



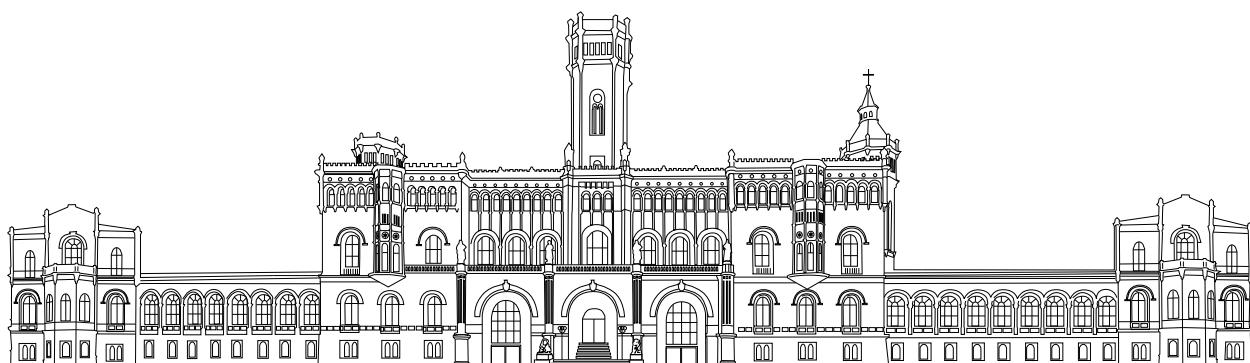
Fakultät für Elektrotechnik und Informatik

Computational Health Informatics

Optimierung von Neuronalen Netzen für die Analyse des Pupillenreflexes als mo- bile Diagnoseunterstützung

Masterarbeit im Studiengang Informatik (M.Sc.),
eingereicht von DANIEL KAMPHAKE am 22.11.2021

Erstprüferin: Prof. Dr.-Ing. Gabriele von Voigt
Zweitprüfer: Prof. Dr. Michael Rohs
Betreuer: Marcel Schepelmann
Matrikelnummer: 3217480



Erklärung der Selbständigkeit

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind, und die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.


Daniel Kamphake

Hannover, den 22.11.2021

Erklärung zur Plagiatsprüfung

Mit der Übermittlung meiner Arbeit auch an externe Dienste zur Plagiatsprüfung durch Plagiatssoftware erkläre ich mich einverstanden.


Daniel Kamphake

Hannover, den 22.11.2021

Abstract

Neural networks have led to breakthroughs in data analysis. They can autonomously infer patterns from training sets to process features of unknown data to solve arbitrary tasks. Especially in object recognition, there have been advances in processing images with architectures such as RCNN, SSD, and YOLO. These advances can be used in the medical setting for eye, iris, and pupil recognition. Previous work has already been done with the goal of creating an app for diagnosing craniocerebral trauma. This work follows up on the results and sets an optimization of the neural network as a goal.

As a result, the UnityEyes dataset is presented, which was created due to the poor availability of other data and is intended to serve as a robust base for training. We examine the impact of various hyperparameters and particularly highlights changes in the size of the model. Estimates of performance efficiencies with respect to the usage in the app are discussed, and finally an updated version of the previously presented app is presented.

While the largest models achieve accuracies of 97.8 % mAP.95, the optimized, small version is able to overcome the 98 % mAP.95 hurdle. It does this while relying on only one fiftieth of the parameters and power requirements. Even smaller models examined and reduce the requirements even further, but decrease in accuracy.

Zusammenfassung

Neuronale Netze haben zu Durchbrüchen in der Datenanalyse geführt. Sie können aus Datensätzen autonom Muster ableiten, um so die Merkmale unbekannter Daten zu verarbeiten und beliebige Aufgaben zu lösen. Besonders in der Objekterkennung gab es mit Architekturen wie dem RCNN, SSD und YOLO Fortschritte in der Verarbeitung von Bildern. Diese Fortschritte können im medizinischen Umfeld für die Erkennung von Augen, Iris und Pupille verwendet werden. Mit dem Ziel eine App für die Diagnose eines Schädel-Hirn-Traumas zu erstellen wurde bereits eine Arbeit angefertigt. Diese Arbeit knüpft an den Ergebnissen an und setzt eine Optimierung des neuronalen Netzes als Ziel.

Als Ergebnis wird der UnityEyes Datensatz vorgestellt, der aufgrund der schlechten Verfügbarkeit anderer Daten entstanden ist und als robuste Trainingsgrundlage dienen soll. Die Optimierung von neuronalen Netzen untersucht den Einfluss verschiedener Hyperparameter und hebt besonders die Änderungen in der Größe des Modells hervor. Es werden Abschätzungen zu Leistungseffizienz hinsichtlich des Einsatzes in der App erörtert und abschließend eine aktualisierte Version der bereits vorgestellten App präsentiert.

Während die größten Modelle Genauigkeiten von 97.8 % mAP.95 erreichen, gelangt die optimiert, kleine Version über die 98 % mAP.95 Hürde. Sie schafft dies mit nur einem Fünfzigstel der Parameter und Leistungsanforderung. Noch kleinere Modelle werden betrachtet und reduzieren die Anforderungen noch weiter, sinken allerdings in der Genauigkeit.

Inhaltsverzeichnis

Zusammenfassung	iv
1 Einleitung	1
1.1 Motivation	1
1.2 Aufbau dieser Arbeit	3
2 Verwandte Arbeiten	5
2.1 Zugrundeliegende Arbeit	6
2.2 Bildsegmentierung	7
2.3 Gaze Estimation	7
2.4 Automated Machine Learning	8
2.5 Soziale Verantwortung	8
3 Grundlagen	11
3.1 Das menschliche Auge	12
3.1.1 Aufbau des Auges	12
3.1.2 Der Pupillenreflex	13
3.1.3 Schädel-Hirn-Trauma	14
3.2 Android	16
3.2.1 NNAPI	17
3.3 Neuronale Netze	19
3.3.1 Convolutional Neural Networks	20
3.3.2 Lernmethoden	21
3.3.3 Ensemblestrukturen	22
3.3.4 Transfer Learning	22
3.4 Deep Neural Networks	23
3.5 Objekterkennung mit neuronalen Netzen	25
3.5.1 Region-based Convolutional Network	25
3.5.2 Single Shot Detector	26
3.5.3 Non-Maximum Suppression	27
3.6 Die YOLO Architektur	27
3.6.1 Ultralytics YOLOv5	30

4 Methodik	31
4.1 Datensätze	32
4.1.1 Datensatz-Recherche	33
4.1.2 Eigene Vorverarbeitung mit Unity-Eyes	35
4.2 Metriken	36
4.2.1 Klassifizierung	36
4.2.2 Precision und Recall	36
4.2.3 Intersection over Union	37
4.2.4 Mean Average Precision	38
4.2.5 Konfidenz durch Wiederholung	38
4.3 Hyperparameter	39
4.3.1 Architektur	39
4.3.2 Bildaugmentationen	40
4.3.3 Trainingsfunktionen	40
4.3.4 Trainings-Batches	41
4.3.5 Quantisierung und Pruning	41
4.4 Hyperparameteroptimierung	42
4.4.1 Raster- und zufällige Suche	42
4.4.2 Bayes'sche Suche	43
4.5 Tools	45
4.5.1 Weights and Biases	45
4.5.2 Roboflow	45
4.5.3 Technisches Setup	46
5 Auswertung bestehender Daten	47
5.1 Diskussion vorliegender Ergebnisse	47
5.2 Eigenschaften vorliegender Datensätze	49
5.2.1 Datensatz: Augen	49
5.2.2 Datensatz: Iris und Pupillen	51
5.2.3 Bewertung	53
5.3 Eigener Datensatz	53
6 Experimente	59
6.1 Einordnung der YOLO Modell	60
6.2 Hyperparameteroptimierung	61
6.2.1 Erste Iteration: Random Search	62
6.2.2 Zweite Iteration: Bayes'sche Suche	64
6.2.3 Analyse der Batch Size	66
6.3 Finales Modell	68
6.4 Schrumpfen der Architektur	70
6.4.1 Aufbau der Modelle	71
6.4.2 Trainingsverhalten	71

7 Android Anwendung	73
7.1 Änderungen der Implementation	74
7.1.1 Kotlin	74
7.1.2 Änderungen in der Kameraimplementierung	75
7.2 PyTorch Mobile	75
7.2.1 Export	75
7.2.2 Benchmark	76
8 Schluss	77
8.1 Fazit	77
8.2 Ausblick	79
A Anhang	81
A.1 Daten zum UnityEyes Datensatz	81
A.2 Erklärung der Hyperparameter	81
A.2.1 Trainingsparameter	81
A.2.2 Eingabeparameter	82
A.3 Hyperparameteroptimierung	82
A.3.1 Zufällige Suche	82
A.3.2 Bayes'sche Suche	82

Abbildungsverzeichnis

3.1	Schemenhafter Aufbau eines Auges links [G5]. Markierung der Pupille und Iris auf einem echten Bild rechts. (geändert nach [DATA11])	13
3.2	Der Pupillenreflex. Links ist eine Pupille im Ausgangszustand zu sehen, mittig die zusammengezogene Pupille nach einem Lichtimpuls. (geändert nach [EYE9]). Rechts: Qualitativ eingezeichneter Verlauf der Pupillengröße.	14
3.3	Aufbau des Sehapparats mit Fokus auf die Steuerung der Pupillengröße [EYE14].	15
3.4	Die Systemarchitektur der Android NNAPI [25].	18
3.5	Das LeNet-5 von <i>LeCun et al.</i> [NN12] stellt das erste CNN dar. Es zeigt den Aufbau aus Convolutions, Subsampling und Dense Layern.	20
3.6	Ein einzelner ResNet-Block [NN29], Inception-Modul [NN27] und eine Verbindung aus beiden, dem Inception-ResNet [NN39]. (von links nach rechts)	24
3.7	Architektur des SSD Modells [NN52].	27
3.8	Erhaltene Featuremaps aus dem CNN Feature Extractor werden mittels definierter Anchors in jeder Rasterzelle analysiert [NN41].	28
3.9	Links: Lokale Bindung der Bounding Box an den Anchor durch Training des Offsets zu dem Mittelpunkt des Anchors [NN38]. Rechts: Aufteilung der Featuremaps in unterschiedliche Größen, um so Objekte unterschiedlicher Größen besser aufzubereiten [NN41].	29
4.1	Beispiele aus dem Syntheyes Datensatz [DATA4].	32
4.2	Beispiele von links nach rechts aus den Datensätzen von <i>Fusek</i> [DATA6], <i>Fuhl et al.</i> [DATA7] und <i>Sokolova et al.</i> [DATA10].	33
4.3	Beispiele von links nach rechts aus den Datensätzen von <i>Luo et al.</i> [DATA9] und <i>Wu et al.</i> [G7].	34
4.4	Das Interface von Unity-Eyes [DATA5] und eine Auswahl von Bildern, die von dem Tool generiert wurden.	35
4.5	Visualisierung von Precision und Recall innerhalb einer Menge von Klassifizierungen. (geändert nach [10])	36
4.6	Die IoU von verschiedenen inferierten Bounding Boxen in Relation zur Ground Truth [NN69].	38
4.7	Verschiedene Augmentationen, die mit dem YOLOv5 Framework auf den Unity Datensatz angewandt wurden.	40

4.8	Quantisierung von Float32, Float16 und Int8 und eine Anwendung vom Pruning mit 50 % und 70 %.	41
4.9	Vergleich von Raster und zufälliger Suche mit zwei Parametern [HPO7].	43
4.10	Links: Exploration und Exploitation über die Erwartungswerte eines Modells in einer Bayes'schen Suche [HPO8]. Rechts: Aufspannen des Problems in zwei Dimensionen [HPO10].	44
5.1	Trainingsverläufe von <i>Moryäner</i> [26] für (links) den Augendatensatz und (rechts) den Pupillendatensatz.	48
5.2	Precision, Recall und mAP.95 für das YOLO Small und Xtra auf dem bestehenden Augen-Datensatz über 50 Epochen.	50
5.3	Verteilung und Korrelation der Höhe und Breite der Bounding Box für den Augen-Datensatz links; Probleme mit den Ground Truth Daten rechts. (geändert nach [DATA11])	51
5.4	Precision, Recall und mAP.95 für das YOLO Small und Xtra auf dem bestehenden Iris-Datensatz über 200 Epochen.	52
5.5	Precision, Recall und mAP.95 für das YOLO Small und Xtra auf dem bestehenden Pupillen-Datensatz über 200 Epochen	53
5.6	Verteilung und Korrelation der Höhe und Breite der Bounding Box für den Iris- und Pupillen-Datensatz.	54
5.7	Der generierte Datensatz vor und nach dem Zuschneiden. Markiert sind links die Annotationen für die Iris und die 50 % Punkte der Höhe und Breite. Rechts: Die ausgeschnittene Version mit angepasster Bounding Box.	55
5.8	Einlesen und Vorverarbeiten der ersten Iteration der Pupillen-Annotationen.	56
5.9	Trainingsfortschritt in 20 Epochen mit dem neuen Unity Datensatz. Die verwendeten Metriken sind mAP.95, Precision und Recall.	57
5.10	Beispiele der Pupillen, für die mit der zweiten Iteration des Pupillenmodells keine Annotation erzeugt werden konnte.	57
5.11	Erstellen des Iris- und Pupillendatensatzes aus dem Unity Eyes Modell unter Zuhilfenahme von Deep Learning zum Annotieren der Bounding Box für Pupillen.	58
6.1	Die mAP.95 der vier verwendeten Größen des YOLO Modells auf dem UnityEyes Datensatz.	60
6.2	Der Recall der vier verwendeten YOLO Größen auf dem UnityEyes Datensatz.	61
6.3	Die mAP.95 der getesteten, zufällig gezogenen Parameter für ein Nano Modell über 42 Stichproben.	62
6.4	Die mAP.95 der getesteten Parameter für ein Nano Modell über 60 Stichproben in der Bayes'schen Suche.	64
6.5	Parallel Coordinate Plot der überprüften Trainingsparameter während der Bayes'schen Suche. Die Farbe repräsentiert die erreichte mAP.95.	65

6.6	Parallel Coordinate Plot der überprüften Eingabeparameter während der Bayes'schen Suche. Die Farbe repräsentiert die erreichte mAP.95.	65
6.7	Die GPU Auslastung und benötigte Zeit für das Training von 20 Epochen des Nano Modells.	66
6.8	Das Trainingsverhalten über 20 Epochen des Nano Modells mit unterschiedlichen Batch Sizes und optimierten Hyperparametern.	67
6.9	Die mAP.95 für Large, Nano und optimiertes Nano auf dem Unity Datensatz über 20 Epochen.	68
6.10	Die Precision für Large, Nano und optimiertes Nano auf dem Unity Datensatz über 20 Epochen.	68
6.11	Der Recall und die mAP.95 für Large, Nano und optimiertes Nano auf dem Unity Datensatz über 20 Epochen.	69
6.12	Precision und $F_1 - Score$ für das finale Nano Modell.	70
6.13	Die mAP.95 der verkleinerten Modelle auf dem UnityEyes Datensatz.	72
6.14	Der Recall der verkleinerten Modelle auf dem UnityEyes Datensatz.	72
A.1	Die Verteilung der Label und der Bounding Boxen im finalen Datensatz.	82
A.2	Parallel Coordinate Plot der Zufälligen Suche. Verwendet die Maxima und Minima des jeweiligen Parameters als oberes und unteres Limit.	83
A.3	Parallel Coordinate Plot der Bayes'schen Suche. Verwendet die Maxima und Minima des jeweiligen Parameters als oberes und unteres Limit.	83

Akronyme

AI Artificial Intelligence

AP@.50 Average Precision mit einem Schwellwert von 50%

AP@.75 Average Precision mit einem Schwellwert von 75%

API Application Programming Interface

CARS Computer Assisted Radiography and Surgery

CDSS Clinical Decision Support System

CHI Computational Health Informatics

CHT Circular Hough Transformation

CNN Convolutional Neural Network

CPU Central Processing Unit

DL Deep Learning

DNN Deep Neural Networks

DSP Digital Signal Processor

FN False Negative

FP False Positive

FPS Frames per Second

GFLOPs Giga Floating Operations

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

GUI Graphical User Interface

ICP Intracranial Pressure

IoU Intersection over Union

JIT just-in-time

JVM Java Virtual Machine

LSTM Long Short-Term Memory

mAP mean Average Precision

mAP.95 mean Average Precision zwischen 0.5 und 0.95 mit Schrittgröße 0.05

mAR mean Average Recall

NNAPI Neural Networks Application Programming Interface

NPU Neural Processing Unit

R-CNN Region-based Convolutional Network

RoI Region of Interest

SoC System-on-Chip

SSD Single Shot MultiBox Detector

TN True Negative

TP True Positive

TPU Tensor Processing Unit

YOLO You Only Look Once

1

KAPITEL

Einleitung

1.1 Motivation	1
1.2 Aufbau dieser Arbeit	3

1.1 Motivation

Was umfasst der Begriff der „Medizin“? Die Heilung von kranken Menschen ist eine erste Assoziation, die in den Kopf gelangt. Ärzte sind dabei meist die Personen, denen solche Heilungen zu verdanken sind. Sie diagnostizieren Krankheitsbilder von Patienten basierend auf deren Symptomatik, Stammdaten und ihrer eigenen Expertise. Im besten Fall begleiten sie den Genesungsverlauf des Patienten, stehen aber auch bei Komplikationen beratend zur Seite. Durch die technologischen Fortschritte der letzten Jahrzehnte, besonders in dem Bereich der Artificial Intelligence (AI), gewinnt die Computer-assistierte Diagnose und Behandlung in der Medizin, neben der Erfahrung des behandelnden Personals, an Bedeutung.

Über die generelle Verwendung und Optimierung von Robotik in der Chirurgie wird intensiv geforscht und vermehrt werden diese Ergebnisse auch in klinischen Tests untersucht [9, 12, 21]. Diese technischen Ergänzungen in der Medizin lassen sich nicht nur in den physischen Eingriffen, sondern auch in der Auswertung der bildgebenden Verfahren wiederfinden: Das Clinical Decision Support System (CDSS) unterstützt durch eine Analyse der vorliegenden Daten die behandelnden Ärzte, eine Diagnose zu finden oder dabei, neue Einblicke in die Krankheitsbilder zu gewähren. Journals, wie das International Journal of Computer Assisted Radiography and Surgery (CARS), berichten von und erstellen Wettbewerbe für die Auswertung von Aufnahmen der Lunge. Hierbei sollen Verletzungen und Tumore [14] oder COVID-19 [18] mittels AI erkannt werden.

Mit Hinblick auf die technischen Innovationen in der Medizin und einer starken Verbreitung von Smartphones in den letzten Jahren [20] entstand 2021 in der Masterarbeit von *Moryäner* [26], unter Leitung des Fachgebiets Computational Health Informatics (CHI), eine App als Indikator für Schädel-Hirn-Traumata. Ziel war es, per Smartphone den Lichtreflex der Pupillen zu vermessen, um so eine schnelle Auswertung zu liefern, ob eine akute Gefahr auf bleibende Schäden besteht und der Patient entsprechend das Krankenhaus aufsuchen soll. In der Arbeit wurden verschiedene klassische Algorithmen auf ihre Präzision und Geschwindigkeit überprüft, erwiesen sich aber trotz Optimierungen als nicht zufriedenstellend. Ein weiterer Ansatz, der in Grundzügen bearbeitet, implementiert und überprüft wurde, war das Klassifizieren durch ein Deep Learning (DL) Modell. Es zeigt, trotz geringer Optimierung, eine hohe Genauigkeit und Geschwindigkeit und wird somit als sehr vielversprechend bewertet.

DL hat im letzten Jahrzehnt mit neuen Durchbrüchen in der Informatik eine Renaissance erfahren dürfen [NN25, NN22]. Losgelöst von früheren Beschränkungen hinsichtlich der Leistung, der Algorithmik und der Verfügbarkeit von Datensätzen, gewinnt dieser Bereich immer mehr Bedeutung in der Industrie und Forschung, sodass auch staatliche Fördermaßnahmen dieses Design-Paradigma fördern [23, 24]. In der Industrie sind dabei besonders die Arbeiten von Google relevant. Mit Android bieten sie eine offene Plattform für die Entwicklung von Anwendungen für Smartphones; mit Tensorflow ein quelloffenes Framework für die Entwicklung und Ausführung neuronaler Netze [NN31]. Weiterhin ist Facebook mit ihrem Tensorflow Konkurrenten Torch und einer sehr aktive Open-Source Community hervorzuheben.

Mit Hilfe einer Analyse verschiedener Architekturen von Modellen zur Objekterkennung und einer Optimierung ebenderer, soll die mobile Applikation, die in der Arbeit von *Moryäner* entwickelt wurde, verbessert werden. Der Fokus liegt dabei auf einer höheren Präzision und besseren Leistungseigenschaften des DL Modells, um so Patienten einen möglichst optimalen Indikator zu geben, um Schädel-Hirn-Traumata frühzeitig sicher zu erkennen. Weiterhin soll eine Anleitung geschaffen werden, wie und in welchen Teilbereichen der Konzeption und des Trainings einer solchen mobilen Applikation optimiert werden kann. Dabei werden verschiedene Architekturen auf deren Anwendbarkeit untersucht, Vor- und Nachteile beleuchtet und gezeigt, wie diese verbessert werden können. Für künftige Arbeiten mit DL Modellen lässt sich hierin eine Anleitung sehen, welchen Einfluss verschiedene Optimierungsmethoden haben und wie stark diese das Trainingsergebnis beeinflussen. Weiterhin soll aufgeführt werden, welchen Einfluss die Hyperparameter auf das Training nehmen. Die auftretenden Problematiken werden erörtert und eine Lösung beschrieben.

1.2 Aufbau dieser Arbeit

Das erste Kapitel stellt die Notwendigkeit technischer Assistenz im medizinischen Umfeld dar und legt ein Ziel fest. Im zweiten Kapitel werden verwandte Forschungsfelder und die App von *Moryäner* vorgestellt. Das dritte Kapitel behandelt die Grundlagen zur Anatomie des Auges, der Android App-Entwicklung, neuronaler Netze und deren Funktionsweise in der Bilderkennung. Die Methodik in Kapitel vier umfasst die Recherche nach Datensätzen, die verwendeten Metriken und Hilfswerkzeuge zum Trainieren der Modelle, sowie eine Beschreibung der Hyperparameter und verwandter Methoden, die zur Optimierung verwendet werden. Mit dem fünften Kapitel werden die bestehenden Datensätze untersucht und es wird argumentiert, warum die Erstellung eines neuen Datensatzes nötig ist. Die Optimierung der Netze selbst und Vergleiche dieser, werden in Kapitel sechs aufgeführt. In Kapitel sieben wird die überarbeitete Android Implementation vorgestellt und beschrieben, welche Änderungen vorgenommen wurden. Das achte Kapitel schließt mit einem Fazit und einem breiteren Ausblick auf die Optimierung von DL Modellen, die aktuell in Entwicklung sind und künftig in neue Hard- und Software einfließen werden. Zudem werden die Probleme, welche in der Arbeit aufgetreten sind, erörtert.

KAPITEL 1. EINLEITUNG

2

KAPITEL

Verwandte Arbeiten

2.1	Zugrundeliegende Arbeit	6
2.2	Bildsegmentierung	7
2.3	Gaze Estimation	7
2.4	Automated Machine Learning	8
2.5	Soziale Verantwortung	8

Diese Arbeit befasst sich mit der Nutzung von Methoden des maschinellen Lernens für eine medizinische Anwendung. Entsprechend dieser interdisziplinären Aufgabenstellung werden viele verschiedene Themengebiete angesprochen und können Richtungen vorgeben, in die weitere Forschung führen kann. In der zugrundeliegenden Arbeit werden primär klassische Verfahren der Bilderkennung und die biologischen Grundlagen erläutert, letztendlich aber eine weitere Implementation mit neuronalen Netzen nahegelegt, da diese bereits in ersten Tests sehr gute Ergebnisse bei überzeugender Leistung geboten haben. Die Gaze Estimation (dt. Blickschätzung) behandelt die Augensegmentierung und bietet somit Überschneidungen in der Methodik und interessante Anhaltspunkte für die Forschung in diesem Gebiet. Algorithmen zur Bildsegmentierung stellen dabei eine Methode dar, die bereits in *Moryäners* Arbeit mit eingeschränktem Erfolg verwendet wurde; die Leistungsansprüche stellen sich als zu hoch für eine Echtzeitanwendung auf dem Mobiltelefon heraus. Die Alternative auf Modelle im Bereich des DL zu setzen, weist erste Erfolge auf. Diese Modelle können meist eine Optimierung der Hyperparameter erfahren, welche eine bessere Grundvoraussetzung für das folgende Training bieten. Dabei gibt es verschiedene Methoden, wie eine solche Optimierung durchgeführt werden kann. Letztlich wird die soziale Verantwortung, die mit einer medizinischen Anwendung einhergeht, angeschnitten, da diese für alle Patienten geeignet sein soll.

2.1 Zugrundeliegende Arbeit

Als Inspiration für diese Arbeit steht eine vorherige Masterthesis aus dem Fachgebiet CHI, die von *Alexander Moryäner* [26] verfasst wurde. Dessen Arbeit behandelt die Konzeption, Implementation und Optimierung einer Anwendung für mobile Endgeräte, mit welcher ein mobiler und einfach zugänglicher Indikator geschaffen werden soll, der potentielle Hirn-Schädeltraumata erkennt. Als Plattform für diese Anwendung dient, aufgrund der Verfügbarkeit und Zugänglichkeit, das von Google entwickelte Betriebssystem Android.

Moryäner schafft die wissenschaftlichen Grundlagen, dass einerseits der Pupillenreflex ein zuverlässiger Indikator für ebendiese Traumata darstellen kann, andererseits, dass dieser Lichtreflex durch den LED-Blitz einer üblichen Smartphone-Kamera ausgelöst wird. Für die Implementation der Erkennung und Vermessung der Pupille wurde erst auf klassische Verfahren zur Bildsegmentation und Objekterkennung gesetzt.

Namentlich waren dies die Circular Hough Transformation (CHT) [CV1, CV6, CV9, CV12] und der Daugmans Integro Differential Operator [CV2, CV8, CV13], denen sich jeweils einige Arbeiten im Bereich der Augensegmentation widmeten. Weiterhin wurde das k-Means-Clustering verwendet, da es ein simpler und effektiver Segmentationsalgorithmus ist, der durch zahlreiche Verbesserungen und die Nutzung spezialisierter Metriken auch effizient einsetzbar sein kann [CV3, CV4, CV5]. Eine hier relevante Metrik ist die approximative Abschätzung des Iris- und Pupillenmittelpunktes, der durch Gradienten im Vektorfeld berechnet geschätzt wird, von *Timm et al.* [CV7] beschrieben, berechnet wird. Mit diesem Mittelpunkt konvergiert der k-Means-Algorithmus konsistenter und schneller. Andere Methoden hierfür werden in [CV10, CV11] vorgestellt.

Trotz verschiedener Optimierungen, wie der Parallelisierung durch Multithreading, der Reduktion auf einen Teilbereich der Iris und dem Verwenden der erwähnten Mittelpunktsheuristik, bewegte sich die Laufzeit im Bereich von der CHT, dem Daugman Algorithmus und k-Means im Durchschnitt bei 1,08 s, 1,55 s respektive 11,23 s für jede einzelne Eingabe. Gleichzeitig lag die Erkennungsrate im Durchschnitt bei nur etwa 72-75 %. Aufgrund dieser durchwachsenen Ergebnisse wurde ein anderer Ansatz verfolgt: Eine künstliche Intelligenz basierend auf dem SSDLite MobileNetV3 [NN50] wurde implementiert, ausgewertet und ergab auf einem selbst erstellten Datensatz bei einer Laufzeit von 0,22 s eine Erkennungsrate von ca. 93 %.

In dieser Arbeit soll dieser auf neuronalen Netzen basierende Ansatz aufgegriffen und optimiert werden.

2.2 Bildsegmentierung

Neben der Objekterkennung, die in dieser Arbeit verwendet wird, kann das Problem der Augenerkennung auch mit Hilfe von Segmentationsansätzen per maschinellem Lernen gelöst werden. Traditionelle Algorithmen wie das k-Means Clustering [CV4] wurden bereits in der vorherigen Arbeit erörtert und boten Probleme auf.

Im Bereich der neuronalen Netze hingegen ist das Lernen eines Autoencoders eine mögliche Lösungsstrategie. In Autoencodern werden in einem ersten Schritt iterativ die Informationen aus dem Eingabebild durch Convolutional Layer extrahiert, bis nur noch ein dichte besetzter Vektor entsteht. Dieser verkörpert den latenten Raum des Bildes und soll somit eine Segmentierung in die wichtigsten Bestandteile des Bildes ermöglichen. Aus diesem kann durch die Deconvolution (dt. Entfaltung), also ein iteratives Upsampling (dt. Hochskalierung), im zweiten Schritt wieder ein Bild generiert werden. Das generierte Bild wird also aus einer Karte, die die Informationen der Segmentation beinhaltet, aufgebaut. Beispielhaft für diese Architektur ist das U-Net [NN24], das sich verschiedener Methoden bedient, um ein möglichst originalgetreues Ausgabebild zu erzeugen.

Dieser Ansatz kann potentiell robuster gegenüber atypischen Eingaben, beispielsweise einem Kolombom, einer Störung der Iris, sein. Allerdings stellen ihm sich eine höhere Leistungsanforderung und schwer zugängliche Datensätze entgegen. Dementsprechend wird dieser Ansatz nicht weiter verfolgt.

2.3 Gaze Estimation

Im Feld der Gaze Estimation wird versucht, anhand einer Kameraaufnahme, sei es Bild oder Video, die Blickrichtung des Nutzers zu ermitteln. Für diese Analyse gibt es in verschiedenen Bereichen Motivation, da sie helfen kann, Krankheiten zu erkennen, eine neue Mensch-Computer-Interaktion darstellen und zum Erstellen und Optimieren von Werbung genutzt werden kann [G5]. *Zhu et al.* [G4] verwenden die Gaze Estimation zum Implementieren eines interaktiven Graphical User Interface (GUI). *Ji et al.* [G1, G2, G3] zeigen in drei Arbeiten Methoden auf, um die Aufmerksamkeit von Autofahrern zu bestimmen, um so die Sicherheit im Straßenverkehr zu erhöhen. Wie *Morimoto et al.* [G5] zeigen, besteht seit Jahrzehnten Interesse in diesem Gebiet, jedoch konnten die Verfahren zur Auswertung noch keine überzeugende Präzision erreichen und waren somit noch nicht vollständig dafür geeignet, in einem allgemeinen Kontext außerhalb des Labors, verwendet zu werden.

DL Methoden haben in diesem Bereich für Neuerungen gesorgt, die zu einer höheren Genauigkeit bei gleichzeitig geringerer Berechnungszeit führen, wie *Kafka et al.* [G6] zeigen. Daher wird dieser Ansatz aufgegriffen und in der Arbeit weiter ausgeführt.

2.4 Automated Machine Learning

Im Feld des automatisierten maschinellen Lernens ist der Handlungspunkt die Optimierung der Hyperparameter, also der änderbaren Rahmenbedingungen eines Modells, die nicht während eines Trainingsvorgangs angepasst werden. Sie liegt semantisch somit über der Gewichtsoptimierung, die während eines Trainingsvorganges stattfindet. Das Kernproblem ist es, einen ausgewogenen Kompromiss zwischen der Exploration (dt. Erforschung) und der Exploitation (dt. Ausnutzung) zu finden. Die Exploration befasst sich mit der Suche nach geeigneten Startkonfigurationen, die ein hohes Potential zur lokalen Optimierung, der Exploitation, bieten.

Erste Algorithmen zum Vergleich unterschiedlicher Konfigurationen verwendeten ein simples Racing-Verfahren, ließen den Gewinner in einem Portfolio von Konfigurationen gegen die Gewinner anderer Portfolios antreten. Mit dem F-Race Algorithmus wurde die Trainingszeit auf das Nötigste reduziert, indem diese „Rennen“ abgebrochen wurden, wenn ein signifikanter Gewinner bereits vor dem Ende ermittelt werden konnte [HPO1]. Für diesen Racing-Ansatz folgte die Erweiterung um eine lokale Suche [HPO2] und eine weitere Optimierung für das Verfahren der vorzeitigen Auslese [HPO4]. Weitere Optimierungen in der Loss-Function (dt. Verlustfunktion) und der Konfigurationsselektion wurden mit Hydra getroffen [HPO6]. Ein genetischer Algorithmus soll mit dem Paradigma einer lokalen Suche in verschiedenen Portfolios konkurrieren, indem von vielen Startpunkten parallel eine Evolution stattfindet [HPO3].

Während erste Ansätze also auf ein übergeordnetes Modell zur Auswertung gelieferter Ergebnisse verzichteten oder ein solches nur in eingeschränktem Ausmaß anwandten, verwenden aktuelle Ansätze meist Modelle, um die Trainingsvorgänge möglichst effizient zu verwalten. SMAC verwendet dabei einen Ansatz mit Random Forests, um ein Modell zu erstellen, anhand dessen vielversprechende Parameterkonfigurationen gefunden werden können [HPO5]. Mit SMAC3 und anderen Optimierungen wird an diesem Ansatz weiterhin entwickelt [HPO14]. *Shahriari et al.* [HPO8] zeigen in ihrer Arbeit eine detaillierte Übersicht der verschiedenen Verfahren, die auf einer solchen Bayes'schen Suche basieren, auf und analysieren sie.

2.5 Soziale Verantwortung

Im Rahmen der Verwendung von computergestützten Verfahren für Vorverarbeitung und Klassifizierung von Themen, die auch Menschen im Alltag direkt berühren, ist es besonders wichtig, eine soziale Verträglichkeit ebendieser Anwendungen zu gewährleisten. Im ersten Schritt, dem Erstellen eines Datensatzes, ist es wichtig, sich bereits vorher Gedanken über die Daten, die aufgezeichnet werden, zu machen. Nur Daten, die auch aufgenommen und explizit als solche gespeichert werden, beeinflussen das letztendliche Ergebnis, wohingegen nicht aufgezeichnete Daten zu einer sozial problematischen

Qualität in der Klassifikation führen können [SR6]. Es muss darauf geachtet werden, dass besonders im öffentlichen Sektor schädliche Implementationen zu Unmut und Benachteiligungen in der Bevölkerung führen können [SR7, SR5]. Weiterhin müssen die aufgenommenen Daten trotz dessen auf weitere Proxies (dt. Stellvertreter) untersucht werden, damit eine Klassifikation nicht indirekt, beispielsweise durch die Hautfarbe oder das Geschlecht von Personen in Bereichen, in denen diese Attribute irrelevant sein sollen, beeinflusst wird.

Für das optimierte Training von Modellen gibt es Verfahren zum Lernen, um die Privatsphäre der Geber der Trainingsdaten zu verbessern. Diese Verfahren zählen zur Differential Privacy (dt. differentielle Privatsphäre) und haben grundsätzlich das Problem, die Qualität des Trainings zu verschlechtern. Durch Methoden, wie dem Gradient Clipping [NN63] oder dem Hinzufügen von Rauschen auf die Trainingsdaten, werden bereits schlecht abschneidende Klassen überproportional benachteiligt [SR4].

Das Pruning (dt. Beschneidung) und die Quantisierung sind Methoden der Performanzoptimierung für Modelle. Sie nehmen starken Einfluss auf eine Verteilung der Klassifikationsgenauigkeit der Klassen untereinander. Besonders Klassen, die wenig Eigenschaften mit anderen Klassen teilen, sind überproportional vom Pruning betroffen. Schwache Neuronenverbindungen, die beim Optimieren entfernt werden, sorgen für die Differenzierung dieser unterrepräsentierten Klassen. Hierdurch kann eine Unfairness für seltene Klassen oder Klassen, die kaum Gemeinsamkeiten mit den anderen teilen, entstehen [SR8].

Chawla et al. [SR1] nehmen sich dem Problem der Unterrepräsentation an und stellen den Ansatz der „Synthetic Minority Oversampling Technique“ (dt. Methode zur synthetischen Überrepräsentation von Minderheiten), kurz SMOTE, vor. *Galar et al.* [SR2] bauen auf diesem Ansatz auf und zeigen, dass Ensembles von Modellen mit einem spezialisierten Training den Konsequenzen der Unterrepräsentation entgegenwirken können. Ein Vergleich des Umgangs mit diesem Problem wird von *Branco et al.* [SR3] über verschiedene Sektoren und Ansätze hinweg gezeigt und es wird erklärt, wie erfolgreich verschiedene Methoden angewandt werden.

KAPITEL 2. VERWANDTE ARBEITEN

3

KAPITEL

Grundlagen

3.1	Das menschliche Auge	12
3.1.1	Aufbau des Auges	12
3.1.2	Der Pupillenreflex	13
3.1.3	Schädel-Hirn-Trauma	14
3.2	Android	16
3.2.1	NNAPI	17
3.3	Neuronale Netze	19
3.3.1	Convolutional Neural Networks	20
3.3.2	Lernmethoden	21
3.3.3	Ensemblestrukturen	22
3.3.4	Transfer Learning	22
3.4	Deep Neural Networks	23
3.5	Objekterkennung mit neuronalen Netzen	25
3.5.1	Region-based Convolutional Network	25
3.5.2	Single Shot Detector	26
3.5.3	Non-Maximum Suppression	27
3.6	Die YOLO Architektur	27
3.6.1	Ultralytics YOLOv5	30

In diesem Kapitel werden die Grundlagen, auf denen diese Arbeit aufbaut, beschrieben. Die biologischen Grundlagen des Auges und der Zusammenhang zwischen dem Pupillenreflex und einem Schädel-Hirn-Trauma werden erklärt. Aufgrund der Verwendung von Android für die Anwendung wird dieses zusammen mit dessen Application Programming Interface (API) für die Verwendung neuronaler Netze erläutert. Diese Netze und deren Architekturen zur Bilderkennung werden auch erklärt.

3.1 Das menschliche Auge

In der Evolution werden die besten Veränderungen hinsichtlich der Anpassung an die Umgebungsbedingungen iterativ zu dominanten Charakteristika dieser Spezies. Beim Menschen sind an dieser Stelle besonders der aufrechte Gang und das Gehirn in den Vordergrund zu stellen. Diese Arbeit befasst sich mit der Perzeption durch die Augen und die Reaktionen, die im Gehirn bei einem starken Lichtimpuls ausgelöst werden. Die Augen stellen nicht nur lyrisch das Fenster zur Seele dar, sondern sind auch gesellschaftlich und medizinisch von großer Bedeutung [EYE3]. So sagt die *Cooperative eye hypothesis* aus, dass die Sclera sich evolutionär von einem bräunlichen Ton bei (Menschen-)Affen zu einem weißen Ton entwickelte, weil der Kontrast zur Iris so stärker zum Vorschein kommt und damit der Blick des Gegenübers besser nachvollzogen werden kann, welches Vertrauen innerhalb einer Gruppe schafft [EYE4, EYE10]. Dieser Kontrast kann aber auch für die Auswertung der Augen genutzt werden: Die Gaze Estimation verwendet die Augen- und Pupillenerkennung, um den Blickwinkel der Person abzuschätzen und stellt somit die technische Auswertung dieser Gruppenkontrolle dar. Aufbauend auf diesen Methoden, können diese Daten auch anderweitig medizinisch, wie in dieser Arbeit, verwendet werden.

Neben der Farbe der Augen blieb auch die Vielzahl von Muskeln, Nerven und Mechanismen, die für den Sehapparat zuständig sind, nicht unverändert. Diese teils winzigen, bio-mechanischen Abläufe müssen präzise zusammenarbeiten, um eine klare Sicht zu gewährleisten. Entsprechend sind Fehler in diesem Ablauf meist auf Schäden zurückzuführen, die sich unterschiedlich äußern können. Im folgenden Abschnitt wird der Aufbau des Auges skizziert, die gesuchte Segmentierung der Augen aufgeführt und die biologischen Grundlagen des Pupillenreflexes mitsamt dessen Bedeutsamkeit für ein Schädel-Hirn-Trauma beschrieben, um so die Problemlösung zu spezifizieren.

3.1.1 Aufbau des Auges

In der Abbildung 3.1 ist links eine schemenhafte Darstellung des menschlichen Auges zu sehen. Die weiße Sclera (dt. Lederhaut) umschließt schützend das Auge und wird auf der Hinterseite durch den Sehnerv, auf der Vorderseite durch die Cornea (dt. Hornhaut) unterbrochen. Die Cornea ist lichtdurchlässig und sorgt somit für den Lichteinfall auf die Linse, die dieses Licht auf die Retina (dt. Netzhaut) bricht. Vor der Linse befindet sich die Iris (dt. Regenbogenhaut), welche die Menge des einfallenden Lichts bei Bedarf durch das Steuern eines Ringmuskels und eines radiären Muskels regulieren kann. Der Muskel ist der Sphincter Pupillae, welcher um die Pupille liegt und diese bei Kontraktion verengt, bei Muskelrelaxation wieder in den Ausgangszustand wachsen lässt. Der radiäre Dilator Pupillae hingegen spannt sich grob orthogonal zur Pupille und sorgt bei Kontraktion für eine Weitung der Pupille. Je stärker verengt die Pupille ist, desto geringer ist der Lichteinfall auf die Netzhaut. Die Netzhaut kann nur eine beschränkte Menge Licht als Reiz über den Sehnerv an das Gehirn übertragen. Entsprechend wird die Größe der Pupille

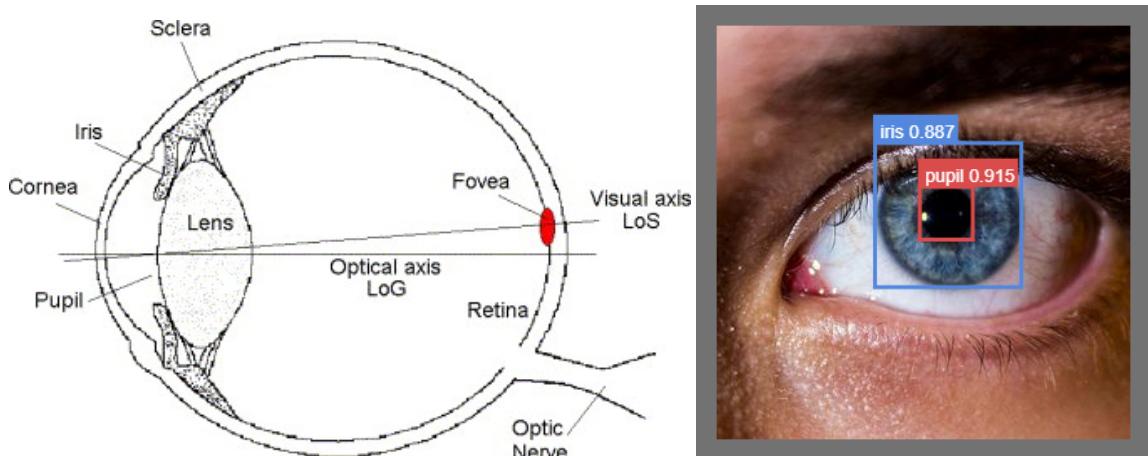


Abbildung 3.1: Schemenhafter Aufbau eines Auges links [G5]. Markierung der Pupille und Iris auf einem echten Bild rechts. (geändert nach [DATA11])

durch eine komplexe Interaktion mit dem Stammhirn durch gezielte Kontraktion oder Relaxion der beiden Muskeln gesteuert. Details dazu werden in Abschnitt 3.1.2 erläutert.

Auf der Abbildung 3.1 rechts ist die gewünschte Objekterkennung zu sehen: Innerhalb der Sclera liegt die Iris, welche wiederum die Pupille einschließt. Die Größe der Iris ist konstant, variiert zwischen Menschen aber von 11.4 mm und 12.7 mm; durchschnittlich liegt sie somit beim Menschen bei etwa 12 mm [EYE13, EYE12]. Im Gegensatz dazu ändert sich die Größe der Pupille stetig je nach Lichteinfall. Bei hellem Licht beträgt der Durchmesser der Pupille während der Miosis etwa 1.5 bis 4 mm, in der Dunkelheit während der Mydriasis rund 4 bis 8 mm [EYE2]. Unter alltäglichen Bedingungen misst sie durchschnittlich etwa 3-5 mm [EYE12]. Sie ist weder vom Geschlecht noch von der Farbe der Iris abhängig [EYE11], korreliert jedoch negativ mit dem Alter [EYE5, EYE15].

3.1.2 Der Pupillenreflex

Als Fundament dieser Arbeit steht ein Lichtreflex, der die Iris bei plötzlicher, starker Beleuchtung dazu bringt, sich zusammenziehen. Dabei wird die Menge des Einfallslichts reduziert, um das Sehen weiterhin zu ermöglichen und die Netzhaut vor einer Überbelastung zu schützen. Dieser Reflex wird als phasischer Pupillenreflex bezeichnet und dessen Auswirkung wird in Abbildung 3.2 gezeigt [EYE1, EYE6]. Klar zu erkennen ist dabei die unterschiedliche Größe der Pupille. Im linken, entspannten Zustand beträgt die Größe rund 4 mm. In der Mitte wurde der Durchmesser auf etwa 2 mm verkleinert.

Der exakte zeitliche Verlauf der Pupillengröße wird aus der Literatur nicht genau erkennbar; ein qualitatives Muster zeigt sich insgesamt hingegen schon. Dieses wurde in Abbildung 3.2 dargestellt und zeigt zu Beginn eine mindestens 200 ms lange Latenz, vom Lichtimpuls bis zum Zusammenziehen des Muskels. Nach mindestens weiteren 280 ms, tendenziell mehr, setzt die maximale Pupillenverengung ein. Diese hält für einige hundert Millisekunden an, wonach die Pupille langsam wieder an Größe zunimmt und die Ausgangsgröße annimmt. Je nach Umgebungshelligkeit, Stärke des Impulses und dem

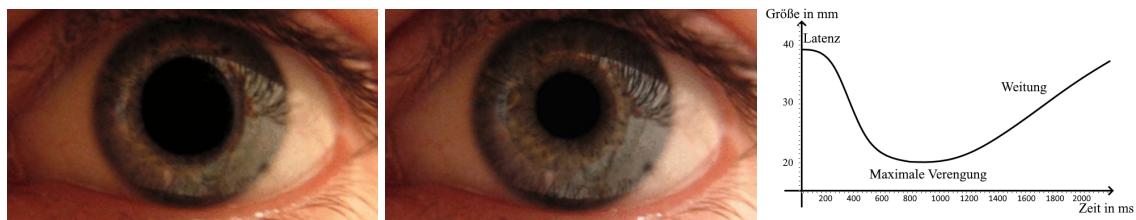


Abbildung 3.2: Der Pupillenreflex. Links ist eine Pupille im Ausgangszustand zu sehen, mittig die zusammengezogene Pupille nach einem Lichtimpuls. (geändert nach [EYE9]). Rechts: Qualitativ eingezeichneter Verlauf der Pupillengröße.

Lichtspektrum des Impulses, sowie der physiologischen Beschaffenheit des Probanden, können die Werte dabei anders ausfallen [EYE8, EYE6, EYE7].

Der Pupillenreflex ist keine direkte, lokale Reaktion, die von der Iris oder dem Auge ausgelöst wird, sondern muss erst im Hirnstamm ausgewertet und über das Nervensystem an die jeweiligen Muskeln weitergeleitet werden, bis diese die Pupille verengen. In Abbildung 3.3 ist der Aufbau der relevanten Nerven, Muskeln und Gehirnareale dargestellt. Dieser Vorgang erklärt die Latenz im Bereich von 200-500 ms nach dem Beleuchten des Auges, bis die Pupille eine Kontraktion erfährt, da erst eine Verarbeitung des Reizes stattfinden muss. Im Detail werden die Reizinformationen nach der Aufnahme auf der Netzhaut über den Nervus Opticus einerseits in die Hirnrinde, in der das Bewusstsein und die visuelle Perzeption stattfindet, nicht aber die Reflexe verarbeitet werden, weitergeleitet. Andererseits spalten sich einige Nervenfasern vom Nervus Opticus ab und leiten den Reiz an den Hirnstamm, im Besonderen der Commissura Posterior, weiter. In der Area Praetectalis wird aus der Stärke des Reizes die Umgebungshelligkeit bestimmt und mit biologisch festgelegten Werten verglichen, unabhängig davon, von welchem Auge dieser Reiz stammt. Bei Differenzen zwischen dem Ist- und dem Soll-Wert, werden entsprechende Befehle an den Nucleus Oculomotorius Accessorius, auch als Edinger-Westphal Kern bekannt, gegeben, die über parasympathische Fasern in den Ganglion Ciliare gelangen. Dort werden diese Signale nahe dem Auge auf postsynaptische Fasern umgeschaltet, die das letztendliche Signal zur Miosis an die Sphincter Pupillae beider Augen senden [EYE16].

3.1.3 Schädel-Hirn-Trauma

Schädel-Hirn-Traumata sind mit ca. 330 Fällen pro 100.000 Einwohnern in Deutschland ein häufige Diagnose und verdient mit 7.700 Opfern jährlich große Aufmerksamkeit. Etwa die Hälfte aller Vorfälle sind Folge eines Sturzes, 27 % von Verkehrsunfällen und über 5 % entstehen beim Sport. Dabei ist es möglich, dass der Patient zunächst nur wenig Auffälligkeiten zeigt, innerlich aber bereits starke Verletzungen erlitt und Blutungen vorliegen, die zu Sekundärschäden oder zum Tod durch einen zu großen Hirndruck, also dem Intracranial Pressure (ICP), führen können. Die primären Schäden, beispielsweise an den Nervenzellen, sind meist nicht reversibel und führen somit bei zu später Erkennung zu permanenten Schäden am Patienten. Sekundäre Schäden sind meist behandelbar, ber-

gen trotz dessen aber massive Risiken [6, 7, 8].

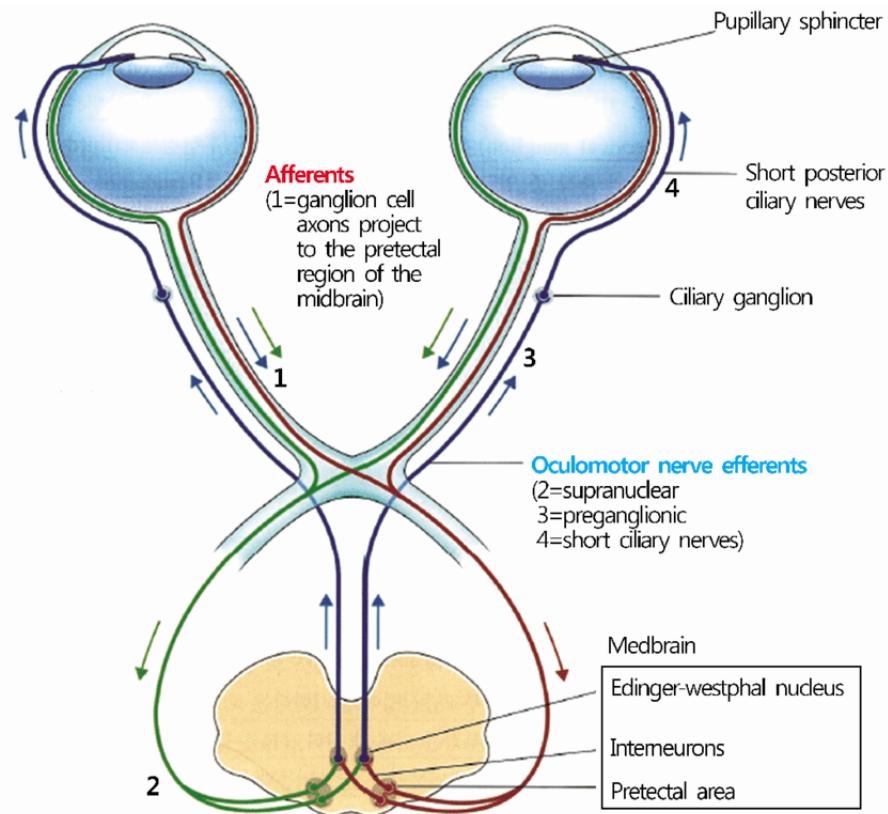


Abbildung 3.3: Aufbau des Sehapparats mit Fokus auf die Steuerung der Pupillengröße [EYE14].

Diese Traumata lassen sich neben klassischen Methoden wie der Glasgow Coma Scale [5] häufig auch über die Augen symptomatisch erkennen. Afferente Störungen sind dabei verantwortlich dafür, dass eine Kommunikation über den Nervus Opticus von einem Auge gestört oder unterbrochen ist, wodurch bei einseitiger Belichtung ein schwächerer oder gar kein Reiz übermittelt werden kann. Dies hat zur Folge, dass der Pupillenreflex nicht oder nur abgeschwächt ausgelöst wird. Efferente Störungen hingegen behindern die Kommunikation vom Hirnstamm in Richtung der Muskeln in der Iris, wodurch der Reflex nur einseitig oder nicht ausgeführt wird. Eine Kombination aus afferenten und efferenten Störungen ist auch möglich und äußert sich in einer unterschiedlichen Reaktionsstärke auf einen eingehenden Reiz [EYE16, 1].

Für diese Arbeit ergibt sich somit, dass die Diagnostik über das bisher verwendete Verfahren hinaus optimiert werden kann. Statt wie bisher über ein Bild vor und ein Bild nach dem Reizimpuls die Pupillengröße beider Augen und deren jeweiliger Größenveränderung zu vermessen, können die Augen einzeln betrachtet werden, um so einseitige afferente oder efferente Störung festzustellen. Weiterhin kann mit einer Videoaufnahme und einer Auswertung dieser ein zeitlicher Verlauf der Pupillengröße, wie er qualitativ in Abbildung 3.2 rechts gezeigt wurde, errechnet werden. Liegen eine zu große Abweichung oder andere

Auffälligkeiten zwischen beiden Augen vor, besteht also eine hohe Wahrscheinlichkeit, dass beim Patienten ein solches Trauma vorliegt und es kann bestimmt werden, in welchem Bereich ein Schaden vorliegt.

3.2 Android

Diese Arbeit adaptiert die Implementation von *Moryäner* [26], welche auf Android setzt. Android ist die verbreitetste Plattform für mobile Anwendungen [27] und wird seit 2008 von Google, einem Unterkonzern von Alphabet, entwickelt und vertrieben. Neben Google selbst verwendet eine Vielzahl von Unternehmen dieses Betriebssystem, da es quelloffen angeboten wird, mit Android Studio eine vollständige Toolchain frei verfügbar anbietet und mit dem Google Play Store eine Möglichkeit eröffnet, die Apps zu vertreiben und zu monetarisieren. Weiterhin setzt Google bei der Programmierung in Java und Kotlin auf die Java Virtual Machine (JVM), also einem virtuellen Interpreter, der den Java-Bytecode in hardwarenahe Instruktionen präkompiliert oder just-in-time (JIT) übersetzt und dabei die benötigten Ressourcen verwaltet. Mit regelmäßigen Updates werden dabei aktuelle Sicherheitsansprüche umgesetzt, neue Features implementiert und die Performanz in vielen Aspekten optimiert.

Mobile Endgeräte haben in dem letzten Jahrzehnt große Sprünge hinsichtlich der Leistung verzeichneten können. Diese sind auf verschiedene Verbesserungen in der Architektur des System-on-Chip (SoC) zurückzuführen. Neben dem Einführen von Prozessoren mit acht Kernen, die selbst für eine höhere Effizienz in der von ARM vorgestellten „big.LITTLE“ Architektur [22] gefertigt werden, haben auch weitere Spezialisierungen Einzug in das Smartphone gefunden. Während eine Graphics Processing Unit (GPU) standardmäßig auch in einem Heimcomputer vorzufinden ist, werden in aktuellen SoCs auch ein Digital Signal Processor (DSP) für die Kamerasignalverarbeitung oder eine Neural Processing Unit (NPU) zum Berechnen von neuronalen Netzen implementiert. Übergeordnet lässt sich viel vom Fortschritt auch über die Verbesserungen in der Fertigungsgröße von aktuell 7-5 nm verorten. Diese spezialisierten Bauteile dienen als Beschleunigern für ihre jeweiligen Aufgaben und ermöglichen somit deutliche Vorteile hinsichtlich Geschwindigkeit und Effizienz gegenüber den allgemein gehaltenen Rechenwerken der Central Processing Unit (CPU).

In der vorherigen Arbeit wurden für eine Anwendung bereits Konzeptionen aufgearbeitet, Assets erstellt oder zusammengetragen und schließlich eine funktionsfähige Anwendung vorgestellt. Da diese allerdings auf einem 6 Jahre alten Android API der Version 5.0 aufbaut, können neuartige Features nicht verwendet werden. Wichtig ist dabei insbesondere das neue Neural Networks Application Programming Interface (NNAPI), das von Google für die Nutzung der integrierten NPUs vorgesehen ist.

3.2.1 NNAPI

Mit der Android Version 8.1 wurde von Google die NNAPI vorgestellt: Es soll eine einheitliche, zentralisierte Schnittstelle für Bibliotheken sein, um auf die Hardwaregegebenheiten der einzelnen Smartphones, trotz ihrer Diversität, zuzugreifen. Wie bereits beschrieben gibt es viele unterschiedliche Ausbaustufen der SoCs und dementsprechend eine große Variabilität darin, für welche Aufgaben passende Beschleuniger vorliegen. Hinsichtlich der Beschleunigung von Berechnungen im Graphen eines neuronalen Netzes sollten CPUs als Fallback (dt. Ausweichmöglichkeit), nicht aber als primäres Rechenwerk gesehen werden. Sie sind nicht für diese Aufgabe optimiert worden und benötigen dementsprechend viel Leistung, um diese Berechnung auszuführen [25].

Im Gegensatz dazu stehen GPUs, welche aufgrund ihrer Anpassung an die Berechnung von Grafiken eine höhere Parallelität und Optimierung für die Berechnung von Matrizen aufweisen. Primär sind sie als Grafikbeschleuniger für Videospiele und aufwendige Animationen gedacht, bieten aber wegen dieser Eigenschaft auch Vorteile für das Berechnen neuronaler Netze. In einem einfachen qualitativen Vergleich von Tensorflow 2.0 auf einem Computer mit einem AMD Ryzen 2700x und einer Nvidia 2080ti ergibt sich ein relativer Unterschied in der Effektivität um den Faktor 2 hinsichtlich der Kosten und 3 hinsichtlich der benötigten Energie [NN49]. In anderen Aufgaben, Modellen oder Vergleichssystemen unterscheiden sich die Werte stärker; allerdings bleibt das Muster stetig, dass eine GPU effizienter in dieser Art der Berechnung bleibt.

NPUs hingegen sind eine noch weitaus neuere Erscheinung in Heimcomputern und Smartphones. Genauso wie die eigens von Google und Nvidia entwickelte Tensor Processing Unit (TPU) sollen sie die Berechnungsgraphen von Modellen und somit die Berechnung von Tensoren, eine Bezeichnung für n-dimensionale Matrizen, beschleunigen. Sie sind deutlich effizienter in der Ausführung als sowohl CPUs als auch GPUs. Diese TPUs haben als Echtzeitbeschleuniger für das Ray-Tracing ihren Weg auf Konsumentenhardware von Nvidia gefunden. Sie sind zwar für einen anderen Zweck auf den Grafikkarten verbaut worden, bieten aber in beiden Gebieten durch das Verarbeiten von Tensoren gleichermaßen Geschwindigkeitsvorteile. Für spezialisierte Umgebungen wie das Cloud Computing rechtfertigt sich auch die Verwendung von Karten, die ausschließlich auf diese TPUs setzen. Die NPUs hingegen werden in aktuellen Prozessoren von Intel auf dem PC-Markt und von Herstellern wie Apple, Qualcomm, Huawei und Google auf dem Smartphone-Markt in ihren eigenen SoCs verbaut. Aber selbst diese NPUs unterscheiden sich untereinander in der Menge der realisierten Befehle und bieten dementsprechend standardmäßig keine homogene API. Um diese schiere Vielfalt von Prozessorarchitekturen zu vereinen und um das volle Potential jedes einzelnen SoCs auszuschöpfen, entwickelt Google also ihre NNAPI [25].

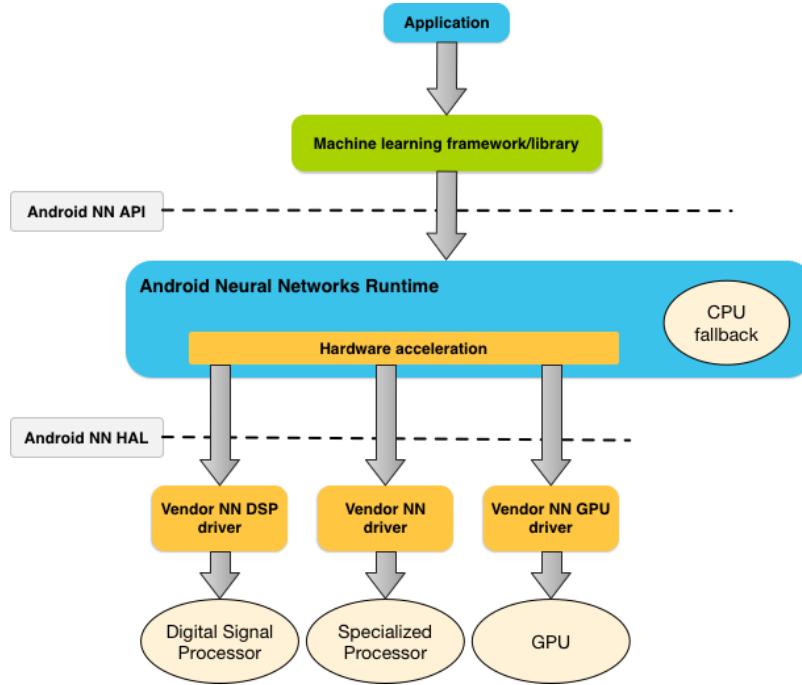


Abbildung 3.4: Die Systemarchitektur der Android NN API [25].

Durch das Ausführen der Modelle auf dem Gerät selbst ergeben sich für den Anwender Vorteile hinsichtlich der Privatsphäre, der Verfügbarkeit und der Latenz der Anwendung, da die Daten niemals das Smartphone verlassen. Für den Anbieter ergeben sich besonders Vorteile hinsichtlich der Kosten, da keine Server für die Inferenz bereitgestellt werden müssen [25].

Die NN API stellt allerdings, wie der Name impliziert, nur das Interface dar. Entsprechend muss also entweder das Modell selbst aufwendig auf dem Gerät implementiert werden oder es wird auf ein Framework, wie Tensorflow Lite, Caffe2 oder Torch Mobile gesetzt. Diese stellen die Kompatibilität zwischen dem Modell, das auf dem Computer trainiert und exportiert wurde, mit der Hardware des Smartphones ohne weiteres Zutun seitens des Entwicklers sicher. Aufgrund der schnellen Entwicklungen der letzten Jahre und weiteren graduellen Fortschritten kann nicht immer eine Kompatibilität für alle Operationen, Modelle und Formate sichergestellt werden.

3.3 Neuronale Netze

Bereits 1943 versuchten *McCulloch et al.* [NN1] sich an einem Konzept zur Nachbildung des menschlichen neuronalen Netzes, dem Gehirn. In ihrem Modell treffen sie die entscheidende Annahme, dass diese Netze mit Hilfe von komplexen, zusammengesetzten Logikgattern nachgebaut werden können. Sie definieren ihr Modell als kreisfrei, statisch und mit einer temporalen Komponente. Frank Rosenblatt [NN2] stellte 1958 basierend auf dieser Arbeit sein Perzeptron, das mit einigen Vereinfachungen und weitreichenden Annahmen einher ging, vor. Es gibt keinen festen Speicher, sondern Eindrücke beziehungsweise Eingaben werden beim Durchqueren des Netzes verarbeitet. Muster werden erst durch ein Training, in dem durch selbstverstärkende Verfahren die Verbindungen zwischen ähnlichen Mustern hergestellt werden und dabei in ihre eigene Struktur aufnehmen, für das Netz erkennbar.

Maßgeblich für die Weiterentwicklung von neuronalen Netzen war die von *Rumelhart et al.* [NN5] vorgestellte Methode der Backpropagation, die eine Verallgemeinerung der Delta Rule, also dem Gradientenabstiegsverfahren, mit dem die Neuronenverbindungen lernen, darstellt. Es ermöglichte das komfortable Lernen von mehrschichtigen Netzen und legte so den Grundstein für die Deep Neural Networks (dt. tiefe neuronale Netze), die heute die Forschung und den Markt beherrschen. Verschiedene Beweise attestierte neuronalen Netzen den Status eines universellen Approximierers, wenn sie mindestens eine Schicht zwischen der Eingabe und der Ausgabe, einem sogenannten Hidden Layer (dt. versteckte Schicht), mit einer Sigmoid Aktivierungsfunktion beinhalteten [NN6, NN7]. Andere Arbeiten verallgemeinerten dies erst auf kontinuierliche, beschränkte, nicht-konstante Funktionen [NN8, NN9] und schließlich auch zusammengesetzte, lokal kontinuierlich und beschränkte Funktionen, solange sie kein Polynom sind [NN11]. Solange eine solche Funktion gewählt wurde, mindestens ein Hidden Layer vorhanden ist und eine adäquate Anzahl von Neuronen gewählt wurde, ist also jede beliebige Abbildung erlernbar.

Im Verlauf der Entwicklung neuronaler Netze wurden zuerst meist an simplen, meist vollvermaschten Schichten geforscht. Diese Art von Layer, auch als Dense Layer bekannt, ist nur eine erste Iteration der verarbeitenden Schichten. Sie weisen die Eigenschaft auf, allgemein anwendbar zu sein, wenn sie wie beschrieben nicht-lineare Aktivierungen innerhalb des Netzes einbinden. Allerdings kommt ihre Allgemeinheit mit den Einbußen einher, dass ihr Rechenaufwand sehr hoch ist, da die Kosten, also die Anzahl der Verbindungen und damit Parameter, mit jedem weiteren Neuron quadratisch steigen. Für eine effizientere Nutzung, besonders in der Bildverarbeitung, werden stattdessen Convolutions (dt. Faltung) verwendet, welche im nächsten Abschnitt 3.3.1 erklärt werden. Des Weiteren werden rekurrente Schichten insbesondere für temporale Daten, wie Sprache oder Videos, verwendet, da hier Informationen aus vorherigen Eingaben wiederverwendet werden können. Beispiele hierfür sind Long Short-Term Memory (LSTM) [NN13] oder

Gated Recurrent Unit (GRU) [NN16] Schichten.

3.3.1 Convolutional Neural Networks

Als Inspiration für eine optimierte Bildverarbeitung mit neuronalen Netzen diente die Arbeit von *Hubel und Wiesel* [NN3]. In der Arbeit wurde gezeigt, dass es unterschiedliche Topologien in dem Aufbau der Neuronenstrukturen im Gehirn gibt. Besonders in dem visuellen Cortex waren Gruppierungen von Neuronen aufzufinden, die besonders auf Kanten reagierten und diese Signale gebündelt weiterleiteten, um so diese erkannten Muster weiter zu verarbeiten. Das Neocognitron von *Fukushima et al.* [NN4] griff diesen Ansatz auf und stellte mit den „Simple“ und „Complex Cells“ einen Vorläufer der Convolutional und Pooling Layer vor. Bei diesen werden die Ergebnisse in einer Feature-Map (dt. Merkmalskarte) für die folgenden Schichten zwischengespeichert.

Grundsätzliche wird sich bei der Convolution am Konzept des Weight Sharings (dt. Teilen der Gewichte) [NN10] bedient. Konzeptionell werden dabei einzelne Neuronen oder Cluster für mehrere Aufgaben gleichzeitig verwendet, welches einerseits den Vorteil bietet, dass weniger Neuronen verwendet werden müssen und somit die Leistungsanforderungen niedriger sind. Andererseits wird so eine höhere Generalisierbarkeit erreicht, da ein Overfitting (dt. Überangepasstheit) besser umgangen werden kann, weil die Neuronen für verschiedene Aufgaben gleichzeitig verwendet werden. Dies reduziert die Anzahl der Parameter und damit den Suchraum des Trainingsalgorithmus enorm.

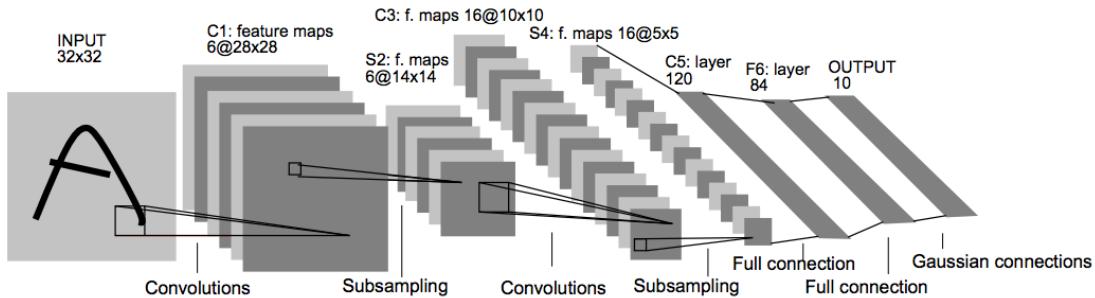


Abbildung 3.5: Das LeNet-5 von *LeCun et al.* [NN12] stellt das erste CNN dar. Es zeigt den Aufbau aus Convolutions, Subsampling und Dense Layern.

Mit dem LeNet5 [NN12], das in Abbildung 3.5 aufgeführt wird, wurde das erste Convolutional Neural Network (CNN) vorgestellt, dessen Aufbau sich bis heute fortsetzt. Fundamental werden hier die Convolutional Layer in Kombination mit Subsampling vorgestellt und im Verbund mit einem Abschluss aus Dense Layern verwendet. In Convolutional Layern werden mehrere Neuronen zu einem $n \times m$ Filter, dem Kernel, zusammengefasst, der mit einer bestimmten Schrittweite, der Stride, über die Eingabe geschoben wird. Im Netzwerk wird die Eingabe als Matrix an die Convolutional Layer übergeben, welche wiederum ihre Ergebnisse in den Featuremaps zwischenspeichern. Diese werden durch Subsampling Layer vor der nächsten Convolution komprimiert. In diesem Subsampling

wird die Auflösung der Featuremaps reduziert, indem nur wieder ein Filter über die Eingabe gelegt wird, der je nach Anwendungszweck den Maximalwert, Minimalwert oder den Durchschnitt jedes betrachteten Ausschnitts speichert. So können die besonders auffälligen Features (dt. Merkmale) hervorgehoben und die unwichtigen Informationen entfernt werden. Dieser Vorgang wird mehrfach wiederholt. Bei dem Erreichen der finalen Dense Layer wird jeder Matrix-Eintrag an jedes Neuron übergeben, um so die Features in den globalen Kontext einzurordnen, zu korrelieren und schließlich zu klassifizieren. Das CNN arbeitet positionsinvariant, da die Filter in den Convolutions für Iteration beständig sind und immer nur den lokalen Kontext auswerten. Somit sind sie nicht vom größeren Kontext abhängig und können sich besser auf einzelne, lokal gebundene Aufgaben fokussieren als Dense Layer.

3.3.2 Lernmethoden

Für das Training von neuronalen Netzen wird grundsätzlich das von *Rumelhart et al.* [NN5] vorgestellte Gradientenabstiegsverfahren mit Backpropagation verwendet. Dieses Verfahren setzt auf die Fehlerwerte einer Loss-Function (dt. Verlustfunktion), die eine Differenz zwischen einer vorliegenden Ground Truth (dt. Grundwahrheit) und einem inferierten Ist-Wert aus dem Modell für jedes beliebige Bild berechnet und eine Anpassung durch alle Netzwerkschichten propagiert. Dementsprechend ist die Datenlage für das Training besonders wichtig. Fehlerbehaftete Daten führen entsprechend zu falsch gelernten Mustern, was wiederum zu einer niedrigen Erkennungsqualität führt. Dieser Lernansatz wird auch als „Supervised Learning“ (dt. Überwachtes Lernen) bezeichnet, da es eine Schüler-Lehrer-Struktur bildet, bei der der Lehrer den Lernvorgang des Schülers überwacht, bewertet und verbessert.

Als Gegenstück dazu fungiert das „Unsupervised Learning“ (dt. Unüberwachtes Lernen), das auf explizite Ground Truth Daten zum Verbessern des Algorithmus verzichtet und stattdessen Metriken zum Bewerten der Fitness (dt. Angepasstheit) der Lösung verwendet. Diese Fitness kann beispielsweise, wie bei k-Means [CV4], eine mittlere Distanz von Instanzen zu einem Clustermittelpunkt ausdrücken und schafft so eine vergleichbare Bewertungsgrundlage. Auf dieser Grundlage können meist keine idealen Lösungen trivial mit analytischen Methoden gefunden werden. Es werden also iterative Verbesserungen, basierend auf einer Heuristik, vorgeschlagen und erprobt, die in das Modell implementiert werden, um die Bewertungsmetrik zu verbessern.

Mit dem Semi-Supervised Learning (dt. Teilüberwachtes Lernen) werden beide Ansätze konzeptionell verbunden. A priori liegt ein kleiner, annotierter und ein großer, nicht annotierter Datensatz vor. Mittels der annotierten Daten wird ein Modell überwacht trainiert. Dieses Training setzt den Startpunkt für die Erweiterung des Datensatzes. Einerseits können die vorliegenden Daten augmentiert, also mit verschiedenen Methoden transformiert werden, um so eine größere Varianz in die Trainingsdaten zu bringen, wie *Tarvainen et al.*

[NN40] vorschlagen. Auf der anderen Seite können die nicht annotierten Daten schrittweise zu den Trainingsdaten hinzugefügt werden, indem verschiedene Klassifikatoren diese mit ihrem unvollständigen Wissen annotieren, wie *Chen et al.* [NN42] vorschlagen. *Athiwaratkun et al.* [NN47] zeigen in ihrer Arbeit, dass beim Annotieren unbekannter Daten eine mehrfache Klassifizierung der Daten gefolgt von einem Mitteln der Ergebnisse die interne Validität der Annotationen erhöht. Zusammenstellungen von Modellen werden auch als Ensembles bezeichnet und in Abschnitt 3.3.3 näher erläutert. Methoden zum Anheben der Qualität der neuen Annotationen, werden von *Xie et al.* [NN62] aufgegriffen, fortgeführt und evaluiert. Augmentationen, die in diesem Ensemble verwendet werden, können die Repräsentation der Features so ändern, dass Teilmodule des Ensembles diese besser extrahieren und klassifizieren können.

Ein ähnlicher Ansatz, der in dieser Arbeit verwendet wird, setzt darauf, iterativ Annotationen für den unbekannten Datensatz zu erzeugen. Es wird ein Ensemble auf den vorliegenden Daten trainiert und eine Inferenz auf die unbekannten Daten ausgeführt. Weisen die Ergebnisse eine hinreichende Konfidenz und eine ausreichende Stabilität auf, werden diese Annotationen als neue Ground Truth Daten gespeichert. Somit wird iterativ ein unbekannter Datensatz für das Supervised Learning vorbereitet.

3.3.3 Ensemblestrukturen

Ensembles bezeichnen die Verknüpfung vieler Modelle, die im Idealfall auf verschiedenen Datensätzen mit der selben Problemstellung optimiert sind, um so die Diversität der Ergebnisse zu erhöhen und eine Überangepasstheit an einen Datensatz auszuschließen, wodurch die Validität der Ergebnisse erhöht wird. Weiterhin können so Konfidenzwerte für die Erkennungen, die über die Konfidenz eines einzelnen Modells hinaus gehen, errechnet werden, wodurch Ausreißer oder Unsicherheit bei der Erkennung leichter auffindbar sind oder sich sogar dann erst zeigen. *Sollich et al.* [2] zeigen, dass grundsätzlich zwei verschiedene Arten von Ensembles konstruiert werden sollten: Einerseits ein Verbund vieler, kleiner Modelle, im Besonderen auch Random Forests, wie sie von *Ho et al.* [4] vorgestellt wurden. Auf der anderen Seite wenige, optimierte Modelle, die auf diversen Datensätzen trainiert wurden.

3.3.4 Transfer Learning

Das Transfer Learning (dt. Übertragendes Lernen) beschreibt eine Methode aus dem maschinellen Lernen, bei dem ein Modell auf einen Datensatz trainiert wird und dieses destrillierte Wissen für andere Zwecke verwendet wird. Verbreitet ist dies besonders für das Pre-Training (dt. Vortraining) von Netzwerken. In der Bildklassifikation ist ein verbreiteter Datensatz ImageNet [DATA1]; analog ist ein verbreiteter Datensatz für die Objekterkennung der COCO-Datensatz [DATA3]. Der Vorteil daran, Netzwerke mit entsprechendem Pre-Training zu verwenden, ist, dass bereits grundsätzliche Filter zur Mustererkennung vorliegen. Es beschleunigt den Trainingsvorgang, da es bereits bei einem funktionalen Startpunkt beginnt und nur noch ein Umtraining auf die gewünschten Klassen im neu-

en Datensatz stattfinden muss. Entsprechend können erste Trainingserfolge bereits nach den ersten Trainingsschritten vorliegen, da kein vollständiges, neues Training von neuen Strukturen stattfinden muss. *Pan et al.* [NN14] erforschten bereits vor der Verbreitung von neuronalen Netzen, wie sich gelernte Metriken eines Datensatzes auf den Trainingsvorgang von Sprachmodellen auf andere Datensätze auswirkt. Die Ergebnisse implizieren, dass in Sprachdatensätzen intrinsische Muster vorliegen, die sich nicht in den Wörtern selbst, sondern in unsichtbaren Mustern auf einer niedergelegenen Repräsentationsebene zeigen. Ähnliche Ergebnisse lassen sich auch für das Training mit Bildern feststellen. Entsprechend muss auf diese Grundlage nur noch ein Fine-Tuning (dt. Feinabstimmung) vorgenommen werden und die Ausgabeschicht auf die gewünschte Struktur angepasst werden.

Shin et al. [NN33] verwenden ein ImageNet Pre-Training für einige CNNs im Bereich der medizinischen Bilderkennung. Sie überprüfen, ob dieses Training Vorteile besonders bei kleinen und weniger genau annotierten Datensätzen, was gerade im medizinischen Bereich häufig vorkommt, bieten kann. Die Modelle wiesen besonders zu Beginn des Trainings bessere Erkennungswerte als ohne ein Pre-Training, da die zufällige Initialisierung der Gewichte mehr Trainingsiterationen benötigen. Insgesamt werden zudem bessere Werte erreicht.

Auch *Tajbakhsh et al.* [NN35] haben sich dem Thema des Transfer Learnings im medizinischen Bereich angenommen. Ihre Ergebnisse verweisen ebenfalls darauf, dass vortrainierte Modelle mit Fine-Tuning auf den Zieldatensatz mindestens genauso gute, meist sogar bessere Ergebnisse erzielen, als Modelle ohne diese Gewichte. Weiterhin ist dieser Ansatz laut ihnen robuster gegenüber zu geringen Datenmengen. Sie finden allerdings, dass in ihrem Ansatz ein Fine-Tuning, bei der jede einzelne Schicht nachtrainiert wird, zum größten Erfolg führt. *Ma et al.* [NN60] hingegen sagen, dass ein Training, das Schicht für Schicht stattfindet und die anderen dabei in ihrem Zustand einfriert, bei einem modernen Deep Neural Networks (DNN) schlechtere Ergebnisse zeigt, als eine dauerhafte, globale Optimierung.

3.4 Deep Neural Networks

DNN bezeichnet den Trend der Forschung, dass neuronale Netzen eine immer größere Tiefe erreichen. Während das LeNet von *LeCun et al.* [NN12] und das AlexNet von *Krizhevsky et al.* [NN15] mit 8 respektive 12 Schichten noch recht flach waren, stießen neuere Architekturen, wie das VGG-net von *Simonyan et al.* [NN26] mit ihren 19 Schichten, an die Grenzen des effektiv Trainierbaren. In einem Paper stellten *He et al.* ihr ResNet [NN29] vor und fokussierten sich dabei auf das Problem des vanishing (dt. verschwindendem) und des exploding (dt. explodierendem) Gradienten. Dies war der Fall, weil Forschern auffiel, dass der Fehlergradient, welcher in der Backpropagation für das Anpassen der Gewichtsparameter im Modell verwendet wird, durch vielfache Multiplika-

tionen nahe Null oder mit großen Werten zu instabil wird, um aus ihm zu lernen, weil die Anpassungen entsprechend gering oder überdimensional werden. Sie stellen residuale Verbindungen zwischen Blöcken von Convolutions her, indem die Identitätsfunktion vom Anfang des Blocks an die Ausgabe des Blocks konkateniert wird, wie es in Abbildung 3.6 links abgebildet ist. Um die Leistungsanforderungen zu reduzieren, werden gleichzeitig sogenannte „Bottlenecks“ (dt. Falschenhalse) eingeführt, welche mittels $[1 \times 1]$ Convolutions die Anzahl der Featuremaps vor dem Konkatenieren reduzieren. Diese Architektur erlaubt es den Autoren, ResNets mit einer Tiefe von 152 Layern zu trainieren [NN29].

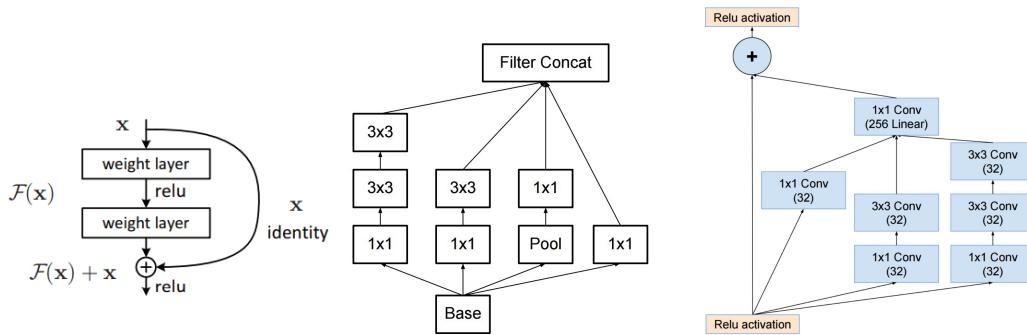


Abbildung 3.6: Ein einzelner ResNet-Block [NN29], Inception-Modul [NN27] und eine Verbindung aus beiden, dem Inception-ResNet [NN39]. (von links nach rechts)

Parallel dazu erforschten Szegedy et al. [NN27] die effizientere Verwendung von Convolutions in einem Netzwerk. Sie gingen dem Problem nach, dass in unterschiedlichen Szenarien unterschiedliche Filtergrößen von Vorteil sein können; große Filter bei großen Mustern, kleine Filter bei kleinen Mustern. Weiterhin wurden CNNs primär in der Tiefe und der Anzahl der Parameter vergrößert, nicht hingegen in der Breite, also durch parallele Pfade und Abläufe. Sie entwickelten die Inception-Module, welche den Netzwerkgraphen innerhalb des Moduls aufspalten, um auf jedem Pfad unterschiedliche Filtergrößen zu verarbeiten, wie in Abbildung 3.6 mittig gezeigt wird. Auch sie verwenden Bottlenecks zur Dimensionsreduktion und konkatenieren die resultierenden Filtermaps jedes Pfads als Ausgabe des Moduls. In weiteren Iterationen des Inception-Moduls [NN34] zeigen Szegedy et al. verschiedene Methoden zur Faktorisierung, also dem Aufspalten größerer Filtergrößen wie $[5 \times 5]$ oder $[7 \times 7]$ in kleinere Filter, auf. Diese können beispielsweise als $[1 \times n]$ und $[n \times 1]$ oder wiederholten $[3 \times 3]$ -Filtern bestehen. Mit der vierten Iteration von Inception [NN39] werden auch die Inception-ResNets vorgestellt, die beide Architekturen vereinen: Wie in Abbildung 3.6 rechts gezeigt wird, können diese Ansätze grundsätzlich leicht verbunden werden, indem neben dem Inception-Modul auch die Identitätsfunktion in die finale Konkatenation übergeben wird.

Getreu dem Geiste der Wissenschaft — auf den Schultern von Giganten — bauen moderne Architekturen meist auf diesen Konzepten auf. Zu diesen Architekturen gehören auch das von Howard et al. vorgestellte MobileNet [NN36] und das You Only Look Once (YOLO)-Net von Redmon et al. [NN32], welches maßgebend für diese Arbeit sein wird.

Diese beiden Architekturen werden nach einer Einführung in die Objekterkennung mittels neuronaler Netze genauer vorgestellt.

3.5 Objekterkennung mit neuronalen Netzen

Mit der Präsentation von CNNs und deren Fähigkeit, Objekte auf Bildern zu klassifizieren, entstand auch eine weitere Anwendungsmöglichkeit. Statt wie bisher mit Hilfe von Computer Vision Algorithmen die Points of Interest (dt. Signifikante Punkte; Sehenswürdigkeiten) herauszuarbeiten und auf die Ähnlichkeit mit bekannten Features zu vergleichen, ergeben sich mit DL neue Möglichkeiten. Grundsätzlich sind dabei zwei Ansätze zu unterscheiden: Die Objekterkennung, welche extern generierte Bounding Boxen iterativ als Klassifikationsproblem betrachtet, und das Detektionsproblem, welches die Bounding Boxen als Regressionsproblem definiert.

Zum ersten Beispiel gehört der Sliding Window Ansatz, bei dem die Eingabe in einzelnen Ausschnitten in ein CNN gegeben wird, um daraus eine Klassifikation zu erhalten. Das Window wird dabei Schritt für Schritt über die Eingabe geschoben, welches den Namen des *Sliding Windows* Ansatzes erklärt. Um unterschiedliche Größen von Objekten zu berücksichtigen, werden auch die Fenster in unterschiedlichen Größen dimensioniert. Da dieser Ansatz abhängig von der gewählten Fenstergröße, des Strides und der Anzahl der Fenstergrößen sehr häufig wiederholt werden muss, ergeben sich hohe Berechnungskosten, denn auch der Klassifizierer muss für jede Erkennung neu berechnet werden. *Sermanet et al.* [NN18] stellten eine Beschleunigung dieses Ansatzes vor, indem sie die Dense Layer, welche für die Klassifikation genutzt werden, ähnlich wie die Convolutions, nicht als Vektor, sondern als 3-dimensionalen Raum betrachten. Sie versuchen somit, mehrere Windows gleichzeitig zu erkennen, da so viele redundante Rechenschritte eingespart werden können.

3.5.1 Region-based Convolutional Network

Mit dem Region-based Convolutional Network (R-CNN) stellen *Girshick et al.* [NN17] eine andere Art der Optimierung für den Lösungsweg als Klassifikationsproblem. Statt eines festen Suchrasters, das mit den Windows abgetastet wird, setzen sie auf eine Segmentation, die auffällige Objekte im Bild finden soll und schlägt diese Regionen als potentielle Bounding Box vor. Diese Bounding Boxen werden in der weiteren Klassifikation, analog zum Fenster des Sliding Windows Ansatzes, verwendet. Die maximale Anzahl an Bounding Boxen, die klassifiziert werden müssen, wird so von mehreren zehntausend auf 2.000 reduziert, wodurch die benötigte Leistung deutlich vermindert wird.

Fast R-CNN stellt eine Weiterentwicklung von *Girshick et al.* [NN19] dar und verzichtet auf einen Segmentationsansatz mit klassischen Algorithmen und stellen stattdessen die Region of Interest (RoI) Layer vor und ändert die Reihenfolge, in der die Schritte angewandt

werden. Die Eingabebilder werden hier als Ganzes betrachtet und von dem Feature Extractor in Form eines CNNs vorverarbeitet. Es behandelt die Bounding Boxen also als Regressionsproblem und gewinnt durch verschiedene Verbesserungen an Performance. Dieses CNN verwendet Pyramid Pooling Layer [NN20], welche auf den Ergebnissen von *Sermanet et al.* [NN18] aufbauen. In den Pyramid Pooling Layern werden durch Aggregation, ähnlich der eines Bag of Words Ansatzes aus dem Natural Language Processing, die Featuremaps unterschiedlicher Größe, die bei Eingabe variabel großer Bilder in das CNN entstehen, auf eine einheitliche Größe komprimiert. Diese Output Featuremap hat dabei die Eigenschaft, die Lokalitäten aus der Eingabe beizubehalten und ist somit als Übergangsschicht zwischen einem CNN Feature Extractor und der Klassifikation in den Dense Layern mit dem Vorteil geeignet, dass nun auch Bilder variabler Größe eingegeben werden können.

Die Featuremap, die aus der Pyramide hervorgeht, wird in dem Fast R-CNN für die Analyse durch die ROI Layer verwendet, um so Bounding Boxen, die klassifiziert werden sollen, herauszuarbeiten. Dieser gesamte Vorgang ist deutlich effizienter berechenbar und sie sorgt für Beschleunigungen in der Trainingszeit um den Faktor 3-9, in der Inferenz um 10-213. Als Nachfolger des Fast R-CNNs wird das Faster R-CNN von *Ren et al.* [NN23] vorgestellt. Sie ändern die Architektur des Modells, indem sie es zweiteilen: Die erste Hälfte inkludiert den Feature Extractor in Form eines CNNs und ein Region Proposal Network, welches Bounding Boxen ungeachtet ihrer Klasse vorschlägt. Die zweite Hälfte besteht aus dem Klassifizierer, welche mit seinen ROI Pooling Layer aus dem Fast R-CNN übernommen wird. Insgesamt erhalten sie damit für sehr geringe Extrakosten eine höhere Genauigkeit. Eine aktuelle Erweiterung, die neben der Objekterkennung auch eine Segmentierung vornimmt, ist das Mask-R-CNN von *He et al.* [NN54].

3.5.2 Single Shot Detector

Der Single Shot MultiBox Detector (SSD) [NN30] stellt nicht wie das Fast R-CNN Versprechen auf, wo ein Objekt liegen könnte, sondern erstellt mit Hilfe eines CNN als Feature Extractor verschieden-dimensionalen Featuremaps, welche jeweils vorgegebene Bounding Boxen in verschiedenen Größen haben. Dies hilft dabei, Objekte verschiedener Größe in der Eingabe zu finden. Für diese Boxen wird jeweils eine Konfidenz berechnet und ein greedy Non-Maximum Suppression Algorithmus durchgeführt, um die redundanten Boxen herauszufiltern. In dem Detektor wird eine genauere Bounding Box, basierend auf der bereits gefundenen, prädiert. Im gleichen Schritt wird für diese Box auch die Klassifikation vorgenommen. Die Architektur wird in Abbildung 3.7 gezeigt und verwendet in der ersten Iteration das VGG-Net [NN26] ohne die Klassifikationslayer als Feature Extractor. Die Autoren weisen darauf hin, dass ihr Modell bessere Benchmarkwerte als YOLO [NN32] erreicht.

Diese Architektur findet sich in dem MobileSSDNet wieder. Das VGG-Net wird durch das

MobileNet [NN36] ersetzt, da es deutlich bessere Performanzeigenschaften aufweisen kann und somit besser für mobile Geräte geeignet ist.

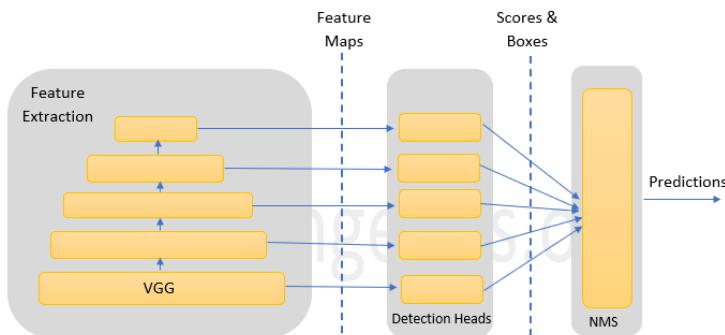


Abbildung 3.7: Architektur des SSD Modells [NN52].

3.5.3 Non-Maximum Suppression

Ein Eigenschaft der meisten Methoden zur Bilderkennung ist, dass die Modelle mehr als eine Bounding Box (dt. Begrenzungsrahmen) für jedes Objekt finden, da diese nicht absolut eindeutig für den Algorithmus sind. Entsprechend muss ein Algorithmus verwendet werden, um eine Bounding Box über die umliegenden Boxen für jedes Objekt zu priorisieren, da sonst Mehrfacherkennungen für die Objekte vorliegen und sich dies schlecht auf die Precision auswirkt. Als Algorithmus kann hierfür die Non-Maximum Supression verwendet werden.

Im ersten Schritt werden die vom Modell gefundenen Bounding Boxen nach ihrer Konfidenz pro Klasse sortiert. Mit dem zweiten Schritt wird die Intersection over Union (Intersection over Union (IoU)), welche im Abschnitt 4.2.3 näher beschrieben wird, dieser Bounding Boxen und der höchsten der jeweiligen Klasse berechnet. Wird dabei ein bestimmter Schwellwert überschritten, wird die Erkennung mit der geringeren Konfidenz verworfen. Ist der Schwellwert eingehalten, wird eine weitere Non-Maximum Supression Analyse für diese Erkennung durchgeführt.

3.6 Die YOLO Architektur

You only look once (YOLO) von Redmon et al. [NN32] stellt als erste Arbeit eine Methode dar, die in einem Schritt des Modells die Feature Extraction, Bounding Box Erstellung und die Klassifizierung dieser auf der gesamten Eingabe durchführt. Sie unterteilen die Eingabe in ein Raster aus $S \times S$ Zellen und ordnen diesen jeweils B trainierbare Bounding Boxen zu. Jede dieser Bounding Boxen wird als Regressionsproblem interpretiert: Sie werden numerisch als Vektor in der Ausgabe prediziert. Die Ausgabe besteht dabei aus der Wahrscheinlichkeit P_c , dass die Bounding Box ein Objekt enthält, dem normalisierten Mittelpunkt der Box b_x, b_y , der Höhe und Weite der Box b_h, b_w und einer Wahrscheinlich-

keit c_n für jedes der n Objekte, welche in dieser Box enthalten sein könnten. Entsprechend ergeben sich für jede Zelle des Rasters $B * (5 + n)$ Einträge, die für die Ausgabe erst eine Non-Maximum Suppression durchlaufen:

$$y = [P_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3, P_c, b_x, \dots]^T$$

Die Architektur selbst ist ein vom GoogLeNet [NN27] inspiriertes CNN, welches durch das spezielle AusgabefORMAT eine feste Größe hat und durch auf die Bounding Boxen abgestimmte Trainingsdaten das Detektionsproblem als Regression behandeln kann. Trotz der deutlich höheren Geschwindigkeit und Genauigkeit für Echtzeitanwendungen, traten noch einige Probleme in der ersten Iteration auf: Aufgrund der Anzahl der vordefinierten Bounding Boxen, die mit 98 im Vergleich zu den 2.000 des R-CNN deutlich spärlicher verteilt sind, kann an die Qualität von langsameren Ansätzen nicht aufgeschlossen werden. Gleichzeitig können die Bounding Boxen sich nicht auf viele verschiedene Seitenverhältnisse und Größen spezialisieren. Auch wird in dem Loss die absolute Distanz als Maß für die Qualität der Bounding Boxen verwendet, was bei unterschiedlicher Größe der Boxen einen großen relativen Unterschied ausmacht [NN64]. Letztlich ist ein Nachteil, dass nur Bilder fester Größe als Eingabe verwendet werden können.

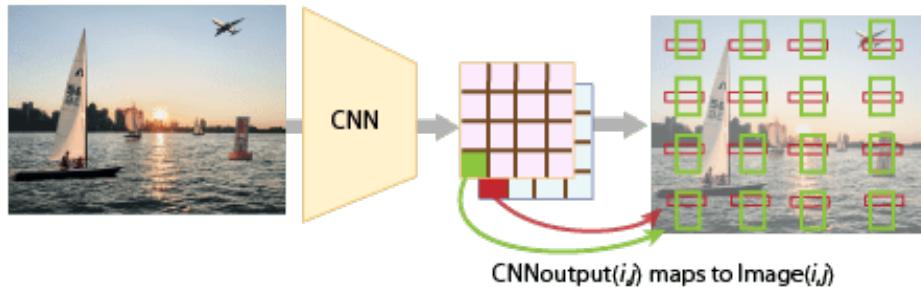


Abbildung 3.8: Erhaltene Featuremaps aus dem CNN Feature Extractor werden mittels definierter Anchors in jeder Rasterzelle analysiert [NN41].

Mit YOLOv2 verwenden *Redmon und Farhadi* [NN38] die Darknet 19 Architektur, welche auf Layer zur Batch-Normalisierung statt Dropout setzt, Anchor Boxen (dt. Anker) statt Dense Layer verwendet und diese Anchor Boxen durch eine spezielle Aktivierungsfunktion lokal an die Rasterzellen bindet [NN46]. Die Anchors sind wie in Abbildung 3.8 gleichmäßig in jedem Zellenraster definiert und in den Anchor Layern werden die x- und y-Verschiebungen und die Weite und Höhe in Relation zum Mittelpunkt des Anchors mit der höchsten IoU gesucht. Diese Werte sind in Abbildung 3.9 links gezeigt. Weiterhin werden mehrere Auflösungen von Featuremaps für die Detektion verwendet, um auch kleine Objekte besser zu lokalisieren und zu klassifizieren. Bilder in unterschiedlichen Größen werden unterstützt, um so eine bessere Verallgemeinerung auf verschiedene Objektgrößen zu finden, wie es in Abbildung 3.9 rechts gezeigt wird. Insgesamt reduziert sich die Geschwindigkeit wegen der zahlreichen Erneuerungen ein wenig, jedoch nimmt die Average Precision mit einem Schwellwert von 50% (AP@.50) von 63.4 % auf 78.6 %

auf dem VOC2007 Datensatz von *Everingham et al.* [DATA2] zu.

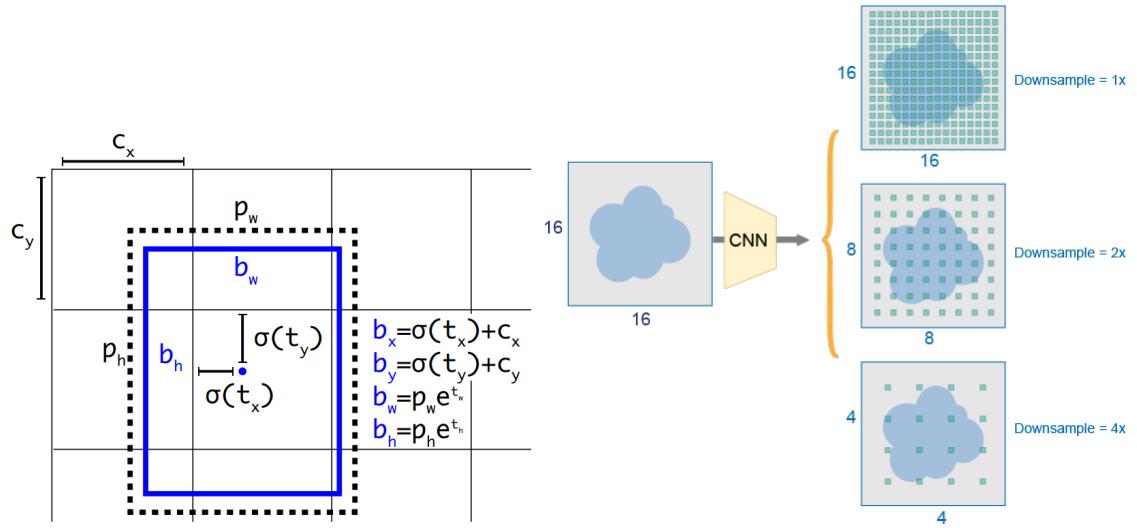


Abbildung 3.9: Links: Lokale Bindung der Bounding Box an den Anchor durch Training des Offsets zu dem Mittelpunkt des Anchors [NN38]. Rechts: Aufteilung der Featuremaps in unterschiedliche Größen, um so Objekte unterschiedlicher Größen besser aufzubereiten [NN41].

YOLOv3 von *Redmon und Farhadi* [NN45] ändert das Feature Extractor CNN vom Darknet 19 zum Darknet 51, um so auf mehr Layern eine größere Varietät von Filtern zu verwenden und verwendet dabei Methoden wie die Inception Module aus *Szegedy et al.* [NN27] und Skip Connections aus *He et al.* [NN29]. Durch das Aufspalten in mehrere Detektionsköpfe, die jeweils auf verschiedenen Auflösungen der Featuremaps arbeiten, wie in Abbildung 3.9 abgebildet ist, werden etwa zehn mal mehr Bounding Boxen detektiert. Weiterhin verwendet YOLOv3 für die Aktivierung der Klassifikationen der Bounding Boxen keine Softmax Aktivierung mehr, welche die Gesamtwahrscheinlichkeit der Erkennung über alle Klassen auf eins normalisiert, sondern setzt unabhängige logistische Klassifizierer ein, was eine mehrdeutige Erkennung in jeder Box zulässt. Insgesamt verliert YOLO durch die neuen Funktionen weiter an Geschwindigkeit, gewinnt dafür an Genauigkeit.

Bochkovskiy et al. [NN53] stellen in ihrer Arbeit die State of the Art Auswahl von Hyperparametern und architektonischen Entscheidungen für das beste Training und die besten Erkennungsraten von CNNs und Detektoren vor. Für ihre Architektur verwenden sie als Feature Extractor eine neue Version des Darknet53, welches um CSP-Verbindungen [NN61] erweitert wurde. Diese Verbindungen konkatenieren die Featuremaps, die sich in jeder Schicht eines Blocks ergeben und leiten diese als heterogenen Verbund von verschiedener Häufigkeit der Faltung, in das weitere Modell zur Analyse. Im Neck, also der Zusammenkunft verschiedener Pfade eines Feature Extractors, auf denen Filter unterschiedlicher Größe verwendet werden, werden Pyramid [NN20] und PAN [NN43] Pooling Layer verwendet, um die Featuremaps zu aggregieren und auf eine fixe Größe zu kom-

primieren. Der Detektor selbst bleibt vom YOLOv3 erhalten. Insgesamt verbessert der überarbeitete Datenfluss die Geschwindigkeit und Genauigkeit deutlich [NN53].

3.6.1 Ultralytics YOLOv5

Mit der fünften Iteration von YOLO knüpft Ultralytics [NN59] an die Arbeit von *Bochkovskiy et al.* [NN53] an und bindet es in das PyTorch Framework [NN37] ein. Es ist eine umstrittene Iteration, da es nicht vom originalen Forschungsteam weitergeführt wurde, die Wurzeln zum Darknet kappt und trotz mehrfacher öffentlicher Nachfrage keine Veröffentlichung zu der Arbeit verfasst wurde [16, 19]. Dementsprechend gibt es kein Peer-Review für ihre Publikationen und es muss auf ihre Aussagen vertraut werden. Da die Ergebnisse allerdings vielversprechend und mit denen des YOLOv4 vergleichbar sind, wird in dieser Arbeit diese Iteration des Netzwerks verwendet, da eine Vielfalt von Erweiterungen in diese Plattform eingebunden sind.

In der initialen Veröffentlichung von YOLOv5 berichten *Glenn Jocher et al.* [NN55] von Genauigkeiten, die den Stand der Technik widerspiegeln bei einer besseren Performanz als noch die vorherigen YOLO Versionen. Mit der ersten Aktualisierung auf Version 2.0 geben sie nur an, dass viele Fehler behoben und einige Funktionen hinzugefügt wurden, welche das Training verbessern [NN56]. Das Update 3.0 stellt weitere Kompatibilität mit Funktionalitäten von NVIDIA Grafikkarten her und erweitert die Benutzerfreundlichkeit um das automatisierte Herunterladen von Modellen, Datensätzen und um Code zum Einfrieren der Schichten, was für den Export benötigt wird [NN57]. Mit Versionen 3.1 und 4.0 werden weitere Komfortfeatures implementiert [NN58, NN65].

Versionen 5.0 und 6.0 stellen besonders die Integration von verschiedenen Services wie Amazon Web Services, Weights and Biases und Roboflow vor. Des Weiteren wird der Export um die Funktionalität erweitert, das Modell in Tensorflow Formaten zu exportieren und es wird eine neue Modellgröße vorgestellt: Nano [NN66, NN67]. In Tabelle 3.6.1 wird dabei eine Übersicht zum Einordnen der Genauigkeitswerte und der Geschwindigkeit der verschiedenen Modellgrößen aufgeführt. Sie beziehen sich in den Werten auf die Genauigkeit der Modelle auf dem COCO Datensatz [DATA3], welcher von Microsoft bereitgestellt wird. Die Rechenschritte werden in Giga Floating Operations (GFLOPs) angegeben, welches die Anzahl der Rechnungen mit Gleitkommazahlen ausdrückt.

Model	mAP.95	AP@50	CPU (ms)	V100 (ms)	Params (M)	GFLOPs@640
Nano	28.4	46.0	45	6.3	1.9	4.5
Small	37.2	56.0	98	6.4	7.2	16.5
Medium	45.2	63.9	224	8.2	21.2	49.0
Large	48.8	67.2	430	10.1	46.5	109.1
Xtra	50.7	68.9	766	12.1	86.7	205.7

Tabelle 3.1: Vergleich der Performanz und Größen der angebotenen YOLO Modelle von Ultralytics [NN59] auf COCO [DATA3].

4

KAPITEL

Methodik

4.1	Datensätze	32
4.1.1	Datensatz-Recherche	33
4.1.2	Eigene Vorverarbeitung mit Unity-Eyes	35
4.2	Metriken	36
4.2.1	Klassifizierung	36
4.2.2	Precision und Recall	36
4.2.3	Intersection over Union	37
4.2.4	Mean Average Precision	38
4.2.5	Konfidenz durch Wiederholung	38
4.3	Hyperparameter	39
4.3.1	Architektur	39
4.3.2	Bildaugmentationen	40
4.3.3	Trainingsfunktionen	40
4.3.4	Trainings-Batches	41
4.3.5	Quantisierung und Pruning	41
4.4	Hyperparameteroptimierung	42
4.4.1	Raster- und zufällige Suche	42
4.4.2	Bayes'sche Suche	43
4.5	Tools	45
4.5.1	Weights and Biases	45
4.5.2	Roboflow	45
4.5.3	Technisches Setup	46

An erster Stelle für das Training eines neuronalen Netzes steht der Datensatz, der in der Lage sein muss das Modell gut auf das zu bearbeitende Problem vorzubereiten. Hinsichtlich der Methoden zum Bewerten eines neuronalen Netzes gibt es die für die Klassifizierung binäre Verfahren. Für die Objekterkennung bedarf es zusätzlich komplexer Metriken um die Genauigkeit der Bounding Box zu bewerten. Des Weiteren werden in diesem Kapitel die Hyperparameter, deren Optimierung und Tools dafür vorgestellt.

4.1 Datensätze

In der Arbeit von *Moryäner* [26] werden drei Datensätze verwendet: Der erste dient zur Erkennung der Augen einer Person, um so die beiden Augen zu lokalisieren, die relevanten Bildausschnitte zu fokussieren und für die weitere Erkennung vorzubereiten. Der zweite behandelt die Iriserkennung, mithilfe derer der Mittelpunkt des Auges gefunden werden soll, um so einen Anhaltspunkt für Heuristiken zur Optimierung weiterer Verfahren zu schaffen. Im dritten Datensatz werden dieselben Bilder wie im zweiten verwendet, jedoch ist hier die Pupille, nicht die Iris annotiert. Es sollen damit Bounding Boxen für die Pupillen gefunden werden. Wie in Abschnitt 3.1.3 beschrieben, wird der Ansatz zur Pupillenerkennung geändert, indem nur ein Auge auf einmal betrachtet wird. Zudem wird die Iris- und die Pupillenerkennung in einem Schritt durchgeführt, weshalb ein passender Datensatz beide Objekte beinhalten muss.

Als erster Datensatz wurde der FFHQ-Datensatz von NVIDIA verwendet, der auf Bildern von Flickr aufbaut [DATA8]. Dieser Datensatz ist mit rund 12.000 Trainings- und 500 Validierungsdaten ausreichend umfangreich und bildet ein weites Spektrum von möglichen Eingaben ab, weswegen er für die vorherige Arbeit verwendet wurde. Er wird in dieser Arbeit besprochen und analysiert, da eine künftige Änderung der Implementation diesen wieder benötigen könnte; für diese Arbeit ist er aber nicht von Bedeutung.

Für das Vermessen der Iris und Pupille wird ein weiterer Datensatz benötigt. Dafür wurde in der vorherigen Arbeit eine Mischung aus einem Datensatz von www.flickr.com und Syntheyes [DATA4] verwendet. Die 206 respektive 196 Bilder von Pupille und Iris von Flickr wurden per Hand um die Annotationen erweitert und auf eine Auflösung von 320x320 Pixel herunterskaliert.



Abbildung 4.1: Beispiele aus dem Syntheyes Datensatz [DATA4].

Der Syntheyes Datensatz hingegen ist mithilfe eines 3D-Modells vom menschlichen Auge durch *Wood et al.* [DATA4] erstellt worden; Beispiele sind in Abbildung 4.1 gezeigt und stellen mit insgesamt 460 Bildern einen größeren Teil des Datensatzes dar. Dieses Modell wurde von *Wood et al.* an der Cambridge Universität für die Generierung möglichst realistischer, menschlicher Augen erstellt, um so auch eine hohe Diversität des Alters, Blickwinkels und Lichteinfalls zu gewährleisten. Wie *Moryäner* in seiner Arbeit allerdings bereits anmerkt, ist die Auflösung der Bilder mit 80x120 Pixel sehr gering und musste aus Kompatibilitätsgründen für das Training mit dem SSDLite MobileNetV3 auf eine Auflösung von 320x320 Pixeln gebracht werden. In einem ersten Schritt wurden die

Bilder um den Faktor 2.4 auf eine Größe von 192x288 Pixel skaliert. Für den zweiten Schritt wurden sie auf einem weißen Hintergrund platziert, der keinen Einfluss auf das Training nehmen sollte.

Da in dieser Arbeit eine möglichst hohe Genauigkeit in den Annotationen unabdingbar ist, um mit dem finalen Modell eine präzise Erkennung zu trainieren, muss ein besser geeigneter Datensatz gefunden werden. Die Iris hat auf verschiedenen Bildern in dem Syntheseyes Datensatz eine Breite von etwa von 80 Pixeln, ohne Skalierung folglich eine Höhe von $\frac{80}{2.4} = 33.3$ Pixel. Entsprechend ergibt sich also bei einer durchschnittlichen Irisgröße von 12mm eine Auflösung von $\frac{12}{33.3} = 0.36$ mm/Pixel in den originalen Daten. Durch die Skalierung werden Ungenauigkeiten in den Annotationen verstärkt. Als Ziel sollte aber eine möglichst geringe Ungenauigkeit erreicht werden, was nur durch höher auflösende Daten geschehen kann. Der Anteil der Daten von Flickr erfüllt durch die Auflösung von 320x320 Pixel diese höheren Auflagen. Eine Irisbreite von 200-260 Pixel und keine Skalierung resultiert in einer Auflösung von $\frac{12}{230} = 0.05$ mm/Pixel, also einer Verbesserung um den Faktor 7. Da hier die Daten im Rahmen der vorherigen Arbeit per Hand von einer einzelnen Person annotiert wurden und keine Verfahren zur Fehlerreduktion angewandt worden sind, muss die Präzision hinterfragt werden. Weiterhin bezieht sich der Datensatz auf nur knapp 200 Bilder, was im Bereich des maschinellen Lernens eine starke Einschränkung darstellt.

Im Folgenden werden verschiedene Datensätze, die im Rahmen der Recherche gefunden wurden, und deren Problematiken vorgestellt.

4.1.1 Datensatz-Recherche

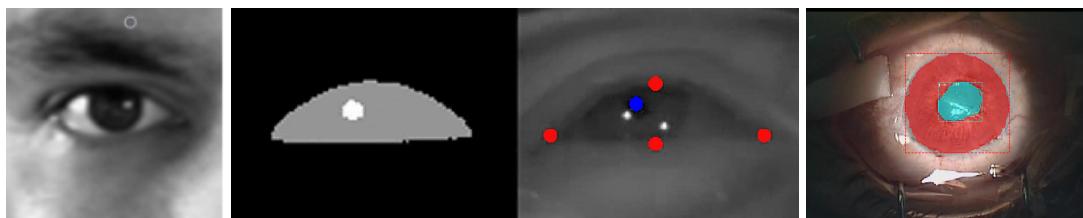


Abbildung 4.2: Beispiele von links nach rechts aus den Datensätzen von *Fusek* [DATA6], *Fuhl et al.* [DATA7] und *Sokolova et al.* [DATA10].

In dem Datensatz von *Fusek* [DATA6] werden die Daten mit einer Infrarotkamera während einer Autofahrt aufgenommen. Die ca. 85.000 Bilder sind nur in Graustufen verfügbar und haben eine geringe Auflösung im Bereich der Augenpartien. Dem Forscher war bei der Aufnahme des Datensatzes besonders die Blickrichtung und das Zwinker-Muster sowie einige Stammdaten des jeweiligen Fahrers und Umgebungsdaten wichtig. Neben der puren Bilder, wovon ein Beispiel in Abbildung 4.2 links gezeigt wird, sind hier folglich keine relevanten Informationen für diese Arbeit vorhanden.

Der Datensatz von *Fuhl et al.* [DATA7] der Universität Tübingen, Abbildung 4.2 mittig,

bringt zwar eine genaue Vermessung der Pupille mit, jedoch steht der Datensatz auf der eigenen Website nicht mehr zur Verfügung. Zudem handelt es sich bei den Bildern ausschließlich um Schwarz-Weiß-Aufnahmen, was eine Einschränkung für diese Arbeit darstellt, da mit dem Smartphone eine Farbkamera gegeben ist, die möglicherweise mehr nützliche Informationen für die AI bereitstellen kann. Weiterhin müssten die Bilder für einen Nutzen in dieser Arbeit vorverarbeitet werden, um aus den Segmentationsdaten die jeweilige Bounding Box zu extrahieren.

Der Datensatz von *Sokolova et al.* [DATA10], in Abbildung 4.2 rechts, besteht aus Fotos, die während Operationen an Augen entstanden sind. Die Auflösung und die Annotationen sind qualitativ sehr gut, jedoch entsprechen diese Daten nicht dem Anwendungszweck, da verletzte Augen mitsamt Operationswerkzeug und Abdeckungen abgebildet sind. Ein Vorteil jedoch sind die bereits vorgegebenen Bounding Box auf jedem Bild.

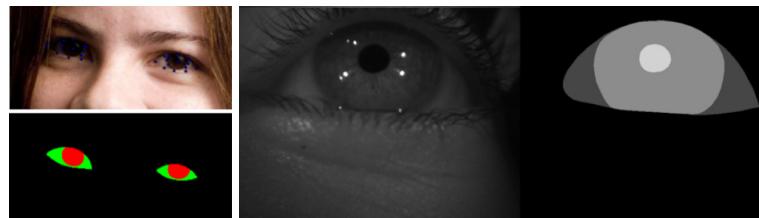


Abbildung 4.3: Beispiele von links nach rechts aus den Datensätzen von *Luo et al.* [DATA9] und *Wu et al.* [G7].

Auf der Suche nach passenden Datensätzen für die Objekterkennung sind auch interessante Datensätze besonders im Bereich der Gaze Estimation für die Segmentation dieser Objekte aufgefallen. Der Datensatz von *Luo et al.* [DATA9], abgebildet in Abbildung 4.3 links, segmentiert die Iris und das Auge, genauer die Lederhaut des Auges. Er ist mit 4461 Bildern von Gesichtspartien und 8882 einzelnen, segmentierten Augen, recht umfangreich und bildet eine große Varietät in der Auflösung, der Beleuchtung und der Positionierung des Kopfes ab. Fehlende für diese Arbeit ist hier eine Segmentation der Pupille.

Der Segmentationsdatenatz von *Wu et al.* [G7] stellt mit insgesamt 80.000 per Hand annotierten Bildern von 587 Probanden einen sehr umfangreichen Datensatz dar. Ein Beispiel wird in Abbildung 4.3 gezeigt. Problematisch ist hier allerdings, dass diese Daten nur in Graustufen vorliegen. Mit diesem Datensatz könnten potentiell Arbeiten in dem Bereich der Augensegmentierung oder der Gaze Estimation durchgeführt werden, wenn Bilder in Graustufen ausreichend sind.

4.1.2 Eigene Vorverarbeitung mit Unity-Eyes

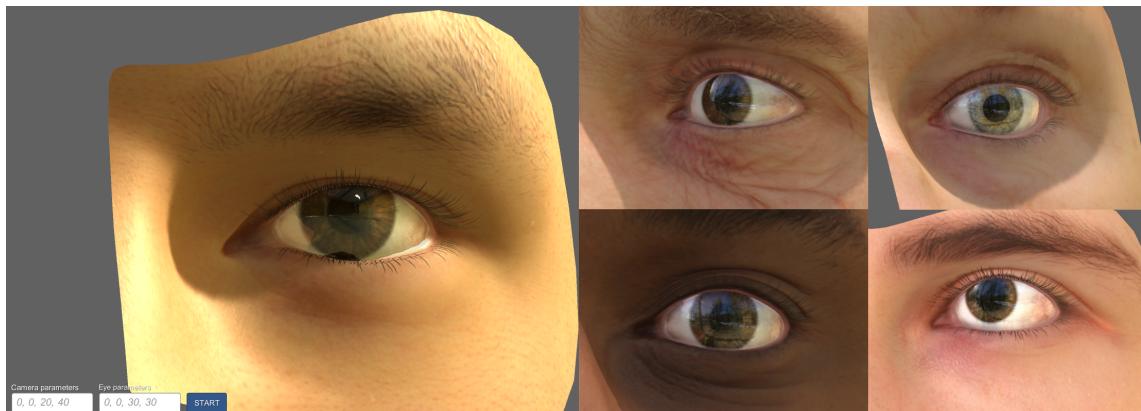


Abbildung 4.4: Das Interface von Unity-Eyes [DATA5] und eine Auswahl von Bildern, die von dem Tool generiert wurden.

Da alle vorliegenden Datensätze über einen kleinen Umfang, geringe Auflösung, Probleme mit den Annotationen verfügen oder einschränkend im Farbraum sind, muss ein anderer Ansatz gefunden werden. Unity Eye 3D [DATA5] stellt eine Weiterentwicklung des Modells, das für die Generierung vom Syntheseyes-Datensatz gedient hat, dar. Es liegt als Implementierung im Unity-Framework vor und bietet dem Anwender verschiedene Einstellungsmöglichkeiten, um ein breites Spektrum von Bildern zu generieren und so eine hohe Diversität im Datensatz zu erhalten, wie in Abbildung 4.4 zu sehen ist. Diese Diversität umfasst das Alter (nicht angegeben), den Hauttypen in einigen Abstufungen (fair 1-6, medium 1-15), die Irisfarbe (5 Farben) und den Blickwinkel mit samt der Kopfdrehung. Vor Start des Programms werden Rahmenbedingungen erfragt, wie die Auflösung und die Qualität des zu rendernden Augenmodells. Für die Qualität können mehrere Werte zwischen „Fastest“ und „Fantastic“ gewählt werden, wobei diese sich besonders in den Reflektionen aus der Umgebung und den Details von Haut und Haaren widerspiegeln. Die Reflektionen werden durch Umgebungsassets samt eigener Lichtquelle realisiert. Dabei entstehen auch automatisch die Reflektionen und weißen Lichtpunkte, welche in Moryäners Arbeit als Augmentation versucht wurden umzusetzen. In Abbildung 4.4 ist unten links eine Auswahl für das Intervall, in dem die Parameter der Kameraposition und dem Blickwinkel liegen sollen. Mit Auslösen des „Start“ Buttons werden die einzelnen Bilder gerendert und zusammen mit ihrer Annotation als .json Dateien in einem Unterordner von der ausführbaren Datei gespeichert.

4.2 Metriken

Eine Performanz-Metrik bezeichnet in der Mathematik eine Abstandsfunktion, die Elemente in einem metrischen Raum nach ihrer Distanz bewertet. Sie führt die Genauigkeit in der Klassifikation und in der Objekterkennung mit vergleichbaren Werten auf und hilft so zu bestimmen, wie gut ein Modell für eine spezielle Aufgabe geeignet ist und wo dessen Schwächen liegen.

4.2.1 Klassifizierung

In einer binären Klassifizierung kann zwischen vier Fällen unterschieden werden, die eintreten können. Allesamt beziehen sich dabei auf das Verhältnis zwischen der durch das Modell erkannten Klasse und der eigentlichen Klasse, die erkannt werden sollte. In Abbildung 4.5 sind diese dargestellt. Sie sind wie folgt definiert:

- True Positive (TP): Beschreibt die Instanzen, welche als zugehörig zur Klasse erkannt wurden und auch in der Ground Truth dieser Klasse zugeordnet sind.
- False Positive (FP): Beschreibt die Instanzen, welche fälschlicherweise nicht der Klasse zugeordnet wurden.
- True Negative (TN): Beschreibt die Instanzen, die richtig als nicht zugehörig zur Klasse identifiziert wurden.
- False Negative (FN): Beschreibt die Instanzen, die fälschlicherweise als nicht zugehörig klassifiziert wurden.

4.2.2 Precision und Recall

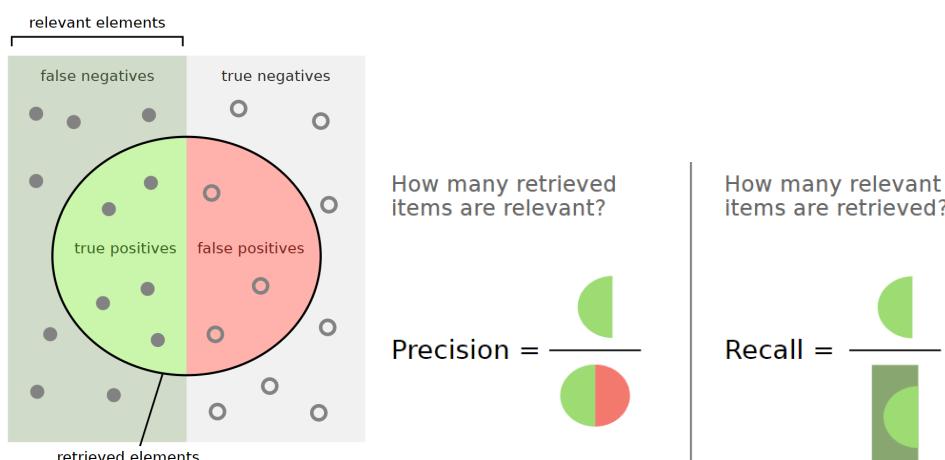


Abbildung 4.5: Visualisierung von Precision und Recall innerhalb einer Menge von Klassifizierungen. (geändert nach [10])

Eine fundamentale Metrik zum Bestimmen der Klassifikationsgenauigkeit eines neuronalen Netzes ist die Anzahl der Klassifikationen von Instanzen im Verhältnis zur Korrektheit

ebendieser. Da die meisten Modelle einen kontinuierlichen Wert hinsichtlich ihrer Konfidenz der Erkennung ausgeben, lässt sich ein beliebiger Schwellwert für das Erkennen oder Ablehnen festlegen.

Die Precision (dt. Präzision) beschreibt, wie genau der Klassifikator arbeitet, indem die die Menge der korrekt positiv gefundenen Instanzen in Verhältnis zur gesamten Anzahl der positiv bewerteten Instanzen setzt. Es ergibt sich also:

$$\text{Precision} = \frac{TP}{P_{predicted}} = \frac{TP}{TP + FP}$$

Der Recall hingegen befasst sich mit der Menge der relevanten Instanzen, die gefunden wurden. Er vergleicht, wie viele der zur Klasse zugehörigen Instanzen auch als solche identifiziert wurden. Dies entspricht:

$$\text{Recall} = \frac{TP}{P} = \frac{TP}{TP + FN}$$

In Abbildung 4.5 wird gezeigt, wie die beiden Werte sich zueinander verhalten. Der Recall verbessert sich grundsätzlich, wenn der Schwellwert für die positive Erkennung abgesenkt wird, da so mehr Elemente gefunden werden. Er verbessert sich aber nur solange, wie noch FN vorhanden sind. Entgegengesetzt verbessert sich der Wert der Precision weiter, wenn der Schwellwert erhöht wird, solange es noch FPs gibt. Je nach Anwendungsfall müssen somit diese beiden Metriken miteinander abgewogen werden.

Um ein Optimum beider Metriken zu finden kann ein harmonisches Mittel beider Werte gebildet werden. Dieser Wert wird im Allgemeinen als $F_\beta - Score$ (dt. F-Maß) bezeichnet, wobei das β für das Verhältnis zwischen Precision und Recall steht. Die allgemeine Funktion wird wie folgt beschrieben:

$$F_\beta = (1 + \beta^2) * \frac{\text{precision} * \text{recall}}{(\beta^2 * \text{precision}) + \text{recall}}$$

Soll das Verhältnis beider Werte gleich sein, so ergibt sich:

$$F_1 = (1 + 1^2) * \frac{\text{precision} * \text{recall}}{(1^2 * \text{precision}) + \text{recall}} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

4.2.3 Intersection over Union

Die IoU beschreibt das Verhältnis von der inferierten Bounding Box zur Ziel Bounding Box. Sie gibt prozentual an, wie weit sich die beiden Boxen überschneiden. Wie in Grafik 4.6 zu sehen ist, weisen kleine Werte auf wenig Überschneidung hin und implizieren somit eine ungenaue Bounding Box, da diese auch mit einer rein zufälligen Platzierung mit hoher Wahrscheinlichkeit Teile des Objektes trifft. Dementsprechend muss auch hier ein Schwellwert gefunden werden, der festgelegt, ob ein Objekt gefunden wurde oder nicht. Liegt der Schwellwert zu niedrig, wird eine geringe Präzision erzielt. Liegt er zu hoch, können bereits kleine Ungenauigkeiten in den Annotationen zu einem schlechten Recall

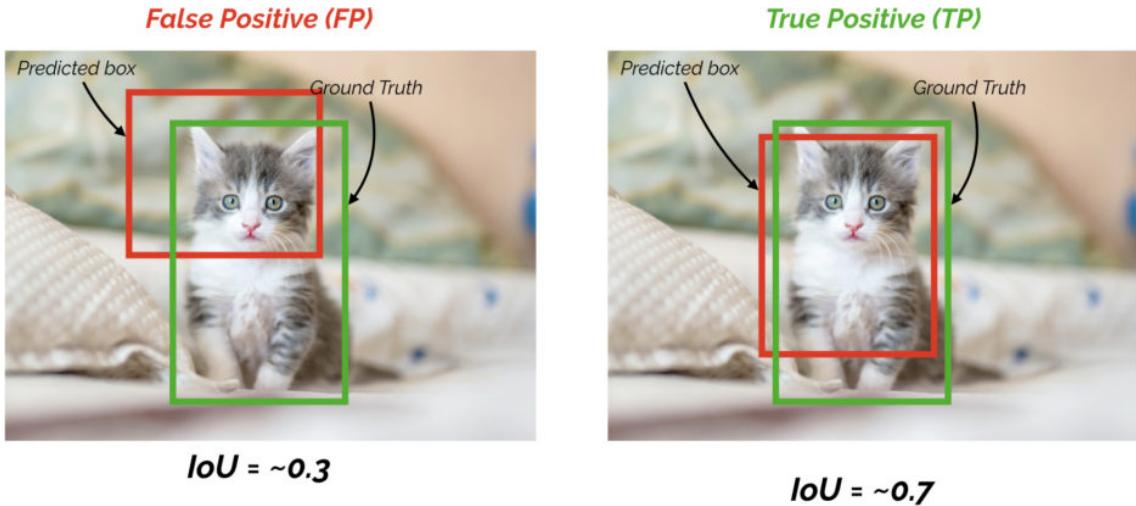


Abbildung 4.6: Die IoU von verschiedenen inferierten Bounding Boxen in Relation zur Ground Truth [NN69].

führen. In der Literatur werden einerseits als feste Schwellwerte 50 % und 75 % IoU vorgeschlagen. Andererseits eine progressive Skala, die einen Mittelwert über mehrere Schwellwerte bilden soll. Diese werden im nächsten Abschnitt 4.2.4 genauer erläutert.

4.2.4 Mean Average Precision

Mit mean Average Precision (mAP) wird die durchschnittliche Precision aller Klassen in der Objekterkennung bezeichnet. Sie beschreibt die Fläche unter der Precision-Recall Kurve für beliebige Schwellwerte der IoU. Die AP@.50 und die Average Precision mit einem Schwellwert von 75% (AP@.75) stellen nur eine punktuelle Betrachtung dar. Dahingegen soll die mean Average Precision zwischen 0.5 und 0.95 mit Schrittgröße 0.05 (mAP.95) (oder auch mAP:50:.05:.95) die Schwellwerte von 50 % bis 95 % in 5 % Schritten abbilden. Es ergibt sich also folgende Funktion für die Berechnung der mAP.95:

$$mAP.95 = \frac{1}{10} \sum_{n=0}^9 Precision_{IoU=50\%+n\times5\%}(Classifier)$$

Analog zur AP@.50 gibt es auch den mean Average Recall (mAR), welcher in dieser Arbeit allerdings nicht angewandt wird, da die Analyse der Präzision als wichtiger erachtet wird, als die des Recalls. Dies begründet sich darin, dass nur eine möglichst geringe Differenz zwischen der Inferenz und der Realität liegen soll. Die Vermutung liegt nahe, dass fehlende Werte besser durch eine Interpolation ersetzt werden können, als Fehlerkennungen zu filtern, die einen falschen Wert implizieren.

4.2.5 Konfidenz durch Wiederholung

Ein großer Kritikpunkt an dem gesamten Bereich des DLs ist, dass es Probleme in der Reproduzierbarkeit und der Nachvollziehbarkeit der Ergebnisse gibt. Es besteht die begründete Vermutung, dass die Ergebnisse neuer Methoden auch durch den Zufall be-

einflusst werden. Meist unterliegen Forscher Restriktionen in der verfügbaren Leistung und dem Budget, was sie davon abhält, Experimente zufriedenstellend häufig zu wiederholen und so eine Konfidenz zu bilden. So zeigt *David Picard* [NN68], dass auch ein verbreitetes und recht aktuelles ResNet-50 Modell [NN29] auf dem bekannten ImageNet Datensatz [DATA1] dem Zufall Opfer fällt. Allein die Wahl des Seeds, also dem Startwert eines Zufallsprozesses, der für das Training verwendet wird, kann Unterschiede in der Klassifikationsqualität von 0.5% ausmachen, was in dem Benchmark als signifikanter Unterschied gilt. Diese Differenz wurde beim Testen von nur 50 unterschiedlichen Seeds erreicht und zeigt, dass ein Training von Modellen in einer nicht vernachlässigbaren Größenordnung vom Zufall abhängt. Experimente sollten somit nur mit einer Konfidenz oder dem gesamten abgedeckten Bereich angegeben werden, damit kein zufälliges Cherry Picking (dt. Rosinenpickerei) betrieben wird.

Auf der anderen Seite muss berücksichtigt werden, dass aufgrund der Einschränkungen in Rechenzeit und Abgabefristen die Experimente nicht beliebig häufig wiederholt werden können. In dieser Arbeit werden die Experimente, sofern sie quantitativ ausgewertet werden, mindestens fünf mal wiederholt und mit den jeweiligen minimalen und maximalen erreichten Werten nebst dem mittleren Wert abgebildet.

4.3 Hyperparameter

Unter Hyperparameter werden im Allgemeinen Eigenschaften und einstellbare Werte eines Modells zusammengefasst, die vor dem eigentlichen Training angepasst werden. Sie geben diesem Training also die Rahmenbedingungen vor. Dabei gibt es unterschiedliche Parameter, die den Aufbau eines Netzes, die Vorverarbeitung der Eingabedaten oder auch das Verhalten des Optimierers während des Trainings beeinflussen. Zudem können durch ein Anpassen der Auflösung von Variablen und der Anzahl der Parameter besonders die Performanz-Metriken verbessert werden.

4.3.1 Architektur

Die Architektur des Modells umfasst viele Designentscheidungen, die von groben, fundamentalen Änderungen bis hin zu detaillierten Anpassungen reichen können. Fundamental kann die Entscheidung zwischen ausgefeilten Architekturen, die durch Features in der Vorverarbeitung und Auswertung an komplexe Frameworks erinnern, wie dem YOLONet [NN59] und dem MobileSSDNet [NN50] stehen. Diese Architektur bestimmt dabei über den Aufbau aus einzelnen Modulen, wie den Inception-Modulen oder den ResNet-Verknüpfungen, oder über die generelle Philosophie beim Design des Netzes. Kleinschrittiger ist die Anzahl der Layer und die Wiederholung dieser beim Aufbauen eines tiefen neuronalen Netzes. Angepasst werden kann auch die Anzahl der Filter und die Größe der Kernel. Letztlich können auch die Aktivierungsfunktionen, also die Funktionen, welche die Ausgaben der Neuronen transformieren, angepasst werden oder final auch die Wahrscheinlichkeitsverteilung, aus der die Initialisierung der Parameter gezogen wird.

4.3.2 Bildaugmentationen

Mit Augmentationen der Trainingsbilder wird auf der einen Seite häufig versucht, den Datensatz künstlich zu vergrößern, ohne weitere Daten aufnehmen und annotieren zu müssen. Auf der anderen Seite können sie mögliche Einblicke in eine effizientere Repräsentation der Daten darstellen [NN51]. In beiden Fällen erhöhen sie die Diversität der Daten, können aber unter Umständen die Integrität des Datensatzes gefährden, wenn unrealistische Motive entstehen. Bildaugmentationen können dabei Änderungen am Inhalt des Bildes vornehmen, wie es bei Änderungen des Kontrasts, der Helligkeit oder des Wertes einzelner Farbwerte passiert. Affine Transformationen wie die Spiegelung oder Rotation sind dabei auch eine Option. Weiterhin können Filter, die ein Rauschen herbeiführen oder reduzieren sollen, angewandt werden, um eine Überangepasstheit an bereinigte Daten zu vermeiden und um mehr Dynamik in den Datensatz einzufügen. Andere Augmentationen, die an Bildern vorgenommen werden können, sind die Skalierung, der Algorithmus, welcher dafür verwendet wird, oder das Verwenden eines Scherwinkels, mit dem das Bild geschert wird. Je nach Anwendungszweck können auch einzelne Bereiche eines Bildes gelöscht oder überschrieben werden oder neue Elemente eingebunden werden, wie es von *Moryäner* [26] mit konvexen Polygonen zur Imitation eines Lichtblitzes getan hat. Eine weitere Augmentation stellt *Ultralytics* mit ihrer Mosaic-Bildung dar, welche im Training für das YOLONet verwendet wird. Diese und weitere Augmentationen werden beispielhaft in Abbildung 4.7 wiedergegeben.

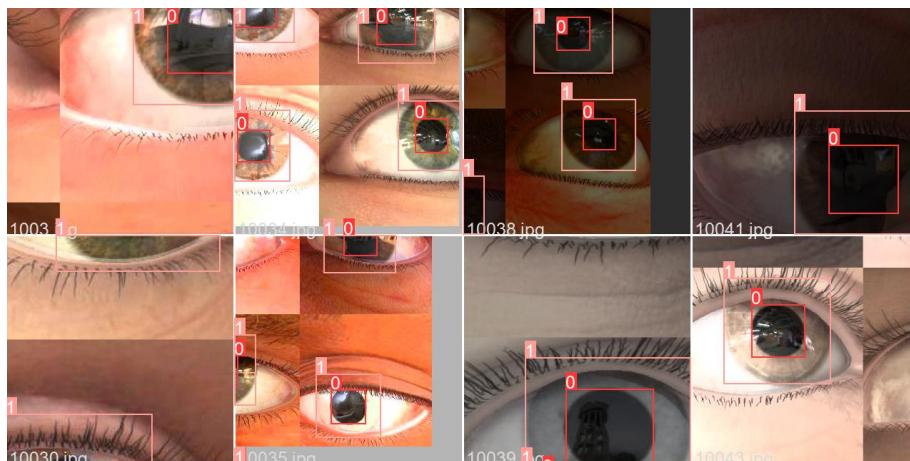


Abbildung 4.7: Verschiedene Augmentationen, die mit dem YOLOv5 Framework auf den Unity Datensatz angewandt wurden.

4.3.3 Trainingsfunktionen

Verschiedene Optimierungsfunktionen können bei unterschiedlichen Architekturen und Datensätzen unterschiedliche Trainingsverläufe vorweisen. Es besteht eine Auswahl aus vielen Optimierern, die eigene Verfahren zur Kostenoptimierung einsetzen, wie dem einfachen Gradientenabstieg (GD), dem stochastischen Gradientenabstieg (SGD), der Adaptiven Schwung Einschätzung (Adam) [NN21] oder weiteren, hier nicht aufgeführten Optimierungsalgorithmen [NN48]. Weiterhin kann vor dem Training auch eine Lernrate

angegeben werden, welche während des Trainings vom Optimierer angepasst wird, um einem globalen Minimum auf der Fehleroberfläche der Kostenfunktion möglichst nahe zu kommen. Diese Lernrate kann in mehrere Phasen eingeteilt sein und beeinflusst entsprechend die Trainingsgeschwindigkeit.

4.3.4 Trainings-Batches

Entscheidend für die Eingabe neuer Daten ist auch die Form dieser Daten. Während die Eingabe von einzelnen Bildern möglich ist, ist sie meist sehr langsam, da für jedes Bild ein vollständiger Backpropagation Vorgang durchgeführt werden muss. Um eine bessere Parallelisierung und damit eine bessere Geschwindigkeit im Training zu erreichen oder den Mittelwerten mehrerer Kostenfunktionen zu berechnen, um eine stabilere Konvergenz zu erreichen, können auch viele Einzelbilder zu einem Batch zusammengefasst werden. *Masters und Luschi* [NN44] stellen hierzu eine Arbeit vor, in der sie zeigen, dass kleinere Batches zu einer besseren Generalisierbarkeit des Modells führen können. Gleichzeitig sorgen sie während des Trainings für einen geringeren Speicherverbrauch, wobei allerdings die Trainingszeit erhöht wird. Sie evaluieren die Verwendung von Mini-Batches in Kombination mit einer Batchnormalisierung, was die Eingabewerte skaliert und standardisiert, um die Repräsentation der Daten zu entzerren. Diese Normalisierungen können auch innerhalb des Modells passieren, erzielen aber laut *Masters und Luschi* besonders bei größeren Batches ein besseres Ergebnis.

4.3.5 Quantisierung und Pruning

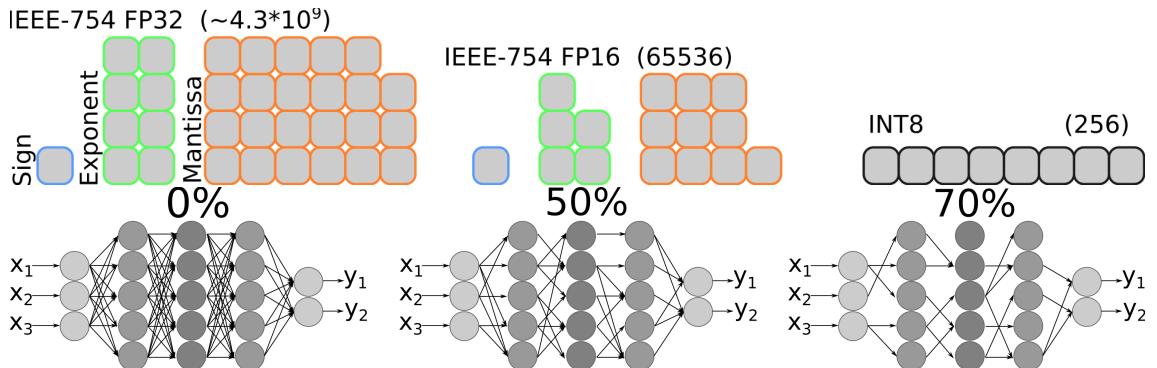


Abbildung 4.8: Quantisierung von Float32, Float16 und Int8 und eine Anwendung vom Pruning mit 50 % und 70 %.

Bei einer Quantisierung wird die Auflösung der verwendeten Variablen für Parameter reduziert, sodass die Recheneinheiten besser ausgenutzt werden. Single und Half Precision (dt. einfache und halbe Genauigkeit) sind nach IEEE-754 Standard [13] definiert und werden auf Abbildung 4.8 mit dem INT8 Format verglichen. Während komplexe Systeme mit möglichst hoher Genauigkeit rechnen müssen, sind für neuronale Netze auch geringe Genauigkeiten ausreichend und können durch eine effizientere Auslastung der Recheneinheiten Zeit sparen. Durch das Mixed Precision Training kann durch temporär

höher auflösende Variablen der Gradient besser durch das Netz propagiert werden, um das Trainingsverhalten zu verbessern [HPO12].

Pruning ist eine andere Methode zur Effizienzsteigerung, die unwichtige Gewichte permanent aus dem Netz entfernt. Dabei wird das gelernte Wissen destilliert und der Rechenaufwand reduziert. Eine Kombination dieser beiden Techniken hilft *Han et al.* [NN28] bei einer Reduktion der Modellgröße vom AlexNet [NN15] und VGG [NN26] um 96-97 % ohne nennenswerte Verluste in der Vorhersagequalität auf dem ImageNet [DATA1]. *Jacob et al.* [HPO11] verwenden sie für ihre Quantisierung Integer und zeigen, dass auch 8-Bit Variablen als Gewichte für die Neuronen verwendet werden können, ohne dass ein signifikanter Verlust in der Klassifikationsqualität folgen muss. *Wu et al.* [HPO9] zeigen, dass diese Quantisierung auch für die Verwendung von neuronalen Netzen auf Smartphones zu tragen kommt. Diese Erkenntnis wird in dem MobileNet von *Howard et al.* [NN36] angewandt, um eine ideale Performanz auf mobilen Geräten zu erhalten.

4.4 Hyperparameteroptimierung

Für die Hyperparameteroptimierung gibt es verschiedene Verfahren zur Suche der passenden Werte. Wie bereits in Kapitel 2 vorgestellt, wurden Racing-Algorithmen vorgestellt, welche Modelle mit verschiedenen Konfigurationen auf verschiedene Probleme gegeneinander antreten lassen, um so die besten Werte zu bestimmen. Bei diesem klassischen Verfahren werden Algorithmen verglichen, bei denen es bis auf die geänderten Hyperparameter und einem geänderten Seed bei Nichtdeterminismus meist zu keinen weiteren Änderungen kommt. Dadurch ist der Suchraum trotz der kontinuierlichen Parameter in einer kleineren Größenordnung, als es bei neuronalen Netzen der Fall ist. Neuronale Netze bieten einen beliebig großen Suchraum, da neben den Hyperparametern, die auch die Architektur einschließen können, auch Parameter im Millionenbereich, welche sich in keinem Training zufällig gleich verhalten werden. Ergo muss ein Modell zur Analyse des Einflusses der Hyperparameter auf die Trainingsmetriken erstellt und genutzt werden, um eine effektive Optimierung durchzuführen.

4.4.1 Raster- und zufällige Suche

In der Rastersuche gibt der Forscher für die verschiedenen Parameter jeweils diskrete Werte vor, die von dem Algorithmus durchprobiert werden. *Bergstra und Bengio* [HPO7] zeigen, dass es, ähnlich einer manuellen Suche, einer Hybris des Forschers bedarf, um a priori zu wissen, welcher Wertebereich der Parameter ideal für ein optimiertes Verhalten ist. Auch mit einer vorherigen Testreihe, die diesen Wertebereich ohne Zutun des Forschers einschränkt, ist es aufgrund der fehlenden Variabilität schlecht geeignet, einen hoch-dimensionalen Raum zu analysieren. Sie zeigen, dass eine zufällige Suche deutlich effizienter den Suchraum nach erfolgreichen Konfigurationen erforschen kann, wie

in der Abbildung 4.9 gezeigt wird. Dabei verwenden sie die Begründung, dass nicht alle Hyperparameter einen gleich starken Einfluss auf die Qualität des Modells nehmen und dieser Einfluss sich je nach Datensatz ändern kann. Somit verwendet die Rastersuche unweigerlich Zeit auf Parameter, welche einen geringen Einfluss haben.

In ihrer Versuchsreihe zeigen sie, dass die Zufallssuche bei gleichem Zeitbudget im Schnitt bessere Ergebnisse in der Optimierung von neuronalen Netzen liefert als die Rastersuche. Entsprechend kann bei geringerem Budget eine vergleichbare Performanz erwartet werden. Selbst bei einer sequentiellen, manuellen Optimierung, die auf den Erfahrungen von Forschern beruht, erreicht die Zufallssuche in vier von sieben Fällen vergleichbare, in zwei Fällen schlechtere und in einem Fall bessere Werte. *Bergstra und Bengio* empfehlen also, dass die zufällige Suche, bei der Hyperparameter aus einer Normal-, Gleich- oder Log-Verteilung gezogen werden, als Benchmark für weitere Methoden zur Hyperparameteroptimierung verwendet werden sollen.

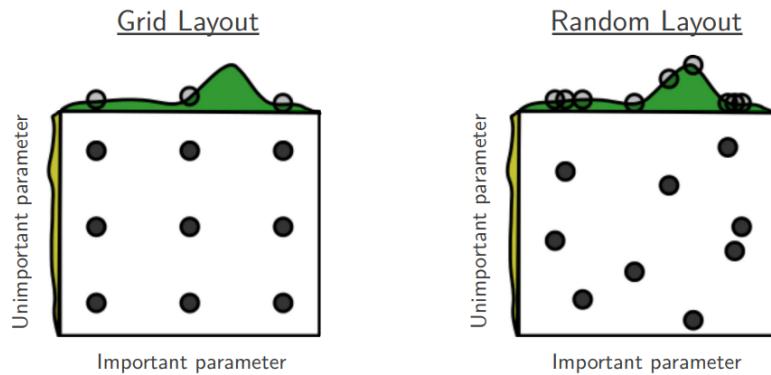


Abbildung 4.9: Vergleich von Raster und zufälliger Suche mit zwei Parametern [HPO7].

4.4.2 Bayes'sche Suche

Eine Bayes'sche Suche ist eine informierte Suche, welche auf Basis der bereits gefundenen Werte ein Modell trainiert. Dieses Modell kann für die Vorhersage von neuen Parametern verwendet werden, um so schnell eine optimierte Konfiguration zu finden. *Weights and Biases* [17] trainiert für dieses Vorhersagemodell einen Random Forest [HPO13]. *Hutter et al.* [HPO5] beschreiben in ihrem Framework SMAC, dass ein solcher Forest aus einzelnen Bäumen besteht, die wiederum anhand eines zufälligen Subsets der verwendeten Parameter aus der Konfiguration versuchen, die Klassifikationsgenauigkeit zu bestimmen. Je mehr Bäume für den Forest trainiert werden, desto stärker kann dieser auf die Hyperparameter verallgemeinern.

Aus einem trainierten Random Forest entsteht also eine Optimierungsoberfläche, wobei für jeden Wert auch eine Konfidenz vorliegt. In Abbildung 4.10 rechts wird eine solche Oberfläche für zwei Parameter gezeigt. Wärmere Farben stehen für eine hohe Erwartung an das Modell mit der gewählten Konfiguration. Links hingegen werden die Auswir-

kungen von Stichproben auf das Explorations- und Exploitationsverfahren gezeigt: Mit zunehmender Menge von Proben verkleinern sich die geschätzten Konfidenzintervalle der objective function, wodurch genauere Aussagen über den Einfluss des Parameters auf die Klassifikation getroffen werden können. Die Acquisition Function (dt. Erfassungsfunktion) hingegen drückt aus, wie hoch der erwartete Wissensgewinn durch Testen der Stichprobe ist. Dabei wird nach einer Heuristik die Exploration mit der Exploitation abgewogen, um so einen möglichst hohen Erwartungswert in der Stichprobe zu erhalten. Sind genügend Stichproben vorhanden und im Modell trainiert, kann ein posterior berechnet werden, der anhand dieser Daten die effektivste Konfiguration vorschlägt.

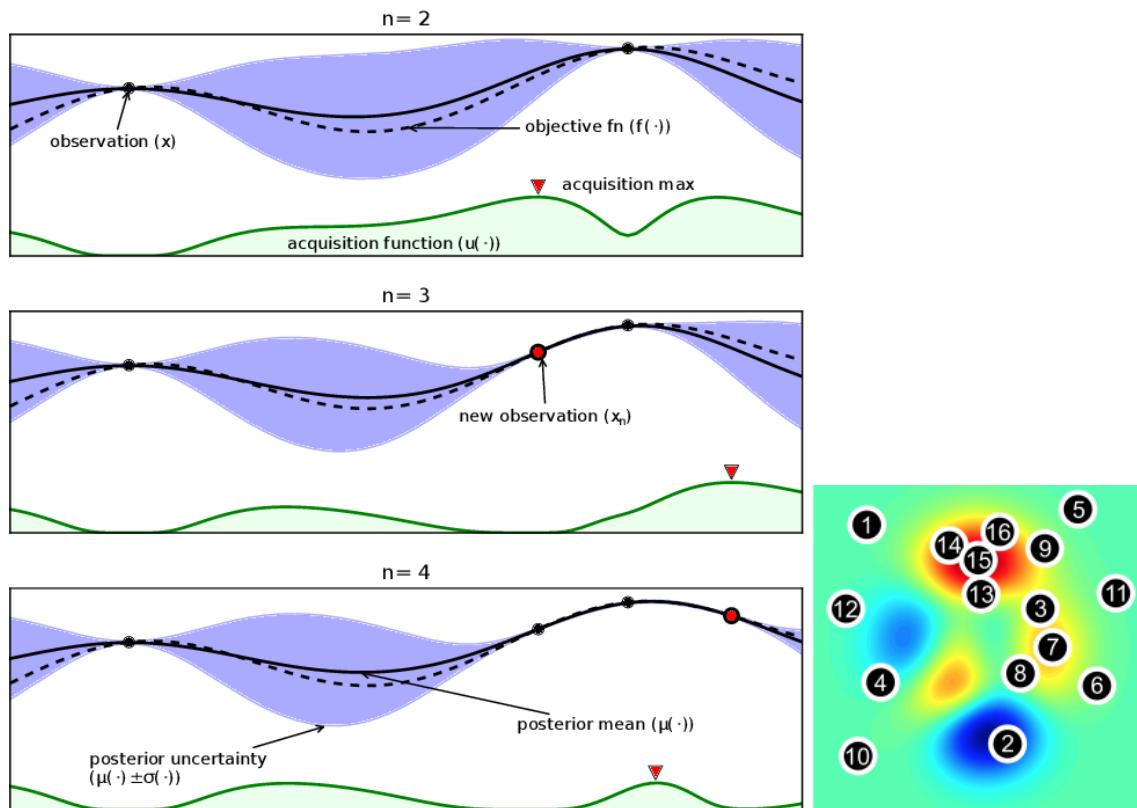


Abbildung 4.10: Links: Exploration und Exploitation über die Erwartungswerte eines Modells in einer Bayes'schen Suche [HPO8]. Rechts: Aufspannen des Problems in zwei Dimensionen [HPO10].

Weights and Biases [17] berechnet aus den Random Forests zudem eine Visualisierung, die die Korrelation und die Einflussstärke der jeweiligen Parameter auf die zu optimierende Metrik zeigt [HPO13]. Da eine Bayes'sche Suche in diesem Bereich, entgegen der Intuition, schlecht auf hoch-dimensionale Räume verallgemeinern kann, werden weitere Optimierungen für das Training, wie einem early stopping mit dem Hyperband Algorithmus von *Li et al.* [HPO10], vorgeschlagen.

4.5 Tools

Für diese Arbeit werden neben den fundamentalen Tools zur Implementierung von neuronalen Netzen, namentlich *PyTorch* und *Tensorflow*, auch weitere Tools verwendet. *Weights and Biases* <https://wandb.ai/> [17] ist dabei ein sehr umfangreiches Tool zum Verwalten des Trainings, der Trainingsmetriken, der Auswertung der Metriken und einer Hyperparameteroptimierung. *Roboflow* <https://roboflow.com/> hingegen spezialisiert sich auf die Verwaltung der Trainingsdatensätze und bietet Augmentationsmöglichkeiten und eine Untersuchung der Datensätze kostenfrei an.

4.5.1 Weights and Biases

Weights and Biases stellen eine Web-Oberfläche bereit, über die Trainingsabläufe gesteuert und verwaltet werden können. Sobald die API im Training des neuronalen Netzes eingebunden ist, werden automatisch die Metriken zur Auswertung des Modells gespeichert und für eine Ansicht in Echtzeit aufbereitet. Es werden auch Statistiken zum Datensatz aufgeführt, wie beispielsweise die Verteilung der Bounding Boxen. Weiterhin lassen sich Daten über das System, auf dem das Training stattfindet, auslesen, wie die GPU-Auslastung, der Stromverbrauch oder im Allgemeinen die Trainingsdauer. Für größere Aufgaben kann das Training über die Website zentralisiert verwaltet und auf die verschiedenen Instanzen verteilt werden.

Ein besonders nützliches Feature für diese Arbeit ist der Sweep: Auf das vorgegebene Netzwerk wird eine Hyperparameteroptimierung durchgeführt. Bei dieser erstellt das Tool automatisch eine Visualisierung der Konfigurationen und deren Erkennungsqualität. Die Hyperparameter werden je nach ausgewählter Strategie durch eine Zufalls-, Raster- oder Bayes'sche Suche nach der besten Konfiguration durchsucht. Zudem wird eine Korrelation zwischen Hyperparameter und Trainingsergebnis berechnet.

Um ein Abonnement und die Notwendigkeit Daten, besonders vom Trainingssatz, auf fremden Servern zu speichern zu umgehen, kann auch eine lokale Instanz von Weights and Biases eingerichtet werden. Eine Anleitung dafür ist auf deren Website gegeben. Es ist auch möglich, mit einem Abonnement ein Training auf deren Servern durchzuführen. Da aber ausreichende Ressourcen zur Verfügung standen, wurde dieses Angebot nicht wahrgenommen.

4.5.2 Roboflow

Auch bei Roboflow handelt es sich um ein Tool, welches online betrieben wird. Es fokussiert sich auf die Verwaltung von Datensätzen und bietet mit einem kostenfreien Abonnement die Möglichkeit, bis zu 10.000 Einzelbilder als Datensatz aufzubereiten. Die Bilder können mit Annotationen hochgeladen werden oder erst online mit eigenen Annotationen versehen werden. Sind die Daten vollständig, kann in einen Trainings-, Test- und Validierungsdatensatz eingeteilt werden. Der Trainingsdatensatz kann mit einer großen

Menge festgelegter Augmentationen transformiert werden. Weiterhin bringt es eine Versionierung mit, die eine Übersicht über die Datensätze schaffen soll. Der Download findet entweder im ausgewählten Format oder als API-call im Code selbst statt. Bei den Formaten ist aus einer weiten Reihe gängiger Formate für verschiedene Frameworks und Bibliotheken auszuwählen. Beispiele umfassen `.tfrecord` oder die Ordnerstruktur aus YOLOv5, die für diese Arbeit benötigt werden. Letztlich werden auch hier Kapazitäten für ein Training angeworben.

4.5.3 Technisches Setup

Wie bereits beschrieben wurde, werden in dieser Arbeit besonders die Modelle SSDLite MobileNetV3 [NN50] und YOLOv5 [NN59] verwendet. Für das Training der Modelle werden die verbreiteten Frameworks Tensorflow [NN31] und PyTorch [NN37] verwendet. Der Computer, auf dem das Training stattfindet, enthält einen AMD Ryzen 7 3700x, 32GB RAM und eine Nvidia 2070. Das Training findet unter Ubuntu 20.04 statt. Als Smartphones werden das OnePlus 6 und Google Pixel 6 verwendet.

5

KAPITEL

Auswertung bestehender Daten

5.1	Diskussion vorliegender Ergebnisse	47
5.2	Eigenschaften vorliegender Datensätze	49
5.2.1	Datensatz: Augen	49
5.2.2	Datensatz: Iris und Pupillen	51
5.2.3	Bewertung	53
5.3	Eigener Datensatz	53

In diesem Kapitel werden die in den Grundlagen angesprochenen Datensätze und Architekturen von neuronalen Netzen auf ihre Verwendbarkeit untersucht. Es sollen empirisch quantitative Maßstäbe gefunden werden, anhand derer eine Optimierung stattfinden kann. Explizit sollen die Datensätze auf Vollständigkeit, natürliche Fehlerrate und Eignung für das vorliegende Problem überprüft werden. Die Modellarchitekturen hingegen sollen grundsätzlich auf ihre Performanz in Relation zur Parameterzahl untersucht werden; es soll ein Balance zwischen Genauigkeit und Trainierbarkeit auf den Datensätzen für weitere Experimente in der Hyperparameteroptimierung gefunden werden.

5.1 *Diskussion vorliegender Ergebnisse*

Moryäner [26] verwendet in seiner Arbeit das SSDLite MobileNetV3 [NN50] und selbst erstellte Augmentationen wie Änderungen des Kontrasts, Transformationen, zufälligem Löschen von Ausschnitten und konvexen Polygonen, die einer Spiegelung von Licht im Auge nachempfunden sein sollen. Die Ergebnisse für das Training mit dem SSDLite MobileNetV3 auf dem vorliegenden Augen- und Pupillendatensatz werden in Tabelle 5.1 aufgeführt. Dabei werden für beide Datensätze jeweils die Version mit und ohne Daten-augmentation herangezogen. Als Metriken werden die mAP.95 und die AP@.75 verwendet und als Trainingzeitraum werden 30.000 Steps zur Anpassung der Gewichte für die Pupillenerkennung, 100.000 Steps für die Augenerkennung angegeben. Laut Code wird

	Auge	Auge_augmented	Pupil	Pupil_augmented
mAP.95	0.605	0.64	0.565	0.6
AP@.75	0.72	0.77	-	-

Tabelle 5.1: Trainingsergebnisse aus *Moryäner* [26] für das SSDLite MobileNetV3 auf dem Augen- und Pupillendatensatz, jeweils mit oder ohne Augmentationen.

eine Batchsize von 4 verwendet; es fanden also insgesamt 120.000 respektive 400.000 Inferenzen mit Bildern statt. In Bezug auf Epochen ergeben sich, unter Einbezug der von ihm angegebenen, verwendeten Datensatzgrößen, $\frac{100.000}{6.000} = 16.67$ für die Augen und $\frac{30.000}{656} = 45.7$ für die Pupille. Erfahrungsgemäß sollte diese Menge an Trainingsschritten für ein Training ausreichen, welches sich auch in den angegebenen Trainingsverlaufsgraphen in Abbildung 5.1 widerspiegelt, da die Graphen stagnieren und keine weiteren Zuwächse in der Genauigkeit verzeichnen.

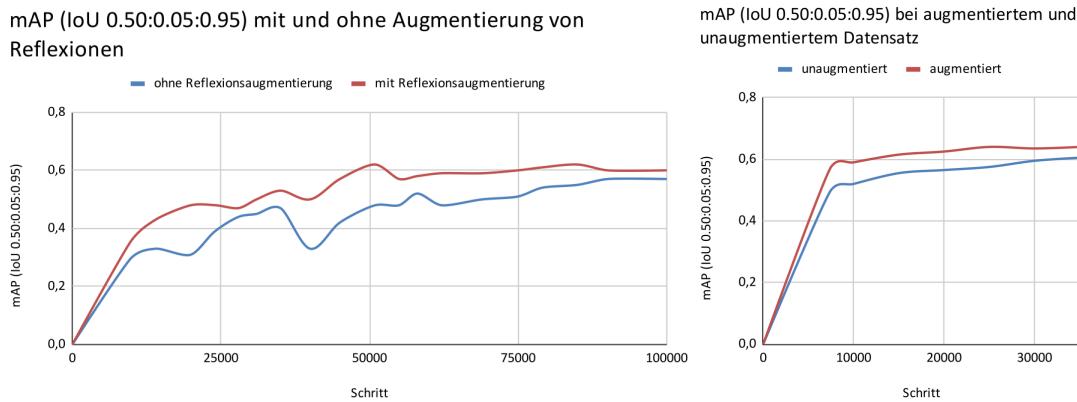


Abbildung 5.1: Trainingsverläufe von *Moryäner* [26] für (links) den Augendatensatz und (rechts) den Pupillendatensatz.

Die angegebenen Werte beziehen sich vermutlich nur auf einen einzigen Trainingsdurchlauf, geben keine Konfidenz an und geben somit keine verlässlichen Werte wieder. Daher ist hier auf eine geringe Qualität im Datensatz, eine zu kurze Trainingsdauer oder eine Unterdimensionierung der verwendeten Netzstruktur zu schließen. Die letzteren beiden Punkte lassen sich allerdings durch einen Vergleich mit anderen Anwendungen der Netzstruktur vorläufig widerlegen: *Howard et al.* [NN50] zeigen, dass ihr SSDLite MobileNetV3 auf dem ImageNet Datensatz [DATA1] eine hohe Genauigkeit in der Klassifikation der Objekte erreicht. Weiterhin erreicht es auf dem COCO Datensatz [DATA3] eine AP@.50 von 16% in der kleinen, 22% in der großen Variante. Es ist sollte also möglich sein, trotz der geringen Größe des Modells auch komplexe Probleme angemessen zu bearbeiten. Es wird erwartet, dass die Erkennung von Pupillen und der Iris ein deutlich einfacheres Problem ist, wie aus der Gaze Estimation hervorgeht. Entsprechend lässt sich hier als Arbeitshypothese formulieren, dass die verwendeten Datensätze von niedriger Qualität sind und eine hohe natürliche Fehlerrate beinhalten. Das Netz wird somit sowohl in dem Training als auch in der Verifikation von eben diesen Datensätzen behindert. Es muss folglich ein neuer Datensatz gefunden oder erstellt werden, der diese Probleme löst. Da

eine Recherche nach Datensätzen keinen Erfolg ergab, wird in Abschnitt 5.3 beschrieben, wie in dieser Arbeit ein neuer Datensatz erstellt wird.

5.2 Eigenschaften vorliegender Datensätze

Im folgenden Abschnitt werden die von *Moryäner* [26] verwendeten Datensätze analysiert. Da auf die Qualität der Datensätze, nicht aber die Qualität der verwendeten Netze und deren Optimierung eingegangen werden soll, wird auf die größte, leicht verfügbaren Variante des YOLOv5 Modells [NN59], das Xtra Modell, zurückgegriffen. Es liefert State of the Art Ergebnisse analog zu dem Vorgänger YOLOv4 [NN53]. Das YOLOv5 Small hingegen stellt als kleineres Modell einen Vergleichswert dar und soll als Referenzpunkt dienen, dass bessere Trainingsergebnisse nicht ausschließlich der Größe des Modells geschuldet sind. Es soll ausgeschlossen werden, dass der Datensatz entgegen der Erwartungen eine ähnliche Komplexität, wie ImageNet [DATA1] oder COCO [DATA3] hat.

Das Training findet mit einem auf COCO vortrainierten Modell statt, wodurch schnell gute Erkennungsraten erreicht werden sollen, wie aus dem Transfer Learning bekannt ist. Deutlich längere Trainingszeiträume von 50 respektive 200 Epochen werden verwendet, um das Trainingsverhalten auf längere Zeit zu betrachten, um so ein Underfitting vorzubeugen und zu überprüfen, ob und wann ein Overfitting einsetzt. Zudem soll somit die unterschiedliche Menge an Instanzen der Datensätze berücksichtigt werden. Die Trainingsvorgänge werden pro Netzgröße 5 mal wiederholt; abgebildet wird mit einer breiten Linie der geometrische Durchschnittswert der mAP.95. In einer beigefügten Grafik wird im gleichen Format die Precision und der Recall pro Epoche abgebildet. Die umliegende Fläche beschreibt die jeweils besten und schlechtesten Werte pro Epoche. Die Achsenbeschriftung ist auf den relevanten Bereich für die jeweiligen Metriken beschränkt, um einen anschaulichen Ausschnitt, besonders im austrainierten Bereich, zu gewährleisten. Letztlich wird die lokale Verteilung der Features in den Datensätzen aufgezeigt. Hierbei wird die Korrelation der Höhe und Breite, sowie der Position im Bild kurz untersucht.

5.2.1 Datensatz: Augen

Der Datensatz umfasst circa 12.500 Bilder und hat eine große Auswahl hochauflösender Bilder. Auf der Abbildung 5.2 wird deutlich, dass das Training nach etwa 25 Epochen bei einem Höchstwert in der mAP.95 von 80 % angelangt. Diese Werte gelten sowohl für das Slim, als auch das Xtra Modell, womit beide Modelle deutlich genauer arbeiten, als das von Moryäner eingesetzt SSDLite MobileNetV3, welches nur knapp über 60 % erreichte. Neben dem signifikanten Größenunterschied vom SSDLite MobileNetV3 zum Slim-Modell, trägt die Verwendung des vollständigen Datensatzes und die Anwendung von vorkonfigurierten Bildaugmentationen des YOLO-Frameworks, zur signifikanten Verbesserung der mAP.95 bei. Weiterhin zeigt sich in dem Recall, dass nahezu alle Augen auf den Bildern gefunden werden und bei Betrachtung der Precision die mit hoher

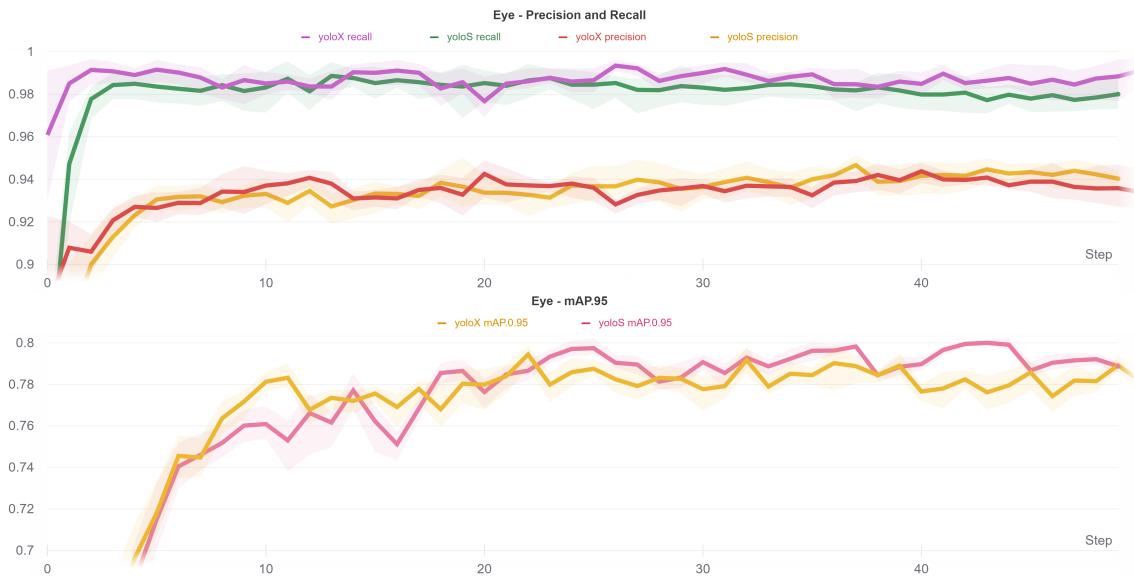


Abbildung 5.2: Precision, Recall und mAP.95 für das YOLO Small und Xtra auf dem bestehenden Augen-Datensatz über 50 Epochen.

Genaugigkeit.

Da zwischen den beiden verwendeten Netzen in keiner Metrik anhaltende, signifikanten Differenzen bestehen, kann ausgeschlossen werden, dass ein weiteres, größeres Modell bessere Werte liefern kann. Der Recall ist nahezu ausgeschöpft und einer höhere Genauigkeit scheint unerreichbar. Aus diesen Gründen ist die Diskrepanz zwischen dem hohen Precision Wert und der mittelmäßigen mAP.95 auf den Datensatz und Probleme in dessen Annotationen zurückzuführen. Wie in Abbildung 5.3 rechts zu sehen ist, sind auf einigen Validierungsdaten, dementsprechend auch auf den Trainingsdaten, die Augen von Personen im Hintergrund häufig nicht annotiert. Dies verringert den Precision-Wert, berührt aber nicht den Recall. Auf der Verteilung der Bounding Box-Zentren links ist zu sehen, dass selten auch Augen aus dem Hintergrund der Person im Datensatz annotiert wurden. Anscheinend liegt hier eine inkonsistente Annotation vor. Weitere Fakten, die aus der Verteilung abzulesen sind, beinhalten, dass die Augen meist zentriert und deutlich breiter als hoch sind. Ohne die Mosaik-Augmentierung vom YOLO-Framework könnte dies zu einer schlechten Fähigkeit führen, Augen auf dem gesamten Bild zu erkennen, da diese Fälle selten gelernt werden.

Insgesamt liegt hier also ein Datensatz vor, mit dem eine Augenerkennung erfolgreich trainierbar ist, jedoch nur mit einer mittelmäßigen mAP.95 von etwa 80%. Der hohe Recall verspricht eine Erkennung nahezu aller Augen auf den Bildern, wird jedoch durch die unpräzisen Annotationen eingeschränkt. Der Datensatz lässt sich folglich aufgrund der Validierungsdaten nur bedingt für eine Feinoptimierung verwenden, da die Ergebnisse inkonsistent sind und keine Werte deutlich über 80% möglich sind. Zudem kann die natürliche Fehlerrate nicht genau geschätzt werden.

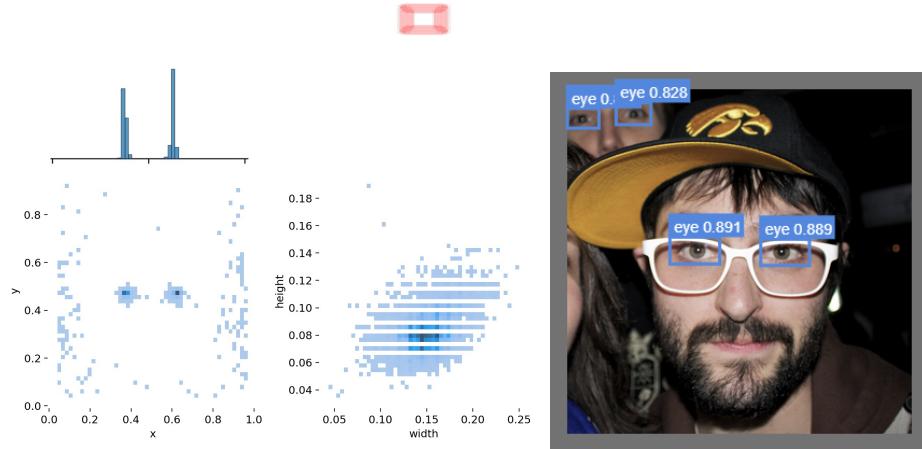


Abbildung 5.3: Verteilung und Korrelation der Höhe und Breite der Bounding Box für den Augen-Datensatz links; Probleme mit den Ground Truth Daten rechts. (geändert nach [DATA11])

5.2.2 Datensatz: Iris und Pupillen

Der Iris- und Pupillen-Datensatz baut auf denselben Bildern auf, wobei sie nicht mit demselben Modell trainiert wurden und deshalb als disjunkt betrachtet werden. Um die Verifikationsdaten nicht wiederzuverwenden, wurden für den Iris-Datensatz 19 Bilder aus dem Trainings-Anteil in den Validierungs-Anteil übertragen; die bisherigen Verifikationsdaten wurden verworfen. Grundsätzlich wird infrage gestellt, wie effektiv das Training auf ca. 650 und die Validierung auf 19 Datenpunkten stattfinden kann. Im ersten Teil werden die Ergebnisse für die Iris beschrieben.

Auswertung des Iris-Datensatzes

In der Grafik 5.4 ist oben im Recall und der Precision zu sehen, dass die Werte Anfangs einzelnen Schwankungen unterliegen, später im Training aber konstant bei 100 % respektive 94.7 %. Die Differenz in der Precision lässt sich auf eine einzelne fehlerhafte Instanz im Validierungsset festlegen, da $\frac{18}{19} = 94.7\%$. Es scheinen keine Verbesserungsmöglichkeiten vorzuliegen, da der Fehler sich über die Trainingsepochen hinweg durchsetzt. Die robustere mAP.95 zeigt unter der Annahme, dass die Netzstruktur alle Informationen extrahieren kann, dass im Datensatz eine natürliche Fehlerrate von knapp 9 % vorliegt, also eine Optimierung maximal diesen Wert von etwa $mAP.95 = 91\%$ anstreben kann. Da es keinen markanten Unterschied zwischen den Ergebnissen des Small und des Xtra Modells gibt, wird die Annahme als wahr angenommen, wodurch eine nahezu perfekte Erkennung nicht erreicht werden kann. Vergleichswerte zum Training von Moryäner liegen nicht vor.



Abbildung 5.4: Precision, Recall und mAP.95 für das YOLO Small und Xtra auf dem bestehenden Iris-Datensatz über 200 Epochen.

Auswertung des Pupillen-Datensatzes

Der Pupillen-Datensatz hingegen zeigt in der Precision und dem Recall instabilere Werte, da häufiger Schwankungen in der Validierung auftreten. In diesen Metriken unterscheiden sich die Modelle nicht und konvergieren beide wieder bei 94.7 % respektive 100 % für den Recall. Die mAP.95 hingegen zeigt ein anderes Bild auf: Das Xtra Modell schneidet mit 80 % etwa 8 Prozentpunkte besser ab, als das Small Modell. Da Trainingsverläufe über alle Wiederholungen hinweg einen klaren Trend abzeichnen, werden die Ergebnisse als signifikant interpretiert. Sie weisen darauf hin, dass das komplexere Modell eine unerwartete Komplexität in den Bildern aufgreifen kann, um so eine bessere Erkennung zu liefern. Es scheint unklar, welche Faktoren zu dieser unerwarteten Komplexität beitragen. Eine Hypothese ist, dass Reflexionen in der Pupille deren Umriss unterbrechen, wodurch es besonders bei der kleinen Auflösung der Bilder zu einer Erschwernis in der Erkennung kommen kann. Als zweite Hypothese kann formuliert werden, dass aufgrund der kleinen Außenmaße der Pupille die Anchors, welche von der YOLO-Architektur verwendet werden, nicht präzise genug angewendet werden können. Da das Xtra Modell über mehr dieser Anchorpunkte verfügt, könnten die kleinen Differenzierungen besser wiedergegeben werden als im Small Modell. Weiterhin wirkt sich die geringe Größe bei Fehlerkennungen im Bereich einzelner Pixel stärker auf die IoU aus, wodurch die Schwellwerte der Precision schneller unterschritten werden und Schwankungen in der Metrik stärker ausfallen.

Die Abbildung 5.6 zeigt links die Verteilung der Bounding Boxen für den Iris-Datensatz. Zu sehen ist, dass die Iriden im Intervall von 0.4 bis 0.6 auf der x- und y-Achse annähernd gleichverteilt sind. Es gibt zudem eine starke Korrelation zwischen der Höhe und der Breite; die hier vorliegenden Bounding Boxen sind annähernd quadratisch. Die meisten Bounding Boxen nehmen weniger als 10 % der Bildfläche ein. Für die Pupille lassen sich

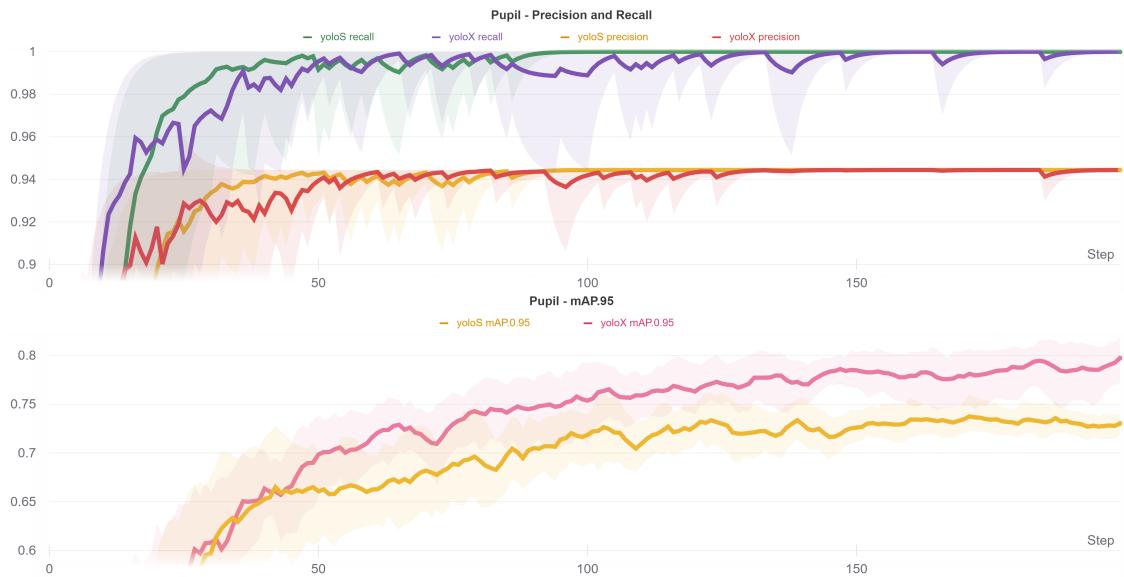


Abbildung 5.5: Precision, Recall und mAP.95 für das YOLO Small und Xtra auf dem bestehenden Pupillen-Datensatz über 200 Epochen

ähnliche Beobachtungen aus der Grafik 5.6 rechts ableiten, mit der Ausnahme, dass sie eher einer Normalverteilung folgen. Da die Pupille zentriert in der Iris liegt und nicht in verschiedene Richtungen elliptisch expandieren kann, ist dies vermutlich den Annotationen geschuldet. Die meisten Pupillen haben eine Größe von 10-15 % der Außendimensionen des Bildes, machen also ca. 1-1.3 % der Bildpixel aus.

5.2.3 Bewertung

Auf keinem der Datensätze lassen sich Werte jenseits der 90 % mAP.95 erreichen. Bei der Pupillenerkennung werden kaum 80 % erreicht. Jedoch liegt überall ein hoher bis fast perfekter Recall vor; die Precision mit weit über 90 % ausreichend. Dass die mAP.95 schlechtere Werte liefert, liegt dementsprechend vermutlich an den Annotationen der Datensätze und nicht am Training und dessen Methoden verortet werden. Auffällig und unerwartet hingegen ist jedoch die unterschiedliche Performanz der beiden Modelle auf dem Pupillen-Datensatz. Es wird in späteren Experimenten untersucht werden müssen, ob die Komplexität dieser Aufgabe die Erwartungen überschreitet oder inhärente Probleme von der YOLO-Architektur in Bezug auf Objekte in sehr geringer Auflösung diese Diskrepanz erklären. Aufgrund dieser Argumentation wird im folgenden Teil die Erstellung und Bewertung eines eigenen Datensatzes mithilfe des Unity-Eyes [DATA5] Modells beschrieben und durchgeführt.

5.3 Eigener Datensatz

Da die vorhandenen Datensätze für die Iris- und Pupillenerkennung für eine genaue Objekterkennung ungeeignet sind, muss ein Ersatz für diesen Datensatz gefunden werden. Wie bereits in Kapitel 4.1.2 beschrieben wurde, wird in dieser Arbeit auf der Weiterent-

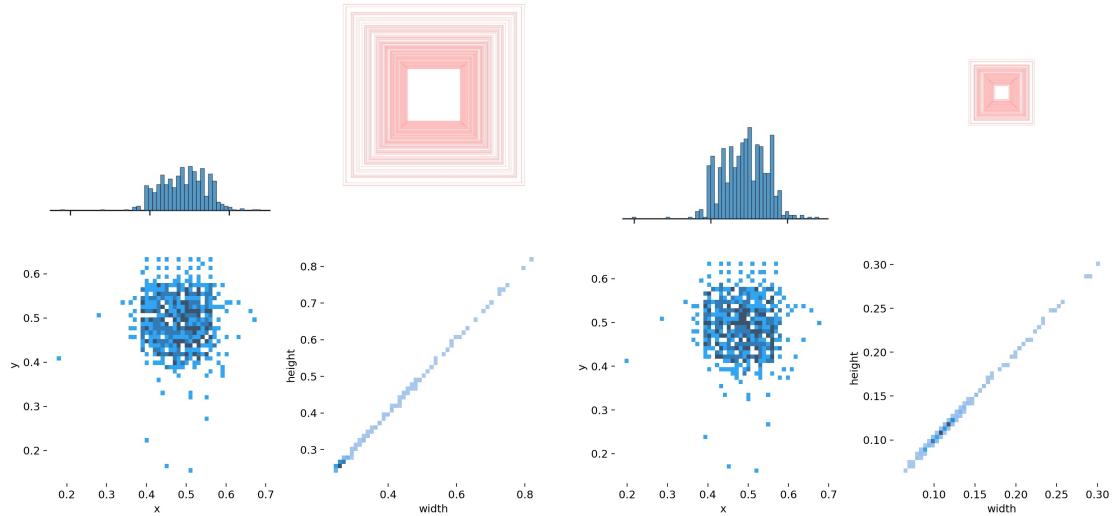


Abbildung 5.6: Verteilung und Korrelation der Höhe und Breite der Bounding Box für den Iris- und Pupillen-Datensatz.

wicklung von SynthesisEyes [DATA4], also Unity-Eyes [DATA5], aufgebaut. Es beinhaltet ein detailliertes 3D Modell von menschlichen Augen und der umliegenden Gesichtspartie. Diese Anwendung wurde unter Windows im Detailgrad „fantastic“ für einen höchstmöglichen Detailgrad, besonders in den Reflexionen auf dem Augapfel, ausgeführt. Die Auflösung wurde auf 1600x1200 Pixel gesetzt, die Parameter für die Kameraposition und den Augenwinkel auf [0, 0, 15, 15], also ein Intervall zwischen 0-15 Grad in den X- und Y-Winkel, gestellt. Aus den Einstellungen wurden mit dem Drücken des „Start“-Buttons 22.000 Bilder erstellt und zusammen mit den jeweiligen Annotationen in einem Unterordner erstellt. Die Annotationen sind umfangreich und die Auflösung der Bounding Box von Objekten ist mit 32 Fixpunkten hoch. Sie sind genügend umfangreich, dass das Modell für weitere Aufgaben und Analysen im Bereich der Augen verwendet werden könnte. Es ist anzumerken, dass die Bounding Box jeweils das Objekt ohne Überdeckung durch beispielsweise das Augenlid beschreibt. Folgende Merkmale werden aufgezeichnet:

- `interior_margin_2d`: Die äußere Bounding Box der Sclera.
- `caruncle_2d`: Die Bounding Box des Tränenkranukels an der nasenseitigen Verengung der Augenlider.
- `iris_2d`: Die Bounding Box der Iris mit 32 Fixpunkten.
- `eye_details`: Der Blickwinkel, die Farbe der Iris, die relative Größe der Iris (Mittelpunkt bei 1.0) und Pupille (Mittelpunkt bei 0).
- `lightingv_details`: Die Textur der Skybox für die Spiegelungen, deren Rotation, die Rotation der Lichtquelle und die Stärke dieser.
- `eye_region_details`: Die PCA-Koeffizienten und Hautfarbe.
- `head_pose`: Die Drehung des Kopfes in Euler-Koordinaten.

Das wichtigste Merkmal, das in der Liste nicht aufgeführt wird, ist die Bounding Box der Pupille. Weiterhin werden Daten wie das Geschlecht, das Alter, potentielle Augenerkrankungen oder Deformierungen und auch ein linke Gesichtshälfte nicht aufgeführt. Trotz der Abwesenheit dieser Eigenschaften wird auf diesem Modell weiter aufgebaut, da es eine nahezu unbegrenzte, kontrollierte Auswahl von Augenpartien generieren kann. Dementsprechend stellt es ein vielversprechendes Modell für künftige Arbeiten im Bereich der Augenerkennung dar. Weiterhin beschreibt Nvidia [DATA12], ein Führer auf dem Gebiet der AI, mit Berufung auf Forschungsartikel von unter anderem dem Forschungsinstitut *Gartner*, dass synthetische Modelle im kommenden Jahrzehnt die dominante Quelle für Datensätze werden. Dies wird durch eine höhere Qualität, bessere Verfügbarkeit und geringere Kosten begründet. Durch quasi kostenfreie Generierung weiterer Daten lassen sich die Datensätze auch ohne Augmentation mit zusätzlichen Punkten im latenten Raum befüllen, was zu einer besseren Generalisierbarkeit führen soll.

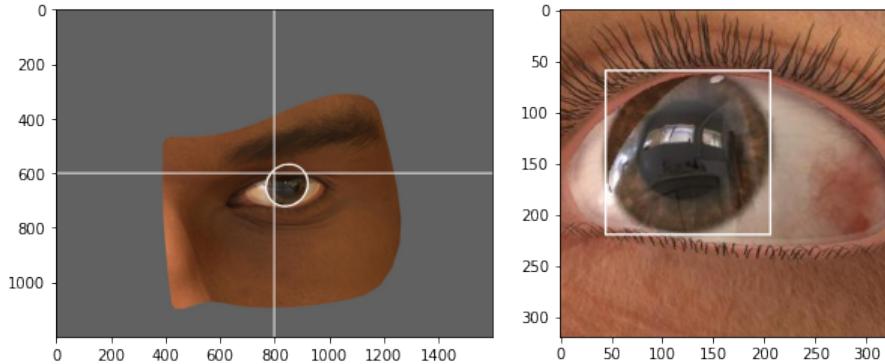


Abbildung 5.7: Der generierte Datensatz vor und nach dem Zuschneiden. Markiert sind links die Annotationen für die Iris und die 50 % Punkte der Höhe und Breite. Rechts: Die ausgeschnittene Version mit angepasster Bounding Box.

Im Folgenden wird die Abbildung 5.3 erklärt, die den Ablauf beschreibt, wie der generierte Datensatz mittels Ensembles und anderen Methoden vorverarbeitet wird, um als neuer Datensatz für diese Arbeit zu fungieren. Im ersten Schritt wird der graue Rahmen, der die Augenpartie umschließt, entfernt. Dieser informationsarme Bereich wird in Abbildung 5.7 gezeigt. Das resultierende Bild wird von 1600x1200 Pixel auf eine Auflösung von 320x320 Pixel zugeschnitten. Beim Ausschneiden wird der Ausschnitt mittig gewählt, jedoch ist eine kleine Varianz eingebaut, sodass das Auge nicht immer in der exakten Mitte ist. Im gleichen Schritt werden auch die Rohdaten der Iris-Annotationen durch eine einfache Maximalwertfindung in eine Bounding Box übersetzt und skaliert, womit die Vorverarbeitung für diese bereits beendet ist.

Für die Bounding Box der Pupille hingegen liegen keine Annotationen vor, weswegen auf das Semi Supervised Learning zurückgegriffen wird. Da aus der Analyse des Pupillen-Datensatzes hervorging, dass die Pupillen gefunden werden, dies aber nur mit einer schlechten Präzision einhergeht, wurde ein Ensemble aus fünf YOLO Xtra Modellen trainiert. Mit jedem dieser Modelle im Ensemble wurde eine Inferenz für jedes der 22.000

Bilder im Datensatz durchgeführt. Auf die Ergebnisse wurde der Non-Max-Suppression Algorithmus angewandt und eine Untergrenze der zur Erkennung benötigten Konfidenz von 70 % angelegt. Diese Annotationen werden in einem temporären Ordner zwischen- gespeichert und mit der Funktion 5.8 ausgewertet. Für jedes Bild aus dem Datensatz soll der Name extrahiert werden, um die Rohdaten der fünf Modelle einzulesen und in den Listen CX, CY, DX, DY abzuspeichern. In CX, CY werden die zentralen x- und y- Koordinaten der Bounding Box gespeichert, in DX, DY die Distanzen bis zur Außenkan- te der Bounding Box, um dem Format, welches YOLO fordert, gerecht zu werden. Der Mittelwert und die Standardabweichung jeder Liste werden für die verfügbaren Werte berechnet. Schließlich wird überprüft, dass einerseits mehr als ein Modell eine Annota- tion für das Eingabebild gefunden hat, andererseits, dass die Standardabweichung der erkannten Bounding Box über die Modelle hinweg nicht 0.01, also etwa 3 Pixel, über- schreitet. In der ersten Iteration erfüllen die Inferenzen von ca. 7.000 der 22.000 Bilder diese Anforderungen. Besonders Bilder mit starken Reflexionen, in denen die Außenlinie der Pupille unterbrochen wurde, wurden hierbei nicht gefunden. Die Modelle, die für diesen Schritt verwendet wurden, entsprechen denen aus Abschnitt 5.2.2. Für künftige Arbeiten können diese Werte restriktiver eingestellt werden, um das Rauschen in den Annotationen weiter zu verringern.

```

for index, file in enumerate(files):
    name, ext = os.path.splitext(file)
    CX, CY, DX, DY = get_raw_annotations(EXP_1, name)
    mn_cx, mn_cy, mn_dx, mn_dy = calc_metric(CX, CY, DX, DY, np.mean)
    std_cx, std_cy, std_dx, std_dy = calc_metric(CX, CY, DX, DY, np.std)

    if len(CX) > 1 and std_cx < 0.01 and std_cy < 0.01
        and std_dx < 0.01 and std_dy < 0.01:
        annotation = get_annotation('pupil', mn_cx, mn_cy, mn_dx, mn_dy)

```

Abbildung 5.8: Einlesen und Vorverarbeiten der ersten Iteration der Pupillen- Annotationen.

Im nächsten Schritt wird aus den neu gewonnenen, kuratierten Annotationen mit den zugehörigen Bildern ein neuer Datensatz erstellt. Mit diesem werden die vorherigen Schritte wiederholt und es werden fünf neue Modelle für die Pupillenerkennung trainiert. Der Trainingsfortschritt wird in der Grafik 5.9 gezeigt. Das Training wurde auf 20 Epochen verkürzt, da das Trainingsset circa elf mal so viele Daten umfasst, wie der originale Iris- und Pupillen-Datensatz. Zudem sind die Precision und der Recall konstant bei 100 %, die mAP.95 ist mit etwa 91 % auch zufriedenstellend, da vorher Werte größer 80 % unmöglich waren. Insgesamt scheint der Datensatz seinen Zweck zu erfüllen, da die Precision perfekte Werte liefern kann. Entsprechend werden alle Pupillen und Iris auf den Bildern gefunden. Mit der höheren Auflösung und dem komplexeren Aufbau der Spiegelungen in den Augen, erscheinen die mAP.95-Werte gerechtfertigt; sie werden aber im Verlauf der Arbeit weiter analysiert und optimiert.

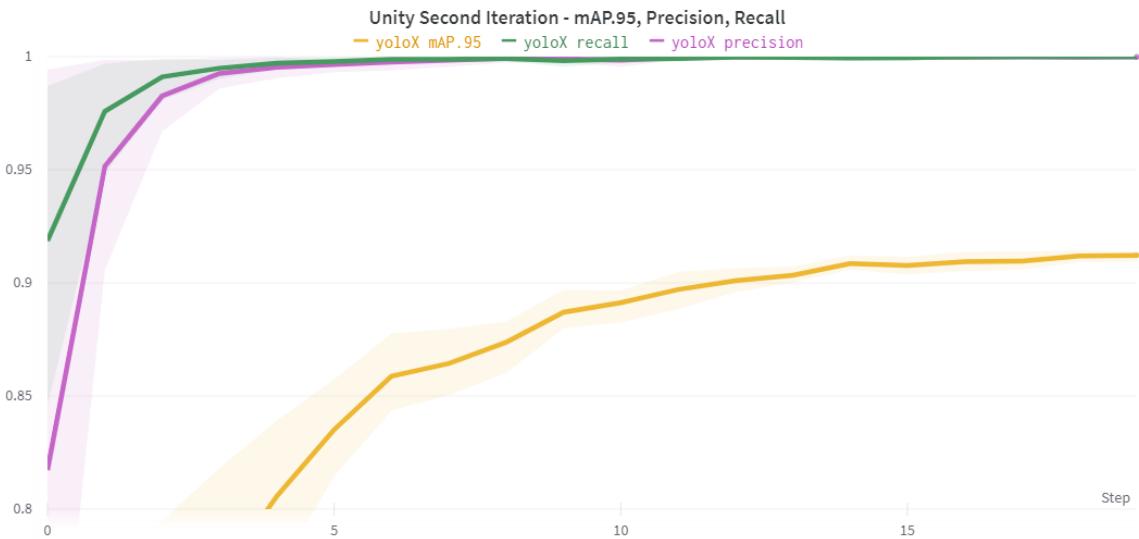


Abbildung 5.9: Trainingsfortschritt in 20 Epochen mit dem neuen Unity Datensatz. Die verwendeten Metriken sind mAP.95, Precision und Recall.

Unter Verwendung dieser neuen Modelle wird ein neues Ensemble konstruiert und es wird eine neue Inferenz für die vollen 22.000 ausgeschnittenen Bilder durchgeführt. Mit den gleichen Restriktionen wie in dem Code 5.8 werden nun neue Annotationen für die Bilder generiert und gespeichert. Dabei wird eine 90/10-Unterteilung in Trainings- und Validierungsdaten vorgenommen. Insgesamt ergeben sich so etwa 19.500 Trainingsdaten und 1950 Validierungsdaten. Bilder, für die keine Annotationen gefunden wurden, sind in Abbildung 5.3 dargestellt. Sie haben meist großflächige Spiegelungen in der Pupille, welche aufgrund der beschränkten Datensätze nur schlecht trainierbar waren. Weiterhin sind in einigen Beispielen die Grenzen zwischen Pupille und Iris durch dunkle Iris-Farben oder eine schlechtes Lichtverhältnis nur schwer zu differenzieren. Weitere Verbesserungen am Datensatz könnten mehr Iterationen dieser Inferenzen sein. Letztlich wird zum Datensatz eine CSV-Datei erstellt, die die Bounding Box in relativen und absoluten Zahlen beinhaltet. Zudem werden die gesammelten Stammdaten wie die Iris-Farbe, der Hau ton und die relativen Größenangaben der Pupille und Iris gespeichert.

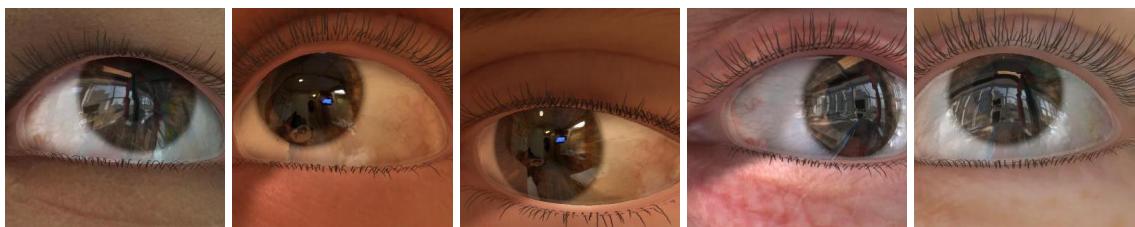


Abbildung 5.10: Beispiele der Pupillen, für die mit der zweiten Iteration des Pupillenmodells keine Annotation erzeugt werden konnte.

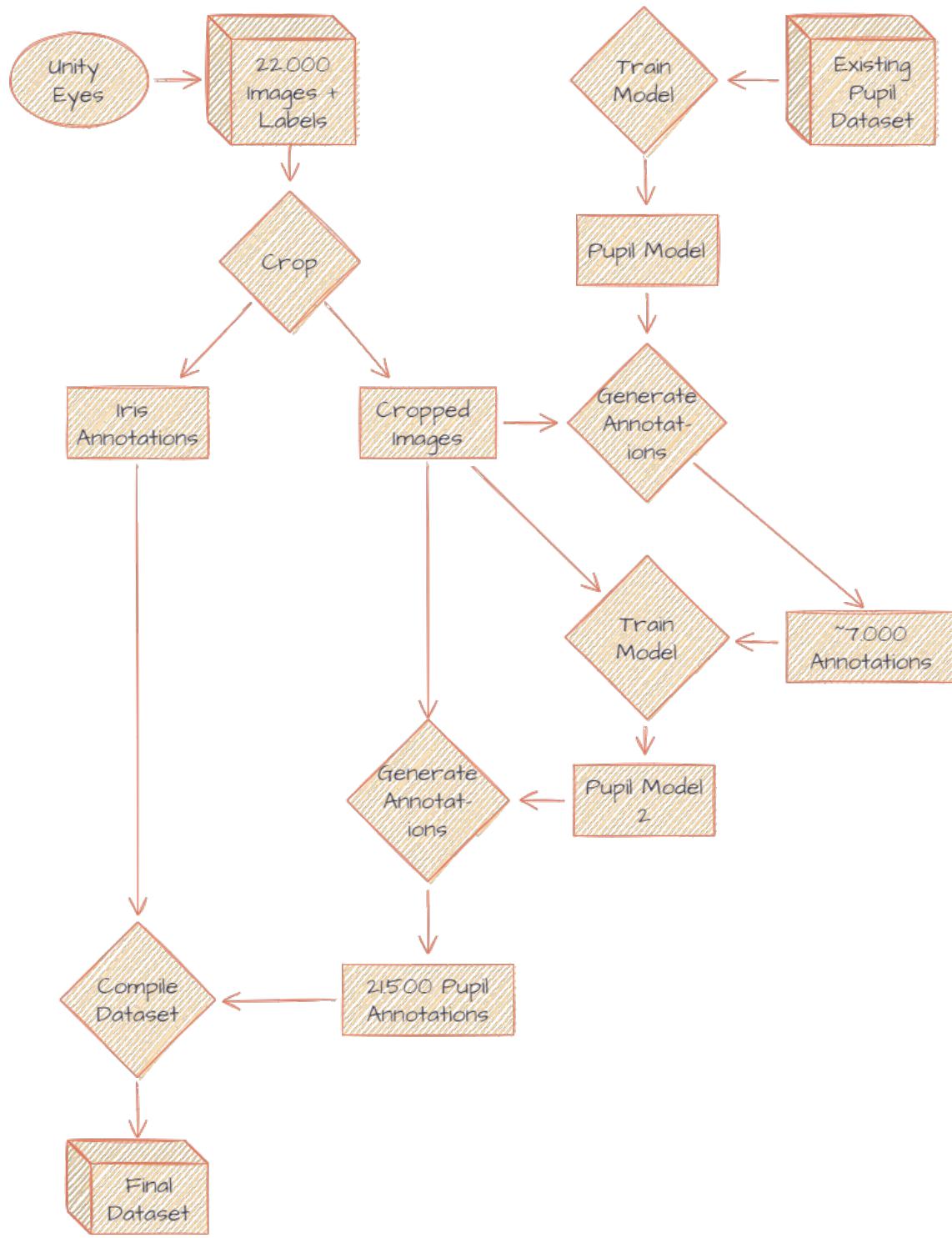


Abbildung 5.11: Erstellen des Iris- und Pupillendatensatzes aus dem Unity Eyes Modell unter Zuhilfenahme von Deep Learning zum Annotieren der Bounding Box für Pupillen.

6

KAPITEL

Experimente

6.1	Einordnung der YOLO Modell	60
6.2	Hyperparameteroptimierung	61
6.2.1	Erste Iteration: Random Search	62
6.2.2	Zweite Iteration: Bayes'sche Suche	64
6.2.3	Analyse der Batch Size	66
6.3	Finales Modell	68
6.4	Schrumpfen der Architektur	70
6.4.1	Aufbau der Modelle	71
6.4.2	Trainingsverhalten	71

In diesem Kapitel werden die Experimente zum Vergleich der verschiedenen YOLO Modelle auf dem eigens erstellten UnityEyes Datensatz durchgeführt. Es werden zu Beginn das Large, Medium, Small und Nano miteinander verglichen, um einen Ausgangspunkt und so eine Benchmark für weitere Optimierungen zu finden. Alle Experimente werden zehnfach wiederholt und über 20 Epochen trainiert, um eine Vergleichbarkeit und Konfidenz zu gewährleisten. Zudem haben vorherige Experimente in Kapitel 5 gezeigt, dass die Modelle in dieser Zeit ausreichend Training erfahren. Abgebildet sind die minimal und maximal erreichten Werte zu jeder Epoche; die σ -Umgebung wird nicht gezeigt, umfasst aber per Definition weniger Fläche. Der Abschnitt 6.2 führt erst die Ergebnisse einer zufälligen Suche nach Hyperparametern auf. Dann die Ergebnisse einer Bayes'schen Suche und beschreibt letztlich den Einfluss und die ideale Wahl dieser Parameter für das Nano Modell. Diese ideal gewählten Parameter werden für das Training weiterer Nano Modelle verwendet und mit der Benchmark verglichen. Im letzten Abschnitt wird das Nano Modell herunterskaliert und ein Training ohne COCO Pretraining durchgeführt. Die Modelle Shallow, Thin, Micro und Super Micro werden vorgestellt und zeigen eine radikale Reduktion der Parameter und Verbesserungen in dem Ressourcenverbrauch mit entsprechenden Verlusten in den Bewertungsmetriken.

6.1 Einordnung der YOLO Modell

Ultralytics stellt, wie in Abschnitt 3.6 beschrieben, verschiedene Modellgrößen von ihrem YOLOv5 samt COCO vortrainierter Gewichte zur Verfügung. Diese umfassen Xtra, Large, Medium, Small und Nano. Das Xtra Modell wird nicht weiter betrachtet, da es aufgrund der Komplexität und Inferenzgeschwindigkeit nicht für Echtzeitanwendungen in einem Smartphone geeignet ist. Laut Angabe von Ultralytics [NN59] benötigt es mit 205.7 GFLOPs ca. 45 mal mehr Leistung als das Nano Modell. Weiterhin sind die Performancewerte aus dem vorherigen Kapitel bereits bekannt und überschreiten die des Small Modells nur geringfügig.

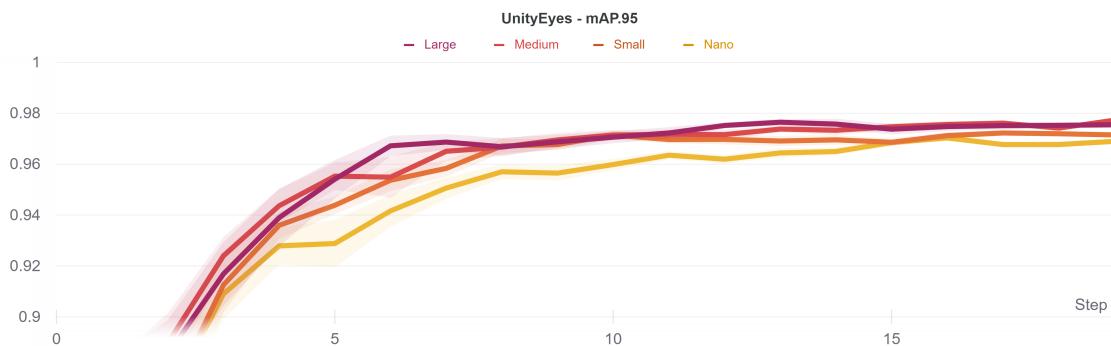


Abbildung 6.1: Die mAP.95 der vier verwendeten Größen des YOLO Modells auf dem UnityEyes Datensatz.

Auf der Abbildung 6.1 ist zu sehen, dass alle Modelle eine sehr hohe mAP.95 von 97-98 % auf dem neuen Datensatz erreichen. Dieser umfasst sowohl die Iris als auch die Pupille, weswegen auf eine zufriedenstellende Qualität des Datensatzes geschlossen werden kann. Die Large und Medium Modelle setzen sich mit 97.7 % respektive 97.8 % von Small und Nano, die jeweils 97.3 % und 97.1 % erreichten, ab. Während der ersten 15 Epochen verläuft die Trainingskurve des Nano Modell signifikant schlechter als die der anderen Modellen, kann aber am Ende wieder zu Small aufschließen. Die Precision bestätigt diese Beobachtung und zeigt ein ähnliches Bild, allerdings sind hier alle Modelle auf einem ähnlich guten Niveau zwischen 99.8 % und 99.9 %.

	AP@.50	mAP.95	Precision	Recall
Large	99.45 % \pm 0.03	97.69 % \pm 0.11	99.86 % \pm 0.04	99.28 % \pm 0.30
Medium	99.43 % \pm 0.04	97.82 % \pm 0.10	99.88 % \pm 0.03	98.75 % \pm 0.48
Small	99.46 % \pm 0.02	97.26 % \pm 0.11	99.87 % \pm 0.03	99.59 % \pm 0.11
Nano	99.39 % \pm 0.01	97.09 % \pm 0.10	99.81 % \pm 0.06	98.86 % \pm 0.62

Tabelle 6.1: Vergleich der Maximalwerte für jedes YOLO Modell mit einer σ -Umgebung.

Wie die Tabelle 6.1 zeigt, gibt es hinsichtlich der AP@.50 und dem Recall zwischen den Modellen keinen klaren Gewinner und keine Aussage, ob eine bestimmte Netzwerkgröße besser geeignet ist als die anderen. In der AP@.50 unterscheiden sich die Modelle

um maximal 0.07 Prozentpunkte. Der Recall fällt noch unklarer aus, da der beste Wert von Small verzeichnet wurde, der schlechteste von Medium und Nano. Letztere beiden Modelle haben jedoch im Kontext auch recht große σ -Konfidenzen, was weiter auf starke Schwankungen hindeutet. Es zeigt sich, dass die Messwerte aufgrund der kleinen Stichprobe und der insignifikanten Unterschiede der Netzqualität keinen klaren Gewinner ausweisen können.

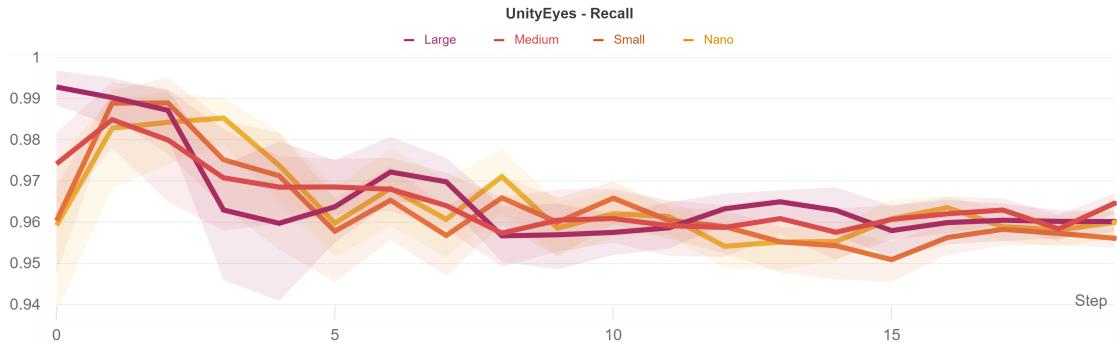


Abbildung 6.2: Der Recall der vier verwendeten YOLO Größen auf dem UnityEyes Datensatz.

Eine Beobachtung, die sich nicht aus der Tabelle ablesen lässt, da diese nur die Maximalwerte abbildet, kann in Abbildung 6.2 gemacht werden. Nach anfänglich sehr guten Werten in der AP@.50 und dem Recall fallen die Ergebnisse nach der dritten Epoche stark ab und erholen sich nicht. Das Finetuning auf den COCO Gewichten führt innerhalb der ersten wenigen Epochen zu einer sehr hohen Erkennungsrate, welches wiederum mit der niedrigen mAP.95 einhergeht, da zwar alle Objekte gefunden werden, diese Erkennungen aber keine hohe Genauigkeit aufweisen. Diese Beobachtung wird in Abschnitt 6.4 weiter untersucht. Die Präzision ist mit Werten von 99 % oder darüber für alle Modelle so hoch, dass der Recall nicht durch einen zu niedrigen IoU Schwellwert und damit einhergehenden FPs überproportional beeinflusst wurde.

Aus diesem Vergleich der standardmäßig vorgegebenen und vortrainierten Modelle von Ultralytics geht also hervor, dass keine maßgeblichen Unterschiede in der Performanz auf dem UnityEyes Datensatz bestehen. Es kann nachgewiesen werden, dass das Nano Modell, trotz der deutlich geringeren Leistungsansprüche von 4.5 GFLOPs, überzeugende Werte liefert und von allen verfügbaren Modellen das am besten geeignete ist, um auf ein Smartphone übertragen zu werden. Dementsprechend wird es für eine weitere Optimierung ausgewählt und die anderen Netze für die folgenden Experimente verworfen.

6.2 Hyperparameteroptimierung

Für das Training mit verschiedenen Hyperparametern wurden die Bibliothek Weights and Biases [17] und deren „Sweep“-Funktionalität verwendet. Dabei stellt Weights and Bi-

ses einen Server-Agent zur Verfügung, welcher das Training und die Parameter nach den vorgegebenen Einstellungen überwacht. Vom Anwender werden die Intervalle und die Verteilungen vorgegeben, aus denen die konkreten Werte der Hyperparameter gezogen werden. Weiterhin sind dabei die Metrik, für die optimiert werden soll und der Suchalgorithmus, welche in Abschnitt 4.4 dargelegt wurden, auszuwählen. In der ersten Iteration wird die zufällige Suche angewandt, um den hochdimensionalen Optimierungsraum einzugrenzen, da ein Random Forest, welcher serverseitig trainiert wird, die Einflussstärke auf und die Korrelation mit der gewählten Metrik berechnet. Als zu optimierende Metrik wird die mAP.95 gewählt, da diese das beste Mittel aus dem Recall, der Precision und der tatsächlichen IoU vorhersagt.

Um das Training automatisiert durchführen zu lassen, musste die `train.py`, welche von Ultralytics [NN59] mitgeliefert wird, um einen Wrapper, also eine Kompatibilitätsschicht, erweitert werden. Dieser Wrapper erstellt aus den Parametern, die beim Start des Trainingsscripts über die Konsole eingegeben werden, ein Dictionary (dt. Wörterbuch) in Form einer `hyperparameter.yaml` Datei und führt das Training mit diesen Parametern durch. Die Anzahl der Optimierungsschritte kann vor Beginn festgelegt oder manuell durch Stoppen des Vorgangs beendet werden. Bei der Erstellung eines Sweeps kann zwischen 86 Parametern gewählt werden, wobei einige Dopplungen und Einstellungsmöglichkeiten wie Zwischenspeicherschritte auftreten.

6.2.1 Erste Iteration: Random Search

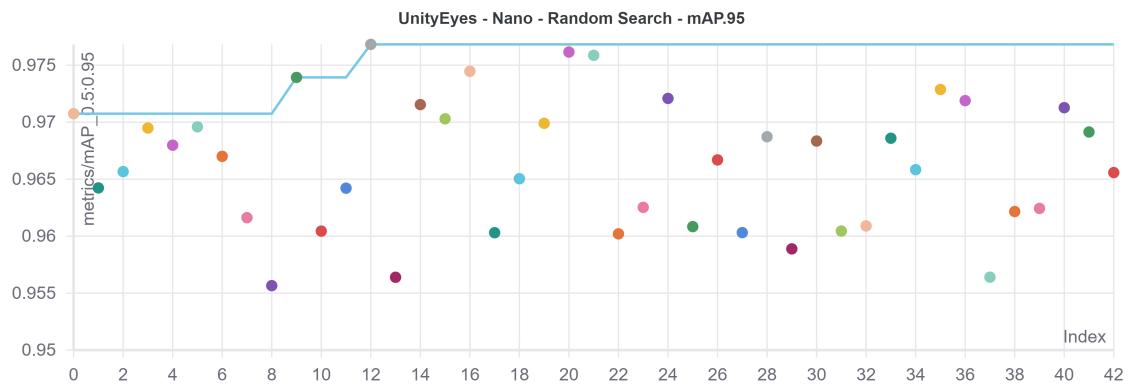


Abbildung 6.3: Die mAP.95 der getesteten, zufällig gezogenen Parameter für ein Nano Modell über 42 Stichproben.

Wie in Abbildung 6.3 gezeigt, werden die Hyperparameter zufällig gezogen und die erreichten Werte folgen keinem Muster. Nach nur zwölf der 42 Schritte wird der Maximalwert von 97.7 % mAP.95 erreicht; die anderen Werte liegen im Intervall zwischen diesem und dem Minimalwert von 95.6 % mAP.95. Es ist dabei zu beachten, dass dies nur Stichproben sind und keine konfidente Aussage über die gewählte Konfiguration treffen können. Die beste Konfiguration in Iteration zwölf kann mit Training eines anderen Seeds schlechtere oder bessere Werte, als die nächstbesten Werte während Iteration 20 und

21, bieten. Mit Hilfe dieser Messreihe wird quantitativ untersucht, wie stark der Einfluss der jeweiligen Parameter auf das Trainingsverhalten ist.

In Tabelle 6.2 sind die Stärke und die Korrelation der jeweiligen Parameter angegeben. Es wird dabei in Parameter, die für das Training des Netzes Verantwortung tragen und Parameter, die die Eingabe von Trainingsdaten beeinflussen, differenziert und nach ihrer jeweiligen Stärke sortiert. Die Korrelation drückt aus, wie konsistent die Messwerte mit der Vorhersage übereinstimmen; ist sie positiv, sollte der Parameter größer gewählt werden. Analog dazu sollte er kleiner gewählt werden, wenn die Korrelation negativ ist. Wird der Parameter allerdings zu groß oder klein gewählt, kann die Korrelation wieder sinken. Ist ein Idealwert gefunden, so sinkt die mAP.95 sowohl, wenn der Wert größer als auch kleiner wird, was eine Korrelation nahe null zur Folge hat.

Training	Stärke	Korrelation	Eingabe	Stärke	Korrelation
Class Loss Gain	0.419	-0.657	FlipLR	0.144	-0.447
Box Loss	0.072	0.290	Scale	0.084	-0.252
Learning Rate 0	0.043	0.182	Translate	0.050	-0.073
Momentum	0.020	0.211	Batch Size	0.042	0.289
Weight Decay	0.019	-0.097	Color Value	0.032	0.064
Learning Rate F	0.018	0.028	Color Saturation	0.027	0.054
IOU Threshold	0.018	-0.050	Color Hue	0.025	-0.250

Tabelle 6.2: Die Auswirkungen der 14 betrachteten Hyperparameter auf die mAP.95. Eingeht in Trainingsparameter und Parameter, die die Eingabe der Daten verändern.

Aus der Tabelle 6.2 geht also hervor, dass besonders der Class Loss Gain, welcher die Stärke des Gradienten skaliert, einen Einfluss auf das Trainingsverhalten hat. Mit einer Stärke von etwa 0.42 muss dieser Wert angepasst werden, da er einen signifikanten Einfluss auf das Training hat. Da die Korrelation negativ bei -0.657 liegt, sollte dieser also reduziert werden, was das Training verlangsamt, da die Gradienten kleiner ausfallen. Der Box Loss, Learn Rate 0 und das Momentum haben jeweils nur einen kleinen Einfluss und sind beide positiv korreliert, was bedeutet, dass diese beiden Werte für die nächste Iteration angehoben werden sollten. Die anderen Werte auf Seiten der Trainingsparameter haben nur einen sehr schwachen Einfluss und werden nicht weiter besprochen.

Seitens der Eingabeparameter weisen die FlipLR, Scale und Translate Parameter mit ihrer negativen Korrelation darauf hin, dass nicht zu viele Bilder gespiegelt, skaliert oder auf der Leinwand verschoben werden sollten. Eine größere Batch Size wird von der Auswertung empfohlen, genauso wie Variationen an dem HSV-Farbraum [3] an Color (dt. Farbe) und Saturation (dt. Sättigung). Der Hue sollte gesenkt werden.

Mit dieser Analyse wurde eine Baseline für den Einfluss der verschiedenen Hyperparameter gelegt, wodurch für das nächste Experiment neue Intervalle gezogen werden. Die Änderungen werden wie vorgestellt vollzogen und für die Bayes'sche Suche verwendet.

Die Parameter Learning Rate F und der IOU Threshold werden für weitere Optimierungen ausgeschlossen, da Änderungen keinen starken Einfluss haben und die Korrelationen nahe null sind. Sie sind somit entweder auf einem Optimum oder beeinflussen das Training nicht.

6.2.2 Zweite Iteration: Bayes'sche Suche

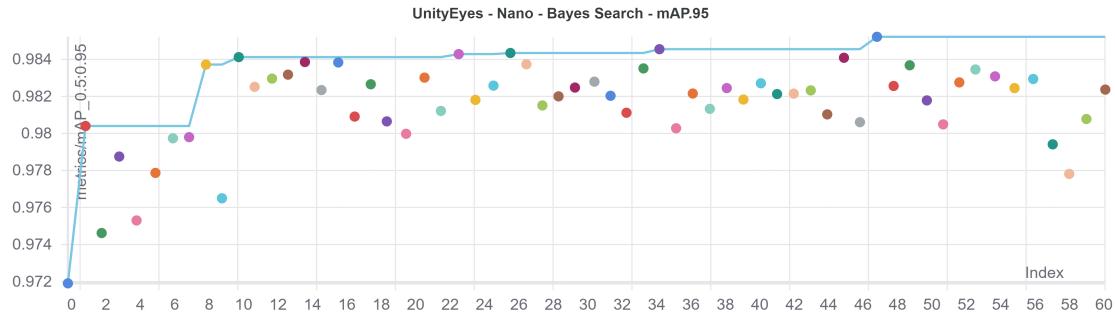


Abbildung 6.4: Die mAP.95 der getesteten Parameter für ein Nano Modell über 60 Stichproben in der Bayes'schen Suche.

Auf der Abbildung 6.4 ist die Bayes'sche Suche für Hyperparameter zu sehen. Im Vergleich zur zufälligen Suche fällt das Charakteristikum einer informierten Suche auf: besonders zu Beginn verbessern sich die Ergebnisse und bleiben auch auf einem erhöhten Niveau. Dies ist damit zu erklären, dass mit Hilfe der Random Forests bereits während des Trainings die Parameter bewertet werden und Prognosen gestellt werden, welche Konfigurationen vielversprechend sind. War der beste erreichte Wert unter den 42 Stichproben in der Zufallssuche noch 97.7 % mAP.95, könnte sich dieser Wert nur noch gegen vier der 60 neuen Stichproben behaupten. Das neue Maximum erreicht 98.5 % mAP.95 nach 20 Epochen Trainingszeit.

Training	Stärke	Korrelation	Eingabe	Stärke	Korrelation
Class Loss Gain	0.442	-0.724	Scale	0.200	0.500
Box Loss	0.068	0.352	Color Saturation	0.038	0.142
Momentum	0.043	0.165	Batch Size	0.035	0.181
Learning Rate 0	0.042	-0.317	FlipLR	0.031	0.048
Weight Decay	0.030	-0.025	Color Value	0.026	0.156
			Color Hue	0.025	0.013
			Translate	0.019	-0.017

Tabelle 6.3: Die Auswirkungen der 12 betrachteten Hyperparameter auf die mAP.95. Eingeht in Trainingsparameter und Parameter, die die Eingabe der Daten verändern.

In der Tabelle 6.3 sind erneut die Einflussstärken und Korrelationen der Hyperparameter auf die Trainingsgenauigkeit gezeigt. Erneut hat der Class Loss Gain den größten Einfluss auf das Trainingsverhalten und ist noch immer stark negativ korreliert. Der Einfluss

anderer Trainingsparameter liegt eine Größenordnung niedriger und ist dementsprechend unbedeutsam für die Genauigkeit. Wie allerdings in Abbildung 6.5 zu sehen ist, liegen höhere Dichten der Auswahl von Stichproben für den Box Loss bei einem Wert von etwa 0.11 und beim Momentum um 0.89 vor. Dies impliziert, dass hier die besten Werte vermutet werden, ihr Einfluss allerdings von dem Class Loss Gain dominiert wird. Für die Learn Rate 0 und den Weight Decay können keine vergleichbaren Dichteverteilungen ausgemacht werden. Es wird also impliziert, dass alle Werte in den jeweils gewählten Intervallen ausreichend gut gewählt sind, um das Training nicht zu beeinträchtigen.

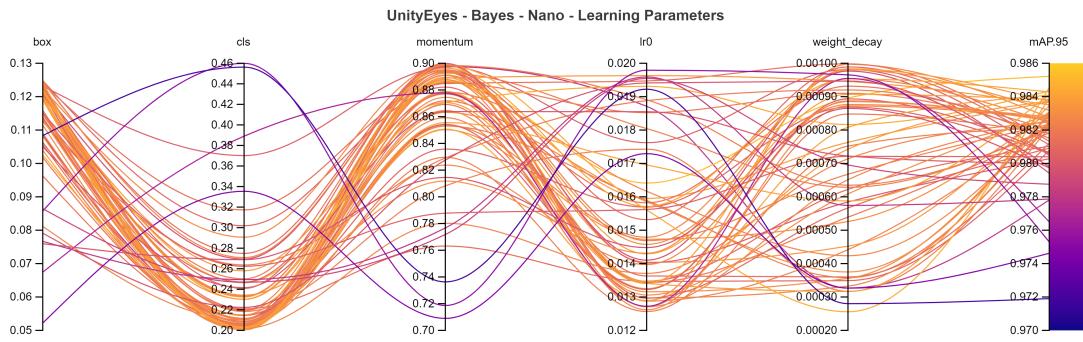


Abbildung 6.5: Parallel Coordinate Plot der überprüften Trainingsparameter während der Bayes'schen Suche. Die Farbe repräsentiert die erreichte mAP.95.

In der Abbildung 6.6 ist kein derartig klares Bild zu erkennen. Für Batch Size wurden von der Suche insgesamt Werte am oberen Ende des Intervalls präferiert, für die Spiegelung eher im unteren Bereich. Für die Skalierung, welche laut Tabelle 6.3 den stärksten Einfluss hat, wurden große Werte bevorzugt.

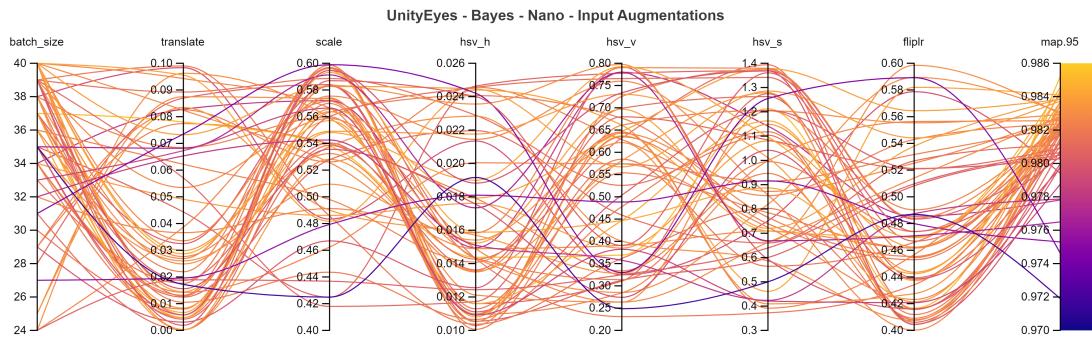


Abbildung 6.6: Parallel Coordinate Plot der überprüften Eingabeparameter während der Bayes'schen Suche. Die Farbe repräsentiert die erreichte mAP.95.

Insgesamt sollten mit dieser Bayes'schen Suche Parameter für das Training eines optimierten Nano Modells gesucht werden. Für die Auswahl der Annäherungen an die beste Konfiguration verwendet. Besonders bei Parametern mit geringer Korrelation und Einflussstärke wäre eine Auswahl anhand der verfügbaren Daten nicht nachhaltig zu begründen. Für Parameter mit einem sehr hohen Einfluss wurden leichte manuelle Anpassungen getroffen, die aber keine bedeutsame Abweichung von der angesprochenen Konfiguration haben. In der Tabelle 6.4 ist die so ermittelte Konfiguration abgebildet.

Training	Wert	Eingabe	Wert
Class Loss Gain	0.2	Batch Size	38
Box Loss Gain	0.123	FlipLR	0.43
Learning Rate 0	0.0165	Scale	0.54
Learning Rate F	0.02	Translate	0.08
Momentum	0.89	Color Hue	0.014
Weight Decay	0.0009	Color Value	0.37
IOU Threshold	0.017	Color Saturation	1.2

Tabelle 6.4: Finale Konfiguration für das Training des Nano Modells.

6.2.3 Analyse der Batch Size

Da die Batch Size im Training regulär einerseits keinen nennenswerten Einfluss auf die Trainingsgenauigkeit nimmt und andererseits das Training durch eine effektivere Ausnutzung der Grafikkarte beschleunigen kann, wurde hierzu eine kurze Messreihe aufgenommen. Die Versuche wurden nur sechs mal wiederholt und sind dementsprechend eher qualitativ zu betrachten. Das Training wurde mit dem Nano Modell ausgeführt, die Batch Sizes sind logarithmisch verteilt, um eine große Spanne mit wenig Stichproben abzudecken. Das Maximum beträgt 256, da die Speicheranforderungen sonst die verfügbaren Ressourcen übersteigen. Das Training wird ansonsten mit den oben angegebenen Parametern für das optimierte Netz durchgeführt. In Abbildung 6.7 sind deutliche Verbesserungen in der Geschwindigkeit bis zu einer Batch Size von 32 zu verzeichnen.

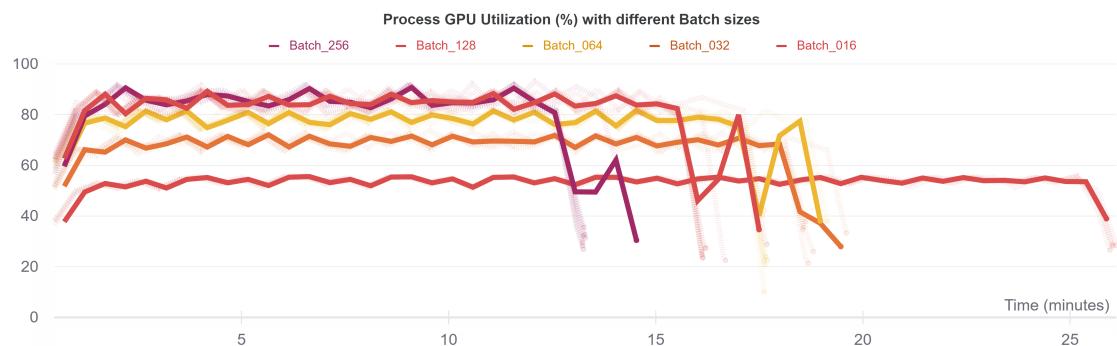


Abbildung 6.7: Die GPU Auslastung und benötigte Zeit für das Training von 20 Epochen des Nano Modells.

Auf der anderen Seite ist ein deutlich inkonsistenteres Training auf der Abbildung 6.8

zu sehen. Während der Verlauf mit Batch Sizes von 8, 16, 32 und 64 Bildern nahezu identisch verläuft, kann das 128 Batch Modell erst nach 14 Epochen, das 256 Batch Modell nicht innerhalb der 20 Epochen zu den anderen aufschließen. Es wird hier also gezeigt, dass kleine bis mittlere Batch Sizes einen sehr ähnlichen Trainingsverlauf haben, größere hingegen benötigen mehr Epochen, um eine ähnliche Genauigkeit zu erreichen.

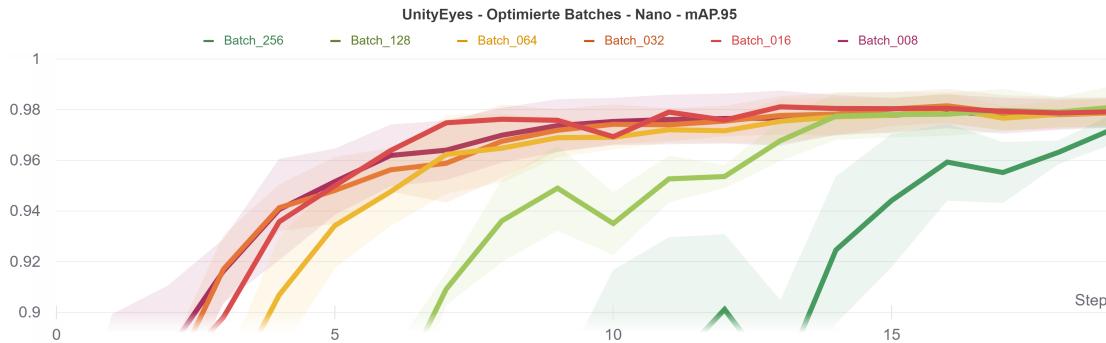


Abbildung 6.8: Das Trainingsverhalten über 20 Epochen des Nano Modells mit unterschiedlichen Batch Sizes und optimierten Hyperparametern.

Die Tabelle 6.5 gibt an, dass in diesem Szenario eine Batch Size von mindestens 32 gewählt werden sollte, da es die Trainingszeit pro Epoche mehr als halbiert. Weiterhin erzielt es mit 98.15 % die beste mAP.95, auch wenn die statistische Aussagekraft dieser Messreihe zweifelhaft ist. Qualitativ lässt sich weiterhin sagen, dass in dem Training mit einer Batch Size von 128 nur eine geringe Verbesserung im Zeitverhalten gegenüber 32 und 64 zu verzeichnen ist. Das Trainingsergebnis ist nur von dem 256 Batch markant schlechter als von den anderen getesteten Konfigurationen. Eine letzte interessante Feststellung ist, dass das Speicherverhalten nicht wie erwartet exakt linear skaliert.

Batch Size	Memory Usage	Time per 20 Epochs	mAP.95
256	6.04 GB	13:10 min	97.26 %
128	4.62 GB	16:01 min	98.10 %
64	2.25 GB	17:30 min	98.05 %
32	1.16 GB	18:29 min	98.15 %
16	0.60 GB	25:58 min	98.11 %
8	0.28 GB	44:15 min	97.94 %

Tabelle 6.5: Trainingsverlauf und Ressourcenverhalten von einem Nano Modell bei verschiedenen Batch Sizes

6.3 Finales Modell

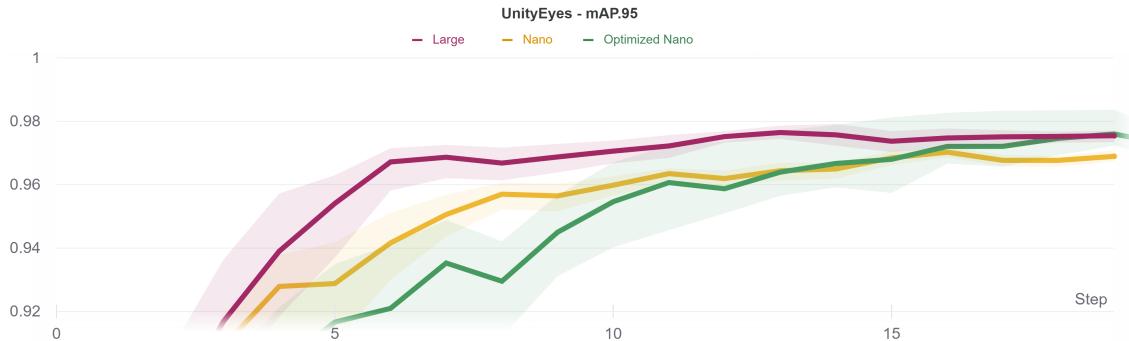


Abbildung 6.9: Die mAP.95 für Large, Nano und optimiertes Nano auf dem Unity Datensatz über 20 Epochen.

Das finale und optimierte Nano Modell erreicht bei weniger als einem Zwanzigstel der Parameter und GFLOPs sehr vergleichbare Werte in der mAP.95 mit dem Large Modell von Ultralytics. Es hat dabei allerdings deutlich größere Schwankungen, die bis an die 98.5 % aus der Bayes'schen Suche heranreichen. Dieser Wert kann im Mittel aber nicht reproduziert werden. Eine Verbesserung um über 1 % ist allerdings trotz dessen ein sehr gutes Ergebnis für die Hyperparameteroptimierung. In der Abbildung 6.9 ist auch zu sehen, dass, im Einklang mit den Ergebnissen aus der Untersuchung der Batch Sizes, das Training ein wenig langsamer stattfindet. Es dauert 12 Epochen bis es zum Nano, 19 Epochen bis es zum Large Modell aufgeschlossen hat.

Die Precision, welche in Abbildung 6.10 illustriert ist, zeigt erneut die großen Schwankungen. Zudem ist ernüchternd, dass die Precision insgesamt niedriger ausfällt, als sowohl im Large als auch im Nano Modell. Diese beiden sind mit etwa 99.8 % vergleichbar; das optimierte Modell fällt mit maximal 99.6 % ein wenig zurück.

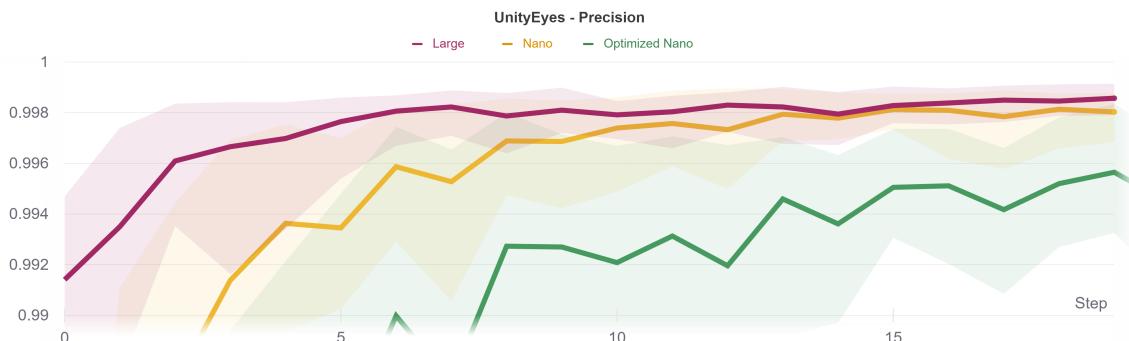


Abbildung 6.10: Die Precision für Large, Nano und optimiertes Nano auf dem Unity Datensatz über 20 Epochen.

Der Recall und die AP@.50 hingegen zeigen ein anderes Bild. In Abbildung 6.11 bleibt das Modell, wenn es auf einem hohen Level angekommen ist, auch recht konstant auf diesem Niveau. Während der Anfang besonders in der AP@.50 langsamer trainiert als in den unoptimierten Modellen, hat es dort ab der zehnten Epoche mit 99.4 % im Mittel

einen 1.5 % besseren Wert. Für den Recall zeichnet sich ein ähnliches Bild ab: Während der Recall von Large und Nano von knappen 99 % auf 96 % abfällt, können vom optimierten Modell die 99 % gehalten werden.

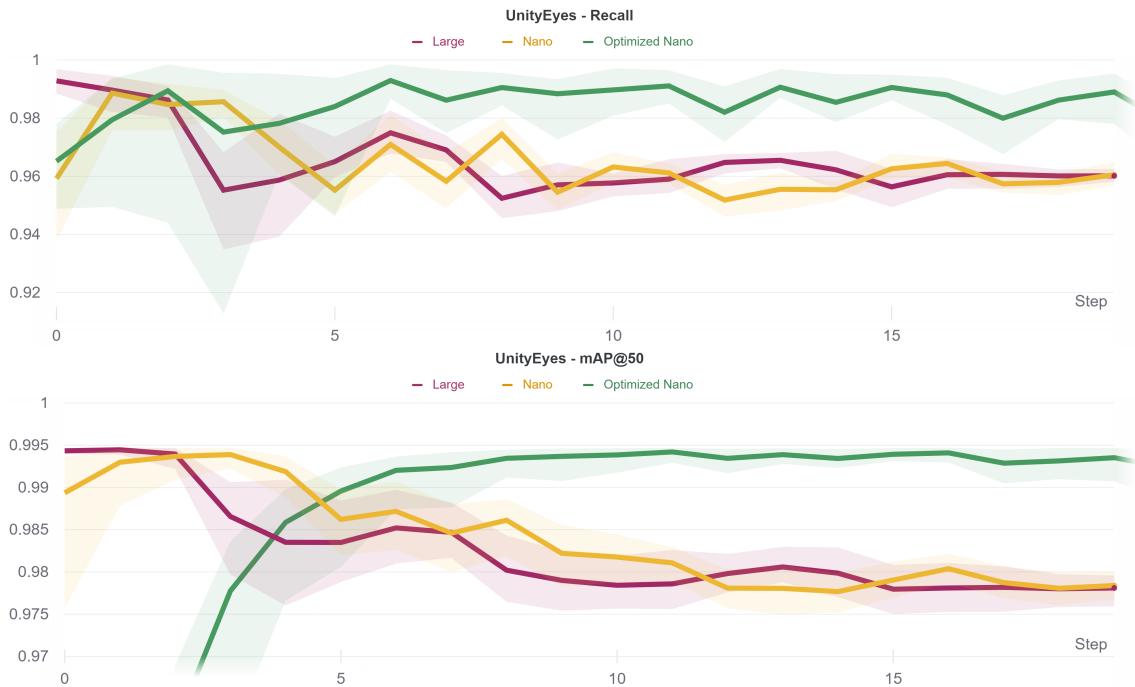


Abbildung 6.11: Der Recall und die mAP.95 für Large, Nano und optimiertes Nano auf dem Unity Datensatz über 20 Epochen.

Insgesamt kann neben der Verbesserung in der AP@.50, mAP.95, und dem Recall und der Verschlechterung der Precision durch die Hyperparameteroptimierung folgendes festgehalten werden: Das Training findet langsamer statt. Die drastische Verringerung des Class Loss Gains und der Learning Rate 0 bei gleichzeitiger Erhöhung der Batch Size wirken sich verlangsamt auf das Training aus. Dies ist Folge dessen, dass als Entscheidungskriterium für die Optimierung die mAP.95 punktuell nach der zwanzigsten Epoche gewählt wurde. Dadurch wurde implizit vorausgesetzt, dass ein Overfitting an die Daten erst nach diesem Punkt stattfinden darf.

Auf der anderen Seite konnte durch die Optimierung ein signifikant besserer mAP.95 Wert im Training erreicht werden, sodass das Modell auch mit dem deutlich größeren Large Modell konkurrieren kann. Eine weitere Optimierung, die eine flexible Anzahl von Epochen integriert, wird an dieser Stelle nicht durchgeführt, da eine Verbesserung der mAP.95 zu einem zufriedenstellenden Maß erreicht wurde. Es bleibt also noch aus, die richtigen Schwellwerte für eine Implementierung zu finden.

In der Abbildung 6.12 sind der F_1 -Score und die Precision der einzelnen Klassen für das optimierte Modell mit dem höchsten Mittel aus Präzision, Recall und mAP.95 zu sehen. Links ist zu sehen, dass die Precision erst bei einem Konfidenzwert von 92.5 % der Inferenz des Modells keine Fehlerkennungen für beide Klassen misst. Ab einer Konfidenz

von bereits 55 % werden Werte nahe einer perfekten Erkennung, die als Kompromiss betrachtet werden können, geliefert. Dieser Wert lässt sich mit dem Graphen rechts bestätigen. Bei einer Gleichgewichtung von Precision und Recall ergibt sich das Maximum von 98 % bei einem Konfidenzschwellwert von 54.1 %. Ohne weitere Annahmen und Anforderungen kann dieser Schwellwert für die Implementation in einer Anwendung genutzt werden. Da aber die Annahme getroffen werden kann, dass die Iris bei der Augenerkennung die Größe nicht verändert, können die Ergebnisse über mehrere Messungen interpoliert werden, wodurch im Mittel ein stabiler Referenzwert in Form der Iris gebildet werden kann. Unter dieser Annahme kann der Konfidenzschwellwert ein wenig vergrößert werden.

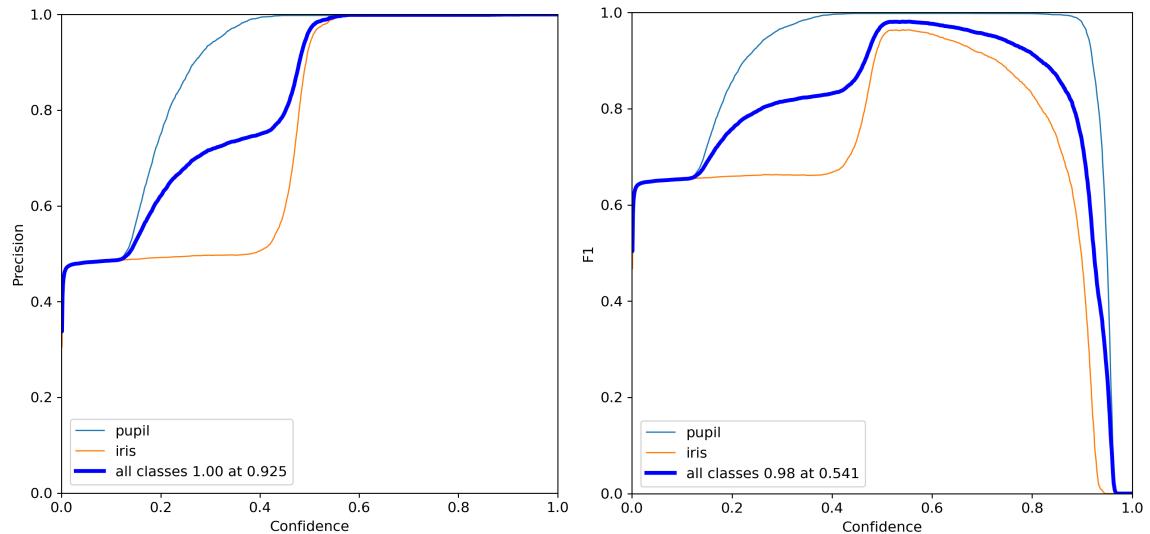


Abbildung 6.12: Precision und $F_1 - Score$ für das finale Nano Modell.

6.4 Schrumpfen der Architektur

Ultralytics bietet das YOLO-Modell in den fünf bekannten Größen, jeweils mit vortrainierten COCO Gewichten, an. Die Architektur dabei bleibt jeweils dieselbe; der Unterschied zwischen den Modellen beläuft sich auf die Tiefen- und Breitenskalierung der Schichten zwischen der Eingabe und der Ausgabe. Die Block-Convolutionen können weniger häufig wiederholt werden und aus weniger einzelnen Schichten bestehen, wodurch die Tiefe abnimmt. Die Breite hingegen kann durch eine Reduktion der Filter gesenkt werden. In diesem Abschnitt werden vier neue Varianten von YOLO vorgestellt, welche die Anzahl der Parameter und die Berechnungszeit verringern sollen. Ziel ist es, die Erkennungsqualität mit der Performanz des Modells abzuwägen, um so auch für leistungsschwache Geräte ein passendes Modell zu schaffen.

6.4.1 Aufbau der Modelle

Der Aufbau der Modelle ist jeweils in einer `.yaml` Datei definiert. Diese Dateien enthalten neben der Ein- und Ausgabeschicht und der Anzahl der Klassen, auf die trainiert werden soll, auch das `depth-scaling` und das `width-scaling` als konfigurierbare Faktoren. Das Xtra-Modell wird als normalisierte Referenz herangezogen, welche für beide Faktoren den Wert 1.0 gegeben hat. Nano hingegen setzt auf die Skalierungsfaktoren 0.33 respektive 0.25 für die Tiefe und Breite.

Für diesen Ansatz wird analysiert, welche Auswirkungen eine Reduktion der jeweiligen Faktoren mit sich führt. Das Nano Modell wird dabei als Referenzwert für die neuen Modelle Shallow, Thin, Micro und Super Micro verwendet. Wie die Namen bereits suggerieren, ist die Tiefenskalierung für das Shallow Modell verringert worden, für Thin die Breitenskalierung. Micro vereint diese beiden Reduktionen und Super Micro halbiert die Faktoren noch einmal. Die exakten Werte, besonders auch die Reduktion der Parameter von 1.76 Millionen auf 94.6 Tausend bei nur noch einem Zehntel der notwendigen Gleitkommarechnungen für das Super Micro Modell, sind in der Tabelle 6.6 hinterlegt. Als weiterer wichtiger Unterschied zu der bisherigen Analyse ist hervorzuheben, dass nur für den Header und den Output vortrainierte Gewichte zur Verfügung stehen. Dementsprechend wird hier auch untersucht, wie ein Training mit nur wenigen vortrainierten Gewichten verläuft.

Model	Depth Scaling	Width Scaling	Layers	Parameters	GFLOPs
Nano	0.33	0.25	270	1.766.623	4.2
Shallow	0.125	0.25	243	1.673.823	3.8
Thin	0.33	0.1	270	329.207	1.0
Micro	0.125	0.1	243	310.679	0.9
Super Micro	0.05	0.05	192	94.651	0.4

Tabelle 6.6: Vergleich der Skalierungsfaktoren und deren Auswirkungen auf die Größe und benötigte Rechenleistung.

6.4.2 Trainingsverhalten

In Abbildung 6.13 wird die mAP.95 der fünf Modelle auf dem UnityEyes Datensatz visualisiert. Das Nano und Shallow Modell liefern insignifikante Unterschiede und erreichen beide Werte knapp über 96 %. Diese beiden Modelle haben allerdings auch eine sehr vergleichbare Anzahl von Parametern, was die beieinander liegenden Werte erklärt. Das Thin Modell erreicht mit etwa einem Fünftel der Parameter eine Genauigkeit von 95 %, ein signifikant besserer Wert, als der des Micro Modells, welches trotz ähnlicher Parameterzahl nur 94 % erreicht. Wird das Modell ein weiteres Mal um den Faktor drei kondensiert, erreicht es im Mittel nur noch 90 % mAP.95.

Es ist also zu beobachten, dass die Performanz selbst mit einem Zehntel der Rechen-

KAPITEL 6. EXPERIMENTE

operationen und einem Zwanzigstel der Parameter noch brauchbar ist . Hinsichtlich der fehlenden Gewichte kann festgehalten werden, dass das Training des Nano Modells mit einer Batch Size von 40 deutlich länger Verbesserungen zeigt, als in den vorherigen Versuchen.

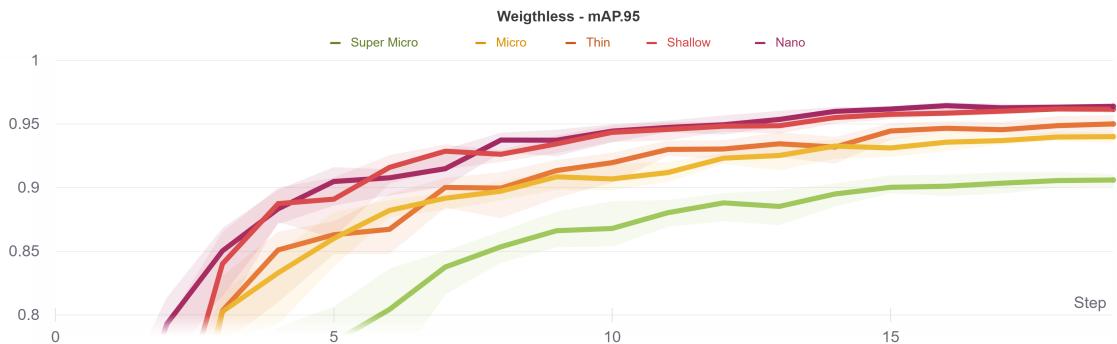


Abbildung 6.13: Die mAP.95 der verkleinerten Modelle auf dem UnityEyes Datensatz.

Der Recall, welcher in Abbildung 6.14 gezeigt wird, gibt auch ein anderes Bild, als die vorherigen Trainingsabläufe. Statt wie bisher startet das Nano Modell nicht mit nahezu 100 %, sondern hat erfährt ein richtiges Training, das dieses mal langsamer wieder abfällt. Das Verhalten vom Shallow, Thin und Micro Modell ist sehr ähnlich, das Super Micro hingegen weist auch bis Ende der 20 Epochen einen Recall von über 98 % auf und ist somit besser als die anderen Modelle, trotz der geringen Größe.

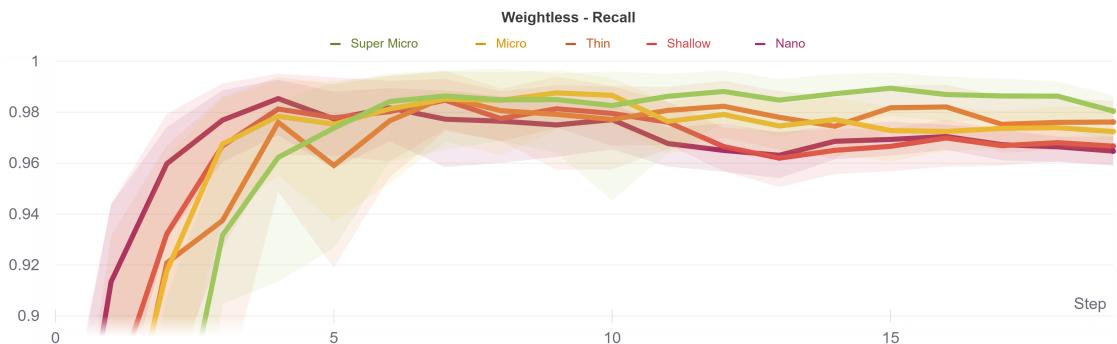


Abbildung 6.14: Der Recall der verkleinerten Modelle auf dem UnityEyes Datensatz.

7

KAPITEL

Android Anwendung

7.1	Änderungen der Implementation	74
7.1.1	Kotlin	74
7.1.2	Änderungen in der Kameraimplementierung	75
7.2	PyTorch Mobile	75
7.2.1	Export	75
7.2.2	Benchmark	76

In dieser Arbeit wird ein Modell für die Iris- und Pupillenanalyse optimiert, welches in einer Android Anwendung Einsatz finden soll. Diese Anwendung basiert auf der Arbeit von *Moryäner* [26], verwendet die meisten Assets wieder und behält den generellen Aufbau bei. Wie in den Grundlagen beschrieben wird allerdings das Ziel der API-Version von Android von Version 4, Ice Cream Sandwich, auf 8.1, Oreo, angehoben. Oreo bietet neue Funktionen und Optimierungen, die besonders in der NNAPI widergespiegelt werden. Diese API für neuronale Netze wurde in Kapitel 3.2.1 erläutert und ermöglicht durch weitreichende Optimierungen und der Implementation einer NPU auf der SoC Verbesserungen in der Inferenzgeschwindigkeit. Diese Verbesserungen sollen in der entstehenden App eingebbracht werden, indem eine Echtzeiterkennung, also eine Auswertung bereits während der Bildaufnahme, ermöglicht werden soll. Dies soll die Bedienbarkeit und Nutzererfahrung verbessern, wodurch eine höhere Akzeptanz für eine künftige Anwendung geschaffen werden soll.

7.1 Änderungen der Implementation

In der Implementierung dieser Arbeit werden drei große Änderungen im Vergleich zu der bestehenden Arbeit vorgenommen:

1. **Übersetzung zu Kotlin:** Bisher wird für die Codebasis Java verwendet. Kotlin bietet einige Komfortfunktionen und wird von Google als präferierte Sprache für die Android Entwicklung tituliert.
2. **Kamera API:** Die Implementierung der Kamera verwendet noch die veraltete Camera API. Die neue CameraX bietet Vorteile besonders hinsichtlich einer weiteren Analyse der Aufnahmen.
3. **PyTorch Mobile:** Während die alte Arbeit mit einem SSDLite MobileNetV3 auf der Tensorflow Plattform aufbaut, soll in dieser Arbeit eine Implementation des YOLO Modells auf PyTorch Basis stattfinden.

7.1.1 Kotlin

Die erste zentrale Änderung besteht aus einer Umstellung von Java auf Kotlin. Weitere kleinere Anpassungen, die für einen Komfort in der Programmierung sorgen, werden damit eingeschlossen. Kotlin ist in Zusammenarbeit von Google und JetBrains, dem Entwickler der Android Studio IDE, entstanden und wurde 2017 als Alternative zur Programmierung von Android Apps vorgestellt [11]. Bereits 2019 hat Google den Status von Kotlin zur präferierten Programmiersprache in der App Entwicklung angehoben [15]. Die Sprache ist deutlich weniger verbos, benötigt also für die gleichen Algorithmen weniger Code. Zur Verbesserung tragen Daten-Klassen, Einflüsse der funktionalen Programmierung, wie Lambda- und Scope-Funktionen, und abschließend eine verbesserte Null-Pointer Sicherheit bei. Das erhöht die Lesbarkeit und Wartbarkeit des Codes.

Kotlin setzt auf die Interoperabilität mit Java und baut entsprechend auch auf der JVM auf. Die JVM bietet den entscheidenden Vorteil, dass sie einen eigenen, virtuellen Bytecode verwendet. Dadurch muss die Runtime Environment einmalig für die Plattform angepasst werden, die Anwendung selbst bleibt davon unbetroffen. So ist es potentiellen Entwicklern einfacher ihre Produkte auf verschiedene Geräte zu portieren, da ihnen nötige Anpassungen, je nach Hardware- und Software-Gegebenheiten, erspart werden. Android Studio bietet die Möglichkeit den Java-Code automatisiert in Kotlin zu übersetzen, liefert dabei aber keine perfekten Ergebnisse. Meist werden noch händische Optimierungen benötigt, um die Lesbarkeit und Wartbarkeit des Codes zu verbessern. potentielle Performanzoptimierungen zu implementieren.

Eine nebensächliche Änderung, die implementiert wird, ist das View Binding. Dieses Binding ermöglicht einen direkten Zugriff auf die Views in der XML Layout Datei der Activity und deren Properties. Indirekt wird so die `findViewById` Funktion ersetzt.

7.1.2 Änderungen in der Kameraimplementierung

Neben einer Umstellung auf eine moderne API Version, wird auch eine neue Versionen der Kamera API verwendet. CameraX kommt mit Vereinfachungen in der Programmierung, Konsistenz über verschiedene Geräte und API Versionen hinweg und, besonders relevant für diese Arbeit, der eingebauten Image analysis (dt. Bildanalyse) Klasse. Mit dieser Klasse kann das Echtzeit-Kamerabild für ein DL Modell vorverarbeitet werden. So können Auflösung, Rotation, Filter oder weitere Augmentationen nativ auf den Videostream angewandt werden.

In der Funktionsweise soll die Restriktion, dass beide Augen gleichzeitig zu sehen sein müssen, gelöst werden. Falls benötigt kann so eine höhere Auflösung des jeweiligen Auges erfasst werden und es benötigt weniger Vorverarbeitungsschritte, da die Augen selbst nicht zuerst erkannt werden müssen. Weiterhin kann mit diesem Ansatz eine Unterscheidung in afferente und efferente Schäden bei einem Schädel-Hirn-Trauma, wie in Kapitel 3.1.3 beschrieben, vorgenommen werden. Um die Größe der Pupille zu bestimmen, kann in der neuen Implementation die relative Größenveränderung der Bounding Box im Vergleich zu der der Iris verwendet werden. Es entstehen dabei keine weiteren Rechenkosten, da das Modell beide Bounding Boxen in dem gleichen Schritt verwendet. Bis Ende der Bearbeitungszeit konnten nicht alle angestrebten Verarbeitungsschritte der Anwendung umgesetzt werden. Letztlich werden aber aufgetretene Hindernisse aufgeführt und eine Empfehlung über geeignete Modellgrößen getätigt.

7.2 PyTorch Mobile

PyTorch stellt mit ihrer Mobile Bibliothek eine Plattform zum Ausführen der trainierten Modell bereit. Diese Plattform soll dem Nutzer die Kompatibilität zur aktuellen NNAPI gewährleisten und bietet eigene Funktionen zur Vorberarbeitung der Daten. Um die Modelle nach dem Training auf das Smartphone zu bringen, müssen sie aber erst als PyTorch Lite .ptl, analog zum Tensorflow Lite Format, exportiert werden.

7.2.1 Export

Der Export und das Einbinden eines YOLOv5 Modells sollte laut Angaben von Ultralytics sehr einfach funktionieren. Sie liefern dafür auch ein entsprechendes Script zum Export des Modells für verschiedene Formate mit. Darin enthalten sind neben PyTorch Lite auch Formate für Tensorflow und Tensorflow Lite. In einem naiven Test sind das Ausgabeformat dieses Exportscripts und das Format, welches in der bestehenden Arbeit verwendet wird, trotz Kompatibilität mit Tensorflow inkompatibel. Das SSDLite MobileNetV3 hat eine vier-geteilte Ausgabe, welche von dessen Interpreter erwartet und verarbeitet wird. Das YOLOv5 hingegen hat als Ausgabe einen einzelnen Vektor, welcher bereits in Kapitel 3.6 vorgestellt wurde. Der Interpreter ist nicht anpassbar, weswegen auf den PyTorch Interpreter umgestiegen wird.

Der Modellexport von Ultralytics bietet weiterhin keine Möglichkeit es als PyTorch Lite Modell zu exportieren; es musste hier eine kleine Anpassung im Code vorgenommen werden, damit auch dieses funktioniert. Nachdem die Auflösung des Bildes für die Inferenz in der Android Anwendung korrekt konfiguriert ist, kann das Modell für die Inferenz verwendet werden.

7.2.2 Benchmark

Benchmarks von *Dahlmann* [NN64] setzen die Erwartung, dass das YOLOv4-fastest eine Inferenzgeschwindigkeit von etwa 160 Bilder pro Sekunde mit Verwendung der GPU auf einem Google Pixel 3XL erreichen kann. Dieses Modell weist eine ähnliche Größe wie das hier vorgestellte Super Micro auf.

Es fällt auf, dass die Performanz schlechter als erwartet ausfällt: Weder das OnePlus 6 noch das Pixel 6 mit spezialisierterem Tensor Prozessor erreichen eine Echtzeitverarbeitung bei 30 Frames per Second (FPS). Vermutlich werden die GPU und NPU durch eine Inkompatibilität zwischen Hardware, Interpreter und dem Modell nicht ausgenutzt. Dieser Punkt wurde aber nicht weiter erforscht; stattdessen wird ein Benchmark mit den verfügbaren Ressourcen durchgeführt.

Mit dem Ziel einer wirklichen Echtzeitanwendung, kann nur das Super Micro Modell in 320 x 320 px Auflösung verwendet werden. Es erreicht Werte im Bereich von 20 FPS, wie in Tabelle 7.1 zu sehen ist. Eine Option zur Optimierung beim Export der Modelle kann aufgrund von Inkompatibilitäten nicht verwendet werden. Die Quantisierung in 8-Bit Integer unterliegt einem ähnlichen Problem. Die getesteten Modelle sind stattdessen als 16 Bit Floats quantisiert worden. Ein weiteres Pruning des Super Micro Modells sollte nicht durchgeführt werden, da die Parameterzahl bereits stark reduziert wurde.

Model	Resolution	#1	#2	#3	#3	#5	Avg. FPS	STDEV
Medium	640x640	0,66	0,48	0,48	0,42	0,43	0,49	±0,03
Medium	320x320	2,15	2,39	2,31	1,72	1,84	2,08	±0,29
Small	640x640	0,78	0,95	0,94	0,87	0,87	0,88	±0,04
Small	320x320	2,69	3,21	3,33	3,55	3,68	3,29	±0,20
Nano	640x640	1,86	2,06	2,15	2,12	2,15	2,07	±0,04
Nano	320x320	6,75	7,59	6,62	7,24	7,03	7,05	±0,35
Shallow	640x640	1,75	2,06	2,04	1,91	1,71	1,89	±0,14
Shallow	320x320	5,92	7,26	7,66	7,66	7,69	7,24	±0,23
Thin	640x640	3,08	3,68	3,59	3,61	3,36	3,46	±0,13
Thin	320x320	8,51	11,11	11,57	11,56	11,42	10,83	±0,32
Micro	640x640	—	—	—	—	—	—	—
Micro	320x320	13,03	15,80	15,83	16,17	14,93	15,15	±0,52
Super Micro	640x640	4,80	5,09	5,15	5,02	5,43	5,10	±0,16
Super Micro	320x320	14,19	19,92	21,58	21,57	21,90	19,83	±1,00

Tabelle 7.1: Inferenzgeschwindigkeit in FPS der jeweiligen Konfigurationen auf einem Google Pixel 6. Das Mittel wird aus 150 Messungen bestimmt.

8

KAPITEL

Schluss

8.1 Fazit	77
8.2 Ausblick	79

In dieser Arbeit wird die Problematik der Optimierung eines neuronalen Netzes erforscht. Es zeigt sich, dass auch Jahre nach der Renaissance des maschinellen Lernens der Bestand an Datensätzen noch unzureichend ist. Fortschritte in der Auswahl von Modellen und die Verwaltung des Trainings von diesen hat es viele gegeben.

8.1 Fazit

In dieser Arbeit wurde die Problematik und Problemlösung eines AI basierten Ansatzes zur Schädel-Hirn-Trauma-Detektion bearbeitet. Kern der Arbeit ist die Optimierung des neuronalen Netzes, das auf einem Smartphone ausgeführt werden soll. Mit YOLOv5 wird es ein verbreitetes Framework verwendet, das umfangreiche Bibliotheken und Services integriert, um das Training, die Analyse und die Optimierung eines Modells zu vereinfachen. Die Grundlagen und verwendeten Methoden werden vorgestellt, geplant, ausgeführt und ausgewertet.

Datensätze aus der vorherigen Arbeit sind analysiert worden, stellen aber keine hinreichende Basis für eine Optimierung dar. Weitere Datensätze, die in einer Recherche vorgestellt wurden, sind nicht für die Aufgabenstellung geeignet. Sie bilden keine hochauflösenden, generalisierbaren Darstellungen von Augen ab. Deshalb ist mit Hilfe des Unity Eyes Modells [DATA5] ein eigener Datensatz kreiert und die Pipeline zum Erstellen eben dieses erörtert worden. Da das Modell selbst keine Annotationen für die Pupillen liefert, wurde dies mit Hilfe vom Semi-Supervised Learning und mehrerer Iterationen von Ensembles gelöst.

KAPITEL 8. SCHLUSS

Mit dem neuen UnityEyes Datensatz vorliegend und der YOLO Pipeline eingerichtet, sind alle standardisierten Modelle von Ultralytics trainiert und analysiert worden. Das Größte Modell bietet im Vergleich mit dem Nano Modell nur einen geringen Mehrwert. Sie schneiden in den verschiedenen Metriken ähnlich ab, haben aber massive Unterschiede in der Geschwindigkeit. Nach einer zweistufigen Optimierung war das Nano Modell sogar in der Lage zur Performanz des größten Modells aufzuschließen. Eine weitere Versuchsreihe hat die Auswirkungen weiterer Verkleinerungen der Modellarchitektur untersucht und kam zum Schluss, dass selbst eine 15-fache Verringerung der Parameter noch gute Werte liefert.

Die Implementation der Android Anwendung ist auf Komplikationen und Inkompatibilitäten gestoßen und deshalb nicht so umfangreich ausgefallen, wie gewünscht. Da diese Implementation aber nicht Fokus der Arbeit war, wurden nur die Grundlagen für eine weitere Implementation gelegt. Eine Einordnung und Empfehlung der Modellgrößen wurde gemacht und einem Benchmark festgehalten.

Insgesamt ist in dieser Arbeit eine Zusammenfassung zur Optimierung von neuronalen Netzen entstanden. Es wird sich dabei neuer Frameworks bedient, welche umfangreiche Möglichkeiten bieten, eine Analyse der Datensätze und Modelle durchzuführen. Der UnityEyes Datensatz wird vorgestellt und eine Möglichkeit zu Erweiterung oder Veränderung mitgeliefert. Die Android Anwendung liegt als Grundausbau vor, benötigt aber noch weitere Arbeit.

8.2 Ausblick

In dieser Arbeit wurden sehr viele verschiedene Aspekte angesprochen, bearbeitet und ausgewertet. Dabei sind einige Teilespekte nicht vollständig untersucht oder abgeschlossen worden. Als größter Punkt fällt hier die Android Anwendung auf. Die Auswertung selbst findet noch nicht statt; das Modell ist aber einsatzbereit und erkennt die Bounding Boxen von Auge und Iris. Wegen verschiedener Probleme kann die Android NNAPI nicht oder nur eingeschränkt verwendet werden, welches die Auswertung auf weniger als Echtzeit einschränkt. Algorithmen zur Interpolation bei FN können bei einer weiteren Ausarbeitung für stabile Werte sorgen.

Der vorbereitete Datensatz stellt einen sehr guten Ausgangspunkt dar, ist aber nicht perfekt. Durch das Semi-Supervised Learning unter Hilfenahme des alten Pupillendatensatzes kann nicht sichergestellt werden, dass die neuen Annotationen von höchster Güte sind. Anstelle einer Auswertung der Bilder könnte eine Kooperation mit *Wood et al.* [DATA5] angestrebt werden, um das verfügbare Modell vollends auszunutzen. Dabei könnten Punkte wie eine Segmentierungskarte, Annotationen für die Pupille, Stammdaten der simulierten Patienten oder Augenkrankheiten wie Kolobome implementiert werden. So könnte eine größere Diversität und Qualität im Datensatz erreicht werden.

Eine weiter Auswertung, die angesprochen, aber nicht umgesetzt, ist die der sozialen Verträglichkeit. Im Zusammenspiel der Diversifizierung in Europa und der Welt können Auswertungen einzelner Gruppen Missstände vorbeugen. Besonders für den medizinischen Bereich kann im gleichen Schritt auch eine Auswertung über die Begründung der Ergebnisse des Modells geführt werden. Erstrebungen in diesem Bereich gibt es bereits, sind allerdings noch nicht in der breiten Masse angekommen.

Letztlich ist zu sagen, dass das Feld des maschinellen Lernens in den letzten Jahren Sprünge vollzogen hat. Modelle erreichen mit deutlich geringeren Leistungsanforderungen bessere Ergebnisse. Datensätze und Modelle, um neue zu generieren, nehmen zu und ermöglichen präzisere Modelle. Aber besonders die Frameworks zum Analysieren all dieser Neuerungen gewinnen stetig an Funktionalität, sodass mehr in weniger Zeit erreicht werden kann. Die Verwendung dieser Art der Datenauswertung wird künftig allgegenwärtig werden und das Leben der Menschen ergänzen.

A

ANHANG

Anhang

A.1	Daten zum UnityEyes Datensatz	81
A.2	Erklärung der Hyperparameter	81
A.2.1	Trainingsparameter	81
A.2.2	Eingabeparameter	82
A.3	Hyperparameteroptimierung	82
A.3.1	Zufällige Suche	82
A.3.2	Bayes'sche Suche	82

A.1 Daten zum UnityEyes Datensatz

A.2 Erklärung der Hyperparameter

A.2.1 Trainingsparameter

- Class Loss Gain: Stärke der Verlustfunktion für Klassifizierungen im Training
- Box Loss Gain: Stärke der Verlustfunktion für Bounding Boxen im Training
- IOU Threshold: Der IoU Schwellwert, der während des Trainings verwendet wird. Filtert im Non-Maximum suppression Schritt Boxen unter diesem Wert heraus.
- Learning Rate 0: Initiale Lernrate des gewählten Optimierers. Für SGD wird normal 0.01, für Adam 0.001 verwendet.
- Learning Rate F: Die initiale OneCycle Lernrate des Optimierers; sie beeinflusst die Lernrate über die Zeit.
- Momentum: Das Momentum vom SGD und Beta1 vom Adam Optimizer.
- Weight Decay: Gewichtsverlust des Optimierers über Zeit.

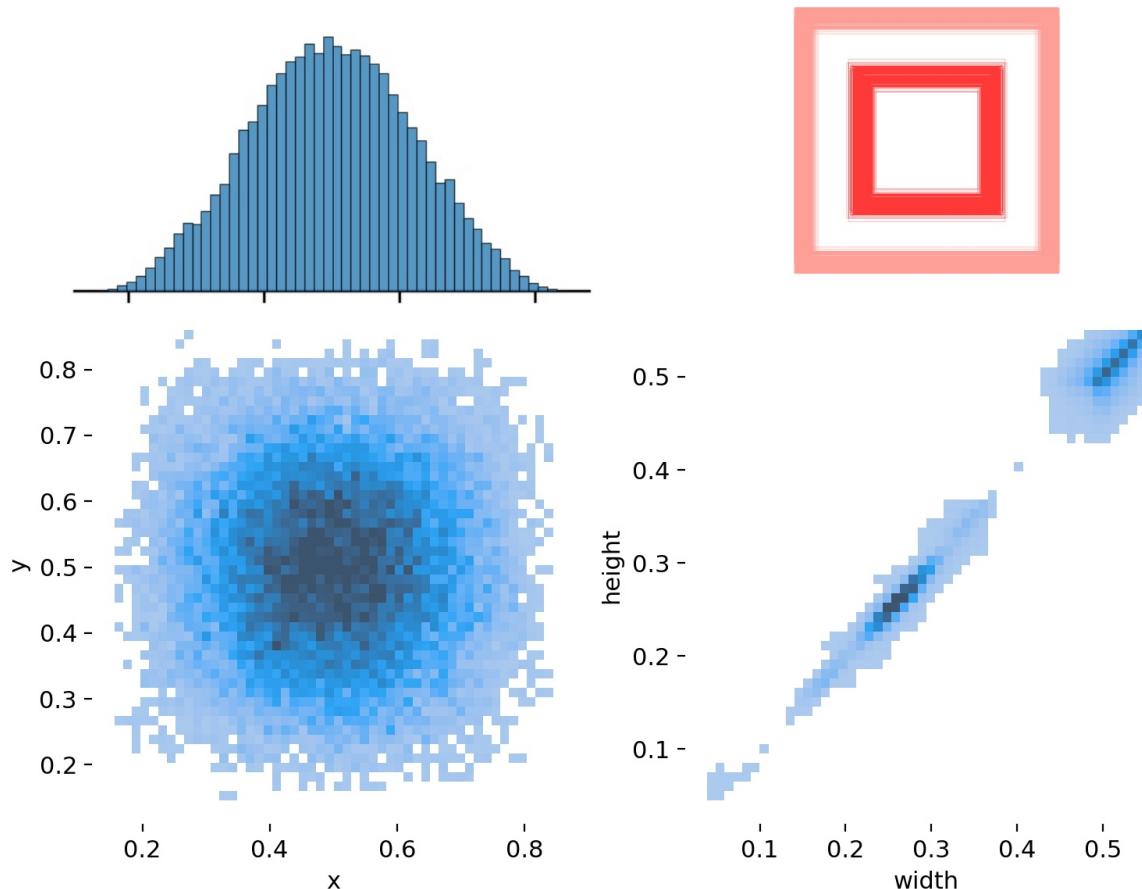


Abbildung A.1: Die Verteilung der Label und der Bounding Boxen im finalen Datensatz.

A.2.2 Eingabeparameter

- Batch Size: Anzahl der Bilder, die während eines Lernschritts verarbeitet werden.
- Scale: Skalierungsfaktor der Eingabebilder. Symmetrisch um Null.
- Translate: Verschiebung des Bildes auf der Leinwand in Prozent der Bildgröße.
- FlipLR: Wahrscheinlichkeit, mit der Bilder Horizontal gespiegelt werden.
- Color Hue: Faktor zur Veränderung des HSV-Hues.
- Color Value: Faktor zur Veränderung des HSV-Values.
- Color Saturation: Faktor zur Veränderung der HSV-Saturation.

A.3 Hyperparameteroptimierung

A.3.1 Zufällige Suche

A.3.2 Bayes'sche Suche

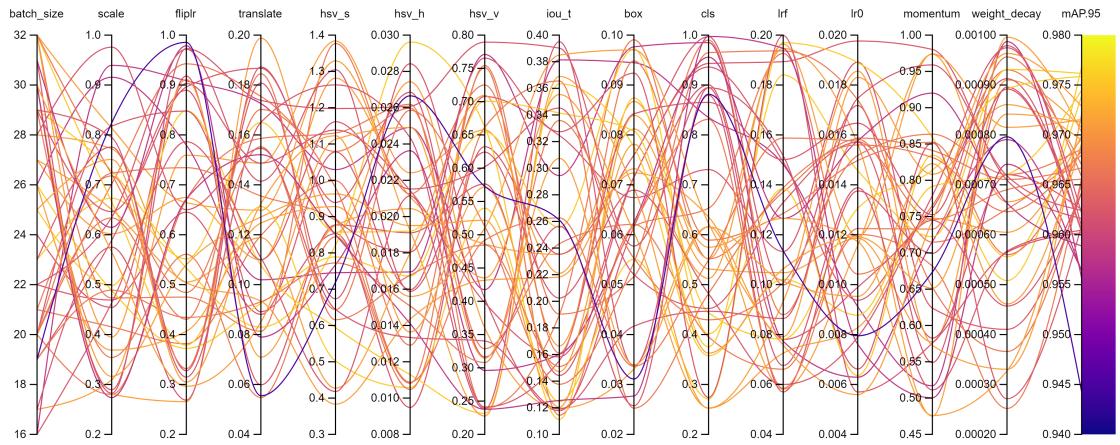


Abbildung A.2: Parallel Coordinate Plot der Zufälligen Suche. Verwendet die Maxima und Minima des jeweiligen Parameters als oberes und unteres Limit.

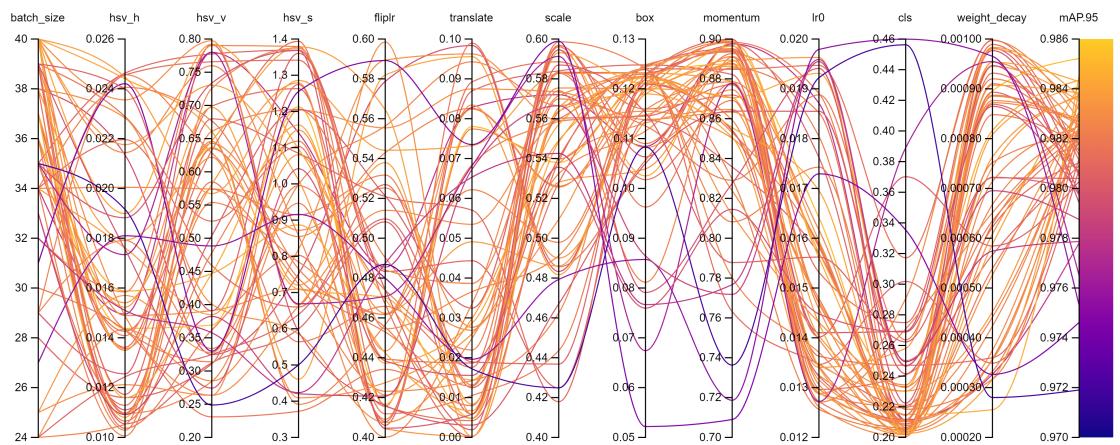


Abbildung A.3: Parallel Coordinate Plot der Bayes'schen Suche. Verwendet die Maxima und Minima des jeweiligen Parameters als oberes und unteres Limit.

Literaturverzeichnis

- [1] H. Altenburg et al. "Zur Problematik von Anisokorie und Operationszeitpunkt beim schwer Schädelhirnverletzten". In: 1982.
- [2] Peter Sollich and Anders Krogh. "Learning with ensembles: How overfitting can be useful". In: *NIPS*. 1995.
- [3] Thomas Ertl. "Computer graphics—principles and practice". In: 1997.
- [4] Tin Kam Ho. "The Random Subspace Method for Constructing Decision Forests". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 20 (1998), pp. 832–844.
- [5] J. Piek. "Schädel-Hirn-Trauma". In: *Notfall & Rettungsmedizin* 5 (2002), pp. 309–318.
- [6] E. Rickels. "Das Schädel-Hirn-Trauma: Epidemiologie, Therapie und Prognose". In: *Intensivmedizin Und Notfallmedizin* 40 (2003), pp. 658–671.
- [7] Jakob Matschke, Klaus Püschel, and Markus Glatzel. "Schädel-Hirn-Trauma und Sport". In: *Rechtsmedizin* 21 (2011), pp. 191–196.
- [8] Lorenz Wanke-Jellinek, Martijn van Griensven, and P. Biberthaler. "Diagnostische Biomarker des Schädel-Hirn-Traumas". In: *Der Unfallchirurg* 117 (2013), pp. 693–698.
- [9] J. Weissman and M. Zinner. "Comparative effectiveness research on robotic surgery." In: *JAMA* 309 7 (2013), pp. 721–2.
- [10] Walber. *Precision and recall*. https://en.wikipedia.org/wiki/Precision_and_recall#/media/File\protect\leavevmode@ifvmode\kern+.2222em\relaxPrecisionrecall.svg. [Online; accessed 9.10.2021]. 2014.
- [11] Frederic Lardinois. *Google makes Kotlin a first-class language for writing Android apps*. <https://techcrunch.com/2017/05/17/google-makes-kotlin-a-first-class-language-for-writing-android-apps/>. [Online; accessed 21.11.2021]. 2017.
- [12] Jason D. Wright. "Robotic-Assisted Surgery: Balancing Evidence and Implementation." In: *JAMA* 318 16 (2017), pp. 1545–1547.

- [13] IEEE. "IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84. DOI: 10.1109/IEEESTD.2019.8766229.
- [14] Jeremy A. Irvin et al. "CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison". In: *AAAI*. 2019.
- [15] Frederic Lardinois. *Kotlin is now Google's preferred language for Android app development*. <https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/>. [Online; accessed 21.11.2021]. 2019.
- [16] amusi. *The project should not be named yolov5!* <https://github.com/ultralytics/yolov5/issues/2>. [Online; accessed 15.11.2021]. 2020.
- [17] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from [wandb.com](https://www.wandb.com/). 2020. URL: <https://www.wandb.com/>.
- [18] Joseph Paul Cohen, Paul Morrison, and Lan Dao. "COVID-19 Image Data Collection". In: *ArXiv* abs/2003.11597 (2020).
- [19] eeric. *where is yolov5 paper?* <https://github.com/ultralytics/yolov5/issues/1333>. [Online; accessed 15.11.2021]. 2020.
- [20] Tianyi Gu. *43% of Active Smartphones Will Be 5G-Ready by 2023: The Global Mobile Market Is on Track for Substantial Growth and Game-Related Engagement*. <https://newzoo.com/insights/articles/mobile-game-market-2020-smartphone-users-game-revenues-5g-ready-engagement/>. [Online; accessed 21.08.2021]. 2020.
- [21] K. Sheetz, J. Claflin, and J. Dimick. "Trends in the Adoption of Robotic Surgery for Common Surgical Procedures". In: *JAMA Network Open* 3 (2020).
- [22] ARM. *Processing Architecture for Power Efficiency and Performance*. <https://www.arm.com/why-arm/technologies/big-little/>. [Online; accessed 10.10.2021]. 2021.
- [23] Bundesministerium für Bildung und Forschung. *Künstliche Intelligenz*. https://www.bmbf.de/bmbf/de/forschung/digitale-wirtschaft-und-gesellschaft/kuenstliche-intelligenz/kuenstliche-intelligenz_node.html. [Online; accessed 21.08.2021]. 2021.
- [24] Deutsche Forschungsgemeinschaft. *Förderinitiative „Künstliche Intelligenz“*. <https://www.dfg.de/foerderung/ai-initiative/index.html>. [Online; accessed 21.08.2021]. 2021.
- [25] Google. *Neural Networks API*. <https://developer.android.com/ndk/guides/neuralnetworks/>. [Online; accessed 10.10.2021]. 2021.
- [26] Alexander Moryäner. *Entwicklung einer App zur Diagnoseunterstützung durch Analyse der Pupillengröße per Smartphonekamera*. Jan. 2021.

- [27] StatCounter. *Mobile Operating System Market Share Worldwide*. <https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200909-202101/>. [Online; accessed 23.08.2021]. 2021.

Neural Networks

- [NN1] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: <https://doi.org/10.1007/BF02478259>.
- [NN2] F. Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65 6 (1958), pp. 386–408.
- [NN3] David H. Hubel and Torsten N. Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of Physiology* 160 (1962).
- [NN4] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition”. In: *Competition and Cooperation in Neural Nets*. Ed. by Shun-ichi Amari and Michael A. Arbib. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 267–285. ISBN: 978-3-642-46466-9.
- [NN5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.
- [NN6] George V. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314.
- [NN7] Ken-ichi Funahashi. “On the approximate realization of continuous mappings by neural networks”. In: *Neural Networks* 2 (1989), pp. 183–192.
- [NN8] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. “Multilayer feed-forward networks are universal approximators”. In: *Neural Networks* 2 (1989), pp. 359–366.
- [NN9] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4 (1991), pp. 251–257.
- [NN10] Steven J. Nowlan and Geoffrey E. Hinton. “Simplifying Neural Networks by Soft Weight-Sharing”. In: *Neural Computation* 4 (1992), pp. 473–493.
- [NN11] Moshe Leshno et al. “Multilayer Feedforward Networks with a Non-Polynomial Activation Function Can Approximate Any Function”. In: *New York University Stern School of Business Research Paper Series* (1993).

- [NN12] Yann André LeCun et al. “Gradient-based learning applied to document recognition”. In: 1998.
- [NN13] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. “Learning to Forget: Continual Prediction with LSTM”. In: *Neural Computation* 12 (2000), pp. 2451–2471.
- [NN14] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22 (2010), pp. 1345–1359.
- [NN15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60 (2012), pp. 84–90.
- [NN16] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”. In: *EMNLP*. 2014.
- [NN17] Ross B. Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 580–587.
- [NN18] Pierre Sermanet et al. “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”. In: *CoRR* abs/1312.6229 (2014).
- [NN19] Ross B. Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1440–1448.
- [NN20] Kaiming He et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (2015), pp. 1904–1916.
- [NN21] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).
- [NN22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: <https://doi.org/10.1038/nature14539>.
- [NN23] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2015), pp. 1137–1149.
- [NN24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *MICCAI*. 2015.
- [NN25] J. Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks : the official journal of the International Neural Network Society* 61 (2015), pp. 85–117.
- [NN26] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2015).

- [NN27] Christian Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 1–9.
- [NN28] Song Han, Huizi Mao, and William J. Dally. “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”. In: *arXiv: Computer Vision and Pattern Recognition* (2016).
- [NN29] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.
- [NN30] W. Liu et al. “SSD: Single Shot MultiBox Detector”. In: *ECCV*. 2016.
- [NN31] MartínAbadi et al. “TensorFlow: A system for large-scale machine learning”. In: *OSDI*. 2016.
- [NN32] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 779–788.
- [NN33] Hoo-Chang Shin et al. “Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning”. In: *IEEE Transactions on Medical Imaging* 35 (2016), pp. 1285–1298.
- [NN34] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2818–2826.
- [NN35] Nima Tajbakhsh et al. “Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?” In: *IEEE Transactions on Medical Imaging* 35 (2016), pp. 1299–1312.
- [NN36] Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *ArXiv* abs/1704.04861 (2017).
- [NN37] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: 2017.
- [NN38] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 6517–6525.
- [NN39] Christian Szegedy et al. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: *AAAI*. 2017.
- [NN40] Antti Tarvainen and Harri Valpola. “Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results”. In: *NIPS*. 2017.
- [NN41] Sanjana Babu. *Anchor Boxes for Object Detection*. <https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html/>. [Online; accessed 16.11.2021]. 2018.

- [NN42] Dongdong Chen et al. “Tri-net for Semi-Supervised Deep Learning”. In: *IJCAI*. 2018.
- [NN43] Shu Liu et al. “Path Aggregation Network for Instance Segmentation”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 8759–8768.
- [NN44] Dominic Masters and Carlo Luschi. “Revisiting Small Batch Training for Deep Neural Networks”. In: *ArXiv* abs/1804.07612 (2018).
- [NN45] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *ArXiv* abs/1804.02767 (2018).
- [NN46] Sik-Ho Tsang. *Review: YOLOv2 & YOLO9000 — You Only Look Once (Object Detection)*. <https://towardsdatascience.com/review-yolov2-yolo9000-you-only-look-once-object-detection-7883d2b02a65/>. [Online; accessed 17.11.2021]. 2018.
- [NN47] Ben Athiwaratkun et al. “There Are Many Consistent Explanations of Unlabeled Data: Why You Should Average”. In: *ICLR*. 2019.
- [NN48] Dami Choi et al. “On Empirical Comparisons of Optimizers for Deep Learning”. In: *ArXiv* abs/1910.05446 (2019).
- [NN49] DATAmadness. *TensorFlow 2 - CPU vs GPU Performance Comparison*. <https://datamadness.github.io/TensorFlow2-CPU-vs-GPU/>. [Online; accessed 9.11.2021]. 2019.
- [NN50] Andrew G. Howard et al. “Searching for MobileNetV3”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 1314–1324.
- [NN51] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. In: *Journal of Big Data* 6 (2019), pp. 1–48.
- [NN52] Sanjana Babu. *SSD MobileNetV1 architecture*. <https://iq.opengenus.org/ssd-mobilenet-v1-architecture/>. [Online; accessed 16.11.2021]. 2020.
- [NN53] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *ArXiv* abs/2004.10934 (2020).
- [NN54] Kaiming He et al. “Mask R-CNN”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42 (2020), pp. 386–397.
- [NN55] Glenn Jocher et al. *ultralytics/yolov5: Initial Release*. <https://zenodo.org/record/3908560>. [Online; accessed 15.11.2021]. 2020. DOI: <https://doi.org/10.5281/zenodo.3908560>.
- [NN56] Glenn Jocher et al. *ultralytics/yolov5: v2.0*. <https://zenodo.org/record/3958273>. [Online; accessed 15.11.2021]. 2020. DOI: <https://doi.org/10.5281/zenodo.3958273>.

- [NN57] Glenn Jocher et al. *ultralytics/yolov5: v3.0*. <https://zenodo.org/record/3983579>. [Online; accessed 15.11.2021]. 2020. DOI: <https://doi.org/10.5281/zenodo.3983579>.
- [NN58] Glenn Jocher et al. *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements*. <https://zenodo.org/record/4154370>. [Online and accessed 15.11.2021]. 2020. DOI: <https://doi.org/10.5281/zenodo.4154370>.
- [NN59] Glenn Jocher et al. *yolov5*. <https://github.com/ultralytics/yolov5>. commit cd35a009ba964331abcccd30f6fa0614224105d39. 2020.
- [NN60] WENCHI MA et al. "Why Layer-Wise Learning is Hard to Scale-up and a Possible Solution via Accelerated Downsampling". In: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)* (2020), pp. 238–243.
- [NN61] Chien-Yao Wang et al. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2020), pp. 1571–1580.
- [NN62] Qizhe Xie et al. "Unsupervised Data Augmentation for Consistency Training". In: *arXiv: Learning* (2020).
- [NN63] J. Zhang et al. "Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity". In: *arXiv: Optimization and Control* (2020).
- [NN64] FLORIAN DAHLMANN. *Pedestrian Light Detection for Visually Impaired People Using Machine Learning on Mobile Devices*. Aug. 2021.
- [NN65] Glenn Jocher et al. *ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration*. <https://zenodo.org/record/4418161>. [Online; accessed 15.11.2021]. 2021. DOI: <https://doi.org/10.5281/zenodo.4418161>.
- [NN66] Glenn Jocher et al. *ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations*. <https://zenodo.org/record/4679653>. [Online; accessed 15.11.2021]. 2021. DOI: <https://doi.org/10.5281/zenodo.4679653>.
- [NN67] Glenn Jocher et al. *ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support*. <https://zenodo.org/record/5563715>. [Online; accessed 15.11.2021]. 2021. DOI: <https://doi.org/10.5281/zenodo.5563715>.
- [NN68] David Picard. "Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision". In: *ArXiv abs/2109.08203* (2021).
- [NN69] *mAP (mean Average Precision) might confuse you!* <https://xailient.com/map-mean-average-precision-might-confuse-you/>. [Online; accessed 7.11.2021].

Hyperparameteroptimierung

- [HPO1] M. Birattari et al. "A Racing Algorithm for Configuring Metaheuristics". In: *GECCO*. 2002.
- [HPO2] Frank Hutter, Holger H. Hoos, and Thomas Stützle. "Automatic Algorithm Configuration Based on Local Search". In: *AAAI*. 2007.
- [HPO3] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. "A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms". In: *CP*. 2009.
- [HPO4] Frank Hutter et al. "ParamILS: An Automatic Algorithm Configuration Framework". In: *ArXiv* abs/1401.3492 (2009).
- [HPO5] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *LION*. 2011.
- [HPO6] Lin Xu et al. "Hydra-MIP : Automated Algorithm Configuration and Selection for Mixed Integer Programming". In: 2011.
- [HPO7] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *J. Mach. Learn. Res.* 13 (2012), pp. 281–305.
- [HPO8] Bobak Shahriari et al. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104 (2016), pp. 148–175.
- [HPO9] Jiaxiang Wu et al. "Quantized Convolutional Neural Networks for Mobile Devices". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 4820–4828.
- [HPO10] Lisha Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *J. Mach. Learn. Res.* 18 (2017), 185:1–185:52.
- [HPO11] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 2704–2713.
- [HPO12] Paulius Micikevicius et al. "Mixed Precision Training". In: *ArXiv* abs/1710.03740 (2018).
- [HPO13] SAYAK PAUL. *Bayesian Hyperparameter Optimization - A Primer*. [Online; accessed 17.11.2021]. 2020. URL: <https://wandb.ai/site/articles/bayesian-hyperparameter-optimization-a-primer/>.
- [HPO14] Marius Lindauer et al. *SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization*. 2021. arXiv: 2109.09831 [cs.LG].

Datensätze

- [DATA1] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
- [DATA2] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88 (2009), pp. 303–338.
- [DATA3] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *ECCV*. 2014.
- [DATA4] Erroll Wood et al. "Rendering of Eyes for Eye-Shape Registration and Gaze Estimation". In: *Proc. of the IEEE International Conference on Computer Vision (ICCV 2015)*. Dec. 12, 2015.
- [DATA5] E. Wood et al. "Learning an appearance-based gaze estimator from one million synthesised images". In: *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications* (2016).
- [DATA6] R. Fusek. "Pupil localization using geodesic distance". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11241 LNCS (2018), pp. 433–444. DOI: 10.1007/978-3-030-03801-4_38.
- [DATA7] Wolfgang Fuhl, W. Rosenstiel, and Enkelejda Kasneci. "500, 000 Images Closer to Eyelid and Pupil Segmentation". In: *CAIP*. 2019.
- [DATA8] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 4396–4405.
- [DATA9] Bingnan Luo et al. "Shape Constrained Network for Eye Segmentation in the Wild". In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2020), pp. 1941–1949.
- [DATA10] Natalia Sokolova et al. "Pixel-Based Iris and Pupil Segmentation in Cataract Surgery Videos Using Mask R-CNN". In: *2020 IEEE 17th International Symposium on Biomedical Imaging Workshops (ISBI Workshops)* (2020), pp. 1–4.
- [DATA11] *Flickr Eye*, licensed under <https://creativecommons.org/licenses/by-nc-sa/4.0/>. <https://www.flickr.com/>. [Online; accessed 9.09.2021].
- [DATA12] Nvidia. *What Is Synthetic Data?* <https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/>. [Online; accessed 8.11.2021].

Computer Vision

- [CV1] R. Duda and P. Hart. "Use of the Hough transformation to detect lines and curves in pictures". In: *Commun. ACM* 15 (1972), pp. 11–15.

- [CV2] J. Daugman. "High Confidence Visual Recognition of Persons by a Test of Statistical Independence". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 15 (1993), pp. 1148–1161.
- [CV3] A. Likas, N. Vlassis, and J. Verbeek. "The global k-means clustering algorithm". In: *Pattern Recognit.* 36 (2003), pp. 451–461.
- [CV4] David Arthur and Sergei Vassilvitskii. "k-means++: the advantages of careful seeding". In: *SODA '07*. 2007.
- [CV5] Shi Na, Liu Xumin, and Guan Yong. "Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm". In: *2010 Third International Symposium on Intelligent Information Technology and Security Informatics* (2010), pp. 63–67.
- [CV6] Nourredine Cherabit, F. Chelali, and A. Djeradi. "A robust iris localization method of facial faces". In: *2011 International Conference on Multimedia Computing and Systems* (2011), pp. 1–5.
- [CV7] Fabian Timm and E. Barth. "Accurate Eye Centre Localisation by Means of Gradients". In: *VISAPP*. 2011.
- [CV8] Zu-wei Zhou et al. "A robust algorithm for iris localization based on radial symmetry and circular integro differential operator". In: *2011 6th IEEE Conference on Industrial Electronics and Applications* (2011), pp. 1742–1745.
- [CV9] N. Cherabit, F. Chelali, and A. Djeradi. "Circular Hough Transform for Iris localization". In: 2012.
- [CV10] Roberto Valenti and T. Gevers. "Accurate Eye Center Location through Invariant Isocentric Patterns". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (2012), pp. 1785–1798.
- [CV11] A. Villanueva et al. "Hybrid method based on topography for robust detection of iris center and eye corners". In: *ACM Trans. Multim. Comput. Commun. Appl.* 9 (2013), 25:1–25:20.
- [CV12] R. Bozomitu et al. "Pupil centre coordinates detection using the circular Hough transform technique". In: *2015 38th International Spring Seminar on Electronics Technology (ISSE)* (2015), pp. 462–465.
- [CV13] A. Memis, S. Varli, and F. Bilgili. "Computerized 2D detection of the multiform femoral heads in magnetic resonance imaging (MRI) sections with the integro-differential operator". In: *Biomed. Signal Process. Control.* 54 (2019).

Augen, Pupille und Iris

- [EYE1] E. Alexandridis. "Lichtsinn und Pupillenreaktion". In: 1973.
- [EYE2] Hurst JW Walker HK Hall WD. "Clinical Methods: The History, Physical, and Laboratory Examinations. 3rd edition". In: *Butterworth* 264 (1990), pp. 2808–2809.

- [EYE3] Nathan J. Emery. "The eyes have it: the neuroethology, function and evolution of social gaze". In: *Neuroscience & Biobehavioral Reviews* 24 (2000), pp. 581–604.
- [EYE4] H. Kobayashi and Shiro Kohshima. "Unique morphology of the human eye and its adaptive meaning: comparative studies on external morphology of the primate eye." In: *Journal of human evolution* 40 5 (2001), pp. 419–35.
- [EYE5] Yabo Yang, Keith Thompson, and Stephen A. Burns. "Pupil location under mesopic, photopic, and pharmacologically dilated conditions". eng. In: *Investigative ophthalmology & visual science* 43.7 (July 2002). 12091457[pmid], pp. 2508–2512. ISSN: 0146-0404. URL: <https://pubmed.ncbi.nlm.nih.gov/12091457/>.
- [EYE6] E. Alexandridis and E. R. Koeppe. "Die spektrale Empfindlichkeit der für den Pupillenlichtreflex verantwortlichen Photorezeptoren beim Menschen". In: *Albrecht von Graefes Archiv für klinische und experimentelle Ophthalmologie* 177 (2004), pp. 136–151.
- [EYE7] A. Müller-Jensen and R. Hagenah. "Untersuchungen zur Variabilität des phasischen Pupillenlichtreflexes". In: *Journal of Neurology* 212 (2004), pp. 123–132.
- [EYE8] W. Brunn et al. "Untersuchungen über die Pupillenreflexe beim Menschen". In: *Pflüger's Archiv für die gesamte Physiologie des Menschen und der Tiere* 244 (2005), pp. 644–658.
- [EYE9] Greyson Orlando. *Eye dilate*. https://commons.wikimedia.org/wiki/File:Eye_dilate-thumb_300px.gif. [Online; accessed 9.09.2021]. 2006.
- [EYE10] Michael Tomasello et al. "Reliance on head versus eyes in the gaze following of great apes and human infants: the cooperative eye hypothesis." In: *Journal of human evolution* 52 3 (2007), pp. 314–20.
- [EYE11] J. Bradley et al. "The effect of gender and iris color on the dark-adapted pupil diameter." In: *Journal of ocular pharmacology and therapeutics : the official journal of the Association for Ocular Pharmacology and Therapeutics* 26 4 (2010), pp. 335–40.
- [EYE12] Juan Alberto Sanchis-Gimeno, Daniel Sánchez-Zuriaga, and Francisco Martínez-Soriano. "White-to-white corneal diameter, pupil diameter, central corneal thickness and thinnest corneal thickness values of emmetropic subjects". In: *Surgical and Radiologic Anatomy* 34 (2011), pp. 167–170.
- [EYE13] Li Chen and D. Chernyak. "Pupil Changes under Scotopic and Photopic illumination". In: 2013.
- [EYE14] Sung Hee Kim. "Approach to pupillary abnormalities via anatomical pathways". In: *Yeungnam University Journal of Medicine* 34 (2017), pp. 11–18.

- [EYE15] Yong Wang. "Comparison of Different Methods for Photopic and Scotopic Pupil Diameter Measurement and Clinical Analysis on the Pupil diameter changes of cataract patients". In: *Investigative Ophthalmology & Visual Science* 60.9 (2019), pp. 513–513.
- [EYE16] Astrid Högemann. *Pupillenreflex*. <https://flexikon.doccheck.com/de/Pupillenreflex>. [Online; accessed 2.11.2021].

Gaze Estimation

- [G1] Q. Ji and Xiaojie Yang. "Real-Time Eye, Gaze, and Face Pose Tracking for Monitoring Driver Vigilance". In: *Real Time Imaging* 8 (2002), pp. 357–377.
- [G2] Q. Ji and Zhiwei Zhu. "Non-intrusive Eye and Gaze Tracking for Natural Human Computer Interaction". In: *MMI Interakt.* 6 (2003).
- [G3] Q. Ji, Zhiwei Zhu, and Peilin Lan. "Real-time nonintrusive monitoring and prediction of driver fatigue". In: *IEEE Transactions on Vehicular Technology* 53 (2004), pp. 1052–1068.
- [G4] Zhiwei Zhu and Q. Ji. "Eye and gaze tracking for interactive graphic display". In: *Machine Vision and Applications* 15 (2004), pp. 139–148.
- [G5] C. Morimoto and Marcio R. M. Mimica. "Eye gaze tracking techniques for interactive applications". In: *Comput. Vis. Image Underst.* 98 (2005), pp. 4–24.
- [G6] K. Kafka et al. "Eye Tracking for Everyone". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2176–2184.
- [G7] Zhengyang Wu et al. "MagicEyes: A Large Scale Eye Gaze Estimation Dataset for Mixed Reality". In: *ArXiv* abs/2003.08806 (2020).

Social Responsibility

- [SR1] N. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *J. Artif. Intell. Res.* 16 (2002), pp. 321–357.
- [SR2] Mikel Galar et al. "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42 (2012), pp. 463–484.
- [SR3] Paula Branco, Luís Torgo, and Rita P. Ribeiro. "A Survey of Predictive Modeling on Imbalanced Domains". In: *ACM Computing Surveys (CSUR)* 49 (2016), pp. 1–50.
- [SR4] Eugene Bagdasaryan and Vitaly Shmatikov. "Differential Privacy Has Disparate Impact on Model Accuracy". In: *NeurIPS*. 2019.

- [SR5] David Leslie. "Understanding Artificial Intelligence Ethics and Safety: A Guide for the Responsible Design and Implementation of AI Systems in the Public Sector". In: *SSRN Electronic Journal* (2019).
- [SR6] "4. ‘What Gets Counted Counts’". In: *Data Feminism*. MIT press, Mar. 16, 2020. URL: <https://data-feminism.mitpress.mit.edu/pub/h1w0nbqp>.
- [SR7] Emily L. Denton et al. "Bringing the People Back In: Contesting Benchmark Machine Learning Datasets". In: *ArXiv* abs/2007.07399 (2020).
- [SR8] Sara Hooker et al. "What Do Compressed Deep Neural Networks Forget". In: *arXiv: Learning* (2020).