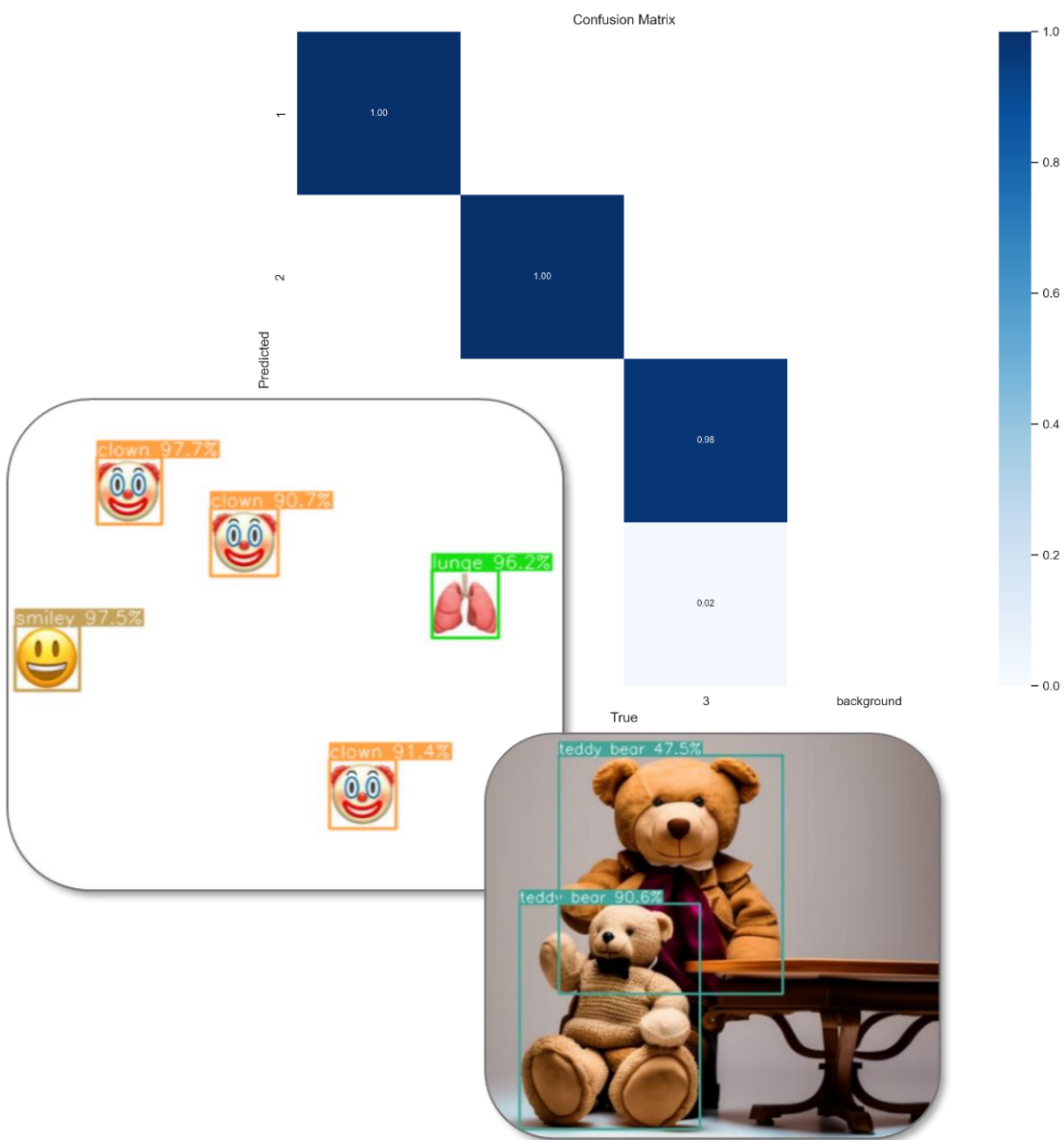


# Training von YoloV8

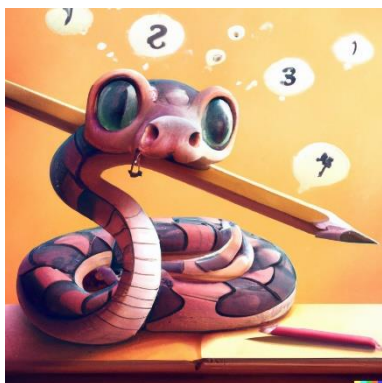
## Mit einem synthetischen selbst erstellten Datensatz

Natalia & Thomas Jörg, [thomas@iludis.de](mailto:thomas@iludis.de)

Stand 26. Februar 2023



Einführung in die Aufgabe .....	3
Was ist YOLO? .....	3
Was ist die Technologie hinter YOLO? .....	3
Was macht die „Regressionsschicht“ bei YOLO? .....	3
Standalone-YOLO .....	4
Transferlearning mit YOLO .....	5
Installation und Inbetriebnahme von YOLOv8.....	6
Entscheidung im Schulkontext: Installation von Edupyter .....	6
Yolo: Grundinstallation .....	7
0. Jupyter-Skript: Die erste Objekterkennung: Testbilder und Live-Webcam .....	7
Das Projekt: Yolo-Transfer Learning .....	10
1. Jupyter-Skript: Installation und Konfiguration von YOLOv8 .....	10
2. Jupyter-Skript: Erstellung eines synthetischen Datensatzes .....	12
3. Jupyter-Skript: Trainieren des Modells .....	16
4. Jupyter-Skript: Testen und Deployment .....	17
Ausblick: Label-Studio, Installation und Einführung .....	19
Quellen:.....	20



*Alle Bilder in diesem Skript sind entweder selbst erstellt, erzeugt oder mit Hilfe von Stable Diffusion / DALL-E2 generiert. Die Emoji-Bilder laufen unter Creative Commons 4.0 und sind frei verteilbar. Mit diesem Skript werden darüber hinaus keinerlei kommerzielle Ziele verfolgt, es handelt sich um Lernmaterialien, die in Schulen eingesetzt werden.*

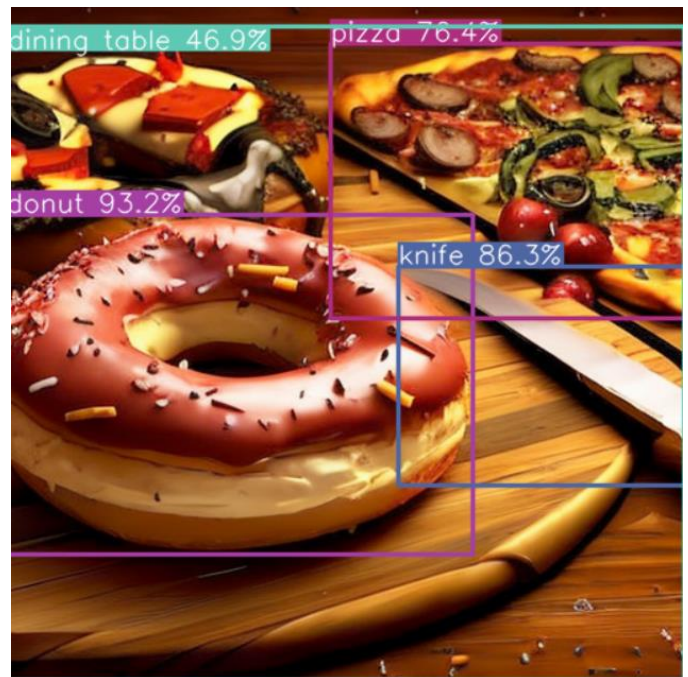
## Einführung in die Aufgabe

### Was ist YOLO?

Die YOLO-Objekterkennung (kurz für "You Only Look Once") ist eine Technologie, die es einem Computerprogramm ermöglicht, Gegenstände in Bildern oder Videos zu erkennen.

Dazu wird das Bild in viele kleine Bereiche aufgeteilt, und für jeden Bereich wird ein Vorschlag gemacht, welches Objekt in diesem Bereich zu sehen ist. Diese Vorschläge werden dann mithilfe von KI-Technologie überprüft und verbessert, um möglichst genaue Ergebnisse zu erzielen.

Im Gegensatz zu anderen Objekterkennungsmethoden ist YOLO sehr schnell und kann in Echtzeit arbeiten, was es ideal für Anwendungen wie selbstfahrende Autos oder Überwachungssysteme macht.



### Was ist die Technologie hinter YOLO?

Die YOLO-Objekterkennung basiert auf Deep Learning, also einem neuronalen Netz mit vielen Schichten.

Das Kernkonzept von YOLO besteht darin, dass das Bild nur einmal analysiert wird. Dazu wird das Bild in viele kleine Bereiche unterteilt und jedes dieser Bereiche wird in ein sogenanntes "Feature-Vector" umgewandelt. Der Feature-Vector ist ein Vektor, der alle Informationen über das Objekt in diesem Bereich des Bildes enthält, wie z.B. Farbe, Form und Textur.

Diese Feature-Vektoren werden dann in einem neuronalen Netzwerk verarbeitet, das darauf trainiert wurde, die verschiedenen Merkmale von Objekten zu erkennen und zu klassifizieren. Das neuronale Netzwerk verwendet eine Klassifikationsschicht, um zu entscheiden, welches Objekt in welchem Bereich des Bildes zu sehen ist, und eine Regressionschicht, um die Position und Größe des Objekts im Bild genau zu bestimmen.

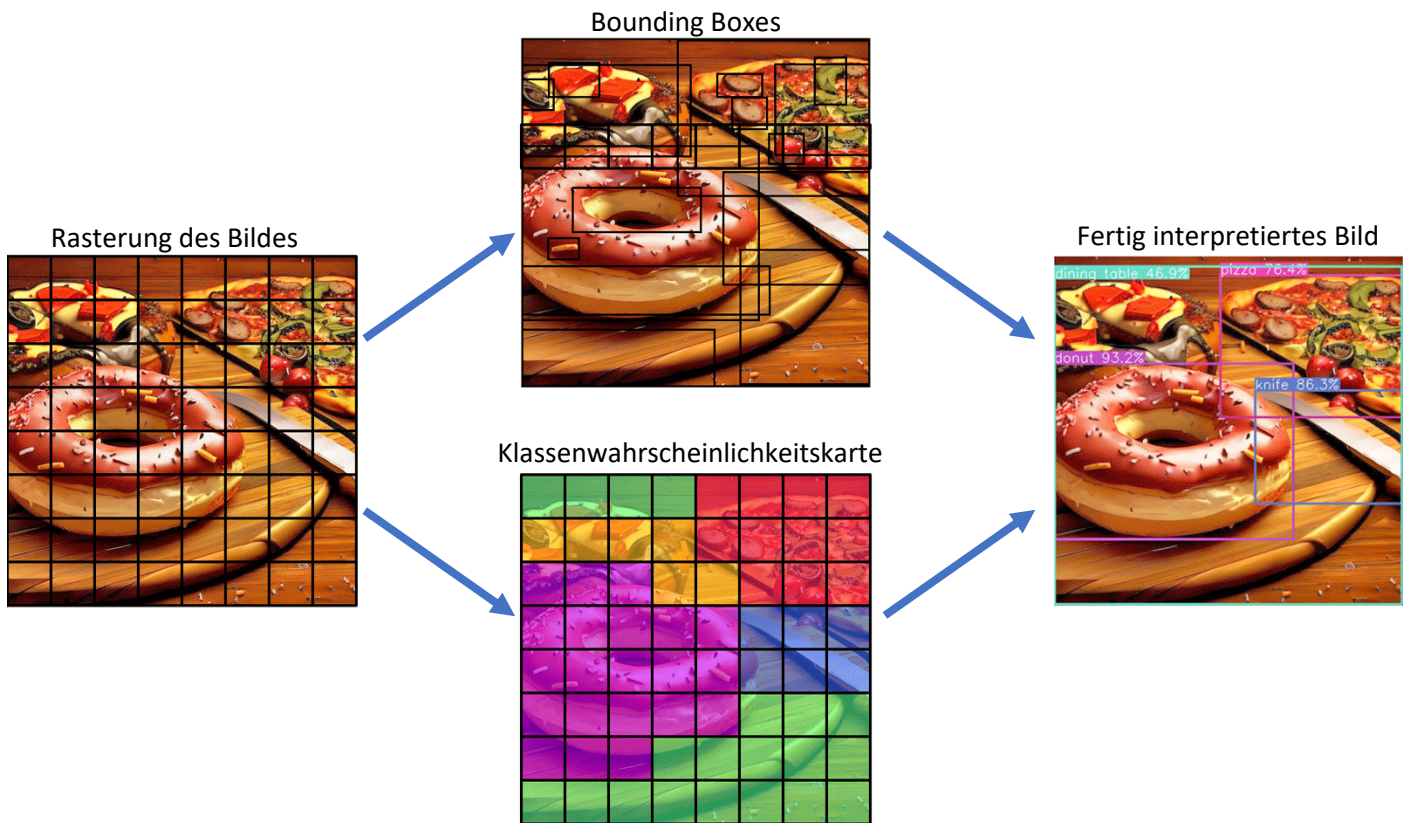
Das Training des neuronalen Netzwerks erfolgt durch die Verwendung großer Datensätze von Bildern mit bekannten Objekten und deren Positionen. Das Netzwerk wird schrittweise verbessert, indem es wiederholt mit neuen Daten trainiert wird, bis es eine hohe Genauigkeit bei der Objekterkennung erreicht.

Das Ergebnis ist eine sehr schnelle und genaue Objekterkennungstechnologie, die in Echtzeit auf Videostreams angewendet werden kann. YOLO hat zahlreiche Anwendungen, darunter die automatische Erkennung von Verkehrsschildern, die Überwachung von Überwachungskameras und die Erkennung von Gesichtern in sozialen Medien.

### Was macht die „Regressionschicht“ bei YOLO?

Der Regression-Anteil bei YOLO ist verantwortlich für die Bestimmung der genauen Position und Größe eines erkannten Objekts im Bild.

Das neuronale Netzwerk von YOLO gibt für jedes Bildsegment einen Vorschlag ab, welches Objekt in diesem Bereich des Bildes zu sehen ist. Die Regressionschicht nimmt diesen Vorschlag und berechnet die genaue Position des Objekts im Bild, indem sie die Koordinaten des linken oberen Eckpunkts und des rechten unteren Eckpunkts des Objekts schätzt.



Dieser Prozess wird mithilfe von "Bounding Boxes" durchgeführt, welche eine begrenzende Umrahmung um das erkannte Objekt legen. Die Bounding Box wird durch vier Werte definiert:

- die x- und y-Koordinaten des linken oberen Eckpunkts und die x- und y-Koordinaten des rechten unteren Eckpunkts.
- Neu bei YOLOv8: x- und y-Koordinaten des linken oberen Eckpunktes und die Breite und Höhe des Rahmens.

Der Regressions-Anteil des neuronalen Netzwerks lernt, wie man diese Bounding-Box-Koordinaten so genau wie möglich schätzt, indem er mit großen Datensätzen von Bildern trainiert wird, die mit den entsprechenden Koordinaten markiert sind. Zitat aus

[https://www.researchgate.net/publication/337146307\\_Object\\_Recognition\\_Using\\_Convolutional\\_Neural\\_Networks](https://www.researchgate.net/publication/337146307_Object_Recognition_Using_Convolutional_Neural_Networks)

*„Jede Bounding-Box besteht aus 5 Vorhersagen:  $x$ ,  $y$ ,  $w$ ,  $h$  und Konfidenz. Die Koordinaten  $(x, y)$  geben den Mittelpunkt des Kastens relativ zu den Grenzen der Gitterzelle an. Die Breite und Höhe werden in Bezug auf das gesamte Bild vorhergesagt.“*

## Standalone-YOLO

YOLO kann direkt ‚out of the Box‘ verwendet werden, also zur Vorhersage von:

- welchen Objekten (Klassifizierung),
- Orten von Objekten auf dem Bild (Objekt-Detektierung und Segmentierung)
- Häufigkeiten von Objekten (Objekte können gezählt werden).

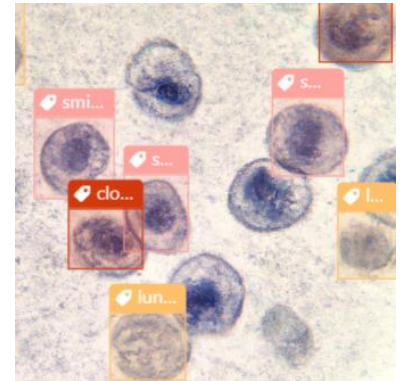
Dabei sind die Objekte bestimmten vortrainierten Klassen zuzuordnen. Davon gibt es 80 Stück, hauptsächlich Alltagsgegenstände von Personen über Stühle, Verkehrsampeln, Skateboards, Handtaschen und Bananen. Siehe z.B. hier: <https://medium.com/@cunhafh/transfer-learning-with-yolo-v3-darknet-and-google-colab-7f9a6f9c2afc>

## Transferlearning mit YOLO

Es ist aber möglich, YOLO auf eigene Aufgabenstellungen hin anzupassen. Dazu können die vorgefertigten Modelle mittels eigener Trainingsdaten angepasst werden. Diesen Prozess, ein vortrainiertes Neuronales Netz anzupassen, indem man es mittels eigener Daten erneut trainiert, bezeichnet man als „Transferlearning“. Anstatt also ein neues Modell von Grund auf zu trainieren, kann das bestehende YOLO-Modell verwendet werden, um schneller und effektiver zu lernen.

YOLO muss dann nicht erneut von Anfang an trainiert werden, sondern es wird nur ein kleiner Teil des Modells auf die neue Aufgabe angepasst.

Das Prinzip des Transfer-Learnings ermöglicht es, Zeit und Ressourcen zu sparen, da nicht jedes Mal ein neues Modell erstellt werden muss. Es ist auch ein wichtiger Aspekt des maschinellen Lernens und der künstlichen Intelligenz, da es ermöglicht, dass Modelle schnell und einfach an neue Aufgaben angepasst werden können.



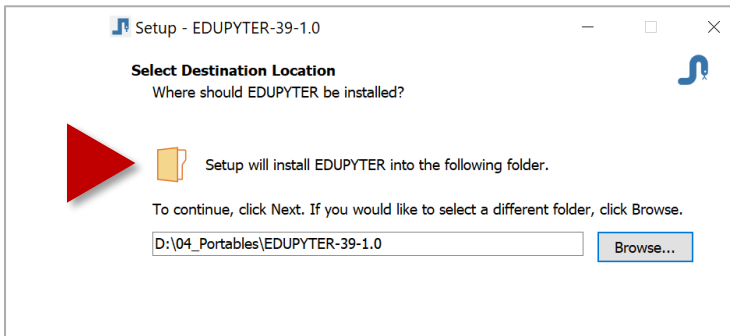
# Installation und Inbetriebnahme von YOLOv8

## Entscheidung im Schulkontext: Installation von Edupyter

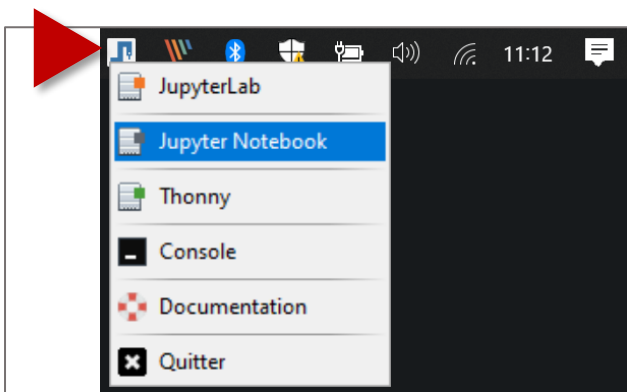
Aus Gründen von Einfachheit, Praktikabilität und der Umgehung von Rechte-Restriktionen auf Schulrechnern wird Edupyter gewählt. Edupyter lässt sich portabel installieren und kann hier heruntergeladen werden:

<https://www.portabledevapps.net/edupyter.php>

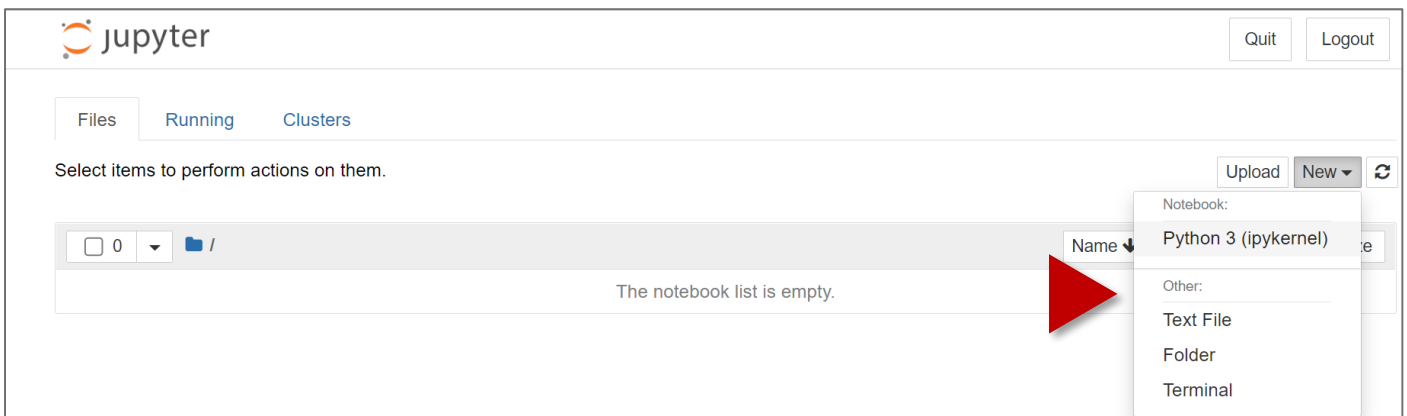
Die Installation sollte nicht in den von Edupyter vorgeschlagenen Standardordner erfolgen, sondern in einen selbst gewählten Ordner, wie zum Beispiel auf einem USB-Stick oder einem externen Laufwerk:



Startet man die Entwicklungsumgebung, erscheint ein Icon im Systray:



Hier startet man ein Jupyter-Notebook; es sollte sich ein Browser öffnen, in dem das Jupyter-Notebook läuft:



Innerhalb des neu erstellten jupyter-Notebooks gibt man nun den folgenden Befehl ein und führt ihn aus,

- entweder mit einem Klick auf ‚Run‘
- oder mit dem gleichzeitigen Drücken der Tasten ‚Strg‘ und ‚Enter‘

## Yolo: Grundinstallation

Yolo in der aktuellen Version 8 wurde von dem US-Amerikanischen Unternehmen ‚Ultralytics‘ weiterentwickelt.

Die Software wird für Lernzwecke kostenfrei (GNU General Public License v3.0, <https://ultralytics.com/license>) abgegeben und kann deshalb problemlos aus dem Internet heruntergeladen werden. Die Pakete sind sehr komfortabel gestaltet; der User muss nur minimal in die hochautomatisierten Skripte eingreifen und kaum etwas nachkonfigurieren. Deshalb kann es gut im Schulkontext eingesetzt werden.



### Wichtige Links:

<https://ultralytics.com/yolov8>

<https://github.com/ultralytics/ultralytics>

<https://www.kaggle.com/code/ultralytics/yolov8/notebook>

<https://blog.roboflow.com/whats-new-in-yolov8/>

### Die Ordnerstruktur

Alle heruntergeladenen Dateien gehören in den Eduplyter\IPYNB-Ordner:

Name	Änderungsdatum	Typ	Größe
Icons	26.02.2023 15:49	Dateiordner	
0_yoloV8Standalone.ipynb	26.02.2023 18:56	IPYNB-Datei	11 KB
1_EmojiYolo_InstallTestYolo.ipynb	26.02.2023 18:56	IPYNB-Datei	9 KB
2_EmojiYolo_CreateDataset.ipynb	26.02.2023 18:56	IPYNB-Datei	8 KB
3_EmojiYolo_TrainSmileyData.ipynb	26.02.2023 18:56	IPYNB-Datei	3 KB
4_EmojiYolo_Predict.ipynb	26.02.2023 18:56	IPYNB-Datei	4 KB
5_Yolo_LabelStudio.ipynb	26.02.2023 18:56	IPYNB-Datei	1 KB
Testpic1.jpg	25.02.2023 10:33	JPG-Datei	99 KB
Testpic2.jpg	25.02.2023 10:33	JPG-Datei	159 KB
Testpic3.jpg	25.02.2023 10:33	JPG-Datei	185 KB
Testpic4.jpg	25.02.2023 10:34	JPG-Datei	98 KB
Testpic5.jpg	25.02.2023 20:03	JPG-Datei	142 KB

Alle weiteren Dateien werden im Laufe des Projektes erzeugt.

## 0. Jupyter-Skript: Die erste Objekterkennung: Testbilder und Live-Webcam

### 0\_yoloV8Standalone.ipynb

Im folgenden wird ein Skript beschrieben, das als Standalone-Version zu Demozwecken jederzeit und überall aufgerufen werden kann. Es funktioniert eigentlich immer und ein Live-Feed der Webcam ist auf aktuellen Rechnern schnell genug, um in Echtzeit abzulaufen.

---

#### Zelle 1: Installation Ultralytics Yolo

Die Installation erfolgt einmal; anschließend liegen alle Pakete innerhalb des Edupyter-Ordners. Und sind nahezu so portabel wie Edupyter selbst. Bei Pfadproblemen – die bei Umzügen des Edupyter-Ordners vorkommen können – müssen mit `os.setdir(„directorxyz“)` eventuell die Pfade nachkorrigiert werden.

---

```
# ----- Nur einmal: Yolo installieren und überprüfen
!pip install ultralytics
import ultralytics
ultralytics.checks()
```

---

#### Zelle 2: Yolo-Bibliotheken laden

In der Grundinstallation von Edupyter sollten all diese Bibliotheken

```
# ----- IMMER: Bibliotheken laden
from ultralytics import YOLO
import numpy as np
from PIL import Image
import requests
from io import BytesIO
import cv2
```

---

#### Zelle 3: Neuronales Netz laden und BoundingBox/Labeling-Funktionen definieren

Für das jeweilige Modell `n = „nano“` / `s = „standard“` / `m = „medium“` / `l = „large“` / `x = „xtra large“` wird die jeweilige Zeile ent-kommentiert. YOLO lädt dann automatisch die geforderte Neuronale-Netzdatei mit dem Modell herunter auf die Festplatte. Dies geschieht einmal und muss anschließend nicht mehr wiederholt werden.

Die beiden Funktionen `„box_label“` und `„plot_bboxes“` sind standardisierte Umsetzungen der Ausgaben der YOLO-Outputneuronen. Der Quelltext ist abgeleitet von hier:

<https://inside-machinelearning.com/en/yolov8-how-to-use/>

---

```
# ----- IMMER: Neuronales Netzwerk laden.

model = YOLO("yolov8n.pt")
#model = YOLO("yolov8s.pt")
#model = YOLO("yolov8m.pt")
#model = YOLO("yolov8l.pt")
#model = YOLO("yolov8x.pt")
```



```

# ----- label definieren, welches an der Box gezeigt wird.
def box_label(image, box, label='', color=(128, 128, 128), txt_color=(255, 255, 255)):
    lw = max(round(sum(image.shape) / 2 * 0.001), 2)
    p1, p2 = (int(box[0]), int(box[1])), (int(box[2]), int(box[3]))
    cv2.rectangle(image, p1, p2, color, thickness=lw, lineType=cv2.LINE_AA)
    if label:
        tf = max(lw - 1, 1) # font thickness
        w, h = cv2.getTextSize(label, 0, fontScale=lw / 3, thickness=tf)[0] # text width, height
        outside = p1[1] - h >= 3
        p2 = p1[0] + w, p1[1] - h - 3 if outside else p1[1] + h + 3
        cv2.rectangle(image, p1, p2, color, -1, cv2.LINE_AA) # filled
        cv2.putText(image,
                    label, (p1[0], p1[1] - 2 if outside else p1[1] + h + 2),
                    0,
                    lw / 3,
                    txt_color,
                    thickness=tf,
                    lineType=cv2.LINE_AA)

# ----- Box malen

def plot_bboxes(image, boxes, conf=0.01):
    #Define COCO Labels and color for labels

    labels = {0: u'__background__', 1: u'person', 2: u'bicycle', 3: u'car', 4: u'motorcycle', 5:
u'airplane', 6: u'bus', 7: u'train', 8: u'truck', 9: u'boat', 10: u'traffic light', 11: u'fire
hydrant', 12: u'stop sign', 13: u'parking meter', 14: u'bench', 15: u'bird', 16: u'cat', 17:
u'dog', 18: u'horse', 19: u'sheep', 20: u'cow', 21: u'elephant', 22: u'bear', 23: u'zebra', 24:
u'giraffe', 25: u'backpack', 26: u'umbrella', 27: u'handbag', 28: u'tie', 29: u'suitcase', 30:
u'frisbee', 31: u'skis', 32: u'snowboard', 33: u'sports ball', 34: u'kite', 35: u'baseball bat',
36: u'baseball glove', 37: u'skateboard', 38: u'surfboard', 39: u'tennis racket', 40: u'bottle',
41: u'wine glass', 42: u'cup', 43: u'fork', 44: u'knife', 45: u'spoon', 46: u'bowl', 47:
u'banana', 48: u'apple', 49: u'sandwich', 50: u'orange', 51: u'broccoli', 52: u'carrot', 53:
u'hot dog', 54: u'pizza', 55: u'donut', 56: u'cake', 57: u'chair', 58: u'couch', 59: u'potted
plant', 60: u'bed', 61: u'dining table', 62: u'toilet', 63: u'tv', 64: u'laptop', 65: u'mouse',
66: u'remote', 67: u'keyboard', 68: u'cell phone', 69: u'microwave', 70: u'oven', 71:
u'toaster', 72: u'sink', 73: u'refrigerator', 74: u'book', 75: u'clock', 76: u'vase', 77:
u'scissors', 78: u'teddy bear', 79: u'hair drier', 80: u'toothbrush'}

    colors = [(89, 161, 197), (67, 161, 255), (19, 222, 24), (186, 55, 2), (167, 146, 11), (190, 76,
98), (130, 172, 179), (115, 209, 128), (204, 79, 135), (136, 126, 185), (209, 213, 45), (44, 52,
10), (101, 158, 121), (179, 124, 12), (25, 33, 189), (45, 115, 11), (73, 197, 184), (62, 225,
221), (32, 46, 52), (20, 165, 16), (54, 15, 57), (12, 150, 9), (10, 46, 99), (94, 89, 46), (48, 37,
106), (42, 10, 96), (7, 164, 128), (98, 213, 120), (40, 5, 219), (54, 25, 150), (251, 74, 172), (0,
236, 196), (21, 104, 190), (226, 74, 232), (120, 67, 25), (191, 106, 197), (8, 15, 134), (21, 2,
1), (142, 63, 109), (133, 148, 146), (187, 77, 253), (155, 22, 122), (218, 130, 77), (164, 102,
79), (43, 152, 125), (185, 124, 151), (95, 159, 238), (128, 89, 85), (228, 6, 60), (6, 41, 210), (11,
1, 133), (30, 96, 58), (230, 136, 109), (126, 45, 174), (164, 63, 165), (32, 111, 29), (232, 40,
70), (55, 31, 198), (148, 211, 129), (10, 186, 211), (181, 201, 94), (55, 35, 92), (129, 140,
233), (70, 250, 116), (61, 209, 152), (216, 21, 138), (100, 0, 176), (3, 42, 70), (151, 13, 44), (216,
102, 88), (125, 216, 93), (171, 236, 47), (253, 127, 103), (205, 137, 244), (193, 137, 224), (36, 152,
214), (17, 50, 238), (154, 165, 67), (114, 129, 60), (119, 24, 48), (73, 8, 110)]

    #plot each boxes
    for box in boxes:

        #add score
        label = labels[int(box[-1])+1] + " " + str(round(100 * float(box[-2]), 1)) + "%"

        #filter every box under conf threshold if conf threshold setted
        if box[-2] > conf:
            color = colors[int(box[-1])]
            box_label(image, box, label, color)

    #show image
    cv2.namedWindow('test', cv2.WINDOW_NORMAL) # macht das Bild 'resziabile'
    cv2.namedWindow("test", cv2.WINDOW_AUTOSIZE)
    cv2.imshow("test", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

```

# Das Projekt: Yolo-Transfer Learning

## 1. Jupyter-Skript: Installation und Konfiguration von YOLOv8

### 1\_EmojiYolo\_InstallTestYolo.ipynb

#### Zelle 1 und 2 und 3: Installation Ultralytics Yolo, Bibliotheken laden und BBox-Skripten definieren

Siehe vorhergehende Seite, „Grundinstallation“. Hier der Vollständigkeit halber, falls das Projekt nur für Transferlearning ausgeführt wird:

```
# ----- Nur einmal: Yolo installieren und überprüfen
!pip install ultralytics
import ultralytics
ultralytics.checks()

# ----- IMMER: Bibliotheken laden
from ultralytics import YOLO
import numpy as np
from PIL import Image
import requests
from io import BytesIO
import cv2

# ----- Neuronales Netzwerk laden.
model = YOLO("yolov8s.pt")

# ----- label definieren, welches an der Box gezeigt wird.
def box_label(image, box, label='', color=(128, 128, 128), txt_color=(255, 255, 255)):
    lw = max(round(sum(image.shape) / 2 * 0.001), 2)
    p1, p2 = (int(box[0]), int(box[1])), (int(box[2]), int(box[3]))
    cv2.rectangle(image, p1, p2, color, thickness=lw, lineType=cv2.LINE_AA)
    if label:
        tf = max(lw - 1, 1) # font thickness
        w, h = cv2.getTextSize(label, 0, fontScale=lw / 3, thickness=tf)[0] # text width,
height
        outside = p1[1] - h >= 3
        p2 = p1[0] + w, p1[1] - h - 3 if outside else p1[1] + h + 3
        cv2.rectangle(image, p1, p2, color, -1, cv2.LINE_AA) # filled
        cv2.putText(image, label, (p1[0], p1[1] - 2 if outside else p1[1] + h + 2), 0, lw /
3, txt_color, thickness=tf, lineType=cv2.LINE_AA)

# ----- Box malen
def plot_bboxes(image, boxes, conf=0.01):
    labels = {0: u'__background__', 1: u'person', 2: u'bicycle', 3: u'car', 4: u'motorcycle', 5:
u'airplane', 6: u'bus', 7: u'train', 8: u'truck', 9: u'boat', 10: u'traffic light', 11: u'fire
hydrant', 12: u'stop sign', 13: u'parking meter', 14: u'bench', 15: u'bird', 16: u'cat', 17:
u'dog', 18: u'horse', 19: u'sheep', 20: u'cow', 21: u'elephant', 22: u'bear', 23: u'zebra', 24:
u'giraffe', 25: u'backpack', 26: u'umbrella', 27: u'handbag', 28: u'tie', 29: u'suitcase', 30:
u'frisbee', 31: u'skis', 32: u'snowboard', 33: u'sports ball', 34: u'kite', 35: u'baseball bat',
36: u'baseball glove', 37: u'skateboard', 38: u'surfboard', 39: u'tennis racket', 40: u'bottle',
41: u'wine glass', 42: u'cup', 43: u'fork', 44: u'knife', 45: u'spoon', 46: u'bowl', 47:
u'banana', 48: u'apple', 49: u'sandwich', 50: u'orange', 51: u'broccoli', 52: u'carrot', 53:
u'hot dog', 54: u'pizza', 55: u'donut', 56: u'cake', 57: u'chair', 58: u'couch', 59: u'potted
plant', 60: u'bed', 61: u'dining table', 62: u'toilet', 63: u'tv', 64: u'laptop', 65: u'mouse',
66: u'remote', 67: u'keyboard', 68: u'cell phone', 69: u'microwave', 70: u'oven', 71:
u'toaster', 72: u'sink', 73: u'refrigerator', 74: u'book', 75: u'clock', 76: u'vase', 77:
u'scissors', 78: u'teddy bear', 79: u'hair drier', 80: u'toothbrush'}
```

```

colors = [(89, 161, 197),(67, 161, 255),(19, 222, 24),(186, 55, 2),(167, 146, 11),(190, 76,
98),(130, 172, 179),(115, 209, 128),(204, 79, 135),(136, 126, 185),(209, 213, 45),(44, 52,
10),(101, 158, 121),(179, 124, 12),(25, 33, 189),(45, 115, 11),(73, 197, 184),(62, 225,
221),(32, 46, 52),(20, 165, 16),(54, 15, 57),(12, 150, 9),(10, 46, 99),(94, 89, 46),(48, 37,
106),(42, 10, 96),(7, 164, 128),(98, 213, 120),(40, 5, 219),(54, 25, 150),(251, 74, 172),(0,
236, 196),(21, 104, 190),(226, 74, 232),(120, 67, 25),(191, 106, 197),(8, 15, 134),(21, 2,
1),(142, 63, 109),(133, 148, 146),(187, 77, 253),(155, 22, 122),(218, 130, 77),(164, 102,
79),(43, 152, 125),(185, 124, 151),(95, 159, 238),(128, 89, 85),(228, 6, 60),(6, 41, 210),(11,
1, 133),(30, 96, 58),(230, 136, 109),(126, 45, 174),(164, 63, 165),(32, 111, 29),(232, 40,
70),(55, 31, 198),(148, 211, 129),(10, 186, 211),(181, 201, 94),(55, 35, 92),(129, 140,
233),(70, 250, 116),(61, 209, 152),(216, 21, 138),(100, 0, 176),(3, 42, 70),(151, 13, 44),(216,
102, 88),(125, 216, 93),(171, 236, 47),(253, 127, 103),(205, 137, 244),(193, 137, 224),(36, 152,
214),(17, 50, 238),(154, 165, 67),(114, 129, 60),(119, 24, 48),(73, 8, 110)]
#plot each boxes
for box in boxes:

    #add score
    label = labels[int(box[-1])+1] + " " + str(round(100 * float(box[-2]),1)) + "%"

    #filter every box under conf threshold if conf threshold setted
    if box[-2] > conf:
        color = colors[int(box[-1])]
        box_label(image, box, label, color)

cv2.namedWindow("test", cv2.WINDOW_AUTOSIZE)
cv2.imshow("test",image)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

---

#### Zelle 4: Test der Installation mit vier Testbildern

Die vier mitgelieferten Testbilder sind allesamt in StableDiffusion erstellt und deshalb (nach aktuellem Stand des Rechts) frei von Copyrights. Bitte machen Sie mit den Bildern, was immer sie möchten! Ich hab mit den Dingen nix vor. Außer natürlich Yolo zu testen 😊

---

```

# ----- Bild von der Festplatte laden
# ----- vorhanden sind Testpic1, Testpic2, Testpic3, Testpic4
(StableDiffusion)
image = cv2.imread('Testpic5.jpg')
print(image.shape)

# ----- Umwandeln in Numpy-Array und Yolo-Bildanalyse
image = np.asarray(image)
results = model.predict(image)
image = np.asarray(image)
plot_bboxes(image, results[0].boxes.boxes, conf = 0.46)

```

---

## 2. Jupyter-Skript: Erstellung eines synthetischen Datensatzes

### 2\_EmojiYolo\_CreateDataset.ipynb

Um einen Datensatz zu erstellen, gibt es zwei Möglichkeiten, eine schnelle und eine langwierige. Die schnelle Möglichkeit ist die Erstellung eines synthetischen Datensatzes, bei dem man die Bilder selbst generiert, damit auch die Labels kennt und direkt mit erstellt. Mit dem vollständigen Datensatz wird nun das Netz trainiert.

Hier wird zunächst der schnelle Weg des synthetischen Datensatzes beschrieben.

#	emoji	unicode	name		
0	1	😊	U+1F600	grinning face	data:image
1	2	😄	U+1F603	grinning face with big eyes	data:image
2	3	😁	U+1F604	grinning face with smiling eyes	data:image
3	4	😆	U+1F601	beaming face with smiling eyes	data:image

### Emojis-Datensatz: Aufbau und Erzeugung

Es werden in ein weißes 640x640-Pixelbild eine zufällige Anzahl an Emojis an einer zufälligen Stelle des Bildes hineinkopiert und als Jpeg-Datei abgespeichert. Die jeweiligen Zufallszahlen kennt man; deshalb lassen sich die Label-Dateien mit den jeweiligen Emoji-Klassen und den Bounding-Box-Koordinaten optimal mit erzeugen. Das ganze geht vollautomatisiert:

1. Alle Bilder und Labels werden erzeugt,
2. und in die jeweiligen Ordnerstrukturen ( datasets \ {train, test, valid} \ {images, labels} ) eingeordnet,
3. und die zugehörige Anweisungsdatei für das YOLO-Training wird miterzeugt (emoji.yaml)

Die fertigen Daten können direkt mit Skript 3 weiterverwendet werden.

### Die Emoji-Daten:

Die Emoji-Datei kann hier heruntergeladen werden:

<https://www.kaggle.com/datasets/subinium/emojiimage-dataset>

Die Daten liegen in folgender Lizenzierung vor:

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

**Sie sind also frei verfügbar und nutzbar!**

Die Daten liegen in Form einer CSV-Datei vor; diese besitzt die Größe von 31.6MB (!) und beinhaltet neben vielen ASCII-Daten auch alle Bildinformationen von Emojis in der folgenden Datenform:  
72Pixel x 72 Pixel x 4 Kanäle

Die Datei muss in den selben Ordner wie das Jupyter-Notebook gelegt werden, damit es geladen werden kann.



## Zelle 1: Bibliotheken laden

---

```
# ----- Bibliotheken laden
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt

from PIL import Image
import io
import os
import glob
import base64
import cv2
```

---

## Zelle 2: Emoji-CSV laden

---

```
# ----- CSV-Datei einlesen
emoji = pd.read_csv('full_emoji.csv')
emoji.head(6)
```

---

## Zelle 3: Testweise einige Emojis ausgeben

---

```
# ----- Einige Emojis testweise ausgeben. Nicht unbedingt nötig.
dpi = mpl.rcParams['figure.dpi'] #Kommentar: aktuelle DPI-Auflösung auslesen
print("Auflösung in DPI:", dpi)

for i in range(0,5):
    base64_decoded = base64.b64decode(emoji['Apple'][i].split(',')[1])
    image = Image.open(io.BytesIO(base64_decoded)).convert('RGBA')
    #print(image) #Kommentar: PIL-Darstellung. Weniger flexibel als numpy-Array
    #plt.imshow(image) #Kommentar: PIL-Darstellung. Weniger flexibel als numpy-Array
    #print(image.size) #Kommentar: PIL-Darstellung. Weniger flexibel als numpy-Array
    string="emoji" + str(i) + ".png"
    imgarray = np.asarray(image)
    #print(imgarray.shape)#Kommentar: Debugging-Ausgabe für Bildauflösung
    h, w, depth = imgarray.shape
    plt.figure(figsize=(w / float(dpi), h / float(dpi))).add_axes([0, 0, 1, 1])
    plt.axis('off')
    plt.imshow(imgarray, aspect=1)
    plt.savefig(string)
```

---

## Zelle 4: Ordnerstruktur löschen und neu erzeugen

---

```
# ----- Hiermit werden alle Directories gelöscht und neu initialisiert
import os
import shutil
if os.path.exists("datasets"):
    shutil.rmtree('datasets')
if not os.path.exists("datasets/test"):
    os.makedirs("datasets/test/images")
    os.makedirs("datasets/test/labels")
if not os.path.exists("datasets/train"):
    os.makedirs("datasets/train/images")
    os.makedirs("datasets/train/labels")
if not os.path.exists("datasets/valid"):
    os.makedirs("datasets/valid/images")
    os.makedirs("datasets/valid/labels")
```

## Zelle 5: Bibliotheken laden

---

```
# ----- Erzeugung des Datensatzes

import random
random.seed(42) #Kommentar: Erzeugt immer die selben Pseudo-Zufallszahlen

dpi = mpl.rcParams['figure.dpi']

i=1 #Kommentar: Frei wählbare Emoji-Nummer
base64_decoded = base64.b64decode(emoji['Apple'][i].split(',')[1])
image = Image.open(io.BytesIO(base64_decoded)).convert('RGBA')
sourceA = np.asarray(image)

i=100 #Kommentar: Frei wählbare Emoji-Nummer
base64_decoded = base64.b64decode(emoji['Apple'][i].split(',')[1])
image = Image.open(io.BytesIO(base64_decoded)).convert('RGBA')
sourceB = np.asarray(image)

i=200 #Kommentar: Frei wählbare Emoji-Nummer
base64_decoded = base64.b64decode(emoji['Apple'][i].split(',')[1])
image = Image.open(io.BytesIO(base64_decoded)).convert('RGBA')
sourceC = np.asarray(image)

bildanzahl = 200
for bildnummer in range(bildanzahl): #Kommentar: Anzahl der Trainingsdaten
    canvas = np.zeros((640,640,4), dtype=np.uint8)

    #Kommentar: Für YOLO korrekte Auflösung
    #print("canvas-shape:", canvas.shape) #Kommentar: Debuggen des Arrays
    #print("canvas-dtype:", canvas.dtype) #Kommentar: Debuggen des Arrays
    anzahl = random.randint(2, 5)
    text = ""

    for j in range(anzahl):
        dx = random.randint(0, 640-72)
        dy = random.randint(0, 640-72)
        who = random.randint(0,2)
        if who == 0:
            h, w, depth = sourceA.shape
            canvas[dx:dx+w, dy:dy+h] = sourceA[:, :]
        if who == 1:
            h, w, depth = sourceB.shape
            canvas[dx:dx+w, dy:dy+h] = sourceB[:, :]
        if who == 2:
            h, w, depth = sourceC.shape
            canvas[dx:dx+w, dy:dy+h] = sourceC[:, :]
        text = text+str(who)+" "+str((dy+(h/2))/640)+" "+str((dx+(w/2))/640)+"
"+str(h/640)+" "+str(w/640)+"\n"

    #print(text) #Kommentar: Ausgabe der Bounding-Box-Berechnungen
    plt.figure(figsize=(640 / dpi, 640 / dpi)).add_axes([0, 0, 1, 1])
    plt.axis('off')
    plt.imshow(canvas, aspect=1)
    #train 70%, valid 20%, test 10%
    if bildnummer <= int(bildanzahl*0.7):
        pfad_bilder = "datasets/train/images/"
        pfad_labels = "datasets/train/labels/"
    if bildnummer > int(bildanzahl*0.7) and bildnummer < int(bildanzahl*0.9):
        pfad_bilder = "datasets/valid/images/"
        pfad_labels = "datasets/valid/labels/"
    if bildnummer > int(bildanzahl*0.9):
        pfad_bilder = "datasets/test/images/"
        pfad_labels = "datasets/test/labels/"

    dateiname = "emojis"+str(bildnummer)
```

```
plt.savefig(pfad_bilder + dateiname + ".jpg") #Kommentar: Der Alpha-Kanal wird so
verworfen.
```

```
if os.path.exists(pfad_labels + dateiname + ".txt"):
    os.remove(pfad_labels + dateiname + ".txt")
f = open(pfad_labels + dateiname + ".txt", "a")
f.write(text)
f.close()
```

#Kommentar: Wenn man close() auskommentiert, sieht man zwar alle Bilder, aber es gibt eine Speicherplatzwarnung:

```
plt.close()
```

```
yaml_pfad="datasets/"
dateiname="emojis.yaml"
```

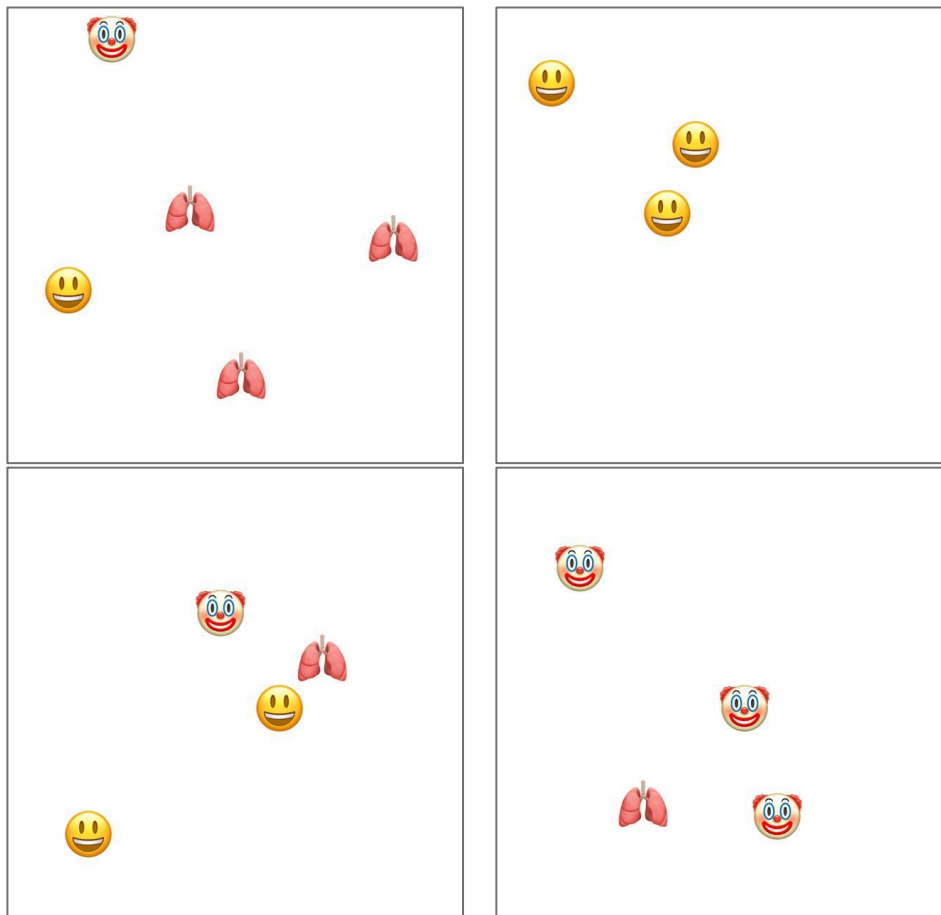
```
#-----Erläuterung Yaml-Datei:
# train: ../datasets/train/images #Pfad für Trainingsdaten
# val: ../datasets/valid/images #Pfad für Validierungsdaten
# test: ../datasets/test/images #Pfad für Testdaten
# nc: 3 #Anzahl an Klassen
# names: ['1', '2', '3'] #Nummerierung der Klassen. ACHTUNG: NIEMALS die 0
verwenden.
```

```
yaml_text="train: ../datasets/train/images\nval: ../datasets/valid/images\ntest:
../datasets/test/images\nnc: 3\nnames: ['1', '2', '3']"
```

```
f = open(yaml_pfad + dateiname, "a")
f.write(yaml_text)
f.close()
```

---

### Einige typische Bilder, die erzeugt werden:



### 3. Jupyter-Skript: Trainieren des Modells

#### 3\_EmojiYolo\_TrainSmileyData.ipynb

Das nun folgende eigentliche Training läuft weitestgehend automatisiert ab. Lediglich bei den Pfadangaben muss man etwas aufpassen. Eventuell müssen Pfade nochmals nachjustiert werden. Die hier abgebildeten Skripte laufen aber weitestgehend reibungslos.

#### Zelle 1: Pfade setzen und Training initialisieren

```
# ----- Pfade setzen und Training initiieren
# ----- Achtung: hier passiert etwa 15 Minuten lang nichts. Das ist ok.
import os
HOME = os.getcwd()
print(HOME)
os.chdir('datasets')
print(os.getcwd())

# ----- Erstellung des neu trainierten Neuronalen Netzes.
# ----- Bezeichnung: "best.pt". Diese liegt im Ordner
# ----- IPYNB\datasets\runs\detect\train\weights
!yolo task=detect mode=train model=yolov8s.pt data=emojis.yaml epochs=5 plots=True
```

#### Zelle 2: Confusion-Matrix anzeigen lassen

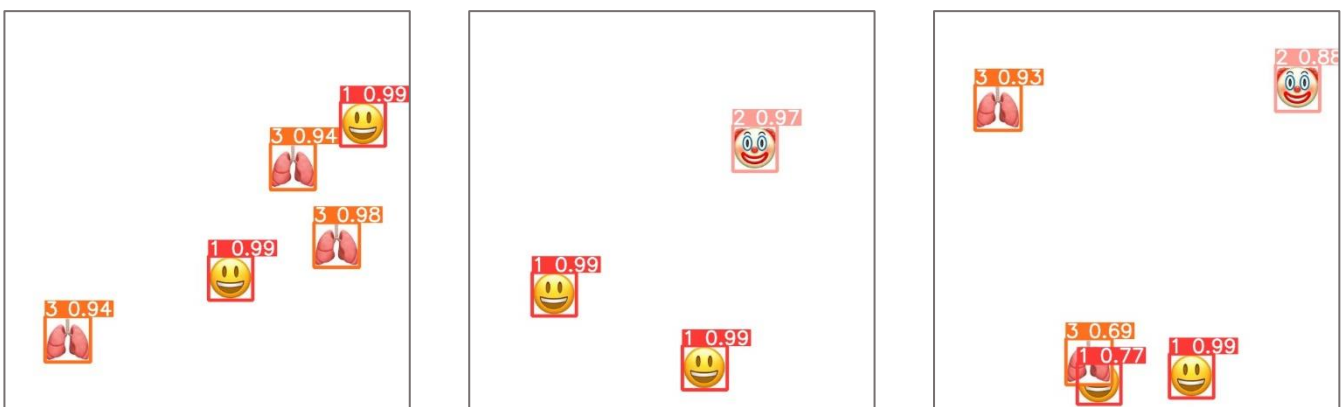
```
# ----- Darstellung der Confusion-Matrix aus dem Ordner
# ----- IPYNB\datasets\runs\detect\train
from IPython.display import Image
Image(filename='runs/detect/train/confusion_matrix.png', width=600)
```

#### Zelle 3: Validieren und testen.

```
# ----- Validierung, Ergebnisse im Ordner
# ----- IPYNB\datasets\runs\detect\val
!yolo task=detect mode=val model=runs/detect/train/weights/best.pt data=emojis.yaml

# ----- Erstellt Vorhersagen für alle Testbilder im Ordner
# ----- IPYNB\datasets\runs\detect\predict
!yolo task=detect mode=predict model=runs/detect/train/weights/best.pt conf=0.25
source=test/images save=True
```

#### Einige typische Validierungs-Daten:





## 4. Jupyter-Skript: Testen und Deployment

### 4\_EmojiYolo\_Predict.ipynb

```
In [*]: # ----- Prediction
image = cv2.imread('datasets/train/images/emojis3.jpg')
print(image.shape)

image = np.asarray(image)
results = model.predict(image)
print(results[0].boxes.boxes)
plot_bboxes(image, results[0].boxes.boxes, conf = 0.5)

(640, 640, 3)

Ultralytics YOLOv8.0.45 Python-3.10.8 torch-1.13.1+cpu CPU
Model summary (fused): 168 layers, 11126745 parameters, 0 gradients, 28.4 GFLOP
0: 640x640 1 1, 3 2s, 1 3, 267.3ms
Speed: 1.0ms preprocess, 267.3ms inference, 8.0ms postprocess per image at sha
tensor([[ 96.00000, 103.00000, 166.00000, 175.00000,  0.97727,  1.00000],
        [  7.00000, 285.00000,  77.00000, 355.00000,  0.97508,  0.00000],
        [459.00000, 225.00000, 531.00000, 298.00000,  0.96159,  2.00000],
        [348.00000, 430.00000, 420.00000, 505.00000,  0.91372,  1.00000],
        [219.00000, 158.00000, 292.00000, 231.00000,  0.90698,  1.00000]])
```

The 'test' window displays the following detections:

- clown 97.7%
- clown 90.7%
- smiley 97.5%
- lunge 96.2%
- clown 91.4%

Die fertige selbsttrainierte YOLO-NeuralNet-Datei „best.pt“ wird in das selbe Verzeichnis kopiert, in der auch die Jupyter-Notebooks liegen, damit sie direkt geladen werden kann. In Zelle 3 wird dann ein Bild aus den Datasets geladen und interpretiert. Natürlich können – bzw. sollten – hier auch eigene Bilder mit den jeweiligen Pfaden gesetzt werden.

#### Zelle 1: Bibliotheken laden

```
# ----- Bibliotheken laden
from ultralytics import YOLO
import numpy as np
from PIL import Image
import requests
from io import BytesIO
import cv2
```

## Zelle 2: Neuronales Netz laden und BoundingBox/Labeling-Funktionen definieren

---

```
# ----- Bibliotheken laden Neuronales Netzwerk laden.
model = YOLO("best.pt")

# ----- Label definieren, welches an der Box gezeigt wird.
def box_label(image, box, label='', color=(128, 128, 128), txt_color=(255, 255, 255)):
    lw = max(round(sum(image.shape) / 2 * 0.001), 2)
    p1, p2 = (int(box[0]), int(box[1])), (int(box[2]), int(box[3]))
    cv2.rectangle(image, p1, p2, color, thickness=lw, lineType=cv2.LINE_AA)
    if label:
        tf = max(lw - 1, 1) # font thickness
        w, h = cv2.getTextSize(label, 0, fontScale=lw / 3, thickness=tf)[0] # text width,
height
        outside = p1[1] - h >= 3
        p2 = p1[0] + w, p1[1] - h - 3 if outside else p1[1] + h + 3
        cv2.rectangle(image, p1, p2, color, -1, cv2.LINE_AA) # filled
        cv2.putText(image,
                    label, (p1[0], p1[1] - 2 if outside else p1[1] + h + 2),
                    0,
                    lw / 3,
                    txt_color,
                    thickness=tf,
                    lineType=cv2.LINE_AA)

# ----- Box malen
# ----- ACHTUNG: HIER EIGENE KLASSEN ANZEIGEN
def plot_bboxes(image, boxes, conf=0.5):
    #Define COCO Labels and color for labels
    labels = {0: u'__background__', 1: u'smiley', 2: u'clown', 3: u'lunge', 4:
u'gibtsnet'}
    colors = [(89, 161, 197), (67, 161, 255), (19, 222, 24), (186, 55, 2), (255, 0, 0)]
    #plot each boxes
    for box in boxes:

        #add score
        label = labels[int(box[-1])+1] + " " + str(round(100 * float(box[-2]),1)) + "%"

        #filter every box under conf threshold if conf threshold setted
        if box[-2] > conf:
            color = colors[int(box[-1])]
            box_label(image, box, label, color)

cv2.namedWindow("test", cv2.WINDOW_AUTOSIZE)
cv2.imshow("test", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

---

## Zelle 3: Bild laden und vorhersagen

---

```
# ----- Prediction
image = cv2.imread('datasets/train/images/emojis3.jpg')
print(image.shape)

image = np.asarray(image)
results = model.predict(image)
print(results[0].boxes.boxes)
plot_bboxes(image, results[0].boxes.boxes, conf = 0.5)
```

---



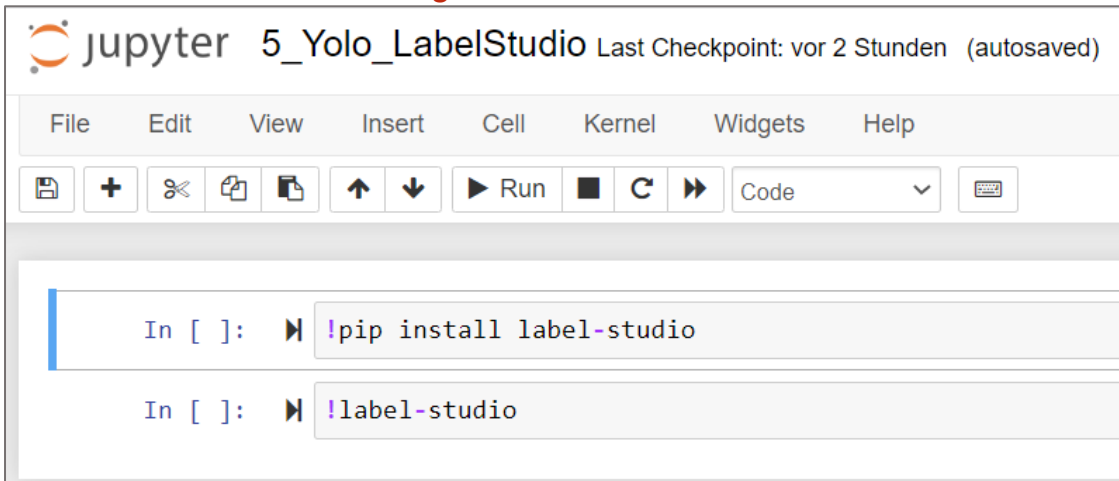
Offizielles Logo Label-Studio, <https://github.com/heartexlabs/label-studio#install-locally-with-pip>

Will man eigene Bild-Datensätze erstellen, so muss man die Bilder allesamt einzeln labeln und die Daten exportieren, und in Ordner aufteilen. Das ist nichts für Anfänger, da der investierte Zeitaufwand erheblich ist und man sich ernsthaft die Frage stellen muss, was der Schüler dabei denn lernen soll.

Für individuelle Projekte, zum Beispiel für AGs, Seminarkurse, Jugend Forscht o.ä., kann so etwas aber Sinn ergeben. Deshalb soll hier ein Tool kurz vorgestellt werden, welches sich sehr einfach installieren lässt Label-Studio.

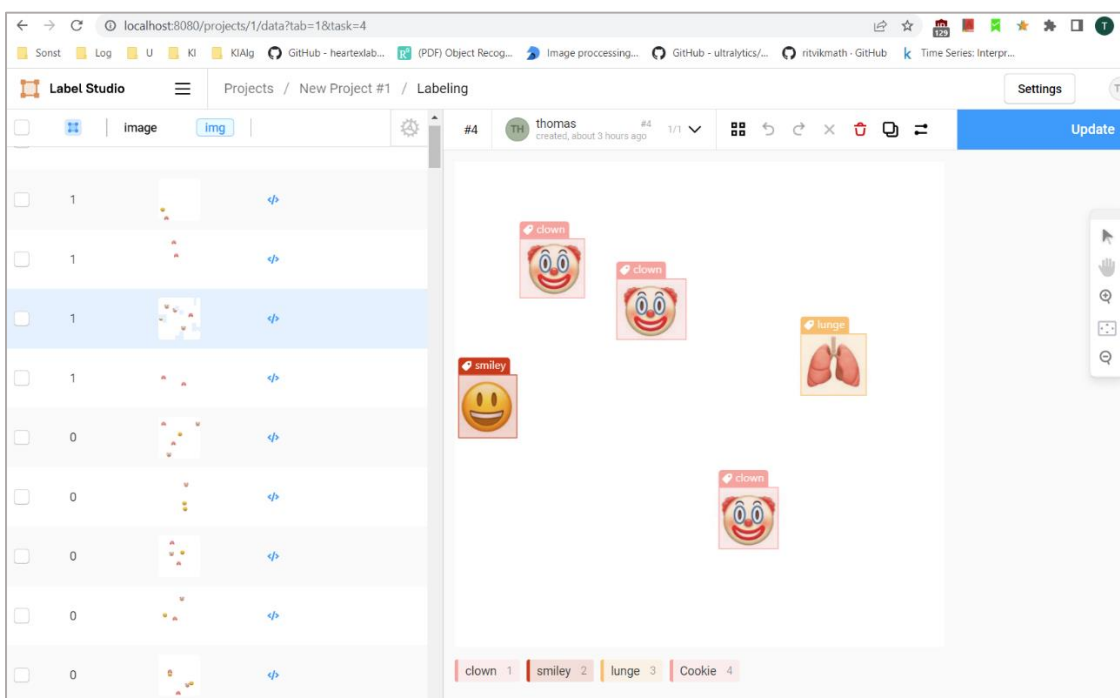
Es existieren auch professionelle, web-basierte Anwendungen wie z.B. „RoboFlow“, die beim Labeln unterstützen. Hat man einige Bilder gelabelt, so versucht die Software anschliessend eigenständig vorzulabeln. Aber: Man weiß nicht, wie sich die Preispolitik von solchen Unternehmen zukünftig entwickeln wird.

### Label-Studio installiert und erzeugt man mit einem Zweizeiler:



```
In [ ]: !pip install label-studio

In [ ]: !label-studio
```



## Quellen:

<https://inside-machinelearning.com/en/yolov8-how-to-use/>

<https://inside-machinelearning.com/en/bounding-boxes-python-function/>

<https://www.kaggle.com/datasets/subinium/emojiiimage-dataset>

<https://the-decoder.de/yolov8-zeigt-die-enormen-faehigkeiten-des-maschinellen-sehens/>

<https://kikaben.com/yolov5-transfer-learning-dogs-cats/>

<https://github.com/heartexlabs/label-studio#install-locally-with-pip>

<https://blog.roboflow.com/whats-new-in-yolov8/>

<https://cocodataset.org/#overview>

<https://learnopencv.com/train-yolov8-on-custom-dataset/>