

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Künstliche Intelligenz

Studienarbeit Deep Vision

von

André Kestler

**Vergleich der YOLOX- und YOLOv8-Modelle für die
Objekterkennung im Kontext des Udacity Self Driving
Car-Datensatzes**

Bearbeitungszeitraum: von 21. Juni 2023
bis 19. Juli 2023

1. Prüfer: Prof. Dr. phil. Tatyana Ivanovska

Inhaltsverzeichnis

1 Einleitung	1
1.1 Aufgabenstellung	1
1.2 Übersicht	1
2 Datensatz	2
2.1 Beschreibung	2
2.2 Klassenaufteilung	2
2.3 Aufbau	3
2.3.1 Rohdaten	3
2.3.2 YOLOX	4
2.3.3 YOLOv8	4
3 Übersicht über die Modelle	5
3.1 YOLOX	5
3.1.1 Architektur	5
3.1.2 Methoden	7
3.1.3 Verlustfunktion	11
3.2 YOLOv8	13
3.2.1 Architektur	13
3.2.2 Methoden	14
3.2.3 Verlustfunktion	15
4 Modellauswertung	17
4.1 Metriken	17
4.2 Vorhersage	18
5 Zusammenfassung und Ausblick	19

Literaturverzeichnis	20
Abbildungsverzeichnis	22
6 Anhang	23
6.1 Ordnerstruktur	23
6.2 Dateien: Datensatz	24
6.3 Dateien: YOLOX	24
6.4 Dateien: YOLOv8	25

Abkürzungsverzeichnis

CSPDarknet	Cross Stage Partial Network Darknet
FPN	Feature Pyramid Network
IoU	Intersection over Union
OTA	Optimal Transport Assignment
PAFPN	Path Aggregation Feature Pyramid Network (Kombination von FPN und PAN)
PAN	Path Aggregation Network
SPP	Spatial Pyramid Pooling
YOLO	You only look once

Kapitel 1

Einleitung

1.1 Aufgabenstellung

Im Rahmen der Vorlesung Deep Vision ist ein Projekt im Themenbereich des Kurses zu bearbeiten. Die Bearbeitung erfolgt als Einzelarbeit. Als Projekt werden zwei YOLO (You only look once) Netzwerke mit dem Udacity Self Driving Car Datensatz trainiert und miteinander verglichen. In der folgenden Arbeit werden YOLOX und YOLOv8 verwendet.

1.2 Übersicht

In Kapitel 2 wird zunächst der Datensatz beschrieben. Dabei wird auf die Klasseneinteilung und die Datenstruktur eingegangen. In Kapitel 3.1 wird das YOLOX-Netzwerk vorgestellt. Dabei wird auf die verwendete Verlustfunktion, die Architektur und die Methoden eingegangen. Kapitel 3.2 beschreibt die verwendete Architektur von YOLOv8. In Kapitel 4 werden die Ergebnisse mit dem Datensatz vorgestellt. Das letzte Kapitel befasst sich mit einer Zusammenfassung über die Arbeit und gibt einen Ausblick über die weitere Bearbeitung. Im Anhang wird beschrieben, wie die angegebenen Skripte verwendet werden, um die Netzwerke selbst zu trainieren.

Kapitel 2

Datensatz

2.1 Beschreibung

Der Udacity Self Driving Car Dataset [16] ist eine umfangreiche Sammlung von Bildern, die von Kameras in Fahrzeugen aufgenommen wurden. Der Datensatz beinhaltet 15000 Samples mit einer Auflösung von 512x512 Pixeln. Er besteht aus Bildern und die zugehörigen Annotationen, die Informationen über die enthaltenen Objekte in der Umgebung enthalten.

Die Bilder in dem Datensatz umfassen verschiedene Szenarien im Straßenverkehr. Darunter befinden sich Stadt- und Landstraßen. Die Bilder wurden bei unterschiedlichen Lichtbedingungen aufgenommen, um ein breite Vielfalt an Situationen in den Daten abzudecken.

2.2 Klassenaufteilung

Die ursprüngliche Klasseneinteilung ist in Abbildung 2.2 dargestellt. Dort lässt sich erkennen, dass insbesondere die Aufteilung der Klasse *Ampel* in Unterkategorien unterrepräsentiert



Abbildung 2.1: Beispielbilder aus dem Datensatz. Quelle: [16]

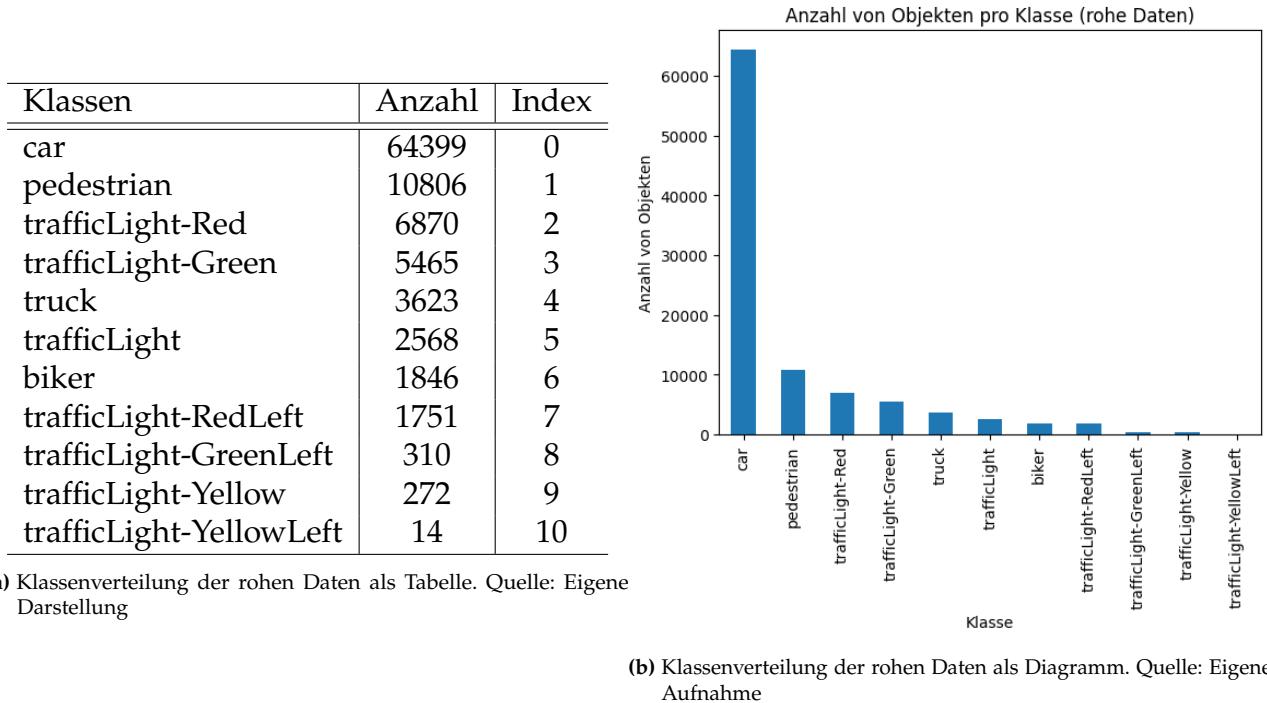


Abbildung 2.2: Klassenverteilung der rohen Daten

ist. Um dieses Problem zu umgehen, wurden die *Ampel*-Klassen zu einer Klasse *trafficLight* zusammengeführt. Für das Training der Modelle wurde der Datensatz mit dem passenden Skript (`datasetPreprocessing.ipynb`) in einen Trainings-, Validierungs- und Testdatensatz aufgeteilt. Diese Aufteilung kann aus den farblichen Säulen in Abbildung 2.3 entnommen werden. Eine weitere Bearbeitung des Datensatzes wurde nicht vorgenommen, um zu überprüfen, wie die verwendeten Netzwerke mit einem unbalancierten Datensatz umgehen.

2.3 Aufbau

2.3.1 Rohdaten

Die Rohdaten sind in einem Ordner gespeichert. Dieser Ordner enthält die Bilder im jpg-Format und eine csv-Datei in der alle Annotationen enthalten sind. Der Header der Datei enthält den Dateinamen des Bildes, die Breite und Höhe, die Klasse und die vier Bounding Box Koordinaten zu jedem Objekt.

Die folgenden Umwandlungen in die Formate werden mit dem Skript `datasetConversion.ipynb` gemacht. Die jeweiligen Ordnerstrukturen können in Kapitel 6 nachgelesen werden.

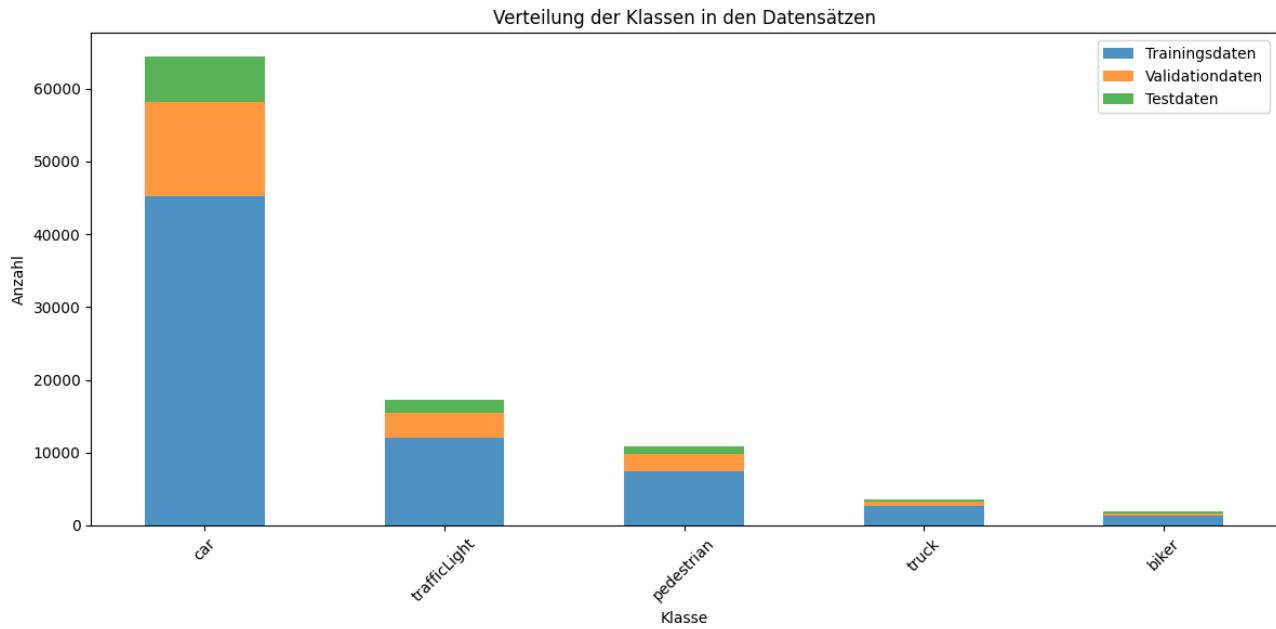


Abbildung 2.3: Aufteilung in Trainings-, Validation- und Testdaten. Quelle: Eigene Aufnahme

2.3.2 YOLOX

Das Netzwerk kann das COCO-Format und das PASCAL VOC-Format verarbeiten. In der folgenden Arbeit wird das COCO-Format verwendet. Zu diesem Zweck werden die Annotationen in einem separaten Ordner und die jeweiligen Bilddateien für Training, Validierung und Test, ebenfalls in einem separaten Ordner abgelegt. Die Annotationen zu den drei Teildatensätzen liegen im json-Format vor. Dabei wird jeder Klasse, jedem Bild und jeder Annotation eine ID zugewiesen, die die Zuordnung der Objekte zu den jeweiligen Bildern ermöglicht.

2.3.3 YOLOv8

Dieses Netzwerk verwendet das YOLO-Format. Dabei werden die Bilder für Training, Validierung und Test in einem separaten Ordner gespeichert. Die dazugehörigen Labels stehen in txt-Dateien. Jedes Bild erhält eine zugehörige Textdatei mit dem gleichen Namen wie das Bild und der Struktur Klasse, x-Zentrum, y-Zentrum, Breite und Höhe. Die Koordinaten müssen auf die Bildgröße normiert sein. Die Labels liegen in einer korrespondierenden Ordnerstruktur. Die zugehörige Konfigurationsdatei gibt anschließend den Pfad zu den Datensatz und die Anzahl der Klassen an.

Kapitel 3

Übersicht über die Modelle

3.1 YOLOX

3.1.1 Architektur

Die YOLOX Architektur besteht aus einem Backbone-Netz, dem Neck und einem Head.

Backbone

YOLOX verwendet das CSPDarknet als Backbone, um Merkmale auf drei verschiedenen Ebenen zu extrahieren. Die Ausgänge haben Dimensionen von $(H/8 \times W/8 \times 256)$, $(H/16 \times W/16 \times 512)$ und $(H/32 \times W/32 \times 1024)$. Diese Skalierungen ermöglichen die Erzeugung von Merkmalen für unterschiedliche Größen von Objekten. Durch die erhöhte Anzahl von Kanälen wird der Informationsverlust in den kleineren Feature-Maps ausgeglichen. Die tiefere Feature-Map (auf der Abbildung 3.1 unten) besitzt ein größeres Receptive Field und ein Pixel kodiert Informationen über einen größeren Bereich des ursprünglichen Bildes.

Das CSPDarknet (Cross Stage Partial) ist eine Modifizierung des ursprünglichen Darknet-Frameworks, das in YOLOv3 schon implementiert wurde. Die Anpassungen bieten Verbesserungen in Bezug auf Geschwindigkeit und Genauigkeit bei der Erkennung von Objekten in Bildern.

CSP steht für Cross Stage Partial Network. Diese Architektur besteht aus einem CSP-Block, der in verschiedenen Stufen des Netzwerks eingefügt ist. Der CSP-Block spaltet den Eingang in zwei Zweige auf, wobei ein Teil unverändert durch den Block läuft und der andere Teil

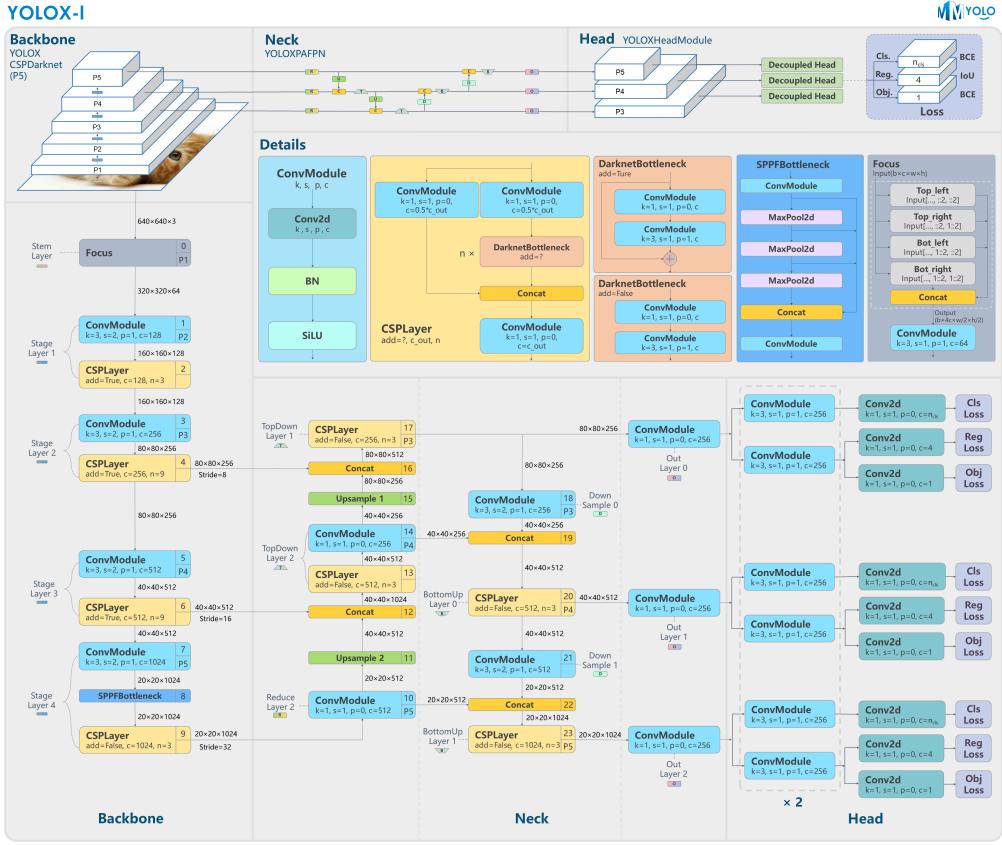


Abbildung 3.1: Übersicht über die Architektur von YOLOX. Quelle: [2, 4, 9]

durch eine Kombination aus Faltungsoperationen und Verbindungsschichten verarbeitet wird. Das Ziel dieser Kombination ist es, eine effizientere Erfassung des Netzes auf den verschiedenen Ebenen zu ermöglichen.

In dem Backbone wird außerdem noch am Ende der Verarbeitung ein SPP verwendet. SPP steht für Spartial Pyramid Pooling. Sie ermöglicht es Objekte unterschiedlicher Größen besser zu erkennen. Dass SPP-Modul teilt das Eingangsbild auf und reduziert die Dimensionen mithilfe von mehreren Pooling Operationen. Die unterschiedlichen Stufen werden anschließend wieder miteinander verbunden und weitergereicht. Ziel dieses Moduls ist es Informationen von verschiedenen Skalierungen zu verbinden, um eine verbesserte Erkennung von Objekten zu gewährleisten. [14]

Neck

YOLOX verwendet im Neck das PAFPN (Path Aggregation Feature Pyramid Network). Dies ist eine Kombination des PAN (Path Aggregation Network) und dem FPN (Feature Pyramid

Network).

Das PAN ist verantwortlich für die Zusammenführung von Informationen aus verschiedenen Netzwerkpfaden und die Integration dieser Informationen in einen einzigen Satz von Merkmalen. Es verbindet die Ausgänge des Backbone-Netzwerks auf unterschiedlichen Skalierungsebenen und passt die Dimensionen durch Upsampling aneinander an. Dadurch soll das Netzwerk ein umfassenderes Verständnis, über die aus dem Backbone generierten Merkmale erhalten.

Das FPN ermöglicht eine robuste Objekterkennung in Bildern unterschiedlicher Skalierungen. Das Netzwerk erzeugt eine Hierarchie von Feature-Maps auf verschiedenen Skalierungen und verbindet sie miteinander. Dadurch sollen feine Details und auch semantische Informationen erfasst werden. FPN verwendet top-down und bottom-up-Verbindungen, um die Merkmale auf verschiedenen Ebenen des Netzwerks zu aggregieren. Die sich daraus ergebende Merkmalspyramide wird an den Head weitergeleitet. [7, 8]

Head

Der Head befindet sich am Ende des Netzwerks und ist für die Vorhersage der Objekte und deren Positionen in den Eingabebildern zuständig. Dort wird die Verlustfunktion berechnet. YOLOX verwendet einen Decoupled Head, der aus zwei Teilen besteht. Dieser Mechanismus ist mit YOLOX neu eingeführt worden und wird in Kapitel 3.1.2 beschrieben.

3.1.2 Methoden

Decoupled Head

Der Decoupled Head trennt die Vorhersage von Objekten und Bounding-Boxes in zwei Zweige auf. Zusätzlich zum Pfad der Bounding-Box wird dort der Konfidenzwert vorhergesagt. Bei den herkömmlichen YOLO-Netzwerken wird die Vorhersage (Klasse, Bounding Box und Konfidenzwert) in einer einzigen Vorhersage gemacht. Dies kann zu Schwierigkeiten bei der Erkennung von Objekten unterschiedlicher Größe führen.

Wie in der Abbildung 3.2 (unten) zu sehen ist, wird im Decoupled Head die Dimension des Eingangs durch eine 1x1-Faltung reduziert und anschließend in zwei Pfade aufgeteilt. Das bedeutet, dass das Modell zuerst die Präsenz von Objekten vorhersagt und in einem parallelen Zweig die Bounding-Box-Koordinaten und den Objektscore für die erkannten Objekte berechnet. Dieser Head wird für jede der drei Neck-Feature-Maps ausgeführt. [10]

Die drei Tensorausgaben von YOLOX enthalten die gleichen Informationen wie die Ausgänge des großen Tensors von YOLOv3:

- Cls: Die Klasse jeder Bounding Box
- Reg: Die 4 Teile der Bounding Box
- Obj: Wie sicher ist das Netzwerk, das innerhalb der Bounding Box ein beliebiges Objekt ist

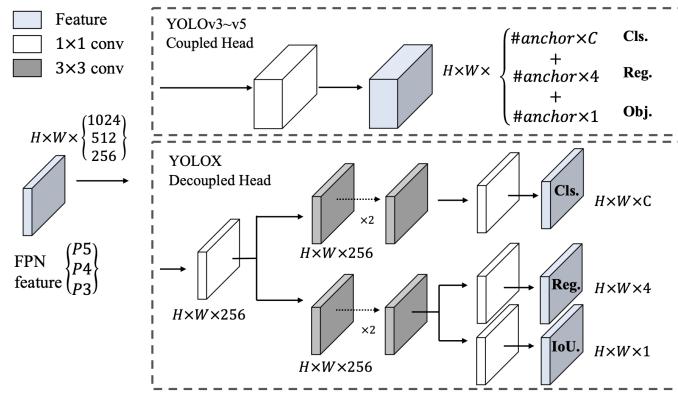


Abbildung 3.2: Illustration des Unterschieds zwischen dem YOLOv3-Head und dem neuen Decoupled-Head. Quelle: [4]

Anchor Free Prediction

Bei der **Anchor-based Prediction** werden vordefinierte Ankerboxen verwendet, um Objekte in verschiedenen Größen zu repräsentieren. Diese Ankerboxen dienen als Referenzpunkte, auf die die Modelle während des Trainings ausgerichtet werden. Das Modell weist der Ground-Truth (Echten) Box die ähnlichste Ankerbox zu und sagt die Verschiebung und Abmessungen der Ankerbox voraus, um sie dem Objekt anzupassen. Dazu muss vor Beginn des Trainings die Skalierung und Anzahl der Ankerboxen vorgegeben werden. Im Grunde ist eine Ankerbox eine Hilfe für das Modell, damit es nicht direkt eine Bounding Box vorhersagen muss.

Die implementierte Methode zur **Anchor-Free Prediction** entstammt ursprünglich aus dem Paper "FCOS: Fully Convolutional One-Stage Object Detection" [15]. Anstelle von Ankerboxen verwendet die Anchor-Free-Methode ein Grid-basiertes Konzept. Im YOLOX Algorithmus wird eine Stride von 32, 16 und 8 verwendet, um die Ausgabebilder des Necks in ein Gitter zu unterteilen. Wenn ein Stride von 32 für ein 256×256 großes Bild verwendet wird, ergeben sich insgesamt $256/32 = 8$ Schnittpunkte in jeder Dimension, also insgesamt

64 Schnittpunkte. Jeder dieser Schnittpunkte heißt Ankerpunkt. Ein Ankerpunkt ist ein Offset, mit dem die (x, y) -Position einer Vorhersage verschoben wird. Die Position des Ankers kann auf dem Bild mit den folgenden Formeln ermittelt werden:

$$x = \frac{s}{2} + s * i, \quad y = \frac{s}{2} + s * j \quad (3.1)$$

Dabei ist s die Schrittweite, i ist der i -te Schnittpunkt auf der x -Achse und j ist der j -te Schnittpunkt auf der y -Achse. Bei YOLOX werden die Gitterpunkte als linker oberer Offset der Bounding Box verwendet. Die folgenden Formeln werden verwendet, um eine vorhergesagte Bounding Box (p_x, p_y, p_w, p_h) auf die tatsächliche Position auf dem Bild (l_x, l_y, l_w, l_h) abzubilden, wenn (x, y) der Schnittpunkt auf dem Gitter ist, zudem die Vorhersage gehört und s die Schrittweite der aktuellen FPN-Ebene ist:

$$l_x = p_x + x, \quad l_y = p_y + y, \quad l_w = s * e^{p_w}, \quad l_h = s * e^{p_h} \quad (3.2)$$

Wir verschieben den vorhergesagten Punkt, indem wir die Vorhersage zum Anker-Punkt (dem (x,y) -Punkt, der dieser Vorhersage zugewiesen ist) hinzufügen. Durch die e -Funktion wird sichergestellt, dass die Höhe und Breite nicht negativ ist und verschieben diese auf der Grundlage der Schrittweite s eines Bildes. Das Beispiel zu dieser Methode kann in der angegeben Quelle nachgelesen werden. [10]

SimOTA Label Assignment

Die Methode soll die Zuordnung der Vorhersagen zu den Ground-Truth-Objekten optimieren, da nicht alle Vorhersagen gut sind und das Modell nicht versuchen soll diese zu optimieren. Dazu werden die Ankerpunkte der vorherigen Methode von 3.1.2 in positive und negative Gruppen aufgeteilt.

OTA (Optimal Transport Assignment) ist ein Ansatz, der das Zuordnungsproblem in der Objekterkennung als Optimal Transport (OT)-Problem formuliert. Es geht darum, den besten Plan zu finden, um Güter (Objekte) von Anbietern (Ground-Truth) zu Nachfragern (Vorhersagen oder Anker) zu minimalen Kosten zu transportieren. OTA verwendet OT, um die Labels den Ankerorten zuzuweisen und die Anker als positiv oder negativ zu kennzeichnen. Der Hintergrund wird als zusätzlicher „Lieferant“ betrachtet.

Das Problem des OTA-Algorithmus ist, dass dieser das Training verlangsamt. Aus diesem Grund gibt es eine Vereinfachung des Algorithmus, indem der optimale Zuweisungsplan

angenähert wird. Vereinfacht funktioniert der Algorithmus folgendermaßen:

- Berechne die Klassen- und Regressionsvorhersage für eine gegebene Eingabe durch das Modell.
- Erstelle einen Liefervektor, der das Angebot (durch Dynamic k estimation festgelegt) für jeden der Ground-Truths repräsentiert.
- Initialisiere den Nachfragevektor mit Einsen für jede Vorhersage.
- Berechne die Kosten für Klassenverluste ($FocalLoss(P^{cls}, G^{cls})$), Regressionsverluste ($IoULoss(P^{cls}, G^{cls})$)) und Zentrumsprior zwischen Vorhersagen und Ground-Truths.
- Berechne die Kosten für den Hintergrund und den Vordergrund basierend auf den Kosten der einzelnen Komponenten.
- Konstruiere eine Kostenmatrix, die die Hintergrund- und Vordergrundkosten enthält.
- Wähle die besten Vorhersagen basierend auf den Kosten und dem verfügbaren Angebot.
- Gebe die ausgewählten Vorhersagen zurück.

Der SimOTA-Algorithmus verwendet das Konzept von Angebot und Nachfrage, um die optimale Zuordnung zwischen Vorhersagen und Ground-Truths zu finden. Durch die Berücksichtigung von Kosten und verfügbaren Angebot werden die besten Vorhersagen ausgewählt, um eine präzisere Objekterkennung zu ermöglichen. Der vollständige Algorithmus kann in der angegebenen Quelle nachgelesen werden. [12]

Advanced Augmentation

YOLOX verwendet Advanced Data Augmentation-Techniken wie Mosaic und MixUp, um die Datenvielfalt während des Trainings zu erhöhen und die Leistungsfähigkeit des Modells zu verbessern.

Beim **Mosaic**-Verfahren werden vier zufällig ausgewählte Bilder aus dem Trainingsdatensatz genommen und zu einem Mosaikbild kombiniert. Dabei werden die Bilder in vier quadratische Bereiche aufgeteilt und zu einem einzigen großen Bild zusammengefügt. Die Bounding Boxes der Objekte werden entsprechend angepasst und die Klassenlabels beibehalten. Mithilfe des CutOuts wird ein beliebiger Teil aus dem Bild ausgeschnitten und die Bounding Boxen noch einmal angepasst. Das Mosaikbild wird dann als Eingabe für das

Modell verwendet. Der Grund für die Verwendung von Mosaic ist, dass es die Fähigkeit des Modells verbessert, mit komplexen Szenarien und Objektüberlappungen umzugehen.

MixUp ist eine Technik, bei der zwei zufällig ausgewählte Bilder und ihre entsprechenden Bounding Boxen und Klassenlabel gemischt werden, indem ihre Annotationen gewichtet kombiniert werden. Das erste Bild wird mit λ und das zweite Bild mit $1 - \lambda$ multipliziert. Die beiden resultierenden Ergebnisse werden addiert. [11]

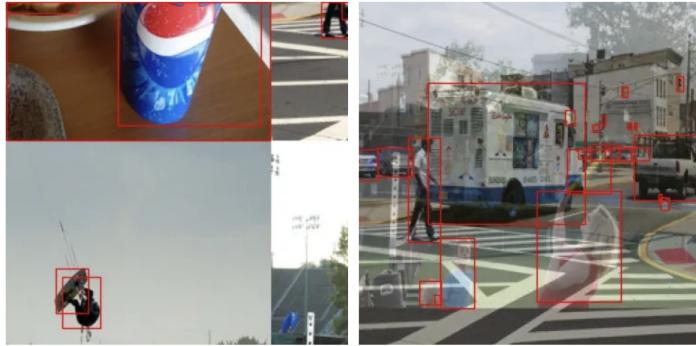


Abbildung 3.3: Beispielbild nach Anwendung der beiden Data Augmentation Methoden (links: Mosaic, rechts: MixUp) Quelle: [11]

3.1.3 Verlustfunktion

Gesamtverlust

Die Gesamtverlustfunktion des Modells setzt sich aus drei einzelnen Verlustfunktionen für die jeweiligen Aufgaben zusammen:

$$\mathcal{L}^{total} = \frac{1}{N_{pos}} * \mathcal{L}^{cls} + \alpha * \frac{1}{N_{pos}} * \mathcal{L}^{iou} + \frac{1}{N_{pos}} * \mathcal{L}^{obj} \quad (3.3)$$

Die Verlustfunktion ist dabei die Summe der einzelnen Komponenten, gemittelt über die Menge der positiven Labels. Die positiven Labels werden durch SimOTA (3.1.2) bestimmt. Der Parameter α ist ein Gewichtungsterm, der von den Autoren des Papers auf 5.0 festgelegt ist.

Klassifikation

Für die Verlustfunktion des Klassifikationsteils, verwendet YOLOX die Binary Cross Entropy (BCE) mit Logits. Dabei handelt es sich um die normale BCE-Funktion mit einer vorgeschal-

teten Sigmoidfunktion für die Prognosen.

Die Formel lautet:

$$\mathcal{L}^{cls} = \frac{1}{N} \sum_i^N y_i * \log(\sigma(\hat{y}_i)) + (1 - y_i) * \log(1 - \sigma(\hat{y}_i)), \quad (3.4)$$

wobei \hat{y} der Vektor der Klassenvorhersagen für C Klassen ist. Die Sigmoidfunktion σ wird verwendet, um die Elemente in einem Wertebereich von [0,1] abzubilden. Die Werte in y sind ein One-Hot-Vektor, der eine 1 für die richtige Klasse und eine 0 für alle anderen Klassen enthält. Die Werte der Vektoren y und \hat{y} stammen aus der positiv gekennzeichneten Menge. Das Ziel des BCE-Verlustes ist, dass das Modell lernt, eine 1 für die richtige Klasse und eine 0 für alle anderen Klassen in der Bounding Box vorherzusagen. [10]

Bounding Box

In YOLOX wird die IoU (Intersection over Union) Loss Funktion verwendet, um die Regressionsverluste für die Bounding Box Koordinaten zu berechnen. Diese Loss Funktion misst die Ähnlichkeit zwischen der vorhergesagten Bounding Box und der Ground Truth Bounding Box anhand des IoU-Werts.

Der IoU-Wert wird berechnet, indem der Flächenanteil des überlappenden Bereichs der Bounding Boxen durch die Summe der Flächen beider Bounding Boxen geteilt wird. Mathematisch kann der IoU-Wert wie folgt ausgedrückt werden:

$$IoU = \frac{\text{Area}_{\text{Intersection}}}{\text{Area}_{\text{Union}}}, \quad IoU \in [0, 1] \quad (3.5)$$

Je höher der IoU-Wert, desto besser ist die Übereinstimmung zwischen der Vorhersage und der Ground Truth Bounding Box.

Die resultierende Verlustfunktion ist folgendermaßen definiert:

$$\mathcal{L}^{iou} = \frac{1}{N} \sum_i^N (1 - IoU_i)^2, \quad (3.6)$$

wobei N die Anzahl der positiven Elemente ist und die Bounding Boxen aus dieser Menge

entstammen. [10]

Objectness

Das Ziel des Objectness Loss in YOLOX ist es, dass das Modell einen Wert nahe 1 hat, wenn es schätzt, dass sich ein Objekt in der Bounding Box befindet, einen Wert nahe 0, wenn es schätzt, dass sich nichts in der Box befindet, und einen Wert dazwischen (vorzugsweise etwa 0.5), wenn es unsicher ist.

Um den Wert zu optimieren wird die Binary Cross Entropy with Logits verwendet, die Funktion die auch in 3.1.3 verwendet wird. Dabei werden die positiven und negativen Labels von SimOTA verwendet.

Für die positiven Vorhersagen wird der IoU-Wert zwischen vorhergesagter und Ground Truth Bounding Box verwendet, um den Wert zu bestimmen, den das Modell vorhersagen soll. Bei negativen Vorhersagen wird der größte IoU-Wert zwischen der vorhergesagten negativen Bounding Box und allen Ground Truth Bounding Boxen verwendet, um die Ground Truth Werte für die negativen Vorhersagen zu bestimmen. [10]

3.2 YOLOv8

3.2.1 Architektur

YOLOv8 basiert ebenfalls auf dem CSPDarknet als **Backbone**-Netzwerk, das auch von YOLOX verwendet wird. Allerdings gibt es einige Unterschiede in der Architektur. Das CSP-Modul besitzt mehr Skip-Connections, das SPP-Bottleneck-Modul ist mit einem CSP-Modul vertauscht worden und das ConvModule am Ende des Backbones wurde entfernt. Im Neck-Bereich des Netzwerks wurden minimale Änderungen an einzelnen Schichten vorgenommen, wie beispielsweise das Vertauschen der Position einiger ConvModule. Genaueres ist aus der Abbildung 3.4 zu entnehmen. Der **Head** des Netzwerks verwendet auch den Decoupled Head mit einer Anchor-Free-Prediction. Innerhalb des Zweigs für die Bounding Box Regression wurde die Verlustfunktion für den Objectness Score entfernt. Ähnlich wie YOLOX verwendet auch YOLOv8 während des Trainings die Mosaic-Augmentierung und die MixUp-Augmentierung. Diese Techniken dienen dazu, die Trainingsdaten zu erweitern und die Robustheit des Modells zu verbessern. [13]

Für die Funktion des Decoupled Head mit der Anchor-Free-Prediction wird auf Seite 7 und

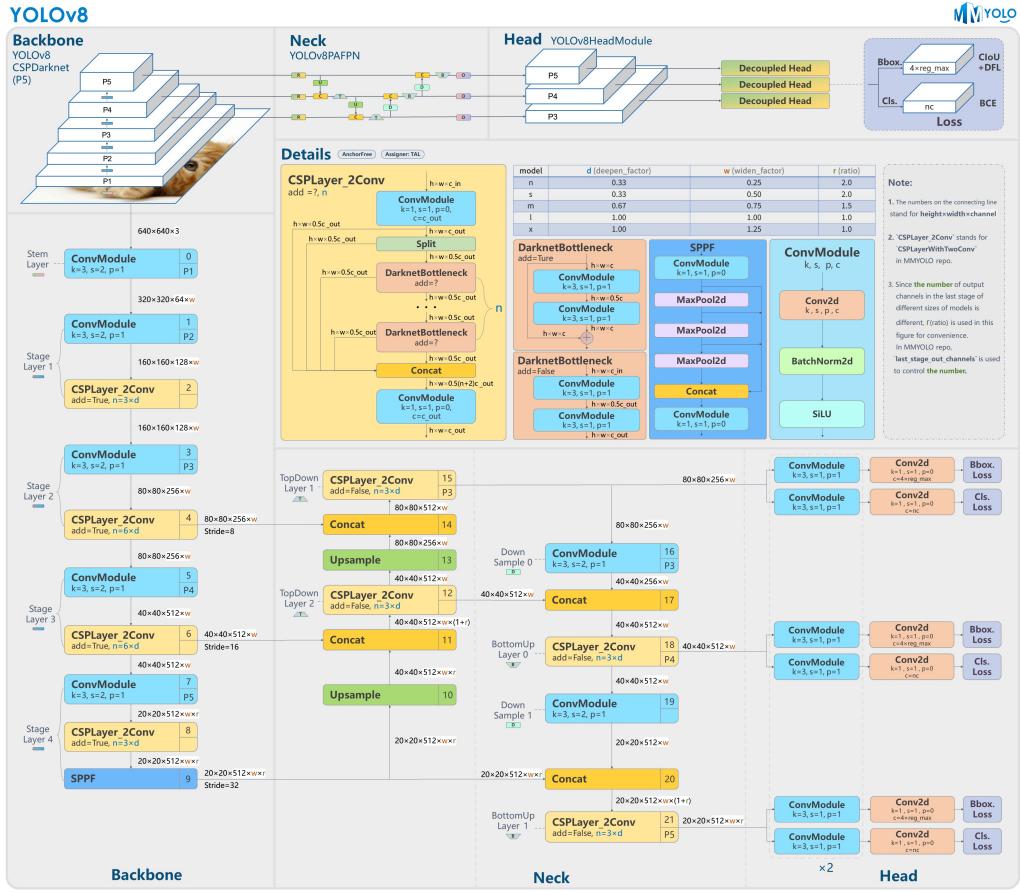


Abbildung 3.4: Übersicht über die Architektur von YOLOv8. Quelle: [2]

Seite 8 verwiesen. Die Datenvorverarbeitung ist unter Kapitel 3.1.2 beschrieben.

3.2.2 Methoden

TOOD

Der YOLOv8-Algorithmus übernimmt die Zuweisungsstrategie von TaskAlignedAssigner, da sie als effektiv angesehen wird. Die TaskAlignedAssigner-Strategie wählt positive Proben, basierend auf gewichteten Klassifizierungs- und Regressionsergebnissen, aus.

Die TaskAlignedAssigner verwendet eine Zuweisungsstrategie, bei der für jede Ground-Truth-Bounding Box ein Ausrichtungsmetrik-Wert für jeden Anker berechnet wird. Dieser Wert wird durch das gewichtete Produkt zweier Werte ermittelt: der vorhergesagten Klassifikationspunktzahl der entsprechenden Klasse und dem Intersection over Union (IoU) zwischen der vorhergesagten Bounding Box und der Ground-Truth-Bounding Box. Anschließend werden für jede Ground-Truth-Bounding Box die größten Top-k-Proben basierend auf

den Werten der Ausrichtungsmetriken, direkt als positive Proben ausgewählt. [3, 13]

3.2.3 Verlustfunktion

Gesamtverlust

YOLOv8 verwendet im Klassifikationszweig das BCELoss und im Regressionszweig eine Kombination aus dem Distribution Focal Loss und dem CIoU Loss. Die drei Teilfunktionen werden anschließend gewichtet summiert:

$$\mathcal{L}^{total} = \lambda_{cls} * \mathcal{L}^{cls} + \lambda_{iou} \mathcal{L}^{iou} + \lambda_{dfl} * \mathcal{L}^{dfl}, \quad (3.7)$$

wobei λ_{cls} auf 7.5, λ_{iou} auf 0.5 und λ_{dfl} auf 1.5 gesetzt ist.

Klassifikation

Für die Verlustfunktion der Klassifikation verwendet YOLOv8 die Binary Cross Entropy (BCE) mit Logits. Dabei handelt es sich, um die BCE-Funktion mit einer vorgeschalteten Sigmoidfunktion für die Prognosen. Die Formel lautet wie auch bei YOLOX:

$$\mathcal{L}^{cls} = \frac{1}{N} \sum_i^N y_i * \log(\sigma(\hat{y}_i)) + (1 - y_i) * \log(1 - \sigma(\hat{y}_i)), \quad (3.8)$$

Die Parameter können in Kapitel 3.1.3 nachgelesen werden. Die positiven Lables werden allerdings nicht durch SimOTA, sondern durch TOOD berechnet.

Regression

Die Verlustfunktion für den Regressionzweig besteht aus zwei Teilen. Der erste Teil ist das Complete-IoU-Loss (CIoU). Der CIoU-Verlust führt im Vergleich zum IoU-Verlust zwei neue Konzepte ein:

- Konzept des Mittelpunktabstandes, das den Abstand zwischen dem tatsächlichen Mittelpunkt der Bounding Box und der vorhergesagten Bounding Box berechnet.
- Konzept des Seitenverhältnisses, indem es die Seitenverhältnisse der tatsächlichen Box, mit den Seitenverhältnissen der vorhergesagten Boxen vergleicht.

Die Formel lautet:

$$\mathcal{L}^{iou} = \sum_i^N (1 - CIoU) * weight, \quad CIoU = IoU - \frac{d^2}{c^2} + \alpha * v \quad (3.9)$$

$$v = \frac{4}{\pi} (\arctan(\frac{w^{gt}}{h^{gt}}) - \arctan(\frac{w^{dt}}{h^{dt}})), \quad \alpha = \frac{v}{(1 - IoU) + v} \quad (3.10)$$

Dabei ist d^2 die euklidische Distanz zwischen den Mittelpunkten der Bounding Boxen der Ground Truth (gt) und Vorhersage (dt). Der Parameter c^2 ist die Diagonale der kleinsten Box, die beide Bounding Boxen umschließt. Der Parameter v entsteht aus den Seitenverhältnissen der Bounding Boxen und α ist ein Gewichtungsparameter, der eine Funktion des IoUs ist. [1]

Der zweite Teil des Regressionszweigs ist das Distributed Focal Loss. Die Formel für das DFL lautet:

$$\mathcal{L}^{df l} = \frac{1}{N} \sum_i^N (CE(predDist_i, tl_i) * wl_i + CE(predDist_i, tr_i) * wr_i) \quad (3.11)$$

Das DFL verwendet eine Verteilung der Bounding Boxen, um die Vorhersage genauer anzupassen. Anstatt nur eine Box zu erzeugen, wird eine Verteilung von möglichen Bounding Boxen berücksichtigt. Die Verteilung wird durch die Ankerpunkte und einen maximalen Regressionswert definiert. Der Regressionswert legt fest, in welchen Bereich die Boxen liegen sollen. Anschließend wird die Cross-Entropy-Funktion (CE) mit dem linken Bereich der Zielverteilung und mit dem rechten Bereich der Zielverteilung berechnet. Die Ergebnisse werden mit den für die Seiten spezifischen Gewichten multipliziert. Eine detaillierte Beschreibung der Verlustfunktion kann im Paper nachgelesen werden. [6]

Kapitel 4

Modellauswertung

4.1 Metriken

Der mAP (mean Average Precision) ist eine Metrik, mit der die Leistung von Modellen zur Objekterkennung bewertet werden kann. Der mAP-Wert wird für verschiedene IoU-Schwellenwerte berechnet. mAP@0.5:0.95 deckt einen großen Bereich von IoU-Schwellen ab. Der mAP@0.5 gibt an, wie gut das Modell Objekte bei einer IoU-Schwelle von 0.5 erkennt. mAP@0.75 gibt die Leistung bei einer höheren IoU-Schwelle von 0.75 an. Diese Metriken bewerten die Erkennungs- und Lokalisierungsgenauigkeit des Modells bei verschiedenen Überlappungsschwellen (IoU-Werten). Sie dienen dem Vergleich und der Bewertung der Modelle. Ein höherer mAP-Wert bedeutet eine bessere Leistung bei der Erkennung von Objekten mit dieser spezifischen Überlappungsschwelle.

Die beiden Tabellen 4.1 und 4.2 liefern eine Bewertung der Leistung von YOLOX und YOLOv8 anhand verschiedener Metriken. Die erste Tabelle basiert auf dem Validierungsdatensatz, der während des Trainings zur Validierung verwendet wurde, während die zweite Tabelle den unabhängigen Testdatensatz zeigt, der zu Beginn der Arbeit abgespalten wurde.

Bei dem Vergleich der Gesamtleistung des Modells (Klasse: all) ist YOLOv8 in allen Werten um 2 % besser als YOLOX. Die relativ ähnlichen Werte sind wahrscheinlich auf die ähnliche Architektur der beiden Modelle zurückzuführen. Die Erhöhung der Leistung ist auf die Verwendung des DFL zurückzuführen.

	mAP@0.5:0.95		mAP@0.5		mAP@0.75	
	YOLOX	YOLOv8	YOLOX	YOLOv8	YOLOX	YOLOv8
all	0.484	0.501	0.776	0.793	0.508	0.528
car	0.567	0.607	0.845	0.873	0.633	0.684
pedestrian	0.348	0.342	0.690	0.67	0.307	0.297
trafficLight	0.473	0.499	0.784	0.831	0.465	0.501
truck	0.628	0.646	0.864	0.877	0.726	0.743
biker	0.403	0.412	0.696	0.713	0.412	0.416

Tabelle 4.1: Vergleich der mAP Werte für den Validierungsdatensatz. Quelle: Eigene Darstellung

	mAP@0.5:0.95		mAP@0.5		mAP@0.75	
	YOLOX	YOLOv8	YOLOX	YOLOv8	YOLOX	YOLOv8
all	0.484	0.498	0.776	0.794	0.508	0.523
car	0.568	0.609	0.838	0.873	0.638	0.691
pedestrian	0.365	0.36	0.724	0.703	0.304	0.316
trafficLight	0.459	0.493	0.770	0.817	0.465	0.505
truck	0.594	0.612	0.83	0.835	0.665	0.696
biker	0.431	0.414	0.732	0.741	0.484	0.406

Tabelle 4.2: Vergleich der mAP Werte für den Testdatensatz. Quelle: Eigene Darstellung

4.2 Vorhersage

In der Abbildung 4.1 ist auf der linken Seite die Ausgabe von YOLOX dargestellt und auf der rechten Seite die Ausgabe von YOLOv8. Beide Netzwerke sind mit dem gleichen Testdatensatz durchlaufen, um die Bounding Boxen auf den Bildern zu erzeugen.



Abbildung 4.1: Vorhersage der Netzwerke auf vier Beispielbildern aus dem Testdatensatz (links: YOLOX, rechts: YOLOv8). Quelle: Eigene Aufnahme

Kapitel 5

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Vergleich zwischen den Modellen YOLOX und YOLOv8 anhand des Udacity Self Driving Car Datensatzes durchgeführt. Beide Modelle weisen eine ähnliche Architektur auf und benutzen ähnliche Methoden zur Verarbeitung der Daten (gleiche Datenvorverarbeitung, Labelzuweisung, Decoupled Head, Anchor-Free-Detection). Der Vorteil von YOLO8 besteht darin, dass es einfach anzuwenden ist, indem es mit pip das *ultralytics*-Paket installiert.

Für zukünftige Arbeiten wird empfohlen, den Datensatz anzupassen, um eine ausgewogene Verteilung der Klassen zu erreichen. Derzeit gibt es eine übermäßige Anzahl von Objekten der Klasse *car*. Durch eine Anpassung des Datensatzes kann die Leistung der Modelle weiter verbessert werden, insbesondere wenn es darum geht, andere Objektklassen korrekt zu erkennen.

Insgesamt bieten sowohl YOLOX als auch YOLOv8 vielversprechende Ansätze für die Objekterkennung im Bereich des autonomen Fahrens. Durch weitere Optimierungen und Anpassungen des Datensatzes können die Ergebnisse noch weiter verbessert werden, um eine zuverlässige und genaue Erkennung von Verkehrssituationen zu ermöglichen.

Literaturverzeichnis

- [1] CHANDRA, Prakash: *IoU Loss Functions for Faster and More Accurate Object Detection.* <https://learnopencv.com/iou-loss-functions-object-detection/#ciou-complete-iou-loss>, Abruf: 13. Juli 2023
- [2] CONTRIBUTORS, MMYOLO: *mmyolo*. <https://github.com/open-mmlab/mmyolo>, Abruf: 10. Juli 2023
- [3] FENG, Chengjian ; ZHONG, Yujie ; GAO, Yu ; SCOTT, Matthew R. ; HUANG, Weilin: TOOD: Task-aligned One-stage Object Detection. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), S. 3490–3499
- [4] GE, Zheng ; LIU, Songtao ; WANG, Feng ; LI, Zeming ; SUN, Jian: YOLOX: Exceeding YOLO Series in 2021. In: *arXiv preprint arXiv:2107.08430* (2021)
- [5] KESTLER, Andre: *deepVision-project*. <https://github.com/AndreKest/deepVision-project>. Version: 2023
- [6] LI, Xiang ; WANG, Wenhui ; WU, Lijun ; CHEN, Shuo ; HU, Xiaolin ; LI, Jun ; TANG, Jinhui ; YANG, Jian: Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection. In: *ArXiv abs/2006.04388* (2020)
- [7] LIN, Tsung-Yi ; DOLLÁR, Piotr ; GIRSHICK, Ross B. ; HE, Kaiming ; HARIHARAN, Bharath ; BELONGIE, Serge J.: Feature Pyramid Networks for Object Detection. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), S. 936–944
- [8] LIU, Shu ; QI, Lu ; QIN, Haifang ; SHI, Jianping ; JIA, Jiaya: Path Aggregation Network for Instance Segmentation. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), S. 8759–8768
- [9] MEGVII-BASEDETECTION: YOLOX. <https://github.com/Megvii-BaseDetection/YOLOX>. Version: 2021

- [10] MONGARAS, Gabriel: *YOLOX Explanation — How Does YOLOX Work?* <https://medium.com/mlearning-ai/yolox-explanation-how-does-yolox-work-3e5c89f2bf78>, Abruf: 12. Juli 2023
- [11] MONGARAS, Gabriel: *YOLOX Explanation — Mosaic and Mixup For Data Augmentation.* <https://medium.com/mlearning-ai/yolox-explanation-mosaic-and-mixup-for-data-augmentation-3839465a3adf>, Abruf: 12. Juli 2023
- [12] MONGARAS, Gabriel: *YOLOX Explanation — SimOTA For Dynamic Label Assignment.* <https://medium.com/mlearning-ai/yolox-explanation-simota-for-dynamic-label-assignment-8fa5ae397f76>, Abruf: 12. Juli 2023
- [13] OPENMMLAB: *Dive into YOLOv8: How does this state-of-the-art model work?* <https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1>, Abruf: 12. Juli 2023
- [14] SHAH, Deval: *YOLOv4 — Version 3: Proposed Workflow.* <https://medium.com/visionwizard/yolov4-version-3-proposed-workflow-e4fa175b902>, Abruf: 13. Juli 2023
- [15] TIAN, Zhi ; SHEN, Chunhua ; CHEN, Hao ; HE, Tong: FCOS: Fully Convolutional One-Stage Object Detection. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), S. 9626–9635
- [16] ZHANG, Edward: *Udacity Self Driving Car Dataset.* <https://www.kaggle.com/datasets/sshikamaru/udacity-self-driving-car-dataset>, Abruf: 23. Juni 2023

Abbildungsverzeichnis

2.1	Beispielbilder aus dem Datensatz	2
2.2	Klassenverteilung der rohen Daten	3
2.3	Aufteilung in Trainings-, Validation- und Testdaten	4
3.1	Übersicht über die Architektur von YOLOX	6
3.2	Illustration des Unterschieds zwischen dem Yolov3-Head und dem neuen Decoupled-Head	8
3.3	Beispielbild nach Anwendung der beiden Data Augmentation Methoden . .	11
3.4	Übersicht über die Architektur von YOLOv8	14
4.1	Vorhersage der Netzwerke auf vier Beispielbildern aus dem Testdatensatz . .	18
6.1	Ordnerstruktur des GitHub Repositories	23
6.2	Ordnerstruktur des src/dataset Ordners.	24
6.3	Ordnerstruktur des src/dataset Ordners.	24
6.4	Ordnerstruktur des src/dataset Ordners.	25

Kapitel 6

Anhang

Das gesamte Projekt liegt öffentlich auf meinem privaten GitHub-Account. [5]

Um PreTrained Gewichte und die Gewichte des Trainings herunterzuladen mus git lfs installiert sein. Die Schritte schauen folgendermaßen aus:

- git clone <repo>
- cd repo_name
- git lfs pull

6.1 Ordnerstruktur

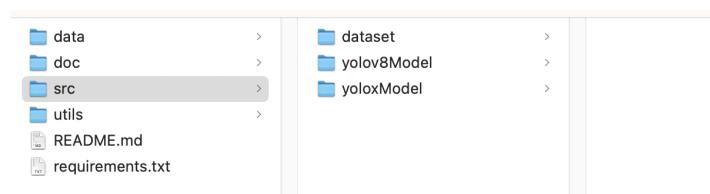


Abbildung 6.1: Ordnerstruktur des GitHub Repositories. Quelle: Eigne Aufnahme

- data: Darin liegen die Datensätze (in GitHub nicht mit hochgeladen).
 - dataFiltered: Der gefilterte Datensatz (nicht annotierte Bilder gelöscht, Klassen zusammengefügt, Train/Val/Test split) mit *datasetPreprocessing.ipynb*
 - dataRaw: Der unbearbeitete Datensatz [16]
 - dataYolov8: Der Datensatz im Format für YOLOv8 mit *datasetConversion.ipynb*

- dataYoloX: Der Datensatz im Format für YOLOX mit *datasetConversion.ipynb*
- doc: Dort liegt die Dokumentation im LaTex-Format
- src: Dort liegen die Skripte (Datenvorverarbeitung, YOLOX, YOLOv8)
- utils: GitHub Repository von YOLOX
- requirements.txt: Datei für die Installation der Pakete mit *pip*

6.2 Dateien: Datensatz

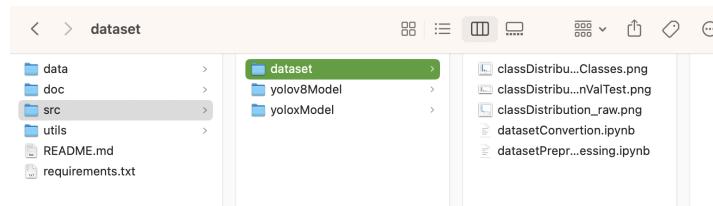


Abbildung 6.2: Ordnerstruktur des src/dataset Ordners. Quelle: Eigne Aufnahme

- datasetConversion.ipynb: Skript zur Umwandlung der Daten in die Formate der Netzwerke
- datasetPreprocessing.ipynb: Skript zur Vorverarbeitung des Datensatzes

6.3 Dateien: YOLOX

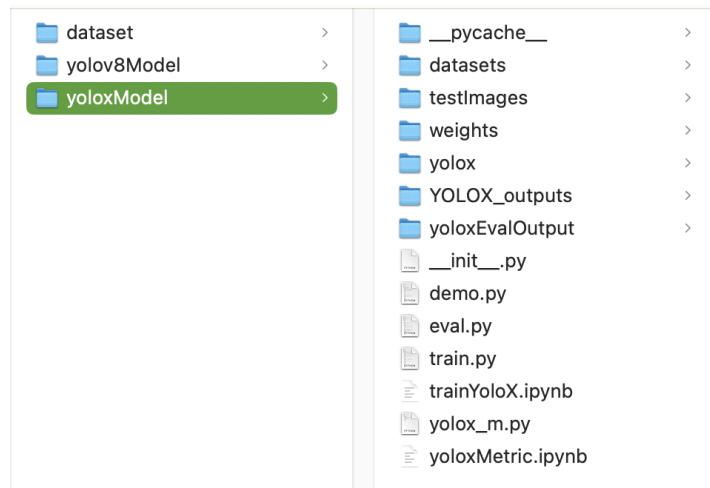


Abbildung 6.3: Ordnerstruktur des src/dataset Ordners. Quelle: Eigne Aufnahme

- datasets: Dort liegt der Datensatz im YOLOX Format (Kopie von ./dataset/dataYoloX)
- testImages: Vier Testdateien aus dem Testdatensatz
- weights: Initialisierungsgewichte aus COCO-Training
- yolox: Das yolox-Netzwerk aus den GitHub Repository [9]
- YOLOX_ouputs: Ausgabe nach dem Training (Gewichte, Log, Visualisierung, ...)
- yoloxEvalOutput: Ausgabe im json-Format für die Evaluierung
- demo.py: Um das Netzwerk auszuführen auf Eingabebild (aus dem GitHub Repository [9])
- eval.py: Um das Netzwerk zu evaluieren (aus dem GitHub Repository [9])
- train.py: Um das Netzwerk zu trainieren (aus dem GitHub Repository [9])
- trainYoloX.ipynb; Sammlung der Befehle um Netzwerk zu trainieren/evaluieren/auszuführen
- yolox_m.py: Konfiguration des Netzwerks
- yoloxMetric.ipynb: Metriken zur Evaluierung berechnen

6.4 Dateien: YOLOv8

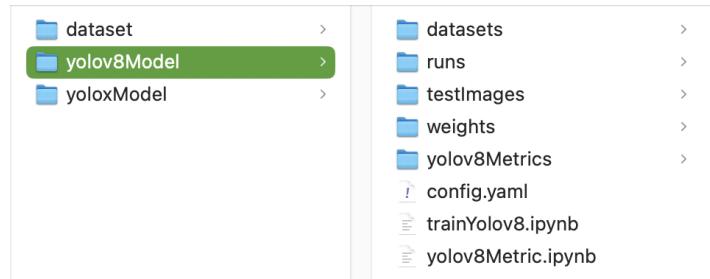


Abbildung 6.4: Ordnerstruktur des src/dataset Ordners. Quelle: Eigne Aufnahme

- datasets: Dort liegt der Datensatz im YOLOv8 Format (Kopie von ./dataset/dataYolo8)
- runs: Ausgabe nach dem Training (Gewichte, ...)
- testImages: Vier Testdateien aus dem Testdatensatz

- weights: Initialisierungsgewichte aus COCO-Training
- yolov8Metrics: Dateien zur Evaluierung des Netzwerks
- config.yaml: Konfiguration des Netzwerks
- trainYolov8.ipynb: Training des Netzwerks
- yolov8Metric.ipynb: Metriken zur Evaluierung berechnen