h-da HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Einführung in die künstliche Intelligenz

EKI06 – Computer Vision

Prof. Dr. A. del Pino

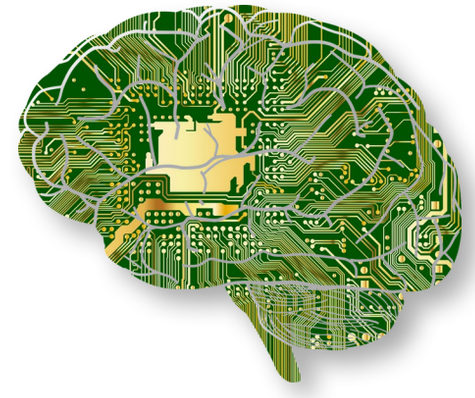


Bild: https://en.wikipedia.org/wiki/Machine_learning#/media/File:Anatomy-1751201_1280.png

Dieser Foliensatz basiert teilweise auf dem “Introduction to Artificial Intelligence”-Kursmaterial von Prof. Dr. B. Humm

Aufbau

- Überblick
- CV Anwendungen
- Neuronale Netze
- Deep Learning mit CNN
- Herausforderung: Ziffernerkennung
- Mini-Test

Die KI-Landkarte

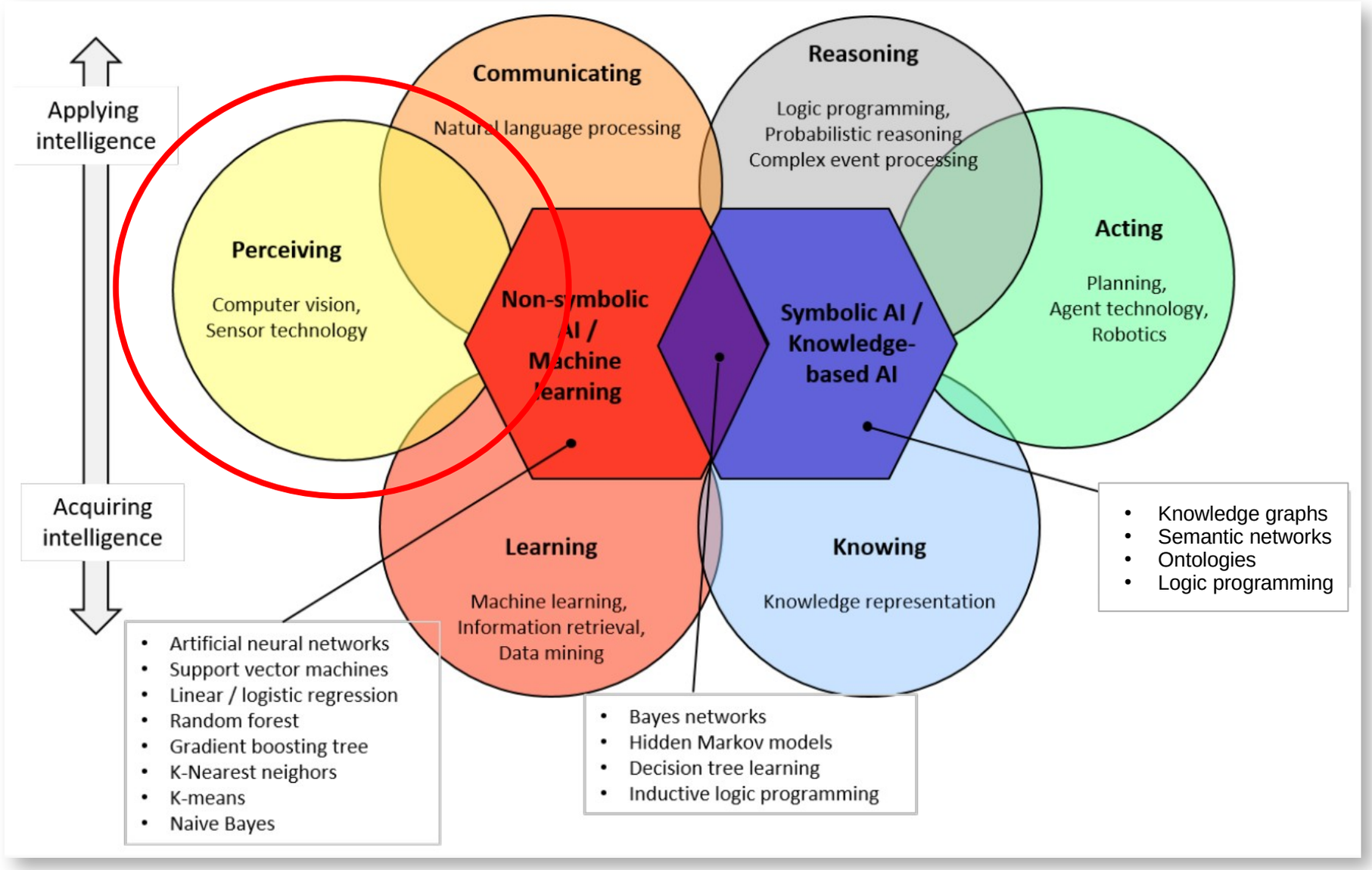


Bild: B. Humm “Applied Artificial Intelligence”, S. 4

Aufbau

- Überblick
- CV Anwendungen
- Neuronale Netze
- Deep Learning mit CNN
- Herausforderung: Ziffernerkennung
- Mini-Test

Optische Zeichenerkennung (optical character recognition, OCR)

Eingabe:
Bild von hand- oder maschinen-geschriebenem Text

Ausgabe:
Textdokument

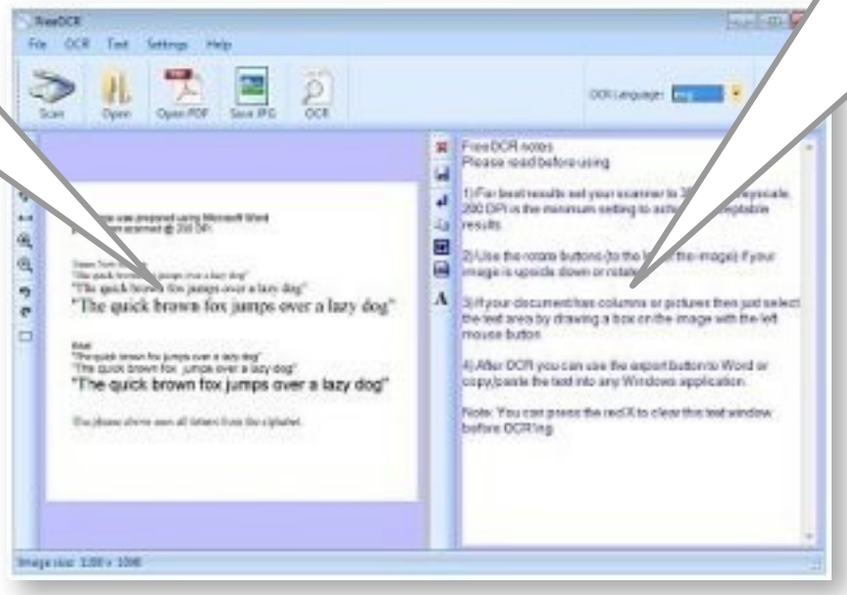


Bild: http://www.paperfile.net/

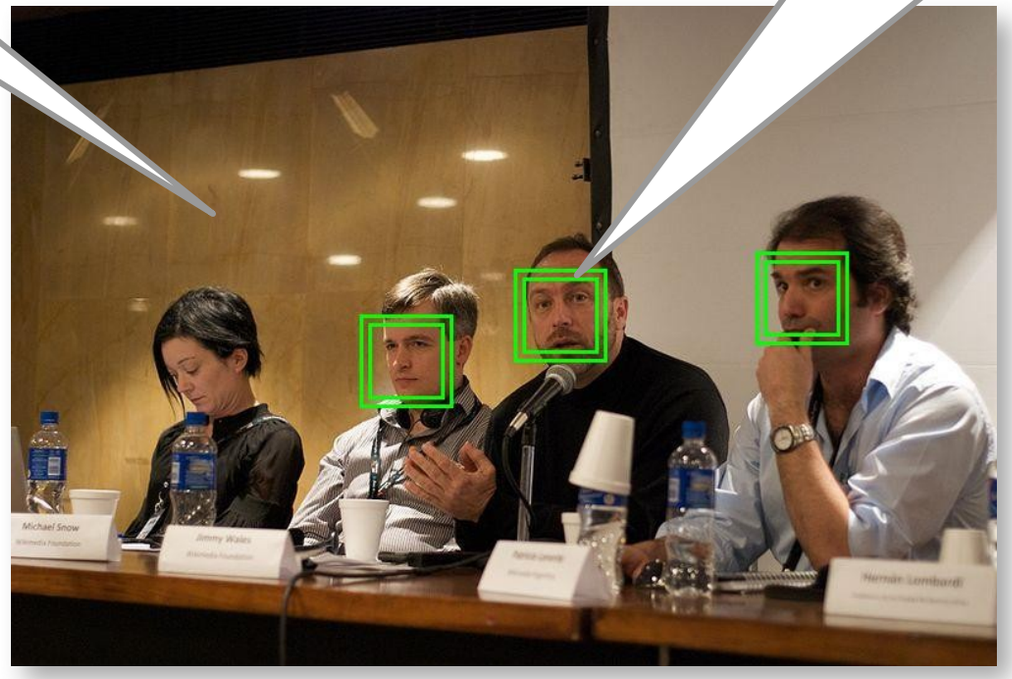
Anwendungsfälle:

- Archivierung und Zugriff auf alte Dokumente
- Adressen scannen auf Briefen
- ...

Gesichtserkennung

Eingabe:
Bild

Ausgabe:
Information, ob auf dem Bild Gesichter sind, Gesichts-Position und Identifikation



Quelle: https://de.wikipedia.org/wiki/Gesichtserkennung

Anwendungsfälle:

- Autotagging von Kamerabildern
- Bilder im Internet finden
- Menschen erkennen
  (z.B. im Flughafen)
- ...

Anwendungen im medizinischen Bereich

Anwendungsfälle:

- Anomalien in CT / PET / MRI / Ultraschall-Bildern
- ...

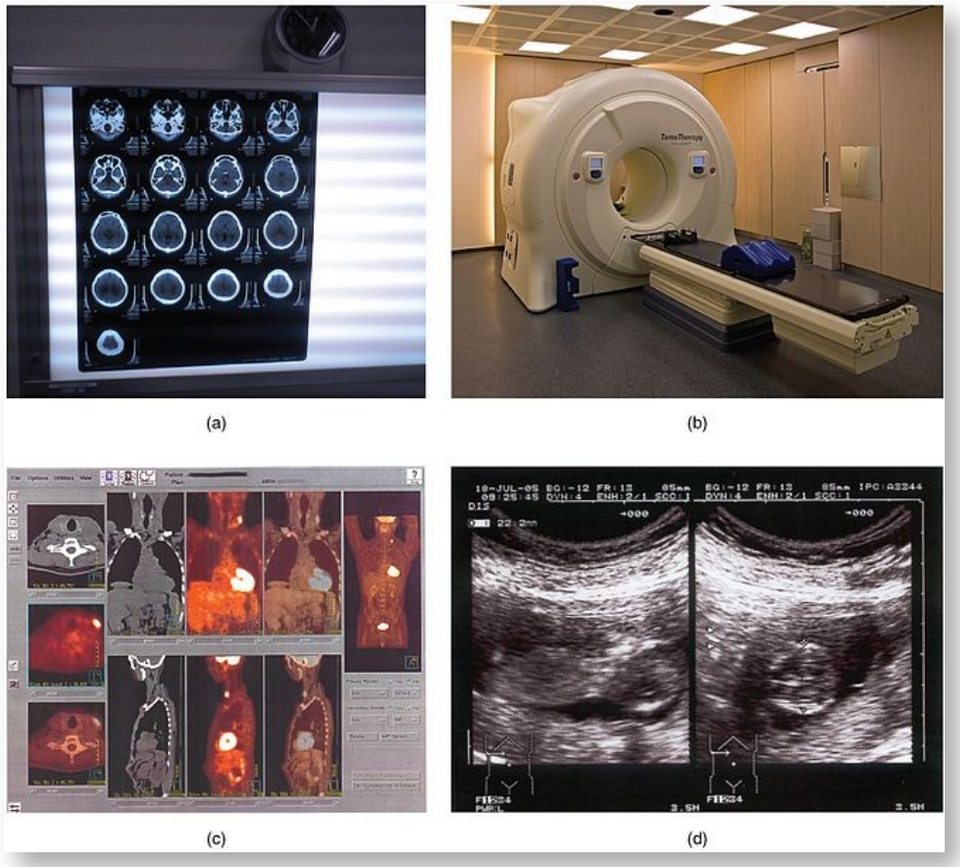


Bild: https://commons.wikimedia.org/wiki/File:113abcd_Medical_Imaging_Techniques.jpg

Industrielle / Landwirtschaftliche Anwendungen

Anwendungsfälle:

- Früchte sortieren
- QM in Fertigungsprozessen
- Roboter in Fertigungsprozessen
- ...

Quelle: https://en.wikipedia.org/wiki/Optical_sorting

Quelle: https://commons.wikimedia.org/wiki/File:KUKA_robot_for_flat_glas_handling.jpg

Anwendungen im Militär und der Luftfahrt

Anwendungsfälle:

- Erkennen von feindlichen Soldaten und Fahrzeugen
- Raketenlenkung
- Autonome Fahrzeuge
- Drohnen, ...

Bild: https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle

Bild: https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle

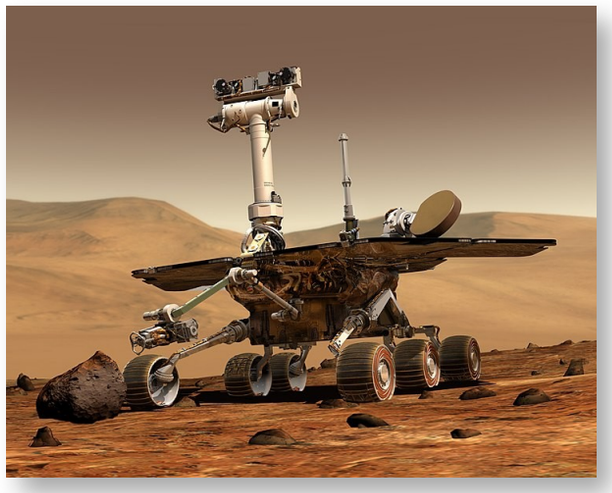


Bild: https://commons.wikimedia.org/wiki/File:NASA_Mars_Rover.jpg

Anwendungen im Automobil-Bereich

Anwendungsfälle:

- Kollisionswarnung
- Parkhilfe / automatisches Parken
- Verkehrsschild-Erkennung
- Autonome Fahrzeuge
- ...

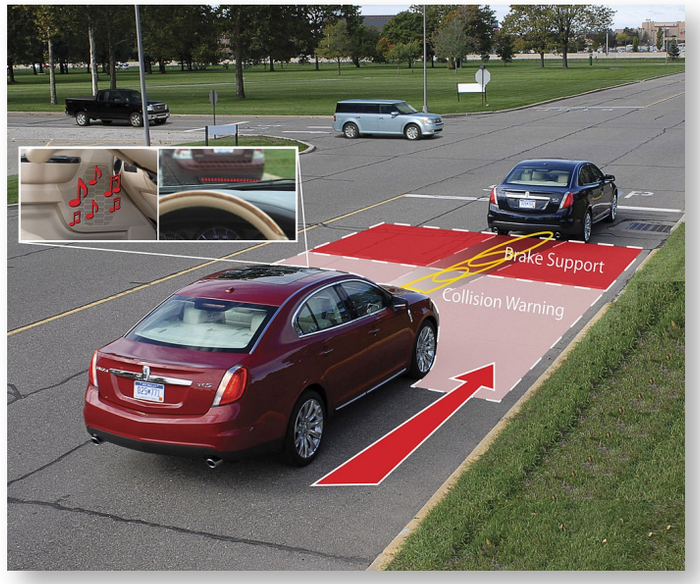


Bild: https://en.wikipedia.org/wiki/Collision_avoidance_system#/media/File:Collision_Warning_Brake_Support.jpg

Bild: https://en.wikipedia.org/wiki/Self-driving_car#/media/File:Waymo_Chrysler_Pacifica_in_Los_Altos,_2017.jpg

Bild: https://en.wikipedia.org/wiki/Automotive_night_vision#/media/File:Audi_A8_2013_(11209949525).jpg

Film

Anwendungsfälle:

- Erzeugen von Szenen in Filmen
- Motion capturing
- ...

Bild: https://en.wikipedia.org/wiki/Avatar_(2009_film)#/media/File:Avatarmotioncapture.jpg

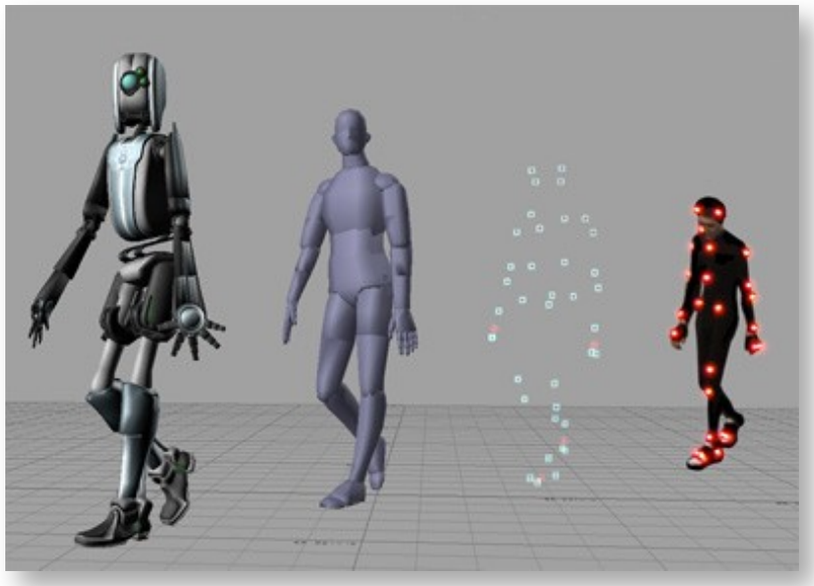


Bild: https://en.wikipedia.org/wiki/Motion_capture#/media/File:Activemarker2.PNG

Aufbau

- Überblick
- CV Anwendungen
- Neuronale Netze
- Deep Learning mit CNN
- Herausforderung: Ziffernerkennung
- Mini-Test

Neuron

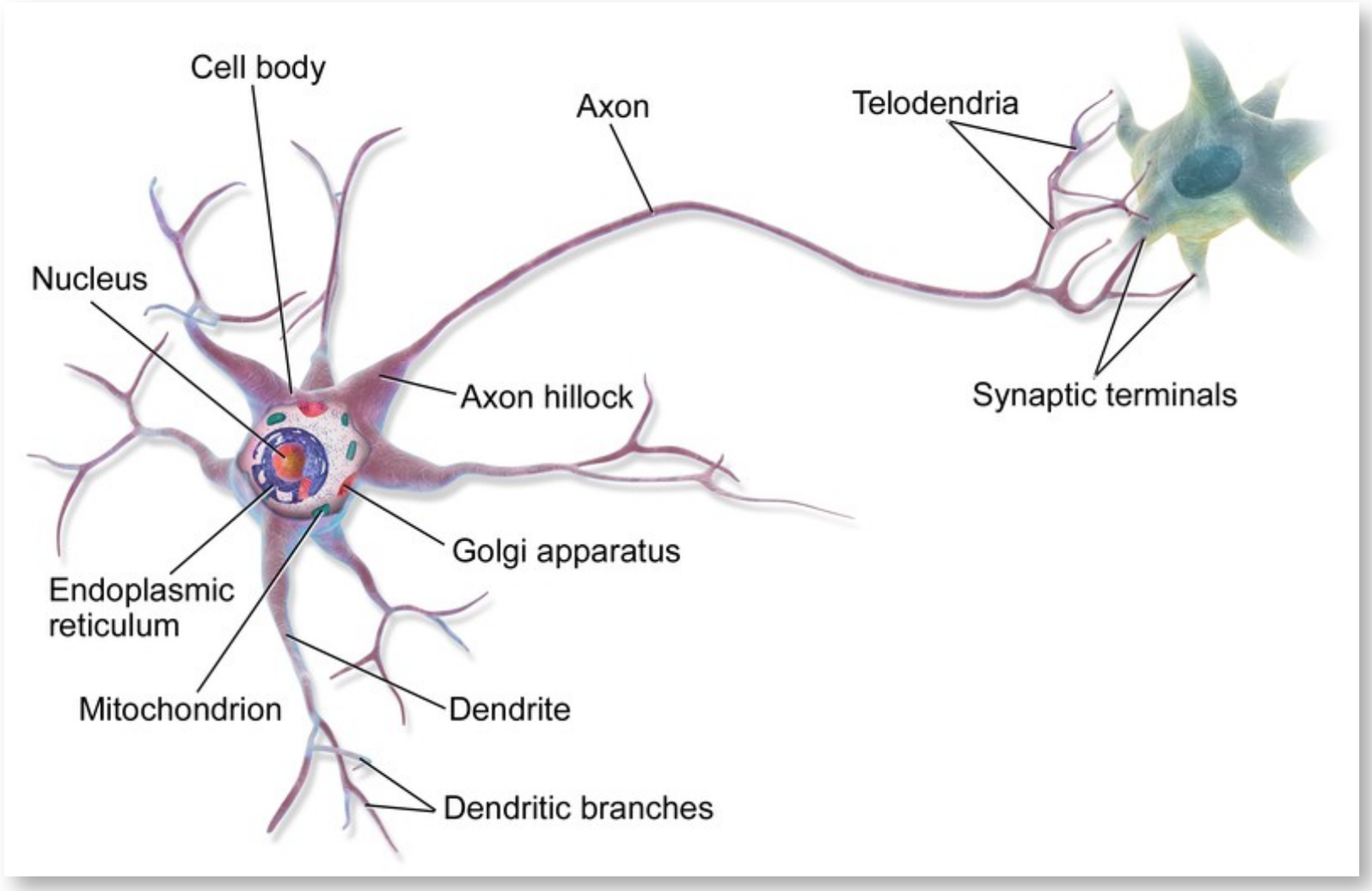
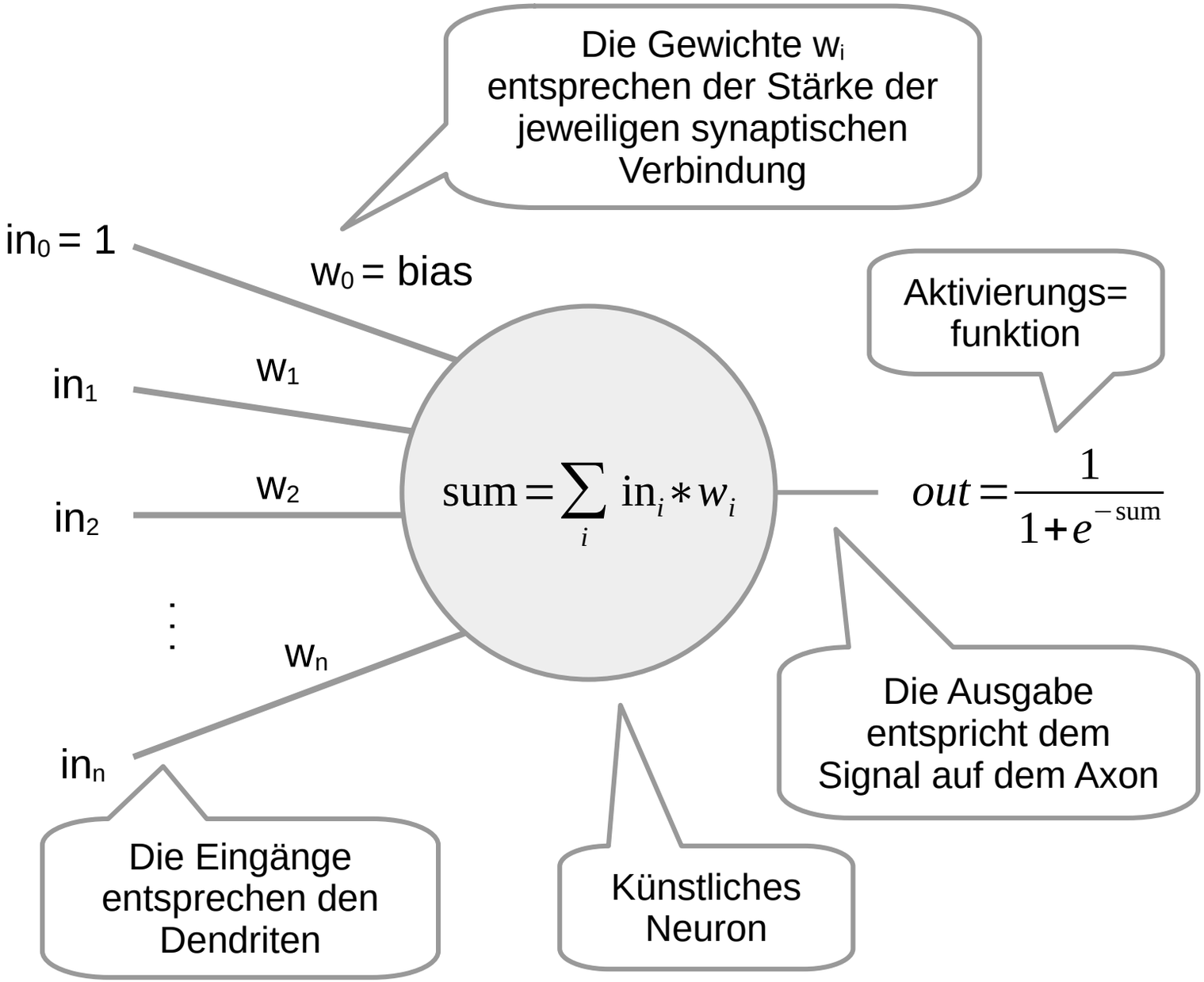
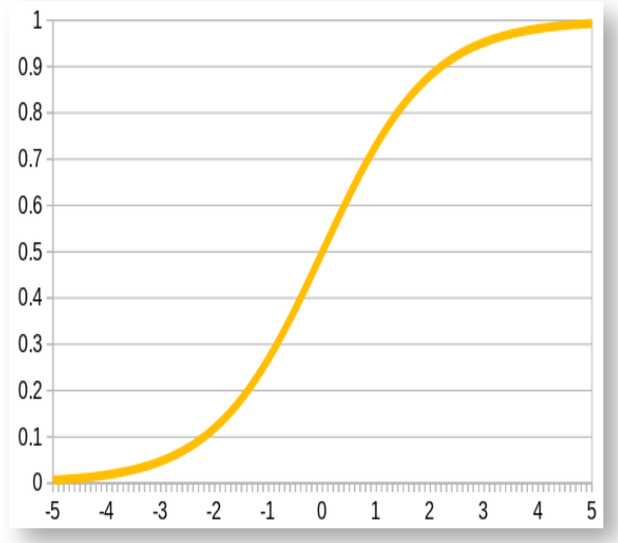


Bild: https://en.wikipedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png

Künstliches Neuron (artificial neuron)



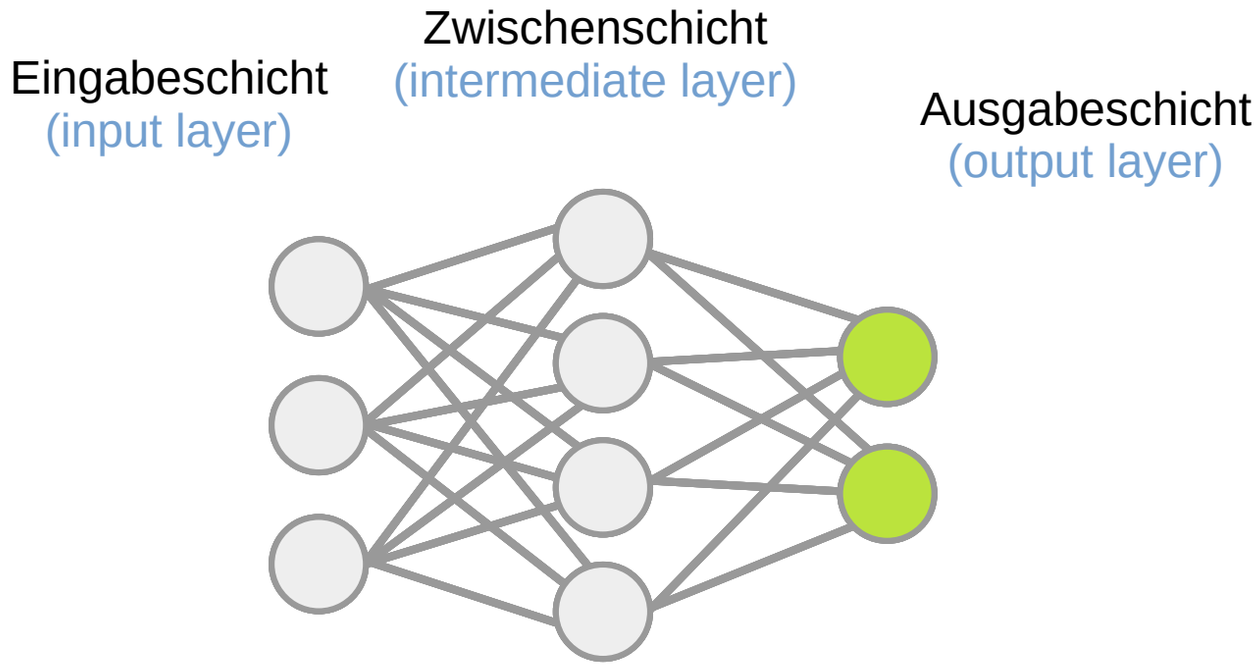
Sigmoid-Funktion



$$f(x) = \frac{1}{1+e^{-x}}$$

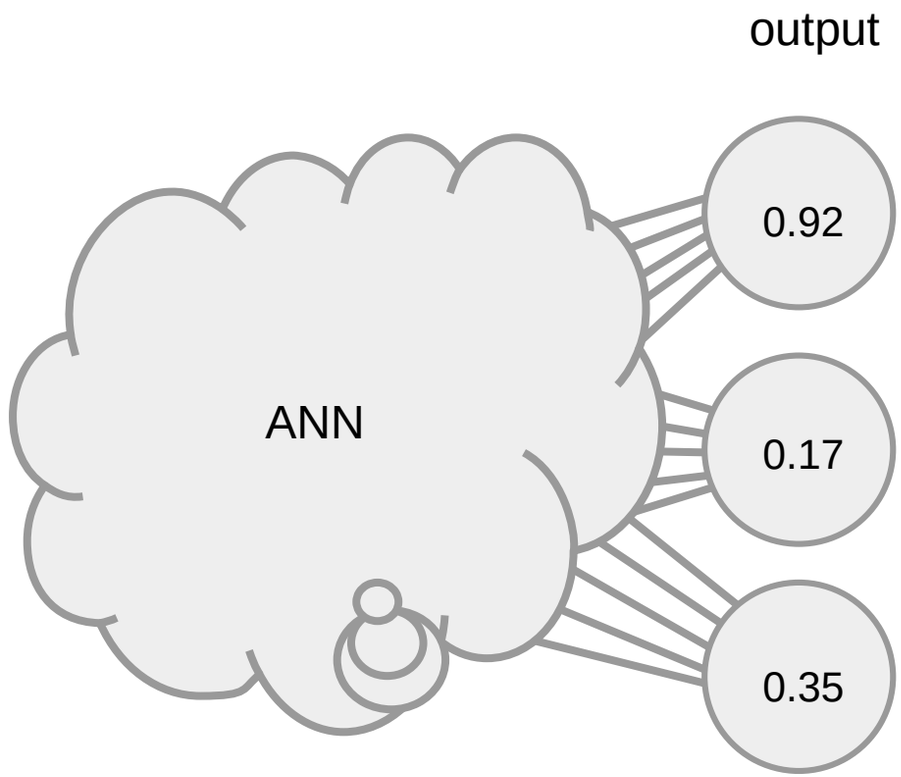
$$f'(x) = f(x) * (1 - f(x))$$

Künstliche neuronale Netzwerke (artificial neural networks, ANN)



- feed-forward network – Ausgaben der einen Schicht sind die Eingaben der nächsten Schicht.
- recurrent network – Ausgaben werden zu Eingängen einer früheren Schicht zurückgeleitet.

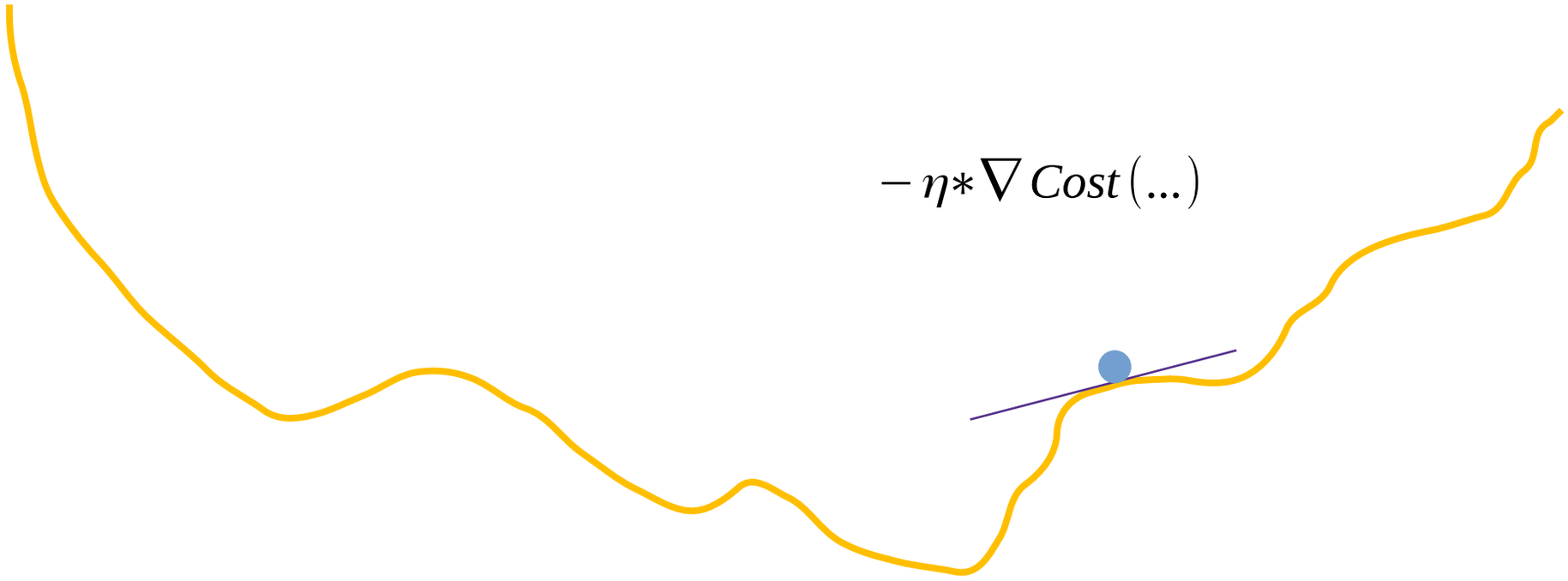
Künstliche neuronale Netzwerke – Fehler



$$Fehler = Cost(W; \text{🚦}) = \sum_i (target - out)^2 = (1.0 - 0.92)^2 + (0 - 0.17)^2 + (0 - 0.35)^2 = 0.1578$$

Kostenfunktion
(cost function)

Künstliche neuronale Netzwerke – Gradientenabstieg (gradient descent)

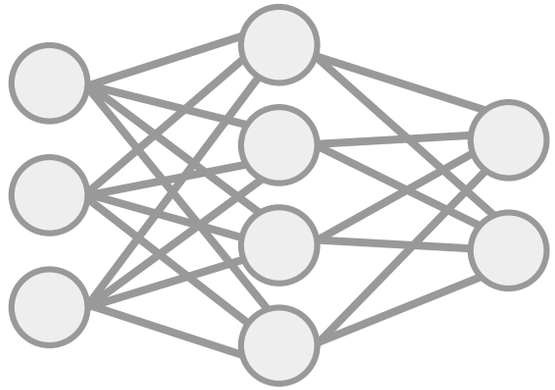


Training künstlicher neuronaler Netzwerke

Bild: https://en.wikipedia.org/wiki/File:Geoffrey_Hinton_-_Collision_2023_-_Centre_Stage_RCZ_1307_(cropped).jpg

Backpropagation

(D.E. Rumelhart, G.E. Hinton, R.J. Williams)

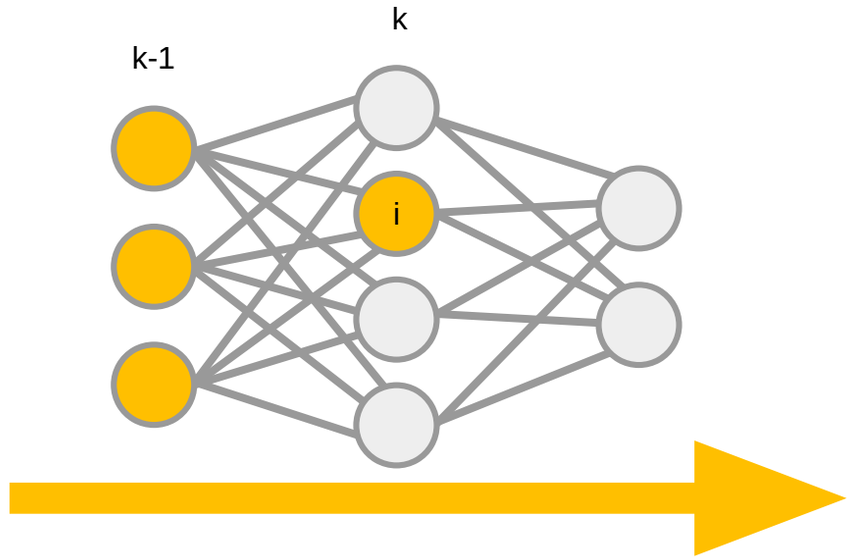


- Schritt 1 – Leite die Trainingsdaten vorwärts durch das Netzwerk.
- Schritt 2 – Berechne den Unterschied (d. h. Fehler) zwischen den tatsächlichen und den gewünschten Ausgaben in der letzten Schicht.
- Schritt 3 – Berechne für jedes Neuron in der letzten Schicht seinen Korrekturwert.
- Schritt 4 – Iteriere rückwärts von der vorletzten Schicht bis zur Eingabeschicht und berechne für jedes Neuron seinen Korrekturwert unter Berücksichtigung der bereits berechneten Korrekturwerte aus der nächsten Schicht.
- Schritt 5 – Korrigiere jedes Gewicht mit Hilfe des Korrekturwerts, der zu dem jeweiligen Neuron gehört.

Training künstlicher neuronaler Netzwerke mit Backpropagation

Schritt 1 – Leite die Trainingsdaten vorwärts durch das Netzwerk.

```
public void forward(double [] input) {
    output[0] = input;
    for (int k=1; k<numLayers; k++) {
        for (int i=0; i<layerSize[k]; i++) {
            double sum = weight[k][i][layerSize[k-1]]; // bias
            for (int j=0; j<layerSize[k-1]; j++) {
                sum += output[k-1][j] * weight[k][i][j];
            }
            output[k][i] = 1.0 / (1.0 + Math.exp(-sum));
        }
    } // next layer k
}
```



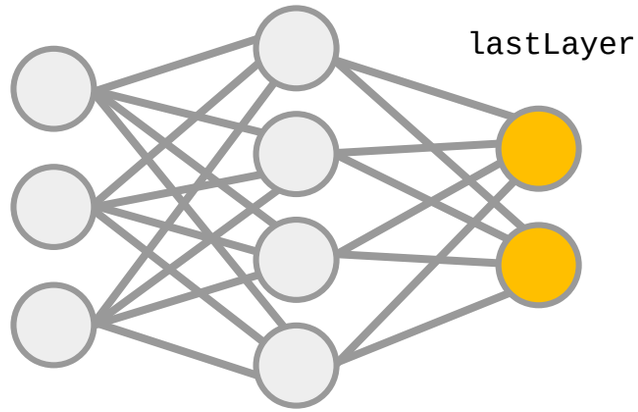
$$\text{sum}_i = \sum_j \text{in}_j * w_{j,i}$$

$$out_i = \frac{1}{1+e^{-\text{sum}_i}}$$

Training künstlicher neuronaler Netzwerke mit Backpropagation

Schritt 2 – Berechne den Unterschied (d. h. Fehler) zwischen den tatsächlichen und den gewünschten Ausgaben in der letzten Schicht.

Schritt 3 – Berechne für jedes Neuron in der letzten Schicht seinen Korrekturwert.




```
public void backpropagate(double [] target) {
    // compute the delta values for the neurons in the last layer
    // and compute the training error
    int lastLayer = numLayers - 1;
    for (int i=0; i < layerSize[lastLayer]; i++) {
        double out = output[lastLayer][i];
        error += (target[i] - out) * (target[i] – out);
        delta[lastLayer][i] = out * (1.0 - out) * (target[i] - out);
    }
    //...
```

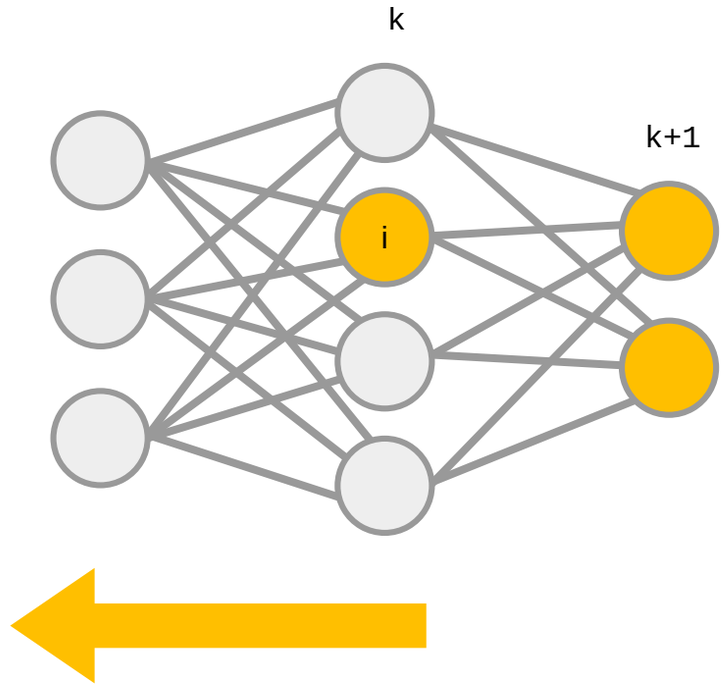
$$e = \sum_i \left(target_i - out_i\right)^2$$

$$\Delta_i = f'\left(\mathrm{sum}_i\right) * \left(target_i - out_i\right)$$

Training künstlicher neuronaler Netzwerke mit Backpropagation

Schritt 4 – Iteriere rückwärts von der vorletzten Schicht bis zur Eingabeschicht und berechne für jedes Neuron seinen Korrekturwert unter Berücksichtigung der bereits berechneten Korrekturwerte aus der nächsten Schicht.

```
// compute the delta values for the upstream layers
for (int k=lastLayer-1; k>0; k--) {
    for (int i=0; i<layerSize[k]; i++) {
        double sum = 0;
        for (int j=0; j<layerSize[k+1]; j++) {
            sum += weight[k+1][j][i] * delta[k+1][j];
        } // next neuron j in layer k+1
        delta[k][i] = output[k][i] * (1.0 - output[k][i]) * sum;
    } // next neuron i in layer k
} // next layer k
```

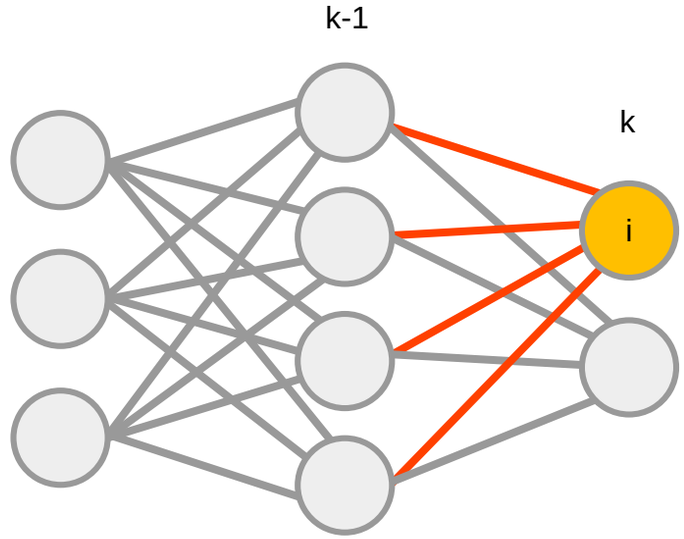


$$\Delta_i = f'(\text{sum}_i) * \sum_j w_{i,j} * \Delta_j$$

Training künstlicher neuronaler Netzwerke mit Backpropagation

Schritt 5 – Korrigiere jedes Gewicht mit Hilfe des Korrekturwerts, der zu dem jeweiligen Neuron gehört.

```
// requires that backpropagate computed the delta values
public void adjust(double eta) {
    for (int k=numLayers-1; k>0; k--) {
        for (int i=0; i<layerSize[k]; i++) {
            for (int j=0; j<layerSize[k-1]; j++) {
                weight[k][i][j] += eta * delta[k][i] * output[k-1][j];
            }
            // adjust the weight from the bias
            weight[k][i][layerSize[k-1]] += eta * delta[k][i] * 1.0;
        } // next neuron i in layer k
    } // next layer k
}
```



$$w_{j,i} = w_{j,i} + \eta * \Delta_i * \text{out}_j$$

Training künstlicher neuronaler Netzwerke mit Backpropagation - Beispiel

Epoche	Trainingfehler	Lernrate	Validationsfehler	Anzahl Treffer	Fehlergenauigkeit
0	45661.53	0.060	4146.35	8006	80.06%
1	13995.44	0.057	1503.75	9049	90.49%
2	9426.43	0.054	1359.95	9155	91.55%
3	8248.04	0.051	1181.44	9266	92.66%
4	7476.28	0.048	1171.29	9270	92.70%
5	6935.89	0.045	1156.09	9293	92.93%
6	6635.06	0.042	1051.59	9354	93.54%
7	6275.22	0.039	982.17	9381	93.81%
8	5999.42	0.036	963.33	9398	93.98%
9	5777.54	0.033	904.77	9434	94.34%
10	5570.45	0.030	917.38	9435	94.35%
11	5322.61	0.027	868.89	9472	94.72%
12	5115.63	0.024	854.24	9475	94.75%
13	4935.88	0.021	807.96	9502	95.02%
14	4747.00	0.018	862.51	9470	94.70%
15	4607.83	0.015	820.57	9493	94.93%
16	4450.35	0.012	792.56	9511	95.11%
17	4368.73	0.009	763.66	9532	95.32%
18	4285.81	0.006	733.84	9543	95.43%
19	4249.72	0.003	699.82	9568	95.68%

Aufbau

- Überblick
- CV Anwendungen
- Neuronale Netze
- Deep Learning mit CNN
- Herausforderung: Ziffernerkennung
- Mini-Test

Deep Learning

- ANN mit einer Kaskade, bestehend aus vielen Zwischenschichten
- Regulärer Aufbau: Jede Schicht nutzt die Ausgabe der voherigen Schicht als Eingabe
- Erkennung von zunehmend komplexeren Elementen: low level (Pixels) über medium level (Formen) hin zu high level (Klassen)

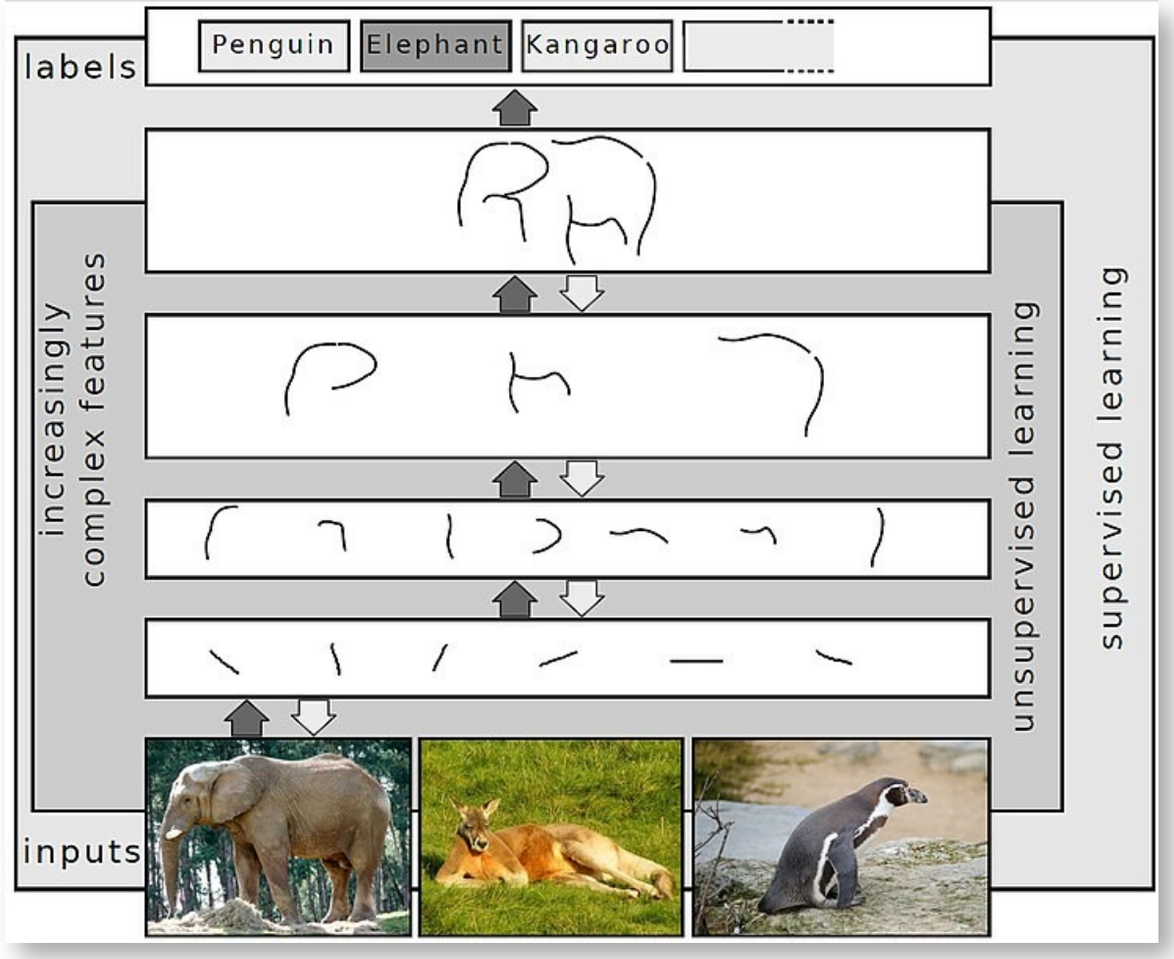
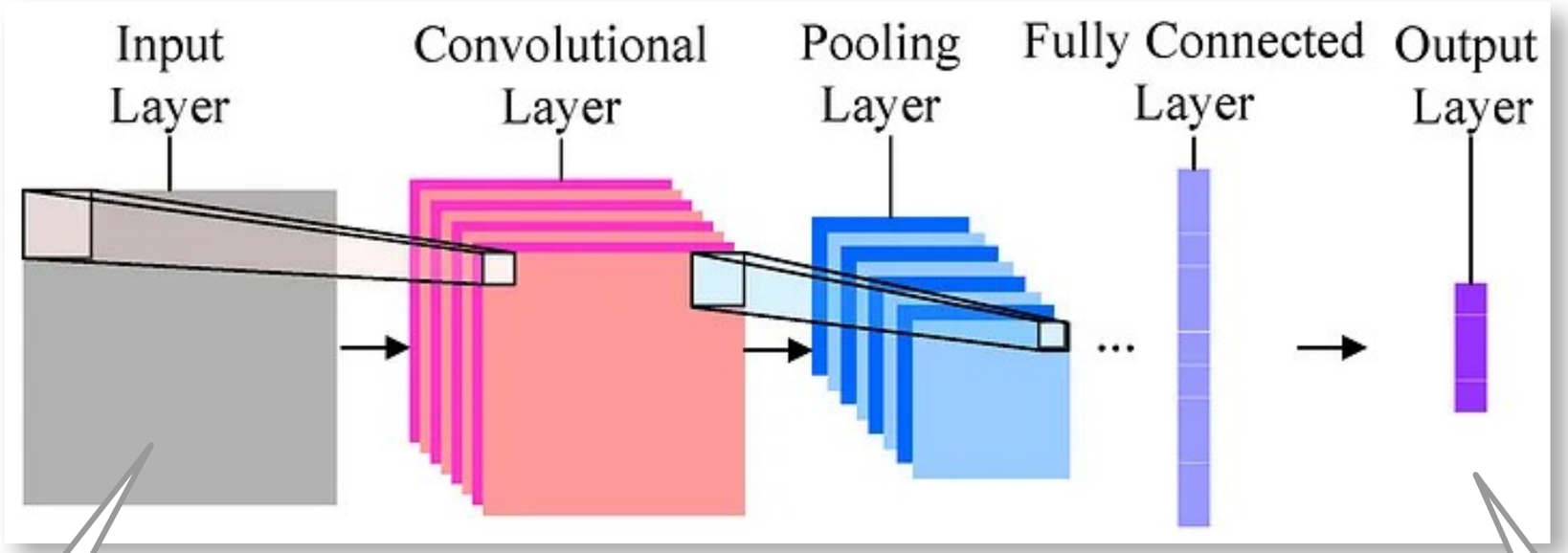


Bild: https://commons.wikimedia.org/w/index.php?curid=82466022

Convolutional Neural Networks (CNN)



Quelle: https://towardsdatascience.com/keras-transfer-learning-for-beginners-6c9b8b7143e

Eingabe: Low level

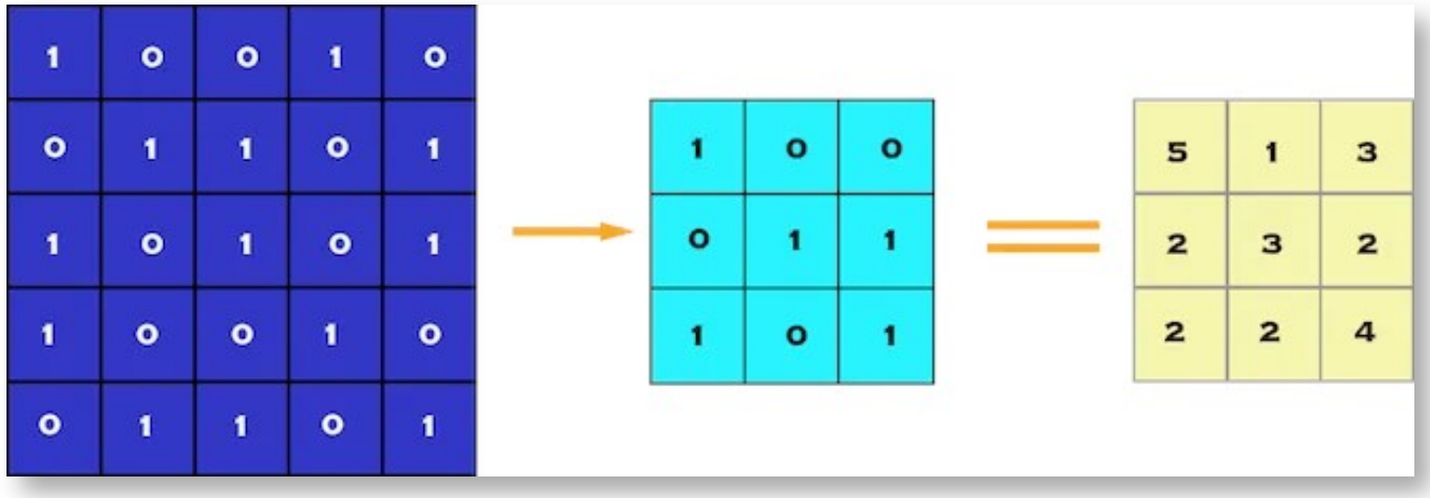
Features erlernen (Medium level)

Klassifikation

Ausgabe: High level

Convolutional Layers

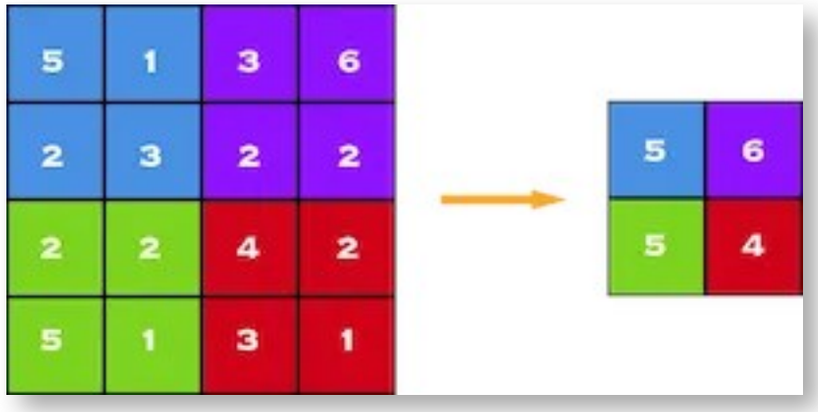
- Pixel werden nur im Zusammenhang mit unmittelbar benachbarten und nahestehenden Pixels betrachet.
  Erhalten der Beziehungen zwischen den verschiedenen Teilen eines Bildes
- Convolution: Filtern des Bildes mit einem kleineren Pixel-Filter.
  Verkleinert die Bildgröße ohne die Beziehung zwischen den Pixeln zu verlieren.



Quelle: https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d

Pooling Layers

- Selektiert einen Bereich von Pixels und berechnet das Maximum, den Durchschnitt oder die Summe der Pixels.
- Beispiel: Max-Pooling mit einem 2x2 Bereich

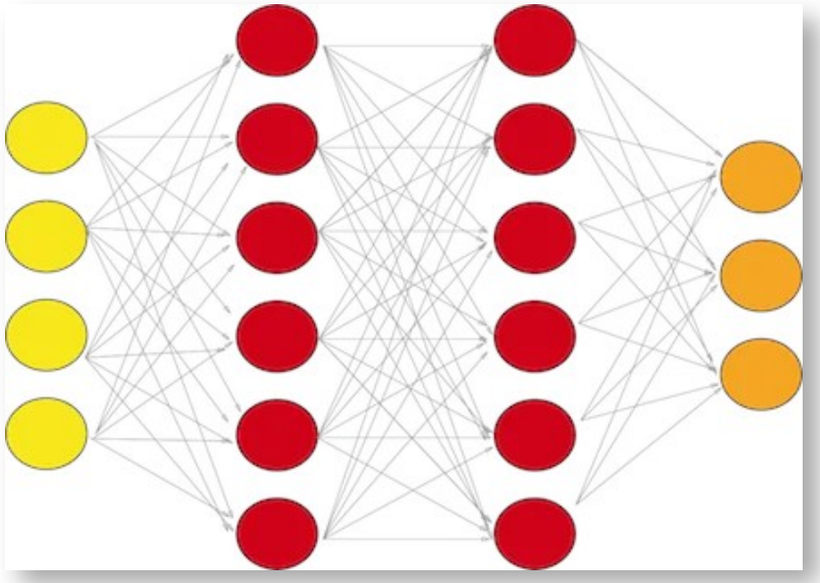


Quelle: https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d

- Einfügen eines pooling layer nach jedem convolution layer reduziert die Komplexität.

Fully connected layers

- Fully connected layer: Jedes Neuron ist mit allen anderen Neuronen verbunden.
- Wird bei der Klassifikation von Bildern benutzt, die Anzahl der Ausgabe-Neuronen entspricht der Anzahl der Klassen.
- Beispiel:



Quelle: https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d

Aufbau

- Überblick
- CV Anwendungen
- Neuronale Netze
- Deep Learning mit CNN
- Herausforderung: Ziffernerkennung
- Mini-Test

Kaggle Wettbewerb: Ziffernerkennung

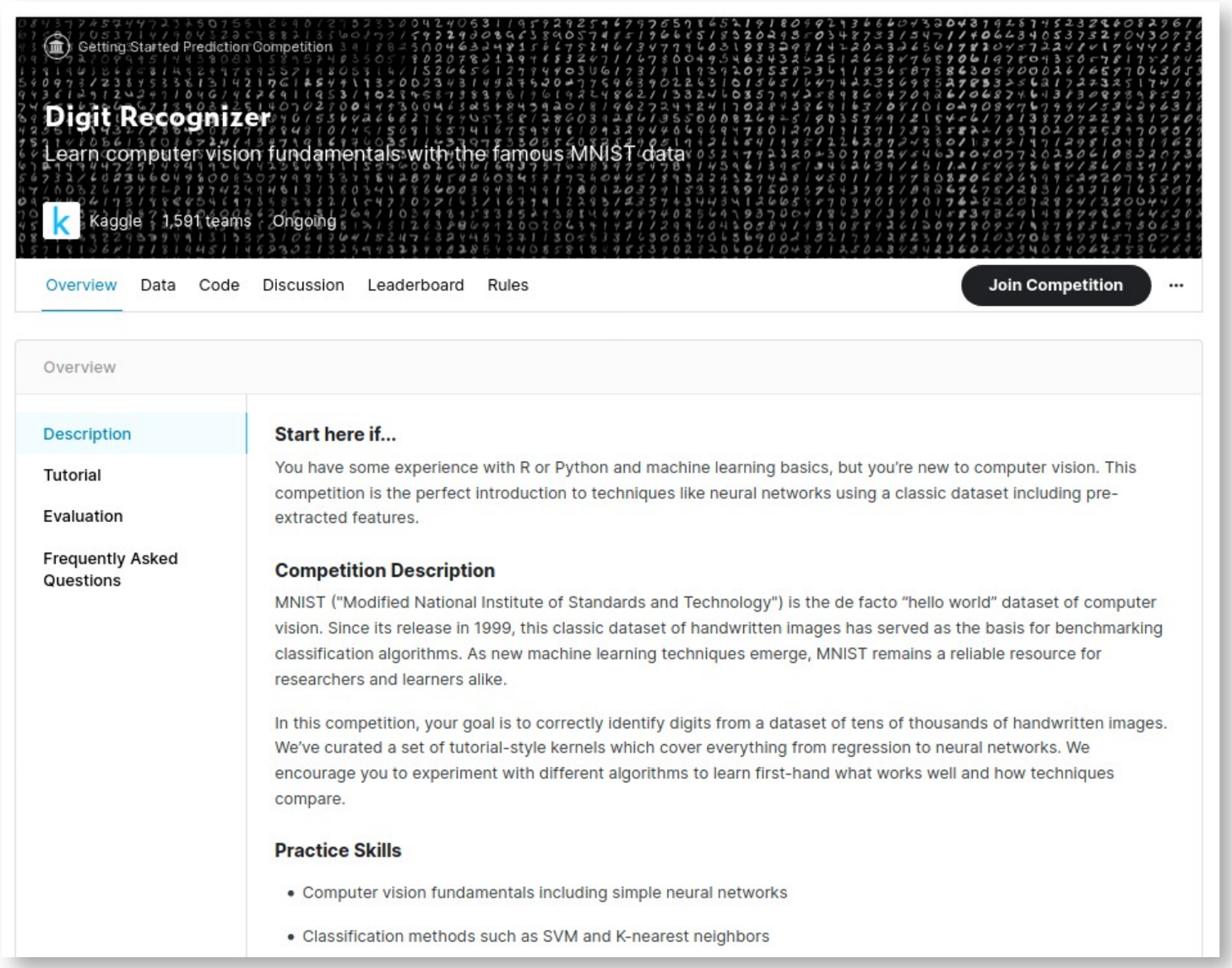


Bild: https://www.kaggle.com/c/digit-recognizer

Die MNIST Datenbank mit handgeschriebenen Ziffern

- MNIST Datenbank: http://yann.lecun.com/exdb/mnist/
- Aufgabe: Identifiziere eine Ziffer in einem Bild (OCR).
- Beispiel:

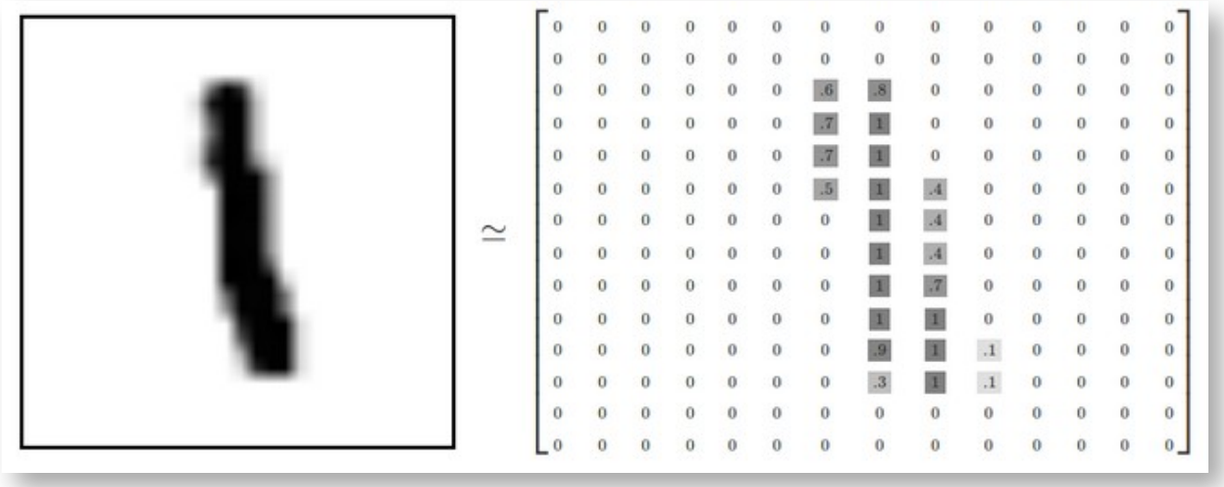
  Eingabe:

Quelle: https://nesusws-tutorials-bd-dl.readthedocs.io/en/latest/hands-on/tensorflow/mnist/

  Ausgabe: 5 0 4 1

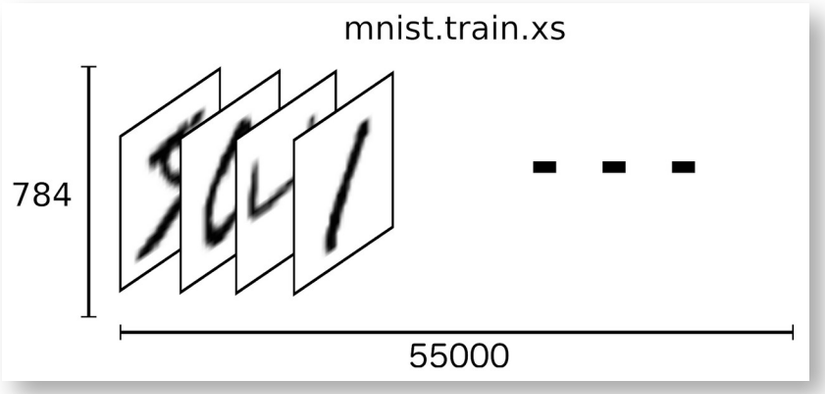
Repräsentation von Bildern

- Einzelbild: Vektor, bestehend aus 28x28 = 784 Zahlen



Quelle: https://nesusws-tutorials-bd-dl.readthedocs.io/en/latest/hands-on/tensorflow/mnist/

- Datensatz zum Training: [55000,784]



Quelle: https://www.w3cschool.cn/doc_tensorflow_guide/tensorflow_guide-get_started-mnist-beginners.html

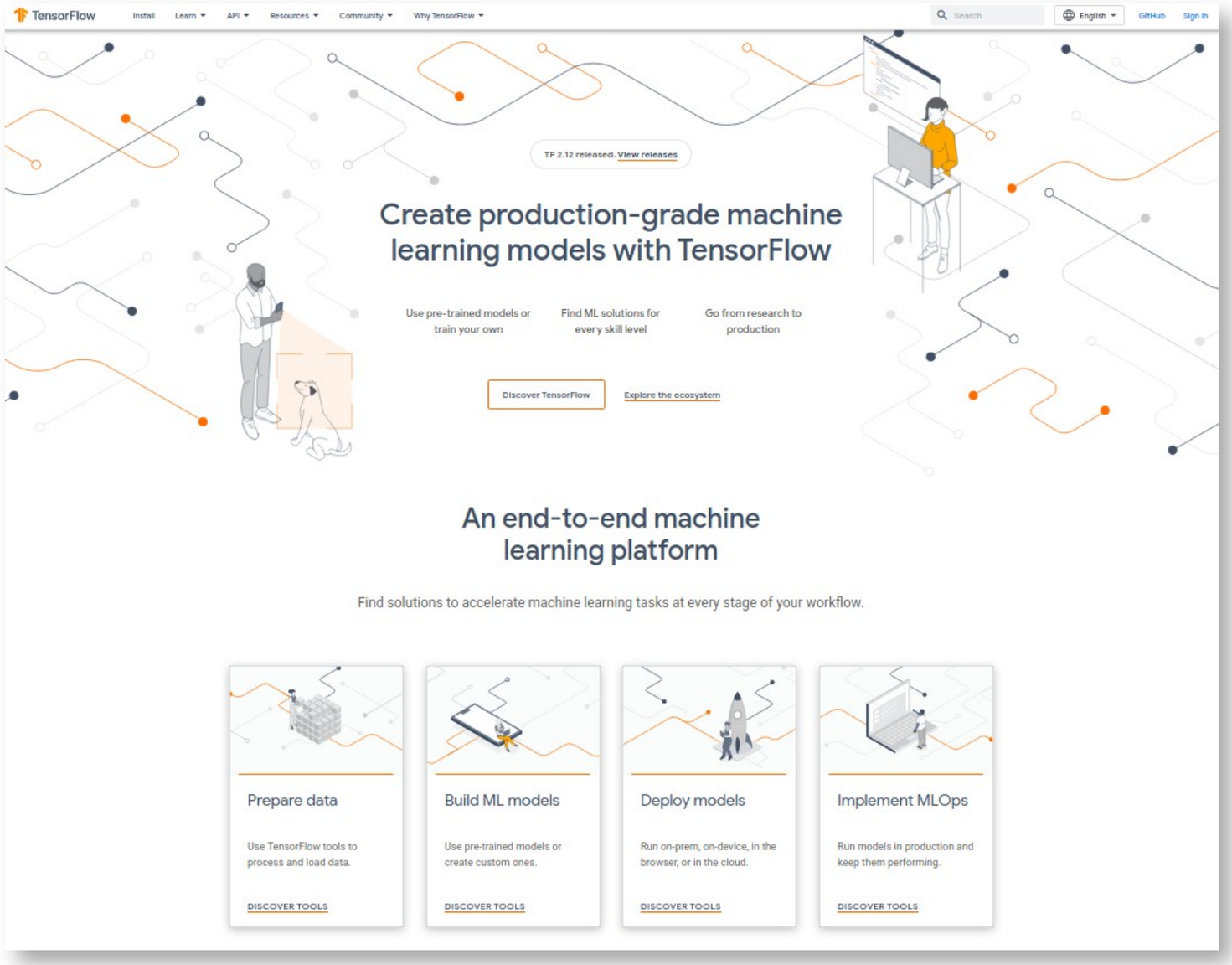
Repräsentation des Datensatzes

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 785 columns

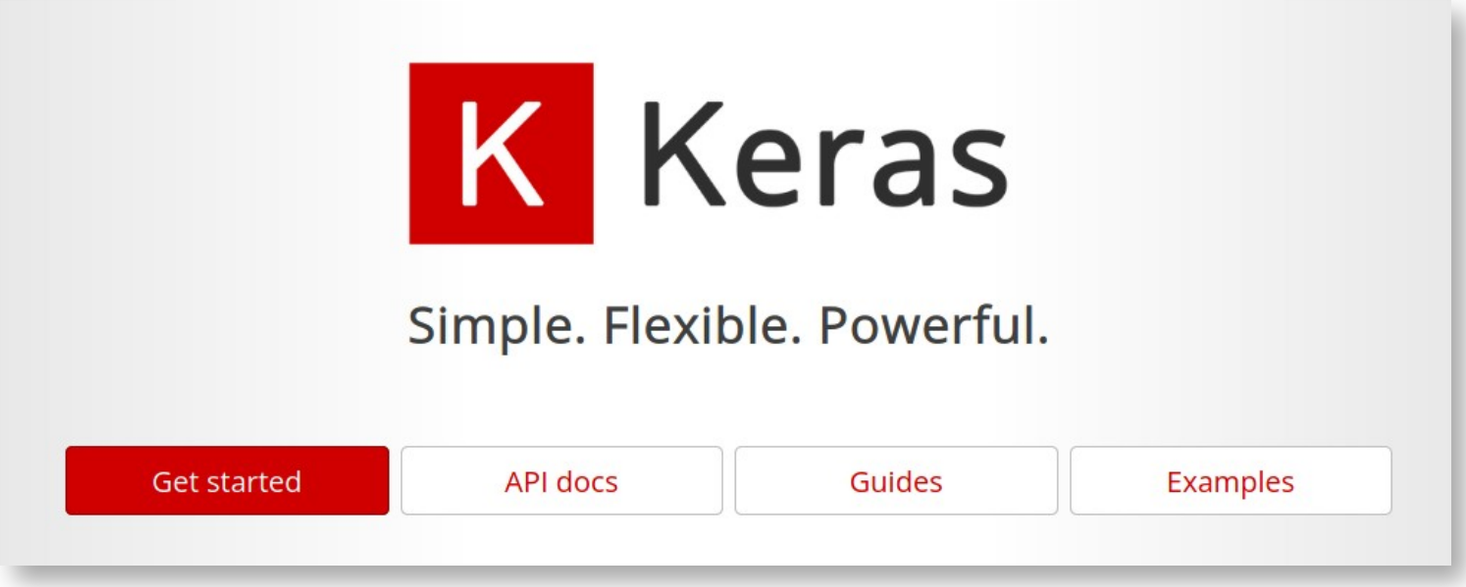
Bild: https://www.kaggle.com/code/nitinbasantwani/digit-recognizer-using-basic-cnn-my-1st-notebook/notebook

TensorFlow: ML Bibliothek von Google



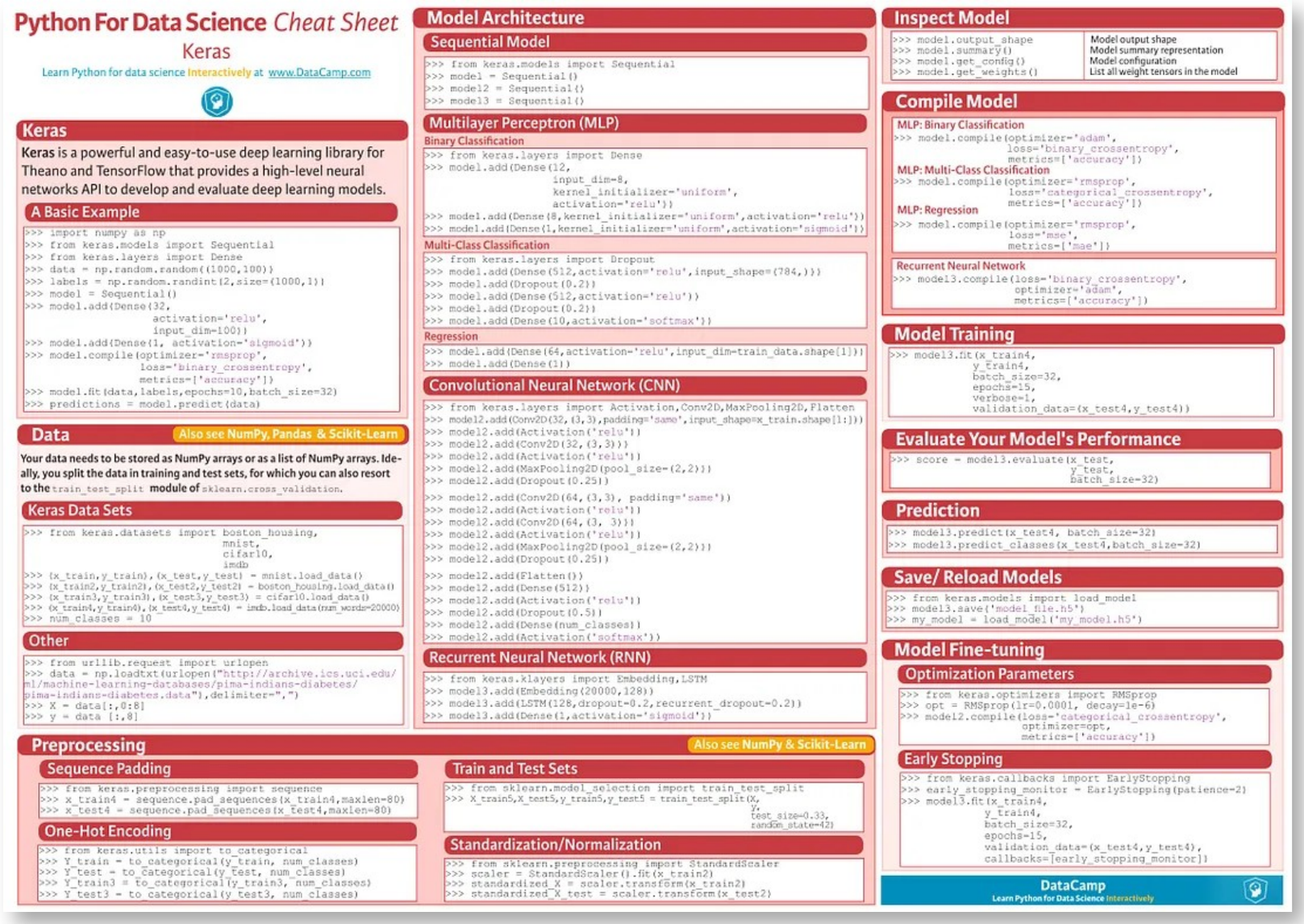
Quelle: https://www.tensorflow.org/

Deep Learning API



Quelle: https://keras.io/

Keras Cheat Sheet



Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

```
>>> model.output_shape
>>> model.summary()
>>> model.get_config()
>>> model.get_weights()
```

Model output shape
Model summary representation
Model configuration
List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

DataCamp
Learn Python for Data Science Interactively

Quelle: https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

Daten laden

```
import pandas as pd


# loading data
train = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')
```

Vorverarbeitung der Daten

```
# Split features and labels in training set
X = train.drop(labels = ['label'], axis = 1)
y = train['label']
# One hot encode labels
Y = to_categorical(y, num_classes = 10)
# Normalize pixel data
X = X/255.0
# Reshape image in 3 dimensions (height = 28px, width = 28px , canal = 1)
X = X.values.reshape(-1, 28, 28, 1)
```

One-Hot Kodierung

Label
3
5
9
1
0
6
5

0	1	2	3	4	5	6	7	8	9
0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	0	0

Definition des Modells

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import RMSprop

model = Sequential([
      Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
      MaxPooling2D(2, 2),
      Conv2D(64, (3,3), activation='relu'),
      MaxPooling2D(2,2),
      Flatten(),
      Dense(512, activation='relu'),
      Dense(10, activation='softmax')
])
```

ANN Topologie: Sequenz von Schichten (layers)

2 convolutional und 2 pooling layers

Aktivierungsfunktion: berechnet die Ausgabe einzelner Neuronen

Flatten in einen Vektor mit einem einzigen Feature

2 fully connected layers

Trainieren des Modells

Optimizer wird für die Optimierung der Metrik benutzt (gradient decent)

Loss function berechnet den Fehler, d.h. der Unterschied zwischen der aktuellen und der vorhergesagten Ausgabe

Metrik: Was soll optimiert werden

```
# Compiling model

model.compile(optimizer=RMSprop(), loss='categorical_crossentropy', metrics=['accuracy‘])

# Fitting model

history = model.fit(X, Y, batch_size=100, epochs = 1)
```

Batch size: Anzahl der Trainingsdaten, die in einer Iteration des Optimierungsprozesses benutzt wird

Anzahl der Epochen: Wie oft wird der ganze Traningsdatensatz für die Optimierung benutzt. Größere Werte verbessern die Treffergenauigkeit, aber auch die Laufzeit des Trainingsvorgangs

Evaluieren des Modells

Das Trainieren eines ANN benötigt eine lange Trainingszeit. Kreuzvalidierung braucht auch eine lange Zeit.

Alternative: train_test_split() erlaubt das einmalige Splitten der Daten im Trainings- und Testdatensatz

```
# Split the data in training and test set

X_train, X_test, Y_train, Y_test = train_test_split(X, Y)

# TODO: train and evaluate
```

Erzeugen einer Submission Datei

Die Testdaten müssen wie die Trainingsdaten vorverarbeitet worden sein

```
# Predict results
results = model.predict(test)
# One hot decoding: Select the index with the maximum probability
results = np.argmax(results, axis = 1)
# Add columns ImageID and Label
results = pd.Series(results, name="Label")
submission = pd.concat([pd.Series(range(1, len(results)+1), name = "ImageId"), results], axis = 1)
# Generate CSV file
submission.to_csv('data/submission.csv', index=False)
```

	A	B
1	ImageID	Label
2	1	2
3	2	0
4	3	9
5	4	9
6	5	3
7	6	7
8	7	0
9	8	3
10	9	0
11	10	3
12	11	5
13	12	7
14	13	4
15	14	0
16	15	4
17	16	3
18	17	3
19	18	1
20	19	9
21	20	0
22	21	9

Aufbau

- Überblick
- CV Anwendungen
- Neuronale Netze
- Deep Learning mit CNN
- Herausforderung: Ziffernerkennung
- Mini-Test

Mini-Test “Computer Vision”, fällig am (Siehe Moodle)

- Nennen Sie Computer Vision Anwendungen
- Was ist ein ANN?
- Was ist back-propagation?
- Was ist deep learning?
- Was ist ein CNN?
- Wie werden die Bilder in der MNIST Datenbank repräsentiert?
- Was ist Keras und Tensorflow? Wie werden Sie benutzt?