ITP-Projekt
Erkennung von angekreuzten und nicht-angekreuzten Kästchen

Christos Tsinaridis, IFD12B

Schritt 1 – Erstellung eines synthetisch generierten Datensatzes:

Um ein Machine Learning Algorithmus zu trainieren, benötigen wir Bilder. Das Modell soll mithilfe dieser Bilder trainiert werden und im Anschluss evaluiert werden.

Leider hatten wir keinen Datensatz mit Bildern von angekreuzten und nicht angekreuzten Kästchen. Aufgrund der knappen Projektzeit, hätte es sehr lange gebraucht diesen händisch selbst zu erstellen. Daher haben wir uns entschieden, diesen Schritt zu automatisieren.

Unter ITP_Projekt/image_generation finden sie das Skript dataset_generation.py. Dieses Skript ermöglicht es, leere und angekreuzte Kästchen automatisiert zu erstellen. Dabei wird ein neuer Ordner erstellt, in dem die Bilder abgespeichert werden.

Bei angekreuzten Kästchen wird ein X auf eine zufällig generierte Position innerhalb des Kästchens generiert. Die Größe des X variiert dabei. Zudem ist es möglich angekreuzte Kästchen in verschiedenen Schriftarten darzustellen.

Wir haben uns der einfachheitshalber dafür entschieden, nur eine Schriftart zu nutzen, da sonst ein größerer Datensatz notwendig wäre und das Modell deutlich länger zum Trainieren bräuchte.

Mit der Funktion create_box_with_X() können nun die Bilder generiert werden. Als Übergabeparamter muss die Anzahl der gewünschten Bilder übergeben werden. Im Anschluss werden die gewünschten Bilder zufällig generiert und im Ordner checkbox_dataset abgespeichert.

Für unsere Modell haben wir 240 Bilder erstellt. Diese haben wir dann in 80% (200 Bilder) Trainingsbilder und 20% (40 Bilder) in Testbilder unterteilt.

Die generierten Bilder müssen in den Ordner data/images/train (Trainingsbilder) und data/images/test (Testbilder) kopiert werden.

Schritt 2 – Bilder labeln:

Damit das Modell die Bilder trainieren kann, müssen diese mithilfe eines Tools gelabelt werden. Das heißt dass wir dem Modell sagen müssen, was es erkennen soll. Das Tool generiert für jedes Bild eine txt-Datei im YOLO Format, mit der Information, wo das Objekt zu finden ist.

Dieser Schritt ist sehr wichtig, denn ohne diese Zusatzinformationen (im englischen auch Annotations gennant) kann das Modell nicht trainiert werden. Außerdem sollte man darauf achten, dass die Markierung der Objekte relativ genau abläuft, damit das Modell so gut wie möglich trainiert werden kann.

In diesem Artikel wird auf das Thema Labelling nochmal genauer eingegangen: Erklärung Labelling

Wurden alle Bilder gelabelt, so müssen diese im YOLO Format exportiert, und in dem Ordner data/labels/train kopiert werden.

Es müssen nur Trainingsbilder gelabelt werden, in unserem Beispiel haben wir 200 Bilder gelabelt.

Die Bilddateien und die Labeldateien müssen denselben Namen haben z.B.:

Bilddatei: Checkbox1.png
Labeldatei: Checkbox1.txt

Sind alle Bilder und Label vorhanden kann das Modell im nächsten Schritt trainiert werden.

Schritt 3 – Modell trainieren:

Bevor das Training gestartet werden kann, muss eine Datei angelegt werden. Diese soll config.yaml benannt werden und im Ordner ITP_Projekt gespeichert werden. In dieser Datei werden die Pfade definiert, die das Modell zum Trainieren benötigt. Außerdem werden die Anzahl der Klassen, die erkannt werden sollen, sowie die Namen der Klassen definiert. Dieser Schritt ist nicht zwingend notwendig, allerdings ratsam, da sonst bei jedem Training die Pfade angegeben werden müssen.

Anschließend muss Ultralytics installiert werden. Im Notebook train.ipynb muss dafür die erste Zelle ausgeführt werden, anschließend wird Ultralytics installiert.

Ich empfehle für diesen Vorgang eine Virtuelle Umgebung über Anaconda oder Python selber zu erstellen. Mehr Informationen über Ultralytics finden sie unter diesen Link: Ultralytics Installation

In der nächsten Zelle kann nun das Training gestartet werden.

Dafür müssen einige Hyperparamter angepasst werden:

- experiment: Name des Experiments, bei mehreren Experimenten ist es ratsam ein Schema zu verwenden
- data_path: Pfad der config.yaml
- n_epochs: Gesamtzahl der Trainingsepochen. Jede Epoche stellt einen vollständigen Durchlauf durch den gesamten Datensatz dar. Die Anpassung dieses Wertes kann sich auf die Trainingsdauer und die Modellleistung auswirken.
- bs (batch size): Stapelgröße für das Training, die angibt, wie viele Bilder verarbeitet werden, bevor die internen Parameter des Modells aktualisiert werden.
- n_workers: Anzahl der Worker-Threads für das Laden von Daten. Beeinflusst die Geschwindigkeit der Datenvorverarbeitung und der Einspeisung in das Modell, besonders nützlich in Multi-GPU-Setups.
- Val: Ermöglicht die Validierung während des Trainings, so dass die Leistung des Modells regelmäßig an einem separaten Datensatz bewertet werden kann.
- Device: Legt die Recheneinheit(en) für das Training fest: eine einzelne GPU (device=0), mehrere GPUs (device=0,1), CPU (device=cpu), oder MPS für Apple-Silizium (device=mps).

Die komplette Auflistung aller Hyperparamter finden sie unter diesem Link: Hyperparamter

Im letzten Schritt muss ein vortrainiertes Modell ausgewählt werden, dass den von uns generierte Datensatz trainieren soll. Wir haben uns entschieden die Yolov8 Modelle zu nutzen, diese sind die neusten Modelle von Ultralytics und versprechen die besten Ergebnisse.

Eine Auflistung aller Modelle finden sie unter diesem Link: Yolov8 Modelle

Nachdem das Modell fertig trainiert wurde, werden die Ergebnisse im Folgenden Ordner abgespeichert: ITP_Projekt/Ergebnisse

Für jedes Modell haben wir einen Ordner erstellt. Um die Performance der Modelle vergleichen zu können haben wir alle Modelle mit denselben Einstellungen trainiert.

Alle Modelle wurden über 20 Epochs mit einer Batch Size von 8 trainiert.

Schritt 4 - Modelle evaluieren:

In dem Ordner Ergebnisse finden sich aller Ergebnisse vom jeweiligen Lauf des Modells. In diesem Ordner sind einige Dateien die Aufschluss auf die Performance des Modells geben.

Dabei sind für uns primär zwei Dateien interessant: results.png & results.csv

In results.png ist ein Bild mit verschiedenen Metriken die die Performance des Modells prüfen:

Box Loss:

Der Box Loss misst die Unterschiede zwischen den Bounding-Boxen, die das Modell vorhersagt, und den tatsächlichen Bounding-Boxen im Trainingsdatensatz. Er ist eine wichtige Komponente in den Optimierungsverfahren während des Trainings von Objekterkennungsmodellen. Ein niedriger Box Loss deutet darauf hin, dass das Modell in der Lage ist, die Positionen und Größen der Objekte im Bild genau zu bestimmen.

Cls Loss:

Der CLS Loss misst die Diskrepanz zwischen den vom Modell vorhergesagten Klassen und den tatsächlichen Klassen der Objekte in den Trainingsdaten. Ein niedriger CLS Loss deutet darauf hin, dass das Modell die Klassen der Objekte in Bildern genau vorhersagt.

DFL Loss:

Der DFL Loss misst, wie gut das Netzwerk in der Lage ist, den räumlichen Kontext zu erfassen, ohne auf eine explizite Objekterkennung angewiesen zu sein.

Precision(B):

Die Präzision konzentriert sich speziell auf die Rate der korrekten positiven Vorhersagen im Verhältnis zu allen vorhergesagten positiven Instanzen. Ein hoher Präzisionswert bedeutet, dass das Modell nur eine geringe Anzahl von falsch positiven Vorhersagen macht, während ein niedriger Präzisionswert darauf hindeutet, dass das Modell viele falsch positive Vorhersagen macht.

Recall(B):

Der Recall konzentriert sich speziell auf die Rate der korrekten positiven Vorhersagen im Verhältnis zu allen tatsächlich positiven Instanzen. Ein hoher Recall-Wert bedeutet, dass das Modell eine hohe Anzahl von tatsächlich positiven Instanzen korrekt identifiziert hat, während ein niedriger Recall-Wert darauf

hinweist, dass das Modell viele tatsächlich positive Instanzen fälschlicherweise als negativ vorhergesagt hat.

mAP@50:

Dies ist der Durchschnitt der Average Precisions über alle Klassen oder Objekte hinweg in einem Datensatz, wobei nur die Top-50-Vorhersagen für jedes Bild berücksichtigt werden.

Eine hohe mAP@50 bedeutet, dass das Modell sowohl eine hohe Präzision als auch einen hohen Recall für die Top-50-Vorhersagen in jedem Bild erzielt. Dies ist ein wichtiger Indikator für die Leistungsfähigkeit eines Objekterkennungsmodells, insbesondere wenn nur eine begrenzte Anzahl von Vorhersagen pro Bild betrachtet wird.

mAP50-95 (mean Average Precision at IoU thresholds 0.5 to 0.95):

Diese Metrik berechnet den Durchschnitt der APs über alle Klassen oder Objekte hinweg, wobei die Objekte basierend auf verschiedenen IoU-Schwellenwerten von 0.5 bis 0.95 bewertet werden.

- Die IoU-Schwelle (Intersection over Union) misst die Überlappung zwischen der vorhergesagten Bounding-Box und der tatsächlichen Bounding-Box eines Objekts. Ein IoU von 1 bedeutet eine perfekte Vorhersage, während ein IoU von 0 bedeutet, dass es keine Überlappung gibt.
-
- Durch die Berücksichtigung eines Bereichs von IoU-Schwellenwerten werden Modelle auf ihre Fähigkeit evaluiert, Objekte mit verschiedenen Überlappungsniveaus genau zu erkennen und zu lokalisieren.

Eine hohe mAP50-95 deutet darauf hin, dass das Modell konsistent gute Leistung bei der Objekterkennung über verschiedene Überlappungsniveaus erzielt.

Diese Metrik ist besonders wichtig, um die Zuverlässigkeit eines Objekterkennungsmodells unter verschiedenen Bedingungen zu bewerten, einschließlich Fällen, in denen Objekte teilweise verdeckt sind oder sich teilweise überlappen.

In results.csv befinden sich die gleichen Informationen, jedoch pro Epoch unterteilt.

Schritt 5 – Testbilder an das trainierte Modell übergeben und Ergebnisse in neuen Ordner abspeichern (Inference):

Nun ist das Modell trainiert und die Testbilder können an das Modell übergeben werden, damit dieses eine Vorhersage des Objektes treffen kann.

In unserem Falls soll erkannt werden, ob das Kästchen angekreuzt ist oder nicht.

Dazu muss die letzte Zelle im train.ipynb Notebook ausgeführt werden. Allerdings müssen davor noch einige Parameter geändert werden:

- Experiment: Pfad unter dem die erkannten Bilder gespeichert werden sollen
- Source: Pfad unter dem die Testbilder abgespeichert sind
- Show: Falls auf True gesetzt: Zeigt eine Diashow der erkannten Bilder, kann auf False gesetzt werden falls nicht gewünscht
- Save: Falls True: Bilder werden gespeichert, kann auf False gesetzt werden falls nicht gewünscht
- Name: siehe experiment

Die Bilder finden sie nun unter dem jeweiligen Ordner.

Fazit:

Alle 3 trainierten YOLO-Modelle konnten eine gute Performance vorweisen. Bei allen Modellen wurden alle Testbilder zuverlässig erkannt.

Während die Trainingszeit für das Yolov8n Modell kurz war (ca. 20 min), haben die anderen Modelle Yolov8s (ca.40min) & Yolov8m (ca. 1Std 15min) deutlich länger gebraucht.

Da wir nur 200 Bilder für das Training benutzt haben, sind die Trainingszeiten relativ gut, möchte man jedoch größere Modelle trainieren (>1000 Bilder) so braucht man einen guten Computer bzw. eine gute Grafikeinheit. Hier empfiehlt sich eine Nvidia Grafikkarte, die CUDA beherrscht.

Leider konnten wir keine GUI mehr erstellen, mit der ein Nutzer interagieren kann. Dafür war die Projektzeit leider zu kurz.

Zudem wurde nur mit einer Schriftart trainiert, möchte man mehrere Schriftarten trainieren, so bräuchte man einen deutlichen größeren Datensatz. Dies resultiert in der oben angesprochenen Problematik.