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
Tilmann Alexander Lorenz

Bearbeitungszeitraum	01.09.2022 - 16.06.2023
Matrikelnummer, Kurs	2447899, INF20B
Ausbildungsfirma	Robert Bosch GmbH - Werk Bamberg
Betreuer der Dualen Hochschule	Prof. Dr. Carsten Müller

15. Mai 2023

Inhaltsverzeichnis

Abkürzungsverzeichnis

YOLO You Only Look Once Algorithm

HPC High Performance Computer

Abbildungsverzeichnis

Tabellenverzeichnis

1 Voraussetzungen

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

- Python >= 3.9
- PyTorch → latest stable Version
- NVIDIA CUDA Toolkit → entsprechende Version (**muss NICHT die neuste sein**)
- You Only Look Once Algorithm (YOLO)v5

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.

2 YOLO

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**] genommenm, da diese folgende Vorteile bietet:

- aktive Developer am Projekt
- große Community
- Docker deployable
- CUDA- und GPU-Unterstützung
- Pytorch Beschleungigung
- viele Tutorials
  und Ressourcen für das Erlernen des Algorithmus

2.1 Generelle Ordnerstruktur

Der YOLO-Algorithmus
erwartet eine gewisse Ordnerstruktur.
Diese muss eingehalten werden, damit der Algorithmus funktioniert. Ein Diagramm der Ordnerstruktur
ist zu sehen in Abbildung 2.1. Der hierarchisch gesehen höchste Ordner ist der Projektordner. Darin sollten alle Daten für den YOLO-Algorithmus enthalten sein. Dieser Ordner enthält die Trainings- und Testdateien. Optional kann auch ein Validierungsdatensatz enthalten sein und die *.yaml* Datei.

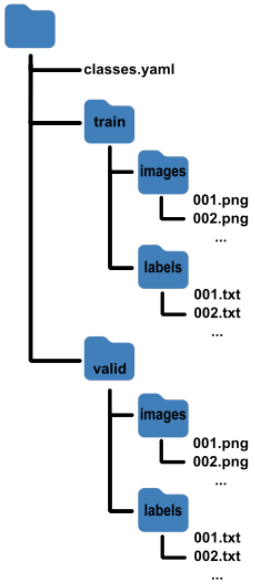

Abbildung 2.1: Ordnerstruktur für einen Datensatz

Es wird empfohlen die *.yaml* Datei in den
Projektordner zu inkludieren. Der Aufbau der Datei
wird in Abschnitt 2.3 genauer beschrieben. Alle Dateien sind innerhalb des Projektordners einzugliedern.

Der Test, Train und Validordner müssen alle jeweils 2 Ordner besitzen mit dem Namen *images* und *labels*. Diese müssen auch über die verschiedenen Ordner exakt gleich, inklusive der Groß- und Kleinschreibung benannt sein.

Der *images* Ordner enthält die entsprechenden Bilder. Zum Trainieren akzeptiert der YOLO-Algorithmus unter anderem die Bildformate *png,jpg* und *tiff*. Innerhalb des Ordners müssen alle Bilder einen einzigartigen Namen besitzen (es wird an dieser Stelle empfohlen, alle Bilder durchzunummerieren)

Der *labels*-Ordner enthält die Label zu den entsprechenden Bildern in *images*. Dabei müssen alle Label für ein entsprechendes Bild in einer gleichnamigen Textdatei mit der Endung *.txt* unter dem Ordner *labels* abgespeichert werden. Ein Beispiel: In dem Trainingsordner im Unterordner *images* gibt es ein Bild mit dem Namen *test.jpg*. Die dazugehörige Labeldatei wird im Ordner *labels* unter dem Namen *test.txt* gespeichert.

Im folgenden Abschnitt sollen nun der Aufbau einer solchen Labeldatei erläutert werden.

2.2 Aufbau der Labeldatei

Eine Labeldatei ist wie in Abschnitt 2.1 definiert eine Textdatei mit entsprechendem Dateiname. Es wird empfohlen die Datei UTF-8 zu enkodieren.

Jede Datei besteht aus einer unterschiedlichen Anzahl an Zeilen. Jede Zeile der Datei beschreibt ein Objekt, das in der Abbildung zu sehen ist. Es ist möglich, dass diese Bildbereiche eiinander überlappen. Eine Zeile ist wie folgt aufgebaut:

```
<class> <x-center> <y-center> <width> <height>
```

- <class>gibt die Klasse an des markierten Objektes. Klassen werden durch die *.yaml*-Datei definiert. Gekennzeichnet wird diese durch eine Zahl. Weiteres in Abschnitt 2.3
- <x-center>→ Angabe des Zentrums des Rechteckes in x-Richtung
- <y-center>→ Angabe des Zentrums des Rechteckes in y-Richtung
- <width>→ die Breite des Rechteckes bzw. des markierten Bereiches
- <height>→ die Höhe des Rechteckes bzw. des markierten Bereiches

Abbildung 2.2: Struktur YOLO Format veranschaulicht

Ein Beispiel ist zu sehen in Abbildung 2.2. *x_ center, y_ center, width, height* werden dargestellt als Verhältnis zur Gesamtlänge/-breite. Die Zahlen sind definiert durch $n \in [0, 1]$.

Die Klassen werden im *.yaml* Dateiformat definiert. Diese wird im folgenden Abschnitt genauer beleuchtet.

2.3 YAML Klassifizierungsfile

Für jedes Projekt gibt es im Projektordner eine *.yaml* Datei mit einem beliebigen Namen. Die *.yaml*-Datei muss allerdings einem gewissen Aufbau folgen der folgend gezeigt wird. Dabei muss die Ordnerstruktur eingehalten werden wie in Abbildung 2.1 gezeigt:

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

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

- <path-to-project>→ Absoluter Pfad zum Projektordner (inklusive des Projektordners)
- <rel-path-to-train-dir>→ relativer Pfad zu den Bildern des Trainingsdatensatz ausgehend vom <path-to-project>-Pfad
- <rel-path-to-val-dir>→ relativer Pfad zu den Bildern des Validierungsdatensatz ausgehend vom <path-to-project>-Pfad

- *names* → Liste aller Klassen und der Verbindung zwischen Name der Klasse angegeben durch <name>und der Nummer. Die Nummer, beginnend mit 0 muss auch äquivalent sein zu der Nummerierung in den Textdateien. Der YOLO-Algorithmus assoziiert die Namen mit den Nummern durch die *.yaml*-Datei

3 Anleitung für Docker starten

3.1 Docker starten

1. Verbinden mit High Performance Computer (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 3.1

   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 3.2

   h) Kommandozeile öffnen → wechseln in Directory wo Key legt

   i) Folgendes Kommando eingeben (alles eine Zeile)

```
scp -i <keyfile-name> -P 2022 <path-to-data>
docker_user@193.197.11.229:/mnt/data/outside-data/
<name-project>/
```

   - <path-to-data>→ Pfad zur Base Directory → alles darin wird rüberkopiert

Abbildung 3.1: Berechtigungen wie sie meistens voreingestellt sind durch Windows

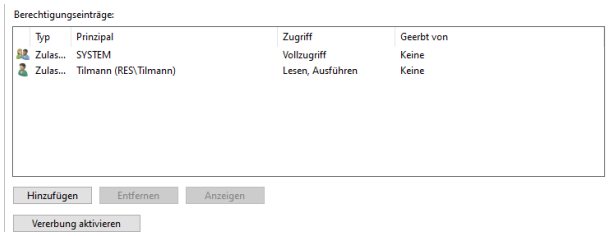

Abbildung 3.2: Zielzustand Berechtigungen

- <name-project>ist ein vorher angelegter Ordner, der den Namen des Projektes trägt, Password eingeben → wird von Herrn Müller übergeben

5. Dockercontainer installieren und laufen lassen

3.2 Docker YOLOv5 Container Setup und Vorbereitungen

Wichtig: Die Umgebung mit YOLO und PyTorch ist schon aufgesetzt mit dem Dockercontainer der von Dockerhub gezogen wird. Insofern keine Änderungen am Algorithmus vorgenommen wurden, reicht dieser Container vollkommen aus und liefert alle nötigen Umgebungswerkzeuge.

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>" --network host -v
/mnt/data/outside-data/<projektname>/:/usr/src/datasets
ultralytics/yolov5:latest
```

- <memory-limit>→ Angabe wie viel RAM der Container nutzen darf → Vorzeichen wichtig: Gigabyte → g / Megabyte → m

- <cpu-limit>→ Angabe der CPU Begrenzung / wie viele CPUs dürfen maximal genutzt werden. Angabe als Gleitkommazahl: 2 CPUs entspräche 2.0
- sowohl die memory flag als auch cpu flag dürfen weg gelassen werden, dann läuft der Container mit den gesammten Resourcen des weg gelassenen Flag die der Rechner zur Verfügung stellt
- wenn der Container nicht die GPU bzw. der YOLO Algorithmus nutzen soll, muss die Flag: *–gpus all* weggelassen werden

3. versichern Sie sich, dass die Daten unter */usr/src/datasets/* vorhanden sind
4. geben sie *exit* ein in die Kommandozeile
5. Eingeben des folgenden Kommandos:

```
docker start <cont_id>; docker exec <cont_id> <yolo_befehl> &
```

- <cont_id>ist dabei die ID des jeweiligen Docker Containers
- <yolo_befehl>Befehl zum Ausführen des Detektionsskriptes siehe Unterabschnitt 3.2.1

6. Jetzt trainiert der Docker Container

3.2.1 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.

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>

--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*.

3.3 Den Docker Contaioner anschauen

Dieser Abschnitt befasst sich damit wie man während des Laufes den Docker Container betreten kann ohne, dass der Trainingsprozess stoppt

1. Docker Container anzeigen lassen mit *docker ps*
2. Konsole des Dockercontainer verbinden *docker attach <cont_id>* → damit wird die Konsolenausgabe des Dockercontainers mit der eigenen verbunden
3. bei Verlassen erst *Strg + P* und anschließend *Strg + Q* drücken

3.4 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. *docker stop* nutzt die ID des Docker Containers.

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

4 Bilderkennung und Leistungsnachweise

Nach der Trainingsphase kann aus dem Verzeichniss */usr/src/app/runs/train/«number_ exp>/weights* die Gewichtsdatei: *best.pt* heraus copiert werden dies kann mit Hilfe des Docker Kommandos *docker cp* realisiert werden. Ahnschließend kann dies wieder auf den lokalen Rechner übertragen werden. Um nun innerhalb eines Videos oder eines Bildes docker die Objekte zu erkennen muss folgender Kommando eingegeben werden. Das Kommando muss ausgeführt werden im Basis Verzeichnis des YOLO Algorithmus. Dies kann entweder innerhalb des Docker Container sein oder auf dem eigenen Rechner, wo man den *YOLOv5*-Algorithmus erhält, indem man das entsprechende Repository von ultralytics klont.

```
python detect.py --source <source> --weights <pfad_zu_gewicht>
--conf <confidence>
```

- <Source>→ ist das Bild/ Video, wobei Objekte erkannt werden. Dieses kann ein Video-, Bild-, oder Datenstream-Format sind. Welche Formate genau unterstütz werden ist in der Dokumentation/ Q&A einzusehen.
- <pfad_to_gewicht>→ ist der Pfad wo *best.pt* abliegt. Dies muss der Absolute Pfad sein. Allgemein gehen alle Gewichte die durch den *YOLOv5*-Algorithmus trainiert wurden
- <confidence>→ definiert die Konfidenz, ab der ein Label erstellt werden soll. Diese wird als Dezimalstelle von conf$\in [0, 1]$ dargestellt
- für weitere Optionen wird an dieser Stelle gebeten die offizielle Dokumentation einzusehen

5 Docker aufsetzen auf dem HPC

Dieses Kapitel beschäftigt sich mit dem aufsetzen der Docker Umgebung.

Disclaimer: Dieses Tutorial ist ein Weg, wie man alles aufsetzen kann. Dabei gibt es mehrere Wege und aufgrund der unterschiedlichen Konfigurationen von Systemen. Ist dies kein allgemeiner Guide. Dies ist ein Weg wie der autor den HPC aufgesetzt hat. Generell werden hier viele Quellen zitiert, da dies für jedes System unterschiedlich ist.

5.1 Installation Docker

Wichtig: Docker funktioniert auf den Ubuntuversionen 20.10 und 22.10. Für weitere Informationen dazu bitte die *Docker Dokumentation* aufrufen.

Für die Installation der Docker Umgebung sollten folgende Schritte befolgt werden:

1. mit *sudo apt-get update* das gesamte System updaten bzw. die Resourcen updaten
2. Docker installieren nach der *Docker Dokumentation*. Dabei ist vor allem wichtig die URLs für das Installationstool *apt*. Für den Fall, dass dies nicht funktioniert
3. mit *sudo docker run hello-world* sollte danach funktionieren und ohne Probleme laufen

Die folgenden Punkte beziehen sich auf verschiedene Einstellungen.

5.1.1 Konfiguration Dockergruppe und zuweisung rechte

Damit manche Leute Dockerbefehle ausführen können, bräuchten diese normalerweise root rechte bzw. sudo rechte. Damit dies nicht gewährleistet werden muss, wird eine Gruppe namens docker erstellt, die Zugriffe haben auf den docker dämon.

Achtung: Es sollten die Personen, die Teil der Gruppe sind sorgfältig ausgeäwählt werden da durch den Dämon indirekt die Personen root Rechte haben bzw. durch Docker Dinge als root ausführen können. Somit ist dies zu vermeiden.

1. Für das konfigurieren folgen Sie der *Anleitung*.
2. Anschließend können sie über den Befehl *sudo usermod -aG docker <user>* einen

Nutzer hinzufügen. Dafür muss <user>ersetzt werden durch den eigentlichen Benutzernamen

3. anschließend können per *docker <command>* Docker Container gestartet, bearbeitet und gestoppt werden

5.1.2 Verschieben der Dockerdaten

Hier wird gezeigt wie die Dockerdaten in ein anderes Verzeichnis geschoben werden können und somit auch die Daten auf eine andere Platte geschoben werden können um evtl. Speicherplatz zu sparen oder große Plattenkapazitäten auszureizen. Dafür ist es zu empfehlen der Anleitung von *GuguWeb* oder *IBM* zu folgen.