

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 Modell 1: YOLOX	5
3.1 Architektur	5
3.1.1 Backbone	6
3.1.2 Neck	6
3.1.3 Head	7
3.2 Methoden	7
3.2.1 Decoupled Head	7
3.2.2 Anchor Free Prediction	8
3.2.3 SimOTA Label Assignment	9
3.2.4 Advanced Augmentation	10
3.3 Verlustfunktion	11
3.3.1 Klassen	11
3.3.2 Klassifikation	12
3.3.3 Bounding Box	12

3.3.4 Objectness	13
3.4 Modellauswertung	13
4 Modell 2: YOLOv8	14
4.1 Architektur	14
4.2 Verlustfunktion	15
4.3 Modellauswertung	15
5 Zusammenfassung und Ausblick	16
Literaturverzeichnis	17
Abbildungsverzeichnis	19
6 Anhang	20
6.1 Ordnerstruktur	20
6.2 Dateien: Datensatz	20
6.3 Dateien: YOLOX	20
6.4 Dateien: YOLOv8	20

Abkürzungsverzeichnis

CSPDarknet	Cross Stage Partial Network Darknet
FPN	Feature Pyramide 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 wird das YOLOX-Netzwerk vorgestellt. Dabei wird auf die verwendete Verlustfunktion, die Architektur und die Ergebnisse mit dem Datensatz eingegangen. Kapitel 4 beschreibt die verwendete Architektur von YOLOv8. Außerdem werden die Ergebnisse anhand des verwendeten Datensatzes präsentiert. Das letzte Kapitel Zusammenfassung und Ausblick befasst sich mit dem direkten Vergleich der Ergebnisse 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 [11] 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: [11]

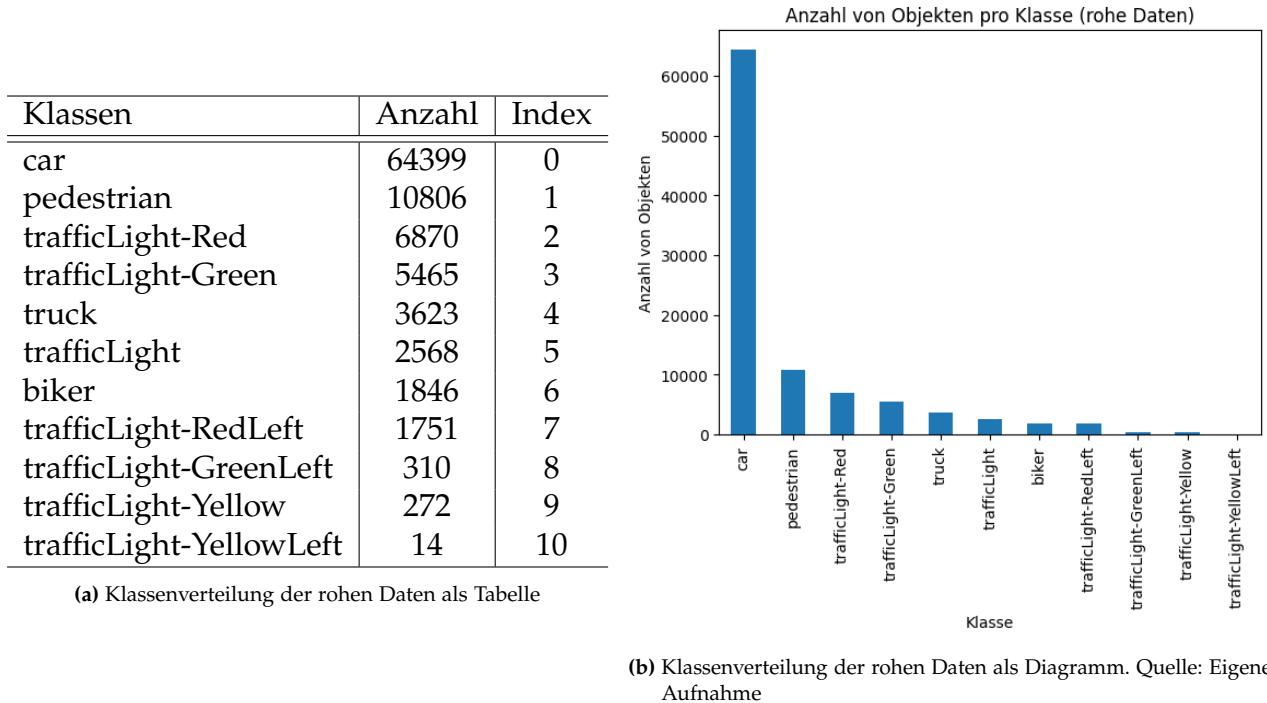


Abbildung 2.2: Klassenverteilung der rohen Daten

ist. Um dieses Problem zu umgehen, wurden die *Ampel*-Klassen zu einer Klasse *trafficLight* zusammengeführt. Für das Auswerten des Datensatzes wurde dieser 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.

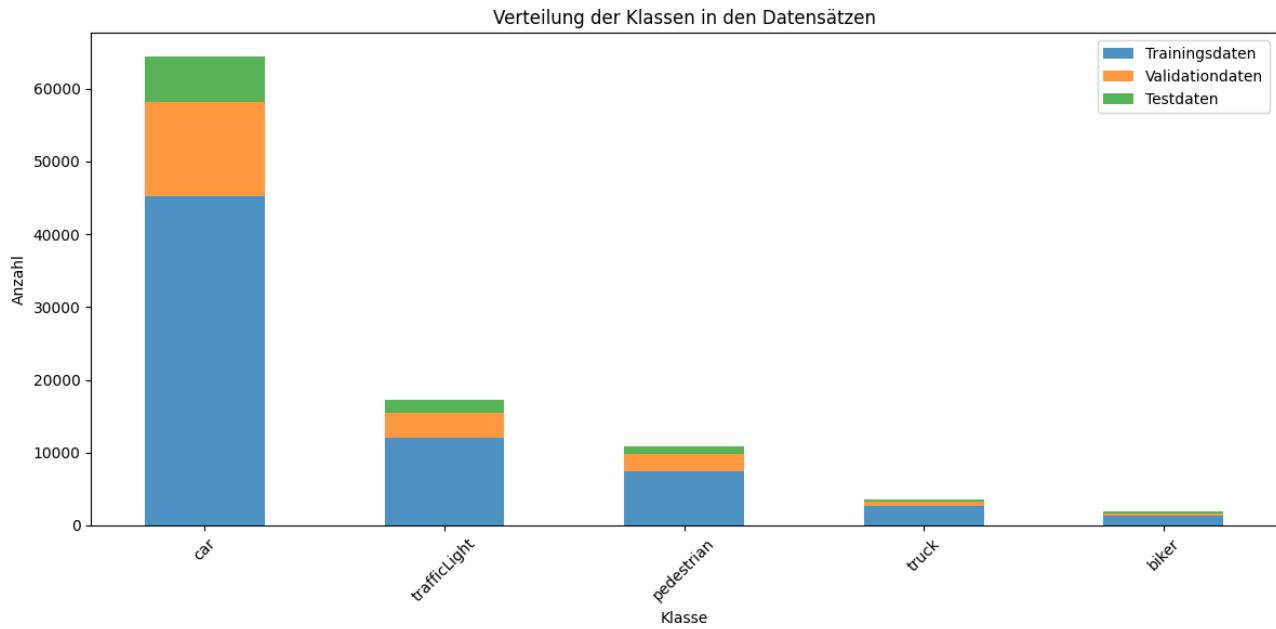


Abbildung 2.3: Aufteilung in Trainings-, Validation- und Testdaten. Quelle: Eigene Aufnahme

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 einzelnen .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

Modell 1: YOLOX

3.1 Architektur

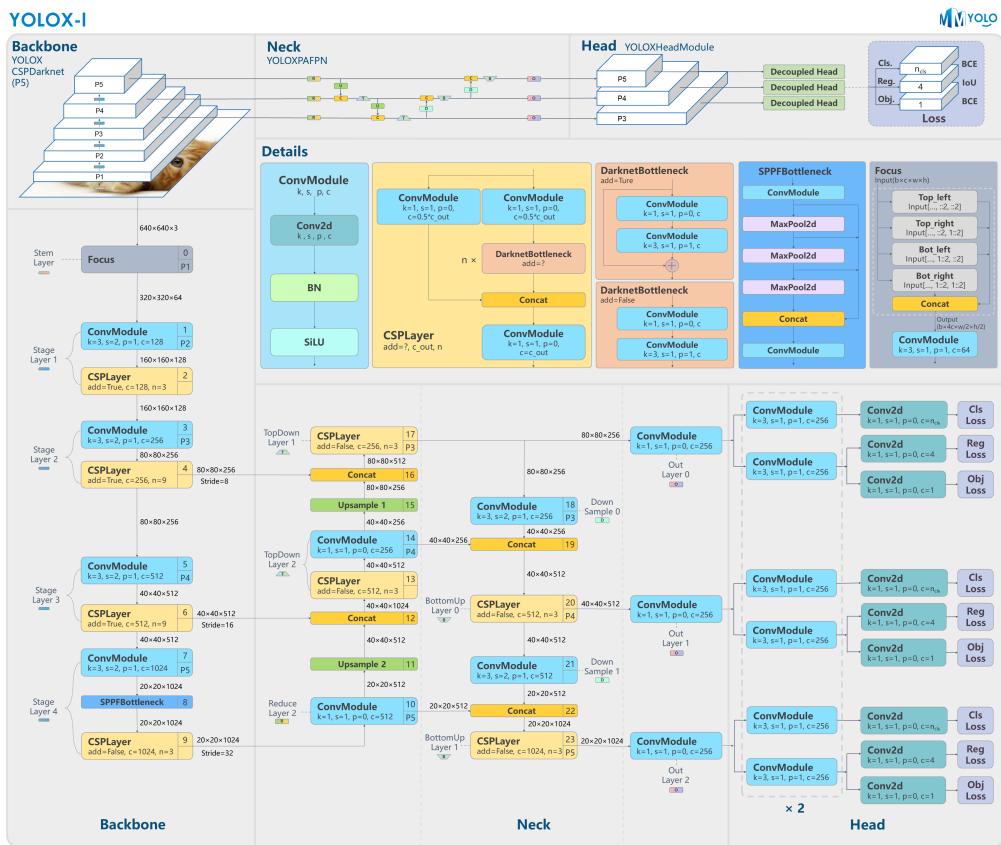


Abbildung 3.1: Übersicht über die Architektur von YOLOX. Quelle: [1, 2, 5]

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

3.1.1 Backbone

YOLOX verwendet das CSPDarknet als Backbone, um Merkmale auf drei verschiedenen Maßstäben 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 durch eine Kombination aus Faltungsoperationen und Verbindungsschichten verarbeitet wird. Ziel der Kombination ist, dass das Netzwerk auf unterschiedlichen Ebenen des Netzwerks effektiver zu erfassen.

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 Modul ist es Informationen von verschiedenen Skalierungen zu verbinden, um eine verbesserte Erkennung von Objekten zu gewährleisten. [9]

3.1.2 Neck

YOLOX verwendet im Neck das PAFFN (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 das Zusammenführen von Informationen aus verschiedenen Netzwerkpfaden und die Integration dieser Informationen in einen einzigen Merkmalssatz.

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 mit diesem Verfahren entstehende Merkmalspyramide wird an den Head weitergegeben. [3, 4]

3.1.3 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 3.2.1 beschrieben.

3.2 Methoden

3.2.1 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. [6]

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

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

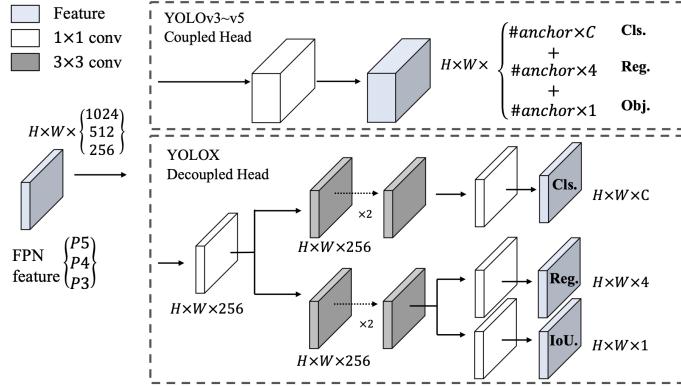


Abbildung 3.2: Illustration des Unterschieds zwischen dem YOLOv3-Head und dem neuen Decoupled-Head Quelle: [2]

3.2.2 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 weißt 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" [10]. 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 Anker-Punkte. 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 angegebenen Quelle nachgelesen werden. [6]

3.2.3 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 Anker-Punkte der vorherigen Methode von 3.2.2 in positive und negative Gruppen aufgeteilt. Die positive Gruppe beinhaltet ein Objekt und die negative Gruppe beinhaltet kein Objekt.

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 verfügbarem Angebot werden die besten Vorhersagen ausgewählt, um eine präzisere Objekterkennung zu ermöglichen. Der vollständige Algorithmus kann in der angegebenen Quelle nachgelesen werden. [8]

3.2.4 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 Anwendung von Mosaic liegt darin, dass es

die Fähigkeit des Modells verbessert, mit komplexen Szenarien und Überlappungen von Objekten umzugehen

MixUp ist eine Technik, bei der zwei zufällig ausgewählte Bilder und ihre entsprechenden Bounding Boxes 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. [7]

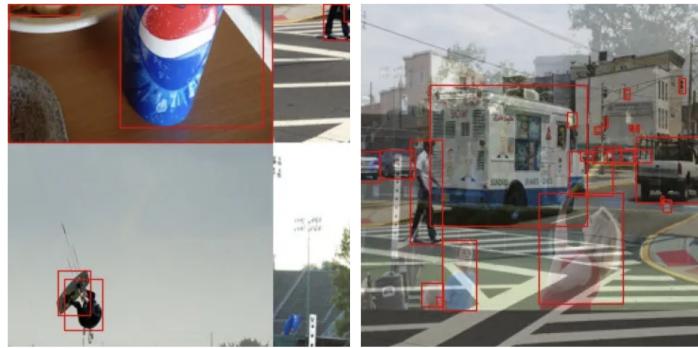


Abbildung 3.3: Beispielbild nach Anwendung der beiden Data Augmentation Methoden (links: Mosaic, rechts: MixUp) Quelle: [7]

3.3 Verlustfunktion

3.3.1 Klassen

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

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

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

3.3.2 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 vorgeschalteten 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] zu 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. [6]

3.3.3 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:

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

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

Die resultierende Verlustfunktion wird folgendermaßen definiert:

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

wobei N die Anzahl der positiven Elemente ist und die Bounding Boxen aus dieser Menge entstammen. [6]

3.3.4 Objectness

Das Ziel des Objectness Loss in YOLOX ist es, dass das Modell einen Wert nahe 1 hat, wenn es glaubt, dass sich ein Objekt in der Bounding Box befindet, einen Wert nahe 0, wenn es glaubt, 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 3.3.2 verwendet wird. Dabei werden die positiven und negativen Labels von SimOTA verwendet.

Für die positive 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 Bounding Box für die negative Vorhersage zu bestimmen. [6]

3.4 Modellauswertung

Kapitel 4

Modell 2: YOLOv8

4.1 Architektur

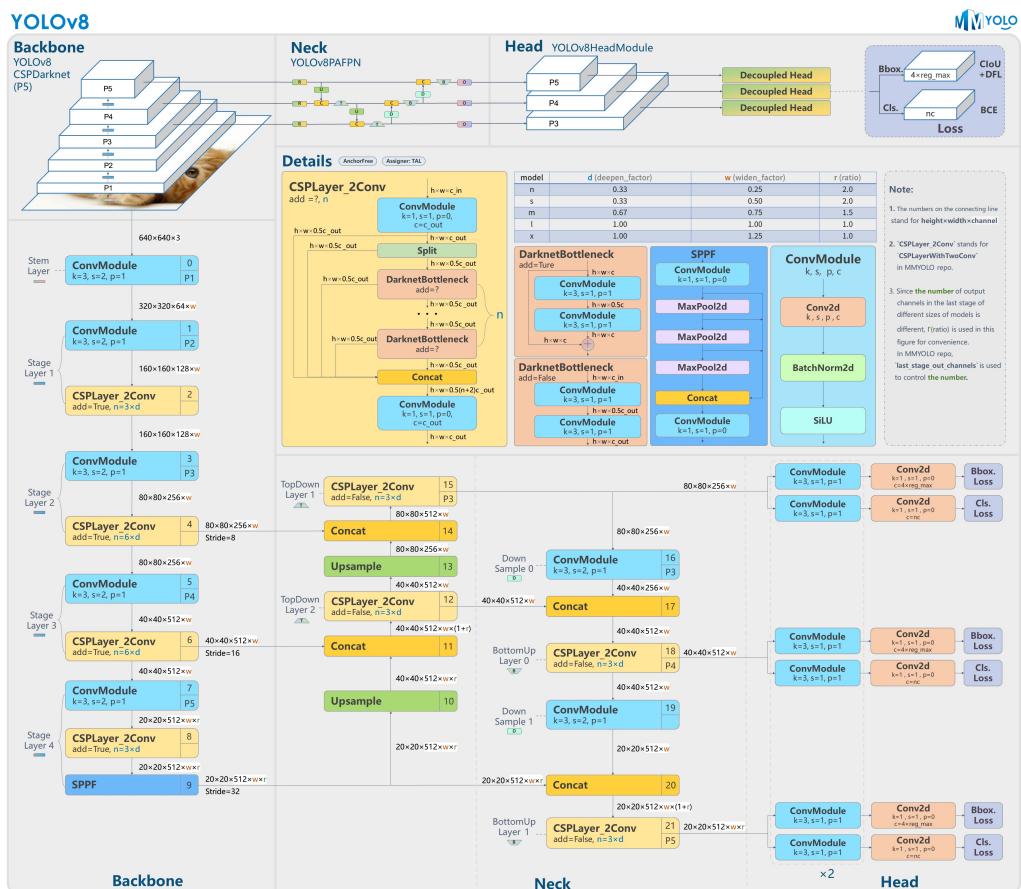


Abbildung 4.1: Übersicht über die Architektur von YOLOv8. Quelle: [1]

4.2 Verlustfunktion

4.3 Modellauswertung

Kapitel 5

Zusammenfassung und Ausblick

Literaturverzeichnis

- [1] CONTRIBUTORS, MMYOLO: *mmyolo*. <https://github.com/open-mmlab/mmyolo>, Abruf: 10. Juli 2023
- [2] GE, Zheng ; LIU, Songtao ; WANG, Feng ; LI, Zeming ; SUN, Jian: YOLOX: Exceeding YOLO Series in 2021. In: *arXiv preprint arXiv:2107.08430* (2021)
- [3] 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
- [4] 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
- [5] MEGVII-BASEDETECTION: YOLOX. <https://github.com/Megvii-BaseDetection/YOLOX>. Version: 2021
- [6] MONGARAS, Gabriel: *YOLOX Explanation — How Does YOLOX Work?* <https://medium.com/mlearning-ai/yolox-explanation-how-does-yolox-work-3e5c89f2bf78>, Abruf: 12. Juli 2023
- [7] 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
- [8] 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

- [9] SHAH, Deval: *YOLOv4 — Version 3: Proposed Workflow.* <https://medium.com/visionwizard/yolov4-version-3-proposed-workflow-e4fa175b902>, Abruf: 13. Juli 2023
- [10] 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
- [11] 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	5
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
4.1	Übersicht über die Architektur von YOLOv8	14

Kapitel 6

Anhang

6.1 Ordnerstruktur

6.2 Dateien: Datensatz

6.3 Dateien: YOLOX

6.4 Dateien: YOLOv8