

Convolutional Neural Networks

Christian Wilms

Computer Vision Group
Universität Hamburg

Sommersemester 2018

31. Mai 2018

Übersicht

1 Motivation

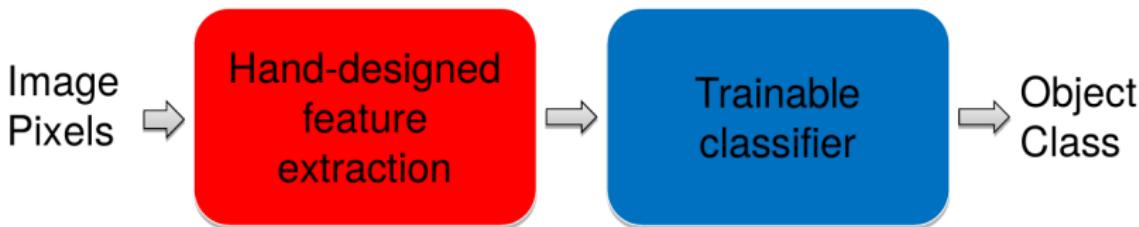
2 Convolutional Neural Networks

3 CNNs mit GPUs

4 Literatur

Pipeline der Klassifikation

Klassifikation wie wir sie bisher gemacht haben...



- Mittelwert
 - Standardabweichung
 - Histogramme
 - Seitenverhältnis
 - HOG
 - ...
-
- Nearest Neighbour Klassifikator
 - *k*-Nearest Neighbour Klassifikator
 - Multi Layer Perceptron

Merkmale

lokal

- Statistiken v. Obj.
- Seitenverhältnis
- Fläche

global

- Bildstatistiken
- Histogramme
- HOG

Was wir bisher nicht aktiv betrachtet haben

Hierarchien von Merkmalen:

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

Übersicht

1 Motivation

2 Convolutional Neural Networks

3 CNNs mit GPUs

4 Literatur

Deep Learning

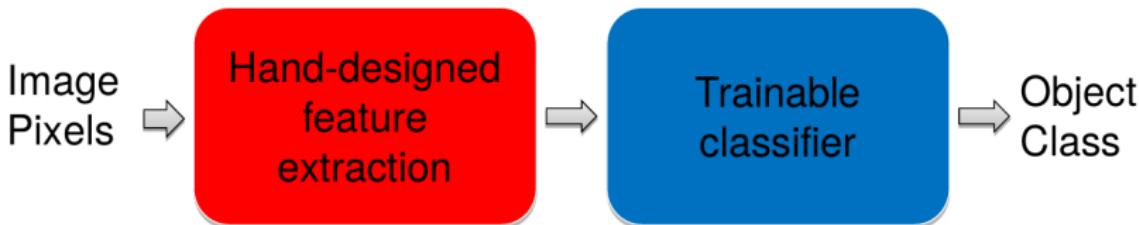
Wir wollen die Merkmale nicht mehr selber bauen!

- Merkmale lassen sich hierarchisch aufbauen wie die Netze
- jeder Layer könnte bestimmte Merkmale aus den vorherigen Layern kombinieren
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,... n -ter lernt Autos

Deep Learning

Wir wollen die Merkmale nicht mehr selber bauen!

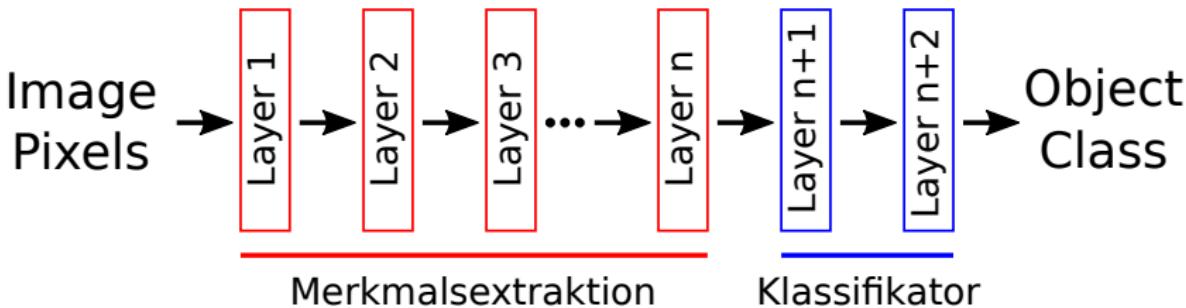
- Merkmale lassen sich hierarchisch aufbauen wie die Netze
- jeder Layer könnte bestimmte Merkmale aus den vorherigen Layern kombinieren
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,..., n -ter lernt Autos



Deep Learning

Wir wollen die Merkmale nicht mehr selber bauen!

- Merkmale lassen sich hierarchisch aufbauen wie die Netze
- jeder Layer könnte bestimmte Merkmale aus den vorherigen Layern kombinieren
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,..., n -ter lernt Autos



Deep Learning

Wir wollen die Merkmale nicht mehr selber bauen!

- Merkmale lassen sich hierarchisch aufbauen wie die Netze
- jeder Layer könnte bestimmte Merkmale aus den vorherigen Layern kombinieren
- erster Layer lernt Kanten, zweiter Kantenzüge, dritter Kreise,..., n -ter lernt Autos

Das Bild als Eingabe

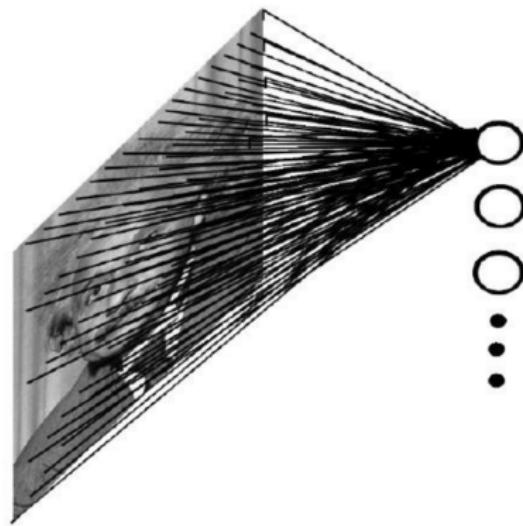
- Bild-Größe 224×224
- 3 Hidden-Layer mit 1024 Neuronen
- 10 Output-Neuronen für 10 Klassen

⇒ über 53 Millionen Parameter ohne große Tiefe oder Breite

Convolutional Neural Networks (CNN)

Beobachtungen

- die meisten Gewichte sind zwischen dem ersten Hidden Layer und dem Bild zu finden (ca. 51 Millionen)
- für einfache Merkmale braucht man nur einen sehr begrenzten, lokalen Bereich

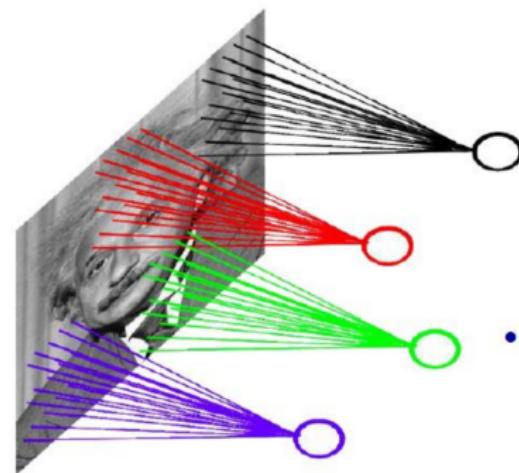


Convolutional Neural Networks (CNN)

Beobachtungen

- die meisten Gewichte sind zwischen dem ersten Hidden Layer und dem Bild zu finden (ca. 51 Millionen)
- für einfache Merkmale braucht man nur einen sehr begrenzten, lokalen Bereich

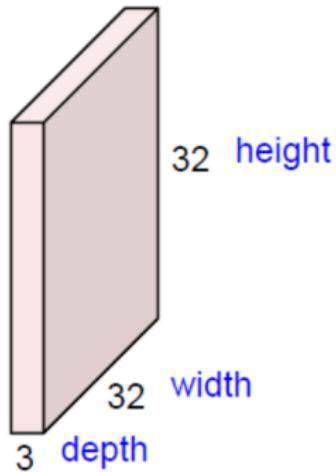
Wir benutzen Faltungen und lernen die Gewichte in den Faltungskernen.



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

Convolutional Layer

32x32x3 image



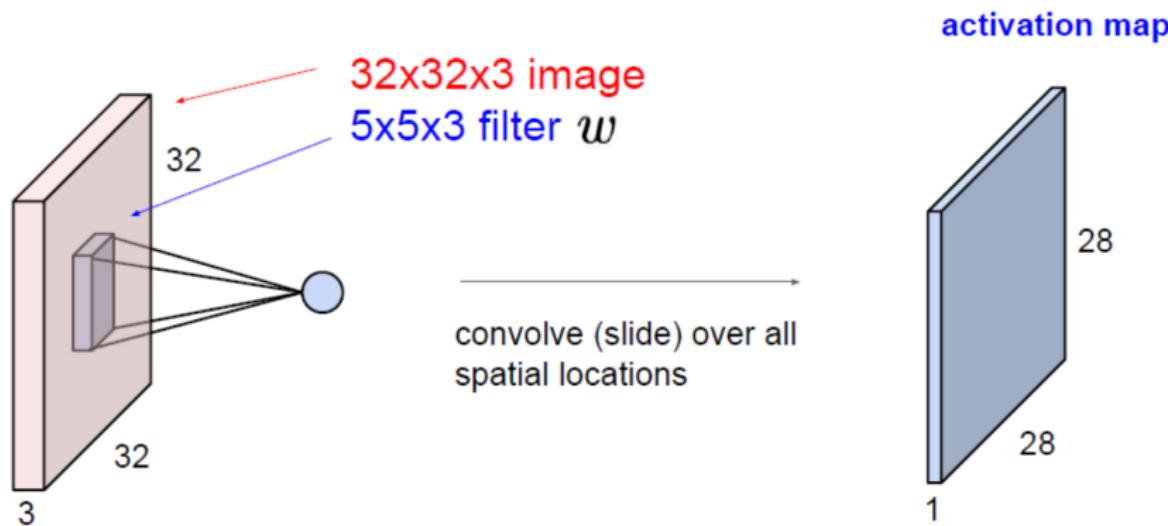
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

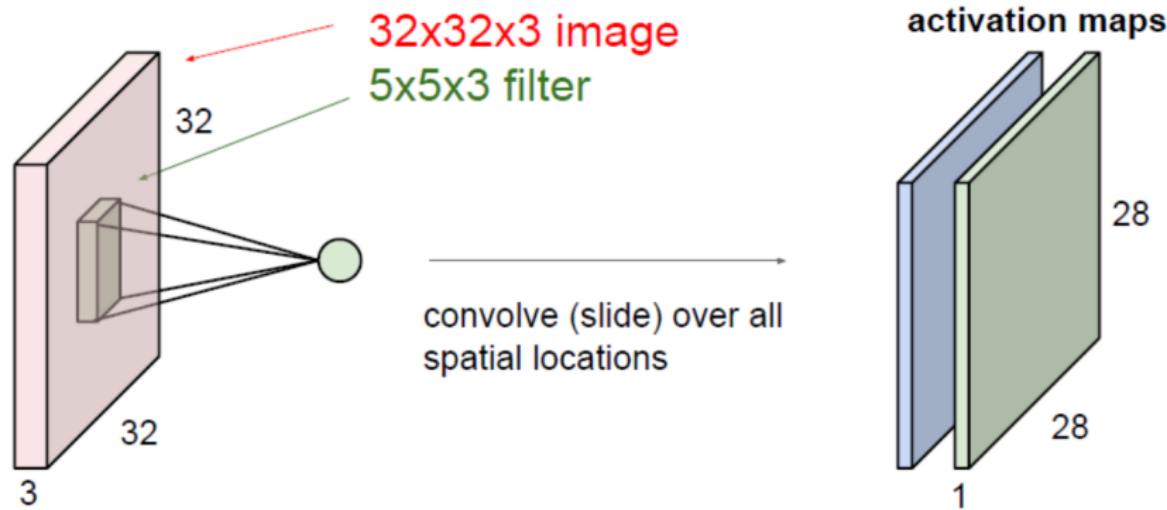
Wir falten das RGB-Bild mit einem 3D-Faltungskern (hier 5×5 ,
aber oft 3×3)

Convolutional Layer



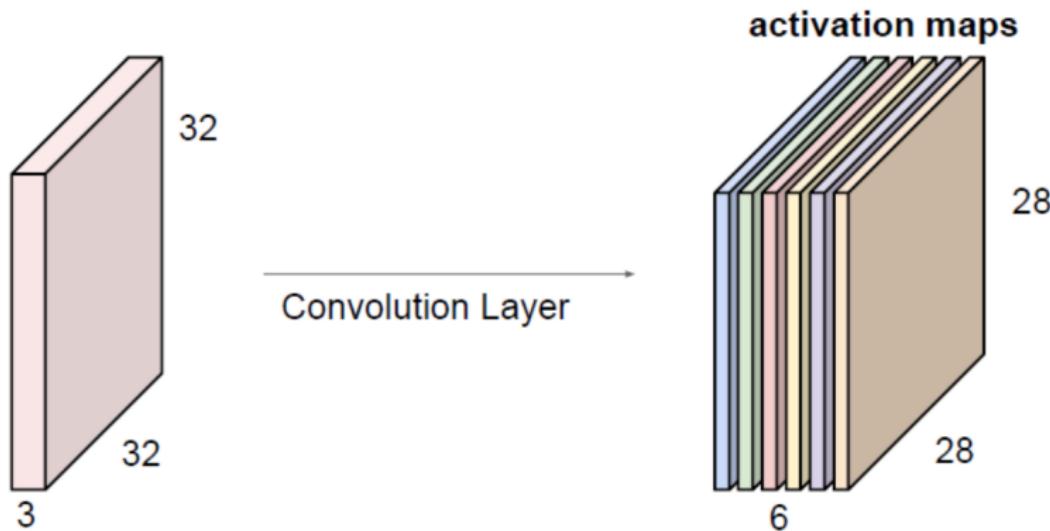
Das Ergebnis ist ein "Graustufenbild", das als Merkmalskarte interpretiert werden kann

Convolutional Layer



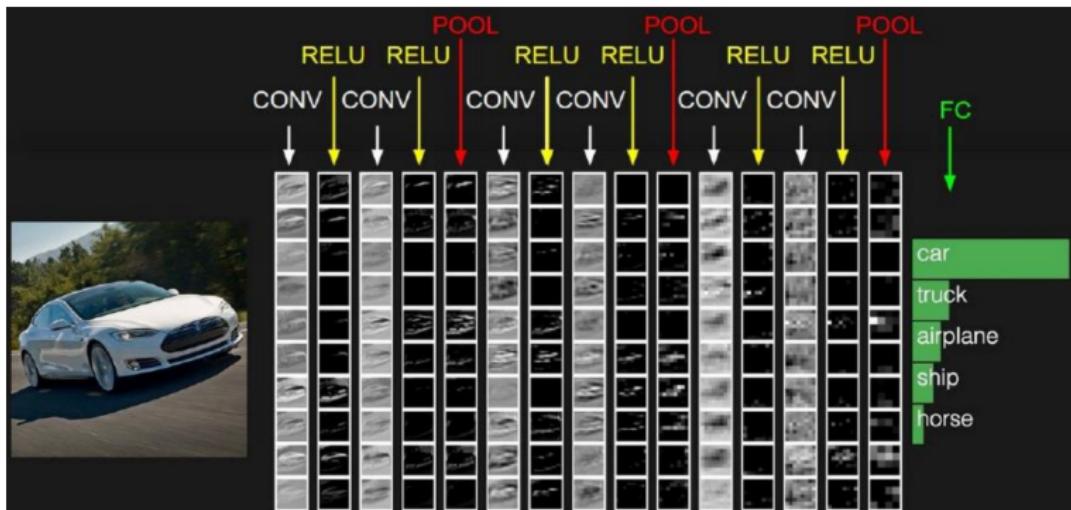
Dies wird nicht nur einmal gemacht...

Convolutional Layer



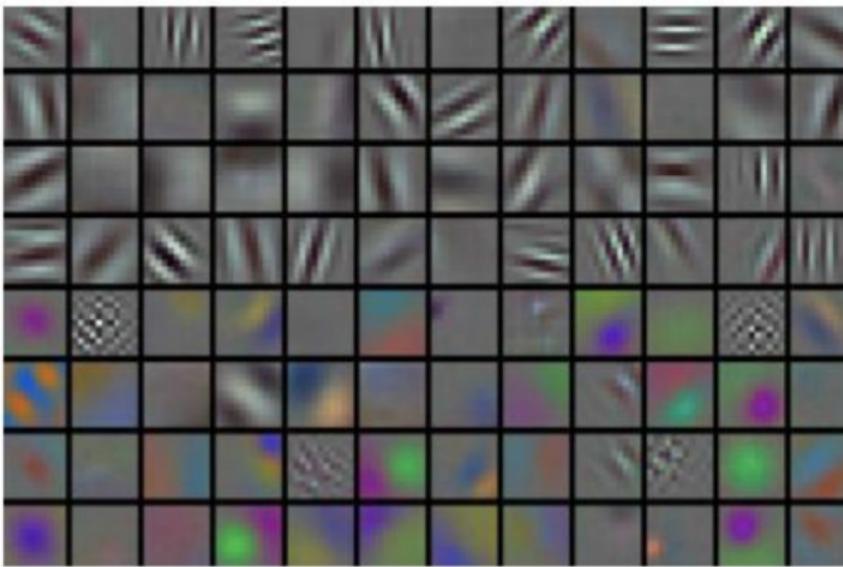
sondern oft. Es entsteht somit ein neues "Bild" mit vielen Kanälen für verschiedene Merkmale.

CNN-Struktur



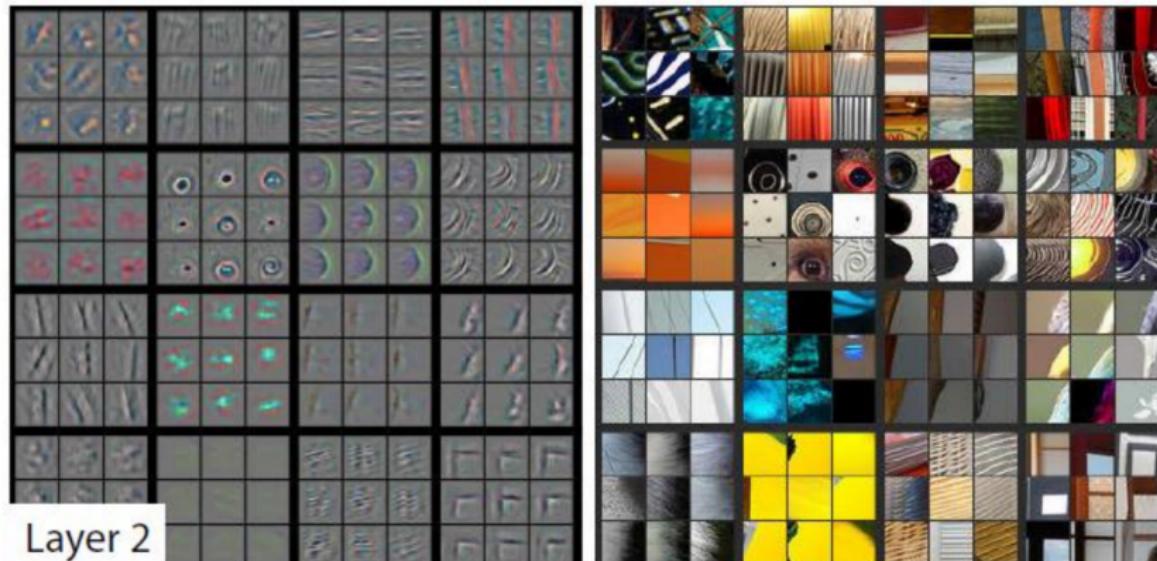
Prinzip CNN: Convolutional Layer werden aufeinander gestapelt/angewendet → es entsteht eine Hierarchie von Merkmalen

Welche Merkmale werden gelernt?



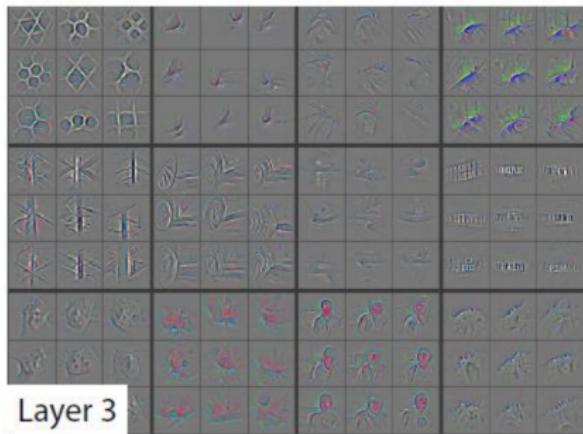
Erkannt werden Kontraste, Kanten und Wellen.

Welche Merkmale werden gelernt?



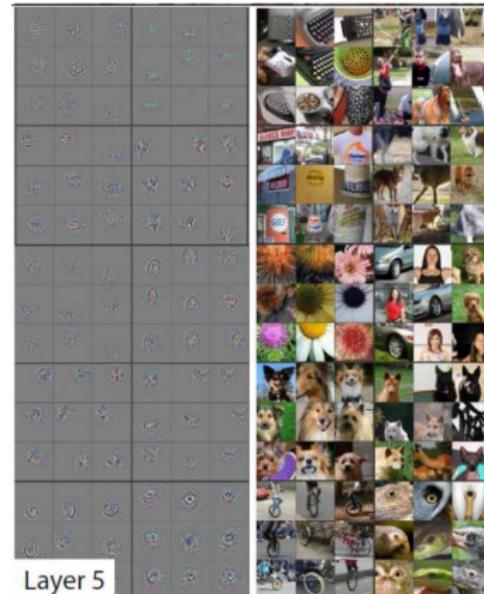
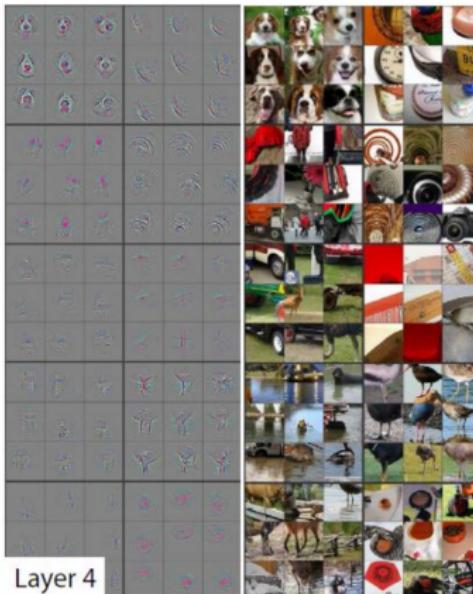
Erkannt werden Formen und homogene Flächen.

Welche Merkmale werden gelernt?



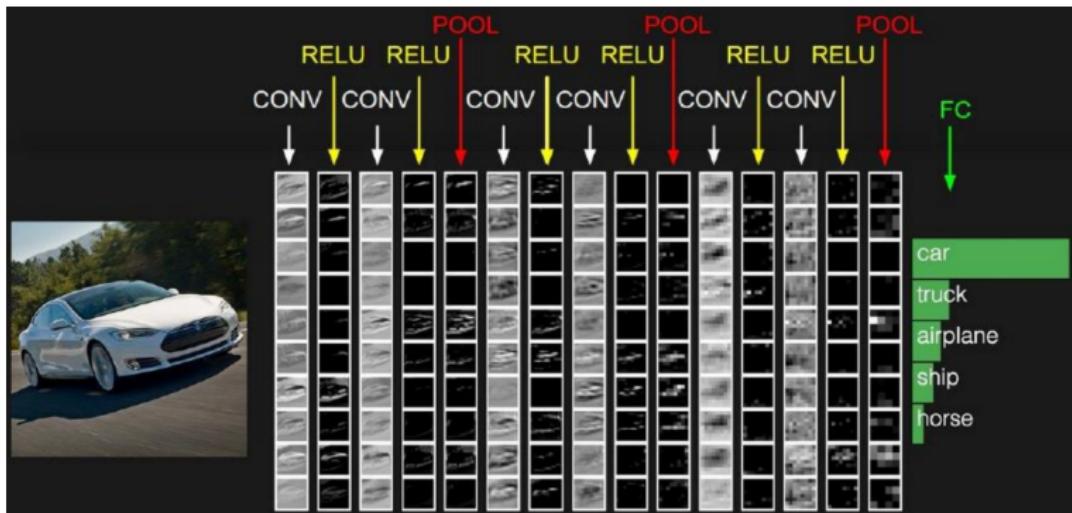
Erkannt werden Texturen und Objektteile.

Welche Merkmale werden gelernt?



Erkannt werden Objekte und Objektteile.

CNN-Struktur



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

Weitere Layer

ReLU

Pooling

Fully-Connected (FC)

Weitere Layer

ReLU

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

Pooling

Fully-Connected (FC)

Weitere Layer

ReLU

Pooling

- beim Pooling wird aus jeweils $n \times n$, meist 2×2 "Pixeln", das Maximum (Max-Pooling) genommen
- diese $n \times n$ Fenster überlappen sich nicht
- wird auf jedem Kanal individuell angewendet
- sorgt für eine Verkleinerung bei zunehmender Tiefe
- reduziert die Anzahl an Berechnungen
- es ist für die Klassifikation auch nicht entscheidend, wo das Auto im Bild lokalisiert ist

Fully-Connected (FC)

Weitere Layer

ReLU

Pooling

Fully-Connected (FC)

- nach dem letzten Conv-Layer mit ReLU und Pooling wird das Ergebnisbild abgerollt
- die "Pixel"/Merkmale dienen nun als Eingabe in ein normales Neuronales Netz
- hier ein Neuronales Netz zu benutzen hat den Vorteil, dass es weiterhin alles zusammen trainiert werden kann

Training

Ein CNN trainiert sich vom Prinzip so wie ein normales Neuronales Netz.

From Scratch

- Training wie vorher beschrieben
- nur sinnvoll, wenn der Datensatz groß ist oder das Netz klein

Fine-tuning (Transfer-Learning)

- das Netz wurde bereits auf einem (großen) Datensatz trainiert
- viele Merkmale werden vermutlich ähnlich sein
- daher werden nur die FC-Layer oder die obersten n Conv-Layer trainiert

Faustregel: Mit vielen Daten kann ich viele Layer trainieren, mit wenig Daten nur wenige.

Wenn nicht genug Daten da sind...

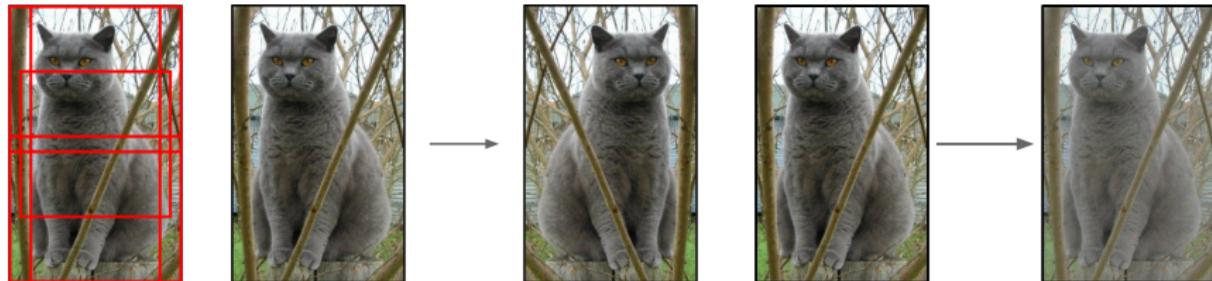
... kann man sich aus bestehenden Daten weitere erzeugen.

Data Augmentation

Prinzip: Transformationen auf das Bild anwenden, die das Label nicht ändern.

- Spiegeln, meist an der vertikalen Achse
- Vergrößern/Verkleinern und Ausschnitte nehmen
- Bild verrauschen
- color jitter (Farben minimal verändern, Kontrast verschlechtern)
- Hintergrund austauschen
- ...

Data Augmentation - Beispiel



Data Augmentation kann auch beim Testen genutzt werden,
bspw.:

- Bild auf 5 Größen skalieren (224, 256, 384, 480, 640 für die kürzere Achse)
- für jede Größe 10 224×224 Ausschnitte:
(4 Ecken + Zentrum) · Spiegelung

Wie soll mein CNN aussehen?

- für den Anfang solltet ihr mit kleinen Netzen arbeiten (z.B. 2 bis 5 Layer mit je 32 Faltungskernen)
- es gibt auch bereits eine ganze Reihe Standardnetze, die deutlich größer sind

AlexNet 5 Conv-Layer

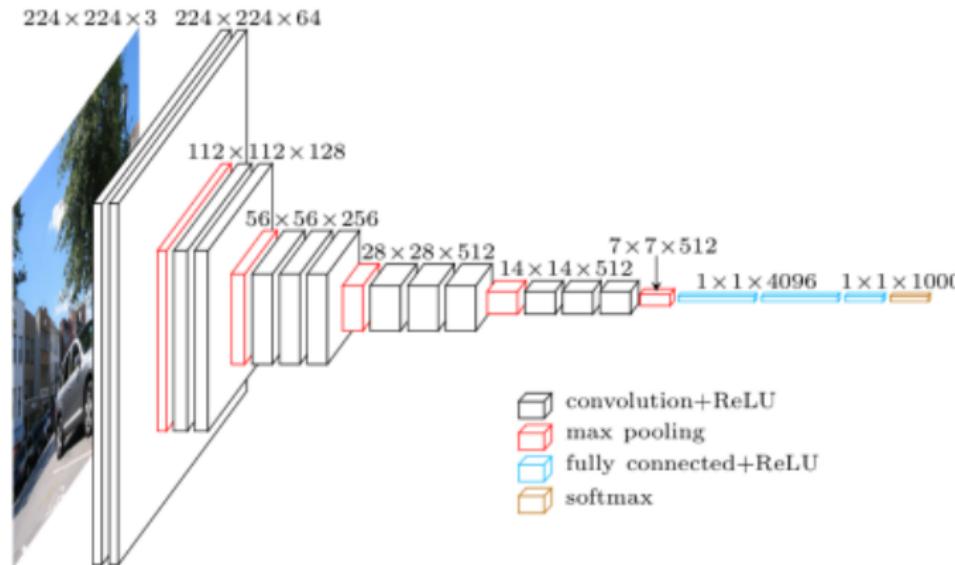
VGG 13 oder 16 Conv-Layer

GoogLeNet viele

Res-Net noch mehr

- alle diese Netze enden mit FC-Layern
- für diese Netze lassen sich auch bereits die Imagenet-Gewichte herunterladen → nur noch fine-tuning nötig

Beispiel VGG-16



Architektur von VGG-16, alle Conv-Layer sind 3×3 ¹

¹<http://www.cs.toronto.edu/~frossard/post/vgg16/vgg16.png>

Designprinzipien

- erst Conv-Layer und Pooling-Layer zur Merkmalsextraktion, dann FC-Layer zur Klassifikation
- Conv-Layer haben 2^n Faltungskerne (Kernels), meist mit 3×3 Gewichten
- mit zunehmender Tiefe steigt die Anzahl der Faltungskerne je Conv-Layer tendenziell
- vor der Erhöhung der Anzahl der Faltungskerne steht meist ein Pooling-Layer
- die Aktivierungsfunktion in CNNs ist immer ReLU (mit Ausnahme des letzten Layers)

Übersicht

1 Motivation

2 Convolutional Neural Networks

3 CNNs mit GPUs

4 Literatur

Wie kann ich das alles nutzen?

Technische Voraussetzungen

- je tiefer das Netz, umso länger dauert alles
- kleine Netze kann man auf der CPU rechnen
- große Netze muss man auf der GPU rechnen
- geht aktuell quasi nur mit NVIDIA-GPUs → CUDA

Bibliotheken

Tensorflow Bibliothek für Deep Learning mit Python, die recht viel Spielraum für eigene Veränderungen lässt

Keras High-Level Bibliothek, die u.a. auf Tensorflow aufsetzt und die Benutzung tlw. stark vereinfacht

CNNs in Keras

- Prinzip identisch zu (normalen) Neuronalen Netzen
- Convolutional Layer (Merkmalsextraktion) und Dense Layer (Klassifikation) werden in einem Model definiert und dadurch **gemeinsam trainiert**
- es gibt einige neue Layer-Typen

Conv-Layer

Pooling

CNNs in Keras

- Prinzip identisch zu (normalen) Neuronalen Netzen
- Convolutional Layer (Merkmalsextraktion) und Dense Layer (Klassifikation) werden in einem Model definiert und dadurch **gemeinsam trainiert**
- es gibt einige neue Layer-Typen

Conv-Layer

```
Conv2D(32, (3, 3), activation='relu', padding='same',  
name='conv1')
```

32 Faltungsmasken, je 3×3

Pooling

CNNs in Keras

- Prinzip identisch zu (normalen) Neuronalen Netzen
- Convolutional Layer (Merkmalsextraktion) und Dense Layer (Klassifikation) werden in einem Model definiert und dadurch **gemeinsam trainiert**
- es gibt einige neue Layer-Typen

Conv-Layer

Pooling

```
MaxPooling2D(pool_size=(2,2))
```

Beispiel

```
# Generate dummy data
x_train = np.random.random((100, 100, 100, 3))
y_train = keras.utils.to_categorical(np.random.
    randint(10, size=(100, 1)), num_classes=10)
x_test = np.random.random((20, 100, 100, 3))
y_test = keras.utils.to_categorical(np.random.randint
    (10, size=(20, 1)), num_classes=10)

model = Sequential()
# input: 100x100 images with 3 channels -> (100, 100,
    3) tensors
# this applies 32 conv filters of size 3x3 each
model.add(Conv2D(32, (3, 3), activation='relu',
    padding='same', input_shape=(100, 100, 3)))
```

Wie man lernt

Aufteilung der Daten

Trainingsdaten werden genutzt, um die Gewichte (Parameter) des Modells zu tunen.

Validierungsdaten nutzen, um die Hyperparameter und das Modell zu bestimmen und das Training zu beenden (Anzahl der Trainingsiterationen).

Testdaten nur benutzen, um die Endergebnisse für den Bericht zu erzeugen.

Wenn man das Training weiter laufen lässt, bis der Fehler auf den Trainingsdaten minimiert wurde, hat das Netz vermutlich die Trainingsdaten auswendig gelernt!

Daher, die Genauigkeit der Validierungsdaten beobachten und bei bestem Ergebnis/Plateau stoppen!

Validierungsdaten

Zwei Möglichkeiten die Validierungsdaten zu spezifizieren:

```
model.fit(X_train, Y_train, batch_size=32, epochs=10,  
          validation_split = 0.2, verbose=1)
```

`validation_split` Anteil der zur Validierung genutzten
Trainingsdaten

```
model.fit(X_train, Y_train, batch_size=32, epochs=10,  
          validation_data = (X_val,Y_val), verbose=1)
```

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

Training flexibilisieren

Problem

- größere Netzze brauchen mehr Epochen
- die Anzahl der Epochen ist a priori nicht bestimmbar

Lösung

Erweiterung

Training flexibilisieren

Problem

Lösung

- Early Stopping
- Training wird beendet, wenn sich der Validation Loss in n aufeinander folgenden Epochen um weniger als δ_{min} verbessert
- als 1er-Liste von Callbacks der fit-Methode unter dem Parameter callbacks übergeben

```
keras.callbacks.EarlyStopping(monitor='val_loss',  
min_delta=0, patience=5)
```

Erweiterung

Training flexibilisieren

Problem

Lösung

Erweiterung

- Learning Rate nach Konvergenz senken: $\alpha_{neu} = \frac{1}{10} \alpha_{alt}$
- Training wieder aufnehmen →
`keras.callbacks.ReduceLROnPlateau`

Speichern und Laden von Gewichten

Speichern der Gewichte eines Modells:

```
model.save_weights('cifar10weights.h5')
```

Laden der Gewichte in ein anderes Modell:

```
model.load_weights('cifar10weights.h5', by_name=True)
```

Wichtig: Die Gewichte werden anhand der Namen der Layer zugeordnet! D.h., Layer mit gleichem Namen müssen die gleiche Größe haben!

Für Standardnetze hat Keras eigene Funktionen zum Laden der Struktur plus vortrainierten Gewichten (Imagenet).

Standard-Datensätze

CIFAR-10

- keras.datasets.cifar10
- Datensatz zur Objektklassifizierung
- 10 verschiedene Objektklassen
- RGB-Bilder der Größe 32×32

```
from keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.
    load_data()

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

Zum Weiterlesen...

Dropout

- Dropout-Layer verhindern, dass die Trainingsdaten auswendig gelernt werden
- dadurch entsteht eine bessere Generalisierung (auf ungewohnte Daten)
- durch Dropout-Layer werden je Trainingsdurchlauf zufällig $n\%$ der Neuronen/Kernels eines Layers ausgeschaltet
- Dropout-Layer können sowohl nach Conv-Layern als auch nach FC-Layern positioniert werden

Strided Convolution

Skip Connections

Zum Weiterlesen...

Dropout

Strided Convolution

- bei Strided Convolution wird ein Faltungskern nicht auf jedes, sondern nur auf jedes n -te Pixel zentriert
- dadurch wird das Ergebnis verkleinert
- eine Strided Convolution mit dem Faktor 2 halbiert die Seitenlängen des Bildes wie ein Pooling-Layer mit dem Faktor 2

Skip Connections

Zum Weiterlesen...

Dropout

Strided Convolution

Skip Connections

- Skip Connections werden im ResNet eingeführt und dienen zum Überspringen von Layern
- es entstehen damit zwei Informationsflüsse (einer durch einen Layer und eine Umleitung an diesem vorbei)
- der Zusammenschluss der Flüsse erfolgt durch einfaches elementweises Aufsummieren
- Skip Connections unterstützt vor allem das Training in sehr tiefen Netzen (bis zu 1000 Layer)

Präsentation Projektidee

- Stellt eure Domäne vor!
 - Was sind eure Klassen?
 - Wie sehen sie aus?
 - Worin unterscheiden sich die Klassen?
 - Was sind Gemeinsamkeiten?
- Was ist euer Plan?
 - Woher kommen die Daten?
 - Was für Merkmale wollt ihr nutzen?
 - Habt ihr euch eigene Merkmale überlegt?
 - Wollt ihr das Bild binarisieren?
 - Wird das Problem auch wissenschaftlich betrachtet?
 - Was soll euer Klassifikator sein?
 - Wie wollt ihr die CNNs nutzen?

Das, was ihr nächste Woche erzählt, ist nicht in Stein gemeißelt!

Übersicht

1 Motivation

2 Convolutional Neural Networks

3 CNNs mit GPUs

4 Literatur

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
 - 7.12 Dropout
- Erklärung: Early Stopping - but when? von Lutz Prechelt ↗
Anmerkung: Vor allem Kapitel 1 ist gut, um den Grund für Early Stopping zu verstehen.
- Erklärung: Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning ↗
Anmerkung: Adaptive Learning Rate Methods haben wir nicht betrachtet.

Data Augmentation

- [GB]: Kapitel 7 Regularization for Deep Learning
 - 7.4 Data Augmentation
- Erklärung/Beispiel: Data Augmentation: How to use Deep Learning when you have Limited Data - Part 2 ↗

Standardnetze

- [VL]: Kapitel 4 Convolutional Neural Networks
 - Convolutional Neural Networks
 - Case Study 2: AlexNet
 - Case Study 3: GoogLeNet and the Inception Module
 - Case Study 4: VGG-19
 - Case Study 5: Residual Net
 - Erklärung: Convolutional neural networks are fantastic for visual recognition tasks. ↗
 - Video: Lecture 9 — CNN Architectures - Stanford University School of Engineering @ YouTube ↗
 - Vertiefung: Originalpaper zu AlexNet ↗
 - Vertiefung: Originalpaper zu VGG ↗
 - Vertiefung: Originalpaper zu GoogLeNet ↗
 - Vertiefung: Originalpaper zu ResNet ↗

CNNs mit Keras

- Erklärung/Beispiel: Keras - Getting started with the Keras Sequential model ↗
Anmerkung: Vor allem ist jetzt VGG-like convnet wichtig.
- Erklärung/Beispiel: Keras - Usage examples for image classification models ↗
- Erklärung/Beispiel: Keras tutorial – build a convolutional neural network in 11 lines ↗
Anmerkung: Der Teil zu Logging Metrics ist nicht unbedingt relevant für uns.
- Erklärung/Beispiel: Keras Tutorial : Fine-tuning using pre-trained models ↗
Anmerkung: Nur bis einschließlich Create a new model relevant.
- Erklärung/Beispiel: How to Check-Point Deep Learning Models in Keras ↗

Referenzen I



[GW], R. Gonzalez und R. Woods

Digital Image Processing

4th ed., Pearson, 2018.

Bib-Katalog ↗



[GB], I. Goodfellow, Y. Bengio und A. Courville

Deep Learning

1st ed., MIT Press, 2016.

Bib-Katalog ↗

E-Book ↗

Referenzen II



[VL], R. Venkatesan und B. Li

Convolutional neural networks in visual computing : a concise guide

1st ed., CRC Press, 2017.

Bib-Katalog ↗

E-Book ↗