

# Realisierung einer Plattform für Maschinelles Sehen auf Basis Docker mit PyTorch und YOLO

T3\_3101 Studienarbeit

des Studienganges Angewandte Informatik an der  
Dualen Hochschule Baden-Württemberg Mosbach



von  
Tilman Alexander Lorenz

Bearbeitungszeitraum	9 Monate
Matrikelnummer, Kurs	2447899, INF20B
Ausbildungsfirma	Robert Bosch GmbH - Werk Bamberg
Betreuer der Dualen Hochschule	Prof. Dr. Carsten Müller

2. Mai 2023

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>iii</b>
<b>1 Voraussetzungen</b>	<b>1</b>
1.1 Probleme + Lösungen der Installation . . . . .	1
<b>2 YOLO</b>	<b>2</b>
2.1 Generelle Ordnerstruktur . . . . .	2
2.2 Aufbau der Label . . . . .	3
2.3 YAML Klassifizierungsfile . . . . .	4
<b>3 Ausführen des Modells</b>	<b>5</b>
3.1 Docker Setup und Vorbereitungen . . . . .	5
3.2 Daten kopieren . . . . .	5
3.3 YOLO Modell Trainieren . . . . .	6
3.3.1 Befehlserklärung für Training des Modells . . . . .	6
3.3.2 Nach Ausführung . . . . .	8
<b>4 Bilderkennung</b>	<b>9</b>
4.1 Programm zur erkennung . . . . .	9
<b>5 Anleitung für Docker starten</b>	<b>10</b>
5.1 Docker starten . . . . .	10
5.2 Docker Container stoppen und löschen . . . . .	12

# Abkürzungsverzeichnis

**YOLO** You Only Look Once

# Abbildungsverzeichnis

2.1	Ordnerstruktur für einen Datensatz . . . . .	2
2.2	Struktur YOLO Format veranschaulicht . . . . .	3
3.1	Docker Container ausschnitt mit der ID markiert in einem Roten Rahmen	6
5.1	Berechtigungen wie sie meistens voreingestellt sind durch Windows . . . .	11
5.2	Zielzustand Berechtigungen . . . . .	11

# Tabellenverzeichnis

# 1 Voraussetzungen

Für eine erfolgreiche Installation der Umgebung auf dem Testsystem (für Development oder Debugging) sind folgende Sprachen/ Frameworks/ Pakete notwendig:

- Python  $\geq 3.9$
- PyTorch  $\rightarrow$  latest stable Version
- NVIDIA CUDA Toolkit  $\rightarrow$  entsprechende Version (**muss NICHT die neuste sein**)
- (yolov5)

Zum Zeitpunkt des Schreibens ist die neuste PyTorch Version 1.13.1. Dazu muss der entsprechende CUDA Compiler ausgesucht werden. Das wäre in diesem Fall die Version 11.6 und nicht die aktuellste CUDA Version. Es wird empfohlen nicht CUDA selber zu installieren, sondern dies mit PyTorch zusammen installieren. Installiert wird PyTorch zusammen mit CUDA mit:

```
pip3 install torch torchvision torchaudio
--extra-index-url https://download.pytorch.org/whl/cu117
```

Der Link kann je nachdem geändert werden, was für eine CUDA Version benötigt wird. Dies kann sich von der Version des NVIDIA Treibers abhängig sein.

## 1.1 Probleme + Lösungen der Installation

Bei der Installation kann es zu verschiedenen Probleme kommen, die zumindest beim Bearbeiten der Arbeit entstanden sind.

**Problem 1:** Der Dockercontainer Startet nicht. **Lösung:**

1. Das GitHub Repository von ultralytics [**glennjocher.2023**] neu clonen
2. Das Dockerfile von `./utils/docker/Dockerfile` in `./` rein kopieren
3. Dockerfile ausführen (das kann je nach Anwendung eine sehr lange Zeit in Anspruch nehmen aufgrund der Größe)
4. Dockercontainer starten

## 2 YOLO

You Only Look Once (YOLO) ist ein populäres Objekt Erkennungsmodell, dass dafür bekannt ist besonders schnell und akkurat Objekte in Bildern zu erkennen. Dies liegt an der kleinen Modellgröße und den hohen Berechnungsgeschwindigkeiten. Des Weiteren nutzt YOLO das gesamte Bild als Trainingsgrundlage, was es sowohl ermöglicht Videos zu klassifizieren als auch die Fehlerrate reduziert, dass der Hintergrund als Teil des Objektes klassifiziert wird. In diesem Kapitel geht es darum wie die Schnittstelle bzw. der Algorithmus gefüttert werden muss und was der Algorithmus für Datenformate erwartet [Jiang.2022].

Für die Studienarbeit wird der YOLOv5 von ultralytics [glennjocher.2023] genommen, da diese folgende Vorteile bietet:

- aktive Developer am Projekt
- große Community
- Docker deployable
- CUDA- und GPU-Unterstützung
- Pytorch Beschleunigung

### 2.1 Generelle Ordnerstruktur

Der YOLO-Algorithmus erwartet zum funktionieren eine generelle Ordnerstruktur, gezeigt in Abbildung 2.1. Der oberste Ordner ist dabei der Projektnameordner. In diesem müssen alle Projektrelevanten Bilder und Label enthalten sein.

Auch darin enthalten sein muss das *.yaml* file. Das wird in Abschnitt 2.3 noch mal genauer beschrieben. Ansonsten müssen neben dem *.yaml* file noch ein Ordner für die Trainingsdaten enthalten sein. Optional auch ein Testdatenordner und Validierungsdatenordner. Diese müssen wie der Trainingsdatenordner unter dem Projektordner eingeordnet werden.



Abbildung 2.1: Ordnerstruktur für einen Datensatz

## 2.2. AUFBAU DER LABEL



Abbildung 2.2: Struktur YOLO Format veranschaulicht

Der Test, Train und Validordner müssen alle jeweils 2 Ordner haben für images und labels. Diese müssen auch über die verschiedenen Ordner exakt gleich benannt sein. Das heißt auch in der Groß- und Kleinschreibung.

In dem images Ordner sind die vielen Testbilder enthalten, und in Labels die entsprechenden label dazu. Als Bildformate werden von dieser YOLO Version png, jpg und tiff Formate verarbeitet. Jeder Bildname muss einzigartig vorhanden sein (ich empfehle einfach die Bilder durchnummerieren).

Bei den labels werden nur .txt Dateien akzeptiert. Wichtig ist hier, dass die Daten zu einem Bild in einer gleichnamigen .txt-Datei enthalten ist. Wenn als Beispiel ein Bild mit dem Namen *test.jpg* vorhanden ist muss im Ordner labels auch das dazugehörige Labelfile *test.txt* heißen.

## 2.2 Aufbau der Label

Jede Textdatei beinhaltet alle Label, die in dem dazugehörigen Bild zu finden sind. Dabei steht jede Zeile für ein Objekt das auf der Abbildung auffindbar ist. Diese können sich auch überlappen. Eine Zeile ist wie aufgebaut:

```
<class> <x_center> <y_center> <width> <height>
```

Die "class" gibt an, welche Klasse das markierte Objekt hat. Die Klasse und dessen Name wird in dem YAML File definiert, worin auch angegeben wird wie viele Klassen es insgesamt gibt aber das wird in Abschnitt 2.3 noch einmal genauer erklärt. Der *x\_center* gibt den Mittelpunkt des Labels an entlang der x-Achse (horizontal). *y\_center* gibt das gleiche in der Vertikalen Richtung an. Genau kann das in



## 2.3. YAML KLASSIFIZIERUNGSFILE

Abbildung 2.2 sehen. Der Punkt in der Mitte ist definiert durch *x\_center* und *y\_center*. Die komplette Breite und Länge wird entsprechend durch *width* und *height* dargestellt.

Insgesamt werden alle Maße nicht in der px Anzahl dargestellt. Es wird dargestellt als das Verhältnis von der Länge/ Breite zur Gesamtlänge und -breite. Es wird dargestellt als eine *float*-Nummer zwischen 0 und 1. Bei dem *x\_center* und *y\_center* wird zwischen den Punkt und dem Nullpunkt die Länge und Verhältnis zum gesamten Bild genommen in der entsprechenden Dimension.

## 2.3 YAML Klassifizierungsfile

Für jedes Projekt gibt es im Basis Folder ein *.yaml* File mit einem selbstgewählten Namen. Dieses File besteht aus drei Teilen. Diese sind wie folgt aufgebaut:

```
path: <path-to-yolo>
train: <rel-path-to-train-dir>
val: <rel-path-to-val-dir>

names:
  0: <name>
  ...
```

<rel-path-to-train-dir> ist der relative Pfad zum Trainingsdatensatz. Wenn der Standardtrainingssatzaufbau wie in Abbildung 2.1 aufgesetzt ist, dann ist dies *./train/*. Der Pfad für den Validierungsdatensatz <rel-path-to-val-dir> ist ähnlich zu dem Trainingsdatensatzpfad. Beim Standardformat wäre es *./valid/*. Der Pfad gibt den momentanen Pfad an. Dieser wird durch die API ersetzt und kann standardisiert mit einem „“ angegeben.

Unter *names* werden die Namen aller relevanten Klassen angegeben. Dies sind die Klassen der jeweiligen Label und deren Bedeutung. Das heißt für ein Label einer Roten und gelben Ampel die geradeaus fährt würde es beispielsweise heißen *0: rot-gelb-geradeaus*. Alle Label werden aufsteigend nummeriert. Die jeweilige Nummer in dem YAML File muss mit der jeweiligen Nummer in den *.txt* Files übereinstimmen. Wenn die Nummer dieselbe ist, geht der YOLO-Algorithmus aus, dass die jeweilige Nummer des Labels den Text beschreibt der im *.yaml* File steht.

## 3 Ausführen des Modells

### 3.1 Docker Setup und Vorbereitungen

Damit der Docker Container funktioniert muss vorher Docker auf der Maschine installiert sein. Für weitere Informationen bitte Die Docker Dokumentation aufrufen: *Siehe hier*.

Wenn der Dockercontainer installiert wurde, kann der Dockercontainer auf 2 Weisen installiert werden. Entweder das folgende Kommando:

```
docker pull ultralytics/yolov5
```

Damit wird vom Dockerhub der ultralytics/yolov5 container gezogen und auf dem eigenen System installiert. Dies ist vor allem ratsam, wenn das Modell nicht geändert wird und der Algorithmus nicht geändert wird. Dabei wird es empfohlen dem *Quickstart* von ultralytics zu folgen.

Die zweite Möglichkeit ist den Dockercontainer selber aufzubauen. Dies wurde bereits in Abschnitt 1.1 erklärt. Hierbei ist der Vorteil, dass man noch zusätzlich den Dockercontainer beeinflussen kann und andere Anwendungen installieren oder den Dockercontainer mit anderen Anwendungen verknüpfen kann.

Wenn der Dockercontainer auf dem Zielsystem läuft, können die Daten kopiert werden.

### 3.2 Daten kopieren

Die Daten müssen wie in Kapitel 2 vorliegen und auf demselben System liegen, wo auch der Docker Container läuft. Dann kann man anschließend die Daten mit dem folgenden Befehl kopieren:

```
docker cp <local_path> <docker_container_id>:<path_in_container>
```

Erklärung:

- <local\_path> ist der Pfad in dem System zu dem Basisordner des Datenpakets. Das Datenpaket ist ein Projekt beschrieben wie in Kapitel 2. Dieser Pfad sollte

### 3.3. YOLO MODELL TRAINIEREN

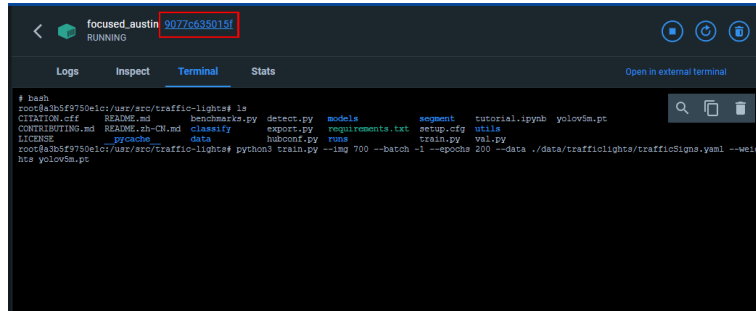


Abbildung 3.1: Docker Container ausschnitt mit der ID markiert in einem Roten Rahmen

den Basisordner mit einfügen. Wenn der Pfad zum Basisordner `/usr/src/test-application/data` ist und der Basisordner heißt *traffic*, dann ist der local\_path `/usr/src/test-application/data/traffic`

- `<docker_container_id>` ist die ID des Containers. Dies ist nicht der Name. Zu sehen ist die, wenn man auf den Container klickt und sich die Details des Containers anzuschauen siehe Abbildung 3.1
- `<path_in_container>` dies ist der Pfad innerhalb des Containers am besten ist dies der Pfad des Working directories + *data*

Nach dem Kopieren der Daten muss das *.yaml*-file im Base Folder noch unbenannt werden. Da der YOLO Algorithmus annimmt, dass das Beschreibungsskript nicht immer im selben Ordner liegt wie die Daten muss der Pfad zu den Daten in das *.yaml*-file eingeschrieben werden und das momentane Directory anstelle des `/` eingesetzt werden.

## 3.3 YOLO Modell Trainieren

Um das Modell zu trainieren sollte in den Docker Container gewechselt werden und das Terminal geöffnet werden. Mit dem Kommando *bash* wird in Bash gewechselt und es kann gesehen werden in welchem Ordner man sich befindet.

### 3.3.1 Befehlserklärung für Training des Modells

Für das Training des Modells muss in das Base Directory von dem YOLOv5 Repository gewechselt werden. Darin befindet sich die *train.py* datei. Hierbei kann man das Training mit dem folgenden Befehl starten:

```
python train.py --img <size_of_img> --batch <size_of_batches> --epochs <epochs>
```

### 3.3. YOLO MODELL TRAINIEREN

```
--data <data_path_to_yaml> --weights <weights> [--device <device_numbers> ]
```

Die in eckickegen Klammern stehende Begriffe sollen nun nochmal erklärt werden:

- `<size_of_img>` definiert die Breite des Bildes in Pixel. **Achtung:** Je höher der Pixelcount, desto höher der RAM / VRAM Verbrauch aber der Anstieg ist nicht linear, sondern Exponentiell
- `<size_of_batch>` gibt an wie viele Bilder in einem Durchgang (Epoche) analysiert werden soll. Diese kann per Hand vorgegeben werden, allerdings passiert es dann, dass aufgrund unzureichenden Videospeichers das Training abbricht. Für eine dynamische Anpassung des Videospeichers -1 eingeben. Allerdings kann es sein, dass dann alle Ressourcen genutzt werden.
- `<epochs>` gibt an wie oft trainiert wird. Für jede neue epoche werden die nächsten Batch an Images analysiert bis alle Bilder analysiert sind und der Algorithmus beginnt wieder von vorne. Somit wird sichergestellt, dass bei einer hohen Anzahl an Bildern alle Bilder untersucht werden.
- `<weights>` Für das Training gibt es 5 verschiedene Gewichte: Nano, Small, Medium, Large und XLarge. Für die jeweiligen Gewichte muss folgendes eingegeben werden. Es sei zu beachten, dass mit zunehmender größe die Rechendauer und der Platz an benötigten Arbeitsspeicher massiv zunimmt.
  - Nano → *yolov5n.pt*
  - Small → *yolov5s.pt*
  - Medium → *yolov5m.pt*
  - Large → *yolov5l.pt*
  - XLarge → *yolov5x.pt*
- `<device_numbers>` dies ist auch für die Arbeit ein optionaler Parameter. Dieser gibt an welche GPUs genutzt werden sollen. Wird, das ausgelassen geht der Algorithmus von einer CPU Berechnung aus und nutzt den vorliegenden Arbeitsspeicher und die CPU. Damit GPU(s) genutzt werden können müssen die Nummern der GPUs angegeben werden. Diese wird durch das Betriebssystem bestimmt. Wenn nur eine GPU vorhanden ist, ist dies i.d.R. 0. Damit wird der komplette VRAM der GPU genutzt. Voraussetzung dafür ist es, dass das NVIDIA CUDA Toolkit installiert ist und der GPU für den Docker freigeschaltet ist, was jedoch der default ist. Für mehrere GPUs einfach mit Kommata die GPUs auflisten. Für mehr Informationen *[hier klicken](#)*.

Falls diese Arbeit älter ist ein Jahr oder die bisherigen Angaben funktionieren nicht sei an dieser Stelle auf das Repository von ultralytics verwiesen, die eine ausführliche Dokumentation des Codes haben und all den wichtigen Befehlen.

### 3.3. YOLO MODELL TRAINIEREN

#### 3.3.2 Nach Ausführung

Nach der Ausführung sollte das Programm im ersten Schritt alle Trainingsdaten einlesen und anschließend je Epoche das Netz trainieren. Egal ob es fertig trainiert hat oder nicht wird in aufsteigender Reihenfolge im Working Directory unter `./data/` zu finden sind.

Wenn das Trainieren fehlgeschlagen sein sollte enthält der Ordner keine Gewichte. Gewichtsdateien sind mit der Endung `.pt` gekennzeichnet.

Bei erfolgreichem TRainieren werden zwei Dateien Abelegt. Einmal *latest.pt* und *best.pt* die auf den jeweils zuletzt ausgeführten run und den besten run hinweisen. Diese können dann für die spätere Objekterkennung bzw. Ampel und Traffic Signs Erkennung genutzt werden.

## 4 Bilderkennung

Zur Nutzung der Bilderkennung des Yolo Algorithmus vor allem mit eigenständigen Labeln müssen die vorherigen beschriebenen Schritte beachtet werden. Nun können Bilder, Videos oder andere Objekte eingespeist werden und werden erkannt.

### 4.1 Programm zur erkennung

Um Modelle erkennen zu können gibt es zwei Wege dies zu tun. Für beide wird das in Unterabschnitt 3.3.2 beschrieben *best.pt* genutzt. Die eine Möglichkeit ist das von YOLOv5 bereitgestellte *detect.py* file zu nutzen. Dazu den Kommentaren in der *offiziellen Dokumentation* folgen.

Die zweite Möglichkeit, die auch genutzt werden kann um multiple Videos oder custom Schnittstellen zu programmieren ist diese Auswertung selber zu programmieren. Dazu sei für den Start an die ausführliche Dokumentation in der *YOLOv5 Dokumentation* verwiesen.

Nach abgeschlossenem Erkennung werden die Ergebnisse als Bilddatei in das Directory *./* abgespeichert. Auch hier werden die erkannten Daten in einem Ordner abgespeichert der sich mit mehreren Erkennungssessions inkrementell erhöht. Gleichzeitig wird ein Objekt zurückgegeben, indem die erkannten Objekte inklusive deren Zuversichtlichkeiten errechnet durch den YOLO Algorithmus zurückgibt.

# 5 Anleitung für Docker starten

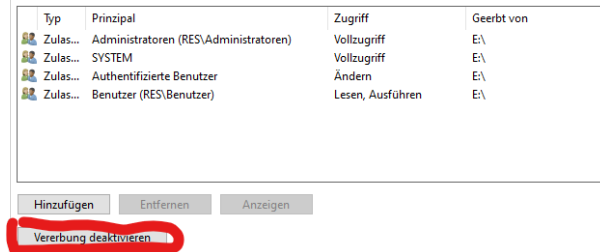
## 5.1 Docker starten

1. Verbinden mit HPC
2. Unter `/mnt/data/outside-data/` Ordner mit Namen Projekt anlegen für Studierenden (`mkdir <Projektname>`)
3. Studierenden privaten Key vom Nutzer `docker_user` geben
4. Studierende kopiert Daten auf `/mnt/data/outside-data/` → die folgenden Schritte beziehen sich auf ein Windows Betriebssystem, für Linuxsysteme sind die ähnlichen Schritte nur die Berechtigungsschritte werden per `chown 700 <user><key>`
  - a) Studierende speichert Private Key auf PC ab
  - b) Verbindung des PCs mit Mosbach VPN (Lehre Netz)
  - c) Rechtsklick auf private Key im File Explorer → Properties → Sicherheit → erweitert → sollte Berechtigung sehen wie in Abbildung 5.1
  - d) klicken auf Vererbung Deaktivieren
  - e) Hinzufügen klicken → SYSTEM suchen → Lese- und Execute und Schreibrechte geben → Ok klicken
  - f) Hinzufügen klicken → <eigenen Nutzer>suchen → Lese- und Execute und Schreibrechte geben → Ok klicken
  - g) Danach sollte Berechtigung aussehen wie in Abbildung 5.2
  - h) Kommandozeile öffnen → wechseln in Directory wo Key legt
  - i) Folgendes Kommando eingeben (alles eine Zeile)

```
scp -i <keyfile Name> -P 2022 <Pfad zu Datenablage>  
docker_user@193.197.11.229:/mnt/data/outside-data/<Name des Projektes>
```

- j) Information: <Pfad zu Datenablage>ist der Pfad zur Base Directory → alles darin wird rüberkopiert / <Name des Projektes>ist ein vorher angelegter Ordner, der den Namen des Projektes trägt, Password eingeben → wird von

## 5.1. DOCKER STARTEN

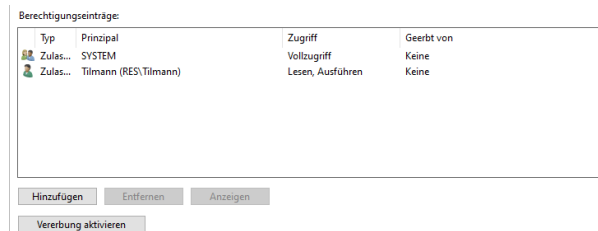


Typ	Prinzipal	Zugriff	Geerbt von
Zulas...	Administratoren (RES\Administratoren)	Vollzugriff	E\
Zulas...	SYSTEM	Vollzugriff	E\
Zulas...	Authentifizierte Benutzer	Ändern	E\
Zulas...	Benutzer (RES\Benutzer)	Lesen, Ausführen	E\

Hinzufügen Entfernen Anzeigen

Vererbung deaktivieren

Abbildung 5.1: Berechtigungen wie sie meistens voreingestellt sind durch Windows



Typ	Prinzipal	Zugriff	Geerbt von
Zulas...	SYSTEM	Vollzugriff	Keine
Zulas...	Tilmann (RES\Tilmann)	Lesen, Ausführen	Keine

Hinzufügen Entfernen Anzeigen

Vererbung aktivieren

Abbildung 5.2: Zielzustand Berechtigungen

Herrn Müller übergeben

5. Dockercontainer installieren → wenn YOLOv5 bitte unten den Anweisungen folgen
6. Client schließen und laufen lassen

### DIE FOLGENDEN SCHRITTE BEZIEHEN SICH AUSSCHLIEßLICH AUF YOLOv5

1. Entsprechenden Dockercontainer Pullen per Kommando: `docker pull ultralytics/yolov5:latest`
2. eingeben Kommando unten (alles eine Zeile)

```
docker run --ipc=host -it --gpus all --memory="<memory limit>"  
--cpus="<cpu limit>" -v  
/mnt/data/outside-data/<projektname>/:/usr/src/datasets  
ultralytics/yolov5:latest
```

- <memory limit> wird eingegeben wie viel GB der Container nutzen darf, dabei immer nur das Vorzeichen Gigabyte → g / Megabyte → m
- <cpu limit> gibt an wie viele CPUs maximal genutzt werden dürfen vom Container als Gleitkommazahl → 2 CPUs entspräche 2.0
- sowohl die gpus flag, memory flag als auch cpu flag dürfen weg gelassen werden, dann läuft der Container auf der gesamten Hardware



## 5.2. DOCKER CONTAINER STOPPEN UND LÖSCHEN

3. wechseln in directory datasets und wechseln bis Base File des *.yaml* gefunden wurde
4. entsprechenden Base Path ändern um dem Pfad im Docker container zu repräsentieren
5. wechseln in app directory
6. eingeben Kommando: siehe Abschnitt 3.3 und nach Angabe Studierenden
7. client schließen

Für weitere Information zu Yolov5 starten: [Link zu Yolov5 Starten](#)

## 5.2 Docker Container stoppen und löschen

Zum Stoppen des Containers kann der folgende Befehl gesetzt werden:

```
docker stop <container name>
```

Der <Container name> kann durch den Befehl *docker ps* angezeigt werden. Dabei wird in der letzten Spalte der Containername angezeigt und in der ersten Spalte die id. Stop funktioniert mit der Id.

Mit dem Befehl *docker rm <Container id>* kann dann der Container gelöscht werden