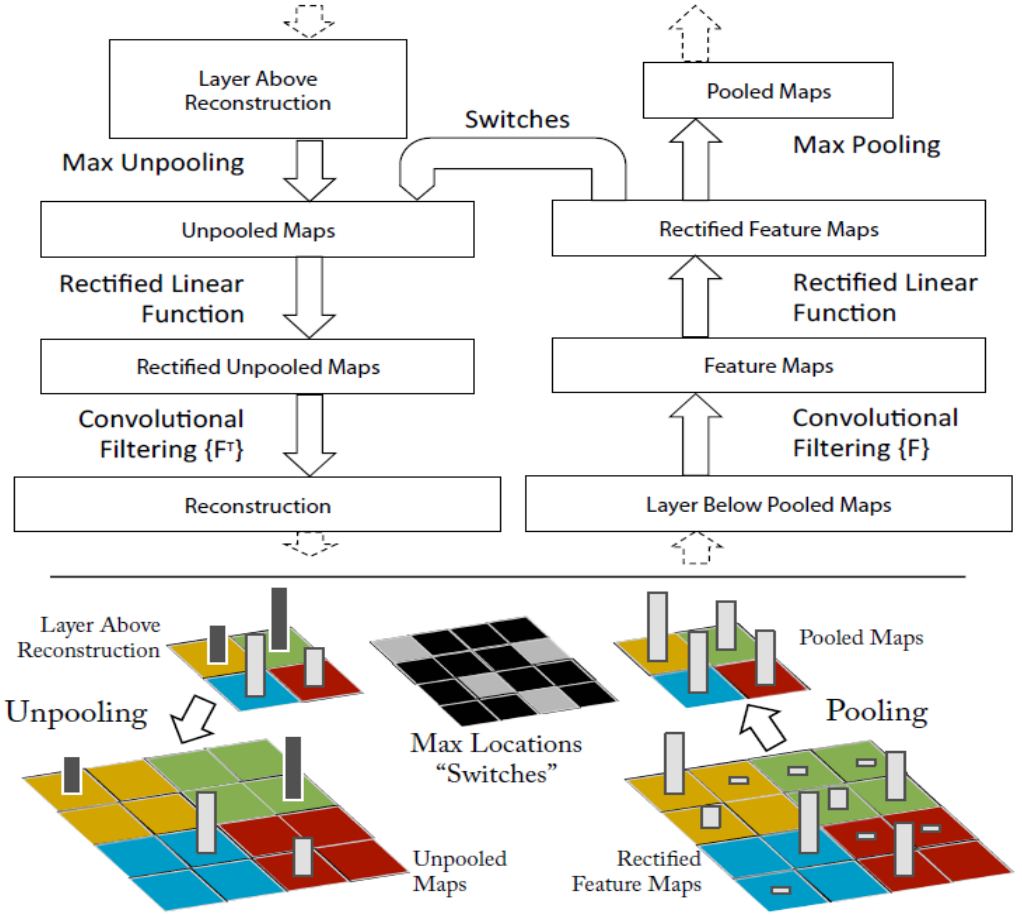
- Bedingt durch Architektur der CNNs
  - Feature Hierarchie
- Features eines Layers basieren auf den Ausgaben der Features des vorherigen Layers


- Erinnerung:
  - „Bildverarbeitung“ bei Säugetieren

# Wie sehen solche Filter aus?

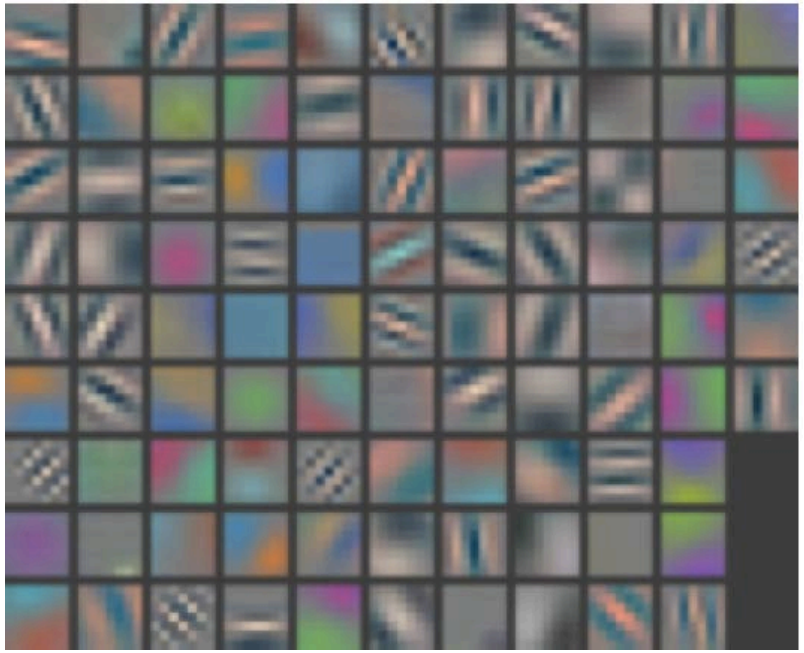
- Visualisierung von Filtern aus [Zeiler]
  - Anzeige der Feature Maps in jedem Layer
  - Nutzung von Deconvolutional Networks
    - Invertierung der einzelnen Schichten
  - Projektion der Feature Activations in den (Pixel-) Eingaberaum

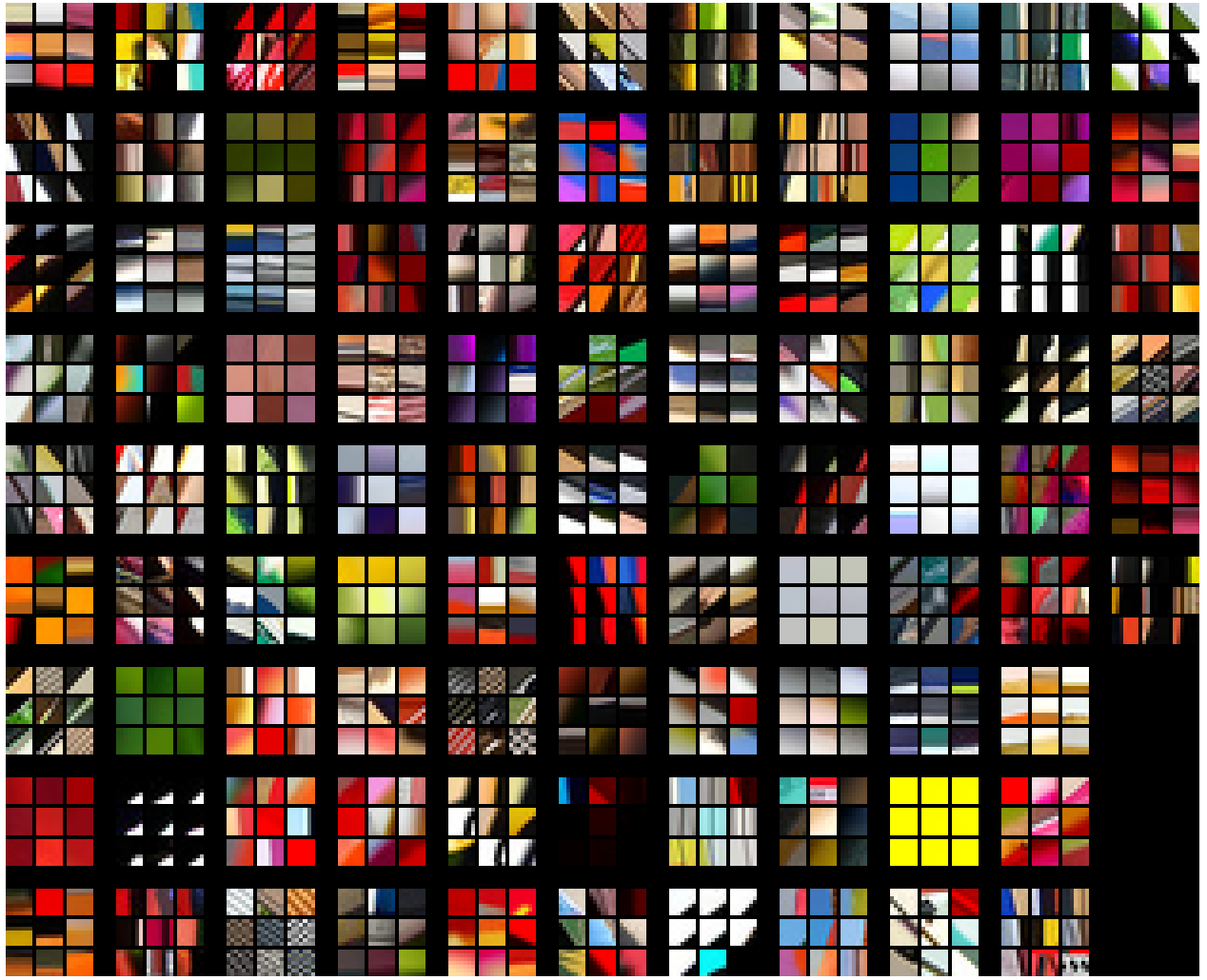
- Motivation
  - Tieferes Verständnis für die inneren Layer
  - Beobachtung der Feature Map Entstehung
    - Diagnose potentieller Probleme mit dem Lernproblem



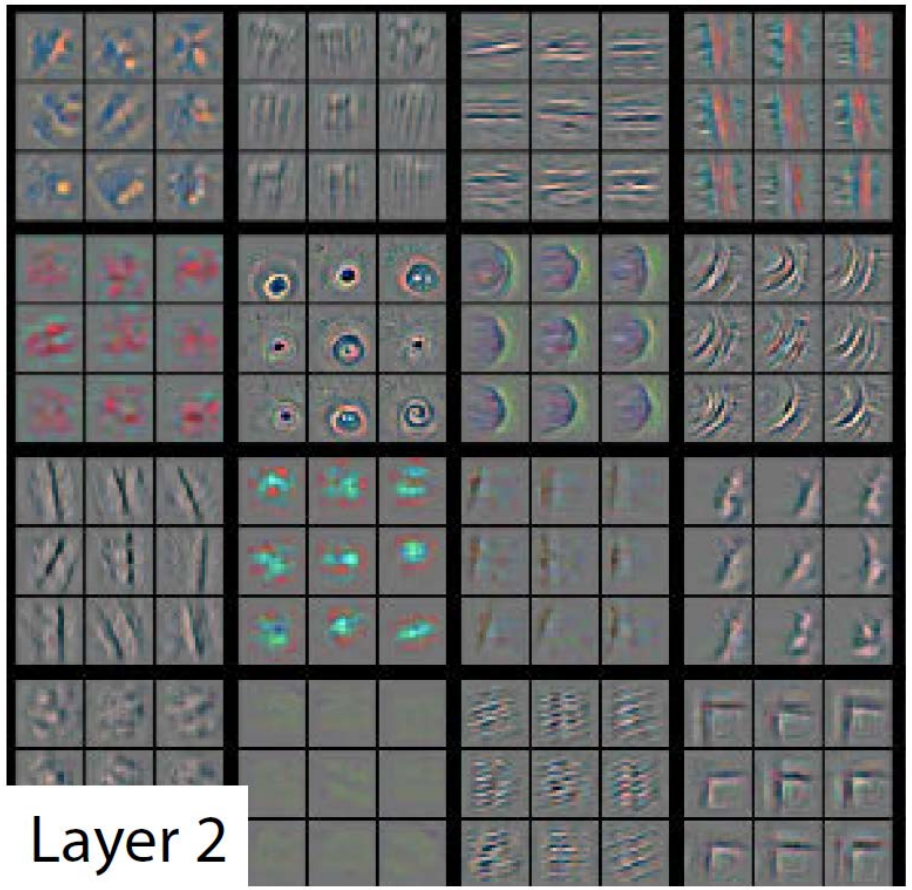

[Zeiler]: Zeiler et al., 2013: Visualizing and Understanding Convolutional Neural Networks

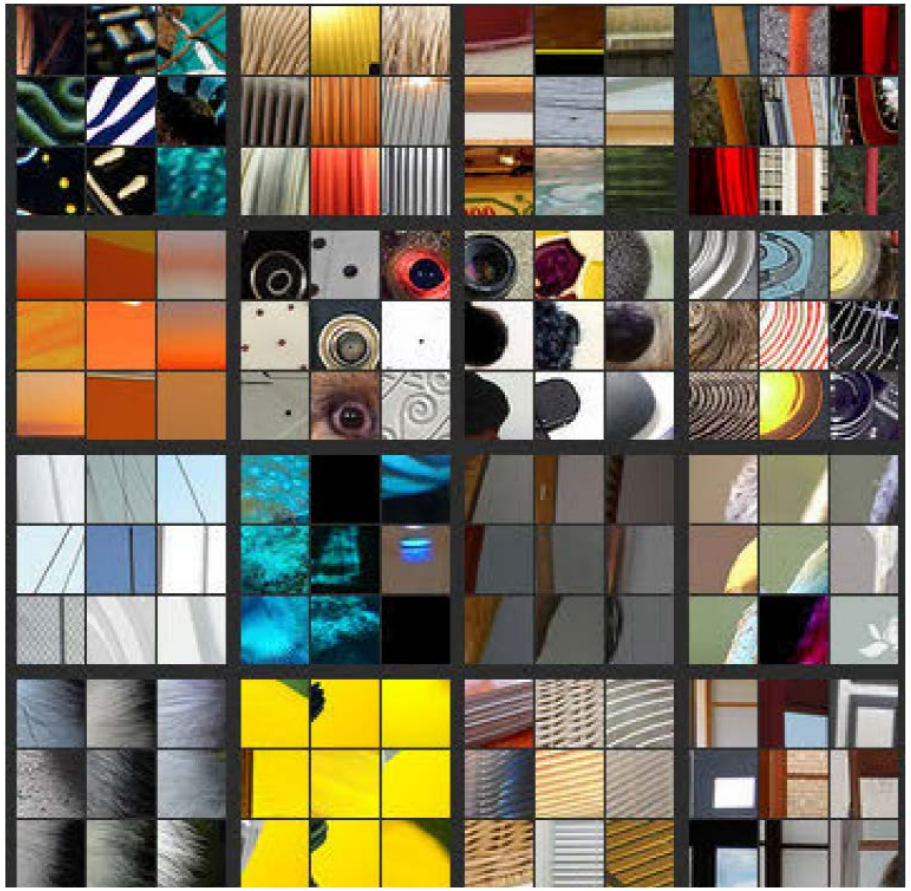
# Filter in Layer 1



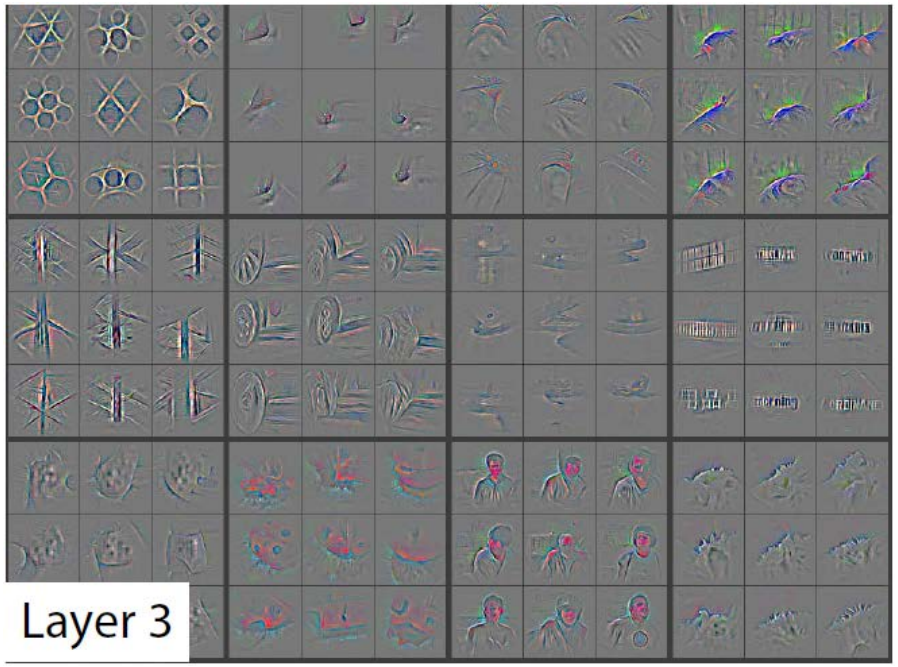


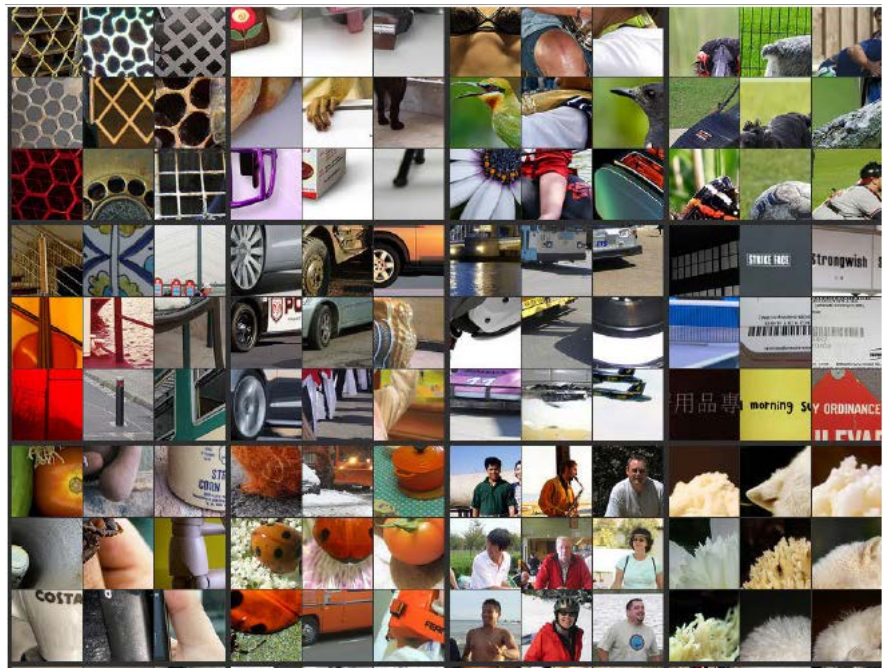
# Filter in Layer 2



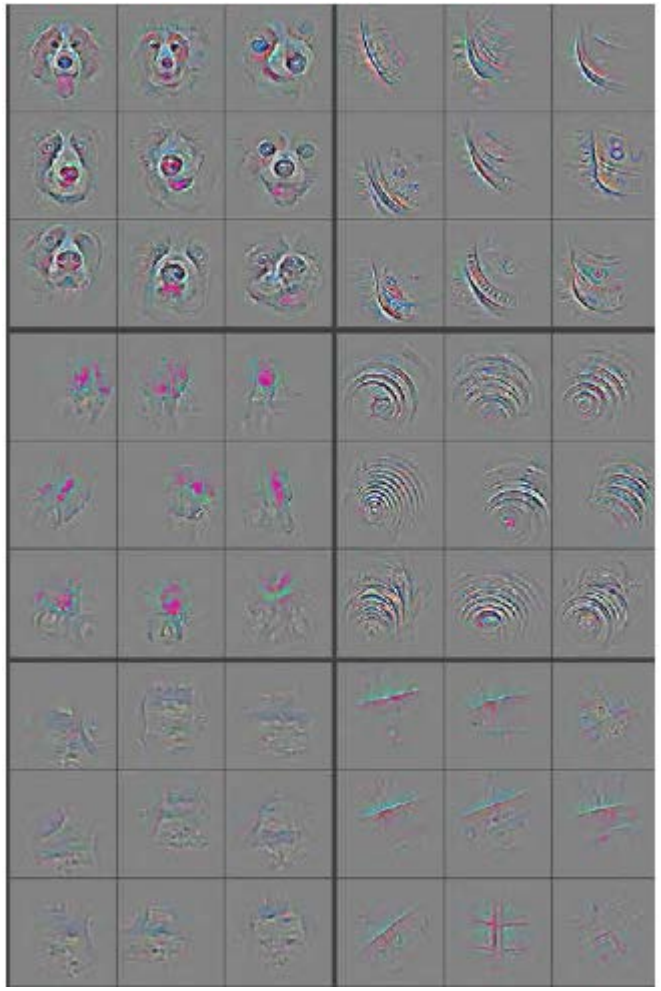



# Filter in Layer 3

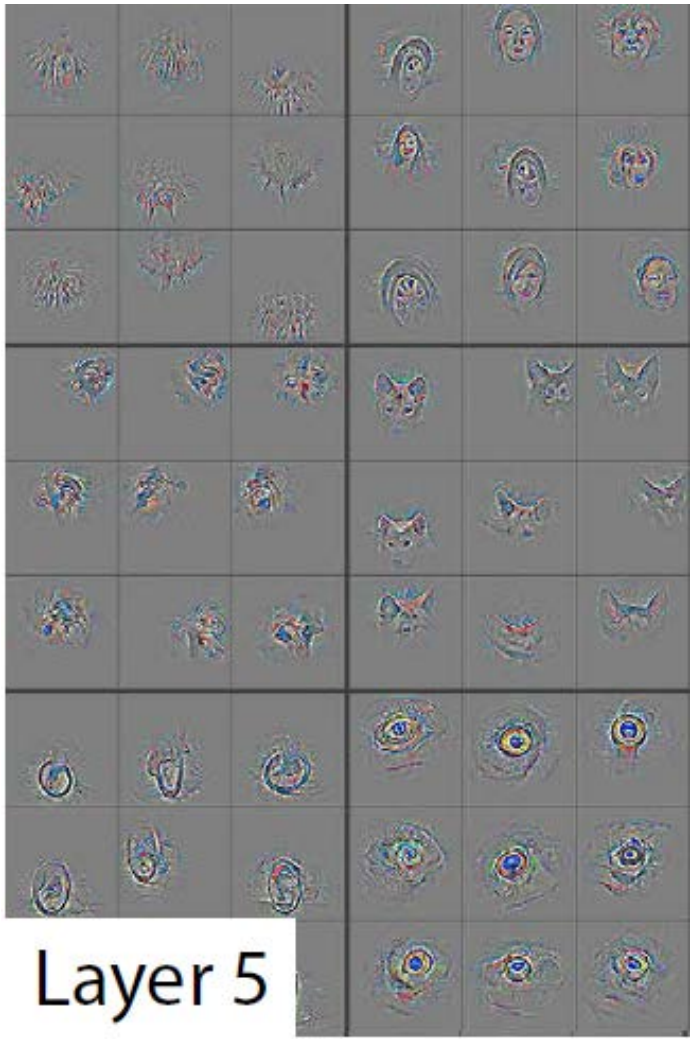


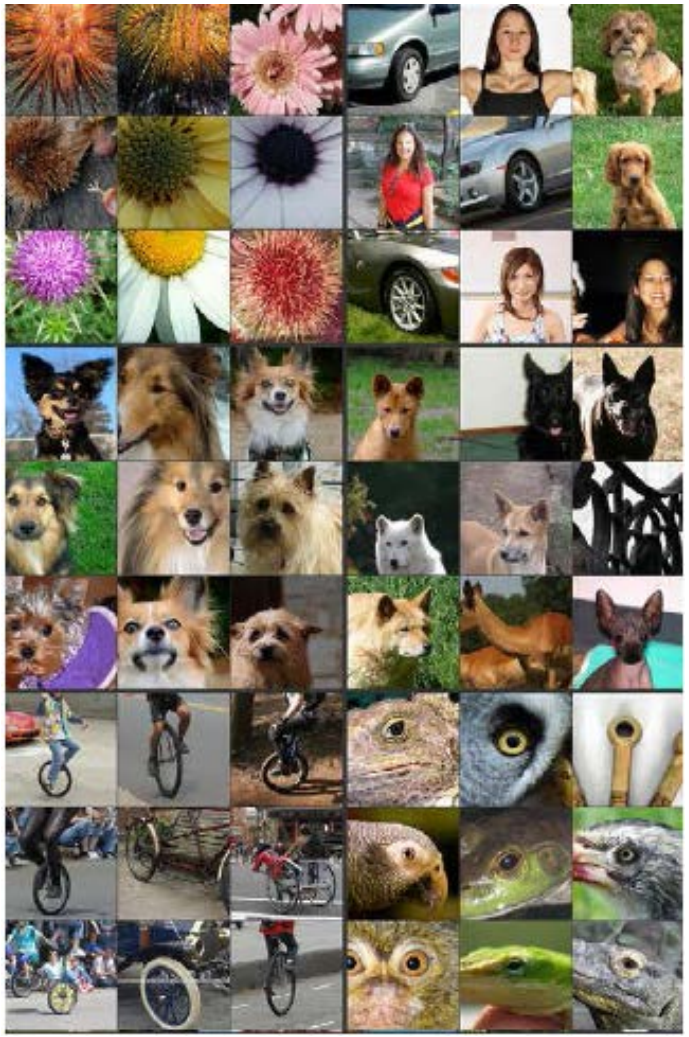



# Filter in Layer 4

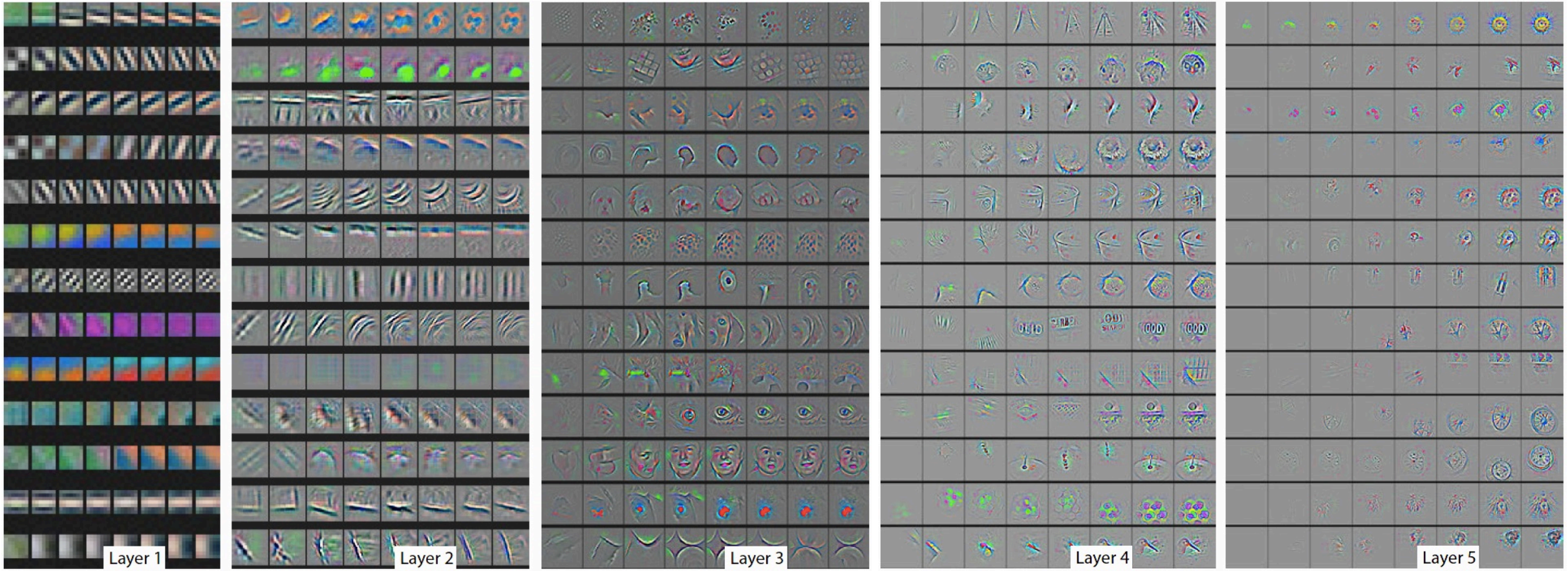


# Filter in Layer 5



# Entwicklung der Features beim Training



# Weight Sharing

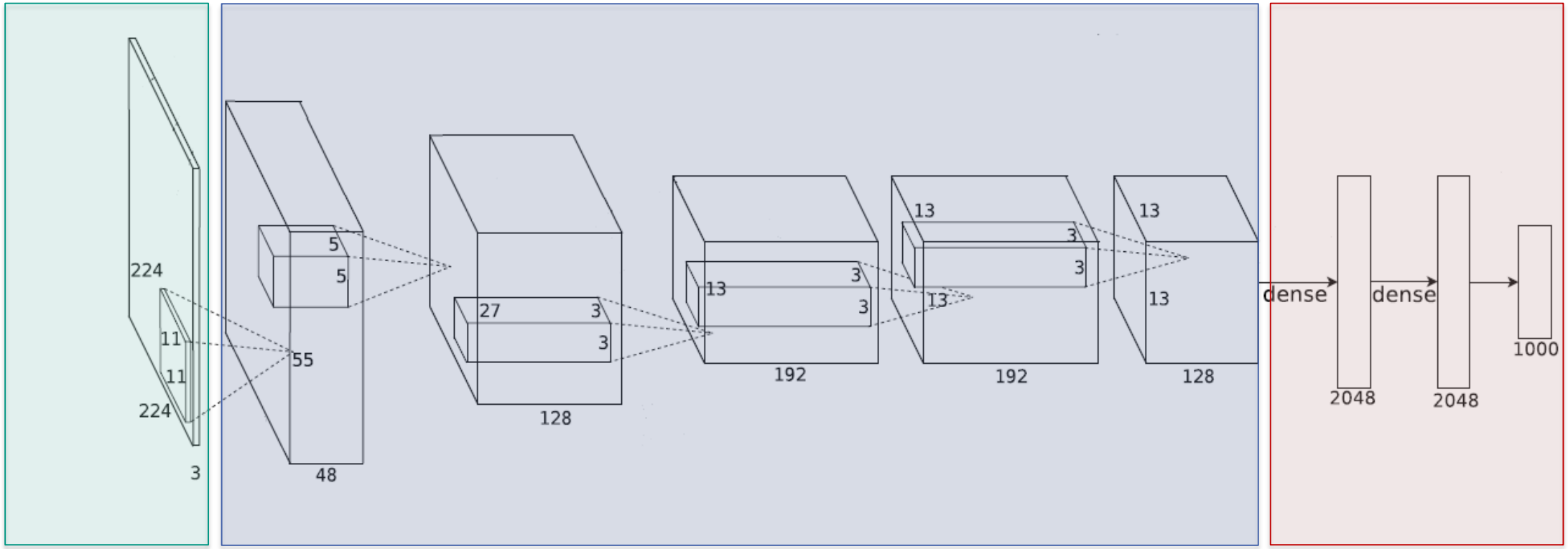
- Im klassischen Neuronalen Netz:
  - Jede Verbindung zwischen zwei Schichten enthält ein zu lernendes Gewicht
  - Bei Verarbeitung von Bildern
    - Ein Neuron verbunden zu jedem Bildpixel
    - schnell sehr viele Gewichte

- Im Convolutional Neural Network
  - Gewichte werden nur in Filtern gelernt
  - Lokale Wiederbenutzung von Gewichten
  - Hierdurch deutliche Reduzierung der zu lernenden Gewichte (Parameter)
    - Geringere Anzahl Lerndaten benötigt
    - Kürzere Trainingszeiten

# Initialisierung der Gewichte

- Zu Beginn des Trainings müssen Gewichte initialisiert werden
- Je nach Anwendung verschiedene Methoden:
  - Random Initialization
    - Gewichte initialisiert über Zufallsfunktion
  - Fixed Feature Extractor
    - Übernahme der Gewichte von trainiertem Netz
    - Fixieren der Gewichte der Feature Extraction Schichten
  - Fine-Tuning
    - Übernahme der Gewichte von trainiertem Netz
    - Geringe Lernrate für ausgewählte Schichten (Meist frühe Schichten)
  - Pretrained Initialization
    - Übernahme der Gewichte von trainiertem Netz lediglich zur Initialisierung
    - Normale Lernrate

# Wo stehen wir bis jetzt?

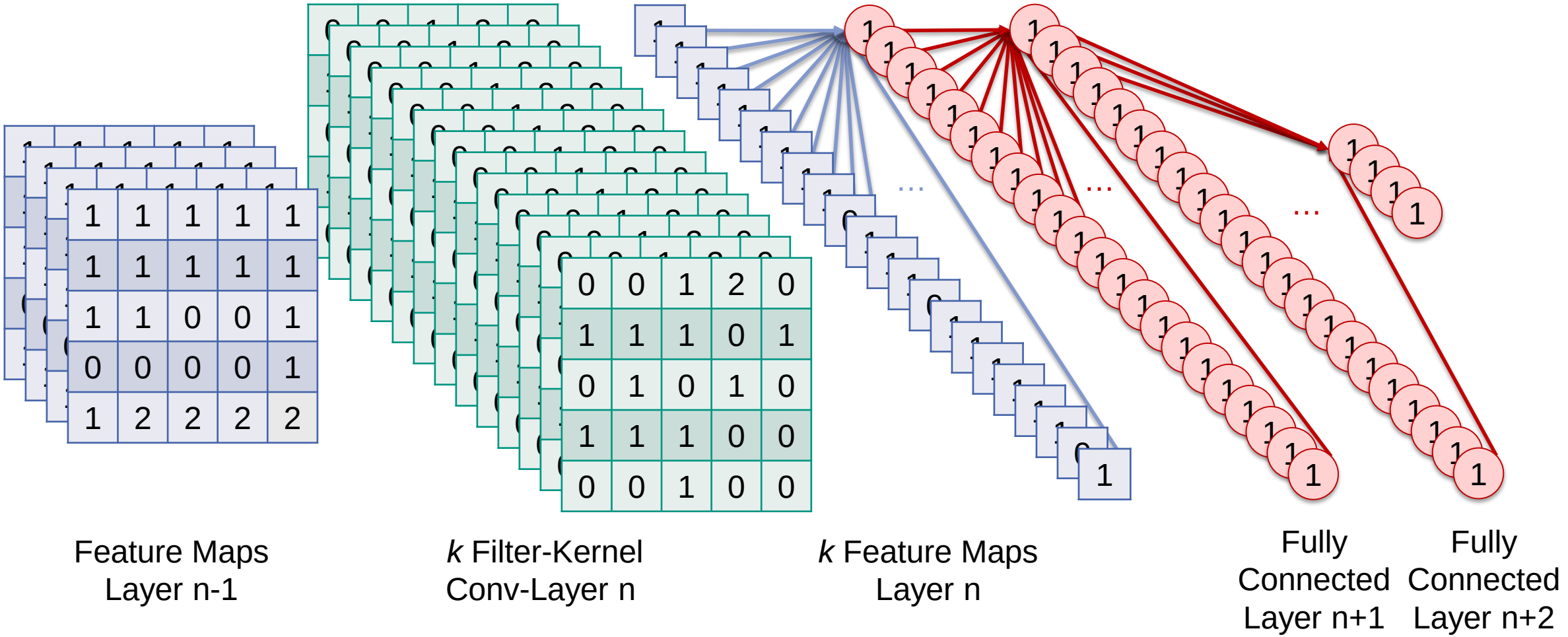



Eingabe

Feature Extraktion

Klassifikator

# Klassifikator (Fully Connected Network)



# Performance

- Bei Objektklassifikation CNNs derzeit führend:


- ImageNet ILSVRC 2012: Krizhevsky: 16,4% Error
  - 2. Platz: 26,1% Error
- ILSVRC 2013: 11,7% Error
  - 2. Platz: 13,0%
- ILSVRC 2014: Alle Top-Ergebnisse mit CNN
  - 1. Platz: 6,7%

- ~14 Millionen gelabelte Bilder, 20k classes
- Bilder aus dem Internet
- Von Menschen gelabelt via Amazon Turk
- Challenge: 1.2 MillionenTrainingsbilder, 1000 Klassen

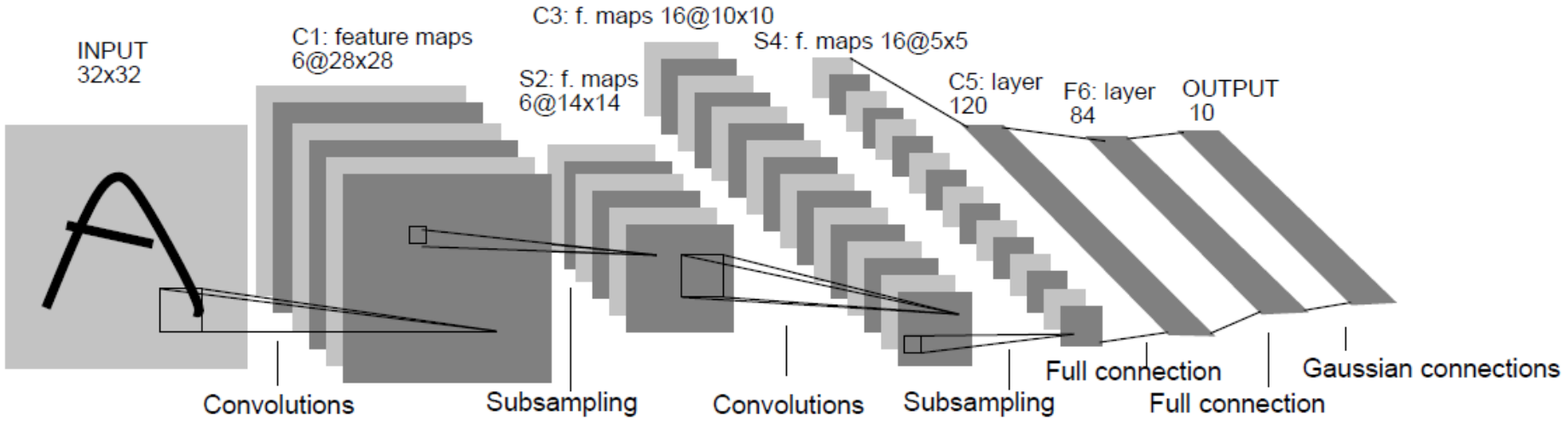
# MNIST

- Modified NIST Dataset
- Datensatz zur Handschrifterkennung
  - Eine Zahl pro Ausschnitt
  - 20x20 Pixel pro Ausschnitt
  - 60.000 Trainingsbilder
  - 10.000 Testbilder
- CNN Fehler bei MNIST:
  - 0,23%

# Warum jetzt diese Erfolge?

- Objekterkennung funktioniert erst ab einer gewissen Netzgröße “gut”
- Rechenkapazität zum Training großer Netze durch Nutzung der GPU verfügbar
  - Verfügbare Rechenkapazität wächst ständig weiter
- Immer größere Datensätze verfügbar
  - In Kombination mit ausreichender Rechenkapazität

# Architekturanalyse: LeNet

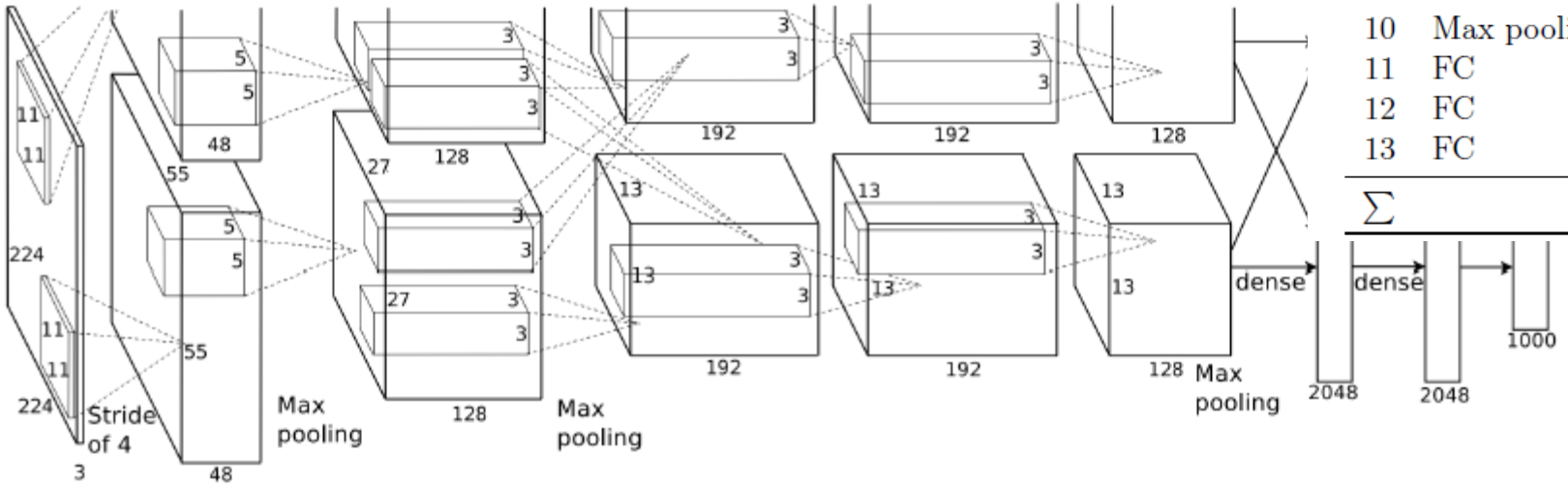



| # | Type | Filters @ Patch size / stride | Parameters |
|---|---|---|---|
| | Input | | 0 |
| 1 | Convolution | $6 @ 5 \times 5 \times 1 / 1$ | 156 |
| 2 | Scaled average pooling | $2 \times 2 / 2$ | 2 |
| 3 | Convolution | $16 @ 5 \times 5 \times 6 / 1$ | 2 416 |
| 4 | Scaled average pooling | $2 \times 2 / 2$ | 2 |
| 5 | Fully Connected | 120 neurons | **48 120** |
| 6 | Fully Connected | 84 neurons | 10 164 |
| 7 | Fully Connected (output) | 10 neurons | 850 |
| $\sum$ | | | 61 710 |

Quelle: Martin Thoma

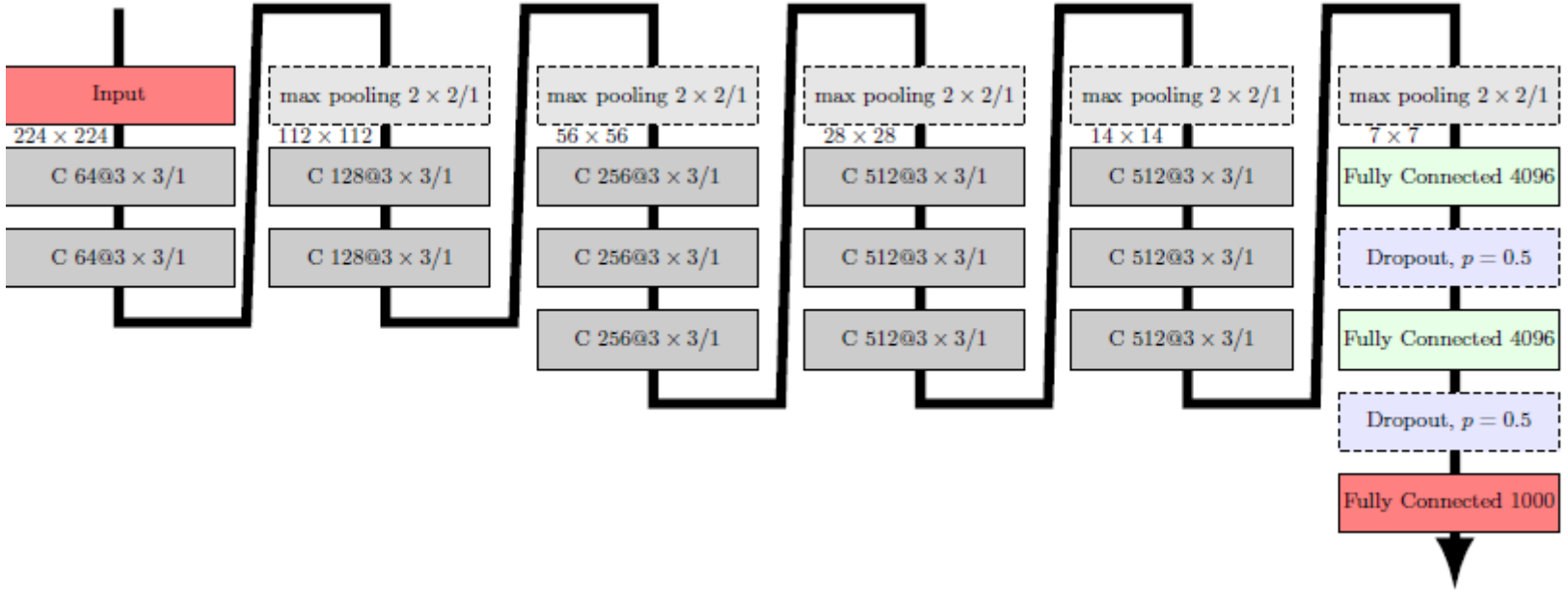
# Architekturanalyse: AlexNet

| # | Type | Filters @ Patch size / stride | Parameters |
|---|---|---|---|
| | Input | | |
| 1 | Convolution | 96 @ 11 × 11 × 3 / 4 | 34 944 |
| | LCN | | |
| 2 | Max pooling | 3 × 3 / 2 | 0 |
| 3 | Convolution | 256 @ 5 × 5 × 48 / 1 | 307 456 |
| | LCN | | |
| 4 | Max pooling | 3 × 3 / 2 | 0 |
| 5 | Convolution | 384 @ 3 × 3 × 256 / 1 | 885 120 |
| 7 | Convolution | 384 @ 3 × 3 × 192 / 1 | 663 936 |
| 9 | Convolution | 256 @ 3 × 3 × 192 / 1 | 442 624 |
| 10 | Max pooling | 3 × 3 / 2 | 0 |
| 11 | FC | 4096 neurons | **37 752 832** |
| 12 | FC | 4096 neurons | 16 781 312 |
| 13 | FC | 1000 neurons | 4 097 000 |
| ∑ | | | 60 965 224 |




Quelle: Martin Thoma

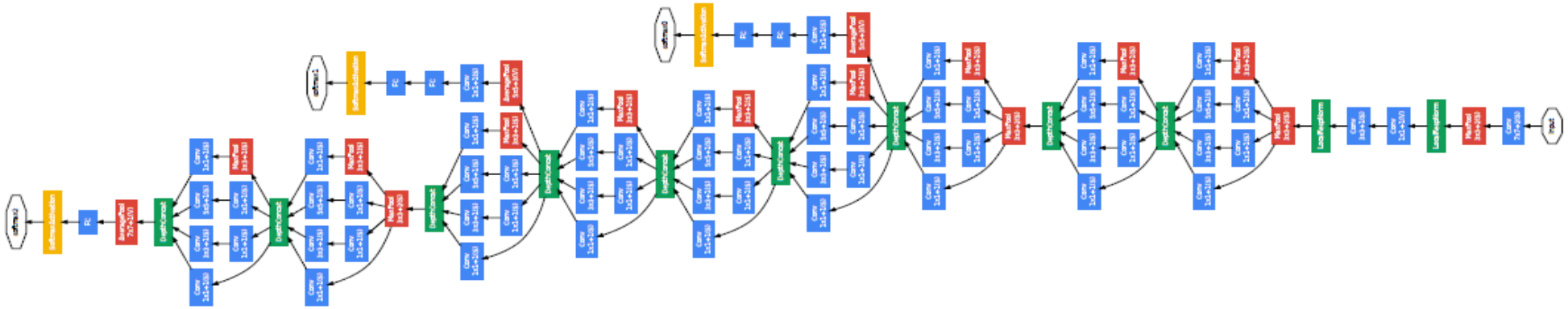
# Architekturanalyse: VGG-16

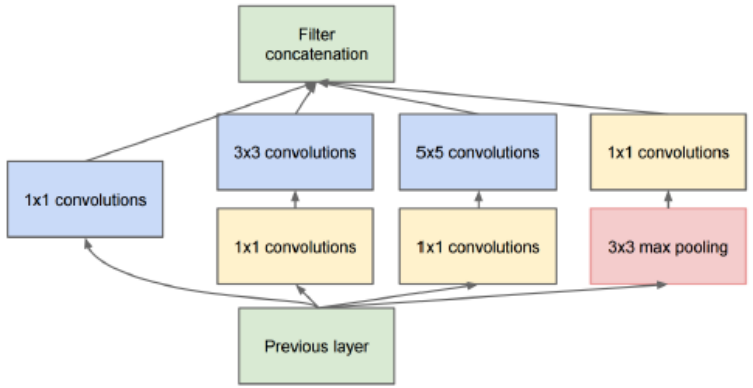



| # | Type | Filters @ Patch size / stride | Parameters |
|---|---|---|---|
| | Input | | |
| 1 | Convolution | 64 @ 3 × 3 × 3 / 1 | 1 792 |
| 2 | Convolution | 64 @ 3 × 3 × 64 / 1 | 36 928 |
| | Max pooling | 2 × 2 / 2 | 0 |
| 3 | Convolution | 128 @ 3 × 3 × 64 / 1 | 73 856 |
| 4 | Convolution | 128 @ 3 × 3 × 128 / 1 | 147 584 |
| | Max pooling | 2 × 2 / 2 | 0 |
| 5 | Convolution | 256 @ 3 × 3 × 128 / 1 | 295 168 |
| 6 | Convolution | 256 @ 3 × 3 × 256 / 1 | 590 080 |
| 7 | Convolution | 256 @ 3 × 3 × 256 / 1 | 590 080 |
| | Max pooling | 2 × 2 / 2 | 0 |
| 8 | Convolution | 512 @ 3 × 3 × 256 / 1 | 1 180 160 |
| 9 | Convolution | 512 @ 3 × 3 × 512 / 1 | 2 359 808 |
| 10 | Convolution | 512 @ 3 × 3 × 512 / 1 | 2 359 808 |
| | Max pooling | 2 × 2 / 2 | 0 |
| 11 | Convolution | 512 @ 3 × 3 × 512 / 1 | 2 359 808 |
| 12 | Convolution | 512 @ 3 × 3 × 512 / 1 | 2 359 808 |
| 13 | Convolution | 512 @ 3 × 3 × 512 / 1 | 2 359 808 |
| | Max pooling | 2 × 2 / 2 | 0 |
| 14 | FC | 4096 neurons | **102 764 544** |
| | Dropout | | 0 |
| 15 | FC | 4096 neurons | 16 781 312 |
| | Dropout | | 0 |
| 16 | FC | 1000 neurons | 4 097 000 |
| $\sum$ | | | 138 357 544 |

Quelle: Martin Thoma

# Architekturanalyse: GoogleNet



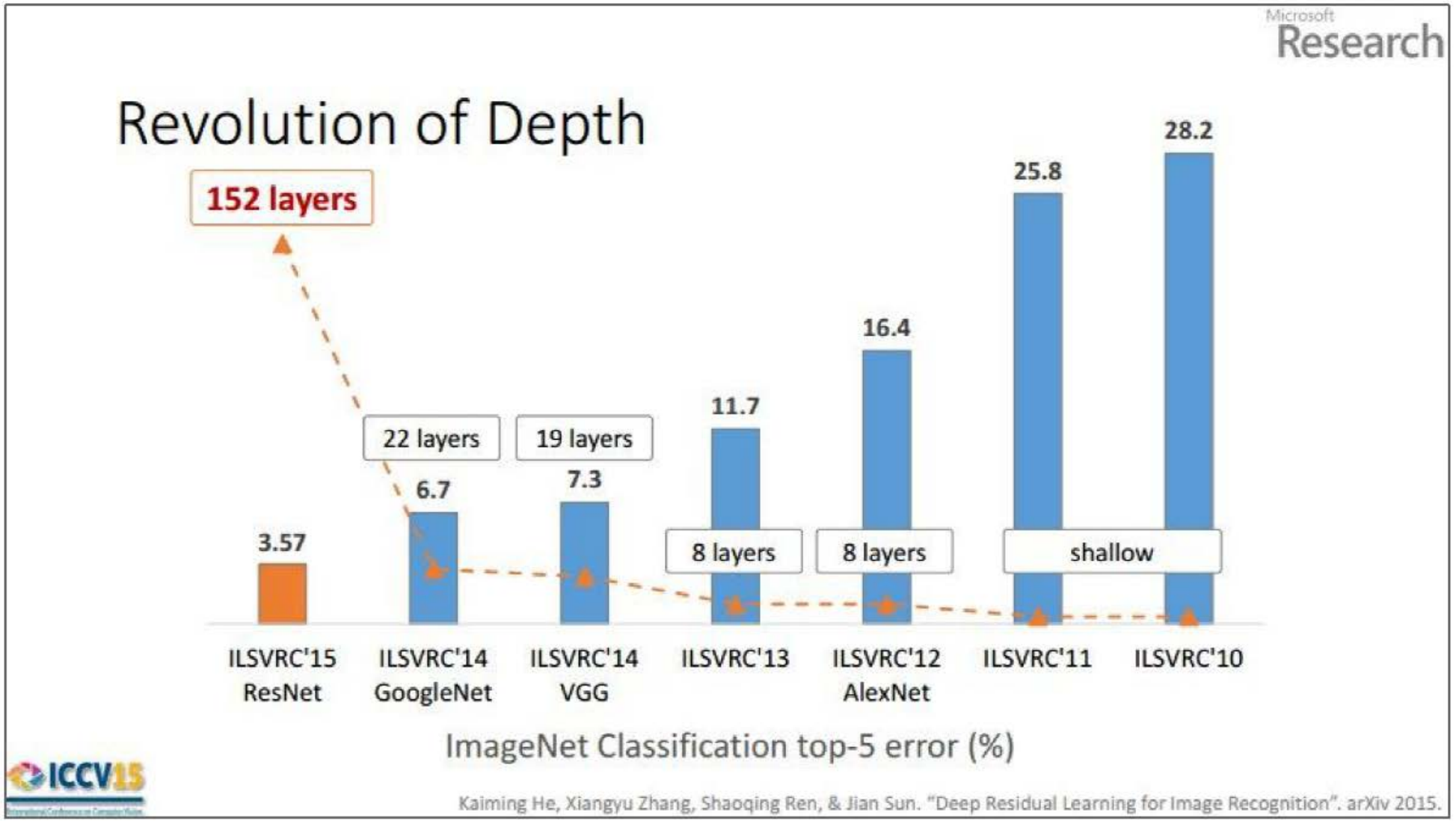



GoogLeNet [Google 2014]

| # | × | Type | Parameters | Output size |
|---|---|---|---|---|
| | | Input | | 3 @ 299 × 299 |
| 1 | | Stem | 605 728 | 384 @ 35 × 35 |
| 2 | 4× | Inception A | 317 632 | 384 @ 35 × 35 |
| 3 | | Reduction A | 2 306 112 | 1024 @ 17 × 17 |
| 4 | 7× | Inception B | 2 936 256 | 1024 @ 17 × 17 |
| 5 | | Reduction B | 2 747 392 | 1536 @ 8 × 8 |
| 6 | 3× | Inception C | 4 553 088 | 1536 @ 8 × 8 |
| | | Global Average Pooling | 0 | 1536 @ 1 × 1 |
| | | Dropout (p=0.8) | 0 | 1536 @ 1 × 1 |
| 7 | | Softmax | 1 537 000 | 1000 |
| $\sum$ | | | 42 679 816 | |

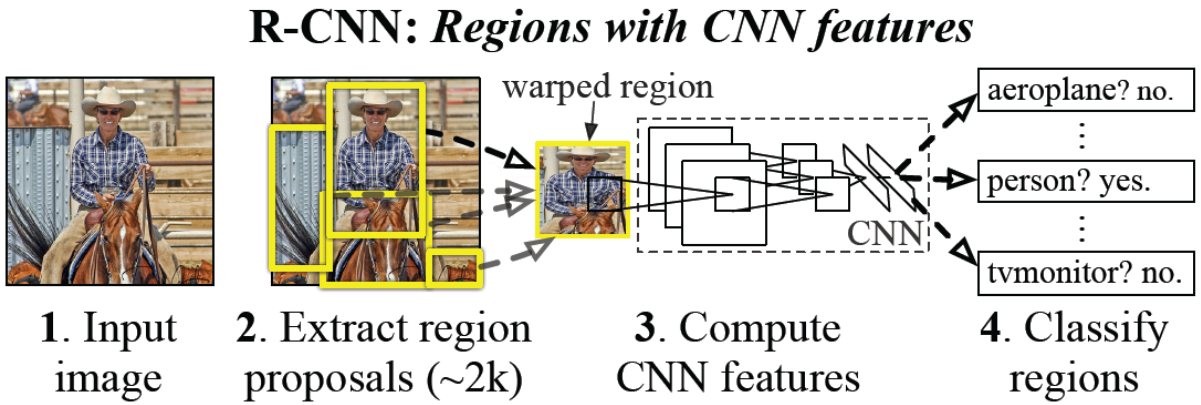
Quelle: Martin Thoma

# Revolution of Depth

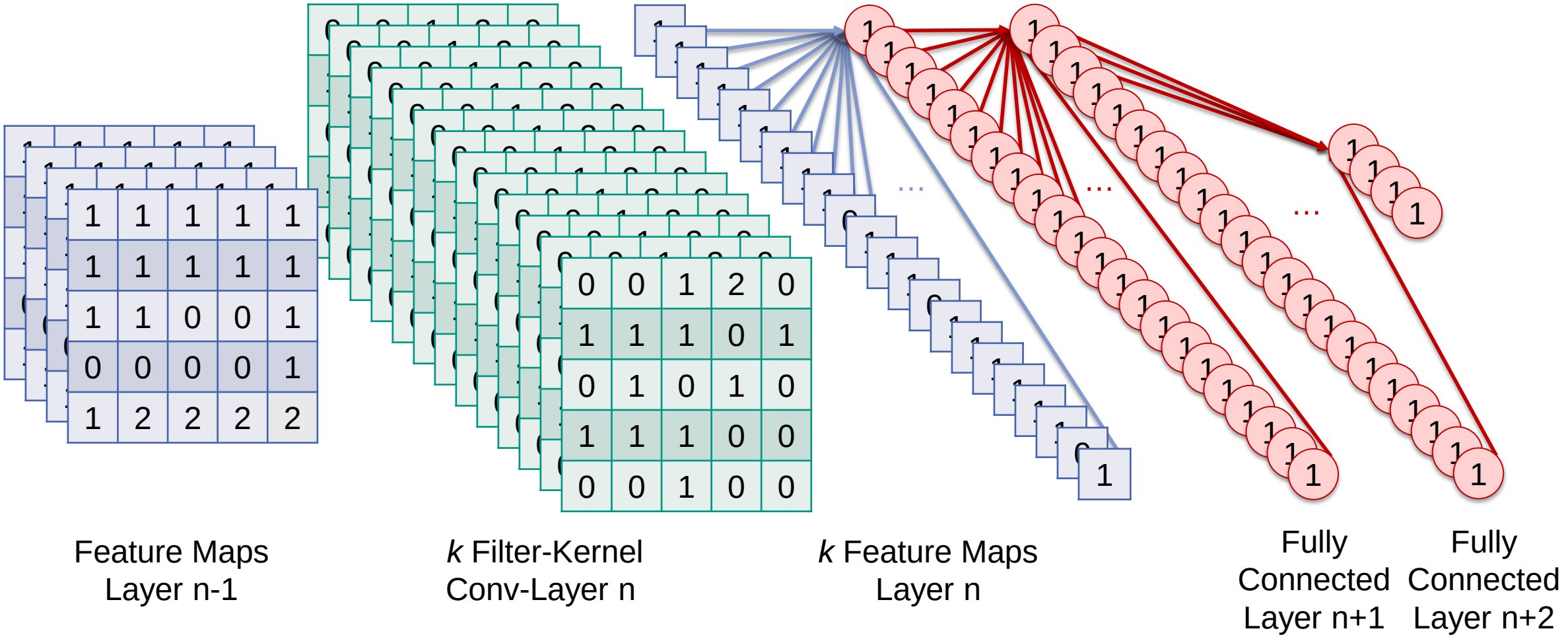


# Objektdetektion mit CNNs

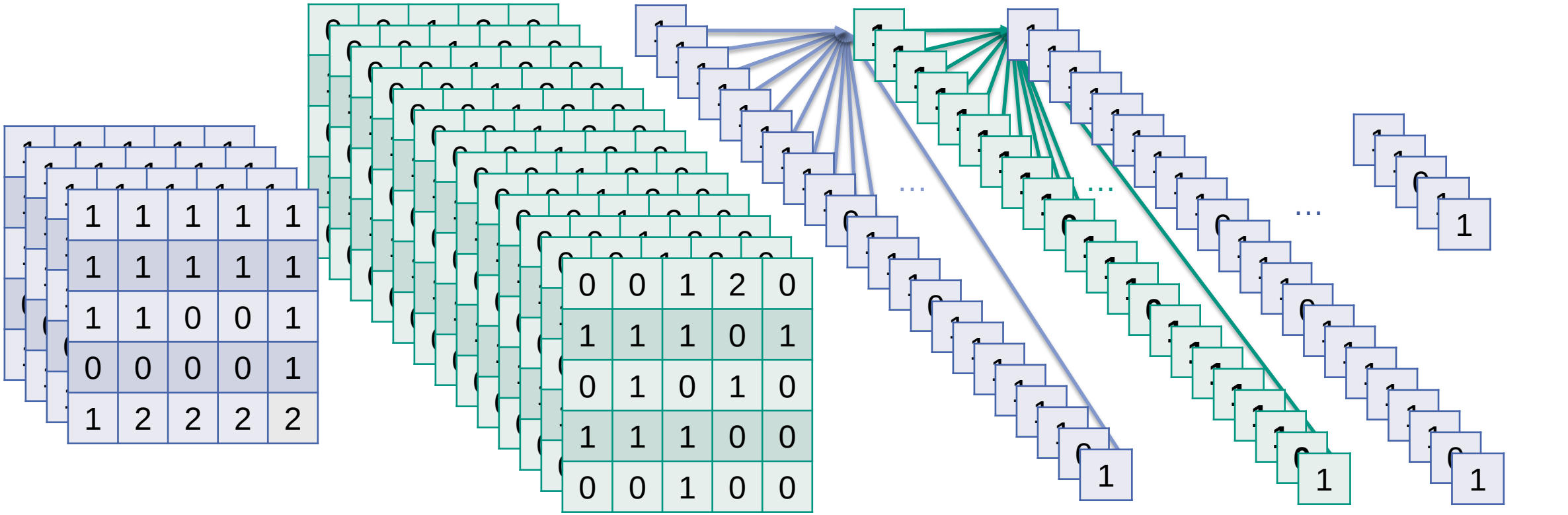
- Bisher behandelt: Klassifikationsnetze
- Naheliegender Ansatz zur Detektion
  - Auswahl von Regionen
  - Klassifikation der Regionen durch Klassifikationsnetze
- Möglichkeiten zur Regionsauswahl
  - Sliding Window
    - Mehrere Mio Regionen je nach Bildgröße
  - Interest Point Detektor (Region Proposals)
    - Meist mehrere 100 bis 1000 Regionen
- Nachteile
  - Eine Netzausführung pro Region
  - Zeitintensiv



# Warum nicht Detektion im Netz?

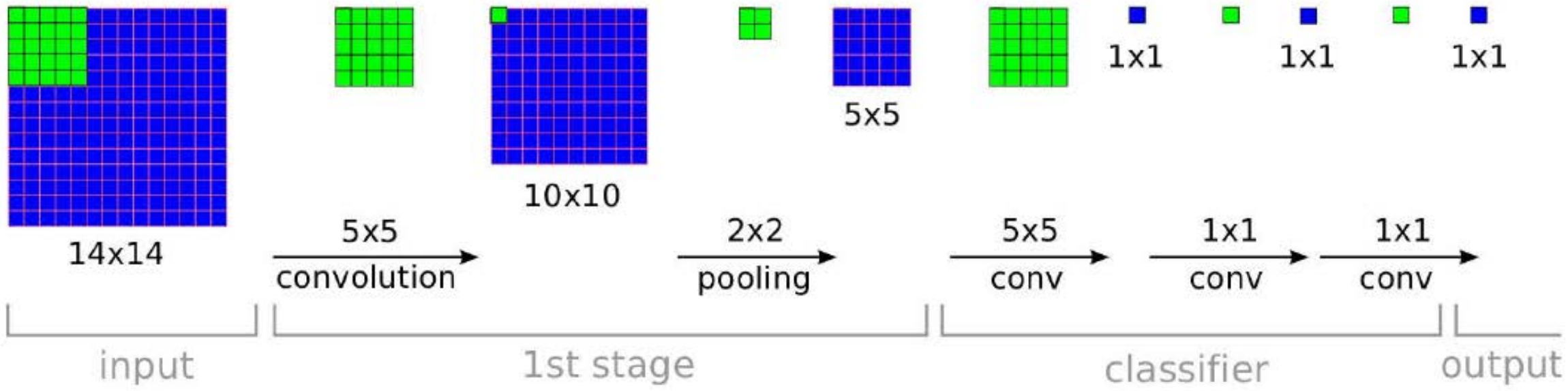


# Warum nicht Detektion im Netz?



# Fully Convolutional Networks (FCN)

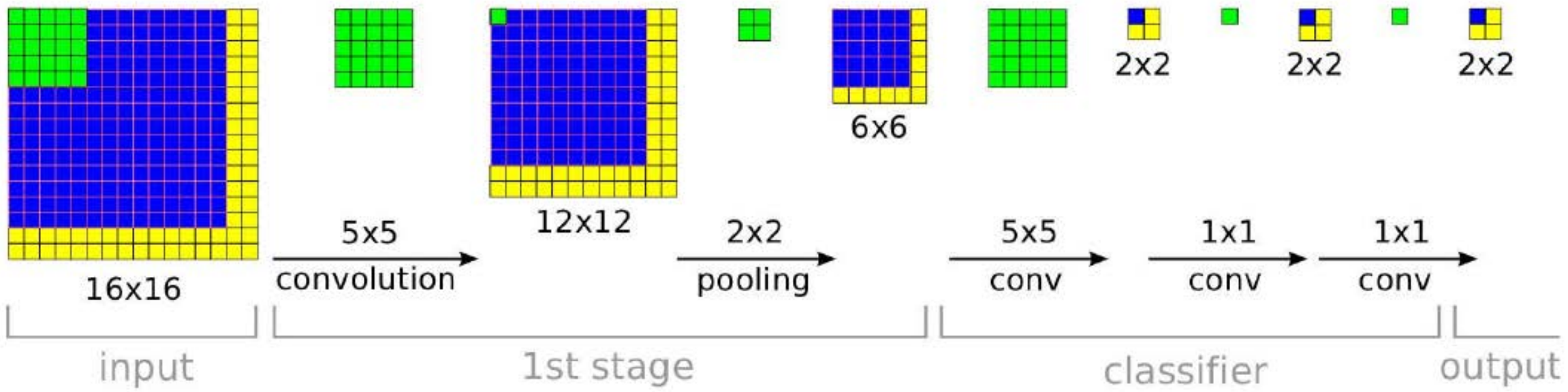
- Konvertiertierung der Fully Connected Schichten in Convolutional Schichten
- Netze äquivalent bei gleicher Eingabegröße
- Im Gegensatz zu Fully Connected CNNs Eingabegröße nicht fix
  - Größere Eingabebilder möglich




Bildquelle: Sermanet et al.: OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks

# Fully Convolutional Networks (FCN)

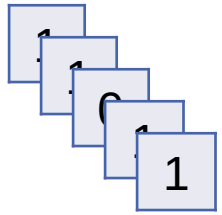
- Konvertiertierung der Fully Connected Schichten in Convolutional Schichten
- Netze äquivalent bei gleicher Eingabegröße
- Im Gegensatz zu Fully Connected CNNs Eingabegröße nicht fix
  - Größere Eingabebilder möglich



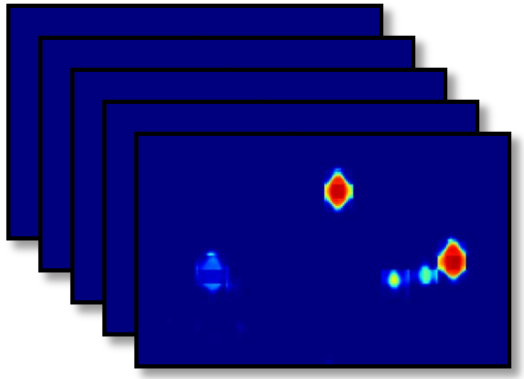

Bildquelle: Sermanet et al.: OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks

# Ausgabe eines FCN

- Ausgabe Fully Connected CNN (Klassifikationsnetz)
  - Ein Wahrscheinlichkeitswert pro Klasse
  - Klassifikation ≡ Wahrscheinlichste Klasse

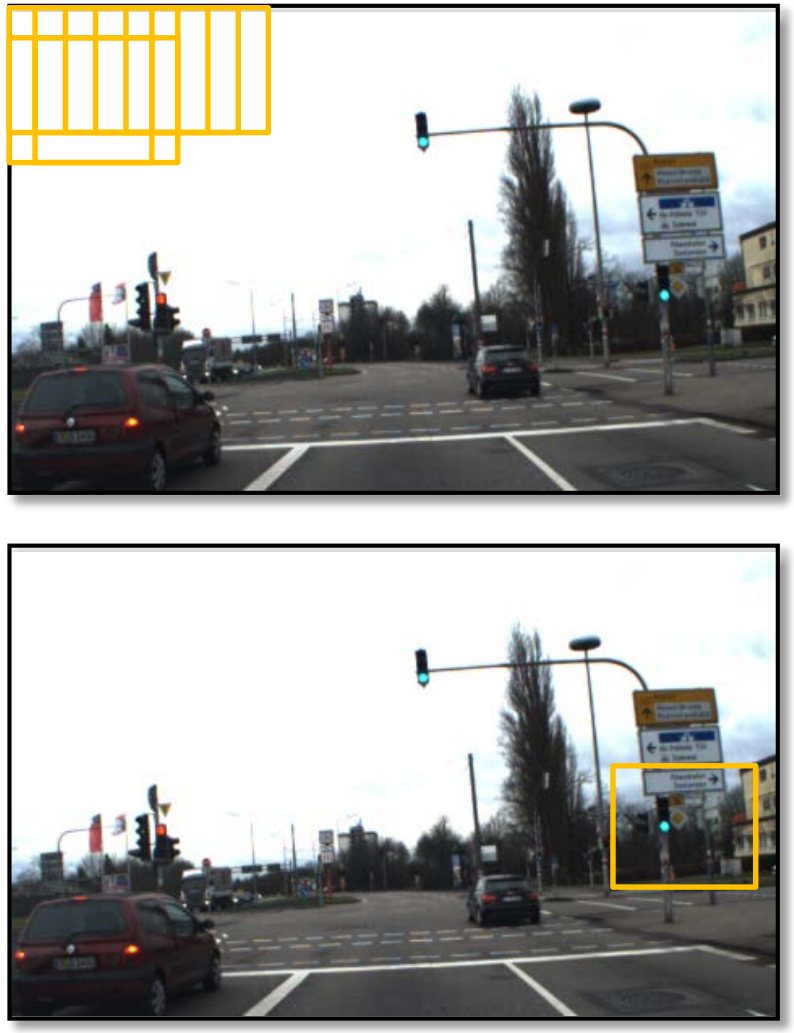



- Ausgabe Fully Convolutional Network
  - Eine Wahrscheinlichkeitskarte pro Klasse
  - Gibt die Wahrscheinlichkeit dafür an, dass ein Bereich/Pixel im Bild zur entsprechenden Klasse gehört
  - Größe eines Bereichs abhängig von der Netzarchitektur
    - Im Maximalfall pixelgenau
    - Meist Regionen von ca. 4 x 4 Pixeln



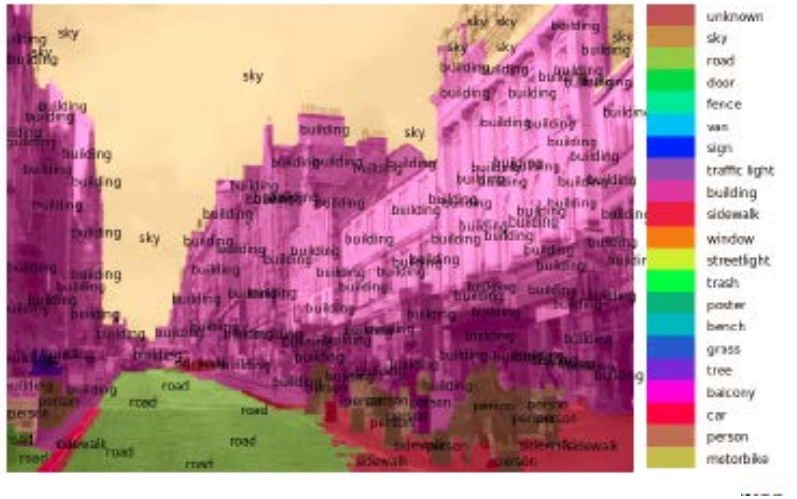
# Implizites Sliding Window

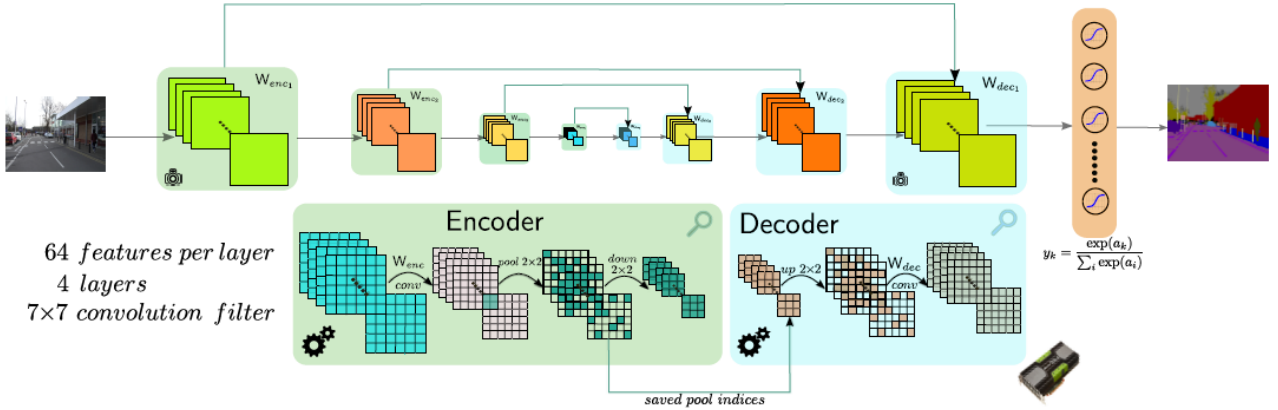
- Klassifikation jeder Region anhand eines größeren Field of View
  - Größe des Field of View abhängig von der Netzarchitektur
- Einzelne Fields of View entsprechen implizit einem Sliding Window Ansatz
  - Allerdings keine explizite Berechnung
  - Berechnung implizit durch Anwendung von Faltungen & Pooling
  - Optimale Berechnung
    - Jede Operation wird pro Bildbereich nur einfach ausgeführt



# Segmentierung

- Mit diesen Informationen:
  - Segmentierung des Bildes möglich
- Vorgehen
  - Pro Region / Pixel Übernahme der Klasse mit max. Wahrscheinlichkeit
  - Quasi Maximum über die Wahrscheinlichkeitskarten aller Klassen
  - Bei entsprechender Netz-
  - architektur pixelgenaue
  - Segmentierung möglich




Badrinarayanan et al., SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling

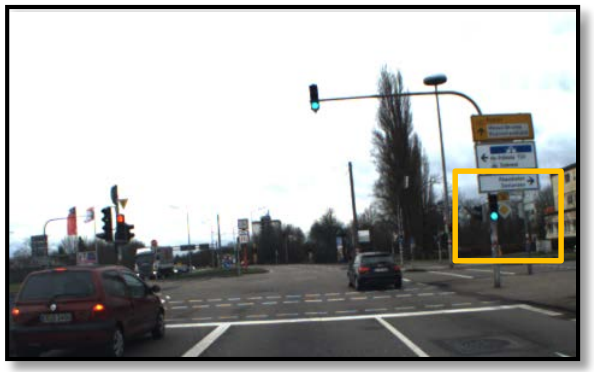
# Objektdetektion in CNNs

- Können damit Objekte mit einem CNN detektiert werden?
  - Noch nicht ganz
  - Durch Segmentierung allein können keine Objektinstanzen unterschieden werden
  - Beispiel:
    - Parkende Autos am Straßenrand
    - Netz kann lediglich bestimmen, dass der Bereich der Klasse Auto zuzuordnen ist
    - Detektion einzelner Autos nicht möglich
    - Somit Erkennung von z. B. Parklücken schwierig

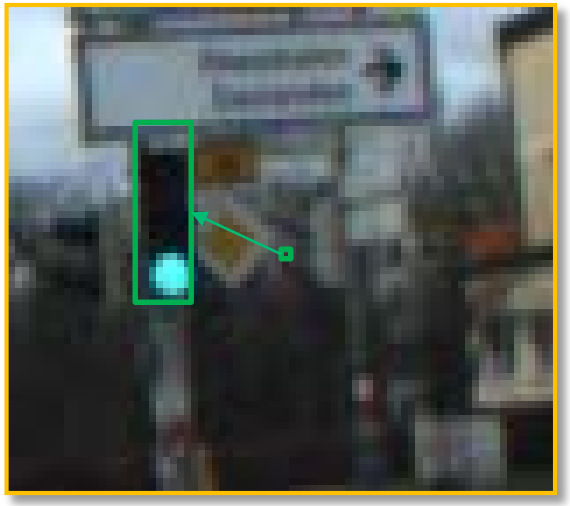

Bildquelle: semanticscholar.org

# Bounding Box Regression

- Bisher
  - Ausgabe einer Wahrscheinlichkeit pro Klasse für jede Region / Pixel
  - Segmentierung anhand der Wahrscheinlichkeitskarten

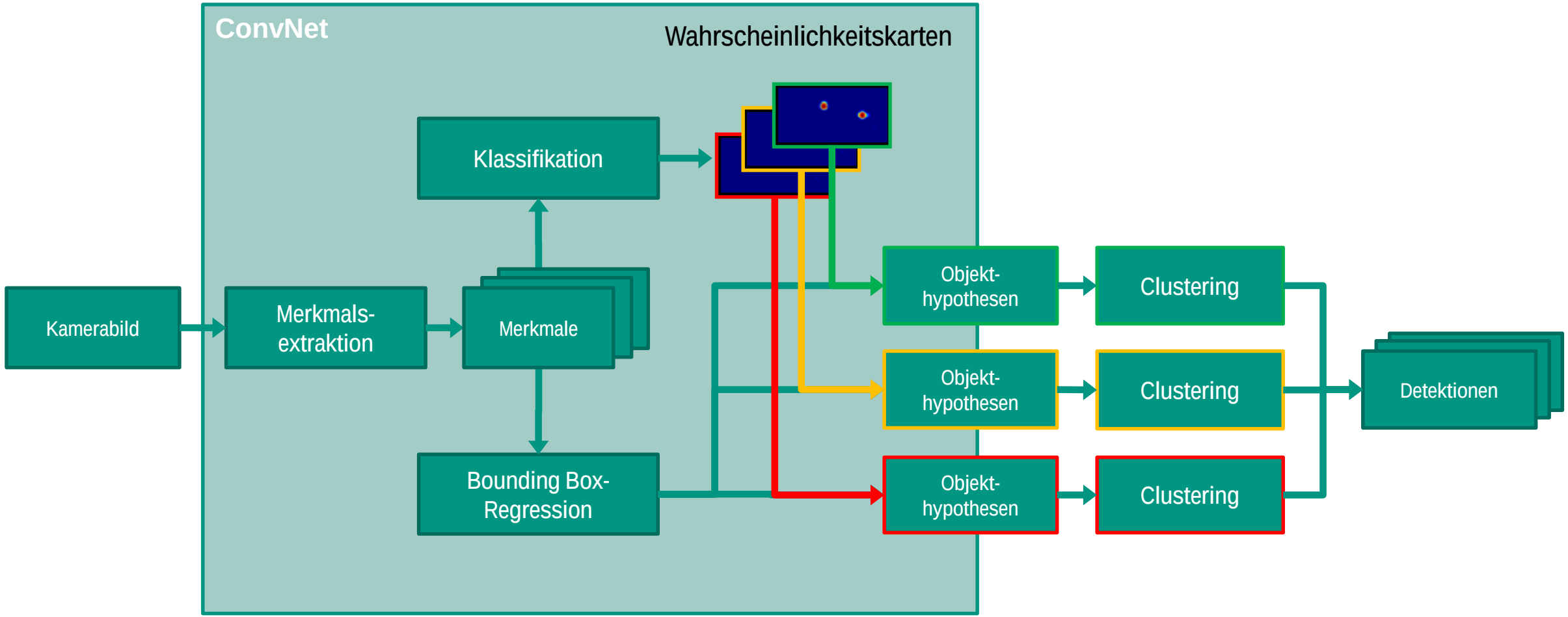


- Jetzt [Overfeat]
  - Zusätzlich Vorhersage einer Objekt-Bounding Box pro Region
  - Bounding Box ist klassenunabhängig
  - Anhand der extrahierten Features
  - Vorhergesagte Bounding Box im Field of View der entsprechenden Region



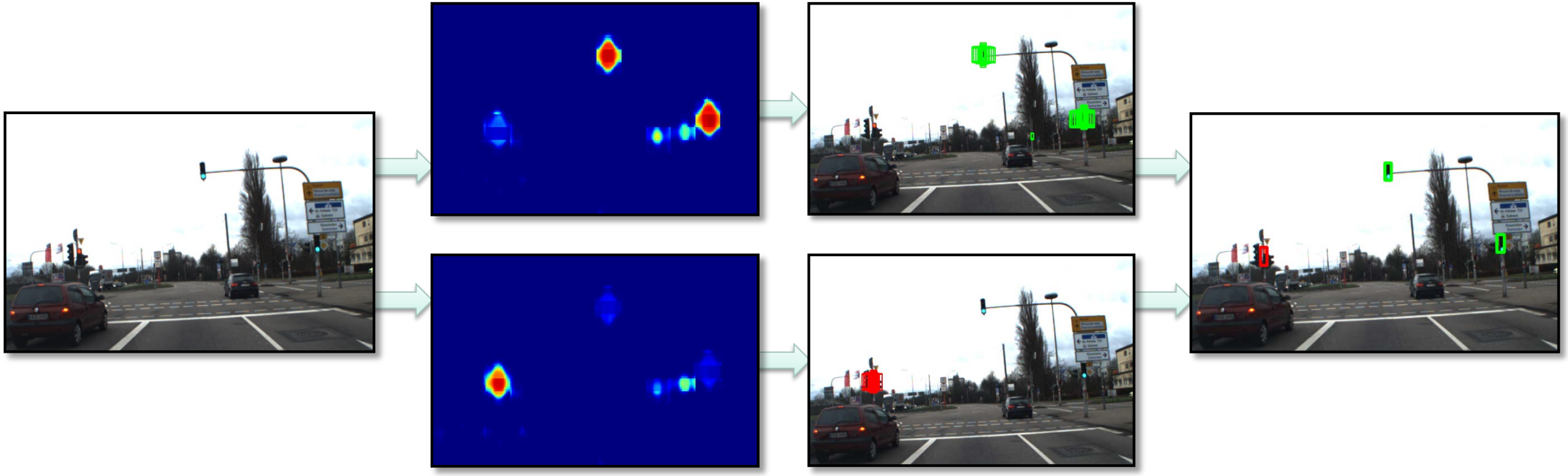
[Overfeat]: Sermanet et al.: OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks

# Beispiel: DeepTLR




Weber, Wolf, Zöllner, 2016: DeepTLR: A single Deep Convolutional Network for Detection and Classification of Traffic Lights

# Beispiel: DeepTLR



(a) Input

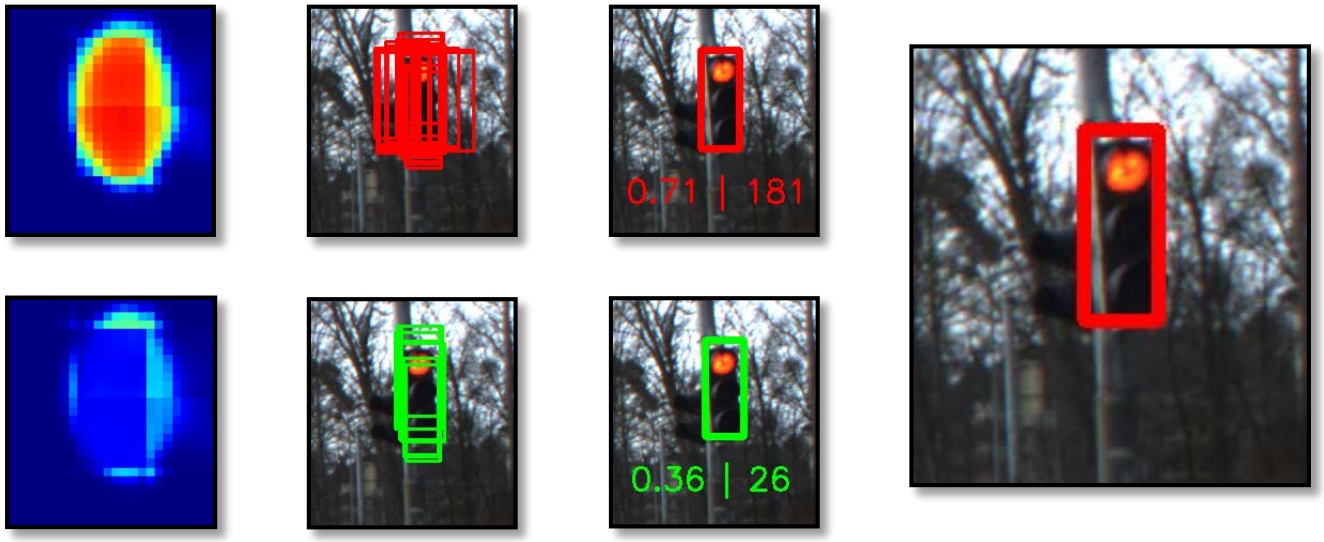
(b) Class Probability Estimation

(c) Bounding Box Regression

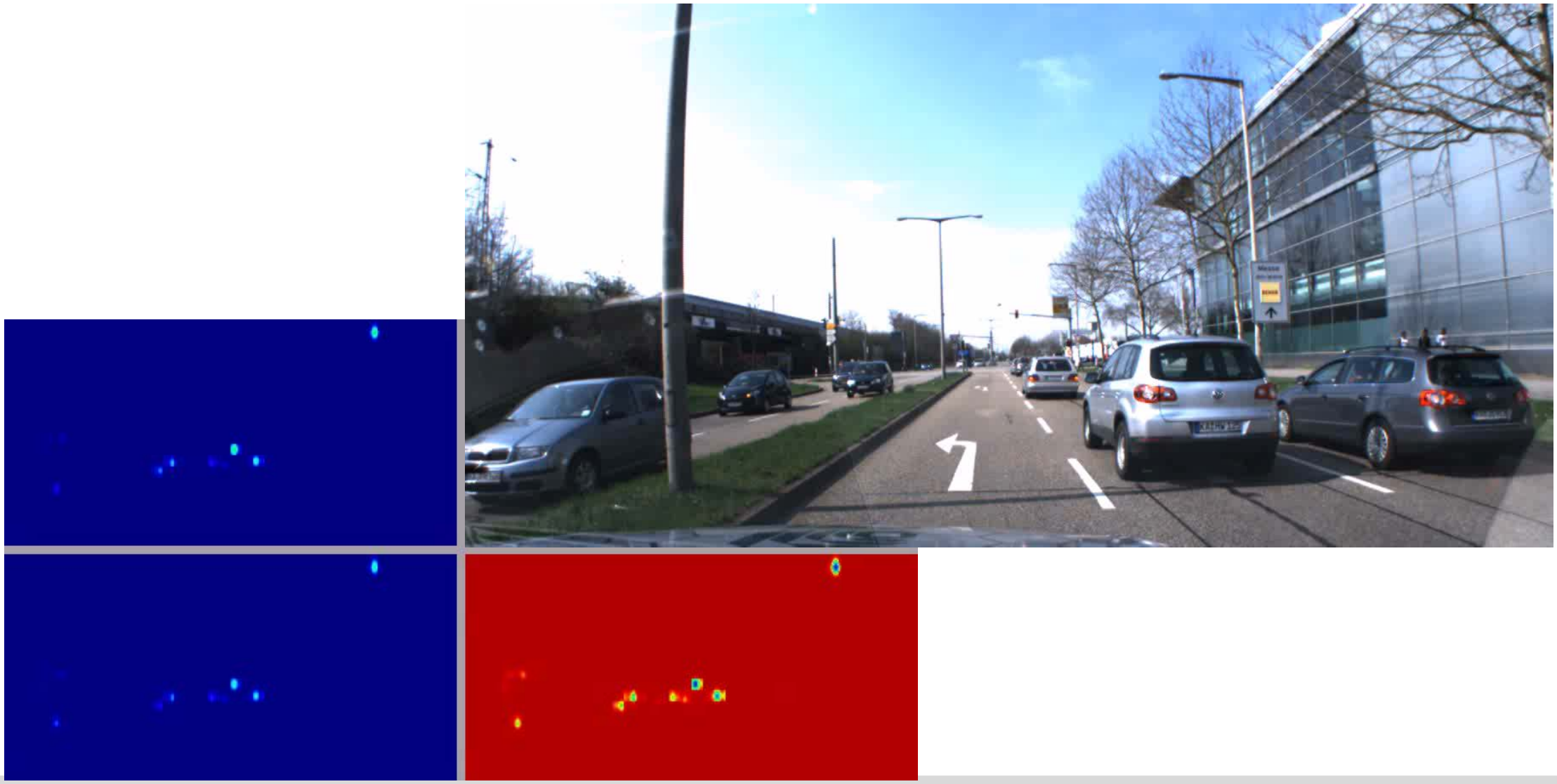
(d) Final Detections

# Auflösung überdeckender Detektionen

- Überdeckende Detektionen grundsätzlich möglich
  - Region kann Bounding Box an beliebiger Stelle im Bild vorhersagen
- Auflösung
  - Konfidenz aus Wahrscheinlichkeitskarten für jeden Kandidaten
  - Finale Detektion erhält kombinierte Konfidenzwerte



# DeepTLR Video



# Multi-Purpose Netzwerke

Erfüllung mehrerer Aufgaben in einem Netz

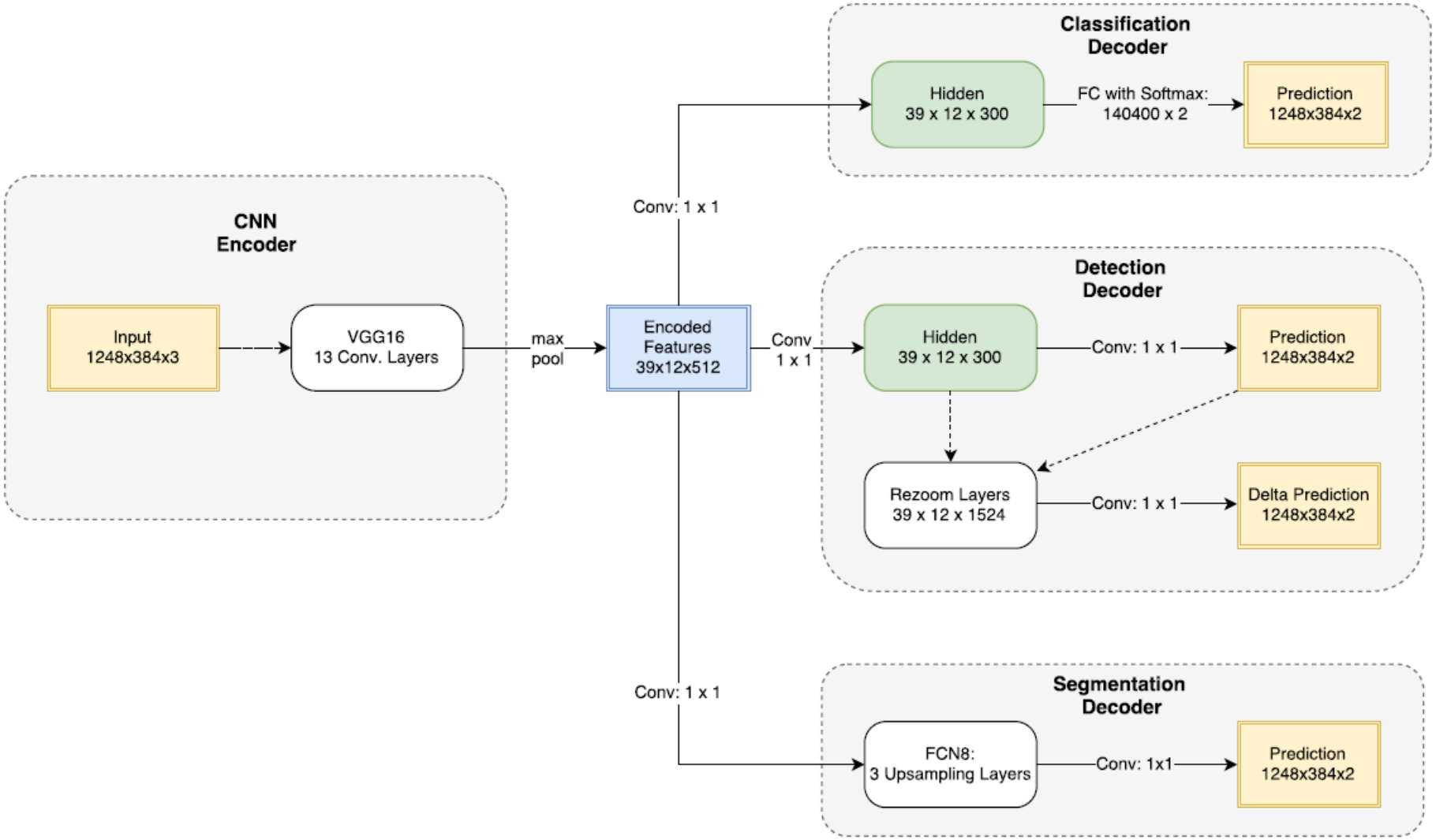
- Motivation
  - Ein Netz pro Aufgabe ressourcenintensiv
  - Wieso nicht Berechnungen wie Feature Extraktion teilen?

→ MultiNet

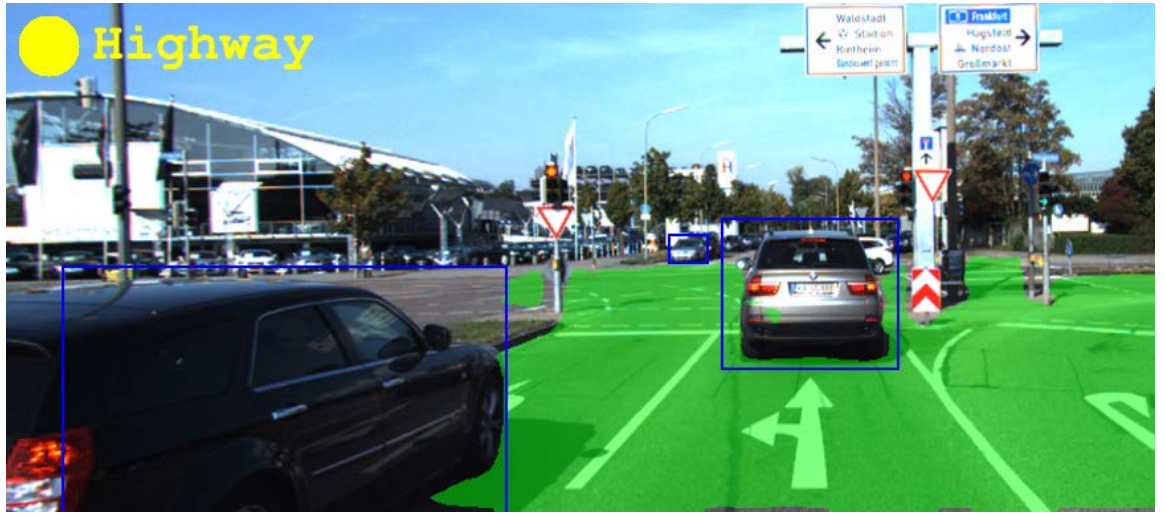
- Fähigkeiten:
  - Fahrzeugdetektion
  - Straßensegmentierung
  - Klassifikation des Straßentyps


Teichmann, Weber, Zöllner, Cipolla, Urtasun: “MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving“ arXiv preprint

# MultiNet Architektur



# Ergebnisse

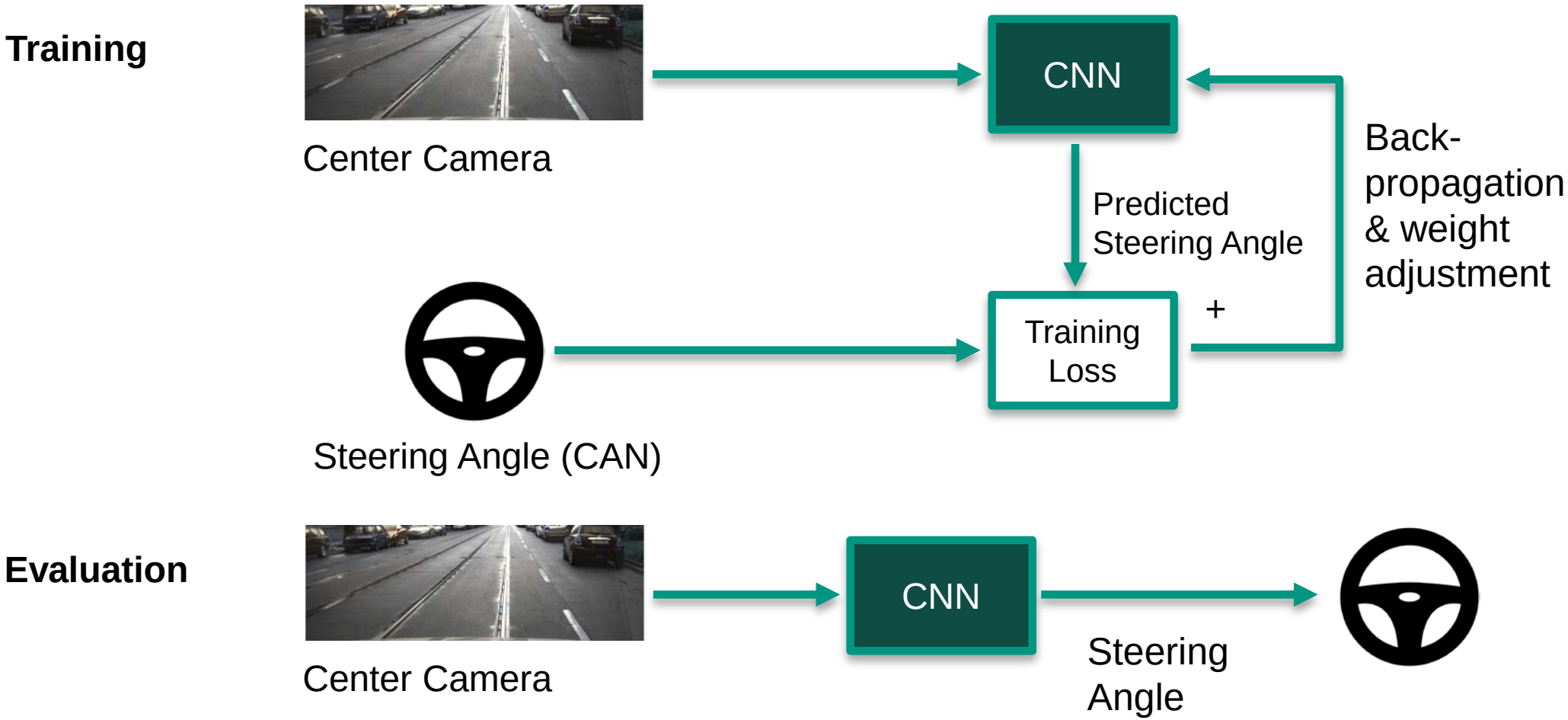





# Anwendung: End to End Steering

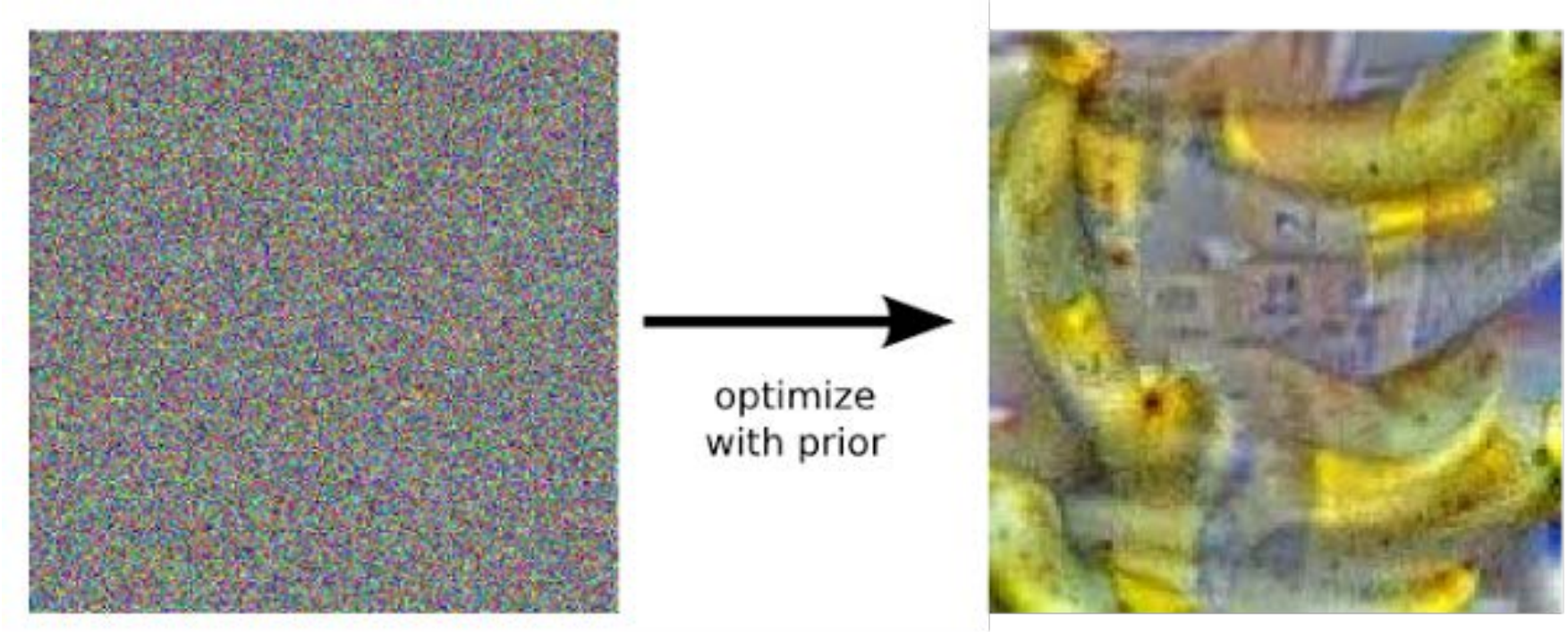
- Anwendung
  - Steuern eines Fahrzeugs in realer Umgebung
- Technik
  - Convolutional Neural Networks
- Lernverfahren
  - Imitation Learning (überwachtes Lernen)
- Daten
  - Kamerabild
  - Realer Lenkwinkel eines menschlichen Fahrers
  - Ca. 4 Stunden Trainingsdaten (ca. 300K Bilder + Augmentation)
- Geschwindigkeit
  - CNN forward pass : Ca. 90 Hz auf GPU (GTX 970)

# Model Setup


Training
Center Camera
CNN
Predicted
Steering Angle
Back-
propagation
& weight
adjustment
+
Training
Loss
Steering Angle (CAN)
Evaluation
Center Camera
CNN
Steering
Angle

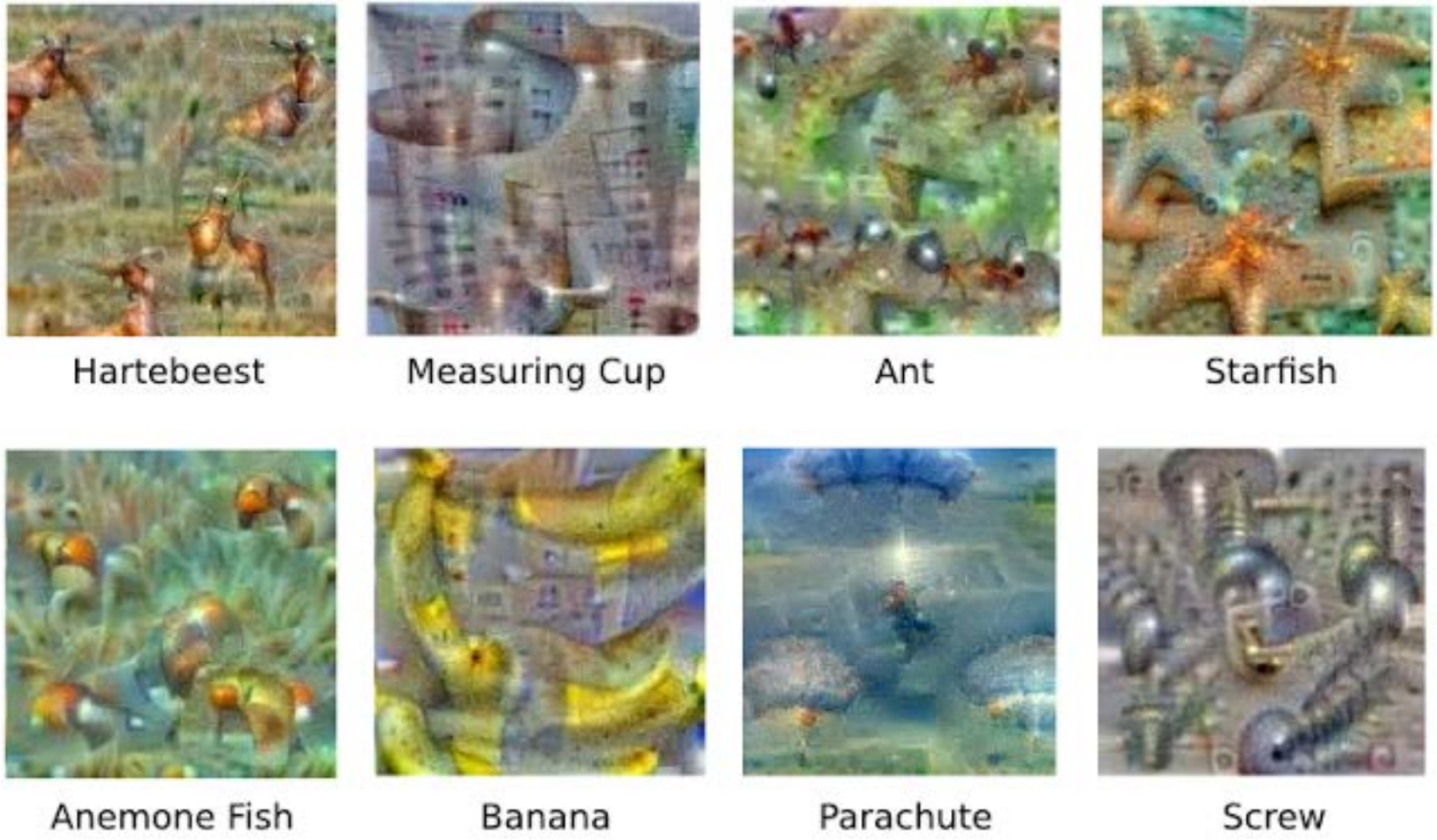
# Weitere Anwendungen: Google DeepDream

- Umkehrung eines trainierten Netzes
- Eingabe eines verrauschten Bildes




http://googleresearch.blogspot.de/2015/06/inceptionism-going-deeper-into-neural.html

# Weitere Anwendungen: Google DeepDream

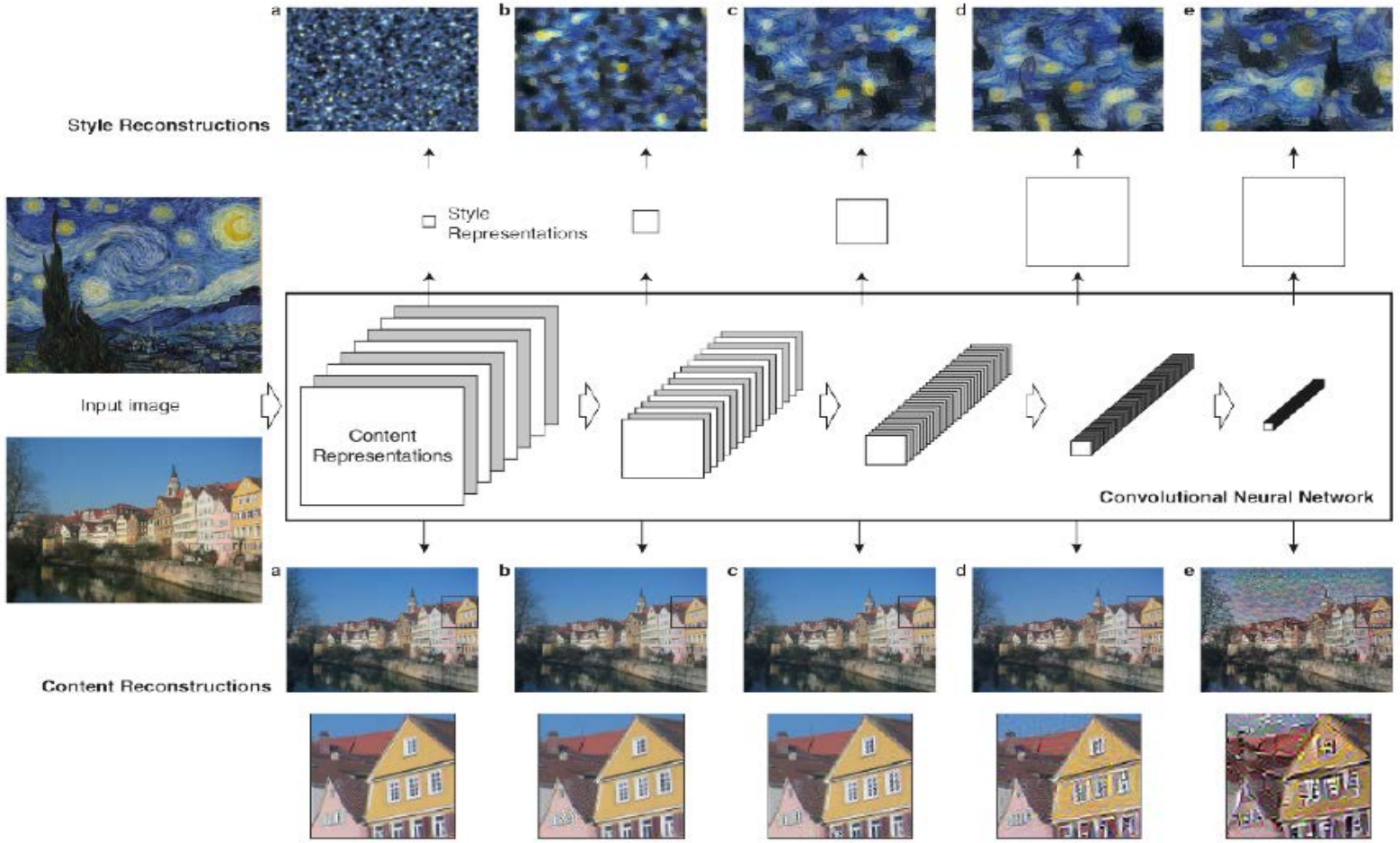


# „Fehler-“erkennung

- Training der Klasse Hanteln
- Erkennbarer „Fehler“
  - Viele Trainingsbilder enthalten Oberarm
  - Netz hat gelernt:
    - Oberarm wichtig für Klasse Hantel
  - Mögliches Problem: Hantel ohne Oberarm schlechter erkannt

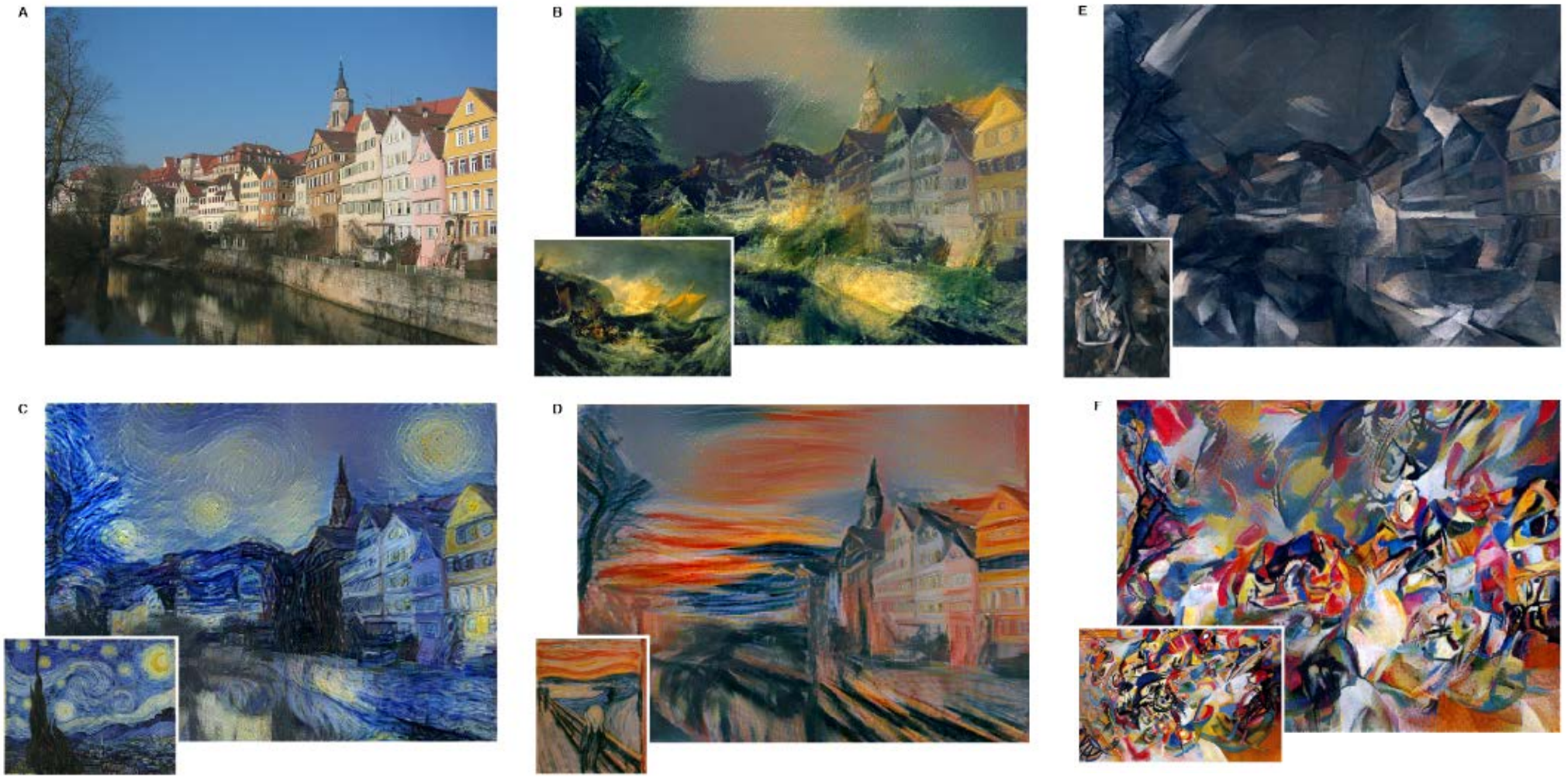
# Kunst von Google

# DeepArt – Universität Tübingen




Gatys et al.: A Neural Algorithm of Artistic Style

# DeepArt – Universität Tübingen



# Literatur

- [LeCun 89] – Gradient-Based Learning Applied to Document Recognition
- [Google 2014] – Going deeper with convolutions
- [Zeiler 2013] – Visualizing and Understanding Convolutional Neural Networks
- http://deeplearning.net/
- http://caffe.berkeleyvision.org/ (CNN Framework)
- www.deeplearningbook.org
- http://cs231n.stanford.edu/