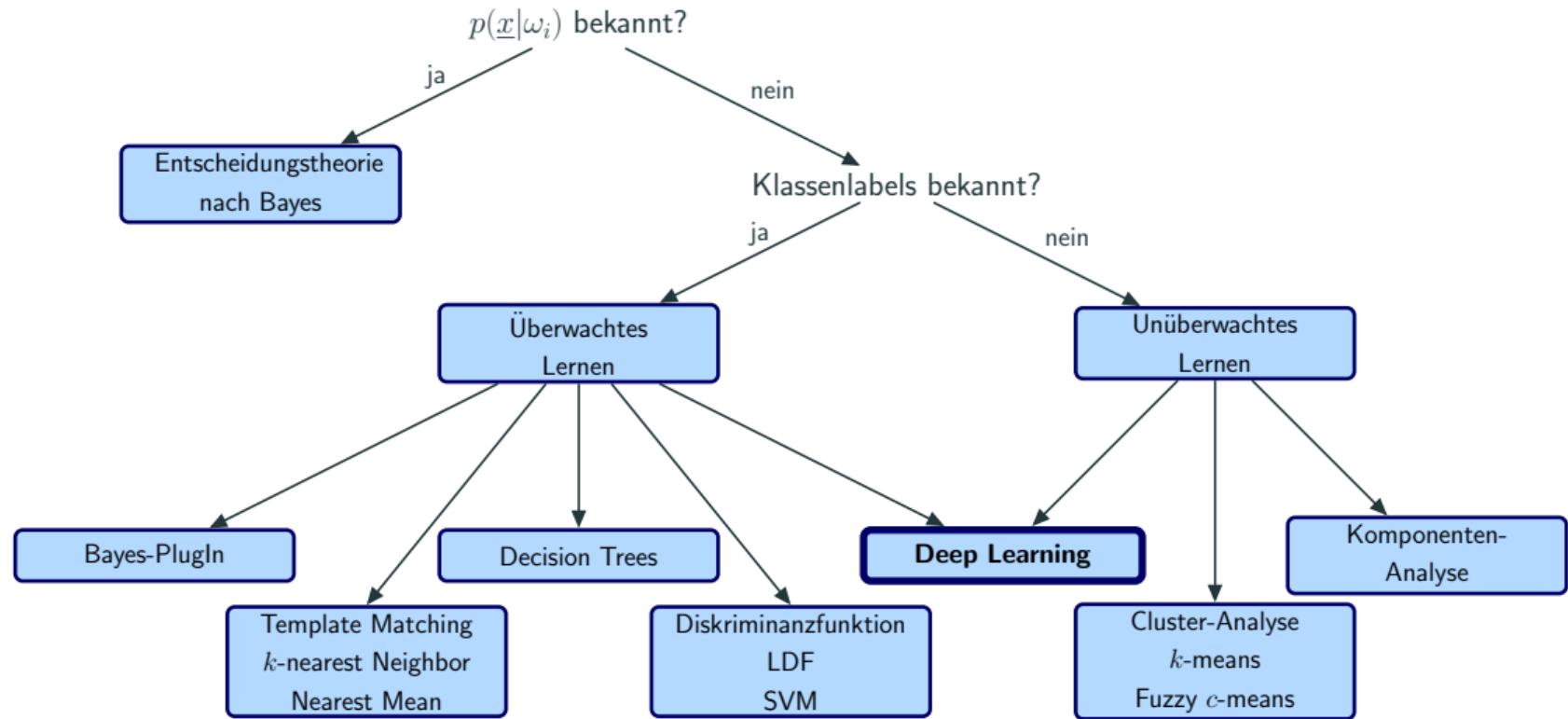


Kapitel 7 - Neuronale Netze und Deep Learning

Annika Liebgott

December 3, 2025

Übersicht



7.1 - Einleitung

Biologische und künstliche neuronale Netze

Biologische neuronale Netze

Biologisches Neuron: Nervenzelle

Das Gehirn (=biologisches neuronales Netz) ist ein Netzwerk hochgradig vernetzter biologischer Neuronen. Es besitzt eine ausgeprägte Fähigkeit zu lernen, zu abstrahieren und sich anzupassen.

Künstliche neuronale Netze

Künstliches Neuron: Schaltung oder Software

Künstliche neuronale Netze (NN) versuchen, die Funktionsprinzipien biologischer neuronaler Netze zu immitieren. Sie sind das Herzstück auf dem Weg zu künstlicher Intelligenz.

Unterschiede zu Kapitel 5 + 6

Bisher: "klassisches maschinelles Lernen"

⇒ zunächst *müssen* Merkmale aus den zu verarbeitenden Daten extrahiert und vorbereitet werden, bevor ein Klassifikator trainiert werden kann.

Unterschiede zu Kapitel 5 + 6

Bisher: "klassisches maschinelles Lernen"

⇒ zunächst *müssen* Merkmale aus den zu verarbeitenden Daten extrahiert und vorbereitet werden, bevor ein Klassifikator trainiert werden kann.

Jetzt: Neuronale Netze/Deep Learning

⇒ in den meisten Fällen lernt das neuronale Netz selbstständig, wie die Eingabedaten am besten repräsentiert werden, d.h. Merkmalsextraktion und -Reduktion sind nicht mehr zwingend nötig.

Geschichte der neuronalen Netze: der Weg zum heutigen Deep Learning

Der Anfang: bereits in den 1940er-1970er Jahren kam erstmals die Idee auf, das menschliche Gehirn künstlich zu immitieren. Simple künstliche Neuronen (Perceptron) konnten aber nur simple lineare Aufgaben lösen.

1. Ära der Neuronalen Netze: in den 1980er Jahren wurden Feedforward NN mit nichtlinearen Neuronen und der Backpropagation-Algorithmus entwickelt. Die Lösung simpler nichtlinearer Aufgaben wurde möglich.

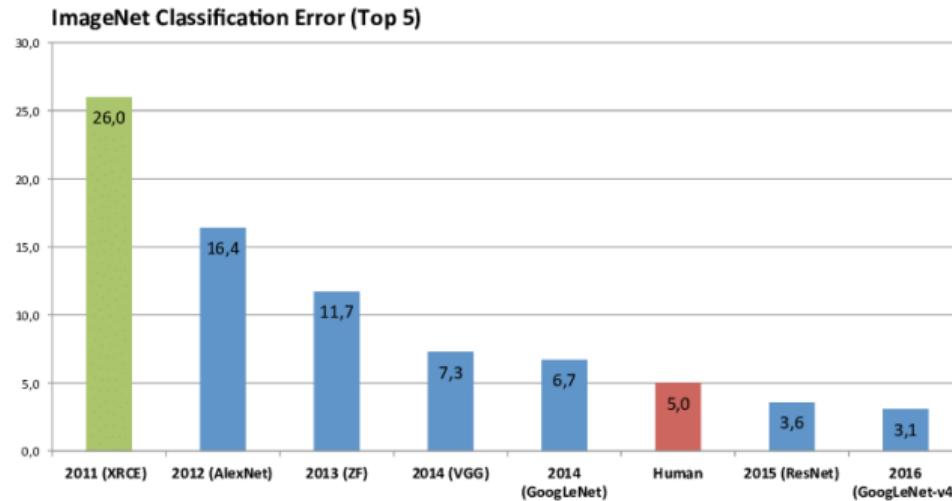
“NN-Winter”: in den 1990er - 2000er Jahren kaum Aufmerksamkeit und kein Fortschritt, da andere Methoden (z.B. SVM, RF) komplexere Aufgaben lösen konnten.

Deep-Learning-Ära: seit etwa 2010 erfolgt, befeuert durch die signifikante Steigerung verfügbarer Rechenleistung, eine rapide Entwicklung. Tiefere Netze, neue Architekturen und Regularisierungen, die Verarbeitung größerer Datenmengen und effiziente Tools ermöglichen die Lösung immer komplexerer Aufgaben.

Beispiel 7.1: ImageNet-Challenge

ImageNet large scale visual recognition challenge¹ (ILSVRC):

- 1.000 Klassen, 1.2 Millionen Trainings- und 50.000 Testbilder, 1 Klasse pro Bild
- Performance-Metrik: Top-5-ER (wahre Klasse nicht in den 5 wahrscheinlichsten)²



¹Russakovsky, Deng et al.: "ImageNet Large Scale Visual Recognition Challenge", IJCV, 2015

²Bildquelle: von Zitzewitz: "Survey of neural networks in autonomous driving", 2017

Beispiel 7.2: AlphaGo (1) ³

Eine Anwendung von Deep Learning, die es in die weltweiten Schlagzeilen geschafft hat.

Go: Ältestes chinesisches Brettspiel (über 2.500 Jahre alt) mit einfachen Regeln, aber komplexen Strategien.

Zustandsraum-Komplexität des Spiels: $\approx 10^{172}$ (zum Vergleich: es existieren im gesamten Kosmos etwa $10^{84} - 10^{89}$ Atome)



AlphaGo: Google DeepMind trainiert Neuronales Netzwerk auf Basis vieler menschlicher Go-Partien, um Wahrscheinlichkeits-Karten für die besten nächsten Züge, ausgehend von der aktuellen Spielsituation, zu berechnen.

2015: AlphaGo Fan schlägt professionellen Go-Spieler Fan Hui

2016: AlphaGo Lee schlägt Lee Sedol, einen Go-Spieler des 9. Dan, mit 4:1

2017: AlphaGo Master schlägt Ke Jie, den amtierenden weltbesten Go-Spieler, mit 3:0

³Bildquelle:<http://www.yukai-japan.de>

Beispiel 7.2: AlphaGo (2)

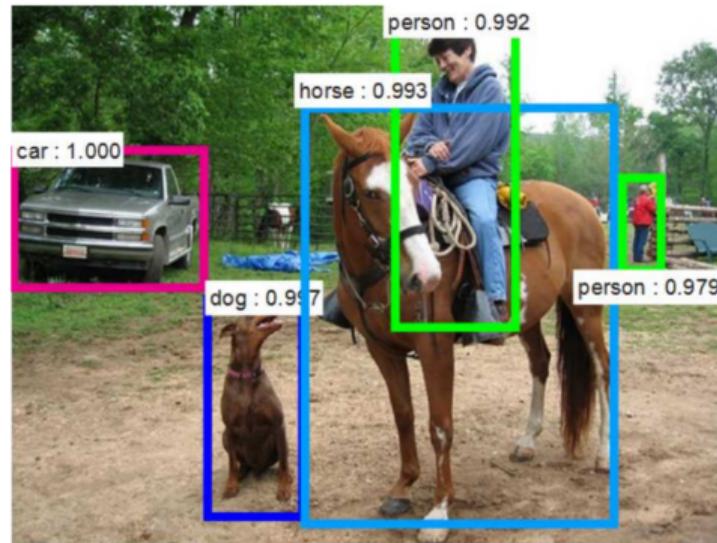
Ebenfalls 2017: AlphaGo Zero

- Erlernen des Spiels von Grund auf *ohne* menschliche Partien zu kennen
 - Training erfolgt über Reinforcement Learning (vgl. Kapitel 3.6.3) anhand von Spielen von AlphaGo Zero vs. AlphaGo
 - Nach 3 Tagen Training: Level von AlphaGo Lee erreicht (100:0)
 - Nach 21 Tagen Training: Level von AlphaGo Master erreicht
 - Level von AlphaGo Zero unerreichbar durch menschliche Spieler
- ⇒ Demonstration der Möglichkeiten künstlicher Intelligenz: weniger als 1 Monat Training schlägt 2.500 Jahre menschlicher Erfahrungen

Anwendungsgebiet: Objekterkennung

Aufgabe: Detektion und Lokalisierung von (mehreren) Objekten

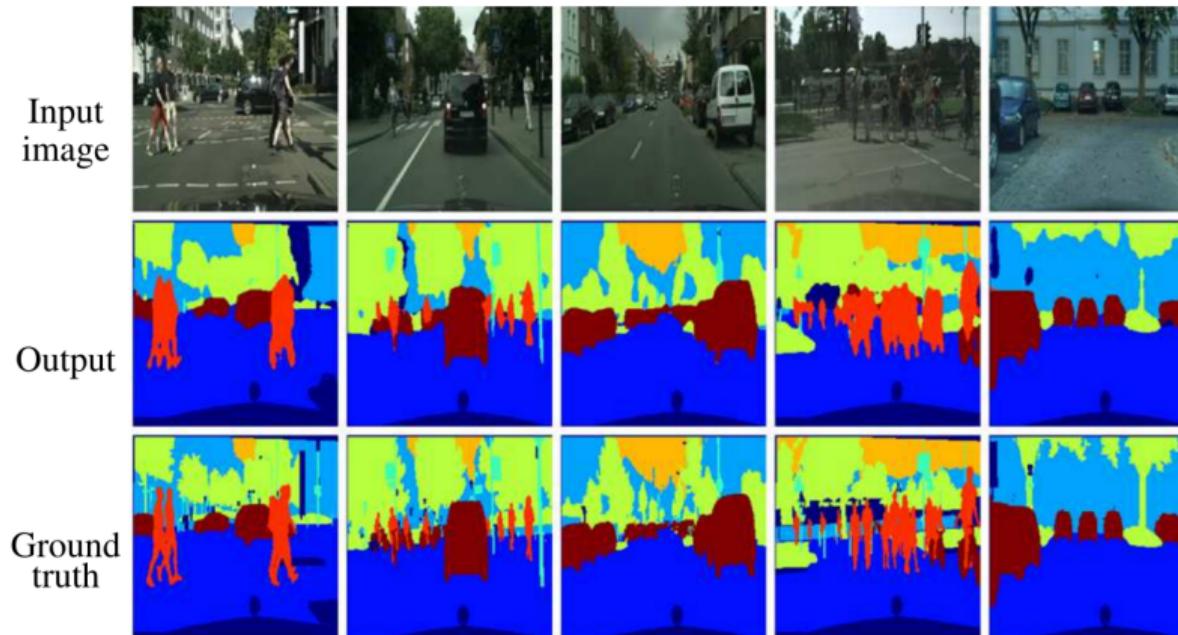
- Bounding-Boxes für gefundene Objekte (siehe Beispielbild⁴)
- meistens kombiniert mit Klassifikation der gefundenen Objekte



⁴Ren et al.: “Faster R-CNN: Towards real-time object detection with region proposal networks”, 2016

Anwendungsgebiet: Semantische Bildsegmentierung (Bsp: Auton. Fahren⁵)

Aufgabe: Klassifikation des Bildinhaltes auf Pixel-Ebene \Rightarrow pro Pixel eine Klasse



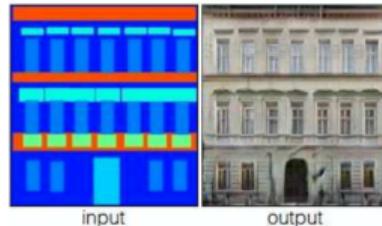
⁵Wang et al.: "On semantic image segmentation using deep convolutional neural network with shortcuts and easy class extension", IPTA, 2016

Anwendungsgebiet: Domain-to-Domain-Translation

Aufgabe: Übertragung eines gegebenen Datensatzes in eine andere Darstellungsform (z.b. Text-to-Image-Translation oder Image-to-Image-Translation)

Beispiel: Image-to-Image-Translation mit cGAN⁶

facade labels → house photo



edges → bag



black/white image → color image



day photo → night photo



⁶Isola: "Image-to-image translation with conditional adversarial networks", 2017

Weitere Anwendungsgebiete

Deep Learning kann für alle möglichen Aufgaben angewendet werden:

- Natural Language Processing
- Automatische Spracherkennung
- Empfehlungssysteme
- Medizinische Bildanalyse
- Bioinformatik
- Bildwiederherstellung/Datenwiederherstellung allgemein
- Betrugserkennung im Finanzsektor
- Militärische Anwendungen
- Datengenerierung
- ...

Bekannte Datensets: MNIST⁷

- Datenset zur Erkennung handgeschriebener Zahlen 0-9 (10 Klassen)
- 70.000 Graustufenbilder, vorsegmentierte handschriftliche Zahlen
- Bildgröße: 28×28 Pixel



⁷LeCun et al.: "The MNIST database of handwritten digits", online

Bekannte Datensets: Fashion-MNIST⁸

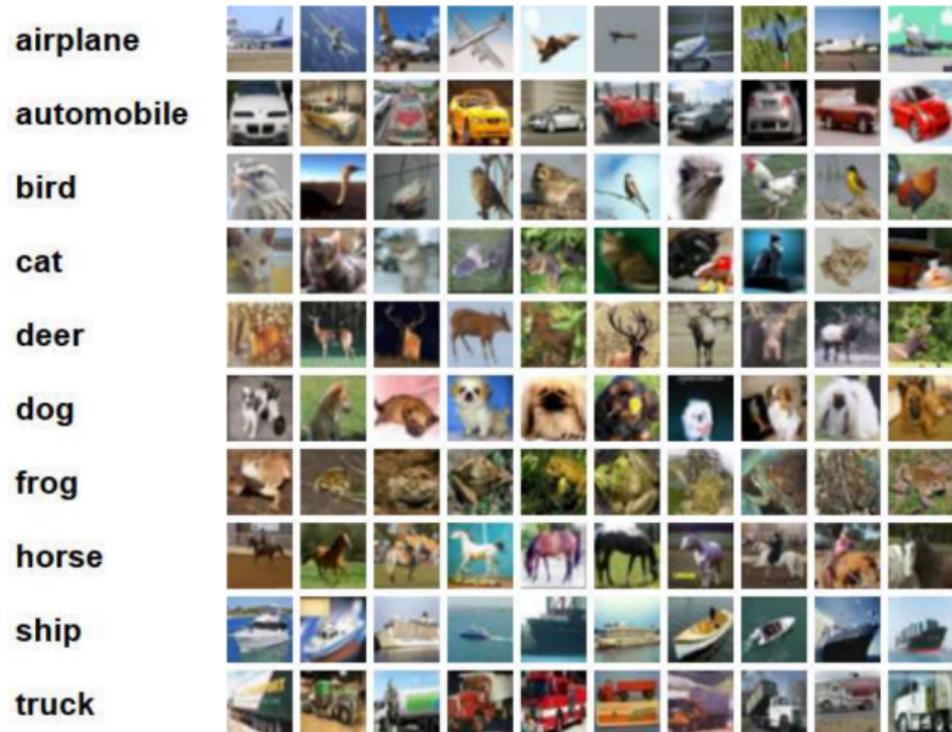
- Datenset zur Klassifizierung von Kleidungsstücken (10 Klassen)
- 70.000 Graustufenbilder, vorsegmentierte Artikel aus dem Zalando-Sortiment
- Bildgröße: 28×28 Pixel



⁸<https://research.zalando.com/welcome/mission/research-projects/fashion-mnist>

Bekannte Datensets: CIFAR-10/CIFAR-100⁹

- Datenset zur Objekterkennung
- 60.000 Farbbilder, unterschiedlicher Bildkontext
- CIFAR-10: 10 Klassen
- CIFAR-100: 100 Klassen
- Bildgröße: 32×32 Pixel



⁹www.cs.toronto.edu/~kriz/cifar.html

Bekannte Datensets: ImageNet^{10,11}

- Sehr großes Datenset zur Objekterkennung
- über 14 Millionen Farbbilder, über 20.000 Klassen
- Bildgröße: variabel



¹⁰www.image-net.org

¹¹Bildquelle: karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

Bekannte Datensets: Cityscapes¹²

- Großes Datenset mit Staßenaufnahmen für semantische Bildsegmentierung
- 20.000/5.000 gelabelte Bilder mit groben/feinen Labels
- 50 Städte, 30 Objektklassen aus 8 Objektgruppen



¹²www.cityscapes-dataset.com

7.2 - Künstliche Neuronale Netze

Wozu Aktivierungsfunktionen?

Anforderungen an Aktivierungsfunktionen $\phi()$:

- im Allgemeinen: Nichtlinearität \Rightarrow Nachbildung menschlicher Neuronen
- Differenzierbarkeit \Rightarrow notwendig für Training
- einfache Berechenbarkeit \Rightarrow Senkung der Komplexität

Warum ist Nichtlinearität wichtig?

Bei linearer Aktivierung $\phi(a) = a$ ist der Output \underline{x}_l einer Schicht l immer eine affine Funktion ihres Inputs \underline{x}_{l-1}

\Rightarrow Gesamt-Output \underline{x}_L des NN ist affine Funktion des Netzwerk-Inputs \underline{x}_0

\Rightarrow die Hidden Layer sind somit nutzlos

\Rightarrow Netzwerk kann keine nichtlinearen Zusammenhänge lernen

Aktivierungsfunktionen¹³ (1)

Es gibt viele verschiedene Aktivierungsfunktionen. Einige beliebte:

1) Lineare Aktivierung (Identität)

Nicht als Aktivierungsfunktion in Hidden Layern möglich

Wird in der Output-Layer von Regressions-Netzen genutzt

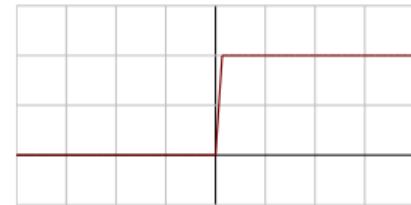


2) Sprungfunktion

Neuron feuert für $a > 0$, sonst nicht

Wenn Neuron feuert, wird gleicher Wert wie Input an
nächste Schicht gegeben

Funktion undefiniert für $a = 0$



¹³Bildquelle: Wikipedia

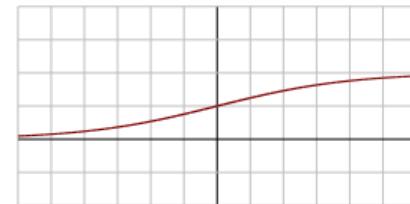
Aktivierungsfunktionen¹⁴ (2)

Nachteil an 1) und 2): keine komplexen Beziehungen zwischen Input/Output lernbar.

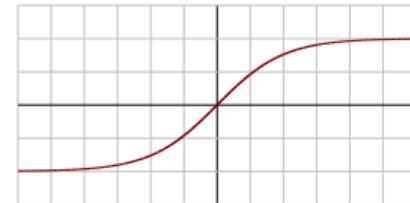
Lösung: nichtlineare Aktivierungsfunktionen

3) Sigmoid

Output zwischen 0 und 1 normiert.



4) Tangens Hyperbolicus



Vorteil: Weicher Gradientenverlauf (keine Sprünge im Output)

¹⁴Bildquelle: Wikipedia

Aktivierungsfunktionen¹⁵ (3)

5) Rectified Linear Unit (ReLU)

$\phi(a) = \max(a, 0)$ ⇒ entspricht Diode

Aktuell beliebteste Aktivierungsfunktion im Deep Learning

Problem: “Dying ReLU” für Inputwerte nahe 0



6) Parametric Rectified Linear Unit (PReLU)

Leichte Steigung für negative Werte, verhindert “Dying ReLU”



7) Exponential Linear Unit (ELU)

Exponentielle Variante der ReLU-Funktionen



¹⁵Bildquelle: Wikipedia

8) Softmax-Funktion

Wird bei Klassifikationsnetzwerken in der Output-Layer eingesetzt, um Klassenwahrscheinlichkeiten zu berechnen.

Für ein Klassifikationsproblem mit c Klassen besitzt die Output-Layer c Neuronen. Es wird die Klasse ω_i gesucht, für die der maximale Wert für $\phi_i(\underline{a})$ erreicht wird.

$$\phi_i(\underline{a}) = \frac{e^{a_i}}{\sum_{j=1}^c e^{a_j}}, \quad 1 \leq i \leq c \quad \sum_{i=1}^c \phi_i(\underline{a}) = 1$$

Spezialfall $c = 2$: Im Fall von nur zwei Klassen reicht eine binäre Entscheidung anhand eines einzelnen Output-Neurons. Statt Softmax wird die Sigmoid-Funktion verwendet.

Aktivierungsfunktionen: Zusammenfassung

Frage: Wann welche Aktivierungsfunktion verwenden?

Zwei Punkte, die zur Beantwortung geklärt werden müssen:

1. Verwendungsort: Hidden Layer oder Output-Layer?
2. Aufgabe: Klassifikation oder Regression?

	Klassifikation	Regression
Hidden Layers	<i>nichtlineare</i> Aktivierungsfunktion z.B. ReLU, Sigmoid, Tanh,...	wie für Klassifikation
Output-Layer	Softmax oder Sigmoid	Lineare Aktivierung/Identität

Aktivierungsfunktionen in TF/Keras: siehe <https://keras.io/api/layers/activations/>
Beispiel ReLU: `tf.keras.activations.relu()`

7.3 - Training eines Neuronalen Netzes

Loss-Funktionen für Regression und Klassifikation

Regression:

Mean Squared Error (MSE), Weighted MSE, Mean Absolute Error (MAE)

Klassifikation:

$c = 2$: Binary Cross-Entropy, Hinge oder Squared Hinge

$c > 2$: Categorical Cross-Entropy, Sparse Categorical Cross-Entropy,
Kullback-Leibler-Divergence

Loss-Funktionen in TF/Keras: siehe <https://keras.io/api/losses/>

Beispiel Sparse Categorical Cross-Entropy:

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()
```

Weitere Loss-Funktionen

Für manche Anwendungen können über die in Keras implementierten Funktionen hinaus gehende Loss-Funktionen sinnvoll sein, um die Performance zu steigern.

Beispiel Bildsegmentierung:

Segmentierung entspricht einer Klassifikation auf Pixel-Ebene. Aber: für Bildinhalt spielt der Kontext eine größere Rolle, als die Klassifikation des einzelnen Pixels.
⇒ statt Loss auf Basis einzelner Pixel zu berechnen, besser Metrik zur Bestimmung der *globalen Bildähnlichkeit* verwenden

Beliebte Loss-Funktionen für Bildsegmentierung basieren auf:

Dice-Koeffizient, Intersection-Over-Union (IoU), Focal Loss, Tversky-Loss, Lovasz-Loss, Hausdorff-Distanz, Jaccard-Loss

Welche Aufgabe hat ein Optimizer beim Training eines NN?

Er bestimmt, wie die Gewichte während des Trainings angepasst werden, um die Loss-Funktion zu minimieren.

Wichtiger Parameter für Optimizer: Lernrate, d.h. wie stark die Gewichte angepasst werden.

Beliebte Optimizer:

- Stochastic Gradient Descent (SGD) ⇒ vgl. Kapitel 2.3
- Root Mean Square Propagation (RMSProp)
- Adaptive Gradient Algorithm (Adagrad)
- Adadelta (Variante von Adagrad)
- Adam (beliebtester Optimizer mit adaptiver Lernrata)

7.4 - Tiefe Neuronale Netze (DNN)

7.4 - Tiefe Neuronale Netze (DNN)

7.4.1 - Aufbau und Training von DNNs

Warum ist “tief” besser?

Signale besitzen oft eine natürliche hierarchische Darstellung.

z.B.:

- Spracherkennung: Sample → Laute → Worte → Sätze
- Bildverarbeitung: Pixel → Kanten → Formen → Objekte

Veranschaulichung aus einem anderen Bereich: n -Bit Analog-Digital-Wandler

- Flacher A/D-Wandler: Flash A/D-Wandler, 1 Level für alle n Bits
⇒ $2^n - 1$ Komparatoren benötigt
- Tiefer A/D-Wandler: Serieller A/D-Wandler, 1 Bit pro Level wird bestimmt
⇒ n Komparatoren benötigt

Flaches neuronales Netz: die komplexeste Abstraktionsebene wird direkt gelernt.

Deep Learning: jede Schicht lernt eine eigene Abstraktionsebene. Einfache Merkmale aus der ersten Abstraktionsebene werden zu immer komplexeren in der letzten Schicht kombiniert.

Wann ist ein NN tief?

⇒ Deep Neural Networks (DNN): $L \geq 3$, d.h. mindestens 2 Hidden Layer.

Universelles Approximierungstheorem

Behauptung:

In der Theorie kann ein Feedforward NN mit linearer Output-Layer und mindestens einer Hidden Layer mit nichtlinearer Aktivierungsfunktion jede kontinuierliche Funktion mit beliebig hoher Genauigkeit annähern.

Anmerkungen:

- Beliebige Genauigkeit: mit steigender Anzahl versteckter Neuronen
- Funktioniert für viele nichtlineare Aktivierungsfunktionen mit Ausnahme polynomieller Funktionen
- Tiefere Neuronale Netze funktionieren in der Regel besser als flachere (Annahme: ausreichend Trainingsdaten/kein Hang zu Overfitting)

Input Scaling und Batch Normalization

Input Scaling

Um ein Modell mit guter Generalisierbarkeit zu trainieren, wird empfohlen, die Input-Daten zu skalieren.

⇒ Meistens: zero-mean, unit-variance

Batch Normalization

Input Scaling wird auf die kompletten Trainingsdaten \mathcal{D}_{train} angewendet.

Problem: eventuell sind Batches nicht normalisiert

⇒ Batch Normalization, d.h. Standardisierung jedes einzelnen Batches

Parameter-Initialisierung

Als Ausgangspunkt für die Optimierung müssen die Gewichte initialisiert werden. Dazu gibt es verschiedene Möglichkeiten:

- Nullwert-Initialisierung
- Konstante Initialisierung
- Zufällige Initialisierung
- He-Initialisierung
- Benutzerdefinierte Initialisierung
- Gewichte eines früheren Trainings ⇒ Transfer-Learning (Kapitel 7.9)

Ein Hauptvorteil von Deep Learning ist, dass Merkmale aus Eingabedaten selbstständig extrahiert werden können.

Aber: Neuronale Netze können auch merkmalsbasiert trainiert werden.

Input: Trainingssamples (\underline{x}_n, y_n) , in der Regel 1 Input-Neuron pro Feature

Hyper-Parameter: Anzahl hidden Layers/hidden Neurons, Schrittweite für Gradient Descent, Initialisierung der Gewichte für Backpropagation-Algorithmus

Training: über Backpropagation-Algorithmus

Klassifikation eines neuen Merkmalsvektors \underline{x} :

- Berechne c Diskriminanzfunktionen $f_j(\underline{x})$
- Klasse ω_j mit dem höchsten Wert für $f_j(\underline{x})$ gewinnt

Merkmalsbasiertes Machine Learning vs. Deep Learning

Merkmalsbasiertes ML

- + Interpretierbarkeit
- + Teilweise sehr schnell (ohne GPU)
- + Funktionieren unter den passenden
 auch mit kleinen Datensets

- Passende Merkmale finden: zeitaufwändig
- Nicht zwangsläufig robust
- Oft limitierte Fähigkeit zum Lösen
 komplexer Probleme

Deep Learning

Merkmalsbasiertes Machine Learning vs. Deep Learning

Merkmalsbasiertes ML

- + Interpretierbarkeit
- + Teilweise sehr schnell (ohne GPU)
- + Funktionieren unter den passenden auch mit kleinen Datensets
- Passende Merkmale finden: zeitaufwändig
- Nicht zwangsläufig robust
- Oft limitierte Fähigkeit zum Lösen komplexer Probleme

Deep Learning

- + Bessere Modell-Fähigkeiten
- + Keine explizite Merkmalsextraktion
- + Keine Merkmalsselektion/-reduktion
- + Höchst innovatives Forschungsgebiet
- Schlechter interpretierbar
- Allgemein: mehr Trainingsdaten nötig
- Höhere Rechenkapazitäten nötig

Merkmalsbasiertes Machine Learning vs. Deep Learning

Merkmalsbasiertes ML

- + Interpretierbarkeit
- + Teilweise sehr schnell (ohne GPU)
- + Funktionieren unter den passenden auch mit kleinen Datensets

- Passende Merkmale finden: zeitaufwändig
- Nicht zwangsläufig robust
- Oft limitierte Fähigkeit zum Lösen komplexer Probleme

Deep Learning

- + Bessere Modell-Fähigkeiten
- + Keine explizite Merkmalsextraktion
- + Keine Merkmalselektion/-reduktion
- + Höchst innovatives Forschungsgebiet

- Schlechter interpretierbar
- Allgemein: mehr Trainingsdaten nötig
- Höhere Rechenkapazitäten nötig

Fähigkeiten des Modells

Interpretierbarkeit des Modells

7.4 - Tiefe Neuronale Netze (DNN)

7.4.2 - Regularisierung

Warum Regularisierung?

Recap Kapitel 3: Overfitting, Underfitting und Generalisierungsfähigkeit von Modellen

⇒ Hohe Generalisierungsfähigkeit von Modellen ist das Ziel, Under- und Overfitting sollen minimal werden

Problem von DNNs:

- können Zusammenhänge zwar sehr gut approximieren, neigen aber zu Overfitting
- mehr Schichten: höheres Risiko für Overfitting
- Challenge: Modell finden, das komplex genug zur Lösung einer Aufgabe ist, aber nicht **zu** komplex

Lösung: große DNNs mit Regularisierungsmethoden kombinieren

Formen der Regularisierung

Häufig verwendete Regularisierungen:

- L₂-Regularisierung ⇒ Bestrafung großer Gewichte auf Basis von L₂-Norm
- Early Stopping ⇒ Abbruch des Trainings, wenn Fehlermaß eine Zeit lang stagniert oder wieder zunimmt
- Data Augmentation ⇒ künstliche Vergrößerung des Trainingsdatensets durch Generierung realistischer Samples aus den vorhandenen Trainingsdaten (Intuition: Mehr Trainingssamples, weniger Overfitting)
- Ensemble Learning ⇒ Abmilderung von Zufallseffekten im Training
- Dropout ⇒ Deaktivierung eines bestimmten Prozentsatzes von zufälligen Neuronen für jeden Batch (Intuition: zwinge das Modell, generalisierte Zusammenhänge zu lernen)

7.5 - Faltende neuronale Netze

7.5 - Faltende neuronale Netze

7.5.1 Warum faltende neuronale Netze?

Nachteile von vollvernetzten neuronalen Netzen

Vollvernetzte neuronale Netze besitzen einige Nachteile:

- Sehr hohe Anzahl zu lernender Parameter
- Schnell sehr hohe Anforderungen an Ressourcen
- Hohes Risiko für Overfitting
- Vollvernetzung sorgt dafür, dass lokale Muster/Merkmale nicht sinnvoll gelernt werden können

⇒ Vollvernetzte Schichten: gut geeignet für Klassifikation und Regression.
Aber *nicht* für das hierarchische Lernen von Merkmalen

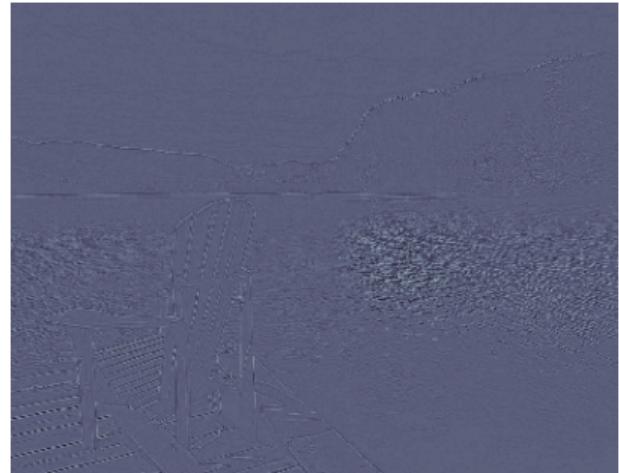
Recap Kapitel 2: Faltungen

Faltungen können genutzt werden, um Merkmale aus Signalen zu extrahieren.

Beispiel: 2-dimensionale Faltung zur Kantendetektion:



$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$



Eigenschaften faltender Schichten

Faltende neuronale Netze (engl. convolutional neural networks, CNNs) machen sich die mathematische Operation der Faltung und deren Eigenschaften zunutze.

Faltende Schichten (engl. convolutional layers):

- falten Eingangsdaten mit einem Kernel
- extrahieren lokale Merkmale
- bilden biologisches Vorbild des rezeptiven Feldes nach ⇒ Neuron reagiert auf Reize einer lokalen Umgebung der vorhergehenden Schicht, nicht das ganze Bild
- reduzieren Größe des Inputs (vgl. Kapitel 2)
- nutzen Parameter-Sharing ⇒ Gewichtsmatrix mit weniger Parametern, als bei vollvernetzter Schicht
- reduzieren Modellkomplexität verglichen mit vollvernetzter Schicht

Modifikationen von Convolutional Layers (1)

Padding

Probleme durch Faltungsoperationen:

- Faltung sorgt dafür, dass Dimension des Bildes verringert wird \Rightarrow tiefe CNNs nicht möglich, weil Bilder zu klein werden
- Informationen in inneren Teilen des Bildes werden stärker gewichtet, als Informationen am Rand

Lösung: Erweiterung des Input-Bildes durch Zero-Padding, sodass bei Faltung die Originalgröße des Inputs erhalten bleibt

Keras: `padding='valid'` entspricht keinem Padding, `padding='same'` entspricht Padding zum Erhalt der Originalgröße des Inputs

Modifikationen von Convolutional Layers (2)

Stride

Der Stride gibt an, um wie viele Pixel der Faltungskernel verschoben wird (Default: 1 Pixel in jede Richtung)

Resultat: stärkeres Downsampling

Dilatation

Verwendet man eine Dilatationsrate $r > 1$ an, wird nur jedes r -te Pixel durch den Faltungskernel beachtet.

Resultat: Rezeptives Feld wird bei gleichbleibender Kernel-Dimension vergrößert

Wiederherstellung der Originalgröße des Inputs

In manchen CNN-Architekturen ist es wichtig, dass der Output die gleichen Dimensionen besitzt, wie der Input (z.B. für Segmentierungen). Um dies zu erreichen, gibt es zwei Möglichkeiten:

1. Upsampling:

Beim Upsampling um einen Faktor s werden Einträge in Zeilen und Spalten eines Bildes s -fach wiederholt.

Keras: `keras.layers.UpSampling2D()`

2. Inverse Faltung:

Statt einfachem Upsampling wird eine Faltung durchgeführt, die das Bild vergrößert, statt verkleinert.

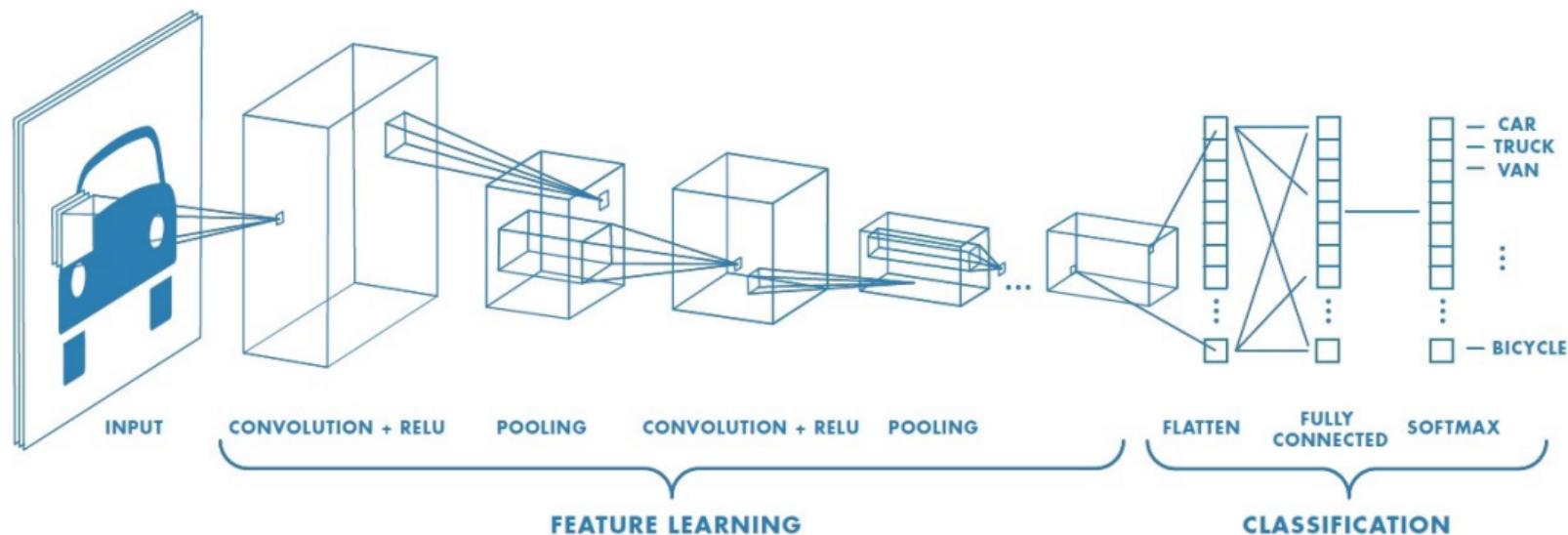
Keras: `keras.layers.Conv2dTranspose()`

7.5 - Faltende neuronale Netze

7.5.2 - Aufbau von CNNs

Grundstruktur eines CNN

CNNs bestehen aus einer Kombination verschiedener Typen von Schichten:¹⁶



¹⁶ Bildquelle: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>

Pooling (1)

Auf eine faltende Schicht folgt in der Regel eine **Pooling-Schicht**.

Grund: Fokus auf zu viele Details kann für das Lernen robuster Feature-Maps hinderlich sein

Zwei Möglichkeiten, diesem Problem zu begegnen:

- Downsampling zwischen Convolutional Layers
- Pooling von Informationen

⇒ Meistens genutzt: Pooling

Pooling (2)

Prinzip:

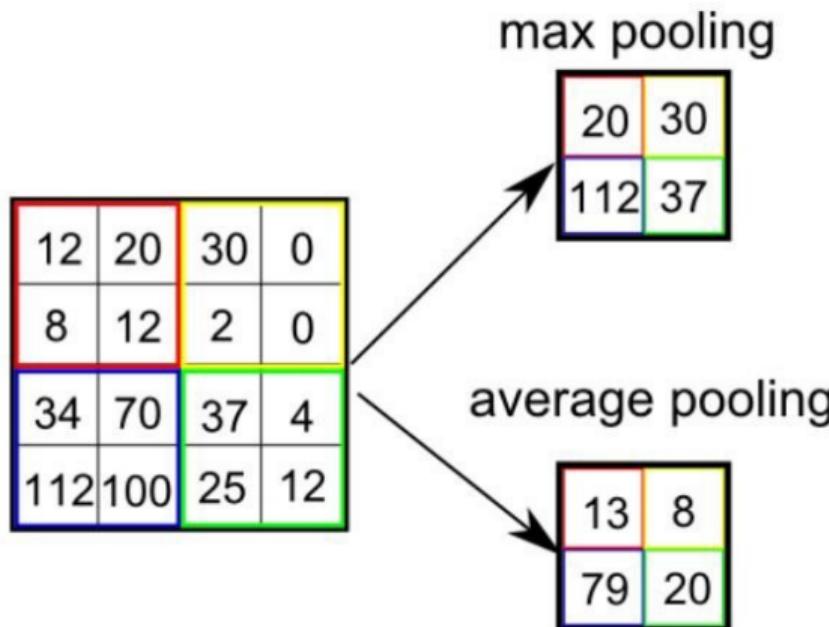
- ein Fenster wird über das Input-Bild geschoben
- Metrik (abhängig von Strategie) wird aus den Bildwerten innerhalb des Fensters berechnet
- berechneter Wert wird stellvertretend für Fensterinhalt an nächste CNN-Schicht weitergegeben

Vorteile von Pooling:

- Verringelter Speicherbedarf \Rightarrow tiefere/komplexere Architektur möglich
- Verbesserung der Toleranz gegenüber leichten Änderungen (z.B. Translationen)
- Rezeptive Felder wachsen automatisch in tieferen Layern
- Reduktion des Risikos von Overfitting

Max Pooling und Average Pooling

Zwei Strategien werden hauptsächlich genutzt: Max und Average Pooling¹⁷



Max Pooling: Nehme den höchsten Wert eines Pooling-Fensters als Stellvertreter-Wert

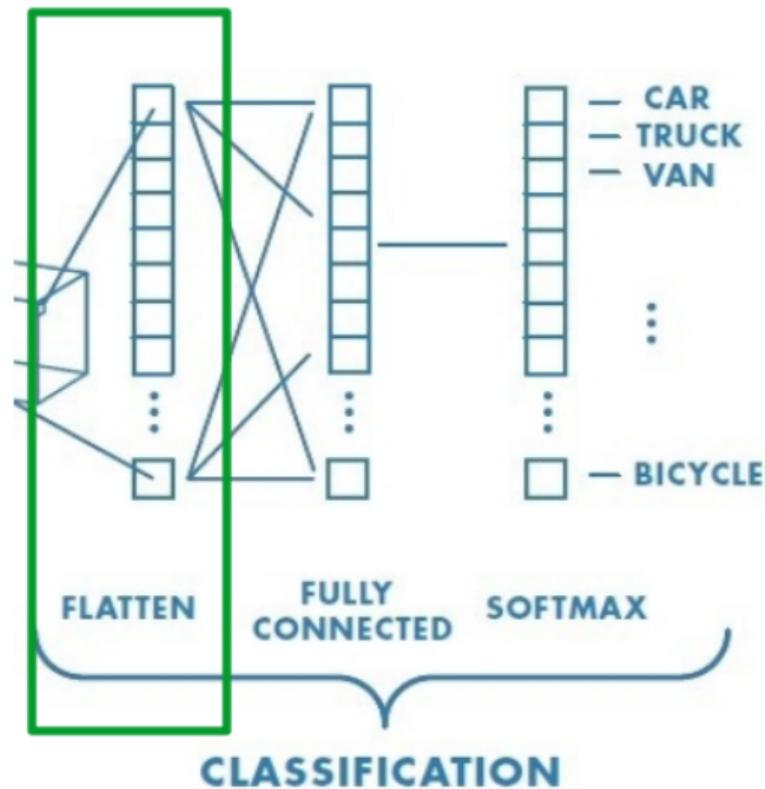
Average Pooling: Nehme den Mittelwert eines Pooling-Fensters als Stellvertreter-Wert

¹⁷ Bildquelle: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

Flattening Layer

Eine Flattening Layer befindet sich typischerweise zwischen faltenden und vollvernetzten Schichten

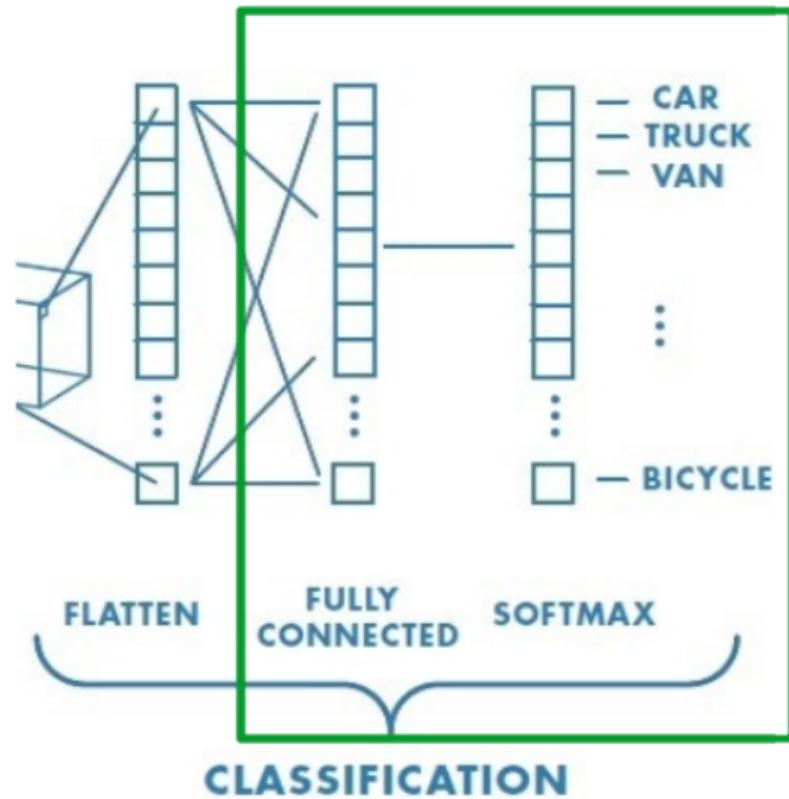
Grund: der Matrix-förmige Output faltender Schichten muss zunächst in Vektor-Form konvertiert werden, um an die nachfolgende vollvernetzte Schicht weitergegeben werden zu können



Dense Layer + Softmax-Aktivierung

Den Abschluss einer CNN-Architektur bilden in der Regel eine Dense-Layer und Softmax-Aktivierung

Grund: wie bei normalen DNNs ermöglicht eine vollvernetzte Schicht mit Softmax-Aktivierung (oder Sigmoid im binären Fall) die Klassifikation des Outputs

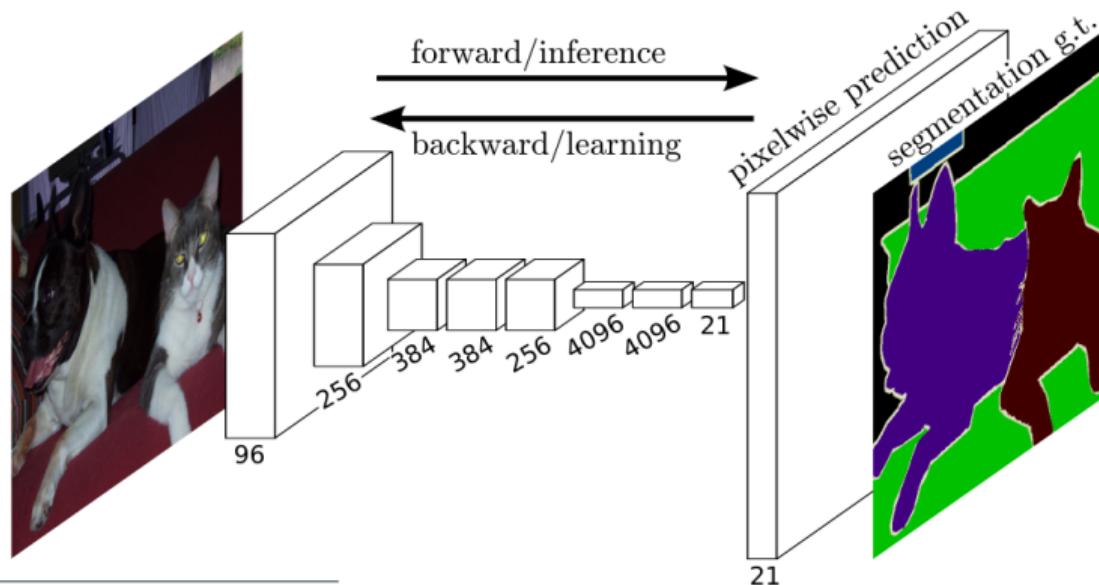


7.5 - Faltende neuronale Netze

7.5.3 - Beliebte Architekturen

Fully Convolutional Networks¹⁸

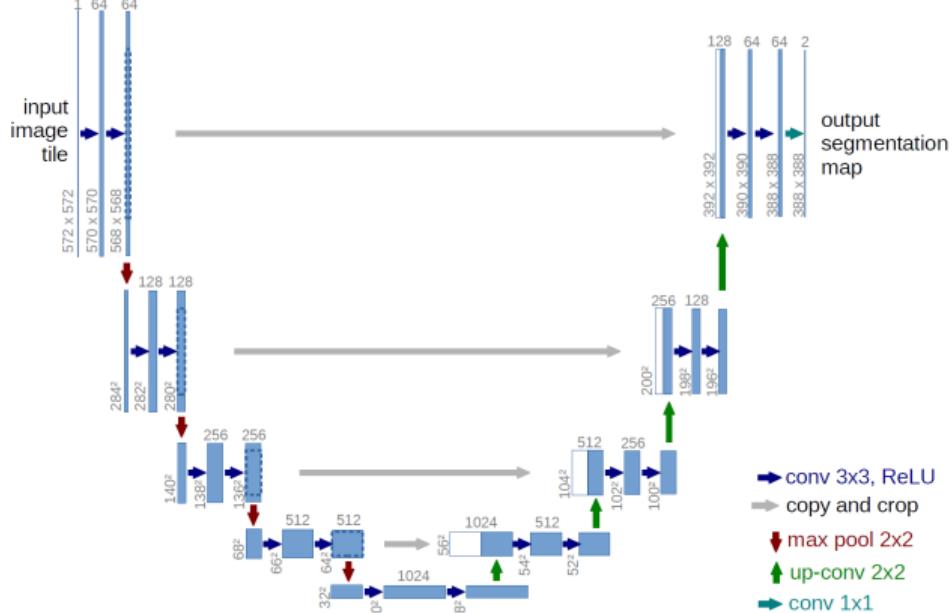
Idee: Ersetzen der vollvernetzten Schichten zur Klassifikation am Ende eines CNNs durch 1×1 -Faltungsschichten mit anschließendem Upsampling zur Pixel-weisen Bildklassifikation für Segmentierung



¹⁸Long et al.: "Fully Convolutional Networks for Semantic Segmentation", 2015

Idee:

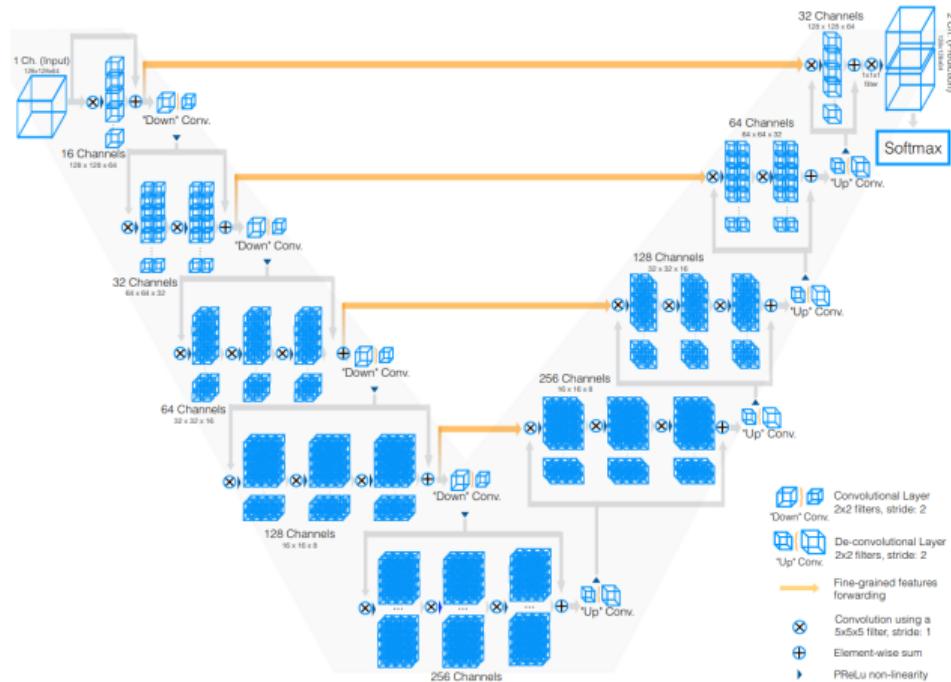
- Basiert auf dem Fully Convolutional Networks
- Upsampling wird durch einen Deconvolution-Pfad ersetzt
- Einsatz von Skip-Connections zwischen Convolution- und Deconvolution-Pfaden
- Name stammt vom U-förmigem Aussehen



¹⁹Ronneberg et al.: "U-Net: Convolutional Networks for Biomedical Image Segmentation", 2015

Idee:

- Erweiterung des U-Net zur 3-dimensionalen Bildsegmentierung
- Ersatz

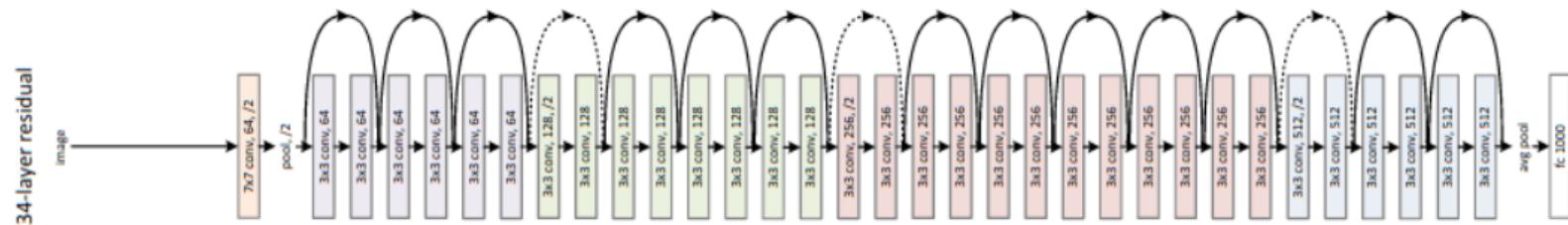
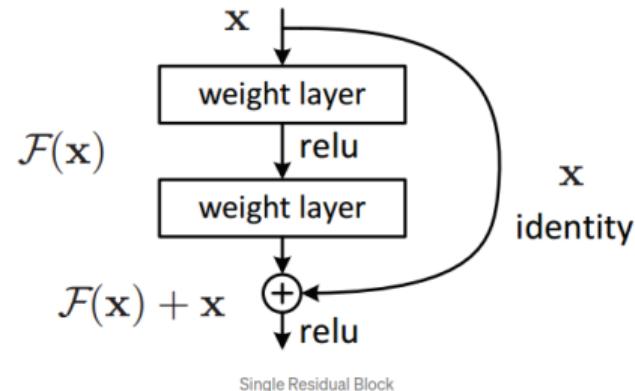


²⁰Milletari et al.: "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation", 2016

Residual Networks²¹

Prinzip:

- Inspiriert durch Pyramidenzellen im cerebralen Kortex
- Aufbau aus *Residual Blocks* (siehe Bild)
- Skip-Connections zwischen Layer vor und nach Residual Block zur Vermeidung von verschwindenden Gradienten

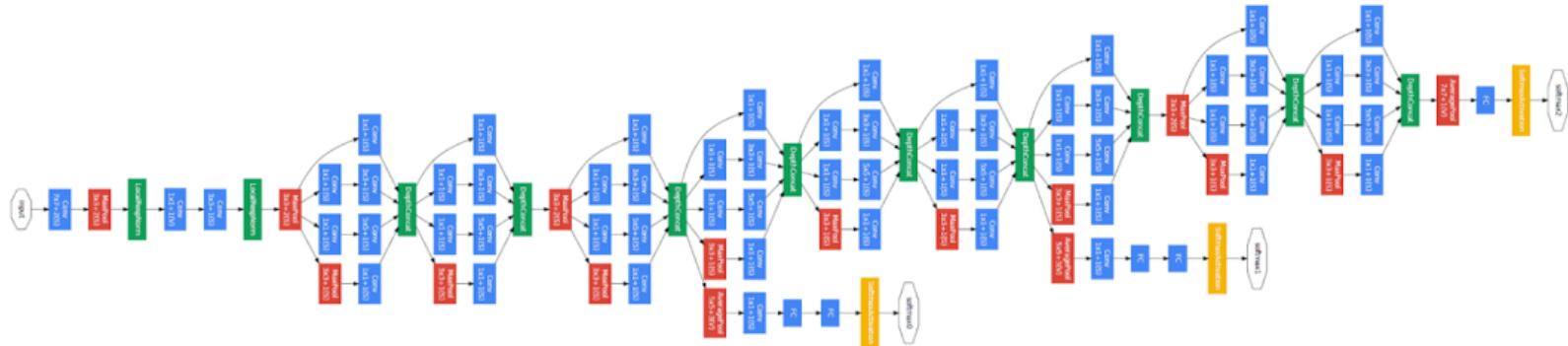
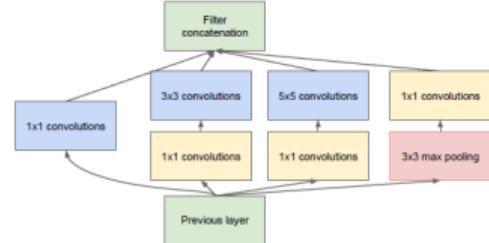


²¹He et al.: "Deep Residual Learning for Image Recognition", 2016

Inception Networks²²

Prinzip:

- Nicht nur in die Tiefe, sondern auch in die Breite gehen
- Aufbau aus *Inception Cells* (siehe Bild)
- Allgemeineres Lernen durch Verwendung unterschiedlich großer Filter-Kernel (\Rightarrow weniger Overfitting)



²²Szegedy et al.: "Going deeper with convolutions", 2014

7.5 - Faltende neuronale Netze

7.5.4 - Anwendungsbeispiele

Beispiel 7.3: Bildsegmentierung für Autonomes Fahren⁵(1)

Aufgabe: Semantische Segmentierung des Cityscapes-Datensets

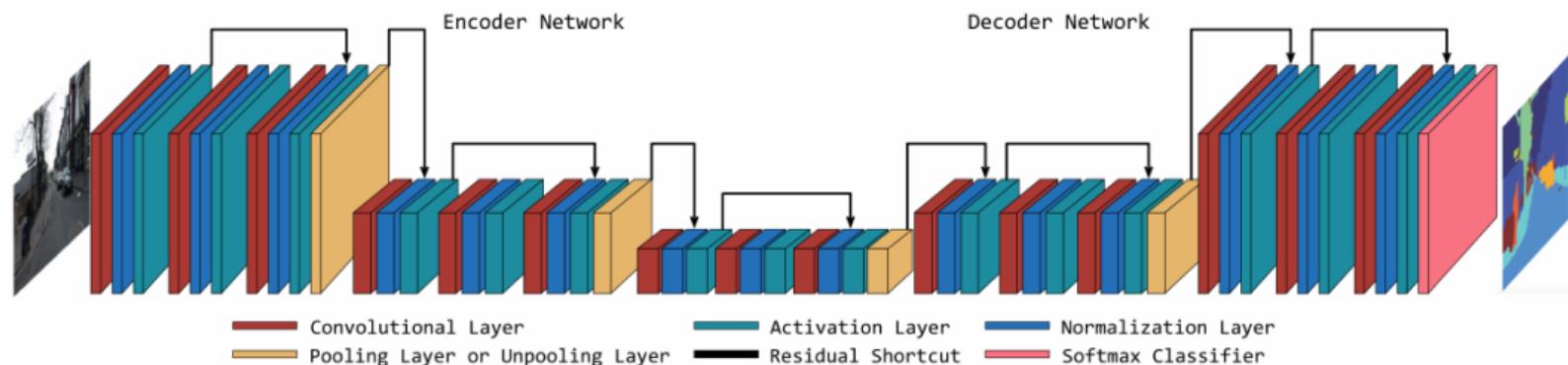


Fig. 1. Architecture. Notice that the network is trimmed for better presentation. More layers were used in our experiments.

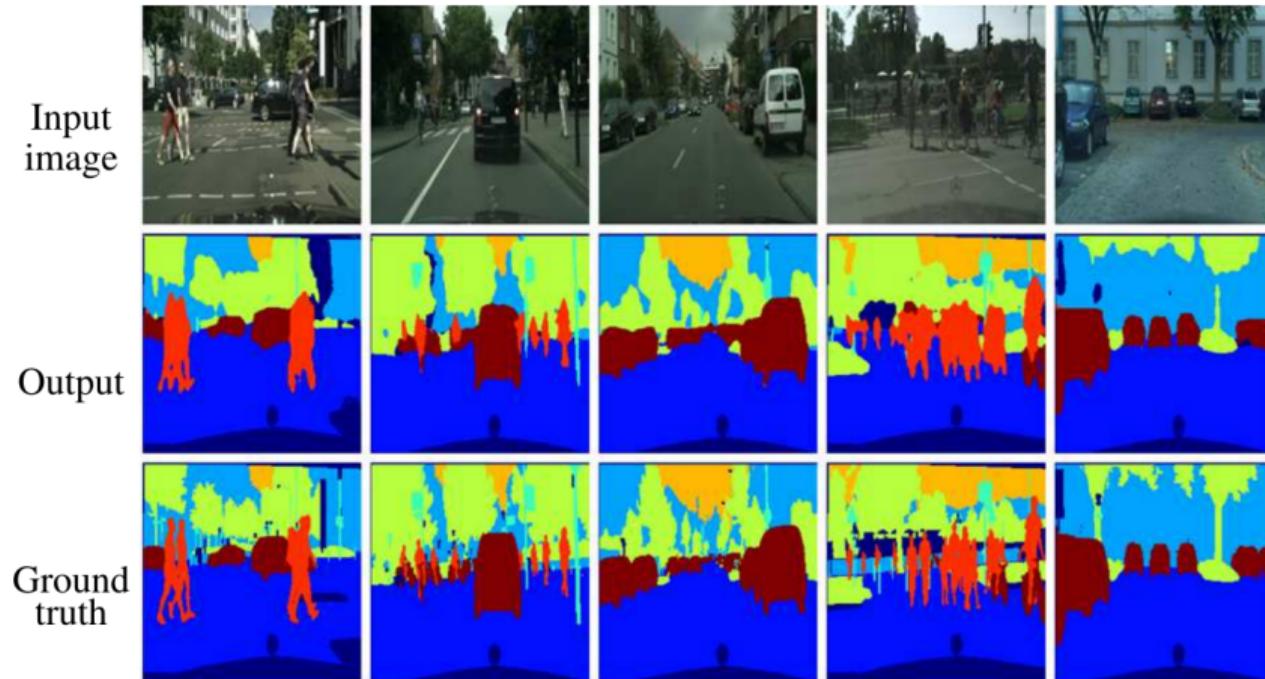
Table 3. Comparison of segmentation results on Cityscape Dataset

Method	Road	Construction	Object	Nature	Sky	Human	Vehicle	Global	Average	Mean IoU
Without Shortcuts	94.9	88.0	9.0	94.0	88.9	47.0	93.0	89.1	73.5	63.4
With Shortcuts	98.0	89.9	36.0	93.0	91.0	69.9	88.9	92.3	80.9	73.1

⁵ Wang et al.: "On semantic image segmentation using deep convolutional neural network with shortcuts and easy class extension", IPTA, 2016

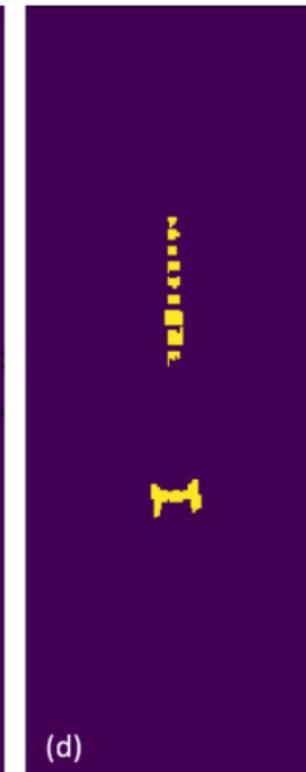
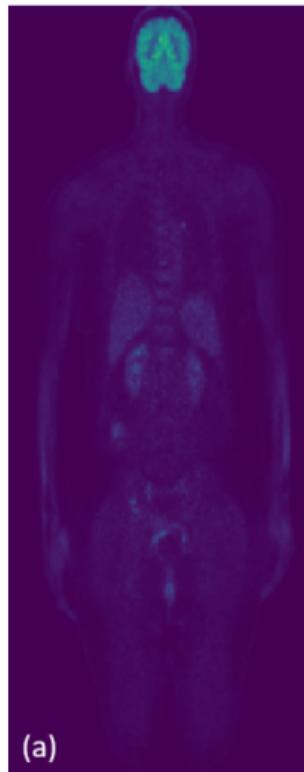
Beispiel 7.3: Bildsegmentierung für Autonomes Fahren⁵(2)

Visuelle Ergebnisse:



⁵ Wang et al.: "On semantic image segmentation using deep convolutional neural network with shortcuts and easy class extension", IPTA, 2016

Beispiel 7.4: Organsegmentierung in PET-Bildern²³(1)



Task:

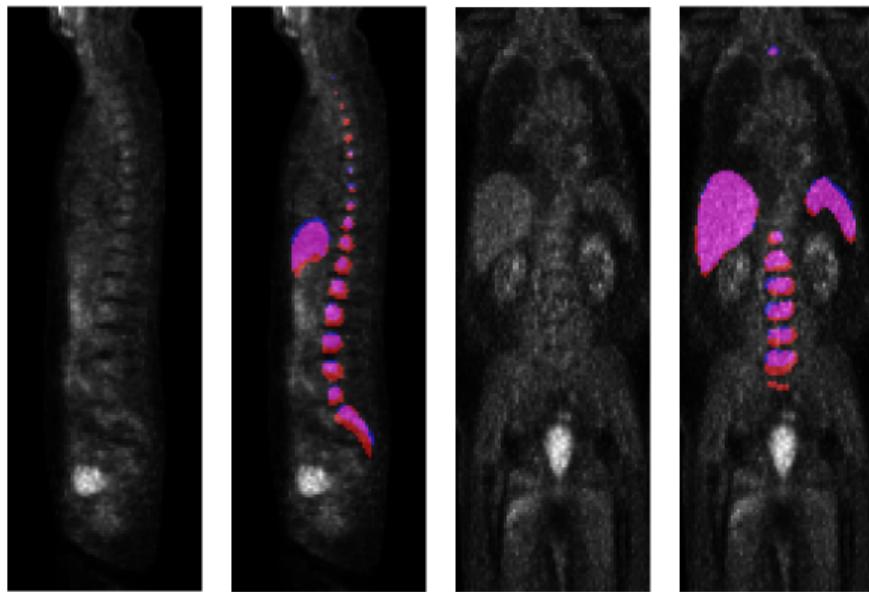
Segmentierung
von Leber, Milz
und Wirbelsäule
in PET-Bildern
mit CNNs

Challenge:
schlechte
räumliche
Auflösung von
PET-Bildern

²³ Liebgott et al., "Automated Multi-Organ Segmentation in PET Images Using Cascaded Training of a 3D U-Net and Convolutional Autoencoder", under review

Beispiel 7.4: Organsegmentierung in PET-Bildern²³

Architektur: kaskadiertes Training von 3D U-Net und convolutional Autoencoder



Ergebnisse:

- Dice-Score Hintergrund: 0.99
- Dice-Score Leber: 0.88
- Dice-Score Milz: 0.82
- Dice-Score Wirbelsäule: 0.59

	background	liver	spleen	spine
background	9,204,087	23,406	5,129	19,606
liver	12,550	118,857	3	0
spleen	3,337	0	19,492	0
spine	10,226	0	0	20,494

²³ Liebgott et al., "Automated Multi-Organ Segmentation in PET Images Using Cascaded Training of a 3D U-Net and Convolutional Autoencoder", under review

7.6 - Autoencoder

Was sind Repräsentationen?

Aufgenommene Signale können verschiedene Darstellungsformen haben, die ihren Wert repräsentieren. Rohdaten sind nicht immer die optimal für jede Anwendung.

Veranschaulichung: Zahlen.

Die Bedeutung von Zahlen verändert sich nicht. Ihre Darstellung kann aber sehr unterschiedlich sein und für unterschiedliche Aufgaben geeignet sein:

Arabische Zahlen sind besser zum Rechnen geeignet, als römische Zahlen

Binäre Zahlen sind die geeignete Repräsentation von Zahlen auf Computern

Passende Repräsentationen zu finden ist wichtiger Bestandteil maschinellen Lernens.

Konventionelles maschinelles Lernen:

Merkmale müssen so gewählt werden, dass sie die Rohdaten so repräsentieren, dass die Machine-Learning-Aufgabe möglichst gut gelöst werden kann

Deep Learning:

Neuronale Netze lernen selbstständig, die Daten möglichst gut zu repräsentieren, um eine Aufgabe zu lösen.

Unüberwachtes Erlernen von Repräsentationen wird in der Regel durch Autoencoder realisiert

Prinzip eines Autoencoders

Idee:

- Unüberwachtes Lernverfahren
- Training eines Neuronalen Netzes zum eigenständigen extrahieren wichtiger Merkmale, anhand derer Input rekonstruiert werden kann
- Encoder-Decoder-Struktur
- Evaluation basierend auf Ähnlichkeit zwischen Input und Rekonstruktion

Ziel eines Autoencoders: Erlernen einer Variable \underline{z} , die eine versteckte und komprimierte Repräsentation des Inputs \underline{x} im sogenannten Latent Space ist.

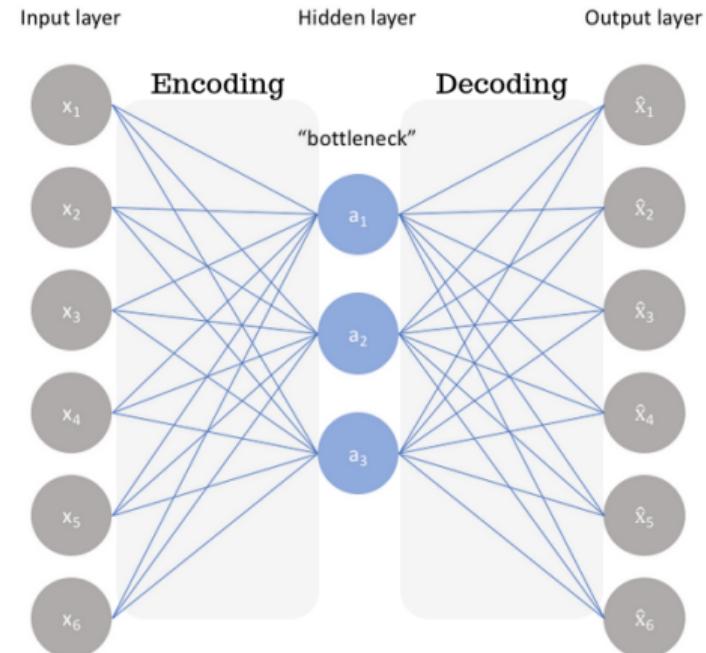
Anmerkung: eine exakte Rekonstruktion von \underline{x} aus \underline{z} (d.h. Generierung einer detaillierten Kopie von \underline{x}) ist nicht erstrebenswert. Dies würde das Erlernen der wichtigsten Eigenschaften verhindern.

Aufbau von Autoencodern²⁴

Encoder: Eine oder mehrere Schichten, mit denen Eigenschaften von Input \underline{x} extrahiert werden

Bottleneck: Latent-Space, codierte Version des Inputs

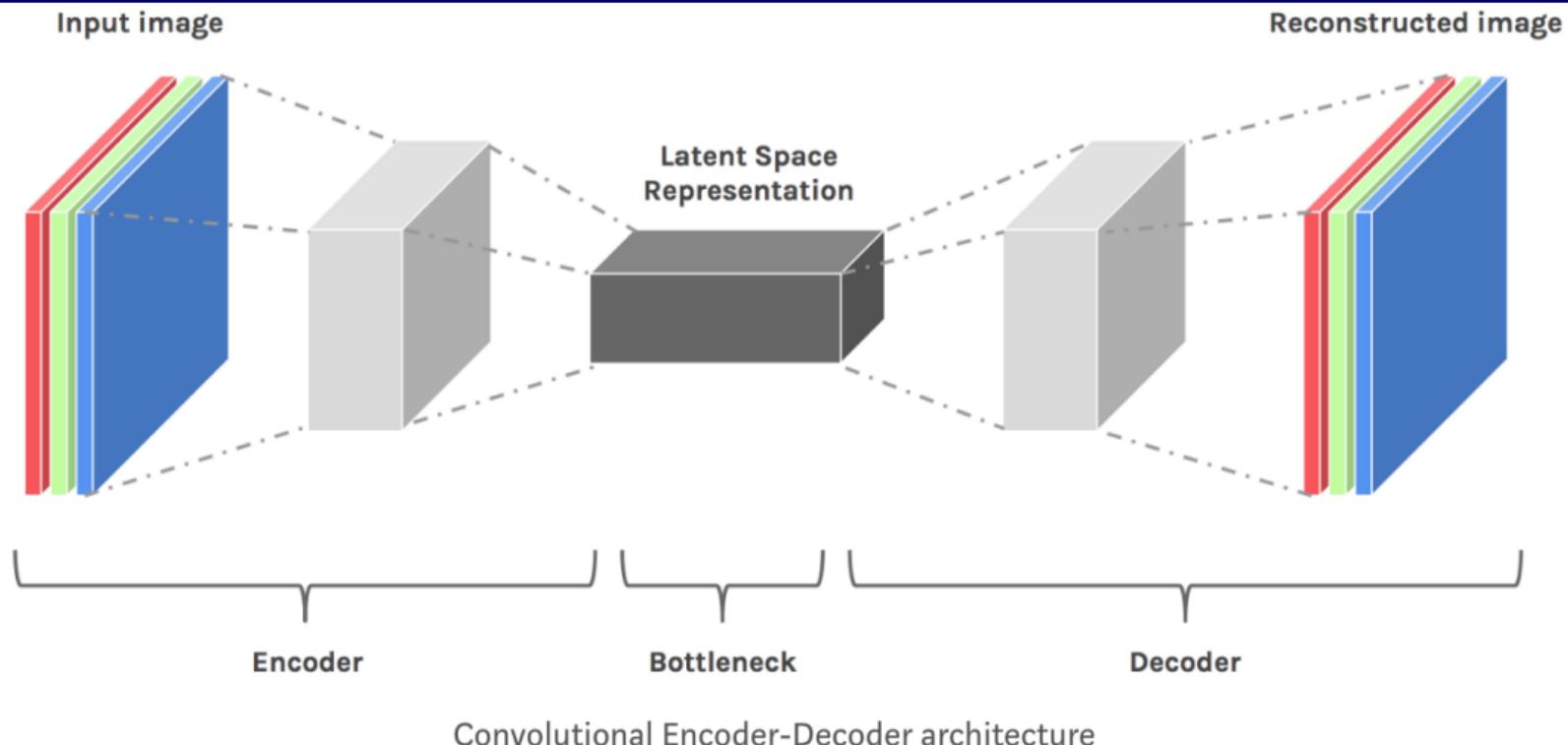
Decoder: Eine oder mehrere Schichten, mit denen aus den Latent Space Features der Input rekonstruiert wird



²⁴Bildquelle:

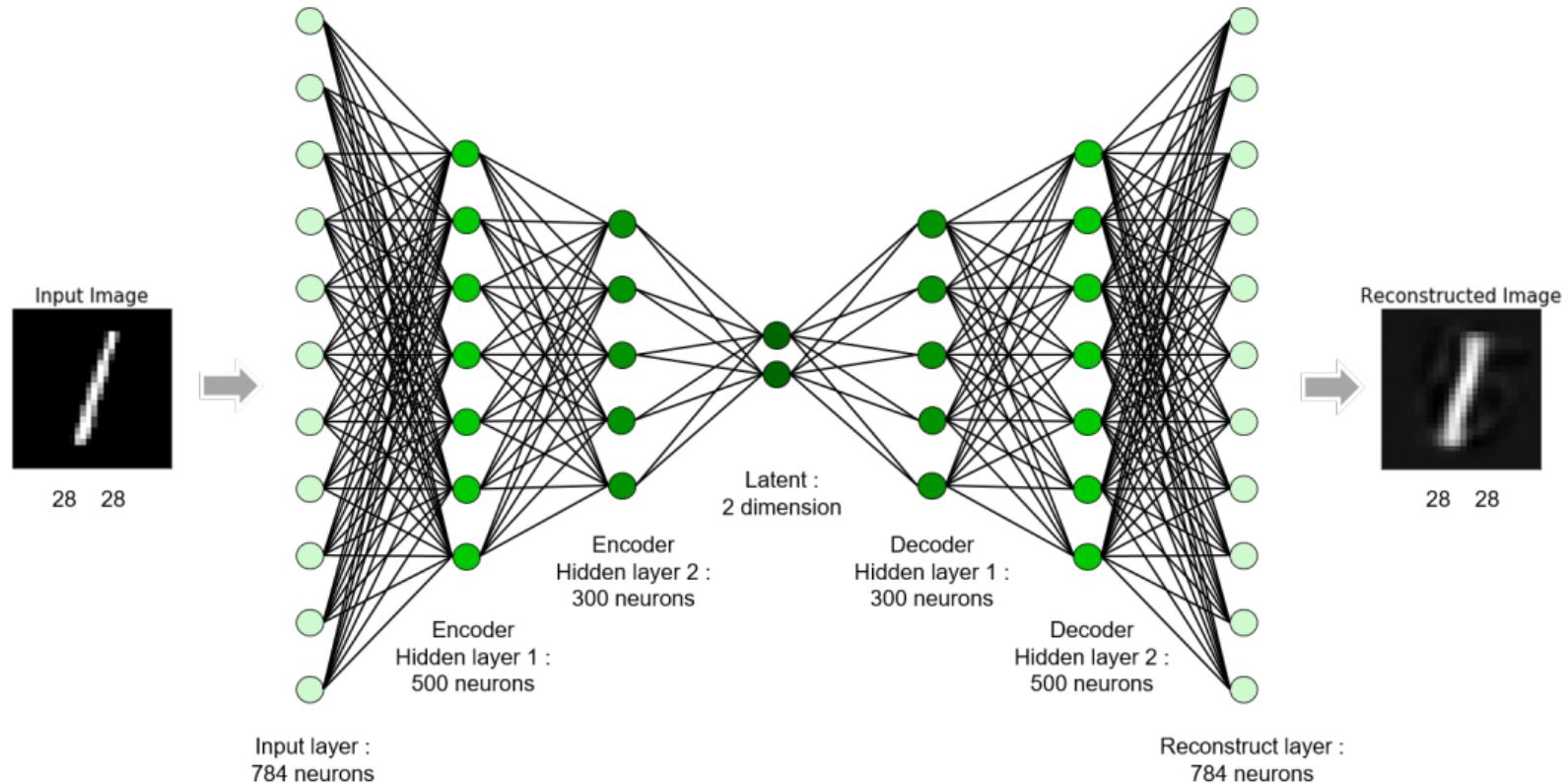
<https://medium.com/analytics-vidhya/autoencoders-denoising-understanding-b41315fd7fa>

Convolutional Autoencoder²⁵



²⁵ Bildquelle: <https://towardsdatascience.com/how-to-generate-new-data-in-machine-learning-with-vae-variational-autoencoder-applied-to-mnist-ca68591acdcf>

Beispiel 7.5: MNIST²⁶



²⁶Bildquelle: <https://medium.com/@encodebox/auto-encoder-in-biology-9264da118b83>

Anwendungsbeispiele

Autoencoder werden in vielen Gebieten angewendet, z.B.:

- Dimensionsreduzierung²⁷
- Anomaliedetektion²⁸
- Bildkompression²⁹
- Denoising, z.B. in medizinischen Bildern³⁰
- Social Media, z.B. Popularity-Prediction für Social Media Posts³¹

²⁷Hinton und Salakhutdinov: "Reducing the Dimensionality of Data with Neural Networks", 2006

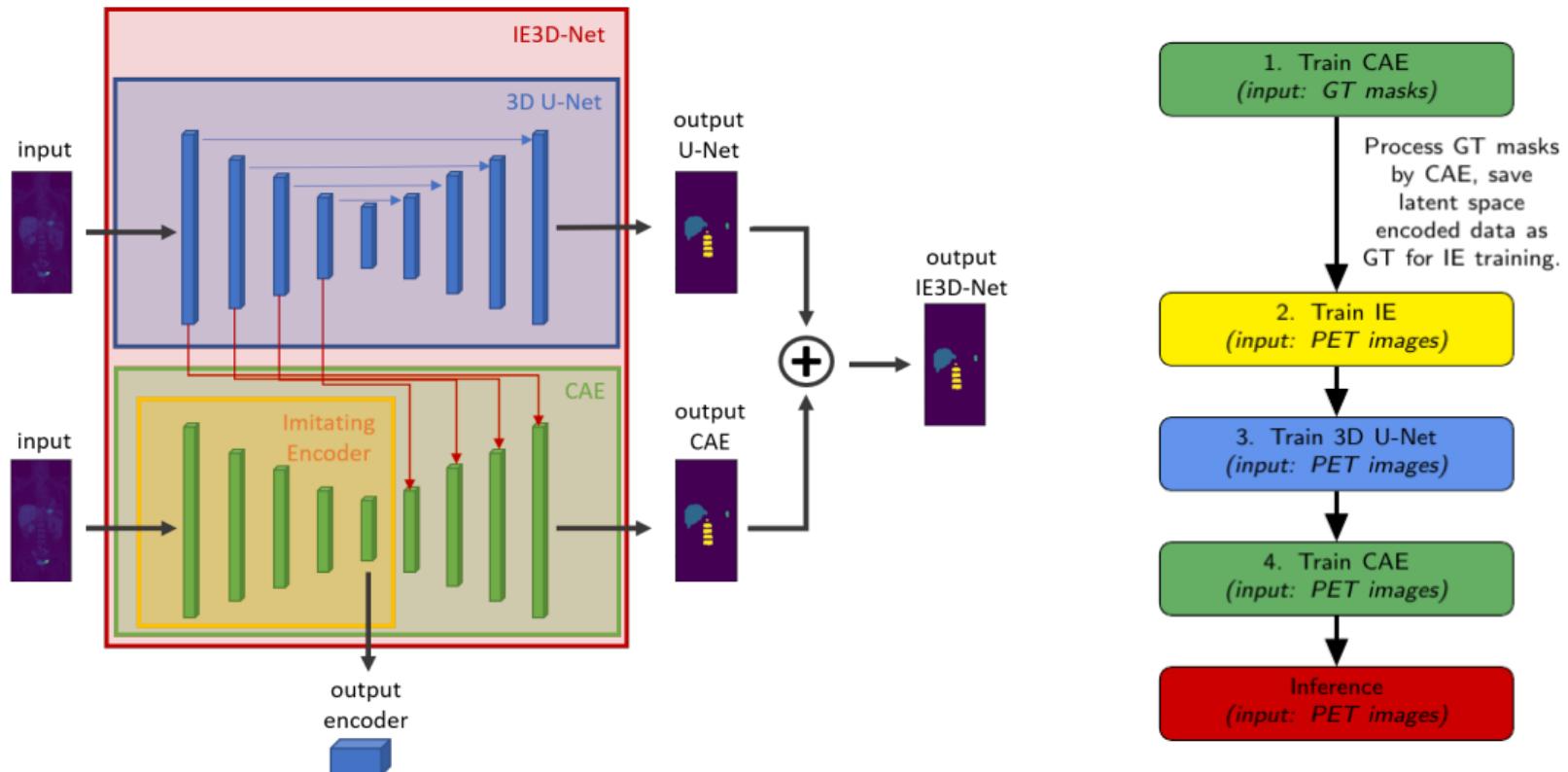
²⁸Ribeiro et al.: "A study of deep convolutional auto-encoders for anomaly detection in videos", 2018

²⁹Theis et al.: "Lossy Image Compression with Compressive Autoencoders", 2017

³⁰L. Gondara: "Medical Image Denoising Using Convolutional Denoising Autoencoders", 2016

³¹De et al.: "Predicting the popularity of instagram posts for a lifestyle magazine using deep learning", 2019

Beispiel 7.6: Organsegmentierung in PET-Bildern²³ (Architektur)



²³ Liebgott et al., "Automated Multi-Organ Segmentation in PET Images Using Cascaded Training of a 3D U-Net and Convolutional Autoencoder", under review

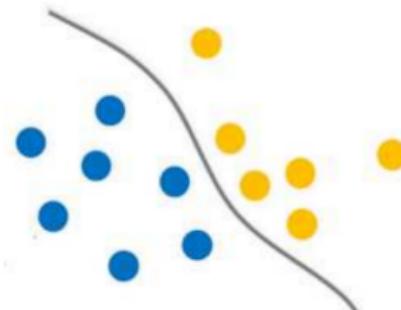
7.7 - Generative Modelle

Diskriminative vs. generative Modelle

Man unterscheidet zwischen diskriminativen und generativen ML-Modellen.

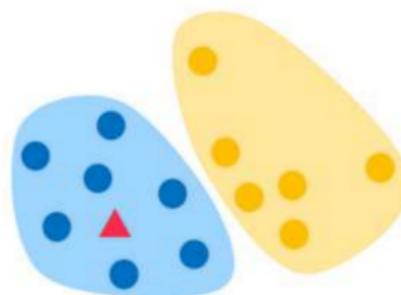
Diskriminative Modelle:

Hauptsächlich für Klassifikation und Regression genutzt, der Fokus liegt auf Lernen der Entscheidungsgrenze



Generative Modelle:

Hauptsächlich für Generation neuer Daten genutzt, der Fokus liegt auf dem Lernen der Datenverteilung



Generative Modelle sind im Allgemeinen herausfordernder

Beispiel 7.7: diskriminative vs. generative Modelle (1)

a) Bild \underline{x} und Stil eines Bildes y^{32}



Diskriminatives Modell: kann zwischen den beiden Stilen unterscheiden, kann die Bilder aber nicht malen

Generatives Modell: kann Bilder in beiden Stilen malen und sie unterscheiden

³²Bildquelle: Vorlesung "Deep Learning", Prof. Dr.-Ing. Bin Yang, Universität Stuttgart

Beispiel 7.7: diskriminative vs. generative Modelle (2)

b) Gesprochenes Wort \underline{x} und Sprache y

Generativ sind wir in Sprachen, die wir beherrschen (z.B. Deutsch, Englisch).

Diskriminativ können wir auch in Sprachen sein, die wir selber nicht sprechen, aber am Klang unterscheiden können (z.B. Arabisch, Chinesisch, Hindi)

Wozu nützen generative Modelle?

Generative Modelle werden hauptsächlich eingesetzt, um neue Daten zu generieren.

Dies kann aus verschiedenen Gründen nützlich sein, z.B:

- um Trainingsdaten im maschinellen Lernen zu erweitern, wenn die Aufnahme ausreichender Mengen realer Daten nicht möglich ist
- um Daten zu erzeugen, deren reale Erzeugung kostspielig ist
- um ein System robust gegen falsche, aber sehr realistisch wirkende Daten zu machen

Prinzip:

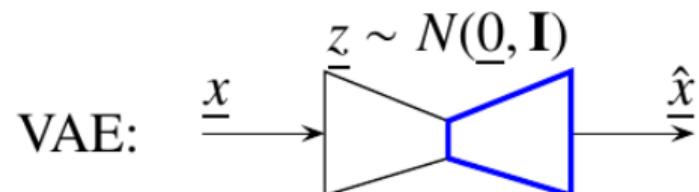
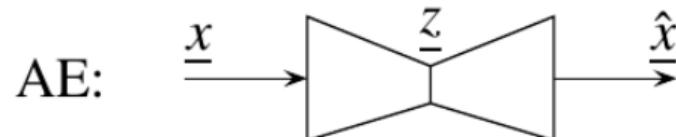
- Encoder-Decoder-Struktur
- Input-Daten werden durch den Encoder im Latent Space codiert
- Modell lernt die Verteilung der Daten im Latent Space
- Nach dem Training ist der Decoder in der Lage, aus der Latent-Space-Verteilung neue Daten zu generieren

³³Kingma und Welling: "Auto-Encoding Variational Bayes", 2013

Variational Autoencoder vs. Autoencoder

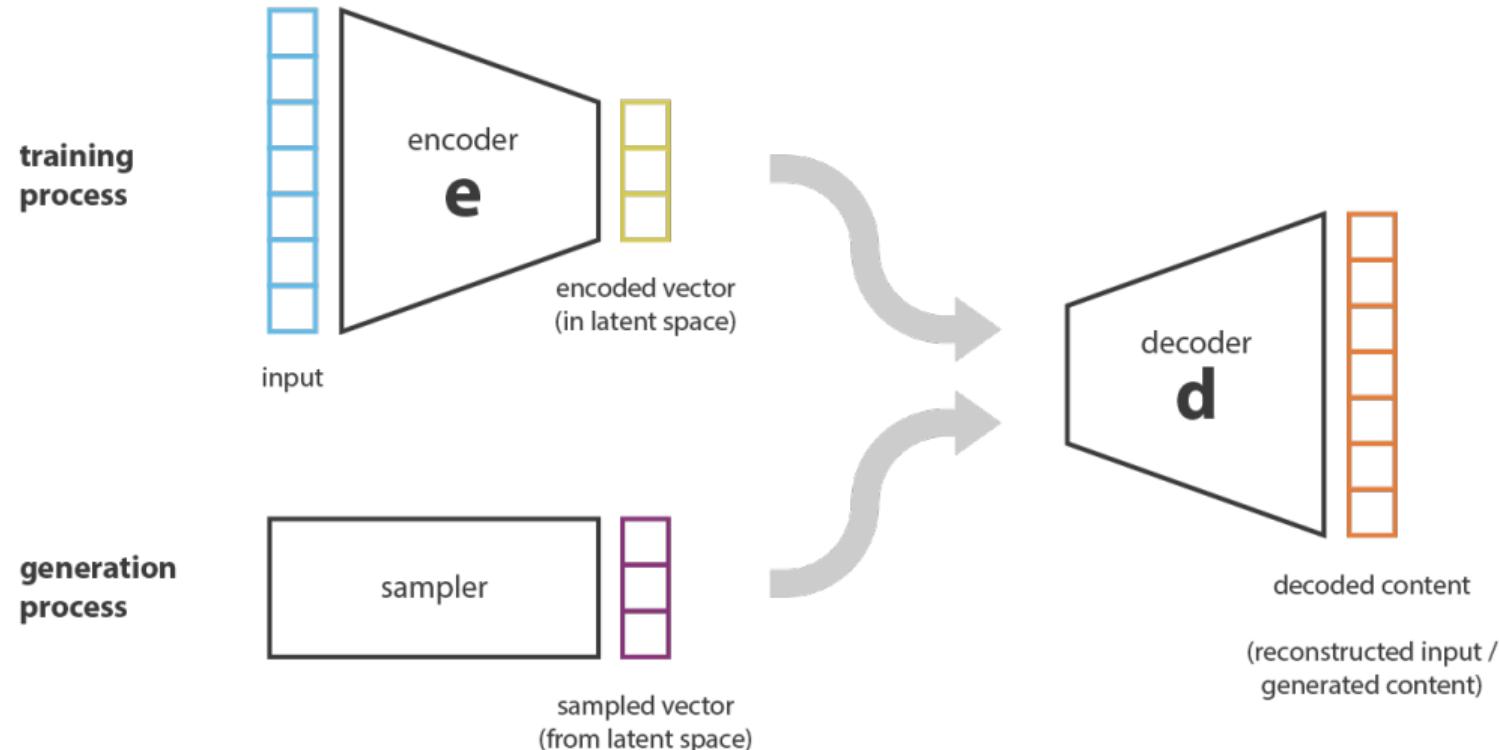
Unterschied zwischen einem VAE und Autoencoder³⁴:

Ein Autoencoder kann ohne einen Input \underline{x} keine Daten erzeugen. Ein VAE hingegen kann nach dem Training aus der Verteilung $\underline{z} \sim \mathcal{N}$ mithilfe des Decoders ohne Input neue Samples kreieren.



³⁴Bildquelle: Vorlesung "Deep Learning", Prof. Dr.-Ing. Bin Yang, Universität Stuttgart

VAE-Architektur³⁵



³⁵Bildquelle: <https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/>

Anwendungsbeispiele VAE

Einige Beispiele, in denen VAEs zur Anwendung kommen:

- Anomalie-Detektion³⁶
- Robotik³⁷
- Sprachverarbeitung³⁸
- Regressionsanalyse, z.B. Analyse von Alterungsprozessen im Gehirn³⁹
- Novelty Detection⁴⁰
- Denoising⁴¹

³⁶ Pol et al.: "Anomaly Detection with Conditional Variational Autoencoders", 2019

³⁷ Inoue et al.: "Transfer learning from synthetic to real images using variational autoencoders for robotic applications", 2017

³⁸ Blaauw und Bonada: "Modeling and transforming speech using variational autoencoders", 2016

³⁹ Zhao et al.: "Variational AutoEncoder for Regression: Application to Brain Aging Analysis", 2019

⁴⁰ Vasilev et al.: "q-Space Novelty Detection with Variational Autoencoders", 2020

⁴¹ Jung et al.: "Joint Learning using Denoising Variational Autoencoders for Voice Activity Detection", 2018

Generative Adversarial Networks⁴²

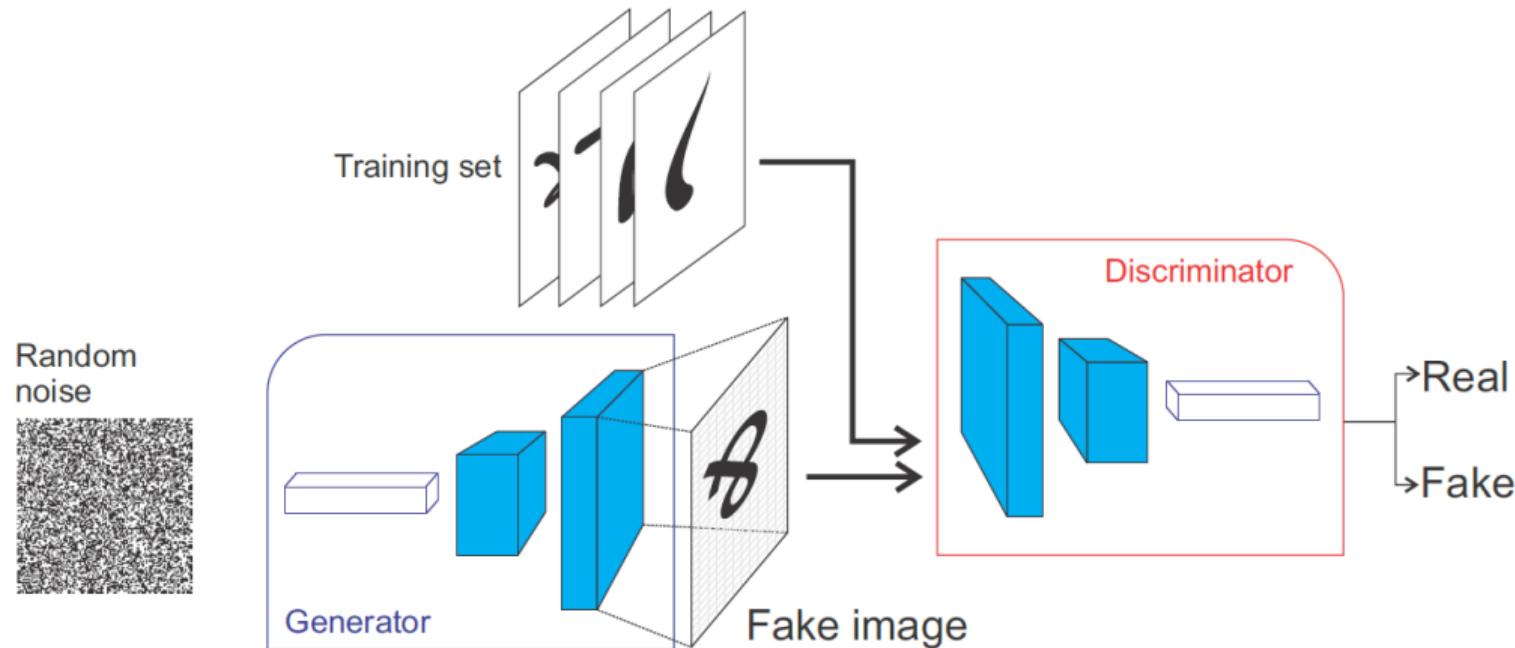
Prinzip:

- Architektur besteht aus zwei Netzwerken: Generator und Diskriminator
- Generator und Diskriminator werden als Konkurrenten trainiert
- Generator: versucht, aus Rauschen möglichst realistische Daten zu erstellen
- Diskriminator: versucht, reale Samples von den durch den Generator erstellten zu unterscheiden

⇒ Grundgedanke: Generator wird trainiert, immer realistischere Samples zu erstellen. Diskriminator wird trainiert, immer realistischere unechte Samples von den echten Samples unterscheiden zu können. Dabei spornen sie sich sozusagen gegenseitig an.

⁴²Goodfellow et al.: "Generative Adversarial Networks", 2014

GAN-Architektur⁴³



⁴³ Bildquelle: <https://www.oreilly.com/library/view/java-deep-learning/9781788997454/60579068-af4b-4bbf-83f1-e988fbe3b226.xhtml>

Veranschaulichung: Falschgeld

Beispiel: Geldfälscher-Bande (Generator) und Kriminalpolizei (Diskriminator)

Eine Geldfälscher-Bande bringt Falschgeld in Umlauf \Rightarrow Polizei entdeckt Falschgeld

Falschgeld entdeckt und Banken sind gewarnt \Rightarrow Geldfälscher müssen sich mehr Mühe bei der Fälschung geben

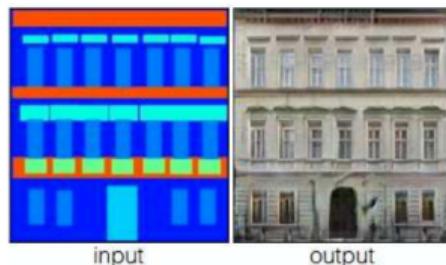
Geldfälscher bringen bessere Blüten in Umlauf \Rightarrow Polizisten müssen lernen, Fake-Scheine von echten zu unterscheiden

Auf Dauer: je besser die Geldfälscher werden, desto schwerer fällt es den Polizisten, die falschen Scheine zu erkennen. Je besser die Polizisten lernen, Falschgeld zu erkennen, desto mehr Mühe müssen sich die Geldfälscher geben. Irgendwann sind sie in der Lage, so gute Fälschungen zu erstellen, dass man sie kaum noch von echtem Geld unterscheiden kann.

Beispiel 7.8: Image-to-Image-Translation mit cGAN⁶

Aufgabe: Übertragung eines gegebenen Datensatzes in eine andere Darstellungsform

facade labels → house photo



edges → bag



black/white image → color image



day photo → night photo



⁶Isola: "Image-to-image translation with conditional adversarial networks", 2017

Anwendungsbeispiele GANs

GANs finden Anwendung in vielen Bereichen, z.B.:

- Modebranche und Werbung⁴⁴ - GANs können genutzt werden, um künstliche Models zu erzeugen, anstatt reale Models zu nutzen, für die zusätzlich weitere Kosten (Fotograf, Stylist, Location, Transportkosten,...) anfallen
- Videospiele⁴⁵ - GANs werden beispielsweise zum Erstellen realistischer Charaktere, von Landschaften, zum Verbessern der Grafik alter Spiele oder zum eigenständigen Erstellen ganzer Levels eingesetzt
- Veränderung von Bildern, z.B. simulierter Alterungsprozess⁴⁶
- Text-to-Image-Translation, z.B. Schätzung des Aussehens eines Menschen basierend auf Sprachaufnahmen⁴⁷

⁴⁴ Wong: "The Rise of AI Supermodels" <https://www.cdotrends.com/story/14300/rise-ai-supermodels>

⁴⁵ Torrado et al.: "Bootstrapping Conditional GANs for Video Game Level Generation", 2020

⁴⁶ Antipov et al: "Face Aging With Conditional Generative Adversarial Networks", 2017

⁴⁷ Oh et al.: "Speech2Face: Learning the Face Behind a Voice", 2019

7.8 - Recurrent Neural Networks

Recurrent Neural Networks

Bisher angesprochene Neuronale Netze betrachten Daten als einzelne Samples.

Problem: in manchen Anwendungsfällen wäre es sinnvoll, Daten als Sequenz zu betrachten (z.B. Zeitreihen, Genome, Texte)

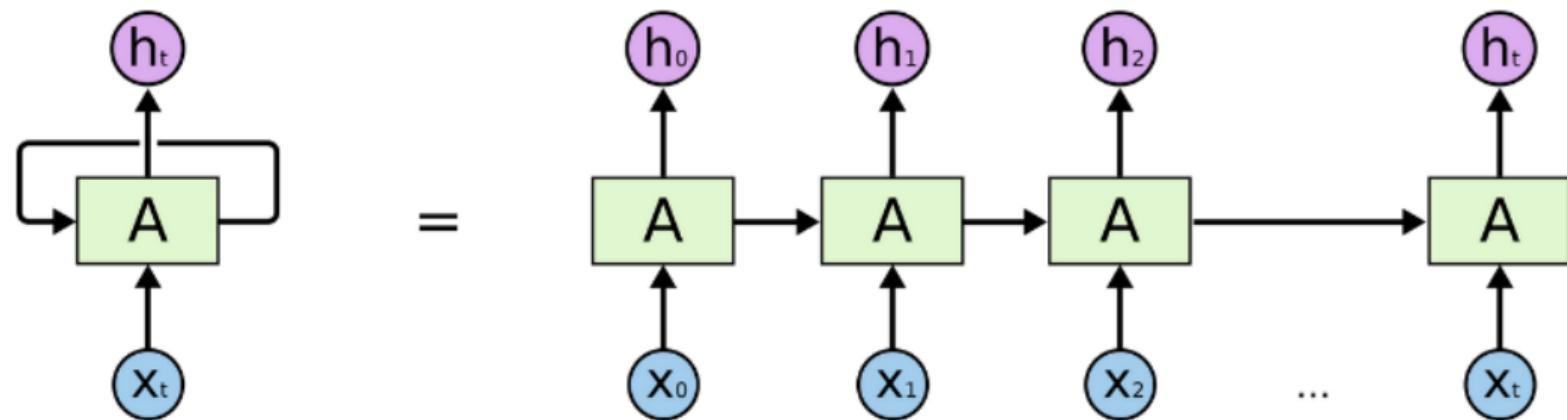
⇒ **Recurrent Neural Networks (RNN)**

Unterschied zwischen RNNs und Standard-DNNs/CNNs:

RNNs besitzen eine zusätzliche *zeitliche Dimension*, die es erlaubt, dass vergangene Ereignisse die Verarbeitung des aktuellen Samples beeinflussen

Aufbau von RNNs⁴⁸

Prinzip: Integration einer zusätzlichen Verbindung zur Weitergabe von Informationen an den nächsten Schritt im Training



⇒ RNN funktioniert wie Kaskade miteinander verbundener einfacher neuronaler Netze

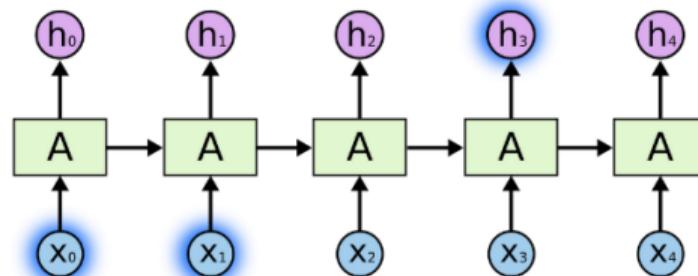
⁴⁸Bildquelle: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Kurz- und Langzeitgedächtnis in RNNs⁴⁸ (1)

Standard-RNNs funktionieren sehr gut, wenn relevante Informationen kurz zurück liegen.

Beispiel Sprachverarbeitung: Ein RNN-Modell soll das nächste Wort vorhersagen, um folgenden Satz zu vollenden: “Ein Fisch schwimmt im Wasser.”

Die relevanten Informationen (“Fisch” und “schwimmt”) liegen nicht weit in der Vergangenheit und kann leicht vorhergesagt werden.



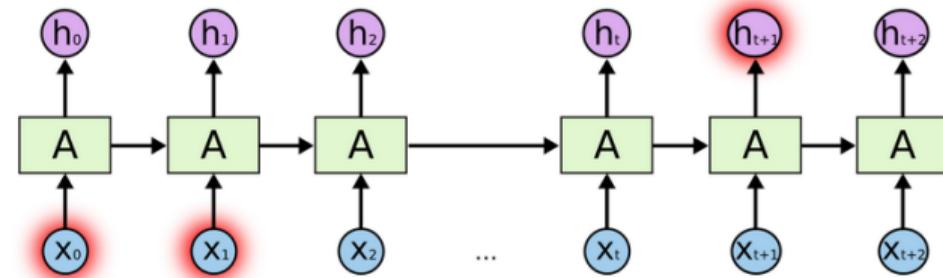
⁴⁸Bildquelle: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Kurz- und Langzeitgedächtnis in RNNs⁴⁸ (2)

Problem: je mehr Kontext benötigt wird, desto schwerer wird es für ein Standard-RNN, etwas vorherzusagen.

Beispiel Sprachverarbeitung: Der Satz lautet diesmal “Ich bin als Kind in Frankreich aufgewachsen. Und später habe ich dort jedes Jahr meine Großeltern mehrmals besucht. Deshalb spreche ich fließend *Französisch*”.

Die relevante Information (“Frankreich”) liegt schon weiter zurück, zu weit für Standard-RNNs.



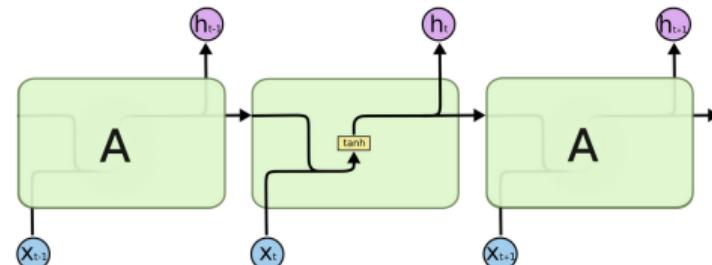
⁴⁸Bildquelle: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Long Short Term Memory Networks⁴⁹

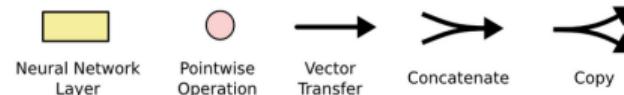
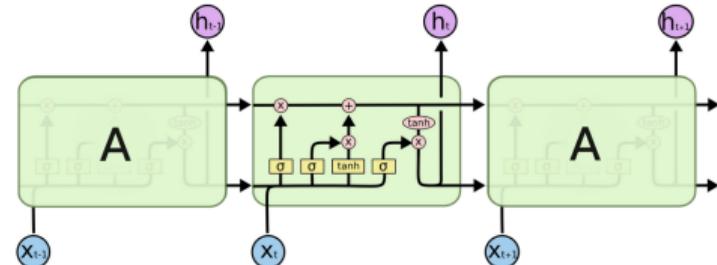
Lösung: Long Short Term Memory Networks (LSTMs) besitzen ein “Langzeitgedächtnis” und werden heute hauptsächlich eingesetzt.

Unterschied zwischen Standard-RNN und LSTM⁴⁸:

Standard-RNN:



LSTM:



⁴⁸Bildquelle: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

⁴⁹Hochreiter und Schmidhuber: “Long Short-Term Memory”, 1997

Anwendungsbeispiele

LSTMs werden beispielsweise angewendet für

- Robotik⁵⁰
- Medikamenten-Design⁵¹
- Musik-Komposition⁵²
- Verkehrsregelung⁵³
- Sprachverarbeitung⁵⁴
- Bioinformatik⁵⁵

⁵⁰ Mayer et al.: "A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks", 2006

⁵¹ Gupta et al.: "Generative Recurrent Networks for De Novo Drug Design", 2018

⁵² Eck und Schmidhuber: "Learning the Long-Term Structure of the Blues. Artificial Neural Networks", 2002

⁵³ Zhao et al.: "LSTM network: A deep learning approach for Short-term traffic forecast", 2017

⁵⁴ Schmidhuber et al.: "Learning nonregular languages: A comparison of simple recurrent networks and LSTM", 2002

⁵⁵ Thireou und Reczko: "Bidirectional Long Short-Term Memory Networks for Predicting the Subcellular Localization of Eukaryotic Proteins", 2007

7.9 - Weiterführende Konzepte

Transfer Learning (1)

Wir haben gelernt: Deep-Learning-Modelle können sehr mächtige Tools sein.

Aber: um komplexe Netze von Grund auf zu trainieren, benötigt man große Mengen Trainingsdaten, hohe Rechenkapazitäten und oftmals viel Zeit.

Mögliche Lösungsstrategie: Transfer Learning⁵⁶

Grundidee: Wiederverwenden von bereits trainierten Netzwerken (oder Teilen davon)

Hintergrund: Wenn grundlegende Eigenschaften passen sollte neuronales Netz in tieferen Schichten allgemeine Merkmale gelernt haben, die sich auf andere Aufgaben übertragen lassen, sodass nur noch die höheren Schichten (meist Output-Layer) für die neue Aufgabe trainiert werden muss

⁵⁶Tensorflow-Tutorial: https://www.tensorflow.org/tutorials/images/transfer_learning

Beispiel: Organsegmentierung in medizinischen Bildern

Es gibt einige Informationen bei der Segmentierung von Organen, die unabhängig von der Art einer medizinischen Aufnahme sind: z.B. Lage der Organe im Körper, grobe Form, Orientierung, Größe in Relation zum Körper

⇒ um Organe in PET-Bildern (seltener, als MRT- oder CT-Bilder) zu segmentieren, könnten Netzwerke auf großen MRT- oder CT-Datensätzen vorge trainiert werden.

Anschließend werden die trainierten Modelle geladen und mit einem kleineren Trainingsdatensatz aus PET-Bildern auf die Segmentierung von PET-Bildern umtrainiert (nur letzte Schicht).

(Anmerkung: bei diesem Beispiel handelt es sich um aktuell laufende Forschung, vorläufige Ergebnisse sehen vielversprechend aus ;-))

Continual Learning

Nachteil an den meisten Trainingsverfahren im maschinellen Lernen: Modelle sind nur auf das Lösen eines bestimmten Tasks und nur für bestimmte Outputs (z.B. im Trainingsset vorhandene Klassen) ausgelegt.

Problem: in der Praxis ist es häufiger der Fall, dass Modelle adaptiv sein müssen, d.h. sich an neue Gegebenheiten anpassen müssen. Fähigkeiten erweitern zu können ist auch ein wichtiger Baustein auf dem Weg zu stärkerer KI, die menschen ähnlicher wird.

Lösungsmöglichkeit: Continual Learning⁵⁷

⇒ Forschungszweig im Deep Learning, der sich mit Möglichkeiten befasst, Modelle Wissen erweitern zu lassen, ohne dass früher gelerntes aber wichtiges Wissen vergessen wird (“catastrophic forgetting”)

⁵⁷ Ph.D Thesis mit gutem Überblick über Methoden: R. Aljundi: “Continual Learning in Neural Networks”, 2019, <https://arxiv.org/abs/1910.02718>

Explainable Deep Learning

Nachteil an neuronalen Netzen: Es ist nicht leicht, ihre Ergebnisse zu interpretieren oder zu verstehen, wie ein Modell seine Entscheidung trifft.

Problem: in der Praxis (z.B. Medizin, sicherheitskritische Anwendungen im Auto,...) ist oft erwünscht, die Entscheidung eines Modells nachvollziehen zu können.

Lösungsmöglichkeit: Explainable Deep Learning

⇒ Forschungszweig des Deep Learnings, der sich damit befasst, Entscheidungen neuronaler Netze nachvollziehbarer zu machen

Beispiele für Explainable-DL-Ansätze^{58,59,60}

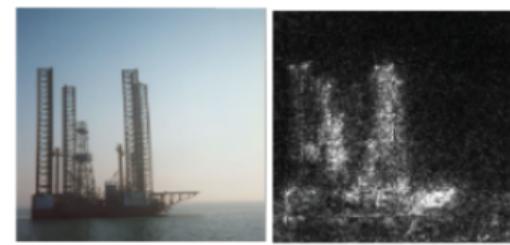
Prediction Difference Analysis (PDA)



Layer-wise Relevance Propagation (LRP)



SmoothGrad



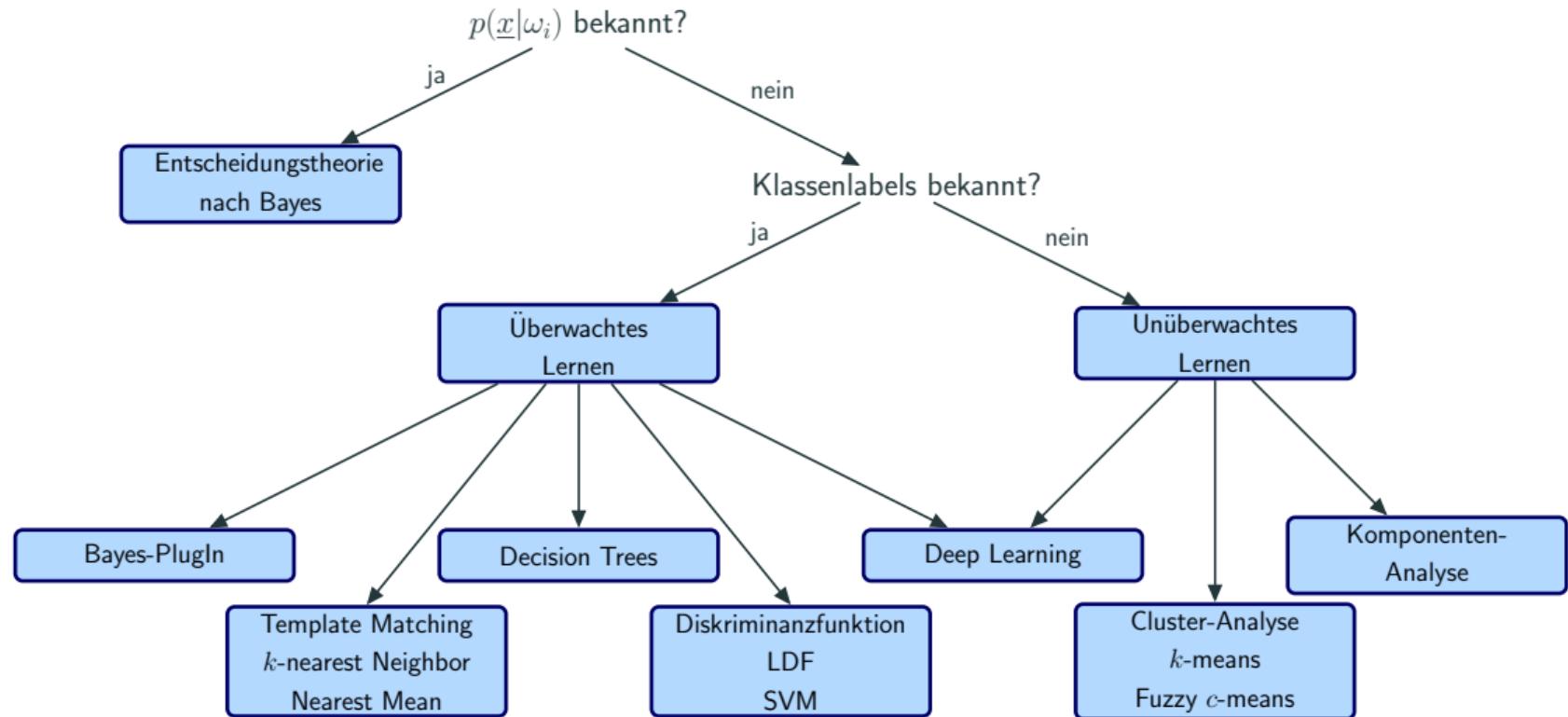
⁵⁸ Zintgraf et al.: "Visualizing deep neural network decisions: prediction difference analysis" (PDA)

⁵⁹ Binder et al.: "Layer-wise relevance propagation for deep neural network architectures" (LRP)

⁶⁰ Smilkov et al.: "SmoothGrad: removing noise by adding noise"

Zusammenfassung

Übersicht



Worauf muss ich achten, wenn ich mit Machine Learning arbeite?

Daten

Fragen, die man sich stellen sollte:

- Wie ist die *Qualität* meiner Daten?
- Sind meine Daten *repräsentativ*?
- Habe ich *genug* Daten?

Worauf muss ich achten, wenn ich mit Machine Learning arbeite?

Daten

Fragen, die man sich stellen sollte:

- Wie ist die *Qualität* meiner Daten?
- Sind meine Daten *repräsentativ*?
- Habe ich *genug* Daten?

Labels

Wichtige Regel: Jeder Klassifikator kann *nur* so gut sein, wie die *Qualität der Labels!*

Wenn Klassenlabels auf *subjektiver* Einschätzung von Experten basieren: *mehrere Fragen* und Ergebnis mitteln!

Worauf muss ich achten, wenn ich mit Machine Learning arbeite?

Daten

Fragen, die man sich stellen sollte:

- Wie ist die *Qualität* meiner Daten?
- Sind meine Daten *repräsentativ*?
- Habe ich *genug* Daten?

Labels

Wichtige Regel: Jeder Klassifikator kann *nur* so gut sein, wie die *Qualität der Labels!*

Wenn Klassenlabels auf *subjektiver* Einschätzung von Experten basieren: *mehrere Fragen* und Ergebnis mitteln!

ML-Modell

Ein passendes Modell auswählen und dabei folgende Dinge bedenken:

- das Datenset - Größe, Verfügbarkeit von Labels,...
- die ML-Aufgabe - Klassifikation oder Regression? Binär oder Multi-class?
- die Anwendung - könnten z.B. nach dem Training noch Klassen hinzukommen?
- die verfügbaren Rechenkapazitäten - ist Geschwindigkeit/Speicher kritisch?

7 Schritte auf dem Weg zu erfolgreichem Machine Learning

1. Das vorliegende Problem verstehen und zugehörige Aufgabenstellung definieren

7 Schritte auf dem Weg zu erfolgreichem Machine Learning

1. Das vorliegende Problem verstehen und zugehörige Aufgabenstellung definieren
2. Literaturrecherche durchführen und ML-Ansatz wählen, der für *diese* Aufgabe passt

7 Schritte auf dem Weg zu erfolgreichem Machine Learning

1. Das vorliegende Problem verstehen und zugehörige Aufgabenstellung definieren
2. Literaturrecherche durchführen und ML-Ansatz wählen, der für *diese* Aufgabe passt
3. Daten sorgfältig vorbereiten (z.B. Labeling)

7 Schritte auf dem Weg zu erfolgreichem Machine Learning

1. Das vorliegende Problem verstehen und zugehörige Aufgabenstellung definieren
2. Literaturrecherche durchführen und ML-Ansatz wählen, der für *diese* Aufgabe passt
3. Daten sorgfältig vorbereiten (z.B. Labeling)
4. Für Merkmals-basierte Ansätze: *sinnvolle* Merkmale extrahieren

7 Schritte auf dem Weg zu erfolgreichem Machine Learning

1. Das vorliegende Problem verstehen und zugehörige Aufgabenstellung definieren
2. Literaturrecherche durchführen und ML-Ansatz wählen, der für *diese* Aufgabe passt
3. Daten sorgfältig vorbereiten (z.B. Labeling)
4. Für Merkmals-basierte Ansätze: *sinnvolle* Merkmale extrahieren
5. Modell trainieren und optimieren (Hyperparameter beachten)

7 Schritte auf dem Weg zu erfolgreichem Machine Learning

1. Das vorliegende Problem verstehen und zugehörige Aufgabenstellung definieren
2. Literaturrecherche durchführen und ML-Ansatz wählen, der für *diese* Aufgabe passt
3. Daten sorgfältig vorbereiten (z.B. Labeling)
4. Für Merkmals-basierte Ansätze: *sinnvolle* Merkmale extrahieren
5. Modell trainieren und optimieren (Hyperparameter beachten)
6. Trainiertes Modell anhand neuer Daten evaluieren (Cross-Validation)

7 Schritte auf dem Weg zu erfolgreichem Machine Learning

1. Das vorliegende Problem verstehen und zugehörige Aufgabenstellung definieren
2. Literaturrecherche durchführen und ML-Ansatz wählen, der für *diese* Aufgabe passt
3. Daten sorgfältig vorbereiten (z.B. Labeling)
4. Für Merkmals-basierte Ansätze: *sinnvolle* Merkmale extrahieren
5. Modell trainieren und optimieren (Hyperparameter beachten)
6. Trainiertes Modell anhand neuer Daten evaluieren (Cross-Validation)
7. Deployment: erst dann, wenn das Modell sicher robust genug ist!