# Object Detection

1. Daten sammeln mit aufnahme(), Im Fall von Object Detection kann klasse beliebig angegeben werden, da diese nicht weiter relevant ist.
2. Bilder in Roboflow annotieren:

* Neues Object Detection Projekt anlegen
* Bilder/Ordner mit Bildern hochladen
* Bilder annotieren in dem Bounding Boxen eingezeichnet werden
* Neue Version des Datensatzes erstellen: Train-/Val-/Test-Daten balancen (Für jede Kategorie mindestens ein Bild!), Resize, Augmentation
* Falls man bereits annotierte Bilder aus einem Roboflow Projekt in ein anderes übertragen will: Neuen Datensatz erstellen (am besten ohne Augmentation), Bilder als zip Downloaden (bspw. in v8-Format), einen Ordner erstellen und in diesen data.yaml, die Bilder und die Labels kopieren, den neuen Ordner normal in Roboflow hochladen

1. Annotierten Datensatz laden:

* In Roboflow: Versioons -> Download Dataset -> Show Download Code
* Kopieren Des Codes der Form:
* from roboflow import Roboflow
* rf = Roboflow(api\_key="<key>")
* project = rf.workspace("karlsruher-institut-fr-technologie-7bdnc").project("hksd-gsu3r")
* version = project.version(2)
* dataset = version.download("yolov8")
* In den Python Code einfügen´

**Funktion**: training\_detection()

Trainiert ein neuronales Netz zur Object detection.

**Argumente**:

* dataset = <dataset (Aus dem von Roboflow kopierten Code)>
* epochen = <Anzahl Epochen>
* img\_size = < Form: (width, height), Alle Bilder werden auf angegebene Form skaliert. Default: (640,640)>

**Beispiel**:

training\_detection(dataset,   
 epochen=15,   
 img\_size=(320,320))

**Anmerkungen**:

Das genutzte Modell ist auf einen Datensatz des Formats 640x640 vortrainiert. Dieses Format muss jedoch für das eigene Training nicht beibehalten werden.

Die Bilder werden unter Anwendung von Padding in das gewünschte Format gebracht.

Die Zusammenfassung des Trainingsergebnisses wird automatisch gespeichert. Der Pfad dazu wird automatisch auf der Konsole ausgegeben. Bspw.: runs\detect\train152. Es werden sowohl verschiedene Auswertungsmetriken, als auch das Ergebnis des Testens auf den Testbildern angezeigt.

Das Modell wird ebenfalls in einem Unterordner ‚weights‘ gespeichert. Bspw.: Results saved to runs\detect\train15.

**Funktion**: testen\_detection()

Nimmt ein Bild mit der EspCam auf und führt basierend auf einem angegebenen Modell Object detection durch.

**Argumente**:

* url = <URL/IP der EspCam>
* model = <Pfad zum gewünschten Modell, übliche Form: 'runs/detect/train/weights/best.pt'>
* conf\_thresh = <Sicherheit ab der ein Objekt detektiert werden soll, zwischen 0 und 1>
* img\_size = <optinale Angabe, Form: (width, height) mit default=(640,640), falls das Bild zuvor in ein bestimmtes Format gebracht werden soll>

**Beispiel**:

testen\_detection(url='http://192.168.0.235/capture',  
 model='runs/detect/train12/weights/best.pt',  
 conf\_thresh=0.3,   
 img\_size=None)

**Anmerkungen**:

Das Modell kann jegliche Bild-Eingabegröße verarbeiten, skaliert diese Standardmäßig jedoch immer auf 640x640 für die Verarbeitung. Zum Schluss wird es jedoch automatisch wieder zurück in seine originale Größe skaliert. Ist dies nicht gewünscht, so kann eine explizite Angabe für img\_size gemacht werden.

Das aufgenommene Bild wird als latest\_capture.pic gespeichert und das annotierte Bild mit vorhergesagten bounding Boxen unter latest\_capture\_annotated.pic. Auf der Konsole werden die vorhergesagten Labels und die Koordinaten der bounding Boxen ausgegeben.