Convolutional Neural Networks

Christian Wilms

Computer Vision Group
Universität Hamburg

Wintersemester 2023/24

06. Dezember 2023

Übersicht

Projektaufgabe (Woche 10-14)

Aufgabenstellung

- Bildklassifikationsaufgabe stellen und lösen
- Domäne ist völlig offen
- eine Variante aus:
  - klassischer Ansatz (Merkmale selber wählen)
  - Deep Learning basiert
- es dürfen nur Methoden genutzt werden, die auch verstanden wurden

Beispieldomänen

- Obst
- Verkehrsschilder
- Zahlen
- Blätter
- Zellen
- Landnutzung
- ...

Fragestellungen bei der Themenauswahl

Woher kommen die Daten?

- existierende Datensätze aus dem Internet (bspw. Kaggle oder Paper)
- Bilder selbst aufnehmen/heraussuchen

Welchen Ansatz wollt ihr verfolgen?

- Lassen sich gut Merkmale selbst definieren? (klassischer Ansatz)
- Gibt es genügend Daten? (Deep Learning)

Sind die Bilder zu komplex?

Fragestellungen bei der Themenauswahl

Woher kommen die Daten?

Welchen Ansatz wollt ihr verfolgen?

Sind die Bilder zu komplex?

- nur eine Klasse pro Bild
- große Variabilität innerhalb der Klassen ein Problem
- nicht-uniforme Hintergründe können problematisch sein

Zeitplan

13. Dezember

- Diskussion der Ideen mit den Kleingruppen
- mindestens zwei Ideen für Domänen pro Gruppe

20. Dezember

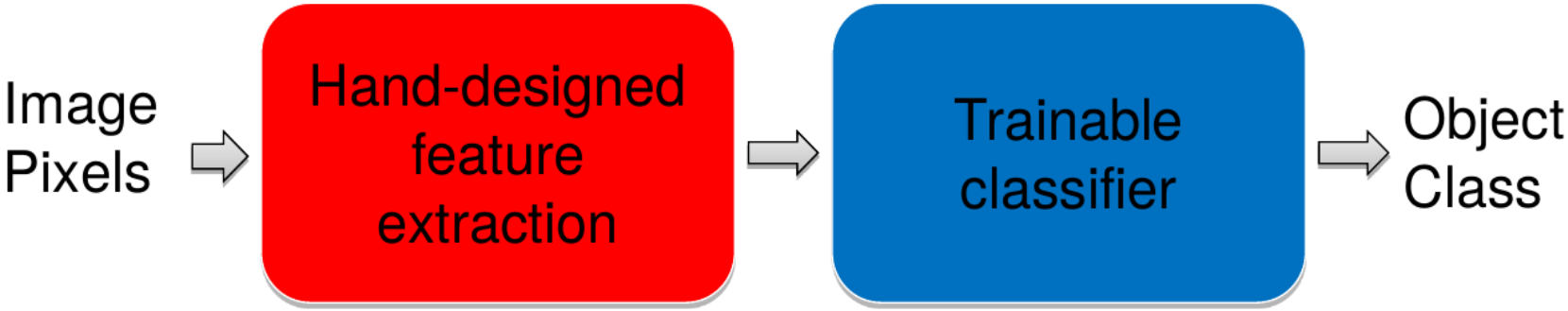
- Präsentation der Idee und des Plans
- 5 Minuten Vortrag (eine Person)

31. Januar

- Abschlusspräsentation der Methoden und Ergebnisse
- 10 Minuten Vortrag (zwei Personen)

Übersicht

Pipeline der Klassifikation



- Mittelwert
- *Standardabweichung*
- Histogramme
- Exzentrizität
- *HOG*
- ...

- Nächster-Nachbar-Klassifikator
- *k-Nächster-Nachbar-Klassifikator*
- **Neuronales Netz**

Merkmale

Was haben die bisherigen Merkmale beschrieben?

- Farbe eines Objekts/Bildes
- Textur eines Objekts/Bildes
- Form eines Objekts

$\Rightarrow$ ein Merkmal für das gesamte Objekt

Wie kann man ein Objekt noch beschreiben?

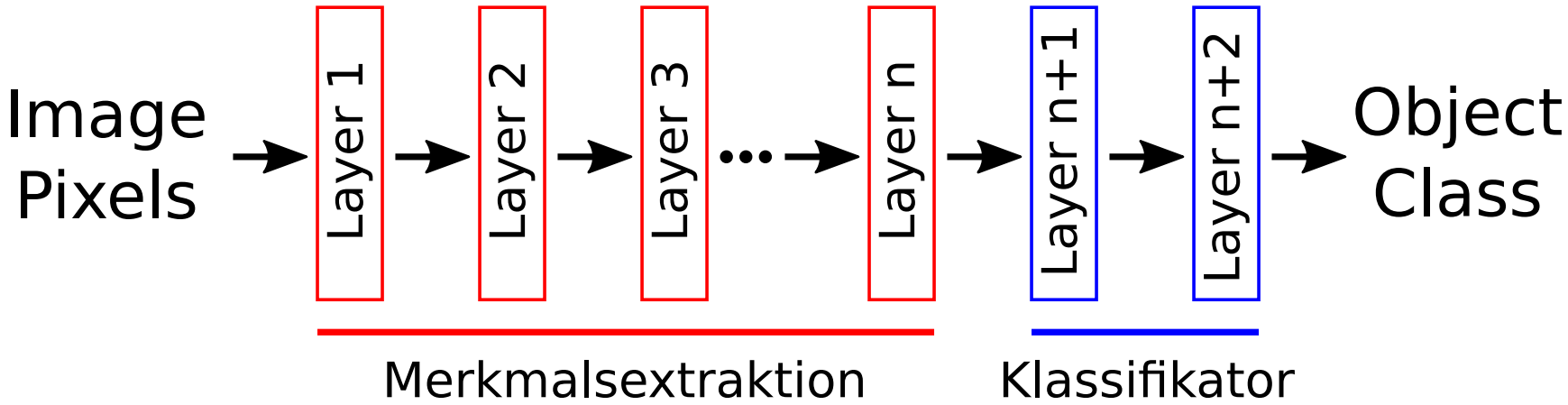
Hierarchisch:

- Auto: Karosserie, Räder
- Räder: Kreise
- Kreise: Kantenzüge
- Kantenzüge: Kanten

Deep Learning

Wir wollen die Merkmale nicht mehr selber bauen!

- Merkmale lassen sich hierarchisch aufbauen
- jeder Layer lernt bestimmte Merkmale aus vorherigen Merkmalen
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,.... *n*-ter lernt Autos



Das Bild als Eingabe

Rechenbeispiel

- RGB-Bild der Größe 600×400
- erster Layer mit 64 Neuronen

Wie viele Gewichte lernt man nur für den ersten Layer?

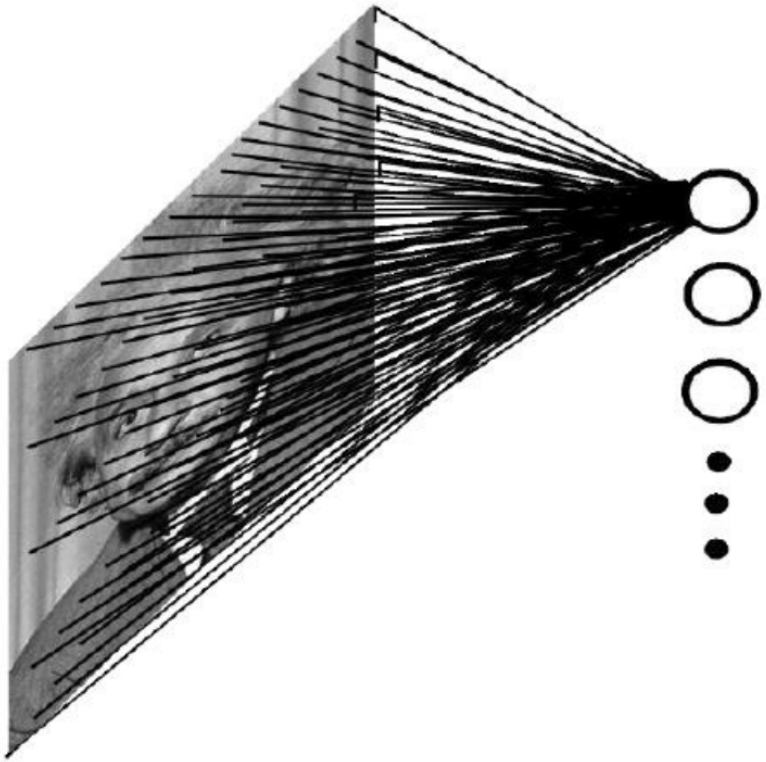
$$600 \cdot 400 \cdot 3 \cdot 64 + 64 = \mathbf{46.080.064}$$

Warum macht es auch inhaltlich keinen Sinn?

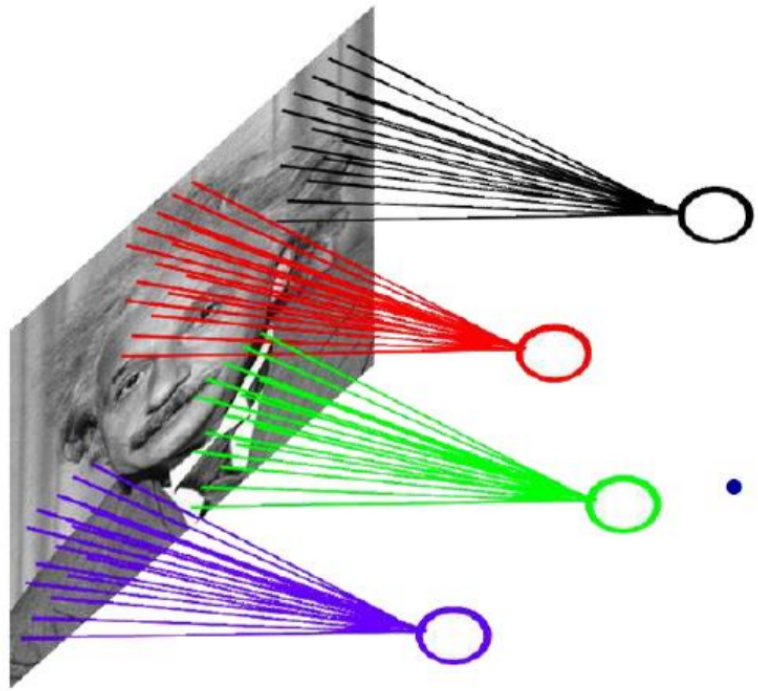
Merkmale sind...

- ... positionsunabhängig
- ... lokal

Vom NN zum CNN

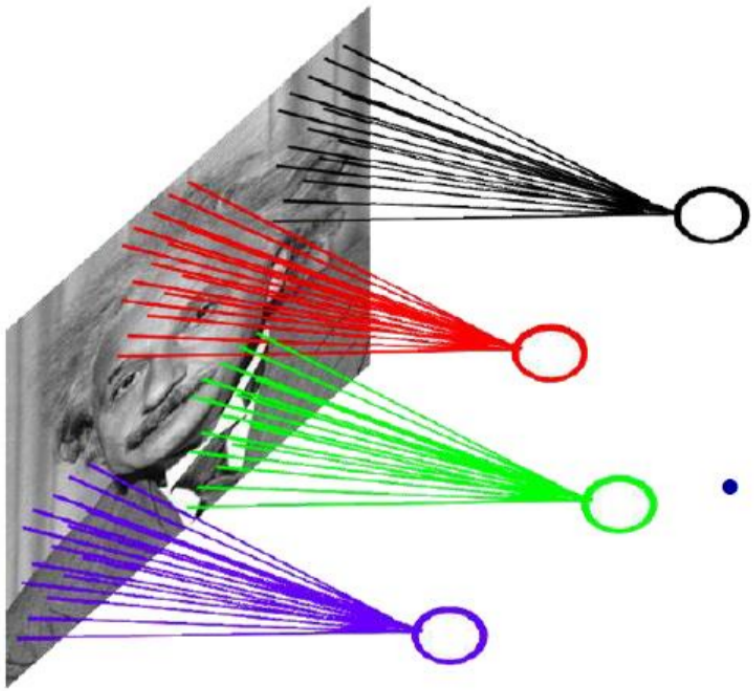


Neuronales Netz



Convolutional Neural Network (CNN)

Convolutional Neural Network



Convolutional Neural Network (CNN)

Basisprinzipien

- Neuron sieht nur wenige Pixel
- Neuron wird über das Bild *gezogen*

Neuron lernt einen Filter:

$$w = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix}$$

$\Rightarrow$ Faltung von Bild und Filter

Einschub: Faltung eines Bildes mit einem Filter - I

Prinzip

- Filter wird auf jedes Pixel zentriert
- überlappende Werte werden multipliziert und addiert
- Ergebnis wird in neues Bild geschrieben

Beispielfilter

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Weichzeichnen

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Kantendetektion (vertikal)

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Kantendetektion (horizontal)

Einschub: Faltung eines Bildes mit einem Filter - II

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0.5	1	1	1	0.5	0	0	0
0	0	0	0.25	0.5	1	0.5	0.25	0	0	0
0	0	0	0.1	0.25	0.5	0.25	0.1	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Einschub: Faltung eines Bildes mit einem Filter - II

0 •1	0 •0	0 •-1	0	0	0	0	0	0	0	0
0 •2	0 •0	0 •-2	0	0	0	0	0	0	0	0
0 •1	0 •0	0 •-1	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0.5	1	1	1	0.5	0	0	0
0	0	0	0.25	0.5	1	0.5	0.25	0	0	0
0	0	0	0.1	0.25	0.5	0.25	0.1	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

	0									

$$1 \cdot 0 + 2 \cdot 0 + 1 \cdot 0 + (-1) \cdot 0 + (-2) \cdot 0 + (-1) \cdot 0 = 0$$

Einschub: Faltung eines Bildes mit einem Filter - II

0	0 •1	0 •0	0 •-1	0	0	0	0	0	0	0
0	0 •2	0 •0	0 •-2	0	0	0	0	0	0	0
0	0 •1	0 •0	0 •-1	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0.5	1	1	1	0.5	0	0	0
0	0	0	0.25	0.5	1	0.5	0.25	0	0	0
0	0	0	0.1	0.25	0.5	0.25	0.1	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

	0	0								

$$1 \cdot 0 + 2 \cdot 0 + 1 \cdot 0 + (-1) \cdot 0 + (-2) \cdot 0 + (-1) \cdot 0 = 0$$

Einschub: Faltung eines Bildes mit einem Filter - II

0	0	0	0	0	0	0	0	0	0	0
0	0 •1	0 •0	0 •-1	0	0	0	0	0	0	0
0	0 •2	0 •0	0 •-2	0	0	0	0	0	0	0
0	0 •1	0 •0	1 •-1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0.5	1	1	1	0.5	0	0	0
0	0	0	0.25	0.5	1	0.5	0.25	0	0	0
0	0	0	0.1	0.25	0.5	0.25	0.1	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

	0	0	0	0	0	0	0	0	0	
	0	-1								

$$1 \cdot 0 + 2 \cdot 0 + 1 \cdot 0 + (-1) \cdot 0 + (-2) \cdot 0 + (-1) \cdot 1 = -1$$

Einschub: Faltung eines Bildes mit einem Filter - II

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0 •1	0 •0	0 •-1	0	0	0	0	0
0	0	0	0 •2	0 •0	0 •-2	0	0	0	0	0
0	0	0	1 •1	1 •0	1 •-1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0.5	1	1	1	0.5	0	0	0
0	0	0	0.25	0.5	1	0.5	0.25	0	0	0
0	0	0	0.1	0.25	0.5	0.25	0.1	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

	0	0	0	0	0	0	0	0	0	
	0	-1	-1	0						

$$1 \cdot 0 + 2 \cdot 0 + 1 \cdot 1 + (-1) \cdot 0 + (-2) \cdot 0 + (-1) \cdot 1 = 0$$

Einschub: Faltung eines Bildes mit einem Filter - II

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0.5	1	1	1	0.5 •1	0 •0	0 •-1	0
0	0	0	0.25	0.5	1	0.5	0.25 •2	0 •0	0 •-2	0
0	0	0	0.1	0.25	0.5	0.25	0.1 •1	1 •0	1 •-1	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

	0	0	0	0	0	0	0	0	0	
	0	-1	-1	0	0	0	1	1	0	
	0	-3	-3	0	0	0	3	3	0	
	0	-3.5	-4	-0.5	0	0.5	4	3.5	0	
	0	-2.25	-3.5	-1.75	0	1.75	3.5	2.25	0	
	0	-1.1	-2.25	-2.4	0	2.4	2.25	1.1		

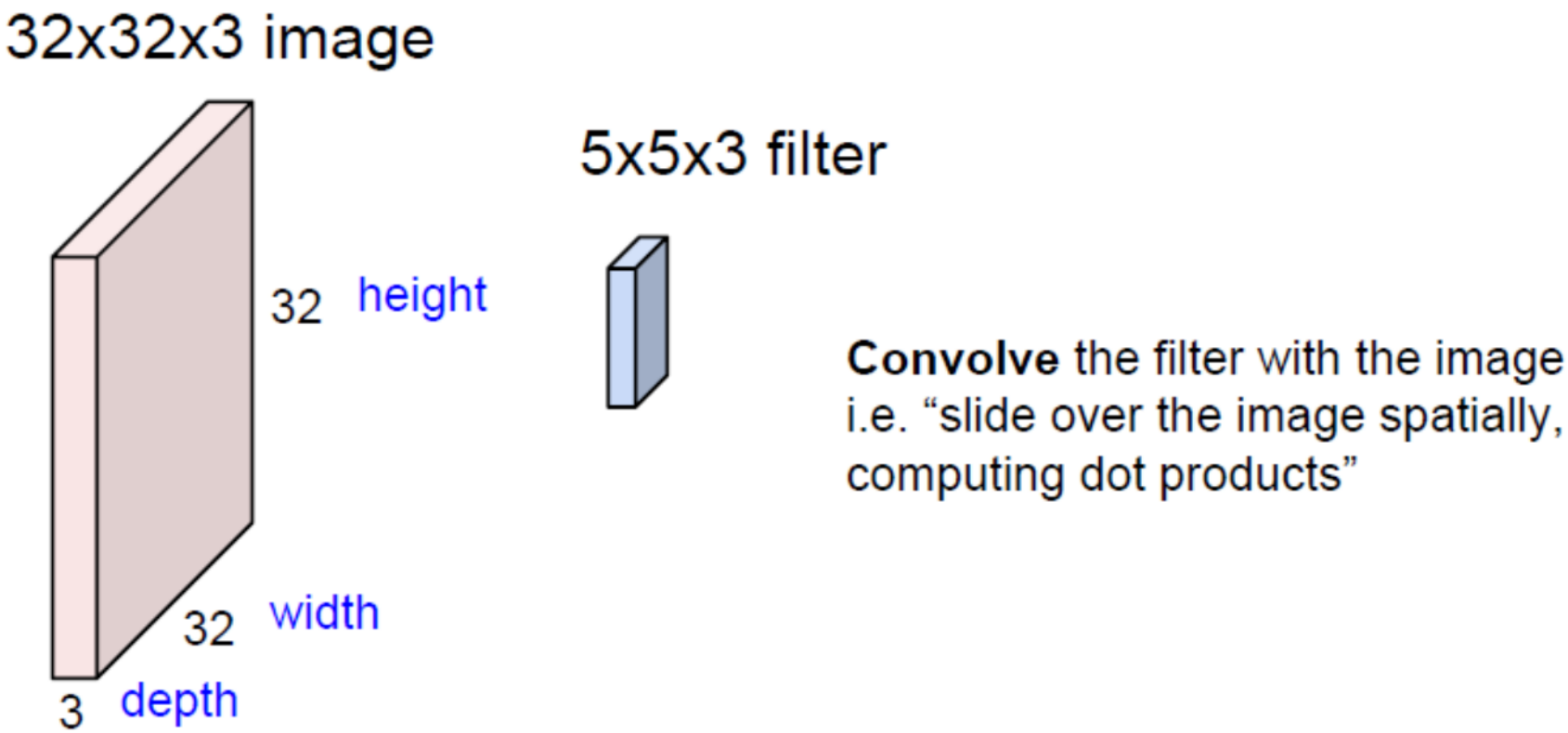
$$1 \cdot 0.5 + 2 \cdot 0.25 + 1 \cdot 0.1 + (-1) \cdot 0 + (-2) \cdot 0 + (-1) \cdot 1 = 1.1$$

Einschub: Faltung eines Bildes mit einem Filter - II

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0	0	0
0	0	0	0.5	1	1	1	0.5	0	0	0
0	0	0	0.25	0.5	1	0.5	0.25	0	0	0
0	0	0	0.1	0.25	0.5	0.25	0.1	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

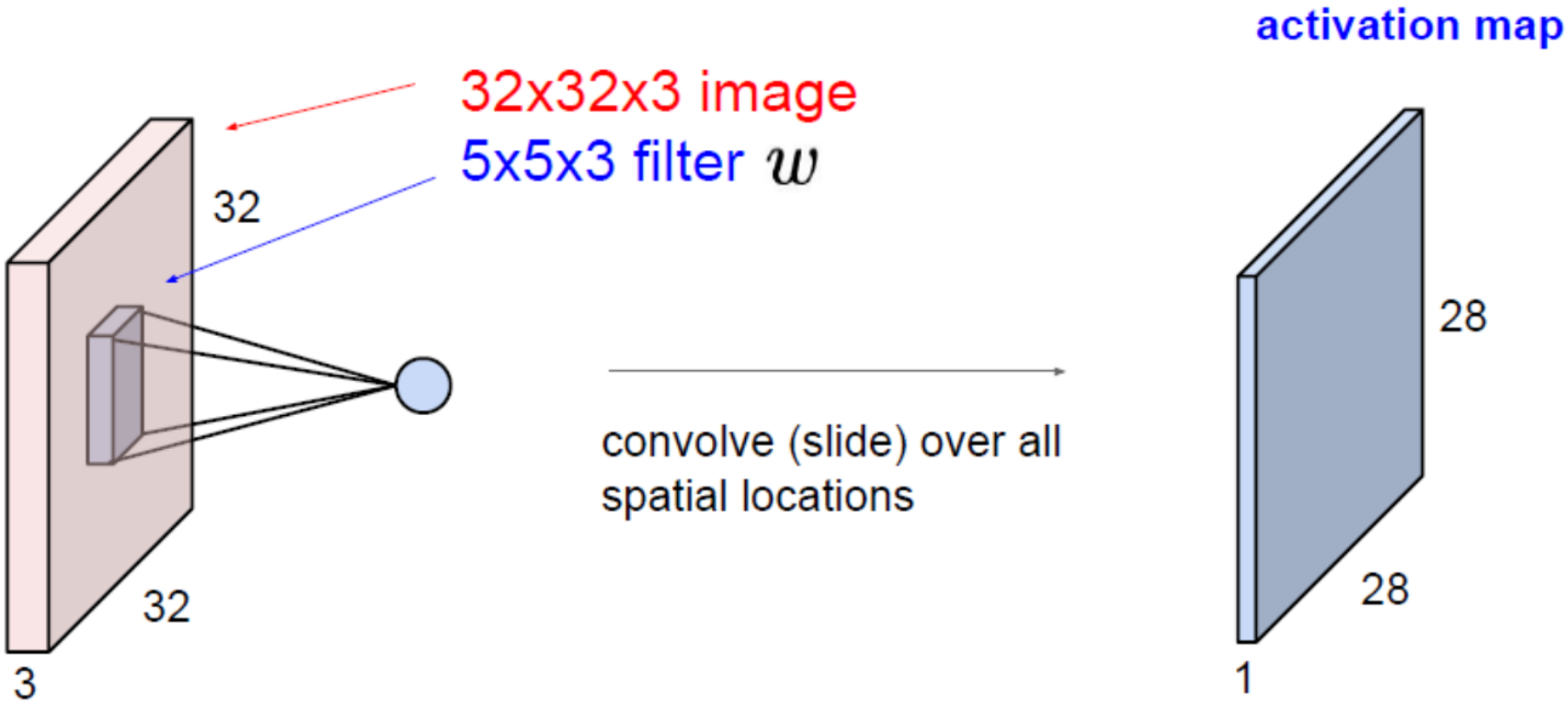
	0	0	0	0	0	0	0	0	0	
	0	-1	-1	0	0	0	1	1	0	
	0	-3	-3	0	0	0	3	3	0	
	0	-3.5	-4	-0.5	0	0.5	4	3.5	0	
	0	-2.25	-3.5	-1.75	0	1.75	3.5	2.25	0	
	0	-1.1	-2.25	-2.4	0	2.4	2.25	1.1	0	
	0	-0.45	-1	-1.55	0	1.55	1	0.45	0	
	0	-0.1	-0.25	-0.4	0	0.4	0.25	0.1	0	
	0	0	0	0	0	0	0	0	0	

Convolutional Layer - I



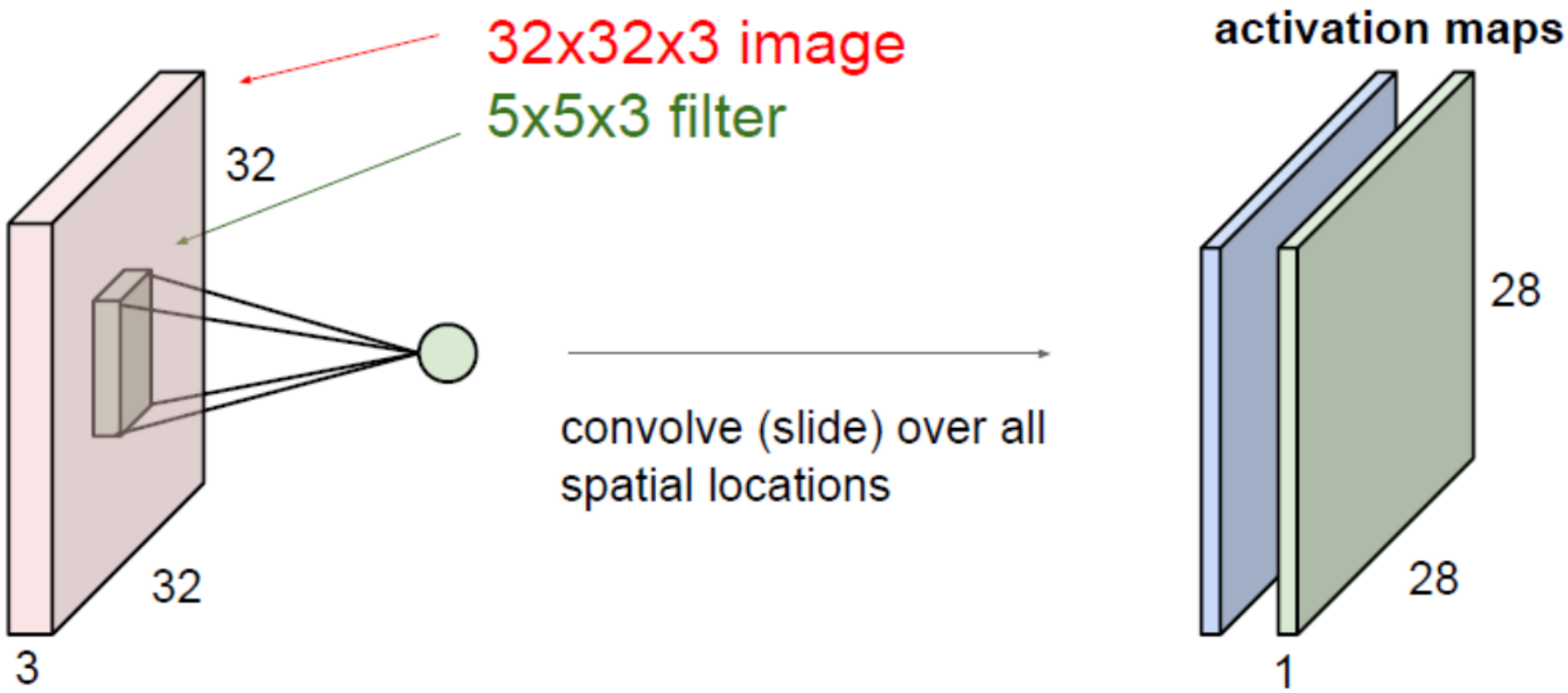
Wir falten das RGB-Bild mit einem 3D-Filter (hier 5×5, aber oft 3×3)

Convolutional Layer - II



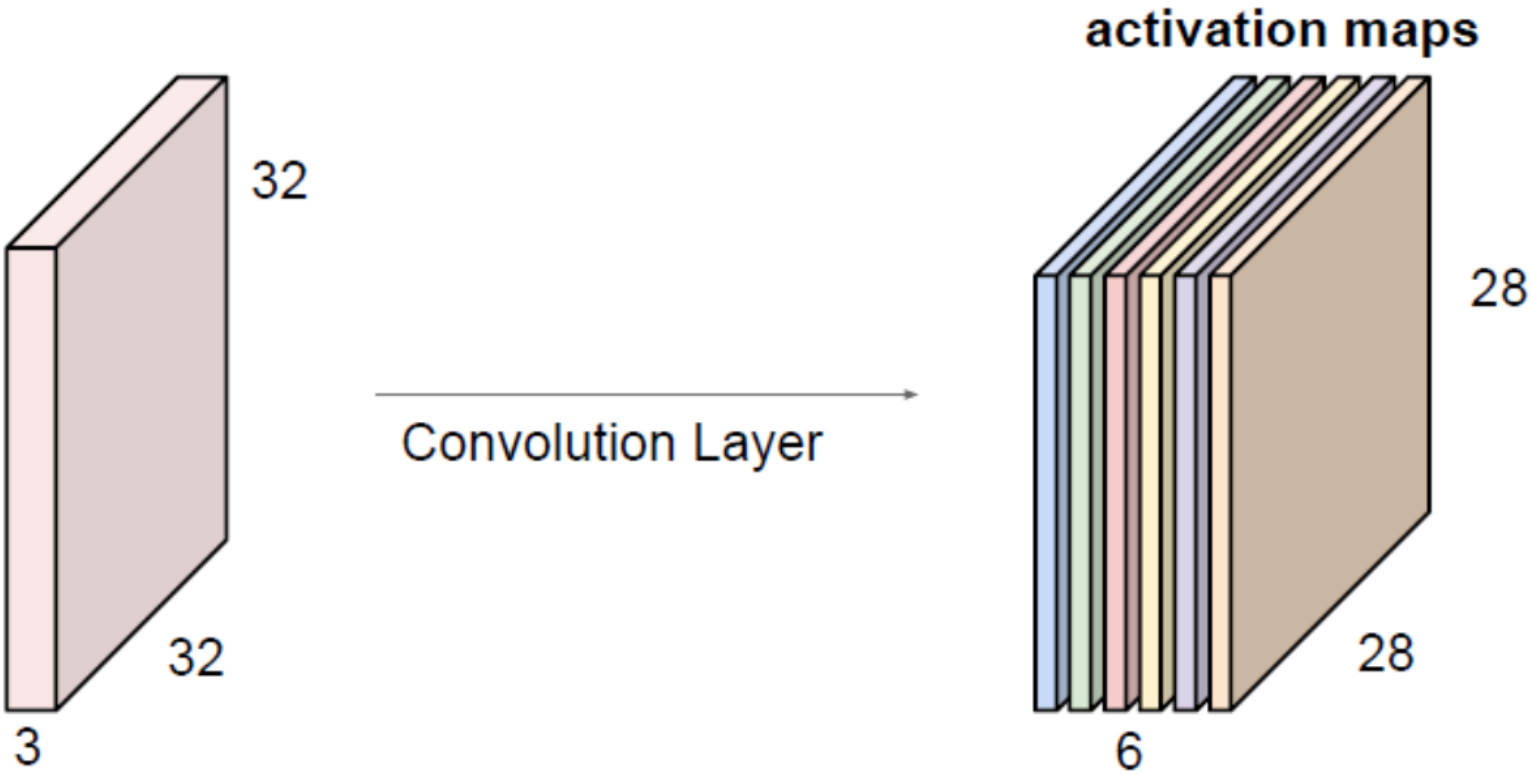
Das Ergebnis ist ein "Graustufenbild", das als Merkmalskarte interpretiert werden kann

Convolutional Layer - III



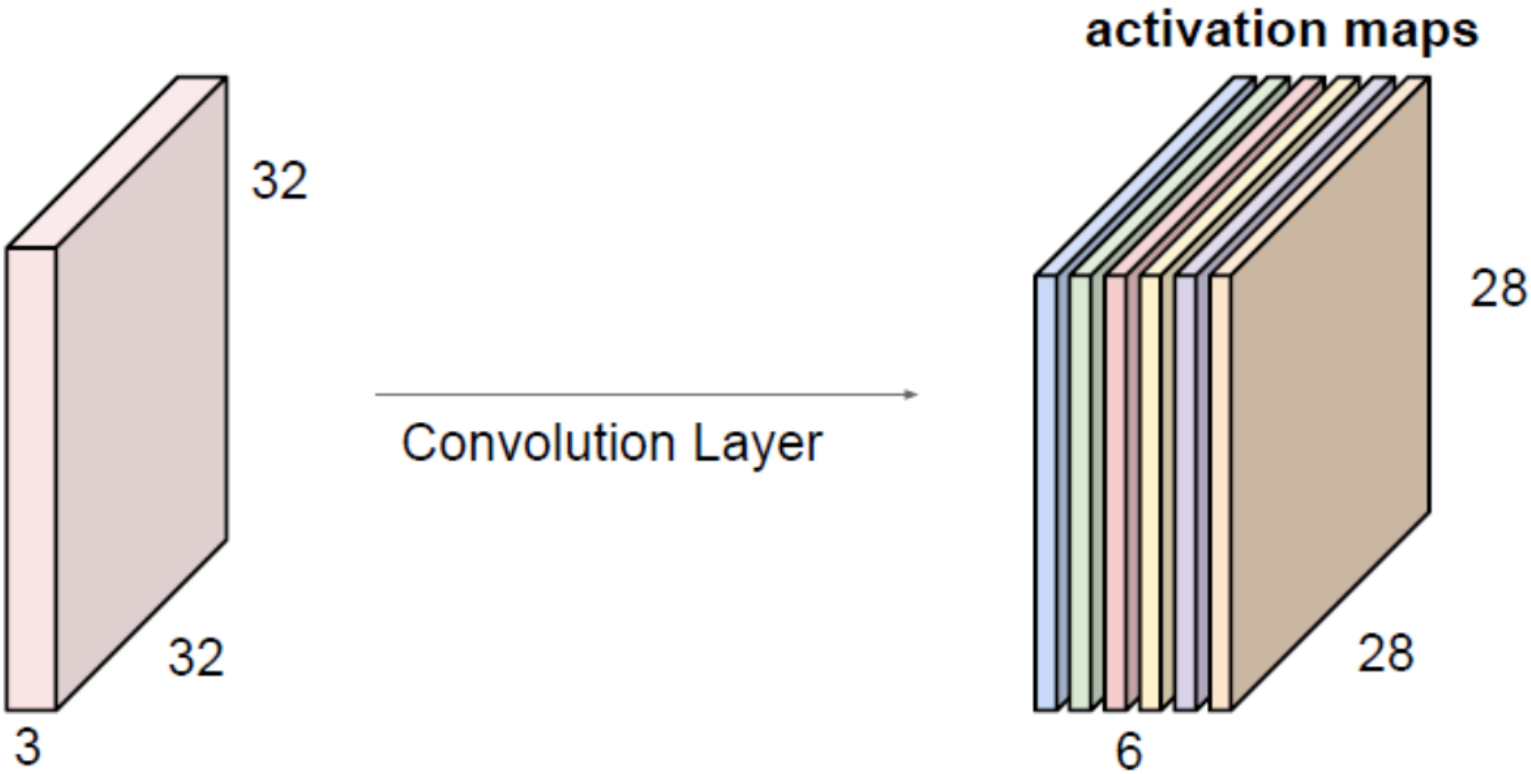
Dies wird nicht nur einmal gemacht...

Convolutional Layer - IV



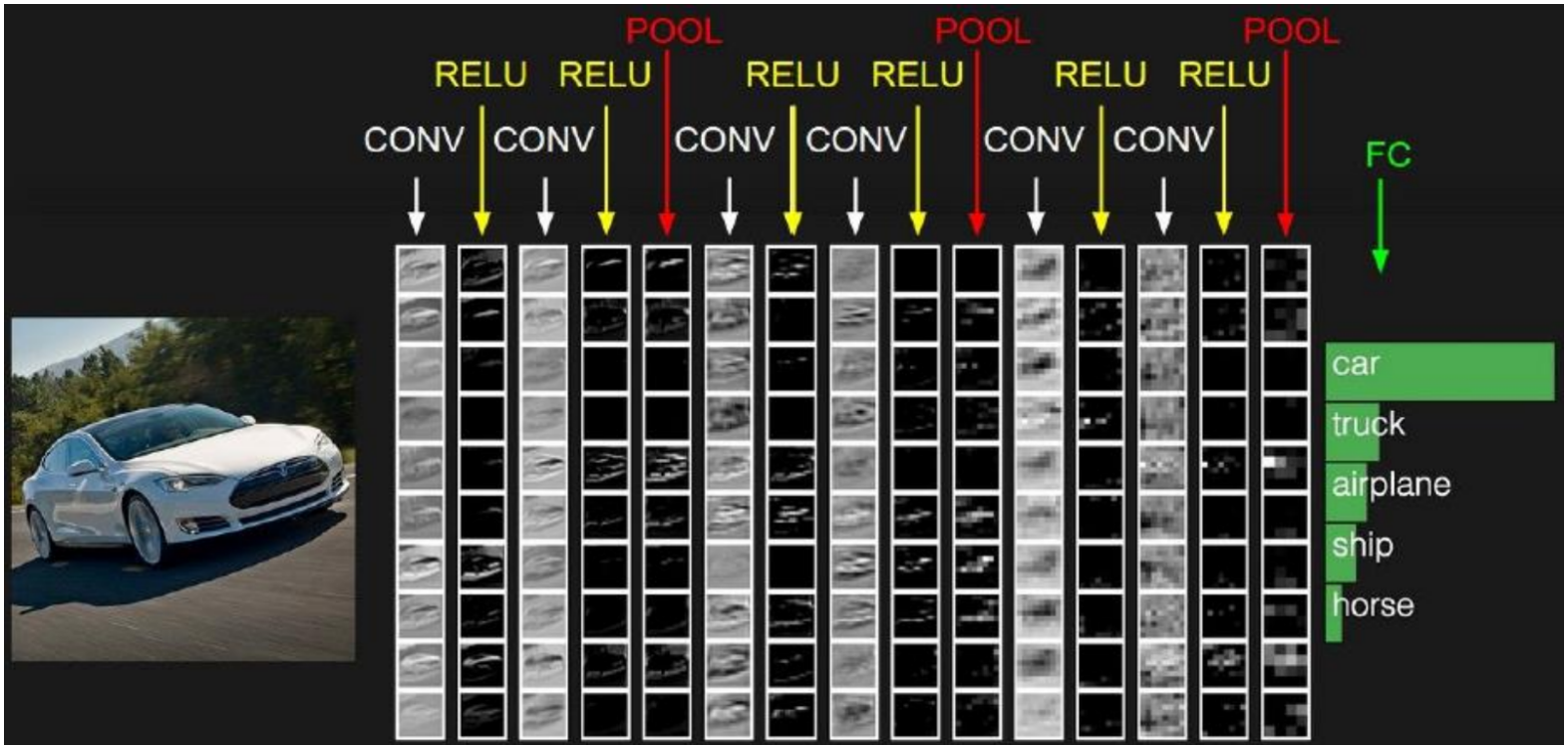
sondern oft. Es entsteht somit ein neues "Bild" mit vielen Kanälen für verschiedene Merkmale.

Convolutional Layer - V



Im Training werden die Gewichte in den Filtern gelernt, um unterschiedliche Merkmale zu erkennen.

CNN-Struktur



Conv. Layer hintereinander ausführen → Hierarchie von Merkmalen

Was ist mit den anderen Layern: RELU, POOL und FC?

Weitere Layer

ReLU

- ReLU (Rectified Linear Unit) dient als Aktivierungsfunktion
- erzeugt also die Nichtlinearität

Pooling

- Reduzierung der Auflösung einer Merkmalskarte
- meist Maximum je $n \times n$ Kachel einer Merkmalskarte
- reduziert die Anzahl an Berechnungen

Fully-Connected (FC)

- klassische Layer eines Neuronalen Netzes (`Dense`)
- letzte Merkmalskarte wird abgerollt und klassifiziert

Training

Ein CNN trainiert sich wie ein normales Neuronales Netz.

Ablauf

1. Trainingsbilder mit Labeln bereitstellen
2. zufällige Initialisierung aller Gewichte
3. Optimierung der Gewichte in den Filtern (Conv) und an den Neuronen (FC) via Backpropagation

Vorteile

Nachteile

Training

Ein CNN trainiert sich wie ein normales Neuronales Netz.

Ablauf

Vorteile

- es müssen keine Merkmale vorgegeben werden
- Merkmalsextraktion (Conv) und Klassifikation (FC) trainieren zusammen

Nachteile

- sehr viele Daten nötig
- viel Zeit/Rechenleistung/Energie nötig

Designprinzipien

- erst Conv-Layer und ggf. Pooling-Layer im Wechsel zur Merkmalsextraktion
- danach FC-Layer zur Klassifikation
- weitere folgen nächste Woche...

Übersicht

Wie kann ich das alles nutzen?

Technische Voraussetzungen

- kleine Netze kann man auf der CPU rechnen
- große Netze muss man auf der GPU rechnen
- NVIDIA-GPU $\rightarrow$ CUDA

Bibliotheken

Wir nutzen Tensorflow und Keras wie in der letzten Woche.

CNNs in Keras

- Prinzip identisch zu normalen Neuronalen Netzen
- Convolutional Layer und Dense Layer werden in das selbe Model eingefügt
- es gibt drei neue Layer-Typen:

`Conv2D`	Convolutional Layer
`MaxPooling2D`	Pooling der Merkmalskarten
`Flatten`	Übergang zwischen Conv- und Dense Layern

Convolutional Layer in Keras

`Conv2D(filters, kernel_size, padding, input_shape)`

`filters` Anzahl der Filter

`kernel_size` Größe der Filter in Pixel

`padding` Art des Paddings

`input_shape` Form der Daten, die in den Layer kommen (nur erster Layer)

Beispiel

```
from tensorflow.keras.layers import Conv2D
model.add(Conv2D(32, (3, 3), activation='relu',
    padding='same', input_shape=(32,32,3)))
model.add(Conv2D(32, (3, 3), activation='relu',
    padding='same')) #32 Filter je 3x3
```

MaxPooling Layer in Keras

`MaxPooling2D(pool_size)`

`pool_size` Größe der Kachel zum Pooling

Beispiel

```
from tensorflow.keras.layers import MaxPooling2D
model.add(Conv2D(32, (3, 3), activation='relu',
    padding='same', input_shape=(32,32,3)))
model.add(Conv2D(32, (3, 3), activation='relu',
    padding='same'))
model.add(MaxPooling2D((2,2))) #Ergebnis hat Shape
    (16,16,32)
```

Flatten Layer in Keras

`Flatten()`

Beispiel

```
from tensorflow.keras.layers import Flatten
model.add(Conv2D(32, (3, 3), activation='relu',
    padding='same', input_shape=(32,32,3))) #
    (32,32,32)
model.add(Conv2D(32, (3, 3), activation='relu',
    padding='same')) #(32,32,32)
model.add(MaxPooling2D((2,2))) #(16,16,32)
model.add(Flatten()) #(8192,)
model.add(Dense(128, activation='relu')) #128
```

Wie man richtig lernt

Aufteilung der Daten

Trainingsdaten nutzen, um die Gewichte des Modells zu trainieren.

Validierungsdaten nutzen, um die Hyperparameter zu bestimmen.

Testdaten nutzen, um die Endergebnisse für den Bericht zu erzeugen.

Was sind Hyperparameter?

- Anzahl der Layer
- Anzahl der Filter/Neuronen je Layer
- Anzahl der Epochen
- Größe der Batches

Validierungsdaten verbessern die Generalisierbarkeit

Validierungsdaten erzeugen

Validierungsdaten werden meist von den Trainingsdaten abgezweigt.

```
model.fit(X_train, Y_train, batch_size=32, epochs=10,
          validation_split = 0.2)
```

`validation_split` Anteil der zur Validierung genutzten Trainingsdaten

oder

```
model.fit(X_train, Y_train, batch_size=32, epochs=10,
          validation_data = (X_val,Y_val))
```

`validation_data` Validierungsdaten als Tuple aus `(X_val, Y_val)`

Training flexibilisieren

Problem

Die Anzahl der Epochen ist a priori nicht bestimmbar.

Lösung

- Training wird beendet, wenn sich der Validation Loss in n aufeinander folgenden Epochen nicht verbessert
- Implementation in Keras als EarlyStopping Callback
- Callbacks werden der `fit`-Methode übergeben

EarlyStopping Callback in Keras

`EarlyStopping(patience)`

`patience` Anzahl Epochen ohne Verbesserung bis abgebrochen wird

Rückgabe ist das Callback-Objekt für die `fit`-Methode

Beispiel

```
from tensorflow.keras.callbacks import EarlyStopping
es=EarlyStopping(patience=3)
model.fit(X_train, Y_train, batch_size=32, epochs=10,
          validation_data = (X_val,Y_val),
          callbacks=[es])
```

Speichern eines trainierten Modells per Callback

`ModelCheckpoint(filepath, save_best_only)`

`filepath` Dateipfad für das trainierte Modell

`save_best_only` nur die besten Gewichte speichern

Vor Evaluation (`model.evaluate()`) Gewichte mit `model.load_weights(filepath)` laden.

Beispiel

```
from tensorflow.keras.callbacks import
    ModelCheckpoint
mc=ModelCheckpoint(filepath='bestModel.h5',
    save_best_only=True)
model.fit(X_train, Y_train, batch_size=32, epochs=10,
          validation_data = (X_val,Y_val),
          callbacks=[es,mc])
```

Übersicht

CNNs

- [GW]: Kapitel 12.6 Deep Convolutional Neural Networks
  - A Basic CNN Architecture
- Erklärung: Convolutional Neural Networks (CNNs / ConvNets)
- Video: Convolutional Networks - Udacity @ YouTube
- Video: Convolutional Neural Networks (CNNs) explained - deeplizard @ YouTube
- Video: Max Pooling in Convolutional Neural Networks explained - deeplizard @ YouTube

Training von CNNs

- [GW]: Kapitel 12.6 Deep Convolutional Neural Networks
  - The Equations of Backpropagation used to Train CNNs
    - Example 12.16
    - Example 12.17
    - Example 12.18
- [GB]: Kapitel 7 Regularization for Deep Learning
  - 7.8 Early Stopping
- Erklärung: Early Stopping - but when? von Lutz Prechelt
  Anmerkung: Vor allem Kapitel 1 ist gut, um den Grund für Early Stopping zu verstehen.

CNNs mit Keras

- Erklärung/Beispiel: Keras - The Sequential model
- Beispiel: Simple MNIST convnet
- Erklärung/Beispiel: Building a Convolutional Neural Network (CNN) in Keras
  Anmerkung: Benutzt eine ältere Keras-Version.
- Erklärung/Beispiel: How to Check-Point Deep Learning Models in Keras

Referenzen I


[GW], R. Gonzalez und R. Woods
Digital Image Processing
4th ed., Global Edition, Pearson, 2018.
siehe Moodle


[GB], I. Goodfellow, Y. Bengio und A. Courville
Deep Learning
1st ed., MIT Press, 2016.
Bib-Katalog
E-Book