Intelligente Sehsysteme

13 Convolutional Neural Networks

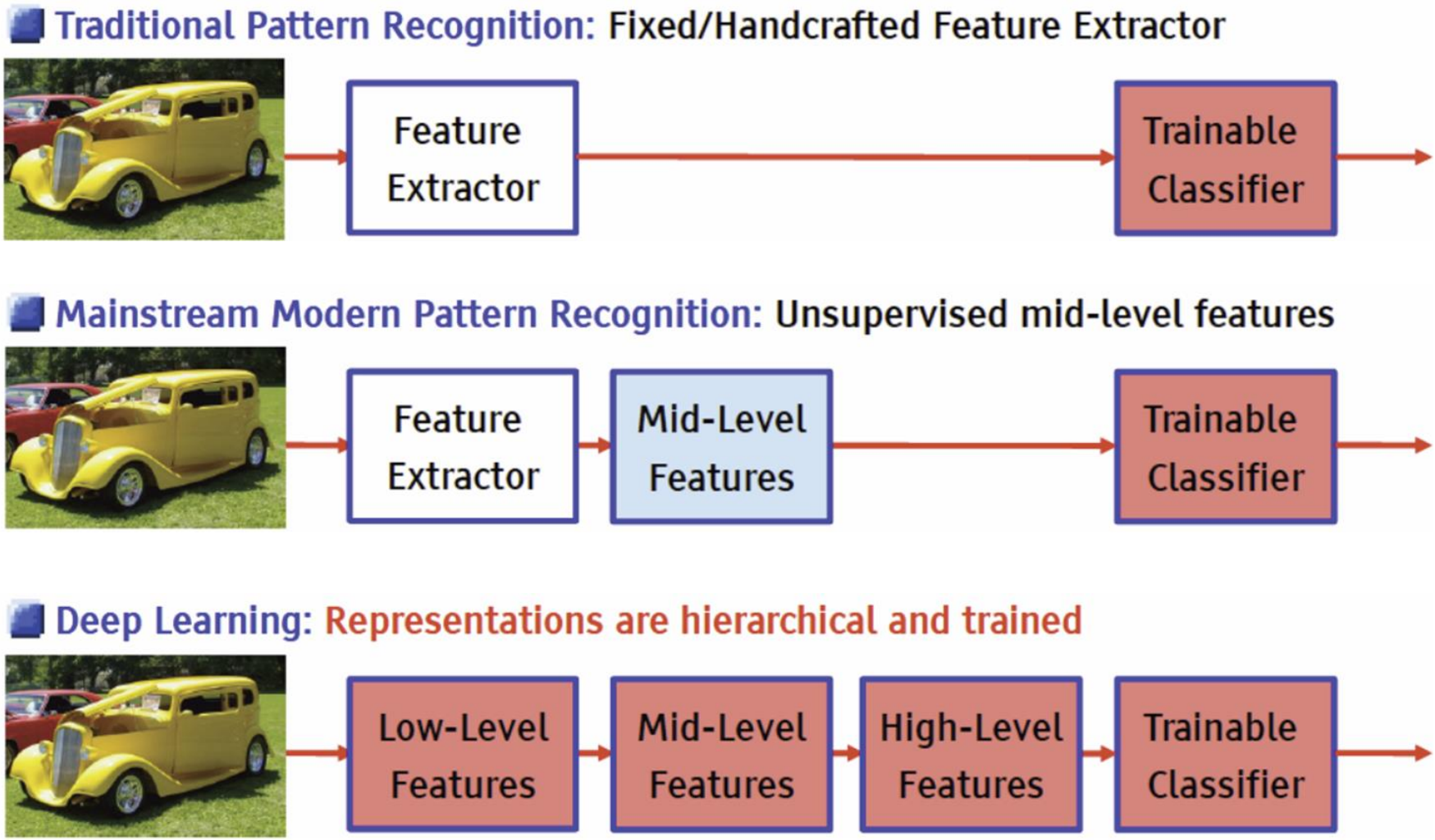
Convolutional Layer, Pooling Layer, Fully Connected Layer
Visual Fields, Receptive Fields, Feature Maps
Fully Convolutional Networks, Encoder - Decoder

Volker Steinhage

Inhalt

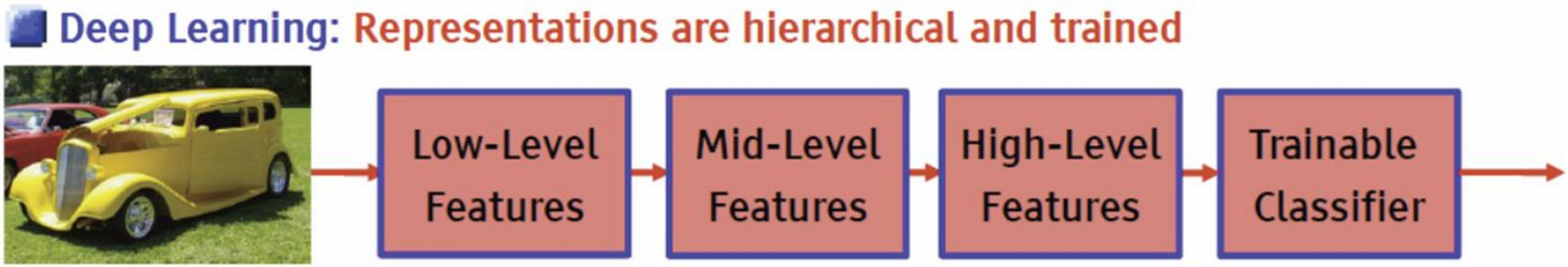
- Motivation: hierarchische Merkmalsrepräsentation
- Convolutional Layer und Fully Connected Layer
- Vier Hyperparameter der Convolutional Layer:
  filter size, stride, padding, depth
- Nonlinear Layer
- Pooling Layer
- Output Layer: Softmax-Funktion
- Loss function, Backpropagation, Minibatches
- Fully Convolutional Networks: Encoder - Decoder

Wiederholung: Ansätze des Feature Engineerings



Bildquelle: http://www.pamitc.org/cvpr15/files/lecun-20150610-cvpr-keynote.pdf (26.11.2019)

Wiederholung: Hierarchical and trained Features



- Deep Learning reduziert Feature Engineering auf das Lernen von hierarchischen Merkmalsrepräsentationen

- Das Lernen der Merkmalsrepräsentationen erfolgt alleine durch Trainingsmengen von unbearbeiteten Bilddaten und ihren Zuordnungen zu Objektklassen.

- Im optimalen Fall ergibt sich ein vollständiges End-to-End Learning: durch annotierte Bilddaten werden sowohl hierarchische Merkmalsrepräsentationen als auch die Klassifikation erlernt.

Feature Hierarchies

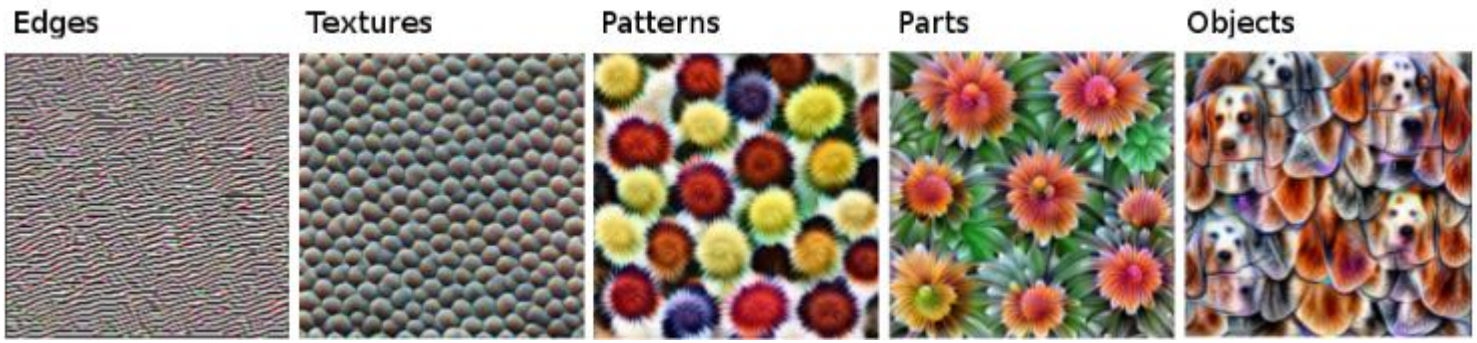


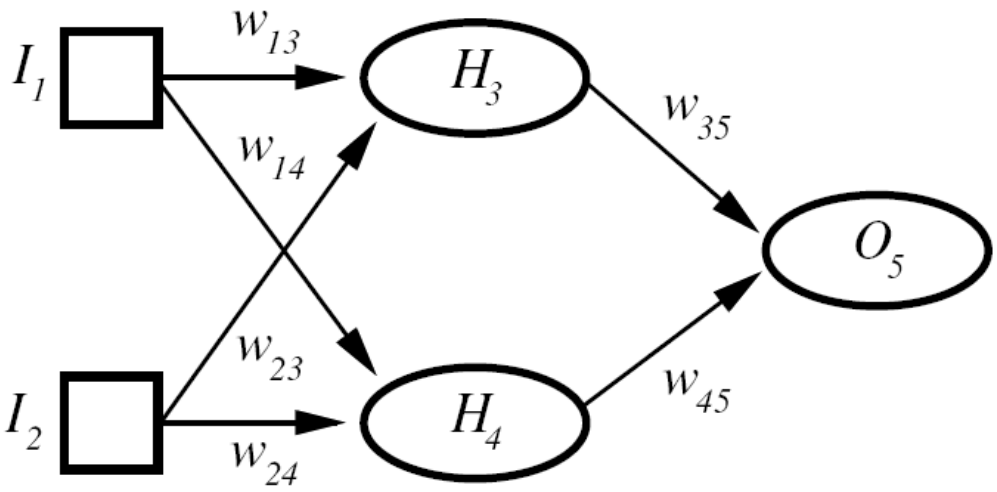
FIGURE 7.1: Features learned by a convolutional neural network (Inception V1) trained on the ImageNet data. The features range from simple features in the lower convolutional layers (left) to more abstract features in the higher convolutional layers (right). Figure from Olah, et al. 2017 (CC-BY 4.0) https://distill.pub/2017/feature-visualization/appendix/.

- The first convolutional layer(s) learn features such as edges and simple textures.
- Later convolutional layers learn features such as more complex textures and patterns.
- The last convolutional layers learn features such as objects or parts of objects.
- The fully connected layers learn to connect the activations from the high-level features to the individual classes to be predicted.

Quelle: https://christophm.github.io/interpretable-ml-book/cnn-features.html (19.12.2019)

Grundlagen (1)

- Grundlage sind hier künstliche neutonale Netze in Form von geschichteten Feed-Forward-Netzen (GFF) *

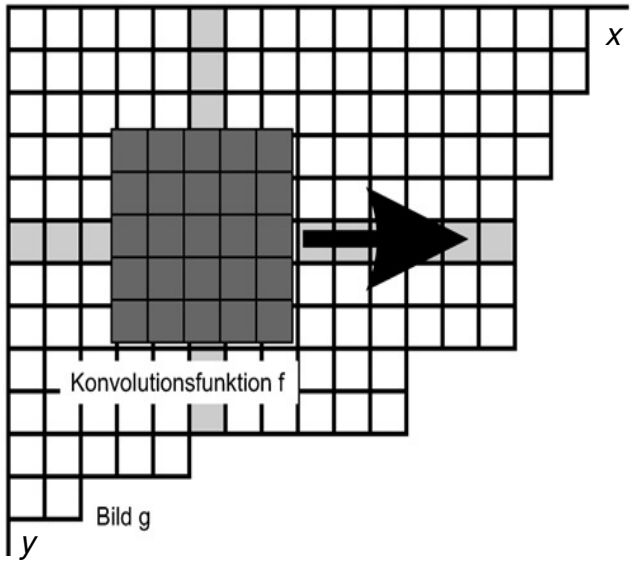


* vgl. Vorl. 15, Grundlagen der Künstlichen Intelligenz (BA-INF 110)

Grundlagen (2)

- Die Faltung einer Bildfunktion $g(x,y)$ mit einer Konvolutionsfunktion $f(u,v)$ berechnet eine mit den $f(u,v)$ gewichtete Summe der $g(x-u,y-v)$:

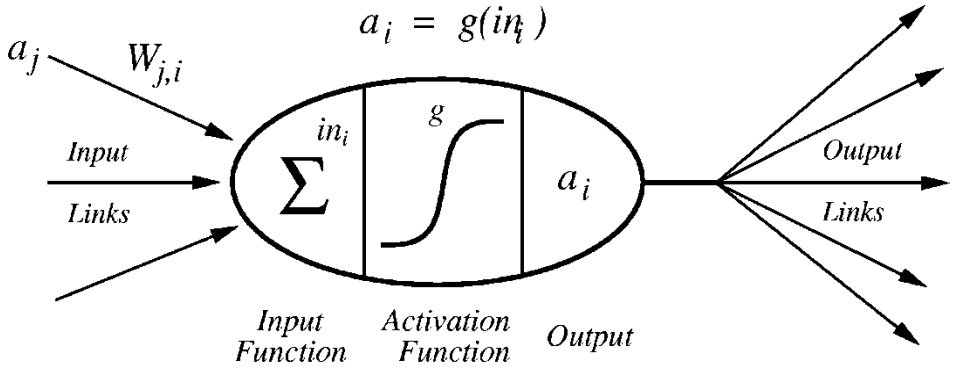
$$(f * g)(x,y) = \sum_{u \in \boldsymbol{D}} \sum_{v \in \boldsymbol{D}} f(u,v) \cdot g(x-u,y-v)$$



Bildquelle: Klaus Tönnies: Grundlagen der Bildverarbeitung, Pearson Studium, 2005.

- Die Eingabefunktion in_i eines Neurons n_i berechnet die Stärke der Eingabe als eine mit den Gewichten $w_{j,i}$ gewichtete Summe der Eingabe-aktivierung a_j über alle Neuronen j, die direkt mit i verbunden sind:

$$in_i = \sum_{j=0,\ldots,n} w_{j,i} \cdot a_j$$



Grundlagen (3)

Zur Erzeugung einer Hierarchie von Merkmalsrepräsentationen:

- Low-Level-Merkmale sind lokal (z.B. Kanten, Ecken)
  - → lokale begrenzte Konnektivität
    - → Reduktion der Anzahl von Parametern

Convolution

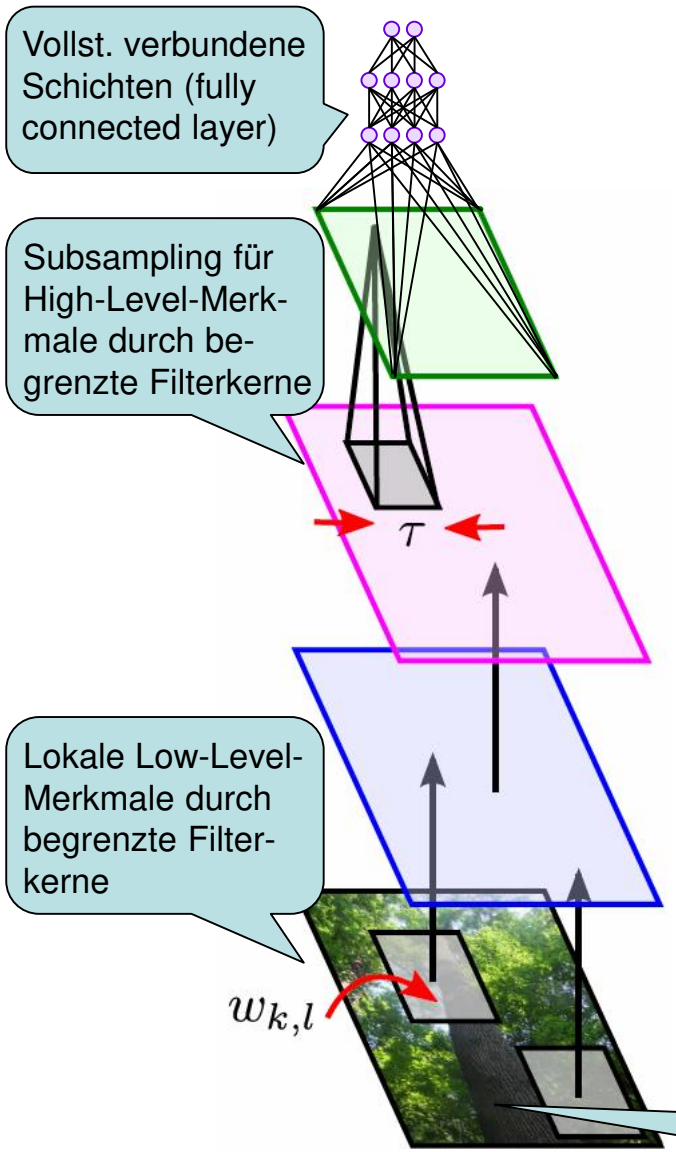
- High-Level-Merkmale sind globaler
  - → Schrittweises Subsampling
    - → Reduktion der Anzahl von Parametern

Pooling

- Netze sollen merkmalsbasierte Entscheidungen treffen
  - → Nichtlineare Aktivierungen

Thresholding

Aufbau eines CNNs



$$a_i = f(in_i)$$
$$= f(\Sigma_{j=0,\ldots,n} w_{j,i} \cdot a_j)$$

Klassifikation

$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

Pooling

auch Mittelwert oder Subsampling

$$y_{i,j} = f(in_{i,j})$$

z.B. $f(in_{i,j}) = [in_{i,j}]_+$

$$f(in_{i,j}) = sigmoid(in_{i,j})$$

Nichtlineare Aktivierung

$$in_{i,j} = \sum_{k,l} w_{k,l} \cdot z_{i-k,j-l}$$

Parameter

Konvolution

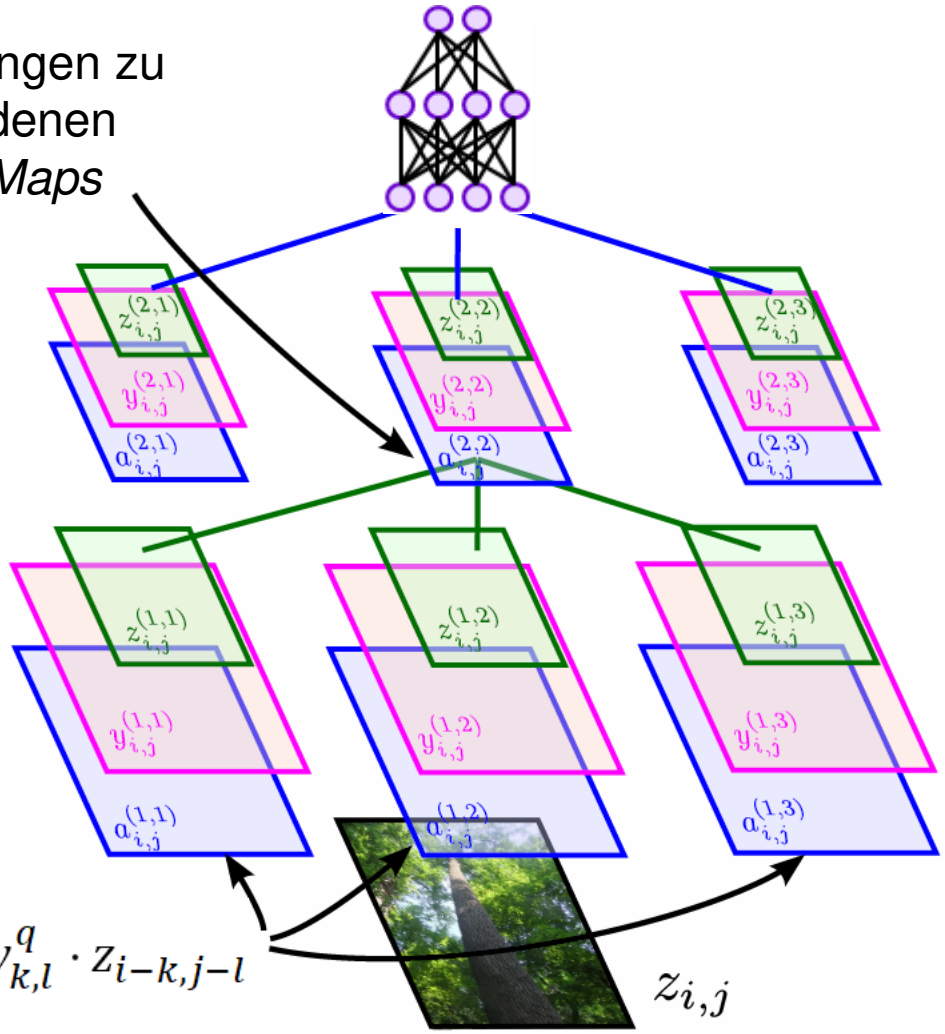
$z_{i,j}$

Eingabebild

Translationsinvarianz: dasselbe Filter mit denselben Gewichten wird über das gesamte Bild allokiert

Feature Maps

Verbindungen zu verschiedenen *Feature Maps*



- Pooling
- Nonlinear
- Convolution

- Pooling
- Nonlinear
- Convolution

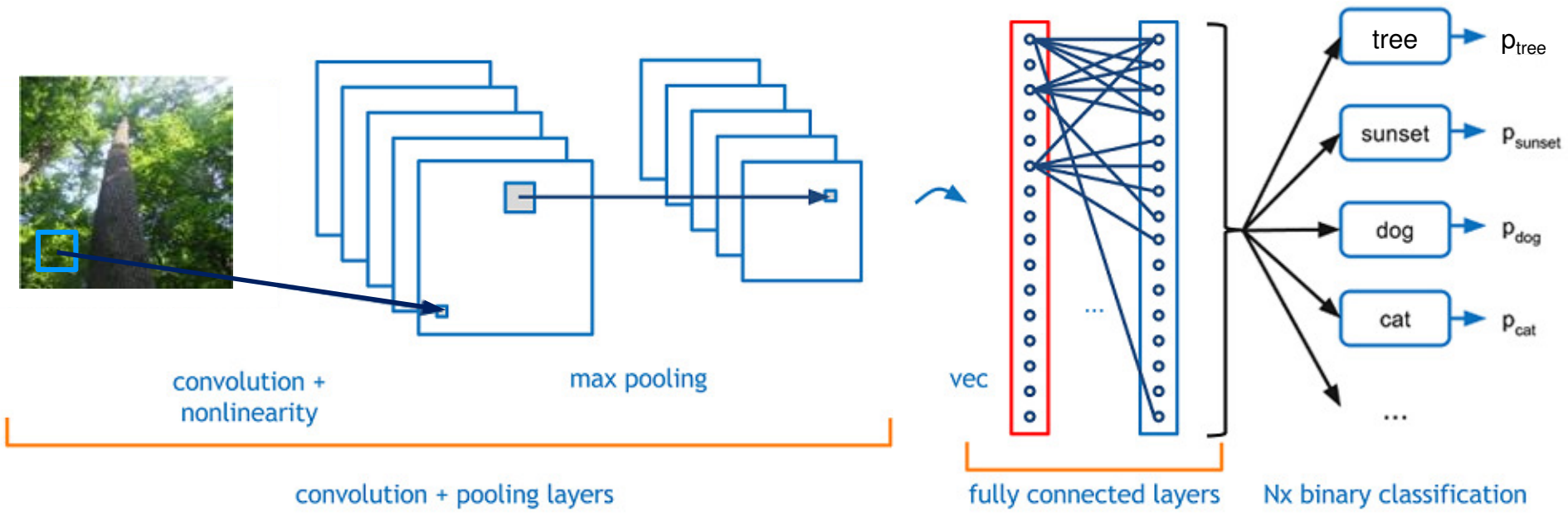
Verschied. Filter

$$in_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} \cdot z_{i-k,j-l}$$

Folienquelle: Richard Turner, Cambridge University, Department of Engineering

Bausteine eines einfachen CNNs

- Convolutional Layer
- Nonlinear Layer
- Pooling Layer
- Fully connected layer



Bildquelle: Adit Deshpande, UCLA, A Beginner's Guide To Understanding Convolutional Neural Networks (https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/)

Lokale Merkmale

- Konvolutionsfilter in erster Konvolutionsschicht kodieren lokale Merkmale

  → Bsp. hier: 9×9 Filter für rechtsgekrümmtes lokales Kontursegment

0	0	0	0	0	0	**1**	**1**	0
0	0	0	0	0	**1**	**1**	0	0
0	0	0	0	**1**	**1**	0	0	0
0	0	0	**1**	**1**	0	0	0	0
0	0	0	**1**	**1**	0	0	0	0
0	0	**1**	**1**	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0
1	**1**	0	0	0	0	0	0	0
1	**1**	0	0	0	0	0	0	0

Konvolutions-
filter

0	0	0	0	**1**	**1**	0	0	0
0	0	0	0	**1**	**1**	0	0	0
0	0	0	**1**	**1**	0	0	0	0
0	0	**1**	**1**	**1**	0	0	0	0
0	0	**1**	**1**	**1**	0	0	0	0
0	0	**1**	**1**	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0

Bildfunktion 1

1	**1**	0	0	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0
0	0	**1**	**1**	0	0	0	0	0
0	0	**1**	**1**	0	0	0	0	0
0	0	0	**1**	**1**	0	0	0	0
0	0	0	0	**1**	**1**	0	0	0
0	0	0	0	**1**	**1**	0	0	0

Bildfunktion 2

→ Starke Erkennung in Bildfunktion 1 mit Konv.-Ergebnis 11

→ Schwache Erkennung in Bildfunktion 2 mit Konv.-Ergebnis 4

Globalere Merkmale

- Konvolutionsfilter in späteren Konv-Schichten kodieren globalere Merkmale

  → Bspl. hier: 9×9 Filter für eine Raute

0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0
0	0	0	**1**	0	**1**	0	0	0
0	0	**1**	0	0	0	**1**	0	0
0	**1**	0	0	0	0	0	**1**	0
0	0	**1**	0	0	0	**1**	0	0
0	0	0	**1**	0	**1**	0	0	0
0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0

Konvolutions-
filter

0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0
0	0	0	**1**	0	**1**	0	0	0
0	0	**1**	0	0	0	**1**	0	0
0	**1**	0	0	0	0	0	**1**	0
0	0	**1**	0	0	0	**1**	0	0
0	0	0	**1**	0	**1**	0	0	0
0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0

Bildfunktion 1

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	**1**	0	0	0	**1**	0	0
0	0	**1**	0	0	0	**1**	0	0
0	0	**1**	0	0	0	**1**	0	0
0	0	**1**	0	0	0	**1**	0	0
0	0	**1**	**1**	**1**	**1**	**1**	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Bildfunktion 2

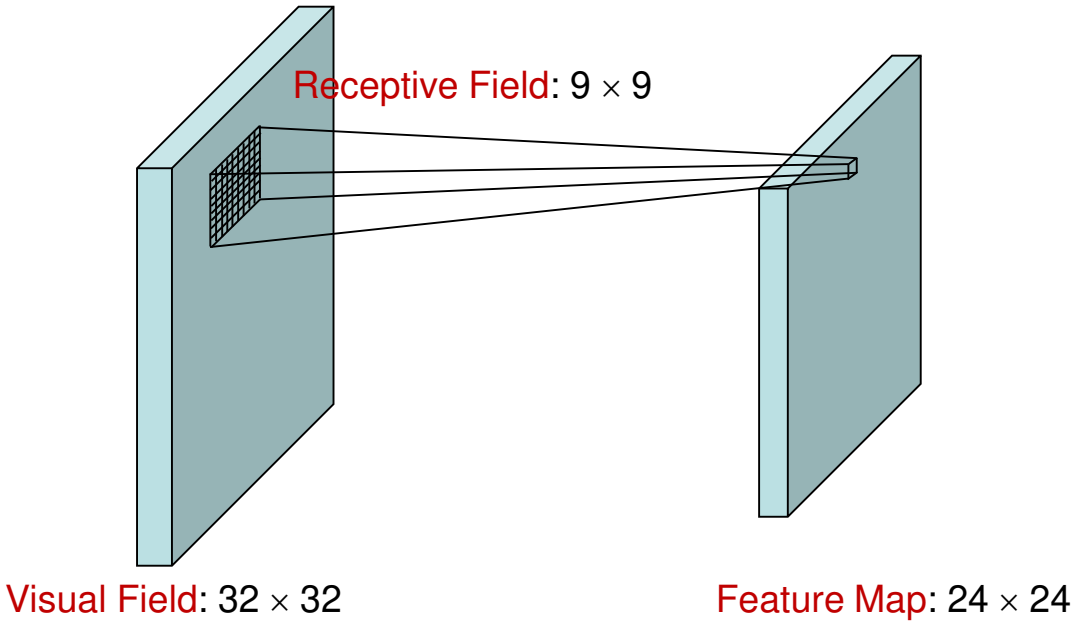
→ Starke Erkennung in Bildfunktion 1 mit Konv.-Ergebnis 12

→ Schwache Erkennung in Bildfunktion 2 mit Konv.-Ergebnis 6

Convolutional Layer: Receptive field/Size

Vier Hyperparameter definieren eine Konvolutionsschicht:

1) Filtergröße (size) – auch rezeptives Feld



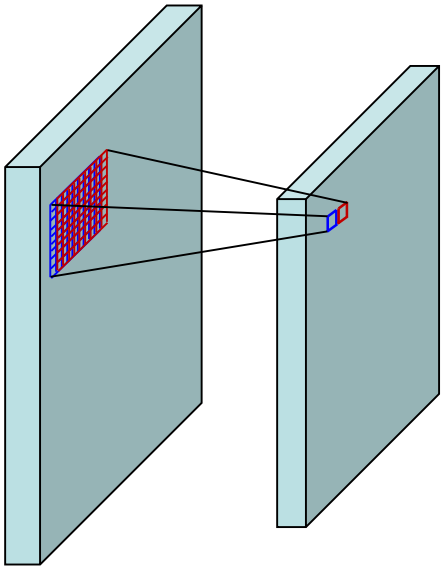
- Filtergröße bestimmt *rezeptives Feld* jedes einzelnen Neurons in Konv-Schicht
- Alle Neuronen aus Konvolutionen mit demselben Filter bilden: *Feature Map*
- Sichtfeld aller Neuronen einer Feature Map: *Visual Field*

Convolutional Layer: Stride

Vier Hyperparameter definieren eine Konvolutionsschicht:

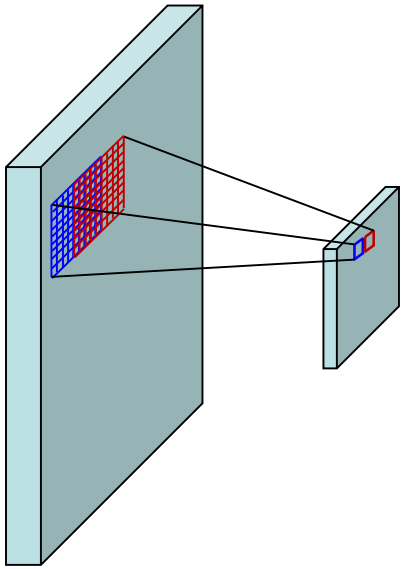
2) Schrittlänge (stride) für horizontalen und vertikalen Versatz

Receptive Field: 9×9 Stride: 1



Visual Field: 33×33 Feature Map: 25×25

Receptive Field: 9×9 Stride: 4



Visual Field: 33×33 Feature Map: 7×7

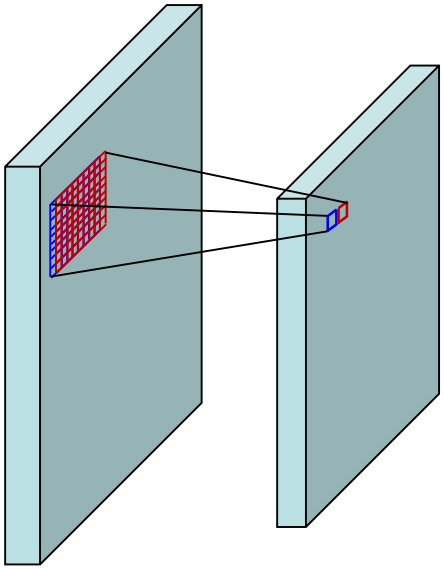
- Kleine Schrittlänge: hoher Überlappungsgrad & große Feature Maps
- Randproblem

Convolutional Layer: Padding

Vier Hyperparameter definieren eine Konvolutionsschicht:

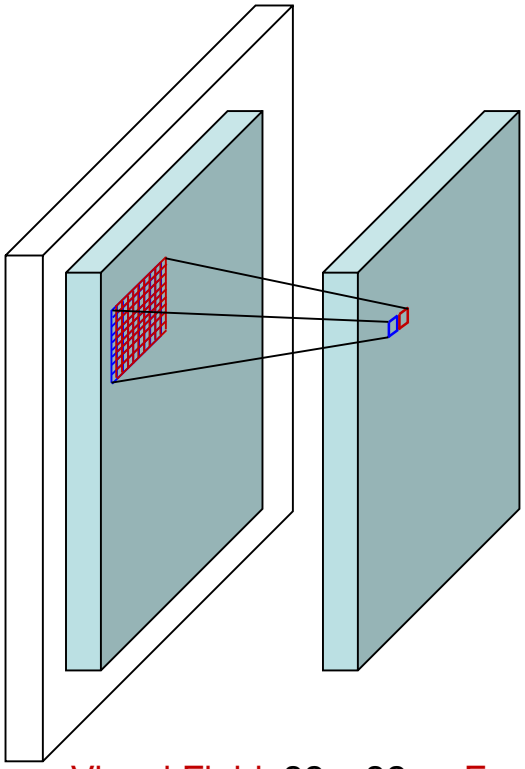
3) Zero-padding: Randbehandlung der Faltung durch Auffüllen mit Nullen

Receptive Field: 9 × 9 Stride: 1



Visual Field: 32 × 32 Feature Map: 24 × 24

Receptive Field: 9 × 9 Stride: 1 Padding: 4



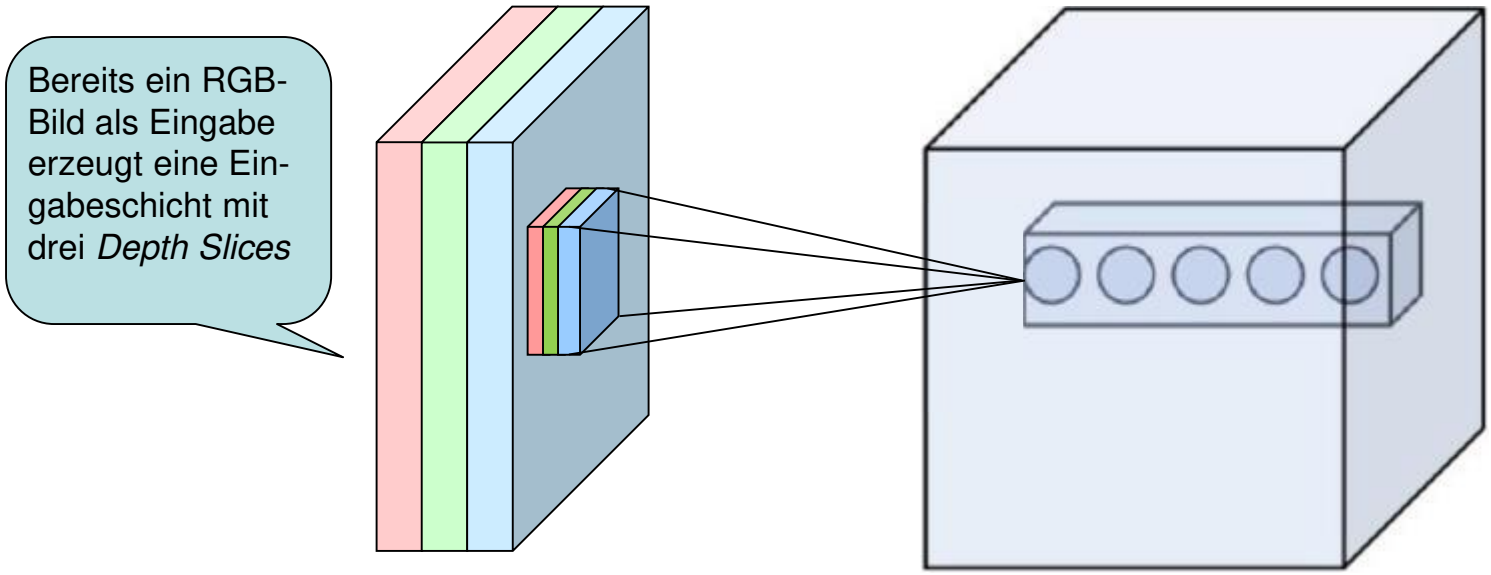
Visual Field: 32 × 32 Feature Map: 32 × 32

Hier: Auffüllen mit Rand von P = 4 Zeilen und Spalten mit Nullen
Allg.: P = (K-1)/2 mit Filtergröße K

Convolutional Layer: Depth

Vier Hyperparameter definieren eine Konvolutionsschicht:

4) Tiefe (depth) → Zahl der Feature Maps



- Verschiedene Neuronen entlang der Tiefendimension (hier 5) kodieren verschiedene Merkmale wie orientierte Kanten, Farben, Texturen, etc.
- Alle Neuronen einer *Feature Map* (auch *Depth Slice*) nutzen dieselben Gewichte in ihren eingehenden Kanten (*Parameter Sharing*/*Shared Weights*)

Convolutional Layer: Hyperparameter

Zur Wahl der Hyperparameter:

- Geg. Sei Größe W der Eingabeschicht: bei $32 \times 32 \rightsquigarrow W = 32$
- Die Größe O der folgenden Konvolutionsschicht ergibt sich nach:

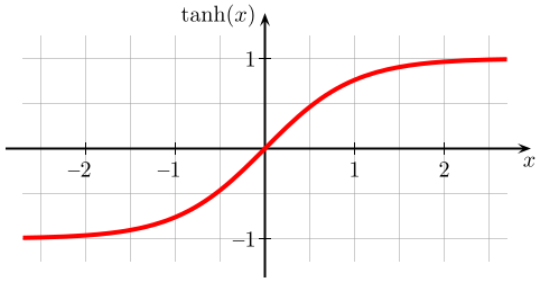
$$O = (W - K + 2 \cdot P)/S + 1$$

  mit Konvolutionskerngröße $K = 9$, Padding P und Stride S

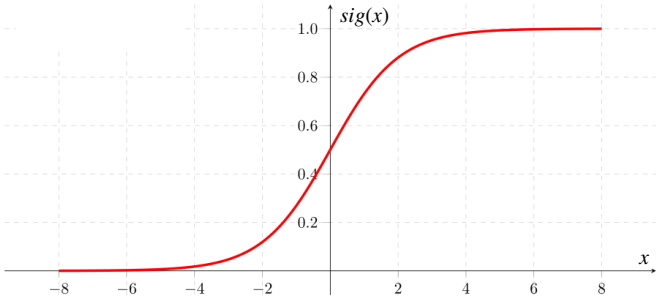
- Bespiele: $P = 0$, $S = 1$: $24 = (32 - 9 + 2 \cdot 0)/1 + 1$
  $P = 4$, $S = 1$: $32 = (32 - 9 + 2 \cdot 4)/1 + 1$
  $P = 0$, $S = 4$: $7 = (33 - 9 + 2 \cdot 0)/4 + 1$
- Ein- und Ausgabeschichten haben dieselbe Größe bei $P = (K - 1)/2$ und $S = 1$

Nonlinear Layer (1)

- Konvolutionsschichten (*Conv Layers*) setzen nur lineare Operationalität um
- Nichtlinearität wird durch *Nonlinear Layers* umgesetzt
- Bekannt sind als nichtlinearen Aktivierungsfunktionen insbes.:
  - Tangens hyperbolicus $tanh(x)$



  - Sigmoidfunktion $sig(x)$



Abbildungsquellen: https://de.wikipedia.org/wiki/Tangens_hyperbolicus_und_Kotangens_hyperbolicus (07.01.2018)
https://de.wikipedia.org/wiki/Sigmoidfunktion (07.01.2018)

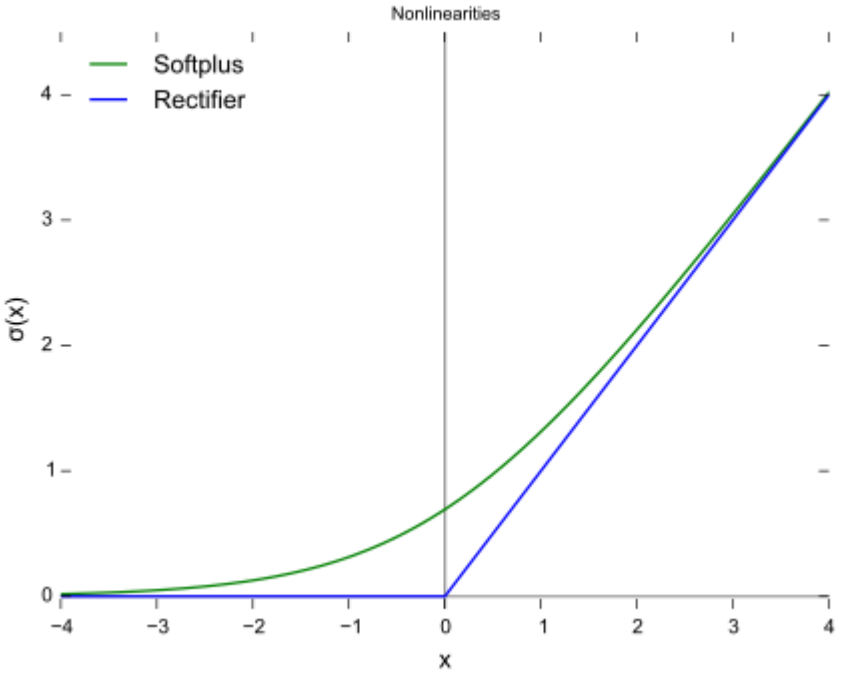
Nonlinear Layer (2)

- Viel effizienter und ohne signifikanten Verlust an Generalisierungsgenauigkeit im Vergleich zu $tanh(x)$ und $sig(x)$ ist die *Rectifier*-Aktiviervierungsfunktion (*Rectified Linear Unit*) zu berechnen, die den positiven Teil der Eingabe ausgibt:[1]

$$ReLu(x) = x^+ = \max(0,x)$$

- Da Backpropagation die Berechnung der Gradienten verlangt, wird die *Softplus function* als differenzierbare Approximation benutzt:

$$f(x) = \ln(1 + e^x)$$

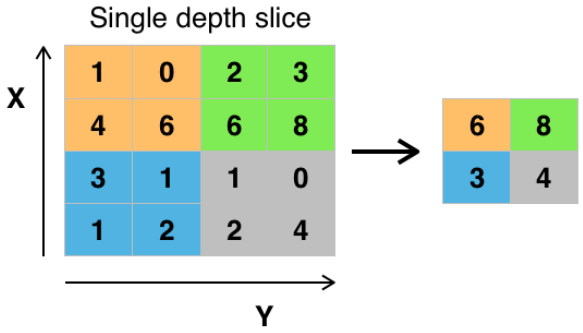


(1) Krizhevsky, A.; Sutskever, I.; Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks". *Advances in Neural Information Processing Systems 1:1097–1105*

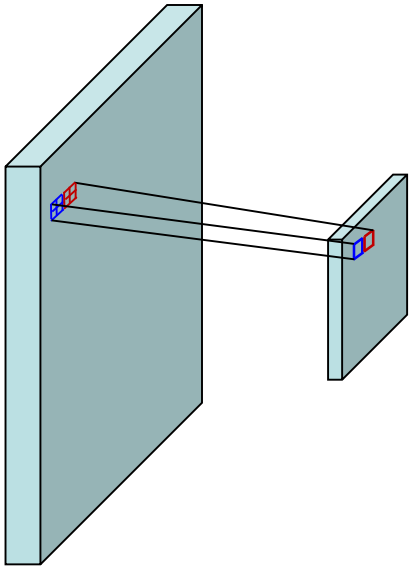
(2) Abbildungsquelle: https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

Pooling Layer

- Pooling setzt Down-Sampling um. Zur visuellen Objekterkennung ist die *exakte* Position von Merkmalen z. B. einer Kante im Bild von vernachlässigbarem Interesse – die ungefähre Lokalisierung eines Features ist hinreichend
- Am stärksten verbreitet ist Max-Pooling mit Kernelgröße $K = 2$ und Stride $S = 2$: aus allen disjunkten 2×2 Quadraten aus Neuronen wird jeweils nur die Aktivität des aktivsten (daher "Max") Neurons für die weiteren Berechnungsschritte übernommen; 75% aller Neuronenwerte werden verworfen



Max-Pooling mit $K = 2$ und $S = 2$ [1]



Receptive Field: 2×2

Stride: 2

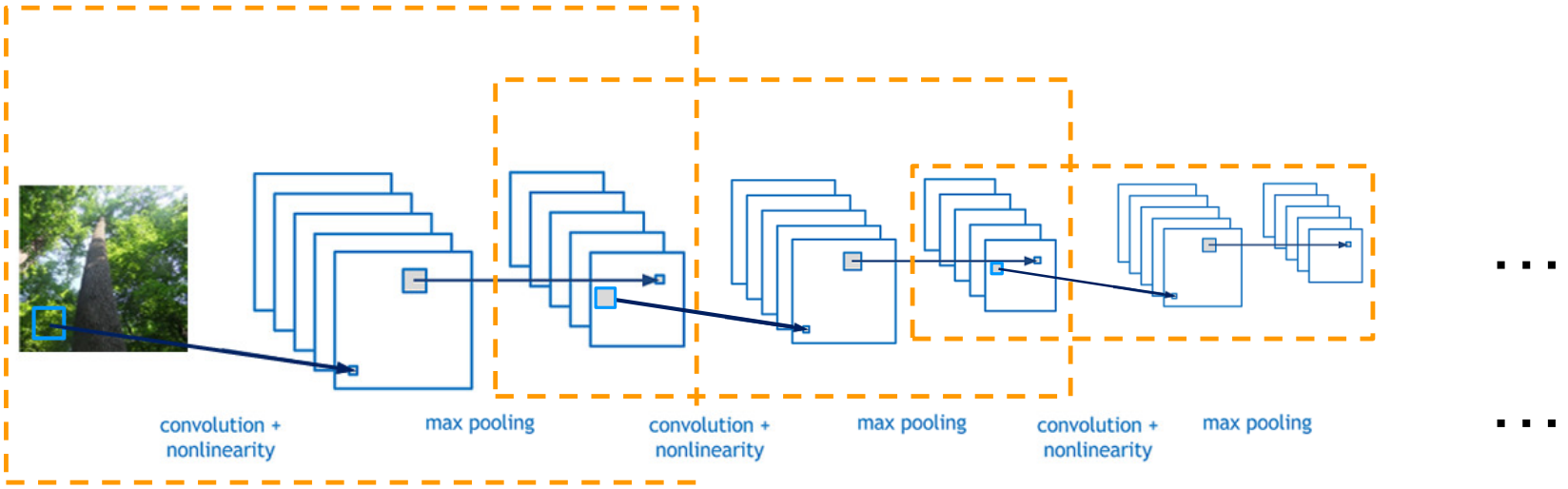
Visual Field: 32×32

Max Pooling → 16×16

[1] Bildquelle: https://en.wikipedia.org/wiki/Convolutional_neural_network

Deep Convolutional Networks

- Tiefe Konvolutionsnetze werden durch Wiederholungen von Blöcken aus Konvolutions-, ReLu- und Pooling-Schichten erzeugt:

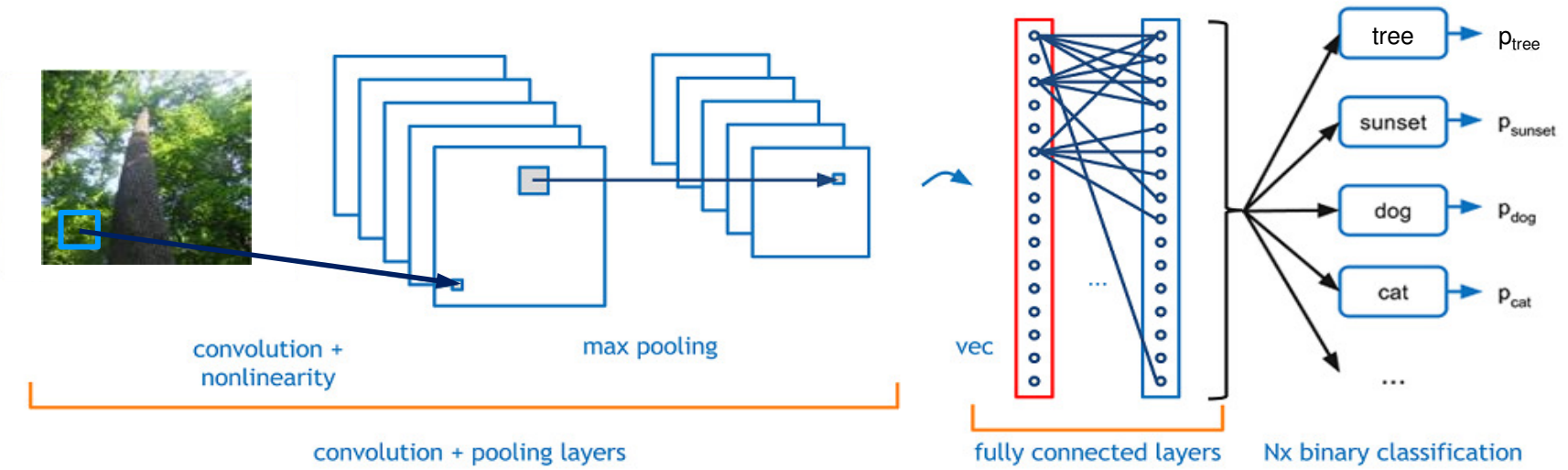


- Mit jedem neuen Block werden abstraktere/globalere Features erzeugt
- In jedem Block werden verschiedene Features in einer Skala erzeugt

Fully Connected Layer

Nach der Erzeugung der letzten Abstraktionsstufe von Merkmalen werden vollständig verbundene Schichten im Sinne vom klassischen Feed-Forward-Netzen (auch mehrstufige Perzeptrons) zur Klassifikation installiert:

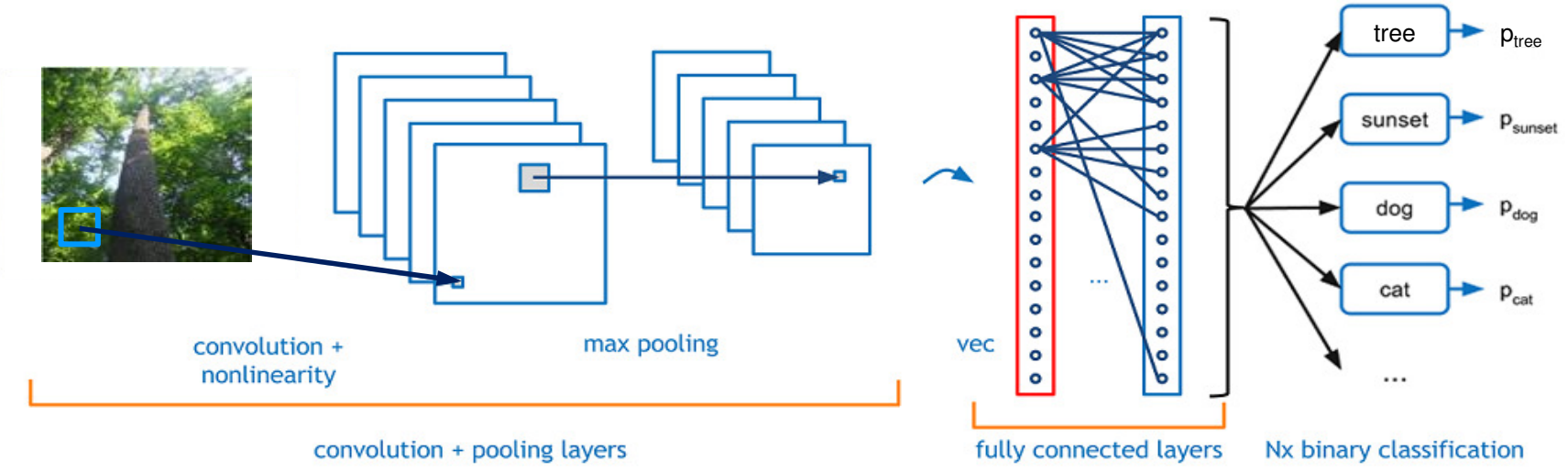
- Jedes Neuron einer Schicht ist mit jedem Neuron der Vorgängerschicht verbunden
- Die Neuronen der ersten Schicht sind mit allen Neuronen aller Feature Maps der letzten Konvolutionsschicht (i.A. nach Passieren von ReLu + Pooling) verbunden
- Die Neuronen der letzten Schicht zeigen *N* Neuronen, wobei *N* der Anzahl der möglichen Objektklassen entspricht



Output Layer (1)

Die Neuronen der letzten Schicht zeigen *N* Neuronen:

- *N* entspricht der Anzahl der möglichen Objektklassen
- jedes Ausgabeneuron liefert die Wahr‘keit für die entspr. Klassenzugehörigkeit
- die Berechnung dieser Wahr‘keiten erfolgt durch die Softmax-Funktion



Output Layer (2)

Die Softmax-Funktion σ skaliert einen *k*-dim. Vektor $\boldsymbol{v} = (v_1,\ldots,v_k)$ mit reellen Komponenten v_j zu einem *k*-dim. Vektor $\sigma(\boldsymbol{v})$:

$$\sigma\colon \mathbb{R}^k \to \mathbb{R}^k,\ \sigma(\boldsymbol{v})_j = e^{v_j} \Big/ \sum_{i=1}^{k} e^{v_i}$$

mit $\sigma(v_j) \in [0,1]\ , j = 1,\ldots,k$ und $\sum_{j=1,\ldots,k} \sigma(v_j) = 1$

Die Softmax-Funktion kann so genutzt werden, um Zuordnungswahr‘keiten über *k* verschiedene Klassen zu repräsentieren

* Die Softmax-Funktion wird auch als normalisierte Exponentialfunktion bezeichnet

Output Layer (3)

Für die Ausgabeschicht wird

- die übliche Aktivierungsfunktion also durch die Softmax-Funktion ersetzt
- die Zuordnungswahr‘keit $P(y = j \mid \boldsymbol{x})$ für die j-te Klasse berechnet nach

$$P(y = j|\boldsymbol{x}) = \frac{e^{\boldsymbol{x}^T \boldsymbol{w}_j}}{\sum_{i=1}^{k} e^{\boldsymbol{x}^T \boldsymbol{w}_i}}$$

mit Input-Vektor $\boldsymbol{x}$ und Gewichtsvektor $\boldsymbol{w}_j$ für das j-te Ausgabeneuron

Loss function

Ziel: Lernen der Netzparameter θ über Minimieren einer *Loss function* L

Geg.: zwei Verteilungen *P* (Ground Truth) und *Q* (Netzwerk)

- *Loss function* L = Kreuzentropie $H(P,Q)$ (Cross entropy) von *P* und *Q:*

$$H(P,Q) = -\sum_{j=1,..k} P(j) \log(\mathrm{Q}(j))$$

- Minimieren von L durch Backpropagation:

$$w_{ij} \leftarrow w_{ij} - \partial L/\partial w_{ij}$$

Mini-batches

- Aktualisierung der Parameter nach Bearbeitung der gesamten Trainingsmenge (eine Epoche) → langsamer Lernfortschritt
- Aktualisierung der Parameter nach Bearbeitung jedes Trainingsbeispiels → ungenau und ineffizient
- Mini-Batch als Kompromiss zw. Lern- und Rechenperformanz
- Bspl.: Trainingsdatensatz mit 20.000 Beispielen und Batch-Size = 20 führt zu 1.000 Mini-Batches
- Fehlerberechnung nach einem Batch-Durchlauf:

$$E_{MB} = \frac{1}{T}\sum_{t=1}^{T} E_t$$

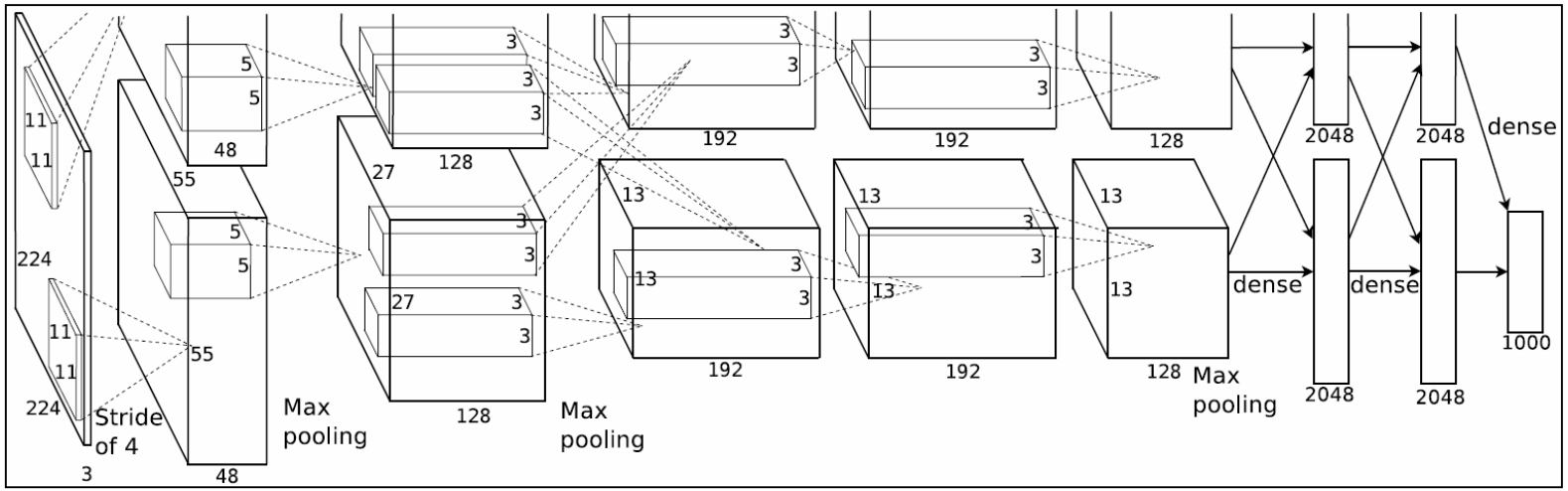
mit T = |Trainingsdatensatz|, E_t = Fehler für Trainingsbeispiel t

AlexNet: 2010

ILSVRC-2010: (1)

- "We trained the network for roughly 90 cycles through the training set of 1.2 million images, which took five to six days on two NVIDIA GTX 580 3GB GPUs." (2)
- "Our network achieves top-1 and top-5 test set error rates of 37.5% and 17.0%." (2)

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**



(1) ILSVRC = ImageNet Large-Scale Visual Recognition Challenge ("annual Olympics of computer vision")

(2) Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, „Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems, 2012* (NIPS 2012), pp. 1097-1105.

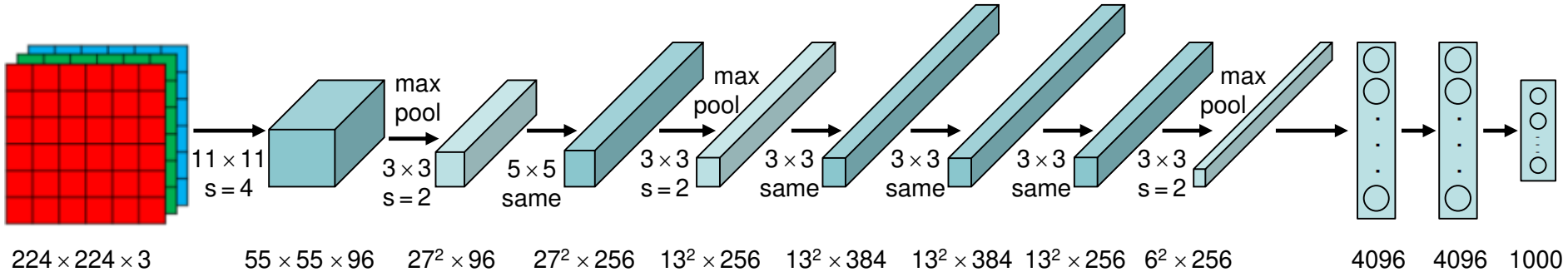
AlexNet: Architektur

Anzahl Neuronen:

- Input: $224 \times 224 \times 3 =$ 150.528
- C1: $55 \times 55 \times 96 =$ 290.400
- C2: $27 \times 27 \times 256 =$ 186.624
- C3: $13 \times 13 \times 384 =$ 64.896
- C4: $13 \times 13 \times 384 =$ 64.896
- C5: $13 \times 13 \times 256 =$ 43.246
- F1: 4.096
- F2: 4.096
- Output: 1.000

Anzahl Parameter:

- ca. 60 M (AlexNet)



„same“ = same padding = Erhaltung der Layer-Dimension: bei Filtergröße $k \times k \rightarrow p = (k\text{-}1)/2$

AlexNet: 2012

ILSVRC-2012:

- "Training one CNN, with an extra sixth convolutional layer over the last pooling layer, to classify the entire ImageNet Fall 2011 release (15M images, 22K categories), and then "fine-tuning" it on ILSVRC-2012 gives an error rate of 16.6%."

- "Combining the predictions of many different models is a very successful way to reduce test errors. […] Averaging the predictions of two CNNs that were pre-trained on the entire Fall 2011 release […] gives an error rate of 15.3%. The second-best contest entry achieved an error rate of 26.2% […] ."

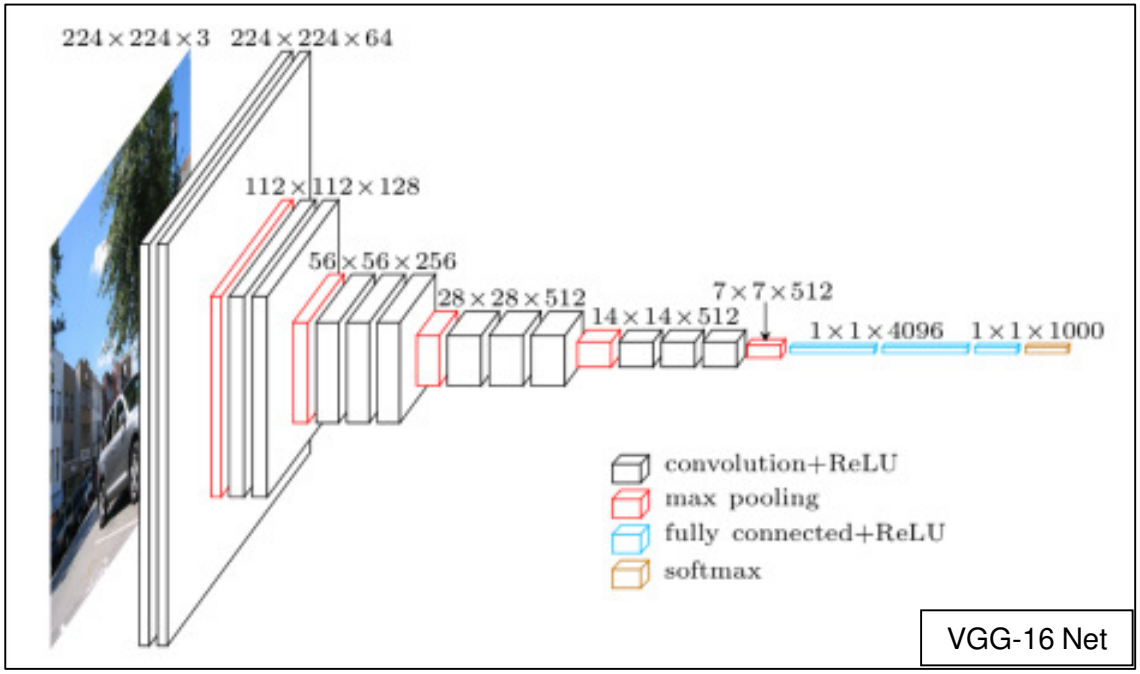
Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, „Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems, 2012* (NIPS 2012), pp. 1097-1105.

VGG Net (2014)

"Keep it deep. Keep it simple":

- VVG-Net reduziert die Zahl der Hyperparameter durch ausnahmslosen Einsatz von
  - 3×3 Konvolutionsfiltern mit Stride = 1 und Padding = 1
  - 2×2 Max-Pooling mit Stride = 2
- Nach 2-3-wöchigen Training (versionsabhängig) auf 4 NVIDEA-Titan-Black-GPUs erzielte Konfiguration E einen Top-5-Fehler von 7.3% auf dem Testdatensatz

Visual Geometry Group, Dept. of Engineering Science, Oxford Univ.



VGG-16 Net

Karen Simonyan, Andrew Zisserman, „Very deep convolutional networks for large-scale image recognition." *ArXiv preprint arXiv:1409.1556, 2014.* 6th Intern.Conf. on Learning Representations 2015 (ICLR 2015).

VGG Net: Basics

- Die Kombination von zwei 3×3-Konvolutionen haben effektiv dasselbe rezeptive Feld wie eine 5×5-Konvolution
- Die Kombination von drei 3×3-Konvolutionen haben effektiv dasselbe rezeptive Feld wie eine 7×7-Konvolution
- Vorteile:
  - Im Vergleich zu einer ReLu-Ebene tragen zwei bzw. drei ReLu-Ebenen stärker zur gesamten Entscheidungsfunktion bei
  - Reduktion der Parameter

    Beispiel: drei 3×3-Konvolutionen mit C Kanälen als Ein- und Ausgabe haben $3 \cdot (3^2 \cdot C^2) = 27 \cdot C^2$ Gewichte; eine 7×7-Konvolution mit C Kanälen als Ein- und Ausgabe hat $7^2 \cdot C^2 = 49 \cdot C^2$ Gewichte

Karen Simonyan, Andrew Zisserman, „Very deep convolutional networks for large-scale image recognition.“ *ArXiv preprint arXiv:1409.1556, 2014.* 6th Intern.Conf. on Learning Representations 2015 (ICLR 2015).

VGG Net: Versions

VGG-Versionen:

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

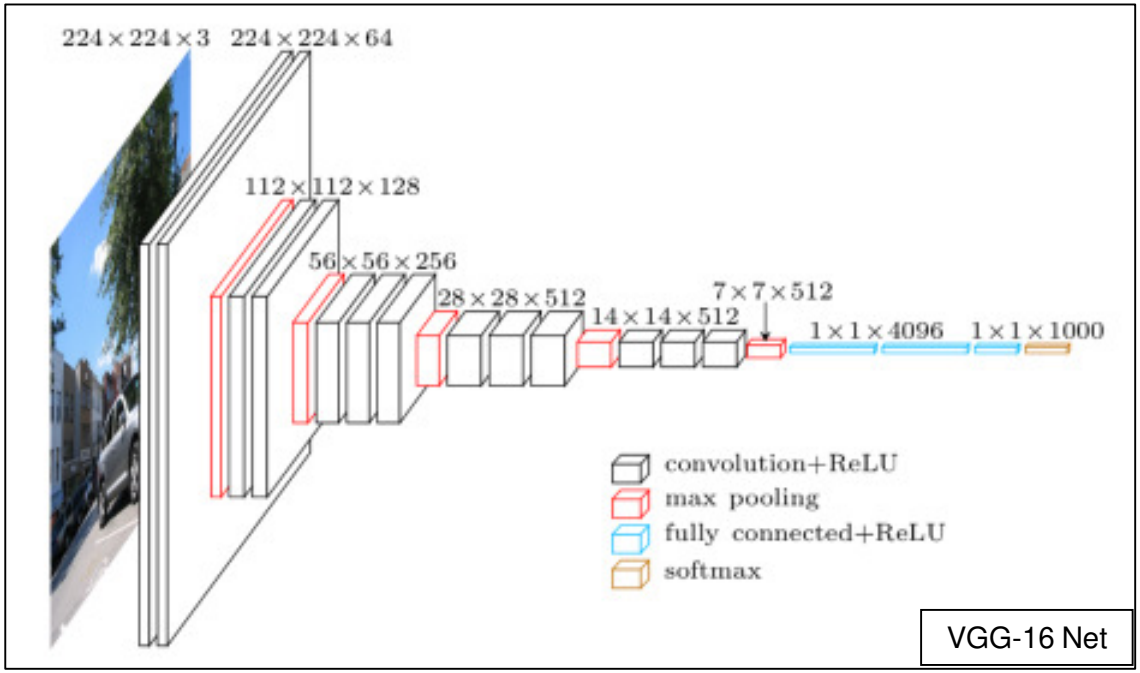
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Karen Simonyan, Andrew Zisserman: *Very deep convolutional networks for large-scale image recognition*. ArXiv preprint arXiv:1409.1556, 2014 and 6th Intern.Conf. on Learning Representations 2015 (ICLR 2015).

VGG Net: ILSVRC-2014

ImageNet Challenge 2014:

- Platz 1 im Lokalisierungswettbewerb
  ("the last fully connected layer predicts the bounding box location instead of the class scores")
- Platz 2 im Klassifikationswettbewerb



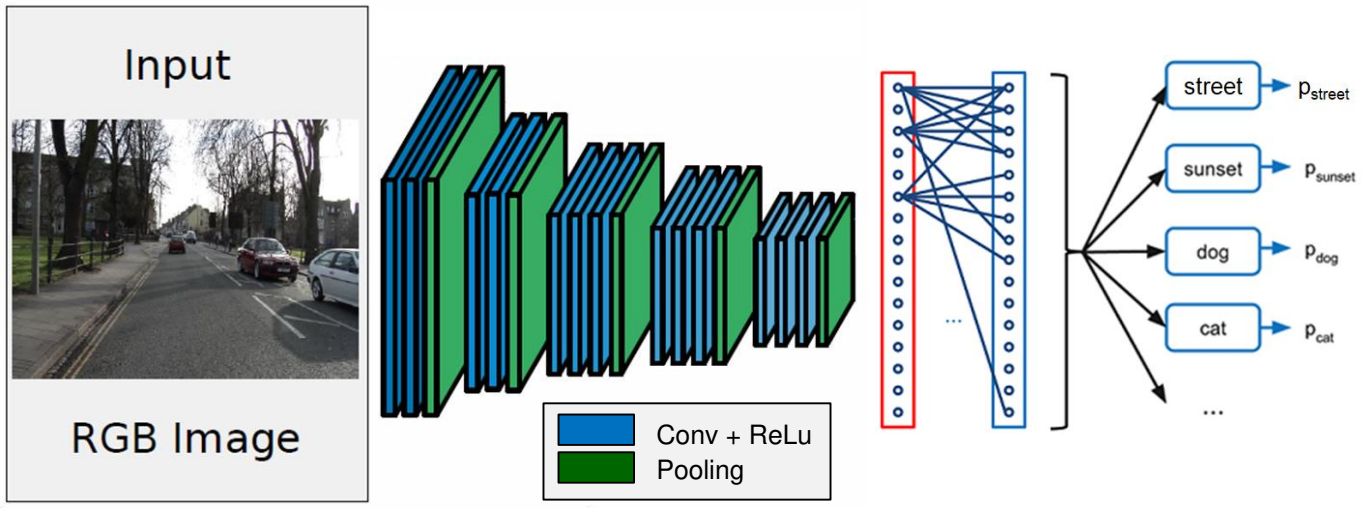
VGG-16 Net

Karen Simonyan, Andrew Zisserman: *Very deep convolutional networks for large-scale image recognition*. ArXiv preprint arXiv:1409.1556, 2014 and 6th Intern.Conf. on Learning Representations 2015 (ICLR 2015).

Convolutional Networks zur Klassifikation

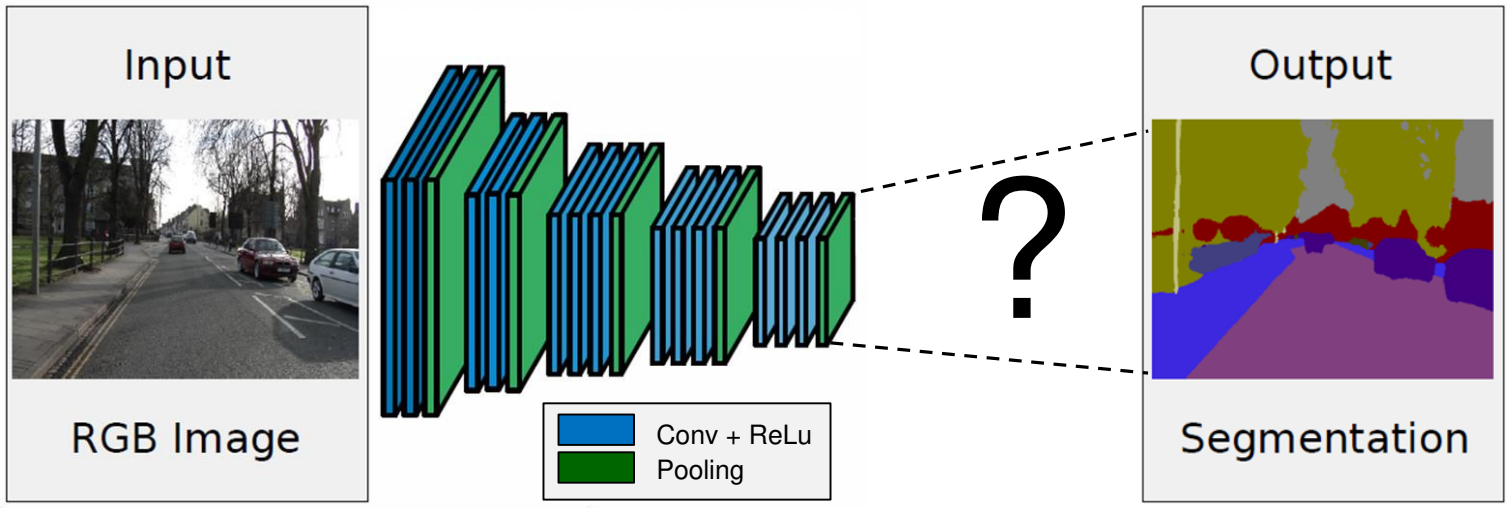
- Farbbild $\boldsymbol{x}_i$ → n·CNN-Layer + Fully Connected Layer → W'keitsvektor $\boldsymbol{v}_j$ über c verschied. Klassen

- Jedes CNN-Layer:
  - Konvolution über vorherige Layerwerte bzw. Eingabewerte
  - ReLu(x) = max(x;0)
  - MaxPooling (optional)



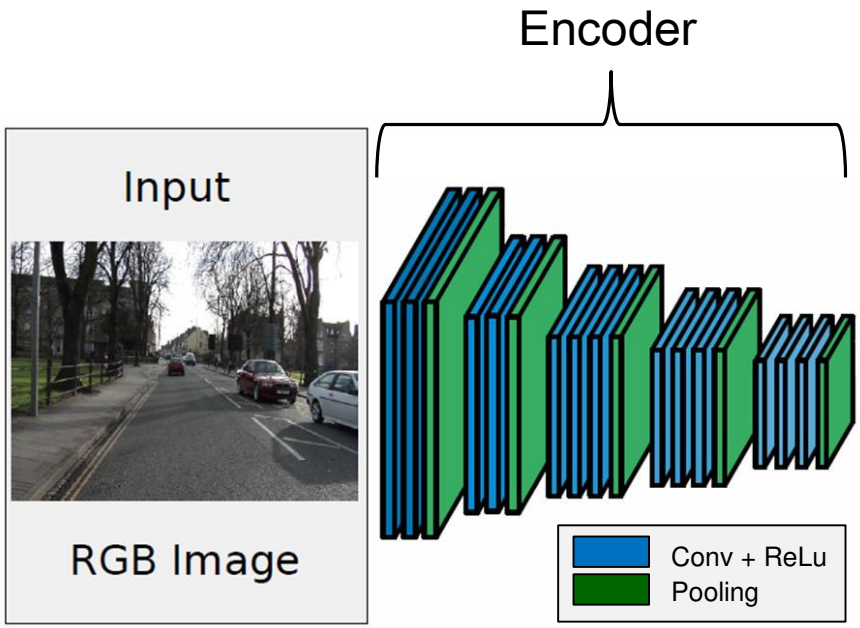
Convolutional Networks zur Segmentierung

- Farbbild $\boldsymbol{x}_i \rightarrow$ n·CNN-Layer $\rightarrow$ … $\rightarrow$ W'keitsvektoren $\boldsymbol{v}_{x,y}$ über c verschiedene Klassen für alle Pixel



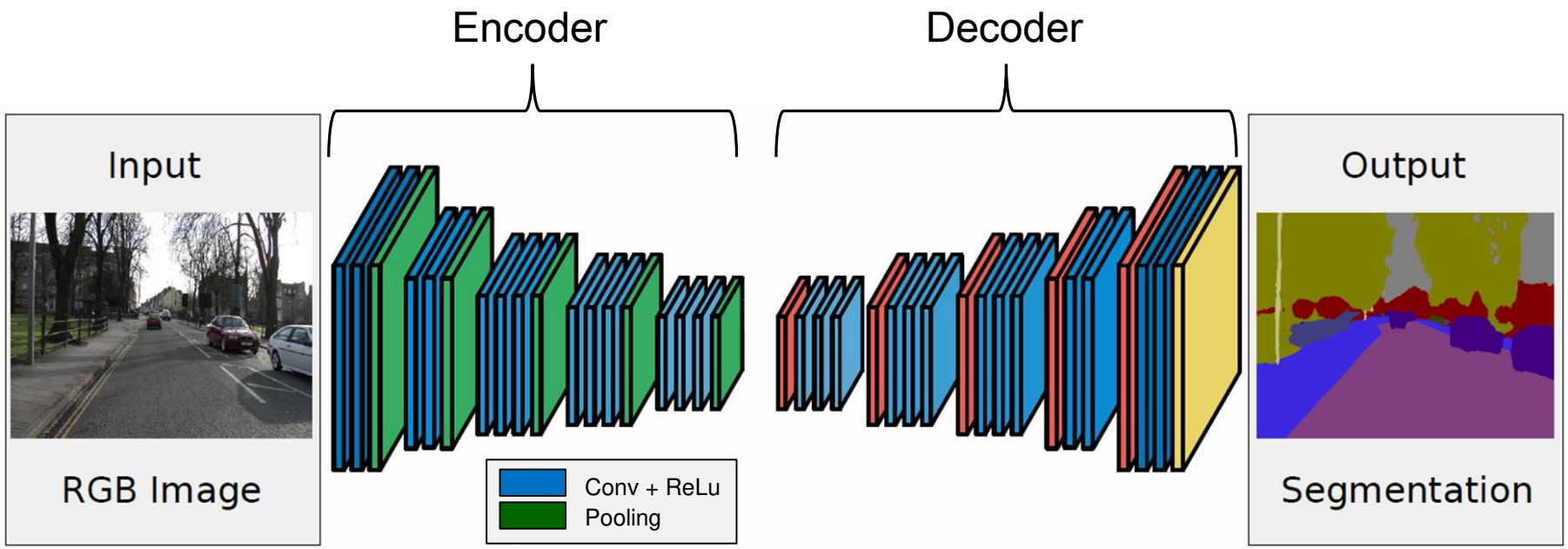
Encoder

- Alle CNN-Layer (jeweils Konvolution + ReLu + Pooling) eines Convolutional Networks lernen zunehmend komprimierte Repräsentationen (Encodings) der Eingabedaten unter Dimensionsreduktion

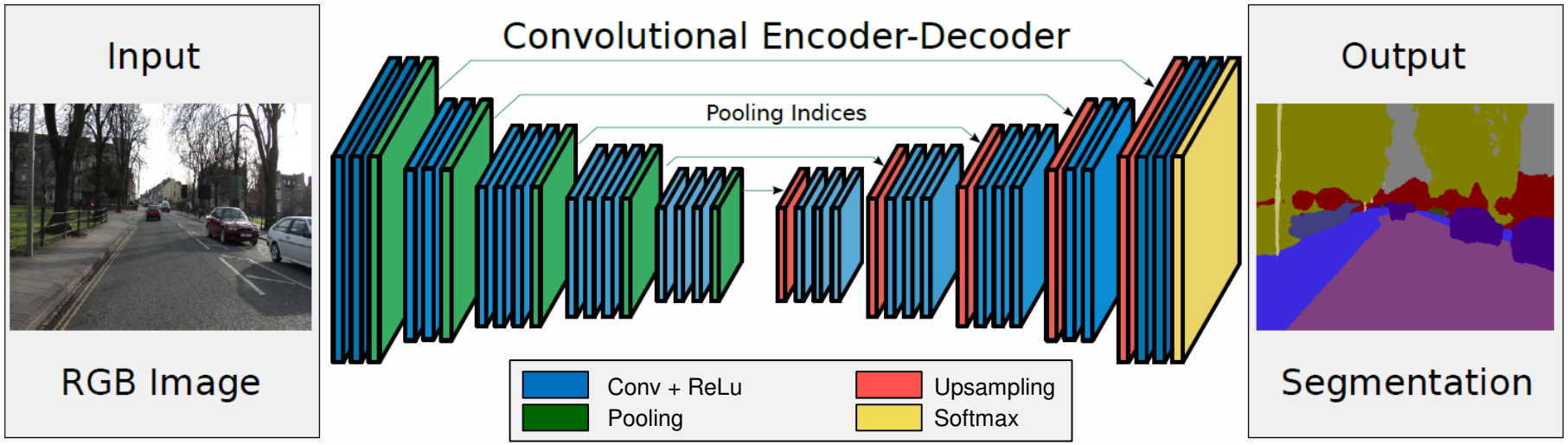


Decoder

- Die Decoder-Layer eines Convolutional Networks müssen lernen, die gelernten komprimierten Repräsentationen unter Dimensionssteigerung wieder auf die Dimensionen der Eingabedaten zu dekodieren



SegNet (2017)

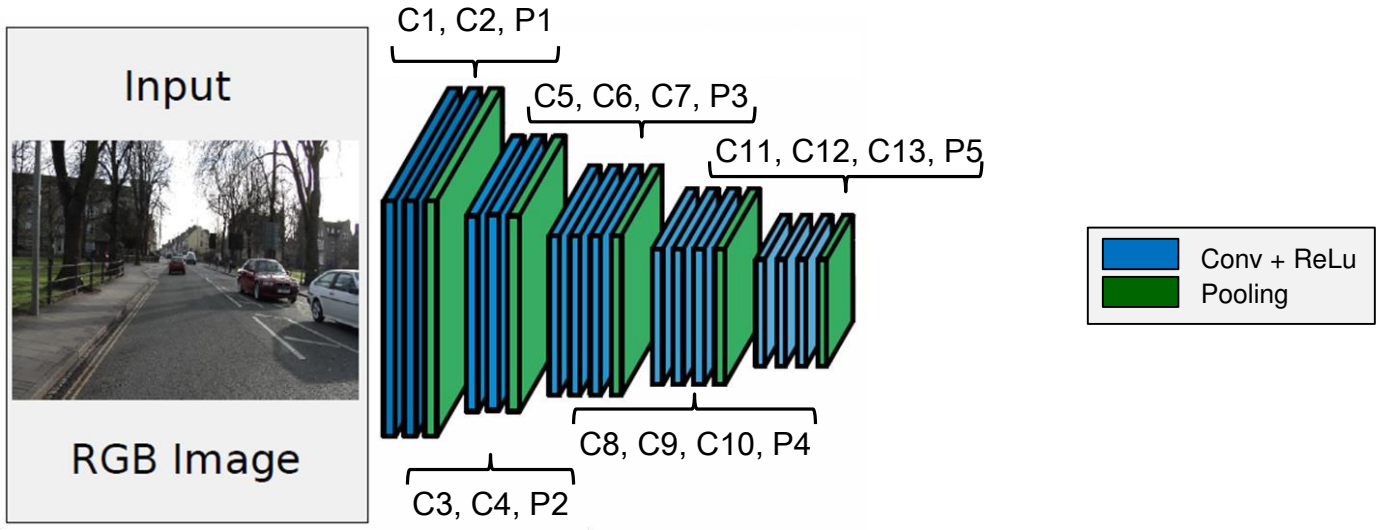


SegNet-Architektur:

- Encoder = Encoder von VGG-16
- Keine Fully Connected Layer wie bei VGG-16
- Decoder = Upsampling auf Layer-Dimensionen des Encoders
- Finale Feature Maps des Decoders → Soft-Max-Klassifikation aller Pixel

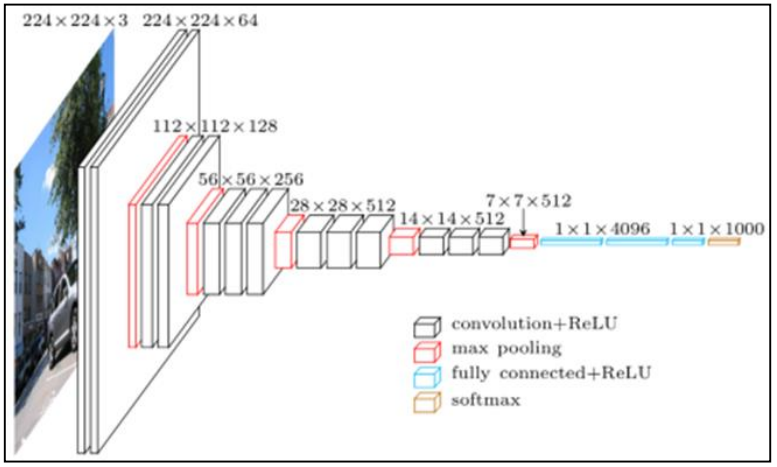
Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." PAMI, 2017.

SegNet: Encoder



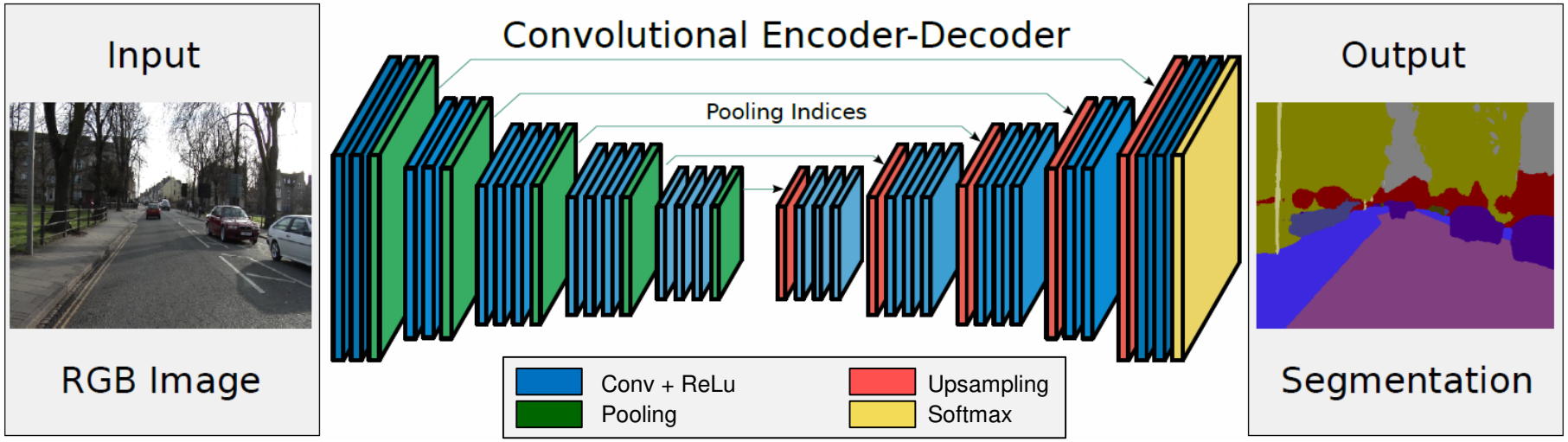
Encoder von VGG-16 (Konfig. D):

- Input: $224 \times 224 \times 3$
- C1, C2: $224 \times 224 \times 64$ P1: $112 \times 112 \times 128$
- C3, C4: $112 \times 112 \times 128$ P2: $56 \times 56 \times 256$,
- C5, C6, C7: $56 \times 56 \times 256$ P3: $28 \times 28 \times 512$
- C8, C9, C10: $28 \times 28 \times 512$ P4: $14 \times 14 \times 512$
- C11, C12, C13: $14 \times 14 \times 512$ P5: $7 \times 7 \times 512$,



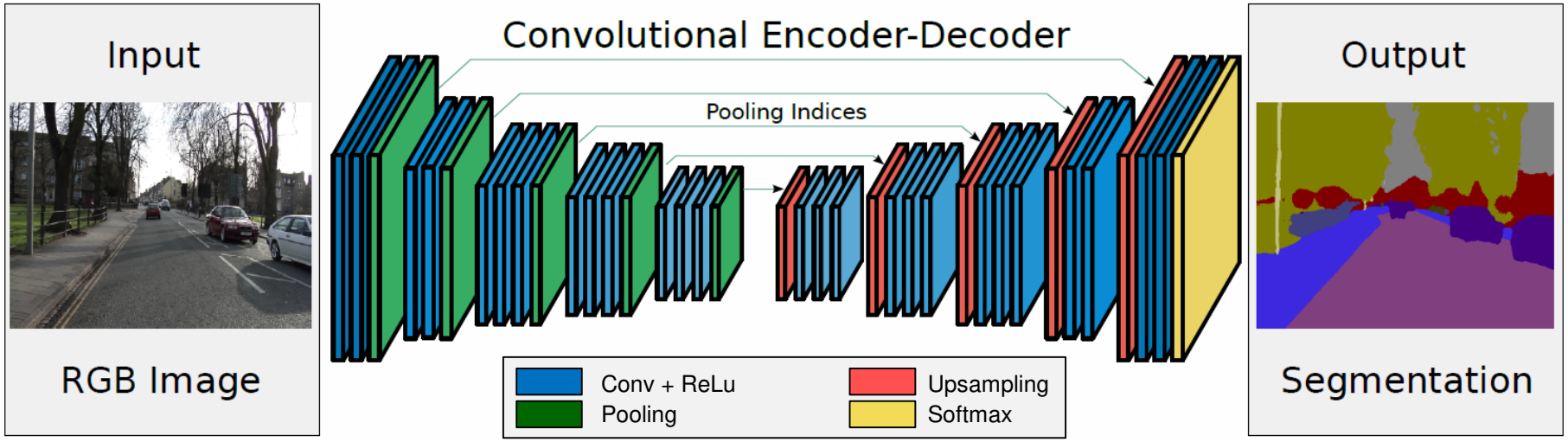
- Alle Konvolutionsfilter: Filtersize = 3×3 mit Stride =1, Padding = 1 sowie ReLu
- Alle Poolings: 2×2-Max-Pooling mit Stride = 2

SegNet: Decoder-Architektur

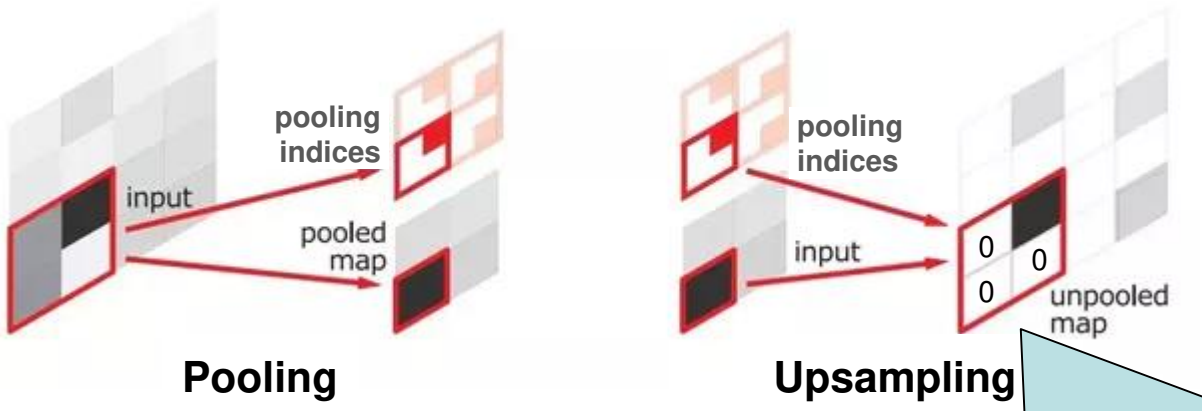


- Pooling bewirkt schrittweise Dimensionsreduktion (Downsampling) von 224×224 Pixeln auf 7×7 Pixel
- Upsampling in SegNet durch schrittweise Dimensionssteigerung von 7×7 Pixel auf 224×224 Pixel
- 5 Pooling-Layern entsprechen 5 Upsampling-Layer

SegNet: Pooling Indices

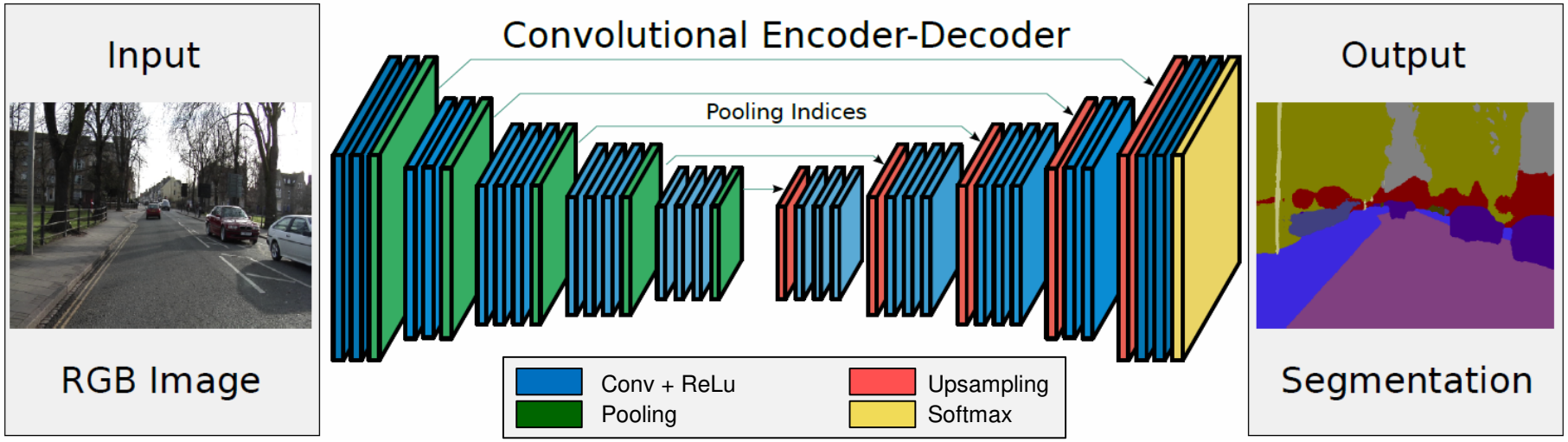


- Für Upsampling werden *Pooling Indices* verwendet



Nur der maximale Wert wird in der dem Pooling-Index entspr. Position eingetragen, die restl. 3 Pixel werden auf Null gesetzt

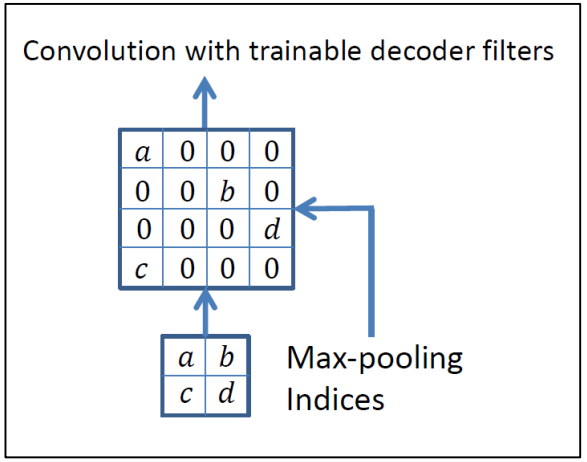
SegNet: Sparse Feature Maps



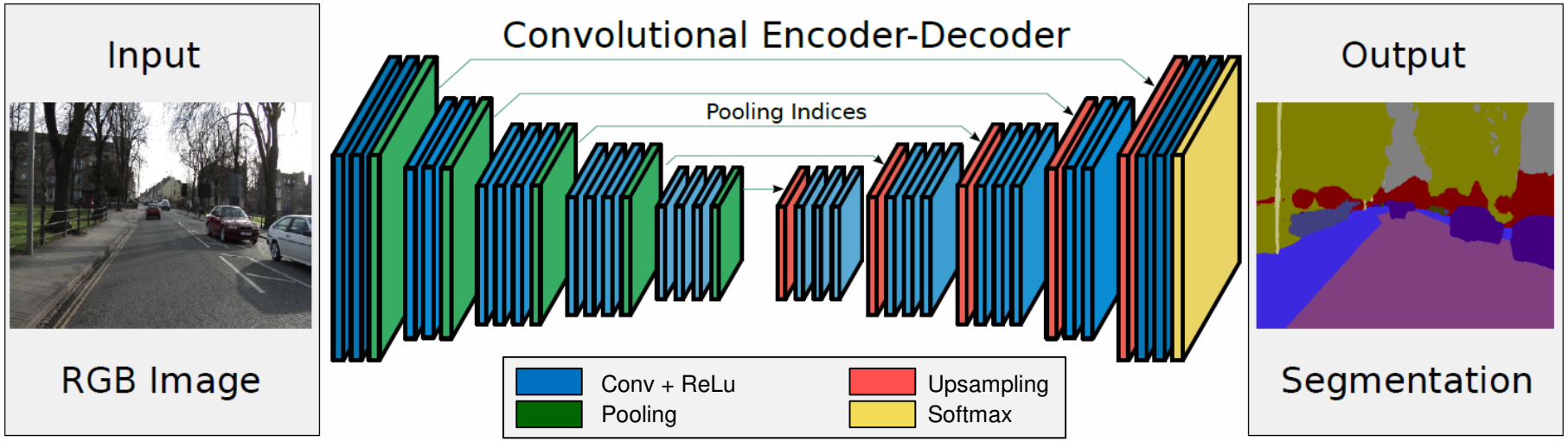
- Das Upsampling kann nur die den Max-Pooling-Indices entsprechenden Pixel rekonstruieren

→ Dies führt zu dünn besetzten Feature Maps (wegen der Null-Einträge für 3 von 4 Pixeln)

- Durch trainierbare Konvolutions-Layer im Dekoder-Teil werden dicht besetzte Feature Maps erzeugt



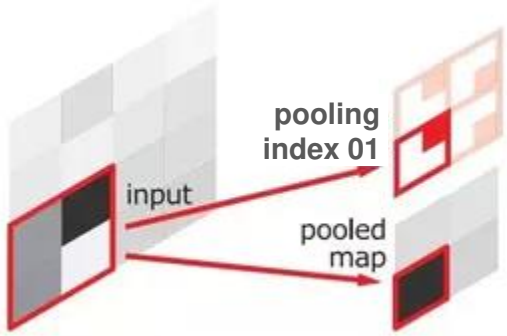
SegNet: Coding Pooling Indices



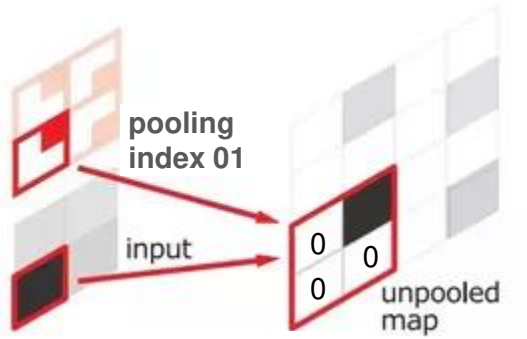
- Speicheraufwand für Pooling Indices: 2 bit / 2×2-Pooling-Fenster

00	01
10	11

Pooling Indices

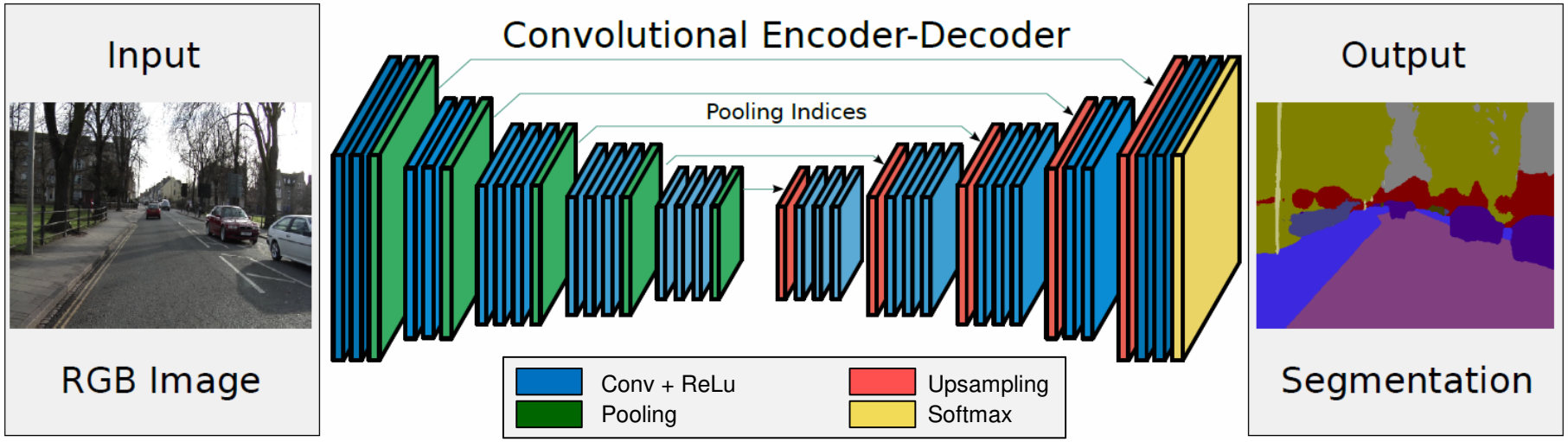


Pooling



Upsampling

SegNet: Softmax



- Ausgabe der pixelweisen Klassifikation sind *K* Layer für *K* Objektklassen
- Segmentierung resultiert für jedes Pixel aus den maximalen W‘keiten unter den *K* Layern

SegNet: Evaluierung (1)

- Global accuracy: 84,0 %
- Mean class accuracy: 54,6 %
- Mean Intersection over Union (mIoU): 46,3 %
- #Parameter: 1,425 Mio

Network	Forward pass(ms)	Backward pass(ms)	GPU training memory (MB)	GPU inference memory (MB)	Model size (MB)
SegNet	422.50	488.71	6803	**1052**	117
DeepLab-LargeFOV [3]	**110.06**	**160.73**	**5618**	1993	83
FCN (learnt deconv) [2]	317.09	484.11	9735	1806	539
DeconvNet [4]	474.65	602.15	9731	1872	877

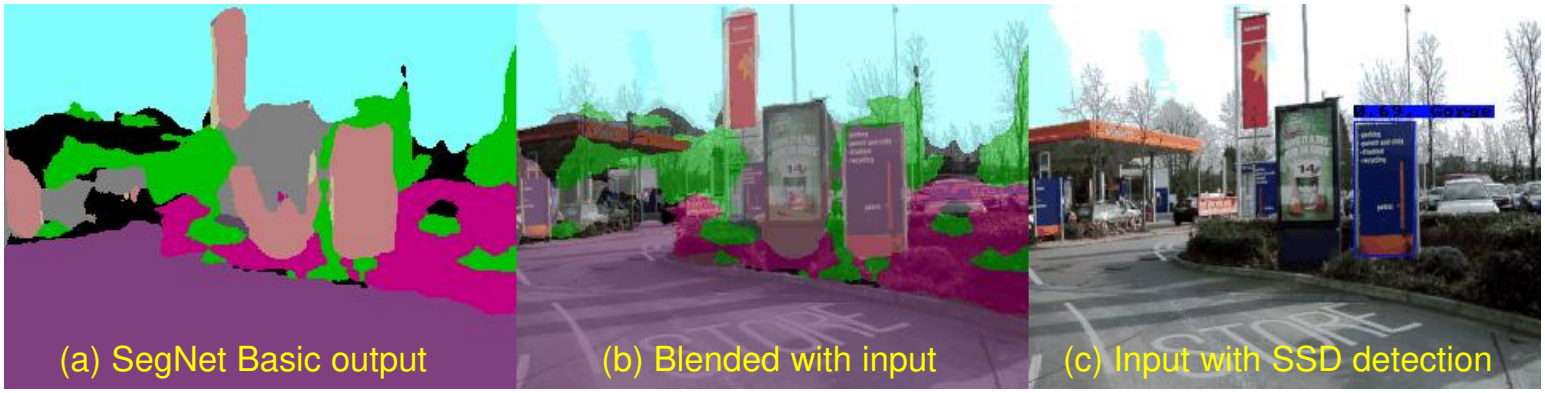
TABLE 6
A comparison of computational time and hardware resources required for various deep architectures. The caffe time command was used to compute time requirement averaged over 10 iterations with mini batch size 1 and an image of 360×480 resolution We used nvidia-smi unix command to compute memory consumption. For training memory computation we used a mini-batch of size 4 and for inference memory the batch size was 1. Model size was the size of the caffe models on disk. SegNet is most memory efficient during inference model.

Global accuracy: Anteil aller korrekt klassifizierten Pixel
Mean class accuracy: Anteil aller korrekt klassifizierten Pixel pro Objektklasse gemittelt über alle Klassen
Mean IoU: Intersection over Union gemittelt über alle Klassen

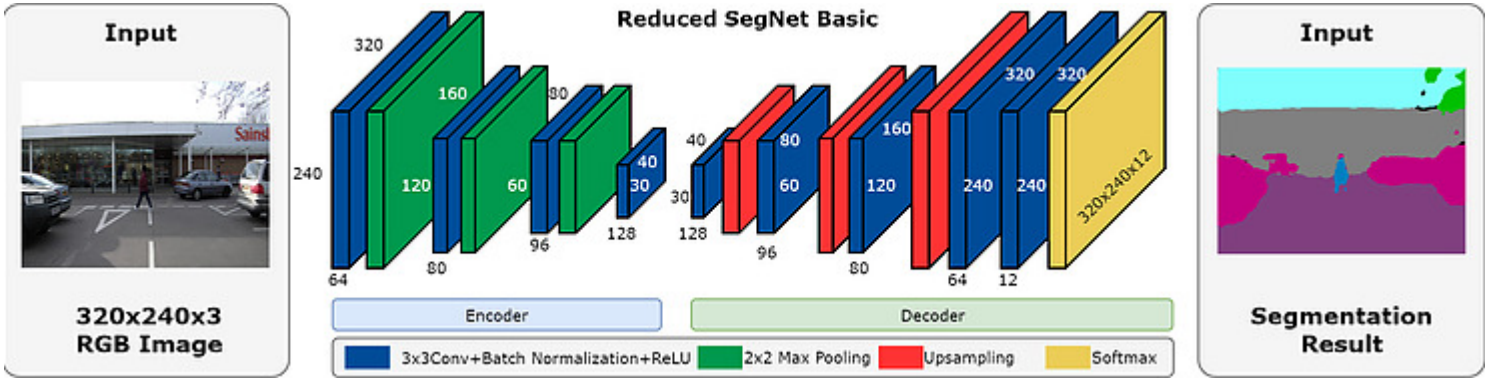
Intersection over Union: $\mathrm{IoU} = \frac{|T \cap D|}{|T \cup D|}$ mit T = wahrem Segment, D = erkanntem Segment, |.| = Fläche in Pixeln

SegNet: Evaluierung (2)

- Runtime performance
  - SegNet Basic: 250ms/frame (4 frames/sec)
  - SSD: 50ms/frame (20 frames/sec)



- SSD: a light SegNet implementation running on an AI accelerator: DV700 FPGA



Bildquellen: https://www.cyberailab.com/home/segnet-an-image-segmentation-neural-network (14.01.2019)

Zusammenfassung

- CNNs zur Bildklassifikation zeigen
  - einen Encoder-Teil der zunehmend abstrakte und globale Features bei gleichzeitigem Downsampling der Bildrepräsentation durch Pooling umsetzt
  - einen anschließenden Klassifikationsteil, der durch eine vollständig verbundene Feed-Forward-Architektur umgesetzt wird
  - Die Parameter der Encoder- und der Fully-Connected-Layer sind zusammen „End-to-End“-trainierbar: zum Training werden nur annotierte Bilder benötigt (Annotation = Klassenzuweisung des gesamten Bildes)

- CNNs zur Bildsegmentierung werden als Fully Convolutional Networks (FCN) umgesetzt. Sie zeigen
  - einen Encoder-Teil
  - ein Decoder-Teil zum schrittweisen Upsampling der downgesampelten Konvolutionsschichten des Encoderteils und zur pixelweise Klassifikation
  - Die Parameter der Encoder- und der Decoder-Layer eines FCNs sind zusammen „End-to-End“-trainierbar: zum Training werden Bilder und ihre Segmentierungen benötigt (Annotation hier = Klassenzuweisung pro Pixel)