Digital Applications & Data Management

WS25/26

Modul 9

Dr. Jens Kohl

Roadmap Vorlesung

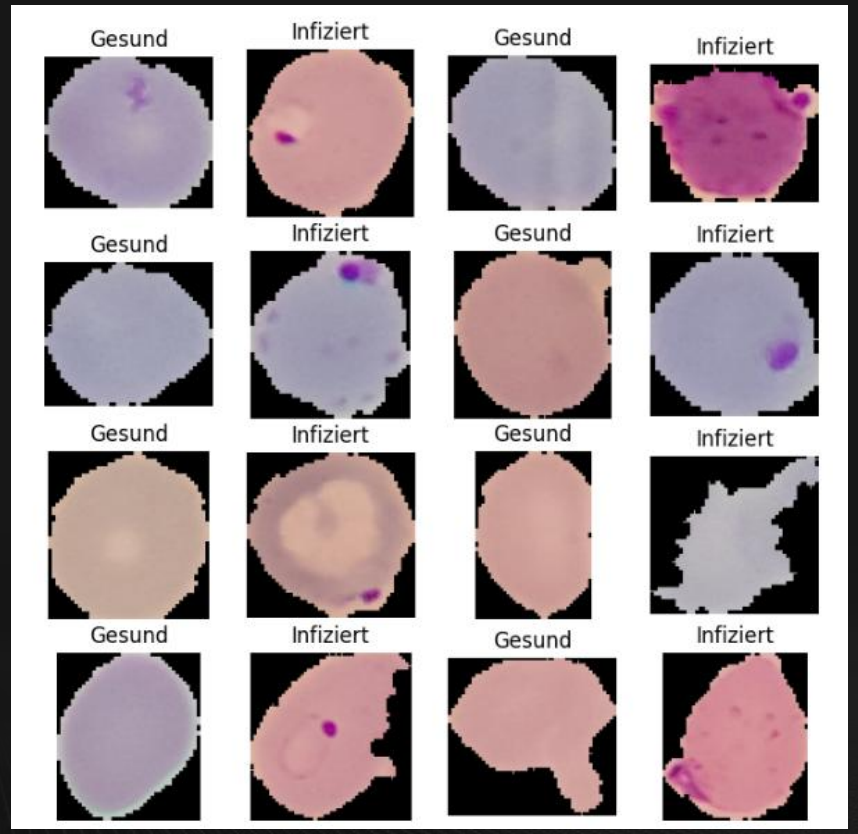
1. Grundlagen Künstlicher Intelligenz, Machine Learning und Daten
2. Grundlagen Data Science
3. Angewandte Data Science (Teil 1)
4. Angewandte Data Science (Teil 2)
5. Data Science Use Case
6. Grundlagen unüberwachtes Lernen
7. Grundlagen überwachtes Lernen (tabellarische Daten)
8. Machine Learning Use Case
9. Grundlagen überwachtes Lernen (Bilddaten)
10. Transfer Learning Bilddaten und Fallbeispiel
11. Grundlagen Generative AI
12. Prompt Engineering, Agenten
13. Ausblick, Wiederholung, Fragestunde
14. Fragestunde

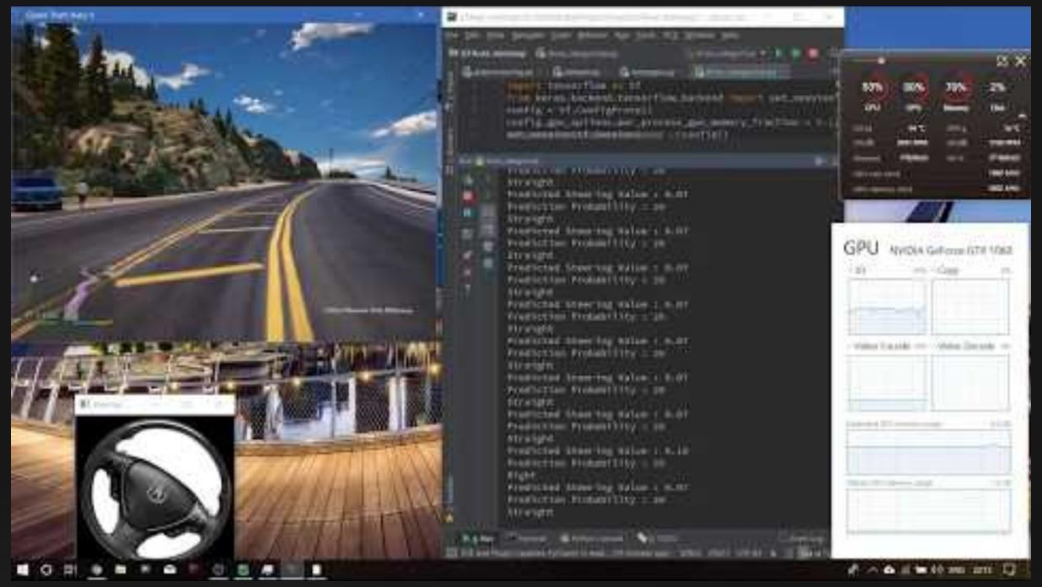
9. ÜBERWACHTES (SUPERVISED) LERNEN:

BILDDATEN

Was machen wir heute?

Motivation



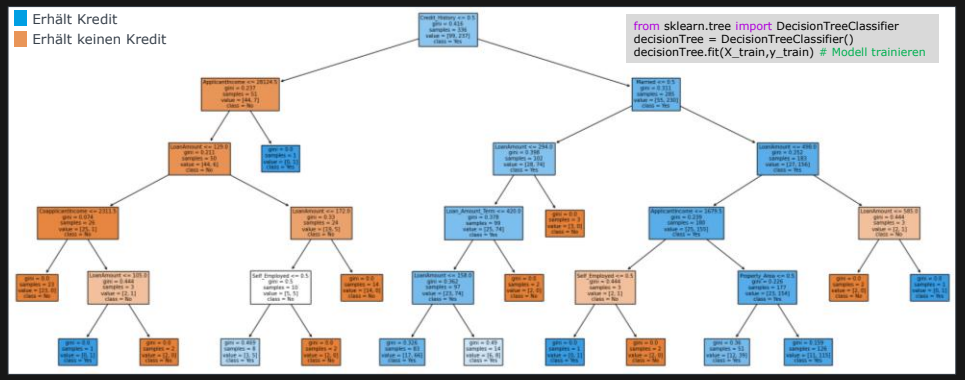


9.1 Neuronale Netze

Neuronale Netze

Motivation

- Wir haben in der letzten Vorlesung einige supervised machine learning verfahren kennengelernt:
  - Diese sind schnell einsetzbar und trainierbar.
  - die Ergebnisse sind MEIST leicht verständlich



- Aber:
  - Diese Verfahren haben Probleme mit (sehr) grossen Datenmengen und vielen Features.
  - Sie liefern schlechte Performance bei komplex(er)en Daten wie Bildern aufgrund einer „statischen“ Sicht.

Neuronale Netze können mit sehr großen Datenmengen besser umgehen als bisher betrachteten Verfahren. Dafür sind sie komplexer, benötigen länger zum Trainieren und Erklärbarkeit Ergebnisse ist nicht ganz einfach.

Neuronale Netze

Übersicht

ZIEL: Modell um für beliebige Eingaben x eine Zielgrösse y möglichst genau zu bestimmen.

Ermöglicht:

- Regression: Vorhersage kontinuierlicher Wert für y (bspw. natürliche oder reelle Zahl)
- Klassifikation: Vorhersage diskreter Wert für y (bspw. wahr/falsch, bestimmte Tierart, …)
- Zusätzlich: Erkennen von Bildinhalten (CNN) und zeitabhängigen Daten (RNN)

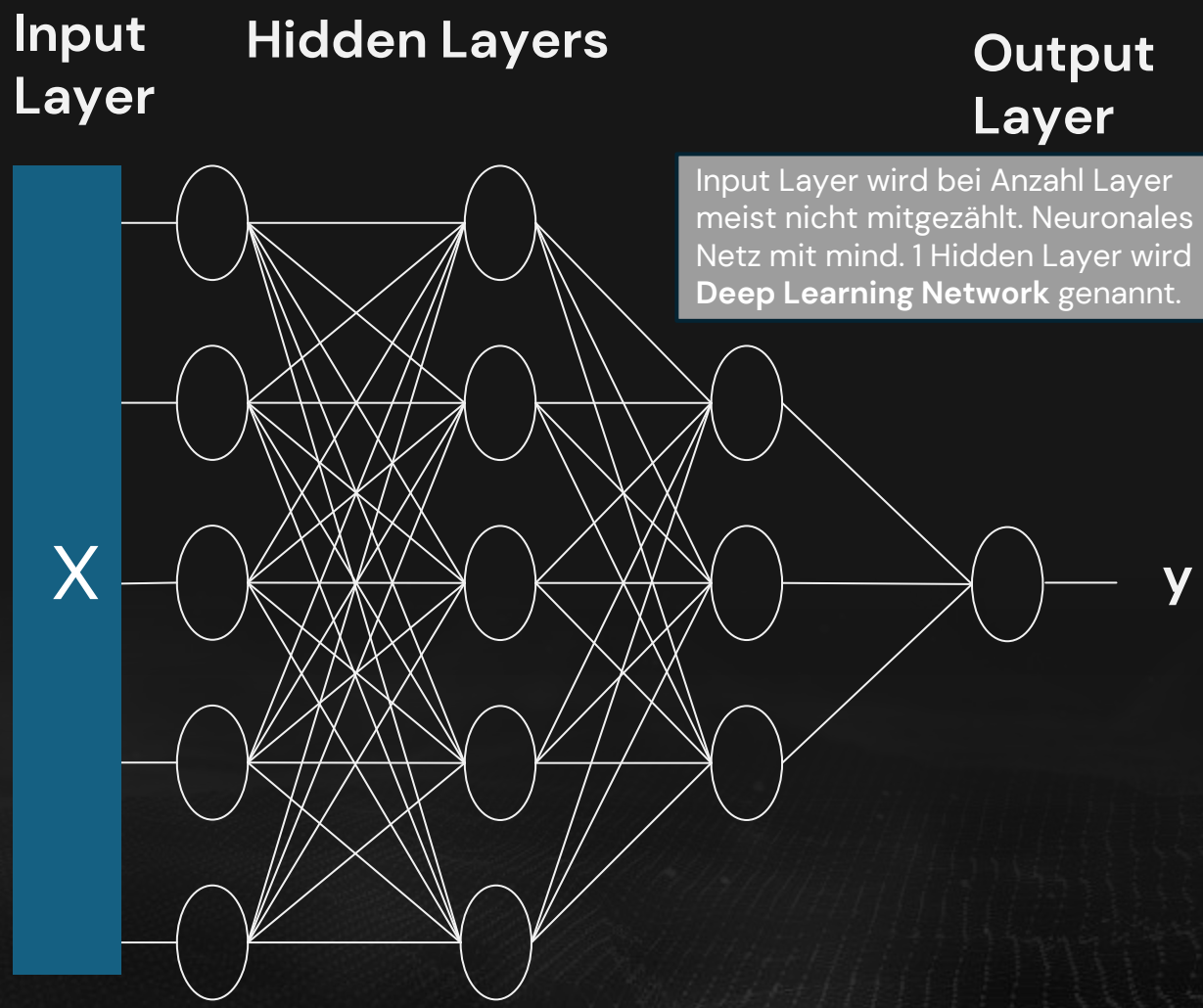
Welche Varianten neuronaler Netze schauen wir uns an?

- Convolutional neural networks (CNN): durch Faltungsoperatoren werden Bildmuster gelernt
- Transfer learning: vortrainiertes, gutes Modell wird auf speziellen use case angepasst (spart ressourcen)
- Rekurrente neuronale Netze (RNN): lernen sequentieller daten (bspw. gesprochener satz, film) → spätere Vorlesung

Neuronale Netze

Graphische Darstellung

- Input Layer: Erhält Eingaben X und reicht diese an Hidden Layer weiter. Keine Datenumwandlung/-bearbeitung! Input Layer skaliert mit/ bildet die Eingabegröße ab (bspw. ein Input je Anzahl Pixel eines Bildes, ….).
- Hidden Layer: hier erfolgt Lernen innerhalb einzelnen Nodes sowie in Kombination der nachgelagerten Layers. Durch Verwenden mind. eines Hidden Layers können beliebig komplexe Funktionen abgebildet/ angenähert werden[1].
- Connections: Verbindungen zwischen einzelnen Elementen der Layer. Ist ein Layer mit allen Elementen des vorherigen und nachherigen Layer verbunden, spricht man von fully connected layer.
- Output Layer: letzter Layer. Generiert das finale Ergebnis y. Anzahl entspricht dem zu prädizierenden Ergebnis (bspw. 1 für Erkennen Hund/ Katze oder 10 bei Ziffern).



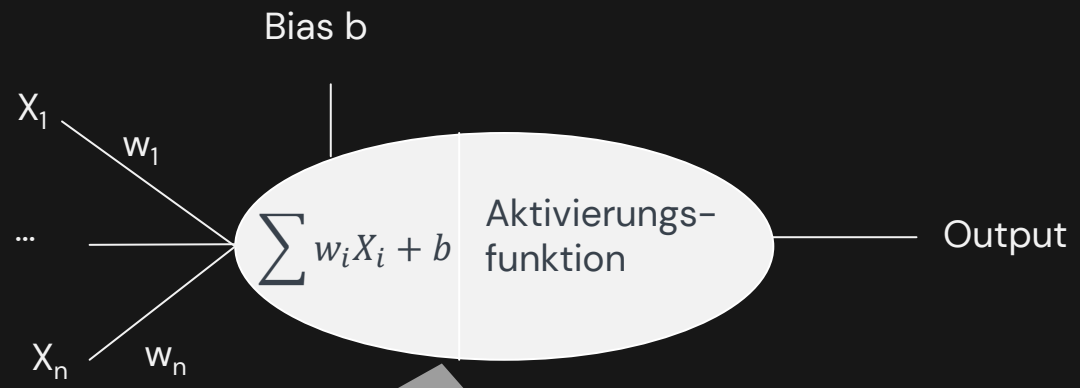
Input Layer wird bei Anzahl Layer meist nicht mitgezählt. Neuronales Netz mit mind. 1 Hidden Layer wird **Deep Learning Network** genannt.

Eingaben X für Input Layer werden durch Nodes der Hidden Layers sowie Output Layer so umgewandelt, daß Zielgröße y möglichst genau bestimmt oder angenähert wird

Quellen:
1 Lippmann: "An introduction to computing with neural nets", 1987 sowie Cybenko: "Approximation by superposition of a sigmoidal function", 1989.

Neuronale Netze

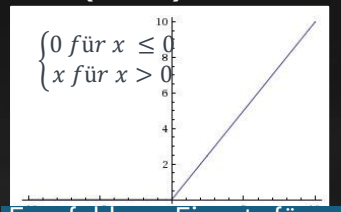
Detaillierung Node/ perceptron.



Der Name neuronales Netz stammt von der Ähnlichkeit eines Perceptrons zu einem Neuron im Gehirn. Beide funktionieren aber unterschiedlich.

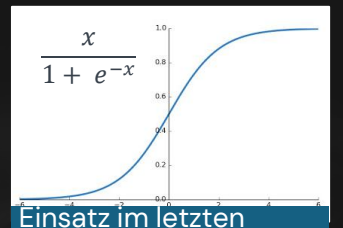
- **Weights w_i:** jeder Input hat ein Gewicht, das angibt, wie stark der Einfluß des Inputs auf den Node ist.
- **Bias b_i:** stellt Aktivierung Node auch bei Inputs = 0 sicher. (vgl. Geradengleichung mx + t für x=0).
- **Aktivierungsfunktion[1]:** verarbeitet die Gewichte w, Eingaben X_i und Bias b. Ermöglicht Abbilden komplexer Beziehungen zwischen Input und Output.
  Meist setzen wir eine der folgenden Funktionen ein:

Rectified Linear Unit (ReLU)[2]



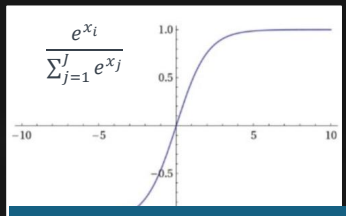
Empfehlung: Einsatz für alle Schichten (bis auf letzten Layer)

Sigmoid



Einsatz im letzten Layer f. binäre Klassifikation

SoftMax



Einsatz im letzten Layer für Klassifik. vieler Zielklassen

Weights und Bias sind die Parameter, die im Training gelernt werden, um die Zielgröße/n anzunähern.

1 Funktion sollte:
- Kont. Differenzierbar/ keine Lücken haben, da sonst Gradient Descent nicht funktioniert
- begrenzte Wertemenge (meist zwischen 0 und 1) haben, damit Gradient Descent stabiler ist
- nicht linear sein, um komplexere Funktionen als Geraden abzubilden. Aber: ReLU funktioniert sehr gut!

Quellen: 2 Nair, V. & Hinton, G.: "Rectified linear units improve restricted Boltzmann machines". ICML, 2010.

9.2 CONVOLUTIONAL NEURAL NETWORKS (CNN)

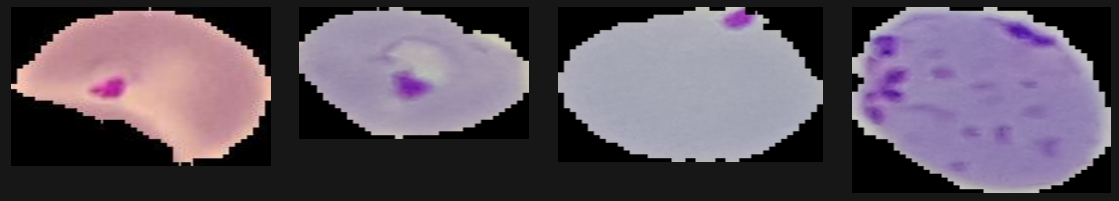
CNN

Motivation

Bei der Klassifikation von Bildern haben wir zwei grundlegende Probleme:

- Bilder können sich sehr unterscheiden
  - Bildgrösse
  - Ausrichtung: hoch- vs. Querformat
  - Farbig/ schwarz-weiss,
  - Anordnung im Bild
  - .....

- Bilder haben enorme Grösse verglichen mit Texten und Zahlen: 48 megapixel = 48 Mio. punkte!!



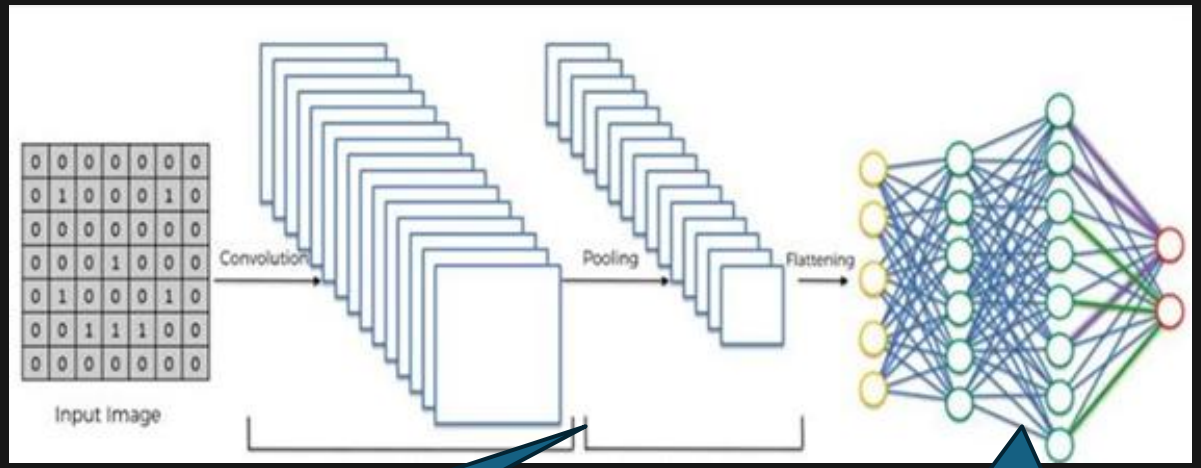
Ähnliche/ gleiche Objekte, aber verschiedene Darstellungen.
Die bisher vorgestellten Verfahren würden die Positionen einzelner Pixel lernen, was bei leichten Änderungen Bild nicht mehr funktionieren würde.

Quellen:
Bilder Zellen: National Library of Medicine: Malaria Dataset, Link
Bilder Labradore: Wikipedia-Artikel

CNN

Vorgehensweise

- Erfassen der wesentlichen Bildinformationen durch
  - Reduktion/ Verdichtung Bildinhalte
  - Erkennen der wesentlichen Strukturen im Bild
- Klassifizieren der erkannten wesentlichen Strukturen



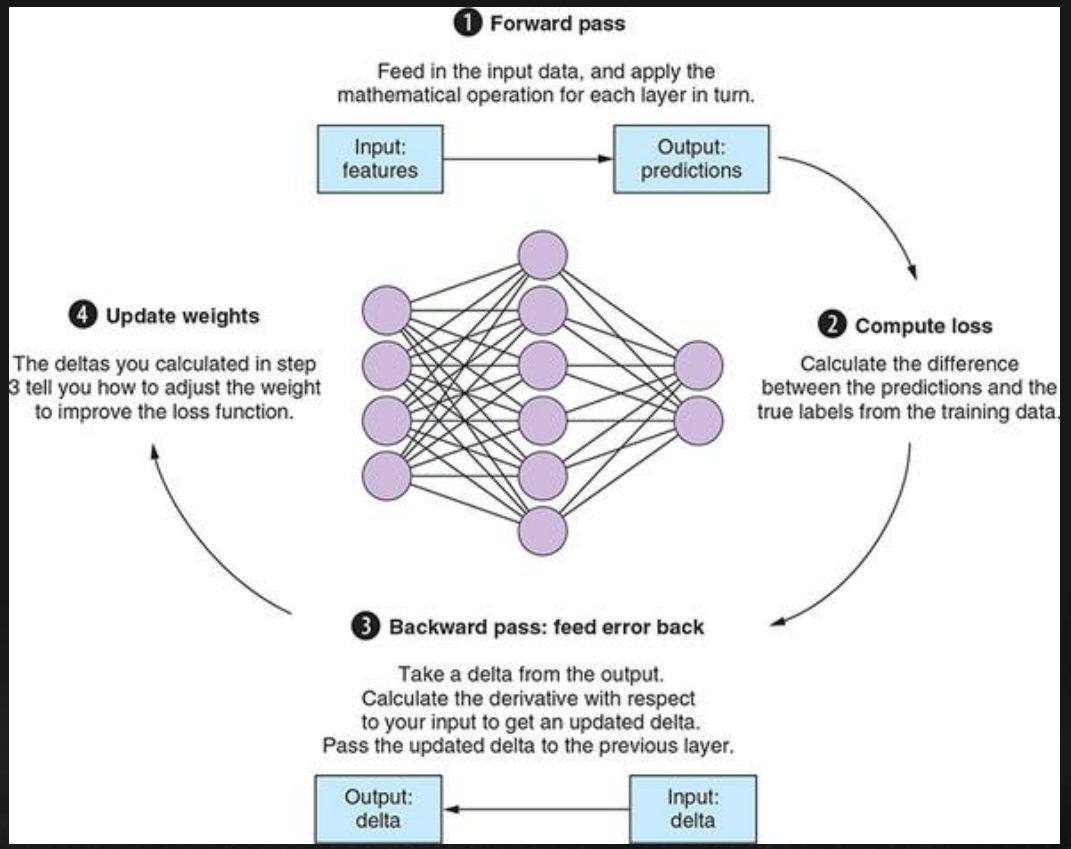
1. **LERNEN FEATURES/ CHARAKTERISTIKA** (BSPW. WAS IST WICHTIG IM BILD UND WAS IST RAUSCHEN? WAS MACHT EINEN HUND AUS? WAS IST UNTERSCHIED ZUR KATZE?)

2. **LERNEN KLASSIFIKATION** DURCH EINSPEISEN MUSTER IN NEURAL NETWORK UND KLASSIFIKATION (ANALOG BISHERIGE METHODEN SUPERVISED LEARNING)

CNN

Trainieren

1. Forward pass: Eingabedaten (z. B. Bild) werden durch Netzwerk geleitet. Jede Schicht verarbeitet Daten und wandelt sie Schritt für Schritt um.
   Letzte Schicht erzeugt Ausgabe, die bei einer Klassifizierungsaufgabe (z. B. Identifizierung von Objekten in Bildern) eine Klassenbezeichnung sein kann.
2. Verlustberechnung: Nach Vorwärtsdurchlauf berechnet das Netzwerk wie weit die Vorhersage aus 1) von der wahren Antwort unterscheidet. Ist
3. Rückwärtsdurchlauf (Backpropagation): der berechnete Verlust wird von der letzten Schicht durch das ganze Netz bis nach dem Eingabe-Layer geleitet.
4. Anpassen Gewichte: Die einzelnen Elemente der Layer werden angepaßt um für den nächsten Vorwärtsdurchlauf den Fehler zu reduzieren.



Durchführen Schritte 1-4 inkl. Update Modellparameter für Durchlauf des gesamten Trainingsset heißt Epoche, das Durchlaufen eines Teils des Trainingsset nennt man Batch. Anzahl Batches und Epochen ist erfahrungsabhängig inkl. Abbruchkriterien (ausreichende oder stagnierende Güte).

CNN

Live Demos (Browser)

- https://poloclub.github.io/cnn-explainer/
- http://cs231n.stanford.edu/2022/
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html
- https://playground.tensorflow.org

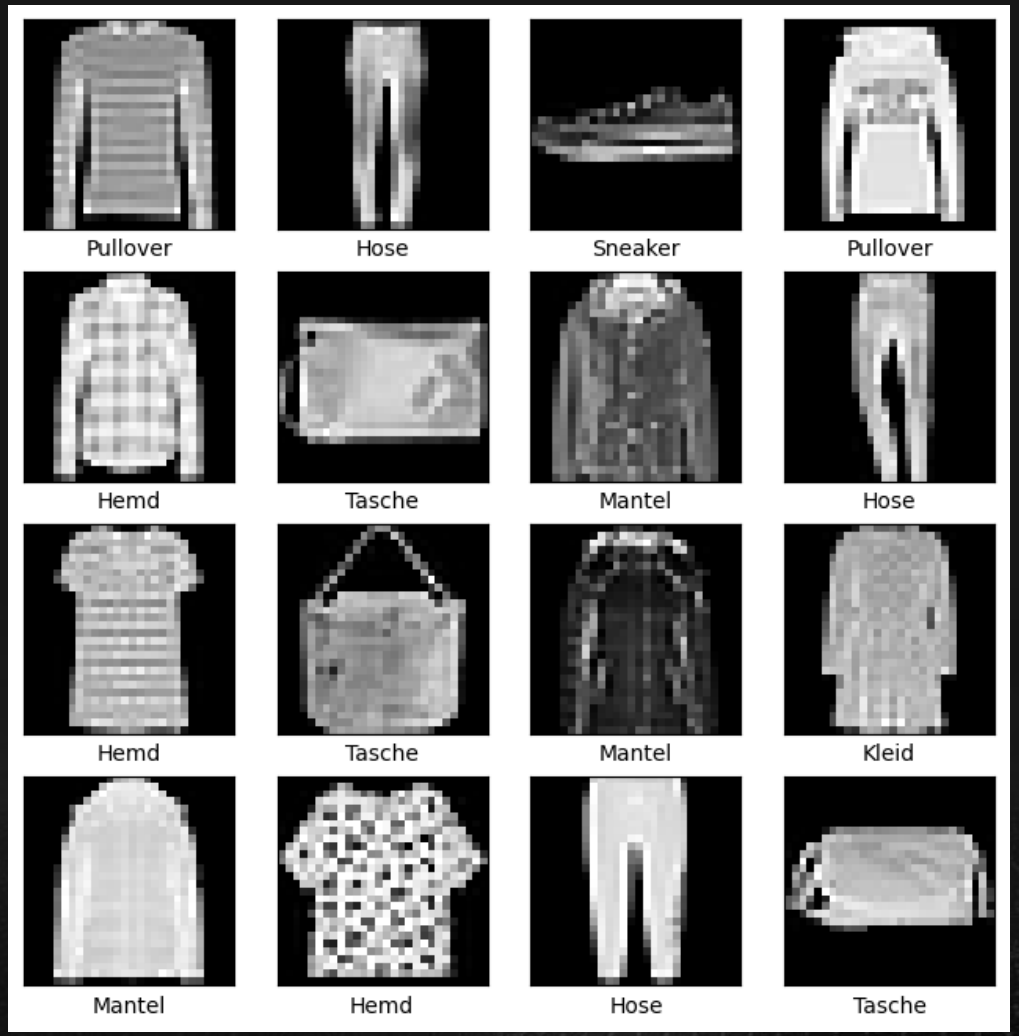
CNN:
Case Study

Case Study CNN

Übersicht

Fashion MNIST Datensatz von Zalando:

- Anzahl Bilder:
  - 60 000 Trainings-
  - 10 000 Testbilder
- Bilderformat: grau-schwarz 28x28 Pixel
- 10 Klassen von Modeartikeln
- Referenz/ Benchmark für neue Verfahren



Case Study CNN

Übersicht

In der heutigen Vorlesung machen wir:

- Import/ Laden der Daten
- Erstellen, trainieren und Optimieren eines Modells
- Testen des Modells

Case Study CNN

Vorgehensweise

Ask an interesting question

„Klassifizierung eines Modeartikels in verschiedene Klassen anhand eines Fotos“

Get the Data

Explore the Data

Model the Data

Communicate/Visualize the Results

Case Study CNN

Vorgehensweise

Ask an interesting question

Get the Data

Daten organisieren

Explore the Data

Model the Data

Communicate/Visualize the Results

Case Study CNN

Schritt 2: Get the data

- Für das Laden und Auswerten der Daten gibt es in Python viele Programmbibliotheken (Libraries), die uns die Detailarbeit der Programmierung abnehmen.
- Am Beginn jedes Cases laden wir diese Libraries. In der Programmierumgebung in Python gilt:
  - Lila markierter Text sind Standard-Befehle der Programmiersprache Python
  - Weiss markierter Text sind Variablen oder Libraries
  - Grün markierter Text sind Kommentare und werden mit einer Raute eingeleitet.

Case Study CNN

Schritt 2: Get the data

- Öffnen https://colab.research.google.com/
- File → New Notebook
- Benennen der Datei, z.B. Fashion MNIST
- + Code → Eingabe Code, + Text → Eingabe erklärender Text (was macht der untenstehende Code) zum Nachvollziehen

Case Study CNN

Schritt 2: Get the Data

Laden der wichtigsten Libraries:

```
import pandas as pd

import numpy as np

from google.colab import files
```

Case Study CNN

Schritt 2: Get the Data

```
# Fashion_MNIST ist ein beispielhafter Datensatz der TensorFlow-Library, d.h. wir können den direkt einlesen
from tensorflow.keras.datasets import fashion_mnist
# wir teilen den Datensatz auf in Trainings- und Testdaten inkl. Label für Trainings- und Testdaten
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

Case Study CNN

Vorgehensweise

Ask an interesting question

Get the Data

Explore the Data

Daten ansehen: sehen wir erste Eigenarten? Muster? Probleme mit Datenqualität bzw. Bildern?

Model the Data

Communicate/Visualize the Results

Case Study CNN

Schritt 3: Explore the data

```
# Statistiken zum Datensatz ausgeben
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

Zeile 1: Menge Trainingsbilder: 60.000 Bilder mit jeweils 28 Pixel breit und 28 Pixel hoch
Zeile 2: Labels für die Trainingsmenge: Spalte mit 60. 0000 Werten
Zeile 3: Menge Testbilder: 10.000 Bilder mit jeweils 28 Pixel breit und 28 Pixel hoch
Zeile 4: Labels für die Testmenge: Spalte mit 10.0000 Werten

Case Study CNN

Schritt 3: Explore the data - Umbauen Datenvektors für Machine Learning

```
X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)
```

Update Datensatz: wir fügen 1 Dimension am Ende hinzu
Trainingsmenge: 60 000 Bilder mit jeweils 28 Pixel breit, 28 Pixel hoch und 1 Kanal
Testmenge: 10 000 Bilder mit jeweils 28 Pixel breit, 28 Pixel hoch und 1 Kanal (Farbe)

```
# Statistiken zum Datensatz ausgeben
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28, 1)
(60000,)
(10000, 28, 28, 1)
(10000,)
```

- Zeile 1: X_training als Matrix mit Trainingsmenge von 60 000 Bilder mit jeweils 28x28 Pixel in 1 Farbe.
- Zeile 2: y_train als Vektor mit 60 000 Einträgen Label (welches Kleidungsstück)
- Zeile 3: X_test als Testmenge mit 10 000 Bildern
- Zeile 4: y_test als dazugehöriger Vektor mit Label

Wieso wird der Datensatz so umgebaut? CNN-Lernverfahren möchten Farbkanäle eines Bildes separat haben, damit sie einzelne Muster in den einzelnen Farbkanälen erkennen können.

Case Study CNN

Schritt 3: Explore the data – Normalisieren Daten

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Der Wert eines Pixels kann maximal 256 sein das Zählen startet bei 0!).
Teilen durch 255 sorgt dafür daß alle Werte zwischen 0 und 1 liegen.
Mit diesen Werten kann der Computer schneller rechnen, das Training ist somit schneller!

Case Study CNN

Schritt 3: Explore the data – Anzeigen zufälliger Bilder

```
[ ]  import matplotlib.pyplot as plt

     number_cols = 4 # Anzahl der Spalten für das Bild
     number_rows = 4 # Anzahl der Reihen für das Bild
     run_index = 0 # durchlaufende Index für einzelne Bilder, startet mit 0 und geht bis 4*4 = 16

     plt.figure(1 , figsize = (8 , 8))

     for i in range(number_cols * number_rows):
       run_index += 1 #erhöhe den durchlaufenden Index
       # definiere ein Teilbild
       cur_plot = plt.subplot(number_cols, number_rows, run_index)
       plt.xticks([]) # keine Beschriftung x-Achse
       plt.yticks([]) # keine Beschriftung y-Achse
       plt.grid(False) # kein Grid/ Rahmen zeichnen

       pic_index = np.random.randint(0, 60000) # wähle ein zufälliges Bild
       # lade und stelle ein Bild dar mit dem gewählten zufälligen Index
       plt.imshow(X_train[pic_index], cmap=plt.get_cmap('gray'))
       # schreib den Wert des Labels für das Bild als Wert unter das Bild
       plt.xlabel(class_names[y_train[pic_index]])
     plt.show()
```

1. Definieren Plots mit 16 Subbildern:
2. Plotten zufälliger Bilder

Case Study CNN

Schritt 3: Explore the data –Definieren eigener Namen für die Labels

```
# definiere eine Liste mit den verschiedenen Zielklassen
class_names = ['T-shirt/ Top',
               'Hose',
               'Pullover',
               'Kleid',
               'Mantel',
               'Sandale',
               'Hemd',
               'Sneaker',
               'Tasche',
               'Stiefel']
```

Case Study CNN

Vorgehensweise

Ask an interesting question

Get the Data

Explore the Data

Model the Data

Definieren, Trainieren, Optimieren und Testen Modell

Communicate/Visualize the Results

Case Study CNN

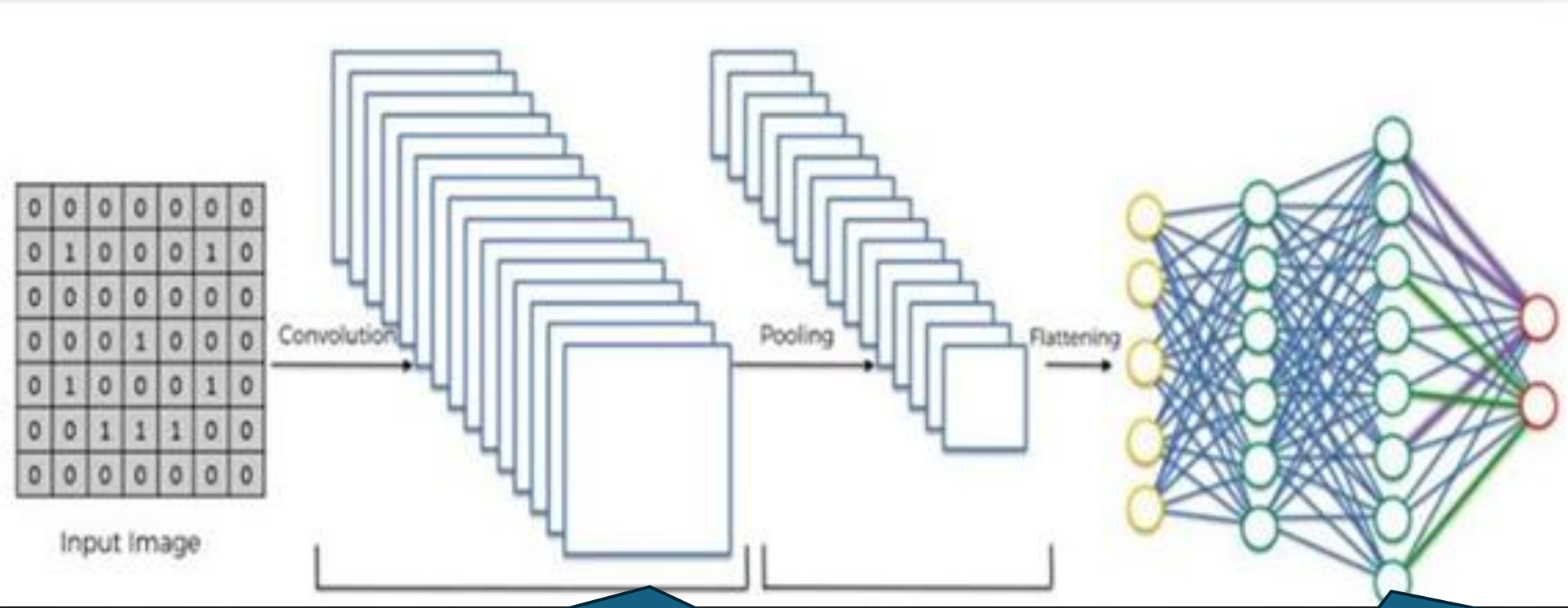
Schritt 4: Model the data - Übersicht

Vorgehen:

1. Definieren des Modells und Modellelemente (Faltungsoperatoren, Pooling, Dropout, …).
   Modell besteht zuerst aus mehreren Schichten für die Analyse der Bildinhalte.
   Diese Schichten erfassen wesentliche Bildinhalte bei gleichzeitiger Reduktion Datenmenge.
   An letzte Schicht der Bildanalyse wird dann ein „normale“ neuronale Schicht angehängt.
   Aufgabe dieser Schichten ist das Lernen der Einteilung in die vorgegebenen Zielklassen („Klassifizierung“).
2. Definieren Hyperparameter (Trainingsdauer, Batch_size, Optimierungsverfahren, …)
3. Modell kompilieren
4. Modell trainieren
5. Modell optimieren: Schritt 1-4 bis Metriken gut genug sind.

Case Study CNN

Schritt 4: Model the data - Übersicht



1. Lernen Features/ Charakteristika (bspw. was IST WICHTIG IM BILD und was ist rauschen? WAS macht einen Hund aus? Was ist Unterschied zur Katze?)

2. Lernen Klassifikation durch Einspeisen Muster in Neural Network und Klassifikation (analog bisherige Methoden Supervised Learning)

Case Study CNN

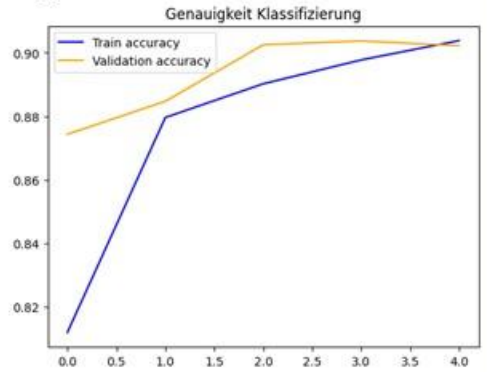
Schritt 4: Model the data – Übersicht

Trainieren Machine Learning Model (vereinfacht)

```
from tensorflow import keras   # Keras als Schnittstelle zu den "komplizierteren Umfängen"
from keras import layers
from keras import Model

# wie oben gezeigt, hat jedes Bild 28x28 Pixel.
# Da wir die Bilder nur in Graustufen haben, gibt es zusätzlich nur 1 Kanal.
INPUT_SHAPE = (28,28, 1)

num_target_classes = 10 # wir haben 10 Zielklassen

# wir bauen den Input Layer. Jeder Pixel ist ein Input für das CNN.
inp = tf.keras.layers.Input(shape=INPUT_SHAPE)

# wir bauen den ersten Bildverarbeitungs-Layer. Erst Faltung, dann Pooling, dann Dropout
conv1 = tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same')(inp)
pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2)) (conv1)
drop1 = tf.keras.layers.Dropout(0.5) (pool1)

# wir bauen den 2. Bildverarbeitungs-Layer. Erst Faltung, dann Pooling, dann Dropout
conv2 = tf.keras.layers.Conv2D(64,
                               kernel_size=(3, 3),
                               activation='relu',
                               padding='same') (drop1)

pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv2)
#drop2 = tf.keras.layers.Dropout(0.5) (pool2)  # vor Flatten nimmt man kein Pooling.

# nach dem 2. Bildverarbeitungs-Layer geben wir die Daten in ein "normales" CNN rein.
flat = tf.keras.layers.Flatten()(pool2)

# hier der erste Hidden Layer des CNN
hidden1 = tf.keras.layers.Dense(512, activation='relu')(flat)
drop3 = tf.keras.layers.Dropout(rate=0.5)(hidden1)

# hier der zweite Hidden Layer des CNN
hidden2 = tf.keras.layers.Dense(256, activation='relu')(drop3)
drop4 = tf.keras.layers.Dropout(rate=0.5)(hidden2)

# und hier gehts raus aus dem CNN.
# Wichtig ist die Wahl des Dense Layers und der Softmax-Aktivierungsfunktion.
# Diese brauchen wir, da wir mit dieser das Ergebnis klassifizieren können.
# Als Ergebnis erhalten wir für jede mögliche Zielklasse einen Wert.
# Out erhält dann den Wert, der die höchste Wahrscheinlichkeit hat.
out = tf.keras.layers.Dense(num_target_classes, activation='softmax')(drop4)

# jetzt speichern wir die ganze Struktur in ein Modell
model = tf.keras.Model(inputs=inp, outputs=out)
```

Messen Modellgüte per Metriken

Plotten Genauigkeit



Accuracy

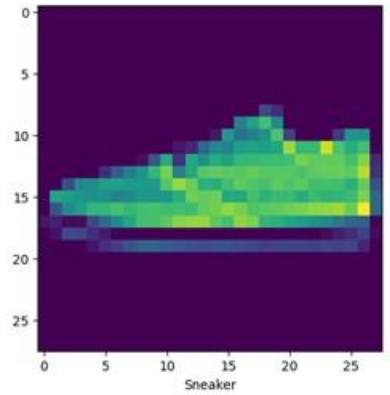
```
Test accuracy : 0.9021000266075134
```

Precision/ Recall

	precision	recall	f1-score	support
T-shirt/ Top	0.78	0.94	0.85	1000
Hose	0.99	0.98	0.99	1000
Pullover	0.80	0.90	0.85	1000
Kleid	0.93	0.89	0.91	1000
Mantel	0.83	0.84	0.83	1000
Sandale	0.99	0.98	0.98	1000
Hemd	0.82	0.59	0.68	1000
Sneaker	0.93	0.99	0.96	1000
Tasche	0.99	0.97	0.98	1000
Stiefel	0.99	0.94	0.96	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

Anwenden Modell

```
classify_pic(X_test[60], 60)
```




```
def classify_pic(samplePicture, picindex):
  samplePicture /= 255.  # Bild normalisieren

  sample_pic_aufbereitet = samplePicture.reshape (1, 28, 28, 1)
  predicted_model = model.predict(sample_pic_aufbereitet)

  temp_picture = samplePicture.reshape(28,28) # wir löschen den Schwarz-weiß Kanal fürs Anzeigen des Bildes

  plt.imshow(temp_picture)
  plt.xlabel(class_names[y_test[picindex]])

  plt.show()
  plt.bar(range(10), predicted_model[0])
  plt.grid(False)
  x_vals = range(0, len(class_names))
  plt.xticks(x_vals, class_names, rotation=45)
  plt.xticks(range(10))
  plt.yticks([])

  print("Klasse vorhergesagt: {}".format(class_names[np.argmax(predicted_model[0])]))
  print("Gesamte Wahrscheinlichkeiten:")
```

Case Study CNN

Definition Input layer


```
(60000, 28, 28, 1)
(60000,)
(10000, 28, 28, 1)
(10000,)
```

tf.keras.layers.Input():
Definition, wie der Input Layer aussehen soll

```
from tensorflow import keras    # Keras als Schnittstelle zu                plizierteren Umfängen"
from keras import layers
from keras import Model

# wie oben gezeigt, hat jedes Bild 28x28 Pixel.
# Da wir die Bilder nur in Graustaufen haben, gibt es zusät:    n nur 1 Kanal.
INPUT_SHAPE = (28,28, 1)
```

```
# wir bauen den Input Layer. Jeder Pixel ist ein Input für das CNN.
inp = tf.keras.layers.Input(shape=INPUT_SHAPE)
```

Der Input Layer muß individuell und genau auf die Größe und Struktur der Eingabebilder angepaßt werden.
Diese Struktur ist bekannt durch X_train.shape()

Case Study CNN

Definition Output Layer

```
class_names = ['T-shirt/ Top',
               'Hose',
               'Pullover',
               'Kleid',
               'Mantel',
               'Sandale',
               'Hemd',
               'Sneaker',
               'Tasche',
               'Stiefel']
```

tf.keras.layers.Input():
Definition, wie der Input Layer aussehen soll

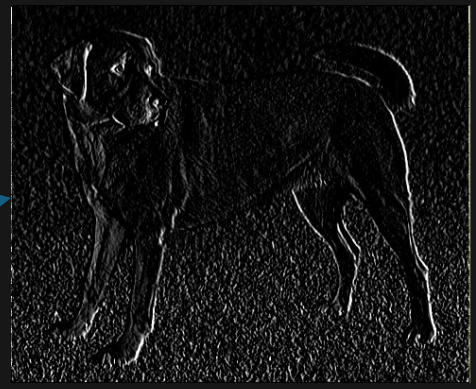
```
num_target_classes = 10 # wir haben 10 Zielklassen
```

Der Input Layer muß individuell und genau auf die Größe und Struktur der Eingabebilder angepaßt werden.
Diese Struktur ist bekannt durch X_train.shape()

Case Study CNN

Wie werden Strukturen eines Bildes erkannt?

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



„Entdecken“ vertikaler Linien

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

„Entdecken“ horizontaler Linien

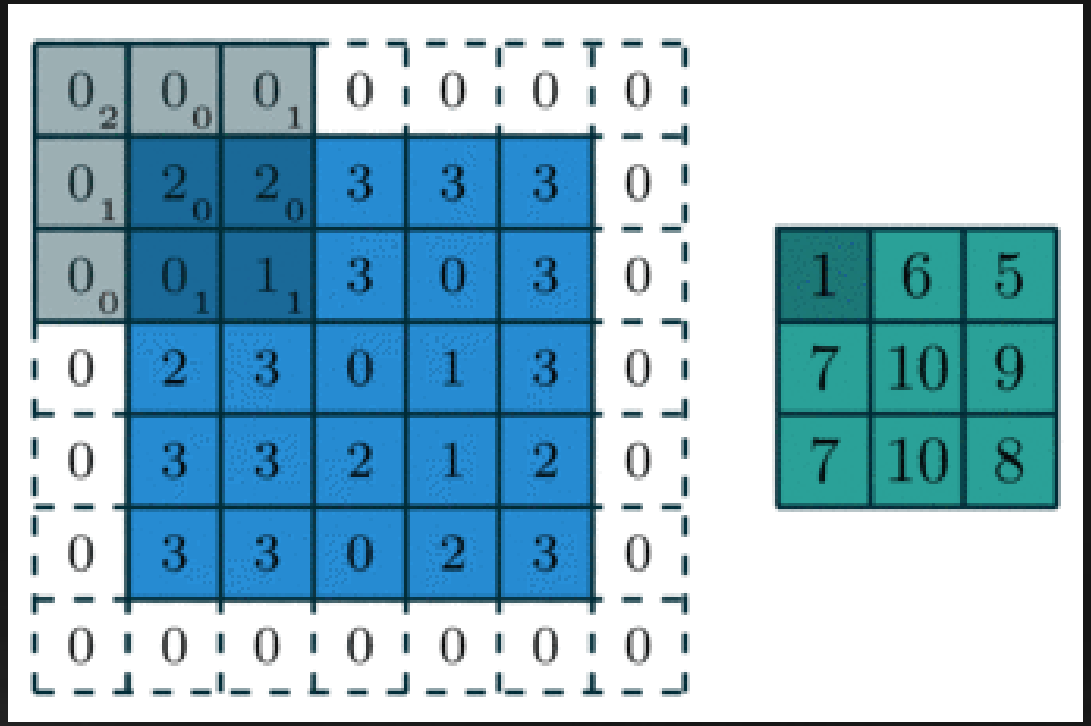
Die Faltung geschieht indem rollierend Zeile für Zeile jeweils mit einem Filter multipliziert wird

Quellen: Link, Link

Case Study CNN

Faltungs-Layer für das Erkennen von Strukturen

- Filtermatrix/ Kernel (hier 3x3) durchläuft analog einer Lupe Zeile für Zeile des Originalbildes.
- Stride bestimmt Schrittweise des Filters
- Padding: ist die Breite und/oder Höhe des Bildes nicht genau durch Höhe/ Breite des Filters teilbar, bleibt Rest über. Padding kann das auffüllen (im Bild ist padding = (1,1)).
- Das Ergebnis einer Faltung ist eine verkleinerte Matrix.



Ziel ist, durch mehrere kombinierte Faltungen nacheinander das Bild auf wichtigste Informationen zu reduzieren.

Case Study CNN

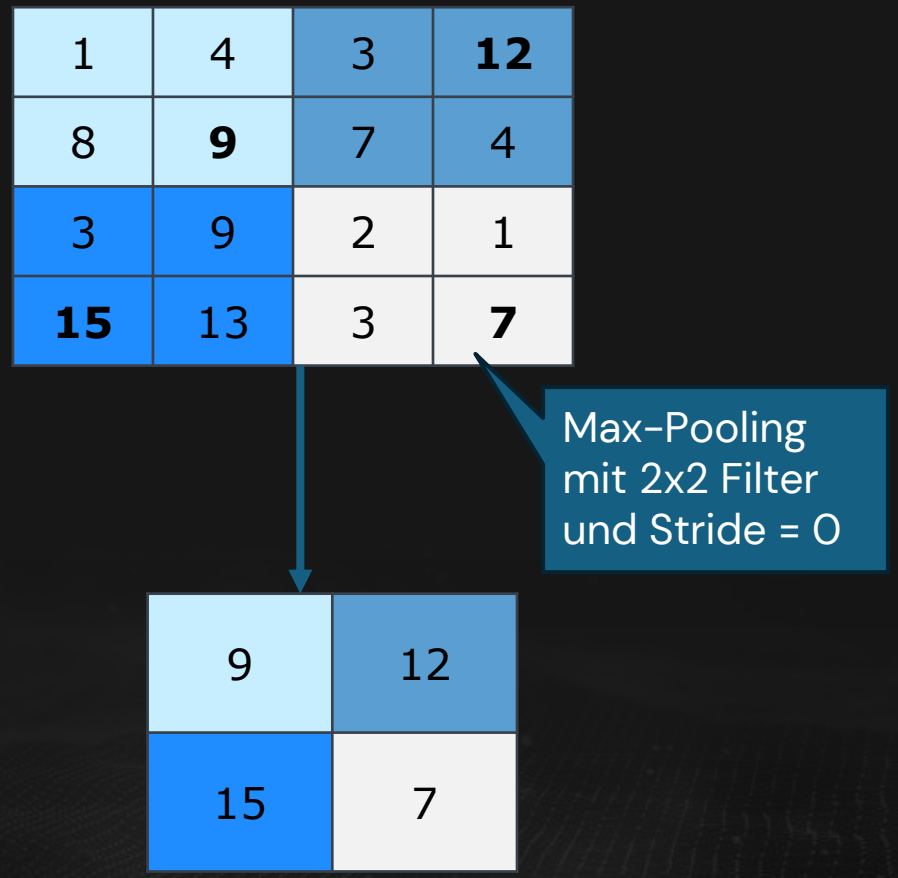
Faltungs-Layer für das Erkennen von Strukturen

- Conv2D: Schicht, die in CNNs zur Verarbeitung von Bilddaten verwendet wird.
- Conv2D verwendet Filter (oder Kernel), die kleine Fenster sind, die über das Eingabebild gleiten (oder "falten").
- Jeder Filter ist so konzipiert, dass er bestimmte Merkmale, wie Kanten, Texturen oder Muster, im Bild erkennt.
- Der Filter bewegt sich in Schritten über das Bild, die durch "Schritt" angegeben werden. An jeder Position deckt der Filter einen kleinen Bereich oder Fleck des Bildes ab.
- Für jede Position des Filters wird ein Punktprodukt zwischen den Filterwerten und den Pixelwerten, die er abdeckt, berechnet. Mit diesem Vorgang wird im Wesentlichen gemessen, wie sehr der Filter mit diesem Teil des Bildes übereinstimmt.
- Aktivierungsfunktion: Nach dem Punktprodukt wird eine Aktivierungsfunktion (z. B. ReLU - Rectified Linear Unit) angewendet, um Nichtlinearität einzuführen.

Case Study CNN

Pooling-Layer für das Reduzieren der Daten

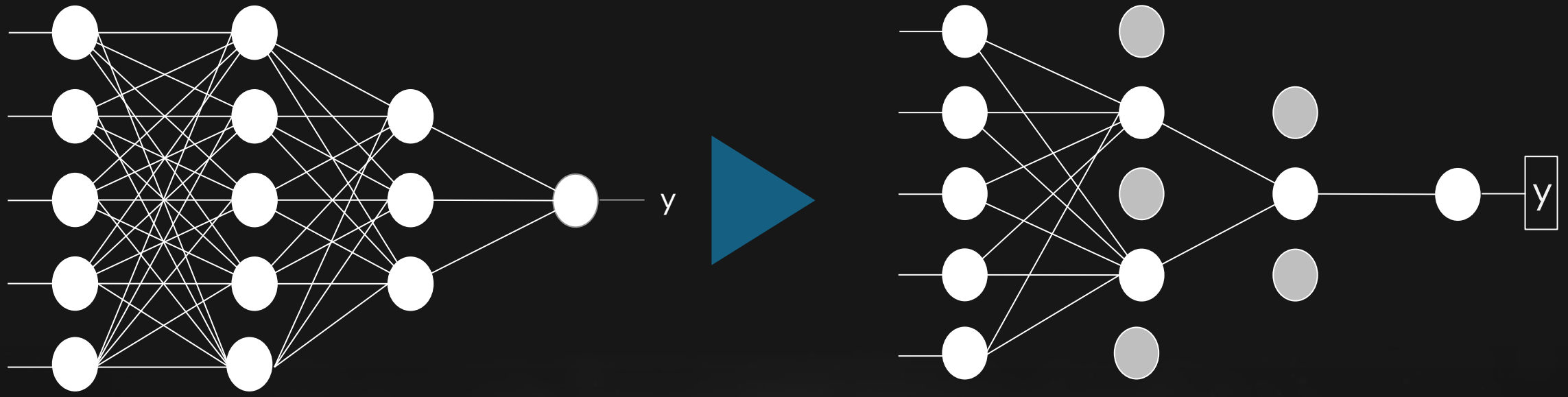
- Pooling reduziert Bilddatenmenge in Höhe und Breite und auch Anzahl Parameter und Berechnungen im verringert
- Max-Pooling: Es wird nur der maximale Wert aus dem vom Pooling-Filter abgedeckten Teil des Bildes genommen. Dies ist die am häufigsten verwendete Pooling-Methode.
- Pooling-VORGEHEN:
  - Pooling-Filter gleitet über Eingabebild
  - Für jede Position des Filters wird Maximal- (Max-Pooling) oder Durchschnittswert (Average-Pooling) aus vom Filter abgedeckten Bildbereich genommen.
- Stride: Genau wie bei Faltungsschichten gibt es bei Pooling-Schichten Parameter für Stride und Padding.
  - Schrittweite: bestimmt, wie viele Pixel der Filter über das Bild bewegt. Ein größerer Stride führt zu einem aggressiveren Down-Sampling.



Pooling ermöglicht Reduzierung Daten, Entfernen von Rauschen und verhindert Overfitting.

Case Study CNN

Dropout-Layer für Verhindern Overfitting



CNN neigen aufgrund Vielfalt an Parametern zu Overfitting

Dropout schaltet jeden Knoten mit Wahrscheinlichkeit 1-p als „passiv“. In jedem Trainingsdurchlauf werden andere Knoten passiv geschalten (hier: 50% Knoten).

Dropout reduziert Overfitting (und beschleunigt das Training).

Case Study CNN

Kombinieren der Layer

```
# wir bauen den ersten Bildverarbeitungs-Layer. Erst Faltung, dann Pooling, dann Droput
conv1 = tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same')(inp)
pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2)) (conv1)
drop1 = tf.keras.layers.Dropout(0.5) (pool1)
```

- tf.keras.layers.Conv2D() → Faltungsschicht für Erkennen von Kanten und Farbverläufen.
  Filters: Anzahl Ausgangsfilter/ Kanäle der Faltung
  Kernel_size: Gibt die Höhe und Breite des Faltungsfensters an. Immer ungerade Werte nehmen!
  Activation: Normalerweise wird 'relu' (Rectified Linear Unit) verwendet.
  Strides: Schrittweise Faltung
  Padding: 'valid' (keine Auffüllung) oder 'same' (Auffüllung, sodaß Ausgabe- gleich Eingabegröße)
- tf.keras.layers.MaxPooling2D() → Reduzierung Höhe/ Breite um Rechenlast und Overfitting zu verringern.
  Pool_size: Fenstergröße, die auf 1 Wert reduziert wird. Hier (2,2)
- tf.keras.layers.Dropout() → Regularisieren durch zufälliges Abschalten von Teilen des Netzes.
  Dropout_rate: Prozentzahl, wie viele Elemente des Netzes zufällig ausgeblendet werden in Prozent

Case Study CNN

Kombinieren der Layer

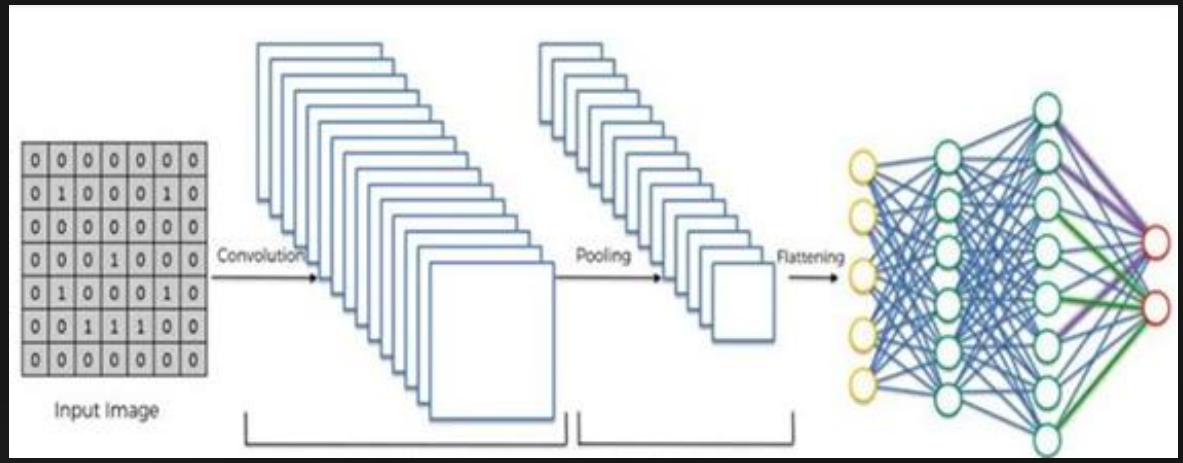
```
# wir bauen den 2. Bildverarbeitungs-Layer. Erst Faltung, dann Pooling, dann Droput
conv2 = tf.keras.layers.Conv2D(64,
                               kernel_size=(3, 3),
                               activation='relu',
                               padding='same') (drop1)

pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv2)
#drop2 = tf.keras.layers.Dropout(0.5) (pool2)  # vor Flatten nimmt man kein Pooling.
```

Gleiches Vorgehen wie bei erster Schicht. Aber: wir haben hier nur zwei Schichten, deshalb entfällt der Schitt Dropout (vor Flatten kein Pooling)

Case Study CNN

Kopplung CNN mit „normalem“ Neuronalen Netz

- Wir haben gesehen, wie Features in Bildern gelernt werden.
- Aber: für Klassifizieren Bilder fehlt noch ein Output-Vektor. Dies geschieht durch Verknüpfen der gelernten Features mit einem „einfachem“ neuronalen Netz.
- Mittels des Flatten-Operator werden die gelernten Features des Bildes als 1-dimensionaler Vektor abgespeichert.
- Weitere Vorgehensweise analog „einfachem“ neuronalen Netz oder den bisher gelernten Verfahren.



Case Study CNN

Flatten Operator

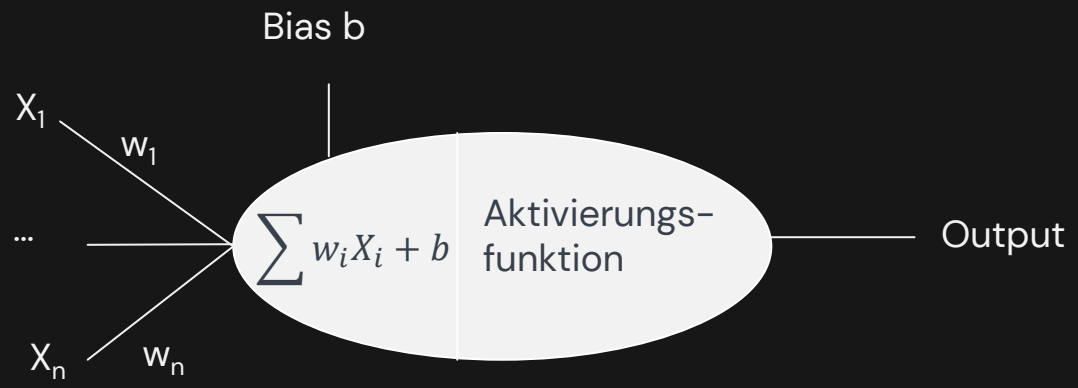
```
# nach dem 2. Bildverarbeitungs-Layer geben wir die Daten in ein "normales" CNN rein.
flat = tf.keras.layers.Flatten()(pool2)
```

Flatten wandelt die bisherigen Daten des Netzes in ein eindimensionales Array von Merkmalen. Sie können sich die Klassifizierung als einteilen in eine schablone oder schublade vorstellen und das Flatten als Vorbereiten der daten für das Einteilen.

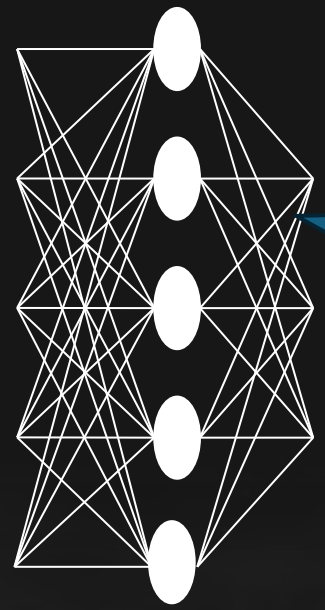
Case Study CNN

Dense Operator

Detailansicht eines Knoten



Mehrere Knoten ergeben einen Dense layer



Anzahl Dense-Layer und deren Knoten erfahrungsgetrieben. Achtung: Under- vs. Overfitting

- Weights w_i: jeder Input hat Gewicht, das angibt, wie stark Einfluß der Eingaben $X_1 \ldots X_n$ auf Node ist.
- Bias b_i: stellt Aktivierung Node auch bei Inputs = 0 sicher (vgl. Geradengleichung mx + t für x=0).
- Aktivierungsfunktion[1]: verarbeitet Gewichte w, Eingaben X_i und Bias b. Ermöglicht Abbilden komplexer Beziehungen zwischen Input und Output.
  Wir verwenden für Dense-Layer nur ReLU[2].

Weights & Bias Dense-layers sind die Parameter, die im Training gelernt werden, um die Zielgröße/n anzunähern

Quellen:
1 Aktivierungsfunktion sollte:
- Kont. Differenzierbar/ keine Lücken haben, da sonst Gradient Descent nicht funktioniert
- begrenzte Wertemenge (meist zwischen 0 und 1) haben, damit Gradient Descent stabiler ist

2 Nair, V. & Hinton, G.: "Rectified linear units improve restricted Boltzmann machines", ICML, 2010.

Case Study CNN

Dense Layer

```
# hier der erste Hidden Layer des CNN
hidden1 = tf.keras.layers.Dense(256, activation='relu')(flat)
drop3 = tf.keras.layers.Dropout(rate=0.5)(hidden1)

# hier der zweite Hidden Layer des CNN
hidden2 = tf.keras.layers.Dense(64, activation='relu')(drop3)
drop4 = tf.keras.layers.Dropout(rate=0.5)(hidden2)
```

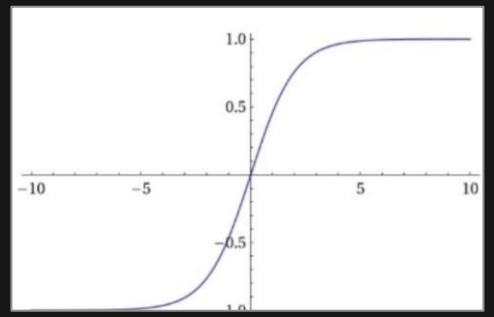
FLAT -> HIDDEN1 -> DROP3 -> HIDDEN2

Die Anzahl der Einheiten in der Dense-Schicht ist ein Parameter, den Sie einstellen können, und der optimale Wert hängt oft von verschiedenen Faktoren ab, z. B. von der Komplexität der Aufgabe, der Größe und Art Ihres Datensatzes und der Architektur Ihres neuronalen Netzwerks. Für die erste Denseschicht, insbesondere nach dem Flatten der Ausgabe von Faltungsschichten in einem CNN, kann die Anzahl der Einheiten durch die Größe der geflatteten Daten bestimmt werden.

Case Study CNN

Output layer

Sigmoid

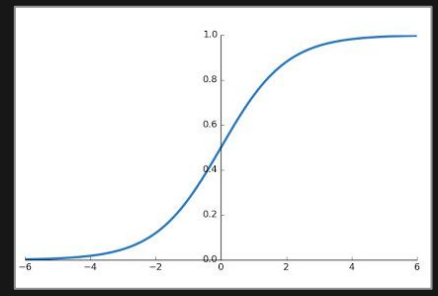


Einsatz bei binären Zielklassen

```
# und hier gehts raus aus dem CNN.
# Wichtig ist die Wahl des Dense Layers und der Softmax-Aktivierungsfunktion.
# Diese brauchen wir, da wir mit dieser das Ergebnis klassifizieren können.
# Als Ergebnis erhalten wir für jede mögliche Zielklasse einen Wert.
# Out erhält dann den Wert, der die höchste Wahrscheinlichkeit hat.
out = tf.keras.layers.Dense(num_target_classes, activation='softmax')(drop4)
```

Erster Parameter ist Anzahl Zielklassen, bei mehreren Zielklassen für activation softmax wählen, sonst sigmoid.

Softmax



Einsatz bei vielen Zielklassen

Softmax wird üblicherweise in der Ausgabeschicht eines neuronalen Netzmodells für Klassifizierungsaufgaben mit mehreren Klassen verwendet. Sie wandelt die Ausgabewerte in Wahrscheinlichkeiten um, die sich zu 1 summieren, was effektiv eine Wahrscheinlichkeitsverteilung über die Zielklassen ergibt.

Case Study CNN

Abschließen des Netzes

```
# jetzt speichern wir die ganze Struktur in ein Modell
model = tf.keras.Model(inputs=inp, outputs=out)
```

Case Study CNN

Anzeigen des ganzen Modells

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 64)        640

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 256)               803072

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 64)                16448

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 857738 (3.27 MB)
Trainable params: 857738 (3.27 MB)
Non-trainable params: 0 (0.00 Byte)
```

Diese Darstellung hilft Ihnen, zu sehen, ob die Definition des Modells erfolgreich war. Hier sollten dann alle Schichten dargestellt sein, die Sie definiert haben. Falls welche fehlen, dann haben Sie die vielleicht falsch verknüpft

Case Study CNN

Hyperparameter

- Nachdem wir das Modell gebaut haben, definieren wir jetzt die Hyperparameter:
  - Batch_size: wie viele Samples werden aus der Trainingsmenge genommen bevor die Gewichte und Bias des Modells angepaßt werden. Im unteren Fall heißt das, nach 256 Bildern passen wir die Gewichte an.
  - Epochen: wie viele Durchläufe durch das gesamte Datenset werden gemacht.
  - Learning_rate: Geschwindigkeit mit der das Modell lernt. Genauer gesagt stellt es die Schrittweite dar, die bei jeder Iteration gegangen wird, um das Minimum der Loss-Funktion zu erreichen.
  - Optimizer: Algorithmus der eingesetzt wird, um die Gewichte und Bias so zu reduzieren, daß die Loss-Funktion minimiert wird.
- Diese Parameter ermöglichen deutliche Verbesserungen des Ergebnisses und stehen auch miteinander in Verbindung (bspw. small batch size sollte mit niedriger Lernrate kombiniert werden).
- Nachdem wir das gemacht haben, starten wir das Trainieren des Models.

Case Study CNN

Hyperparameter: Deep dive – Anzahl Epochen

- Ein vollständiger Durchlauf der Trainingsmenge wird eine Epoche genannt.
- Faustregel: man nimmt eine hohe Epochenanzahl und trainiert bis zu dem Punkt, an dem gleichzeitig die Validation Accuracy abnimmt und die Training Accuracy zunimmt (Overfitting)

```
EPOCHS = 100
```

Case Study CNN

Hyperparameter: Deep dive – Anzahl Batches

- Batch size ist Teil einer Epoche und definiert, nach wie vielen Bildern/ Daten einer Epoche eine Anpassung der Modellparameter während des Trainings erfolgt, um das Modell zu verbessern.
- Trade-Off zwischen Genauigkeit und Geschwindigkeit:
  - je grösser die Batch Size, desto schnelleres Training und so weniger Modellanpassungen, umso schlechter der Lerneffekt für unbekannte Daten (Overfitting)
  - je kleiner die Batch Size, desto häufiger die Anpassungen (aber optimales Modell wird nicht erreicht)
  - Als Faustregel bietet sich der Wert 32 an

```
# Hyperparameter definieren
BATCH_SIZE = 32 # 32 ist eine faustregel für gute Batchsize
```

Case Study CNN

Hyperparameter: Deep dive – Optimizer

- Ein Modell wird trainiert, in dem die einzelnen Elemente einer Batch vom Modell eingelesen werden und das Modell anhand Inputs ein Ergebnis vorhersagt.
- Ergebnis wird mit dem realen verglichen und ein Ergebnisdelta durch die Loss-Funktion berechnet.
- Kostenfunktion wird dann auf Basis der einzelnen Loss-Funktionsergebnisse je Batch berechnet.
- Optimizer hat die Aufgabe, die Parameter des Modells so zu optimieren, daß das Ergebnis der Kostenfunktion minimiert wird. Dadurch wird die Güte des Modells verbessert.
- Es gibt verschiedene Optimizer, aber wir verwenden meist ADAM oder RSMPROP.

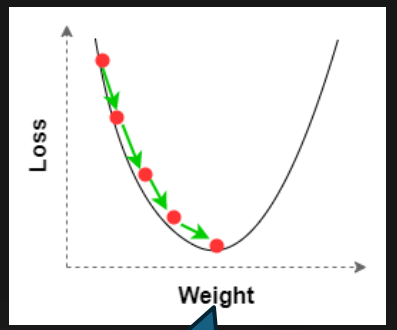
```
# definieren des Optimierers und der Lernrate.
learning_rate = 0.001
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
```

Case Study CNN

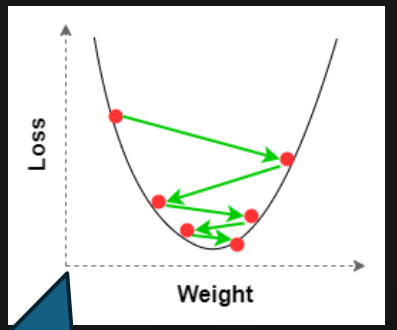
Hyperparameter: Deep dive – Optimizer

- Lernrate ist Schrittweite Optimizers je Durchlauf und ist somit wichtigster Hyperparameter.
- Die Wahl ist erfahrungswertabhängig sie sollte weder zu klein noch zu groß sein. Ein guter Erfahrungswert ist 0.001.
- Die Lernrate kann während des Trainings angepaßt werden. Experten fangen oft mit höherem Wert an und reduzieren die Lernrate Schritt für Schritt (Learning rate Decay).

```
# definieren des Optimierers und der Lernrate.
learning_rate = 0.001
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
```



Lernrate zu klein:
Dauert sehr lange bis optimale Werte gefunden werden (oder sogar nie)



Lernrate zu groß:
Algorithmus macht sehr große Schritte und „überspringt“ die optimalen Werte

Case Study CNN

Modell kompilieren

```
model.compile(optimizer=optimizer,
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
```

- optimizer = optimizer verwendet den auf vorherigen Seite definierten Optimierer
- loss = auf welche Zielgröße wir das Modell optimieren sollen. Bei einer Klassifikation nehmen wir bei 2 Zielklassen „binary_crossentropy“, bei mehreren Zielgrößen „categorical_crossentropy“ oder „sparse_categorial_cross_entropy“ (falls Label eine Zahl ist wie hier)
- Metrics = auf welche Metrik verbessert werden soll.

Case Study CNN

Modell kompilieren

```
model.compile(optimizer=optimizer,
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
```

- optimizer = optimizer verwendet den auf vorherigen Seite definierten Optimierer
- loss = auf welche Zielgröße wir das Modell optimieren sollen. Bei einer Klassifikation nehmen wir bei 2 Zielklassen „binary_crossentropy“, bei mehreren Zielgrößen „categorical_crossentropy“ oder „sparse_categorial_cross_entropy“ (falls Label eine Zahl ist wie hier)
- Metrics = auf welche Metrik verbessert werden soll.

Case Study CNN

Modell trainieren

```
history = model.fit(x=X_train,  # zuweisung trainingsdaten
                    y= y_train, # zuweisung label für die trainingsdaten
                    validation_data = (X_test, y_test),  # an welchen daten wird modell validiert
                    batch_size=BATCH_SIZE, # Definition Batch size
                    epochs = EPOCHS, # definition Anzahl Trainingsdurchläufe
                    use_multiprocessing=True)
```

Die Ergebnisse des Trainierens werden in der Liste history gespeichert

Case Study CNN

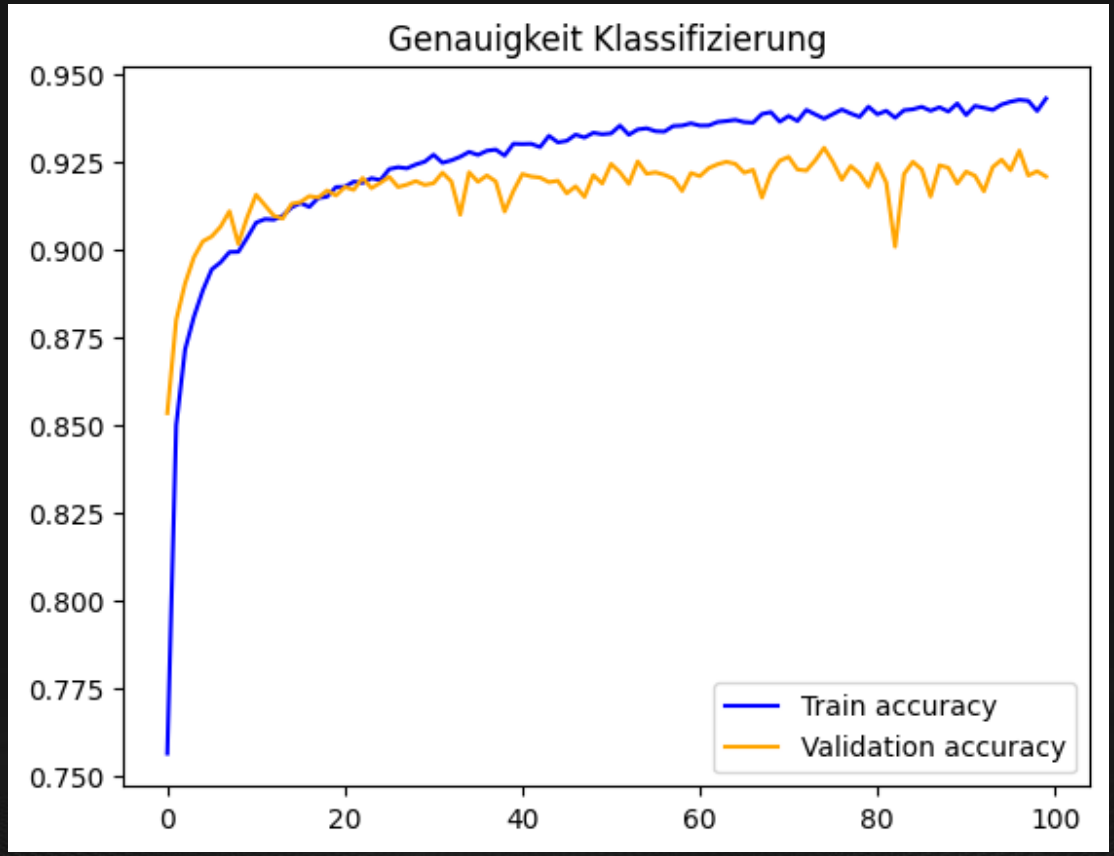
Modellgüte visualisieren

```
# Plotte Genauigkeit
plt.title('Genauigkeit Klassifizierung')
plt.plot(history.history['accuracy'], color='blue', label='Train accuracy')
plt.plot(history.history['val_accuracy'], color='orange', label='Validation accuracy')
plt.legend()
plt.show()
```

plt.title('Genauigkeit Klassifizierung'): definiert den Bildtitel mit „Genauigkeit Klassifizierung“
plt.plot(history.history['accuracy'], color='blue', label='Train accuracy') = Plotte einen Graphen für die Trainingsgenauigkeit in blau
plt.plot(history.history['val_accuracy'], color='orange', label='Validation accuracy') = Plotte einen Graphen für die Trainingsgenauigkeit in orange
plt.legend() = Plotte eine Legende für den Graphen
plt.show() = Zeige den Graphen an.

Case Study CNN

Modellgüte visualisieren

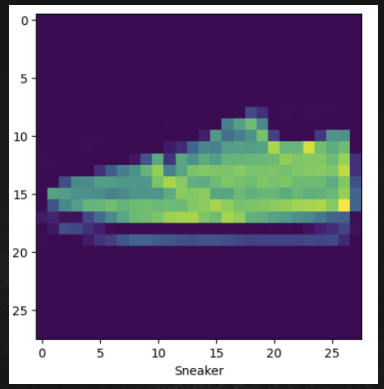


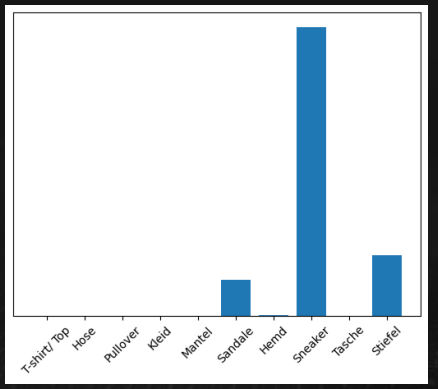
Case Study CNN

Modell anwenden („Inferenz“)

```
[119]  def classify_pic(samplePicture, picindex):

         sample_pic_aufbereitet = samplePicture.reshape (1, 28, 28, 1)
         predicted_model = model.predict(sample_pic_aufbereitet)

         temp_picture = samplePicture.reshape(28,28) # wir löschen den Schwarz-weiß Kanal fürs Anzeigen des Bildes

         plt.imshow(temp_picture)

         plt.xlabel(class_names[y_test[picindex]])

         plt.show()

         print("Klasse vorhergesagt: {}".format(class_names[np.argmax(predicted_model[0])]))
         print("Gesamte Wahrscheinlichkeiten:")

         plt.bar(range(10), predicted_model[0])
         plt.grid(False)
         x_vals = range(0, len(class_names))
         plt.xticks(x_vals, class_names, rotation=45)
         plt.xticks(range(10))
         plt.yticks([])
```

```
classify_pic(X_test[60], 60)
```





Case Study CNN

Testen des gesamten Modells

```
from sklearn.metrics import classification_report

loss_model, accuracy_model = model.evaluate(X_test, y_test)
print('Test accuracy :', accuracy_model)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2784 - accuracy: 0.9223
Test accuracy : 0.9222999811172485
```

Case Study CNN

Testen des gesamten Modells: Metriken

```
y_pred = model.predict(X_test, batch_size=64, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)
print(classification_report(y_test, y_pred_bool, target_names=class_names))
```

```
157/157 [==============================] - 1s 3ms/step
              precision    recall  f1-score   support

T-shirt/ Top       0.89      0.86      0.87      1000
        Hose       1.00      0.98      0.99      1000
    Pullover       0.88      0.89      0.88      1000
       Kleid       0.91      0.94      0.93      1000
      Mantel       0.86      0.88      0.87      1000
     Sandale       0.99      0.98      0.99      1000
        Hemd       0.78      0.77      0.78      1000
     Sneaker       0.94      0.99      0.96      1000
      Tasche       0.99      0.99      0.99      1000
     Stiefel       0.99      0.95      0.97      1000

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

- Precision: wie viele der vorhergesagten labels waren richtig?
- Recall: wie viele Elemente wurden korrekt vorhergesagt?
- F1 Score: Kombination aus beiden.

Die Metriken zeigen, daß das Modell für Hosen recht gut ist, für Tshirts aber optimierbar ist

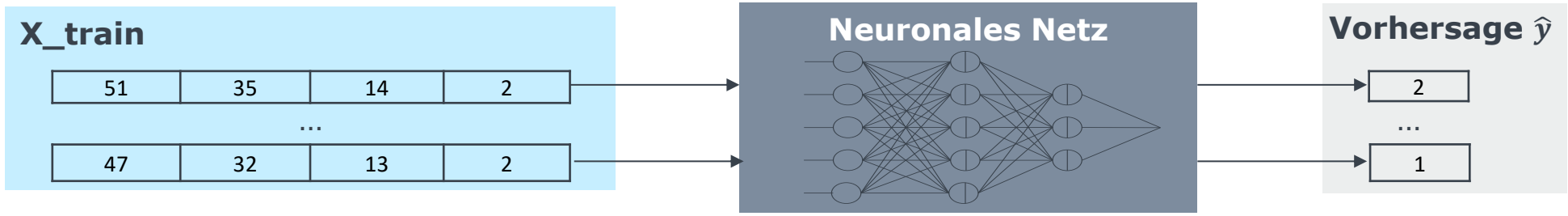
Sources

- https://alzaibkarovalia.medium.com/building-a-self-driving-vehicle-in-gta-v-using-deep-learning-and-convolutional-neural-network-696b38b4c81e
- https://github.com/Alzaib/Autonomous-Self-Driving-Car-GTA-5
- https://pythonprogramming.net/next-steps-python-plays-gta-v/

Backup

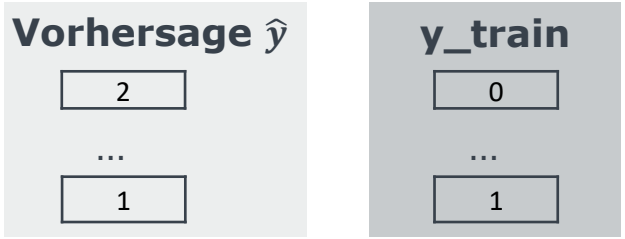
TRAINING VON NEURONALEN NETZEN

1. Feed Forward: Vorhersage Wert je **Sample** (individuelle Zeile Datensatz).



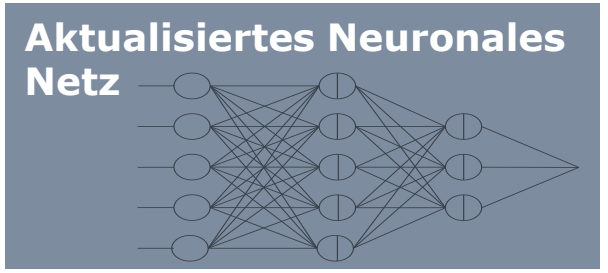
2. Berechnen Modellgüte und Fehler des Modells

- **Loss Function:** Berechnen Delta zwischen realem und vorhergesagtem Wert.
- **Cost Function:** Durchschnitt der Losses für jedes Sample des Trainingsset.



3. Minimierung Fehler und Anpassen Modellparameter:

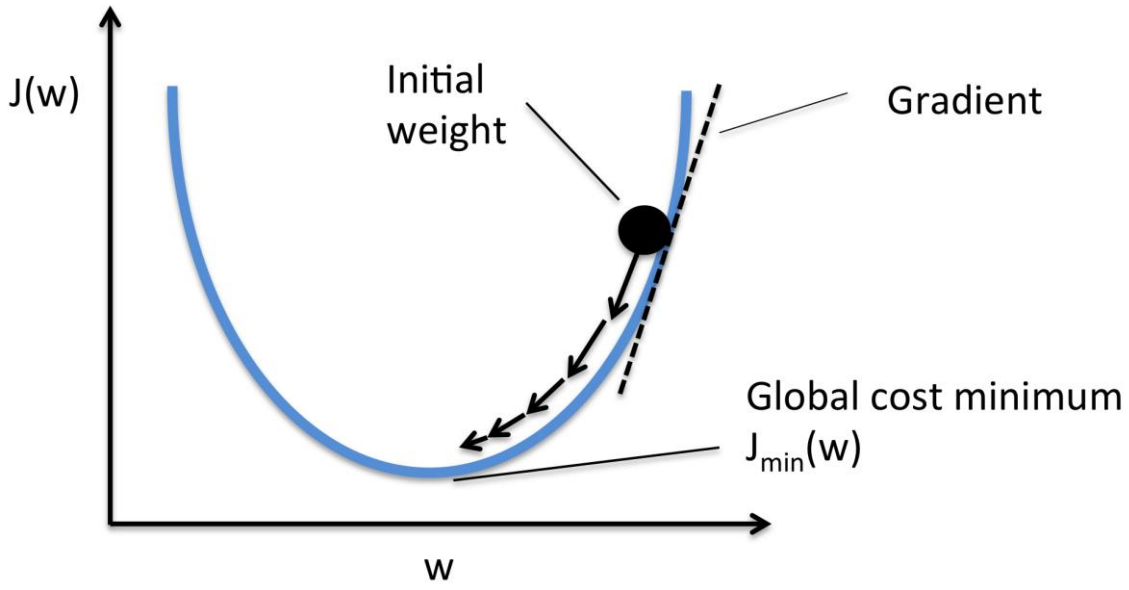
- Aufstellen Gleichung für Minimierung Cost Function.
- Lösen Gleichung für die Modellparameter per Gradient Descent[1].
- Aktualisierung Modellparameter per **Backpropagation**[2].



Quellen: 1 Gradient Descent: Hadamard, J.: "Memoire sur le problème d'analyse relatif a l`equilibre des plaques 'elastiques' encastrees", 1908. Zuerst eingesetzt für Cost Function durch Dreyfus, S. E.: "The computational solution of optimal control problems with time lag", 1973.
2 Linnainmaa, S.: "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors", 1970. Erster Einsatz in NN: Werbos, P.: "Applications of advances in nonlinear sensitivity analysis". In Proceedings of the 10th IFIP Conference, 1981.

DIE ANPASSUNG DER PARAMETER DES NEURONALEN NETZES ERFOLGT DURCH GRADIENT DESCENT.

2 Möglichkeiten für Berechnen globales Minimum[1]:
1. Lösen Gleichung Cost-Fkt J(w) = 0 für alle Gewichte/ Bias
2. Iteratives Verfahren Gradient Descent



Ablauf Gradient Descent Algorithmus:

1. Initialisiere zufällig gewählte Gewichte w sowie Bias b.
2. Iteriere über alle Parameter:
   1. Feed-Forward: Berechne $\hat{y}$ (prädizierter Wert).
   2. Berechne Loss-Funktion J(w) für w und b.
   3. Berechne Gradienten, d.h. ziehe von w und b ein kleines Delta η*w und η*b ab → **Graduelles Absteigen!** Dieses η ist die **Lernrate,** ein wichtiger **Hyperparameter**.
3. Gehe zu 2 solange bis sich Loss nicht mehr signifikant ändert.

Bildlich veranschaulicht: Stellen Sie sich vor, daß Sie vom Gipfel eines Berges zum tiefsten Punkt laufen wollen.

Ziel Algorithmus: iteratives Wählen von Werten für Gewichte und Bias, um Cost-Funktion so weit wie möglich zu minimieren. Ein iteratives Verfahren ist weniger rechenaufwendig als die Alternative Einsetzen aller Parameter.

Quelle Bild: Link
1 Bestimmen globales Minimum einer Funktion erfolgt über Bilden erste Ableitung und gleich 0 setzen.

FALLBEISPIEL KLASSIFIKATION ANHAND GROSSER DATENSÄTZE ZEIGT VORTEILE NEURONALER NETZE.

Zufällig generiertes Dataset mit 50‘000 Zeilen und 20 Features:

Algorithmus	**Genauigkeit**	**Dauer Training** (ohne GPU)
Logistische Regression	0.938	296 ms
XGBoost	0.966	5.86 s
Neuronales Netz (nur 20 Epochen)	0.968	1 min 11 s

Zufällig generiertes Dataset mit 500‘000 Zeilen und 30 Features:

Algorithmus	**Genauigkeit**	**Dauer Training** ohne GPU/ mit GPU
Logistische Regression	0.90	2,4 s/ 2,14 s
XGBoost	0.913	1,46 min/ 1,33 min
Neuronales Netz (nur 20 Epochen)	0.955	14 min/ 22 min

Bei großen bis sehr großen Datenmengen bietet neuronales Netz sehr gute Performance bei hoher Trainingszeit

1 Gestoppt in Google Colab ohne Grafikkarte (Abkürzung GPU). Grafikkarten beschleunigen Training enorm, da ihre vielen arithmetischen Recheneinheiten die notwendigen Matrizenmultiplikationen parallel rechnen können statt im Prozessor, der weniger solcher Recheneinheiten hat und deshalb sequentiell berechnet.