MASTER-THESIS [OBJECT DETECTION YOLO/DETR]
SVEN GOLDINGER

MASTER THESIS

MASTER OF ADVANCED STUDIES DATA SCIENCE
AN DER ZÜRCHER HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN (ZHAW)

2023

OBJECT DETECTION

REAL-TIME OBJEKT ERKENNUNG IM STRASSENVERKEHR UND DER LUFTFAHRT

EINGEREICHT VON:

Name: Sven Goldinger
E-Mail: goldinger@gmail.com

ERSTGUTACHTER / ZWEITGUTACHTER

Prof. Dr. Frank-Peter Schilling
scik@zhaw.ch

Dr. Ahmed Abdulkadir
ahmed.abdulkadir@zhaw.ch

DATEN

Projektstart: 06. März 2023
Eingereicht am: 29. Juni 2023

Management Summary

Ausgangslage

Eine Object Detection in Echtzeit durchführen zu können, ist in verschiedenen Bereichen wünschenswert oder gar zwingend erforderlich. So beispielsweise bei der Erkennung von Verkehrsschildern in einem Fahrassistenzsystem, um dem Fahrer auf einem Display z.B. die aktuell gültige Höchstgeschwindigkeit anzuzeigen oder vor Gefahrenstellen (Baustellen, scharfe Kurven, etc.), die mit Verkehrsschildern signalisiert sind, zu warnen. Ein weiterer Bereich ist die Luftfahrt oder im Speziellen die General Aviation als zivile Luftfahrt, wo gewisse Flüge nach Sichtflugregeln (VFR) durchgeführt werden. Bei solchen VFR-Flügen gilt das «See and Avoid»-Prinzip, wonach der Pilot die Lage des Luftfahrzeuges im Raum und insbesondere die Lage relativ zu anderen Luftfahrzeugen oder Hindernissen visuell kontrollieren und diesen gegebenenfalls selbständig ausweichen muss. Die Zahl der gemeldeten Zwischenfälle in Bezug auf sicherheitsgefährdende Annäherungen zwischen Luftfahrzeugen ist verhältnismässig gross, obschon es bereits technische Geräte gibt, die vor einer drohenden Kollision warnen können. Diese FLARM-Geräte aber haben den gewichtigen Nachteil, dass sie nur zuverlässig funktionieren und warnen, wenn beide Luftfahrzeug über ein solches Gerät verfügen und es aktiviert haben.

Bei beiden Anwendungsgebieten besteht zusätzlich die Anforderung, dass die Object Detection nicht nur in Echtzeit erfolgen, sondern auch bei schlechten Licht- und ganz unterschiedlichen Wetterverhältnissen funktionieren muss. Zudem ist der verfügbare Platz für spezielle Kamera- oder Hardware-Ausrüstung für die Object Detection-Systeme in Fahr- bzw. Luftfahrzeugen beschränkt. Wobei die Erkennung von Verkehrsschildern in Fahrzeugen noch auf dem Mobiltelefon mit entsprechender Kamera erfolgen könnte, müssen entsprechende Detektions-Systeme für Luftfahrzeuge hingegen in deren bestehenden Board-Systemen eingebettet und integriert werden können.

Fragestellung und Ziele

Aufgrund der Ausgangslage und den entsprechenden Problemstellungen ergeben sich für diese Master-Arbeit die folgenden Zielsetzungen:

Das Hauptziel liegt in der real-time Objekt-Erkennung von Verkehrsschildern und Luftfahrzeugen (Helikopter). Diese Objekte sollen basierend auf YOLOv5 und DETR-Algorithmen in Echtzeit detektiert, mit Bounding-Boxen markiert und korrekt klassifiziert werden. Die beiden Modelle bzw. Algorithmen unterscheiden sich grundlegend, weshalb beide Architekturen einander gegenübergestellt werden sollen. Dabei sollen die beiden Ansätze unter anderem in Bezug auf Architektur, Trainingsaufwand und Leistungsfähigkeit in der anwendungsspezifischen Objekterkennung verglichen und bewertet werden. Zusätzlich soll im Rahmen eines Prototyps, eine mobile Applikation implementiert werden, welche auf Basis der trainierten Modelle die Detektion und Klassifizierung auf dem Live-Bild der Mobiltelefon-Kamera vornimmt. Dieser Prototyp soll zeigen, ob und inwiefern solche Computer Vision Modelle auf mobilen Endgeräten und damit später allenfalls auch auf Embedded Systemen, wie sie auch in Luftfahrzeugen existieren, genutzt werden können.

Methodisches Vorgehen

In einem ersten Schritt erfolgte die Literaturrecherche und das Studium der Architekturen, Funktionsweisen und Implementationen von YOLO, DETR und ihren Komponenten. Nach der Prüfung der Datensätze wurden die Bereinigung und die Integration der Trainings- und Test-Bilder vorgenommen. Anschliessend wurden anhand systematischer Experimente verschiedene Modelle für die Detektion von Verkehrsschildern und Luftfahrzeugen trainiert. Die einzelnen Trainings-Experimente wurden für folgende Analysen, allfällige Optimierungen und den späte-

ren Vergleich laufend dokumentiert. Parallel dazu erfolgten Recherchen bezüglich Integration von Computer Vision Modellen auf mobilen Endgeräten und der Entwicklung entsprechender mobilen Applikationen. In einem nächsten Schritt folgten dann die Implementation des Prototyps und die Tests zur Inferenz auf Mobiltelefonen. Abschliessend wurden die Ergebnisse interpretiert und die beiden Architekturen miteinander verglichen.

Resultate

Für den Anwendungsbereich im Strassenverkehr wurde ein YOLOv5-Modell trainiert, das mit einer Mean Average Precision(IoU 0.5) von rund 94% und einer mAP(0.5:0.95) von 51% eine relativ hohe Genauigkeit erreicht. Auf Standbildern zeigt das Modell eine gute Leistung in der Erkennung von über 40 verschiedenen Strassenschild-Klassen. Trotz des Einsatzes von über 10% Background-Images, wie von den YOLO-Autoren empfohlen, ist die Inferenz auf Videos aber nicht gleichermassen zufriedenstellend.

Nach einer Überarbeitung des Datensatzes für den Bereich der Luftfahrt wurde auch für die Detektion von Helikoptern ein YOLOv5-Modell aufgebaut. Dieses erzielt mit einer mAP(0.5) von rund 94% bzw. einer mAP(0.5:0.95) von rund 78% eine hohe Genauigkeit – sowohl die Inferenz auf Bildern, wie auch auf Videos ist mit durchschnittlich nur rund 20 Millisekunden schnell und akkurat. Damit eignet sich dieses Modell insbesondere für den Einsatz unter real-time Bedingungen.

Zusätzlich wurde ein DETR-Modell entwickelt und für den Anwendungsfall in der Luftfahrt trainiert. Auch dieses Modell erreicht eine hohe Genauigkeit. Die mAP(0.5) liegt bei rund 92% und die mAP(0.5:0.95) bei über 75%. Die Inferenz ist mit über 200ms zwar deutlich langsamer als mit dem YOLOv5-Modell, erreicht auf Standbildern aber eine vergleichbare Leistung in der Detektion von Helikoptern.

Mit der Entwicklung einer iOS-App wird zudem aufgezeigt, wie sich solche Computer Vision Modelle (YOLOv5) in Anwendungen auf mobilen Endgeräten integrieren lassen. Die App kann auf dem Live-Bild der Mobiltelefon-Kamera sowohl Helikopter, als auch Verkehrsschilder detektieren, mit Bounding-Boxen markieren und dem Benutzer unmittelbar signalisieren. Das DETR-Modell wurde nicht für den Einsatz auf dem Mobiltelefon adaptiert.

Abschliessend wurden die zwei Modell-Architekturen und deren Leistung in der problemspezifischen Object Detection miteinander verglichen. Die Ergebnisse zeigen, dass die unterschiedlichen Architekturen sowohl für die Detektion von Verkehrsschildern, wie auch von Luftfahrzeugen im 3-dimensionalen Raum eingesetzt werden können. Das YOLOv5-Modell ist mit rund 7.1 Mio. Parametern und nur 20MB benötigtem Speicher aber wesentlich leichtgewichtiger. Bei DETR sind es rund 28.5 Mio. Parameter und ein Speicherbedarf von ca. 115 MB.

Zusammen mit den deutlichen Unterschieden in den Inferenz-Zeiten zeigt sich, dass mit den YOLOv5-Modellen durchaus real-time Object Detection auch auf handelsüblichen Mobiltelefonen durchgeführt werden kann, die Tauglichkeit von DETR diesbezüglich aber eingeschränkt ist.

Inhaltsverzeichnis

Management Summary ..... **II**

Inhaltsverzeichnis ..... **IV**

Weitere Verzeichnisse ..... **VI**

Abbildungsverzeichnis VI
Tabellenverzeichnis VII
Abkürzungsverzeichnis VII

Hinweise zum Dokument ..... **VIII**

1. Einleitung **1**

1.1 Ausgangslage 1
1.2 Abgrenzung 1
1.3 Ziele und Fragestellungen 2
1.3.1 Objekt-Erkennung basierend auf YOLO/DETR 2
1.3.2 Prototyp Mobile Applikation 2
1.4 Aufbau der Arbeit 2

2. Computer Vision / Object Detection **3**

2.1 Computer Vision 3
2.2 Convolutional Neural Network (CNN) 3
2.2.1 Übersicht 3
2.2.2 CNN-Layers 4
2.3 Object Detection 4
2.3.1 Data Augmentation 5
2.4 Mean Average Precision Metrik 5
2.4.1 Confusion Matrix 5
2.4.2 Intersection over Union 6
2.4.3 Precision/Recall 6
2.4.4 Precision-Recall Curve 6

3. Datensätze **8**

3.1 German Traffic Sign Recognition Benchmark 8
3.2 See And Avoid 8

4. Umgebungen/Hardware **8**

4.1 Programm-Bibliotheken 9

5. YOLO You Only Look Once **9**

5.1 Einführung 9
5.2 YOLOv5 9
5.3 Architektur 10
5.3.1 High-Level Object Detection Architektur 10
5.3.2 Basis-Architektur YOLO 11
5.3.3 YOLOv5 Architektur 13
5.3.4 PyTorch Training Procedures 14

5.4 Wie funktioniert Object Detection mit YOLO? 16
5.4.1 Residual Blocks 16
5.4.2 Bounding Box Regression 16
5.4.3 Intersection over Union 17
5.4.4 Non-Maximum Suppression 17
5.5 Modell Training YOLOv5 18
5.5.1 Datensatz GTSRB 18
5.5.2 Datensatz See And Avoid 22
5.5.3 Datensatz See And Avoid Small 24
6. DETR Detection Transformer 25
6.1 Einführung 25
6.2 Architektur 25
6.2.1 Backbone 25
6.2.2 Transformer-Encoder 26
6.2.3 Transformer-Decoder 26
6.2.4 Final Prediction Feed-Forward Network 26
6.3 Modell-Training DETR 27
6.3.1 Datensatz See And Avoid Small 27
7. Modell-Vergleich 28
7.1 Architektur 28
7.2 Training und Inferenz 29
7.3 Modell-Grösse 31
8. Mobile Object Detection Applikation 32
8.1 Modell-Konversion 32
8.1.1 Model-Optimierung und Post-Processing 32
8.2 Applikations-Struktur 33
8.2.1 App Views und Layers 34
8.2.2 Kamerabild-Caption: ViewController 35
8.2.3 Real-time Inferenz: VisionObjectRecognitionViewController 35
9. Fazit und Ausblick 37
9.1 Fazit/Ergebnisse 37
9.2 Kritische Diskussion 38
9.3 Ausblick 39
10. Literaturverzeichnis 40
11. Selbständigkeitserklärung 43
12. Anhang 44

Weitere Verzeichnisse

Abbildungsverzeichnis

Abbildung 1: Aufbau eines Convolutional Neural Networks, Quelle: (dshahid380 2019) 3

Abbildung 2: Image Classification vs. Image Localization vs. Object Detection 4

Abbildung 3: Definition Confusion-Matrix 6

Abbildung 4: Precision-Recall Curve Metriken, Quelle: (Solawetz 2020a) 7

Abbildung 5: Precision-Recall Curves mit unterschiedlichem IoU-Threshold, Quelle: (Solawetz 2020a) 7

Abbildung 6: Modell-Vergleich Average Precision ./. Inference Speed auf dem COCO-Eval Dataset, Quelle: (ultralytics/yolov5 2023) 10

Abbildung 7: Object Detection Process, Quelle: (Bochkovskiy et al. 2020) 10

Abbildung 8: YOLO Architektur, Quelle: (Redmon et al. 2015) 11

Abbildung 9: Loss-Funktion YOLO, Quelle: (Redmon et al. 2015) 12

Abbildung 10: Cross Stage Partial DenseNet, Quelle: (Wang et al. 2019) 13

Abbildung 11: Feature Network Design, Quelle: (Wang et al. 2019) 14

Abbildung 12: Mosaik-Augmentation Trainings-Batch 15

Abbildung 13: Bounding-Box Vorhersage basierend auf einer Anker-Box, Quelle: (Redmon und Farhadi 2018) 17

Abbildung 14: BBox Annotation GTSRB 18

Abbildung 15: Generierung Background-Images 19

Abbildung 16: Metriken Experiment "Run35" 21

Abbildung 17: Metriken Experiment "Exp7" 21

Abbildung 18: False Positive (Inferenz Mobiltelefon) 22

Abbildung 19: Confusion-Matrix Exp21 YOLO/See And Avoid 22

Abbildung 20: Unpassende/fehlende Bounding-Box Labels Open Image v7 23

Abbildung 21: Übersicht DETR mit Bipartite Matching zwischen Vorhersagen und Ground Truth, Quelle: (Carion et al. 2020) 25

Abbildung 22: DETR-Architektur, Quelle: (Carion et al. 2020) 26

Abbildung 23: Validation-Loss Run5-11 TensorBoard 28

Abbildung 24: Modell-Konversion iOS und Android 32

Abbildung 25: Struktur iOS Mobile Application 34

Abbildung 26: Klassen- und Modell-Struktur iOS Applikation 34

Abbildung 27: Views und Layers iOS Mobile Application 35

Abbildung 28: AVCaptureSession mit Input/Output 35

Abbildung 29: Real-time Inferenz iPhone8 und iPhone13 mini 37

Tabellenverzeichnis

Tabelle 1: Umgebungen/Hardware für die Experimente....9
Tabelle 2: Inferenz YOLOv5 mit GTSRB Datensatz (links: Input-Bild, rechts: nach der Inferenz) 16
Tabelle 3: Experimente YOLO/GTSRB (Best Epoch)....20
Tabelle 4: Experimente YOLO/See And Avoid Small (Best Epoch)....24
Tabelle 5: Experimente DETR/See And Avoid Small....27
Tabelle 6: Vergleich Inferenz (Testbilder) YOLO/DETR....31

Abkürzungsverzeichnis

ADAS....*Advanced driver-assistance systems*
AUC....*Area Under the Curve*
CNN....*Convolutional Neural Network*
CPU....*Central Processing Unit*
DETR....Computer Vision Framework: *Detection Transformer*
FFN....*Feed-Forward Network*
FPN....*Feature Pyramid Network*
FPS....*Frames per Second*
GIoU....*Generalized Intersection over Union*
GPU....*Graphics Processing Unit*
mAP....*mean Average Precision*
NMS....*Non-maximum suppression*
NNAPI....*Neural Networks API*
RT-DETR....*Real Time Detection Transformer*
SPP....*Spatial Pyramid Pooling aka Spatial Pyramid Matching*
TSR....*Traffic Sign Recognition*
VFR....*Visual Flight Rules*
YOLOR....*You Only Learn One Representation*
YOLOv8....Computer Vision Framework: *You Only Look Once*
YOLOX....*Exceeding YOLO Series in 2021*

Hinweise zum Dokument

Um wichtige Definitionen oder Fakten zu erklären, wird das folgende grafische Objekt verwendet:

Dies ist eine Notiz und erklärt wichtige Fakten zu den letzten Paragraphen.

Für den Beschrieb getroffener Annahmen oder bestimmten Voraussetzungen/Hinweisen, welche in Bezug auf entsprechende Paragraphen definiert werden, wird das folgende, grafische Objekt verwendet:

Dies ist eine beispielhafte Beschreibung für eine Annahme oder Voraussetzung.

In gewissen Abschnitten werden Features, Funktionen oder Erfüllungsgrade beschrieben. Die folgende Legende beschreibt die Bedeutung der verwendeten Symbole:

✓ Vollständig zutreffend, unterstützt oder abgedeckt
~ Teilweise zutreffend, unterstützt oder abgedeckt
× Gar nicht zutreffend, unterstützt oder abgedeckt

Hinweis:
Aus Gründen der Lesefreundlichkeit wird auf eine geschlechtsspezifische Differenzierung mehrheitlich verzichtet. Dieselben Aussagen gelten uneingeschränkt auch in der weiblichen bzw. männlichen Form.

Zudem werden an verschiedenen Stellen Anglizismen oder bewusst englische Wörter genutzt, weil diese, als im Fachgebiet verbreitete Fachausdrücke betrachtet und so stehen gelassen werden.

1. Einleitung

1.1 Ausgangslage

Traffic-Sign Recognition (TSR) ist eine Technologie zur Erkennung von Strassen- bzw. Verkehrsschilder wie z.B. «zulässige Höchstgeschwindigkeit» mithilfe einer vorwärts gerichteten Kamera. TSR ist ein Teilbereich von den sogenannten erweiterten Fahrassistenzsystemen (ADAS). Die Technologie verwendet Digital Image Processing und Computer Vision, um die Schilder zu erkennen. Ein grundlegender Anwendungsfall ist hierbei, die Informationen möglichst schnell und früh aus dem Kamerabild zu extrahieren und beispielsweise auf dem Armaturenbrett anzuzeigen oder den Fahrer zu warnen. TSR ist ein multi-class Klassifizierungs-Problem mit unausgeglichenen Klassen-Häufigkeiten. Zudem muss der Klassierer aufgrund verschiedener Wetterbedingungen oder Lichtverhältnissen mit einer hohen Vielfalt visueller Erscheinungsbilder umgehen können.

Dieser Ansatz der Objekt-Erkennung kann auch in anderen Bereichen zur Behandlung von multiclass Klassifizierungs-Problemen angewendet werden, die ebenfalls dem Einfluss wechselnder Licht- und Wetterverhältnissen ausgesetzt sind und quasi in Echtzeit funktionieren müssen: in der General Aviation[1] bezeichnet man Flüge, welche nach den hierfür gültigen Sichtflugregeln oder Visual Flight Rules (VFR) durchgeführt werden, als Sichtflüge. Typischerweise findet ein Flug nach Sichtflugregeln statt, ohne dass ein Fluglotse eine Staffelung vom anderem Flugverkehrt vornimmt. Dabei gilt das Prinzip «See And Avoid» - dies bedeutet, dass der Pilot die Lage seines Luftfahrzeugs im Raum und insbesondere die Lage relativ zu anderen Luftfahrzeugen oder Hindernissen visuell kontrolliert und diesen gegebenenfalls selbständig ausweicht (dr.jur. Müller 2014). Die Zahl der, beim Bundesamt für Zivilluftfahrt gemeldeten Zwischenfälle, schweren Störungen oder Unfälle in Bezug auf sicherheitsgefährdende Annäherungen zwischen Luftfahrzeugen (Airprox) zeigt, dass dieses Prinzip «See And Avoid» und das frühzeitige Erkennen von anderen Luftfahrzeugen (wie z.B. Flugzeuge, Helikopter, Ballone oder Gleitschirme) von grösster Wichtigkeit ist (BAZL 2023).

Für die beiden beschriebenen Zwecke wird im Rahmen dieser Master-Arbeit ein grundlegendes Computer Vision Modell trainiert, welches zeigen soll, wie Verkehrsschilder bzw. Luftfahrzeuge auf Bildern und insbesondere auf bewegten Bildern (Videostreams) schnellstmöglich detektiert und erkannt werden können. Zusätzlich wird im Rahmen eines Prototyps, eine mobile Applikation (App) implementiert, welche auf Basis der trainierten Modelle die Klassifizierung auf dem Live-Bild der Mobiltelefon-Kamera vornimmt und diese Information entsprechend ausgibt.

Dieser Prototyp soll zeigen, ob bzw. wie solche Computer Vision Modelle auf mobilen Endgeräten und damit später allenfalls auch auf Embedded Systems[2], wie sie beispielsweise in Verkehrsmitteln eingesetzt werden, genutzt werden können. Dies ist insbesondere für den Anwendungsfall «See And Avoid» wichtig, weil die Modelle damit in die Bordsysteme der Luftfahrzeuge integriert und für die Detektion verwendet werden könnten.

1.2 Abgrenzung

Das Model zur real-time Objekterkennung wird basierend auf zwei bestehenden Algorithmen bzw. Architekturen entwickelt. Dabei dienen die Algorithmen von YOLOv5 und DETR als Basis und werden mit problemspezifischen Datensätzen weiterentwickelt oder optimiert. Die Ent-

[1] Allgemeine Luftfahrt ist die Bezeichnung für die zivile Luftfahrt (ausgenommen Linienverkehr)

[2] Eine Kombination aus Hardware und Software. Die Systeme können programmierbar oder mit einer vorab festgelegten Funktion ausgestattet sein.

wicklung vollständig neuer Modelle oder gänzlich neuer Algorithmen und Architekturen zur Objekterkennung sind nicht Bestandteil dieser Arbeit. Die mobile Applikation, welche zusätzlich entwickelt wird, soll ausschliesslich als Prototyp fungieren und nur grundsätzlich zeigen, wie trainierte Computer Vision Modelle in Apps integriert und unter Nutzung der Mobiltelefon-Kamera angewendet werden können. Dabei wird geprüft, welche Möglichkeiten für die verschiedenen Plattformen wie iOS oder Android bestehen oder ob auch hybride Lösungen für die Integration von Computer Vision Modellen in mobile Apps denkbar sind.

1.3 Ziele und Fragestellungen

1.3.1 Objekt-Erkennung basierend auf YOLO/DETR

Das Hauptziel dieser Master-Arbeit liegt in der real-time Objekt-Erkennung von Verkehrsschildern und Luftfahrzeugen (Helicopter, Airplane, Balloon). Diese Objekte sollen basierend auf YOLOv5 und DETR-Algorithmen mit Bounding-Boxen markiert und korrekt klassifiziert werden. YOLO ist ein Modell in der «You Only Look Once» Familie von Computer Vision Modellen und dient üblicherweise der real-time Objekt-Erkennung. Dieses Modell ist auf dem COCO-Datensatz vortrainiert. Die Algorithmen von DETR unterscheiden sich grundlegend in der Architektur. DETR ist ein set-basiertes Vorgehen und verwendet einen, dem Convolutional Backbone nachgelagerten Transformer. Damit dabei mit beiden Frameworks ein möglichst hoher Genauigkeitsgrad bei der Erkennung von Verkehrsschildern und Luftfahrzeugen erreicht wird, sollen die beiden Modelle mit problemspezifischen Datensätzen trainiert und optimiert werden. Abschliessend sollen die beiden verschiedenen Ansätze (YOLO und DETR) einander gegenübergestellt und unter anderem in Bezug auf Architektur, Trainingsaufwand und Leistungsfähigkeit in der anwendungsspezifischen Objekterkennung verglichen und bewertet werden.

1.3.2 Prototyp Mobile Applikation

Ein weiteres Ziel dieser Arbeit ist die Implementierung eines Prototyps einer mobilen Applikation (App). Diese App soll zeigen, wie ein trainiertes Computer Vision Modell in eine solche Anwendung integriert und auf dem Output der Mobiltelefon-Kamera zur real-time Objekterkennung angewendet werden kann.

1.4 Aufbau der Arbeit

In den ersten Kapiteln erfolgt die Einführung in das Themengebiet der Computer Vision und Object Detection. Anschliessend folgt die Dokumentation der Datensätze und die Beschreibung beider Architekturen YOLO bzw. DETR. Darauffolgend werden die unterschiedlichen Implementationen und die dazugehörigen Trainings-Experimente beschrieben. Zudem werden die beiden Modelle in Bezug auf ihre Architektur und ihre Leistungsfähigkeit in der problemspezifischen Objekt Detection miteinander verglichen. Im Kapitel 8 wird schliesslich beschrieben, wie ein solches Computer Vision Modell in eine mobile Applikation integriert und zur real-time Object Detection genutzt werden kann. Abschliessend folgt das Fazit und der Ausblick für weitere Forschungsschritte in Kapitel 9.

2. Computer Vision / Object Detection

2.1 Computer Vision

Computer Vision bedeutet im Grunde genommen «Computer-Sehen» und beschreibt ein Teilgebiet der künstlichen Intelligenz, welches dem Computer ermöglichen soll, die visuelle Welt wie z.B. Bilder oder Videos aufnehmen und interpretieren zu können. Der Computer soll also darin trainiert werden, Informationen aus visuellen Daten zu erfassen und zu interpretieren und damit Bilder auf die gleiche Weise wie ein Mensch verstehen zu können (Klette 2014). Dafür wird im Bereich der Computer Vision mit bestimmten Algorithmen versucht, die Arbeitsweise des menschlichen Gehirns mit den Neuronen (Nervenzellen) nachzubilden – den sogenannten künstlichen neuronalen Netzwerken. Diese neuronalen Netze extrahieren vereinfacht dargestellt, Muster aus den bereitgestellten Trainings-Bildern (vgl. Kapitel 2.4). Ähnlich wie beim biologischen Vorbild, werden dazu künstliche Neuronen mit zufälligen Anfangsgewichten bzw. Werten (einfaches Perzeptron) auf verschiedenen Schichten angeordnet und miteinander zu einem Netzwerk verbunden. Die Eingabewerte (hier Bildinformationen) werden dann durch dieses Netzwerk geleitet und von jedem Neuron mit den individuellen Werten gewichtet. Diese gewichteten Daten werden folglich als Reiz (Aktivierungsfunktion) an alle verknüpften Neuronen über die einzelnen Schichten hinunter bis zur Ausgabeschicht weitergeleitet. In der letzten Schicht werden die resultierenden Entscheidungen, beispielsweise Wahrscheinlichkeiten, dass es sich auf dem Bild um ein bestimmtes Objekt handelt, ausgegeben. In einem solchen System werden, gemessen an einer definierten Loss-Funktion, die einzelnen Gewichte der Neuronen schichtweise laufend aktualisiert (Backpropagation). Durch diesen Aufbau werden schlechte Vorhersagen bestraft und richtige Klassifizierungen belohnt. So «lernt» das Neuronale Netzwerk bzw. das Modell, immer genauere, der Wahrheit entsprechende Vorhersagen zu machen.

2.2 Convolutional Neural Network (CNN)

2.2.1 Übersicht

Das Convolutional Neural Network (CNN), wie in Abbildung 1 schematisch dargestellt, ist die Unterform eines künstlichen neuronalen Netzwerks und ist insbesondere für die Anwendung im Bereich der Bilderkennung bzw. Verarbeitung ausgelegt. Dies auch deshalb, weil der Aufbau dieser neuronalen Netzwerke dem biologischen Aufbau des menschlichen visuellen Kortexes nachempfunden ist.

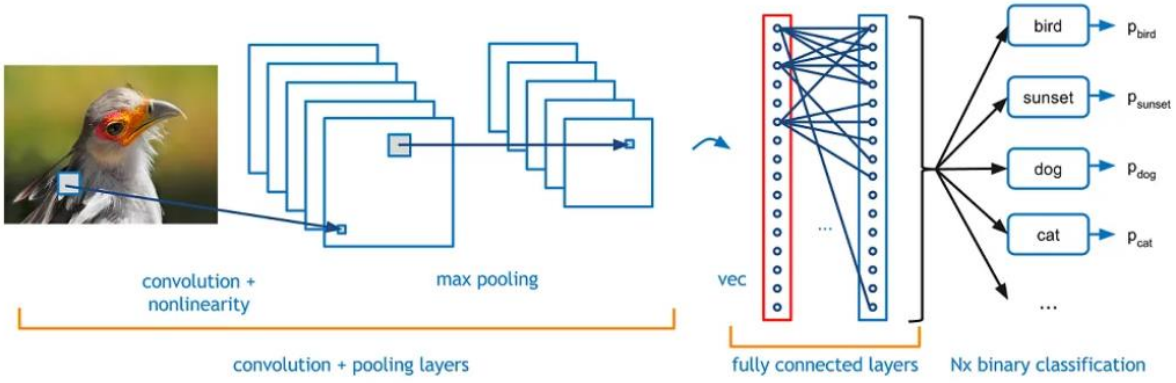


Abbildung 1: Aufbau eines Convolutional Neural Networks, Quelle: (dshahid380 2019)

Ein CNN besteht grundsätzlich aus den folgenden 6 Schichten:

1. Input Layer
2. Convolutional Layer (+ ReLU)
3. Pooling Layer
4. Fully connected Layer (FC)
5. Softmax/logistic Layer
6. Output Layer

2.2.2 CNN-Layers

Entsprechend des Inputs mit den Bildern als zwei- oder dreidimensionale Matrix (Höhe, Breite, Farbkanäle), haben die Neuronen im Convolutional Layer eine 2D oder 3D Anordnung. Über die sogenannte Faltungsmatrix (Kernel) wird die Aktivität der Neuronen in diesem Layer berechnet. Dazu wird der Filter schrittweise über das Bild bewegt und das Skalarprodukt der Faltungsmatrix und dem aktuell darunter liegenden Bildausschnitt berechnet (Dumoulin und Visin 2016). Durch diesen Vorgang im Convolutional Layer werden die verschiedenen Features der Bilder extrahiert (z.B. Linien, Kanten). Im Netzwerk folgende, tiefer liegende Convolutional Layers erkennen komplexere Features (wie z.B. Texturen oder Bereiche eines Gesichts). Die Pooling-Layer werden verwendet, um die Dimensionalität der Eingabe zu reduzieren, ohne wichtige Feature-Informationen zu verlieren. Der zuletzt folgende Fully connected Layer mit dem Softmax-Layer wird für die Klassifizierung der Bilder und die Zuordnung zu den verschiedenen Klassen (z.B. Verkehrsschild «Speed Limit (50km/h)» oder «No Passing», etc.) benötigt. Der Output-Layer schliesslich enthält diese Resultate bzw. Klassen-Labels (dshahid380 2019).

Die im Rahmen dieser Masterarbeit genutzten Modelle (YOLO, DETR) basieren in Bestandteilen ebenfalls auf solchen Convolutional Neural Network.

2.3 Object Detection

Die Object Detection ist eine Aufgabe im Bereich der Computer Vision, um auf Bildern Objekte zu detektieren. Zu dieser Aufgabe gehört neben der Identifikation der Objekt-Position und Objekt-Grenzen auch das Klassifizieren der detektierten Objekte. Auf Abbildung 2 werden die Unterschiede zwischen der reinen Image Classification, der Image Localization bzw. dann der Object Detection aufgezeigt. Aktuell gibt es zwei state-of-the-art Methoden in der Object Detection: «one-stage» Methoden (wie z.B. YOLO, SSD) und «two-stage» Methoden (z.B. Faster R-CNN, Mask R-CNN). Ebenfalls zum Einsatz kommen aber auch andere Architekturen, wie z.B. die transformer-basierte DETR.

Abbildung 2: Image Classification vs. Image Localization vs. Object Detection

2.3.1 Data Augmentation

Im Bereich der Computer Vision und konkret auch bei der Object Detection, wird Data Augmentation eingesetzt, um den eigentlichen Trainings-Datensatz zu vergrössern und damit hoffentlich die spätere Modell-Performance bei den Vorhersagen zu verbessern. Dazu werden mit der Bild Augmentation die ursprünglichen Trainingsbilder verändert und so, zusammen mit den Original-Bildern, ein künstlicher, grösserer Datensatz generiert. Bild Augmentation beinhaltet z.B. das Spiegeln, Rotieren oder Abschneiden von Bildern. Es kann bei Bildern auch «Noise» hinzugefügt oder Teile des Bildes abgedeckt, verzerrt werden, etc. (Yang et al. 2022)

Bei den durchgeführten Experimenten und den genutzten Modellen in dieser Arbeit, wird diese Data Augmentation teilweise ebenfalls eingesetzt.

2.4 Mean Average Precision Metrik

Die Mean Average Precision (mAP) ist eine weit verbreitete Metrik, um die Performance von Computer Vision Modellen zu messen und miteinander zu vergleichen. Die mAP entspricht dem Durchschnitt der Average Precision Metrik (AP) über alle Objekt-Klassen eines Modells. Die mAP kann verwendet werden, um verschiedene Modelle (z.B. YOLO vs. DETR), oder aber auch verschiedenen Versionen des gleichen Modells miteinander zu vergleichen (z.B. zwischen einzelnen Experimenten). Die Mean Average Precision wird gemessen mit Werten zwischen 0 und 1.

Die Formel für die mAP baut auf der Basis der folgenden Sub-Metriken auf:

- Confusion Matrix
- Intersection over Union (IoU)
- Recall
- Precision

2.4.1 Confusion Matrix

Um eine Confusion Matrix aufzubauen, benötigt es vier verschiedene Attribute:

- True Positives (TP)
    - Das Modell sagt ein Objekt mit dem richtigen Label voraus und stimmt mit einem Objekt der Ground Truth überein
- True Negatives (TN)
    - Das Modell sagt kein Objekt voraus und das Objekt ist auch nicht in der Ground Truth
- False Positives (FP)
    - Das Modell sagt ein Objekt voraus, wo in der Ground Truth kein Objekt existiert
- False Negatives (FN)
    - Das Modell sagt kein Objekt voraus, wo in der Ground Truth aber ein Objekt existiert

Diese vier Gruppen formen entsprechend die Confusion-Matrix wie in Abbildung 3 gezeigt:

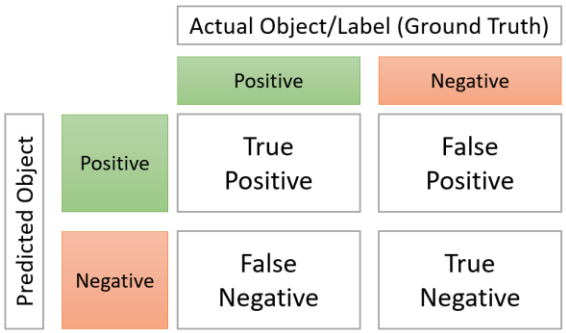


Abbildung 3: Definition Confusion-Matrix

2.4.2 Intersection over Union

Wie in Kapitel 5.4.3 detaillierter erklärt, misst die Intersection over Union (IoU) die Überlappung von zwei Flächen wie zweier Bounding-Boxes (z.B. von vorhergesagter Bounding-Box und Ground-Truth Bounding-Box).

2.4.3 Precision/Recall

Die Precision misst, wie oft das Modell True Positives findet, gemessen an allen Objekt-Detektion des Modells und beantwortet damit die Frage: wie oft sagt das Modell ein richtiges Objekt voraus, wenn das Modell ein Objekt vorhersagt? Die Precision ist definiert als TP/TP+FP. Eine hohe Precision entspricht einer tiefen FP-Rate.

Recall misst, wie viele oder wie gut das Modell alle True Positives findet und ist definiert als TP/TP+FN. Ein hoher Recall bedeutet eine tiefe FN-Rate.

Für Modelle kann entsprechend zwischen Precision und Recall abgewogen und in die gewünschte Richtung optimiert werden. Dies geschieht, indem das Level für die Confidence (Confidence Threshold) verändert wird. Wenn es für das Modell beispielsweise wichtiger ist, anstelle von False Negatives, False Positives zu vermeiden, dann kann der Confidence Threshold erhöht werden.

2.4.4 Precision-Recall Curve

Die Precision-Recall Curve (PR-Curve) als Funktion der Modell-Confidence gezeichnet, gibt für ein Computer Vision Modell an, wie stark sich der Recall bei einer gegebenen Precision ändert (und umgekehrt). Eine grosse Fläche unter der Kurve bedeutet, dass das Modell einen hohen Recall und eine hohe Precision hat. Wie Abbildung 4 zeigt, kann die Precision-Recall Curve in einzelne Metriken gefasst werden. Ein einzelner Punkt auf der Kurve gibt den F1 Score bei gegebener Precision und Recall an. Die integrierte PR-Kurve gibt die Gesamtfläche unter der Kurve an und wird als Area Under the Curve (AUC) bezeichnet.

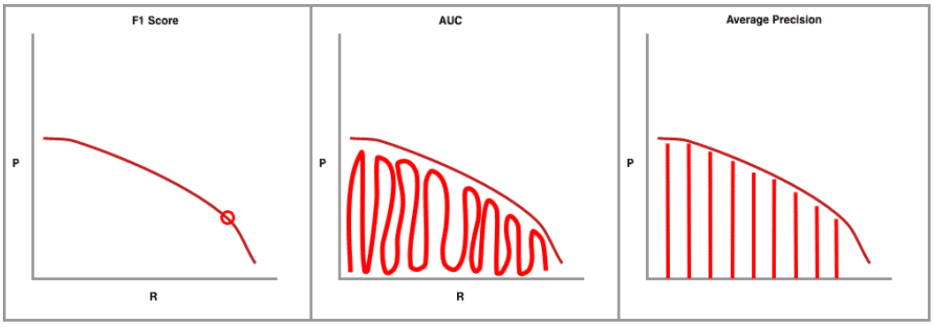


Abbildung 4: Precision-Recall Curve Metriken, Quelle: (Solawetz 2020a)

Die Average Precision (AP) zuletzt, berechnet sich als gewichtetes Mittel der Precision auf unterschiedlichen Threshold-Stufen (Henderson und Ferrari 2016). Um nun die mAP zu berechnen, können verschiedene Precision-Recall Curves mit einem unterschiedlichen IoU-Threshold gezeichnet werden.

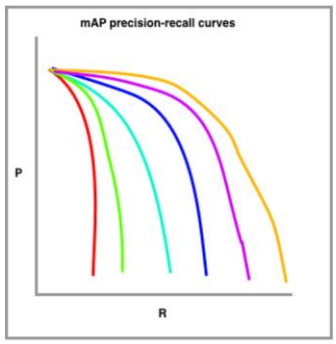


Abbildung 5: Precision-Recall Curves mit unterschiedlichem IoU-Threshold, Quelle: (Solawetz 2020a)

Abbildung 5 zeigt beispielhaft in Rot die Precision-Recall Curve mit der höchsten Anforderung an die IoU (z.B. 90% Überlappung), während bei der orangen Kurve der IoU Threshold beispielsweise auf 10% gesetzt ist.

Für die Berechnung der Mean Average Precision muss also zuerst dieser IoU Threshold festgelegt werden. Es kann einerseits ein einzelner Wert wie z.B. 0.5 (mAP:0.5) oder aber ein Bereich wie z.B. von 0.5 bis 0.95 mit Schrittgrösse 0.05 (mAP0.5:0.95) gewählt werden. Im letzteren Fall wird die mAP für jeden Wert aus dem Bereich einzeln berechnet und anschliessend der Durchschnitt genommen.

In einem zweiten Schritt müssen alle Detektionen, basierend auf der vorhergesagten Objekt-Klasse, in Gruppen aufgeteilt werden. Zunächst wird die Average Precision (AP) für jede Gruppe und für die unterschiedlichen IoU-Thresholds berechnet und dann wiederum der Durchschnitt über alle Gruppen genommen, was in der mAP mit gegebenem IoU-Threshold resultiert (Solawetz 2020a).

Für den Vergleich der beiden Modelle DETR und YOLO bzw. der einzelnen Modell-Versionen, werden im Rahmen dieser Masterarbeit ebenfalls die beiden Metriken mAP(0.5) und mAP(0.5:0.95) eingesetzt.

3. Datensätze

Initial wurden zwei Datensätze für diese Master-Arbeit identifiziert. Anhand derer soll die Leistung in der Object Detection mit YOLO bzw. DETR-Modellen, insbesondere in Bezug auf die real-time Objekt Erkennung auf mobilen Endgeräten, untersucht werden. Die beiden Ausgangs-Datensätze werden in den folgenden Kapiteln beschrieben.

3.1 German Traffic Sign Recognition Benchmark

Der German Traffic-Sign Recognition Benchmark ist ein multi-klassen, single-image Klassifikations-Wettbewerb, welcher an der International Joint Conference on Neural Networks (IJCNN) 2011 ausgeschrieben wurde. Im Rahmen dieses Wettbewerbs sollten europäische Verkehrsschilder möglichst akkurat detektiert werden. Der Wettbewerb wurde so gestaltet, dass kein spezielles Domänen-Wissen vorausgesetzt war. Gleichzeitig haben die Autoren für diesen Wettbewerb auch einen «real-world» Benchmark-Datensatz und dazugehörige Evaluations-Metriken und Baseline-Resultate für Strassenschild-Erkennung vorgegeben. Dieser Datensatz wiederspiegelt eine grosse Vielfalt an unterschiedlichen Strassenschild-Aufnahmen in Bezug auf Distanz, Beleuchtung, Wetterverhältnisse, Bildrotationen und teilweisen Schild-Abdeckungen (German Traffic Sign Benchmarks 2020).

- Eigenschaften
  - Mehr als 40 Verkehrsschild-Klassen (siehe Anhang A.)
  - Gesamthaft über 50'000 Bilder
  - Die Bildgrösse variiert zwischen 15x15 und 250x250 Pixel

3.2 See And Avoid

Der Datensatz für das zweite Anwendungsgebiet ist ein Subset des Open Image Dataset V7. Open Image ist ein Computer Vision Datensatz mit über 9 Millionen mit Bounding-Boxes und Labels annotierten Bildern. Das Subset, welches verwendet wird, umfasst die Klassen «Helicopter», «Airplane» und «Balloon».

- Eigenschaften
  - 3 Klassen
  - Gesamthaft über 12'000 Bilder
  - Die Bildgrösse ist unterschiedlich, in der Grössenordnung 1024x768 Pixel

4. Umgebungen/Hardware

Für die Experimente wurden vier Umgebungen mit unterschiedlicher Hardware verwendet. Dabei hat sich gezeigt, dass sich die kostenfreie oder günstigere Pro-Version von Google Colab nur bedingt für praxisnahe Computer Vision Experimente eignen.

In Tabelle 1 sind die Umgebungen aufgeführt, die eingesetzt wurden.

Umgebung	GPU	RAM (System/GPU)	Laufzeit/Ausführung	Preis
Colab	K80, Tesla T4	16GB/16GB	max. 12h / Ressource nicht garantiert	Gratis
Colab Pro	T4/P100	-/32GB	24h	9 Euro/Monat (Recheneinheiten

				begrenzt)
Colab Pro+	T4/P100/A100	-84GB/-52GB	>24h / Ausführung im Hintergrund	42 Euro/Monat (Recheneinheiten begrenzt)
ZHaW Openstack Ubuntu/nVidia Cuda	T4	16GB/16GB	Unbegrenzt	-

Tabelle 1: Umgebungen/Hardware für die Experimente

4.1 Programm-Bibliotheken

Im Rahmen dieser Arbeit werden für die Implementation der verschiedenen Computer-Vision Modelle grundsätzlich die Programmiersprache Python und die Programm-Bibliothek PyTorch verwendet. PyTorch ist eine auf Machine Learning ausgerichtete Open Source Programmbibliothek basierend auf Python und der Bibliothek Torch. Damit können beispielsweise unter der Nutzung von GPUs und CPUs optimierte Tensor-Berechnungen für Deep Learning durchgeführt werden (PyTorch 2023).

5. YOLO You Only Look Once

5.1 Einführung

You Only Look Once (YOLO) ist ein state-of-the-art Object Detection Algorithmus. Die Architektur ist äusserst schnell - das Basis Modell kann Bilder in Echtzeit mit bis zu 45 frames per second (FPS) verarbeiten. Eine schlankere Version des Netzwerks (Fast YOLO) erreicht auf einer Titan X GPU sogar 155 FPS. You Only Look Once wurde 2015 mit dem berühmten Forschungs-Artikel «You Only Look Once: Unified, Real-Time Object Detection" eingeführt (Redmon et al. 2015). Anders als bei früheren Ansätzen für Object Detection, formulieren die Autoren dieses Artikels das Problem als Regressions-Problem von räumlich getrennten Bounding Boxes mit dazugehörigen Wahrscheinlichkeiten für eine bestimmte Klasse. Dabei werden die Bounding Boxes und Klassen-Wahrscheinlichkeiten von einem einzelnen neuronalen Netzwerk und für das gesamte Bild in einem Durchgang vorhergesagt. Seit der Einführung von YOLO wurden bis Januar 2023 etliche, neue und verschiedene Versionen entwickelt und veröffentlicht. Dazu gehören YOLOv2, 9000 (Redmon und Farhadi 2016), YOLOR, YOLOX, YOLOv4-5, YOLOv7 (Wang et al. 2022) und Version 8.

5.2 YOLOv5

Im Juni 2020 wurde YOLOv5 von Glenn Jocher, Ultralytics herausgegeben. Im Vergleich zu anderen Modellen der YOLO-Familie, wurde die Version 5 nicht im Rahmen eines wissenschaftlichen Artikels eingeführt, basiert jedoch auf den Arbeiten von «You Only Look Once: Unified, Real-Time Object Detection» (Redmon et al. 2015) und einer ersten PyTorch Implementation von YOLOv3 durch Erik Lindernoren (Erik Lindernoren 2023). Damit ist YOLOv5 die erste Version, die mit PyTorch- und nicht mit dem Darknet[3] -Framework geschrieben wurde. Die Architektur von YOLOv5 ist der Version 4 relativ ähnlich und nutzt ebenfalls das CSPDarknet53 Convolutional Neural Network als Backbone (Solawetz 2020b).

[3] Darknet ist ein in C-geschriebenes open source Framework für neuronale Netze

Dieser Release von You Only Look Once stellt mit v5s (small), v5m (medium), v5l (large), und v5x (extra large) vier unterschiedlich grosse Haupt-Modelle zur Verfügung. Jedes dieser Modelle bringt zunehmende Genauigkeit bzw. Leistung in der Object Detection, benötigt jedoch für das Trainieren entsprechend grösseren zeitlichen Aufwand (ultralytics/yolov5 2023). Abbildung 6 zeigt die Inferenz-Geschwindigkeiten und Genauigkeit der unterschiedlichen YOLO-Modell Versionen bei der Validation auf dem COCO Dataset.

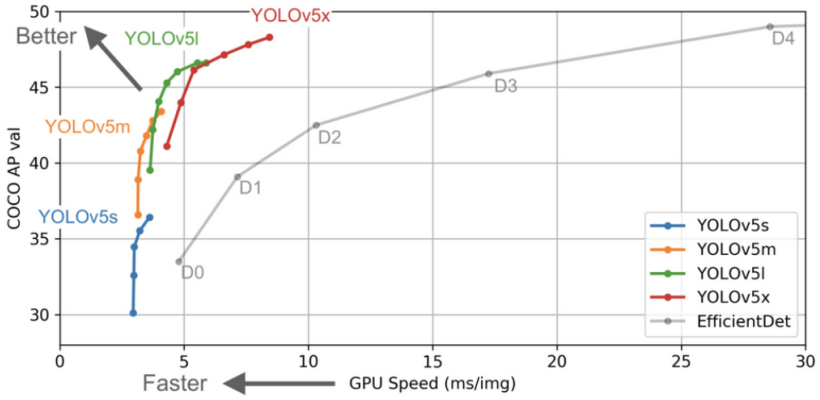


Abbildung 6: Modell-Vergleich Average Precision ./. Inference Speed auf dem COCO-Eval Dataset, Quelle: (ultralytics/yolov5 2023)

Eine der stärksten Verbesserungen in der Architektur von YOLOv5 bringt die Integration des Focus Layers. Dieser ersetzt als einzelner Layer die ersten drei Schichten von YOLOv3 und reduziert damit die Gesamtanzahl an Layer und Modell-Parameter. Zusätzlich erhöht er die Geschwindigkeit beim Forward- und Backward-Pass, ohne dass die erzielte mAP stark beeinflusst wird (ultralytics/yolov5 Focus Layer 2023).

5.3 Architektur

5.3.1 High-Level Object Detection Architektur

Grundsätzlich bestehen CNN-based Object Detectors aus drei Hauptkomponenten, wie in Abbildung 7 gezeigt.

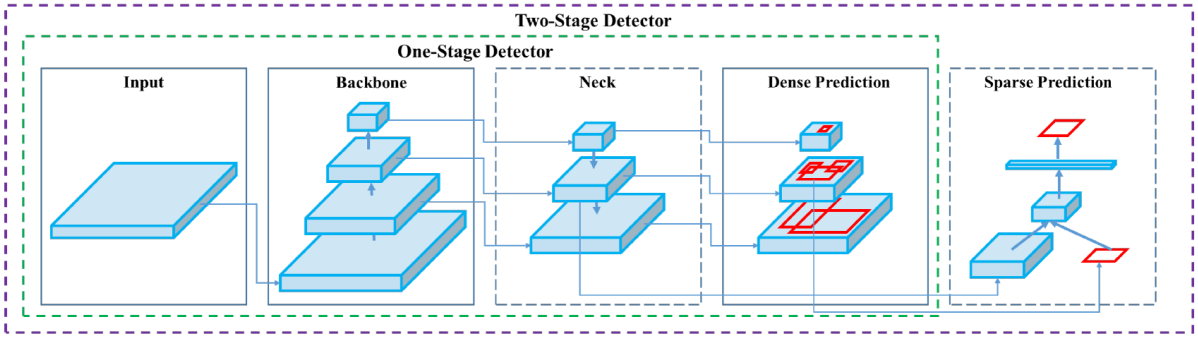


Abbildung 7: Object Detection Process, Quelle: (Bochkovskiy et al. 2020)

1. **Backbone**
   Ein Convolutional Neural Network, welches die Image Features auf unterschiedlichen Granularitätsstufen extrahiert und aggregiert.

2. **Neck**
   Eine Reihe von Convolutional Layers, um Image Features zu kombinieren und an den Head weiterzugeben.

3. **Head**
   Basierend auf den Features, welche vom «Neck» geliefert werden, übernimmt der Kopf die Voraussage der Bounding Boxes und entsprechenden Objekt-Klassen.

5.3.2 Basis-Architektur YOLO

YOLO gehört zu den one-stage Detektoren, wobei nur ein einzelner Durchgang pro Input-Bild benötigt wird, um eine Vorhersage über die Präsenz und Position von Objekten im Bild machen zu können. Grundsätzlich aber kann gesagt werden, dass single-shot Object Detection weniger akkurat ist als andere, two-stage Methoden und auch weniger effektiv bei der Detektion von kleinen Objekten. Im Gegensatz dazu aber sind solche Modelle bezüglich Rechenleistung sehr effizient und können deshalb insbesondere für real-time Object Detection eingesetzt werden (Bochkovskiy et al. 2020).

Das ursprüngliche YOLO-Modell (v1) ist auch als Convolutional Neural Network (CNN) implementiert und besteht aus den Layers, wie auf Abbildung 8 dargestellt.

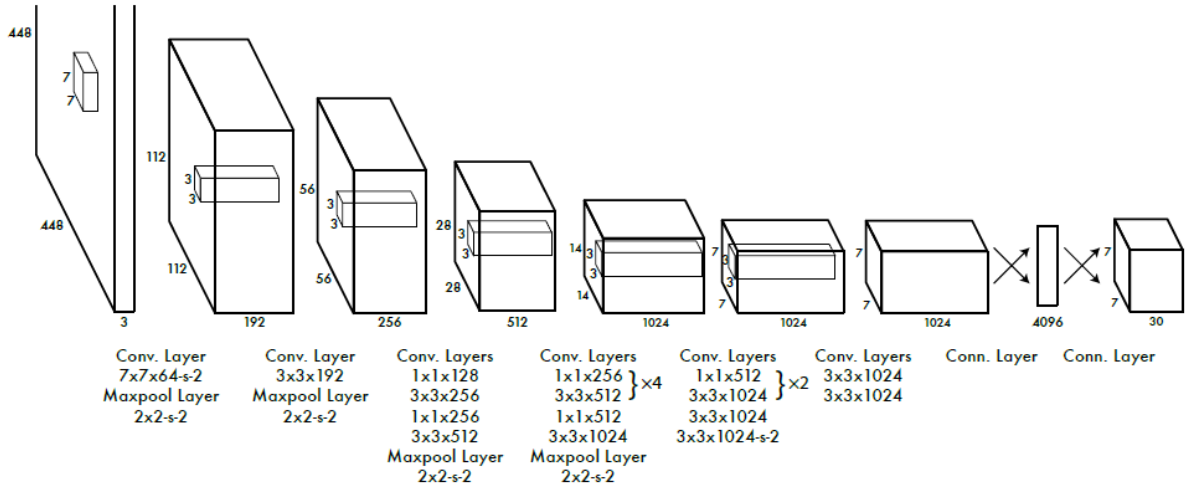
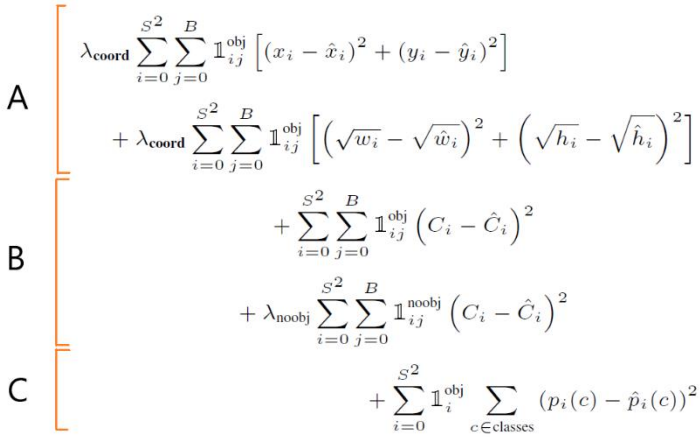


Abbildung 8: YOLO Architektur, Quelle: (Redmon et al. 2015)

Die initialen Convolutional Layers des Netzes extrahieren Features vom Input-Bild, während die Fully Connected Layers die Output-Wahrscheinlichkeiten und die Koordinaten der Bounding-Boxes vorhersagen. YOLO hat 24 Convolutional Layers gefolgt von 2 Fully Connected Layers. Wie in Abbildung 8 ersichtlich, werden jeweils 1x1 Reduction Layers zur Reduktion des Features-Spaces mit darauffolgenden 3x3 Convolutional Layers genutzt. Der finale Output ist ein Tensor der Grösse S x S x (B*5 + C). Dabei entspricht S der Kantenlänge der Residual Blocks. Die Anzahl Klassen im genutzten Datensatz gibt C vor und B entspricht der Anzahl vorhergesagter Bounding-Boxen pro Gitter-Zelle (vgl. dazu Kapitel 5.4). In der Basis-Architektur mit der Evaluation auf PASCAL VOC nutzten die Autoren S = 7, B = 2 und C bei einer Anzahl Klassen im PASCAL VOC Datensatz = 20. Damit also einen Tensor 7x7x30.

5.3.2.1 Loss Function

Während des Modell-Trainings optimiert YOLO die folgende, mehrteilige Loss-Funktion:



$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Abbildung 9: Loss-Funktion YOLO, Quelle: (Redmon et al. 2015)

Wie wir in folgenden Kapiteln sehen werden, sagt YOLO pro Gitter-Zelle mehrere Bounding-Boxes *B* voraus. Um den Loss für die True-Positive Detektionen zu berechnen, soll nur eine solche Zelle für die Vorhersage von einem bestimmten Objekt verantwortlich sein (vgl. Absatz 5.4.1). Zu diesem Zweck wird diejenige Zelle und die Prediction mit der höchsten IoU zur Ground Truth bestimmt.

YOLO nutzt «sum-squared errors» für die Losses, um die Güte der Vorhersagen in Bezug auf die Ground Truth zu bewerten. Die Loss-Funktion besteht aus den folgenden drei Teilen:

- Box Loss (Bounding Box Regression Loss) vgl. Abbildung 9 A
- Confidence Loss vgl. Abbildung 9 B
- Classification Loss (Cross Entropy) vgl. Abbildung 9 C

Der Box- bzw. Localization-Loss misst den Fehler der Grösse und Position der vorhergesagten Bounding-Box. Wobei *x*/*y* die Koordinaten des Zentrums der Box und *w*/*h* die entsprechende Breite und Höhe sind. Auch hier wird der Bounding-Box Fehler nur bestraft, wenn der entsprechende «Predictor» für die Ground-Truth Box bzw. das Objekt auch verantwortlich ist (sprich, die grösste IoU aller Vorhersagen in dieser Gitter-Zelle hat).

Der Confidence Loss beschreibt die Konfidenz über das Vorhandensein eines Objekts.

Der Classification Loss ist der quadrierte Fehler der bedingten Klassen-Wahrscheinlichkeit für jede Klasse. Wobei die Loss Funktion den Klassifikations-Fehler nur bestraft, wenn tatsächlich ein Objekt in der entsprechenden Gitter-Zelle existiert (Redmon et al. 2015).

5.3.3 YOLOv5 Architektur

Im Gegensatz zu YOLO nutzt YOLOv5 grundsätzlich die komplexere EfficientDet Architektur, welche wiederum auf der EfficientNet Architektur basiert. Dadurch erreicht die Version 5 eine höhere Genauigkeit und bessere Generalisierung für eine grössere Anzahl von Objekt-Klassen (Tan et al. 2020).

5.3.3.1 CSPBackbone

Beide Versionen YOLOv4 und v5 implementieren den CSP-Backbone[4] - konkret den CSPDarknet53 Backbone, wobei CSP für «Cross Stage Partial Network» steht. Die Autoren vom CSPNet wollen das Problem der rechenintensiven Inferenz früherer Arbeiten abschwächen. Sie schreiben die Ursache des Problems der duplizierten Gradient Information zu, welche andere grössere ConvNet Backbones aufweisen.

Die CSP-Modelle basieren auf DenseNet. Dieses wurde entwickelt, um die verschiedenen Schichten im Convolutional Neural Network mit jedem anderen Layer (feed-forward) des Netzes zu verbinden (Huang et al. 2016). Dies insbesondere aus den folgenden Gründen:

- Mildern des «Vanishing Gradient Problems»
- Feature Propagation zu stärken
- Das Netzwerk zu ermutigen, Features wieder zu verwenden
- Die Anzahl Netzwerk-Parameter zu reduzieren

In CSPDarknet53 wurde das DenseNet dahingehend angepasst, sodass die Feature Map des Base Layers kopiert wird. Eine Kopie wird anschliessend durch den Dense Block und einen Transition Layer geschickt, der andere Teil hingegen wird direkt zum nächsten Stage geschickt (vgl. Abbildung 10) (Wang et al. 2019).

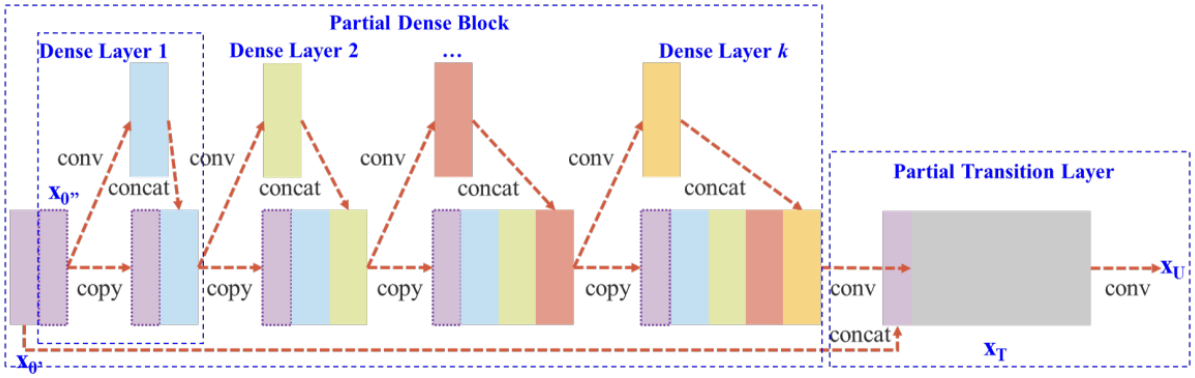


Abbildung 10: Cross Stage Partial DenseNet, Quelle: (Wang et al. 2019)

[4] Cross Stage Partial Networks

5.3.3.2 Path Aggregation Networks (PANet)

Zur Feature Aggregation wird im «Neck» des YOLOv5 Objekt Detektors das PANet implementiert.

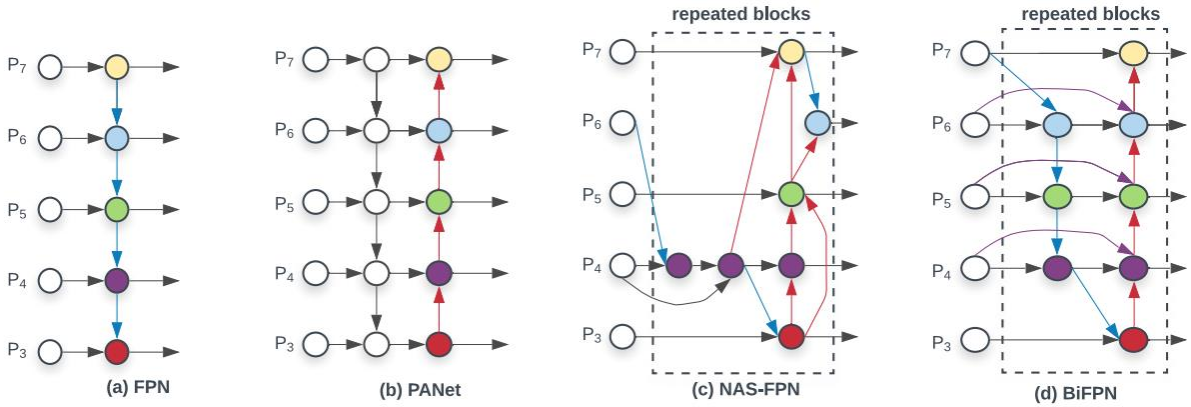


Abbildung 11: Feature Network Design, Quelle: (Wang et al. 2019)

Abbildung 11 stammt aus der Forschung vom Google Research, Brain Team und zeigt unterschiedliche Feature Network Designs. Jedes Level P_i repräsentiert einen Feature Layer des CSP-Backbones. (a) zeigt das konventionelle top-down Feature Pyramid Network (Lin et al. 2016), welches die multi-scale Input-Features von Level 3-7 top-down aggregiert. Die Limitation, dass damit die Informationen nur in eine Richtung fliessen, heben die Autoren von PANet auf, in dem sie ein extra bottom-up Pfad-Netzwerk hinzufügen (b). Vergleiche dazu Anhang E.

5.3.3.3 Spatial Pyramid Pooling (SPP)

YOLOv5 führt auch das Konzept des «Spatial Pyramid Poolings» (SPP) ein. Hierbei handelt es sich um eine Art Pooling Layer, um die räumliche Auflösung der Feature Maps zu reduzieren bzw. das Receptive Field zu vergrössern. SPP wird im YOLOv5-Modell genutzt, um die Leistung der Detektion von kleinen Objekten zu verbessern. Dieses Pooling erlaubt es dem Modell, Objekte in verschiedenen Grössenverhältnissen zu sehen (He et al. 2014).

5.3.4 PyTorch Training Procedures

Wie einführend erwähnt, ist die Architektur von YOLOv5 grundsätzlich ziemlich nahe an der von Version 4. Die Beiträge, welche die beiden Modelle aber leisten, liegen auch in anderen Bereichen der Computer Vision und sie zeigen damit, dass sie als Ganzes die YOLO Object Detection wesentlich verbessern können. Damit sind die Prozeduren, welche für das Trainieren eines Modells verwendet werden, ebenso wichtig für die gesamtheitliche Leistung in der Detektion von Objekten. Diese werden in den folgenden Abschnitten auszugsweise beschrieben.

5.3.4.1 Loss Calculations

Die Loss Funktion von YOLO ist im Absatz 5.3.2.1 beschrieben. Im Gegensatz zur ursprünglichen Loss-Funktion wurde mit YOLOv5, neben anderen Optimierungen, bei der Loss-Funktion ein neuer Term «GIoU» eingeführt. Die Generalized Intersection over Union (GIoU) ist eine Metrik und wird in der Loss-Funktion für die Bounding Box Regression verwendet. Dabei wird die Schwäche des klassischen IoU, nicht optimiert werden zu können, wenn z.B. zwei Bounding Boxes nicht überlappen (vorhergesagte Bounding Box und Ground Truth Box), adressiert und behoben (Rezatofighi et al. 2019).

5.3.4.2 Data Augmentation

Mit jedem Trainings-Batch schickt YOLOv5 die Trainings-Daten durch einen Data-Loader, welcher die Bilder augmentiert. Dabei werden drei Arten von Bild-Anpassungen vorgenommen:

- Skalierung
- Farbraum-Anpassung
- Mosaik-Augmentation

Die neuste Erweiterung ist dabei die Mosaik-Augmentation, wo jeweils vier Trainings-Bilder in ein einzelnes Bild mit 4 Kacheln von dynamischer Grösse kombiniert werden. Diese Augmentation soll vor allem beim «Small Object Problem» helfen, wo kleine Objekte in einem Bild nicht gleich akkurat detektiert werden, wie grosse Objekte. Die Mosaik-Augmentation auf dem Trainings-Batch ist in Abbildung 12 auszugsweise dargestellt.

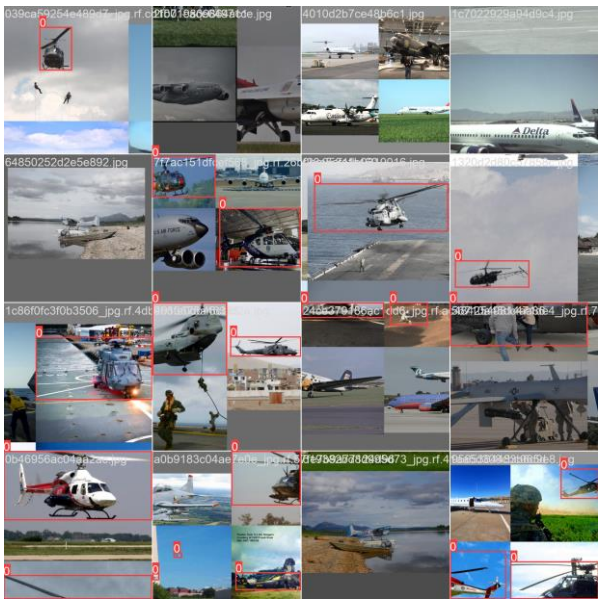


Abbildung 12: Mosaik-Augmentation Trainings-Batch

5.3.4.3 Auto Learning Bounding Box Anchors

Mit der ersten PyTorch-Version von YOLO hat Ultralytics ein neues Konzept für die Anker-Boxen eingeführt. Dabei werden, basierend auf der Verteilung der Ground Truth Bounding-Boxen im individuellen Datensatz, die «Anchor Bounding Boxes» mithilfe von K-means und einem genetischen Lern-Algorithmus selbständig neu gelernt und müssen nicht weiter manuell definiert werden. Die Bounding-Box Regression von YOLOv5 mit der Nutzung dieser Anker-Boxen ist im folgenden Kapitel 5.4 beschrieben.

5.3.4.4 16 Bit Floating Point Precision

Die Umsetzung von YOLOv5 mit dem PyTorch Framework lässt zu, dass die Floating Point Präzision von 32bit auf 16bit (FP16) halbiert werden kann. Damit lassen sich signifikante Verbesserungen der Inferenz-Zeiten erzielen.

5.4 Wie funktioniert Object Detection mit YOLO?

Der Algorithmus, um von einem Input Bild variabler Grösse bzw. einem Input-Stream (z.B. Video oder Mobiltelefon-Kamera-Stream) zu einem Output-Bild mit markierten Objekten (Bounding-Boxes inkl. Confidence) zu gelangen (vgl. Tabelle 2), nutzt YOLO folgende Ansätze:

1. Residual Blocks
2. Bounding Box Regression
3. Intersection Over Union (IOU)
4. Non-Maximum Suppression

Wie bereits erwähnt, werden in YOLOv5 die Vorhersagen (Predictions) grundsätzlich in der Form $\boldsymbol{S}$ x $\boldsymbol{S}$ x $(\boldsymbol{B}*\mathbf{5} + \boldsymbol{C})$ kodiert. Für die entsprechende Definition siehe Kapitel 5.3.2. Im GTSRB-Datensatz ist C = 43, bei See And Avoid ist C = 1.

Tabelle 2: Inferenz YOLOv5 mit GTSRB Datensatz (links: Input-Bild, rechts: nach der Inferenz)

5.4.1 Residual Blocks

In einem ersten Schritt wird das Input-Bild in S x S Gitter-Zellen mit der gleichen Grösse geteilt. Wenn nun das Zentrum eines Objekts in eine dieser Zellen fällt, ist diejenige Zelle dafür verantwortlich, das Objekt zu lokalisieren und die entsprechenden Objekt-Klasse vorherzusagen. Jede Gitter-Zelle sagt B Bounding Boxes mit den dazugehörigen Confidence-Scores voraus.

Diese Confidence-Scores reflektieren dabei, wie sicher das Modell ist, dass diese Bounding-Box ein Objekt enthält und wie akkurat diese Box ist.

Zusätzlich sagt jede Zelle C abhängige Klassen-Wahrscheinlichkeiten voraus: $Pr(Klasse_i|Objekt)$.

In P5 Modellen wie YOLOv5 werden hier anstelle einer fixen Grid-Grösse, drei multi-scale Outputs mit den Stride 8, 16 und 32 genutzt.

5.4.2 Bounding Box Regression

Jede dieser Bounding-Box Vorhersagen enthält konkret fünf Vorhersagen x, y, w, h und die entsprechende Confidence pc. Die x, y Koordinaten repräsentieren das Zentrum der Box relativ zu den Grenzen der umschliessenden Gitter-Zelle. Die Breite w und Höhe h werden relativ zur Gesamtgrösse des Bildes vorhergesagt. Wie vorangehend beschrieben, repräsentiert der Con-

fidence-Score grundsätzlich die IOU zwischen der vorhergesagten Box und der Ground Truth Box. Wenn kein Objekt in der Gitter-Zelle existiert, sollte der Confidence-Score 0 sein, ansonsten soll er dem IOU-Wert entsprechen.

YOLO bestimmt diese Attribute der Bounding-Box mit einem einzelnen Regression Modul im folgenden Format:

y = [*pc, x, y, h, w, c1, c2, c2,...c$_i$*]

5.4.3 Intersection over Union

Abgeleitet von den vorgegebenen Anker-Boxen, sagt YOLO pro Gitter-Zelle, als Offset zu diesen Anker, mehrere Bounding-Box Kandidaten voraus und optimiert diese in weiteren Schritten. Wie bereits erwähnt, soll während des Modell-Trainings nur ein Bounding-Box-Predictor für ein bestimmtes Objekt verantwortlich sein. Die Auswahl dieses Predictors geschieht basierend darauf, welche aktuelle Box-Voraussage den höchsten IOU mit der Ground Truth hat. Dies führt schliesslich zu Spezialisierungen der einzelnen Box-Predictors.

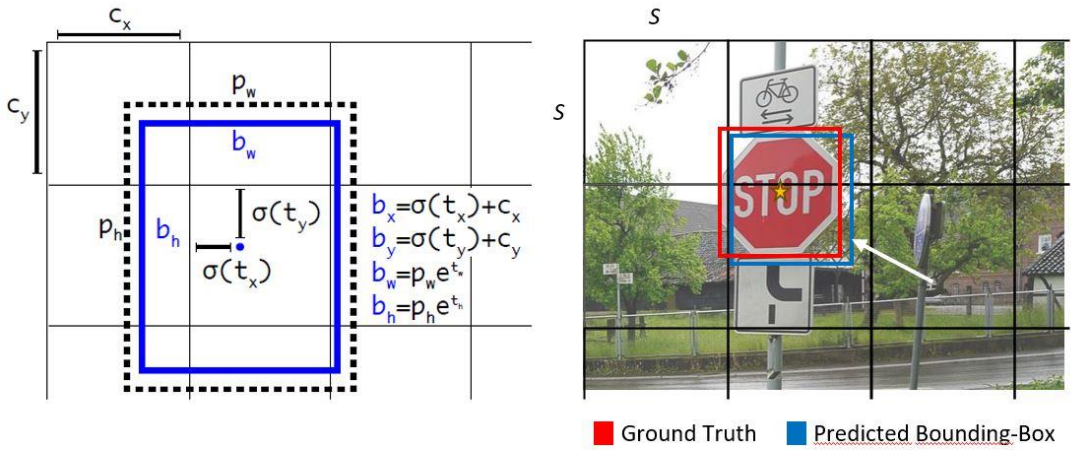


Abbildung 13: Bounding-Box Vorhersage basierend auf einer Anker-Box, Quelle: (Redmon und Farhadi 2018)

In Abbildung 13 ist beispielhaft gezeigt, dass im Input-Bild in der verantwortlichen Gitter-Zelle nur eine Bounding-Box für das detektierte «STOP»-Schild Objekt vorhergesagt wurde. Ist der Wert für die Intersection Over Union (IoU) dieser vorhergesagten Bounding-Box mit IoU = Area of Overlap / Area of Union grösser als der IoU-Wert aller anderen, allfälligen Box-Predictors für diese Zelle und grösser als ein bestimmter Schwellenwert (Threshold), wird diese Vorhersage weiterhin berücksichtigt, in den anderen Fällen aber ignoriert (Redmon und Farhadi 2018). Indem die unterschiedlichen Bounding Box Vorhersagen entweder belohnt oder bestraft werden (Loss-Funktion), wird das Modell langsam dahin geführt, echte Objekte zu lokalisieren.

5.4.4 Non-Maximum Suppression

Genau wie beim Trainieren des YOLO-Modells, wird auch bei der Object Detection für Test-Bilder nur ein Netz-Durchlauf benötigt. Der beschriebene Gitter-Netz Ansatz mit den Anker-Boxen erzwingt auch eine räumliche Verteilung der Bounding-Box Vorhersagen. Oft ist auch klar, welche Zelle für ein Objekt verantwortlich ist und das Modell sagt entsprechend nur eine Box pro Objekt voraus. In bestimmten Fällen aber, mit grossen Objekten oder Objekten nahe an der Grenze von verschiedenen Zellen, können verschiedene Gitter-Zellen diese Objekte gut de-

tektieren. Hier kann Non-Maximum Suppression (NMS) genutzt werden, um dieses Problem von multiplen Detektionen zu lösen. NMS ist eine Technik, welche vorwiegend in der Object Detection verwendet wird. Mit dieser Technik wird versucht, die beste Bounding-Box Vorhersage aus einem Set von Box-Vorhersagen zu wählen.

Der NMS-Algorithmus hat als Input eine Liste von Box-Vorhersagen *A* mit entsprechenden Confidence-Scores *S* und einen IoU-Schwellwert *N*. Die beste Bounding-Box wird dann wie folgt ausgewählt:

1. Auswahl der Box-Prediction mit dem höchsten Confidence-Score
2. Vergleich dieser Box mit allen anderen Bounding-Box Vorhersagen *i* aus dem Set basierend auf der IoU. Wenn der IoU-Wert grösser ist als *N*, wird die Prediction *i* verworfen
3. Dann wird wiederum die Box-Prediction mit der höchsten Confidence aus *A* ausgewählt
4. Mit der neu ausgewählten Box wird ebenfalls wieder Schritt 2 durchlaufen.
5. Dies passiert so lange, bis keine Box-Prediction mehr in der Input-Liste *A* ist.

5.5 Modell Training YOLOv5

5.5.1 Datensatz GTSRB

Das Ziel des ersten Experiments war es, ein Baseline-Resultat für die real-time Object Detection von Verkehrsschildern mit YOLO-Modellen zu setzen.

5.5.1.1 Daten-Preprocessing / Datensatz-Erweiterung

Der ursprüngliche GTSRB-Datensatz liegt aufgeteilt in einem Train- und Test-Split (~80%/20%) vor. Der Trainings-Satz ist zusätzlich unterteilt in Unterordner, einen für jede der 43 Objekt-Klassen. Die Annotationen der Bounding-Boxes und Klassen sind in einem einzelnen separaten csv-File für die Trainings- und Test-Bilder. Die Annotationen (vgl. Abbildung 14) enthalten folgende Informationen:

- Dateiname: Der File-Name des entsprechenden Bildes
- Width: Breite des gesamten Bildes
- Height: Höhe des gesamten Bildes
- ROI.x1: X-Koordinate der oberen linken Ecke der Bounding-Box
- ROI.y1: Y-Koordinate der oberen linken Ecke der Bounding-Box
- ROI.x2: X-Koordinate der unteren rechten Ecke der Bounding-Box
- ROI.y2: Y-Koordinate der unteren rechten Ecke der Bounding-Box
- ClassId: Zugewiesene Klasse (0-42)

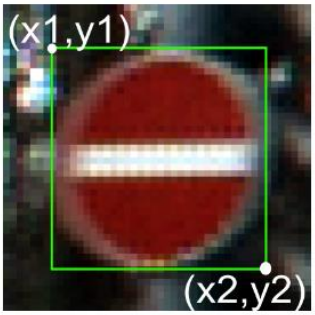


Abbildung 14: BBox Annotation GTSRB

Das YOLO-Input Format verlangt aber pro Trainings-Bild ein separates Text-File mit einem Zeilen-Eintrag pro Objekt in der Form:

[KlassenID] [X-Koordinate des Box-Zentrums] [Y-Koordinate Box-Zentrum] [Box-Breite] [Box-Höhe]

Die Koordinaten, sowie die Bounding-Box Höhe und Breite müssen auf die gesamte Bild-Breite bzw. Höhe normalisiert angegeben werden. Dürfen also keine Werte > 1 haben.

Der GTSRB Datensatz enthält auch keine «Negativ-Bilder» - d.h. Bilder, welche keine Verkehrsschilder enthalten (später als Background-Images referenziert). Die Autoren von YOLOv5 empfehlen aber rund 10% Background-Images (ultralytics/yolov5 2023). Um diese zusätzlichen Bilder zu generieren, habe ich zufällig Strassenverkehrs-Bilder ohne Verkehrsschilder selektiert und diese automatisiert in 64x64 Pixel grosse Kacheln unterteilt. Diese Kacheln habe ich anschliessend abgedunkelt und zusammen mit dem Original als jeweils neues Bild abgespeichert (vgl. Abbildung 15). Damit entstanden zur späteren Verwendung rund 7'000 Background-Images für diesen Datensatz.

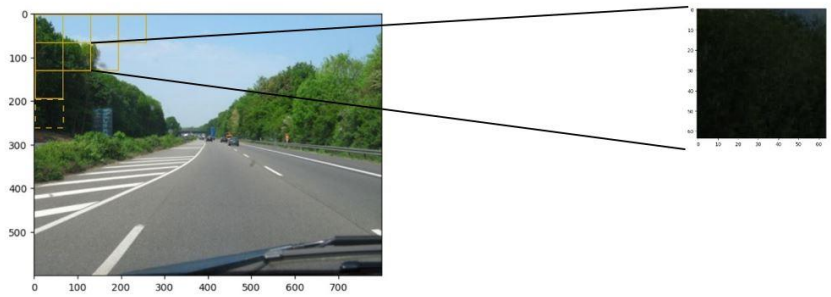


Abbildung 15: Generierung Background-Images

5.5.1.2 Experimente

Die Tabelle 3 zeigt einen Auszug der durchgeführten Experimente bzw. Trainings des YOLO-Modells mit dem German Traffic Sign Recognition Benchmark Datensatz. Mit einem initialen Lauf (vgl. Run35) über 100 Epochs auf dem vortrainierten YOLOv5 Small-Modell konnte bereits eine gute Baseline mit einer Mean Average Precision mAP(0.5) von rund 93% bzw. eine mAP(0.5:0.95) von 43% erreicht werden.

Exp. Name	Weights	Epochs	BatchSize	Augmentation	BG Images[5]	Image Size	mAP 0.5	mAP 0.5:0.95
Run28	Yolo5s	3	1024	Keine	keine	64	-	-
Run35	Best Run28	100	1024	Ja (Albumentation[6])	keine	64	0.93	0.43
Exp2	Yolo5s	100	128	Ja	keine	64	0.95	0.42
Exp3	Yolo5s	3	1024	Ja	keine	64	-	-
Exp4	Best Exp3	100	256	Ja	keine	64	0.98	0.45
Exp6	Yolo5s	100	256	Ja	Ja (5%)	64	0.94	0.43
Exp7	Yolo5s	100	128	**Ja**	**Ja (10%)**	64	0.94	0.51
Exp8	Yolo5s	100	128	**Ja (modifiziert)**	Ja (10%)	64	0.87	0.67

[5] Background Images
[6] Albumentation nach Buslaev et al. 2020.

Exp9	Yolo5s	3	1024	Keine	Ja	64	-	-
Exp10	Best Exp9	100	256	Keine	Ja	64	0.88	0.72
Exp11	Best Exp10	100	128	Keine	Ja	64	0.91	0.77
Exp12	Yolo5s	100	128	Ja	Ja	64	-	-
Exp15	Best Exp12	100	256	Ja	Ja	64	-	-
Exp16	Best Exp15	100	128	Ja	Ja	64	0.9	0.72

Tabelle 3: Experimente YOLO/GTSRB (Best Epoch)

Abbildung 16 und Abbildung 17 zeigen die Entwicklung verschiedener Metriken bzw. Losses im Verlauf der einzelnen YOLO-Trainings vom Start bis zur letzten Epoch. Die Losses (siehe Kapitel 5.3.2.1 Loss Function) werden sowohl für das Trainings-, wie auch das Validation-Set angezeigt. Gezeigt werden:

- **Training-Losses**
  - Box Loss (train/box_loss)
  - Confidence Loss (train/obj_loss)
  - Classification Loss (train/cls_loss)
- **Validation-Losses**
  - Box Loss (val/box_loss)
  - Confidence Loss (val/obj_loss)
  - Classification Loss (val/cls_loss)
- **Metriken** (vgl. Kapitel 2.4)
  - Precision
  - Recall
  - mAP(0.5)
  - mAP(0.5:0.95)

Die relativ hohe Mean Average Precision von Experiment «Run35» täuscht etwas über die effektiv erreichte Inferenz-Leistung hinweg. Bei genauerer Betrachtung stellt man fest, dass der Bounding-Box-Loss (box_loss) und der Klassifikations-Loss (cls_loss) zwar mit zunehmender Epoch stetig abfallen, der Object-Loss aber stark oszilliert und gegen Ende des Trainings wieder deutlich zunimmt. Auch die mAP(0.5:0.95) fällt nach rund 2/3 der Epochs deutlich ab (vgl. orange Markierungen in Abbildung 16).

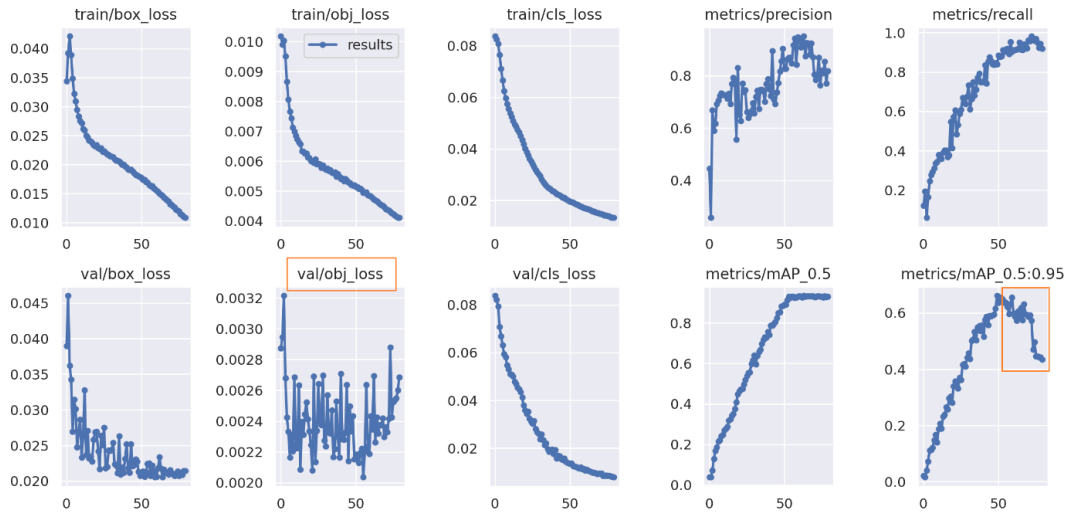


Abbildung 16: Metriken Experiment "Run35"

Insbesondere die Confusion-Matrix nach der Validierung zeigt bei einigen, ähnlichen Klassen falsche Vorhersagen (vgl. Anhang B.). So wird die Klasse «Dangerous Curve Right» oft falsch als «Slippery Road» klassifiziert. Diese beobachteten Effekte können auch auf ein mögliches Overfitting des Modells hindeuten. Interessanterweise hat aber die Anpassungen der beiden Hyperparameter «Weight Decay» und der «Learning Rate» zwischen einzelnen Experimenten weder einen nennenswerten Einfluss auf die Leistung des späteren Modells, noch konnten diese Effekte bei den Losses und Confusion-Matrix abgeschwächt werden. Aufgrund der gegebenen Situation kann davon ausgegangen werden, dass allenfalls auch eine zu geringe Bild-Varianz während des Trainings zu den genannten Problemen geführt hat. Aber auch der Einsatz unterschiedlicher Bild-Augmentationen (in Art und Stärke) in der Folge hat auf die Inferenz-Leistung bei neuen Test-Bildern kaum Effekte. Ein weiterer spannender Aspekt ist, dass ab einer gewissen Grössenordnung der Anzahl Background-Images (>10% der Gesamtanzahl Trainingsbilder) der Verlauf des Object-Losses aber deutlich geglättet und damit auch die Güte des Modells bei der Erkennung von Verkehrsschildern verbessert werden konnte. Die Trainings-Metriken von Experiment «Exp7» zeigen diese deutliche Verbesserung (vgl. Abbildung 17).

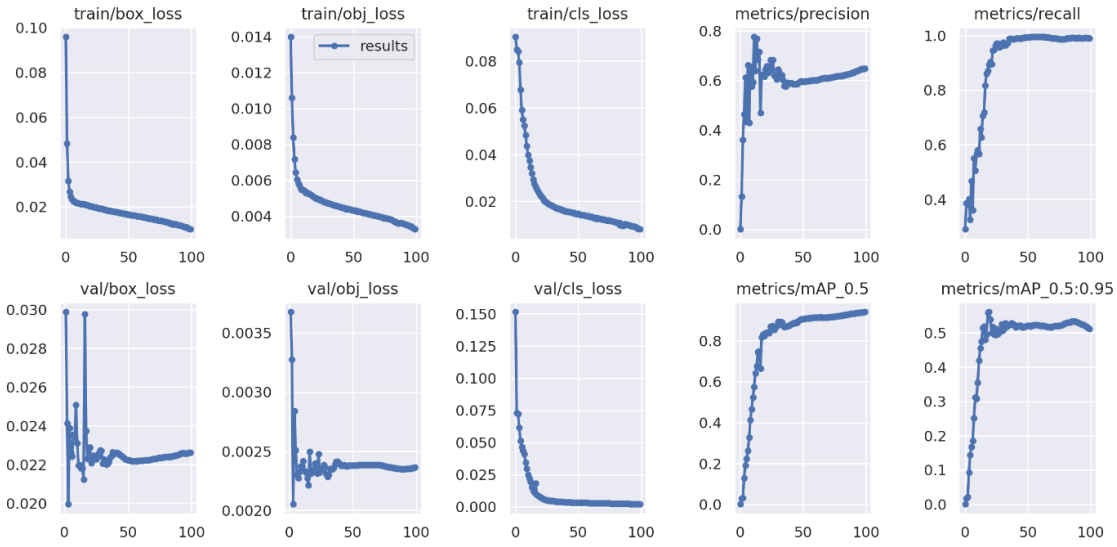


Abbildung 17: Metriken Experiment "Exp7"

5.5.2 Datensatz See And Avoid

Auch für den See And Avoid Datensatz sollte zuerst eine Baseline gesetzt werden. Der Datensatz mit den Klassen «Airplane», «Helicopter» und «Balloon» wurde in einen Train-, Test- und Validation Split geteilt (~75%/15%/10%).

5.5.2.1 Daten-Preprocessing/Datensatz-Erweiterung

In der ursprünglichen Form des Datensatzes ist bereits ein Annotationen-File pro Trainings-Bild vorhanden. Jede Zeile in diesem Text-File beschreibt die Bounding-Box für ein Objekt, jedoch im Format [ImageID] [LabelName] [Confidence] [XMin] [XMax] [YMin] [YMax] und weiteren zusätzlichen Feldern. Diese Files habe ich automatisiert ins YOLO-Format überführt.

Abbildung 18: False Positive (Inferenz Mobiltelefon)

Die ersten Modell-Trainings, welche wiederum auf dem vortrainierten YOLOv5 Small Modell ausgeführt wurden, haben mit einer mAP(0.5) von rund 72% und einer mAP(0.5:0.95) von 49% nicht die gewünschte Genauigkeit erreicht. Zudem hat die Confusion-Matrix nach der Validierung gezeigt, dass sehr viele «False Positives», sprich viele Objekte vorhergesagt wurden, wo in der Ground Truth gar keine Objekte vorhanden sind. Dies hat sich insbesondere während der Inferenz auf dem Mobiltelefon gezeigt, wo der Hintergrund fast ständig als Objekt (Airplane) detektiert wurde. Die Abbildung 19 mit der Confusion-Matrix zeigt, dass bei vielen Detektionen (87%) Hintergrund (True Label) als Airplane (Predicted Label) erkannt wurde.

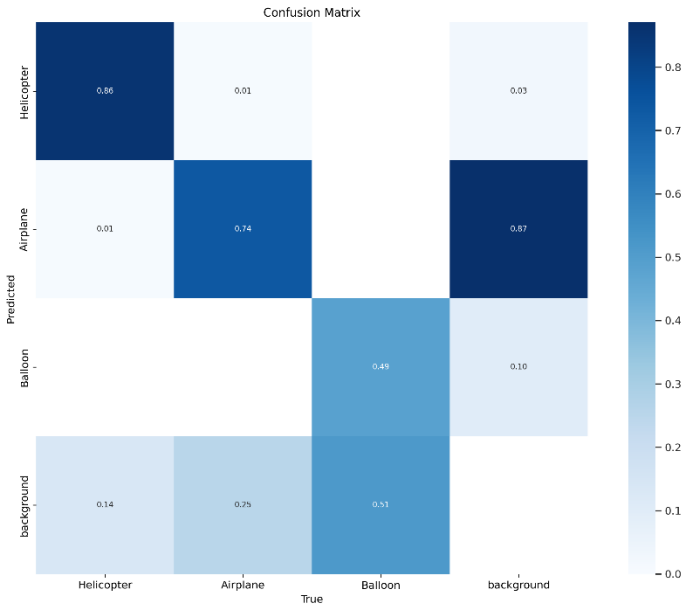


Abbildung 19: Confusion-Matrix Exp21 YOLO/See And Avoid

Aus diesem Grund habe ich zunächst die Ground Truth Labels des Trainings-Datensatzes visualisiert und mit Stichproben geprüft. Nachdem einige Bilder mit schlecht annotierten Bounding-Boxes gefunden wurden, habe ich eine halb-automatisierte Grob-Prüfung und Korrektur der Annotation-Files vorgenommen. In der Klasse «Balloon» waren überwiegend Bilder mit Party-Ballonen und nur wenige Heissluftballone vorhanden. Deshalb habe ich mich dafür entschieden, diese Klasse gänzlich aus den Trainings-Daten zu entfernen.

Bei der Klasse «Airplane» waren viele Objekte falsch klassifiziert, mit unpassenden Bounding-Boxes markiert oder zeigten lediglich gezeichnete oder unechte Objekte[7]. Weitere Trainings auf dem bereits bereinigten Datensatz mit den beiden Klassen «Airplane» und «Helicopter» zeigten aber keine wesentliche Verbesserung.

Folglich habe ich den ursprünglichen Datensatz auf die Klasse «Helicopter» reduziert und sämtliche verbleibenden Bilder manuell geprüft, korrigiert und als neuen Datensatz «See And Avoid Small» definiert. Die Abbildung 20 zeigt links beispielhaft ein ungenaues und rechts ein fehlendes Bounding-Box Label im Open Image Datensatz.

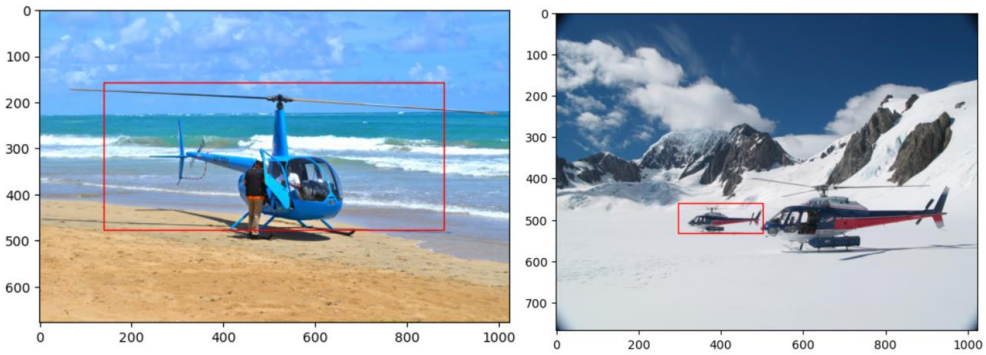


Abbildung 20: Unpassende/fehlende Bounding-Box Labels Open Image v7

Der neue Datensatz umfasst rund 1900 Bilder der Klasse «Helicopter» und wurde wiederum in einen Train-, Test- und Validation-Split geteilt (85%/10%/5%).

[7] Obwohl der Open Image v7 Datensatz über einen Parameter –image_IsDepiction gefiltert heruntergeladen werden kann

5.5.3 Datensatz See And Avoid Small

Auf diesem reduzierten Datensatz habe ich wiederum unterschiedliche Experimente durchgeführt. Neben der Anzahl Background-Images habe ich zusätzliche Variationen in der Bild-Augmentation geprüft. Basierend auf den Erkenntnissen aus den Experimenten mit dem vollständigen See And Avoid Datensatz, habe ich unterschiedliche Hyperparameter-Konfigurationen angewendet und die Resultate miteinander verglichen. Bei einzelnen Experimenten habe ich zusätzlich den Effekt untersucht, der auftritt, wenn das Training auf dem grösseren vortrainierten YOLO-Modell Medium ausgeführt wird oder die Gewichte von Grund auf neu trainiert werden. Mit dem Training auf dem grösseren Modell konnte zu Lasten des Trainingsaufwandes zwar eine teilweise Verbesserung der beiden mAP-Metriken erzielt werden, gleichbleibend blieb aber das Problem mit der Confusion-Matrix und der schlechten Inferenz auf dem Mobiltelefon. Gleiche Effekte zeigten sich bereits bei den Experimenten mit dem grossen Datensatz (vgl. Kapitel 5.5.2). Die Tabelle 4 zeigt einen Auszug der ausgeführten Experimente mit dem YOLO-Modell auf dem «See And Avoid Small»-Datensatz.

Exp. Name	Weights	Epochs	BatchSize	Augmentation	BG Images[8]	Image Size	mAP 0.5	mAP 0.5:0.95
Exp35	Yolo5s	100	32	Anpassungen (Blur, Mosaic)	keine	1024	0.98	0.7
Exp13	Yolo5s	100	16	Ja (Albumentation)	keine	640	0.99	0.86
Exp37	From scratch	300	16	Anpassungen (Blur, Mosaic)	keine	640	0.97	0.76
Exp59	Yolo5s	100	16	Anpassungen (Blur, Mosaic)	>10%	640	0.95	0.78

Tabelle 4: Experimente YOLO/See And Avoid Small (Best Epoch)

Erst mit der Erweiterung des Trainings-Datensatzes um >10% Background-Bilder wurde die False Positive-Rate wieder deutlich reduziert und damit die Inferenz auf dem Mobiltelefon deutlich besser. Auch die Confusion-Matrix nach der Validierung des Modells zeigt mit einer guten Verteilung diesen Effekt (vgl. Anhang C.). Mit diesem verbesserten Modell konnten Detektionen reduziert werden, wo ganze Flächen im Hintergrund oder auch nur einfache Kanten als Helikopter detektiert und klassifiziert wurden (vgl. Abbildung 18).

[8] Background Images

6. DETR Detection Transformer

6.1 Einführung

DETR oder Detection Transformer ist ein neueres Object Detection Modell, welches aus dem Bereich des «Natural Language Processing» (NLP) inspiriert ist. DETR wurde mit dem Paper «End-to-End Object Detection with Transformers» vom Facebook AI Forschungsteam eingeführt (Carion et al. 2020) und baut auf dem multi-headed Attention-Modell von (Vaswani et al. 2017) auf. DETR ist ein set-basierter Ansatz, welcher wie z.B. YOLO ebenfalls auf einem Convolutional Backbone aufsetzt. Bei DETR wird auf dem Backbone aber ein Encoder-Decoder Transformer eingesetzt. Der konventionelle CNN-Backbone wird genutzt, um eine 2D Repräsentation des Input-Bildes zu lernen, dieses mit «Positional Encodings» zu erweitern und dem Transformer-Encoder weiterzugeben. Der Transformer-Decoder nimmt in der Folge als Input die «Encoder-Output-Attention» zusammen mit einer vordefinierten Anzahl gelernter Positional Embeddings (sogenannter Object Queries). Anschliessend wird jedes Output-Embedding des Decoders für die Vorhersage von Bounding-Boxes und Klassenlabels oder der «No-Object» Klasse einem Feed-Forward Network (FFN) zugeführt. Der Aufbau von DETR ist in Abbildung 21 schematisch dargestellt.

6.2 Architektur

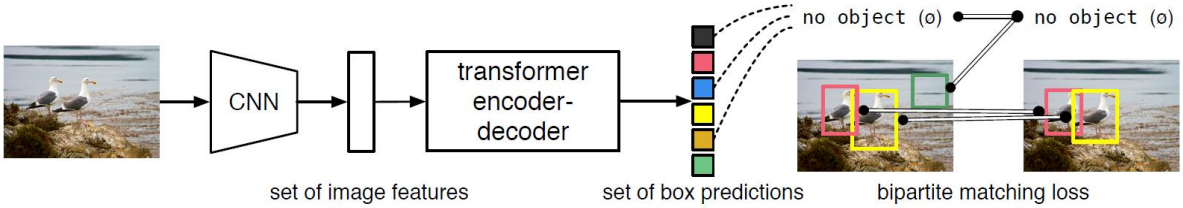


Abbildung 21: Übersicht DETR mit Bipartite Matching zwischen Vorhersagen und Ground Truth, Quelle: (Carion et al. 2020)

Zwei Bestandteile sind essenziell für eine direkte set-basierte Vorhersage in Object Detection:

- Ein Set-Prediction-Loss, welcher eine eindeutige Zuweisung zwischen den vorhergesagten Bounding Boxes und der Ground Truth erzwingt (Bipartite Matching)
- Eine Architektur, die in einem single-pass ein Set von Objekten und deren Beziehung zueinander vorhersagt

Die Gesamt-Architektur von DETR ist relativ einfach und besteht aus den folgenden drei Hauptkomponenten, wie in Abbildung 22 gezeigt:

- CNN-Backbone
- Encoder-Decoder Transformer
- Feed-Forward Network

6.2.1 Backbone

Ausgehend von einem Input-Bild mit drei Farbkanälen und der Höhe H_0 und Breite W_0 generiert das Convolutional Neural Network (z.B. ResNet-50) eine Activation- bzw. Feature-Map mit einer kleineren Auflösung und den Dimensionen C x H x W. Dabei sind typische Werte für C = 2048 und $H = H_0/32$, $W = W_0/32$. In diesem Schritt werden die Features aus den Input-Bilder extrahiert (ähnlich dem Backbone in der YOLO-Architektur). Um sicher zu stellen, dass alle Input-Bilder eines Batchs die gleiche Grösse haben, werden die Dimensionen des grössten Bildes genommen und für die restlichen Bilder ein 0-padding durchgeführt.

6.2.2 Transformer-Encoder

Da der Encoder eine Sequenz als Input erwartet, wird die vorher verkleinerte Activation-Map aus dem Backbone über eine 1x1 Convolution in eine kleinere Dimension *d* überführt und ausgeflacht. Dies führt zu einer Feature-Map mit den Dimensionen *d* x *HW*. Der Encoder folgt der Standard-Architektur (Vaswani et al. 2017) und besteht aus einem multi-headed Attention Modul und einem Feed Forward Network. Zusammen mit den Positional-Encodings wird jeder Input Feature-Vektor in eine gleich lange Output-Sequenz von Feature-Vektoren transformiert.

6.2.3 Transformer-Decoder

Auch der Decoder folgt grundsätzlich der Standard-Architektur von Transformer. Mit den Output-Embeddings vom Encoder (Encoder-Memory) als Side-Input, transformiert der Decoder *N* Input-Embeddings (mit 0 initialisierte Object Queries) der gleichen Grösse *d*. Die Transformation des Decoders erfolgt damit ebenfalls unter der Nutzung eines multi-headed self- and encoder-decoder Attention-Mechanismus. Die Output-Embeddings des Decoders entsprechen den eigentlichen Bounding Box Predictions.

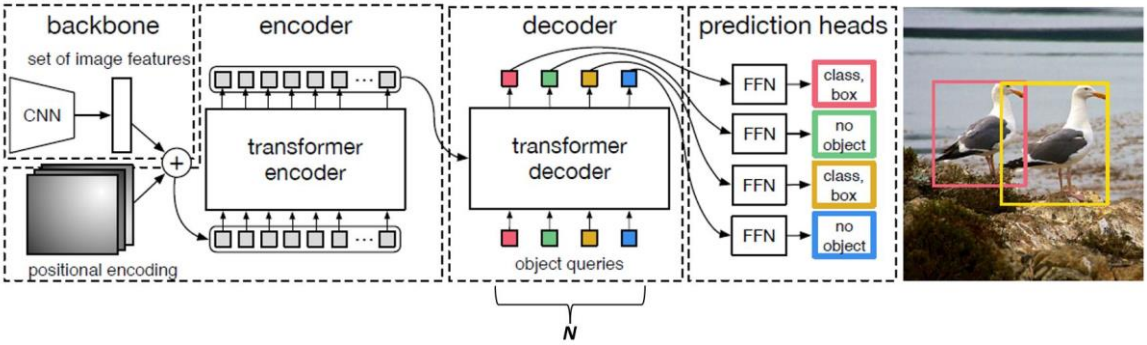


Abbildung 22: DETR-Architektur, Quelle: (Carion et al. 2020)

Bildlich gesprochen können die *N* Object Queries als Queries bezeichnet werden, die durch den Attention-Mechanismus im Decoder trainiert bzw. gelernt werden, sich auf verschiedene Features/Bereiche des Input Bildes zu konzentrieren. Die Anzahl der Object Queries definiert auch die maximale Anzahl an Objekten, die in einem einzelnen Bild detektiert werden können. Standard-Wert von *N* = 100. Vergleiche Anhang D. für die Detail-Architektur des Transformers von DETR.

6.2.4 Final Prediction Feed-Forward Network

Die Output-Embeddings des Decoders werden zur finalen Vorhersage an ein geteiltes (shared) Feed Forward Network (FFN) weitergegeben. Das FFN besteht aus einem 3-Layer Perzeptron mit der ReLU Aktivierungs-Funktion, Hidden Dimension *d,* gefolgt von einem linearen Layer. Durch dieses Netzwerk werden die normalisierten x- und y-Koordinaten des Bounding-Box Zentrums und die Breite bzw. Höhe der Box aus den Embeddings entschlüsselt und vorhergesagt. Mittels einer Softmax-Funktion sagt der Linear-Layer letztlich die Klassen-Labels voraus. Da in jedem Fall die festgelegte Anzahl *N* an Bounding-Boxen vorhergesagt wird und *N* üblicherweise deutlich grösser ist als die eigentliche Anzahl an Objekten, wurde das spezielle Klassen-Label «No object» eingeführt.

Ein Bipartite-Matching Algorithmus wird genutzt, um jedes vorhergesagte Objekt mit einem Ground-Truth Objekt oder eben dem «No object» zu verknüpfen. Wenn ein Mismatch bei den Objekten besteht, ist der Algorithmus gezwungen Vorhersagen mit wahren Objekten einer ande-

ren Klasse zu verbinden. Wenn hingegen zu viele Objekte vorhergesagt werden, wird eine Verknüpfung mit dem «No object» erzwungen (Wilson 2023). Durch diesen Aufbau lernt das Modell die richtige Anzahl an Objekten vorherzusagen. Deshalb kann bei der DETR-Architektur im Gegensatz zur YOLO-Architektur auch auf den Einsatz von Non-Maximum Suppression verzichtet werden.

6.3 Modell-Training DETR

Um einen sauberen Vergleich zwischen YOLO und DETR in Bezug auf den Trainings-Aufwand, die Modell-Genauigkeit und Inferenz-Geschwindigkeit machen zu können, wurde DETR ausschliesslich mit dem bereinigten Datensatz «See And Avoid Small» trainiert (vgl. Kapitel 5.5.3).

6.3.1 Datensatz See And Avoid Small

Das Ziel des ersten Experiments mit DETR war es wiederum, ein erstes Baseline-Resultat für die real-time Object Detection von Helikoptern zu setzen. Tabelle 5 zeigt einen Auszug der durchgeführten Experimente.

Exp. Name	Weights[9]	Epochs	BatchSize	Weight Decay	BG Images[10]	Object Queries	Trainigs-Dauer (h)	mAP 0.5	mAP 0.5:0.95
Run1	ResNet-50	50	4	0.0001	Keine	100	-	0.86	0.71
Run2	ResNet-50	100	8	0.0001	Keine	100	-	0.88	0.73
Run5	ResNet-50	100	4	0.0001	Keine	100	11.2	0.92	0.77
Run6	ResNet-50	100	4	0.0001	>10%	100	11.1	0.874	0.73
Run7	ResNet-50	100	4	0.0005	>10%	40	11	0.879	0.75
Run8	ResNet-50	100	4	0.0005	Keine	40	11	0.927	0.75
Run9	ResNet-18	100	4	0.0001	Keine	40	7.25	0.61	0.31
Run10	ResNet-18	100	4	0.0001	>10%	40	7	0.58	0.29
Run11	ResNet-18	100	4	0.0001	Keine	5	7.25	0.57	0.28

Tabelle 5: Experimente DETR/See And Avoid Small

Wie aus der Tabelle 5 und Abbildung 23 entnommen werden kann, konnte mit dem ResNet-50 Backbone erwartungsgemäss eine wesentlich bessere Mean Average Precision (~+30%) erreicht werden als mit dem RestNet-18 Backbone. Deutlich ist aber auch, dass der Trainingsaufwand mit knapp 11 gegen 7 Stunden deutlich grösser ist.

Vergleicht man Run5 bis Run8 miteinander, ist zu erkennen, dass unabhängig von der Anzahl der Object Queries eine ähnlich hohe Genauigkeit erreicht wurde. Da schon bei den Experimenten Run5/Run6 ein Anstieg des Validation Losses gegen Ende des Trainings zu erkennen ist, wurde für die nächsten Experimente eine grössere Weight-Decay gewählt. Ein allfälliges Overfit-

[9] Eingesetzter Backbone
[10] Background Images

ting sollte damit abgeschwächt werden. Interessant ist aber die Erkenntnis, dass bei den Runs mit der tieferen Anzahl von 40 Object Queries die Kurve des Validation-Losses schon im ersten Drittel des Trainings wieder deutlich ansteigt, was ebenfalls auf ein Overfitting des Modells hindeutet. Mit dem, von den DETR-Autoren gewählten Standard-Wert von 100 Object Queries erreicht man eine wesentlich bessere Kurve für den Validation-Loss. Was in dieser Untersuchung offen bleibt, ist die Frage, welchen Einfluss der hierbei unterschiedliche Wert für die Weight-Decay hat. Weiterführende Experimente mit 100 Object Queries, aber dem kleineren Wert für die Weight-Decay von 0.0001 könnten hier bessere Aussagen zulassen.

Gut zu erkennen ist auch, dass der Einsatz von Background-Images keinen positiven Effekt auf die Modell-Genauigkeit hat. Wie die Loss-Kurven dieser Experimente zeigen, kann, im Gegensatz zu den Experimenten mit den YOLO-Modellen, hier auch nicht davon ausgegangen werden, dass ein allfälliges Overfitting durch den Einsatz von >10% Background-Images abgeschwächt oder eine bessere Inferenz erzielt werden kann.

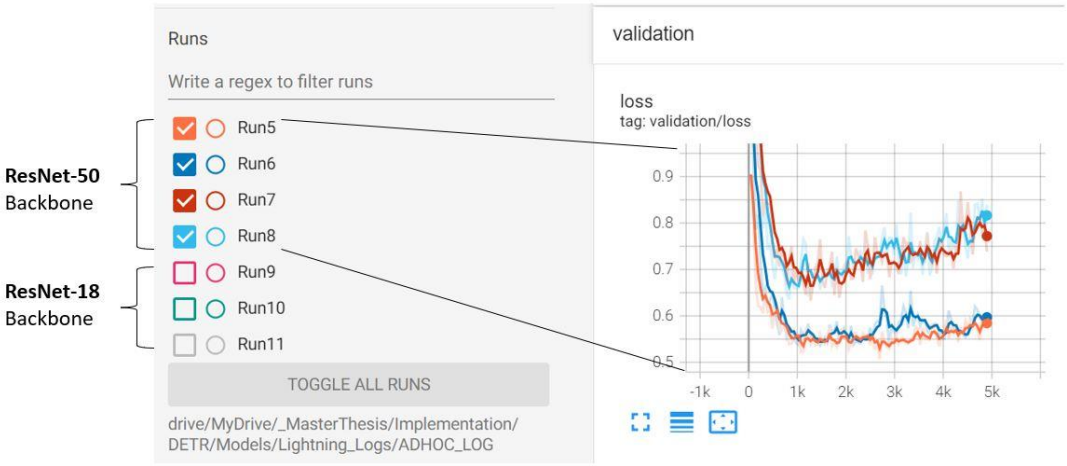


Abbildung 23: Validation-Loss Run5-11 TensorBoard

7. Modell-Vergleich

7.1 Architektur

Im Bereich der Object Detection wurden bereits viele unterschiedliche Methoden entwickelt. Wie einführend aufgezeigt, gibt es two-stage Architekturen, die in einem ersten Schritt beispielsweise «Selective Search» oder «Region Proposal Net» (RPN) nutzen, um sogenannte Region Proposals für die spätere Object Detection zu machen. Dann folgen Klassifizierungs- und Regressions-Prozesse, um die Bounding-Boxes und Klassen final vorherzusagen.

Im Gegensatz dazu stehen die one-stage Architekturen, wie die von YOLOv5. Bei solchen Modellen erfolgt die Objekt-Klassifizierung und die Bounding-Box Regression in einem direkten Durchgang. Diese Methoden brauchen aber verschiedene handgefertigte Variablen, wie die Anker-Boxen oder den IoU-Threshold, welcher für die Non Maximum Suppression benötigt wird.

Der neue, transformer-basierte Ansatz von DETR vereinfacht die Object Detection Pipeline dahingehend, dass die handgefertigten Komponenten, die quasi Vorwissen kodieren, gänzlich entfernt werden (Carion et al. 2020). Dennoch kann DETR ein fixes Set von *N* Vorhersagen in einem einzelnen Durchgang durch den Decoder vornehmen.

Gemeinsam haben die beiden Architekturen YOLO und DETR aber den Convolutional Neural Network-Backbone, um Features aus den Input-Bildern zu extrahieren. Zudem ist auch der Prediction-Head mit den zwei Fully Connected Layers in YOLO bzw. den Feed Forward Networks in der DETR-Architektur zur finalen Vorhersage der Bounding-Boxes und den dazugehörigen Labels ähnlich.

Die Autoren beider Modelle erwähnen, dass die gewählten Architekturen eine schwächere Leistung bei der Detektion von kleinen Objekten haben (Carion et al. 2020). Bei YOLO insbesondere bei kleinen Objekten, die in Gruppen (wie z.B. einem Vogelschwarm) auftreten (Redmon et al. 2015).

7.2 Training und Inferenz

Auch wenn mit beiden Modellen auf dem «See And Avoid Small»-Datensatz (~1'900 Bilder) mit einer mAP(0.5) >90% / mAP(0.5:0.95) >75% eine ähnlich hohe Mean Average Precision erreicht wurde, zeigen sich insbesondere im Trainings-Aufwand doch deutliche Unterschiede. Das YOLOv5 Modell musste hierbei rund 1 Stunde trainiert werden (vgl. YOLO-Experiment exp59). Im Gegensatz dazu aber lag der Trainings-Aufwand beim DETR-Modell mit dem ResNet-50 Backbone (Experiment Run5) bei über 11 Stunden.

Dieser Unterschied, der offensichtlich auf die Architektur der beiden Modelle zurückzuführen ist, zeigt sich ebenfalls in den Inferenz-Zeiten. Das YOLO-Modell erreicht auf neuen Test-Bildern eine durchschnittliche Inferenz-Zeit (inkl. NMS) von 19 Millisekunden. Die Inferenz mit DETR auf den gleichen Bildern dauert im Durchschnitt über 300 Millisekunden. Diese Tests auf der ZHaW-Openstack Umgebung (vgl. Kapitel 4), aber auch die Inferenz-Tests auf dem Mobiltelefon (nur YOLO) zeigen, dass mit dem YOLOv5-Modell durchaus real-time Object Detection auf mobilen Endgeräten durchgeführt werden kann. Die Tauglichkeit von DETR diesbezüglich ist aber eingeschränkt.

Die folgenden Beispiele in der Tabelle 6 zeigen, dass sich zwischen den YOLOv5 und DETR-Modellen trotz der ähnlich hohen mAP doch diverse Unterschiede bei der Inferenz zeigen. Grundsätzlich zeigt DETR bei richtigen Detektionen eine deutlich höhere Confidence, erkennt aber auch deutlich mehr False Positives. Wie erwartet, zeigen zudem beide Modelle Schwächen bei der Detektion von kleinen Objekten, insbesondere wenn sie nahe beieinander liegen.

YOLOv5 (Modell aus Exp59)	DETR (Modell aus Run5)
	

Yolo erkennt mit einer Confidence von 0.94 und 0.93 beide Helikopter sauber. DETR hat für beide Objekte eine Confidence = 1, erkennt aber auch ein False Positive.

Beide Modelle erkennen nur zwei von drei Helikoptern. Die beiden Helikopter, die auf dem Bild hintereinander liegen, werden als ein einzelnes Objekt detektiert.

Von den sechs nahe beieinander liegenden Helikoptern erkennt YOLO nur zwei und DETR vier. Allerdings sind die vorhergesagten Bounding Boxes bei beiden Modellen sehr ungenau.

	
YOLO erkennt kein Objekt. DETR hingegen detektiert sowohl den Helikopter als auch das Flugzeug im Hintergrund als Helikopter mit einer Confidence von über 90%.	
	
YOLO links erkennt den einzigen Helikopter im Bild (roter Kreis in der Bildmitte) nicht und detektiert fälschlicherweise den Teil einer Fahnenstange als Helikopter (rechts oben im Bild). DETR hingegen hat gar kein Objekt detektiert.	
	
YOLO detektiert keine Objekte. DETR aber detektiert wieder ein Flugzeug (Bildmitte) als Helikopter.	

Tabelle 6: Vergleich Inferenz (Testbilder) YOLO/DETR

7.3 Modell-Grösse

Auch die Grösse des trainierten Modells unterscheidet sich bei YOLO und DETR wesentlich. So hat das DETR-Modell (Run5) rund 28.5 Mio. Parameter und benötigt rund 115 MB physischen Speicher. Bei YOLOv5 (Exp59) hingegen sind es nur rund 7.1 Mio. Parameter mit einem Speicherbedarf von durchschnittlich nur 20 MB.

8. Mobile Object Detection Applikation

Ein weiteres Ziel dieser Masterarbeit war es, einen Prototypen einer mobilen Applikation (App) zu entwickeln und aufzuzeigen, inwiefern und wie ein trainiertes Computer Vision Modell in eine App integriert werden kann. Zudem soll gezeigt werden, wie real-time Object Detection unter Nutzung der Mobiltelefon-Kamera umgesetzt werden kann.

Aufgrund der verfügbaren Hardware wurde die Anwendung für iOS-Geräte (iPhone 13 mini) entwickelt. Beim Modell habe ich mich aufgrund der Inferenz-Performance und der geringen Grösse für YOLO entschieden.

8.1 Modell-Konversion

Für die Integration eines PyTorch-Modells in eine App, muss das Modell zuerst in ein passendes Format konvertiert werden (vgl. Abbildung 24). Bei Android-Geräten erlaubt TensorFlow Lite (TFLite) die direkte Integration von TFLite-Modellen in die Android-App. TFLite ist eine mobile Library, um Modelle auf mobilen Geräten oder Mikrokontrollern bereitzustellen. Wenn ein PyTorch-Modell vorliegt, muss dieses z.B. über Open Neural Network Exchange (ONNX) (Introduction to ONNX - ONNX 1.15.0 documentation 2023) zuerst in ein TensorFlow Modell und anschliessend in ein TFLite konvertiert werden. Das Modell bzw. die Inferenz kann bei Android-Geräten anschliessend auf der CPU, der GPU und NNAPI ausgeführt werden.

Bei iOS-Geräten hingegen können mittels des Vision-Frameworks (Apple Developer Documentation 2023c) von Apple CoreML-Modelle (Apple Developer Documentation 2023a) in iOS-Applikationen integriert und deployed werden. Dazu muss ein PyTorch-Modell in ein CoreML überführt werden. Hierfür hat man die folgenden zwei Optionen:

1. Direkte Konvertierung von PyTorch zu CoreML mit den zwei Schritten:
    a. Extrahieren der TorchScript Repräsentation des Modells mit JIT Tracer
    b. Dieses «traced» Modell mit den «coremltools» von Apple in das CoreML Format konvertieren
2. Das PyTorch Modell ebenfalls via ONNX Format in CoreML umwandeln

Bei der Inferenz auf iOS-Geräten hat man als Entwickler den Vorteil, dass das Vision- bzw. CoreML-Framework automatisch die Hardware für die Ausführung der einzelnen Layers wählt.

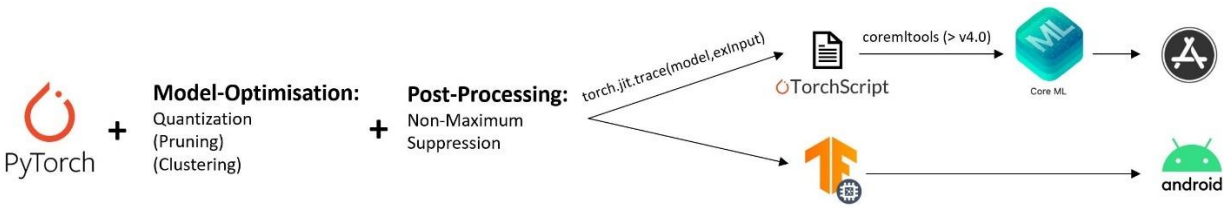


Abbildung 24: Modell-Konversion iOS und Android

8.1.1 Model-Optimierung und Post-Processing

Insbesondere bei der Anwendung von Computer-Vision Modellen auf mobilen Endgeräten stehen oft nur limitierter Speicher und limitierte Rechen- bzw. Batterieleistung zur Verfügung. Aus diesem Grund ist es hier besonders wichtig, die Modelle in Bezug auf Grösse oder Latenz bei der Inferenz zu optimieren. Natürlich können solche Optimierungen auch zu Lasten der Genauigkeit des Modells gehen. Diese Effekte müssen bei der Applikations-Entwicklung berücksichtigt werden.

Die betrachteten Frameworks unterstützen verschiedene Optimierungen. So bietet z.B. TensorFlow Lite Quantization, Pruning und Clustering (TensorFlow). Auch PyTorch bietet Möglichkeiten wie Quantization und Pruning oder andere Methoden (z.B. Fusing Layers), um die Komplexität und den Speicherbedarf der Modelle zu reduzieren und damit z.B. die Inferenz-Geschwindigkeit und Modell-Genauigkeit zu erhöhen.

8.1.1.1 Quantization

Bei der Quantization werden die Datentypen der Parameter von einigen oder auch allen Modell-Schichten in kleinere Datentypen umgewandelt. Default PyTorch Modelle brauchen beispielsweise die 32bit Floating Point Repräsentation für die Gewichte und Berechnungen. In meiner Anwendung habe ich die Präzision in einen 16-bit float umgewandelt. Gleichzeitig ist das die einzige Optimierung, die ich am Modell für die Prototyp-iOS Applikation vorgenommen habe.

8.1.1.2 Pruning

Beim Pruning werden im Modell Parameter entfernt, welche nur einen kleinen Einfluss auf die Vorhersagen haben. Dadurch optimierte Modelle können wesentlich effektiver komprimiert werden.

8.1.1.3 Clustering

Das Clustering gruppiert die Gewichte von den einzelnen Modell-Schichten in eine vordefinierte Anzahl von Cluster. Danach werden die Werte der Cluster-Schwerpunkte (Centroid) als Gewichte der einzelnen Schichten geteilt. Dies reduziert die Anzahl einmaliger (unique) Gewichtswerte und dadurch die Komplexität des Modells.

Nebst den optionalen Optimierungen ist es für die Integration eines CoreML Modells in eine iOS-Applikation (iOS 16.2) erforderlich, dass dem Modell ein Non Maximum Suppression «Layer» angefügt wird (vgl. Kapitel 4).

8.2 Applikations-Struktur

Die Prototyp iOS-Applikation (App) für die real-time Object Detection wurde basierend auf Beispiel-Code von Apple entwickelt (Apple Developer Documentation 2023b). Basierend auf Storyboards[11] nutzt die iOS App SwiftUI. Wie in Abbildung 25 gezeigt, verteilt sich die Grundstruktur der mobilen Applikation auf den Ordner «CoreMLModels» mit den trainierten und konvertierten CoreML-Modellen der einzelnen Experimente, die zwei ViewControllers und den DetectionHelper.

[11] Grafische Repräsentation von iOS User Interfaces

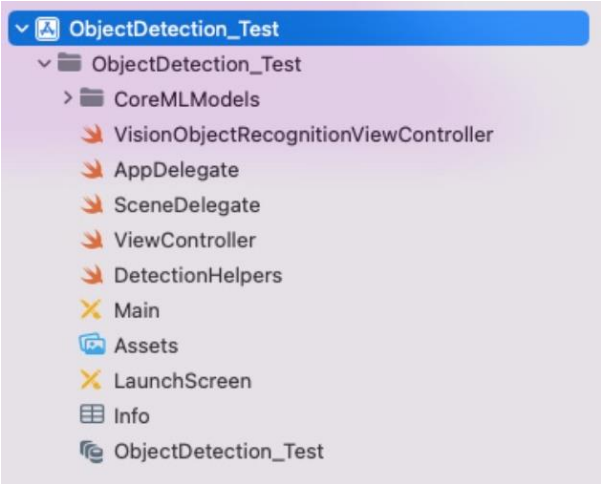


Abbildung 25: Struktur iOS Mobile Application

Wie die Abbildung 26 zeigt, implementiert der ViewController.swift die UIViewController-Klasse und bildet die Haupt-Sicht (previewView) der App. Zusätzlich wird in diesem Controller das Live-Bild der Rückkamera des Mobiltelefons gelesen und in der App ausgegeben. Die gesamte Inferenz mit den Aufrufen des CoreML-Modells und der Darstellung der Outputs (Object Detections mit Bounding Boxes) wird im VisionObjectRecognitionViewController.swift ausgeführt. Letzterer Controller erbt vom ViewController. Im DetectionHelpers.swift sind Hilfs-Funktionen implementiert, welche das Zeichnen und Schreiben von Bounding-Boxen bzw. Klassen-Labels mit den Confidence-Werten übernehmen.

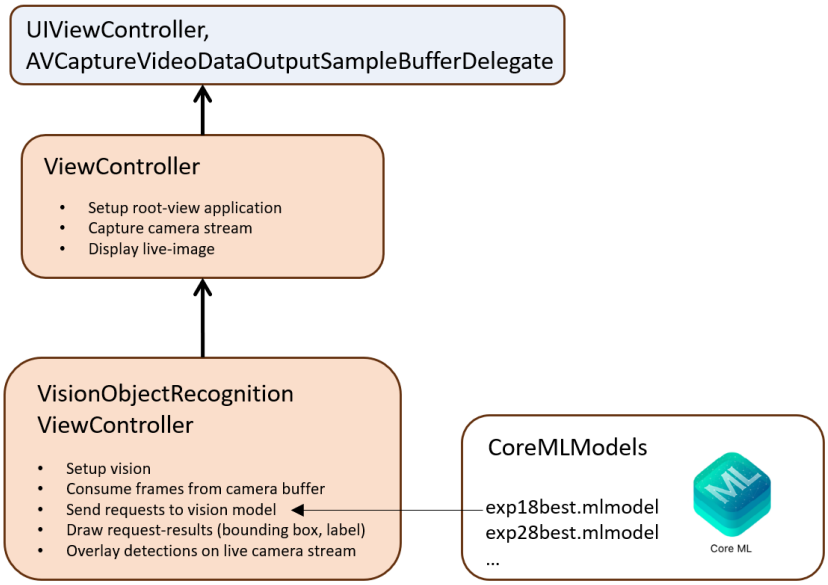


Abbildung 26: Klassen- und Modell-Struktur iOS Applikation

8.2.1 App Views und Layers

In Swift sind UIViews elementare Bauteile für das User Interface. Sie dienen dazu, Inhalte für rechteckige Bereiche auf dem Bildschirm zu organisieren. Die Views werden von Layer gestützt, die für das Rendering (Zeichnen, Animationen) zuständig sind. Diese können über das «.layer»-Property der View erreicht werden. Es können mehrere hierarchische Layers zu einer View hinzugefügt werden. Abbildung 27 zeigt die Struktur der View mit den unterschiedlichen Layers.

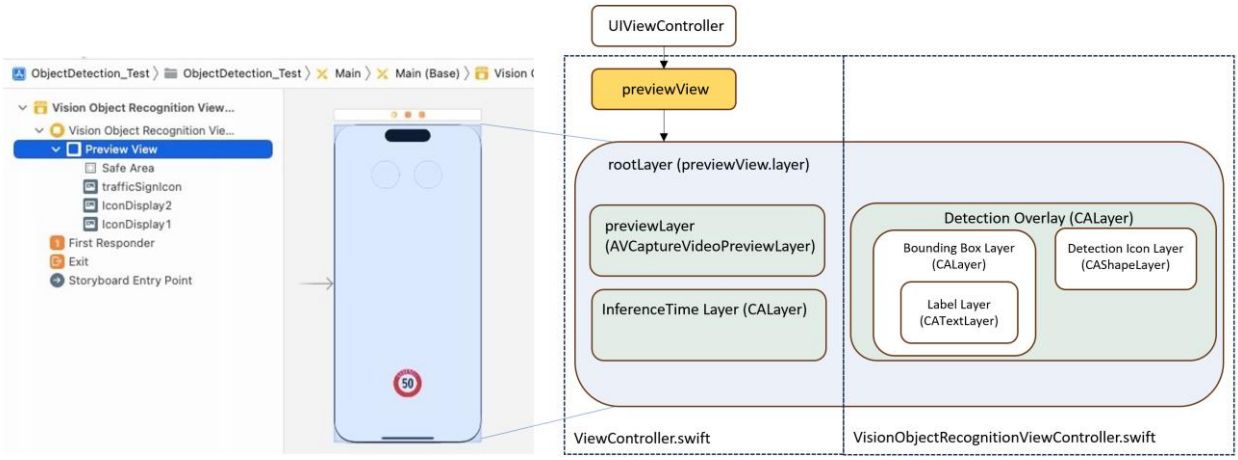


Abbildung 27: Views und Layers iOS Mobile Application

8.2.2 Kamerabild-Caption: ViewController

Wie bereits erwähnt, ist der ViewController dafür zuständig, dass das Live-Bild von der Kamera gelesen und in der App dargestellt wird. Zusätzlich müssen die «Raw Pixel-Werte» des Input-Bildes in einem sogenannten Pixel-Buffer gespeichert werden, damit das Modell später aus dem Buffer die einzelnen Frames lesen und die Inferenz darauf ausführen kann. Um eine real-time Aufnahme des Kamerabildes zu lesen, muss der ViewController eine Capture-Session instanzieren (AVCaptureSession Klasse) und dieser das Input-Device bzw. die Outputs für den Preview und den Pixel-Buffer zuweisen (vgl. Abbildung 28).

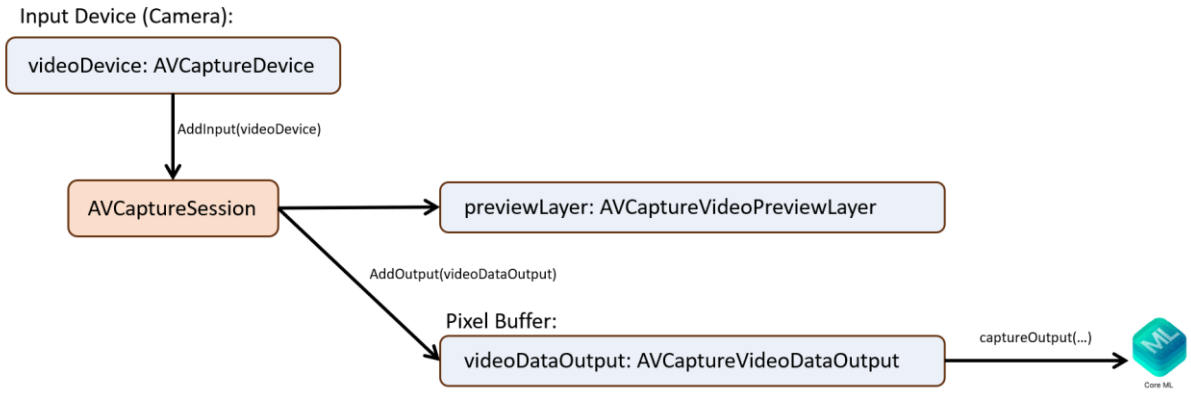


Abbildung 28: AVCaptureSession mit Input/Output

8.2.3 Real-time Inferenz: VisionObjectRecognitionViewController

In diesem Sub-ViewController erfolgt die eigentliche real-time Object Detection. Dazu lesen wir die Frames aus dem Pixel Buffer aus und geben diese für die Inferenz an das Modell weiter. Die Resultate des Modells bzw. die detektierten Objekte werden anschliessend mit Bounding-Boxen im Detection Layer visualisiert und auf den previewLayer mit dem Livekamerabild projiziert.

Wie der folgende Code-Ausschnitt zeigt, wird der VNImageRequestHandler genutzt, um die Bilder vom Buffer zu lesen. Das Frame wird dann über einen VNRequest mittels perform(self.request) an das CoreML-Modell übergeben.

Wichtig hierbei ist, dass beim verwendeten iPhone 13 mini der Sensor der Kamera und damit die Aufnahmen standardmässig im Landscape-Modus sind. Das Computer-Vision Modell erwartet den Input aber im Portrait-Modus. Aus diesem Grund muss das Bild mit «.right» nach rechts rotiert werden, bevor wir es an das Modell übergeben.

```
override func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer,
from connection: AVCaptureConnection) {
        guard let pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer) else {
            return
        }
        let exifOrientation = exifOrientationFromDeviceOrientation()
        let imageRequestHandler =
       VNImageRequestHandler(cvPixelBuffer: pixelBuffer, orientation: .right, options: [:])
        do {
            let start = CACurrentMediaTime()
            try imageRequestHandler.perform(self.requests)
            inferenceTime = (CACurrentMediaTime() - start)
        } catch {
            print(error)
        }
    }
```

Die eigentliche Inferenz wird bei CoreML mit einem VNCoreMLRequest ausgeführt. Damit die Object Detection auf den Frames asynchron erfolgt, wird hier eine DispatchQueue genutzt. Der VNCoreMLRequest returniert ein Objekt, das die Bounding-Box Koordinaten, Klassen-Labels und Confidence-Scores enthält. In der Funktion «drawVisionRequestResults» werden die Detektions-Informationen zunächst aus dem Objekt ausgelesen. Über den entsprechenden Layer werden die Bounding-Boxes bzw. Labels dann gezeichnet und im Detection Overlay visualisiert. Der Detection Overlay selbst wird anschliessend mit den richtigen Faktoren skaliert und auf den rootLayer projiziert. Zusätzlich werden im «Detection Icon Layer» für die vorhergesagten Klassen entsprechende Icons eingeblendet.

```
do {
        let visionModel = try VNCoreMLModel(for: MLModel(contentsOf: modelURL))
        let objectRecognition = VNCoreMLRequest(model: visionModel, completionHandler:
        {(request, error) in
                DispatchQueue.main.async(execute: {
                    // perform all the UI updates on the main queue
                    if let results = request.results {
                        self.drawVisionRequestResults(results)
                    }
                })
            })
            self.requests = [objectRecognition]
        }
```

Die entwickelte iOS-Applikation hat gezeigt, dass mit Computer-Vision Modellen (YOLOv5) für beide Datensätze, den «GTSRB»-Datensatz und den See And Avoid Small»-Datensatz, real-time Object Detection mit der Mobiltelefonkamera des iPhones durchgeführt werden kann. Je nach verwendeter Hardware ergaben sich aber deutliche Unterschiede in den Inferenz-Zeiten. Ab-

bildung 29 zeigt auf der linken Seite zwei Beispiele, die auf dem iPhone8 ausgeführt wurden und für die Inferenz über 200 Millisekunden benötigten. Bei den zwei Beispielen auf der rechten Seite kam das iPhone13 mini zum Einsatz, wobei mit rund 20 Millisekunden wesentlich kürzere Inferenz-Zeiten erreicht werden konnten. Damit konnten mit dieser Implementation die Input-Daten mit über 45 FPS verarbeitet werden. Die Geräte haben folgende unterschiedlichen Hardware-Spezifikationen:

- **iPhone 13 mini**
  - A15 Bionic Chip
  - 6-Core CPU mit 2 Performance-Kernen und 4-Effizienz Kernen
  - 4-Core GPU
  - 16-Core Neural Engine
- **iPhone 8**
  - A11 Bionic Chip
  - 6-Core CPU
  - 3-Core GPU
  - Mit Neural Engine

Abbildung 29: Real-time Inferenz iPhone8 und iPhone13 mini

9. Fazit und Ausblick

9.1 Fazit/Ergebnisse

Ziel der vorliegenden Master-Thesis war es, mit YOLOv5- und DETR-Modellen real-time Object Detection auf Bildern und Videos auszuführen. Dabei sollten zwei verschiedene Datensätze für die Detektion von Strassenverkehrsschildern und Luftfahrzeugen (Helikopter) genutzt werden. Zudem sollte im Rahmen eines Prototyps eine App für Mobiltelefone entwickelt werden, um die Möglichkeit der real-time Object Detection unter der Nutzung der Geräte-Kamera auf mobilen Endgeräten aufzeigen zu können.

In einem ersten Schritt wurden verschiedene Modelle basierend auf der YOLOv5-Architektur und dem GTSRB Datensatz für die Detektion von Strassenschildern trainiert. Es konnte schliesslich ein finales YOLO-Modell trainiert werden, dass eine hohe Genauigkeit (mAP) und auf Standbildern eine gute Leistung in der Erkennung von über 40 verschiedenen Strassenschild-Klassen erreicht. Trotz des Einsatzes von über 10% Background-Images war die Inferenz auf Videos aber nicht gleichermassen zufriedenstellend. Viele «False Positives» und damit Objekte, die lediglich in der Form den Strassenschildern ähnlich waren, wurden detektiert. In weiteren Schritten wurde beim Training mit dem «See And Avoid» Datensatz festgestellt, dass die Qualität der Bounding-Box Annotationen bei den Open Image v7 Bilder in den entsprechenden Kategorien «Helicopter», «Airplane» und «Balloon» oft relativ schlecht ist. So waren viele Bounding-Boxen nicht akkurat definiert, falsch klassifiziert oder fehlten gänzlich. Aus diesem Grund wurde der Datensatz auf die Klasse Helikopter reduziert, alle dazugehörigen Annotationen vollständig kontrolliert, nötigenfalls korrigiert und als neuer Datensatz «See And Avoid Small» definiert. Nach dieser Überarbeitung und dem erneuten Einsatz von vielen Background-Images konnte auch für die Detektion von Helikoptern ein YOLOv5-Modell (small) mit hoher Genauigkeit aufgebaut werden - sowohl die Inferenz auf Bildern wie auch auf Videos war schnell und akkurat.

In einem zweiten Teil der Arbeit wurde eine iOS-Applikation (iOS 16.2) für das iPhone entwickelt, welche die Nutzung wechselnder CoreML-Modelle für die Detektion von Verkehrsschildern und auch Helikoptern zulässt. Die App zeigt ein Live-Bild der Rückkamera des Mobiltelefons, projiziert bei der Detektion eines Objektes in Echtzeit eine entsprechende Bounding-Box auf das Kamera-Bild und markiert damit alle einzelnen Detektionen separat. Zusätzlich zeigt die Applikation ein Icon, das dem detektierten Objekt entspricht und gibt die entsprechende Inferenz-Zeit auf dem Bildschirm aus (vgl. Abbildung 29). Während der Entwicklung dieser App hat sich gezeigt, wie wichtig es ist, dass alle Versionen, wie die des MacOS auf dem Entwicklungs-Computer, des iOS auf dem iPhone und der integrierten Entwicklungsumgebung (XCode) mit den entsprechenden Zielplattformen sauber aufeinander abgestimmt sind.

Mit der Prototyp-App konnte zudem aufgezeigt werden, dass real-time Object Detection insbesondere mit YOLO-Modellen auch auf handelsüblichen Mobiltelefonen möglich ist. Allerdings müssen die Mobiltelefone über entsprechend dimensionierte Hardware (CPU, GPU, etc.) verfügen. Schon beim Einsatz auf Modellen wie dem iPhone 8 nimmt die Inferenz-Zeit massiv zu. Solche Geräte erhitzen unter entsprechender Beanspruchung auch deutlich schneller, was zu verringerten Einsatzmöglichkeiten der App führt.

Im letzten Schritt dieser Arbeit wurde ein DETR-Modell entwickelt und ebenfalls auf dem «See And Avoid Small» Datensatz trainiert. Auch hier konnte ein Modell mit hoher Genauigkeit aufgebaut werden. Die Inferenz war zwar deutlich langsamer als mit dem YOLOv5-Modell, erreichte auf Standbildern aber eine vergleichbare Leistung. Das DETR-Modell wurde nicht explizit auf bewegten Bildern geprüft und nicht für den Einsatz auf dem Mobiltelefon adaptiert.

Abschliessend wurden die zwei Modell-Architekturen und deren Leistung in der problemspezifischen Object Detection miteinander verglichen. Die Ergebnisse zeigen deutlich, dass beide unterschiedlichen Architekturen sowohl für die Detektion von Verkehrsschildern, wie auch von Luftfahrzeugen im 3-dimensionalen Raum eingesetzt werden können.

9.2 Kritische Diskussion

Der «See And Avoid Small» Datensatz ist mit den rund 1'900 Bildern relativ klein. Sicher sinnvoll wäre die Prüfung, ob die Modell-Genauigkeit insbesondere bei der Inferenz auf dem Mobiltele-

fon mit einem grösseren Datensatz nochmals deutlich verbessert werden könnte. Auch ein Datensatz mit allen drei Kategorien (Helicopter, Airplane und Balloon) und einer besseren Qualität bei den Annotationen dürfte zu einer wesentlichen Verbesserung beitragen. Da bei der Detektion von Luftfahrzeugen Objekte von allen Seiten erkannt werden müssen, dürfte eine grössere Variation des Aufnahmewinkels bei den Trainings-Bilder zusätzlich zu einem besseren Modell für den Problembereich «See And Avoid» führen.

Beim GTSRB-Datensatz mit den rund 50'000 Bildern ist die Menge zwar ausreichend gross, die Bilder aber zeigen jeweils nur den Ausschnitt des Verkehrsschildes und kaum Hintergrund bzw. Umgebung. Auch die Auflösung der Bilder ist mit maximal 250x250 Pixel relativ klein. Zu prüfen wäre hier, ob die zwei Architekturen YOLO und DETR mit grösseren Bildern von Verkehrsschildern und Umgebung bessere Resultate liefern könnten.

Eine rasante Entwicklung zeigt sich insbesondere bei Transformer-basierten Ansätzen oder bei der Adaption bekannter Modelle für den Einsatz in der Computer Vision oder der real-time Object Detection. So könnte beispielsweise die Schwäche der beiden eingesetzten Modelle bei der Detektion von kleinen und zusammenliegenden Objekten in neueren oder optimierten Versionen weniger gross sein. YOLOv8 soll z.B. aufgrund angepasster Convolutional Blocks, eines neuen Detektion-Systems ohne Anker-Boxen und Veränderungen an den Augmentation zur Trainingszeit nochmals höhere Genauigkeit bringen (Solawetz 2023).

Die Defizite von DETR bei der Objekt Detektion unter real-time Bedingungen wurden in Adaptionen oder Weiterentwicklungen von DETR wie z.B. RT-DETR (Lv et al. 2023) oder Deformable-DETR weiter vermindert (Zhu et al. 2020).

9.3 Ausblick

Wie im vorangehenden Kapitel beleuchtet, gibt es auf der Seite von YOLO verschiedene, weiterentwickelte Versionen. Die letzte Version YOLOv8 wurde im Januar 2023 von Ultralytics, den Entwicklern der Version 5, herausgegeben. Auf der Seite der Detection Transformers gibt es zudem einige Adaptionen der beschriebenen Architektur, die insbesondere für den Einsatz bei real-time Object Detection-Aufgaben entwickelt wurden. In nächsten Schritten soll untersucht werden, inwiefern solche neueren Architekturen und Modelle wie YOLOv8, RT-DETR oder Deformable DETR eine bessere Leistung bei der Detektion von Strassenverkehrsschildern und Luftfahrzeugen unter real-time Bedingungen erbringen können. Dabei soll der Fokus darauf liegen, die Detektion auch von kleinen und eng beieinander liegenden Objekten zu optimieren. Die Erkennung von kleinen Objekten ist in der problemspezifischen Anwendung von grosser Bedeutung, da Strassenschilder oder auch Luftfahrzeuge so möglichst frühzeitig und bereits aus grösserer Entfernung erkannt werden könnten. Ein weiterer Aspekt ist die Möglichkeit des Betriebes der Computer Vision Modelle auf mobilen Endgeräten. Auch die, im Rahmen dieser Arbeit trainierten DETR-Modelle sollen in künftigen Schritten konvertiert und die Inferenz-Leistung auf Mobiltelefonen geprüft werden.

Die mobile Applikation soll ebenfalls weiterentwickelt werden und Verbesserungen insbesondere bei der Darstellung von parallelen Detektionen verschiedener Objekt-Klassen bringen. Zusätzlich sollen die erfolgten Detektionen in einer Historie gespeichert und die Aktuellsten in einer Art «Rolling-Window» angezeigt werden. Aktuell funktioniert der Prototyp nur im Portrait-Modus. Hier soll die Kamerabild-Vorschau und Projektion der Bounding-Boxen dahingehend weiterentwickelt werden, dass die Detektion und Darstellung mit der richtigen Skalierung auch während und nach einer Drehung des Mobiltelefons funktionieren.

10. Literaturverzeichnis

Apple Developer Documentation (2023a): Core ML | Apple Developer Documentation. Online verfügbar unter https://developer.apple.com/documentation/coreml, zuletzt aktualisiert am 27.06.2023, zuletzt geprüft am 27.06.2023.

Apple Developer Documentation (2023b): Recognizing Objects in Live Capture | Apple Developer Documentation. Online verfügbar unter https://developer.apple.com/documentation/vision/recognizing_objects_in_live_capture, zuletzt aktualisiert am 27.06.2023, zuletzt geprüft am 27.06.2023.

Apple Developer Documentation (2023c): Vision | Apple Developer Documentation. Online verfügbar unter https://developer.apple.com/documentation/vision, zuletzt aktualisiert am 27.06.2023, zuletzt geprüft am 27.06.2023.

BAZL, Bundesamt für Zivilluftfahrt (2023): Vermeidung von Annäherungen (Airprox) unter Sichtflugregeln in den Lufträumen G und E. Online verfügbar unter https://www.bazl.admin.ch/bazl/de/home/themen/sicherheit/sicherheits-und-risikomanagement/safety-promotion/empfehlungen--sand-/vermeidung-von-annaeherungen--airprox--unter-sichtflugregeln-in-.html, zuletzt aktualisiert am 12.03.2023, zuletzt geprüft am 12.03.2023.

Bochkovskiy, Alexey; Wang, Chien-Yao; Liao, Hong-Yuan Mark (2020): YOLOv4: Optimal Speed and Accuracy of Object Detection. Online verfügbar unter https://arxiv.org/pdf/2004.10934.

Buslaev, Alexander; Iglovikov, Vladimir I.; Khvedchenya, Eugene; Parinov, Alex; Druzhinin, Mikhail; Kalinin, Alexandr A. (2020): Albumentations: Fast and Flexible Image Augmentations. In: *Information* 11 (2), S. 125. DOI: 10.3390/info11020125.

Carion, Nicolas; Massa, Francisco; Synnaeve, Gabriel; Usunier, Nicolas; Kirillov, Alexander; Zagoruyko, Sergey (2020): End-to-End Object Detection with Transformers. Online verfügbar unter https://arxiv.org/pdf/2005.12872.

dr.jur. Müller, Roland (2014): Recht der Luftfahrt.

dshahid380 (2019): Convolutional Neural Network - Towards Data Science. In: *Towards Data Science*, 24.02.2019. Online verfügbar unter https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529, zuletzt geprüft am 14.06.2023.

Dumoulin, Vincent; Visin, Francesco (2016): A guide to convolution arithmetic for deep learning. Online verfügbar unter https://arxiv.org/pdf/1603.07285.

Erik Lindernoren (2023): Minimal PyTorch implementation of YOLOv3. Online verfügbar unter https://github.com/eriklindernoren/PyTorch-YOLOv3, zuletzt aktualisiert am 29.05.2023, zuletzt geprüft am 29.05.2023.

German Traffic Sign Benchmarks (2020). Online verfügbar unter https://benchmark.ini.rub.de/gtsrb_news.html, zuletzt aktualisiert am 11.12.2020, zuletzt geprüft am 09.06.2023.

He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2014): Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. Online verfügbar unter https://arxiv.org/pdf/1406.4729.

Henderson, Paul; Ferrari, Vittorio (2016): End-to-end training of object class detectors for mean average precision. Online verfügbar unter https://arxiv.org/pdf/1607.03476.

Huang, Gao; Liu, Zhuang; van der Maaten, Laurens; Weinberger, Kilian Q. (2016): Densely Connected Convolutional Networks. Online verfügbar unter https://arxiv.org/pdf/1608.06993.

Introduction to ONNX - ONNX 1.15.0 documentation (2023). Online verfügbar unter https://onnx.ai/onnx/intro/, zuletzt aktualisiert am 27.06.2023, zuletzt geprüft am 27.06.2023.

Klette, Reinhard (2014): Concise Computer Vision. An Introduction into Theory and Algorithms. London: Springer (Springer eBook Collection Computer Science).

Lin, Tsung-Yi; Dollár, Piotr; Girshick, Ross; He, Kaiming; Hariharan, Bharath; Belongie, Serge (2016): Feature Pyramid Networks for Object Detection. Online verfügbar unter https://arxiv.org/pdf/1612.03144.

Lv, Wenyu; Xu, Shangliang; Zhao, Yian; Wang, Guanzhong; Wei, Jinman; Cui, Cheng et al. (2023): DETRs Beat YOLOs on Real-time Object Detection. Online verfügbar unter https://arxiv.org/pdf/2304.08069.

PyTorch (2023). Online verfügbar unter https://pytorch.org/, zuletzt aktualisiert am 27.06.2023, zuletzt geprüft am 27.06.2023.

Redmon, Joseph; Divvala, Santosh; Girshick, Ross; Farhadi, Ali (2015): You Only Look Once: Unified, Real-Time Object Detection. Online verfügbar unter https://arxiv.org/pdf/1506.02640.

Redmon, Joseph; Farhadi, Ali (2016): YOLO9000: Better, Faster, Stronger. Online verfügbar unter https://arxiv.org/pdf/1612.08242.

Redmon, Joseph; Farhadi, Ali (2018): YOLOv3: An Incremental Improvement. Online verfügbar unter https://arxiv.org/pdf/1804.02767.

Rezatofighi, Hamid; Tsoi, Nathan; Gwak, JunYoung; Sadeghian, Amir; Reid, Ian; Savarese, Silvio (2019): Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression. Online verfügbar unter https://arxiv.org/pdf/1902.09630.

Solawetz, Jacob (2020a): Mean Average Precision (mAP) in Object Detection. In: *Roboflow Blog*, 06.05.2020. Online verfügbar unter https://blog.roboflow.com/mean-average-precision/, zuletzt geprüft am 26.06.2023.

Solawetz, Jacob (2020b): What is YOLOv5? A Guide for Beginners. In: *Roboflow Blog*, 29.06.2020. Online verfügbar unter https://blog.roboflow.com/yolov5-improvements-and-evaluation/, zuletzt geprüft am 14.03.2023.

Solawetz, Jacob (2023): What is YOLOv8? The Ultimate Guide. In: *Roboflow Blog*, 11.01.2023. Online verfügbar unter https://blog.roboflow.com/whats-new-in-yolov8/, zuletzt geprüft am 22.06.2023.

Tan, Mingxing; Pang, Ruoming; Le V, Quoc (2020): EfficientDet: Scalable and Efficient Object Detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (*Online verfügbar unter https://arxiv.org/pdf/1911.09070.

TensorFlow: Model optimization. Online verfügbar unter https://www.tensorflow.org/lite/performance/model_optimization, zuletzt geprüft am 17.06.2023.

ultralytics/yolov5 (2023): Git Repository ultralytics/yolov5. Online verfügbar unter https://github.com/ultralytics/yolov5?ref=blog.roboflow.com, zuletzt aktualisiert am 29.05.2023, zuletzt geprüft am 29.05.2023.

ultralytics/yolov5 Focus Layer (2023): YOLOv5 Focus Layer · ultralytics/yolov5 · Discussion #3181. Online verfügbar unter https://github.com/ultralytics/yolov5/discussions/3181m1, zuletzt aktualisiert am 28.05.2023, zuletzt geprüft am 28.05.2023.

Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N. et al. (2017): Attention Is All You Need. Online verfügbar unter https://arxiv.org/pdf/1706.03762.

Wang, Chien-Yao; Bochkovskiy, Alexey; Liao, Hong-Yuan Mark (2022): YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. Online verfügbar unter https://arxiv.org/pdf/2207.02696.

Wang, Chien-Yao; Liao, Hong-Yuan Mark; Yeh, I-Hau; Wu, Yueh-Hua; Chen, Ping-Yang; Hsieh, Jun-Wei (2019): CSPNet: A New Backbone that can Enhance Learning Capability of CNN. Online verfügbar unter https://arxiv.org/pdf/1911.11929.

Wilson, Joshua (2023): Using Bipartite Matching and Detection Transformers for Document Layout Analysis Transfer Learning.

Yang, Suorong; Xiao, Weikang; Zhang, Mengcheng; Guo, Suhan; Zhao, Jian; Shen, Furao (2022): Image Data Augmentation for Deep Learning: A Survey. Online verfügbar unter https://arxiv.org/pdf/2204.08610.

Zhu, Xizhou; Su, Weijie; Lu, Lewei; Li, Bin; Wang, Xiaogang; Dai, Jifeng (2020): Deformable DETR: Deformable Transformers for End-to-End Object Detection. Online verfügbar unter https://arxiv.org/pdf/2010.04159.

11. Selbständigkeitserklärung

Mit der Abgabe dieser Abschlussarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat (Bei Teamarbeiten gelten die Leistungen der übrigen Teammitglieder nicht als fremde Hilfe).

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Abschlussarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind

Ort, Datum: Schönenberg ZH, 29.06.2023

Unterschrift Studierende/r:

Sven Goldinger

12. Anhang

A. Verkehrsschild-Klassen im GTSRB Datensatz 45

B. Confusion-Matrix Experiment «Run35» YOLO 46

C. Vergleich Confusion-Matrix YOLO Experimente (False Positives) 46

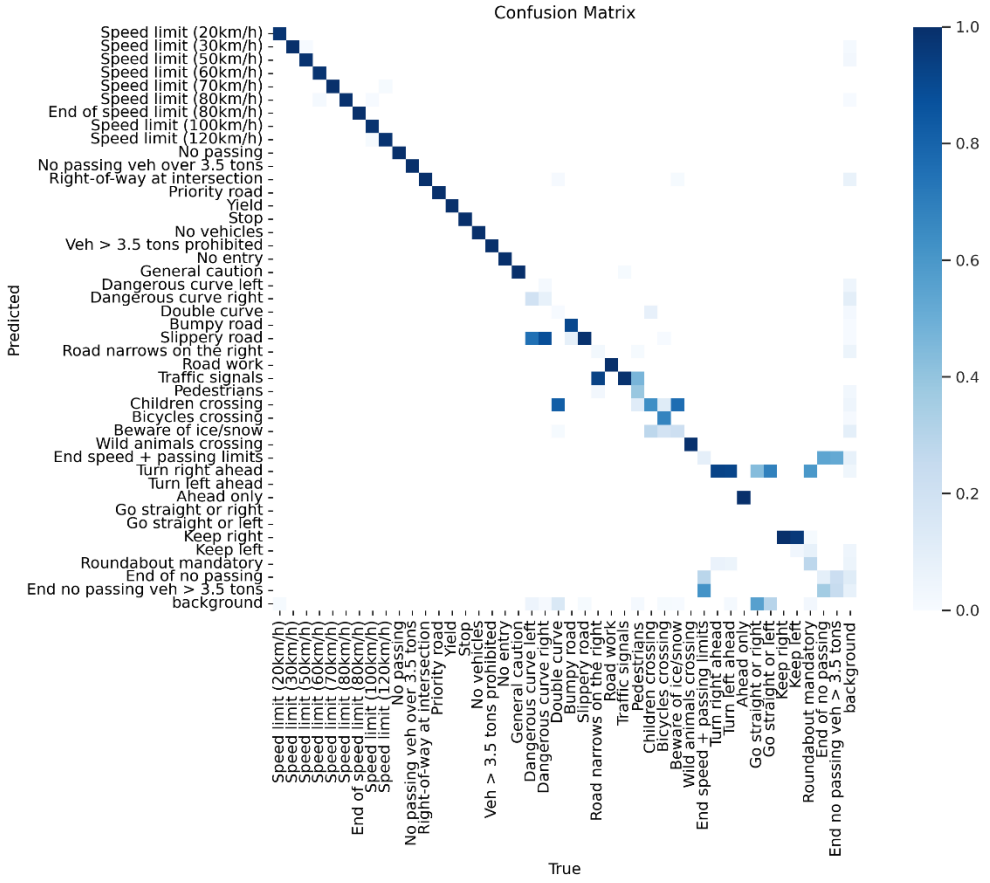
D. Detail-Architektur Transformer in DETR 47

E. EfficientDet Architektur mit Feature Pyramid Network (FPN) 47

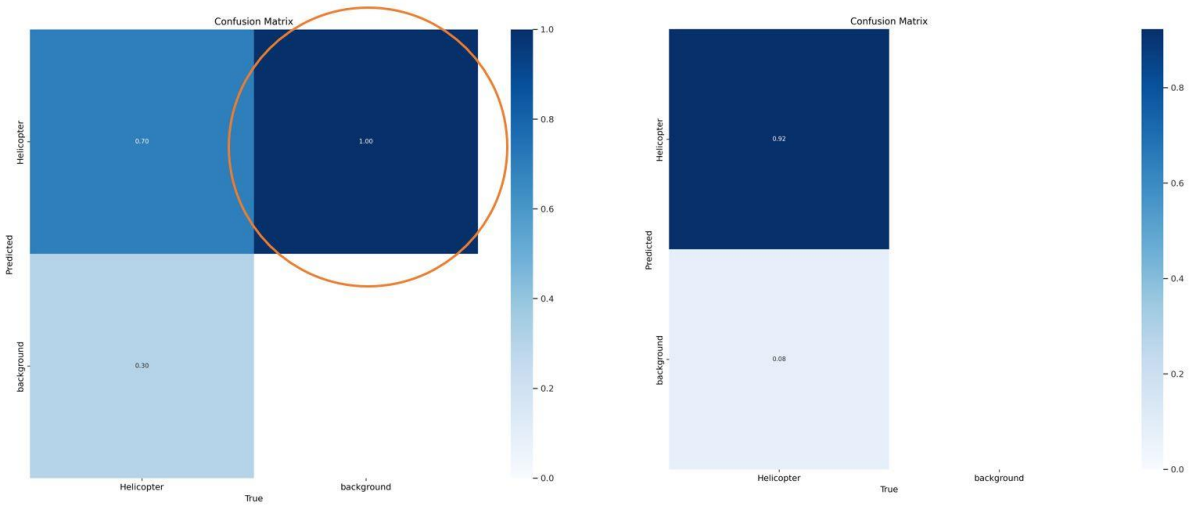
A. Verkehrsschild-Klassen im GTSRB Datensatz

```
# Classes
names:
  0: Speed limit (20km/h)
  1: Speed limit (30km/h)
  2: Speed limit (50km/h)
  3: Speed limit (60km/h)
  4: Speed limit (70km/h)
  5: Speed limit (80km/h)
  6: End of speed limit (80km/h)
  7: Speed limit (100km/h)
  8: Speed limit (120km/h)
  9: No passing
  10: No passing veh over 3.5 tons
  11: Right-of-way at intersection
  12: Priority road
  13: Yield
  14: Stop
  15: No vehicles
  16: Veh > 3.5 tons prohibited
  17: No entry
  18: General caution
  19: Dangerous curve left
  20: Dangerous curve right
  21: Double curve
  22: Bumpy road
  23: Slippery road
  24: Road narrows on the right
  25: Road work
  26: Traffic signals
  27: Pedestrians
  28: Children crossing
  29: Bicycles crossing
  30: Beware of ice/snow
  31: Wild animals crossing
  32: End speed + passing limits
  33: Turn right ahead
  34: Turn left ahead
  35: Ahead only
  36: Go straight or right
  37: Go straight or left
  38: Keep right
  39: Keep left
  40: Roundabout mandatory
  41: End of no passing
  42: End no passing veh > 3.5 tons
```

B. Confusion-Matrix Experiment «Run35» YOLO

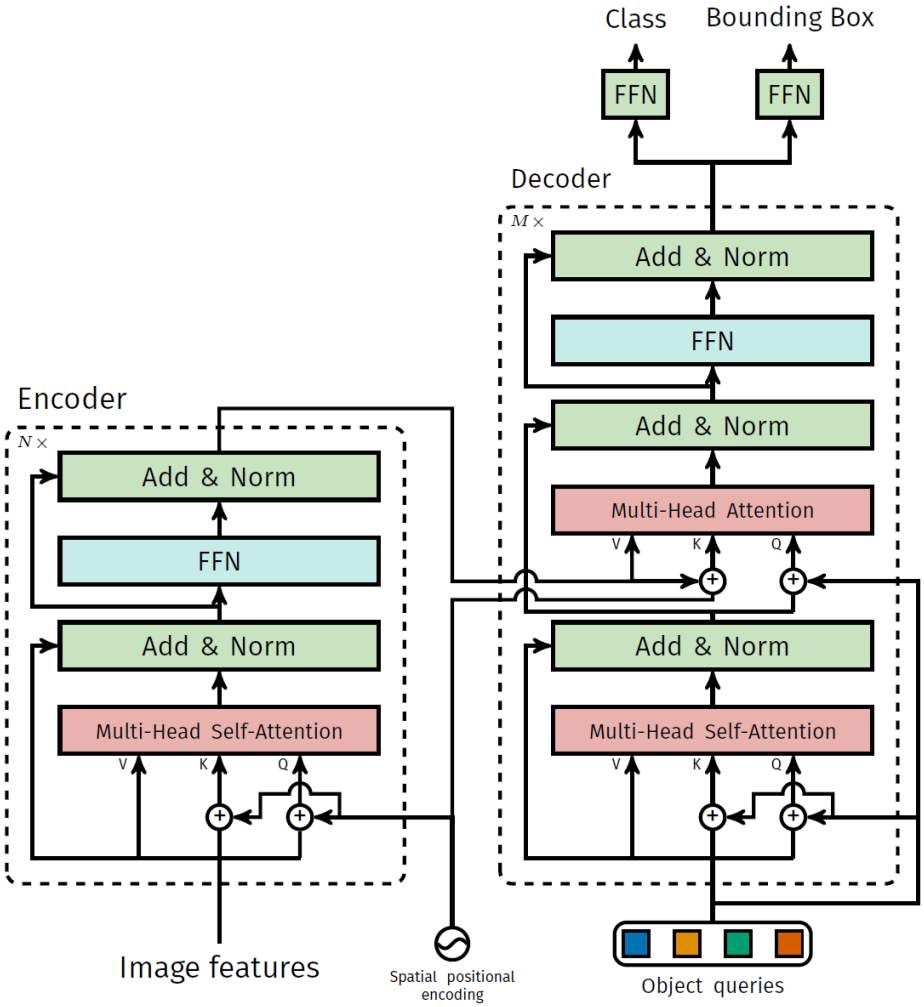


C. Vergleich Confusion-Matrix YOLO Experimente (False Positives)



YOLO exp13 YOLO exp59

D. Detail-Architektur Transformer in DETR



E. EfficientDet Architektur mit Feature Pyramid Network (FPN)