EMB-Lab, Fak. N

Activations & Applications

DLM – Deep Learning Methoden

Benjamin Kraus

Kevin Höfle, Prof. Dr. Marcus Vetter

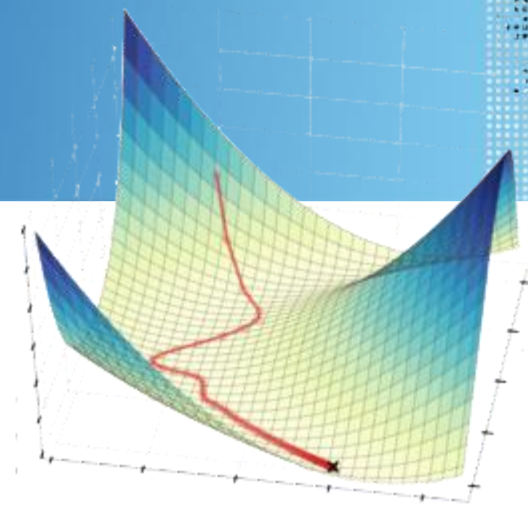
Mannheim, 11.11.2019

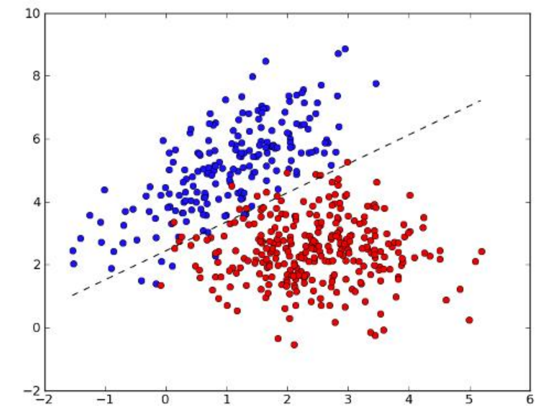
Inhalt

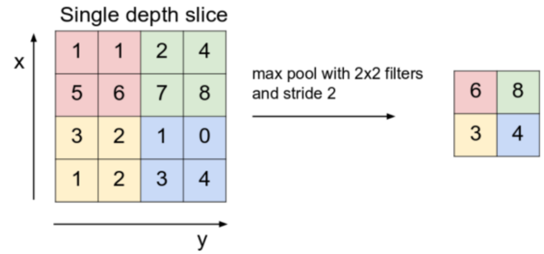
- Wiederholung: Fully Connected Layer, ConvLayer
- Activations
- Batchnorm
- Klassifikation, Regression:
  - Vgg16, GoogLeNet, ResNet
- Segmentierung:
  - U-Net, RCNN, YOLO
- Transfer Learning

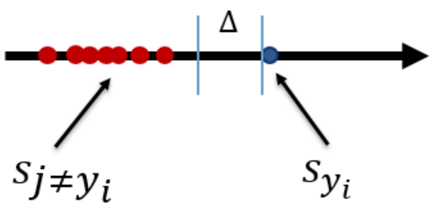
Recap

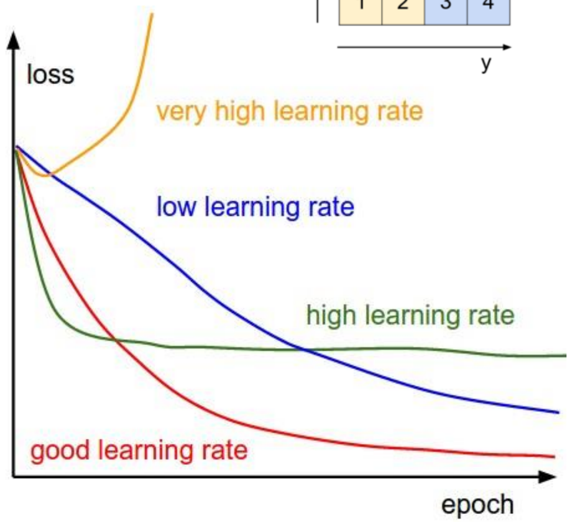
- Lineare Klassifikation
- Loss Function
- Gradienten Abstieg
- Back Propergation

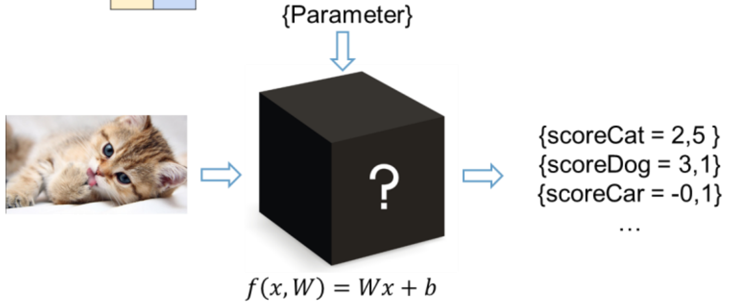


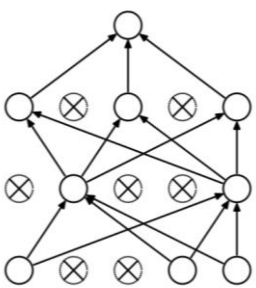


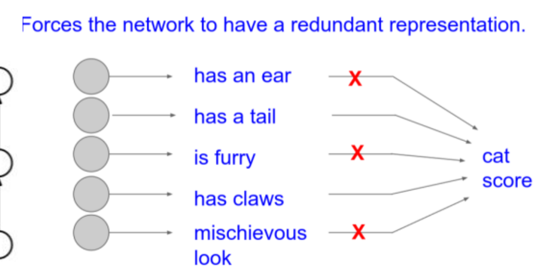












Neuronale Netze Recap

- Zwei Layer NN:
  - $f(x, W_1, W_2) = W_2 * \tanh(W_1 x)$

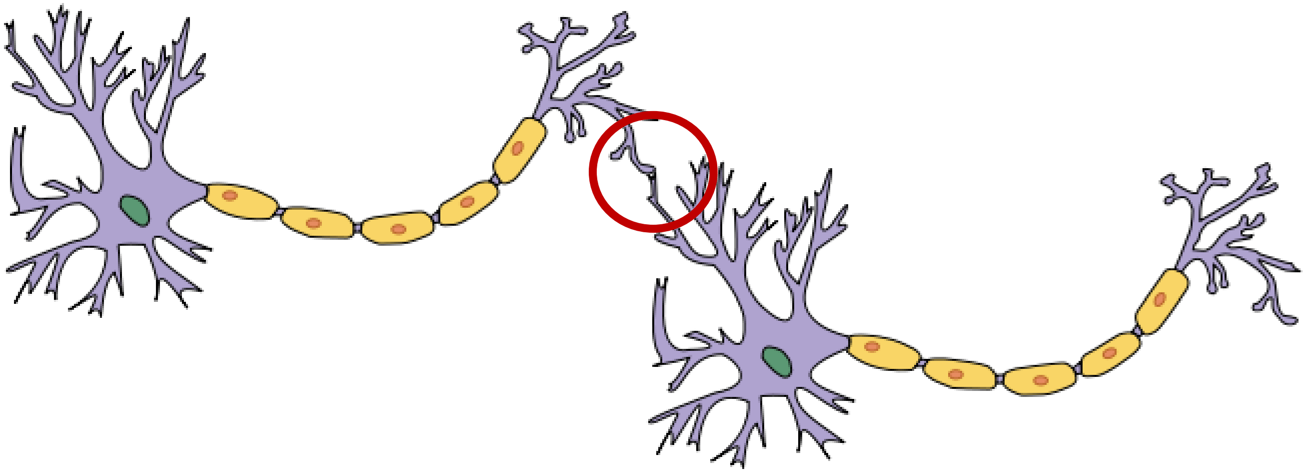
Netz

Gewichte Layer 2

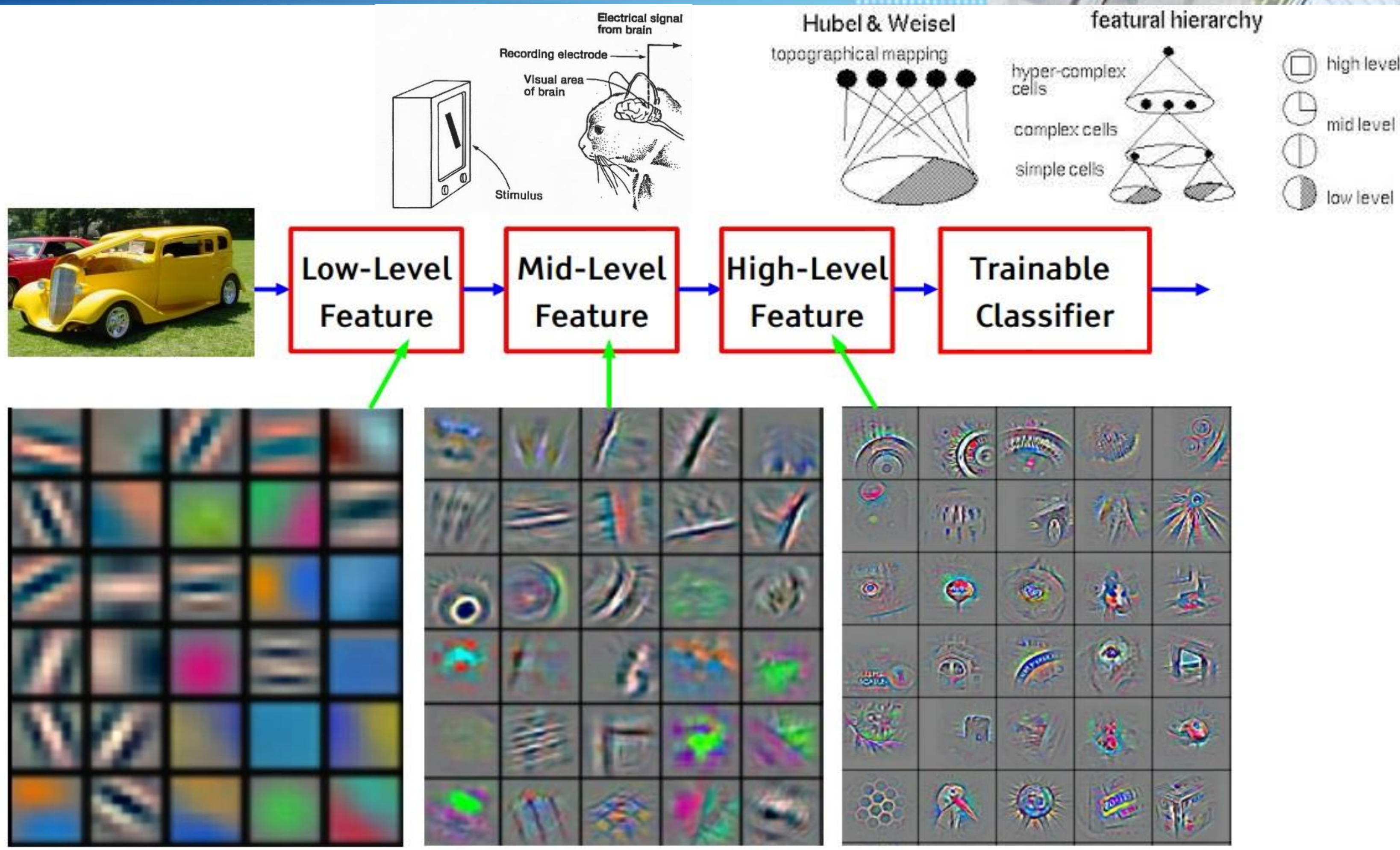
Übertragungs-funktion Layer 1

Gewichte Layer 2

Input

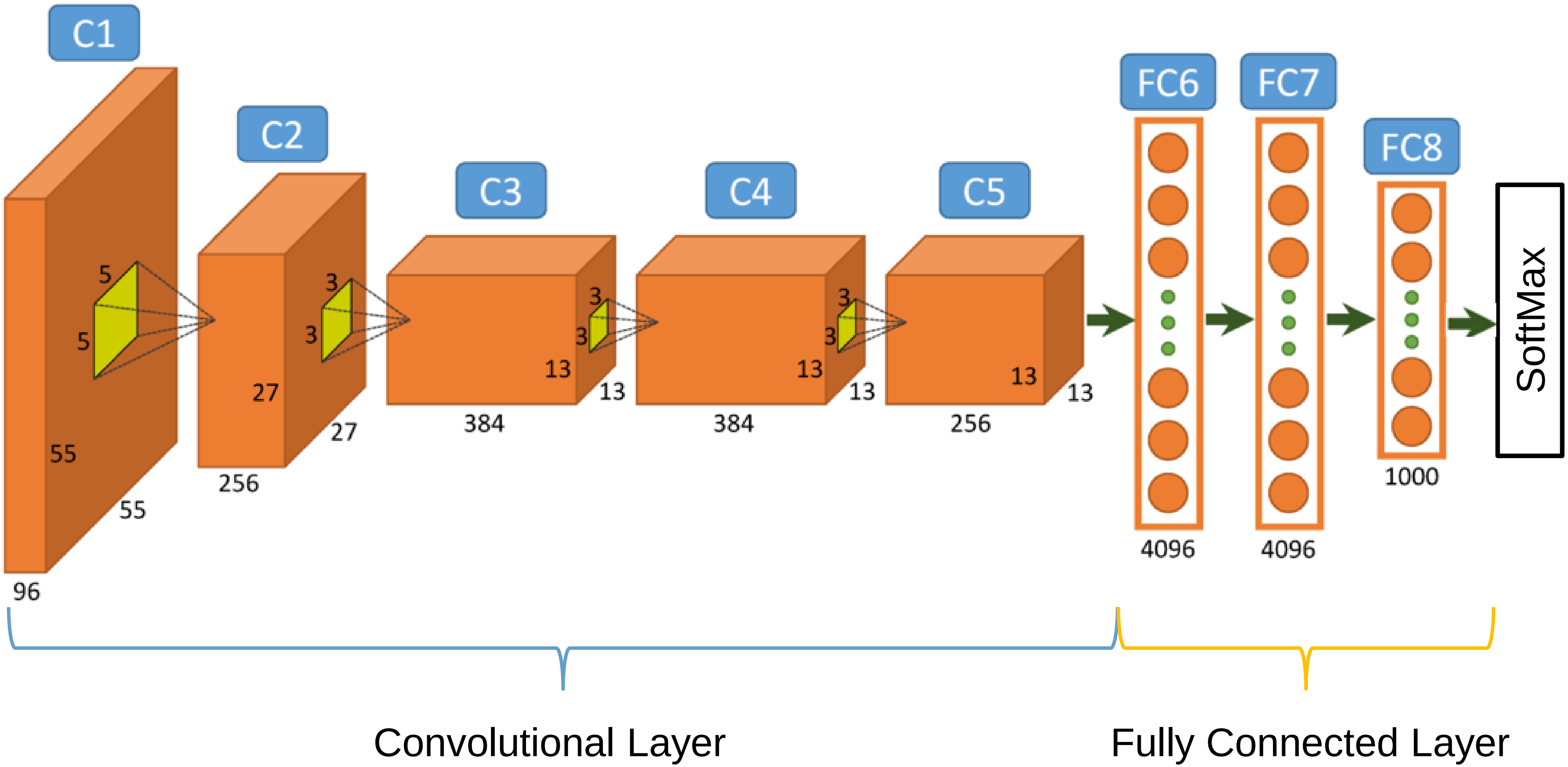


Neuronale Netze Recap



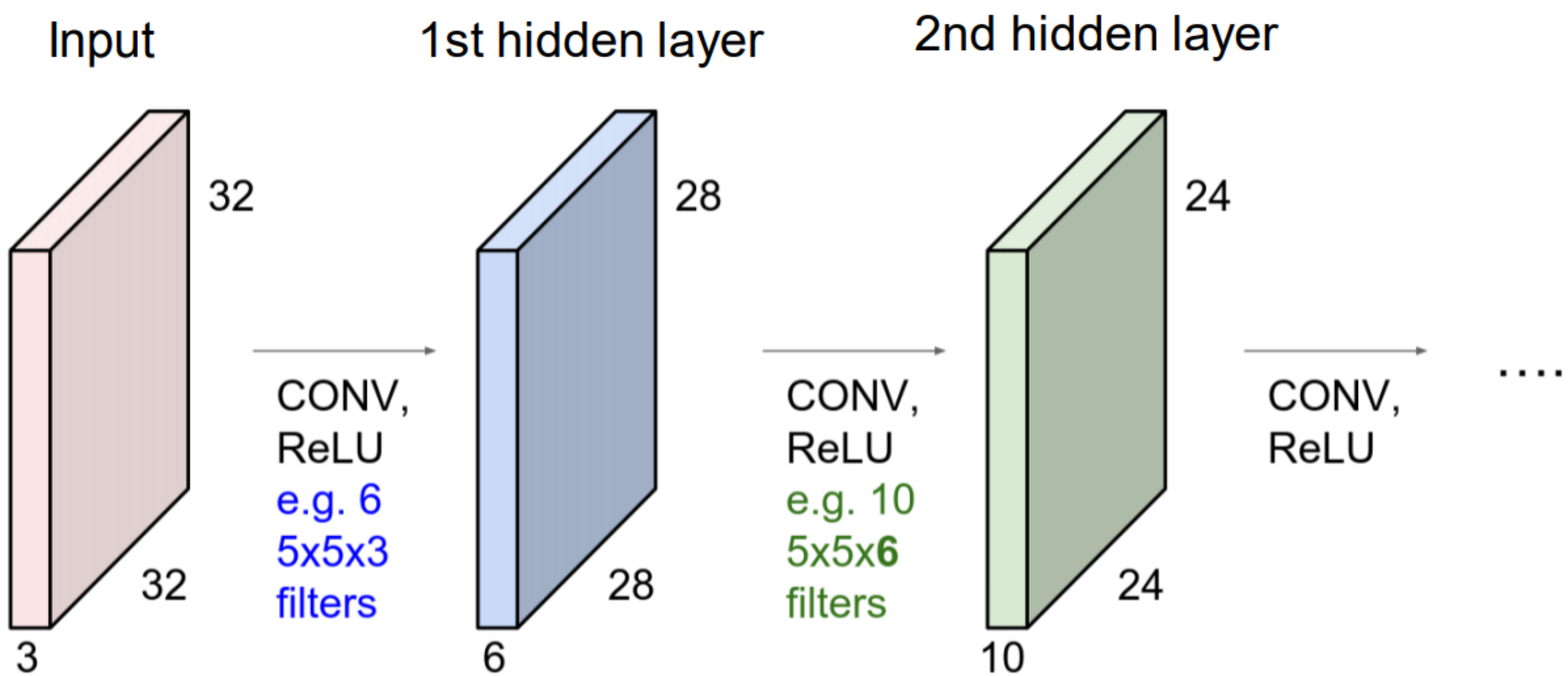
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Neuronale Netze Recap



Neuronale Netze Recap

Ein ConvNet besteht aus einer Folge von ConvLayern verbunden mit Aktivierungsfunktionen

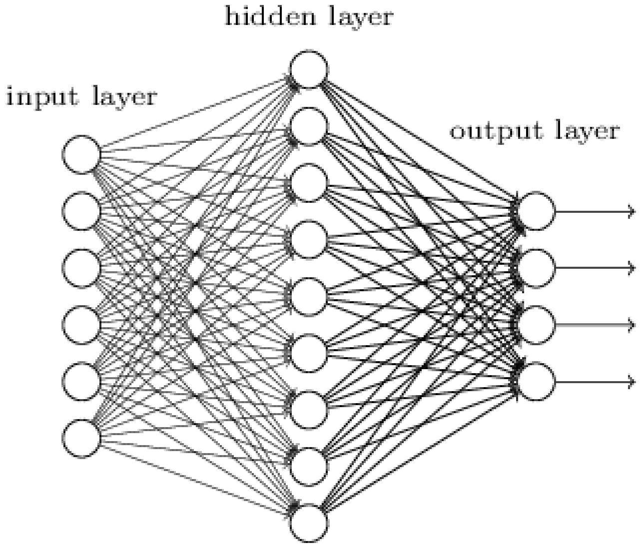


[Karpathy, CS231n]

Layer: Fully Connected

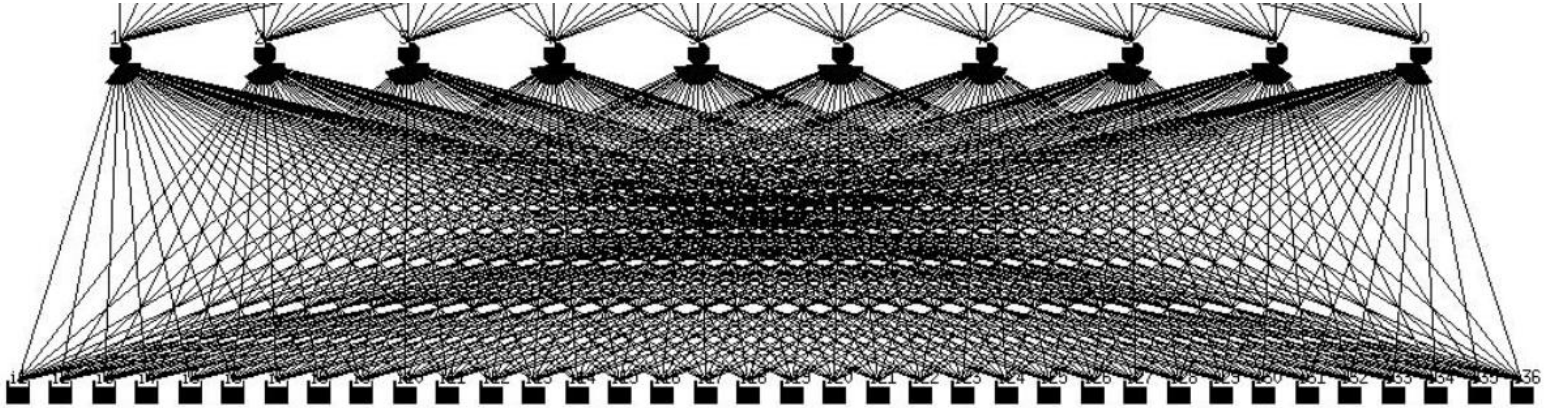
- Auch Dense Layer
- Gewichtete Verknüpfung jedes Eingangs mit jedem Ausgang
  - Sehr viele Gewichte
  - dim(Input)*dim(Output)

$$\sum_{i=1}^{n} xW + b$$



Layer: Fully Connected

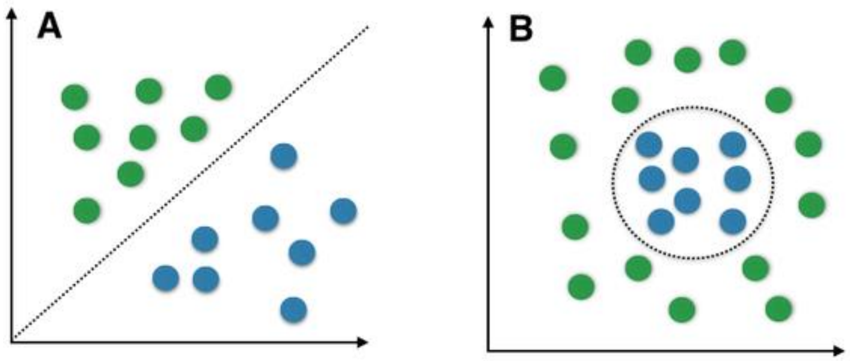
- $W = w_{ij} \in \mathbb{R}^{m \times n}$
- Bsp. Bild 32x32x3 auf 10 Neuronen
  - W hat 30.720 Elemente
- Bei einem UHD-Bild
  - 248.832.000 Gewichte
  - Ca. 1GB bei Float32



[Grafik: Yoshi Komiri]

Layer: Aktivierungsfunktionen

- Lineare und nicht Lineare Probleme



- Wir müssen Nichtlinearität in unseren Klassifikator einführen
- Zwei Möglichkeiten:
  - Kombinationen aus den Eingangsdaten
  - Aktivierungsfunktionen nach den Layern

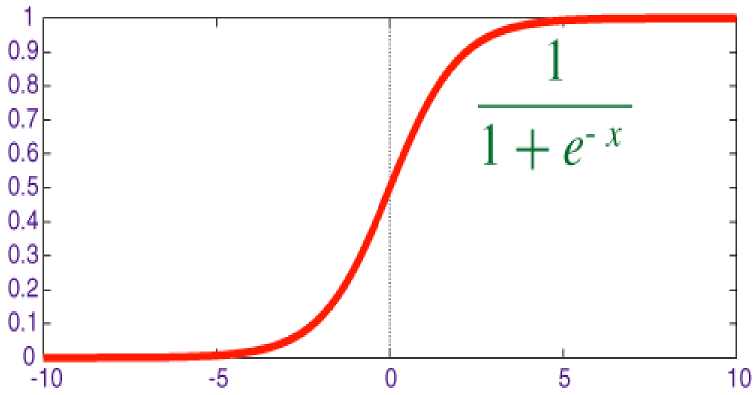
Layer: Aktivierungsfunktionen

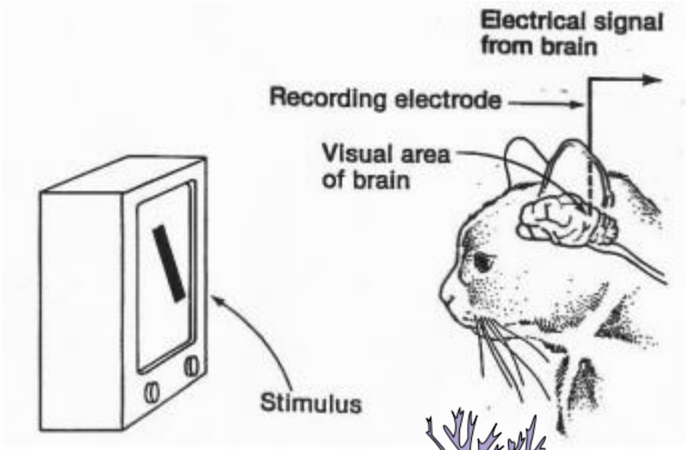
- Activation Functions:
  - Sigmoid $f(x) = 1/(1 + e^{x})$
  - Tanh $f(x) = \ tanh(x)$
  - ReLU $f(x) \ = \ \max(0, x)$
  - Leaky ReLU $f(x) \ = \ \max(ax, x)$

[Daniel Smilkov and Shan Carter]

Layer: Aktivierungsfunktionen

- Sigmoid





“Feuerendes Neuron"

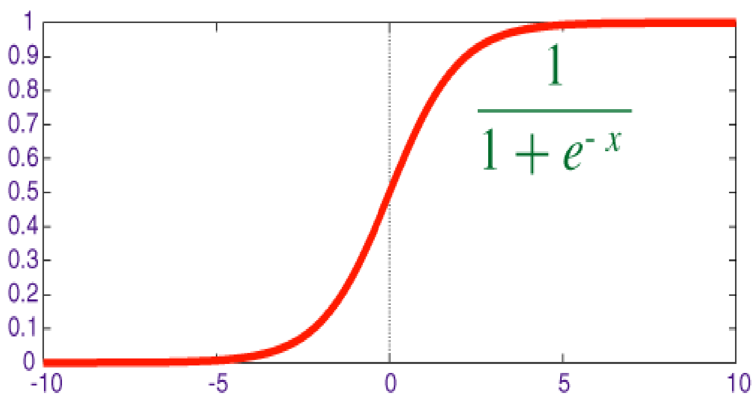
Geht in Sättigung

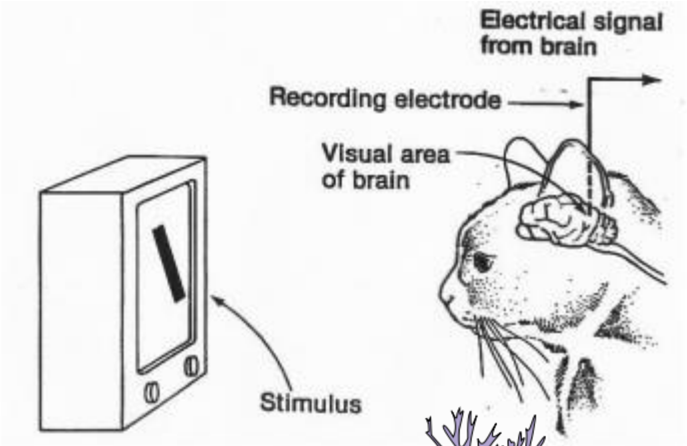
Non zero-centered

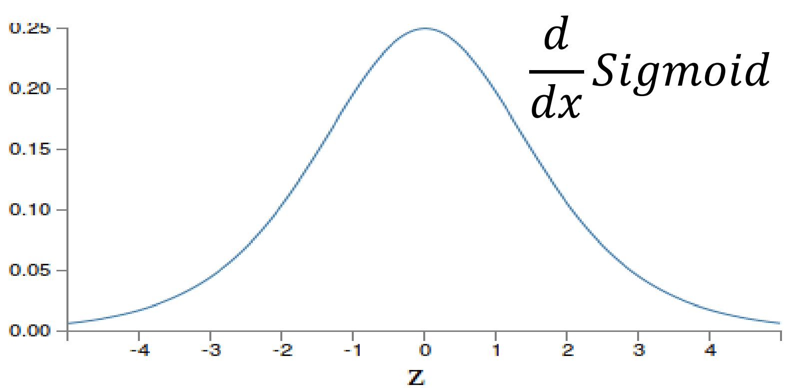
Computationally Expensive

Layer: Aktivierungsfunktionen

- Sigmoid







“Feuerendes Neuron"

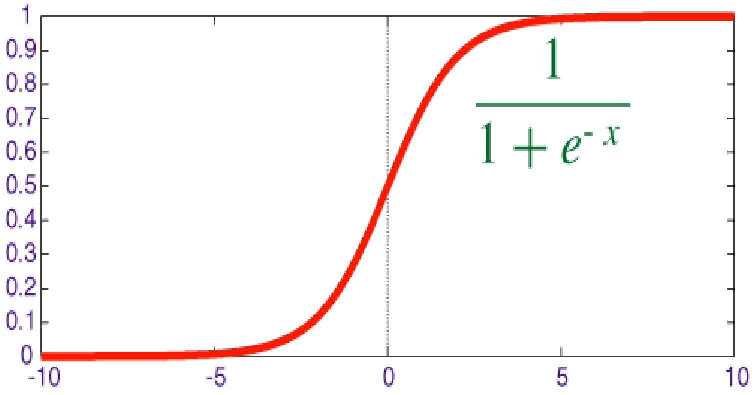
Geht in Sättigung

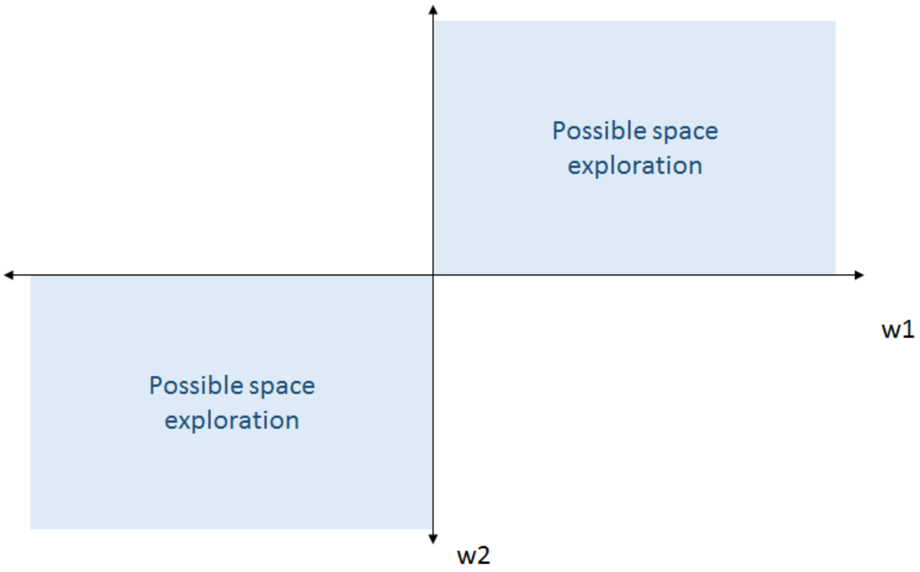
Non zero-centered

Computationally Expensive

Layer: Aktivierungsfunktionen

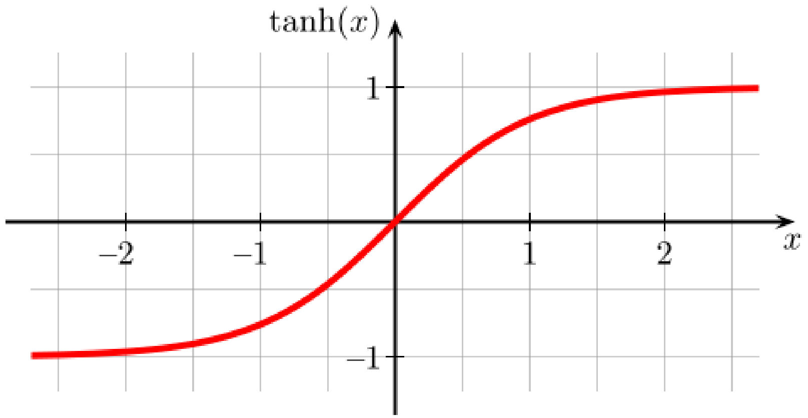
- Sigmoid
  - Non zero-centered
  - Wertebereicht nur Positiv
  - Beschränkt den Freiheitsgrad beim Lernen





Layer: Aktivierungsfunktionen

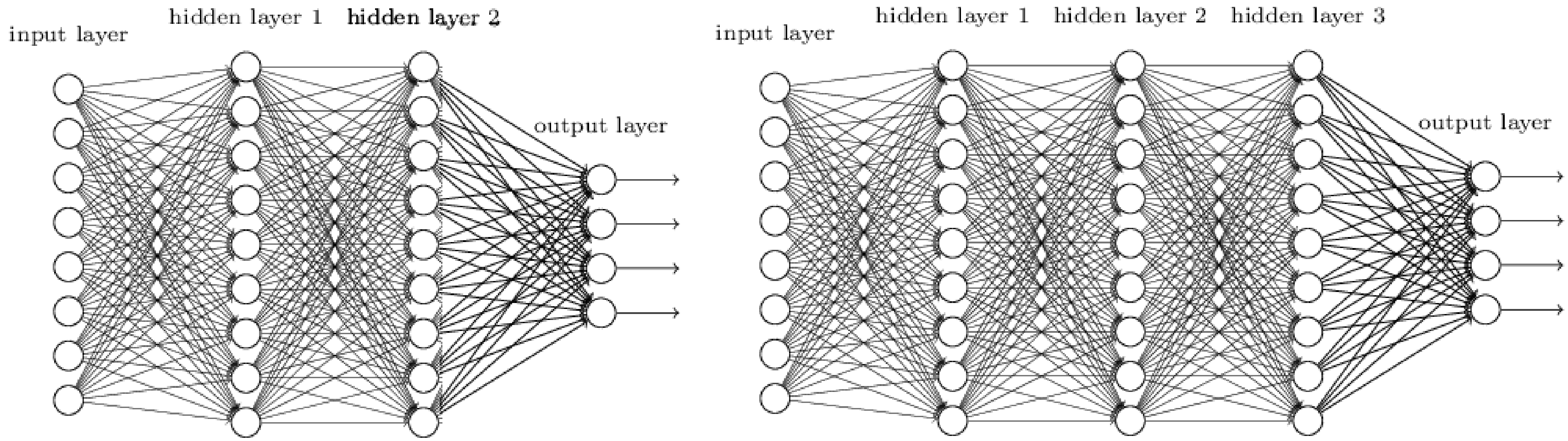
- Tangens hyperbolicus (tanh)



- Sehr ähnlich zur Sigmoid Funktion
- Wertebereich [-1, 1]
- Immernoch Sättigung

Layer: Aktivierungsfunktionen

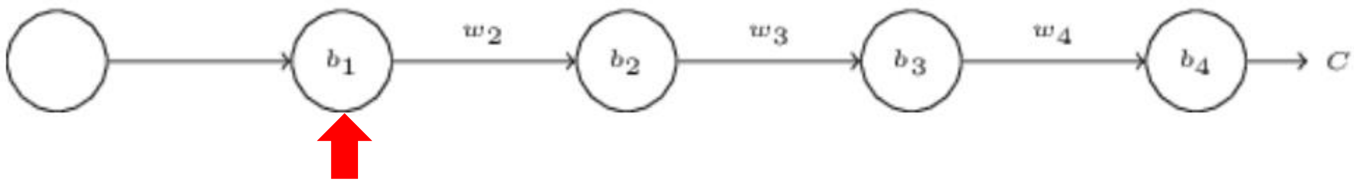
- Sigmoid / tanh
- Was Passiert bei tieferen Netzen?

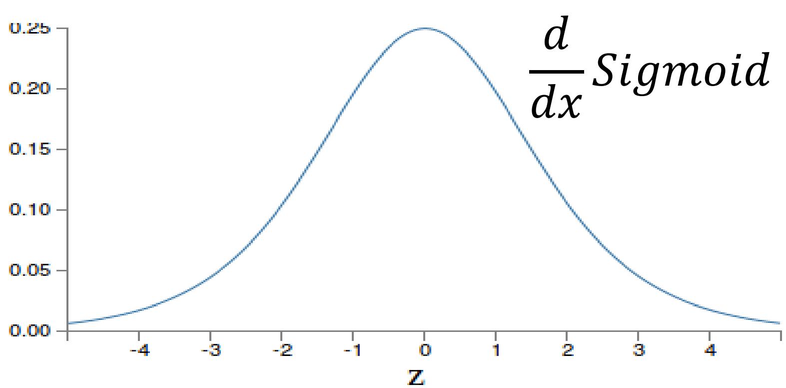


Layer: Aktivierungsfunktionen

- Sigmoid / tanh

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



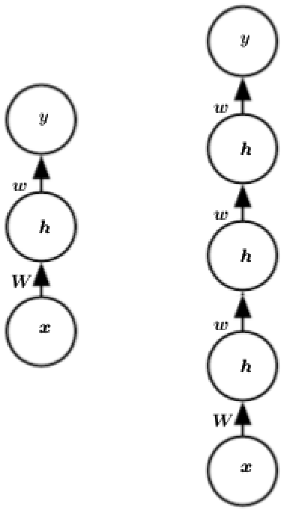


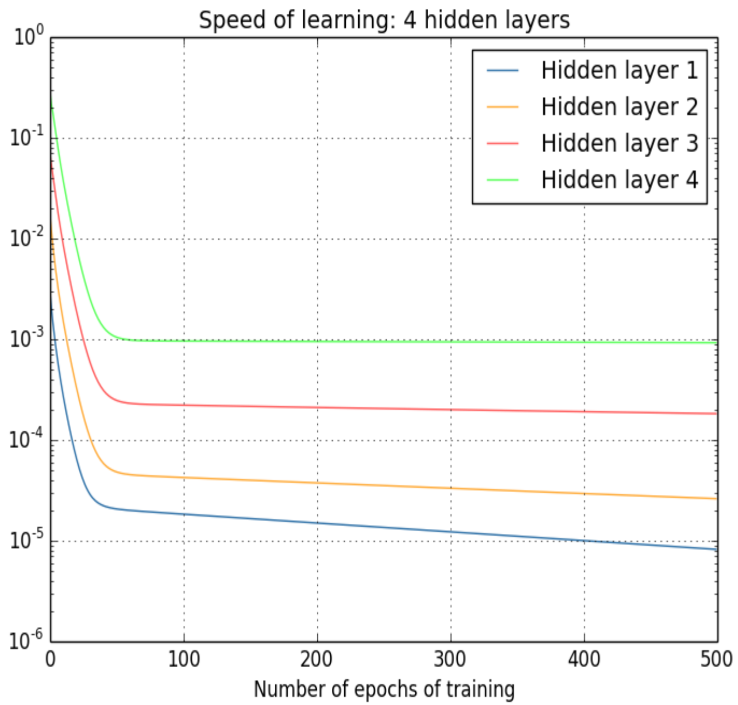
Geht in Sättigung?

Was bedeutet das für den Gradient?

Layer: Aktivierungsfunktionen

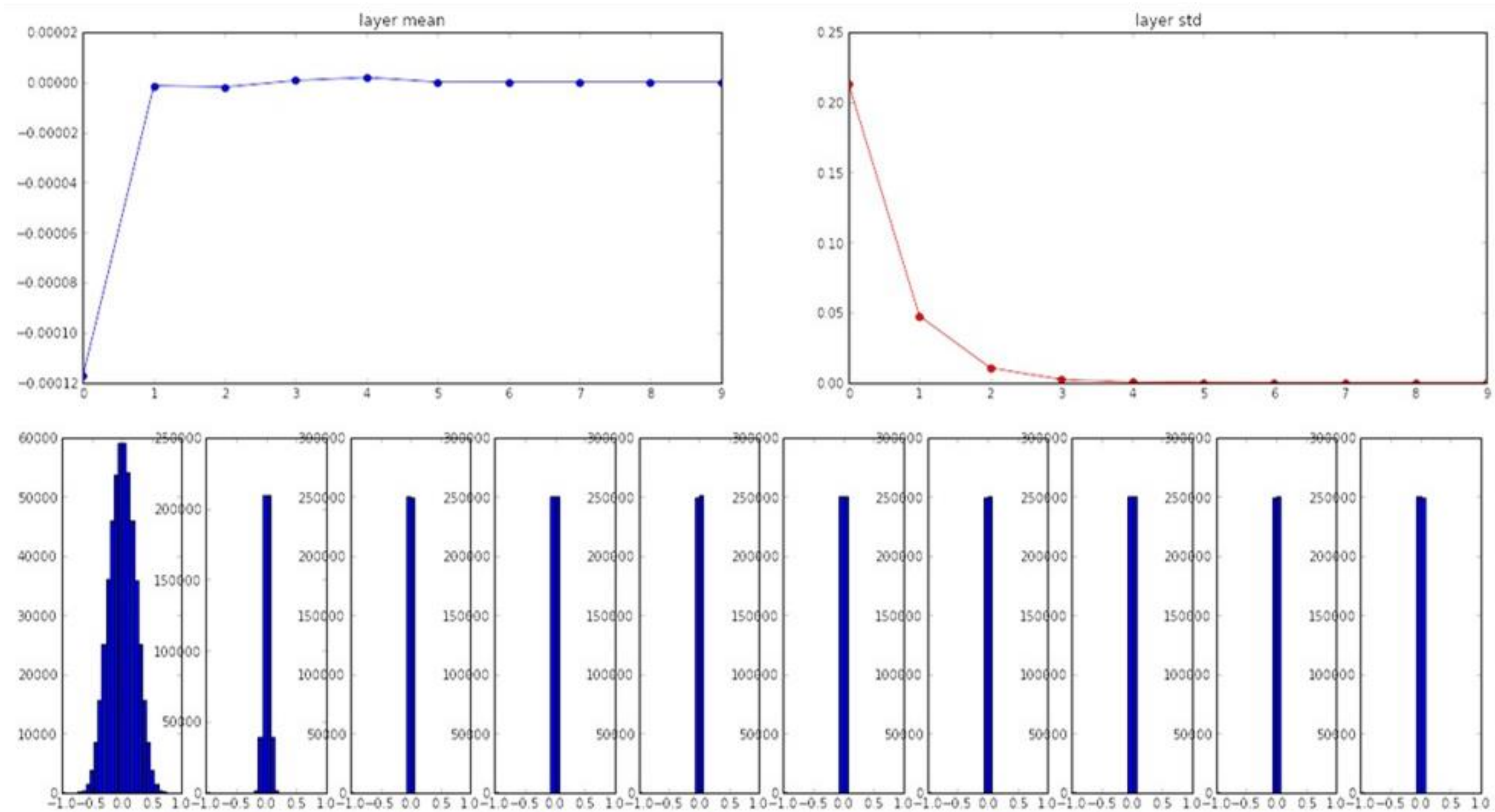
- Sigmoid / tanh
- Was Passiert bei tieferen Netzen?
- Sättigung heißt Gradient ~0
- Die Gradienten werden pro Layer mit einer Zahl >1 multipiziert
- Der weitergegebene Gradient wird immer kleiner





Layer: Aktivierungsfunktionen

- Statistik: 10 Layer mit 500
  - Mittelwert und Standardabweichung der Aktivierungen:



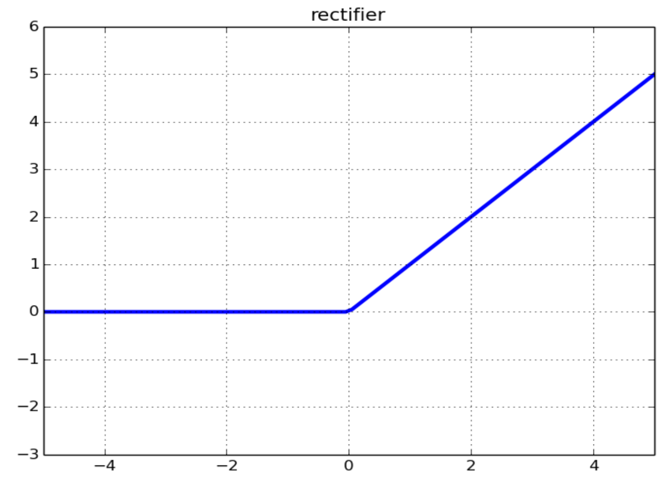
Layer: Aktivierungsfunktionen

- Dieses Problem wird als „Vanishing Gradient“ bezeichnet

- Warum nicht einfach die Gradienten skalieren?
  - Das Gegenteil „Exploding Gradient“ kann entstehen

Layer: Aktivierungsfunktionen

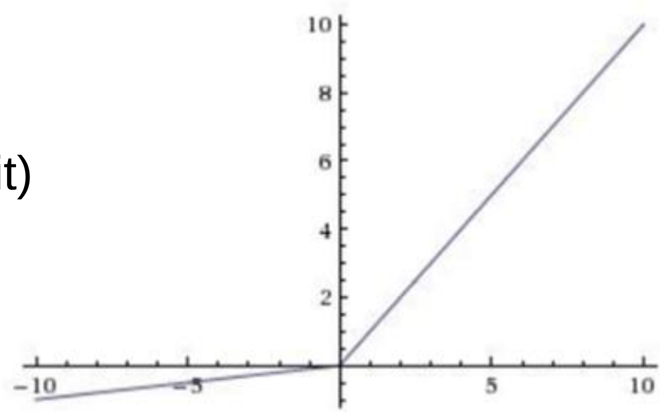
- ReLu
  - Rectified Linear Unit
    - Gradient Null (negativer Bereich)
    - Keine Sättigung (positiver Bereich)
    - Effiziente Berechnung
    - Non zero-centered
    - Konvergiert schnell



max(0, x)

Layer: Aktivierungsfunktionen

- Leaky ReLU
  - $\max(0.01x, x)$
- PReLU (Parametric Rectified Linear Unit)
  - $\max(\alpha x, x)$



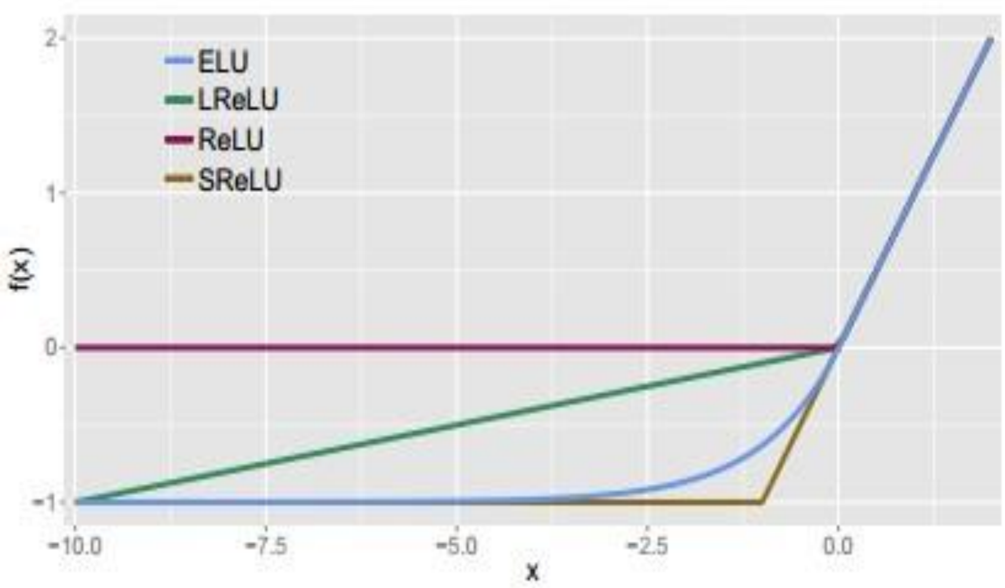
- Gradient != 0 (negativer Bereich)
- Keine Sättigung
- Effiziente Berechnung
- Non zero-centered
- Konvergiert schnell

Layer: Aktivierungsfunktionen

- Exponential Linear Unit (ELU)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

- Gradient != 0 (negativer Bereich)
- Keine Sättigung
- ~~Effiziente Berechnung~~
- Non zero-centered
- Konvergiert schnell



Layer: Aktivierungsfunktionen

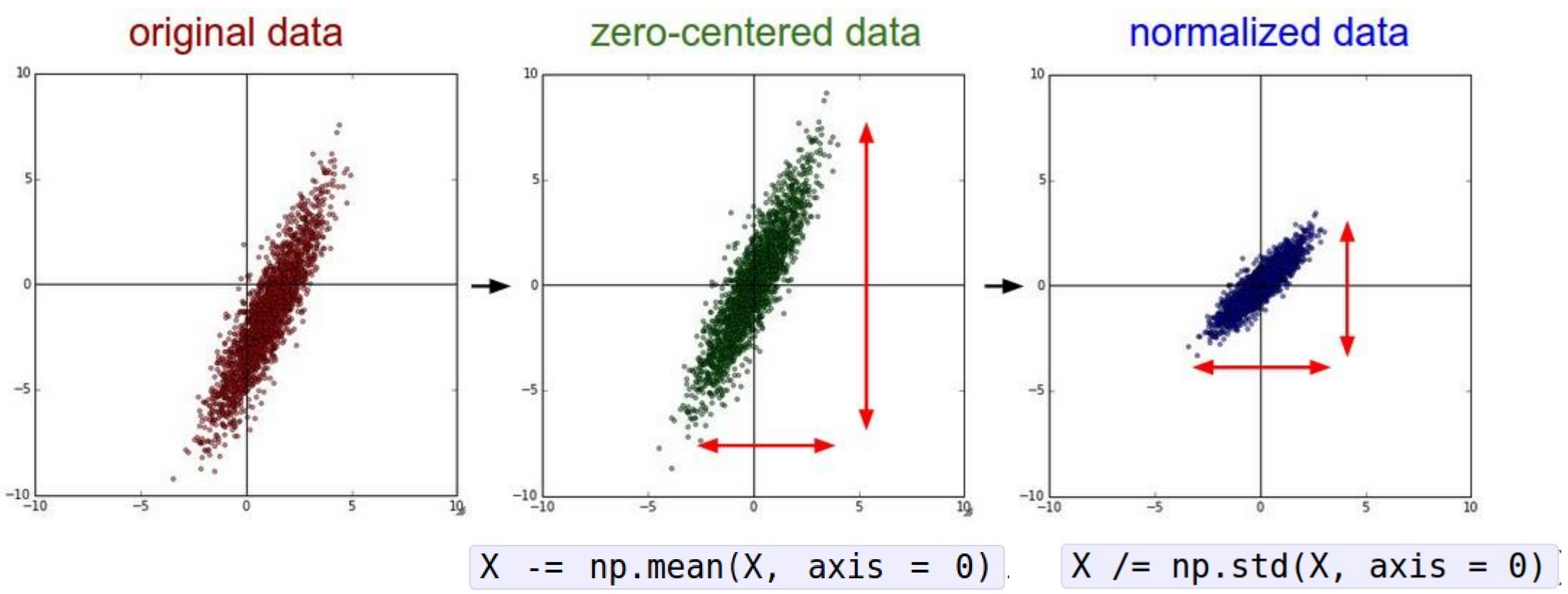
- Aus der Praxis:
  - Im Netzwerk
    - ReLU funktioniert meistens.
    - Leaky ReLU, ELU usw. sind einen Versuch wert, wenn alles andere stimmt.
    - Tanh funktioniert manchmal. Aber nicht oft.
    - Sigmoid noch weniger.
  - Am Ende des Netzwerks (nach dem Letzten Layer):
    - Was ist Ihre Problemstellung? Klassifikation? Regression?
    - Was möchten Sie? Wahrscheinlichleiten? Koordinaten?
    - Dementsprechend wählen.

Batch Normalisierung

- Eingangsdaten:
- Bilder haben oft einen Wertebereich von $[0 \dots 255]$
- Exploding Gradient Problem
- Wir hätten gern Daten von $[-1\ \dots\ 1]$ mit $\sigma = 1$ und $\mu = 0$

Batch Normalisierung

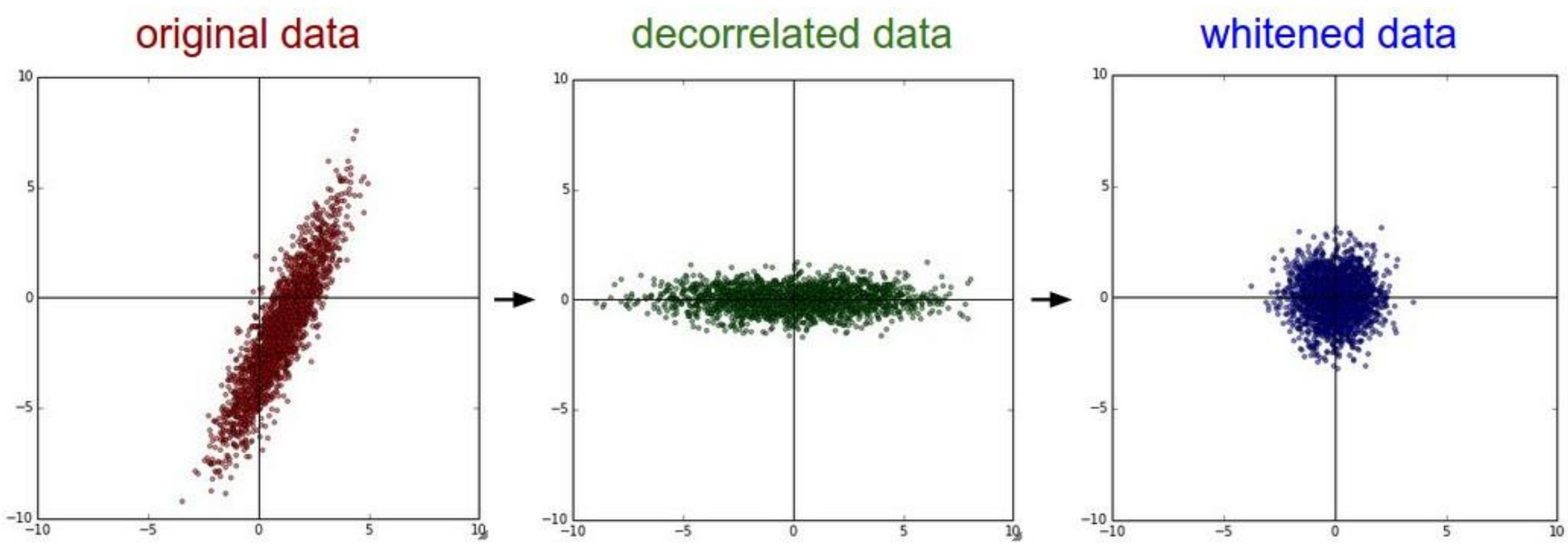
- Wir hätten gern Daten von $[-1\ \ldots\ 1]$ mit $\sigma = 1$ und $\mu = 0$
- Also machen wir das einfach!



[Grafik: Karpathy]

Batch Normalisierung

- Wir können auch noch weitergehen
- Dekorreliern und „weiß machen“
  - z.B. mit einer PCA
  - Nicht sehr verbreitet



Kovarianz Matrix ist nur auf der Hauptachse besetzt

Kovarianz Matix ist eine Einheitsmatrix

[Grafik: Karpathy]

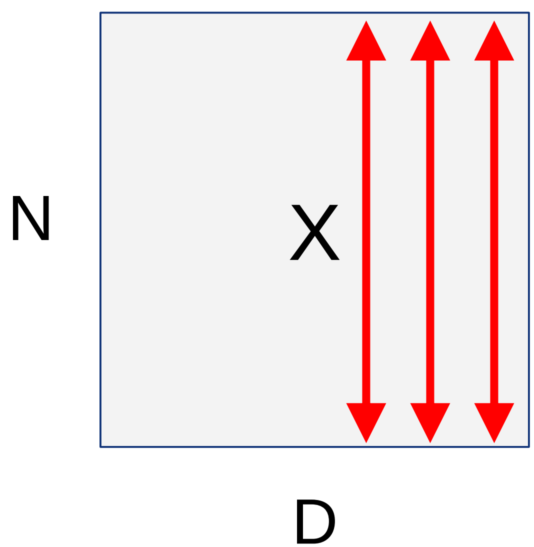
Batch Normalisierung

- Probleme:
  - Was ist im inneren des Netzes?
  - Was ist zur Test Zeit?

Batch Normalisierung

- Was ist im inneren des Netzes?

- Wir können auch die Aktivierungen im Netz normieren
  - Batch Norm Layer



1. Erwartungswert und Varianz für jede Dimension der Akivierung berechen

2. Normalisiern

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

[Grafik: Karpathy; Quelle: Ioffe and Szegedy]

Batch Normalisierung

- Batch Norm Layer

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Ist ableitbar -> BackProp funktioniert
- Wir erlauben es dem Netz $\hat{x}$ zu skalieren

$$y^{(k)} = \gamma^{(k)} \widehat{x}^{(k)} + \beta^{(k)}$$

- Mit γ und β als lernbaren Parametern

Das Netzwerk kann lernen:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

Um eine Einheitsabbildung zu erreichen

[Grafik: Karpathy; Quelle: Ioffe and Szegedy]

Batch Normalisierung

- Batch Norm Layer:
- Macht unsere Initialisierung stabiler
- Werte zwischen -1 und 1 (fast)
- Hilft gegen Vanishing- und Exploding-Gradient

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

[Grafik: Karpathy; Quelle: Ioffe and Szegedy]

Batch Normalisierung

- Was ist zur Test Zeit?
- Wir nutzen eine Schätzung von σ und μ (z.B. das mittel aller Trainingsbatches)

- Wir können das mit einem laufenden Mittelwert anpassen

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

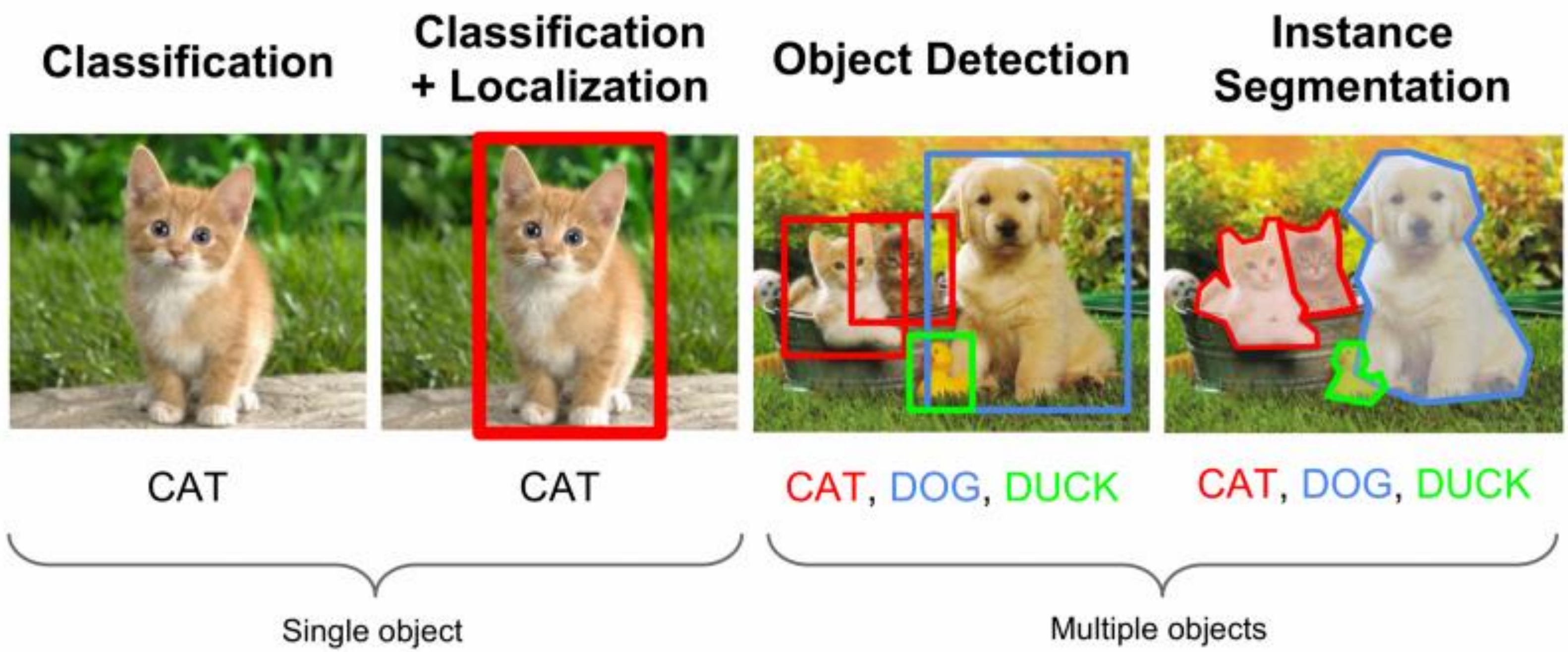
[Grafik: Karpathy; Quelle: Ioffe and Szegedy]

Fragen bis hier?

?

Computer Vision Tasks

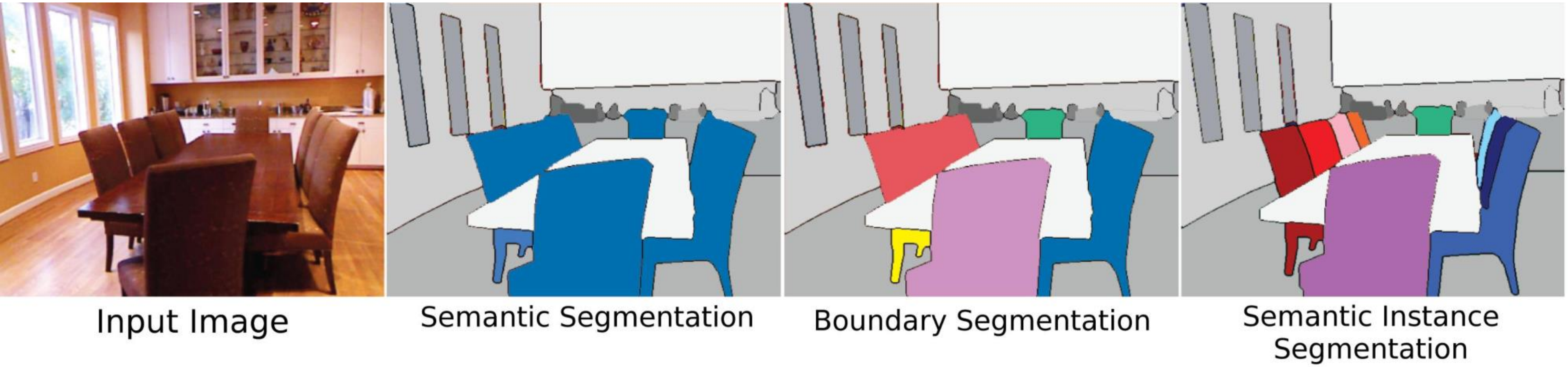
- Klassifikation
- Lokalisation
- Segmentierung



[Grafik: Jibin Mathew]

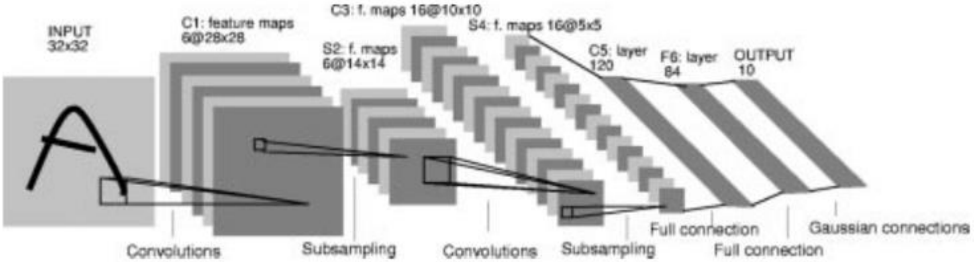
Computer Vision Tasks

- Semantic Segmentation
- Instance Segmentation

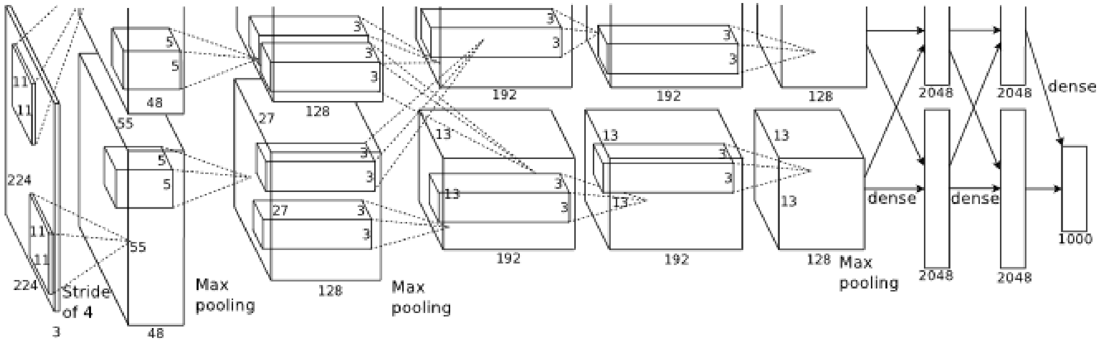


Klassifikatoren

- LeNet

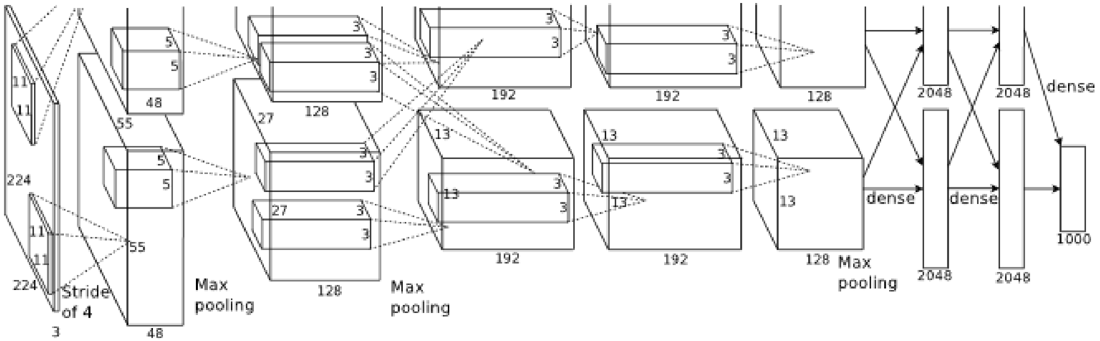


- AlexNet



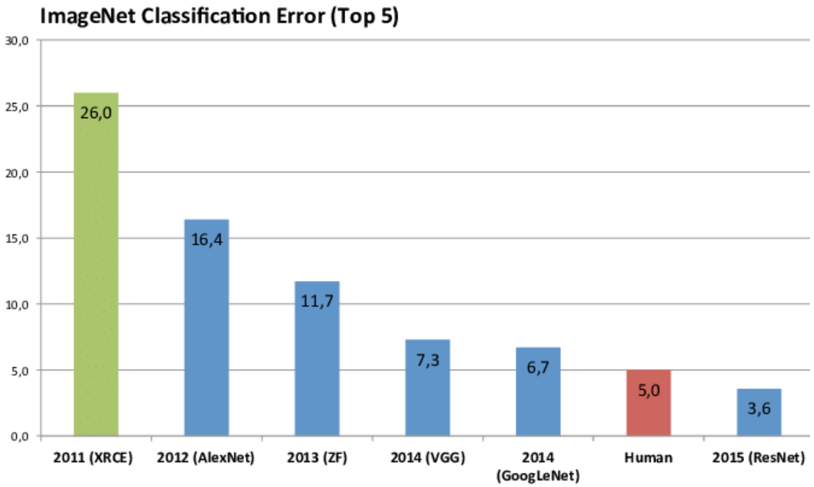
Klassifikatoren

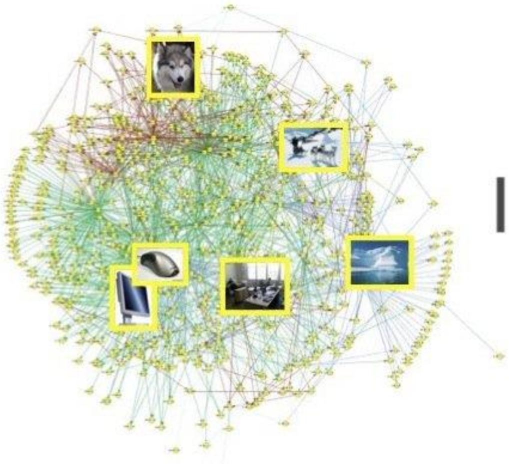
- AlexNet



Wie vergleicht man Klassifikatoren?

- Gewinner der ILSVRC (~1,2M Bilder, 1000 Klassen)

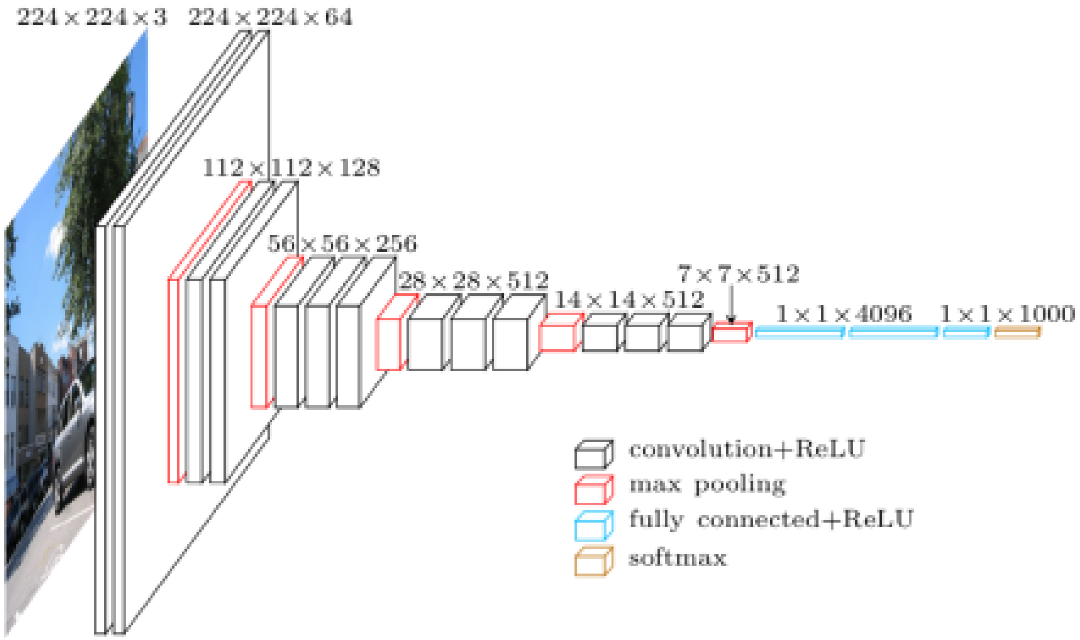




IMAGENET

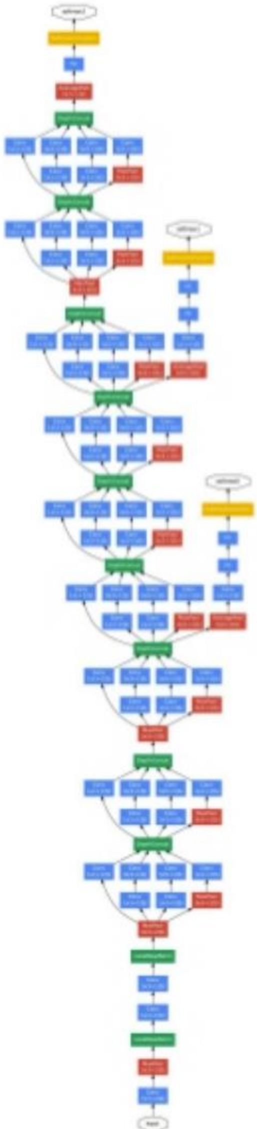
Klassifikator

- VGG16
- Sehr Einfach
- Conv -> ReLu -> Pool



[Grafik: Davi Frossard]

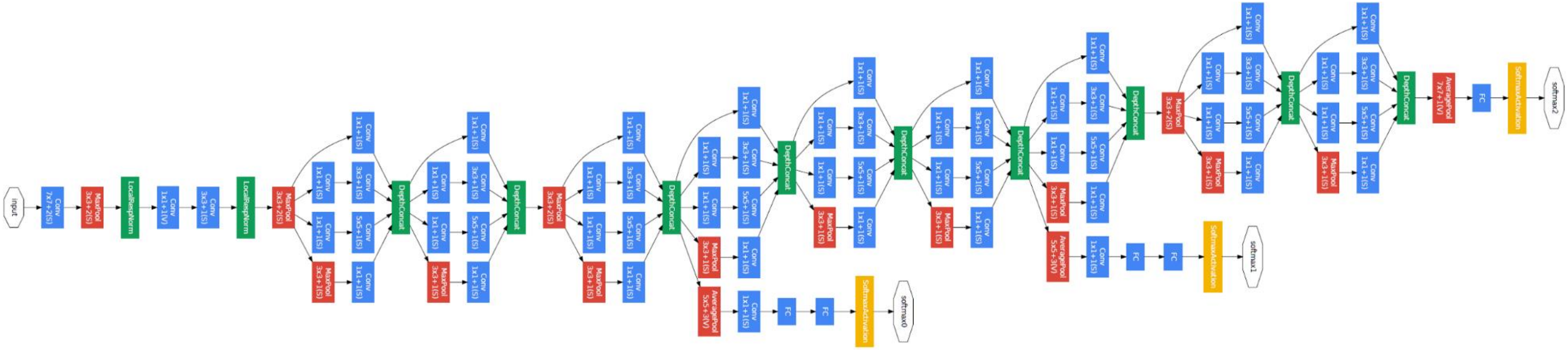
ImageNet Challenge: 2014



[Karpathy, CS231n]

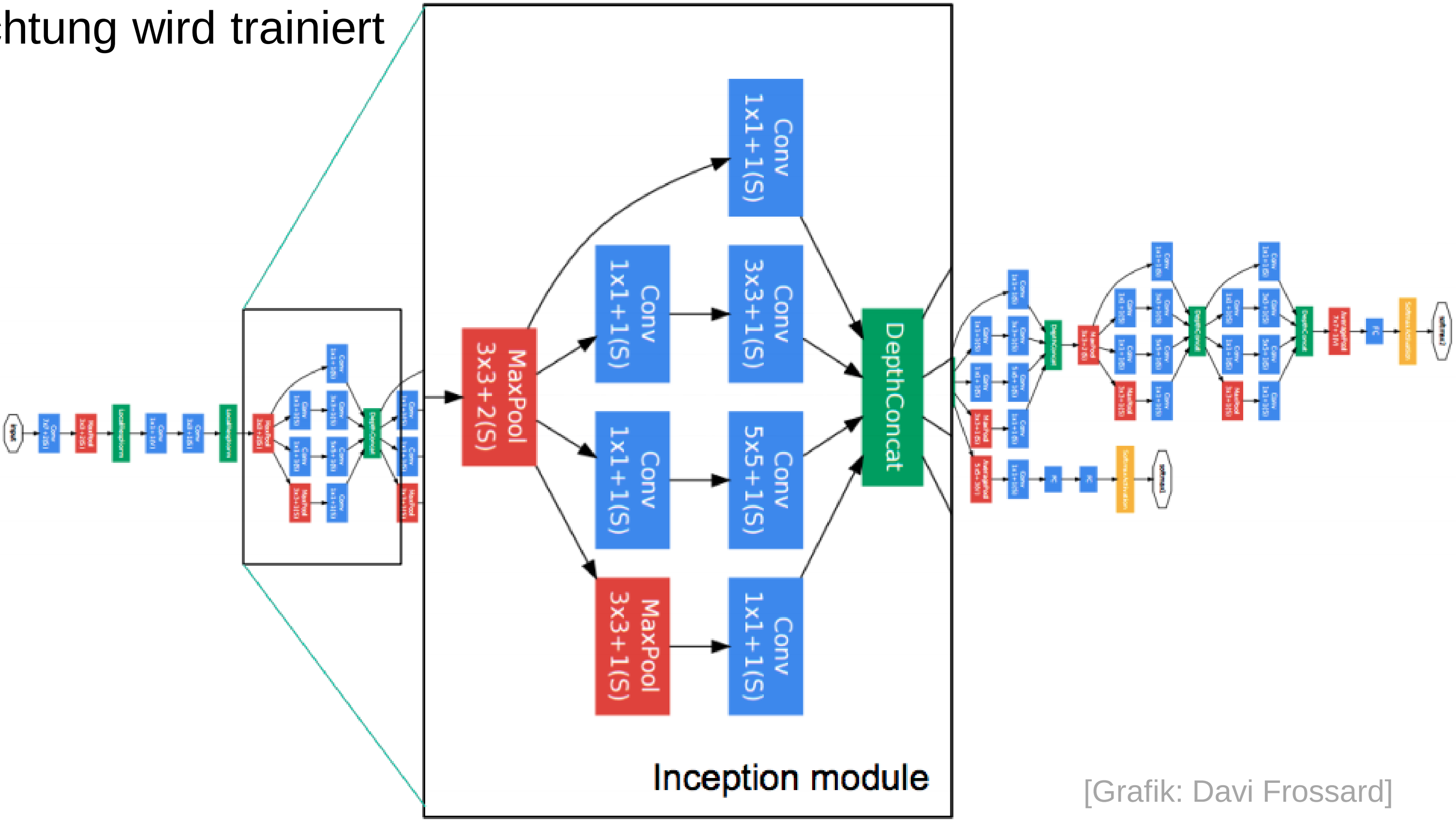
Klassifikator

- GoogLeNet (auch: Inception v1)
  - Going Deeper with Convolution [2014]
  - Christian Szegedy et.al.



Klassifikator

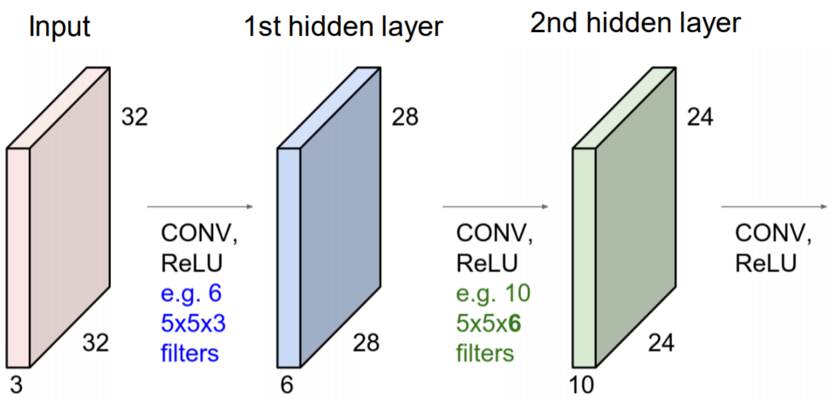
- GoogLeNet
  - Inception module
  - Ermöglicht verschiedene Faltungsoperationen
  - Gewichtung wird trainiert



[Grafik: Davi Frossard]

Klassifikator

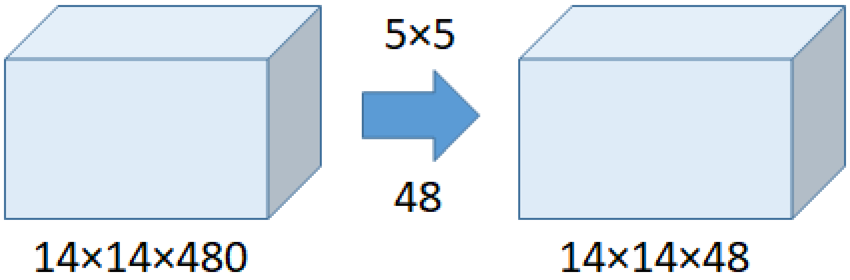
- 1x1-Kernel Faltungen?
- Was macht sowas?



[Karpathy, CS231n]

Klassifikator

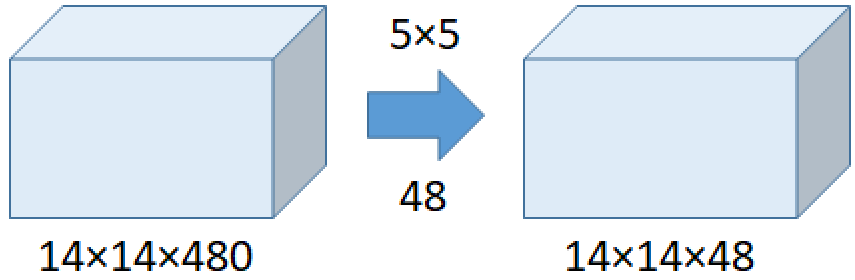
- 1x1-Kernel Faltungen?
- Was macht sowas?
- Einfaches Beispiel:



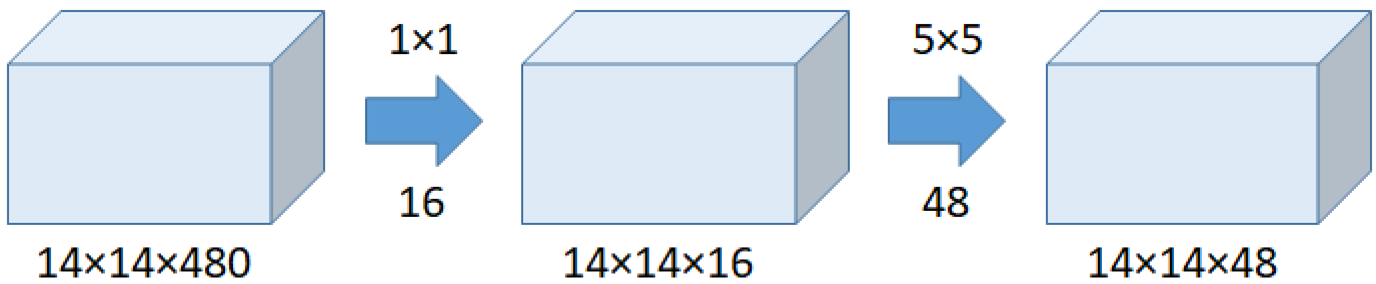
- Anzahl der Operationen 5×5 = (14×14×48)×(5×5×16) = 112.9M

[Grafik: SH Tsang]

Klassifikator



- Anzahl der Operationen = (14×14×48)×(5×5×480) = 112.9M

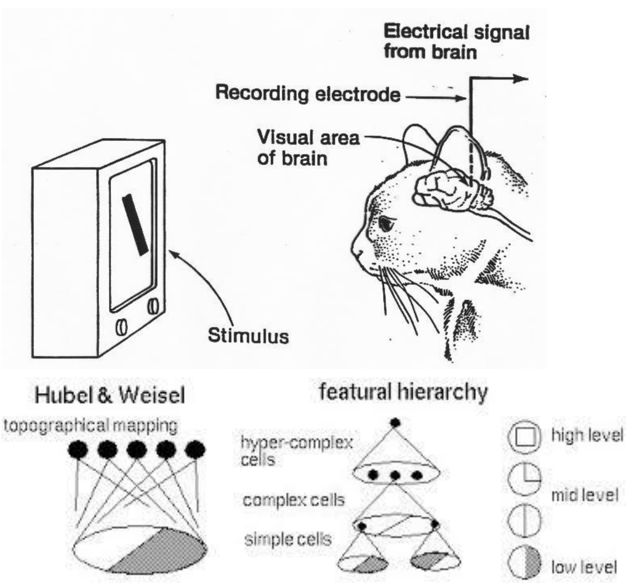


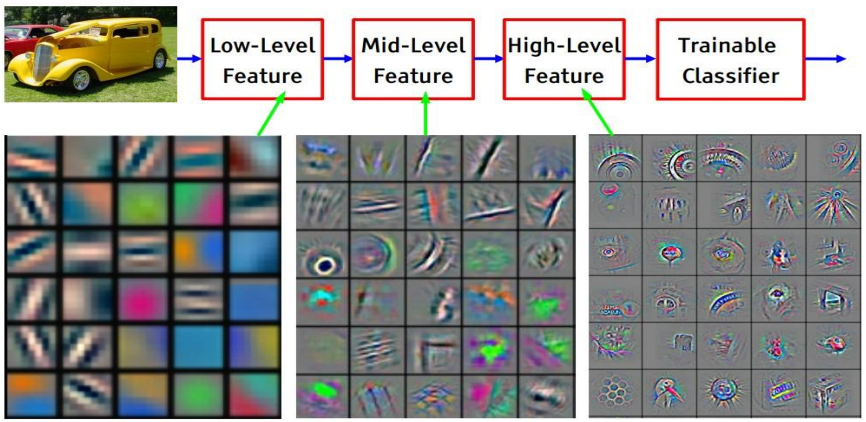
- Anzahl der Operationen für 1×1 = (14×14×16)×(1×1×480) = 1.5M
- Anzahl der Operationen für 5×5 = (14×14×48)×(5×5×16) = 3.8M
- Anzahl der Operationen Total = 1.5M + 3.8M = 5.3M

[Grafik: SH Tsang]

Klassifikator

- 1x1 Faltung:
  - Reduziert Rechenaufwand enorm
  - Dimensions Reduktion (oder?)
    - Eine zusätzliche Verknüpfung der Features findet statt!
    - Nicht-Linearitäten werden hinzugefügt

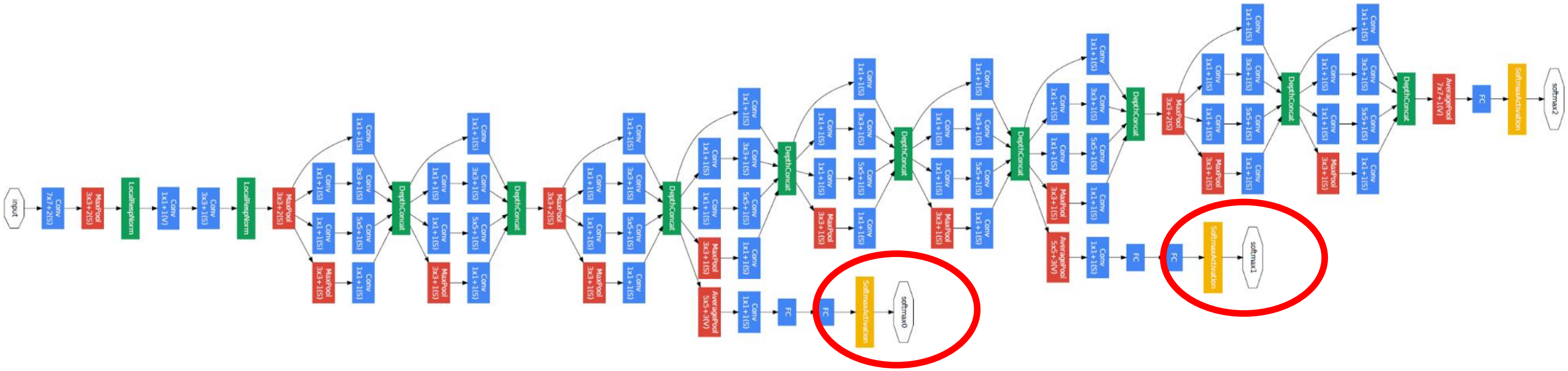




Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

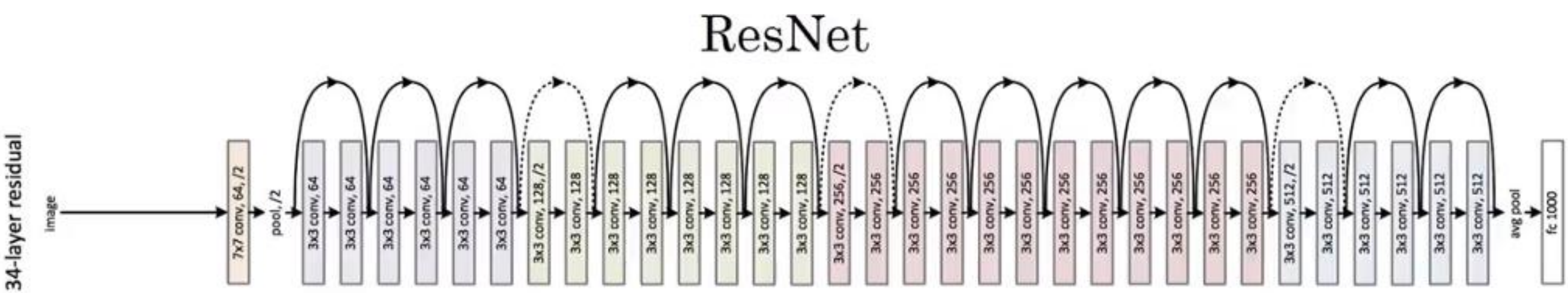
Klassifikator

- GoogLeNet
  - Auxiliary Classifiers mit Softmax
  - Nur zum Training genutzt
    - Loss wird in Zwischenschichten eingebracht
    - Hilft gegen Vanishing Gradient



Klassifikator

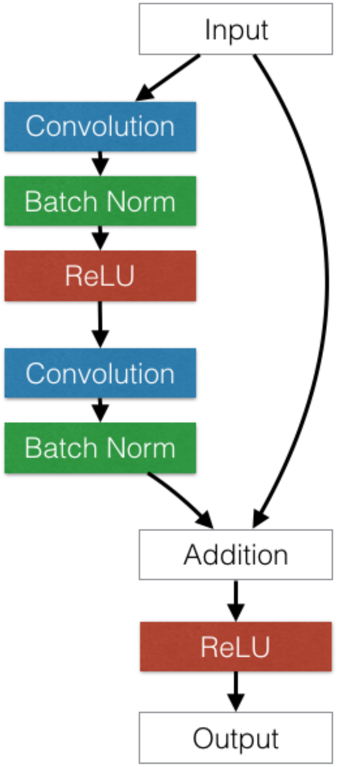
- ResNet
  - Resiudal Network
  - Deep Residual Learning for Image Recognition (Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015)
  - ResNet-16, ResNet-34, ResNet-50…



Klassifikator

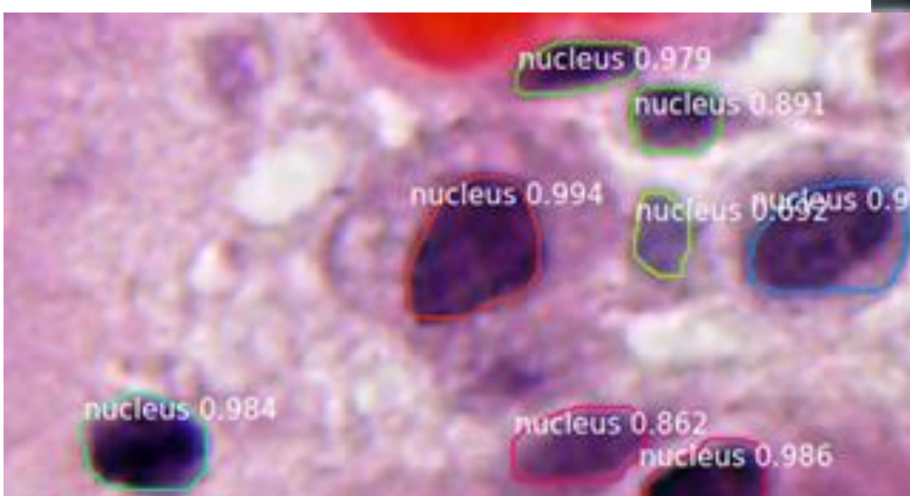
- ResNet
  - Information aus dem Input bleibt erhalten
  - Gewichtete Addition am Ende jedes Blocks

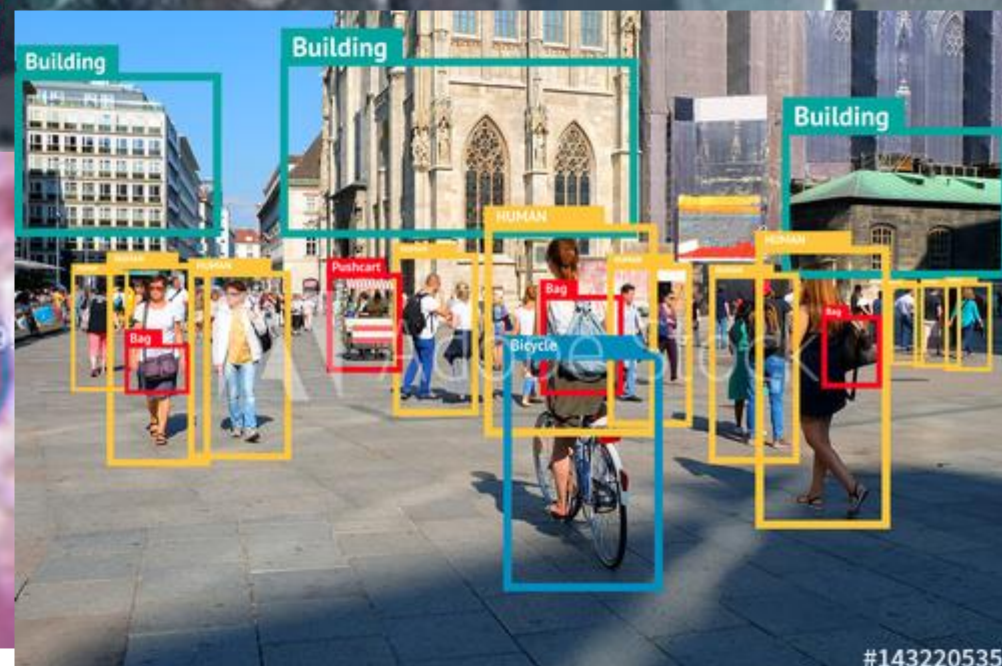
  - Verringert Vanishing Gradient Problem



Lokalisierung und Segmentierung

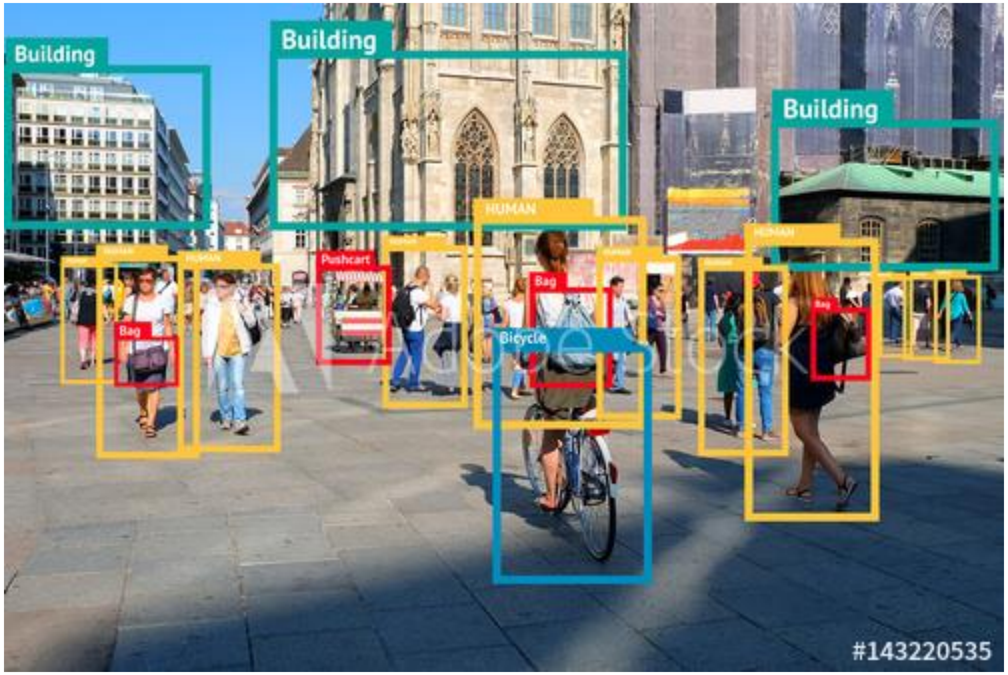
- Die R-CNN Familie
- YOLO (v1, v2…)
- U-Net





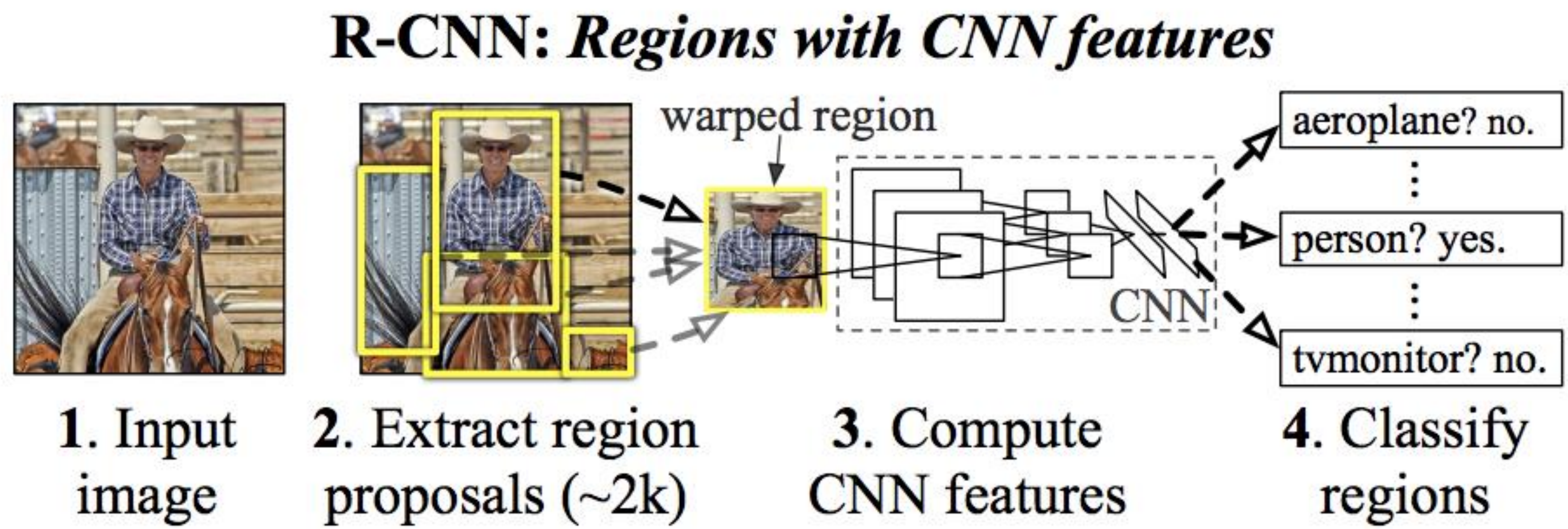
Lokalisierung und Segmentierung

- Bounding boxes
- Vier Eckpunkte als Regression (bzw. x, y, w, h)



Lokalisierung und Segmentierung

- R-CNN (Regions with CNN features, 2014, Girshick et al.)
  - Zwei Schritte:
  - Region Proposals & CNN Classify



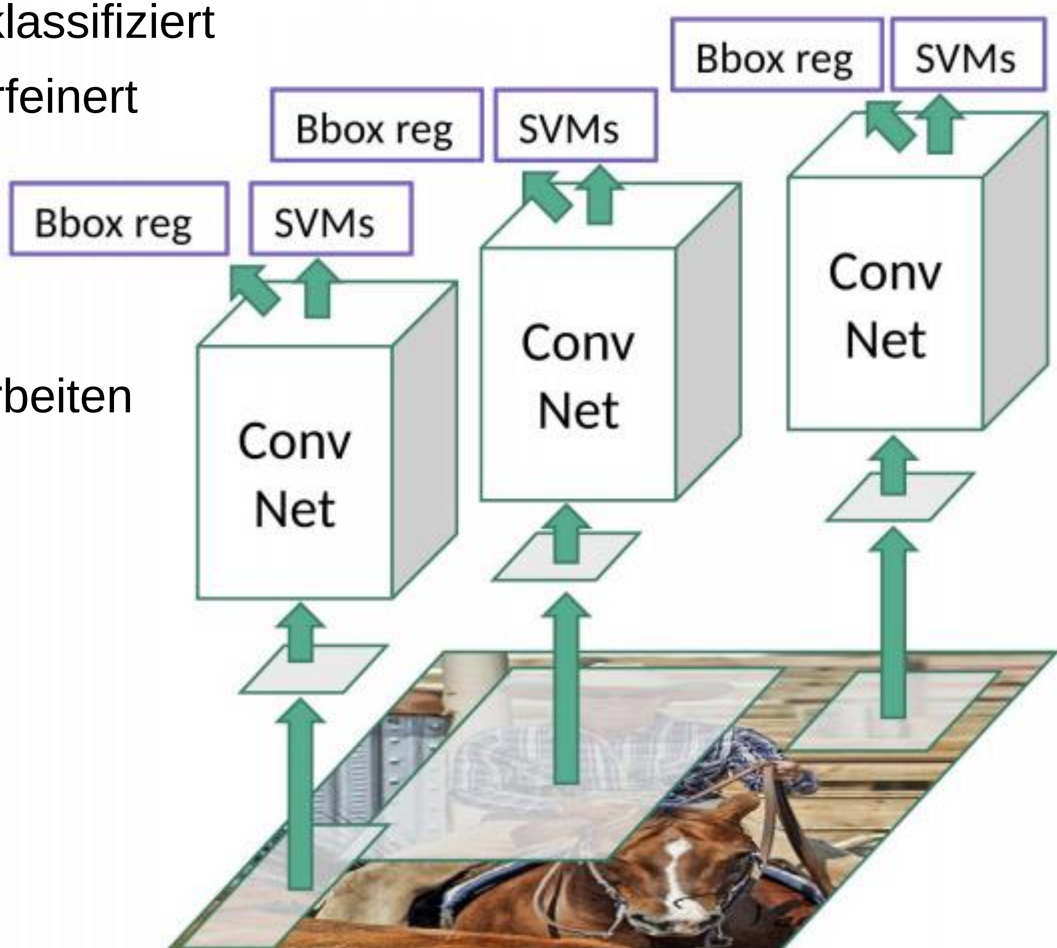
[Grafik: Rohith Gandhi]

Lokalisierung und Segmentierung

- Region Proposals:
  - Selective Search for Object Recognition [J.R.R. Uijlings et.al, 2012]
  - Statischer Algorithmus

Lokalisierung und Segmentierung

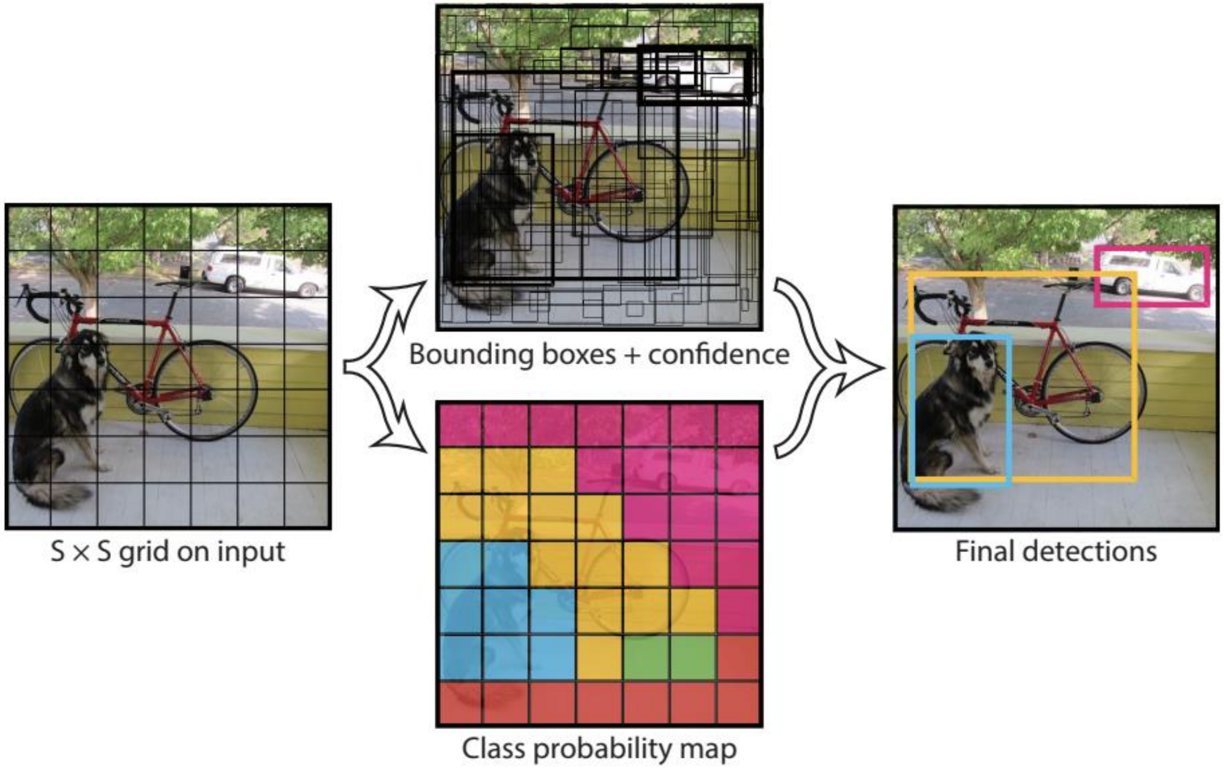
- CNN Classify und Bounding Box Regression
  - Region Proposals werden klassifiziert
  - BoundingBoxes werden verfeinert
- Sehr langsam (ca. 0.07fps)
- Mittlerweile viele Nachfolgearbeiten
  - Fast-R-CNN
  - Faster-R-CNN
  - Mask-R-CNN (ca. 5fps)



[Grafik: Rohith Gandhi]

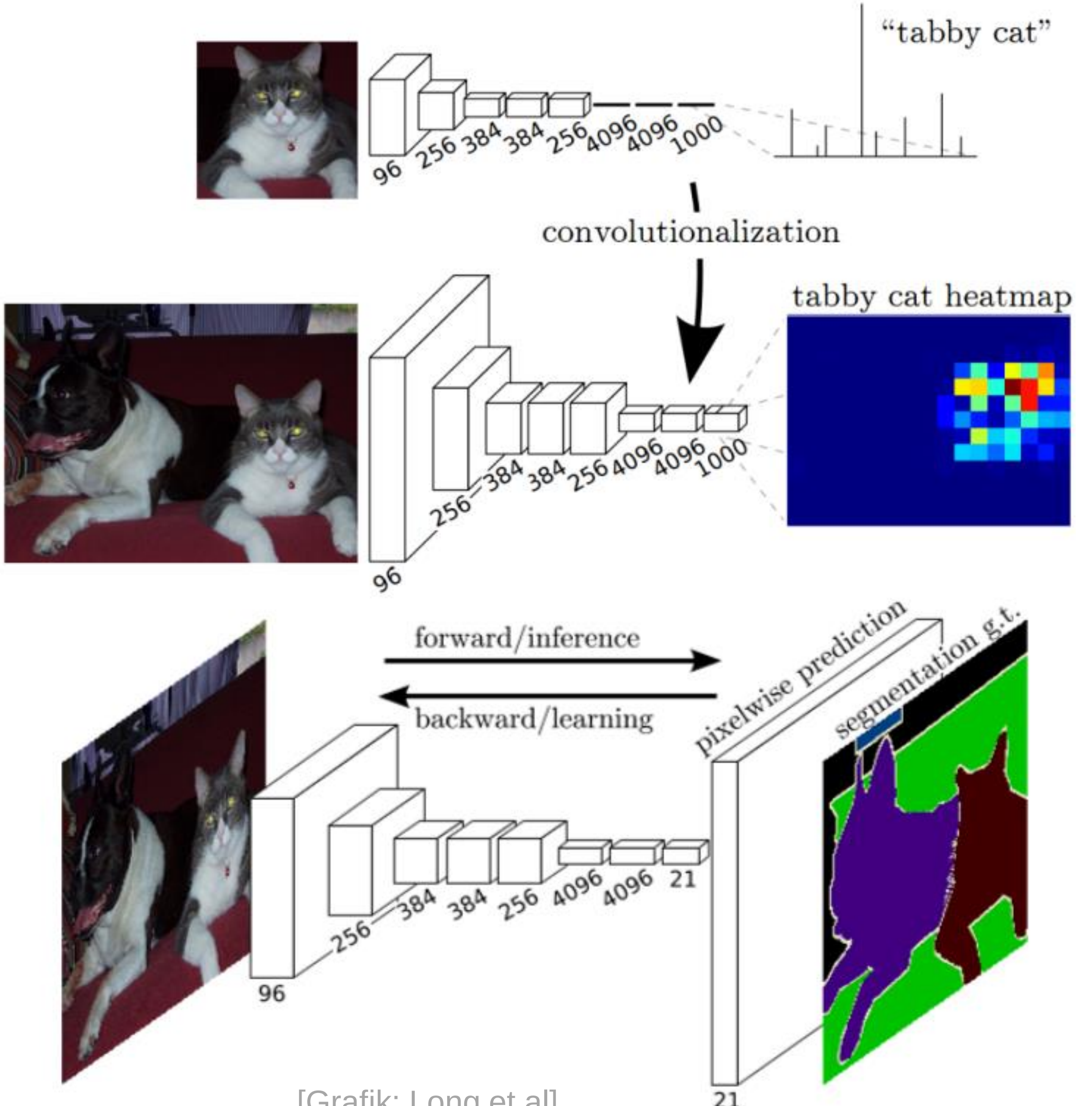
Lokalisierung und Segmentierung

- YOLO
  - You Only Look Once: Unified, Real-Time Object Detection
  - Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi [2016]
  - Ca. 25 fps



Lokalisierung und Segmentierung

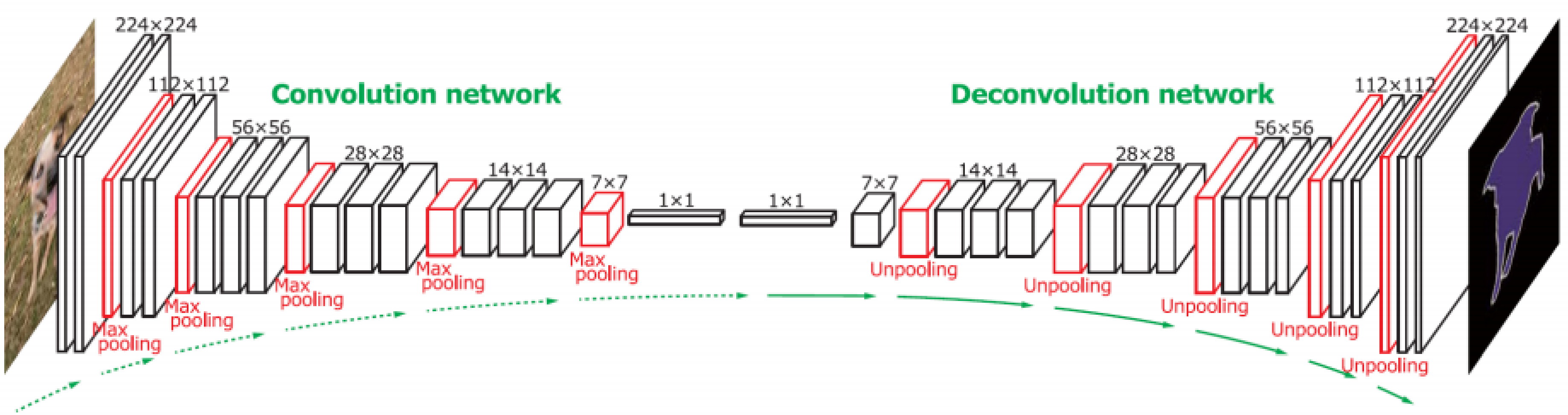
- Segmentierung
- Auch hier kommen Faltungen zum Einsatz
- Fully Convolutional
- Der Output muss wieder ein Bild sein



[Grafik: Long et al]

Lokalisierung und Segmentierung

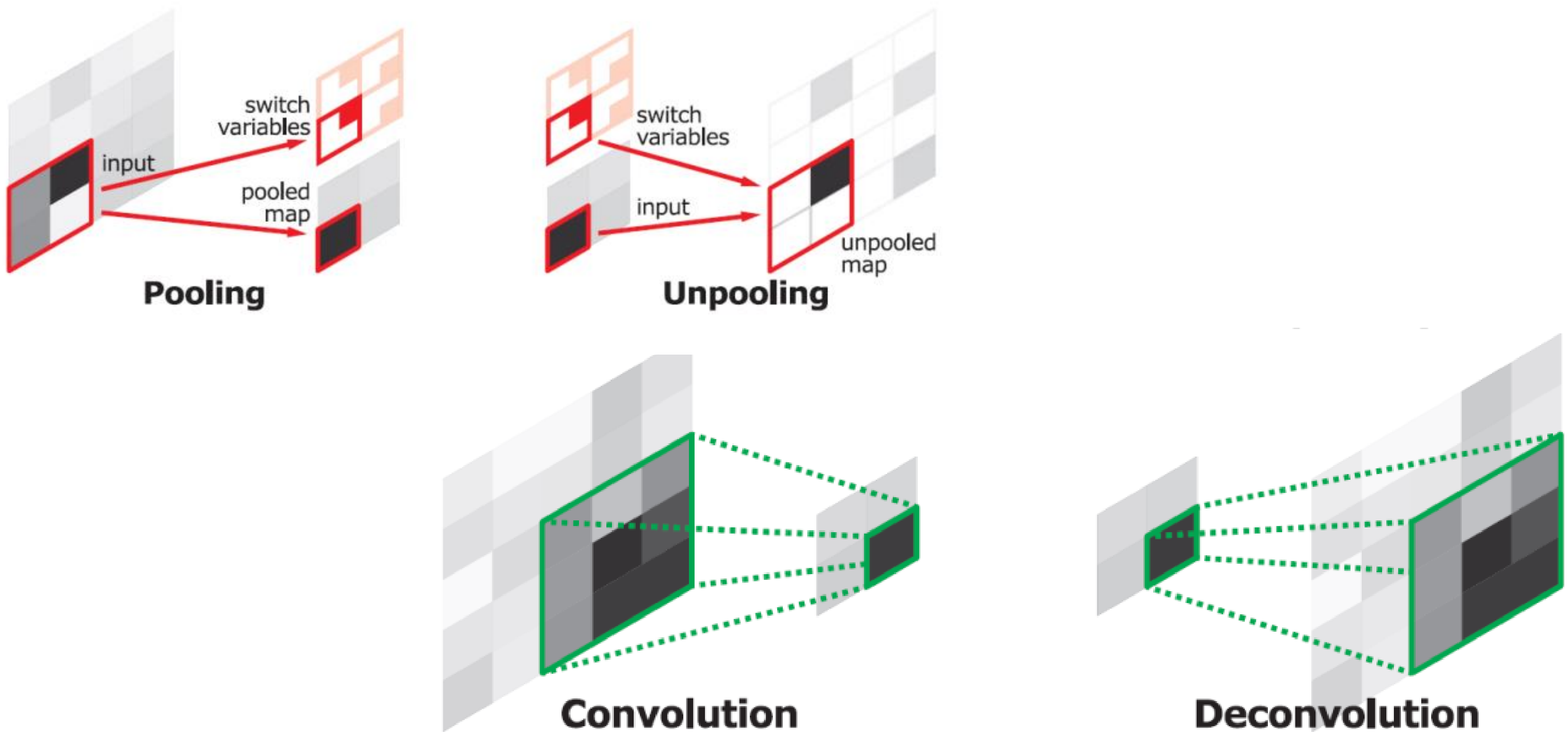
- Deconvolutional Networks



Credit: Noh et al

Lokalisierung und Segmentierung

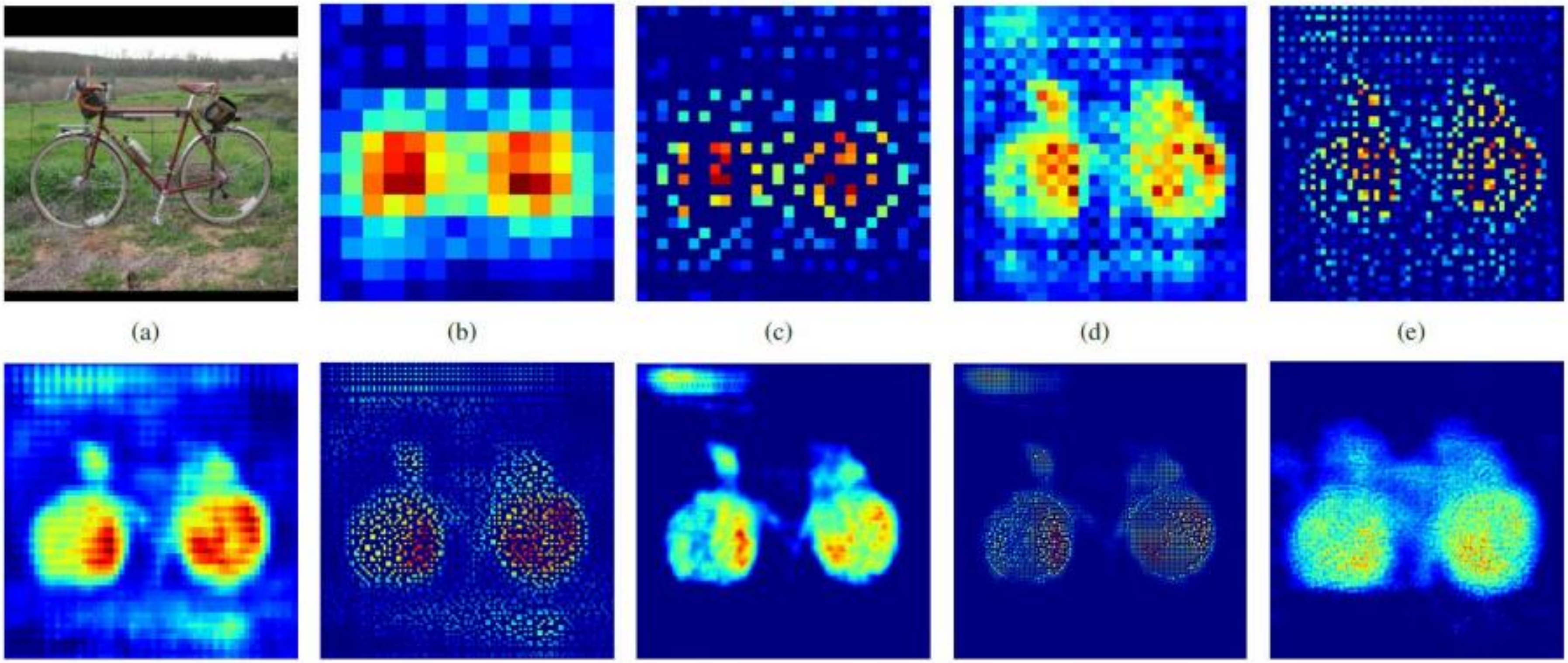
- Deconvolutional Networks
- Unpooling & Deconv



Credit: Noh et al

Lokalisierung und Segmentierung

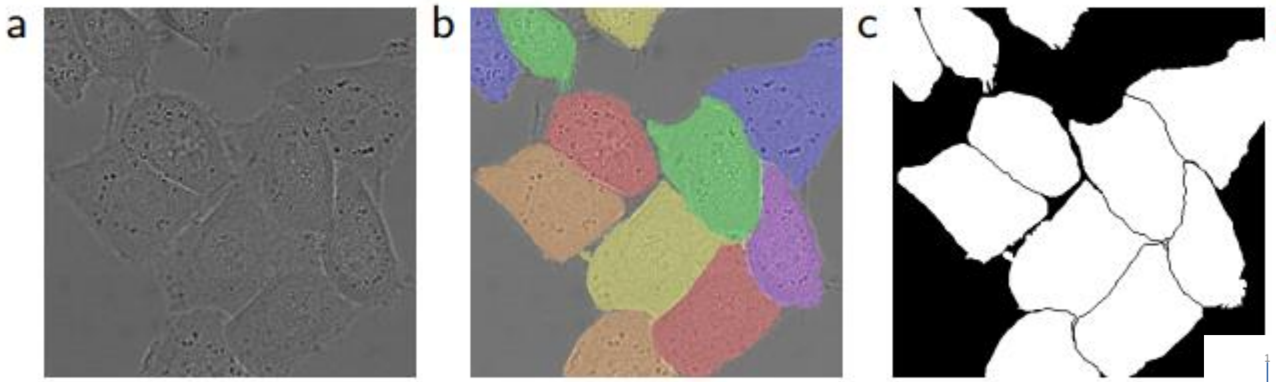
- Deconvolutional Networks
- Unpooling & Deconv

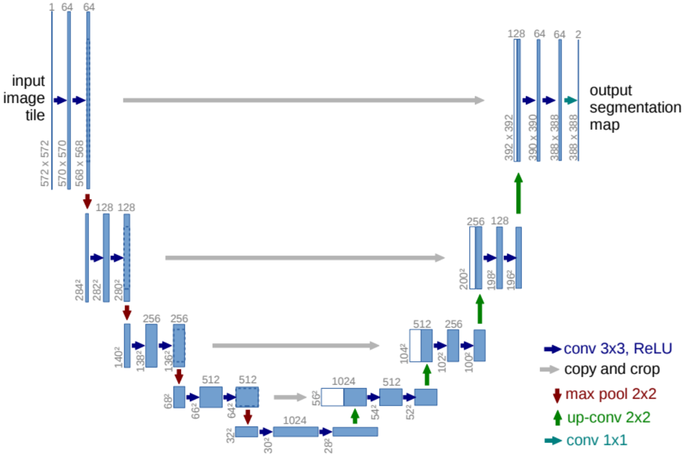


Credit: Noh et al

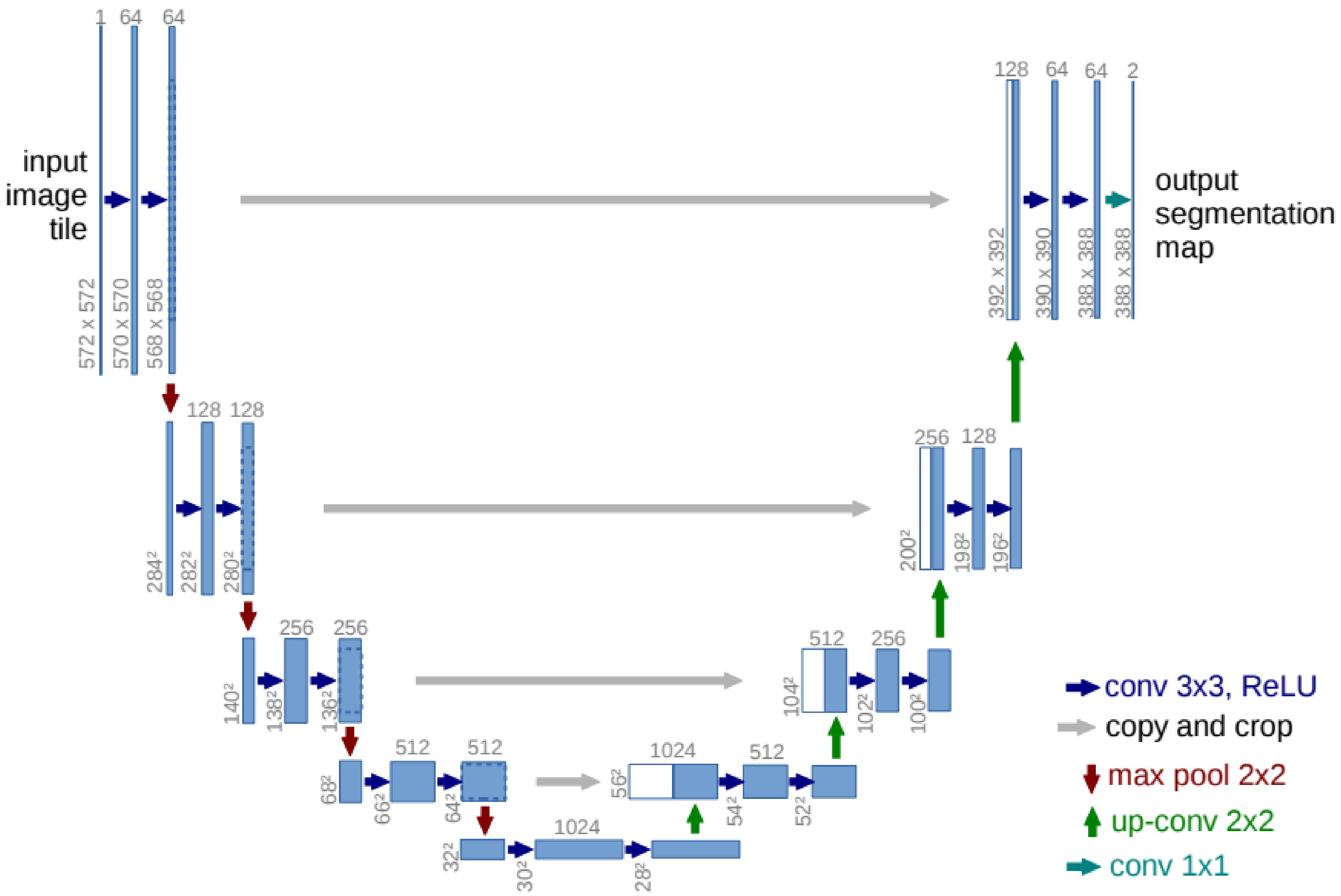
Lokalisierung und Segmentierung

- U-Net: Convolutional Networks for Biomedical Image Segmentation
  - Olaf Ronneberger, Philipp Fischer, Thomas Brox
  - Uni Freiburg [2015]





Lokalisierung und Segmentierung



Transfer Learning

- Manchmal reicht es nicht eine passende Architektur zu haben
- ConvNets brauchen viele Daten zum Training
- Oftmals sucht man Lösungen für einen Spezialfall
  - sehr kleiner Datensatz

- Lösung:
  - mit einem anderen (größeren) Datensatz vortrainieren
  - Die vorbereiteten Gewichte auf das eigene Problem anpassen

[Grafik: Karpathy]

Transfer Learning

1. Training mit ImageNet

2. Bei kleinem Dataset: fixieren aller Weights (Das CNN als feature extractor), Nachtrainieren des Klassifikators

FC und Softmax layer am Ende des Netzes austasuchen

3. Bei mittelgroßem Dataset: **“finetuning”** Die Vortrainierten Weights als Initialisierung beibehalten. Das ganze Netz oder mehrere Layer trainieren

Größeren Teil des Netzes modifizieren

[Grafik: Karpathy]

Transfer Learning

- Es existieren sehr viele vortrainierte Netzwerke
- Keras bietet diese als „Applications“:

Available models

Models for image classification with weights trained on ImageNet:

- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet
- DenseNet
- NASNet
- MobileNetV2

Transfer Learning

- Generell ist die Keras Docu sehr gut!
- FAQs:

What does "sample", "batch", "epoch" mean?

Below are some common definitions that are necessary to know and understand to correctly utilize Keras:

- **Sample:** one element of a dataset.
- *Example:* one image is a **sample** in a convolutional network
- *Example:* one audio file is a **sample** for a speech recognition model
- **Batch:** a set of *N* samples. The samples in a **batch** are processed independently, in parallel. If training, a batch results in only one update to the model.
- A **batch** generally approximates the distribution of the input data better than a single input. The larger the batch, the better the approximation; however, it is also true that the batch will take longer to process and will still result in only one update. For inference (evaluate/predict), it is recommended to pick a batch size that is as large as you can afford without going out of memory (since larger batches will usually result in faster evaluating/prediction).
- **Epoch:** an arbitrary cutoff, generally defined as "one pass over the entire dataset", used to separate training into distinct phases, which is useful for logging and periodic evaluation.
- When using `evaluation_data` or `evaluation_split` with the `fit` method of Keras models, evaluation will be run at the end of every **epoch**.
- Within Keras, there is the ability to add callbacks specifically designed to be run at the end of an **epoch**. Examples of these are learning rate changes and model checkpointing (saving).

- Paper Referenzen:

ResNet50

```
keras.applications.resnet50.ResNet50(include_top=True, weights='imagenet', input_tensor=None, input_shape=None,
```

Arguments

- include_top: whether to include the fully-connected layer at the top of the network.
- weights: one of `None` (random initialization) or `'imagenet'` (pre-training on ImageNet).
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(224, 224, 3)` (with `'channels_last'` data format) or `(3, 224, 224)` (with `'channels_first'` data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. `(200, 200, 3)` would be one valid value.
- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.
  - `None` means that the output of the model will be the 4D tensor output of the last convolutional layer.
  - `'avg'` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
  - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.

References

- Deep Residual Learning for Image Recognition

Transfer Learning

- Wenn keine Application existiert:
  - Paper lesen
  - GitHub
  - Model Zoo

Ausblick

- Wie geht es Weiter?

  - Was ist wenn meine Daten keine Bilder sind?
    - Audio und Video?

  - Wie Trainiere ich meine Netze richtig?
    - Überwachung des Trainingsvorgangs
    - Hyperparameter
    - Initialisierungen
    - Versuchsplanung