

Künstliche Intelligenz kapieren und programmieren

Teil 8: Anwendung von KI

Michael Weigend
Universität Münster



mw@creative-informatics.de
www.creative-informatics.de
2024



Materialien bei GitHub:
<https://github.com/mweigend/ki-workshop>

Tag 2

Zeit	Thema	Inhalte
9.00	Perzeptron	Neuron, Aktivierungsfunktion, Daten visualisieren mit Matplotlib, Rosenblatt-Perzeptron für logische Operationen
11.00	Aus Fehlern lernen	Error-Backpropagation, einfaches künstliches neuronales Netz (KNN) mit verborgenen Knoten
12.45	<i>Mittagspause</i>	
13.45	Ziffern erkennen	NumPy, KNN mit Array-Operationen, das Ziffern erkennen kann
15.00	Anwendung von KI	Verkehrsschilder erkennen, Gesichter erfassen, Experimente mit OpenCV, Schlussrunde
16.00	<i>Ende</i>	

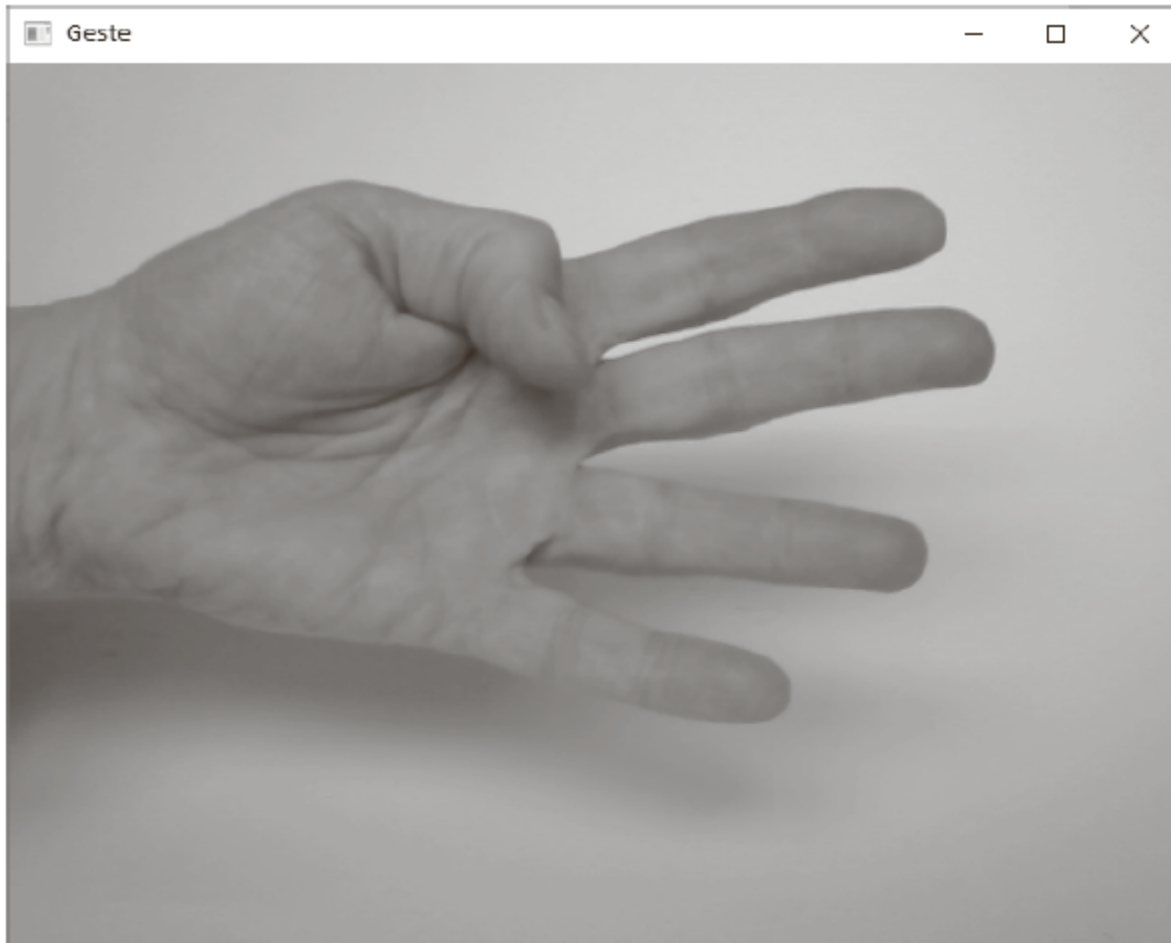
Projekte mit dem selbstprogrammieren KNN

Ideen?

Probleme?

Trainingsdaten!

Projekt: Gesten erkennen



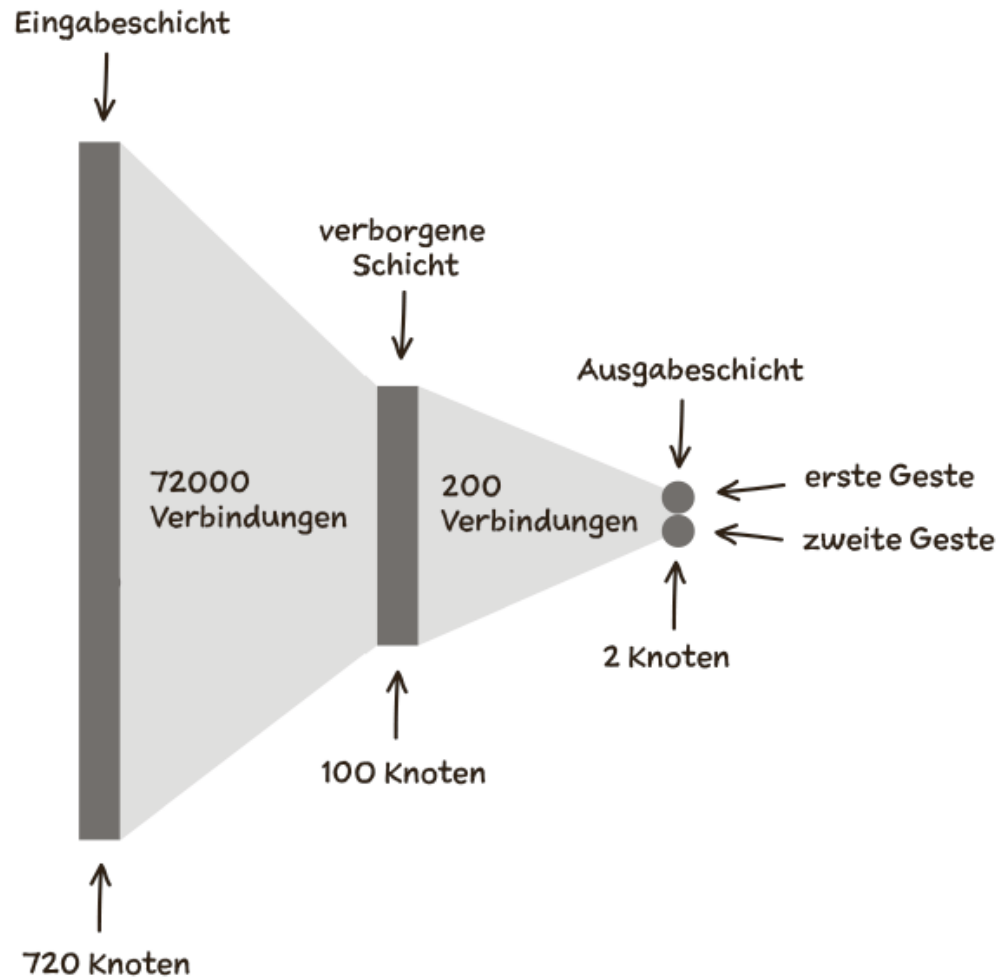
Nur zwei Gesten unterscheiden

- Daumen hoch und Daumen runter
- Hulk und Catwoman

Aus dem Kamerabild ein Array gewinnen

```
# kamera_test.py
import cv2
kamera = cv2.VideoCapture(0)
if not kamera.isOpened():
    print('Kamera nicht geöffnet!')
else:
    input('Drücke ENTER um ein Foto zu machen')
    get, frame = kamera.read()
    graustufen = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    print(graustufen)
    cv2.imshow('Geste', graustufen)
    cv2.waitKey(0)
kamera.release()
```

Neuronales Netz



Programmierung

```
# gestalten.py
import cv2
import numpy as np
import math

EPOCHEN = 100      # Anzahl Trainingsdurchläufe
LR = 0.1           # Lernrate
I_KNOTEN = 720     # Anzahl Eingabeknoten
H_KNOTEN = 100     # Anzahl verborgene Knoten
O_KNOTEN = 2       # Anzahl Ausgabeknoten
```

Symbolische und subsymbolische KI

Symbolische KI

Arbeitsweise der KI ist durch Regeln bestimmt und für den Menschen nachvollziehbar

- Chatbot
- Spielgegner
- Lernender Entscheidungsbaum

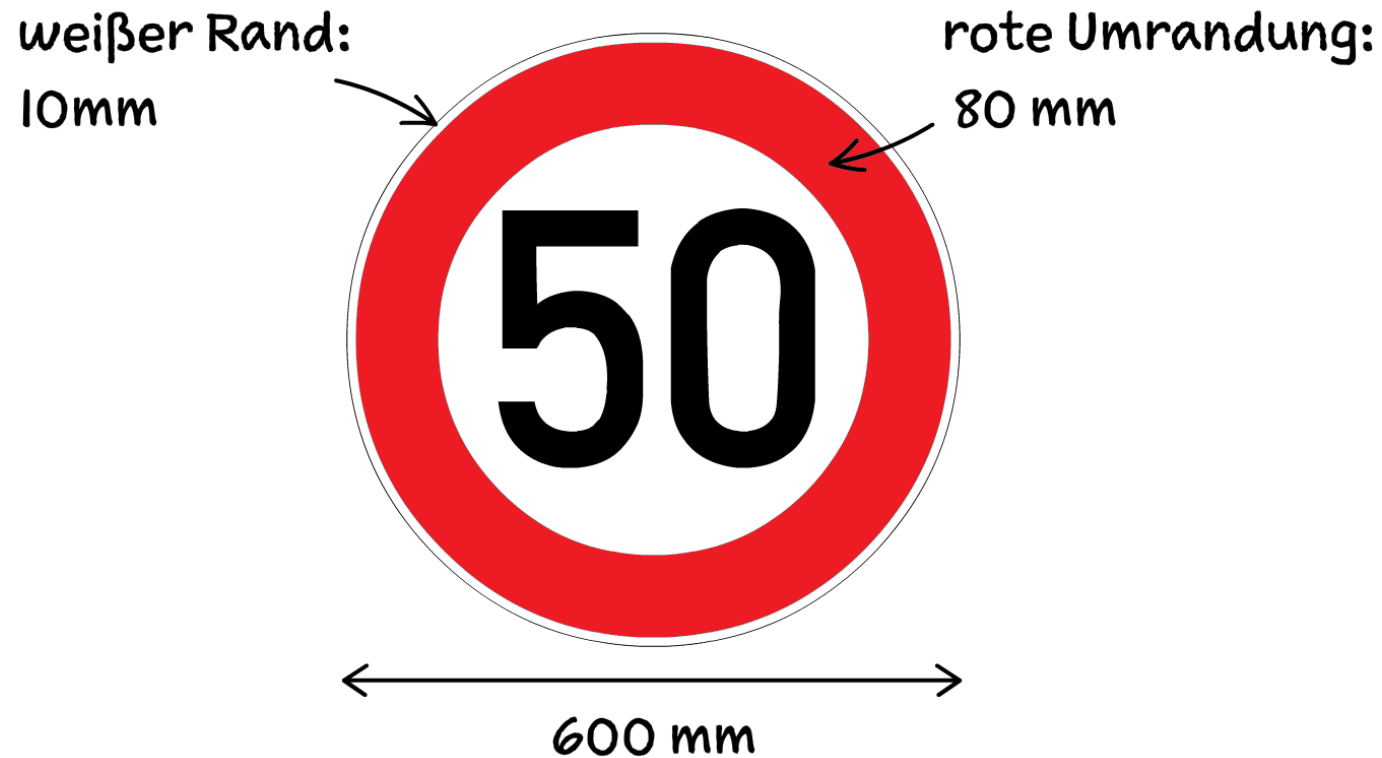
Subsymbolische KI

Arbeitsweise der KI ist **nicht** durch Regeln bestimmt und für den Menschen **nicht nachvollziehbar**

- Neuronales Netz

Reale Systeme sind eine Mischung mehrerer Techniken

Verkehrszeichen erkennen



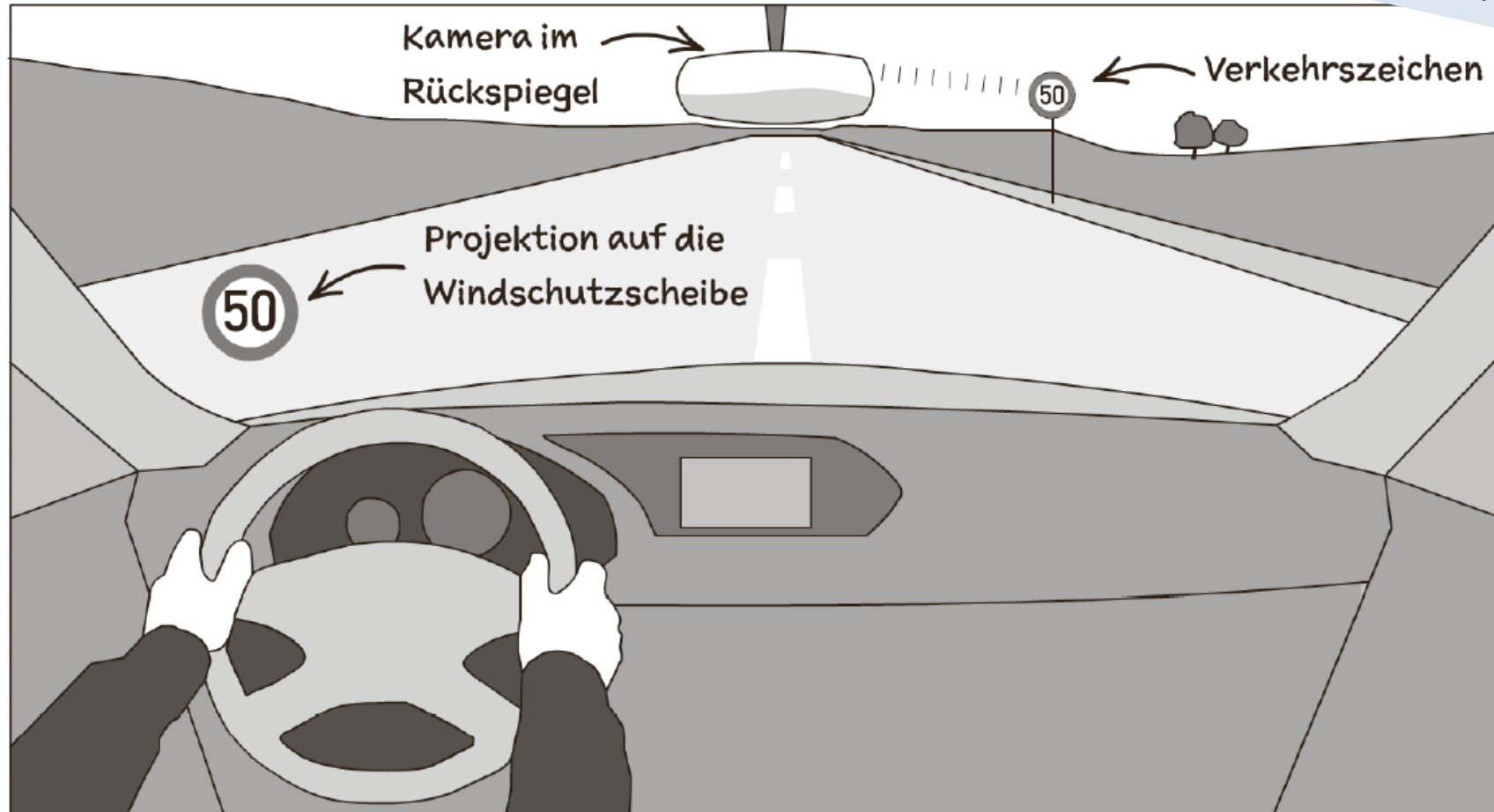
Aussehen in
Straßenverkehrsordnung
festgelegt,
Nummer: VZ-274-55

Aussehen der Verkehrszeichen in der Realität



Traffic Sign Recognition (TSR) im Auto

seit 2008



1. Schritt: Verkehrszeichen erfassen (Traffic Sign Detection)

Bildausschnitte,
die Verkehrszeichen
enthalten (ROI)



Training eines KNN mit etikettierten Bildern von Verkehrsszenen



German Traffic Sign Detection Benchmark (GTSDb), 900 Bilder

2. Schritt: Verkehrszeichen erkennen

German Traffic Sign Recognition Benchmark (GTSRB), 50000 Bilder, 40 unterschiedliche Verkehrszeichen



274-62

276

277

301

306

222-20

222-10

215

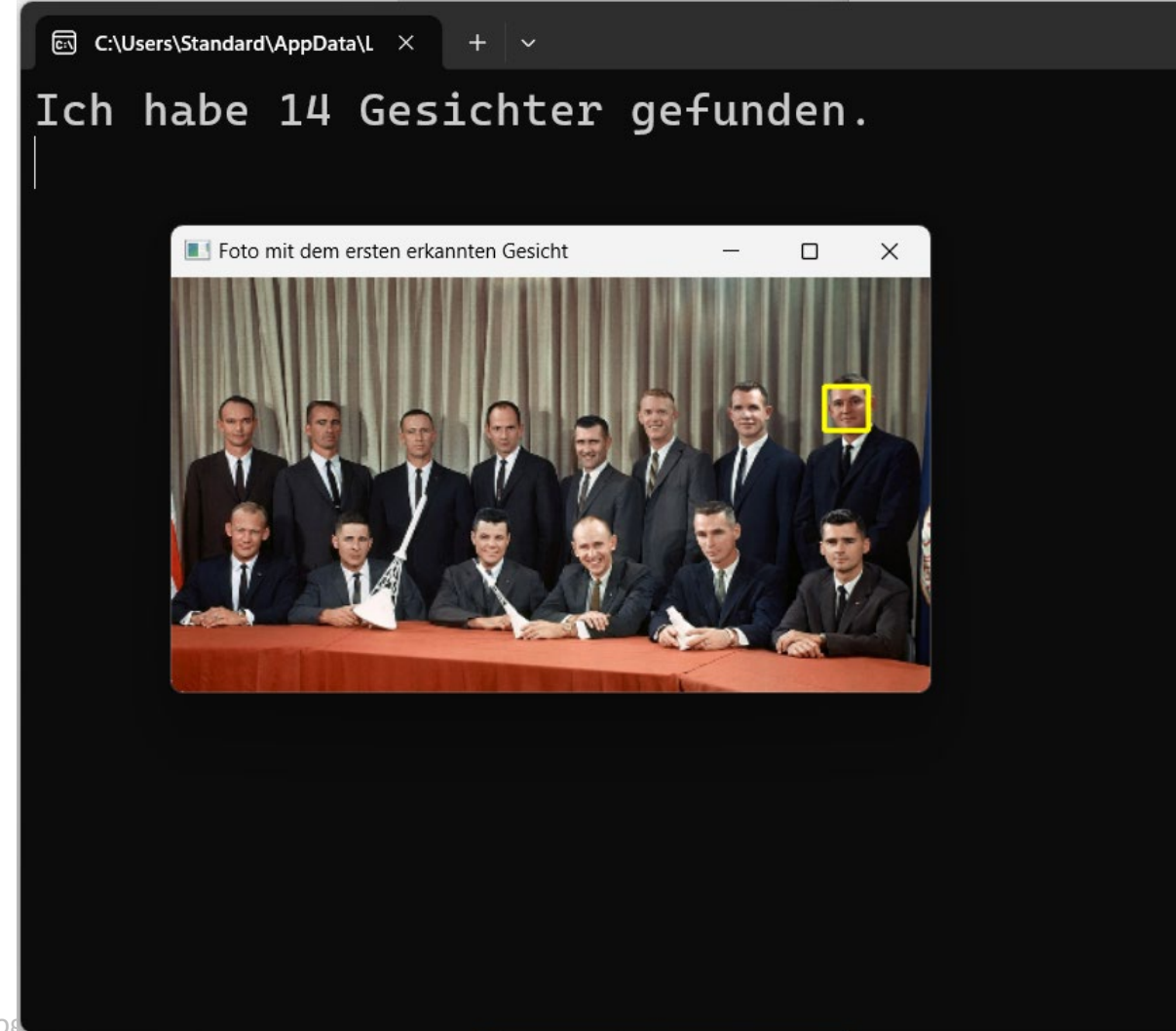
280

↑
Etikett (Label)

Projekt: Gesichter erfassen

Ziel:

- Auf einem Foto die Anzahl der Gesichter ermitteln
- Gesichter einrahmen
- Gesichter unkenntlich machen (Übung)



Vorbereitung


















Projektordner enthält:

- Bilddatei: `astronauten.png`
- XML-Datei zum Klassifizieren: `haarcascade_frontalface_default.xml`
- Programmdatei: `gesichter.py`

Nach der Installation von `opencv-python` findet man Klassifiziererdateien in folgendem Ordner:

```
... \Python311\Lib\site-packages\cv2\data
```

Haar-Kaskaden-Klassifizierer zum Erfassen von Augen, Lächeln, Gesicht von der Seite, ...

 <code>haarcascade_eye.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	334 KB
 <code>haarcascade_eye_tree_eyeglasses.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	588 KB
 <code>haarcascade_frontalcatface.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	402 KB
 <code>haarcascade_frontalcatface_extended....</code>	11.05.2023 10:19	Microsoft Edge HT...	374 KB
 <code>haarcascade_frontalface_alt.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	661 KB
 <code>haarcascade_frontalface_alt_tree.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	2.627 KB
 <code>haarcascade_frontalface_alt2.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	528 KB
 <code>haarcascade_frontalface_default.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	909 KB
 <code>haarcascade_fullbody.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	466 KB
 <code>haarcascade_lefteye_2splits.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	191 KB
 <code>haarcascade_license_plate_rus_16stag...</code>	11.05.2023 10:19	Microsoft Edge HT...	47 KB
 <code>haarcascade_lowerbody.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	387 KB
 <code>haarcascade_profileface.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	810 KB
 <code>haarcascade_righteye_2splits.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	192 KB
 <code>haarcascade_russian_plate_number.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	74 KB
 <code>haarcascade_smile.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	185 KB
 <code>haarcascade_upperbody.xml</code>	11.05.2023 10:19	Microsoft Edge HT...	768 KB

Haar-Kaskaden



Beispiele für Haar-Features (benannt nach Alfred Haar)



Algorithmus von Paul Viola und Michael Jones (2001)
Grundidee: Bilder haben typische Hell-Dunkelbereiche (Haar –Features). Kamerabild wird auf Haar-Features untersucht.

Programmierung

NumPy-Array mit den
Pixeln des Bildes

Graustufenbild

Klassifiziererobjekt

Liste von Tupeln
(x, y, Breite, Höhe)

Erstes Rechteck
einzeichnen

```
import cv2
FOTO = 'astronauten.png'
XMLDATEI = 'haarcascade_frontalface_default.xml'
bild = cv2.imread(FOTO)
grau = cv2.cvtColor(bild, cv2.COLOR_BGR2GRAY)
klassifizierer = cv2.CascadeClassifier(XMLDATEI)
rechtecke = klassifizierer.detectMultiScale(grau,
                                             scaleFactor=1.05,
                                             minNeighbors=5)

n = len(rechtecke)
print('Ich habe', n, 'Gesichter gefunden.')
x, y, w, h = rechtecke[0]
cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 255), 2)
cv2.imshow('Foto mit dem ersten erkannten Gesicht', bild)
cv2.waitKey(0) # warte bis Taste gedrückt
cv2.destroyAllWindows() # SchlieÙe das Viewer-Fenster
```

Übung 8.1

Aufgabe 1

Testen Sie das Starterprojekt.

Aufgabe 2

Erweitern Sie das Starterprojekt. Fügen Sie einige `print()`-Anweisungen ein, die die Arbeitsweise des Programms verständlich machen (z.B. Ausgabe des Numpy-Arrays, der das Foto darstellt).

Aufgabe 3

Wandeln Sie das Programm ab, sodass auf dem Bild alle Gesichter durch ein graues Rechteck unkenntlich gemacht werden.

Hinweis: Wenn Sie beim Aufruf der Funktion `cv2.rectangle()` als Liniendicke (letztes Argument) `-1` angeben, wird ein gefülltes Rechteck gezeichnet.



Lösungen 8.1

Aufgabe 2

Erweitern Sie das Starterprojekt. Fügen Sie einige `print()`-Anweisungen ein, die die Arbeitsweise des Programms verständlich machen (z.B. Ausgabe des Numpy-Arrays, der das Foto darstellt).

```
...
bild = cv2.imread(FOTO)
print('Numpy-Array des Bilds:')
print(bild) #
grau = cv2.cvtColor(bild, cv2.COLOR_BGR2GRAY)
print('Graustufenbild: ')
print(grau)
```

Aufgabe 3

Wandeln Sie das Programm ab, sodass auf dem Bild alle Gesichter durch ein graues Rechteck unkenntlich gemacht werden.

```
...
print('Ich habe', n, 'Gesichter gefunden.')
for x,y,w,h in rechtecke:
    cv2.rectangle(img, (x, y), (x+w, y+h), (150, 150, 150), -1)
cv2.imshow('Foto mit unkenntlich gemachten Gesichtern', img)
```

...

Rückblick

- Das selbstprogrammierte KNN kann man z.B. für eigene Projekte zur Objekterkennung mit der Kamera verwenden. Es können aber nur wenige Objekte unterschieden werden, wenn man keine umfangreichen Trainingsdaten hat.
- Bei der **Erfassung** (detection) von Objekten werden auf einen größeren Bild Bildausschnitte lokalisiert, die jeweils ein Objekt des gesuchten Typs (z.B. Gesicht) enthalten.
- Bei der **Erkennung** (recognition) eines Objekts wird auf einem Bild standardisierter Größe ein Objekt identifiziert (z.B. Verkehrsschild oder Ziffer) .

Schlussrunde